Cisco APO Use Case Runbook

## Technology: DC

## Level: FLY

## Name: **Ansible + ACI, NX-OS, & UCS**

**Data Center Ansible Configuration**

*Scenario: Enterprise Account wishes to automate their Data Center networking with Ansible scripts running from a Webhook on a Github push. They can use this code to check/monitor and create/update/remove a component to the aspect of the domain that is being managed for ACI, NX-OS, and UCS components.*

Customer Story: Customer J has a multi-data center footprint running Cisco ACI, NX-OS, and UCS servers. Today they manually make changes to their components which can lead to errors and configuration convergence. By automating changes using master Ansible playbook scripts in Github, they provide reusable, standardized, and repeatable configuration and can **save multiple outages a year, saving both $100,000 of outage time and keeping customer satisfaction high.**

Partner Story: VAR Partner J taught their deployment engineers to use the Ansible playbooks and was able to help their customers automate the provisioning of their datacenter changes, reduce configuration errors, and document how the system was deployed, increasing the services revenue on the deal deal through the sale of follow on automation services after the install.

**Code Exchange Link:** **Data Center Ansible Configuration** -<https://developer.cisco.com/codeexchange/github/repo/movinalot/ansible-configurations>

## 

## **Use Case Playbook**

This use case consists of 3 learning labs and a challenge. The 3 learning labs are written in YAML as Ansible playbooks and are located at: **Ansible Configurations**

<https://github.com/movinalot/ansible-configurations>.

*You will need to install Ansible (Version 2.7.10 was used when developing this repository and it’s dependency of Python3 with all the Python dependencies in the ‘requirements.txt’ file (by running ‘pip install -r requirements.txt’)*

***NOTE: if you run more recent releases of Ansible, some of the commands and the names have changed for the ‘ansible\_facts’ fields!*** For example, instead of looking for a vlan list for nxos, you will need to change ‘response.ansible\_facts.vlan\_list’ to ‘response.ansible\_facts.ansible\_net\_\_vlan\_list’. If in doubt of the naming, you can always create a ‘debug’ task and use a ‘msg:’ to display the entire response and look for the returned variable name. For Ansible 2.9 you should also change your variable from ‘ansible\_connection: httpapi’ to ‘ansible\_connection: network\_cli’. To speed up the queries that Ansible makes, you can also change the “gather\_subset: 'all’” to “gather\_subset: '!config'”.

### Usage

The labs contain code for ACI, NX-OS or UCS servers. Because we will be changing the config, we recommend either using the ACI or NX-OS code to run against either the Cisco DevNet "Open NX-OS with 9K" Reservable Sandbox or the Cisco DevNet ACI V4 Reservable Sandbox. If you wish to use the Ansible UCS Manager libraries for automation, you will need to reserve the sandbox and to **‘pip install ucsmsdk’** to enable Ansible to interact with UCS Manager.

To use the "Open NX-OS with 9K" Reservable Sandbox, go to: <https://devnetsandbox.cisco.com/RM/Diagram/Index/43b6200b-9fd1-4542-8e2e-759db2e345e4>

To use the ACI V4 Reservable Sandbox, go to: <https://devnetsandbox.cisco.com/RM/Diagram/Index/4fc5c0c5-088c-4813-a860-b58d31520dee?diagramType=Topology>

To make a reservation to use the UCS Manager Sandbox, go to: [https://devnetsandbox.cisco.com/RM/Diagram/Index/3323b7b0-b70b-4b1e-a929-6bdbff3aac8a](https://devnetsandbox.cisco.com/RM/Diagram/Index/3323b7b0-b70b-4b1e-a929-6bdbff3aac8a?diagramType=Topology).

From these links you can get the address, username, password, and port for running Ansible code against the sandbox.On reserved sandboxes, you will need to establish a VPN connection.

You will want to make sure you pull down the Ansible & Python code for the Labs to your local machine. The code is at: <https://github.com/movinalot/ansible-configurations>. You will need to insure that you have Ansible, Python3, and the Python dependency libraries installed on your machine. Appendix A discusses the installation of Ansible and its dependencies.

You will need to update the hosts you will use by editing the **‘inventory’** files in the aci, nxos or ucs folder with the appropriate addresses for the sandbox you are using.

To store usernames and passwords we will be using Ansible Vault to keep them encrypted and secure. To read more about Ansible Vault and how it works please see: <https://docs.ansible.com/ansible/latest/user_guide/vault.html>.

