

Cisco **IMPACT**

Managing Cisco UCS with Ansible

DNW07

Speaker:

David Soper, Technical Marketing Engineer

Table of Contents

Learning Objectives	2
Overview	2
Prerequisites	2
Getting Started	2
Lab Tasks	3
Task 1: Verify Ansible and the UCSM Python SDK are Installed	3
Step 1: Install Ansible	3
Step 2: Install UCSM Python SDK	3
Task 2: Get Example Playbooks	3
Task 3: View and Customize Example Playbooks	4
Step 1: Connecting to UCS: Edit the Inventory File with UCS API Connection Information	4
Step 2: Run a Server Configuration and Deployment Playbook	4
Step 3: View Roles for the Server Deployment Playbook	5
Step 4: Checking what Ansible will Change.....	6
Step 5: Run the Server Deployment Playbook	7
Step 6: View Policy/Profile Roles	8
Task 4 (Optional): ucs_managed_objects and User Defined Configuration	9

Learning Objectives

Overview

Cisco Unified Computing System™ (Cisco UCS®) and Cisco HyperFlex® platforms offer an intelligent level of management that enables IT organizations to analyze, simplify, and automate their environments.

Within this workshop, you will use Ansible to interact with the UCS API and perform a variety of resource management tasks.

Prerequisites

While not required prior to starting this lab, familiarity with the Linux/MacOS command line and use of a text editor such as Vi will be helpful. Working knowledge of Ansible will also be helpful, but again is not required.

Getting Started


This lab will interact with the UCS API from a Linux/MacOS workstation.

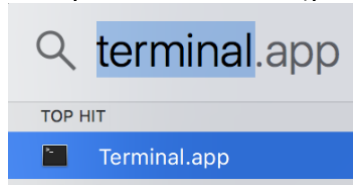
Lab Tasks

The following tasks will configure Ansible on your system, retrieve a set of example Ansible playbooks, and customize the playbooks for use in this lab.

Task 1: Verify Ansible and the UCSM Python SDK are Installed

Step 1: Install Ansible

Open a terminal on your workstation (you can spotlight search  for “terminal” if you



are on MacOS (Terminal.app), and type “ansible --version”. You should see 2.8 or later:

```
$ ansible --version
ansible 2.8.2
  config file = None
  configured module search path =
  ['/Users/dsoper/.ansible/plugins/modules',
  '/usr/share/ansible/plugins/modules']
  ansible python module location =
  /usr/local/lib/python3.6/site-packages/ansible
  executable location = /usr/local/bin/ansible
  python version = 3.6.5 (default, Apr 20 2018, 18:22:17) [GCC
  4.2.1 Compatible Apple LLVM 9.0.0 (clang-900.0.39.2)]
```

If Ansible is not installed or reports an older version, you can type “pip install -U ansible” to install/update.

Step 2: Install UCSM Python SDK

To install the UCS Manager python SDK, type “pip install -U ucsm-sdk”. You can verify that the Python SDK is installed with the following commands:

```
$ python
Python 3.6.5 (default, Apr 20 2018, 18:22:17)
[GCC 4.2.1 Compatible Apple LLVM 9.0.0 (clang-900.0.39.2)] on
darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> from ucsm-sdk import ucshandle
```

Task 2: Get Example Playbooks

Several example playbooks are hosted on GitHub at <https://github.com/CiscoUcs/ucsm-ansible>. In the terminal window, create a working directory and change to it:

```
$ mkdir ~/dnw07; cd ~/dnw07
```

Then use git to clone the ucsm-ansible repo and change directories to it:

```
$ git clone https://github.com/CiscoUcs/ucsm-ansible
Cloning into 'ucsm-ansible'...
<snip>
Resolving deltas: 100% (74/74), done.
```

```
$ cd ucsm-ansible/
```

Task 3: View and Customize Example Playbooks

There are many ways to organize data and connection information for resources you want to manage with Ansible. In this lab, we'll use a basic inventory file with UCS login information and playbooks with default values used in configuration (you can change values used with Ansible variable substitution).

