Software-Defined Networking (SDN)

What is software-defined networking?

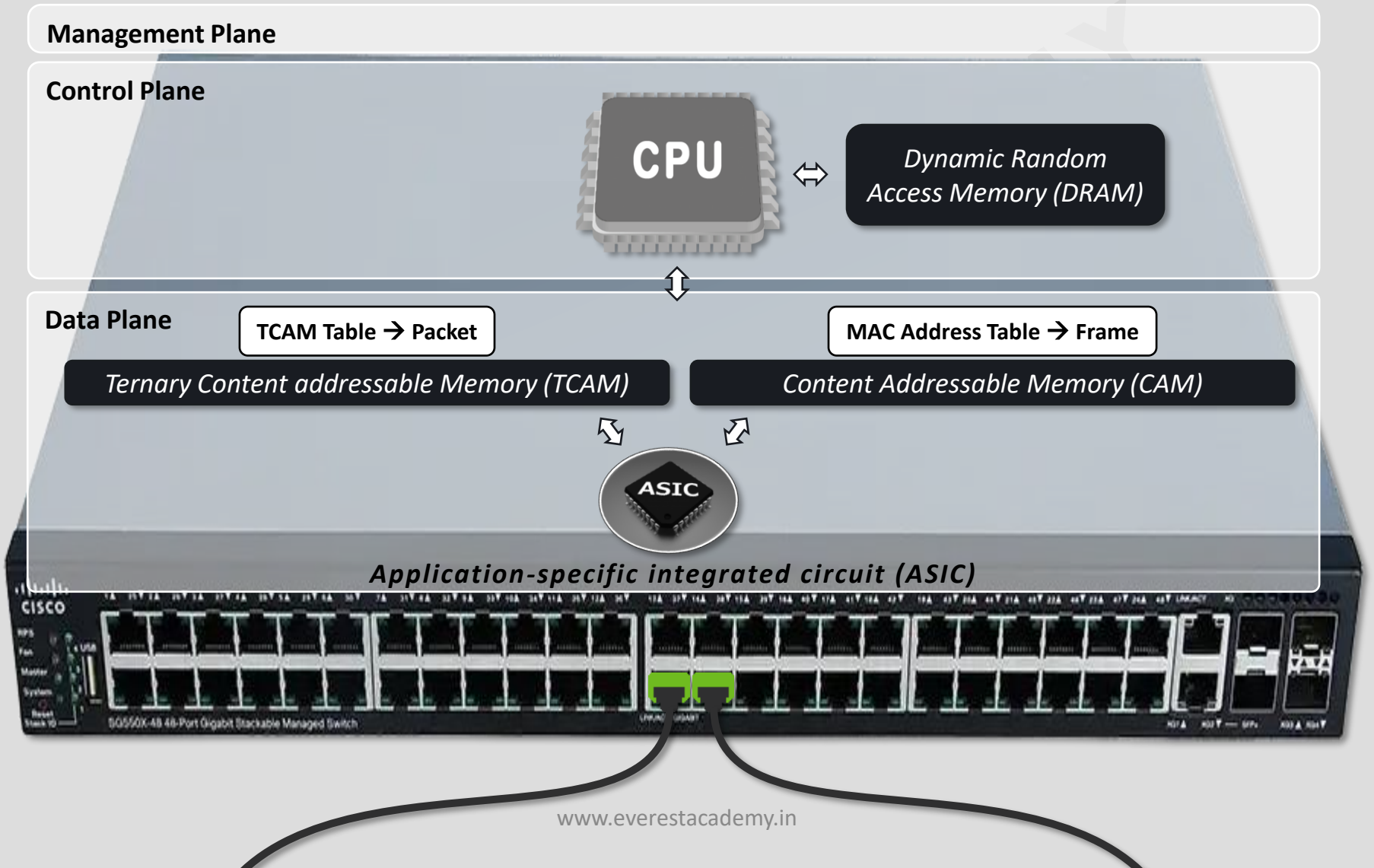
Software-defined networking (SDN) is an architecture designed to make a network more flexible and easier to manage. SDN centralizes management by abstracting the control plane from the data forwarding function in the discrete networking devices.

Controller-Based Networking

Data, Control, and Management Planes

- Management Plane:
  - SSH
  - SNMP
  - Telnet
  - TFTP
  - HTTP
  - HTTPS
- Control Plane:
  - RIP
  - OSPF
  - ARP
  - STP
  - VTP
- Data Plane (Forwarding Plane):
  - De-encapsulation and re-encapsulation.
  - Adding or removing an 802.1Q trunking header.
  - Matching the destination MAC address.
  - Matching the destination IP address.
  - Encrypting the data (VPN).
  - Discarding a message due to a filter (ACLs, port security).

Traditional Multilayer Switches

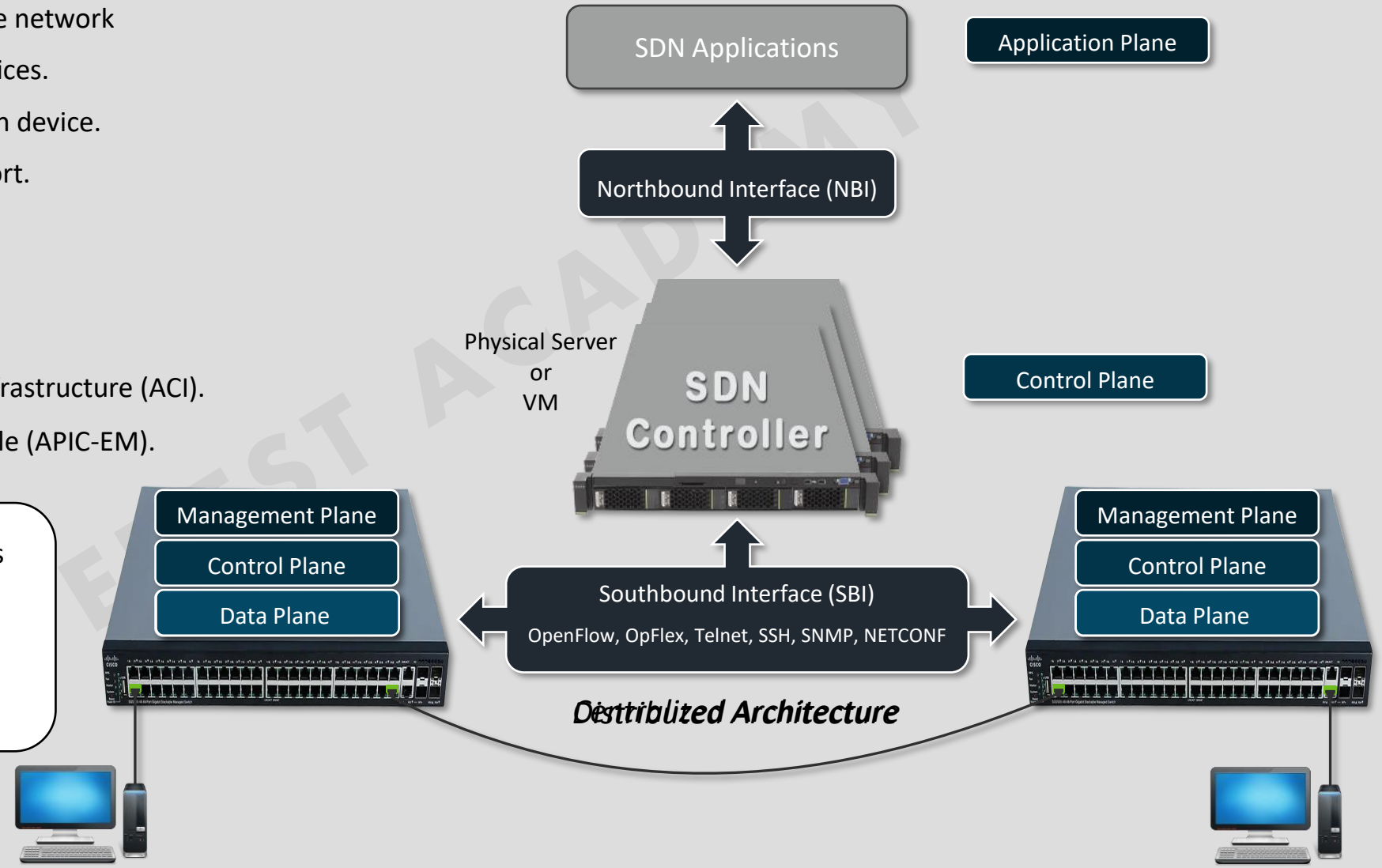


Controllers and Centralized Control

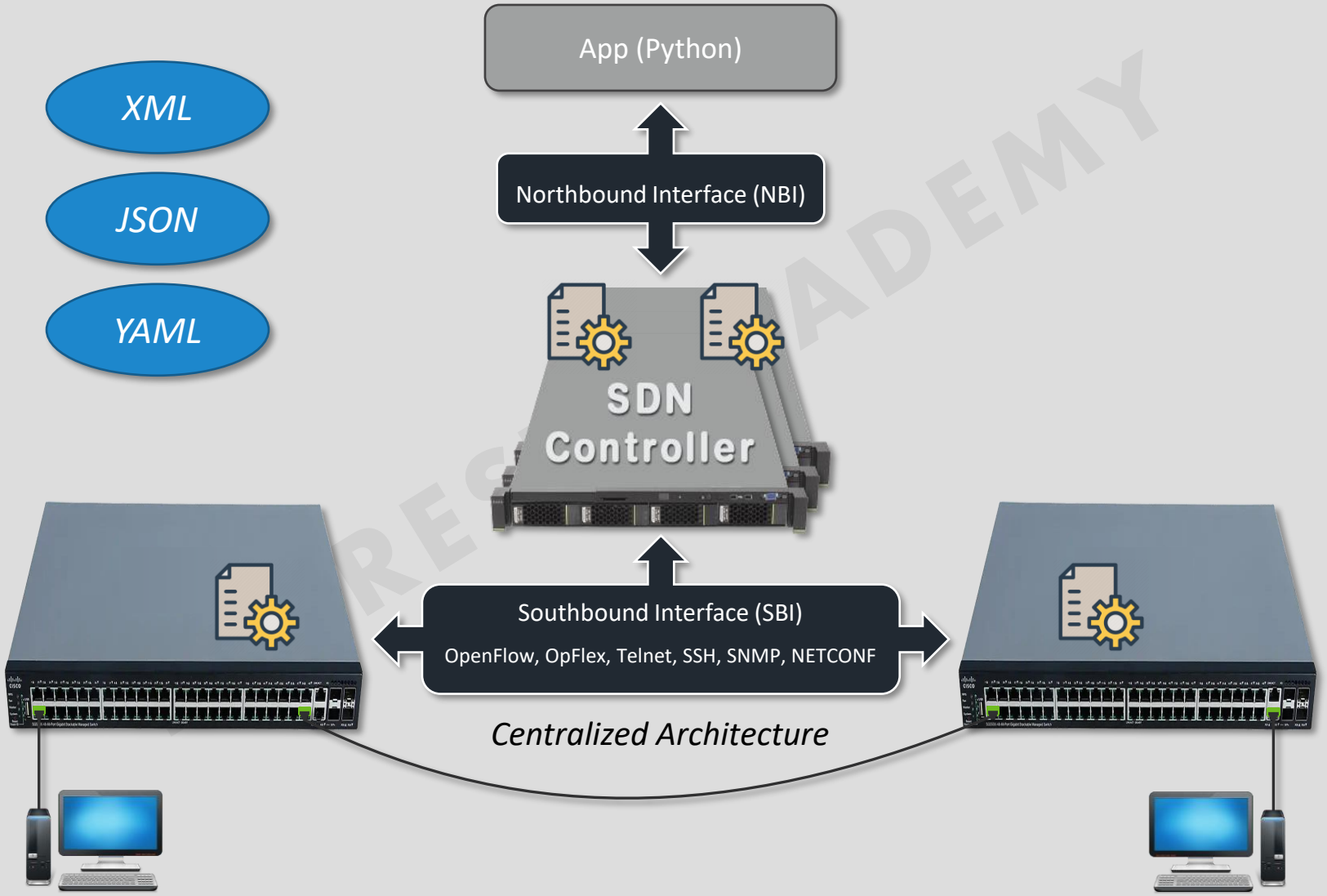
- A list of all the devices in the network
- The capabilities of each devices.
- The interfaces/ports on each device.
- The current state of each port.
- Network topology.
- Device configuration.

- OpenDaylight Controller.
- Cisco Application Centric Infrastructure (ACI).
- Cisco APIC Enterprise Module (APIC-EM).

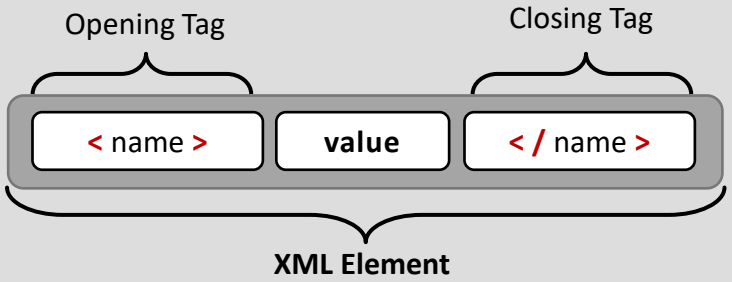
- Application Programming Interfaces (APIs).
- REST API.
- JavaScript Object Notation (JSON).



Data Serialization Languages



Extensible Markup Language (XML)



R1# show ip interface brief | **format**

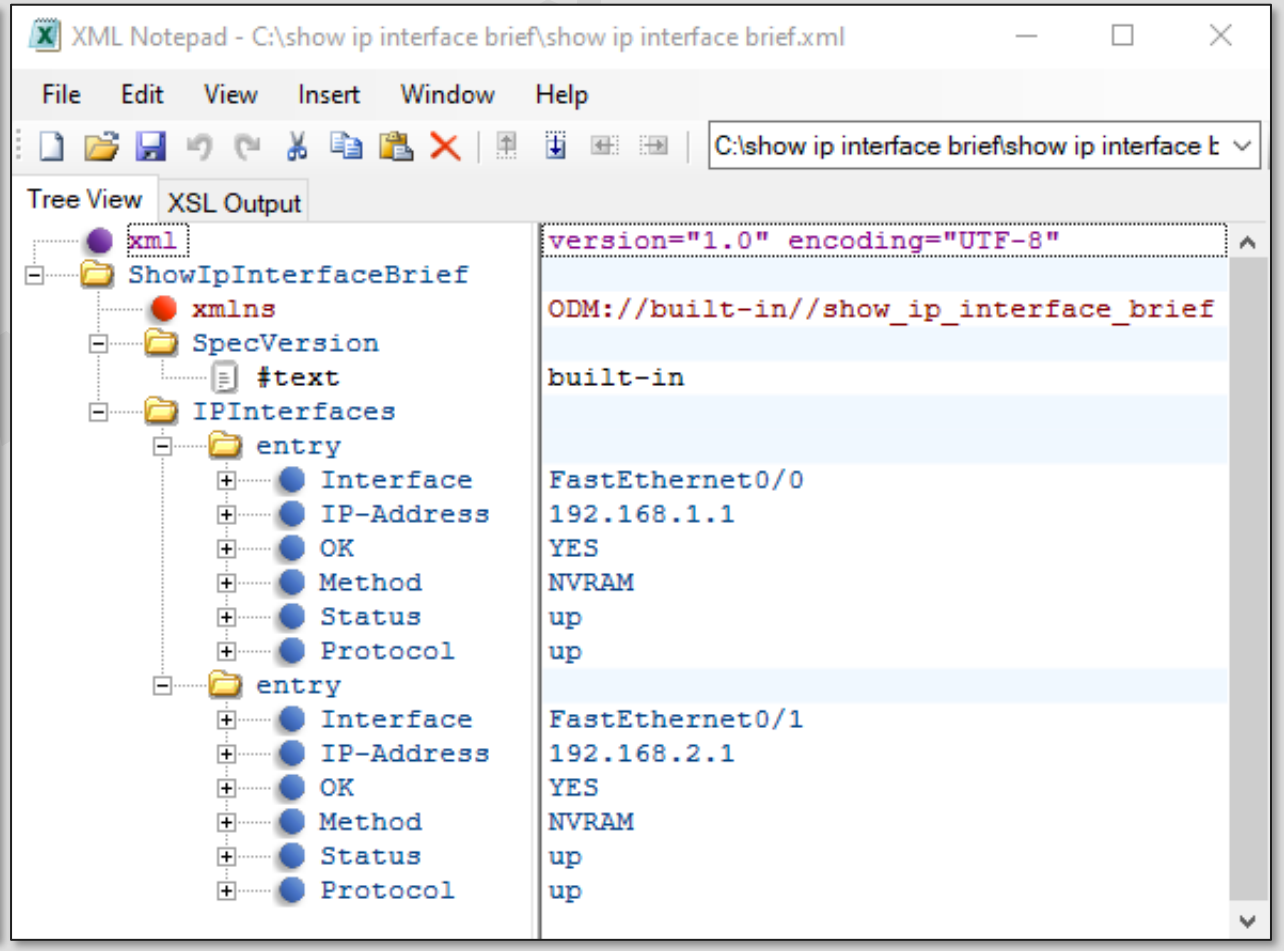
Interface	IP-Address	OK?	Method	Status	Protocol
FastEthernet0/0	192.168.1.1	YES	NVRAM	up	up
FastEthernet0/1	192.168.2.1	YES	NVRAM	up	up

```
<?xml version="1.0" encoding="UTF-8"?>
  <ShowIpInterfaceBrief xmlns="ODM://built-in//show_ip_interface_brief">
    <SpecVersion>built-in</SpecVersion>
    <IPInterfaces>
      <entry>
        <Interface>FastEthernet0/0</Interface>
        <IP-Address>192.168.1.1</IP-Address>
        <OK>YES</OK>
        <Method>NVRAM</Method>
        <Status>up</Status>
        <Protocol>up</Protocol>
      </entry>
      <entry>
        <Interface>FastEthernet0/1</Interface>
        <IP-Address>192.168.2.1</IP-Address>
        <OK>YES</OK>
        <Method>NVRAM</Method>
        <Status>up</Status>
        <Protocol>up</Protocol>
      </entry>
    </IPInterfaces>
  </ShowIpInterfaceBrief>
```

Extensible Markup Language (XML)

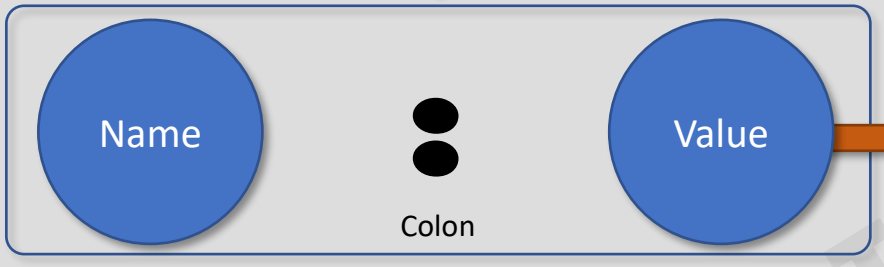
show ip interface brief.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ShowIpInterfaceBrief xmlns="ODM://built-in//show_ip_interface_brief">
  <SpecVersion>built-in</SpecVersion>
  <IPInterfaces>
    <entry>
      <Interface>FastEthernet0/0</Interface>
      <IP-Address>192.168.1.1</IP-Address>
      <OK>YES</OK>
      <Method>NVRAM</Method>
      <Status>up</Status>
      <Protocol>up</Protocol>
    </entry>
    <entry>
      <Interface>FastEthernet0/1</Interface>
      <IP-Address>192.168.2.1</IP-Address>
      <OK>YES</OK>
      <Method>NVRAM</Method>
      <Status>up</Status>
      <Protocol>up</Protocol>
    </entry>
  </IPInterfaces>
</ShowIpInterfaceBrief>
```



JavaScript Object Notation (JSON)

Curly Brackets




```
{   name_1 : value_1 ,
    name_2 : value_2 ,
    ..........
    name_N : value_N   }
```

Value
String
Number
Boolean (true, false)
Array (List)
null
Object

Array (List) → [value_1 ,..... , value_N]

Object →

```
{   name_1 : value_1 ,
    name_2 : value_2 ,
    ..........
    name_N : value_N   }
```

JavaScript Object Notation (JSON)

Code Beautify | JSON Formatter | XML Formatter | Hex Color Codes | Calculators | JSON Beautifier | Recent Links | Sitemap | Favs

XML to JSON Converter

Add to Fav | New | Save & Share

Sample

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ShowIpInterfaceBrief xmlns="ODM://built-in
    //show_ip_interface_brief">
  <SpecVersion>built-in</SpecVersion>
  <IPInterfaces>
    <entry>
      <Interface>FastEthernet0/0</Interface>
      <IP-Address>192.168.1.1</IP-Address>
      <OK>YES</OK>
      <Method>NVRAM</Method>
      <Status>up</Status>
      <Protocol>up</Protocol>
    </entry>
    <entry>
      <Interface>FastEthernet0/1</Interface>
      <IP-Address>192.168.2.1</IP-Address>
      <OK>YES</OK>
      <Method>NVRAM</Method>
      <Status>up</Status>
      <Protocol>up</Protocol>
    </entry>
  </IPInterfaces>
</ShowIpInterfaceBrief>
```

File | URL

Auto Update

2 space

XML to JSON

Output

```json
{
  "ShowIpInterfaceBrief": {
    "SpecVersion": "built-in",
    "IPInterfaces": {
      "entry": [
        {
          "Interface": "FastEthernet0/0",
          "IP-Address": "192.168.1.1",
          "OK": "YES",
          "Method": "NVRAM",
          "Status": "up",
          "Protocol": "up"
        },
        {
          "Interface": "FastEthernet0/1",
          "IP-Address": "192.168.2.1",
          "OK": "YES",
          "Method": "NVRAM",
          "Status": "up",
          "Protocol": "up"
        }
      ]
    }
  }
}
```

JavaScript Object Notation (JSON)

Code Beautify | JSON Formatter | XML Formatter | Hex Color Codes | Calculators | JSON Beautifier | Recent Links | Sitemap | Favs

XML to JSON Converter

Add to Fav | New | Save & Share

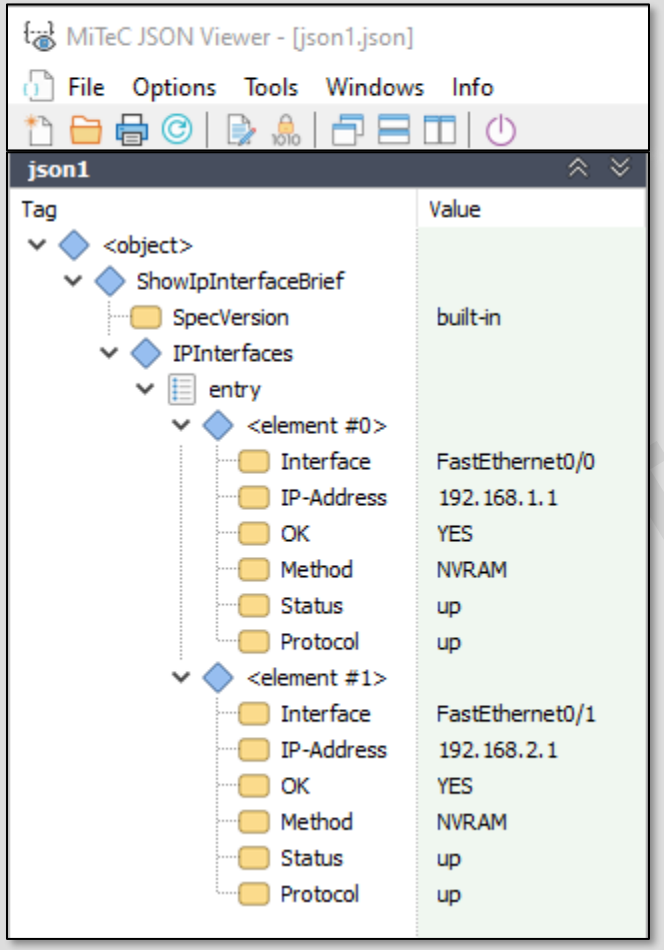
Sample | File | URL

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ShowIpInterfaceBrief xmlns="ODM://built-in
    //show_ip_interface_brief">
  <SpecVersion>built-in</SpecVersion>
  <IPInterfaces>
    <entry>
      <Interface>FastEthernet0/0</Interface>
      <IP-Address>192.168.1.1</IP-Address>
      <OK>YES</OK>
      <Method>NVRAM</Method>
      <Status>up</Status>
      <Protocol>up</Protocol>
    </entry>
    <entry>
      <Interface>FastEthernet0/1</Interface>
      <IP-Address>192.168.2.1</IP-Address>
      <OK>YES</OK>
      <Method>NVRAM</Method>
      <Status>up</Status>
      <Protocol>up</Protocol>
    </entry>
  </IPInterfaces>
</ShowIpInterfaceBrief>
```

Output

```json
{
  "ShowIpInterfaceBrief": {
    "SpecVersion": "built-in",
    "IPInterfaces": {
      "entry": [
        {
          "Interface": "FastEthernet0/0",
          "IP-Address": "192.168.1.1",
          "OK": "YES",
          "Method": "NVRAM",
          "Status": "up",
          "Protocol": "up"
        },
        {
          "Interface": "FastEthernet0/1",
          "IP-Address": "192.168.2.1",
          "OK": "YES",
          "Method": "NVRAM",
          "Status": "up",
          "Protocol": "up"
        }
      ]
    }
  }
}
```

JavaScript Object Notation (JSON)



Output

```
{
  "ShowIpInterfaceBrief": {
    "SpecVersion": "built-in",
    "IPInterfaces": {
      "entry": [
        {
          "Interface": "FastEthernet0/0",
          "IP-Address": "192.168.1.1",
          "OK": "YES",
          "Method": "NVRAM",
          "Status": "up",
          "Protocol": "up"
        },
        {
          "Interface": "FastEthernet0/1",
          "IP-Address": "192.168.2.1",
          "OK": "YES",
          "Method": "NVRAM",
          "Status": "up",
          "Protocol": "up"
        }
      ]
    }
  }
}
```

Ain't Markup Language (YAML)

- **Dictionaries** are collections of (**key: value**) pairs (the colon must be followed by a space).

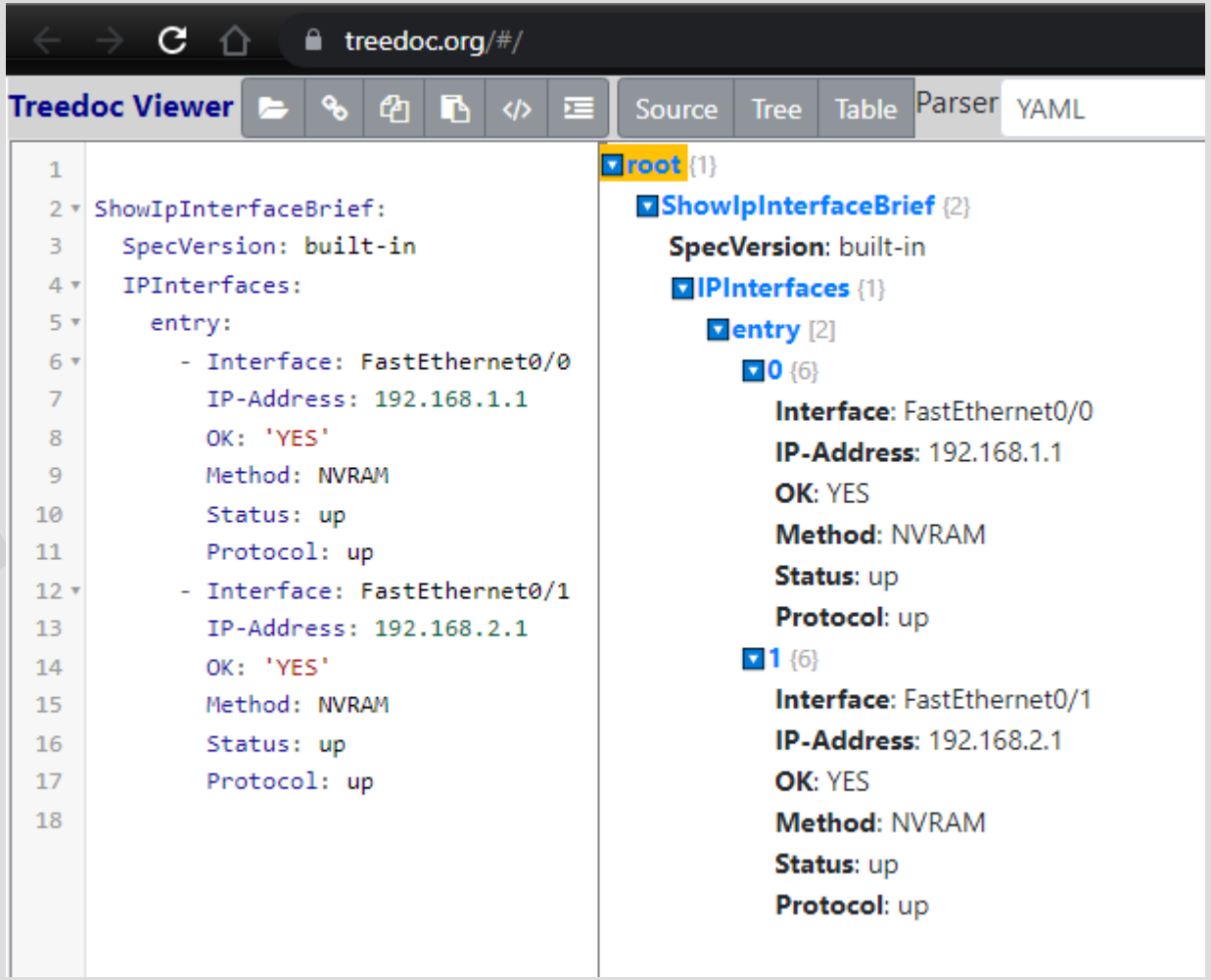
- **Lists** are collections of elements and Every element of a list is indented and starts with a **dash** and a **space**.

- **Optional inline syntax** can be used for lists and dictionaries:

router: {name: Router1, model: 7204}

interfaces: [FastEthernet0/0, FastEthernet0/1]

speed: [10, 100, Auto]



xml1.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ShowIpInterfaceBrief xmlns="ODM://built-in//show_ip_interface_brief">
  <SpecVersion>built-in</SpecVersion>
  <IPInterfaces>
    <entry>
      <Interface>FastEthernet0/0</Interface>
      <IP-Address>192.168.1.1</IP-Address>
      <OK>YES</OK>
      <Method>NVRAM</Method>
      <Status>up</Status>
      <Protocol>up</Protocol>
    </entry>
    <entry>
      <Interface>FastEthernet0/1</Interface>
      <IP-Address>192.168.2.1</IP-Address>
      <OK>YES</OK>
      <Method>NVRAM</Method>
      <Status>up</Status>
      <Protocol>up</Protocol>
    </entry>
  </IPInterfaces>
</ShowIpInterfaceBrief>
```

