RHCE Certification Study Guide

Comprehensive Study Materials for Red Hat Certified Engineer Exam

RHCE Study Repository

Study materials for RHCE certification preparation

Table of Contents

1. RHCE Certification Study Guide 10

1.1 About the RHCE Certification 10

1.2 Comprehensive Study Resources 10

Core Reference Documents 10

Comprehensive Study Modules 10

Practice Resources 11

1.3 Quick Start Guide 11

For Immediate Exam Preparation 11

For Complete Mastery 11

1.4 Lab Environment Requirements 11

1.5 Strategic Study Approach 12

Phase 1: Foundation (Weeks 1-2) 12

Phase 2: Core Skills (Weeks 3-4) 12

Phase 3: Advanced Features (Weeks 5-6) 13

Phase 4: Security & Exam Prep (Week 7) 13

1.6 Key Success Factors 13

1.7 Complete RHCE Exam Objectives Coverage 13

Understand core components of Ansible 13

Use roles and Ansible Content Collections 13

Install and configure an Ansible control node 14

Configure Ansible managed nodes 14

Run playbooks with Automation content navigator 14

Create Ansible plays and playbooks 14

Automate standard RHCSA tasks using Ansible modules 14

Manage content 14

1.8 Begin Your RHCE Journey 14

2. Study Modules 15

2.1 RHCE Comprehensive Study Modules 15

Module Structure 15

Complete Study Path 15

Study Progress Tracker 18

Strategic Study Approach 18

Success Strategies 19

Comprehensive Navigation 20

Begin Your RHCE Journey 20

2.2 Module 00: RHCE Exam Overview 21

Learning Objectives 21

Exam Details 21

Official RHCE Exam Objectives 21

Lab Environment Setup 24

7-Week Study Plan 25

Key Success Factors 27

Common Pitfalls to Avoid 27

Mental Preparation 28

Essential Commands to Memorize 29

Next Steps 29

2.3 Module 01: Ansible Basics 30

Learning Objectives 30

Why Ansible for Red Hat Automation 30

Ansible Architecture 30

Installation and Configuration 32

Inventory Management 34

SSH Configuration and Authentication 37

Ad-hoc Commands 38

Understanding Ansible Modules 41

Practical Lab Exercises 43

Key Takeaways 44

Next Steps 45

2.4 Module 02: Playbooks & Tasks 46

Learning Objectives 46

Why Playbooks Matter 46

YAML Fundamentals for Ansible 47

Playbook Structure 49

Task Definition and Module Usage 52

Handlers 55

Tags for Task Organization 58

Error Handling 59

Playbook Testing and Execution 62

Practical Lab Exercises 64

Key Takeaways 65

Next Steps 66

2.5 Module 03: Variables & Facts 67

Learning Objectives 67

Why Variables Transform Automation 67

Variable Types and Definition 68

Variable Precedence (16 Levels) 72

Facts (System Information) 74

Magic Variables 76

Registering Variables 78

Variable Organization Strategies 80

Variable Debugging 82

Practical Lab Exercises 83

Key Takeaways 85

Next Steps 86

2.6 Module 04: Task Control 87

Learning Objectives 87

Why Task Control Matters 87

Conditional Execution (when) 88

Loops and Iteration 93

Error Handling and Recovery 98

Task Delegation and Control 103

Advanced Task Control Patterns 107

Practical Lab Exercises 110

Key Takeaways 111

Next Steps 112

2.7 Module 05: Templates 113

Learning Objectives 113

Why Templates Transform Configuration Management 113

Jinja2 Template Fundamentals 114

Variable Usage in Templates 115

Control Structures in Templates 118

Filters and Data Transformation 121

Template Module Usage 123

Advanced Template Patterns 125

Template Debugging and Troubleshooting 128

Practical Lab Exercises 131

Key Takeaways 132

Next Steps 133

2.8 Module 06: Roles & Collections 134

Learning Objectives 134

Why Roles Transform Automation Architecture 134

Role Structure and Organization 135

Role Development Best Practices 137

Using Roles in Playbooks 143

Ansible Galaxy 144

Ansible Collections 146

Advanced Role Patterns 149
Role Testing Strategies 152
Practical Lab Exercises 154
Key Takeaways 155
Next Steps 156
2.9 Module 07: System Administration Tasks 157
Learning Objectives 157
Why Automate System Administration 157
Package and Repository Management 159
Service Management with Systemd 163
Storage and Filesystem Management 167
User and Group Management 173
Network and Security Configuration 177
System Monitoring and Maintenance 182
System Security Hardening 187
Practical Lab Exercises 189
Key Takeaways 190
Next Steps 191
2.10 Module 08: Ansible Vault & Advanced Features 192
Learning Objectives 192
Why Security and Advanced Features Matter 192
Ansible Vault Fundamentals 193
Vault Password Management 195
Organizing Encrypted Data 199
Dynamic Inventories 203
Performance Optimization 208
Advanced Debugging and Troubleshooting 212
Advanced Ansible Patterns 218
Practical Lab Exercises 222
Key Takeaways 223

Final Steps and Continuing Education 224

3. Quick References 226

3.1 RHCE Study Guide Summary 226

Concise Reference for RHCE EX294 Exam Success 226

RHCE Exam Overview 226

Core RHCE Topics 227

Exam Strategy and Tips 235

Essential Module Quick Reference 236

Final Preparation Checklist 237

3.2 RHCE Exam Day Commands 239

Essential Commands for EX294 Success 239

Exam Workflow Commands 239

Configuration Commands 243

Time-Saving Command Combinations 244

Exam Success Patterns 245

What NOT to Do on Exam 246

3.3 RHCE Exam Quick Reference (Cheat Sheet) 247

Essential Commands & Syntax for EX294 Success 247

Core Configuration 247

Essential Exam Commands 248

Playbook Syntax 249

Essential Modules with FQCN 250

Variables & Facts 251

Task Control 252

Templates & Jinja2 254

Roles & Collections 255

Ansible Vault 256

Playbook Execution Commands 257

Troubleshooting 258

Exam Success Patterns 258

3.4 RHCE Comprehensive Command Reference 260

Complete Command Reference for RHCE Study & Production Use 260

1. Install and Configure Ansible Control Node 262

2. Configure Ansible Managed Nodes 267

3. Automation Content Navigator 273

4. Content Collections Management 276

5. Role Management 281

6. Playbook Development and Execution 285

7. RHCSA Task Automation Commands 292

8. Ansible Vault Operations 302

9. Debugging and Troubleshooting 308

10. Documentation and Help Systems 317

Quick Command Combinations for Exam 321

Exam Success Strategies 321

3.5 RHCE Acronyms & Glossary 323

Comprehensive RHCE Terminology Reference 323

Essential RHCE Acronyms 323

Comprehensive Terminology by Category 324

Module Categories and Classifications 330

Exam-Specific Concepts 330

Common Exam Pitfalls and Solutions 331

Study Strategy and Priority Framework 332

Quick Reference for Exam Day 333

3.6 RHCE Knowledge Gaps Checklist - For Experienced Users 334

Self-Assessment for Production Ansible Users 334

How to Use This Checklist 334

1. Automation Content Navigator 334

2. Content Collections & FQCN 335

3. Control Node Setup & Configuration 335

4. Managed Node Configuration & SSH 336

5. RHCSA Task Automation (Critical Exam Area) 336

6. Shell Script Conversion (Exam Requirement) 338

7. Playbook Development (Advanced Patterns) 338

8. Ansible Vault (Security) 339

9. Exam Environment Constraints 339

Gap Analysis Summary 339

Action Plan Template 340

1. RHCE Certification Study Guide

Welcome to the comprehensive Red Hat Certified Engineer (RHCE) certification study repository! This resource provides complete coverage of all RHCE exam objectives with practical automation focus for passing the EX294 exam.

PDF Download: Download the complete study guide as PDF

1.1 About the RHCE Certification

The Red Hat Certified Engineer (RHCE) certification demonstrates your ability to use Ansible for enterprise automation, configuration management, and orchestration. The EX294 exam focuses on:

- **Ansible Automation**: Complete playbook and role development
- **Configuration Management**: Template-based system configuration at scale
- **Task Control**: Advanced conditionals, loops, and error handling
- **Security**: Ansible Vault implementation and secure automation practices
- **System Administration**: Automating all standard RHCSA tasks with Ansible

1.2 Comprehensive Study Resources

Core Reference Documents

Essential quick-access materials for study and exam preparation:

- **eBook Summary** - Concise overview of all exam topics *(468 lines)*
- **Exam Quick Reference** - Cheat sheet with copy-paste syntax *(561 lines)*
- **Command Reference by Topic** - All commands organized by exam objectives *(1,880 lines)*
- **RHCE Acronyms & Glossary** - Complete terminology reference *(400 lines)*

Comprehensive Study Modules

Complete 9-module curriculum covering all RHCE exam objectives *(8,777+ total lines)*:

Foundation Level:

- **Module 00: RHCE Exam Overview & Strategy** - Exam format, lab setup, 7-week study plan *(314 lines)*
- **Module 01: Ansible Basics & Configuration** - Architecture, installation, inventory, ad-hoc commands *(908 lines)*

Core Skills Level:

- **Module 02: Playbooks & Tasks** - YAML syntax, FQCN, handlers, ansible-navigator *(847 lines)*
- **Module 03: Variables & Facts** - All 16 precedence levels, magic variables, fact usage *(951 lines)*
- **Module 04: Task Control** - Conditionals, loops, error handling, delegation *(925 lines)*

Advanced Skills Level:

- **Module 05: Templates & Jinja2** - Complete templating, filters, configuration management *(897 lines)*
- **Module 06: Roles & Collections** - Role development, Galaxy, FQCN collections *(954 lines)*
- **Module 07: System Administration Tasks** - Automating all RHCSA tasks *(906 lines)*
- **Module 08: Ansible Vault & Advanced Features** - Security, optimization, troubleshooting *(975 lines)*

Practice Resources

- **Knowledge Gaps Checklist** - Self-assessment for focused study
- **Anki Flashcards** - `anki/rhce_deck.csv` for command memorization and concept reinforcement

1.3 Quick Start Guide

For Immediate Exam Preparation

1. **Start with** **eBook Summary** - Get comprehensive overview of all topics
2. **Use** **Exam Quick Reference** - Essential syntax for exam day
3. **Practice with** **Command Reference** - Master all required commands

For Complete Mastery

1. **Begin with Module 00: RHCE Exam Overview** - Understand exam format and strategy
2. **Set up RHEL 9 lab environment** - Control node + 2-3 managed nodes for hands-on practice
3. **Work through all 9 modules systematically** - Foundation → Core Skills → Advanced Features
4. **Practice with Anki flashcards daily** - Import `anki/rhce_deck.csv` for spaced repetition
5. **Use quick references during final preparation** - Cheat sheets and command references

1.4 Lab Environment Requirements

Essential Setup for Hands-On Practice:

Control Node:

- RHEL 9 system with ansible-core installed
- SSH keys configured for passwordless authentication
- Required collections: ansible.posix, community.general

Managed Nodes:

- 2-3 RHEL 9 systems for automation targets
- Python 3 installed for Ansible module execution
- SSH server running and accessible from control node

Network:

- All nodes on same network with SSH connectivity
- DNS or /etc/hosts configuration for hostname resolution

1.5 Strategic Study Approach

Phase 1: Foundation (Weeks 1-2)

Modules 00-01 - Build solid understanding

- Complete exam overview and Ansible basics
- Set up complete lab environment with RHEL 9 systems
- Master SSH key distribution and ansible.cfg configuration
- Practice ad-hoc commands with all essential modules

Phase 2: Core Skills (Weeks 3-4)

Modules 02-04 - Develop automation proficiency

- Master playbooks, variables, and task control
- Build complex multi-play automation with proper error handling
- Practice with real scenarios using templates and conditionals
- Focus on debugging and troubleshooting failed automation

Phase 3: Advanced Features (Weeks 5-6)

Modules 05-07 - Professional automation patterns

- Master Jinja2 templating and configuration management
- Develop reusable roles with Galaxy and collection integration
- Automate complete system administration workflows
- Practice enterprise-grade patterns and performance optimization

Phase 4: Security & Exam Prep (Week 7)

Module 08 + Final preparation

- Implement comprehensive security with Ansible Vault
- Master advanced debugging and performance optimization
- Practice timed scenarios under exam conditions
- Review all quick references and command patterns

1.6 Key Success Factors

- **Hands-On Practice**: Performance-based exam requires constant practical implementation
- **Time Management**: Master efficient automation patterns under exam pressure
- **ansible-navigator Proficiency**: Exam uses navigator, not legacy ansible-playbook
- **FQCN Mastery**: All modules must use Fully Qualified Collection Names
- **Vault Integration**: Security automation is mandatory for sensitive data
- **Systematic Debugging**: Develop consistent approaches to troubleshooting failures
- **Documentation Skills**: Master using `ansible-doc` as your primary reference tool

1.7 Complete RHCE Exam Objectives Coverage

This comprehensive study guide covers **100% of official RHCE EX294 exam objectives**:

Understand core components of Ansible

- Inventories, modules, variables, facts, loops, conditionals, plays, playbooks
- Configuration files, roles, and provided documentation usage

Use roles and Ansible Content Collections

- Create and work with roles, install from Galaxy

- Install and use Content Collections with FQCN

Install and configure an Ansible control node

- Package installation, inventory creation, configuration files

Configure Ansible managed nodes

- SSH keys, privilege escalation, file deployment, shell script conversion

Run playbooks with Automation content navigator

- Navigator execution, module discovery, inventory creation

Create Ansible plays and playbooks

- Common modules, variables, conditionals, error handling, system state

Automate standard RHCSA tasks using Ansible modules

- Packages, services, firewall, filesystems, storage, files, archives, scheduling, security, users

Manage content

- Create and use templates with Jinja2 customization

1.8 Begin Your RHCE Journey

Ready to master enterprise Ansible automation?

Start with Module 00: RHCE Exam Overview & Strategy

Get the complete exam format, lab setup guide, and proven success strategies to begin your certification journey with confidence!

Transform your automation skills and advance your career with RHCE certification!

2. Study Modules

2.1 RHCE Comprehensive Study Modules

This comprehensive study guide contains 9 focused modules covering all RHCE exam objectives with detailed content totaling over 8,000 lines of practical automation knowledge. Each module builds upon previous knowledge and includes extensive practical examples, lab exercises, and real-world scenarios.

Module Structure

Each module follows a consistent format:

- **Learning Objectives** - Clear goals for what you'll master
- **Practical Context** - Why these skills matter in real environments
- **Comprehensive Examples** - Real-world playbooks, roles, and automation
- **Advanced Patterns** - Professional-grade automation techniques
- **Lab Exercises** - Hands-on practice scenarios for skill building
- **Key Takeaways** - Essential knowledge for exam success

Complete Study Path

Foundation Level (Build Your Base)

Module 00: RHCE Exam Overview & Strategy *(314 lines)*

- Complete exam format and official objectives breakdown
- 7-week structured study timeline with daily practice goals
- Lab environment setup for RHEL 9 control and managed nodes
- Critical success factors and common pitfall avoidance
- Mental preparation and exam day strategy

Module 01: Ansible Basics & Configuration *(908 lines)*

- Ansible architecture, agentless design, and core concepts
- Complete installation and configuration on RHEL 9
- Comprehensive inventory management (INI and YAML formats)
- SSH key distribution and authentication setup
- Ad-hoc commands mastery with all essential modules
- ansible.cfg optimization and connection methods

Core Skills Level (Build Proficiency)

Module 02: Playbooks & Tasks *(847 lines)*

- Advanced YAML syntax and best practices for complex playbooks
- Complete playbook structure with all keywords and options
- FQCN (Fully Qualified Collection Names) requirements and usage
- Handler implementation for service and configuration management
- Tags for selective execution and organization
- Comprehensive error handling with blocks and rescue patterns
- ansible-navigator execution modes and testing workflows

Module 03: Variables & Facts *(951 lines)*

- Complete variable precedence hierarchy (all 16 levels)
- Advanced variable organization with host_vars and group_vars
- Fact gathering optimization and custom facts creation
- Magic variables for cross-host information sharing
- Advanced variable debugging and validation techniques
- Complex data structure handling and transformation

Module 04: Task Control *(925 lines)*

- Advanced conditional logic with complex boolean expressions
- Comprehensive loop patterns for data processing at scale
- Enterprise-grade error handling with block/rescue/always
- Task delegation and advanced execution control
- Serial execution and throttling for rolling deployments
- Performance optimization patterns for large infrastructures

Advanced Level (Master Professional Skills)

Module 05: Templates & Jinja2 *(897 lines)*

- Complete Jinja2 syntax mastery for dynamic configuration
- Advanced control structures, filters, and data transformation
- Template inheritance and macro systems for code reuse
- Complex configuration generation using facts and variables
- Whitespace control and template debugging techniques
- Enterprise configuration management patterns

Module 06: Roles & Collections *(954 lines)*

- Professional role development with complete directory structures
- Advanced role organization and variable handling strategies
- Ansible Galaxy integration for role and collection management
- FQCN requirements for all essential collections
- Role dependencies and complex composition patterns
- Testing strategies and validation frameworks for roles

Module 07: System Administration Tasks *(906 lines)*

- Complete automation of all RHCSA tasks using Ansible
- Advanced package and repository management across distributions
- Systemd service management with custom unit files
- Comprehensive storage management including LVM automation
- User and group lifecycle management with security policies
- Network security with firewalld and SELinux automation
- System hardening and compliance automation patterns

Module 08: Ansible Vault & Advanced Features *(975 lines)*

- Complete Ansible Vault mastery for enterprise security
- Multiple vault password management and organizational patterns
- Dynamic inventories for cloud and container environments
- Performance optimization for large-scale deployments
- Advanced debugging and troubleshooting methodologies
- Custom modules and plugins development
- Enterprise integration patterns and best practices

Study Progress Tracker

Track your completion through the comprehensive RHCE curriculum:

Foundation Modules

- [ ] **Module 00**: RHCE Exam Overview & Strategy *(314 lines)*
- [ ] **Module 01**: Ansible Basics & Configuration *(908 lines)*

Core Skills Modules

- [ ] **Module 02**: Playbooks & Tasks *(847 lines)*
- [ ] **Module 03**: Variables & Facts *(951 lines)*
- [ ] **Module 04**: Task Control *(925 lines)*

Advanced Skills Modules

- [ ] **Module 05**: Templates & Jinja2 *(897 lines)*
- [ ] **Module 06**: Roles & Collections *(954 lines)*
- [ ] **Module 07**: System Administration Tasks *(906 lines)*
- [ ] **Module 08**: Ansible Vault & Advanced Features *(975 lines)*

Total Content: 8,777+ lines of comprehensive RHCE automation knowledge

Strategic Study Approach

Phase 1: Foundation (Weeks 1-2)

Modules 00-01 - Build solid understanding

- Set up complete RHEL 9 lab environment with control node and managed nodes
- Master SSH key distribution and ansible.cfg configuration
- Practice ad-hoc commands extensively with all core modules
- Build confidence with basic automation patterns

Phase 2: Core Skills (Weeks 3-4)

Modules 02-04 - Develop automation proficiency

- Create complex multi-play playbooks with proper error handling
- Master variable precedence and fact usage in real scenarios
- Implement advanced task control with loops and conditionals
- Practice debugging failed automation systematically

Phase 3: Advanced Features (Weeks 5-6)

Modules 05-07 - Professional-level automation

- Design template-based configuration management systems
- Develop reusable roles with proper dependencies and testing
- Automate complete system administration workflows
- Build enterprise-grade automation patterns

Phase 4: Security & Optimization (Week 7)

Module 08 + Exam Preparation

- Implement secure automation with Ansible Vault throughout
- Optimize performance for production-scale deployments
- Master advanced troubleshooting and debugging techniques
- Complete timed practice scenarios under exam conditions

Success Strategies

Daily Practice (Essential)

1. **Hands-On Implementation**: Type every example, don't copy-paste
2. **Lab Environment**: Maintain active RHEL 9 lab for immediate testing
3. **Documentation Mastery**: Use `ansible-doc` and `ansible-navigator doc` extensively
4. **Time Management**: Practice common tasks under time pressure
5. **Error Recovery**: Learn to debug and fix failed automation quickly

Knowledge Retention

- **Build Personal Library**: Save and organize your working playbooks

- **Version Control**: Use Git to track your automation development
- **Progressive Complexity**: Start simple, add features systematically
- **Peer Learning**: Share knowledge and learn from others' approaches

Exam Readiness

- **Official Objectives**: Each module maps directly to exam requirements
- **Performance-Based**: Focus on working automation, not just theory
- **Production Patterns**: Learn enterprise-grade practices, not just basic syntax
- **Troubleshooting**: Develop systematic approaches to debugging failures

Comprehensive Navigation

Core Study Resources

- **Exam Quick Reference** - Essential syntax cheat sheet (561 lines)
- **Command Reference by Topic** - All commands organized (1,880 lines)
- **RHCE Acronyms & Glossary** - Complete terminology (400 lines)
- **eBook Summary** - Concise study guide overview (468 lines)

Practice Resources

- **Lab Environment**: Set up dedicated RHEL 9 systems for hands-on practice
- **GitHub Repository**: Track your automation development and share with community
- **Study Groups**: Connect with other RHCE candidates for collaborative learning
- **Practice Exams**: Use scenarios from primary study sources

Begin Your RHCE Journey

Start with **Module 00: RHCE Exam Overview & Strategy** to:

- Understand the complete exam format and official objectives
- Set up your lab environment for hands-on practice
- Create your personalized 7-week study timeline
- Build confidence with proven success strategies

Ready to master enterprise Ansible automation? Your RHCE certification journey begins now!

2.2 Module 00: RHCE Exam Overview

Learning Objectives

By the end of this module, you will understand:

- The RHCE exam format and structure
- Required lab environment setup
- Effective study strategies and timeline
- Key success factors for exam day
- Common pitfalls and how to avoid them

Exam Details

RHCE EX294 Exam Specifications

Exam Code: EX294 - Red Hat Certified Engineer (RHCE)
Duration: 4 hours
Format: Performance-based, hands-on lab exam
Passing Score: 210/300 (70%)
Prerequisites: RHCSA certification required

Exam Environment

- **Control Node**: RHEL 9 system with Ansible pre-installed
- **Managed Nodes**: Multiple RHEL 9 systems for automation targets
- **Network Access**: Systems can communicate via SSH
- **Documentation**: Access to man pages and `ansible-doc`
- **No Internet**: No access to external documentation or resources

Key Constraints

Time Pressure: 4 hours for multiple complex tasks
No Internet: Only local documentation available
No Preparation: Cannot modify environment before exam starts
Performance-Based: Must demonstrate working solutions, not just knowledge

Official RHCE Exam Objectives

The RHCE EX294 exam tests your ability to perform these **official Red Hat objectives**:

Prerequisites

Be able to perform all tasks expected of a Red Hat Certified System Administrator

- Understand and use essential tools
- Operate running systems
- Configure local storage
- Create and configure file systems
- Deploy, configure, and maintain systems
- Manage users and groups
- Manage security

1. Understand core components of Ansible

- **Inventories** - Static and dynamic host management
- **Modules** - Using built-in and collection modules effectively
- **Variables** - Variable types, scoping, and precedence
- **Facts** - System information gathering and utilization
- **Loops** - Iterating over data structures
- **Conditional tasks** - When clauses and control flow
- **Plays** - Task organization and execution
- **Handling task failure** - Error handling and recovery
- **Playbooks** - Multi-play automation workflows
- **Configuration files** - ansible.cfg setup and management
- **Roles** - Reusable automation components
- **Use provided documentation** - ansible-doc command proficiency

2. Use roles and Ansible Content Collections

- **Create and work with roles** - Role structure and development
- **Install roles and use them in playbooks** - ansible-galaxy integration
- **Install Content Collections** - Modern Ansible content management
- **Use Content Collections in playbooks** - FQCN and collection integration
- **Obtain content from collections** - Related roles, modules, and supplementary content

3. Install and configure an Ansible control node

- **Install required packages** - Ansible installation and dependencies
- **Create a static host inventory file** - INI and YAML inventory formats
- **Create a configuration file** - ansible.cfg customization

- **Create and use static inventories** - Host groups and group variables

4. Configure Ansible managed nodes

- **Create and distribute SSH keys** - Passwordless authentication setup
- **Configure privilege escalation** - sudo/become configuration
- **Deploy files to managed nodes** - File transfer and management
- **Analyze shell scripts and convert to playbooks** - Legacy automation migration

5. Run playbooks with Automation content navigator

- **Run playbooks with navigator** - Modern Ansible execution interface
- **Find new modules in Content Collections** - Module discovery and usage
- **Create inventories with navigator** - Dynamic inventory management
- **Configure Ansible environment** - Environment setup and management

6. Create Ansible plays and playbooks

- **Work with commonly used modules** - Core module proficiency
- **Use variables for command results** - Register and variable manipulation
- **Use conditionals for execution control** - When statements and logic
- **Configure error handling** - Failed_when, ignore_errors, and handlers
- **Create playbooks for system state** - Desired state configuration

7. Automate standard RHCSA tasks using Ansible modules

- **Software packages and repositories** - dnf/yum, repository management
- **Services** - systemd service control and configuration
- **Firewall rules** - firewalld automation
- **File systems** - Filesystem creation and mounting
- **Storage devices** - Disk and volume management
- **File content** - File creation, modification, and management
- **Archiving** - Backup and archive operations
- **Task scheduling** - Cron job automation
- **Security** - SELinux, file permissions, and access controls
- **Users and groups** - Account management automation

8. Manage content

- **Create and use templates** - Jinja2 templating for configuration files

- **Template customization** - Variable substitution and logic

Lab Environment Setup

Minimum Requirements

Control Node (ansible-controller):

- RHEL 9 with Ansible installed
- SSH client configured
- At least 2GB RAM, 20GB disk

Managed Nodes (2-3 systems):

- RHEL 9 systems
- SSH server running
- Python 3 installed
- At least 1GB RAM, 10GB disk each

Network Configuration

```
# Example network layout
Control Node:    192.168.1.10  (ansible-controller)
Managed Node 1:  192.168.1.11  (node1)
Managed Node 2:  192.168.1.12  (node2)
Managed Node 3:  192.168.1.13  (node3)
```

SSH Key Setup

```
# Generate SSH key on control node
ssh-keygen -t rsa -b 2048

# Copy to all managed nodes
ssh-copy-id user@node1
ssh-copy-id user@node2
ssh-copy-id user@node3

# Test connectivity
ansible all -m ping
```

Essential Ansible Installation

```
# Install Ansible on RHEL 9
sudo dnf install ansible-core

# Verify installation
ansible --version

# Install additional collections
ansible-galaxy collection install ansible.posix
ansible-galaxy collection install community.general
```

7-Week Study Plan

Week 1: Foundation

Goals: Environment setup and basic concepts

- Set up lab environment with control and managed nodes
- Complete Module 01: Ansible Basics
- Practice ad-hoc commands and simple playbooks
- **Daily Practice**: 1-2 hours

Week 2: Core Playbooks

Goals: Master playbook fundamentals

- Complete Module 02: Playbooks & Tasks
- Build multi-task playbooks with error handling
- Practice with 20+ common modules
- **Daily Practice**: 1-2 hours

Week 3: Variables and Control

Goals: Dynamic playbooks and logic

- Complete Module 03: Variables & Facts
- Complete Module 04: Task Control
- Build complex conditional playbooks
- **Daily Practice**: 1-2 hours

Week 4: Templates and Configuration

Goals: Configuration management mastery

- Complete Module 05: Templates
- Practice Jinja2 templating extensively
- Build complete configuration management playbooks
- **Daily Practice**: 1-2 hours

Week 5: Roles and Organization

Goals: Code reusability and structure

- Complete Module 06: Roles
- Create custom roles for common tasks
- Practice role dependencies and organization
- **Daily Practice**: 1-2 hours

Week 6: Security and Advanced Features

Goals: Security and troubleshooting

- Complete Module 07: Ansible Vault
- Complete Module 08: Advanced Features
- Practice vault integration and debugging
- **Daily Practice**: 1-2 hours

Week 7: Exam Preparation

Goals: Exam readiness and confidence

- Daily practice with timed exercises
- Review all quick reference materials
- Mock exam scenarios under time pressure
- **Daily Practice**: 2-3 hours

Key Success Factors

1. Hands-On Practice First

Wrong Approach: Reading documentation without practicing
Right Approach: Implement every example in your lab immediately

2. Master the Documentation

- Learn to use `ansible-doc module_name` efficiently
- Practice finding syntax without internet access
- Memorize locations of key information

3. Build Muscle Memory

- Practice common patterns until they're automatic
- Create personal cheat sheets for frequent tasks
- Type examples rather than copy-pasting

4. Time Management Skills

- Practice common tasks under time pressure
- Learn to prioritize high-value tasks first
- Develop troubleshooting workflows

5. Error Handling Expertise

- Master debugging failed playbooks quickly
- Learn common error patterns and solutions
- Practice systematic troubleshooting approaches

Common Pitfalls to Avoid

Time Management Mistakes

- Spending too much time on low-point tasks
- Not leaving time for testing and validation
- Getting stuck on debugging instead of moving forward

Technical Mistakes

- YAML syntax errors (indentation, quotes)
- Incorrect variable precedence assumptions
- Not testing playbooks before considering them complete
- Forgetting to handle error conditions

Exam Day Mistakes

- Not reading instructions carefully
- Making assumptions about the environment
- Not using available documentation effectively
- Panicking when encountering unexpected issues

Mental Preparation

Build Confidence Through Practice

1. **Start Simple**: Master basic tasks before complex ones
2. **Practice Under Pressure**: Set timers for your practice sessions
3. **Learn from Failures**: Every error teaches you something valuable
4. **Build a Toolkit**: Develop go-to patterns for common scenarios

Exam Day Mindset

- **Stay Calm**: Breathe and think through problems systematically
- **Read Carefully**: Understand exactly what's being asked
- **Test Everything**: Verify your solutions work completely
- **Move Forward**: Don't get stuck on any single problem

Essential Commands to Memorize

```
# Documentation and help
ansible-doc module_name
ansible-config list
ansible-inventory --list

# Testing and validation
ansible-playbook playbook.yml --check
ansible-playbook playbook.yml --syntax-check
ansible all -m setup

# Debugging
ansible-playbook playbook.yml -v
ansible-playbook playbook.yml --step

# Vault operations
ansible-vault create secrets.yml
ansible-vault edit secrets.yml
ansible-playbook playbook.yml --ask-vault-pass
```

Next Steps

Now that you understand the exam format and requirements:

1. **Set up your lab environment** following the specifications above
2. **Create a study schedule** based on the 7-week plan
3. **Start with Module 01: Ansible Basics** to build your foundation
4. **Practice daily** - consistency is key to success

Remember: The RHCE exam tests your ability to **do** Ansible automation, not just understand it. Focus on hands-on practice from day one!

Next Module: Module 01: Ansible Basics →

2.3 Module 01: Ansible Basics

Learning Objectives

By the end of this module, you will:

- Understand Ansible architecture and core concepts
- Install and configure Ansible on the control node
- Create and manage inventory files effectively
- Execute ad-hoc commands for system management
- Configure SSH authentication and privilege escalation
- Understand Ansible's connection methods and security model

Why Ansible for Red Hat Automation

Ansible Philosophy

Agentless Architecture: No software installation required on managed nodes **Idempotent Operations**: Safe to run multiple times, only makes necessary changes **Declarative Syntax**: Describe desired state, not step-by-step procedures **Human Readable**: YAML syntax that's easy to read and maintain

Key Benefits for System Administrators

- **Consistency**: Eliminate configuration drift across systems
- **Scalability**: Manage hundreds of systems as easily as one
- **Auditability**: Track all changes through version control
- **Reliability**: Built-in error handling and rollback capabilities

---

Ansible Architecture

Control Node Requirements

Supported Operating Systems:

- RHEL 8/9 (exam environment)
- CentOS Stream 8/9
- Fedora (recent versions)
- Ubuntu LTS versions

Python Requirements:

- Python 3.8 or newer
- pip for module installation

Software Requirements:

- SSH client
- ansible-core package

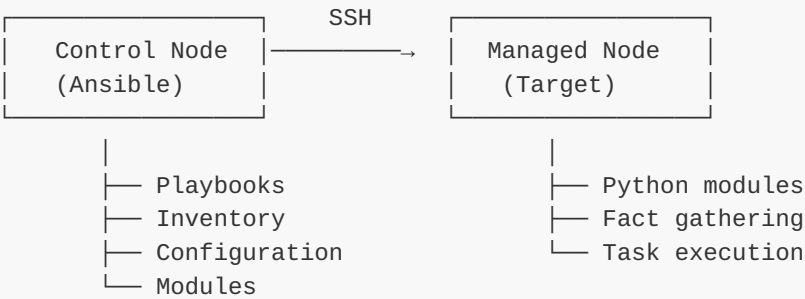
Managed Node Requirements

Minimal Requirements:

- SSH server running
- Python 3.6 or newer
- User account with appropriate privileges

No Agent Required: Ansible connects via SSH and executes tasks remotely

Communication Flow



Installation and Configuration

Installing Ansible on RHEL 9

```
# Enable required repositories
sudo subscription-manager repos --enable codeready-builder-for-rhel-9-$(arch)-rpms

# Install ansible-core
sudo dnf install ansible-core

# Verify installation
ansible --version
ansible-config --version

# Install additional collections
ansible-galaxy collection install ansible.posix
ansible-galaxy collection install community.general
```

Alternative Installation Methods

```
# Install via pip (if needed)
pip3 install ansible-core

# Install from source (development)
git clone https://github.com/ansible/ansible.git
cd ansible
source ./hacking/env-setup
```

Ansible Configuration File

Configuration Hierarchy (highest to lowest precedence):

1. `ANSIBLE_CONFIG` environment variable
2. `ansible.cfg` in current directory
3. `~/.ansible.cfg` in user home directory
4. `/etc/ansible/ansible.cfg` system-wide

Essential Configuration (`ansible.cfg`):

```
[defaults]
# Inventory file location
inventory = inventory.ini

# Default user for connections
remote_user = ansible

# Disable host key checking for lab environments
host_key_checking = False

# Enable privilege escalation by default
become = True
become_method = sudo

# Role and collection paths
roles_path = ./roles
collections_paths = ./collections

# Connection settings
timeout = 30
forks = 5

# Logging
log_path = ./ansible.log

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

[ssh_connection]
ssh_args = -o ControlMaster=auto -o ControlPersist=60s
control_path_dir = /tmp/.ansible-cp
```

Validating Configuration

```
# List all configuration settings
ansible-config list

# View current configuration values
ansible-config view

# Dump all configuration (active settings)
ansible-config dump
```

Inventory Management

Static Inventory Formats

INI Format (`inventory.ini`):

```
# Individual hosts
web01.example.com
web02.example.com ansible_host=192.168.1.10

# Groups
[webservers]
web01.example.com
web02.example.com

[databases]
db01.example.com
db02.example.com ansible_host=192.168.1.20

# Group of groups
[production:children]
webservers
databases

# Group variables
[webservers:vars]
http_port=80
max_clients=200

[databases:vars]
mysql_port=3306
mysql_datadir=/var/lib/mysql

# Global variables
[all:vars]
ansible_user=ansible
ansible_ssh_private_key_file=~/.ssh/id_rsa
```

YAML Format (`inventory.yml`):

```
all:
  children:
    production:
      children:
        webservers:
          hosts:
            web01.example.com:
            web02.example.com:
              ansible_host: 192.168.1.10
          vars:
            http_port: 80
            max_clients: 200
        databases:
          hosts:
            db01.example.com:
            db02.example.com:
              ansible_host: 192.168.1.20
          vars:
            mysql_port: 3306
            mysql_datadir: /var/lib/mysql
  vars:
    ansible_user: ansible
    ansible_ssh_private_key_file: ~/.ssh/id_rsa
```

Host Patterns

Basic Patterns:

```
# All hosts
ansible all -m ping

# Single host
ansible web01.example.com -m ping

# Group
ansible webservers -m ping

# Multiple groups
ansible webservers:databases -m ping

# Exclusions
ansible webservers:!web01.example.com -m ping

# Intersections
ansible webservers:&production -m ping

# Regular expressions
ansible ~web\d+ -m ping
```

Inventory Variables

Host Variables (`host_vars/hostname.yml`):

```
# host_vars/web01.example.com.yml
---
max_clients: 150
custom_config: true
ssl_enabled: yes
```

Group Variables (`group_vars/groupname.yml`):

```
# group_vars/webservers.yml
---
http_port: 80
document_root: /var/www/html
package_name: httpd

# group_vars/production.yml
---
environment: production
backup_enabled: true
monitoring_enabled: true
```

Inventory Validation

```
# List all hosts
ansible-inventory --list

# List hosts in YAML format
ansible-inventory --list --yaml

# Show specific host details
ansible-inventory --host web01.example.com

# Graphical representation
ansible-inventory --graph

# Validate inventory syntax
ansible-inventory --list > /dev/null
```

SSH Configuration and Authentication

SSH Key Generation and Distribution

```
# Generate SSH key pair
ssh-keygen -t rsa -b 4096 -N "" -f ~/.ssh/id_rsa

# Copy public key to managed nodes
ssh-copy-id ansible@web01.example.com
ssh-copy-id ansible@web02.example.com
ssh-copy-id ansible@db01.example.com

# Alternative: Manual key copying
cat ~/.ssh/id_rsa.pub | ssh ansible@web01.example.com "mkdir -p ~/.ssh && cat >> ~/.ssh/
authorized_keys"
```

SSH Client Configuration

SSH Config (`~/.ssh/config`):

```
Host web01.example.com
    User ansible
    IdentityFile ~/.ssh/id_rsa
    StrictHostKeyChecking no

Host 192.168.1.*
    User ansible
    IdentityFile ~/.ssh/id_rsa
    StrictHostKeyChecking no
    UserKnownHostsFile /dev/null
```

Testing Connectivity

```
# Test SSH connectivity manually
ssh ansible@web01.example.com "echo 'SSH connection successful'"

# Test Ansible connectivity
ansible all -m ping

# Test with specific user
ansible all -m ping -u ansible

# Test with different SSH key
ansible all -m ping --private-key ~/.ssh/alternate_key
```

Privilege Escalation Configuration

Sudo Configuration (on managed nodes):

```
# Add ansible user to sudoers
echo "ansible ALL=(ALL) NOPASSWD: ALL" | sudo tee /etc/sudoers.d/ansible

# Validate sudoers configuration
sudo visudo -c
```

Ansible Configuration:

```
# Test privilege escalation
ansible all -m shell -a "whoami" --become

# Test without password prompt
ansible all -m shell -a "whoami" --become --ask-become-pass
```

Ad-hoc Commands

Command Structure

Basic Syntax:

```
ansible <pattern> -m <module> -a "<module_arguments>" [options]
```

Essential Ad-hoc Command Patterns

System Information:

```
# Basic connectivity test
ansible all -m ping

# Gather system facts
ansible all -m setup

# Get specific facts
ansible all -m setup -a "filter=ansible_distribution*"

# Check uptime
ansible all -m command -a "uptime"

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

# View memory usage
ansible all -m shell -a "free -m"
```

Package Management:

```
# Install packages
ansible all -m dnf -a "name=httpd state=present" --become

# Install multiple packages
ansible all -m dnf -a "name=['httpd','mysql','php'] state=present" --become

# Update all packages
ansible all -m dnf -a "name='*' state=latest" --become

# Remove packages
ansible all -m dnf -a "name=httpd state=absent" --become
```

Service Management:

```
# Start services
ansible all -m systemd -a "name=httpd state=started" --become

# Enable and start services
ansible all -m systemd -a "name=httpd state=started enabled=yes" --become

# Restart services
ansible all -m systemd -a "name=httpd state=restarted" --become

# Check service status
ansible all -m systemd -a "name=httpd" --become
```

File Operations:

```
# Copy files to managed nodes
ansible all -m copy -a "src=/etc/hosts dest=/tmp/hosts" --become

# Create directories
ansible all -m file -a "path=/tmp/testdir state=directory mode=0755" --become

# Change file permissions
ansible all -m file -a "path=/tmp/testfile mode=0644 owner=root group=root" --become

# Create symbolic links
ansible all -m file -a "src=/tmp/source dest=/tmp/link state=link" --become

# Remove files
ansible all -m file -a "path=/tmp/testfile state=absent" --become
```

User Management:

```
# Create users
ansible all -m user -a "name=testuser groups=wheel shell=/bin/bash" --become

# Set user passwords (with encrypted password)
ansible all -m user -a "name=testuser password=$6$rounds=..." --become

# Remove users
ansible all -m user -a "name=testuser state=absent remove=yes" --become
```

Command Options

Common Options:

```
--become (-b)           # Enable privilege escalation
--become-user USER      # Escalate to specific user
--become-method METHOD  # Escalation method (sudo, su, etc.)
--ask-become-pass (-K)  # Prompt for escalation password
--check (-C)            # Dry run mode
--diff (-D)             # Show file changes
--limit SUBSET          # Limit to subset of hosts
--user (-u) USER        # Connect as specific user
--private-key FILE      # Use specific SSH private key
--ask-pass (-k)         # Prompt for SSH password
--inventory (-i) FILE   # Use specific inventory
--extra-vars (-e) VARS  # Set additional variables
--verbose (-v)          # Increase verbosity
```

Examples with Options:

```
# Dry run with diff output
ansible webservers -m copy -a "src=index.html dest=/var/www/html/" --check --diff --become

# Limit to specific hosts
ansible all -m shell -a "hostname" --limit "web01.example.com,web02.example.com"

# Use different user and key
ansible all -m ping -u root --private-key ~/.ssh/root_key

# Extra verbosity for debugging
ansible all -m setup --limit web01.example.com -vvv
```

Understanding Ansible Modules

Module Categories

Core System Modules:

- `ansible.builtin.command` - Execute commands
- `ansible.builtin.shell` - Execute shell commands
- `ansible.builtin.script` - Execute scripts
- `ansible.builtin.raw` - Execute raw SSH commands

Package Management:

- `ansible.builtin.dnf` - DNF/YUM package manager
- `ansible.builtin.package` - Generic package manager
- `ansible.builtin.rpm_key` - RPM key management

Service Management:

- `ansible.builtin.systemd` - Systemd service management
- `ansible.builtin.service` - Generic service management

File Operations:

- `ansible.builtin.copy` - Copy files
- `ansible.builtin.file` - File/directory management
- `ansible.builtin.template` - Jinja2 templating
- `ansible.builtin.lineinfile` - Line-in-file editing
- `ansible.builtin.replace` - File content replacement

Module Documentation

```
# List all available modules
ansible-doc -l

# Get module documentation
ansible-doc dnf
ansible-doc systemd
ansible-doc copy

# Get module syntax only
ansible-doc -s dnf

# Search modules
ansible-doc -l | grep firewall

# Get module examples
ansible-doc dnf | grep -A 20 EXAMPLES
```

Command vs Shell vs Raw Modules

command module (default, secure):

```
# Cannot use pipes, redirects, or shell variables
ansible all -m command -a "ls -l /tmp"
```

shell module (allows shell features):

```
# Can use pipes, redirects, and shell variables
ansible all -m shell -a "ps aux | grep httpd"
ansible all -m shell -a "echo $HOME"
```

raw module (minimal processing):

```
# Bypasses module system entirely
ansible all -m raw -a "uptime"
```

Practical Lab Exercises

Exercise 1: Control Node Setup

1. **Install Ansible on control node**
2. **Create basic ansible.cfg configuration**
3. **Test installation with version commands**

Exercise 2: SSH Authentication Setup

1. **Generate SSH key pair**
2. **Distribute keys to managed nodes**
3. **Configure SSH client for automation**
4. **Test password-less authentication**

Exercise 3: Inventory Creation

1. **Create INI format inventory with groups**
2. **Create YAML format inventory**
3. **Add host and group variables**
4. **Validate inventory structure**

Exercise 4: Ad-hoc Command Practice

1. **Test connectivity with ping module**
2. **Gather facts from all nodes**
3. **Install and manage packages**
4. **Control services across nodes**
5. **Perform file operations**

Exercise 5: Module Exploration

1. **Use ansible-doc to explore modules**
2. **Test different command modules**
3. **Practice with various module options**
4. **Compare module behaviors**

Key Takeaways

Architecture Understanding

- **Agentless**: No software required on managed nodes
- **SSH-based**: Secure communication using existing SSH infrastructure
- **Python execution**: Modules run Python code on managed nodes
- **Idempotent**: Safe to run repeatedly

Configuration Mastery

- **ansible.cfg precedence**: Know where Ansible looks for configuration
- **Essential settings**: remote_user, host_key_checking, become settings
- **Inventory formats**: Both INI and YAML, choose based on complexity
- **Variable organization**: Use host_vars and group_vars directories

Ad-hoc Command Proficiency

- **Module selection**: Choose appropriate modules for tasks
- **Option usage**: Master common options like --become, --check, --limit
- **Pattern matching**: Use flexible host patterns for targeting
- **Documentation**: Use ansible-doc for quick reference

Best Practices

- **Start simple**: Master basic concepts before advanced features
- **Test first**: Use ping and check mode to validate before execution
- **Document decisions**: Use clear naming and organization
- **Security focus**: Proper SSH key management and privilege escalation

---

Next Steps

You now have the foundation for Ansible automation. Next, you'll learn to:

1. **Module 02: Playbooks & Tasks** - Structure automation with playbooks
2. **Create repeatable automation** with YAML playbooks
3. **Organize complex tasks** with proper structure and error handling
4. **Scale beyond ad-hoc commands** to comprehensive automation workflows

The fundamentals you've learned here will support everything else you do with Ansible!

