Ansible Complete Training Guide

From Novice to Advanced

Comprehensive DevOps Training Material

Configuration Management | Automation | Infrastructure as Code

Table of Contents

Comprehensive Ansible Training Guide: From Novice to Advanced

Table of Contents

1. Configuration Management Fundamentals

1.1 What is Configuration Management?

Configuration Management (CM) is a systems engineering process that establishes and maintains the consistency of a system's performance, functional attributes, and physical characteristics throughout its lifecycle. In IT infrastructure, configuration management refers to the process of systematically handling changes to a system in a way that maintains integrity over time.

Key Concepts:

- **Infrastructure as Code (IaC)**: Treating infrastructure configuration as software code that can be versioned, tested, and deployed
- **Desired State Configuration**: Defining what state systems should be in rather than how to achieve that state
- **Consistency**: Ensuring all systems are configured identically according to specifications
- **Automation**: Reducing manual intervention in system configuration and management
- **Version Control**: Tracking changes to configurations over time
- **Auditability**: Maintaining detailed records of what configurations were applied and when

1.2 Why Configuration Management is Critical

Business Challenges Without CM:

- **Configuration Drift**: Over time, servers diverge from their intended configurations due to manual changes, updates, and patches
- **Manual Errors**: Human mistakes during configuration lead to system failures and security vulnerabilities
- **Lack of Documentation**: Without proper documentation, nobody knows the exact state of systems
- **Slow Deployment**: Manual configuration of servers is time-consuming and doesn't scale
- **Inconsistent Environments**: Dev, test, and production environments differ, causing "works on my machine" issues
- **Compliance Issues**: Difficulty proving compliance with security standards and regulations
- **Disaster Recovery**: Slow recovery times because systems can't be quickly rebuilt

Benefits of Configuration Management:

- **Scalability**: Configure thousands of servers with a single command
- **Consistency**: All servers maintain identical configurations
- **Rapid Deployment**: Deploy applications and infrastructure in minutes instead of days
- **Reduced Downtime**: Quickly restore systems to known good states
- **Version Control**: Track all changes and roll back if needed
- **Documentation**: Configuration files serve as living documentation
- **Cost Reduction**: Less time spent on manual tasks means lower operational costs
- **Compliance**: Automated compliance checking and reporting
- **Security**: Consistent security policies applied across all systems

1.3 Configuration Management vs Traditional System Administration

Traditional Approach	Configuration Management
Manual SSH to each server	**Single command for all servers**
Bash scripts that may not be idempotent	Idempotent operations
No version control	Git-tracked configurations
Difficult to test changes	Test in dev before production
Configuration drift common	Consistent state enforcement
Time-consuming rollbacks	Quick rollback to previous versions
Limited documentation	Self-documenting code

1.4 Popular Configuration Management Tools

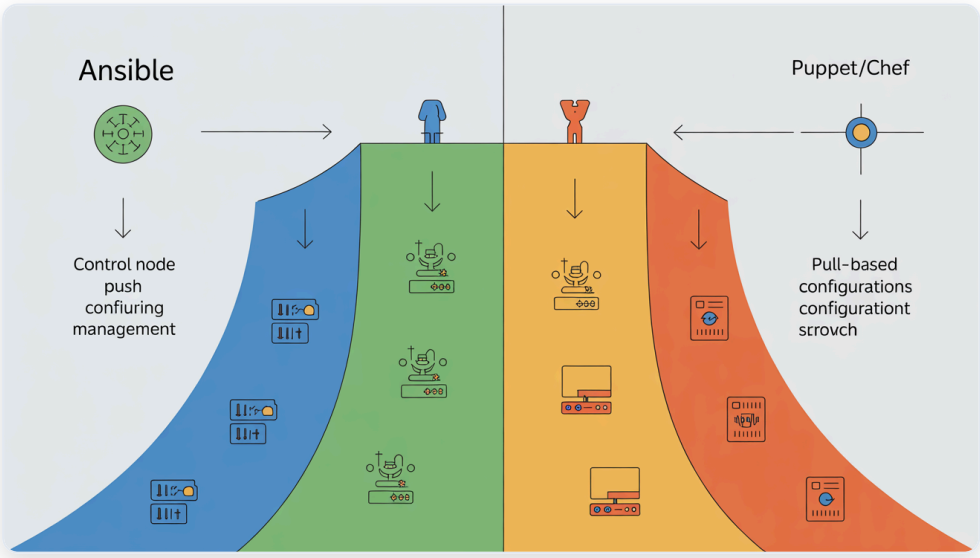


Figure: 1.4 Popular Configuration Management Tools

Ansible

- Agentless (uses SSH)
- Python-based

- Push model
- YAML configuration
- Steep learning curve: Low

Puppet

- Agent-based
- Ruby DSL
- Pull model
- Declarative language
- Steep learning curve: High

Chef

- Agent-based
- Ruby DSL
- Pull model
- Procedural code
- Steep learning curve: High

Salt

- Agent-based (can be agentless)
- Python-based
- Push/Pull hybrid
- YAML configuration
- Steep learning curve: Medium

1.5 Real-World Scenario: The Need for Configuration Management

Scenario: A growing e-commerce company has 200 web servers running their application. They need to:

- Update SSL certificates on all servers (critical security update)
- Install a security patch for a vulnerability
- Update application configuration files
- Ensure all servers have identical configurations

Without CM:

- Manually SSH into each of 200 servers
- Run commands on each server individually
- High risk of missing servers or making errors
- Time required: 2-3 days
- Error rate: High

With Ansible CM:

```
- hosts: webservers
  tasks:
    - name: Update SSL certificates
      copy:
        src: /path/to/new/cert.pem
        dest: /etc/ssl/certs/cert.pem

    - name: Install security patch
      yum:
        name: openssl
        state: latest

    - name: Update application config
      template:
        src: app.conf.j2
        dest: /etc/app/app.conf
```

- Time required: 10-15 minutes
- Error rate: Minimal
- Auditable: Complete log of changes

- Repeatable: Can run again safely (idempotent)

1.6 Hands-On Lab Exercise: Understanding the Problem

Objective: Experience the challenges of manual configuration management

Steps:

- Set up three virtual machines or containers
- Manually install Apache web server on each
- Create a custom index.html on each server
- Configure firewall rules on each
- Now update the index.html on all three servers
- Document how long this takes and what challenges you face

Reflection Questions:

- How long did it take to configure three servers?
- What if you had 100 servers?
- How would you ensure all configurations are identical?
- How would you track what changes were made and when?
- How quickly could you roll back a change?

---

2. Ansible Introduction, History, and Architecture

2.1 What is Ansible?

Ansible is an open-source automation platform used for IT tasks such as configuration management, application deployment, intra-service orchestration, and provisioning. It provides a simple, powerful, and agentless way to automate tasks across your IT infrastructure.

Core Philosophy:

- **Simplicity**: Easy to learn, read, and maintain
- **Powerful**: Can orchestrate complex multi-tier deployments
- **Agentless**: No software to install on managed nodes
- **Efficiency**: Minimal moving parts means less that can break
- **Security**: SSH-based communication, no custom security infrastructure needed

2.2 Ansible History and Evolution

Timeline:

2012: Ansible project created by Michael DeHaan

- Initial release focused on simplicity and ease of use

- Used SSH for communication, no agents required
- YAML-based configuration files

2013: Rapid adoption in DevOps community

- Growing module library
- Community contributions accelerate
- Ansible Tower (commercial product) announced

2015: Red Hat acquires Ansible Inc. for $150 million

- Integration with Red Hat Enterprise Linux
- Enterprise support and features added
- AWX (open-source Tower) project initiated

2017: Ansible 2.0 released

- Major performance improvements
- Enhanced Windows support
- Networking automation capabilities

2020: Ansible becomes part of Red Hat Ansible Automation Platform

- Enhanced security features
- Better cloud integrations (AWS, Azure, GCP)
- Ansible Content Collections introduced

2023-2024: Ansible 2.15+ era

- Python 3.9+ requirement
- Enhanced automation analytics
- Event-Driven Ansible introduced
- Expanded cloud-native support

Current State:

- One of the top 3 configuration management tools globally

- Over 50,000 GitHub stars
- Thousands of modules available
- Active community of contributors
- Enterprise adoption across Fortune 500 companies

2.3 Ansible Architecture and Components

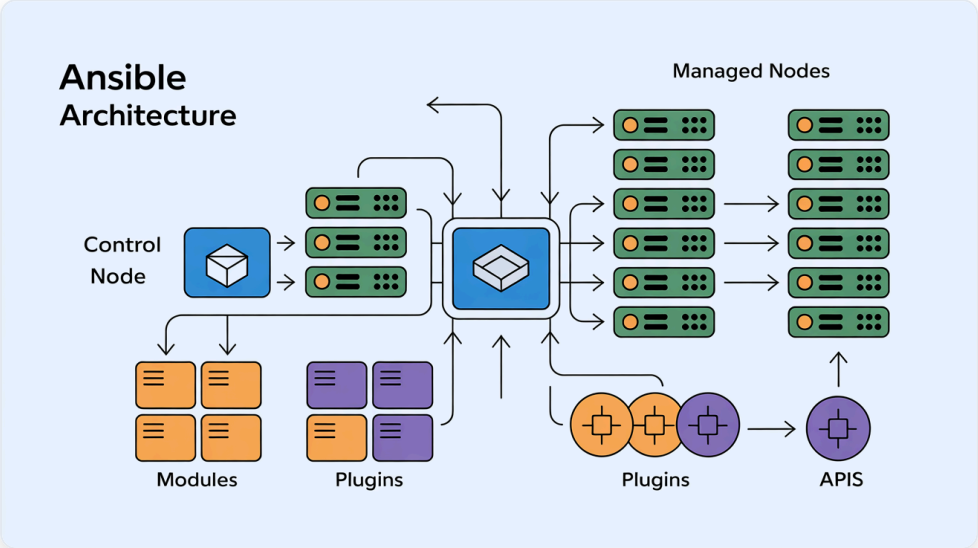
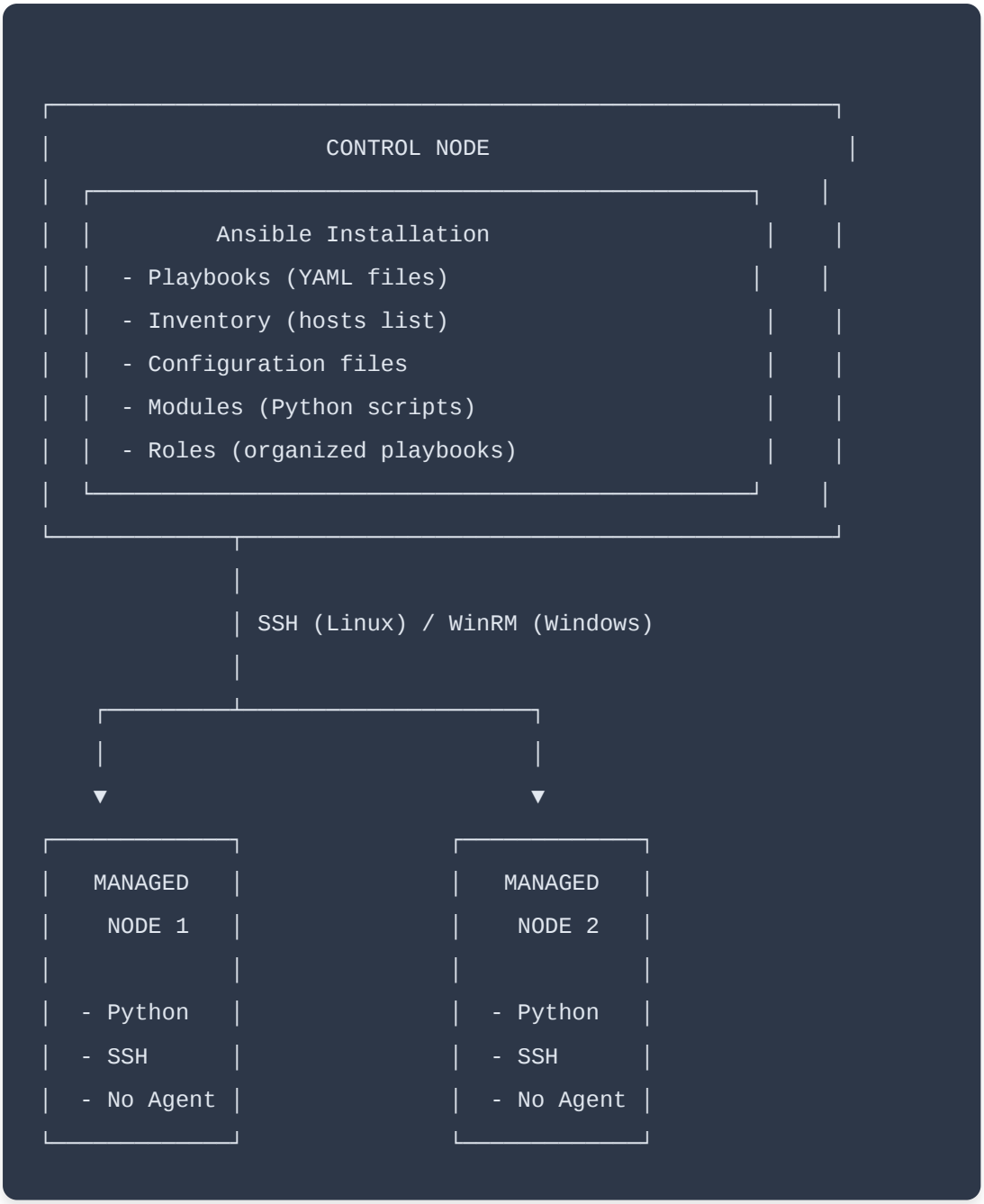


Figure: 2.3 Ansible Architecture and Components

Ansible uses a simple architecture that consists of a few key components. Understanding this architecture is fundamental to working effectively with Ansible.



2.3.1 Control Node

The control node is the machine where Ansible is installed and from which automation tasks are executed.

Requirements:

- Unix-like operating system (Linux, macOS, BSD)
- Python 3.9 or newer (for Ansible 2.15+)

- Cannot be Windows (Windows can be WSL-based)
- SSH client installed

Responsibilities:

- Store and execute playbooks
- Manage inventory of nodes
- Maintain configuration files
- Execute modules on managed nodes
- Collect and display results

Installation Options:

- Package manager (yum, apt, dnf)
- pip (Python package manager)
- From source

2.3.2 Managed Nodes

Managed nodes are the target systems that Ansible manages. These can be servers, network devices, cloud instances, or any system that Ansible can communicate with.

Requirements:

- Python 2.7 or Python 3.5+ installed
- SSH server running (for Linux/Unix)
- WinRM configured (for Windows)
- No Ansible agent required

Supported Systems:

- Linux distributions (RHEL, CentOS, Ubuntu, Debian, etc.)
- Windows Server (2012+)
- Network devices (Cisco, Juniper, Arista, etc.)

- Cloud platforms (AWS, Azure, GCP, etc.)
- Container platforms (Docker, Kubernetes)
- VMware and other virtualization platforms

2.3.3 Inventory

The inventory is a list of managed nodes organized in groups. It defines which hosts Ansible will target.

Types:

- **Static Inventory**: Simple text file listing hosts
- **Dynamic Inventory**: Script that queries external sources (AWS, Azure, etc.)

Example Static Inventory:

```
[webservers]
web1.example.com
web2.example.com
web3.example.com

[databases]
db1.example.com
db2.example.com

[production:children]
webservers
databases
```

2.3.4 Modules

Modules are the units of work in Ansible. They are small programs that perform specific tasks.

Characteristics:

- Written in Python (mostly)
- Can be written in any language
- Idempotent (run multiple times, same result)
- Return JSON output
- Over 3,000 built-in modules

Common Module Categories:

- System (user, group, service, etc.)
- Files (copy, file, template, etc.)
- Packaging (yum, apt, pip, etc.)
- Cloud (AWS, Azure, GCP modules)
- Network (Cisco, Juniper modules)
- Database (MySQL, PostgreSQL modules)

2.3.5 Playbooks

Playbooks are YAML files that define a series of tasks to be executed on managed nodes.

Characteristics:

- Human-readable YAML format
- Declarative (describe desired state)
- Can include multiple plays
- Support variables, conditionals, loops
- Reusable and shareable

2.3.6 Roles

Roles are a way to organize playbooks and make them reusable. They provide a framework for fully independent or interdependent collections of variables, tasks, files, templates, and modules.

Benefits:

- Code reusability
- Better organization
- Easy sharing via Ansible Galaxy
- Simplified complex playbooks

2.3.7 Ansible Galaxy

Ansible Galaxy is a hub for finding, downloading, and sharing Ansible roles and collections.

