A Mid Term Progress Report

on

TITANICOPS:
ML-DRIVEN ENGINEERING INSIGHTS

Submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

COMPUTER SCIENCE AND ENGINEERING

SUBMITTED BY

CHANDAN GOYAL (2203580)

KAJAL YADAV (2203585)

UNDER THE GUIDANCE OF

Er. KULJIT KAUR

Er. PALAK SOOD

(April - 2025)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GURU NANAK DEV ENGINERING COLLEGE,

LUDHIANA

TABLE OF CONTENT

LIST OF FIGURES

Chapter 1 - Introduction

In today’s technology-driven world, the seamless integration of machine learning (ML) and DevOps practices has become a cornerstone for developing efficient, scalable, and reliable systems. This project, "TitanicOps: ML-Driven Engineering Insights", focuses on enhancing predictive analytics by combining advanced ML models with an automated CI/CD pipeline and real-time monitoring capabilities.

The project leverages the Titanic Spaceship dataset, which contains complex passenger data such as cryosleep status, spending patterns, and cabin proximity, to predict interdimensional travel outcomes. By employing feature engineering, model optimization, and hyperparameter tuning, the project aims to improve prediction accuracy significantly.

To ensure seamless deployment and operational efficiency, the project integrates a fully automated CI/CD pipeline using tools like Jenkins, Docker, and SonarQube. This setup ensures automated testing, containerization, and deployment, reducing manual interventions and accelerating delivery cycles. Additionally, Prometheus and Grafana are incorporated to monitor system performance in real time, while Slack notifications provide instant updates on build and deployment events.

Furthermore, a user-friendly interface is developed using Streamlit, enabling real-time data exploration and prediction generation. The project is deployed on AWS for cloud scalability and reliability, with Kubernetes handling container orchestration for efficient resource allocation. Additionally, the interface allows for easy interaction with the ML models, providing users with intuitive visualizations and insights to make informed decisions based on the predictions.

By merging predictive modeling with DevOps automation, this project showcases a robust, scalable, and adaptable solution for tackling futuristic challenges in predictive analytics and automated deployments.

In addition to its technical components, "TitanicOps: ML-Driven Engineering Insights" emphasizes the importance of continuous improvement and collaboration across teams. Through the integration of machine learning workflows with DevOps practices, the project fosters a culture of rapid iteration, where new models can be deployed seamlessly and monitored for performance in real time. This approach not only enhances the overall efficiency of predictive analytics but also enables teams to quickly adapt to evolving business requirements and data sources. By leveraging version control for models, automatic rollback features, and comprehensive testing frameworks, the project ensures that the system remains robust, secure, and capable of handling diverse, high-volume workloads as it scales.

Objectives:

Objective 1: To implement ML models and feature engineering to improve model accuracy.

The success of a predictive model largely depends on the selection of appropriate machine learning algorithms and effective feature engineering. This project focuses on utilizing advanced ML models, such as Random Forest, XGBoost, and Gradient Boosting, to improve the accuracy of predictions related to Titanic Spaceship travel outcomes. The dataset is preprocessed using techniques like handling missing values, categorical encoding, and feature scaling to ensure high-quality input data. Additionally, feature selection and hyperparameter tuning are applied to enhance model performance, ensuring that only the most relevant features contribute to the final prediction model.

Objective 2: To implement an automated CI/CD pipeline for faster and reliable software delivery.

A critical component of modern software development is the integration of Continuous Integration (CI) and Continuous Deployment (CD) to ensure fast, efficient, and reliable delivery of machine learning models. This project leverages Jenkins to automate the pipeline, ensuring that every code change undergoes automated testing, validation, and deployment without manual

intervention. The CI/CD workflow is designed to automatically build, test, and deploy updated ML models, reducing deployment time and eliminating human errors. This enables a seamless transition from model development to production while maintaining consistency across environments.

Objective 3: To implement a Streamlit-based GUI for real-time data exploration and predictions.

To make the model more accessible and user-friendly, this project integrates a Streamlit-based graphical user interface (GUI) for real-time interaction. The GUI allows users to explore the dataset, input passenger details, and generate instant predictions based on the trained ML model. Additionally, the interface provides visual analytics, such as feature importance scores and confidence levels of predictions, enabling users to gain deeper insights into the model's decision-making process. This web-based interface enhances usability, making it easy for non-technical users to interact with the ML system without requiring programming knowledge.

Objective 4: To implement real-time notifications and live monitoring for builds and deployments.

To maintain system reliability and operational efficiency, the project incorporates real-time monitoring and notification mechanisms. Using Prometheus and Grafana, key performance metrics such as model accuracy, inference time, and system resource utilization are continuously tracked. Any anomalies or failures in builds and deployments are immediately reported through Slack notifications, allowing quick issue resolution. This proactive monitoring approach ensures that the ML system remains stable, scalable, and highly available, minimizing downtime and improving overall performance.

Chapter 2 - System Requirements

The successful implementation of the TitanicOps project requires a well-defined set of hardware System Requirements.

The TitanicOps project requires a combination of hardware and software resources to ensure seamless execution, deployment, and monitoring. This section outlines the necessary system requirements for both development and production environments.

Hardware Requirements

To support machine learning model training, containerized deployments, and real-time monitoring, the following hardware specifications are recommended:

1. **Development System:**

   For local development, an Intel Core i5 (10th Gen) or AMD Ryzen 5 processor, 16GB RAM (32GB recommended), and at least a 256GB SSD are essential. A dedicated NVIDIA GPU with CUDA support is optional but beneficial for deep learning. A high-speed inter net connection is also crucial for cloud integration and CI/CD workflows.

2. **Production/Cloud Deployment:**

   For cloud deployment, a t2.large AWS EC2 instance or Azure Virtual Machine ensures optimal model inference and API performance. A minimum of 8GB RAM is required, with 16GB recommended for high-traffic applications. A 50GB SSD is necessary for logs, user data, and model artifacts, with scalability options based on dataset size. An NVIDIA Tesla T4 GPU is recommended for faster inference when using deep learning models. Additionally, a secure Virtual Private Cloud (VPC) with firewall configuration

and load balancing is essential for securing API endpoints and managing high request volumes.

Software Requirements

The TitanicOps project consists of multiple software components, including machine learning frameworks, DevOps tools, and monitoring utilities to ensure smooth operation.

1. **Operating System:**

   The project is compatible with multiple operating systems for both development and production environments. For local development, Windows 11, Ubuntu 20.04 LTS, or macOS can be used. For production, Ubuntu 20.04 LTS is preferred due to its stability, security, and strong support for containerized applications.

2. **Programming Languages & Frameworks:**

   The project primarily relies on Python and various ML libraries for data processing and model development. Python 3.8+ is used as the core language for machine learning model development and automation. Libraries like Pandas, NumPy, and Scikit-learn are employed for efficient data manipulation, feature engineering, and model training. XGBoost, RandomForest, and Gradient Boosting algorithms are utilized to build high-accuracy predictive models. Additionally, Streamlit is integrated to create an interactive web-based GUI for real-time data exploration and model inference.

3. **DevOps & CI/CD Tools:**

   To automate the build, testing, and deployment process, several DevOps tools are utilized. Jenkins is implemented to automate the CI/CD pipeline, ensuring reliable software delivery. Docker is used for containerizing applications, maintaining

environment consistency across different platforms. Kubernetes manages and orchestrates scalable deployments in production environments. GitHub Actions serves as an alternative CI/CD automation tool for version control and build management. Additionally, SonarQube ensures code quality and security by performing static code analysis.

4. Monitoring & Notification Tools:

For real-time performance monitoring, logging, and alerting, the following tools are used. Prometheus and Grafana are integrated to track application performance metrics, server health, and API response times. Slack Webhooks provide real-time notifications on build and deployment statuses, alerting developers to system failures or errors.

5. Cloud Services:

The TitanicOps project leverages various AWS cloud services for hosting, storage, and database management. AWS EC2/S3 is used for hosting the ML models and storing logs and datasets. AWS Lambda (optional) provides a serverless function execution environment to handle lightweight processes. AWS RDS/PostgreSQL is utilized as the database system for structured data storage, including user records and model outputs.

Chapter 3 - Software Requirement Analysis

Problem Statement

The TitanicOps project aims to build a machine learning deployment pipeline that enables automated model training, deployment, and real-time monitoring within a DevOps framework. The platform integrates ML models for predictive analytics, ensuring streamlined CI/CD automation, containerized deployments, and real-time performance tracking.

Using tools like Jenkins for automation, Docker for containerization, Kubernetes for orchestration, and Prometheus for monitoring, the system ensures a scalable and reliable ML workflow. Additionally, a Streamlit-based web interface allows users to interact with the model and receive real-time predictions. The system also integrates real-time alerts via Slack, providing instant feedback on build status, deployment health, and model performance metrics.

Modules and Their Functionalities:

Module 1: Data Input Interface

This module serves as the entry point for user interaction with the system, allowing them to input data for predictions and perform tasks related to monitoring and managing the system.

- **Data Input Interface:** The user interface is built using Streamlit, a powerful tool for rapidly creating web applications. This interface allows users to input passenger details (for example, information like age, gender, travel class, etc.) through user-friendly forms. Once the user enters the required details, the interface triggers the prediction process, where the backend processes this data and feeds it into the machine learning model to return predictions in real time.
- **User Profiles:** The interface supports different roles:

- **Standard Users:** These users can input data and view real-time predictions generated by the machine learning models.
- **Administrators:** Administrators have access to additional functionality like managing model deployments, viewing system logs, monitoring performance, and handling updates or system configurations.

Module 2: Data Processing and Feature Engineering

This module is responsible for preparing and transforming raw data into a format suitable for machine learning models. It ensures that the data is clean, well-structured, and contains features that will optimize model performance.

- **Data Preprocessing:**
  - **Handling Missing Values:** The system utilizes libraries like Pandas and NumPy to clean and impute missing values in the dataset. Missing data can be replaced with the mean, median, or mode, or predictive models can be used to estimate missing values.
  - **Feature Encoding:** For categorical variables, techniques like One-Hot Encoding or Label Encoding are employed to convert non-numeric data into numeric format that can be processed by machine learning algorithms.
  - **Normalization/Standardization:** Data is scaled appropriately using techniques like Min-Max Scaling or Z-score Standardization to ensure that all features are on the same scale, helping algorithms perform optimally.
- **Feature Engineering:**
  - **Feature Selection:** Based on the dataset and the problem at hand, the most relevant features are selected using techniques like Correlation Analysis, Random Forest Feature Importance, or Recursive Feature Elimination. This reduces dimensionality and focuses the model on the most influential variables.

