Flask App - Jenkins CI/CD & K8s Deployment

Creating a Flask Application and Pushing to GitHub

Step 1: Create the Flask Application

1. Create a simple Flask application with basic functionality.
2. Ensure your project has the necessary files, including app.py, requirements.txt, and any other needed configurations.

Step 2: Initialize Git Repository

1. Initialize a Git repository in your project folder:

   ```
   git init
   ```

2. Add your files to the repository:

   ```
   git add .
   ```

3. Commit the files:

   ```
   git commit -m "Initial commit of Flask app"
   ```

Step 3: Push the Application to GitHub

1. Create a new repository on GitHub.
2. Add the remote repository:

   ```
   git remote add origin https://github.com/your-username/your-repo.git
   ```

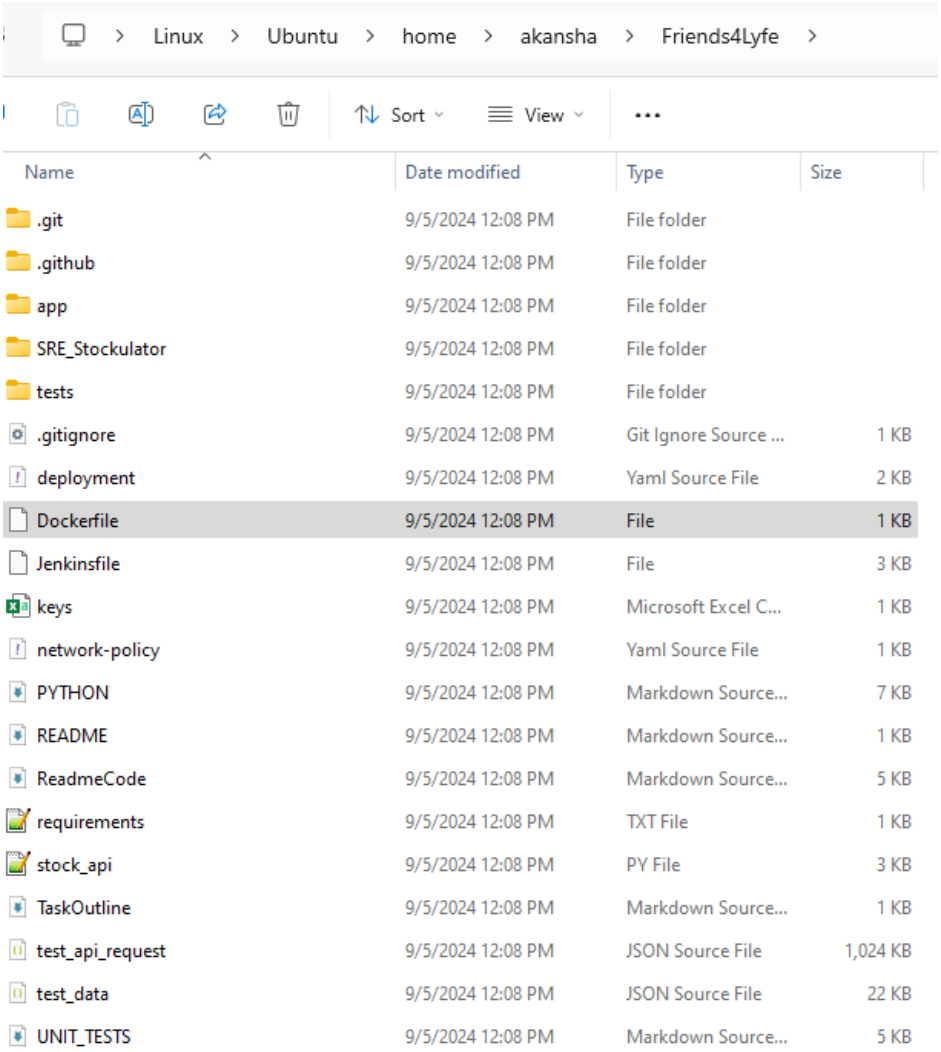
3. Push your code to GitHub:

   ```
   git push -u origin main
   ```

Dockerize the Application

Step 1: Create a Dockerfile

1. Add a Dockerfile to your project root.



Step 2: Build the Docker Image

1. Build the Docker image using the following command:

   docker build -t stock-calculator .

