**IST105 - Introduction to Programming**

**Assignment #9: Network Automation with Cisco DNA Center API**

**Assignments are designed to encourage students to apply the knowledge and skills they have acquired throughout various courses. Technology is constantly evolving, with procedures, protocols, algorithms, and software continually advancing or becoming obsolete. As such, assignments are not simple copy-paste tasks; they require students to integrate concepts and practical examples discussed in class, along with the foundational knowledge they have gained from previous courses. This approach emphasizes critical thinking, problem-solving, and the ability to adapt to the ever-changing landscape of technology.**

**Objectives**

By completing this assignment, you will:

* Authenticate to Cisco DNA Center using REST API
* Retrieve and display network device inventory
* Fetch and analyze interface configurations
* Understand key network automation protocols
* Implement error handling in API interactions
* **Deploy the project on Django framework**
* **Use MongoDB to persist interaction logs**
* **Host Django and MongoDB on separate EC2 instances**

**Background**

**What is Cisco DNA Center?**

Cisco **Digital Network Architecture (DNA) Center** is a **network management and automation platform** that provides:

* **Centralized control** of network devices (routers, switches, wireless)
* **Automation** of network configurations and policies
* **Analytics** for monitoring and troubleshooting
* **Intent-based networking** (networks adapt based on business needs)

**Why use DNA Center?**

* Reduces manual configuration errors
* Speeds up network deployment
* Provides **API access** for automation (REST, NETCONF)

**Why Automate Networks?**

Traditional network management is **manual**, leading to:

* Slow deployments
* Human errors in configurations
* Difficulty in scaling large networks

**Automation solves these problems by:**

* **Reducing manual work** (scripts handle repetitive tasks)
* **Ensuring consistency** (same config applied everywhere)
* **Enabling faster troubleshooting** (automated checks & alerts)

**Examples of Network Automation:**

* Automatically **backing up** device configurations
* **Deploying VLANs** across multiple switches
* **Monitoring network health** with scripts

**NETCONF & YANG for Automation**

**What is NETCONF?**

* **Network Configuration Protocol (NETCONF)**
* Uses **YANG data models** to structure network configurations
* Allows **programmatic control** of network devices

**What is YANG?**

* **Data modeling language** for network devices
* Defines how devices **should be configured**
* Used by **NETCONF** and **RESTCONF**

**Why use NETCONF/YANG?**

* **Standardized** way to configure devices (no vendor-specific CLI)
* **Supports automation** (Python scripts can push configs)

**REST API for Network Automation**

**What is REST API?**

• **Representational State Transfer** (REST) architectural style  
• Uses **HTTP/HTTPS** (GET/POST/PUT/DELETE) for communication  
• Exchanges data in **JSON/XML** formats

**Key Characteristics of DNA Center's REST API:**

• **Token-based authentication** (OAuth2)  
• **CRUD operations** mapped to HTTP methods:

* GET (retrieve), POST (create), PUT (update), DELETE (remove)  
  • **Stateless** – each request contains all necessary information

