Managed Campus

Operations Guide

Table of Contents

1.0 About This Guide

1.1 Objective

The objective of this document is to provide an in-depth overview of the Cisco Managed Campus offering, detailing its components, benefits, and operational advantages. This document aims to explain how the Cisco Managed Campus, powered by the Cisco Catalyst Center platform, enables Managed Service Providers (MSPs) to deliver reliable, secure, and easily manageable networking services. By integrating networking hardware, software, and devops services into a cohesive managed services package, the document will demonstrate how this solution supports the full lifecycle of digital campus transformation, simplifies network infrastructure management, and addresses the evolving needs of modern businesses and their remote workforce.

1.2 Scope

This guide covers the operational aspects of Catalyst Center, with emphasis on intelligent automation of Day1 and Day2 activities. While Day 1 typically focuses on initial deployment and configuration, Day 2 operations involve activities such as monitoring, provisioning, automation and assurance. We will also touch upon automation during onboarding, showcasing its relevance for partners and how it can streamline the deployment process.

Who should use this guide?

The primary objective of this guide is to equip Network engineers with the knowledge and skills needed to navigate the complexities of day-to-day operations with Catalyst center. Whether you are responsible for managing policies, performing upgrades, or creating reports, this guide is tailored to address your needs and enhance your proficiency in leveraging the capabilities of Cisco Catalyst Center with the former name as Cisco Catalyst Center.

1.3 Target Audience

This document is crafted for MSPs and Network Professionals, recognizing their unique role in safeguarding the digital assets of diverse clients. The content within this document is tailored to resonate with their challenges and requirements ensuring that every operational insight aligns with the objectives and responsibilities of managing security in a service provider context.

This guide is intended for:

- Managed Services Providers (MSP) - Those who work with Catalyst Center, providing services to customers. The guide will showcase intelligent automation, offering insights into efficient deployment and assurance practices.
- Network professionals - Responsible for day-to-day management and maintenance of the network security infrastructure. Individuals involved in creating and monitoring security policies, reporting, and standardizing firmware.

2.0 Introduction

2.1 Managed Campus Overview

The Cisco Managed Campus is a robust Network-as-a-Service (NaaS) solution tailored for Managed Service Providers (MSPs). Utilizing the Cisco Catalyst Center platform, this offering delivers reliable, secure networking with simplified management to meet the dynamic needs of contemporary businesses and their remote workforce. The solution integrates networking hardware, software, and security services into a cohesive managed services package, supporting the full lifecycle of digital campus transformation. Designed to streamline the management and operation of network infrastructure, the Cisco Managed Campus enhances efficiency and performance for organizations.

How do MSPs benefit by offering Network-as-a-Service / Managed Campus as a Managed Service offering?

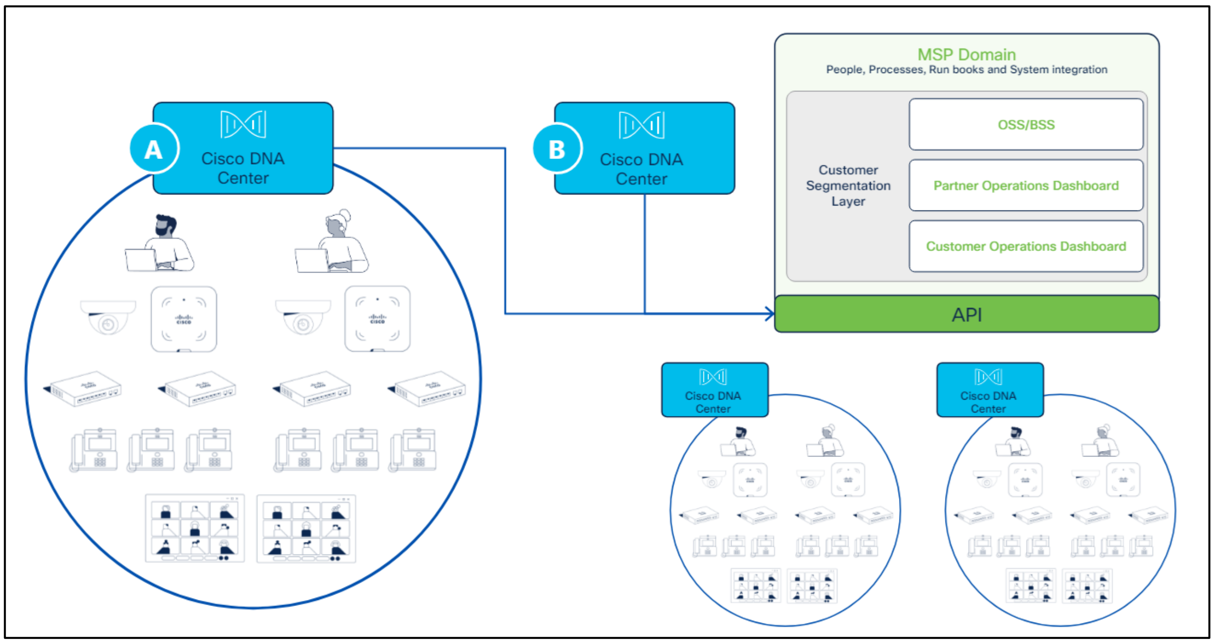
Beyond the practical features and functions of the Cisco Catalyst platform, MSPs have several compelling business incentives to create managed services around Cisco's Managed Campus offering:

- **High Market Demand** – The growing complexity of IT systems and the essential need for consistent network performance have led to an increased market demand for managed services. Companies, especially those with limited IT resources, are seeking MSPs to manage their network infrastructure efficiently.
- **Streamlined Operations through Automation** – Cisco's Managed Campus solutions provide automation tools that can enhance network operations efficiency. MSPs can utilize these capabilities to minimize the labour and costs associated with network management, which can translate into improved profit margins.
- **Diversified Service Portfolio** – By leveraging Cisco's Managed Campus, MSPs can expand their offerings to include tailor-made configurations, in-depth analytics, cybersecurity enhancements, and compliance services. These additional options can elevate client satisfaction and open new revenue opportunities for MSPs.
- **Scalability and Adaptability** – The scalable nature of Cisco's Managed Campus solutions ensures MSPs can adjust to their clients' expansion needs with ease. This adaptability enables MSPs to cater to various client sizes and types, from small firms to sizable corporations.
- **Steady Revenue Stream** – Offering managed services via Cisco's Managed Campus facilitates a transition for MSPs to a subscription-based business model, moving away from one-time transactions. This shift results in a more consistent and reliable revenue stream as customers pay regularly for ongoing network management and support.

2.2 Lab Topology Components and Versions

The following table provides a quick reference for the types, versions, and descriptions of the key components in the lab topology, facilitating an understanding of the technology stack used in the deployment. For this demo we are utilizing the below lab components:

Component	Model	Version
Catalyst Center	DN2-HW	2.3.3.7
Fusion Device	ISR 4331	17.9.2a
Edge Device	Catalyst 9300	19.9.2
WLC Controller	Catalyst 9800	17.8.1
Access Point	Catalyst 9130	17.9.2
Wired Client	Windows	11
Wireless Client	Windows	11



A
Cisco DNA Center
B
Cisco DNA Center
MSP Domain
People, Processes, Run books and System integration
Customer Segmentation Layer
OSS/BSS
Partner Operations Dashboard
Customer Operations Dashboard
API
Cisco DNA Center
Cisco DNA Center

The diagram highlights how Managed Service Providers (MSPs) can create and support Cisco Catalyst Center services at scale. Here's a breakdown of the components and their roles:

1. **MSP Domain**
Customer Segmentation Layer: This is a foundational layer that allows the MSP to segment and manage different customer environments effectively.
OSS/BSS (Operational Support Systems / Business Support Systems): These systems handle the backend processes such as network management, service provisioning, billing, and customer support.
Partner Operations Dashboard: A tool for managing and monitoring the operations and performance of the services provided to various customers.
Customer Operations Dashboard: A dashboard for managing individual customer networks, providing visibility into their specific operations and performance metrics.

2. **Cisco Domain**
Catalyst Center (DNAC): Cisco's network management and command center, providing automation, analytics, and assurance for the network.
Webex: Cisco's collaboration tool, which could be integrated into the managed services.

3. **Customer Domain**
This domain represents the different customers being managed by the MSP. Each customer domain includes:

Customer A Branch/Campus/Data Center
Customer B Branch/Campus/Data Center
Customer C Branch/Campus/Data Center
Each of these customer environments is managed independently but through a unified approach provided by the MSP.

4. **Interactions and Integrations**
Cisco and 3rd Party Integrations: The MSP domain interacts with Cisco's platforms and potentially other third-party tools for comprehensive service delivery.
Intent APIs: These Application Programming Interfaces allow for the automation of network configurations and policies based on business intent.
Event & Notifications: The system is capable of sending alerts and notifications based on events occurring within the network.

3.0 Operational Use Cases

This guide is organized around key use cases, aligning with the three pivotal phases of the infrastructure lifecycle: **Monitoring**, **Provisioning**, **Automation** & **Assurance**.

Each set of use cases has been curated from the perspective of an operations engineer, ensuring a comprehensive exploration of capabilities throughout the entire lifecycle.

3.1 Use Cases for Monitoring

3.1.1 MSP Managing Multiple Catalyst-Center: Event Notification, Issue Identification and Automated Mitigation

Objective: To configure Cisco Catalyst Center to send event notifications and alerts directly to Webex Teams, enabling a Managed Service Provider (MSP) to efficiently manage multiple client networks. This setup aims to improve real-time issue identification and automated mitigation, ensuring timely and reactive network management while enhancing communication and collaboration within the MSP's team.

Implementation:

Step1: Event Notification -- Steps to configure Cisco Catalyst Center Event Notifications for Webex Teams

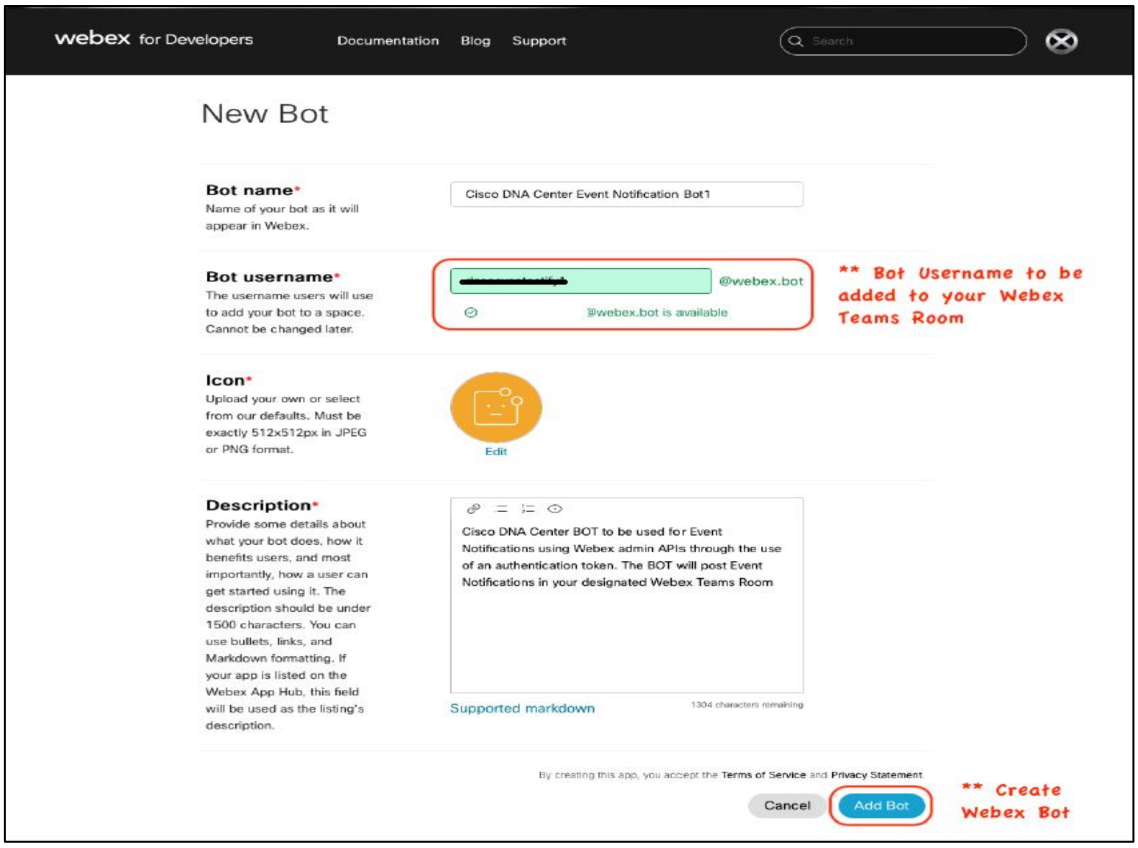
1. **Create a Webex Bot:**

Access Webex Developer Portal: Go to the **Webex Developer portal** and log in with your Cisco credentials.

Create a New Bot: Navigate to the "My Apps" section and click on "Create a New App." Choose "Create a Bot" from the available options.

Configure Bot Details: Enter the required details such as bot name, bot username, and a unique icon. Note the bot's access token for later use.

Save Bot Configuration: Save the bot configuration. The bot is now created and ready to be integrated with Webex Teams and Cisco Catalyst Center.



webex for Developers Documentation Blog Support Search

New Bot

Bot name* Name of your bot as it will appear in Webex. — Cisco DNA Center Event Notification Bot1

Bot username* The username users will use to add your bot to a space. Cannot be changed later. — [redacted] @webex.bot — @webex.bot is available

** Bot Username to be added to your Webex Teams Room

Icon* Upload your own or select from our defaults. Must be exactly 512x512px in JPEG or PNG format. — Edit

Description* Provide some details about what your bot does, how it benefits users, and most importantly, how a user can get started using it. The description should be under 1500 characters. You can use bullets, links, and Markdown formatting. If your app is listed on the Webex App Hub, this field will be used as the listing's description.

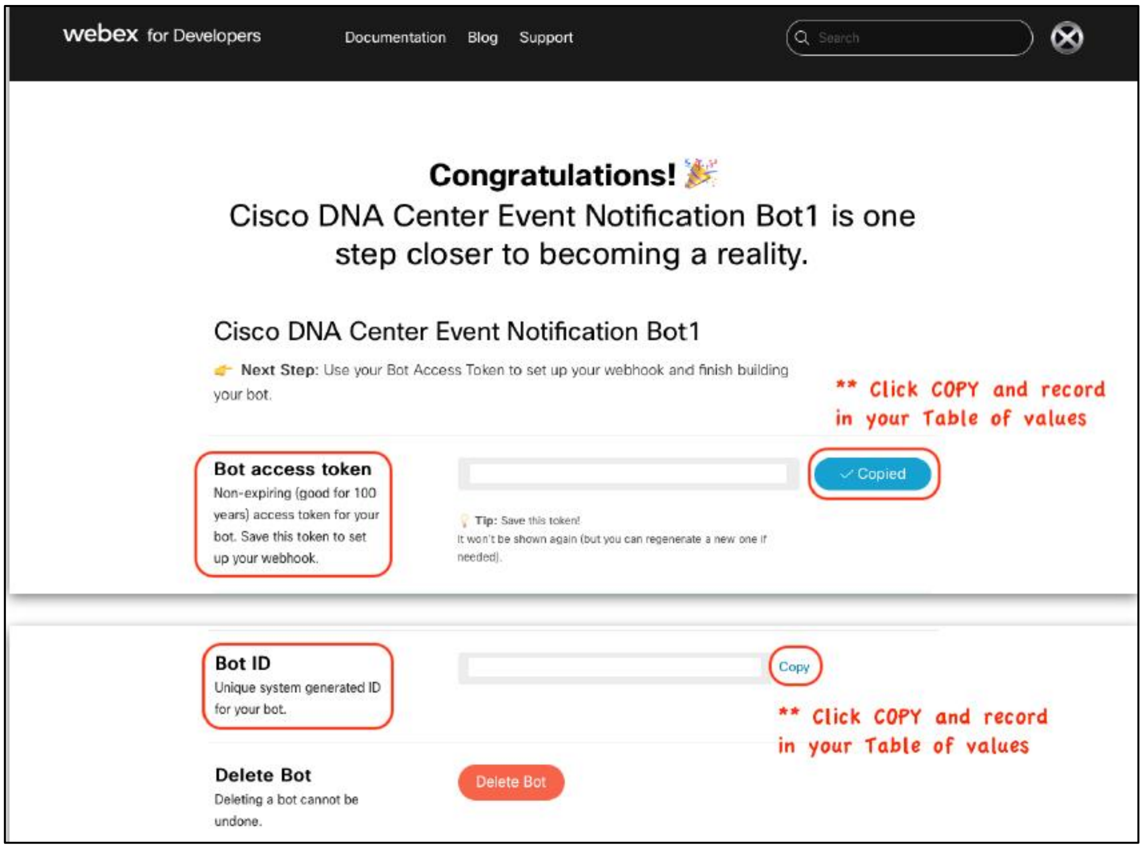
Cisco DNA Center BOT to be used for Event Notifications using Webex admin APIs through the use of an authentication token. The BOT will post Event Notifications in your designated Webex Teams Room

Supported markdown — 1304 characters remaining

By creating this app, you accept the Terms of Service and Privacy Statement.

Cancel | Add Bot

** Create Webex Bot



webex for Developers Documentation Blog Support Search

Congratulations!

Cisco DNA Center Event Notification Bot1 is one step closer to becoming a reality.

Cisco DNA Center Event Notification Bot1

Next Step: Use your Bot Access Token to set up your webhook and finish building your bot.

** Click COPY and record in your Table of values

Bot access token Non-expiring (good for 100 years) access token for your bot. Save this token to set up your webhook. — ✓ Copied

Tip: Save this token! It won't be shown again (but you can regenerate a new one if needed).

Bot ID Unique system generated ID for your bot. — Copy

** Click COPY and record in your Table of values

Delete Bot Deleting a bot cannot be undone. — Delete Bot

2. **Create a Webex Space for Cisco Catalyst Center Event Notifications:**

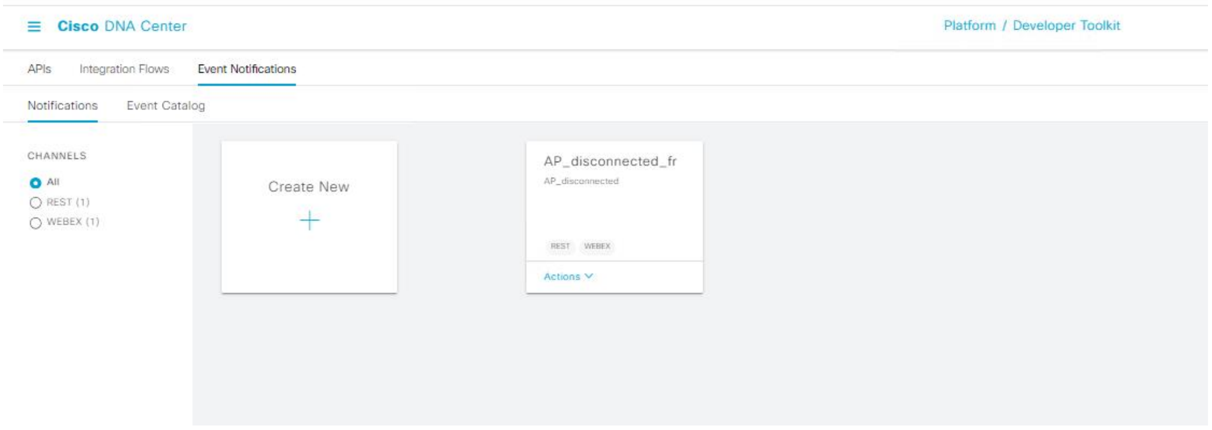
Access Webex: Log in to Webex using your credentials.

Create a New Space: Click on the "+" icon to create a new space. Name the space appropriately, e.g., "Cisco Catalyst Center Event Notifications."

