

Project Report on

**DMZ**

**Submitted by**

**Gauri Gagare**

Under the guidance of

**Mr. Gajanan Taur**



**Sunbeam Institute of Information Technology,**

**Pune (Maharashtra)**

# CERTIFICATE

This is to certify that the project report entitled **“Devops”**, submitted by **Gauri Gagare** is the bonafide work completed under our supervision and guidance in partial fulfillment for the award of Post Graduate Diploma in IT Infrastructure, Systems and Security (PG-DITISS) of Sunbeam Institute of Information Technology, Pune (M.S.).

Place: Pune

Date:

**Mr. Sandeep Walvekar Mr. Vishal Salunkhe**

Guide Course Coordinator

**Mr. Nitin Kudale**

CEO

Sunbeam Institute of Information Technology

Pune (M.S.) – 411057

# ABSTRACT

**This project focuses on deploying a Flask-based Todo application by applying modern DevOps practices. The work begins with containerization using Docker, where the application and its database are packaged into separate containers. The project then progresses to orchestration using Kubernetes by setting up a two-node cluster, deploying the application through Deployments and Services, and ensuring high availability.**

**To address data persistence, Persistent Volumes (PV) and Persistent Volume Claims (PVC) are configured for the database. Automation is achieved through a CI/CD pipeline: GitHub Actions for continuous integration and ArgoCD for continuous delivery into the Kubernetes cluster. Finally, Prometheus is integrated for monitoring both the Kubernetes infrastructure and the application, enabling proactive system health management.**

**This project demonstrates the complete DevOps lifecycle—from containerization, orchestration, and persistence, to automation and monitoring—providing a scalable, reliable, and observable solution for modern application deployment.**

# INTRODUCTION

**In today’s software industry, the speed and reliability of application delivery play a critical role in business success. Organizations want to deploy applications faster, manage them efficiently, and ensure they can handle user demand while maintaining high availability. Traditional software deployment practices often fail to meet these requirements because they involve manual steps, inconsistent environments, and difficulty in scaling. This is where DevOps practices, containerization, orchestration, and continuous delivery pipelines come into play.**

**This project, which focuses on deploying a Flask-based Todo application, serves as a practical demonstration of how modern DevOps tools and methodologies can be combined to deliver a scalable, reliable, and observable system. The entire project life cycle covers the use of Docker for containerization, Kubernetes for orchestration, Persistent Volumes (PV) and Persistent Volume Claims (PVC) for persistence, GitHub Actions and ArgoCD for CI/CD automation, and Prometheus for monitoring.**

**The Need for DevOps**

**DevOps is a combination of development and operations. It emphasizes collaboration between developers and system administrators, aiming to shorten the software development life cycle and deliver high-quality applications continuously. In the traditional software model, developers wrote code and handed it to operations teams for deployment and maintenance. This often created silos, miscommunication, and deployment failures. DevOps addresses this by unifying processes, using automation, and introducing tools that ensure consistency from development to production.**

**For this project, the DevOps methodology was adopted to ensure the Todo application could be built, tested, deployed, monitored, and scaled without manual intervention. By implementing DevOps practices, the project illustrates how organizations can improve delivery pipelines and achieve faster innovation.**

**Containerization with Docker**

**Before DevOps practices, applications were deployed on physical servers and virtual machines. These approaches required heavy resource allocation, complex dependency management, and often led to the “works on my machine” problem. Containerization solved these challenges by introducing lightweight, isolated environments that package both the application and its dependencies.**

**Docker, the most popular containerization platform, was chosen for this project. Docker allows applications to run consistently across environments. By writing a Dockerfile, the Todo application was packaged into an image. Separate containers were created for the application and its database, ensuring modularity and scalability. The Docker stage of this project highlights the importance of containerization in modern application development, where portability, resource efficiency, and consistency are crucial.**

**Orchestration with Kubernetes**

**While Docker is excellent for creating and running containers, managing containers at scale requires orchestration. If an organization needs to run hundreds of containers across multiple servers, it becomes challenging to handle scheduling, scaling, fault tolerance, and networking manually. Kubernetes, often abbreviated as K8s, is the industry-standard orchestration tool designed to automate these processes.**

**In this project, Kubernetes was set up on two virtual machines: one serving as the master node and the other as the worker node. The master node handles cluster management, including scheduling, maintaining desired states, and responding to failures. The worker node runs the application workloads (pods). Deployments and Services were created to ensure that the Todo application and its database could run reliably, restart automatically in case of failures, and remain accessible to users.**

**Kubernetes provides advanced features such as rolling updates, scaling pods up and down, and load balancing traffic across multiple instances. By using Kubernetes, this project demonstrates how applications can be made resilient, self-healing, and highly available.**

**Data Persistence in Kubernetes**

**One of the unique challenges of containerized environments is data persistence. Containers are ephemeral by design, meaning their data is lost if they restart or are deleted. For applications like a Todo app that relies on a database, losing data would be unacceptable.**

**Kubernetes addresses this issue through Persistent Volumes (PV) and Persistent Volume Claims (PVC). A PV is a piece of storage in the cluster, while a PVC is a request for storage by a user. By configuring PV and PVC, the database data for the Todo application was made permanent, ensuring that even if pods restarted, the data would remain intact. This was a critical step in making the application production-ready.**

