Since you've provided a comprehensive guide with steps for several distinct tasks (Jenkins, Docker, Kubernetes, and ML Environment setup), here is the step-by-step execution guide for a new Linux system, including all necessary terminal commands and file contents.

0. Initial System Setup (Prerequisites)

You need to ensure **Git**, **Python**, **Docker**, and **Kubernetes Tools** are installed. **Jenkins** will be installed/run via Docker for simplicity.

Install Basic Tools (Git, Python)

Bash

```bash
# Update package list
sudo apt update

# Install git and python3-venv (for virtual environments)
sudo apt install -y git python3-venv

# Verify installations
git --version
python3 --version
```

Install Docker

Bash

```bash
# Add Docker's official GPG key:
sudo apt update
sudo apt install -y ca-certificates curl
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

# Add the repository to Apt sources:
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 update

# Install Docker packages
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin
docker-compose-plugin

# Add current user to the docker group (to run docker without sudo)
sudo usermod -aG docker $USER

# Log out and log back in, or run 'newgrp docker' for the change to take effect immediately in the
current shell.
# Run 'newgrp docker' now:
newgrp docker

# Verify Docker installation
docker run hello-world
```

Install Kubernetes Tools (kubectl, minikube)

We will use **minikube** to run a local Kubernetes cluster.

Bash

```
# Install kubectl
curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

# Install minikube
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube && rm minikube-linux-amd64

# Start minikube (using docker driver since it's installed)
minikube start --driver=docker

# Verify kubectl
kubectl get nodes
```

Set up GitHub Repository

This example assumes you have a GitHub repository named student-portal with an index.html file. Replace YOUR_USERNAME with your actual GitHub username.

Bash

```bash
# Create and clone the repo (assuming you set it up on GitHub first)
mkdir student-portal
cd student-portal

# Initialize a simple index.html file
echo '<!doctype html>
<html><body><h1>Event Registration</h1></body></html>' > index.html

# Initialize git and link to GitHub (replace YOUR_USERNAME)
git init
git add index.html
git commit -m "Initial commit for student portal site"
git remote add origin https://github.com/YOUR_USERNAME/student-portal.git

# Push to GitHub (requires credentials/token)
git push -u origin main
cd .. # Go back to the main directory
```

1. Jenkins Installation and Setup

We will run Jenkins using Docker.

A. Install Jenkins and Plugins

Bash

```bash
# Run Jenkins in a Docker container
docker run -d -p 8080:8080 -p 50000:50000 --name jenkins -v
jenkins_home:/var/jenkins_home jenkins/jenkins:lts

# Wait a minute, then retrieve the initial admin password
```

```
echo "Wait for Jenkins to start..."
sleep 60
docker exec jenkins cat /var/jenkins_home/secrets/initialAdminPassword
# Copy the outputted password (e.g., d30c4e0b5f6a7c8d9e0f1a2b3c4d5e6f)
```

Manual Steps in Jenkins UI (http://localhost:8080):

1. Open **http://localhost:8080** in your browser.
2. Enter the retrieved **Initial Admin Password**.
3. Choose **Install suggested plugins**.
4. Create the **First Admin User** (e.g., admin/password).
5. Set the **Jenkins URL** (default is fine).
6. *Ensure the following plugins are installed (they are typically in the suggested set):* **Git plugin**, **GitHub plugin (GitHub Hook)**.

B. Create Jenkins Job

Manual Steps in Jenkins UI:

1. **Jenkins UI → New Item**.
2. Enter **Item name:** student-portal-site.
3. Select **Freestyle project** and click **OK**.

- **General:** (Optional) Check **Discard old builds** for cleanup.
- **Source Code Management:**
  - Choose **Git**.
  - **Repository URL:** https://github.com/YOUR_USERNAME/student-portal.git (replace YOUR_USERNAME).
  - *If your repo is private, click **Add** under Credentials and provide your GitHub username/Personal Access Token.*
- **Build Triggers:**
  - Check **GitHub hook trigger for GITScm polling**.
- **Build:**
  - Click **Add build step** and choose **Execute shell**.
  - Paste the following script:

    Bash
    ```
    echo "Building student-portal"
    ls -la
    echo "Contents of index.html:"
    sed -n '1,80p' index.html
    ```

- Click **Save**.

