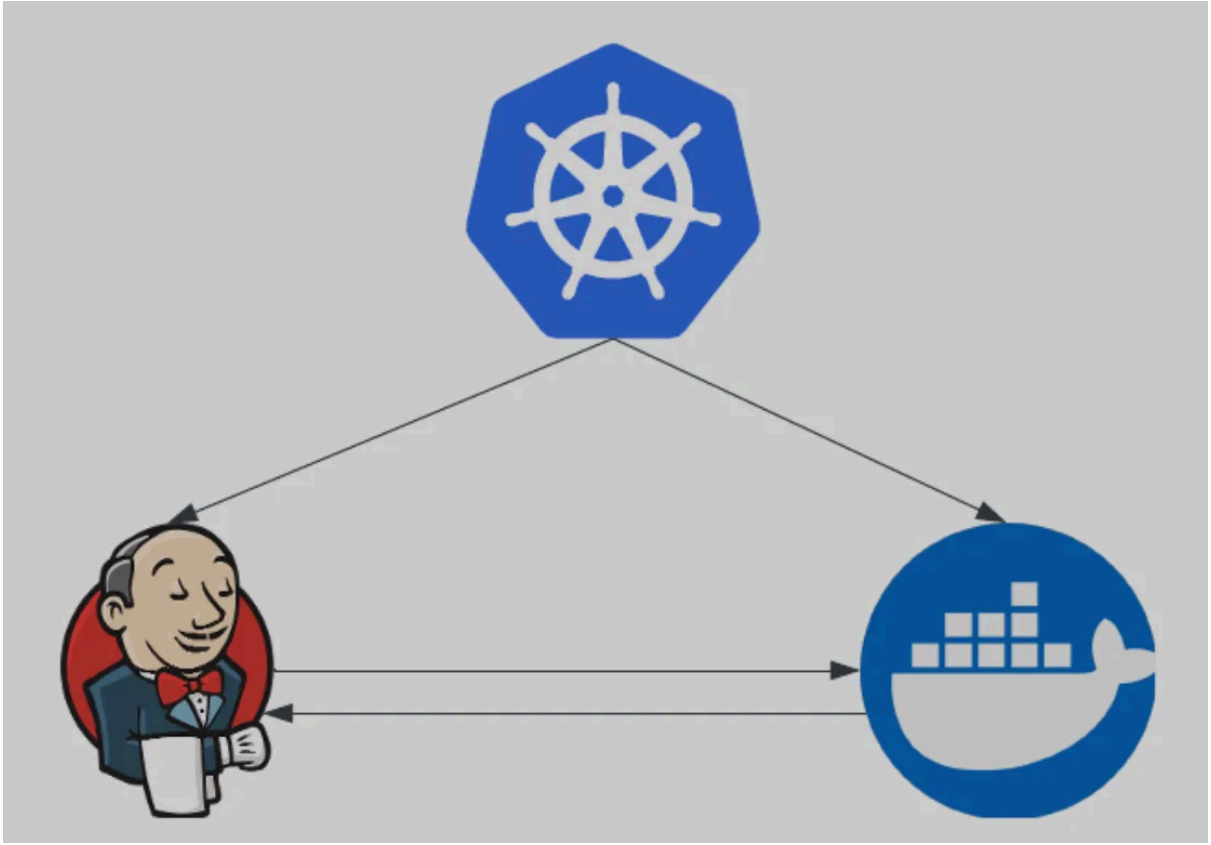
Deploying a Flask Application with Jenkins to a Kubernetes Cluster: CI/CD Pipeline Setup



Introduction

In this guide, you will learn how to automate the deployment of a Flask application using Jenkins, Docker, and Kubernetes in a CI/CD (Continuous Integration and Continuous Deployment) pipeline. This step-by-step tutorial covers everything from setting up Jenkins and Docker to configuring Jenkins, pushing the Dockerized Flask application to DockerHub, and deploying it on Kubernetes

Prerequisites:

- A server with Ubuntu 24.04
- Basic knowledge of Linux commands is required
- Access to a Kubernetes cluster (you can use Minikube, Kind, or DigitalOcean Kubernetes).

Step 1: Installing Jenkins

First, install Jenkins on your Ubuntu server.

Install Java:

```
sudo apt update -y
sudo apt install openjdk-17-jdk -y
```

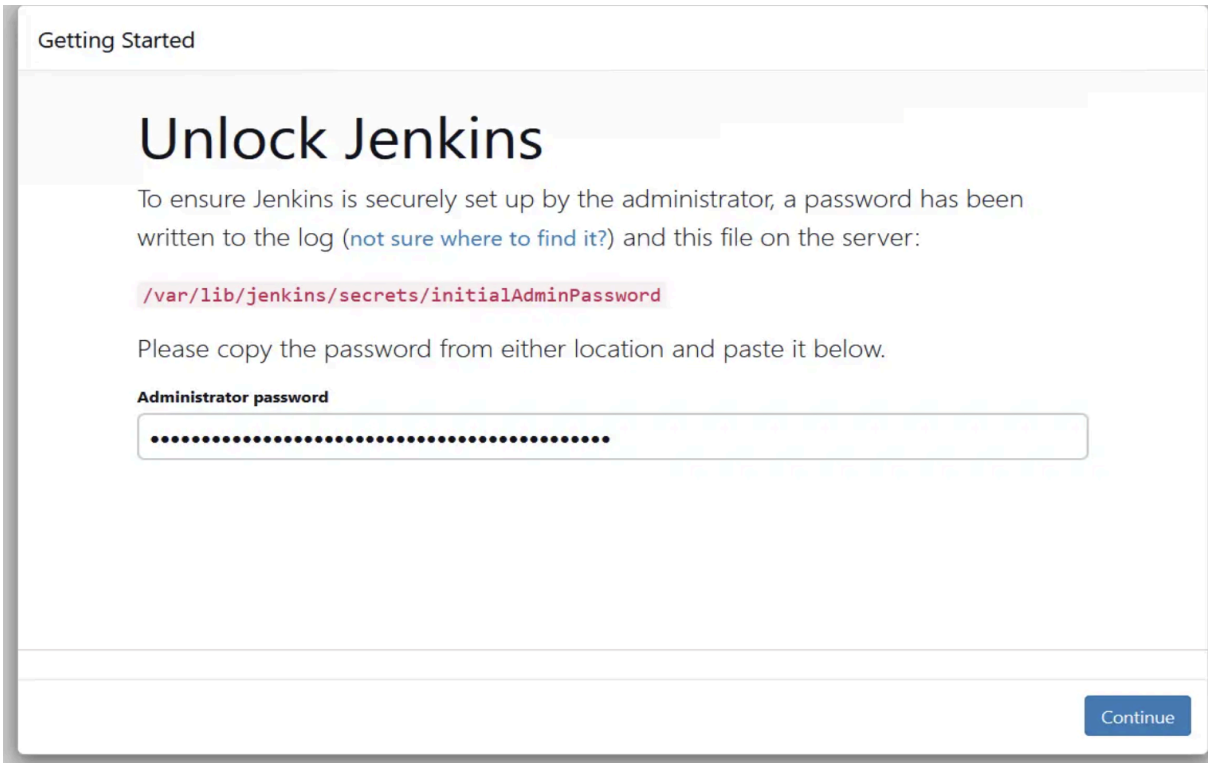
Install Jenkins:

```
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
https://pkg.jenkins.io/debian/jenkins.io-2023.key
echo "deb
[signed-by=/usr/share/keyrings/jenkins-keyring.asc]" \
https://pkg.jenkins.io/debian binary/ | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update -y
sudo apt-get install jenkins -y
```

Start and Enable Jenkins:

