Thinknyx®
You Trust, We Deliver!

ANSIBLE

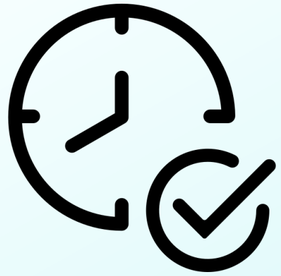
Automation with Ansible – Hands-on

Software Deployment

Configuration Management

System Updates

Automate Repetitive Tasks



Saves Time

Reduce Humar Error

Yogesh Raheja

ANSIBLE

CERTIFIED

Puppet for the Absolute Beginners – Hands-On

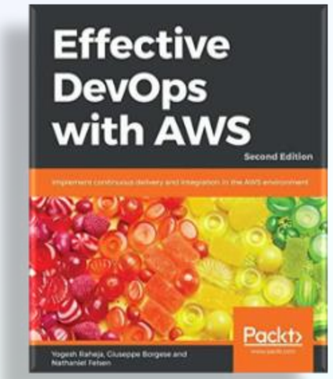
The Ultimate Linux Bootcamp for DevOps SRE & Cloud Engineers

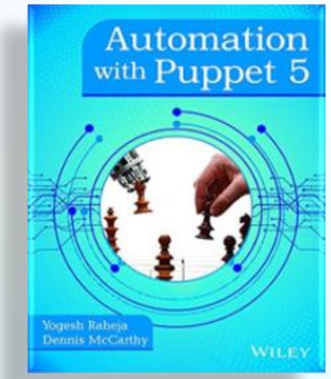
Practical Kubernetes – Beyond CKA and CKAD

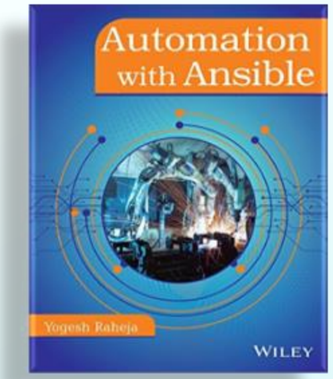
Argo CD for the Absolute Beginners - Hands-On

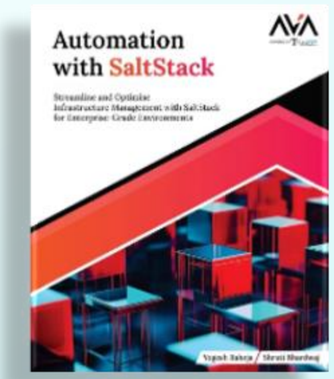
Kubernetes and Cloud Native Associate

Mastering Docker Essentials Hands-on



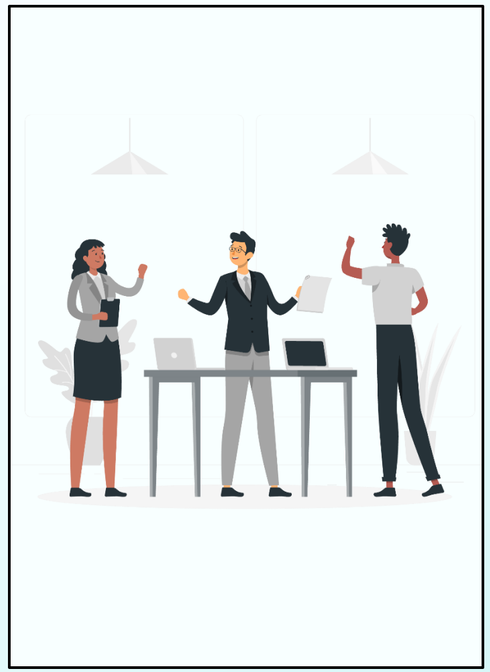






Yogesh Raheja

Madhuri Jha



Thinknyx Team

How Will This Course Work?

- Ansible Fundamentals
- Key Terminologies
- Ansible Environment Set Up
- Automating Tasks & Managing Configurations
- Ansible Vault
- Web-Based Interfaces with AWX

Course Overview

Lectures

Demonstrations

Practice Tests

Playbooks

Ansible Galaxy

Roles

Collections

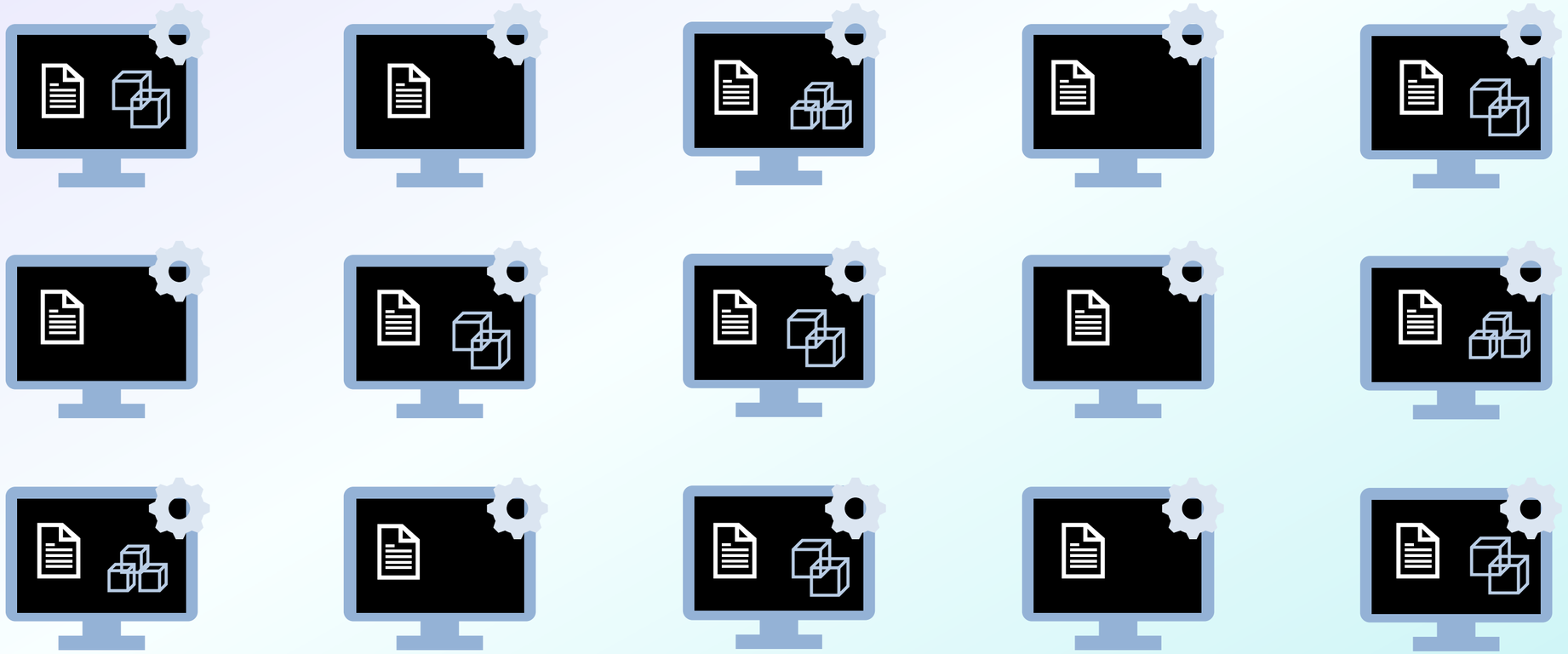
Introduction to Ansible

- What is Ansible?
- History
- Why Ansible?
- Features & Capabilities
- Ansible Use Cases

1.1

Introduction

On-premises or Cloud



Introduction

Introduction

Open Source

- ✓ YAML

Michael DeHaan
February 2012

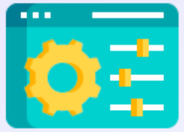
Red Hat
2015

Operating Systems

- ✓ Red Hat
- ✓ CentOS
- ✓ Ubuntu
- ✓ Oracle
- ✓ Linux
- ✓ macOS
- ✓ Solaris
- ✓ Windows

1.2

Why Ansible?



Configuration Management

Application Deployment

Salt

Puppet

Chef

Terraform

Provisioning

Orchestration

Automation

Key Features

Agentless Architecture

Declarative Language

Versatility

Community & Documentation

Scalability

Integration & Extensibility

Idempotency

Open Source

Security

Simplicity

1.3

Use Cases

Configuration Management

Cloud Infra Automation

Network Automation

Application Deployment

CI/CD

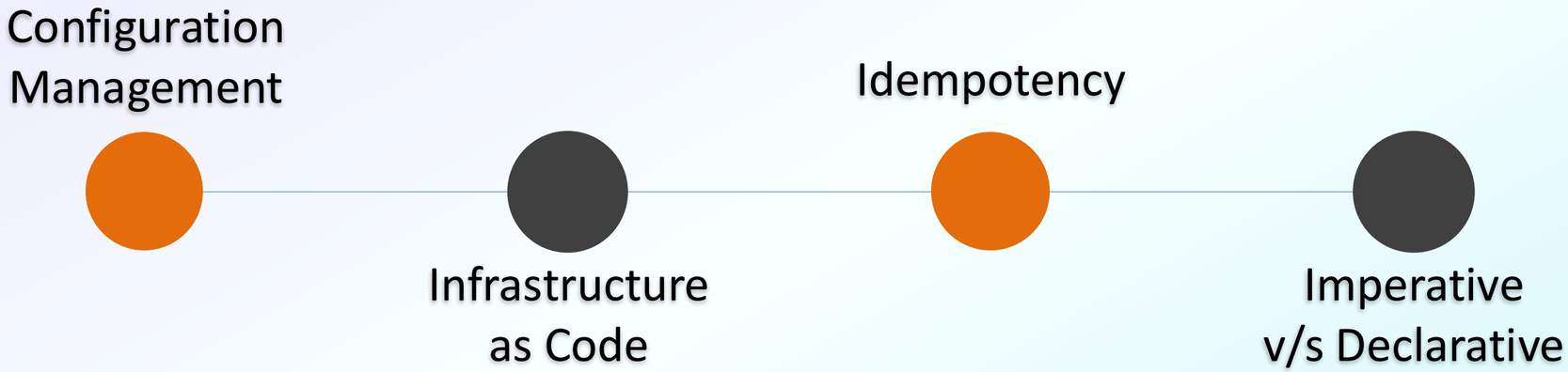
Database Administration

Security & Compliance

User & Access Management

1.4

Prerequisite Concepts



Configuration Management


Red Hat


Ubuntu


Ubuntu


Fedora


Suse

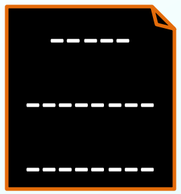

Ubuntu


Red Hat


Red Hat


Red Hat

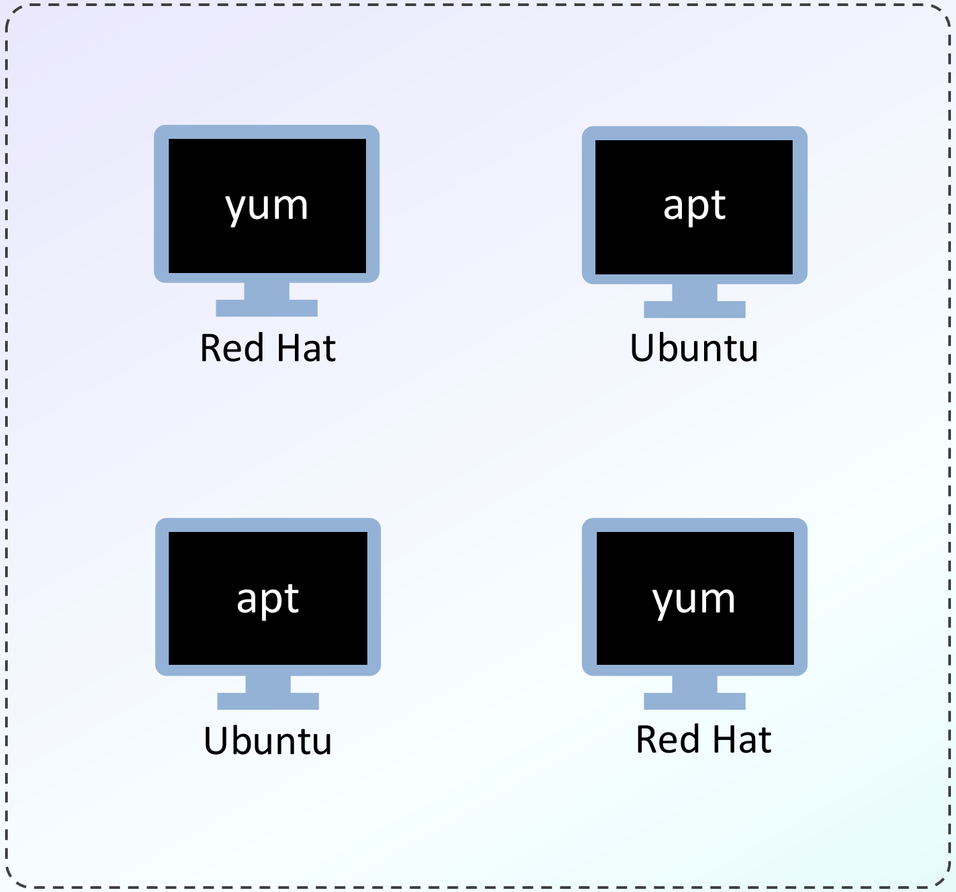

Ubuntu

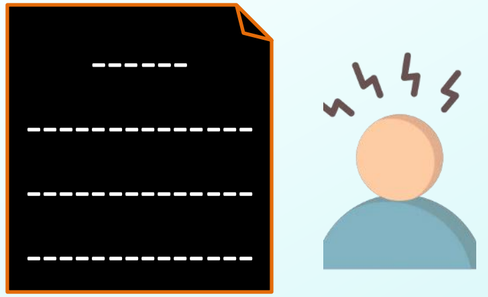

shell script

× **Complex scripts**

× **Hard to maintain**

Configuration Management



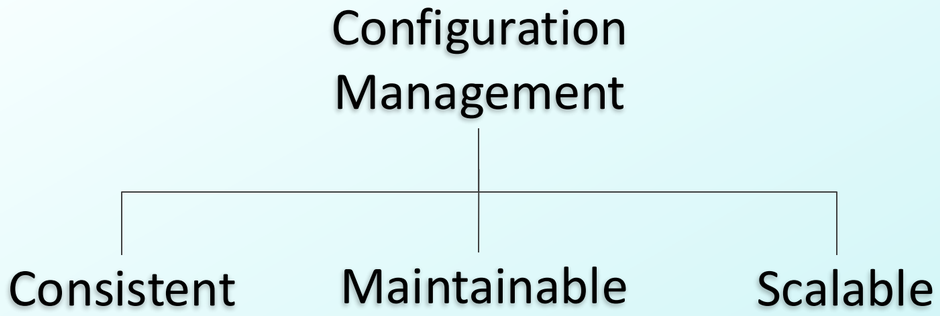


Configuration Management

- ✓ Configuration management is the automated process of organizing and controlling changes to a system's configurations throughout their lifecycle

apache.yml

```
---
- name: Install and configure Apache web server
  hosts: webservers
  become: yes
  tasks:
    - name: Ensure Apache is installed
      ansible.builtin.apt:
        name: apache2
        state: present
        update_cache: yes
    - name: Start and enable Apache service
      ansible.builtin.service:
        name: apache2
        state: started
        enabled: yes
```



Infrastructure as Code (IaC)

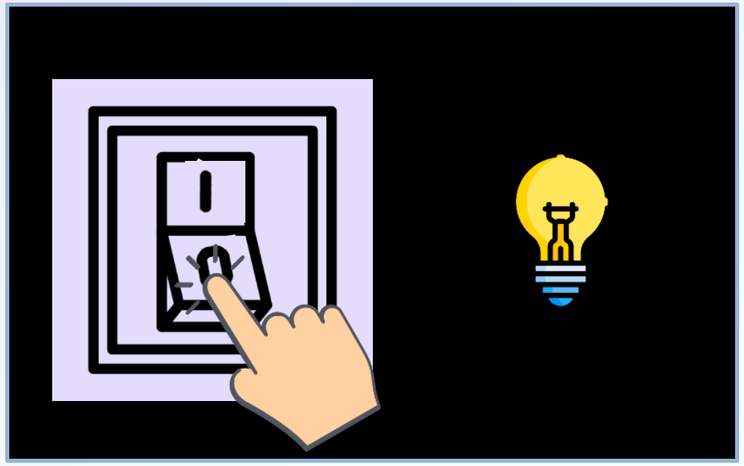
- ✓ Infrastructure as Code (IaC) is a method of configuring and managing IT infrastructure using code written in a human-readable format

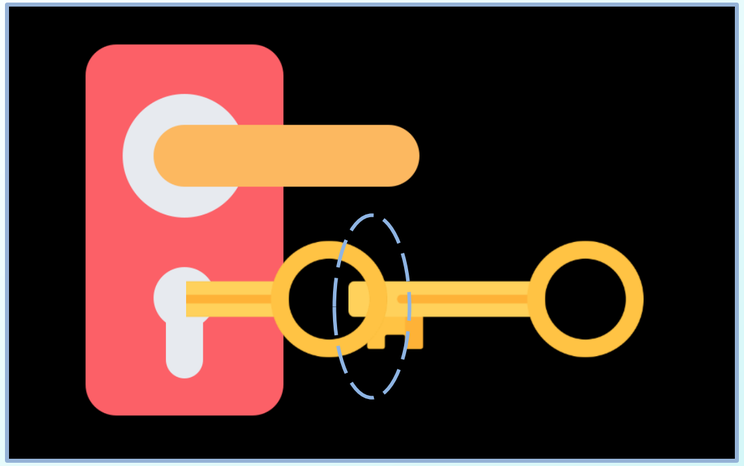
s3.yml

```
---
- name: Sample IaC
  hosts: localhost
  tasks:
  - name: Create a simple S3 bucket
    amazon.aws.aws_s3:
      name: s3bucketusingansible
      aws_access_key: XXXXXXXXXX
      aws_secret_key: XXXXXXXXXX
      state: present
      region: us-east-1
```

Idempotency

- ✓ Idempotency refers to the ability to execute an operation that produces the same result whether it is performed once or multiple times





- ✓ Executing Ansible code multiple times will consistently produce the same result

Imperative vs Declarative

Imperative

- ✓ Providing detailed instructions for every step
- ✓ Cooking from Scratch

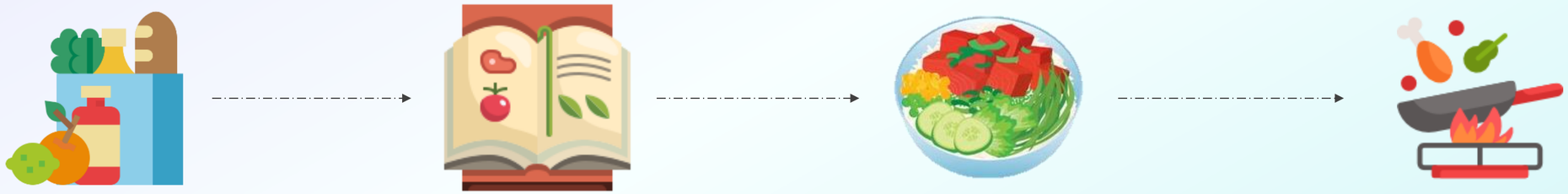
Declarative

- ✓ Stating the desired outcome and letting the system determine how to achieve it
- ✓ Ready-to-eat meal

Imperative vs Declarative

Imperative

Cooking from Scratch



Imperative vs Declarative

Declarative

Ready-to-eat meal

Imperative vs Declarative

Imperative

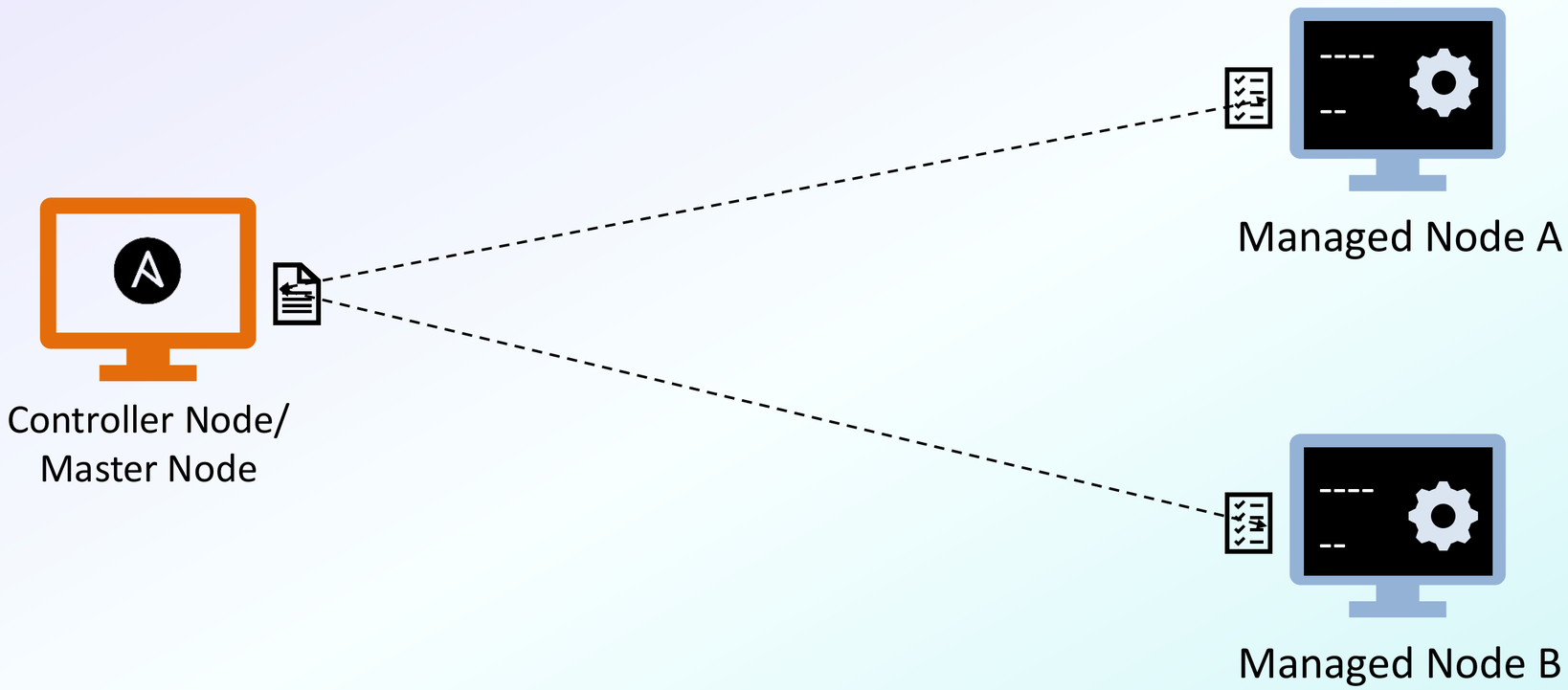
```
#!/bin/bash

echo "Enter the username: "
read username


echo "Creating user $username"
sudo useradd -m -s /bin/bash $username
```

Declarative

```
---
- name: Creating user Alex
  hosts: all
  ansible.builtin.user:
    name: Alex
    state: present
```

1.5

How Ansible Works?