C. Configure GitHub Webhook

Manual Steps in GitHub UI:

1. Go to your GitHub repository: https://github.com/YOUR_USERNAME/student-portal.
2. Click **Settings** tab.
3. Click **Webhooks** in the sidebar.
4. Click **Add webhook**.
    - **Payload URL:** http://<YOUR_JENKINS_IP_OR_HOSTNAME>:8080/github-webhook/
        - *If Jenkins is running locally, you can use the host machine's local IP or http://172.17.0.2:8080/github-webhook/ (get container IP with docker inspect -f '{{.NetworkSettings.IPAddress}}' jenkins if needed).* For simple testing, http://<Your_Linux_IP>:8080/github-webhook/ is best.
    - **Content type:** application/json
    - **Secret:** (Leave blank)
    - **Just the push event** (Select this option).
    - Ensure **Active** is checked.
5. Click **Add webhook**.

D. Make a Small Update and Push

Run these commands inside your local student-portal directory:

Bash

```bash
cd student-portal

# Create a new branch
git checkout -b tiny-update

# Change index.html (updates the heading)
sed -i 's/Event Registration/Event Registration - Updated/' index.html

# Commit and push the branch
git add index.html
git commit -m "Tiny update: change heading for CI test"
git push -u origin tiny-update

# Merge to main
git checkout main
git merge tiny-update
git push origin main

cd ..
```

E. Expected Jenkins Behavior

Verification Steps:

1. **GitHub:** Go back to your GitHub repo **Settings → Webhooks → student-portal-site webhook**. Check **Recent Deliveries**; the latest one should show a **green checkmark** and a **status 200**.
2. **Jenkins:** Go back to the student-portal-site job page (http://localhost:8080/job/student-portal-site/).
   - A new build should have been automatically triggered and should be running or finished.
   - Click the **Build Number** (e.g., #2) -> **Console Output**.
   - You should see the output confirming the new heading:

     ```
     ...
     Building student-portal
     total 8
     -rw-r--r-- 1 jenkins jenkins 234 Dec 11 10:00 index.html
     Contents of index.html:
     <!doctype html>
     <html><body>
     <h1>Event Registration - Updated</h1>
     </body></html>
     Finished: SUCCESS
     ```

---

2. Docker Web Application and Commands

4. Create a simple web application (Flask) + Dockerfile, build image, run container

Create a new directory for the application:

Bash

```bash
mkdir flask-app
cd flask-app
```

Create app.py:

Bash

```bash
cat << 'EOF' > app.py
from flask import Flask, render_template_string
app = Flask(__name__)
HTML = """
<!doctype html>
<html><body>
<h1>Student Portal</h1>
<p>Registration form:</p>
<form>
Name: <input name="name"><br>
Email: <input name="email"><br>
Phone: <input name="phone"><br>
Department: <input name="department"><br>
</form>
</body></html>
"""
@app.route("/")
def home():
return render_template_string(HTML)
if __name__ == "__main__":
app.run(host="0.0.0.0", port=5000)
EOF
```

Create requirements.txt:

Bash

```bash
echo "Flask==2.3.2" > requirements.txt
```

Create Dockerfile:

Bash

```
cat << EOF > Dockerfile
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY app.py .
EXPOSE 5000
CMD ["python", "app.py"]
EOF
```

Build and Run Commands:

Bash

```
# Build image
docker build -t student-portal:v1 .

# Run container mapping host port 5000 -> container 5000
docker run -d --name student-portal -p 5000:5000 student-portal:v1

# Verify in browser or curl (You should see the HTML output)
curl -s http://localhost:5000 | head -n 10

cd ..
```

5. Docker commands: list images/containers, stop container, remove container/image

Bash

```
# List running containers
docker ps

# List all containers (running and stopped)
docker ps -a

# List images
```

```
docker images

# Stop the running container
docker stop student-portal

# Remove the container (must be stopped first)
docker rm student-portal

# Remove the image
docker rmi student-portal:v1
```

---

3. Kubernetes Deployment and Scaling

6. Deploy Docker image on Kubernetes using a Deployment YAML

First, ensure your image is available to minikube. Since we built it with Docker, we need to load it into the minikube environment.

