Ansible Advanced Topics: Loops, Conditionals, and Templating

Comprehensive PowerPoint Outline for Labs 08-11

Course Overview & Learning Objectives

Today's Focus: Advanced Ansible automation techniques

Learning Objectives:

- Master loops and iteration in Ansible playbooks
- Implement advanced loop control mechanisms
- Apply conditional logic for dynamic task execution
- Create and use Jinja2 templates for dynamic configuration

Lab Preparation: Understanding concepts for Labs 08-11

Review - Ansible Fundamentals

Key Concepts Recap:

Playbooks

YAML-based automation scripts

Tasks

Individual units of work

Modules

Built-in Ansible functions (e.g., user, package, file)

Variables

Data storage ({{ variable_name }})

Inventory

Host definitions and groups

Facts

System information gathered automatically

SECTION 1: ANSIBLE LOOPS

Introduction to Loops

Why Use Loops?

- **Eliminate repetitive tasks**
- **Reduce code duplication**
- **Increase maintainability**
- **Scale operations efficiently**

Example Problem:

```
# Without loops - repetitive and error-prone
- name: Create user alice
  user: name=alice state=present
- name: Create user bob
  user: name=bob state=present
- name: Create user charlie
  user: name=charlie state=present
```

The loop Keyword

Modern Ansible Looping Syntax:

```
- name: Create multiple users
  user:
    name: "{{ item }}"
    state: present
  loop:
    - alice
    - bob
    - charlie
```

Key Points:

- loop is the modern, preferred method
- {{ item }} represents each list element
- More readable than legacy with_* constructs
- Better error handling

Loop with Simple Lists

String Lists:

```
- name: Install packages
  package:
    name: "{{ item }}"
    state: present
  loop:
    - nginx
    - curl
    - git
```

Number Lists:

```
- name: Create numbered directories
  file:
    path: "/tmp/dir_{{ item }}"
    state: directory
  loop: "{{ range(1, 6) | list }}"
# Creates dir_1 through dir_5
```

Loop with Complex Data Structures

Dictionaries in Lists:

```
- name: Create users with specific properties
  user:
    name: "{{ item.name }}"
    shell: "{{ item.shell }}"
    uid: "{{ item.uid }}"
    state: present
  loop:
    - { name: 'alice', shell: '/bin/bash', uid: 1001 }
    - { name: 'bob', shell: '/bin/zsh', uid: 1002 }
    - { name: 'charlie', shell: '/bin/bash', uid: 1003 }
```

Accessing Nested Data:

- item.name accesses the 'name' key
- item.shell accesses the 'shell' key
- item.uid accesses the 'uid' key

Loop with Variables

Using Variables for Dynamic Lists:

```
- name: Manage dynamic content
  hosts: all
  vars:
    packages_list:
      - httpd
      - php
      - mysql
    users_data:
      - { username: 'dev1', group: 'developers' }
      - { username: 'admin1', group: 'administrators' }
  tasks:
    - name: Install packages from variable
      package:
        name: "{{ item }}"
        state: present
      loop: "{{ packages_list }}"
    - name: Create users from complex variable
      user:
        name: "{{ item.username }}"
        group: "{{ item.group }}"
      loop: "{{ users_data }}"
```

Loop with Registered Variables

Using Results from Previous Tasks:

```
- name: Get list of log files
  find:
    paths: /var/log
    patterns: "*.log"
  register: log_files
- name: Display each log file
  debug:
    msg: "Found log file: {{ item.path }}"
  loop: "{{ log_files.files }}"
```

Key Points:

01

register stores task output

02

Loop over registered results

03

Access nested attributes (e.g., item.path)

Legacy Loop Constructs (with_items)

Understanding with_items:

```
# Legacy syntax (still functional)
- name: Create users with with_items
  user:
    name: "{{ item }}"
    state: present
  with_items:
    - alice
    - bob
    - charlie
```

Migration to Modern Syntax:

1

with_items → loop

2

with_dict → loop: "{{ dict | dict2items }}"

3

with_nested → loop: "{{ list1 | product(list2) | list }}"

Why Modernize?

