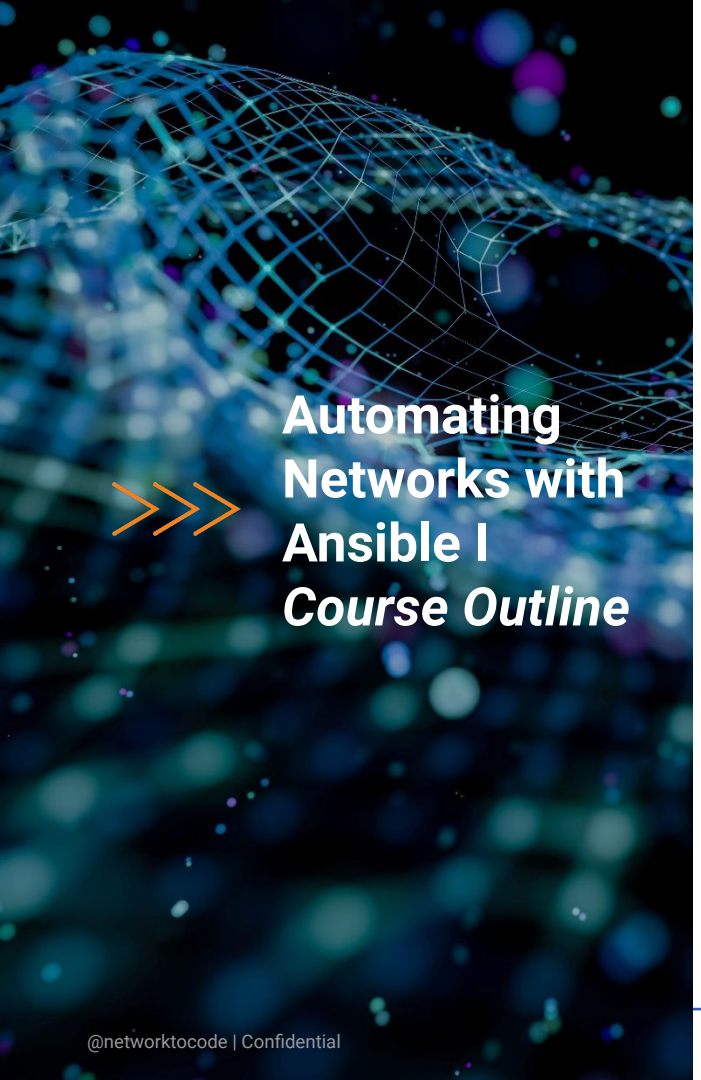

>>> network .toCode()
Network Programming & Automation
Module 2: Automating Networks with Ansible



Automating Networks with Ansible I
Course Outline

Introduction

- Course Overview
- What's Possible with Ansible
- Introduction to Ansible Vocabulary

Ansible Variables and Password Security

- Quick Look at YAML & JSON
- Understanding Ansible Variables
- Ansible Vault for Password Security
- Debug Module to Display Ansible Variable Content

Showing Network Device Configurations

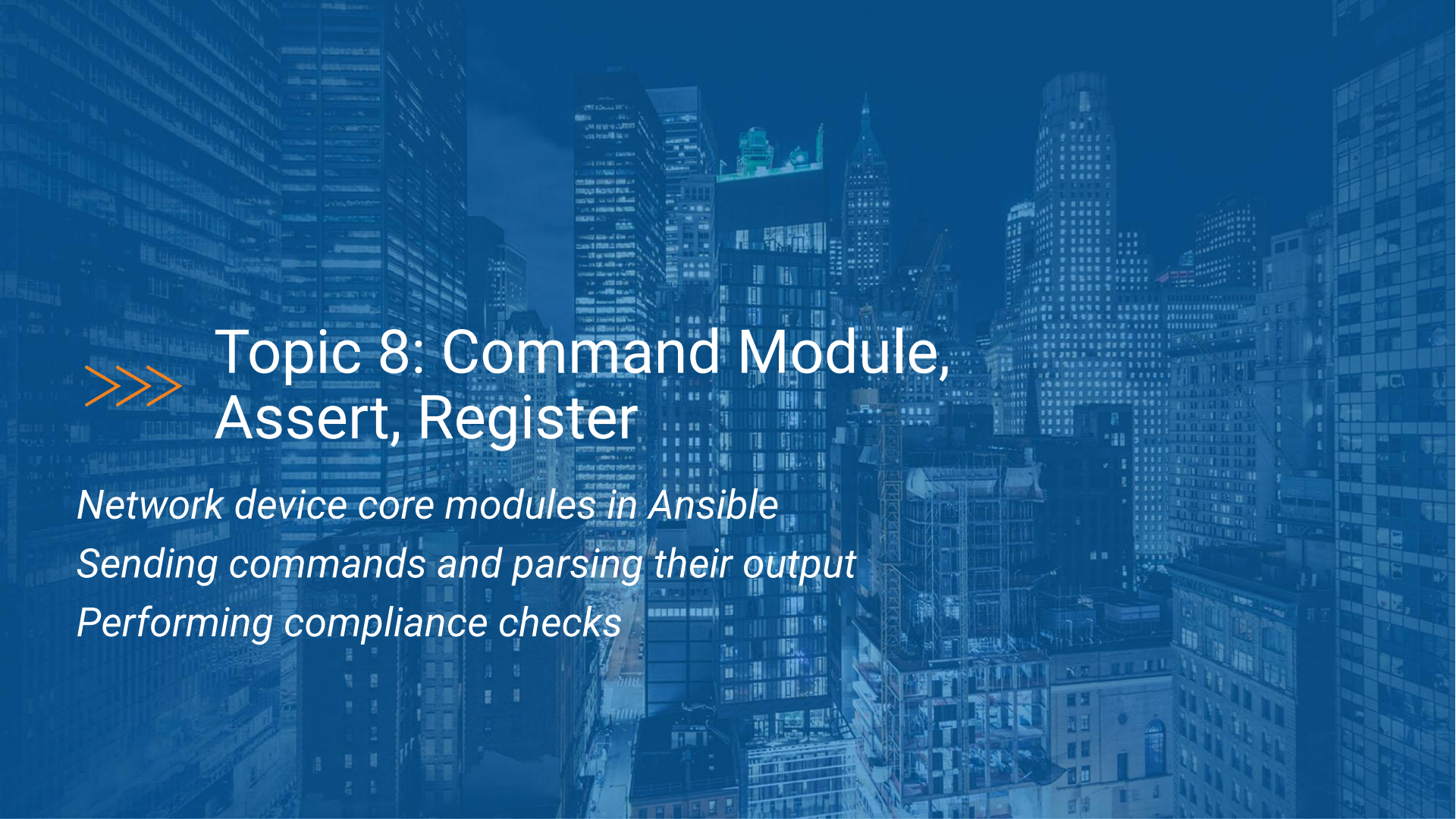
- Show Commands on Network Devices
- How do you see the data being gathered?
- Jinja2 Templates
- Loops and Registers
- Parsing Unstructured Data

Module
Table of Contents

Lecture 4: Operational Compliance with Ansible

Topic 8 - Command Module, Assert, Register

Labs 8-10



Topic 8: Command Module, Assert, Register

Network device core modules in Ansible

Sending commands and parsing their output

Performing compliance checks

>>> Core Modules

We cover three types of core modules:

- `*_command` - Run arbitrary commands on devices
- `*_config` - Manage configuration sections on devices
- `*_facts` - Gather facts on devices

The modules are vendor specific and usually support SSH plus a vendor API:

- `iosxr_*`
- `ios_*`
- `eos_*`
- `junos_*`
- `nxos_*`
- actively growing

>>> *_command

The *_command modules are used to send enable and exec mode commands to the device, e.g. execute show commands and gather data from devices.

`commands` parameter can accept a single command or a list of commands

Refer to `ansible-doc` for full list of parameters

```
---
- name: "EXECUTE SHOW COMMANDS IOS DEVICES"
  hosts: "ios"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false
  tasks:
    - name: "EXECUTE A SINGLE COMMAND"
      cisco.ios.ios_command:
        commands: "show version"
    - name: "EXECUTE LIST OF COMMANDS"
      cisco.ios.ios_command:
        commands:
          - "show version"
          - "show ip int brief"
```

>>> Demo

- Review `ansible-doc` for `ios_command` and see available parameters
- Take note of the examples at the bottom of the output

>>> *_command (cont'd)

- There are a number of options available to change the format of data returned (for select OS types).

```
- name: "SINGLE COMMAND"
  cisco.nxos.nxos_command:
    commands: "show version"
- name: "LIST OF COMMAND STRINGS"
  cisco.nxos.nxos_command:
    commands:
      - "show version"
      - "show hostname"
- name: "LIST OF DICTIONARIES"
  cisco.nxos.nxos_command:
    commands:
      - command: "show version"
        output: "json"
      - command: "show version"
        output: "text"
```

```
- name: "SHOW COMMANDS TO JUNOS"
  hosts: "junos"
  connection: "ansible.netcommon.netconf"
  gather_facts: false
  tasks:
    - name: "EXECUTE JUNOS COMMANDS"
      junipernetworks.junos.junos_command:
        commands:
          - "show version"
          - "show interfaces"
    - name: "EXECUTE JUNOS COMMANDS - TEXT"
      junipernetworks.junos.junos_command:
        format: "text"
        commands:
          - "show version"
          - "show interfaces"
```

>>> Playbook Execution

Sample playbook execution:

```
$ ansible-playbook -i inventory gather.yml

PLAY [RUN COMMANDS] ************************************************

TASK [SINGLE COMMAND]
********************************************************************
ok: [csr1]
ok: [csr2]
ok: [csr3]

PLAY RECAP *********************************************************
csr1            : ok=1    changed=0    unreachable=0    failed=0
csr2            : ok=1    changed=0    unreachable=0    failed=0
csr3            : ok=1    changed=0    unreachable=0    failed=0
```

>>> Viewing Response Data

There are three ways to view data returned from a module.

Remember, every module returns JSON data.

There are three ways to see it:

1. Execute playbooks in verbose mode,e.g. -v
2. Use the register task attribute with the debug module
3. Use the set_fact module to define a variable

Using `register` saves the JSON return data as a dictionary that can be consumed as a variable within a playbook or template.

```
- name: "EXECUTE COMMANDS"
  cisco.nxos.nxos_command:
    commands:
      - "show version"
  register: "output"
- ansible.builtin.debug:
    var: "output"
```

>>> set_fact Module

- Store value into a variable in another way besides the register task attribute
- The `set_fact` module allows a task to define a variable
- Sometimes `set_fact` is used to simplify the naming of a variable.
- `register` stores the output of a task into a variable.

```yaml
vars:
  locations:
    amer:
      nyc:
        - "nyc-dc"
        - "nyc-campus"
      sjc:
        - "sjc-branch"
    apac:
      hk:
        - "hk-dc"
        - "hk-campus"
tasks:
  - name: "PRINT ALL LOCATIONS"
    ansible.builtin.debug:
      var: "locations"

  - name: "SJC LOCATIONS"
    ansible.builtin.set_fact:
      sjc_locations: "{{ locations['amer']['sjc'] }}"

  - name: "PRINT ALL LOCATIONS"
    ansible.builtin.debug:
      var: "sjc_locations"
```

>>> Demo

- Demo playbook using `ios_command`
- Show sending both 1 and 2 commands and see the difference in response data
- Check length of `stdout`

>>> *_command

- Run arbitrary commands on devices.
- Show command data stored in stdout (always a list)
- The stdout list has a length equal to the number of commands sent to the device

```
- name: "SEND SHOW VERSION TO DEVICE"
    cisco.ios.ios_command:
      commands:
        - 'show version'
    register: "output"
- name: "TEST REGISTERED OUTPUT"
  ansible.builtin.debug:
    var: "output"
- name: "SEE ALL KEYS OF REGISTERED
DICTIONARY"
  ansible.builtin.debug:
    var: "output.keys()"
- name: "TEST GETTING SHOW DATA"
  ansible.builtin.debug:
    var: "output['stdout'][0]"
```

```
TASK [TEST REGISTERED OUTPUT] ******************************************
ok: [csr1] => {
    "output": {
        "changed": false,
        "stdout": ["Cisco IOS XE Software, Version 16.08.01a\nCisco
            IOS Software [Everest], Virtual XE Software
(X86_64_LINUX_IOSD-UNIVERSALK9-M),Version 16.6.2, RELEASE SOFTWARE
(fc2)\nTechnical Support: http://www.cisco.com/techsupport\nCopyright (c)
1986-2017 by Cisco Systems, Inc.\nCompiled Wed 01-Nov-17 07:24 by
mcpre\n\n\nCisco IOS-XE software, Copyright (c) 2005-2017 by cisco Systems,
Inc.\nAll rights reserved.Certain components of Cisco IOS-XE software
are\nlicensed under the GNU General Public License (\"GPL\") Version 2.0.
The\nsoftware code licensed under GPL Version 2.0 is free software that
comes\nwith ABSOLUTELY NO WARRANTY.You can redistribute and/or modify
such\nGPL code under the terms of GPL Version 2.0.For more details, see
the\ndocumentation or \"License Notice\" file accompanying the IOS-XE
software,\nor the applicable URL provided on the flyer accompanying the
IOS-XE\nsoftware.\n\n\nROM: IOS-XE ROMMON\n\ncsr1 uptime is 30 - output
omitted"],
        "stdout_lines": [], # list closed for slide formatting
        "warnings": []
    }
}
```

>>> Saving show command data to a file

- Using templates to save data to a file (use any logic as necessary)
- Use copy module to just dump show response to a file

```
---
- name: "BACKUP SHOW VERSION"
  hosts: "ios"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false
  tasks:
    - name: "GET SHOW COMMANDS"
      cisco.ios.ios_command:
        commands:
          - "show version"
      register: "output"
    - name: "OPTION 1 - SAVE SH COMMAND TO FILE"
      ansible.builtin.template:
        src: "basic-copy.j2"
        dest: "./commands/{{ inventory_hostname}}-ver.txt"
    - name: "OPTION 2 - SAVE SH COMMAND TO FILE"
      ansible.buitin.copy:
        content: "{{ output['stdout'][0] }}"
        dest: "./commands/{{ inventory_hostname}}-ver.txt"
```

```
# basic-copy.j2

{{ output['stdout'][0] }}
```

>>> Multi Vendor cli_command Module

- You can create a single task to send a single command for two different vendor devices

```
---
- name: "SHOW VERSION FOR IOS"
  hosts: "csr1,vmx1"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false
  tasks:
    - name: "GET SHOW COMMANDS"
      ansible.netcommon.cli_command:
        command: "show version"
      register: "output"
```

- You can also send a list of commands using a single module for both vendors

```
---
- name: "HOW VERSION FOR IOS"
  hosts: "csr1,vmx1"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false
  tasks:
    - name: "GET SHOW COMMANDS"
      ansible.netcommon.cli_command:
        command: "{{ item }}"
      register: "output"
      loop:
        - "show version"
        - "show interface"
```

Note: Not all CLI commands are compatible for all vendors, so make sure the command you are sending works for the devices being targeted.

>>> cli_command Single Command

arbitrary Run commands on devices.