**CI/CD Automation**

**Modern applications require frequent updates and continuous delivery. Manually building, testing, and deploying applications introduces delays and increases the risk of human error. This is why a CI/CD (Continuous Integration and Continuous Delivery) pipeline was incorporated into the project.**

**For Continuous Integration (CI), GitHub Actions was used. Every time code is pushed to the repository, GitHub Actions automatically builds and tests Docker images. This ensures that issues are caught early in the development cycle.**

**For Continuous Delivery (CD), ArgoCD was implemented. ArgoCD follows the GitOps principle, where the Git repository serves as the single source of truth. Any changes made in the repository are automatically synced with the Kubernetes cluster. This eliminates the need for manual kubectl apply commands and ensures consistency between development and production environments.**

**Monitoring with Prometheus**

**Even with automation and orchestration in place, monitoring is vital to ensure system health. Applications and infrastructure must be continuously observed to detect failures, performance bottlenecks, or security issues. Prometheus, an open-source monitoring and alerting toolkit, was integrated into the project.**

**Prometheus collects metrics from Kubernetes nodes, pods, and the Flask Todo application. These metrics include CPU usage, memory consumption, pod readiness, and application-specific data. By analyzing these metrics, administrators can proactively identify issues and resolve them before they impact users. Prometheus also supports integration with Alertmanager and Grafana, allowing for advanced alerting and visualization.**

**Bringing It All Together**

**The Todo Application Deployment Project demonstrates the complete DevOps lifecycle. Starting from Docker containerization, moving to Kubernetes orchestration, solving persistence issues, automating deployment pipelines, and finally setting up monitoring, the project showcases how modern applications are deployed and managed in production environments.**

**This comprehensive approach ensures the application is:**

* **Portable (via Docker)**
* **Scalable and fault-tolerant (via Kubernetes)**
* **Persistent (via PV/PVC)**
* **Automated (via GitHub Actions and ArgoCD)**
* **Observable and reliable (via Prometheus)**

**By building this project, I gained practical experience in multiple aspects of DevOps. It not only strengthened my understanding of individual tools like Docker, Kubernetes, and Prometheus but also highlighted how these tools integrate into a larger pipeline that supports modern application development.**

## 1.2 Project Plan

**Table: Activities Details**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr.**  **No.** | **ACTIVITY** | **WEEK** | | | |
| **1** | **2** | **3** | **4** |
| 1 | Project group formation |  |  |  |  |
| 2 | Project work to be started in respective labs |  |  |  |  |
| 3 | First review with PPT presentation |  |  |  |  |
| 4 | Design Use-Case view as per project |  |  |  |  |
| 5 | Design Block diagram as per project |  |  |  |  |
| 6 | Second review with PPT presentation |  |  |  |  |
| 7 | Selection |  |  |  |  |
| 8 | Final review with PPT presentation |  |  |  |  |
| 9 | Implementation coding as per project |  |  |  |  |
| 10 | Testing, Troubleshooting with different techniques |  |  |  |  |
| 11 | Created Soft copy of project and then final hard copy |  |  |  |  |

# SYSTEM DEVELOPMENT AND DESIGN

**System Deployment and Design**

The deployment and design of the Todo Application were carried out by following modern DevOps practices to ensure scalability, reliability, and observability. This section provides details about the architecture, deployment strategy, and design considerations used in the project.

**1. System Architecture Overview**

The system consists of three major layers:

1. **Application Layer** – A Flask-based Todo web application containerized with Docker.
2. **Database Layer** – A separate container for the database (e.g., PostgreSQL/MySQL), responsible for storing user tasks.
3. **Orchestration and Infrastructure Layer** – A Kubernetes cluster with one master node and one worker node to run the application and database pods.

Persistent Volumes (PV) and Persistent Volume Claims (PVC) are used to ensure database data is not lost even if pods restart.

**2. Deployment Workflow**

The deployment process can be described in stages:

1. **Containerization** – The Todo application and its database were packaged into Docker images using Dockerfiles.
2. **Cluster Setup** – A Kubernetes cluster was deployed using kubeadm, with a master node controlling the cluster and a worker node hosting workloads.
3. **Application Deployment** –
   * A **Deployment YAML** file was written to define replicas of the Todo application and database pods.
   * A **Service YAML** file was created to expose the Todo application to external users and enable communication between the app and the database.
4. **Persistence** – Persistent Volumes (PV) and Persistent Volume Claims (PVC) were configured to store database data permanently.
5. **Automation** –
   * GitHub Actions pipeline was used for Continuous Integration to build and test Docker images.
   * ArgoCD was used for Continuous Delivery, automatically syncing the Kubernetes cluster with the GitHub repository.
6. **Monitoring** – Prometheus was deployed to collect both infrastructure and application metrics, ensuring visibility into system health.