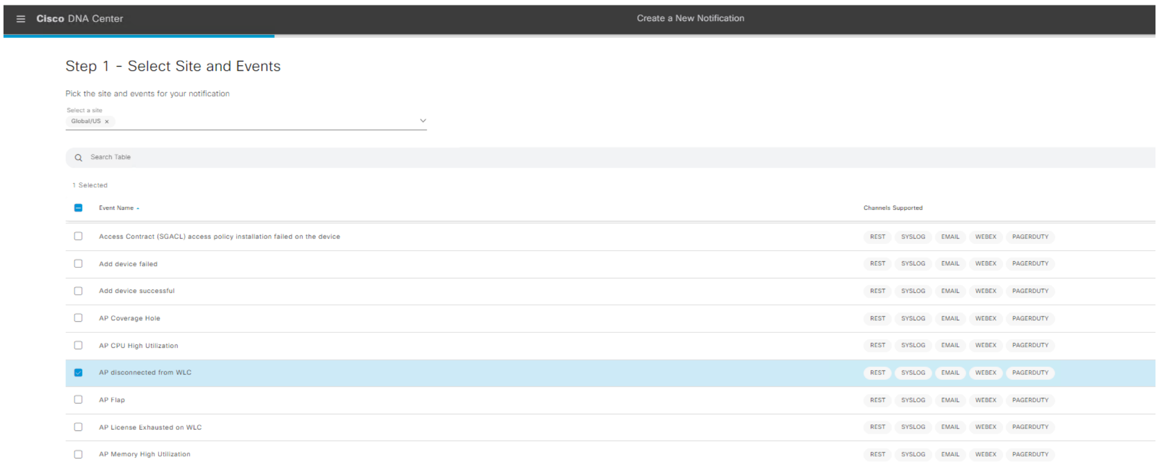
Add Members: Invite team members who should receive the notifications. Also, add the newly created Webex bot to this room using the bot's username.

3. **Create Event Notification on the Cisco Catalyst Center**:

Log in to the Cisco Catalyst Center using your admin credentials.
Navigate to Event Notifications: Go to the Platform -> Developer Toolkit -> Event notifications tab



Create a New Notification: Click on "Create New" and select the site(optional) and event type that should trigger notifications.


Cisco DNA Center — Create a New Notification

Step 1 - Select Site and Events

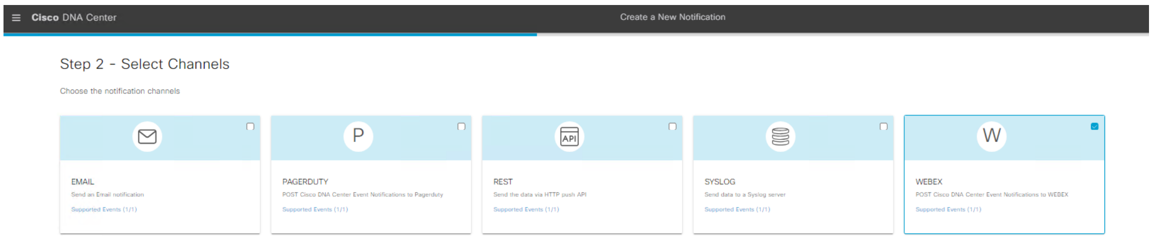
Pick the site and events for your notification

Select a site: Global/US

1 Selected

	Event Name	Channels Supported
☐	Access Contract (SGACL) access policy installation failed on the device	REST SYSLOG EMAIL WEBEX PAGERDUTY
☐	Add device failed	REST SYSLOG EMAIL WEBEX PAGERDUTY
☐	Add device successful	REST SYSLOG EMAIL WEBEX PAGERDUTY
☐	AP Coverage Hole	REST SYSLOG EMAIL WEBEX PAGERDUTY
☐	AP CPU High Utilization	REST SYSLOG EMAIL WEBEX PAGERDUTY
✓	AP disconnected from WLC	REST SYSLOG EMAIL WEBEX PAGERDUTY
☐	AP Flap	REST SYSLOG EMAIL WEBEX PAGERDUTY
☐	AP License Exhausted on WLC	REST SYSLOG EMAIL WEBEX PAGERDUTY
☐	AP Memory High Utilization	REST SYSLOG EMAIL WEBEX PAGERDUTY

Then select the channel you want to send the notifications through – Webex. Then click Next


Cisco DNA Center — Create a New Notification

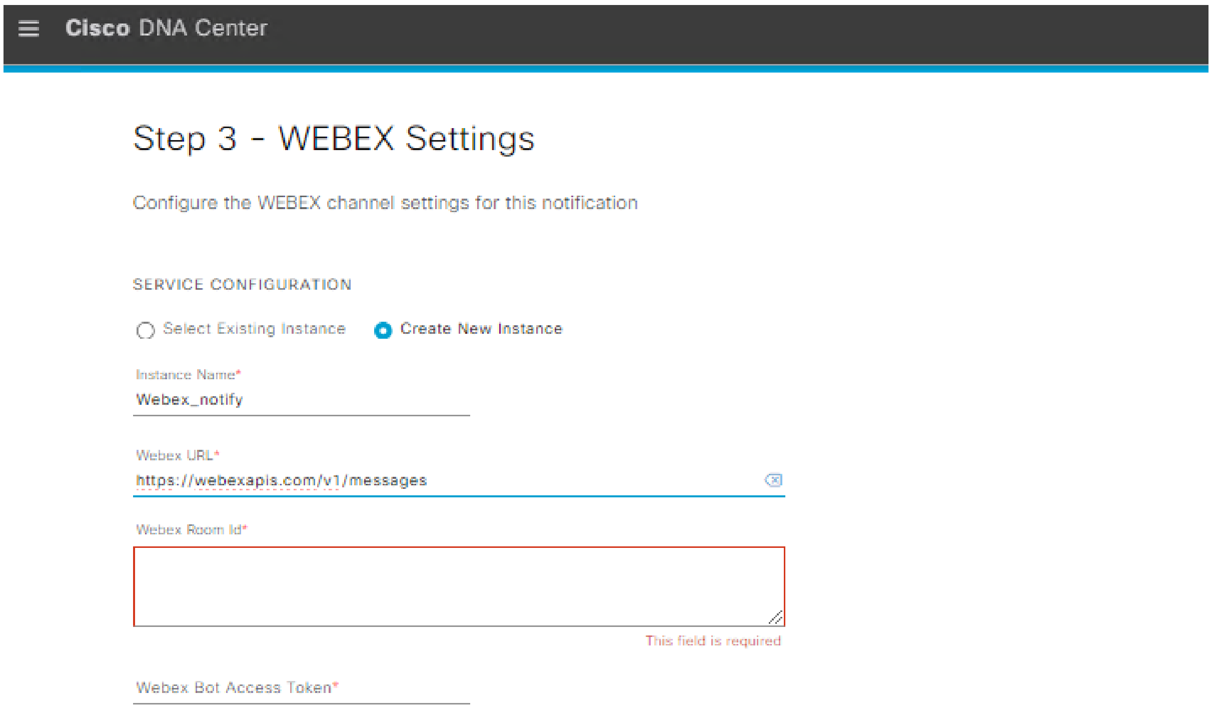
Step 2 - Select Channels

Choose the notification channels

☐	EMAIL	Send an Email notification. Supported Events (1/1)
☐	PAGERDUTY	POST Cisco DNA Center Event Notifications to Pagerduty. Supported Events (1/1)
☐	REST	Send the data via HTTP push API. Supported Events (1/1)
☐	SYSLOG	Send data to a Syslog server. Supported Events (1/1)
✓	WEBEX	POST Cisco DNA Center Event Notifications to WEBEX. Supported Events (1/1)

On the next step you need to populate the Webex room ID and Webex Bot token (that we created earlier)

To find the Webex room ID, login to developer.webex.com, login with your Webex credentials, go to the Documentation -> API -> Messaging -> Reference -> Rooms
Make the API request and you'll receive a response containing all rooms with their Room Ids. Copy the Room ID of your space and then paste it on the Catalyst Center configuration.



Step 3 - WEBEX Settings

Configure the WEBEX channel settings for this notification

SERVICE CONFIGURATION

Select Existing Instance [✓ Create New Instance]

Instance Name*
Webex_notify

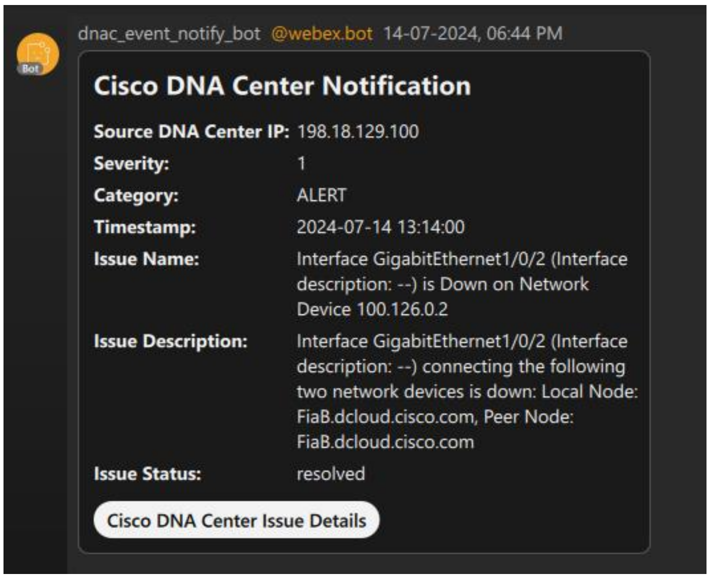
Webex URL*
https://webexapis.com/v1/messages

Webex Room Id*

This field is required

Webex Bot Access Token*

Save the configuration. Now you are set to receive the configured notification on Webex, like below:



dnac_event_notify_bot @webex.bot 14-07-2024, 06:44 PM

Cisco DNA Center Notification

Source DNA Center IP:	198.18.129.100
Severity:	1
Category:	ALERT
Timestamp:	2024-07-14 13:14:00
Issue Name:	Interface GigabitEthernet1/0/2 (Interface description: --) is Down on Network Device 100.126.0.2
Issue Description:	Interface GigabitEthernet1/0/2 (Interface description: --) connecting the following two network devices is down: Local Node: FiaB.dcloud.cisco.com, Peer Node: FiaB.dcloud.cisco.com
Issue Status:	resolved

Step2: Issue Identification -- Workflow for the Script: Correlating Events to Issues and Identifying Active or Resolved Issues

Let us begin with installation of python on the client machine.

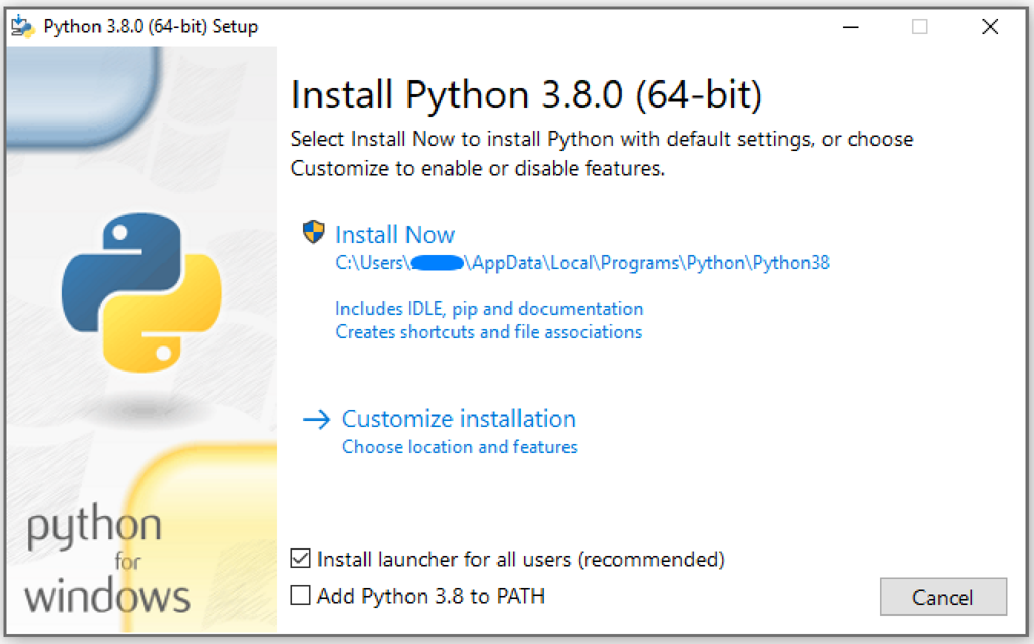
1. **Setup Python on Windows Machine**

There are several different installers available for Windows, each with certain benefits and downsides. The full installer contains all components and is the best option for developers using Python for any kind of project.

2. **Installation Steps**

Four Python 3.12 installers are available for download - two each for the 32-bit and 64-bit versions of the interpreter. The *web installer* is a small initial download, and it will automatically download the required components as necessary.

After starting the installer, one of two options may be selected:



Python 3.8.0 (64-bit) Setup

Install Python 3.8.0 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\[redacted]\AppData\Local\Programs\Python\Python38

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☐ Add Python 3.8 to PATH

If you select “Install Now”:

- You will *not* need to be an administrator (unless a system update for the C Runtime Library is required or you install the Python Launcher for Windows for all users)
- Python will be installed into your user directory
- The Python Launcher for Windows will be installed according to the option at the bottom of the first page
- The standard library, test suite, launcher and pip will be installed
- If selected, the install directory will be added to your PATH
- Shortcuts will only be visible for the current user

Selecting “Customize installation” will allow you to select the features to install, the installation location and other options or post-install actions. To install debugging symbols or binaries, you will need to use this option.

To perform an all-users installation, you should select “Customize installation”. In this case:

- You may be required to provide administrative credentials or approval
- Python will be installed into the Program Files directory
- The Python Launcher for Windows will be installed into the Windows directory
- Optional features may be selected during installation
- The standard library can be pre-compiled to bytecode
- If selected, the install directory will be added to the system PATH
- Shortcuts are available for all users

3. Workflow for fetching the Issues (ACTIVE or RESOLVED)

Objective: The objective of this script is to automate the retrieval and processing of events from multiple Cisco Catalyst Center, correlate these events with network issues, identify their status (active or resolved), and log the detailed information, including suggested actions, into an output file.

API Referenced: **Issues** and **Get_Device_by_ID**
Workflow:

1. **Initialization**
   Import Libraries: The script begins by importing necessary libraries such as **‘json’**, **‘csv’**, **‘requests’**, **‘time’**, and **‘PrettyTable’**.
   Suppress Warnings: Warnings related to insecure HTTP requests are suppressed using **‘warnings.filterwarnings’**.

2. **Configuration**
   User Input for Time Window: Prompt the user to input the duration in minutes for which data is to be pulled.
   Calculate Time Window: Calculate the start and end time in milliseconds based on the current time and the user input.
   Set Catalyst Center Credentials: Define Catalyst Center IP address (multiple entries to be pushed via a csv file), username, and password.

3. **CSV File Setup**
   Create CSV File: Open a CSV file named **‘commands_to_run.csv’** to write suggested actions.
   Write CSV Headers: Write the header row in the CSV file.

4. **Authentication**
   Get Authentication Token: Call **‘get_X_auth_token’** to obtain an authentication token from the Catalyst using the provided credentials.

5. **Fetch Issues**
   Define API Endpoint: Set the URL for the issues endpoint.
   Set Headers: Include the authentication token in the headers.
   Set Parameters: Define parameters for the API request, including start and end times, and issue status.

Send Request: Send a GET request to fetch active issues within the specified time window.
Parse Response: Extract the issues from the response JSON.

6. **Process Each Issue**
   Open Output File: Open a text file **'output.txt'** for writing detailed issue information.
   Iterate Through Issues: Loop through each issue in the fetched list.
   Create Table for Issue: Create a **'PrettyTable'** instance to display issue details.
   Fetch Device Details: Call **'get_device_detail'** to get details of the device associated with the issue.
   Fetch Enrichment Details: Send a request to the enrichment details endpoint with the issue ID.
   Extract Issue Details: Extract relevant details such as issue ID, name, description, priority, summary, and suggested actions.
   Add Row to Table: Add the extracted details to the **'PrettyTable'**.

7. **Handle Suggested Actions**
   Iterate Through Suggested Actions: For each suggested action in the issue details:
   Write Action Details: Write the suggested action message to the output file.
   Check for Steps: If there are steps in the suggested action:
   Iterate Through Steps: Loop through each step.
   Fetch Step Device Details: Call **'get_device_detail'** to get details of the device associated with the step.
   Write Step Details: Write the step details including device name, description, and command to the output file.
   Write to CSV: Write the step details to the CSV file.

8. **Finalize Output**
   Close Files: Close the output text file and the CSV file after writing all the details.

9. **Error Handling**
   Connection Errors: Handle connection errors when attempting to get the authentication token.
   Missing Data: Skip issues without enrichment details and continue with the next issue.

This workflow describes each step in the script, focusing on handling multiple Catalyst Centers by iterating through the list of issues, extracting relevant details, and writing the results to output files. The script ensures proper handling of API requests, data extraction, and file operations to manage and document network issues effectively.

Step3: Automated Mitigation -- Workflow for the Script: Automated Execution of Suggested Actions by Catalyst Center

Objective: To execute CLI commands on network devices through the Catalyst Center API, retrieve command execution results, and generate a formatted output

API Referenced: Get_Issue_Enrichment_Details and Command Runner
Workflow:

1. **Authentication:**
   The script initiates by authenticating with a given Catalyst Center (DNAC) using basic

authentication.
The function **‘get_X_auth_token’** is responsible for obtaining an authentication token required for subsequent API calls.

2. **Command Execution:**
   The script reads a list of commands from a CSV file (**‘commands_to_run.csv’**) along with device details. For each command in the CSV file, it initiates command execution on the specified network device.
   The **‘cmd_runner’** function sends the command execution request to the Catalyst Center API and returns a task ID.

3. **Task Status Checking:**
   The script then checks the status of the command execution task using the obtained task ID. The **‘get_task’** function retrieves the status of the task and waits if the task is still processing.
   Once the task is completed, it retrieves the command execution result.

4. **Result Processing:**
   If the command execution is successful, it writes the output to a temporary file (**‘temp.txt’**).
   If the command execution fails, it writes the error message to the same temporary file.

5. **Output Generation:**
   After processing all commands, the script generates a formatted output using **‘PrettyTable’**.
   The output includes details such as ‘Issue No.’, ‘Suggested Action’, ‘Device Name’, ‘Device ID’, ‘Command’, ‘Description’, and ‘Output’.
   It writes the formatted output to a final output file (**‘final_output.txt’**).

6. **Error Handling:**
   The script handles potential errors such as connection errors and timeouts gracefully by printing error messages and exiting the script.

7. **Security Considerations:**
   The script disables warnings related to insecure HTTPS requests (**‘InsecureRequestWarning’**) to suppress warnings about self-signed certificates. However, this might introduce security risks if used in production environments without proper certificate validation.

8. **Cleanup:**
   Temporary files (**‘temp.txt’**) are used for storing command execution results and are closed after processing.

This workflow provides a structured approach to executing CLI commands on network devices through the Catalyst Center API, handling authentication, command execution, status checking, result processing, and output generation efficiently while ensuring error handling and security considerations are addressed.

Notes:

If a device specified in the CSV file is unreachable during CLI command execution, the script may experience timeouts due to network issues or device unresponsiveness. To address this, we have implemented timeout mechanisms to prevent indefinite waiting periods for unresponsive devices.

3.1.2 Webhooks notification and SNOW ticket creation for selected events

Objective: The objective of this use case is to enhance the incident management process for the Managed Services Provider (MSP) by implementing an automated system that utilizes webhooks for real-time event notifications and integrates with ServiceNow (SNOW) to create support tickets for selected network events. This automation aims to improve the efficiency, responsiveness, and accuracy of the MSP's operations, ensuring that critical incidents are promptly addressed and resolved.