Show command data stored in stdout (always a dictionary)

```
- name: "GET SHOW COMMANDS"
  ansible.netcommon.cli_command:
    command: "show version"
  register: "output"
- name: "TEST REGISTERED OUTPUT --> SEE RIGHT"
  ansible.builtin.debug:
    var: "output"
- name: "SEE ALL KEYS OF REGISTERED DICTIONARY"
  ansible.builtin.debug:
    var: "output.keys()"
- name: "TEST GETTING SHOW DATA"
  ansible.builtin.debug:
    var: "output['stdout']"
```

```
TASK [TEST REGISTERED OUTPUT ]
*********************************************************************
****************
ok: [vmx1] => {
    "output": {
        "changed": false,
        "failed": false,
        "stdout": "Hostname: vmx1\nModel: vmx\nJunos: 18.2R1.9\nJUNOS OS
Kernel 64-bit  [20180614.6c3f819_builder_stable_11]\nJUNOS OS libs
       ....output omitted
        "stdout_lines": [
            "Hostname: vmx1",
            "Model: vmx",
            "Junos: 18.2R1.9",
            ....output omitted
            ]
    }
}
ok: [csr1] => {
    "output": {
        "changed": false,
        "failed": false,
        "stdout": "Cisco IOS XE Software, Version 16.08.01a\nCisco IOS
Software [Fuji], Virtual XE Software (X86_64_LINUX_IOSD-UNIVERSALK9-M),
        ....output omitted
        "stdout_lines": [
            "Cisco IOS XE Software, Version 16.08.01a",
            ....output omitted
               ]
    }
}
```

>>> cli_command List of Commands

- When it's a list of commands, we get a list of results
- Each element inside the results will be stdout per command

```
- name: "GET SHOW COMMANDS"
  cli_command:
    command: "{{ item }}"
  register: "output"
  loop:
    - "show version"
    - "show interface"
- name: "TEST REGISTERED OUTPUT" #--> SEE TO RIGHT
  ansible.builtin.debug:
    var: "output"

- name: "SEE ALL KEYS OF REGISTERED DICTIONARY"
  ansible.builtin.debug:
    var: "output.keys()"

- name: "TEST GETTING SHOW VERSION"
  ansible.builtin.debug:
    var: "output['results'][0]['stdout']"

- name: "TEST GETTING SHOW INTERFACE"
  ansible.builtin.debug:
    var: "output['results'][1]['stdout']"
```

```
TASK [SEE ALL KEYS OF REGISTERED DICTIONARY]
*************************
ok: [csr1] => {
    "output.keys()": "dict_keys(['results', 'msg', 'changed'])"
}
ok: [vmx1] => {
    "output.keys()": "dict_keys(['results', 'msg', 'changed'])"
}
TASK [TEST GETTING SHOW VERSION]
************************************************************
ok: [csr1] => {
    "output['results'][0]['stdout']": "Cisco IOS XE Software,
Version 16.08.01a\nCisco IOS Software [Fuji], Virtual XE Software
    ...output omitted
}
ok: [vmx1] => {
    "output['results'][0]['stdout']": "Hostname: vmx1\nModel:
vmx\nJunos: 18.2R1.9\nJUNOS OS Kernel 64-bit
[20180614.6c3f819_builder_stable_11]\nJUNOS OS libs
    ...output omitted
```

>>> Performing Compliance Checks

- Using the assert module
- Ensure certain conditions exist within the network
- Leverage data that you previously registered
- Validate routes exist, changes happen, and configuration is as desired

```
- name: "IOS SHOW VERSION"
  cisco.ios.ios_command:
    commands:
      - "show version"
  register: "output"
- name: "ENSURE OS VERSION IS CORRECT"
  ansible.builtin.assert:
    that:
      - "'Version 16.6.2' in output['stdout'][0]"
      - "'0x2102' in output['stdout'][0]"
```

>>> Demo

- What happens when an assertion fails?
- When a task fails, the default is that no other task runs for that device in a playbook.
- An assertion failure counts as a task failure
- Introduce `ignore_errors: True`
- Introduce `fail_msg` and `success_msg`

>>> Auto Create Files

- Manually creating directories can be a tedious task
- You can use the Ansible file module to auto create directories, empty files and symlinks.

Check the docs to see more info and parameters

```
ntc@jump-host:ansible$ ansible-doc file
> FILE
(/home/ntc/.local/lib/python3.6/site-pac
kages/ansible/modules/files/file.py)

Sets attributes of files, symlinks, and
directories,
or removes files/symlinks/directories.
Many other modules support the same
options as the
`file' module - including [copy],
[template], and [assemble].
For Windows targets, use the [win_file]
module instead.
```

Manual Way:

```
ntc@jump-host:ansible$ mkdir ios junos nxos
arista
ntc@jump-host:ansible$
```

Automated Way:

```
- name: "CREATE DIRECTORIES BASED ON OS"
  ansible.builtin.file:
    path: "./{{ ansible_network_os }}/"
    state: "directory"
```

Results:

```
ntc@jump-host:ansible$ tree
.
├── arista
├── ios
├── junos
└── nxos

4 directories, 0 files
```

Lecture 4: Operational Compliance with Ansible

Labs 8-10

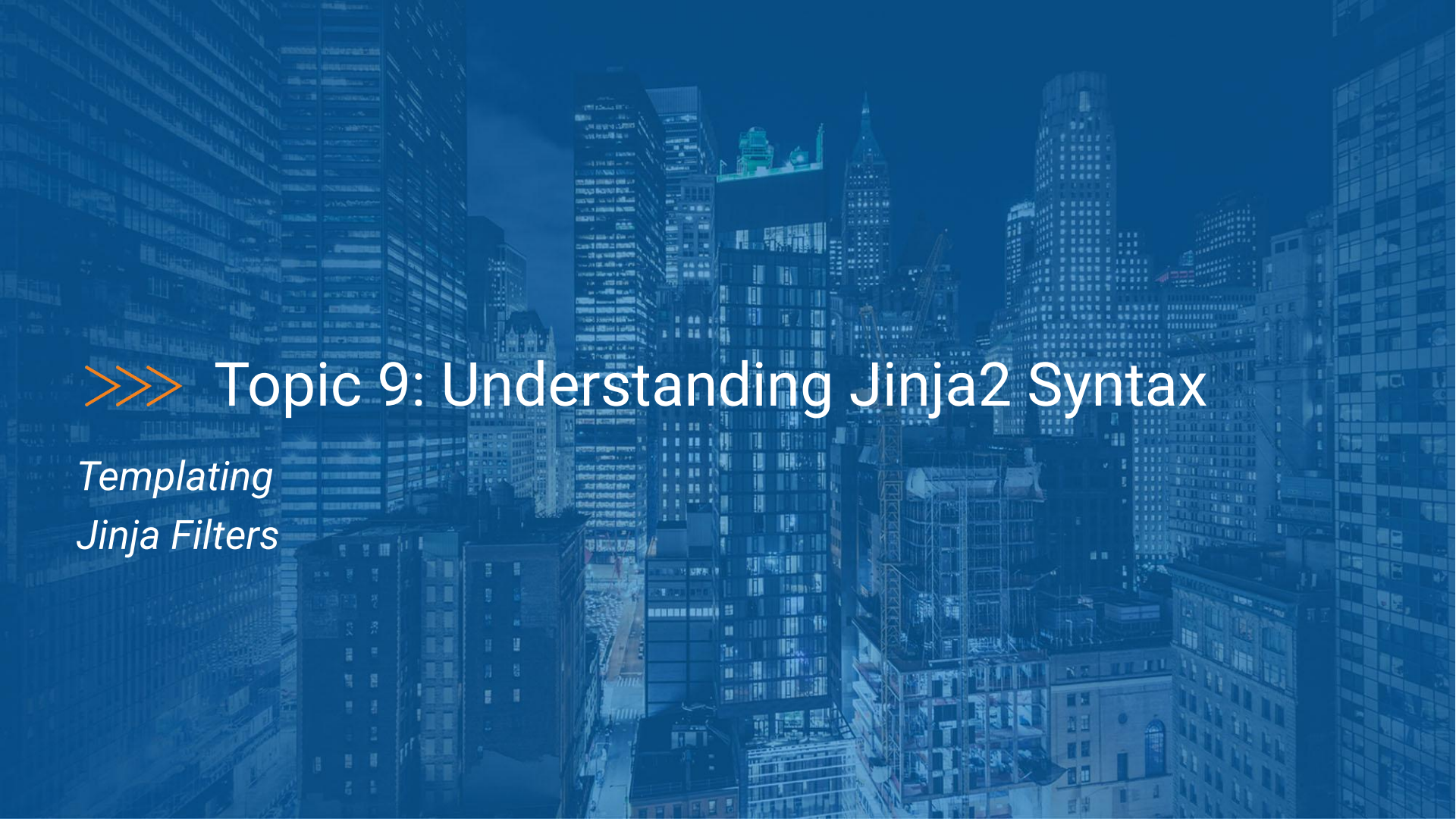
>>> Lab Time

- Lab 8 - Auto-Create Directories using the File Module
- Lab 9 - Getting Started with the Command Module
- Lab 10 - Continuous Compliance with Ansible

Lecture 5: Introducing Jinja2 Templating

Topic 9 - Understanding Jinja2 Syntax

Labs 11-12



Topic 9: Understanding Jinja2 Syntax

Templating

Jinja Filters

>>> Templating

- How do I automate multiple devices?
  - What if the IP addresses, VLANS or SNMP communities are different on each device?
- Manual method:
  - Enterprises have golden configuration standard for type of device, location of device etc
  - This is updated and deployed, for a new device coming into the network
- Automated templates:
  - Jinja2 provides a way to write these golden configs as skeleton templates with variables
  - Ansible provides the variables to the template and renders configuration using the template module.

>>> What is Jinja2?

Templating language for Python

- Each programming language has one or more templating language
- Used heavily within HTML programming too
- Double curly braces denote a variable {{ variable }}
  - Strings, Lists, Dictionaries (Just like we've seen in Ansible / Python)
  - Jinja2 is built for Python
- Supports conditionals, loops, and more functionality (lightweight programming)
  - Syntax changes to {% %} for conditionals and logic
- GOAL: Keep templates simple
- More info go to https://jinja.palletsprojects.com/

>>> Jinja2 Templating

Two components are required:

1. Jinja2 template
2. Variables (or data) to insert into the template

To expand or replace a section of code, use double curly brackets {{ }}

De-constructing configs/files into templates:

Example

```
snmp-server community PUBLIC RO
```

Text file

```
Dear John,
 Thanks for attending.
```

Example

```
snmp-server community {{ ro_string }} RO
```

Text file

```
Dear {{Name}},
 Thanks for attending.
```

>>> Jinja2 Basics

Accessing variables in Jinja2 templates:

Basic variable:

Template:

```
{{ hostname }}
```

Output:

```
NYCR01
```

Template - Loop through a list of strings:

```
{% for item in interfaces_list %}
  {{ item }}
{% endfor %}
```

Output:

```
  Eth1
  Eth2
  Eth3
```

Data Variables (inputs):

```
---
hostname: "NYCR01"
interfaces_list:
  - "Eth1"
  - "Eth2"
  - "Eth3"
```

>>> Jinja2 String concatenation

- Operator `+` follows Python rules
  - Type error will be raised if arguments are incompatible
- Use `~` operator when building strings
  - With `~` Jinja will cast arguments to string whenever possible

```
{{ 'Vlan' ~ 34 }}
```

```
Interface Vlan34
```

```
{{ 'Vlan' + 34 }}
```

```
      Interface {{ 'Vlan' + 34 }}
TypeError: can only concatenate str (not "int") to str
```

>>> Jinja2 Loops

Loop through a list of dictionaries:

Template:

```
{% for item in interfaces_list %}
  interface {{ item['name'] }}
    {{ item['state'] }}
{% endfor %}
```

Output:

```
  interface Eth1
    down
  interface Eth2
    up
  interface Eth3
    up
```

Data Variables (inputs):

```
---
  interfaces_list:
    - name: "Eth1"
      state: "down"
    - name: "Eth2"
      state: "up"
    - name: "Eth3"
      state: "up"
```

>>> Jinja2 set Directive

- Use the set directive to assign values to variables
- There are different ways to insert values into a script.
- One of the ways is to add it to our YAML files like we have been doing or to use the set directive statement to define the variable within the jinja2 template itself.

```
{% set install = 'new' %}

{% if install == 'new' %}
{% include 'new_install.conf' %}

{% else %}
{% include 'merge_install.conf' %}

{% endif %}
```

>>> Jinja2 Basics

Conditional Logic (if)

```
{% set install = 'new' %}
{% for item in interfaces_list %}
  interface {{ item['name'] }}
  {% if (item['state'] == "up") and
        (install == 'new') %}
    no shutdown
  {% elif (item['state'] == "down") and
        (install == 'new') %}
    shutdown
  {% endif %}
{% endfor %}
```

Output:

```
  interface Eth1
    shutdown
  interface Eth2
    no shutdown
  interface Eth3
    no shutdown
```

Data Variables (inputs):

```
---

  interfaces_list:
    - name: "Eth1"
      state: "down"
    - name: "Eth2"
      state: "up"
    - name: "Eth3"
      state: "up"
```

Use {# . . . #} to enclose comments

```
{# This code not needed, keep for
reference
    {% for vlan in vlans %}
        ...
    {% endfor %}
#}
```

>>> Jinja2 include Directive

- Use include statements to pull in other Jinja2 templates
- Included Jinja2 templates have access to the variables of the parent template

```
{% if install == 'new' %}
{% include 'new_install.conf' %}
{% else %}
{% include 'merge_install.conf' %}
{% endif %}
```

- ignore missing prevents Jinja2 from throwing an error if there is a missing file

```
{% include 'merge_install.conf' ignore missing %}
```

>>> The Ansible template module

The basic usage of the template module in Ansible:

```
- name: "RENDER CONFIGURATIONS"
  ansible.builtin.template:
    src: "device_config.j2"
    dest: "./configs/{{ inventory_hostname }}.cfg"
```

>>> Jinja2 and the *_config modules

Within the network core config modules (like cisco.ios.ios_config, junipernetworks.junos.junos_config, etc), you can specify a Jinja2 template as a src file rather than a config file.

```
- name: "ENSURE THAT SNMP IS CONFIGURED ON IOS DEVICES"
  cisco.ios.ios_config:
    src: 02-ios-snmp.j2
```

```
- name: "ENSURE THAT SNMP IS CONFIGURED ON JUNOS DEVICES"
  junipernetworks.junos.junos_config:
    src: 02-junos-snmp.j2
```

Jinja Filters

- Filters transform data within a parameter or Jinja Expression
- Are used with the operator | like `hostname | upper` will transform the hostname variable using the upper built-in filter to be uppercase
- Custom filters are possible, and Ansible has built-in filters in addition to Jinja2 built-in filters
- Complete list of Ansible filters here
- Easily create your own filter

```
vars:
  hostname: "nycr1"
  device_ip: "10.1.1.1"
  bad_ip: "X.10.Y.2"

tasks:
  - name: "CONVERT HOSTNAME TO UPPERCASE"
    ansible.builtin.debug:
      var: "hostname | upper"

  - name: "CHECK TO SEE IF AN IP ADDR IS VALID"
    ansible.builtin.debug:
      var: "device_ip | ipaddr"

  - name: "CHECK TO SEE IF AN IP ADDR IS VALID"
    ansible.builtin.debug:
      var: "bad_ip | ipaddr"
```

Sample Output:

```
"hostname | upper": "NYCR1"
"device_ip | ipaddr": "10.1.1.1"
"bad_ip | ipaddr": false
```

>>> Demo

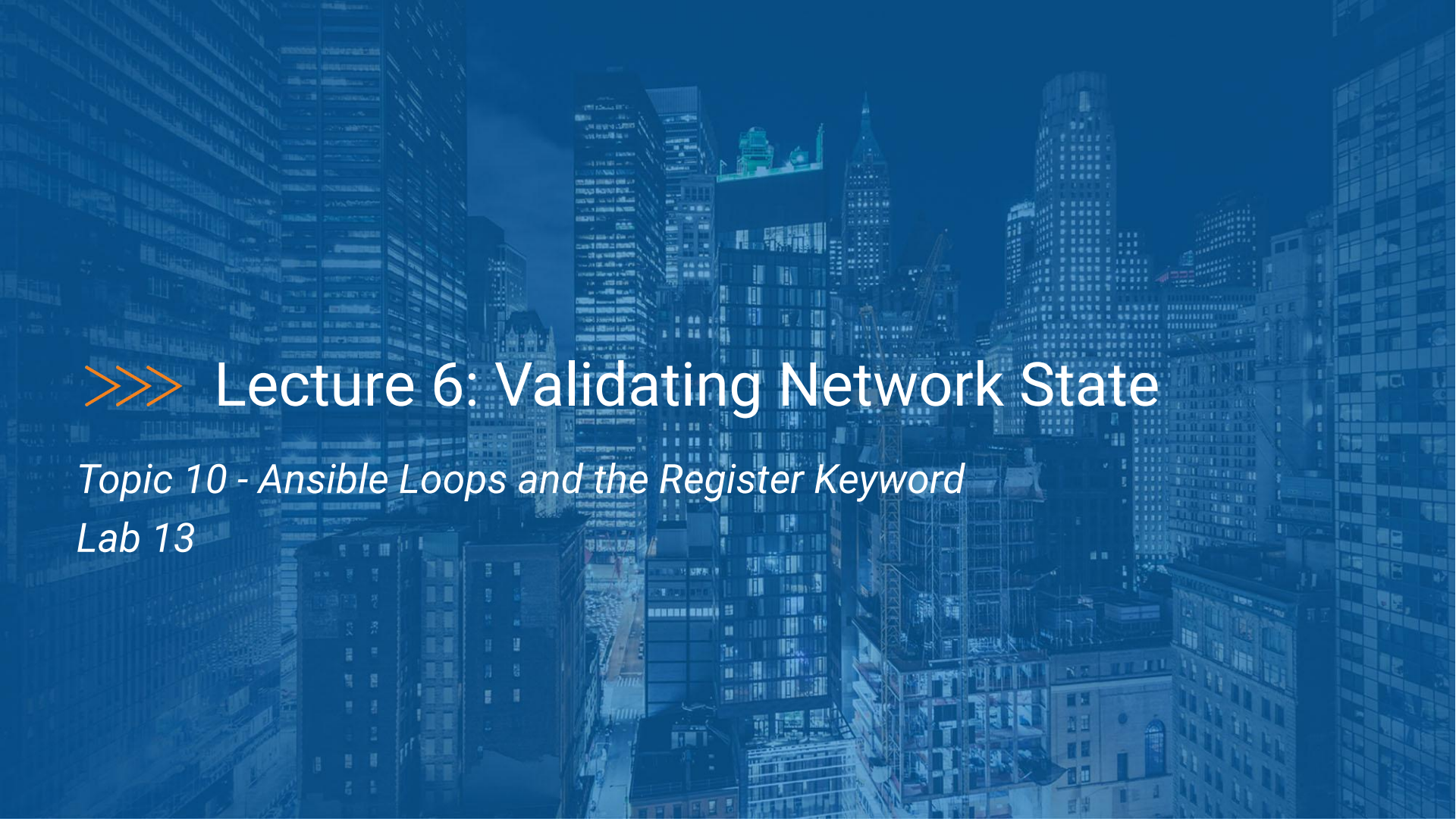
- Testing Jinja filters
- You do not need to create a Jinja template to test Jinja2 filters
- Just use the `ansible.builtin.debug` module!
- Learn about a few other helpful filters