**3. System Design Diagram (Textual Description)**

If represented diagrammatically, the design looks like this:

+---------------------------+

| User / Client App |

+------------+--------------+

|

v

+---------+---------+

| Kubernetes |

| Service (NodePort)|

+---------+---------+

|

+--------------+------------------+

| |

+--------+--------+ +---------+---------+

| Todo App Pods | | Database Pod |

| (Flask) | <----------> | (MySQL/Postgres) |

+-----------------+ +-------------------+

| |

| v

| +--------------------+

| | Persistent Volume |

| +--------------------+

|

Monitoring Layer

+--------------------------------------------+

| Prometheus + Alertmanager + Grafana |

+--------------------------------------------+

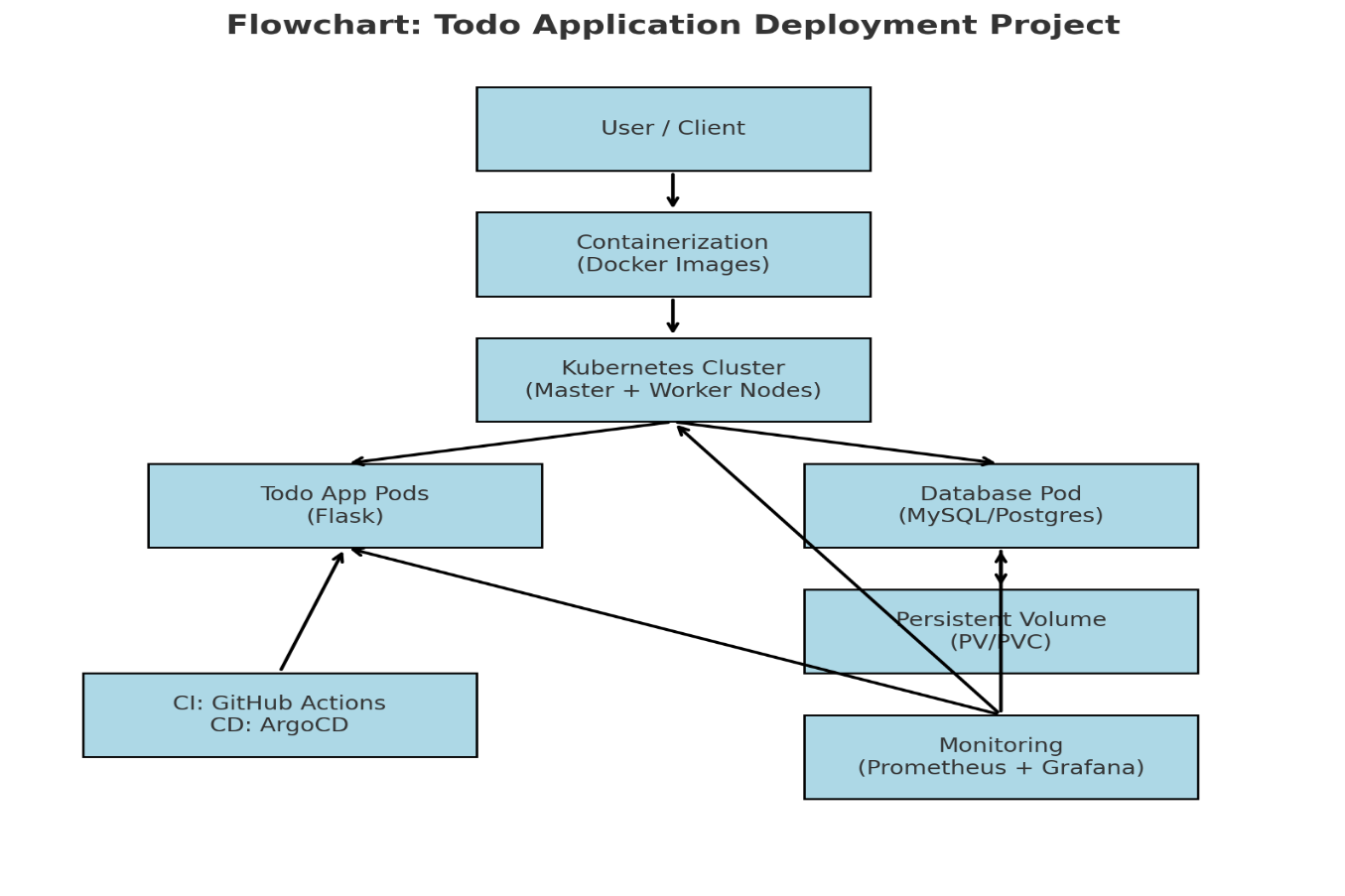
**4. Design Considerations**

* **Scalability**: The Kubernetes Deployment ensures the application can scale horizontally by adding more replicas of the Todo app.
* **Reliability**: Kubernetes automatically restarts failed pods and distributes load evenly across nodes.
* **Persistence**: PV and PVC guarantee that database data is retained even if the database pod is rescheduled.
* **Automation**: GitHub Actions and ArgoCD reduce manual intervention by automating build, test, and deployment steps.
* **Monitoring and Alerting**: Prometheus with Alertmanager provides proactive monitoring, and Grafana can be added for visualization.

