### **Ansible Variables**

### **Defining Variables**

#### **In Playbooks**

- name: Example playbook

hosts: all

vars:

http\_port: 80

max\_clients: 200

tasks:

- name: Print the HTTP port

debug:

msg: "HTTP port is {{ http\_port }}"

#### **In Inventory Files**

[webservers]

web1 ansible\_host=192.168.1.100 http\_port=8080

web2 ansible\_host=192.168.1.101 http\_port=9090

#### **In Roles**

**defaults/main.yml**:  
  
http\_port: 80

max\_clients: 200

**vars/main.yml**:  
  
http\_port: 8080

### **Using Variables**

#### **In Tasks**

Variables are referenced using the Jinja2 templating syntax, typically {{ variable\_name }}.

- name: Start web server

service:

name: apache2

state: started

port: "{{ http\_port }}"

#### **In Templates**

server {

listen {{ http\_port }};

server\_name {{ server\_name }};

location / {

proxy\_pass http://{{ proxy\_host }};

}

}

#### **In Handlers**

- name: Restart web server

service:

name: apache2

state: restarted

**Reserved Keywords in Ansible**

Avoid using these reserved keywords as variable names in Ansible

[**Playbook Keywords — Ansible Community Documentation**](https://docs.ansible.com/ansible/latest/reference_appendices/playbooks_keywords.html)

### **Use Cases**

#### **Dynamic Configuration Management**

Variables allow for dynamic configurations, such as setting different ports for web servers based on the environment.

#### **Reusability and Modularity**

Using variables in roles enables you to create modular and reusable code. For example, a role for deploying a web server can be reused across different environments (dev, staging, production) by changing the variables.

#### **Conditional Execution**

Variables can be used for conditional execution of tasks.

- name: Install Apache on CentOS

yum:

name: httpd

state: present

when: ansible\_os\_family == "RedHat"

### **Example: Combining Variables**

- name: Deploy web application

hosts: webservers

vars:

http\_port: 8080

max\_clients: 100

tasks:

- name: Install Nginx

apt:

name: nginx

state: present

when: ansible\_os\_family == "Ubuntu"

- name: Start Nginx

service:

name: nginx

state: started

enabled: yes

- name: Deploy configuration file

template:

src: nginx.conf.j2

dest: /etc/nginx/nginx.conf

notify:

- Restart Nginx

handlers:

- name: Restart Nginx

service:

name: nginx

state: restarted

### **Jinja2 Template**

### Jinja2 is a modern and designer-friendly templating engine for Python. In Ansible, Jinja2 templates are used to create dynamic content. They allow you to generate configuration files and scripts dynamically based on variables and conditions. Here's an overview of how to use Jinja2 templates in Ansible:

### **1. Basic Structure**

### A Jinja2 template is essentially a text file with placeholders for variables and expressions. These placeholders are enclosed in double curly braces {{ }}. Control structures like loops and conditionals are enclosed in {% %}.

### **2. Using Variables**

### You can use variables in your templates by enclosing them in double curly braces. For example, if you have a variable hostname, you can use it in your template as follows:

### server {

### listen 80;

### server\_name {{ hostname }};

### ...

### }

### **Using the template Module in Ansible**

### The template module in Ansible is used to copy a Jinja2 template file from the control node to the managed nodes. This module processes the template file, rendering it with variables and logic defined within the template, and then places the rendered file on the managed nodes.

### **What Happens When We Use the template Module**

### When we use the template module, the following happens:

### **Template Rendering**: The Jinja2 template file is rendered with the provided variables.

### **File Transfer**: The rendered file is transferred to the specified destination on the managed nodes.

### **Change Detection**: Ansible detects if the contents of the destination file have changed.

### **Notification**: If a change is detected, handlers notified by this task are triggered.

#### **Example Using the template Module**

### **Jinja2 Template File (nginx.conf.j2)**

### server {

### listen 80;

### server\_name {{ hostname }};

### 

### location / {

### proxy\_pass http://backend;

### {% for header in headers %}

### add\_header {{ header.name }} "{{ header.value }}";

### {% endfor %}

### }

### }

### 

### 

### 

### 

### **Playbook (site.yml)**

### - name: Configure Nginx

### hosts: webservers

### vars:

### hostname: "example.com"

### headers:

### - { name: "X-Frame-Options", value: "DENY" }

### - { name: "X-Content-Type-Options", value: "nosniff" }

### - { name: "X-XSS-Protection", value: "1; mode=block" }

### tasks:

### - name: Generate Nginx config file from template

### template:

### src: templates/nginx.conf.j2

### dest: /etc/nginx/nginx.conf

### notify:

### - restart nginx

### 

### handlers:

### - name: restart nginx

### service:

### name: nginx

### state: restarted

### 

### **What Happens in This Example**

### **Rendering**: The nginx.conf.j2 template is rendered with the values of hostname and headers.

### **File Transfer**: The rendered nginx.conf file is placed in /etc/nginx/nginx.conf on the managed nodes.

### **Change Detection**: Ansible checks if the content of /etc/nginx/nginx.conf has changed.

### **Handler Notification**: If the content has changed, the restart nginx handler is notified to restart the Nginx service.