Lecture 5: Introducing Jinja2 Templating

Labs 11-12

>>> Lab Time

- Lab 11 - Getting Started with Jinja2 Templates
- Lab 12 - Expanding the use of Jinja2 Templates



Lecture 6: Validating Network State

Topic 10 - Ansible Loops and the Register Keyword

Lab 13

Topic 10: Ansible Loops and the Register Keyword

Embedding Loops

Using the register keyword within a loop

>>> Embedding Loops within a task

- Iterate over a list of strings using the `loop` task attribute
- `item` is built-in variable equal to an element of the list as you're iterating
- In this case, `item` is a string

```
- name: "ITERATE OVER LIST OF STRINGS"
  hosts: "ios"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false
  vars:
    commands:
      - "show ip int brief"
      - "show version"
      - "show ip route"
  tasks:
    - name: "SEND A SERIES OF SHOW COMMANDS"
      cisco.ios.ios_command:
        commands: "{{ item }}"
      loop: "{{ commands }}"
```

>>> loop (cont'd)

- Iterate over a list of dictionaries. item is built-in variable equal to an element of the list as you're iterating. In this case, item is a dictionary.

```
---
- name: "ITERATE OVER LIST OF DICTIONARIES"
  hosts: "csr1"
  connection: "local"
  gather_facts: false
  vars:
    vlans:
      - id: 10
        name: "web_vlan"
      - id: 20
        name: "app_vlan"
      - id: 30
        name: "db_vlan"
  tasks:
    - name: "PRACTICE DEBUGGING WITH LOOPS"
      ansible.builtin.debug:
        msg: "The VLAN name is {{ item['name'] }} and the ID is {{ item['id'] }}"
      loop: "{{ vlans }}"
```

>>> dict2items filter to loop over dictionaries

- Iterate over a dictionary
- Root keys are item['key']
- Values are item['value']

```
---
  vars:
    locations:
      amer: "sjc-branch"
      apac: "hk-dc"
  tasks:
    - name: "PRINT ALL LOCATIONS"
      ansible.builtin.debug:
        msg: "Region is {{ item['key'] }} and Site is {{ item['value'] }}"
      loop: "{{ locations | dict2items }}"
```

- Sample output

```
"msg": "Regions are amer and Sites is sjc-branch"
"msg": "Regions are apac and Sites is hk-dc"
```

>>> Register (with loop)

- We saw earlier that register gives you access to the JSON data returned from a given module

```
- name: "SEND SHOW VERSION TO DEVICE"
  cisco.ios.ios_command:
    commands:
      - 'show version'
  register: "output"

- name: "TEST REGISTERED OUTPUT"
  ansible.builtin.debug:
    var: "output"

- name: "TEST GETTING SHOW DATA"
  ansible.builtin.debug:
    var: "output['stdout'][0]"
```

Remember, when registering the response from a *_command module, there are 4 keys and we primarily focused on `stdout`.

```
TASK [TEST REGISTERED OUTPUT]
*************************************************************
*********
ok: [csr1] => {
    "output": {
        "changed": false,
        "stdout": ["Cisco IOS XE Software, Version
16.08.01a\nCisco IOS Software [Everest], Virtual XE Software
(X86_64_LINUX_IOSD-UNIVERSALK9-M),Version 16.6.2, RELEASE
SOFTWARE (fc2)\nTechnical Support:
http://www.cisco.com/techsupport\nCopyright (c) 1986-2017 by
Cisco Systems, Inc.\nCompiled Wed 01-Nov-17 07:24 by
mcpre\n\n\nCisco IOS-XE software, Copyright (c) 2005-2017 by
cisco Systems, Inc.\nAll rights reserved.  Certain components
of Cisco IOS-XE software are\nlicensed under the GNU General
Public License (\"GPL\") Version 2.0. The\nsoftware code
licensed under GPL Version 2.0 is free software that
comes\nwith ABSOLUTELY NO WARRANTY.  You can redistribute
and/or modify such\nGPL code under the terms of GPL Version
2.0.  For more details, see the\ndocumentation or \"License
Notice\" file accompanying the IOS-XE software,\nor the
applicable URL provided on the flyer accompanying the
IOS-XE\nsoftware.\n\n\nROM: IOS-XE ROMMON\n\n  csr1 uptime is
30 - output omitted"],
        "stdout_lines": [], # list closed for slide
formatting
        "warnings": []
    }
}
```

>>> Register (cont'd)

Using `register` with `loop`

```
- name: "SEND PING COMMANDS TO DEVICES"
  cisco.ios.ios_command:
    commands: "ping {{ item }} repeat 2"
  register: "ping_responses"
  loop:
    - 8.8.8.8
    - 4.4.4.4
    - 198.6.1.4
- name: "TEST REGISTERED OUTPUT"
  ansible.builtin.debug:
    var: "ping_responses"
```

Now, there is a `results` key that is a list of dictionaries. Each dictionary contains the "standard" JSON data along with an `item` key to access the item that is being iterated over.

```
TASK [TEST REGISTERED OUTPUT] ************************************************
ok: [csr1] => {
    "ping_responses": {
        "changed": false,
        "msg": "All items completed",
        "results": [
            {
                "_ansible_ignore_errors": null,
                "_ansible_item_result": true,
                "_ansible_no_log": false,
                "_ansible_parsed": true,
                "changed": false,
                "failed": false,
                "invocation": {
                    "module_args": {
                        "auth_pass": null,
                        "authorize": null,
                        "commands": [
                            "ping 8.8.8.8 repeat 2"
                        ],
                        # omitted for brevity
                    }
                },
                "item": "8.8.8.8",
                "stdout": [
                    "Type escape sequence to abort.\nSending 2, 100-byte ICMP Echos
to 8.8.8.8, timeout is 2 seconds:\n!!\nSuccess rate is 100 percent (2/2), round-trip
min/avg/max = 2/2/2 ms"
                ],
                "stdout_lines": []
            }
            }]
```

>>> Register (cont'd)

- Since `items` is a key in the last slide and item is also a built-in variable when using loops, be cautious of `item['item']`
- `item` is the item in the list being iterated over

```
- name: "TEST LOOPING OVER REGISTERED VARIABLE"
  ansible.builtin.debug:
    msg: "{{ item }}"
  loop: "{{ ping_responses['results'] }}"
```

The inside key `item` is the IP address for that iteration, e.g. 8.8.8.8

```
- name: "TEST LOOPING OVER REGISTERED VARIABLE"
  ansible.builtin.debug:
    msg: "{{ item['item'] }}"
  loop: "{{ ping_responses['results'] }}"
```

Lecture 6: Validating Network State

Lab 13

>>> Lab Time

- Lab 13 - Validating Reachability with the Command Module

Lecture 7: Parsing Unstructured Data

Topic 11 - Parsing Response Data with Jinja2 Filters

Lab 14

Topic 11: Parsing Response Data with Jinja2 Filters

Introduce TextFSM

>>> Parsing Response Data

- When running commands use the core command module it is often necessary to parse needed information from the command response data.
- The following methods can be used to parse the response data:
- `parse_cli_textfsm`
- `regex_search`
- `regex_findall`
- You may notice that these methods are the same as methods used to parse data in Python.

>>> TextFSM Overview

- Python module for parsing semi-formatted text.
- Originally developed to allow programmatic access to information given by the output of CLI driven devices, such as network routers and switches
  - It can however be used for any such textual output.

>>> Using TextFSM

- The engine takes two inputs
  - Template file
  - Text input (such as command responses from the CLI of device)
- Returns a list of records that contains the data parsed from the text.
- Note: A template file is needed for each uniquely structured text input.

>>> Example 1: Text Input

show vlan (Arista EOS)

Filename: `arista_eos_show_vlan.raw`

```
VLAN  Name                              Status    Ports
----- --------------------------------- --------- ----------
1     default                           active    Et1
10    Test1                             active    Et1, Et2
20    Test2                             suspended
30    VLAN0030                          suspended
```

>>> Example 1: Template File

- `show vlan` (Arista EOS)
- Order is important
- Filename: arista_eos_show_vlan.template

```
Value VLAN_ID (\d+)
Value NAME (\w+)
Value STATUS (active|suspended)

Start
  ^${VLAN_ID}\s+${NAME}\s+${STATUS} -> Record
```

>>> Example 1: Executing textfsm

```
VLAN  Name                              Status    Ports
----- ---------------------------------- --------- --------
1     default                           active    Et1
```

```
ntc@jump-host$ python textfsm.py arista_eos_show_vlan.template
arista_eos_show_vlan.raw
FSM Template:
Value VLAN_ID (\d+)
Value NAME (\w+)
Value STATUS (active|suspended)

Start
  ^${VLAN_ID}\s+${NAME}\s+${STATUS} -> Record

FSM Table:
['VLAN_ID', 'NAME', 'STATUS']
['1', 'default', 'active']
['10', 'Test1', 'active']
['20', 'Test2', 'suspended']
['30', 'VLAN0030', 'suspended']
```

>>> Parsing Data Using parse_cli_textfsm

The `parse_cli_textfsm` Jinja2 filter can use the same `textfsm` templates that are used in Python to parse data in Ansible.

```
---
- name: "TEST PARSE USING PARSE_CLI_TEXTFSM"
  hosts: "csr1"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false
  vars:
    template_path: "/etc/ntc/ansible/library/ntc-ansible/ntc-templates/templates/"
    show_version_path: "{{ template_path }}cisco_ios_show_version.template"
  tasks:
    - name: "GET SHOW COMMANDS"
      cisco.ios.ios_command:
        commands: "show version"
      register: "config_data"
    - ansible.builtin.set_fact:
        show_version: "{{ config_data['stdout'][0] | parse_cli_textfsm(show_version_path) }}"
    - ansible.builtin.debug:
        var: "show_version"
```

>>> Parsing Data Using parse_cli_textfsm

You will see that `parse_cli_textfsm` will return structured data.

This is the output from the ansible.builtin.debug module on the previous slide:

```
TASK [debug] ***************************************************
ok: [csr1] => {
    "show_version": [
        {
            "CONFIG_REGISTER": "0x2102",
            "HARDWARE": [
                "CSR1000V"
            ],
            "HOSTNAME": "csr1",
            "ROMMON": "IOS-XE",
            "RUNNING_IMAGE": "packages.conf",
            "SERIAL": [
                "9KIBQAQ3OPE"
            ],
            "UPTIME": "6 hours, 18 minutes",
            "VERSION": "16.6.2"
        }
    ]
}
```

>>> NTC Templates

- Network to Code maintains the largest open source repository of TextFSM templates for network "show command" parsing
- They are broken down based on vendor and OS:
- https://github.com/networktocode/ntc-templates/tree/master/ntc_templates/templates

>>> Parsing Data Using regex filters

You can also use the `regex_search` and `regex_findall` Jinja2 filters to parse data. In this example we issued the ping command from an IOS device and want to parse out the success percentage. We are registering the response from the ping command to the `output` variable.

```
---
  - name: "PING TEST AND TRACEROUTE"
    hosts: "csr1"
    connection: "ansible.netcommon.network_cli"
    gather_facts: false
    vars:
      dest: "8.8.8.8"
    tasks:
    - name: "ISSUE PING"
      cisco.ios.ios_command:
        commands: "ping {{ dest }} repeat 2"
      register: "output"
```

>>> Parsing Data Using regex_search

Using `regex_search` we can parse the stdout from the `output` variable to find the success percentage.

```
- name: "PARSE PING RESPONSE TO OBTAIN % OF SUCCESS"
    ansible.builtin.set_fact:
      ping_pct: "{{ output['stdout'][0] | regex_search('Success rate is (\\d+)\\s+percent') }}"
```

`ping_pct` is equal to Success rate is 100 percent

```
- name: "PARSE PING RESPONSE TO OBTAIN % OF SUCCESS"
    ansible.builtin.set_fact:
      ping_pct: "{{ output['stdout'][0] | regex_search('Success rate is (\\d+)\\s+percent') | regex_search('(\\d+)') }}"
```

`ping_pct` is equal to "100"

The solution chains together two regex_search filters due to how the `regex_search` filter works. The first part returns everything matched within parentheses and the second captures the percentage within the first capture and saves that value in the `ping_pct` fact (variable).

>>> Parsing Data Using regex_findall

Using `regex_findall` is another way to parse the `stdout` from the output variable to find the success percentage.

```
- name: "PARSE PING RESPONSE TO OBTAIN % OF SUCCESS"
       ansible.builtin.set_fact:
         ping_pct: "{{ output['stdout'][0] | regex_findall('Success rate is (\\d+)\\s+percent') | first }}"
```

The filter works as you'd expect (unlike `regex_search`). It only returns what's inside parentheses (capture group), but it's always a list.

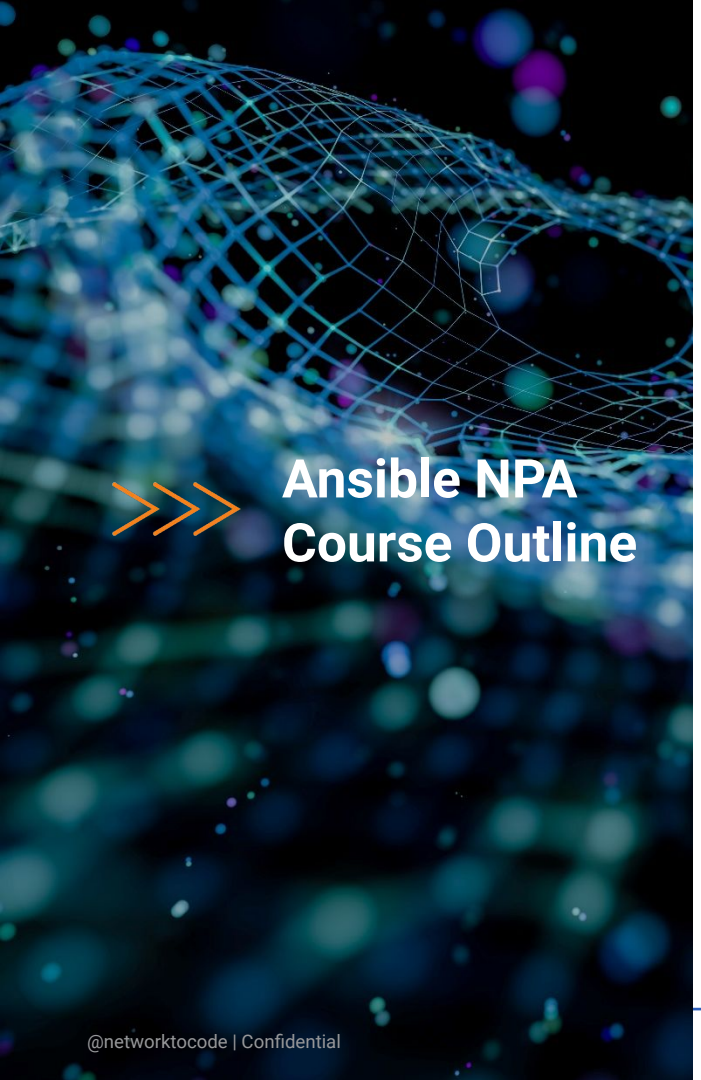
The `first` filter returns the first element in the list. This result is assigned to the variable `ping_pct`.