- **Feature Transformation:** New features may be derived from the existing ones. For instance, you might create interaction features or polynomial features to capture more complex relationships in the data.

Module 3: Machine Learning Model Training and Evaluation

This module involves selecting, training, and evaluating machine learning models, ensuring that the best model is deployed for predictions.

- **Model Selection:** A variety of machine learning algorithms are considered to optimize the prediction process:
  - **Random Forest:** A robust ensemble method that aggregates predictions from multiple decision trees, improving accuracy and reducing overfitting.
  - **XGBoost:** A powerful gradient boosting method known for its efficiency and predictive power, especially in handling large datasets.
  - **Gradient Boosting:** Another popular boosting algorithm that builds trees sequentially to reduce errors and improve predictions.

Each model is evaluated based on how well it performs with the given dataset, and hyperparameters are tuned to maximize accuracy.

- **Model Evaluation:** Performance metrics are used to assess the quality of the models:
  - **Accuracy:** Measures the overall percentage of correct predictions.
  - **Precision:** Focuses on how many of the predicted positive cases were actually positive.
  - **Recall:** Evaluates how many of the actual positive cases were identified correctly.
  - **F1-Score:** A balance between precision and recall, particularly useful when dealing with imbalanced datasets.

These metrics help in selecting the model that performs best based on the business requirements.

Module 4: CI/CD Automation Pipeline

This module automates the process of testing, validating, and deploying machine learning models, ensuring that changes to the system are tested before being deployed to production.

- **Continuous Integration:**
  - Jenkins and GitHub Actions are integrated into the workflow to automatically test and validate any code changes before they are merged into the main codebase. This ensures that updates to the model or code do not introduce errors or break the system.
  - Unit tests and integration tests are run for each update to ensure the reliability of the code and the model's functionality.
- **Continuous Deployment:**
  - Once changes pass testing, the system automatically deploys the updated model to production. Tools like Docker and Kubernetes ensure that the deployment process is smooth, with minimal downtime. This allows for frequent updates without impacting the user experience.

Module 5: Containerization and Deployment

This module focuses on ensuring that the machine learning model and its dependencies are packaged in a way that makes them portable and easy to deploy across different environments.

- **Dockerized Deployment:**
  - Docker is used to create a container that encapsulates the machine learning model, its dependencies, and any necessary environment configurations. This ensures consistency and eliminates issues that may arise from differences between development and production environments.

- The Docker containers can be easily deployed to different systems, such as on-premise servers or cloud platforms.

- **Kubernetes Orchestration:**
- Once the model is containerized, Kubernetes is used to manage the deployment. Kubernetes handles the orchestration of containers, ensuring that the model scales efficiently in response to varying load demands, and that it remains highly available, even in the event of failures.
- Kubernetes automatically manages the deployment, scaling, and monitoring of containerized applications, making the system resilient and scalable.

Module 6: Real-Time Monitoring and Notifications

This module ensures that the system remains healthy, and any potential issues are detected in real time, allowing administrators to take action promptly.

- **System Monitoring:**
  - Tools like Prometheus and Grafana are employed to monitor the health and performance of the system. Key metrics such as CPU usage, memory consumption, and inference times are continuously tracked to detect bottlenecks and optimize performance.
  - Grafana provides a rich, interactive dashboard for visualizing the data, which helps administrators quickly identify trends or potential issues.
- **Real-Time Alerts:**
  - The system integrates with Email to send real-time notifications for critical events like failed builds, deployment errors, or unusual system behavior (e.g., inference time spikes or resource exhaustion).

Chapter 4 - Software Design

1. INTRODUCTION

PURPOSE: The purpose of the TitanicOps project is to develop an automated machine learning deployment and monitoring system that streamlines ML model training, deployment, and real-time monitoring within a DevOps framework. The system aims to provide a seamless, scalable, and efficient ML workflow by integrating automated CI/CD pipelines, containerized deployments, and real-time performance tracking.

The project focuses on enhancing model accuracy, ensuring continuous integration and deployment (CI/CD), and providing an interactive web-based interface for users to explore data and make predictions in real time. By incorporating real-time notifications and live monitoring, TitanicOps ensures reliability, scalability, and faster issue resolution in production environments.

FEATURES AND FUNCTIONALITIES

The TitanicOps project will offer the following key features and functionalities:

1. **Data Processing and Feature Engineering:** Data processing involves cleaning and transforming data, handling missing values, detecting outliers, and normalizing values for consistency across datasets. Pandas and NumPy provide efficient data manipulation and transformation capabilities. Feature selection techniques focus on retaining relevant features while removing unnecessary or redundant ones, thereby improving model performance and reducing computational complexity.
2. **Machine Learning Model Implementation:** Models like Random Forest, XGBoost, and Gradient Boosting are used to predict outcomes based on input data. Hyperparameter tuning and cross-validation are applied to enhance the model's accuracy and reduce

overfitting. Trained models are stored and versioned for real-time inference and easy deployment, allowing for seamless updates as new data is available.

3. **Automated CI/CD Pipeline:** Jenkins and GitHub Actions automate testing, validation, and deployment, ensuring that only validated, error-free code is pushed to production. Version control maintains system stability by tracking changes over time, while model validation guarantees that the performance remains consistent before deployment to production environments.
4. **Containerization and Deployment:** Docker ensures that models and services are packaged into containers, providing consistency and portability across different environments. Kubernetes is employed to manage scalable, highly available deployments and orchestrate containers across clusters. The solution is compatible with multiple cloud platforms (AWS, GCP, Azure) to provide flexibility and improve disaster recovery strategies.
5. **Real-Time Web Interface (Streamlit GUI):** The Streamlit interface allows for real-time data input, instant predictions, and visual representation of results, making model decisions more understandable for users. The interface also enables users to upload datasets for immediate analysis, providing quick feedback on model accuracy and predictions through dynamic, interactive visualizations.
6. **Real-Time Monitoring and Alerts:** Prometheus and Grafana monitor critical system metrics, including CPU/memory usage, API response times, and model inference speed, ensuring the system is operating optimally. Automated alerts via Slack or email notify the team of any critical issues such as system failures, performance drops, or model drift, enabling rapid troubleshooting and system recovery.
7. **Logging and Debugging Support:** Centralized logging using the ELK Stack (Elasticsearch, Logstash, and Kibana) enables detailed tracking, analysis, and real-time visualization of system performance. Logs are parsed and enriched with context to

quickly identify, troubleshoot, and debug issues across various components. This streamlined approach allows teams to detect and resolve problems swiftly, ensuring high availability, consistency, and optimized system performance across all environments. Additionally, the use of automated alerts based on log anomalies enhances proactive issue detection.

8. **Scalability and Cloud Integration:** The system is designed for horizontal and vertical scaling to meet fluctuating workload demands, ensuring it can efficiently handle increasing data volume and user load. It also supports multi-cloud deployments across AWS, GCP, and Azure, offering flexibility in resource allocation. Additionally, serverless execution, such as AWS Lambda, is used to optimize resource consumption and reduce operational costs, allowing for more efficient handling of less-demanding tasks.

2. COMPONENTS

1. User Interface (UI)

Purpose:

The UI serves as the main interaction hub, enabling users to input data, view predictions, and monitor performance in a user-friendly, responsive environment.

Implementation:

- Use Streamlit for quick UI development with interactive elements.
- Integrate React for dynamic front-end and real-time updates.
- Implement Matplotlib and Plotly for visualizing data insights.
- Ensure full functionality across desktop and mobile devices.
- Implement keyboard navigation, screen reader support, and WCAG standards compliance.

2. Data Processing and Feature Engineering

Purpose:

To clean, preprocess, and transform raw data into a structured format, ensuring it is accurate, consistent, and ready for machine learning models. This enhances the overall accuracy, reliability, and performance of the models by providing high-quality input data.

Implementation:

- Use Pandas and NumPy for data preprocessing and feature extraction.
- Implement One-Hot Encoding, Standardization, and Normalization for feature optimization.
- Automate the data pipeline for continuous ingestion and transformation.

3. Machine Learning Model

Purpose:

To analyze and predict outcomes using advanced machine learning algorithms, ensuring high accuracy and adaptability.

Implementation:

- Train models using Random Forest, XGBoost, and Gradient Boosting.
- Use Scikit-learn and TensorFlow for model training and deployment.
- Store models with Pickle or MLflow for versioning.

4. CI/CD Pipeline

Purpose:

Automates testing, validation, and deployment of machine learning models, ensuring seamless integration and updates.

Implementation:

- Use Jenkins or GitHub Actions for continuous integration.
- Package models with Docker for portability.
- Deploy using Kubernetes for scalability.

5. Real-Time Monitoring and Logging

Purpose:

To continuously track system performance, identify issues in real-time, and enable rapid troubleshooting.

Implementation:

- Monitor with Prometheus and Grafana for system health.
- Use Slack or email for real-time alerts.
- Implement Elasticsearch and Kibana for centralized logging.

6. Deployment and Scalability

Purpose:

To ensure machine learning models are deployed efficiently and can seamlessly scale based on fluctuating demand, maintaining high availability, performance, and resilience. This ensures that the models can handle varying workloads without compromising user experience or system stability.

Implementation:

- Use Docker to containerize models for deployment.
- Deploy on AWS, GCP, or Azure for scalability.
- Implement Kubernetes for orchestration and high availability.

