

ANSIBLE CONFIGURATION MANAGEMENT – PROJECT AUTOMATION

This Project will enhance our appreciation of DevOps tools even more because it will enable us to automate most of the routine tasks with Ansible Configuration Management. At the same time, we will become confident at writing codes using declarative language such as YAML.

We will develop Ansible scripts to simulate the use of a Jump box/Bastion host to access our Web Servers.

Tasks

1. **Install and configure Ansible client to act as a Jump Server/Bastion Host** – (A Jump Server, sometimes also referred as Bastion Host, is an intermediary server through which access to internal network can be provided.

In Ansible configuration management, a jump server or bastion host is a server that acts as an intermediary or bridge between the Ansible control machine and the target hosts. It provides a secure access point to reach and manage hosts in a remote network or environment.

When using Ansible to manage systems in a different network or with restricted access, you can configure a jump server to establish a secure connection. The Ansible control machine connects to the jump server, and then the jump server allows access to the target hosts. This setup adds an additional layer of security by limiting direct access to the target hosts from the control machine.

By using a jump server or bastion host, you can enforce security policies, control access to remote hosts, and manage network isolation. It's a common practice in scenarios where direct connectivity to the target hosts is restricted or undesirable.

If you think about the current architecture we have been working on, ideally, the web servers would be inside a secured network which cannot be reached directly from the Internet. That means, even other DevOps engineers cannot **SSH** into the Web servers directly and can only access it through a **Jump Server** – it provides for better security and reduces attack surface.)

2. **Create a simple Ansible playbook to automate server's configuration.**

The workflow below describes how you can automate the deployment process using Jenkins and Ansible, version control your code with GitHub, and deploy to different environments with specific configurations using Ansible's flexibility.

Set up your environments:

Development Environment: This is where you develop and test your code changes. It typically includes a local machine or a dedicated development server.

Staging Environment: This is a replica of your production environment where you can perform additional testing before deploying changes to production.

Production Environment: This is the live environment where your applications are running and serving end-users.

Version Control with GitHub:

Create a repository on GitHub to store your codebase.

Clone the repository to your development environment using Git.

Make changes to your code, commit them, and push them back to the repository.

Configuration Management with Ansible:

Install Ansible on your development environment.

Create an Ansible playbook to define the desired state of your infrastructure and applications.

Configure inventory files to define the target hosts for deployment in different environments (development, staging, production).

Define variables and group variables to manage environment-specific configurations.

Continuous Integration with Jenkins:

Install and configure Jenkins on a dedicated server or in your development environment. *Don't forget to open port 8080 on the security inbound rules.*

Create a Jenkins job or pipeline to automate the build, test, and deployment process.

Configure the Jenkins job to listen for changes in the GitHub repository.

Set up build triggers to automatically trigger the Jenkins job on code commits or pull requests.

Deploying to Different Environments:

Set up Jenkins to trigger the deployment process based on the appropriate environment (development, staging, production).

Configure the Jenkins job to run the Ansible playbook using the appropriate inventory and variables for the target environment.

Use Ansible's conditional statements or variable handling to differentiate between environments and apply specific configurations accordingly.

Testing and Validation:

Define tests to ensure the desired state of your infrastructure and applications.
Integrate these tests into your Jenkins job or pipeline to run after the deployment process.

Use Ansible's test modules or custom scripts to validate the deployment and report any issues.

STEP 1. INSTALL AND CONFIGURE ANSIBLE ON EC2 INSTANCE

1. We will update Name tag on our Jenkins EC2 Instance in previous project 9 to **Jenkins-Ansible**. We will use this **Jenkins-Ansible** server to run our playbooks.

2. In our GitHub account we'll create a new repository and name it **ansible-config-mgt**.

3. Install Ansible

```
$ sudo apt update
```

```
$ sudo apt install ansible
```

Check your Ansible version by running the command below:

```
$ ansible --version
```

```
ubuntu@ip-172-31-37-181:~$ ansible --version
ansible 2.9.6
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/ubuntu/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  executable location = /usr/bin/ansible
  python version = 3.8.10 (default, May 26 2023, 14:05:08) [GCC 9.4.0]
ubuntu@ip-172-31-37-181:~$
```

4. Configure Jenkins build job to save your repository content every time you change it.

Create a new **Freestyle project** called “ansible” in Jenkins and point it to your ‘ansible-config-mgt’ repository.

Configure Webhook in GitHub and set webhook to trigger ansible build.

Configure a Post-build job to save all (**) files, like you did it in Project 9.

ansible Config [Jenkins] x +

Not secure | http://13.41.23.124:8080/job/ansible/configure

Dashboard > ansible > Configuration

Configure

- General
- Source Code Management
- Build Triggers
- Build Environment
- Build Steps
- Post-build Actions

Project url ?

https://github.com/obaigbenaa/ansible-config-mgt/

Advanced ▾

☐ This project is parameterised ?

☐ Throttle builds ?

☐ Execute concurrent builds if necessary ?

Advanced ▾

Source Code Management

☐ None

☒ Git ?

Repositories ?

Repository URL ?

https://github.com/obaigbenaa/ansible-config-mgt.git

Credentials ?

- none - ▾

Add +

Save Apply

ansible Config [Jenkins] x +

Not secure | http://13.41.23.124:8080/job/ansible/configure

Dashboard > ansible > Configuration

Configure

- General
- Source Code Management
- Build Triggers
- Build Environment
- Build Steps
- Post-build Actions

Branch Specifier (blank for 'any') ?

*/main

Add Branch

Repository browser ?

(Auto)

Additional Behaviours

Add ▾

Build Triggers

☐ Trigger builds remotely (e.g., from scripts) ?

☐ Build after other projects are built ?

☐ Build periodically ?

☒ GitHub hook trigger for GITScm polling ?

☐ Poll SCM ?