Lecture 7: Parsing Unstructured Data

Lab 14

>>> Lab Time

- Lab 14 - Performing a Conditional Traceroute Based on Ping Failures



Diving Deeper into Command and Config Modules

- *_config Modules
- diff_against Parameter
- Declarative Configuration
- Data Collection and Reporting
- Core *_facts Modules

Reusable and Repeatab

Lecture 8: A Deeper Look at Managing Configuration

Topic 12 - Exploring the Config Module Capabilities

Lab 15

Topic 12: Exploring the Config Module Capabilities

Introduce the parents, before, after and save_when parameters

Introduce the diff_against parameter

Making configuration changes in a declarative Way

>>> *_config

By default, it compares the lines against the running configuration

You can pass commands into the module a few different ways:

- Using the `lines` parameter
- Using the `src` parameter and point to a pre-built config file
- Using the `src` parameter and point to a Jinja2 template

```
---
- name: "DEPLOY SNMP COMMUNITY STRINGS ON IOS DEVICES"
  hosts: "ios"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false
  tasks:
    - name: "USE COMMANDS IN THE PLAYBOOK"
      cisco.ios.ios_config:
        lines:
          - "snmp-server community ntc123 ro"
    - name: "DEPLOY FROM CONFIG FILE"
      cisco.ios.ios_config:
        src: "configs/snmp.cfg"
```

>>> *_config

```
# Ensure these lines are present in the configuration
- cisco.nxos.nxos_config:
    commands:
      - "snmp-server community public group network-operator"
      - "snmp-server community networktocode group network-operator"
```

```
# Ensure these lines are present in the configuration
- junipernetworks.junos.junos_config:
    src: "snmp.conf"
```

>>> *_config (cont'd)

- Modules support many parameters
- Use `ansible-doc` to view them all
- `parents` - ordered list of commands that identify the section the commands should be checked against

```
- name: "ENSURE GIGE4 IS CONFIGURED PROPERLY"
  cisco.ios.ios_config:
    parents:
      - "interface GigabitEthernet4"
    lines:
      - "description Configured by Ansible"
      - "ip address 10.100.100.1 255.255.255.0"
```

>>> *_config (cont'd)

- Modules support many parameters
- Use `ansible-doc` to view all
- `parents` - ordered list of commands that identify the section the commands should be checked against
- `before` - ordered list of commands to prepend to lines if a change needs to be made
- `after` - ordered list of commands to append to lines if a change needs to be made

```
# this would remove any other commands previously covered for
# the other ASN
    - name: "ENSURE BGP CONFIG IS CORRECT"
      cisco.ios.ios_config:
        before: "['no router bgp 65512']
        parents:
          - router bgp 65512
        lines:
          - bgp router-id 10.10.10.10
          - bgp log-neighbor-changes
          - network 10.101.1.0 mask 255.255.255.0
          - network 10.101.2.0 mask 255.255.255.0
          - timers bgp 5 15
          - neighbor 10.10.10.2 remote-as 102
          - neighbor 10.10.10.2 description ISP_CARRIER_X
          - neighbor 10.10.10.2 send-community
          - neighbor 10.10.10.2 soft-reconfiguration inbound"
        after: "['copy run start']"
```

>>> *_config (cont'd)

The save_when parameter is needed to commit running config to the NVRAM. (Deprecated command save no longer works with Ansible 2.4 and above) Available options for the save_when parameter:

- always
- modified
- never

```
- name: "ENSURE THAT LOOPBACK 222 IS CONFIGURED"
  cisco.ios.ios_config:
    parents:
      - "interface loopback 222"
    commands:
      - "ip address 10.222.222.222 255.255.255.0"
    save_when: "modified"
```

>>> The diff_against Parameter

- Introduced in Ansible 2.4. Test **running configuration** against:
  - The startup configuration
    - Check if there are ephemeral configurations
  - A configuration intent
    - Check whether running configuration deviates from compliance/golden configuration
  - Pending configuration lines
    - Check exact configuration impact of config lines being pushed
    - Invoked with `--diff` flag

>>> diff_against - startup

```
- name: "COMPARE RUNNING CONFIG WITH STARTUP"
  cisco.ios.ios_config:
    diff_against: "startup"
```

```
TASK [COMPARE RUNNING CONFIG WITH STARTUP]
************************************
--- before
+++ after
@@ -36,6 +63,8 @@
 redundancy
 lldp run
 cdp run
+interface Loopback222
+ ip address 10.222.222.222 255.255.255.255
 interface GigabitEthernet1
  vrf forwarding MANAGEMENT
  ip address 10.0.0.51 255.255.255.0
```

>>> The lookup plugin

Powerful Ansible plugin that is used access data from outside sources :

- Regular text file content
- CSV
- INI
- DNS Lookup
- MongoDB and many more

Can be used to assign values to variables.

```
vars:
  config_file: "{{ lookup('file', './backups/' + inventory_hostname + '.cfg') }}"

tasks:
  - ansible.builtin.debug:
      msg: "The file name is {{ config_file }}"
```

```
ntc@jump-host:ansible$ ansible-playbook -i inventory file_lookup_demo.yml

PLAY [DEMO FILE LOOKUPS] ***************************************************

TASK [debug] ***************************************************************
ok: [csr1] => {
    "msg": "The file name is snmp-server community PUBLIC123 RO 5\nsnmp-server
community PRIVATE123 RW 95\nsnmp-server location GLOBAL\nsnmp-server contact
LOCAL_ADMIN\nsnmp-server host 1.1.1.1\n\nvlan 10\n name web_servers\nvlan
20\nvlan 30\n name db_servers"
}
```

>>> The lookup plugin (continued)

Find out what plugins are available with:

`“ansible-doc -t lookup -l”`

Get details with: `“ansible-doc -t lookup <plugin name>”`

Some values include:

- `config` Lookup current Ansible configuration values
- `dict` returns key/value pair items from dictionaries
- `env` read the value of environment variables
- `file` read file contents
- `first_found` return first file found from list
- `template` retrieve contents of file after templating with Jinja2
- `together` merges lists into synchronized list
- `varnames` Lookup matching variable names
- `vars` Lookup templated value of variables

>>> The lookup plugin with multiple results

Return a list of results with any of the following commands:

- `"lookup('file', './backups/{{ inventory_hostname }}.cfg', wantlist=True)"`
- `"query('file', './backups/{{ inventory_hostname }}.cfg')"`
- `q('file', './backups/{{ inventory_hostname }}.cfg')"`

Note the last 2 commands both imply wantlist = True

See https://docs.ansible.com/ansible/latest/plugins/lookup.html for more information

>>> diff_against - intended

```
  tasks:
    - name: "VALIDATE CONFIGURATION INTENT"
      cisco.ios.ios_config:
        diff_against: "intended"
        intended_config: "{{ lookup('file', './backups/{{ inventory_hostname }}.cfg') }}"
```

```
TASK [VALIDATE CONFIGURATION INTENT] ************************************
--- before
+++ after
@@ -63,6 +63,8 @@
 redundancy
 lldp run
 cdp run
+interface Loopback222
+ ip address 10.222.222.222 255.255.255.255
 interface GigabitEthernet1
  vrf forwarding MANAGEMENT
  ip address 10.0.0.51 255.255.255.0
```

>>> diff_against - impending configuration lines

```
- name: "ENSURE THAT LOOPBACK222 IS CONFIGURED"
  cisco.ios.ios_config:
    commands:
      - "ip address 10.222.222.222 255.255.255.255"
    parents:
      - "interface loopback 222"
    diff_against: "running"
```

```
TASK [ENSURE THAT LOOPBACK 222 IS CONFIGURED]
************************************
+++ after
@@ -63,6 +63,8 @@
 redundancy
 lldp run
 cdp run
+interface Loopback222
+ ip address 10.222.222.222 255.255.255.255
 interface GigabitEthernet1
  vrf forwarding MANAGEMENT
  ip address 10.0.0.51 255.255.255.0
```

>>> lineinfile

- Add/Remove a line to a file
- Use regexes to match for position
- Insert before/after

```
- name: "ENSURE A LINE DOES NOT EXIST"
  ansible.builtin.lineinfile:
    line: "Building configuration..."
    dest: "backups/{{ inventory_hostname }}.cfg"
    state: "absent"
```

```
- name: "ENSURE THE LINE THAT MATCHES THE REGEX DOES NOT EXIST"
  ansible.builtin.lineinfile:
    dest: "backups/{{ inventory_hostname }}.cfg"
    regexp: "Current configuration .*"
    state: "absent"
```

```
!
Building configuration...

Current configuration 3942 Bytes
!
hostname nycr01
```

>>> Declarative Configuration

- Data Model:

```
snmp_communities:
  - community: "ntc-public"
    group: "network-operator"
  - community: "ntc-private"
    group: "network-admin"
```

- Template for Nexus:

```
{% for snmp in snmp_communities %}
snmp-server community {{ snmp.community }} group {{ snmp.group }}
{% endfor %}
```

>>> Declarative Configuration (cont'd)

- Generate Configuration

```
---
- name: "DECLARATIVE CONFIGURATION"
  hosts: "nxos"
  connection: "ansible.netcommon.network_cli"
  gather_facts: False

  tasks:
    - name: "GENERATE CONFIGURATION"
      ansible.builtin.template:
        src: "./templates/snmp.j2"
        dest: "./snmp-config.cfg"
```

- snmp-config.cfg

```
snmp-server community ntc-public group network-operator
snmp-server community ntc-private group network-admin
```

>>> Declarative Configuration (cont'd)

- Push configuration

```
- name: "PUSH SNMP COMMUNITIES"
  cisco.nxos.nxos_config:
    src: "./snmp-config.cfg"
```

>>> Declarative Configuration (cont'd)

Get existing SNMP communities and set fact

```
    - name: "GET CONFIG FOR SNMP PARSING"
      cisco.nxos.nxos_command:
        commands:
          - "show run section snmp"
      register: "output"

    - name: "GET EXISTING SNMP COMMUNITIES AND SET FACT"
      ansible.builtin.set_fact:
        existing_snmp_communities: "{{ output['stdout'][0] | regex_findall('snmp-server community (\\S+)') }}"

    - ansible.builtin.debug:
        var: "existing_snmp_communities"
TASK [GET CONFIG FOR SNMP PARSING] *****************************************
ok: [nxos]
TASK [GET EXISTING SNMP COMMUNITIES AND SET FACT] ******************************
ok: [nxos]
TASK [debug] ***************************************************************
ok: [nxos] => {
    "existing_snmp_communities": [
        "ntc-public",
        "ntc-private",
        "public",
        "networktocode"
    ]
}
```

>>> Declarative Configuration (cont'd)

- **public** and **networktocode** communities are not part of the data model. Therefore, they may be undesired and need to be removed.

```
    - name: "SET FACT FOR PROPOSED (DESIRED) COMMUNITIES"
      ansible.builtin.set_fact:
        proposed_snmp_communities: "{{ snmp_communities|map(attribute='community')|list }}"

    - name: "CALCULATE AND SET FACT FOR COMMUNITIES TO REMOVE"
      ansible.builtin.set_fact:
        snmp_communities_to_remove: "{{
existing_snmp_communities|difference(proposed_snmp_communities) }}"

    - ansible.builtin.debug:
        var: "snmp_communities_to_remove"
```

```
TASK [SET FACT FOR PROPOSED (DESIRED) COMMUNITIES] ****************************
ok: [nxos]

TASK [CALCULATE AND SET FACT FOR COMMUNITIES TO REMOVE] ************************
ok: [nxos]

TASK [debug] ******************************************************************
ok: [nxos] => {
    "snmp_communities_to_remove": [
        "public",
        "networktocode"
    ]
}
```

>>> Declarative Configuration (cont'd)

- Remove undesired communities

```
    - name: "PURGE SNMP COMMUNITIES"
      cisco.nxos.nxos_config:
        commands:
          - "no snmp-server community {{ item }}"
      with_items: "{{ snmp_communities_to_remove }}"
```

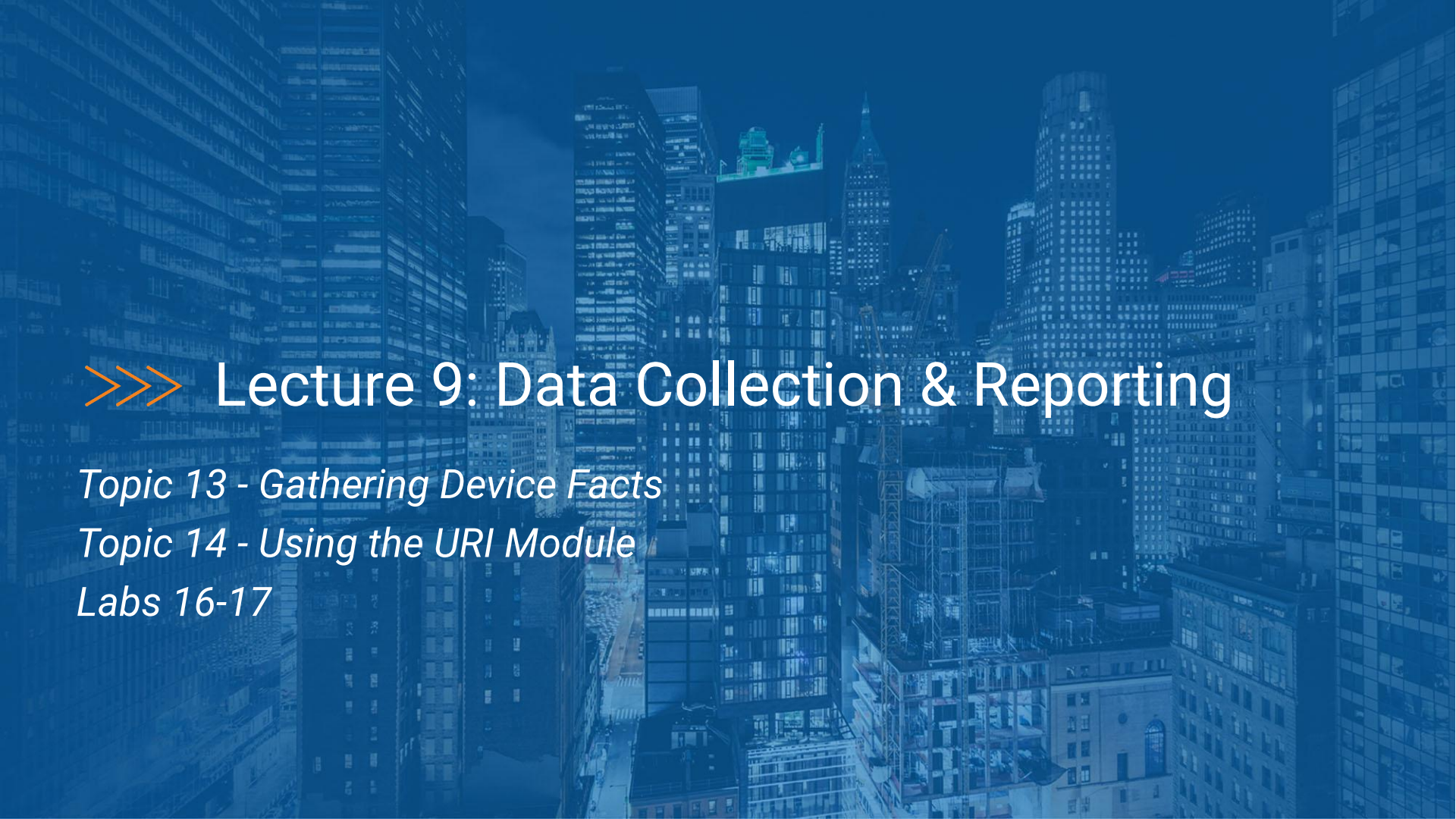
```
TASK [PURGE SNMP COMMUNITIES]
************************************************
changed: [nxos] => (item=public)
changed: [nxos] => (item=networktocode)
```

Lecture 8: A Deeper Look at Managing Configuration

Lab 15

>>> Lab Time

- Lab 15 - Exploring the Config Module



Lecture 9: Data Collection & Reporting

Topic 13 - Gathering Device Facts

Topic 14 - Using the URI Module

Labs 16-17

Topic 13: Gathering Device Facts

Gathering device data with the facts module

>>> Core *_facts Modules

- Ansible facts are data related to your remote systems. You can access this data in the ansible_facts variable. By default, you can also access some Ansible facts as top-level variables with the ansible_ prefix.
- Core facts modules collect a number of useful pieces of information:
  - All IP addresses
  - Filesystems
  - Hostname
  - Image
  - All interfaces with some Layer 2 and Layer 3 attributes
  - Serial Number
  - OS Version
  - Neighbors broken down by interface

https://docs.ansible.com/ansible/latest/user_guide/playbooks_vars_facts.html

>>> Collecting Facts

Sample playbook gathering IOS facts:

```
- name: "GATHER IOS FACTS"
  hosts: "ios"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false

  tasks:
    - name: "GET FACTS"
      cisco.ios.ios_facts:
```

