Assignment 4

KubeFunction

Virtualization and Cloud Computing

by

Kumar Ankur and Shrayansh
(Roll No. 23M0829 and 23M0778)
Team Name : faas8

Department of Computer Science and Engineering
Indian Institute of Technology Bombay

Introduction

Kubernetes can be used to manage a cluster of nodes to instantiate container-based applications. As part of this work, we are building FaaS platform as a wrapper around K8s. This FaaS platform supports function registration, trigger registration, trigger dispatch, metrics and more. This report provides an outline of a Function-as-a-Service (FaaS) platform that uses Kubernetes (K8s) for managing many functions as pods, docker for containerization, and Redis Queue with Celery for multiple task execution asynchronously. This platform allows users to submit Python-based functions through a user interface (UI), map triggers to functions, and then dispatch triggers for execution in a scalable environment. This report describes the platform architecture, function execution flow, task scheduling, and performance.

Platform Architecture

The platform architecture consists of the following major components:

1. User Interface (UI)

A frontend interface where users can submit their Python function files, specify a trigger name, and create triggers to execute functions.

2. Docker Environment

The Python function submitted by the user is given to the Dockerfile as a dependency, which creates a Docker image. This image is stored in a Docker repository.

3. Kubernetes Cluster

Minikube is used for creating Kubernetes cluster. The cluster manages nodes for deploying pods to execute the Docker image.

4. Redis Queue and Celery

Redis stores queuing function execution tasks and Celery distributes tasks across worker nodes for parallel execution.

Function Execution Flow

The following steps describes the process from function registration to execution and result retrieval:

1. Function Registration

The user uploads a Python function file through the UI, providing a function name and trigger name.

2. Trigger Registration

The trigger is mapped to the function, allowing the user to dispatch the function using the trigger.

3. Docker Image Creation

The function file is included in a Dockerfile with image of "python". The Dockerfile is used to create a Docker image.

4. Docker Image Storage

The Docker image is uploaded to a Docker repository (dockerhub) for later retrieval.

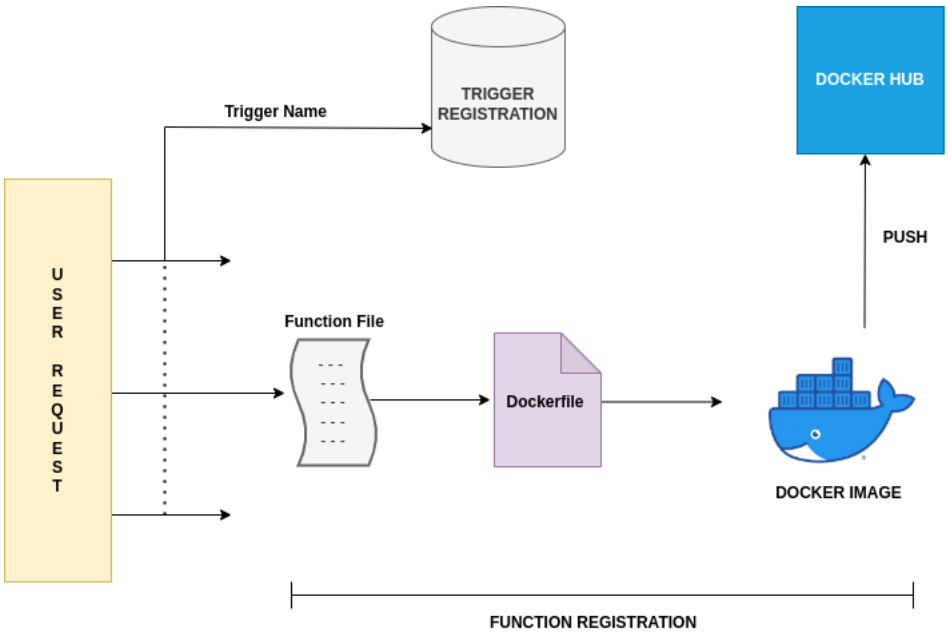

Figure 1: Function and Trigger registration

5. Trigger Dispatch

When the user triggers a dispatch, the corresponding Docker image is pulled from the Docker repository.

6. Task Execution with Celery

Redis Queue receives the dispatch request. A task id is also created corresponding to that task. After that, Celery assigns the task stored in Redis Queue to an available worker process. Each worker process can execute one task at a time. the output of worker process is stored in the Redis Queue database corresponding to its task id. User can check the output using this task id.

7. Pod Deployment in Kubernetes

The Docker container runs inside a pod within a Kubernetes cluster.

