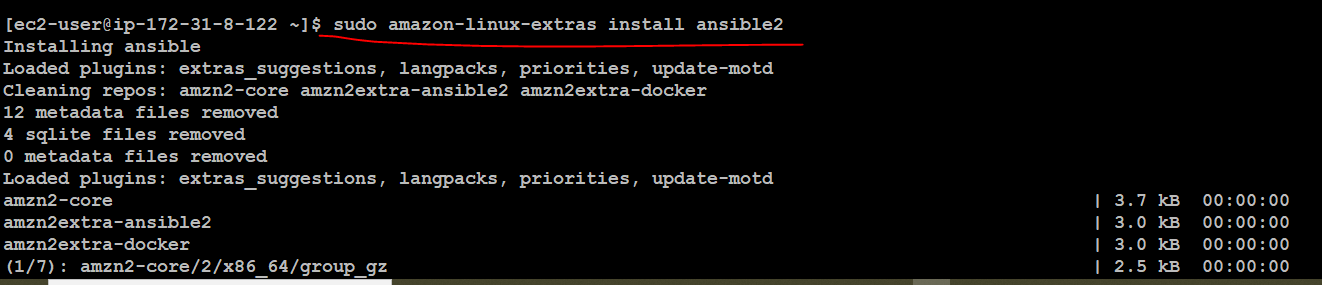
Ansible

Ansible is a configuration Management/automation tool for managing the existing application servers.

**Ansible is written in Python language, so python is its prerequisites. And its working is based on SSH protocol. (SSH Protocol means Key-pair (Pri & Pub) key combination security)**

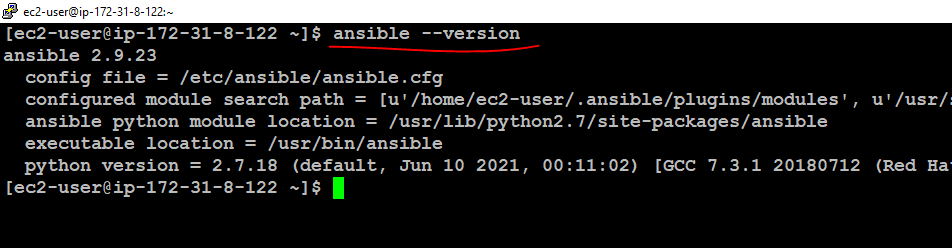
* Ansible is based on master and Slave concept, but the master is called **Control node** and the slave machine(s) is called **Managed node**.
* It is based on **PUSH** mechanism/ Agent-less mechanism
* Ansible has to only be installed on Master machine alone
* Windows can never be a Master for Ansible.
* Python has to be installed on both the MASTER and all the SLAVES in the Ansible structure.
* ***ANSIBLE IS IDEMPOTANT – i.e. if a particular request/command is received more than 1 time, it will only be executed once if the state of the machine is not in that intended state, in case for the second time if we execute the same command, if the state of the machine is in the needed state already, then the command will not be making any changes again.***
* ADHOC commands are used to run a single task (-a) in Ansible
* Ansible Playbooks are used to do Configuration Automation in the Slave Machines from the MASTER
* Modules of Ansible are used in writing ANSIBLE PLAYBOOKS(-m)
* Vaults are used for encrypting files in Ansible
* Roles are used for dependency package installing so that we can reuse them during other creation
* We will work on ansible as a normal user only (i.e. ec2-user/Ubuntu) and not as Root user

# Ansible Installation



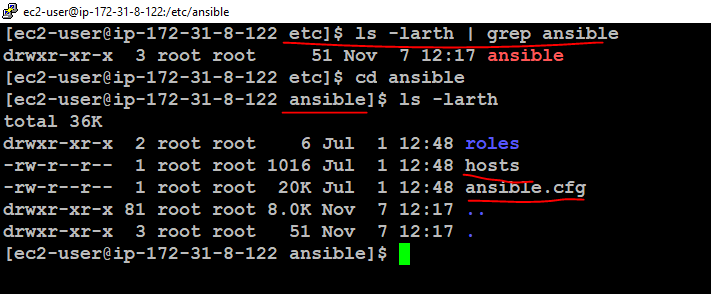
Previously this cmd below was working but right now the above cmd is used to install ansible.





Once we install we can find the ansible files under the /etc path

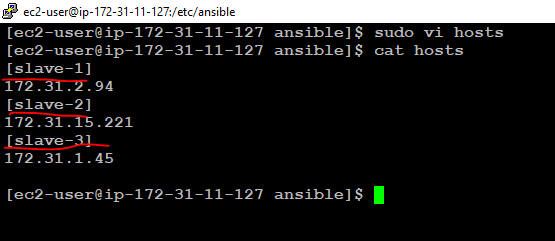
# ANSIBLE INVENTORY FILE



The **ansible.cfg** file is where we have all the configuration details about the MASTER machine, and the **host** file is the file where we store the Slave ip addresses.

In the

# **sudo** **vi /etc/ansible/hosts** file we have added the list of slave ip addresses and also grouped them [ ] based on the server type.



# TYPES of COMMANDS & Modules:

## 1. Adhoc Commands: For Single tasking

Adhoc commands are single line command for a single task.

Eg :

#ansible all –i slaves.txt –m yum –a “name=httpd state=present” –b

Here,

* -m -> modules
* -i -> Inventory file
* -a -> action
* -b -> become root

This command will execute the installation of httpd on all the slave machine ip’s available in the slaves.txt inventory file.