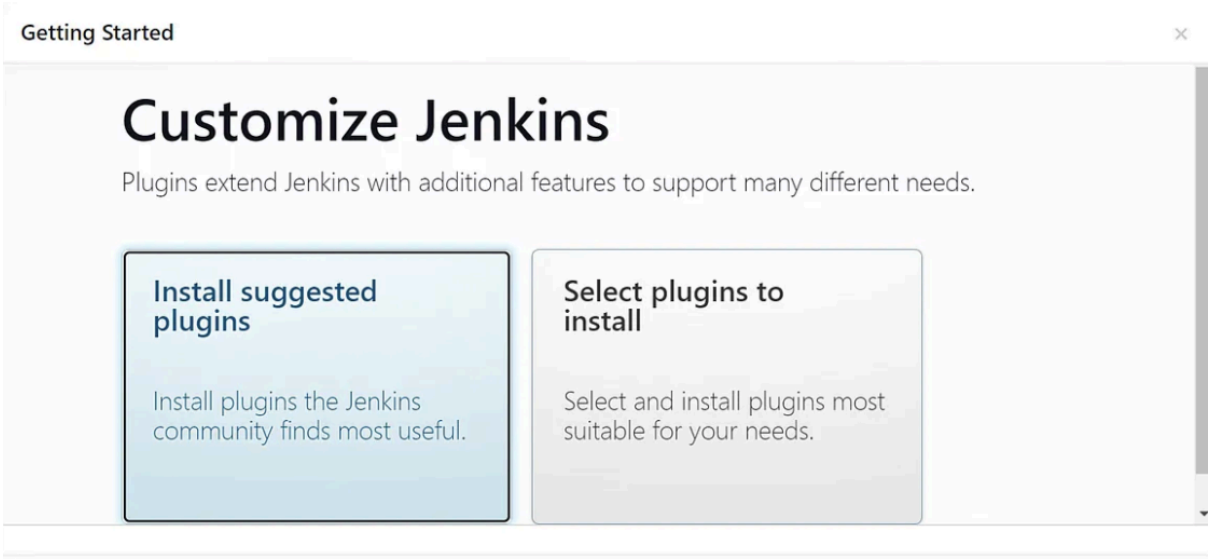
```
sudo systemctl start jenkins
sudo systemctl enable jenkins
```

Access Jenkins: Open your browser and navigate to `http://your-server-ip:8080`. Unlock Jenkins using the initial admin password:

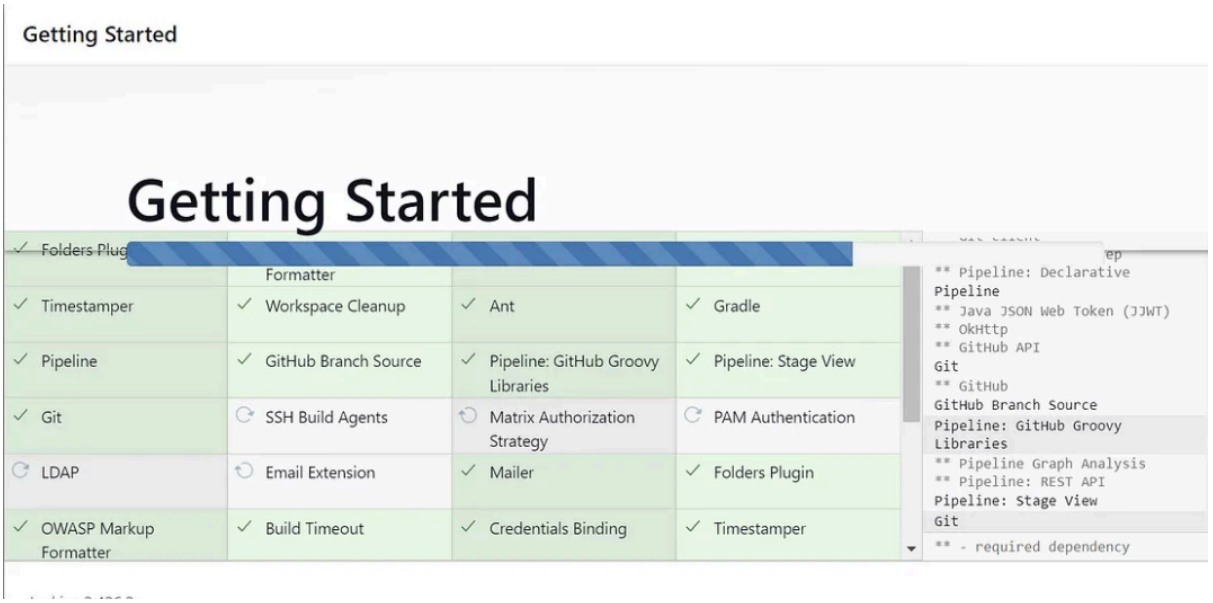


```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

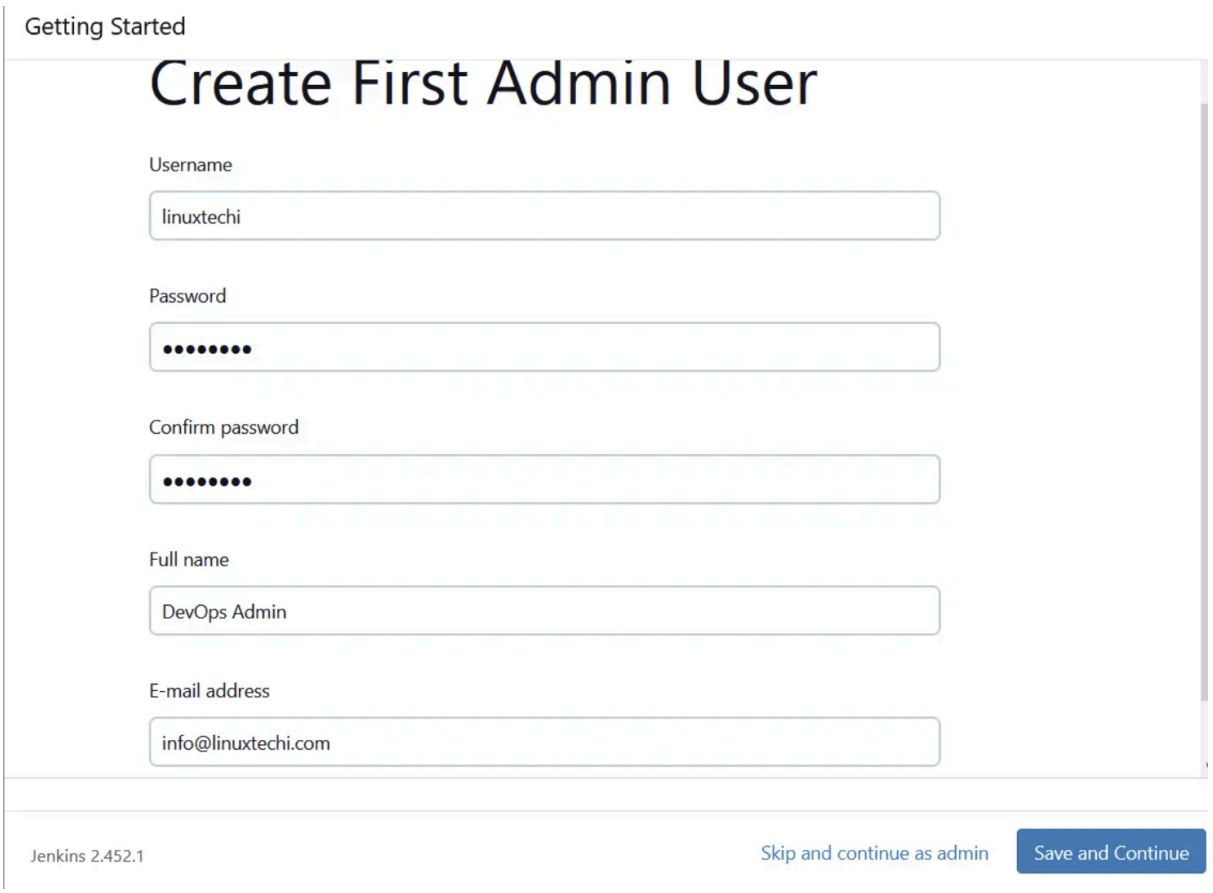
Complete the setup by installing suggested plugins and creating the first admin user.



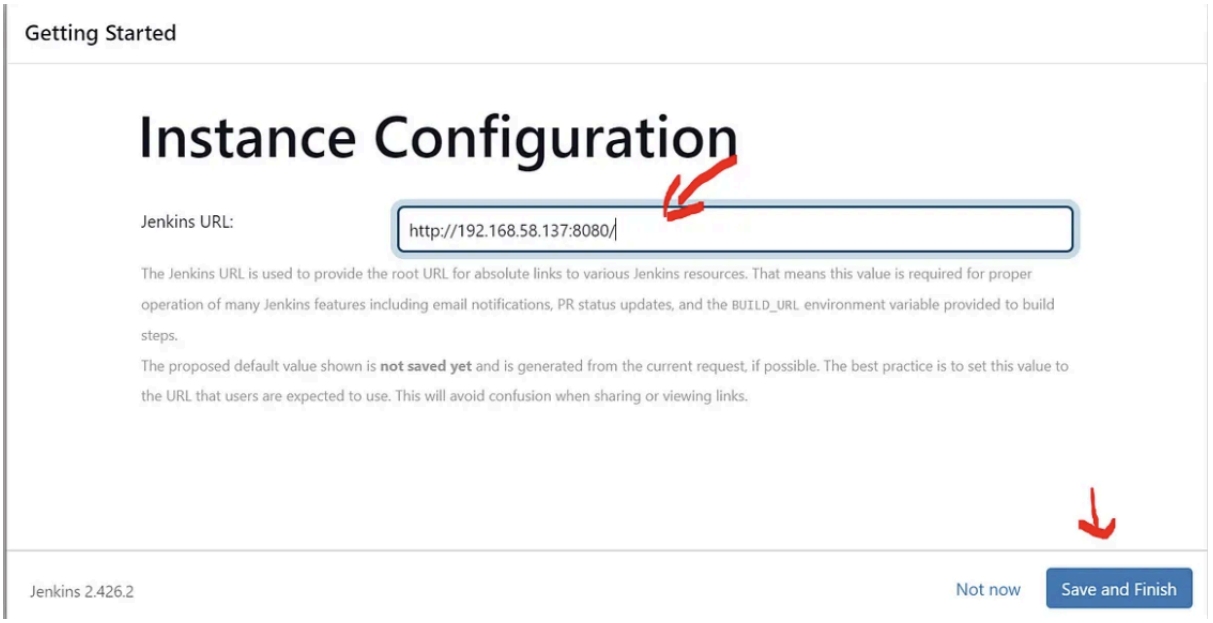