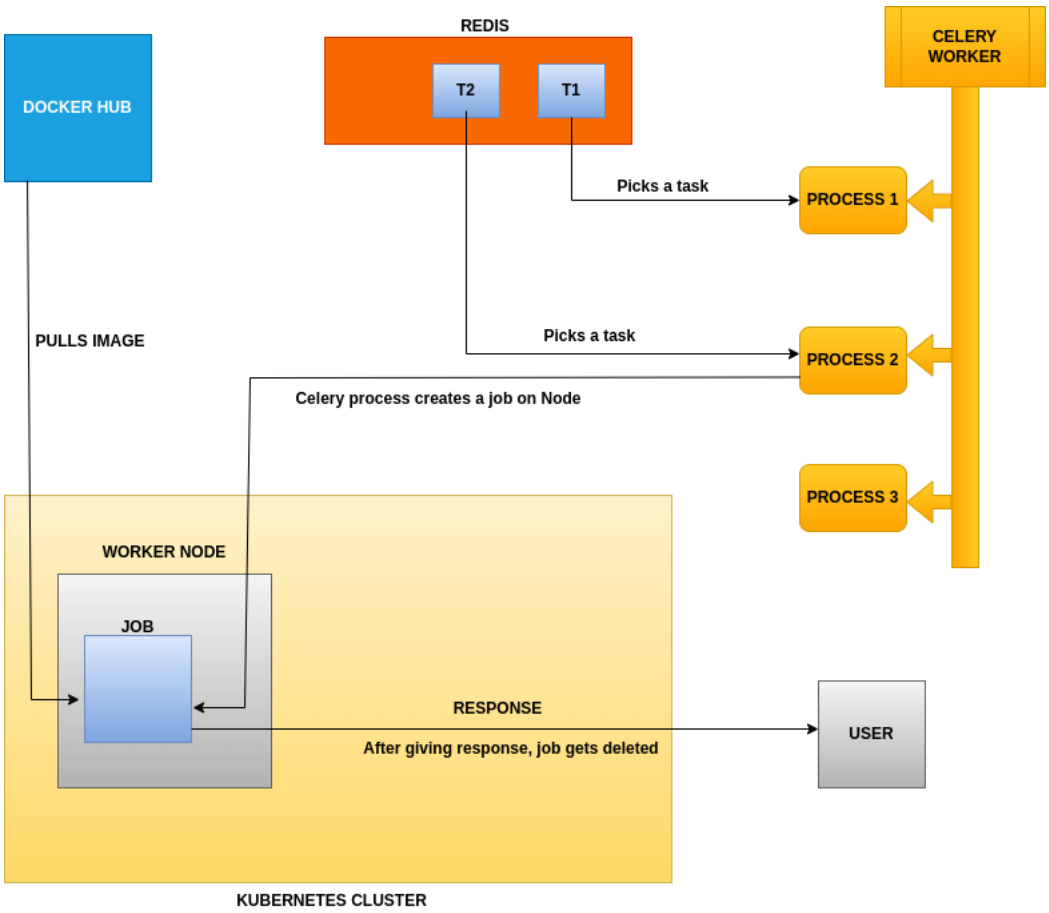

Figure 2: Trigger Dispatch

8. Task Scheduling and Load Balancing

Celery, integrated with Redis Queue, manages task scheduling and load balancing. Redis Queue allows for reliable queuing of dispatch requests, while Celery ensures parallel task execution with multiple worker nodes. If there are multiple dispatch requests, Celery assigns them to different worker nodes, enabling simultaneous execution and improving throughput.

9. Result Retrieval

Once the task is completed, the response is sent back to the user, providing the result of the function execution.

Performance Testing

Initial performance tests focused on the following metrics to evaluate the system's efficiency as the number of requests increased from 1 to 10:

1. Retrieval Time

The time taken from the user's function dispatch to the completion and retrieval of results. This metric helps determine the responsiveness of the system.

2. CPU Utilization

The percentage of CPU usage during task execution, indicating system load as the request volume increases.

Observations will be provided in the slides.

Conclusion

This report details a Function-as-a-Service (FaaS) platform that uses Kubernetes to manage container-based tasks, Docker for containerization, and Redis Queue with Celery for task scheduling. The platform allows users to submit functions through a user interface and dispatch them to a Kubernetes cluster for execution.

Redis Queue provides queuing of tasks, while Celery allows for parallel task execution across multiple worker nodes. This design promotes simultaneous task processing. As a result, the platform can handle multiple user request.