NETCONF

json1.json

```json
{
  "ShowIpInterfaceBrief": {
    "SpecVersion": "built-in",
    "IPInterfaces": {
      "entry": [
        {
          "Interface": "FastEthernet0/0",
          "IP-Address": "192.168.1.1",
          "OK": "YES",
          "Method": "NVRAM",
          "Status": "up",
          "Protocol": "up"
        },
        {
          "Interface": "FastEthernet0/1",
          "IP-Address": "192.168.2.1",
          "OK": "YES",
          "Method": "NVRAM",
          "Status": "up",
          "Protocol": "up"
        }
      ]
    }
  }
}
```

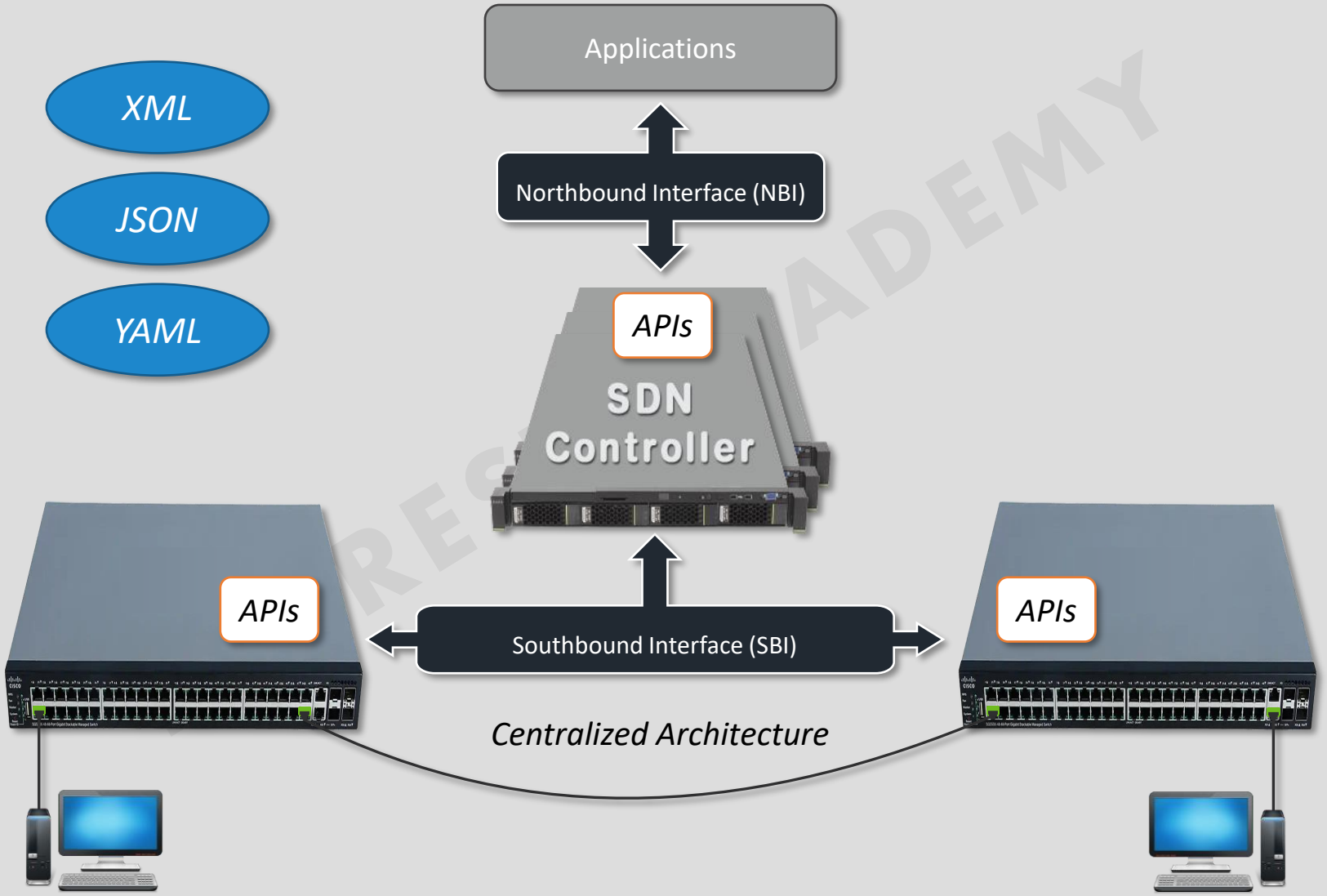
{ REST }

yaml1.yaml

```yaml
ShowIpInterfaceBrief:
  SpecVersion: built-in
  IPInterfaces:
    entry:
      - Interface: FastEthernet0/0
        IP-Address: 192.168.1.1
        OK: "YES"
        Method: NVRAM
        Status: up
        Protocol: up
      - Interface: FastEthernet0/1
        IP-Address: 192.168.2.1
        OK: "YES"
        Method: NVRAM
        Status: up
        Protocol: up
```

ANSIBLE

Application Programming Interface (API)



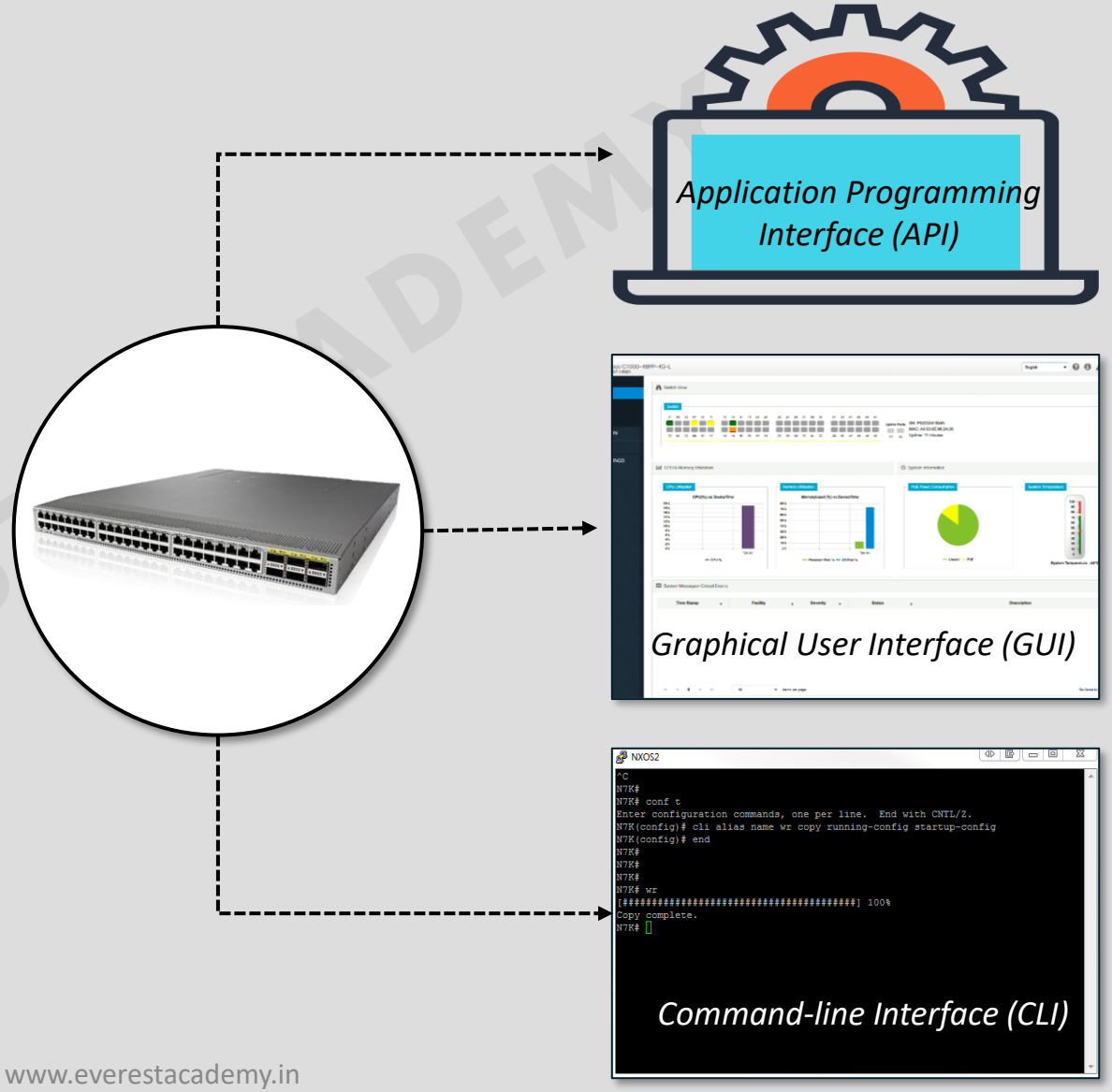
Application Programming Interface (API)

- **An application programming interface (API)** is a type of software interface or a set of programming code that enables data transmission between one software product and another.

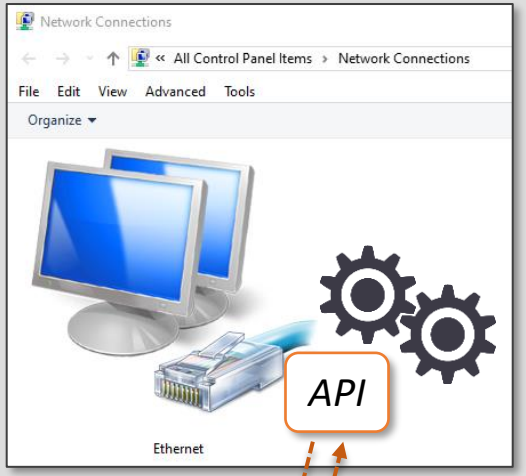
- **An API components:**

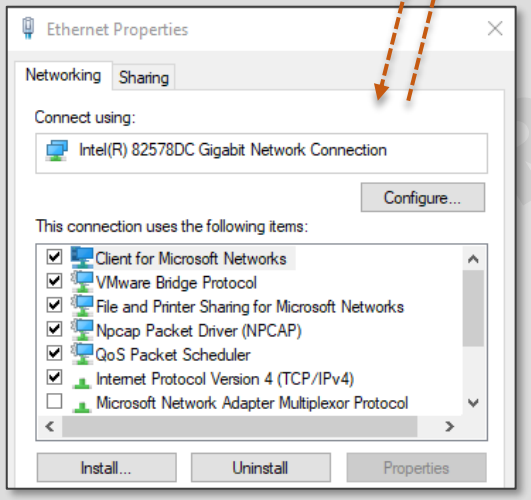
  - **The technical specification** that describes the data exchange options.

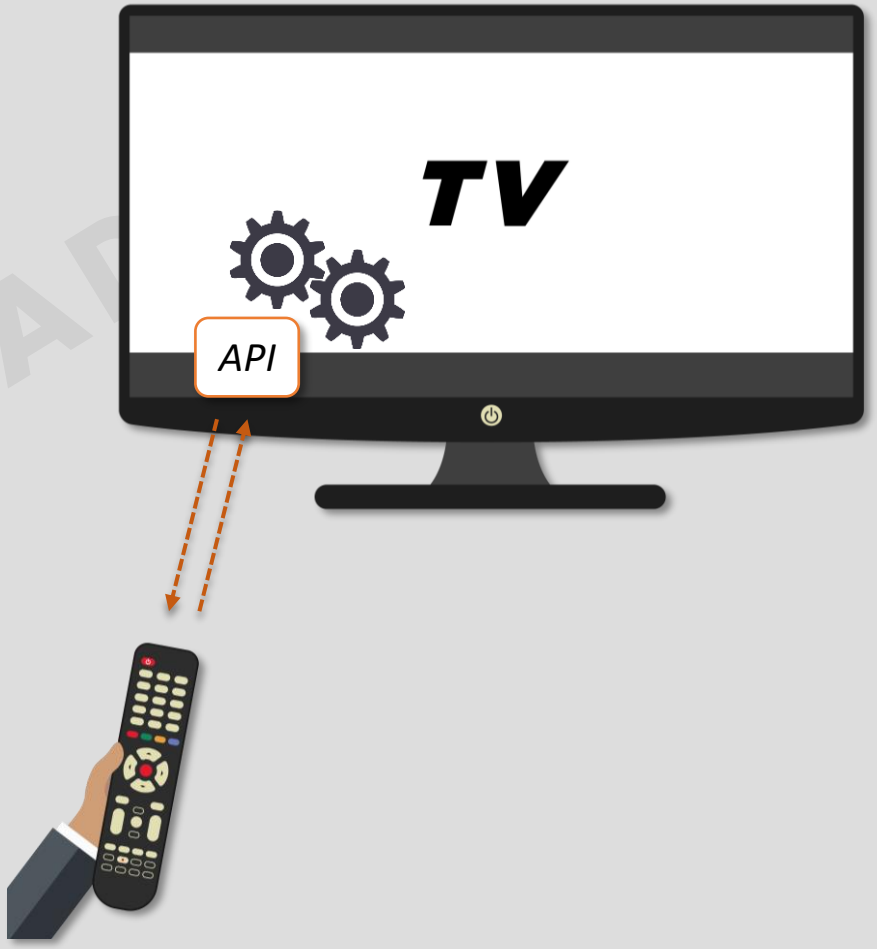
  - **The software interface** which is written to the specification that represents it.



Application Programming Interface (API)



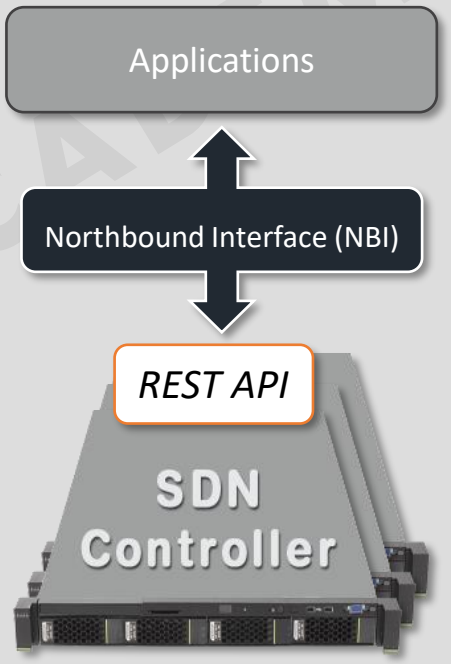




API Types and Architectural Styles

Types of APIs
• Database API.
• Operating System API.
• Remote API.
• Web API.

API Architectural Styles
• Remote Procedure Call (RPC).
• Service Object Access Protocol (SOAP).
• GraphQL.
• Representational State Transfer (REST).



REST-Based APIs

- **Representational State Transfer (REST)** is a software architectural approach for creating distributed applications that can communicate via a communication protocol.

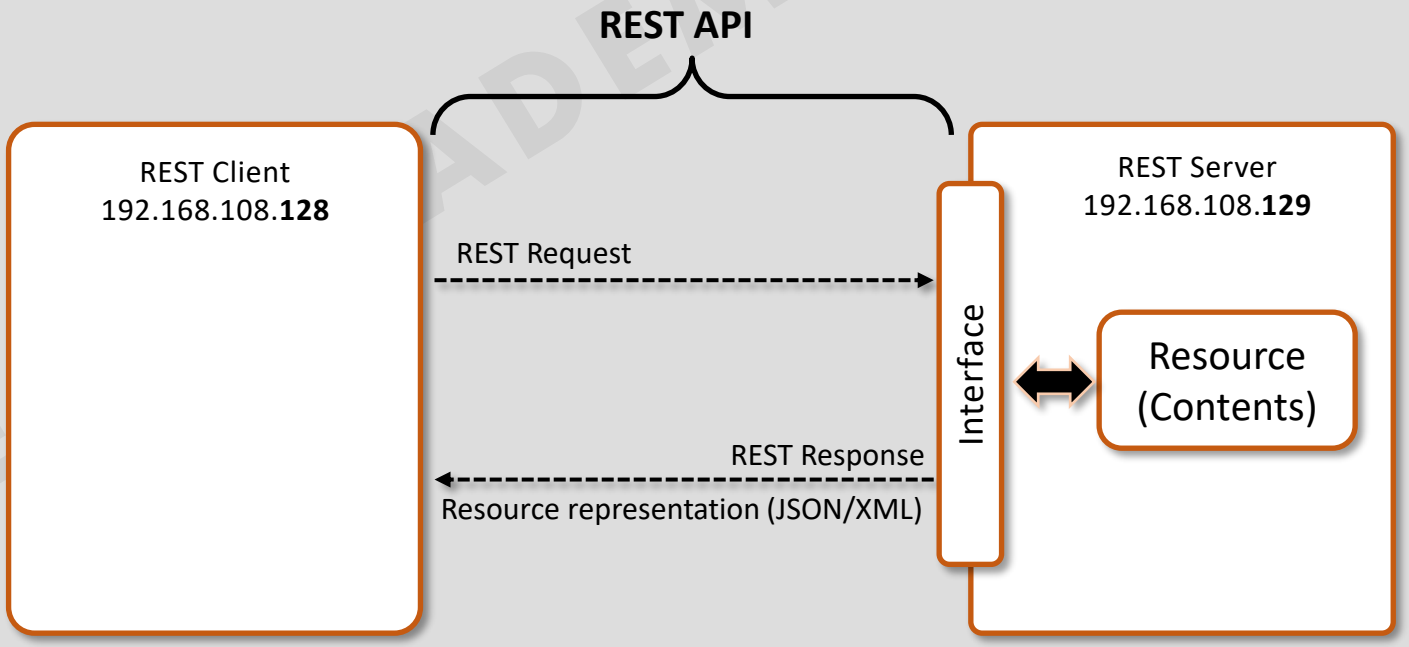
- **Elements of the REST Paradigm:**
  - Server.
  - Client.
  - Resource.

- **Each resource** is identified by

  URIs (Uniform Resource Identifier):

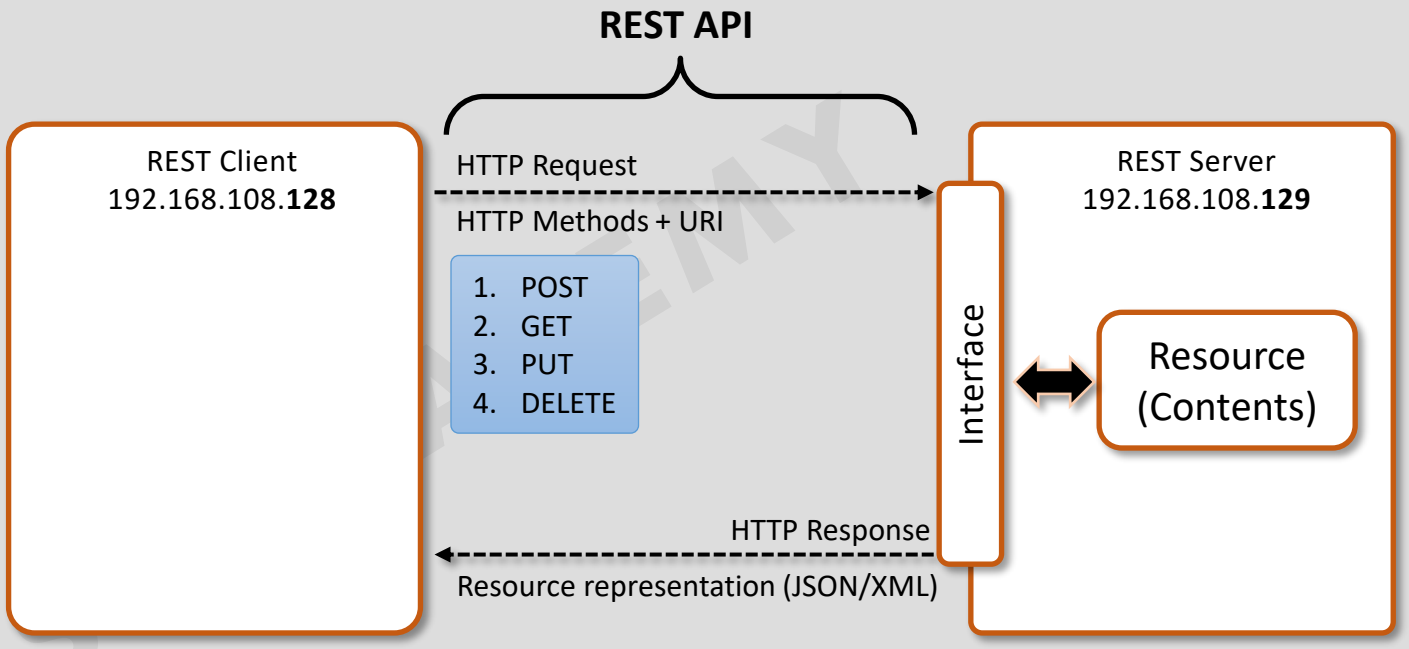
- http://192.168.108.129/devices
- http://192.168.108.129/devices/{device_id}

Protocol | Hostname/Address | Path(Resource)



REST-Based APIs

- **REST** can use HTTP protocol to communicate between client and server. A client sends a message as an ***HTTP Request*** and the server sends a response message as an ***HTTP Response***.



❑ **HTTP Request:**

Verb	URI	HTTP Version
Request Header		
Request Body		

- **Verb** represents HTTP methods such as GET, POST, PUT, DELETE, etc.

Verb	Description	
POST	Used to **c**reate a new resource.	CRUD
GET	Used to **r**ead or request data.	
PUT	Used to **u**pdate a resource.	
DELETE	Used to **d**elete a resource.	

- **URI** represents the Uniform Resource Identifier to identify the resource on the server.
- **HTTP Version i**ndicates the HTTP version.
- **Request Header c**ontains the metadata for the HTTP request. These are represented as Key-Value Pairs.
- **Request Body** contains the message content**.**

REST-Based APIs

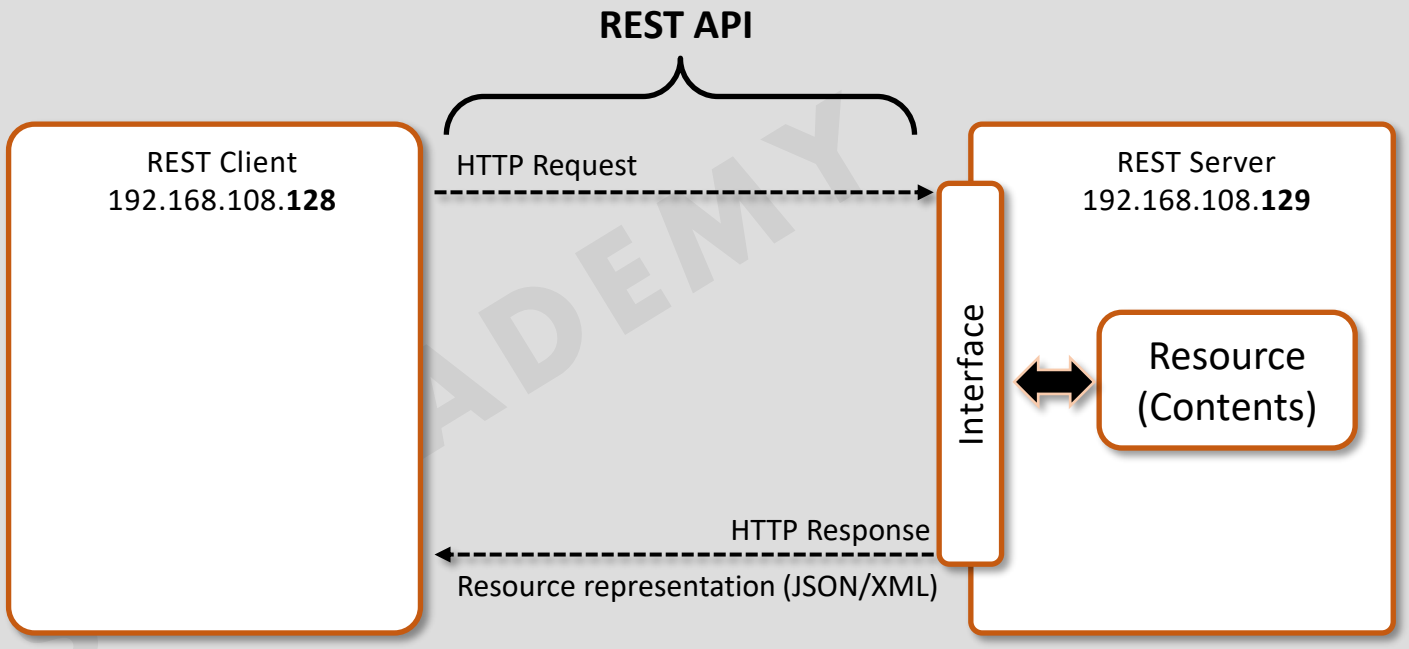
❑ HTTP Response:

Response Code	HTTP Version
Response Header	
Response Body	

- **Response Code** represents the Server status.

