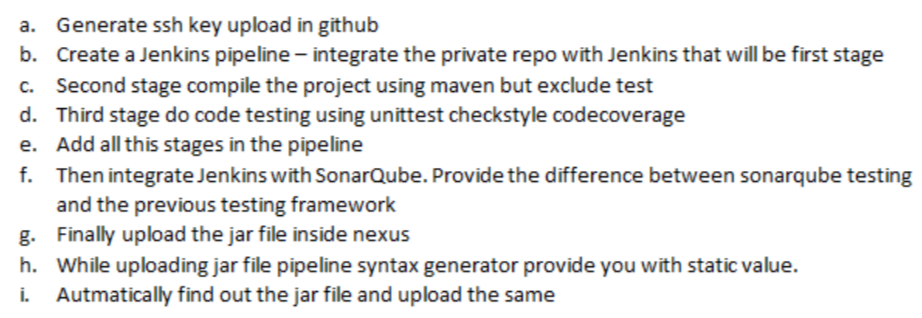
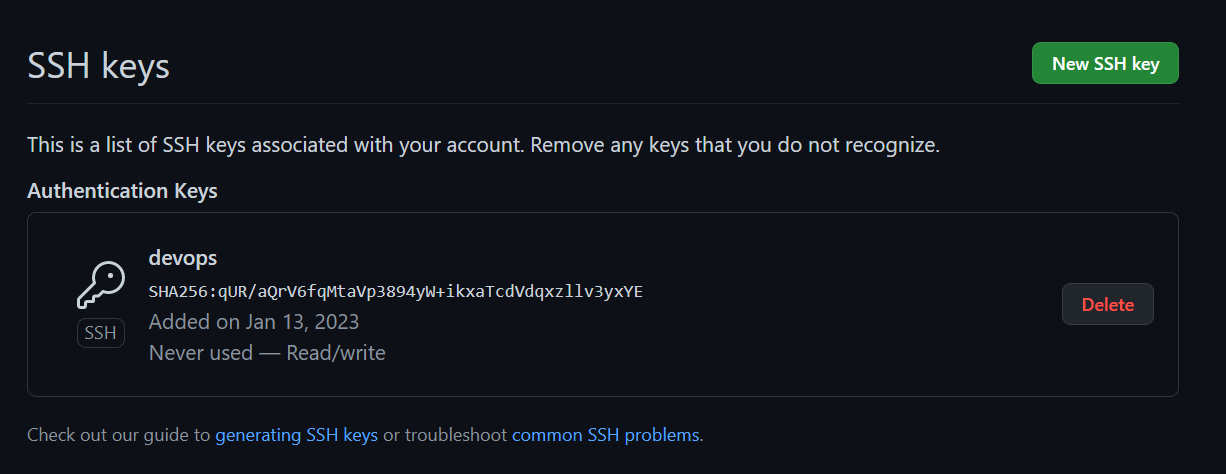
**Building an entire CICD pipeline in Jenkins**

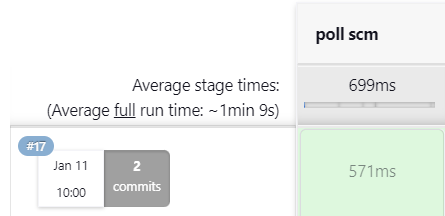
**Part-1:**

****

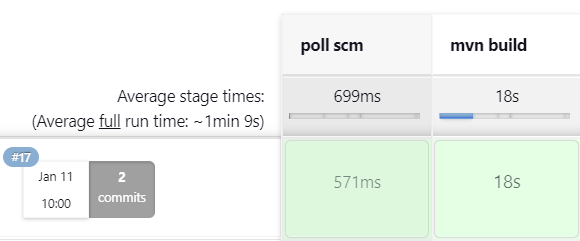
1. Generate ssh key upload in github

****

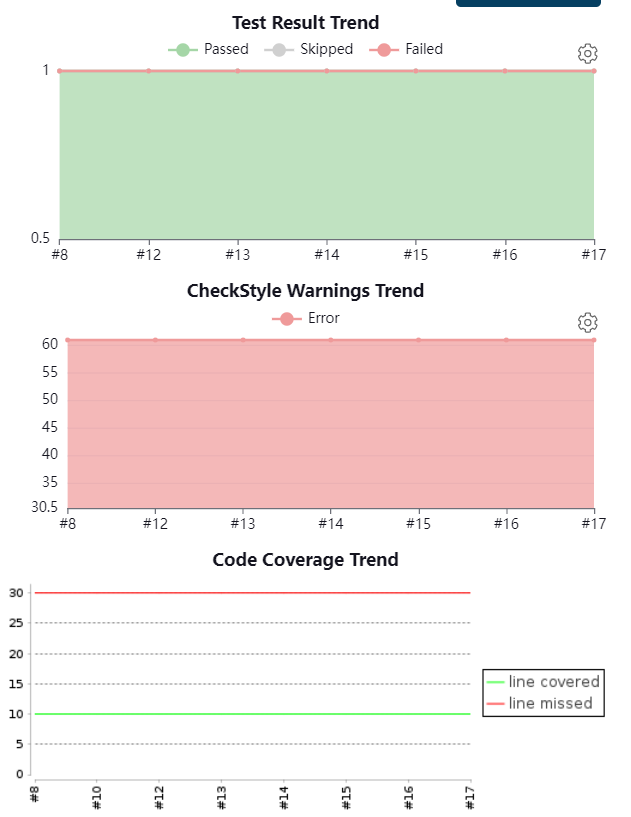
1. Create a Jenkins pipeline – integrate the private repo with Jenkins that will be first stage

**** ****

1. Second stage compile the project using maven but exclude test

****

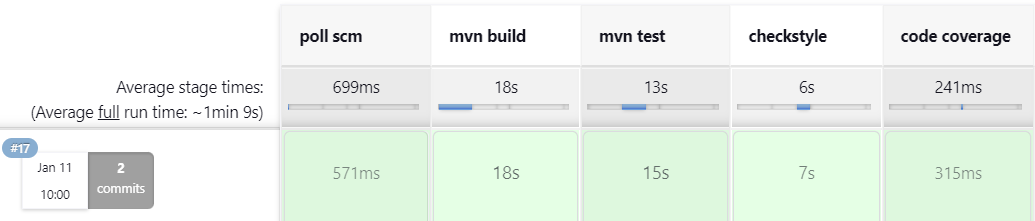
1. Third stage do code testing using unittest checkstyle codecoverage