The setup prompts to either **Install suggested plugins** or **Select plugins to install**. It's fine to simply install the suggested plugins



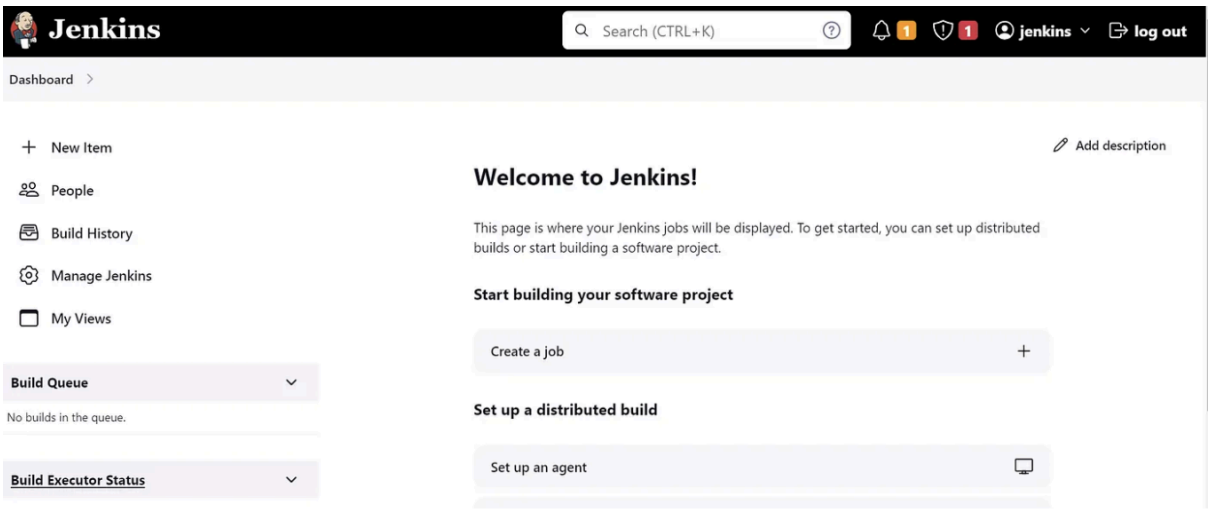
The next step is to the **Create First Admin User**. Enter the credentials you want to use for your Jenkins administrator, then click **Save and Continue**.



Once you specify the Jenkins URL, click **Save and Finish**



You should see a page that says **Jenkins is ready!** Click **Start using Jenkins** to open the Jenkins dashboard



Step 2: Installing Docker

Next, we need Docker to build and manage our containerized applications.

Install Docker:

```
sudo apt-get update -y
sudo apt-get install ca-certificates curl -y
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
/etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
echo \
 "deb [arch=$(dpkg --print-architecture)
signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu \
 $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update -y
sudo apt-get install docker-ce docker-ce-cli containerd.io
docker-buildx-plugin docker-compose-plugin -y
```

Add Jenkins User to Docker Group:

```
sudo systemctl enable docker
sudo usermod -aG docker jenkins
```

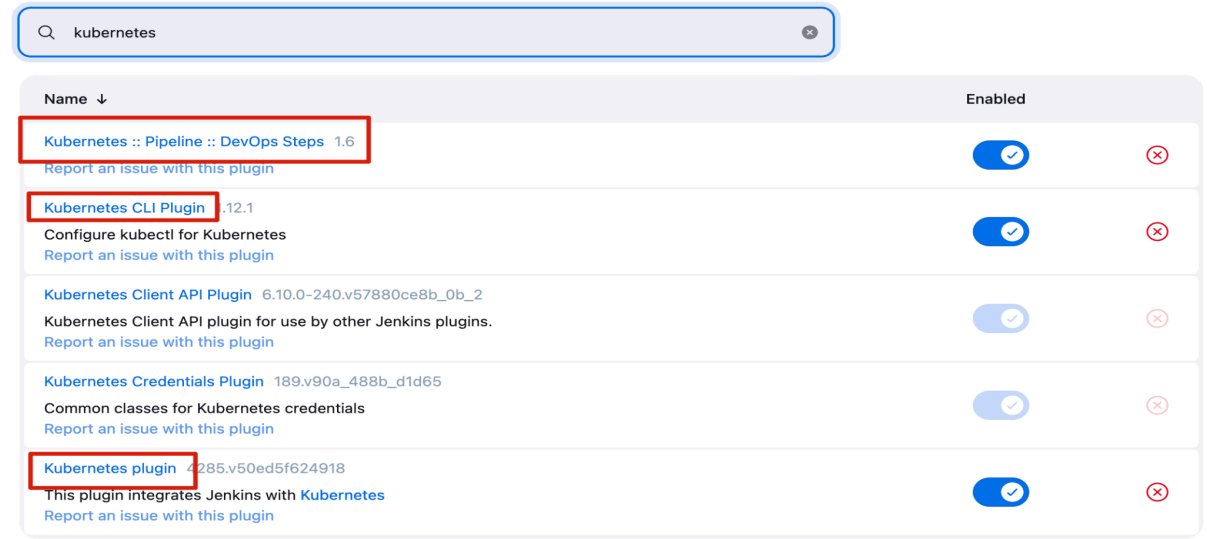
Step 3: Configuring Jenkins for Docker and Kubernetes

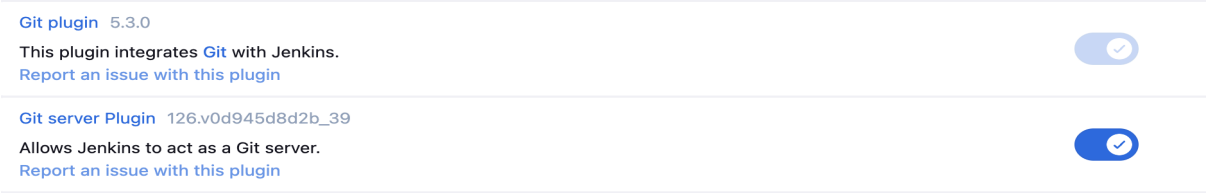
Now that Jenkins and Docker are installed, let's configure Jenkins for Docker and Kubernetes deployments.

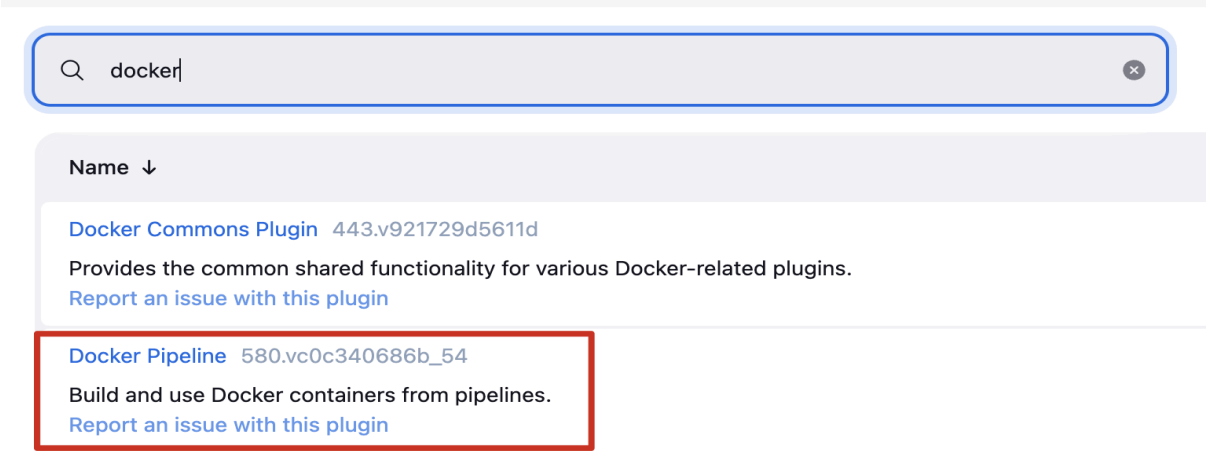
Install Required Plugins:

Go to **Manage Jenkins > Manage Plugins** and install the following plugins:

- **Docker Pipeline**
- **Kubernetes Plugin**
- **Kubernetes Pipeline**
- **Git Plugin**







Configure Docker Hub Credentials in Jenkins:

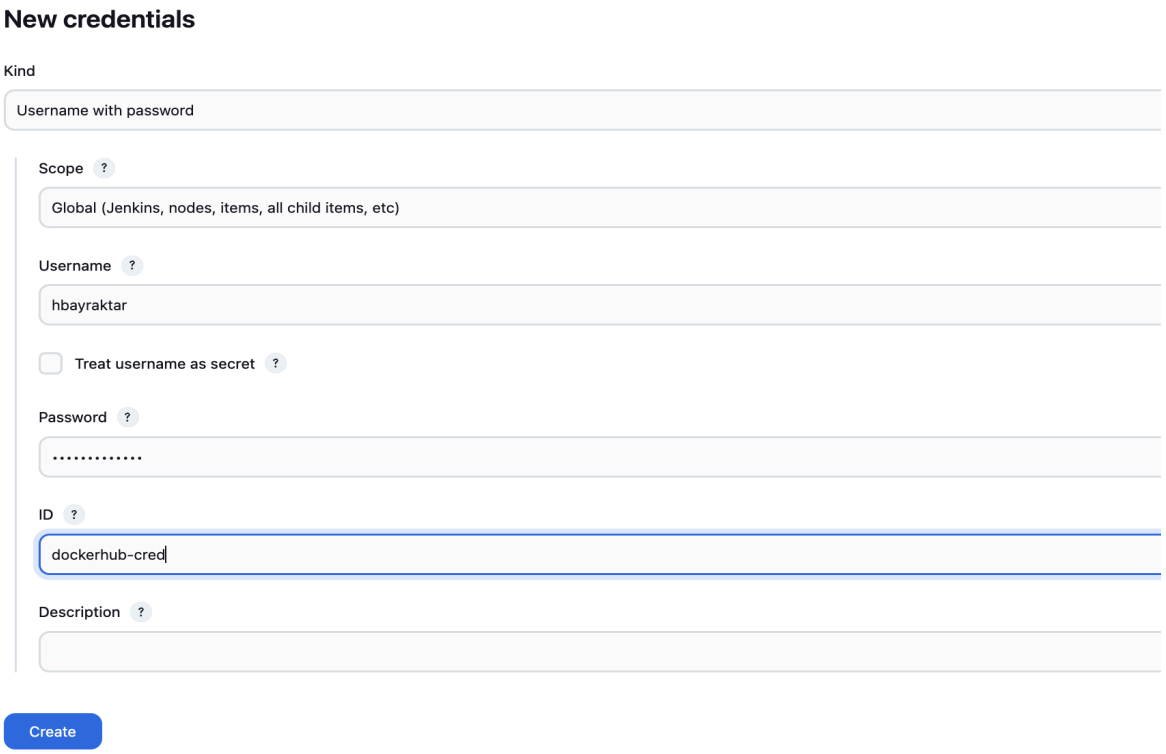
Go to **Manage Jenkins > Credentials> System**>**Global credentials (unrestricted)**

Select **Add Credentials** button

Kind: Username with password

Username: hbayraktar (your dockerhub username)

Password: Dockerhub password



Configure Kubernetes:

Generate secret for Kubernetes service account

Create a kubernetes service account

```
kubectl create serviceaccount jenkins
```

Create a role binding based on the permission needed by the application using below

```
cat <<EOF | kubectl create -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
 name: jenkins-integration
 labels:
   k8s-app: jenkins-image-builder
roleRef:
  apiGroup: rbac.authorization.k8s.io
```

```
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: jenkins
  namespace: default
EOF
```

Extract Service account token using kubectl

```
kubectl get secrets $(kubectl get serviceaccounts jenkins -o
jsonpath='{.secrets[].name}') -o jsonpath='{.data.token}' | base64 -d
```

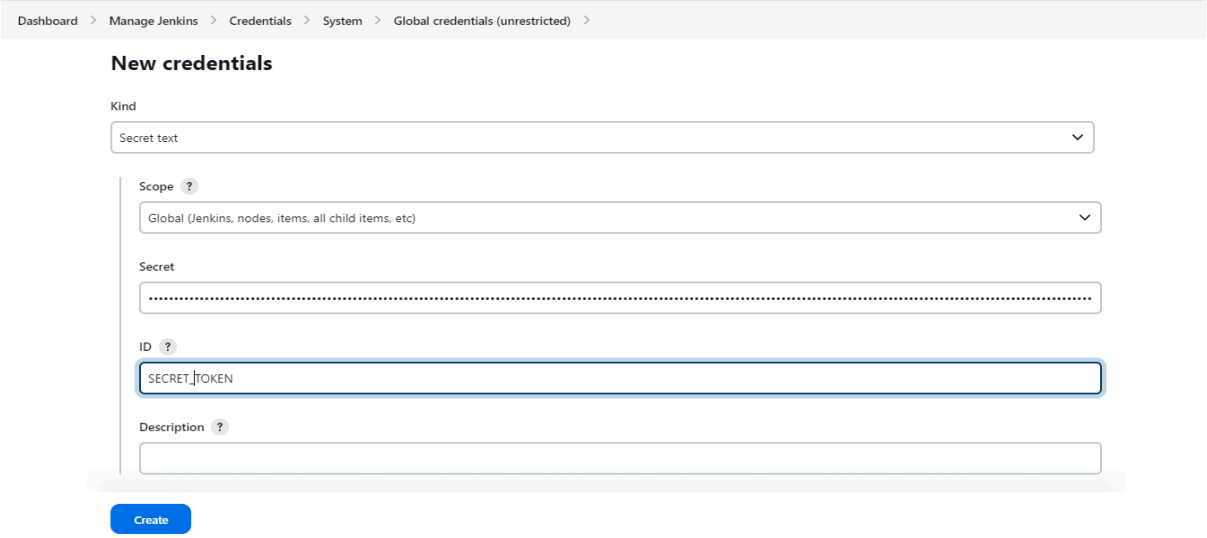
Create a secret for service account jenkins