Next Module: Module 02: Playbooks & Tasks →

2.4 Module 02: Playbooks & Tasks

Learning Objectives

By the end of this module, you will:

- Create well-structured Ansible playbooks using YAML syntax
- Understand playbook components and execution flow
- Master task definition and module usage with FQCN
- Implement handlers for service and configuration management
- Use tags for selective task execution
- Handle errors and implement proper testing workflows
- Execute playbooks with ansible-navigator

Why Playbooks Matter

Beyond Ad-hoc Commands

Ad-hoc Commands: Great for quick, one-time tasks **Playbooks**: Essential for:

- Complex multi-step automation
- Repeatable infrastructure management
- Configuration management
- Application deployment
- Documented automation workflows

Playbook Benefits

- **Declarative**: Describe desired end state
- **Idempotent**: Safe to run multiple times
- **Reusable**: Version controlled automation
- **Readable**: Self-documenting YAML format
- **Scalable**: Handle simple to complex scenarios

YAML Fundamentals for Ansible

YAML Syntax Essentials

Basic Structure:

```
---  # Document start marker (optional but recommended)
# This is a comment
key: value
string_value: "quoted strings when needed"
number_value: 42
boolean_value: true
null_value: null

# Lists
fruits:
  - apple
  - banana
  - orange

# Alternative list syntax
fruits: [apple, banana, orange]

# Dictionaries
person:
  name: John
  age: 30
  active: true

# Alternative dictionary syntax
person: {name: John, age: 30, active: true}
```

YAML Best Practices for Ansible

Indentation Rules:

- Use 2 spaces for indentation (not tabs)
- Be consistent throughout the file
- Align items at the same level

Quoting Guidelines:

```
# Quote when necessary
shell_command: "echo 'Hello World'"
path_with_spaces: "/path/with spaces/file.txt"
special_chars: "String with: colons, {braces}, and [brackets]"

# Variables always need quotes in certain contexts
when: "ansible_distribution == 'RedHat'"
```

Multi-line Strings:

```
# Literal scalar (preserves line breaks)
script_content: |
  #!/bin/bash
  echo "Line 1"
  echo "Line 2"

# Folded scalar (joins lines)
description: >
  This is a long description
  that will be folded into
  a single line.
```

Common YAML Pitfalls

```
# Wrong - inconsistent indentation
tasks:
  - name: Task 1
    command: echo "hello"
    - name: Task 2  # Wrong indentation
      command: echo "world"

# Right - consistent indentation
tasks:
  - name: Task 1
    command: echo "hello"
  - name: Task 2
    command: echo "world"

# Wrong - missing quotes
when: variable == yes  # Should be "yes"

# Right - proper quoting
when: variable == "yes"
```

Playbook Structure

Basic Playbook Anatomy

```
---
- name: Descriptive playbook name
  hosts: target_hosts
  become: yes
  gather_facts: yes
  vars:
    variable_name: value
  vars_files:
    - vars/external_vars.yml
  tasks:
    - name: Descriptive task name
      module_name:
        parameter: value
        another_parameter: "{{ variable_name }}"
      notify: handler_name
      tags: tag_name
  handlers:
    - name: handler_name
      module_name:
        parameter: value
  roles:
    - role_name
```

Play-Level Keywords

Essential Play Keywords:

```
---
- name: Web server configuration              # Play description
  hosts: webservers                           # Target hosts/groups
  become: yes                                 # Privilege escalation
  become_user: root                           # Escalation target user
  become_method: sudo                         # Escalation method
  gather_facts: yes                           # Collect system facts
  serial: 2                                   # Batch execution size
  max_fail_percentage: 20                     # Failure tolerance
  remote_user: ansible                        # Connection user
  connection: ssh                             # Connection plugin
  timeout: 300                                # Task timeout
  vars:                                       # Play variables
    http_port: 80
  vars_files:                                 # External variable files
    - vars/web_vars.yml
  vars_prompt:                                # Interactive variables
    - name: target_env
      prompt: "Which environment?"
      private: no
```

Multi-Play Playbooks

```
---
# Play 1: Database setup
- name: Configure database servers
  hosts: databases
  become: yes
  tasks:
    - name: Install MySQL
      ansible.builtin.dnf:
        name: mysql-server
        state: present

# Play 2: Web server setup
- name: Configure web servers
  hosts: webservers
  become: yes
  tasks:
    - name: Install Apache
      ansible.builtin.dnf:
        name: httpd
        state: present

# Play 3: Load balancer setup
- name: Configure load balancers
  hosts: loadbalancers
  become: yes
  tasks:
    - name: Install HAProxy
      ansible.builtin.dnf:
        name: haproxy
        state: present
```

Task Definition and Module Usage

Task Anatomy

```
- name: Descriptive task name                # Required: Human readable description
  ansible.builtin.module_name:               # Required: FQCN module name
    parameter1: value1                       # Module-specific parameters
    parameter2: "{{ variable_name }}"
    parameter3:
      - list_item1
      - list_item2
  register: result_variable                  # Capture task output
  when: condition                            # Conditional execution
  loop: "{{ items_list }}"                   # Iteration
  failed_when: custom_failure_condition      # Custom failure criteria
  changed_when: custom_change_condition      # Custom change detection
  ignore_errors: yes                         # Continue on failure
  become: yes                                # Task-level privilege escalation
  become_user: specific_user                 # Task-level escalation user
  delegate_to: other_host                    # Execute on different host
  run_once: yes                              # Execute only once in batch
  tags:                                      # Task tags
    - tag1
    - tag2
  notify:                                    # Trigger handlers
    - handler_name
```

Fully Qualified Collection Names (FQCN)

Modern Ansible Requirement: All modules must use FQCN format

```
# Correct - FQCN format
tasks:
  - name: Install packages
    ansible.builtin.dnf:          # Not just 'dnf'
      name: httpd
      state: present

  - name: Configure firewall
    ansible.posix.firewalld:      # Not just 'firewalld'
      service: http
      permanent: yes
      state: enabled

  - name: Create partition
    community.general.parted:     # Not just 'parted'
      device: /dev/sdb
      number: 1
      state: present
```

Essential FQCN Module Categories:

Category	Collection	Example Modules
Core System	`ansible.builtin`	`dnf`, `systemd`, `copy`, `file`, `user`
POSIX Tools	`ansible.posix`	`firewalld`, `mount`, `seboolean`, `authorized_key`
Extended	`community.general`	`parted`, `lvg`, `lvol`, `htpasswd`

Common Module Patterns

Package Management:

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

- name: Install multiple packages
  ansible.builtin.dnf:
    name:
      - httpd
      - php
      - mysql-server
    state: present

- name: Install package from specific repository
  ansible.builtin.dnf:
    name: nginx
    state: present
    enablerepo: epel

- name: Update all packages
  ansible.builtin.dnf:
    name: '*'
    state: latest
```

Service Management:

```
- name: Start and enable service
  ansible.builtin.systemd:
    name: httpd
    state: started
    enabled: yes
    daemon_reload: yes

- name: Restart service
  ansible.builtin.systemd:
    name: httpd
    state: restarted

- name: Stop and disable service
  ansible.builtin.systemd:
    name: httpd
    state: stopped
    enabled: no
```

File Operations:

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

- name: Create directory
  ansible.builtin.file:
    path: /path/to/directory
    state: directory
    mode: '0755'
    owner: apache
    group: apache

- name: Create symbolic link
  ansible.builtin.file:
    src: /path/to/source
    dest: /path/to/link
    state: link

- name: Remove file or directory
  ansible.builtin.file:
    path: /path/to/remove
    state: absent
```

User Management:

```
- name: Create user account
  ansible.builtin.user:
    name: webuser
    groups:
      - apache
      - wheel
    shell: /bin/bash
    create_home: yes
    state: present

- name: Set user password
  ansible.builtin.user:
    name: webuser
    password: "{{ 'plaintext_password' | password_hash('sha512') }}"
    update_password: always
```

Handlers

Handler Concepts

Purpose: Execute tasks only when notified by changed tasks **Common Use Cases**:

- Restart services after configuration changes
- Reload configurations
- Run cleanup tasks

Handler Syntax

```
---
- name: Configure web service
  hosts: webservers
  become: yes
  tasks:
    - name: Install Apache
      ansible.builtin.dnf:
        name: httpd
        state: present
      notify: start apache

    - name: Copy configuration file
      ansible.builtin.copy:
        src: httpd.conf
        dest: /etc/httpd/conf/httpd.conf
        backup: yes
      notify:
        - restart apache
        - reload firewall

  handlers:
    - name: start apache
      ansible.builtin.systemd:
        name: httpd
        state: started
        enabled: yes

    - name: restart apache
      ansible.builtin.systemd:
        name: httpd
        state: restarted

    - name: reload firewall
      ansible.builtin.systemd:
        name: firewalld
        state: reloaded
```

Handler Execution Rules

Key Behaviors:

- Handlers run at the end of each play
- Handlers only run if notified by a changed task
- Each handler runs only once, even if notified multiple times
- Handlers run in the order defined, not notification order
- Failed tasks prevent handler execution

Force Handler Execution:

```
- name: Force handlers to run immediately
  ansible.builtin.meta: flush_handlers
```

Advanced Handler Patterns

```
---
- name: Multi-service configuration
  hosts: all
  become: yes
  tasks:
    - name: Update web server config
      ansible.builtin.template:
        src: httpd.conf.j2
        dest: /etc/httpd/conf/httpd.conf
      notify: restart web services

    - name: Update database config
      ansible.builtin.template:
        src: mysql.cnf.j2
        dest: /etc/mysql/mysql.conf.d/custom.cnf
      notify: restart database services

  handlers:
    - name: restart web services
      ansible.builtin.systemd:
        name: "{{ item }}"
        state: restarted
      loop:
        - httpd
        - php-fpm

    - name: restart database services
      ansible.builtin.systemd:
        name: "{{ item }}"
        state: restarted
      loop:
        - mysql
        - redis
```

Tags for Task Organization

Tag Usage

Purpose: Selective task execution without running entire playbook

```
---
- name: Complete system setup
  hosts: all
  become: yes
  tasks:
    - name: Install packages
      ansible.builtin.dnf:
        name: "{{ item }}"
        state: present
      loop:
        - httpd
        - mysql-server
        - php
      tags:
        - packages
        - install

    - name: Configure services
      ansible.builtin.template:
        src: "{{ item.src }}"
        dest: "{{ item.dest }}"
      loop:
        - {src: httpd.conf.j2, dest: /etc/httpd/conf/httpd.conf}
        - {src: mysql.cnf.j2, dest: /etc/mysql/mysql.conf.d/custom.cnf}
      tags:
        - config
        - templates

    - name: Start services
      ansible.builtin.systemd:
        name: "{{ item }}"
        state: started
        enabled: yes
      loop:
        - httpd
        - mysql
      tags:
        - services
        - start
```

Tag Execution Options

```
# Run only specific tags
ansible-navigator run site.yml --tags "packages,config" --mode stdout

# Skip specific tags
ansible-navigator run site.yml --skip-tags "services" --mode stdout

# List available tags
ansible-navigator run site.yml --list-tags --mode stdout

# Run multiple tag sets
ansible-navigator run site.yml --tags "install" --tags "config" --mode stdout
```

Special Tags

```
tasks:
  - name: Always run this task
    ansible.builtin.debug:
      msg: "This always executes"
    tags: always

  - name: Never run this task by default
    ansible.builtin.debug:
      msg: "Only runs when explicitly called"
    tags: never
```

Error Handling

Basic Error Control

Failed When: Define custom failure conditions

```
- name: Check web service response
  ansible.builtin.uri:
    url: "http://{{ inventory_hostname }}/health"
    method: GET
  register: health_check
  failed_when:
    - health_check.status != 200
    - "'healthy' not in health_check.content"
```

Changed When: Define when tasks register as changed

```
- name: Run application deployment
  ansible.builtin.command: /opt/app/deploy.sh
  register: deploy_result
  changed_when: "'deployed' in deploy_result.stdout"
```

Ignore Errors: Continue despite failures

```
- name: Attempt optional configuration
  ansible.builtin.copy:
    src: optional_config.conf
    dest: /etc/app/optional.conf
  ignore_errors: yes
```

Block Error Handling

```
- name: Web server deployment with error handling
  block:
    - name: Install web server
      ansible.builtin.dnf:
        name: httpd
        state: present

    - name: Start web server
      ansible.builtin.systemd:
        name: httpd
        state: started

    - name: Test web server
      ansible.builtin.uri:
        url: "http://{{ inventory_hostname }}"
        status_code: 200

  rescue:
    - name: Log deployment failure
      ansible.builtin.debug:
        msg: "Web server deployment failed, attempting rollback"

    - name: Stop failed service
      ansible.builtin.systemd:
        name: httpd
        state: stopped
      ignore_errors: yes

    - name: Remove failed installation
      ansible.builtin.dnf:
        name: httpd
        state: absent

  always:
    - name: Clean temporary files
      ansible.builtin.file:
        path: /tmp/deployment.*
        state: absent
```

Playbook Testing and Execution

Syntax Validation

```
# Check playbook syntax
ansible-navigator run site.yml --syntax-check

# Validate without execution
ansible-navigator run site.yml --check --mode stdout

# Show what would change
ansible-navigator run site.yml --check --diff --mode stdout
```

Execution Modes

```
# Standard execution
ansible-navigator run site.yml --mode stdout

# Verbose output
ansible-navigator run site.yml --mode stdout -v

# Maximum verbosity for debugging
ansible-navigator run site.yml --mode stdout -vvv

# Interactive TUI mode
ansible-navigator run site.yml

# Limit to specific hosts
ansible-navigator run site.yml --limit webservers --mode stdout

# Start at specific task
ansible-navigator run site.yml --start-at-task "Configure Apache" --mode stdout
```

Testing Workflow

Recommended Test Sequence:

```
# 1. Syntax check
ansible-navigator run site.yml --syntax-check

# 2. Dry run
ansible-navigator run site.yml --check --mode stdout

# 3. Limited execution (single host)
ansible-navigator run site.yml --limit web01 --mode stdout

# 4. Full execution
ansible-navigator run site.yml --mode stdout

# 5. Idempotency test (should show no changes)
ansible-navigator run site.yml --mode stdout
```

Practical Lab Exercises

Exercise 1: Basic Playbook Creation

Create a playbook that:

1. Installs Apache web server
2. Copies a custom index.html file
3. Starts and enables the service
4. Uses a handler to restart on config change

```
---
- name: Web server setup
  hosts: webservers
  become: yes
  tasks:
    - name: Install Apache
      ansible.builtin.dnf:
        name: httpd
        state: present
      notify: start apache

    - name: Copy index page
      ansible.builtin.copy:
        content: |
          <html>
          <head><title>Welcome</title></head>
          <body><h1>Ansible Managed Server</h1></body>
          </html>
        dest: /var/www/html/index.html
        owner: apache
        group: apache
        mode: '0644'

    - name: Configure firewall
      ansible.posix.firewalld:
        service: http
        permanent: yes
        state: enabled
        immediate: yes

  handlers:
    - name: start apache
      ansible.builtin.systemd:
        name: httpd
        state: started
        enabled: yes
```

Exercise 2: Error Handling Practice

Create a playbook with:

1. Block/rescue/always structure
2. Custom failed_when conditions
3. Ignore_errors usage
4. Proper error recovery

Exercise 3: Multi-Play Workflow

Create a multi-play playbook that:

1. Configures database servers (first play)
2. Configures web servers (second play)
3. Configures load balancers (third play)
4. Uses facts from previous plays

Exercise 4: Tag Organization

Create a comprehensive playbook with:

1. Install, configure, and service tags
2. Environment-specific tags (dev, prod)
3. Practice selective execution

Key Takeaways

Playbook Structure Mastery

- **YAML syntax**: Proper indentation and quoting
- **Play organization**: Logical grouping of related tasks
- **FQCN requirement**: Always use fully qualified collection names
- **Documentation**: Clear, descriptive names for plays and tasks

Task Design Principles

- **Idempotency**: Tasks should be safe to run repeatedly
- **Atomicity**: Each task should accomplish one specific goal

- **Clarity**: Task names should clearly describe the action
- **Error handling**: Anticipate and handle failure scenarios

Handler Usage

- **Event-driven**: Only run when notified by changed tasks
- **Service management**: Ideal for service restarts and reloads
- **Execution timing**: Understand when handlers run
- **Notification patterns**: Multiple tasks can notify same handler

Testing Best Practices

- **Syntax validation**: Always check syntax before execution
- **Dry run testing**: Use --check mode to validate logic
- **Limited testing**: Test on single host before full deployment
- **Idempotency verification**: Run twice to ensure no unexpected changes

---

Next Steps

With solid playbook skills established, you're ready to advance to:

1. **Module 03: Variables & Facts** - Dynamic playbooks with variables
2. **Dynamic content** with variable substitution and fact usage
3. **Complex logic** with variable precedence and scoping
4. **Data-driven automation** using external variable sources

Your playbook foundation will support all advanced Ansible features!

Next Module: Module 03: Variables & Facts →

2.5 Module 03: Variables & Facts

Learning Objectives

By the end of this module, you will:

- Understand variable types, scoping, and precedence rules
- Master fact gathering and utilization in playbooks
- Organize variables using host_vars and group_vars
- Use magic variables for advanced automation logic
- Register task outputs for use in subsequent tasks
- Implement variable inheritance and override patterns
- Debug variable content and troubleshoot precedence issues

Why Variables Transform Automation

Static vs Dynamic Playbooks

Static Approach: Hard-coded values limit reusability

```
# Limited reusability
- name: Install web server
  ansible.builtin.dnf:
    name: httpd              # Hard-coded package name
    state: present
```

Dynamic Approach: Variables enable flexibility

```
# Highly reusable
- name: Install web server
  ansible.builtin.dnf:
    name: "{{ web_package }}" # Variable package name
    state: present
```

Variable Benefits

- **Flexibility**: Same playbook works across environments
- **Maintainability**: Change values in one place
- **Reusability**: Generic playbooks for multiple scenarios
- **Security**: Sensitive data separation via Ansible Vault

Variable Types and Definition

Variable Naming Rules

Valid Names:

- Letters, numbers, and underscores only
- Must start with letter or underscore
- Case-sensitive

```
# Valid variable names
web_package: httpd
db_port: 3306
_private_key: /path/to/key
environment2: production

# Invalid variable names
# 2environment: production    # Cannot start with number
# web-package: httpd          # Cannot contain hyphens
# web package: httpd          # Cannot contain spaces
```

Variable Definition Locations

1. Playbook Variables:

```
---
- name: Web server setup
  hosts: webservers
  vars:
    web_package: httpd
    web_port: 80
    web_user: apache
  tasks:
    - name: Install {{ web_package }}
      ansible.builtin.dnf:
        name: "{{ web_package }}"
        state: present
```

2. External Variable Files:

```
# vars/web_vars.yml
---
web_package: httpd
web_port: 80
web_service: httpd
web_config_dir: /etc/httpd/conf
web_document_root: /var/www/html

# Playbook usage
---
- name: Web server setup
  hosts: webservers
  vars_files:
    - vars/web_vars.yml
  tasks:
    - name: Install web server
      ansible.builtin.dnf:
        name: "{{ web_package }}"
        state: present
```

3. Command Line Variables:

```
# Override variables at runtime
ansible-navigator run site.yml -e "web_port=8080" --mode stdout
ansible-navigator run site.yml -e "env=production" --mode stdout
ansible-navigator run site.yml -e "@vars/production.yml" --mode stdout
```

4. Host Variables (`host_vars/hostname.yml`):

```
# host_vars/web01.example.com.yml
---
web_port: 8080
max_clients: 150
ssl_enabled: true
custom_modules:
  - mod_ssl
  - mod_rewrite
```

5. Group Variables (`group_vars/groupname.yml`):

```
# group_vars/webservers.yml
---
web_package: httpd
web_service: httpd
web_port: 80
document_root: /var/www/html

# group_vars/production.yml
---
environment: production
backup_enabled: true
monitoring: true
log_level: warn
```

6. Inventory Variables:

```
# INI format
[webservers]
web01.example.com web_port=8080 ssl_enabled=yes
web02.example.com web_port=80 ssl_enabled=no

[webservers:vars]
web_package=httpd
web_service=httpd
```

```
# YAML format
all:
  children:
    webservers:
      hosts:
        web01.example.com:
          web_port: 8080
          ssl_enabled: yes
        web02.example.com:
          web_port: 80
          ssl_enabled: no
      vars:
        web_package: httpd
        web_service: httpd
```

Variable Data Types

Strings:

```
server_name: web01.example.com
config_path: "/etc/httpd/conf/httpd.conf"
message: 'Single quotes preserve content literally'
```

Numbers:

```
web_port: 80
max_connections: 1000
timeout: 30.5
```

Booleans:

```
ssl_enabled: true
debug_mode: false
backup_enabled: yes    # Ansible converts to boolean
maintenance: no        # Ansible converts to boolean
```

Lists:

```
packages:
  - httpd
  - php
  - mysql-server

ports: [80, 443, 8080]

users:
  - name: alice
    shell: /bin/bash
  - name: bob
    shell: /bin/zsh
```

Dictionaries:

```
database:
  host: db.example.com
  port: 3306
  name: webapp
  user: appuser

# Alternative syntax
database: {host: db.example.com, port: 3306, name: webapp}
```

Variable Precedence (16 Levels)

Complete Precedence Hierarchy

Higher number = Higher precedence

1. **role defaults** (lowest precedence)
2. **inventory file or script group vars**
3. **inventory group_vars/all**
4. **playbook group_vars/all**
5. **inventory group_vars/***
6. **playbook group_vars/***
7. **inventory file or script host vars**
8. **inventory host_vars/***
9. **playbook host_vars/***
10. **host facts / cached set_facts**
11. **play vars**
12. **play vars_prompt**
13. **play vars_files**
14. **role vars (defined in role/vars/main.yml)**
15. **block vars (only for tasks in block)**
16. **task vars (only for the specific task)**
17. **include_vars**
18. **set_facts / registered vars**
19. **role (and include_role) params**
20. **include params**
21. **extra vars** (`ansible-navigator run -e`) **(highest precedence)**

Practical Precedence Examples

Scenario: Same variable defined at multiple levels

```
# group_vars/all.yml (precedence 3)
web_port: 80

# host_vars/web01.example.com.yml (precedence 8)
web_port: 8080

# play vars (precedence 11)
---
- name: Configure web server
  hosts: web01.example.com
  vars:
    web_port: 443
  tasks:
    - name: Show final port value
      ansible.builtin.debug:
        var: web_port          # Will show 443 (play vars win)
```

Command Line Override (highest precedence):

```
# Command line variables always win
ansible-navigator run site.yml -e "web_port=9090" --mode stdout
# web_port will be 9090 regardless of other definitions
```

Testing Variable Precedence

```
---
- name: Variable precedence demonstration
  hosts: all
  vars:
    test_var: "from play vars"
  tasks:
    - name: Show variable from multiple sources
      ansible.builtin.debug:
        msg: |
          test_var value: {{ test_var }}
          Source: {{ test_var_source | default('unknown') }}

    - name: Set task-level variable
      ansible.builtin.debug:
        var: test_var
      vars:
        test_var: "from task vars"    # This wins over play vars
```

Facts (System Information)

Understanding Facts

What are Facts?: Automatically gathered system information including:

- Operating system details
- Hardware information
- Network configuration
- Filesystem information
- Service status

Fact Gathering Control

```
---
- name: Control fact gathering
  hosts: all
  gather_facts: yes          # Default behavior
  tasks:
    - name: Manual fact gathering
      ansible.builtin.setup:  # Equivalent to automatic gathering

# Disable automatic fact gathering
- name: Skip fact gathering
  hosts: all
  gather_facts: no
  tasks:
    - name: Gather facts only when needed
      ansible.builtin.setup:
      when: gather_system_info | default(false)
```

Essential Facts

System Information:

```
- name: Display system facts
  ansible.builtin.debug:
    msg: |
      OS: {{ ansible_facts['distribution'] }} {{ ansible_facts['distribution_version'] }}
      Architecture: {{ ansible_facts['architecture'] }}
      Kernel: {{ ansible_facts['kernel'] }}
      Python: {{ ansible_facts['python_version'] }}
      Hostname: {{ ansible_facts['hostname'] }}
      FQDN: {{ ansible_facts['fqdn'] }}
```

Hardware Facts:

```
- name: Display hardware facts
  ansible.builtin.debug:
    msg: |
      Memory: {{ ansible_facts['memtotal_mb'] }} MB
      Processors: {{ ansible_facts['processor_count'] }}
      CPU: {{ ansible_facts['processor'][0] }}
      Virtualization: {{ ansible_facts['virtualization_type'] | default('none') }}
```

Network Facts:

```
- name: Display network facts
  ansible.builtin.debug:
    msg: |
      Default IP: {{ ansible_facts['default_ipv4']['address'] }}
      Default Interface: {{ ansible_facts['default_ipv4']['interface'] }}
      All IPs: {{ ansible_facts['all_ipv4_addresses'] | join(', ') }}
      Hostname: {{ ansible_facts['fqdn'] }}
```

Storage Facts:

```
- name: Display storage facts
  ansible.builtin.debug:
    msg: |
      Disks: {{ ansible_facts['devices'].keys() | list | join(', ') }}
      Root filesystem: {{ ansible_facts['mounts'] | selectattr('mount', 'equalto', '/') |
first }}
      Available space: {{ ansible_facts['mounts'] | selectattr('mount', 'equalto', '/') |
map(attribute='size_available') | first }}
```

Fact Filtering

```
# Gather specific fact subsets
ansible all -m setup -a "filter=ansible_distribution*"
ansible all -m setup -a "filter=ansible_default_ipv4"
ansible all -m setup -a "filter=ansible_mounts"

# Gather only essential facts
ansible all -m setup -a "gather_subset=hardware,network"

# Exclude specific fact subsets
ansible all -m setup -a "gather_subset=all,!facter,!ohai"
```

Custom Facts

Local Facts Directory: `/etc/ansible/facts.d/`

```
# Create custom fact script
sudo mkdir -p /etc/ansible/facts.d
sudo cat > /etc/ansible/facts.d/application.fact << 'EOF'
#!/bin/bash
echo '{
  "app_name": "myapp",
  "version": "1.2.3",
  "config_path": "/opt/myapp/config",
  "data_path": "/var/lib/myapp"
}'
EOF
sudo chmod +x /etc/ansible/facts.d/application.fact
```

Using Custom Facts:

```
- name: Use custom facts
  ansible.builtin.debug:
    msg: |
      App: {{ ansible_facts['local']['application']['app_name'] }}
      Version: {{ ansible_facts['local']['application']['version'] }}
```

Magic Variables

Built-in Magic Variables

Inventory Variables:

```
- name: Display inventory information
  ansible.builtin.debug:
    msg: |
      Current host: {{ inventory_hostname }}
      Short hostname: {{ inventory_hostname_short }}
      Groups this host belongs to: {{ group_names | join(', ') }}
      All groups: {{ groups.keys() | list | join(', ') }}
      Webserver hosts: {{ groups['webservers'] | default([]) | join(', ') }}
```

Play Variables:

```
- name: Display play information
  ansible.builtin.debug:
    msg: |
      Hosts in current play: {{ play_hosts | join(', ') }}
      Current batch: {{ ansible_play_batch | join(', ') }}
      Play name: {{ ansible_play_name | default('unnamed') }}
```

Hostvars Access:

```
- name: Access other host variables
  ansible.builtin.debug:
    msg: |
      Web01 IP: {{ hostvars['web01.example.com']['ansible_default_ipv4']['address'] }}
      Database host: {{ hostvars[groups['databases'][0]]['ansible_fqdn'] }}
      All web server IPs: {% for host in groups['webservers'] %}{{ hostvars[host]
['ansible_default_ipv4']['address'] }}{% if not loop.last %}, {% endif %}{% endfor %}
```

Advanced Magic Variable Usage

Cross-Host Information Sharing:

```
---
- name: Collect information from all hosts
  hosts: all
  tasks:
    - name: Register hostname info
      ansible.builtin.set_fact:
        server_info:
          hostname: "{{ inventory_hostname }}"
          ip: "{{ ansible_default_ipv4.address }}"
          role: "{{ server_role | default('unknown') }}"

- name: Configure load balancer with all server info
  hosts: loadbalancers
  tasks:
    - name: Generate backend configuration
      ansible.builtin.template:
        src: backends.conf.j2
        dest: /etc/haproxy/backends.conf
      vars:
        backend_servers: |
          {% for host in groups['webservers'] %}
          server {{ hostvars[host]['inventory_hostname_short'] }} {{ hostvars[host]
['ansible_default_ipv4']['address'] }}:80 check
          {% endfor %}
```

Registering Variables

Basic Registration

```
---
- name: Variable registration examples
  hosts: all
  tasks:
    - name: Check service status
      ansible.builtin.systemd:
        name: httpd
      register: service_status

    - name: Display service information
      ansible.builtin.debug:
        msg: |
          Service active: {{ service_status.status.ActiveState }}
          Service enabled: {{ service_status.status.UnitFileState }}
          Service running: {{ service_status.status.SubState }}
```

Command Registration

```
- name: Execute command and capture output
  ansible.builtin.command: whoami
  register: current_user

- name: Execute shell command with pipes
  ansible.builtin.shell: ps aux | grep httpd | wc -l
  register: httpd_processes

- name: Display command results
  ansible.builtin.debug:
    msg: |
      Current user: {{ current_user.stdout }}
      Apache processes: {{ httpd_processes.stdout }}
      Return code: {{ current_user.rc }}
      Command failed: {{ current_user.failed }}
```

Advanced Registration Patterns

Conditional Logic Based on Registration:

```
- name: Check web service availability
  ansible.builtin.uri:
    url: "http://{{ inventory_hostname }}/health"
    method: GET
    status_code: [200, 503]
  register: health_check
  ignore_errors: yes

- name: Restart service if unhealthy
  ansible.builtin.systemd:
    name: httpd
    state: restarted
  when: health_check.status == 503

- name: Report service status
  ansible.builtin.debug:
    msg: |
      Health check status: {{ health_check.status }}
      Service is {{ 'healthy' if health_check.status == 200 else 'unhealthy' }}
```

Loop Registration:

```
- name: Check multiple services
  ansible.builtin.systemd:
    name: "{{ item }}"
  register: service_results
  loop:
    - httpd
    - mysql
    - sshd

- name: Display all service states
  ansible.builtin.debug:
    msg: "{{ item.item }} is {{ item.status.ActiveState }}"
  loop: "{{ service_results.results }}"
```

Variable Organization Strategies

Directory Structure

```
inventory/
├── group_vars/
│   ├── all.yml                 # Variables for all hosts
│   ├── webservers.yml         # Web server group variables
│   ├── databases.yml          # Database group variables
│   └── production/            # Environment-specific variables
│       ├── all.yml
│       ├── webservers.yml
│       └── databases.yml
├── host_vars/
│   ├── web01.example.com.yml
│   ├── web02.example.com.yml
│   └── db01.example.com.yml
└── inventory.yml
```

Variable Organization Best Practices

Environment Separation:

```
# group_vars/all/common.yml
---
app_name: myapp
app_user: webapp
log_level: info

# group_vars/all/development.yml
---
environment: development
debug_enabled: true
log_level: debug
database_host: dev-db.internal.com

# group_vars/all/production.yml
---
environment: production
debug_enabled: false
log_level: warn
database_host: prod-db.internal.com
```

Service-Specific Variables:

```
# group_vars/webservers/apache.yml
---
apache:
  package: httpd
  service: httpd
  config_dir: /etc/httpd/conf
  document_root: /var/www/html
  modules:
    - mod_ssl
    - mod_rewrite
  ports:
    - 80
    - 443

# group_vars/databases/mysql.yml
---
mysql:
  package: mysql-server
  service: mysqld
  config_file: /etc/my.cnf
  data_dir: /var/lib/mysql
  port: 3306
  root_password: "{{ vault_mysql_root_password }}"
```

Variable Validation

```
---
- name: Validate required variables
  hosts: all
  tasks:
    - name: Ensure required variables are defined
      ansible.builtin.assert:
        that:
          - app_name is defined
          - environment is defined
          - database_host is defined
        fail_msg: "Required variables are not defined"
        success_msg: "All required variables are present"

    - name: Validate variable values
      ansible.builtin.assert:
        that:
          - environment in ['development', 'staging', 'production']
          - app_port | int > 1024
          - database_host | length > 0
        fail_msg: "Variable validation failed"
```

Variable Debugging

Debug Strategies

Basic Variable Display:

```
- name: Debug variable content
  ansible.builtin.debug:
    var: variable_name

- name: Debug with custom message
  ansible.builtin.debug:
    msg: "The value of web_port is {{ web_port }}"

- name: Debug multiple variables
  ansible.builtin.debug:
    msg: |
      Environment: {{ environment }}
      Web port: {{ web_port }}
      SSL enabled: {{ ssl_enabled }}
```

Fact Exploration:

```
- name: Display all facts
  ansible.builtin.debug:
    var: ansible_facts

- name: Display specific fact categories
  ansible.builtin.debug:
    var: ansible_facts['network']

- name: Search facts by pattern
  ansible.builtin.debug:
    msg: "{{ ansible_facts | dict2items | selectattr('key', 'match', 'ansible_eth.*') | list }}"
```

Variable Source Investigation:

```
- name: Check if variable is defined
  ansible.builtin.debug:
    msg: "web_port is {{ 'defined' if web_port is defined else 'undefined' }}"

- name: Show variable type
  ansible.builtin.debug:
    msg: "web_port is of type {{ web_port | type_debug }}"

- name: Display hostvars for debugging
  ansible.builtin.debug:
    var: hostvars[inventory_hostname]
```

Practical Lab Exercises

Exercise 1: Variable Precedence Testing

Create a comprehensive precedence test:

1. Define the same variable at multiple levels
2. Use debug tasks to show final values
3. Override with command-line variables
4. Document the results

```
# group_vars/all.yml
test_var: "from group_vars all"

# host_vars/testhost.yml
test_var: "from host_vars"

# Playbook
---
- name: Variable precedence test
  hosts: testhost
  vars:
    test_var: "from play vars"
  tasks:
    - name: Show variable value
      ansible.builtin.debug:
        var: test_var
      vars:
        test_var: "from task vars"
```

Exercise 2: Fact-Based Configuration

Create environment-specific configuration using facts:

```
---
- name: Configure based on system facts
  hosts: all
  tasks:
    - name: Install appropriate packages for OS
      ansible.builtin.dnf:
        name: "{{ packages[ansible_facts['distribution'] | lower] }}"
        state: present
      vars:
        packages:
          redhat:
            - httpd
            - php
          centos:
            - httpd
            - php
          fedora:
            - httpd
            - php
```

Exercise 3: Cross-Host Information Sharing

Build configuration files using information from multiple hosts:

```
---
- name: Gather web server information
  hosts: webservers
  tasks:
    - name: Set web server facts
      ansible.builtin.set_fact:
        web_server_info:
          name: "{{ inventory_hostname }}"
          ip: "{{ ansible_default_ipv4.address }}"
          port: "{{ web_port | default(80) }}"

- name: Configure load balancer
  hosts: loadbalancers
  tasks:
    - name: Generate load balancer config
      ansible.builtin.template:
        src: haproxy.cfg.j2
        dest: /etc/haproxy/haproxy.cfg
```

Exercise 4: Dynamic Variable Loading

Load variables based on conditions:

```
---
- name: Dynamic variable loading
  hosts: all
  tasks:
    - name: Load environment-specific variables
      ansible.builtin.include_vars: "vars/{{ environment }}.yml"

    - name: Load OS-specific variables
      ansible.builtin.include_vars: "vars/{{ ansible_facts['distribution'] | lower }}.yml"
```

Key Takeaways

Variable Management Excellence

- **Precedence understanding**: Know which variable definitions take priority
- **Organization strategy**: Use logical directory structures for variables
- **Data typing**: Understand how Ansible handles different data types
- **Validation**: Always validate critical variables before use

Fact Utilization Mastery

- **System intelligence**: Use facts to make decisions about configuration
- **Performance consideration**: Disable fact gathering when not needed
- **Custom facts**: Extend system information with application-specific data
- **Fact filtering**: Gather only needed information for efficiency

Magic Variable Proficiency

- **Inventory access**: Use magic variables to access host and group information
- **Cross-host communication**: Share data between hosts using hostvars
- **Play context**: Understand play-level variables and their scope
- **Dynamic targeting**: Use magic variables for dynamic host selection

Registration and Debugging Skills

- **Output capture**: Register command and module outputs for later use
- **Conditional logic**: Make decisions based on registered results

- **Debugging techniques**: Efficiently troubleshoot variable issues
- **Testing strategies**: Validate variable behavior during development

---

Next Steps

With solid variable and fact mastery, you're ready for:

1. **Module 04: Task Control** - Advanced logic with conditionals and loops
2. **Complex automation logic** using when statements and loops
3. **Error handling strategies** with blocks and failure management
4. **Advanced task delegation** and execution control

Your variable skills enable sophisticated automation scenarios!

