Configuration Automation

Asghar Ghori
Humber College Institute of Technology & Advanced Learning
Toronto, Ontario, Canada ©2024
www.humber.ca www.endtech.ca

Roadmap for Part 2

- **Configuration Management with Ansible**
  - Module 10: Introduction to Ansible
  - Module 11: Prepare for Ansible'ing
  - Module 12: Invoke Ad-hoc Commands
  - Module 13: Understand, Create, and Use Playbooks
  - Module 14: Parametrize Input Values
  - Module 15: Manipulate Task Output and Ansible Facts
  - Module 16: Employ Conditionals and Loops
  - Module 17: Implement Handlers
  - Module 18: Manage Files and Folders
  - Module 19: Understand, Create, and Use Roles

Module 10

Introduction to Ansible

What is Ansible?

- Open source automation platform
  - Provides a simple language to describe infrastructure or configuration
  - Provides an automation engine
- Code is easy to read, understand, and troubleshoot
- Agentless, masterless, idempotent, procedural
- Version-controlled
- Cross-platform support
- Secure (uses **SSH** and **WinRM**)
- Employs Python

Use Cases

- (1) Post-provisioning activities
- (2) Scheduled configuration management
- (3) One-time or repeatable management tasks

- Tasks include:
  - Patching, software installation, upgrades, or removal, user creation and management, storage management, time sync, DNS, firewall, SELinux, file copy, file changes, owning user/group, system hardening, etc.

Basic Terminology

- **Control node**: Linux/Windows machine to run Ansible engine; requires Python and inventory of managed nodes
- **Managed node**: a machine running Linux, UNIX, or Windows
- **Module**: Python program (command) that runs on a managed node (3000+ modules available)
- **Task**: a single operation performed with a module (command)
- **Play**: includes one or more tasks
- **Playbook**: file that includes plays; repeatability and reusability
- **Inventory**: resolvable hostnames or IP addresses

Tour of Ansible Documentation Site

Ansible Documentation Site Tour

Module 11

Prepare for Ansible'ing

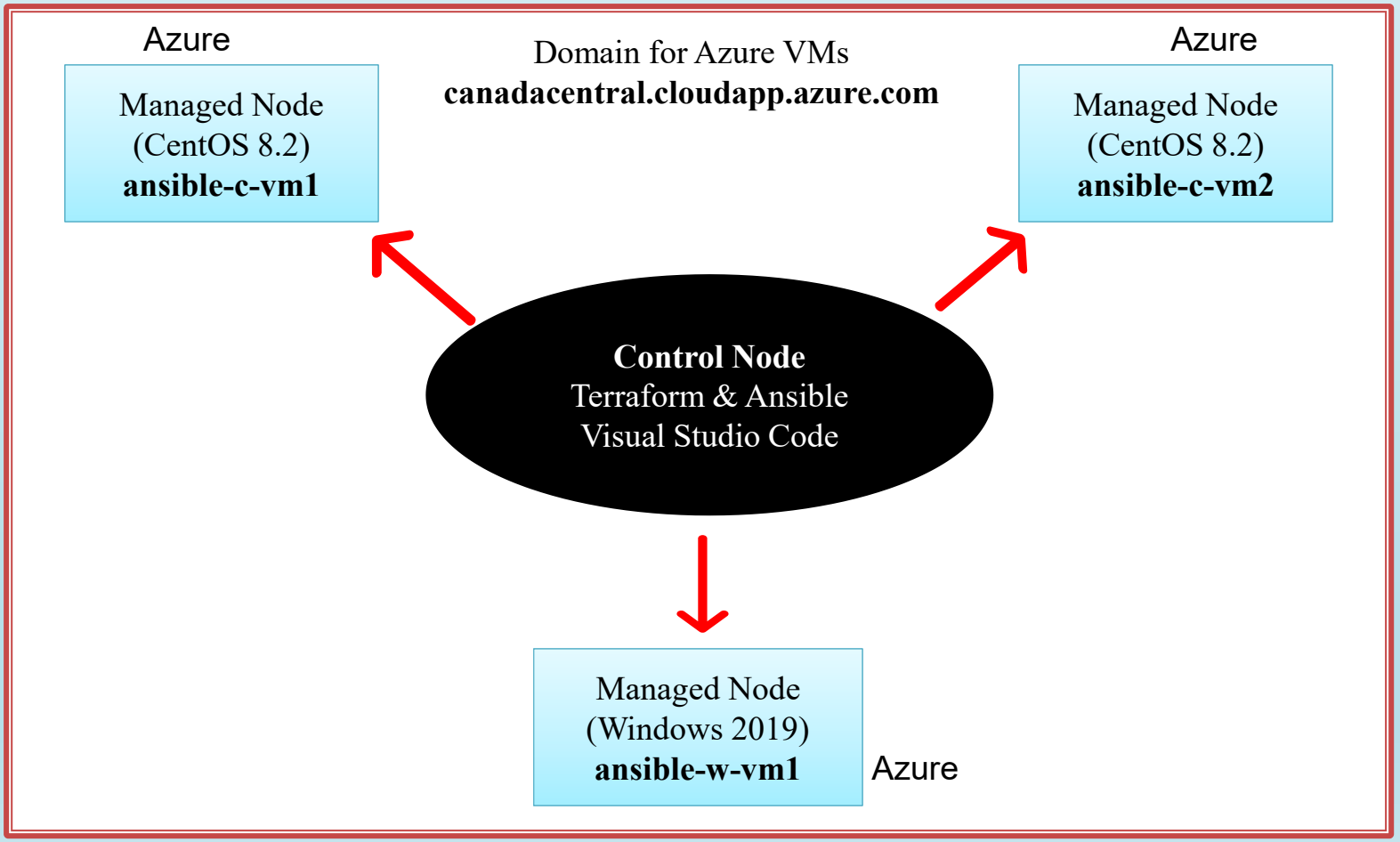
LAB Environment

- Use your **local computer** as the Ansible control node
- Use Linux and Windows VMs as managed nodes
- Build 3 nodes (VMs) in Azure using Terraform:
  - humberID-c-vm1 (Standard_B1s) CentOS 8.2
  - humberID-c-vm2 (Standard_B1s) CentOS 8.2
  - humberID-w-vm1 (Standard_B1s) Windows 2019