Build Environment

Save Apply

ansible Config [Jenkins] x +

Not secure | http://13.41.23.124:8080/job/ansible/configure

Dashboard > ansible > Configuration

Configure

- General
- Source Code Management
- Build Triggers
- Build Environment
- Build Steps
- Post-build Actions

☐ Add timestamps to the Console Output

☐ Inspect build log for published build scans

☐ Terminate a build if it's stuck

☐ With Ant ?

Build Steps

Add build step ▾

Post-build Actions

Archive the artifacts ?

Files to archive ?

**

Advanced ▾

Add post-build action ▾

Save Apply

5. Test your setup by making some change in readme "Ansible.md" file in /main branch and make sure that builds starts automatically, and Jenkins saves the files (build artifacts) in following folder

Jenkins

Dashboard > ansible > #3 > Console Output

Status

Changes

Console Output

View as plain text

Edit Build Information

Delete build '#3'

Polling Log

Git Build Data

Previous Build

Next Build

Console Output

Started by GitHub push by [redacted]
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/ansible
The recommended git tool is: NONE
No credentials specified
> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/ansible/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/obaigbenaa/ansible-config-mgt.git # timeout=10
Fetching upstream changes from https://github.com/obaigbenaa/ansible-config-mgt.git
> git --version # timeout=10
> git --version # 'git version 2.25.1'
> git fetch --tags --force --progress -- https://github.com/obaigbenaa/ansible-config-mgt.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision d0557dc056132610b5a186961b15516be8908e06 (refs/remotes/origin/main)
> git config core.sparsecheckout # timeout=10
> git checkout -f d0557dc056132610b5a186961b15516be8908e06 # timeout=10
Commit message: "Update Ansible.md"
First time build. Skipping changelog.
Archiving artifacts
Finished: SUCCESS

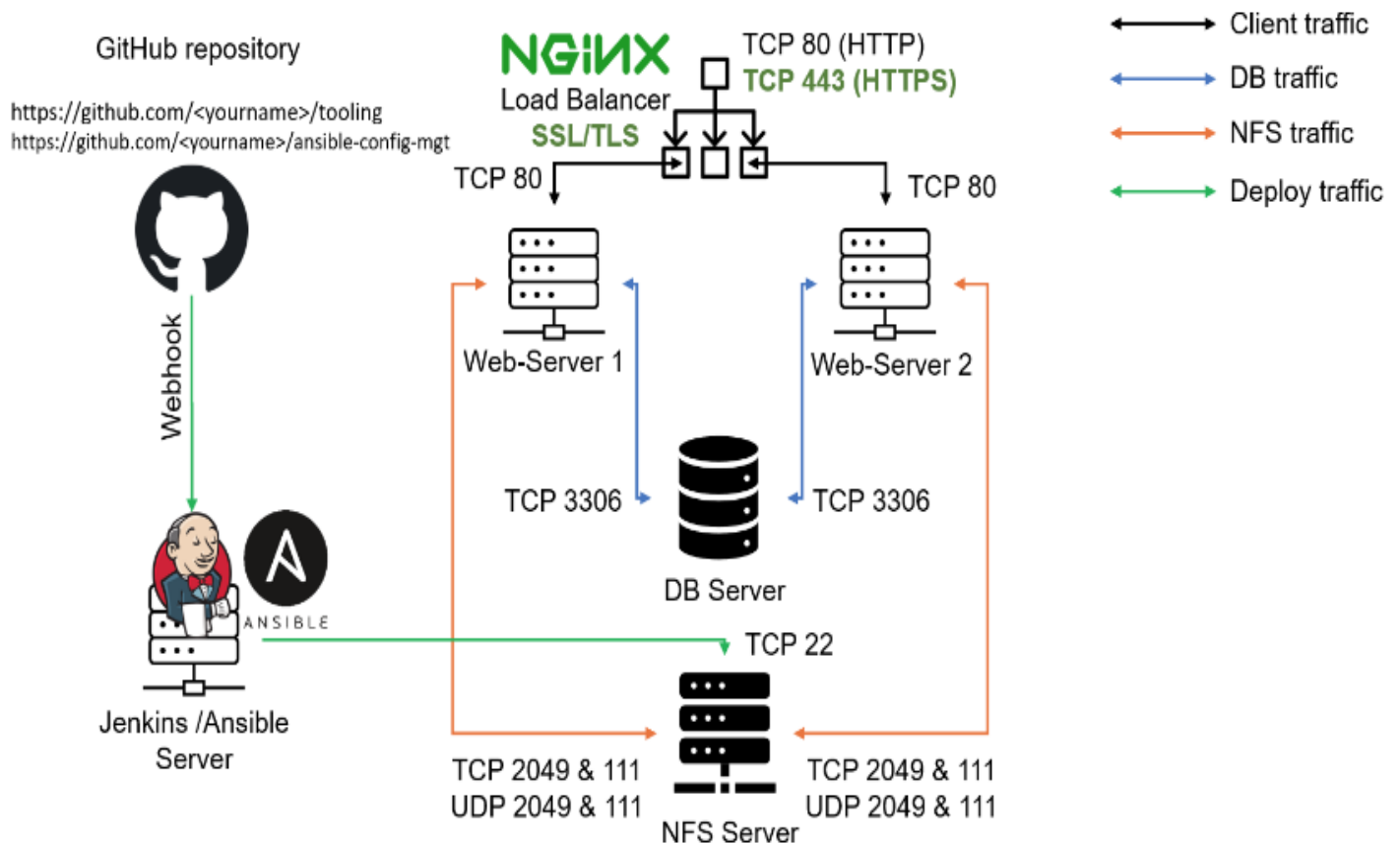
You will find the readme file named "ansible.md" in this path.

```
$ ls /var/lib/jenkins/jobs/ansible/builds/3/archive/
```

```
ubuntu@ip-172-31-37-181:~$ ls /var/lib/jenkins/jobs/ansible/builds/
2 3 4 legacyIds permalinks
ubuntu@ip-172-31-37-181:~$ ls /var/lib/jenkins/jobs/ansible/builds/3
archive build.xml changelog.xml log polling.log
ubuntu@ip-172-31-37-181:~$ ls /var/lib/jenkins/jobs/ansible/builds/3/archive/
Ansible.md
ubuntu@ip-172-31-37-181:~$
```

Note that We will only trigger Jenkins project execution only for /main (master) branch.