3. METHODOLOGY

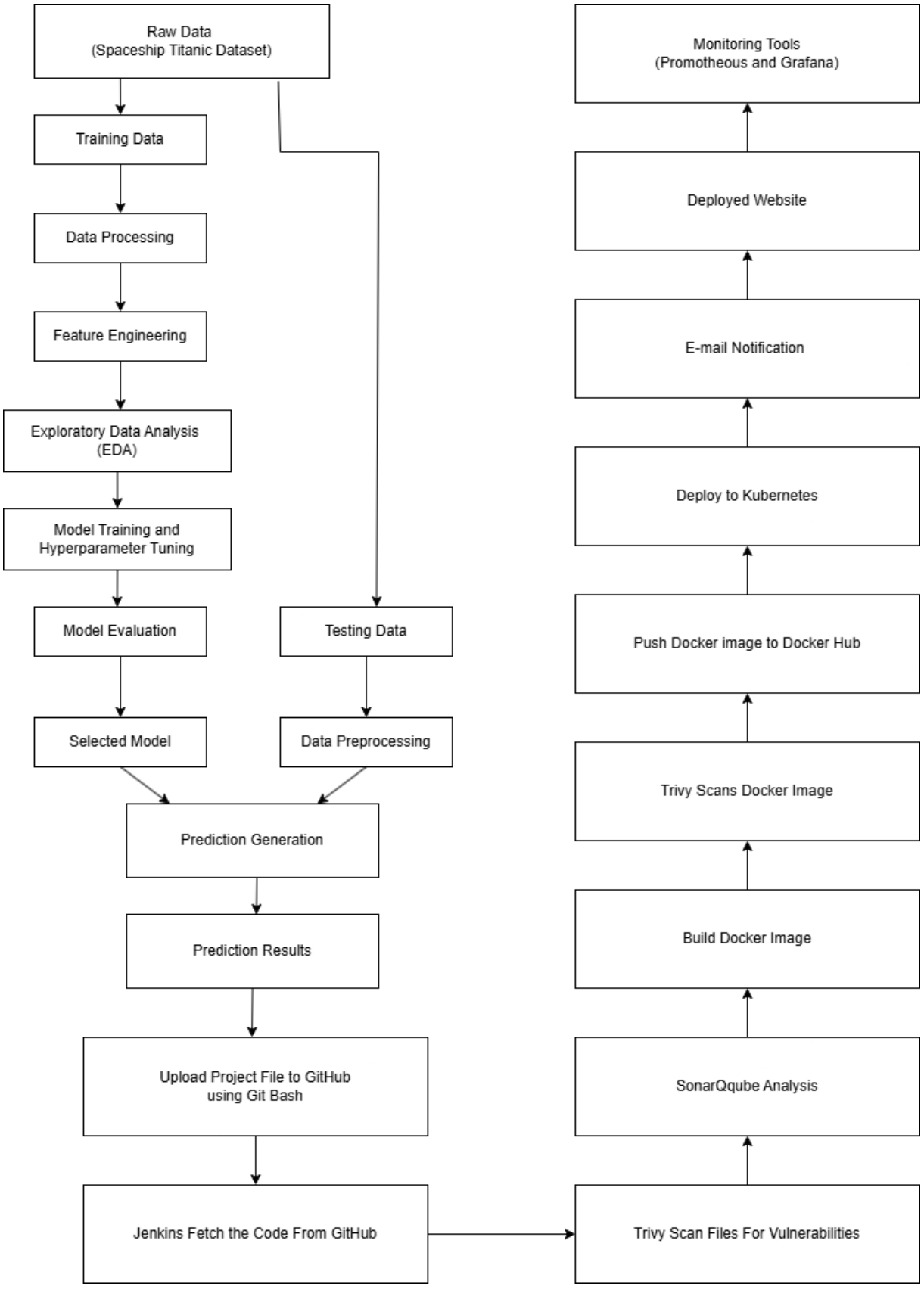

Fig 4.1 Methodology of project

1. Raw Data (Spaceship Titanic Dataset)

The exercise starts with accessing the Spaceship Titanic dataset, a make-believe but structured dataset reflecting passenger data from an interstellar disaster. The dataset contains demographic information, consumption patterns, location in the cabin, travel intentions, and if a passenger was carried or not. These are used as inputs for constructing a classification model and reflect real-life scenarios well-suited for implementing machine learning approaches.

2. Training Data

The data is separated into training and test sets, with the training set usually accounting for 70–80% of the data. This sub-set contains both features and labelled outcomes. It is used by models to learn from past patterns—identifying which sets of variables (such as age or cabin) go with passengers being carried—allowing them to make educated predictions in the future.

3. Data Preprocessing

This phase guarantees the dataset is clean, complete, and machine-learning-friendly. Missing data are addressed using methods such as mean/mode imputation or predictive filling. Duplicates are dropped to avoid bias. Categorical features are encoded numerically using one-hot or label encoding. Scaling (e.g., Min Max or Standard Scaler) guarantees numerical homogeneity. Preprocessing provides a solid basis for training and improves model accuracy.

4. Feature Engineering

New information features are built or old ones are converted. For instance, extracting deck data from the cabin column, consolidating all the spend columns into a single "Total Spend" variable, or creating "Is Alone" from group size. Such engineered features usually unveil underlying patterns and significantly enhance model performance by making inputs more relevant.

5. Exploratory Data Analysis (EDA)

EDA discovers structure, outliers, and patterns through visualization and statistical methods. Histograms, heat maps, and pair plots expose distributions and relationships. Skewed variables, class imbalance, or outliers are detected and resolved. EDA informs feature selection choices, data transformations, and model choice, ensuring that modeling is data-driven.

6. Model Training and Hyperparameter Tuning

Different machine learning algorithms like Random Forest, Decision Trees, SVM, and Gradient Boosting are trained. Each one is run with a variety of hype rparameters—max depth, learning rate, etc.—to optimize performance. Tuning involves Grid Search or Random Search methods on cross-validated data to identify best configurations, avoiding overfitting and increasing accuracy.

7. Model Evaluation

Every model is validated against several performance metrics:

- Accuracy – overall correctness
- Precision – accuracy of positive predictions
- Recall – finding all positives
- F1-Score – equality between precision and recall
- F1-Score – equality between precision and recall

8. Selected Model

The best-scoring model on the selected metrics is now finalized. It is exported through serialization (e.g., with `joblib` or `pickle`) and version-controlled to ensure reproducibility. This chosen model is now the main asset for downstream deployment and inference in production.

9. Testing Data

Reserved throughout the first split, the test dataset mimics unseen real data. It holds only the features (not the target label) and is an unbiased test on the final model's capacity to generalize and predict accurately outside of the training context.

10. Data Preprocessing on Testing Data

To avoid inconsistencies or data exposure, the same preprocessing and transformation processes used in training data are again applied to the test set. The same encoders, scalers, and feature builders are used. Such automation of the pipeline guarantees reliability and integrity.

11. Prediction Generation

The trained model is used to predict on the test data, generating predictions for every passenger—representing the probability of being transported. These results are returned as either probability scores or binary values (e.g., 0 = Not Transported, 1 = Transported), depending on use-case requirements.

12. Prediction Results

Predictions are stored into formatted files (CSV, JSON) for review, analysis, or submission. Results can also be shown on dashboards or APIs.

They are used as performance indicators and drive decisions or actions based on the model output.

13. Push Project Files to GitHub using Git Bash

With the assets, notebooks, and codebase ready, the entire project is committed and pushed to GitHub using Git Bash.

14. Jenkins Pulls the Code From GitHub

Jenkins is set up with a webhook or a scheduled job to fetch the latest code from GitHub. This triggers the CI/CD process. Jenkins pipelines re used to execute automated tests, build scripts.

15. Trivy Scans Files for Vulnerabilities

Prior to development, Trivy performs a full scan of source files to detect insecure packages, misconfigurations, and third-party library vulnerabilities.

This advance detection reduces risk and makes security a development concern.

16. SonarQube Analysis

The source code is statically analyzed using SonarQube to detect bugs, vulnerabilities, security hotspots, code smells, and maintainability. It provides quality ratings and coverage reports to ensure a clean, modular, and scalable code base.

17. Build Docker Image

The application, along with all dependencies, configurations, and environment variables, is containerized using a Dockerfile. The Docker image created is a self-contained entity that maintains uniform performance on any system or cloud provider.

18. Trivy Scans Docker Image

After the Docker image is created, it is again scanned by Trivy to detect any new vulnerabilities added during containerization.

This encompasses base image problems or application layer dependencies to ensure the image is secure end to end.

19. Push Docker Image to Docker Hub

The authenticated Docker image is pushed to Docker Hub, where it is available across environments. Tags are added (e.g., `latest`, `v1.0`) for versioning.

20. Deploy to Kubernetes

Through Kubernetes manifests or Helm charts, the image is deployed into a managed cluster. Kubernetes takes care of pod scheduling, auto scaling, rolling updates, and self-healing features—providing high availability and fault tolerance of the application.

Chapter 5 - Testing Module

Testing is a crucial phase in the TitanicOps project to ensure the reliability, accuracy, and efficiency of the implemented machine learning models, DevOps pipeline, and user interface. This section outlines the testing techniques and the relevant test cases for different components of the system.

1. Testing Techniques

1.1 Unit Testing

Unit testing focuses on testing individual modules and components in isolation to validate their functionality.

- **Test Cases:**
    - Verify ML model training with different feature sets and hyperparameters.
    - Check data preprocessing (handling of missing values, feature scaling, encoding).
    - Validate individual API endpoints for model predictions.
    - Ensure that the CI/CD pipeline correctly triggers builds and deployments.
    - Test Streamlit-based GUI components for proper rendering and interaction.

1.2 Integration Testing

Integration testing ensures that different modules interact correctly and data flows seamlessly between components.

- **Test Cases:**
    - Validate data flow between the ML pipeline and the Streamlit UI for real-time predictions.

- Test the integration of Jenkins, Docker, SonarQube, and Kubernetes to ensure automated deployments.
- Check logging and monitoring mechanisms (Prometheus, Grafana) for real-time tracking.
- Validate real-time notifications through Slack or email alerts.

1.3 Functional Testing

This testing verifies whether the system meets all the functional requirements as per the project objectives.

- **Test Cases:**
  - Ensure the user can upload datasets and receive predictions through the UI.
  - Verify if the ML models produce accurate survival predictions.
  - Test the real-time error detection and alerting system.

1.4 Performance Testing

Performance testing assesses the speed, scalability, and efficiency of the system under different conditions.

- **Test Cases:**
  - Check model inference time with large datasets.
  - Test the deployment pipeline speed with multiple commits.
  - Assess resource utilization of Docker containers and Kubernetes pods.

1.5 Security Testing

Security testing ensures that the system is protected against vulnerabilities and unauthorized access.

- **Test Cases:**
    - Verify authentication and authorization mechanisms in the GUI and API endpoints.
    - Check for SQL injection, XSS, and CSRF vulnerabilities.
    - Ensure secure communication between components using HTTPS.