### **What Happens When We Don't Use the template Module**

### If we don't use the template module and instead use another method like copy or manual file editing, we lose the advantages of dynamic content generation and change detection.

#### **Example Without the template Module**

### **Static Configuration File (nginx.conf)**

### server {

### listen 80;

### server\_name example.com;

### 

### location / {

### proxy\_pass http://backend;

### add\_header X-Frame-Options "DENY";

### add\_header X-Content-Type-Options "nosniff";

### add\_header X-XSS-Protection "1; mode=block";

### }

### }

### 

### **Playbook (site.yml)**

### - name: Configure Nginx

### hosts: webservers

### tasks:

### - name: Copy Nginx config file

### copy:

### src: files/nginx.conf

### dest: /etc/nginx/nginx.conf

### notify:

### - restart nginx

### 

### handlers:

### - name: restart nginx

### service:

### name: nginx

### state: restarted

### 

### **Differences and Impact**

### **Static Content**: The configuration file is static and does not adapt to different variables or conditions.

### **No Dynamic Rendering**: Variables like hostname and headers are hard-coded and cannot be dynamically adjusted based on different environments or hosts.

### **Limited Flexibility**: If you need to change the hostname or add headers, you must manually edit the configuration file or create multiple static files for different environments.

### **Change Detection and Efficiency**: The copy module will overwrite the file every time the playbook runs, leading to unnecessary service restarts if the file hasn't changed.

### 

### **3. Conditionals**

### Jinja2 supports conditionals using the {% if %} statement.

### {% if environment == 'production' %}

### server {

### listen 80;

### server\_name {{ hostname }};

### }

### {% else %}

### server {

### listen 8080;

### server\_name {{ hostname }};

### }

### {% endif %}

### 

### **4. Loops**

### You can iterate over lists or dictionaries using {% for %}.

### server {

### listen 80;

### server\_name {{ hostname }};

### location / {

### proxy\_pass http://backend;

### {% for header in headers %}

### add\_header {{ header.name }} "{{ header.value }}";

### {% endfor %}

### }

### }

### 

### **5. Filters**

### Jinja2 provides various filters to transform the output. Filters are applied using the pipe | symbol.

### {{ some\_variable | upper }}

### {{ list\_variable | join(", ") }}

### 

### **6. Template File**

### Save your Jinja2 template as a .j2 file. For example, nginx.conf.j2.

### **7. Using Templates in Ansible Playbook**

### Use the template module in your Ansible playbook to process the template and generate the file.

### - name: Generate Nginx config file

### hosts: webservers

### vars:

### hostname: "example.com"

### environment: "production"

### headers:

### - { name: "X-Frame-Options", value: "DENY" }

### - { name: "X-Content-Type-Options", value: "nosniff" }

### tasks:

### - name: Create Nginx config file from template

### template:

### src: templates/nginx.conf.j2

### dest: /etc/nginx/nginx.conf

### notify:

### - restart nginx

### 

### **8. Handlers**

### Use handlers to perform actions like restarting a service when the template changes.

### 

### handlers:

### - name: restart nginx

### service:

### name: nginx

### state: restarted

### 

### **Example Jinja2 Template**

### Here’s a complete example of an Nginx configuration file template nginx.conf.j2:

### server {

### listen 80;

### server\_name {{ hostname }};

### 

### location / {

### proxy\_pass http://backend;

### {% for header in headers %}

### add\_header {{ header.name }} "{{ header.value }}";

### {% endfor %}

### }

### 

### {% if environment == 'production' %}

### error\_log /var/log/nginx/error.log;

### access\_log /var/log/nginx/access.log;

### {% else %}

### error\_log /var/log/nginx/error\_dev.log;

### access\_log /var/log/nginx/access\_dev.log;

### {% endif %}

### }

### 

### **Handlers in Ansible**

Handlers in Ansible are special tasks that are triggered by other tasks using the notify directive. They are typically used for tasks that need to run when certain conditions are met, such as restarting a service after a configuration file has been changed. Handlers are only run once, at the end of a playbook, regardless of how many tasks notify them.