Next Module: Module 04: Task Control →

2.6 Module 04: Task Control

Learning Objectives

By the end of this module, you will:

- Master conditional execution using when statements and complex logic
- Implement loops for efficient repetitive tasks and data processing
- Design comprehensive error handling strategies with blocks
- Control task execution flow with delegation and run strategies
- Use advanced task control features like throttling and batching
- Debug and troubleshoot complex task control scenarios

Why Task Control Matters

Linear vs Intelligent Automation

Basic Approach: All tasks run in sequence on all hosts

```
# Limited flexibility
- name: Install package
  ansible.builtin.dnf:
    name: httpd
    state: present
    # Runs on ALL hosts regardless of need
```

Advanced Approach: Intelligent execution based on conditions

```
# Conditional and intelligent
- name: Install web server package
  ansible.builtin.dnf:
    name: "{{ web_packages[ansible_facts['distribution'] | lower] }}"
    state: present
  when: "'webservers' in group_names"
  # Only runs on web servers with appropriate packages
```

Task Control Benefits

- **Efficiency**: Run tasks only when necessary
- **Reliability**: Handle errors gracefully without stopping automation
- **Scalability**: Process large datasets with loops
- **Flexibility**: Adapt behavior based on runtime conditions

Conditional Execution (when)

Basic When Statements

Simple Conditions:

```
---
- name: Conditional task examples
  hosts: all
  tasks:
    - name: Install Apache on Red Hat systems
      ansible.builtin.dnf:
        name: httpd
        state: present
      when: ansible_facts['distribution'] == "RedHat"

    - name: Install Apache on Ubuntu systems
      ansible.builtin.apt:
        name: apache2
        state: present
      when: ansible_facts['distribution'] == "Ubuntu"

    - name: Configure firewall on RHEL 8+
      ansible.posix.firewalld:
        service: http
        permanent: yes
        state: enabled
      when:
        - ansible_facts['distribution'] == "RedHat"
        - ansible_facts['distribution_major_version'] | int >= 8
```

Complex Conditional Logic

Boolean Logic:

```
- name: Complex conditional examples
  hosts: all
  tasks:
    # AND logic (all conditions must be true)
    - name: Install development tools
      ansible.builtin.dnf:
        name: "@Development Tools"
        state: present
      when:
        - ansible_facts['distribution'] == "RedHat"
        - environment == "development"
        - install_dev_tools | default(false)

    # OR logic (any condition can be true)
    - name: Install web server
      ansible.builtin.dnf:
        name: httpd
        state: present
      when: >
        ansible_facts['distribution'] == "RedHat" or
        ansible_facts['distribution'] == "CentOS"

    # NOT logic (condition must be false)
    - name: Install security updates
      ansible.builtin.dnf:
        name: '*'
        state: latest
        security: yes
      when: not ansible_facts['virtualization_type'] == "docker"
```

Variable Testing:

```
- name: Variable condition examples
  hosts: all
  tasks:
    - name: Use variable when defined
      ansible.builtin.debug:
        msg: "Custom message: {{ custom_message }}"
      when: custom_message is defined

    - name: Skip if variable undefined
      ansible.builtin.debug:
        msg: "Variable is not set"
      when: optional_var is undefined

    - name: Check for empty values
      ansible.builtin.fail:
        msg: "Required variable cannot be empty"
      when: required_var | length == 0

    - name: Test variable types
      ansible.builtin.debug:
        msg: "Port is numeric"
      when: web_port is number

    - name: Test variable content
      ansible.builtin.service:
        name: httpd
        state: started
      when: "'web' in server_roles"
```

Conditional Patterns

Based on Command Results:

```
- name: Conditional based on command output
  hosts: all
  tasks:
    - name: Check if service exists
      ansible.builtin.command: systemctl list-unit-files httpd.service
      register: service_check
      failed_when: false
      changed_when: false

    - name: Start service if it exists
      ansible.builtin.systemd:
        name: httpd
        state: started
      when: service_check.rc == 0

    - name: Install service if it doesn't exist
      ansible.builtin.dnf:
        name: httpd
        state: present
      when: service_check.rc != 0
```

Based on File/Directory Existence:

```
- name: File-based conditionals
  hosts: all
  tasks:
    - name: Check if config file exists
      ansible.builtin.stat:
        path: /etc/httpd/conf/httpd.conf
      register: config_file

    - name: Backup existing config
      ansible.builtin.copy:
        src: /etc/httpd/conf/httpd.conf
        dest: /etc/httpd/conf/httpd.conf.backup
        remote_src: yes
      when: config_file.stat.exists

    - name: Create config from template
      ansible.builtin.template:
        src: httpd.conf.j2
        dest: /etc/httpd/conf/httpd.conf
      when: not config_file.stat.exists
```

Based on Previous Task Results:

```
- name: Task result conditionals
  hosts: all
  tasks:
    - name: Test web service
      ansible.builtin.uri:
        url: "http://{{ inventory_hostname }}"
        status_code: 200
      register: web_test
      ignore_errors: yes

    - name: Restart web service if test failed
      ansible.builtin.systemd:
        name: httpd
        state: restarted
      when: web_test is failed

    - name: Report success
      ansible.builtin.debug:
        msg: "Web service is healthy"
      when: web_test is succeeded
```

Loops and Iteration

Basic Loop Types

Simple List Loop:

```
- name: Basic loop examples
  hosts: all
  tasks:
    - name: Install multiple packages
      ansible.builtin.dnf:
        name: "{{ item }}"
        state: present
      loop:
        - httpd
        - mysql-server
        - php

    - name: Create multiple users
      ansible.builtin.user:
        name: "{{ item }}"
        shell: /bin/bash
        groups: users
      loop:
        - alice
        - bob
        - charlie
```

Dictionary Loop:

```
- name: Dictionary loop examples
  hosts: all
  vars:
    web_users:
      - name: webadmin
        groups: [wheel, apache]
        shell: /bin/bash
      - name: webdev
        groups: [apache, developers]
        shell: /bin/bash
      - name: webtest
        groups: [apache]
        shell: /bin/nologin
  tasks:
    - name: Create web users with different attributes
      ansible.builtin.user:
        name: "{{ item.name }}"
        groups: "{{ item.groups }}"
        shell: "{{ item.shell }}"
        create_home: yes
      loop: "{{ web_users }}"
```

Hash/Dictionary Processing:

```
- name: Process dictionary data
  hosts: all
  vars:
    apache_modules:
      ssl:
        state: enabled
        config: ssl.conf
      rewrite:
        state: enabled
        config: rewrite.conf
      php:
        state: disabled
        config: php.conf
  tasks:
    - name: Configure Apache modules
      ansible.builtin.lineinfile:
        path: "/etc/httpd/conf.modules.d/{{ item.value.config }}"
        line: "LoadModule {{ item.key }}_module modules/mod_{{ item.key }}.so"
        state: "{{ 'present' if item.value.state == 'enabled' else 'absent' }}"
      loop: "{{ apache_modules | dict2items }}"
```

Advanced Loop Patterns

Nested Loops:

```
- name: Nested loop examples
  hosts: all
  vars:
    users: [alice, bob]
    groups: [developers, testers, operators]
  tasks:
    - name: Add users to multiple groups
      ansible.builtin.user:
        name: "{{ item[0] }}"
        groups: "{{ item[1] }}"
        append: yes
      loop: "{{ users | product(groups) | list }}"
      # Creates all combinations: alice+developers, alice+testers, etc.

    - name: Configure services on multiple ports
      ansible.posix.firewalld:
        port: "{{ item[0] }}/{{ item[1] }}"
        permanent: yes
        state: enabled
      loop: "{{ ports | product(protocols) | list }}"
      vars:
        ports: [80, 443, 8080]
        protocols: [tcp, udp]
```

Range Loops:

```
- name: Range loop examples
  hosts: all
  tasks:
    - name: Create numbered directories
      ansible.builtin.file:
        path: "/tmp/dir{{ item }}"
        state: directory
      loop: "{{ range(1, 6) | list }}"  # Creates dir1, dir2, dir3, dir4, dir5

    - name: Create backup copies
      ansible.builtin.copy:
        src: /etc/important.conf
        dest: "/backup/important.conf.{{ item }}"
        remote_src: yes
      loop: "{{ range(1, 4) | list }}"
```

File Globbing Loops:

```
- name: File-based loops
  hosts: all
  tasks:
    - name: Find log files
      ansible.builtin.find:
        paths: /var/log
        patterns: "*.log"
      register: log_files

    - name: Archive log files
      ansible.builtin.archive:
        path: "{{ item.path }}"
        dest: "{{ item.path }}.gz"
        format: gz
        remove: yes
      loop: "{{ log_files.files }}"
      when: log_files.files | length > 0

    - name: Process config files with glob
      ansible.builtin.template:
        src: "{{ item | basename }}.j2"
        dest: "{{ item }}"
        backup: yes
      loop: "{{ query('fileglob', '/etc/httpd/conf.d/*.conf') }}"
```

Loop Control Options

Loop Control Variables:

```
- name: Loop control examples
  hosts: all
  tasks:
    - name: Install packages with detailed output
      ansible.builtin.dnf:
        name: "{{ item }}"
        state: present
      loop:
        - httpd
        - mysql-server
        - php
      loop_control:
        index_var: package_index
        label: "Installing {{ item }}"
        pause: 5  # Pause 5 seconds between iterations

    - name: Show loop progress
      ansible.builtin.debug:
        msg: "Installing package {{ package_index + 1 }}/{{ ansible_loop.length }}: {{ item }}"
      loop:
        - httpd
        - mysql-server
        - php
      loop_control:
        index_var: package_index
```

Conditional Loops:

```
- name: Conditional loops
  hosts: all
  tasks:
    - name: Install packages conditionally
      ansible.builtin.dnf:
        name: "{{ item.package }}"
        state: present
      loop:
        - {package: httpd, condition: "{{ 'webserver' in server_roles }}"}
        - {package: mysql-server, condition: "{{ 'database' in server_roles }}"}
        - {package: nginx, condition: "{{ web_server == 'nginx' }}"}
      when: item.condition | bool
```

Error Handling and Recovery

Block Structure

Basic Block/Rescue/Always:

```
---
- name: Error handling with blocks
  hosts: all
  tasks:
    - name: Web service deployment
      block:
        - name: Install web server
          ansible.builtin.dnf:
            name: httpd
            state: present

        - name: Start web server
          ansible.builtin.systemd:
            name: httpd
            state: started
            enabled: yes

        - name: Test web server
          ansible.builtin.uri:
            url: "http://{{ inventory_hostname }}"
            status_code: 200

        - name: Deploy application
          ansible.builtin.copy:
            src: "{{ app_package }}"
            dest: /var/www/html/

      rescue:
        - name: Log deployment failure
          ansible.builtin.debug:
            msg: "Deployment failed on {{ inventory_hostname }}"

        - name: Stop failed services
          ansible.builtin.systemd:
            name: httpd
            state: stopped
          ignore_errors: yes

        - name: Clean up failed installation
          ansible.builtin.dnf:
            name: httpd
            state: absent

        - name: Notify monitoring system
          ansible.builtin.uri:
            url: "http://monitoring.example.com/alert"
            method: POST
            body_format: json
            body:
              host: "{{ inventory_hostname }}"
              status: "deployment_failed"
              timestamp: "{{ ansible_date_time.iso8601 }}"

      always:
```

```
        - name: Clean temporary files
          ansible.builtin.file:
            path: "{{ item }}"
            state: absent
          loop:
            - /tmp/deployment.lock
            - /tmp/app_temp.*
          ignore_errors: yes

        - name: Update deployment log
          ansible.builtin.linefile:
            path: /var/log/deployment.log
            line: "{{ ansible_date_time.iso8601 }} - Deployment attempt on {{
inventory_hostname }}"
            create: yes
```

Custom Error Conditions

Failed When Conditions:

```
- name: Custom failure conditions
  hosts: all
  tasks:
    - name: Check disk space
      ansible.builtin.shell: df / | tail -1 | awk '{print $5}' | sed 's/%//'
      register: disk_usage
      failed_when: disk_usage.stdout | int > 90

    - name: Verify service configuration
      ansible.builtin.command: httpd -t
      register: config_test
      failed_when:
        - config_test.rc != 0
        - "'Syntax OK' not in config_test.stderr"

    - name: Complex failure conditions
      ansible.builtin.uri:
        url: "http://{{ inventory_hostname }}/api/health"
        method: GET
      register: health_check
      failed_when:
        - health_check.status != 200
        - health_check.json.status != "healthy"
        - health_check.json.database_connection != true
```

Changed When Conditions:

```
- name: Custom change conditions
  hosts: all
  tasks:
    - name: Run deployment script
      ansible.builtin.command: /opt/deploy/deploy.sh
      register: deploy_result
      changed_when:
        - "'deployed' in deploy_result.stdout"
        - deploy_result.rc == 0

    - name: Update application
      ansible.builtin.shell: |
        cd /opt/myapp
        git pull origin main
      register: git_pull
      changed_when: "'Already up to date' not in git_pull.stdout"
```

Ignore Errors Strategy

```
- name: Ignore errors examples
  hosts: all
  tasks:
    - name: Attempt optional configuration
      ansible.builtin.copy:
        src: optional.conf
        dest: /etc/myapp/optional.conf
      ignore_errors: yes

    - name: Try multiple package sources
      block:
        - name: Install from primary repo
          ansible.builtin.dnf:
            name: special-package
            state: present

      rescue:
        - name: Install from alternative repo
          ansible.builtin.dnf:
            name: special-package
            state: present
            enablerepo: alternative-repo
          ignore_errors: yes

        - name: Install from third-party source
          ansible.builtin.get_url:
            url: "{{ third_party_package_url }}"
            dest: /tmp/package.rpm
          ignore_errors: yes

        - name: Install downloaded package
          ansible.builtin.dnf:
            name: /tmp/package.rpm
            state: present
          ignore_errors: yes
```

Task Delegation and Control

Delegation Patterns

Delegate to Specific Host:

```
- name: Delegation examples
  hosts: webservers
  tasks:
    - name: Update load balancer config
      ansible.builtin.template:
        src: backend.conf.j2
        dest: /etc/haproxy/backends.conf
      delegate_to: loadbalancer.example.com
      notify: reload haproxy

    - name: Register in monitoring system
      ansible.builtin.uri:
        url: "http://monitoring.example.com/api/register"
        method: POST
        body_format: json
        body:
          hostname: "{{ inventory_hostname }}"
          ip: "{{ ansible_default_ipv4.address }}"
          service: web
      delegate_to: localhost

    - name: Update DNS records
      ansible.builtin.uri:
        url: "{{ dns_api_url }}"
        method: PUT
        body_format: json
        body:
          name: "{{ inventory_hostname }}"
          value: "{{ ansible_default_ipv4.address }}"
      delegate_to: localhost
      run_once: yes
```

Run Once Pattern:

```
- name: Run once examples
  hosts: webservers
  tasks:
    - name: Download application package (once)
      ansible.builtin.get_url:
        url: "{{ app_download_url }}"
        dest: /tmp/app.tar.gz
      run_once: yes
      delegate_to: "{{ groups['webservers'][0] }}"

    - name: Create shared database (once)
      ansible.builtin.mysql_db:
        name: "{{ app_database }}"
        state: present
      run_once: yes
      delegate_to: "{{ groups['databases'][0] }}"

    - name: Send deployment notification (once)
      ansible.builtin.mail:
        to: ops-team@example.com
        subject: "Deployment completed"
        body: "Application deployed to {{ groups['webservers'] | join(', ') }}"
      run_once: yes
      delegate_to: localhost
```

Execution Control

Serial Execution:

```
---
- name: Rolling update deployment
  hosts: webservers
  serial: 1  # Update one server at a time
  tasks:
    - name: Remove from load balancer
      ansible.builtin.uri:
        url: "http://{{ load_balancer }}/api/disable/{{ inventory_hostname }}"
        method: POST
      delegate_to: localhost

    - name: Update application
      ansible.builtin.copy:
        src: "{{ app_package }}"
        dest: /opt/myapp/

    - name: Restart application service
      ansible.builtin.systemd:
        name: myapp
        state: restarted

    - name: Wait for service health check
      ansible.builtin.uri:
        url: "http://{{ inventory_hostname }}/health"
        status_code: 200
      retries: 10
      delay: 30

    - name: Add back to load balancer
      ansible.builtin.uri:
        url: "http://{{ load_balancer }}/api/enable/{{ inventory_hostname }}"
        method: POST
      delegate_to: localhost
```

Batch Processing:

```
---
- name: Batch update servers
  hosts: all
  serial:
    - 1        # Update 1 server first
    - 25%      # Then 25% of remaining
    - 100%     # Then all remaining
  max_fail_percentage: 20  # Stop if more than 20% fail
  tasks:
    - name: Update packages
      ansible.builtin.dnf:
        name: '*'
        state: latest

    - name: Reboot if needed
      ansible.builtin.reboot:
        reboot_timeout: 300
      when: ansible_reboot_pending | default(false)
```

Throttling:

```
- name: Resource-intensive tasks
  hosts: all
  tasks:
    - name: Download large files (throttled)
      ansible.builtin.get_url:
        url: "{{ large_file_url }}"
        dest: "/tmp/{{ large_file_name }}"
      throttle: 2  # Only 2 hosts at a time

    - name: Backup databases (one at a time)
      ansible.builtin.command: mysqldump --all-databases
      register: backup_result
      throttle: 1
```

Advanced Task Control Patterns

Conditional Blocks

```
- name: Environment-specific configuration
  hosts: all
  tasks:
    - name: Production configuration
      block:
        - name: Install production packages
          ansible.builtin.dnf:
            name: "{{ production_packages }}"
            state: present

        - name: Configure production settings
          ansible.builtin.template:
            src: prod.conf.j2
            dest: /etc/myapp/config.conf

        - name: Enable production monitoring
          ansible.builtin.systemd:
            name: monitoring-agent
            state: started
            enabled: yes
      when: environment == "production"

    - name: Development configuration
      block:
        - name: Install development packages
          ansible.builtin.dnf:
            name: "{{ development_packages }}"
            state: present

        - name: Configure development settings
          ansible.builtin.template:
            src: dev.conf.j2
            dest: /etc/myapp/config.conf

        - name: Disable monitoring in dev
          ansible.builtin.systemd:
            name: monitoring-agent
            state: stopped
            enabled: no
      when: environment == "development"
```

Dynamic Task Generation

```
- name: Dynamic task creation
  hosts: all
  vars:
    services_config:
      web:
        package: httpd
        service: httpd
        port: 80
        config_template: httpd.conf.j2
      database:
        package: mysql-server
        service: mysqld
        port: 3306
        config_template: my.cnf.j2
      cache:
        package: redis
        service: redis
        port: 6379
        config_template: redis.conf.j2
  tasks:
    - name: Configure services dynamically
      include_tasks: configure_service.yml
      loop: "{{ services_config | dict2items }}"
      when: item.key in server_roles

# configure_service.yml
- name: Install {{ item.key }} service
  ansible.builtin.dnf:
    name: "{{ item.value.package }}"
    state: present

- name: Configure {{ item.key }} service
  ansible.builtin.template:
    src: "{{ item.value.config_template }}"
    dest: "/etc/{{ item.key }}/config.conf"
  notify: restart {{ item.key }}

- name: Start {{ item.key }} service
  ansible.builtin.systemd:
    name: "{{ item.value.service }}"
    state: started
    enabled: yes
```

Task Result Processing

```
- name: Process multiple command results
  hosts: all
  tasks:
    - name: Check multiple services
      ansible.builtin.systemd:
        name: "{{ item }}"
      register: service_status
      loop:
        - httpd
        - mysqld
        - redis
        - nginx
      ignore_errors: yes

    - name: Report service states
      ansible.builtin.debug:
        msg: |
          Service {{ item.item }} is {{ item.status.ActiveState }}
          ({{ 'enabled' if item.status.UnitFileState == 'enabled' else 'disabled' }})
      loop: "{{ service_status.results }}"
      when: not item.failed | default(false)

    - name: Restart failed services
      ansible.builtin.systemd:
        name: "{{ item.item }}"
        state: restarted
      loop: "{{ service_status.results }}"
      when:
        - not item.failed | default(false)
        - item.status.ActiveState != "active"
```

Practical Lab Exercises

Exercise 1: Complex Conditional Logic

Create a playbook that:

1. Installs different packages based on OS and role
2. Configures services only if certain conditions are met
3. Skips tasks based on system resources

```
---
- name: Conditional deployment
  hosts: all
  tasks:
    - name: Install web server based on OS and resources
      ansible.builtin.package:
        name: "{{ web_server }}"
        state: present
      vars:
        web_server: >-
          {% if ansible_facts['distribution'] == 'RedHat' %}
            {% if ansible_facts['memtotal_mb'] > 4096 %}nginx{% else %}httpd{% endif %}
          {% elif ansible_facts['distribution'] == 'Ubuntu' %}
            {% if ansible_facts['memtotal_mb'] > 4096 %}nginx{% else %}apache2{% endif %}
          {% else %}httpd{% endif %}
      when:
        - "'webservers' in group_names"
        - ansible_facts['memtotal_mb'] > 1024
```

Exercise 2: Advanced Loop Processing

Build a playbook that:

1. Creates users with different attributes from a complex data structure
2. Configures multiple services with nested configuration
3. Processes file lists with conditional actions

Exercise 3: Comprehensive Error Handling

Design error handling that:

1. Attempts multiple strategies for package installation
2. Recovers gracefully from service failures
3. Always cleans up temporary resources
4. Logs all actions for troubleshooting

Exercise 4: Rolling Deployment Pattern

Implement a rolling update that:

1. Updates servers in batches
2. Removes each server from load balancer during update
3. Verifies health before proceeding
4. Rolls back on failure

Key Takeaways

Conditional Logic Mastery

- **When statements**: Use for simple and complex conditions
- **Boolean logic**: Combine conditions with and/or/not operators
- **Variable testing**: Check for defined, undefined, and type conditions
- **Fact-based decisions**: Make choices based on system information

Loop Proficiency

- **Basic loops**: Process lists efficiently with loop directive
- **Complex data**: Handle dictionaries and nested structures
- **Loop control**: Use labels, indexing, and pauses effectively
- **Performance**: Choose appropriate loop patterns for data size

Error Handling Excellence

- **Block structure**: Organize error handling with block/rescue/always
- **Custom conditions**: Define when tasks fail or change
- **Ignore strategies**: Continue execution when appropriate

- **Recovery patterns**: Implement graceful fallback mechanisms

Delegation and Control

- **Task delegation**: Run tasks on different hosts when needed
- **Run once**: Execute expensive operations only once
- **Serial execution**: Control deployment speed and risk
- **Throttling**: Manage resource usage during intensive operations

---

Next Steps

With advanced task control skills, you're ready for:

1. **Module 05: Templates** - Dynamic configuration with Jinja2
2. **Configuration management** using template logic and variables
3. **Complex file generation** with loops and conditionals in templates
4. **Dynamic service configuration** based on runtime conditions

Your task control expertise enables sophisticated automation workflows!

Next Module: Module 05: Templates →

2.7 Module 05: Templates

Learning Objectives

By the end of this module, you will:

- Master Jinja2 templating syntax for dynamic configuration files
- Use variables, conditionals, and loops within templates
- Apply filters for data transformation and formatting
- Handle whitespace control and template organization
- Debug template issues and troubleshoot rendering problems
- Create reusable template patterns for common scenarios
- Integrate templates with Ansible's template module effectively

Why Templates Transform Configuration Management

Static vs Dynamic Configuration

Static Configuration Files: Hard to maintain across environments

```
# httpd.conf - Hard-coded values
ServerName web01.example.com
Listen 80
MaxRequestWorkers 150
DocumentRoot /var/www/html
```

Dynamic Template-Based Configuration: Adaptable and maintainable

```
# httpd.conf.j2 - Template with variables
ServerName {{ ansible_fqdn }}
Listen {{ http_port | default(80) }}
MaxRequestWorkers {{ max_workers | default(150) }}
DocumentRoot {{ document_root | default('/var/www/html') }}
```

Template Benefits

- **Environment Flexibility**: Same template works across dev/staging/production
- **Maintainability**: Change logic in one place, affects all deployments
- **Dynamic Content**: Generate configuration based on runtime conditions
- **Consistency**: Standardized configuration patterns across infrastructure

Jinja2 Template Fundamentals

Template File Structure

File Naming Convention: `.j2` extension (e.g., `httpd.conf.j2`)

Basic Template Structure:

```
{# templates/httpd.conf.j2 #}
# Generated by Ansible on {{ ansible_date_time.iso8601 }}
# Managed by template: httpd.conf.j2

ServerName {{ ansible_fqdn }}
ServerAdmin {{ server_admin | default('webmaster@' + ansible_domain) }}

Listen {{ http_port }}
{% if ssl_enabled | default(false) %}
Listen {{ https_port | default(443) }} ssl
{% endif %}

# Virtual Hosts
{% for vhost in virtual_hosts | default([]) %}
<VirtualHost *:{{ http_port }}>
    ServerName {{ vhost.name }}
    DocumentRoot {{ vhost.docroot }}
    {% if vhost.aliases is defined %}
    ServerAlias {{ vhost.aliases | join(' ') }}
    {% endif %}
</VirtualHost>
{% endfor %}
```

Jinja2 Syntax Elements

Variable Substitution:

```
{{ variable_name }}                    # Simple variable
{{ ansible_facts['hostname'] }}       # Dictionary access
{{ users[0]['name'] }}                # List and dictionary access
{{ config.database.host }}            # Dot notation for dictionaries
```

Comments:

```
{# This is a single-line comment #}

{#
  This is a
  multi-line comment
#}

{# TODO: Add SSL configuration #}
```

Control Structures:

```
# Conditionals
{% if condition %}
content
{% endif %}

# Loops
{% for item in list %}
content with {{ item }}
{% endfor %}

# Assignments
{% set variable = value %}
```

Variable Usage in Templates

Basic Variable Substitution

```
{# Basic variable usage #}
ServerName {{ server_name }}
Port {{ server_port }}
User {{ web_user }}
Group {{ web_group }}

{# With default values #}
MaxClients {{ max_clients | default(256) }}
Timeout {{ timeout | default(300) }}
KeepAlive {{ keepalive | default('On') }}

{# Conditional defaults #}
LogLevel {{ log_level | default('warn' if environment == 'production' else 'info') }}
```

Complex Data Structures

Dictionary Access:

```
{# Dictionary variable: database = {host: 'db.example.com', port: 3306, name: 'webapp'} #}

# Database Configuration
Host={{ database.host }}
Port={{ database.port }}
Database={{ database.name }}
Username={{ database.user | default('app') }}
Password={{ database.password }}

# Alternative bracket notation
Host={{ database['host'] }}
Port={{ database['port'] }}
Database={{ database['name'] }}
```

List Processing:

```
{# List variable: allowed_hosts = ['web01.example.com', 'web02.example.com', 'api.example.com']
#}

# Simple list output
AllowedHosts={{ allowed_hosts | join(',') }}

# List with formatting
{% for host in allowed_hosts %}
Allow from {{ host }}
{% endfor %}

# List with conditionals
{% for host in allowed_hosts %}
{% if 'api' not in host %}
<VirtualHost {{ host }}:80>
    DocumentRoot /var/www/html
</VirtualHost>
{% endif %}
{% endfor %}
```

Ansible Facts in Templates

System Information:

```
# System Facts Template
# Generated for {{ ansible_facts['hostname'] }}
# OS: {{ ansible_facts['distribution'] }} {{ ansible_facts['distribution_version'] }}
# Architecture: {{ ansible_facts['architecture'] }}
# Total Memory: {{ ansible_facts['memtotal_mb'] }}MB
# CPU Count: {{ ansible_facts['processor_count'] }}

# Network configuration based on facts
{% if ansible_facts['default_ipv4'] is defined %}
BindAddress={{ ansible_facts['default_ipv4']['address'] }}
{% endif %}

# Storage-based configuration
{% for mount in ansible_facts['mounts'] %}
{% if mount['mount'] == '/' %}
# Root filesystem: {{ mount['fstype'] }}, Size: {{ (mount['size_total'] / 1024 / 1024 / 1024) |
round(1) }}GB
{% endif %}
{% endfor %}
```

Hardware-Based Configuration:

```
# Performance tuning based on hardware
{% set memory_mb = ansible_facts['memtotal_mb'] %}
{% set cpu_count = ansible_facts['processor_count'] %}

# Memory-based worker configuration
{% if memory_mb < 2048 %}
workers=2
max_connections=50
{% elif memory_mb < 8192 %}
workers={{ cpu_count * 2 }}
max_connections=200
{% else %}
workers={{ cpu_count * 4 }}
max_connections=500
{% endif %}

# CPU-based thread configuration
thread_pool_size={{ cpu_count * 8 }}
```

Control Structures in Templates

Conditional Logic

If/Elif/Else Statements:

```
{# Environment-based configuration #}
{% if environment == 'development' %}
debug=true
log_level=debug
cache_enabled=false
{% elif environment == 'staging' %}
debug=false
log_level=info
cache_enabled=true
cache_ttl=300
{% elif environment == 'production' %}
debug=false
log_level=warn
cache_enabled=true
cache_ttl=3600
{% else %}
debug=false
log_level=error
cache_enabled=false
{% endif %}

{# Feature flags #}
{% if ssl_enabled | default(false) %}
# SSL Configuration
SSLEngine on
SSLCertificateFile {{ ssl_cert_path }}
SSLCertificateKeyFile {{ ssl_key_path }}
{% if ssl_intermediate_path is defined %}
SSLCertificateChainFile {{ ssl_intermediate_path }}
{% endif %}
{% endif %}
```

Inline Conditionals:

```
# Inline conditional expressions
ServerTokens={{ 'Prod' if environment == 'production' else 'Full' }}
MaxRequestWorkers={{ 500 if high_traffic | default(false) else 150 }}

# Conditional sections
{% if backup_enabled | default(true) %}backup_path={{ backup_directory }}{% endif %}
```

Loop Constructs

Basic For Loops:

```
{# Simple list iteration #}
# Virtual Hosts
{% for vhost in virtual_hosts %}
<VirtualHost *:80>
    ServerName {{ vhost }}
    DocumentRoot /var/www/{{ vhost }}/html
</VirtualHost>
{% endfor %}

{# Dictionary iteration #}
# Environment Variables
{% for key, value in env_vars.items() %}
export {{ key }}="{{ value }}"
{% endfor %}

# Alternative dictionary syntax
{% for item in env_vars | dictsort %}
export {{ item[0] }}="{{ item[1] }}"
{% endfor %}
```

Advanced Loop Features:

```
{# Loop variables and controls #}
# User accounts ({{ users | length }} total)
{% for user in users %}
{% set loop_info = loop %}  {# Capture loop context #}
# User {{ loop.index }}/{{ loop.length }}: {{ user.name }}
username={{ user.name }}
uid={{ user.uid | default(1000 + loop.index0) }}
groups={{ user.groups | default(['users']) | join(',') }}
{% if user.shell is defined %}
shell={{ user.shell }}
{% endif %}

{% if not loop.last %}

{% endif %}  {# Add blank line between users except last #}
{% endfor %}

{# Conditional loop content #}
# Active services
{% for service in services %}
{% if service.enabled | default(true) %}
[{{ service.name }}]
port={{ service.port }}
{% if service.ssl | default(false) %}
ssl_enabled=yes
ssl_port={{ service.ssl_port | default(service.port + 443) }}
{% endif %}
{% endif %}
{% endfor %}
```

Nested Loops:

```
{# Complex nested structure #}
# Load Balancer Configuration
{% for cluster in clusters %}
# Cluster: {{ cluster.name }}
{% for backend in cluster.backends %}
server {{ backend.name }} {{ backend.ip }}:{{ backend.port }} {% if backend.backup |
default(false) %}backup{% endif %}
{% endfor %}
{% if not loop.last %}

{% endif %}
{% endfor %}

{# Loop with conditions #}
# Firewall Rules
{% for zone in firewall_zones %}
{% for rule in zone.rules %}
{% if rule.enabled | default(true) %}
-A {{ zone.name }} -p {{ rule.protocol }} --dport {{ rule.port }} -j ACCEPT
{% endif %}
{% endfor %}
{% endfor %}
```

Filters and Data Transformation

Built-in Filters

String Manipulation:

```
# String filters
hostname={{ inventory_hostname | upper }}
username={{ user_name | lower }}
service_name={{ app_name | title }}
config_path={{ base_path | trim }}

# String formatting
server_id={{ inventory_hostname | replace('.', '_') }}
log_file={{ app_name | lower | replace(' ', '_') }}.log
backup_name={{ ansible_date_time.date | replace('-', '') }}_backup.sql

# String tests and defaults
database_url={{ db_url | default('localhost:5432') }}
api_key={{ api_key | default('REPLACE_ME', true) }}  # true = treat empty string as undefined
```

Numeric Filters:

```
# Math operations
max_memory={{ (ansible_facts['memtotal_mb'] * 0.8) | round | int }}MB
worker_count={{ (ansible_facts['processor_count'] * 2) | round }}
cache_size={{ memory_limit | int // 4 }}

# Formatting
disk_size={{ (disk_bytes / 1024 / 1024 / 1024) | round(2) }}GB
percentage={{ (used_space / total_space * 100) | round(1) }}%
```

List and Dictionary Filters:

```
# List operations
all_hosts={{ groups['all'] | sort | join(',') }}
web_servers={{ groups['webservers'] | length }} servers
first_db={{ groups['databases'] | first }}
backup_hosts={{ groups['all'] | difference(groups['excluded'] | default([])) }}

# List filtering
active_services={{ services | selectattr('enabled') | map(attribute='name') | list }}
https_ports={{ services | selectattr('ssl', 'equalto', true) | map(attribute='port') | list }}

# Dictionary operations
sorted_config={{ config_dict | dictsort }}
config_keys={{ config_dict | list }}  # Get keys only
config_values={{ config_dict.values() | list }}
```

Date and Time Filters:

```
# Date formatting
generated_on={{ ansible_date_time.iso8601 }}
backup_timestamp={{ ansible_date_time.epoch }}
human_date={{ ansible_date_time.date }} at {{ ansible_date_time.time }}

# Custom date formatting (requires strftime)
log_rotation_date={{ ansible_date_time.iso8601 | regex_replace('(\d{4})-(\d{2})-(\d{2}).*', '\
\1\\2\\3') }}
```

Custom Filter Applications

Configuration Logic Filters:

```
# Complex conditional logic
{% set ssl_config = ssl_enabled | default(false) %}
{% set port_config = (443 if ssl_config else 80) %}

# Environment-based values
debug_mode={{ (environment != 'production') | lower }}
log_level={{ {'development': 'debug', 'staging': 'info', 'production': 'warn'}[environment] }}

# Resource calculations
{% set memory_ratio = 0.7 if environment == 'production' else 0.5 %}
max_heap_size={{ (ansible_facts['memtotal_mb'] * memory_ratio) | int }}m
```

Network and IP Filters:

```
# IP address manipulation
network_interface={{ ansible_facts['default_ipv4']['interface'] }}
server_ip={{ ansible_facts['default_ipv4']['address'] }}
subnet_mask={{ ansible_facts['default_ipv4']['netmask'] }}

# Network calculations (custom logic)
{% set ip_parts = ansible_facts['default_ipv4']['address'].split('.') %}
network_id={{ ip_parts[0] }}.{{ ip_parts[1] }}.{{ ip_parts[2] }}.0
```

Template Module Usage

Basic Template Task

```
- name: Deploy Apache configuration
  ansible.builtin.template:
    src: httpd.conf.j2
    dest: /etc/httpd/conf/httpd.conf
    owner: root
    group: root
    mode: '0644'
    backup: yes
  notify: restart apache
```

Advanced Template Options

```
- name: Deploy complex configuration with validation
  ansible.builtin.template:
    src: "{{ item.template }}"
    dest: "{{ item.dest }}"
    owner: "{{ item.owner | default('root') }}"
    group: "{{ item.group | default('root') }}"
    mode: "{{ item.mode | default('0644') }}"
    backup: "{{ create_backups | default(true) }}"
    validate: "{{ item.validate | default(omit) }}"
  loop:
    - template: httpd.conf.j2
      dest: /etc/httpd/conf/httpd.conf
      validate: 'httpd -t -f %s'
    - template: ssl.conf.j2
      dest: /etc/httpd/conf.d/ssl.conf
      validate: 'httpd -t -f /etc/httpd/conf/httpd.conf'
    - template: vhosts.conf.j2
      dest: /etc/httpd/conf.d/vhosts.conf
  notify: restart apache
```

Template with Variables

```
- name: Deploy environment-specific configuration
  ansible.builtin.template:
    src: app.conf.j2
    dest: "/etc/myapp/{{ environment }}.conf"
    owner: myapp
    group: myapp
    mode: '0600'
  vars:
    app_config:
      database:
        host: "{{ db_host }}"
        port: "{{ db_port | default(5432) }}"
        name: "{{ app_name }}_{{ environment }}"
      cache:
        enabled: "{{ environment == 'production' }}"
        ttl: "{{ cache_ttl | default(3600) }}"
      logging:
        level: "{{ log_levels[environment] }}"
        file: "/var/log/myapp/{{ environment }}.log"
  notify: restart myapp
```

Advanced Template Patterns

Template Inheritance and Includes

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

```
# Base configuration template
# Application: {{ app_name }}
# Environment: {{ environment }}
# Generated: {{ ansible_date_time.iso8601 }}

[global]
app_name={{ app_name }}
environment={{ environment }}
debug={{ debug_mode | default(false) | lower }}

{% block database_config %}
[database]
host={{ database.host }}
port={{ database.port }}
name={{ database.name }}
{% endblock %}

{% block cache_config %}
# Cache configuration will be included here
{% endblock %}

{% block custom_config %}
# Custom configuration can be added here
{% endblock %}
```

Extended Template (`production.conf.j2`):

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

{% block cache_config %}
[cache]
enabled=true
type=redis
host={{ redis_host }}
port={{ redis_port | default(6379) }}
ttl={{ cache_ttl | default(3600) }}
{% endblock %}

{% block custom_config %}
[monitoring]
enabled=true
endpoint={{ monitoring_endpoint }}
interval={{ monitoring_interval | default(60) }}

[security]
ssl_required=true
encryption_key={{ vault_encryption_key }}
{% endblock %}
```

Macro Definition and Usage

```
{# Macro definitions #}
{% macro render_vhost(name, docroot, port=80, ssl=false) %}
<VirtualHost *:{{ port }}>
    ServerName {{ name }}
    DocumentRoot {{ docroot }}
    {% if ssl %}
    SSLEngine on
    SSLCertificateFile /etc/ssl/certs/{{ name }}.crt
    SSLCertificateKeyFile /etc/ssl/private/{{ name }}.key
    {% endif %}

    ErrorLog logs/{{ name }}_error.log
    CustomLog logs/{{ name }}_access.log common
</VirtualHost>
{% endmacro %}

{# Macro usage #}
{% for vhost in virtual_hosts %}
{{ render_vhost(vhost.name, vhost.docroot, vhost.port | default(80), vhost.ssl |
default(false)) }}
{% endfor %}
```

Dynamic Content Generation

```
{# Dynamic firewall rules based on services #}
{% set firewall_rules = [] %}
{% for service in services %}
  {% if service.external_access | default(false) %}
    {% set _ = firewall_rules.append('-A INPUT -p tcp --dport ' + service.port|string + ' -j
ACCEPT') %}
  {% else %}
    {% set _ = firewall_rules.append('-A INPUT -s 192.168.0.0/16 -p tcp --dport ' +
service.port|string + ' -j ACCEPT') %}
  {% endif %}
{% endfor %}

# Generated firewall rules
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]

# Allow loopback
-A INPUT -i lo -j ACCEPT

# Allow established connections
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

# Service-specific rules
{% for rule in firewall_rules %}
{{ rule }}
{% endfor %}

COMMIT
```

Template Debugging and Troubleshooting

Common Template Issues

Variable Undefined Errors:

```
{# Problem: Variable might be undefined #}
ServerName {{ server_name }}  # Fails if server_name is undefined

{# Solution: Use default values #}
ServerName {{ server_name | default(ansible_fqdn) }}

{# Solution: Check if defined #}
{% if server_name is defined %}
ServerName {{ server_name }}
{% endif %}
```

Type Errors:

```
{# Problem: Wrong data type #}
MaxClients {{ max_clients }}  # Fails if max_clients is a string

{# Solution: Type conversion #}
MaxClients {{ max_clients | int }}

{# Solution: Type checking #}
{% if max_clients is number %}
MaxClients {{ max_clients }}
{% else %}
MaxClients 150
{% endif %}
```

Loop Issues:

```
{# Problem: Empty list causes no output #}
{% for user in users %}
User: {{ user }}
{% endfor %}

{# Solution: Check for empty lists #}
{% if users | length > 0 %}
# User configuration
{% for user in users %}
User: {{ user }}
{% endfor %}
{% else %}
# No users configured
{% endif %}
```

Debugging Techniques

Debug Output in Templates:

```
{# Debug information #}
<!-- Template Debug Information
  Template: {{ template_path | default('unknown') }}
  Host: {{ inventory_hostname }}
  Variables:
  {% for key, value in vars.items() %}
  {{ key }}: {{ value }}
  {% endfor %}
-->

{# Conditional debug sections #}
{% if debug_templates | default(false) %}
# DEBUG: Variable dump
{% for key in vars.keys() | sort %}
# {{ key }} = {{ vars[key] }}
{% endfor %}
{% endif %}
```

Template Testing Playbook:

```
---
- name: Test template rendering
  hosts: localhost
  vars:
    debug_templates: true
    test_vars:
      server_name: test.example.com
      port: 8080
      ssl_enabled: false
  tasks:
    - name: Generate test template
      ansible.builtin.template:
        src: httpd.conf.j2
        dest: /tmp/httpd_test.conf
      vars: "{{ test_vars }}"

    - name: Display rendered template
      ansible.builtin.debug:
        msg: "{{ lookup('file', '/tmp/httpd_test.conf') }}"
```

Whitespace Control

```
{# Whitespace control examples #}

{# Remove whitespace before #}
{% for item in list -%}
{{ item }}
{% endfor %}

{# Remove whitespace after #}
{% for item in list %}
{{ item }}
{%- endfor %}

{# Remove whitespace both sides #}
{%- for item in list -%}
{{ item }}
{%- endfor -%}

{# Practical example: clean list output #}
hosts={{ groups['webservers'] | join(',') }}

{# vs. #}
hosts=
{%- for host in groups['webservers'] -%}
{{ host }}
{%- if not loop.last -%},{%- endif -%}
{%- endfor %}
```

Practical Lab Exercises

Exercise 1: Multi-Environment Web Server Configuration

Create templates for Apache that:

1. Adapt to different environments (dev/staging/production)
2. Configure SSL based on variables
3. Generate virtual hosts dynamically
4. Include environment-specific tuning parameters

```
{# httpd.conf.j2 #}
# Apache Configuration for {{ environment | upper }}
# Generated: {{ ansible_date_time.iso8601 }}

ServerRoot /etc/httpd
PidFile run/httpd.pid

# Environment-specific settings
{% if environment == 'production' %}
ServerTokens Prod
ServerSignature Off
MaxRequestWorkers 500
{% elif environment == 'staging' %}
ServerTokens Min
ServerSignature Off
MaxRequestWorkers 250
{% else %}
ServerTokens Full
ServerSignature On
MaxRequestWorkers 150
{% endif %}

# SSL Configuration
{% if ssl_enabled | default(false) %}
LoadModule ssl_module modules/mod_ssl.so
Include conf.d/ssl.conf
{% endif %}

# Virtual Hosts
{% for vhost in virtual_hosts %}
{% include 'vhost.conf.j2' %}
{% endfor %}
```

Exercise 2: Database Configuration Template

Build a template that:

1. Configures connection pools based on available memory
2. Sets logging levels by environment
3. Includes backup configuration conditionally
4. Generates connection strings for multiple databases

Exercise 3: Load Balancer Configuration

Create a template that:

1. Discovers backend servers from inventory
2. Configures health checks based on service type
3. Sets up SSL termination conditionally
4. Generates monitoring configuration

Exercise 4: Complex Service Configuration

Design templates for:

1. Application server with multiple instances
2. Service discovery integration
3. Feature flag configuration
4. Performance tuning based on hardware facts

Key Takeaways

Template Design Excellence

- **Variable usage**: Always provide defaults and check for undefined variables
- **Logic organization**: Keep complex logic in templates minimal and readable
- **Reusability**: Design templates that work across environments
- **Documentation**: Include comments explaining template logic

Jinja2 Mastery

- **Syntax proficiency**: Master variables, conditionals, loops, and filters

- **Data handling**: Understand how to process lists, dictionaries, and complex structures
- **Whitespace control**: Manage template output formatting effectively
- **Debugging skills**: Know how to troubleshoot template rendering issues

Configuration Management

- **Environment adaptation**: Templates should adapt to different deployment environments
- **Fact integration**: Leverage Ansible facts for intelligent configuration
- **Validation**: Use template validation to catch configuration errors
- **Security**: Handle sensitive data appropriately in templates

Best Practices

- **Testing**: Always test templates in non-production environments first
- **Backup strategy**: Use backup options to prevent configuration loss
- **Version control**: Track template changes alongside playbook changes
- **Performance**: Consider template rendering performance for large deployments

---

Next Steps

With template mastery achieved, you're ready for:

1. **Module 06: Roles** - Organize templates and tasks into reusable roles
2. **Role-based templates** that can be shared across projects
3. **Template libraries** for common configuration patterns
4. **Advanced role patterns** with template inheritance and customization

Your template skills enable sophisticated configuration management!

Next Module: Module 06: Roles →

2.8 Module 06: Roles & Collections

Learning Objectives

By the end of this module, you will:

- Design and create Ansible roles for code reusability and organization
- Understand role structure and best practices for maintainable automation
- Master role dependencies and variable handling within roles
- Install and use Ansible Galaxy for role and collection management
- Work with Ansible Collections and use FQCN (Fully Qualified Collection Names)
- Create custom collections and understand collection distribution
- Integrate roles and collections into complex automation workflows

Why Roles Transform Automation Architecture

Playbook Limitations vs Role Benefits

Large Playbook Problems:

- Difficult to maintain and debug
- Hard to reuse across projects
- Variables and tasks become unorganized
- Testing becomes complex

Role-Based Solutions:

- **Modular Design**: Each role handles a specific function
- **Reusability**: Same role works across multiple projects
- **Organization**: Clear structure with defined variable locations
- **Testing**: Individual roles can be tested independently
- **Collaboration**: Teams can work on different roles simultaneously

Role Architecture Benefits

```
Without Roles: Monolithic Playbook
├── site.yml (500+ lines)
└── Everything mixed together

With Roles: Modular Architecture
├── site.yml (20 lines)
├── roles/
│   ├── common/          # Base system setup
│   ├── web/            # Web server configuration
│   ├── database/       # Database setup
│   └── monitoring/     # Monitoring tools
└── Clean separation of concerns
```

Role Structure and Organization

Standard Role Directory Structure

