CMPT 756 Kube Squad/Square Term Project Report

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Team Name** | Kube Squad/Square |  |  |  |  |
| **GitHub project repo URL** | https://github.com/scp756-221/term-project-kube-squad/tree/music-service-port |  |  |  |  |
| **Team Member Name** | Aidan Vickars | Rishabh Kaushal | Anant Awasthy | Karthik Srinatha | Abhishek Nair |
| **Email (SFU)** | avickars@sfu.ca | rka73@sfu.ca | asa404@sfu.ca | ksa166@sfu.ca | asn4@sfu.ca |
| **Github ID** | avickars | rishabhkaushal07 | asa-404 | Karthiksrinatha | abs990 |
| **Additional Notes** | Currently the development branch we are using is "music-service-port". We will be changing to "main" once the project is complete. | | | | |

# Summary of Application

Due to the large number of members in our group, and general interest we developed three original micro-services that work together to form a playlist application with corresponding authentication and subscription services. The application is visualized below in Figure 1.

Graphical user interface, diagram

Description automatically generatedFigure 1: Playlist Application

The architecture of the application has been split into three separate micro-services that run on Kubernetes. Beginning with the Authentication service. This service allows users to create or login into their account respectively and subsequently access the Playlist microservice that will be described shortly. Like any normal application, accessing features often requires a paid subscription. Our application is no different, accessing the features in the playlist micro-service requires a paid subscription. We have implemented this in the Subscription service, where users after logging in can subscribe by adding a credit card that passes through a simulated validation. Once users have logged in and subscribed to the service, they can access the Playlist service that contains a variety of features. In this micro-service users can create, view, and edit playlists, as well as find information related to specific songs such as the lyrics, genre, and artist. All three services leverage Dynamo DB and independently query Dynamo DB to access information like users, song information, subscription status etc. Finally, from the user perspective, we have modified the given MCLI application to make HTTP requests to each micro-service. The MCLI application continues to run as a command line application.

# GitHub Guide

Beginning with the structure of the GitHub repository, the central branch is “[music-service-port](https://github.com/scp756-221/term-project-kube-squad/tree/music-service-port)” that contains the most recent working build of the application. It is expected that any team member that merges their work into this branch, verifies that the branch remains in an error free condition. It should be noted that once the application is completed, the “[music-service-port](https://github.com/scp756-221/term-project-kube-squad/tree/music-service-port)” branch will be merged into the [“main”](https://github.com/scp756-221/term-project-kube-squad/tree/main) branch. Furthermore, we utilize branching extensively during the development process. When a developer is actively working on a feature or bug, they create a new branch to use as their working copy and merge the branch into the “[music-service-port](https://github.com/scp756-221/term-project-kube-squad/tree/music-service-port)” branch once their work is complete. This is reflected in at the time of writing, there are 15 separate branches within the repository.

To augment our development, we make extensive use of GitHub’s Project features and log every item of work as an issue in the repository. The project we use to visualize the to be completed, in progress and completed work is the [“756 Project”](https://github.com/scp756-221/term-project-kube-squad/projects/2). In this project, whenever a user embarks on a new piece of work, they create the corresponding tickets and place them in the “To do” column. When the issue is actively being worked on, it is moved into the “In progress” column and finally once the issue is completed it is moved to the “Done” column. It should be noted, we do attempt to link our commits to the relevant issues, however we have found that remembering to do this a hit or miss.

# Observations

## Reflection on Development

**What did you observe from applying and using the scrum methodology? What worked well? What didn’t? What surprised you?**

After applying and using scrum methodology, we made three crucial observations. The first is that it allowed us to assess more accurately our currently velocity towards completing the project. Due to the nature of course work that often comes in waves, team members development capacity often significantly fluctuated week to week. By having weekly scrums, we were able to properly define what the definition of done for the upcoming sprint such that it was achievable. Furthermore, in the case a team members capacity changed on short notice (ex: assignment was pushed back), having weekly sprints allowed us to pivot quickly.

The second observation we made was that the consistent cadence that scrum often requires did not work. As mentioned above, team members development capacity would significantly fluctuate week to week. As a result, there were sprints where the entire team’s development capacity was effectively zero and the resulting sprint goal was nothing. Thus, the team was unable to maintain a consistent cadence that at times defeated the purpose of scrum because there was simply no point in having one since no one had capacity to perform any work.

The third and final observation we made was that it normalized the behaviour of demonstrating one’s work to the team. By applying scrum, it normalized the expectation of demonstrating the completed items from the last sprint. We found this to be particularly useful because it served as both an opportunity for everyone to demonstrate their work, but also as an opportunity to teach others how to utilize the completed work. An example of this was the “dockerization” of the application where each service was ported to work as deployable containers. After demonstrating the applications running as containers, each team member became familiar with the process of running the application as a series of containers.

**Reflect on the readings over the course of the term. What ideas were you able to apply? How did these turn out?**

Beginning with the second reading that featured a comparison between mono and poly repositories, we applied a mono-repository to great success. Due to the size of the application, using a poly-repository would have created additional overhead in managing the location of each micro-service. In contrast, by using a mono-repository we were able to easily create a single source of truth that contained the application that facilitated rapid deployment and iteration. This, rapid deployment was further enhanced by the “infrastructure as code” ideas that were also presented in the second reading. To implement this, we wrote a comprehensive make file that contains every command required to deploy the application from in both a local and cloud format. This enabled a consistent deployment for every team member that eliminated all confusion around the deployment. It should be noted, that our use of containers and make files to facilitate deployment captures ideas from the third reading as well that discusses containerization and make files.

TO BE DISCUSSED – KUBENETES IMPLEMENTATION – Reading 4 – The cloud based deployment is currently in progress. We are waiting to see how Kubernetes works under load with gatling before adding this section.

# Analysis