Implementation:

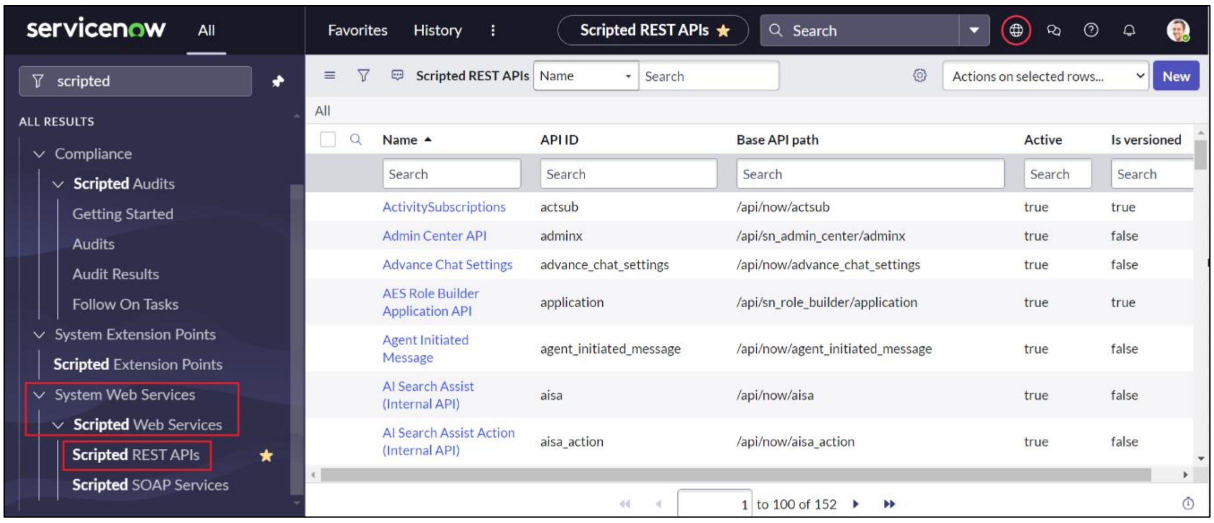
Step1: Integrate Webhooks with Service Now (SNOW)

In this section, we are going to dive into how to integrate Cisco Catalyst Center webhooks into ServiceNow. First let's talk about what a webhook is. A webhook is a user defined http call back to an external system. Basically, a webhook is a way to feed information from your application to an external system based on some event or action. Why is this important for your instance? Well, if you are monitoring or discovering a custom application for CI creation or updates, this could be one use for this type of integration.

Step2: Configuring Scripted REST API within System Web Services on ServiceNow (SNOW)

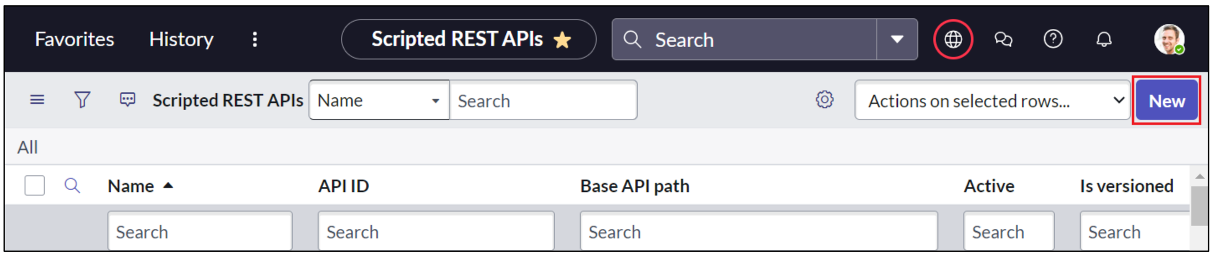
The purpose of scripted Web Services in ServiceNow is to allow for the creation of custom APIs to extend and integrate with the platform. Scripted Web Services in ServiceNow are custom APIs created using the Scripted REST API functionality.

We should have a ServiceNow instance spun up. It should have valid admin credentials for accessing scripted REST APIs in System Web Services. To start we need to create a Scripted REST API.

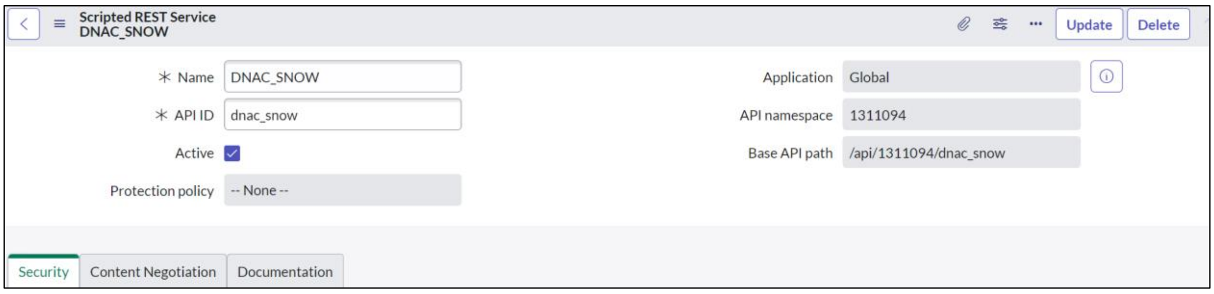


Name	API ID	Base API path	Active	Is versioned
ActivitySubscriptions	actsub	/api/now/actsub	true	true
Admin Center API	adminx	/api/sn_admin_center/adminx	true	false
Advance Chat Settings	advance_chat_settings	/api/now/advance_chat_settings	true	false
AES Role Builder Application API	application	/api/sn_role_builder/application	true	true
Agent Initiated Message	agent_initiated_message	/api/now/agent_initiated_message	true	false
AI Search Assist (Internal API)	aisa	/api/now/aisa	true	false
AI Search Assist Action (Internal API)	aisa_action	/api/now/aisa_action	true	false

Create a new Scripted REST API by clicking on 'New'. Fill out the form and click submit.

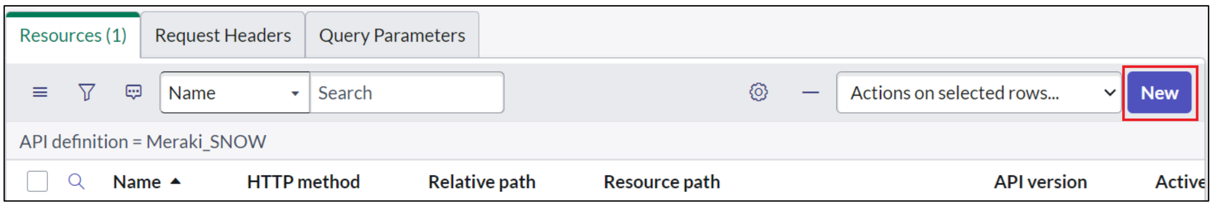


Name	API ID	Base API path	Active	Is versioned
Search	Search	Search	Search	Search



Name	DNAC_SNOW	Application	Global
API ID	dnac_snow	API namespace	1311094
Active	✓	Base API path	/api/1311094/dnac_snow
Protection policy	-- None --		

Now go back to the Scripted REST API's and find the one we just created and click on it. Change the HTTP method from GET to POST.

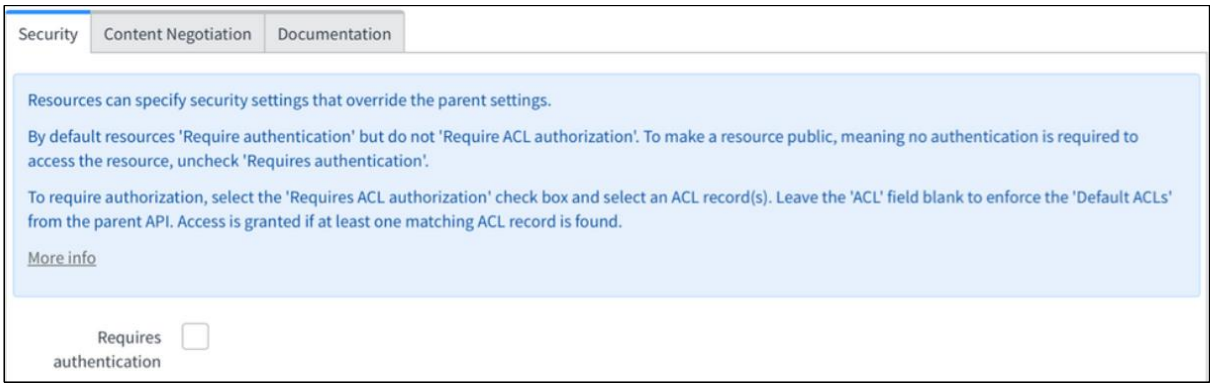


API definition = Meraki_SNOW

Name	HTTP method	Relative path	Resource path	API version	Active

Scroll down to Security and clear the check box for Requires Authentication.

- **Note when you deselect Requires Authentication you are allowing unauthenticated access to this Method on your instance. Since this is a demo and an example, we will not cover the security aspects of this.**



Resources can specify security settings that override the parent settings.

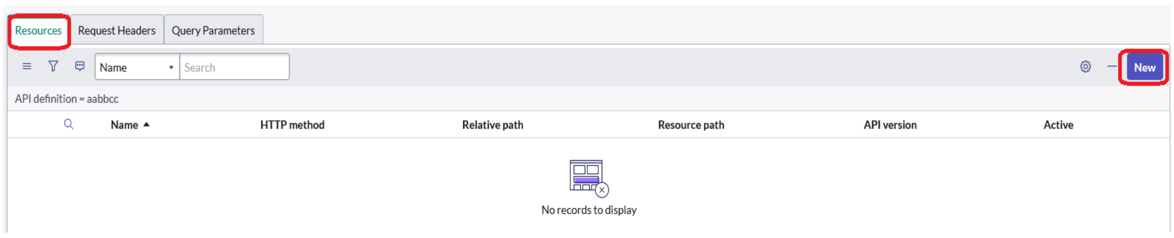
By default resources 'Require authentication' but do not 'Require ACL authorization'. To make a resource public, meaning no authentication is required to access the resource, uncheck 'Requires authentication'.

To require authorization, select the 'Requires ACL authorization' check box and select an ACL record(s). Leave the 'ACL' field blank to enforce the 'Default ACLs' from the parent API. Access is granted if at least one matching ACL record is found.

More info

Requires authentication ☐

Scroll down and under the Resources tab, create a new resource by clicking on “New”

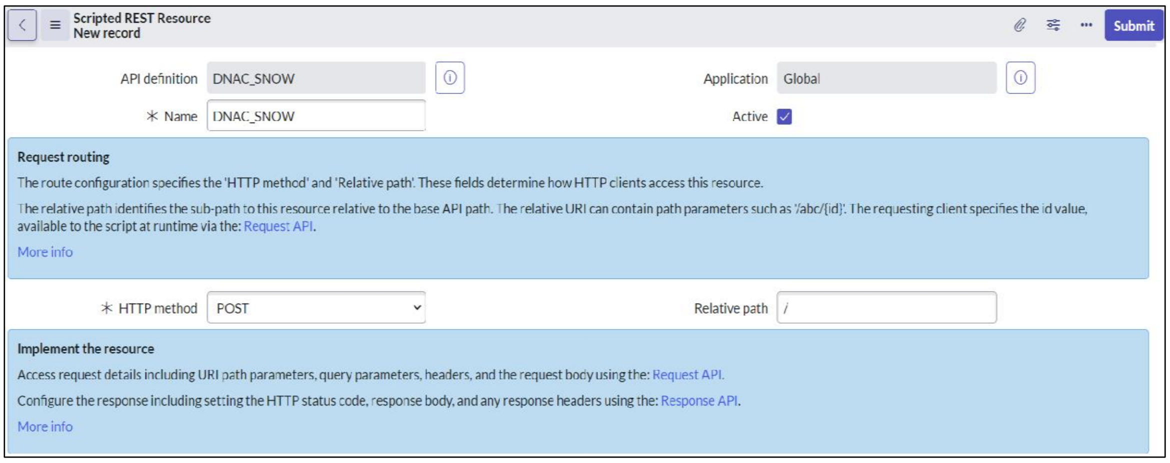


API definition = aabbcc

	Name ▲	HTTP method	Relative path	Resource path	API version	Active

No records to display

Fill in the name of the resource and relative path for this resource. Select HTTP method as POST. In the "Script" field, write a script to create an incident. Here's a basic example script:



Scripted REST Resource
New record

API definition: DNAC_SNOW
Application: Global
* Name: DNAC_SNOW
Active: ✓

Request routing

The route configuration specifies the 'HTTP method' and 'Relative path'. These fields determine how HTTP clients access this resource.

The relative path identifies the sub-path to this resource relative to the base API path. The relative URI can contain path parameters such as '/abc/{id}'. The requesting client specifies the id value, available to the script at runtime via the: Request API.

More info

* HTTP method: POST
Relative path: /

Implement the resource

Access request details including URI path parameters, query parameters, headers, and the request body using the: Request API.

Configure the response including setting the HTTP status code, response body, and any response headers using the: Response API.

More info

You will see there is some example code already provided for the resource.

Default Code:

```
(function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {
      // implement resource here
      })(request, response);
```

Change default to with this:

```
var grIncident = new GlideRecord('incident');
grIncident.initialize();
var x = request.body.dataString;
grIncident.category =  "Network";
grIncident.short.description = "Incident from DNAC Alert";
var parseddata =  JSON.parse(x);
grIncident.description = "Name : "+ parseddata.name +"\nDescription : "+ parseddata.descript
ion +"\nDNAC Link : "+ parseddata. ciscoDnaEventLink;
grIncident.work_notes = x;
grIncident.insert();
```

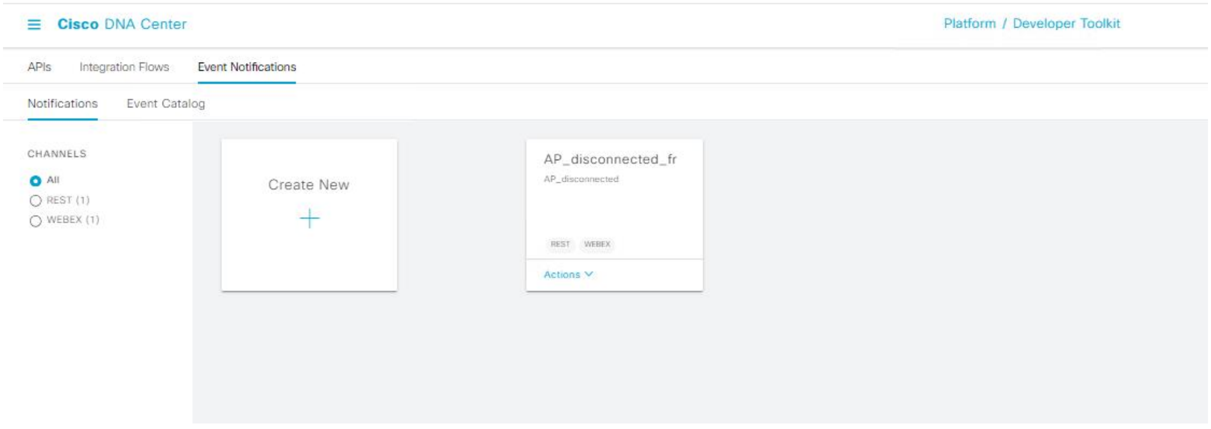
The Resource Path URL for our Scripted REST API is here.

Resource Path:	/api/1311094/dnac_snow

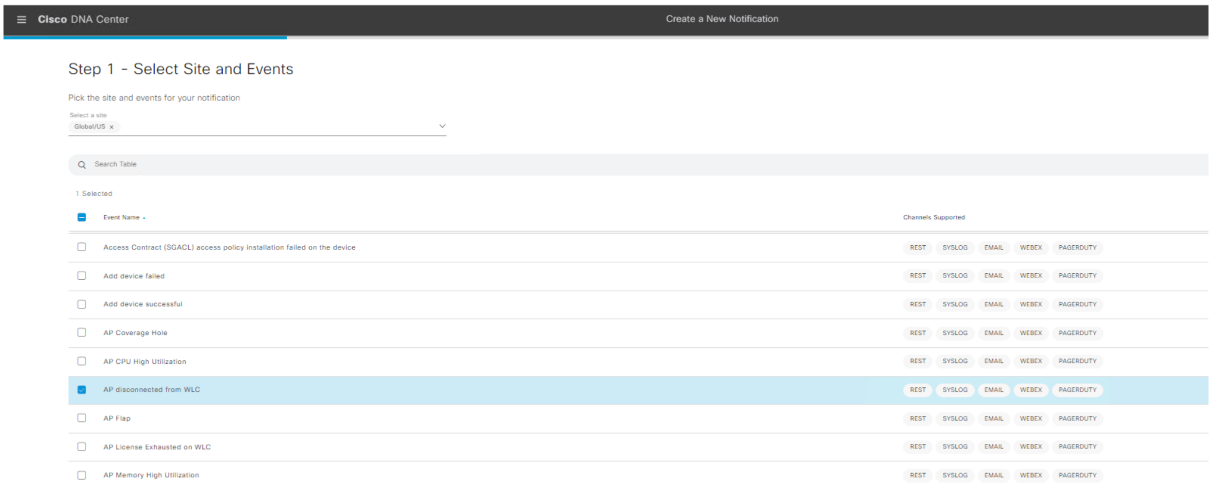
So, the URL to this resource would be https://<yourInstance>.service-now.com/<resource_Path>. Click Submit/Update on this resource.
Now that we have created our own custom REST API endpoint we can configure it as a webhook destination in Catalyst Center.

Step3: Configuring Event notifications through Webhooks on Catalyst Center

Log in to the Cisco Catalyst Center using your admin credentials.
Navigate to Event Notifications: Go to the Platform -> Developer Toolkit -> Event notifications tab



Create a New Notification: Click on "Create New" and select the site(optional) and event type that should trigger notifications.


Cisco DNA Center — Create a New Notification

Step 1 - Select Site and Events

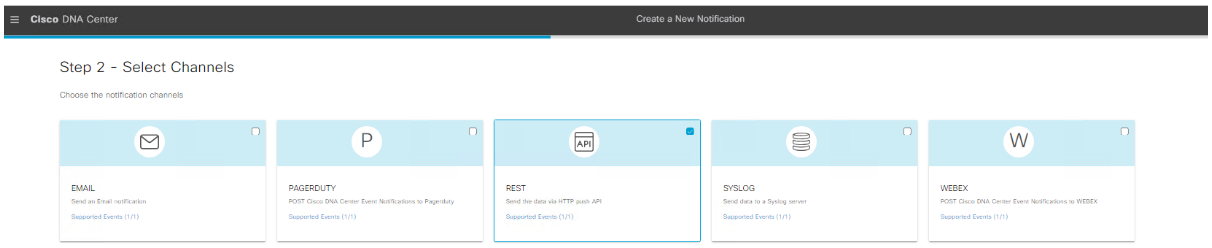
Pick the site and events for your notification

Select a site: Global/US

1 Selected

	Event Name	Channels Supported
☐	Access Contract (SGACL) access policy installation failed on the device	REST SYSLOG EMAIL WEBEX PAGERDUTY
☐	Add device failed	REST SYSLOG EMAIL WEBEX PAGERDUTY
☐	Add device successful	REST SYSLOG EMAIL WEBEX PAGERDUTY
☐	AP Coverage Hole	REST SYSLOG EMAIL WEBEX PAGERDUTY
☐	AP CPU High Utilization	REST SYSLOG EMAIL WEBEX PAGERDUTY
☑	AP disconnected from WLC	REST SYSLOG EMAIL WEBEX PAGERDUTY
☐	AP Flap	REST SYSLOG EMAIL WEBEX PAGERDUTY
☐	AP License Exhausted on WLC	REST SYSLOG EMAIL WEBEX PAGERDUTY
☐	AP Memory High Utilization	REST SYSLOG EMAIL WEBEX PAGERDUTY

Then select the channel you want to send the notifications through like REST(for webhook)

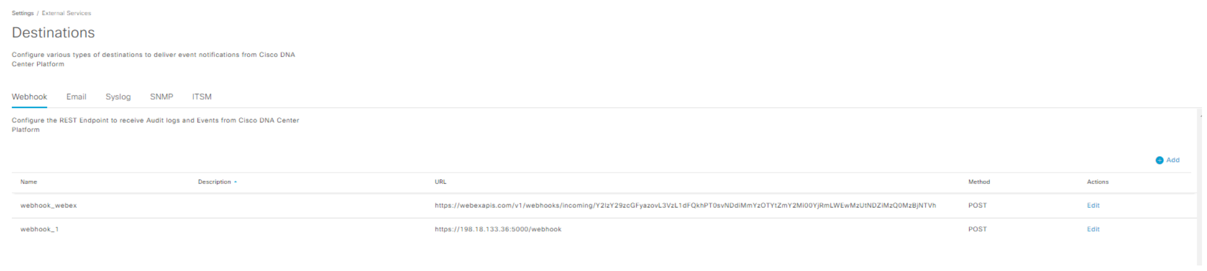

Cisco DNA Center — Create a New Notification

Step 2 - Select Channels

Choose the notification channels

☐ EMAIL	☐ PAGERDUTY	☑ REST	☐ SYSLOG	☐ WEBEX
Send an Email notification	POST Cisco DNA Center Event Notifications to Pagerduty	Send the data via HTTP push API	Send data to a Syslog server	POST Cisco DNA Center Event Notifications to WEBEX
Supported Events (1/1)	Supported Events (1/1)	Supported Events (1/1)	Supported Events (1/1)	Supported Events (1/1)

Next select the option to create a new setting instance and it will redirect to the 'Destination' page.