- Your user account on the **control node** must be able to **ssh** into Linux managed nodes without being prompted for password
- The **winadm** user in Terraform configuration must be able to **WinRM** into Windows managed nodes

Note: We focus more on Linux management via Ansible

LAB Architecture



Lab07s1: Build Ansible Lab Environment with Terraform

- See Lab07 Section 1

Common Commands

- **ansible-config:** shows Ansible configuration
- **ansible-inventory**: shows Ansible inventory
- **ansible**: runs a single (ad-hoc) task
- **ansible-playbook**: runs a playbook
- **ansible-doc**: provides help on modules

- Run **ansible-doc -l** to list all modules
- Run **ansible-doc user** to view help for the user module
- Run **ansible-doc win_user** to view help for the Windows user module

Common Modules

- Ansible modules are split into several categories:

  Ansible Module Categories

  - **Files**: acl, copy, fetch, file, find, stat, synchronize, etc.
  - **Packaging**: dnf, apt, pip, zypper, etc.
  - **System**: SELinux, parted, LVM, filesystem, cron, at, systemd, etc.
  - **Windows**: Windows management modules
  - **Database**: MySQL, ProgreSQL, etc.

- We will employ:
  - **Linux** modules from Files, Packaging, and System categories
  - **Windows** modules from Windows category

Control Ansible Behaviour

- Default global configuration file: **/etc/ansible/ansible.cfg**
- Several sections:
  - Two main sections are **defaults** and **privilege_escalation**

- Priority:
  - Global defaults (/etc/ansible/ansible.cfg) (priority 4)
  - Home directory (~/.ansible.cfg) (priority 3)
  - Current directory (./ansible.cfg) (priority 2)
  - Environment variable ANSIBLE_CONFIG (priority 1)
- Ansible uses the first config file it finds and ignores the rest

- True | true | yes | 1 (all the same)
- False | false | no | 0 (all the same)

Ansible Configuration

The ansible.cfg file

```
[defaults]
inventory       = ~/automation/ansible/hosts
roles_path    = ~/automation/ansible/roles
forks           = 5
host_key_checking = False
deprecation_warnings=False

[privilege_escalation]
become=True
become_method=sudo
become_user=root
become_ask_pass=False
```

Number of managed nodes to run against in parallel

Guards against server spoofing and man-in-the-middle attacks

To view settings:

$ **ansible-config dump**

$ **ansible-config view**

For Linux

Lab07s2: Set Configuration Defaults

- See Lab07 Section 2

Understand Host Inventory

- Text file containing a list of managed nodes
- Default inventory file: **/etc/ansible/hosts**
- Common practice is to create your **own** inventory file and store it in the Ansible root directory
- Example: **~/automation/ansible/hosts**
- "localhost" is always part of inventory whether defined or not
- Nodes can be defined in separate sections:
  - ungrouped, group, range, or nested/child

Define Inventory

- **Ungrouped** (individual hostnames/IPs)

  192.168.0.23

  192.168.0.24

  192.168.0.25

  ansible-c-vm1

- **Group** (collection of hostnames/IPs)

  [webservers]

  ansible-c-vm1

  192.168.0.30

  ansible-w-vm1

Define Inventory contd.

- **Range**

  192.168.[0:1].[23:25]

  ansible-w-vm[1:10].canadacentral.cloudapp.azure.com

- **Nested/Child** (includes other groups or ungrouped)

  [allservers:children]

  appservers (group)

  dbservers (group)

  webservers (group)

  ansible-c-vm2 (ungrouped)

```
[linux]
ansible-c-vm1
ansible-c-vm2

[windows]
ansible-w-vm1

[os:children]
linux
windows
```

Define Inventory (Windows)

```
[linux]
ansible-c-vm1
ansible-c-vm2

[windows]
ansible-w-vm1

[os:children]
linux
windows
```

```
[windows]
ansible-w-vm1

[windows:vars]
ansible_user=winadm
ansible_password="Winadm!23"
ansible_connection=winrm
ansible_port=5985
ansible_winrm_transport=ntlm
ansible_winrm_server_cert_validation=ignore
```

Windows inventory

Privilege escalation

Inventory Configuration contd.

$ ansible --list-hosts all
$ ansible --list-hosts ansible-c-vm2
$ ansible-inventory --graph

$ ansible --list-hosts ungrouped
$ ansible --list-hosts windows

Lab07s3: Set Up and View Inventory

- See Lab07 Section 3

Module 12

Invoke Ad-hoc Commands

What is an Ad-hoc Command?

