

Session 3: Ansible Roles, Templates & Error Handling

Advanced Automation Techniques for Beginners

Introduction to Ansible Roles

What Are Ansible Roles?

Roles provide a framework for organizing playbooks into reusable, modular components. They follow the separation of concerns principle and enable collaboration.

Key Benefits

- **Modular Structure:** Break complex automation into manageable pieces
- **Reusability:** Use roles across multiple projects and playbooks
- **Separation of Concerns:** Organize code by function and purpose
- **Collaboration:** Multiple team members can work on different roles
- **Galaxy Integration:** Share and download community-created roles

Benefits of Using Roles

- **Code Organization:** Logical grouping of related tasks, variables, templates
- **Maintainability:** Easier to update and troubleshoot isolated components
- **Scalability:** Add new functionality without modifying existing code
- **Testing:** Test roles independently before integration
- **Documentation:** Self-contained units with clear purpose
- **Version Control:** Track changes to specific roles independently

Role Directory Structure

```
roles/
  common/          # Role name
    tasks/         # Main tasks for the role
      main.yml     # Entry point for tasks
    handlers/       # Event-driven handlers
      main.yml
    templates/      # Jinja2 templates
      config.j2
    files/          # Static files to copy
```

```
script.sh
vars/          # Role-specific variables
  main.yml
defaults/      # Default variables (lowest precedence)
  main.yml
meta/          # Role metadata and dependencies
  main.yml
library/        # Custom modules (optional)
module_utils/   # Supporting code for modules (optional)
```

Understanding Role Components

- **tasks/**: Contains the main list of tasks to be executed
- **handlers/**: Handlers triggered by notify from tasks
- **templates/**: Jinja2 template files for dynamic configs
- **files/**: Static files copied to managed hosts
- **vars/**: Variables specific to the role (high precedence)
- **defaults/**: Default variables that can be easily overridden
- **meta/**: Role metadata including dependencies and Galaxy info
- **library/**: Custom modules specific to the role

Creating Roles with ansible-galaxy

```
# Create a new role skeleton
ansible-galaxy role init role_name

# Create role in specific directory
ansible-galaxy role init --init-path ./roles webserver

# Force creation (overwrite existing)
ansible-galaxy role init --force webserver

# View role structure
tree roles/webserver
```

The ansible-galaxy command creates a standardized role directory structure automatically, saving time and ensuring consistency.

Creating and Using Roles

Step 1: Initialize Role Structure

```
# Navigate to your project directory
cd ~/ansible-project

# Create roles directory
mkdir -p roles

# Initialize new role
ansible-galaxy role init roles/webserver

# Output:
# - Role webserver was created successfully
```

Step 2: Define Role Tasks

```
# roles/webserver/tasks/main.yml
---
- name: Install Nginx
  package:
    name: nginx
    state: present

- name: Start and enable Nginx
  service:
    name: nginx
    state: started
    enabled: yes

- name: Deploy configuration
  template:
    src: nginx.conf.j2
    dest: /etc/nginx/nginx.conf
  notify: restart nginx
```

Step 3: Define Role Variables

```
# roles/webserver/defaults/main.yml
---
nginx_port: 80
nginx_user: www-data
worker_processes: auto
client_max_body_size: 10m

# roles/webserver/vars/main.yml
---
nginx_version_required: "1.18"
log_directory: /var/log/nginx
```

Step 4: Create Templates

```
# roles/webserver/templates/nginx.conf.j2
user {{ nginx_user }};
worker_processes {{ worker_processes }};

http {
    client_max_body_size {{ client_max_body_size }};

    server {
        listen {{ nginx_port }};
        server_name {{ ansible_fqdn }};

        location / {
            root /var/www/html;
            index index.html;
        }
    }
}
```

Step 5: Add Handlers

```
# roles/webserver/handlers/main.yml
---
- name: restart nginx
  service:
    name: nginx
    state: restarted

- name: reload nginx
  service:
    name: nginx
    state: reloaded

- name: validate nginx config
  command: nginx -t
```