Settings / External Services

Destinations

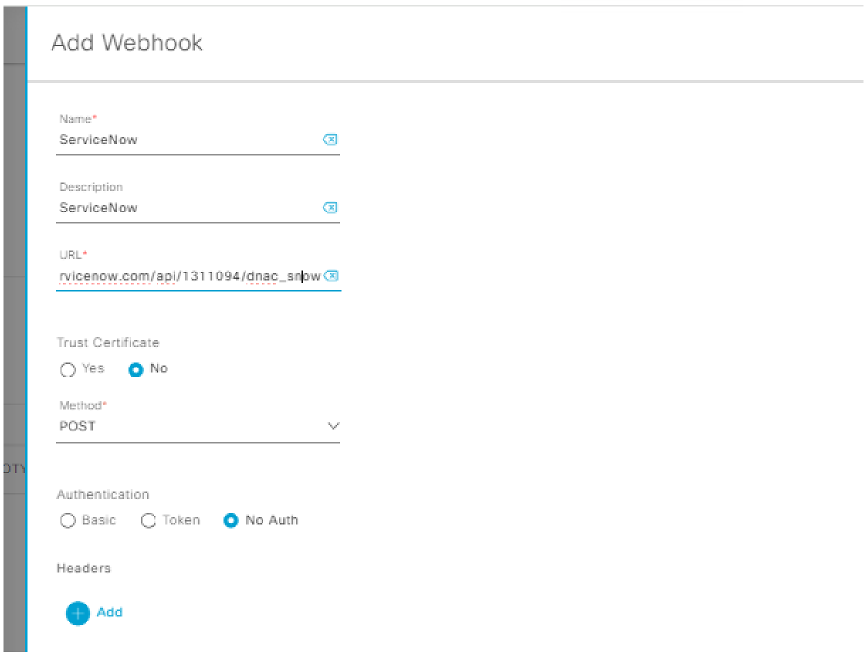
Configure various types of destinations to deliver event notifications from Cisco DNA Center Platform

Webhook | Email | Syslog | SNMP | ITSM

Configure the REST Endpoint to receive Audit logs and Events from Cisco DNA Center Platform

Name	Description	URL	Method	Actions
webhook_webex		https://webexapis.com/v1/webhooks/incoming/Y2lzY29zcGFyazovL3VzL1dFQkhPT0svNDdiMmYzOTYtZmY2Mi00YjRmLWEwMzUtNDZiMzQ0MzBjNTVh	POST	Edit
webhook_1		https://198.18.133.36:5000/webhook	POST	Edit

Click on 'Add' and Fill in the details you got from ServiceNow.



Add Webhook

Name*
ServiceNow

Description
ServiceNow

URL*
rvicenow.com/api/1311094/dnac_snow

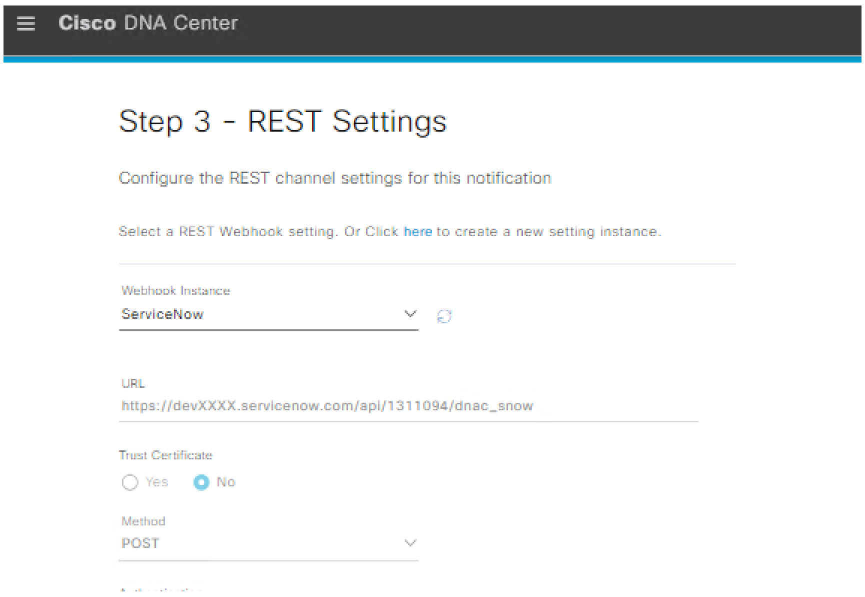
Trust Certificate
Yes [✓ No]

Method*
POST

Authentication
Basic Token [✓ No Auth]

Headers

Go back to the notification step 3 and select the newly created destination.



Cisco DNA Center

Step 3 - REST Settings

Configure the REST channel settings for this notification

Select a REST Webhook setting. Or Click here to create a new setting instance.

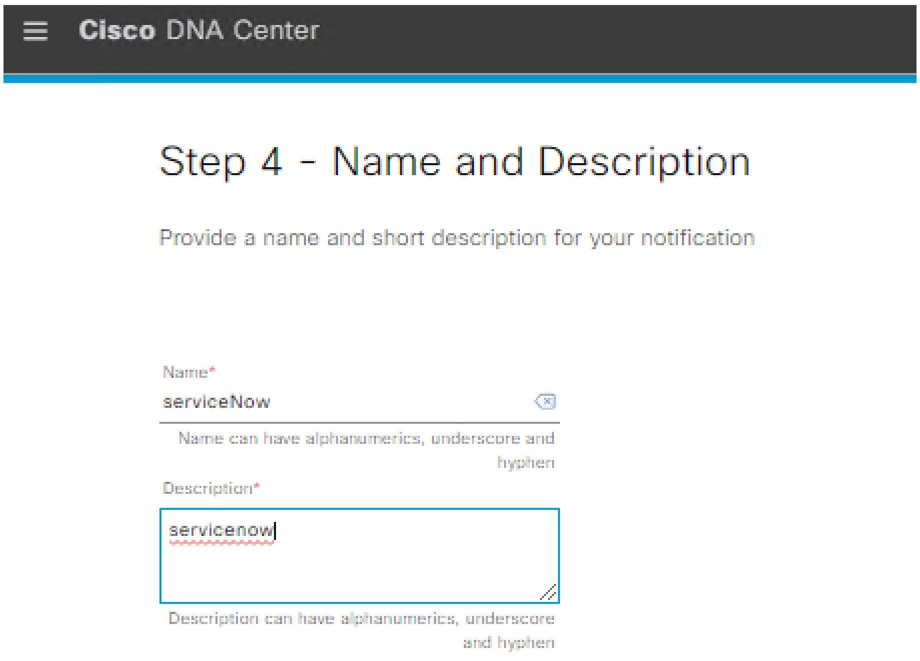
Webhook Instance
ServiceNow

URL
https://devXXXX.servicenow.com/api/1311094/dnac_snow

Trust Certificate
Yes [✓ No]

Method
POST

Give a descriptive name to the notification and save


Cisco DNA Center

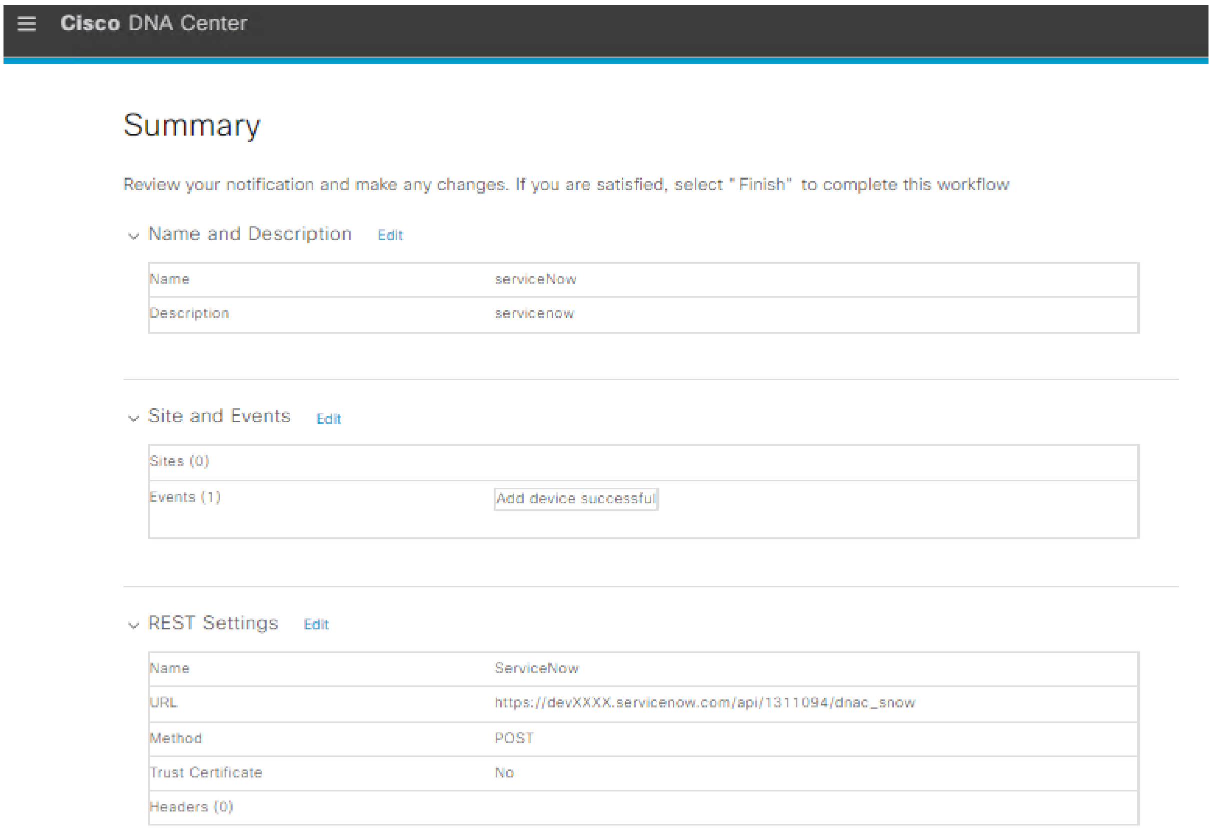
Step 4 - Name and Description

Provide a name and short description for your notification

Name*	serviceNow
Description*	servicenow

Name can have alphanumerics, underscore and hyphen

Description can have alphanumerics, underscore and hyphen


Cisco DNA Center

Summary

Review your notification and make any changes. If you are satisfied, select " Finish" to complete this workflow

Name and Description Edit

Name	serviceNow
Description	servicenow

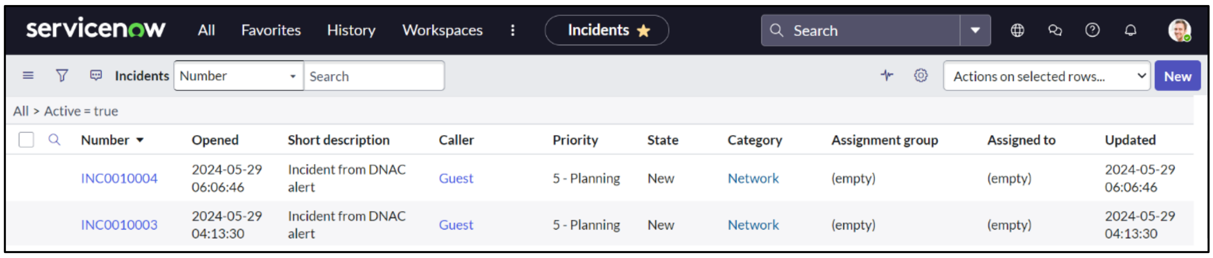
Site and Events Edit

Sites (0)	
Events (1)	Add device successful

REST Settings Edit

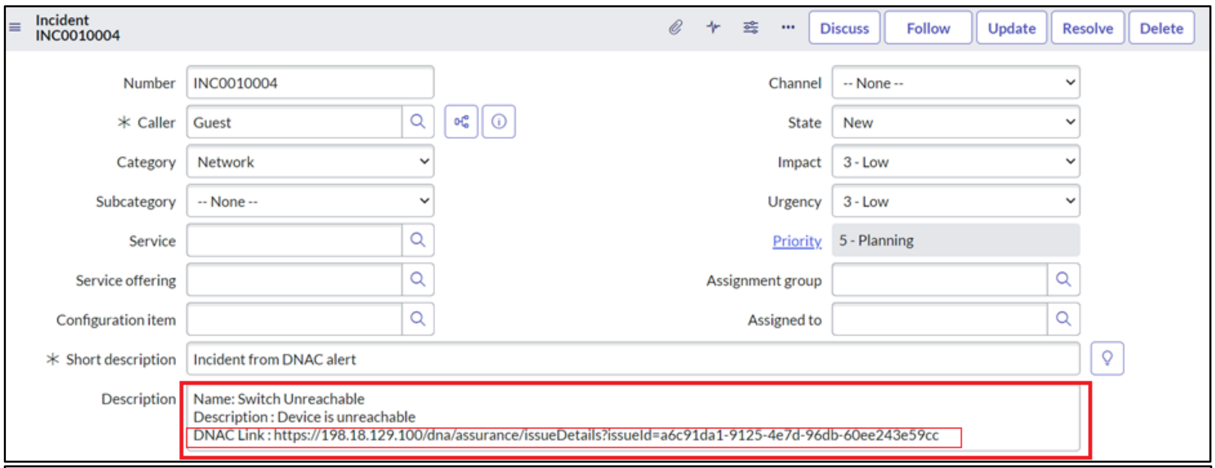
Name	ServiceNow
URL	https://devXXXX.servicenow.com/api/1311094/dnac_snow
Method	POST
Trust Certificate	No
Headers (0)	

Once configured successfully, we will start receiving the real-time alerts from Cisco Catalyst Center in the form of Incident ticket on ServiceNow (SNOW).



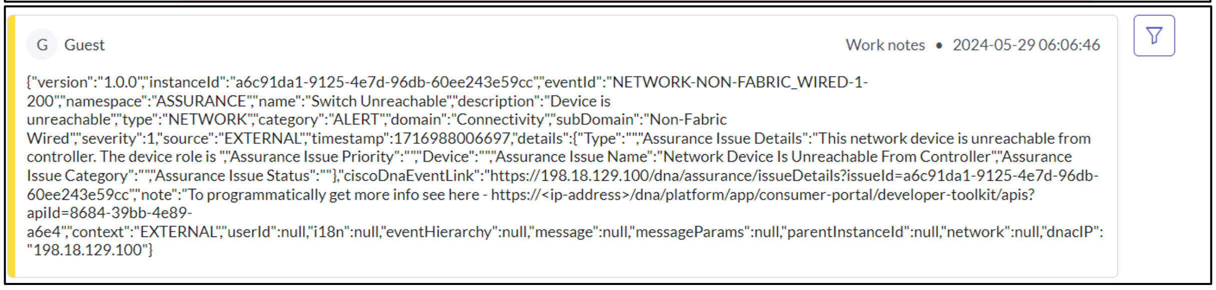
Number	Opened	Short description	Caller	Priority	State	Category	Assignment group	Assigned to	Updated
INC0010004	2024-05-29 06:06:46	Incident from DNAC alert	Guest	5 - Planning	New	Network	(empty)	(empty)	2024-05-29 06:06:46
INC0010003	2024-05-29 04:13:30	Incident from DNAC alert	Guest	5 - Planning	New	Network	(empty)	(empty)	2024-05-29 04:13:30

<u>Summary:</u>
When we dive into the incident, we can figure out the incident record (which includes Description and Work Notes) has been updated with the attributes mentioned in the script. The network administrator can directly access the device experiencing issues by using the link provided in the description.



Number	INC0010004	Channel	-- None --
* Caller	Guest	State	New
Category	Network	Impact	3 - Low
Subcategory	-- None --	Urgency	3 - Low
Service		Priority	5 - Planning
Service offering		Assignment group	
Configuration item		Assigned to	
* Short description	Incident from DNAC alert		

Description:
Name: Switch Unreachable
Description : Device is unreachable
DNAC Link : https://198.18.129.100/dna/assurance/issueDetails?issueId=a6c91da1-9125-4e7d-96db-60ee243e59cc


G Guest — Work notes • 2024-05-29 06:06:46

```
{"version":"1.0.0","instanceId":"a6c91da1-9125-4e7d-96db-60ee243e59cc","eventId":"NETWORK-NON-FABRIC_WIRED-1-
200","namespace":"ASSURANCE","name":"Switch Unreachable","description":"Device is
unreachable","type":"NETWORK","category":"ALERT","domain":"Connectivity","subDomain":"Non-Fabric
Wired","severity":1,"source":"EXTERNAL","timestamp":1716988006697,"details":{"Type":"","Assurance Issue Details":"This network device is unreachable from
controller. The device role is ","Assurance Issue Priority":"","Device":"","Assurance Issue Name":"Network Device Is Unreachable From Controller","Assurance
Issue Category":"","Assurance Issue Status":""},"ciscoDnaEventLink":"https://198.18.129.100/dna/assurance/issueDetails?issueId=a6c91da1-9125-4e7d-96db-
60ee243e59cc","note":"To programmatically get more info see here - https://<ip-address>/dna/platform/app/consumer-portal/developer-toolkit/apis?
apiId=8684-39bb-4e89-
a6e4","context":"EXTERNAL","userId":null,"i18n":null,"eventHierarchy":null,"message":null,"messageParams":null,"parentInstanceId":null,"network":null,"dnacIP":
"198.18.129.100"}
```

Notes:

ServiceNow (SNOW) trial subscription limitations would apply.
For Issue enrichment we can refer to Automated Mitigation Section.

3.2 Use Cases for Provisioning

3.2.1 Automated bulk upgrade of network devices based on Vulnerability Assessment for devices managed by multiple Catalyst-Center

Objective: The objective of this use case is to streamline and provision the process of upgrading network devices managed by multiple Catalyst Centers. This automation is driven by vulnerability assessments to ensure that all network devices are up-to-date with the recommended security patches and firmware versions as suggested by Catalyst Center. By achieving these objectives, organizations can enhance their network security, streamline device management, and ensure consistent and up-to-date protection against vulnerabilities across their entire network infrastructure.

API Referenced: **Token_Generation** , **Get_Site** , **Get_Membership** and **Get_Advisory_Per_Device**

Implementation:

Step1: Workflow for the Script: Security Advisory Retrieval -- Steps to Retrieves list of advisories on the network.

This script is designed to streamline the process of fetching security advisories for network devices managed by Cisco Catalyst Center. It reads credentials from an input CSV file, authenticates with the Catalyst Center API, retrieves site and device details, and compiles security advisory information into an output CSV file. This automation helps in efficiently managing and mitigating network security risks.
This workflow outlines the steps for retrieving and saving network security advisories from multiple Catalyst Centers using a Python script.