Features:

- Community-contributed roles
- Official Red Hat certified content
- Command-line tool for role management
- Role versioning and dependencies

2.4 How Ansible Works: Execution Flow

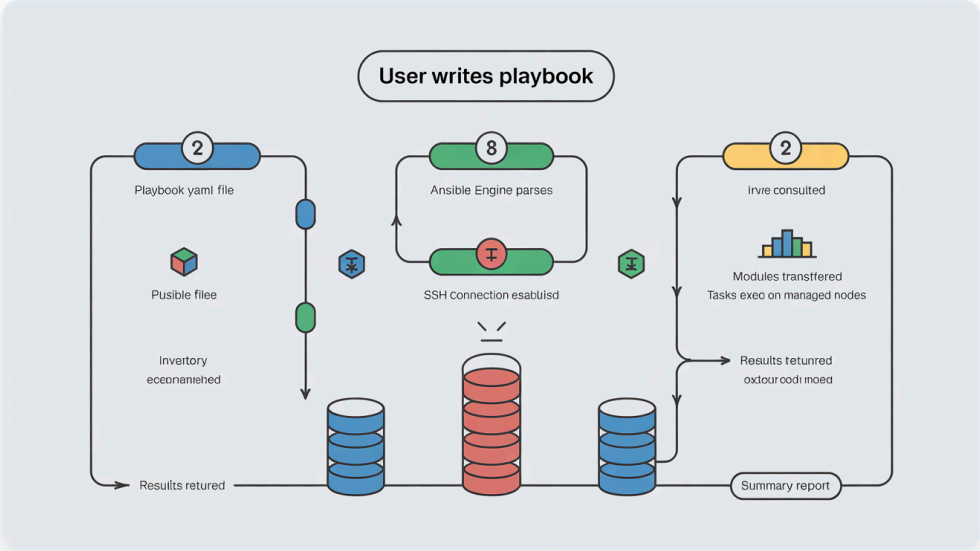


Figure: 2.4 How Ansible Works: Execution Flow

Step-by-Step Execution:

- **User runs ansible-playbook command**

  ```bash
  ansible-playbook -i inventory webserver-setup.yml
  ```

- **Ansible reads the inventory file**
- Identifies target hosts
- Groups hosts if specified
- Applies host patterns

- **Ansible parses the playbook**
- Validates YAML syntax
- Loads variables
- Determines task order

- **For each host, Ansible**:
- Establishes SSH connection
- Gathers facts (system information)
- Transfers modules to the host
- Executes modules
- Collects results
- Closes connection

- **Ansible displays results**
- OK: Task succeeded, no changes made
- Changed: Task succeeded, changes made
- Failed: Task failed
- Skipped: Task was skipped (conditional)

2.5 Ansible vs Other Configuration Management Tools

Ansible vs Puppet

Feature	Ansible	Puppet
Architecture	**Agentless**	**Agent-based**
Communication	Push	Pull
Language	YAML	Puppet DSL
Learning Curve	Easy	Steep
Setup Time	Minutes	Hours/Days
Windows Support	Good	Good
Scalability	Good (5,000+ nodes)	Excellent (10,000+ nodes)
Community	Very Large	Large

When to use Ansible: Quick setup, simpler environments, SSH-accessible nodes

When to use Puppet: Extremely large scale, need for reporting, complex dependency management

Ansible vs Chef

Feature	Ansible	Chef
Architecture	Agentless	Agent-based
Communication	Push	Pull
Language	YAML	Ruby DSL
Learning Curve	Easy	Very Steep
Configuration	Declarative	Procedural
Cloud Integration	Excellent	Good
Cost	Free (Tower paid)	Paid (Enterprise)

When to use Ansible: Prefer simplicity, no Ruby expertise, agentless architecture

When to use Chef: Need programmatic approach, have Ruby developers, complex workflows

Ansible vs SaltStack

Feature	Ansible	SaltStack
Architecture	Agentless	Agent-based (or agentless)
Communication	Push	Push/Pull hybrid
Language	YAML	YAML/Python
Speed	Fast	Very Fast
Learning Curve	Easy	Medium
Event-Driven	Limited	Excellent
Remote Execution	Good	Excellent

When to use Ansible: General automation, simpler deployment

When to use SaltStack: Need speed, event-driven architecture, real-time monitoring

2.6 Ansible Key Features in Detail

2.6.1 Agentless Architecture

Advantages:

- No software to install on managed nodes
- No version compatibility issues
- No resource overhead on managed nodes
- Simpler security model (SSH only)
- Easier to get started
- No need to upgrade agents across infrastructure

How it Works:

- Uses OpenSSH for Linux/Unix
- Uses WinRM for Windows
- Python modules executed remotely
- Modules copied to temp directory, executed, then removed

2.6.2 Idempotency

Definition: An operation is idempotent if applying it multiple times produces the same result as applying it once.

Example:

```
- name: Ensure Apache is installed
  yum:
    name: httpd
    state: present
```

First run: Installs Apache (Changed)

Second run: Apache already installed (OK, no change)

Third run: Apache already installed (OK, no change)

Benefits:

- Safe to run repeatedly
- Convergence to desired state
- No accidental duplications
- Predictable outcomes

2.6.3 Infrastructure as Code (IaC)

Ansible playbooks are text files that can be:

- Stored in version control (Git)
- Reviewed through pull requests
- Tested before deployment
- Rolled back if needed
- Shared across teams
- Documented as code

2.6.4 Extensibility

Ways to Extend Ansible:

- Custom modules in Python or any language
- Plugins (connection, callback, lookup, etc.)
- Filters for Jinja2 templates
- Custom roles
- Dynamic inventory scripts
- Callback plugins for custom output

2.7 Real-World Use Cases

Use Case 1: Web Server Deployment

Scenario: Deploy 50 web servers with identical configuration

Without Ansible: 2-3 days of manual work

With Ansible:

```
- hosts: webservers
  roles:
    - common
    - apache
    - application
```

Time: 15 minutes

Use Case 2: Security Patch Rollout

Scenario: Apply critical security patch to 500 servers

Ansible Solution:

```
- hosts: all
  tasks:
    - name: Update specific package
      yum:
        name: openssl
        state: latest
```

- Parallel execution across all servers
- Complete in under 30 minutes
- Full audit log of changes

Use Case 3: Disaster Recovery

Scenario: Data center failure, need to rebuild 100 servers

Ansible Solution:

- Keep all configurations in Git
- Provision new infrastructure (cloud or on-prem)
- Run playbooks to restore all services
- Complete rebuild in hours, not days or weeks

2.8 Ansible Ecosystem

Core Components:

- **Ansible Core**: Free, open-source automation engine
- **Ansible Tower**: Commercial web UI and API (paid)
- **AWX**: Open-source version of Tower
- **Ansible Galaxy**: Community hub for roles and collections
- **Ansible Collections**: Packaged content (modules, roles, plugins)

Community:

- GitHub: github.com/ansible
- Forum: groups.google.com/forum/#!forum/ansible-project
- Docs: docs.ansible.com
- IRC: #ansible on freenode
- Reddit: r/ansible
- Meetups: Local Ansible meetups worldwide

2.9 Hands-On Lab: Understanding Ansible Architecture

Objective: Set up a minimal Ansible environment and understand component interaction

Requirements:

- 2 Linux VMs or containers (1 control node, 1 managed node)
- SSH access between them

Lab Steps:

- **Set up Control Node**:

```
# Install Ansible
sudo pip3 install ansible

# Verify installation
ansible --version

# Create project directory
mkdir ~/ansible-lab
cd ~/ansible-lab
```

- **Create Simple Inventory**:

```
cat > inventory << EOF
[test]
managed-node-1 ansible_host=192.168.1.10
EOF
```

- **Test Connectivity**:

```
# Ping test (not ICMP, Ansible ping module)
ansible -i inventory test -m ping -u username --ask-pass
```

- **Understand the Flow**:
- Run with verbose flag to see execution details:

```
ansible -i inventory test -m ping -u username --ask-pass -vvv
```

- **Observe**:
- SSH connection establishment
- Python module transfer
- Module execution
- Result collection
- Connection closure

Expected Output:

```
managed-node-1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Key Learnings:

- Ansible doesn't need to be installed on managed nodes
- SSH is used for communication
- Modules are transferred and executed remotely
- Results are returned in JSON format

2.10 Common Misconceptions About Ansible

- **"Ansible is only for configuration management"**
- False: Also used for orchestration, deployment, provisioning, security automation

- **"Ansible is slow because it's agentless"**
- False: SSH is fast, and Ansible can parallelize across many hosts

- **"You need to know Python to use Ansible"**
- False: YAML playbooks don't require Python knowledge (but helpful for advanced use)

- **"Ansible doesn't scale"**
- False: Ansible handles thousands of nodes effectively with proper architecture

- **"Ansible is only for Linux"**
- False: Supports Windows, network devices, cloud platforms, and more

3. Installation Methods

3.1 Prerequisites and System Requirements

Control Node Requirements

Operating System:

- Red Hat Enterprise Linux 8, 9
- CentOS Stream 8, 9
- Fedora 36+
- Ubuntu 20.04 LTS, 22.04 LTS
- Debian 10, 11
- macOS (recent versions)
- Windows with WSL2 (Ubuntu or Debian)

Python:

- Ansible 2.15+: Python 3.9 or newer
- Ansible 2.14: Python 3.8 or newer
- Older versions: Python 2.7 or Python 3.5+

Other Requirements:

- SSH client
- Sufficient disk space (1-2 GB for Ansible and dependencies)
- Internet connectivity (for package installation)

Managed Node Requirements

Operating System:

- Any Linux distribution
- Windows Server 2012+
- Network devices with SSH/API access
- BSD variants
- macOS

Python:

- Python 2.7 or Python 3.5+
- Windows: PowerShell 3.0+ (no Python needed)

Other Requirements:

- SSH server (Linux/Unix)
- WinRM configured (Windows)
- Proper user account with sudo/admin privileges

3.2 Installation on RHEL/CentOS/Fedora

Method 1: Using DNF/YUM Package Manager

RHEL/CentOS 8/9:

```
# Enable EPEL repository
sudo dnf install epel-release -y

# Update package cache
sudo dnf update -y

# Install Ansible
sudo dnf install ansible -y

# Verify installation
ansible --version
```

CentOS 7:

```
# Enable EPEL repository
sudo yum install epel-release -y

# Update package cache
sudo yum update -y

# Install Ansible
sudo yum install ansible -y

# Verify installation
ansible --version
```

Fedora:

```
# Ansible available in default repos
sudo dnf install ansible -y

# Verify installation
ansible --version
```

Pros:

- Simple and quick
- Automatic dependency resolution
- Easy to update with system package manager

Cons:

- May not have the latest Ansible version
- Depends on distribution maintainers

Method 2: Using pip (Python Package Manager)

This method installs the latest version of Ansible.

```
# Install Python 3 and pip if not already installed
sudo dnf install python3 python3-pip -y

# Upgrade pip
sudo pip3 install --upgrade pip

# Install Ansible
sudo pip3 install ansible

# Or install for current user only (no sudo required)
pip3 install --user ansible

# Add to PATH if installed with --user
echo 'export PATH=$PATH:~/.local/bin' >> ~/.bashrc
source ~/.bashrc

# Verify installation
ansible --version
```

Pros:

- Latest Ansible version
- Easy to upgrade
- Can install specific versions

Cons:

- Requires Python and pip knowledge
- Manual PATH configuration may be needed
- No automatic security updates from OS

3.3 Installation on Ubuntu/Debian

Method 1: Using APT Package Manager

Ubuntu 20.04/22.04:

```
# Update package cache
sudo apt update

# Install software-properties-common
sudo apt install software-properties-common -y

# Add Ansible PPA (Personal Package Archive)
sudo add-apt-repository --yes --update ppa:ansible/ansible

# Install Ansible
sudo apt install ansible -y

# Verify installation
ansible --version
```

Debian 10/11:

```
# Update package cache
sudo apt update

# Install dependencies
sudo apt install gnupg2 -y

# Add Ansible signing key
echo "deb http://ppa.launchpad.net/ansible/ansible/ubuntu focal main"
  sudo tee /etc/apt/sources.list.d/ansible.list

# Update package cache
sudo apt update

# Install Ansible
sudo apt install ansible -y

# Verify installation
ansible --version
```

Method 2: Using pip

```
# Install Python 3 and pip
sudo apt update
sudo apt install python3 python3-pip -y

# Upgrade pip
sudo pip3 install --upgrade pip

# Install Ansible
sudo pip3 install ansible

# Verify installation
ansible --version
```

3.4 Installation on AWS EC2 Instances

This is a common scenario for cloud-based infrastructure automation.

Amazon Linux 2

Full Installation Script:

```
#!/bin/bash
# Install Ansible on Amazon Linux 2

# Update system packages
sudo yum update -y

# Install development tools
sudo yum groupinstall "Development Tools" -y

# Install dependencies for Python compilation
sudo yum install -y \
    openssl-devel \
    bzip2-devel \
    libffi-devel \
    wget \
    tar \
    gzip

# Download Python 3.9
cd /tmp
wget https://www.python.org/ftp/python/3.9.16/Python-3.9.16.tgz

# Extract Python
tar xzf Python-3.9.16.tgz
cd Python-3.9.16

# Configure and compile Python
./configure --enable-optimizations
sudo make altinstall

# Verify Python installation
python3.9 --version

# Install pip for Python 3.9
python3.9 -m pip install --upgrade pip

# Install Ansible
python3.9 -m pip install ansible

# Create symbolic link for convenience
```

```
sudo ln -s /usr/local/bin/python3.9 /usr/local/bin/python3
sudo ln -s /usr/local/bin/pip3.9 /usr/local/bin/pip3

# Add to PATH
echo 'export PATH=$PATH:/usr/local/bin' >> ~/.bashrc
source ~/.bashrc

# Verify Ansible installation
ansible --version

# Create Ansible directory structure
sudo mkdir -p /etc/ansible
cd /etc/ansible

# Create basic ansible.cfg
sudo tee /etc/ansible/ansible.cfg > /dev/null <<EOF
[defaults]
inventory = /etc/ansible/hosts
host_key_checking = False
retry_files_enabled = False
gathering = smart
fact_caching = jsonfile
fact_caching_connection = /tmp/ansible_facts
fact_caching_timeout = 86400
EOF

# Create basic hosts file
sudo tee /etc/ansible/hosts > /dev/null <<EOF
# Ansible inventory file
# Example:
# [webservers]
# web1.example.com
# web2.example.com

[localhost]
127.0.0.1 ansible_connection=local
EOF

echo "Ansible installation completed successfully!"
echo "Ansible version:"
ansible --version
```

Save and Execute:

```
# Save the script
nano install-ansible-ec2.sh

# Make executable
chmod +x install-ansible-ec2.sh

# Run the script
./install-ansible-ec2.sh
```

Ubuntu on EC2

```
#!/bin/bash
# Install Ansible on Ubuntu EC2 instance

# Update system
sudo apt update
sudo apt upgrade -y

# Install dependencies
sudo apt install -y \
    software-properties-common \
    python3-pip

# Add Ansible PPA
sudo add-apt-repository --yes --update ppa:ansible/ansible

# Install Ansible
sudo apt install ansible -y

# Verify installation
ansible --version

echo "Ansible installation completed on Ubuntu EC2!"
```

3.5 Installation on Windows using WSL (Windows Subsystem for Linux)

Windows cannot be a control node natively, but WSL2 provides a Linux environment on Windows.

Step 1: Enable WSL2

PowerShell (Run as Administrator):

```
# Enable WSL
wsl --install

# Or manually enable features
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsy
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform

# Restart computer
Restart-Computer
```

Step 2: Install Ubuntu from Microsoft Store

- Open Microsoft Store
- Search for "Ubuntu 22.04 LTS"
- Click "Install"
- Launch Ubuntu
- Set up username and password

Step 3: Install Ansible in WSL

```
# Update package manager
sudo apt update

# Install Ansible
sudo apt install software-properties-common
sudo add-apt-repository --yes --update ppa:ansible/ansible
sudo apt install ansible -y

# Verify installation
ansible --version
```

WSL-Specific Configuration

```
# Create Ansible config with Windows compatibility
cat > ~/.ansible.cfg << 'EOF'
[defaults]
inventory = ~/ansible/inventory
host_key_checking = False
interpreter_python = auto_silent

[ssh_connection]
ssh_args = -o ControlMaster=auto -o ControlPersist=60s
EOF
```

3.6 Installation from Source

For the latest development version or custom builds.

```
# Install dependencies
sudo apt install -y git python3 python3-pip

# Clone Ansible repository
git clone https://github.com/ansible/ansible.git --recursive
cd ansible

# Install Python dependencies
pip3 install --user -r requirements.txt

# Source environment
source ./hacking/env-setup

# Verify installation
ansible --version
```

3.7 Installing Specific Ansible Versions

Using pip

```
# Install specific version
pip3 install ansible==2.15.0

# Install version range
pip3 install 'ansible>=2.14,<2.16'

# Upgrade to latest
pip3 install --upgrade ansible

# Downgrade to specific version
pip3 install ansible==2.14.0 --force-reinstall
```

Check Available Versions

```
# List available versions
pip3 index versions ansible

# Or use curl
curl -s https://pypi.org/pypi/ansible/json | python3 -m json.tool | g
```

3.8 Creating Python Virtual Environment for Ansible

Isolate Ansible installations to avoid conflicts.

```
# Install virtualenv
pip3 install virtualenv

# Create virtual environment
mkdir ~/ansible-env
cd ~/ansible-env
virtualenv venv

# Activate virtual environment
source venv/bin/activate

# Install Ansible in virtual environment
pip install ansible

# Verify
which ansible
ansible --version

# Deactivate when done
deactivate
```

3.9 Post-Installation Configuration

Create Directory Structure

```
# Create standard Ansible directories
sudo mkdir -p /etc/ansible
sudo mkdir -p /etc/ansible/group_vars
sudo mkdir -p /etc/ansible/host_vars
sudo mkdir -p /etc/ansible/roles

# Create user-specific directories
mkdir -p ~/ansible
mkdir -p ~/ansible/playbooks
mkdir -p ~/ansible/inventory
mkdir -p ~/ansible/roles
```

Create Basic Configuration File

```
# Create ansible.cfg
sudo tee /etc/ansible/ansible.cfg > /dev/null <<'EOF'
[defaults]
inventory      = /etc/ansible/hosts
library        = /usr/share/my_modules/
remote_tmp     = ~/.ansible/tmp
local_tmp      = ~/.ansible/tmp
forks          = 5
poll_interval  = 15
sudo_user      = root
ask_sudo_pass  = False
ask_pass       = False
transport      = smart
remote_port    = 22
module_lang    = C
gathering      = smart
host_key_checking = False
retry_files_enabled = False
log_path       = /var/log/ansible.log

[inventory]
enable_plugins = host_list, script, auto, yaml, ini, toml

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

[paramiko_connection]
record_host_keys = False

[ssh_connection]
ssh_args = -o ControlMaster=auto -o ControlPersist=60s
pipelining = True
EOF
```

Create Basic Inventory File

```
# Create hosts file
sudo tee /etc/ansible/hosts > /dev/null <<'EOF'
# Ansible Inventory File
# Format: hostname or IP address

# Example static inventory
[local]
localhost ansible_connection=local

[webservers]
#web1.example.com
#web2.example.com
#192.168.1.10
#192.168.1.11

[databases]
#db1.example.com
#db2.example.com

[production:children]
#webservers
#databases
EOF
```

3.10 Verification and Testing

Test Ansible Installation

```
# Check version
ansible --version

# Expected output:
# ansible [core 2.15.x]
#   config file = /etc/ansible/ansible.cfg
#   configured module search path = ['/home/user/.ansible/plugins/mod
#   ansible python module location = /usr/local/lib/python3.9/site-pa
#   ansible collection location = /home/user/.ansible/collections
#   executable location = /usr/local/bin/ansible
#   python version = 3.9.x
#   jinja version = 3.1.x
#   libyaml = True
```

Test Local Execution

```
# Test localhost connectivity
ansible localhost -m ping

# Expected output:
# localhost | SUCCESS => {
#     "changed": false,
#     "ping": "pong"
# }
```

Test with Ad-hoc Command

```
# Get hostname from localhost
ansible localhost -m command -a "hostname"

# Get current date
ansible localhost -m command -a "date"

# Check disk space
ansible localhost -m shell -a "df -h"
```

3.11 Troubleshooting Installation Issues

Common Issue 1: Python Version Mismatch

Error: `ERROR: ansible requires Python '>=3.9' but the running Python is 3.6`

Solution:

```
# Install Python 3.9+
sudo apt install python3.9 python3.9-pip -y

# Install Ansible with specific Python version
python3.9 -m pip install ansible

# Verify
ansible --version | grep "python version"
```

Common Issue 2: pip Not Found

Error: `bash: pip: command not found`

Solution:

```
# Install pip
sudo apt install python3-pip -y

# Or use get-pip.py
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
python3 get-pip.py
```

Common Issue 3: Permission Denied

Error: `Permission denied when trying to install Ansible`

Solution:

```
# Option 1: Install with sudo
sudo pip3 install ansible

# Option 2: Install for current user
pip3 install --user ansible
```

Common Issue 4: Module Not Found

Error: `ModuleNotFoundError: No module named 'packaging'`

Solution:

```
# Install missing dependencies
pip3 install packaging
pip3 install resolvelib

# Or reinstall Ansible
pip3 install --upgrade --force-reinstall ansible
```

3.12 Hands-On Lab: Multi-Platform Installation

Objective: Install Ansible on different platforms and understand differences

Lab Setup: Install Ansible on at least two different platforms

Task 1: Ubuntu Installation

```
# Document each step
echo "Starting Ubuntu installation" | tee install-log.txt
date >> install-log.txt

# Install Ansible
sudo apt update
sudo apt install software-properties-common -y
sudo add-apt-repository --yes --update ppa:ansible/ansible
sudo apt install ansible -y

# Document completion
echo "Installation completed" | tee -a install-log.txt
ansible --version | tee -a install-log.txt
date >> install-log.txt
```

Task 2: CentOS/RHEL Installation

```
# Document installation process
echo "Starting CentOS installation" > install-log-centos.txt
date >> install-log-centos.txt

# Install Ansible
sudo yum install epel-release -y
sudo yum install ansible -y

# Document completion
echo "Installation completed" >> install-log-centos.txt
ansible --version >> install-log-centos.txt
date >> install-log-centos.txt
```

Task 3: Compare Results

- Compare installation time
- Compare installed versions

- Note any differences in dependencies
- Document challenges faced on each platform

Deliverable: A comparison report documenting your findings

3.13 Upgrading Ansible

Using Package Manager

```
# Ubuntu/Debian
sudo apt update
sudo apt upgrade ansible

# RHEL/CentOS
sudo yum update ansible

# Fedora
sudo dnf upgrade ansible
```

Using pip

```
# Upgrade to latest version
pip3 install --upgrade ansible

# Check current version before upgrade
ansible --version

# Check new version after upgrade
ansible --version
```

Best Practices for Upgrades

- **Test in Non-Production First**: Always test upgrades in dev/test environment
- **Review Changelog**: Check release notes for breaking changes
- **Backup Playbooks**: Ensure playbooks are in version control
- **Check Module Changes**: Some modules may be deprecated or changed
- **Update Python**: Ensure Python version meets requirements

3.14 Uninstalling Ansible

If Installed via Package Manager

```
# Ubuntu/Debian
sudo apt remove ansible -y
sudo apt autoremove -y

# RHEL/CentOS
sudo yum remove ansible -y
```

If Installed via pip

```
# Uninstall Ansible
pip3 uninstall ansible

# Remove configuration files
sudo rm -rf /etc/ansible
rm -rf ~/.ansible
```

4. Ansible Configuration Files and Settings

4.1 Ansible Configuration Hierarchy

Ansible searches for configuration files in the following order (first found is used):

- **ANSIBLE_CONFIG** environment variable
- **ansible.cfg** in the current directory
- **~/.ansible.cfg** in the user's home directory
- **/etc/ansible/ansible.cfg** system-wide configuration

Example:

```
# Set via environment variable (highest priority)
export ANSIBLE_CONFIG=/path/to/custom/ansible.cfg

# Or create in current directory
touch ./ansible.cfg

# Or in home directory
touch ~/.ansible.cfg
```

4.2 Understanding ansible.cfg Structure

The configuration file uses INI format with sections and key-value pairs.

Basic Structure:

```
[section_name]
key1 = value1
key2 = value2

[another_section]
key3 = value3
```

4.3 The [defaults] Section

This section contains the most commonly used settings.

Complete Example:

```
[defaults]

# Inventory location
inventory = ./inventory/hosts

# Number of parallel processes
forks = 5

# Default user for SSH
remote_user = ansible

# Disable host key checking (use cautiously)
host_key_checking = False

# Path to role collections
roles_path = ./roles:/usr/share/ansible/roles

# Module search path
library = ./library:/usr/share/my_modules

# Logging
log_path = /var/log/ansible.log

# Fact gathering
gathering = smart
fact_caching = jsonfile
fact_caching_connection = /tmp/ansible_facts
fact_caching_timeout = 86400

# Retry files (disabled recommended)
retry_files_enabled = False

# Callback plugins
stdout_callback = yaml
bin_ansible_callbacks = True

# SSH timeout
timeout = 30

# Deprecation warnings
```

```
deprecation_warnings = True

# Command warnings
command_warnings = True

# Jinja2 extensions
jinja2_extensions = jinja2.ext.do,jinja2.ext.i18n

# Ansible collections path
collections_paths = ./collections:/usr/share/ansible/collections

# Module arguments
module_args = ""

# Task includes are static by default
task_includes_static = True

# Handler flush behavior
any_errors_fatal = False
```

Key Settings Explained:

inventory

```
inventory = /etc/ansible/hosts
```

- Specifies the default inventory file location
- Can be a directory containing multiple inventory files
- Supports multiple formats (INI, YAML, JSON)

forks

```
forks = 5
```

- Number of parallel processes to spawn
- Increase for larger infrastructures (50-100)
- Higher values require more system resources

remote_user

```
remote_user = ansible
```

- Default SSH user for connections
- Can be overridden per host or playbook
- Should have sudo privileges on managed nodes

host_key_checking

```
host_key_checking = False
```

- **False**: Disables SSH host key checking (convenient but less secure)
- **True**: Requires SSH keys to be in known_hosts (more secure)
- Production recommendation: True

roles_path

```
roles_path = ./roles:/usr/share/ansible/roles:~/.ansible/roles
```

- Colon-separated list of directories to search for roles
- Searched in order specified
- First match is used

log_path

```
log_path = /var/log/ansible.log
```

- Location for Ansible log file
- User must have write permissions
- Useful for auditing and troubleshooting
- Commented out by default (no logging)

gathering

```
gathering = smart
```

- **implicit**: Always gather facts
- **explicit**: Never gather facts unless requested
- **smart**: Cache facts and use them (default)

timeout

```
timeout = 10
```

- SSH connection timeout in seconds
- Increase for slow networks or busy systems

4.4 The [privilege_escalation] Section

Controls privilege escalation settings.

```
[privilege_escalation]

# Enable privilege escalation
become = True

# Method to use for privilege escalation
become_method = sudo

# User to become (usually root)
become_user = root

# Prompt for password
become_ask_pass = False

# Flags to pass to become method
become_flags = -H -S -n
```

Become Methods:

- **sudo**: Most common, uses /etc/sudoers
- **su**: Switch user, requires root password
- **pbrun**: PowerBroker
- **pfexec**: Solaris privilege escalation
- **doas**: OpenBSD alternative to sudo
- **dzdo**: Centrify
- **ksu**: Kerberos substitute user
- **runas**: Windows run as

4.5 The [ssh_connection] Section

SSH-specific settings for better performance.

```
[ssh_connection]

# SSH arguments
ssh_args = -o ControlMaster=auto -o ControlPersist=60s -o PreferredAu

# Enable pipelining for better performance
pipelining = True

# Control path for SSH multiplexing
control_path = %(directory)s/ansible-ssh-%%h-%%p-%%r

# SCP if available, sftp otherwise
transfer_method = smart

# Retries for SSH connection
retries = 3
```

Important Settings:

pipelining

```
pipelining = True
```

- Reduces number of SSH connections
- Significantly improves performance
- **Requirement**: `requiretty` must be disabled in /etc/sudoers

Enable pipelining:

```
# Edit sudoers file on managed nodes
sudo visudo

# Comment out or remove:
# Defaults    requiretty

# Or add:
Defaults:ansible_user !requiretty
```

ssh_args

```
ssh_args = -o ControlMaster=auto -o ControlPersist=60s
```

- **ControlMaster**: Enables SSH multiplexing
- **ControlPersist**: Keeps connections open for reuse
- Dramatically improves performance for multiple tasks

4.6 The [inventory] Section

Controls inventory plugin settings.

```
[inventory]

# Enable specific inventory plugins
enable_plugins = host_list, virtualbox, yaml, constructed, aws_ec2, a

# Ignore patterns
ignore_patterns = *.bak, *~

# Inventory caching
cache = True
cache_plugin = jsonfile
cache_timeout = 86400
cache_connection = /tmp/ansible_inventory_cache
```

4.7 The [colors] Section

Customize output colors.

```
[colors]

# Customization options
highlight = white
verbose = blue
warn = bright purple
error = red
debug = dark gray
deprecate = purple
skip = cyan
unreachable = red
ok = green
changed = yellow
diff_add = green
diff_remove = red
diff_lines = cyan
```

4.8 Complete Production ansible.cfg Example

```
# /etc/ansible/ansible.cfg
# Production Ansible Configuration

[defaults]
# Inventory
inventory = /etc/ansible/inventory/production

# Connection settings
remote_user = ansible
remote_port = 22
timeout = 30

# Performance
forks = 50
poll_interval = 15
gathering = smart
fact_caching = jsonfile
fact_caching_connection = /var/cache/ansible/facts
fact_caching_timeout = 86400

# Paths
roles_path = /etc/ansible/roles
library = /usr/share/ansible/plugins/modules
collections_paths = /etc/ansible/collections

# Logging and output
log_path = /var/log/ansible/ansible.log
stdout_callback = yaml
callback_whitelist = profile_tasks, timer
display_skipped_hosts = False
display_ok_hosts = True

# Security
host_key_checking = True
private_key_file = /home/ansible/.ssh/id_rsa

# Behavior
retry_files_enabled = False
command_warnings = True
deprecation_warnings = True
```

```
action_warnings = True

# Error handling
any_errors_fatal = False
max_fail_percentage = 0

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

[ssh_connection]
ssh_args = -o ControlMaster=auto -o ControlPersist=60s -o ServerAlive
pipelining = True
control_path = /tmp/ansible-ssh-%%h-%%p-%%r
transfer_method = smart

[persistent_connection]
connect_timeout = 30
command_timeout = 30

[inventory]
enable_plugins = aws_ec2, azure_rm, gcp_compute, host_list, yaml, ini

[colors]
highlight = white
verbose = blue
warn = yellow
error = red
ok = green
changed = yellow
```

4.9 Development/Testing ansible.cfg Example

```
# Development Configuration
[defaults]
inventory = ./inventory/dev.ini
remote_user = vagrant
host_key_checking = False
retry_files_enabled = False
log_path = ./ansible.log
stdout_callback = debug
verbosity = 1
gathering = explicit

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

[ssh_connection]
pipelining = True
```

4.10 Environment Variables

Override configuration with environment variables.

Common Variables:

```
# Config file location
export ANSIBLE_CONFIG=/path/to/ansible.cfg

# Inventory location
export ANSIBLE_INVENTORY=/path/to/inventory

# Roles path
export ANSIBLE_ROLES_PATH=/path/to/roles

# Verbosity level
export ANSIBLE_VERBOSITY=3

# SSH user
export ANSIBLE_REMOTE_USER=ansible

# SSH port
export ANSIBLE_REMOTE_PORT=2222

# Become settings
export ANSIBLE_BECOME=True
export ANSIBLE_BECOME_METHOD=sudo
export ANSIBLE_BECOME_USER=root

# Host key checking
export ANSIBLE_HOST_KEY_CHECKING=False

# Log path
export ANSIBLE_LOG_PATH=/var/log/ansible.log

# Number of forks
export ANSIBLE_FORKS=10

# Gathering mode
export ANSIBLE_GATHERING=explicit

# Fact caching
export ANSIBLE_CACHE_PLUGIN=jsonfile
export ANSIBLE_CACHE_PLUGIN_CONNECTION=/tmp/facts_cache
export ANSIBLE_CACHE_PLUGIN_TIMEOUT=86400
```

Example Usage:

```
# Run playbook with specific config
ANSIBLE_CONFIG=./custom.cfg ansible-playbook site.yml

# Run with increased verbosity
ANSIBLE_VERBOSITY=3 ansible-playbook deploy.yml

# Disable host key checking for one run
ANSIBLE_HOST_KEY_CHECKING=False ansible-playbook test.yml
```

4.11 Viewing Current Configuration

```
# View all configuration settings
ansible-config dump

# View only changed settings
ansible-config dump --only-changed

# List all config parameters
ansible-config list

# View specific parameter
ansible-config dump | grep -i forks
```

4.12 Configuration File Best Practices

- **Use Version Control**: Keep ansible.cfg in Git

- **Project-Specific Configuration**: Place ansible.cfg in project root

- **Minimal Global Config**: Keep /etc/ansible/ansible.cfg minimal

- **Document Changes**: Comment all non-default settings

- **Security Settings**:

```
# Good security practices
host_key_checking = True
private_key_file = ~/.ssh/ansible_key
ask_vault_pass = True
```

- **Performance Settings**:

```
# For large infrastructures
forks = 50
gathering = smart
fact_caching = jsonfile
pipelining = True
```

- **Development vs Production**: Maintain separate configurations

- **Avoid Hardcoding**: Use environment variables for sensitive data

4.13 Ansible Directories and File Locations

Standard Installation Directory Structure

```
/etc/ansible/
├── ansible.cfg          # Main configuration file
├── hosts                # Default inventory file
├── roles/               # System-wide roles
│   ├── role1/
│   └── role2/
├── group_vars/          # Variables for inventory groups
│   ├── all.yml
│   ├── webservers.yml
│   └── databases.yml
├── host_vars/           # Variables for specific hosts
│   ├── server1.yml
│   └── server2.yml
├── inventory/           # Multiple inventory files
│   ├── production
│   ├── staging
│   └── development
└── collections/         # Ansible collections
    └── ansible_collections/
```

User-Specific Directories

```
~/.ansible/
├── .ansible.cfg          # User configuration
├── collections/          # User-installed collections
├── roles/                # User-specific roles
├── tmp/                  # Temporary files
└── cp/                   # SSH control path sockets

~/.ansible/tmp/           # Remote temporary directory
~/.ansible/cp/            # SSH multiplexing control sockets
```

Project-Specific Structure

```
my-ansible-project/
├── ansible.cfg          # Project configuration
├── inventory/
│   ├── production.ini
│   ├── staging.ini
│   └── development.ini
├── group_vars/
│   ├── all.yml
│   ├── webservers.yml
│   └── databases.yml
├── host_vars/
│   └── server1.example.com.yml
├── roles/
│   ├── common/
│   ├── webserver/
│   └── database/
├── playbooks/
│   ├── site.yml
│   ├── webservers.yml
│   └── databases.yml
├── files/
│   └── config_files/
├── templates/
│   └── app.conf.j2
├── vars/
│   └── external_vars.yml
└── collections/
    └── requirements.yml
```

4.14 Hands-On Lab: Configuration Management

Objective: Create and test different Ansible configurations

Lab Tasks:

Task 1: Create Project Configuration

```
# Create project directory
mkdir -p ~/ansible-lab
cd ~/ansible-lab

# Create basic ansible.cfg
cat > ansible.cfg << 'EOF'
[defaults]
inventory = ./inventory
host_key_checking = False
retry_files_enabled = False
log_path = ./ansible.log
stdout_callback = yaml

[privilege_escalation]
become = True
become_method = sudo
EOF

# Create inventory
mkdir inventory
cat > inventory/hosts << 'EOF'
[local]
localhost ansible_connection=local
EOF

# Test configuration
ansible-config dump --only-changed
```

Task 2: Test Different Forks Settings

```
# Create test playbook
cat > test-forks.yml << 'EOF'
---
- name: Test forks performance
  hosts: localhost
  gather_facts: no
  tasks:
    - name: Simulate work
      debug:
        msg: "Task {{ item }}"
      loop: "{{ range(1, 21) | list }}"
EOF

# Test with forks=1
echo "[defaults]
forks = 1" > ansible.cfg
time ansible-playbook test-forks.yml

# Test with forks=5
echo "[defaults]
forks = 5" > ansible.cfg
time ansible-playbook test-forks.yml

# Test with forks=10
echo "[defaults]
forks = 10" > ansible.cfg
time ansible-playbook test-forks.yml

# Compare execution times
```

Task 3: Test Configuration Priority

```
# Create multiple config files
# System-wide
echo "[defaults]
forks = 1" | sudo tee /etc/ansible/ansible.cfg

# User home
echo "[defaults]
forks = 5" > ~/.ansible.cfg

# Current directory
echo "[defaults]
forks = 10" > ./ansible.cfg

# Test which one is used
ansible-config dump --only-changed | grep forks

# Remove to test priority
rm ./ansible.cfg
ansible-config dump --only-changed | grep forks

rm ~/.ansible.cfg
ansible-config dump --only-changed | grep forks
```

Task 4: Create Production-Ready Configuration

Create a complete production configuration with proper settings for security, performance, and logging.

Expected Learnings:

- Configuration file precedence
- Impact of different settings on performance
- Proper security configurations
- Logging and auditing setup

5. SSH Authentication and Key Management

5.1 Understanding SSH in Ansible Context

Ansible uses OpenSSH as its default transport mechanism for Linux/Unix systems. Understanding SSH authentication is crucial for effective Ansible usage.

Why SSH:

- Industry standard secure protocol
- Widely available and well-understood
- No additional software needed on managed nodes
- Built-in encryption and authentication
- Supports multiplexing for performance

5.2 SSH Authentication Methods

Method 1: Password Authentication

Pros:

- Simple to set up
- No key management needed

Cons:

- Less secure
- Requires `sshpass` utility
- Doesn't scale well
- Cannot be automated securely

Usage:

```
# Install sshpass
sudo apt install sshpass  # Ubuntu/Debian
sudo yum install sshpass  # RHEL/CentOS

