

# Session 2: Ansible Playbooks and Variables

## Detailed Training Notes for Beginners

# YAML Syntax Essentials for Ansible

## Understanding YAML Fundamentals

YAML (YAML Ain't Markup Language) serves as Ansible's primary configuration language due to its human-readable structure and simplicity.

## Core YAML Principles

- **Document Start:** Every YAML file begins with `---`
- **Indentation:** Uses spaces only (never tabs), typically 2 spaces per level
- **Case Sensitivity:** All keys, values, and variable names are case-sensitive
- **Data Types:** Supports strings, numbers, booleans, lists, and dictionaries

## YAML Syntax Elements

### Key-Value Pairs

```
# Basic syntax - space after colon is mandatory
key: value
name: "John Doe"
port: 80
enabled: true
```

### Lists (Arrays)

```
# List using hyphens
fruits:
  - apple
  - banana
  - orange

# Inline format
colors: [red, green, blue]
```

## Dictionaries (Objects)

```
# Nested dictionary structure
server:
  name: web-server-01
  ip: 192.168.1.100
  services:
    - nginx
    - mysql
  configuration:
    max_connections: 1000
    timeout: 30
```

## YAML Best Practices

- **Consistent Indentation:** Always use 2 spaces for each level
- **Quote Special Characters:** Use quotes for strings with colons, brackets, or special characters
- **Meaningful Comments:** Document complex configurations using #
- **Validate Syntax:** Use `ansible-playbook --syntax-check` to verify structure
- **Line Length:** Keep lines under 120 characters for readability

# Ansible Playbook Structure

## Understanding Playbook Architecture

Playbooks are YAML files that define automation tasks across multiple hosts. They serve as executable instructions for system configuration and application deployment.

## Basic Playbook Structure

```
---
# Playbook starts with document separator
- name: Descriptive name for the play
  hosts: target_hosts_or_groups
  become: yes # Enable privilege escalation
  gather_facts: yes # Collect system information
  vars:
    # Play-level variables
    variable_name: value
  tasks:
    # List of tasks to execute
    - name: Task description
      module_name:
        parameter: value
  handlers:
    # Event-driven tasks
    - name: Handler description
```

```
module_name:
  parameter: value
```

## Playbook Components Explained

### Play Definition

- **name:** Descriptive identifier for the play (optional but recommended)
- **hosts:** Target hosts or groups from inventory
- **become:** Enables privilege escalation (sudo/su)
- **gather\_facts:** Controls automatic fact collection (default: yes)

### Task Structure

```
tasks:
  - name: Install Nginx web server
    package:
      name: nginx
      state: present
    notify: restart nginx

  - name: Copy configuration file
    template:
      src: nginx.conf.j2
      dest: /etc/nginx/nginx.conf
      backup: yes
    notify: restart nginx
```

## Multi-Play Playbooks

```
---
# First play - configure web servers
- name: Configure web servers
  hosts: webservers
  become: yes
  tasks:
    - name: Install Apache
      package:
        name: httpd
        state: present

# Second play - configure database servers
- name: Configure database servers
  hosts: databases
  become: yes
  tasks:
    - name: Install MySQL
      package:
```

```
name: mysql-server
state: present
```

## Executing Playbooks

### Basic Execution Commands

```
# Run playbook with default inventory
ansible-playbook playbook.yml

# Specify custom inventory
ansible-playbook -i custom_inventory.ini playbook.yml

# Verbose output for debugging
ansible-playbook -v playbook.yml

# Check syntax before execution
ansible-playbook --syntax-check playbook.yml

# Dry run - show what would change
ansible-playbook --check playbook.yml

# Show differences for changed files
ansible-playbook --diff playbook.yml
```

# Variables in Ansible

## Dynamic Configuration Management

Variables provide flexibility and reusability in Ansible automation, allowing playbooks to adapt to different environments without code duplication.