****

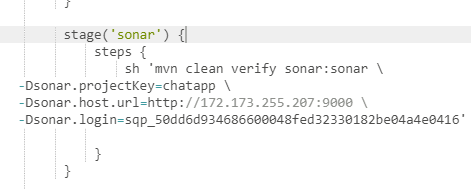
****

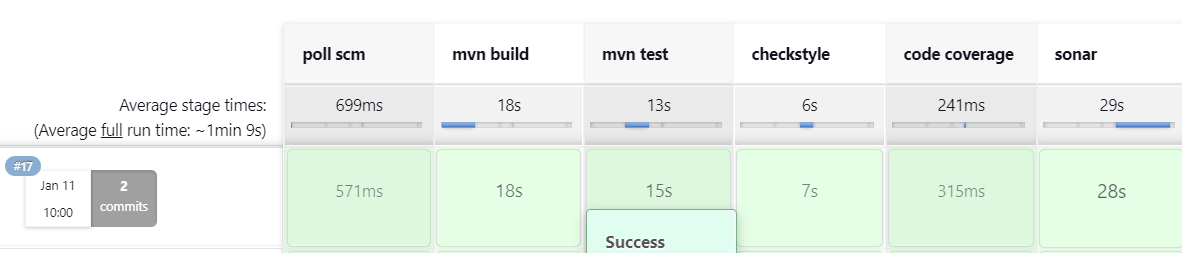
1. Add all this stages in the pipeline

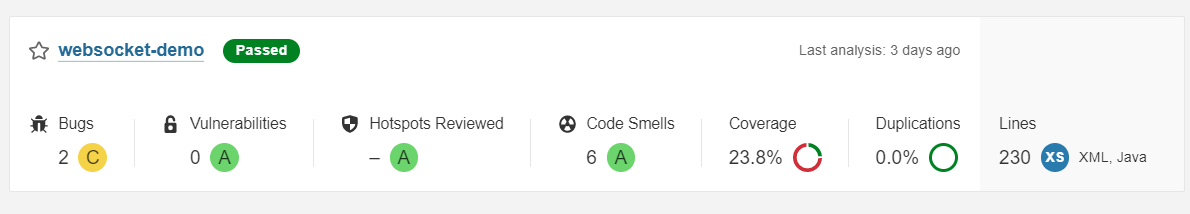
****

****

1. Then integrate Jenkins with SonarQube. Provide the difference between sonarqube testing and the previous testing framework

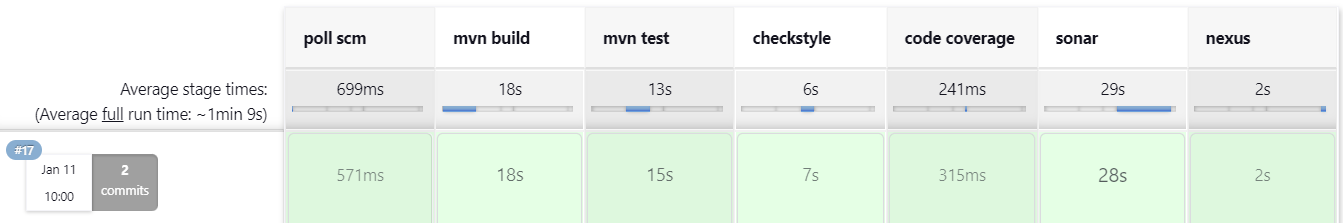
****

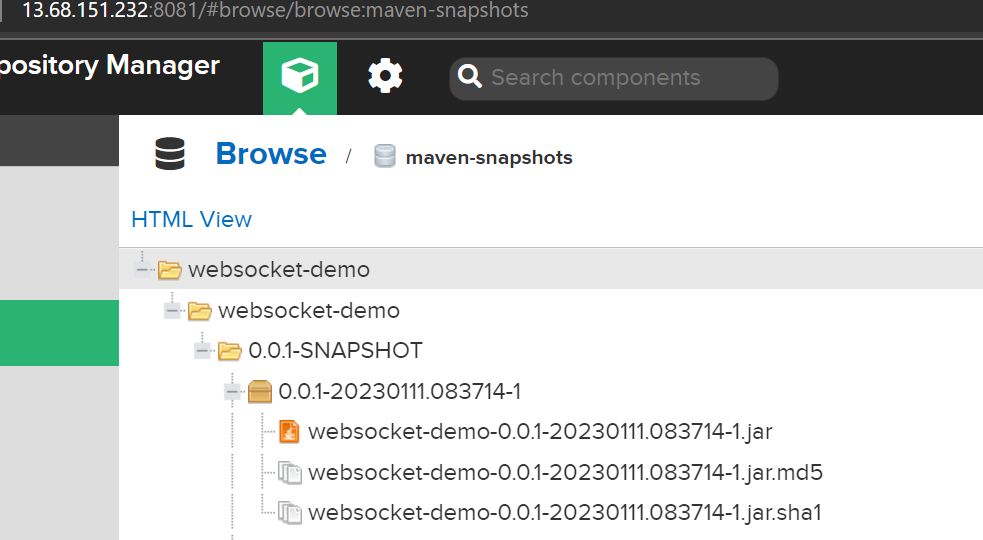
****

****

1. Finally upload the jar file inside nexus

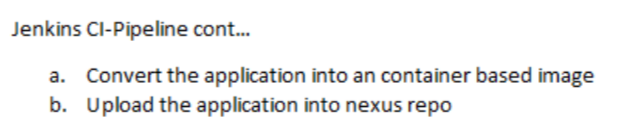
****

****

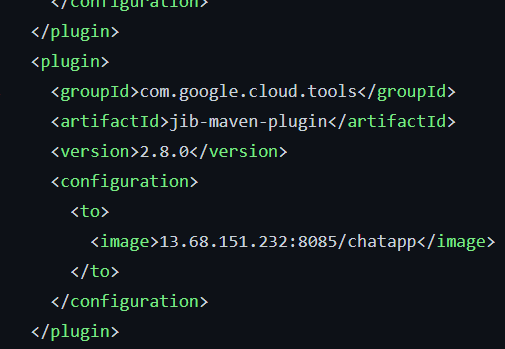
****

1. While uploading jar file pipeline syntax generator provide you with static value.
2. stage('nexus') {
3. steps {
4. nexusArtifactUploader artifacts: [[artifactId: 'websocket-demo', classifier: '', file: 'target/websocket-demo-0.0.1-SNAPSHOT.jar', type: 'jar']], credentialsId: 'nexus-cred', groupId: 'websocket-demo', nexusUrl: '13.68.151.232:8081', nexusVersion: 'nexus3', protocol: 'http', repository: 'maven-snapshots', version: '0.0.1-SNAPSHOT'
5. }
6. }
7. Autmatically find out the jar file and upload the same
8. stage('nexus') {
9. steps {
10. script {
11. pom = readMavenPom file: "pom.xml";
12. filesByGlob = findFiles(glob: "target/\*.${pom.packaging}");
13. echo "${filesByGlob[0].name} ${filesByGlob[0].path} ${filesByGlob[0].directory} ${filesByGlob[0].length} ${filesByGlob[0].lastModified}"
14. artifactPath = filesByGlob[0].path;
15. }
16. nexusArtifactUploader artifacts: [[artifactId: pom.artifactId, classifier: '', file: artifactPath, type: pom.packaging, type: 'jar']], credentialsId: 'nexus-cred', groupId: pom.artifactId, nexusUrl: '13.68.151.232:8081', nexusVersion: 'nexus3', protocol: 'http', repository: 'maven-snapshots', version: pom.version
18. }
19. }

