

Deploy a website on an Apache Web Server on Multiple EC2 Instances using Ansible Playbook

Overview

This is my Ansible project documentation. It provides a step-by-step guide to deploying a website on multiple EC2 instances using an Ansible playbook. It covers instance setup, security group configuration, Ansible role creation, and playbook execution.



Prerequisites

- Two EC2 instances (t2.micro) launched in the **Mumbai region**.
 - A **key pair** for SSH authentication.
 - Security groups configured to allow **HTTP (port 80)** and **HTTPS (port 443)** traffic.
 - Ansible installed on the control node.

[Launch an instance](#) | EC2 | ap-south-1 | +

ap-south-1.console.aws.amazon.com/ec2/home?region=ap-south-1#LaunchInstances:

aws | Search [Alt+S] | Asia Pacific (Mumbai) | Versha Jain |

EC2 > Instances > Launch an instance

Launch an instance Info

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

Name and tags Info

Name Add additional tags

Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below.

Search our full catalog including 1000s of application and OS images

Recent AMIs: Amazon Linux, macOS, Ubuntu, Windows, Red Hat, SUSE Linux, Debian. Browse more AMIs

Amazon Machine Image (AMI)

Red Hat Enterprise Linux 9 (HVM), SSD Volume Type
Free tier eligible

ami-02ddbb77fbf93ca4ca (64-bit (x86)) / ami-00bf39a0a3cddb14d (64-bit (Arm))

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Instance type Info | Get advice

t2.micro Free tier eligible

Family: t2 1 vCPU 1 GiB Memory Current generation: true
On-Demand Linux base pricing: 0.0124 USD per Hour On-Demand Windows base pricing: 0.017 USD per Hour
On-Demand Ubuntu base pricing: 0.0268 USD per Hour
On-Demand SUSE base pricing: 0.0142 USD per Hour

All generations Compare instance types

Additional costs apply for AMIs with pre-installed software

Key pair (login) Info

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required Create new key pair

Network settings Info

Network Info Edit
vpc-06a1b936935de1ea

Subnet Info
No preference (Default subnet in any availability zone)

Auto-assign public IP Info
Enable

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Summary

Number of instances Info
2

When launching more than 1 instance, consider EC2 Auto Scaling

Software Image (AMI)
Provided by Red Hat, Inc.
ami-02ddbb77fbf93ca4ca

Virtual server type (instance type)
t2.micro

Firewall (security group)
New security group

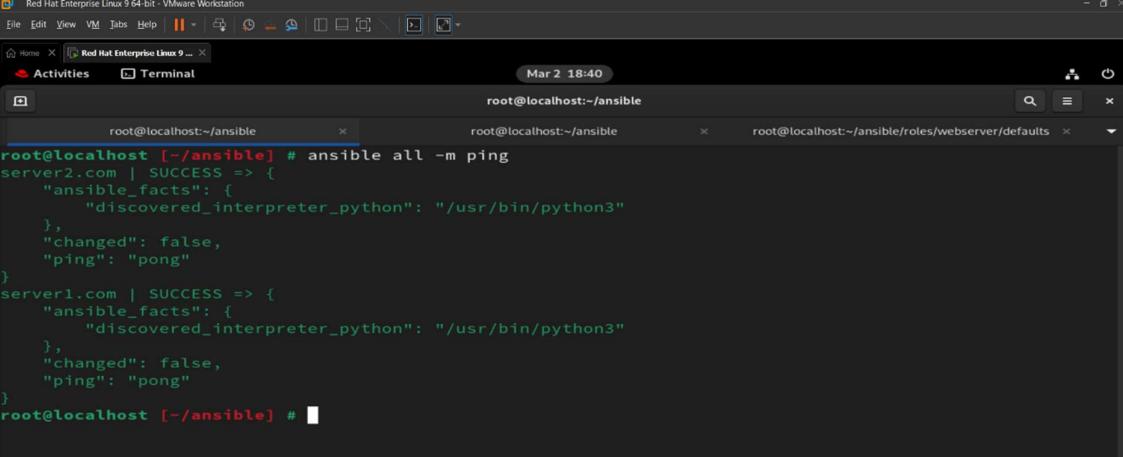
Storage (volumes)
1 volume(s) - 10 GiB

Cancel Launch instance Preview code

Step 1: Verify Connectivity

Ensure that Ansible can communicate with the EC2 instances:

```
# ansible all -m ping
```