1. Informational response: **1xx**
2. Successful responses: **2xx**
3. Redirection messages: **3xx**
4. Client error responses: **4xx**
5. Server error responses: **5xx**

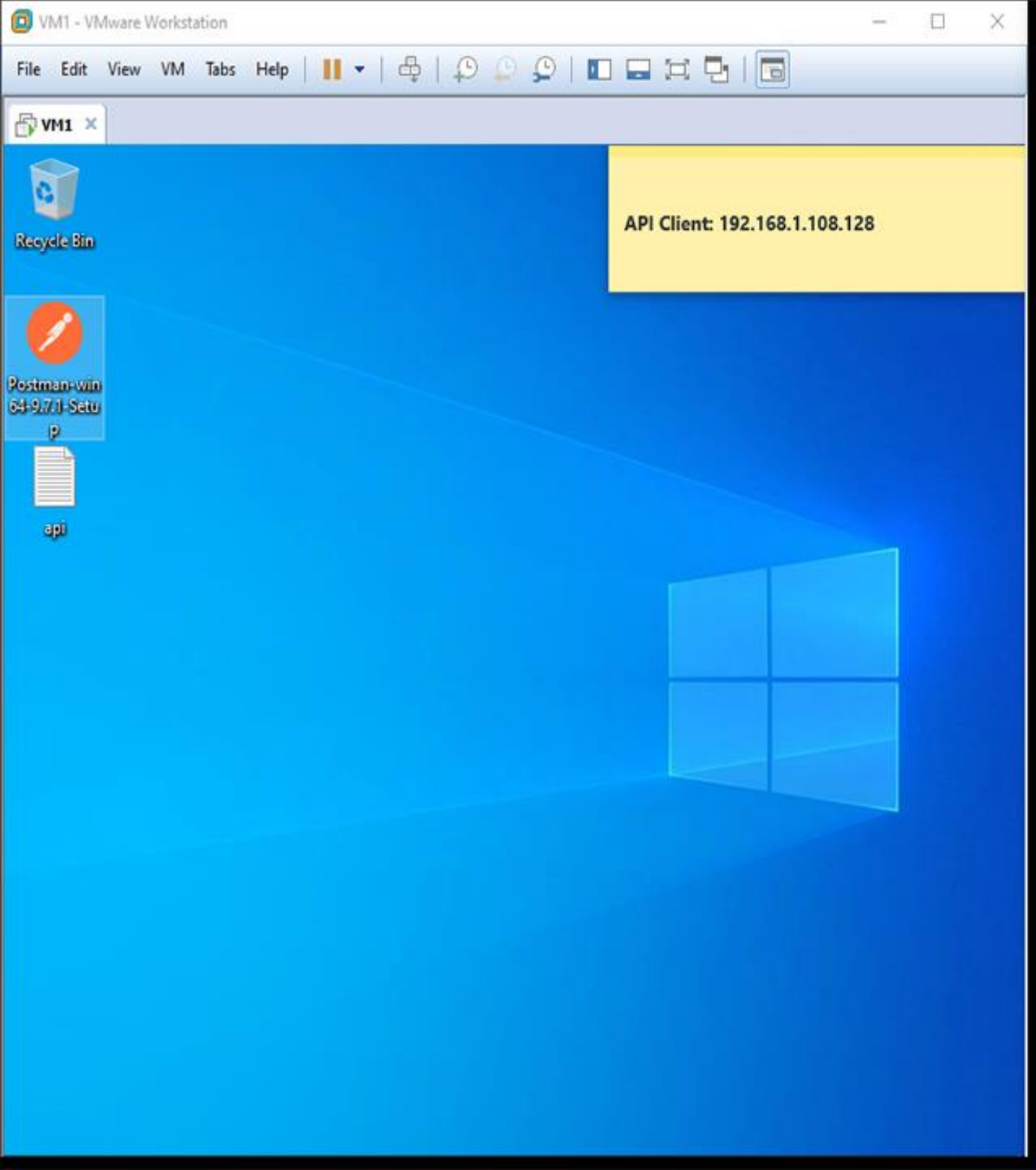
- **HTTP Version i**ndicates the HTTP version.
- **Response Header —** Contains the metadata for the HTTP response. These are represented as Key-Value Pairs.
- **Response Body** contains the message content**.**

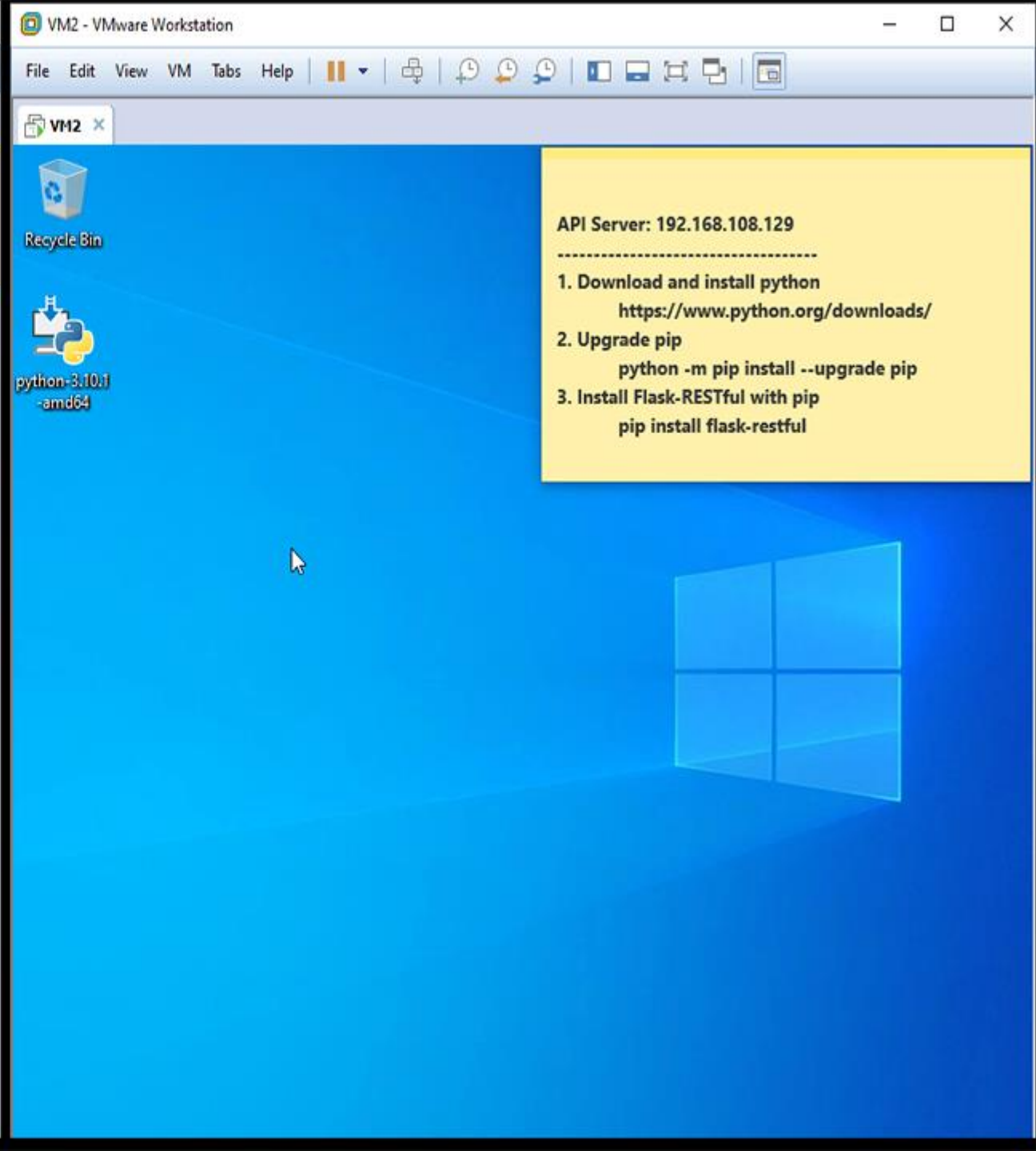


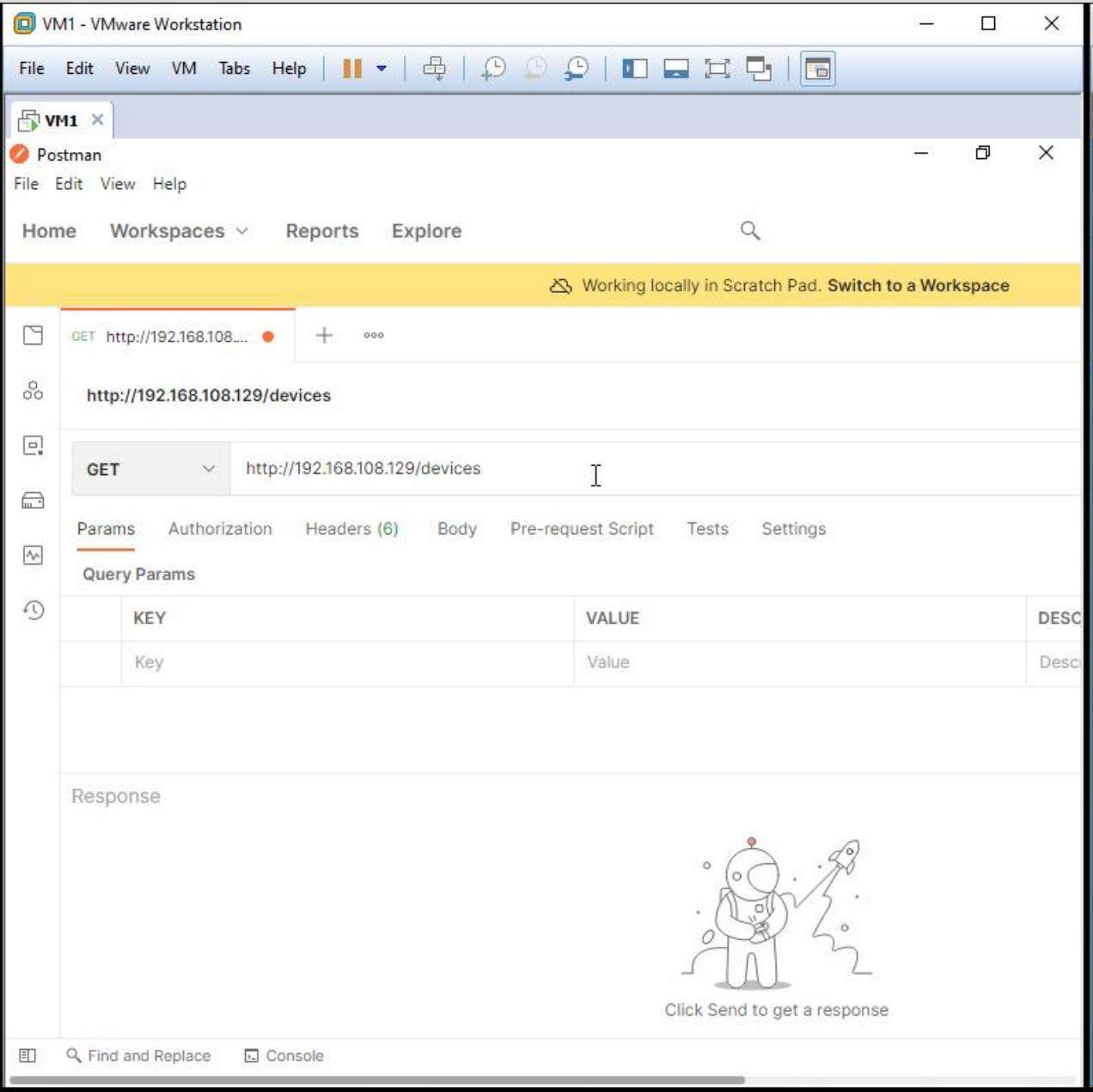
- **200 OK -** It's the basic status code to tell the client everything went good.
- **201 Created -** It indicates that the request has succeeded and has led to the creation of a resource.
- **204 No Content -** It tells the client the deletion is complete and return no response body.
- **404 Not Found –** It's response status code indicates that the server cannot find the requested resource.

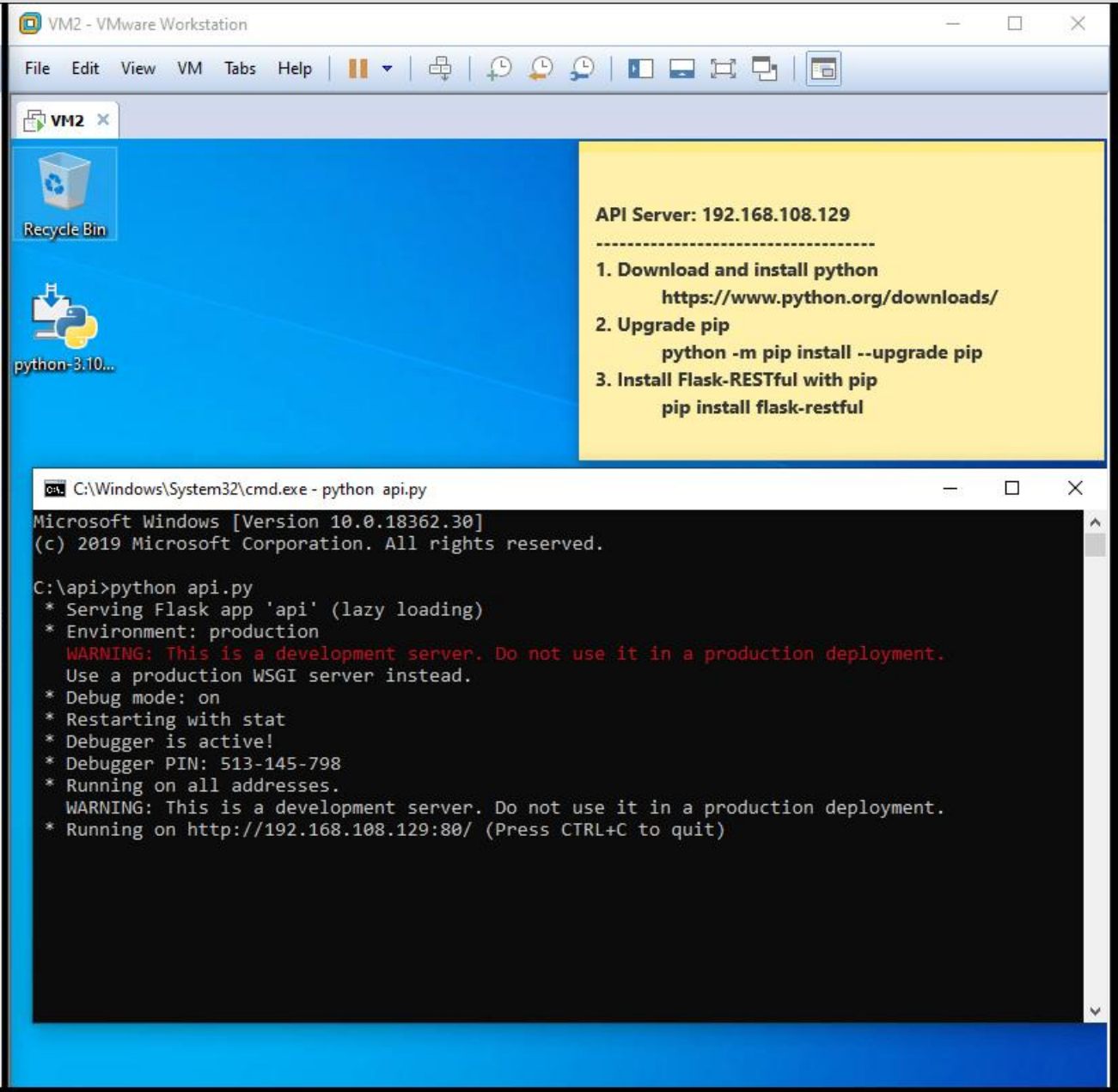
RESTful Architectural Constraints

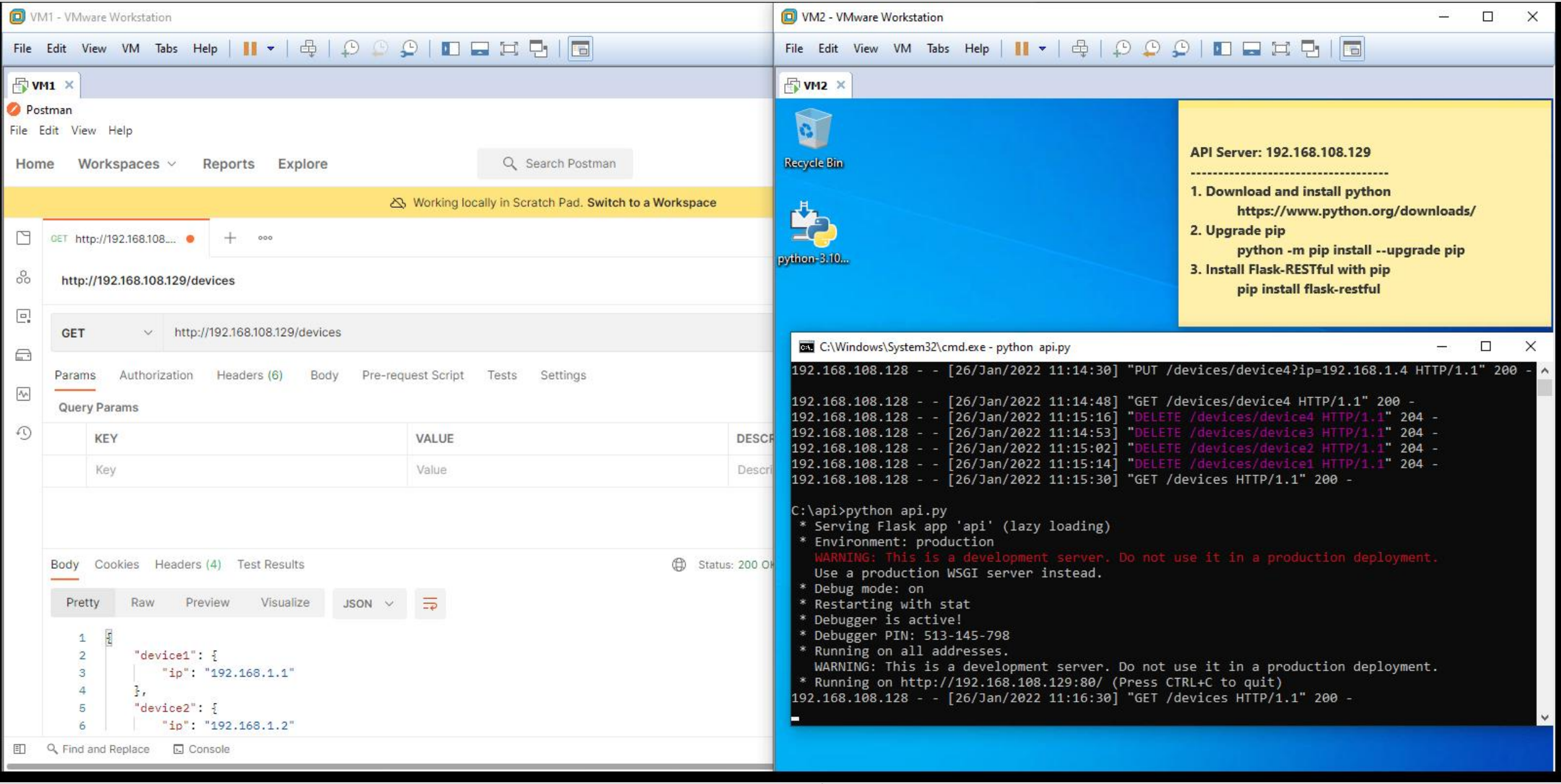
REST Architectural Constraints	
▪ **Client–server**	It means that client applications and server applications must be able to work separately without any dependency on each other. A client should know only resource URIs.
▪ **Stateless**	It means that the server will not store anything about the latest request the client made. The server treats every request as new.
▪ **Uniform interface**	It means that the resource in the server should have only one logical URI, and that should provide a way to fetch related or additional data.
▪ **Cacheable**	It means that caching should be applied to resources when applicable, and then these resources must declare themselves cacheable. Caching can be implemented on the server or client-side
▪ **Layered system**	It means that between the client who sends the request and the server who sends the response back, there might be a number of servers in the middle.
▪ **Code-on-demand**	This constraint is optional. It means that the client can request code from the server, and then the response from the server will contain some code, usually in the form of a script,

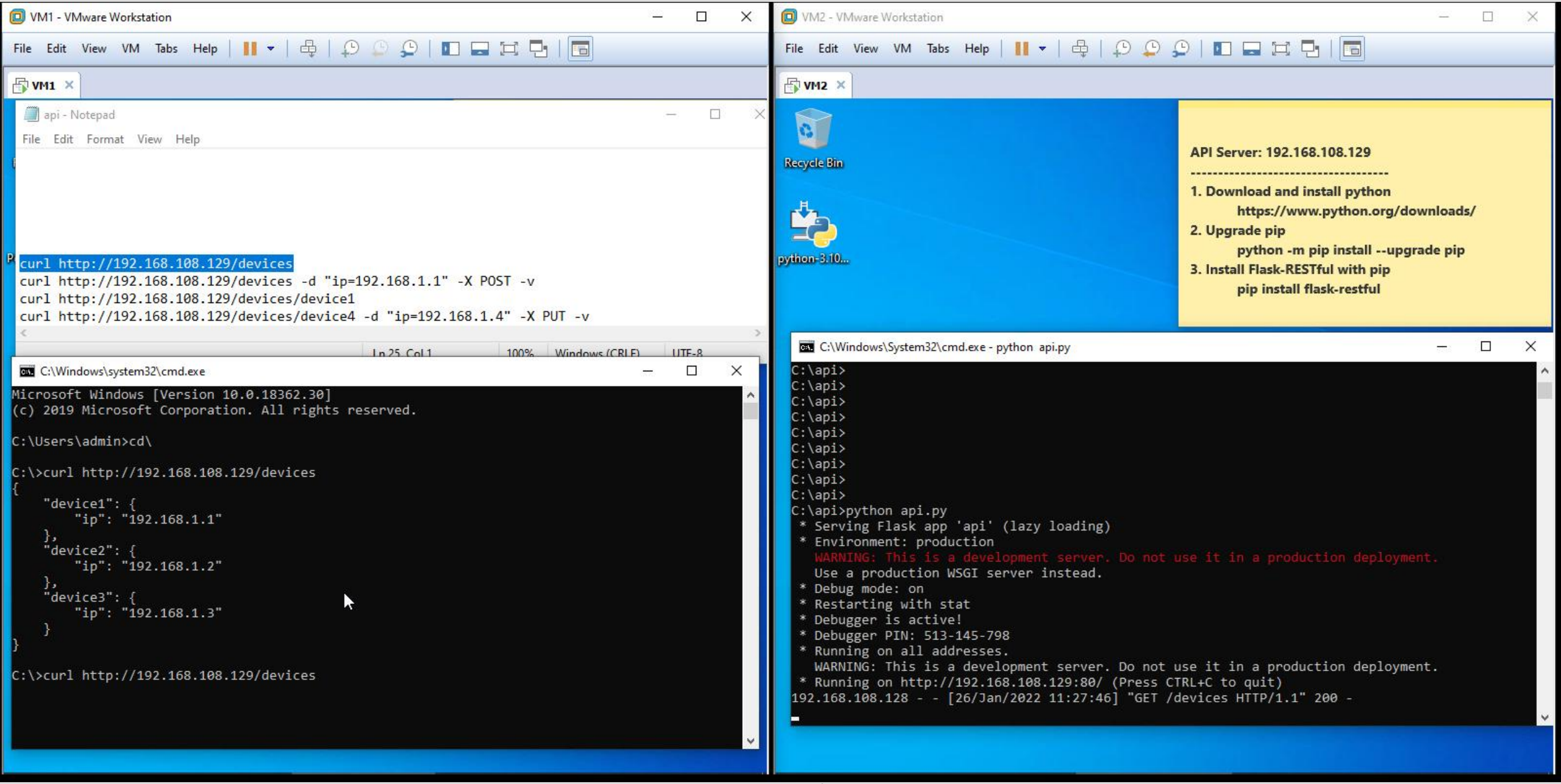

VM1 - VMware Workstation
File Edit View VM Tabs Help
VM1
Recycle Bin
Postman-win64-9.7.1-Setup
api
API Client: 192.168.1.108.128


VM2 - VMware Workstation
File Edit View VM Tabs Help
VM2
Recycle Bin
python-3.10.1-amd64
API Server: 192.168.108.129
1. Download and install python
https://www.python.org/downloads/
2. Upgrade pip
python -m pip install --upgrade pip
3. Install Flask-RESTful with pip
pip install flask-restful


VM1 - VMware Workstation
File Edit View VM Tabs Help
VM1
Postman
File Edit View Help
Home Workspaces Reports Explore
Working locally in Scratch Pad. Switch to a Workspace
GET http://192.168.108....
http://192.168.108.129/devices
GET
http://192.168.108.129/devices
Params Authorization Headers (6) Body Pre-request Script Tests Settings
Query Params
KEY VALUE DESC
Key Value
Response
Click Send to get a response
Find and Replace Console


VM2 - VMware Workstation
File Edit View VM Tabs Help
VM2
Recycle Bin
python-3.10...
API Server: 192.168.108.129
1. Download and install python
https://www.python.org/downloads/
2. Upgrade pip
python -m pip install --upgrade pip
3. Install Flask-RESTful with pip
pip install flask-restful
C:\Windows\System32\cmd.exe - python api.py
Microsoft Windows [Version 10.0.18362.30]
(c) 2019 Microsoft Corporation. All rights reserved.
C:\api>python api.py
* Serving Flask app 'api' (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger PIN: 513-145-798
* Running on all addresses.
WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://192.168.108.129:80/ (Press CTRL+C to quit)


VM1 - VMware Workstation
File Edit View VM Tabs Help
VM1
Postman
File Edit View Help
Home Workspaces Reports Explore
Search Postman
Working locally in Scratch Pad. Switch to a Workspace
GET http://192.168.108....
http://192.168.108.129/devices
GET
http://192.168.108.129/devices
Params Authorization Headers (6) Body Pre-request Script Tests Settings
Query Params
KEY VALUE
Key Value
Body Cookies Headers (4) Test Results
Status: 200 OK
Pretty Raw Preview Visualize JSON
{
"device1": {
"ip": "192.168.1.1"
},
"device2": {
"ip": "192.168.1.2"
Find and Replace Console
VM2 - VMware Workstation
File Edit View VM Tabs Help
VM2
Recycle Bin
python-3.10...
API Server: 192.168.108.129
1. Download and install python
https://www.python.org/downloads/
2. Upgrade pip
python -m pip install --upgrade pip
3. Install Flask-RESTful with pip
pip install flask-restful
C:\Windows\System32\cmd.exe - python api.py
192.168.108.128 - - [26/Jan/2022 11:14:30] "PUT /devices/device4?ip=192.168.1.4 HTTP/1.1" 200 -
192.168.108.128 - - [26/Jan/2022 11:14:48] "GET /devices/device4 HTTP/1.1" 200 -
192.168.108.128 - - [26/Jan/2022 11:15:16] "DELETE /devices/device4 HTTP/1.1" 204 -
192.168.108.128 - - [26/Jan/2022 11:14:53] "DELETE /devices/device3 HTTP/1.1" 204 -
192.168.108.128 - - [26/Jan/2022 11:15:02] "DELETE /devices/device2 HTTP/1.1" 204 -
192.168.108.128 - - [26/Jan/2022 11:15:14] "DELETE /devices/device1 HTTP/1.1" 204 -
192.168.108.128 - - [26/Jan/2022 11:15:30] "GET /devices HTTP/1.1" 200 -
C:\api>python api.py
* Serving Flask app 'api' (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger PIN: 513-145-798
* Running on all addresses.
WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://192.168.108.129:80/ (Press CTRL+C to quit)
192.168.108.128 - - [26/Jan/2022 11:16:30] "GET /devices HTTP/1.1" 200 -


VM1 - VMware Workstation
File Edit View VM Tabs Help
VM1
api - Notepad
File Edit Format View Help
curl http://192.168.108.129/devices
curl http://192.168.108.129/devices -d "ip=192.168.1.1" -X POST -v
curl http://192.168.108.129/devices/device1
curl http://192.168.108.129/devices/device4 -d "ip=192.168.1.4" -X PUT -v
Ln 25, Col 1 100% Windows (CRLF) UTF-8
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.18362.30]
(c) 2019 Microsoft Corporation. All rights reserved.
C:\Users\admin>cd\
C:\>curl http://192.168.108.129/devices
{
"device1": {
"ip": "192.168.1.1"
},
"device2": {
"ip": "192.168.1.2"
},
"device3": {
"ip": "192.168.1.3"
}
}
C:\>curl http://192.168.108.129/devices
VM2 - VMware Workstation
File Edit View VM Tabs Help
VM2
Recycle Bin
python-3.10...
API Server: 192.168.108.129
1. Download and install python
https://www.python.org/downloads/
2. Upgrade pip
python -m pip install --upgrade pip
3. Install Flask-RESTful with pip
pip install flask-restful
C:\Windows\System32\cmd.exe - python api.py
C:\api>
C:\api>python api.py
* Serving Flask app 'api' (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger PIN: 513-145-798
* Running on all addresses.
WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://192.168.108.129:80/ (Press CTRL+C to quit)
192.168.108.128 - - [26/Jan/2022 11:27:46] "GET /devices HTTP/1.1" 200 -

VM1 - VMware Workstation

File Edit View VM Tabs Help

VM1

*api - Notepad

File Edit Format View Help

```
curl http://192.168.108.129/devices
curl http://192.168.108.129/devices -d "ip=192.168.1.1" -X POST -v
curl http://192.168.108.129/devices/device4
curl http://192.168.108.129/devices/device4 -d "ip=192.168.1.4" -X PUT -v
curl http://192.168.108.129/devices/device4 -X DELETE -v
```

Ln 29, Col 1 100% Windows (CRLF) UTF-8

C:\Windows\system32\cmd.exe

```
C:\>curl http://192.168.108.129/devices/device1 -X DELETE -v
*   Trying 192.168.108.129...
* TCP_NODELAY set
* Connected to 192.168.108.129 (192.168.108.129) port 80 (#0)
> DELETE /devices/device1 HTTP/1.1
> Host: 192.168.108.129
> User-Agent: curl/7.55.1
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 204 NO CONTENT
< Content-Type: application/json
< Server: Werkzeug/2.0.2 Python/3.10.1
< Date: Wed, 26 Jan 2022 06:00:22 GMT
<
* Closing connection 0

C:\>
C:\>curl http://192.168.108.129/devices
{}

C:\>
```

VM2 - VMware Workstation

File Edit View VM Tabs Help

VM2

Recycle Bin

python-3.10...