Execution Flow

Stage 0: Create the Playbook

webserver.yaml

webserver.yaml

```
---
- hosts: webservers
  become: yes
    tasks:
    - name: Ensure Apache is installed
      ansible.builtin.apt:
        name: apache2
        state: present
        update_cache: yes
    - name: Start and enable Apache service
      ansible.builtin.service:
        name: apache2
        state: started
        enabled: yes
```

Execution Flow

Stage 1: Run the Playbook

Controller Node

```
ansible-playbook webserver.yaml
```

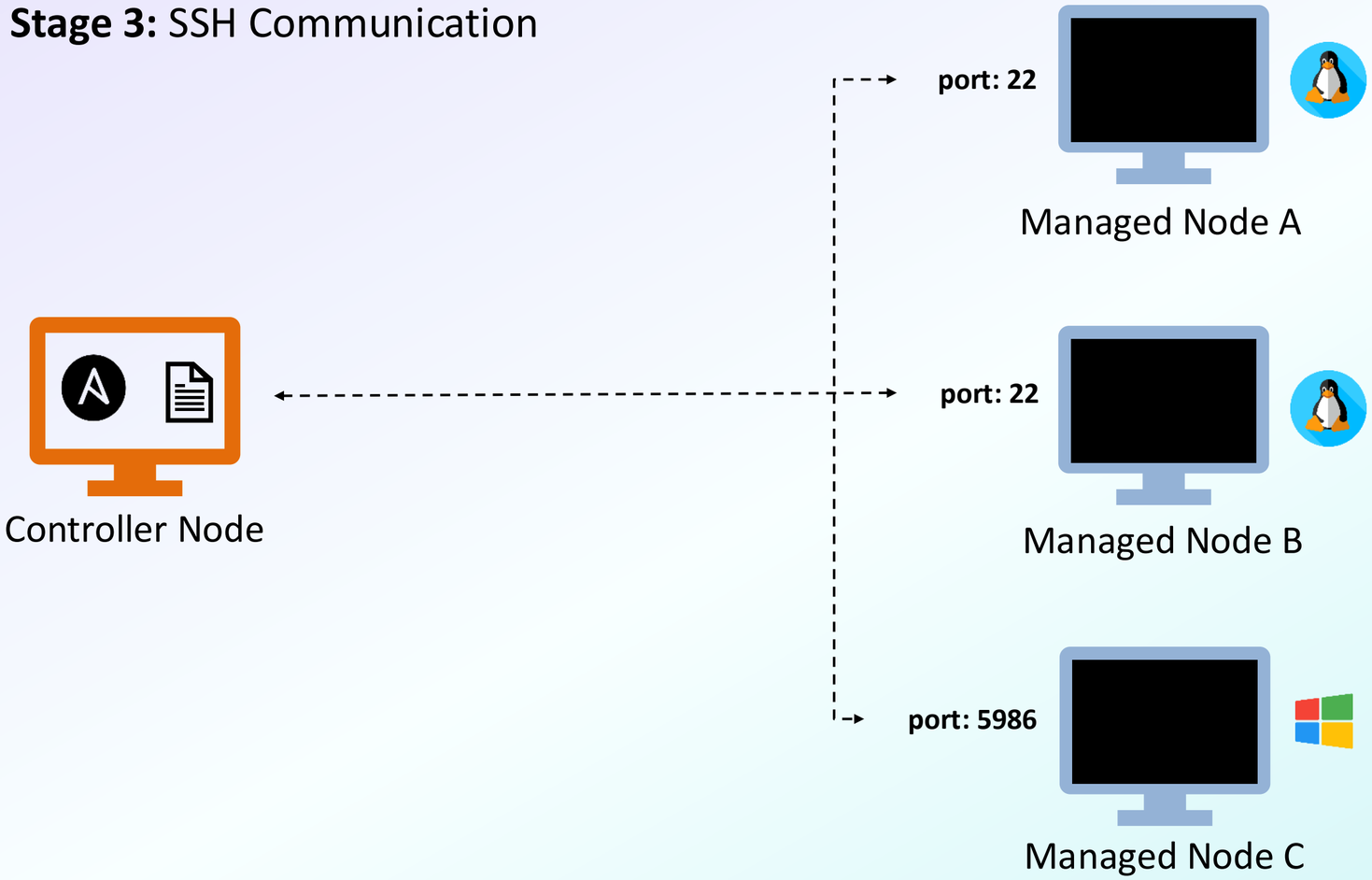
Execution Flow

Stage 2: Controller Node Processing

Controller Node

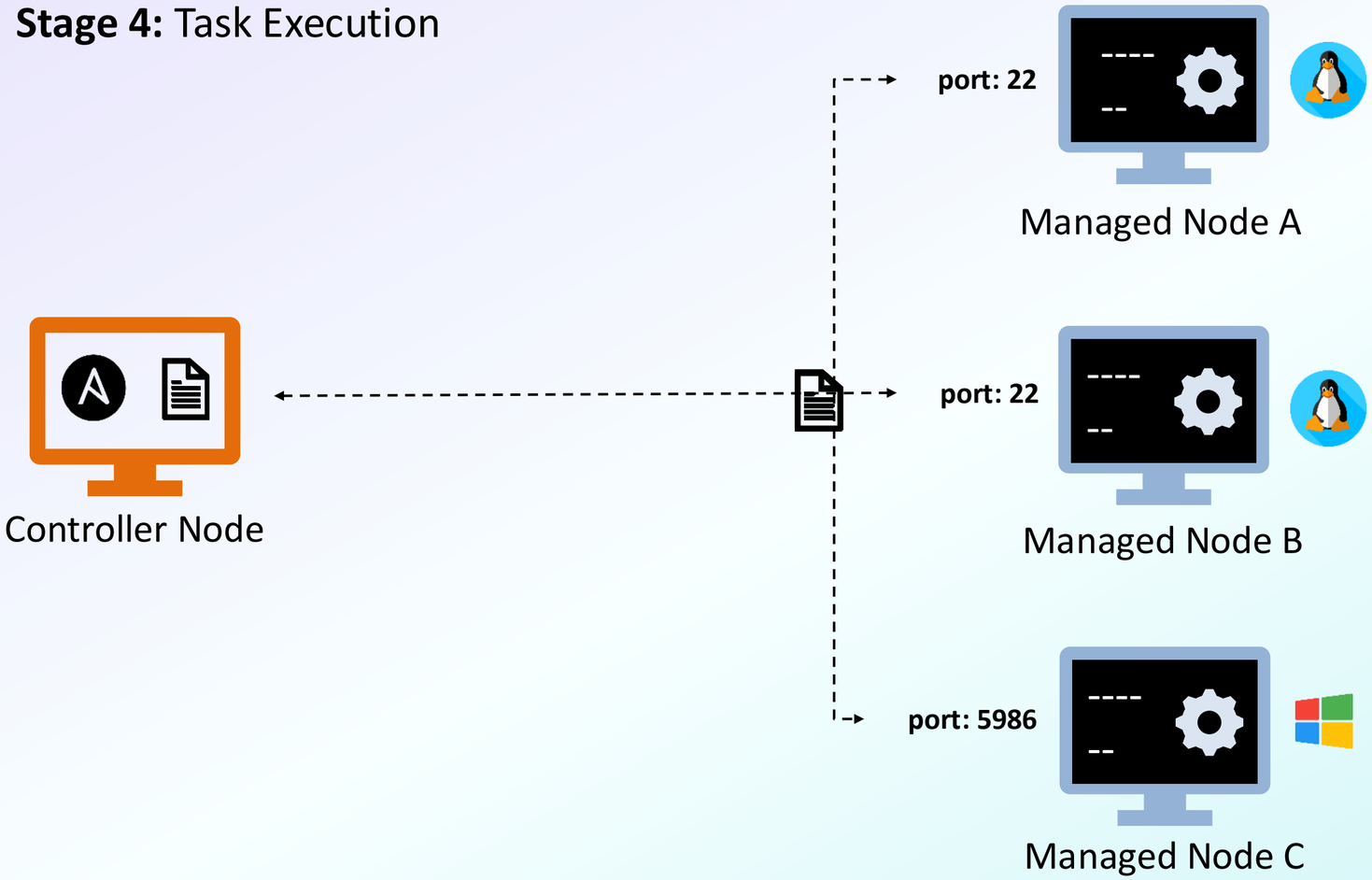
Execution Flow

Stage 3: SSH Communication



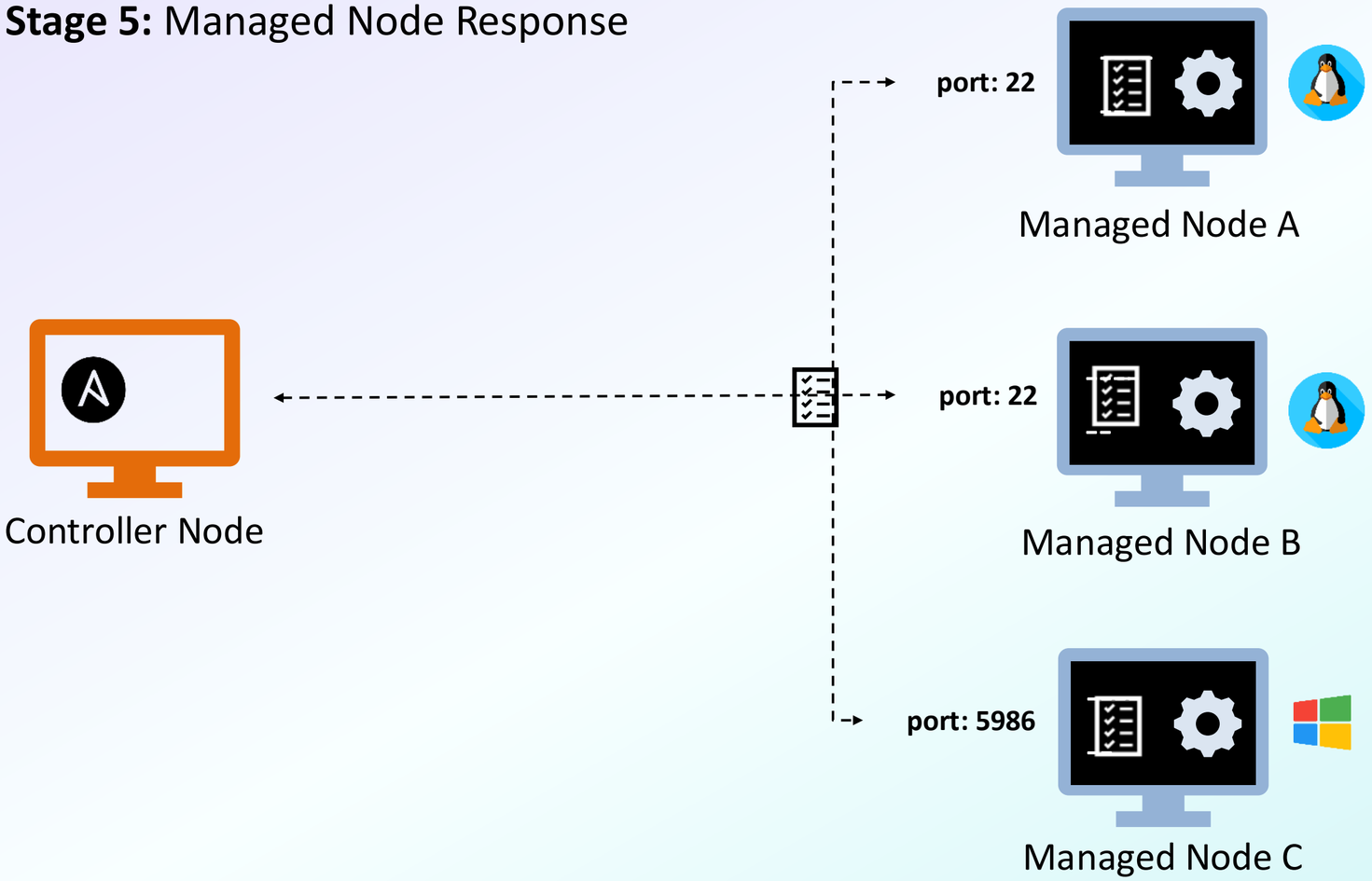
Execution Flow

Stage 4: Task Execution



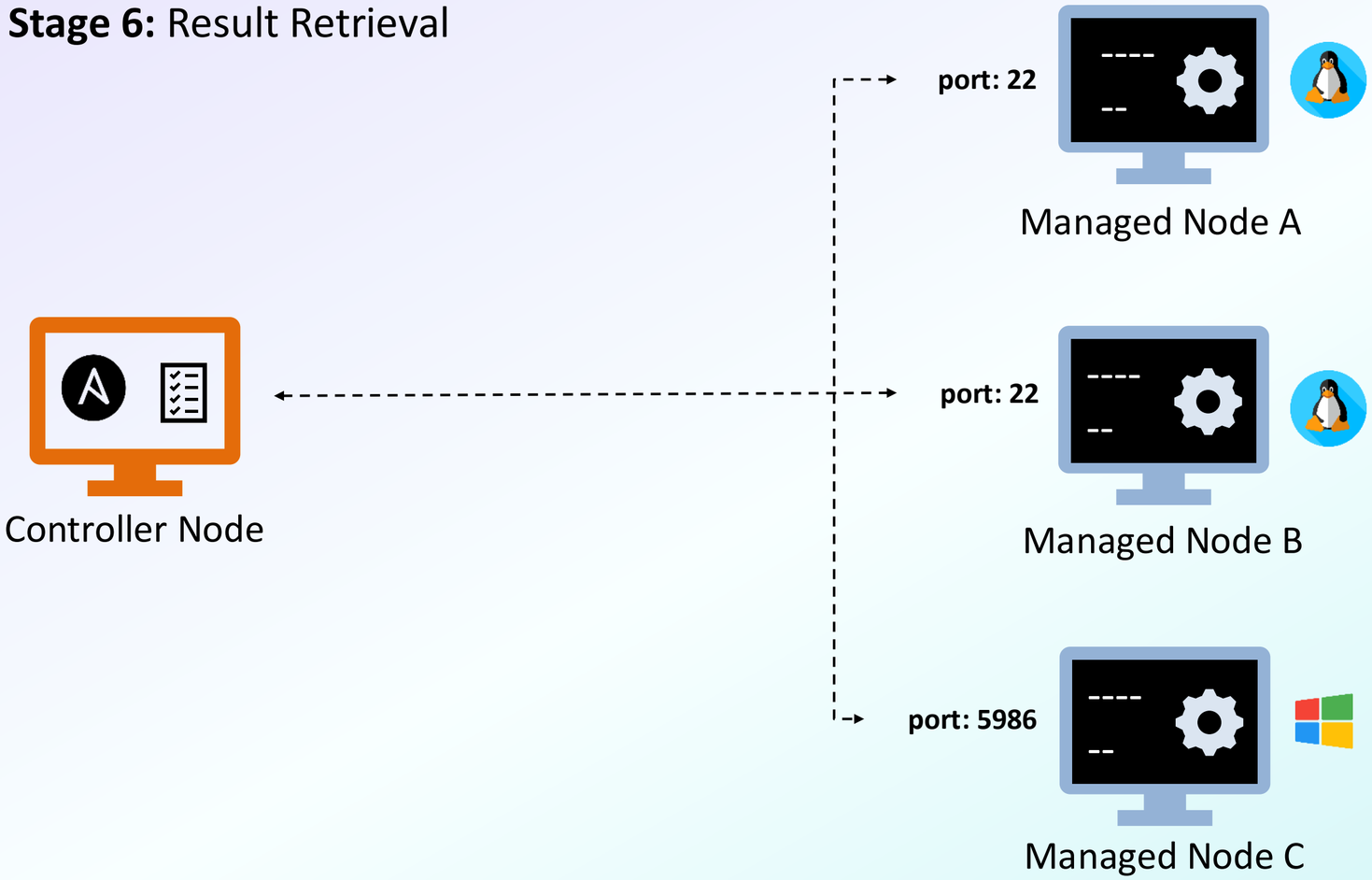
Execution Flow

Stage 5: Managed Node Response



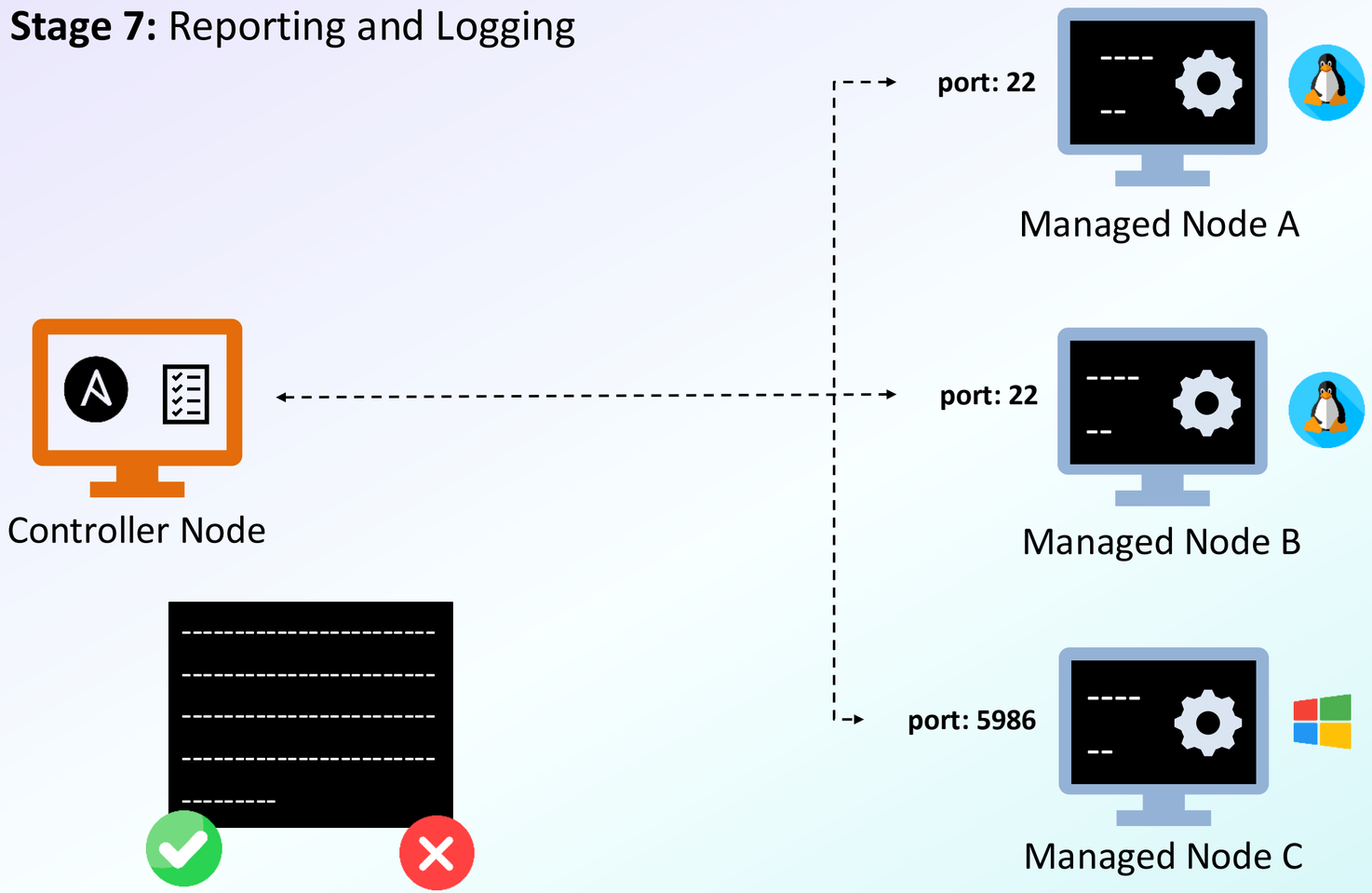
Execution Flow

Stage 6: Result Retrieval



Execution Flow

Stage 7: Reporting and Logging



1.6

Essential Ansible Terminologies

Ansible Manager Console AWX

Inventory

```
group1:
  hosts:
    webserver
    dbserver
    server1
```

Vault

Modules

- Ad-Hoc
- Core
- Custom

YAML

```
---
- name: thinknyx
  hosts: group1
  remote_user: root
```

Playbooks

```
---
- name: thinknyx_user
  hosts: group1
  remote_user: root

  tasks:
    - name: Create user
      ansible.builtin.user:
        name: thinknyx
        state: present
        shell: /bin/bash
```