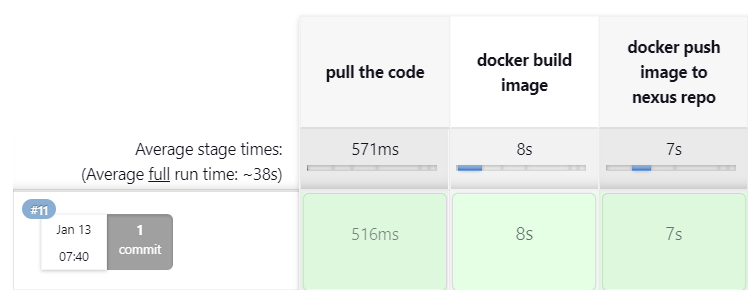
**Part-2:**

****

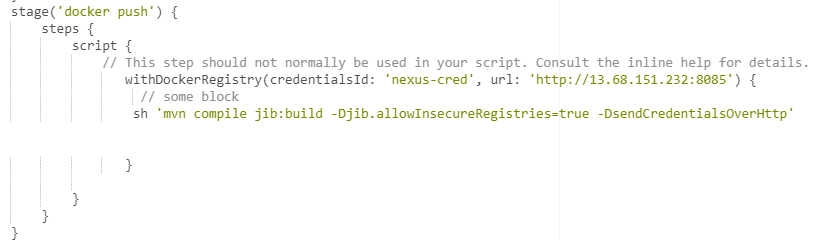
1. Convert the application into an container based image

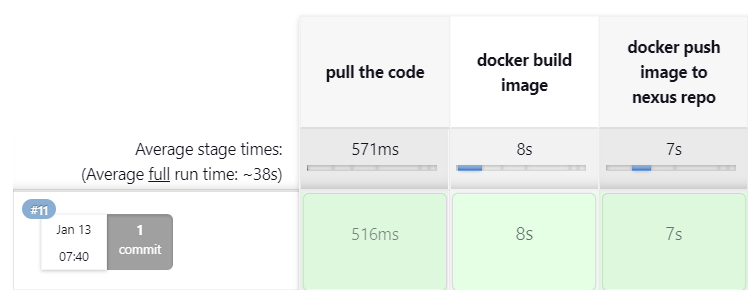
****

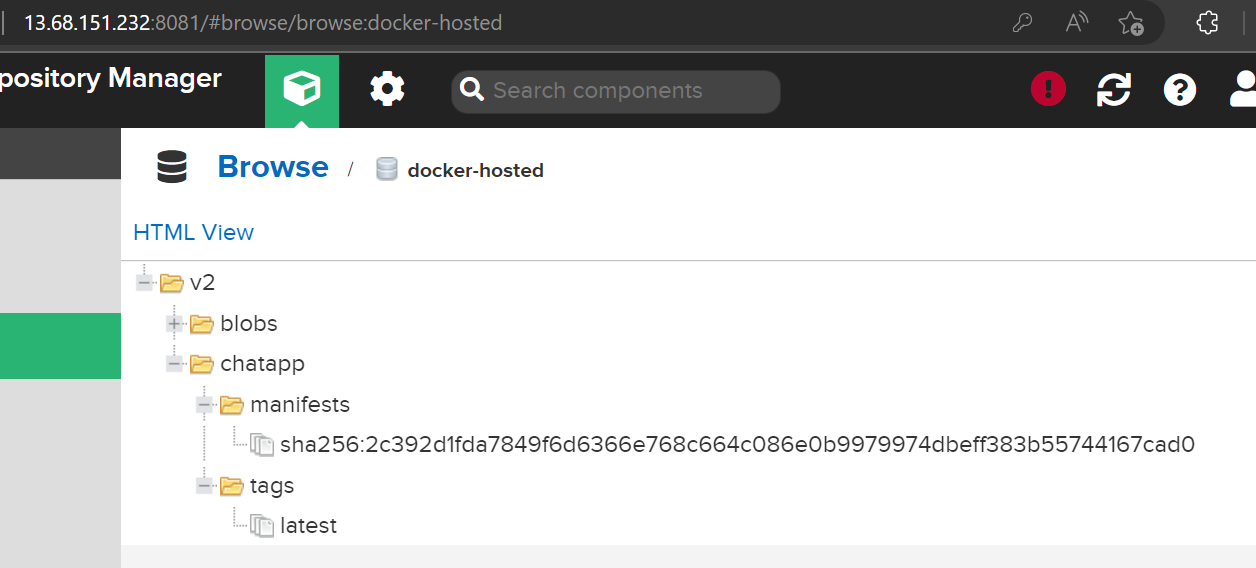
****

****

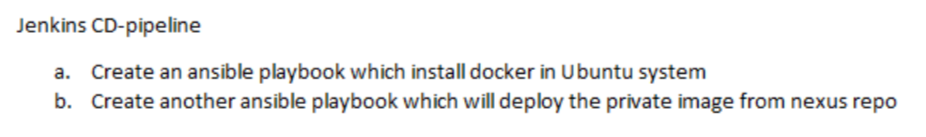
1. Upload the application into nexus repo