>>> Collecting Facts

Sample playbook gathering IOS facts:

```
- name: "GATHER IOS FACTS"
  hosts: "ios"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false

  tasks:
    - name: "GET FACTS"
      cisco.ios.ios_facts:
```

Default value for gather_subset is **!config**

```
- name: "GATHER ALL FACTS"
  cisco.ios.ios_facts:
    gather_subset: "all"

- name: "GATHER A SHOW RUN AND DEFAULT SYSTEM FACTS"
  cisco.ios.ios_facts:
    gather_subset:
      - "config"

- name: "GATHER ALL FACTS EXCEPT HARDWARE FACTS"
 cisco.ios.ios_facts:
  gather_subset:
    - "!hardware"
```

>>> Collecting Facts

```
- name: "GATHER IOS FACTS"
  hosts: "ios"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false
  tasks:
    - name: "GET FACTS"
      cisco.ios.ios_facts:
      register: "ntc_ios_facts"
    - ansible.builtin.debug:
        var: "ntc_ios_facts"
```

>>> Sample Response (IOS)


```
"ntc_ios_facts": {
        "ansible_facts": {
            "ansible_net_all_ipv4_addresses": [
                "10.0.0.53"
            ],
            "ansible_net_all_ipv6_addresses": [],
            "ansible_net_filesystems": [
                "bootflash:"
            ],
            "ansible_net_hostname": "csr3",
            "ansible_net_image": "bootflash:packages.conf",
            "ansible_net_interfaces": {
                "GigabitEthernet1": {
                    "bandwidth": 1000000,
                    "description": null,
                    "duplex": "Full",
                    "ipv4": {
                        "address": "10.0.0.53",
                        "masklen": 24
                    },
                    "lineprotocol": "up ",
                    "macaddress": "2cc2.604c.4e06",
                    "mediatype": "RJ45",
                    "mtu": 1500,
                    "operstatus": "up",
                    "type": "CSR vNIC"
                }
            }
```

```
            "ansible_net_memfree_mb": 322777,
            "ansible_net_memtotal_mb": 2047264,
            "ansible_net_model": null,
            "ansible_net_neighbors": {
                "Gi1": [
                    {
                        "host": "eos-leaf1.ntc.com",
                        "port": "Management1"
                    },
                    {
                        "host": "eos-leaf2.ntc.com",
                        "port": "Management1"
                    }
                ]
            },
            "ansible_net_serialnum": "9KXI0D7TVFI",
            "ansible_net_version": "16.3.1"
        }
```

>>> Viewing Facts For a Device

Option 1:

```
- name: "GET FACTS"
  cisco.ios.ios_facts:
  register: "ntc_ios_facts"

- ansible.builtin.debug:
    var: "ntc_ios_facts"

- ansible.builtin.debug:
    var: "ntc_ios_facts['ansible_facts']['ansible_net_hostname']"
```

Option 2:

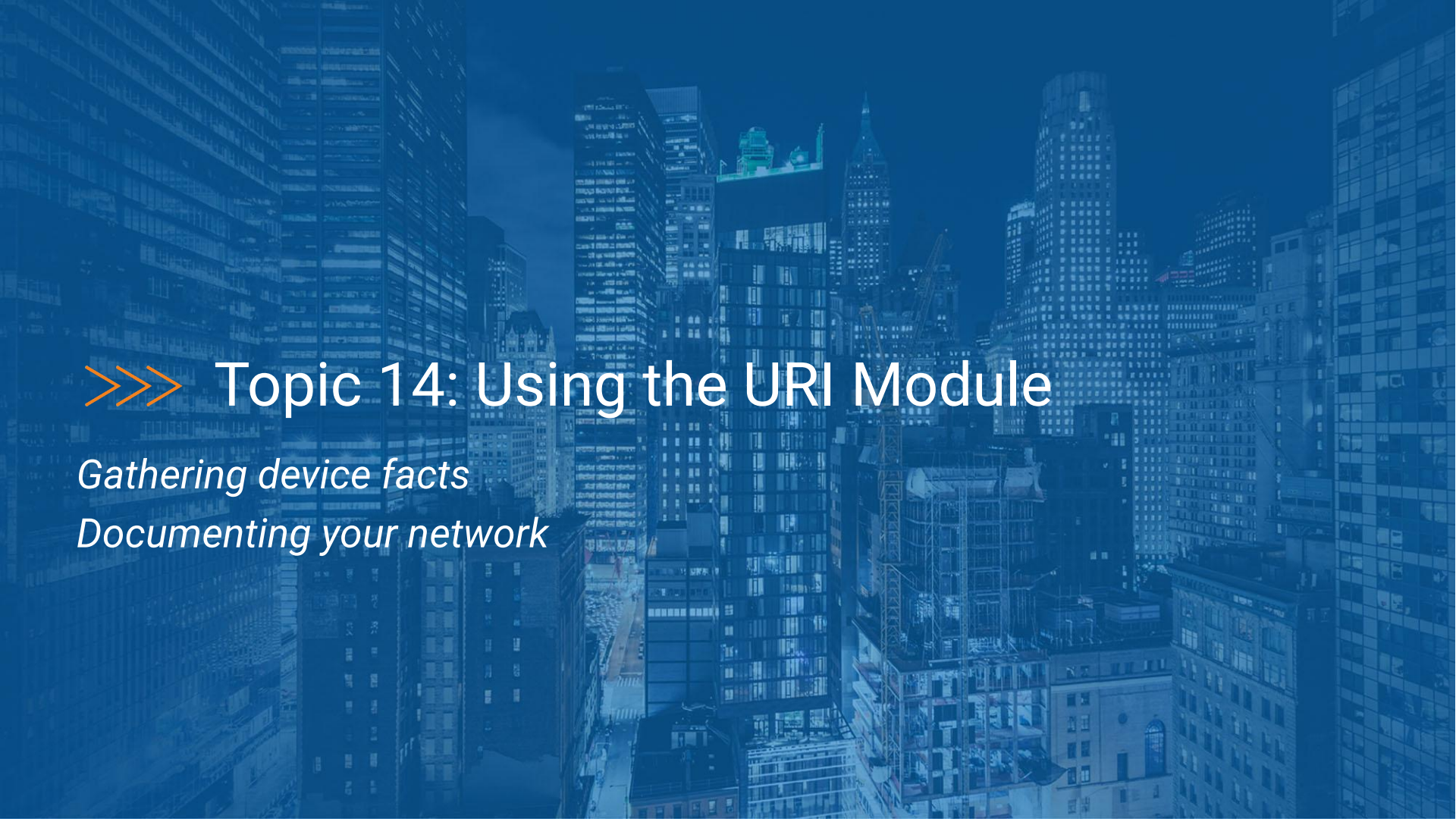
```
- name: "GET FACTS"
  cisco.ios.ios_facts:

- name: "Display variables/facts known for a given host"
  ansible.builtin.debug:
    var: "hostvars[inventory_hostname]"

- ansible.builtin.debug:
    var: "ansible_net_hostname"
```

Note: any key inside ansible_facts can be accessed directly

```
"ntc_ios_facts": {
   ansible_facts": {
     "ansible_net_all_ipv4_addresses": [
       "10.0.0.53"
     ],
     "ansible_net_all_ipv6_addresses":
[],
     "ansible_net_filesystems": [
       "bootflash:"
     ],
       "ansible_net_hostname": "csr3",
  }
}
```



Topic 14: Using the URI Module

Gathering device facts

Documenting your network

>>> Using the URI Module to Gather Device Facts

The `uri` module can be used to make HTTP-based API calls.

```
- name: "GET INTERFACE IP ADDRESS"
    uri:
      url: "https://{{inventory_hostname
}}/restconf/data/Cisco-IOS-XE-native:native/interface=GigabitEthernet/1/ip/address"
      method: GET
      user: "{{ ansible_user }}"
      password: "{{ ansible_ssh_pass }}"
      return_content: yes
      validate_certs: no
      headers:
        Content-Type: "application/yang-data+json"
        Accept: "application/yang-data+json"
      register: "response"
```

This example shows how to make an API call against an IOS device to pull the IP address information for the `GigabitEthernet1` interface and assigning the returned value to the `response` variable.

You can test all of these settings using Postman before building your Ansible tasks.

>>> Creating Documentation

You choose:

- text
- html
- markdown
- asciidoc
- …

Then:

- publish
- alert
- chatops
- mail

>>> Know the Available Facts/Variables

- Template

```
# general.j2
Device: {{ inventory_hostname }}

Vendor:           {{ ntc_vendor }}
Platform:         {{ platform }}
Operating System: {{ ansible_network_os }}
Image:            {{ ansible_net_image }}
```

- Playbook

```
---
- name: "DC P1"
  hosts: "nxos-spine1"
  connection: "local"
  gather_facts: false
  tasks:
    - cisco.nxos.nxos_facts:
    - ansible.builtin.template:
        src: "general.j2"
        dest: "files/general.md"
```

>>> Know the Available Facts/Variables

- Template

```
# general.j2
Device: {{ inventory_hostname }}

Vendor:           {{ ntc_vendor }}
Platform:         {{ platform }}
Operating System: {{ ansible_network_os }}
Image:            {{ ansible_net_image }}
```

- Playbook

```
---
- name: "DC P1"
  hosts: "nxos-spine1"
  connection: "local"
  gather_facts: false
  tasks:
    - nxos_facts:
    - template:
        src: "general.j2"
        dest: "files/general.md"
```

- Document

```
Device: nxos-spine1
Vendor:           cisco
Platform:         NX-OSv Chassis
Operating System: nxos
Image:
bootflash:///titanium-d1.7.3.1.D1.0.10.bin
```

>>> Documenting Neighbors

- Template

```
# neighbors.j2

DEVICE: {{ inventory_hostname }}
{%  for local_int, details in
ansible_net_neighbors.items() %}
LOCAL INTERFACE:    {{ local_int }}
{% for neigh_data in details %}
NEIGHBOR:           {{ neigh_data['sysname'] }}
NEIGHBOR INTERFACE: {{ neigh_data['port'] }}

{% endfor %}
{% endfor %}
```

- Playbook

```
---
- name: "DC P1"
  hosts: "nxos-spine1"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false
  tasks:
    - name: "LLDP NEIGHBORS"
      cisco.nxos.nxos_facts:

    - name: "BUILD TABLE"
      ansible.builtin.template:
        src: "neighbors.j2"
        dest: "files/neighbors.md"
```

- Document

```
DEVICE: nxos-spine1

LOCAL INTERFACE:    Ethernet2/3
NEIGHBOR:           nxos-spine2(TB604B14E3B)
NEIGHBOR INTERFACE: Ethernet2/3

LOCAL INTERFACE:    Ethernet2/4
NEIGHBOR:           nxos-spine2(TB604B14E3B)
NEIGHBOR INTERFACE: Ethernet2/4

LOCAL INTERFACE:    mgmt0
NEIGHBOR:           nxos-spine2(TB604B14E3B)
NEIGHBOR INTERFACE: mgmt0

LOCAL INTERFACE:    Ethernet2/2
NEIGHBOR:           nxos-spine2(TB604B14E3B)
NEIGHBOR INTERFACE: Ethernet2/2

LOCAL INTERFACE:    Ethernet2/1
NEIGHBOR:           nxos-spine2(TB604B14E3B)
NEIGHBOR INTERFACE: Ethernet2/1
```

>>> Using a Table (Neighbors)

- Template

```
# neighbors-table.j2
| Source     | Interface    |         Neighbor          |
Interface   |
| ----------
|--------------|--------------------------|-------------|
{% for local_int, details in ansible_net_neighbors.items() %}
{% for neigh_data in details %}
| {{ inventory_hostname }}|{{ local_int }} | {{
neigh_data['sysname'] }} | {{ neigh_data['port'] }} |
{% endfor %}
{% endfor %}
```

- Playbook

```
---
- name: "DC P1"
  hosts: "nxos-spine1"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false
  tasks:
    - name: "LLDP NEIGHBORS"
      cisco.nxos.nxos_facts:

    - ansible.builtin.template:
        src: "neighbors.j2"
        dest: "files/neighbors-table.md"
```

>>> Using a Table (Neighbors)

Template

```
# neighbors-table.j2
| Source | Interface  | Neighbor | Interface |
| -------|------------|----------|-----------|
{% for local_int, details in ansible_net_neighbors.items()
%}
{% for neigh_data in details %}
| {{ inventory_hostname }}|{{ local_int }} | {{
neigh_data['sysname'] }} | {{ neigh_data['port'] }} |
{% endfor %}
{% endfor %}
```

Playbook

```
---
- name: "DC P1"
  hosts: "nxos-spine1"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false
  tasks:
    - name: "LLDP NEIGHBORS"
      cisco.nxos.nxos_facts:
    - ansible.builtin.template:
        src: "neighbors.j2"
        dest: "files/neighbors-table.md"
```

Markdown generated table

Source	Interface	Neighbor	Interface
nxos-spine1	Ethernet2/4	nxos-spine2(TB604B14E3B)	Ethernet2/4
nxos-spine1	Ethernet2/3	nxos-spine2(TB604B14E3B)	Ethernet2/3
nxos-spine1	Ethernet2/2	nxos-spine2(TB604B14E3B)	Ethernet2/2
nxos-spine1	Ethernet2/1	nxos-spine2(TB604B14E3B)	Ethernet2/1

>>> Summary

- Understand the various ways to collect data about the network.
- Templating is great for documentation (not just configurations)
- Know your variables
  - Any variable can be used in the playbook and template
- Know your document formats
- Understand Jinja2
- Auto publish to github, web server, etc.
- Key modules:
  - `ansible.builtin.template`
  - `ansible.builtin.assemble`

Lecture 9: Data Collection & Reporting

Labs 16-17

>>> Lab Time

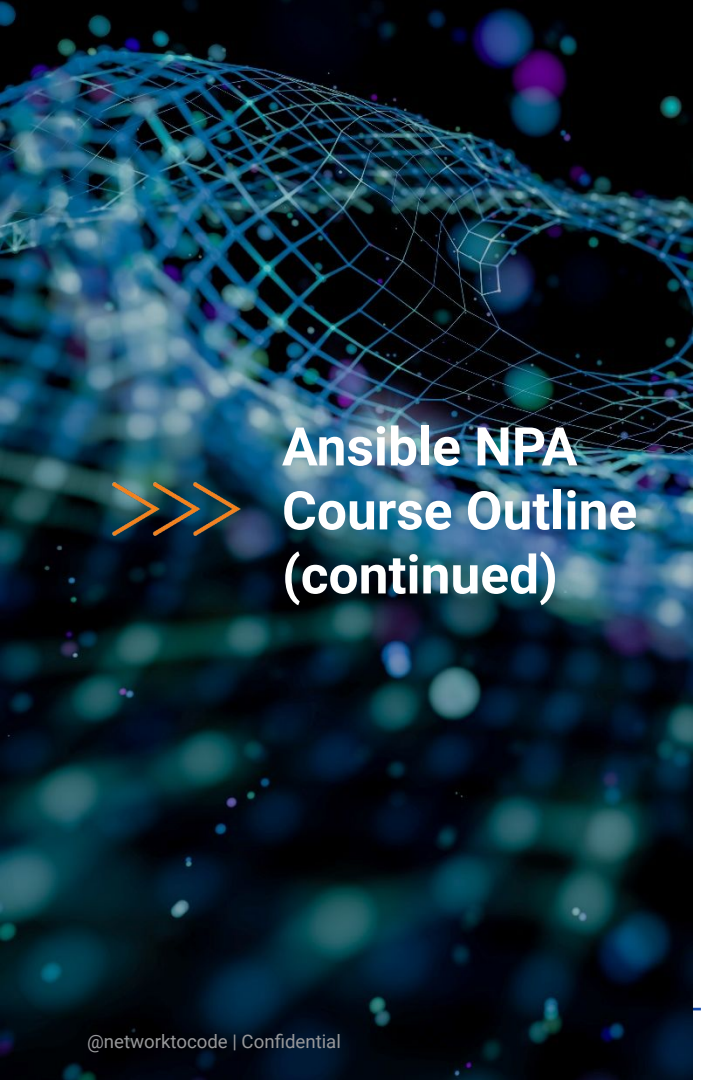
- Lab 16 - Making REST API Calls with Ansible
- Lab 17 - Data Collection Modules & Reporting

