CS695 Assignment 4 Report

Team: Yaml Yoda

Members: Soumik Dutta(23m0826), Sm Arif Ali(23m0822)

Background:

Traditional cloud computing requires developers to provision and manage virtual machines or containers to run their applications. This approach involves concerns such as scaling, server maintenance, and resource optimization. However, as applications become more complex and demand more agility and scalability, managing these resources can be cumbersome and costly.

Serverless computing addresses these challenges by abstracting away the infrastructure management. Instead of provisioning servers or containers, developers write code in the form of functions, which are small, event-triggered pieces of code that execute specific tasks. Cloud providers manage the execution environment, automatically scaling resources up or down based on demand.

Motivation:

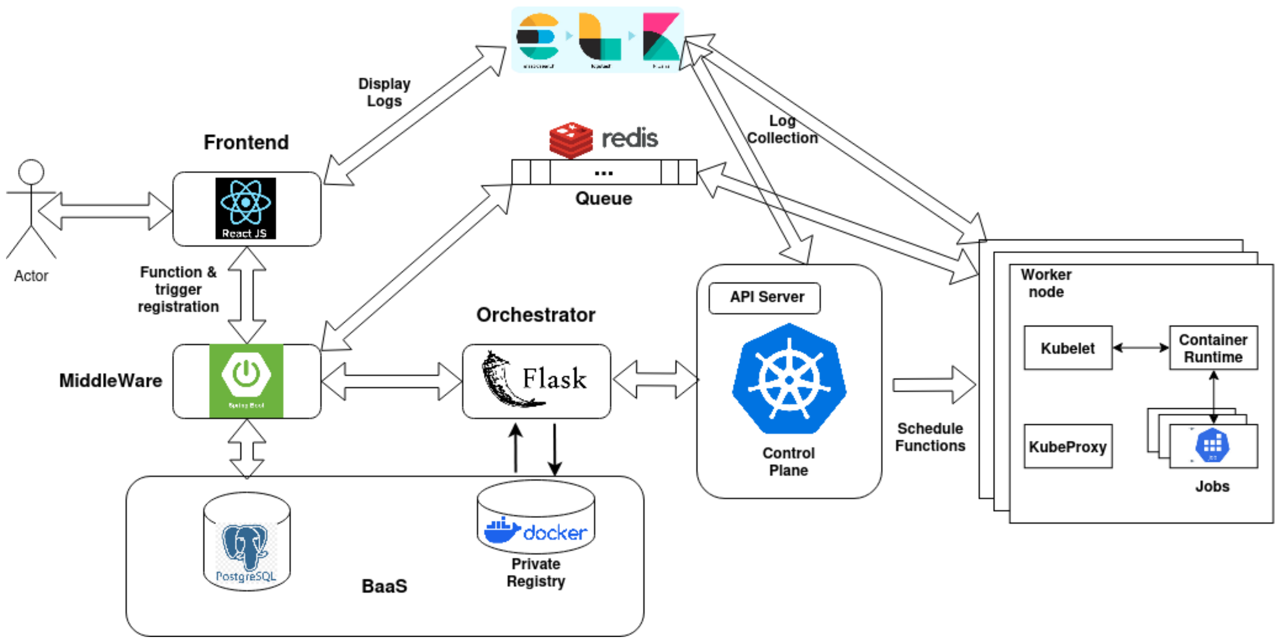
The motivation for serverless platforms and the development of Function as a Service (FaaS) stems from several key factors:

1. **Simplified Infrastructure Management:** Eliminate the need for developers to manage servers or containers.
2. **Cost Efficiency:** Pay only for the resources used during function execution, leading to significant cost savings.
3. **Scalability and Elasticity:** Automatically scale resources up or down based on demand, ensuring high performance under varying loads.
4. **Faster Time to Market:** Abstracting away infrastructure concerns allows developers to focus on writing code and delivering features quickly.

5. **Event-Driven Architecture:** Enable highly responsive applications where functions are triggered by events such as HTTP requests or database changes.

Here in this assignment we have built a basic Faas platform running on kubernetes cluster.

Design:



Features:

Our platform offers the following features:

1. Function and trigger registration via Web UI
2. Integrated IDE in web app for writing code.
3. Ability to list dependencies (requirements.txt).
4. Functions can be triggered via image upload to preferred S4 bucket.
5. Supports concurrent function triggers.
6. All CRUD operations on S4 bucket.
7. Ability to check logs of functions via kibana dashboard.
8. Basic autoscaling of jobs based on current work queue size.

Experimental Evaluation:

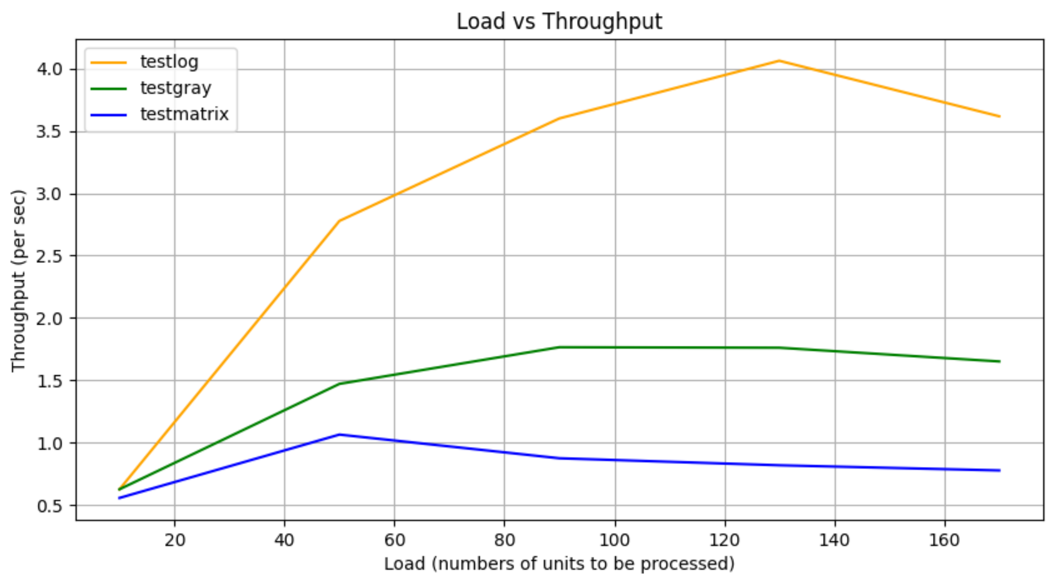
General Setup: In all of our experiments, we ran the flask server and spring boot middleware in separate laptops connected via LAN. Our cluster consists of one control plane & one worker node setup using kubeadm, running Linux ubuntu with 8GB of RAM and i5 11th 2.4GHz processor connected via LAN. All functions run were in python.

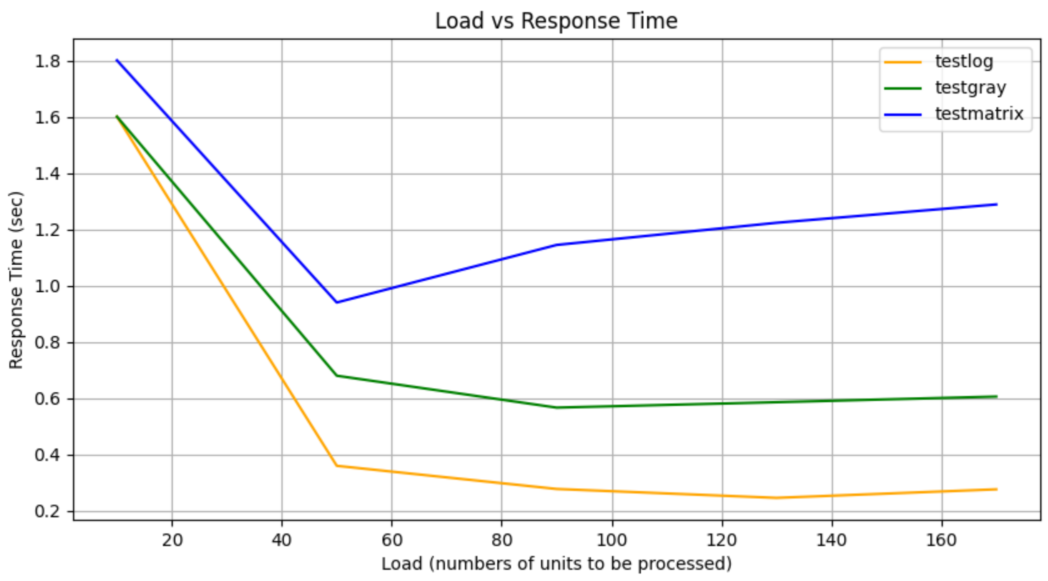
1. **Question:** Does the built platform able to register any user function and execute them on trigger event by resolving their dependencies?

Answer: We ran a simple image processing function using pillow that converts color image to grayscale. On registering such function and by triggering it we were able to see the output grayscale images in the bucket.

2. **Question:** How our Faas application is behaving in face of different kinds of user defined functions: light weight, IO heavy and CPU heavy while increasing load?

Setup: A light weight logging function, an IO heavy image processing function & a CPU heavy matrix multiplication is used as test functions. Max CPU : 8, Max RAM: 2048Mb Pods: 1.





Inference: Here we see saturation after load 60.

Challenges:

The main challenge faced by us were:

1. Complexity in coordinating multiple functions to divide a large batch job.
2. High latency for pushing built function image to docker hub.
3. Contention for database connection.

4. Implementing an asynchronous model of function execution.
5. Implementing a distributed architecture where cluster, flask backend, spring boot middleware all run in different machines.

Future Scope:

There are several future scopes for this project such as:

1. **Wide language Support:** Currently we only experimented with running function in python code, it could be extended to support more languages like Java, C++, Go, NodeJS etc
2. **Database Abstraction:** Currently we run a boilerplate code for each user function that connects with the backend image data store and feeds the data to userfunctions. Instead of this, we could build expose simple API via libraries that users can use to explicitly access the uploaded data.
3. **Resolving Dependency:** We are prebuilding a base image with all our basic libraries and adding user code on top of that, this saves time during function registration but isn't flexible enough to serve richer function domains. Methods can be implemented to detect the best image on the fly based on user code and given dependencies.
4. **Autoscaling:** As of now, kubernetes doesn't support automatic scaling of Job resources based on some metrics. An intelligent autoscaler can be built that checks the current load level & perhaps resource utilization to take a decision for horizontal scaling.