## 2. Playbooks: For Multitasks using code

Ansible playbooks are blocks of code written in YAML format for running a fleet of operations on the slave machine. We can customize the playbook yaml codes based on time and need. The playbook codes are reuseable when we create them as roles. The concepts in Playbook are

* Vaults
* Roles

**NOTE: yaml codes and Case and Space sensitive**

**Before going into the playbooks, we can learn about the colour codes for the state of the machine**

**RED - failed  
YELLOW - Changed  
GREEN – ok / installed**

Now I have written a sample yaml code for installation of Httpd on all slave machines. The name of the file is **firstcode.yaml**

****

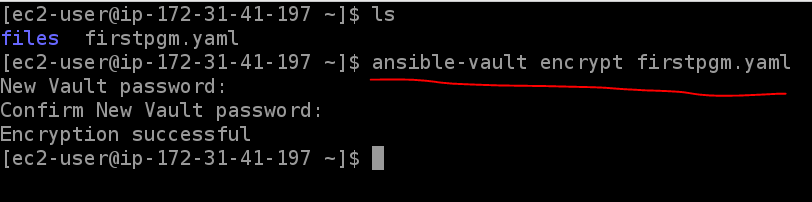
# Ansible Vaults

The Vault property in ansible is used for encrypting any important file that carries sensitive information like passwords, user date. Many times we pass files as a whole to playbooks like how we pass an environment variable.

The command to encrypt a file in ansible

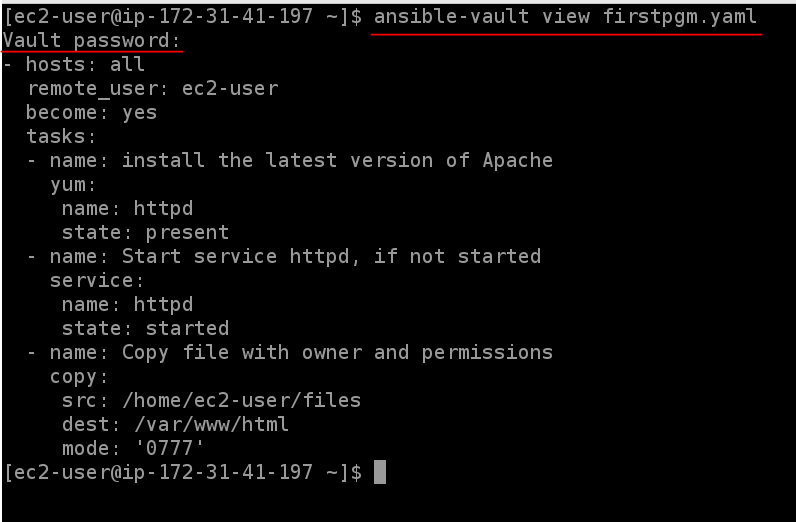
**# ansible-vault encrypt <filename>**

It will prompt for the password for the file.

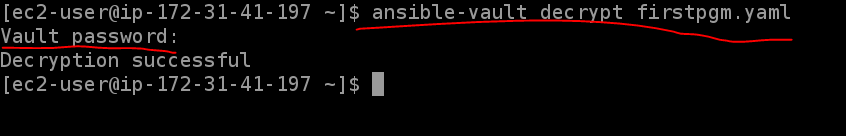


Once the file is encrypted using ansible vault even a user with administrative permissions/even the administrator of the server cannot view the content of the file without the password.

**# ansible-vault view <filename>**



We can permanently decrypt an encrypted file using the below command



# Ansible Roles

Roles are predominantly used for code re-use. We need roles to be assigned to a server/node for it for performing that function of the assigned role. Like in real world scenario, we need roles to upgrade a normal server to a server with specific functionality(s).

So we write roles in ansible that can be stored and reused by other developers for future use.

Roles help in structure the task by organizing them in various folders. ***These roles have to be created in the same folder were the playbook is.***

In real time we write all the dependencies/prerequisites as roles like for Jenkins, Java is the dependency because Jenkins is written Java Pg. language. For Ansible, python is the dependency, so these prerequisites are most of the times written as roles in ansible so that just by calling the roles in a playbook, we can install them. The roles usually have codes that can be used by different developers.

**ROLES ARE BASICALLY USED FOR CODE REUSABLITY.**

## ANSIBLE GALAXY:

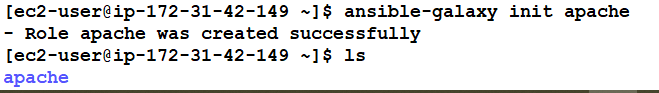
URL: **https://galaxy.ansible.com/**

These roles can be globally shared using the ansible galaxy community.

The command to create a role folder structure

**# ansible-galaxy init <rolename>**

By default all roles we create will fall in **/etc/ansible/roles** folder (as mentioned in the ansible.conf file). This location is the location designated by the system. ***Manually also we can create roles but we have to create the directories inside it manually, but the names have to be same as the /tasks and main.yml respectively.***



But how will the playbook know where the mentioned roles is? We can keep these roles in a folder inside our playbooks containing folders. In case if the roles are not found there, ansible by default will check the /etc/ansible/roles folder as it’s the default location as specified in the ansible.conf folder.

So once we create a role we can upload it to the ansible galaxy using a github repository.

# EXERCISE:

Write a playbook for installing tomcat and change it port number

**Ans:**  
First identify the Linux commands to do this steps.

* STEP 1 : Yum install updates
* STEP 2 : Install java
* STEP 3 : Download tomcat from official website
* STEP 4: Unzip tomcat
* STEP 5 : Change the tomcat port number
* STEP 6 : Stop/Start the tomcat
* STEP 7 : Run a sample application (copy paste an application)

CODE :

- hosts: all

  remote\_user: ec2-user

  become: yes

  tasks:

   - name: update all

     yum:

      name: '\*'

      state: latest

   - name: Install java for tomcat- dependency

     yum:

      name: java-1.8.0-openjdk

      state: present

   - name: Download tomcat webserver

     get\_url:

      url: https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.55/bin/apache-tomcat-9.0.55.tar.gz

      dest: /opt

      mode: '0777'

   - name: Unarchive a file that is already on the remote machine

     unarchive:

      src: /opt/apache-tomcat-9.0.55.tar.gz

      dest: /opt

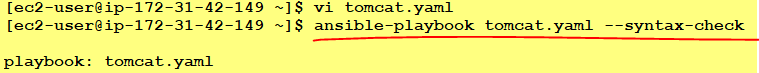
      remote\_src: yes

      mode: '0777'

Now we shall Copy paste this code from Visual studio to a filename **tomcat.yaml** in the root path of the ***Ansible Master server***.

