NETWORK AUTOMATION WORKSHOP

Introduction to Ansible for network engineers and operators
Housekeeping

- Timing
- Breaks
- Takeaways
What You Will Learn

Ansible is capable of handling many powerful automation tasks with the flexibility to adapt to many environments and workflows.

- What is Ansible, its common use cases
- How Ansible works and terminology
- Network modules
  - Backup and Restore network devices
  - Self documentating networks
- Using roles
- Ansible Tower
MANAGING NETWORKS HASN’T CHANGED IN 30 YEARS.
According to Gartner

<table>
<thead>
<tr>
<th>Method</th>
<th>Percentage of Respondents</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLI on individual devices</td>
<td>71</td>
</tr>
<tr>
<td>GUI on individual devices</td>
<td>8</td>
</tr>
<tr>
<td>Vendor’s network management system</td>
<td>10</td>
</tr>
<tr>
<td>Network automation tool</td>
<td>6</td>
</tr>
<tr>
<td>API</td>
<td>3</td>
</tr>
<tr>
<td>Others</td>
<td>2</td>
</tr>
</tbody>
</table>

**Figure 1**
Primary Method for Making Network Changes

**Source:** Gartner, *Look Beyond Network Vendors for Network Innovation*. January 2018. Gartner ID: G00349636. (n=64)
Automation considerations

- Compute is no longer the slowest link in the chain
- Businesses demand that networks deliver at the speed of cloud
- Automation of repeatable tasks
- Bridge silos
WHAT IS ANSIBLE AUTOMATION?

Ansible Automation is the enterprise framework for automating across IT operations.

Ansible Engine runs Ansible Playbooks, the automation language that can perfectly describe an IT application infrastructure.

Ansible Tower allows you scale IT automation, manage complex deployments and speed productivity.
WHY ANSIBLE?

SIMPLE

- Human readable automation
- No special coding skills needed
- Tasks executed in order
- Usable by every team
- Get productive quickly

POWERFUL

- App deployment
- Configuration management
- Workflow orchestration
- Network automation
- Orchestrate the app lifecycle

AGENTLESS

- Agentless architecture
- Uses OpenSSH & WinRM
- No agents to exploit or update
- Get started immediately
- More efficient & more secure
ANSIBLE NETWORK AUTOMATION

50
Network Platforms

700+
Network Modules

12*
Galaxy Network Roles

ansible.com/networking
galaxy.ansible.com/ansible-network

*Ansible Network modules comprise 1/3 of all modules that ship with Ansible Engine
Common use cases

- Backup and restore device configurations
- Upgrade network device OS
- Ensure configuration compliance
- Apply patches to address CVE
- Generate dynamic documentation
- Discrete Tasks
  - Ensure VLANs are present/absent
  - Enable/Disable netflow on WAN interfaces
  - Manage firewall access list entries

Basically anything an operator can do manually, Ansible can automate.
How Ansible Works

Module code is executed locally on the control node.

Module code is copied to the managed node, executed, then removed.

NETWORKING DEVICES

LINUX/WINDOWS HOSTS

LOCAL EXECUTION

REMOTE EXECUTION
ANSIBLE AUTOMATION ENGINE

- PUBLIC / PRIVATE CLOUD
- CMDB
- PUBLIC / PRIVATE CLOUD
- USERS
- ANSIBLE PLAYBOOK
- INVENTORY
- MODULES
- CLI
- PLUGINS
- HOSTS
- NETWORK DEVICES
PLAYBOOKS ARE WRITTEN IN YAML

Tasks are executed sequentially

Invoke Ansible modules
MODULES ARE "TOOLS IN THE TOOLKIT"
Python, Powershell, or any language
Extend Ansible simplicity to the entire stack
INVENTORY

[web]
webserver1.example.com
webserver2.example.com

[db]
dbserver1.example.com

[switches]
leaf01.internal.com
leaf02.internal.com

[firewalls]
checkpoint01.internal.com

[lb]
f5-01.internal.com
Understanding Inventory

<table>
<thead>
<tr>
<th>IP Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.1.1.2</td>
</tr>
<tr>
<td>10.1.1.3</td>
</tr>
<tr>
<td>172.16.1.1</td>
</tr>
<tr>
<td>172.16.1.2</td>
</tr>
<tr>
<td>192.168.1.2</td>
</tr>
<tr>
<td>192.168.1.3</td>
</tr>
</tbody>
</table>
Understanding Inventory - Groups

