DevOps CA2 Project Report:

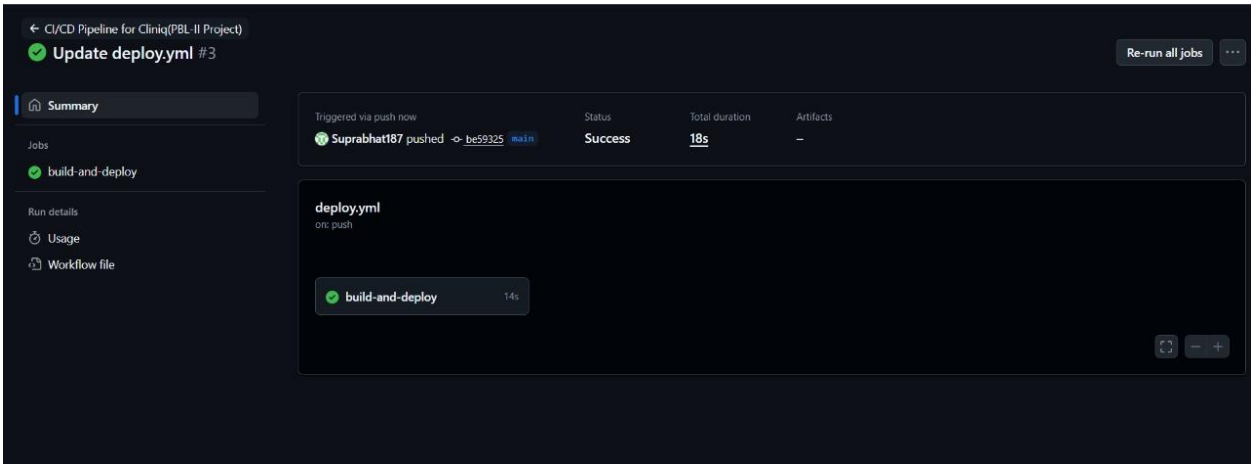
Group Members:
1. Annem Siva Pranav Reddy - 22070122019
2. Aren Dias - 22070122026
3. Argya Yuvraj Singh - 22070122027
4. Kasibhatta Suprabhat - 22070122095

Task 1: Deployment Strategy - GitHub Actions

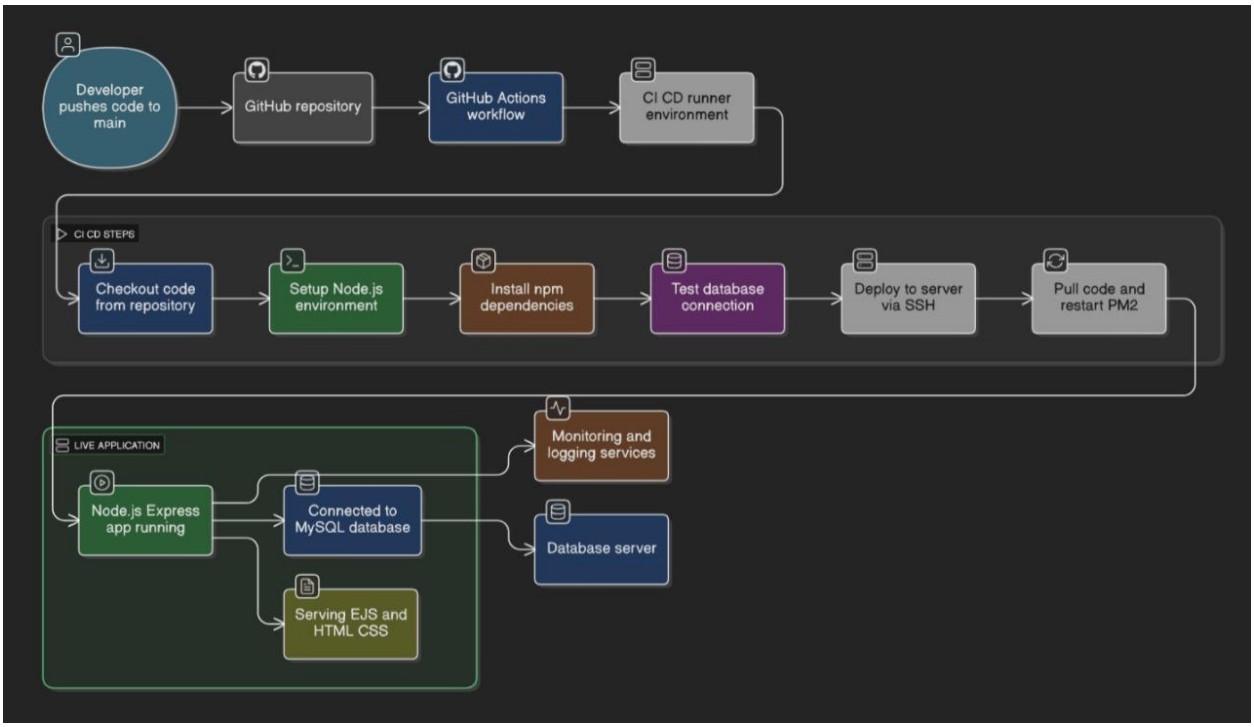
For the deployment strategy, we selected **GitHub Actions** as our CI/CD tool for its seamless integration with our GitHub repository. The pipeline is triggered automatically on every push to the main branch, automating the build and initial verification steps.

About the Tool:
GitHub Actions is an all-in-one CI/CD platform built into GitHub that automates your software development workflows such as building, testing, and deploying code. Workflows are defined using YAML files in the .github/workflows/ directory and can be triggered by events like pushes, pull requests, or scheduled runs.
It is widely used because:

- **Automation**: Reduces manual tasks by automating builds, tests, and deployments.
- **CI/CD Integration**: Implements Continuous Integration and Continuous Deployment within GitHub.
- **Custom Workflows**: Allows flexible, event-driven workflow configurations.
- **Improved Efficiency**: Speeds up the development process and reduces human error.
- **Consistent Code Quality**: Automatically runs tests and checks on every change or pull request.
- **Seamless GitHub Integration**: No need for third-party CI tools—it's built right into your repository.



Pipeline Diagram :



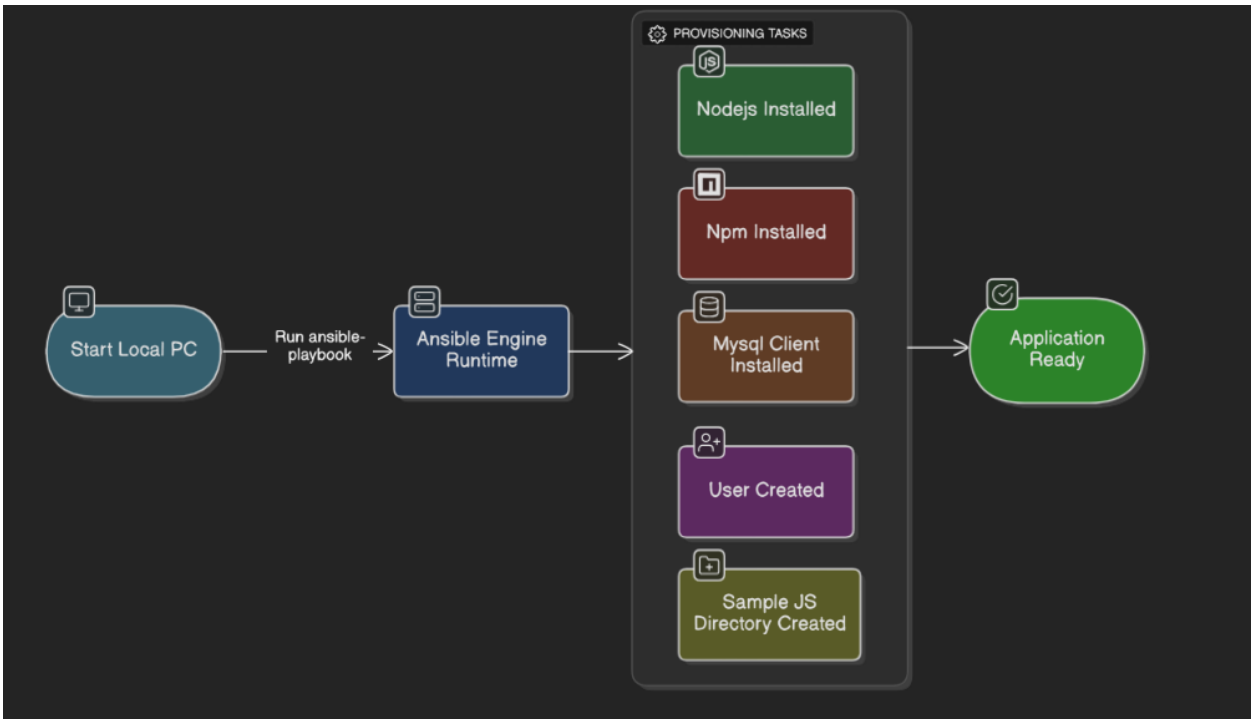
Task 2: Configuration Management & IaC - Ansible

Ansible was used for Infrastructure as Code (IaC) to configure the target environment consistently and reliably.

About the Tool:
Ansible is an open-source automation tool used for software provisioning, configuration management, and application deployment. It uses simple, human-readable YAML files called **"playbooks"** to define tasks and workflows. Ansible operates agentlessly over SSH, making it easy to set up and use across various systems.
It is widely used because:

- **Automation**: Simplifies repetitive tasks like server setup, configuration, and deployment.
- **Agentless Architecture**: No need to install any software on target machines; it uses SSH.
- **YAML Playbooks**: Easy-to-read and write configurations using plain-text YAML files.
- **Scalability**: Can manage a single machine or thousands of servers simultaneously.
- **Consistency**: Ensures environments are configured the same way every time.
- **Cross-Platform Support**: Works across Linux, Unix, and Windows systems.

Architecture Diagram:



Execution Result:

```
supi0@Pavilion:/mnt/c/Users/supi0/Desktop/SYMBIOSIS Institute of Technology/Semester 6/PBL-II/finalproject - Copy/finalproject/ansible$ ansible-playbook -i
hosts.ini playbook.yml

PLAY [Configure Node.js + MySQL environment] ***********************************************************************

TASK [Gathering Facts] *********************************************************************************************
ok: [localhost]

TASK [Update apt cache] ********************************************************************************************
ok: [localhost]

TASK [Install Node.js] *********************************************************************************************
changed: [localhost]

TASK [Install npm] *************************************************************************************************
changed: [localhost]

TASK [Install MySQL client] ****************************************************************************************
changed: [localhost]

TASK [Create application user] *************************************************************************************
changed: [localhost]

TASK [Create application directory] ********************************************************************************
changed: [localhost]

TASK [Deploy sample index.js file] *********************************************************************************
changed: [localhost]

PLAY RECAP *********************************************************************************************************
localhost                  : ok=8    changed=6    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
supi0@Pavilion:/mnt/c/Users/supi0/Desktop/SYMBIOSIS Institute of Technology/Semester 6/PBL-II/finalproject - Copy/finalproject/ansible$
```

Task 3: Containerization & Orchestration - Docker & Kubernetes

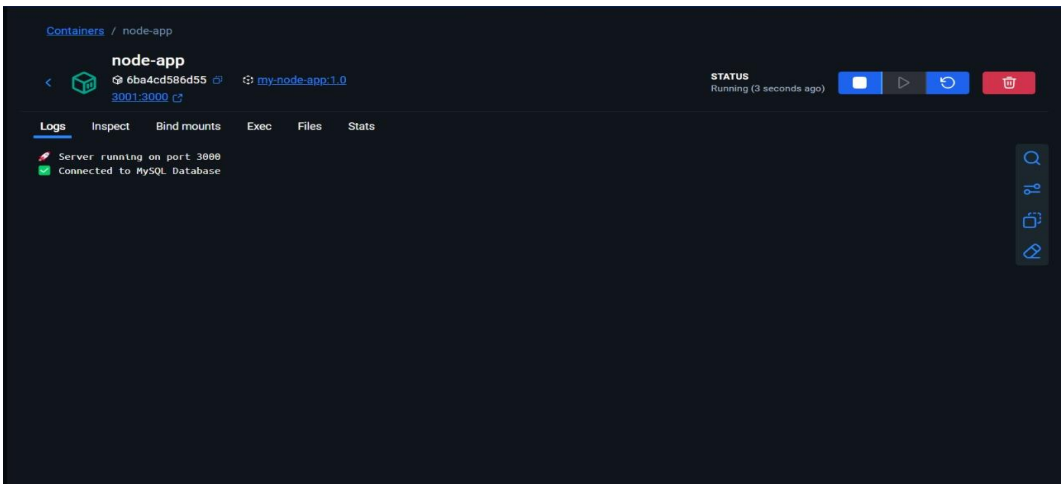
The application and its database were **containerized using Docker** and **orchestrated with Kubernetes** to ensure portability, scalability, and efficient resource management. This modern deployment approach enables consistent environments from development to production and supports dynamic scaling based on demand.

It is widely used because:

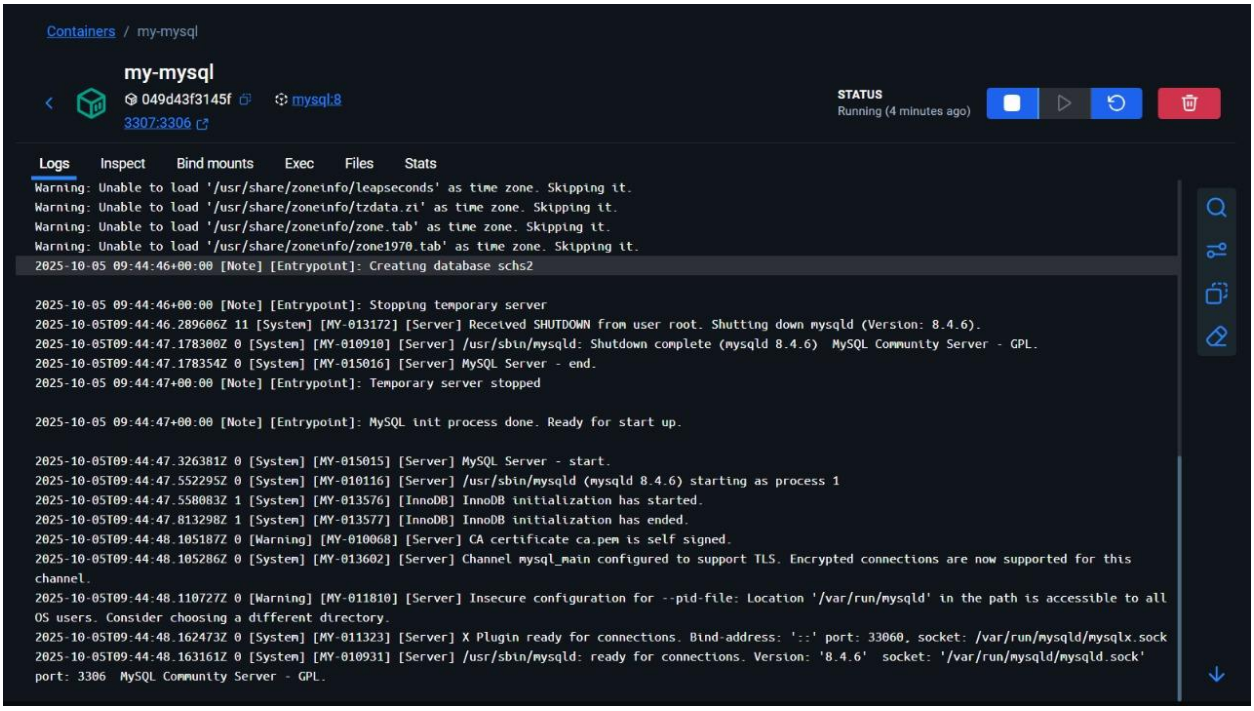
- **Portability**: Docker containers package applications with all dependencies, ensuring they run consistently across environments.
- **Scalability**: Kubernetes automates scaling of applications based on traffic and resource usage.
- **Efficient Management**: Simplifies deployment, monitoring, and updates through orchestration.
- **Isolation**: Containers keep applications and databases isolated for security and stability.
- **Fault Tolerance**: Kubernetes automatically restarts failed containers and distributes workloads.
- **Rapid Deployment**: Speeds up testing and delivery by using container images for consistent rollouts.

Demonstration Screenshots:

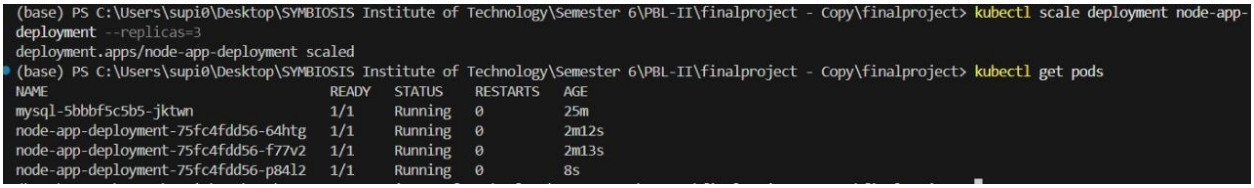
- **Node-js_Container:**



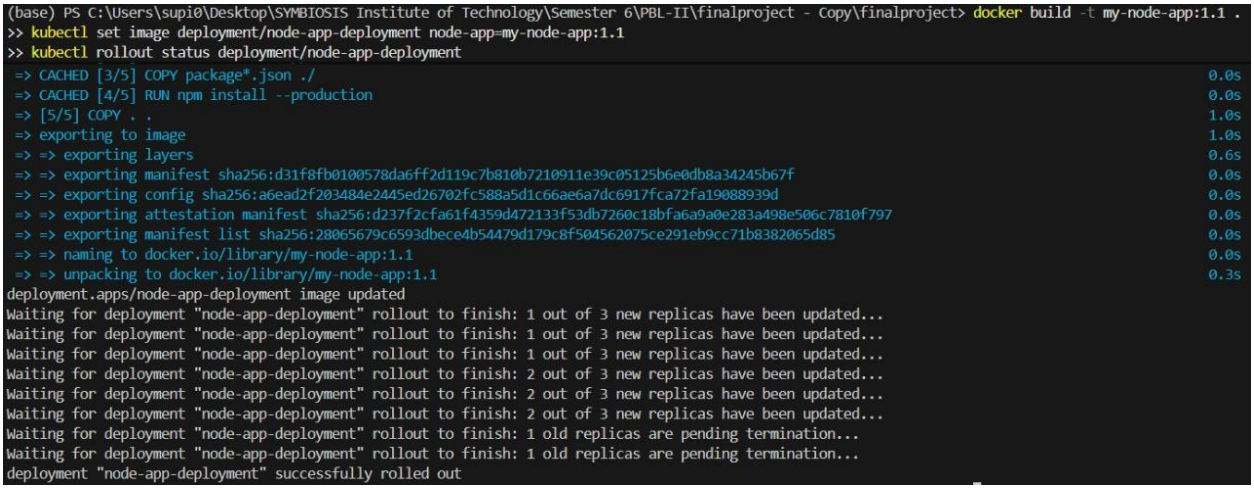
MySql container:



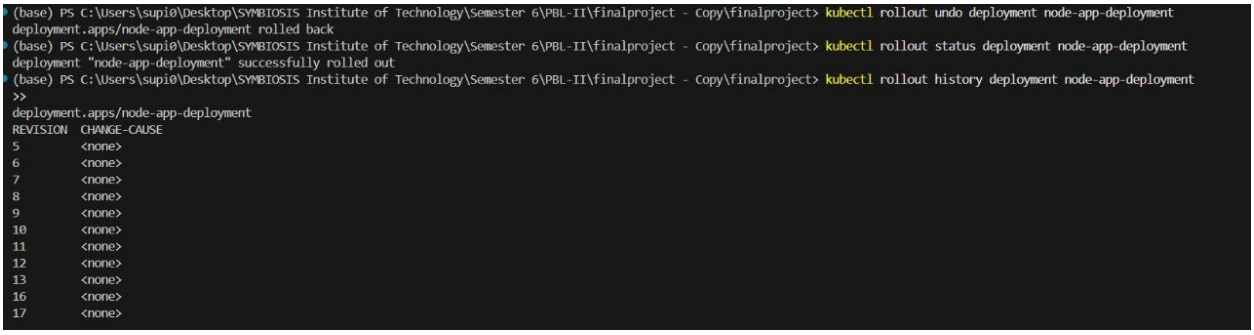
- **Scaling:**



- **Rolling Updates:**



- **Rollbacks:**



Task 4: Monitoring & Logging - Prometheus & Grafana

To ensure application health and performance, a **monitoring stack** was set up using **Prometheus** for metric collection and **Grafana** for visualization. This setup provides real-time insights into system behavior, helping identify and resolve issues proactively.

It is widely used because:

- **Metric Collection**: Prometheus efficiently gathers time-series data from applications and infrastructure.
- **Visualization**: Grafana provides interactive dashboards to monitor key metrics and trends.
- **Real-Time Monitoring**: Enables quick detection of performance issues and anomalies.
- **Alerting**: Prometheus supports alert rules to notify teams of critical conditions.
- **Open Source & Extensible**: Easily integrates with various data sources and services.
- **Improved Reliability**: Helps maintain system uptime and performance through continuous monitoring.

Dashboard:

Task 5: Reflection & Report

Project Architecture

The architecture combines several modern DevOps tools to create a **robust, automated pipeline** for application delivery. It integrates **version control, CI/CD, configuration management, containerization, orchestration**, and **monitoring**, ensuring a consistent, scalable, and reliable development workflow.

It includes:

- **Version Control (GitHub)**: Manages source code and tracks changes through branches and pull requests.
- **CI/CD (GitHub Actions)**: Automates the build, test, and deployment process upon code changes.
- **Configuration Management (Ansible)**: Automates environment setup and ensures consistent system configurations.
- **Containerization (Docker)**: Packages applications and their dependencies into portable containers.
- **Orchestration (Kubernetes)**: Manages, scales, and maintains containers across clusters.
- **Monitoring (Prometheus & Grafana)**: Tracks system metrics and visualizes application performance in real time.

This setup ensures that:

- Code changes are automatically built, tested, and deployed.
- Deployments are consistent across environments.
- The system can scale efficiently and recover from failures.
- Teams can monitor and respond to issues proactively.

Challenges Faced

- Kubernetes Image Pull Error (ImagePullBackOff):
  One of the significant challenges encountered was an ImagePullBackOff error in Kubernetes. This status indicates that a pod could not start because Kubernetes was unable to pull the container image. This was resolved by ensuring the imagePullPolicy was set to IfNotPresent in the deployment manifest, forcing Kubernetes to use the local Docker image that was built on the same machine.

- Prometheus Metrics Not Showing / No Data:
  Even after setting up ServiceMonitor, some metrics like HTTP request latency or error rates may show "No Data" in Grafana.
  This often happens because the application isn't exposing the metrics in the expected format (Prometheus metrics endpoint) or at all.
  Required fixing: add proper Prometheus instrumentation in the Node.js app (prom-client) and ensure metrics are exposed on /metrics

- Service & ServiceMonitor Configuration:
  Matching the labels of the service with the ServiceMonitor is crucial.
  Mistakes here can prevent Prometheus from scraping metrics even if the app is running.
  Ports and paths must match the actual service configuration; wrong ports or targetPort mismatches cause scraping failures.

- MySQL Container Setup in Kubernetes:
  Unlike a local MySQL instance, deploying MySQL in Kubernetes requires persistent storage (PVC) to ensure data isn't lost if the pod restarts. Major challenges faced in this are Configuring proper volume mounts for data persistence: ensuring network connectivity between Node.js pods and MySQL service. Handling initialization scripts to set up schema and seed data. Any misconfiguration can result in pods crashing, CrashLoopBackOff, or inability to connect from the app.

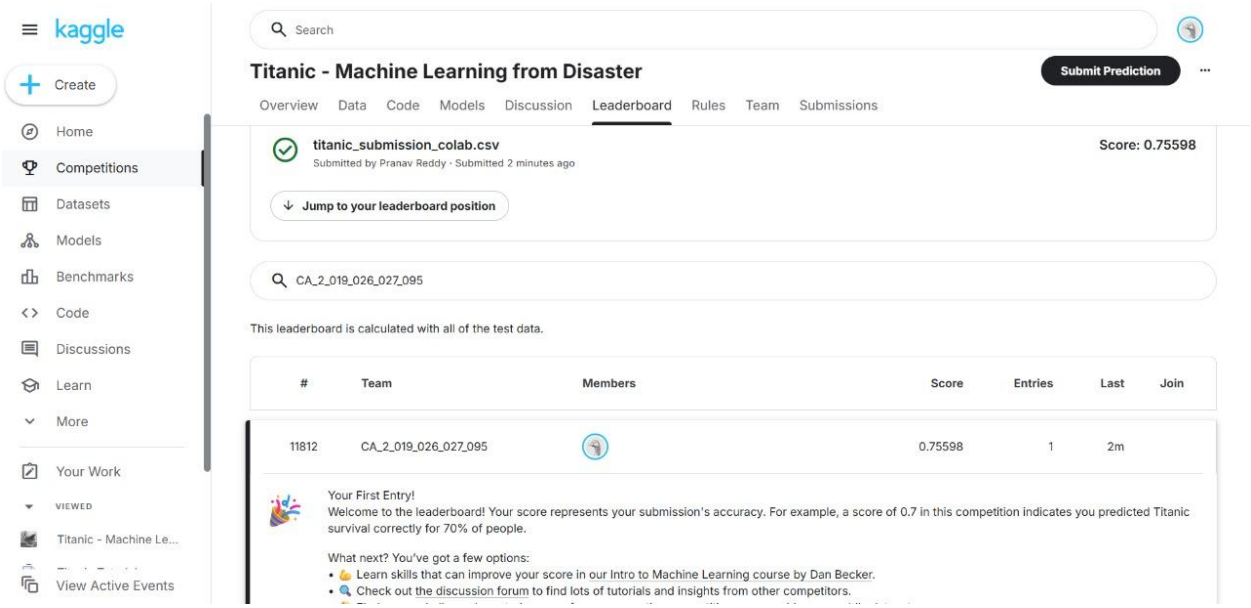
Lessons Learned

- **Automation is Key:** Automating the entire pipeline reduces manual errors and accelerates delivery.
- **Infrastructure as Code (IaC):** Using Ansible ensures consistent and repeatable environment setup.
- **The Power of Containerization:** Docker and Kubernetes provide a powerful combination for building scalable and resilient applications.
- **Observability Matters:** Proactive monitoring with Prometheus and Grafana is essential for understanding application behavior and troubleshooting issues.

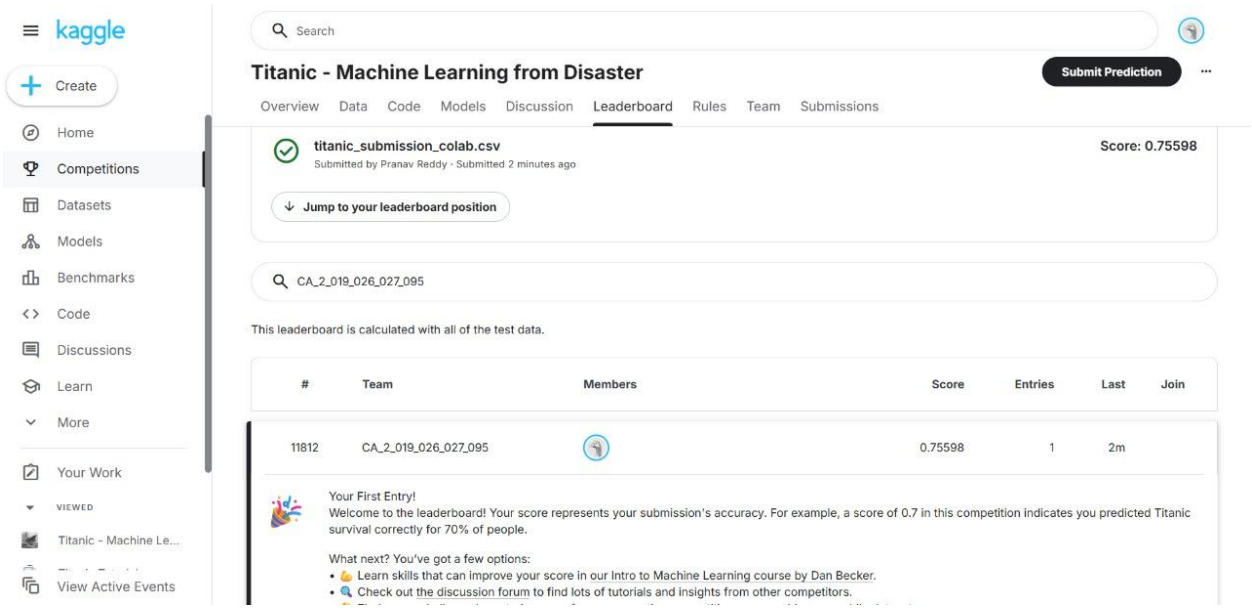
Task 6: Bonus - Kaggle Challenge

As a bonus task, we participated in the **"Titanic - Machine Learning from Disaster"** competition on Kaggle. This demonstrated applying DevOps principles of automation and reproducibility to a machine learning workflow.

Submission and Leaderboard Proof:



Our team's position on the competition leaderboard with a score of 0.75598.



Prediction Script

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load data
train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')
passenger_ids = test_data['PassengerId']

# Prepare data
y_train = train_data['Survived']
X_train = train_data.drop('Survived', axis=1)
full_data = pd.concat([X_train, test_data], axis=0, ignore_index=True)
features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare']
data = full_data[features].copy()

# Handle missing values
data['Age'].fillna(data['Age'].median(), inplace=True)
data['Fare'].fillna(data['Fare'].median(), inplace=True)

# Encode categorical features
data = pd.get_dummies(data, columns=['Sex'], drop_first=True)
data = pd.get_dummies(data, columns=['Pclass'], prefix='Pclass')

# Split back into train/test
```

```
X_train_cleaned = data.iloc[:len(X_train)]
X_test_cleaned = data.iloc[len(X_train):]

# Train model
model = LogisticRegression(solver='liblinear', random_state=42)
model.fit(X_train_cleaned, y_train)

# Make predictions
predictions = model.predict(X_test_cleaned)

# Generate submission file
submission_df = pd.DataFrame({
    'PassengerId': passenger_ids,
    'Survived': predictions
})
submission_df.to_csv('titanic_submission.csv', index=False)

print("Submission file created successfully!")
```