   - **docker build**: Builds a Docker image from the Dockerfile in the current directory.
   - **-t stock-calculator**: Tags the image with the name stock-calculator.

```
akansha@Pawan:~/Friends4Lyfe$ docker build -t stock-calculator-final .
[+] Building 3.1s (13/13) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 800B
 => [internal] load metadata for docker.io/library/python:3.12-slim
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load build context
 => => transferring context: 1.86MB
 => [1/8] FROM docker.io/library/python:3.12-slim@sha256:cec3038ab6478f8c170b2f27174d74a2a6ff4d4fccb4bebacbbb5793e379c20f
 => CACHED [2/8] WORKDIR /app
 => CACHED [3/8] RUN apt-get update && apt-get install -y sqlite3 libsqlite3-dev
 => CACHED [4/8] COPY requirements.txt .
 => CACHED [5/8] RUN pip install --no-cache-dir -r requirements.txt
 => [6/8] COPY . .
 => [7/8] RUN mkdir -p /app/db && chmod -R 755 /app/db
 => [8/8] RUN mkdir -p /app/static && chmod -R 777 /app/static
 => exporting to image
 => => exporting layers
 => => writing image sha256:dd67b22aae297e3147b9a9aba5571c40d442063420bd46dd27b5a4740c97bd10
 => => naming to docker.io/library/stock-calculator-final
akansha@Pawan:~/Friends4Lyfe$ docker run -p 5000:5000 stock-calculator-final
```

2. Run a Docker image using the following command:

   docker run -p 5000:5000 stock-calculator-final

   - **docker run:** This command creates and starts a container from a specified image.
   - **-p 5000:5000**: This option maps port 5000 of your local machine (host) to port 5000 of the Docker container. This is necessary because Flask runs on port 5000 by default. It allows you to access the Flask application in the container via http://localhost:5000 on your local machine.
   - **stock-calculator-final:** This is the name (or tag) of the Docker image you want to run. Ensure that this image is available locally or on a Docker registry.

```
akansha@Pawan:~/Friends4Lyfe$ docker run -p 5000:5000 stock-calculator-final
 * Serving Flask app 'app/main.py'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
172.17.0.1 - - [05/Sep/2024 11:08:35] "GET / HTTP/1.1" 200 -
[*********************100%***********************]  1 of 1 completed
172.17.0.1 - - [05/Sep/2024 11:08:41] "POST / HTTP/1.1" 200 -
172.17.0.1 - - [05/Sep/2024 11:08:41] "GET /static/plot.png HTTP/1.1" 200 -
[*********************100%***********************]  1 of 1 completed
172.17.0.1 - - [05/Sep/2024 11:08:47] "POST / HTTP/1.1" 200 -
172.17.0.1 - - [05/Sep/2024 11:08:47] "GET /static/plot.png HTTP/1.1" 200 -
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
172.17.0.1 - - [05/Sep/2024 11:08:57] "POST / HTTP/1.1" 200 -
172.17.0.1 - - [05/Sep/2024 11:08:57] "GET /static/plot.png HTTP/1.1" 200 -
```

Step 3: Log into Docker Hub

1. Log into your Docker Hub account:

   docker login

   - This command prompts you to enter your Docker Hub credentials.

Step 4: Tag the Docker Image

1. Tag the image for Docker Hub:

   docker tag stock-calculator:latest akanshapal/stock-calculator:latest

   - **docker tag**: Tags an existing Docker image.
   - **akanshapal/stock-calculator:latest**: Tags the image using your Docker Hub username and repository name.

Step 5: Push the Docker Image to Docker Hub

1. Push the image to your Docker Hub repository:

   docker push akanshapal/stock-calculator:latest

   - **docker push**: Uploads the tagged image to Docker Hub.

```
akansha@Pawan:~/Friends4Lyfe$ docker tag stock-calculator-final:latest akanshapal/stock-calculator-final:latest
akansha@Pawan:~/Friends4Lyfe$ docker push akanshapal/stock-calculator-final:latest
The push refers to repository [docker.io/akanshapal/stock-calculator-final]
0ac218f77711: Pushed
9da33c15bc88: Pushed
126bbecfb2e6: Pushed
a915f683d2a1: Layer already exists
fd8e4a736fa3: Layer already exists
2335d63c6334: Layer already exists
9f24d56397f1: Layer already exists
b28d7eb1de61: Layer already exists
a0a1e3b9f056: Layer already exists
7392a6b0f7cb: Layer already exists
34e6cc4b0ffc: Layer already exists
8e2ab394fabf: Layer already exists
latest: digest: sha256:82d12f37926e3eb18680244d03d42c6e0cc857410e4030df252f8b2f03c9b85a size: 2831
akansha@Pawan:~/Friends4Lyfe$ ls -lrt
```

Step 6: Docker container checks (Optional)

1. Check running containers:

   docker ps

   The output of docker ps typically includes the following columns:

   - **CONTAINER ID:** The unique identifier for the container.
   - **IMAGE:** The Docker image that the container is running.
   - **COMMAND:** The command that was used to start the container.
   - **CREATED:** The time when the container was created.
   - **STATUS:** The current status of the container (e.g., "Up 5 minutes").
   - **PORTS:** The ports that are mapped between the host and the container.
   - **NAMES:** The name assigned to the container.

```
akansha@Pawan:~/Friends4Lyfe$ docker ps
CONTAINER ID   IMAGE                   COMMAND                  CREATED          STATUS          PORTS                    NAMES
84b3d5aca52a   stock-calculator-final  "flask run --host 0.…"   26 minutes ago   Up 26 minutes   0.0.0.0:5000->5000/tcp   hopeful_engelbart
```

2. Stop or kill running docker if required or port used error:

   docker stop <container_id>

```
akansha@Pawan:~/Friends4Lyfe$ docker stop 84b3d5aca52a
84b3d5aca52a
```

Kubernetes Setup with EKS

Step 1: Setup AWS CLI, KUBECTL and EKSCTL

- **AWS CLI**: Install and configure to interact with AWS services.
- **kubectl**: Install to manage Kubernetes clusters.
- **eksctl**: Install to simplify EKS cluster management. This command sets up an EKS cluster named cluster1 with a node group named ng1.