So first do the syntax check for this code

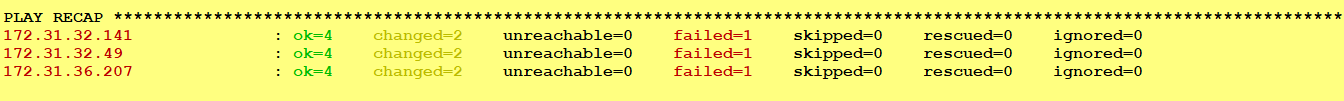
**# ansible-playbook tomcat.yaml --syntax-check**



As a best practice also do a dry run for the playbook

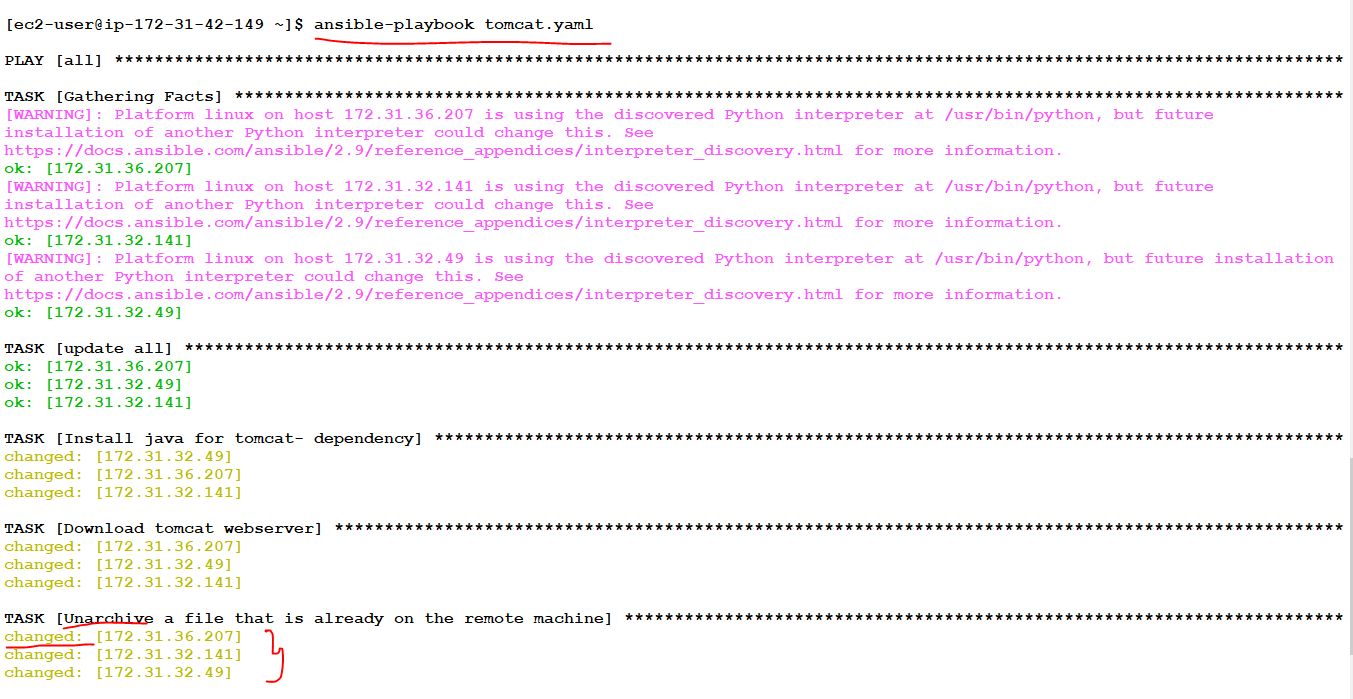
**# ansible-playbook tomcat.yaml –check**