**5. Deployment Results**

* The Todo application is accessible through a Kubernetes Service.
* The application and database run in isolated pods, ensuring modularity.
* Data persistence has been achieved, eliminating the risk of losing user tasks.
* Automated pipelines enable faster updates with minimal human error.
* Monitoring ensures high availability and system observability.

Flow chart



**Figure: Flowchart**

## Technology used

**Technologies Used**

This project integrates multiple modern technologies, each addressing specific challenges in software deployment, scalability, and monitoring. The following technologies were used:

**1. Docker**

* **Purpose**: Containerization platform.
* **Role in Project**:
  + The Todo application and the database were containerized using Docker.
  + A Dockerfile was written to build an image of the Flask-based Todo application.
  + Containers allow the application to run in isolated environments with consistent configurations, eliminating the “works on my machine” problem.
* **Importance**: Provides portability, lightweight deployments, and reproducible environments.

**2. Flask (Python Framework)**

* **Purpose**: Backend framework for the Todo application.
* **Role in Project**:
  + Flask was used to create the Todo app API and logic.
  + This application handles CRUD (Create, Read, Update, Delete) operations for user tasks.
* **Importance**: Simple, lightweight, and efficient for small-scale web applications.

**3. Database (MySQL / PostgreSQL / SQLite)**

* **Purpose**: Store user data persistently.
* **Role in Project**:
  + The database container was created separately to store all Todo tasks.
  + It connects to the Flask application container via Kubernetes Service.
* **Importance**: Ensures data persistence, relational structure, and reliability.

**4. Kubernetes (K8s)**

* **Purpose**: Container orchestration platform.
* **Role in Project**:
  + A two-node Kubernetes cluster (master + worker) was set up using kubeadm.
  + The Todo app and database were deployed as pods via Deployment YAMLs.
  + Services were created to expose the application and enable pod-to-pod communication.
  + Ensured scalability (replica sets), high availability, and automated recovery of pods.
* **Importance**: Manages containers at scale, self-healing, load balancing, and rolling updates.

**5. Persistent Volumes (PV) and Persistent Volume Claims (PVC)**

* **Purpose**: Provide durable storage in Kubernetes clusters.
* **Role in Project**:
  + Configured PV and PVC for the database.
  + Ensured that even if database pods restart, the stored Todo data remains intact.
* **Importance**: Critical for stateful applications like databases where data loss is unacceptable.

**6. GitHub Actions**

* **Purpose**: Continuous Integration (CI) platform.
* **Role in Project**:
  + Automated the building and testing of Docker images when code is pushed to GitHub.
  + Integrated with the Docker Hub registry for image storage.
* **Importance**: Reduces manual work, ensures early bug detection, and maintains code quality.

**7. ArgoCD**

* **Purpose**: Continuous Delivery (CD) tool based on GitOps.
* **Role in Project**:
  + Automatically deployed updates from GitHub repository to the Kubernetes cluster.
  + Provided a web UI to monitor applications’ health and synchronization status.
* **Importance**: GitOps-driven delivery ensures that cluster state always matches the Git repository.

**8. Prometheus**

* **Purpose**: Monitoring and alerting system.
* **Role in Project**:
  + Collected metrics from Kubernetes nodes, pods, and the Todo app.
  + Tracked CPU usage, memory, pod readiness, and application-level stats.
  + Configured alerts using Prometheus Alertmanager.
* **Importance**: Provides visibility into system performance and reliability.

**9. Grafana**

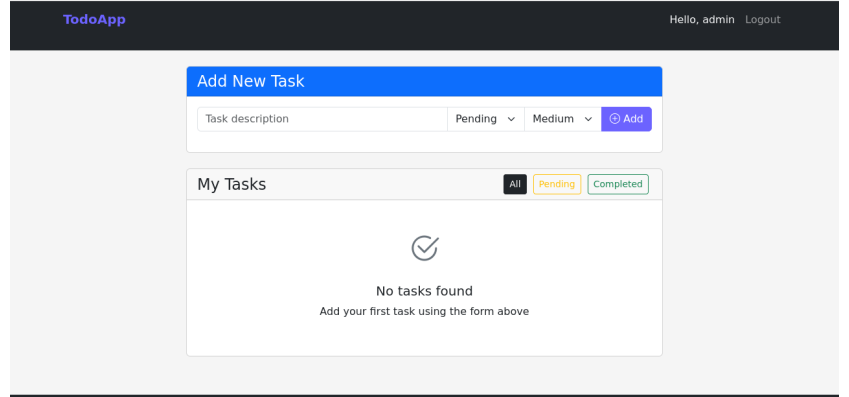
* **Purpose**: Visualization tool.
* **Role in Project**:
  + Used to create dashboards for Prometheus metrics.
  + Displayed application and infrastructure health in an interactive way.
* **Importance**: Simplifies monitoring by visualizing data in real-time dashboards.