**Why Use REST API?**

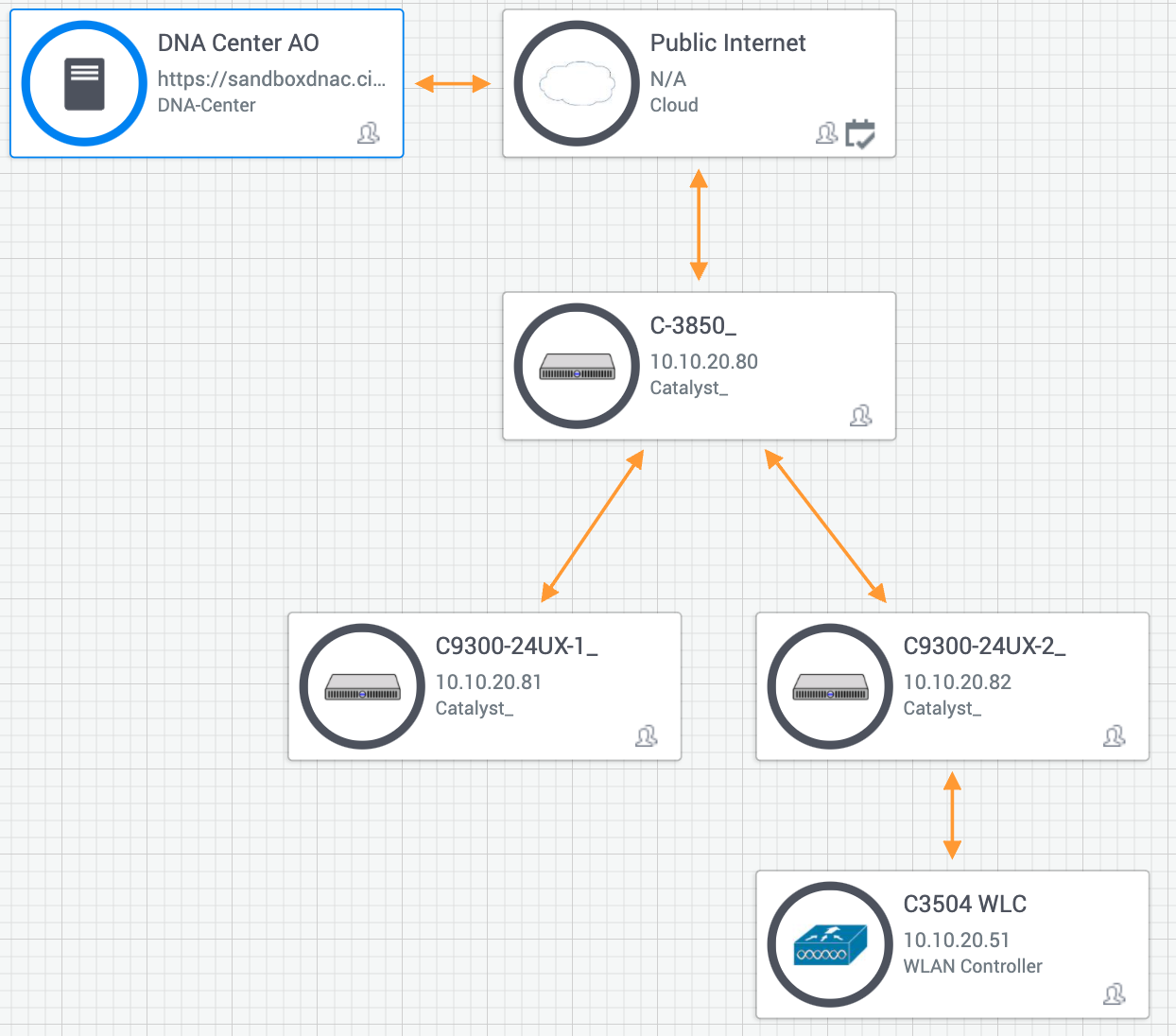
* **Simpler than NETCONF** for basic automation tasks
* **Human-readable** (JSON) vs. XML in NETCONF
* **Faster integration** with modern tools (Python, Postman, Ansible)
* **Scalable** for cloud-based network controllers (like DNA Center)

**Network Automation Protocols**

|  |  |  |
| --- | --- | --- |
| Feature | REST API (DNA Center) | NETCONF/YANG (Traditional) |
| Protocol | HTTP/S | SSH/TLS |
| Data Format | JSON | XML |
| Access | Through DNA Center | Direct device access |
| Best For | High-level automation | Device-specific configurations |
| Example Use | Device inventory, VLAN creation | Hostname changes, interface configs |

*Note: This assignment focuses on DNA Center's REST API*

**Lab Topology**



**Lab Setup**

**Required Files**

📂 cisco\_dnac\_assignment/

├── 📄 dnac\_config.py # API credentials

└── 📄 10\_dnac\_network.py # Main automation script

**Complete Code Files**

**dnac\_config.py**

DNAC = {

"host": "sandboxdnac.cisco.com",

"port": 443,

"username": "devnetuser",

"password": "Cisco123!"

}

**10\_dnac\_network.py**

**\_\_init\_\_(self)**

The constructor method that initializes the DNAC\_Manager class. It sets the initial token attribute to None, which will later store the authentication token obtained from DNA Center.