### **Purpose of Handlers**

1. **Efficiency**: Handlers prevent unnecessary actions by ensuring that the action (e.g., service restart) is only performed if there was a change.
2. **Order and Dependency Management**: Handlers run in a specific order, after all tasks have been executed, ensuring that dependent services are restarted in the correct sequence.
3. **Idempotence**: They help maintain idempotence (ensuring the same result is produced even if the playbook runs multiple times) by only triggering actions when needed.

### **What Happens If We Don't Use Handlers?**

If handlers are not used, tasks that need to be conditionally executed based on changes in other tasks will either:

* Not be executed when necessary, leading to outdated configurations or services not being properly restarted.
* Be executed every time, leading to unnecessary actions and potentially longer playbook execution times.

### **Example Without Handlers**

Consider the scenario where an Nginx configuration file is updated, and the Nginx service needs to be restarted if the configuration file changes.

#### **Playbook Without Handlers**

- name: Configure Nginx

hosts: webservers

tasks:

- name: Generate Nginx config file from template

template:

src: templates/nginx.conf.j2

dest: /etc/nginx/nginx.conf

# No notify directive here

- name: Restart Nginx service

service:

name: nginx

state: restarted

### **Issues Without Handlers**

1. **Inefficiency**: The Nginx service will be restarted every time the playbook runs, regardless of whether the configuration file was actually changed.
2. **Unnecessary Downtime**: Restarting a service unnecessarily can lead to unwanted downtime and performance issues.

### **Example With Handlers**

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

server {

listen 80;

server\_name {{ hostname }};

location / {

proxy\_pass http://backend;

{% for header in headers %}

add\_header {{ header.name }} "{{ header.value }}";

{% endfor %}

}

}

#### **Playbook With Handlers**

- name: Configure Nginx

hosts: webservers

vars:

hostname: "example.com"

headers:

- { name: "X-Frame-Options", value: "DENY" }

- { name: "X-Content-Type-Options", value: "nosniff" }

- { name: "X-XSS-Protection", value: "1; mode=block" }

tasks:

- name: Generate Nginx config file from template

template:

src: templates/nginx.conf.j2

dest: /etc/nginx/nginx.conf

notify:

- restart nginx

handlers:

- name: restart nginx

service:

name: nginx

state: restarted

### **Explanation**

1. **Tasks Section**:
   * The template module generates the Nginx config file and places it at /etc/nginx/nginx.conf.
   * The notify directive is used to notify the handler named restart nginx if the template task results in a change.
2. **Handlers Section**:
   * The handler restart nginx is defined to restart the Nginx service.
   * This handler will only be triggered if the template task reports a change.

### **Benefits With Handlers**

* **Efficiency**: The Nginx service is only restarted if the configuration file changes.
* **Reduced Downtime**: Avoid unnecessary service restarts, leading to fewer disruptions.
* **Clarity and Maintainability**: The playbook is easier to read and maintain, with a clear separation of configuration changes and service management.

**Playbook Example for Jinja2 Template**

- name: Configure Nginx

hosts: web

become: true

vars:

hostname: "example.com"

header:

- { name: "X-Frame-Options", value: "DENY" }

- { name: "X-Content-Type-Options", value: "nosniff" }

- { name: "X-XSS-Protection", value: "1; mode=block" }

tasks:

- name: installing nginx

yum:

name: nginx

state: present

update\_cache: true

- name: Generate Nginx config file from template

template:

src: templates/nginx.conf.j2

dest: /etc/nginx/nginx.conf

notify:

- restart nginx

handlers:

- name: restart nginx

ansible.builtin.service:

name: nginx.service

state: restarted

+

### **Lab Credentials**

**Username: Ubuntu**

**Key file for below users**

### [**https://drive.google.com/file/d/1w0-o6pJlqixVzKXyDR841xJg60Ak63GE/view?usp=sharing**](https://drive.google.com/file/d/1w0-o6pJlqixVzKXyDR841xJg60Ak63GE/view?usp=sharing)

### **Poonam: 54.151.61.21**

### **Nensi: 54.151.4.59**

### **Yash: 13.56.224.86**

### **Palash: 13.56.163.175**

### **Jasminbanu: 54.177.197.158**

### **Shiv: 54.193.192.205**

### **Chirag: 54.183.199.95**

### **Shreya: 54.215.48.130**

### **—----------------------**

### 

**Key File:** [**https://drive.google.com/file/d/17Qw7j-A6wlmF3ra0CjqHgJ2vH8fghjh2/view?usp=sharing**](https://drive.google.com/file/d/17Qw7j-A6wlmF3ra0CjqHgJ2vH8fghjh2/view?usp=sharing)