**10. Linux (Ubuntu/Debian)**

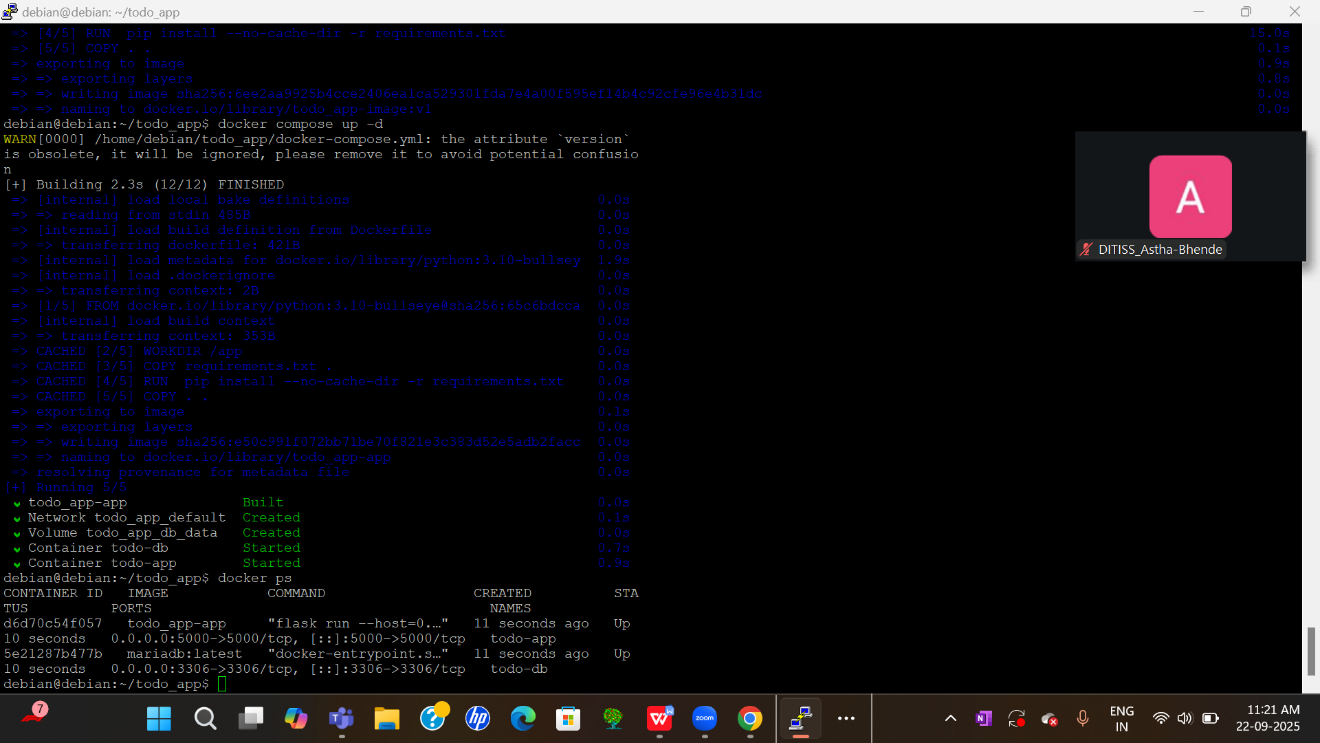
* **Purpose**: Operating System environment.
* **Role in Project**:
  + Docker, Kubernetes, and monitoring stack were installed on Ubuntu/Debian machines.
* **Importance**: Stable, open-source, and widely supported OS for DevOps tools.