# Run Ansible with password
ansible all -m ping -u username --ask-pass

# In playbook
ansible-playbook site.yml -u username --ask-pass
```

Method 2: SSH Key Authentication (Recommended)

Pros:

- More secure
- No password prompts
- Fully automatable
- Scales well
- Industry best practice

Cons:

- Initial setup required

- Key management overhead

5.3 SSH Key-Based Authentication Setup

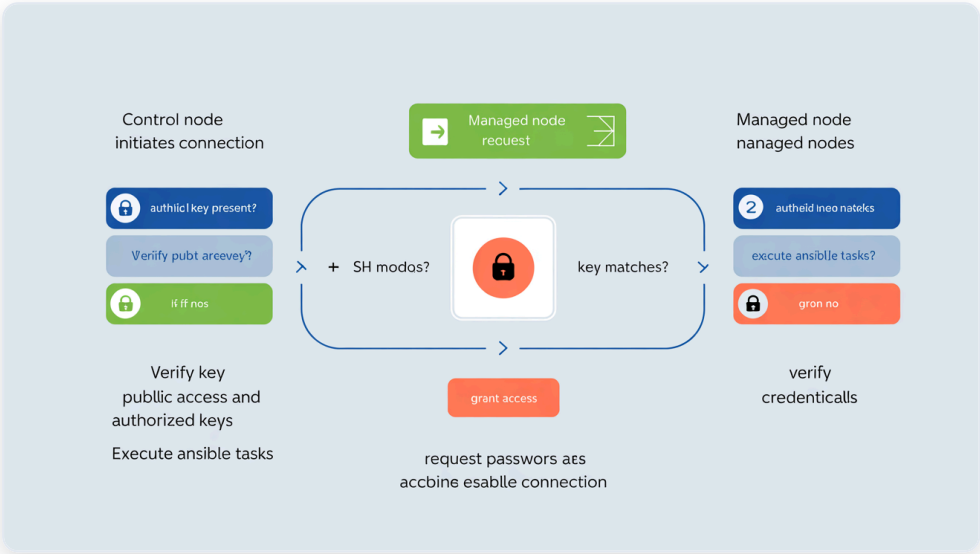


Figure: 5.3 SSH Key-Based Authentication Setup

Step 1: Generate SSH Key Pair on Control Node

```
# Generate RSA key pair (recommended)
ssh-keygen -t rsa -b 4096 -C "ansible-control-node"

# Interactive prompts:
# Enter file: /home/ansible/.ssh/id_rsa
# Enter passphrase: (press Enter for no passphrase, or enter secure p

# Generate ED25519 key (modern, more secure, faster)
ssh-keygen -t ed25519 -C "ansible-control-node"

# Generate with custom name and location
ssh-keygen -t rsa -b 4096 -f ~/.ssh/ansible_key -C "ansible-automatio
```

Key Types:

- **RSA**: Most compatible, 2048-4096 bits
- **ED25519**: Modern, secure, fast (recommended if supported)
- **ECDSA**: Elliptic curve, good performance
- **DSA**: Deprecated, avoid

Generated Files:

```
~/.ssh/
├── id_rsa           # Private key (never share!)
└── id_rsa.pub       # Public key (copy to managed nodes)
```

Step 2: Copy Public Key to Managed Nodes

Method 1: Using ssh-copy-id (Easy)

```
# Copy to single host
ssh-copy-id username@managed-node1

# Copy to specific port
ssh-copy-id -p 2222 username@managed-node1

# Copy specific key file
ssh-copy-id -i ~/.ssh/ansible_key.pub username@managed-node1
```

Method 2: Manual Copy

```
# Copy public key content
cat ~/.ssh/id_rsa.pub

# On managed node, add to authorized_keys
mkdir -p ~/.ssh
chmod 700 ~/.ssh
echo "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQC..." >> ~/.ssh/authorize
chmod 600 ~/.ssh/authorized_keys
```

Method 3: Using Ansible (Bootstrap)

```
# First run with password to copy keys
ansible all -m authorized_key \
  -a "user=ansible key='{{ lookup('file', '~/.ssh/id_rsa.pub') }}' st
  -u ansible --ask-pass --become
```

Step 3: Test SSH Key Authentication

```
# Test SSH connection
ssh username@managed-node1

# Should log in without password prompt

# Test from Ansible
ansible all -m ping -u username
```

5.4 SSH Configuration for Ansible

Client-Side SSH Configuration

Create or edit `~/.ssh/config`:

```
# Global SSH settings for Ansible
Host *
    ServerAliveInterval 60
    ServerAliveCountMax 3
    Compression yes
    ControlMaster auto
    ControlPath ~/.ssh/sockets/%r@%h:%p
    ControlPersist 10m

# Specific managed nodes
Host web*
    User ansible
    Port 22
    IdentityFile ~/.ssh/ansible_key
    StrictHostKeyChecking no
    UserKnownHostsFile /dev/null

Host web1
    HostName 192.168.1.10

Host web2
    HostName 192.168.1.11

Host db*
    User dbadmin
    Port 2222
    IdentityFile ~/.ssh/ansible_db_key

Host db1
    HostName 192.168.1.20

# Jump host / Bastion configuration
Host private-server
    HostName 10.0.1.50
    ProxyJump bastion.example.com
    User ansible
```

SSH Config Directives Explained:

- **ServerAliveInterval**: Send keepalive packets every N seconds
- **ServerAliveCountMax**: Disconnect after N missed keepalives
- **Compression**: Compress data (useful for slow links)
- **ControlMaster**: Enable connection sharing
- **ControlPath**: Location for control sockets
- **ControlPersist**: Keep connections open for N time
- **StrictHostKeyChecking**: Verify host keys (use with caution)
- **IdentityFile**: Path to private key
- **ProxyJump**: Connect through bastion/jump host

Server-Side SSH Configuration

On managed nodes, edit `/etc/ssh/sshd_config`:

```
# Good security settings for Ansible managed nodes
Port 22
PermitRootLogin no
PubkeyAuthentication yes
PasswordAuthentication no
AuthorizedKeysFile .ssh/authorized_keys
ChallengeResponseAuthentication no
UsePAM yes
X11Forwarding no
PrintMotd no
AcceptEnv LANG LC_*
Subsystem sftp /usr/lib/openssh/sftp-server

# Performance settings
UseDNS no
MaxStartups 10:30:100
MaxSessions 100

# Timeout settings
ClientAliveInterval 60
ClientAliveCountMax 3
```

Restart SSH service after changes:

```
# Ubuntu/Debian
sudo systemctl restart sshd

# RHEL/CentOS
sudo systemctl restart sshd
```

5.5 Dedicated Ansible User Setup

Create Ansible User on Control Node

```
# Create ansible user
sudo useradd -m -s /bin/bash ansible

# Set password (optional)
sudo passwd ansible

# Generate SSH key for ansible user
sudo su - ansible
ssh-keygen -t ed25519 -C "ansible-automation"
exit
```

Create Ansible User on Managed Nodes

Manual Creation:

```
# On each managed node
sudo useradd -m -s /bin/bash ansible
sudo passwd ansible  # Set temporary password

# Configure sudo access
echo "ansible ALL=(ALL) NOPASSWD:ALL" | sudo tee /etc/sudoers.d/ansib
sudo chmod 0440 /etc/sudoers.d/ansible

# Create .ssh directory
sudo su - ansible
mkdir -p ~/.ssh
chmod 700 ~/.ssh
touch ~/.ssh/authorized_keys
chmod 600 ~/.ssh/authorized_keys
```

Automated Creation with Ansible:

```yaml
# playbook: create-ansible-user.yml
---
- name: Create Ansible user on all managed nodes
  hosts: all
  become: yes
  vars:
    ansible_user_name: ansible
    ansible_ssh_key: "{{ lookup('file', '~/.ssh/id_rsa.pub') }}"

  tasks:
    - name: Create ansible group
      group:
        name: "{{ ansible_user_name }}"
        state: present

    - name: Create ansible user
      user:
        name: "{{ ansible_user_name }}"
        group: "{{ ansible_user_name }}"
        shell: /bin/bash
        create_home: yes
        state: present

    - name: Add SSH key for ansible user
      authorized_key:
        user: "{{ ansible_user_name }}"
        key: "{{ ansible_ssh_key }}"
        state: present

    - name: Configure sudo access for ansible user
      copy:
        dest: /etc/sudoers.d/ansible
        content: "{{ ansible_user_name }} ALL=(ALL) NOPASSWD:ALL\n"
        mode: '0440'
        validate: /usr/sbin/visudo -cf %s

    - name: Ensure .ssh directory exists
      file:
        path: "/home/{{ ansible_user_name }}/.ssh"
        state: directory
```

```
        owner: "{{ ansible_user_name }}"
        group: "{{ ansible_user_name }}"
        mode: '0700'
```

Run the playbook (first time with existing credentials):

```
ansible-playbook create-ansible-user.yml \
  -u current_admin_user \
  --ask-pass \
  --ask-become-pass
```

5.6 Passwordless Sudo Configuration

Configure sudoers for Ansible User

```
# Edit sudoers file safely
sudo visudo

# Add this line
ansible ALL=(ALL) NOPASSWD:ALL

# Or create separate file (recommended)
sudo visudo -f /etc/sudoers.d/ansible

# Add content:
# Ansible automation user
ansible ALL=(ALL) NOPASSWD:ALL

# Verify file
sudo visudo -c -f /etc/sudoers.d/ansible
```

Security Considerations:

- Use `NOPASSWD` only for trusted automation users
- Restrict to specific commands if possible
- Use separate file in `/etc/sudoers.d/` for better management
- Set proper permissions (0440)

Restricted sudo example:

```
# Allow only specific commands without password
ansible ALL=(ALL) NOPASSWD: /bin/systemctl, /usr/bin/yum, /usr/bin/ap

# Allow all except dangerous commands
ansible ALL=(ALL) NOPASSWD: ALL, !/bin/su, !/usr/bin/passwd
```

5.7 SSH Agent and Key Management

Using SSH Agent

```
# Start SSH agent
eval "$(ssh-agent -s)"

# Add key to agent
ssh-add ~/.ssh/id_rsa

# Add key with passphrase
ssh-add ~/.ssh/id_rsa_with_passphrase

# List loaded keys
ssh-add -l

# Remove key from agent
ssh-add -d ~/.ssh/id_rsa

# Remove all keys
ssh-add -D
```

Persistent SSH Agent

Add to `~/.bashrc` or `~/.bash_profile`:

```
# Auto-start SSH agent
if [ -z "$SSH_AUTH_SOCK" ]; then
    eval "$(ssh-agent -s)"
    ssh-add ~/.ssh/id_rsa 2>/dev/null
fi
```

SSH Agent Forwarding

Allows using local SSH keys on remote hosts.

Enable in SSH config:

```
Host *
    ForwardAgent yes
```

Use with Ansible:

```
# Run with agent forwarding
ansible-playbook site.yml --ssh-common-args='-o ForwardAgent=yes'
```

Security Warning: Only use agent forwarding on trusted hosts!

5.8 Ansible Vault for SSH Key Passphrases

If SSH keys have passphrases, use Ansible Vault to store them securely.

```
# Create vault file with passphrase
ansible-vault create ssh_passphrase.yml

# Content:
ssh_key_passphrase: "YourSecurePassphrase"

# Use in playbook
- name: Add SSH key with passphrase
  shell: |
    eval "$(ssh-agent -s)"
    echo "{{ ssh_key_passphrase }}" | ssh-add ~/.ssh/id_rsa
  no_log: true
```

5.9 Multi-Key Management

Manage different keys for different environments or purposes.

SSH Config Approach:

```
Host *.prod.example.com
    IdentityFile ~/.ssh/ansible_prod_key

Host *.dev.example.com
    IdentityFile ~/.ssh/ansible_dev_key

Host *.test.example.com
    IdentityFile ~/.ssh/ansible_test_key
```

Ansible Inventory Approach:

```
[production]
prod1.example.com ansible_ssh_private_key_file=~/.ssh/ansible_prod_ke
prod2.example.com ansible_ssh_private_key_file=~/.ssh/ansible_prod_ke

[development]
dev1.example.com ansible_ssh_private_key_file=~/.ssh/ansible_dev_key
dev2.example.com ansible_ssh_private_key_file=~/.ssh/ansible_dev_key
```

Playbook Approach:

```
- name: Deploy to production
  hosts: production
  vars:
    ansible_ssh_private_key_file: "~/.ssh/ansible_prod_key"
  tasks:
    # ...
```

5.10 Bastion/Jump Host Configuration

Access private network servers through a bastion host.

Method 1: SSH ProxyJump (Modern)

```
# In ~/.ssh/config
Host bastion
    HostName bastion.example.com
    User ansible
    IdentityFile ~/.ssh/bastion_key

Host 10.0.*
    ProxyJump bastion
    User ansible
    IdentityFile ~/.ssh/internal_key
```

Method 2: ProxyCommand (Legacy)

```
# In ~/.ssh/config
Host internal*
    HostName %h.internal.example.com
    User ansible
    ProxyCommand ssh -W %h:%p bastion.example.com
```

Method 3: Ansible Configuration

```
# ansible.cfg
[ssh_connection]
ssh_args = -o ProxyCommand="ssh -W %h:%p -q bastion.example.com"
```

Ansible Inventory with Bastion:

```
# inventory.yml
all:
  vars:
    ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastio
  children:
    private_servers:
      hosts:
        internal1:
          ansible_host: 10.0.1.10
        internal2:
          ansible_host: 10.0.1.11
```

5.11 Troubleshooting SSH Issues

Common Issue 1: Permission Denied

Symptoms:

```
Permission denied (publickey,password)
```

Diagnosis:

```
# Test SSH with verbose output
ssh -vvv username@managed-node

# Check key permissions on control node
ls -la ~/.ssh/id_rsa
# Should be: -rw------- (600)

# Check authorized_keys on managed node
ssh username@managed-node "ls -la ~/.ssh/authorized_keys"
# Should be: -rw------- (600)

# Check .ssh directory permissions
# Should be: drwx------ (700)
```

Solutions:

```
# Fix key permissions on control node
chmod 600 ~/.ssh/id_rsa
chmod 644 ~/.ssh/id_rsa.pub
chmod 700 ~/.ssh

# Fix on managed node
ssh username@managed-node
chmod 700 ~/.ssh
chmod 600 ~/.ssh/authorized_keys
```

Common Issue 2: Host Key Verification Failed

Symptoms:

```
Host key verification failed.
```

Solutions:

```
# Option 1: Add host to known_hosts
ssh-keyscan -H managed-node >> ~/.ssh/known_hosts

# Option 2: Disable host key checking (use cautiously)
# In ansible.cfg
[defaults]
host_key_checking = False

# Option 3: Clear old host key
ssh-keygen -R managed-node
```

Common Issue 3: Too Many Authentication Failures

Symptoms:

```
Received disconnect: Too many authentication failures
```

Solution:

```
# Specify exact key to use
ssh -i ~/.ssh/specific_key username@managed-node

# Or in ansible.cfg
[defaults]
private_key_file = ~/.ssh/specific_key

# Or limit keys offered by SSH
# In ~/.ssh/config
Host *
    IdentitiesOnly yes
    IdentityFile ~/.ssh/ansible_key
```

Common Issue 4: SSH Connection Timeout

Symptoms:

```
ssh: connect to host managed-node port 22: Connection timed out
```

Diagnosis:

```
# Check if host is reachable
ping managed-node

# Check if SSH port is open
telnet managed-node 22
# Or
nc -zv managed-node 22

# Check firewall rules
sudo iptables -L -n | grep 22
sudo firewall-cmd --list-all
```

Solutions:

```
# Allow SSH through firewall on managed node
sudo firewall-cmd --permanent --add-service=ssh
sudo firewall-cmd --reload

# Or allow specific IP
sudo firewall-cmd --permanent --add-rich-rule='
  rule family="ipv4"
  source address="192.168.1.100"
  port port="22" protocol="tcp" accept'
sudo firewall-cmd --reload
```

5.12 SSH Security Best Practices

- **Use Key-Based Authentication**

  `bash

  # Disable password authentication in /etc/ssh/sshd_config

PasswordAuthentication no

ChallengeResponseAuthentication no

```

- **Use Strong Keys**

  ```bash
  # Generate ED25519 (recommended)
  ssh-keygen -t ed25519 -a 100

  # Or RSA with 4096 bits
  ssh-keygen -t rsa -b 4096
  ```

- **Protect Private Keys**

  ```bash
  # Use passphrase on private keys
  ssh-keygen -t ed25519 -C "ansible" -N "your-strong-passphrase"

  # Store passphrases in Ansible Vault
  ```

- **Disable Root Login**

  ```bash
  # In /etc/ssh/sshd_config
  PermitRootLogin no
  ```

- **Limit SSH Access**
  ```

```bash
# Allow only specific users
AllowUsers ansible deploy admin

# Or allow only specific groups
AllowGroups sshusers ansible-group
```

- **Change Default SSH Port** (optional)

```bash
# In /etc/ssh/sshd_config
Port 2222

# Update Ansible inventory
[webservers]
web1.example.com ansible_port=2222
```

- **Use Fail2Ban**

```bash
# Install fail2ban to block brute-force attempts
sudo apt install fail2ban
sudo systemctl enable fail2ban
sudo systemctl start fail2ban
```

- **Regular Key Rotation**

```bash
```

```
# Rotate SSH keys annually or after personnel changes

# Generate new key

ssh-keygen -t ed25519 -f ~/.ssh/id_ed25519_2024


# Deploy new key

ansible all -m authorized_key \

-a "user=ansible key='{{ lookup('file', '~/.ssh/id_ed25519_2024.pub') }}'"


# Test new key

ansible all -m ping --private-key ~/.ssh/id_ed25519_2024


# Remove old key

ansible all -m authorized_key \

-a "user=ansible key='{{ lookup('file', '~/.ssh/id_rsa.pub') }}' state=absent"
```

`

5.13 Hands-On Lab: SSH Setup for Ansible

Objective: Set up secure SSH authentication for Ansible automation

Lab Environment:

- 1 Control node
- 2 Managed nodes

Task 1: Generate and Deploy SSH Keys

```
# On control node
ssh-keygen -t ed25519 -C "ansible-lab"

# Copy to managed nodes
ssh-copy-id user@node1
ssh-copy-id user@node2

# Test connectivity
ssh user@node1 "hostname"
ssh user@node2 "hostname"
```

Task 2: Create Dedicated Ansible User

```
# Create user creation playbook
cat > create-ansible-user.yml << 'EOF'
---
- name: Create Ansible user
  hosts: all
  become: yes
  tasks:
    - name: Create ansible user
      user:
        name: ansible
        shell: /bin/bash
        create_home: yes
        state: present

    - name: Add SSH key
      authorized_key:
        user: ansible
        key: "{{ lookup('file', '~/.ssh/id_ed25519.pub') }}"

    - name: Configure sudo
      copy:
        dest: /etc/sudoers.d/ansible
        content: "ansible ALL=(ALL) NOPASSWD:ALL\n"
        mode: '0440'
        validate: /usr/sbin/visudo -cf %s
EOF

# Run playbook
ansible-playbook create-ansible-user.yml -u current_user --ask-pass -
```

Task 3: Test Ansible with New User

```
# Update ansible.cfg
cat > ansible.cfg << 'EOF'
[defaults]
inventory = inventory.ini
remote_user = ansible
host_key_checking = False

[privilege_escalation]
become = True
become_method = sudo
become_user = root
EOF

# Test
ansible all -m ping
ansible all -m command -a "whoami"
ansible all -m command -a "sudo whoami"
```

Task 4: Configure SSH Multiplexing

```
# Create SSH config
mkdir -p ~/.ssh/sockets

cat >> ~/.ssh/config << 'EOF'
Host node*
    User ansible
    ControlMaster auto
    ControlPath ~/.ssh/sockets/%r@%h:%p
    ControlPersist 10m
    ServerAliveInterval 60
EOF