Now our setup will look like this below:



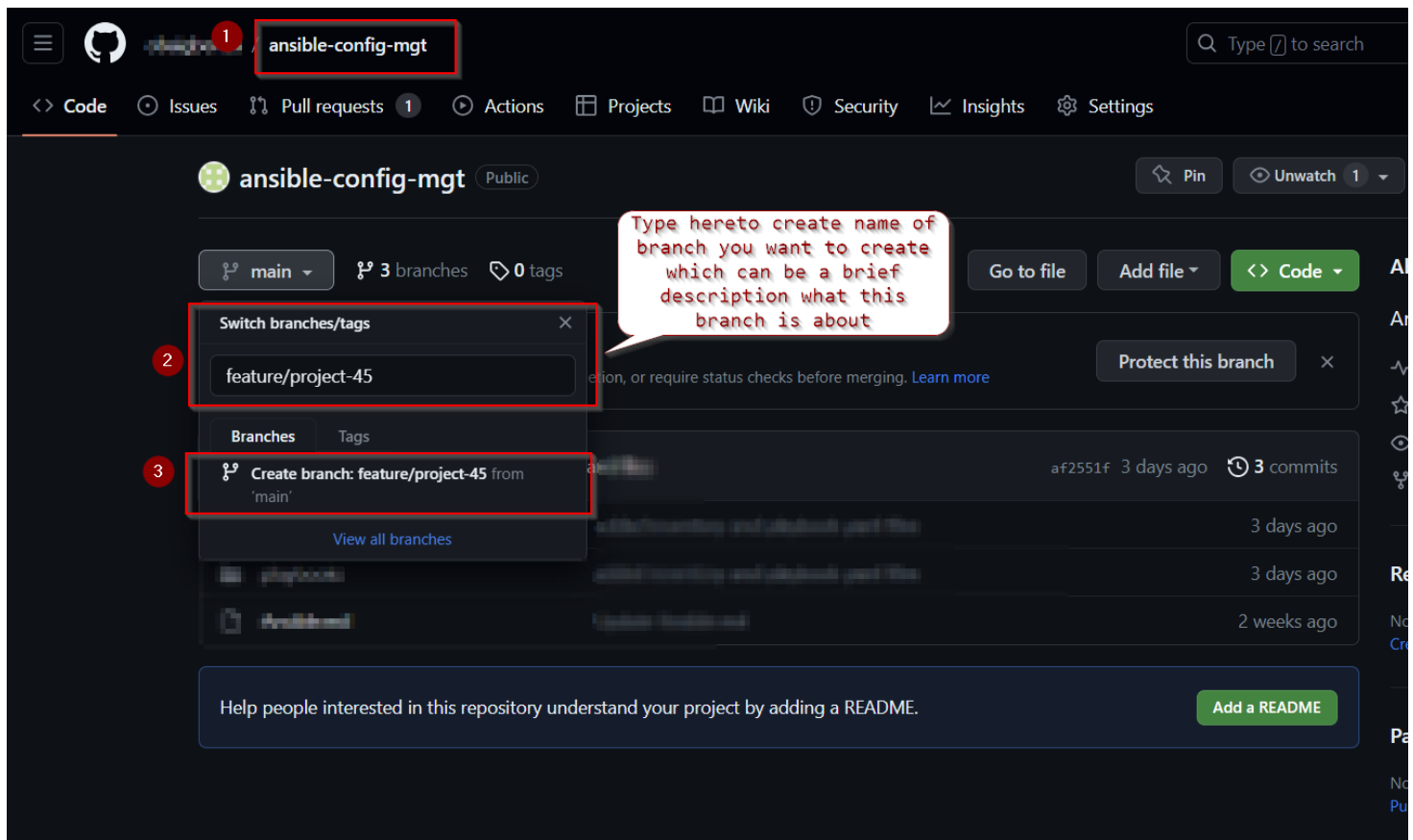
As we have learned in our previous project, it makes sense to allocate an **Elastic IP** to our **Jenkins-Ansible** server because every time you stop/start your Jenkins-Ansible server - you have to reconfigure GitHub webhook to a new IP address. This avoids the repetitive change of public IP addresses and GitHub reconfiguration.

STEP 2. PREPARE YOUR DEVELOPMENT ENVIRONMENT USING VISUAL STUDIO CODE

1. We will require to write some codes and we need to have proper tools that will make our coding and debugging comfortable - we need a **Source-code Editor**. It is recommended to use one free and universal editor that will fully satisfy your needs - **Visual Studio Code (VSC)**
2. After you have successfully installed VSC, configure it to newly created GitHub repository 'ansible-config-mgt'.

STEP 3. BEGIN ANSIBLE DEVELOPMENT

1. In your ansible-config-mgt GitHub repository, create a new branch that will be used for development of a new feature.



2. Checkout the newly created feature branch to your local machine and start building your code and directory structure.

```
$ git checkout -b feature/proj-45
```

Running this command on your VSC (ansible-config-mgt directory) should create a branch named feature/proj-45 and move into the branch automatically for you to start building our code and directory structure.