Roles

user.yml

service.yml

install.yml

Jinja Template

```
The hostname is {{
ansible_facts['hostna
me'] }}
```

Collections

hosts.yml

Roles

Modules

Plugins

Ansible Galaxy

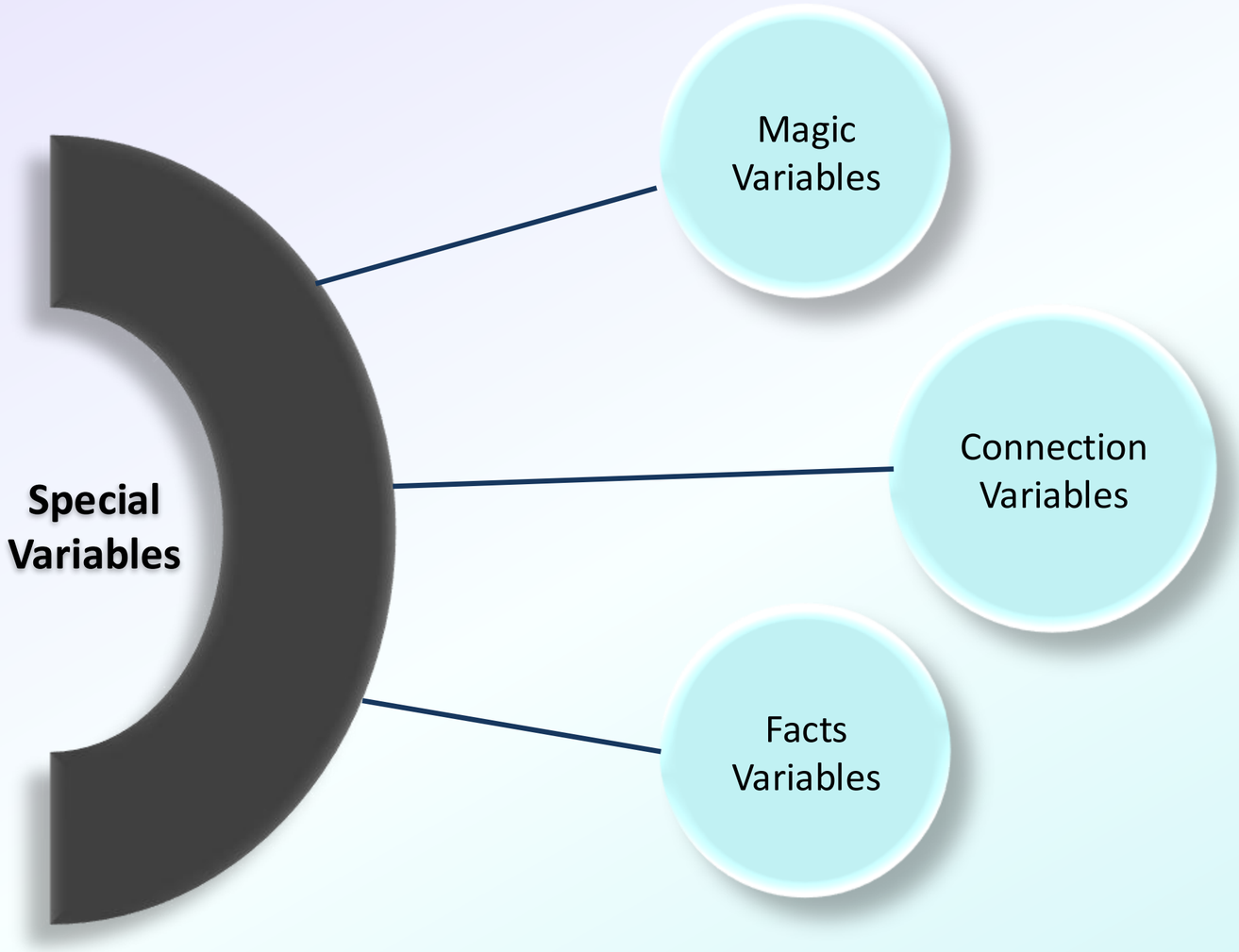
```
galaxy.anisble.com

Roles and Collections:
✓ Discovering
✓ Downloading
✓ Sharing
```

ANSIBLE

Master Node / Controller Node

Managed Node

Special Variables in Ansible



ANSIBLE_

1.7

Connection Plugins in Ansible

Master Node /
Controller Node

ssh protocol

Managed Node

- ✓ Email
- ✓ Reporting
- ✓ Logging

Special Variables in Ansible

- ✓ Magic Variables
  - ✓ cannot be set directly by user
  - ✓ many magic variables starts from ANSIBLE_
  - ✓ reserved magic variables : hostvars, groups, group_names, inventory_hostname etc
- ✓ Facts Variables
  - ✓ contain information pertinent to the managed host
- ✓ Connection Variables
  - ✓ used to specify how to connect and execute actions on the managed nodes
  - ✓ most of connection variables corresponds to connection plugins

https://docs.ansible.com/ansible/latest/reference_appendices/special_variables.html#special-variables

Ansible Releases

ansible-core

- ✓ ansible.builtin Modules
- ✓ Designed for minimum installation
- ✓ Provides basic set of functionalities
- ✓ Current version of ansible-core is 2.17

ansible

- ✓ Referred as “batteries included”
- ✓ Covers the complete ansible ecosystem
- ✓ Includes built-in features, modules, functionalities, community-curated Ansible Collections etc
- ✓ Includes ansible-core package
- ✓ Commonly referred as ansible community package
- ✓ Latest version of ansible is 10

Ansible Releases

ansible-core

- ✓ ansible.builtin Modules
- ✓ Designed for minimum installation
- ✓ Provides basic set of functionalities

ansible

Additional features:

- ✓ Access to a vast collection of modules, plugins, and roles contributed by the community
- ✓ Automation capabilities across a broad spectrum of devices, systems, and platforms.

Ansible Releases

ansible

```
$ sudo apt update
$ sudo apt install software-properties-common
$ sudo add-apt-repository --yes --update ppa:ansible/ansible
$ sudo apt install ansible
```

ansible-core

```
$ sudo apt update
$ sudo apt install software-properties-common
$ sudo add-apt-repository --yes --update ppa:ansible/ansible
$ sudo apt install ansible-core
```

Ansible Releases

ansible

```
ubuntu@ansible:~#    ansible --version
ansible [core 2.16.7]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/ubuntu/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ……..

ubuntu@ansible:~#    ansible-doc -l |  wc –l
8755
```

ansible-core

```
ubuntu@ansiblecore:~ ansible --version
ansible [core 2.16.7]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/ubuntu/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ……..

ubuntu@ansiblecore:~ ansible-doc -l |  wc –l
71
```

Ansible Release Cycle

Ansible Community Package Release	Status	Core version dependency
10.0.0	In development (unreleased)	2.17
9.x Changelogs	Current	2.16
8.x Changelogs	Unmaintained (end of life) after Ansible 8.7.0	2.15
7.x Changelogs	Unmaintained (end of life)	2.14
6.x Changelogs	Unmaintained (end of life)	2.13
5.x Changelogs	Unmaintained (end of life)	2.12
4.x Changelogs	Unmaintained (end of life)	2.11
3.x Changelogs	Unmaintained (end of life)	2.10
2.10 Changelogs	Unmaintained (end of life)	2.10

Demo

Ansible Documentation Overview

Summary

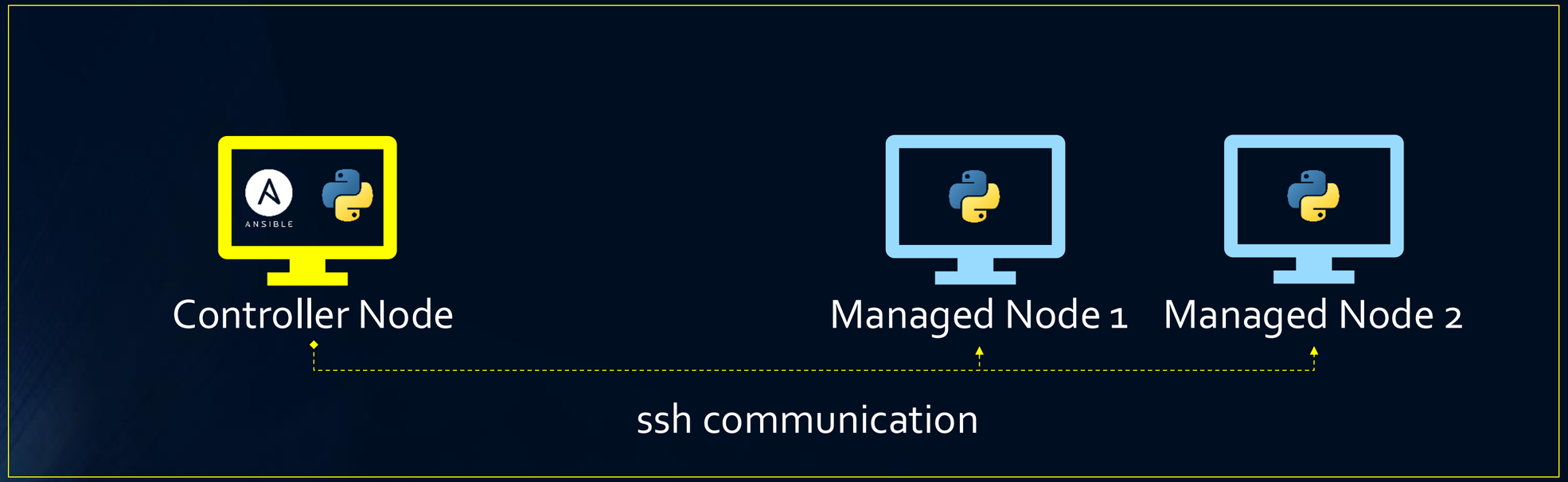
- ✓ What is Ansible
- ✓ Pre-requisite concepts
    - Configuration management
    - Infrastructure as code
    - Idempotency
    - Imperative and declarative programming approaches
- ✓ Ansible architecture
- ✓ Essential Terminologies
- ✓ Ansible package options

Installing & Configuring Ansible

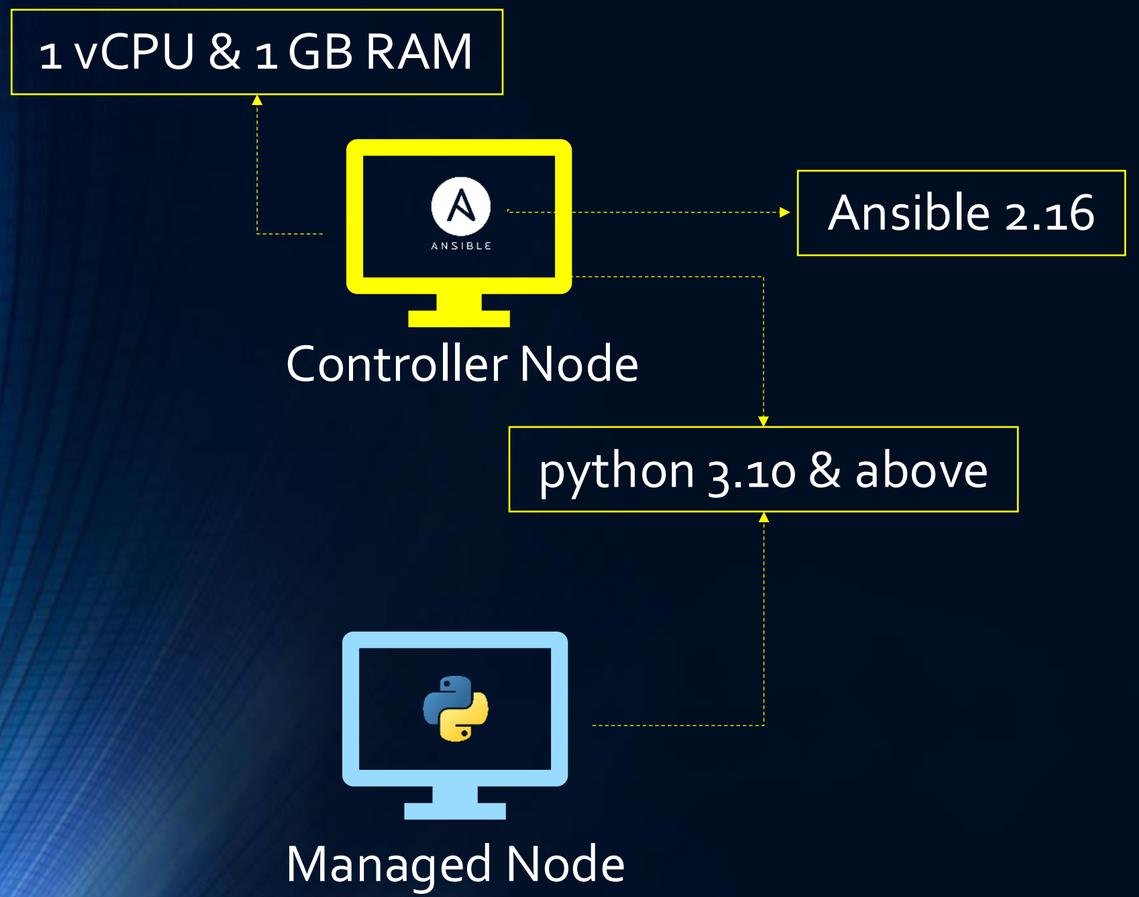
- Overview
- Demo environment specs
- Prepare nodes
- Setup SSH Keys
- Install Ansible & Validate

Thinknyx®

Overview



Overview



- ✓ Host for Controller Node
- ✓ Supported OS (Red Hat, Debian, Ubuntu, macOS, BSDs etc.), Windows with WSL
- ✓ Network connectivity between all nodes
- ✓ User account for SSH
- ✓ Persistent hostname
- ✓ Ansible 2.16 supports Python 2.7 & PowerShell version 3 to 5.1
- ✓ Latest Ansible Documentation

Overview

Summary of Installation Steps

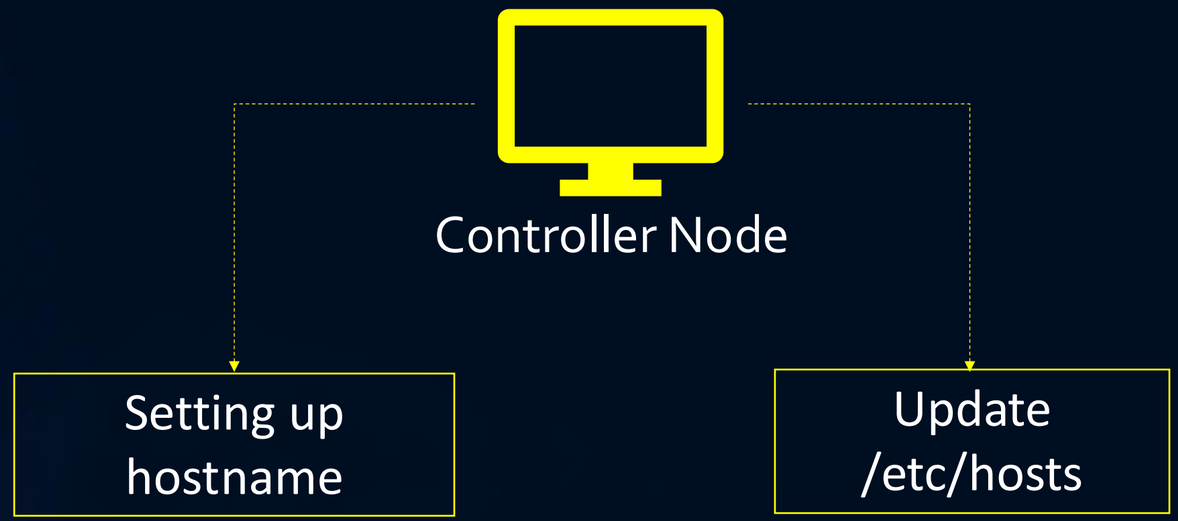
- ✓ Preparing the controller node
- ✓ Setting up SSH keys for passwordless communication
- ✓ Installing Ansible on the controller node
- ✓ Validating the Ansible installation

Demo

Preparing the controller node

Overview: Steps

- ✓ Preparing the controller node

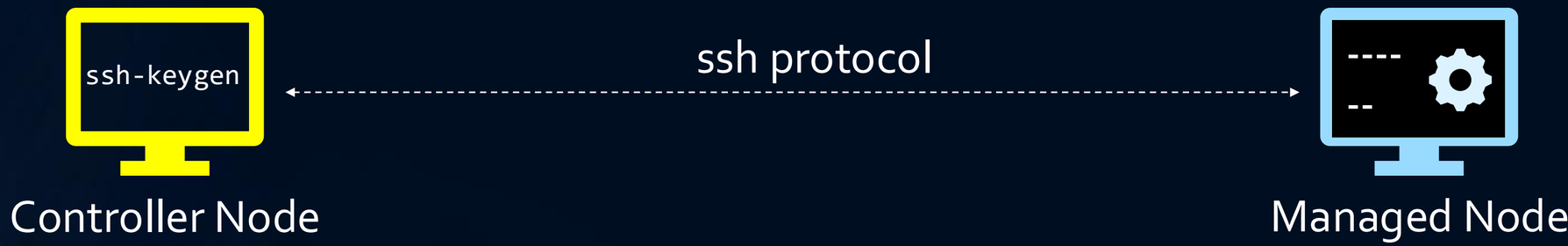


Demo

Setting up SSH keys for passwordless communication

Overview: Steps

- ✓ Setting up SSH keys for password less communication



Demo

Installing ansible on the controller node

Overview: Steps

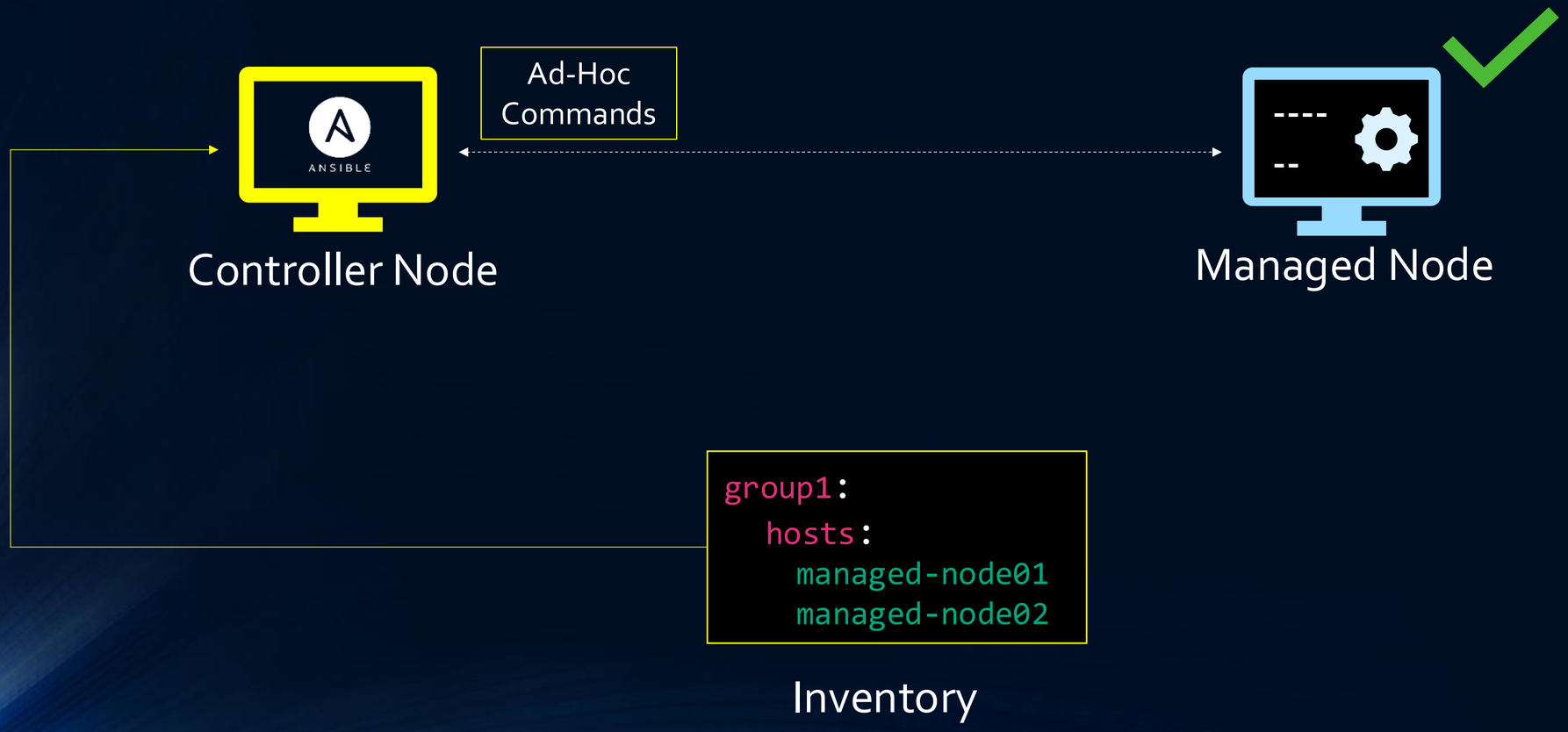
- ✓ Installing Ansible on the controller node