# Project Output

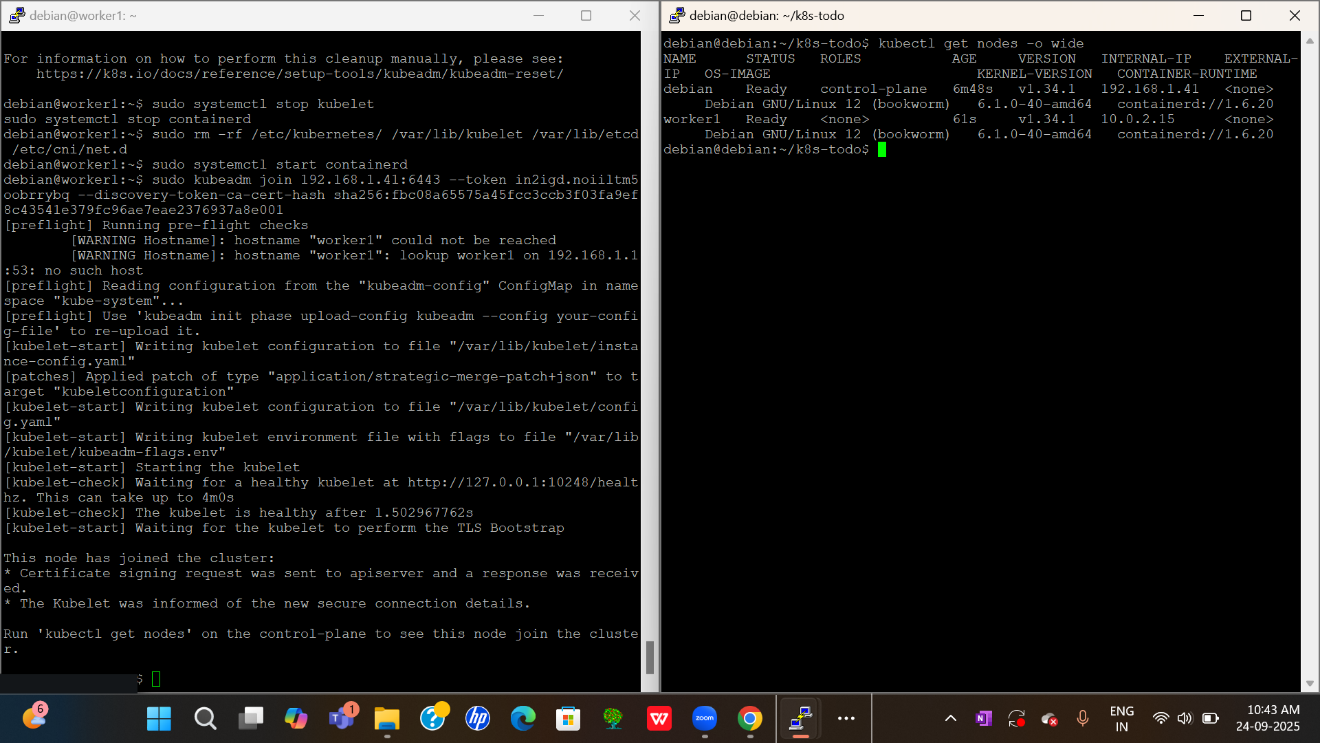
## Todo App



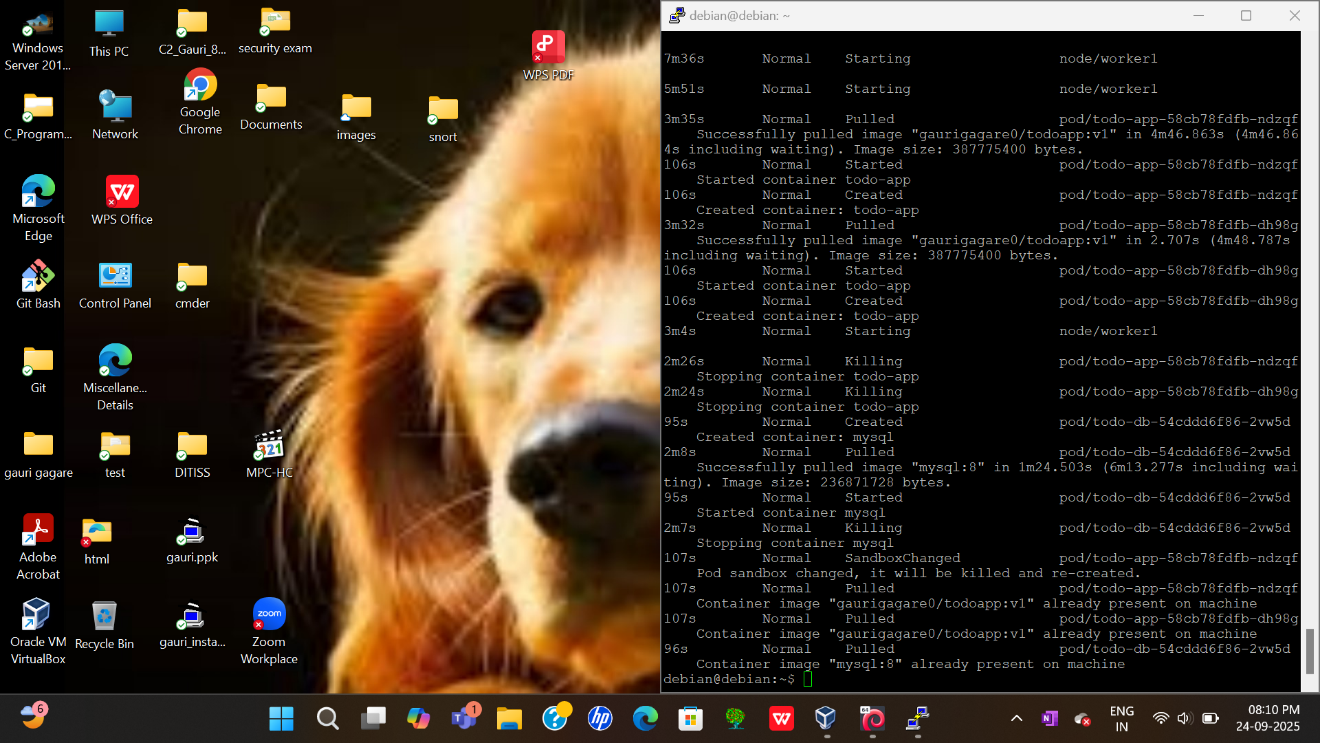
## Container

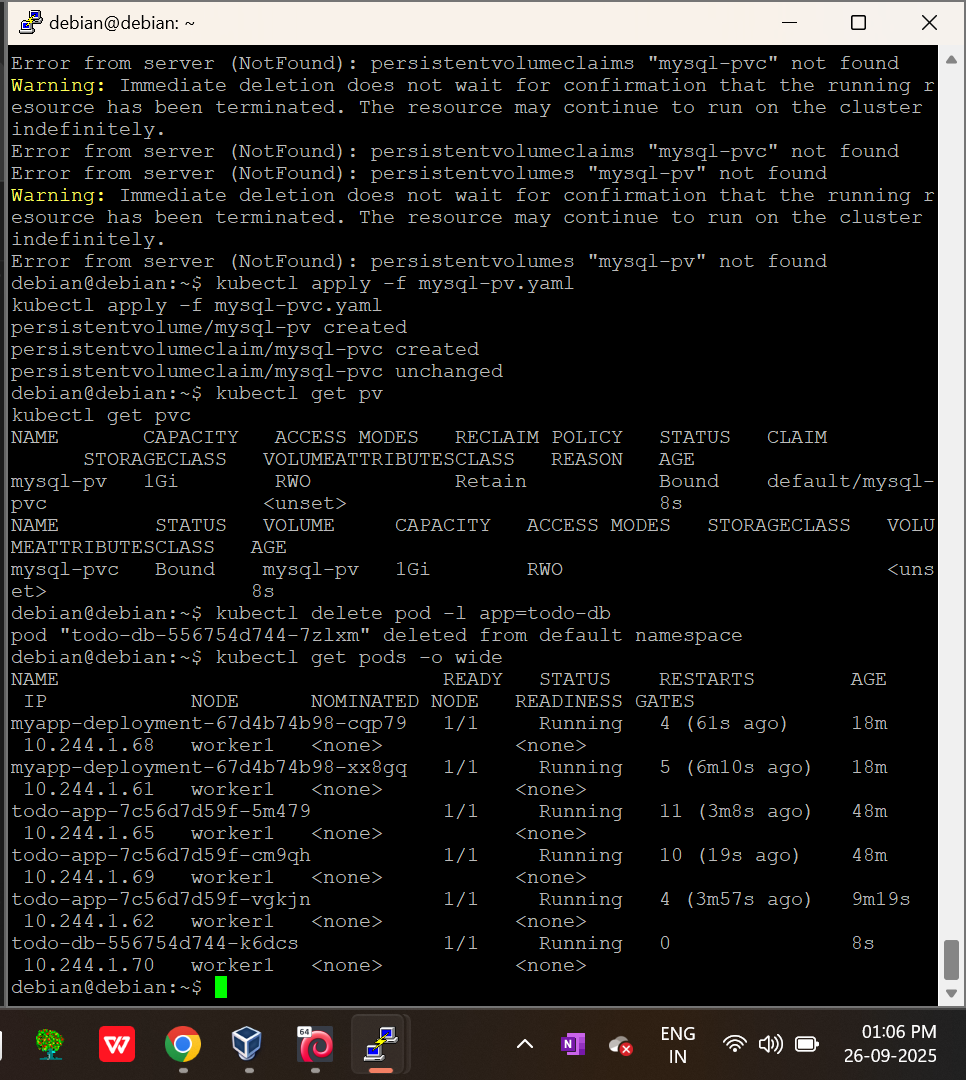


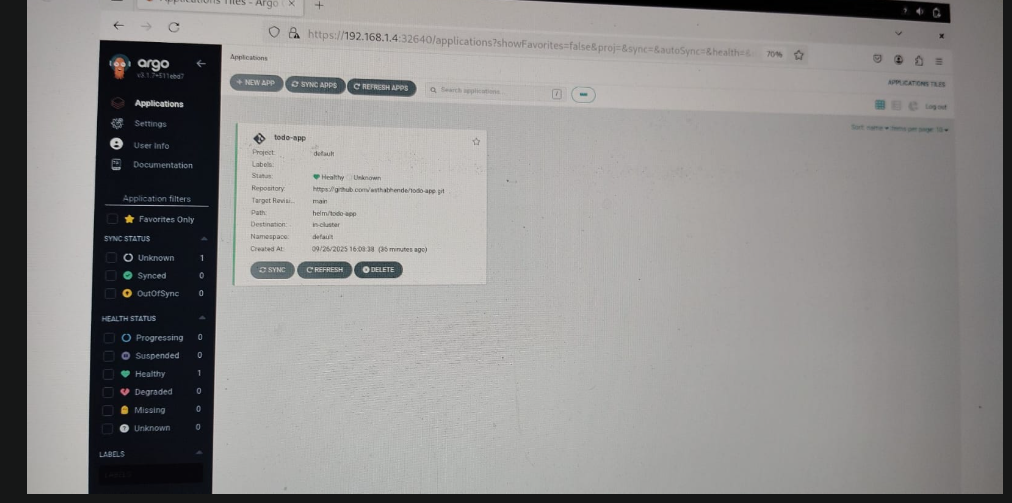
## Web App

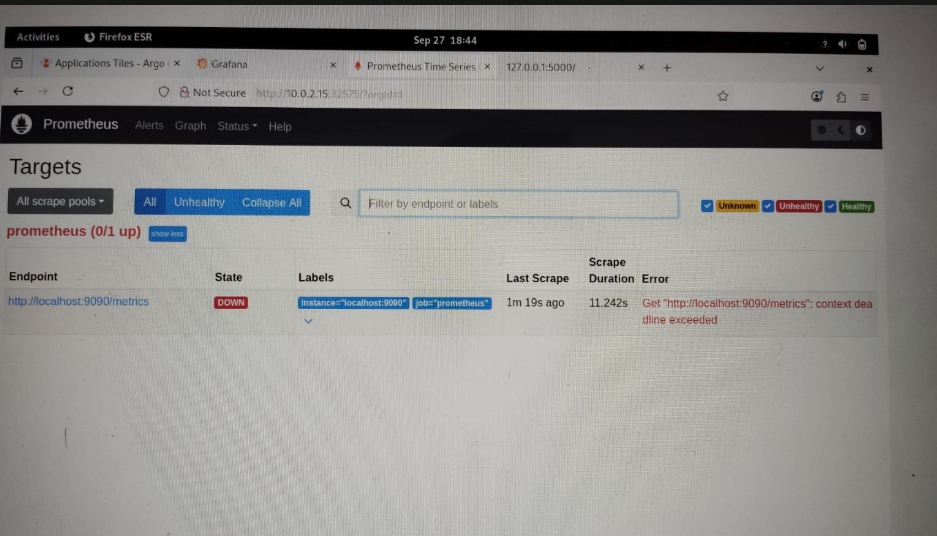


* 1. **Nagios**

****

****

* 1. **Snort** ****

****

# CONCLUSION

## Conclusion

**Conclusion**

The deployment of the Todo application using Docker, Kubernetes, CI/CD pipelines, and monitoring tools represents a complete demonstration of the DevOps lifecycle in practice. This project has not only provided a working application environment but also highlighted the importance of modern technologies in creating scalable, reliable, and observable systems.

At the beginning, the Todo application was containerized using Docker. This simple yet powerful step solved the long-standing issue of application portability by ensuring that the software runs identically across environments. By packaging the Flask application and its database into separate Docker containers, the project achieved modularity, isolation, and consistency. This proved how containerization can simplify deployments and increase developer productivity.

The journey then advanced to Kubernetes, the industry standard for container orchestration. By setting up a two-node cluster with one master and one worker, the project demonstrated how workloads can be efficiently distributed and managed. Kubernetes Deployments ensured that application pods and database pods remained highly available, while Services exposed the application to users and facilitated pod-to-pod communication. This orchestration layer provided self-healing, scalability, and fault tolerance, ensuring that the application would continue to function even in the face of failures.