Step 6: Define Role Metadata

```
# roles/webserver/meta/main.yml
---
galaxy_info:
  author: Your Name
  description: Nginx web server role
  company: Your Company
  license: MIT
  min_ansible_version: 2.9

  platforms:
    - name: Ubuntu
```

```
versions:
  - focal
  - jammy
- name: CentOS
  versions:
    - 8

galaxy_tags:
  - webserver
  - nginx

dependencies: []
```

Using Roles in Playbooks

```
# site.yml
---
- name: Configure web servers
  hosts: webservers
  become: yes

  roles:
    - webserver

# With variable overrides
- name: Configure web servers
  hosts: webservers
  become: yes

  roles:
    - role: webserver
      vars:
        nginx_port: 8080
        worker_processes: 4
```

Alternative Ways to Use Roles

```
# Using include_role
- name: Configure servers
  hosts: all
  tasks:
    - name: Include webserver role
      include_role:
        name: webserver

# Using import_role (static)
- name: Configure servers
  hosts: all
  tasks:
    - name: Import webserver role
      import_role:
```

```
name: webserver

# With tags
- name: Configure servers
  hosts: all
  roles:
    - role: webserver
      tags: [webserver, nginx]
```

Advanced Jinja2 Templates

Template Filters and Transformations

```
# String filters
{{ server_name | upper }}
{{ config_path | lower }}
{{ app_name | replace(' ', '_') }}

# List filters
{{ servers | join(',') }}
{{ packages | length }}
{{ items | first }}
{{ items | last }}

# Math filters
{{ memory_mb | int }}
{{ (cpu_count * 1.5) | round }}
{{ price | abs }}

# Default values
{{ port | default(80) }}
{{ ssl_enabled | default(false) }}
```

Advanced Conditionals in Templates

```
# templates/config.j2
{% if environment == 'production' %}
debug_mode = false
log_level = error
{% elif environment == 'staging' %}
debug_mode = true
log_level = warning
{% else %}
debug_mode = true
log_level = debug
{% endif %}

{% if ssl_enabled and ssl_cert_path is defined %}
ssl_certificate {{ ssl_cert_path }}
ssl_certificate_key {{ ssl_key_path }}
{% endif %}
```

```
{% if ansible_facts['memtotal_mb'] > 4096 %}
large_memory_config = enabled
{% endif %}
```

Loops in Templates

```
# Simple loop
{% for server in backend_servers %}
server {{ server.name }} {{ server.ip }}:{{ server.port }};
{% endfor %}

# Loop with index
{% for item in items %}
item_{{ loop.index }}: {{ item }}
{% endfor %}

# Loop with conditionals
{% for user in users %}
{% if user.active %}
User {{ user.name }} is active
{% endif %}
{% endfor %}

# Nested loops
{% for app in applications %}
[{{ app.name }}]
{% for server in app.servers %}
    server{{ loop.index }} = {{ server }}
{% endfor %}
{% endfor %}
```

Loop Variables and Control

```
{% for server in servers %}
# Server {{ loop.index }} of {{ loop.length }}
name: {{ server.name }}
{% if loop.first %}
primary: true
{% endif %}
{% if loop.last %}
backup: true
{% endif %}
{% if not loop.last %}

'
{% endif %}
{% endfor %}
```

Available Loop Variables

- **loop.index**: Current iteration (1-indexed)
- **loop.index0**: Current iteration (0-indexed)
- **loop.first**: True on first iteration
- **loop.last**: True on last iteration
- **loop.length**: Total number of items

Template Filters for Data Transformation

```
# Ansible-specific filters
{{ admin_password | password_hash('sha512') }}
{{ config_data | to_json }}
{{ yaml_data | to_yaml }}
{{ app_id | to_uuid }}

# Date and time
{{ ansible_date_time.date }}
{{ ansible_date_time.iso8601 }}

# Network filters
{{ '192.168.1.0/24' | ipaddr('network') }}
{{ ip_address | ipv4 }}

# File path filters
{{ '/path/to/file.txt' | basename }}
{{ '/path/to/file.txt' | dirname }}
```

Error Handling with Blocks

Understanding Block, Rescue, Always

Blocks group tasks together and provide error handling similar to try-catch-finally in programming languages.