Controller Node

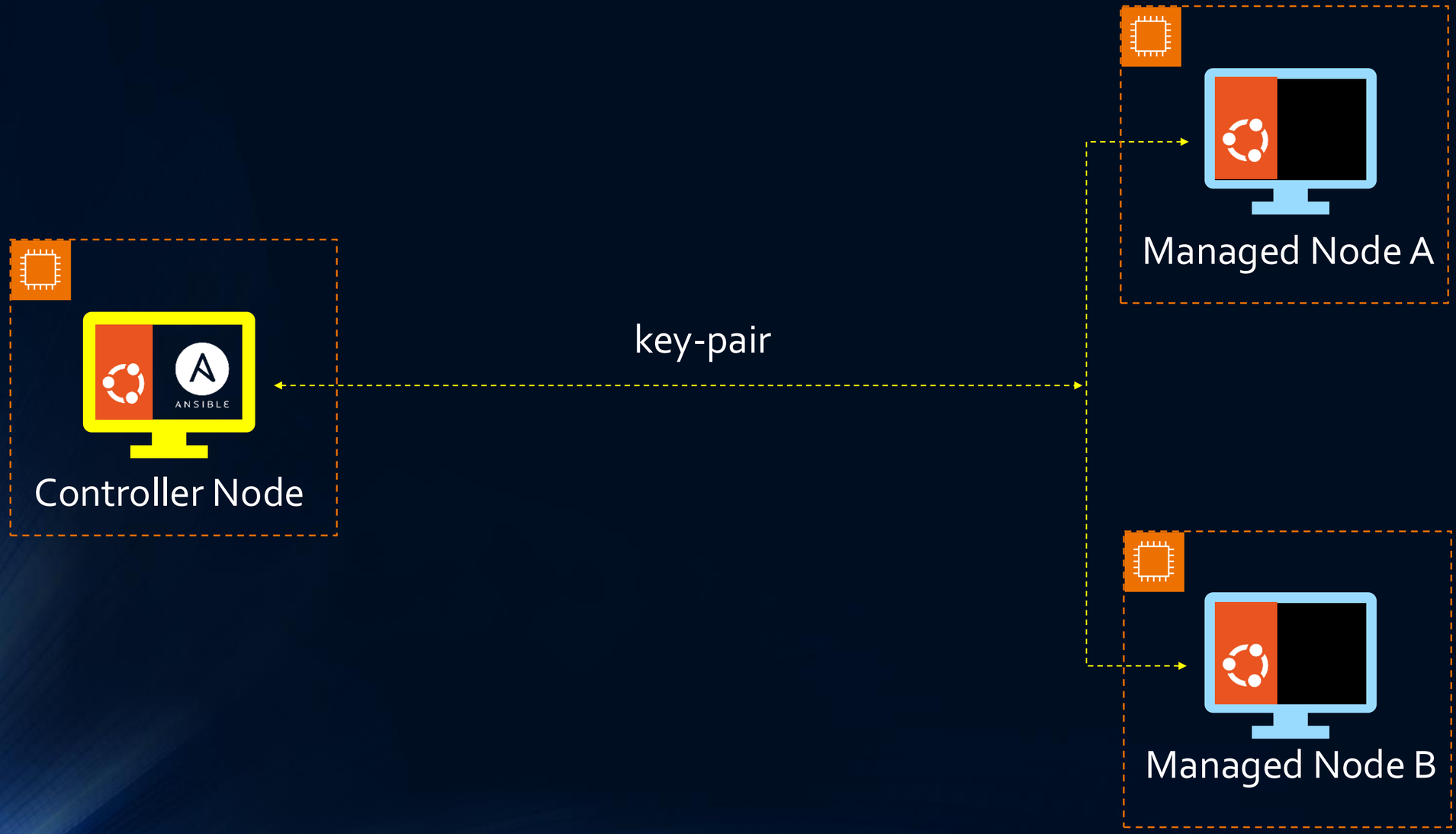
```
sudo apt update
sudo apt install software-properties-common
sudo add-apt-repository --yes --update ppa:ansible/ansible
sudo apt install ansible
ansible --version
```

Overview: Steps

- ✓ Validating the Ansible installation



Introduction to the Environment



Introduction to the Environment

- ✓ Ubuntu VM x 3
- ✓ 1 vCPU and 1 GB RAM
- ✓ Internet
- ✓ PowerShell or Terminal
- ✓ Modern browser

Demo

Validating the
Ansible installation

Summary

- ✓ Preparing the Controller Node
- ✓ Setting Up SSH Keys for Passwordless Communication
- ✓ Install Ansible on Controller Node
- ✓ Validate Installation by executing Ad-Hoc Commands

Ansible Inventory

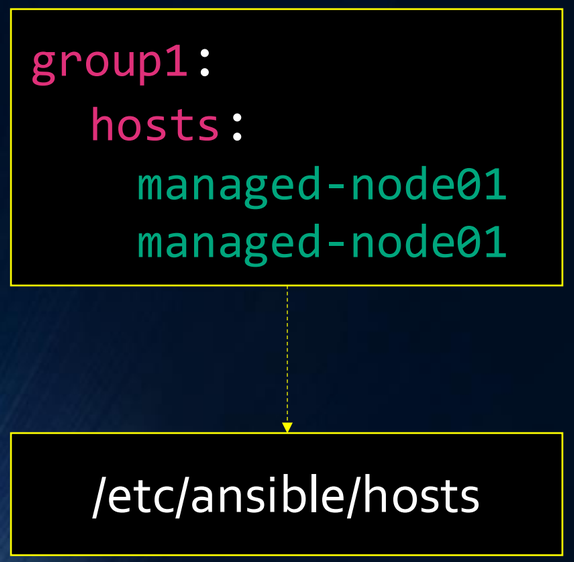
- Types of Inventories
- File Formats
- Hosts & Groups
- Custom Inventory Files
- Organizing Inventory Files
- *ansible-inventory* command line tool
- Variables in Inventory files

Thinknyx®

Overview of Ansible Inventory


```
group1:
  hosts:
    managed-node01
    managed-node01
```

Overview of Ansible Inventory

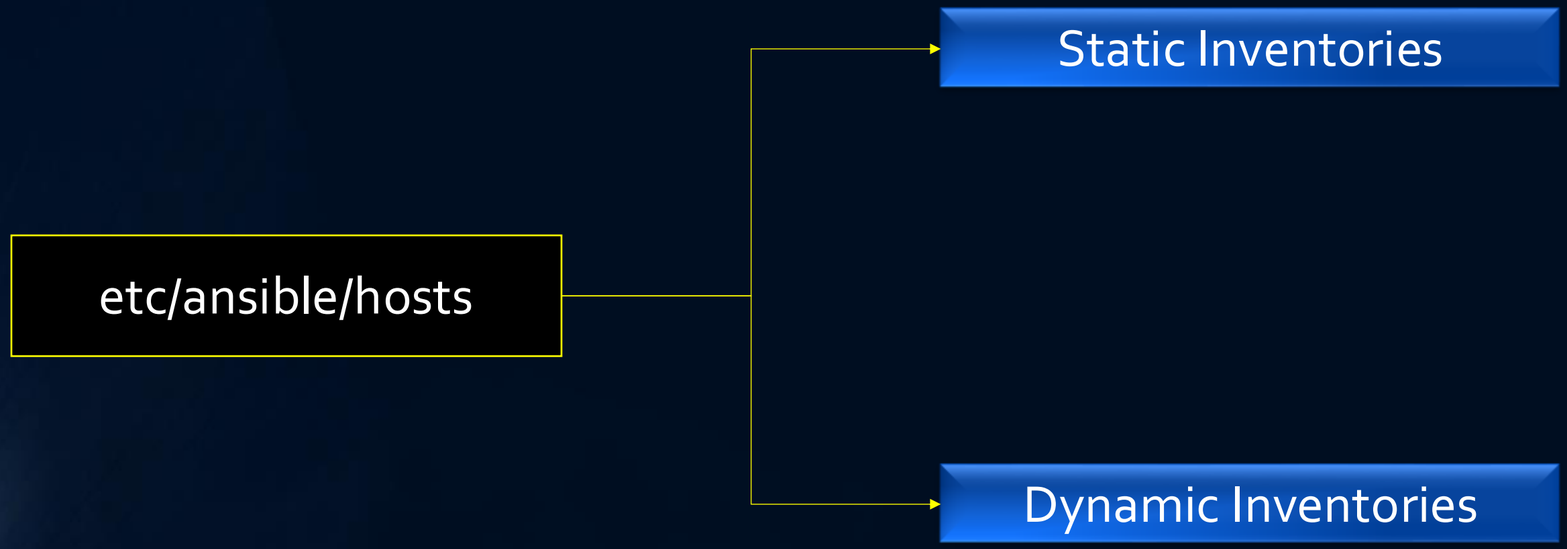


- ✓ List of nodes managed for automation, organized through inventory files
- ✓ Can be singular or grouped for efficient management and execution of tasks across multiple hosts
- ✓ Facilitates organized management by allowing nodes to be grouped, improving efficiency and task execution
- ✓ Specific hosts or groups can be targeted for tasks using patterns based on the inventory setup

Overview of Ansible Inventory

- ✓ Execute tasks on hosts through inventories
- ✓ Better approach is to create inventory files instead of using hostnames directly on command line
- ✓ Inventory files define managed nodes and groups
- ✓ Patterns target specific hosts/groups for tasks
- ✓ Default inventory file location : **/etc/ansible/hosts**

Types of Ansible Inventory



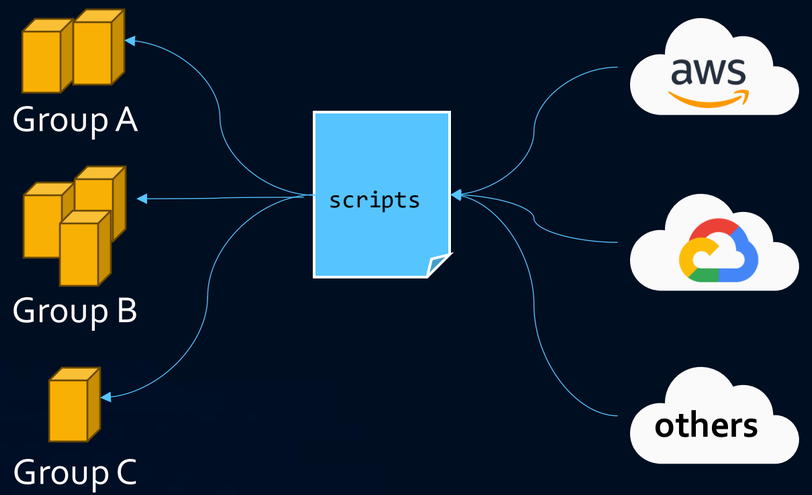
Types of Ansible Inventory

Static Inventories

- ✓ Text-based file with a predefined list of hosts or IP addresses
- ✓ Hosts are organized into groups for better management
- ✓ Commonly stored in INI or YAML file formats

Dynamic Inventories

- ✓ Utilizes scripts (often written in Python, PHP, or other languages) to manage host information dynamically
- ✓ Ideal for cloud setups where hosts may change frequently
- ✓ Reduces human error by automatically updating host information



Static Inventory: Formats, Hosts, Groups

- ✓ INI and YAML stand out as the most prevalent formats

1. INI Format

```
# INI format
mail.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
server1.example.com
server2.example.com
server3.example.com
```

- ✓ INI format allows to organize infrastructure into logical groups
- ✓ groups are denoted by square brackets-enclosed headings
- ✓ serves dual purpose:
  - categorizing hosts for organizational clarity
  - specifying target hosts for Ansible's Operations

Static Inventory: Formats, Hosts, Groups

1. YAML Format

```
ungrouped:
  hosts:
    mail.example.com:
webservers:
  hosts:
    foo.example.com:
    bar.example.com:
dbservers:
  hosts:
    one.example.com:
    two.example.com:
    three.example.com:
```

- ✓ In YAML, inventory is organized into section with hosts under the respective group
- ✓ Colons are used to separate keys from values. This is a fundamental part of the syntax and helps define mappings like dictionaries.
- ✓ Colons and indentation together define a hierarchical structure

Ansible's Default Groups: 'all' and 'ungrouped'

all

- ✓ all group is inclusive of every host listed in your inventory

ungrouped

- ✓ consists all hosts that are not assigned to any specific group apart from all

```
ungrouped:
  hosts:
    mail.example.com:
webservers:
  hosts:
    foo.example.com:
    bar.example.com:
dbservers:
  hosts:
    one.example.com:
    two.example.com:
    three.example.com:
```

Automatically included in both all and ungrouped groups

finds its place in both all group and dbservers group

Assigning Hosts

Adding Ranges of Hosts

✓ Ansible simplifies inventory management with ranges

1. Numeric Ranges in INI Format

```
[app_servers]
app[01:10].thinknyx.com
```

Numeric Ranges in YAML Format

```
app_servers:
  hosts:
    app[01:10].thinknyx.com:
```

2.. Adding a Stride to Numeric Ranges:

```
[db_servers]
db[01:20:2].thinknyx.com
```

3. Alphabetic Ranges

```
[storage_servers]
storage-[a:d].thinknyx.com
```

Adding Hosts to Multiple Groups

- ✓ Ansible's inventory flexibly assigns hosts to multiple groups, facilitating role, location, and deployment definition
- ✓ Hosts can be categorized based on their function, location, and development lifecycle stage.

what

Defines the function or role of the host, such as database server or web server

where

Represents the physical or geographical location, such as us_east and us_west

when

Indicates the stage in the development lifecycle, aiding in resource segregation for production or testing

Adding Hosts to Multiple Groups

```
ungrouped:
  hosts:
    gateway.example.com:
web_servers:
  hosts:
    frontend-1.example.com:
    frontend-2.example.com:
database_servers:
  hosts:
    db-node-1.example.com:
    db-node-2.example.com:
    db-node-3.example.com:
us_east:
  hosts:
    frontend-1.example.com:
    db-node-1.example.com:
    db-node-2.example.com:
us_west:
  hosts:
    frontend-2.example.com:
    db-node-3.example.com:
production:
  hosts:
    frontend-1.example.com:
    db-node-1.example.com:
    db-node-2.example.com:

testing:
  hosts:
    frontend-2.example.com:
    db-node-3.example.com:
```

serve as both web servers and are located in different regions

serve as database servers and are distributed across regions

Other Inventory options

```
-i <path> option
```

- ✓ Ansible supports a variety of formats and sources via Inventory plugins

Organizing inventory in a directory

- ✓ Implements a structured directory containing various inventory files, allowing for the utilization of multiple formats such as YAML or INI

Working with dynamic inventory

- ✓ Leverages automatically automatic retrieval and listing of resources from cloud providers or dynamic environments

Dynamic external inventory system

- ✓ If Inventory fluctuates over time, Ansible offers two ways to connect with an external inventory system –
  - Inventory plugins
  - Inventory scripts

Structuring Ansible Inventory with Directories

- Example: Inventory Directory Structure

```
my_hosts/
  aws/
      production.yml       # AWS production environment
      staging.yml          # AWS staging environment
  azure/
      dev.yml              # Azure development environment
  on-prem/
     servers.yml           # On-premises servers
  teams/
     teamA.yml             # Team A's specific hosts and groups
     teamB.yml             # Team B's specific hosts and groups
```

- Targeting an Inventory Directory

```
ansible-playbook example.yml -i inventory
```

- Configuring Inventory Directories

```
[defaults]
inventory = /thinknyx/inventory/file.yml
```

Utilizing Multiple Inventory Sources in Ansible

- ✓ Ansible allows the use of multiple inventory sources simultaneously
- ✓ Ansible commands can accept multiple -i options, each specifying a different inventory source

```
ansible-playbook get_logs.yml -i staging -i production
```

- ✓ For a persistent setup, the ANSIBLE_INVENTORY environment variable can be configured to include multiple inventory sources

ansible-inventory tool

- ✓ ansible-inventory tool is a handy command-line tool, used to check & understand the current information in ansible inventory
- ✓ ansible-inventory can read Ansible inventory files in different formats, such as INI, YAML, and JSON

`--list`	shows a simple list of all the servers (hosts) in our inventory
`--graph`	displays the list of hosts in a tree-like view
`--help`	displays the lists of all options available in ansible-inventory tool

```
ansible-inventory -i <hosts-filename> <options>
```

Variables in Ansible Inventory

Declared In line with the Host

```
[thinknyx_servers]
ansible-managedone ansible_port=22 ansible_host=192.168.1.10 http_port=80
ansible-managedtwo http_port=80
```

Declared in a :vars Section

```
[thinknyx_servers]
ansible-managedone ansible_port=22 ansible_host=192.168.1.10
ansible-managedtwo

[thinknyx_servers:vars]
http_port=80
```

Ansible Modules, Plugins & Ad hoc commands

- Ansible Modules
  - Core Modules
  - Custom Modules
- Ansible Plugins
- Modules v/s Plugins
- Ad hoc commands
- Demonstration on Modules & Ad hoc commands

Thinknyx®

Introduction to Ansible Modules

Ansible Modules

- Small & self-contained pieces of code
- Majorly written in Python
- Used to execute tasks on the managed nodes

Fundamental building blocks of Ansible's automation

Introduction to Ansible Modules

Ansible Modules

- Small & self-contained pieces of code
- Used to execute tasks on the managed nodes