### Variable Definition Methods

#### Playbook Variables

```
---
- name: Web server configuration
  hosts: webservers
  vars:
    http_port: 80
    document_root: /var/www/html
    server_admin: admin@example.com
  tasks:
    - name: Configure Apache port
      lineinfile:
        path: /etc/apache2/ports.conf
```

```
regex: '^Listen'  
line: "Listen {{ http_port }}"
```

## External Variables Files

```
# vars.yml  
---  
database_host: db.example.com  
database_port: 3306  
database_name: webapp  
max_connections: 200  
  
# playbook.yml  
---  
- name: Configure application  
  hosts: appservers  
  vars_files:  
    - vars.yml  
  tasks:  
    - name: Template database config  
      template:  
        src: database.conf.j2  
        dest: /etc/app/database.conf
```

## Command Line Variables

```
# Pass variables at runtime  
ansible-playbook -e "env=production" -e "debug=false" playbook.yml  
  
# Use variable file  
ansible-playbook --extra-vars "@production_vars.yml" playbook.yml
```

## Variable Precedence Hierarchy

Understanding variable precedence prevents conflicts and ensures predictable behavior:

1. **Extra vars** (command line `-e`) - **Highest Priority**
2. **Task vars** (including block and `include_vars`)
3. **Play vars\_prompt and vars\_files**
4. **Play vars**
5. **Host facts / cached set\_facts**
6. **Playbook host\_vars**
7. **Playbook group\_vars**
8. **Inventory host\_vars**

### 9. Inventory group\_vars

### 10. Inventory file variables

### 11. Role defaults - Lowest Priority

## Practical Precedence Example

```
# group_vars/all.yml (lower precedence)
app_version: "1.0.0"
debug_mode: false

# host_vars/web01.yml (higher precedence)
app_version: "1.1.0"

# Command line (highest precedence)
# ansible-playbook -e "debug_mode=true" playbook.yml

# Result: web01 gets app_version=1.1.0, debug_mode=true
```

# Host and Group Variables

## Organized Variable Management

Host and group variables provide structured approaches to managing configuration differences across your infrastructure.

## Directory Structure for Variables

```
project/
├── ansible.cfg
├── inventory.ini
├── playbooks/
├── group_vars/
│   ├── all.yml           # Variables for all hosts
│   ├── webserver.yml     # Variables for webserver group
│   └── database.yml      # Variables for database group
├── host_vars/
│   ├── web01.yml         # Variables specific to web01
│   └── db01.yml          # Variables specific to db01
```

## Group Variables Examples

## group\_vars/all.yml - Global Configuration

```
---
# Variables applied to all hosts
timezone: "UTC"
ntp_servers:
  - "0.pool.ntp.org"
  - "1.pool.ntp.org"
common_packages:
  - vim
  - htop
  - curl
backup_retention_days: 30
```

## Web Server Specific Variables

### group\_vars/webserver.yml

```
---
# Variables for webserver group
web_port: 80
ssl_port: 443
document_root: /var/www/html
max_connections: 500
worker_processes: auto
ssl_certificate: /etc/ssl/certs/server.crt
ssl_private_key: /etc/ssl/private/server.key
```

## Database Specific Variables

### group\_vars/databases.yml

```
---
# Variables for database group
db_port: 3306
max_connections: 200
buffer_pool_size: "1G"
log_file_size: "256M"
backup_schedule: "0 2 * * *"
replication_enabled: true
```

## Host-Specific Overrides

### host\_vars/web01.yml

```
---
# Specific configuration for web01
server_id: 1
ip_address: "192.168.1.10"
max_connections: 750 # Higher than group default
special_modules:
  - mod_rewrite
  - mod_ssl
maintenance_window: "Sunday 02:00-04:00"
```

# Ansible Facts and Magic Variables

## System Discovery and Dynamic Information

Ansible facts provide discovered information about target systems, enabling intelligent, conditional automation based on actual system states.

## Understanding Ansible Facts

Facts are system information automatically gathered by the `setup` module at the beginning of each play (unless `gather_facts: no` is specified).