- Better error messages
- Consistent syntax
- Enhanced features
- Future-proof code

SECTION 2: LOOP CONTROL

Introduction to Loop Control

What is Loop Control?

Fine-tune loop behavior

Control output verbosity

Add timing controls

Manage variable scope

Implement break conditions

Basic Syntax:

```
- name: Task with loop control
  module_name:
    # module parameters
  loop: "{{ list_variable }}"
  loop_control:
    # loop control directives
```

Loop Control - Label

Simplifying Console Output:

```
- name: Create servers with clean output
  debug:
    msg: "Creating {{ item.name }} with {{ item.ram }} RAM"
  loop:
    - { name: 'web-server', ram: '8GB', disk: '100GB' }
    - { name: 'db-server', ram: '16GB', disk: '500GB' }
  loop_control:
    label: "{{ item.name }}"
```

Benefits:

Reduced console clutter

Focus on relevant information

Easier debugging

Professional output appearance

Loop Control - Index Variable

Tracking Loop Position:

```
- name: Create numbered configuration files
  template:
    src: config.j2
    dest: "/etc/app/config_{{ my_idx }}.conf"
  loop: "{{ server_configs }}"
  loop_control:
    index_var: my_idx
```

Custom Index Names:

```
- name: Process items with custom counter
  debug:
    msg: "Processing item {{ counter + 1 }}: {{ item }}"
  loop: "{{ item_list }}"
  loop_control:
    index_var: counter
```

Loop Control - Pause

Adding Delays Between Iterations:

```
- name: Create servers with delay
  ec2_instance:
    name: "{{ item }}"
    state: present
  loop:
    - web-server-1
    - web-server-2
    - web-server-3
  loop_control:
    pause: 30  # Wait 30 seconds between each server creation
```

Use Cases:

1. **Rate limiting API calls**
2. **Preventing resource exhaustion**
3. **Staged deployments**
4. **System resource management**

Loop Control - Break When

Conditional Loop Exit:

```
- name: Search for configuration file
  stat:
    path: "{{ item }}"
  loop:
    - "/etc/app/config.yml"
    - "/opt/app/config.yml"
    - "/home/user/config.yml"
  register: config_search
  loop_control:
    break_when: config_search.stat.exists
```

Complex Break Conditions:

```
- name: Generate password until criteria met
  set_fact:
    password: "{{ lookup('password', '/dev/null chars=ascii_letters,digits,punctuation length=12') }}"
  loop: "{{ range(0, 10) }}"
  loop_control:
    break_when: password is match('^(?=.*[A-Z])(?=.*[a-z])(?=.*[0-9])(?=.*[!@#$%^&*]).*$')
```

Loop Control - Custom Loop Variable

Renaming the Loop Variable:

```
- name: Configure services with custom variable names
  service:
    name: "{{ service_item.name }}"
    state: "{{ service_item.state }}"
  loop:
    - { name: 'nginx', state: 'started' }
    - { name: 'mysql', state: 'started' }
  loop_control:
    loop_var: service_item
```

Avoiding Variable Conflicts:

```
- name: Nested loop scenario
  include_tasks: configure_service.yml
  loop: "{{ services }}"
  loop_control:
    loop_var: outer_item
```

SECTION 3: CONDITIONALS

Introduction to Conditionals

Why Use Conditionals?

- **Dynamic task execution**
- **Environment-specific behaviors**
- **Error handling and recovery**
- **Platform compatibility**
- **Conditional configuration**

The when Statement:

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

Basic Conditional Syntax

Simple Conditions:

```
- name: Run task only when variable is true
  debug:
    msg: "Condition is met!"
  when: enable_feature == true
- name: Check if variable is defined
  debug:
    msg: "Variable exists"
  when: my_variable is defined
```

Important Notes:

> **No {{ }} brackets in when statements**

> **Raw Jinja2 expressions**

> **Boolean evaluation**

Conditionals with Facts

Using Ansible Facts:

```
- name: Install package based on OS
  package:
    name: "{{ item }}"
    state: present
  loop:
    - nginx
    - curl
  when: ansible_distribution == "Ubuntu"
- name: Configure firewall on specific OS versions
  firewalld:
    service: http
    permanent: yes
    state: enabled
  when:
    - ansible_distribution == "CentOS"
    - ansible_distribution_major_version == "8"
```

Common Facts for Conditions:

OS	DIST	VER	ARCH
ansible_os_family	**ansible_distribution**	**ansible_distribution_version**	**ansible_architecture**
OS family (Debian, RedHat, etc.)	Specific OS (Ubuntu, CentOS, etc.)	OS version	System architecture

Logical Operators

AND Conditions (Multiple conditions as list):

```
- name: Install Docker on specific systems
  package:
    name: docker
    state: present
  when:
    - ansible_distribution == "Ubuntu"
    - ansible_distribution_version >= "18.04"
    - ansible_architecture == "x86_64"
```

OR Conditions:

```
- name: Install package on multiple OS families
  package:
    name: vim
    state: present
  when: ansible_os_family == "Debian" or ansible_os_family == "RedHat"
```

NOT Conditions:

```
- name: Skip on production environment
  debug:
    msg: "Running in non-production"
  when: environment != "production"
```

Conditionals with Registered Variables

Using Task Results:

```
- name: Check if service exists
  command: systemctl is-enabled nginx
  register: service_check
  ignore_errors: yes
- name: Configure nginx if it exists
  template:
    src: nginx.conf.j2
    dest: /etc/nginx/nginx.conf
  when: service_check.rc == 0
- name: Handle service failure
  debug:
    msg: "Nginx service not found"
  when: service_check.rc != 0
```

Common Registered Variable Attributes:

result.rc

Return code

result.stdout

Standard output

result.stderr

Standard error

result.changed

Whether task changed anything

Complex Conditional Expressions

String Operations:

```
- name: Check if string contains value
  debug:
    msg: "Production environment detected"
  when: inventory_hostname.find('prod') != -1
- name: Check if string starts with prefix
  debug:
    msg: "Web server detected"
  when: inventory_hostname.startswith('web')
```

List and Dictionary Conditionals:

```
- name: Check if item in list
  debug:
    msg: "Required package found"
  when: "'nginx' in required_packages"
- name: Check dictionary key exists
  debug:
    msg: "Database configuration present"
  when: "'database' in config_dict"
```

Conditionals in Loops

Per-Item Conditions:

```
- name: Install packages conditionally
  package:
    name: "{{ item.name }}"
    state: present
  loop:
    - { name: 'nginx', condition: true }
    - { name: 'apache2', condition: false }
    - { name: 'mysql', condition: true }
  when: item.condition == true
```

Fact-Based Loop Conditions:

```
- name: Configure services per OS
  service:
    name: "{{ item }}"
    state: started
  loop:
    - nginx
    - mysql
  when: ansible_os_family == "Debian"
```

SECTION 4: JINJA2 TEMPLATING

Introduction to Jinja2 Templating

What is Jinja2?

Powerful templating engine for Python

Native integration with Ansible

Dynamic file generation

Variable manipulation and logic

Template Structure:

```
# Basic template syntax
Welcome to {{ ansible_hostname }}!
System: {{ ansible_distribution }} {{ ansible_distribution_version }}
Architecture: {{ ansible_architecture }}
```

Template File Extension: .j2

Template Module Basics

Using the Template Module:

```
- name: Deploy configuration from template
  template:
    src: config.conf.j2
    dest: /etc/app/config.conf
    owner: root
    group: root
    mode: '0644'
```

Directory Structure:

```
playbook_directory/
├──── templates/
│     └──── config.conf.j2
├──── playbook.yml
└──── inventory
```

Variables in Templates

Simple Variable Substitution:

```
# config.conf.j2
server_name = {{ server_name }}
port = {{ server_port }}
debug_mode = {{ debug_enabled }}
```