- An ad-hoc command performs a **one-time** execution of a **single** task (module)
- Same as running a **single** Linux or Windows command
- Invoked at the command prompt on the control node
- Inventory path may be specified to run on multiple nodes

Common Modules and Host Specification

Linux Modules: user | group | setup | copy | file | dnf | parted | lvg

Windows Modules: win_user | win_group | setup | win_copy | win_file | win_package

Host specification: localhost | all | ansible-c-vm1 | <group>

Connection Protocols: SSH and WinRM

- Ansible uses **ssh** to connect to Linux machines
  - Enabled by default on all Linux distributions and versions
  - No additional configuration required on new machines
  - Ensure port **22** is open in the Terraform NSG for Linux VMs

- Ansible uses **WinRM** to connect to Windows machines
  - Enabled by default on Windows Server 2016 and later
  - Ensure that your Terraform Windows virtual machine resource includes the **winrm_listener** sub-block with protocol "**Http**"
  - Ensure that port **5985** is open in the NSG
  - "**dnf install pip**" and "**pip install pywinrm**" on the control node
  - Enable ICMP in Windows Defender (firewall on the managed node)

Ad-hoc Command Syntax

- Syntax:

$ **ansible <host> -m <module> -a <arguments>**

- Examples:

$ **ansible ansible-c-vm2 -m setup**

$ **ansible -m user -a 'name=user10 uid=2000 state=present' localhost**

$ **ansible all -m copy -a 'content="Managed by Ansible\n" dest=/etc/motd'**

$ **ansible linux -a 'cat /etc/motd'**

$ **ansible localhost -m file -a 'dest=/tmp/newdir state=absent'**

$ **ansible linux -m dnf -a 'name=dcraw state=present'** (run once to install; run again to verify)

Three Similar Modules: command, shell, and raw

- All three can perform the same tasks, however:
  - The **command** module is considered more secure; however, it does not support certain shell features such as piping and redirection
  - Use the **shell** module instead
  - The **raw** module behaves the way the shell module does; however, it does not require the presence of Python software on managed nodes

$ ansible ansible-c-vm1 -m command -a date

$ ansible ansible-c-vm1 -m shell -a 'ls > ls.out'

$ ansible ansible-c-vm1 -m raw -a 'env | grep -i shell'

Lab07s4: Run Ad-hoc Commands

- See Lab07 Section 4

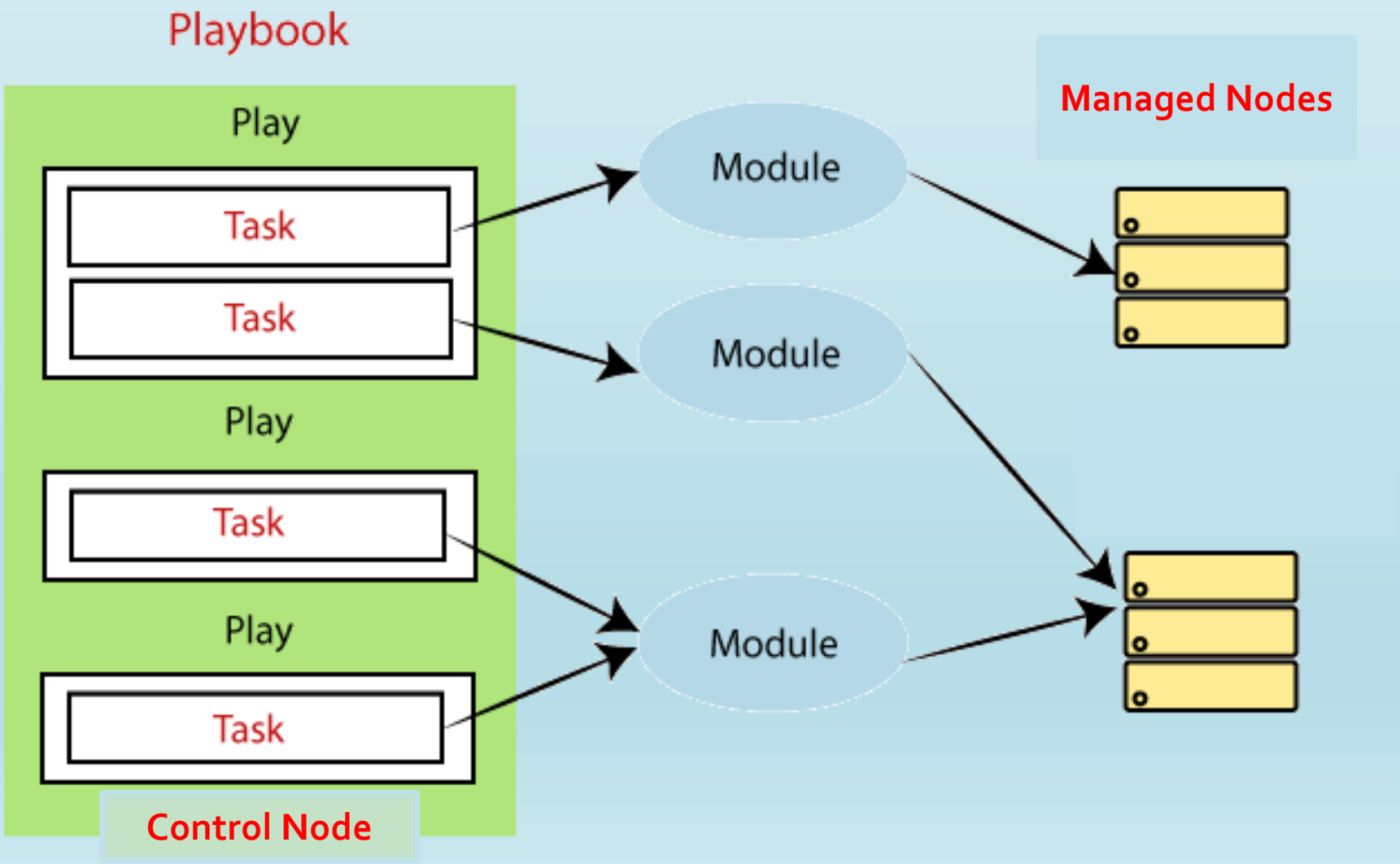
Module 13

Understand, Create, and Use Playbooks

What is a Playbook?