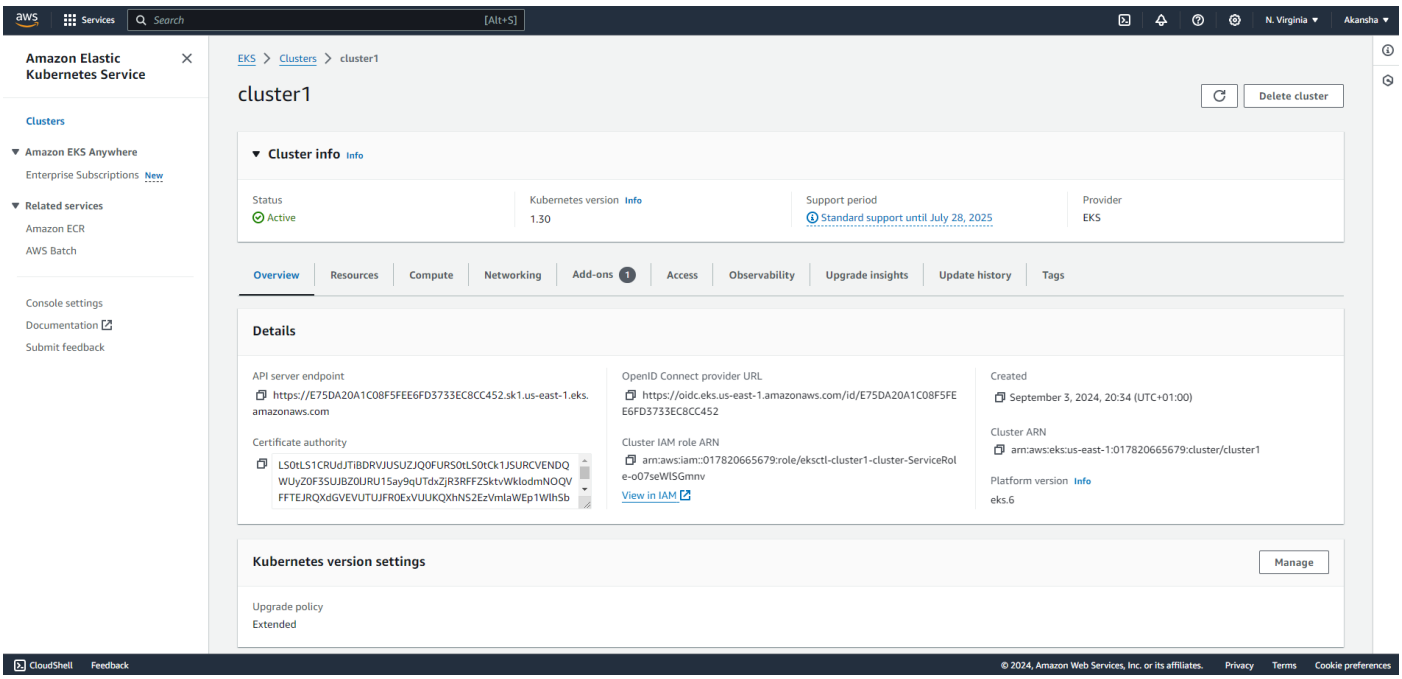
Follow the installation instructions available on https://docs.aws.amazon.com/eks/latest/userguide/setting-up.html.