You will want to add the encrypted username and password to the YAML file in the **‘group\_vars’** folder using the Vault encryptor. Because ansible-vault is used, the vault password is required to run the playbook in an automated fashion. You will need to put your private vault password in your home directory. To do this, first ‘cd ~’ to get to your home directory, then create a file named ‘.vault\_pass.txt’ with your encrypt/de-encrypt password in it.

Then go to your aci, nxos or ucs folder and go to the group\_vars folder.

Encrypt your username and password. The example commands are in the YAML file. Run them once for the username and once for the password. Paste the outcome in place of the sample ones in the YAML file. You now have your Vault credentials created. You should also check your **‘connection’** and **‘environment’** variables to make sure they are correct for your system. We will also add a line for the port number from your lab sandbox, such as “port: 8181” for the NX-OS sandbox.

Therefore, to run an ansible playbook use the following command line in the appropriate domain directory: ‘ansible-playbook -i inventory site.yml --vault-password-file=~/.vault\_pass.txt’

You may also find it useful to increase the timeout values in the ‘ansible.cfg’ file if you receive timeout error messages. The values for ‘connect\_timeout’ and ‘command\_timeout’ under the [persistent\_connection] section are in seconds and will increase the waiting time for slow commands to run.

You can then follow the 3 Learning Labs in order to learn how to use Ansible to read and configure either NX-OS, ACI, or UCS (choose one) devices. While the guidelines below are written from the NX-OS lab, the others are similar enough to translate to.

In **Lab1** you will use the check.yml files in each domain to check for the specific configuration status. For example VLANs configured on UCS, NX-OS, or a tenant configured on ACI. Lab 1 should take approximately 30 minutes.

Using the ‘ansible-playbook -i inventory vlan-check.yml --vault-password-file=~/.vault\_pass.txt’ command we can see the VLAN id’s that we requested in the ‘app1.yml’ file are available to us to use in configuring a tenant for ACI.

Try changing the VLAN id’s to a range 100-101 and re-running to see if they are available.

In **Lab2** you will use the site.yml files in each domain to add/update/remove the specific configurations in each domain. For example add/update/delete VLANs on UCS, NX-OS, or a tenant on ACI. Lab2 should take approximately 30 minutes.

You will have verified in Lab1 that the VLAN id’s are available for your tenant, so now we will add them to you configuration. The ‘site.yml’ file builds on the information in the ‘vlan-check.yml’ and contains both the Configure VLAN and Remove VLAN code blocks.

The ‘site.yml’ file will either add VLAN id’s or remove them based on the ‘nxos\_state: present’ or ‘nxos\_state: absent' variable in the ‘group\_vars’ folder ‘nxos.yml’ file.

Try running the code to create the VLAN block at first. Then run the ‘vlan-check’ playbook to make sure they are not available. Then change the variable to ‘nxos\_state: absent' and run the ‘site.yml’ playbook again. Finally run the ‘vlan-check’ playbook to make sure they are available again.

In **Lab3** you will enable the Flask-based weblistener to process the payload from a Github push event webhook. Lab3 should take approximately 90 minutes (excluding prerequisites, below).

There are a few background technologies that this lab uses to complete the task. The first is the Python Flask library, which provides a lightweight web server that you can use as a GUI front end for your programs, based on HTML5. For a background on Flask, please see: <https://pythonspot.com/flask-web-app-with-python/>.

In addition, we will need to understand the Webex Teams API. A good lab to run is the Teams API introduction at: <https://developer.cisco.com/learning/tracks/devnet-express-cloud-collab-it-pro/automating-spark-itp/collab-spark-calling-apis-from-python-itp/step/1>. While doing this lab, be sure to capture your token for use with the webhook, below.

Finally, we will need to understand how a webhook works with Webex Teams. Before starting the lasb you should read: <https://developer.webex.com/docs/api/guides/webhooks> and it would be helpful to run the DevNet Webhook lab at: <https://developer.cisco.com/learning/lab/collab-sparkwebhook/step/1>.

To run the lab, you will need to fill in your Webex Teams token and room ID in the the ‘webhook\_vars.template’ file in the ‘webhook-listener’ folder.

We can then start the server by running the ‘run\_webhook\_listener.sh’ bash script which uses the Ansible Vault credentials we created earlier to access the sandbox. This shell script calls the ‘webhook\_listener.py’ file to start the server. The server runs on localhost on port 5403. If you need to change where it runs you can adjust the last line of the ‘webhook\_listener.py’ file here it has: ‘host='0.0.0.0', port=5403’

You can then hit the server at the ‘localhost:5403’ address in your web browser to make sure it is running.