Fundamental building blocks of Ansible's automation

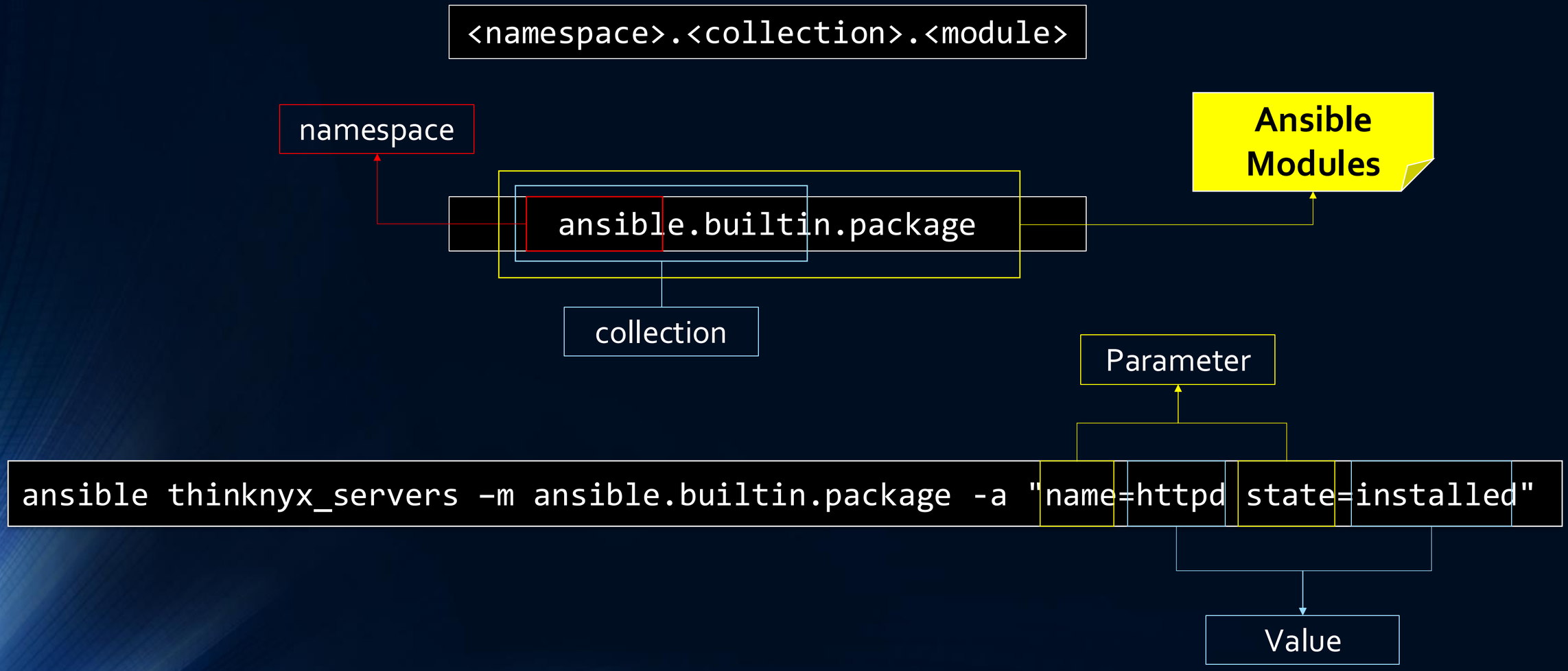
Executing commands

Installing packages

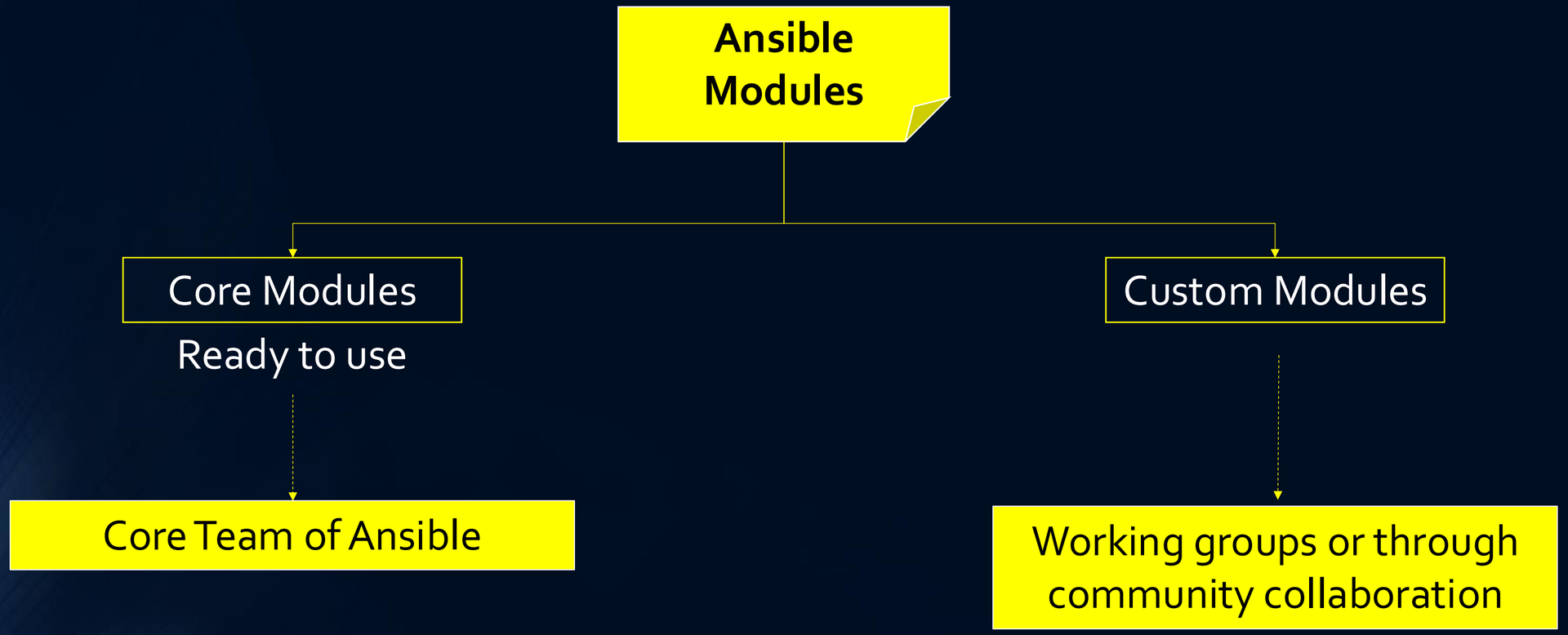
Managing system resources

File and directory management

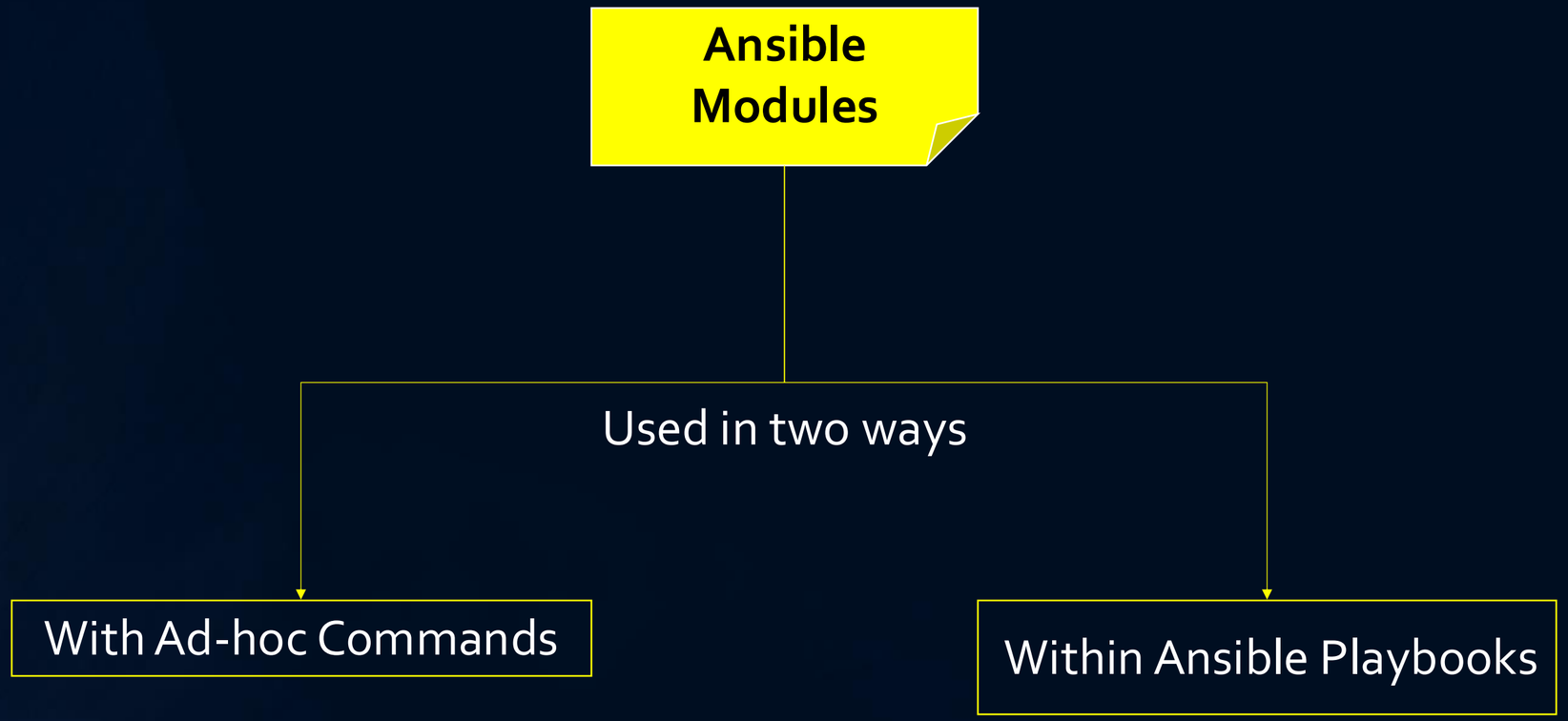
Ansible Core Modules



Introduction to Ansible Modules



Introduction to Ansible Modules



Ansible Core Modules

✓ To Install apache on Managed Nodes

- In Ad-hoc Commands

```
ansible thinknyx_servers –m ansible.builtin. package -a "name=httpd state= present"
```

- In Ansible Playbooks

```
- name: Install httpd package
  ansible.builtin.package:
    name: httpd
    state: present
```

Ansible Custom Modules

Extend Functionality

Core Modules

Custom Modules

```
def main():
   module = AnsibleCoreModule(
     ------------
     ------------
     ------------
   )

def custom():
 module = AnsibleCustomModule(
     ------------
     ------------
     ------------
   )
```

Python

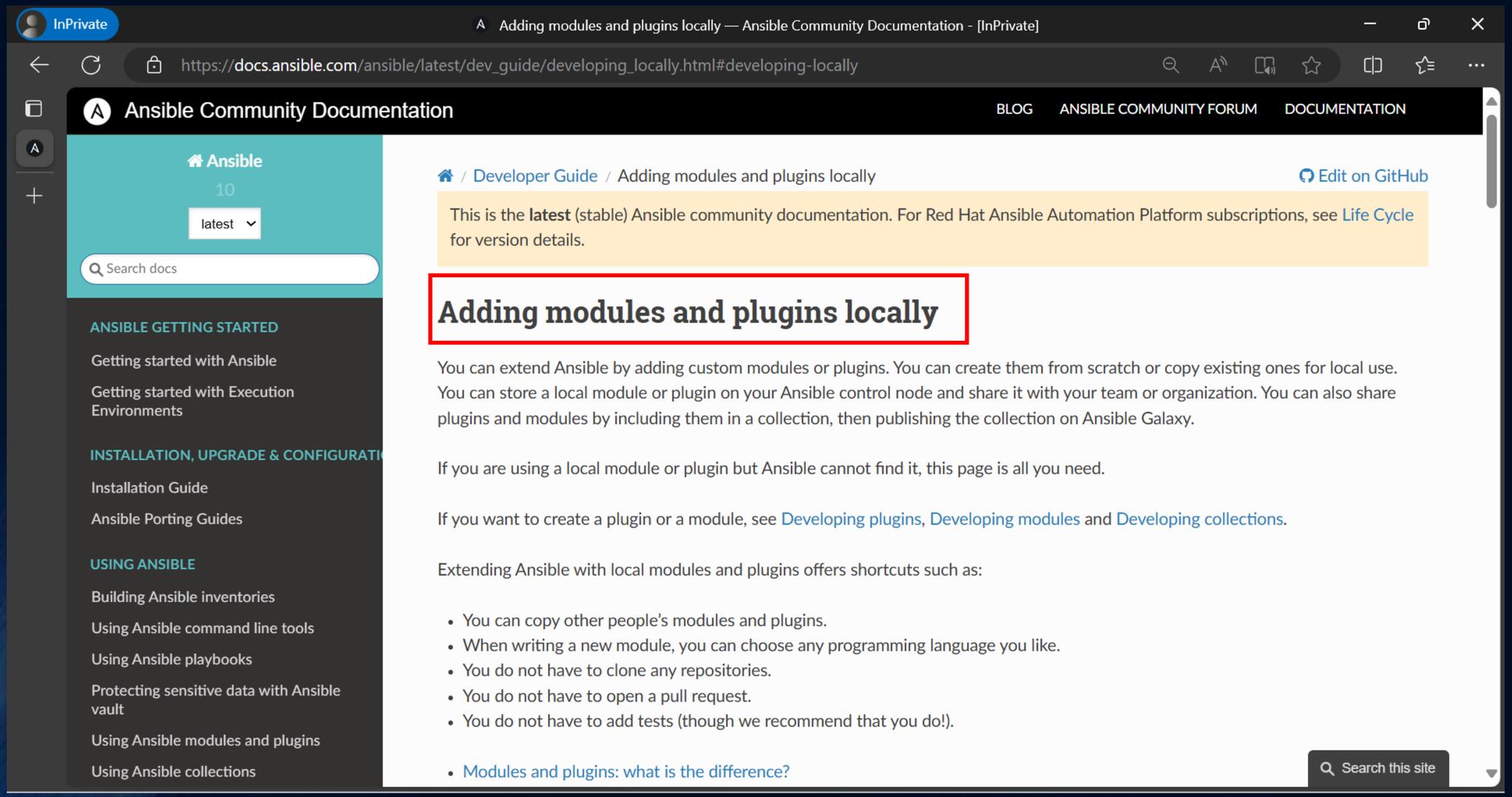
Shell

Ruby

Perl

JSON

Ansible Custom Modules



https://docs.ansible.com/ansible/latest/dev_guide/developing_locally.html#developing-locally

Ansible Plugins

Core Modules

Extra features & abilities

Plugins

- ✓ Transforming data
- ✓ Managing connections
- ✓ Logging information

- ✓ **Callback plugins:** Control & customize output during command-line execution
- ✓ **Lookup plugins:** Fetch data from external sources
- ✓ **Inventory plugins:** Dynamically discover & manage inventory from various sources
- ✓ **Filter plugins:** Provide custom Jinja2 filters to manipulate data within playbooks

Collection

Ansible Plugins

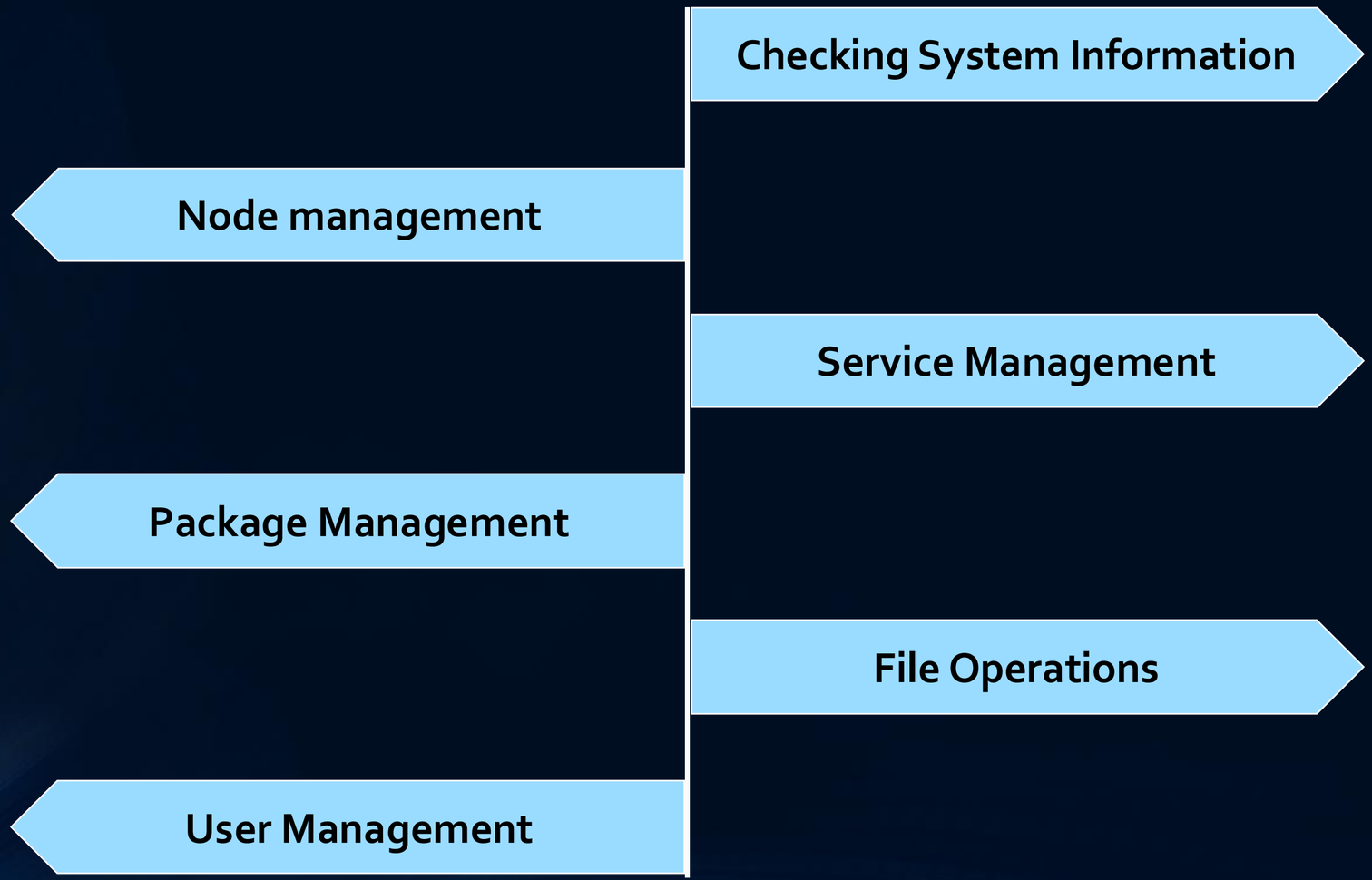
FACTORS	PLUGINS	MODULES
Purpose	Extend Ansible's core functionality	Perform specific automation tasks on target nodes
Functionality/ Use cases	Transforming data, logging output, and connecting to inventory	Installing packages, managing services, and configuring systems
Language	Must be written in Python	Must be written in Python or PowerShell for Ansible core, Can be written in any language for local use
Execution	Most plugins execute on the control node within the **`/usr/bin/ansible`** process	Executes in their own process on the managed node
Interface	Integrate directly with Ansible's core	Interact with Ansible primarily through JSON

Overview of Ad-Hoc Commands

- ✓ Ad hoc commands provide a direct and instant way to communicate with managed nodes in Ansible
- ✓ Ad hoc commands are perfect for on-the-fly operations
- ✓ Ad hoc commands eliminate the need for intricate scripts or elaborate setups
- ✓ Uses Ansible modules to execute tasks
- ✓ Ad hoc commands do not provide a history of executions and are not reusable

Overview of Ad-Hoc Commands

Use Cases:



Overview of Ad-Hoc Commands

Controller Node

- ✓ Ad hoc commands allow you to target specific managed nodes based on roles, collections, and more
- ✓ Can also be used to target all managed nodes in an inventory

Overview of Ansible Inventory

```
$ ansible "*" -m ansible.builtin.package -a "name=apache2 state=present"
```

Syntax for Ad-Hoc Commands

```
ansible <pattern> -m <module_name> -a "<module options>"
```

```
$ ansible "*" -m ansible.builtin.ping
```

- `ansible` — command-line utility
- `"*"` — all managed nodes
- `-m ansible.builtin.ping` — Specify module to use

```
$ ansible 'managed-node01' -m ansible.builtin.ping
```

```
$ ansible -l 'managed-node01, managed-node02' -m ansible.builtin.ping
```

```
$ ansible '*' -m ansible.builtin.shell -a 'hostname'
```

Ansible Custom Modules

Extend Functionality

Core Modules

Custom Modules

```
def main():
   module = AnsibleCoreModule(
     ------------
     ------------
     ------------
   )
```

```
def main():
   module = AnsibleCoreModule(
     ------------
     ------------
     ------------
   )

def custom():
 module = AnsibleCustomModule(
     ------------
     ------------
     ------------
   )
```

Ansible Custom Modules

Extend Functionality

Core Modules

Custom Modules

```
def main():
   module = AnsibleCoreModule(
     ------------
     ------------
     ------------
   )
```

```
def main():
   module = AnsibleCoreModule(
     ------------
     ------------
     ------------
   )

def custom():
 module = AnsibleCustomModule(
     ------------
     ------------
     ------------
   )
```

Ansible Plugins

FACTORS	PLUGINS	MODULES
Purpose	Extend Ansible’s core functionality.	Perform specific automation tasks on target nodes
Functionality/ Use cases	Transforming data, logging output, and connecting to inventory	Installing packages, managing services, and configuring systems
Language	Must be written in Python.	Must be written in Python or PowerShell for Ansible core, Can be written in any language for local use
Execution	Most plugins execute on the control node within the /usr/bin/ansible process.	Executes in their own process on the managed node
Interface	Integrate directly with Ansible's core.	Interact with Ansible primarily through JSON

Ansible Playbooks

- Use Case of Playbooks
- Advantages over ad-hoc commands
- Structure of a playbook
- Developing and running playbooks
- Hands-on Demonstration
  - ansible-playbook
  - Check syntax, Validate and Execute Playbook
  - Multi-task playbooks

Thinknyx®

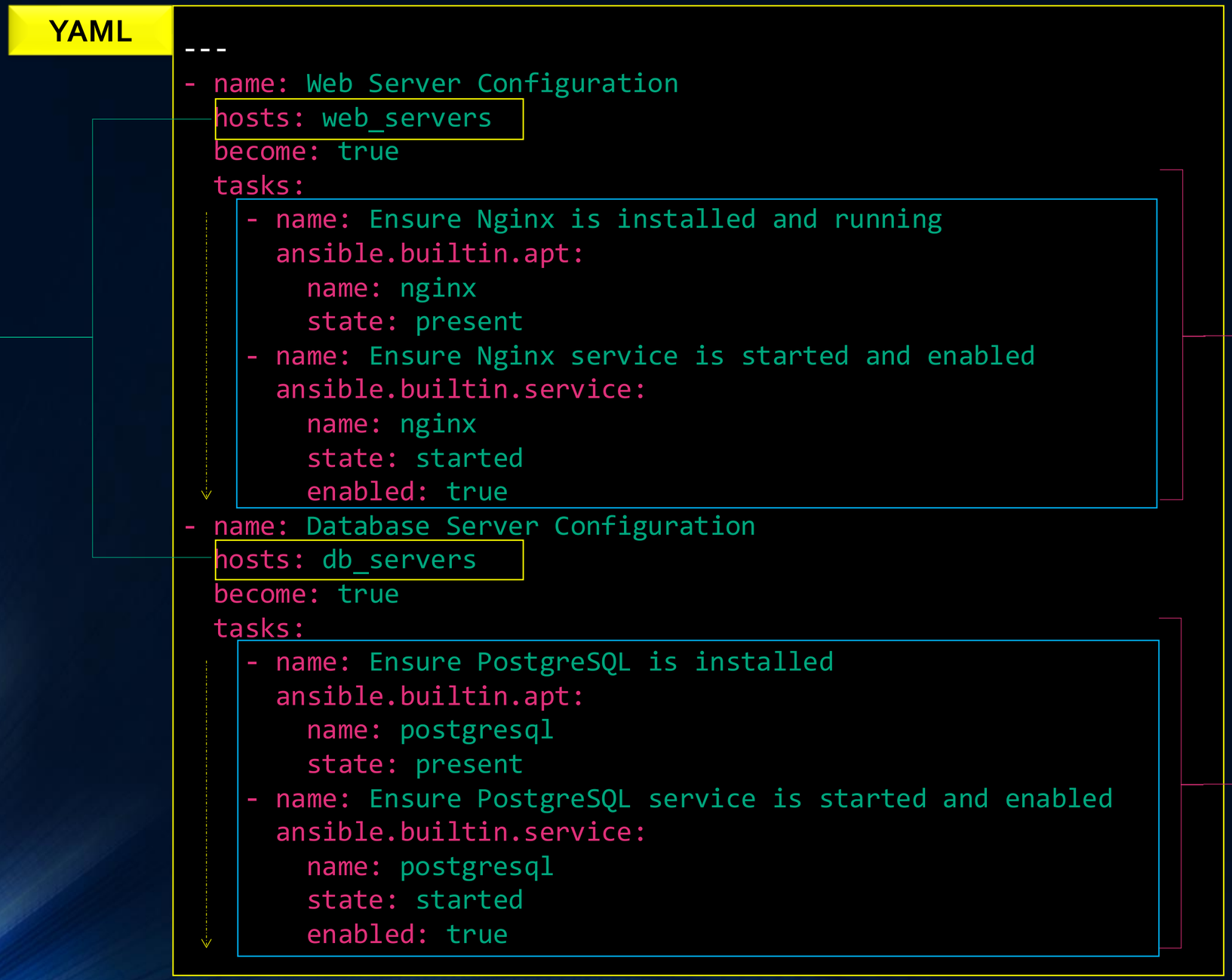
Introduction

Ansible Playbooks

Playbook

```
---
- name: thinknyx_user
  hosts: group1
  remote_user: root

  tasks:
    - name: Create user
      user:
        name: thinknyx
        state: present
        shell: /bin/bash
```

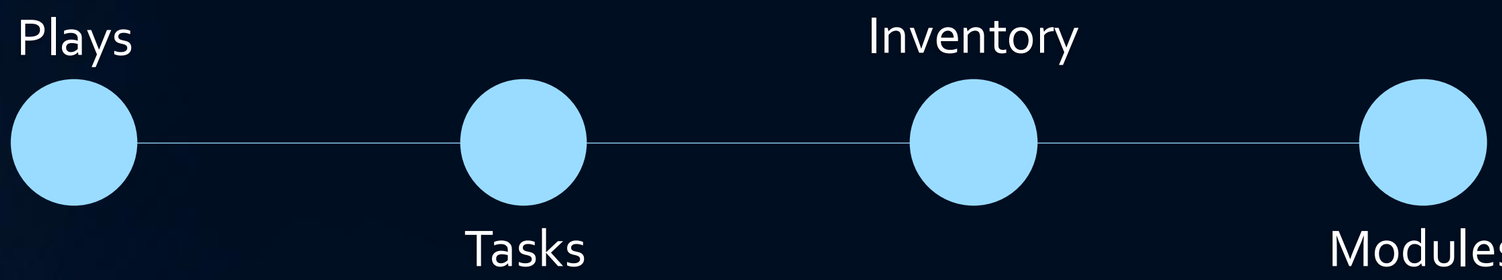
- ✓ Set of attributes
- ✓ Execution instructions
- ✓ Sequences of operations



Target hosts

List of tasks

Introduction



Benefits of Playbooks

Repeatable & Reusable

Configuration Management

Multi-machine Deployments

Source Control Integration

Modularity

Declarative Nature

YAML

Yet Another Markup Language

- ✓ Used to write Ansible Playbook
- ✓ Simple and human-readable format
- ✓ Case sensitive

YAML

Rule 1: Indentation

- ✓ Fixed indentation to represent relationships between data layers
- ✓ Two-space indentation for each level or any consistent number of spaces
- ✓ Tabs must be avoid

example.yml

```
user:
  name: john
  age: 30
```

YAML

Rule 2: Colons

- ✓ Dictionary keys are represented as strings followed by trailing colon
- ✓ Values expressed either as string following the colon or through indentation

example.yml

```
person:
  name: Allen
  occupation: Engineer
```

YAML

Rule 3: Dashes

- ✓ To denote a list of items, use a single dash followed by a space
- ✓ Items with the same level of indentation are part of same list

example.yml

```
fruits:
  - apple
  - banana
  - orange
```

YAML

Additional rules

- ✓ Comments are denoted by "#"

comments.yml

```
# This is a comment
person:
  name: Bob
```

- ✓ start indicator of YAML structure is three dashes "---"

start.yml

```
---
person:
  name: Charlie
```

http://www.yamllint.com/

Structure of Playbook

webserver.yml

```
---
- name: Verify apache installation
  hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root

  tasks:
    - name: Ensure apache is at the latest version
      ansible.builtin.yum:
        name: httpd
        state: latest

    - name: Ensure apache is running
      ansible.builtin.service:
        name: httpd
        state: started

  handlers:
    - name: Restart apache
      ansible.builtin.service:
        name: httpd
        state: restarted
```

Generic Section

Tasks Section

Handlers Section

Structure of Playbook

To create playbooks:

- ✓ Text editor
- ✓ Basic understanding of YAML

Structure of Playbook

webserver.yml

Generic Section

```
---
- name: Update Nginx configuration
  hosts: webservers
  become: yes
```

Tasks Section

```
  tasks:
    - name: Copy Nginx configuration file
      copy:
        src: /path/to/your/local/nginx.conf
        dest: /etc/nginx/nginx.conf
      notify: Restart Nginx
```

Handlers Section

```
  handlers:
    - name: Restart Nginx
      service:
        name: nginx
        state: restarted
```

Execution Flow

servers.yml

```
---
- name: Update web servers
  hosts: webservers
  remote_user: root

  tasks:
  - name: Ensure apache is at the latest version
    ansible.builtin.yum:
      name: httpd
      state: latest

  - name: Write the apache config file
    ansible.builtin.template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf

- name: Update db servers
  hosts: databases
  remote_user: root

  tasks:
  - name: Ensure postgresql is at the latest version
    ansible.builtin.yum:
      name: postgresql
      state: latest
```

Top-to-bottom approach

Set of Tasks

- ✓ Sequence of plays and tasks are important.