Lecture 10: Managing Ansible Content

Topic 15 - Ansible Collections

Topic 16 - Ansible Roles

Lab 18



Diving Deeper into Command and Config Modules

- *_config Modules
- diff_against Parameter
- Declarative Configuration
- Data Collection and Reporting
- Core *_facts Modules

Reusable and Repeatable Code

- Ansible Collections
- Ansible Roles
- Ansible Dynamic Inventories
- Third Party Modules

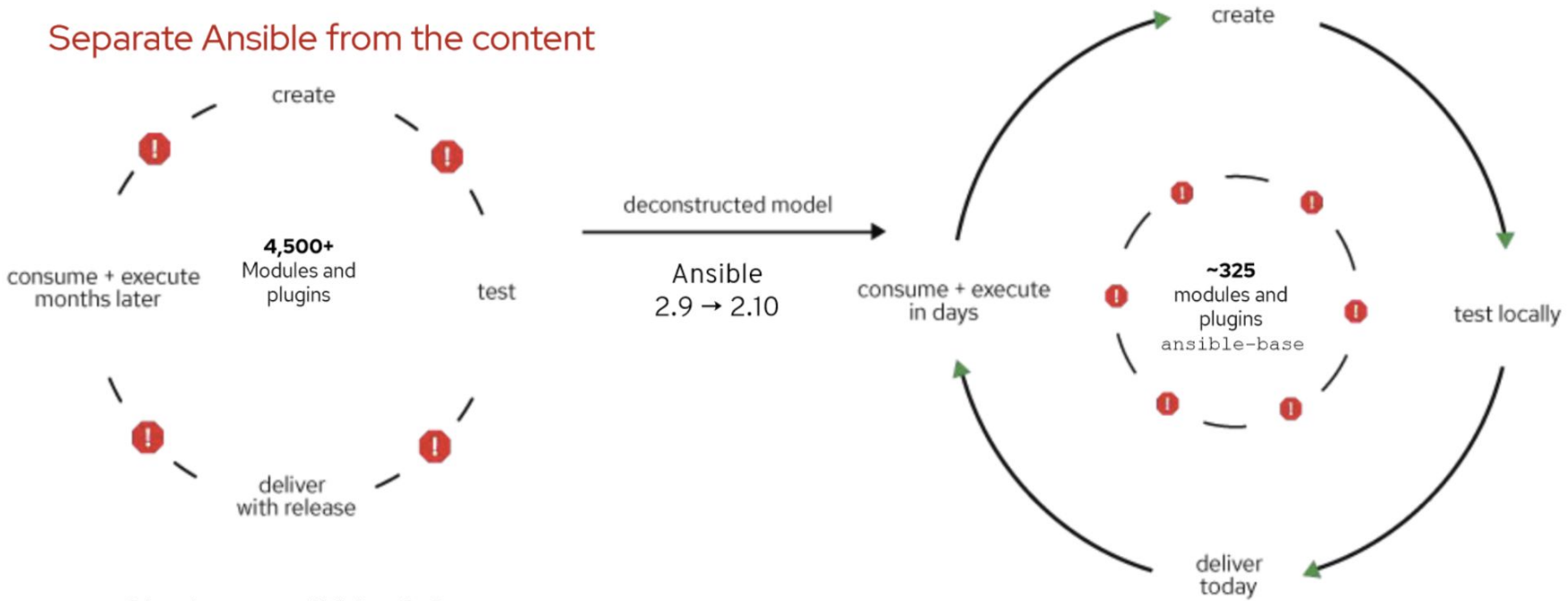
Topic 15: Ansible Collections

Understand how to work with Ansible Content Collections

>>> Using Ansible Content Collections

- Introduced in Ansible 2.8, usable in 2.9, getting better in 2.10+
- Collections are a packaging format for Ansible related content
  - Modules and Plugins (filters, connection, inventory, etc.)
  - Roles, Playbooks (including Templates, Files, Vars, etc.)
- Fully Qualified Collection Name (FQCN)
  - namespace.collection.content_name
  - e.g., cisco.ios.ios_config instead of the former ios_config only
- Public central repository at https://galaxy.ansible.com
  - Can also install collections from Git repositories
- CLI tool: ansible-galaxy
  - Install, List, Verify Collections on the Ansible control host
  - If the collection is not present, install it (e.g. ansible-galaxy collection install cisco.ios)
  - Ansible Documentation on Using Collections

>>> Why Ansible collection?



Single monolithic: 3-6 months release cycle

inner circle: slower release cycle (ansible-base)
outer circle: faster release cycle (content in collections)

https://github.com/networktocode/interop2020-slides/blob/main/interop-2020-Ganesh%20Nalawade%20-%20Deep%20Dive%20on%20Ansible%20Collections.pdf

>>> Collections are the “future of Ansible content delivery”

The Inside Playbook

The Future of Ansible Content Delivery

July 23, 2019 by Dylan Silva

>>> Ansible component name changes

"Core" product name	ansible	ansible-base	ansible-core
Software version	2.9.0	2.10.0	2.11.0
Release date	2019-10-31	2020-09-22	2021-04-26
# of modules	4,500+	~325	~325
"Community" name	ansible	community	community
Software version	2.9.0	3.0.0	4.0.0
Release date	2019-10-31	2021-02-16	2021-05-18
# of modules	Included above	4,000+	4,000+

>>> Reusable Abstractions

- `ansible.builtin.include` statement
- includes tasks from another file

```
# main playbook
---
- name: "PB"
  hosts: "all"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false
  tasks:
    - ansible.builtin.include: "get-facts.yml"
```

```
---
# get-facts.yml
- name: "GET FACTS FROM ARISTA DEVICES"
  arista.eos.eos_facts:

  ...<more getter tasks>...
```

>>> Parameterized Include

- Pass parameters to included tasks “when” the expression is True

```
# main playbook
---
- name: "CONFIGURE DEVICES"
  hosts: "all"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false
  tasks:
    - ansible.builtin.include: "get-facts.yml vendor={{ ntc_vendor }}"
```

```
# get-facts.yml
---
- name: "GET FACTS FROM ARISTA DEVICES"
  arista.eos.eos_facts:
  when: "vendor == 'arista'"

- name: "GET FACTS FROM CISCO NXOS DEVICES"
  cisco.nxos.nxos_facts:
  when: "vendor == 'cisco'"
```

Topic 16: Ansible Roles

Introduce using and creating roles

>>> Roles not in Collections

- Reusable abstraction of code (tasks, variables, and handlers)
- Ansible Galaxy shares roles

```
ntc@ntc$ pwd
/home/ntc/ansible
ntc@ntc$ mkdir roles
ntc@ntc$ ansible-galaxy role -h
usage: ansible-galaxy role [-h] ROLE_ACTION ...

positional arguments:
  ROLE_ACTION
      init        Initialize new role with the base structure of a role.
      remove      Delete roles from roles_path.
      delete      Removes the role from Galaxy. It does not remove or alter the actual GitHub repository.
      list        Show the name and version of each role installed in the roles_path.
      search      Search the Galaxy database by tags, platforms, author and multiple keywords.
      import      Import a role into a galaxy server
      setup       Manage the integration between Galaxy and the given source.
      info        View more details about a specific role.
      install     Install role(s) from file(s), URL(s) or Ansible Galaxy

optional arguments:
  -h, --help    show this help message and exit
```

>>> Roles not in Collections

```
ntc@ntc$ ansible-galaxy init role  roles/vlans
- Role roles/vlans was created successfully
ntc@ntc$ tree
.
└── roles
      └── vlans
      ├── README.md
      ├── defaults
      │   └── main.yml
      ├── files
      ├── handlers
      │   └── main.yml
      ├── meta
      │   └── main.yml
      ├── tasks
      │   └── main.yml
      ├── templates
      ├── tests
      │   ├── inventory
      │   └── test.yml
      └── vars
            └── main.yml

10 directories, 8 files
```

>>> Roles not in Collections

- Reusable abstraction of code (tasks, variables, and handlers)
- Ansible Galaxy shares roles

```
site.yml
inventory
roles/
   common/
     files/
     templates/
     tasks/
     handlers/
     vars/
     defaults/
   vlans/
     templates/
     tasks/
       main.yml
       eos.yml
       nxos.yml
       junos.yml
     vars/
   snmp/
     templates/
     tasks/
     vars/
```

```yaml
---
- name: "SAMPLE PLAYBOOK USING ROLES"
  hosts: "leaves"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false
  roles:
     - "common"
     - "vlans"
```

>>> Parameterized Roles

```
---
- hosts: "spine"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false
  roles:
    - "common"
    - "snmp"
    - role: "vlan"
      vlan_id: 10
    - role: "snmp"
      contact: "Bob"
```

>>> VLAN Role

```
# main playbook
---
- name: "DC P1"
  hosts: "datacenter"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false
  roles:
    - "vlans"

# roles/vlans/tasks/main.yml
---
- name: "ARISTA VLANs"
  arista.eos.eos_vlan:
    vlanid: "{{ item['id'] }}"
  loop: "{{ vlans }}"
  when: "vendor == 'arista'"
- name: "CISCO VLANs"
  cisco.nxos.nxos_vlan:
    vlan_id: "{{ item['id'] }}"
  loop: "{{ vlans }}"
  when: "vendor == 'cisco'"
```

```
[datacenter]
spine1 vendor=arista
n9k1   vendor=cisco
```

```
# group_vars/all.yml
---
vlans:
  - id: 10
    name: "web_servers"
  - id: 20
  - id: 30
    name: "db_servers"
```

>>> VLAN Role

```
# main playbook
---
- name: "DC P1"
  hosts: "datacenter"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false
  roles:
    - "vlans"
```

```
# roles/vlans/tasks/main.yml
---
- ansible.builtin.include: "{{ vendor }}.yml"
```

```
# roles/vlans/tasks/arista.yml
---
- name: "ARISTA VLANs"
  arista.eos.eos_vlan:
    vlanid: "{{ item['id'] }}"
  loop: "{{ vlans }}"
```

```
---
# roles/vlans/tasks/cisco.yml
- name: "CISCO VLANs"
  cisco.nxos.nxos_vlan:
    vlan_id: "{{ item['id'] }}"
  loop: "{{ vlans }}"
```

```
[datacenter]
spine1 vendor=arista
n9k1   vendor=cisco
```

```
# group_vars/all.yml
---
vlans:
  - id: 10
    name: "web_servers"
  - id: 20
  - id: 30
    name: "db_servers"
```

>>> Roles within Collections

```
$ pwd
/home/ntc/ansible
ntc@ntc$ tree
.

0 directories, 0 files
ntc@ntc$ mkdir mynamespace
ntc@ntc$ cd mynamespace
ntc@ntc$ pwd
/home/ntc/ansible/mynamespace
ntc@ntc$ ansible-galaxy collection init vlans
ERROR! Invalid collection name 'vlans', name must be in the format <namespace>.<collection>.
Please make sure namespace and collection name contains characters from [a-zA-Z0-9_] only.
```

>>> Roles within Collections

```
$ ansible-galaxy collection init mynamespace.vlans
- Collection mynamespace.vlans was created successfully
ntc@ntc$ tree
.
└── mynamespace
      └── vlans
      ├── README.md
      ├── docs
      ├── galaxy.yml
      ├── plugins
      │   └── README.md
      └── roles

5 directories, 3 files
```

>>> Summary

- Think about the functions, features, or applications of the device
- Big Picture vs. Details
- Encapsulate
- For networking - think multi-vendor features

Lecture 10: Managing Ansible Content

Lab 18

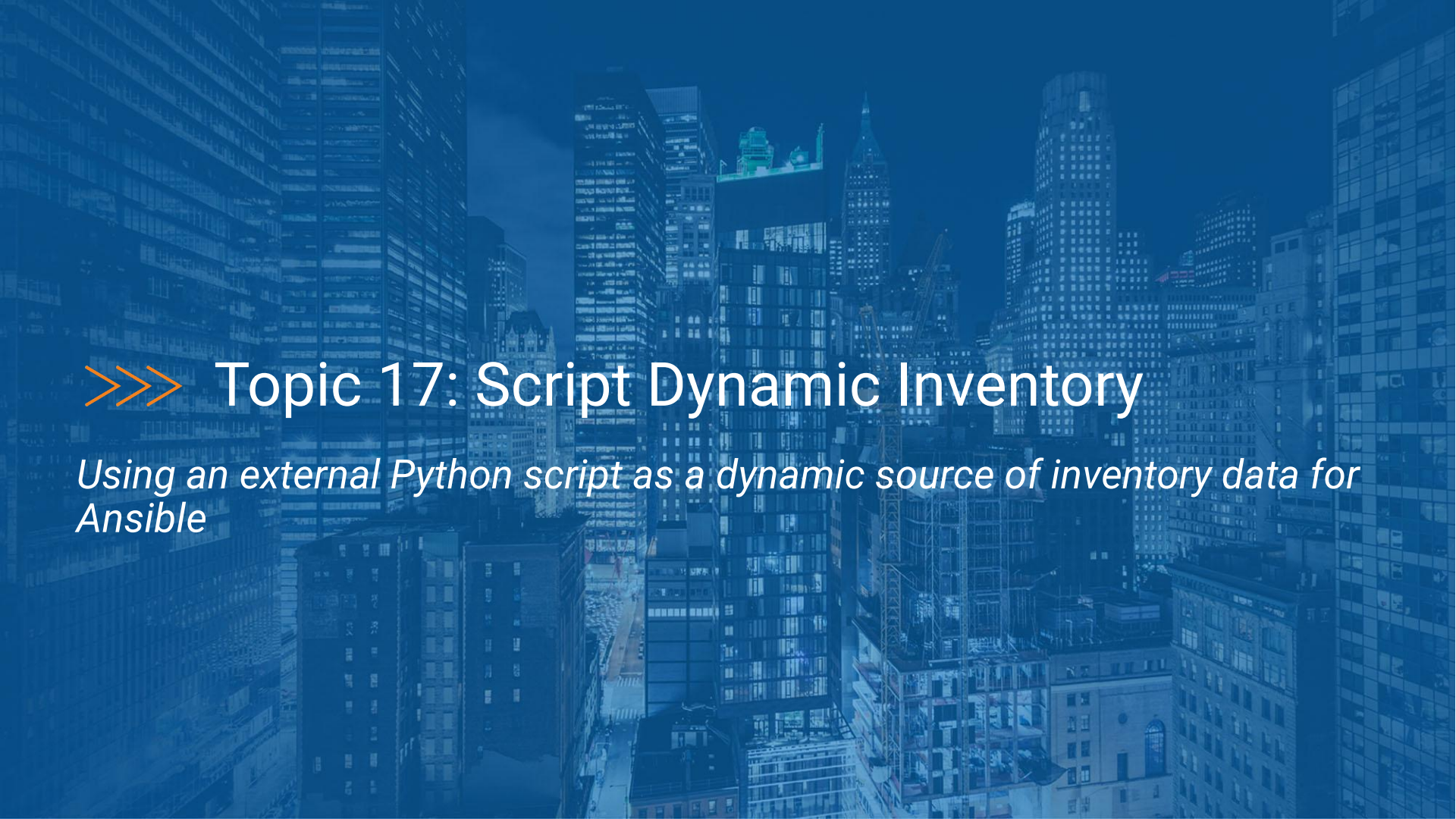
>>> Lab Time

- Lab 18 - Getting Started with Roles

Lecture 11: Ansible Dynamic Inventories

Topic 17 - Script Dynamic Inventory

Lab 19



Topic 17: Script Dynamic Inventory

Using an external Python script as a dynamic source of inventory data for Ansible

>>> Inventory

- Ansible Inventory files are static
- Managing large YAML files doesn't scale
- Manage lots of YAML files doesn't scale
- Users usually already have a CMDB, NMS, etc.
- What if you have a dynamic environment?
  - AWS, Rackspace, VMs
- **Enter Dynamic Inventory**

>>> Executing Playbooks

You currently use the -i flag to specify the inventory file - You can also specify a script that is used to generate an inventory - needs to return JSON k/v pairs - `ansible-playbook -i dynamic_script.py site.yml`

Script needs to be an executable (x)]

```
{
    "hp": {
        "hosts": [
            "hp1.ntc.com",
            "hp2.ntc.com",
            "hp3.ntc.com",
            "hp4.ntc.com"
        ],
        "vars": {
            "platform": "comware7"
        }
    },
    "cisco": {
        "hosts": [
            "n9k1.ntc.com",
            "n9k2.ntc.com"
        ],
        "vars": {
            "platform": "nexus"
        }
    },
    "juniper": [
        "jnprfw.ntc.com"
    ],
    "arista": [
        "arista1.ntc.com",
        "arista2.ntc.com"
    ]
}
```