- A file containing changes to be implemented
- Includes one or more plays each with one or more tasks
- Comments may be added for documentation purposes
- No compilation required
- Concept is similar to that of a shell script (automation of lengthy and multiple tasks)
- Runs against one or more managed nodes
- Repeatable, shareable, and predictable
- **Host inventory** and **privilege escalation** can also be defined inside a playbook (for all plays or for an individual play)

Playbook Execution and Interaction



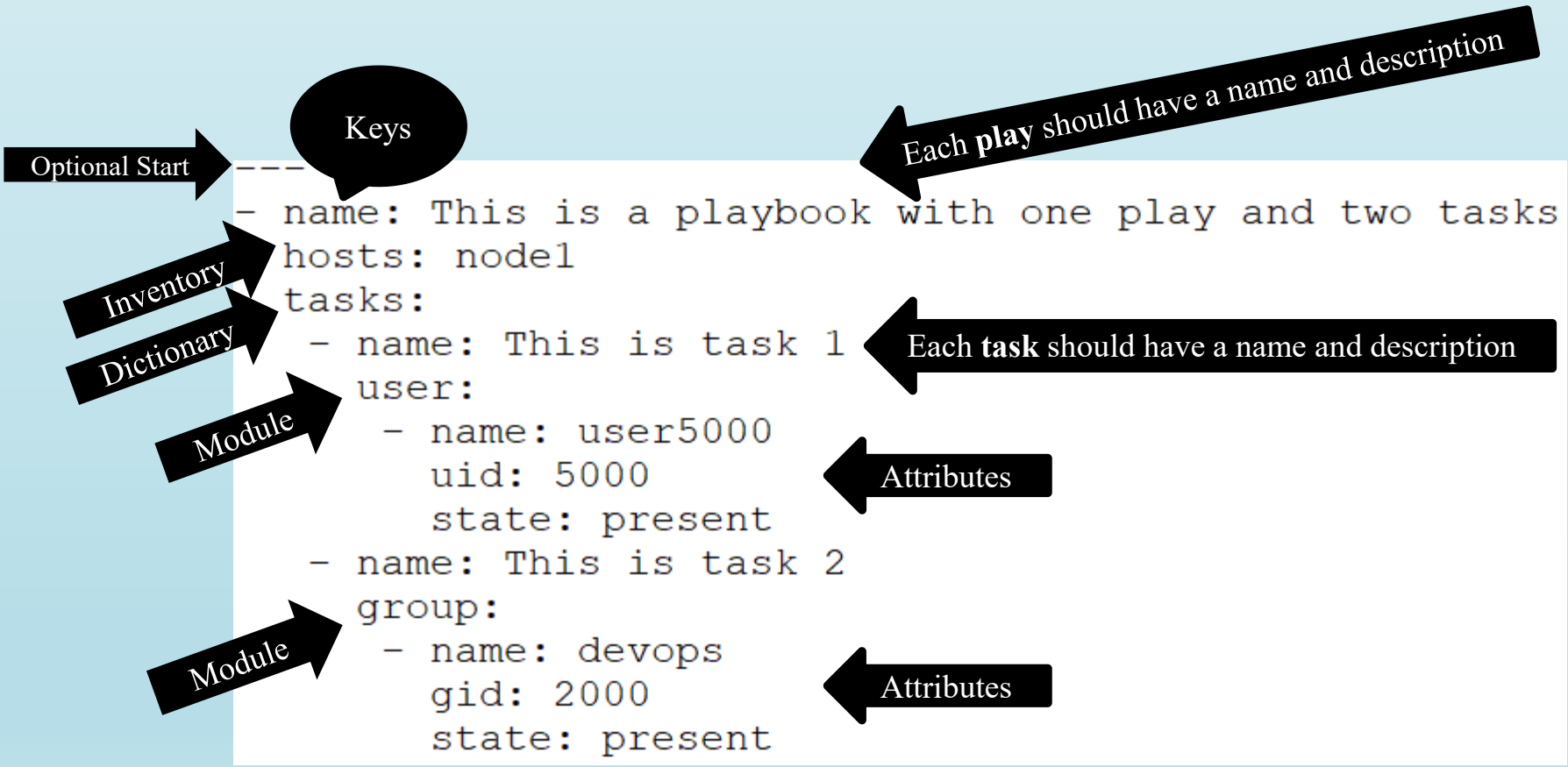
Execution Order

- Each play within a playbook runs in order
- Each task within a play runs in order
- A task must run successfully against all hosts before the next task is initiated

Playbook Format

- Written in YAML format (**Y**et **A**nother **M**arkup **L**anguage)
- **Indentation** is important:
  - Use spaces; tabs are not allowed
  - Python is extremely sensitive to indentation
- Optional start (---) and end (…)
- Key/value pairs separated with a colon (:) and a space
  - Name: user1
  - Phone: “(123) 456-7890”

Analysis of a Playbook



You can also add comments with the #

Analysis of a Playbook contd.

```
tasks:
 - name: Install the software
   dnf:
     name:
      - apache
      - nmap
     state: latest
```

Playbook Execution and Verbosity

$ **ansible-playbook -C playbook.yml**	(dry run)
$ **ansible-playbook playbook.yml --syntax-check**	(syntax check)
$ **ansible-playbook --list-tasks playbook.yml**	(show tasks that will be run)
$ **ansible-playbook --list-hosts playbook.yml**	(show which tasks will run on which managed node)
$ **ansible-playbook playbook.yml**	(run playbook)