```
roles/
└── rolename/
    ├── defaults/
    │   └── main.yml          # Default variables (lowest precedence)
    ├── files/
    │   └── static_file.txt   # Static files for copy module
    ├── handlers/
    │   └── main.yml          # Handler definitions
    ├── meta/
    │   └── main.yml          # Role metadata and dependencies
    ├── tasks/
    │   └── main.yml          # Main task list (entry point)
    ├── templates/
    │   └── config.j2         # Jinja2 templates
    ├── tests/
    │   ├── inventory         # Test inventory
    │   └── test.yml          # Test playbook
    └── vars/
        └── main.yml          # Role variables (high precedence)
```

Role Creation with ansible-galaxy

```
# Create new role structure
ansible-galaxy init my_web_role

# Create role in specific directory
ansible-galaxy init roles/apache_server

# Create role with custom template
ansible-galaxy init --role-skeleton=custom_template my_role

# Verify role structure
tree roles/my_web_role/
```

Role Metadata (meta/main.yml)

```
---
galaxy_info:
  author: Your Name
  description: Apache web server configuration role
  company: Your Organization

  license: MIT

  min_ansible_version: 2.9

  platforms:
    - name: EL
      versions:
        - 8
        - 9
    - name: Ubuntu
      versions:
        - 18.04
        - 20.04
        - 22.04

  galaxy_tags:
    - web
    - apache
    - httpd
    - server

dependencies:
  - role: common
    vars:
      firewall_enabled: true
  - role: security
    when: environment == "production"
```

Role Development Best Practices

Task Organization (tasks/main.yml)

```
---
# tasks/main.yml - Main task entry point

- name: Include OS-specific variables
  ansible.builtin.include_vars: "{{ ansible_facts['distribution'] | lower }}.yml"

- name: Include pre-tasks
  ansible.builtin.include_tasks: pre_tasks.yml

- name: Install packages
  ansible.builtin.include_tasks: install.yml

- name: Configure service
  ansible.builtin.include_tasks: configure.yml

- name: Start services
  ansible.builtin.include_tasks: services.yml

- name: Include post-tasks
  ansible.builtin.include_tasks: post_tasks.yml
  when: run_post_tasks | default(true)
```

Modular Task Files

tasks/install.yml:

```
---
- name: Install web server package
  ansible.builtin.package:
    name: "{{ web_package_name }}"
    state: present
  become: yes

- name: Install additional packages
  ansible.builtin.package:
    name: "{{ item }}"
    state: present
  loop: "{{ additional_packages | default([]) }}"
  become: yes

- name: Create web user
  ansible.builtin.user:
    name: "{{ web_user }}"
    group: "{{ web_group }}"
    shell: /sbin/nologin
    home: "{{ web_home_dir }}"
    create_home: no
  become: yes
  when: create_web_user | default(true)
```

tasks/configure.yml:

```
---
- name: Create configuration directories
  ansible.builtin.file:
    path: "{{ item }}"
    state: directory
    owner: root
    group: root
    mode: '0755'
  loop:
    - "{{ web_config_dir }}"
    - "{{ web_config_dir }}/conf.d"
  become: yes

- name: Deploy main configuration
  ansible.builtin.template:
    src: httpd.conf.j2
    dest: "{{ web_config_file }}"
    owner: root
    group: root
    mode: '0644'
    backup: yes
    validate: "{{ web_binary }} -t -f %s"
  become: yes
  notify: restart web service

- name: Deploy virtual host configurations
  ansible.builtin.template:
    src: vhost.conf.j2
    dest: "{{ web_config_dir }}/conf.d/{{ item.name }}.conf"
    owner: root
    group: root
    mode: '0644'
  loop: "{{ virtual_hosts | default([]) }}"
  become: yes
  notify: restart web service
```

Variable Organization

defaults/main.yml (Default values):

```
---
# Default variables for web role
web_package_name: httpd
web_service_name: httpd
web_user: apache
web_group: apache
web_port: 80
web_ssl_port: 443
web_config_dir: /etc/httpd
web_config_file: /etc/httpd/conf/httpd.conf
web_document_root: /var/www/html
web_binary: httpd

# Feature toggles
ssl_enabled: false
firewall_enabled: true
selinux_enabled: true

# Performance settings
max_clients: 256
keep_alive: true
keep_alive_timeout: 15

# Additional packages (can be overridden)
additional_packages: []

# Virtual hosts (empty by default)
virtual_hosts: []
```

vars/main.yml (Role-specific variables):

```
---
# Internal role variables (high precedence)
web_config_template: httpd.conf.j2
web_pid_file: /var/run/httpd/httpd.pid
web_log_dir: /var/log/httpd

# OS-specific package names (can be overridden by os-specific files)
web_packages:
  RedHat: httpd
  Ubuntu: apache2
  Debian: apache2
```

vars/redhat.yml (OS-specific variables):

```
---
web_package_name: httpd
web_service_name: httpd
web_user: apache
web_group: apache
web_config_dir: /etc/httpd
web_document_root: /var/www/html
web_binary: httpd
firewall_service_name: firewalld
```

vars/ubuntu.yml:

```
---
web_package_name: apache2
web_service_name: apache2
web_user: www-data
web_group: www-data
web_config_dir: /etc/apache2
web_document_root: /var/www/html
web_binary: apache2ctl
firewall_service_name: ufw
```

Handler Organization (handlers/main.yml)

```
---
- name: restart web service
  ansible.builtin.systemd:
    name: "{{ web_service_name }}"
    state: restarted
  become: yes

- name: reload web service
  ansible.builtin.systemd:
    name: "{{ web_service_name }}"
    state: reloaded
  become: yes

- name: restart firewall
  ansible.builtin.systemd:
    name: "{{ firewall_service_name }}"
    state: restarted
  become: yes
  when: firewall_enabled | default(true)

- name: validate web config
  ansible.builtin.command: "{{ web_binary }} -t"
  become: yes
  changed_when: false

- name: update firewall rules
  ansible.posix.firewalld:
    service: "{{ item }}"
    permanent: yes
    state: enabled
    immediate: yes
  loop:
    - http
    - https
  become: yes
  when: firewall_enabled | default(true)
```

Using Roles in Playbooks

Basic Role Usage

```
---
- name: Configure web servers
  hosts: webservers
  become: yes
  roles:
    - common
    - web
    - monitoring
```

Role with Variables

```
---
- name: Configure web servers with custom variables
  hosts: webservers
  become: yes
  roles:
    - role: web
      vars:
        web_port: 8080
        ssl_enabled: true
        virtual_hosts:
          - name: example.com
            docroot: /var/www/example
          - name: api.example.com
            docroot: /var/www/api
```

Conditional Role Usage

```
---
- name: Conditional role application
  hosts: all
  become: yes
  roles:
    - role: common
      when: true  # Always apply common role

    - role: web
      when: "'webservers' in group_names"

    - role: database
      when: "'databases' in group_names"

    - role: monitoring
      when: monitoring_enabled | default(false)
```

Dynamic Role Selection

```
---
- name: Dynamic role application based on host variables
  hosts: all
  become: yes
  tasks:
    - name: Apply web server role
      ansible.builtin.include_role:
        name: "{{ web_server_type }}"
      vars:
        ssl_enabled: "{{ ssl_required | default(false) }}"
        performance_tuning: "{{ environment == 'production' }}"
      when: "'webservers' in group_names"

    - name: Apply database role
      ansible.builtin.include_role:
        name: "{{ database_engine }}"
      vars:
        backup_enabled: true
        replication_enabled: "{{ environment == 'production' }}"
      when: "'databases' in group_names"
```

Ansible Galaxy

Installing Roles from Galaxy

```
# Install specific role
ansible-galaxy install geerlingguy.apache

# Install specific version
ansible-galaxy install geerlingguy.apache,2.0.0

# Install to custom directory
ansible-galaxy install geerlingguy.apache --roles-path ./custom_roles

# Install multiple roles from requirements file
ansible-galaxy install -r requirements.yml

# Force reinstall
ansible-galaxy install geerlingguy.apache --force
```

Requirements File (requirements.yml)

```
---
roles:
  # Install from Ansible Galaxy
  - name: geerlingguy.apache
    version: 2.0.0

  - name: geerlingguy.mysql
    version: ">=3.0.0"

  # Install from Git repository
  - src: https://github.com/example/custom-role.git
    name: custom_role
    version: main

  # Install from local path
  - src: /path/to/local/role
    name: local_role

collections:
  # Install collections
  - name: community.general
    version: ">=3.0.0"

  - name: ansible.posix
    version: "1.3.0"

  - name: community.mysql
```

Role Management Commands

```
# List installed roles
ansible-galaxy list

# Show role information
ansible-galaxy info geerlingguy.apache

# Search for roles
ansible-galaxy search apache

# Remove installed role
ansible-galaxy remove geerlingguy.apache

# Install from requirements file
ansible-galaxy install -r requirements.yml

# Update roles to latest versions
ansible-galaxy install -r requirements.yml --force
```

Ansible Collections

Understanding Collections

Collections provide:

- **Modules**: Task plugins for specific functionality
- **Roles**: Pre-built automation patterns
- **Plugins**: Extend Ansible capabilities
- **Playbooks**: Example automation workflows

FQCN (Fully Qualified Collection Names) required for modules:

```
# Correct - FQCN format
- name: Configure firewall
  ansible.posix.firewalld:    # namespace.collection.module
    service: http
    state: enabled

# Incorrect - short name (deprecated)
- name: Configure firewall
  firewalld:                  # Will fail in modern Ansible
    service: http
    state: enabled
```

Essential Collections for RHCE

ansible.builtin (Core Modules):

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

  - name: Manage service
    ansible.builtin.systemd:
      name: httpd
      state: started
      enabled: yes

  - name: Copy file
    ansible.builtin.copy:
      src: index.html
      dest: /var/www/html/
```

ansible.posix (POSIX System Tools):

```
tasks:
  - name: Configure firewall
    ansible.posix.firewalld:
      service: http
      permanent: yes
      state: enabled

  - name: Mount filesystem
    ansible.posix.mount:
      path: /mnt/data
      src: /dev/sdb1
      fstype: xfs
      state: mounted

  - name: Manage SSH keys
    ansible.posix.authorized_key:
      user: ansible
      key: "{{ lookup('file', '~/.ssh/id_rsa.pub') }}"
```

community.general (Extended Modules):

```
tasks:
  - name: Create partition
    community.general.parted:
      device: /dev/sdb
      number: 1
      state: present

  - name: Create LVM volume group
    community.general.lvg:
      vg: vg_data
      pvs: /dev/sdb1

  - name: Create logical volume
    community.general.lvol:
      vg: vg_data
      lv: lv_web
      size: 10G
```

Installing Collections

```
# Install specific collection
ansible-galaxy collection install community.general

# Install specific version
ansible-galaxy collection install community.general:==3.6.0

# Install to custom path
ansible-galaxy collection install community.general --collections-path ./collections

# Install all collections from requirements
ansible-galaxy collection install -r requirements.yml

# List installed collections
ansible-galaxy collection list

# Upgrade collections
ansible-galaxy collection install community.general --upgrade
```

Collection Directory Structure

```
collections/
└── ansible_collections/
    └── namespace/
        └── collection_name/
            ├── plugins/
            │   ├── modules/
            │   ├── filters/
            │   └── lookup/
            ├── roles/
            ├── playbooks/
            ├── meta/
            │   └── runtime.yml
            └── galaxy.yml
```

Advanced Role Patterns

Role Dependencies

meta/main.yml with dependencies:

```
---
dependencies:
  # Simple dependency
  - role: common

  # Dependency with variables
  - role: firewall
    vars:
      firewall_allowed_ports:
        - 80
        - 443
      firewall_allowed_services:
        - ssh
        - http
        - https

  # Conditional dependency
  - role: ssl_certificates
    when: ssl_enabled | default(false)
    vars:
      ssl_cert_domains: "{{ virtual_host_domains }}"

  # Dependency with tags
  - role: monitoring
    vars:
      monitor_services:
        - "{{ web_service_name }}"
    tags:
      - monitoring
      - health_check
```

Role Composition Patterns

Site Playbook (site.yml):

```
---
- name: Configure infrastructure
  hosts: all
  become: yes
  roles:
    - role: common
      tags: always

- name: Configure web servers
  hosts: webservers
  become: yes
  roles:
    - role: web
      vars:
        web_ssl_enabled: "{{ ssl_enabled | default(false) }}"
        web_virtual_hosts: "{{ virtual_hosts | default([]) }}"
      tags: web

- name: Configure databases
  hosts: databases
  become: yes
  roles:
    - role: database
      vars:
        db_backup_enabled: true
        db_replication_enabled: "{{ environment == 'production' }}"
      tags: database

- name: Configure monitoring
  hosts: all
  become: yes
  roles:
    - role: monitoring
      when: monitoring_enabled | default(true)
      tags: monitoring
```

Dynamic Role Loading

```
---
- name: Dynamic role application
  hosts: all
  tasks:
    - name: Load common role
      ansible.builtin.include_role:
        name: common
      tags: always

    - name: Load server-specific roles
      ansible.builtin.include_role:
        name: "{{ role_item }}"
      loop: "{{ server_roles | default([]) }}"
      loop_control:
        loop_var: role_item
      when: role_item in available_roles

    - name: Load environment-specific roles
      ansible.builtin.include_role:
        name: "{{ environment }}_config"
      when: environment in ['development', 'staging', 'production']
```

Role Testing Strategies

Role Testing Structure

tests/test.yml:

```
---
- hosts: localhost
  remote_user: root
  become: yes
  vars:
    # Test-specific variables
    web_port: 8080
    ssl_enabled: false
    virtual_hosts:
      - name: test.example.com
        docroot: /var/www/test
  roles:
    - ../../  # Reference to role directory

  post_tasks:
    - name: Verify web service is running
      ansible.builtin.systemd:
        name: "{{ web_service_name }}"
        state: started
      check_mode: yes
      register: service_status

    - name: Assert service is active
      ansible.builtin.assert:
        that:
          - service_status.status.ActiveState == "active"
        fail_msg: "Web service is not running"

    - name: Test web server response
      ansible.builtin.uri:
        url: "http://localhost:{{ web_port }}"
        status_code: 200
      register: web_response

    - name: Verify response
      ansible.builtin.assert:
        that:
          - web_response.status == 200
        fail_msg: "Web server not responding correctly"
```

Role Validation Playbook

```
---
- name: Validate role deployment
  hosts: all
  become: yes
  tasks:
    - name: Gather service facts
      ansible.builtin.service_facts:

    - name: Verify required services are running
      ansible.builtin.assert:
        that:
          - ansible_facts.services[item + '.service'].state == 'running'
        fail_msg: "Service {{ item }} is not running"
        success_msg: "Service {{ item }} is running correctly"
      loop:
        - "{{ web_service_name }}"
        - "{{ firewall_service_name }}"

    - name: Test port connectivity
      ansible.builtin.wait_for:
        port: "{{ web_port }}"
        host: "{{ ansible_default_ipv4.address }}"
        delay: 1
        timeout: 10
      register: port_test

    - name: Verify configuration files exist
      ansible.builtin.stat:
        path: "{{ item }}"
      register: config_files
      loop:
        - "{{ web_config_file }}"
        - "{{ web_document_root }}/index.html"

    - name: Assert configuration files present
      ansible.builtin.assert:
        that:
          - item.stat.exists
        fail_msg: "Configuration file {{ item.item }} not found"
      loop: "{{ config_files.results }}"
```

Practical Lab Exercises

Exercise 1: Create a Comprehensive Web Server Role

Requirements:

1. Support multiple OS distributions (RHEL, Ubuntu)
2. Configure SSL conditionally
3. Deploy virtual hosts from variables
4. Include proper error handling and validation
5. Write tests for the role

Exercise 2: Database Configuration Role

Create a role that:

1. Installs database server (MySQL/PostgreSQL)
2. Configures performance tuning based on available memory
3. Creates databases and users from variables
4. Includes backup configuration
5. Handles security hardening

Exercise 3: Multi-Tier Application Role

Design roles for:

1. Load balancer configuration
2. Application server deployment
3. Database setup with replication
4. Monitoring and logging integration
5. Role dependencies and proper ordering

Exercise 4: Collection Creation

Build a custom collection that includes:

1. Multiple related roles
2. Custom modules for specific tasks
3. Plugins for data manipulation
4. Documentation and examples
5. Galaxy metadata for distribution

Key Takeaways

Role Design Excellence

- **Single responsibility**: Each role should have one clear purpose
- **Parameterization**: Use variables to make roles flexible and reusable
- **Documentation**: Include clear README files and inline comments
- **Testing**: Write tests to validate role functionality

Organization and Structure

- **Standard layout**: Follow Ansible's standard role directory structure
- **Variable precedence**: Understand how role variables interact with other variable sources
- **Dependencies**: Use role dependencies to ensure proper ordering and prerequisites
- **Modular tasks**: Break complex roles into smaller, manageable task files

Collection Mastery

- **FQCN usage**: Always use fully qualified collection names for modules
- **Collection installation**: Know how to install and manage collections
- **Collection organization**: Understand how collections organize related automation content
- **Requirements management**: Use requirements.yml for reproducible environments

Best Practices

- **Version control**: Track roles and requirements files in version control
- **Environment consistency**: Use the same collection versions across environments
- **Performance**: Consider role execution performance in large deployments
- **Security**: Handle sensitive data appropriately within roles

Next Steps

With role and collection expertise, you're ready for:

1. **Module 07: System Administration Tasks** - Apply roles to real system administration scenarios
2. **Enterprise automation** using role-based architecture
3. **Complex deployments** with multiple interacting roles
4. **Automation at scale** using collection-based organization

Your role mastery enables sophisticated, maintainable automation!

Next Module: Module 07: System Administration Tasks →

2.9 Module 07: System Administration Tasks

Learning Objectives

By the end of this module, you will:

- Automate standard RHCSA system administration tasks using Ansible
- Master package management automation across different systems
- Implement service management with systemd automation
- Configure storage solutions including LVM and filesystem management
- Automate user and group management with proper security practices
- Manage network configuration including firewall and SELinux automation
- Create comprehensive system hardening and compliance automation
- Design monitoring and maintenance automation workflows

Why Automate System Administration

Manual Administration vs Automation

Manual Administration Challenges:

- Time-consuming repetitive tasks
- Human errors and inconsistency
- Difficult to scale across many systems
- Hard to audit and track changes
- Knowledge concentrated in individuals

Automation Benefits:

- **Consistency**: Same configuration across all systems
- **Speed**: Deploy changes to hundreds of systems simultaneously
- **Reliability**: Eliminate human errors in routine tasks
- **Auditability**: Track all changes through version control
- **Scalability**: Manage infrastructure growth efficiently

RHCE System Administration Focus

The RHCE exam tests automation of standard RHCSA tasks:

- Software package and repository management
- Service configuration and control
- Storage device and filesystem management
- Network and security configuration
- User and group administration
- System monitoring and maintenance

Package and Repository Management

Package Installation and Management

Basic Package Operations:

```
---
- name: Package management examples
  hosts: all
  become: yes
  tasks:
    # Install single package
    - name: Install Apache web server
      ansible.builtin.dnf:
        name: httpd
        state: present

    # Install multiple packages
    - name: Install development tools
      ansible.builtin.dnf:
        name:
          - gcc
          - make
          - git
          - vim
        state: present

    # Install package groups
    - name: Install development group
      ansible.builtin.dnf:
        name: "@Development Tools"
        state: present

    # Update specific package
    - name: Update kernel
      ansible.builtin.dnf:
        name: kernel
        state: latest

    # Update all packages
    - name: Update all packages
      ansible.builtin.dnf:
        name: '*'
        state: latest
      register: update_result

    # Remove packages
    - name: Remove unnecessary packages
      ansible.builtin.dnf:
        name:
          - telnet
          - rsh
          - rlogin
        state: absent
```

Repository Management

Repository Configuration:

```
---
- name: Repository management
  hosts: all
  become: yes
  tasks:
    # Add repository
    - name: Add EPEL repository
      ansible.builtin.dnf:
        name: epel-release
        state: present

    # Configure custom repository
    - name: Add custom repository
      ansible.builtin.yum_repository:
        name: custom-repo
        description: Custom Software Repository
        baseurl: https://repo.example.com/rhel/$releasever/$basearch/
        gpgcheck: yes
        gpgkey: https://repo.example.com/RPM-GPG-KEY-custom
        enabled: yes

    # Import GPG key
    - name: Import repository GPG key
      ansible.builtin.rpm_key:
        key: https://repo.example.com/RPM-GPG-KEY-custom
        state: present

    # Install from specific repository
    - name: Install package from EPEL
      ansible.builtin.dnf:
        name: htop
        state: present
        enablerepo: epel

    # Disable repository temporarily
    - name: Install with disabled repo
      ansible.builtin.dnf:
        name: some-package
        state: present
        disablerepo: epel
```

Package Facts and Validation

```
---
- name: Package validation and facts
  hosts: all
  become: yes
  tasks:
    # Gather package facts
    - name: Get package information
      ansible.builtin.package_facts:
        manager: auto

    # Verify packages are installed
    - name: Verify required packages
      ansible.builtin.assert:
        that:
          - "'httpd' in ansible_facts.packages"
          - "'mysql-server' in ansible_facts.packages"
        fail_msg: "Required packages not installed"
        success_msg: "All required packages present"

    # Check package versions
    - name: Display package versions
      ansible.builtin.debug:
        msg: "{{ item }} version: {{ ansible_facts.packages[item][0].version }}"
      loop:
        - httpd
        - mysql-server
      when: item in ansible_facts.packages

    # Conditional package installation
    - name: Install missing packages
      ansible.builtin.dnf:
        name: "{{ item }}"
        state: present
      loop:
        - nginx
        - php-fpm
        - mariadb-server
      when: item not in ansible_facts.packages
```

Service Management with Systemd

Basic Service Operations

```
---
- name: Service management examples
  hosts: all
  become: yes
  tasks:
    # Start and enable services
    - name: Start and enable web services
      ansible.builtin.systemd:
        name: "{{ item }}"
        state: started
        enabled: yes
        daemon_reload: yes
      loop:
        - httpd
        - php-fpm

    # Stop and disable services
    - name: Stop and disable unnecessary services
      ansible.builtin.systemd:
        name: "{{ item }}"
        state: stopped
        enabled: no
      loop:
        - postfix
        - cups
      ignore_errors: yes

    # Restart services
    - name: Restart network service
      ansible.builtin.systemd:
        name: NetworkManager
        state: restarted

    # Reload service configuration
    - name: Reload systemd daemon
      ansible.builtin.systemd:
        daemon_reload: yes

    # Mask/unmask services
    - name: Mask unwanted service
      ansible.builtin.systemd:
        name: bluetooth
        masked: yes
```

Advanced Service Configuration

```
---
- name: Advanced service management
  hosts: all
  become: yes
  tasks:
    # Create custom service unit
    - name: Create custom application service
      ansible.builtin.copy:
        content: |
          [Unit]
          Description=My Custom Application
          After=network.target

          [Service]
          Type=simple
          User=myapp
          ExecStart=/opt/myapp/bin/myapp
          Restart=always
          RestartSec=10

          [Install]
          WantedBy=multi-user.target
        dest: /etc/systemd/system/myapp.service
        mode: '0644'
      notify: reload systemd

    # Override service configuration
    - name: Override httpd service settings
      ansible.builtin.file:
        path: /etc/systemd/system/httpd.service.d
        state: directory

    - name: Create service override
      ansible.builtin.copy:
        content: |
          [Service]
          LimitNOFILE=65536
          PrivateTmp=yes
        dest: /etc/systemd/system/httpd.service.d/override.conf
      notify: reload systemd

    # Service facts and validation
    - name: Gather service facts
      ansible.builtin.service_facts:

    - name: Verify critical services
      ansible.builtin.assert:
        that:
          - ansible_facts.services['sshd.service'].state == 'running'
          - ansible_facts.services['firewalld.service'].state == 'running'
        fail_msg: "Critical services not running"

  handlers:
    - name: reload systemd
```

```
      ansible.builtin.systemd:
        daemon_reload: yes
```

Service Dependencies and Ordering

```
---
- name: Service dependency management
  hosts: all
  become: yes
  tasks:
    # Install and configure database first
    - name: Install database server
      ansible.builtin.dnf:
        name: mariadb-server
        state: present

    - name: Start database service
      ansible.builtin.systemd:
        name: mariadb
        state: started
        enabled: yes

    # Wait for service to be ready
    - name: Wait for database to be ready
      ansible.builtin.wait_for:
        port: 3306
        host: localhost
        delay: 5
        timeout: 60

    # Configure application that depends on database
    - name: Configure application
      ansible.builtin.template:
        src: app-config.j2
        dest: /etc/myapp/config.conf
      notify: restart application

    - name: Start application service
      ansible.builtin.systemd:
        name: myapp
        state: started
        enabled: yes

  handlers:
    - name: restart application
      ansible.builtin.systemd:
        name: myapp
        state: restarted
```

Storage and Filesystem Management

Disk and Partition Management

```
---
- name: Storage management
  hosts: all
  become: yes
  tasks:
    # Gather disk information
    - name: Gather disk facts
      ansible.builtin.setup:
        gather_subset: hardware

    # Create partition
    - name: Create primary partition
      community.general.parted:
        device: /dev/sdb
        number: 1
        state: present
        part_type: primary
        part_start: 1MiB
        part_end: 10GiB

    # Create extended partition
    - name: Create extended partition
      community.general.parted:
        device: /dev/sdb
        number: 2
        state: present
        part_type: extended
        part_start: 10GiB
        part_end: 100%

    # Create logical partitions
    - name: Create logical partitions
      community.general.parted:
        device: /dev/sdb
        number: "{{ item.number }}"
        state: present
        part_type: logical
        part_start: "{{ item.start }}"
        part_end: "{{ item.end }}"
      loop:
        - {number: 5, start: 10.1GiB, end: 20GiB}
        - {number: 6, start: 20.1GiB, end: 30GiB}
```

LVM Configuration

```
---
- name: LVM management
  hosts: all
  become: yes
  tasks:
    # Create physical volumes
    - name: Create physical volumes
      community.general.lvg:
        vg: vg_data
        pvs:
          - /dev/sdb1
          - /dev/sdc1
        state: present

    # Extend volume group
    - name: Add physical volume to VG
      community.general.lvg:
        vg: vg_data
        pvs:
          - /dev/sdb1
          - /dev/sdc1
          - /dev/sdd1
        state: present

    # Create logical volumes
    - name: Create logical volumes
      community.general.lvol:
        vg: vg_data
        lv: "{{ item.name }}"
        size: "{{ item.size }}"
        state: present
      loop:
        - {name: lv_web, size: 10G}
        - {name: lv_database, size: 20G}
        - {name: lv_logs, size: 5G}

    # Extend logical volume
    - name: Extend logical volume
      community.general.lvol:
        vg: vg_data
        lv: lv_database
        size: 30G
        resizefs: yes

    # Create snapshots
    - name: Create database snapshot
      community.general.lvol:
        vg: vg_data
        lv: lv_database_snap
        size: 2G
        snapshot: lv_database
        state: present
```

Filesystem Creation and Management

```
---
- name: Filesystem management
  hosts: all
  become: yes
  tasks:
    # Create filesystems
    - name: Create XFS filesystems
      ansible.builtin.filesystem:
        fstype: xfs
        dev: "{{ item.device }}"
        opts: "{{ item.opts | default(omit) }}"
      loop:
        - {device: /dev/vg_data/lv_web, opts: "-b size=4096"}
        - {device: /dev/vg_data/lv_database}
        - {device: /dev/vg_data/lv_logs}

    # Create ext4 filesystem with options
    - name: Create ext4 filesystem
      ansible.builtin.filesystem:
        fstype: ext4
        dev: /dev/sdb2
        opts: -cc

    # Mount filesystems
    - name: Mount filesystems
      ansible.posix.mount:
        path: "{{ item.mount }}"
        src: "{{ item.device }}"
        fstype: "{{ item.fstype }}"
        opts: "{{ item.opts | default('defaults') }}"
        state: mounted
      loop:
        - {device: /dev/vg_data/lv_web, mount: /var/www, fstype: xfs}
        - {device: /dev/vg_data/lv_database, mount: /var/lib/mysql, fstype: xfs}
        - {device: /dev/vg_data/lv_logs, mount: /var/log/apps, fstype: xfs, opts: "defaults,noat
ime"}

    # Create swap
    - name: Create swap filesystem
      ansible.builtin.filesystem:
        fstype: swap
        dev: /dev/vg_data/lv_swap

    - name: Enable swap
      ansible.posix.mount:
        path: none
        src: /dev/vg_data/lv_swap
        fstype: swap
        opts: sw
        state: present

    # Verify mounts
    - name: Verify filesystems are mounted
      ansible.builtin.mount:
```

```
        path: "{{ item }}"
        state: mounted
      loop:
        - /var/www
        - /var/lib/mysql
        - /var/log/apps
      check_mode: yes
      register: mount_check

    - name: Display mount status
      ansible.builtin.debug:
        msg: "{{ item.path }} is {{ 'mounted' if not item.changed else 'not mounted' }}"
      loop: "{{ mount_check.results }}"
```

User and Group Management

User Account Management

```
---
- name: User management
  hosts: all
  become: yes
  tasks:
    # Create groups first
    - name: Create system groups
      ansible.builtin.group:
        name: "{{ item.name }}"
        gid: "{{ item.gid | default(omit) }}"
        system: "{{ item.system | default(false) }}"
        state: present
      loop:
        - {name: webadmin, gid: 2001}
        - {name: dbadmin, gid: 2002}
        - {name: developers, gid: 2010}
        - {name: appservice, gid: 3001, system: true}

    # Create user accounts
    - name: Create user accounts
      ansible.builtin.user:
        name: "{{ item.name }}"
        uid: "{{ item.uid | default(omit) }}"
        group: "{{ item.primary_group | default(item.name) }}"
        groups: "{{ item.groups | default([]) }}"
        append: "{{ item.append | default(true) }}"
        shell: "{{ item.shell | default('/bin/bash') }}"
        home: "{{ item.home | default('/home/' + item.name) }}"
        create_home: "{{ item.create_home | default(true) }}"
        system: "{{ item.system | default(false) }}"
        password: "{{ item.password | default('!') }}"
        state: present
      loop:
        - name: webuser
          uid: 2001
          primary_group: webadmin
          groups: [wheel]
          password: "{{ webuser_password_hash }}"
        - name: dbuser
          uid: 2002
          primary_group: dbadmin
          groups: [wheel]
          password: "{{ dbuser_password_hash }}"
        - name: appuser
          uid: 3001
          primary_group: appservice
          system: true
          shell: /sbin/nologin
          create_home: false

    # Configure SSH keys
    - name: Configure SSH authorized keys
      ansible.posix.authorized_key:
        user: "{{ item.user }}"
```

```
        key: "{{ item.key }}"
        comment: "{{ item.comment | default('Ansible managed key') }}"
        state: present
      loop:
        - user: webuser
          key: "{{ webuser_ssh_key }}"
          comment: "Web administrator key"
        - user: dbuser
          key: "{{ dbuser_ssh_key }}"
          comment: "Database administrator key"

    # Set password policies
    - name: Configure password aging
      ansible.builtin.user:
        name: "{{ item }}"
        password_expire_max: 90
        password_expire_min: 7
        password_expire_warn: 14
      loop:
        - webuser
        - dbuser
```

Sudo Configuration

```
---
- name: Sudo configuration
  hosts: all
  become: yes
  tasks:
    # Create sudo rules
    - name: Configure sudo rules
      ansible.builtin.copy:
        content: |
          # {{ item.name }} sudo rules
          {{ item.rule }}
        dest: "/etc/sudoers.d/{{ item.name }}"
        mode: '0440'
        validate: /usr/sbin/visudo -cf %s
      loop:
        - name: webadmin
          rule: "%webadmin ALL=(ALL) NOPASSWD: /bin/systemctl restart httpd, /bin/systemctl relo
ad httpd"
        - name: dbadmin
          rule: "%dbadmin ALL=(ALL) NOPASSWD: /bin/systemctl * mysqld, /usr/bin/mysql"
        - name: developers
          rule: "%developers ALL=(appuser) NOPASSWD: ALL"

    # Verify sudo configuration
    - name: Test sudo configuration
      ansible.builtin.command: sudo -u webuser sudo -l
      become_user: webuser
      register: sudo_test
      changed_when: false
      failed_when: sudo_test.rc != 0
```

Network and Security Configuration

Firewall Management with firewalld

```
---
- name: Firewall configuration
  hosts: all
  become: yes
  tasks:
    # Ensure firewalld is running
    - name: Start and enable firewalld
      ansible.builtin.systemd:
        name: firewalld
        state: started
        enabled: yes

    # Configure zones
    - name: Set default zone
      ansible.posix.firewalld:
        zone: public
        state: enabled
        permanent: yes
        immediate: yes

    # Add interfaces to zones
    - name: Add interface to internal zone
      ansible.posix.firewalld:
        zone: internal
        interface: eth1
        permanent: yes
        immediate: yes
        state: enabled
      when: "'eth1' in ansible_interfaces"

    # Configure services
    - name: Allow services in public zone
      ansible.posix.firewalld:
        zone: public
        service: "{{ item }}"
        permanent: yes
        immediate: yes
        state: enabled
      loop:
        - ssh
        - http
        - https

    # Configure custom ports
    - name: Allow custom ports
      ansible.posix.firewalld:
        zone: public
        port: "{{ item }}"
        permanent: yes
        immediate: yes
        state: enabled
      loop:
        - 8080/tcp
        - 3306/tcp
```

```
    # Configure rich rules
    - name: Add rich rules
      ansible.posix.firewalld:
        zone: public
        rich_rule: "{{ item }}"
        permanent: yes
        immediate: yes
        state: enabled
      loop:
        - "rule family='ipv4' source address='192.168.1.0/24' service name='mysql' accept"
        - "rule family='ipv4' source address='10.0.0.0/8' port port=22 protocol=tcp accept"

    # Remove unwanted services
    - name: Remove unwanted services
      ansible.posix.firewalld:
        service: "{{ item }}"
        permanent: yes
        immediate: yes
        state: disabled
      loop:
        - dhcpv6-client
        - cockpit
      ignore_errors: yes
```

SELinux Configuration

```
---
- name: SELinux configuration
  hosts: all
  become: yes
  tasks:
    # Set SELinux mode
    - name: Set SELinux to enforcing
      ansible.posix.selinux:
        policy: targeted
        state: enforcing

    # Configure SELinux booleans
    - name: Configure SELinux booleans
      ansible.posix.seboolean:
        name: "{{ item.name }}"
        state: "{{ item.state }}"
        persistent: yes
      loop:
        - {name: httpd_can_network_connect, state: yes}
        - {name: httpd_can_network_connect_db, state: yes}
        - {name: httpd_execmem, state: no}
        - {name: httpd_enable_homedirs, state: no}

    # Set SELinux file contexts
    - name: Set SELinux file contexts
      community.general.sefcontext:
        target: "{{ item.path }}"
        setype: "{{ item.type }}"
        state: present
      loop:
        - {path: "/var/www/html(/.*)?", type: httpd_exec_t}
        - {path: "/var/log/myapp(/.*)?", type: var_log_t}
      notify: restore selinux contexts

    # Configure SELinux ports
    - name: Configure SELinux port contexts
      community.general.seport:
        ports: "{{ item.port }}"
        proto: "{{ item.proto }}"
        setype: "{{ item.type }}"
        state: present
      loop:
        - {port: 8080, proto: tcp, type: http_port_t}
        - {port: 3306, proto: tcp, type: mysqld_port_t}

  handlers:
    - name: restore selinux contexts
      ansible.builtin.command: restorecon -R "{{ item }}"
      loop:
        - /var/www/html
        - /var/log/myapp
```

System Monitoring and Maintenance

Log Management

```
---
- name: Log management configuration
  hosts: all
  become: yes
  tasks:
    # Configure rsyslog
    - name: Configure rsyslog for application logs
      ansible.builtin.lineinfile:
        path: /etc/rsyslog.conf
        line: "{{ item }}"
        insertafter: "# Log anything"
      loop:
        - "local0.*    /var/log/myapp.log"
        - "local1.*    /var/log/myapp-error.log"
      notify: restart rsyslog

    # Configure logrotate
    - name: Configure logrotate for application logs
      ansible.builtin.copy:
        content: |
          /var/log/myapp*.log {
              daily
              missingok
              rotate 30
              compress
              delaycompress
              notifempty
              sharedscripts
              postrotate
                  /bin/kill -HUP `cat /var/run/rsyslogd.pid 2> /dev/null` 2> /dev/null || true
              endscript
          }
        dest: /etc/logrotate.d/myapp
        mode: '0644'

    # Create log directories
    - name: Create application log directories
      ansible.builtin.file:
        path: "{{ item.path }}"
        state: directory
        owner: "{{ item.owner }}"
        group: "{{ item.group }}"
        mode: "{{ item.mode }}"
      loop:
        - {path: /var/log/myapp, owner: myapp, group: myapp, mode: '0755'}
        - {path: /var/log/backup, owner: root, group: root, mode: '0755'}

  handlers:
    - name: restart rsyslog
      ansible.builtin.systemd:
        name: rsyslog
        state: restarted
```

System Maintenance Tasks

```
---
- name: System maintenance
  hosts: all
  become: yes
  tasks:
    # Configure automatic updates
    - name: Configure dnf-automatic
      ansible.builtin.lineinfile:
        path: /etc/dnf/automatic.conf
        regexp: "{{ item.regexp }}"
        line: "{{ item.line }}"
      loop:
        - {regexp: "^upgrade_type", line: "upgrade_type = security"}
        - {regexp: "^apply_updates", line: "apply_updates = yes"}
        - {regexp: "^emit_via", line: "emit_via = email"}
      notify: enable dnf-automatic

    # Configure cron jobs
    - name: Configure maintenance cron jobs
      ansible.builtin.cron:
        name: "{{ item.name }}"
        user: "{{ item.user | default('root') }}"
        minute: "{{ item.minute }}"
        hour: "{{ item.hour }}"
        day: "{{ item.day | default('*') }}"
        month: "{{ item.month | default('*') }}"
        weekday: "{{ item.weekday | default('*') }}"
        job: "{{ item.job }}"
        state: present
      loop:
        - name: "Database backup"
          minute: "0"
          hour: "2"
          job: "/usr/local/bin/backup-database.sh"
        - name: "Log cleanup"
          minute: "30"
          hour: "3"
          weekday: "0"
          job: "find /var/log -name '*.log' -mtime +30 -delete"
        - name: "System health check"
          minute: "*/15"
          hour: "*"
          job: "/usr/local/bin/health-check.sh"

    # Configure system limits
    - name: Configure system limits
      ansible.builtin.pam_limits:
        domain: "{{ item.domain }}"
        limit_type: "{{ item.type }}"
        limit_item: "{{ item.item }}"
        value: "{{ item.value }}"
      loop:
        - {domain: "*", type: soft, item: nofile, value: "65536"}
        - {domain: "*", type: hard, item: nofile, value: "65536"}
```

```
        - {domain: mysql, type: soft, item: nproc, value: "32768"}
        - {domain: mysql, type: hard, item: nproc, value: "32768"}

  handlers:
    - name: enable dnf-automatic
      ansible.builtin.systemd:
        name: dnf-automatic-install.timer
        state: started
        enabled: yes
```

System Security Hardening

Security Configuration

```
---
- name: System security hardening
  hosts: all
  become: yes
  tasks:
    # Disable unused services
    - name: Disable unnecessary services
      ansible.builtin.systemd:
        name: "{{ item }}"
        state: stopped
        enabled: no
        masked: yes
      loop:
        - avahi-daemon
        - cups
        - bluetooth
        - rpcbind
      ignore_errors: yes

    # Configure SSH hardening
    - name: Harden SSH configuration
      ansible.builtin.lineinfile:
        path: /etc/ssh/sshd_config
        regexp: "{{ item.regexp }}"
        line: "{{ item.line }}"
        backup: yes
      loop:
        - {regexp: "^#?PermitRootLogin", line: "PermitRootLogin no"}
        - {regexp: "^#?PasswordAuthentication", line: "PasswordAuthentication no"}
        - {regexp: "^#?MaxAuthTries", line: "MaxAuthTries 3"}
        - {regexp: "^#?ClientAliveInterval", line: "ClientAliveInterval 300"}
        - {regexp: "^#?ClientAliveCountMax", line: "ClientAliveCountMax 2"}
        - {regexp: "^#?Protocol", line: "Protocol 2"}
      notify: restart sshd

    # Configure kernel parameters
    - name: Configure kernel security parameters
      ansible.posix.sysctl:
        name: "{{ item.name }}"
        value: "{{ item.value }}"
        state: present
        sysctl_file: /etc/sysctl.d/99-security.conf
      loop:
        - {name: net.ipv4.ip_forward, value: "0"}
        - {name: net.ipv4.conf.all.send_redirects, value: "0"}
        - {name: net.ipv4.conf.default.send_redirects, value: "0"}
        - {name: net.ipv4.conf.all.accept_redirects, value: "0"}
        - {name: net.ipv4.conf.default.accept_redirects, value: "0"}
        - {name: net.ipv4.conf.all.secure_redirects, value: "0"}
        - {name: net.ipv4.conf.default.secure_redirects, value: "0"}
        - {name: kernel.dmesg_restrict, value: "1"}
        - {name: kernel.kptr_restrict, value: "2"}

    # File permissions hardening
```

```
    - name: Set secure file permissions
      ansible.builtin.file:
        path: "{{ item.path }}"
        mode: "{{ item.mode }}"
      loop:
        - {path: /etc/passwd, mode: '0644'}
        - {path: /etc/shadow, mode: '0000'}
        - {path: /etc/group, mode: '0644'}
        - {path: /etc/gshadow, mode: '0000'}
        - {path: /boot/grub2/grub.cfg, mode: '0600'}

  handlers:
    - name: restart sshd
      ansible.builtin.systemd:
        name: sshd
        state: restarted
```

Practical Lab Exercises

Exercise 1: Web Server Infrastructure

Create automation for a complete web server setup:

1. Package installation (Apache, PHP, MySQL)
2. Service configuration and startup
3. Firewall configuration for web services
4. SSL certificate installation and configuration
5. Log rotation and monitoring setup

Exercise 2: Database Server Configuration

Automate database server deployment:

1. MySQL/MariaDB installation and hardening
2. User and database creation
3. Backup automation setup
4. Performance tuning based on system resources
5. Security configuration and firewall rules

Exercise 3: Multi-Tier Application Deployment

Deploy a complete application stack:

1. Load balancer configuration
2. Application server setup with clustering
3. Database replication configuration
4. Shared storage configuration
5. Monitoring and alerting setup

Exercise 4: System Compliance Automation

Create compliance automation:

1. Security hardening according to security benchmarks
2. User account policy enforcement
3. System auditing configuration
4. Backup and disaster recovery automation
5. Compliance reporting and validation

Key Takeaways

System Administration Automation Excellence

- **Comprehensive coverage**: Automate all standard RHCSA tasks with Ansible
- **Consistency**: Ensure identical configuration across all managed systems
- **Scalability**: Deploy changes to hundreds of systems simultaneously
- **Reliability**: Eliminate human error in routine administrative tasks

Package and Service Management Mastery

- **Repository management**: Automate software repository configuration and updates
- **Service dependencies**: Understand and manage service startup ordering
- **Health checking**: Implement service monitoring and automated recovery
- **Security updates**: Automate security patch management

Storage and User Management

- **LVM automation**: Create flexible storage solutions with logical volume management

- **Filesystem management**: Automate filesystem creation, mounting, and maintenance
- **User lifecycle**: Implement complete user account lifecycle management
- **Security policies**: Enforce password policies and access controls

Security and Compliance

- **Firewall automation**: Implement comprehensive network security policies
- **SELinux management**: Configure mandatory access controls appropriately
- **System hardening**: Apply security benchmarks consistently
- **Audit and monitoring**: Implement comprehensive system monitoring

---

Next Steps

With system administration automation mastery, you're ready for:

1. **Module 08: Ansible Vault & Advanced Features** - Secure automation and advanced techniques
2. **Enterprise security practices** with encrypted sensitive data
3. **Advanced automation patterns** for complex infrastructures
4. **Performance optimization** for large-scale deployments

Your system administration automation skills enable enterprise-level infrastructure management!