1.6 Usability Testing

Usability testing assesses how easy and intuitive the system is for users to interact with. This is particularly crucial for user interfaces like Streamlit.

Test Cases:

- Verify that the Streamlit UI is user-friendly and intuitive.
- Ensure the system provides clear feedback for successful or failed actions (e.g., when a user uploads a dataset).
- Test accessibility features (keyboard navigation, screen reader compatibility).

1.7 Regression Testing

Regression testing ensures that new updates or changes to the system do not break any existing functionality.

Test Cases:

- Verify that previously passed unit tests still pass after new code is added.
- Test that the ML model's core features (like prediction accuracy) have not been negatively impacted by updates.
- Check that any changes to the CI/CD pipeline or infrastructure do not affect previous build and deployment cycles.

Chapter 6 - Performance of the Project Developed

The TitanicOps project has made significant progress in terms of feature implementation, system stability, and efficiency. The development so far has focused on building a robust machine learning pipeline, a streamlined DevOps workflow, and a responsive user interface. Below is an analysis of the current system performance based on various key aspects.

1. Machine Learning Model Performance

- **Accuracy & Model Metrics:**
  - **Ensemble Methods:** Implement ensemble techniques such as Random Forests or XGBoost alongside the existing model to boost prediction accuracy through model averaging.
  - **Cross-validation:** Expand the cross-validation process to ensure the model generalizes well across different datasets and conditions, potentially improving robustness.
- **Inference Speed:**
  - **Model Quantization:** In addition to using ONNX or TensorFlow Lite, applying model quantization techniques can further reduce inference time without compromising accuracy.

2. DevOps and CI/CD Pipeline Performance

- **Automated Deployment & Testing:**
  - **Canary Deployments:** Integrate canary deployment strategies to reduce risk during updates and ensure new features are gradually rolled out, improving the reliability of the deployment process.

- **Advanced Testing:** Implement load testing and stress testing into the CI/CD pipeline to better simulate real-world conditions, ensuring the system handles unexpected spikes in traffic and usage.

- **Build & Deployment Efficiency:**
  - **Parallelized Builds:** Introduce parallel builds and testing pipelines, enabling faster validation and deployment times, potentially reducing build times even further.
  - **Artifact Caching:** Implement artifact caching strategies in CI/CD to avoid redundant builds by storing previously built artifacts (such as compiled code, Docker images, or dependencies) in a cache. This allows the pipeline to reuse them in subsequent builds, speeding up the process by eliminating the need to rebuild or re-download components that haven't changed.

3. Web Application and User Experience

- **Responsiveness:**
  - **WebSocket Integration:** For real-time updates and notifications (e.g., when predictions are ready), consider using WebSockets to establish a persistent connection between the front-end and back-end, providing immediate results to users.
  - **Frontend Optimization:** Improving frontend performance by utilizing techniques like lazy loading and code splitting ensures that only the necessary resources are loaded initially, reducing the time to first render. This results in faster page loads, a more responsive UI, and a smoother user experience, especially for larger applications.
- **Scalability:**
  - **Auto-scaling:** Implement auto-scaling for the backend using Kubernetes to automatically adjust resource allocation (e.g., CPU, memory, replicas) based on traffic patterns or workload demands.

4. System Monitoring and Logging

- **Real-Time Monitoring:**
  - **Alerting and Anomaly Detection:** Enhance Prometheus & Grafana by implementing automated alerting based on predefined thresholds (e.g., high CPU usage, memory spikes, or API latency) to notify the team of potential issues. Additionally, integrate anomaly detection algorithms to identify unusual patterns or outliers in system.
- **Error Handling & Logging:**
  - **Distributed Tracing:** Integrating distributed tracing with tools like Jaeger or OpenTelemetry provides a comprehensive view of how requests propagate across different microservices. This enables better identification of latencies, resource contention, and failure points, ultimately improving system reliability and debugging efficiency. It also allows teams to monitor the overall health of the application, optimize performance, and ensure quicker issue resolution.

5. Security & Reliability

- **Authentication & Authorization:**
  - **OAuth Integration:** Consider integrating OAuth 2.0 for authentication alongside JWT to allow users to log in with existing credentials from other trusted providers, improving security and user convenience.
  - **Two-Factor Authentication (2FA):** Implement two-factor authentication for more secure access control, especially for sensitive operations or admin interfaces.
- **System Uptime & Reliability:**
  - **Multi-region Deployment:** To further improve reliability, implement a multi-region or multi-availability-zone deployment strategy, reducing the risk of downtime in case of regional failures.

Chapter 7 - Output Screens

The image shows the AWS EC2 console displaying a list of running instances. It includes details such as the instance names, IDs, instance types, status checks, and availability zones. The instances shown are part of a CI/CD infrastructure, with notable ones including jenkins-master for continuous integration and eks-space-node instances, likely used for Kubernetes cluster deployment. The environment appears to be hosted in the Asia Pacific (Singapore) region.

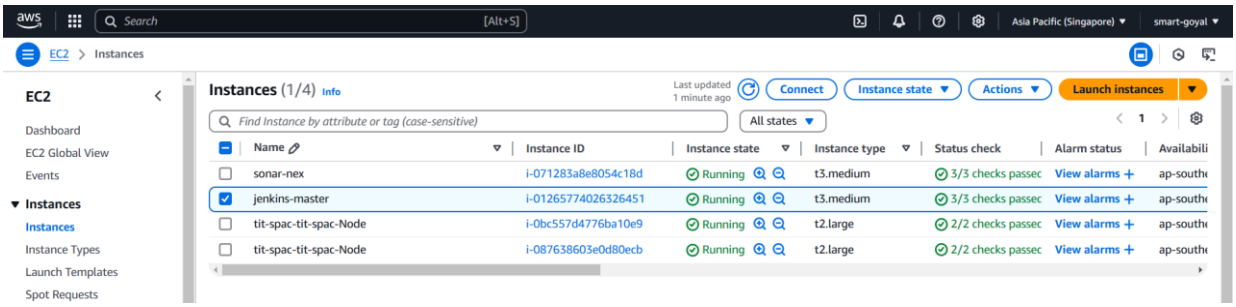

Fig 7.1 AWS EC2 Console

The image shows the AWS CloudFormation console displaying two successfully created stacks. These stacks are related to Amazon Elastic Kubernetes Service (EKS), with one managing nodes and SSH access, and the other handling the EKS cluster infrastructure. The "CREATE_COMPLETE" status indicates that the stacks were successfully deployed.

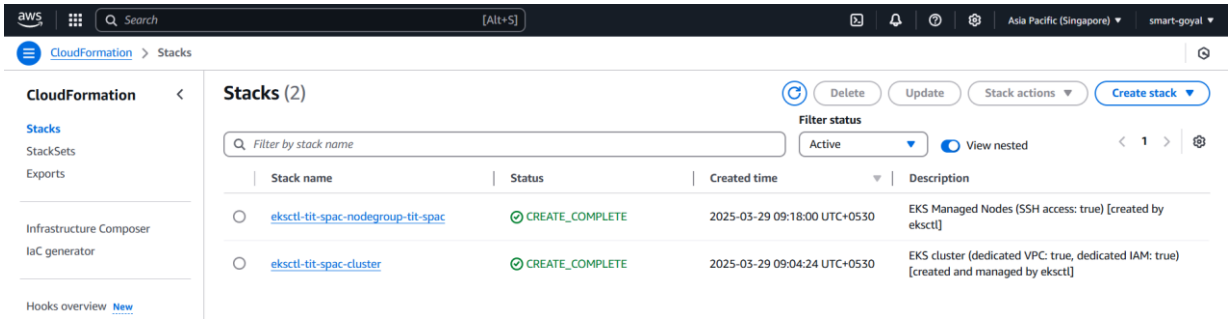

Fig 7.2 AWS CloudFormation Stacks

The dashboard displays pipeline information including build status, names, durations, and other metrics. It appears to be showing at least one pipeline with its current status (indicated by a green

circle), along with timestamps for the last success and last failure. The interface includes options to view build history, project relationships, and manage Jenkins items.

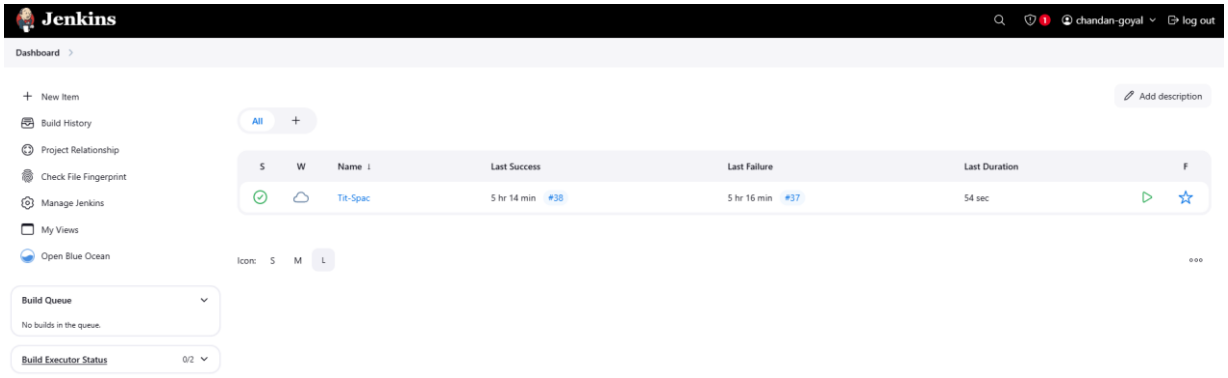

Fig 7.3 Jenkins Dashboard

The Jenkins pipeline visualization for "Tri-Spec" also features detailed logs for each build stage, providing insights into errors or warnings for easier troubleshooting. Additionally, the build history is available for quick reference, allowing team members to compare the success or failure of previous builds. The integration of SonarQube quality gate information helps track code quality in real-time, ensuring that the project maintains high standards throughout the development lifecycle.