3. Create a directory and name it *playbooks* - it will be used to store all your playbook files.

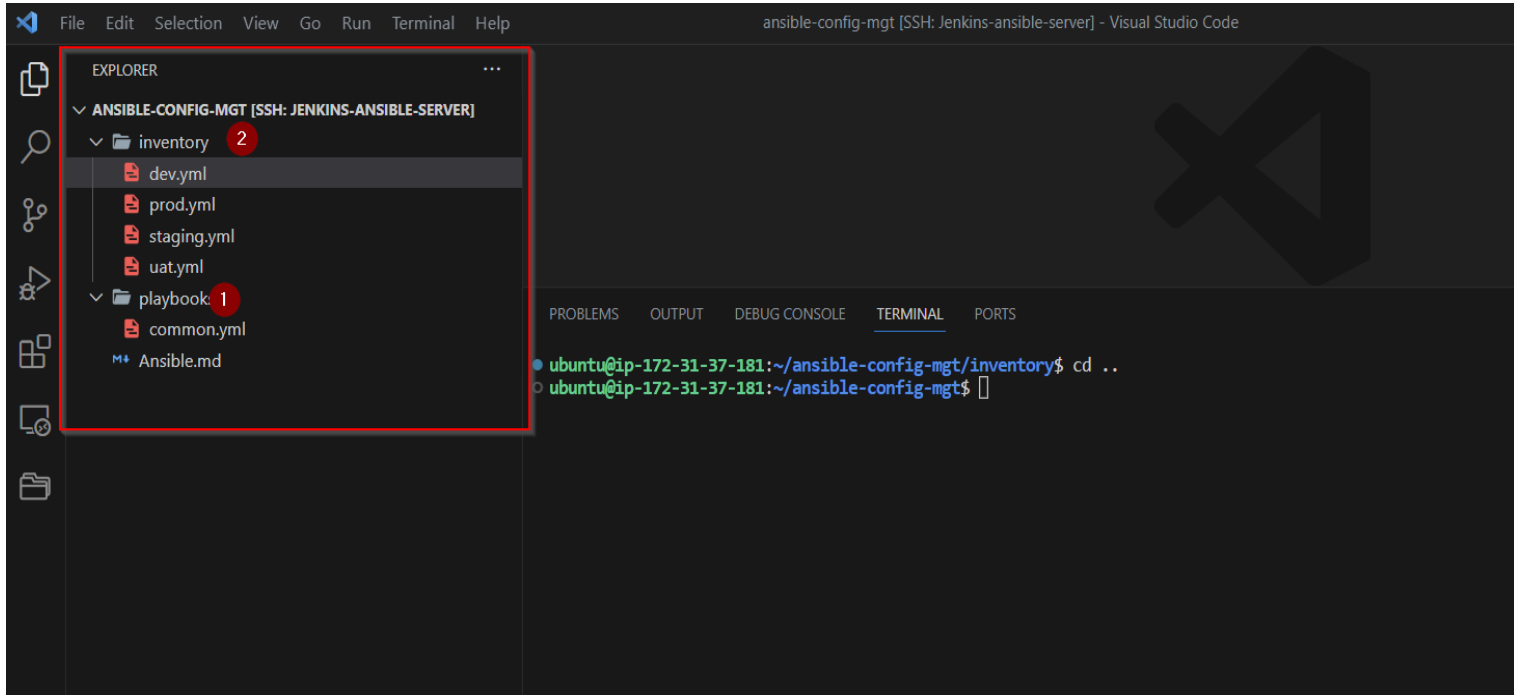
```
$ mkdir playbooks
```

4. Create a directory and name it *inventory* - it will be used to keep your hosts organised.

```
$ mkdir inventory
```

5. Within the playbooks folder, create your first playbook, and name it *common.yml*

6. Within the inventory folder, create an inventory file (.yml) for each environment (Development, Staging Testing and Production) *dev*, *staging*, *uat*, and *prod* respectively.
- 7.



STEP 4. SET UP AN ANSIBLE INVENTORY

An Ansible inventory file is a text file that contains a list of hosts and groups that Ansible can connect to and manage. The inventory file is used by Ansible to determine which hosts to target and perform tasks on. In other words, an Ansible inventory defines the hosts and groups of hosts upon which commands, modules, and tasks in a playbook operate.

Since our intention is to execute Linux commands on remote hosts and ensure that it is the intended configuration on a particular server that occurs. It is important to have a way to organize our hosts in such an Inventory.

Save/update below inventory structure in the *inventory/dev* file to start configuring your development servers. Ensure to replace the IP addresses according to your own setup.

```
$ sudo vi inventory/dev.yml
```

```
[nfs]
```

```
<NFS-Server-Private-IP-Address> ansible_ssh_user='ec2-user'
```

```
[webservers]
```

```
<Web-Server1-Private-IP-Address> ansible_ssh_user='ec2-user'
```