Step 1: Connecting to UCS: Edit the Inventory File with UCS API Connection Information

Ansible uses SSH to connect to UCS just like it would with any other host, right? Actually, we use the UCS API for UCS Manager connections and management, and the API uses the same HTTPS connections you use with a web browser to access the UCS Manager web UI.

The API is the basis for everything in UCS (including CLI commands that you might use via SSH), so we'll cut out the middle man and use the API for configuration. Our API access is driven through the UCS Manager Python SDK, but you usually don't have to deal with Python (unless you want to) outside of installing the SDK which you did above.

To tell Ansible how to connect to UCS's API, we'll specify our ip/hostname in the inventory file along with our credentials (username and password). Ansible has several options such as vault for securely storing system credentials, but in this lab it's in the inventory for simplicity.

```
[ucs]
13.58.22.56
```

```
[ucs:vars]
username=admin
password=password
```

Edit the ip address (which is what Ansible will use for a hostname) if needed for the UCS Manager IP you want to configure. In this basic inventory, we have a 'ucs' host group where we could add other hosts as needed and they'll all share variables setup in the [ucs:vars] block.

Step 2: Run a Server Configuration and Deployment Playbook

The ucsm-ansible repository has a Server configuration and deployment playbook named server_deploy.yml. View this file:

```
---
#
# Configure UCS, Associate Service Profiles, and Install OS
#
...
#
- hosts: "{{ group | default('ucs') }}"
  connection: local
  gather_facts: false
  vars:
    # The UCS domain hostname can be set in the inventory or on the
    command line as needed
    hostname: "{{ inventory_hostname }}"
```

```

# Names for Service Profiles, Policies, and number of Profiles
template_name: auto-template
vmedia_policy: cdd-nfs
profile_name: auto-profile
num_profiles: 2
tasks:
  - block:
      # Configure default pools and other settings
      ...

```

Note that the “vars” section above contains Service Profile Template, Virtual Media Policy, and other variables that you can change if you’d like. Variables would usually come from Ansible `group_vars` and `host_vars` directories (or the inventory), but we’ve placed them in the playbook for simplicity (check out Ansible’s excellent documentation pages if you need more information on how to organize your inventory and variables that Ansible will use in configuration).

Recommended Steps to Check YAML Syntax

What’s going to happen if I run a playbook but I have a typo?

Ansible doesn’t always tell you right away what’s wrong with your YAML, and YAML syntax can be challenging to debug. Have no fear though, as there are utilities such as `yamllint` that can help identify issues prior to running playbooks:

```
$ yamllint server_deploy.yml
```

(“pip install yamllint” if yamllint is not already installed on the workstation)

You can ignore “line too long” errors/warnings, and note that these can be turned off by creating and editing a `~/.config/yamllint/config` file:

```

# yamllint config file
extends: default

rules:
  # 140 chars should be enough, but don't fail if a line is
  # longer
  line-length:
    max: 140
    level: warning

```

`Yamllint` is pretty simple to use and should save you some headaches. Are you using something better for checking/debugging YAML – let the instructor know!

Step 3: View Roles for the Server Deployment Playbook

The `server_deploy.yml` playbook has 1 play and several tasks that configure all of the policies and profiles needed to deploy service profiles from templates (in this case the default will be 2 profiles deployed from a single template).

The playbook uses roles defined in the “roles” subdirectory of the repo to configure each specific policy and profile. Here’s what the tasks look like:

```

tasks:
  - block:
      # Configure default pools and other settings
      - import_role:
          name: servers/defaults
          tags: defaults
      # Configure Service Profile Template with default settings
      - import_role:
          name: servers/service_profile_templates
          tags: templates
      # Create Service Profiles from template and associate
      - import_role:
          name: servers/service_profiles
          tags: profiles
      # Use the localhost's environment and Python
      delegate_to: localhost

```

A few things to note in the example playbook:

- We use “block” to wrap related tasks so there are common directives for all the enclosed tasks. “delegate_to: localhost” is used to run with the same python interpreter used to install Ansible (what “which python” shows for the current user).
- Each import_role line will run tasks in the roles subdirectory. For example, the servers/defaults will run tasks in roles/servers/defaults/tasks/main.yml. Vars in the playbook (or defined anywhere else that Ansible looks them up) will be passed to the roles tasks.
- Tags can be used to run only a specific part of the playbook if needed.