Using in Playbook:

```
- name: Deploy application config
  template:
    src: config.conf.j2
    dest: /etc/app/config.conf
  vars:
    server_name: "web-server-01"
    server_port: 8080
    debug_enabled: true
```

Complex Variable Access:

```
# Accessing nested variables
Database Host: {{ database.host }}
Database Port: {{ database.port }}
Database Name: {{ database.name }}
# Accessing list items
Primary DNS: {{ dns_servers[0] }}
Secondary DNS: {{ dns_servers[1] }}
```

Conditionals in Templates

If-Else Statements:

```
# nginx.conf.j2
server {
  listen {{ port }};
  server_name {{ server_name }};
  {% if enable_ssl %}
  listen 443 ssl;
  ssl_certificate {{ ssl_cert_path }};
  ssl_certificate_key {{ ssl_key_path }};
  {% endif %}
  {% if environment == "production" %}
  access_log /var/log/nginx/access.log;
  error_log /var/log/nginx/error.log;
  {% else %}
  access_log /var/log/nginx/debug_access.log;
  error_log /var/log/nginx/debug_error.log;
  {% endif %}
}
```

Boolean Checks:

```
{% if backup_enabled %}
# Backup configuration
backup_path = {{ backup_directory }}
backup_frequency = {{ backup_interval }}
{% endif %}
```

Loops in Templates

Basic Loop Structure:

```
# Generate user list
{% for user in users %}
User: {{ user.name }}
Shell: {{ user.shell }}
UID: {{ user.uid }}

{% endfor %}
```

Loop with Conditions:

```
# hosts.j2 - Generate hosts file entries
{% for host in web_servers %}
{% if host.environment == "production" %}
{{ host.ip }} {{ host.hostname }}.prod.example.com {{ host.hostname }}
{% endif %}
{% endfor %}
```

Loop Variables:

```
# Using loop variables
{% for item in items %}
Item {{ loop.index }}: {{ item.name }}
{% if not loop.last %}, {% endif %}
{% endfor %}
```

Jinja2 Filters

Common String Filters:

```
# String manipulation
Name (uppercase): {{ user_name | upper }}
Name (lowercase): {{ user_name | lower }}
Name (title case): {{ user_name | title }}
# String operations
Trimmed: "{{ message | trim }}"
Default value: {{ undefined_var | default('N/A') }}
```

List and Dictionary Filters:

```
# List operations
Total servers: {{ web_servers | length }}
First server: {{ web_servers | first }}
Unique items: {{ server_list | unique }}
# Dictionary operations
{% for key, value in config_dict | dictsort %}
{{ key }}: {{ value }}
{% endfor %}
```

Date and Number Filters:

```
# Date formatting
Generated: {{ ansible_date_time.iso8601 | to_datetime | strftime('%Y-%m-%d %H:%M:%S') }}
# Math operations
Total memory: {{ (ansible_memtotal_mb / 1024) | round(2) }} GB
```

Advanced Template Techniques

Template Inheritance and Includes:

```
# base.j2




 {% block content %}{% endblock %}


# page.j2
{% extends "base.j2" %}
{% block content %}
```

Welcome to {{ site_name }}!

{% endblock %}

Macros (Reusable Template Functions):

```
# macros.j2
{% macro render_user(user) %}
username: {{ user.name }}
shell: {{ user.shell }}
home: /home/{{ user.name }}
{% endmacro %}
# Using macro
{% from 'macros.j2' import render_user %}
{% for user in users %}
{{ render_user(user) }}
{% endfor %}
```

Template Error Handling

Debugging Templates:

```
# Debug output
{% if debug_mode %}
DEBUG: ansible_hostname = {{ ansible_hostname }}
DEBUG: server_list = {{ server_list }}
{% endif %}
# Safe variable access
{% if database is defined %}
Database configured: {{ database.host }}:{{ database.port }}
{% else %}
No database configuration found
{% endif %}
```