Play 1

Play 2

Execution Flow

- ✓ Common best practice is to write Playbooks with a single play

webservers.yml

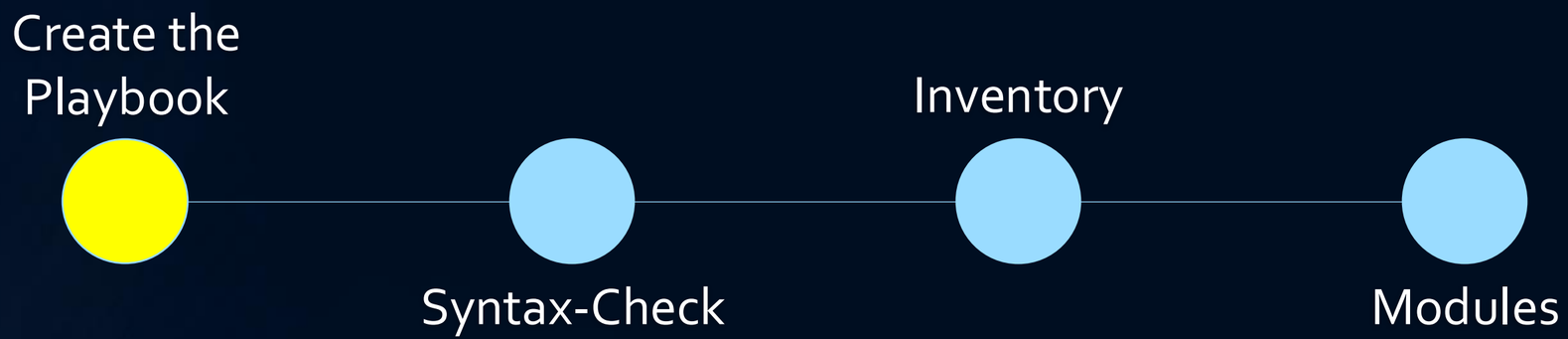
```
---
- name: Update web servers
  hosts: webservers
  remote_user: root

  tasks:
  - name: Ensure apache is at the latest version
    ansible.builtin.yum:
      name: httpd
      state: latest

  - name: Write the apache config file
    ansible.builtin.template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf
```

dbserver.yml

```
---
- name: Update db servers
  hosts: databases
  remote_user: root

  tasks:
  - name: Ensure postgresql is at the latest version
    ansible.builtin.yum:
      name: postgresql
      state: latest

  - name: Ensure that postgresql is started
    ansible.builtin.service:
      name: postgresql
      state: started
```

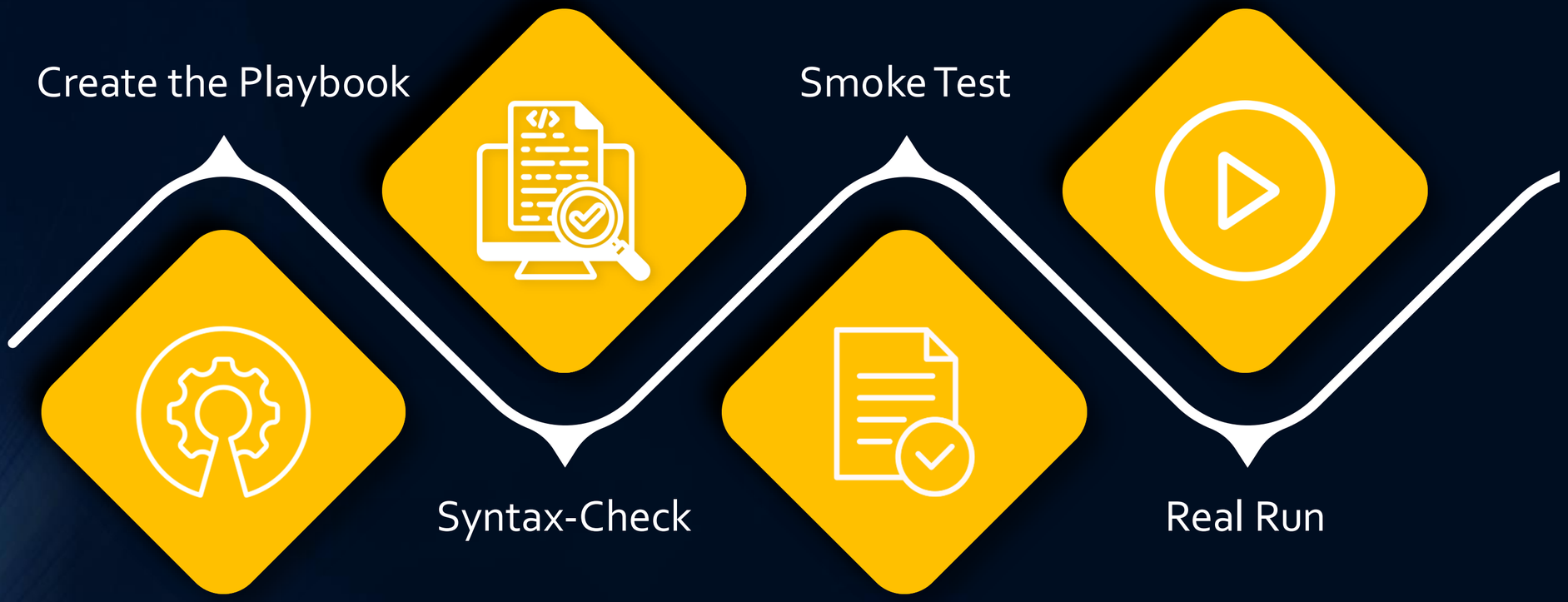
Single play Multiple tasks

A 4-step process for working with Playbooks



- Ansible scripting
- YAML
- Creating playbooks and roles

A 4-step process for working with Playbooks



Command Line Utility: ansible-playbook

- ✓ Use this command when running playbooks

Syntax

```
ansible-playbook <playbook_name.yml>
```

Command Line Utility: ansible-playbook

- ✓ To list the hosts targeted by the playbook

```
ansible-playbook example_playbook.yml --list-hosts
```

- ✓ To check for syntax errors

```
ansible-playbook example_playbook.yml --syntax-check
```

- ✓ To run the playbook in check mode

```
ansible-playbook example_playbook.yml --check
```

Command Line Utility: ansible-playbook

- ✓ To explore options and flags

```
ansible-playbook --help
```

Summary

- ✓ What are playbooks ?
- ✓ Introduction to YAML
- ✓ Structure of playbook
- ✓ ansible-playbook command line utility
- ✓ Creating playbooks with various use cases
- ✓ Checking for syntax errors
- ✓ Performing smoke test and execute them

Ansible for Windows

- Prerequisites for managing Windows servers
- Setting up inventory
- Windows-specific Ansible modules
- Run ad-hoc commands on Windows machines
- Create a Windows-specific playbook

Thinknyx®

Getting Started with Windows Targets

- **Step One: Launch Windows Server**
  - ✓ Create a Windows Server instance (cloud/local)
- **Step Two: Prerequisites**
  - ✓ Create a dedicated user
  - ✓ Verify PowerShell & .NET Framework
  - ✓ Set up WinRM (Windows Remote Management)
- **Step Three: Configure WinRM for Remote Access**
  - ✓ Uses port 5986
  - ✓ Enables secure remote management for Windows

Getting Started with Windows Targets

- **Step Four: Configure Ansible Inventory**

```
[windows]
thinknyxwindowsserver1
[windows:vars]
ansible_user=thinknyx
ansible_password=********
ansible_port=5986
ansible_connection=winrm
ansible_winrm_transport=basic
ansible_winrm_server_cert_validation=ignore
```

- **Step Five: Test Connectivity**

```
ansible "windows" -m ansible.windows.win_ping
```

- **Step Six: Start Automation**

  - ✓ Run ad-hoc commands
  - ✓ Create and execute Windows playbooks

Summary

- ✓ Configured Windows servers as Ansible target nodes
- ✓ Verified connectivity using win_ping module
- ✓ Hands-on demo with ad-hoc commands and playbooks for Windows

Summary

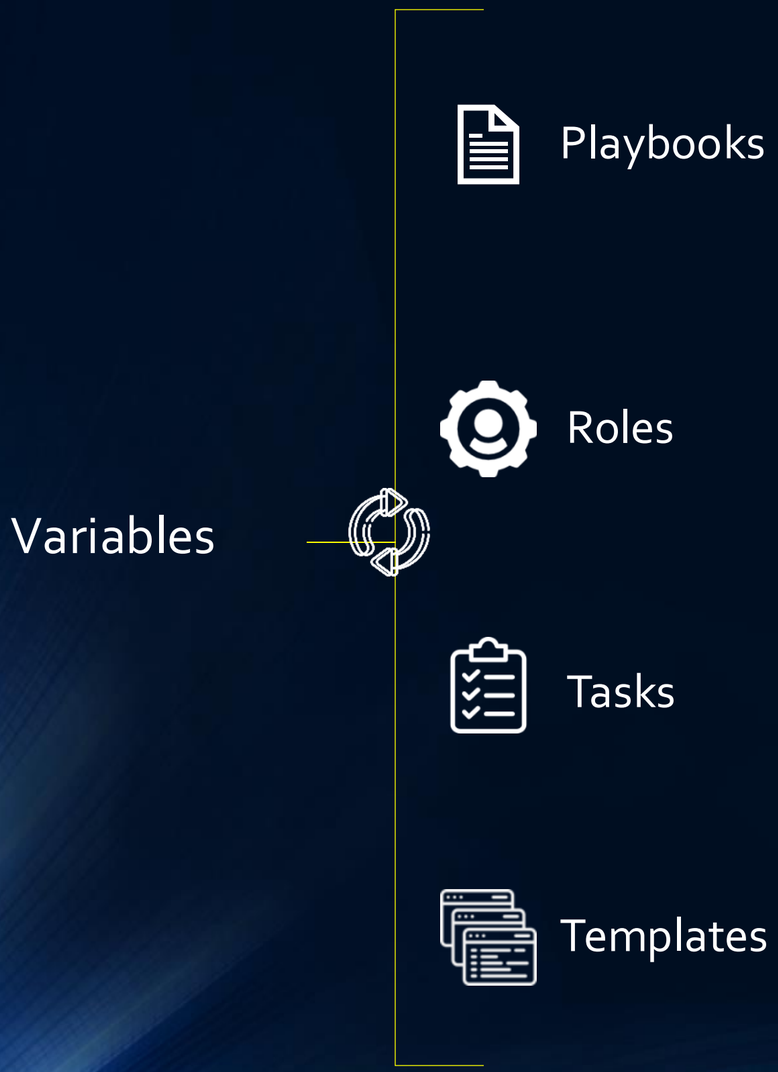
- ✓ Configured Windows servers as Ansible target nodes
- ✓ Verified connectivity using win_ping module
- ✓ Hands-on demo with ad-hoc commands and playbooks for Windows

Ansible Variables

- Introduction to Variables
- Types of Variables
- Define and reference variables in playbooks
- Hand-On Demonstrations on creating and using variables
- Ansible Facts

Thinknyx®

Introduction to Variables



Flexibility

Dynamic configuration

Environments , Hosts & Conditions

Introduction to Variables

Playbook

Application

Allows for

Easier management

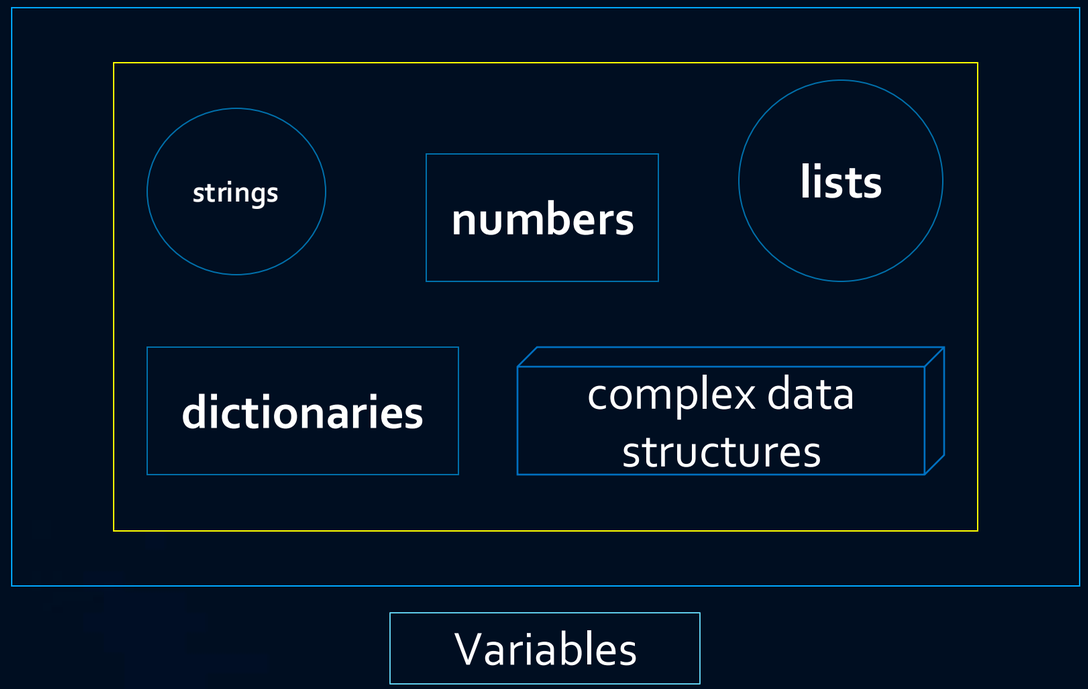
Improves Adaptability

Parameters

- Usernames
- Application
- Configuration file names
- Network port numbers

Introduction to Variables

- Variable is like a label that holds a specific value
- Variable is like a name tag for a value, allowing quick reference later



Types of Variables

:- Inventory Variables

```
[thinknyx_servers]

ansible-managedone ansible_port=22 ansible_host=192.168.1.10 http_port=80

ansible-managedtwo http_port=80
```

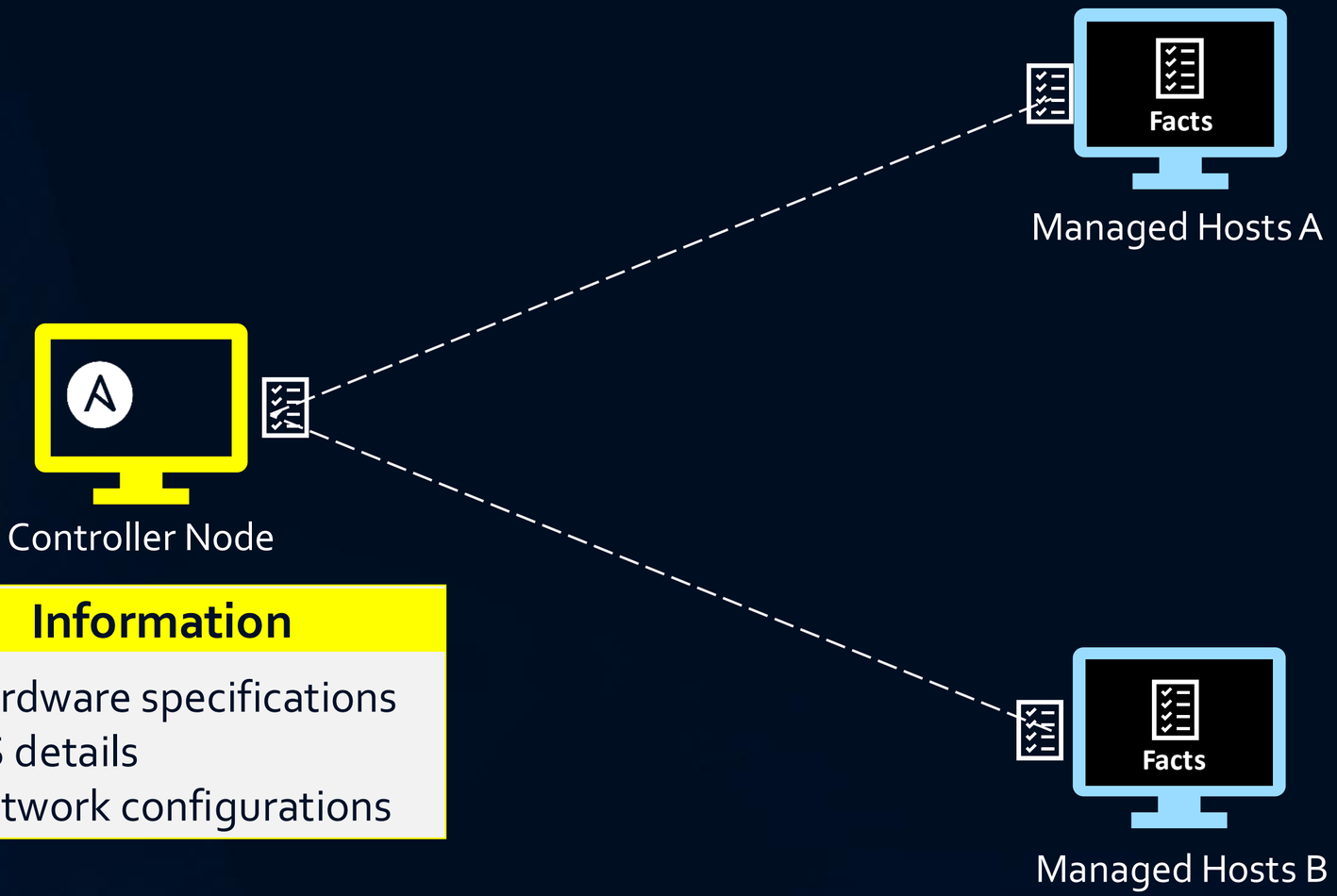
Types of Variables

:- Playbook Variables

webserver.yaml

```
---
- hosts: webservers
-   become: yes

 vars:
    apache_package: apache2
    apache_service: apache2

  tasks:
    - name: Include external variables
      include_vars: external_vars.yml

    - name: Ensure Apache is installed
      apt:
        name: "{{ apache_package }}"
        state: present
        update_cache: yes

    - name: Start and enable Apache service
      service:
        name: "{{ apache_service }}"
        state: started
        enabled: yes
```

Ansible Facts



Information

- Hardware specifications
- OS details
- Network configurations

Types of Variables

:- Role Variables

User-defined

```
#roles/myrole/vars/main.yml
my_custom_variable: "Hello, Ansible!"
```

Built-in

```
# roles/myrole/tasks/main.yml
- name: Print the current host
  debug:
    msg: "The current host is {{ ansible_hostname }}"
```

Types of Variables

:- Global Variables

ansible.cfg

```
[defaults]
remote_user = admin
private_key_file = ~/keys/private_key.pem
```

Demo

Creating and using variables

Conditionals, Loops and Handlers

- Use of Conditionals
- Loops and Handlers in playbooks
- Create dynamic and efficient automation workflows
- Demonstrations of example playbooks

Conditionals

- ✓ Control the flow of execution based on certain conditions
- ✓ Allows to execute tasks selectively based on predefined criteria

System properties

Operating system

Files or directories

Value of variables

Previous tasks

Conditionals

- ✓ Most used conditional statement is **"when"**

- ✓ **'When'** condition is typically evaluated using equality operators such as "=="

Conditionals

Intricate conditions

- Allows you to execute tasks
- Combination of conditions
- Used in conjunction with loops
- Enables to apply conditional logic iteratively