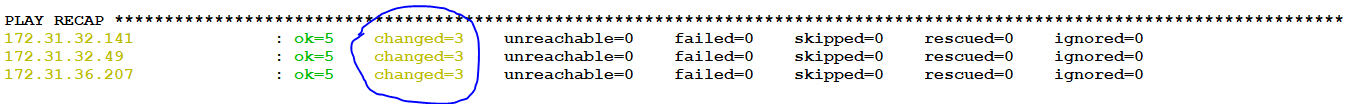




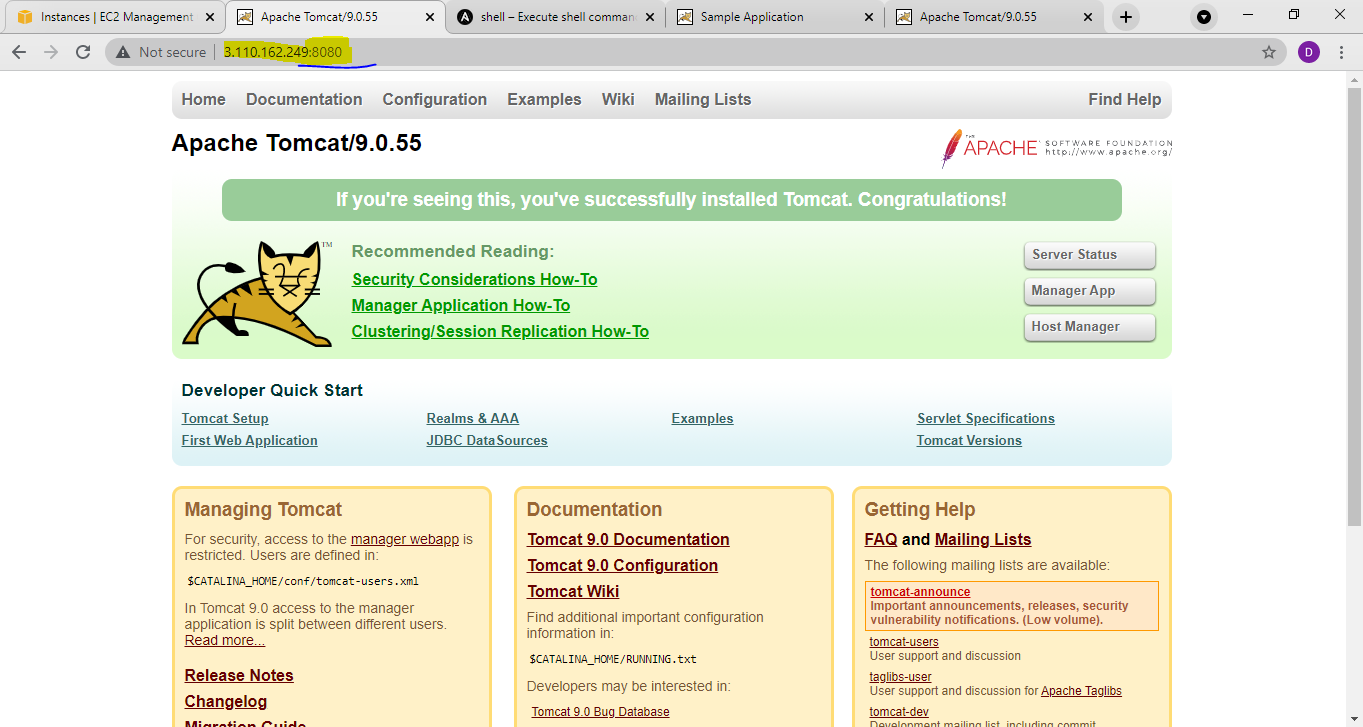
The dry run (--check) will show the unachieved step of tomcat as failed because, dry run is just a fake run and does not do any actually change to the state of the machine. Incase if we run the playbook code for real this unachieve will be success.

**# ansible-playbook tomcat.yaml –b**





**We can check the tomcat installation successfully by ping the browser with the <public ip if the slave machine>:8080**



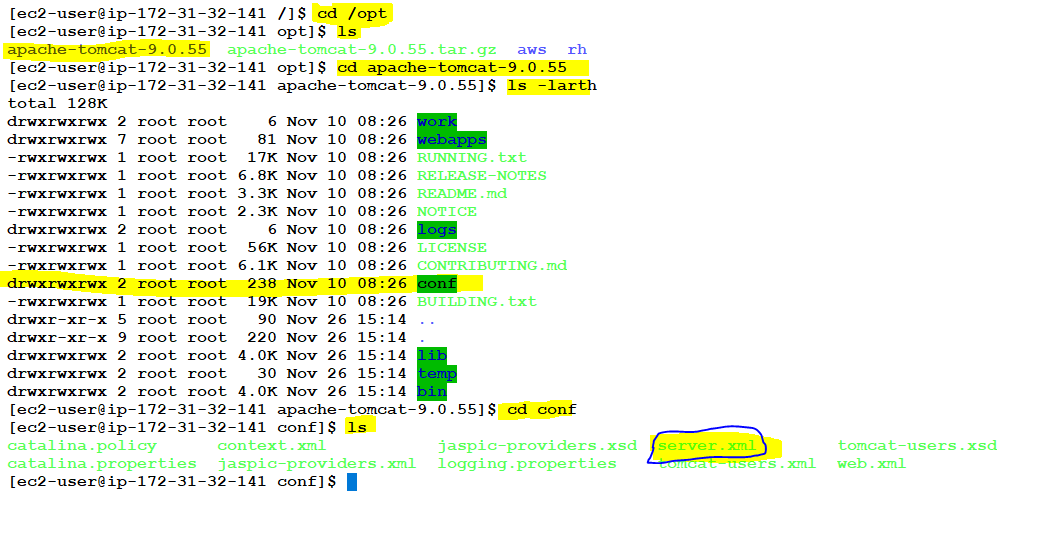
So this playbook code has finished until step 4. Now to run step 5, we need a module call ***template*** in ansible. This module will copy and replace an entire file from source location (ansible master) to destination location (slave machine).

This template is where we use JINJA code {{….}}.

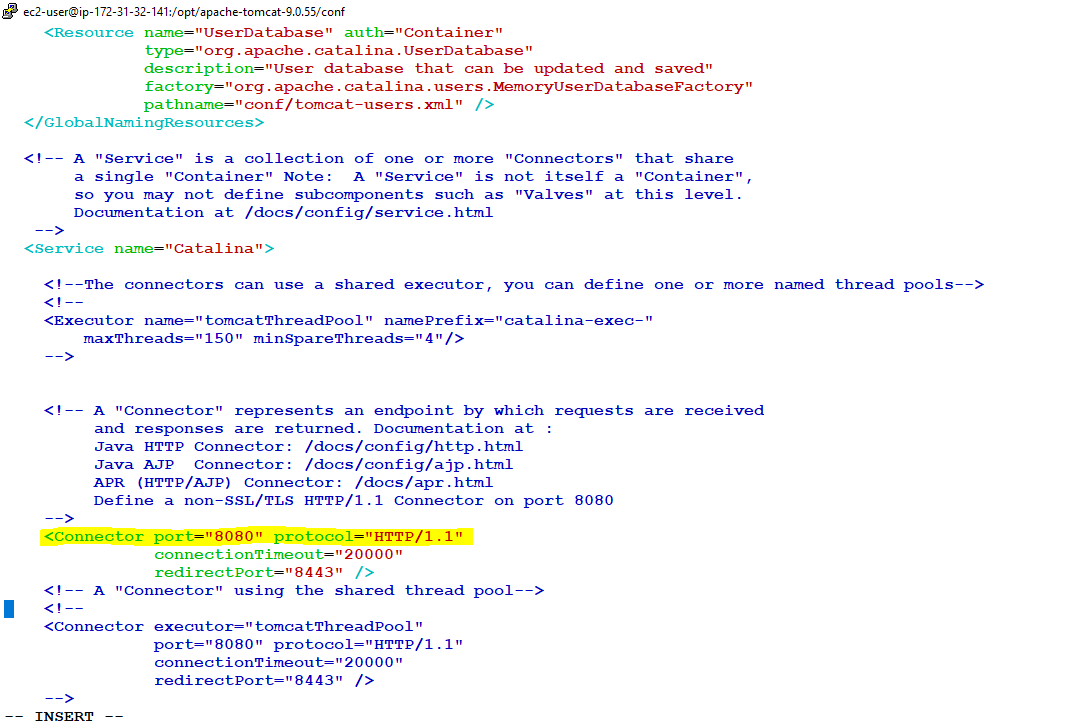
JINJA(j2) is written in PYTHON language. It is a fast, expressive and extensible templating engine.