```
cat <<EOF | kubectl create -f -
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
 name: jenkins
 annotations:
   kubernetes.io/service-account.name: jenkins
EOF
```

```
kubectl get secrets jenkins -o jsonpath='{.data.token}' | base64 -d
```

Add Kubernetes service account secret in Jenkins Credentials
Manage Jenkins> **Credentials** > then click on **global** and click on add credentials

Under **Kind**, scroll on the drop-down list and then choose **Secret text** Under secret, copy the Kubernetes token we generated earlier and paste it there. Then enter ID and description



Integrate Remote Kubernetes Cluster with Jenkins

Go to **Manage Jenkins > Clouds >** Select **New cloud** type it cloud name:
kubernetes-cloud press the create button
Name: kubernetes-cred

Kubernetes URL: Open your config file and enter server URL

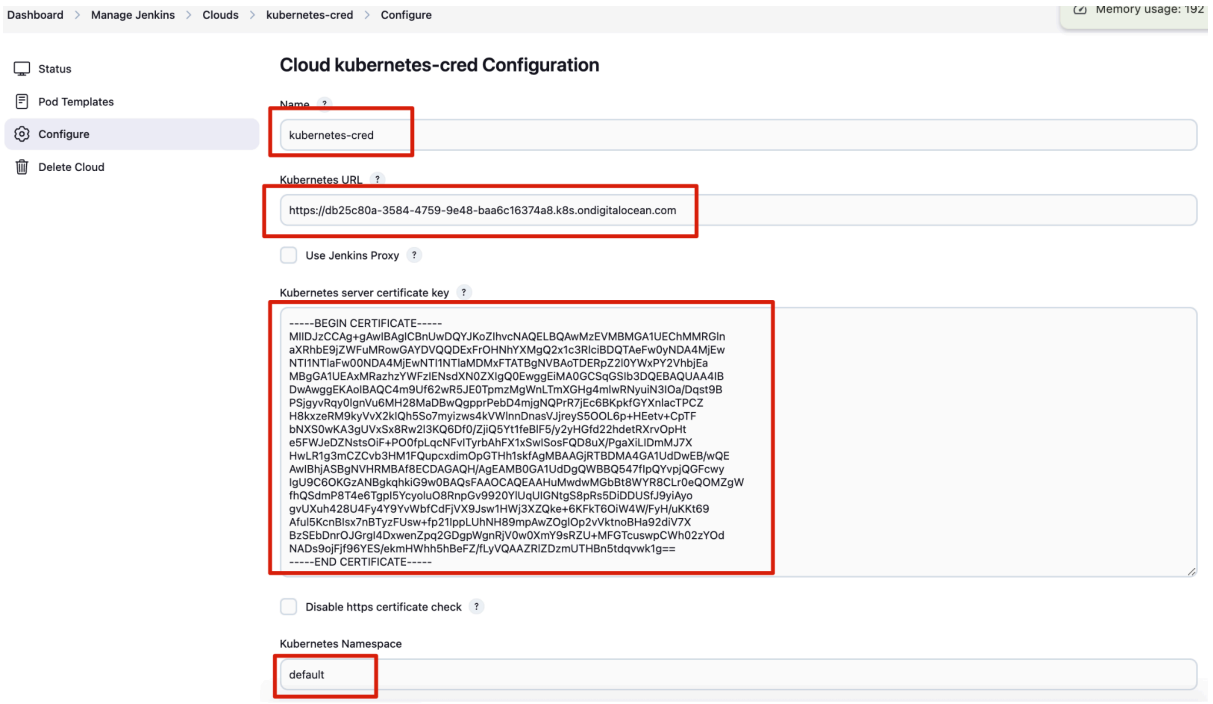
Kubernetes URL: `https://db25c80a-3584-4759-9e48-baa6c16374a8.k8s.ondigitalocean.com`

Kubernetes server certificate key: You need to generate this key using this command
Convert kubernetes server certificate key to base64 format using below command

```
cat ~/.kube/config
```

echo -n <contents_of_the_certificate-authority-data_entry_of_my_kubeconfig_file> | base64 --decode

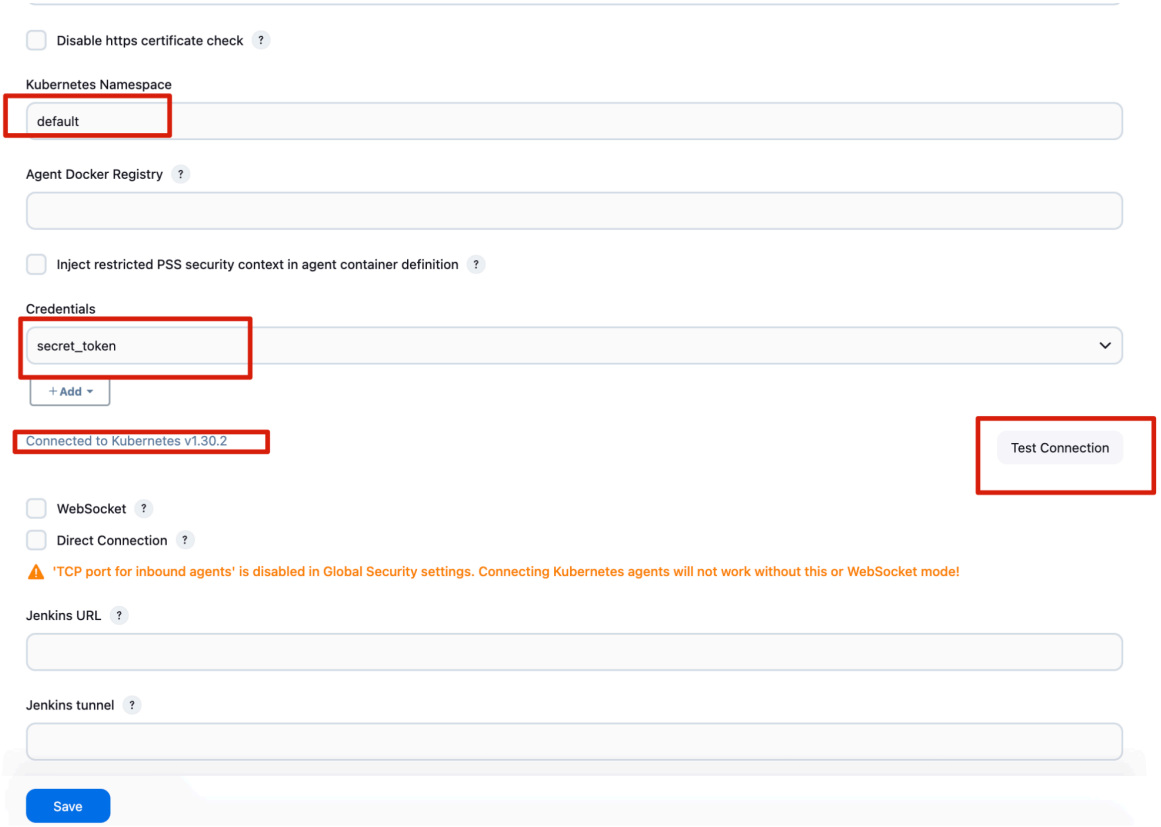
kubernetes server certification key: Copy to echo result
Kubernetes Namespace:default



Credentials: secret_token

Press the **Test Connection** button for check kubernetes cluster connection

If your connection is successfully you should press the **Save** button



Step 4: Flask-monitoring Application Structure

You can fork or clone this repository https://github.com/hakanbayraktar/flask-monitoring.git
git clone https://github.com/hakanbayraktar/flask-monitoring.git