## What Facts Include

- **System Information:** OS family, distribution, version, architecture
- **Hardware Details:** CPU, memory, disk space, network interfaces
- **Network Configuration:** IP addresses, hostnames, routing tables
- **Software Versions:** Installed packages, services status

## Accessing Facts via Ad-hoc Commands

```
# Gather all facts for hosts
ansible all -m setup

# Filter specific facts
ansible all -m setup -a "filter=ansible_os_family"
ansible all -m setup -a "filter=ansible_memory_mb"
ansible all -m setup -a "filter=ansible_network*"

# Gather facts for specific host
ansible web01 -m setup
```



## Using Facts in Playbooks

### Basic Fact Access

```
---
- name: System information playbook
  hosts: all
  tasks:
    - name: Display system information
      debug:
        msg: |
          Hostname: {{ ansible_facts['hostname'] }}
          OS Family: {{ ansible_facts['os_family'] }}
          Distribution: {{ ansible_facts['distribution'] }}
          Version: {{ ansible_facts['distribution_version'] }}
          Architecture: {{ ansible_facts['architecture'] }}
          Total Memory: {{ ansible_facts['memtotal_mb'] }}MB
```

### Conditional Tasks Based on Facts

```
- name: OS-specific package installation
  hosts: all
  tasks:
    - name: Install package on RedHat family
      yum:
        name: httpd
        state: present
      when: ansible_facts['os_family'] == "RedHat"

    - name: Install package on Debian family
      apt:
        name: apache2
        state: present
      when: ansible_facts['os_family'] == "Debian"

    - name: Configure service based on memory
      template:
        src: mysql.conf.j2
        dest: /etc/mysql/mysql.conf
      vars:
        buffer_pool_size: "{{ (ansible_facts['memtotal_mb'] * 0.7)|int }}M"
      when: ansible_facts['memtotal_mb'] > 1024
```

### Magic Variables

Magic variables provide information about the Ansible execution environment and other hosts in the play.

```

- name: Magic variables demonstration
  hosts: all
  tasks:
    - name: Show magic variables
      debug:
        msg: |
          Current host: {{ inventory_hostname }}
          All groups: {{ groups }}
          Web servers: {{ groups['webservers'] }}
          All hosts: {{ groups['all'] }}
          Play hosts: {{ ansible_play_hosts }}

```

## Custom Facts

### Creating Custom Facts Script

```

#!/bin/bash
# /etc/ansible/facts.d/custom.fact

echo '{
  "application_version": "2.1.3",
  "last_deployment": "'$(date -Iseconds)'",
  "deployment_user": "'$USER'",
  "custom_metric": '$RANDOM'
}'

```

## Using Custom Facts

```

- name: Deploy custom fact script
  hosts: all
  tasks:
    - name: Create facts directory
      file:
        path: /etc/ansible/facts.d
        state: directory
        mode: '0755'

    - name: Deploy custom fact script
      copy:
        src: custom.fact
        dest: /etc/ansible/facts.d/custom.fact
        mode: '0755'

    - name: Re-gather facts to include custom facts
      setup:
        filter: ansible_local

    - name: Display custom facts

```

```
debug:
  var: ansible_facts['local']['custom']
```

# Handlers and Notifications

## Event-Driven Automation

Handlers provide a mechanism for executing tasks only when notified by other tasks that have made changes.