API Server: 192.168.108.129

---

1. Download and install python
   https://www.python.org/downloads/
2. Upgrade pip
   python -m pip install --upgrade pip
3. Install Flask-RESTful with pip
   pip install flask-restful

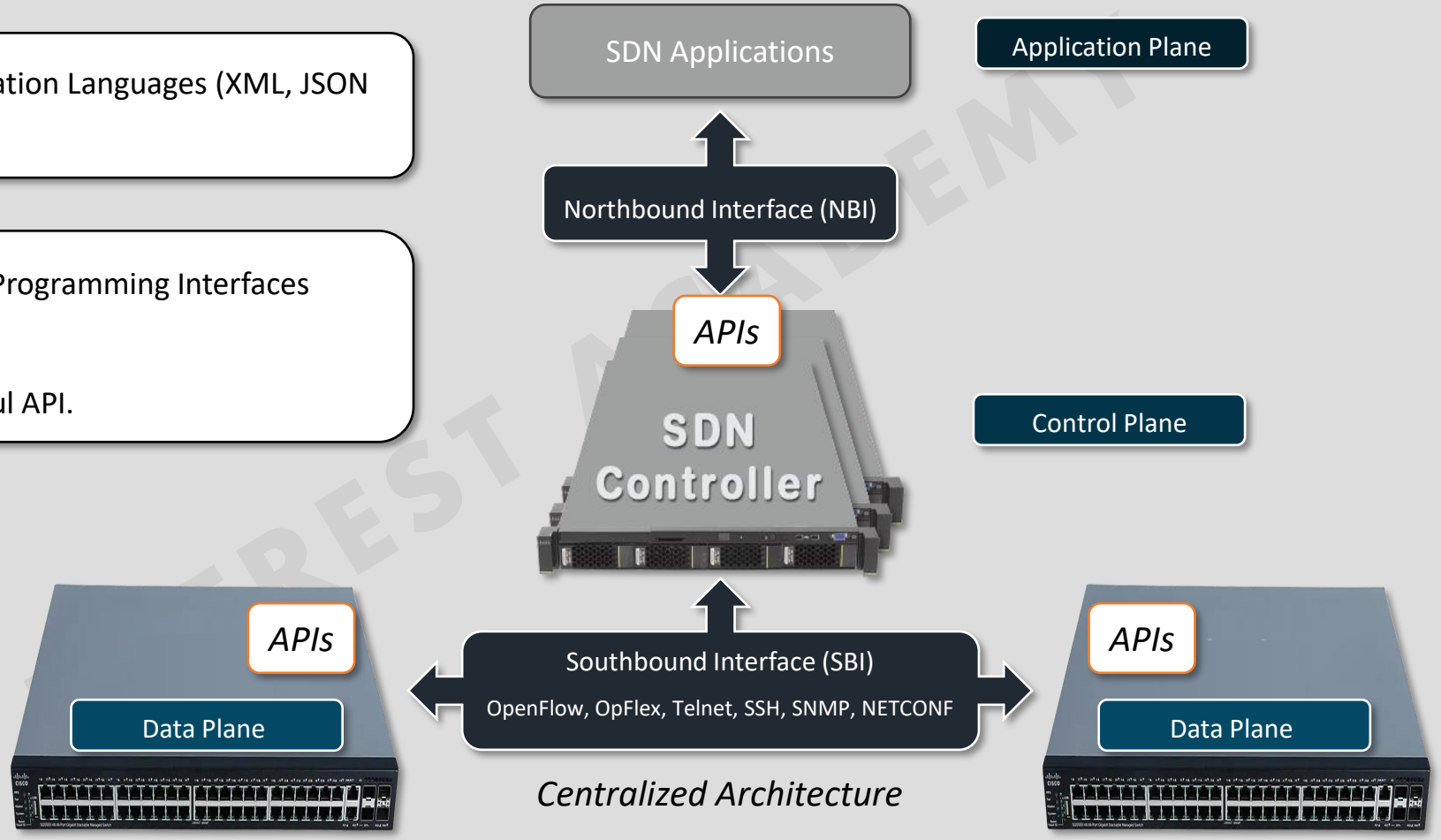
C:\Windows\System32\cmd.exe - python api.py

```
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 513-145-798
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://192.168.108.129:80/ (Press CTRL+C to quit)
192.168.108.128 - - [26/Jan/2022 11:27:46] "GET /devices HTTP/1.1" 200 -
dict_keys(['device1', 'device2', 'device3', 'device4'])
192.168.108.128 - - [26/Jan/2022 11:28:12] "POST /devices HTTP/1.1" 201 -
192.168.108.128 - - [26/Jan/2022 11:28:32] "GET /devices/device4 HTTP/1.1" 200 -
192.168.108.128 - - [26/Jan/2022 11:29:06] "PUT /devices/device4 HTTP/1.1" 200 -
192.168.108.128 - - [26/Jan/2022 11:29:16] "GET /devices/device4 HTTP/1.1" 200 -
192.168.108.128 - - [26/Jan/2022 11:29:38] "DELETE /devices/device4 HTTP/1.1" 204 -
192.168.108.128 - - [26/Jan/2022 11:30:03] "GET /devices HTTP/1.1" 200 -
192.168.108.128 - - [26/Jan/2022 11:30:12] "DELETE /devices/device3 HTTP/1.1" 204 -
192.168.108.128 - - [26/Jan/2022 11:30:18] "DELETE /devices/device2 HTTP/1.1" 204 -
192.168.108.128 - - [26/Jan/2022 11:30:22] "DELETE /devices/device1 HTTP/1.1" 204 -
192.168.108.128 - - [26/Jan/2022 11:30:27] "GET /devices HTTP/1.1" 200 -
```

SDN Solutions

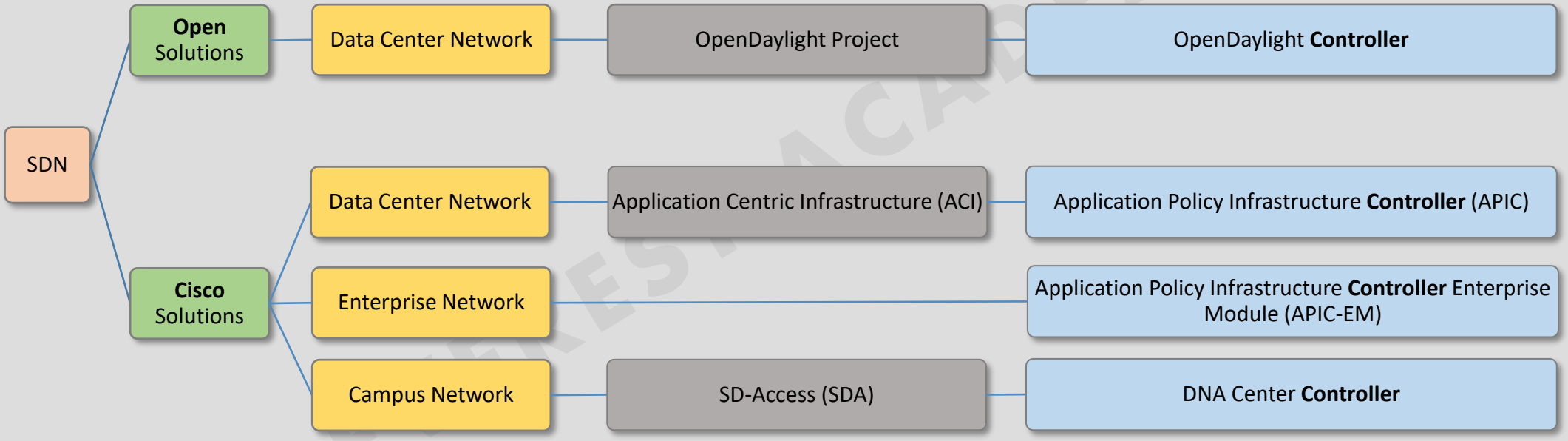
- Data Serialization Languages (XML, JSON and YAML).

- Application Programming Interfaces (APIs).
  - RESTful API.



Centralized Architecture

SDN Solutions



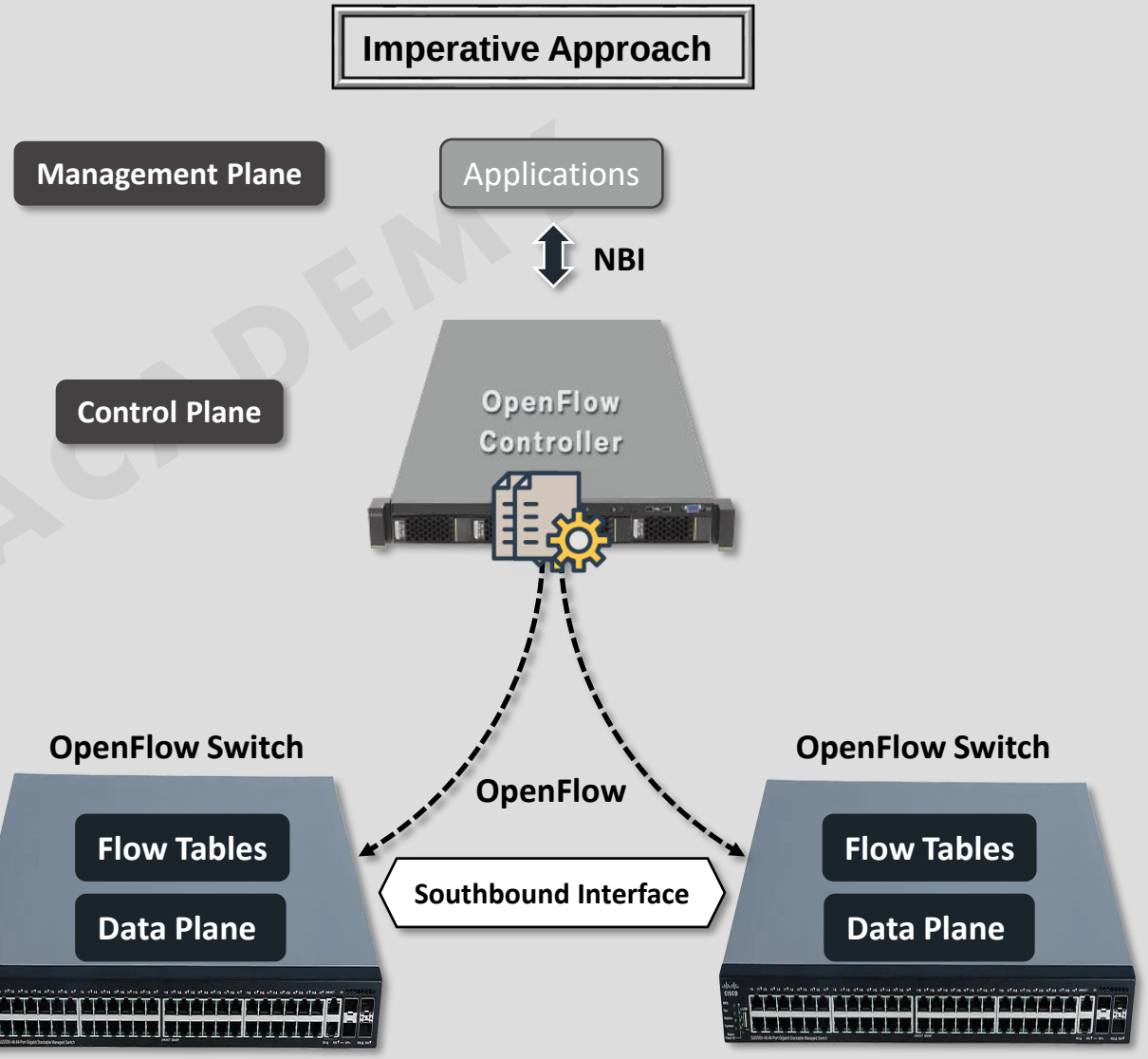
OpenFlow Protocol and OpFlex Protocol

- **OpenFlow** is a communications protocol that gives access to the data plane of a network switch or router over the network.

- **Open Networking Foundation (ONF)**

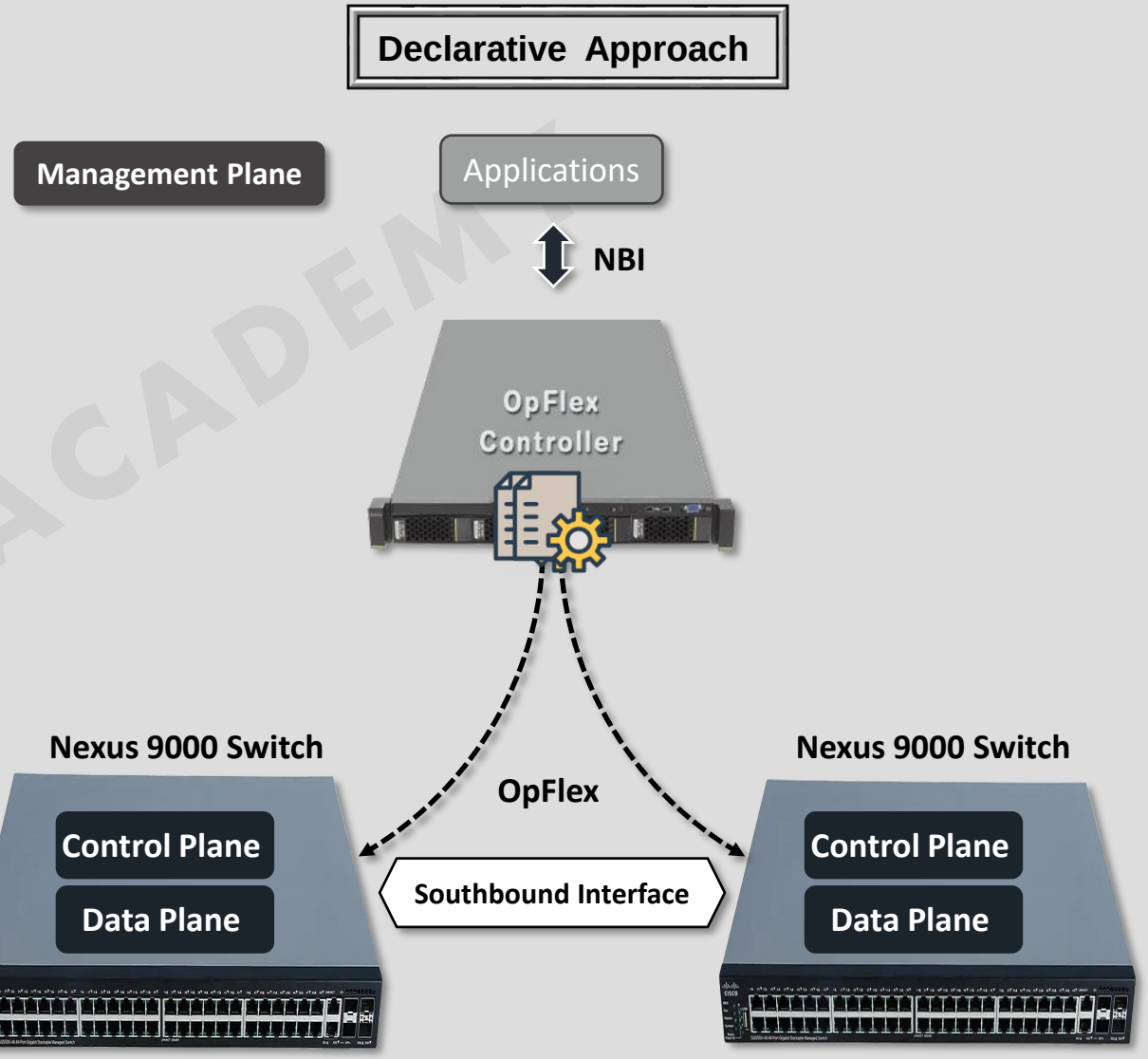
- **Basic operation of an OpenFlow solution:**

- The controller populates the switch with flow table entries.
- The switch evaluates incoming packets and finds a matching flow, then performs the associated action.
- If no match is found, the switch forwards the packet to the controller for instructions on how to deal with the packet.
- Typically the controller will update the switch with new flow entries as new packet patterns are received, so that the switch can deal with them locally.



OpenFlow Protocol and OpFlex Protocol

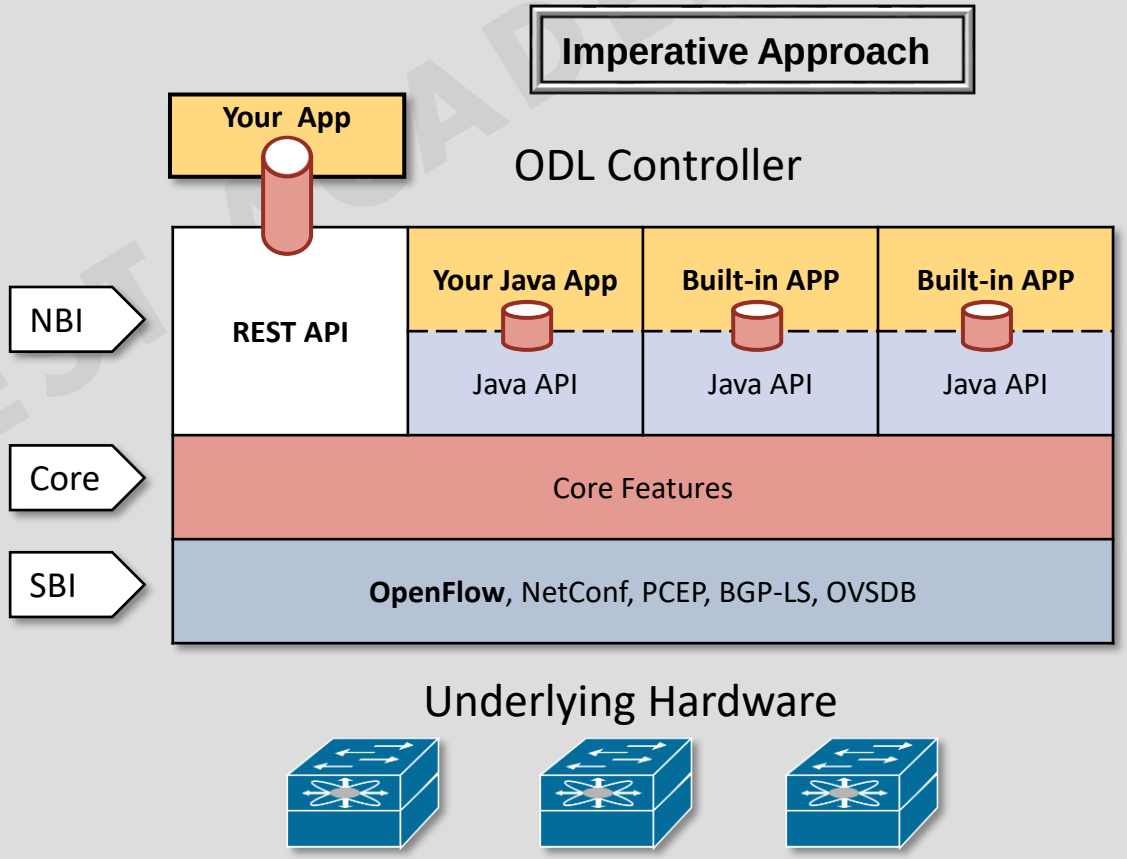
- **OpFlex** is an open and **extensible policy protocol** used to transfer **abstract policy** in **XML** or **JSON** between a policy controller such as the **Cisco APIC** and any device, including **hypervisor switches**, **physical switches**, and **Layer 4 through 7 network services**.



OpenDaylight Controller

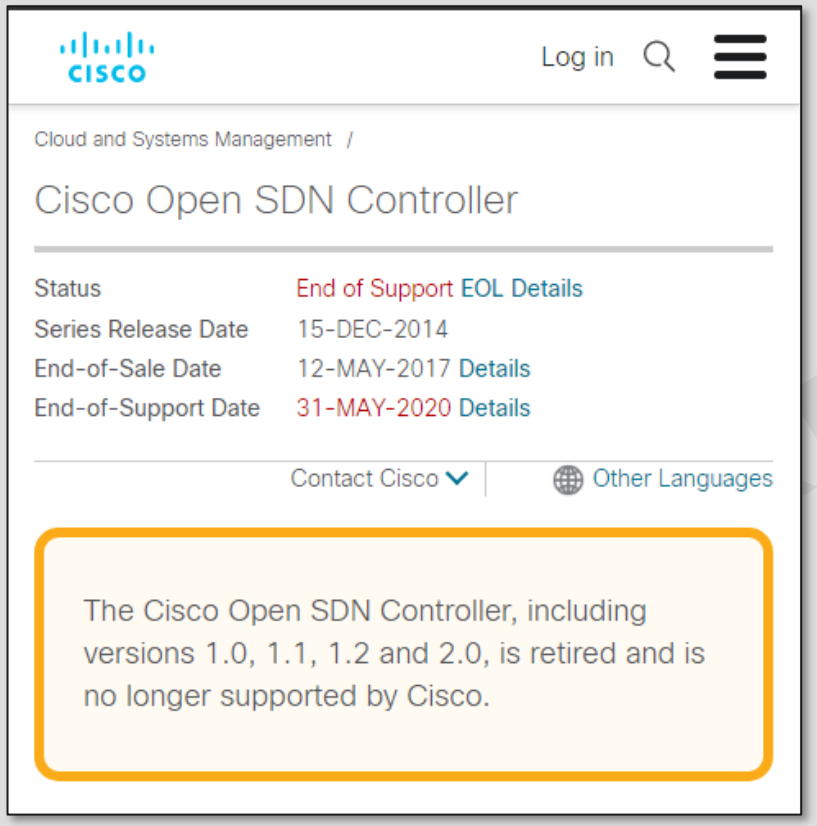
- **OpenDaylight Controller** is an open-source SDN controller, created by an open-source effort of the **OpenDaylight project** under the **Linux foundation**, built with the intent to have a **common SDN controller code base** from which vendors could then take the code and add further features and support to create SDN controller products.

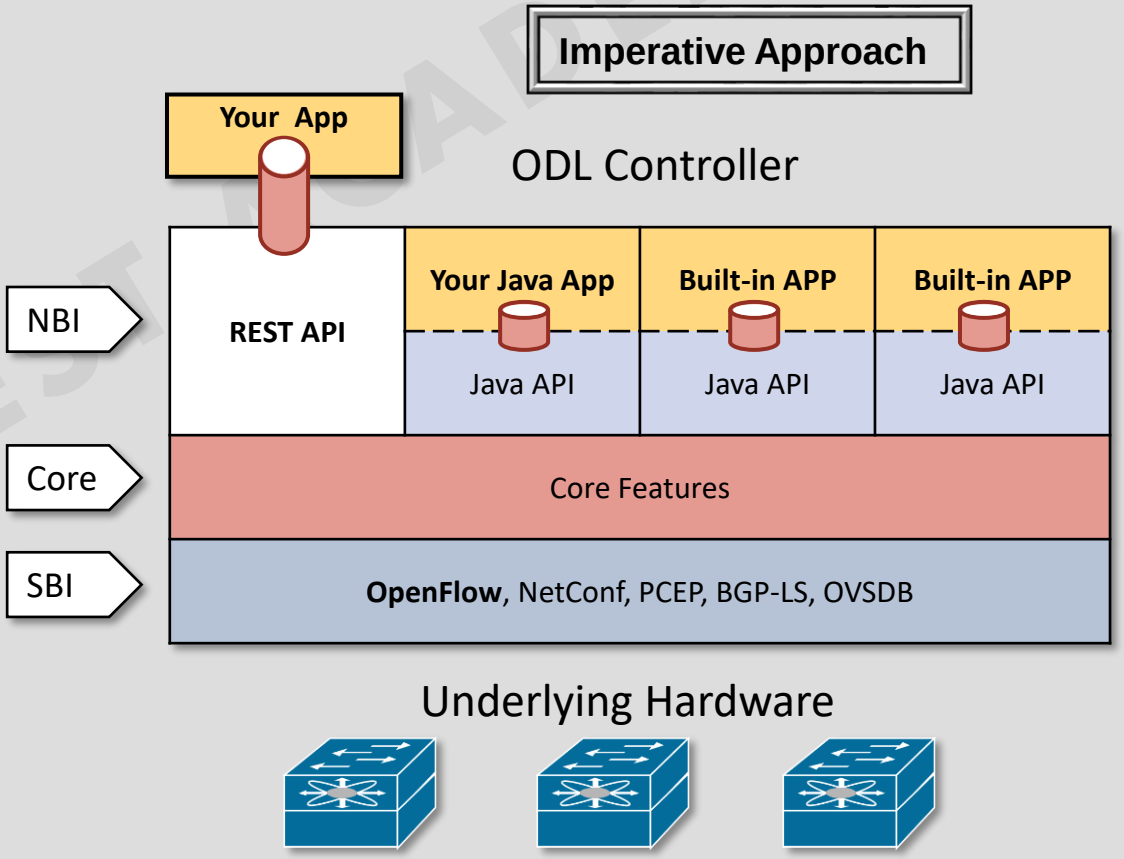
Release Name	Release Date
Hydrogen	February **2014**
Helium	October 2014
Lithium	June 2015
Beryllium	February 2016
Boron	November 2016
Carbon	June 2017
Nitrogen	September 2017
Oxygen	March 2018
Fluorine	August 2018
Neon	March 2019
Sodium	September 2019
Magnesium	March 2020
Aluminium	September **2020**

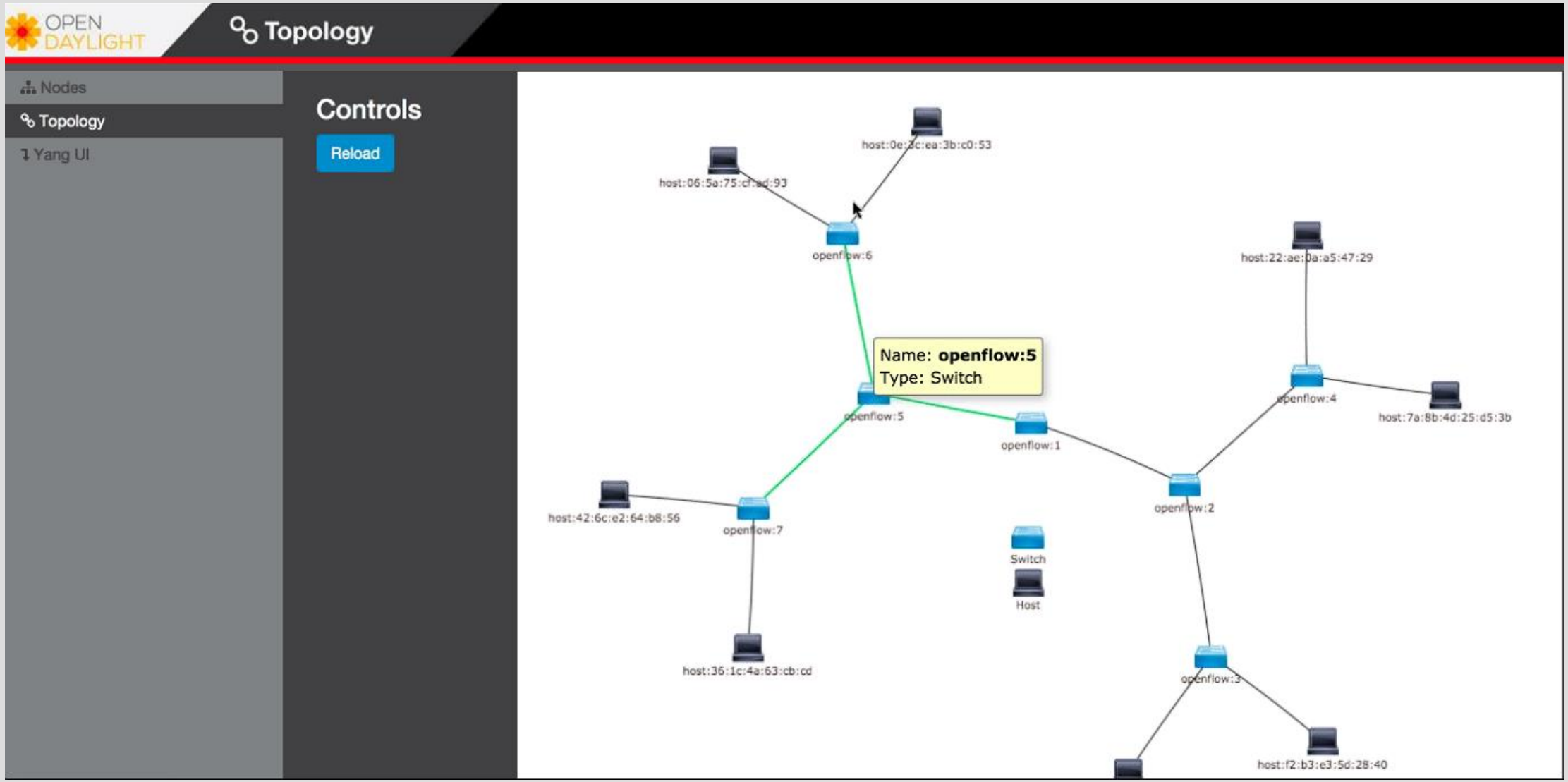


OpenDaylight Controller

- **OpenDaylight Controller** is an open-source SDN controller, created by an open-source effort of the **OpenDaylight project** under the **Linux foundation**, built with the intent to have a **common SDN controller code base** from which vendors could then take the code and add further features and support to create SDN controller products.






OPEN DAYLIGHT
Topology
Nodes
Topology
Yang UI
Controls
Reload
host:0e:3c:ea:3b:c0:53
host:06:5a:75:cf:ad:93
openflow:6
host:22:ae:0a:a5:47:29
Name: openflow:5
Type: Switch
openflow:5
openflow:1
openflow:4
host:7a:8b:4d:25:d5:3b
host:42:6c:e2:64:b8:56
openflow:7
openflow:2
Switch
Host
host:36:1c:4a:63:cb:cd
openflow:3
host:f2:b3:e3:5d:28:40

Cisco Application Centric Infrastructure (ACI)

- **Cisco Application Centric Infrastructure (ACI)** is a Cisco's *data center* SDN solution that defines its network infrastructure based upon network *policies*.