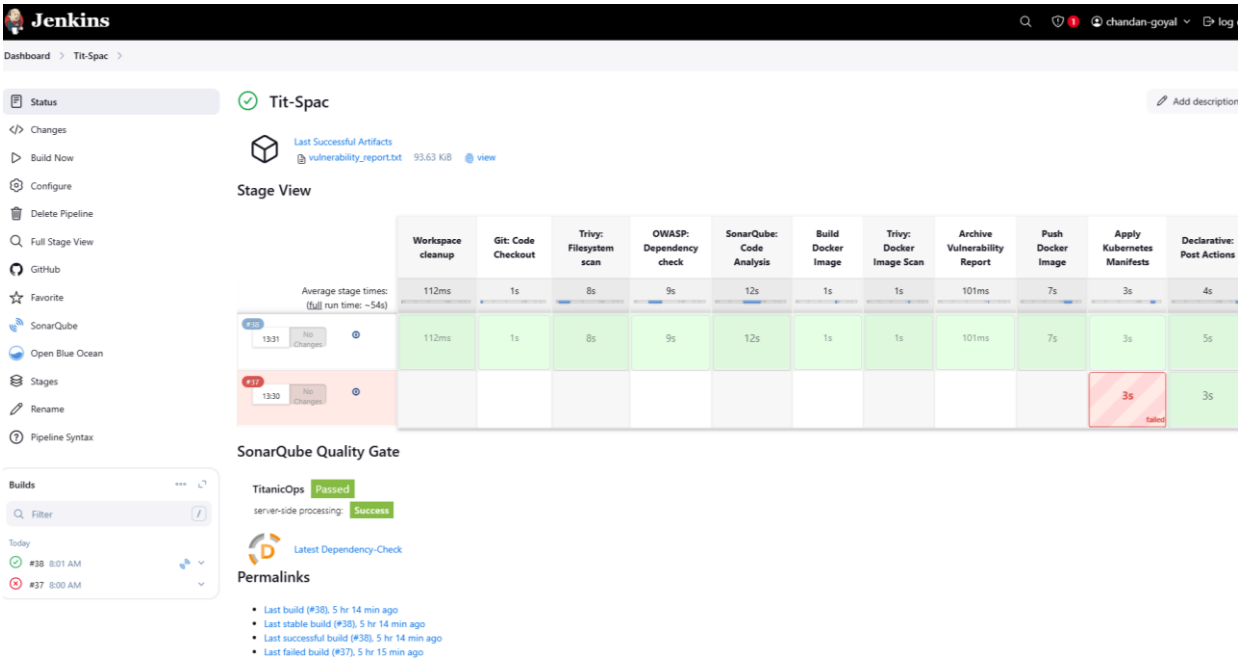

Fig 7.4 Jenkins Pipeline

The image shows a Jenkins build pipeline for "Tit-Spec" with a series of connected stages displayed in a linear workflow. Each stage is represented by a circle with green checkmarks indicating successful completion, connected by lines showing the progression from start to end.

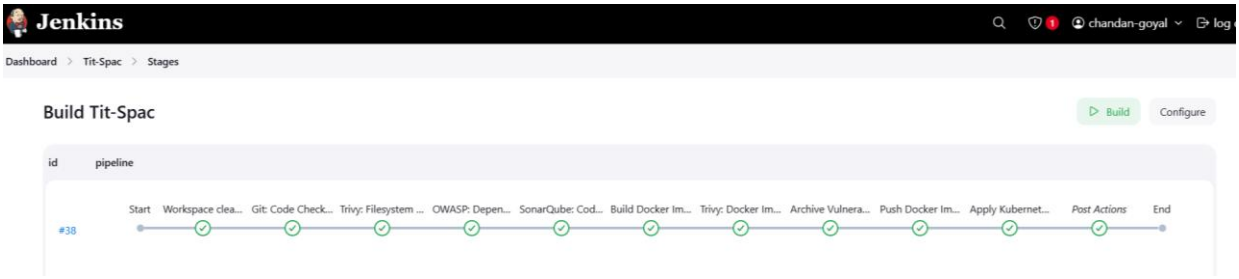

Fig 7.5 Jenkins Pipeline Stages

The image shows a detailed view of the "Tit-Spec" Jenkins pipeline at stage #38. The pipeline visualization displays a series of stages with green indicators showing successful completion. The current focus is on the "Apply Kubernetes Manifests" stage, showing logs with execution details and command outputs with timestamps.

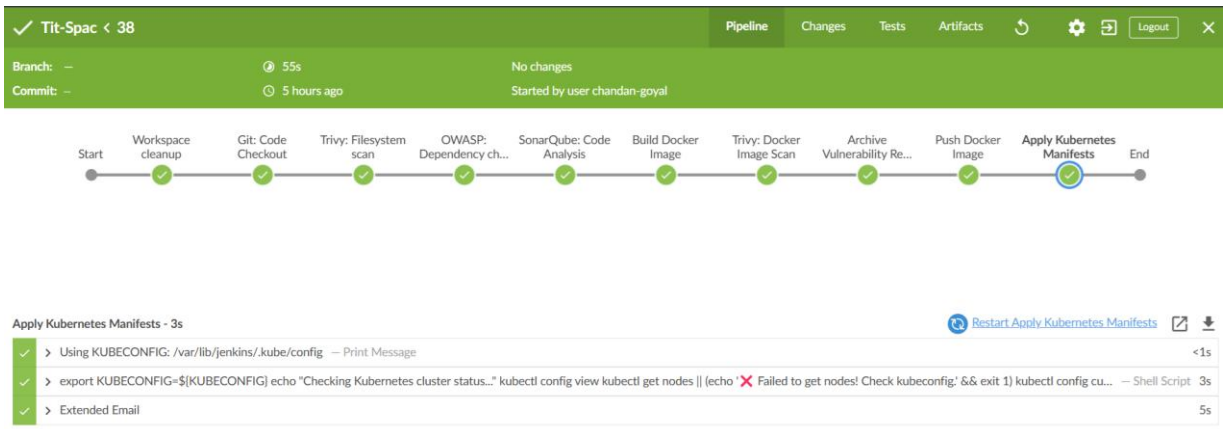

Fig 7.6 Jenkins Pipeline

The image shows a SonarQube dashboard displaying quality metrics for two projects. The interface presents key code quality indicators including bugs, vulnerabilities, code smells, coverage, and duplication for projects named "Titanic-Spaceship" and "TitanicOps". Each metric is visualized with color-coded circles (green, red) to indicate status, alongside percentage values for various quality measures.

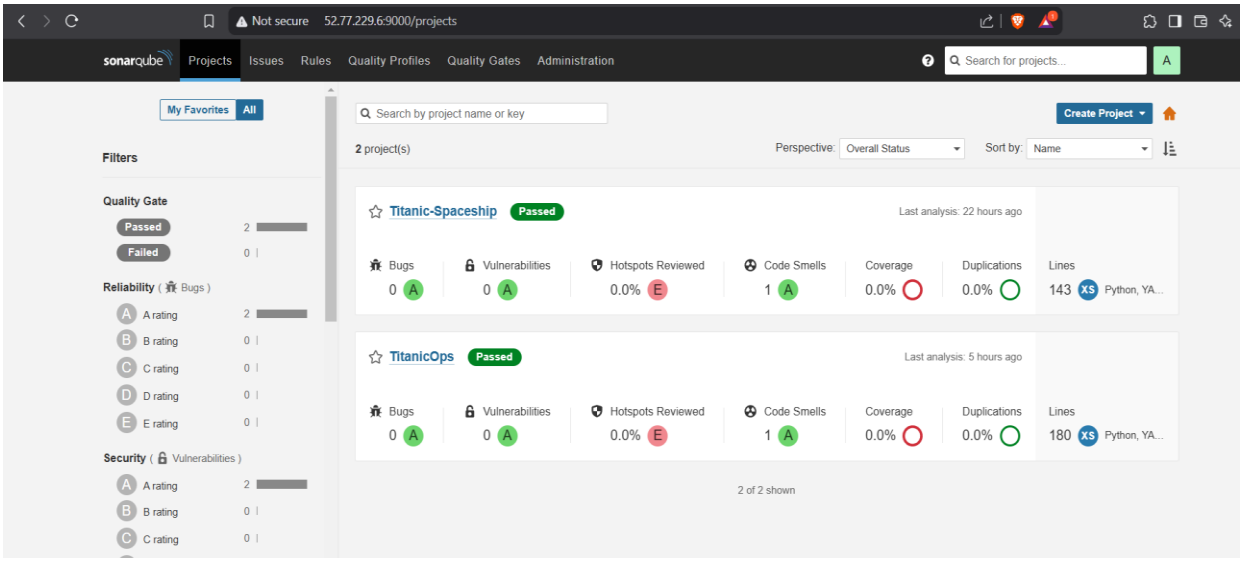

Fig 7.7 Sonarqube Dashboard

The image shows a Docker Hub interface displaying repositories for the "cgoyaldeveloper" namespace. The page shows a repository named "cgoyaldeveloper/titanic-spaceship" which was last pushed about 5 hours ago. The interface includes navigation options, search functionality, and displays repository details such as contents, visibility status (Public), and size. The Docker Hub interface also provides a download count, allowing users to see how many times the "cgoyaldeveloper/titanic-spaceship" image has been pulled.

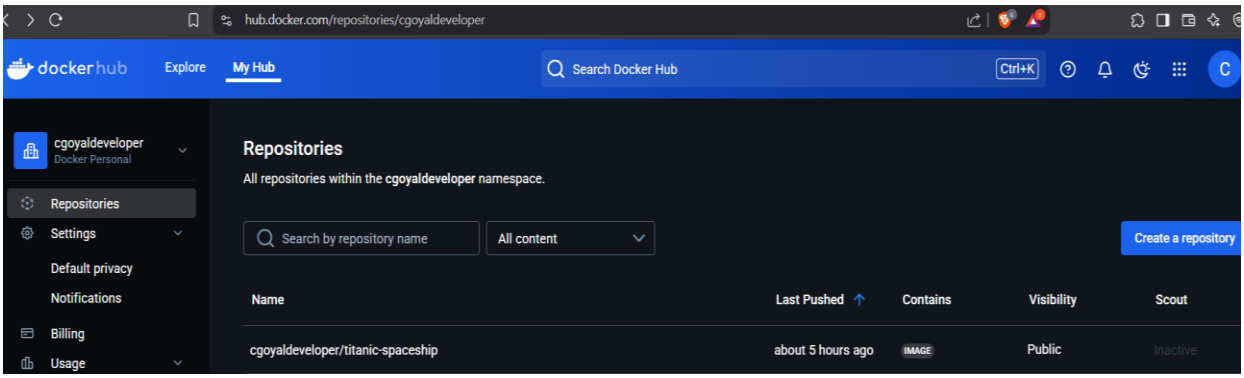

Fig 7.8 Docker Hub Repositories

The image shows a Gmail interface displaying an email with the subject "Titanic-Ops Application build failed - 'FAILURE'". The email is from smartchandan141@gmail.com regarding Project "Tit-Spec" Build Number 5. The message contains color-coded information

sections (red and green) and has one attachment - a build log file. The email appears to be an automated notification from a CI/CD system about a failed build process.