>>> Multiple Sources

- If the parameter used with the -i flag is a directory, multiple sources can be used
  - inventory file with scripts
  - multiple scripts

>>> Example Script

```
#!/usr/bin/env python

import requests

import json

requests.packages.urllib3.disable_warnings()

def main():

    url = 'https://2z3oa80l2c.execute-api.us-east-1.amazonaws.com/prod/switch'

    inventory = requests.get(url, verify=False)

    return inventory.text

if __name__ == "__main__":

    rsp = main()

    print (rsp)
```

>>> Executing a Playbook

```
---
- name: "TEST PLAYBOOK FOR DYNAMIC INVENTORY"
  hosts: "all"
  connection: "local"
  gather_facts: false
  tasks:
    - ansible.builtin.debug:
        var: "inventory_hostname"
```

Viewing the Results

JSON data generated from script

```
{
    "cisco": {
        "hosts": [
            "n9k1.ntc.com",
            "n9k2.ntc.com"
        ],
        "vars": {
            "platform": "nexus"
        }
    },
    "arista": [
        "arista1.ntc.com",
        "arista2.ntc.com"
    ],
    "apic": [
        "aci.ntc.com"
    ]
}
```

Playbook output

```
ansible-playbook -i dynamo.py site.yml

PLAY [TEST PLAYBOOK FOR DYNAMIC INVENTORY]
*****************************************

TASK: [debug var=inventory_hostname]
*****************************************
ok: [n9k2.ntc.com] => {
    "var": {
        "inventory_hostname": "n9k2.ntc.com"
    }
}
ok: [n9k1.ntc.com] => {
    "var": {
        "inventory_hostname": "n9k1.ntc.com"
    }
}
ok: [arista1.ntc.com] => {
    "var": {
        "inventory_hostname":
"arista1.ntc.com"
    }
}
...output omitted
```

>>> Summary

- YAML files do the trick most of the time, especially for testing
- Custom dev work required for dynamic network inventories, i.e. HPOV, Solar Winds, etc.
- As with modules, can be done in any language
  - Just need to return JSON in the proper format

Lecture 11: Ansible Dynamic Inventories

Lab 19

>>> Lab Time

- Lab 19 - Dynamic Inventory

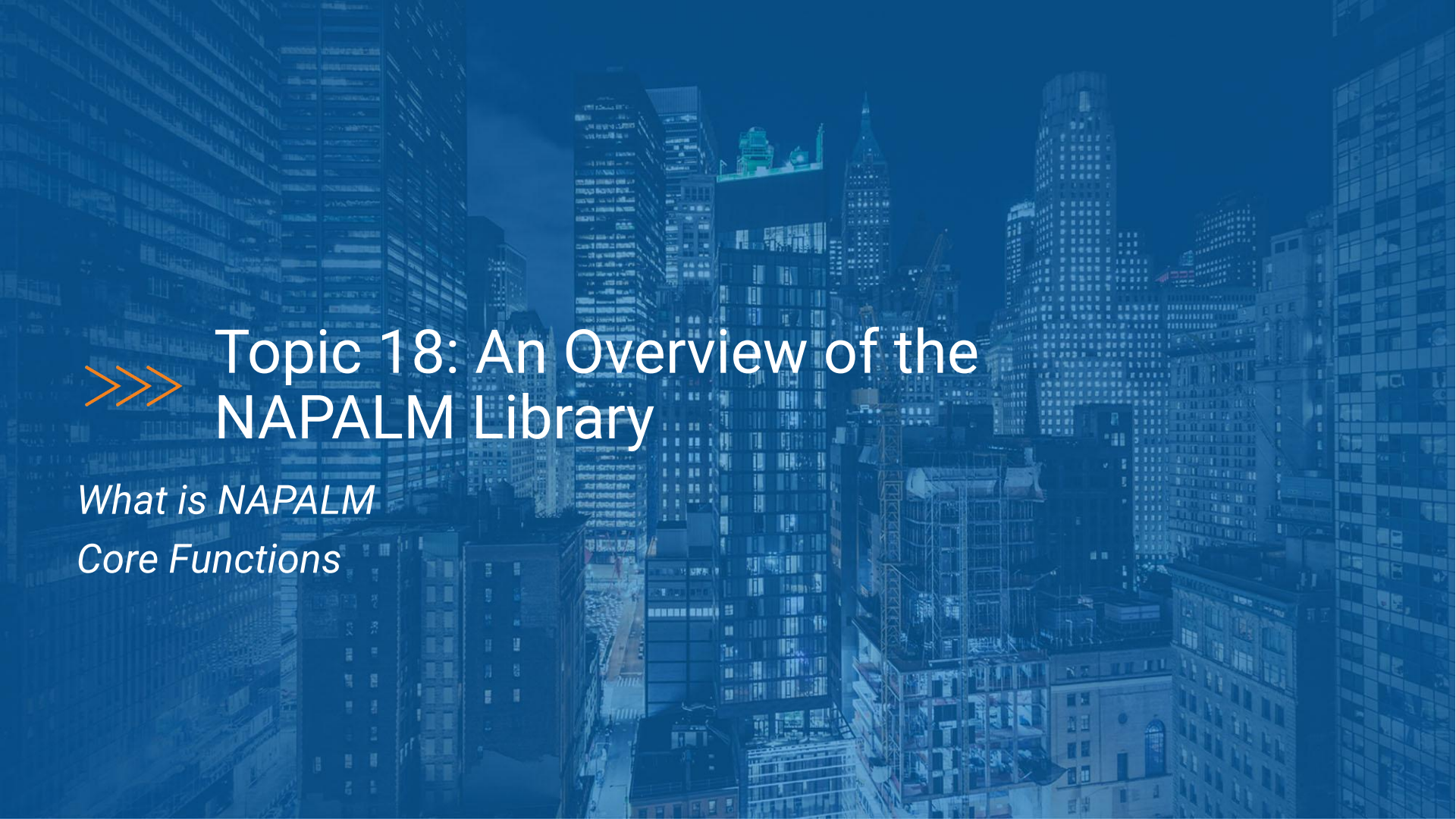
Lecture 12: Third Party Modules

Topic 18 - An Overview of the NAPALM Library

Topic 19 - cisco.nxos. modules*

Topic 20 - junipernetworks.junos. modules*

Labs 20-21



Topic 18: An Overview of the NAPALM Library

What is NAPALM

Core Functions

>>> NAPALM

NAPALM (Network Automation and Programmability Abstraction Layer with Multivendor support) is a Python library that implements a set of functions to interact with different network device Operating Systems using a unified API.

NAPALM supports several methods to connect to the devices, to manipulate configurations or to retrieve data.

Also has associated Ansible Modules

https://napalm.readthedocs.io/en/latest/

https://github.com/napalm-automation/napalm-ansible

To install run either:

pip install napalm-ansible

pip install napalm

Or:

git clone https://github.com/napalm-automation/napalm-ansible

pip install napalm

>>> NAPALM

The following modules are currently available:

- `napalm_cli`
- `napalm_diff_yang`
- `napalm_get_facts`
- `napalm_install_config`
- `napalm_parse_yang`
- `napalm_ping`
- `napalm_translate_yang`
- `napalm_validate`

>>> napalm-ansible/collection/galaxy.yml - now required with Collections

```
---
namespace: napalm
name: napalm
version: 0.9.13
readme: README.md
authors:
 - Kirk Byers
 - Mircea Ulinic
 - David Barroso
description: NAPALM + Ansible Modules
license_file: LICENSE
tags:
 - networking
 - napalm
repository: https://github.com/napalm-automation/napalm-ansible
documentation: https://github.com/napalm-automation/napalm-ansible
homepage: https://github.com/napalm-automation/napalm-ansible
issues: https://github.com/napalm-automation/napalm-ansible/issues
build_ignore:
 - "*.tar.gz"
 - "build.sh"
```

>>> NAPALM Support Matrix

- Arista EOS
- Cisco IOS
- Cisco IOS-XR
- Cisco NX-OS
- Juniper JunOS

>>> NAPALM

Three core functions:

- Retrieving Data
- Declarative Configuration Management
- Deployment Validation

All three are done in a uniform and vendor-neutral fashion

>>> NAPALM Facts

Network Device Facts


```
{
    "os_version": "4.15.2F-2663444.4152F",
    "uptime": 5837,
    "interface_list": [
        "Ethernet1",
        "Ethernet2",
        "Ethernet3",
        "Ethernet4",
        "Ethernet5",
        "Ethernet6",
        "Ethernet7",
        "Management1"
    ],
    "vendor": "Arista",
    "serial_number": "",
    "model": "vEOS",
    "hostname": "eos-spine1",
    "fqdn": "eos-spine1.ntc.com"
}
```

>>> Interfaces


```
{
    "Management1": {
        "is_enabled": true,
        "description": "",
        "last_flapped": 1467419703.0212176,
        "is_up": true,
        "mac_address": "2c:c2:60:0d:52:90",
        "speed": 1000
    },
    "Ethernet2": {
        "is_enabled": true,
        "description": "",
        "last_flapped": 1467419702.7812023,
        "is_up": true,
        "mac_address": "2c:c2:60:12:98:52",
        "speed": 1000
    },
    "Ethernet3": {
        "is_enabled": true,
        "description": "",
        "last_flapped": 1467419702.7812028,
        "is_up": true,
        "mac_address": "2c:c2:60:60:20:9b",
        "speed": 1000
    },
"Ethernet1": {
        "is_enabled": true,
        "description": "",
        "last_flapped": 1467419702.781203,
        "is_up": true,
        "mac_address": "2c:c2:60:48:80:70",
        "speed": 1000
    },
    "Ethernet5": {
        "is_enabled": true,
        "description": "",
        "last_flapped": 1467419702.8092043,
        "is_up": true,
        "mac_address": "2c:c2:60:40:8d:10",
        "speed": 1000
    },
    "Ethernet4": {
        "is_enabled": true,
        "description": "",
        "last_flapped": 1467419702.7692015,
        "is_up": true,
        "mac_address": "2c:c2:60:2e:c6:f8",
        "speed": 1000
    }
}
```

>>> NAPALM Configuration Management

Two main ways to manage device configurations with NAPALM

Configuration Replace

- Declarative configuration always pushing the full configuration
- Only commands required to get the device into its intended state are applied
- No “negation (no)” commands are sent to the device

Configuration Merge

- Send a set of commands or configuration stanza
- Only commands required to get the device into its intended state are applied
- You can use the merge for declarative management on a stanza based on OS

It does vary based on operating system.

>>> NAPALM Configuration Management

Example Workflow

Works slightly different than based on individual drivers and operating systems.

1. Connect to Device
2. Copy desired configuration to device (checkpoint file/rollback, candidate configuration, config session, bootflash as candidate_config.txt)
3. Use a vendor command to view diffs
4. Use a vendor command apply configuration changes
5. Optionally, rollback to a config that exists in the file system.

Note: you dictate if the supplied configuration is a full config file or partial configuration

>>> Configuration Replace

Focus on desired configuration commands.

There are no no commands used. The underlying OS generates the diffs (for most NAPALM drivers).

```
$ more diffs/csr1.diffs
+hostname csr1
-hostname csr_old_name
-interface Loopback100
 -ip address 1.1.1.1 255.255.255.255
-interface Loopback200
 -ip address 22.2.1.1 255.255.255.255
-ip route 10.1.1.0 255.255.255.0 192.0.1.1
```

Full configuration is sent to the device, but only diffs are applied. You do not need to worry about going from A to B - you just focus on B.

>>> Configuration Merge

You can use NAPALM for declarative management for a sectional config too.

Current BGP Config

```
router bgp 65512
   neighbor 10.0.0.0 remote-as 65500
   neighbor 10.0.0.0 maximum-routes 12000
   neighbor 10.0.0.1 remote-as 65512
   neighbor 10.0.0.1 maximum-routes 12000
   network 20.20.20.0/24
!
```

Desired BGP Config (file sent to device)

```
router bgp 65512
   neighbor 10.0.0.2 remote-as 65500
   neighbor 10.0.0.2 maximum-routes 12000
   neighbor 10.0.0.1 remote-as 65512
   neighbor 10.0.0.1 maximum-routes 12000
   neighbor 10.0.0.10 remote-as 65512
   network 100.0.100.0/24
!
```

Configuration Merge

You can use NAPALM for declarative management for a sectional config too.

Current BGP Config

```
router bgp 65512
   neighbor 10.0.0.0 remote-as 65500
   neighbor 10.0.0.0 maximum-routes 12000
   neighbor 10.0.0.1 remote-as 65512
   neighbor 10.0.0.1 maximum-routes 12000
   network 20.20.20.0/24
!
```

Desired BGP Config (file sent to device)

```
router bgp 65512
   neighbor 10.0.0.2 remote-as 65500
   neighbor 10.0.0.2 maximum-routes 12000
   neighbor 10.0.0.1 remote-as 65512
   neighbor 10.0.0.1 maximum-routes 12000
   neighbor 10.0.0.10 remote-as 65512
   neighbor 10.0.0.10 remote-as maximum-routes 12000
   network 100.0.100.0/24
!
```

Diff Generated by NAPALM

```
neighbor 10.0.0.0 maximum-routes 12000
    neighbor 10.0.0.1 remote-as 65512
    neighbor 10.0.0.1 maximum-routes 12000
+   neighbor 10.0.0.2 remote-as 65500
+   neighbor 10.0.0.2 maximum-routes 12000
+   neighbor 10.0.0.10 remote-as 65512
+   neighbor 10.0.0.10 maximum-routes 12000
    network 20.20.20.0/24
+   network 100.0.100.0/24
 !
 management api http-commands
    protocol http
```

>>> Configuration Merge (Advanced)

You can use NAPALM for declarative management for a sectional config too.

Current BGP Config

```
router bgp 65512
   neighbor 10.0.0.0 remote-as 65500
   neighbor 10.0.0.0 maximum-routes 12000
   neighbor 10.0.0.1 remote-as 65512
   neighbor 10.0.0.1 maximum-routes 12000
   network 20.20.20.0/24
!
```

Desired BGP Config (file sent to device)

```
no router bgp 65512
router bgp 65512
   neighbor 10.0.0.2 remote-as 65500
   neighbor 10.0.0.2 maximum-routes 12000
   neighbor 10.0.0.1 remote-as 65512
   neighbor 10.0.0.1 maximum-routes 12000
   neighbor 10.0.0.10 remote-as 65512
   network 100.0.100.0/24
!
```

Diff Generated by NAPALM

```
router bgp 65512
-    neighbor 10.0.0.0 remote-as 65500
-    neighbor 10.0.0.0 maximum-routes 12000
     neighbor 10.0.0.1 remote-as 65512
     neighbor 10.0.0.1 maximum-routes 12000
-    network 20.20.20.0/24
+    neighbor 10.0.0.2 remote-as 65500
+    neighbor 10.0.0.2 maximum-routes 12000
+    neighbor 10.0.0.10 remote-as 65512
+    neighbor 10.0.0.10 maximum-routes 12000
+    network 100.0.100.0/24
 !
```

Be cautious of device support. This is based on NAPALM driver implementation which is dictated by vendor OS support. This example is EOS.

>>> NAPALM Ansible Module

Parameter	Required	Default	Choices	Comments
username	yes			Username
dev_os	yes			OS running the device
config_file	yes			Where to load the configuration from
hostname	yes			IP or FQDN of the device you want to connect to
replace_config	no			If set to True the entire configuration on the device will be replaced during the commit If set to False, we will merge the new config with the existing one. Default is False
diff_file	no			A file where to store the “diff” between the running configuration and the new configuration. If it’s not set the diff between configurations is not saved
password	yes			Password
commit_changes	yes			If set to True the configuration will be actually replaced. If the set to False. We will not apply the changes, just check the difference

>>> Backup Example

NAPALM Backup

```
- name: "BACKUP CONFIG NAPALM"
  napalm_get_facts:
    dev_os: "{{ ansible_network_os }}"
    provider: "{{ connection_details }}"
    filter:
      - "config"

- name: "STORE BACKUP IN FILE"
  copy:
    content: "{{ ansible_facts['napalm_config']['running']}}"
    dest: "./backups/{{ inventory_hostname }}-napalm.cfg"
```

Core Backup

```
- name: "BACKUP CONFIG CORE"
    cisco.ios.ios_config:
      backup: True
```

- Backup Parameter

  This argument will cause the module to create a full backup of the current `running-config` from the remote device before any changes are made.