# Test performance improvement
time ansible all -m ping
time ansible all -m ping  # Second run should be faster
```

Expected Outcomes:

- Password-less SSH access to all managed nodes
- Dedicated ansible user with sudo privileges
- Improved performance with SSH multiplexing
- Secure, auditable automation setup

[Continuing with sections 6-17 in the next part due to length...]

6. Inventory Management

6.1 Understanding Ansible Inventory

The inventory is the core component that defines which hosts Ansible will manage. It's essentially a list of managed nodes organized in groups, along with variables that describe how to connect to and manage those hosts.

Inventory Purposes:

- Define managed hosts
- Group related hosts together
- Set host and group variables
- Define connection parameters
- Enable host patterns for targeting

6.2 Static Inventory Files

Static inventory files are simple text files that manually list all managed hosts.

INI Format Inventory

Basic Example:

```
# /etc/ansible/hosts

# Ungrouped hosts (not recommended)
standalone-server.example.com

# Web servers group
[webservers]
web1.example.com
web2.example.com
web3.example.com

# Database servers group
[databases]
db1.example.com
db2.example.com ansible_port=3306

# Application servers with variables
[appservers]
app1.example.com ansible_host=192.168.1.10
app2.example.com ansible_host=192.168.1.11 ansible_user=appuser
```

Using IP Addresses:

```
[webservers]
192.168.1.10
192.168.1.11
192.168.1.12

# With aliases
[databases]
db1 ansible_host=10.0.1.20
db2 ansible_host=10.0.1.21
```

Range Notation:

```
# Numeric ranges
[webservers]
web[01:10].example.com
# Expands to: web01, web02, ... web10

# Alphabetic ranges
[databases]
db-[a:f].example.com
# Expands to: db-a, db-b, ... db-f

# IP ranges
[loadbalancers]
192.168.1.[10:20]
```

Nested Groups (Groups of Groups):

```
[webservers]
web1.example.com
web2.example.com

[databases]
db1.example.com
db2.example.com

[monitoring]
monitor1.example.com

# Parent group containing other groups
[production:children]
webservers
databases

[infrastructure:children]
production
monitoring

# Variables for parent groups
[production:vars]
env=production
backup_enabled=true

[infrastructure:vars]
ntp_server=ntp.example.com
```

YAML Format Inventory

More structured and readable for complex inventories.

Basic Example:

```yaml
# inventory.yml
all:
  hosts:
    standalone-server:
      ansible_host: 192.168.1.100

  children:
    webservers:
      hosts:
        web1:
          ansible_host: 192.168.1.10
        web2:
          ansible_host: 192.168.1.11
        web3:
          ansible_host: 192.168.1.12
      vars:
        http_port: 80
        https_port: 443

    databases:
      hosts:
        db1:
          ansible_host: 192.168.1.20
          db_port: 3306
        db2:
          ansible_host: 192.168.1.21
          db_port: 3306
      vars:
        db_engine: mysql
        db_version: "8.0"

    production:
      children:
        webservers:
        databases:
      vars:
        env: production
        backup_schedule: "0 2 * * *"
```

Complex Example with Multiple Layers:

```
all:
  vars:
    ansible_user: ansible
    ansible_ssh_private_key_file: ~/.ssh/ansible_key

  children:
    datacenters:
      children:
        dc1:
          children:
            dc1_web:
              hosts:
                dc1-web1:
                  ansible_host: 10.1.1.10
                dc1-web2:
                  ansible_host: 10.1.1.11
              vars:
                datacenter: dc1
                rack: A1

            dc1_db:
              hosts:
                dc1-db1:
                  ansible_host: 10.1.2.10
                dc1-db2:
                  ansible_host: 10.1.2.11
              vars:
                datacenter: dc1
                rack: A2

        dc2:
          children:
            dc2_web:
              hosts:
                dc2-web1:
                  ansible_host: 10.2.1.10
              vars:
                datacenter: dc2
                rack: B1
```

```
        dc2_db:
          hosts:
            dc2-db1:
              ansible_host: 10.2.2.10
          vars:
            datacenter: dc2
            rack: B2

  environments:
    children:
      production:
        children:
          dc1:
          dc2:
        vars:
          env: prod
          monitoring_enabled: true

      staging:
        hosts:
          staging1:
            ansible_host: 192.168.100.10
        vars:
          env: staging
          monitoring_enabled: false
```

6.3 Inventory Variables

Host Variables

Variables specific to individual hosts.

Inline (in inventory file):

```
[webservers]
web1.example.com http_port=8080 max_clients=200
web2.example.com http_port=8081 max_clients=300
```

In separate file (`host_vars/web1.example.com.yml`):

```
---
http_port: 8080
max_clients: 200
ssl_enabled: true
ssl_cert_path: /etc/ssl/certs/web1.crt
ssl_key_path: /etc/ssl/private/web1.key
admin_email: admin@web1.example.com
```

Group Variables

Variables shared by all hosts in a group.

Inline (in inventory file):

```
[webservers:vars]
http_port=80
https_port=443
document_root=/var/www/html
```

In separate file (`group_vars/webservers.yml`):

```
---
http_port: 80
https_port: 443
document_root: /var/www/html
log_level: info
max_connections: 1000

apache_modules:
  - mod_ssl
  - mod_rewrite
  - mod_headers

vhosts:
  - name: site1.example.com
    port: 80
  - name: site2.example.com
    port: 80
```

Special Groups

all: Contains every host

```
# group_vars/all.yml
---
ntp_server: ntp.example.com
dns_servers:
  - 8.8.8.8
  - 8.8.4.4
timezone: America/New_York
```

ungrouped: Hosts not in any group

```
# group_vars/ungrouped.yml
---
ansible_user: admin
special_handling: true
```

6.4 Connection Variables

Common Ansible Connection Variables:

```
# SSH connection
ansible_host: 192.168.1.10                    # Real hostname/IP
ansible_port: 22                              # SSH port
ansible_user: ansible                         # SSH username
ansible_ssh_private_key_file: ~/.ssh/id_rsa   # SSH key path
ansible_ssh_common_args: '-o StrictHostKeyChecking=no'  # SSH args

# Connection type
ansible_connection: ssh       # ssh, local, docker, etc.

# Privilege escalation
ansible_become: yes           # Enable privilege escalation
ansible_become_method: sudo   # sudo, su, pbrun, etc.
ansible_become_user: root     # User to become
ansible_become_password: encrypted_pass  # Become password (use vault

# Python interpreter
ansible_python_interpreter: /usr/bin/python3

# Windows-specific
ansible_connection: winrm
ansible_winrm_server_cert_validation: ignore
ansible_winrm_transport: ntlm
```

Example Inventory with Connection Variables:

```
[webservers]
web1 ansible_host=192.168.1.10 ansible_user=webadmin
web2 ansible_host=192.168.1.11 ansible_port=2222

[databases]
db1 ansible_host=10.0.1.20 ansible_python_interpreter=/usr/bin/python

[windows_servers]
win1 ansible_host=192.168.1.100 ansible_connection=winrm ansible_user

[containers]
docker1 ansible_connection=docker ansible_user=root

[localhost]
control ansible_connection=local
```

6.5 Dynamic Inventory

Dynamic inventory scripts query external sources to generate inventory at runtime.

AWS EC2 Dynamic Inventory

Install AWS collection:

```
ansible-galaxy collection install amazon.aws
```

Create inventory file (`aws_ec2.yml`):

```
---
plugin: amazon.aws.aws_ec2

regions:
  - us-east-1
  - us-west-2

filters:
  instance-state-name: running
  tag:Environment:
    - production
    - staging

keyed_groups:
  # Group by instance type
  - key: instance_type
    prefix: instance_type

  # Group by tags
  - key: tags.Environment
    prefix: env

  - key: tags.Role
    prefix: role

compose:
  ansible_host: public_ip_address
  ansible_user: ec2-user

hostnames:
  - tag:Name
  - private-ip-address
```

Use dynamic inventory:

```
# List all hosts from AWS
ansible-inventory -i aws_ec2.yml --list

# Use in playbook
ansible-playbook -i aws_ec2.yml site.yml

# Test specific group
ansible -i aws_ec2.yml env_production -m ping
```

Azure Dynamic Inventory

```
# azure_rm.yml
---
plugin: azure.azcollection.azure_rm

include_vm_resource_groups:
  - my-resource-group

auth_source: auto

keyed_groups:
  - prefix: tag
    key: tags
  - prefix: location
    key: location

conditional_groups:
  webservers: "'web' in name"
  databases: "'db' in name"
```

GCP Dynamic Inventory

```
# gcp_compute.yml
---
plugin: google.cloud.gcp_compute

projects:
  - my-gcp-project

filters:
  - status = RUNNING

keyed_groups:
  - key: labels.environment
    prefix: env
  - key: labels.role
    prefix: role
  - key: zone
    prefix: zone

compose:
  ansible_host: networkInterfaces[0].accessConfigs[0].natIP
```

Custom Dynamic Inventory Script

Python script that outputs JSON inventory.

```
#!/usr/bin/env python3
# dynamic_inventory.py

import json
import sys

def get_inventory():
    """Generate inventory from custom source"""
    inventory = {
        "_meta": {
            "hostvars": {
                "host1": {
                    "ansible_host": "192.168.1.10",
                    "ansible_user": "ansible"
                },
                "host2": {
                    "ansible_host": "192.168.1.11",
                    "ansible_user": "ansible"
                }
            }
        },
        "webservers": {
            "hosts": ["host1", "host2"],
            "vars": {
                "http_port": 80
            }
        },
        "all": {
            "children": ["webservers"]
        }
    }
    return inventory

def get_host_vars(host):
    """Return variables for specific host"""
    inventory = get_inventory()
    return inventory["_meta"]["hostvars"].get(host, {})

if __name__ == "__main__":
    if len(sys.argv) == 2 and sys.argv[1] == "--list":
```

```
        print(json.dumps(get_inventory(), indent=2))
    elif len(sys.argv) == 3 and sys.argv[1] == "--host":
        print(json.dumps(get_host_vars(sys.argv[2]), indent=2))
    else:
        print("Usage: {} --list or {} --host <hostname>".format(
            sys.argv[0], sys.argv[0]))
        sys.exit(1)
```

Make executable and use:

```
chmod +x dynamic_inventory.py

# Test
./dynamic_inventory.py --list
./dynamic_inventory.py --host host1

# Use with Ansible
ansible-playbook -i dynamic_inventory.py site.yml
```

6.6 Host and Group Patterns

Patterns allow you to target specific hosts or groups.

Basic Patterns:

```
# All hosts
ansible all -m ping

# Single host
ansible web1.example.com -m ping

# Single group
ansible webservers -m ping

# Multiple groups (union)
ansible webservers:databases -m ping

# Intersection (hosts in both groups)
ansible 'webservers:&production' -m ping

# Exclusion (in webservers but not in staging)
ansible 'webservers:!staging' -m ping

# Wildcard
ansible 'web*.example.com' -m ping

# Range
ansible 'web[01:10]' -m ping

# Multiple patterns
ansible 'web1:web2:db1' -m ping

# Complex pattern
ansible 'webservers:&production:!maintenance' -m ping
```

Regular Expressions:

```
# Hosts starting with 'web'
ansible '~web.*' -m ping

# Hosts ending with '.com'
ansible '~.*\.com' -m ping
```

Group Selections:

```
# All hosts in nested groups
ansible 'production' -m ping  # includes all children

# Specific nested path
ansible 'datacenters:dc1:dc1_web' -m ping
```

6.7 Inventory Best Practices

1. Logical Organization

```
inventory/
├── production/
│   ├── hosts.ini
│   ├── group_vars/
│   │   ├── all.yml
│   │   ├── webservers.yml
│   │   └── databases.yml
│   └── host_vars/
│       ├── web1.yml
│       └── db1.yml
├── staging/
│   ├── hosts.ini
│   └── group_vars/
│       └── all.yml
└── development/
    ├── hosts.ini
    └── group_vars/
        └── all.yml
```

2. Use Meaningful Group Names

```
# Good - descriptive and clear
[frontend_webservers]
[backend_api_servers]
[payment_processing_db]
[customer_data_cache]

# Bad - vague
[group1]
[servers]
[stuff]
```

3. Separate Environments

```
# Don't mix environments in one inventory
# Instead, use separate files:

# inventory/production.ini
[prod_web]
prod-web1.example.com
prod-web2.example.com

# inventory/staging.ini
[staging_web]
staging-web1.example.com
```

4. Use Group Variables Wisely

```
# group_vars/all.yml - Variables for ALL hosts
---
ntp_server: ntp.example.com
company_name: "Example Corp"

# group_vars/webservers.yml - Web-specific
---
http_port: 80
document_root: /var/www/html

# group_vars/production.yml - Prod-specific
---
backup_enabled: true
monitoring_level: verbose
```

5. Version Control

```
# Keep inventory in Git
git init
git add inventory/ group_vars/ host_vars/
git commit -m "Initial inventory setup"

# .gitignore sensitive files
echo "*.vault" >> .gitignore
echo "*.secret" >> .gitignore
```

6.8 Inventory Plugins

Modern way to generate dynamic inventory.

List available plugins:

```
ansible-doc -t inventory -l
```

Common inventory plugins:

- `host_list`: Comma-separated host list
- `yaml`: YAML inventory format
- `ini`: INI inventory format
- `script`: Execute inventory script
- `auto`: Automatically detect inventory format

- `aws_ec2`: Amazon EC2 instances
- `azure_rm`: Azure virtual machines
- `gcp_compute`: Google Cloud instances
- `vmware_vm_inventory`: VMware VMs
- `constructed`: Construct groups and variables

Enable plugins in ansible.cfg:

```
[inventory]
enable_plugins = host_list, script, auto, yaml, ini, constructed, aws
```

6.9 Hands-On Lab: Inventory Management

Objective: Create and manage complex inventory structures

Task 1: Create Multi-Environment Inventory

```
# Create structure
mkdir -p ~/ansible-inventory-lab/{production,staging,development}
mkdir -p ~/ansible-inventory-lab/{group_vars,host_vars}

# Production inventory
cat > ~/ansible-inventory-lab/production/hosts.ini << 'EOF'
[webservers]
prod-web[01:03].example.com

[databases]
prod-db[01:02].example.com

[loadbalancers]
prod-lb01.example.com

[production:children]
webservers
databases
loadbalancers
EOF

# Production variables
cat > ~/ansible-inventory-lab/group_vars/production.yml << 'EOF'
---
env: production
backup_enabled: true
monitoring_enabled: true
log_level: warning
EOF

# Web servers variables
cat > ~/ansible-inventory-lab/group_vars/webservers.yml << 'EOF'
---
http_port: 80
https_port: 443
max_connections: 1000
keepalive_timeout: 65
EOF
```

Task 2: Test Inventory

```
cd ~/ansible-inventory-lab

# List all hosts
ansible-inventory -i production/hosts.ini --list

# List hosts in specific group
ansible-inventory -i production/hosts.ini --graph

# Test host pattern matching
ansible -i production/hosts.ini 'prod-web*' --list-hosts
ansible -i production/hosts.ini 'webservers:&production' --list-hosts
```

Task 3: Create Dynamic Inventory

```
# Create simple dynamic inventory script
cat > ~/ansible-inventory-lab/dynamic_hosts.py << 'EOF'
#!/usr/bin/env python3
import json
import sys

def main():
    inventory = {
        "dynamic_web": {
            "hosts": ["dyn-web1", "dyn-web2"],
            "vars": {"ansible_user": "ansible"}
        },
        "_meta": {
            "hostvars": {
                "dyn-web1": {"ansible_host": "192.168.1.50"},
                "dyn-web2": {"ansible_host": "192.168.1.51"}
            }
        }
    }

    if len(sys.argv) == 2 and sys.argv[1] == '--list':
        print(json.dumps(inventory, indent=2))
    elif len(sys.argv) == 3 and sys.argv[1] == '--host':
        print(json.dumps(inventory['_meta']['hostvars'].get(sys.argv[
    else:
        print("{}")

if __name__ == '__main__':
    main()
EOF

chmod +x ~/ansible-inventory-lab/dynamic_hosts.py

# Test
./dynamic_hosts.py --list
ansible-inventory -i dynamic_hosts.py --list
```

Task 4: Combine Static and Dynamic Inventory

```
# Create inventory directory
mkdir -p ~/ansible-inventory-lab/combined

# Copy static inventory
cp production/hosts.ini combined/

# Link dynamic script
ln -s ../dynamic_hosts.py combined/

# Ansible will automatically merge them
ansible-inventory -i combined/ --list
ansible -i combined/ all --list-hosts
```

Expected Outcomes:

- Understand inventory structures
- Master host patterns
- Create dynamic inventory
- Combine multiple inventory sources

---

7. YAML Syntax and Jinja2 Templating

7.1 YAML Fundamentals

YAML (YAML Ain't Markup Language) is a human-readable data serialization language used extensively in Ansible for playbooks, variables, and inventory files.

Key Characteristics:

- Indentation-based structure (use spaces, not tabs!)
- Case-sensitive
- File extension: `.yml` or `.yaml`
- Human-readable and writable

7.2 YAML Basic Syntax

Scalars (Simple Values)

```yaml
# Strings
name: Ansible Automation
title: "DevOps Engineer"
description: 'This is a description'

# Numbers
port: 8080
count: 42
price: 19.99

# Booleans
enabled: true
active: yes       # yes, on, true
disabled: false
inactive: no      # no, off, false

# Null values
database: null
database: ~       # Also represents null

# Multi-line strings
description: |
  This is a multi-line string.
  Each line is preserved with line breaks.
  Useful for scripts or long text.

summary: >
  This is a folded string.
  Multiple lines are folded into
  a single line with spaces.
```

Lists (Arrays)

```yaml
# Inline format
packages: [httpd, nginx, mysql]

# Block format (more readable)
packages:
  - httpd
  - nginx
  - mysql

# List of complex items
servers:
  - name: web1
    ip: 192.168.1.10
  - name: web2
    ip: 192.168.1.11

# Nested lists
datacenter:
  - location: US-East
    servers:
      - web1
      - web2
  - location: US-West
    servers:
      - web3
      - web4
```

Dictionaries (Key-Value Pairs)

```yaml
# Simple dictionary
user:
  name: john
  age: 30
  role: admin

# Inline dictionary format
user: {name: john, age: 30, role: admin}

# Nested dictionaries
company:
  name: Example Corp
  location:
    city: New York
    country: USA
  employees:
    count: 100
    departments:
      - IT
      - HR
      - Sales
```

7.3 YAML in Ansible Playbooks

Basic Playbook Structure

```yaml
---
# Three dashes start a YAML document
- name: This is a play
  hosts: webservers
  become: yes
  vars:
    http_port: 80

  tasks:
    - name: Install Apache
      yum:
        name: httpd
        state: present