A significant challenge in containerized environments is the ephemeral nature of containers, which makes data persistence a major concern. By configuring Persistent Volumes (PV) and Persistent Volume Claims (PVC), this project addressed this issue for the database. Even if pods were terminated or rescheduled, the database data was preserved. This step highlighted the importance of persistence in stateful applications and showed how Kubernetes provides solutions to make production workloads more reliable.

Automation was the next critical aspect of this project. By integrating GitHub Actions for Continuous Integration (CI), the project ensured that code changes triggered automated builds and tests of Docker

## Future Scope

**Future Scope**

While this project successfully demonstrates the deployment of a Todo application using Docker, Kubernetes, CI/CD pipelines, and monitoring tools, there is significant scope for future enhancement and expansion.

1. **Integration with Cloud Platforms**
   * Currently, the project runs on local virtual machines. In the future, it can be deployed on cloud platforms like AWS (EKS), Google Cloud (GKE), or Azure (AKS). This would provide better scalability, managed services, and global availability.
2. **Microservices Architecture**
   * The current Todo application is monolithic. A future scope would be to break it into microservices (e.g., authentication, task management, notifications) managed independently in Kubernetes. This would improve modularity and scalability.
3. **Advanced CI/CD Pipelines**
   * While GitHub Actions and ArgoCD automate builds and deployments, advanced pipelines can be designed with automated security scans, performance testing, and canary/blue-green deployments to make the system more robust.
4. **Enhanced Monitoring and Visualization**
   * Prometheus and Grafana provide metrics and dashboards, but AI-driven monitoring and anomaly detection tools can be added for predictive insights. Integration with tools like ELK (Elasticsearch, Logstash, Kibana) or Loki for centralized logging can further improve observability.
5. **Security Hardening**
   * Future improvements can include role-based access control (RBAC) in Kubernetes, secrets management with Vault, and container image scanning to secure the system against vulnerabilities.
6. **Auto-Scaling and Load Testing**
   * Kubernetes Horizontal Pod Autoscaler (HPA) and load testing tools like JMeter or Locust can be integrated to test and automatically adjust the application performance under varying workloads.
7. **High Availability and Disaster Recovery**
   * Multi-cluster deployments across regions can ensure higher reliability and disaster recovery capabilities. This would be crucial for enterprise-level production systems.

By exploring these future scopes, the project can evolve into a more production-grade, secure, and globally scalable system.

# REFERENCES

 Docker Documentation: <https://docs.docker.com/>

 Kubernetes Documentation: <https://kubernetes.io/docs/>

 GitHub Actions Documentation: <https://docs.github.com/en/actions>

 ArgoCD Documentation: <https://argo-cd.readthedocs.io/>

 Prometheus Documentation: <https://prometheus.io/docs/introduction/overview/>

 Grafana Documentation: <https://grafana.com/docs/>

 Flask Documentation: <https://flask.palletsprojects.com/>