- Add -v, -vv, -vvv, or -vvvv with the **ansible-playbook** command for increased verbosity

Lab08s1: Create Playbooks

- See Lab08 Section 1

Module 14

Parametrize Input Values

Overview

- Input values can be passed via **variables** to allow flexibility
- Same concept as in shell scripting and other languages
- **Examples**: list of users, packages, files, firewall rules
- Input variables can be defined
  - (1) Directly inside a playbook using **vars**
  - (2) In an external file and called via **vars_files** from a playbook
  - (3) In inventory file as:
    - host variables (called via **host_vars**)
    - group variables (called via **group_vars**)
- Precedence: (1) command line (-e), (2) play, (3) playbook, (4) inventory file (host_vars), (5) inventory file (group_vars), 17 more levels

Playbook Variables

Defined directly inside a playbook (not recommend)

```
- name: This play will create groups
  hosts: ansible-c-vm2
  vars:
   userlist: newuser
   grouplist: group1
  tasks:
    - name: Create groups
      group:
       name: "{{ grouplist }}"
       state: present
    - name: Create user
      user:
       name: "{{ userlist }}"
       groups: "{{ grouplist }}"
       append: 1
       uid: 2400
       home: /home/"{{ userlist }}"
```

Defined in an external file (recommended)

```
userlist: newuser
grouplist: group1
```

```
- name: This play will create group
  hosts: ansible-c-vm2
  vars_files:
   - vars/userinfo
  tasks:
    - name: Create groups
      group:
       name: "{{ grouplist }}"
       state: present
    - name: Create user
      user:
       name: "{{ userlist }}"
       groups: "{{ grouplist }}"
       append: 1
       uid: 2400
       home: /home/"{{ userlist }}"
```

Lab08s2: Parametrize Playbooks

- See Lab08 Section 2

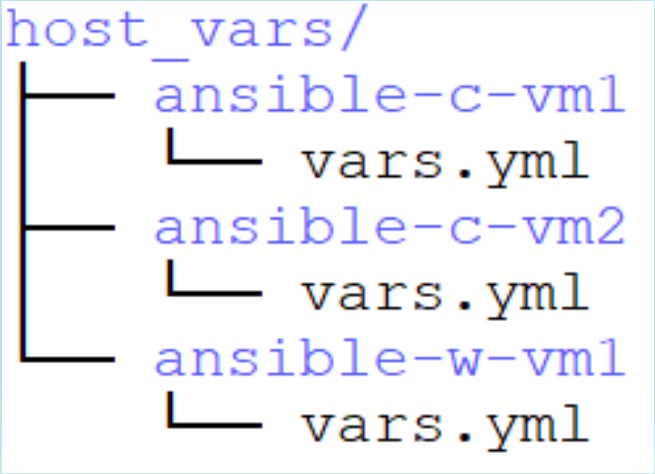
Host Variables

- Defined against a **specific host** in inventory file
- Best practices:
  - Create separate directory called **host_vars**
  - Create subdirectory matching hostname under **host_vars**
  - Create variable file underneath (any name)

- Example:

cat host_vars/ansible-c-vm2/vars.yml

packages: httpd



Host Variables Example

```
hosts: ansible-c-vm2
tasks:
 - name: Install software
   yum:
    name: "{{ packages }}"
    state: latest

 - name: Start Apache and mark it for auto-start on system reb
   systemd:
    name: "{{ packages }}"
    state: started
    enabled: true
```

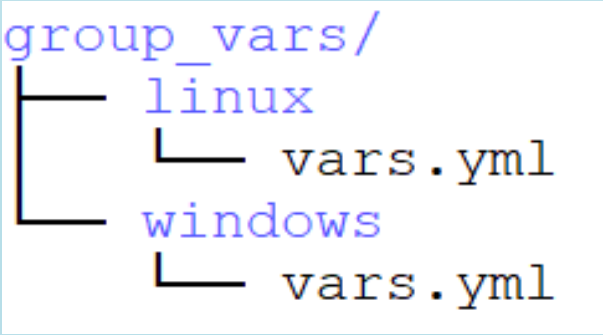
Group Variables

- Defined against a **specific group** in inventory file
- Best practices:
  - Create separate directory called **group_vars**
  - Create subdirectory matching inventory group under **group_vars**
  - Create variable file underneath (any name)

- Examples:

cat group_vars/linux/vars.yml

packages: httpd



Group Variables Example

```
- name: This play will install Apache software
  hosts: linux
  tasks:
   - name: Install software
     apt:
      name: "{{ packages }}"
      state: latest

   - name: Start Apache and mark it for auto-start on system reboots
     systemd:
      name: "{{ packages }}"
      state: started
      enabled: true
```

Lab08s3: Host and Group Vars

- See Lab08 Section 3

Module 15

Manipulate Task Output and Ansible Facts

Work with Task Output

- Output can be captured into a temp variable using **register**
- Defined at the **task** level
- Verbosity level can be added
- Benefits:
  - (1) Can be displayed in its **entirety**
  - (2) Can have values of **certain keys** displayed
  - (3) Can be employed in **other plays** within the playbook
  - (4) Provides **insight** into what is happening **during** task execution
  - (5) Can be used for **debugging**

- **Note: command**, **shell**, and **raw** modules do not display output during execution

Capture Task Output

- How to capture task output into a variable

```
tasks:
 - name: Create user
   user:
    name: user4500
    state: present
   register: creation_result
```