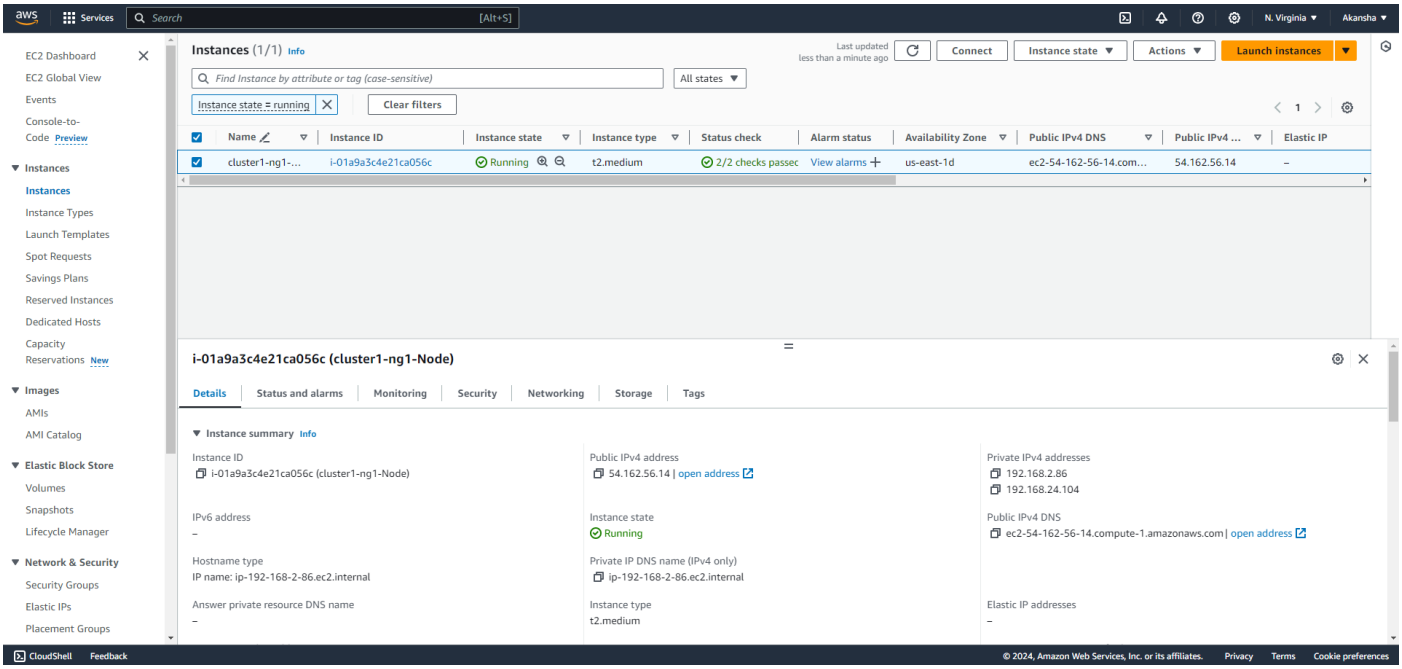
Step 2: Create an EKS Cluster

1. Use the following command to create an EKS cluster:

   eksctl create cluster -n cluster1 --nodegroup-name ng1 --region us-east-1 --node-type t2.medium --nodes 1

   - This command sets up an EKS cluster named cluster1 with a node group named ng1.
   - **--name cluster1:** This is the name of your EKS cluster.
   - **--nodegroup-name ng1:** Name of the node group.
   - **--region us-east-1:** AWS region where the cluster will be created.
   - **--node-type t2.medium:** Type of EC2 instances for the worker nodes. You can choose other instance types based on your needs.
   - **--nodes 1:** Number of worker nodes to start with. You can adjust this number based on your requirements.
   - **--nodes-min 1:** Minimum number of worker nodes for auto-scaling.
   - **--nodes-max 3:** Maximum number of worker nodes for auto-scaling.





Step 3: Setup Jenkins Namespace and Service Account

1. Create a namespace for Jenkins:

   kubectl create namespace jenkins

2. Create a service account for Jenkins:

   kubectl create sa jenkins -n jenkins

3. Create a token for the Jenkins service account:

   kubectl create token jenkins -n jenkins --duration=8760h

4. Create a RoleBinding to grant Jenkins admin access:

   kubectl create rolebinding jenkins-admin-binding --clusterrole=admin --serviceaccount=jenkins:jenkins --namespace=Jenkins

5. Lists all the nodes in your Kubernetes cluster, showing their status and basic information:

   kubectl get nodes

```
C:\Users\pawan>kubectl get nodes
NAME                          STATUS   ROLES    AGE    VERSION
ip-192-168-2-86.ec2.internal  Ready    <none>   3h44m  v1.30.2-eks-1552ad0
```

6. Lists all the pods running in your cluster within the default namespace:

   kubectl get pods

```
C:\Users\pawan>kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
stock-calculator-7d686bf454-6s4bx 1/1     Running   0          48m
```

7. Lists all the pods running in your cluster within the default namespace:

   kubectl config view

```
C:\Users\pawan>kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://E75DA20A1C08F5FEE6FD3733EC8CC452.sk1.us-east-1.eks.amazonaws.com
  name: arn:aws:eks:us-east-1:017820665679:cluster/cluster1
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://E75DA20A1C08F5FEE6FD3733EC8CC452.sk1.us-east-1.eks.amazonaws.com
  name: cluster1.us-east-1.eksctl.io
contexts:
- context:
    cluster: arn:aws:eks:us-east-1:017820665679:cluster/cluster1
    user: arn:aws:eks:us-east-1:017820665679:cluster/cluster1
  name: arn:aws:eks:us-east-1:017820665679:cluster/cluster1
- context:
    cluster: cluster1.us-east-1.eksctl.io
    user: iam-root-account@cluster1.us-east-1.eksctl.io
  name: iam-root-account@cluster1.us-east-1.eksctl.io
current-context: arn:aws:eks:us-east-1:017820665679:cluster/cluster1
kind: Config
preferences: {}
users:
- name: arn:aws:eks:us-east-1:017820665679:cluster/cluster1
  user:
    exec:
      apiVersion: client.authentication.k8s.io/v1beta1
      args:
      - --region
      - us-east-1
      - eks
      - get-token
      - --cluster-name
      - cluster1
      - --output
      - json
      command: aws
      env: null
      interactiveMode: IfAvailable
      provideClusterInfo: false
- name: iam-root-account@cluster1.us-east-1.eksctl.io
  user:
    exec:
      apiVersion: client.authentication.k8s.io/v1beta1
      args:
      - eks
      - get-token
      - --output
      - json
```

