

# 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

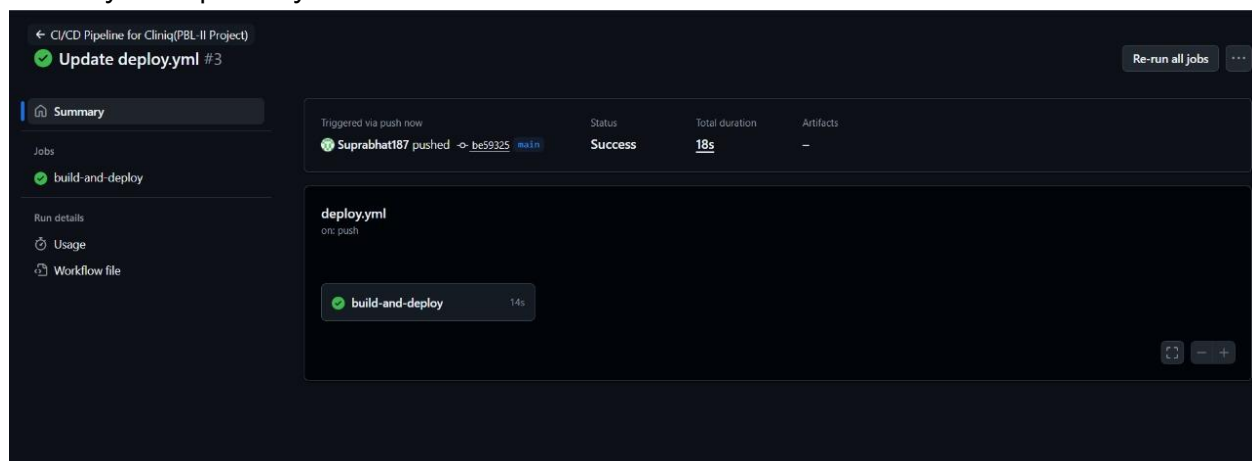
For the deployment strategy, we selected **GitHub Actions** as our CI/CD tool for its seamless integration with our GitHub repository. The pipeline is triggered automatically on every push to the main branch, automating the build and initial verification steps.

About the Tool:

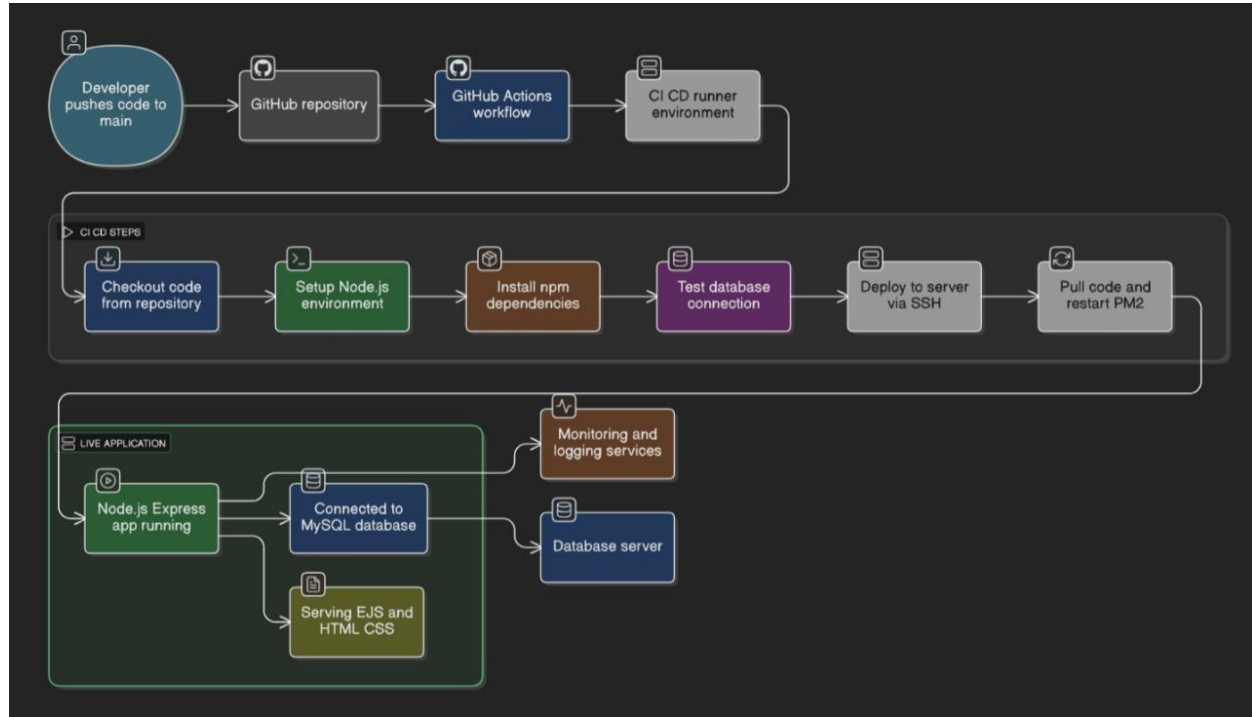
**GitHub Actions** is an all-in-one CI/CD platform built into GitHub that automates your software development workflows such as building, testing, and deploying code. Workflows are defined using YAML files in the `.github/workflows/` directory and can be triggered by events like pushes, pull requests, 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.

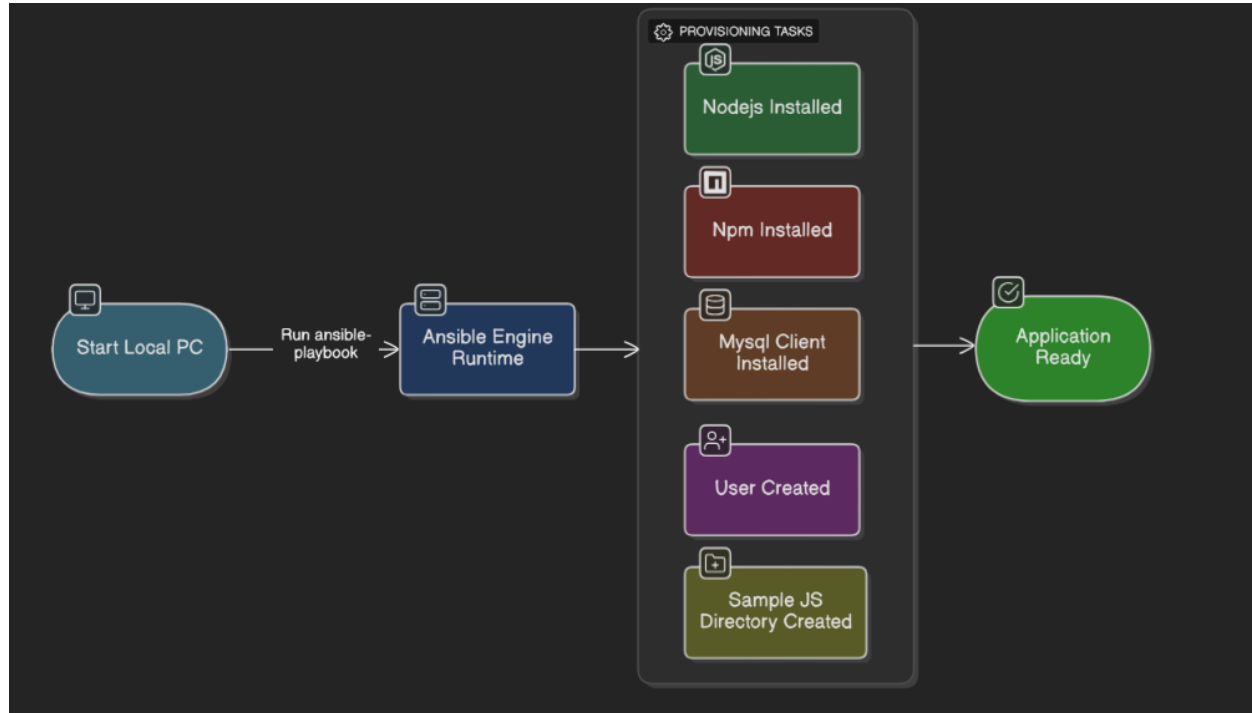
About the Tool:

**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

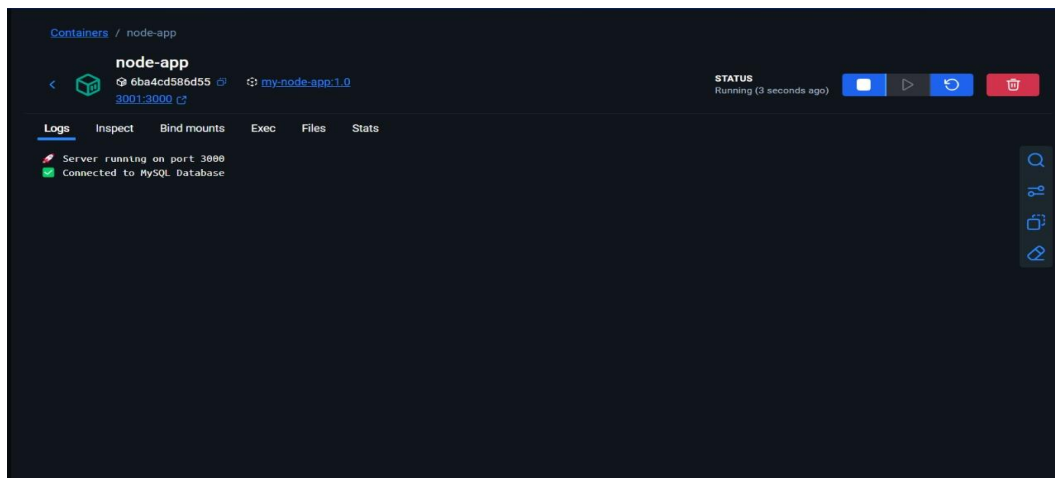
The application and its database were **containerized using Docker** and **orchestrated with Kubernetes** to ensure portability, scalability, and efficient resource management. This modern deployment approach enables consistent environments from development to production and supports dynamic scaling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

The screenshot shows the Docker Desktop interface for a container named 'my-mysql'. The container is running (4 minutes ago). The logs show the MySQL server startup process, including warnings about time zone files and the successful creation of the database 'schs2'.

```
Containers / my-mysql

my-mysql
049d43f3145f mysql:8
3307:3306

Logs Inspect Bind mounts Exec Files Stats

Warning: Unable to load '/usr/share/zoneinfo/leapseconds' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/tzdata.zi' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/zone.tab' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/zone1970.tab' as time zone. Skipping it.
2025-10-05 09:44:46+00:00 [Note] [Entrypoint]: Creating database schs2

2025-10-05 09:44:46+00:00 [Note] [Entrypoint]: Stopping temporary server
2025-10-05T09:44:46.289686Z 11 [System] [MY-013172] [Server] Received SHUTDOWN from user root. Shutting down mysqld (Version: 8.4.6).
2025-10-05T09:44:47.178380Z 0 [System] [MY-010910] [Server] /usr/sbin/mysqld: Shutdown complete (mysqld 8.4.6) MySQL Community Server - GPL.
2025-10-05T09:44:47.178354Z 0 [System] [MY-015016] [Server] MySQL Server - end.
2025-10-05 09:44:47+00:00 [Note] [Entrypoint]: Temporary server stopped

2025-10-05 09:44:47+00:00 [Note] [Entrypoint]: MySQL init process done. Ready for start up.

2025-10-05T09:44:47.326381Z 0 [System] [MY-015015] [Server] MySQL Server - start.
2025-10-05T09:44:47.552295Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.4.6) starting as process 1
2025-10-05T09:44:47.558083Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
2025-10-05T09:44:47.813298Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
2025-10-05T09:44:48.105187Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self signed.
2025-10-05T09:44:48.105286Z 0 [System] [MY-013602] [Server] Channel mysql_main configured to support TLS. Encrypted connections are now supported for this channel.
2025-10-05T09:44:48.110727Z 0 [Warning] [MY-011810] [Server] Insecure configuration for --pid-file: Location '/var/run/mysqld/' in the path is accessible to all OS users. Consider choosing a different directory.
2025-10-05T09:44:48.162473Z 0 [System] [MY-011323] [Server] X Plugin ready for connections. Bind-address: '::' port: 33060, socket: /var/run/mysqld/mysqlx.sock
2025-10-05T09:44:48.163161Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for connections. Version: '8.4.6' socket: '/var/run/mysqld/mysql.sock' port: 3306 MySQL Community Server - GPL.
```

- **Scaling:**

```
(base) PS C:\Users\supi0\Desktop\SYMBIOSIS Institute of Technology\Semester 6\PBL-II\Finalproject - Copy\Finalproject> kubectl scale deployment node-app-deployment --replicas=3
deployment.apps/node-app-deployment scaled
(base) PS C:\Users\supi0\Desktop\SYMBIOSIS Institute of Technology\Semester 6\PBL-II\Finalproject - Copy\Finalproject> kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
mysql-5bbbfc5b5-jktwn	1/1	Running	0	25m
node-app-deployment-75fc4fdd56-64htg	1/1	Running	0	2m12s
node-app-deployment-75fc4fdd56-f77v2	1/1	Running	0	2m13s
node-app-deployment-75fc4fdd56-p84l2	1/1	Running	0	8s

- **Rolling Updates:**

```
(base) PS C:\Users\supi0\Desktop\SYMBIOSIS Institute of Technology\Semester 6\PBL-II\Finalproject - Copy\Finalproject> docker build -t my-node-app:1.1 .
>> kubectl set image deployment/node-app-deployment node-app=my-node-app:1.1
>> kubectl rollout status deployment/node-app-deployment
=> CACHED [3/5] COPY package*.json ./ 0.0s
=> CACHED [4/5] RUN npm install --production 0.0s
=> [5/5] COPY . . 1.0s
=> exporting to image 1.0s
=> => exporting layers 0.6s
=> => exporting manifest sha256:d31f8fb0100578da6ff2d119c7b810b7210911e39c05125b6e0db8a34245b67f 0.0s
=> => exporting config sha256:a6ead2f203484e2445ed26702fc588a5d1c66a6a7dc6917fca72fa19088939d 0.0s
=> => exporting attestation manifest sha256:d237f2cfa61f4359d472133f53db7260c18bfa6a9a0e283a498e506c7810f797 0.0s
=> => exporting manifest list sha256:28065679c6593dbece4b54479d179c8f504562075ce291eb9cc71b8382065d85 0.0s
=> => naming to docker.io/library/my-node-app:1.1 0.0s
=> => unpacking to docker.io/library/my-node-app:1.1 0.3s
deployment.apps/node-app-deployment image updated
Waiting for deployment "node-app-deployment" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "node-app-deployment" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "node-app-deployment" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "node-app-deployment" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "node-app-deployment" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "node-app-deployment" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "node-app-deployment" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "node-app-deployment" rollout to finish: 1 old replicas are pending termination...
deployment "node-app-deployment" successfully rolled out
```

- **Rollbacks:**

```
(base) PS C:\Users\supio\Desktop\SYMBIOSIS Institute of Technology\Semester 6\PBL-II\finalproject - Copy\finalproject> kubectl rollout undo deployment node-app-deployment
deployment.apps/node-app-deployment rolled back
(base) PS C:\Users\supio\Desktop\SYMBIOSIS Institute of Technology\Semester 6\PBL-II\finalproject - Copy\finalproject> kubectl rollout status deployment node-app-deployment
deployment "node-app-deployment" successfully rolled out
(base) PS C:\Users\supio\Desktop\SYMBIOSIS Institute of Technology\Semester 6\PBL-II\finalproject - Copy\finalproject> kubectl rollout history deployment node-app-deployment
>>
deployment.apps/node-app-deployment
REVISION    CHANGE-CAUSE
5            <none>
6            <none>
7            <none>
8            <none>
9            <none>
10           <none>
11           <none>
12           <none>
13           <none>
16           <none>
17           <none>
```

## 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:

The screenshot shows the Kaggle interface for the 'Titanic - Machine Learning from Disaster' competition. On the left is a sidebar with navigation links: Home, Competitions, Datasets, Models, Benchmarks, Code, Discussions, Learn, More, Your Work, and Viewed. The main content area has a search bar and a 'Submit Prediction' button. Below this, the competition title 'Titanic - Machine Learning from Disaster' is displayed with tabs for Overview, Data, Code, Models, Discussion, Leaderboard, Rules, Team, and Submissions. The 'Leaderboard' tab is active, showing a submission 'titanic\_submission\_colab.csv' by 'Pranav Reddy' with a score of 0.75598. A 'Jump to your leaderboard position' button is visible. Below the submission details, a search bar contains 'CA\_2\_019\_026\_027\_095'. A note states 'This leaderboard is calculated with all of the test data.' A table shows the leaderboard with columns: #, Team, Members, Score, Entries, Last, and Join. The table has one entry: #11812, Team CA\_2\_019\_026\_027\_095, Score 0.75598, 1 entry, and last updated 2m. Below the table, a message says 'Your First Entry! Welcome to the leaderboard! Your score represents your submission's accuracy. For example, a score of 0.7 in this competition indicates you predicted Titanic survival correctly for 70% of people.' It also provides next steps: 'Learn skills that can improve your score in our Intro to Machine Learning course by Dan Becker' and 'Check out the discussion forum to find lots of tutorials and insights from other competitors.'

#	Team	Members	Score	Entries	Last	Join
11812	CA_2_019_026_027_095		0.75598	1	2m	

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

The screenshot shows the Kaggle interface for the 'Titanic - Machine Learning from Disaster' competition. The user's submission, 'titanic\_submission\_colab.csv', is highlighted with a score of 0.75598. The leaderboard table shows the user's position as 11812 out of 11812. A message below the table welcomes the user and provides tips for improvement.

#	Team	Members	Score	Entries	Last	Join
11812	CA_2_019_026_027_095		0.75598	1	2m	

## 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!")
```