1. **Objective:** Automate the retrieval of network security advisories for devices within a specified site hierarchy.
2. **Setup and Imports:** Import necessary libraries: **'requests'**, **'csv'**, **'urllib3'**, **'HTTPBasicAuth'**, **'datetime'**, **'getpass'**, **'PrettyTable'**, and **'sys'**.
3. **Functions:**
    a. **get_X_auth_token(ip, uname, pword)**
        - **Purpose:** Obtain an authentication token from the Cisco Catalyst Center API.
        - **Inputs:** IP address of Catalyst Center, username, password.
        - **Process:**
            - Send a POST request to the authentication endpoint.
            - Handle connection errors and invalid responses.
            - Return the authentication token.

    b. find_id_by_hierarchy(data, hierarchy):
        - **Purpose:** Find the site ID based on the site hierarchy.
        - **Inputs:** Data from the site API response, site hierarchy input.
        - **Process:**
            - Convert the hierarchy to lowercase for case-insensitive matching.
            - Search for the hierarchy in the response data.
            - Return the corresponding site ID or exit if not found.

    c. **send_api_request(ip_address, tk):**

- **Purpose:** Retrieve device details and security advisories from Catalyst Center.
- **Inputs:** IP address of Catalyst Center, authentication token.
- **Process:**
  - Fetch site details.
  - Get user input for the site hierarchy.
  - Retrieve the site ID using **'find_id_by_hierarchy'**.
  - Fetch membership details of devices within the site.
  - Handle errors if site or device details cannot be fetched.
  - Retrieve and compile security advisory details for each device.
  - Store the compiled advisory details in a CSV file.

4. **Execution Flow:**
   a. **Disable Warnings:** Suppress SSL warnings using **'urllib3.disable_warnings'**.
   b. **Read Credentials from CSV:**
      - Open the **'input.csv'** file containing Catalyst Center IP, username, and password.
      - Iterate through each row to read credentials.
      - **For each set of credentials:**
        - Obtain the authentication token using **'get_X_auth_token'**.
        - Call **'send_api_request'** to process and fetch security advisories.
5. **Output:**
   - **CSV File:** Store the security advisory details in a file named **'Advisory_output.csv'**.
   - **Feedback:** Print messages indicating the progress and completion of tasks.

Step2: Workflow for the Script: SWIM -- Steps to perform Software Image Management on the devices on network (Onboarded on Catalyst Center).

This script automates the process of managing images on network devices via the Cisco Catalyst Center APIs. It facilitates listing available images, distributing selected images to specific devices, and activating the images on those devices. The script handles authentication, interacts with various API endpoints, tracks task progress, and provides user-friendly feedback through console messages and formatted tables.

API Referenced: Token_Generation , Import_Software_Image, Trigger_Software_Image_Distribution , Get_Device_List , Get_Task_by_ID and Trigger_Software_Image_Activation

Implementation:

1. **Objective:** Automate the process of listing, distributing, and activating images on network devices using the Cisco Catalyst Center API.
2. **Setup and Imports:** Import necessary libraries: **'requests'**, **'csv'**, **'urllib3'**, **'HTTPBasicAuth'**, **'datetime'**, **'getpass'**, **'PrettyTable'**, **'sys'**, **'json'**, **'time'** and **'tqdm'**.
3. **Functions:**
   a. **get_X_auth_token(ip, uname, pword)**
      - **Purpose:** Obtain an authentication token from the Cisco Catalyst Center API.
      - **Inputs:** IP address of Catalyst Center, username, password.
      - **Process:**
        - Send a POST request to the authentication endpoint.

- Handle connection errors and invalid responses.
- Return the authentication token.

b. **list_images(ip, tk):**
   - **Purpose:** List all available images on the Catalyst Center.
   - **Inputs:** IP address of Catalyst Center, authentication token.
   - **Process:**
     - Send a GET request to the image importation endpoint.
     - Parse the response and display image details in a formatted table.
     - Return a dictionary of image details for further use.

c. **distribute(ip, tk, imageUuid, deviceId):**

   - **Purpose:** Distribute an image to a specified device.
   - **Inputs:** IP address of Catalyst Center, authentication token, image UUID, device UUID.
   - **Process:**
     - Create a payload with device and image details.
     - Send a POST request to the image distribution endpoint.
     - Return the task ID for tracking distribution progress.

d. **get_deviceId(ip, tk, mgmtip):**

   - **Purpose:** Retrieve the device UUID using the device management IP address.
   - **Inputs:** IP address of Catalyst Center, authentication token, device management IP address.
   - **Process:**
     - Send a GET request to the network device endpoint with the management IP as a parameter.
     - Parse the response to extract and return the device UUID.

e. **get_task(taskid, token, host, counter):**

   - **Purpose:** Check the status of a task using its task ID.
   - **Inputs:** Task ID, authentication token, host IP address, counter for retries.
   - **Process:**
     - Send a GET request to the task endpoint.
     - Parse the response to check task progress.
     - Print the status and handle retries or completion.

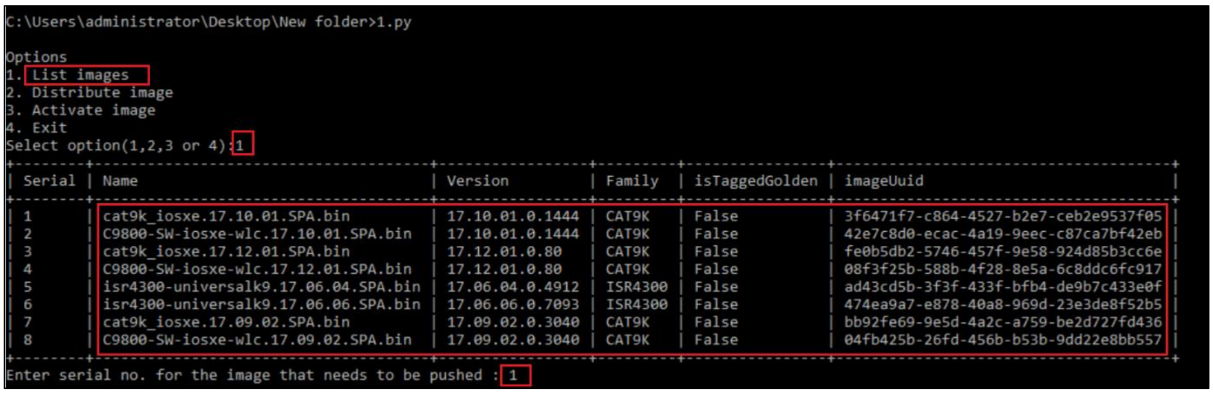
f. **activate(ip, tk, imageUuid, deviceId):**

   - **Purpose:** Activate an image on a specified device.
   - **Inputs:** IP address of Catalyst Center, authentication token, image UUID, device UUID.
   - **Process:**
     - Create a payload with activation details.
     - Send a POST request to the image activation endpoint.
     - Return the task ID for tracking activation progress.

4. **Execution Flow:**
   a. **Disable Warnings:** Suppress SSL warnings using **’urllib3.disable_warnings’**.

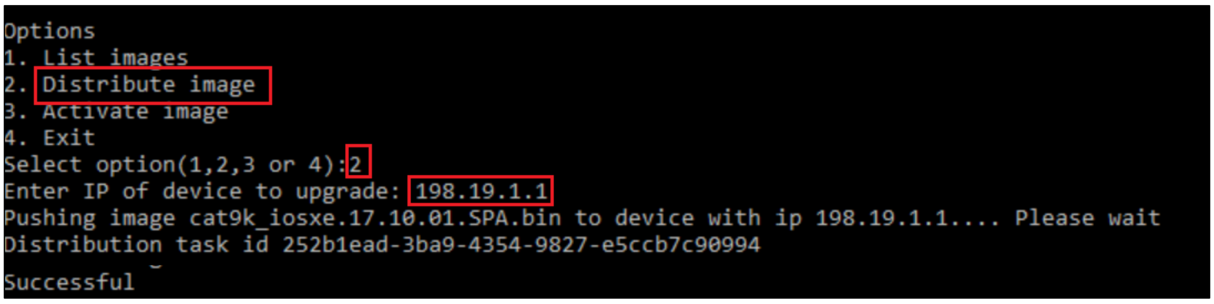
b. **Read Credentials from CSV:**
   - Open the **‘input.csv’** file containing Catalyst Center IP, username, and password.
   - Iterate through each row to read credentials.
   - **For each set of credentials:**
     - Obtain the authentication token using **‘get_X_auth_token’**.

c. **Main Loop:**
   - Present options to the user: list images, distribute image, activate image, or exit.
   - **Option 1 - List Images:**
     - Call **‘list_images’** to display available images.
     - Prompt the user to select an image by its serial number and store the image UUID and name.



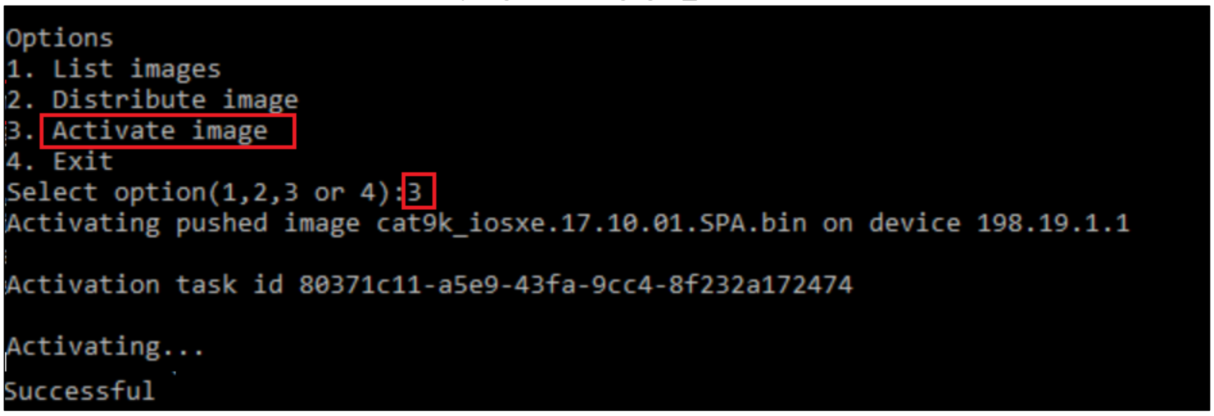
```
C:\Users\administrator\Desktop\New folder>1.py

Options
1. List images
2. Distribute image
3. Activate image
4. Exit
Select option(1,2,3 or 4):1
+--------+-------------------------------------+---------------+----------+----------------+--------------------------------------+
| Serial | Name                                | Version       | Family   | isTaggedGolden | imageUuid                            |
+--------+-------------------------------------+---------------+----------+----------------+--------------------------------------+
| 1      | cat9k_iosxe.17.10.01.SPA.bin        | 17.10.01.0.1444 | CAT9K  | False          | 3f6471f7-c864-4527-b2e7-ceb2e9537f05 |
| 2      | C9800-SW-iosxe-wlc.17.10.01.SPA.bin | 17.10.01.0.1444 | CAT9K  | False          | 42e7c8d0-ecac-4a19-9eec-c87ca7bf42eb |
| 3      | cat9k_iosxe.17.12.01.SPA.bin        | 17.12.01.0.80 | CAT9K    | False          | fe0b5db2-5746-457f-9e58-924d85b3cc6e |
| 4      | C9800-SW-iosxe-wlc.17.12.01.SPA.bin | 17.12.01.0.80 | CAT9K    | False          | 08f3f25b-588b-4f28-8e5a-6c8ddc6fc917 |
| 5      | isr4300-universalk9.17.06.04.SPA.bin | 17.06.04.0.4912 | ISR4300 | False        | ad43cd5b-3f3f-433f-bfb4-de9b7c433e0f |
| 6      | isr4300-universalk9.17.06.06.SPA.bin | 17.06.06.0.7093 | ISR4300 | False        | 474ea9a7-e878-40a8-969d-23e3de8f52b5 |
| 7      | cat9k_iosxe.17.09.02.SPA.bin        | 17.09.02.0.3040 | CAT9K  | False          | bb92fe69-9e5d-4a2c-a759-be2d727fd436 |
| 8      | C9800-SW-iosxe-wlc.17.09.02.SPA.bin | 17.09.02.0.3040 | CAT9K  | False          | 04fb425b-26fd-456b-b53b-9dd22e8bb557 |
+--------+-------------------------------------+---------------+----------+----------------+--------------------------------------+
Enter serial no. for the image that needs to be pushed :1
```

   - **Option 2 - Distribute Image:**
     - Prompt the user to enter the IP address of the device to upgrade.
     - Retrieve the device UUID using **‘get_deviceId’**.
     - Call **‘distribute’** to push the selected image to the device and get the distribution task ID.
     - Track the task progress using **‘get_task’**.



```
Options
1. List images
2. Distribute image
3. Activate image
4. Exit
Select option(1,2,3 or 4):2
Enter IP of device to upgrade: 198.19.1.1
Pushing image cat9k_iosxe.17.10.01.SPA.bin to device with ip 198.19.1.1.... Please wait
Distribution task id 252b1ead-3ba9-4354-9827-e5ccb7c90994
Successful
```

   - **Option 3 - Activate Image:**
     - Call **‘activate’** to activate the distributed image on the device and get the activation task ID.
     - Track the task progress using **‘get_task’.**



```
Options
1. List images
2. Distribute image
3. Activate image
4. Exit
Select option(1,2,3 or 4):3
Activating pushed image cat9k_iosxe.17.10.01.SPA.bin on device 198.19.1.1

Activation task id 80371c11-a5e9-43fa-9cc4-8f232a172474

Activating...
Successful
```

5. **Output:**
   - **Console Messages:** Print messages indicating the progress and completion of tasks, including task IDs and status updates.
   - **PrettyTable:** Display a formatted table of available images when listing images.

Notes:

For upgrades requiring sub-package activation, it is advisable to distribute the sub-package prior to deploying the main image. This step is necessary unless the sub-package is already stored in the device's flash memory.

The script runs for elongated timeframe depending on the size of the image which dictates the time it takes to transfer and activate the image

3.2.2 VLAN based switch port assignment and status retrieval for user devices

Objective: To provision the VLAN-based configuration on switch port and facilitate the assignment and updating of switch ports for user devices, ensuring accurate and efficient network configuration management while reducing manual effort and minimizing configuration errors.

API Referenced: Token_Generation , Get_Device_List , Gets_the_Templates_Available , Gets_all_the_Versions_of_a_Given_Template **,** Deploy_Template(v2) , Get_Task_by_ID **and** Status_of_Template_Deployment

Implementation:

Step1: Workflow for VLAN based Template Management Script

This script automates the deployment of VLAN configuration templates to network devices via the Cisco Catalyste Center API. It reads credentials and device management IPs from input files, handles authentication, interacts with various API endpoints to retrieve, and deploy templates, and provides user-friendly feedback through console messages and printed status updates.

1. **Objective:** Automate the process of deploying VLAN based configuration templates to network devices using Cisco Catalyst Center API.
2. **Setup and Imports:** Import necessary libraries: **'requests'**, **'csv'**, **'urllib3'**, **'HTTPBasicAuth'**, **'datetime'**, **'getpass'**, **'json'** , **'os'** and **'time'**.
3. **Functions:**
   a) **get_X_auth_token:**
      - **Purpose:** Obtain an authentication token from the Cisco Catalyst Center API.
      - **Inputs:** IP address of Catalyst Center, username, password.
      - **Process:**
        - Send a POST request to the authentication endpoint.
        - Handle connection errors and invalid responses.
        - Return the authentication token.

   b) **get_deviceId:**

- **Purpose:** Retrieve the device ID using the device's management IP address. **Inputs:** Catalyst Center IP address, authentication token, device management IP address.
- **Process:**
  - Send a GET request to the network device endpoint with the management IP as a parameter.
  - Return the device ID from the response.

c) **show_templates:**
- **Purpose:** Display available templates in the Cisco Catalyst Center.
- **Inputs:** Catalyst Center IP address, authentication token.
- **Process:**
  - Send a GET request to the template endpoint.
  - Print the list of available templates.

d) **get_template_id:**

- **Purpose:** Retrieve the template ID based on the project and template name.
- **Inputs:** Catalyst Center IP address, authentication token, template name (in "project/template" format).
- **Process:**
  - Send a GET request to the template endpoint.
  - Parse the response to find the template ID and version.
  - Return the template ID, version, and project details.

e) **print_template:**

- **Purpose:** Print the details of a specific template.
- **Inputs:** Catalyst Center IP address, authentication token, template ID
- **Process:**
  - Send a GET request to the template details endpoint.
  - Print the template name, product series and content.

f) **deploy:**

- **Purpose:** Deploy a template to a specified device.
- **Inputs:** Catalyst Center IP address, authentication token, template ID, device management IP address.
- **Process:**
  - Load deployment parameters from a JSON file (if exists).
  - Construct the payload with template ID, device ID, and parameters.
  - Send a POST request to the template deployment endpoint.
  - Return the deployment task ID.

g) **get_task:**

- **Purpose:** Retrieve the progress of a task using its task ID.
- **Inputs:** Task ID, authentication token, Catalyst Center IP address.
- **Process:**
  - Send a GET request to the task endpoint.
  - Return the task progress.

h) **status_template:**

- **Purpose:** Check the deployment status of a template.
- **Inputs:** Catalyst Center IP address, authentication token, deployment ID.
- **Process:**
    - Send a GET request to the deployment status endpoint.
    - Return the deployment status and detailed message.

4. **Execution Flow:**

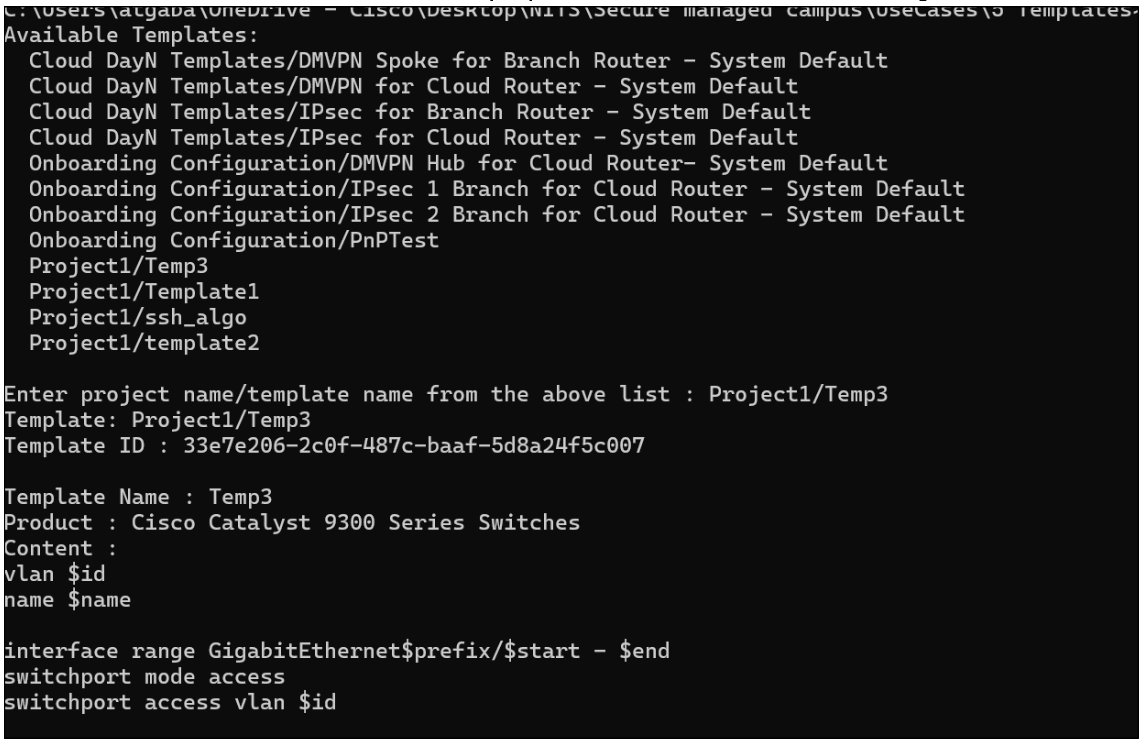
a) **Disable Warnings:** Suppress SSL warnings using **'urllib3.disable_warnings'**.