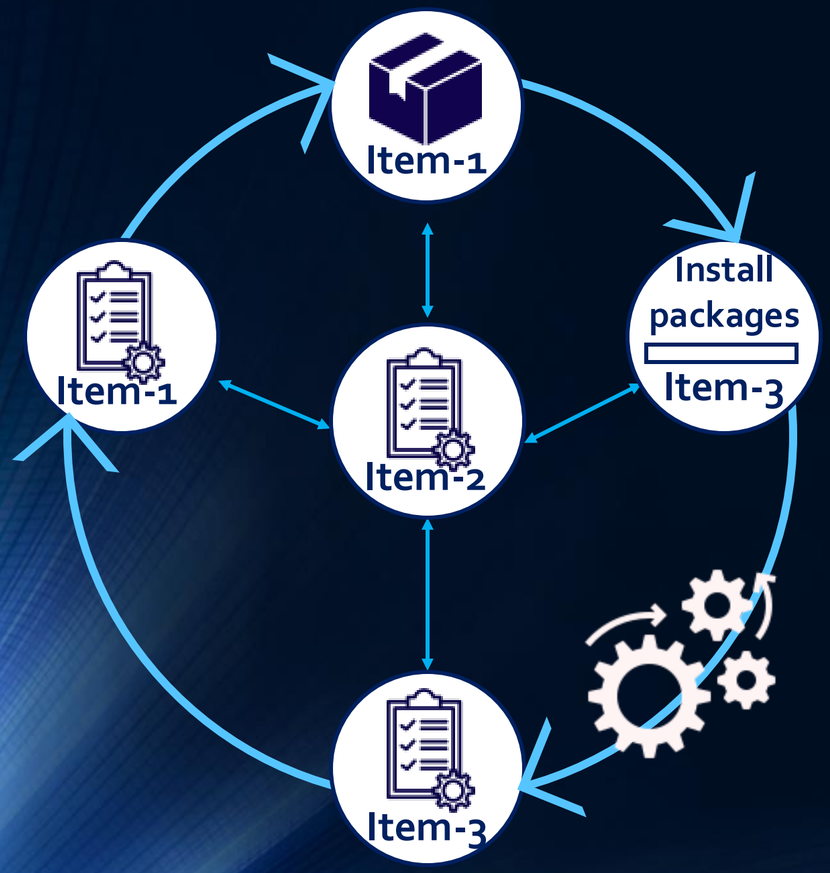
Example of Conditionals

Facts1.yml

```
---
- name: Install packages based on OS
  hosts: all
  become: yes
  tasks:
   - name: Install Apache on Ubuntu 22.04 or higher
     ansible.builtin.package:
       name: apache2
       state: present
     when: ansible_facts['distribution'] == "Ubuntu" and ansible_facts['distribution_version']|int >= 22
```

Loops

- ✓ Loops in Ansible allow you to iterate over a list of items and perform tasks multiple times



Ansible Supports

with_items	*loop*	*loop_control*

Loops

Facts1.yml

```
---
- name: Add multiple users
  hosts: managed-node02
  become: yes
  vars:
    usernames:
      - bob
      - john
      - alice
      - jane
    tasks:
    - name: Add users
      ansible.builtin.user:
        name: "{{ item }}"
        state: present
      loop: "{{ usernames }}"
```

Using loops

- Concise
- Easy to maintain
- Adaptable
- Useful when working with multiple hosts
- Managing configurations
- Performing repetitive operations

Handlers

- ✓ Handlers can be used to automate and restart the web server to apply the changes

Web Application configured

Handlers

- ✓ Handlers can be used to automate and restart the web server to apply the changes

Web Application configured

Handlers

- ✓ Special tasks that are distinct from regular tasks in playbooks

- ✓ Contributing to the idempotency and efficiency of playbook executions

Handlers

my_handlers.yml

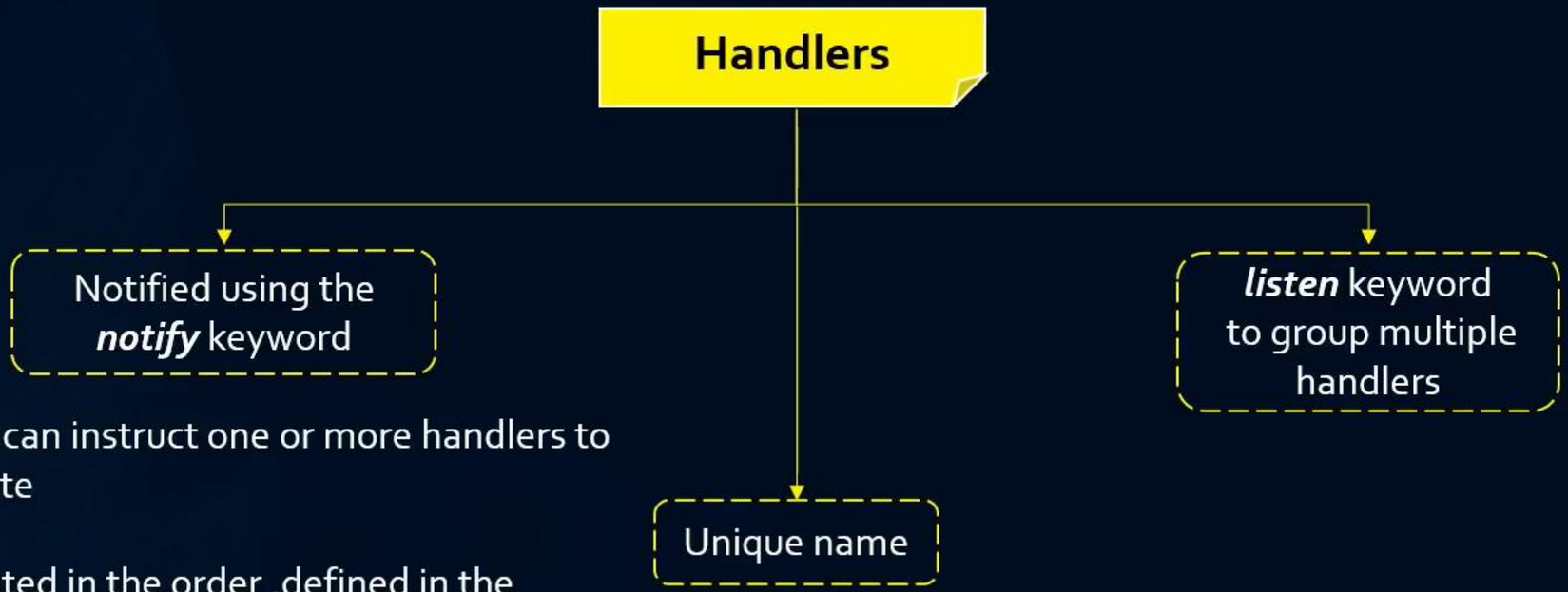
```
  notify: Restart Apache

handlers:
    - name: Restart Apache
      ansible.buitin. service:
        name: apache2
        state: restarted
```

Handlers

my_handlers.yml

```
---
- hosts: thinknyx_servers
  become: yes

  tasks:
    - name: Install Apache web server
      ansible.builtin.apt:
        name: apache2
        state: present

    - name: Update Apache configuration
      ansible.builtin.copy:
        src: files/apache2.conf
        dest: /etc/apache2/apache2.conf
      notify: Restart Apache

  handlers:
    - name: Restart Apache
      ansible.buitin. service:
        name: apache2
        state: restarted
```

Handlers



- Tasks can instruct one or more handlers to execute
- Executed in the order ,defined in the playbook
- Not in the order listed in the notify statement

Handlers

my_handlers.yml

```
handlers:

  - name: Restart Apache
    ansible.builtin.service:
      name: apache2
      state: restarted
    listen: "restart web services"

  - name: Restart MySQL
    ansible.builtin.service:
      name: mysql
      state: restarted
    listen: "restart web services"
```

Handlers

my_handlers.yml

```
handlers:

  - name: Restart Apache
    ansible.builtin.service:
      name: apache2
      state: restarted
    listen: "restart web services"

  - name: Restart MySQL
    ansible.builtin.service:
      name: mysql
      state: restarted
    listen: "restart web services"
```

Handlers

- ✓ Avoid placing variables in the name of handlers
- ✓ Cause playbook failures if the variables are not available

my_handlers.yml

```
tasks:

  - name: Set Apache Service Name
    set_fact:
      apache_service_name: "{{ ansible_distribution ==
'Ubuntu' | ternary('apache2', 'httpd') }}"

  - name: Configure Apache
    ansible.builtin.template:
      src: apache.conf.j2
      dest: /etc/{{ apache_service_name }}/apache2.conf
    notify: Restart Apache

handlers:
  - name: Restart Apache
    ansible.builtin.service:
      name: "{{ apache_service_name }}"
      state: restarted
```

Summary

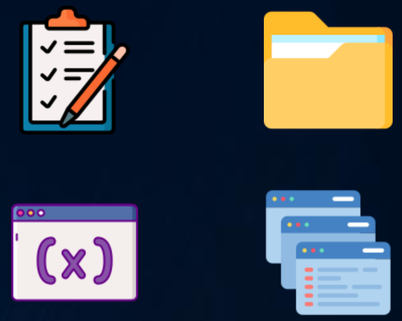
- ✓ Conditionals, particularly with the "when" statement
- ✓ Loops to iterate over a list of items
- ✓ Handlers to manage tasks to restart services after configuration changes
- ✓ Demonstrations used Loops and Conditionals to install Docker
- ✓ A demo on combination of conditionals, loops and handlers

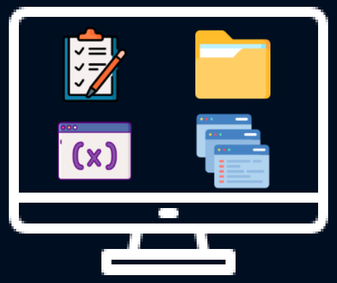
Ansible Roles

- Introduction to roles
- Standard directory structure of a role
- Creating and using roles
- Dependencies in roles
- Demonstrate how role defaults
- Use cases of roles

Introduction to Roles

- ✓ Roles are a way to organize and reuse Ansible playbooks





Structured format

- ✓ Allows for easier management of complex configurations

Introduction to Roles

- ✓ Benefits of Using Roles

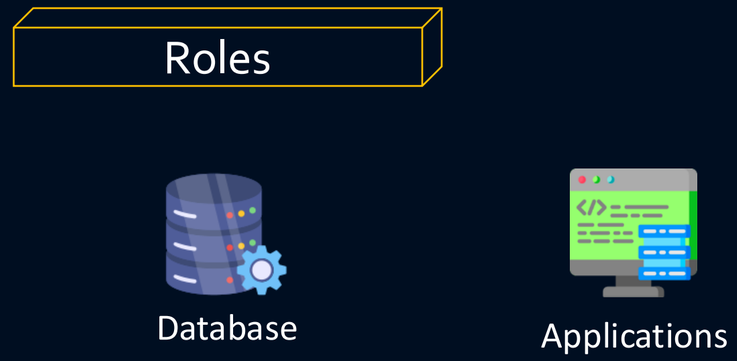
Reusability

Maintainability

Scalability

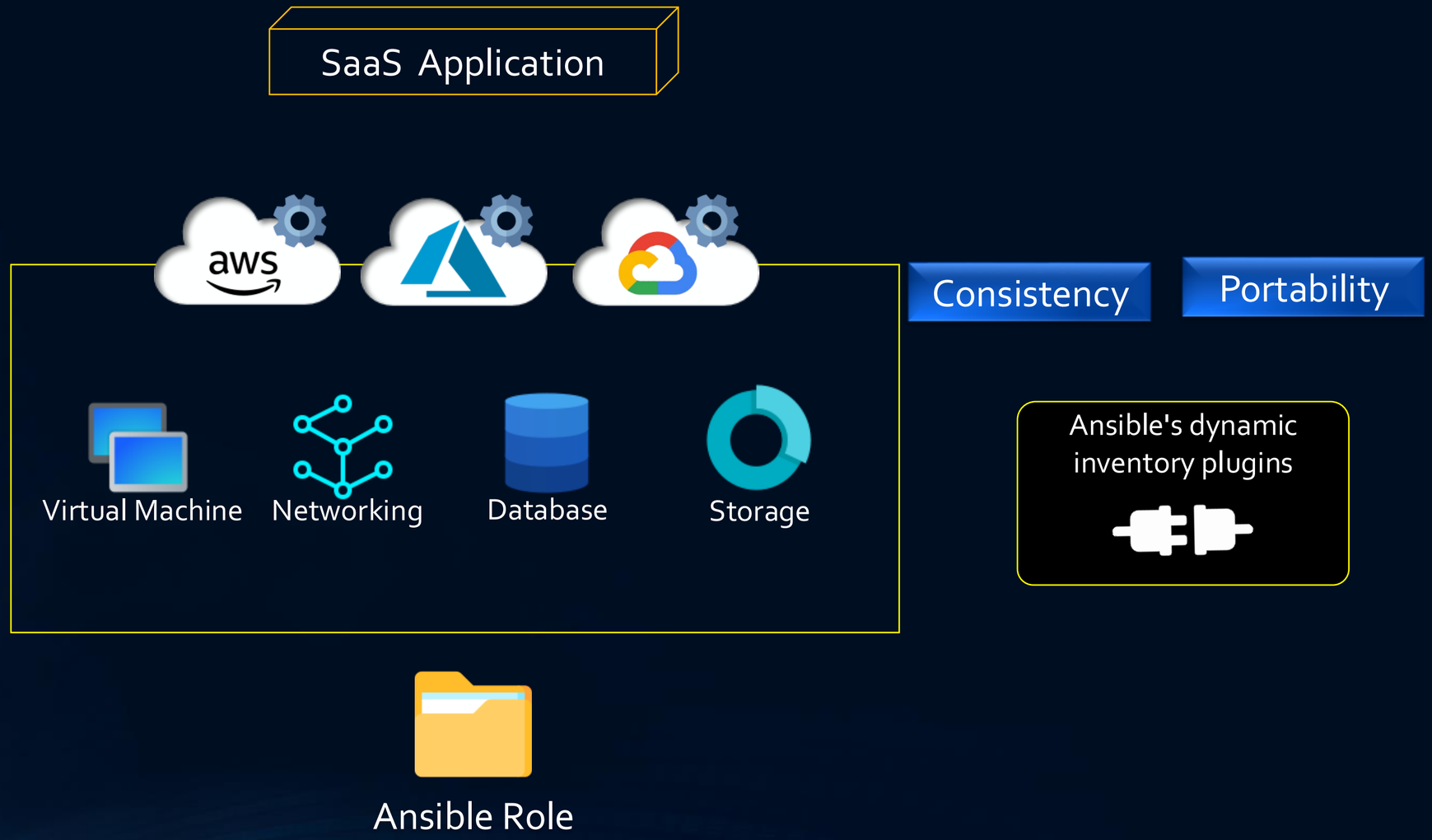
Introduction to Roles

- ✓ Roles are useful in larger projects

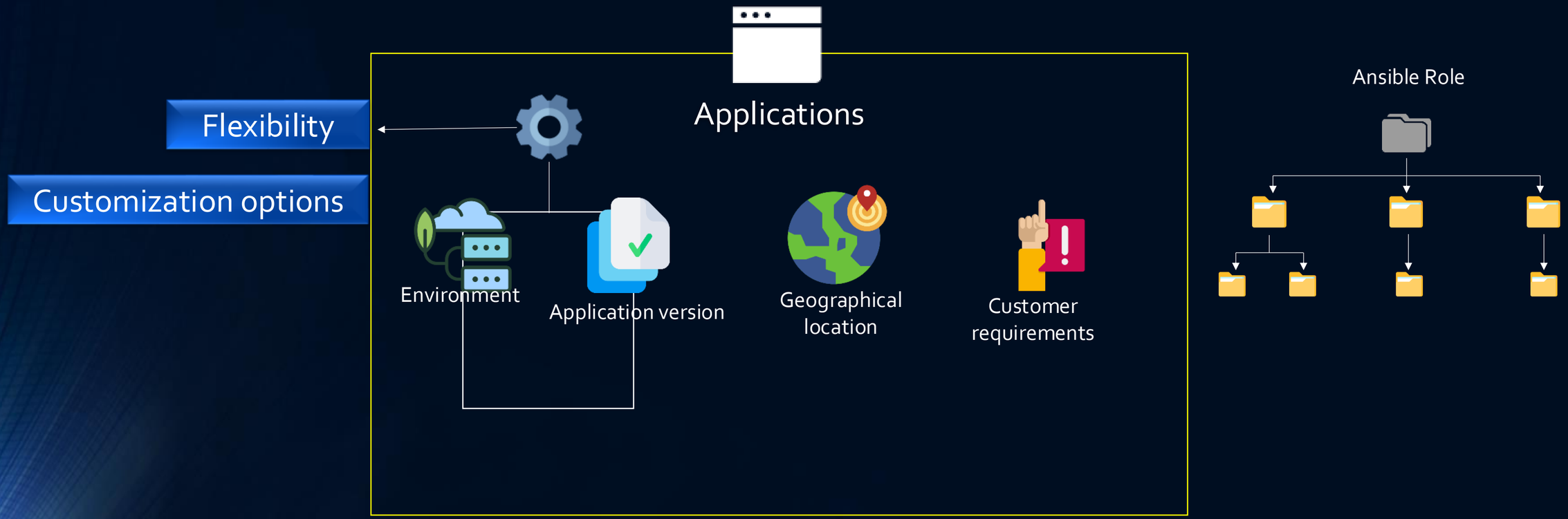


- ✓ Maintaining clarity and efficiency

Use Case: Multi-cloud infrastructure provisioning

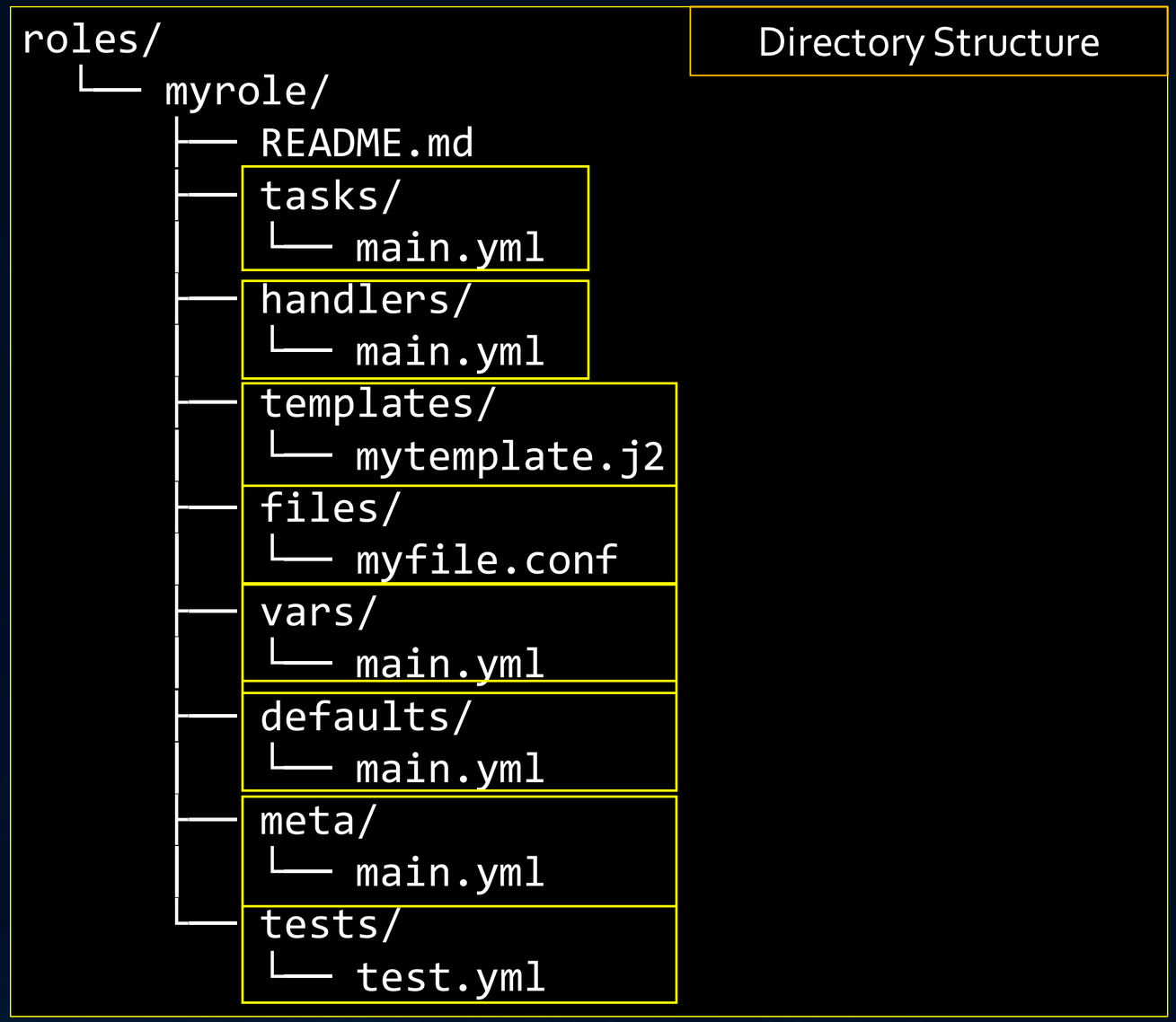


Use Case: Application deployment to multiple environments



8.2

Directory Structure of Roles



8.3

Creating and using Roles

- ✓ Ansible looks for roles in the following places by default

roles/ directory relative to the playbook file

roles_path in **ansible.cfg** file is located at **~/.ansible/roles**

/usr/share/ansible/roles

/etc/ansible/roles

- ✓ Ansible also looks for roles in the following places:

In collections, if they are used

Fully qualified role path in the playbook

Creating and using Roles

- ✓ roles_path = /etc/ansible/roles

ansible.cfg

```
# (string) Sets the login user for the target machines
# When blank it uses the connection plugin's default, normally the user currently executing Ansible
.
;remote_user=

# (pathspec) Colon separated paths in which Ansible will search for Roles.
;roles_path={{ ANSIBLE_HOME ~ "/roles:/usr/share/ansible/roles:/etc/ansible/roles" }}

# (string) Set the main callback used to display Ansible output. You can only have one at a time.
# You can have many other callbacks, but just one can be in charge of stdout.
# See :ref:`callback_plugins` for a list of available options.
;stdout_callback=default

# (string) Set the default strategy used for plays.
;strategy=linear
```

- ✓ Syntax

```
ansible-galaxy init <role_name>
```

- ✓ For example

```
ansible-galaxy init webserver_role
```

Creating and using Roles

```
ubuntu@controller-node01:~/webserver_role$ tree
.
├── README.md
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml

9 directories, 8 files
```

Creating and using Roles

```
# roles/web_server/tasks/ubuntu_apache_install.yml
- name: Install Apache web server on Ubuntu
  ansible.builtin.apt:
    name: "apache2"
    state: present


# roles/web_server/tasks/centos_httpd_install.yml
- name: Install Apache web server on CentOS
  ansible.builtin.yum:
    name: "httpd"
    state: present


# roles/web_server/tasks/main.yml
-  name: Install the appropriate web server based on the OS
     import_tasks: "{{ item.file }}"
     when: ansible_facts['distribution'] == item.distribution
     with_items:
       - { distribution: 'Ubuntu', file: 'ubuntu_apache_install.yml' }
       - { distribution: 'CentOS', file: 'centos_httpd_install.yml' }
```