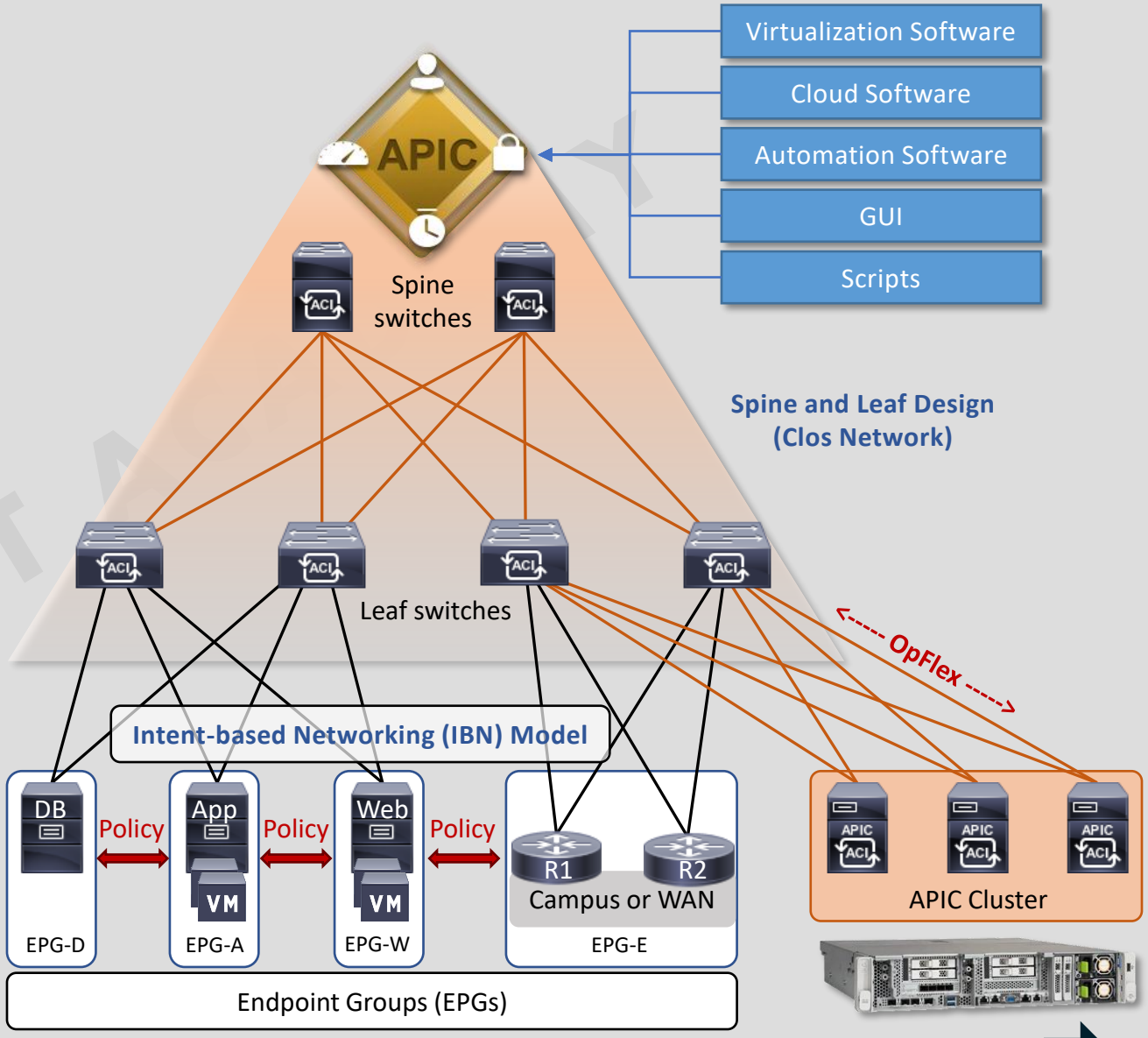
- **Cisco** has created the *ACI Fabric OS*, which is run by all systems within the ACI network. This shared OS makes it possible for the various switches within the ACI network to *translate policies* into *infrastructure designs*.

❑ **Cisco ACI components:**

1. **Application Policy Infrastructure Controller (APIC)** is the main component of the ACI solution. It provides automation and management for the Cisco ACI fabric, policy enforcement, and health monitoring.

2. **Nexus 9000 Switches** use the *ACI Fabric OS* to communicate with *APIC* and create *infrastructure based on policies*. They can be either *Spine* (distribution) or *Leaf* (access) switches.

- **The APIC** pushes the created policies to the switches in the network using the *OpFlex protocol*, with the *partially distributed control plane* in each switch building the *forwarding table entries* to support the policies learned from the controller.



APIC

System | Tenants | Fabric | Virtual Networking | L4-L7 Services | Admin | Operations | Apps

Inventory | Fabric Policies | Access Policies

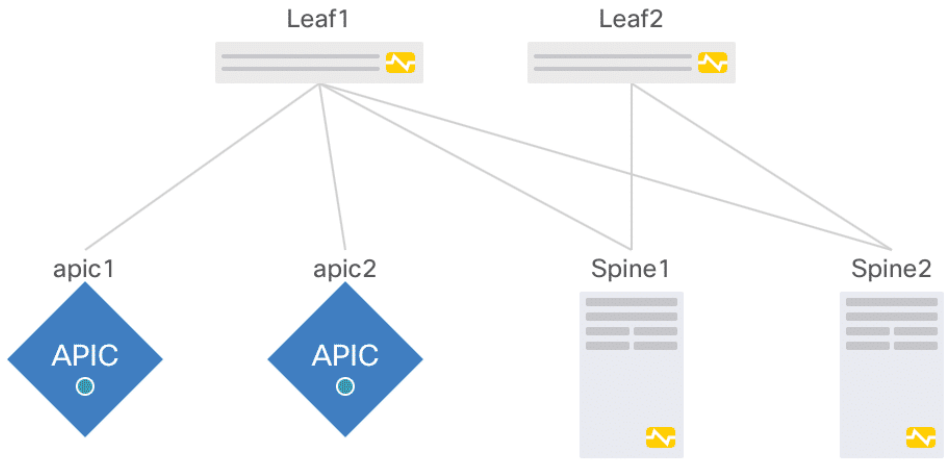
Inventory

- Quick Start
- Topology
- Pod 1
- Pod Fabric Setup Policy
- Fabric Membership
- Disabled Interfaces and Decommissioned Switches
- Duplicate IP Usage

❖ The APIC software is delivered on turn-key UCS C-Series server appliances.

Topology

Summary | Topology | Global End-Points | Interface

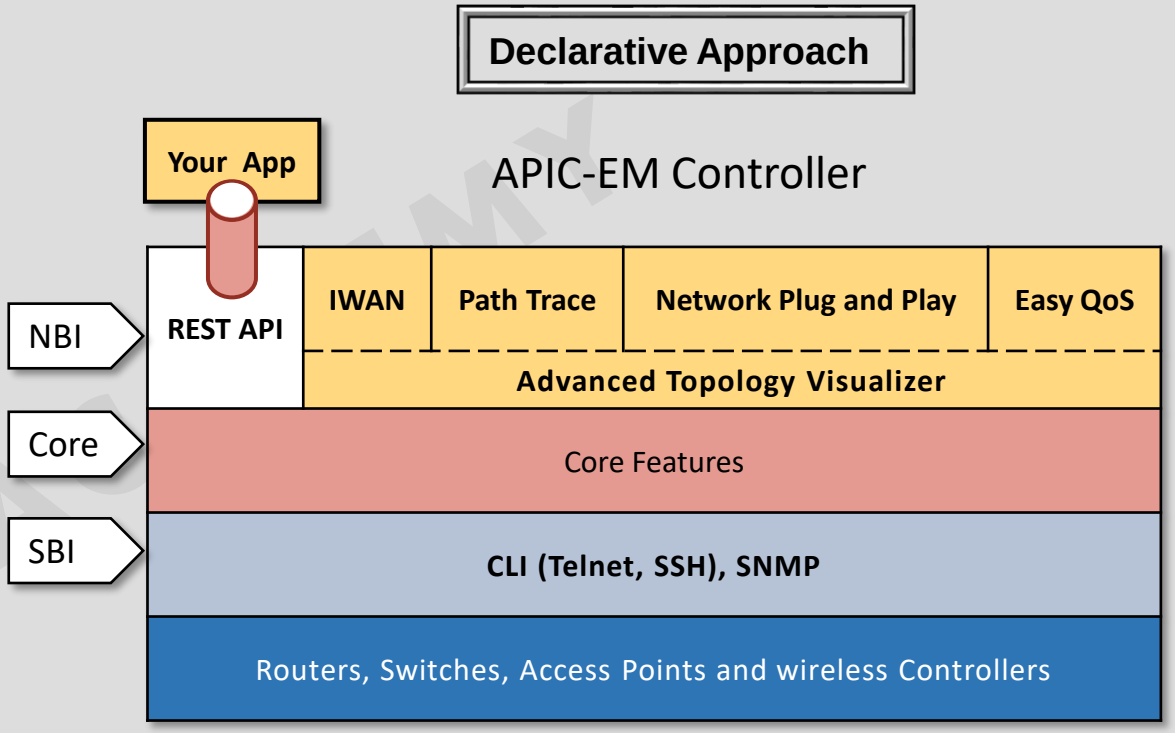


Application Policy Infrastructure Controller Enterprise Module (APIC-EM)

- **APIC-EM** is an SDN controller for policy based automation of the network infrastructure, simplifying deployment and network operations. It delivers software-defined networking to the enterprise branch, campus, and WAN.

- **APIC-EM** has built in applications such as IWAN, Path Trace, Easy QoS and Network PnP that support enterprise *routers*, *switches*, *Access Points* and *wireless controllers*.

1) **Intelligent WAN (IWAN)** - automates the configuration of advanced IWAN features on Cisco routers.
2) **Plug and Play (PnP) -** delivers zero-touch deployment of Cisco Enterprise Network routers, switches and wireless devices.
3) **Easy QoS -** policy-based QoS configuration for the network.
4) **Path Trace -** traces application paths through the network and provides statistics for each hop along the path.



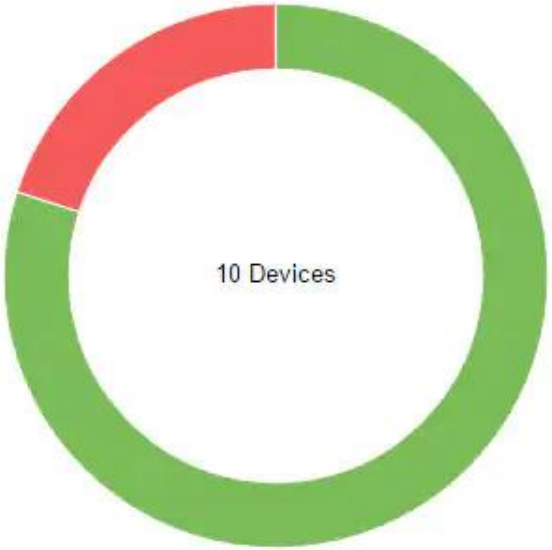
- **The Cisco APIC-EM** is available as an ISO image that can be downloaded from Cisco.com and installed on any Cisco UCS server that meets the minimum server (bare-metal hardware) requirements or in a virtual machine.

- **Hardware Requirements:** 6 CPU cores with 2.4Ghz at least, 64GB of RAM and a 500GB hard disk.

- **Cisco Digital Network Architecture (DNA) Center** is the successor to **Cisco APIC-EM**, It provides new features to automate the management of network devices, assure the health of your network, and integrate with external systems.

DASHBOARD SYSTEM HEALTH SYSTEM INFO

DEVICE INVENTORY

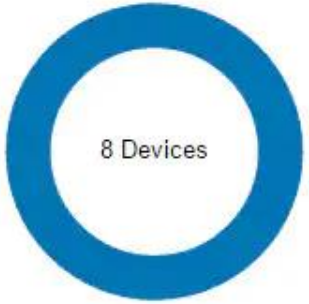


Managed (8)

In-Progress (0)

Collection Failure (2)

DISCOVERY - UNREACHABLE DEVICES



BRANCH SITES

To set up branch sites, use IWAN app!

PNP PROJECTS

To set up zero day configuration for your new device, use PnP app!

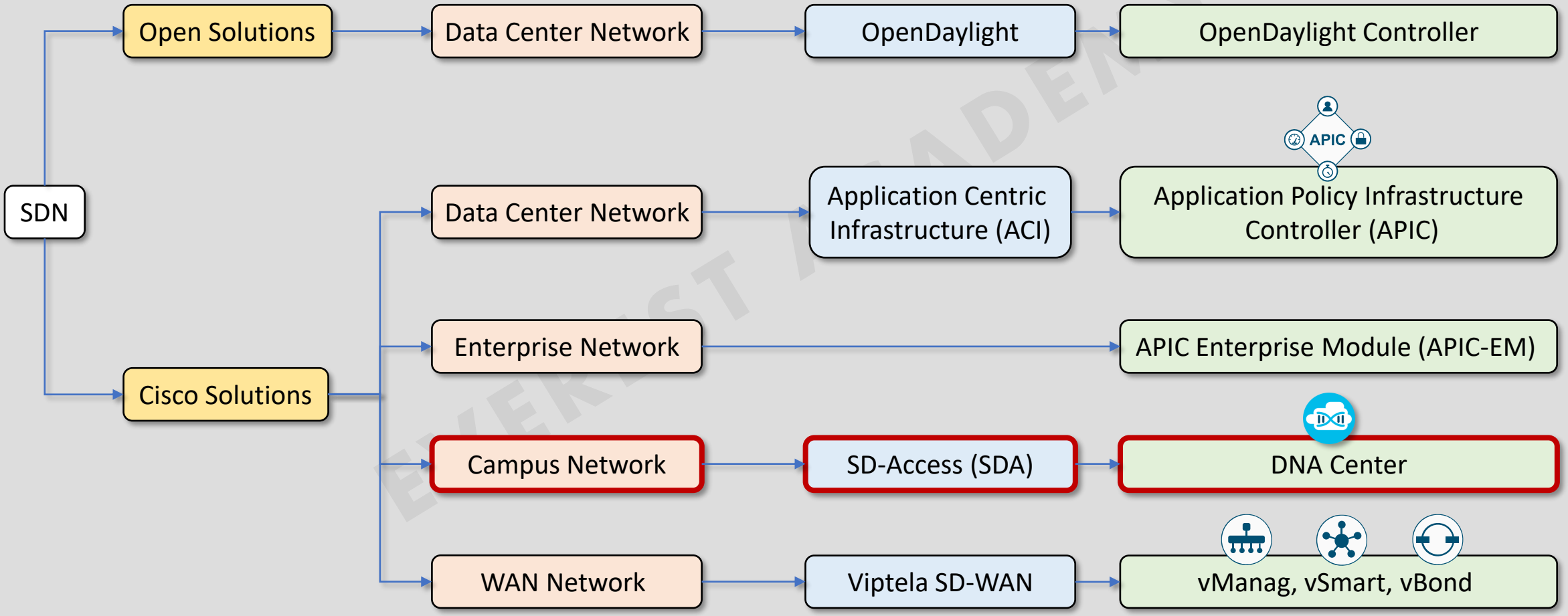
EASYQOS SCOPES

Define scopes to apply easyqos policy.

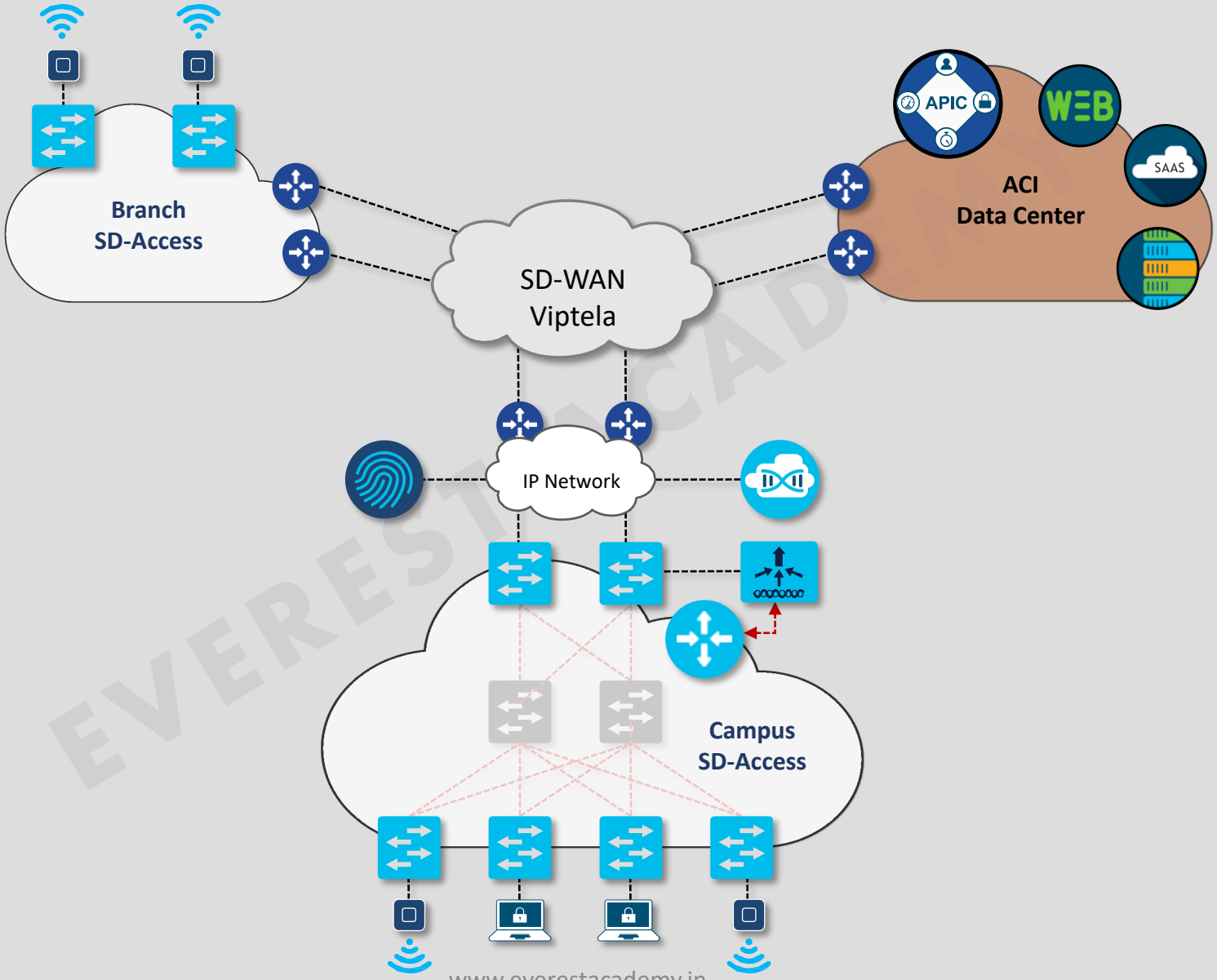
PATH TRACE

Use Path Trace app, to run traces between two end points!

Software-Defined Networking (SDN)

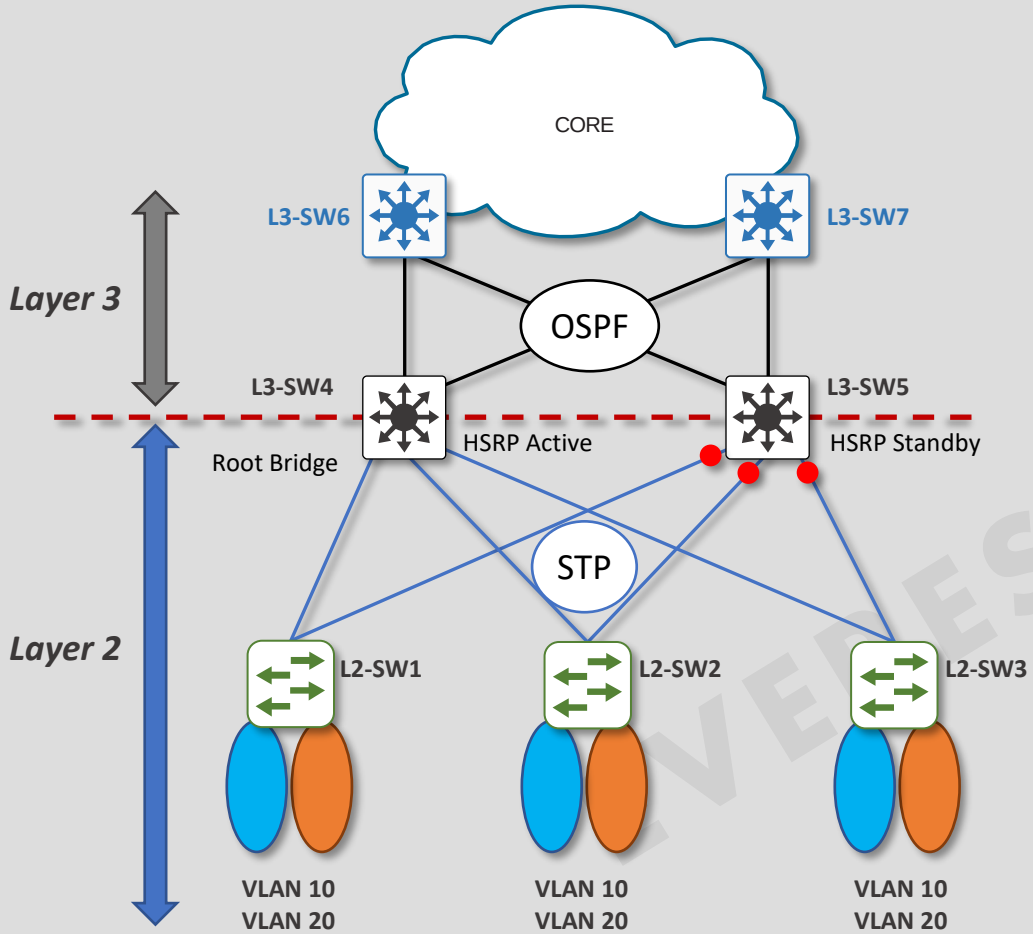


Cisco End-to-End Software-Defined Network

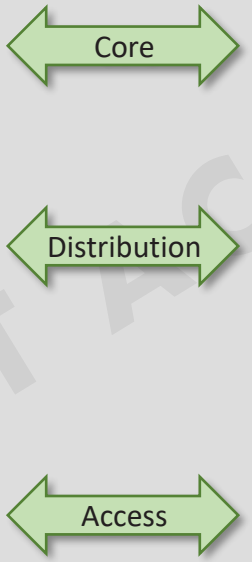


Traditional Campus Network Designs

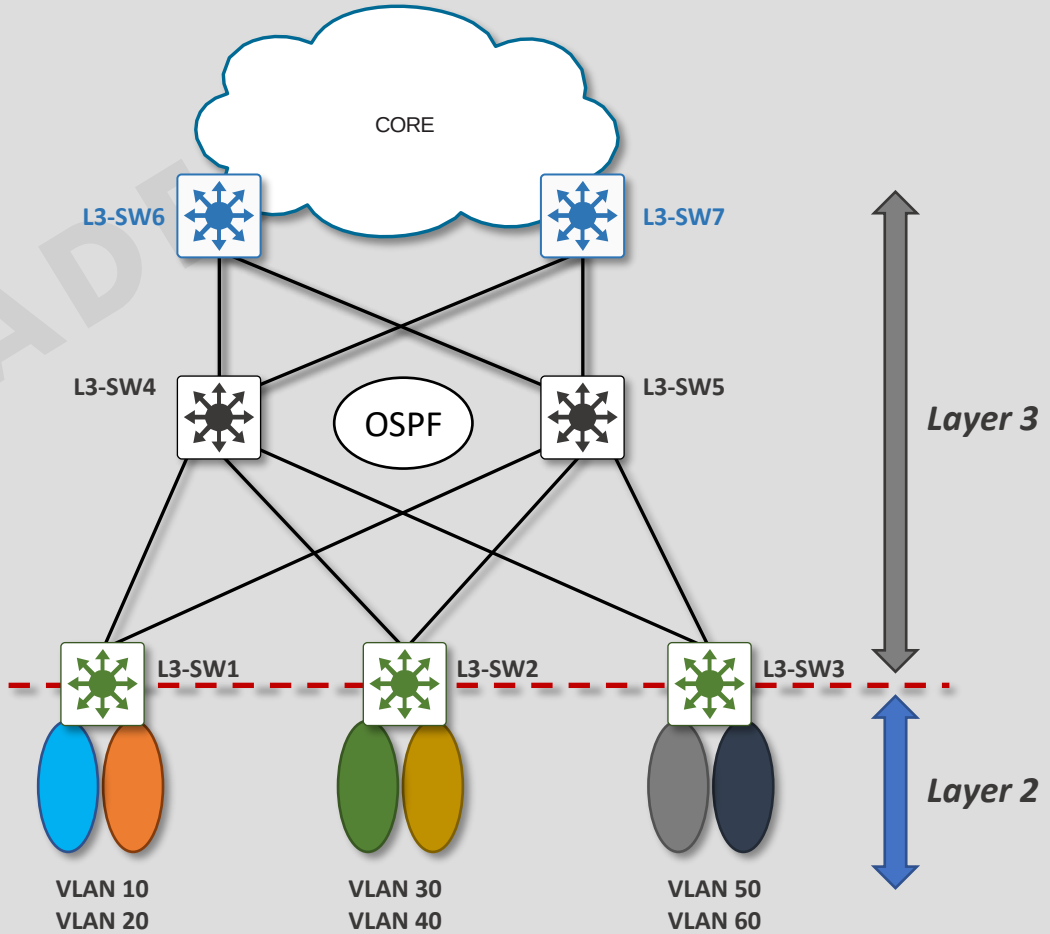
Layer 2 Campus Design



- Layer 2 Loop-Free Topology



Layer 3 Campus Design (Routed Access)



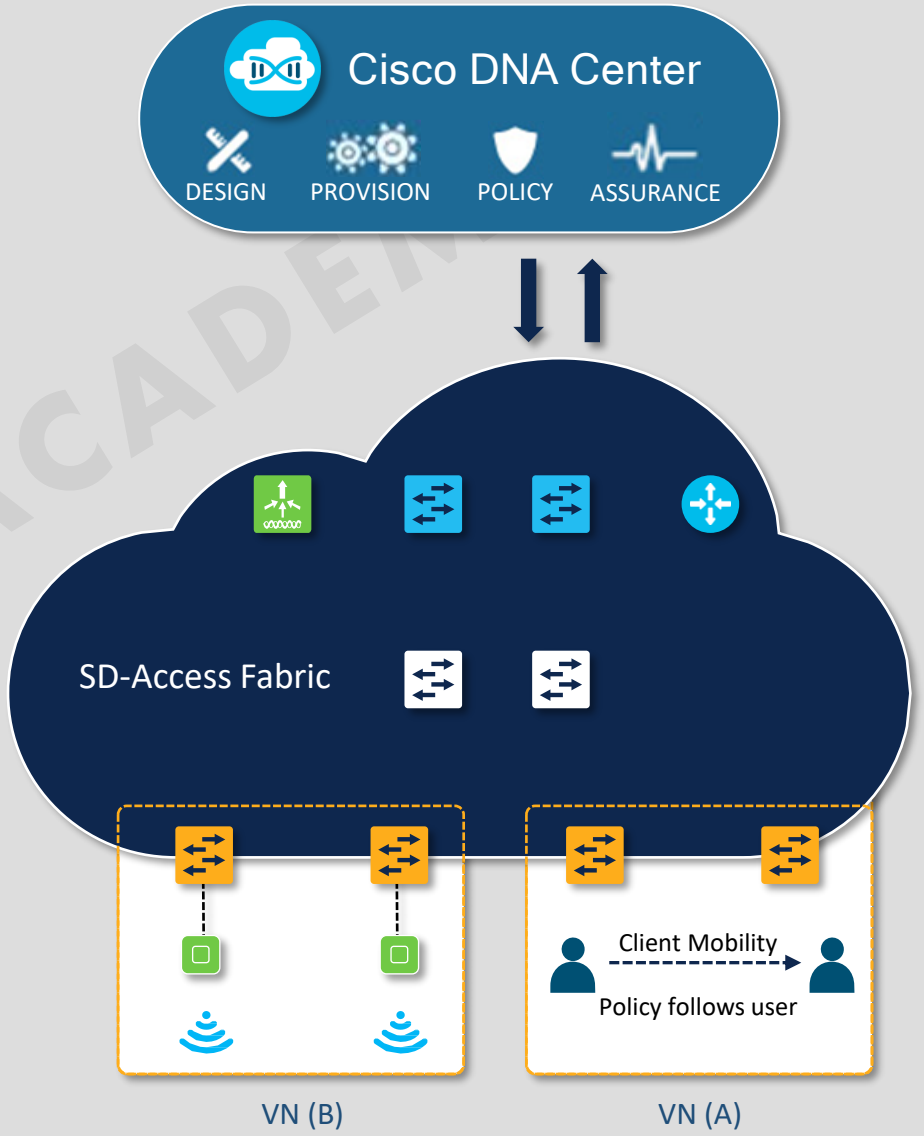
- Layer 3 Loop-Free Topology

The Challenges Of Enterprise Networks Today

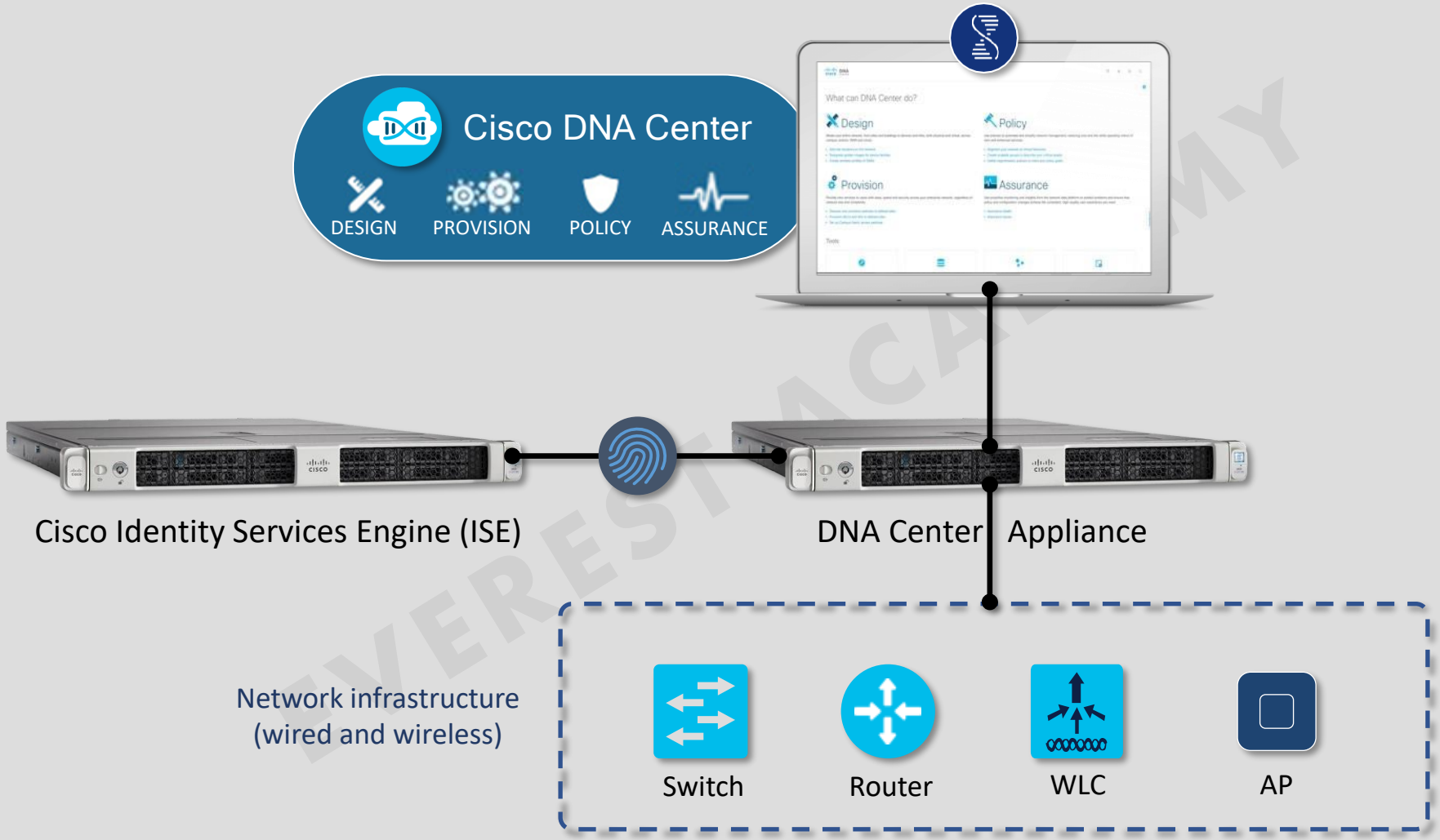
- Applications are moving to the cloud (private and public).
- Mobile devices, BYOD, and guest access.
- High-bandwidth applications.
- Wireless-first connectivity.
- Security and segmentation everywhere.
- Connecting IoT devices to the IT network.