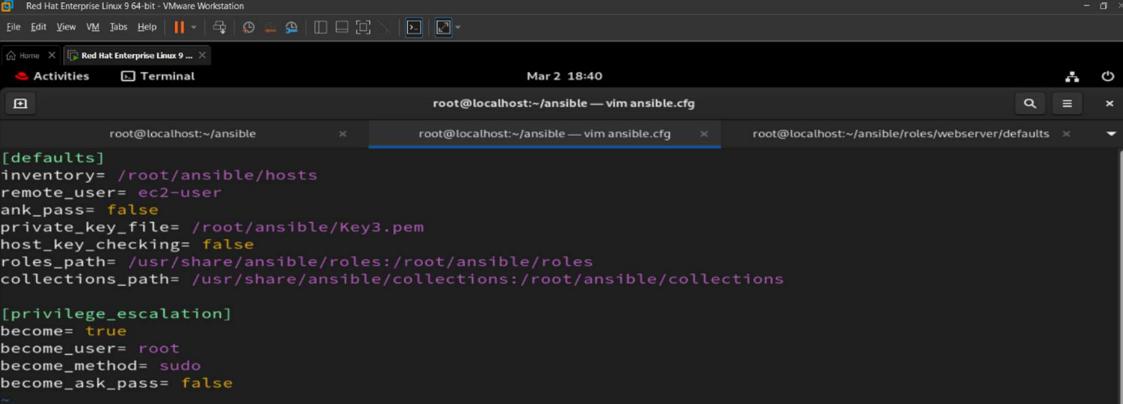


```
root@localhost [~/ansible] # ansible all -m ping
server2.com | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
server1.com | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
root@localhost [~/ansible] #
```

Step 2: Configure Ansible on the Control Node

Navigate to the Ansible directory and configure ansible.cfg with the following settings:

```
[defaults]
inventory= /root/ansible/hosts
remote_user= ec2-user
ask_pass= false
private_key_file= /root/ansible/Key3.pem
```



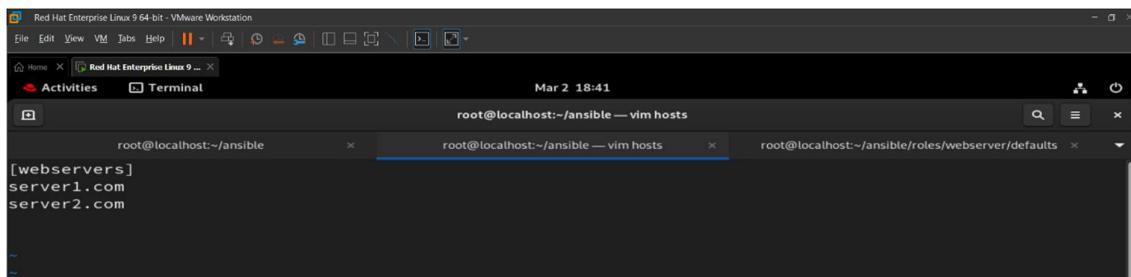
```
[defaults]
inventory= /root/ansible/hosts
remote_user= ec2-user
ask_pass= false
private_key_file= /root/ansible/Key3.pem
host_key_checking= false
roles_path= /usr/share/ansible/roles:/root/ansible/roles
collections_path= /usr/share/ansible/collections:/root/ansible/collections

[privilegeEscalation]
become= true
become_user= root
become_method= sudo
become_ask_pass= false
```

Step 3: Create an Inventory File (hosts)

Inside the Ansible folder, create an inventory file (hosts) with the following content:

```
[webservers]
server1.com
server2.com
```