## Handler Fundamentals

Handlers are special tasks that:

- Execute only when explicitly notified by other tasks
- Run only if the notifying task reports a "changed" state
- Execute at the end of a play, after all tasks complete
- Run only once, regardless of how many times they're notified
- Execute in the order defined in the handlers section

## Basic Handler Implementation

```
---
- name: Web server configuration
  hosts: webservers
  become: yes
  tasks:
    - name: Install Apache
      package:
        name: apache2
        state: present
        notify: start apache

    - name: Configure Apache
      template:
        src: apache2.conf.j2
        dest: /etc/apache2/apache2.conf
        backup: yes
        notify: restart apache

  handlers:
    - name: start apache
      service:
        name: apache2
        state: started
        enabled: yes

    - name: restart apache
```

```
service:
  name: apache2
  state: restarted
```

## Advanced Handler Patterns

### Multiple Notifications

```
tasks:
- name: Update configuration files
  template:
    src: "{{ item.src }}"
    dest: "{{ item.dest }}"
  loop:
    - { src: "nginx.conf.j2", dest: "/etc/nginx/nginx.conf" }
    - { src: "site.conf.j2", dest: "/etc/nginx/sites-available/default" }
  notify:
    - validate nginx config
    - restart nginx
    - update monitoring

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

- name: restart nginx
  service:
    name: nginx
    state: restarted
```

### Handler Groups with Listen

```
tasks:
- name: Update web server configuration
  template:
    src: config.j2
    dest: /etc/webserver/config.conf
  notify: "restart web services"

handlers:
- name: restart nginx
  service:
    name: nginx
    state: restarted
  listen: "restart web services"

- name: restart apache
  service:
    name: apache2
    state: restarted
```

```
listen: "restart web services"

- name: clear cache
  command: /usr/local/bin/clear-cache.sh
  listen: "restart web services"
```

# Jinja2 Templates in Ansible

## Dynamic Configuration Generation

Jinja2 templating enables dynamic file generation based on variables, facts, and conditional logic.

## Jinja2 Syntax Elements

- `{{ variable }}` - Variable substitution and expressions
- `{% statement %}` - Control statements (loops, conditionals)
- `{# comment #}` - Comments for documentation

## Basic Template Example

### Template File (templates/nginx.conf.j2)

```
# Nginx configuration for {{ inventory_hostname }}

user {{ nginx_user|default('nginx') }};
worker_processes {{ ansible_facts['processor_vcpus'] }};

server {
    listen {{ http_port|default(80) }};
    server_name {{ server_name|default(inventory_hostname) }};

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

    worker_connections {{ (ansible_facts['memtotal_mb'] / 4)|int }};

    {% if ssl_enabled|default(false) %}
    # SSL Configuration
    listen 443 ssl;
    ssl_certificate {{ ssl_cert_path }};
    ssl_certificate_key {{ ssl_key_path }};
    {% endif %}

    {% for server in backend_servers %}
    upstream backend_{{ loop.index }} {
        server {{ server.ip }}:{{ server.port }};
    }
}
```

```
{% endfor %}  
}
```

## Playbook Using Template

```
---  
- name: Configure Nginx servers  
  hosts: webservers  
  vars:  
    nginx_user: www-data  
    http_port: 80  
    document_root: /var/www/html  
    ssl_enabled: true  
    ssl_cert_path: /etc/ssl/certs/nginx.crt  
    ssl_key_path: /etc/ssl/private/nginx.key  
    backend_servers:  
      - { ip: "192.168.1.100", port: 8080 }  
      - { ip: "192.168.1.101", port: 8080 }  
  
  tasks:  
    - name: Deploy Nginx configuration  
      template:  
        src: nginx.conf.j2  
        dest: /etc/nginx/sites-available/{{ inventory_hostname }}.conf  
        backup: yes  
        validate: nginx -t -c %s  
        notify: restart nginx
```

# Debug Module and Fact Inspection

## Troubleshooting and Information Display

The debug module provides essential functionality for displaying variable values, system information, and troubleshooting playbook execution.