So to incorporate the this file we have to server copy the tomcat configuration file from slave server and convert it into a j2(jinja) file.

The tomcat conf file which will have the port number of tomcat will be under the ***/opt/apache-tomcat-9.0.55/conf/server.xml*** path on the slave machine



Inside the server.xml file we have the configuration details of the tomcat webserver. We need to modify the line connector port.



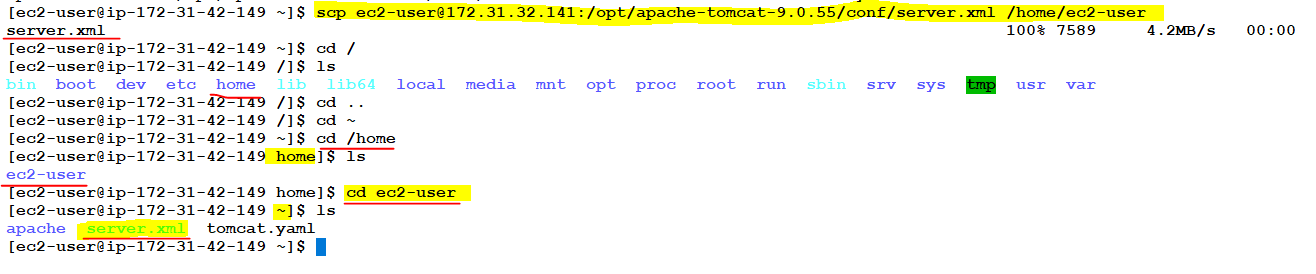
So we are going to copy this file from slave machine to Ansible master machine using the server copy command from master machine

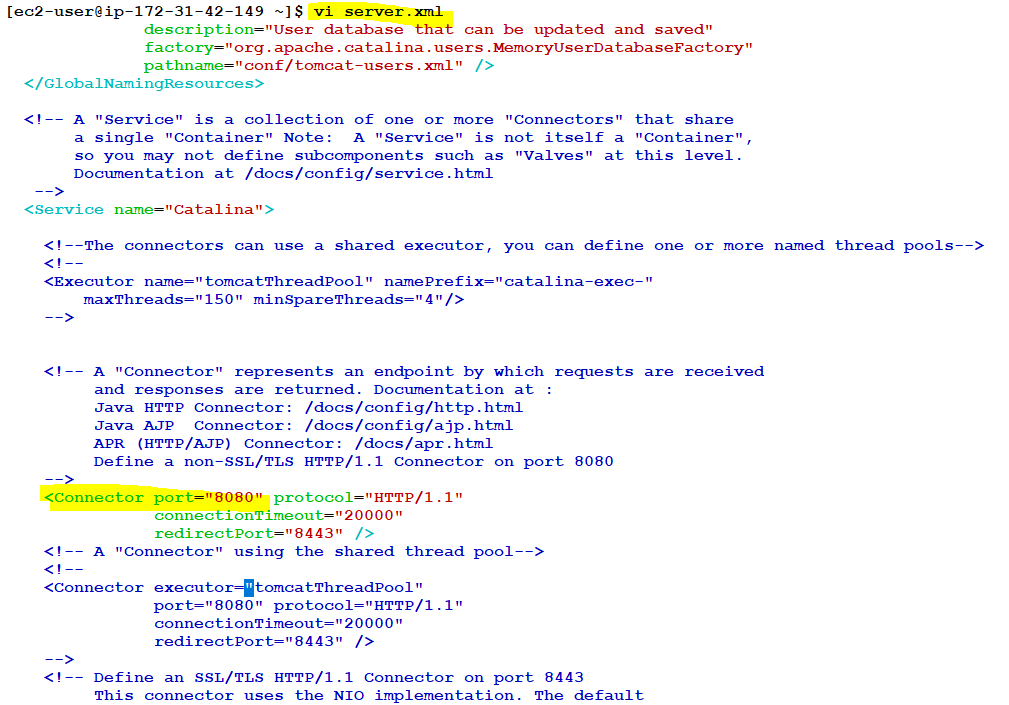
Syntax:

**# scp username@<slave private ip addr>:/path on slave machine to be copied /path in the master sever to be copied into**

Eg:

# scp ec2-user@ 172.31.32.141:**/opt/ apache-tomcat-9.0.55/conf/server.xml** **/home/ec2-user**



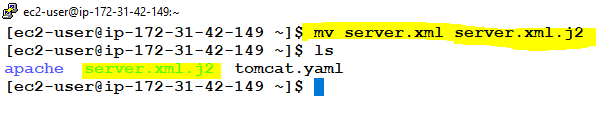


Now we need edit the file and convert it into jinja code file. Server.xml.j2, the modification we are going to do is changing the connector port line to jinja code.