****

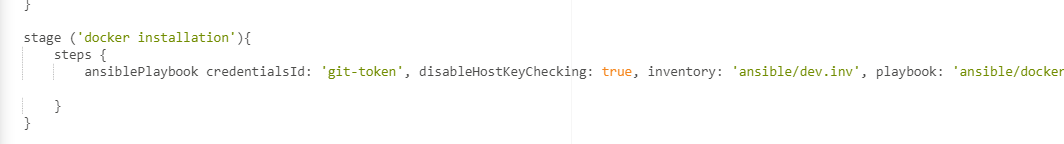
****

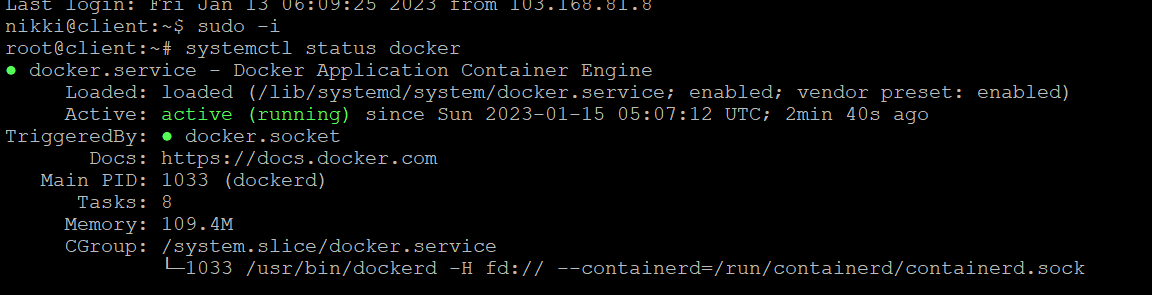
****

**Part-3:**

****

1. Create an ansible playbook which install docker in Ubuntu system

****

****

---

- name: install Docker

  hosts: all

  become: true

  tasks:

    - name: Install apt-transport-https

      ansible.builtin.apt:

        name:

          - apt-transport-https

          - ca-certificates

          - lsb-release

          - gnupg

        state: latest

        update\_cache: true

    - name: Add signing key

      ansible.builtin.apt\_key:

        url: "https://download.docker.com/linux/{{ ansible\_distribution | lower }}/gpg"

        state: present

    - name: Add repository into sources list

      ansible.builtin.apt\_repository:

        repo: "deb [arch={{ ansible\_architecture }}] https://download.docker.com/linux/{{ ansible\_distribution | lower }} {{ ansible\_distribution\_release }} stable"

        state: present

        filename: docker

    - name: Install Docker

      ansible.builtin.apt:

        name:

          - docker

          - docker.io

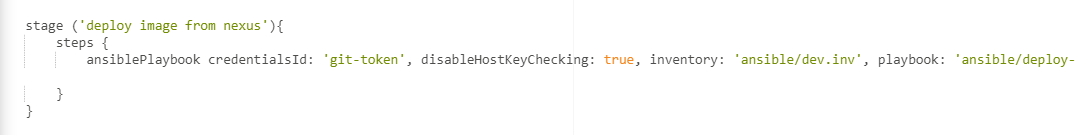
          - docker-compose

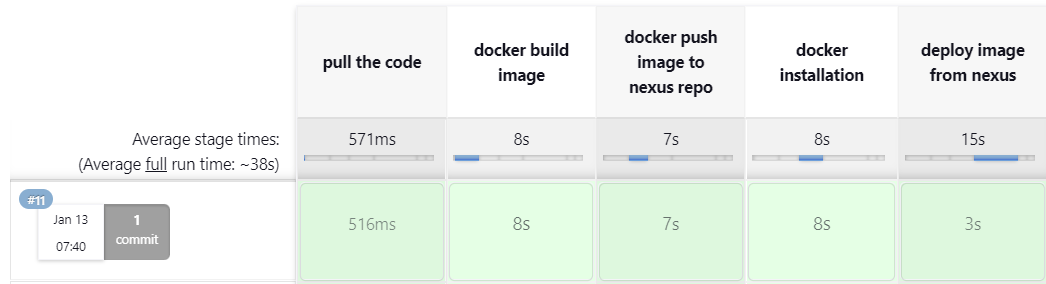
          - docker-registry

        state: latest

        update\_cache: true

1. Create another ansible playbook which will deploy the private image from nexus repo

****

****

---

- name: Deploying docker image

  hosts: all

  become: true

  tasks:

        - name: Log into private registry and force re-authorization

          docker\_login:

            registry\_url: 13.68.151.232:8085

            username: admin

            password: admin@123

            reauthorize: true

        - name: excuting the container

          docker\_container:

            name: chatapp

            image: 13.68.151.232:8085/chatapp

            state: started

            pull: 'true'

            ports:

            - "8082:8080"

        - name: port 8080

          tags: test

          uri:

            url: http://localhost:8080

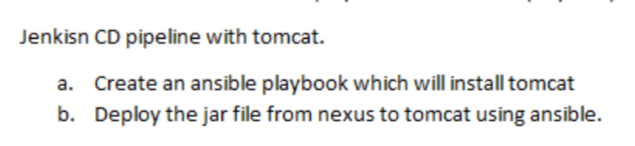
          register: result

          until: "result.status == 200"

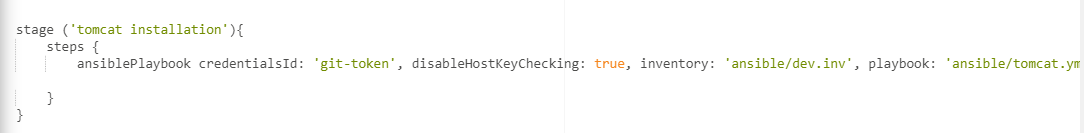
          retries: 5

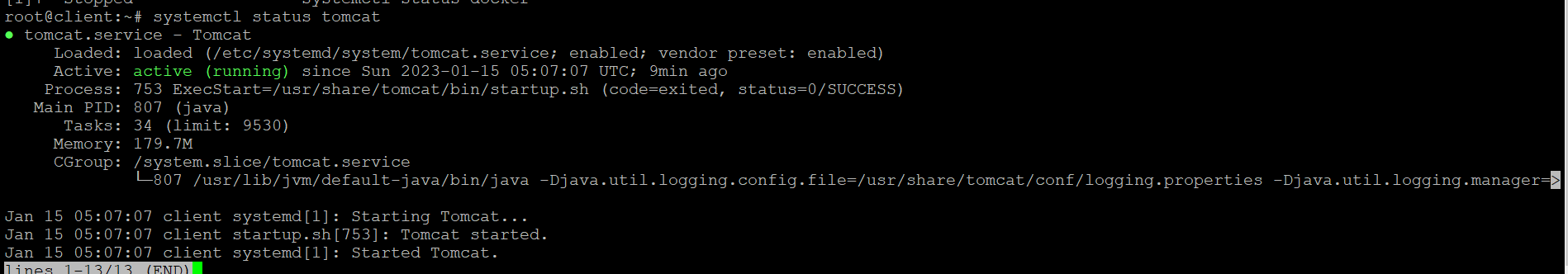
          delay: 10

**Part-4:**

****

1. Create an ansible playbook which will install tomcat

****

****

---

-

  hosts: all

  vars:

    tomcat\_download\_url: https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.70/bin/apache-tomcat-9.0.70.tar.gz

    tomcat\_download\_location: /usr/share/tomcat/apache-tomcat-9.0.70.tar.gz

  ##define list of task that we are executing

  tasks:

    - name: update my endpoint and install openjdk

      ansible.builtin.apt:

        name: default-jdk

        state: latest

        update\_cache: yes

    - name: create a tomcat group

      ansible.builtin.group:

        name: tomcat

        state: present

    - name: Add the user tomcat

      ansible.builtin.user:

        name: tomcat

        group: tomcat

    - name: create tomcat directory

      file:

        path: /usr/share/tomcat

        state: directory

        owner: tomcat

        group: tomcat

    - name: Download Tomcat

      get\_url:

        url: "{{tomcat\_download\_url}}"

        dest: "{{tomcat\_download\_location}}"

    - name: unarchive the tomcat

      unarchive:

        src: "{{tomcat\_download\_location}}"

        dest: /usr/share/tomcat

        remote\_src: yes

        extra\_opts: [--strip-components=1]

        owner: tomcat

        group: tomcat

        creates: /usr/share/tomcat/bin

    - name: copy the tomcat service

      template:

        src: templates/tomcat.service.j2

        dest: /etc/systemd/system/tomcat.service

      when: ansible\_service\_mgr == 'systemd'

    - name: start and enable tomcat

      service:

        daemon\_reload: yes

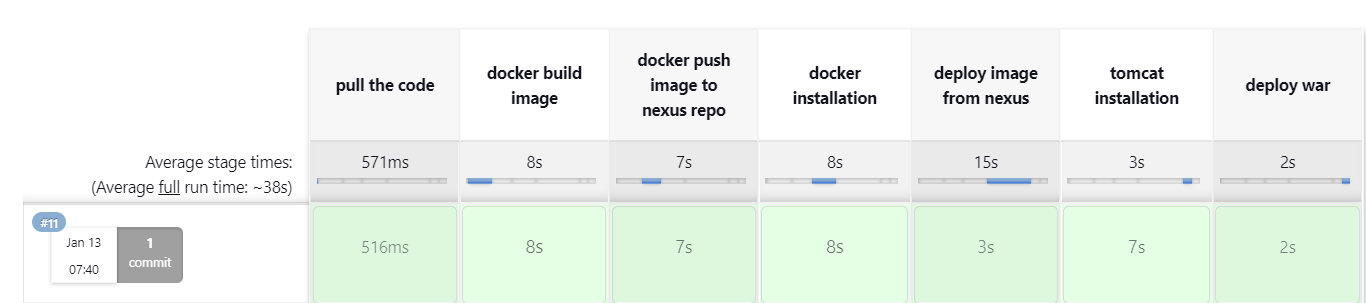
        name: tomcat

        state: started

        enabled: yes

      when: ansible\_service\_mgr == 'systemd'

1. Deploy the jar file from nexus to tomcat using ansible.

****

****

---

- name: Install the war file

  hosts: all

  become: false

  vars:

    myurl: "https://raw.githubusercontent.com/aeimer/java-example-helloworld-war/master/dist/helloworld.war"

    mydest: "/usr/share/tomcat/webapps"

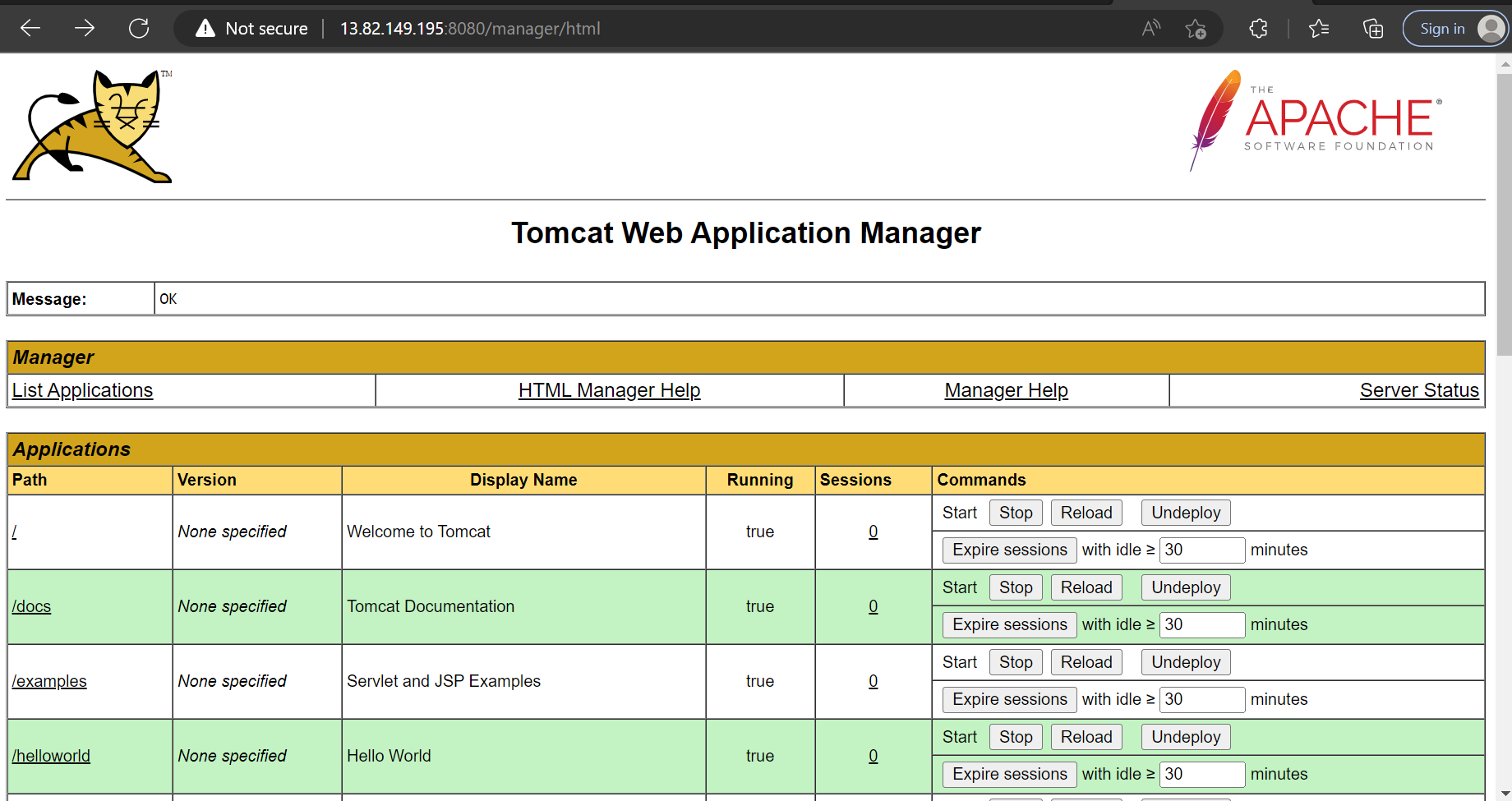
  tasks:

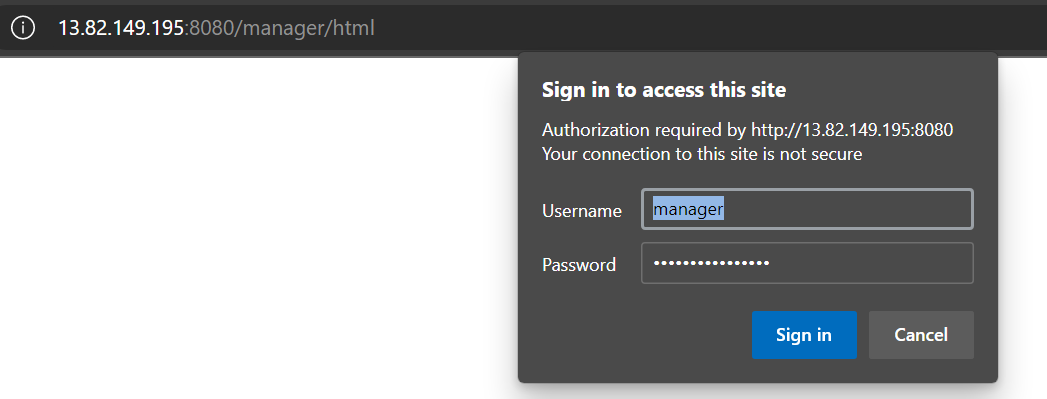
    - name: download file

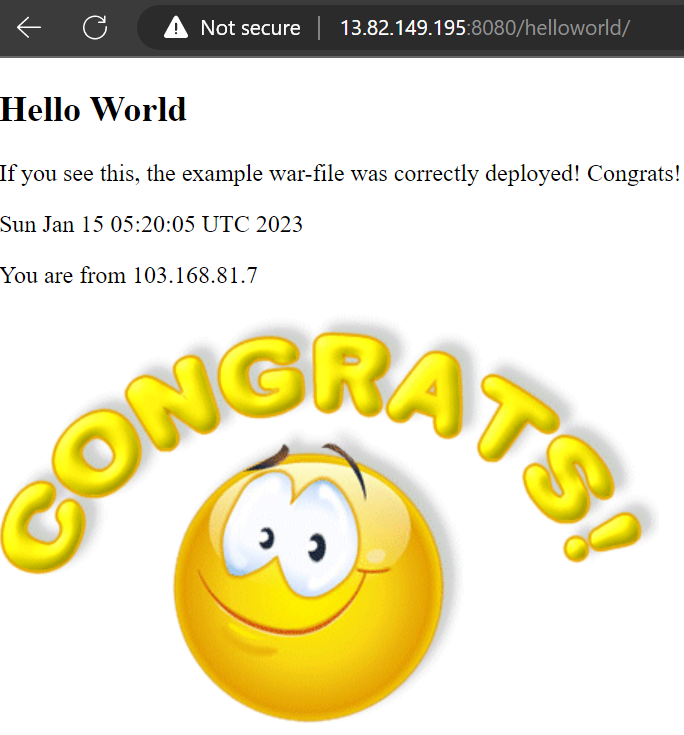
      ansible.builtin.get\_url:

        url: "{{ myurl }}"

        dest: "{{ mydest }}"

****

****

****

---

- hosts: all

  become: true

  vars:

    username: admin

    password: admin@123

  tasks:

    - name: Install required system packages

      apt:

        pkg:

          - apt-transport-https

          - ca-certificates

          - curl

          - software-properties-common

          - python3-pip

          - virtualenv

          - python3-setuptools

        state: latest

        update\_cache: true

    - name: Install lxml

      pip:

        name: lxml

    - name: Download a WAR File to the Tomcat

      maven\_artifact:

            group\_id: websocket-demo

            artifact\_id: websocket-demo

            version: 0.0.1-SNAPSHOT

            extension: war

            repository\_url: 'http://13.68.151.232:8081/repository/maven-snapshots'

            username: "{{ username }}"

            password: "{{ password }}"

            dest: /opt/tomcat/webapps/web-app.war

            mode: "0644"

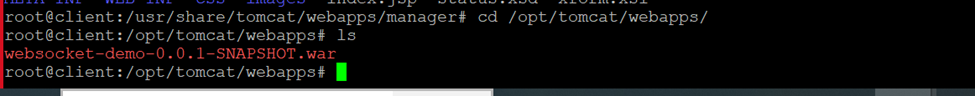
    - name: Restart the Tomcat

      systemd:

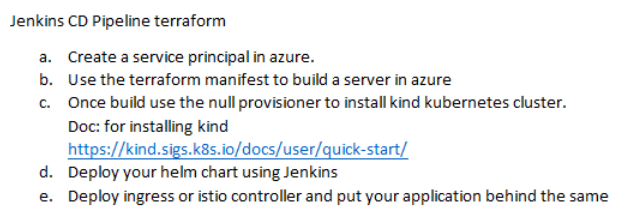
        name: tomcat

        enabled: yes

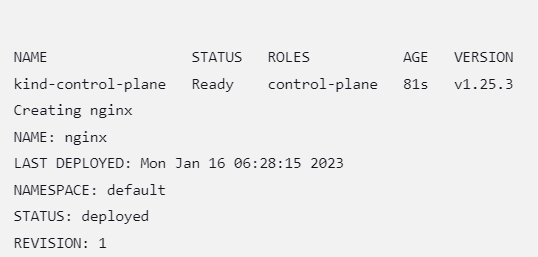
        state: restarted

****

**Part-5:**

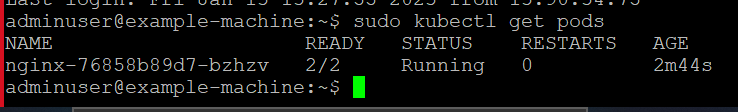
****

1. Use the terraform manifest to build a server in azure
2. Once build use the null provisioner to install kind kubernetes cluster.
3. Deploy your helm chart using Jenkins

****

1. Deploy ingress or istio controller and put your application behind the same

[savitrinikhita/c-terraform (github.com)](https://github.com/savitrinikhita/c-terraform)



****

****

resource "null\_resource" "null\_copy\_ssh" {

    depends\_on = [

    azurerm\_linux\_virtual\_machine.example

  ]

  connection {

    type = "ssh"

    host = azurerm\_linux\_virtual\_machine.example.public\_ip\_address

    user = "adminuser"

    private\_key = file("/var/lib/jenkins/.ssh/id\_rsa")