Key Concepts

- **block**: Contains tasks to execute (like 'try')
- **rescue**: Tasks run if block fails (like 'catch')
- **always**: Tasks run regardless of success/failure (like 'finally')
- **Scope**: Each section applies per host, not globally
- **Variables**: ansible_failed_task and ansible_failed_result available in rescue

Basic Block Structure

```
---
```

```
- name: Error handling with blocks
  hosts: all
  tasks:
    - block:
        - name: Task that might fail
          command: /usr/bin/risky-command

        - name: Another task
          service:
            name: myservice
            state: started

  rescue:
    - name: Handle the error
      debug:
        msg: "Error occurred: {{ ansible_failed_task.name }}"

    - name: Log the failure
      lineinfile:
        path: /var/log/ansible-errors.log
        line: "{{ ansible_date_time.iso8601 }}: {{ ansible_failed_result.msg }}"

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

Practical Example: Service Deployment

```
- name: Deploy application with error handling
  block:
    - name: Stop existing service
      service:
        name: myapp
        state: stopped

    - name: Deploy new version
      copy:
        src: /tmp/myapp-v2.0
        dest: /usr/local/bin/myapp
        backup: yes

    - name: Start service
      service:
        name: myapp
        state: started

  rescue:
    - name: Restore backup
      command: cp /usr/local/bin/myapp~ /usr/local/bin/myapp
```

```

- name: Restart old version
  service:
    name: myapp
    state: started

always:
  - name: Verify service status
    service_facts:

```

Block with Conditional Rescue

```

- block:
  - name: Attempt database migration
    command: /usr/local/bin/migrate-db.sh
    register: migration_result

rescue:
  - name: Check if it's a connection error
    debug:
      msg: "Database connection failed"
      when: "'Connection refused' in ansible_failed_result.stderr"

  - name: Check if it's a migration error
    debug:
      msg: "Migration failed, rolling back"
      when: "'Migration error' in ansible_failed_result.stderr"

  - name: Rollback migration
    command: /usr/local/bin/rollback-db.sh
    when: "'Migration error' in ansible_failed_result.stderr"

```

Conditionals and Control Flow

Using when for Conditionals

```

# Simple condition
- name: Install Apache on Debian
  apt:
    name: apache2
    state: present
  when: ansible_facts['os_family'] == "Debian"

# Multiple conditions (AND)
- name: Install on production Ubuntu
  package:
    name: production-app
  when:
    - environment == "production"
    - ansible_facts['distribution'] == "Ubuntu"

```

```
# OR condition
- name: Install on RedHat or CentOS
  package:
    name: myapp
  when: ansible_facts['os_family'] == "RedHat" or
        ansible_facts['distribution'] == "CentOS"
```

Conditionals with Variables

```
# Check if variable is defined
- name: Use custom config
  template:
    src: custom.j2
    dest: /etc/app/config
  when: custom_config is defined

# Check if variable is undefined
- name: Use default config
  copy:
    src: default.conf
    dest: /etc/app/config
  when: custom_config is not defined

# Boolean checks
- name: Enable SSL
  command: enable-ssl.sh
  when: ssl_enabled | bool

# Check variable content
- name: High memory configuration
  template:
    src: high-mem.j2
    dest: /etc/config
  when: ansible_facts['memtotal_mb'] > 8192
```

failed_when - Custom Failure Conditions

```
# Based on return code
- name: Run custom script
  command: /usr/local/bin/check-status.sh
  register: status_check
  failed_when: status_check.rc > 1