Software-Defined Access (SD-Access)

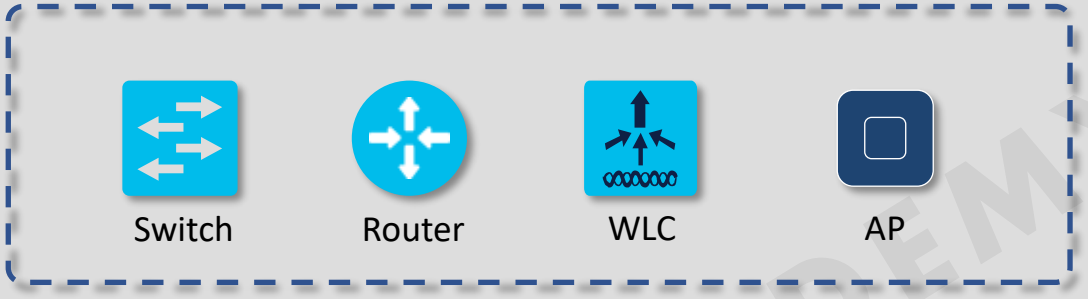
- **Cisco's Software-Defined Access (SD-Access) solution** is a programmable network architecture that provides:
  - Automated network configuration.
  - Automated policy configuration.
  - Identity-based macro segmentation.
  - Identity-based micro segmentation.
  - Dynamic host mobility for wired and wireless network



SD-Access Solution Components

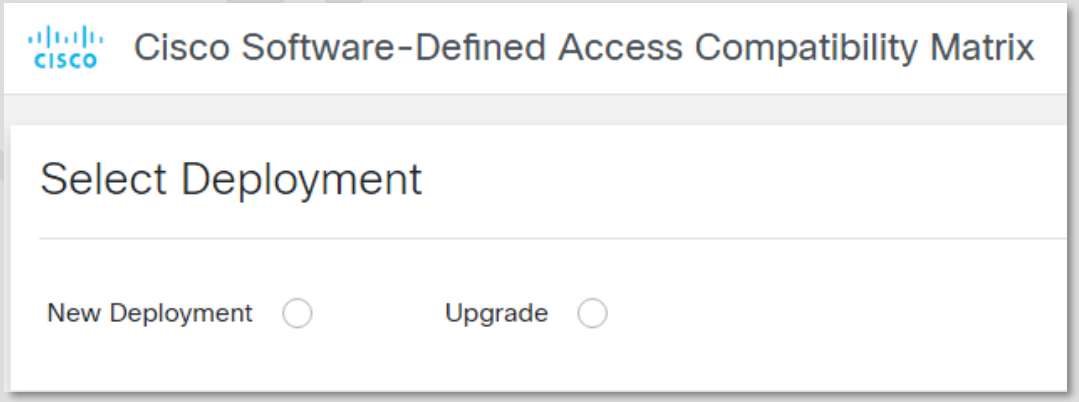


Network infrastructure (wired and wireless)

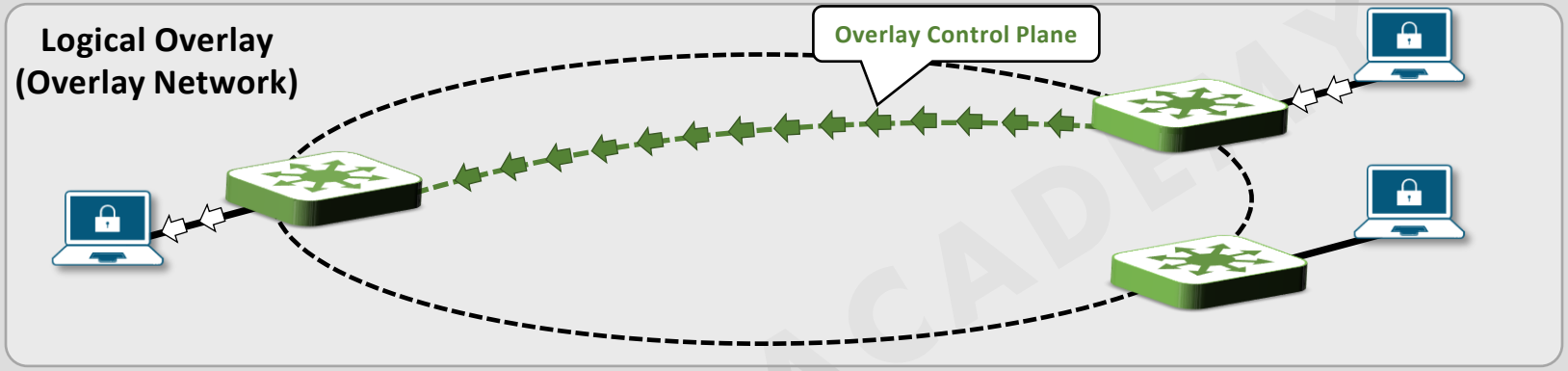


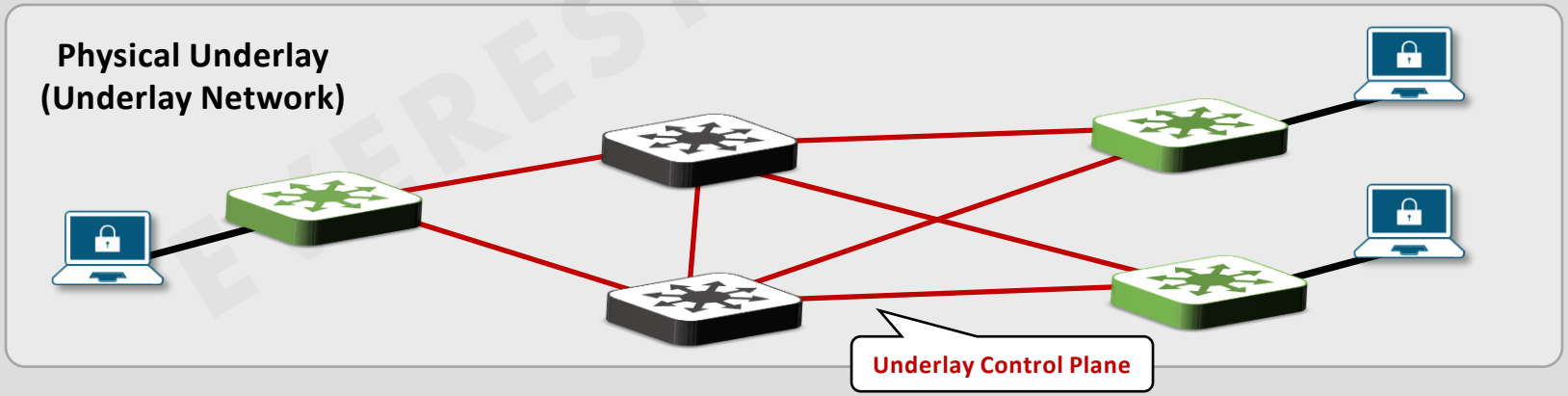
Greenfield Deployment

Brownfield Deployment



Network Virtualization





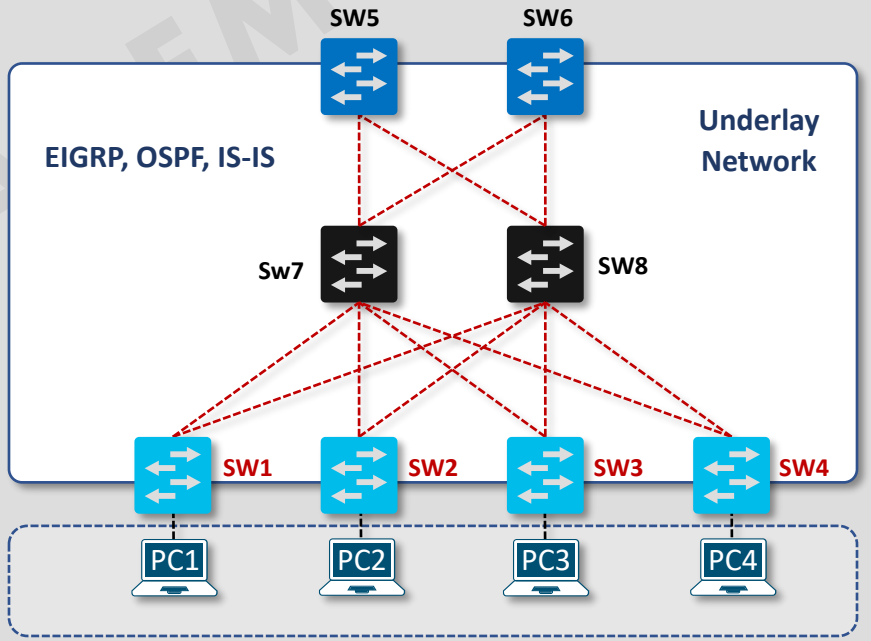
Underlay Network

- **The underlay network** is defined by the physical switches and routers that are used to deploy the SD-Access network

- **All network elements of the underlay** must establish IP connectivity via a routing protocol such as (EIGRP, OSPF, IS-IS,…).

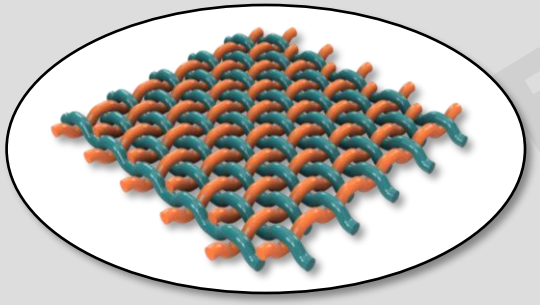
- **Cisco DNA Center** uses IS-IS protocol to build the underlay network during the LAN Automation process.

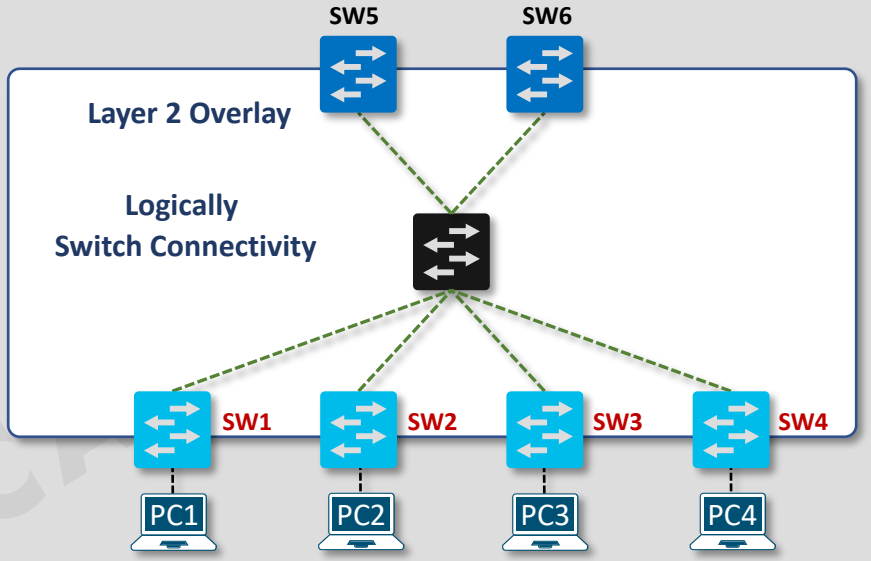
- **In SD-Access** end-user subnets and endpoints are not part of the underlay network, they are part of the automated overlay network.

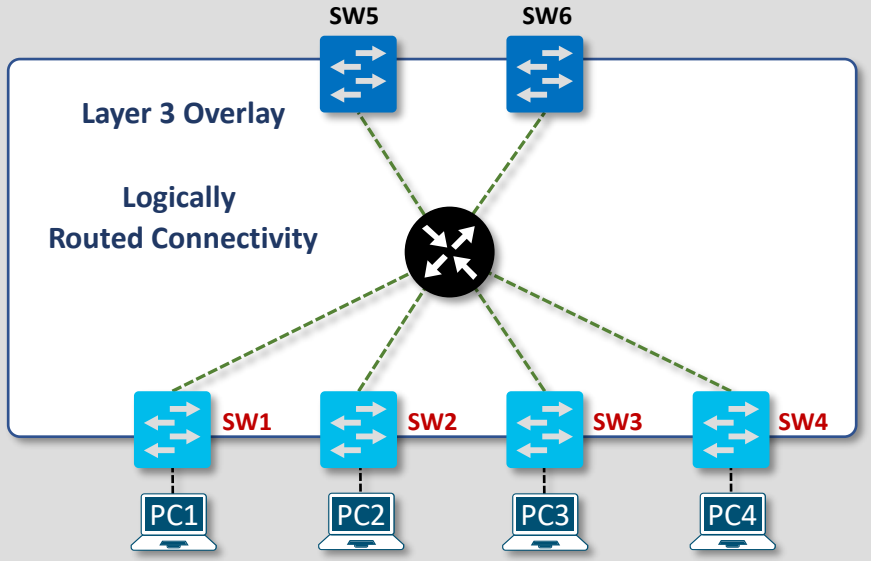


Overlay Network

- **An overlay network** is created on top of the underlay network through virtualization (virtual network).
- **An overlay network** is created through encapsulation, a process which adds additional header(s) to the original packet or frame.
- **Multiple overlay networks** can run across the same underlay network through virtualization.
- **An overlay network** creates a logical topology used to virtually connect devices in a logical full-mesh topology using tunnels.

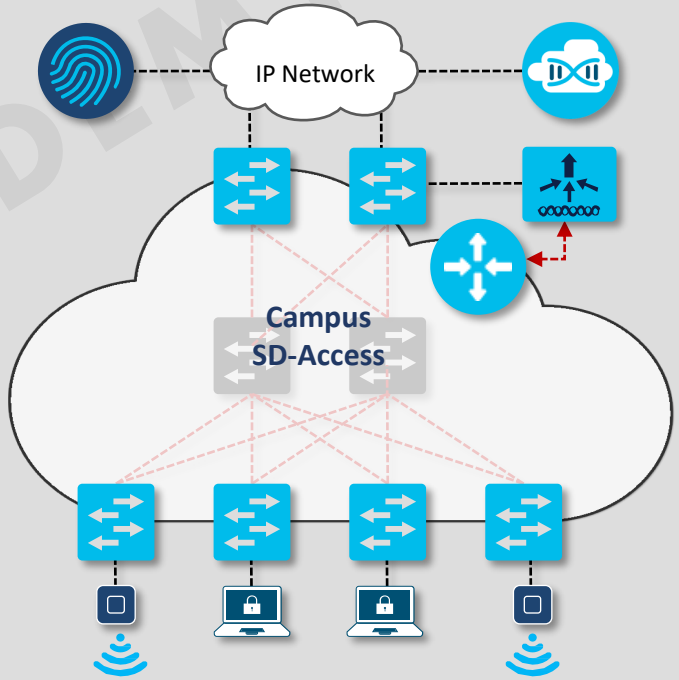






SD-Access Operational Planes

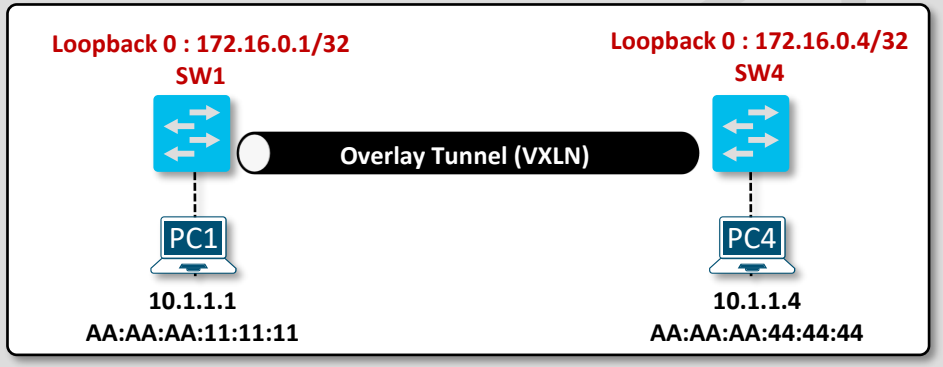
- **Data Plane**: Encapsulation method used for the data packets. It is based on **VXLAN** (Virtual Extensible LAN).
- **Control Plane**: Messaging and communication protocol between infrastructure devices in the fabric. It is based on **LISP** (Locator/ID Separation Protocol).
- **Policy Plane**: Used for security and segmentation. It is based on Cisco TrustSec.
- **Management Plane**: Orchestration, assurance, visibility, and management. It is enabled and powered by Cisco DNA Center.

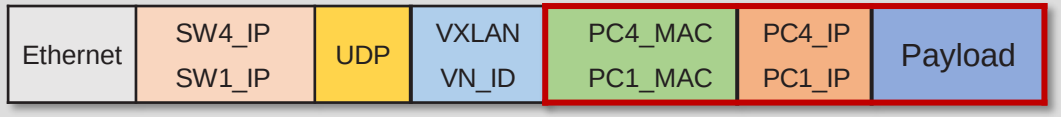


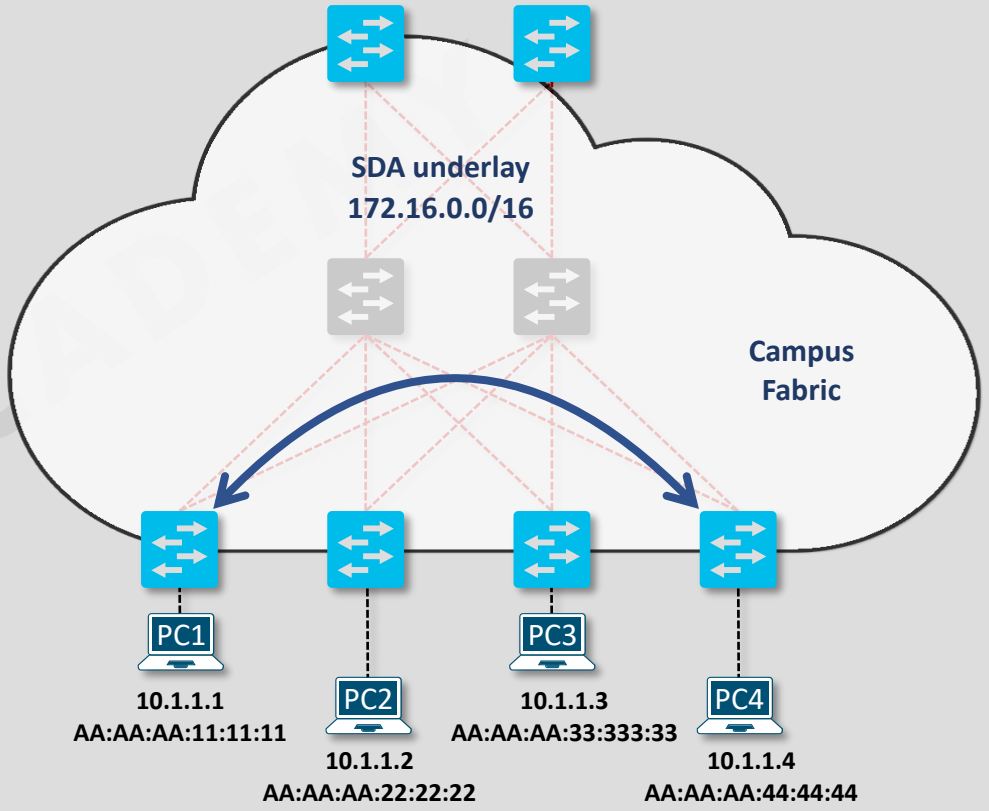
Data Plane – Virtual Extensible LAN (VXLAN)

- **Virtual Extensible LAN (VXLAN)** is an encapsulation technique for data packets. When encapsulation is added to these data packets, a tunnel network is created.

- **VXLAN** is a MAC-in-IP encapsulation method.
- VXLAN network identifier (VNI)
  - 10.0.0.0/8: Entire enterprise
  - 172.16.0.0/16: SDA underlay



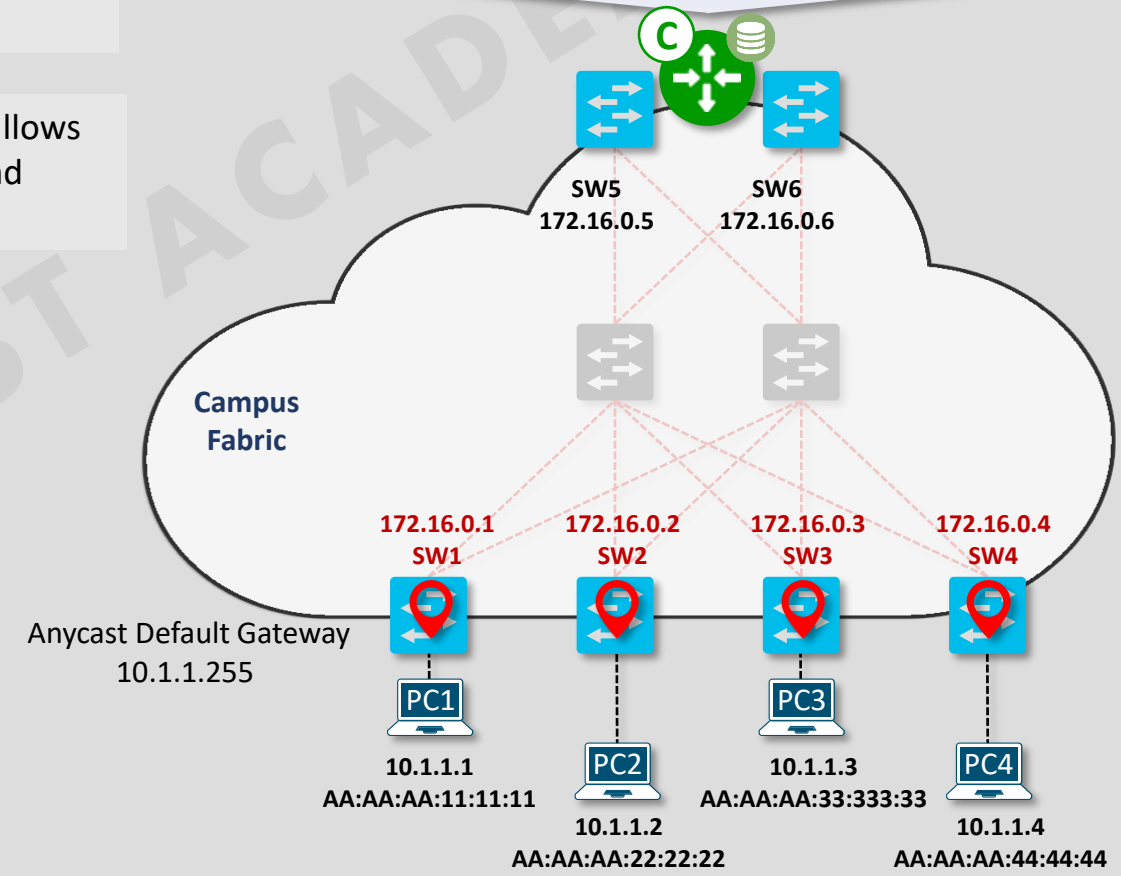




Control Plane – Locator/ID Separation Protocol (LISP)

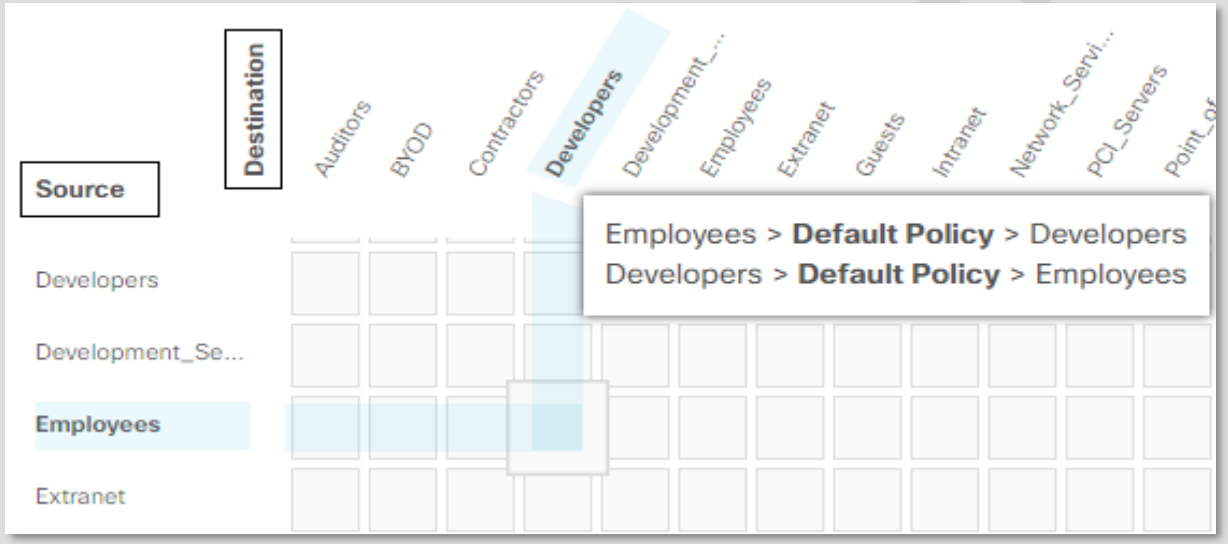
- **Traditional networks :** IP address = identity + location
- **SD-Access uses** Locator/ID Separation Protocol (LISP) that allows the separation of identity and location .
- Endpoint Identifier (EID) => Routing Locator (RLOC)
- The decoupling of the endpoint identity from its location allows addresses in the same IP subnetwork to be available behind multiple Layer 3 gateways in disparate network locations.