```
flask-monitoring/
├── templates
    ├──index.html
├── app.py
├── Dockerfile
├── requirements.txt
├── Jenkinsfile
├── deploymen.yaml
├── jenkins.sh
├── service.yaml
└── README.md
```

Dockerfile:

```
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
```

```
COPY . .
ENV FLASK_RUN_HOST=0.0.0.0
EXPOSE 5000
CMD [ "flask", "run"]
```

app.py

```
import psutil
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")
def index():
   cpu_metric = psutil.cpu_percent()
   mem_metric = psutil.virtual_memory().percent
   Message = None
   if cpu_metric > 70 or mem_metric > 70:
       Message = "High CPU or Memory Detected, scale up!!!"
   return render_template("index.html",
cpu_metric=cpu_metric, mem_metric=mem_metric,
message=Message)

if __name__=='__main__':
   app.run(debug=True, host = '0.0.0.0')
```

requirements.txt

```
Flask==2.2.3
MarkupSafe==2.1.2
Werkzeug==2.2.3
itsdangerous==2.1.2
psutil==6.0.0
plotly==5.5.0
tenacity==8.0.1
boto3==1.34.151
kubernetes==10.0.1
```

index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>System Monitoring</title>
    <script
src="https://cdn.plot.ly/plotly-latest.min.js"></script>
    <style>
        .plotly-graph-div {
            margin: auto;
            width: 50%;
            background-color: rgba(151, 128, 128, 0.688);
            padding: 20px;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>System Monitoring</h1>
        <div id="cpu-gauge"></div>
        <div id="mem-gauge"></div>
        {% if message %}
        <div class="alert alert-danger">{{ message }}</div>
        {% endif %}
    </div>
    <script>
        var cpuGauge = {
            type: "indicator",
            mode: "gauge+number",
            value: {{ cpu_metric }},
            gauge: {
                axis: { range: [null, 100] },
                bar: { color: "#1f77b4" },
                bgcolor: "white",
                borderwidth: 2,
                bordercolor: "#ccc",
                steps: [
                    { range: [0, 50], color: "#d9f0a3" },
                    { range: [50, 85], color: "#ffeb84" },
```

```
                { range: [85, 100], color: "#ff5f5f" }
            ],
            threshold: {
                line: { color: "red", width: 4 },
                thickness: 0.75,
                value: {{ cpu_metric }}
            }
        }
    };

    var memGauge = {
        type: "indicator",
        mode: "gauge+number",
        value: {{ mem_metric }},
        gauge: {
            axis: { range: [null, 100] },
            bar: { color: "#1f77b4" },
            bgcolor: "white",
            borderwidth: 2,
            bordercolor: "#ccc",
            steps: [
                { range: [0, 50], color: "#d9f0a3" },
                { range: [50, 85], color: "#ffeb84" },
                { range: [85, 100], color: "#ff5f5f" }
            ],
            threshold: {
                line: { color: "red", width: 4 },
                thickness: 0.75,
                value: {{ mem_metric }}
            }
        }
    };

    var cpuGaugeLayout = { title: "CPU Utilization" };
    var memGaugeLayout = { title: "Memory Utilization" };
```

```
        Plotly.newPlot('cpu-gauge', [cpuGauge],
cpuGaugeLayout);
        Plotly.newPlot('mem-gauge', [memGauge],
memGaugeLayout);
    </script>
</body>
</html>
```

Jenkinsfile

```
pipeline {
    agent any

    stages {
        stage('Git Cloning') {
            steps {
                echo 'Cloning git repo'
                git url:
'https://github.com/hakanbayraktar/flask-monitoring.git', branch: 'main'
            }
        }
        stage('Build Docker Image') {
            steps {
                echo 'Building the image'
                sh 'docker build -t flask-monitoring .'
            }
        }
        stage('Push to Docker Hub') {
            steps {
                echo 'Pushing to Docker Hub'
                withCredentials([usernamePassword(credentialsId:
'dockerhub-cred', usernameVariable: 'USER', passwordVariable: 'PASS')]) {
                    sh '''
                    echo "${PASS}" | docker login --username "${USER}"
--password-stdin
                    docker tag flask-monitoring ${USER}/flask-monitoring:latest
                    docker push ${USER}/flask-monitoring:latest
                    '''
                }
            }
        }
        stage('Integrate Remote k8s with Jenkins') {
            steps {
```

```
                withCredentials([string(credentialsId: 'secret_token', variable:
'KUBE_TOKEN')]) {
                    sh '''
                    curl -LO
"https://storage.googleapis.com/kubernetes-release/release/v1.20.5/bin/linux/am
d64/kubectl"
                    chmod u+x ./kubectl
                    export KUBECONFIG=$(mktemp)
                    ./kubectl config set-cluster do-fra1-ibb-tech
--server=https://db25c80a-3584-4759-9e48-baa6c16374a8.k8s.ondigitalocean.com
--insecure-skip-tls-verify=true
                    ./kubectl config set-credentials jenkins
--token=${KUBE_TOKEN}
                    ./kubectl config set-context default
--cluster=do-fra1-ibb-tech --user=jenkins --namespace=default
                    ./kubectl config use-context default
                    ./kubectl get nodes
                    ./kubectl apply -f service.yaml
                    ./kubectl apply -f deployment.yaml
                    '''
                }
            }
        }
    }
}
```

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: flask-monitoring
 labels:
   app: flask-monitoring
spec:
 replicas: 1
 selector:
   matchLabels:
     app: flask-monitoring
 template:
   metadata:
     labels:
       app: flask-monitoring
   spec:
     containers:
```

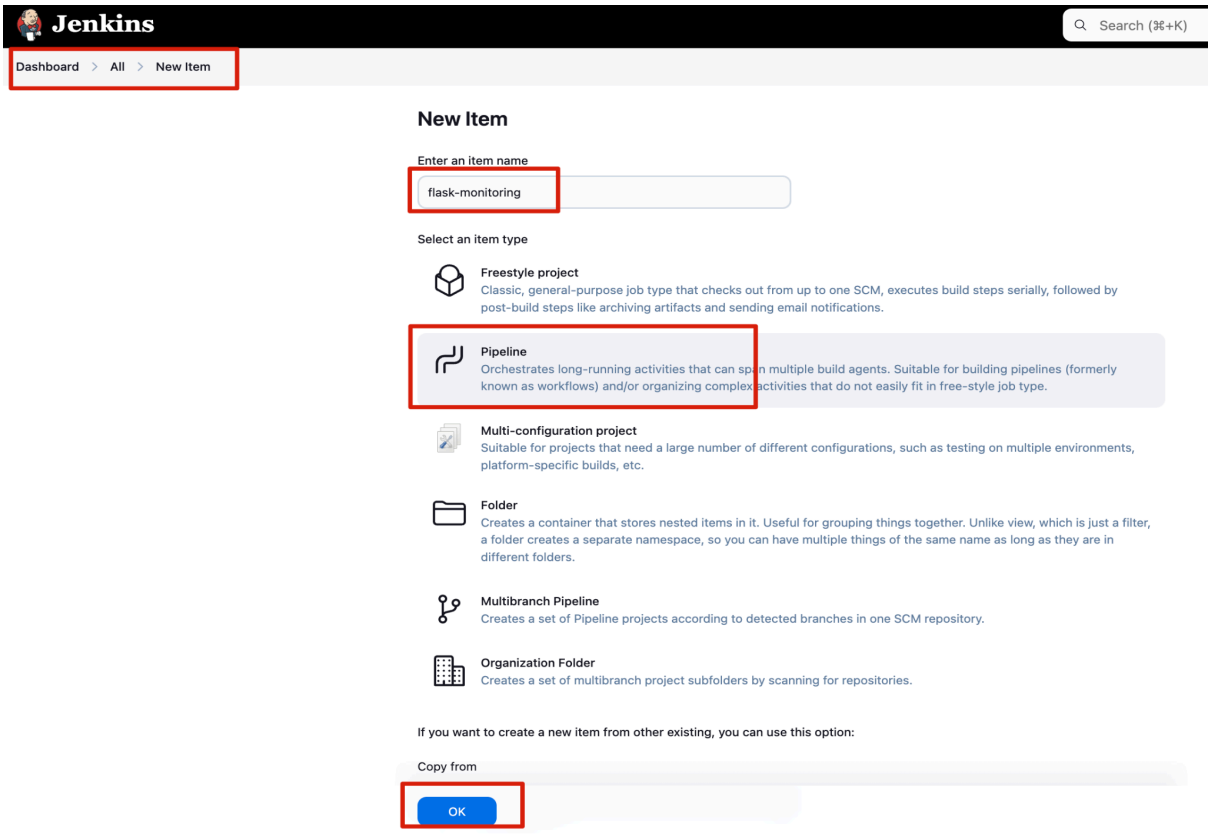
```
    - name: flask-monitoring
      image: hbayraktar/flask-monitoring:latest
      ports:
      - containerPort: 5000
```

service.yaml