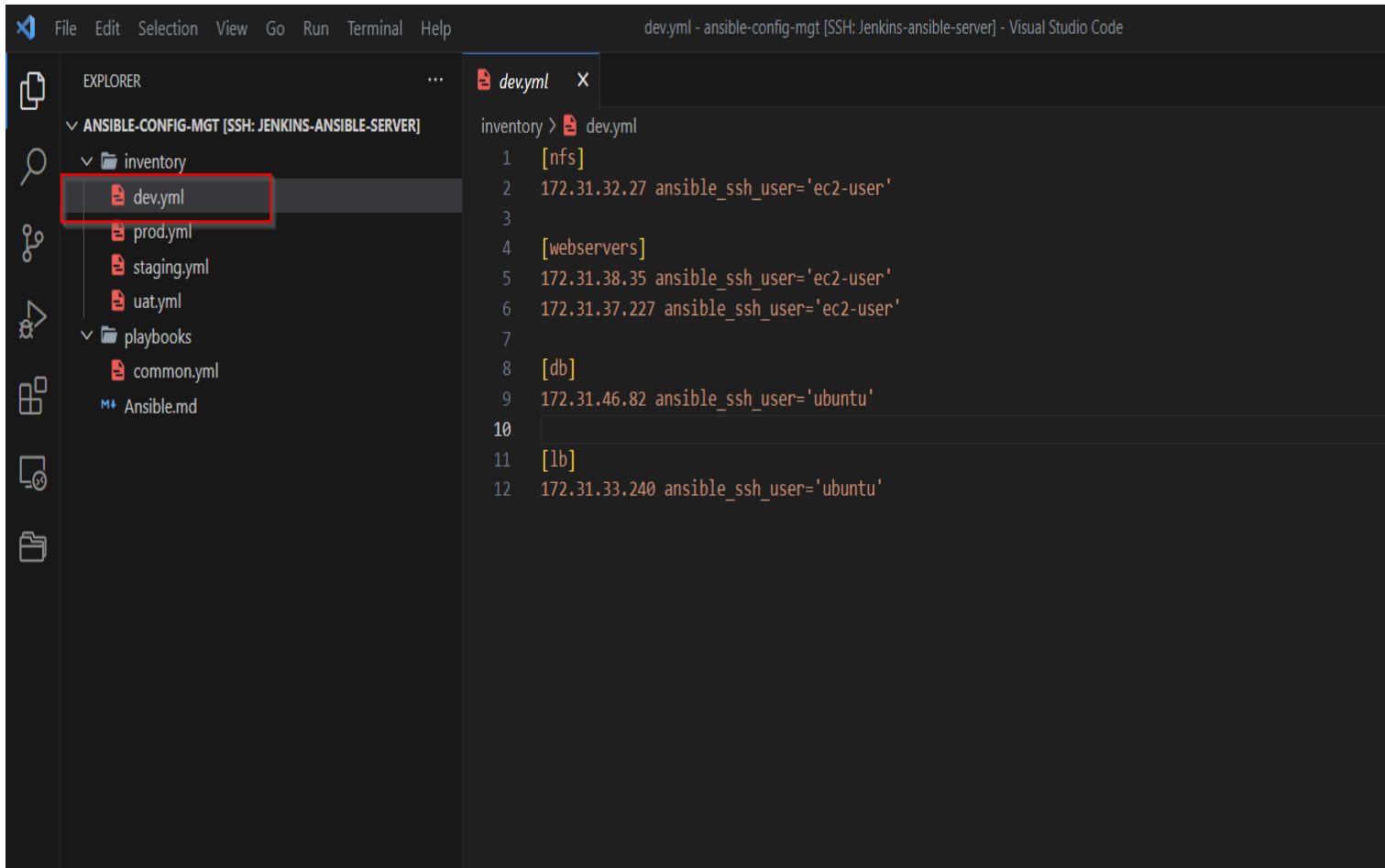
```
<Web-Server2-Private-IP-Address> ansible_ssh_user='ec2-user'
```

```
[db]
```

```
<Database-Private-IP-Address> ansible_ssh_user='ubuntu'
```

```
[lb]
```

```
<Load-Balancer-Private-IP-Address> ansible_ssh_user='ubuntu'
```



Note: Ansible uses TCP port 22 by default, which means it needs to **SSH** into target servers from Jenkins-Ansible host – for this you we should implement the concept of ssh-agent. Now we need to import your key into ssh-agent:

We will go to our local terminal to connect our Jenkins-Ansible public key to the authorised key files of the hosts.

Generate an **SSH key pair** using the ssh-keygen utility.

Open a terminal or command prompt on your local machine.

You will be prompted to specify the file name and location to save the key pair. Press Enter to accept the default location (/home/your_username/.ssh/id_rsa), or specify a different path and file name if desired.

Next, you will be prompted to enter a passphrase for the key. A passphrase adds an extra layer of security to the private key. You can choose to enter a passphrase or press Enter to leave it empty.

```

PS C:\Users\aoabai\Downloads> ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (C:\Users\aoabai/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in C:\Users\aoabai/.ssh/id_rsa
Your public key has been saved in C:\Users\aoabai/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:cDkW0jgSPE1YxzNWRrZSwjU0VrWjitM55Rhh2KH3U6Q aoabai@Alexandero
The key's randomart image is:
+---[RSA 3072]-----+
|  ..*+=+=&...o |
|   = +oB% = o . |
|  o.o0o* E +    |
|   + = o o .    |
|   S . =        |
|   o B .        |
|   o * .        |
|   . .         |
+----[SHA256]-----+
PS C:\Users\aoabai\Downloads> 

```

```

PS C:\Users\aoabai\Downloads> cd ..
PS C:\Users\aoabai> cd .\.ssh\
PS C:\Users\aoabai\.ssh> ls

```

Directory: C:\Users\aoabai\.ssh

Mode	LastWriteTime	
----	-----	-----
-a----	25/06/2023	19:02
-a----	28/04/2023	13:40
-a----	28/04/2023	13:40
-a----	05/07/2023	15:15
-a----	05/07/2023	15:15
-a----	25/06/2023	20:59
-a----	25/06/2023	20:58

Length	Name
-----	-----
259	config
464	id_ed25519
99	id_ed25519.pub
2602	id_rsa
571	id_rsa.pub
33785	known_hosts
33169	known_hosts.old

Now you have generated an SSH key pair. The private key (id_rsa) should be kept secure and not shared, while the public key (id_rsa.pub) can be added to the `~/.ssh/authorized_keys` file on remote servers you want to authenticate with using this key pair.

cd into the `.ssh/` directory of the host/target server and target the `authorized_keys` file is in order connect.

```
$ sudo vi /.ssh/authorized_keys
```

`$ cat id_rsa.pub` and add the content of the `id_rsa.pub` file into this `authorized_keys` and save.

```
esc + :wq!
```


STEP 5. CREATE A COMMON PLAYBOOK

We will now start giving Ansible the instructions on what it needs to perform on all servers listed in *inventory/dev.yml*