  }

  provisioner "remote-exec" {

    inline = [

        "curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.17.0/kind-linux-amd64",

        "chmod +x ./kind",

        "sudo mv ./kind /usr/local/bin/kind",

        "sudo apt-get update",

        "sudo apt-get install -y ca-certificates curl gnupg lsb-release",

        "sudo mkdir -p  /etc/apt/keyrings",

        "curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg",

        " echo \"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable\" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null",

        "sudo apt-get update",

        "sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin",

        "sudo kind create cluster",

        "sudo apt-get install -y ca-certificates curl",

        "sudo apt-get install -y apt-transport-https",

        "sudo curl -fsSLo /etc/apt/keyrings/kubernetes-archive-keyring.gpg https://packages.cloud.google.com/apt/doc/apt-key.gpg",

        "echo \"deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main\" | sudo tee /etc/apt/sources.list.d/kubernetes.list",

        "sudo apt-get update",

        "sudo apt-get install -y kubectl",

        "curl -fsSL -o get\_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3",

        "chmod 700 get\_helm.sh",

        "./get\_helm.sh",

        "sudo kubectl get nodes",

        "curl -L https://istio.io/downloadIstio | sh -",

        "cd istio-1.16.1/bin",

        "sudo cp istioctl /usr/bin/",

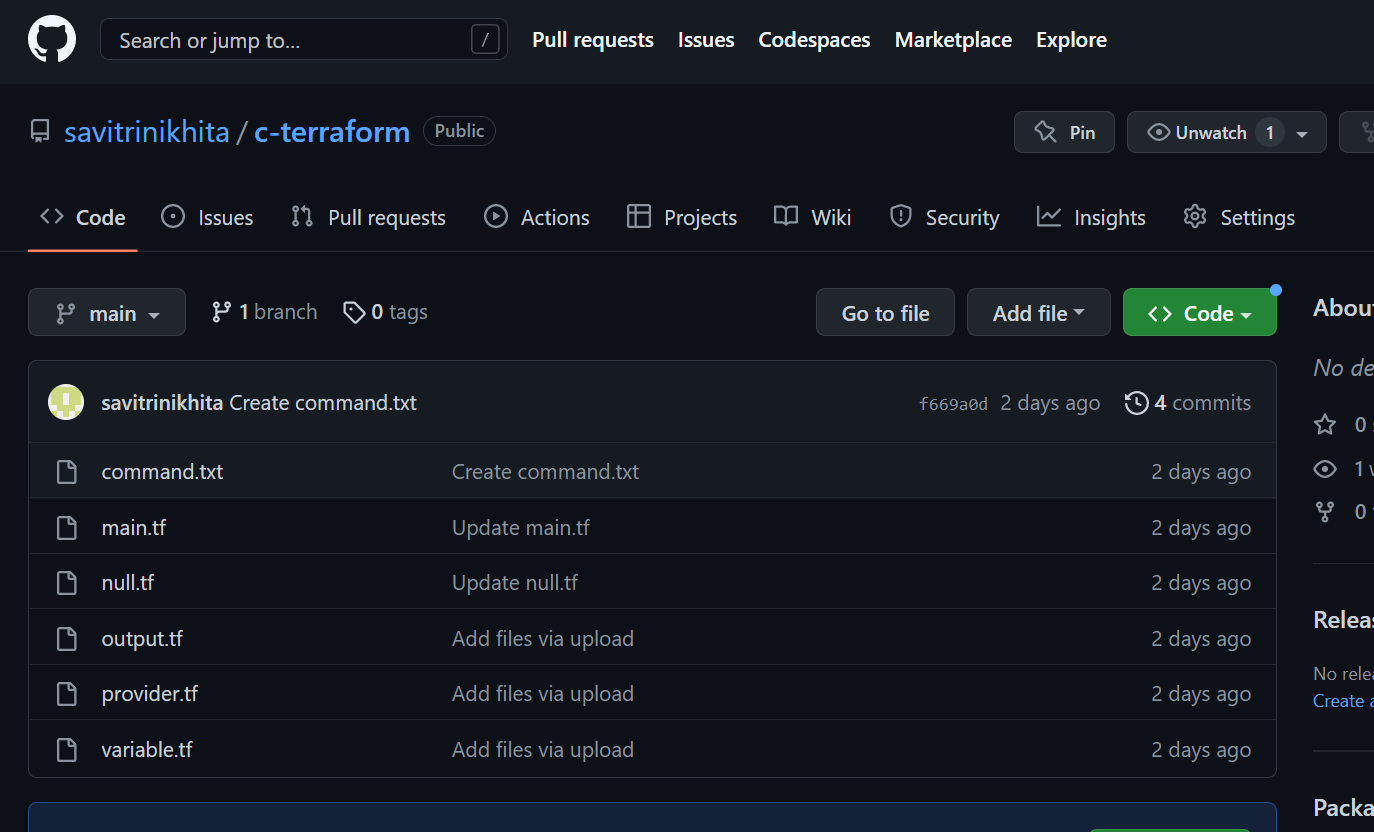
        "sudo istioctl install --set profile=demo -y",

        "sudo kubectl label namespace default istio-injection=enabled"

    ]