There is always a group called "all" by default

Groups can be nested

```yaml
[atl]
access1.atl.com ansible_host=10.1.1.2
access2.atl.com ansible_host=192.168.1.2

[core]
core1.nw.com
core2.nw.com

[access]
access1.nw.com
access2.nw.com
```

```yaml
[DC:children]
core
access

[east-coast:children]
DC
atl

[atl]
access1.atl.com ansible_host=10.1.1.2
access2.atl.com ansible_host=192.168.1.2

[core]
core1.nw.com
core2.nw.com

[access]
access1.nw.com
access2.nw.com
```
Inventory - variables

Group variables apply for all devices in that group

Host variables apply to the host and override group vars

```
[all:vars]
ansible_username=admin
ansible_password=pa55w0rd
snmp_ro=public123
snmp_rw=private123

[east-coast:vars]
ntp_server=10.99.99.99
anycast=169.1.1.1

[DC:children]
core
access

[east-coast:children]
DC
atl

[atl]
access1.atl.com ansible_host=10.1.1.2 snmp_ro=atl123
access2.atl.com ansible_host=192.168.1.2

[core]
core1.nw.com snmp_ro=corepub123 snmp_rw=corepri123
core2.nw.com
```
A Sample Playbook

- name: DEPLOY VLANS
  hosts: access
  connection: network_cli
  gather_facts: no
  tasks:
    - name: ENSURE VLANS EXIST
      nxos_vlan:
        vlan_id: 100
        admin_state: up
        name: WEB

- Playbook is a list of plays.
- Each play is a list of tasks.
- Tasks invoke modules.
- A playbook can contain more than one play.
Exercise 1.0
Exploring the lab environment

In this lab you will explore the lab environment and build familiarity with the lab inventory.

Approximate time: 10 mins
Playbook definition for network automation

- Target play execution using `hosts`
- Define the connection: `network_cli`
- About `gather_facts`
Running a playbook

```yaml
---
- name: GATHER INFORMATION FROM ROUTERS
  hosts: cisco
  connection: network_cli
  gather_facts: no

  tasks:
    - name: GATHER ROUTER FACTS
      ios_facts:

[student1@control-node networking-workshop]$ ansible-playbook gather_ios_data.yml
PLAY [GATHER INFORMATION FROM ROUTERS] *****************************************************
TASK [GATHER ROUTER FACTS] **********************************************************************
ok: [rtr1]
ok: [rtr4]
ok: [rtr3]
ok: [rtr2]
PLAY RECAP ************************************************************************************
rtr1    : ok=1    changed=0    unreachable=0    failed=0
rtr2    : ok=1    changed=0    unreachable=0    failed=0
rtr3    : ok=1    changed=0    unreachable=0    failed=0
rtr4    : ok=1    changed=0    unreachable=0    failed=0

[student1@ip-172-16-101-121 networking-workshop]$
Displaying output

Use the optional **verbose** flag during playbook execution

```
$ ansible-playbook gather_ios_data.yml -v
Using /home/student1/.ansible.cfg as config file

PLAY [GATHER INFORMATION FROM ROUTERS] *****************************************************

TASK [GATHER ROUTER FACTS] *****************************************************
ok: [rtr3] => {
    "ansible_facts": {
        "ansible_net_all_ipv4_addresses": ["10.100.100.3", "192.168.3.103", "172.16.235.46", "192.168.35.101", "10.3.3.103"],
        "ansible_all_ipv6_addresses": [],
        "ansible_net_filesystems": [{"bootflash":[]},
        "ansible_net_gather_subset": [{"hardware": "default", "interfaces": []},
        "ansible_net_hostname": "rtr3",
        "ansible_net_image": "boot:packages.conf",
        "ansible_net_interfaces": [{"GigabitEthernet1": {"bandwidth": 1000000, "description": "null", "duplex": "Full", "ipv4": [{"address": "172.16.235.46", "subnet": "16"}]},
        "lineprotocol": "up", "macaddress": "0e93.7710.e63c", "mediatype": "Virtual",
        "mtu": 1500, "operstatus": "up", "type": "CSR vNIC"},
        "Loopback0": {"bandwidth": 8000000, "description": "null", "duplex": "null", "ipv4": [{"address": "10.100.100.3", "subnet": "24"}]},
        "mtu": 1514, "operstatus": "up", "type": "null"}],
        "Loopback1": {"bandwidth": 8000000, "description": "null", "duplex": "null", "ipv4": [{"address": "10.100.100.3", "subnet": "24"}],
        "lineprotocol": "up", "macaddress": "10.3.3.103/24", "mediatype": "null", "mtu": 1514, "operstatus": "up", "type": "Tunnel10", "bandwidth": 100, "description": "null", "duplex": "null", "ipv4": [{"address": "10.100.100.3", "subnet": "24"}]
        ...
        ...
    }
```

<output truncated for readability>

Increase the level of verbosity by adding more "v's" -vvvv
Limiting Playbook execution

Playbook execution can be limited to a subset of devices using the --limit flag.

$ ansible-playbook gather_ios_data.yml  -v --limit rtr1

Forget a flag / option ?
Just type ansible-playbook then press enter
A note about variables

Other than the user defined variables, Ansible supports many inbuilt variables. For example:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ansible_*</td>
<td>Output of fact gathering</td>
</tr>
<tr>
<td>inventory_hostname</td>
<td><strong>magic</strong> inbuilt variable that is the name of the host as defined in inventory</td>
</tr>
</tbody>
</table>
| hostvars            | **magic** inbuilt variable dictionary variable whose key is inventory_hostname  
|                     | e.g. hostvars[webserver1].my_variable                                        |
Displaying output - The “debug” module

The debug module is used like a "print" statement in most programming languages. Variables are accessed using "{{{ }}") - quoted curly braces
Exercise 1.1
Writing your first playbook

In this lab you will write your first playbook and run it to gather facts from routers. You will also practice the use of "verbose" and "limit" flags in addition to working with variables within a playbook.

Approximate time: 10 mins
Modules

Modules do the actual work in Ansible, they are what gets executed in each playbook task.

- Typically written in Python (but not limited to it)
- Modules can be idempotent
- Modules take user input in the form of parameters
Network modules

Ansible modules for network automation typically references the vendor OS followed by the module name.

- `_facts`
- `_command`
- `_config`

More modules depending on platform

- Arista EOS = eos_*
- Cisco IOS/IOS-XE = ios_*
- Cisco NX-OS = nxos_*
- Cisco IOS-XR = iosxr_*
- F5 BIG-IP = bigip_*
- F5 BIG-IQ = bigiq_*
- Juniper Junos = junos_*
- VyOS = vyos_*
Modules per network platform

```yaml
tasks:
  - name: configure eos system properties
    eos_system:
      domain_name: ansible.com
      vrf: management
      when: ansible_network_os == 'eos'
  - name: configure nxos system properties
    nxos_system:
      domain_name: ansible.com
      vrf: management
      when: ansible_network_os == 'nxos'
```
Modules Documentation

https://docs.ansible.com/
# List out all modules installed
$ ansible-doc -l

...  
ios_banner  Manage multiline banners on Cisco IOS devices
ios_command Run commands on remote devices running Cisco IOS
ios_config  Manage Cisco IOS configuration sections
...

# Read documentation for installed module
$ ansible-doc ios_command
> IOS_COMMAND

Sends arbitrary commands to an ios node and returns the results read from the
device. This module includes an argument that will cause the module to wait for a
specific condition before returning or timing out if the condition is not met. This
module does not support running commands in configuration mode. Please use
[ios_config] to configure IOS devices.

Options (= is mandatory):
...
Limiting tasks within a play

- Tags allow the user to selectively execute tasks within a play.
- Multiple tags can be associated with a given task.
- Tags can also be applied to entire plays or roles.

```
- name: DISPLAY THE COMMAND OUTPUT
debug:
  var: show_output
tags: show
```

Tags are invoked using the --tags flag while running the playbook

```
[userr@ansible]$ ansible-playbook gather_ios_data.yml --tags=show
```

This is useful while working with large playbooks, when you might want to "jump" to a specific task.
Limiting tasks within a play - or skip them!

- --skip-tags allows you to skip everything

```yaml
- name: DISPLAY THE COMMAND OUTPUT
debug:
  var: show_output
tags: show
```

[user@ansible]$ ansible-playbook gather_ios_data.yml --skip-tags=show
Registering the output

The `register` parameter is used to collect the output of a task execution. The output of the task is 'registered' in a variable which can then be used for subsequent tasks.

```yaml
- name: COLLECT OUTPUT OF SHOW COMMANDS
  ios_command:
    commands:
      - show run | i hostname
      - show ip interface brief
  tags: show
  register: show_output
```
Exercise 1.2
Module documentation, Registering output & tags

In this lab you will learn how to use module documentation. You will also learn how to selectively run tasks using tags and learn how to collect task output into user defined variables within the playbook.

Approximate time: 15 mins
Vendor specific config modules allow the user to update the configuration on network devices. Different ways to invoke the `_config` module:

```
tasks:
  - name: ENSURE THAT THE DESIRED SNMP STRINGS ARE PRESENT
    ios_config:
      commands:
        - snmp-server community ansible-public RO
        - snmp-server community ansible-private RW
        - snmp-server community ansible-test RO

tasks:
  - name: ENSURE THAT ROUTERS ARE SECURE
    ios_config:
      src: secure_router.cfg
```
Validating changes before they are applied

Ansible lets you validate the impact of the proposed configuration using the --check flag. Used together with the --verbose flag, it lets you see the actual change being pushed to the device:

```
[student1@control-node networking-workshop]$ ansible-playbook router_configs.yml --check -v
Using /home/student1/.ansible.cfg as config file

PLAY [UPDATE THE SNMP RO/RW STRINGS] ************************************************************
TASK [ENSURE THAT THE DESIRED SNMP STRINGS ARE PRESENT] ***************************************
changed: [rtr3] => {
  "banners": {},
  "changed": true,
  "commands": ["snmp-server community ansible-test RO"],
  "updates": ["snmp-server community ansible-test RO"]
}
```
Exercise 2.0
Updating the router configurations

In this lab you will learn how to make configuration changes using Ansible. The exercise will demonstrate the idempotency of the module. Additionally you will learn how to validate a change before actually applying it to the devices.

Approximate time: 20 mins
Scenario: Day 2 Ops - Backing up and restoring router configuration
Back up router configuration

The backup parameter of the `ios_config` module triggers the backup and automatically stores device configuration backups within a `backups` directory.

```yaml
- name: BACKUP ROUTER CONFIGURATIONS
  hosts: cisco
  connection: network_cli
  gather_facts: no

  tasks:
    - name: BACKUP THE CONFIG
      ios_config:
        backup: yes
        register: config_output
```
Cleaning up the backed up configuration

The backed up configuration has 2 lines that should be removed:

```
Building configuration...
Current configuration with default configurations exposed : 393416 bytes
```

The `lineinfile` module is a general purpose module that is used for manipulating file contents.
Cleaning up (cont’d)

Cleaning up an exact line match:

```yaml
- name: REMOVE NON CONFIG LINES
  lineinfile:
    path: "./backup/{{inventory_hostname}}.config"
    line: "Building configuration..."
    state: absent
```
Cleaning up (cont’d)

Matching using a regular expression:

```yaml
- name: REMOVE NON CONFIG LINES - REGEXP
  lineinfile:
    path: "./backup/{{inventory_hostname}}.config"
    regexp: 'Current configuration.*'
    state: absent
```
Restoring the configuration

If any out of band changes were made to the device and it needs to be restored to the last known good configuration, we could take the following approach:

- Copy over the cleaned up configuration to the devices
- Use vendor provided commands to restore the device configuration

*In our example we use the Cisco IOS command `config replace`. This allows for applying only the differences between running and the copied configuration*
Restoring (cont’d)

```yaml
- name: RESTORE CONFIGURATION
  hosts: cisco
  connection: network_cli
  gather_facts: no

  tasks:
  - name: COPY RUNNING CONFIG TO ROUTER
    command: scp ./backup/{{inventory_hostname}}.config {{inventory_hostname}}:/{{inventory_hostname}}.config

  - name: CONFIG REPLACE
    ios_command:
      commands:
        - config replace flash:{{inventory_hostname}}.config force
```

Note the use of `inventory_hostname` to effect host specific changes
Exercise 2.1 & 2.2
Backup & Restore router configuration

In this lab you will implement a typical Day 2 Ops scenario of backing up and restoring device configurations.

Approximate time: 20 mins
Scenario: Creating living/dynamic documentation
Templates

- Ansible has native integration with the Jinja2 templating engine
- Render data models into device configurations
- Render device output into dynamic documentation

Jinja2 enables the user to manipulate variables, apply conditional logic and extend programmability for network automation.
Using templates to generate configuration

Data model:

vlans:
  - id: 10
    name: WEB
  - id: 20
    name: APP
  - id: 30
    name: DB

Jinja2 template

{% for vlan in vlans %}
  vlan {{ vlan.id }}
    name {{ vlan.name }}
{% endfor %}

Tying it all together

- name: RENDER THE VLAN CONFIGURATION
  template:
    src: vlans.j2
    dest: "vlan_configs/{{ inventory_hostname }}.conf"

leaf1.conf

vlan 10
  name WEB
vlan 20
  name APP
vlan 30
  name DB
Using templates to build dynamic documentation

- Generate documentation that never goes stale
- Build troubleshooting reports
- Same data to generate exec reports and engineering reports using different templates

```perl
{{ inventory_hostname.upper() }}
---
{{ ansible_net_serialnum }} : {{ ansible_net_version }}

<table>
<thead>
<tr>
<th>RTR</th>
<th>9YJXS2VD3Q7 : 16.08.01a</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTR2</td>
<td>9QHUCH0VZ19 : 16.08.01a</td>
</tr>
<tr>
<td>RTR3</td>
<td>92GJ5B1DL14 : 16.08.01a</td>
</tr>
<tr>
<td>RTR4</td>
<td>9TCP27U9TQG : 16.08.01a</td>
</tr>
</tbody>
</table>
```
Assembling the data

The **assemble** module is used to generate a consolidated file by combining fragments. This is a common strategy used to put snippets together into a final document.

```
- name: CONSOLIDATE THE IOS DATA
  assemble:
    src: reports/
    dest: network_os_report.md
    delegate_to: localhost
    run_once: yes
```

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RTR1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>___</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9YJXS2VD3Q7 : 16.08.01a</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RTR2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>___</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9QHUCH0VZI9 : 16.08.01a</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RTR3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>___</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9ZGJ5B1DL14 : 16.08.01a</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RTR4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>___</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9TCM27U9TQG : 16.08.01a</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Exercise 3.0
An introduction to templating

In this lab you will use a basic Jinja2 template to generate a markdown report that contains the device name, serial number and operating system version. You will create a report per device and then use the assemble module to consolidate them.

Approximate time: 15 mins
A quick introduction to roles

The 2 basic files required to get started with Ansible are:

- Inventory
- Playbook
Roles
Roles are Playbooks

- Roles help simplify playbooks.
- Think of them as callable functions for repeated tasks.
- Roles can be distributed/shared; similar to libraries.

Example Playbook
```
# site.yml
---
- hosts: DC
  roles:
    - ntp
    - vlan
```

Directory Structure
```
site.yml
roles/
  ntp/
    tasks/
      main.yml
  vlan/
    tasks/
      main.yml
```
# site.yml

---

- hosts: routers
  roles:
    - ntp
    - vlan

ntp/
  tasks/
  main.yml

vlan/
  tasks/
  main.yml

- name: CONFIGURE NTP
  ios_config:
    lines: ntp server 1.2.3.4

- name: CONFIGURE VLAN
  ios_vlan:
    vlan_id: 100
Ansible Galaxy

http://galaxy.ansible.com

- Ansible Galaxy is a hub for finding, reusing and sharing Ansible roles.

- Jump-start your automation project with content contributed and reviewed by the Ansible community.
Using parsers to generate custom reports

On most network devices, show command output is "pretty" formatted but not structured. The Ansible `network-engine` role provides support for 2 text parsing engines:

- TextFSM
- Command Parser