Step 4: Checking what Ansible will Change

We’re almost ready to run the playbook, but do we really want to make changes? Fortunately, the UCS modules all support Ansible’s check mode. Check mode allows you to see what Ansible would change without actually making any changes to UCSM:

```
$ ansible-playbook -i inventory server_deploy.yml --check
```

```

PLAY [ucs]
*****
*****

TASK [servers/defaults : Configure default IP Pool]
*****

```

You can increase Ansible’s verbosity with the -v option (and multiple Vs, like -vvv) to see exactly what Ansible will do:

```

$ ansible-playbook -i inventory server_deploy.yml --check -vvv
ansible-playbook 2.7.9
  config file = None
  configured module search path =
['/Users/dsoper/.ansible/plugins/modules',
'/usr/share/ansible/plugins/modules']

```

```

ansible python module location =
/usr/local/lib/python3.6/site-packages/ansible
executable location = /usr/local/bin/ansible-playbook
python version = 3.6.5 (default, Apr 20 2018, 18:22:17) [GCC
4.2.1 ...
<snip>
<localhost> EXEC /bin/sh -c
'/usr/local/opt/python/bin/python3.6
/Users/dsoper/.ansible/tmp/ansible-tmp-1556645678.4761322-
240916740175025/AnsiballZ_ucs_ip_pool.py && sleep 0'
<localhost> EXEC /bin/sh -c 'rm -f -r
/Users/dsoper/.ansible/tmp/ansible-tmp-1556645678.4761322-
240916740175025/ > /dev/null 2>&1 && sleep 0'
ok: [172.16.143.175 -> localhost] => {
  "changed": false,
  "invocation": {
    "module_args": {
      "default_gw": "0.0.0.0",
      "descr": "",
      "first_addr": null,
      "hostname": "172.16.143.175",
      "ipv4_blocks": [
        {
          "default_gw": "198.18.0.1",

```

The “module_args” just above are what’s being passed to the ucs_ip_pool module.

Step 5: Run the Server Deployment Playbook

You can run the server_deploy.yml playbook, and when run without the --check option you should see your UCS Manager domain configured. You should also be able to re-run and see ‘ok’ for all the tasks indicating that Ansible hasn’t made any changes.:

```

$ ansible-playbook -i inventory server_deploy.yml

PLAY [ucs]
*****

TASK [servers/defaults : Configure default IP Pool]
*****
ok: [172.16.143.175 -> localhost]

TASK [servers/defaults : Configure default MAC Pool]
*****
ok: [172.16.143.175 -> localhost]

TASK [servers/defaults : Configure default UUID Pool]
*****
ok: [172.16.143.175 -> localhost]

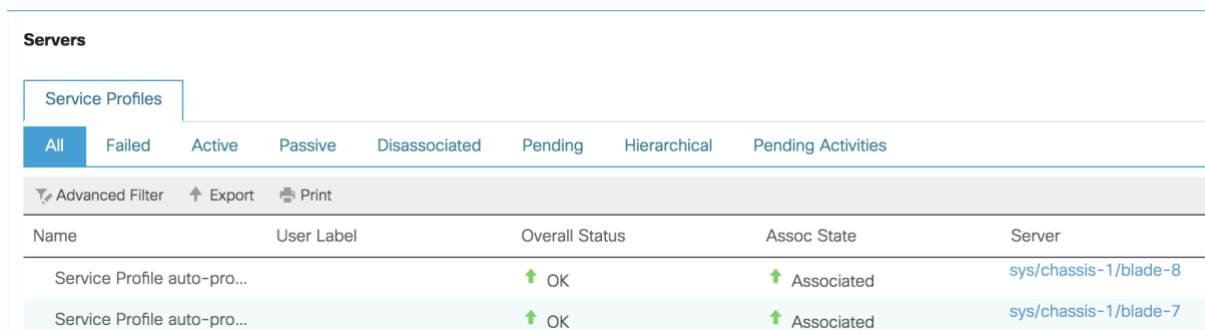
TASK [servers/defaults : Configure default Virtual Media
Policy] *****