Step 4: Update Kubeconfig and Access the Cluster

1. Update your kubeconfig file to access the cluster:

   aws eks --region us-east-1 update-kubeconfig --name <your-cluster-name>

2. Verify access to the cluster by viewing contexts:

kubectl config get-contexts

3. Switch to the desired context:

   kubectl config use-context <your-context-name>

Step 5: Cluster and Deployment status (Optional)

1. Queries the status of your specified EKS cluster:

   aws eks --region us-east-1 describe-cluster --name <your-cluster-name> --query "cluster.status"

```
C:\Users\pawan>aws eks --region us-east-1 describe-cluster --name cluster1 --query "cluster.status"
"ACTIVE"
```

2. Updates your local kubeconfig file to include the context for your EKS cluster:

   aws eks --region us-east-1 update-kubeconfig --name <your-cluster-name>

3. Detailed information about each node in your cluster, including resources, conditions, events, and other metadata:

   kubectl describe nodes

```
C:\Users\pawan>kubectl describe nodes
Name:               ip-192-168-2-86.ec2.internal
Roles:              <none>
Labels:             alpha.eksctl.io/cluster-name=cluster1
                    alpha.eksctl.io/nodegroup-name=ng1
                    beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/instance-type=t2.medium
                    beta.kubernetes.io/os=linux
                    eks.amazonaws.com/capacityType=ON_DEMAND
                    eks.amazonaws.com/nodegroup=ng1
                    eks.amazonaws.com/nodegroup-image=ami-0165f4617734ddca9
                    eks.amazonaws.com/sourceLaunchTemplateId=lt-0097b06e2b6a891c0
                    eks.amazonaws.com/sourceLaunchTemplateVersion=1
                    failure-domain.beta.kubernetes.io/region=us-east-1
                    failure-domain.beta.kubernetes.io/zone=us-east-1d
                    k8s.io/cloud-provider-aws=330d69609712a5e5a86f6cbe2401b011
                    kubernetes.io/arch=amd64
                    kubernetes.io/hostname=ip-192-168-2-86.ec2.internal
                    kubernetes.io/os=linux
                    node.kubernetes.io/instance-type=t2.medium
                    topology.k8s.aws/zone-id=use1-az6
                    topology.kubernetes.io/region=us-east-1
                    topology.kubernetes.io/zone=us-east-1d
Annotations:        alpha.kubernetes.io/provided-node-ip: 192.168.2.86
                    node.alpha.kubernetes.io/ttl: 0
                    volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp:  Thu, 05 Sep 2024 09:17:24 +0100
Taints:             <none>
Unschedulable:      false
Lease:
  HolderIdentity:  ip-192-168-2-86.ec2.internal
  AcquireTime:     <unset>
  RenewTime:       Thu, 05 Sep 2024 13:13:18 +0100
Conditions:
  Type             Status  LastHeartbeatTime                 LastTransitionTime                Reason                       Message
  ----             ------  -----------------                 ------------------                ------                       -------
  MemoryPressure   False   Thu, 05 Sep 2024 13:13:14 +0100   Thu, 05 Sep 2024 09:17:22 +0100   KubeletHasSufficientMemory   kubelet has sufficient memory available
  DiskPressure     False   Thu, 05 Sep 2024 13:13:14 +0100   Thu, 05 Sep 2024 09:17:22 +0100   KubeletHasNoDiskPressure     kubelet has no disk pressure
  PIDPressure      False   Thu, 05 Sep 2024 13:13:14 +0100   Thu, 05 Sep 2024 09:17:22 +0100   KubeletHasSufficientPID      kubelet has sufficient PID available
  Ready            True    Thu, 05 Sep 2024 13:13:14 +0100   Thu, 05 Sep 2024 09:17:34 +0100   KubeletReady                 kubelet is posting ready status
Addresses:
  InternalIP:   192.168.2.86
  ExternalIP:   54.162.56.14
  InternalDNS:  ip-192-168-2-86.ec2.internal
  Hostname:     ip-192-168-2-86.ec2.internal
  ExternalDNS:  ec2-54-162-56-14.compute-1.amazonaws.com
Capacity:
  cpu:                2
  ephemeral-storage:  83873772Ki
  hugepages-2Mi:      0
```

