Continuous Integration and Deployment for Playlist Rules Generator

Cloud Computing Class Project

January 14, 2025

Contents

1 Introduction

This project involves the development and deployment of a playlist recommendation system using association rules. The system generates frequent itemsets from datasets, formulates association rules, and exposes these rules through a REST API. The entire system leverages Kubernetes for deployment and ArgoCD for continuous integration and delivery (CI/CD).

This report discusses the test cases performed using Kubernetes and ArgoCD, the performance of updates to the system, the mechanism for detecting model changes in the REST API, and how the system incorporates new datasets when regenerating the model.

2 Test Cases and Results

2.1 Test Case 1: Kubernetes Deployment and Scaling

The system was deployed on a Kubernetes cluster with the following configuration:

- REST API: 3 replicas
- Model generator: 1 replica

The deployment was tested for:

1. **Initial Deployment**: The initial deployment completed successfully, with all pods reaching the "Running" state in under 1 minute. No downtime was observed during the deployment.
2. **Scaling**: Increasing the number of replicas for the REST API from 3 to 5 was seamless, with new pods starting within 20 seconds. The load balancer handled traffic distribution effectively.
3. **Rollback**: A rollback to a previous deployment version was tested and completed within 30 seconds, restoring the system to a stable state.

2.2 Test Case 2: ArgoCD Continuous Integration

ArgoCD was used to manage the system's continuous integration workflow. The following scenarios were tested:

1. **Code Changes**: Updating the Flask API code triggered an automatic synchronization via ArgoCD. The new version was deployed in under 1 minute with no downtime.
2. **Deployment Changes**: Modifying the number of Kubernetes replicas triggered a seamless synchronization. Pods scaled up or down in less than 30 seconds.
3. **Dataset Updates**: Uploading a new dataset and regenerating the model was completed within 2 minutes. The updated rules were automatically reflected in the REST API.

3 Time for Updates and System Availability

The system was evaluated for the time taken to implement various updates and its impact on availability. Results are summarized below:

3.1 Code Updates (Model or API)

- Average deployment time: 1 minute
- Availability: No downtime observed, as Kubernetes rolling updates ensured seamless transitions.

3.2 Deployment Updates (Kubernetes Replicas)

- Average update time: 20–30 seconds
- Availability: No downtime observed, as the load balancer managed traffic to active replicas.

3.3 Dataset Updates

- Average update time: 2 minutes
- Availability: The REST API remained online, serving existing rules while the model generator container processed the new dataset.

4 Detecting Model Changes in the REST API

The REST API detects model changes by monitoring updates to the serialized rules file (`rules.pkl`). This mechanism works as follows:

1. The model generator container regenerates association rules and overwrites the `rules.pkl` file.
2. A file system watch mechanism or periodic polling in the REST API detects changes to the file's timestamp.
3. When a change is detected, the API reloads the updated rules into memory, making them immediately available to clients.

5 Incorporating New Datasets for Model Regeneration

The system incorporates new datasets using the following process:

1. A new dataset is uploaded to a designated storage location or passed as input during a new Docker build.
2. The model generator container reads the updated dataset and executes the Frequent Itemset Mining algorithm to generate new association rules.

3. The new rules are serialized and saved to the `rules.pkl` file.

4. The REST API detects the updated rules file and loads the new rules automatically, ensuring that the system reflects the latest dataset without requiring downtime.

6 Conclusion

The project successfully demonstrated the use of Kubernetes and ArgoCD for deploying and managing a playlist recommendation system based on association rules. The tests revealed that the system supports seamless updates with no downtime and can efficiently regenerate models using updated datasets. The CI/CD pipeline facilitated by ArgoCD ensures rapid and reliable deployment of changes, making the system robust and scalable.