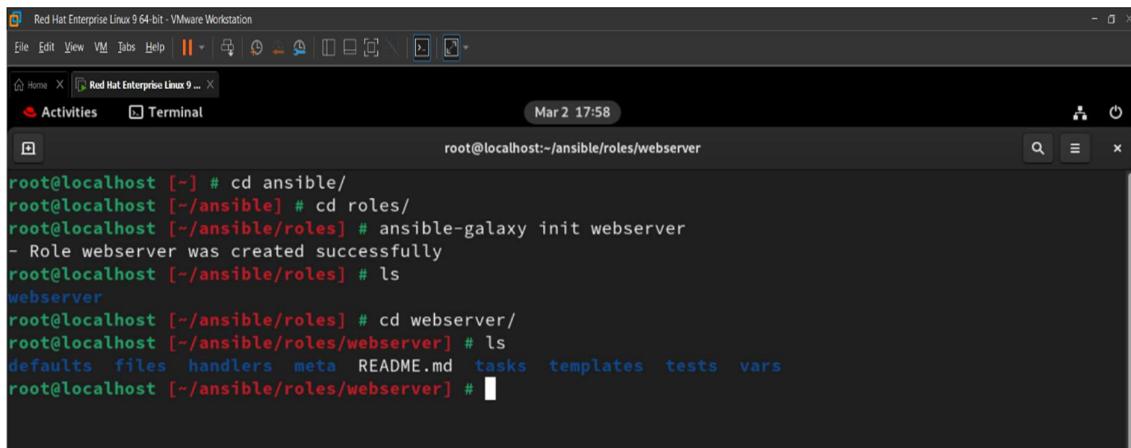


A screenshot of a terminal window titled "root@localhost:~/ansible — vim hosts". The window shows three tabs: "root@localhost:~/ansible", "root@localhost:~/ansible — vim hosts" (which is active), and "root@localhost:~/ansible/roles/webserver/defaults". The active tab displays the [webservers] group definition with two hosts: server1.com and server2.com.

Step 4: Create the Web Server Role

Generate the webserver role structure:

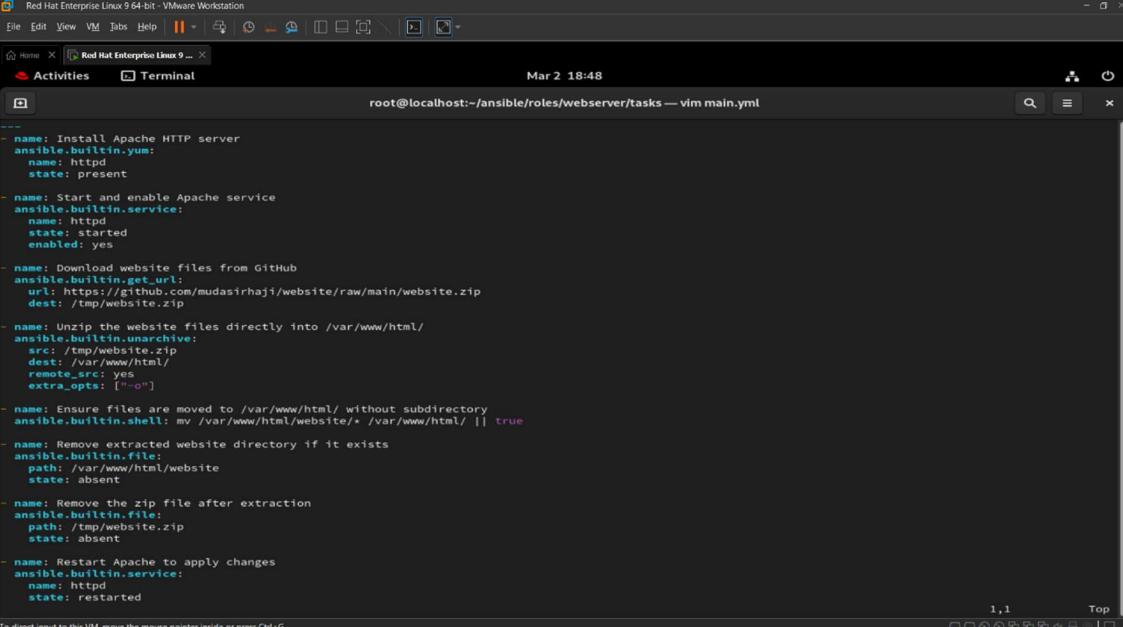
```
# ansible-galaxy init webserver
```



A screenshot of a terminal window titled "root@localhost:~/ansible/roles/webserver". The user runs the command "ansible-galaxy init webserver". The output shows that the "webserver" role was created successfully. The directory structure for the role is then listed with "ls", showing files like defaults, handlers, meta, README.md, tasks, templates, tests, and vars.