**get\_auth\_token(self, display\_token=False)**

Handles authentication with Cisco DNA Center using Basic Auth. It:

* Sends a POST request to the token endpoint
* Stores the retrieved token in the instance variable self.token
* Optionally displays the token if display\_token=True
* Returns True on success or False on failure

**get\_network\_devices(self)**

Retrieves a list of all network devices managed by DNA Center. It:

* Requires a valid authentication token
* Makes a GET request to the network-device endpoint
* Returns a list of devices or None if failed
* Includes error handling for failed requests

**display\_devices(self, devices)**

Formats and displays device information in a readable table. It:

* Takes a list of devices as input
* Prints a table with columns: Hostname, IP Address, Platform, and Status
* Handles empty device lists gracefully
* Uses string formatting for consistent column widths

**get\_device\_interfaces(self, device\_ip)**

Fetches interface details for a specific device. It:

* First locates the device by its management IP
* Retrieves interfaces using the device's internal ID
* Returns a list of interfaces or None if device not found
* Includes comprehensive error handling

**display\_interfaces(self, interfaces)**

Displays interface information in a formatted table. It:

* Takes a list of interfaces as input
* Prints a table with columns: Interface, Status, VLAN, and Speed
* Handles empty interface lists appropriately
* Formats output for easy readability

**main()**

The core function that runs the interactive menu system. It:

* Creates an instance of DNAC\_Manager
* Presents a continuous menu with 4 options
* Handles user input and calls appropriate methods
* Manages program exit cleanly

**if \_\_name\_\_ == "\_\_main\_\_":**

The Python idiom that ensures main() only runs when the script is executed directly (not when imported as a module). This is the script's entry point.

Each function follows the **Single Responsibility Principle**, handling one specific task while working together to provide complete DNA Center automation capabilities. The error handling and user feedback make it robust for real-world use.

import requests

from requests.auth import HTTPBasicAuth

from dnac\_config import DNAC

import urllib3

import sys

# Disable SSL warnings for sandbox

urllib3.disable\_warnings()

class DNAC\_Manager:

def \_\_init\_\_(self):

self.token = None

def get\_auth\_token(self, display\_token=False):

"""Authenticates to DNA Center and stores token"""

try:

url = f"https://{DNAC['host']}:{DNAC['port']}/dna/system/api/v1/auth/token"

response = requests.post(

url,

auth=HTTPBasicAuth(DNAC['username'], DNAC['password']),

verify=False,

timeout=10

)

response.raise\_for\_status()

self.token = response.json()['Token']

if display\_token:

print("\n🔑 Authentication Token:")

print("-"\*50)

print(self.token)

print("-"\*50)

return True

except Exception as e:

print(f"❌ Authentication failed: {str(e)}")

return False

def get\_network\_devices(self):

"""Retrieves all network devices"""

if not self.token:

print("⚠️ Please authenticate first!")

return None

try:

url = f"https://{DNAC['host']}:{DNAC['port']}/api/v1/network-device"

headers = {"X-Auth-Token": self.token}

response = requests.get(

url,

headers=headers,

verify=False,

timeout=10

)

response.raise\_for\_status()

return response.json().get('response', [])

except Exception as e:

print(f"❌ Failed to get devices: {str(e)}")

return None

def display\_devices(self, devices):

"""Formats device list output"""

if not devices:

print("No devices found!")

return

print("\n📡 Network Devices")

print("="\*80)

print(f"{'Hostname':20}{'IP Address':15}{'Platform':20}{'Status':10}")

print("-"\*80)

for device in devices:

print(

f"{device.get('hostname', 'N/A'):20}"

f"{device.get('managementIpAddress', 'N/A'):15}"

f"{device.get('platformId', 'N/A'):20}"

f"{device.get('reachabilityStatus', 'N/A'):10}"

)

def get\_device\_interfaces(self, device\_ip):

"""Retrieves interfaces for specific device"""

if not self.token:

print("⚠️ Please authenticate first!")

return None

try:

# Find device by IP

devices = self.get\_network\_devices()

device = next(

(d for d in devices if d.get('managementIpAddress') == device\_ip),

None

)

if not device:

print(f"❌ Device {device\_ip} not found!")