Show Entire Output

- Show the entire captured output

```
tasks:
 - name: Create user
   user:
    name: user4500
    state: present
   register: creation_result

 - name: Display entire task results
   debug:
     var: creation_result
```

Show Specific Output

- Show a particular value from the captured output

```
tasks:
 - name: Create user
   user:
    name: user4500
    state: present
   register: creation_result

 - name: Display entire task results
   debug:
     var: creation_result

 - name: Display a particular item from task results
   debug:
     msg: The user created is called {{ creation_result['name'] }}
```

msg is used to print a message

Lab09s1: Capture and Display Output

- See Lab09 Section 1

Work with Ansible Facts

- By default, Ansible discovers managed host info using the **setup** module and captures it into a special variable called **ansible_facts**
- Discovered information includes:
  - hostname, kernel version, network interfaces, IP addresses, OS version, environment variables, # of CPUs, free memory, disk usage data, etc.
- May be used in plays, conditionals, loops, etc.
- Use cases:
  - Apply patches to Linux nodes running a matching distribution
  - Update time zone on managed nodes with non-matching hostnames
- Can be turned off:
  - At the play level: **gather_facts: no**
  - Globally: **ansible.cfg** file

Gather and Show Facts

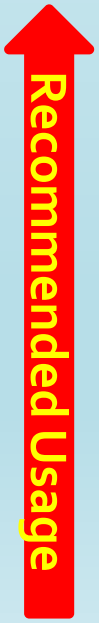
```
- name: This play will gather and display facts
  hosts: ansible-c-vm1
  gather_facts: true
  tasks:
   - name: Display node facts
     debug:
       var: ansible_facts
```

Run the above playbook against **localhost** and examine the output

Common Facts

Common facts and how to reference them

short hostname	ansible_facts['hostname']	ansible_facts.hostname
FQDN	ansible_facts['fqdn']	ansible_facts.fqdn
Kernel version	ansible_facts['kernel']	ansible_facts.kernel
DNS servers	ansible_facts['dns']['nameservers']	ansible_facts.dns.nameservers
IPv4 address	ansible_facts['default_ipv4']['address']	ansible_facts.default_ipv4.address



Inject Specific Facts

```
tasks:
 - name: Display IPv4 addresses
   debug:
     msg: >
       The IP address of {{ ansible_facts['fqdn'] }} is
       {{ ansible_facts['default_ipv4']['address'] }}
```

- > represents line continuation

Manipulate Package Facts

- **package_facts** module gathers info about installed packages
- Stores the output in a special variable called **packages**

```
tasks:
 - name: Gather package information
   package_facts:
    manager: auto

 - name: List installed packages
   debug:
    var: packages

 - name: Display kernel package version
   debug:
     msg:
       The kernel package version is {{ packages['kernel'][0].version }}
```

Lab09s2: Work with Facts

- See Lab09 Section 2

Module 16

Employ Conditionals and Loops

Run Tasks Conditionally

- Tasks can be run based on a test condition
- Conditions can be based on (1) playbook variable values, (2) **register** variables, or (3) Ansible facts
- Variable values can be string, numeric, or boolean
- Sample use cases:
  - Install software if /var has 1GB of free space available
  - Use dnf for RHEL/CentOS; use apt-get for Ubuntu/Debian
  - Check minimum memory and compare against free memory
  - Use **register** output to decide whether to continue if a program fails
  - Tune a web server based on the number of CPUs obtained from **facts**
- Implemented via the **when** keyword

Common Conditions and Syntax

Common Conditionals	
Operation	**Example**
Equal to (string)	ansible_distribution == "Ubuntu"
Equal to (numeric)	max_memory == 2048
Less than	min_memory < 1024
Less than or equal to	min_memory <= 1024
Greater than	min_memory > 1024
Greater than or equal to	min_memory >= 1024
Not equal to	min_memory != 1024
Check if a variable exists	min_memory is defined
Check if a variable does not exist	min_memory is not defined
Boolean (true, 1, yes)	memory_available
Boolean (false, 0, no)	not memory_available

Use Variable as Condition

```
---
- name: This play will install software if defined
  hosts: ansible-c-vm1
  vars:
   packages: nmap

  tasks:
   - name: Install "{{ packages }}" software
     yum:
      name: "{{ packages }}"
      state: latest
     when: packages is defined
```

Use Register as Condition

```
tasks:
 - name: Capture user information
   command: cat /etc/passwd
   register: users

 - name: Show which servers have this user
   debug:
     msg: "This server has the proxy user"
   when: users.stdout.find('proxy') != -1
```

Use Single Fact as Condition

```
tasks:
 - blockinfile:
    path: /etc/systemd/resolved.conf
    insertafter: EOF
    backup: yes
    state: present
    block: |
      Domains=canadacentral.cloudapp.azure.com
   when: ansible_distribution == "Ubuntu"
```

Use Multiple Facts as Condition

```
when: ansible_distribution_version == "8.2" and ansible_kernel
== "4.18.0-193.el8.x86_64"
```

```
when:
  - ansible_distribution_version == "8.2"
  - ansible_kernel == "4.18.0-193.el8.x86_64"
```

```
when: >
  ( ansible_distribution == "Ubuntu" and
    ansible_distribution_major_version == "18" )
  or
  (  ansible_distribution == "Debian" and
     ansible_distribution_major_version == "9" )
```

Example: Use Multiple Facts as Condition

```
tasks:
 - blockinfile:
    path: /etc/resolv.conf
    insertafter: EOF
    backup: yes
    state: present
    block: |
      domain canadacentral.cloudapp.azure.com
   when: ansible_distribution == "RedHat" or ansible_distribution == "CentOS"
```