  }

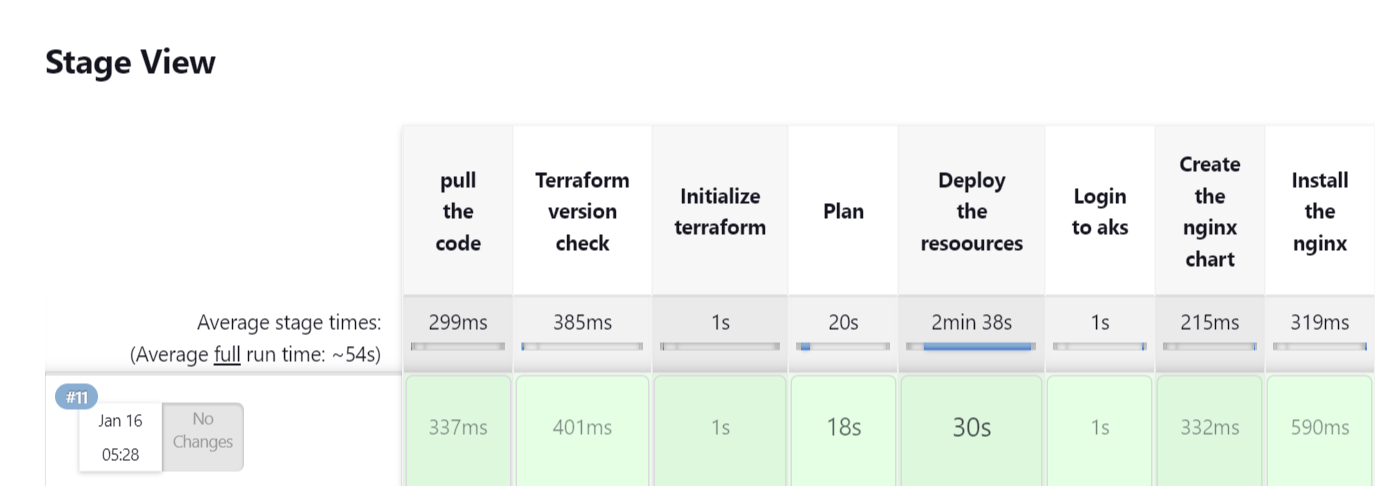
}

****

**Part-6:**

1. use terraform to provision AKS and associated resources in Azure, then sample created helm created(in previous module/exercise) to deploy AKS cluster, use Jenkins as orchestrator, rest all is fine

[savitrinikhita/aks-terraform (github.com)](https://github.com/savitrinikhita/aks-terraform)

****

resource "azurerm\_resource\_group" "example" {

  name     = "example-aks"

  location = "eastus"

}

resource "azurerm\_kubernetes\_cluster" "example" {

  name                = "example-aks1"

  location            = azurerm\_resource\_group.example.location

  resource\_group\_name = azurerm\_resource\_group.example.name

  dns\_prefix          = "exampleaks1"

  default\_node\_pool {

    name       = "default"

    node\_count = 1

    vm\_size    = "Standard\_D2\_v2"

  }

  identity {

    type = "SystemAssigned"

  }

  tags = {

    Environment = "Production"

  }

}

output "client\_certificate" {

  value     = azurerm\_kubernetes\_cluster.example.kube\_config.0.client\_certificate

  sensitive = true

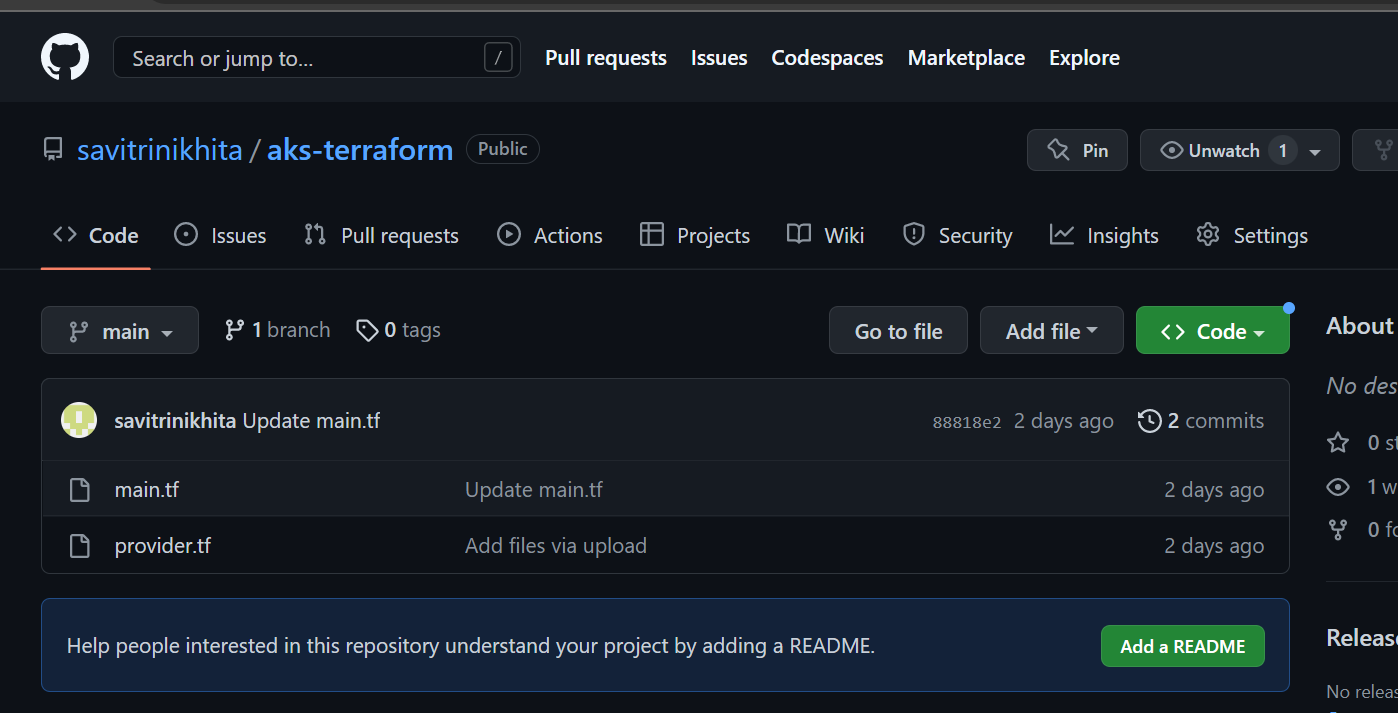
}

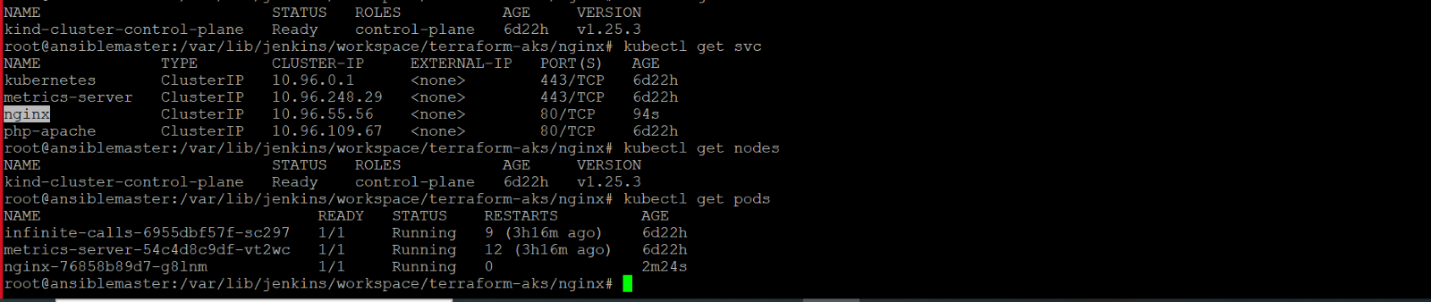
output "kube\_config" {

  value = azurerm\_kubernetes\_cluster.example.kube\_config\_raw

  sensitive = true

}

****

****