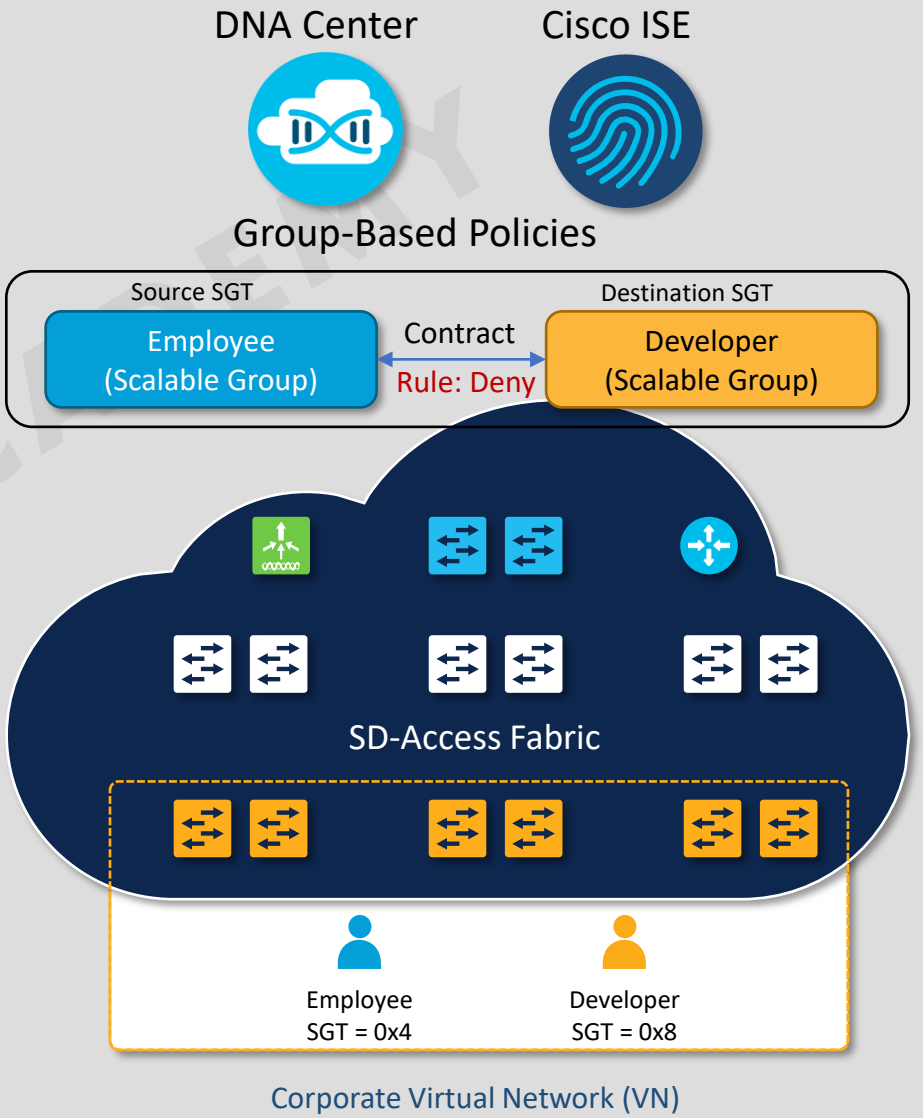
EID (IP) to RLOC	EID (MAC) to RLOC	Address Resolution
10.1.1.1/32 -> 172.16.0.1	AA:AA:AA:11:11:11 -> 172.16.0.1	10.1.1.1 -> AA:AA:AA:11:11:11
10.1.1.2/32 -> 172.16.0.2	AA:AA:AA:22:22:22 -> 172.16.0.2	10.1.1.2 -> AA:AA:AA:22:22:22
10.1.1.3/32 -> 172.16.0.3	AA:AA:AA:33:33:33 -> 172.16.0.3	10.1.1.3 -> AA:AA:AA:33:33:33
10.1.1.4/32 -> 172.16.0.4	AA:AA:AA:44:44:44 -> 172.16.0.4	10.1.1.4 -> AA:AA:AA:44:44:44



Policy Plane – Cisco TrustSec

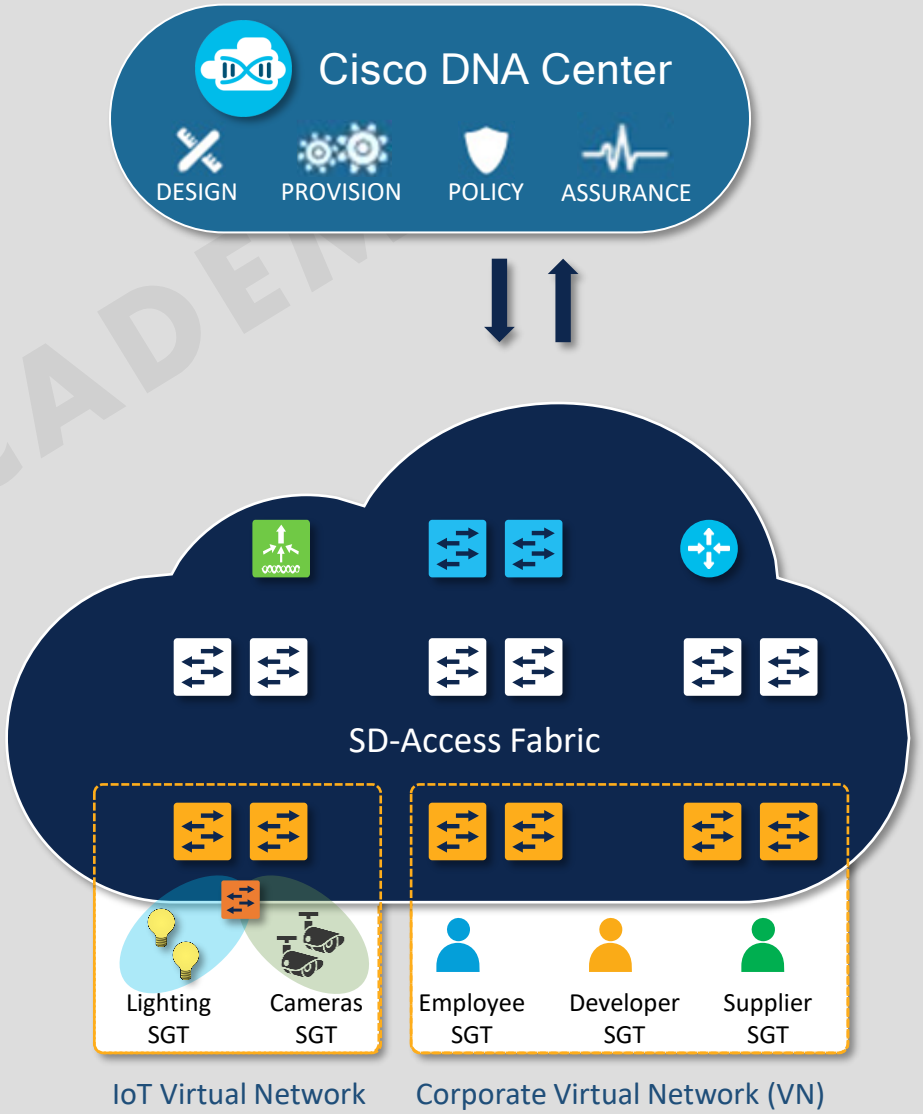
- **Policy plane** is based on Cisco TrustSec (CTS) technology.
- **Cisco TrustSec** provides software defined segmentation and decouples access that is based on IP addresses and VLANs by using logical groupings in a method known as Group-Based Access Control (GBAC).
- **Cisco TrustSec** assigns a scalable group tag (SGT) value to the packet at its ingress point into the network. An access policy elsewhere in the network is then enforced based on this tag information.
- **An SGT** is a form of metadata and is a 16-bit value assigned by ISE when user, device, or application connects to the network.



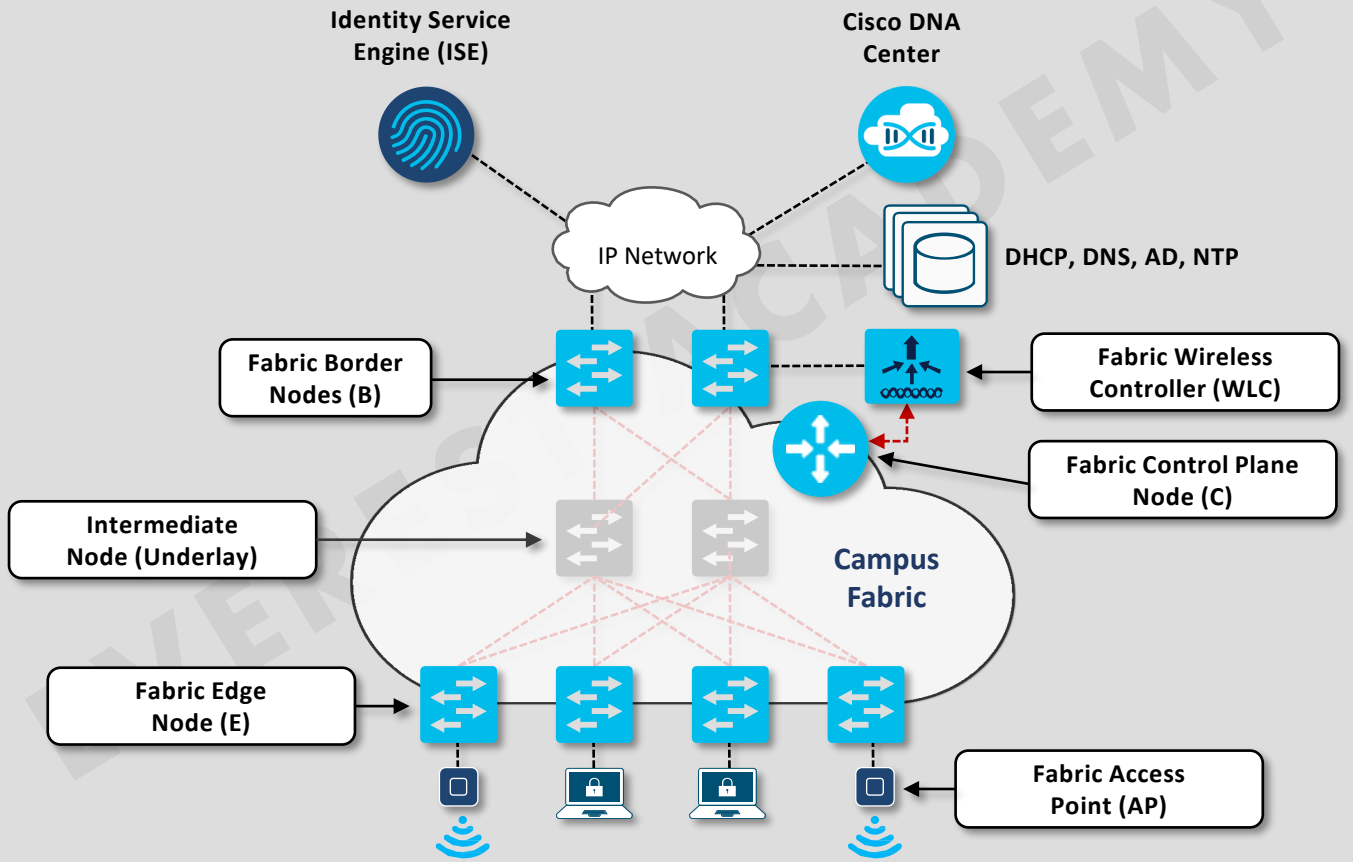


Management Plane – Cisco DNA Center

- **Cisco DNA Center** is an intuitive, centralized management system used to design, provision, and apply policy across the wired and wireless SD-Access network.



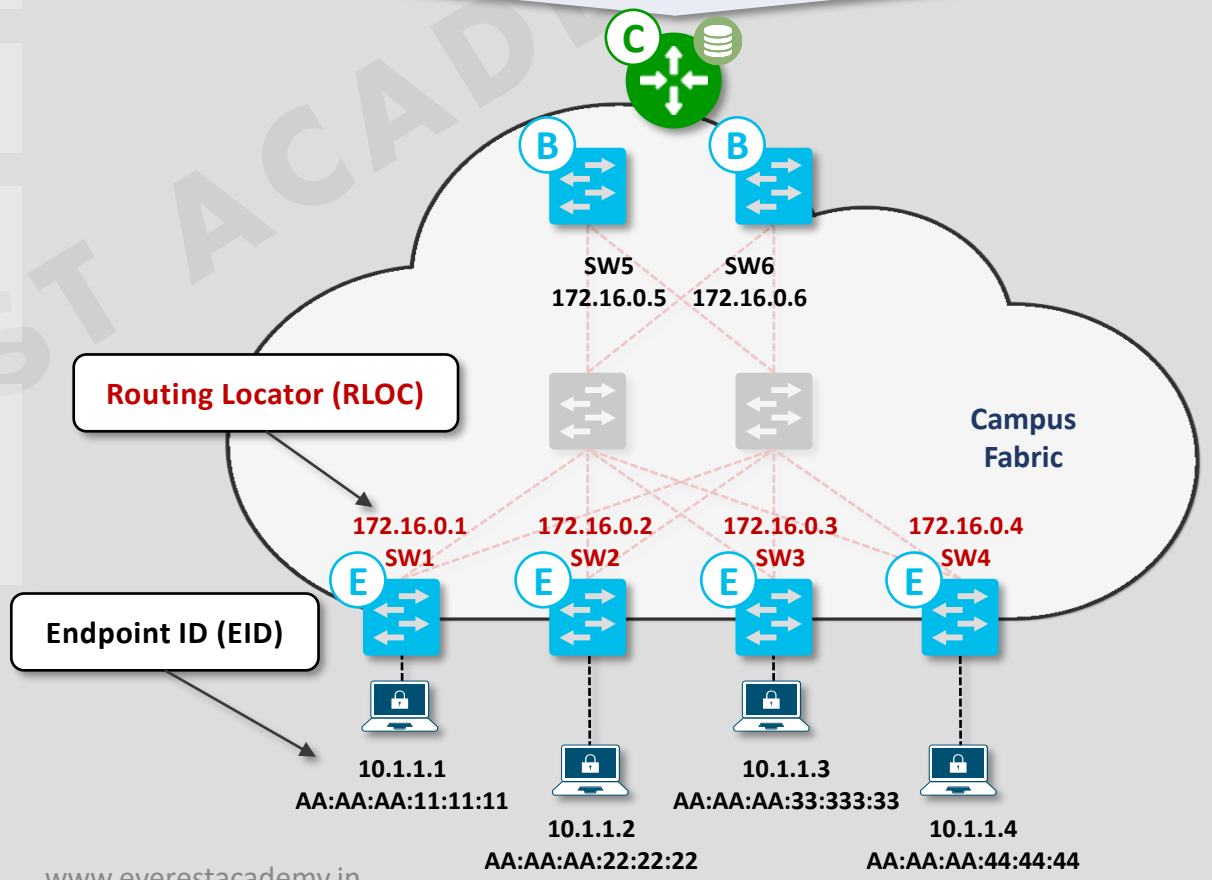
SD-Access Fabric Roles and Terminology



Fabric Control Plane Nodes

- **Control Plane Node:** A host database with mapping of endpoint IDs to a location (loopback of the edge device).
  - It implements the LISP Map Server/Map Resolver (MS/MR) functionality.
  - It receives endpoint ID registrations from edge and border nodes.
  - It also resolves the mapping requests received from border and edge nodes to locate destination endpoint IDs.
- **The control plane node** is the most important node in the Campus Fabric because it is the brains behind the system. For redundancy, there can be multiple control plane nodes in a Cisco SD-Access fabric.

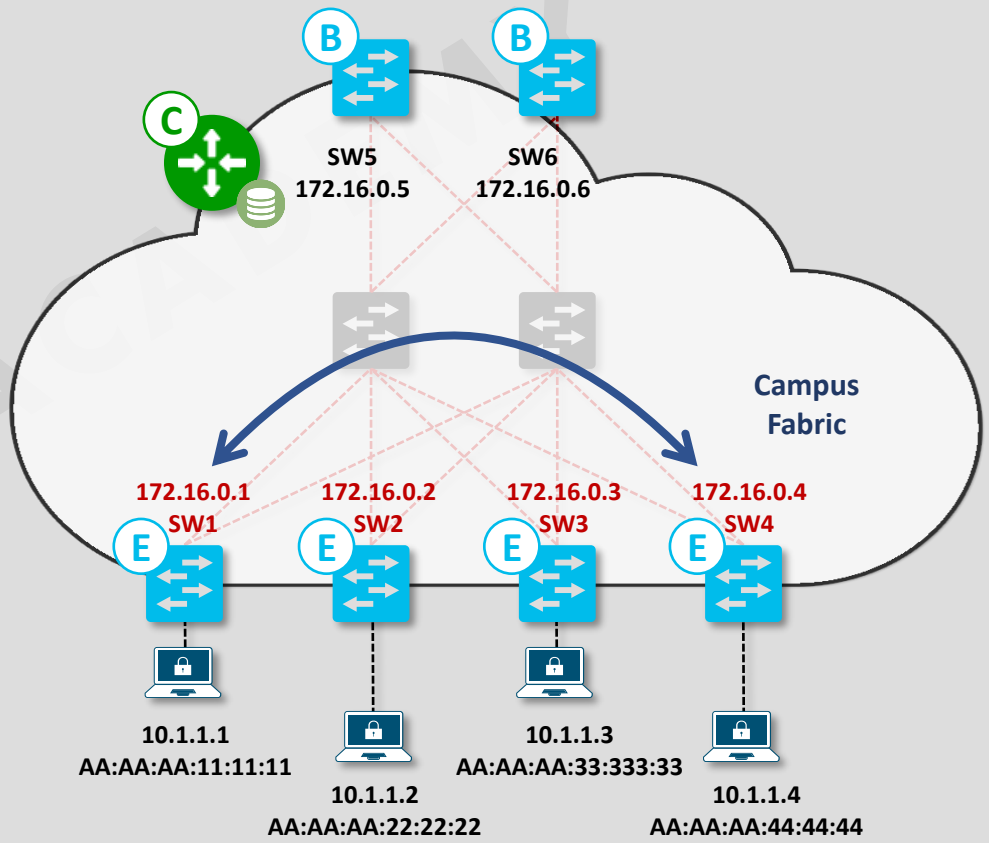
EID (IP) to RLOC	EID (MAC) to RLOC	Address Resolution
10.1.1.1/32 -> 172.16.0.1	AA:AA:AA:11:11:11 -> 172.16.0.1	10.1.1.1 -> AA:AA:AA:11:11:11
10.1.1.2/32 -> 172.16.0.2	AA:AA:AA:22:22:22 -> 172.16.0.2	10.1.1.2 -> AA:AA:AA:22:22:22
10.1.1.3/32 -> 172.16.0.3	AA:AA:AA:33:33:33 -> 172.16.0.3	10.1.1.3 -> AA:AA:AA:33:33:33
10.1.1.4/32 -> 172.16.0.4	AA:AA:AA:44:44:44 -> 172.16.0.4	10.1.1.4 -> AA:AA:AA:44:44:44



Fabric Edge Nodes

- **The SD-Access fabric edge nodes** are the equivalent of access layer switches in Layer 3 Campus Design (Routed Access).
- **The edge node functionality** is based on the Ingress and Egress Tunnel Routers (xTRs) in LISP
- **Edge nodes implement the following functions:**

1. *Endpoint registration.*
2. *AAA Authenticator.*
3. *Mapping of user to virtual network.*
4. *VXLAN encapsulation/de-encapsulation.*
5. *Anycast Layer 3 gateway.*

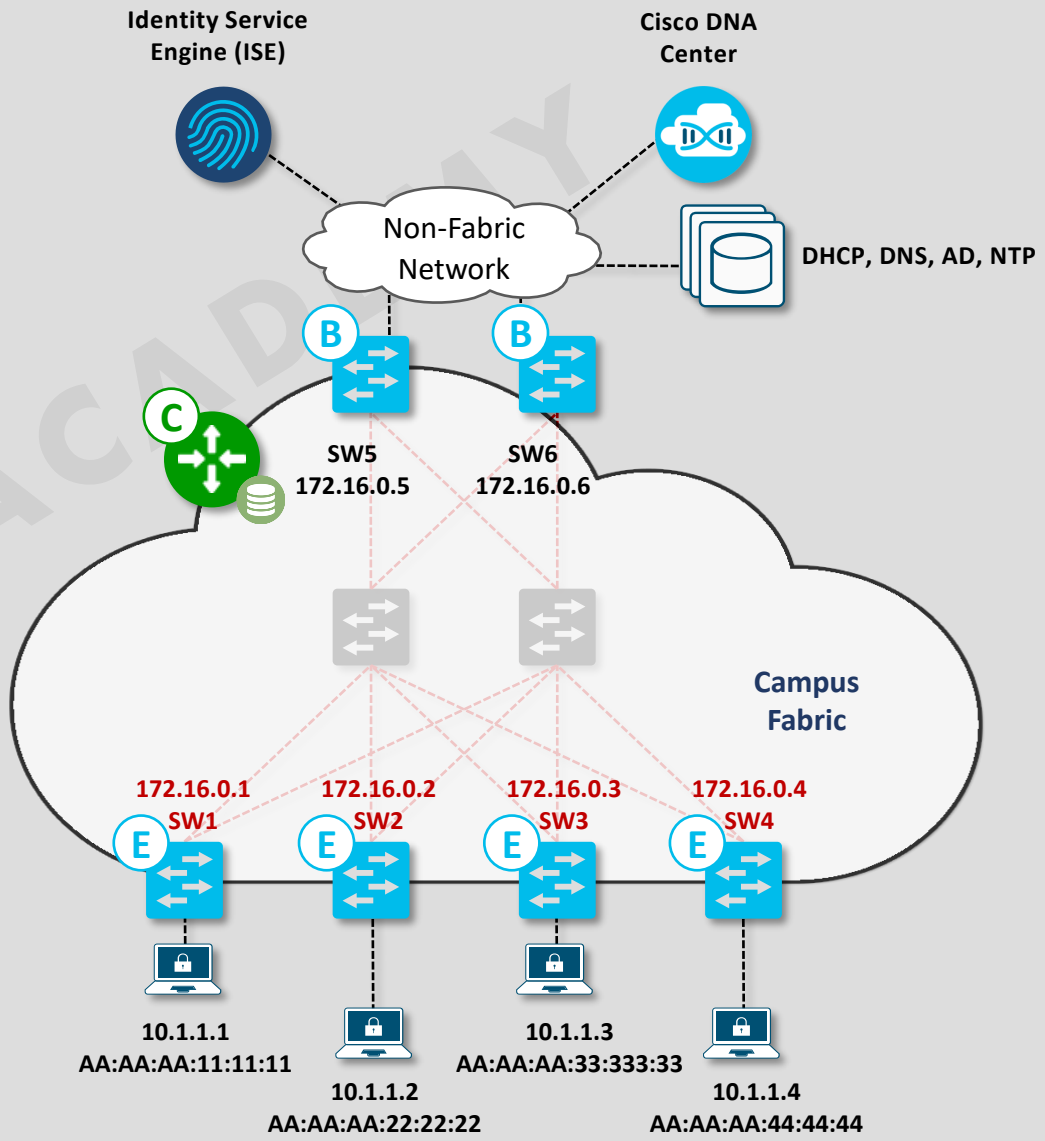


Fabric Border Nodes

- **Fabric Border Node:** A device that speaks fabric and non-fabric to connect the external Layer 3 networks to the Cisco SD-Access fabric.

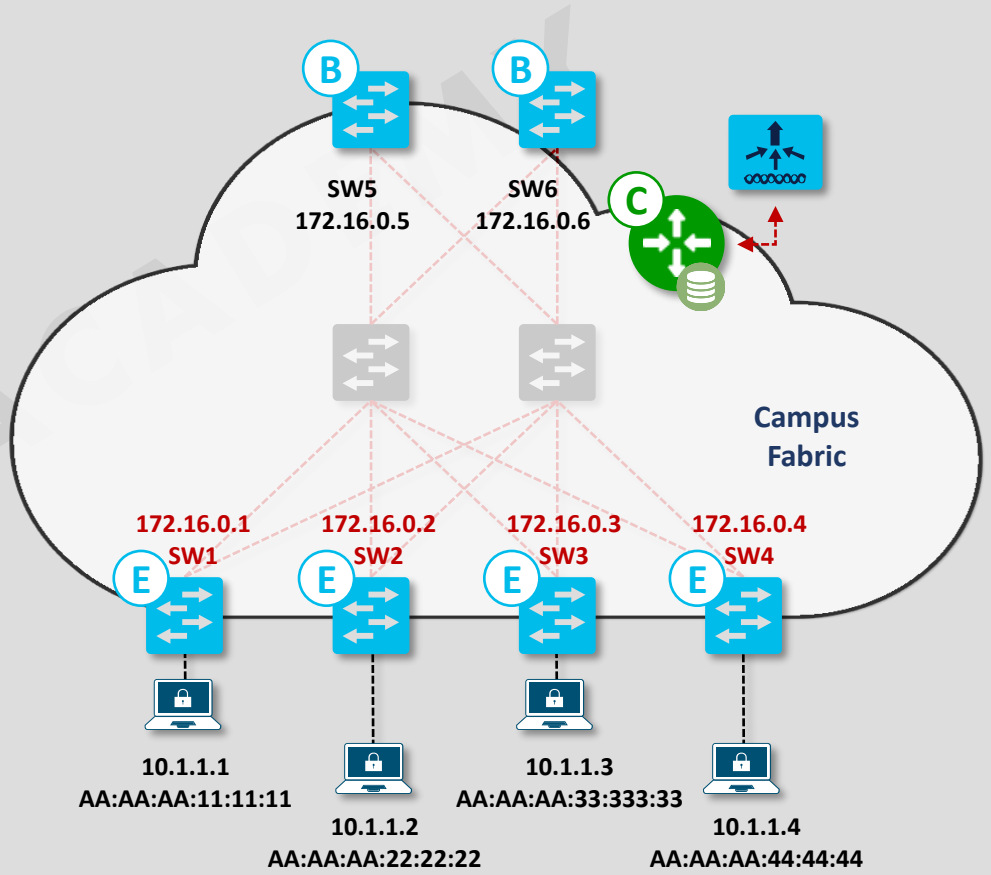
- **Border nodes implement the following functions:**

1. *Fabric site exit point.*
2. *Advertisement of EID subnets outside the fabric.*
3. *Network virtualization extension to the external world.*
4. *VXLAN encapsulation/de-encapsulation.*
5. *Policy mapping.*



Fabric WLC

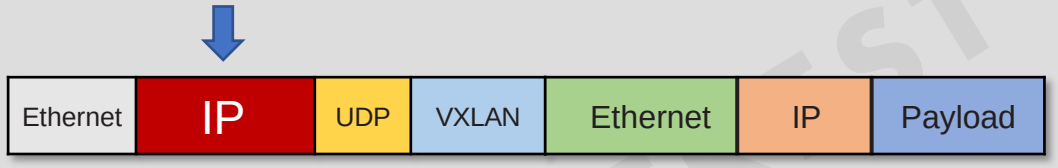
- **Fabric WLCs** provide AP image and configuration management, client session management, mobility services, registering MAC addresses of wireless clients into the host tracking database (HTDB) of the control plane nodes and supplying fabric edge node RLOC-association updates to the HTDB during client roam events.