b) **Read Credentials from CSV:**

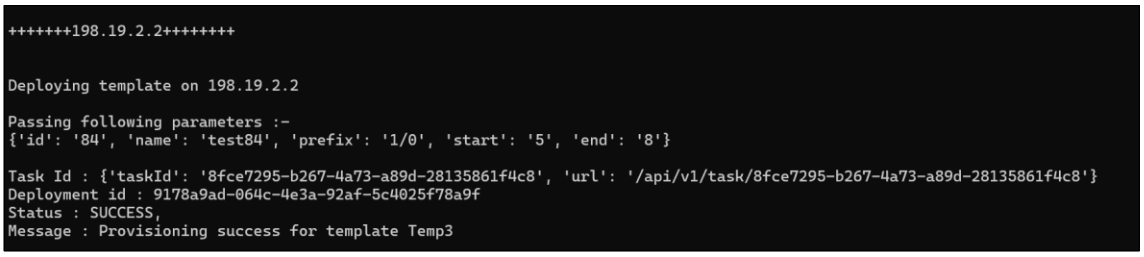
- Open the **'input.csv'** file containing Catalyst Center IP, username, and password.
- Iterate through each row to read credentials.
- **For each set of credentials:**
    - Obtain the authentication token using **'get_X_auth_token'**.
    - Display available templates using **'show_templates'.**
    - Prompt user to enter the desired template name.
    - Retrieve the template ID using **'get_template_id'**.
    - Print the template details using **'print_template'**.

c) **Deploy Template to Devices:**

- Open the **'devices.txt'** file containing device management IP addresses.
- **For each device:**
    - Retrieve the device ID using **'get_deviceId'**.
    - Deploy the template to the device using **'deploy'**.
    - Retrieve and print the deployment task ID.
    - Poll the task progress using **'get_task'** until the deployment ID is obtained.
    - Poll the deployment status using **'status_template'** until the deployment is successful, or a maximum number of attempts is reached.
    - Print the final deployment status and detailed message.



```
C:\Users\atgaba\OneDrive - Cisco\Desktop\NITS\Secure managed campus\UseCases\5 Templates
Available Templates:
  Cloud DayN Templates/DMVPN Spoke for Branch Router - System Default
  Cloud DayN Templates/DMVPN for Cloud Router - System Default
  Cloud DayN Templates/IPsec for Branch Router - System Default
  Cloud DayN Templates/IPsec for Cloud Router - System Default
  Onboarding Configuration/DMVPN Hub for Cloud Router- System Default
  Onboarding Configuration/IPsec 1 Branch for Cloud Router - System Default
  Onboarding Configuration/IPsec 2 Branch for Cloud Router - System Default
  Onboarding Configuration/PnPTest
  Project1/Temp3
  Project1/Template1
  Project1/ssh_algo
  Project1/template2

Enter project name/template name from the above list : Project1/Temp3
Template: Project1/Temp3
Template ID : 33e7e206-2c0f-487c-baaf-5d8a24f5c007

Template Name : Temp3
Product : Cisco Catalyst 9300 Series Switches
Content :
vlan $id
name $name

interface range GigabitEthernet$prefix/$start - $end
switchport mode access
switchport access vlan $id
```



```
+++++++198.19.2.2++++++++


Deploying template on 198.19.2.2

Passing following parameters :-
{'id': '84', 'name': 'test84', 'prefix': '1/0', 'start': '5', 'end': '8'}

Task Id : {'taskId': '8fce7295-b267-4a73-a89d-28135861f4c8', 'url': '/api/v1/task/8fce7295-b267-4a73-a89d-28135861f4c8'}
Deployment id : 9178a9ad-064c-4e3a-92af-5c4025f78a9f
Status : SUCCESS,
Message : Provisioning success for template Temp3
```

5. **Output:**
   - **Console Messages:** Print messages indicating the progress and completion of tasks, including task IDs, deployment IDs, and status updates.
   - **Template and Deployment Details**: Display available templates, template details, task progress, and deployment status.

3.2.3 Discovery of devices using API's (routers/switches/WLC's)

Objective: To automate the discovery of devices such as routers, switches, and wireless LAN controllers (WLCs) on Catalyst Center using APIs, enhancing network visibility, improving inventory management, and ensuring accurate device tracking and monitoring.

API Referenced: Token_Generation , Get_Credentials , Discover, Get_task, Get_devices

Implementation:

Step1: Workflow for Discovery Script

This script automates the process of discovery of network devices via the Cisco Catalyst Center API. It reads credentials for Catalyst Center and an input file that contains the discovery details namely the IP range, credentials to use. The script handles authentication, interacts with various API, tracks the discovery task, and provides feedback through console messages as to what all devices were discovered as part of the discovery.

1. **Objective:** Automate the discovery of devices using the Cisco Catalyst Center API.
2. **Setup and Imports:** Import necessary libraries: **'requests'**, **'csv'**, **'json'** and **'time'**
3. **Functions:**
   a. **get_X_auth_token(ip, uname, pword)**
      - **Purpose:** Obtain an authentication token from the Cisco Catalyst Center API.
      - **Inputs:** IP address of Catalyst Center, username, password.
      - **Process:**
        - Send a POST request to the authentication endpoint.
        - Return the authentication token.

   b. **get_credentials(ip,tk) & creds_to_id(ip,tk,creds,file)**
      - **Purpose:** Obtain the credentials id corresponding to the credentials mentioned in the input discovery file.
      - **Inputs:** IP address of Catalyst Center, token received from the function above and the file containing discovery details('discover.csv').
      - **Process:**
        - Send a GET request and retrieve all the credential ID already created on the Catalyst Center server – CLI, SNMP read and SNMP write.

        - Then read the input file – 'discover.csv' containing the discovery task details and map the credential names to UUIDs.

c. **discover(ip,tk,payload)**
    - **Purpose:** Start the discovery process using the input values.
    - **Inputs:** IP address of Catalyst Center, token, payload containing the details of the discovery task in the required format.
    - **Process:**
        - Create a payload with discovery details.
        - Send a POST request to start the discovery.
        - Return the response containing the task ID.

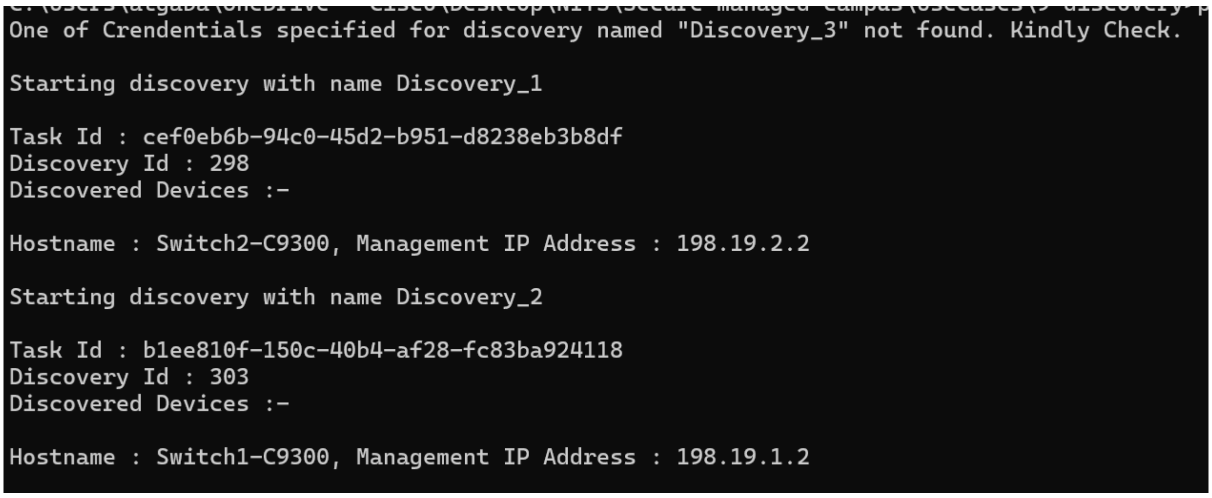
d. **get_task(taskid, token, host):**
    - **Purpose:** Check the status of a task using its task ID.
    - **Inputs:** Task ID, authentication token, host IP address.

    - **Process:**
        - Send a GET request to the task endpoint.
        - Return the task progress containing the discovery id.

e. **get_devices(ip,tk,disc_id)**

    - **Purpose:** Get the list of devices discovered by a discovery task.
    - **Inputs:** Catalyst IP, authentication token, discovery ID.
    - **Process:**
        - Send a GET request to the discovery endpoint.
        - Return the list of devices discovered.

4. **Execution Flow:**
    a. **Disable Warnings:** Suppress SSL warnings using **'urllib3.disable_warnings'**.
    b. **Read Credentials from CSV:**
        - Open the **'input.csv'** file containing Catalyst Center IP, username, and password.
        - Iterate through each row to read credentials.
        - **For each set of credentials:**
            - Obtain the authentication token using **'get_X_auth_token'**.
    c. **Read the parameters for discovery:**
        - Open the 'discover.csv' file containing the discovery details.
        - Read the credential names in the file and find corresponding credential ID that will be used to populate the payload in the discovery API.
        - Once the discovery is completed, post the status, and then poll the Catalyst Center for the network devices discovered in the discovery.
        - Print the device hostname and management IP corresponding to each discovery task.



```
C:\Users\atgaba\OneDrive - Cisco\Desktop\NITS\Secure managed campus\Usecases\3 discovery p
One of Crendentials specified for discovery named "Discovery_3" not found. Kindly Check.

Starting discovery with name Discovery_1

Task Id : cef0eb6b-94c0-45d2-b951-d8238eb3b8df
Discovery Id : 298
Discovered Devices :-

Hostname : Switch2-C9300, Management IP Address : 198.19.2.2

Starting discovery with name Discovery_2

Task Id : b1ee810f-150c-40b4-af28-fc83ba924118
Discovery Id : 303
Discovered Devices :-

Hostname : Switch1-C9300, Management IP Address : 198.19.1.2
```

5. **Output:**
    - **Console Messages:** Print messages indicating the progress and completion of tasks, including task IDs, discovery IDs, and discovered devices.
    - The script also specifies if it's unable to find a credential mentioned in the input file. That particular discovery task isn't performed since the credential id is a required field in the payload for the discovery API.

Notes:

The credentials names mentioned in the input discover.csv file should already be created on the Catalyst Center. If not, then that discovery task won't be processed.

3.3 Use Cases for Automation

3.3.1 Pushing customized template via API to devices spread across multiple sites (topology, inventory, template)

Objective: To automate the deployment of customized configuration templates to network devices distributed across multiple sites by leveraging APIs to manage topology, inventory, and template application, thereby ensuring consistent and efficient configuration management, and reducing manual intervention.

API Referenced: Token_Generation , Get_Device_List , Gets_the_Templates_Available , Gets_all_the_Versions_of_a_Given_Template , Deploy_Template(v2) and Status_of_Template_Deployment

Implementation:

Step1: Workflow for Network Device Template Management Script

This script automates the process of deploying templates on network devices via the Cisco Catalyst Center API. It reads credentials and device IP addresses from input files, lists available templates, allows the user to select and view a template, and deploys the template on specified devices. The script handles authentication interacts with various API endpoints, tracks task and deployment progress, and provides user-friendly feedback through console messages and prints template details.

a. **Objective:** Automate the deployment of templates on network devices using the Cisco Catalyst Center API.
b. **Setup and Imports:** Import necessary libraries: **‘requests’**, **‘csv’**, **‘urllib3’**, **‘HTTPBasicAuth’**, **‘datetime’**, **‘getpass’**, ‘**json’** and **‘time’**
c. **Functions:**
    a. **get_X_auth_token(ip, uname, pword)**
        - **Purpose:** Obtain an authentication token from the Cisco Catalyst Center API.
        - **Inputs:** IP address of Catalyst Center, username, password.
        - **Process:**
            - Send a POST request to the authentication endpoint.
            - Handle connection errors and invalid responses.
            - Return the authentication token.

    b. **get_deviceId(ip, tk, mgmtip):**
        - **Purpose:** Retrieve the device UUID using the device management IP address.
        - **Inputs:** IP address of Catalyst Center, authentication token, device management IP address.
        - **Process:**
            - Send a GET request to the network device endpoint with the management IP as a parameter.
            - Parse the response to extract and return the device UUID.

    c. **show_templates(ip, tk):**
        - **Purpose:** List all available templates on the Catalyst Center.
        - **Inputs:** IP address of Catalyst Center, authentication token.
        - **Process:**
            - Send a GET request to the template programmer endpoint.
            - Print the available templates in a formatted list.

    d. **get_template_id(ip, tk, temp_name):**

        - **Purpose:** Retrieve the template ID based on the project and template name.
        - **Inputs:** IP address of Catalyst Center, authentication token, template name.
        - **Process:**
            - Parse the project and template name.
            - Send a GET request to the template programmer endpoint.
            - Find and return the template ID and version.

    e. **print_template(ip, tk, templateId):**

        - **Purpose:** Display details of a specific template.
        - **Inputs:** IP address of Catalyst Center, authentication token, template ID.
        - **Process:**
            - Send a GET request to the specific template endpoint.
            - Print the template details including name, product series, content, and project name.

    f. **deploy(ip, tk, tmp_id, device):**

- **Purpose:** Deploy a template on a specified device.
- **Inputs:** IP address of Catalyst Center, authentication token, template ID, device UUID.
- **Process:**
    - Create a payload with deployment details.
    - Send a POST request to the template deploy endpoint.
    - Return the response containing the task ID.

g. **get_task(taskid, token, host):**

- **Purpose:** Check the status of a task using its task ID.
- **Inputs:** Task ID, authentication token, host IP address.
- **Process:**
    - Send a GET request to the task endpoint.
    - Return the task progress.

h. **status_template(host, tk, deploymentId):**

- **Purpose:** Check the status of a template deployment using its deployment ID.
- **Inputs:** Deployment ID, authentication token, host IP address.
- **Process:**
    - Send a GET request to the deployment status endpoint.
    - Return the deployment status and detailed message.

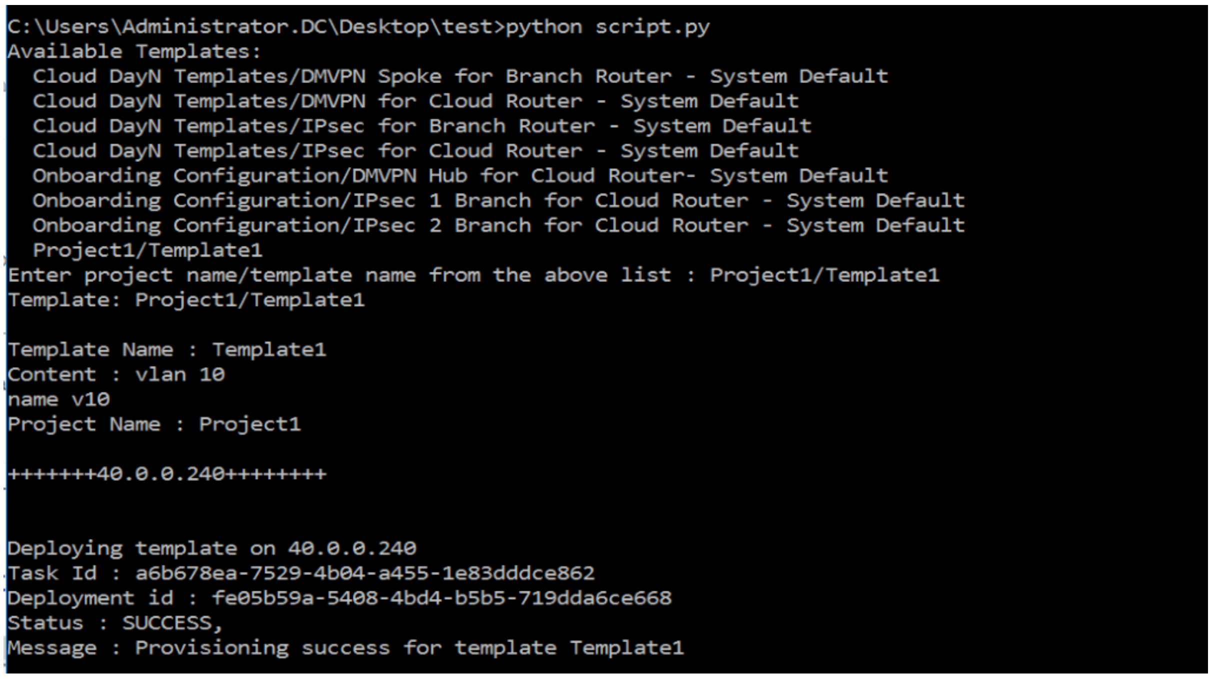
d. Execution Flow:

a. **Disable Warnings:** Suppress SSL warnings using **’urllib3.disable_warnings’**.

b. **Read Credentials from CSV:**
- Open the **‘input.csv’** file containing Catalyst Center IP, username, and password.
- Iterate through each row to read credentials.
- **For each set of credentials:**
    - Obtain the authentication token using **‘get_X_auth_token’**.
    - Call **’show_templates’** to display available templates.

c. **User Input for Template Selection:**
- Prompt the user to input the project name and template name.
- Retrieve the template ID using **‘get_template_id’**.
- Print the template details using **‘print_template’**.

d. **Read Device IPs from File:**
- Open the **‘devices.txt’** file and read device IP addresses.
- Iterate through each IP address to perform the following actions:
    - Retrieve the device UUID using **‘get_deviceId’**.
    - Deploy the template using **‘deploy’** and get the task ID.
    - Track the task progress using **‘get_task’** to obtain the deployment ID.
    - Track the deployment status using **‘status_template’**.



```
C:\Users\Administrator.DC\Desktop\test>python script.py
Available Templates:
  Cloud DayN Templates/DMVPN Spoke for Branch Router - System Default
  Cloud DayN Templates/DMVPN for Cloud Router - System Default
  Cloud DayN Templates/IPsec for Branch Router - System Default
  Cloud DayN Templates/IPsec for Cloud Router - System Default
  Onboarding Configuration/DMVPN Hub for Cloud Router- System Default
  Onboarding Configuration/IPsec 1 Branch for Cloud Router - System Default
  Onboarding Configuration/IPsec 2 Branch for Cloud Router - System Default
  Project1/Template1
Enter project name/template name from the above list : Project1/Template1
Template: Project1/Template1

Template Name : Template1
Content : vlan 10
name v10
Project Name : Project1

+++++++40.0.0.240++++++++


Deploying template on 40.0.0.240
Task Id : a6b678ea-7529-4b04-a455-1e83dddce862
Deployment id : fe05b59a-5408-4bd4-b5b5-719dda6ce668
Status : SUCCESS,
Message : Provisioning success for template Template1
```

6. **Output:**
   - **Console Messages:** Print messages indicating the progress and completion of tasks, including task IDs, deployment IDs, and status updates.
   - **Template Details:** Display the selected template details before deployment.

Notes:

The template should already be created on the Catalyst Center. The script is not creating any templates but pushing them to the intended devices in bulk. For creating templates, kindly refer to the following guide:

https://www.cisco.com/c/en/us/td/docs/cloud-systems-management/network-automation-and-management/dna-center/2-3-5/user_guide/b_cisco_dna_center_ug_2_3_5/b_cisco_dna_center_ug_2_3_5_chapter_01000.html#id_74791

3.2.2 Network setting automation and credential management across multiple Catalyst Centers

Objective: To automate network settings and manage credentials across multiple Catalyst Centres, enhancing operational efficiency, ensuring consistent configurations, and improving security by reducing manual intervention and potential errors in network management.

API Referenced: Token_Generation , Assign_Device_Credential_to_Site(v2) , Credentials_Guide ,Get_Task_by_ID ,Get_Site and Get_Business_API_Execution_Details

Implementation:

Step1: Workflow for Network Device Credential Management Script

This script automates the process of creating and deploying network credentials (CLI, SNMP Read, SNMP Write) via the Cisco Catalyst Center API. It reads credentials from an input file, handles authentication, interacts with various API endpoints, tracks task and execution progress, and provides user-friendly feedback through console messages and printed status updates.