4. Detailed information about each deployment in the namespace, including replica status, strategy, selectors, and recent events:

   kubectl describe deployments

```
C:\Users\pawan>kubectl describe deployments
Name:                   stock-calculator
Namespace:              default
CreationTimestamp:      Tue, 03 Sep 2024 22:11:01 +0100
Labels:                 app=stock-calculator
Annotations:            deployment.kubernetes.io/revision: 7
Selector:               app=stock-calculator
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=stock-calculator
  Containers:
   stock-calculator:
    Image:         akanshapal/stock-calculator-final:latest
    Port:          5000/TCP
    Host Port:     0/TCP
    Environment:   <none>
    Mounts:        <none>
  Volumes:         <none>
  Node-Selectors:  <none>
  Tolerations:     <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  stock-calculator-f97f85bfc (0/0 replicas created), stock-calculator-f5dbc8564 (0/0 replicas created)
NewReplicaSet:   stock-calculator-7d686bf454 (1/1 replicas created)
Events:          <none>
```

5. Monitors the rollout status of the stock-calculator deployment:

   kubectl rollout status deployment/stock-calculator

```
C:\Users\pawan>kubectl rollout status deployment/stock-calculator
deployment "stock-calculator" successfully rolled out
```

6. Detailed information about the pod, including its status, events, container states, and resource usage:

   kubectl describe pod <pod-name>

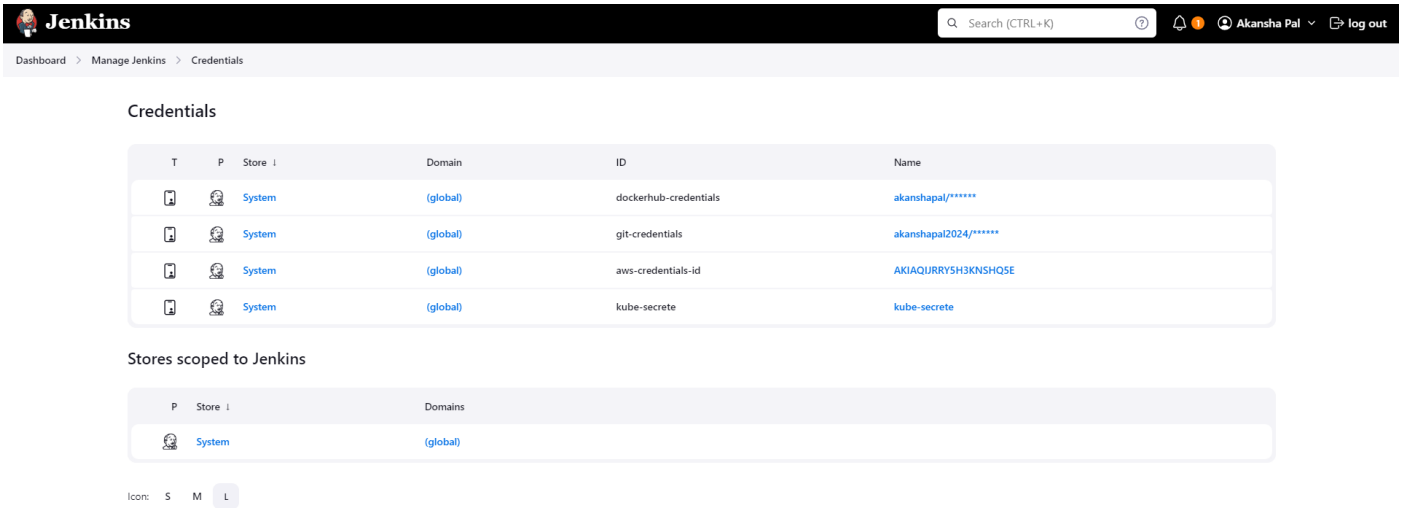
```
C:\Users\pawan>kubectl describe pod stock-calculator-7d686bf454-6s4bx
Name:             stock-calculator-7d686bf454-6s4bx
Namespace:        default
Priority:         0
Service Account:  default
Node:             ip-192-168-2-86.ec2.internal/192.168.2.86
Start Time:       Thu, 05 Sep 2024 12:11:29 +0100
Labels:           app=stock-calculator
                  pod-template-hash=7d686bf454
Annotations:      <none>
Status:           Running
IP:               192.168.27.148
IPs:
  IP:           192.168.27.148
Controlled By:  ReplicaSet/stock-calculator-7d686bf454
Containers:
  stock-calculator:
    Container ID:   containerd://7c82b21a42e3a1bb94be93ca8475269539fdf492defb02cbaf6b665d774092c1
    Image:          akanshapal/stock-calculator-final:latest
    Image ID:       docker.io/akanshapal/stock-calculator-final@sha256:0fa4b16a9f5f961d659b0ded650ba9cbae23f8a349c4c91225f5c84ac2ec87ed
    Port:           5000/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Thu, 05 Sep 2024 12:11:30 +0100
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-4sxxm (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       True
  ContainersReady             True
  PodScheduled                True
Volumes:
  kube-api-access-4sxxm:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:                      <none>
```