# Based on output content
- name: Validate configuration
  command: validate-config /etc/app/config
  register: validation
  failed_when: "'ERROR' in validation.stderr"

# Multiple failure conditions (AND)
- name: Check disk space
```

```

shell: df -h /data
register: disk_check
failed_when:
  - disk_check.rc != 0
  - "'100%' in disk_check.stdout"

# Never fail
- name: Informational command
  command: /usr/bin/info-command
  failed_when: false

```

changed_when - Control Change Status

```

# Based on output
- name: Check if file exists
  command: test -f /tmp/myfile
  register: file_check
  changed_when: false
  failed_when: false

# Based on return code
- name: Create file if needed
  command: touch /tmp/myfile
  register: touch_result
  changed_when: touch_result.rc == 0

# Complex condition
- name: Update configuration
  shell: update-config.sh
  register: update_result
  changed_when: "'Updated' in update_result.stdout"

# Never report change
- name: Read-only check
  command: cat /etc/config
  changed_when: false

```

Combining Error Handling Directives

```

- name: Complex task with full control
  command: /usr/bin/complex-operation
  register: operation_result
  ignore_errors: yes
  failed_when:
    - operation_result.rc != 0
    - "'Ignorable error' not in operation_result.stderr"
  changed_when:
    - operation_result.rc == 0
    - "'Changes made' in operation_result.stdout"

- name: Handle the result

```

```
debug:  
  msg: "Operation {{ item }} if operation_result.failed {{ item }} failed {{ item }} succeeded {{ item }} endif {{ item }}"
```

Loops in Ansible

Basic Loop with loop

```
# Simple list loop  
- name: Install multiple packages  
  package:  
    name: "{{ item }}"  
    state: present  
  loop:  
    - nginx  
    - git  
    - curl  
    - vim  
  
# Loop over variables  
- name: Create multiple users  
  user:  
    name: "{{ item }}"  
    state: present  
  loop: "{{ user_list }}"
```

Loop with Dictionary Items

```
# Loop over dictionary  
- name: Create users with details  
  user:  
    name: "{{ item.name }}"  
    uid: "{{ item.uid }}"  
    groups: "{{ item.groups }}"  
    state: present  
  loop:  
    - { name: 'alice', uid: 1001, groups: 'admin' }  
    - { name: 'bob', uid: 1002, groups: 'users' }  
    - { name: 'charlie', uid: 1003, groups: 'developers' }  
  
# Using dict2items filter  
- name: Configure services  
  service:  
    name: "{{ item.key }}"  
    state: "{{ item.value }}"  
  loop: "{{ services | dict2items }}"  
  vars:  
    services:  
      nginx: started  
      mysql: stopped  
      redis: started
```

Loop Control

```
# Custom loop variable name
- name: Configure applications
  template:
    src: "{{ application.name }}.j2"
    dest: "/etc/{{ application.name }}/config"
  loop: "{{ applications }}"
  loop_control:
    loop_var: application

# Add labels for clarity
- name: Deploy services
  include_role:
    name: "{{ service.role }}"
  loop: "{{ services }}"
  loop_control:
    label: "{{ service.name }}"

# Add pause between iterations
- name: Restart services with delay
  service:
    name: "{{ item }}"
    state: restarted
  loop: "{{ services }}"
  loop_control:
    pause: 10
```

with_items vs loop

```
# Old style (with_items)
- name: Install packages
  package:
    name: "{{ item }}"
  with_items:
    - nginx
    - mysql

# New style (loop) - recommended
- name: Install packages
  package:
    name: "{{ item }}"
  loop:
    - nginx
    - mysql
```

Key Differences

- **loop** is the modern recommended approach
- **with_items** automatically flattens one level
- Use loop with flatten filter for similar behavior

Example:

- `with_items: [1, [2, 3], 4]` results in: 1, 2, 3, 4
- `loop: [1, [2, 3], 4]` results in: 1, [2, 3], 4

Nested Loops