Bash

```bash
cd flask-app
minikube image load student-portal:v1
```

Create deployment.yaml:

Bash

```bash
cat << EOF > deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: student-portal-deploy
spec:
  replicas: 1
  selector:
    matchLabels:
      app: student-portal
  template:
    metadata:
      labels:
        app: student-portal
    spec:
      containers:
      - name: student-portal
        image: student-portal:v1
        imagePullPolicy: Never # Use the locally loaded image
        ports:
        - containerPort: 5000
EOF
```

Apply deployment and verify:

Bash

```
# Apply deployment
kubectl apply -f deployment.yaml

# Verify deployments
kubectl get deployments

# Verify pods
kubectl get pods -l app=student-portal

# Get logs from the running pod (replace <pod-name>)
POD_NAME=$(kubectl get pods -l app=student-portal -o jsonpath='{.items[0].metadata.name}')
kubectl logs $POD_NAME
```

7. Expose application with NodePort and scale to 3 replicas

Create service-nodeport.yaml:

Bash

```
cat << EOF > service-nodeport.yaml
apiVersion: v1
```

```
kind: Service
metadata:
  name: student-portal-nodeport
spec:
  type: NodePort
  selector:
    app: student-portal
  ports:
  - protocol: TCP
    port: 5000
    targetPort: 5000
    nodePort: 30080 # optional, setting it explicitly here
EOF
```

Apply service and verify:

Bash

```
# Apply service
kubectl apply -f service-nodeport.yaml

# Get service details
kubectl get svc student-portal-nodeport

# Get minikube IP to access the service
MINIKUBE_IP=$(minikube ip)
echo "Minikube IP: $MINIKUBE_IP"

# Access service (should return HTML)
curl http://${MINIKUBE_IP}:30080 | head -n 10
```

Scale Deployment:

Bash

```
# Scale deployment to 3 replicas
kubectl scale deployment student-portal-deploy --replicas=3
```

```
# Verify 3 pods are running
kubectl get pods -l app=student-portal
```

---

4. ML Project Environment Setup

8. Set up simple ML project environment

Create a new directory for the ML project:

Bash

```
cd .. # Exit flask-app directory
mkdir ml-project
cd ml-project
```

Create requirements-ml.txt:

Bash

```
cat << EOF > requirements-ml.txt
numpy==1.26.2
pandas==2.2.2
scikit-learn==1.3.2
jupyterlab==4.1.0
EOF
```

Set up virtual environment & install packages:

Bash

```
# Create virtual environment
python3 -m venv venv
```

```
# Activate virtual environment
source venv/bin/activate

# Install packages
pip install -r requirements-ml.txt

# Verify installed packages
pip freeze | grep -E "numpy|pandas|scikit-learn|jupyter"
```

Create ml_setup.ipynb (Jupyter Notebook):

You need to install the Jupyter environment and open the UI to create this file manually, or use a tool like nbformat to create it programmatically.

- **To run Jupyter and create the file manually:**
  Bash
  ```
  jupyter notebook # or jupyter lab
  ```
  (Follow the instructions to open the notebook in your browser, create a new Python 3 notebook, and add the two cells.)
- **Content of ml_setup.ipynb (Cells):**
  - **Cell 1 (Markdown):**
    Markdown
    ```
    # ML Environment Setup
    This notebook verifies that required Python packages are installed and runs a tiny sample.
    ```
  - **Cell 2 (Code):**
    Python
    ```
    import numpy as np
    import pandas as pd
    from sklearn.linear_model import LinearRegression

    print("numpy:", np.__version__)
    print("pandas:", pd.__version__)

    # tiny sample
    X = np.array([[1],[2],[3],[4]])
    y = np.array([2,4,6,8])
    model = LinearRegression().fit(X, y)
    print("coef:", model.coef_, "intercept:", model.intercept_)
    ```

Commit to Git

Assuming you have initialized a Git repo for this ML project:

Bash

```bash
# Initialize Git for this project
git init
git remote add origin https://github.com/YOUR_USERNAME/ml-project.git # Replace with your ML repo URL

# Add files
git add requirements-ml.txt ml_setup.ipynb

# Commit and push
git commit -m "Add ML environment requirements and verification notebook"
git push origin main
```

Sources

1. https://forums.docker.com/t/an-error-while-setting-up-container/140117