Step 5: Configure the main.yml in roles/webserver/tasks/

Create and edit main.yml inside roles/webserver/tasks/:



```
---  
- name: Install Apache HTTP server  
  ansible.builtin.yum:  
    name: httpd  
    state: present  
  
- name: Start and enable Apache service  
  ansible.builtin.service:  
    name: httpd  
    state: started  
    enabled: yes  
  
- name: Download website files from GitHub  
  ansible.builtin.get_url:  
    url: https://github.com/mudasirhaji/website/raw/main/website.zip  
    dest: /tmp/website.zip  
  
- name: Unzip the website files directly into /var/www/html/  
  ansible.builtin.unarchive:  
    src: /tmp/website.zip  
    dest: /var/www/html/  
    remote_src: yes  
    extra_opts: ["-o"]  
  
- name: Ensure files are moved to /var/www/html/ without subdirectory  
  ansible.builtin.shell: mv /var/www/html/website/* /var/www/html/ || true  
  
- name: Remove extracted website directory if it exists  
  ansible.builtin_file:  
    path: /var/www/html/website  
    state: absent  
  
- name: Remove the zip file after extraction  
  ansible.builtin_file:  
    path: /tmp/website.zip  
    state: absent  
  
- name: Restart Apache to apply changes  
  ansible.builtin.service:  
    name: httpd  
    state: restarted
```

Step 6: Configure the main.yml : roles/webserver/handlers/

Create and edit main.yml inside roles/webserver/handlers/:

```
---  
- name: Restart Apache  
  ansible.builtin.service:  
    name: httpd  
    state: restarted
```

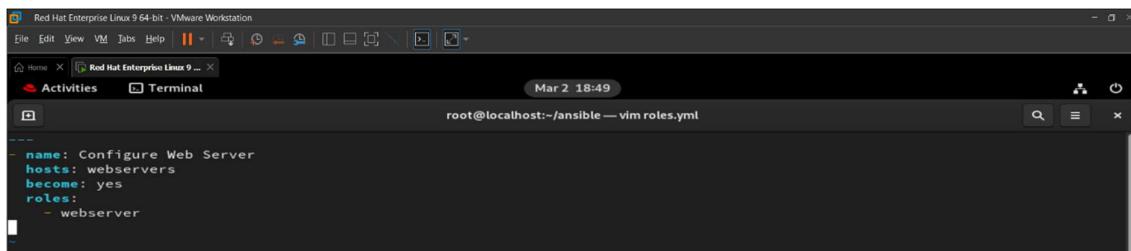


```
---  
- name: Restart Apache  
  ansible.builtin.service:  
    name: httpd  
    state: restarted
```

Step 7: Create the roles.yml Playbook

Navigate to the Ansible directory and create roles.yml with the following content:

```
---
- name: Configure Web Server
  hosts: webservers
  become: yes
  roles:
    - webserver
```



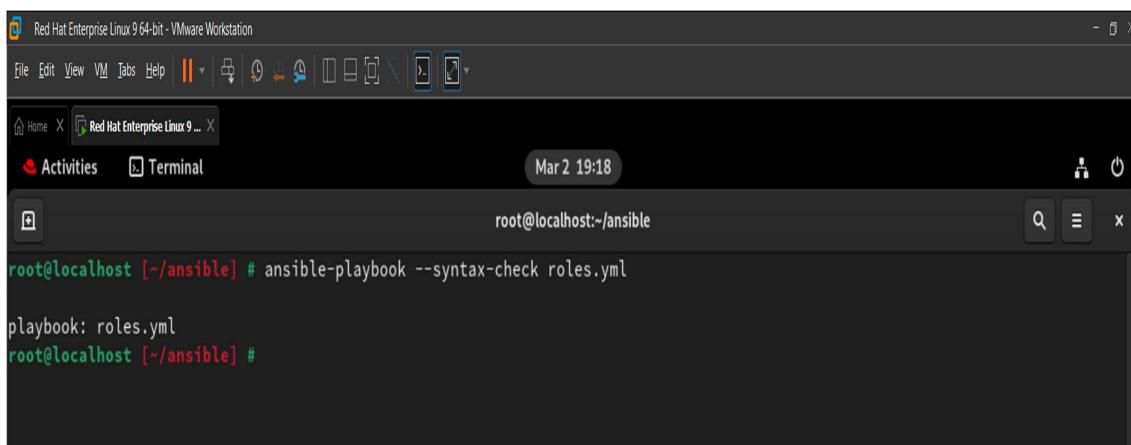
A screenshot of a terminal window titled "Red Hat Enterprise Linux 9 64-bit - VMware Workstation". The window shows the command "root@localhost:~/ansible — vim roles.yml" at the top. The terminal content displays the YAML code for the playbook.