Default Values and Error Prevention:

```
# Provide defaults for undefined variables
Server: {{ server_name | default('localhost') }}
Port: {{ server_port | default(80) }}
# Check if variables exist before using
{% if ssl_config is defined and ssl_config.enabled %}
SSL Certificate: {{ ssl_config.cert_path }}
{% endif %}
```

Real-World Template Examples

Apache Virtual Host Template:

```
# apache-vhost.j2

  ServerName {{ server_name }}
  {% if server_aliases is defined %}
  {% for alias in server_aliases %}
  ServerAlias {{ alias }}
  {% endfor %}
  {% endif %}
  DocumentRoot {{ document_root }}
  {% if enable_logging %}
  ErrorLog {{ log_directory }}/{{ server_name }}_error.log
  CustomLog {{ log_directory }}/{{ server_name }}_access.log combined
  {% endif %}

   {% if allow_override %}
   AllowOverride All
   {% else %}
   AllowOverride None
   {% endif %}
   Require all granted
```

SECTION 5: INTEGRATION & BEST PRACTICES

MySQL Configuration Template Example

```
# my.cnf.j2
[mysqld]
port = {{ db_port | default(3306) }}
bind-address = {{ db_host }}
datadir = {{ mysql_datadir }}

{% if enable_ssl %}
ssl-ca = {{ ssl_ca_cert }}
ssl-cert = {{ ssl_server_cert }}
ssl-key = {{ ssl_server_key }}
{% endif %}

# User permissions
{% for user in db_users %}
GRANT {{ user.privileges }} ON {{ user.database }}.* TO '{{ user.name }}'@'{{ user.host }}' IDENTIFIED BY '{{ user.password }}';
{% endfor %}

# Replication settings
{% if replication_enabled %}
server-id = {{ server_id }}
log-bin = mysql-bin
binlog-format = ROW
read-only = {% if is_replica %}1{% else %}0{% endif %}
{% endif %}
```

Combining Loops, Conditionals, and Templates

Complex Integration Example:

```
- name: Configure web servers dynamically
  template:
    src: "{{ item.template }}"
    dest: "/etc/{{ item.service }}/{{ item.config_name }}"
  loop:
    - { service: 'nginx', template: 'nginx.conf.j2', config_name: 'nginx.conf' }
    - { service: 'apache2', template: 'apache.conf.j2', config_name: 'apache2.conf' }
  when:
    - item.service in enabled_services
    - ansible_os_family == "Debian"
  loop_control:
    label: "{{ item.service }}"
```

Dynamic Template with Nested Logic

```
# complex_config.j2
{% for environment in environments %}
# Environment: {{ environment.name }}
{% for server in environment.servers %}
# Server: {{ server.hostname }} ({{ server.ip_address }})
  {% if server.role == 'web' %}
  # Web server configuration
  Listen {{ server.web_port | default(80) }}
    {% if environment.ssl_enabled %}
    SSLEngine on
    SSLCertificateFile {{ environment.ssl_cert_path }}
    SSLCertificateKeyFile {{ environment.ssl_key_path }}
    {% endif %}
  {% elif server.role == 'database' %}
  # Database server configuration
  [mysqld]
  port = {{ server.db_port | default(3306) }}
    {% if environment.replication_enabled %}
    server-id = {{ loop.index }}
    log-bin = mysql-bin
    {% endif %}
  {% endif %}

  # Common services
  {% for service in server.services %}
    {% if service.enabled %}
  Start service: {{ service.name }}
      {% if service.params is defined %}
      Parameters: {{ service.params | join(', ') }}
      {% endif %}
    {% else %}
  Service {{ service.name }} is disabled.
    {% endif %}
  {% endfor %}
{% endfor %}
{% endfor %}
```

Performance Considerations

Loop Optimization:

- Use loop instead of with_* constructs
- Implement loop_control.label for large datasets
- Use changed_when and failed_when appropriately
- Consider async operations for time-consuming tasks

Template Optimization:

- Keep templates simple and readable
- Use includes for common template sections
- Minimize complex logic in templates
- Cache template results when possible

Conditional Optimization:

- Place most likely conditions first
- Use facts caching to improve performance
- Group related conditions logically