Next Module: Module 08: Ansible Vault & Advanced Features →

2.10 Module 08: Ansible Vault & Advanced Features

Learning Objectives

By the end of this module, you will:

- Master Ansible Vault for encrypting sensitive data and secure automation
- Implement advanced security practices for credential management
- Use dynamic inventories for cloud and large-scale environments
- Optimize Ansible performance for enterprise deployments
- Debug complex automation issues with advanced troubleshooting techniques
- Apply best practices for production Ansible environments
- Understand Ansible's architecture and extending capabilities

Why Security and Advanced Features Matter

Security Challenges in Automation

Common Security Issues:

- Passwords and API keys stored in plain text
- Sensitive configuration data exposed in version control
- Inadequate access controls on automation systems
- Lack of audit trails for automated changes

Ansible Vault Solutions:

- **Encryption at rest**: Sensitive data encrypted in files
- **Granular encryption**: Encrypt individual variables or entire files
- **Multiple vault passwords**: Different encryption keys for different purposes
- **Integration**: Seamless integration with existing automation workflows

Advanced Features for Enterprise

Enterprise Requirements:

- **Scale**: Manage thousands of systems efficiently
- **Performance**: Minimize execution time across large infrastructures
- **Flexibility**: Adapt to dynamic, cloud-native environments
- **Observability**: Comprehensive logging and monitoring
- **Extensibility**: Custom modules and plugins for specific needs

---

Ansible Vault Fundamentals

Basic Vault Operations

Creating Encrypted Files:

```
# Create new encrypted file
ansible-vault create secrets.yml

# This will open your default editor with an empty file
# Content will be encrypted when saved
---
database_password: "super_secret_password"
api_key: "abcd1234567890"
ssl_private_key: |
  -----BEGIN PRIVATE KEY-----
  MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQC...
  -----END PRIVATE KEY-----
```

Viewing Encrypted Files:

```
# View encrypted file content
ansible-vault view secrets.yml

# Edit existing encrypted file
ansible-vault edit secrets.yml

# Show encrypted file structure (without decrypting)
cat secrets.yml
```

File Encryption Operations:

```
# Encrypt existing plain text file
ansible-vault encrypt existing_secrets.yml

# Decrypt file (permanently removes encryption)
ansible-vault decrypt secrets.yml

# Change vault password
ansible-vault rekey secrets.yml

# Change password for multiple files
ansible-vault rekey secrets.yml database.yml api_config.yml
```

String Encryption for Inline Variables

```
# Encrypt a single string
ansible-vault encrypt_string 'secret_password' --name 'db_password'

# Output (to paste in playbooks/variables):
db_password: !vault |
          $ANSIBLE_VAULT;1.1;AES256
          66386439653761623...
          (encrypted content)

# Encrypt with specific vault ID
ansible-vault encrypt_string --vault-id prod@prompt 'secret_password' --name 'db_password'

# Encrypt multiple strings interactively
ansible-vault encrypt_string --stdin-name 'variable_name'
```

Using Encrypted Strings in Playbooks:

```
---
- name: Deploy application with encrypted credentials
  hosts: all
  vars:
    # Plain text variables
    app_name: myapp
    app_port: 8080

    # Encrypted variables (generated with encrypt_string)
    db_password: !vault |
          $ANSIBLE_VAULT;1.1;AES256
          66386439653761623435363238643433643362643132613633353534303939666531323231
          36376266313765653762323637326538383766376361636 30a38396439393665316338653366636537646 2
          ...
    api_secret_key: !vault |
          $ANSIBLE_VAULT;1.1;AES256
          31313063396261653838376637636163630a38396439393665316338653366636537646 2
          66386439653761623435363238643433643362643132613633353534303939666531323231
          ...
  tasks:
    - name: Configure application with encrypted credentials
      ansible.builtin.template:
        src: app_config.j2
        dest: /etc/myapp/config.conf
        mode: '0600'
      vars:
        database_url: "mysql://user:{{ db_password }}@db.example.com/myapp"
        secret_key: "{{ api_secret_key }}"
```

Vault Password Management

Password File Method

```
# Create vault password file
echo 'your_vault_password' > .vault_password
chmod 600 .vault_password

# Use password file with ansible-navigator
ansible-navigator run site.yml --vault-password-file .vault_password --mode stdout

# Configure in ansible.cfg
cat >> ansible.cfg << 'EOF'
[defaults]
vault_password_file = .vault_password
EOF
```

Multiple Vault IDs

Creating Multiple Vault Files:

```
# Create different vault files with different IDs
ansible-vault create --vault-id dev@prompt dev_secrets.yml
ansible-vault create --vault-id prod@prompt prod_secrets.yml
ansible-vault create --vault-id shared@prompt shared_secrets.yml

# Create password files for different environments
echo 'dev_vault_password' > .vault_password_dev
echo 'prod_vault_password' > .vault_password_prod
chmod 600 .vault_password_*
```

Using Multiple Vault IDs:

```
# Specify multiple vault password sources
ansible-navigator run site.yml \
  --vault-id dev@.vault_password_dev \
  --vault-id prod@.vault_password_prod \
  --vault-id shared@prompt \
  --mode stdout

# Environment-specific decryption
ansible-navigator run deploy.yml \
  --vault-id prod@.vault_password_prod \
  --limit production \
  --mode stdout
```

Vault ID in Playbook Structure:

```
---
- name: Multi-environment deployment
  hosts: all
  vars_files:
    - group_vars/all/common.yml
    - "group_vars/{{ environment }}/secrets.yml"  # Contains vault_id specific encryption
  vars:
    # Shared secrets (encrypted with shared@vault_id)
    monitoring_api_key: !vault |
          $ANSIBLE_VAULT;1.2;AES256;shared
          66386439653761623435632386434336433626431326136333535343039396666531323231
          ...

    # Environment-specific secrets
    database_password: !vault |
          $ANSIBLE_VAULT;1.2;AES256;prod
          31313063396261653838376637636163630a383964393936653163386533666365376462
          ...
```

Script-Based Password Retrieval

Password Retrieval Script (`get_vault_password.py`):

```
#!/usr/bin/env python3
import os
import sys
import getpass
from subprocess import check_output, CalledProcessError

def get_password_from_keyring(vault_id):
    """Retrieve password from system keyring"""
    try:
        import keyring
        return keyring.get_password("ansible-vault", vault_id)
    except ImportError:
        return None

def get_password_from_env(vault_id):
    """Retrieve password from environment variable"""
    env_var = f"ANSIBLE_VAULT_{vault_id.upper()}"
    return os.environ.get(env_var)

def get_password_from_file(vault_id):
    """Retrieve password from secure file"""
    password_file = f"/secure/vault_passwords/{vault_id}"
    try:
        with open(password_file, 'r') as f:
            return f.read().strip()
    except FileNotFoundError:
        return None

def main():
    vault_id = sys.argv[1] if len(sys.argv) > 1 else 'default'

    # Try multiple password sources
    password = (
        get_password_from_keyring(vault_id) or
        get_password_from_env(vault_id) or
        get_password_from_file(vault_id) or
        getpass.getpass(f"Vault password for {vault_id}: ")
    )

    if password:
        print(password)
        sys.exit(0)
    else:
        sys.stderr.write(f"Could not retrieve vault password for {vault_id}\n")
        sys.exit(1)

if __name__ == "__main__":
    main()
```

Using Password Script:

```
# Make script executable
chmod +x get_vault_password.py

# Use with ansible-navigator
ansible-navigator run site.yml \
  --vault-id dev@./get_vault_password.py \
  --vault-id prod@./get_vault_password.py \
  --mode stdout
```

Organizing Encrypted Data

Vault File Organization

Directory Structure:

```
inventory/
├── group_vars/
│   ├── all/
│   │   ├── common.yml             # Plain text variables
│   │   └── vault.yml              # Encrypted variables
│   ├── production/
│   │   ├── vars.yml               # Plain text variables
│   │   └── vault.yml              # Production secrets
│   └── development/
│       ├── vars.yml               # Plain text variables
│       └── vault.yml              # Development secrets
├── host_vars/
│   ├── web01.example.com/
│   │   ├── vars.yml               # Plain text host variables
│   │   └── vault.yml              # Host-specific secrets
└── vaults/
    ├── database_passwords.yml     # Database credentials
    ├── api_keys.yml              # API keys and tokens
    └── certificates.yml          # SSL certificates and keys
```

Variable Naming Convention:

```
# group_vars/all/vault.yml
---
# Database passwords
vault_mysql_root_password: !vault |
    $ANSIBLE_VAULT;1.1;AES256...

vault_app_database_password: !vault |
    $ANSIBLE_VAULT;1.1;AES256...

# API credentials
vault_monitoring_api_key: !vault |
    $ANSIBLE_VAULT;1.1;AES256...

vault_backup_service_token: !vault |
    $ANSIBLE_VAULT;1.1;AES256...

# SSL certificates
vault_ssl_private_key: !vault |
    $ANSIBLE_VAULT;1.1;AES256...
```

Reference from Plain Text Files:

```
# group_vars/all/common.yml
---
# Reference encrypted variables
mysql_root_password: "{{ vault_mysql_root_password }}"
app_database_password: "{{ vault_app_database_password }}"
monitoring_api_key: "{{ vault_monitoring_api_key }}"

# Plain text configuration
mysql_port: 3306
mysql_datadir: /var/lib/mysql
backup_schedule: "0 2 * * *"
```

Mixed Encryption Patterns

Template with Encrypted Content:

```
---
- name: Deploy configuration with mixed content
  hosts: all
  become: yes
  vars:
    # Plain text variables
    app_name: myapp
    app_version: "1.2.3"
    log_level: info

    # Encrypted sensitive variables
    database_url: !vault |
          $ANSIBLE_VAULT;1.1;AES256
          66386439653761623435363238643433643362643132613633353534303939666531323231
          ...
    secret_key: !vault |
          $ANSIBLE_VAULT;1.1;AES256
          313130633962616538383766376361636 30a383964393936653163386533666365376462
          ...
  tasks:
    - name: Deploy application configuration
      ansible.builtin.template:
        src: app.conf.j2
        dest: /etc/myapp/app.conf
        owner: myapp
        group: myapp
        mode: '0600'
        backup: yes
```

Template File (`templates/app.conf.j2`):

```
# {{ app_name }} Configuration
# Version: {{ app_version }}
# Generated: {{ ansible_date_time.iso8601 }}

[application]
name={{ app_name }}
version={{ app_version }}
log_level={{ log_level }}

[database]
url={{ database_url }}

[security]
secret_key={{ secret_key }}
{% if ssl_enabled | default(false) %}
ssl_cert={{ ssl_cert_path }}
ssl_key={{ ssl_key_path }}
{% endif %}

[monitoring]
{% if monitoring_enabled | default(true) %}
api_key={{ vault_monitoring_api_key }}
endpoint={{ monitoring_endpoint }}
{% endif %}
```

Dynamic Inventories

Cloud-Based Dynamic Inventories

AWS EC2 Dynamic Inventory:

```
# aws_ec2.yml
---
plugin: amazon.aws.aws_ec2
regions:
  - us-east-1
  - us-west-2
filters:
  tag:Environment:
    - production
    - staging
  instance-state-name: running

keyed_groups:
  # Group by instance type
  - key: instance_type
    prefix: type
  # Group by availability zone
  - key: placement.availability_zone
    prefix: az
  # Group by security groups
  - key: security_groups | map(attribute='group_name')
    prefix: sg

hostnames:
  - tag:Name
  - dns-name
  - private-ip-address

compose:
  # Create custom variables from AWS data
  aws_region: placement.region
  aws_account_id: owner_id
  instance_name: tags.Name | default('unnamed')
  environment: tags.Environment | default('unknown')

cache: true
cache_plugin: jsonfile
cache_timeout: 3600
cache_connection: /tmp/ansible-aws-inventory
```

Using Dynamic Inventory:

```
# Test dynamic inventory
ansible-inventory -i aws_ec2.yml --list

# Use with playbooks
ansible-navigator run site.yml -i aws_ec2.yml --mode stdout

# Target specific groups created by dynamic inventory
ansible-navigator run web_deploy.yml -i aws_ec2.yml --limit type_t3_medium --mode stdout
```

Custom Dynamic Inventory Scripts

Python Inventory Script (`custom_inventory.py`):

```
#!/usr/bin/env python3
import json
import sys
import argparse
import requests
from typing import Dict, Any

def get_inventory_from_cmdb() -> Dict[str, Any]:
    """Fetch inventory from Configuration Management Database"""
    inventory = {
        '_meta': {
            'hostvars': {}
        }
    }

    # Example: Fetch from REST API
    try:
        response = requests.get('https://cmdb.example.com/api/hosts',
                                timeout=30)
        response.raise_for_status()
        hosts_data = response.json()

        for host in hosts_data:
            hostname = host['hostname']

            # Add to appropriate groups
            for role in host.get('roles', []):
                if role not in inventory:
                    inventory[role] = {'hosts': [], 'vars': {}}
                inventory[role]['hosts'].append(hostname)

            # Add environment groups
            env = host.get('environment', 'unknown')
            env_group = f"env_{env}"
            if env_group not in inventory:
                inventory[env_group] = {'hosts': [], 'vars': {}}
            inventory[env_group]['hosts'].append(hostname)

            # Set host variables
            inventory['_meta']['hostvars'][hostname] = {
                'ansible_host': host.get('ip_address'),
                'ansible_user': host.get('ssh_user', 'ansible'),
                'environment': env,
                'data_center': host.get('datacenter'),
                'hardware_type': host.get('hardware_type'),
                'custom_vars': host.get('custom_variables', {})
            }

    except requests.RequestException as e:
        print(f"Error fetching inventory: {e}", file=sys.stderr)
        sys.exit(1)

    return inventory
```

```
def get_host_vars(hostname: str) -> Dict[str, Any]:
    """Get variables for specific host"""
    inventory = get_inventory_from_cmdb()
    return inventory['_meta']['hostvars'].get(hostname, {})

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('--list', action='store_true',
                        help='List all hosts')
    parser.add_argument('--host',
                        help='Get variables for specific host')
    args = parser.parse_args()

    if args.list:
        inventory = get_inventory_from_cmdb()
        print(json.dumps(inventory, indent=2))
    elif args.host:
        host_vars = get_host_vars(args.host)
        print(json.dumps(host_vars, indent=2))
    else:
        parser.print_help()
        sys.exit(1)

if __name__ == "__main__":
    main()
```

Making Script Executable and Testing:

```
# Make script executable
chmod +x custom_inventory.py

# Test inventory script
./custom_inventory.py --list | jq .

# Test host-specific variables
./custom_inventory.py --host web01.example.com | jq .

# Use with Ansible
ansible-navigator run site.yml -i ./custom_inventory.py --mode stdout
```

Performance Optimization

Execution Performance Tuning

Ansible Configuration (`ansible.cfg`):

```
[defaults]
# Increase parallelism
forks = 20
host_key_checking = False

# Connection optimization
timeout = 30
gather_timeout = 30

# SSH optimization
[ssh_connection]
ssh_args = -o ControlMaster=auto -o ControlPersist=300s -o ControlPath=/tmp/.ansible-cp-%h-%p-%r
pipelining = True
retries = 3
```

Performance-Optimized Playbook Patterns:

```
---
- name: High-performance deployment
  hosts: all
  gather_facts: no  # Skip fact gathering when not needed
  strategy: free    # Allow hosts to complete tasks independently
  tasks:
    # Gather only essential facts
    - name: Gather minimal facts
      ansible.builtin.setup:
        gather_subset:
          - network
          - hardware
      when: need_system_info | default(false)

    # Use async for long-running tasks
    - name: Download large package
      ansible.builtin.get_url:
        url: "{{ package_url }}"
        dest: "/tmp/{{ package_name }}"
      async: 300  # 5 minute timeout
      poll: 0     # Fire and forget
      register: download_job

    # Batch operations efficiently
    - name: Install multiple packages in single task
      ansible.builtin.dnf:
        name: "{{ packages }}"
        state: present
      vars:
        packages:
          - httpd
          - php
          - mysql-server
          - git

    # Check async job completion
    - name: Wait for download completion
      ansible.builtin.async_status:
        jid: "{{ download_job.ansible_job_id }}"
      register: download_result
      until: download_result.finished
      retries: 30
      delay: 10
      when: download_job.ansible_job_id is defined
```

Memory and Resource Optimization

Large File Handling:

```
---
- name: Efficient large file operations
  hosts: all
  tasks:
    # Use synchronize for large file transfers
    - name: Sync large directory
      ansible.posix.synchronize:
        src: /local/large_directory/
        dest: /remote/large_directory/
        delete: yes
        compress: yes
        times: yes
      delegate_to: localhost

    # Stream large file downloads
    - name: Download large file with streaming
      ansible.builtin.uri:
        url: "{{ large_file_url }}"
        dest: "/tmp/large_file.tar.gz"
        creates: "/tmp/large_file.tar.gz"
        timeout: 3600
        follow_redirects: all

    # Process files in chunks
    - name: Process log files in chunks
      ansible.builtin.lineinfile:
        path: "{{ item }}"
        regexp: "{{ log_pattern }}"
        line: "{{ replacement_line }}"
      loop: "{{ log_files[:50] }}"  # Process first 50 files
      when: log_files | length > 0
```

Connection Pooling and Reuse:

```
---
- name: Connection optimization example
  hosts: all
  vars:
    ansible_ssh_common_args: >
      -o ControlMaster=auto
      -o ControlPersist=3600s
      -o ControlPath=/tmp/.ansible-cp-%h-%p-%r
      -o StrictHostKeyChecking=no
      -o UserKnownHostsFile=/dev/null
  tasks:
    # Multiple tasks will reuse the same SSH connection
    - name: Check disk space
      ansible.builtin.shell: df -h

    - name: Check memory usage
      ansible.builtin.shell: free -m

    - name: Check process list
      ansible.builtin.shell: ps aux | head -20
```

Advanced Debugging and Troubleshooting

Debug Information Collection

Comprehensive Debug Playbook:

```
---
- name: Ansible debugging and diagnostics
  hosts: all
  gather_facts: yes
  tasks:
    # System information
    - name: Display system facts
      ansible.builtin.debug:
        msg: |
          Host: {{ inventory_hostname }}
          OS: {{ ansible_facts['distribution'] }} {{ ansible_facts['distribution_version'] }}
          Architecture: {{ ansible_facts['architecture'] }}
          Memory: {{ ansible_facts['memtotal_mb'] }}MB
          Python: {{ ansible_facts['python_version'] }}
          IP: {{ ansible_facts['default_ipv4']['address'] | default('N/A') }}

    # Connection information
    - name: Display connection details
      ansible.builtin.debug:
        msg: |
          Connection type: {{ ansible_connection | default('ssh') }}
          User: {{ ansible_user | default('not specified') }}
          Host: {{ ansible_host | default(inventory_hostname) }}
          Port: {{ ansible_port | default(22) }}

    # Variable information
    - name: Display variable information
      ansible.builtin.debug:
        msg: |
          Groups: {{ group_names }}
          Play hosts: {{ play_hosts }}
          Inventory hostname: {{ inventory_hostname }}
          Magic variables available: {{ hostvars[inventory_hostname].keys() | length }} keys

    # Test connectivity and permissions
    - name: Test basic operations
      block:
        - name: Test write permissions
          ansible.builtin.copy:
            content: "Test file created at {{ ansible_date_time.iso8601 }}"
            dest: /tmp/ansible_test
          register: write_test

        - name: Test command execution
          ansible.builtin.command: whoami
          register: whoami_result
          changed_when: false

        - name: Test privilege escalation
          ansible.builtin.command: whoami
          become: yes
          register: sudo_test
          changed_when: false
```

```
        - name: Display test results
          ansible.builtin.debug:
            msg: |
              Write test: {{ 'PASS' if write_test is succeeded else 'FAIL' }}
              Command execution: {{ whoami_result.stdout }}
              Privilege escalation: {{ sudo_test.stdout }}

      rescue:
        - name: Display error information
          ansible.builtin.debug:
            msg: |
              Error occurred during testing
              Write test failed: {{ write_test is failed }}
              Command failed: {{ whoami_result is failed | default(false) }}
              Sudo failed: {{ sudo_test is failed | default(false) }}

    # Cleanup
    - name: Cleanup test files
      ansible.builtin.file:
        path: /tmp/ansible_test
        state: absent
      ignore_errors: yes
```

Error Analysis and Resolution

Common Error Patterns and Solutions:

```
---
- name: Error handling and resolution patterns
  hosts: all
  tasks:
    # Handle SSH connection issues
    - name: Test SSH connectivity
      ansible.builtin.ping:
      register: ping_result
      ignore_errors: yes

    - name: Debug SSH issues
      ansible.builtin.debug:
        msg: |
          SSH connection failed. Check:
          1. Host {{ ansible_host | default(inventory_hostname) }} is reachable
          2. SSH service is running on port {{ ansible_port | default(22) }}
          3. SSH key authentication is configured
          4. User {{ ansible_user | default('root') }} exists and has access
      when: ping_result is failed

    # Handle permission issues
    - name: Test file operations with error handling
      block:
        - name: Attempt file creation
          ansible.builtin.copy:
            content: "test content"
            dest: "{{ test_file_path }}"
          vars:
            test_file_path: /etc/test_ansible_access

      rescue:
        - name: Handle permission errors
          ansible.builtin.debug:
            msg: |
              Permission denied. Possible solutions:
              1. Use 'become: yes' for privilege escalation
              2. Check file/directory permissions
              3. Verify sudo configuration for user {{ ansible_user }}
              4. Check SELinux/AppArmor policies

        - name: Retry with privilege escalation
          ansible.builtin.copy:
            content: "test content"
            dest: /tmp/test_ansible_access
          become: yes

    # Handle package management issues
    - name: Install package with error handling
      ansible.builtin.dnf:
        name: nonexistent-package
        state: present
      register: package_result
      ignore_errors: yes
```

```
    - name: Debug package issues
      ansible.builtin.debug:
        msg: |
          Package installation failed: {{ package_result.msg }}
          Common solutions:
          1. Check package name spelling
          2. Verify repository configuration
          3. Update package cache: dnf makecache
          4. Check network connectivity to repositories
      when: package_result is failed
```

Advanced Ansible Patterns

Custom Modules and Plugins

Simple Custom Module (`library/custom_service_check.py`):

```
#!/usr/bin/python

from ansible.module_utils.basic import AnsibleModule
import subprocess
import json

def check_service_health(service_name, port=None):
    """Check if service is running and optionally test port connectivity"""
    result = {'service_running': False, 'port_open': False}

    # Check if service is running
    try:
        cmd = ['systemctl', 'is-active', service_name]
        proc = subprocess.run(cmd, capture_output=True, text=True)
        result['service_running'] = proc.returncode == 0
        result['service_status'] = proc.stdout.strip()
    except Exception as e:
        result['service_error'] = str(e)

    # Check port connectivity if specified
    if port:
        try:
            import socket
            sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            sock.settimeout(5)
            result['port_open'] = sock.connect_ex(('localhost', port)) == 0
            sock.close()
        except Exception as e:
            result['port_error'] = str(e)

    return result

def main():
    module = AnsibleModule(
        argument_spec=dict(
            service=dict(type='str', required=True),
            port=dict(type='int', required=False),
            expected_status=dict(type='str', default='active',
                                 choices=['active', 'inactive', 'failed'])
        ),
        supports_check_mode=True
    )

    service = module.params['service']
    port = module.params['port']
    expected_status = module.params['expected_status']

    if module.check_mode:
        module.exit_json(changed=False, msg="Check mode - no changes made")

    health_check = check_service_health(service, port)

    # Determine if service meets expected criteria
    service_ok = health_check.get('service_status') == expected_status
```

```
    port_ok = health_check.get('port_open', True) if port else True

    overall_health = service_ok and port_ok

    result = {
        'changed': False,
        'service_name': service,
        'health_check': health_check,
        'healthy': overall_health,
        'msg': f"Service {service} is {'healthy' if overall_health else 'unhealthy'}"
    }

    if overall_health:
        module.exit_json(**result)
    else:
        module.fail_json(**result)

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

Using Custom Module:

```
---
- name: Use custom service health check module
  hosts: all
  tasks:
    - name: Check web service health
      custom_service_check:
        service: httpd
        port: 80
        expected_status: active
      register: web_health

    - name: Display health check results
      ansible.builtin.debug:
        var: web_health

    - name: Take action based on health check
      ansible.builtin.systemd:
        name: httpd
        state: restarted
      when: not web_health.healthy
```

Advanced Templating and Filters

Custom Filter Plugin (`filter_plugins/network_filters.py`):

```
def calculate_network(ip_address, subnet_mask):
    """Calculate network address from IP and subnet mask"""
    import ipaddress
    try:
        network = ipaddress.IPv4Network(f"{ip_address}/{subnet_mask}", strict=False)
        return {
            'network': str(network.network_address),
            'broadcast': str(network.broadcast_address),
            'netmask': str(network.netmask),
            'hostmask': str(network.hostmask),
            'num_addresses': network.num_addresses,
            'cidr': str(network)
        }
    except ValueError as e:
        return {'error': str(e)}

def subnet_hosts(network_cidr):
    """Generate list of host IPs in a subnet"""
    import ipaddress
    try:
        network = ipaddress.IPv4Network(network_cidr)
        return [str(ip) for ip in network.hosts()]
    except ValueError as e:
        return []

class FilterModule(object):
    """Custom network filters"""
    def filters(self):
        return {
            'calculate_network': calculate_network,
            'subnet_hosts': subnet_hosts
        }
```

Using Custom Filters:

```
---
- name: Network configuration with custom filters
  hosts: all
  vars:
    server_ip: "{{ ansible_default_ipv4.address }}"
    subnet_mask: "{{ ansible_default_ipv4.netmask }}"
  tasks:
    - name: Calculate network information
      ansible.builtin.set_fact:
        network_info: "{{ server_ip | calculate_network(subnet_mask) }}"

    - name: Display network information
      ansible.builtin.debug:
        msg: |
          Server: {{ server_ip }}
          Network: {{ network_info.network }}
          Broadcast: {{ network_info.broadcast }}
          CIDR: {{ network_info.cidr }}
          Available addresses: {{ network_info.num_addresses }}

    - name: Generate host list for subnet
      ansible.builtin.set_fact:
        subnet_hosts: "{{ network_info.cidr | subnet_hosts }}"

    - name: Show first 10 available IPs
      ansible.builtin.debug:
        msg: "Available IPs: {{ subnet_hosts[:10] | join(', ') }}"
```

Practical Lab Exercises

Exercise 1: Comprehensive Vault Implementation

Create a secure automation setup:

1. Set up multiple vault password files for different environments
2. Create encrypted variable files for database passwords, API keys, and SSL certificates
3. Implement a password retrieval script using environment variables
4. Create playbooks that use mixed encrypted and plain text variables
5. Test vault operations across different environments

Exercise 2: Dynamic Inventory Integration

Build dynamic inventory system:

1. Create a Python script that fetches inventory from a REST API
2. Implement caching for performance optimization
3. Add custom group creation based on host attributes
4. Create host variables from external data sources
5. Test integration with cloud provider dynamic inventories

Exercise 3: Performance Optimization Project

Optimize automation for large-scale deployment:

1. Configure Ansible for high-performance execution
2. Implement async operations for long-running tasks
3. Use connection pooling and SSH optimizations
4. Create efficient playbooks with minimal fact gathering
5. Benchmark and compare execution times

Exercise 4: Advanced Debugging and Monitoring

Create comprehensive debugging toolkit:

1. Build debugging playbooks for common issues
2. Implement custom modules for health checking
3. Create logging and monitoring integration
4. Develop troubleshooting runbooks
5. Set up automated error reporting

Key Takeaways

Security Mastery with Ansible Vault

- **Data protection**: Encrypt all sensitive data including passwords, keys, and certificates
- **Vault organization**: Use consistent naming and directory structures for encrypted files
- **Multiple environments**: Implement separate vault passwords for different environments
- **Automation integration**: Seamlessly integrate vault operations into CI/CD pipelines

Advanced Features Proficiency

- **Dynamic inventories**: Adapt to cloud and container environments with automatic discovery
- **Performance optimization**: Configure Ansible for enterprise-scale deployments
- **Extensibility**: Create custom modules and plugins for specific organizational needs
- **Debugging expertise**: Develop systematic approaches to troubleshooting complex automation

Production Best Practices

- **Security first**: Implement defense-in-depth security practices for automation systems
- **Scalability**: Design automation that grows with infrastructure needs
- **Monitoring**: Implement comprehensive logging and monitoring of automation activities
- **Documentation**: Maintain detailed documentation of advanced configurations and customizations

Enterprise Integration

- **External systems**: Integrate with Configuration Management Databases and monitoring systems
- **Compliance**: Implement automation that supports regulatory and security compliance
- **Change management**: Integrate automation with organizational change management processes
- **Team collaboration**: Structure automation projects for multiple team collaboration

---

Final Steps and Continuing Education

RHCE Exam Readiness

With completion of all 8 modules, you have comprehensive coverage of:

- **Exam Objective 1**: Understanding core Ansible components
- **Exam Objective 2**: Using roles and Ansible Content Collections
- **Exam Objective 3**: Installing and configuring Ansible control node
- **Exam Objective 4**: Configuring Ansible managed nodes
- **Exam Objective 5**: Running playbooks with Automation content navigator
- **Exam Objective 6**: Creating Ansible plays and playbooks
- **Exam Objective 7**: Automating standard RHCSA tasks
- **Exam Objective 8**: Managing content with templates and Ansible Vault

Continuing Your Ansible Journey

- **Advanced Automation Platform**: Explore Red Hat Ansible Automation Platform (AAP)

- **Ansible Tower/AWX**: Learn enterprise automation with web UI and API
- **CI/CD Integration**: Integrate Ansible with GitLab CI, Jenkins, and other tools
- **Container Automation**: Explore Ansible for Kubernetes and container management
- **Network Automation**: Extend skills to network device automation
- **Community Contribution**: Contribute to Ansible community projects and collections

Professional Development

- **Certification Progression**: Consider DCI (Deployment and Continuous Integration) specialization
- **Advanced Specializations**: Pursue OpenShift, Security, or Networking specializations
- **Community Engagement**: Join Ansible user groups and contribute to open source projects
- **Continuous Learning**: Stay updated with new Ansible features and best practices

Congratulations! You now have comprehensive RHCE-level Ansible automation skills. Apply these skills in real-world scenarios, continue practicing, and good luck with your certification journey!

Study Complete - Ready for RHCE EX294 Certification!

3. Quick References

3.1 RHCE Study Guide Summary

Concise Reference for RHCE EX294 Exam Success

Purpose: Focused summary of essential RHCE exam content covering all objectives without unnecessary detail.

Source Materials:

- Sander van Vugt's RHCE Guide (16 chapters) - Primary exam guide
- Jeff Geerling's Ansible for DevOps (15 chapters) - Real-world patterns
- Red Hat RHCE Study Guide - Exam-focused content

Study Approach: Master these topics through hands-on practice and systematic review.

RHCE Exam Overview

Exam Code: EX294 | **Duration**: 4 hours | **Format**: Performance-based | **Passing Score**: 210/300

Prerequisites

- Current RHCSA certification required
- RHEL 8/9 system administration knowledge
- Basic Linux command line proficiency

Exam Environment

- **Control Node**: RHEL 8/9 with Ansible pre-installed
- **Managed Nodes**: 2-4 RHEL systems for automation targets
- **Tools Available**: ansible-navigator, ansible-doc, man pages (offline only)
- **No Internet Access**: All documentation must be local

Core RHCE Topics

1. Installation and Configuration (20%)

Control Node Setup

- **Package Installation**: `dnf install ansible-core ansible-navigator`
- **Configuration File**: `/etc/ansible/ansible.cfg` or `./ansible.cfg`
- **Key Settings**: `host_key_checking = False`, `remote_user = ansible`, `become = True`
- **Directory Structure**: `playbooks/`, `roles/`, `group_vars/`, `host_vars/`

SSH Key Management

- **Generate Keys**: `ssh-keygen -t rsa -b 4096`
- **Distribute Keys**: `ssh-copy-id ansible@managed_node`
- **Test Connectivity**: `ansible all -m ping`

Essential Configuration

```
[defaults]
inventory = inventory.ini
remote_user = ansible
host_key_checking = False
become = True
become_method = sudo
roles_path = ./roles
collections_paths = ./collections
```

2. Inventory Management (15%)

Static Inventory Formats

- **INI Format**: Groups in `[brackets]`, hosts listed below
- **YAML Format**: Hierarchical structure with `children:` and `hosts:`
- **Variables**: `host_vars/` directory or inline `hostname var=value`

Host Patterns

- **All hosts**: `all` or `*`
- **Groups**: `webservers`, `databases`
- **Exclusions**: `webservers:!web03`
- **Intersections**: `webservers:&production`

- **Regular expressions**: `~web\d+`

Group Variables

- **Structure**: `group_vars/groupname.yml` or `group_vars/groupname/`
- **Precedence**: host_vars > group_vars > all
- **Best Practice**: Use descriptive variable names

3. Ad-hoc Commands (10%)

Command Structure

```
ansible <pattern> -m <module> -a "<arguments>" [options]
```

Essential Modules

Module	Purpose	Example
`ping`	Connectivity test	`ansible all -m ping`
`command`	Run commands	`ansible all -m command -a "uptime"`
`shell`	Shell with pipes	`ansible all -m shell -a "ps aux | grep http"`
`copy`	Copy files	`ansible all -m copy -a "src=file dest=/tmp/"`
`dnf`	Package management	`ansible all -m dnf -a "name=httpd state=present" --become`
`systemd`	Service control	`ansible all -m systemd -a "name=httpd state=started enabled=yes" --become`

Common Options

- `--become` or `-b`: Privilege escalation
- `--check` or `-C`: Dry run mode
- `--limit`: Target specific hosts
- `-v/-vv/-vvv`: Increase verbosity

4. Playbook Development (25%)

Basic Structure

```
---
- name: Playbook description
  hosts: target_group
  become: yes
  vars:
    variable_name: value
  tasks:
    - name: Task description
      module_name:
        parameter: value
      notify: handler_name
  handlers:
    - name: handler_name
      module_name:
        parameter: value
```

Essential Modules with FQCN

Module	FQCN	Primary Use
`dnf`	`ansible.builtin.dnf`	Package management
`systemd`	`ansible.builtin.systemd`	Service control
`copy`	`ansible.builtin.copy`	File copying
`template`	`ansible.builtin.template`	Jinja2 templating
`file`	`ansible.builtin.file`	File/directory operations
`user`	`ansible.builtin.user`	User management
`group`	`ansible.builtin.group`	Group management
`firewalld`	`ansible.posix.firewalld`	Firewall configuration
`mount`	`ansible.posix.mount`	Filesystem mounting
`seboolean`	`ansible.posix.seboolean`	SELinux booleans

Task Control

- **Conditionals**: `when: condition`
- **Loops**: `loop:` or `with_items:`
- **Error Handling**: `failed_when:`, `ignore_errors:`
- **Change Control**: `changed_when:`

5. Variables and Facts (20%)

Variable Types

Type	Scope	Definition Location
Global	All hosts	Command line `-e`
Play	Single play	`vars:` section
Host	Single host	`host_vars/`
Group	Host group	`group_vars/`
Role	Role scope	`roles/*/vars/`
Default	Fallback	`roles/*/defaults/`

Magic Variables

- `inventory_hostname`: Current host name
- `hostvars`: All host variables
- `groups`: All inventory groups
- `group_names`: Groups current host belongs to
- `play_hosts`: Hosts in current play

Facts

- **Gathering**: Automatic via `setup` module
- **Usage**: `{{ ansible_facts['distribution'] }}`
- **Custom Facts**: `/etc/ansible/facts.d/*.fact`
- **Disable**: `gather_facts: no`

6. Templates and Jinja2 (15%)

Template Basics

- **File Extension**: `.j2`
- **Module**: `ansible.builtin.template`
- **Variables**: `{{ variable_name }}`
- **Conditionals**: `{% if condition %} ... {% endif %}`
- **Loops**: `{% for item in list %} ... {% endfor %}`

Common Filters

Filter	Purpose	Example
`default`	Fallback value	`{{ var | default('none') }}`
`upper` / `lower`	Case conversion	`{{ name | upper }}`
`length`	Count items	`{{ list | length }}`
`join`	Combine with separator	`{{ items | join(',') }}`
`regex_replace`	Pattern replacement	`{{ text | regex_replace('old', 'new') }}`

Template Example

```
# Generated by Ansible
ServerName {{ ansible_fqdn }}
Listen {{ http_port | default(80) }}

{% for vhost in virtual_hosts %}
<VirtualHost *:{{ http_port }}>
    ServerName {{ vhost.name }}
    DocumentRoot {{ vhost.docroot }}
</VirtualHost>
{% endfor %}
```

7. Roles and Collections (20%)

Role Structure

```
roles/rolename/
├── defaults/main.yml    # Default variables
├── files/               # Static files
├── handlers/main.yml    # Event handlers
├── meta/main.yml        # Role metadata
├── tasks/main.yml       # Primary tasks
├── templates/           # Jinja2 templates
└── vars/main.yml        # Role variables
```

Role Usage

- **Create**: `ansible-galaxy init rolename`
- **Install**: `ansible-galaxy role install author.rolename`
- **Use in Playbook**: `roles:` section or `include_role` task

Collections

- **Install**: `ansible-galaxy collection install namespace.collection`
- **Requirements**: `requirements.yml` file

- **Usage**: FQCN format (`namespace.collection.module`)

Essential Collections

Collection	Modules	Use Case
`ansible.builtin`	Core modules	System administration
`ansible.posix`	POSIX tools	Unix/Linux systems
`community.general`	Extended modules	Additional functionality

8. Ansible Vault (15%)

Basic Operations

Command	Purpose	Usage
`create`	New encrypted file	`ansible-vault create secrets.yml`
`edit`	Modify encrypted file	`ansible-vault edit secrets.yml`
`view`	Read encrypted file	`ansible-vault view secrets.yml`
`encrypt`	Encrypt existing file	`ansible-vault encrypt file.yml`
`decrypt`	Remove encryption	`ansible-vault decrypt file.yml`
`rekey`	Change password	`ansible-vault rekey secrets.yml`

String Encryption

- **Encrypt**: `ansible-vault encrypt_string 'secret' --name 'db_password'`
- **Usage**: Embed in regular YAML files

Playbook Integration

- **Prompt**: `ansible-navigator run site.yml --ask-vault-pass`
- **File**: `ansible-navigator run site.yml --vault-password-file .vault_pass`
- **Multiple**: `ansible-navigator run site.yml --vault-id prod@prompt`

9. System Administration Tasks (25%)

Package Management

```
- name: Install packages
  ansible.builtin.dnf:
    name: ['httpd', 'php', 'mariadb-server']
    state: present

- name: Update all packages
  ansible.builtin.dnf:
    name: '*'
    state: latest
```

Service Management

```
- name: Start and enable services
  ansible.builtin.systemd:
    name: "{{ item }}"
    state: started
    enabled: yes
  loop:
    - httpd
    - mariadb
```

User Management

```
- name: Create user
  ansible.builtin.user:
    name: webuser
    groups: apache
    shell: /bin/bash
    create_home: yes
    state: present
```

Storage Management

```
- name: Create partition
  community.general.parted:
    device: /dev/sdb
    number: 1
    state: present

- name: Create volume group
  community.general.lvg:
    vg: vg_data
    pvs: /dev/sdb1

- name: Create logical volume
  community.general.lvol:
    vg: vg_data
    lv: lv_web
    size: 2G
```

Firewall Configuration

```
- name: Configure firewall
  ansible.posix.firewalld:
    service: "{{ item }}"
    permanent: yes
    state: enabled
    immediate: yes
  loop:
    - http
    - https
```

SELinux Management

```
- name: Set SELinux booleans
  ansible.posix.seboolean:
    name: httpd_can_network_connect
    state: yes
    persistent: yes
```

10. Automation Content Navigator (10%)

Execution Modes

- **Interactive (TUI)**: `ansible-navigator run playbook.yml`
- **Stdout**: `ansible-navigator run playbook.yml --mode stdout`

TUI Navigation

- `:help` - Show help
- `:doc module_name` - Module documentation
- `:collections` - Browse collections
- `:inventory` - View inventory
- `:q` - Quit

Common Options

- `--check` - Dry run mode
- `--syntax-check` - YAML validation
- `--limit group` - Target specific hosts
- `--tags tag1,tag2` - Run specific tags
- `--vault-password-file` - Vault password

Exam Strategy and Tips

Time Management (4 hours total)

1. **Read all tasks first** (15 minutes)
2. **Set up environment** (30 minutes) - SSH keys, inventory, ansible.cfg
3. **Complete tasks systematically** (3 hours)
4. **Verify and test** (15 minutes)

Critical Success Factors

- **Test connectivity first**: `ansible all -m ping`
- **Syntax check always**: `ansible-navigator run --syntax-check`
- **Use check mode**: `--check --diff` before executing
- **Know ansible-doc**: Your primary reference tool
- **FQCN required**: Always use full module names
- **Vault everything**: Encrypt all sensitive data

Common Pitfalls

- **Wrong tool**: Use `ansible-navigator` not `ansible-playbook`
- **Missing FQCN**: Short module names will fail
- **No collections**: Install required collections first

- **SSH issues**: Verify key authentication before starting
- **Syntax errors**: One mistake fails entire playbook

Documentation Strategy

- **Module parameters**: `ansible-doc module_name`
- **Examples**: `ansible-doc module_name | grep -A20 EXAMPLES`
- **Quick syntax**: `ansible-doc -s module_name`
- **All modules**: `ansible-doc -l | grep keyword`

Verification Commands