### **Username: ubuntu**

### **Mayusha Rathod - 18.117.168.255**

### **Maaz Patel - 3.137.212.104**

### **Abhinav Jha - 3.16.130.73**

### **Aman Mansuri - 3.138.34.54**

### **Arsh Shaikh - 3.142.196.124**

### **Bhavik Chhabria - 3.15.221.144**

### **Bhavin Bhavsar - 18.226.248.82**

### **Eklavya Agal - 18.218.249.103**

### **Jash Shah - 18.118.137.225**

### **Manan Taori - 18.219.134.12**

### **Shital Chauhan - 18.226.93.177**

### **Yaksh Rawal - 3.135.203.28**

### **Farajnazish Ansari - 18.220.193.38**

### **Harshwardhan Patil - 3.145.42.200**

### **Santosh Pagire - 3.17.74.99**

### **Yash Parmar - 3.15.220.2**

### **Utsav Shah - 3.144.35.214**

### **Neel Patel - 3.12.41.67**

### **SURYRAJSINH JADEJA - 3.140.186.71**

### **Vraj Trivedi - 3.15.139.71**

**Siddh - 3.12.41.67**

### 

### **Project 01**

### **Deploy a Database Server with Backup Automation**

**Objective**: Automate the deployment and configuration of a PostgreSQL database server on an Ubuntu instance hosted on AWS, and set up regular backups.

### **Problem Statement**

**Objective**: Automate the deployment, configuration, and backup of a PostgreSQL database server on an Ubuntu instance using Ansible.

**Requirements**:

1. **AWS Ubuntu Instance**: You have an Ubuntu server instance running on AWS.
2. **Database Server Deployment**: Deploy and configure PostgreSQL on the Ubuntu instance.
3. **Database Initialization**: Create a database and a user with specific permissions.
4. **Backup Automation**: Set up a cron job for regular database backups and ensure that backups are stored in a specified directory.
5. **Configuration Management**: Use Ansible to handle the deployment and configuration, including managing sensitive data like database passwords.

### **Deliverables**

1. **Ansible Inventory File**
   * **Filename**: inventory.ini
   * **Content**: Defines the AWS Ubuntu instance and connection details for Ansible.
2. **Ansible Playbook**
   * **Filename**: deploy\_database.yml
   * **Content**: Automates the installation of PostgreSQL, sets up the database, creates a user, and configures a cron job for backups. It also includes variables for database configuration and backup settings.
3. **Jinja2 Template**
   * **Filename**: templates/pg\_hba.conf.j2
   * **Content**: Defines the PostgreSQL configuration file (pg\_hba.conf) using Jinja2 templates to manage access controls dynamically.
4. **Backup Script**
   * **Filename**: scripts/backup.sh
   * **Content**: A script to perform the backup of the PostgreSQL database. This script should be referenced in the cron job defined in the playbook.

### **Project 02**

**Objective**: Automate the setup of a multi-tier web application stack with separate database and application servers using Ansible.

### **Problem Statement**

**Objective**: Automate the deployment and configuration of a multi-tier web application stack consisting of:

1. **Database Server**: Set up a PostgreSQL database server on one Ubuntu instance.
2. **Application Server**: Set up a web server (e.g., Apache or Nginx) on another Ubuntu instance to host a web application.
3. **Application Deployment**: Ensure the web application is deployed on the application server and is configured to connect to the PostgreSQL database on the database server.
4. **Configuration Management**: Use Ansible to automate the configuration of both servers, including the initialization of the database and the deployment of the web application.

### **Deliverables**

1. **Ansible Inventory File**
   * **Filename**: inventory.ini
   * **Content**: Defines the database server and application server instances, including their IP addresses and connection details.
2. **Ansible Playbook**
   * **Filename**: deploy\_multitier\_stack.yml
   * **Content**: Automates:
     + The deployment and configuration of the PostgreSQL database server.
     + The setup and configuration of the web server.
     + The deployment of the web application and its configuration to connect to the database.
3. **Jinja2 Template**
   * **Filename**: templates/app\_config.php.j2
   * **Content**: Defines a configuration file for the web application that includes placeholders for dynamic values such as database connection details.
4. **Application Files**
   * **Filename**: files/index.html (or equivalent application files)
   * **Content**: Static or basic dynamic content served by the web application.