  The backup file is written to the `backup` folder in the playbook root directory or role rootdirectory, if playbook is part of an ansible role. If the directory does not exist, it is created. [Default: no] type: bool version_added: 2.2

Topic 19: cisco.nxos.* modules

Exploring NXOS collection modules: facts, vlan, feature, file_copy, portchannel, vpc

>>> Cisco Nexus Modules - prefix module names with cisco.nxos.

nxos_aaa_server_host.py nxos_hsrp.py nxos_pim.py nxos_udld_interface.py
nxos_aaa_server.py nxos_igmp_interface.py nxos_pim_rp_address.py nxos_udld.py
nxos_acl_interface.py nxos_igmp.py nxos_ping.py nxos_vlan.py
nxos_acl.py nxos_igmp_snooping.py nxos_portchannel.py nxos_vpc_interface.py
nxos_bgp_af.py nxos_install_os.py nxos_reboot.py nxos_vpc.py
nxos_bgp_neighbor_af.py nxos_interface_ospf.py nxos_rollback.py nxos_vrf_af.py
nxos_bgp_neighbor.py nxos_interface.py nxos_smu.py nxos_vrf_interface.py
nxos_bgp.py nxos_ip_interface.py nxos_snapshot.py nxos_vrf.py
nxos_command.py nxos_mtu.py nxos_snmp_community.py nxos_vrrp.py
nxos_config.py nxos_ntp_auth.py nxos_snmp_contact.py nxos_vtp_domain.py
nxos_evpn_global.py nxos_ntp_options.py nxos_snmp_host.py nxos_vtp_password.py
nxos_evpn_vni.py nxos_ntp.py nxos_snmp_location.py nxos_vtp_version.py
nxos_facts.py nxos_nxapi.py nxos_snmp_traps.py nxos_vxlan_vtep.py
nxos_feature.py nxos_ospf.py nxos_snmp_user.py nxos_vxlan_vtep_vni.py
nxos_file_copy.py nxos_ospf_vrf.py nxos_static_route.py nxos_gir_profile_management.py
nxos_overlay_global.py nxos_switchport.py nxos_gir.py nxos_pim_interface.py

>>> nxos_facts

Gathers facts from Nexus devices:

- fan_info
- hostname
- uptime
- interfaces
- last reboot reason - operating system
- vlan_list

```
tasks:
    - cisco.nxos.nxos_facts:
```

>>> nxos_vlan

- Manages VLAN resources on a Nexus switch
- Parameters:
  - `admin_state`
  - `name`
  - `vlan_id`
  - `vlan_state`

```
# ensure vlan 110 exists with a name configured
- cisco.nxos.nxos_vlan:
    vlan_id: 110
    name: "DB_VLAN"
```

>>> nxos_feature

- Manages features on a Nexus switch
- Parameters:
  - feature

```
# ensure eigrp is disabled
- cisco.nxos.nxos_feature:
    feature: "eigrp"
    state: "disabled"
# ensure lacp is enabled
- cisco.nxos.nxos_feature:
    feature: "lacp"
    state: "enabled"
```

>>> nxos_file_copy

- Copies a local file to bootflash (by default) of NXOS device using SCP
- Parameters
  - `dest_file`
  - `source_file`

```
# copy latest NX-OS to NXOS switch
- cisco.nxos.nxos_file_copy:
    source_file: "/home/cisco/Downloads/nxos.7.0.3.I2.1.bin"
```

>>> nxos_portchannel

- Manage port-channel interfaces on Nexus devices
- Parameters:
  - `group`
  - `min_links`
  - `members (list)`
  - `mode`
- Complex args as a parameter must use YAML format

```yaml
# ensure port-channel 100 exists
- cisco.nxos.nxos_portchannel:
    group: 100
    min_links: 2
    members:
      - "Ethernet1/28"
      - "Ethernet1/29"
```

>>> nxos_vpc

- Manages global VPC configuration
- Parameters:
  - domain
  - role_priority
  - system_priority
  - pkl_src
  - pkl_dest
  - pkl_vrf
  - peer_gw
  - auto_recovery
  - delay_restore

```
# ensure port-channel 100 exists
- cisco.nxos.nxos_vpc:
    domain: 100
    role_priority: 1000
    system_priority: 2000
    pkl_dest: 192.168.100.4
    pkl_src: 10.1.100.20
    peer_gw: true
    auto_recovery: true
```

>>> nxos_vpc_interface

- Manages interface VPC configuration
- Parameters:
  - portchannel
  - vpc
  - peer_link

```
# ensure port-channel 100 exists
- cisco.nxos.nxos_vpc_interface:
    portchannel: 10
    vpc: 100
```

Topic 20: junipernetworks.junos.* modules

Exploring Junos collection modules: commit, rollback, get_config, install_config, install_os

Juniper Modules - prefix module names with `junipernetworks.junos.`

* `junos_commit` — Commit candidate configuration on device.
* `junos_get_config` — Retrieve configuration of device.
* `junos_get_facts` — Retrieve device-specific information from the host.
* `junos_install_config` — Modify the configuration of a device running Junos OS.
* `junos_install_os` — Install a Junos OS software package.
* `junos_rollback` — Rollback configuration of device.
* `junos_shutdown` — Shut down or reboot a device running Junos OS.
* `junos_srx_cluster` — Enable/Disable cluster mode for SRX devices
* `junos_zeroize` — Remove all configuration information and reset all key values on a device.
* `junos_cli` - Execute CLI command on device and save file locally
* `junos_rpc` - Execute Junos RPC on device and save file locally
* `junos_get_table` - Retrieve data from device using Junos Tables/Views]

>>> junos_commit

Commit candidate configuration on device.

```
- name: "COMMIT JUNOS CONFIGURATION"
  junipernetworks.junos.junos_commit:
   "host={{ inventory_hostname }}"
   "logfile=changes.log"
   "comment='Commit existing candidate'"
```

>>> junos_rollback

- Rollback configuration of device.
- Parameters:
  - confirm - confirmation in minutes to the commit of the configuration
  - diffs_file - location where diffs are stored

```
- junipernetworks.junos.junos_rollback:
   host: "{{ inventory_hostname }}"
   "logfile=rollback.log"
   "diffs_file=rollback.diff"
   "rollback=1"
   "comment='Rolled back by Ansible'"
   "confirm=5"
```

>>> junos_get_config

Retrieve configuration of device.

```
- name: "GET CONFIG"
  junipernetworks.junos.junos_get_config: "user={{ ansible_user }} passwd={{
ansible_ssh_pass }} host={{ inventory_hostname }} filter="interfaces"
dest=configs/interfaces.conf"
- name: "GET CONFIG"
  junipernetworks.junos.junos_get_config: "user={{ ansible_user }} passwd={{
ansible_ssh_pass }} host={{ inventory_hostname }}
filter="interfaces/protocols" dest=configs/intf_proto.conf"
```

>>> junos_install_config

Modify the configuration of a device running Junos OS.

```
# load merge a change to the Junos OS configuration using NETCONF
- junipernetworks.junos.junos_install_config:
    host: "{{ inventory_hostname }}"
    file: "banner.conf"

# load overwrite a new Junos OS configuration using the CONSOLE port
- junipernetworks.junos.junos_install_config:
    "host={{ inventory_hostname }}"
    "console='--telnet={{TERMSERV}},{{TERMSERV_PORT}}'"
    "file=default_new_switch.conf"
    "overwrite=yes"

# load replace a change to the Junos OS configuration using NETCONF
- junipernetworks.junos.junos_install_config:
    "host={{ inventory_hostname }}"
    "file=snmp.conf"
    "replace=yes"
```

>>> junos_install_os

- Install a Junos OS software package.
- Parameters:
  - reboot - boolean; reboots after the installation completes.
  - reboot_pause - Amount of time in seconds to wait after the reboot is issued. default=10
  - version - Junos OS version string as it would be reported by the show version command
  - package - Absolute path on the local server to the Junos OS software package

```
- junipernetworks.junos.junos_install_os:
    "host={{ inventory_hostname }}"
    "version=12.1X46-D10.2"

"package=/usr/local/junos/images/junos-vsrx-12.1X46-D10.2-domestic.tgz"
```

>>> Summary

- Always test the modules
- Vendors and community are always adding new features
- Understand module idempotency
- Understand if the module supports check mode

Lecture 12: Third Party Modules

Labs 20-21

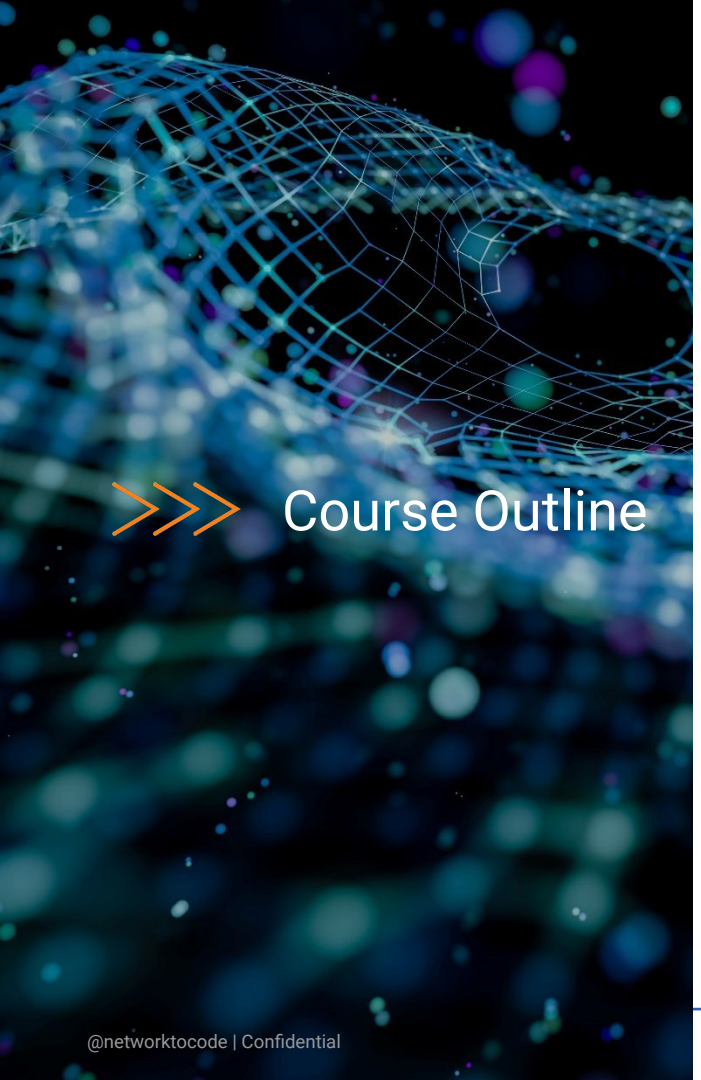
>>> Lab Time

- Lab 20 - Getting Device Facts using NAPALM Library
- Lab 21 - Configuration Management using NAPALM Library

Recap on Material Learned

Course Objectives

- Learn Ansible vocabulary and how to use it
- See Ansible used to accomplish challenges
- Apply Ansible yourself to some challenges
- Look for opportunities to apply within your own organization
- Consider how to apply Ansible to accomplish your challenges



Course Outline

Introduction

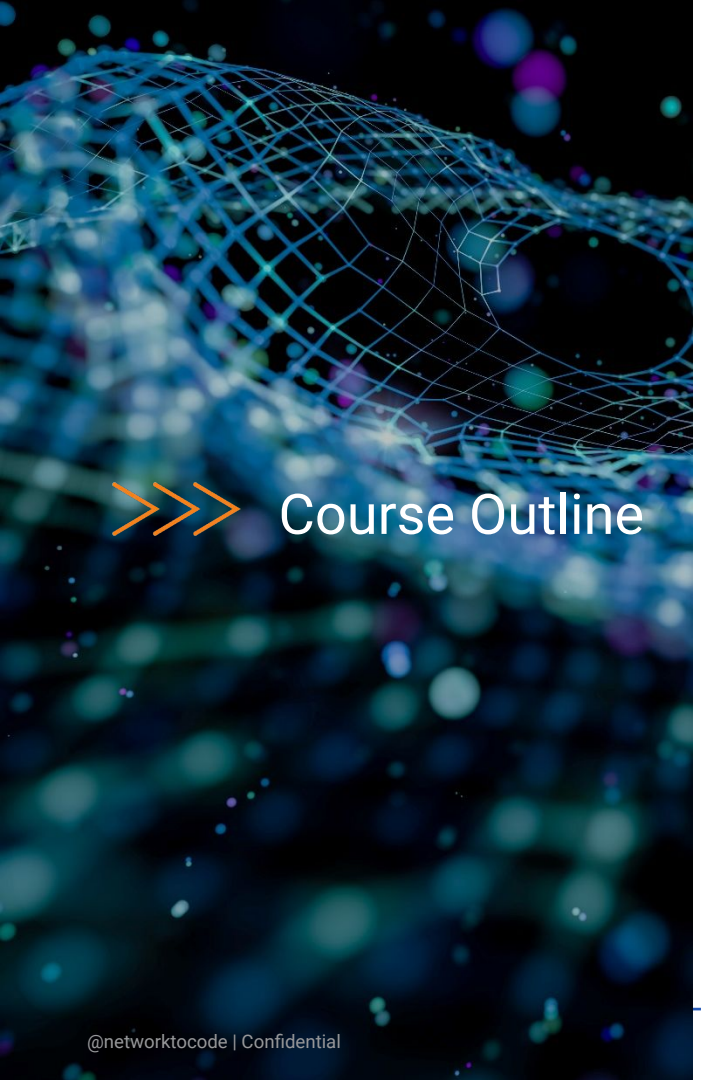
- Course Overview
- What's Possible with Ansible
- Introduction to Ansible Vocabulary

Ansible Variables and Password Security

- Quick Look at YAML & JSON
- Understanding Ansible Variables
- Ansible Vault for Password Security
- Debug Module to Display Ansible Variable Content

Showing Network Device Configurations

- Show Commands on Network Devices
- How do you see the data being gathered?
- Jinja2 Templates
- Loops and Registers
- Parsing Unstructured Data



Diving Deeper into Command and Config Modules

- *_config Modules
- diff_against Parameter
- Declarative Configuration
- Data Collection and Reporting
- Core *_facts Modules

Reusable and Repeatable Code

- Ansible Collections
- Ansible Roles
- Ansible Dynamic Inventories
- Third Party Modules

>>> Next Steps

This course should NOT be a conclusion.

There are more steps to your Network Automation journey:

1. Keep the repository content nearby as a resource to accomplish challenges as each opportunity arises
2. Try to apply this content AT LEAST an hour per week
3. Join the NTC Network Automation Slack channel at https://slack.networktocode.com
4. Partner with others within your organization to cooperate to accomplish challenges


>>> network .toCode()
Ansible - Bonus Topics

>>> Building a Virtual Environment for Experimental Code Versions

In Windows WSL2 with Linux, native Linux, or macOS shell, do the following:

- `apt-get install python-venv` # Install virtual environment code if needed
- `python -m venv venv.lab` # Create virtual environment named venv.lab
- `source venv.lab/bin/activate` # Activate this virtual environment (even after `deactivate`)
- `pip install -U pip` # Update pip to latest version
- `pip install 'ansible==4.*'` # Install Ansible 2.11 Core / 4.x Community
- `pip list` # List what packages are installed
- `ansible --version` # Check Ansible version
- `ansible-galaxy collection list` # List installed Ansible collections
- …
- `deactivate` # Stop virtual environment

>>> Common pip commands

pip is a package installation manager for Python. Common commands (with example packages) include:

- `pip --version` # Current version of pip
- `pip list` # List installed packages
- `pip install -U pip` # Update pip to latest version
- `pip show ansible` # Show information about installed packages (like ansible)
- `pip install napalm` # Install package (like napalm)
- `pip check` # Verify installed packages have compatible dependencies
- `pip uninstall napalm` # Uninstall package (like napalm)

Where is a specific Ansible 2.9 module? Ansible_builtin_runtime.yml

(note that previous shorter commands still work for now)

https://github.com/ansible/ansible/blob/devel/lib/ansible/config/ansible_builtin_runtime.yml

9742 lines (9742 sloc) | 367 KB

```yaml
# Copyright (c) 2020 Ansible Project
# GNU General Public License v3.0+ (see COPYING or https://www.gnu.org/licenses/gpl-3.0.txt)
plugin_routing:
  connection:
    # test entries
    redirected_local:
      redirect: ansible.builtin.local
    buildah:
      redirect: containers.podman.buildah
    podman:
      redirect: containers.podman.podman
    aws_ssm:
      redirect: community.aws.aws_ssm
    chroot:
      redirect: community.general.chroot
    docker:
      redirect: community.docker.docker
    funcd:
      redirect: community.general.funcd
    iocage:
      redirect: community.general.iocage
```