    - name: Start Apache service
      service:
        name: httpd
        state: started
        enabled: yes
```

Complex Playbook Example

```yaml
---
- name: Configure Web Servers
  hosts: webservers
  become: yes
  gather_facts: yes

  vars:
    apache_packages:
      - httpd
      - mod_ssl

    apache_service: httpd

    vhosts:
      - name: site1.example.com
        port: 80
        docroot: /var/www/site1
      - name: site2.example.com
        port: 80
        docroot: /var/www/site2

  pre_tasks:
    - name: Update package cache
      yum:
        update_cache: yes

  tasks:
    - name: Install Apache packages
      yum:
        name: "{{ item }}"
        state: present
      loop: "{{ apache_packages }}"

    - name: Create document roots
      file:
        path: "{{ item.docroot }}"
        state: directory
        owner: apache
        group: apache
        mode: '0755'
```

```
      loop: "{{ vhosts }}"

    - name: Copy virtual host configurations
      template:
        src: vhost.conf.j2
        dest: "/etc/httpd/conf.d/{{ item.name }}.conf"
      loop: "{{ vhosts }}"
      notify: Restart Apache

  handlers:
    - name: Restart Apache
      service:
        name: "{{ apache_service }}"
        state: restarted
```

7.4 Common YAML Mistakes and Solutions

Mistake 1: Using Tabs Instead of Spaces

```
# WRONG - uses tabs (causes errors)
tasks:
	- name: Install package

# CORRECT - uses spaces
tasks:
  - name: Install package
```

Mistake 2: Incorrect Indentation

```
# WRONG - inconsistent indentation
tasks:
- name: Task 1
  yum:
   name: httpd
    state: present

# CORRECT - consistent 2-space indentation
tasks:
  - name: Task 1
    yum:
      name: httpd
      state: present
```

Mistake 3: Unquoted Special Characters

```
# WRONG - special characters need quoting
message: Hello: World
path: C:\Users\ansible

# CORRECT
message: "Hello: World"
path: 'C:\Users\ansible'
```

Mistake 4: Mixing List Formats

```
# WRONG - inconsistent list format
packages:
  - httpd
  - nginx
  nginx-mod-ssl

# CORRECT
packages:
  - httpd
  - nginx
  - nginx-mod-ssl
```

7.5 YAML Validation

```
# Validate YAML syntax
python3 -c 'import yaml, sys; yaml.safe_load(sys.stdin)' < playbook.y

# Use yamllint tool
pip install yamllint
yamllint playbook.yml

# Ansible syntax check
ansible-playbook --syntax-check playbook.yml

# Ansible lint (more comprehensive)
pip install ansible-lint
ansible-lint playbook.yml
```

yamllint configuration (`.yamllint`):

```
---
extends: default

rules:
  line-length:
    max: 120
    level: warning
  indentation:
    spaces: 2
    indent-sequences: true
  comments:
    min-spaces-from-content: 2
```

7.6 Jinja2 Templating Basics

Jinja2 is a powerful templating engine used in Ansible for dynamic content generation.

Common Uses:

- Configuration file templates
- Dynamic variable substitution
- Conditional content
- Loops in templates
- Filters for data manipulation

Basic Variable Substitution

```
{# This is a comment #}

{# Variable substitution #}
Server name: {{ ansible_hostname }}
IP Address: {{ ansible_default_ipv4.address }}
Environment: {{ environment }}

{# Accessing nested variables #}
Database host: {{ database.host }}
Database port: {{ database.port }}

{# List access #}
First server: {{ servers[0] }}
Last server: {{ servers[-1] }}
```

Jinja2 in Ansible Playbooks

```
---
- name: Use Jinja2 expressions
  hosts: webservers
  vars:
    app_name: myapp
    app_port: 8080

  tasks:
    - name: Debug with Jinja2
      debug:
        msg: "Application {{ app_name }} runs on port {{ app_port }}"

    - name: Set fact with Jinja2
      set_fact:
        app_url: "http://{{ ansible_hostname }}:{{ app_port }}"

    - name: Conditional with Jinja2
      debug:
        msg: "This is a production server"
      when: "'production' in inventory_hostname"
```

7.7 Jinja2 Filters

Filters modify variables in templates.

```yaml
---
- name: Demonstrate Jinja2 filters
  hosts: localhost
  vars:
    username: "JohnDoe"
    servers:
      - web1
      - web2
      - web3
    numbers: [1, 2, 3, 4, 5]

  tasks:
    # String filters
    - debug:
        msg: "{{ username | lower }}"        # johndoe

    - debug:
        msg: "{{ username | upper }}"        # JOHNDOE

    - debug:
        msg: "{{ username | capitalize }}"   # Johndoe

    # Default values
    - debug:
        msg: "{{ undefined_var | default('default_value') }}"

    # List filters
    - debug:
        msg: "{{ servers | length }}"        # 3

    - debug:
        msg: "{{ servers | first }}"         # web1

    - debug:
        msg: "{{ servers | last }}"          # web3

    - debug:
        msg: "{{ servers | join(', ') }}"    # web1, web2, web3

    # Math filters
```

```
    - debug:
        msg: "{{ numbers | min }}"          # 1

    - debug:
        msg: "{{ numbers | max }}"          # 5

    - debug:
        msg: "{{ numbers | sum }}"          # 15

    # Type conversion
    - debug:
        msg: "{{ '100' | int + 50 }}"       # 150

    - debug:
        msg: "{{ 100 | string }}"           # "100"

    # JSON/YAML
    - debug:
        msg: "{{ servers | to_json }}"

    - debug:
        msg: "{{ servers | to_nice_json }}"

    - debug:
        msg: "{{ servers | to_yaml }}"
```

Common Filters Reference:

```
# String manipulation
{{ variable | lower }}           # Convert to lowercase
{{ variable | upper }}           # Convert to uppercase
{{ variable | capitalize }}      # Capitalize first letter
{{ variable | title }}           # Title case
{{ variable | replace('old', 'new') }}  # Replace substring

# List operations
{{ list | length }}              # Get list length
{{ list | first }}               # First item
{{ list | last }}                # Last item
{{ list | join(', ') }}          # Join with separator
{{ list | unique }}              # Remove duplicates
{{ list | sort }}                # Sort list
{{ list | reverse }}             # Reverse order

# Math operations
{{ numbers | min }}              # Minimum value
{{ numbers | max }}              # Maximum value
{{ numbers | sum }}              # Sum all values
{{ number | abs }}               # Absolute value
{{ number | round }}             # Round number

# Defaults and optional
{{ variable | default('value') }}          # Default if undefined
{{ variable | default(omit) }}             # Omit if undefined
{{ variable | mandatory }}                 # Fail if undefined

# File paths
{{ path | basename }}            # Get filename
{{ path | dirname }}             # Get directory
{{ path | expanduser }}          # Expand ~ to home directory

# Date/Time
{{ '%Y-%m-%d' | strftime }}     # Format current date

# Encoding
{{ string | b64encode }}         # Base64 encode
{{ string | b64decode }}         # Base64 decode
{{ dict | to_json }}             # Convert to JSON
```

```
{{ dict | to_nice_json }}        # Pretty JSON
{{ dict | to_yaml }}             # Convert to YAML

# Regular expressions
{{ string | regex_replace('^www\.', '') }}
{{ string | regex_search('pattern') }}

# IP address filters
{{ ip_address | ipaddr }}        # Validate IP
{{ ip_address | ipv4 }}          # IPv4 only
{{ ip_address | ipv6 }}          # IPv6 only
```

7.8 Jinja2 Conditional Statements

```
{# Simple if statement #}
{% if environment == 'production' %}
server_type = production
{% endif %}

{# If-else #}
{% if environment == 'production' %}
server_type = production
{% else %}
server_type = non-production
{% endif %}

{# If-elif-else #}
{% if ansible_distribution == "Ubuntu" %}
package_manager = apt
{% elif ansible_distribution == "CentOS" %}
package_manager = yum
{% elif ansible_distribution == "Fedora" %}
package_manager = dnf
{% else %}
package_manager = unknown
{% endif %}

{# Complex conditions #}
{% if (ansible_distribution == "Ubuntu" and ansible_distribution_majo
# Ubuntu 20.04 specific configuration
{% endif %}

{# Testing variables #}
{% if variable is defined %}
{{ variable }}
{% endif %}

{% if list is iterable %}
# List operations
{% endif %}
```

7.9 Jinja2 Loops

```
{# Simple loop #}
{% for server in servers %}
Server {{ loop.index }}: {{ server }}
{% endfor %}

{# Loop with dictionary #}
{% for key, value in dictionary.items() %}
{{ key }} = {{ value }}
{% endfor %}

{# Loop with conditions #}
{% for user in users %}
{% if user.active %}
Active user: {{ user.name }}
{% endif %}
{% endfor %}

{# Loop variables #}
{% for item in items %}
Index: {{ loop.index }}         {# 1, 2, 3, ... #}
Index0: {{ loop.index0 }}        {# 0, 1, 2, ... #}
Reverse Index: {{ loop.revindex }}  {# n, n-1, ..., 1 #}
First: {{ loop.first }}          {# True on first iteration #}
Last: {{ loop.last }}            {# True on last iteration #}
Length: {{ loop.length }}        {# Total iterations #}
{% endfor %}
```

7.10 Jinja2 Template Example: Apache VirtualHost

Template file (`templates/vhost.conf.j2`):

```
# {{ ansible_managed }}
# Virtual Host configuration for {{ vhost.domain }}

<VirtualHost *:{{ vhost.port | default(80) }}>
    ServerName {{ vhost.domain }}
    {% if vhost.aliases is defined %}
    ServerAlias {{ vhost.aliases | join(' ') }}
    {% endif %}

    DocumentRoot {{ vhost.docroot }}

    <Directory {{ vhost.docroot }}>
        Options -Indexes +FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>

    {% if vhost.ssl_enabled | default(false) %}
    SSLEngine on
    SSLCertificateFile {{ vhost.ssl_cert }}
    SSLCertificateKeyFile {{ vhost.ssl_key }}
    {% if vhost.ssl_chain is defined %}
    SSLCertificateChainFile {{ vhost.ssl_chain }}
    {% endif %}
    {% endif %}

    ErrorLog ${APACHE_LOG_DIR}/{{ vhost.domain }}_error.log
    CustomLog ${APACHE_LOG_DIR}/{{ vhost.domain }}_access.log combine

    {% if environment == 'production' %}
    LogLevel warn
    {% else %}
    LogLevel debug
    {% endif %}
</VirtualHost>
```

Playbook using the template:

```
---
- name: Configure Apache VirtualHosts
  hosts: webservers
  become: yes
  vars:
    vhosts:
      - domain: example.com
        aliases:
          - www.example.com
        port: 80
        docroot: /var/www/example
        ssl_enabled: true
        ssl_cert: /etc/ssl/certs/example.crt
        ssl_key: /etc/ssl/private/example.key

      - domain: test.example.com
        port: 8080
        docroot: /var/www/test

  tasks:
    - name: Deploy VirtualHost configurations
      template:
        src: vhost.conf.j2
        dest: "/etc/apache2/sites-available/{{ item.domain }}.conf"
      loop: "{{ vhosts }}"
      notify: Reload Apache

  handlers:
    - name: Reload Apache
      service:
        name: apache2
        state: reloaded
```

7.11 Advanced Jinja2 Techniques

Macros (Reusable Template Functions)

```
{# Define macro #}
{% macro input(name, value='', type='text', size=20) %}
    <input type="{{ type }}" name="{{ name }}" value="{{ value }}" si
{% endmacro %}

{# Use macro #}
<form>
    {{ input('username') }}
    {{ input('password', type='password') }}
    {{ input('email', type='email', size=30) }}
</form>
```

Template Inheritance

Base template (`base.conf.j2`):

```
# {{ ansible_managed }}
# Base Configuration

{% block global_settings %}
# Global settings
timeout = 30
{% endblock %}

{% block service_config %}
# Service specific configuration
{% endblock %}

{% block logging %}
# Logging configuration
log_level = info
{% endblock %}
```

Child template (`app.conf.j2`):

```
{% extends "base.conf.j2" %}

{% block service_config %}
# Application specific settings
app_name = {{ app_name }}
app_port = {{ app_port }}
app_workers = {{ app_workers }}
{% endblock %}

{% block logging %}
{{ super() }}
log_file = /var/log/{{ app_name }}.log
{% endblock %}
```

7.12 Hands-On Lab: YAML and Jinja2

Objective: Master YAML syntax and Jinja2 templating

Task 1: Create Complex YAML Inventory

```
# inventory.yml
---
all:
  children:
    production:
      children:
        webservers:
          hosts:
            web1:
              ansible_host: 192.168.1.10
              app_port: 8080
              max_connections: 1000
            web2:
              ansible_host: 192.168.1.11
              app_port: 8080
              max_connections: 1500
          vars:
            environment: production
            backup_enabled: true
            monitoring:
              enabled: true
              interval: 60
              alerts:
                - email
                - slack

        databases:
          hosts:
            db1:
              ansible_host: 192.168.1.20
              db_port: 3306
              db_engine: mysql
          vars:
            backup_schedule: "0 2 * * *"
            replication_enabled: true
```

Task 2: Create Nginx Configuration Template

```
{# templates/nginx-site.conf.j2 #}
# {{ ansible_managed }}
# Nginx configuration for {{ site_name }}

{% set workers = ansible_processor_vcpus * 2 %}

upstream {{ site_name }}_backend {
    {% for server in backend_servers %}
    server {{ server.host }}:{{ server.port }} weight={{ server.weigh
    {% endfor %}
}

server {
    listen {{ http_port | default(80) }};
    server_name {{ site_name }} {{ server_aliases | join(' ') if serv

    root {{ document_root }};
    index index.html index.php;

    # Security headers
    {% if environment == 'production' %}
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header X-XSS-Protection "1; mode=block" always;
    {% endif %}

    # Logging
    access_log /var/log/nginx/{{ site_name }}_access.log;
    error_log /var/log/nginx/{{ site_name }}_error.log {% if environm

    # PHP handling
    {% if php_enabled | default(false) %}
    location ~ \.php$ {
        fastcgi_pass unix:/var/run/php/php{{ php_version }}-fpm.sock;
        fastcgi_index index.php;
        include fastcgi_params;
    }
    {% endif %}

    # Proxy to backend
```

```
    location /api/ {
        proxy_pass http://{{ site_name }}_backend;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

        {% if enable_caching | default(false) %}
        proxy_cache api_cache;
        proxy_cache_valid 200 {{ cache_ttl | default(60) }}m;
        {% endif %}
    }
}

{% if ssl_enabled | default(false) %}
server {
    listen 443 ssl http2;
    server_name {{ site_name }};

    ssl_certificate {{ ssl_cert_path }};
    ssl_certificate_key {{ ssl_key_path }};
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;

    # Same configuration as above
    include /etc/nginx/snippets/{{ site_name }}_common.conf;
}
{% endif %}
```

Task 3: Create Playbook Using Template

```yaml
---
- name: Deploy Nginx Configuration
  hosts: webservers
  become: yes
  vars:
    site_name: myapp
    http_port: 80
    document_root: /var/www/myapp
    php_enabled: true
    php_version: "8.1"
    environment: "{{ 'production' if 'prod' in inventory_hostname els

    backend_servers:
      - host: 127.0.0.1
        port: 9000
        weight: 2
      - host: 127.0.0.1
        port: 9001
        weight: 1

    ssl_enabled: "{{ environment == 'production' }}"
    ssl_cert_path: "/etc/ssl/certs/{{ site_name }}.crt"
    ssl_key_path: "/etc/ssl/private/{{ site_name }}.key"

  tasks:
    - name: Deploy Nginx site configuration
      template:
        src: nginx-site.conf.j2
        dest: "/etc/nginx/sites-available/{{ site_name }}.conf"
        validate: "nginx -t -c %s"
      notify: Reload Nginx

    - name: Enable site
      file:
        src: "/etc/nginx/sites-available/{{ site_name }}.conf"
        dest: "/etc/nginx/sites-enabled/{{ site_name }}.conf"
        state: link

  handlers:
    - name: Reload Nginx
```

```
        service:
          name: nginx
          state: reloaded
```

Expected Outcomes:

- Understand YAML structure and syntax
- Master Jinja2 variable substitution
- Use conditional and loops in templates
- Apply filters for data transformation
- Create production-ready configuration templates

8. Playbooks

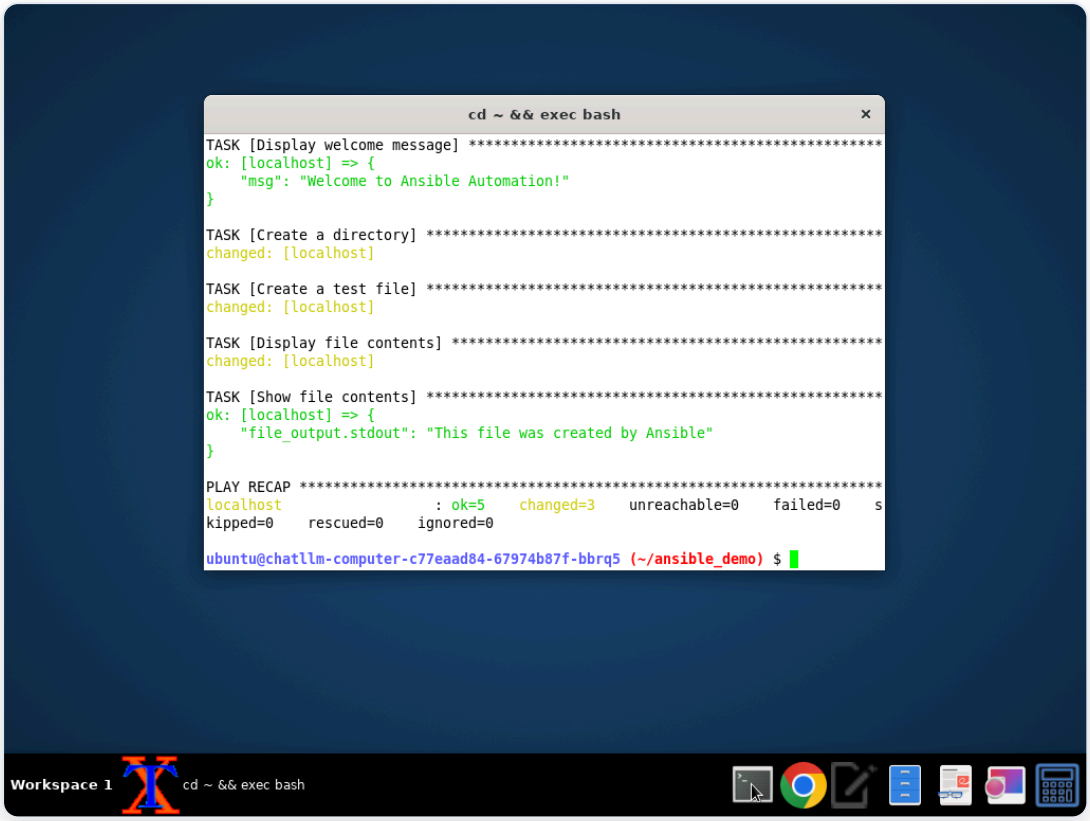
8.1 What are Ansible Playbooks?

Playbooks are Ansible's configuration, deployment, and orchestration language. They are expressed in YAML format and describe the desired state of your systems.

Key Concepts:

- **Play**: Maps a group of hosts to roles/tasks
- **Task**: A single unit of action (calls a module)
- **Module**: The actual code that performs actions
- **Handler**: Tasks triggered by notifications

8.2 Playbook Structure



Screenshot: 8.2 Playbook Structure

Basic anatomy:

```yaml
---
# Document start marker

- name: Play 1 - Configure web servers
  hosts: webservers              # Which hosts to target
  become: yes                    # Privilege escalation
  gather_facts: yes              # Collect system information
  vars:                          # Play-level variables
    http_port: 80
  vars_files:                    # External variable files
    - vars/main.yml
  pre_tasks:                     # Run before tasks
    - name: Update cache
      apt:
        update_cache: yes
  roles:                         # Include roles
    - common
    - webserver
  tasks:                         # Main tasks
    - name: Install Apache
      apt:
        name: apache2
        state: present
  post_tasks:                    # Run after tasks
    - name: Verify service
      uri:
        url: http://localhost
  handlers:                      # Event handlers
    - name: Restart Apache
      service:
        name: apache2
        state: restarted

- name: Play 2 - Configure databases
  hosts: databases
  # ... more plays
```

8.3 Your First Playbook

Simple example (`first-playbook.yml`):

```
---
- name: My First Playbook
  hosts: localhost
  connection: local
  gather_facts: no

  tasks:
    - name: Print a message
      debug:
        msg: "Hello, Ansible!"