Step 8: Run the Playbook

Syntax Check

Before executing the playbook, verify the syntax:

```
# ansible-playbook --syntax-check roles.yml
```

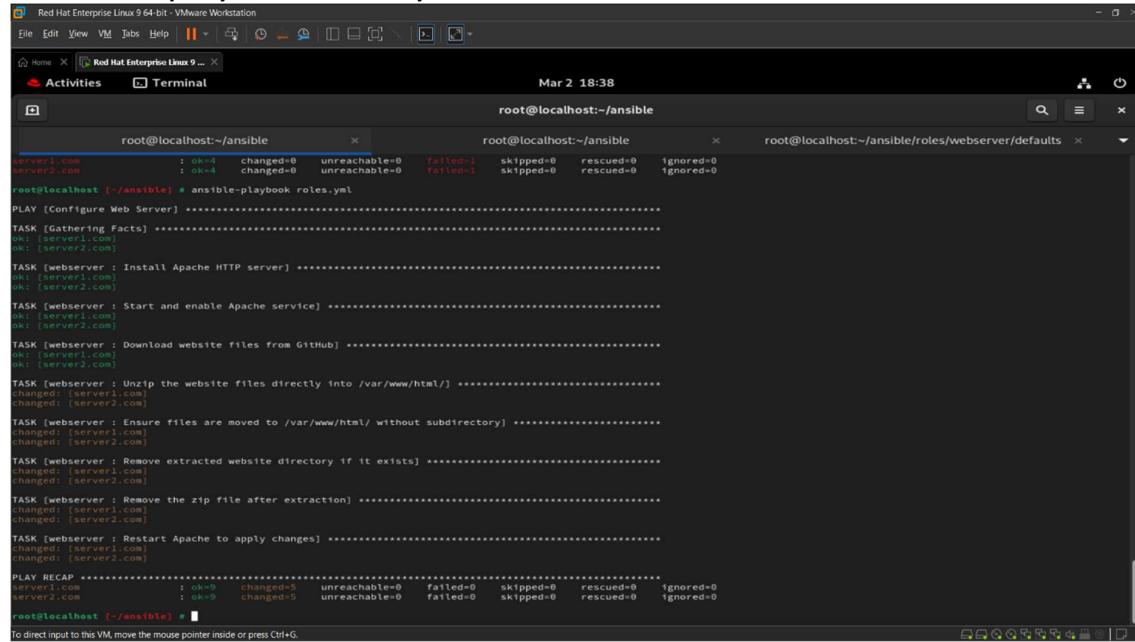


A screenshot of a terminal window titled "Red Hat Enterprise Linux 9 64-bit - VMware Workstation". The window shows the command "root@localhost [~/ansible] # ansible-playbook --syntax-check roles.yml" at the top. The terminal content displays the output of the syntax check command, which is empty, indicating no syntax errors.

Step 9: Run the Playbook

Deploy the website on the EC2 instances:

```
# ansible-playbook roles.yml
```



```
root@localhost:~/ansible          root@localhost:~/ansible          root@localhost:~/ansible/roles/webserver/defaults
server1.com : ok=4     changed=0    unreachable=0    failed=1    skipped=0    rescued=0    ignored=0
server2.com : ok=4     changed=0    unreachable=0    failed=1    skipped=0    rescued=0    ignored=0
root@localhost [-/ansible] # ansible-playbook roles.yml

PLAY [Configure Web Server] *****
TASK [Gathering Facts] *****
ok: [server1.com]
ok: [server2.com]

TASK [webservice : Install Apache HTTP server] *****
ok: [server1.com]
ok: [server2.com]

TASK [webservice : Start and enable Apache service] *****
ok: [server1.com]
ok: [server2.com]

TASK [webservice : Download website files from GitHub] *****
ok: [server1.com]
ok: [server2.com]

TASK [webservice : Unzip the website files directly into /var/www/html/] *****
changed: [server1.com]
changed: [server2.com]

TASK [webservice : Ensure files are moved to /var/www/html/ without subdirectory] *****
changed: [server1.com]
changed: [server2.com]

TASK [webservice : Remove extracted website directory if it exists] *****
changed: [server1.com]
changed: [server2.com]

TASK [webservice : Remove the zip file after extraction] *****
changed: [server1.com]
changed: [server2.com]

TASK [webservice : Restart Apache to apply changes] *****
changed: [server1.com]
changed: [server2.com]

PLAY RECAP *****
server1.com : ok=9    changed=5    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
server2.com : ok=9    changed=5    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
root@localhost [-/ansible] #
```

Conclusion

This setup ensures Apache is installed, running, and serving website files on multiple EC2 instances automatically using Ansible. 

The website is live on both hosts:

Server 1: <http://15.207.110.253/>

Server 2: <http://3.108.215.38/>