```
# Using loop with product filter
- name: Create user-group combinations
  command: "usermod -aG {{ item.1 }} {{ item.0 }}"
  loop: "{{ users | product(groups) | list }}"
  vars:
    users:
      - alice
      - bob
    groups:
      - developers
      - admins

# Nested loops with subelements
- name: Configure SSH keys for users
  authorized_key:
    user: "{{ item.0.name }}"
    key: "{{ item.1 }}"
  loop: "{{ users | subelements('ssh_keys') }}"
  vars:
    users:
      - name: alice
      ssh_keys:
        - "ssh-rsa AAA..."
        - "ssh-rsa BBB..."
```

Role Dependencies and Best Practices

Defining Role Dependencies

```
# roles/webserver/meta/main.yml
---
dependencies:
  - role: common
  vars:
    ntp_servers:
      - 0.pool.ntp.org
```

```
- role: firewall
  vars:
    firewall_rules:
      - { port: 80, protocol: tcp }
      - { port: 443, protocol: tcp }

- role: ssl_certificates
  when: ssl_enabled | default(false)
```

Dependencies are installed and run before the parent role. They can have their own variables and conditions.

Installing Roles from Ansible Galaxy

```
# Install a role
ansible-galaxy role install geerlingguy.nginx

# Install to specific directory
ansible-galaxy role install --roles-path ./roles geerlingguy.nginx

# Install from requirements file
# requirements.yml
---
roles:
  - name: geerlingguy.nginx
    version: 3.1.4
  - name: geerlingguy.mysql
    version: 4.3.3

# Install from requirements
ansible-galaxy role install -r requirements.yml
```

Role Best Practices

- **Single Responsibility:** Each role should have one clear purpose
- **Meaningful Names:** Use descriptive role and variable names
- **Documentation:** Maintain [README.md](#) with usage examples
- **Defaults:** Provide sensible defaults in defaults/main.yml
- **Idempotence:** Ensure roles can run multiple times safely
- **Testing:** Test roles independently and in combination
- **Version Control:** Use semantic versioning for role releases
- **Dependencies:** Minimize dependencies; document required ones

Testing Roles

```
# Using molecule for role testing
# Install molecule
pip install molecule

# Initialize molecule in role directory
cd roles/webserver
molecule init scenario

# Run tests
molecule test

# Test sequence includes:
# 1. Create test instance
# 2. Run role
# 3. Verify with tests
# 4. Destroy instance
```

Testing Best Practices

- **Molecule:** Official testing framework
- **Test-Driven Development:** Write tests first
- **Multiple Platforms:** Test across different OS versions
- **CI Integration:** Automate testing in pipelines

Practical Lab Exercise

Lab: Complete Web Application Deployment Role

Create a role that deploys a complete web application stack with the following features:

- Install and configure Nginx
- Deploy application code
- Configure database connection
- Setup SSL certificates
- Handle deployment failures gracefully
- Support multiple environments (dev, staging, prod)
- Include comprehensive error logging

Lab Requirements

Role Details:

- **Role name:** webapp
- **Components:** Nginx, application code, database config, SSL
- **Error Handling:** Use blocks with rescue for graceful failure handling
- **Templates:** Advanced templates with loops and conditionals
- **Variables:** Proper variable precedence across environments
- **Dependencies:** Include role dependencies as needed
- **Testing:** Verify with different configurations

Session Summary

Key Learning Outcomes

Students completing this session should demonstrate proficiency in:

- **Role Directory Structure:** Understanding standard organization
- **Role Creation:** Using ansible-galaxy to initialize roles
- **Advanced Templates:** Filters, loops, and conditionals in Jinja2
- **Error Handling:** Using block, rescue, and always
- **Conditionals:** when, failed_when, changed_when
- **Loops:** Modern loop syntax and loop control
- **Dependencies:** Managing role relationships
- **Best Practices:** Professional role development standards