Fig 7.9 Build Failure Email

The image shows a Gmail interface displaying an email with the subject "Titanic-Ops Application has been updated and deployed - 'SUCCESS'". The email is from smartchandan141@gmail.com and includes color-coded information sections showing project and build details. It has one attachment - a build log file. The email appears to be an automated notification from a CI/CD system confirming a successful application deployment.

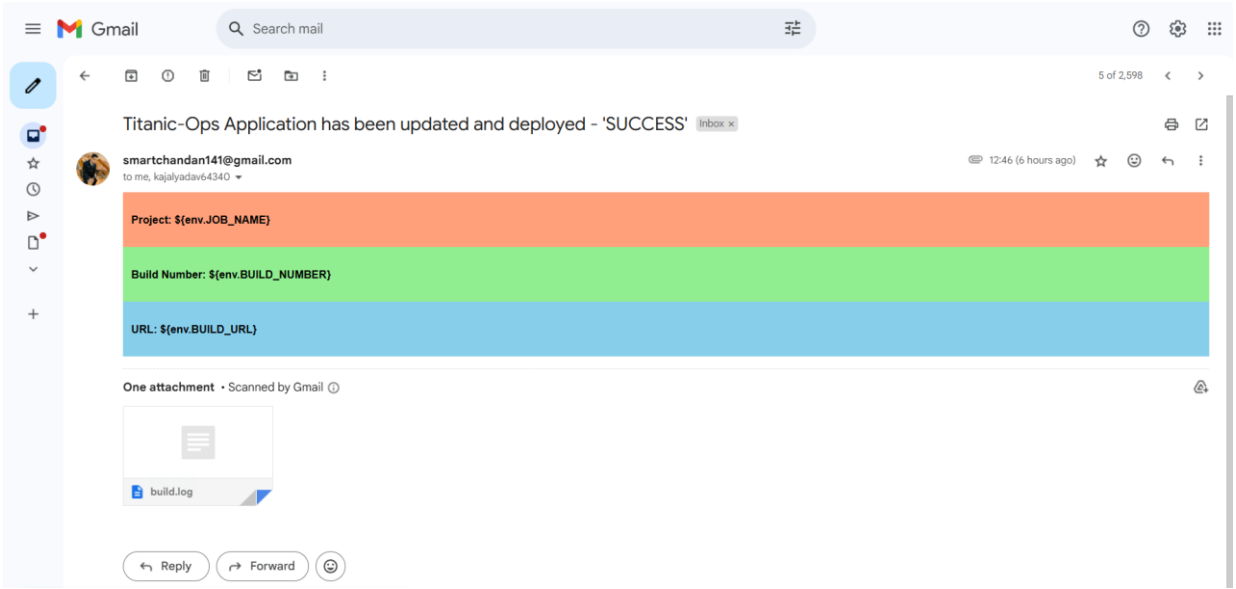

Fig 7.10 Deployment Success Email

The image displays the Argo CD web interface showing an "elastic-app" application details. It includes project information, Git repository URL, revision settings, and timestamps for creation and last sync, along with action buttons for managing the Kubernetes application deployment. Additionally, it offers options to trigger manual syncs, rollback to previous revisions, and configure automatic sync policies for streamlined management.

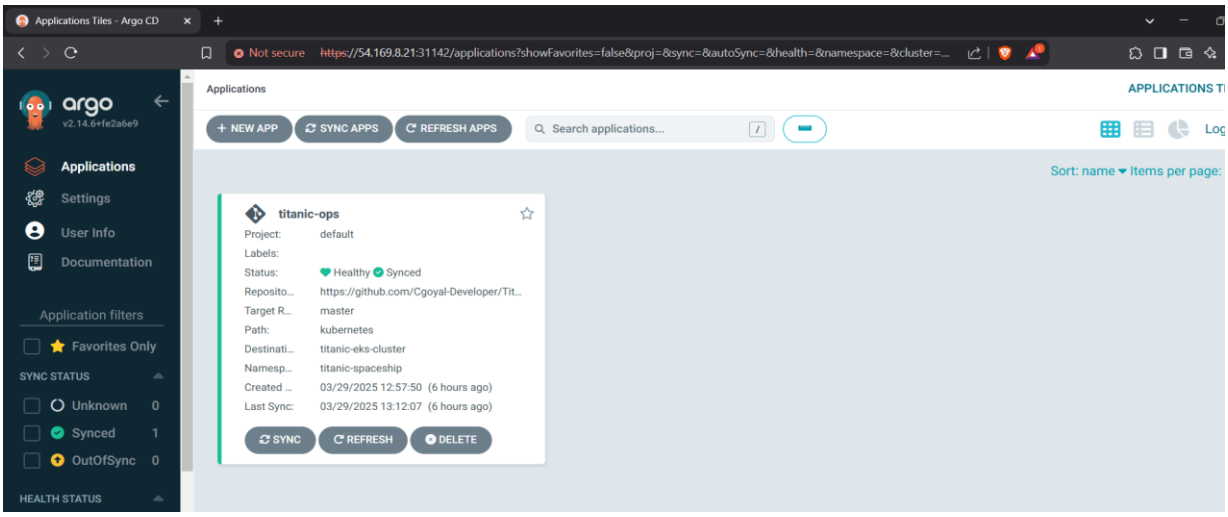

Fig 7.11 Argo CD Dashbaord

The network visualization highlights the dependencies between various Kubernetes resources, such as pods, services, and deployments, making it easier to understand the application's architecture. Additionally, the interface provides real-time monitoring of the deployment process, with status indicators for each resource, ensuring a smooth and efficient rollout.

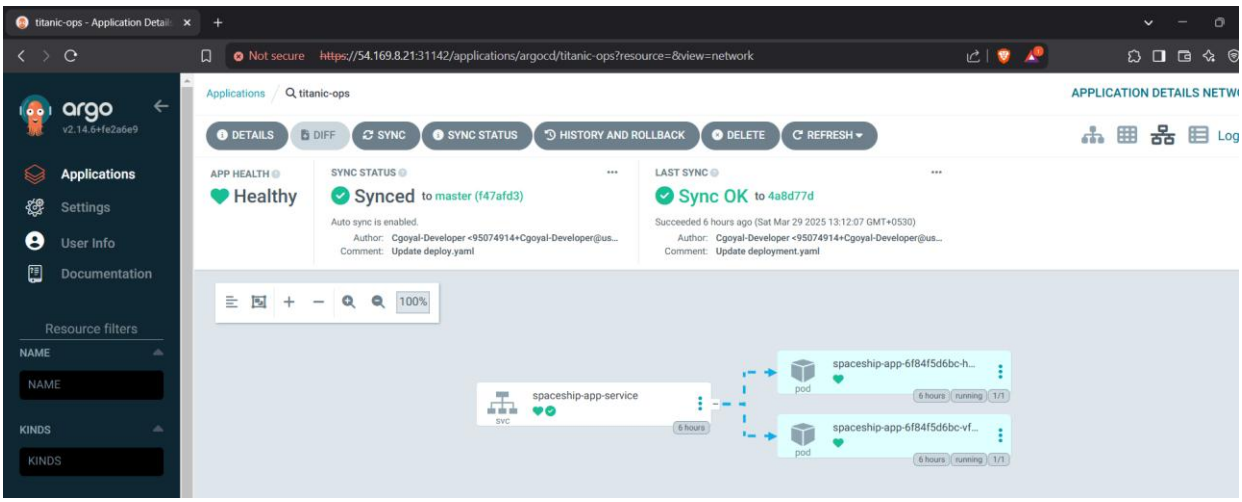

Fig 7.12 Argo CD Status View

This image displays a dark-themed web interface for the "Spaceship Titanic Competition." It includes a sidebar with a dropdown navigation menu and a main section with a bold title, a brief description about predicting passenger survival, and an image of a spaceship near Earth.

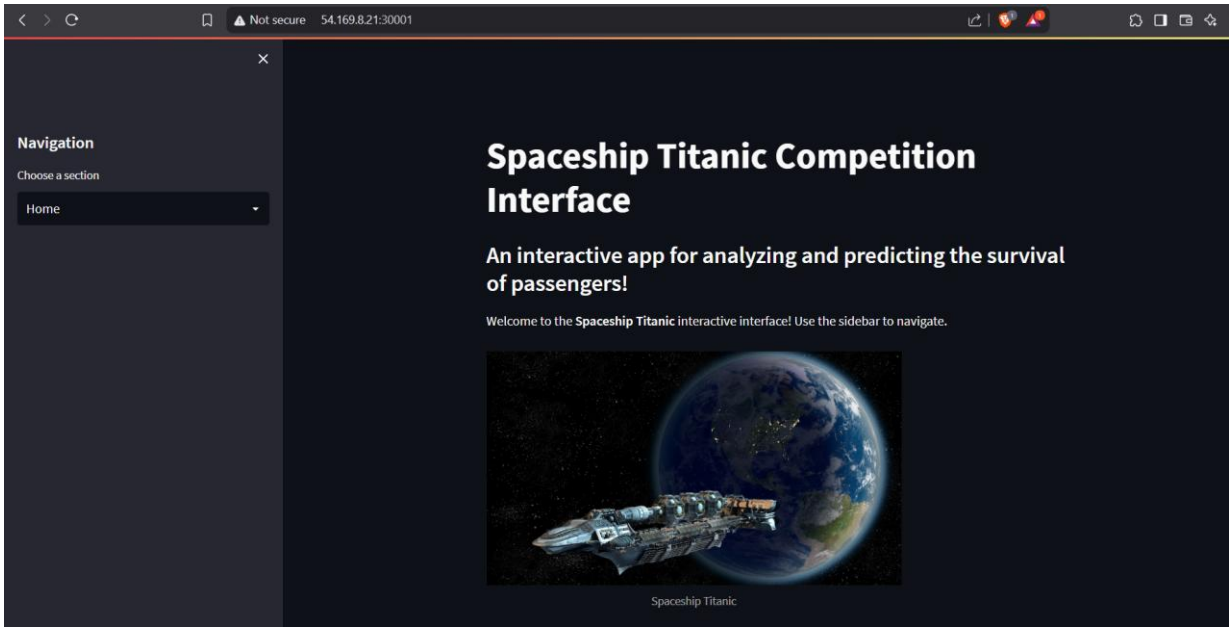

Fig 7.13 Spaceship Titanic Competition Interface

The image is a screenshot of a web-based application titled "Spaceship Titanic Competition Interface." It has a dark-themed design with a sidebar for navigation and a main content area. The sidebar includes a dropdown menu labeled "Choose a section," with "Home" selected, along with options like "Upload Data," "Train Model," and "Make Predictions."