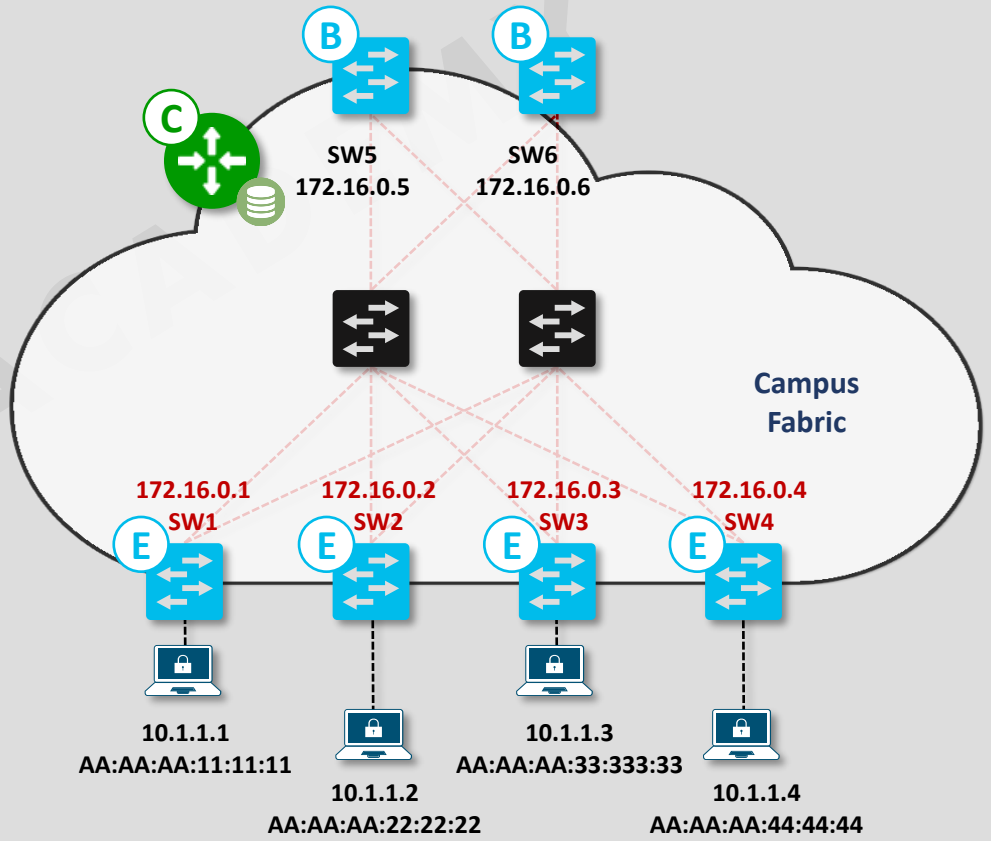


Fabric Intermediate Nodes

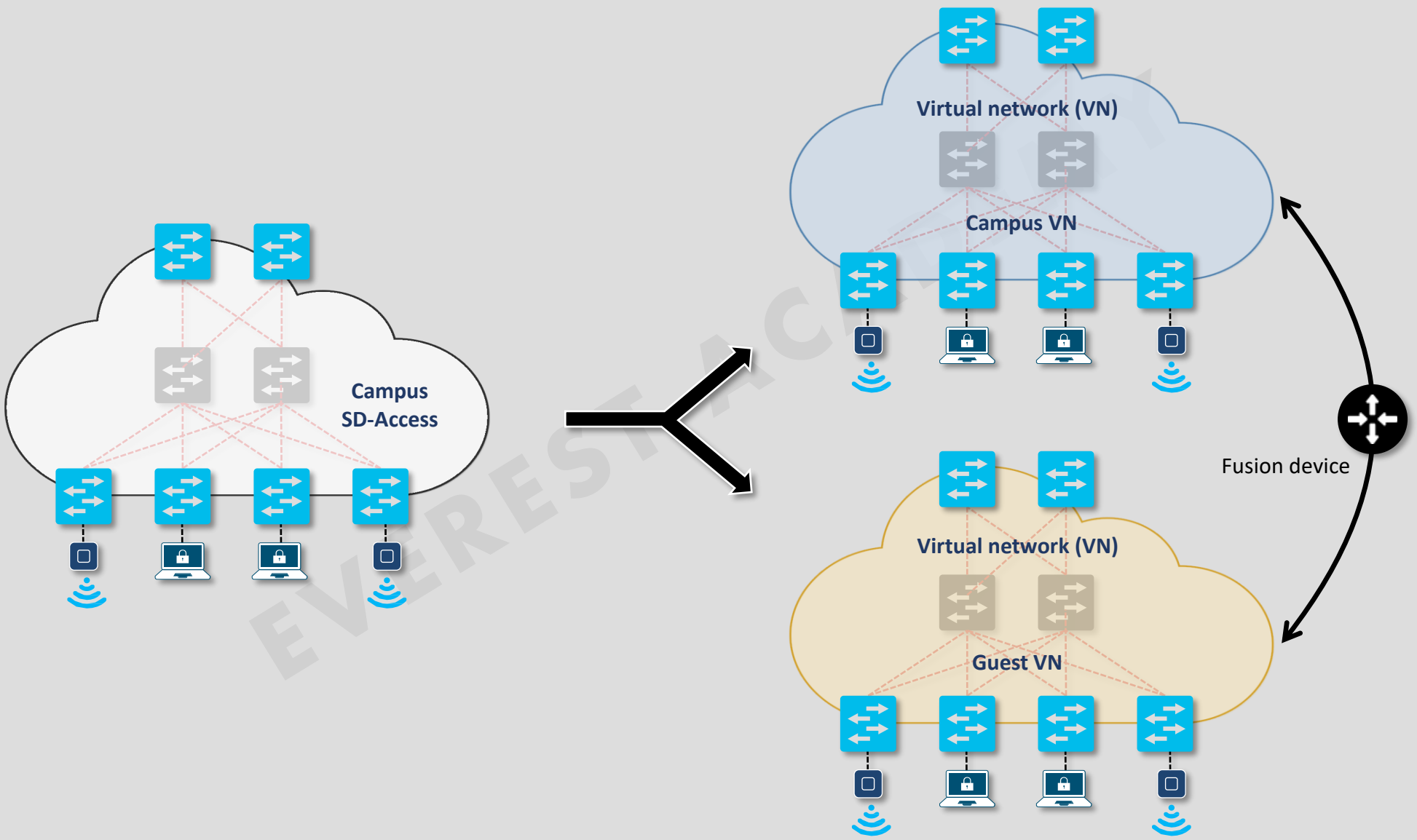
- **Fabric intermediate nodes** provide the Layer-3 underlay transport service to fabric traffic. These nodes are pure layer-3 forwarders that connect the Fabric Edge and Fabric Border nodes.

- **Fabric Intermediate nodes** do not have a requirement for VXLAN encapsulation/de-encapsulation, LISP control plane messaging support, or SGT awareness.

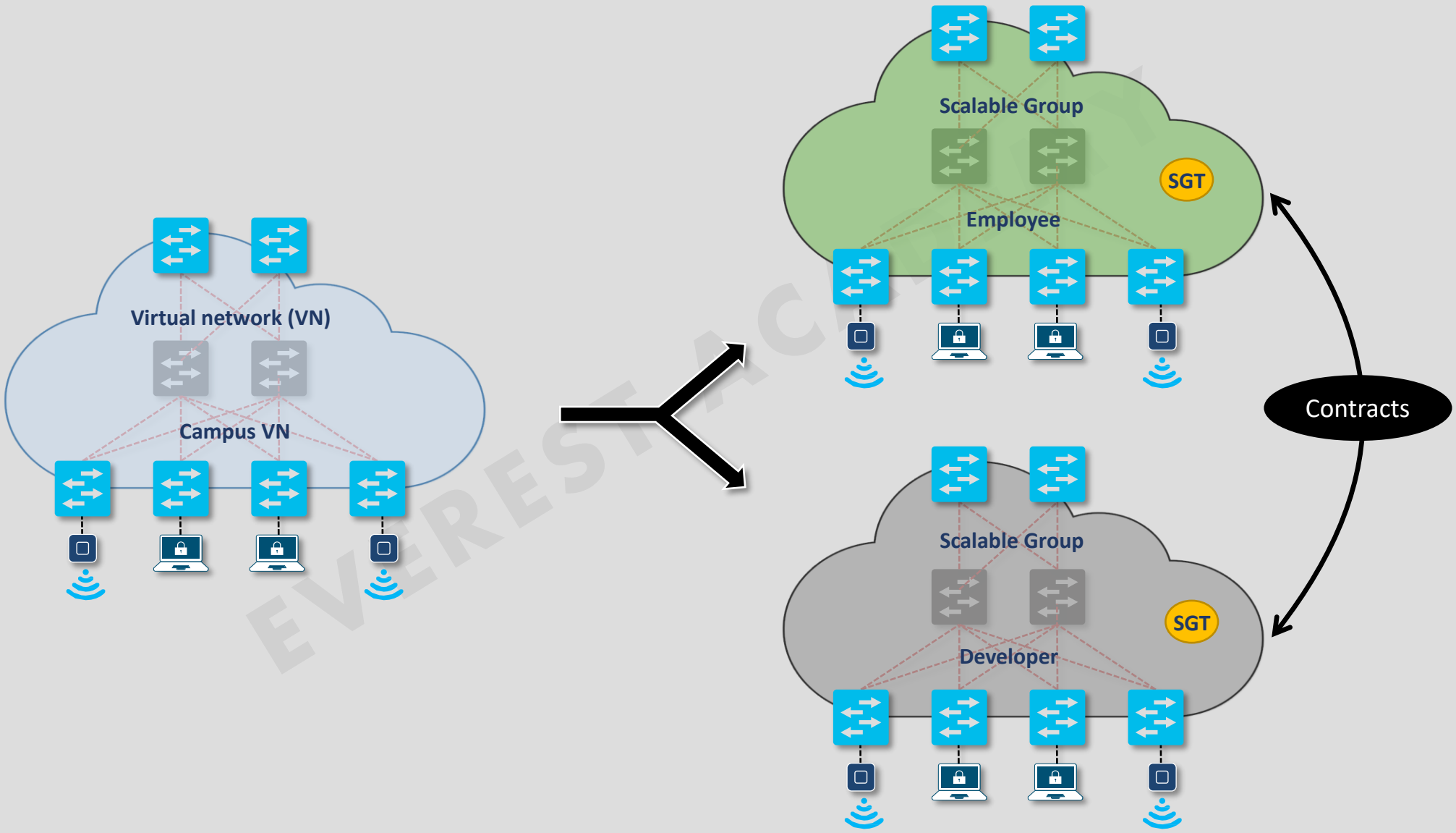




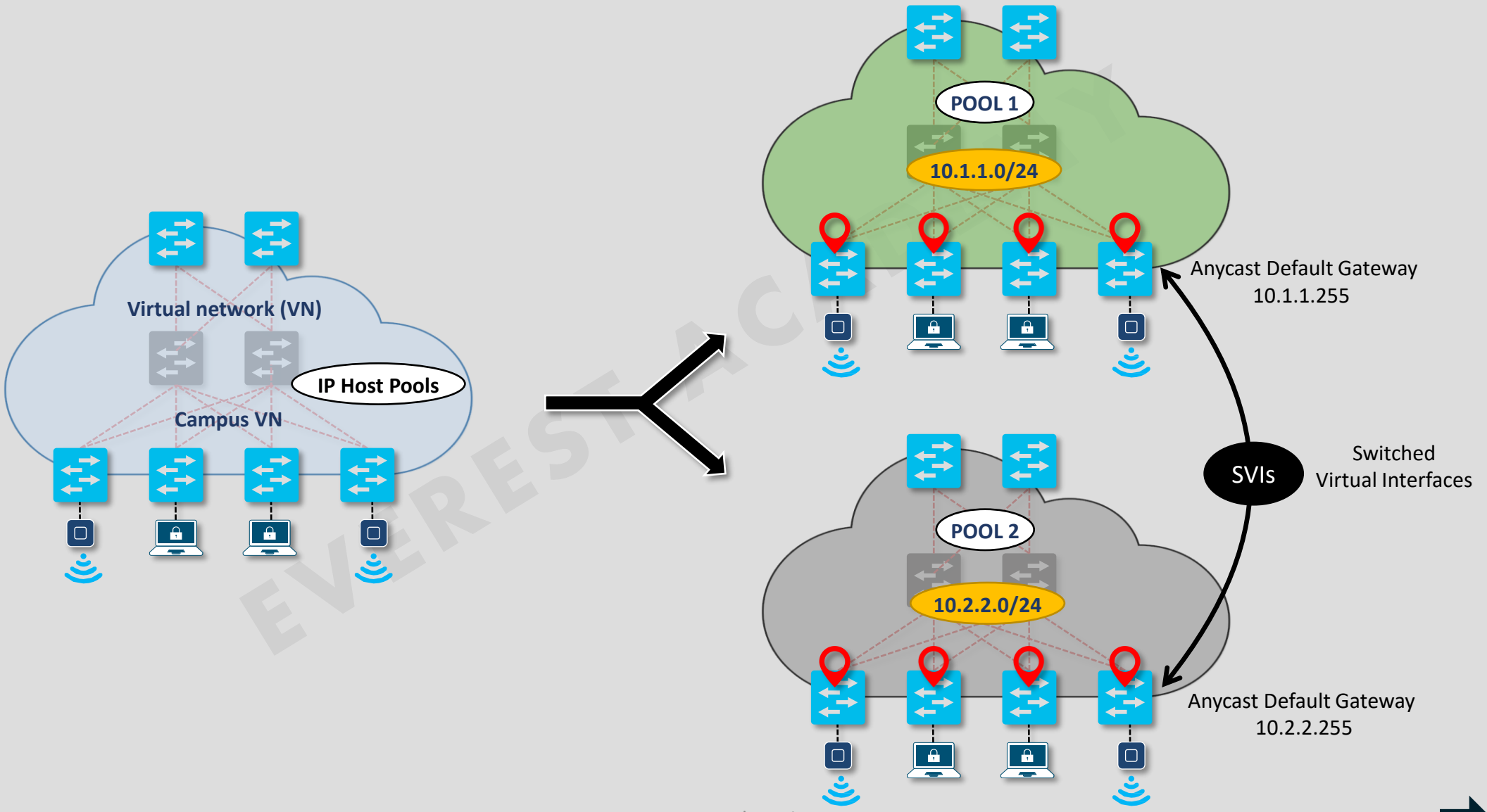
Macro Segmentation



Micro Segmentation



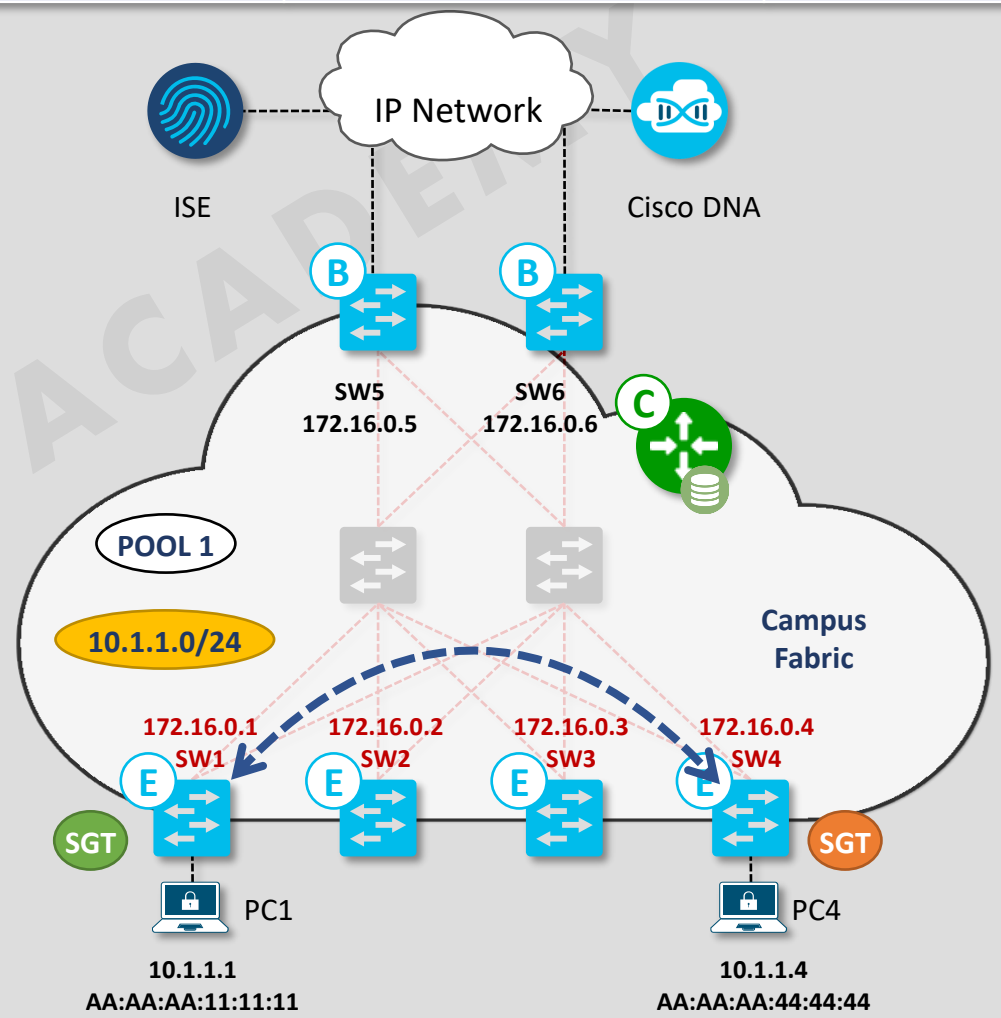
IP Host Pools (Subnets)



An Example of Traffic Flow

- PC1 connects to SW1 through a wired connection. SW1 performs dynamic authentication, the user is assigned an IP address from a host pool (POOL 1), and a Scalable Group Tag (SGT) is pushed by ISE.
- SW1 then registers the client's IP address MAC address and its location (SW1 loopback) with the control plane nodes using LISP.
- PC1 initiates traffic to PC4 on Sw4. SW1 does a mapping lookup with the control plane node for the location of PC4. The control plane node provides the location (e.g., loopback of SW4, 172.16.0.4).
- SW1 encapsulates the user traffic in VXLAN, forming a dynamic VXLAN tunnel with SW4. The encapsulated packet uses the underlay to route the traffic.
- Sw4 receives the VXLAN packet, decapsulates the packet to see the SGT, and forwards the original packet to PC4 if the SGT policy is permit.

IP to RLOC	MAC to RLOC	Address Resolution
10.1.1.4/32 -> 172.16.0.4	AA:AA:AA:44:44:44 -> 172.16.0.4	10.1.1.1 -> AA:AA:AA:11:11:11



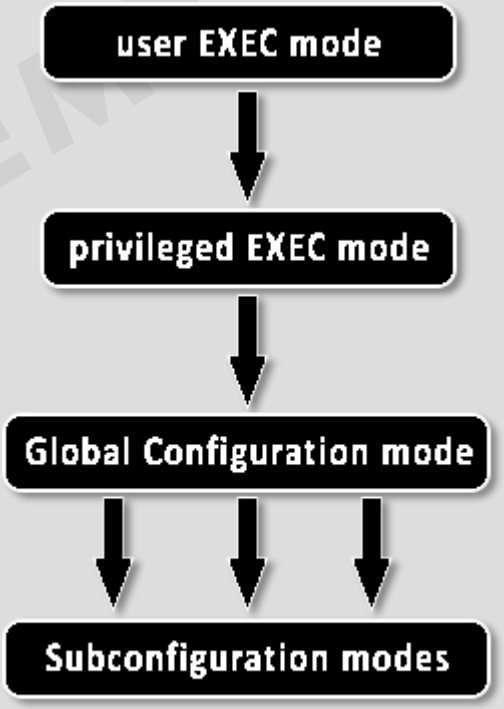
Configuration Management

- **Manual per-device configuration process (IOS CLI).**
  - **copy running-config startup-config**
  - **Configuration Drift** is the phenomenon where running configuration of a device drifts away from the intended configuration over time due to manual changes.
- **Centralized configuration files and version control.**

git GitHub

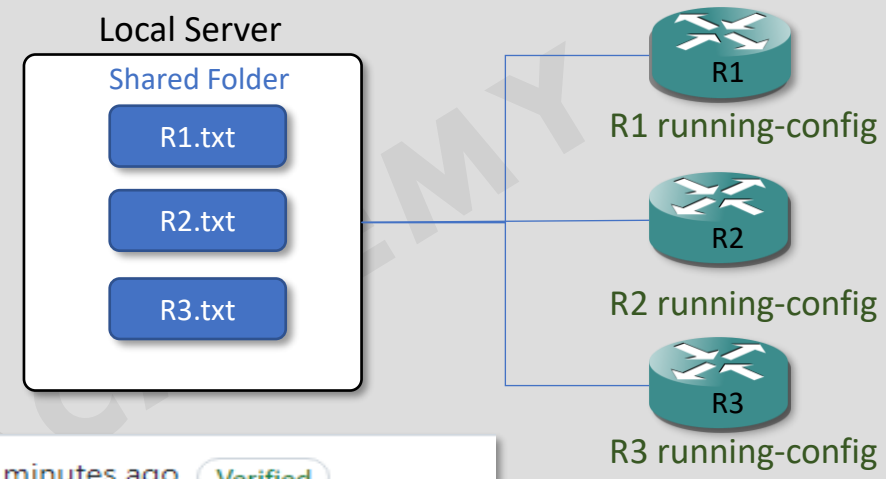
- **Automated configuration management tools.**

ANSIBLE CHEF puppet



Centralized Configuration Files and Version Control

- In this solution the configuration files is stored in a central location in a shared folder.
- The configuration files exist on each device.
- The configuration files exist on a centralized server.
- Version control software (Git).
- Software as a service (SaaS) site GitHub (www.github.com).



everestacademy committed 6 minutes ago Verified

Showing 1 changed file with 1 addition and 1 deletion.

2 R1

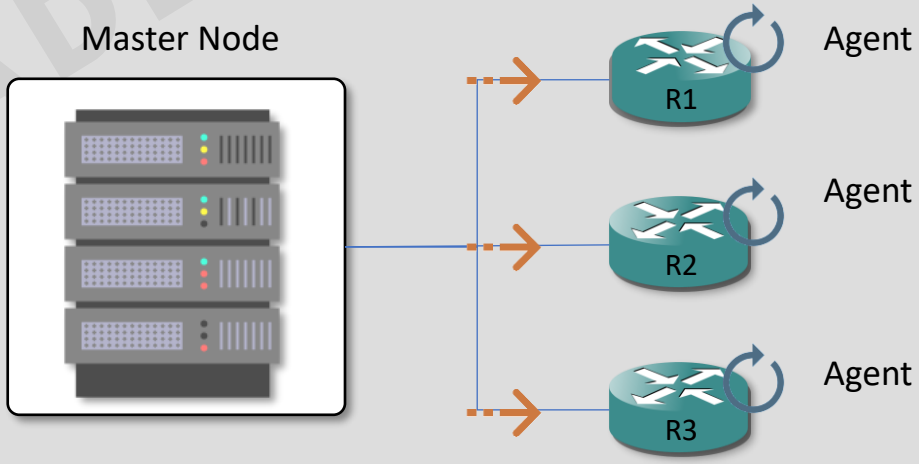
```
...  ...  @@ -1,6 +1,6 @@
1    1    enable
2    2    conf t
3    3    interface fastethernet 0/0
4         - ip address 192.168.1.1 255.255.255.0
     4    + ip address 192.168.1.2 255.255.255.0
5    5    no shutdown
6    6    end
```

Configuration Management Tools

- **Pull Model:**
  - The network nodes run an software that periodically checks from the master node if/when there are any updates to be pulled and applied.

puppet

CHEF



- **Push Model:**
  - Here, it is the master node which takes the responsibility to contact the network nodes to send updates as and when they occur.

ANSIBLE

Configuration Provisioning

- **Configuration provisioning** refers to how to provision or deploy changes to the configuration once made by changing files in the configuration management system.

- Implementing configuration changes in one device after someone has edited the device's centralized configuration file.
- The ability to choose which subset of devices to configure: based on attributes and logic.
- The ability to determine if each change was accepted or rejected.
- For each change, the ability to revert to the original configuration if even one configuration command is rejected on a device
- The ability to validate the change now (without actually making the change) to determine whether the change will work or not when attempted.
- The ability to check the configuration after the process completes.
- The ability to use logic to choose whether to save the running-config to startup-config or not.
- The ability to represent configuration files as templates and variables so that devices with similar roles can use the same template but with different values
- The ability to store the logic steps in a file, scheduled to execute, so that the changes can be implemented by the automation tool without the engineer being present

Configuration Templates and Variables

Hosts (Inventory) File

```
[Routers]
R1 ansible_host=192.168.122.101
R2 ansible_host=192.168.122.102
```

R1 Variables File

```
name: Router1
ip: 192.168.1.1
mask: 255.255.255.0
pool_name: POOL1
network: 192.168.1.0
access_list_name: ACL
wild_card_mask: 0.0.0.255
next_hop_address: 192.168.122.1
```

R2 Variables File

```
name: Router2
ip: 192.168.2.1
mask: 255.255.255.0
pool_name: POOL2
network: 192.168.2.0
access_list_name: ACL
wild_card_mask: 0.0.0.255
next_hop_address: 192.168.122.1
```

Routers Variables File

```
ansible_ssh_user: "admin"
ansible_ssh_pass: "Cisco123"
ansible_network_os: "ios"
ansible_connection: "network_cli"
dns_server: 8.8.8.8
```

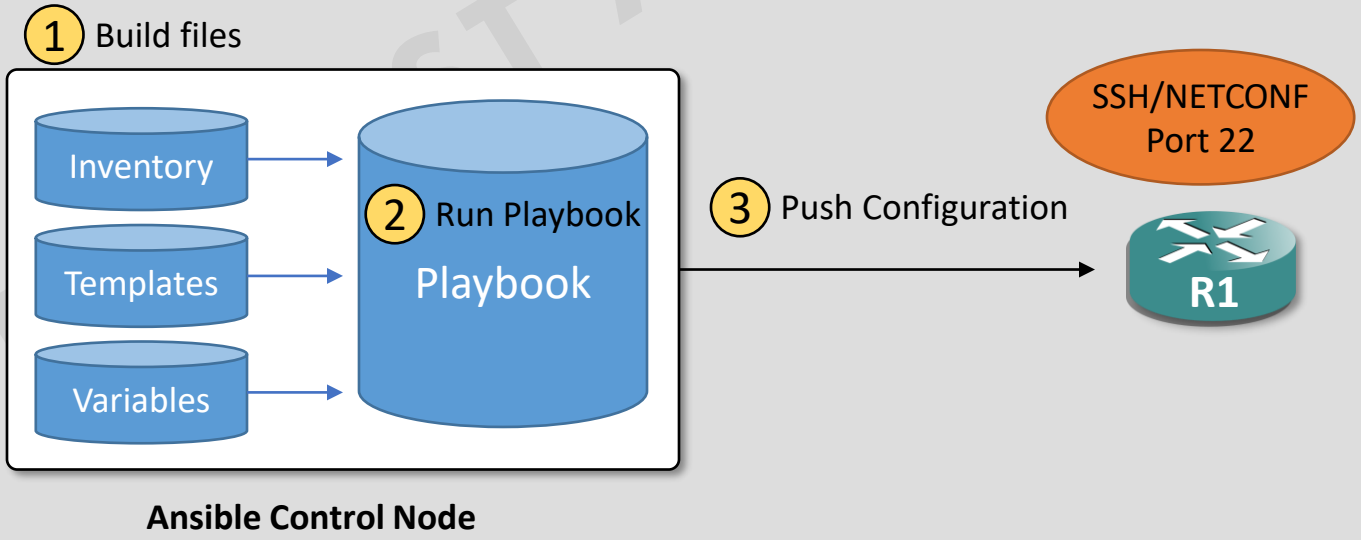
Jinja2 Template with Variables

```
hostname {{name}}
interface Fastethernet0/1
  ip address {{ ip }} {{ mask }}
  ip nat inside
  no shutdown
interface Fastethernet0/0
  ip nat outside
ip dhcp pool {{ pool_name }}
  network {{ network }} {{ mask}}
  default-router {{ ip }}
  dns-server {{ dns_server }}
ip access-list standard {{ access_list_name }}
  permit {{ network }} {{ wild_card_mask }}
ip nat inside source list {{ access_list_name }} interface Fastethernet 0/0 overload
ip route 0.0.0.0 0.0.0.0 {{ next_hop_address }}
```

Ansible (Agentless)

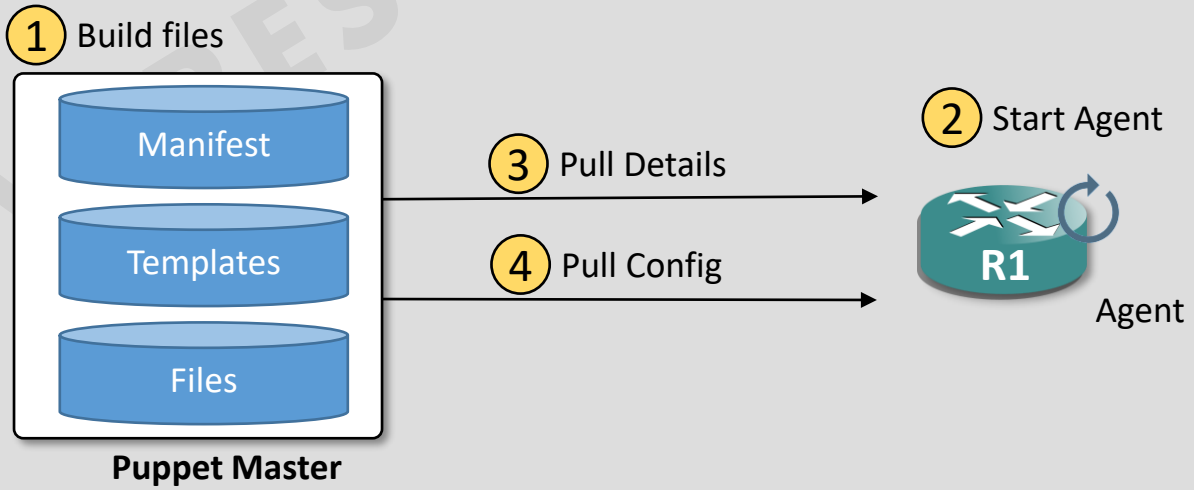
- **Ansible** is an open source IT configuration management (CM) and automation platform, provided by Red Hat.

  - **Playbooks:** These files provide actions and logic about what Ansible should do.
  - **Inventory:** These files provide device hostnames along with information about each device.
  - **Templates:** Using Jinja2 language, the templates represent a device's configuration but with variables.
  - **Variables:** Using YAML, a file can list variables that Ansible will substitute into templates.



Puppet (Agent Based)

- **Puppet** is a Configuration Management tool that is used for deploying, configuring and managing devices, It is produced by Puppet, Inc.
- **Puppet** usually follows client-server architecture. The client is known as an agent and the server is known as the master.
  - **Manifests** are the actual codes for configuring the clients.
  - **Templates** combine code and data to render a final document.
  - **Files** are the static content that can be downloaded by the clients.
  - **Modules** are a collection of manifests, templates, and files.



Chef (Agent Based)

- **Chef is** a configuration management technology used to automate the infrastructure provisioning.

  - **Resource:** The configuration objects whose state is managed by Chef; for instance, a set of configuration commands for a network device.
  - **Recipe:** The Chef logic applied to resources to determine when, how, and whether to act against the resources.
  - **Run-list:** An ordered list of recipes that should be run against a given device.
  - **Cookbooks:** A set of recipes about the same kinds of work, grouped together for easier management and sharing.

Comparing Ansible, Puppet, and Chef

	Ansible	Puppet	Chef
Managed Node Requirements	Agentless	Agentless and Agent Based	Agent Based
Deployment Method	Push Model	Pull Model	Pull Model
Master Server	Linux Only	Linux/Windows	Linux/Windows
File that lists actions	Playbook	Manifest	Recipe/Runlist
Transport Mechanism	SSH/NETCONF	REST	REST
Port Number	22	8140	10002