Creating and using Roles

- ✓ Use the roles option
- ✓ Use **include_role** option to reuse roles dynamically
- ✓ Use **import_role** option to reuse roles statically
- ✓ Use it as a dependency of another role

Creating and using Roles

rolesplaybook.yaml

```
---
- name: My playbook with a role
  hosts: all
  roles:
    - webserver_role
```

Creating and using Roles

Feature	import_role	include_role
Inclusion Mechanism	Static inclusion	Dynamic inclusion
Execution Timing	Tasks are incorporated at playbook parsing	Tasks are included and executed at runtime
Execution Sequence	Tasks are part of the playbook's execution sequence	Included based on runtime conditions
Dependency Management	Dependencies must be managed separately	Dependencies handled dynamically
Conditional Inclusion	Not supported; role is included at parsing	Supports conditional inclusion with when statements

Demo

Creating & Using Roles

Demo

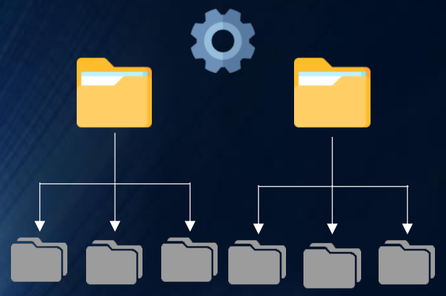
Using include_role and import_role to reuse roles

8.6

Role Dependencies

- ✓ Handling dependencies required a role to execute successfully
- ✓ Dependencies can Include
  - Other roles
  - Ansible collections
  - External packages
  - Other necessary resources for role's tasks

Role Dependencies:



meta/main.yml

```
dependencies:
  - { role: webrole }
```

Demo

Creating & Using Role Dependencies

Demo

Role variables and Defaults

Summary

- ✓ Introduction to Ansible roles
- ✓ Directory structure of roles
- ✓ Creating and using roles
- ✓ Demonstrations
    - Practical role development
    - Dependency management
    - Role variables
    - Role defaults

Ansible Collections

- What are Collections
- Collection index
- Demonstrations
  - Installing and managing a collection
  - Creating your own collection
  - Use collection in playbook

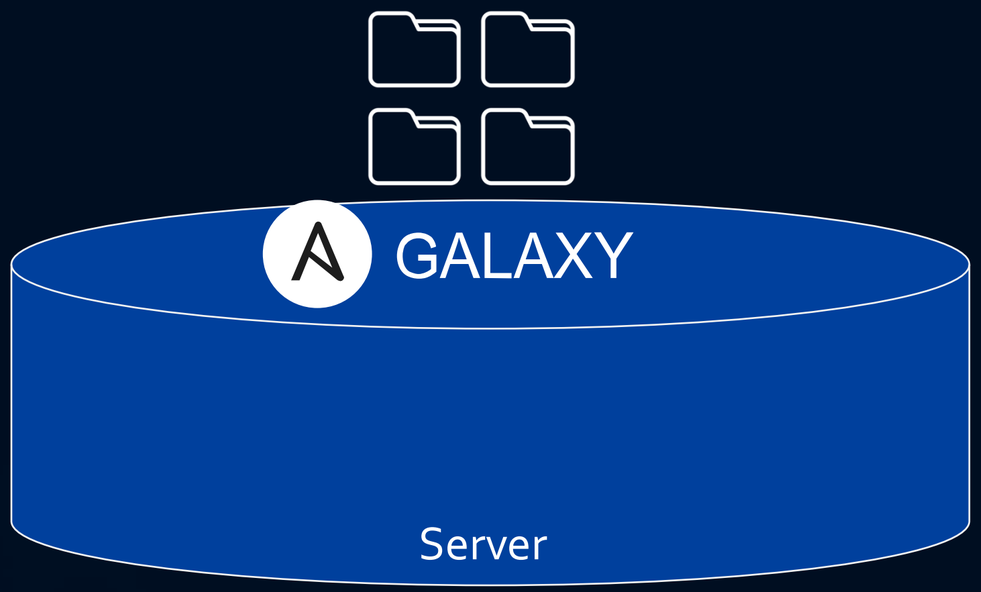
Thinknyx®

Introduction to Ansible Collections

Data Structure like Roles

Introduction to Ansible Collections

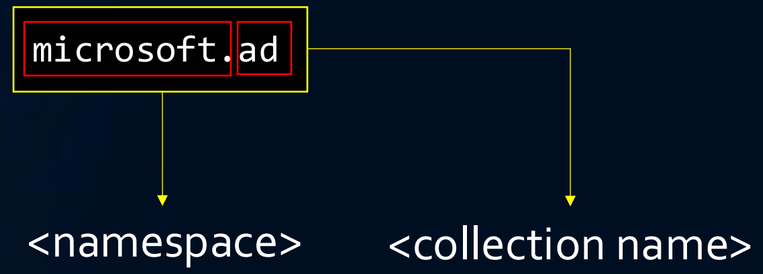
Ansible collection



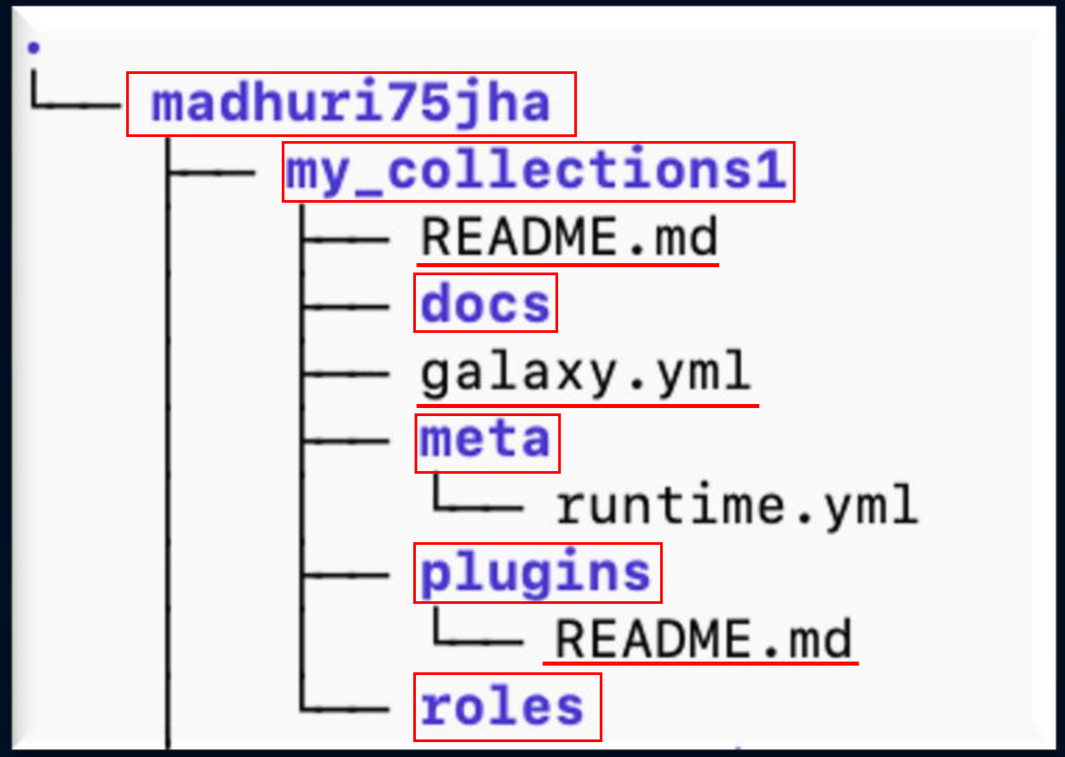
ansible-galaxy collection <command option>

Introduction to Ansible Collections

- ✓ Collection belongs to a namespace which must be unique
- ✓ Name format for fully Qualified Collection **<namespace>.<collection>**
- ✓ For Example



Introduction to Ansible Collections



Introduction to Ansible Collections

- ✓ Ansible picks up installed collections by default from

`/home/ubuntu/.ansible/collections` `/usr/share/ansible/collections`

`ansible.cfg`

`collections/ansible_collections/`

- ✓ Ansible collections are installed by default

`~/.ansible/collections/ansible_collections` `-p option`

- ✓ Parent directory

`ansible_collections`

Demo

Encrypting a single variable with Ansible vault

Summary

ansible-vault

- ✓ A tool for securing sensitive data
- ✓ Allows us to encrypt and decrypt data such as passwords, API keys, tokens
- ✓ Can encrypt individual variables or entire files such as variable files, template files
- ✓ Demonstrations, we practiced how to encrypt a single variable and use it in a playbook
- ✓ To create an encrypted variable file containing database information and integrated it into the playbook
- ✓ To manage and use multiple vault passwords by using separate files and the --vault-password-file option

Ansible Galaxy

- Overview on Galaxy
- Galaxy documentation
  - Features
  - Search capabilities
  - Usage guidelines
- Publish roles to Ansible Galaxy
  - Using command line
  - Galaxy website

Demo

Writing a multi-task Playbook

Overview of Ansible Galaxy

Ansible Galaxy

- ✓ Free to use
- ✓ Content is community developed and shared

With Galaxy you can

- Find Collections and Standalone
- Share Your Work
- Search and Browse
- Read Reviews

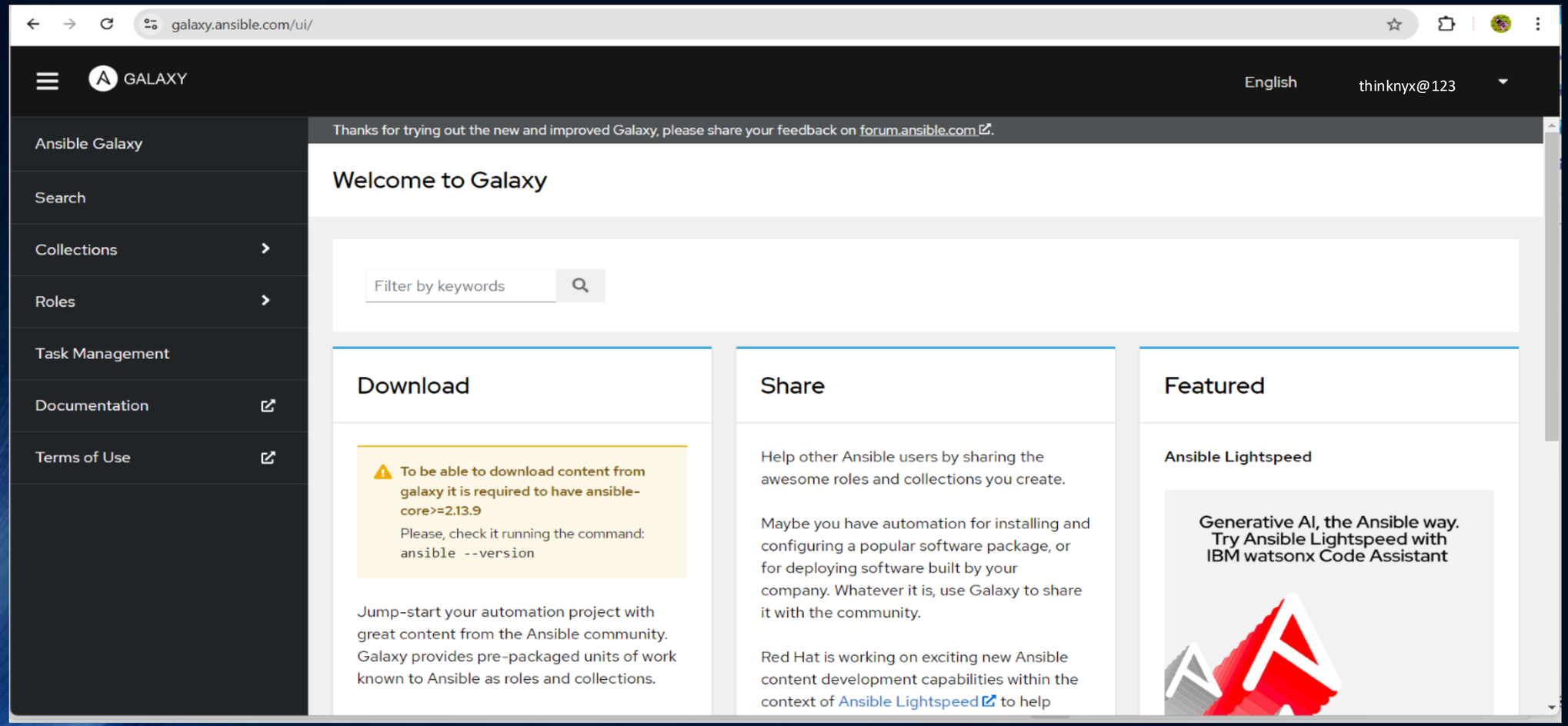
Roles vs. Collections

- ✓ Two types of content

Roles	Collection
These are individual units that handle specific tasks. Think of them as tools designed to perform one job, like installing a web server	These are larger packages that include multiple roles, playbooks, and other Ansible content. It's like a complete toolkit for more complex automation tasks
Roles in Ansible Galaxy are self-contained units that automate specific tasks, like setting up a web server or configuring a database.	Collections, on the other hand, are comprehensive packages of automation that may include multiple playbooks, roles, modules, and plugin

Overview of Ansible Galaxy

- ✓ **Ansible Galaxy** website is primarily for browsing collections and roles
- ✓ *ansible-galaxy* **command-line tool** is used to download, install, upload, or publish collections and roles



Overview of Ansible Galaxy

	Command Syntax	Description
Collections	ansible-galaxy collection install <name>	Installs a collection by name
	ansible-galaxy collection list	Lists installed collections
	ansible-galaxy collection build	Builds an Ansible collection artifact that can be published to Ansible Galaxy
	ansible-galaxy collection publish <collection_path>	Publishes a collection artifact to Ansible Galaxy with the path to the collection tarball
	ansible-galaxy collection init <name>	Creates a new collection structure for development

Overview of Ansible Galaxy

	Command Syntax	Description
Roles	ansible-galaxy role install <name>	Installs a role by name from Ansible Galaxy
	ansible-galaxy role list	Lists installed roles
	ansible-galaxy role remove <name>	Removes an installed role by name
	ansible-galaxy role info <name>	Displays more details about a specific role
	ansible-galaxy role search <name>	Searches for a role by name in Ansible Galaxy
	ansible-galaxy role import <github_user> <github_repo>	Imports a role into a galaxy server
	ansible-galaxy collection init <name>	Creates a new role structure for development

Use Cases

Role Sharing and Reusability

Simplified Role Management

Standardization

Rapid Prototyping

Learning and Documentation

Integration with CI/CD Pipelines

Collaboration and Sharing

Demo

Navigating Ansible Galaxy Website

Demo

Publishing Roles to Ansible Galaxy

Summary

- ✓ Ansible Galaxy
- ✓ Demonstrations
  - Navigating the Galaxy website
  - Documentation
  - Search for collections and roles
  - Import role to galaxy using website and the command line

Ansible Vaults

- Overview of Ansible Vault
- Real-time example of use cases
- Hands on demonstration
  - Encrypt individual variables
  - Encrypt variable files
  - View, edit and decrypt an encrypted file

Introduction to Vaults

Vault

Encrypt

Decrypt

Manage Data

- ✓ You can secure data such as passwords, API keys, and other sensitive information

Real-Time Example with Ansible Vault

- ✓ Database connection strings
- ✓ API tokens
- ✓ Sensitive information
  - Username
  - Passwords

Production Environment

Real-Time Example with Ansible Vault

Real- time scenario

```
-   name: Simple select query to acme db
  community.mysql.mysql_query:
    login_db: orders
    login_user: SecretUser
    login_password: SuperSecretPassword
    query: SELECT * FROM orders
    login_unix_socket: /run/mysqld/mysqld.sock
```

Challenge

```
-   name: Simple select query to acme db
  community.mysql.mysql_query:
    login_db: orders
    login_user: SecretUser
    login_password: SuperSecretPassword
    query: SELECT * FROM orders
    login_unix_socket: /run/mysqld/mysqld.sock
```

Real-Time Example with Ansible Vault

Solution

Seprate.yml

```
-   name: Simple select query to acme db
  community.mysql.mysql_query:
    login_db: orders
    login_user: SecretUser
    login_password: SuperSecretPassword
    query: SELECT * FROM orders
    login_unix_socket: /run/mysqld/mysqld.sock
```

Passwords

- ✓ Reference the encrypted file and the variables in your playbook
- ✓ Encrypted data will be decrypted and used when you provide the vault password

Workflow

Step 1: Create a YAML file

Step 2: Encrypt the variables with a secure vault password

Step 3: Include the encrypted variables file using **'vars_files'**

Step 4: Reference the variables

Step 5: Run the playbook with the '--ask-vault-pass' flag

Step 6: Decrypt the playbook

Demo

Encrypting a single variable with Ansible vault

Demo

Encrypting Files with Ansible vault

Demo

Decrypting encrypted variables and files

Summary

ansible-vault

- ✓ A tool for securing sensitive data
- ✓ Allows us to encrypt and decrypt data such as passwords, API keys, tokens
- ✓ Can encrypt individual variables or entire files such as variable files, template files
- ✓ Demonstrations, we practiced how to encrypt a single variable and use it in a playbook
- ✓ To create an encrypted variable file containing database information and integrated it into the playbook
- ✓ To manage and use multiple vault passwords by using separate files and the --vault-password-file option

Ansible Troubleshooting

- Essential troubleshooting tips for playbooks
- Hands-on Demonstrations
  - Syntax Checking
  - Handle Missing Module Parameters
  - Ansible-lint Utility
  - Importance of register keyword and debug module
  - Run playbooks interactively with --step option
  - Isolate and execute specific tasks using tags

Thinknyx®

Troubleshooting Tips

- ✓ Common Ansible issues and structured troubleshooting approaches
- ✓ Step-by-step strategy and best practices to streamline Ansible troubleshooting

- ❑ **Step One: Start with the Documentation**
  - ✓ Always begin by checking the official Ansible documentation
  - ✓ Offers a wealth of information, including
    - Common troubleshooting tips
    - Usage examples
    - Clarifications on module parameters
  - ✓ Resolve syntax and parameter issues by reviewing official documentation

Troubleshooting Tips

- **Step Two: Check Connectivity**
  - ✓ Check connection first when tasks fail without clear error details
  - ✓ Review inventory, confirm target host is running, and validate firewall and authentication settings (SSH keys, WinRM, etc)
  - ✓ Run a ping test to verify connectivity between Ansible controller and target node
- **Step Three: Validate your Syntax**
  - ✓ Use the --syntax-check & --check options for common syntax errors or misconfigurations before executing the playbook

Troubleshooting Tips

- **Troubleshooting with Verbose Mode**
  - ✓ Use the -v flag to get detailed error messages during playbook execution
  - ✓ Use -vvv or -vvvv for detailed debug-level output

Levels	Description	Levels
0	Normal	No Parameter
1	Verbose	-v
2	More Verbose	-vv
3	Debug	-vvv
4	Connection Debug	-vvvv

  - ✓ For better visibility during playbook execution, make use of the register keyword to capture task outputs

Troubleshooting Tips

- Key Initial Troubleshooting Steps

Start by checking the official Ansible documentation

Confirm connectivity between your controller and target nodes

Verify your inventory file for accuracy

Run a syntax check to catch early errors

Use verbosity flags for more detailed output

Apply the register keyword and debug statements

Summary

- ✓ Common Ansible issues and troubleshooting techniques
- ✓ Checking documentation, verifying connectivity, and using syntax checks
- ✓ Using verbosity levels, register, and debug

Conclusion

Thinknyx®

You Trust, We Deliver!

Automation with Ansible

Hands-on

ANSIBLE

Ansible Environment Setup

Automating Task

Managing Configuration

Orchestrating Deployments

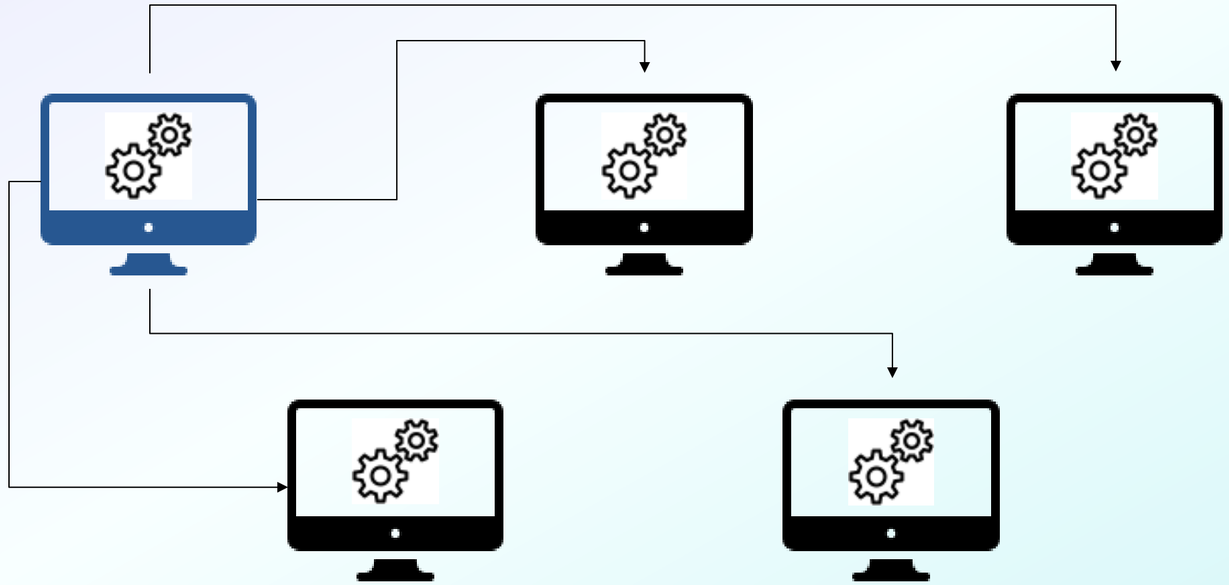
ANSIBLE

Ansible DevOps Streamlining

Windows Managed Nodes

Ansible Vault

Web-Interfaces with AWX



Playbooks

Roles

Collections

Follow us on:

- @thinknyx
- @thinknyx
- @thinknyx-technologies
- @thinknyx
- @thinknyx-technologies