```
apiVersion: v1
kind: Service
metadata:
 name: my-service
spec:
 type: LoadBalancer
 selector:
   app: flask-monitoring
 ports:
   - protocol: TCP
     port: 80
     targetPort: 5000
```

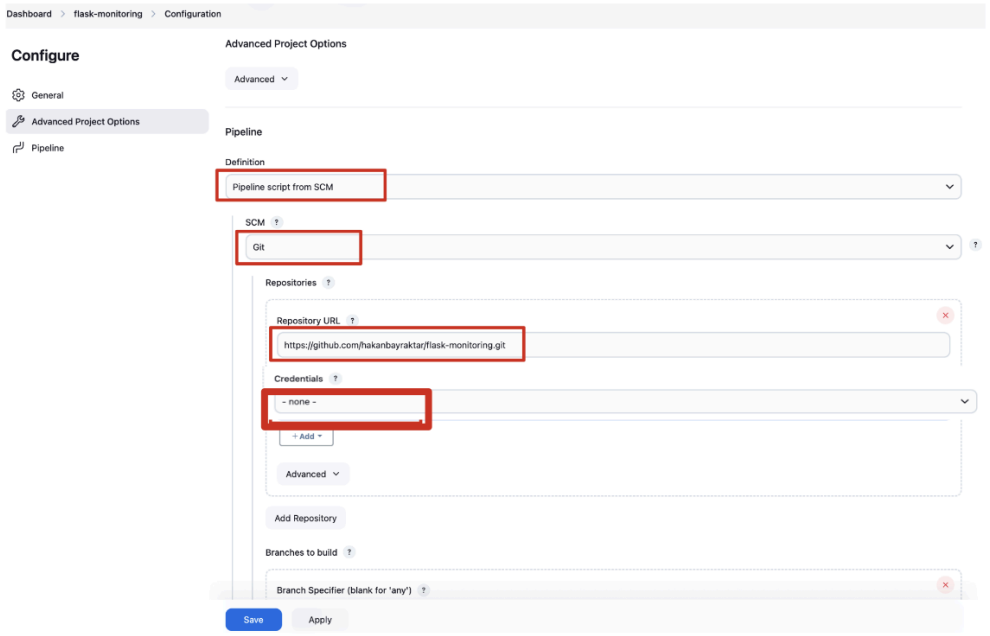
Step 5: Jenkins Pipeline Configuration

Select **New Item** button for create a new job. **Enter an Item Name:** flask-monitoring
Select **Pipeline** and press the **OK** button

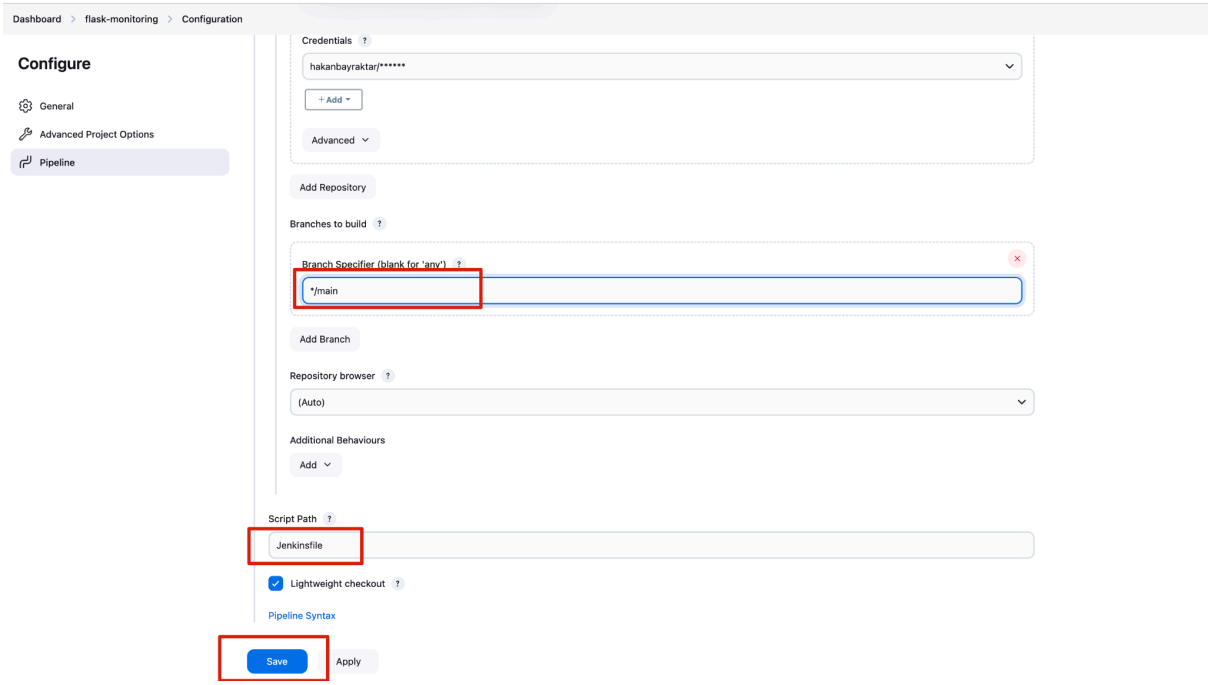


Select **flask-monitoring** job and on the left menu find **Advanced Project Options:**

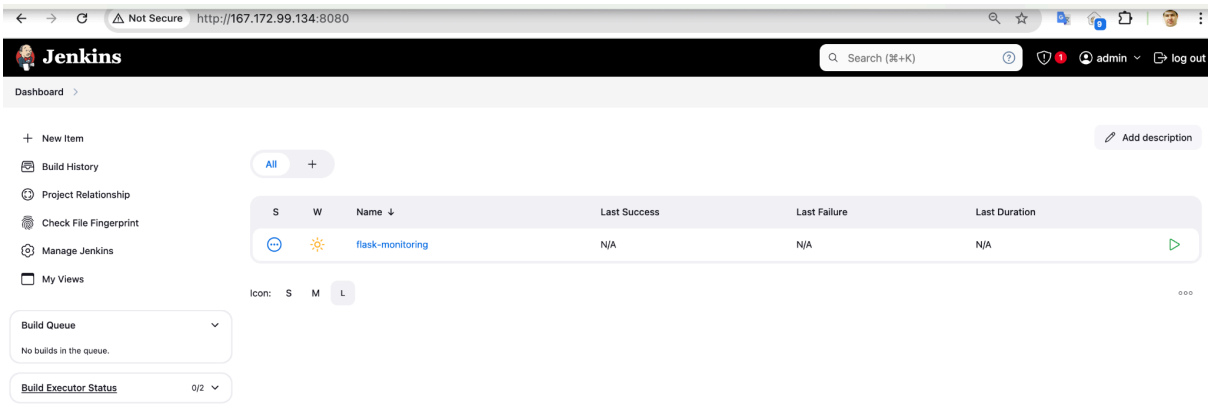
Definitions: Select pipeline script from SCM
SCM: Git
Repository URL:https://github.com/hakanbayraktar/flask-monitoring.git