Lab09s3: Use Conditionals

- See Lab09 Section 3

Task Iteration with Loops

- Execution of the same task more than once
- Loops iterate over a (1) simple list, (2) list of hashes (set of values) (a.k.a. dictionaries), (3) host inventory, and (4) nested lists
- Loops can operate based on conditionals
- Benefits:
  - Eliminate code repetition in plays
  - Less writing
  - Low chances of errors (syntax, typos, etc.)
- Use cases: creating multiple users, initializing multiple disks, creating multiple file systems, etc.
- Implemented via the **loop** keyword

Iteration Over a Simple List

Without Loop

```
tasks:
  - name: Application user
    user:
      name: appuser
      state: present

  - name: Oracle user
    user:
      name: oradba
      state: present
```

1

```
tasks:
  - name: Application and DB users
    user:
      name: "{{ item }}"
      state: present
    loop:
      - appuser
      - oradba
```

2

With Loop

```
vars:
  userlist:
    - appuser
    - oradba
tasks:
  - name: Application and DB users
    user:
      name: "{{ item }}"
      state: present
    loop: "{{ userlist }}"
```

3

An item represents an individual value

Iteration Over a List of Hashes

```
tasks:
  - name: Create appgrp group
    group:
      gid: 4000
      name: appgrp

  - name: Add required user accounts
    user:
      name: "{{ item.username }}"
      uid: "{{ item.userid }}"
      groups: appgrp
      home: "{{ item.homedir }}"
      expires: "{{ item.expiry }}"
      state: present

    loop:
      - username: appuser1
        userid: 4010
        homedir: /home/appuser1
        expiry: 19500
      - username: appuser2
        userid: 4012
        homedir: /home/appuser2
        expiry: 19000
```

Sets of values

Execute Loops Conditionally

- You can run a loop based on a condition
- E.g.: **ansible_mounts** is a list of hashes representing facts about mounted file systems (generated with the **setup** module)
- Two conditions: mounted file system and free space

```
tasks:
  - name: Install mariadb-server
    yum:
      name: mariadb-server
      state: latest

    loop: "{{ ansible_mounts }}"
    when: item.mount == "/" and item.size_available > 400000
```

Lab09s4: Use Loops

- See Lab09 Section 4

Module 17

Implement Handlers

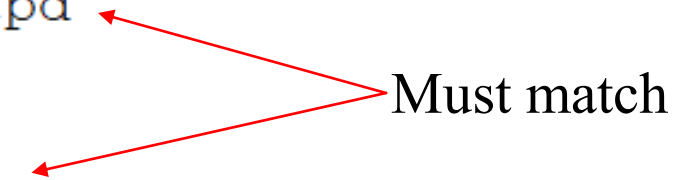
Overview

- Handlers:
  - Are **dormant** tasks
  - Run only if they are **notified** by another **changed** task(s)
  - Can start or restart a service, reboot server, etc.
  - Defined at the **task level**, but must have unique names within a play
  - Run **only once** regardless of how many times they are called upon
  - Run in the **order** in which they are listed in a play
  - Normally run **after** all other tasks in a play have completed
- Default behavior:
  - **Notified** when a task reports **changed** result
  - **Not notified** when a task reports **ok** or **failed** result

Employ Single Handler

```
tasks:
 - name: Restart Apache web service if present
   yum:
     name: httpd
     state: present
   notify: restart_httpd

handlers:
 - name: restart_httpd
   systemd:
     name: httpd
     state: restarted
```



Employ Multiple Handlers

```
tasks:
 - name: Restart MariaDB and Apache web services
   yum:
     name: mariadb-server,httpd
     state: present
   notify:
     - restart_mariadb
     - restart_httpd

handlers:
 - name: restart_mariadb
   systemd:
     name: mariadb
     state: restarted

 - name: restart_httpd
   systemd:
     name: httpd
     state: restarted
```

Lab09s5: Employ Handlers

- See Lab09 Section 5

Module 18

Manage Files and Folders

Overview

- **Set of modules** to perform file and folder management tasks including creating, copying, editing, removing, changing permissions, modifying owning user and group, etc.
- Common modules:
  - **blockinfile**: Inserts, updates, or removes a block of multi-line text
  - **lineinfile**: Replaces a line or confirms whether a line is in
  - **copy**: Uploads a file or folder to the specified location
  - **fetch**: Downloads a file or folder to the control node (opposite of copy)
  - **file**: Creates or removes files or folders, and modifies their attributes (including SELinux settings)
  - **stat**: Pulls the attributes of a file or folder (Linux **stat** command)

Insert Block of Text and Line of Text in File

```
- name: Adding a limit to a file
  blockinfile:
    path: /etc/security/limits.conf
    insertafter: EOF
    backup: yes
    state: present
    block: |
      * hard core 0
```

Note: You can add a marker for **custom** comments with either module.

```
- name: Adding minimum user password validity setting
  lineinfile:
    path: /etc/login.defs
    regex: "PASS_MIN_DAYS"
    line: "PASS_MIN_DAYS 7"
    state: present
```

Upload and Download Files

```
- name: Copying /etc/motd file
  copy:
    src: motd
    dest: /etc/
    owner: root
    group: root
    mode: 0600
```

Note: Add **force: no** to avoid destination overridden

```
- name: Fetching /etc/motd file
  fetch:
    src: /etc/motd
    dest: /tmp/
```

Create and Remove Files and Directories

```
- name: Create a file
  file:
    path: /tmp/newfile
    state: touch
    mode: 0644
    owner: root
    group: root

- name: Create a directory
  file:
    path: /tmp/newdirectory
    state: directory
    mode: 0755
    owner: root
    group: root
```

```
- name: Remove a file
  file:
    path: /tmp/newfile
    state: absent

- name: Remove a directory
  file:
    path: /tmp/newdirectory
    state: absent
```

Retrieve File Status

```
- name: Check the status of /etc/group file
  stat:
    path: /etc/group
  register: stat_output

- name: Show the entire output
  debug:
    var: stat_output
```

Lab10s1: Manage Files/Folders

- See Lab10 Section 1

Module 19

Understand, Create, and Use Roles

Overview

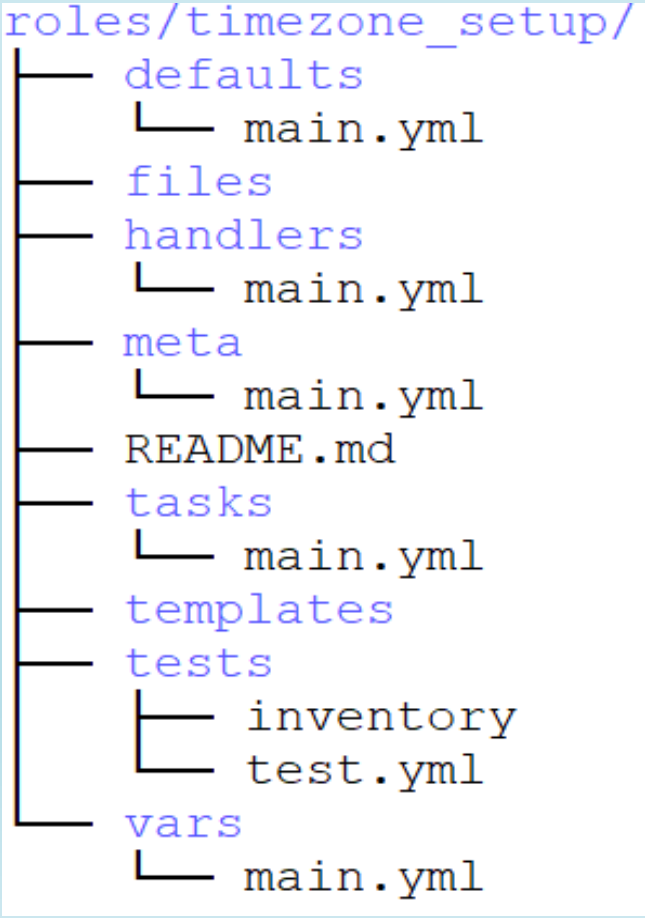
- Set of **similar tasks** performed together
- **Reusable** across projects and among teams
- **Standardized** directory structure and file names to segregate tasks, variables, files, templates, handlers, etc.
- Works as a **single** unit
- Called from a playbook

Examine Role Structure

- Top-level directory is the role name (e.g.: timezone_setup)
- Subdirectory names correspond to their purpose

- **defaults**: default variable values, but they hold the **lowest** priority
- **files:** files to be deployed
- **handlers:** handler configuration
- **meta:** metadata information including inter-role dependencies, author, version, etc.
- **tasks:** the **actual** plays and tasks
- **templates:** (jinja2 templates for **dynamic** content)
- **tests**: inventory and sample **test** code
- **vars:** variables to be passed, hold **higher** precedence than host and group variables

Note: Remove unneeded directories



Call Roles from Playbook

- (Optional) **pre_tasks/post_tasks** can be run
- Roles run in the **order** in which they are listed

```
---
- name: This is the main playbook
  hosts: all
  name: Post VM Deployment Activities
1 pre_tasks:
    - debug:
        msg: "Beginning Post VM Deployment

3 roles:
        - role1
        - role2
  4
6 post_tasks:
    - debug:
        msg: "Finished Post VM Deployment
```

Ansible executes a playbook in the following order:

1. **pre_tasks** run first
2. handlers triggered by **pre_tasks** (not shown)
3. Roles listed under **roles**
4. Any tasks listed under **tasks**
5. Any handlers triggered by **roles** or **tasks** (not shown)
6. **post_tasks**
7. handlers triggered by **post_tasks** run at the end (not shown)

Roles Directory Structure

- Update the **roles_path** variable in ansible.cfg

```
[auto@automation ansible]$ pwd
/home/auto/automation/ansible
[auto@automation ansible]$ mkdir roles
[auto@automation ansible]$ cd roles/
[auto@automation roles]$ ansible-galaxy init timezone_setup
- Role timezone_setup was created successfully
[auto@automation roles]$ ll
total 0
drwxrwxr-x. 10 auto auto 154 Jan 21 15:28 timezone_setup
[auto@automation roles]$
[auto@automation roles]$ tree timezone_setup/
timezone_setup/
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml
```

Lab10s2: Get Ready for Using Ansible Roles

- See Lab10 Section 2

Best Practices

- Create and maintain **README.md** and meta/main.yml files
- **Do not store secrets** in playbooks or in role directory structure; use variables files, environment variables, etc.
- Define one role for **one type or similar types** of tasks
- A role can **depend** on another role, but avoid for simplicity
- **Version control** roles; git repo for sharing/versioning

Lab10s3: Create Package and Patch Management Roles

- See Lab10 Section 3

Lab10s4: Create Profile Management Role

- See Lab10 Section 4

Lab10s5: Create Role for Time Zone Management

- See Lab10 Section 5

Lab10s6: Create Role for Syslog Service Management

- See Lab10 Section 6

Thank You