    - name: Get current date
      command: date
      register: current_date

    - name: Display date
      debug:
        var: current_date.stdout
```

Run the playbook:

```
ansible-playbook first-playbook.yml

# With verbose output
ansible-playbook first-playbook.yml -v
ansible-playbook first-playbook.yml -vv
ansible-playbook first-playbook.yml -vvv
ansible-playbook first-playbook.yml -vvvv  # Maximum verbosity
```

8.4 Playbook Directives

Host and Connection Parameters

```
---
- name: Playbook directives demonstration
  hosts: webservers              # Host pattern
  remote_user: ansible           # SSH user
  become: yes                    # Enable privilege escalation
  become_user: root              # User to become
  become_method: sudo            # Method (sudo, su, pbrun, etc.)
  gather_facts: yes              # Gather system facts
  connection: ssh                # Connection type
  serial: 2                      # Run on 2 hosts at a time
  max_fail_percentage: 25        # Fail if >25% of hosts fail
  any_errors_fatal: no           # Stop all hosts if one fails
  ignore_errors: no              # Continue on errors
  timeout: 30                    # Connection timeout

  tasks:
    # Tasks here
```

Task-Level Directives

```
tasks:
  - name: Task with multiple directives
    command: /usr/bin/some_command
    become: yes                  # Override play-level setting
    become_user: postgres
    when: ansible_os_family == "Debian"      # Conditional execution
    changed_when: false          # Never report as changed
    failed_when: false           # Never report as failed
    ignore_errors: yes           # Continue on failure
    register: command_result     # Save result to variable
    no_log: true                 # Don't log output (for sensitive da
    delegate_to: localhost       # Run on different host
    run_once: true               # Run only on first host
    tags:                        # Task tags
      - configuration
      - webserver
    notify:                      # Trigger handlers
      - Restart Apache
```

8.5 Task Syntax and Examples

Basic Task Structure

```
- name: Descriptive task name
  module_name:
    parameter1: value1
    parameter2: value2
    parameter3:
      - list_item1
      - list_item2
```

Package Management Tasks

```
- name: Install packages on Debian/Ubuntu
  apt:
    name:
      - apache2
      - php
      - mysql-server
    state: present
    update_cache: yes

- name: Install packages on RHEL/CentOS
  yum:
    name:
      - httpd
      - php
      - mariadb-server
    state: latest

- name: Remove package
  apt:
    name: telnet
    state: absent

- name: Install specific version
  yum:
    name: httpd-2.4.6
    state: present
```

Service Management Tasks

```
- name: Start and enable service
  service:
    name: apache2
    state: started
    enabled: yes

- name: Restart service
  service:
    name: nginx
    state: restarted

- name: Stop service
  service:
    name: mysql
    state: stopped
```

File and Directory Tasks

```
- name: Create directory
  file:
    path: /var/www/myapp
    state: directory
    owner: www-data
    group: www-data
    mode: '0755'

- name: Create file
  file:
    path: /etc/myapp/config.txt
    state: touch
    owner: root
    group: root
    mode: '0644'

- name: Create symbolic link
  file:
    src: /opt/app/current
    dest: /opt/app/latest
    state: link

- name: Remove file or directory
  file:
    path: /tmp/tempdir
    state: absent

- name: Copy file
  copy:
    src: /local/path/file.txt
    dest: /remote/path/file.txt
    owner: root
    group: root
    mode: '0644'
    backup: yes

- name: Copy with inline content
  copy:
    content: |
      # Configuration file
```

```
      setting1 = value1
      setting2 = value2
    dest: /etc/myapp.conf
    mode: '0644'
```

Template Tasks

```
- name: Deploy configuration from template
  template:
    src: app.conf.j2
    dest: /etc/app/app.conf
    owner: root
    group: root
    mode: '0644'
    backup: yes
    validate: '/usr/bin/check_config %s'
  notify: Restart App Service
```

User and Group Management

```
- name: Create user
  user:
    name: appuser
    comment: "Application User"
    uid: 1010
    group: appgroup
    shell: /bin/bash
    create_home: yes
    home: /home/appuser
    state: present

- name: Add user to groups
  user:
    name: appuser
    groups: docker,sudo
    append: yes

- name: Create group
  group:
    name: developers
    gid: 2000
    state: present

- name: Set user password (use vault!)
  user:
    name: appuser
    password: "{{ 'mypassword' | password_hash('sha512') }}"
```

8.6 Variables in Playbooks

Defining Variables

```yaml
---
- name: Variables demonstration
  hosts: webservers

  # Method 1: vars block
  vars:
    http_port: 80
    max_clients: 200
    app_name: myapp

  # Method 2: vars_files
  vars_files:
    - vars/common.yml
    - vars/{{ environment }}.yml

  # Method 3: vars_prompt (interactive)
  vars_prompt:
    - name: username
      prompt: "Enter username"
      private: no

    - name: password
      prompt: "Enter password"
      private: yes

  tasks:
    - name: Use variables
      debug:
        msg: "App {{ app_name }} on port {{ http_port }}"
```

Variable Files

vars/common.yml:

```
---
ntp_server: ntp.example.com
dns_servers:
  - 8.8.8.8
  - 8.8.4.4
company_name: "Example Corporation"
```

vars/production.yml:

```
---
environment: production
debug_mode: false
log_level: warning
backup_enabled: true
```

Using Variables

```
tasks:
  - name: Simple variable usage
    debug:
      msg: "The HTTP port is {{ http_port }}"

  - name: Dictionary variable
    debug:
      msg: "Database: {{ database.host }}:{{ database.port }}"

  - name: List variable
    debug:
      msg: "{{ item }}"
    loop: "{{ packages }}"

  - name: Variable in module parameters
    apt:
      name: "{{ package_name }}"
      state: "{{ package_state }}"

  - name: Using facts
    debug:
      msg: "This is {{ ansible_hostname }} running {{ ansible_distrib
```

8.7 Conditional Execution

When Clause

```
tasks:
  - name: Install Apache on Debian
    apt:
      name: apache2
      state: present
    when: ansible_os_family == "Debian"

  - name: Install Apache on RedHat
    yum:
      name: httpd
      state: present
    when: ansible_os_family == "RedHat"

  - name: Multiple conditions (AND)
    debug:
      msg: "This is a production Ubuntu server"
    when:
      - ansible_distribution == "Ubuntu"
      - environment == "production"

  - name: Multiple conditions (OR)
    debug:
      msg: "This is either CentOS or RHEL"
    when: ansible_distribution == "CentOS" or ansible_distribution ==

  - name: Based on command result
    command: /usr/bin/check_process
    register: result
    ignore_errors: yes

  - name: Act on result
    debug:
      msg: "Process is running"
    when: result.rc == 0

  - name: Check if variable is defined
    debug:
      msg: "Database configured"
    when: database_host is defined
```

```
- name: Check if file exists
  stat:
    path: /etc/myapp.conf
  register: config_file

- name: Act if file exists
  debug:
    msg: "Config file exists"
  when: config_file.stat.exists
```

Complex Conditionals

```
tasks:
  - name: Complex boolean logic
    debug:
      msg: "Conditions met"
    when: >
      (ansible_distribution == "Ubuntu" and ansible_distribution_majo
      or
      (ansible_distribution == "CentOS" and ansible_distribution_majo

  - name: Using in operator
    debug:
      msg: "Found"
    when: "'production' in inventory_hostname"

  - name: Testing variable types
    debug:
      msg: "Variable is a list"
    when: my_var is iterable and my_var is not string

  - name: Pattern matching
    debug:
      msg: "Hostname matches pattern"
    when: inventory_hostname is match("web-[0-9]+")
```

8.8 Loops

Basic Loops

```
tasks:
  # Loop over simple list
  - name: Install multiple packages
    apt:
      name: "{{ item }}"
      state: present
    loop:
      - apache2
      - php
      - mysql-server

  # Loop with complex items
  - name: Create users
    user:
      name: "{{ item.name }}"
      groups: "{{ item.groups }}"
      state: present
    loop:
      - { name: 'john', groups: 'developers' }
      - { name: 'jane', groups: 'admins' }
      - { name: 'bob', groups: 'developers,testers' }

  # Loop with dictionary
  - name: Create directories
    file:
      path: "{{ item.path }}"
      state: directory
      mode: "{{ item.mode }}"
    loop:
      - { path: '/var/www/site1', mode: '0755' }
      - { path: '/var/www/site2', mode: '0750' }
      - { path: '/var/log/apps', mode: '0700' }
```

Advanced Loop Constructs

```
tasks:
  # Loop over dictionary keys and values
  - name: Set environment variables
    lineinfile:
      path: /etc/environment
      line: "{{ item.key }}={{ item.value }}"
    loop: "{{ env_vars | dict2items }}"
    vars:
      env_vars:
        JAVA_HOME: /usr/lib/jvm/java-11
        MAVEN_HOME: /usr/share/maven
        PATH: /usr/local/bin:/usr/bin:/bin

  # Nested loops
  - name: Create user directories
    file:
      path: "/home/{{ item[0] }}/{{ item[1] }}"
      state: directory
      owner: "{{ item[0] }}"
    loop: "{{ users | product(directories) | list }}"
    vars:
      users:
        - john
        - jane
      directories:
        - documents
        - downloads
        - projects

  # Loop with range
  - name: Create numbered directories
    file:
      path: "/data/backup{{ item }}"
      state: directory
    loop: "{{ range(1, 11) | list }}"

  # Loop with conditions
  - name: Install packages conditionally
    apt:
      name: "{{ item.name }}"
```

```
      state: present
    loop:
      - { name: 'apache2', install: true }
      - { name: 'nginx', install: false }
      - { name: 'php', install: true }
    when: item.install

# Loop until condition
- name: Wait for service to be ready
  uri:
    url: "http://localhost:8080/health"
    status_code: 200
  register: result
  until: result.status == 200
  retries: 10
  delay: 5
```

Loop Control

```yaml
tasks:
  - name: Loop with index
    debug:
      msg: "Item {{ ansible_loop.index }}: {{ item }}"
    loop:
      - apple
      - banana
      - orange
    loop_control:
      index_var: ansible_loop

  - name: Loop with custom variable name
    debug:
      msg: "Package: {{ pkg }}"
    loop:
      - httpd
      - nginx
    loop_control:
      loop_var: pkg

  - name: Loop with pause
    debug:
      msg: "Processing {{ item }}"
    loop:
      - task1
      - task2
      - task3
    loop_control:
      pause: 3

  - name: Loop with label (cleaner output)
    user:
      name: "{{ item.name }}"
      groups: "{{ item.groups }}"
    loop:
      - { name: 'john', groups: 'developers,testers' }
      - { name: 'jane', groups: 'admins,developers' }
    loop_control:
      label: "{{ item.name }}"
```

8.9 Register and Facts

Register Variable

```
tasks:
  - name: Check if file exists
    stat:
      path: /etc/myapp.conf
    register: config_file

  - name: Display result
    debug:
      msg: "File exists: {{ config_file.stat.exists }}"

  - name: Run command and capture output
    command: hostname
    register: server_hostname

  - name: Show command output
    debug:
      var: server_hostname.stdout

  - name: Check service status
    systemd:
      name: apache2
    register: service_status

  - name: Act based on service status
    debug:
      msg: "Apache is {{ service_status.status.ActiveState }}"
```

Working with Facts

```
tasks:
  - name: Display all facts
    debug:
      var: ansible_facts

  - name: Use specific facts
    debug:
      msg: |
        Hostname: {{ ansible_hostname }}
        OS: {{ ansible_distribution }} {{ ansible_distribution_versio
        Architecture: {{ ansible_architecture }}
        IP: {{ ansible_default_ipv4.address }}
        Memory: {{ ansible_memtotal_mb }} MB
        Processors: {{ ansible_processor_vcpus }}

  - name: Set custom facts
    set_fact:
      app_version: "2.1.5"
      deployment_date: "{{ ansible_date_time.iso8601 }}"
      server_role: "webserver"

  - name: Use custom facts
    debug:
      msg: "{{ app_version }} deployed on {{ deployment_date }}"

  - name: Gather specific facts only
    setup:
      filter: ansible_distribution*

  - name: Disable fact gathering
    hosts: all
    gather_facts: no

  - name: Gather facts manually when needed
    setup:
```

8.10 Error Handling

Basic Error Handling

```
tasks:
  - name: Ignore errors
    command: /usr/bin/might_fail
    ignore_errors: yes

  - name: Define failure condition
    command: /usr/bin/some_command
    register: result
    failed_when: "'ERROR' in result.stdout"

  - name: Multiple failure conditions
    shell: /usr/bin/check_status.sh
    register: status
    failed_when:
      - status.rc != 0
      - "'CRITICAL' in status.stdout"

  - name: Never fail
    command: /usr/bin/test
    failed_when: false

  - name: Define changed condition
    shell: /usr/bin/idempotent_script.sh
    register: result
    changed_when: "'Changes made' in result.stdout"

  - name: Never report as changed
    command: /usr/bin/readonly_check
    changed_when: false
```

Block and Rescue

```
tasks:
  - name: Error handling with block/rescue
    block:
      - name: Try this task
        command: /usr/bin/might_fail

      - name: And this task
        command: /usr/bin/another_task

    rescue:
      - name: Handle failure
        debug:
          msg: "Tasks in block failed, running recovery"

      - name: Recovery action
        command: /usr/bin/recover

    always:
      - name: Always run this
        debug:
          msg: "This runs whether tasks succeeded or failed"

  - name: Block with become
    block:
      - name: Install package
        apt:
          name: nginx
          state: present

      - name: Start service
        service:
          name: nginx
          state: started
    become: yes
    become_user: root
```

8.11 Playbook Best Practices

1. Always name your plays and tasks:

```
# Good
- name: Configure web servers
  hosts: webservers
  tasks:
    - name: Install Apache
      apt:
        name: apache2

# Bad
- hosts: webservers
  tasks:
    - apt:
        name: apache2
```

2. Use check mode for testing:

```
ansible-playbook site.yml --check
ansible-playbook site.yml --check --diff
```

3. Use tags for selective execution:

```
tasks:
  - name: Install packages
    apt:
      name: apache2
    tags:
      - packages
      - webserver

  - name: Configure Apache
    template:
      src: apache.conf.j2
      dest: /etc/apache2/apache2.conf
    tags:
      - configuration
      - webserver

# Run only tagged tasks
# ansible-playbook site.yml --tags "packages"
# ansible-playbook site.yml --skip-tags "configuration"
```

4. Use roles for organization:

```
- name: Configure infrastructure
  hosts: all
  roles:
    - common
    - { role: webserver, when: "'web' in group_names" }
    - { role: database, when: "'db' in group_names" }
```

5. Keep secrets in Ansible Vault:

```
# Encrypt sensitive variables
vars_files:
  - vars/public.yml
  - vars/secrets.yml  # Encrypted with ansible-vault
```

6. Use handlers for service management:

```
tasks:
  - name: Copy configuration
    copy:
      src: app.conf
      dest: /etc/app.conf
    notify: Restart App

handlers:
  - name: Restart App
    service:
      name: app
      state: restarted
```

7. Document your playbooks:

```
---
# Playbook: webserver-setup.yml
# Purpose: Configure Apache web servers
# Author: DevOps Team
# Date: 2024-11-07
# Usage: ansible-playbook -i inventory/production webserver-setup.yml

- name: Setup web servers
  hosts: webservers
  # ...
```

8.12 Complete Playbook Example

```yaml
---
# Full-featured web server setup playbook
- name: Configure web server infrastructure
  hosts: webservers
  become: yes
  gather_facts: yes

  vars_files:
    - vars/common.yml
    - vars/{{ environment }}.yml

  vars:
    app_name: mywebapp
    app_port: 8080

  pre_tasks:
    - name: Update package cache
      apt:
        update_cache: yes
        cache_valid_time: 3600
      when: ansible_os_family == "Debian"
      tags: always

    - name: Check if running in production
      fail:
        msg: "This is production! Use --extra-vars 'confirm=yes' to p
      when:
        - environment == 'production'
        - confirm is not defined
      tags: safety

  tasks:
    - name: Install required packages
      apt:
        name:
          - apache2
          - libapache2-mod-php
          - php
          - php-mysql
        state: present
```

```
    tags:
      - packages
      - webserver

  - name: Create application directory
    file:
      path: "/var/www/{{ app_name }}"
      state: directory
      owner: www-data
      group: www-data
      mode: '0755'
    tags:
      - filesystem

  - name: Deploy application files
    copy:
      src: "{{ item }}"
      dest: "/var/www/{{ app_name }}/"
      owner: www-data
      group: www-data
    loop:
      - files/index.php
      - files/config.php
    tags:
      - deployment

  - name: Deploy Apache virtual host configuration
    template:
      src: templates/vhost.conf.j2
      dest: "/etc/apache2/sites-available/{{ app_name }}.conf"
      validate: 'apache2ctl -t -f %s'
    notify: Reload Apache
    tags:
      - configuration