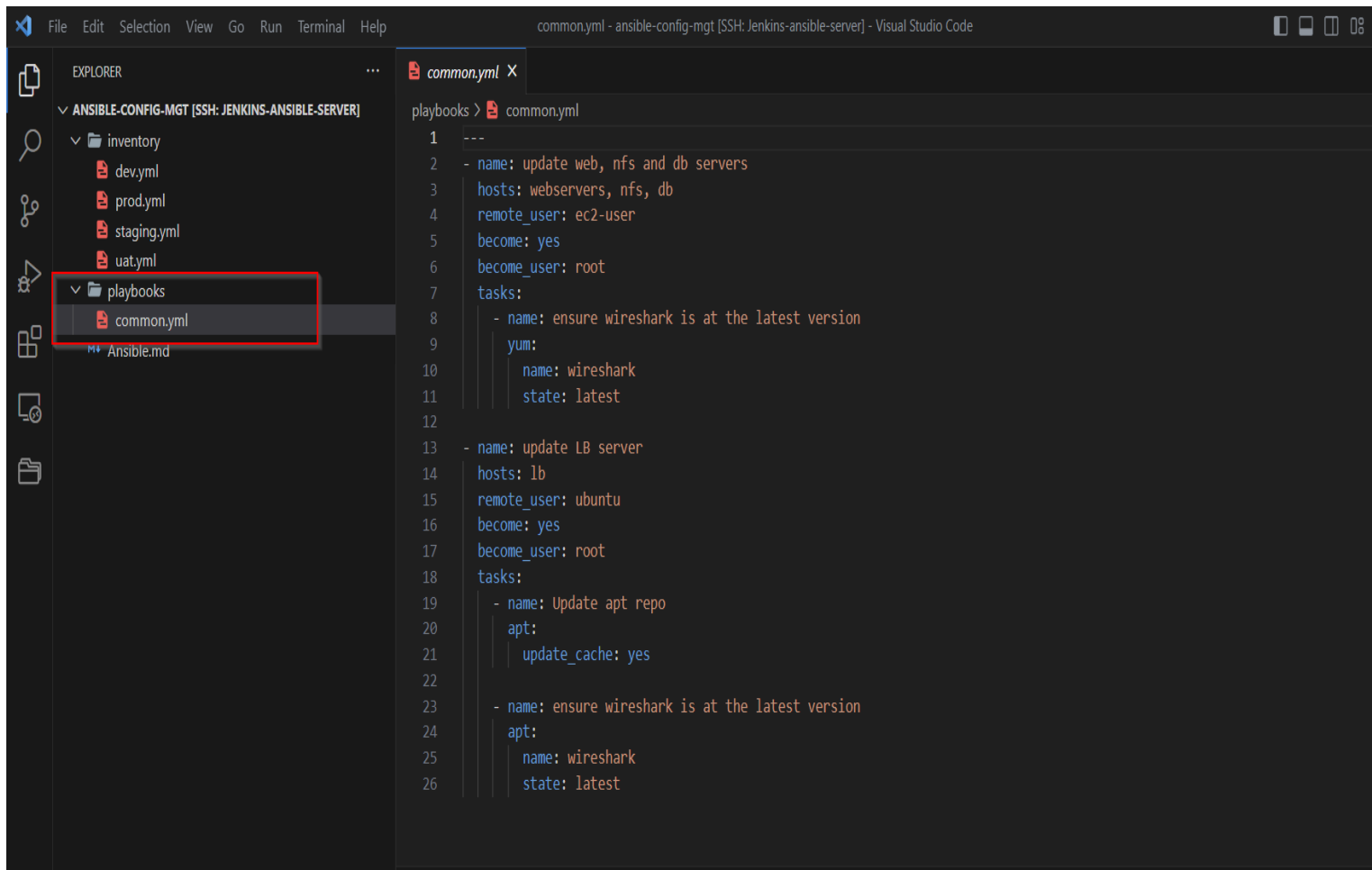
In *common.yml* playbook we will write configuration for repeatable, re-usable, and multi-machine tasks that is common to systems within the infrastructure.

Let's update our *playbook/common.yml* file with the following code

```
---
- name: update web, nfs and db servers
  hosts: webserver, nfs, db
  remote_user: ec2-user
  become: yes
  become_user: root
  tasks:
    - name: ensure wireshark is at the latest version
      yum:
        name: wireshark
        state: latest

- name: update LB server
  hosts: lb
  remote_user: ubuntu
  become: yes
  become_user: root
  tasks:
    - name: Update apt repo
      apt:
        update_cache: yes

    - name: ensure wireshark is at the latest version
      apt:
        name: wireshark
        state: latest
```



```
1 ---
2 - name: update web, nfs and db servers
3   hosts: webserver, nfs, db
4   remote_user: ec2-user
5   become: yes
6   become_user: root
7   tasks:
8     - name: ensure wireshark is at the latest version
9       yum:
10         name: wireshark
11         state: latest
12
13 - name: update LB server
14   hosts: lb
15   remote_user: ubuntu
16   become: yes
17   become_user: root
18   tasks:
19     - name: Update apt repo
20       apt:
21         update_cache: yes
22
23     - name: ensure wireshark is at the latest version
24       apt:
25         name: wireshark
26         state: latest
```

In the code above, the playbook is divided into two parts, each of them is intended to perform the same task: install Wireshark utility (or make sure it is updated to the latest version) on your RHEL 8 and Ubuntu servers. It uses root user to perform this task and respective package manager: yum for RHEL 8 and apt for Ubuntu.

STEP 6. UPDATE GIT WITH THE LATEST CODE

We need to push changes made locally on our machine to GitHub. Because all our file and directories are located locally.

We may already be working within a team of other DevOps engineers and developers. It is important to learn how to collaborate with the help of GIT.

Now we have a separate branch, we will need to know how to raise a **Pull Request (PR)**, get our branch peer reviewed and merged to the **/main or master** branch.

Commit your code into GitHub:

1. Use git commands to **add**, **commit** and **push** your branch to GitHub.

\$ git status -this shows us what branch we are currently working on.

In this case, we are currently on our feature/project-45 branch.