1. **Objective:** Automate the creation and deployment of network credentials (CLI, SNMP Read, SNMP Write) on network devices using the Cisco Catalyst Center API.
2. **Setup and Imports:** Import necessary libraries: **'requests'**, **'csv'**, **'urllib3'**, **'HTTPBasicAuth'**, '**json'** and , **'sys'**.
3. **Functions:**
    a) **get_X_auth_token(ip, uname, pword)**
    - **Purpose:** Obtain an authentication token from the Cisco Catalyst Center API.
    - **Inputs:** IP address of Catalyst Center, username, password.
    - **Process:**
        - Send a POST request to the authentication endpoint.
        - Handle connection errors and invalid responses.
        - Return the authentication token.

    b) **find_id_by_hierarchy:**
    - **Purpose:** Find the site ID based on the site hierarchy name.
    - **Inputs:** Site data, hierarchy name.
    - **Process:**
        - Convert hierarchy to lowercase.
        - Search through site data to find matching hierarchy.
        - Return the site ID or exit if not found.

    c) **create_cred:**
    - **Purpose:** Create CLI, SNMP Read, and SNMP Write credentials.
    - **Inputs:** Catalyst Center IP address, authentication token, input file containing credential details.
    - **Process:**
        - Read the input file for credential details.
        - Construct payloads for each credential type.
        - Send POST requests to create each credential.
        - Return task IDs for each credential creation.

    d) **get_task:**
    - **Purpose:** Retrieve the progress of a task using its task ID.
    - **Inputs:** Task ID, Catalyst Center IP address, authentication token.
    - **Process:**
        - Send a GET request to the task endpoint.
        - Return the task progress.

    e) **get_site:**
    - **Purpose:** Retrieve the site ID based on user input of site hierarchy.
    - **Inputs:** Catalyst Center IP address, authentication token.
    - **Process:**
        - Send a GET request to the site endpoint.

        - Prompt user for site hierarchy input.
        - Use **‘find_id_by_hierarchy’** to get the site ID.
        - Return the site ID.

f) **push_creds:**
    - **Purpose:** Apply the created credentials to the specified site.
    - **Inputs:** Catalyst Center IP address, authentication token, site ID, CLI credential ID, SNMP Read credential ID, SNMP Write credential ID.
    - **Process:**
        - Construct payload with credential IDs.
        - Send a POST request to the credential-to-site endpoint.
        - Return the response data.

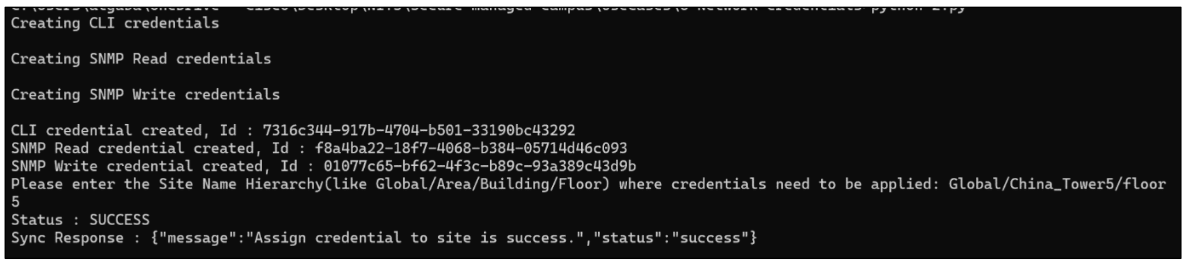
g) **get_execution_status:**
    - **Purpose:** Retrieve the execution status of the credential deployment.
    - **Inputs:** Execution ID, Catalyst Center IP address, authentication token.
    - **Process:**
        - Send a GET request to the execution status endpoint.
        - Return the response data.

4. **Execution Flow:**

a) **Disable Warnings:** Suppress SSL warnings using **’urllib3.disable_warnings’**.

b) **Read Credentials from CSV:**
    - Open the **‘input.csv’** file containing Catalyst Center IP, username, and password.
    - Iterate through each row to read credentials.
    - **For each set of credentials:**
        - Obtain the authentication token using **‘get_X_auth_token’**.
        - Create CLI, SNMP Read, and SNMP Write credentials using **'create_cred'**.
        - Track the progress of each credential creation using **'get_task'**.

c) **Check Credential Creation Status:**
    - Print the status of each credential creation.
    - If all credentials are successfully created:
        - Retrieve the site ID using **'get_site'**.
        - Apply the credentials to the site using **'push_creds'**.
        - Track the execution status using **'get_execution_status'**.
        - Print the final status and sync response.
    - If any credential creation fails, print an error message.



```
Creating CLI credentials

Creating SNMP Read credentials

Creating SNMP Write credentials

CLI credential created, Id : 7316c344-917b-4704-b501-33190bc43292
SNMP Read credential created, Id : f8a4ba22-18f7-4068-b384-05714d46c093
SNMP Write credential created, Id : 01077c65-bf62-4f3c-b89c-93a389c43d9b
Please enter the Site Name Hierarchy(like Global/Area/Building/Floor) where credentials need to be applied: Global/China_Tower5/floor
5
Status : SUCCESS
Sync Response : {"message":"Assign credential to site is success.","status":"success"}
```

5. **Output:**
    - **Console Messages:** Print messages indicating the progress and completion of tasks, including task IDs, execution IDs, and status updates.
    - **Credential Creation Status:** Display the status of CLI, SNMP Read, and SNMP Write credential creation.

- **Deployment Status:** Display the execution status and sync response after applying credentials to the site.

3.3.3 Catalyst Center Backup status check

Objective: To automate the monitoring and verification of backup status in Catalyst Centers, ensuring reliable data protection and recovery processes while reducing manual oversight and minimizing the risk of data loss.

API Referenced: Token_Generation and Catalyst_Centre_Backup

Implementation:

Step1: Workflow for Backup Details Retrieval and Reporting Script

This script automates the retrieval and display of status of backup details from Cisco Catalyst Center using its API. It reads credentials from a CSV file, fetches backup information, processes the data, and outputs the details in a formatted table both to the console and a text file.

1. **Objective:** Automate the process of fetching and displaying backup details from Cisco Catalyst Center using API calls.

2. **Setup and Imports:**
   a. Import necessary libraries: **'requests'**, **'HTTPBasicAuth'**, '**json'**, **'csv'**, **'time'**, **'PrettyTable'**, **'datetime'**

3. **Functions:**
   a. **get_X_auth_token(ip, uname, pword)**
      - **Purpose:** Obtain an authentication token from the Cisco Catalyst Center API.
      - **Inputs:** IP address of Catalyst Center, username, password.
      - **Process:**
        - Construct the URL for the authentication endpoint.
        - Send a POST request to the authentication endpoint.
        - Handle connection errors and invalid responses.
        - Return the authentication token.

   b. **get_backups:**
      - **Purpose:** Retrieve status of backup details from Cisco Catalyst Center.
      - **Inputs:** Catalyst Center IP address, authentication token.
      - **Process:**
        - Construct the URL for the backup details endpoint.
        - Send a GET request to retrieve backup details.
        - Handle unsuccessful responses.
        - Return the response data containing backup details.

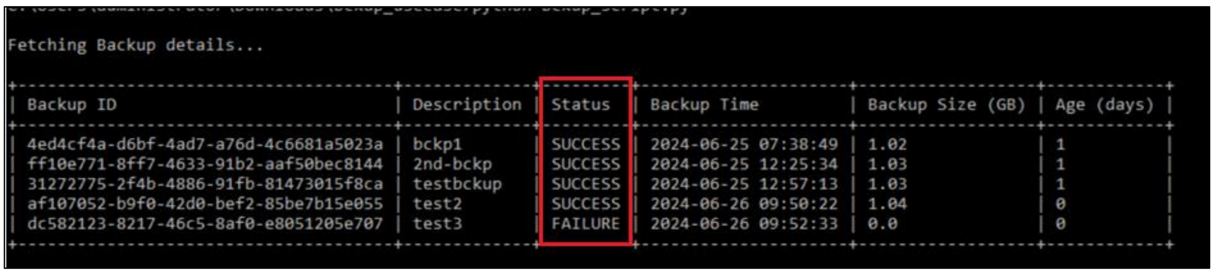
4. **Execution Flow:**
   a. **Disable Warnings:** Suppress SSL warnings using **'urllib3.disable_warnings'**.
   b. **Read Credentials from CSV:**
      - Open the **'input.csv'** file containing Catalyst Center IP, username, and password.
      - Iterate through each row to read credentials.

- **For each set of credentials:**
    - Obtain the authentication token using **‘get_X_auth_token’**.

c. **Fetch Backup Details:**
- Use the authentication token to fetch backup details from Cisco Catalyst Center using **'get_backups'**.

d. **Process and Display Backup Details:**
- Initialize a **‘PrettyTable’** object to format the backup details.
- Iterate through the backup details:
    - Extract relevant information: backup ID, description, status, compatibility, backup time, backup size, and age.
    - Convert timestamps and calculate the age of backups.
    - Add the details to the table.
- Print the formatted table to the console.

e. **Save Backup Details to File:**
- Open a file named **'backup_status.txt'** for writing.
- Write the formatted table to the file.
- Close the file.



```
Fetching Backup details...

+--------------------------------------+-------------+---------+---------------------+------------------+------------+
| Backup ID                            | Description | Status  | Backup Time         | Backup Size (GB) | Age (days) |
+--------------------------------------+-------------+---------+---------------------+------------------+------------+
| 4ed4cf4a-d6bf-4ad7-a76d-4c6681a5023a | bckp1       | SUCCESS | 2024-06-25 07:38:49 | 1.02             | 1          |
| ff10e771-8ff7-4633-91b2-aaf50bec8144 | 2nd-bckp    | SUCCESS | 2024-06-25 12:25:34 | 1.03             | 1          |
| 31272775-2f4b-4886-91fb-81473015f8ca | testbckup   | SUCCESS | 2024-06-25 12:57:13 | 1.03             | 1          |
| af107052-b9f0-42d0-bef2-85be7b15e055 | test2       | SUCCESS | 2024-06-26 09:50:22 | 1.04             | 0          |
| dc582123-8217-46c5-8af0-e8051205e707 | test3       | FAILURE | 2024-06-26 09:52:33 | 0.0              | 0          |
+--------------------------------------+-------------+---------+---------------------+------------------+------------+
```

5. **Output:**
    - **Console Messages:** Display a table of backup details, formatted using **'PrettyTable'**.
    - **File Output:** Save the table of backup details to **'backup_status.txt'**.

Notes:

For configuring backup destinations and scheduling backups kindly refer to the following guide:

https://www.cisco.com/c/en/us/td/docs/cloud-systems-management/network-automation-and-management/dna-center/2-2-3/admin_guide/b_cisco_dna_center_admin_guide_2_2_3/b_cisco_dna_center_admin_guide_2_2_3_chapter_0110.html#id_82736

3.3.4 TrustSec configuration verification

Objective: This use case aims at verification of trustsec configuration that is pushed by Catalyst Center to the edge devices. This serves as a tool to troubleshoot and check the configuration if missed due to some technical glitch during device provisioning by Catalyst Center or some network issue during the configuration push. This use case can be expanded to run other commands on the edge devices using the same commandRunner API.

API Referenced: Token Generation , Command Runner

Workflow:

Step1: Workflow for the device list and command runner Script

The script is split in two steps. The first script "**devices_by_site.py**" is run to get a list of all devices at a site. The site hierarchy is provided as an input and the output of the script, containing the device details is stored in the file "**device_list.txt**". This file can be modified and only the devices that need to be check can be left in there. The second script "**script.py**" reads the list of devices from the step above and commands from a file (**'commands_to_run.txt'**). For each command in the file, it initiates command execution on the network devices in the device list.

1. **Imports:**
   a. Import necessary libraries: **'requests'**, **'HTTPBasicAuth'**, 'j**son'**, **'csv'**, **'time'**, **'PrettyTable'**, **'datetime'**

2. **Functions:**
   a. **get_X_auth_token(ip, uname, pword)**
      - **Purpose:** Obtain an authentication token from the Cisco Catalyst Center API.
      - **Inputs:** IP address of Catalyst Center, username, password.
      - **Process:**
        - Construct the URL for the authentication endpoint.
        - Send a POST request to the authentication endpoint.
        - Handle connection errors and invalid responses.
        - Return the authentication token.

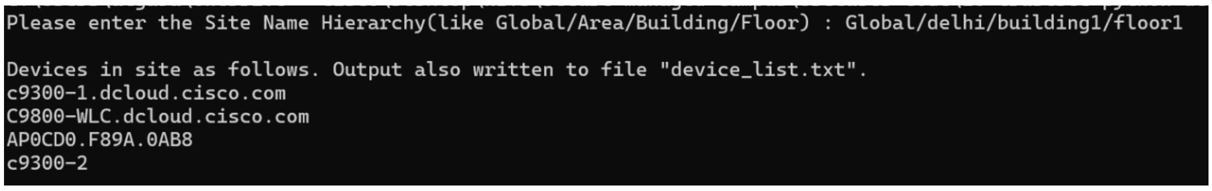
   b. **cmd_runner(token, host, device_id, ios_cmd)**
      - **Purpose:** Execute a command on a specific device
      - **Inputs:** Catalyst Center IP address, authentication token, deviceId and the command to run
      - **Process:**
        - Send a POST request to send the desired commands to the device
        - Handle unsuccessful responses.
        - Return the response data containing taskId.

   c. **get_task(taskid, token, host):**
      - **Purpose:** Check the status of a task using its task ID.
      - **Inputs:** Task ID, authentication token, host IP address.
      - **Process:**
        - Send a GET request to the task endpoint.
        - Return the task progress.

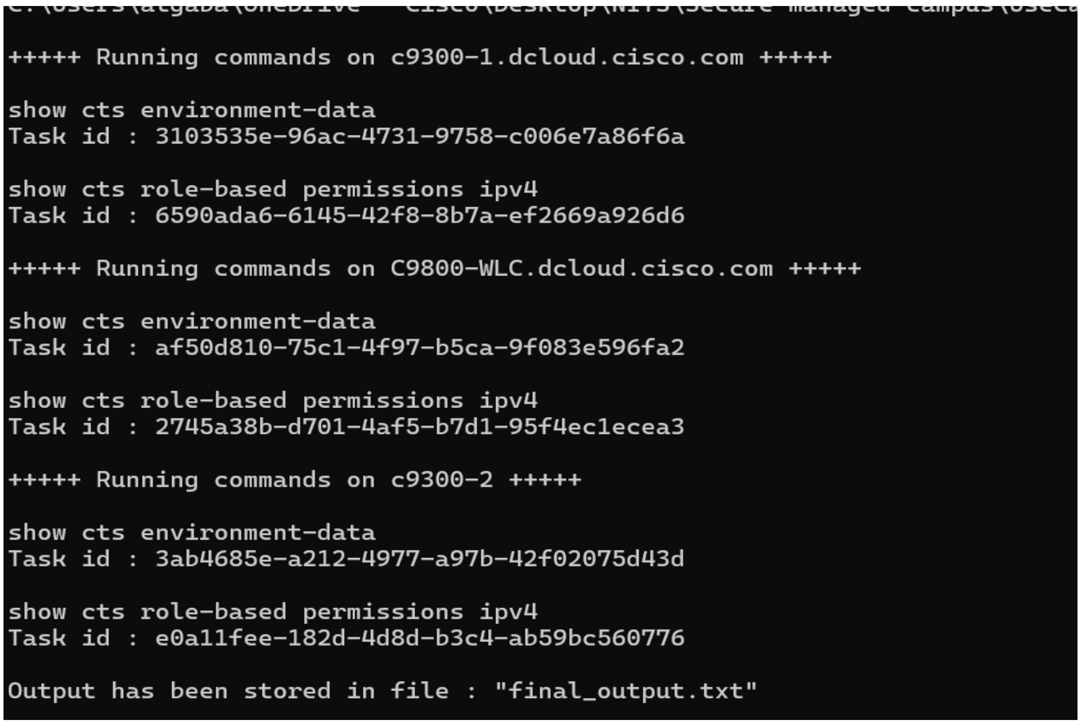
3. **Execution Flow:**
   a. **Disable Warnings:** Suppress SSL warnings using **'urllib3.disable_warnings'**.
   b. **Read Credentials from CSV:**
      - Open the **'input.csv'** file containing Catalyst Center IP, username, and password.
      - Iterate through each row to read credentials.
      - **For each set of credentials:**
        - Obtain the authentication token using **'get_X_auth_token'**.

c. **Device Details:**
   - Run the script devices_by_site.py to get a list of all devices and corresponding details.
   - The output of the device script produces a file that is ingested in the command runner script.

d. **Command execution**:
   - The command runner script reads the above device detail file and a file containing the commands to run and executes the commands on each device and displays/saves the output to a file.



```
Please enter the Site Name Hierarchy(like Global/Area/Building/Floor) : Global/delhi/building1/floor1

Devices in site as follows. Output also written to file "device_list.txt".
c9300-1.dcloud.cisco.com
C9800-WLC.dcloud.cisco.com
AP0CD0.F89A.0AB8
c9300-2
```



```
+++++ Running commands on c9300-1.dcloud.cisco.com +++++

show cts environment-data
Task id : 3103535e-96ac-4731-9758-c006e7a86f6a

show cts role-based permissions ipv4
Task id : 6590ada6-6145-42f8-8b7a-ef2669a926d6

+++++ Running commands on C9800-WLC.dcloud.cisco.com +++++

show cts environment-data
Task id : af50d810-75c1-4f97-b5ca-9f083e596fa2

show cts role-based permissions ipv4
Task id : 2745a38b-d701-4af5-b7d1-95f4ec1ecea3

+++++ Running commands on c9300-2 +++++

show cts environment-data
Task id : 3ab4685e-a212-4977-a97b-42f02075d43d

show cts role-based permissions ipv4
Task id : e0a11fee-182d-4d8d-b3c4-ab59bc560776

Output has been stored in file : "final_output.txt"
```

4. **Output:**
   - **Console Messages:** Display the details of the devices and corresponding commands being run along with the TaskId of each command run.
   - **File Output:** Saves a table consisting of the output of the various commands to a file **'final_output.txt'**.

3.4 Use Cases for Assurance

3.4.1 Network Health Monitoring & Reporting for Catalyst-Center

Objective: The objective of this use case is to implement a comprehensive network health monitoring and reporting system for Catalyst Center. This system aims to provide the Managed Services Provider (MSP) with real-time visibility into the performance, availability, and overall health of the network infrastructure. By leveraging advanced monitoring and reporting tools, the MSP can proactively manage network performance, quickly identify, resolve issues, and deliver detailed reports to customers, thereby enhancing service quality and customer satisfaction.

API Referenced: **Token_Generation** and **System_Performance_API**

Implementation:
This workflow outlines the steps for retrieving performance data from multiple catalyst centres and saving the data to CSV files.

Step1: Prerequisites

1. **Python Installation:** Ensure Python is installed on your machine. You may refer to Use Case 3.1.1 Step2. Install the necessary Python library by running the following command:

```
pip install requests
```

2. **Input File Preparation:** Create a CSV file named input.csv containing the IP addresses, usernames, and passwords for each catalyst center. The file should have the following format:

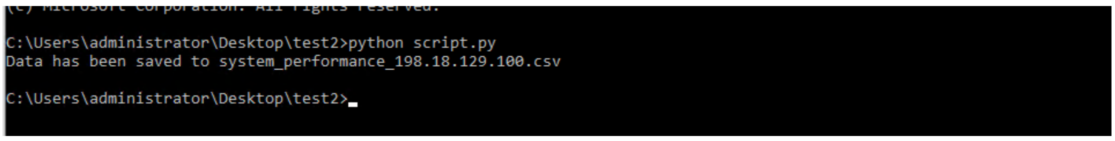
```
dnac_ip, username, password
<ip-address1>,<username1>,<password1>
<ip-address2>,<username2>,<password2>
```

Step2: Workflow for the Script: Retrieving Performance Data for Catalyst Center

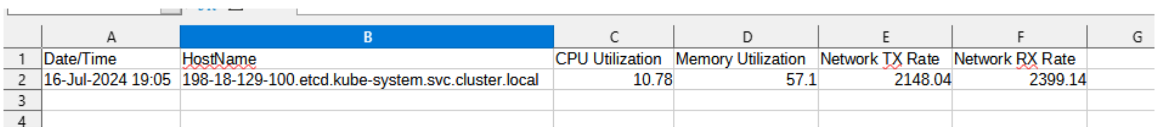
1. **Environment Setup:** Disable SSL warnings to avoid issues with secure requests.
2. **Read Input Data:** Open and read the **'input.csv'** file to obtain the credentials for each catalyst center.
3. **Authenticate and Obtain Token:** For each catalyst center, send a POST request to the authentication endpoint using the provided credentials to obtain an authentication token.
4. **Retrieve Performance Data:** Using the authentication token, send a GET request to the performance data endpoint of the catalyst center.
5. **Save Data to CSV:** Format the retrieved performance data and save it to a CSV file named **'system_performance_<ip_address>.csv'**, where **'<ip_address>'** is the IP address of the catalyst center.
6. **Error Handling:** Include error handling for connection errors and failed API responses to ensure the script continues running for other catalyst centres even if one fails.
7. **Execution:** Execute the script, which will process each entry in the **'input.csv'** file, authenticate, retrieve performance data, and save the data to the respective CSV files.