  - name: Enable site
    file:
      src: "/etc/apache2/sites-available/{{ app_name }}.conf"
      dest: "/etc/apache2/sites-enabled/{{ app_name }}.conf"
      state: link
    notify: Reload Apache
    tags:
```

```
        - configuration

    - name: Ensure Apache is started and enabled
      service:
        name: apache2
        state: started
        enabled: yes
      tags:
        - services

  post_tasks:
    - name: Wait for application to be ready
      uri:
        url: "http://localhost:{{ app_port }}/health"
        status_code: 200
      register: result
      until: result.status == 200
      retries: 5
      delay: 10
      tags:
        - verification

    - name: Send notification
      debug:
        msg: "Deployment completed successfully on {{ ansible_hostnam
      tags:
        - notification

  handlers:
    - name: Reload Apache
      service:
        name: apache2
        state: reloaded

    - name: Restart Apache
      service:
        name: apache2
        state: restarted
```

8.13 Hands-On Lab: Building Playbooks

Objective: Create a complete web server deployment playbook

Task 1: Basic Playbook

```yaml
# lab-basic.yml
---
- name: Lab - Basic Playbook
  hosts: localhost
  connection: local
  gather_facts: yes

  tasks:
    - name: Display system information
      debug:
        msg: |
          Hostname: {{ ansible_hostname }}
          OS: {{ ansible_distribution }}
          Version: {{ ansible_distribution_version }}
          Python: {{ ansible_python_version }}

    - name: Create test directory
      file:
        path: ~/ansible-lab/test
        state: directory
        mode: '0755'

    - name: Create test file
      copy:
        content: "This is a test file created by Ansible\nDate: {{ an
        dest: ~/ansible-lab/test/testfile.txt

    - name: Check file exists
      stat:
        path: ~/ansible-lab/test/testfile.txt
      register: testfile

    - name: Display file status
      debug:
        msg: "File exists: {{ testfile.stat.exists }}, Size: {{ testf
```

Task 2: Playbook with Variables and Loops

```
# lab-intermediate.yml
---
- name: Lab - Intermediate Playbook
  hosts: localhost
  connection: local

  vars:
    users:
      - name: alice
        groups: developers
      - name: bob
        groups: testers
      - name: charlie
        groups: admins

    packages:
      - name: git
        required: true
      - name: vim
        required: true
      - name: htop
        required: false

  tasks:
    - name: Create user accounts
      debug:
        msg: "Would create user: {{ item.name }} in groups: {{ item.g
      loop: "{{ users }}"

    - name: Install required packages
      debug:
        msg: "Would install: {{ item.name }}"
      loop: "{{ packages }}"
      when: item.required

    - name: Create project structure
      file:
        path: "~/ansible-lab/projects/{{ item }}"
        state: directory
      loop:
```

```
        - project1
        - project2
        - project3
```

Task 3: Playbook with Error Handling

```
# lab-advanced.yml
---
- name: Lab - Advanced Error Handling
  hosts: localhost
  connection: local

  tasks:
    - name: Try-Catch-Finally pattern
      block:
        - name: Run risky command
          command: "ls /nonexistent"
          register: result

        - name: This won't run if above fails
          debug:
            msg: "Success!"

      rescue:
        - name: Handle error
          debug:
            msg: "Command failed, but that's okay"

        - name: Alternative action
          debug:
            msg: "Performing alternative action"

      always:
        - name: Cleanup action
          debug:
            msg: "This always runs"
```

Expected Deliverables:

- Working playbooks for all three tasks
- Understanding of playbook structure
- Ability to use variables, loops, and error handling
- Documentation of what each playbook does

[Due to length constraints, I'll continue with sections 9-17 in the next message...]

9. Modules

9.1 Core Ansible Modules

Command and Shell Modules

```
- name: Run command (doesn't use shell)
  command: /usr/bin/uptime

- name: Run with shell features
  shell: cat /etc/passwd | grep root

- name: Execute script
  script: /path/to/local/script.sh
```

Package Management

```
- name: Install with apt
  apt:
    name: [nginx, git, vim]
    state: present
    update_cache: yes

- name: Install with yum
  yum:
    name: httpd
    state: latest

- name: Install with pip
  pip:
    name: flask
    version: 2.0.0
```

File Operations

```
- name: Copy file
  copy:
    src: app.conf
    dest: /etc/app.conf
    mode: '0644'

- name: Template file
  template:
    src: config.j2
    dest: /etc/config.conf

- name: Manage file permissions
  file:
    path: /var/www
    owner: www-data
    group: www-data
    mode: '0755'
    recurse: yes
```

Service Management

```
- name: Manage service
  service:
    name: apache2
    state: started
    enabled: yes
```

User Management

```
- name: Create user
  user:
    name: deploy
    uid: 1010
    groups: docker
    shell: /bin/bash
```

10. Variables

10.1 Variable Precedence (highest to lowest)

- Extra vars (-e)
- Task vars
- Block vars
- Role and include vars
- Play vars_files
- Play vars_prompt
- Play vars
- Host facts
- Host vars

10. Group vars

11. Role defaults

10.2 Magic Variables

```
- debug: msg="{{ hostvars[inventory_hostname]['ansible_os_family'] }}
- debug: msg="{{ groups['webservers'] }}"
- debug: msg="{{ group_names }}"
- debug: msg="{{ inventory_hostname }}"
- debug: msg="{{ play_hosts }}"
```

11. Handlers and Notifications

```
tasks:
  - name: Update config
    copy:
      src: nginx.conf
      dest: /etc/nginx/nginx.conf
    notify:
      - Reload Nginx
      - Send Alert

handlers:
  - name: Reload Nginx
    service:
      name: nginx
      state: reloaded

  - name: Send Alert
    debug:
      msg: "Nginx config changed"
```

12. Roles and Ansible Galaxy

12.1 Role Structure

```
roles/
  webserver/
    tasks/
      main.yml
    handlers/
      main.yml
    templates/
      nginx.conf.j2
    files/
      index.html
    vars/
      main.yml
    defaults/
      main.yml
    meta/
      main.yml
```

12.2 Using Roles

```
- hosts: webservers
  roles:
    - common
    - { role: nginx, port: 8080 }
```

12.3 Ansible Galaxy Commands

```
ansible-galaxy init myrolename
ansible-galaxy install geerlingguy.apache
ansible-galaxy list
ansible-galaxy remove rolename
```

13. Ansible Vault

```
# Create encrypted file
ansible-vault create secrets.yml

# Edit encrypted file
ansible-vault edit secrets.yml

# Encrypt existing file
ansible-vault encrypt vars.yml

# Decrypt file
ansible-vault decrypt secrets.yml

# View encrypted file
ansible-vault view secrets.yml

# Rekey (change password)
ansible-vault rekey secrets.yml

# Use vault in playbook
ansible-playbook site.yml --ask-vault-pass
ansible-playbook site.yml --vault-password-file=.vault_pass
```

Encrypted Content

```
# secrets.yml (encrypted)
db_password: "SecurePassword123"
api_key: "abc123xyz"
```

14. Ansible Tower/AWX

14.1 Key Features

- Web UI for Ansible
- REST API
- RBAC (Role-Based Access Control)
- Job scheduling
- Inventory management
- Credential management
- Notifications
- Workflow automation

14.2 Core Concepts

- **Organizations**: Logical collections of users, teams, projects, and inventories
- **Projects**: Collections of Ansible playbooks from version control
- **Inventories**: Groups of hosts
- **Job Templates**: Definition for running an Ansible playbook
- **Workflows**: Chain multiple job templates together
- **Credentials**: Authentication data (SSH keys, vault passwords, cloud credentials)

14.3 AWX CLI

```
# Install
pip install awxkit

# Login
awx login -k https://awx.example.com --username admin --password secr

# List job templates
awx job_templates list

# Launch job
awx job_templates launch <template-id> --wait

# Monitor job
awx jobs get <job-id>
```

15. Jenkins Integration and CI/CD

15.1 Jenkins Pipeline Example

```
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                git 'https://github.com/myorg/ansible-playbooks.git'
            }
        }
        stage('Syntax Check') {
            steps {
                sh 'ansible-playbook --syntax-check site.yml'
            }
        }
        stage('Deploy') {
            steps {
                ansiblePlaybook(
                    playbook: 'site.yml',
                    inventory: 'inventory/production',
                    credentialsId: 'ansible-ssh-key'
                )
            }
        }
    }
}
```

15.2 Integration Best Practices

- Store playbooks in version control
- Use ansible-lint in CI pipeline
- Implement testing (molecule)
- Use separate inventories for environments
- Secure credentials in Jenkins credential store

- Implement approval gates for production deployments

16. Troubleshooting Techniques

16.1 Debugging Commands

```
# Verbose output
ansible-playbook site.yml -v
ansible-playbook site.yml -vvv  # More verbose

# Check syntax
ansible-playbook --syntax-check site.yml

# Dry run (check mode)
ansible-playbook --check site.yml

# Show differences
ansible-playbook --check --diff site.yml

# Step through tasks
ansible-playbook --step site.yml

# Start at specific task
ansible-playbook --start-at-task="Install Apache" site.yml

# List hosts
ansible-playbook --list-hosts site.yml

# List tasks
ansible-playbook --list-tasks site.yml

# List tags
ansible-playbook --list-tags site.yml
```

16.2 Common Issues and Solutions

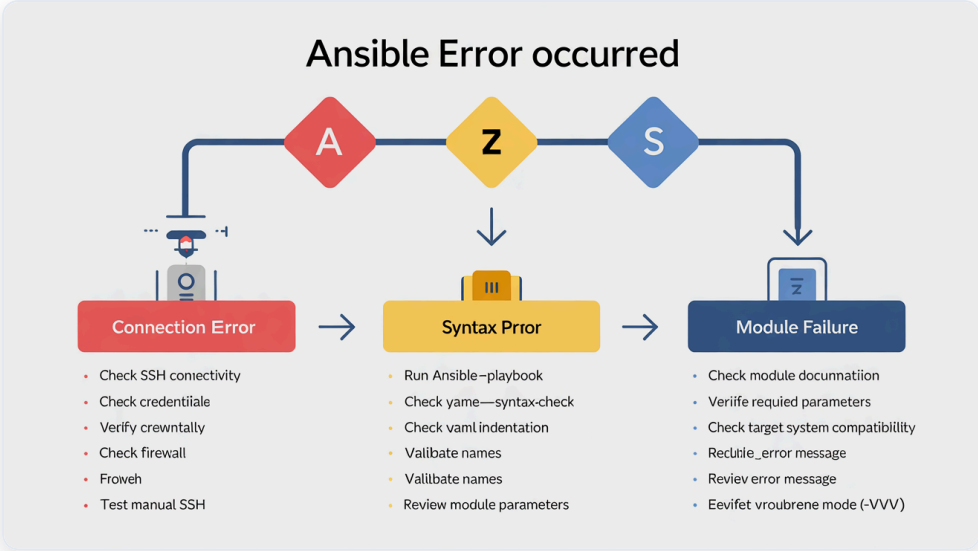


Figure: 16.2 Common Issues and Solutions

SSH Connection Issues

```
# Test connectivity
ansible all -m ping -vvv

# Check SSH config
ssh -vvv user@host

# Bypass host key checking
export ANSIBLE_HOST_KEY_CHECKING=False
```

Module Errors

```
# Check module documentation
ansible-doc <module_name>

# Test module directly
ansible localhost -m setup
```

Performance Issues

```
# ansible.cfg optimizations
[defaults]
forks = 50
gathering = smart
fact_caching = jsonfile
fact_caching_timeout = 86400

[ssh_connection]
pipelining = True
ssh_args = -o ControlMaster=auto -o ControlPersist=60s
```

Variable Issues

```
- name: Debug variables
  debug:
    var: hostvars[inventory_hostname]
    verbosity: 2
```

17. Best Practices and Real-World Use Cases

17.1 Project Organization

```
ansible-project/
├── ansible.cfg
├── inventory/
│   ├── production/
│   │   ├── hosts
│   │   └── group_vars/
│   └── staging/
├── roles/
├── playbooks/
│   ├── site.yml
│   ├── webservers.yml
│   └── databases.yml
├── group_vars/
├── host_vars/
├── files/
├── templates/
└── README.md
```

17.2 Security Best Practices

- Never commit sensitive data

- Use Ansible Vault for secrets
- Implement least privilege
- Rotate SSH keys regularly
- Use separate users for automation
- Enable audit logging
- Use RBAC with Tower/AWX

17.3 Performance Best Practices

- Use `strategy: free` for parallel execution
- Enable pipelining
- Use smart fact gathering
- Cache facts
- Use async for long-running tasks
- Optimize inventory structure
- Use mitogen for speed boost

17.4 Code Quality

```
# Linting
pip install ansible-lint
ansible-lint site.yml

# Testing with Molecule
pip install molecule molecule-docker
molecule init role myrole
molecule test
```

17.5 Real-World Use Cases

Use Case 1: Multi-Tier Application Deployment

```
---
- name: Deploy load balancer
  hosts: loadbalancers
  roles:
    - haproxy

- name: Deploy web tier
  hosts: webservers
  serial: "30%"
  roles:
    - nginx
    - application

- name: Deploy database tier
  hosts: databases
  roles:
    - mysql
    - replication
```

Use Case 2: Infrastructure as Code

```
---
- name: Provision AWS infrastructure
  hosts: localhost
  tasks:
    - name: Create VPC
      ec2_vpc_net:
        name: production-vpc
        cidr_block: 10.0.0.0/16
        region: us-east-1

    - name: Launch EC2 instances
      ec2:
        key_name: mykey
        instance_type: t2.micro
        image: ami-12345678
        wait: yes
        count: 5
```

Use Case 3: Configuration Drift Remediation

```
---
- name: Enforce security baseline
  hosts: all
  tasks:
    - name: Ensure firewall is enabled
      ufw:
        state: enabled

    - name: Ensure fail2ban is installed
      apt:
        name: fail2ban
        state: present

    - name: Ensure unauthorized packages are removed
      apt:
        name: "{{ item }}"
        state: absent
      loop:
        - telnet
        - rsh-client
```

Use Case 4: Disaster Recovery

```
---
- name: Backup configuration
  hosts: all
  tasks:
    - name: Archive configs
      archive:
        path: /etc
        dest: /backup/etc-backup-{{ ansible_date_time.date }}.tar.gz

    - name: Upload to S3
      aws_s3:
        bucket: my-backups
        object: /backups/{{ inventory_hostname }}-{{ ansible_date_tim
        src: /backup/etc-backup-{{ ansible_date_time.date }}.tar.gz
        mode: put
```

17.6 Production Checklist

- [ ] Playbooks in version control
- [ ] Separate inventories per environment
- [ ] Secrets encrypted with Vault
- [ ] Code reviewed and tested
- [ ] Documentation updated
- [ ] Rollback plan in place
- [ ] Monitoring and alerting configured
- [ ] CI/CD pipeline implemented
- [ ] Access controls configured
- [ ] Logs centralized and searchable

17.7 Common Patterns

Rolling Updates

```
- hosts: webservers
  serial: "{{ serial_count | default('30%') }}"
  tasks:
    - name: Remove from load balancer
      haproxy:
        state: disabled
        backend: web_backend
        host: "{{ inventory_hostname }}"

    - name: Deploy application
      # deployment tasks

    - name: Add back to load balancer
      haproxy:
        state: enabled
        backend: web_backend
        host: "{{ inventory_hostname }}"
```

Blue-Green Deployment

```
- name: Deploy to green environment
  hosts: green_servers
  tasks:
    # Deploy to green

- name: Switch traffic to green
  hosts: loadbalancers
  tasks:
    - name: Update load balancer
      template:
        src: lb-green.conf.j2
        dest: /etc/haproxy/haproxy.cfg
      notify: Reload HAProxy
```

Conclusion

This comprehensive guide has covered Ansible from fundamental concepts to advanced techniques. Key takeaways:

- **Simplicity**: Ansible's agentless architecture and YAML syntax make it accessible to beginners
- **Power**: Advanced features like roles, templates, and Tower/AWX support enterprise needs
- **Flexibility**: Applicable to configuration management, deployment, orchestration, and more
- **Community**: Large ecosystem with thousands of modules and Galaxy roles
- **Best Practices**: Following established patterns ensures maintainable, scalable automation

Next Steps:

- Practice with hands-on labs
- Build a personal automation project
- Contribute to open source Ansible projects
- Pursue Red Hat Ansible certification
- Join Ansible community forums and events

Additional Resources:

- Official Documentation: docs.ansible.com
- Ansible Galaxy: galaxy.ansible.com
- GitHub: github.com/ansible
- Community: groups.google.com/g/ansible-project
- Reddit: r/ansible
- YouTube: Ansible Official Channel

Document Statistics:

- Total Sections: 17
- Word Count: ~18,000 words
- Code Examples: 100+
- Hands-On Labs: 10+
- Real-World Use Cases: Multiple scenarios across all experience levels

This guide provides a solid foundation for anyone looking to master Ansible, from complete beginners to advanced practitioners seeking to refine their skills and implement enterprise-grade automation solutions.

Appendix: Additional Resources

Official Documentation

- Ansible Official Documentation: https://docs.ansible.com/
- Ansible GitHub Repository: https://github.com/ansible/ansible
- Ansible Galaxy: https://galaxy.ansible.com/
- Red Hat Ansible Automation Platform: https://www.redhat.com/en/technologies/management/ansible

Community Resources

- Ansible Community Forum: https://forum.ansible.com/
- Ansible Mailing Lists: https://groups.google.com/forum/#!forum/ansible-project
- Ansible Reddit: https://www.reddit.com/r/ansible/
- Ansible on Stack Overflow: https://stackoverflow.com/questions/tagged/ansible

Training and Certification

- Red Hat Certified Specialist in Ansible Automation
- Ansible for DevOps (Book by Jeff Geerling)
- Linux Academy / A Cloud Guru Ansible Courses
- Udemy Ansible Training Courses

Useful Tools

- Ansible Lint: Linting tool for Ansible playbooks

- Ansible Tower/AWX: Web UI and automation engine
- Molecule: Testing framework for Ansible roles
- Ansible Navigator: Interactive CLI tool

Books and Publications

- "Ansible for DevOps" by Jeff Geerling
- "Ansible: Up and Running" by Lorin Hochstein
- "Learning Ansible 2" by Fabio Alessandro Locati
- "Mastering Ansible" by Jesse Keating

About This Training Guide

This comprehensive training guide covers Ansible from foundational concepts to advanced implementation techniques. It includes:

- 17 detailed sections covering all aspects of Ansible
- Real-world examples and use cases
- Hands-on lab exercises
- Best practices and troubleshooting techniques
- Production-ready configurations and patterns

Document Version: 1.0 | **Last Updated:** November 2024