```yaml
- name: GENERATE INTERFACE REPORT
  hosts: cisco
  gather_facts: no
  connection: network_cli

  roles:
  - ansible-network.network-engine

  tasks:
  - name: CAPTURE SHOW INTERFACES
    ios_command:
      commands:
        - show interfaces
      register: output

  - name: PARSE THE RAW OUTPUT
    command_parser:
      file: "parsers/show_interfaces.yml"
      content: "{{ output.stdout[0] }}"
```
Structured data from show commands

rtr2#show interfaces
GigabitEthernet1 is up, line protocol is up
Hardware is CSR vNIC, address is 0e56.1bf5.5ee2 (bia 0e56.1bf5.5ee2)
Internet address is 172.17.16.140/16
MTU 1500 bytes, BW 1000000 Kbit/sec, DLY 10 usec,
   reliability 255/255, txload 1/255, rxload 1/255
Encapsulation ARPA, loopback not set
Keepalive set (10 sec)
Full Duplex, 1000Mbps, link type is auto, media type is Virtual
output flow-control is unsupported, input flow-control is unsupported
ARP type: ARPA, ARP Timeout 04:00:00

<output omitted for brevity>
Exercise 3.1
Building dynamic documentation using the command parser

The objective of this lab is to generate a dynamic documentation from the output of a device `show` command.

Approximate time: 20 mins
AUTOMATION ACROSS THE ENTERPRISE
WHAT IS ANSIBLE TOWER?

Ansible Tower is a UI and RESTful API allowing you to scale IT automation, manage complex deployments and speed productivity.

• Role-based access control

• Deploy entire applications with push-button deployment access

• All automations are centrally logged

• Powerful workflows match your IT processes
RBAC
Allow restricting playbook access to authorized users. One team can use playbooks in check mode (read-only) while others have full administrative abilities.

PUSH BUTTON
An intuitive user interface experience makes it easy for novice users to execute playbooks you allow them access to.

RESTful API
With an API first mentality every feature and function of Tower can be API driven. Allow seamless integration with other tools like ServiceNow and Infoblox.

WORKFLOWS
Ansible Tower's multi-playbook workflows chain any number of playbooks, regardless of whether they use different inventories, run as different users, run at once or utilize different credentials.

ENTERPRISE INTEGRATIONS

CENTRALIZED LOGGING
All automation activity is securely logged. Who ran it, how they customized it, what it did, where it happened - all securely stored and viewable later, or exported through Ansible Tower's API.
Extending Ansible to the Enterprise

Individual

Teams

Enterprise

Windows Team

Network Team

Windows Team

Virtual project or automation Team

Network Team

Network device

Windows

Network device

Windows

Network device

Windows

GitHub

ANSIBLE

Engine

GitHub
Next Steps

Thanks so much for joining the class. Here are some next steps on how to get more information and join the community!
Bookmark the GitHub Project

https://www.github.com/network-automation

- Examples, samples and demos
- Run network topologies right on your laptop
Chat with us
Engage with the community

- Slack
  https://ansiblenetwork.slack.com

- IRC
  #ansible-network on freenode
  http://webchat.freenode.net/?channels=ansible-network
Next Steps

- It's easy to get started
  https://ansible.com/get-started

- Do it again
  https://github.com/network-automation/linklight
  https://network-automation.github.io/linklight/

- Instructor Led Classes
  Class DO457: Ansible for Network Automation
  https://red.ht/2MiAqvA
NEXT STEPS

GET STARTED
ansible.com/get-started
ansible.com/tower-trial

JOIN THE COMMUNITY
ansible.com/community

WORKSHOPS & TRAINING
ansible.com/workshops
Red Hat Training

SHARE YOUR STORY
Follow us @Ansible
Friend us on Facebook