return None

# Get interfaces

url = f"https://{DNAC['host']}:{DNAC['port']}/api/v1/interface"

headers = {"X-Auth-Token": self.token}

params = {"deviceId": device['id']}

response = requests.get(

url,

headers=headers,

params=params,

verify=False,

timeout=10

)

response.raise\_for\_status()

return response.json().get('response', [])

except Exception as e:

print(f"❌ Failed to get interfaces: {str(e)}")

return None

def display\_interfaces(self, interfaces):

"""Formats interface output"""

if not interfaces:

print("No interfaces found!")

return

print("\n🔌 Device Interfaces")

print("="\*80)

print(f"{'Interface':20}{'Status':10}{'VLAN':10}{'Speed':10}")

print("-"\*80)

for intf in interfaces:

print(

f"{intf.get('portName', 'N/A'):20}"

f"{intf.get('status', 'N/A'):10}"

f"{intf.get('vlanId', 'N/A'):10}"

f"{intf.get('speed', 'N/A'):10}"

)

def main():

"""Main program execution"""

print("\n" + "="\*50)

print("Cisco DNA Center Network Automation")

print("Canadian College of Technology and Business (CCTB)")

print("="\*50 + "\n")

dnac = DNAC\_Manager()

while True:

print("\n🔧 Main Menu")

print("1. Authenticate & Show Token")

print("2. List Network Devices")

print("3. Show Device Interfaces")

print("4. Exit")

choice = input("Select option (1-4): ").strip()

if choice == "1":

if dnac.get\_auth\_token(display\_token=True):

print("✅ Authentication successful!")

elif choice == "2":

devices = dnac.get\_network\_devices()

dnac.display\_devices(devices)

elif choice == "3":

device\_ip = input("Enter device IP address: ").strip()

interfaces = dnac.get\_device\_interfaces(device\_ip)

dnac.display\_interfaces(interfaces)

elif choice == "4":

print("Goodbye! 👋")

sys.exit()

else:

print("❌ Invalid choice. Please try again.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Example Output**

**Authentication**

🔑 Authentication Token:

--------------------------------------------------

eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI2MGVj...

--------------------------------------------------

✅ Authentication successful!

**Device Listing**

📡 Network Devices

================================================================================

Hostname IP Address Platform Status

--------------------------------------------------------------------------------

Switch1 10.10.20.1 WS-C3850 Reachable

Router1 10.10.20.2 ISR4331 Reachable

**Interface View**

🔌 Device Interfaces

================================================================================

Interface Status VLAN Speed

--------------------------------------------------------------------------------

GigabitEthernet1/0/1 up 10 1000

GigabitEthernet1/0/2 down 1 auto

### **Assignment Tasks**

### **1. Create EC2 Instances**

* Create **two EC2 instances** using **Amazon Linux 2 AMI**:
  1. WebServer-EC2: for Django.
  2. MongoDB-EC2: for MongoDB database.
* Configure **security groups**:
  1. Allow HTTP (80), SSH (22).
  2. Allow MongoDB (27017) from the web server’s IP.
  3. Allow Django 8000 port.

### **2. Install Required Software**

* On WebServer-EC2:

Install Djago

* On MongoDB-EC2:

Install MongoDB

Ensure MongoDB is running and listening on port 27017.

#### **3. Build the Django Project**

* Project name: **assignment9**
* App name: **dna-center-cisco**

**Implementation Requirements**

**1. Migrate the automation logic into Django**

* Create a Django web interface to:
  + Authenticate and show token
  + List network devices
  + Show interface details by device IP
* Create three views (or API endpoints) corresponding to the options above
* Display responses inside styled HTML templates

**2. MongoDB Integration**

* Store:
  + Timestamp
  + IP address of the device (if applicable)
  + Result (success/failure)

**Submission Requirements**

Create a GitHub repository containing:

* Full Django project folder
* requirements.txt with all dependencies (requests, pymongo, dnac\_config, Django, etc.)
* Setup instructions in a README.md

**Include Screenshots of:**

* Authentication Token displayed in browser (via Django view)
* Device List output rendered via Django
* Interface Details for at least one device IP
* Django app running in browser showing **public IP**
* MongoDB terminal showing saved log entries (you may use mongo, mongosh, or Robo3T)
* Your GitHub repository with all code