Your webhook listener will be waiting to process calls from the GitHub site at your workstations IP address, so this will need to be a public IP address or have a Dynamic DNS entry to work. THe API URI you are exposing will be at “http://{{ your\_IP\_address }}:5403/events” so you will need to set up your GitHub repo to publish commit messages to this address. The guide to set up a webhook on GitHub.com is at: <https://developer.github.com/v3/repos/hooks/>. For a local git repo, see: <https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks>. For other systems you will need to find the appropriate guide.

You can then fire the webhook and see the results in real time in your Webex Teams room.

Once you complete the Learning labs you will be ready for the challenge.

## Completion Challenge

In order to provide a Proof of Performance Challenge we are going to set the following goal. The challenge is to make a Webex Teams room using the Webex Teams API to automatically display the status of the Flask based Weblistener to enable your very own ChatOps pipeline.

Your demo should be to run the ‘check’ file to document the current status and save this output to a file, then run the ‘site’ file to make a change. You should see the change occur in the WebEx Teams room. Re-run the ‘check’ file to document the current status with the change, then run the ‘site’ file to delete the change and see the change occur in the WebEx Teams room. Finally re-re-run the ‘check’ file and compare it to the file you saved in the original check.

Once you have completed the challenge, please host a 2 minute demo of your application running and send the link to {{ [email\_addr@cisco.com](mailto:email_addr@cisco.com) }} where your video will be reviewed for credit. We suggest doing a Webex demo and recording it. Share the recording link via the email for review.

## Appendix A

This appendix is a small guide to help you to:

* Install Python
* Install Ansible

The following list discusses the links to follow to accomplish these tasks.

NOTE: There is no native Ansible Control node for Windows. Therefore, your options are running a Virtual Machine with Linux, running a Linux container, or running Ansible from the Windows Subsystem for Linux (WSL). You will require both Python and Ansible to be installed on the VM/Container/WSL. Please follow the Linux install methods, below.

## For Mac Computers:

The MacOS usually has Python (2.7 or 8) installed. If your system updates are up to date, you should also have an appropriate version of pip installed. If you are using python3 you will want to make sure you define the environment variable in your inventory file. To test, type ‘python --version’ from a terminal window. If it is not installed, I recommend using the homebrew package installer which will install the latest stable version using ‘**brew install python’** and **‘brew install pip’.**

There are 2 ways to install Ansible on top of Python on your laptop. Use a Terminal window to complete either of them:

The first is the homebrew package installer which will install the latest stable version using ‘**brew install ansible**’

THe second is to use pip and install with ‘**pip install --user ansible**’ and to add the underlying transport protocol ‘**pip install --user paramiko**’

Test your install by typing ‘**ansible --version**’

If your installation didn’t copy the ‘ansible.cfg’ file to /etc/ansible/ansible.cfg you should complete this step manually.

* sudo mkdir /etc/ansible
* cd /etc/ansible
* sudo {your\_favorite\_text\_editor} ansible.cfg
* Paste a copy into this file and save. The current ‘ansible.cfg’ file can be found at: <https://github.com/ansible/ansible/blob/devel/examples/ansible.cfg>

There are many web pages that can help you troubleshoot issues if you have trouble.

## For Linux Computers, VMs, Containers, and WSL:

The Linux flavors are similar except for the appropriate package manager name. Where you see {pkg\_mgr} in commands below, please substitute your package manager (apt, yum, rpm, etc) in place of the {pkg\_mgr}. Also, if you are running python3 on a platform as well as python (aka python2) please replace ‘python’ with ‘python3’.

Most Linux systems already have Python installed. You can check this by typing ‘python --version’ in a terminal window . If python is not found you can install it with ‘**sudo {pkg\_mgr} install python**’ and ‘**sudo {pkg\_mgr} install pip**’

There are 2 ways to install Ansible on top of Python in you Linux machine. Use a Terminal window to complete either of them:

The first is the {pkg\_mgr} installer which will install the latest stable version using ‘**sudo {pkg\_mgr} install ansible**’

The second is to use pip and install with ‘**pip install --user ansible**’ and to add the underlying transport protocol ‘**pip install --user paramiko**’

If your installation didn’t copy the ‘ansible.cfg’ file to /etc/ansible/ansible.cfg you should complete this step manually.

* sudo mkdir /etc/ansible
* cd /etc/ansible
* sudo {your\_favorite\_text\_editor} ansible.cfg
* Paste a copy into this file and save. The current ‘ansible.cfg’ file can be found at: <https://github.com/ansible/ansible/blob/devel/examples/ansible.cfg>

For help - I would recommend the Ansible Install Guide page at: <https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html>