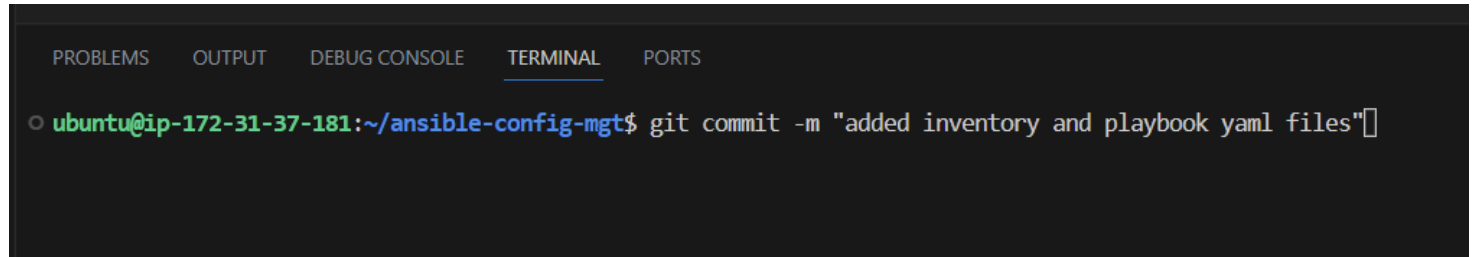
```
$ git add <selected files>
```

```
$ git add inventory/
```

```
$ git add playbooks/
```

```
$ git commit -m "commit message"
```

Commit message can be a short phrase. For example, to indicate what has been added.

A screenshot of a terminal window with a dark background. At the top, there are tabs labeled 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is selected and underlined), and 'PORTS'. Below the tabs, the terminal shows a prompt 'ubuntu@ip-172-31-37-181:~/ansible-config-mgt\$' followed by the command 'git commit -m "added inventory and playbook yaml files"' and a cursor at the end of the line.

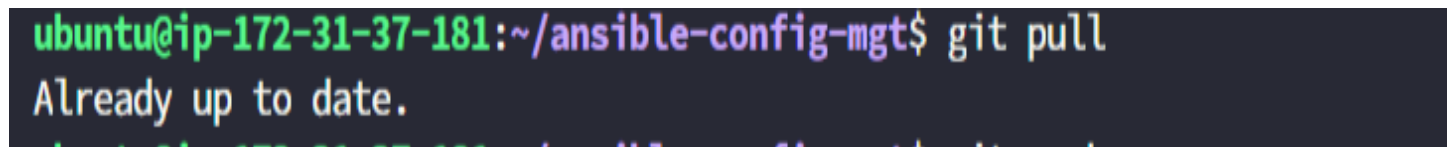
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
ubuntu@ip-172-31-37-181:~/ansible-config-mgt$ git commit -m "added inventory and playbook yaml files"
```

2. Create a Pull request (PR)

Merge the code to the master branch.

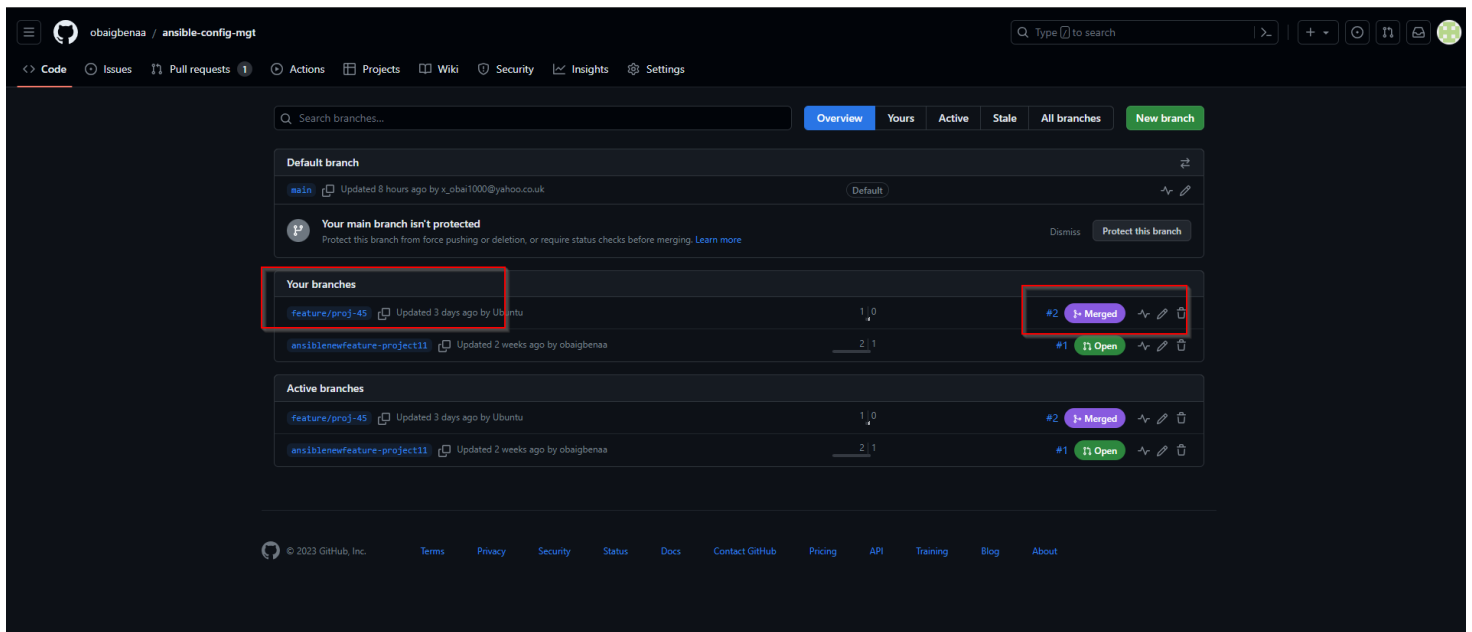
```
$ git push origin
```

Head back on your terminal, checkout from the feature branch into the main, and pull down the latest changes.

A screenshot of a terminal window with a dark background. It shows a prompt 'ubuntu@ip-172-31-37-181:~/ansible-config-mgt\$' followed by the command 'git pull'. Below the command, the text 'Already up to date.' is displayed.

```
ubuntu@ip-172-31-37-181:~/ansible-config-mgt$ git pull
Already up to date.
```


See in image below that the feature/proj-45 branch has been merged with main branch.






Once your code changes appear in master branch – Jenkins will do its job and save all the files (build artifacts) to
/var/lib/jenkins/jobs/ansible/builds/<build_number>/archive/ directory on **Jenkins-Ansible** server.