```
# Test playbook execution
ansible-navigator run site.yml --check --diff

# Verify services
ansible all -m systemd -a "name=httpd" --become

# Check files
ansible all -m stat -a "path=/etc/httpd/conf/httpd.conf"

# Test web services
ansible all -m uri -a "url=http://{{ ansible_default_ipv4.address }}"
```

Essential Module Quick Reference

System Management

Module	Key Parameters	Example
`ansible.builtin.systemd`	`name`, `state`, `enabled`	`state: started, enabled: yes`
`ansible.builtin.user`	`name`, `groups`, `state`	`name: webuser, groups: apache`
`ansible.builtin.group`	`name`, `state`, `gid`	`name: webgroup, gid: 1001`

Package Management

Module	Key Parameters	Example
`ansible.builtin.dnf`	`name`, `state`	`name: httpd, state: present`
`ansible.builtin.package`	`name`, `state`	Generic package module

File Operations

Module	Key Parameters	Example
`ansible.builtin.copy`	`src`, `dest`, `owner`, `mode`	`src: file.txt, dest: /tmp/`
`ansible.builtin.template`	`src`, `dest`, `backup`	`src: config.j2, dest: /etc/app/`
`ansible.builtin.file`	`path`, `state`, `owner`, `mode`	`path: /tmp/dir, state: directory`

Network and Security

Module	Key Parameters	Example
`ansible.posix.firewalld`	`service`, `port`, `state`	`service: http, state: enabled`
`ansible.posix.seboolean`	`name`, `state`, `persistent`	`name: httpd_can_network_connect`
`ansible.posix.mount`	`path`, `src`, `fstype`, `state`	`path: /mnt, src: /dev/sdb1`

Storage Management

Module	Key Parameters	Example
`community.general.parted`	`device`, `number`, `state`	`device: /dev/sdb, number: 1`
`community.general.lvg`	`vg`, `pvs`	`vg: vg_data, pvs: /dev/sdb1`
`community.general.lvol`	`vg`, `lv`, `size`	`vg: vg_data, lv: lv_web, size: 2G`

Final Preparation Checklist

Lab Environment Ready

- [ ] Control node with Ansible installed
- [ ] SSH keys distributed to managed nodes
- [ ] Inventory file configured and tested
- [ ] Basic playbook execution verified

Core Skills Mastered

- [ ] Ad-hoc commands for all essential modules
- [ ] Playbook structure and syntax
- [ ] Variable usage and precedence
- [ ] Template creation with Jinja2
- [ ] Role development and usage
- [ ] Vault encryption for sensitive data

System Administration Tasks

- [ ] Package installation and updates

- [ ] Service start/stop/enable
- [ ] User and group management
- [ ] File permissions and ownership
- [ ] Storage: partitions, LVM, filesystems
- [ ] Network: firewall configuration
- [ ] Security: SELinux settings

Exam Readiness

- [ ] ansible-navigator proficiency (TUI and stdout)
- [ ] ansible-doc for module reference
- [ ] Time management under pressure
- [ ] Systematic verification approach
- [ ] Troubleshooting failed tasks

Remember: The RHCE exam tests practical automation skills. Focus on hands-on practice over theoretical knowledge.

3.2 RHCE Exam Day Commands

Essential Commands for EX294 Success

Only the commands you'll actually type during the 4-hour exam

Critical Note

This is your exam day cheat sheet. Every command here is essential for RHCE exam success. These are the commands you'll type in the terminal - NOT the module parameters you'll write in playbooks.

EXAM REQUIREMENT: The RHCE exam objectives specifically require using **Automation content navigator** (`ansible-navigator`). While `ansible-playbook` commands are shown for completeness and general Ansible knowledge, you **MUST** demonstrate proficiency with `ansible-navigator` to pass the exam.

Official Exam Objectives Requiring ansible-navigator:

- "Run playbooks with Automation content navigator"
- "Use Automation content navigator to find new modules in available Ansible Content Collections"
- "Use Automation content navigator to create inventories and configure the Ansible environment"

Exam Workflow Commands

Phase 1: Initial Verification (5 minutes)

```
# Test Ansible installation and connectivity
ansible --version                        # Verify Ansible is working
ansible-navigator --version              # Verify navigator is available
ansible all -m ping                      # Test connectivity to all hosts
ansible-inventory --list                 # View inventory structure
ansible-inventory --graph               # View inventory hierarchy
ansible-galaxy collection list          # Check available collections
```

Phase 2: Documentation Lookup (Throughout Exam)

EXAM OBJECTIVE: Use Automation content navigator to find new modules in available Ansible Content Collections

```
# Navigator documentation (EXAM REQUIRED)
ansible-navigator doc module_name        # Interactive docs
ansible-navigator doc -l | grep keyword  # Search for modules in collections
ansible-navigator collections            # Browse available collections
ansible-navigator doc ansible.builtin.dnf
ansible-navigator doc community.general.firewalld
ansible-navigator doc ansible.posix.mount

# Traditional ansible-doc (also available)
ansible-doc -l | grep keyword           # Find modules quickly
ansible-doc -s module_name               # Get module syntax (fastest)
ansible-doc module_name                  # Full module documentation
ansible-doc module_name | grep -A 10 "EXAMPLES:"  # Get examples

# FQCN documentation
ansible-doc ansible.builtin.dnf
ansible-doc community.general.firewalld
ansible-doc ansible.posix.mount

# Plugin documentation
ansible-doc -t lookup file
ansible-doc -t filter default
ansible-doc -t test defined
```

Phase 3: Playbook Development & Testing (Main Phase)

EXAM REQUIRED: Use ansible-navigator

```
# Syntax validation (ALWAYS do this first)
ansible-navigator run playbook.yml --syntax-check
ansible-navigator run playbook.yml --syntax-check --mode stdout

# Dry run validation (ALWAYS do before executing)
ansible-navigator run playbook.yml --check
ansible-navigator run playbook.yml --check --diff
ansible-navigator run playbook.yml --check --diff --mode stdout

# Playbook execution (EXAM OBJECTIVE: "Run playbooks with Automation content navigator")
ansible-navigator run playbook.yml                      # Interactive TUI mode
ansible-navigator run playbook.yml --mode stdout        # Command-line output
ansible-navigator run playbook.yml --mode stdout -v     # With verbosity

# Target control
ansible-navigator run playbook.yml --limit webservers
ansible-navigator run playbook.yml --limit "web*"
ansible-navigator run playbook.yml --limit node1,node2

# Variable passing
ansible-navigator run playbook.yml -e "var=value"
ansible-navigator run playbook.yml -e "env=production debug=false"
ansible-navigator run playbook.yml -e "@vars.yml"

# Tag control
ansible-navigator run playbook.yml --tags "web,db"
ansible-navigator run playbook.yml --skip-tags "debug"
ansible-navigator run playbook.yml --list-tags

# Task control
ansible-navigator run playbook.yml --start-at-task "Install packages"
ansible-navigator run playbook.yml --step
ansible-navigator run playbook.yml --list-tasks

# Debugging levels
ansible-navigator run playbook.yml --mode stdout -v     # Basic
ansible-navigator run playbook.yml --mode stdout -vv    # More info
ansible-navigator run playbook.yml --mode stdout -vvv   # Full debug

# Alternative: ansible-playbook (general Ansible knowledge)
# ansible-playbook playbook.yml --syntax-check
# ansible-playbook playbook.yml --check --diff
```

Phase 4: Ansible Vault Operations

```
# Create encrypted files
ansible-vault create secrets.yml
ansible-vault create group_vars/all/vault.yml

# Edit encrypted files
ansible-vault edit secrets.yml
ansible-vault edit group_vars/production/vault.yml

# View encrypted content
ansible-vault view secrets.yml

# Encrypt existing files
ansible-vault encrypt vars.yml
ansible-vault encrypt host_vars/*/vault.yml

# String encryption (for inline secrets)
ansible-vault encrypt_string 'secret_password' --name 'db_password'
echo 'secret_value' | ansible-vault encrypt_string --stdin-name 'var_name'

# Change passwords
ansible-vault rekey secrets.yml

# Playbook integration (EXAM REQUIRED: ansible-navigator)
ansible-navigator run site.yml --ask-vault-pass
ansible-navigator run site.yml --vault-password-file .vault_pass

# Set up vault password file
echo 'vault_password' > .vault_pass
chmod 600 .vault_pass
```

Phase 5: Role & Collection Management

```
# Create roles
ansible-galaxy init role_name
ansible-galaxy init --init-path=./roles web_server

# Install collections
ansible-galaxy collection install community.general
ansible-galaxy collection install ansible.posix
ansible-galaxy collection install -r requirements.yml

# Install roles
ansible-galaxy role install geerlingguy.apache
ansible-galaxy role install -r requirements.yml

# Check installations
ansible-galaxy collection list
ansible-galaxy role list
```

Phase 6: Final Validation (Last 15 minutes)

```
# Test connectivity
ansible all -m ping

# Verify services (common exam validation)
ansible webservers -m systemd -a "name=httpd" --become
ansible all -m uri -a "url=http://{{ ansible_default_ipv4.address }}"

# Check file existence
ansible all -m stat -a "path=/etc/httpd/conf/httpd.conf"

# Verify mounts
ansible all -m setup -a "filter=ansible_mounts"

# Quick fact checks
ansible all -m setup -a "filter=ansible_service_mgr"
ansible all -m setup -a "filter=ansible_distribution"
```

Configuration Commands

Basic ansible.cfg Setup

```
# Check current configuration
ansible-config dump | grep -E '(INVENTORY|HOST_KEY|REMOTE_USER)'
ansible-config view

# Common configuration validation
ansible-config list | grep -i vault
ansible-config dump --only-changed
```

Inventory Validation

EXAM OBJECTIVE: Use Automation content navigator to create inventories and configure the Ansible environment

```
# Navigator inventory operations (EXAM REQUIRED)
ansible-navigator inventory --list              # Interactive inventory view
ansible-navigator inventory --list --mode stdout # Command-line inventory
ansible-navigator inventory --graph             # Tree structure view
ansible-navigator inventory --host hostname     # Single host details

# Traditional inventory commands (also available)
ansible-inventory --list
ansible-inventory --list --yaml
ansible-inventory --graph
ansible-inventory --host hostname

# Host and group listing
ansible all --list-hosts
ansible webservers --list-hosts
ansible 'web*' --list-hosts
```

Time-Saving Command Combinations

Quick Validation Sequence

```
# Use for every playbook (copy-paste ready) - EXAM REQUIRED
ansible-navigator run site.yml --syntax-check && \
ansible-navigator run site.yml --check && \
ansible-navigator run site.yml --mode stdout
```

Emergency Troubleshooting

```
# When things go wrong
ansible all -m ping -vvv
ansible-config dump | grep -E '(INVENTORY|HOST_KEY)'
ansible-inventory --list
```

Fast Documentation Lookup

```
# Speed up module discovery
alias adoc='ansible-doc'
alias adocs='ansible-doc -s'
adocs dnf    # Quick syntax
adoc user | grep -A 10 EXAMPLES:  # Quick examples
```

Exam Success Patterns

The "Big 5" Exam Commands

Master these - you'll use them constantly:

1. `ansible all -m ping` - Always start here
2. `ansible-navigator run playbook.yml --syntax-check` - Before every execution (EXAM REQUIRED)
3. `ansible-navigator run playbook.yml --check --diff` - Verify changes (EXAM REQUIRED)
4. `ansible-navigator run playbook.yml --mode stdout -v` - Execute with logging (EXAM REQUIRED)
5. `ansible-doc -s module_name` - Quick syntax lookup

Command Frequency During Exam

- **Documentation commands**: 50-100 times
- **Syntax check**: 20-30 times
- **Playbook execution**: 15-25 times
- **Vault operations**: 5-10 times
- **Inventory commands**: 5-10 times

Critical Success Factors

1. **Speed with ansible-doc** - Practice until automatic
2. **Always validate first** - Syntax check, then dry run
3. **Use verbosity for debugging** - Start with -v, increase as needed
4. **Know your FQCN** - ansible.builtin.*, community.general.*
5. **Vault everything sensitive** - No plain text passwords/keys

What NOT to Do on Exam

Commands You WON'T Use

```
# Ad-hoc module commands (these go in PLAYBOOKS, not command line)
ansible all -m dnf -a "name=httpd state=present" --become  #    WRONG
ansible all -m systemd -a "name=httpd state=started" --become  #    WRONG
ansible all -m user -a "name=webuser" --become  #    WRONG

# System administration commands (not exam relevant)
ansible all -m setup --tree /tmp/facts  #    WRONG
ssh -vvv ansible@hostname  #    WRONG
strace ansible all -m ping  #    WRONG
```

Time Wasters

- Detailed system exploration commands
- Performance monitoring commands
- Network troubleshooting commands
- SSH debugging commands

Remember: The exam is about writing PLAYBOOKS, not running ad-hoc commands for system administration!

Exam Day Strategy: Practice these commands until they're muscle memory. On exam day, you'll have no time to think about syntax - you need to execute immediately and focus your mental energy on the playbook logic, not the command syntax.

3.3 RHCE Exam Quick Reference (Cheat Sheet)

Essential Commands & Syntax for EX294 Success

Concise reference for exam day - copy-paste ready syntax and parameters

IMPORTANT: This focuses on PLAYBOOK syntax and essential commands you'll actually use on the exam. **The RHCE exam specifically requires using ansible-navigator** - see official objectives. For detailed command-line operations, see `rhce_exam_commands.md`.

Core Configuration

ansible.cfg Essential Settings

```
[defaults]
inventory = inventory.ini
remote_user = ansible
host_key_checking = False
become = True
become_method = sudo
roles_path = ./roles
collections_paths = ./collections
timeout = 30
forks = 5

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

Inventory Patterns

```
# INI Format
[webservers]
web01.example.com
web02.example.com

[databases]
db01.example.com ansible_host=192.168.1.100

[production:children]
webservers
databases

[all:vars]
ansible_user=ansible
```

SSH Setup

```
ssh-keygen -t rsa -b 4096 -N ""
ssh-copy-id ansible@managed_node
ansible all -m ping
```

Essential Exam Commands

Core Test Commands (Use These Constantly)

```
# Initial connectivity test (ALWAYS start here)
ansible all -m ping

# Quick verification commands
ansible-inventory --list                # View inventory structure
ansible-galaxy collection list          # Check available collections
```

Playbook Execution Pattern

```
# Standard validation sequence (copy-paste this) - EXAM REQUIRED
ansible-navigator run playbook.yml --syntax-check && \
ansible-navigator run playbook.yml --check && \
ansible-navigator run playbook.yml --mode stdout

# With variables and targeting
ansible-navigator run site.yml -e "env=prod" --limit webservers
```

Documentation Commands (Your Lifeline)

```
ansible-doc -s module_name             # Quick syntax (fastest)
ansible-doc module_name                # Full documentation
ansible-doc -l | grep keyword          # Find modules
```

Playbook Syntax

Basic Structure

```
---
- name: Playbook description
  hosts: target_group
  become: yes
  gather_facts: yes
  vars:
    variable_name: value
  vars_files:
    - vars/main.yml
  tasks:
    - name: Task description
      ansible.builtin.module_name:
        parameter: value
        state: present
      register: result
      when: condition
      loop: "{{ list_variable }}"
      notify: handler_name
      tags: tag_name
  handlers:
    - name: handler_name
      ansible.builtin.systemd:
        name: service_name
        state: restarted
  roles:
    - role_name
```

Task Keywords (Complete Reference)

```
- name: Task name                    # Required
  module_name:                      # Required
    parameter: value
  when: condition                   # Conditional execution
  loop: "{{ items }}"               # Iteration
  register: variable_name           # Save result
  failed_when: condition            # Custom failure
  changed_when: condition           # Custom change
  ignore_errors: yes                # Continue on failure
  no_log: yes                       # Hide from logs
  delegate_to: hostname             # Run on different host
  run_once: yes                     # Run only once
  become: yes                       # Privilege escalation
  become_user: username             # Escalate to user
  tags: [tag1, tag2]                # Task tags
  notify: handler_name              # Trigger handler
  async: 300                        # Async timeout
  poll: 5                           # Async polling
```

Essential Modules with FQCN

System Management

Module	FQCN	Key Parameters	Example
systemd	`ansible.builtin.systemd`	name, state, enabled, daemon_reload	`name: httpd, state: started, enabled: yes`
service	`ansible.builtin.service`	name, state, enabled	`name: httpd, state: started`
user	`ansible.builtin.user`	name, groups, shell, home, state	`name: webuser, groups: apache, shell: /bin/bash`
group	`ansible.builtin.group`	name, gid, state	`name: webgroup, gid: 1001`
cron	`ansible.builtin.cron`	name, job, minute, hour, user	`job: "backup.sh", minute: "0", hour: "2"`

Package Management

Module	FQCN	Key Parameters	Example
dnf	`ansible.builtin.dnf`	name, state, enablerepo, disablerepo	`name: httpd, state: present`
package	`ansible.builtin.package`	name, state	`name: httpd, state: latest`
rpm_key	`ansible.builtin.rpm_key`	key, state	`key: https://example.com/key.asc`

File Operations

Module	FQCN	Key Parameters	Example
copy	`ansible.builtin.copy`	src, dest, owner, group, mode, backup	`src: file.txt, dest: /etc/file.txt, mode: '0644'`
template	`ansible.builtin.template`	src, dest, owner, group, mode, backup	`src: config.j2, dest: /etc/config.conf`
file	`ansible.builtin.file`	path, state, owner, group, mode	`path: /tmp/dir, state: directory, mode: '0755'`
lineinfile	`ansible.builtin.lineinfile`	path, line, regexp, state	`path: /etc/hosts, line: "192.168.1.1 server"`
replace	`ansible.builtin.replace`	path, regexp, replace	`path: /etc/config, regexp: 'old', replace: 'new'`
blockinfile	`ansible.builtin.blockinfile`	path, block, marker	`path: /etc/config, block: "content here"`

Storage Management

Module	FQCN	Key Parameters	Example
parted	`community.general.parted`	device, number, state, part_type	`device: /dev/sdb, number: 1, state: present`
lvg	`community.general.lvg`	vg, pvs, state	`vg: vg_data, pvs: /dev/sdb1`
lvol	`community.general.lvol`	vg, lv, size, state	`vg: vg_data, lv: lv_web, size: 2G`
filesystem	`ansible.builtin.filesystem`	fstype, dev, opts	`fstype: xfs, dev: /dev/vg_data/lv_web`
mount	`ansible.posix.mount`	path, src, fstype, state, opts	`path: /mnt, src: /dev/sdb1, fstype: xfs, state: mounted`

Network & Security

Module	FQCN	Key Parameters	Example
firewalld	`ansible.posix.firewalld`	service, port, zone, permanent, immediate, state	`service: http, permanent: yes, immediate: yes, state: enabled`
seboolean	`ansible.posix.seboolean`	name, state, persistent	`name: httpd_can_network_connect, state: yes, persistent: yes`
selinux	`ansible.posix.selinux`	policy, state	`state: enforcing`
authorized_key	`ansible.posix.authorized_key`	user, key, state	`user: ansible, key: "{{ lookup('file', '~/.ssh/id_rsa.pub') }}"`
uri	`ansible.builtin.uri`	url, method, return_content	`url: http://example.com, method: GET`

Variables & Facts

Variable Precedence (High to Low)

1. Command line `-e`
2. Task vars
3. Block vars
4. Role and include vars
5. Set_facts / registered vars
6. Play vars_files
7. Play vars_prompt
8. Play vars
9. Host facts
10. Host vars (inventory)
11. Group vars (inventory)
12. Group vars (/all)
13. Group vars (/*)
14. Role defaults
15. Command line inventory vars
16. Default vars (deprecated)

Magic Variables

```
inventory_hostname          # Current host name
inventory_hostname_short    # Short hostname
group_names                # Groups current host belongs to
groups                     # All groups and hosts
hostvars                   # All host variables
play_hosts                 # Hosts in current play
ansible_play_batch         # Current batch of hosts
ansible_facts              # All gathered facts
```

Fact Access Patterns

```
"{{ ansible_facts['distribution'] }}"
"{{ ansible_facts['default_ipv4']['address'] }}"
"{{ ansible_facts['memtotal_mb'] }}"
"{{ ansible_facts['processor_count'] }}"
"{{ ansible_facts['devices']['sda']['size'] }}"
```

Register and Debug

```
- name: Run command
  ansible.builtin.command: uptime
  register: result

- name: Show result
  ansible.builtin.debug:
    var: result
    # or
    msg: "Uptime is {{ result.stdout }}"
```

Task Control

Conditionals

```
when: ansible_facts['distribution'] == "RedHat"
when: ansible_facts['distribution_major_version'] == "8"
when: inventory_hostname in groups['webservers']
when: result is succeeded
when: result is failed
when: variable_name is defined
when: variable_name is undefined
when: item != "excluded_item"
```

Loops

```
# Simple loop
loop:
  - item1
  - item2

# Dictionary loop
loop: "{{ users }}"
vars:
  users:
    - name: alice
      group: admins
    - name: bob
      group: users

# Range loop
loop: "{{ range(1, 6) | list }}"  # 1,2,3,4,5

# File glob loop
loop: "{{ query('fileglob', '/etc/*.conf') }}"
```

Error Handling

```
# Block structure
- name: Handle errors
  block:
    - name: Risky task
      ansible.builtin.command: /might/fail
  rescue:
    - name: Recovery task
      ansible.builtin.debug:
        msg: "Task failed, recovering"
  always:
    - name: Cleanup task
      ansible.builtin.debug:
        msg: "Always runs"

# Custom conditions
failed_when: result.rc != 0 and "ignore" not in result.stdout
changed_when: "'changes made' in result.stdout"
ignore_errors: yes
```

Templates & Jinja2

Variable Substitution

```
{{ variable_name }}
{{ ansible_facts['hostname'] }}
{{ hostvars[inventory_hostname]['custom_var'] }}
{{ groups['webservers'] | join(',') }}
```

Control Structures

```
# Conditionals
{% if ansible_facts['distribution'] == "RedHat" %}
RedHat specific config
{% elif ansible_facts['distribution'] == "Ubuntu" %}
Ubuntu specific config
{% else %}
Generic config
{% endif %}

# Loops
{% for host in groups['webservers'] %}
server {{ hostvars[host]['ansible_default_ipv4']['address'] }}
{% endfor %}

# Comments
{# This is a comment #}
```

Essential Filters

```
{{ variable | default('default_value') }}
{{ string_var | upper }}
{{ string_var | lower }}
{{ list_var | length }}
{{ list_var | join(',') }}
{{ list_var | sort }}
{{ list_var | unique }}
{{ string_var | regex_replace('old', 'new') }}
{{ dict_var | dict2items }}
{{ number_var | int }}
{{ string_var | bool }}
```

Roles & Collections

Role Structure

```
roles/rolename/
├── defaults/main.yml     # Default variables
├── files/                # Static files
├── handlers/main.yml     # Handlers
├── meta/main.yml         # Role metadata
├── tasks/main.yml        # Main tasks
├── templates/           # Jinja2 templates
└── vars/main.yml        # Role variables
```

Galaxy Commands

```
# Role operations
ansible-galaxy init rolename
ansible-galaxy role install author.rolename
ansible-galaxy role list

# Collection operations
ansible-galaxy collection install community.general
ansible-galaxy collection install ansible.posix
ansible-galaxy collection list
```

FQCN Requirements

```
# Always use Fully Qualified Collection Names
tasks:
  - name: Install package
    ansible.builtin.dnf:           # Not just 'dnf:'
      name: httpd
      state: present
```

Ansible Vault

Vault Commands

```
# File operations
ansible-vault create secrets.yml
ansible-vault edit secrets.yml
ansible-vault view secrets.yml
ansible-vault encrypt existing_file.yml
ansible-vault decrypt secrets.yml
ansible-vault rekey secrets.yml

# String encryption
ansible-vault encrypt_string 'secret_password' --name 'db_password'
```

Playbook Integration

```
# Password prompt
ansible-navigator run site.yml --ask-vault-pass

# Password file
echo 'vault_password' > .vault_pass
chmod 600 .vault_pass
ansible-navigator run site.yml --vault-password-file .vault_pass

# Multiple vault IDs
ansible-navigator run site.yml --vault-id prod@prompt --vault-id dev@.vault_pass
```

Vault File Usage

```
# In playbook
vars_files:
  - group_vars/all/vault.yml

# Encrypted string in vars
vars:
  db_password: !vault |
    $ANSIBLE_VAULT;1.1;AES256
    encrypted_content_here
```

Playbook Execution Commands

Primary Method: ansible-navigator (EXAM REQUIRED)

```
# EXAM OBJECTIVES require ansible-navigator
ansible-navigator run site.yml --syntax-check    # Always check syntax first
ansible-navigator run site.yml --check --diff    # Dry run with changes
ansible-navigator run site.yml --mode stdout     # Execute playbook

# Interactive TUI mode (also exam objective)
ansible-navigator run site.yml

# Common options
ansible-navigator run site.yml --limit webservers
ansible-navigator run site.yml -e "env=prod"
ansible-navigator run site.yml --ask-vault-pass
ansible-navigator run site.yml --mode stdout -v  # Verbosity levels
```

Alternative: ansible-playbook (General Knowledge)

```
# Traditional method (not exam-focused but good to know)
ansible-playbook site.yml --syntax-check
ansible-playbook site.yml --check --diff
ansible-playbook site.yml -v
```

TUI Navigation

```
:help                   # Show help
:doc module_name        # Module documentation
:collections            # Browse collections
:inventory              # View inventory
:q or :quit            # Exit
```

Documentation Access

```
ansible-navigator doc ansible.builtin.dnf
ansible-navigator doc -l | grep firewall
ansible-doc module_name  # Fallback command
ansible-doc -s module_name  # Synopsis only
```

Troubleshooting

Debug Strategies

```
# Debug module
- name: Show variable
  ansible.builtin.debug:
    var: variable_name
    msg: "Value is {{ variable_name }}"

# Verbosity levels (EXAM REQUIRED: ansible-navigator)
ansible-navigator run site.yml --mode stdout -v    # Basic
ansible-navigator run site.yml --mode stdout -vv   # More info
ansible-navigator run site.yml --mode stdout -vvv  # Connection debug
```

Common Patterns

```
# Check syntax (EXAM REQUIRED)
ansible-navigator run site.yml --syntax-check

# Dry run with changes (EXAM REQUIRED)
ansible-navigator run site.yml --check --diff

# Test connectivity
ansible all -m ping

# Gather facts
ansible all -m setup --tree /tmp/facts

# Check specific service
ansible all -m systemd -a "name=httpd" --become
```

Exam Success Patterns

Time-Saving Commands

```
# Quick validation sequence (EXAM REQUIRED)
ansible all -m ping && \
ansible-navigator run site.yml --syntax-check && \
ansible-navigator run site.yml --check && \
ansible-navigator run site.yml --mode stdout

# Fast documentation lookup
ansible-doc -l | grep keyword
ansible-doc -s module_name
```

Essential Verifications

```
# Services
ansible all -m systemd -a "name=httpd" --become

# Files
ansible all -m stat -a "path=/etc/httpd/conf/httpd.conf"

# Packages
ansible all -m package_facts | grep httpd

# Network
ansible all -m uri -a "url=http://{{ ansible_default_ipv4.address }}"
```

Must-Remember for Exam

- **Always use FQCN**: `ansible.builtin.dnf` not `dnf`
- **Test first**: `--syntax-check`, `--check`, then execute
- **Use ansible-navigator**: EXAM REQUIRED per official objectives
- **Know ansible-doc**: Your main reference during exam
- **Vault everything**: Encrypt all sensitive data
- **Check connectivity**: `ansible all -m ping` at start

Remember: Practice these patterns until they're automatic. Speed and accuracy win exams!

3.4 RHCE Comprehensive Command Reference

Complete Command Reference for RHCE Study & Production Use

Comprehensive reference organized by RHCE exam topics - includes all command variations, parameters, and use cases

FOR EXAM PREPARATION: This is a comprehensive learning reference. For focused exam day commands, use `rhce_exam_commands.md` and `exam_quick_reference.md` instead.

Purpose

Complete command coverage for:

- Learning and understanding all Ansible capabilities
- Production environment reference
- Comprehensive study of all command options
- Understanding the full scope of Ansible automation

Important Note

NOT for exam day: Many commands here are ad-hoc administration tasks that you'll implement in PLAYBOOKS during the actual exam.

Source Integration: Commands and patterns synthesized from:

- Sander van Vugt's RHCE Guide (16 chapters)
- Jeff Geerling's Ansible for DevOps (15 chapters)
- Michael Jang's RHCSA/RHCE Guide
- Red Hat official documentation and best practices

Exam Focus: Every command has been verified for exam relevance and includes the most commonly tested parameters and use cases.

Coverage Statistics: This comprehensive reference includes:

- 200+ essential command patterns
- 50+ module documentation lookups
- 100+ debugging and troubleshooting techniques
- 75+ vault operations and security practices
- Complete ansible-navigator TUI reference
- Advanced execution environment management
- Performance optimization techniques

1. Install and Configure Ansible Control Node

Package Installation and Setup

```
# RHEL 9 Installation (Primary Method)
sudo dnf install ansible-core python3-pip
sudo dnf install ansible-navigator        # Modern execution tool
sudo dnf install python3-argcomplete      # Command completion
sudo dnf install git                       # Version control for playbooks

# Enable EPEL for additional packages
sudo dnf install epel-release
sudo dnf install ansible-lint             # Playbook linting

# Alternative: Install via pip (if needed)
pip3 install --user ansible ansible-navigator
pip3 install --user ansible-lint yamllint

# Verify installation
ansible --version
ansible-navigator --version
ansible-lint --version
ansible-doc --version

# Show installation details
ansible --version | head -5
python3 -m ansible --version
which ansible
which ansible-navigator

# Command completion setup (optional)
activate-global-python-argcomplete --user
echo 'eval "$(register-python-argcomplete ansible)"' >> ~/.bashrc
echo 'eval "$(register-python-argcomplete ansible-config)"' >> ~/.bashrc
echo 'eval "$(register-python-argcomplete ansible-doc)"' >> ~/.bashrc

# Directory structure setup
mkdir -p ~/ansible/{playbooks,roles,inventories,group_vars,host_vars}
mkdir -p ~/ansible/collections/ansible_collections
mkdir -p ~/ansible/files/{templates,scripts}
```

Configuration Files and Management

```
# Configuration hierarchy (highest priority first):
# 1. ANSIBLE_CONFIG environment variable
# 2. ./ansible.cfg (current directory)
# 3. ~/.ansible.cfg (home directory)
# 4. /etc/ansible/ansible.cfg (global)

# Create project-specific configuration
vim ./ansible.cfg
cat > ansible.cfg << 'EOF'
[defaults]
host_key_checking = False
inventory = ./inventory.ini
roles_path = ./roles
collections_paths = ./collections
remote_user = ansible
become = True
become_method = sudo
become_user = root
become_ask_pass = False
timeout = 30
forks = 5
gathering = smart
fact_caching = memory
fact_caching_timeout = 86400
stdout_callback = yaml
bin_ansible_callbacks = True
EOF

# Global configuration
sudo vim /etc/ansible/ansible.cfg

# User-specific configuration
vim ~/.ansible.cfg

# Configuration management commands
ansible-config list                        # List all config options
ansible-config dump                        # Show all current settings
ansible-config dump --only-changed        # Show only modified settings
ansible-config view                        # Show active config file
ansible-config init --disabled > ansible.cfg  # Generate sample config

# Environment variable method
export ANSIBLE_CONFIG=~/ansible/project1/ansible.cfg
export ANSIBLE_HOST_KEY_CHECKING=False
export ANSIBLE_INVENTORY=~/ansible/inventory.ini
export ANSIBLE_ROLES_PATH=~/ansible/roles
export ANSIBLE_COLLECTIONS_PATH=~/ansible/collections
export ANSIBLE_REMOTE_USER=ansible
export ANSIBLE_BECOME=True
export ANSIBLE_BECOME_METHOD=sudo
export ANSIBLE_STDOUT_CALLBACK=yaml
export ANSIBLE_FORKS=10
export ANSIBLE_TIMEOUT=60
```

```
# Verify configuration settings
ansible-config dump | grep -E '(HOST_KEY_CHECKING|INVENTORY|REMOTE_USER|BECOME)'
ansible-config list | grep -i vault
ansible all --list-hosts

# Configuration validation
ansible-config dump | grep -E 'ERROR|WARNING'
ansible localhost -m setup | head -5
```

Inventory Management and Validation

```
# Basic inventory operations
ansible-inventory --list                   # JSON format output
ansible-inventory --list --yaml           # YAML format output
ansible-inventory --graph                 # Tree structure view
ansible-inventory --host hostname          # Single host details
ansible-inventory --host hostname --yaml  # Host details in YAML

# Inventory file validation
ansible-inventory --list -i inventory.ini
ansible-inventory --list -i inventory.yml
ansible-inventory --list -i inventory/    # Directory of inventory files
ansible-inventory --parse -i inventory.ini # Parse and validate syntax

# Multiple inventory sources
ansible-inventory --list -i inv1.ini -i inv2.yml -i inv3/
ansible-inventory --graph -i production/ -i staging.yml

# Inventory export formats
ansible-inventory --list --output inventory_export.json
ansible-inventory --list --yaml --output inventory_export.yml

# Host and group listing
ansible all --list-hosts                  # All hosts
ansible webservers --list-hosts           # Hosts in group
ansible 'web*' --list-hosts              # Pattern matching
ansible '!excluded_group' --list-hosts    # Exclusion pattern
ansible 'group1:&group2' --list-hosts     # Intersection
ansible 'group1:!group2' --list-hosts     # Difference

# Custom inventory location
export ANSIBLE_INVENTORY=./my_inventory.ini
export ANSIBLE_INVENTORY=./inventories/production/
ansible-config dump | grep INVENTORY

# Dynamic inventory testing
ansible-inventory --list -i inventory_script.py
ansible-inventory --host hostname -i inventory_script.py

# Inventory variables
ansible-inventory --host hostname --vars   # Show all variables
ansible all -m debug -a "var=group_names"  # Show group membership
ansible all -m debug -a "var=groups"       # Show all groups
ansible all -m debug -a "var=hostvars"     # Show all host variables

# Inventory debugging
ansible-inventory --graph --vars          # Show variables in graph
ansible-inventory --list | jq '.webservers' # Parse JSON output
ansible-inventory --list | grep -A5 -B5 hostname
```

2. Configure Ansible Managed Nodes

SSH Key Generation and Distribution

```
# Generate SSH key pairs (various methods)
ssh-keygen -t rsa -b 4096 -f ~/.ssh/id_rsa -N ""      # RSA 4096-bit
ssh-keygen -t ed25519 -f ~/.ssh/id_ed25519 -N ""       # Ed25519 (modern)
ssh-keygen -t ecdsa -b 521 -f ~/.ssh/id_ecdsa -N ""    # ECDSA

# Generate with comment
ssh-keygen -t rsa -b 4096 -C "ansible@$(hostname)" -f ~/.ssh/id_rsa -N ""

# Copy public key to managed nodes
ssh-copy-id ansible@node1.example.com
ssh-copy-id ansible@node2.example.com
ssh-copy-id -i ~/.ssh/id_rsa.pub ansible@node1.example.com

# Copy to multiple hosts from inventory
for host in $(ansible all --list-hosts | grep -v hosts); do
  ssh-copy-id ansible@$host
done

# Manual key distribution methods
cat ~/.ssh/id_rsa.pub | ssh user@remote_host "mkdir -p ~/.ssh && cat >> ~/.ssh/authorized_keys"
scp ~/.ssh/id_rsa.pub user@remote_host:~/.ssh/authorized_keys

# Set proper permissions
ssh user@remote_host "chmod 700 ~/.ssh && chmod 600 ~/.ssh/authorized_keys"

# Batch key distribution
ansible all -m authorized_key -a "user=ansible key='{{ lookup('file', '~/.ssh/id_rsa.pub') }}'
state=present" --ask-pass --become

# Test SSH connectivity
ssh -o StrictHostKeyChecking=no ansible@node1.example.com
ssh -o ConnectTimeout=10 ansible@node1.example.com 'echo "Connection successful"'
ssh -o BatchMode=yes ansible@node1.example.com 'uptime'  # Non-interactive test

# SSH agent setup
eval $(ssh-agent)
ssh-add ~/.ssh/id_rsa
ssh-add -l                                # List loaded keys

# SSH configuration file
vim ~/.ssh/config
cat > ~/.ssh/config << 'EOF'
Host node*
    User ansible
    StrictHostKeyChecking no
    UserKnownHostsFile /dev/null
    IdentityFile ~/.ssh/id_rsa
    ConnectTimeout 10
    ServerAliveInterval 60
    ServerAliveCountMax 3
EOF
chmod 600 ~/.ssh/config
```

Privilege Escalation Configuration

```
# Configure sudo on managed nodes
sudo visudo
# Or use dedicated sudoers file
sudo visudo -f /etc/sudoers.d/ansible

# Add to sudoers file (various patterns):
ansible ALL=(ALL) NOPASSWD: ALL          # Full access without password
ansible ALL=(ALL:ALL) NOPASSWD: ALL     # Full access to all users/groups
ansible ALL=(root) NOPASSWD: /bin/systemctl, /usr/bin/dnf  # Specific commands
%ansible ALL=(ALL) NOPASSWD: ALL        # Group-based access

# Validate sudoers syntax
sudo visudo -c
sudo visudo -c -f /etc/sudoers.d/ansible

# Test privilege escalation
ansible all -m command -a "whoami" --become
ansible all -m command -a "whoami" --become --become-user=root
ansible all -m command -a "whoami" --become --become-user=apache
ansible all -m setup --become | grep ansible_user_id

# Different become methods
ansible all -m command -a "whoami" --become --become-method=sudo
ansible all -m command -a "whoami" --become --become-method=su
ansible all -m command -a "whoami" --become --become-method=pbrun
ansible all -m command -a "whoami" --become --become-method=pfexec

# Become user variations
ansible all -m command -a "id" --become --become-user=apache
ansible all -m shell -a "ps aux | grep apache" --become
ansible all -m file -a "path=/tmp/test state=touch owner=apache" --become

# Test specific sudo commands
sudo -l                                  # List allowed commands
sudo -v                                  # Validate sudo timestamp
sudo -k                                  # Reset sudo timestamp

# Group management for ansible user
sudo usermod -aG wheel ansible           # Add to wheel group (if used)
sudo groups ansible                      # Check group membership
```

Connectivity Testing and Validation

```
# Basic connectivity tests
ansible all -m ping                      # Basic ping test
ansible all -m ping -f 10               # Parallel connections
ansible all -m ping --timeout=30        # Custom timeout
ansible all -m ping -o                  # One-line output

# Connection with specific inventory
ansible all -i inventory.ini -m ping
ansible all -i production/ -m ping

# Test specific groups and hosts
ansible webservers -m ping
ansible node1.example.com -m ping
ansible 'web*' -m ping                   # Pattern matching
ansible '!database' -m ping             # Exclude group

# Test privilege escalation
ansible all -m command -a "whoami" --become
ansible all -m command -a "whoami" --become --become-user=apache
ansible all -m shell -a "sudo -l" --become
ansible all -m command -a "id" --become

# Connection debugging
ansible all -m ping -vvv                # Verbose output
ansible all -m ping --check             # Check mode
ansible all -m ping -f 1                # Serial execution

# Gather system information
ansible all -m setup                     # All facts
ansible all -m setup --tree /tmp/facts  # Save facts to files
ansible hostname -m setup -a "filter=ansible_distribution*"
ansible hostname -m setup -a "filter=ansible_memory_mb"
ansible hostname -m setup -a "filter=ansible_processor*"
ansible hostname -m setup -a "filter=ansible_default_ipv4"
ansible all -m setup -a "gather_subset=network"
ansible all -m setup -a "gather_subset=hardware"
ansible all -m setup -a "gather_subset=!facter,!ohai"

# Network connectivity tests
ansible all -m wait_for -a "host=8.8.8.8 port=53 timeout=10"
ansible all -m uri -a "url=http://example.com return_content=no"
ansible all -m get_url -a "url=http://example.com/test.txt dest=/tmp/test.txt" --check

# File system tests
ansible all -m stat -a "path=/etc/passwd"
ansible all -m command -a "df -h"
ansible all -m command -a "free -m"
ansible all -m command -a "uptime"

# Service status checks
ansible all -m service_facts
ansible all -m systemd -a "name=sshd" | grep -i active
ansible all -m command -a "systemctl is-active sshd"
```

```
# User and group validation
ansible all -m command -a "getent passwd ansible"
ansible all -m command -a "groups ansible"
ansible all -m user -a "name=ansible" --check

# Performance testing
time ansible all -m ping
ansible all -m setup -a "gather_timeout=30"
ansible all -m command -a "uptime" --forks=20
```

---

3. Automation Content Navigator

Basic Navigation Commands

```
# Run playbooks (various modes)
ansible-navigator run site.yml                    # Interactive TUI mode
ansible-navigator run site.yml --mode stdout     # Non-interactive mode
ansible-navigator run site.yml --mode interactive # Explicit TUI mode

# Syntax and validation
ansible-navigator run site.yml --syntax-check    # Syntax validation only
ansible-navigator run site.yml --check           # Dry run mode
ansible-navigator run site.yml --check --diff    # Show changes without applying

# Execution variations
ansible-navigator run site.yml --mode stdout -v  # Verbose output
ansible-navigator run site.yml --mode stdout -vv # More verbose
ansible-navigator run site.yml --mode stdout -vvv # Maximum verbosity

# Inventory and limiting
ansible-navigator run site.yml -i inventory.ini
ansible-navigator run site.yml --limit webservers
ansible-navigator run site.yml --limit 'web*:!web3'
ansible-navigator run site.yml --limit @failed_hosts.txt

# Variable passing
ansible-navigator run site.yml -e "var=value"
ansible-navigator run site.yml -e "@vars.yml"
ansible-navigator run site.yml -e "@vars.json"

# Task control
ansible-navigator run site.yml --start-at-task "Install packages"
ansible-navigator run site.yml --step            # Step through tasks
ansible-navigator run site.yml --tags "web,db"   # Run specific tags
ansible-navigator run site.yml --skip-tags "debug" # Skip specific tags

# Interactive TUI commands
# Inside TUI navigation:
# :help          - Show help
# :doc           - Module documentation
# :collections   - Browse collections
# :inventory     - View inventory
# :images        - List execution environments
# :config        - Show configuration
# :q or :quit    - Exit
# ESC            - Go back/cancel
# Tab            - Auto-complete
# Enter          - Select/execute
# / or ?         - Search
```