8. **Output:** The output for each catalyst center will be a CSV file containing the performance data. The file will be named **’system_performance_<ip_address>.csv’** and will include the following details:

- Date/Time of the data retrieval.
- Hostname of the catalyst center
- CPU utilization
- Memory utilization
- Network transmission rate.
- Network reception rate.



```
(c) Microsoft Corporation. All rights reserved.

C:\Users\administrator\Desktop\test2>python script.py
Data has been saved to system_performance_198.18.129.100.csv

C:\Users\administrator\Desktop\test2>_
```



	A	B	C	D	E	F	G
1	Date/Time	HostName	CPU Utilization	Memory Utilization	Network TX Rate	Network RX Rate	
2	16-Jul-2024 19:05	198-18-129-100.etcd.kube-system.svc.cluster.local	10.78	57.1	2148.04	2399.14	
3							
4							

By following this workflow, we can efficiently manage and retrieve performance data from multiple catalyst center, ensuring that each center's data is securely authenticated, retrieved, and saved for further analysis.

Step3: Creating a Generic Webhook for Catalyst Center

For creating a generic webhook for health subscriptions in Cisco Catalyst Center we can refer **Event configurations through Webhooks** section

Notes:

To generate an alert for 80% threshold utilization, a mechanism should be implemented to continuously poll the Catalyst Center. This mechanism will trigger an email or alert in case of overutilization in the Catalyst Center.

3.4.2 Network Health Monitoring & Reporting for Network Devices Managed by Catalyst Center

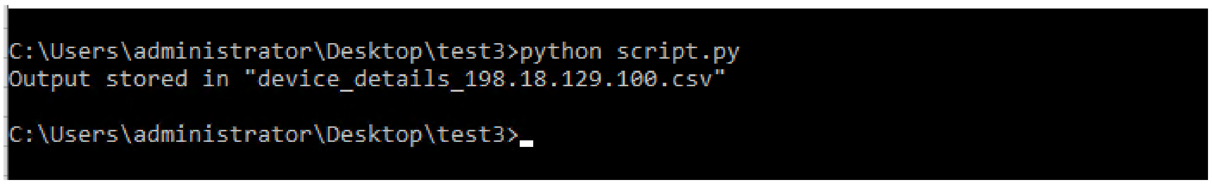
Objective: The objective of this use case is to implement a comprehensive network health monitoring and reporting system for network devices associated with Catalyst Center. This system aims to provide the Managed Services Provider (MSP) with real-time visibility into the performance, availability, and overall health of the network infrastructure. By leveraging advanced monitoring and reporting tools, the MSP can proactively manage network performance, quickly identify, resolve issues, and deliver detailed reports to customers, thereby enhancing service quality and customer satisfaction.

API Referenced: **Token_Generation** **,** **Get_Device_List** and **Get_Device_Detail**

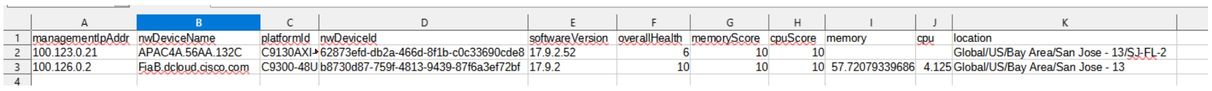
Implementation:
This workflow outlines the steps for the retrieval of device performance details from Cisco Catalyst Center and save the data into a CSV file.

Step1: Workflow for the Script: Retrieving Performance Data for Devices Onboarded on Catalyst Center

1. **Preparation:** Ensure the necessary Python libraries are installed: **'requests'**, **'csv'**, **'sys'**, and **'urllib3'**. Disable warnings for insecure HTTP requests to handle self-signed certificates.
2. **Authentication:** Create a function to send a POST request to the Cisco Catalyst Center API to obtain an authentication token. Handle potential connection errors and exit gracefully if an error occurs.
3. **Retrieve Network Devices:** Create a function to send a GET request to the Cisco Catalyst Center API to retrieve a list of network devices. Extract the device IDs and MAC addresses from the response.
4. **Write Device Details to CSV:** Open a new CSV file for writing. Define the headers for the CSV file, including fields such as management IP address, device name, platform ID, software version, health scores, and location. Write the headers to the CSV file.
5. **Retrieve Detailed Information for Each Device:** For each device, send a GET request to retrieve detailed information using the device ID. Extract the relevant details and write them to the CSV file. Handle potential errors for each API request and log any issues encountered.
6. **Process Input Data:** Read an input CSV file containing the IP address, username, and password for each Cisco Catalyst Center instance. For each entry in the input CSV, retrieve the authentication token and send the API request to retrieve device details.
7. **Output:** Generate a CSV file named **'device_details_<ip_address>.csv'** for each Cisco Catalyst Center instance specified in the input CSV file. The CSV file contains detailed information about the devices managed by the Cisco Catalyst Center instance.
8. **Execution:** Run the script. The script will read the input CSV, authenticate with each Cisco Catalyst Center instance, retrieve device details, and save the details in a CSV file for each instance.
9. **Output:** CSV files containing detailed information about the devices managed by each Cisco Catalyst Center instance specified in the input CSV file. Each CSV file is named based on the corresponding IP address of the Cisco Catalyst Center instance. If certain attribute is not received for a certain type of device in the response, then that column is left blank.



```
C:\Users\administrator\Desktop\test3>python script.py
Output stored in "device_details_198.18.129.100.csv"

C:\Users\administrator\Desktop\test3>
```



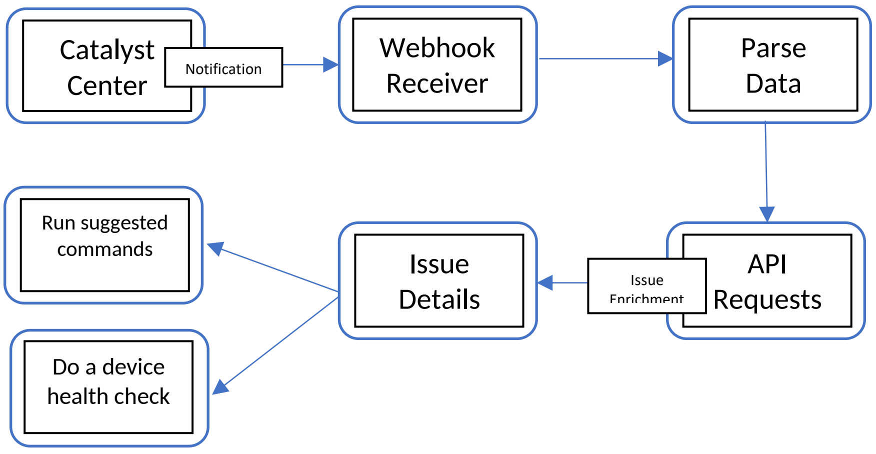
	A	B	C	D	E	F	G	H	I	J	K
1	managementIpAddr	nwDeviceName	platformId	nwDeviceId	softwareVersion	overallHealth	memoryScore	cpuScore	memory	cpu	location
2	100.123.0.21	APAC4A.56AA.132C	C9130AXI	62873efd-db2a-466d-8f1b-c0c33690cde8	17.9.2.52	6	10	10			Global/US/Bay Area/San Jose - 13/SJ-FL-2
3	100.126.0.2	FiaB.dcloud.cisco.com	C9300-48U	b8730d87-759f-4813-9439-87f6a3ef72bf	17.9.2	10	10	10	57.72079339686	4.125	Global/US/Bay Area/San Jose - 13
4											

Notes:-

To generate an alert for 80% threshold utilization, a mechanism should be implemented to continuously poll the devices onboarded on Catalyst Center. This mechanism will trigger an email or alert in case of overutilization of devices.

3.4.3 Wireless Assurance – Automated handling of Catalyst center event notifications

Objective: The objective of this use case is to implement an end-to-end workflow for handling event notifications for wireless assurance. The end-to-end workflow involves configuring the Catalyst Center to send webhook notifications to a receiver application. The receiver parses the data received and then makes additional API requests against the Catalyst Center to get more information about the issue and present the suggested actions for troubleshooting.



API Referenced: **Token_Generation, Get_Device_Detail, Issue details, Command_Runner**

Implementation:

Step1: Creating SSL certificate and key for the webhook receiver

For receiving webhook notifications from the Catalyst Center, we'll be using a Flask app. Flask is a micro web framework for Python applications. The Catalyst Center can only have 'https' protocol as a destination for webhooks so we need an SSL certificate and a private key.

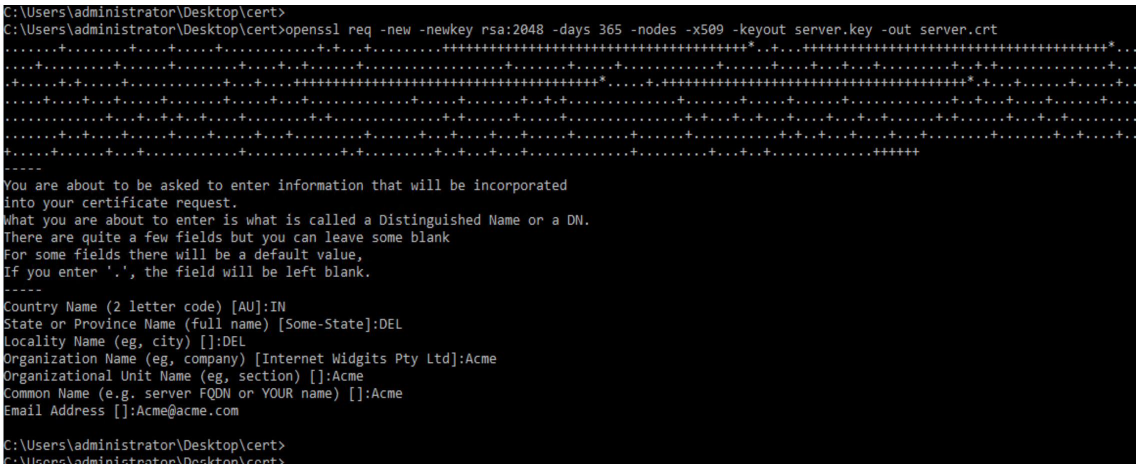
Generating an SSL Certificate and Private Key

Step 1: Download OpenSSL Visit the OpenSSL website and download the version compatible with your operating system (e.g., Windows).

Step 2: Open the OpenSSL Command Prompt After installing OpenSSL, you'll have access to the "OpenSSL Win64" program, which opens a command prompt tailored for OpenSSL operations.

Step 3: Generate SSL Certificate and Private Key in the OpenSSL command prompt, execute the following command to create a self-signed SSL certificate and a private key. Follow the prompts to provide the necessary information.

```
openssl req -new -newkey rsa:2048 -days 365 -nodes -x509 -keyout server.key
-out server.crt
```



```
C:\Users\administrator\Desktop\cert>
C:\Users\administrator\Desktop\cert>openssl req -new -newkey rsa:2048 -days 365 -nodes -x509 -keyout server.key -out server.crt
.......+........+....+.....+............+.+...+.........+++++++++++++++++++++++++++++++++++++++*..+...+++++++++++++++++++++++++++++++++++++++*...
....+.........+......+........+....+..+......+...................+........+.......+.....+............+......+....+...+...+.........+..+.+..............+...
.+.....+.+.....+............+...+....+++++++++++++++++++++++++++++++++++++++*.....+.+++++++++++++++++++++++++++++++++++++++*.+...+......+.....+..
.....+....+...+.....+.......+.....+...+.............+.....+.......+.+.+.............+.......+.....+......+.............+..+...+....+......+...
.............+...+..+.+..+...+........+.+..............+.+......+.....+....+...............+.+...+..+...+....+...+..+......+.+.......+...+..+........
.......+..+....+.....+....+.....+...+.........+......+...+....+...+.....+......+.......+.......+.+..+...+....+...+.........+.......+..+....+..
+.....+......+...+...........+............+.+.........+..+...+...+..............+.........+...+..+.............++++++
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IN
State or Province Name (full name) [Some-State]:DEL
Locality Name (eg, city) []:DEL
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Acme
Organizational Unit Name (eg, section) []:Acme
Common Name (e.g. server FQDN or YOUR name) []:Acme
Email Address []:Acme@acme.com

C:\Users\administrator\Desktop\cert>
C:\Users\administrator\Desktop\cert>
```

Place the server.crt and server.key file in the same folder as the script.

Step2: Workflow for the Script:

API Referenced: Get_Issue_Enrichment_Details

1. **Setup and Imports:** Ensure the necessary Python libraries are installed: **'requests'**, **'csv'**, **'sys'**, **'flask'**, **'gevent'**, **'datetime'** and **'urllib3'**. Disable warnings for insecure HTTP requests to handle self-signed certificates.

2. **Functions:**

   a) **get_X_auth_token:**
      - **Purpose:** Obtain an authentication token from the Cisco Catalyst Center API.
      - **Inputs:** IP address of Catalyst Center, username, password.
      - **Process:**
        - Send a POST request to the authentication endpoint.
        - Handle connection errors and invalid responses.
        - Return the authentication token.

   b) **get_deviceId:**
      - **Purpose:** Retrieve the device ID using the device's management IP address.
        Inputs: Catalyst Center IP address, authentication token, device management IP address.
      - **Process:**
        - Send a GET request to the network device endpoint with the management IP as a parameter.
        - Return the device ID from the response.

3. Setup Catalyst Center to send event notifications to the webhook receiver. The IP and port for the receiver will be specified in the script. For steps to configure the webhook destination refer 3.1.2

4. Once you run the script and it will start listening on the configured IP and port. The application should be accessible from the Catalyst Center over the network.

5. On generation of an event, the Catalyst Center will send the webhook notification as configured, towards the application. The incoming data is parsed by the script. It extracts

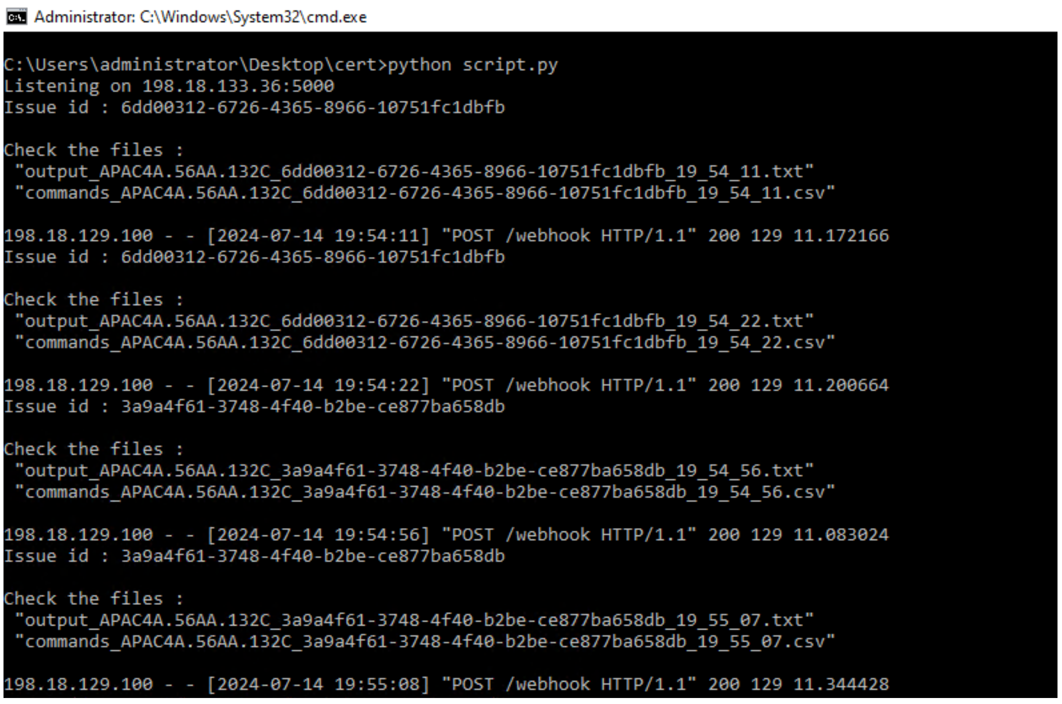
the ‘IssueId’ from it and forms another request to be made against the Catalyst Center to get the details of the issue.

a) The details of the issue are written to a file along with the suggested actions. Another file is generated that contains the commands as per the suggested actions. This commands file can be used as an input for the command runner API as mentioned in Use Case 3.1.1

b) Alternately, the device identified during the issue enrichment can be polled for checking its health to see any performance issues that might have caused the issue. Refer to Use case 3.4.2

6. On the console, the script prints the ‘IssueId’ that is parsed from the incoming webhook notification and the names of the files that consists additional details about the issue and the commands to run as per the suggested actions.

7. The output filenames contain the IssueId and the current time when the file is created for easy identification.



```
Administrator: C:\Windows\System32\cmd.exe

C:\Users\administrator\Desktop\cert>python script.py
Listening on 198.18.133.36:5000
Issue id : 6dd00312-6726-4365-8966-10751fc1dbfb

Check the files :
 "output_APAC4A.56AA.132C_6dd00312-6726-4365-8966-10751fc1dbfb_19_54_11.txt"
 "commands_APAC4A.56AA.132C_6dd00312-6726-4365-8966-10751fc1dbfb_19_54_11.csv"

198.18.129.100 - - [2024-07-14 19:54:11] "POST /webhook HTTP/1.1" 200 129 11.172166
Issue id : 6dd00312-6726-4365-8966-10751fc1dbfb

Check the files :
 "output_APAC4A.56AA.132C_6dd00312-6726-4365-8966-10751fc1dbfb_19_54_22.txt"
 "commands_APAC4A.56AA.132C_6dd00312-6726-4365-8966-10751fc1dbfb_19_54_22.csv"

198.18.129.100 - - [2024-07-14 19:54:22] "POST /webhook HTTP/1.1" 200 129 11.200664
Issue id : 3a9a4f61-3748-4f40-b2be-ce877ba658db

Check the files :
 "output_APAC4A.56AA.132C_3a9a4f61-3748-4f40-b2be-ce877ba658db_19_54_56.txt"
 "commands_APAC4A.56AA.132C_3a9a4f61-3748-4f40-b2be-ce877ba658db_19_54_56.csv"

198.18.129.100 - - [2024-07-14 19:54:56] "POST /webhook HTTP/1.1" 200 129 11.083024
Issue id : 3a9a4f61-3748-4f40-b2be-ce877ba658db

Check the files :
 "output_APAC4A.56AA.132C_3a9a4f61-3748-4f40-b2be-ce877ba658db_19_55_07.txt"
 "commands_APAC4A.56AA.132C_3a9a4f61-3748-4f40-b2be-ce877ba658db_19_55_07.csv"

198.18.129.100 - - [2024-07-14 19:55:08] "POST /webhook HTTP/1.1" 200 129 11.344428
```

Appendix

Related Documentation

Catalyst Center API reference guide
https://developer.cisco.com/docs/dna-center/cisco-dna-center-2-3-7-api-overview/

Catalyst Center configuration guide
https://www.cisco.com/c/en/us/td/docs/cloud-systems-management/network-automation-and-management/dna-center/2-3-3/admin_guide/b_cisco_dna_center_admin_guide_2_3_3.html

ServiceNow developer instance
https://developer.servicenow.com/dev.do#!/learn/learning-plans/washingtondc/new_to_servicenow/app_store_learnv2_buildmyfirstapp_washingtondc_personal_developer_instances