Error Handling and Debugging

Common Loop Issues:

```
# Handle undefined variables in loops
- name: Safe loop with undefined check
  debug:
    msg: "Processing {{ item }}"
  loop: "{{ undefined_list | default([]) }}"
# Error handling in loops
- name: Continue on failure
  command: "process {{ item }}"
  loop: "{{ file_list }}"
  ignore_errors: yes
  register: process_results
```

Template Debugging:

```
# Validate templates before deployment
- name: Test template rendering
  template:
    src: config.j2
    dest: /tmp/config.test
  check_mode: yes
```

Testing and Validation

Validation Strategies:

```
# Validate configuration after template deployment
- name: Deploy and validate configuration
  block:
    - name: Deploy configuration template
      template:
        src: app.conf.j2
        dest: /etc/app/app.conf
      notify: restart application
    - name: Validate configuration syntax
      command: app-binary check-config /etc/app/app.conf
      changed_when: false
  rescue:
    - name: Restore backup configuration
      copy:
        src: /etc/app/app.conf.backup
        dest: /etc/app/app.conf
        remote_src: yes
```

Using Check Mode:

```
# Test playbooks without making changes
ansible-playbook --check --diff playbook.yml
# Validate specific tasks
ansible-playbook --check --tags "configuration" playbook.yml
```

Best Practices Summary

Loops:

- ✅ Use loop instead of with_*
- ✅ Implement loop_control for better output
- ✅ Use meaningful variable names with loop_var
- ✅ Handle empty lists gracefully
- ❌ Don't nest loops unnecessarily

Conditionals:

- ✅ Use facts for environment detection
- ✅ Group related conditions together
- ✅ Test both positive and negative conditions
- ✅ Use registered variables effectively
- ❌ Don't make conditions overly complex

Best Practices Summary - Templates & Integration

Templates:

- ✅ Keep templates simple and readable
- ✅ Use includes for common template sections
- ✅ Minimize complex logic in templates
- ✅ Cache template results when possible
- ❌ Avoid complex Jinja2 filters where simple ones suffice

Integration:

- ✅ Handle undefined variables gracefully
- ✅ Implement robust error recovery (block/rescue)
- ✅ Validate configurations post-deployment
- ✅ Utilize check mode for dry runs
- ✅ Test with `--diff` for clarity

Lab Preparation Checklist

Before Starting Labs 08-11:

01

Environment Setup: Ansible controller and managed nodes ready

02

Inventory Configuration: Proper host groups defined

03

SSH Connectivity: Passwordless access configured

04

Templates Directory: Created for Jinja2 templates

05

Variable Files: Understanding of variable precedence

06

Editor Setup: Syntax highlighting for YAML and Jinja2

Key Commands to Remember:

```
# Run playbooks
ansible-playbook playbook.yml
# Check syntax
ansible-playbook --syntax-check playbook.yml
# Dry run
ansible-playbook --check playbook.yml
# Debug mode
ansible-playbook -vvv playbook.yml
```

Common Troubleshooting

Loop Issues:

- "item is undefined" → Check list variable definition
- "with_items deprecated" → Migrate to loop keyword
- Empty output → Verify list contains data

Conditional Issues:

- Syntax errors → Remove {{ }} from when statements
- Tasks always skipped → Check condition logic and variable values
- Undefined variable errors → Use is defined checks

Template Issues:

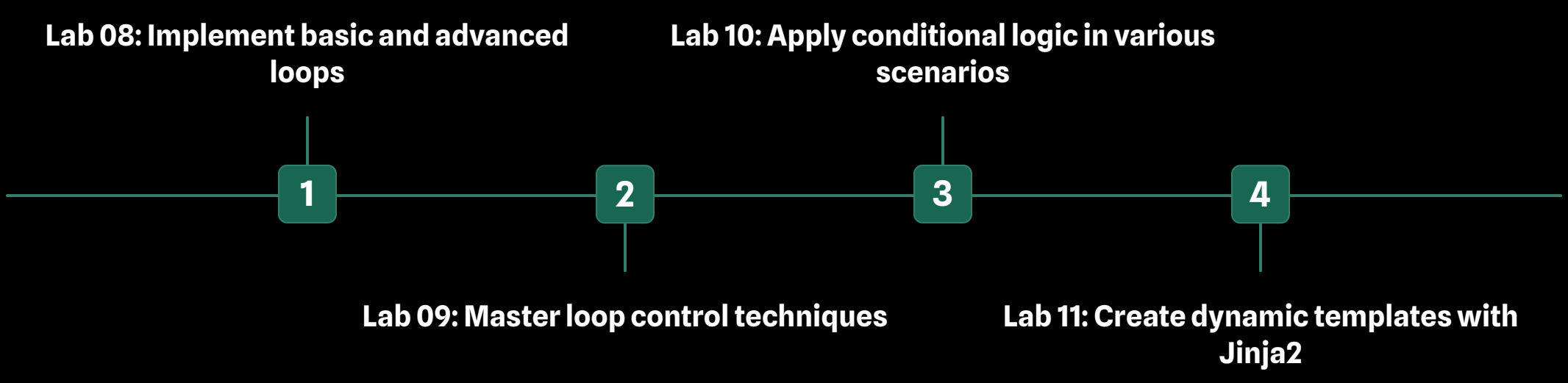
- Template not found → Check src path and templates directory
- Undefined variable in template → Use default filter or conditional blocks
- Syntax errors → Validate Jinja2 syntax

Ready for Labs - What's Next?

You Now Know:

- Loop fundamentals and advanced control
- Conditional logic for dynamic playbooks
- Jinja2 templating for configuration management
- Integration patterns and best practices

Lab Sequence:



Success Tips:

- [ ] Start simple, then add complexity
- [ ] Test frequently with check mode
- [ ] Read error messages carefully
- [ ] Use debug tasks for troubleshooting
- [ ] Refer back to these slides as needed

Good Luck with Your Labs!

Presenter Notes:

- **Key Teaching Points:**
  - Emphasize that the preceding modules have equipped students with foundational knowledge in automation.
  - Reiterate the importance of mastering loops, conditionals, and templating as core Ansible skills that will be used in nearly every automation task.
  - Highlight that these concepts are not just theoretical but have immediate practical application in the upcoming labs.
- **Common Student Questions/Misconceptions:**
  - "Why do I need these concepts?" - Explain that they enable dynamic, flexible, and reusable playbooks.
  - "When should I use a loop vs. a conditional?" - Briefly clarify the distinct use cases without overcomplicating.
  - "Jinja2 looks complicated." - Assure them that with practice, it becomes intuitive, and the labs will provide hands-on experience.
- **Practical Tips for Demonstrating Concepts:**
  - Briefly overview the sequence of labs (08-11), explaining how each lab builds upon the previous one, reinforcing the "You Now Know" concepts.
  - Encourage students to view the "Success Tips" as a practical checklist for debugging and efficient lab completion.
  - Recommend using the `debug` module aggressively to understand variable values and execution flow during troubleshooting.
- **Transitions to Next Topics (Labs):**
  - This card serves as a direct transition to the hands-on lab session.
  - Inform students that they are now ready to apply their knowledge.
- **Lab Preparation Reminders:**
  - Ensure all students have their lab environments provisioned and accessible.
  - Remind them to have the documentation or cheat sheets for Ansible loops, conditionals, and Jinja2 handy.
- **Time Estimates:**
  - Spend about 2-3 minutes on this summary slide to set the stage for the labs.
  - Allocate sufficient time for the lab session (e.g., 60-90 minutes for Labs 08-11 combined, depending on complexity and student pace).
- **Encouragement:**
  - End on an encouraging note, reinforcing that students have all the necessary knowledge.
  - Stress the importance of experimenting, making mistakes, and learning from them.
  - Encourage questions and peer-to-peer learning.
  - Remind them that referring back to the previous troubleshooting slide and this summary is highly recommended if they get stuck.