CI/CD Pipeline Setup with Jenkins

Step 1: Adding Credentials in Jenkins

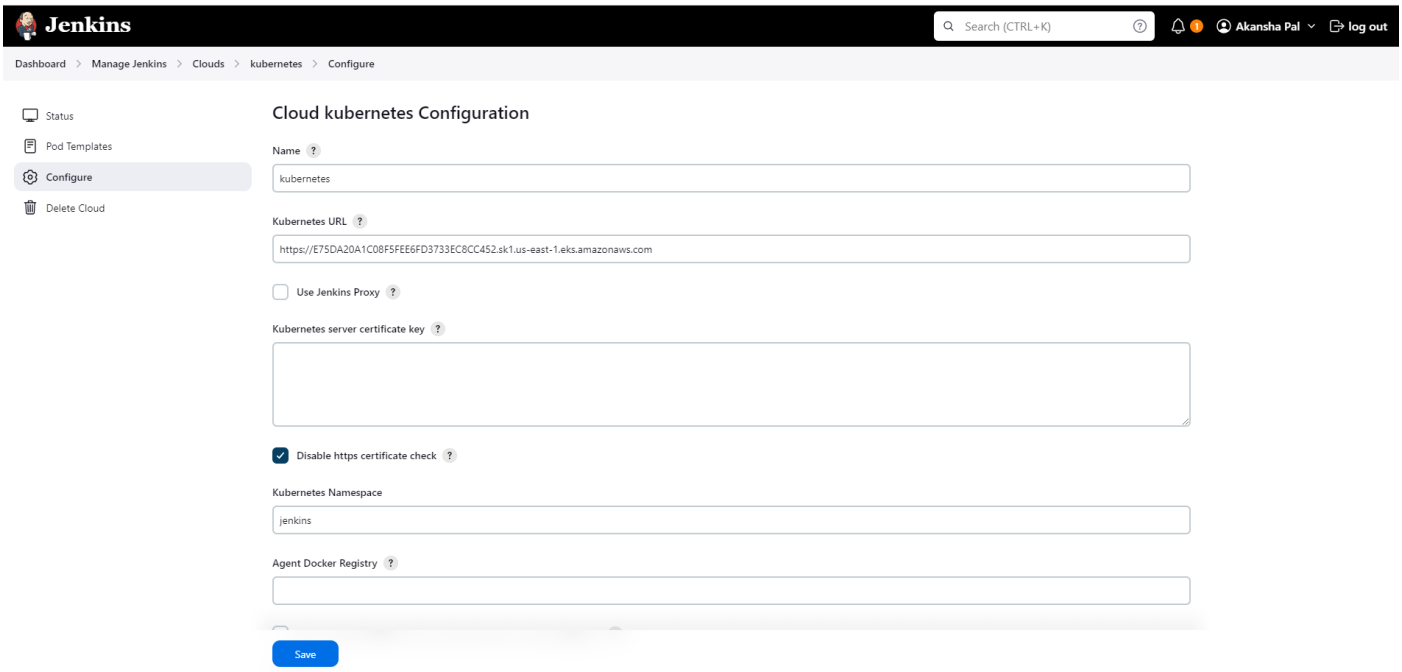
1. Go to the Jenkins Dashboard.
2. Access **Manage Credentials**:
    - Click on **Manage Jenkins > Manage Credentials**.
    - Select the appropriate domain (e.g., (global)).
3. Add new credentials for each service:
    - **DockerHub Credentials**:
        - **Kind**: Username with password
        - **ID**: dockerhub-credentials
        - **Username**: <your-dockerhub-username>
        - **Password**: <your-dockerhub-password or access token>
        - **Description**: DockerHub credentials for pushing/pulling images.
    - **GitHub Credentials**:
        - **Kind**: Username with password / Secret text
        - **ID**: github-credentials-id
        - **Username**: <your-github-username>
        - **Password/Secret**: <your-github-password or access token>
        - **Description**: GitHub credentials for repository access.
    - **AWS Credentials**:
        - **Kind**: AWS credentials
        - **ID**: aws-credentials-id
        - **Access Key ID**: <your-aws-access-key-id>
        - **Secret Access Key**: <your-aws-secret-access-key>
        - **Description**: AWS credentials for accessing AWS services.
    - **Kubernetes Credentials**:
        - **Kind**: Secret text
        - **ID**: kube-secret
        - **Secret**: Token generated using:

          kubectl create token jenkins -n jenkins --duration=8760h
        - **Description**: Kubernetes credentials for cluster access.



Step 2: Jenkins Kubernetes Configuration

1. Go to **Manage Jenkins > Clouds**.
2. Set up the Kubernetes connection:
   - **Name**: Kubernetes
   - **Kubernetes URL**: Paste URL from kubectl config view.
   - **Kubernetes Namespace**: jenkins
   - **Credentials**: Select kube-secret from the dropdown.
   - Click **Test Connection** and ensure successful connection.