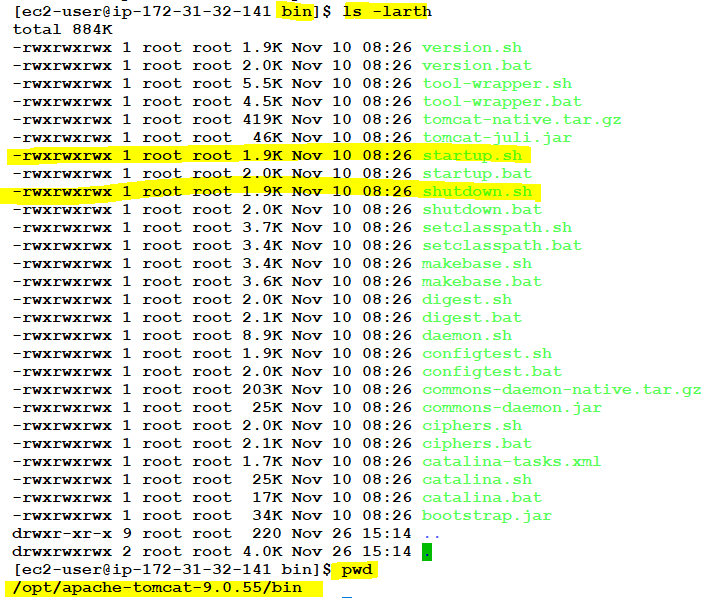
Now **“{{ tomcat\_port }}”** is an environment variable inside {{…}} jinja code, so we can dynamically pass values to this file from the playbook while executing it.

Save this file with a .j2 extension to make it a jinja file.



Now we have to write a script to copy this template module as code to copy the server.xml.j2 file from master ansible machine to slave. So whenever we do any configuration change in the server we need to restart it, so for tomcat we have shell script to do that, so we also use the shell module of ansible to execute the server start and stop command for tomcat servers.

We can find this shell script command of tomcat sever by ssh ‘ing into the slave machine and the path **/opt/apache-tomcat-9.0.55/bin**



 - name: Template module to copy paste the jinja file from ansible master to slave

   template:

    src: /home/ec2-user/server.xm.j2

    dest: /opt/apache-tomcat-9.0.55/conf/server.xml

    mode: '0777'

- name: shell command to stop the tomcat server

   shell: nohup /opt/apache-tomcat-9.0.55/bin/shutdown.sh &

 - name: shell command to stop the tomcat server

   shell: nohup /opt/apache-tomcat-9.0.55/bin/startup.sh &

So we are going to integrate this code and run, but one important step is to mention the new port number to be replace in the {{tomcat\_port}} j2 of the server.xml.j2 file as environment variable. PFB

- hosts: all

  remote\_user: ec2-user

  become: yes

tomcat\_port: 9090

  tasks:

   - name: update all

     yum:

      name: '\*'

      state: latest

   - name: Install java for tomcat- dependency

     yum:

      name: java-1.8.0-openjdk

      state: present

   - name: Download tomcat webserver

     get\_url:

      url: https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.55/bin/apache-tomcat-9.0.55.tar.gz

      dest: /opt

      mode: '0777'

   - name: Unarchive a file that is already on the remote machine

     unarchive:

      src: /opt/apache-tomcat-9.0.55.tar.gz

      dest: /opt

      remote\_src: yes

      mode: '0777'

 - name: Template module to copy paste the jinja file from ansible master to slave

     template:

      src: /home/ec2-user/server.xm.j2

      dest: /opt/apache-tomcat-9.0.55/conf/server.xml

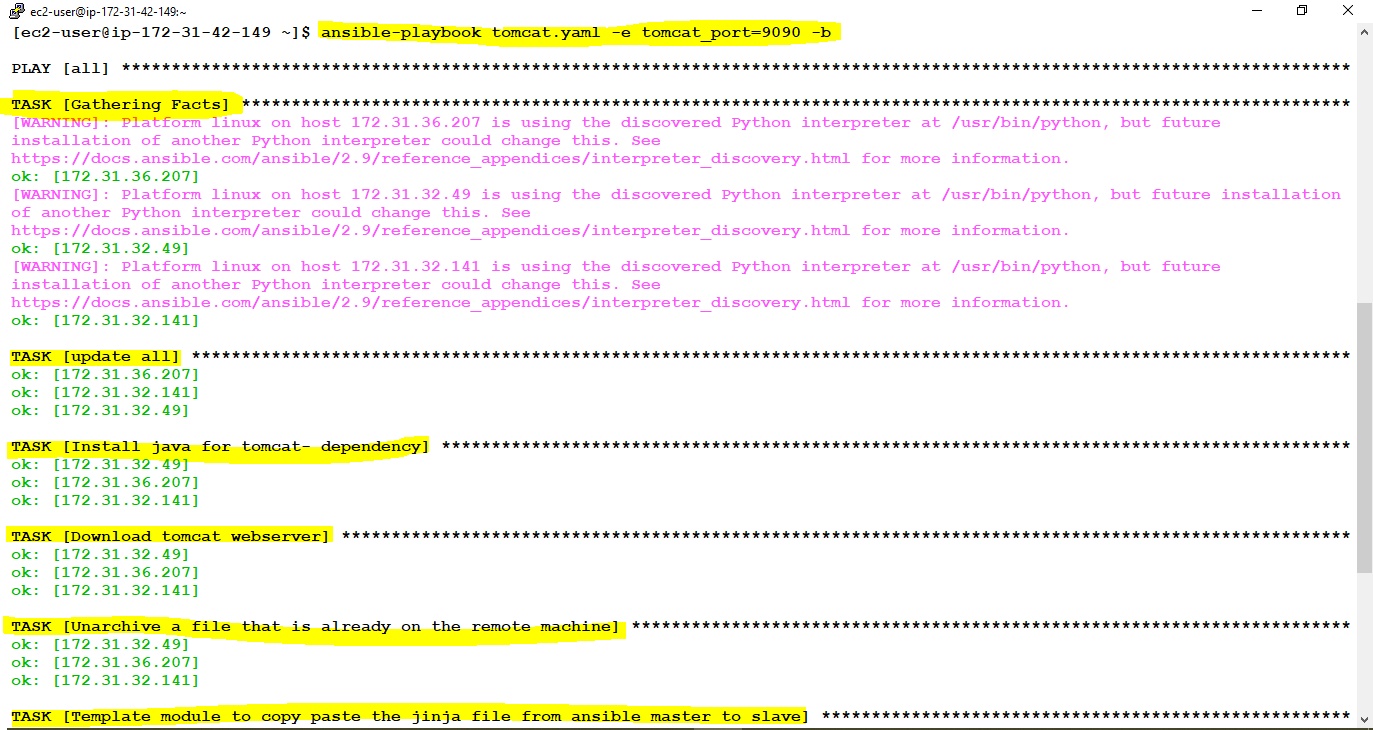
      mode: '0777'

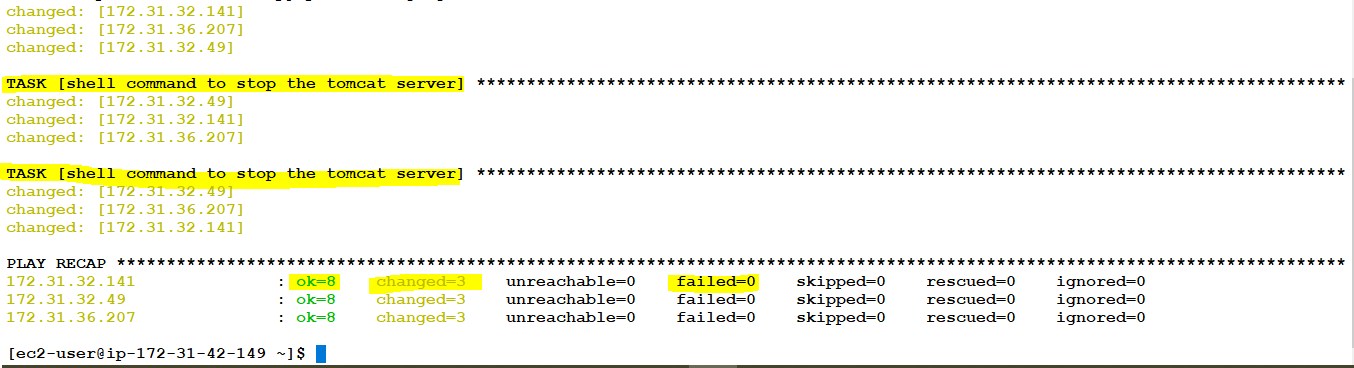
   - name: shell command to stop the tomcat server

     shell: nohup /opt/apache-tomcat-9.0.55/bin/shutdown.sh &

   - name: shell command to stop the tomcat server

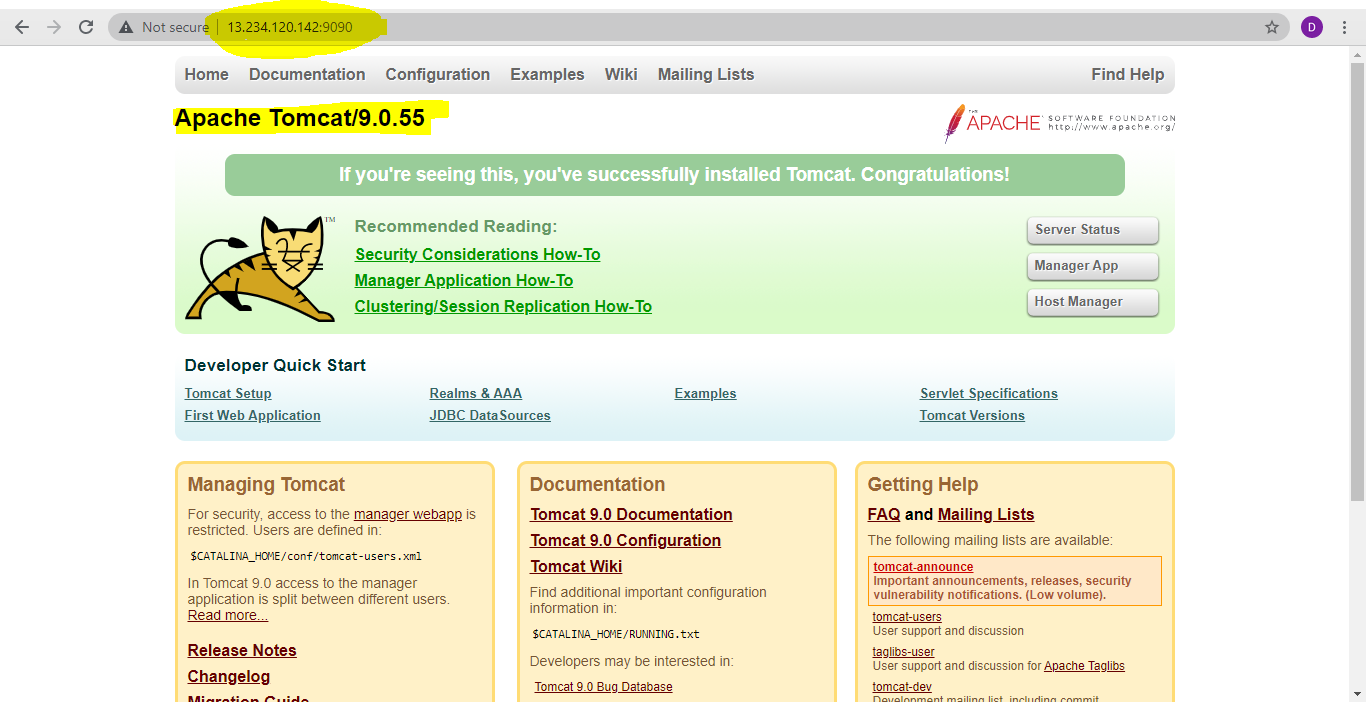
     shell: nohup /opt/apache-tomcat-9.0.55/bin/startup.sh &





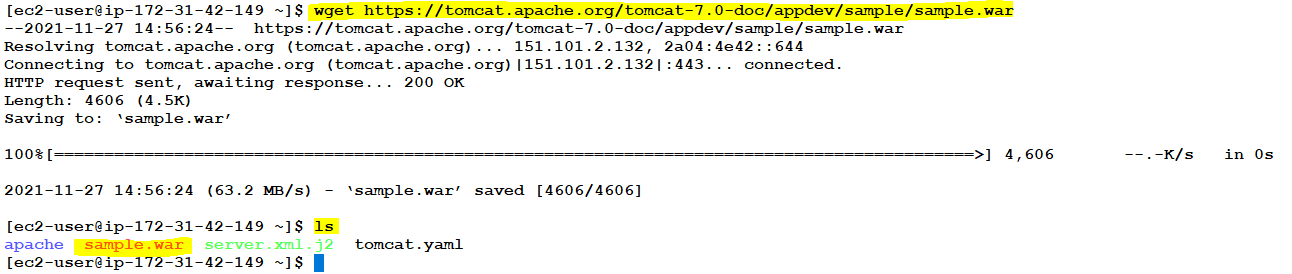
So its successful, now we can check this by pinging any of the slave <public\_ip>:9090 on the browser

**http:// 13.234.120.142:9090**



Now the port number of tomcat is successful changed from default value, 8080 to 9090 via jinja code. Now for the last step we need to copy a sample war file for tomcat deployment

For that we are going to download a sample WAR package for tomcat on the Ansible master server and copy the war package into the slave machines in the path ***/opt/apache-tomcat-9.0.55/webapps***



We are going to use the copy module to copy the war package from ansible master to slave machine.

 - name: copying files

    copy:

      src: /home/ec2-user/sample.war

      dest: /opt/apache-tomcat-9.0.46/webapps/

      mode: '0777'

***FINAL COMPLETE CODE:***

- hosts: all

  remote\_user: ec2-user

  become: yes

  vars:

    tomcat\_port: 9090

  tasks:

   - name: update all

     yum:

      name: '\*'

      state: latest

   - name: Install java for tomcat- dependency

     yum:

      name: java-1.8.0-openjdk

      state: present

   - name: Download tomcat webserver

     get\_url:

      url: <https://tomcat.apache.org/tomcat-9.0-doc/appdev/sample/sample.war>

      dest: /opt

      mode: '0777'

   - name: Unarchive a file that is already on the remote machine

     unarchive:

      src: /opt/apache-tomcat-9.0.55.tar.gz

      dest: /opt

      remote\_src: yes

      mode: '0777'

   - name: Template module to copy paste the jinja file from ansible master to slave

     template:

      src: /home/ec2-user/server.xml.j2

      dest: /opt/apache-tomcat-9.0.55/conf/server.xml

   - name: shell command to stop the tomcat server

     shell: nohup /opt/apache-tomcat-9.0.55/bin/shutdown.sh &

   - name: shell command to start the tomcat server

     shell: nohup /opt/apache-tomcat-9.0.55/bin/startup.sh &

   - name: copying files

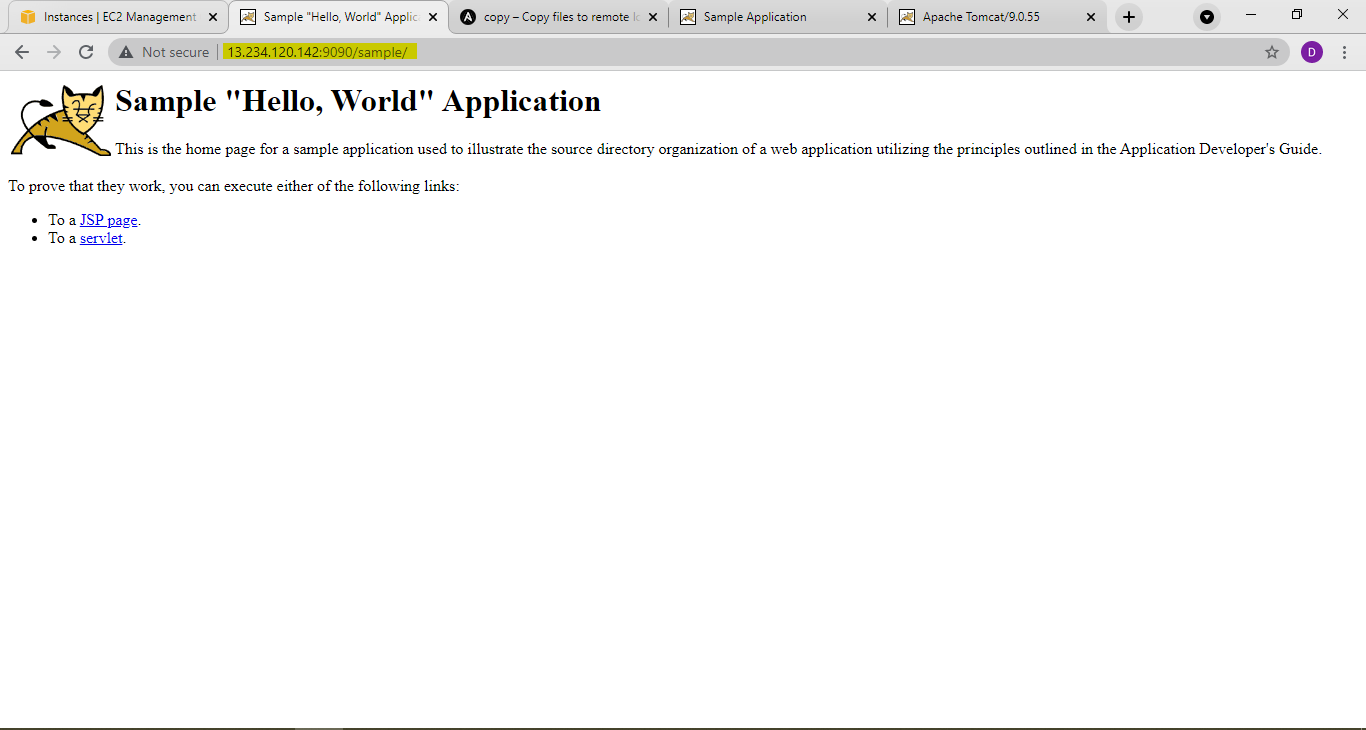
     copy:

      src: /home/ec2-user/sample.war

      dest: /opt/apache-tomcat-9.0.46/webapps/

      mode: '0777'

***FINAL OUTPUT:***





The YAML code of the above tomcat installation