Fig 7.14 Home Page

The image shows the Spaceship Titanic Competition interface for predicting passenger survival in space. It displays uploaded train and test CSV files with their sizes. The interface includes a data preview showing columns like PassengerId, HomePlanet, CryoSleep, and Destination.

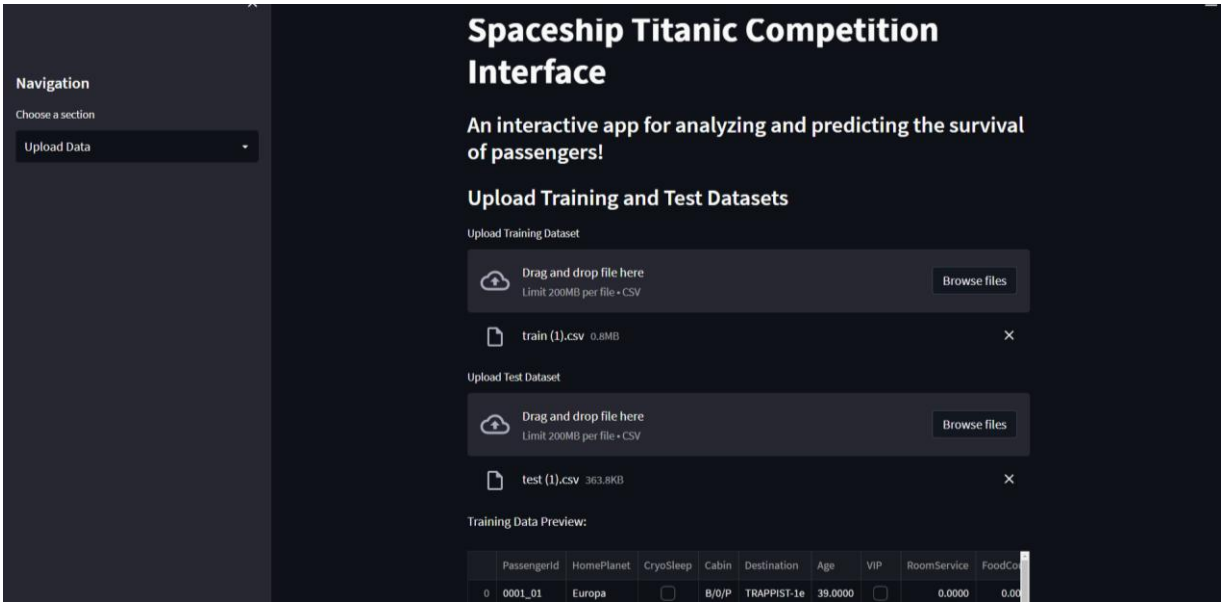

Fig 7.15 Upload Data

The image shows the "Spaceship Titanic Competition Interface" with "Train Model" selected. The main area displays "Train Your Model" with the uploaded "train (1).csv" file (0.8MB). A data preview shows 5 rows of passenger information including columns for PassengerId, HomePlanet, CryoSleep, Cabin, Destination, Age, VIP and service expenses.

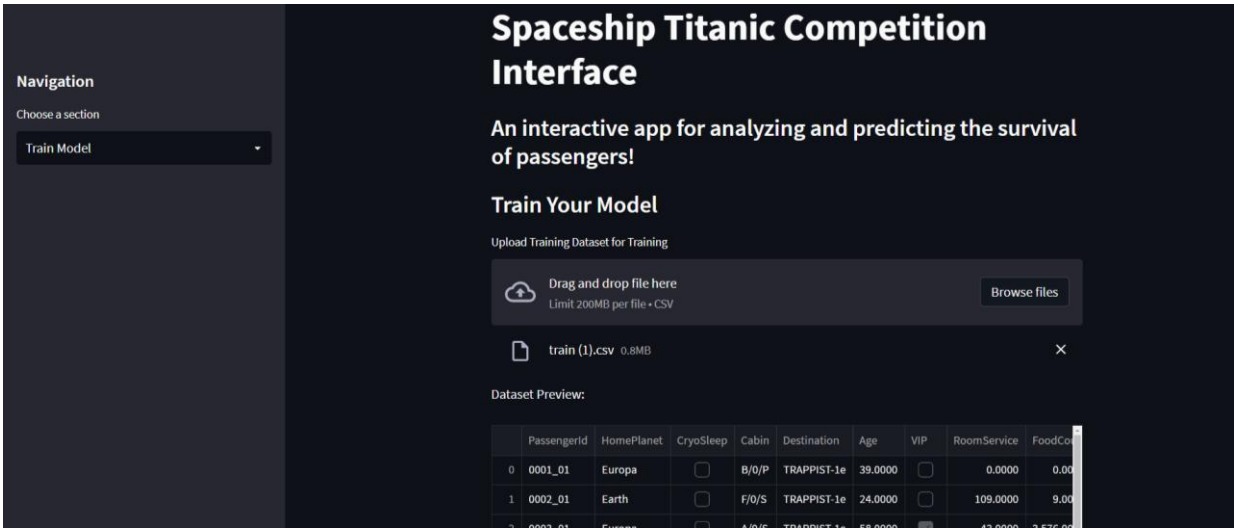

Fig 7.16 Train Model

The image shows the "Spaceship Titanic Competition Interface" in the "Make Predictions" section. The left panel has "Make Predictions" selected, while the main area displays "Make Predictions with Your Model" with an uploaded "test (1).csv" file (363.8KB). Below is a test data preview showing 5 rows with columns like PassengerId, HomePlanet, CryoSleep, Cabin, Destination, Age, VIP, and service expenses. The passengers are from Earth and Europa with ages between 19-38. This interface allows users to apply their trained model to predict survival outcomes.

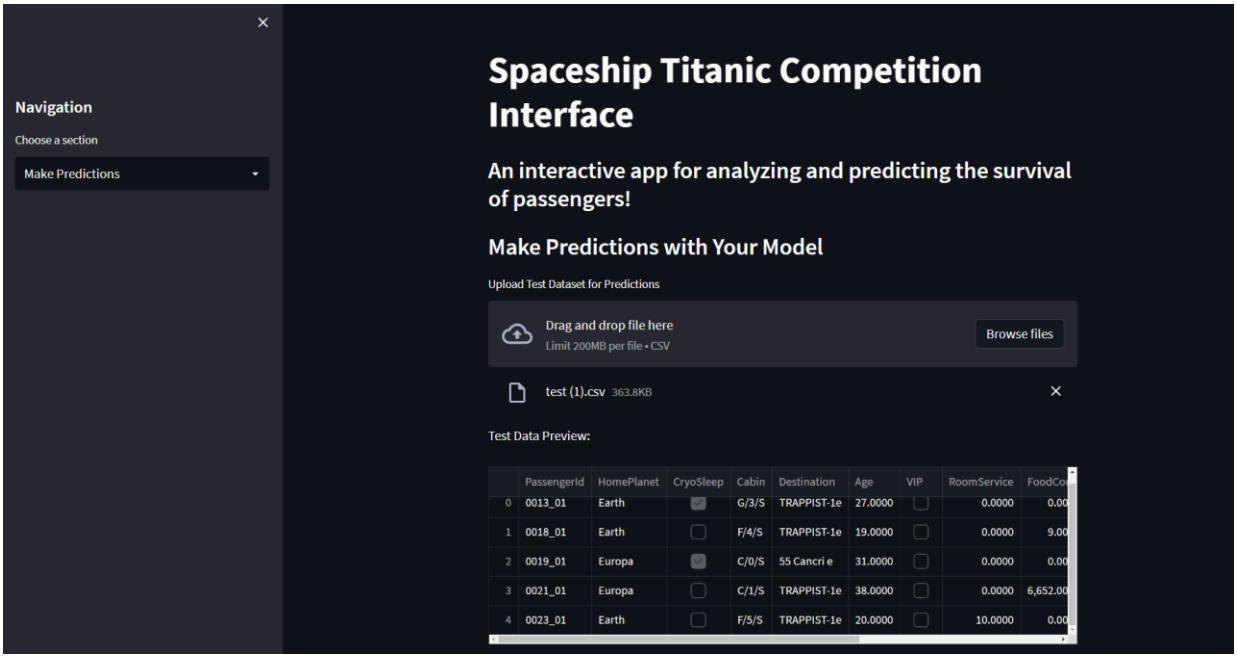

Fig 7.17 Model Predictions

This image shows the "Make Predictions" section of the Spaceship Titanic Competition Interface. The top portion displays test dataset rows with passenger information (IDs, planets, cabin details, ages, etc.). Below is a "Predictions:" table showing model results with "PassengerId" and "Transported" columns. The Transported column shows checkboxes indicating predicted outcomes - some passengers (like 0019_01, 0021_01) are marked as transported while others (0018_01, 0023_01) are not. A "Download Predictions" button appears at the bottom, and the left navigation panel shows "Make Predictions" as the active section.