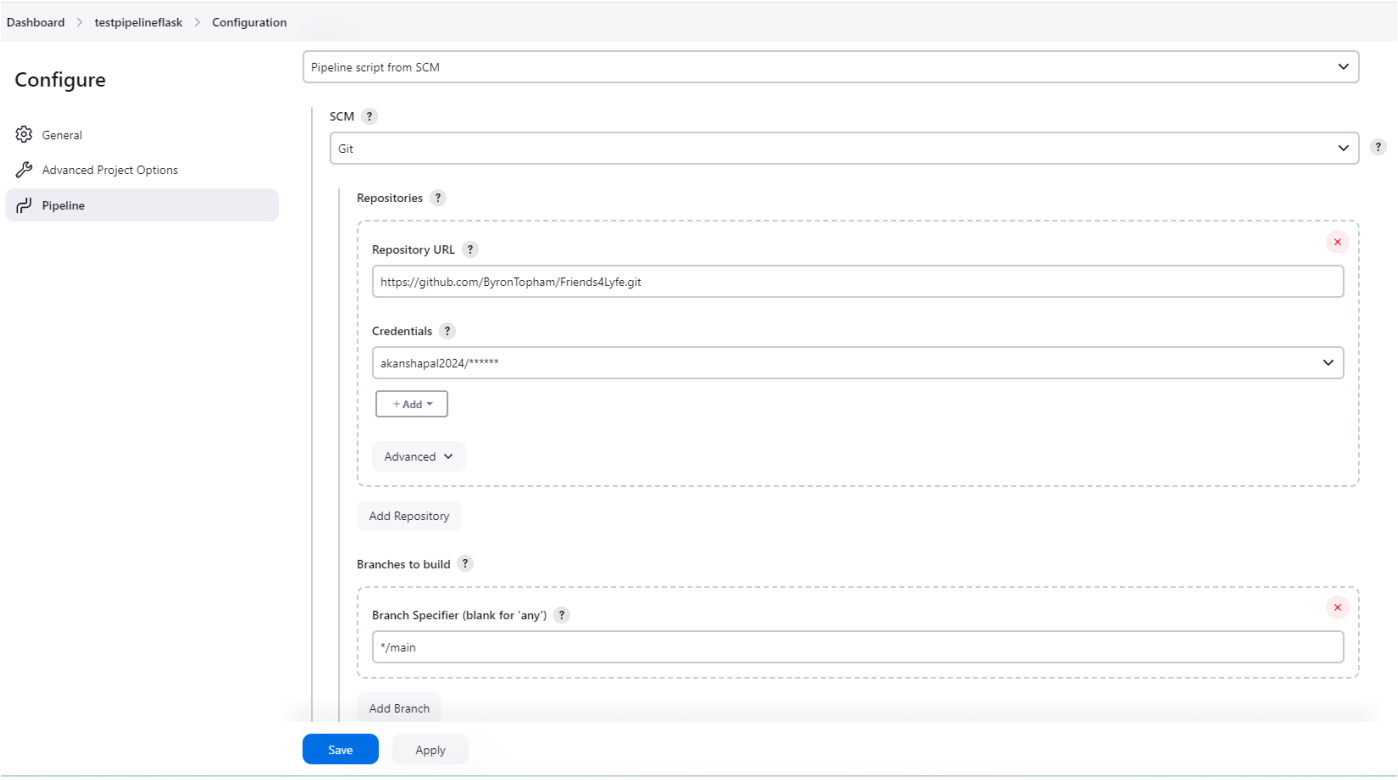


Step 3: Creating a Jenkins Pipeline

1. Create a new pipeline job:
   - Go to **New Item** and select **Pipeline**.
2. Configure the pipeline:
   - Under the **Pipeline** section, set:
     - **Definition**: Pipeline script from SCM

- **SCM**: Git
- **Repository URL**: <your-repo-url>
- **Credentials**: Select the GitHub credentials added earlier.
- **Branch Specifier**: main
- **Script Path**: Jenkinsfile

3. Apply, Save and build the pipeline.



4. Verify the app is deployed by checking the service, look for the EXTERNAL-IP column:

   kubectl get svc stock-calculator-service

```
C:\Users\pawan>kubectl get svc stock-calculator-service
NAME                      TYPE           CLUSTER-IP     EXTERNAL-IP                                                              PORT(S)        AGE
stock-calculator-service  LoadBalancer   10.100.19.62   aa0e69998eed3498fba3076d247c5dcd-669029395.us-east-1.elb.amazonaws.com   80:32288/TCP   39h
```

5. Access the application using the external IP:
   - Open a browser and go to: http://<external-ip>

Not secure aa0e69998eed3498fba3076d247c5dcd-669029395.us-east-1.elb.amazonaws.com

STOCKULATOR

Load Configuration:

None

Load

Stock Name:

AAPL

Start Date:

01/01/2020

End Date:

01/01/2023

Time Step:

1 Day

Color for Stock 1:

Blue

Second Stock Name (optional):

TSLA

Color for Stock 2:

Red

☐ Add Line of Best Fit

Save Configuration As: