



Project Title: CI/CD - DevOps

**Duration:** 01/08/2025 – 27/08/2025

**Organization:** Sunbeam Institute Of Information Technology

**Submitted by:** Neha Sanjay Kumbharde(230344223024)

Under the guidance of

**Mr. Gajanan Taur Sir**

**In partial fulfillment of the award of Internship Program**



**Sunbeam Institute of Information Technology,  
Pune (Maharashtra)**

## **DECLARATION**

We declare that this written submission represents our ideas in our own words and where others ideas or words have been included; we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Place: Pune

Date:

Neha Sanja Kumbharde(93886)

## **CERTIFICATE**

This is to certify that the project report entitled “**CI/CD - DevOps**”, submitted by **Neha Kumbharde** is the bonafide work completed under our supervision and guidance in partial fulfillment for the Internship Program of Sunbeam Institute of Information Technology, Pune

Place: Pune

Date:

**Mr. Nitin Kudale**

CEO

Sunbeam Institute of Information Technology

Pune (M.S.) – 411057

## APPROVAL CERTIFICATE

This Project report entitled “**Security DMZ**(Demilitarized Zone)” by **Neha Sanjay Kumbharde** is approved for Internship Program of Sunbeam Institute of Information Technology, Pune (M.S.).

Place: Pune

Date:

Examiner:

---

**(Signature)**

---

**(Name)**

# CONTENT

TITLE	PAGE NO
Declaration	
Certificate	
Approval Certificate	
Abstract	
1.INTRODUCION	1
2. Technologies Used and Setup Overview	4
3.Final Architecture Overview	6
4.Conclusion	8

## ABSTRACT

This project demonstrates the complete lifecycle of containerized application deployment and orchestration using modern DevOps tools and practices. The implementation begins with containerization using **Docker**, where a simple **To-Do application** is built from a Dockerfile and deployed with separate containers for the application and its database.

Next, a **Kubernetes cluster** is set up across two virtual machines to orchestrate and manage the containers efficiently. Using **Kubernetes Deployments** and **Services**, the application is deployed in a scalable and resilient manner. To ensure data persistence, **Persistent Volumes (PV)** and **Persistent Volume Claims (PVC)** are configured.

The project further integrates **Continuous Integration (CI)** through **GitHub Actions** and **Continuous Deployment (CD)** using **ArgoCD**, streamlining automated application updates. Additionally, **Helm charts** are used to simplify Kubernetes resource management and versioning.

For observability and monitoring, **Prometheus** is configured to collect metrics from both the Kubernetes cluster and the Flask application, while **Grafana** dashboards provide real-time visualization of these metrics. Alerting mechanisms are established for proactive issue detection and response.

Overall, this project provides a hands-on understanding of the DevOps pipeline — from containerization and orchestration to monitoring and automation — ensuring a robust, scalable, and maintainable cloud-native application deployment.

# 1.INTRODUCTION

This project focuses on building, deploying, and monitoring a complete cloud-native application stack using modern DevOps tools and practices. The goal is to understand how containerization, orchestration, continuous integration/continuous deployment (CI/CD), and monitoring work together to create a scalable and automated deployment environment.

The project begins with setting up **Docker** to containerize a simple **Flask To-Do application**, followed by deploying it in isolated containers for both the application and the database. Next, the project transitions into **Kubernetes (K8s)** — the industry-standard orchestration tool — to manage and scale these containers efficiently across multiple virtual machines.

After container orchestration, the focus shifts to **data persistence** using **Persistent Volumes (PV)** and **Persistent Volume Claims (PVC)**, ensuring that the application retains its data even when containers are restarted or redeployed. To automate the software delivery pipeline, **GitHub Actions** is used for **Continuous Integration (CI)** and **ArgoCD** for **Continuous Deployment (CD)**. Additionally, **Helm charts** are introduced to simplify Kubernetes resource management and enable version-controlled deployments.

Finally, to ensure reliability and visibility into the system's performance, **Prometheus** and **Grafana** are integrated for **monitoring and observability**. Prometheus collects real-time metrics from both the Kubernetes cluster and the Flask application, while Grafana visualizes these metrics through interactive dashboards. Alerts and notifications are configured to proactively respond to performance issues or system failures, creating a complete DevOps monitoring workflow.

Through this project, learners gain hands-on experience in:

- **Containerization** with Docker
- **Orchestration** using Kubernetes
- **Data persistence** with PVs and PVCs
- **CI/CD automation** using GitHub Actions and ArgoCD
- **Monitoring and observability** via Prometheus and Grafana

Overall, this end-to-end setup demonstrates how modern DevOps practices streamline application deployment, scaling, and monitoring in a production-like environment.

## 2.Technologies Used and Setup Overview:

### Day 01 (01/09/2025) :

#### Objectives:

- Install Docker on Ubuntu/Debian machine.
- Containerize a Flask-based Todo App.
- Run containers for both application and database.

#### Tools & Technologies:

- Docker
- Dockerfile
- Flask (Python)
- PostgreSQL / SQLite (as DB)

#### Tasks and Implementation:

##### 1.Docker Installation:

- a) Installed Docker Engine using repository:

```
sudo apt update && sudo apt upgrade -y  
sudo apt install apt-transport-https ca-certificates curl software-properties-  
common -y  
sudo apt update  
sudo apt install docker-ce -y  
sudo systemctl start docker  
sudo systemctl enable docker
```

##### 2.Dockerize Todo App:

- a) Created a Dockerfile for the Flask application:

```
FROM python:3.9-slim  
WORKDIR /app  
COPY requirements.txt .  
RUN pip install -r requirements.txt
```



*COPY . .*

*CMD ["python", "app.py"]*

### 3. Docker Image Creation:

*docker build -t todo-app:latest .*

### 4. Run Containers:

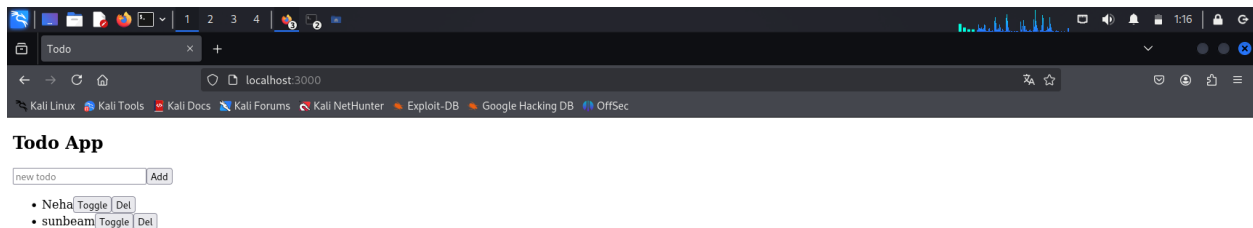
*Docker network create todonet*

*Docker run -d --name db --network todonet -e POSTGRES\_PASSWORD=pass postgres*

*Docker run -d --name todoapp --network todonet -p 5000:5000 todoapp*

## Outcome:

A basic Flask Todo application is containerized and running in isolated Docker containers, connected via a custom network.



## **Day 02 (09/09/2025):**

### **Objectives:**

- Create a Kubernetes Cluster using two VMs (1 Master, 1 Worker)

### **Tools & References:**

- VirtualBox / Cloud VMs
- kubeadm, kubelet, kubectl

### **Tasks and Implementation:**

#### **1) VM Setup:**

- Two Ubuntu VMs created.
- Required packages installed: containerd, kubeadm, kubectl, kubelet.

#### **2) Initialize Master Node:**

```
kubeadm init --pod-network-cidr=192.168.0.0/16
```

#### **3) Set Up kubeconfig:**

```
mkdir -p $HOME/.kube  
cp /etc/kubernetes/admin.conf $HOME/.kube/config
```

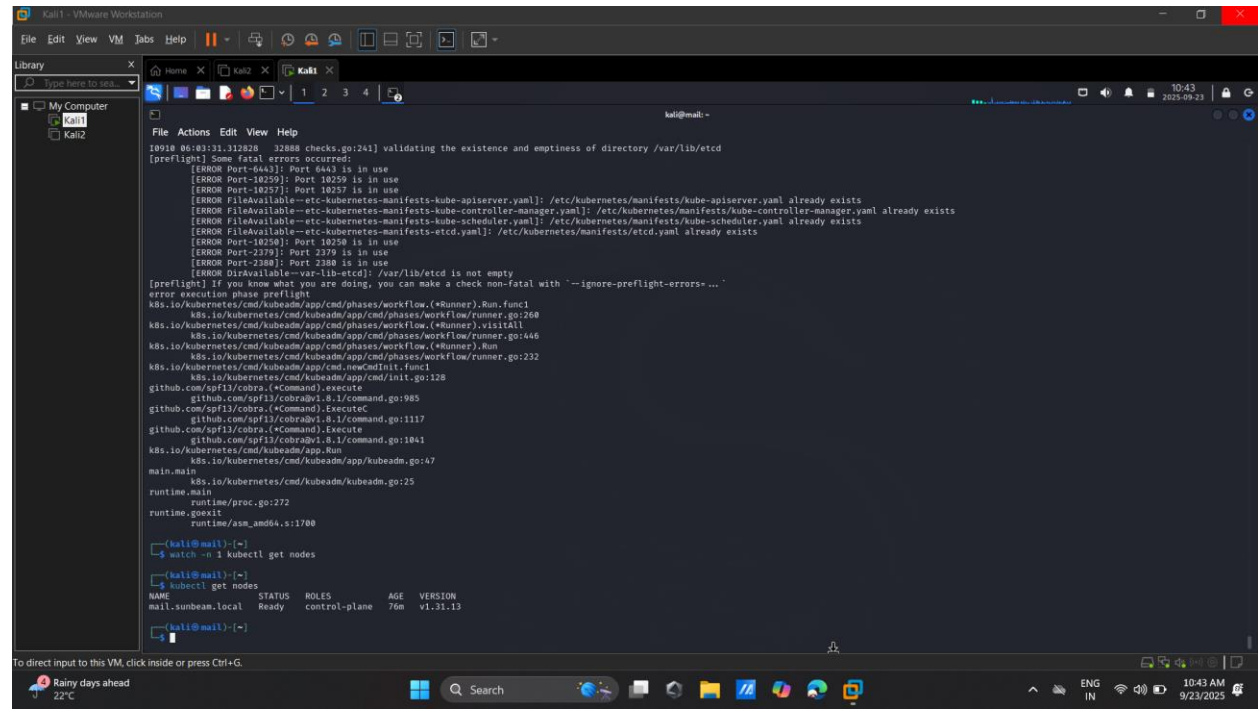
#### **4) Join Worker Node:**

- a ) Used the kubeadm join token command from master to connect worker node.

#### **5) Deploy Network Plugin (Calico/Flannel):**

```
kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml
```

**Outcome:** A working Kubernetes cluster with master and worker node is successfully deployed and ready for application orchestration.



```
1091s 00:01:31.112828 32888 checks.go:243] validating the existence and emptiness of directory /var/lib/etcd
[preflight] Some fatal errors occurred:
[ERROR Port-6443]: Port 6443 is in use
[ERROR Port-10259]: Port 10259 is in use
[ERROR Port-10257]: Port 10257 is in use
[ERROR FileAvailable--etc-kubernetes-manifests-kube-apiserver.yaml]: /etc/kubernetes/manifests/kube-apiserver.yaml already exists
[ERROR FileAvailable--etc-kubernetes-manifests-kube-controller-manager.yaml]: /etc/kubernetes/manifests/kube-controller-manager.yaml already exists
[ERROR FileAvailable--etc-kubernetes-manifests-kube-scheduler.yaml]: /etc/kubernetes/manifests/kube-scheduler.yaml already exists
[ERROR FileAvailable--etc-kubernetes-manifests-etcd.yaml]: /etc/kubernetes/manifests/etcd.yaml already exists
[ERROR Port-10258]: Port 10258 is in use
[ERROR Port-2379]: Port 2379 is in use
[ERROR Port-2380]: Port 2380 is in use
[ERROR DirAvailable--var-lib-etcd]: /var/lib/etcd is not empty
[preflight] If you know what you are doing, you can make a check non-fatal with '--ignore-preflight-errors=...'
error execution phase preflight
k8s.io/kubernetes/cmd/kubeadm/app/cmd/phases/workflow.(*Runner).Run.func1
k8s.io/kubernetes/cmd/kubeadm/app/cmd/phases/workflow.(*Runner).visitAll
k8s.io/kubernetes/cmd/kubeadm/app/cmd/phases/workflow.(*Runner).Run
k8s.io/kubernetes/cmd/kubeadm/app/cmd/phases/workflow.(*Runner).Run
k8s.io/kubernetes/cmd/kubeadm/app/cmd/phases/workflow.(*Runner).Run
k8s.io/kubernetes/cmd/kubeadm/app/cmd/newCluster.func1
k8s.io/kubernetes/cmd/kubeadm/app/cmd/init.go:128
github.com/spf13/cobra.(*Command).execute
github.com/spf13/cobra@v1.8.1/command.go:985
github.com/spf13/cobra.(*Command).ExecuteC
github.com/spf13/cobra.(*Command).Execute
github.com/spf13/cobra.(*Command).Execute
k8s.io/kubernetes/cmd/kubeadm/app.Run
k8s.io/kubernetes/cmd/kubeadm/app/kubeadm.go:47
main.main
k8s.io/kubernetes/cmd/kubeadm/kubeadm.go:29
runtime.main
runtime.proc.go:272
runtime.gosignal
runtime.asm_amd64.s:1700

(kali@mail)-[~]
$ watch -n 1 kubectl get nodes
(kali@mail)-[~]
$ kubectl get nodes
NAME                 STATUS    ROLES    AGE   VERSION
mail-sunbeam.local   Ready    control-plane   76m   v1.31.13
```

## Day 03 (12/09/2025):

### Ojectives:

- deploy the Todo App and Database to Kubernetes.
- Create Deployment and Service YAML files.

### Tools Used:

- Kubernetes YAML files (Deployment, Service)
- kubectl

### Tasks and Implementation:

#### 1. Created Deployments:

a) todoapp-deployment.yaml:

b) db-deployment.yaml: For PostgreSQL or SQLite depending on your app.

## 2. Created Services:

## 3. Applied Configs:

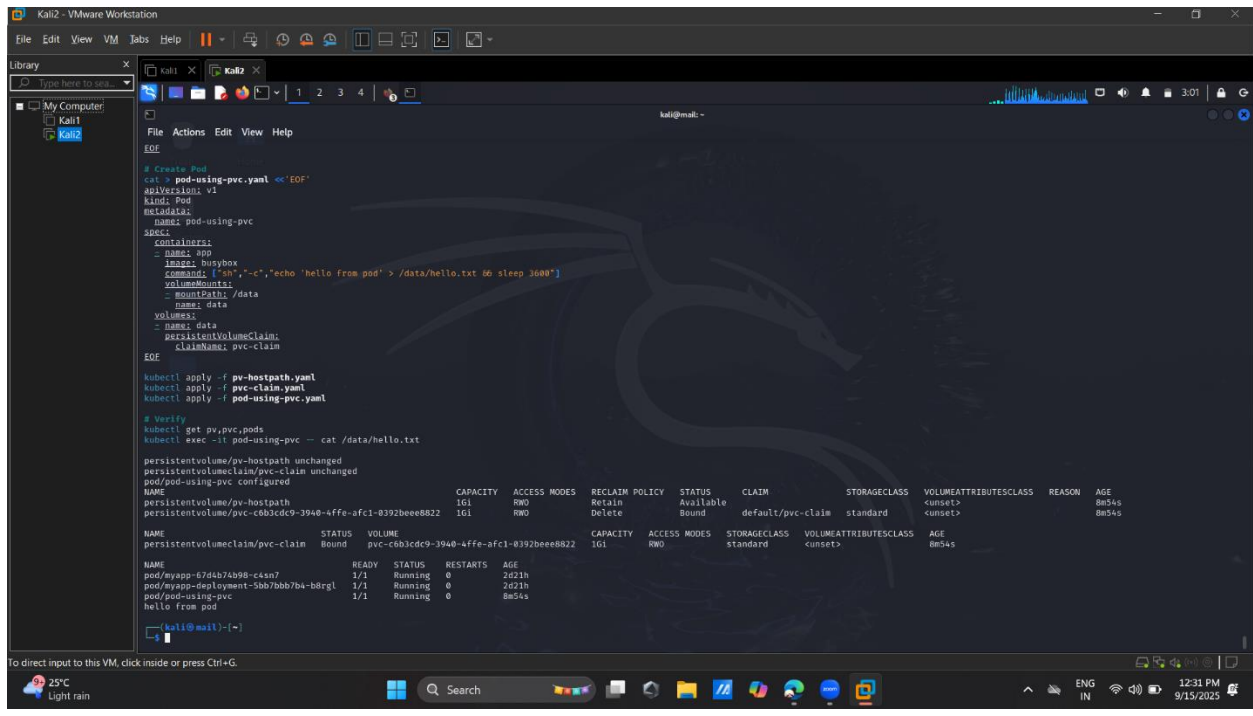
```
kubectl apply -f todoapp-deployment.yaml
```

```
Kubectl apply -f db-deployment.yaml
```

```
Kubectl apply -f todoapp-service.yaml
```

## Outcome:

Todo App is successfully deployed to Kubernetes using declarative manifests.



```
Kali2 - VMware Workstation
File Edit View VM Tabs Help
Library
My Computer
Kali1
Kali2
File Actions Edit View Help
kali@kali: ~
# Create Pod
cat > pod-using-pvc.yaml << EOF
apiVersion: v1
kind: Pod
metadata:
  name: pod-using-pvc
spec:
  containers:
  - name: app
    image: busybox
    command: ["sh", "-c", "echo 'hello from pod' > /data/hello.txt && sleep 3600"]
    volumeMounts:
    - mountPath: /data
      name: data
  volumes:
  - name: data
    persistentVolumeClaim:
      claimName: pvc-claim
EOF

kubectl apply -f pv-hostpath.yaml
kubectl apply -f pvc-claim.yaml
kubectl apply -f pod-using-pvc.yaml

# Verify
kubectl get pv,pvc,pods
kubectl exec -it pod-using-pvc -- cat /data/hello.txt

persistentvolume/pv-hostpath unchanged
persistentvolumeclaim/pvc-claim unchanged
pod/pod-using-pvc configured

NAME                CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS  CLAIM                STORAGECLASS  VOLUMEATTRIBUTESCLASS  REASON  AGE
persistentvolume/pv-hostpath  1Gi      RWO           Retain          Bound   default/pvc-claim    standard                <unset>  <unset>  8m54s

NAME                STATUS  VOLUME                CAPACITY  ACCESS MODES  STORAGECLASS  VOLUMEATTRIBUTESCLASS  AGE
persistentvolumeclaim/pvc-claim  Bound   pvc-cb3cd9-3940-affe-afc1-8392bee8822  1Gi      RWO           Standard                <unset>  8m54s

NAME                READY  STATUS  RESTARTS  AGE
pod/myapp-67dab7ab98-c4sn7  1/1    Running  0          2d21h
pod/myapp-deployment-5bb7bbb7ba-b8rgl  1/1    Running  0          2d21h
pod/pod-using-pvc  1/1    Running  0          8m54s
hello from pod

~(kali@kali)-[~]
To direct input to this VM, click inside or press Ctrl+G.
```

## Day 04 (15/09/2025):

## Objectives:

- Understand and implement Persistent Volume (PV) and Persistent Volume Claim (PVC)
- Set up CI with GitHub Actions and CD with ArgoCD
- Use Helm to template deployments

**Tools & References:**

- Kubernetes (PV, PVC)
- Helm
- GitHub Actions
- ArgoCD

**Tasks and Implementation:****1. Storage with PV/PVC:**

- pv.yaml and pvc.yaml created to persist DB data.

**2. CI with GitHub Actions:**

- .github/workflows/docker-image.yml:

**3. CD with ArgoCD:**

- ArgoCD installed on cluster.
- Linked to GitHub repo to sync Helm chart.