## Basic Debug Usage

```
---  
- name: Debug demonstration playbook  
  hosts: all  
  vars:  
    app_name: "MyWebApp"  
    version: "2.1.0"  
  
  tasks:  
    - name: Display simple message  
      debug:  
        msg: "Configuring {{ app_name }} version {{ version }}"
```

```

- name: Show variable value
  debug:
    var: ansible_facts['os_family']

- name: Display multiple variables
  debug:
    msg: |
      Host: {{ inventory_hostname }}
      IP: {{ ansible_facts['default_ipv4']['address'] }}
      Memory: {{ ansible_facts['memtotal_mb'] }}MB
      CPU Count: {{ ansible_facts['processor_vcpus'] }}

```

## Advanced Debug Techniques

### Conditional Debug Messages

```

- name: Conditional debugging
  debug:
    msg: "Warning: Low memory detected ({{ ansible_facts['memtotal_mb'] }}MB)"
    when: ansible_facts['memtotal_mb'] < 2048

- name: Debug with different verbosity levels
  debug:
    msg: "Detailed debug information"
    verbosity: 2 # Only shown with -vv or higher

```

# Best Practices and Troubleshooting

## Professional Ansible Development Standards

### Playbook Organization Best Practices

### Clear and Descriptive Naming

```

# Good examples
- name: Install and configure Apache web server with SSL
- name: Deploy application configuration for {{ app_name }}
- name: Create backup directory with proper permissions

# Poor examples
- name: Install stuff
- name: Configure things
- name: Do the needful

```

## Error Handling and Debugging

### Comprehensive Error Handling

```
- name: Robust service restart with error handling
  service:
    name: nginx
    state: restarted
    register: service_result
    retries: 3
    delay: 5
    failed_when: false
    changed_when: service_result.state == "started"

- name: Handle service restart failure
  debug:
    msg: "Service restart failed: {{ service_result.msg }}"
  when: service_result.failed
```

### Security Best Practices

```
- name: Secure configuration deployment
  template:
    src: secure_config.j2
    dest: /etc/app/config.conf
    owner: root
    group: app
    mode: '0640'
    backup: yes
  no_log: true # Prevent sensitive data in logs

- name: Use ansible-vault for sensitive variables
  debug:
    msg: "Database password: {{ db_password }}"
  vars:
    db_password: "{{ vault_db_password }}"
```

## Session Summary

### Key Learning Outcomes

Students completing this session should demonstrate proficiency in:

- **YAML Syntax Mastery:** Writing clean, properly formatted YAML
- **Playbook Structure:** Understanding play components and execution flow
- **Variable Management:** Using different variable types and understanding precedence
- **Fact Utilization:** Leveraging system facts for conditional logic



- **Handler Implementation:** Creating event-driven automation
- **Template Creation:** Building dynamic configuration files with Jinja2
- **Debugging Skills:** Using debug module and troubleshooting techniques

## Practical Assessment Exercise

Students should complete a mini-project demonstrating:

1. **Multi-host playbook** targeting different host groups
2. **Variable usage** from multiple sources (playbook, group\_vars, host\_vars)
3. **Conditional tasks** based on system facts
4. **Template deployment** with dynamic content
5. **Handler notifications** for service management
6. **Proper error handling** and debugging output

## Homework Assignments

Before Session 3, students should:

1. **Expand the Nginx lab** to include multiple environments (dev, staging, prod)
2. **Create custom facts** script for application-specific information
3. **Implement variable precedence** testing to understand override behavior
4. **Practice Jinja2 templates** with complex conditional logic and loops
5. **Research Ansible roles** structure for the next session

## Common Troubleshooting Issues

### YAML Syntax Errors

- Inconsistent indentation (mixing tabs and spaces)
- Missing colons in key-value pairs
- Incorrect list formatting
- Unquoted strings with special characters

### Variable Access Problems

- Using wrong variable reference syntax
- Variable undefined errors
- Precedence confusion
- Scope misunderstanding

## **Handler Not Triggering**

- Tasks not reporting "changed" state
- Incorrect handler names in notify
- Handler syntax errors preventing execution
- Understanding handler execution timing

## **Thank You**

### **Questions and Discussion**

This comprehensive foundation in playbooks and variables prepares students for advanced Ansible concepts including roles, advanced templating, and error handling strategies in subsequent sessions.