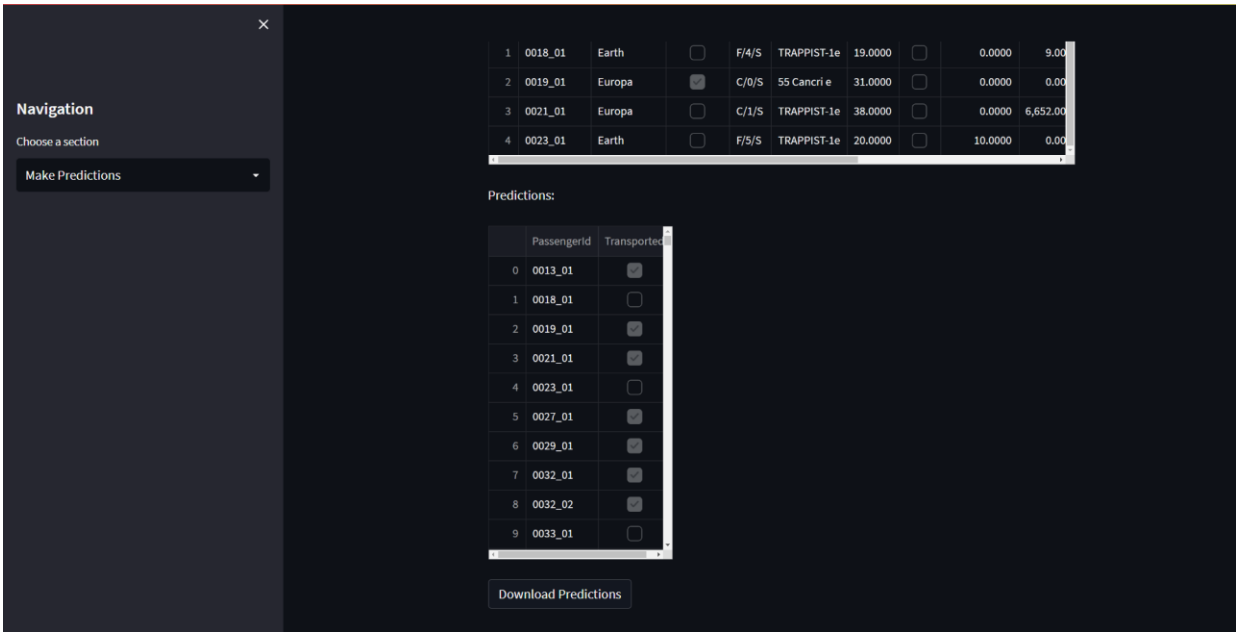

Fig 7.18 Final Predictions

The image shows a Grafana dashboard displaying Kubernetes networking metrics. Two gauge visualizations show current network traffic rates: 2.10 KB/s received and 2.22 KB/s transmitted for the kube-system namespace. Below is a table showing detailed network usage metrics for various pods including current/transmit bandwidth rates and packet statistics.

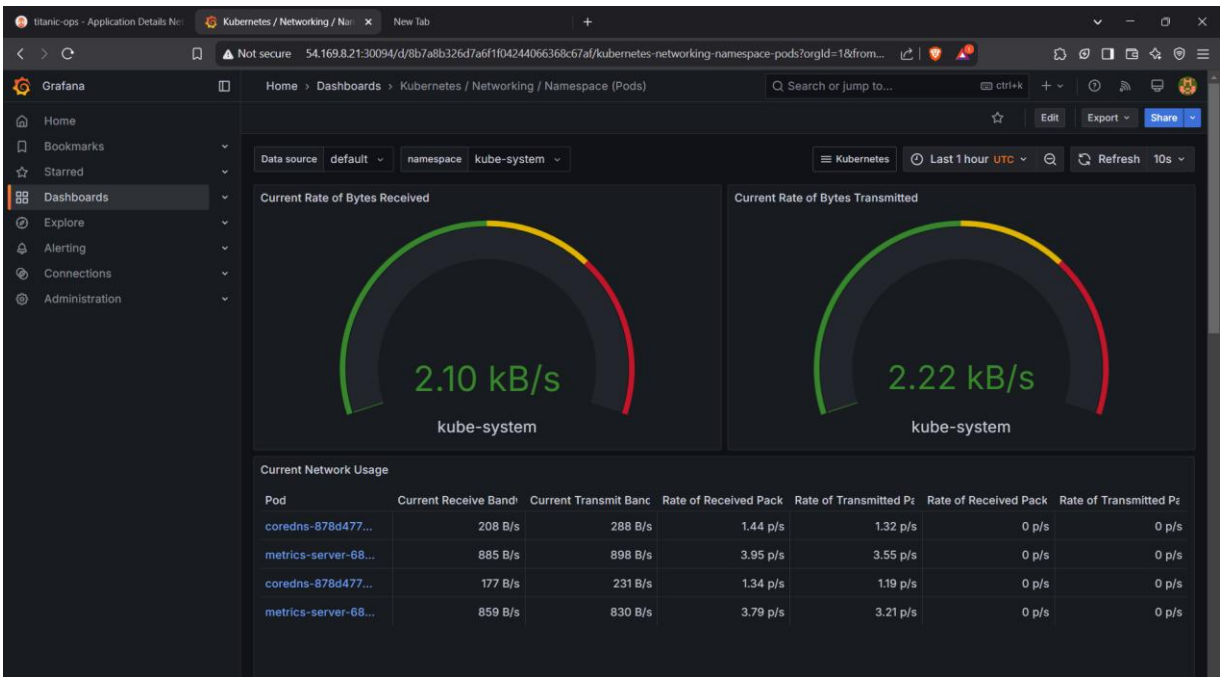

Fig 7.19 Grafana Kubernetes Monitoring

The image shows a Grafana dashboard with time-series graphs tracking network traffic (received and transmitted bytes) across multiple Kubernetes namespaces.

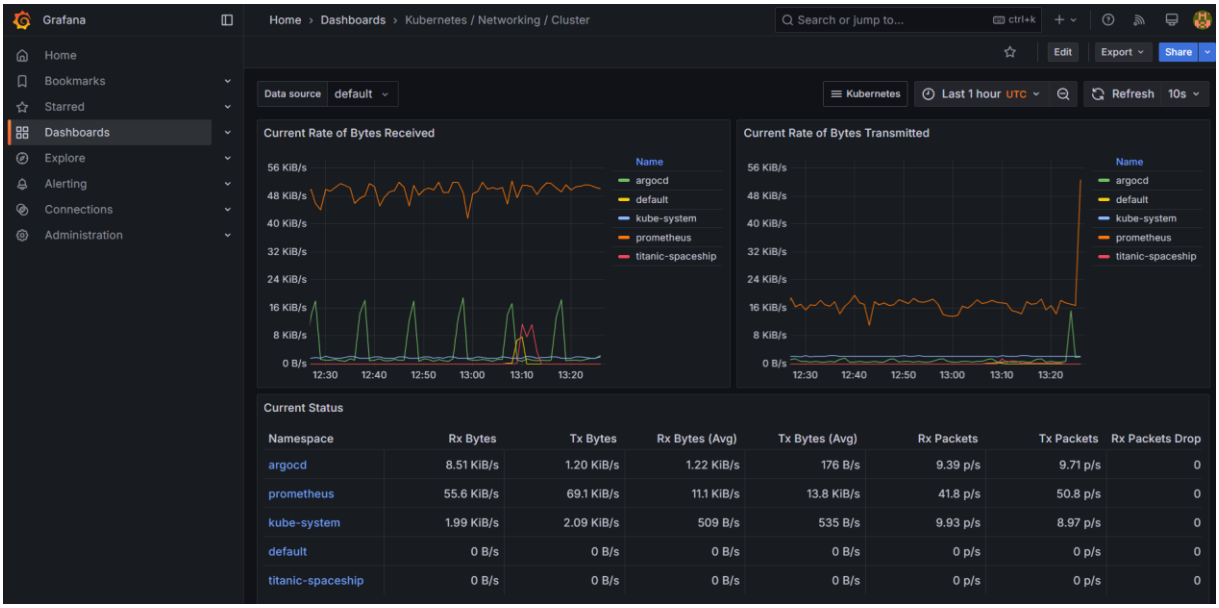

Fig 7.20 Grafana Network Metrics

The image shows a leaderboard of submissions for a machine learning competition, likely hosted on Kaggle. It displays two submissions, both marked as "Complete" and successfully processed. Each submission has a corresponding public score, representing the model's performance. The latest submission, "Spaceship_Submission.csv," achieved a higher score of 0.81201, while the previous submission, "spaceship_prediction_project (1).csv," had a score of 0.80804. The results indicate an improvement in model accuracy. The ranking is sorted by recent submissions, and all results appear successful without errors.

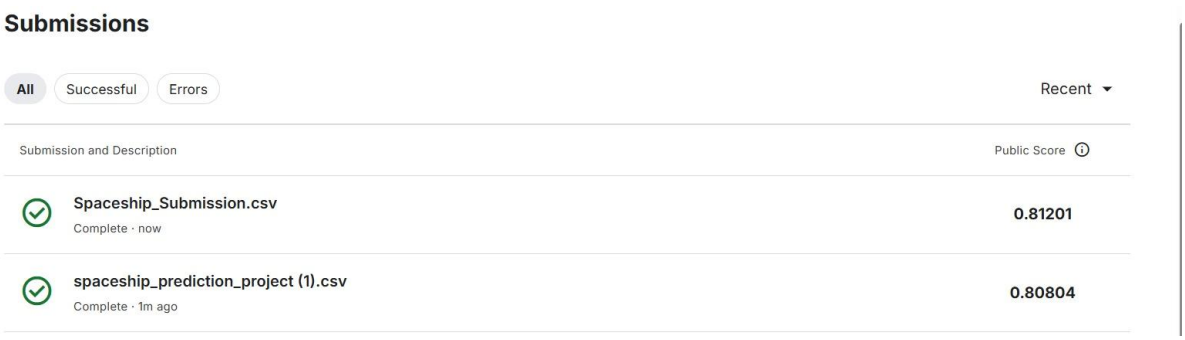

Fig 7.21 Submission Results

Chapter 8 – References

[1] Kaggle, "Machine learning competitions," Kaggle, 2024. Available: https://www.kaggle.com.

[2] M. Sculley et al., "Hidden technical debt in machine learning systems," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst. (NIPS)*, Montreal, Canada, 2015, pp. 2503–2511.

[3] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Boston, MA, USA: Addison-Wesley, 2010.

[4] X. Luo et al., "Kubernetes-based containerized machine learning models: An overview," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, Nicosia, Cyprus, 2018, pp. 324–331.

[5] Amazon Web Services (AWS), "AWS: Cloud computing services," AWS, 2024. Available: https://aws.amazon.com.

[6] Prometheus, "Prometheus Monitoring System," 2024. Available: https://prometheus.io/docs/introduction/overview/.