```

ok: [172.16.143.175 -> localhost]

TASK [servers/defaults : Configure default Boot Order Policy]

ok: [172.16.143.175 -> localhost]



The screenshot shows the 'Servers' page in UCS Manager. It features a navigation bar with tabs for 'Service Profiles', 'All', 'Failed', 'Active', 'Passive', 'Disassociated', 'Pending', 'Hierarchical', and 'Pending Activities'. Below the navigation bar are options for 'Advanced Filter', 'Export', and 'Print'. The main content is a table with the following columns: Name, User Label, Overall Status, Assoc State, and Server. Two rows are visible, both showing 'Service Profile auto-pro...' with an 'OK' status and 'Associated' state, linked to servers 'sys/chassis-1/blade-8' and 'sys/chassis-1/blade-7'.

Name	User Label	Overall Status	Assoc State	Server
Service Profile auto-pro...		↑ OK	↑ Associated	sys/chassis-1/blade-8
Service Profile auto-pro...		↑ OK	↑ Associated	sys/chassis-1/blade-7

There's a lot more you can do with Ansible, so feel free to try other settings or operations. Since this lab is using a shared environment the default playbook may not make any changes, but you can change `profile_name` or other variables to see how Ansible manages changes.

Step 6: View Policy/Profile Roles

The IP Pool and other configuration of UCS Manager is performed in tasks within the roles subdirectory using the `ucs_ip_pool` and other UCS Ansible module. Here's the IP pool configuration portion of the defaults role defined at `roles/servers/defaults/tasks/main.yml`:

```
ucs_ip_pool:
  <<: *login_info
  name: ext-mgmt
  ipv4_blocks:
    - first_addr: 198.18.0.20
      last_addr: 198.18.0.40
      subnet_mask: 255.255.255.0
      default_gw: 198.18.0.1
  tags: ip_pool
```

Ansible's official documentation (on the web) or `ansible-doc ucs_ip_pool` will tell you more about how to use the module and what's supported:

> `UCS_IP_POOL` (/usr/local/lib/python3.6/site-packages/ansible/modules/remote_

Configures IP address pools and blocks of IP addresses on Cisco UCS Manager. Examples can be used with the UCS Platform Emulator <https://communities.cisco.com/ucspe>.

OPTIONS (= is mandatory):

- `default_gw`

The default gateway associated with the IPv4 addresses in the block.

[Default: 0.0.0.0]

Task 4 (Optional): ucs_managed_objects and User Defined Configuration

From the previous task, when you look at boot order or server pool setup, you'll see use of a "general purpose" Ansible module – `ucs_managed_objects`. `ucs_managed_objects` does not abstract away API details like some of the `ip_pool/mac_pool/etc.` modules do, but it does allow for Ansible config of any UCS Managed Object. Also, the module still supports check mode and is idempotent (you can run repeatedly and still get the same result).

`ucs_managed_objects` directly imports Python modules as needed to carry out configuration. Unsure what is needed to create Python definitions in Ansible – checkout the Python DevNet or other Python SDK programming labs in DevNet to learn more about the Python SDK's structure and how to do code gen in Python. Once you have Python code ready, you can just put the Python module names and property values into Ansible and Ansible will handle the rest (including only making changes if needed based on the supplied property values):

```
ucs_managed_objects:
  <<: *login_info
  objects:
    - module: ucsmsdk.mometa.compute.ComputePool
      class: ComputePool
      properties:
        parent_mo_or_dn: org-root
        name: default
      children:
        - module: ucsmsdk.mometa.compute.ComputePooledSlot
          class: ComputePooledSlot
          properties:
            chassis_id: '1'
            slot_id: '1'
```

Congratulations on completing the UCS and Ansible DevNet workshop!