Credentials: None

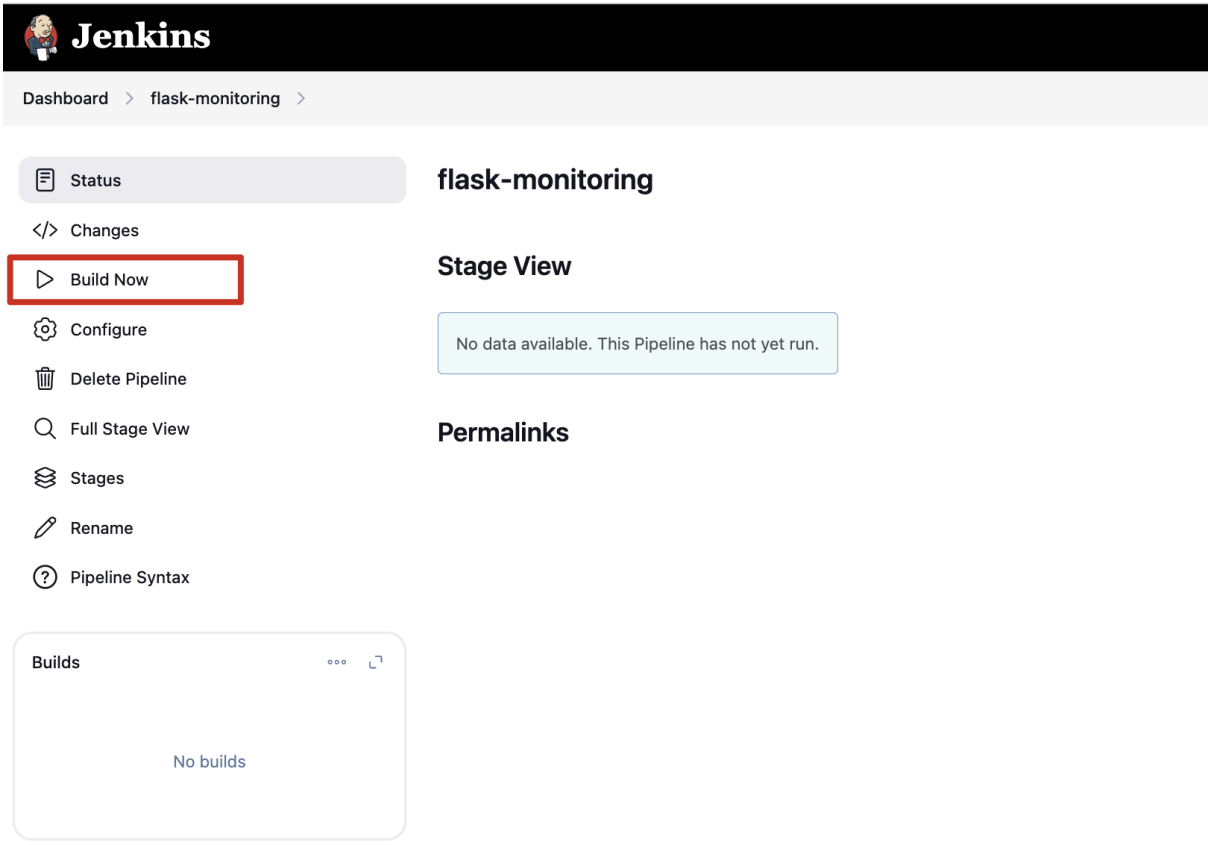


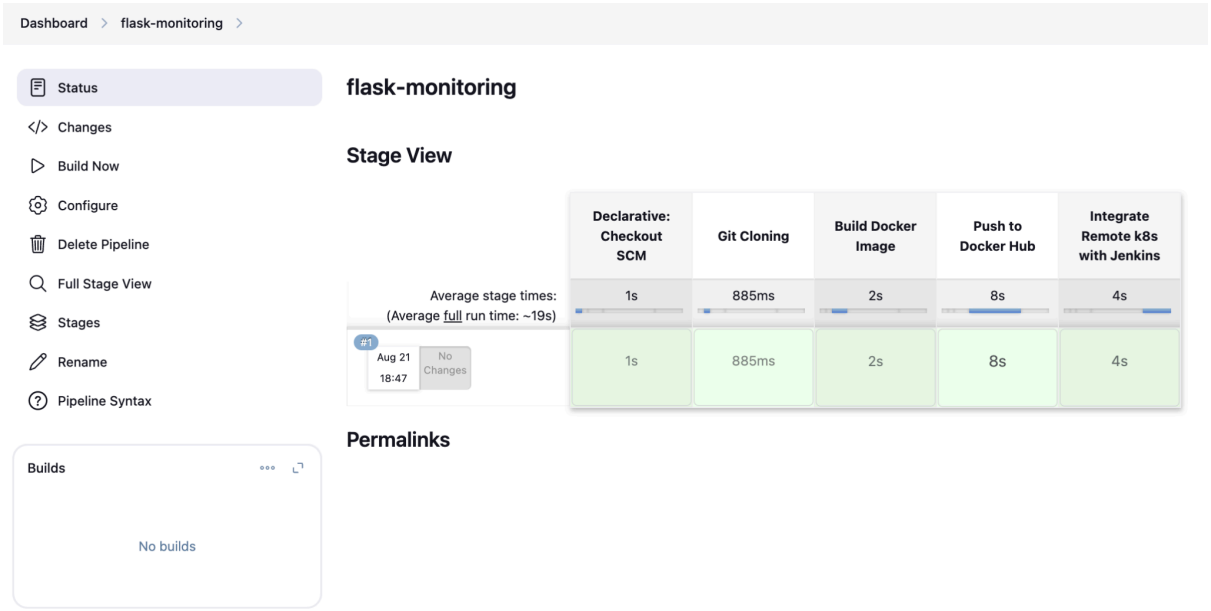
Branch Specifier (blank for 'any'): */main
Script Path: Jenkinsfile
click on **Save** button



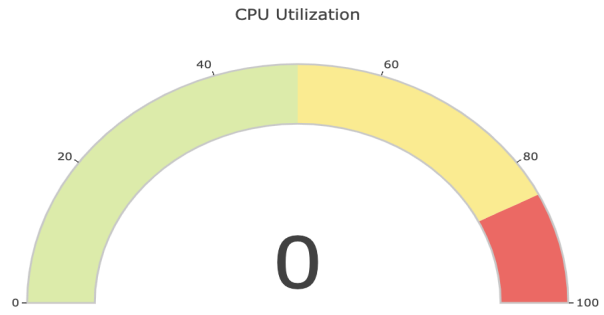
Step 6: Deploying on Kubernetes

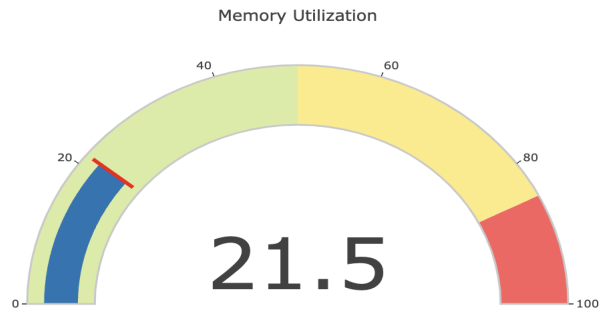






System Monitoring





Conclusion

By following this guide, you've set up a CI/CD pipeline with Jenkins, Docker, and Kubernetes, fully automating the process of building, testing, and deploying a Flask application. This setup can be easily adapted for other types of applications or services, making it a powerful tool for DevOps practices.