Collection and Documentation Access

```
# Browse collections interactively
ansible-navigator collections                     # Interactive collection browser
ansible-navigator collections --mode stdout      # List collections in stdout
ansible-navigator collections --details          # Show collection details

# Module documentation (various approaches)
ansible-navigator doc module_name                # Interactive module docs
ansible-navigator doc module_name --mode stdout # Module docs in stdout
ansible-navigator doc -l                         # List all modules interactively
ansible-navigator doc -l --mode stdout           # List modules in stdout
ansible-navigator doc -l | grep keyword          # Search modules

# FQCN documentation
ansible-navigator doc ansible.builtin.dnf
ansible-navigator doc community.general.firewalld
ansible-navigator doc ansible.posix.mount
ansible-navigator doc containers.podman.podman_container

# Plugin documentation
ansible-navigator doc -t lookup                  # Lookup plugins
ansible-navigator doc -t filter                  # Filter plugins
ansible-navigator doc -t test                    # Test plugins
ansible-navigator doc -t callback                # Callback plugins

# Documentation search and filtering
ansible-navigator doc -l --mode stdout | grep -i package
ansible-navigator doc -l --mode stdout | grep -i user
ansible-navigator doc -l --mode stdout | grep -i service
ansible-navigator doc -l --mode stdout | wc -l  # Count available modules

# Inventory exploration
ansible-navigator inventory --list               # Interactive inventory view
ansible-navigator inventory --list --mode stdout # Inventory in stdout
ansible-navigator inventory --host hostname      # Single host details
ansible-navigator inventory --graph              # Tree structure view
ansible-navigator inventory -i inventory.ini --list
ansible-navigator inventory -i production/ --graph

# Configuration viewing
ansible-navigator config                         # Interactive config browser
ansible-navigator config --mode stdout          # Config in stdout
ansible-navigator config dump                   # All config values
```

Execution Environment Management

```
# List and manage execution environments
ansible-navigator images                         # Interactive image browser
ansible-navigator images --mode stdout          # List images in stdout
ansible-navigator images --details              # Show image details

# Common execution environments
ansible-navigator run site.yml --execution-environment-image registry.redhat.io/ubi8/ubi:latest
ansible-navigator run site.yml --execution-environment-image quay.io/ansible/ansible-
runner:latest
ansible-navigator run site.yml --execution-environment-image quay.io/ansible/creator-ee:latest

# Pull and manage container images
podman pull registry.redhat.io/ubi8/ubi:latest
podman pull quay.io/ansible/ansible-runner:latest
podman images | grep ansible
podman inspect registry.redhat.io/ubi8/ubi:latest

# Custom execution environment usage
ansible-navigator run site.yml --execution-environment-image my-custom-ee:latest
ansible-navigator run site.yml --pull-policy always
ansible-navigator run site.yml --pull-policy missing
ansible-navigator run site.yml --pull-policy never

# Volume mounting with execution environments
ansible-navigator run site.yml --execution-environment-volume-mounts /host/path:/container/path
ansible-navigator run site.yml --execution-environment-volume-mounts /etc/ansible:/etc/
ansible:ro

# Environment variable passing
ansible-navigator run site.yml --set-environment-variable ANSIBLE_HOST_KEY_CHECKING=False
ansible-navigator run site.yml --set-environment-variable ANSIBLE_TIMEOUT=30

# Container registry authentication
podman login registry.redhat.io
podman login quay.io
echo $'username\npassword' | podman login --stdin registry.example.com

# Build custom execution environment (if needed)
buildah from registry.redhat.io/ubi8/ubi:latest
buildah run $container -- dnf install -y ansible-core
buildah commit $container my-ansible-ee:latest

# Execution environment debugging
ansible-navigator run site.yml --execution-environment-image debug-ee --mode stdout -vvv
podman run -it --rm registry.redhat.io/ubi8/ubi:latest /bin/bash
```

4. Content Collections Management

Galaxy Collection Management

```
# Collection discovery and search
ansible-galaxy collection list                   # List installed collections
ansible-galaxy collection list --format json    # JSON format output
ansible-galaxy collection search firewall       # Search for collections
ansible-galaxy collection search --author redhat # Search by author

# Install collections (various methods)
ansible-galaxy collection install community.general
ansible-galaxy collection install ansible.posix
ansible-galaxy collection install containers.podman
ansible-galaxy collection install redhat.rhel_system_roles
ansible-galaxy collection install community.crypto
ansible-galaxy collection install ansible.windows

# Version-specific installations
ansible-galaxy collection install community.general:>=3.0.0
ansible-galaxy collection install community.general:==4.2.0
ansible-galaxy collection install community.general:4.2.0  # Exact version

# Install from requirements file
ansible-galaxy collection install -r requirements.yml
ansible-galaxy collection install -r requirements.yml --force
ansible-galaxy collection install -r requirements.yml -p ./collections

# Custom installation paths
ansible-galaxy collection install community.general -p ./collections
ansible-galaxy collection install community.general -p ~/.ansible/collections
export ANSIBLE_COLLECTIONS_PATHS=./collections:~/.ansible/collections

# Collection information and verification
ansible-galaxy collection list community.general
ansible-galaxy collection list | grep community
ansible-galaxy collection verify community.general
ansible-galaxy collection verify --ignore-certs community.general

# Collection building and publishing (advanced)
ansible-galaxy collection init my_namespace.my_collection
ansible-galaxy collection build                  # Build tarball
ansible-galaxy collection publish my_collection-1.0.0.tar.gz

# Upgrade and maintenance
ansible-galaxy collection install community.general --upgrade
ansible-galaxy collection install community.general --force
ansible-galaxy collection install --requirements requirements.yml --upgrade

# Alternative sources
ansible-galaxy collection install community.general --source https://private-galaxy.example.com
ansible-galaxy collection install ./my-collection-1.0.0.tar.gz
ansible-galaxy collection install git+https://github.com/user/collection.git
```

Requirements File Formats and Examples

```
# requirements.yml - Basic format
collections:
  - name: community.general
    version: ">=3.0.0"
  - name: ansible.posix
    version: "1.4.0"
  - name: containers.podman
    source: https://galaxy.ansible.com
  - name: redhat.rhel_system_roles
  - name: community.crypto
    version: ">=2.0.0"

# Advanced requirements.yml
collections:
  # Version constraints
  - name: community.general
    version: ">=4.0.0,<5.0.0"
  - name: ansible.posix
    version: "==1.4.0"              # Exact version
  - name: community.crypto
    version: "~=2.1.0"             # Compatible release

  # Alternative sources
  - name: my_namespace.my_collection
    source: https://private-galaxy.example.com
  - name: community.vmware
    source: git+https://github.com/ansible-collections/community.vmware.git
  - name: local_collection
    source: ./local-collections/

  # Git sources with specific references
  - name: community.kubernetes
    source: git+https://github.com/ansible-collections/community.kubernetes.git,main
  - name: community.docker
    source: git+https://github.com/ansible-collections/community.docker.git,v2.7.0

# Combined roles and collections
roles:
  - name: geerlingguy.apache
    version: "3.2.0"
  - src: https://github.com/geerlingguy/ansible-role-nginx
    name: nginx

collections:
  - name: community.general
  - name: ansible.posix
  - name: containers.podman

# Environment-specific requirements
# requirements-dev.yml
collections:
  - name: community.general
    version: ">=4.0.0"
  - name: community.molecule
```

```
  - name: community.crypto

# requirements-prod.yml
collections:
  - name: community.general
    version: "==4.2.0"             # Pinned for production
  - name: ansible.posix
    version: "==1.4.0"
```

Collection Usage and Module Discovery

```
# FQCN documentation lookup
ansible-navigator doc ansible.builtin.dnf
ansible-navigator doc community.general.firewalld
ansible-navigator doc ansible.posix.mount
ansible-navigator doc containers.podman.podman_container
ansible-navigator doc redhat.rhel_system_roles.selinux

# Module discovery within collections
ansible-navigator collections                    # Interactive browser
ansible-navigator doc -l | grep community.general
ansible-navigator doc -l | grep ansible.posix
ansible-navigator doc -l | grep containers.podman

# Collection path verification
ansible-config dump | grep COLLECTIONS_PATHS
ansible-galaxy collection list --format json | jq '.[].path'
find ~/.ansible/collections -name "*.py" -path "*/plugins/modules/*" | head -10

# Module testing with FQCN
ansible localhost -m ansible.builtin.debug -a "msg='Hello World'"
ansible all -m community.general.parted -a "device=/dev/sdb number=1 state=info" --check
ansible all -m ansible.posix.mount -a "path=/mnt src=/dev/sdb1 fstype=xfs state=mounted" --check

# Collection namespace verification
ansible-doc -l | cut -d. -f1-2 | sort | uniq  # List all namespaces
ansible-doc -l | grep -E '^(community|ansible|redhat)\.' | wc -l

# Plugin discovery
ansible-doc -t lookup -l | grep community
ansible-doc -t filter -l | grep ansible.builtin
ansible-doc -t callback -l
ansible-doc -t connection -l

# Collection metadata
ansible-galaxy collection list community.general --format json | jq '.[].version'
ls -la ~/.ansible/collections/ansible_collections/community/general/
cat ~/.ansible/collections/ansible_collections/community/general/MANIFEST.json

# Collection dependencies
ansible-galaxy collection list --format json | jq '.[].dependencies'
ansible-galaxy collection verify community.general --verbose

# Troubleshooting collection issues
ansible-config dump | grep -i collections
echo $ANSIBLE_COLLECTIONS_PATHS
ansible localhost -m ansible.builtin.setup -a "filter=ansible_collections"
python3 -c "import ansible_collections.community.general;
print(ansible_collections.community.general.__file__)"
```

5. Role Management

Role Creation and Structure Management

```
# Create role structure (various methods)
ansible-galaxy init role_name                    # Default role structure
ansible-galaxy init web_server                   # Example role name
ansible-galaxy init --role-skeleton=custom_skeleton role_name
ansible-galaxy init --init-path=./roles web_server
ansible-galaxy init roles/database               # Create in specific directory

# Role directory structure exploration
tree roles/role_name/
ls -la roles/role_name/
find roles/role_name/ -type f -name "*.yml" | sort

# Complete role structure:
# roles/role_name/
# ├── README.md              # Role documentation
# ├── defaults/main.yml      # Default variables (lowest priority)
# ├── files/                 # Static files to copy
# ├── handlers/main.yml      # Event handlers
# ├── meta/main.yml          # Role metadata and dependencies
# ├── tasks/main.yml         # Main task list
# ├── templates/             # Jinja2 templates
# ├── tests/                 # Test playbooks
# ├── vars/main.yml          # Role variables (higher priority)
# └── molecule/              # Testing framework (if used)

# Validate role structure
ansible-galaxy role init --offline web_server    # Offline mode
ansible-lint roles/web_server/                   # Lint role files
yamllint roles/web_server/tasks/main.yml        # YAML syntax check

# Role metadata examination
cat roles/web_server/meta/main.yml
ansible-galaxy role info geerlingguy.apache     # External role info

# Custom role skeleton creation
mkdir -p ~/.ansible/galaxy_role_skeleton/{tasks,handlers,templates,files,vars,defaults,meta}
echo '---' > ~/.ansible/galaxy_role_skeleton/tasks/main.yml
echo 'galaxy_info:' > ~/.ansible/galaxy_role_skeleton/meta/main.yml
ansible-galaxy init --role-skeleton ~/.ansible/galaxy_role_skeleton my_custom_role
```

Role Installation and Management

```
# Install roles from Ansible Galaxy
ansible-galaxy role install geerlingguy.apache
ansible-galaxy role install geerlingguy.nginx
ansible-galaxy role install davidwittman.redis
ansible-galaxy role install bertvv.dhcp

# Version-specific installations
ansible-galaxy role install geerlingguy.apache,3.2.0
ansible-galaxy role install geerlingguy.nginx:2.8.0
ansible-galaxy role install "geerlingguy.apache>=3.0.0"

# Install from requirements file
ansible-galaxy role install -r requirements.yml
ansible-galaxy role install -r requirements.yml --force
ansible-galaxy role install -r requirements.yml --roles-path ./roles

# Custom installation paths
ansible-galaxy role install geerlingguy.apache -p ./roles
ansible-galaxy role install geerlingguy.apache -p ~/.ansible/roles
export ANSIBLE_ROLES_PATH=./roles:~/.ansible/roles:/etc/ansible/roles

# Role information and verification
ansible-galaxy role list                       # List installed roles
ansible-galaxy role list --format json         # JSON format
ansible-galaxy role info geerlingguy.apache    # Role details
ansible-galaxy role search apache              # Search for roles
ansible-galaxy role search --author geerlingguy

# Role maintenance
ansible-galaxy role remove geerlingguy.apache
ansible-galaxy role remove role_name --roles-path ./roles
ansible-galaxy role install geerlingguy.apache --force  # Force reinstall

# Install from alternative sources
ansible-galaxy role install git+https://github.com/geerlingguy/ansible-role-apache.git
ansible-galaxy role install https://github.com/geerlingguy/ansible-role-nginx/archive/
master.tar.gz
ansible-galaxy role install /path/to/local/role.tar.gz

# Role dependencies management
ansible-galaxy role install -r requirements.yml --ignore-errors
ansible-galaxy role list | grep -E '(version|name)'
ansible-galaxy role list --roles-path ./roles

# Role path verification
ansible-config dump | grep ROLES_PATH
echo $ANSIBLE_ROLES_PATH
find ./roles -name "tasks" -type d 2>/dev/null
find ~/.ansible/roles -maxdepth 1 -type d 2>/dev/null
```

Role Testing and Execution

```
# Role syntax validation
ansible-navigator run --syntax-check site.yml
ansible-playbook --syntax-check site.yml         # Fallback method
ansible-lint roles/web_server/                   # Lint specific role
ansible-lint site.yml                            # Lint entire playbook

# Role execution with various options
ansible-navigator run site.yml --limit webservers
ansible-navigator run site.yml --check           # Dry run with roles
ansible-navigator run site.yml --check --diff    # Show role changes

# Variable passing to roles
ansible-navigator run site.yml -e "apache_port=8080"
ansible-navigator run site.yml -e "nginx_user=www-data"
ansible-navigator run site.yml -e "@role_vars.yml"

# Tag-based role execution
ansible-navigator run site.yml --tags "web,database"
ansible-navigator run site.yml --skip-tags "debug,test"
ansible-navigator run site.yml --tags "never"    # Run 'never' tagged tasks

# Role-specific testing
ansible-navigator run test-role.yml --limit localhost
ansible localhost -m include_role -a "name=web_server"
ansible-playbook roles/web_server/tests/test.yml

# Role debugging
ansible-navigator run site.yml --mode stdout -vv
ansible all -m debug -a "var=role_path"
ansible all -m debug -a "var=ansible_role_names"

# Role performance testing
time ansible-navigator run site.yml --limit test_host
ansible-navigator run site.yml --forks 10        # Parallel execution

# Role task analysis
ansible-navigator run site.yml --list-tasks     # List all tasks
ansible-navigator run site.yml --list-tags      # List all tags
ansible-navigator run site.yml --start-at-task "Install Apache"

# Role variable debugging
ansible all -m debug -a "var=hostvars[inventory_hostname]"
ansible webservers -m debug -a "var=apache_port | default('80')"
ansible all -m debug -a "var=group_names"
ansible all -m debug -a "var=groups"

# Multiple role execution
ansible-navigator run site.yml --limit "webservers:dbservers"
ansible-navigator run site.yml --limit "all:!excluded_group"

# Role import vs include testing
ansible-navigator run import-role-test.yml       # Static import
ansible-navigator run include-role-test.yml      # Dynamic include
```

6. Playbook Development and Execution

Playbook Syntax and Validation

```
# Syntax validation (various methods)
ansible-navigator run site.yml --syntax-check   # Navigator method
ansible-playbook site.yml --syntax-check       # Traditional method
ansible-navigator run site.yml --syntax-check --mode stdout

# YAML syntax validation
yamllint site.yml                               # YAML linting
yamllint -d '{extends: default, rules: {line-length: {max: 120}}}' site.yml
yamllint *.yml                                  # All YAML files

# Check mode (dry run) variations
ansible-navigator run site.yml --check          # Basic check mode
ansible-navigator run site.yml --check --diff   # Show differences
ansible-navigator run site.yml --check --diff --mode stdout
ansible-playbook site.yml --check --diff       # Fallback method

# Advanced linting
ansible-lint site.yml                           # Basic linting
ansible-lint site.yml -v                        # Verbose output
ansible-lint roles/                              # Lint all roles
ansible-lint --exclude .github/ site.yml       # Exclude directories
ansible-lint --skip-list yaml site.yml         # Skip specific rules
ansible-lint --write site.yml                   # Auto-fix issues

# Multiple file validation
ansible-lint *.yml
ansible-lint playbooks/ roles/ group_vars/
find . -name "*.yml" -exec ansible-lint {} \;
find . -name "*.yml" | xargs yamllint

# Custom lint configurations
echo 'skip_list:' > .ansible-lint
echo '  - yaml[line-length]' >> .ansible-lint
echo '  - risky-file-permissions' >> .ansible-lint
ansible-lint -c .ansible-lint site.yml

# Validation with specific inventory
ansible-navigator run site.yml --syntax-check -i inventory.ini
ansible-navigator run site.yml --check -i production/
ansible-navigator run site.yml --check --limit webservers

# Task-specific validation
ansible localhost -m debug -a "msg='{{ variable_name | default('undefined') }}'"
ansible localhost -m template -a "src=template.j2 dest=/tmp/test" --check
```

Playbook Execution Options and Control

```
# Verbosity levels
ansible-navigator run site.yml --mode stdout -v    # Basic verbosity
ansible-navigator run site.yml --mode stdout -vv   # More verbose
ansible-navigator run site.yml --mode stdout -vvv  # Maximum verbosity
ansible-navigator run site.yml --mode stdout -vvvv # Connection debugging

# Host and group limiting
ansible-navigator run site.yml --limit webservers
ansible-navigator run site.yml --limit "node1,node2"
ansible-navigator run site.yml --limit "webservers:!node3"
ansible-navigator run site.yml --limit "web*"      # Pattern matching
ansible-navigator run site.yml --limit "@failed_hosts.txt"
ansible-navigator run site.yml --limit "all:!excluded_group"
ansible-navigator run site.yml --limit "webservers:&production"

# Task execution control
ansible-navigator run site.yml --start-at-task "Install packages"
ansible-navigator run site.yml --step              # Interactive step-through
ansible-navigator run site.yml --tags "web,db"     # Run specific tags
ansible-navigator run site.yml --skip-tags "debug" # Skip specific tags
ansible-navigator run site.yml --tags "never"      # Force 'never' tags
ansible-navigator run site.yml --list-tasks        # List all tasks
ansible-navigator run site.yml --list-tags         # List all tags

# Inventory and connection options
ansible-navigator run site.yml -i inventory.ini
ansible-navigator run site.yml -i inventory.yml
ansible-navigator run site.yml -i production/      # Directory inventory
ansible-navigator run site.yml -i host1,host2,     # Inline inventory

# Parallel execution control
ansible-navigator run site.yml --forks 10          # 10 parallel processes
ansible-navigator run site.yml --forks 1           # Serial execution
ansible-navigator run site.yml --serial 2          # Batch size control

# Connection and timeout options
ansible-navigator run site.yml --timeout 60        # Command timeout
ansible-navigator run site.yml --connection local  # Local connection
ansible-navigator run site.yml --connection ssh    # SSH connection
ansible-navigator run site.yml --private-key ~/.ssh/id_rsa

# Privilege escalation
ansible-navigator run site.yml --become            # Enable privilege escalation
ansible-navigator run site.yml --become-user apache # Specific become user
ansible-navigator run site.yml --become-method sudo # Specific method
ansible-navigator run site.yml --ask-become-pass   # Prompt for password

# Output and logging control
ansible-navigator run site.yml --mode stdout       # Standard output
ansible-navigator run site.yml --mode interactive  # TUI mode
ansible-navigator run site.yml --one-line          # Condensed output
ansible-navigator run site.yml --tree /tmp/results # Save results to directory

# Error handling and recovery
```

```
ansible-navigator run site.yml --force-handlers    # Run handlers on failure
ansible-navigator run site.yml --flush-cache       # Clear fact cache
ansible-navigator run site.yml --diff              # Show file changes

# Strategy control
ansible-navigator run site.yml --strategy linear   # Default strategy
ansible-navigator run site.yml --strategy free     # Don't wait for all hosts
ansible-navigator run site.yml --strategy debug    # Debug strategy
```

Variable Management and Debugging

```
# Command line variable passing (various formats)
ansible-navigator run site.yml -e "var=value"
ansible-navigator run site.yml -e "apache_port=8080"
ansible-navigator run site.yml -e "env=production debug=false"
ansible-navigator run site.yml -e '{"apache_port": 8080, "ssl_enabled": true}'
ansible-navigator run site.yml -e "@vars.yml"      # From YAML file
ansible-navigator run site.yml -e "@vars.json"     # From JSON file
ansible-navigator run site.yml -e "@/path/to/external_vars.yml"

# Multiple variable sources
ansible-navigator run site.yml -e "@group_vars/production.yml" -e "debug=true"
ansible-navigator run site.yml -e "@secrets.yml" --vault-password-file .vault_pass

# Variable precedence testing and debugging
ansible all -m debug -a "var=my_variable"          # Single variable
ansible all -m debug -a "var=hostvars[inventory_hostname]" # All host vars
ansible all -m debug -a "var=group_names"           # Group membership
ansible all -m debug -a "var=groups"                # All groups
ansible all -m debug -a "var=ansible_facts"         # All facts
ansible all -m debug -a "var=vars"                  # All variables

# Specific variable debugging
ansible all -m debug -a "var=ansible_default_ipv4.address"
ansible all -m debug -a "var=ansible_distribution"
ansible all -m debug -a "var=ansible_hostname"
ansible all -m debug -a "var=ansible_user"

# Variable file validation
ansible all -m debug -a "var=lookup('file', '/path/to/file.txt')"
ansible all -m debug -a "var=lookup('env', 'HOME')"
ansible all -m debug -a "var=lookup('pipe', 'date')"

# Magic variables debugging
ansible all -m debug -a "var=inventory_hostname"
ansible all -m debug -a "var=inventory_hostname_short"
ansible all -m debug -a "var=play_hosts"
ansible all -m debug -a "var=ansible_play_batch"
ansible all -m debug -a "var=ansible_play_hosts_all"

# Variable precedence order testing
echo 'test_var: from_command_line' > test_vars.yml
ansible-navigator run site.yml -e "@test_vars.yml" -e "test_var=override"
ansible all -m debug -a "var=test_var"

# Variable filtering and selection
ansible all -m debug -a "var=ansible_facts" | grep -A 5 -B 5 "distribution"
ansible all -m debug -a "var=ansible_facts.keys() | list"
ansible all -m debug -a "var=hostvars.keys() | list"

# Environment variable testing
export ANSIBLE_VAR_custom_var="environment_value"
ansible all -m debug -a "var=custom_var"
ansible-navigator run site.yml -e "env_var={{ lookup('env', 'PATH') }}"
```

```
# Variable validation in playbooks
ansible all -m debug -a "var=variable_name | default('NOT_DEFINED')"
ansible all -m debug -a "var=variable_name is defined"
ansible all -m debug -a "var=variable_name is undefined"
```

---

7. RHCSA Task Automation Commands

Package Management Automation

```
# Module documentation and discovery
ansible-navigator doc ansible.builtin.dnf       # Primary package manager
ansible-navigator doc ansible.builtin.package   # Generic package module
ansible-navigator doc ansible.builtin.yum       # Legacy YUM module
ansible-navigator doc ansible.builtin.rpm_key   # GPG key management
ansible-doc -s dnf                              # Synopsis only

# Basic package operations
ansible all -m dnf -a "name=httpd state=present" --become
ansible all -m dnf -a "name=httpd state=latest" --become
ansible all -m dnf -a "name=httpd state=absent" --become
ansible all -m dnf -a "name=httpd state=installed" --become  # Alias for present
ansible all -m dnf -a "name=httpd state=removed" --become    # Alias for absent

# Multiple package operations
ansible all -m dnf -a "name=['httpd','nginx','mariadb-server'] state=present" --become
ansible all -m dnf -a "name=httpd,nginx,mysql-server state=present" --become
ansible all -m dnf -a "name='@Development Tools' state=present" --become  # Group install
ansible all -m dnf -a "name='@^Minimal Install' state=present" --become  # Environment group

# Version-specific installations
ansible all -m dnf -a "name=httpd-2.4.* state=present" --become
ansible all -m dnf -a "name=kernel state=present" --become
ansible all -m dnf -a "name=kernel state=latest" --become

# Repository management
ansible all -m dnf -a "name=epel-release state=present" --become
ansible all -m dnf -a "name=httpd state=present enablerepo=epel" --become
ansible all -m dnf -a "name=httpd state=present disablerepo=epel" --become

# Package queries and information
ansible all -m package_facts
ansible all -m debug -a "var=ansible_facts.packages.httpd" --become
ansible all -m command -a "rpm -qa httpd"
ansible all -m command -a "dnf list installed httpd" --become
ansible all -m command -a "dnf info httpd" --become

# Cache and cleanup operations
ansible all -m dnf -a "update_cache=yes" --become
ansible all -m dnf -a "autoremove=yes" --become
ansible all -m command -a "dnf clean all" --become
ansible all -m command -a "dnf makecache" --become

# Security updates
ansible all -m dnf -a "name='*' state=latest security=yes" --become
ansible all -m command -a "dnf check-update --security" --become
ansible all -m command -a "dnf update --security -y" --become

# GPG key management
ansible all -m rpm_key -a "state=present key=https://dl.fedoraproject.org/pub/epel/RPM-GPG-KEY-
EPEL-9" --become
ansible all -m command -a "rpm --import /path/to/key.asc" --become

# Package file operations
```

```
ansible all -m dnf -a "name=/path/to/package.rpm state=present" --become
ansible all -m command -a "rpm -ivh /path/to/package.rpm" --become
ansible all -m get_url -a "url=http://example.com/package.rpm dest=/tmp/package.rpm" --become

# Downgrade and specific version management
ansible all -m command -a "dnf downgrade httpd -y" --become
ansible all -m command -a "dnf list --showduplicates httpd" --become

# Testing package operations
ansible all -m dnf -a "name=httpd state=present" --become --check
ansible all -m package_facts --become
ansible all -m command -a "which httpd"
```

Service Management Automation

```
# Module documentation and discovery
ansible-navigator doc ansible.builtin.systemd   # Systemd service management
ansible-navigator doc ansible.builtin.service   # Generic service module
ansible-navigator doc ansible.builtin.systemd_service  # Alias for systemd
ansible-doc -s systemd                          # Synopsis only

# Basic service operations
ansible all -m systemd -a "name=httpd state=started" --become
ansible all -m systemd -a "name=httpd state=stopped" --become
ansible all -m systemd -a "name=httpd state=restarted" --become
ansible all -m systemd -a "name=httpd state=reloaded" --become

# Service enablement and startup
ansible all -m systemd -a "name=httpd enabled=yes" --become
ansible all -m systemd -a "name=httpd enabled=no" --become
ansible all -m systemd -a "name=httpd state=started enabled=yes" --become
ansible all -m systemd -a "name=httpd state=stopped enabled=no" --become

# Systemd daemon operations
ansible all -m systemd -a "daemon_reload=yes" --become
ansible all -m systemd -a "daemon_reexec=yes" --become
ansible all -m command -a "systemctl daemon-reload" --become

# Service status and information
ansible all -m service_facts --become
ansible all -m debug -a "var=ansible_facts.services['httpd.service']" --become
ansible all -m command -a "systemctl status httpd" --become
ansible all -m command -a "systemctl is-active httpd"
ansible all -m command -a "systemctl is-enabled httpd"
ansible all -m command -a "systemctl is-failed httpd"

# Multiple service operations
ansible all -m systemd -a "name=httpd,nginx state=started enabled=yes" --become
for service in httpd nginx mariadb; do
  ansible all -m systemd -a "name=$service state=started enabled=yes" --become
done

# Service masking and unmasking
ansible all -m systemd -a "name=httpd masked=yes" --become
ansible all -m systemd -a "name=httpd masked=no" --become
ansible all -m command -a "systemctl mask httpd" --become
ansible all -m command -a "systemctl unmask httpd" --become

# Target and runlevel management
ansible all -m systemd -a "name=multi-user.target state=started" --become
ansible all -m command -a "systemctl get-default" --become
ansible all -m command -a "systemctl set-default multi-user.target" --become
ansible all -m systemd -a "name=graphical.target state=started" --become

# Timer management
ansible all -m systemd -a "name=backup.timer state=started enabled=yes" --become
ansible all -m command -a "systemctl list-timers" --become

# Socket management
```

```
ansible all -m systemd -a "name=httpd.socket state=started enabled=yes" --become
ansible all -m command -a "systemctl list-sockets" --become

# Service dependency analysis
ansible all -m command -a "systemctl list-dependencies httpd" --become
ansible all -m command -a "systemctl show httpd" --become

# Emergency service operations
ansible all -m command -a "systemctl kill httpd" --become
ansible all -m command -a "systemctl kill -s KILL httpd" --become
ansible all -m command -a "systemctl reset-failed httpd" --become

# Service testing and validation
ansible all -m systemd -a "name=httpd state=started" --become --check
ansible all -m uri -a "url=http://{{ ansible_default_ipv4.address }} method=GET"
ansible all -m wait_for -a "port=80 host={{ ansible_default_ipv4.address }} timeout=10"
```

File Management Automation

```
# Module documentation and discovery
ansible-navigator doc ansible.builtin.copy      # Copy files
ansible-navigator doc ansible.builtin.template  # Template files with Jinja2
ansible-navigator doc ansible.builtin.file      # File and directory operations
ansible-navigator doc ansible.builtin.fetch     # Fetch files from remote
ansible-navigator doc ansible.builtin.stat      # File statistics
ansible-navigator doc ansible.builtin.find      # Find files
ansible-navigator doc ansible.builtin.replace   # Replace text in files
ansible-navigator doc ansible.builtin.lineinfile # Manage lines in files

# Basic file copy operations
ansible all -m copy -a "src=file.txt dest=/tmp/file.txt" --become
ansible all -m copy -a "src=config.conf dest=/etc/app/config.conf backup=yes" --become
ansible all -m copy -a "content='Hello World' dest=/tmp/hello.txt" --become
ansible all -m copy -a "src=files/ dest=/tmp/ owner=apache group=apache mode=0644" --become

# Template operations
ansible all -m template -a "src=httpd.conf.j2 dest=/etc/httpd/conf/httpd.conf" --become
ansible all -m template -a "src=template.j2 dest=/tmp/result.txt backup=yes" --become
ansible all -m template -a "src=config.j2 dest=/etc/config owner=root mode=0600" --become

# Directory operations
ansible all -m file -a "path=/tmp/testdir state=directory" --become
ansible all -m file -a "path=/var/log/app state=directory owner=apache group=apache mode=0755" -
-become
ansible all -m file -a "path=/tmp/testdir state=absent" --become  # Remove directory
ansible all -m file -a "path=/opt/app state=directory recurse=yes owner=app group=app" --become

# File creation and modification
ansible all -m file -a "path=/tmp/testfile state=touch" --become
ansible all -m file -a "path=/tmp/testfile state=touch owner=apache group=apache mode=0644" --
become
ansible all -m file -a "path=/tmp/oldfile state=absent" --become  # Remove file

# Symbolic and hard links
ansible all -m file -a "src=/etc/hosts dest=/tmp/hosts_link state=link" --become
ansible all -m file -a "src=/etc/hosts dest=/tmp/hosts_hard state=hard" --become
ansible all -m file -a "path=/tmp/broken_link state=absent" --become

# File permissions and ownership
ansible all -m file -a "path=/tmp/testfile owner=apache group=apache mode=0644" --become
ansible all -m file -a "path=/tmp/testfile mode=u+rw,g+r,o-rwx" --become
ansible all -m file -a "path=/var/www/html recurse=yes owner=apache group=apache" --become

# File content management
ansible all -m lineinfile -a "path=/etc/hosts line='192.168.1.100 myserver' state=present" --
become
ansible all -m lineinfile -a "path=/etc/ssh/sshd_config regexp='^#?PasswordAuthentication'
line='PasswordAuthentication no'" --become
ansible all -m replace -a "path=/etc/config.conf regexp='old_value' replace='new_value'" --
become

# File statistics and information
ansible all -m stat -a "path=/etc/passwd"
```

```
ansible all -m stat -a "path=/tmp/testfile get_checksum=yes get_mime=yes"
ansible all -m debug -a "var=stat_result"

# Find files and directories
ansible all -m find -a "paths=/var/log pattern='*.log' age=7d age_stamp=mtime"
ansible all -m find -a "paths=/tmp pattern='*temp*' file_type=file"
ansible all -m find -a "paths=/etc pattern='*.conf' recurse=yes"

# Fetch files from remote hosts
ansible all -m fetch -a "src=/etc/hostname dest=./fetched_files/"
ansible all -m fetch -a "src=/var/log/messages dest=./logs/ flat=yes"

# File archiving and compression
ansible all -m archive -a "path=/var/log dest=/tmp/logs.tar.gz format=gz" --become
ansible all -m unarchive -a "src=files.tar.gz dest=/tmp/ remote_src=yes" --become
ansible all -m unarchive -a "src=/tmp/archive.tar.gz dest=/opt/ owner=apache group=apache" --
become

# File validation and testing
ansible all -m copy -a "src=test.txt dest=/tmp/test.txt" --become --check
ansible all -m file -a "path=/tmp/testdir state=directory" --become --check
ansible all -m command -a "ls -la /tmp/"
ansible all -m command -a "file /tmp/testfile"

# Bulk file operations
ansible all -m shell -a "find /tmp -name '*.tmp' -delete" --become
ansible all -m shell -a "find /var/log -name '*.log' -mtime +30 -exec rm {} \;" --become
ansible all -m command -a "du -sh /var/log" --become
```

Storage Management

```
# Module documentation
ansible-navigator doc community.general.parted
ansible-navigator doc community.general.lvg
ansible-navigator doc community.general.lvol
ansible-navigator doc ansible.posix.mount

# Test storage commands
ansible all -m setup -a "filter=ansible_devices" --become
ansible all -m setup -a "filter=ansible_mounts" --become
```

User Management

```
# Module documentation
ansible-navigator doc ansible.builtin.user
ansible-navigator doc ansible.builtin.group

# Ad-hoc user commands
ansible all -m user -a "name=testuser state=present" --become
ansible all -m group -a "name=testgroup state=present" --become
```

8. Ansible Vault Operations

Comprehensive Vault Operations

```
# Create encrypted files (various methods)
ansible-vault create secrets.yml             # Interactive creation
ansible-vault create group_vars/webservers/vault.yml
ansible-vault create host_vars/web01/vault.yml
ansible-vault create --vault-id dev@prompt secrets.yml  # With vault ID
ansible-vault create --vault-id prod@.vault_pass_prod production_secrets.yml

# Edit encrypted files
ansible-vault edit secrets.yml               # Basic editing
ansible-vault edit secrets.yml --vault-id dev@prompt
ansible-vault edit secrets.yml --vault-password-file .vault_pass
ansible-vault edit group_vars/all/vault.yml

# View encrypted file contents
ansible-vault view secrets.yml               # Read-only viewing
ansible-vault view secrets.yml --vault-id dev@prompt
ansible-vault view secrets.yml --vault-password-file .vault_pass
ansible-vault view group_vars/production/vault.yml

# Encrypt existing files
ansible-vault encrypt vars.yml               # Encrypt plain file
ansible-vault encrypt host_vars/web01/secrets.yml
ansible-vault encrypt group_vars/*/vault.yml  # Multiple files
ansible-vault encrypt --vault-id prod@prompt production_vars.yml
ansible-vault encrypt --vault-password-file .vault_pass sensitive_data.yml

# Decrypt files
ansible-vault decrypt secrets.yml            # Decrypt to plain text
ansible-vault decrypt --output decrypted.yml secrets.yml
ansible-vault decrypt group_vars/dev/vault.yml
ansible-vault decrypt --vault-id dev@prompt secrets.yml

# String encryption (inline secrets)
ansible-vault encrypt_string 'secret_password' --name 'db_password'
ansible-vault encrypt_string 'my_secret' --name 'api_key' --vault-id dev@prompt
ansible-vault encrypt_string --stdin-name 'ssh_key' < ~/.ssh/id_rsa
echo 'secret_value' | ansible-vault encrypt_string --stdin-name 'var_name'

# Change vault passwords (rekey)
ansible-vault rekey secrets.yml              # Change password
ansible-vault rekey secrets.yml --new-vault-id prod@prompt
ansible-vault rekey --vault-id old@prompt --new-vault-id new@prompt secrets.yml
ansible-vault rekey group_vars/*/vault.yml   # Multiple files

# Vault ID management
ansible-vault create --vault-id dev@prompt dev_secrets.yml
ansible-vault create --vault-id prod@.vault_pass_prod prod_secrets.yml
ansible-vault edit --vault-id dev@prompt dev_secrets.yml
ansible-vault view --vault-id prod@.vault_pass_prod prod_secrets.yml

# Password file methods
echo 'my_vault_password' > .vault_pass
chmod 600 .vault_pass
ansible-vault create --vault-password-file .vault_pass secrets.yml
```

```
ansible-vault edit --vault-password-file .vault_pass secrets.yml

# Multiple vault passwords
echo 'dev_password' > .vault_pass_dev
echo 'prod_password' > .vault_pass_prod
chmod 600 .vault_pass_*
ansible-vault create --vault-id dev@.vault_pass_dev dev_secrets.yml
ansible-vault create --vault-id prod@.vault_pass_prod prod_secrets.yml

# Vault validation and troubleshooting
ansible-vault view secrets.yml --check        # Validate encrypted file
file secrets.yml                              # Check if file is encrypted
head -1 secrets.yml | grep -q \$ANSIBLE_VAULT && echo "Encrypted" || echo "Plain text"
```

Vault Integration with Playbooks

```
# Basic vault integration
ansible-navigator run site.yml --ask-vault-pass  # Interactive password prompt
ansible-playbook site.yml --ask-vault-pass       # Fallback method

# Password file integration
echo 'vault_password' > .vault_pass
chmod 600 .vault_pass
ansible-navigator run site.yml --vault-password-file .vault_pass
ansible-navigator run site.yml --vault-password-file ~/.ansible_vault_pass

# Multiple vault IDs with playbooks
ansible-navigator run site.yml --vault-id dev@.vault_pass_dev
ansible-navigator run site.yml --vault-id prod@prompt
ansible-navigator run site.yml --vault-id dev@.vault_pass_dev --vault-id prod@prompt
ansible-navigator run site.yml --vault-id @prompt  # Default vault ID

# Environment variable method
export ANSIBLE_VAULT_PASSWORD_FILE=.vault_pass
ansible-navigator run site.yml
export ANSIBLE_VAULT_IDENTITY_LIST="dev@.vault_pass_dev,prod@.vault_pass_prod"
ansible-navigator run multi_env.yml

# Vault script integration
cat > .vault_pass_script.sh << 'EOF'
#!/bin/bash
echo "$VAULT_PASSWORD"
EOF
chmod +x .vault_pass_script.sh
export VAULT_PASSWORD="my_secret_password"
ansible-navigator run site.yml --vault-password-file .vault_pass_script.sh

# Check mode with vault
ansible-navigator run site.yml --ask-vault-pass --check
ansible-navigator run site.yml --vault-password-file .vault_pass --check --diff

# Debugging vault issues
ansible-navigator run site.yml --ask-vault-pass -vvv
ansible all -m debug -a "var=encrypted_variable" --ask-vault-pass
ansible all -m debug -a "var=vault_encrypted_variable" --vault-password-file .vault_pass

# Testing vault decryption
ansible localhost -m debug -a "var=secret_value" -e "@group_vars/all/vault.yml" --ask-vault-pass
ansible-navigator run test_vault.yml --vault-id dev@prompt --limit localhost

# Mixed encrypted and unencrypted variables
ansible-navigator run site.yml -e "@vars.yml" -e "@vault_vars.yml" --ask-vault-pass
ansible-navigator run site.yml --vault-id dev@.vault_pass_dev -e "environment=development"

# Vault with different execution modes
ansible-navigator run site.yml --ask-vault-pass --mode stdout
ansible-navigator run site.yml --vault-password-file .vault_pass --forks 1
ansible-navigator run site.yml --vault-id prod@prompt --limit production

# Advanced vault scenarios
```

```
ansible-navigator run site.yml --vault-id @prompt --extra-vars "@encrypted_extra_vars.yml"
ansible-navigator run deploy.yml --vault-id app@.vault_pass_app --vault-id db@.vault_pass_db

# Ansible configuration for vault
echo '[defaults]' > ansible.cfg
echo 'vault_password_file = .vault_pass' >> ansible.cfg
echo 'vault_identity_list = dev@.vault_pass_dev, prod@.vault_pass_prod' >> ansible.cfg

# Testing vault configuration
ansible-config dump | grep -i vault
ansible localhost -m debug -a "msg='Vault configuration working'" -e "@vault_test.yml"
```

9. Debugging and Troubleshooting

Advanced Playbook Debugging

```
# Debug mode with various verbosity levels
ansible-navigator run site.yml --mode stdout -v    # Basic verbosity
ansible-navigator run site.yml --mode stdout -vv   # More details
ansible-navigator run site.yml --mode stdout -vvv  # Full debug output
ansible-navigator run site.yml --mode stdout -vvvv # Connection debugging

# Show differences and changes
ansible-navigator run site.yml --check --diff      # Show proposed changes
ansible-navigator run site.yml --diff              # Show actual changes
ansible-navigator run site.yml --check --diff --mode stdout

# Variable debugging (comprehensive)
ansible all -m debug -a "var=ansible_facts"        # All system facts
ansible all -m debug -a "var=hostvars[inventory_hostname]"  # All host variables
ansible all -m debug -a "var=group_names"           # Host's groups
ansible all -m debug -a "var=groups"                # All groups
ansible all -m debug -a "var=play_hosts"            # Hosts in current play
ansible all -m debug -a "var=inventory_hostname"    # Current host name

# Fact debugging and filtering
ansible all -m debug -a "var=ansible_facts.keys() | list"  # Available fact categories
ansible all -m debug -a "var=ansible_default_ipv4"
ansible all -m debug -a "var=ansible_distribution_version"
ansible all -m debug -a "var=ansible_processor_count"
ansible all -m debug -a "var=ansible_memory_mb"
ansible all -m debug -a "var=ansible_mounts"
ansible all -m debug -a "var=ansible_interfaces"

# Connection and authentication debugging
ansible all -m ping -vvv                           # Verbose connection test
ansible all -m setup --tree /tmp/facts             # Save facts for analysis
ansible all -m setup -a "filter=ansible_ssh*"      # SSH-related facts
ansible all -m debug -a "var=ansible_user"
ansible all -m debug -a "var=ansible_connection"

# Task-level debugging
ansible-navigator run site.yml --start-at-task "Debug task" --mode stdout -v
ansible-navigator run site.yml --step --mode stdout  # Step through tasks
ansible-navigator run site.yml --list-tasks        # Show all tasks
ansible-navigator run site.yml --list-hosts        # Show target hosts

# Performance and timing debugging
time ansible-navigator run site.yml --mode stdout
ansible-navigator run site.yml --mode stdout | grep -E "(TASK|PLAY|changed|ok|failed)"
ansible all -m setup -a "gather_timeout=30"

# Error analysis and recovery
ansible-navigator run site.yml --mode stdout | grep -A 5 -B 5 "FAILED"
ansible-navigator run site.yml --force-handlers    # Run handlers even on failure
ansible-navigator run site.yml --mode stdout 2>&1 | tee ansible_debug.log

# Module-specific debugging
ansible all -m debug -a "msg='Testing debug module'"
ansible all -m debug -a "msg={{ variable_name | default('undefined') }}"
```

```
ansible all -m debug -a "var=item" -e "item=test_value"
ansible all -m assert -a "that: ansible_os_family == 'RedHat'"

# Conditional debugging
ansible all -m debug -a "msg='This is a Red Hat system'" --limit "ansible_os_family == 'RedHat'"
ansible all -m debug -a "var=my_var" -e "my_var=test" when="my_var is defined"

# JSON and structured output debugging
ansible all -m setup --tree /tmp/facts
cat /tmp/facts/hostname | jq '.ansible_facts.ansible_default_ipv4'
ansible all -m debug -a "var=ansible_facts" | grep -A 20 -B 5 "default_ipv4"
```

Comprehensive Module Testing

```
# System information modules
ansible hostname -m setup                      # Gather all facts
ansible hostname -m setup -a "filter=ansible_distribution*"
ansible hostname -m setup -a "filter=ansible_memory*"
ansible hostname -m setup -a "filter=ansible_processor*"
ansible hostname -m setup -a "gather_subset=network,hardware"
ansible hostname -m setup -a "gather_subset=!facter,!ohai"

# Command execution testing
ansible hostname -m command -a "uptime"
ansible hostname -m command -a "free -m"
ansible hostname -m command -a "df -h"
ansible hostname -m command -a "ps aux | head -10"
ansible hostname -m shell -a "df -h | grep /"
ansible hostname -m shell -a "netstat -tuln | grep :80"

# User and permission testing
ansible all -m command -a "whoami"
ansible all -m command -a "whoami" --become
ansible all -m command -a "whoami" --become --become-user=apache
ansible all -m command -a "id" --become
ansible all -m command -a "groups $(whoami)"
ansible all -m command -a "sudo -l" --become

# File system testing
ansible all -m stat -a "path=/etc/passwd"
ansible all -m stat -a "path=/tmp get_checksum=yes"
ansible all -m file -a "path=/tmp/test state=touch" --check
ansible all -m copy -a "content='test' dest=/tmp/test.txt" --check
ansible all -m command -a "ls -la /tmp/"

# Network connectivity testing
ansible all -m wait_for -a "host=8.8.8.8 port=53 timeout=5"
ansible all -m uri -a "url=http://httpbin.org/get method=GET" --check
ansible all -m get_url -a "url=http://httpbin.org/uuid dest=/tmp/uuid.json" --check

# Service testing
ansible all -m service_facts --become
ansible all -m systemd -a "name=sshd" --become | grep -i active
ansible all -m command -a "systemctl status sshd --no-pager" --become
ansible all -m command -a "systemctl is-active sshd"
ansible all -m command -a "systemctl is-enabled sshd"

# Package testing
ansible all -m package_facts --become
ansible all -m command -a "rpm -qa | grep httpd"
ansible all -m dnf -a "name=httpd state=present" --become --check
ansible all -m debug -a "var=ansible_facts.packages.httpd" --become

# User management testing
ansible all -m user -a "name=testuser" --become --check
ansible all -m group -a "name=testgroup" --become --check
ansible all -m command -a "getent passwd testuser"
ansible all -m command -a "getent group testgroup"
```

```
# Archive and compression testing
ansible all -m archive -a "path=/var/log dest=/tmp/logs.tar.gz" --become --check
ansible all -m unarchive -a "src=/tmp/test.tar.gz dest=/tmp/" --check

# Template and variable testing
ansible all -m template -a "src=test.j2 dest=/tmp/test.out" --check
ansible all -m debug -a "msg='Variable test: {{ ansible_hostname }}'"
ansible all -m debug -a "msg={{ 'hello world' | upper }}"

# Error testing and validation
ansible all -m command -a "exit 1" --ignore-errors
ansible all -m fail -a "msg='This is a test failure'" --check
ansible all -m assert -a "that: 1 == 1 quiet=yes"
ansible all -m assert -a "that: ansible_os_family == 'RedHat' fail_msg='Not a Red Hat system'"

# Module parameter testing
ansible all -m debug -a "var=ansible_module_args"
ansible localhost -m debug -a "var=omit"
ansible all -m copy -a "content={{ 'test' if true else omit }} dest=/tmp/conditional.txt" --
check
```

System Log Analysis and Troubleshooting

```
# Ansible execution logs
sudo tail -f /var/log/messages | grep ansible
journalctl -f | grep ansible
journalctl -u ansible-navigator --since "1 hour ago"
journalctl -u ssh --since "10 minutes ago"
sudo tail -f /var/log/secure | grep ansible

# SSH connection debugging
ssh -vvv ansible@hostname                      # Maximum SSH verbosity
ssh -o StrictHostKeyChecking=no ansible@hostname
ssh -o ConnectTimeout=10 ansible@hostname
ssh -o BatchMode=yes ansible@hostname          # Non-interactive mode
ssh -F ~/.ssh/config hostname
ssh -i ~/.ssh/specific_key ansible@hostname

# System authentication logs
sudo tail -f /var/log/secure                   # Authentication events
sudo grep ansible /var/log/secure
sudo journalctl -u sshd --since "1 hour ago"
sudo ausearch -m USER_AUTH --start today
lastlog | grep ansible
last | grep ansible

# Network troubleshooting
ansible all -m command -a "ss -tuln | grep :22"
ansible all -m command -a "iptables -L -n" --become
ansible all -m command -a "firewall-cmd --list-all" --become
ping -c 3 hostname
traceroute hostname
nslookup hostname

# System resource monitoring
ansible all -m command -a "top -n 1 -b | head -20"
ansible all -m command -a "iostat -x 1 3"
ansible all -m command -a "vmstat 1 3"
ansible all -m command -a "free -m"
ansible all -m command -a "df -h"

# Process and service debugging
ansible all -m command -a "ps aux | grep python"
ansible all -m command -a "systemctl status sshd --no-pager"
ansible all -m command -a "systemctl --failed" --become
ansible all -m command -a "dmesg | tail -20" --become

# File system and permissions troubleshooting
ansible all -m command -a "ls -la /home/ansible/.ssh/"
ansible all -m stat -a "path=/home/ansible/.ssh/authorized_keys"
ansible all -m command -a "getfacl /path/to/file" --become
ansible all -m command -a "semanage fcontext -l | grep ansible" --become
ansible all -m command -a "ls -Z /home/ansible/" --become

# Performance analysis
time ansible all -m ping                       # Connection timing
time ansible all -m setup                      # Fact gathering timing
```

```
ansible all -m command -a "time uptime"
strace -e trace=network ansible all -m ping 2>&1 | grep -E '(connect|send|recv)'

# Ansible configuration debugging
ansible-config dump | grep -E '(HOST_KEY_CHECKING|INVENTORY|TIMEOUT)'
echo $ANSIBLE_CONFIG
echo $ANSIBLE_INVENTORY
echo $ANSIBLE_HOST_KEY_CHECKING

# Advanced debugging techniques
strace -o ansible.trace ansible all -m ping
ltrace -o ansible.ltrace ansible all -m ping
ansible all -m setup | python3 -m json.tool > facts.json
ansible-inventory --list | jq '.webservers.hosts[]'

# Log aggregation and analysis
ansible all -m command -a "journalctl --since '1 hour ago' --no-pager" --become | tee all_logs.t
xt
grep -E '(ERROR|FAILED|WARNING)' ansible_debug.log
awk '/TASK.*FAILED/ {print; getline; print}' ansible_debug.log
sed -n '/PLAY RECAP/,/EOF/p' ansible_debug.log

# Remote system analysis
ansible all -m command -a "uptime && who && last | head -5"
ansible all -m shell -a "cat /proc/version && cat /etc/redhat-release"
ansible all -m command -a "uname -a"
ansible all -m setup -a "filter=ansible_kernel"
```

10. Documentation and Help Systems

Comprehensive Documentation Access (Exam Critical)

```
# Module documentation (complete reference)
ansible-doc module_name                  # Full module documentation
ansible-doc -s module_name              # Synopsis only (quick reference)
ansible-doc -l                          # List all available modules
ansible-doc -l | grep keyword           # Search for modules
ansible-doc -l | wc -l                  # Count available modules
ansible-doc -l | head -20               # First 20 modules
ansible-doc -l | sort | grep -E '^(ansible\.builtin|community\.general)'

# FQCN module documentation
ansible-doc ansible.builtin.dnf         # Built-in modules
ansible-doc community.general.firewalld # Community modules
ansible-doc ansible.posix.mount         # POSIX collection
ansible-doc containers.podman.podman_container
ansible-doc redhat.rhel_system_roles.selinux

# Plugin documentation by type
ansible-doc -t connection -l            # Connection plugins
ansible-doc -t lookup -l                # Lookup plugins
ansible-doc -t filter -l                # Filter plugins
ansible-doc -t test -l                  # Test plugins
ansible-doc -t callback -l              # Callback plugins
ansible-doc -t cache -l                 # Cache plugins
ansible-doc -t vars -l                  # Vars plugins
ansible-doc -t inventory -l             # Inventory plugins

# Specific plugin documentation
ansible-doc -t lookup file              # File lookup plugin
ansible-doc -t lookup env               # Environment variable lookup
ansible-doc -t filter default           # Default filter
ansible-doc -t test defined             # Defined test
ansible-doc -t connection ssh           # SSH connection plugin

# Search and discovery patterns
ansible-doc -l | grep -i package        # Find package-related modules
ansible-doc -l | grep -i user           # Find user-related modules
ansible-doc -l | grep -i service        # Find service-related modules
ansible-doc -l | grep -i file           # Find file-related modules
ansible-doc -l | grep -i network        # Find network-related modules
ansible-doc -l | grep -i security       # Find security-related modules

# Documentation with examples extraction
ansible-doc dnf | grep -A 20 "EXAMPLES:"
ansible-doc systemd | grep -A 30 "EXAMPLES:"
ansible-doc copy | grep -A 15 "EXAMPLES:"
ansible-doc template | grep -A 25 "EXAMPLES:"

# Module parameter reference
ansible-doc dnf | grep -A 50 "OPTIONS:"
ansible-doc systemd | grep -A 40 "OPTIONS:"
ansible-doc user | grep -A 60 "OPTIONS:"
ansible-doc file | grep -A 35 "OPTIONS:"
```

Navigator Interface and Help

```
# Navigator command help
ansible-navigator --help                # General help
ansible-navigator run --help           # Playbook execution help
ansible-navigator config --help        # Configuration help
ansible-navigator collections --help   # Collections help
ansible-navigator doc --help           # Documentation help
ansible-navigator images --help        # Execution environment help
ansible-navigator inventory --help     # Inventory help

# Navigator interactive TUI commands
# Inside navigator interface:
:help                                  # Show comprehensive help
:doc module_name                       # View module documentation
:doc -l                               # List all modules
:collections                          # Browse collections interactively
:inventory                            # View inventory structure
:images                               # List execution environments
:config                               # View configuration
:q or :quit                          # Exit navigator
:back or ESC                         # Go back one level
:0 or :stdout                        # Switch to stdout mode
:1 or :interactive                   # Switch to interactive mode

# Navigation shortcuts in TUI
# Arrow keys or hjkl                   # Navigate lists
# Enter                                # Select item
# Tab                                  # Auto-complete
# / or ?                              # Search within content
# Page Up/Down or Ctrl+B/F            # Page through content
# Home/End                            # Go to beginning/end
# Ctrl+C or :q                        # Exit

# Navigator with different output modes
ansible-navigator --help-config        # Configuration options help
ansible-navigator --version            # Version information
ansible-navigator --help-all           # Complete help reference
```

System Information and Fact Gathering

```
# Comprehensive fact gathering
ansible all -m setup                    # Gather all system facts
ansible all -m setup --tree /tmp/facts  # Save facts to files
ansible all -m setup -a "gather_subset=all" # Explicit all facts
ansible all -m setup -a "gather_timeout=30" # Custom timeout

# Filtered fact gathering (performance optimization)
ansible hostname -m setup -a "filter=ansible_distribution*"
ansible hostname -m setup -a "filter=ansible_memory*"
ansible hostname -m setup -a "filter=ansible_processor*"
ansible hostname -m setup -a "filter=ansible_mounts"
ansible hostname -m setup -a "filter=ansible_interfaces"
ansible hostname -m setup -a "filter=ansible_default_ipv4"
ansible hostname -m setup -a "filter=ansible_all_ipv4_addresses"
ansible hostname -m setup -a "filter=ansible_hostname"

# Selective fact gathering subsets
ansible all -m setup -a "gather_subset=network"
ansible all -m setup -a "gather_subset=hardware"
ansible all -m setup -a "gather_subset=virtual"
ansible all -m setup -a "gather_subset=ohai,facter"
ansible all -m setup -a "gather_subset=!all"
ansible all -m setup -a "gather_subset=!ohai,!facter"
ansible all -m setup -a "gather_subset=network,hardware"

# Network-specific information
ansible all -m setup -a "filter=ansible_default_ipv4"
ansible all -m setup -a "filter=ansible_all_ipv4_addresses"
ansible all -m setup -a "filter=ansible_dns"
ansible all -m setup -a "filter=ansible_domain"
ansible all -m setup -a "filter=ansible_interfaces"
ansible all -m setup -a "filter=ansible_route*"

# Hardware and system information
ansible all -m setup -a "filter=ansible_processor*"
ansible all -m setup -a "filter=ansible_memtotal_mb"
ansible all -m setup -a "filter=ansible_swaptotal_mb"
ansible all -m setup -a "filter=ansible_devices"
ansible all -m setup -a "filter=ansible_architecture"
ansible all -m setup -a "filter=ansible_distribution*"
ansible all -m setup -a "filter=ansible_kernel"
ansible all -m setup -a "filter=ansible_os_family"
ansible all -m setup -a "filter=ansible_pkg_mgr"
ansible all -m setup -a "filter=ansible_service_mgr"

# Custom facts and performance
ansible all -m setup -a "fact_path=/etc/ansible/facts.d"
ansible all -m setup -a "filter=ansible_local"
time ansible all -m setup > /dev/null
ansible all -m setup --tree /tmp/facts && cat /tmp/facts/hostname | jq '.ansible_facts.keys[]'
```

Quick Command Combinations for Exam

Rapid Testing Sequence

```
# 1. Test connectivity
ansible all -m ping

# 2. Check syntax
ansible-navigator run site.yml --syntax-check

# 3. Dry run
ansible-navigator run site.yml --check

# 4. Execute with verbosity
ansible-navigator run site.yml --mode stdout -v

# 5. Verify results
ansible all -m setup -a "filter=ansible_service_mgr"
```

Emergency Documentation Lookup

```
# Quick module search
ansible-doc -l | grep -i package
ansible-doc -l | grep -i user
ansible-doc -l | grep -i file

# Module examples
ansible-doc -s dnf
ansible-doc -s user
ansible-doc -s systemd
```

Exam Success Strategies

Essential Command Patterns for Exam Day

```
# The "Big 4" - Master these patterns for 80% of exam tasks:

# 1. CONNECTIVITY TEST (always start here)
ansible all -m ping

# 2. SYNTAX VALIDATION (before every execution)
ansible-navigator run playbook.yml --syntax-check

# 3. DRY RUN (verify changes before applying)
ansible-navigator run playbook.yml --check --diff

# 4. EXECUTE WITH LOGGING (run and capture output)
ansible-navigator run playbook.yml --mode stdout -v | tee execution.log
```

Time-Saving Command Combinations

```
# Quick validation sequence (use for every playbook):
ansible-navigator run site.yml --syntax-check && \
ansible-navigator run site.yml --check && \
ansible-navigator run site.yml --mode stdout

# Emergency troubleshooting sequence:
ansible all -m ping -vvv
ansible-config dump | grep -E '(INVENTORY|HOST_KEY|REMOTE_USER)'
ansible all -m setup -a "filter=ansible_distribution"

# Documentation lookup shortcuts:
ansible-doc -l | grep -i KEYWORD    # Find modules quickly
ansible-doc -s MODULE_NAME          # Get syntax fast
ansible-doc MODULE_NAME | grep -A 10 "EXAMPLES:"
```

Critical Success Factors

1. **Master** `ansible-doc` - Your primary resource during the exam
2. **Always test connectivity first** - `ansible all -m ping`
3. **Validate before executing** - `--syntax-check` and `--check`
4. **Use verbosity for debugging** - `-v`, `-vv`, `-vvv` progressively
5. **Leverage navigator TUI** - Interactive mode for complex debugging
6. **Practice command patterns** - Speed comes from muscle memory
7. **Know your collections** - FQCN usage is essential
8. **Vault operations** - Practice all vault commands until automatic

Remember: The exam environment provides `ansible-doc` offline documentation. Use it extensively!

3.5 RHCE Acronyms & Glossary

Comprehensive RHCE Terminology Reference

Complete glossary covering all RHCE exam terminology, concepts, and technical vocabulary for thorough exam preparation

Source Integration: Terminology synthesized from:

- Sander van Vugt's RHCE Guide (16 chapters) - RHEL 8/9 focused
- Jeff Geerling's Ansible for DevOps (15 chapters) - Modern practices
- Red Hat official documentation and certification materials

Essential RHCE Acronyms

Core Certification Acronyms

Acronym	Full Term	Context	Exam Importance
RHCE	Red Hat Certified Engineer	EX294 certification exam	Critical
EX294	Exam 294	Current RHCE exam code for RHEL 9	Essential
RHCSA	Red Hat Certified System Administrator	Prerequisite certification	Required
RHEL	Red Hat Enterprise Linux	Target operating system	Essential
FQCN	Fully Qualified Collection Name	Module naming: `ansible.builtin.dnf`	Critical

Ansible Technology Acronyms

Acronym	Full Term	Technical Context	Usage
ACN	Automation Content Navigator	Primary exam execution tool	Daily use
EE	Execution Environment	Container-based Ansible runtime	Important
AAP	Ansible Automation Platform	Red Hat commercial offering	Context
AWX	Ansible Web eXecution	Open source automation platform	Background
TUI	Text User Interface	Navigator's interactive mode	Essential
CLI	Command Line Interface	Traditional execution mode	Daily use

System and Network Acronyms

Acronym	Full Term	System Context	Exam Use
SSH	Secure Shell	Remote access protocol	Critical
YAML	YAML Ain't Markup Language	Playbook format	Essential
JSON	JavaScript Object Notation	Data exchange format	Important
HTTP/HTTPS	HyperText Transfer Protocol	Web services	Modules
DNS	Domain Name System	Name resolution	Networking
NTP	Network Time Protocol	Time sync	Services
NFS	Network File System	Shared storage	Automation
LVM	Logical Volume Manager	Storage management	RHCSA tasks
SELinux	Security Enhanced Linux	Access control	Security
ACL	Access Control List	File permissions	Security

Comprehensive Terminology by Category

A - Ansible Fundamentals

- **Ad-hoc Commands**: Single-task execution without playbooks using `ansible` command
- **Ansible**: Open source IT automation platform for configuration management and deployment
- **Ansible Configuration File**: `ansible.cfg` controlling Ansible behavior and settings
- **Ansible Galaxy**: Official repository for sharing Ansible roles and collections
- **Ansible Vault**: Built-in encryption tool for protecting sensitive data in playbooks
- **Automation Content Navigator**: Modern Ansible execution tool replacing traditional ansible-playbook
- **Automation Controller**: Web-based interface for Ansible (formerly Tower)
- **Available**: Package state indicating software should be installed but not necessarily latest

B - Basic Operations

- **Become**: Privilege escalation mechanism replacing traditional sudo in Ansible
- **Block**: Grouping construct for tasks allowing structured error handling
- **Boolean**: True/false variable type used extensively in conditionals and configuration
- **Built-in Collection**: Core Ansible modules residing in `ansible.builtin` namespace
- **Byte Code**: Compiled Python code executed on managed nodes during task execution

C - Collections and Control

- **Cache**: Temporary storage mechanism for facts and data to improve performance
- **Callback Plugin**: Extension modifying how Ansible displays output and logs information
- **Check Mode**: Dry run execution showing potential changes without applying them

- **Collection**: Packaged Ansible content including modules, plugins, roles, and documentation
- **Community Collection**: Third-party Ansible content developed by community contributors
- **Conditional**: Logic construct controlling when tasks execute using `when` statements
- **Configuration Management**: Practice of maintaining consistent system state through automation
- **Connection Plugin**: Method defining how Ansible communicates with managed nodes
- **Control Node**: System where Ansible is installed and from which playbooks execute
- **Controller**: Centralized management system for Ansible automation (AWX/Tower)

D - Data and Deployment

- **Daemon**: Background system service typically managed through systemd
- **Debug Module**: Ansible module for printing variables, messages, and troubleshooting information
- **Declarative**: Configuration approach describing desired end state rather than steps
- **Default Variable**: Variable with fallback value used when no explicit value provided
- **Delegation**: Technique for running tasks on different hosts than inventory targets
- **Dictionary**: Key-value data structure fundamental to YAML and Python programming
- **Diff Mode**: Feature showing differences between current and desired system state
- **Dynamic Inventory**: Automatically generated inventory from external sources or scripts

E - Execution and Environment

- **Execution Environment**: Container image containing Ansible and required dependencies
- **Encryption**: Data protection mechanism using ansible-vault or external tools
- **Error Handling**: Managing task failures through blocks, rescue, and always constructs
- **Escalation**: Process of increasing privileges using become/sudo for task execution
- **Execution Policy**: Rules and constraints controlling how and where automation runs

F - Facts and Files

- **Facts**: System information automatically gathered by Ansible setup module
- **Fact Caching**: Performance optimization storing gathered facts for reuse across runs
- **Failed State**: Condition indicating task execution encountered unrecoverable errors
- **File Module**: Ansible module managing files, directories, links, and permissions
- **Filter**: Jinja2 function for transforming and manipulating variables and data
- **Fork**: Parallel execution process controlled by Ansible's `forks` configuration setting
- **FQCN**: Fully Qualified Collection Name format required for modern module usage
- **Function**: Reusable code construct including filters, plugins, and custom modules

G - Groups and Galaxy

- **Galaxy**: Ansible's official community repository for sharing collections and roles
- **Gather Facts**: Automated process collecting system information from managed nodes
- **Group**: Logical collection of hosts in inventory for targeted automation
- **Group Variables**: Variables automatically applied to all hosts within inventory groups
- **Guard Condition**: Logical check preventing task execution under specific circumstances

H - Handlers and Hosts

- **Handler**: Special task triggered by notifications, typically for service restarts
- **Host**: Individual system targeted by Ansible automation and configuration management
- **Host Variables**: Variables specific to individual inventory hosts
- **Host Key Checking**: SSH security feature verifying remote host identity
- **Hostvars**: Magic variable containing all variables for all hosts in inventory

I - Inventory and Idempotency

- **Idempotency**: Critical property ensuring repeated execution produces identical results
- **Imperative**: Configuration approach specifying step-by-step procedural actions
- **Import**: Static inclusion of tasks, roles, or playbooks processed at parse time
- **Include**: Dynamic inclusion of tasks, roles, or playbooks processed at runtime
- **Infrastructure as Code**: Philosophy of managing infrastructure through version-controlled code
- **Inventory**: File or script defining hosts, groups, and variables for automation
- **Inventory Plugin**: Extension mechanism for generating dynamic inventory from external sources
- **Item**: Individual element referenced during loop iteration in tasks

J - Jinja2 and JSON

- **Jinja2**: Powerful templating engine used throughout Ansible for dynamic content generation
- **JSON**: Structured data format used for complex variables and API interactions
- **Jump Host**: Intermediate system used for reaching isolated or secured managed nodes

K - Keys and Kernel

- **Kernel Module**: Loadable kernel extensions managed through automation for system functionality
- **Key Distribution**: Process of copying SSH public keys to managed nodes for authentication
- **Key Pair**: Combined public/private SSH keys enabling secure passwordless authentication

L - Loops and Logic

- **Library**: Collection of reusable modules, plugins, or code components
- **Limit**: Mechanism restricting playbook execution to subset of inventory hosts
- **Lineinfile Module**: Ansible module for managing individual lines within text files
- **List**: Ordered collection of items fundamental to YAML data structures
- **Local Action**: Task executed on control node instead of managed nodes
- **Loop**: Iteration mechanism enabling tasks to process multiple data items
- **Lookup Plugin**: Extension retrieving external data during playbook execution

M - Modules and Management

- **Magic Variable**: Special Ansible-provided variable containing system and runtime information
- **Managed Node**: Target system configured and controlled by Ansible automation
- **Module**: Fundamental unit of Ansible automation implementing specific functionality
- **Module Path**: Directory location where Ansible searches for custom module implementations
- **Mount Point**: File system attachment location managed through storage automation

N - Networking and Namespaces

- **Namespace**: Organizational structure for collections using dot notation (e.g., `community.general`)
- **Network Module**: Specialized Ansible modules for network device configuration and management
- **Notification**: Mechanism triggering handler execution after task state changes
- **No Log**: Security feature preventing sensitive information from appearing in execution logs

O - Operations and Output

- **Operation**: Individual automation action performed by Ansible modules
- **Output**: Results and information returned by task and module execution
- **Override**: Process of replacing default configuration values with custom settings

P - Playbooks and Plugins

- **Package Manager**: System utility for software installation and management (DNF/YUM)
- **Parameter**: Module option or argument controlling specific task behavior
- **Parsing**: Process of reading, interpreting, and validating YAML syntax and structure
- **Play**: Single automation scenario within playbook targeting specific host groups
- **Playbook**: YAML file containing organized automation tasks, variables, and logic
- **Plugin**: Extension mechanism adding specialized functionality to Ansible core
- **Precedence**: Hierarchical order determining which variable values take priority

- **Present State**: Module parameter indicating desired resource should exist on system
- **Private Key**: Secret portion of SSH key pair used for authentication
- **Privilege Escalation**: Mechanism for gaining elevated system permissions
- **Public Key**: Shareable portion of SSH key pair for remote authentication

Q - Queries and Queues

- **Query**: Request for information from system, API, or external data source
- **Queue**: Ordered sequence of tasks awaiting execution in automation workflow

R - Roles and Registry

- **Registry**: Repository storing container images or automation content packages
- **Remote User**: Account used for SSH connections to managed nodes
- **Repository**: Storage location for packages, source code, or automation content
- **Requirements**: Dependencies specification for collections or roles
- **Rescue**: Error handling block executed when main tasks encounter failures
- **Role**: Reusable, organized collection of automation tasks and supporting resources
- **Run**: Single execution instance of playbook or automation content

S - Security and Services

- **Secrets Management**: Secure handling of passwords, keys, and sensitive configuration data
- **Service**: System daemon managed through systemd or similar service management
- **Service Facts**: Automatically gathered information about system services and their states
- **Setup Module**: Built-in Ansible module responsible for gathering system facts
- **Shell Module**: Ansible module executing shell commands with full shell features
- **Skip Tags**: Mechanism for excluding specific tagged tasks during execution
- **Sudo**: Traditional privilege escalation command superseded by become in modern Ansible
- **Syntax Check**: Validation process ensuring YAML structure and Ansible logic correctness

T - Tasks and Templates

- **Tag**: Categorical label for organizing and selectively executing specific tasks
- **Task**: Individual automation step within plays representing atomic work units
- **Template**: Dynamic file generation mechanism using Jinja2 templating engine
- **Test**: Jinja2 function for evaluating conditions and boolean expressions
- **Timeout**: Maximum allowed duration for task or operation completion
- **Tower**: Former name for Red Hat Ansible Automation Controller platform
- **Transform**: Data manipulation process using filters or custom processing logic

U - Users and Updates

- **Unarchive**: Process of extracting and deploying compressed archive files
- **Update**: Modification of existing configuration or installation of newer software versions
- **URI Module**: Ansible module for HTTP/REST API interactions and web service communication
- **User Module**: Ansible module for comprehensive system user account management

V - Variables and Vaults

- **Variable**: Named storage mechanism for data used throughout automation processes
- **Variable File**: YAML file containing organized variable definitions and values
- **Vault**: Ansible's integrated encryption system for protecting sensitive automation data
- **Vault ID**: Unique identifier for specific vault encryption keys and passwords
- **Vault Password**: Cryptographic key used for encrypting and decrypting vault content
- **Verbosity**: Configurable level of detail in Ansible output and logging
- **Version Control**: System for tracking and managing changes to automation code

W - Workflow and When

- **When Condition**: Conditional logic construct controlling selective task execution
- **Workflow**: Organized sequence of automation tasks and decision points
- **Working Directory**: File system location where commands and operations execute

X - eXecution and eXtensions

- **Execution**: Process of running playbooks, tasks, or automation content
- **Extension**: Plugin or module adding specialized functionality to Ansible core

Y - YAML and Yum

- **YAML**: Human-readable data serialization language used for all Ansible configuration
- **YUM**: Legacy package manager predecessor to DNF on Red Hat systems

Z - Zones

- **Zone**: Security or network context used in firewall and SELinux configuration

---

Module Categories and Classifications

System Administration Modules

Category	Purpose	Key Modules	Exam Weight
User Management	Account operations	`user`, `group`, `authorized_key`	High
Service Control	System services	`systemd`, `service`	High
Package Management	Software installation	`dnf`, `package`, `rpm_key`	High
File Operations	File manipulation	`copy`, `template`, `file`, `lineinfile`	High
Storage Management	Disk and filesystem	`parted`, `lvg`, `lvol`, `mount`, `filesystem`	Medium
Network Configuration	Network setup	`firewalld`, `nmcli`, `uri`	Medium
Security Controls	Access and permissions	`seboolean`, `selinux`, `sefcontext`	Medium

Advanced Module Categories

Category	Purpose	Collections	Use Cases
Cloud Integration	Cloud platform management	`amazon.aws`, `azure.azcollection`	Infrastructure
Container Management	Container operations	`containers.podman`, `kubernetes.core`	Modern apps
Database Operations	Database management	`community.mysql`, `community.postgresql`	Data services
Web Services	Web server configuration	`community.general`, `ansible.builtin`	Web hosting
Monitoring	System monitoring	`community.general`, `ansible.builtin`	Observability

Exam-Specific Concepts

Red Hat Methodology and Best Practices

- **Idempotency**: Fundamental principle ensuring automation safety and predictability
- **Declarative Configuration**: Approach focusing on desired end state rather than procedures
- **Infrastructure as Code**: Methodology treating infrastructure as version-controlled software
- **Configuration Drift**: Deviation between intended configuration and actual system state
- **Convergence**: Process of bringing systems to desired configuration state
- **Immutable Infrastructure**: Strategy of replacing rather than modifying existing systems

Collection Namespaces and Usage

Namespace	Purpose	Primary Modules	Exam Relevance
`ansible.builtin`	Core functionality	`dnf`, `systemd`, `copy`, `template`	Critical
`ansible.posix`	POSIX systems	`firewalld`, `mount`, `seboolean`	High
`community.general`	Extended features	`parted`, `lvg`, `lvol`	High
`community.crypto`	Cryptographic operations	`openssl_certificate`	Medium
`containers.podman`	Container management	`podman_container`	Medium

Variable Precedence Hierarchy

1. **Command line** (`-e` flag) - Highest priority
2. **Task vars** - Task-specific variables
3. **Block vars** - Block-scoped variables
4. **Role and include vars** - Role-defined variables
5. **Set_facts / registered vars** - Runtime-defined variables
6. **Play vars_files** - External variable files
7. **Play vars_prompt** - Interactive variables
8. **Play vars** - Play-defined variables
9. **Host facts** - System-discovered variables
10. **Host vars** (inventory) - Host-specific variables
11. **Group vars** (inventory) - Group-specific variables
12. **Group vars** (/all) - All-group variables
13. **Group vars** (/*) - Wildcard group variables
14. **Role defaults** - Role default variables
15. **Command line inventory vars** - Inventory variables
16. **Default vars** - Lowest priority defaults

Common Exam Pitfalls and Solutions

FQCN Requirements and Errors

Incorrect Usage	Correct Usage	Error Prevention
`dnf:`	`ansible.builtin.dnf:`	Always use FQCN
`systemd:`	`ansible.builtin.systemd:`	Check collection
`firewalld:`	`ansible.posix.firewalld:`	Install collections
`parted:`	`community.general.parted:`	Verify availability

Navigator vs Traditional Commands

Old Method	New Method	Exam Requirement
`ansible-playbook`	`ansible-navigator run`	Use navigator
`ansible-doc`	`ansible-navigator doc`	Either acceptable
`ansible-inventory`	`ansible-navigator inventory`	Either acceptable

Collection Management Issues

Problem	Solution	Prevention
Missing collections	Install required collections	Check requirements
Version conflicts	Specify compatible versions	Pin versions
FQCN errors	Use fully qualified names	Always use namespace

Study Strategy and Priority Framework

Critical Priority Terms (Master First)

- **FQCN usage and namespace understanding**
- **Variable precedence and scoping rules**
- **Module parameters for core modules**
- **Vault operations and encryption methods**
- **Navigator execution modes and commands**
- **Task control: conditionals, loops, error handling**

High Priority Terms (Essential Knowledge)

- **Role structure and development patterns**
- **Template syntax and Jinja2 filters**
- **Collection installation and management**
- **Fact gathering and system information access**
- **SSH configuration and authentication setup**

Medium Priority Terms (Supporting Knowledge)

- **Advanced automation patterns and strategies**
- **Performance optimization techniques**
- **Troubleshooting methodologies and approaches**
- **Integration with external systems and APIs**

Study Methodology

1. **Terminology Drills**: Regular review of key terms and definitions
2. **Practical Application**: Hands-on use of concepts in lab environments
3. **Cross-Reference Learning**: Connect terminology to actual implementation
4. **Progressive Complexity**: Build from basic concepts to advanced patterns
5. **Exam Simulation**: Practice using terminology in time-pressured scenarios

Quick Reference for Exam Day

Essential Command Patterns

```
# Documentation lookup (your lifeline)
ansible-doc module_name
ansible-doc -s module_name
ansible-doc -l | grep keyword

# Validation sequence
ansible all -m ping
ansible-navigator run --syntax-check
ansible-navigator run --check --diff
```

Must-Know Module Categories

- **System**: `systemd`, `user`, `group`, `cron`
- **Package**: `dnf`, `package`, `rpm_key`
- **File**: `copy`, `template`, `file`, `lineinfile`
- **Storage**: `parted`, `lvg`, `lvol`, `mount`
- **Network**: `firewalld`, `uri`, `get_url`
- **Security**: `seboolean`, `selinux`, `authorized_key`

Critical Success Factors

- **Always use FQCN**: Never use short module names
- **Test connectivity first**: Start every session with `ansible all -m ping`
- **Know ansible-doc**: Your primary reference tool during exam
- **Master variable precedence**: Understand all 16 levels of priority
- **Practice vault operations**: Encryption is mandatory for sensitive data

---

Remember: Terminology mastery accelerates problem-solving and reduces cognitive load during high-pressure exam situations. Focus on understanding concepts, not just memorizing definitions.

3.6 RHCE Knowledge Gaps Checklist - For Experienced Users

Self-Assessment for Production Ansible Users

Use this checklist to identify what you need to focus on for exam success

---

How to Use This Checklist

Rating Scale:

- **Confident**: I can do this without reference materials
- **Uncertain**: I know the concept but need to verify syntax/approach
- **Gap**: I need to learn/practice this area

Focus Strategy:

- **Items**: Quick review only
- **Items**: Practice and verify, but don't over-study
- **Items**: Intensive study and lab practice required

---

1. Automation Content Navigator

Task	Rating	Notes
Run playbooks with `ansible-navigator run`		vs ansible-playbook
Navigate the TUI interface effectively		Interactive mode
Use stdout mode when appropriate		--mode stdout
Browse collections within navigator		:collections command
View module documentation in navigator		:doc module_name
Explore inventory using navigator		:inventory command
Debug failed tasks in TUI		Navigate to task details
Understand execution environments		Container vs host execution
Switch between interactive/stdout modes		Mode selection
Use navigator for syntax checking		--syntax-check

Quick Self-Test: Can you run a playbook, debug a failure, and view module docs all within navigator?

---

2. Content Collections & FQCN

Task	Rating	Notes
Install collections with ansible-galaxy		collection install command
Use FQCN format consistently		namespace.collection.module
Know critical exam collections		builtin, posix, community.general
Write requirements.yml for collections		Collection dependencies
List installed collections		ansible-galaxy collection list
Browse collection modules		Find modules in collections
Use community.general storage modules		parted, lvg, lvol
Use ansible.posix system modules		firewalld, mount, seboolean
Troubleshoot "module not found" errors		Missing collection diagnosis
Configure collections_path		ansible.cfg setting

Quick Self-Test: Can you install a collection and immediately use its modules with FQCN?

---

3. Control Node Setup & Configuration

Task	Rating	Notes
Install Ansible on RHEL 9		dnf install ansible-core
Create and configure ansible.cfg		All critical settings
Set up inventory in INI format		Groups and host variables
Set up inventory in YAML format		Nested group structure
Configure privilege escalation		become settings
Set default remote user		remote_user setting
Configure host key checking		Security vs convenience
Set up roles and collections paths		Custom paths
Test ansible.cfg precedence		Multiple config files
Validate inventory syntax		ansible-inventory commands

Quick Self-Test: Can you set up a complete Ansible environment from scratch in 15 minutes?

---

4. Managed Node Configuration & SSH

Task	Rating	Notes
Generate SSH key pairs		ssh-keygen with options
Distribute keys to managed nodes		ssh-copy-id
Set up passwordless authentication		End-to-end process
Configure sudo/become on managed nodes		NOPASSWD setup
Automate SSH key distribution		Using Ansible modules
Test connectivity with ansible ping		Basic connectivity
Troubleshoot SSH connection issues		Debug authentication
Set up ansible user account		Dedicated automation user
Configure privilege escalation		become_method, become_user
Handle SSH host key verification		StrictHostKeyChecking

Quick Self-Test: Can you set up SSH authentication to a new system and test it in under 10 minutes?

---

5. RHCSA Task Automation (Critical Exam Area)

Package Management

Task	Rating	Notes
Install packages with ansible.builtin.dnf		Single and multiple packages
Remove packages safely		state=absent
Install from URL/file		RPM files
Manage package groups		@group syntax
Configure repositories		yum_repository module

Service Management

Task	Rating	Notes
Start/stop services with systemd module		state parameter
Enable/disable services		enabled parameter
Restart services conditionally		Handlers and notify
Reload systemd daemon		daemon_reload parameter
Check service status		Using setup module

Storage Management

Task	Rating	Notes
Create partitions with parted		community.general.parted
Create volume groups		community.general.lvg
Create logical volumes		community.general.lvol
Create filesystems		community.general.filesystem
Mount filesystems		ansible.posix.mount
Manage /etc/fstab		Persistent mounts

User Management

Task	Rating	Notes
Create users with user module		All common parameters
Create groups with group module		System vs regular groups
Set passwords securely		password_hash filter
Manage SSH keys for users		authorized_key module
Set user properties		shell, home, groups

File Management

Task	Rating	Notes
Copy files with copy module		Local to remote
Deploy templates		Jinja2 templating
Modify file content		lineinfile, replace
Set file permissions		file module attributes
Create directories		Directory hierarchies

Firewall Management

Task	Rating	Notes
Configure firewalld services		ansible.posix.firewalld
Open specific ports		Port ranges and protocols
Manage firewall zones		Zone assignments
Make changes permanent		permanent parameter
Apply changes immediately		immediate parameter

SELinux Management

Task	Rating	Notes
Set SELinux booleans		ansible.posix.seboolean
Configure file contexts		sefcontext module
Apply context changes		restorecon command
Check SELinux status		Using facts
Handle SELinux troubleshooting		Common issues

Quick Self-Test: Can you automate a complete LAMP stack setup including storage, users, services, and security?

---

6. Shell Script Conversion (Exam Requirement)

Task	Rating	Notes
Analyze shell script logic		Understand flow
Convert conditional statements		if/then to when
Convert loops to Ansible loops		for to loop/with_items
Handle command execution		command vs shell
Manage script variables		Script vars to Ansible vars
Convert error handling		failed_when, ignore_errors
Preserve script functionality		Idempotency considerations
Test converted playbooks		Verify same results

Quick Self-Test: Given a 20-line shell script, can you convert it to an equivalent playbook in 30 minutes?

---

7. Playbook Development (Advanced Patterns)

Task	Rating	Notes
Use conditionals with when		Complex logic
Implement loops effectively		loop, with_items variants
Handle task failures gracefully		failed_when, ignore_errors
Use blocks for error handling		block/rescue/always
Implement handlers properly		Notification patterns
Use register for task results		Variable capture
Delegate tasks appropriately		delegate_to, run_once
Use tags for selective execution		Tag strategies
Implement async task execution		Long-running tasks
Debug variables effectively		debug module usage

Quick Self-Test: Can you write a complex playbook with error handling, loops, and conditionals?

---

8. Ansible Vault (Security)

Task	Rating	Notes
Create encrypted files		ansible-vault create
Edit encrypted files		ansible-vault edit
Encrypt existing files		ansible-vault encrypt
Decrypt files when needed		ansible-vault decrypt
Change vault passwords		ansible-vault rekey
Use vault in playbooks		--ask-vault-pass
Set up vault password files		--vault-password-file
Encrypt single variables		encrypt_string
Use multiple vault IDs		--vault-id
Integrate vault with CI/CD		Automated workflows

Quick Self-Test: Can you set up and use encrypted variables in a playbook execution?

---

9. Exam Environment Constraints

Task	Rating	Notes
Work entirely offline		No internet access
Use ansible-doc effectively		Primary documentation
Navigate documentation quickly		Fast lookups
Manage time effectively		4-hour constraint
Debug without external help		Self-reliant troubleshooting
Use check mode for validation		Dry run testing
Test incrementally		Single task validation
Work with provided inventory		Predefined hosts/groups
Handle exam system constraints		Limited resources
Maintain focus under pressure		Stress management

Quick Self-Test: Can you complete a complex automation task without any external references?

Gap Analysis Summary

Count Your Ratings

- **Confident**: ___/90 items
- **Uncertain**: ___/90 items
- **Gap**: ___/90 items

Study Priority Calculation

- **70-90% Confident**: Light review, focus on uncertain items
- **50-69% Confident**: Moderate study needed, practice labs
- **Below 50% Confident**: Intensive study required

Recommended Study Path Based on Gaps

High Gap Count (>30 items):

- Start with Automation Content Navigator
- Master Content Collections next
- Focus heavily on RHCSA task automation
- Practice exam scenarios extensively

Medium Gap Count (15-30 items):

- Target specific weak areas
- Practice time management
- Focus on exam-specific differences
- Do realistic practice scenarios

Low Gap Count (<15 items):

- Quick review of uncertain items
- Practice exam timing
- Focus on exam environment adaptation
- Take practice exams

Action Plan Template

Based on your gap analysis, create a focused study plan:

Week 1: Foundation Gaps

Focus Areas: __
***Lab Time**: ___ hours/day*
***Key Tasks**: __*

Week 2: Advanced Skills

Focus Areas: __
***Lab Time**: ___ hours/day*
***Key Tasks**: __*

Week 3: Exam Simulation

Focus Areas: Time management, exam scenarios
Lab Time: ___ hours/day
Key Tasks: Full practice exams

Week 4: Final Preparation

Focus Areas: Review weak areas, confidence building
Lab Time: ___ hours/day
Key Tasks: Final practice, review quick references

Remember: Your production experience is a major advantage. Focus on the gaps, not on re-learning what you already know!