```
$ cd /var/lib/jenkins/jobs/ansible/builds
```


```
ubuntu@ip-172-31-37-181:~/ansible-config-mgt$ cd /var/lib/jenkins/jobs/ansible/  
ubuntu@ip-172-31-37-181:/var/lib/jenkins/jobs/ansible$ la  
builds  config.xml  github-polling.log  nextBuildNumber  
ubuntu@ip-172-31-37-181:/var/lib/jenkins/jobs/ansible$ cd builds/  
ubuntu@ip-172-31-37-181:/var/lib/jenkins/jobs/ansible/builds$ ls  
2 3 4 5 legacyIds  permalinks  
ubuntu@ip-172-31-37-181:/var/lib/jenkins/jobs/ansible/builds$
```



 **Jenkins**


Search (CTRL+K) ?  1  Alexandero 


Dashboard > **ansible > #4** > Console Output


 Status


 Changes


 **Console Output**


 View as plain text


 Edit Build Information


 Delete build '#4'


 Polling Log

 Git Build Data

 Previous Build

 Next Build

 **Console Output**

Started by GitHub push by 

Running as SYSTEM

Building in workspace /var/lib/jenkins/workspace/ansible

The recommended git tool is: NONE

No credentials specified

> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/ansible/.git # timeout=10

Fetching changes from the remote Git repository

> git config remote.origin.url https://github.com/obaigbenaa/ansible-config-mgt.git # timeout=10

Fetching upstream changes from https://github.com/obaigbenaa/ansible-config-mgt.git

> git --version # timeout=10

> git --version # 'git version 2.25.1'

> git fetch --tags --force --progress -- https://github.com/obaigbenaa/ansible-config-mgt.git /origin/* # timeout=10

> git rev-parse refs/remotes/origin/main^{commit} # timeout=10

Checking out Revision af2551fd87961a24a75e23f254ced69579441c30 (refs/remotes/origin/main)

> git config core.sparsecheckout # timeout=10

> git checkout -f af2551fd87961a24a75e23f254ced69579441c30 # timeout=10

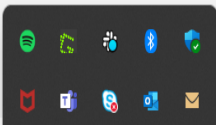
Commit message: "added inventory and playbook yaml files"

> git rev-list --no-walk d0557dc096132610b5a186961b15510be8908e06 # timeout=10

Archiving artifacts

Finished: SUCCESS

we are now in the origin/main branch



STEP 7. RUN FIRST ANSIBLE TEST

Step 7 – Run first Ansible test

Now, it is time to execute ansible-playbook command and verify if your playbook actually works.

```
$ cd ansible-config-mgt
```

```
$ ansible-playbook -i inventory/dev.yml playbooks/common.yml
```

Go to each of the servers and check if wireshark has been installed by running

```
$ which wireshark or
```

```
$ wireshark --version
```

```
[ec2-user@ip-172-31-32-27 ~]$ wireshark --version
Wireshark 2.6.2 (v2.6.2)
```

wireshark
installed on NFS
server

Copyright 1998-2018 Gerald Combs <gerald@wireshark.org> and contributors.
License GPLv2+: GNU GPL version 2 or later <<http://www.gnu.org/licenses/old-licenses/gpl-2.0.html>>
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Compiled (64-bit) with Qt 5.15.3, with libpcap, with POSIX capabilities (Linux),
with libnl 3, with GLib 2.56.4, with zlib 1.2.11, with SMI 0.4.8, with c-ares
1.13.0, without Lua, with GnuTLS 3.6.16, with Gcrypt 1.8.5, with MIT Kerberos,
with MaxMind DB resolver, without nghttp2, without LZ4, without Snappy, without
libxml2, with QtMultimedia, without SBC, without SpanDSP, without bcg729.

Running on Linux 4.18.0-477.13.1.el8_8.x86_64, with Intel(R) Xeon(R) CPU E5-2676
v3 @ 2.40GHz (with SSE4.2), with 767 MB of physical memory, with locale
en_US.UTF-8, with libpcap version 1.9.1 (with TPACKET_V3), with GnuTLS 3.6.16,
with Gcrypt 1.8.5, with zlib 1.2.11, binary plugins supported (0 loaded).

Built using gcc 8.5.0 20210514 (Red Hat 8.5.0-15).
[ec2-user@ip-172-31-32-27 ~]\$

```
[ec2-user@ip-172-31-38-35 ~]$ wireshark --version
Wireshark 2.6.2 (v2.6.2)
```

wireshark
installed on
webserver

Copyright 1998-2018 Gerald Combs <gerald@wireshark.org> and contributors.
License GPLv2+: GNU GPL version 2 or later <<http://www.gnu.org/licenses/old-licenses/gpl-2.0.html>>
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Compiled (64-bit) with Qt 5.15.3, with libpcap, with POSIX capabilities (Linux),
with libnl 3, with GLib 2.56.4, with zlib 1.2.11, with SMI 0.4.8, with c-ares
1.13.0, without Lua, with GnuTLS 3.6.16, with Gcrypt 1.8.5, with MIT Kerberos,
with MaxMind DB resolver, without nghttp2, without LZ4, without Snappy, without
libxml2, with QtMultimedia, without SBC, without SpanDSP, without bcg729.

Running on Linux 4.18.0-477.13.1.el8_8.x86_64, with Intel(R) Xeon(R) CPU E5-2676
v3 @ 2.40GHz (with SSE4.2), with 767 MB of physical memory, with locale
en_US.UTF-8, with libpcap version 1.9.1 (with TPACKET_V3), with GnuTLS 3.6.16,
with Gcrypt 1.8.5, with zlib 1.2.11, binary plugins supported (0 loaded).

Built using gcc 8.5.0 20210514 (Red Hat 8.5.0-15).
[ec2-user@ip-172-31-38-35 ~]\$

Our Ansible architecture now looks like this

