**Introduction to Ansible Templates**

**What are Ansible Templates?**

Ansible templates are files used to dynamically generate configuration files, scripts, and other text-based files in an automated way. They allow variables and logic to be embedded, making deployments more flexible and reusable.

Ansible uses **Jinja2**, a templating language, to process template files. Templates in Ansible typically have the .j2 extension.

**Why Use Templates?**

* Automates configuration file creation
* Reduces human errors
* Makes configuration flexible using variables
* Reusable and easy to maintain

**Understanding Template Basics**

Templates in Ansible are text files containing placeholders (variables and logic) that get replaced with actual values during execution. These templates are then deployed to target machines.

**Example of a Simple Template File**

A sample Apache web server configuration file (apache.conf.j2) might look like this:

<VirtualHost \*:80>

ServerAdmin {{ admin\_email }}

ServerName {{ server\_name }}

DocumentRoot {{ document\_root }}

</VirtualHost>

* {{ admin\_email }} - Placeholder for the server administrator's email.
* {{ server\_name }} - Placeholder for the server name.
* {{ document\_root }} - Placeholder for the website’s root directory.

When Ansible runs the template, it replaces these placeholders with actual values from the inventory or playbook.

**How Templates Work in Ansible**

1. Create a template file (.j2 format).
2. Use the template module in a playbook to deploy the file.
3. Ansible replaces variables with actual values.
4. The final file is generated and placed on the target machine.

**Ansible Template Module**

The template module is responsible for deploying template files to remote machines.

**Syntax of the Template Module**

- name: Deploy Apache Configuration

hosts: webservers

tasks:

- name: Copy Apache config using template

template:

src: apache.conf.j2

dest: /etc/apache2/sites-available/apache.conf

owner: root

group: root

mode: '0644'

**Explanation**

| **Parameter** | **Description** |
| --- | --- |
| src | Specifies the source template file (.j2) on the control node |
| dest | Destination path where the final file will be placed on the target machine |
| owner | Sets the file owner |
| group | Sets the file group |
| mode | Sets the file permissions |

When this playbook runs, Ansible generates the final configuration file using the provided variables and places it on the target machine at /etc/apache2/sites-available/apache.conf.

**Jinja2 Syntax for Ansible Templates**

**Variables**

Variables in Jinja2 are enclosed in {{ }}. Example:

Welcome, {{ user\_name }}!

If user\_name = 'John', the output will be:

Welcome, John!

**Conditional Statements**

You can use conditions inside templates:

{% if environment == 'production' %}

Server is running in Production Mode.

{% else %}

Server is in Development Mode.

{% endif %}

**Loops in Templates**

Loops are used to iterate over lists or dictionaries:

Users:

{% for user in users %}

- {{ user }}

{% endfor %}

If users = ['Alice', 'Bob', 'Charlie'], the output will be:

Users:

- Alice

- Bob

- Charlie

**Example: Generating a Configuration File**

Imagine you need to create an Nginx configuration file dynamically.

**Template File (nginx.conf.j2)**

server {

listen 80;

server\_name {{ domain\_name }};

root {{ document\_root }};

}

**Playbook Using the Template**

- name: Deploy Nginx Configuration

hosts: webservers

tasks:

- name: Generate nginx config

template:

src: nginx.conf.j2

dest: /etc/nginx/sites-available/default

**Output After Execution**

If domain\_name = example.com and document\_root = /var/www/html, the generated file will be:

server {

listen 80;

server\_name example.com;

root /var/www/html;

}

**Best Practices for Using Templates in Ansible**

* **Use Meaningful Variable Names**: Helps readability and maintainability.
* **Avoid Hardcoding Values**: Use variables instead of fixed values.
* **Use Comments in Templates**: Helps others understand your templates.
* **Keep Templates in a Separate Directory**: Organize your project better.
* **Validate Template Syntax**: Before applying, test using ansible-playbook --check.

**Conclusion**

Ansible templates provide a powerful way to manage and deploy configuration files dynamically. Using Jinja2 templates, we can customize files with variables and logic, making infrastructure automation more efficient and flexible. Understanding how to use templates effectively is a key skill for any Ansible user.

**Using Ansible Templates in Practice**

**Introduction**

Ansible templates allow dynamic file generation using the Jinja2 templating language. In this section, we will learn how to create templates, define variables, use loops, and apply conditional statements to generate customized configuration files.

**Creating Variables for Templates**

Before using a template, we need to define variables that will be used inside it. These variables can be stored in an Ansible inventory file or passed through a playbook.

**Example Variables in Ansible Playbook**

vars:

first: "Welcome"

user\_list:

- Alice

- Bob

- Charlie

environment: "dev"

In this example:

* first: A simple text variable.
* user\_list: A list containing names.
* environment: A variable to check conditions.

**Generating and Deploying a Template**

To deploy a template, we use the template module, which requires:

1. src: The source template file (.j2).
2. dest: The target destination on the remote machine.

**Example Playbook Using the Template Module**

- name: Deploy a Template

hosts: all

tasks:

- name: Generate and Deploy Template

template:

src: template.j2

dest: /etc/ansible/generated\_template.txt

This playbook:

* Uses a template file template.j2.
* Generates a final file at /etc/ansible/generated\_template.txt.

**Basic Template Syntax**

**Displaying a Variable**

Hello, {{ first }}!

If first = "Welcome", the output will be:

Hello, Welcome!

**Using Loops in Templates**

Jinja2 supports loops, allowing iteration over lists or dictionaries.

**Example: Looping Through a List**

Users:

{% for user in user\_list %}

- {{ user }}

{% endfor %}

If user\_list = ['Alice', 'Bob', 'Charlie'], the generated output will be:

Users:

- Alice

- Bob

- Charlie

**Using Conditional Statements**

Conditions allow dynamic changes based on variable values.

**Example: Checking an Environment Variable**

{% if environment == "dev" %}

Environment is Development.

{% else %}

Environment is not Development.

{% endif %}

If environment = "dev", the output will be:

Environment is Development.

If changed to environment = "qa", the output will be:

Environment is not Development.

**Updating and Regenerating a Template**

Once variables are modified, the template is regenerated by running the playbook again.

**Example: Changing a Variable Value**

If we change environment from dev to qa and re-run the playbook, the output dynamically adjusts.

vars:

environment: "qa"

After execution, the template reflects the new condition output.

**Other Features of Jinja2 in Ansible**

Jinja2 supports various advanced features, including:

* **Filters** (e.g., formatting text, converting YAML to JSON)
* **Whitespace control** for clean formatting
* **Complex data manipulation** with dictionaries and lists

For more details, refer to the [official Ansible documentation](https://docs.ansible.com/ansible/latest/user_guide/playbooks_templating.html).

**Conclusion**

Ansible templates provide a powerful way to automate configuration file generation. Using Jinja2, templates can dynamically adjust based on variables, loops, and conditions, making deployments more efficient and flexible.