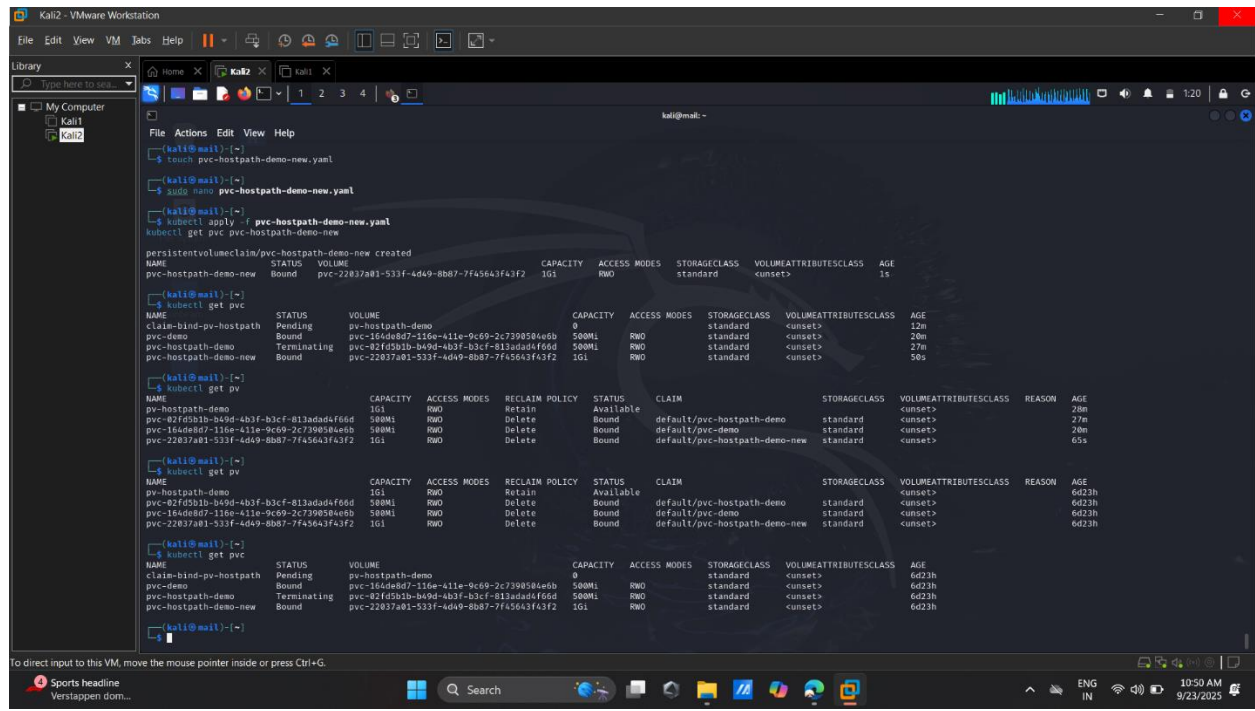
**4. Helm Chart:**

- Created Chart.yaml, values.yaml, templates/ directory for Kubernetes manifests.

**Outcome:**

Complete CI/CD pipeline established. GitHub Actions handles automated builds. ArgoCD handles automated deployment from Helm charts.

---



## Day 05 (22/09/2025) :

### Objectives:

- Set up **Prometheus** and **Grafana** for monitoring
- Create Dashboards and Alerts

### Tools Used:

- Prometheus Operator
- Grafana
- Kubernetes Metrics Server
- Alertmanager

### Tasks and Implementation:

#### 1. Install Prometheus & Grafana:

- Used kube-prometheus-stack Helm chart:

*Helm repo add Prometheus-community <https://prometheus-community.github.io/helm-charts>*

*Helm install monitoring Prometheus-community/kube-prometheus-stack*

## **2. Configure Metrics:**

- Application exposed custom metrics using prometheus\_flask\_exporter.
- Prometheus ServiceMonitor created to scrape metrics from Flask app.

## **3. Grafana Dashboards:**

- Imported pre-built dashboards (Node Exporter, Pod Metrics, Custom Flask App metrics).
- Added visualizations for:
  - API Response Time
  - Number of Requests
  - Container CPU/Memory

## **4. Set up Alerts:**

- Created rules in prometheusRules.yaml

## **5. Integrate Monitoring with CI/CD:**

- Alerting integrated with Slack using Alertmanager.
- Health dashboards embedded into deployment review process.

## **Outcome:**

- Full-stack monitoring and alerting in place with Prometheus + Grafana. Real-time visibility into app and infrastructure metrics.

### 3.Final Architecture Overview:

