Report on Faast-A-Faas-Framework

Arijit Saha (210050017) and Aryan Mathe (210050021)

1 Introduction

Faast-A-Faas-Framework is a project designed to evaluate and compare different cluster configurations for Function as a Service (FaaS) platforms. The goal is to analyze performance metrics such as latency, throughput, and resource utilization across various setups. This report provides an overview of the project's approach, including the types of clusters tested, the workloads used for evaluation, and instructions for setting up and running experiments. By offering insights into the impact of different configurations, this project aims to guide best practices for optimizing serverless infrastructure

2 Cluster Configurations

The project considers the following cluster configurations:

2.1 Single Pod Cluster

2.2 Single Pod with Multi-Container

2.3 Multi-Pod with Single Node

2.4 Multi-Pod with Multi-Node

2.5 Horizontal Pod Autoscaler (HPA)

2.6 Vertical Pod Autoscaler (VPA)

This configuration consists of a single pod with a single container deployed on a single-node cluster. The pod hosts the FaaS service, and all incoming requests are directed to the single container within the pod. This simple setup serves as a baseline for comparison with other configurations.

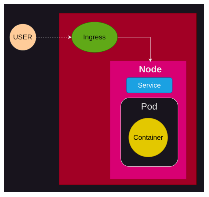

Figure 1: Single Pod Cluster

In this configuration, a single pod contains multiple containers that run the FaaS service on a single-node cluster. An Nginx load balancer pod is also deployed to route incoming requests to the containers within the pod. This setup allows testing the efficiency of using multiple containers within a single pod.

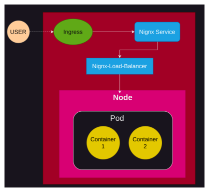

Figure 2: Single Pod with Multi-Container

This configuration includes multiple pods, each with one container running the FaaS service, deployed on a single node. An Nginx load balancer pod manages incoming requests, routing them to the appropriate FaaS service pods. This setup tests how multiple pods interact within a single node and the impact of load balancing.

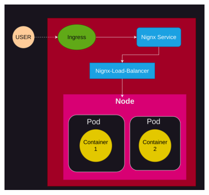

Figure 3: Multi-Pod with Single Node

3 Metrics

This section discusses the metrics used to evaluate the performance and resource utilization of the Function as a Service (FaaS) platform across different cluster configurations. The two main types of workloads tested were a simple loop and a large Lorem Ipsum generator.

3.1 Latency and Throughput

Latency and throughput are critical performance metrics for evaluating FaaS services:

- **Latency:** The time taken for a request to be processed by the FaaS service and return a response. Lower latency is generally preferred as it indicates faster response times and better user experience.
- **Throughput:** The number of requests that can be processed per unit of time. Higher throughput signifies the ability to handle more requests concurrently and efficiently.

To measure these metrics, the ‘wrk‘ tool was used to simulate load on the FaaS service and capture response times and request rates. ‘wrk‘ was configured to send a defined number of requests to the service for each cluster configuration. The results were then analyzed to determine the average latency and throughput.

3.2 Resource Utilization (CPU and Memory)

Resource utilization is another important aspect of evaluating FaaS services:

- **CPU Usage:** The amount of CPU resources consumed by the FaaS service. Lower CPU usage is preferred as it indicates efficient use of processing power.
- **Memory Usage:** The amount of memory consumed by the FaaS service. Lower memory usage is desirable as it implies efficient use of available memory resources.

Resource usage was measured using the Kubernetes ‘metrics-server‘ API, which provides real-time data on the CPU and memory usage of pods and containers within the cluster. This data was collected and analyzed for each cluster configuration.

3.3 Workloads for Testing

The project tested the FaaS platform using two types of simple workloads:

In this configuration, multiple pods are distributed across two different nodes, with each pod containing one container running the FaaS service. An Nginx load balancer pod manages incoming requests, distributing them evenly across the different nodes and their pods. This setup tests how the system performs when multiple pods are deployed across multiple nodes.

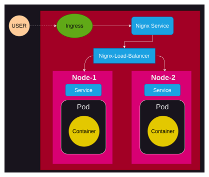

Figure 4: Multi-Pod with Multi-Node

The Horizontal Pod Autoscaler (HPA) is a Kubernetes feature that automatically scales the number of pods in response to changes in resource utilization or other metrics. In this configuration, multiple pods (initially one) are deployed on a single node. The HPA monitors resource usage (such as CPU or memory) and dynamically adjusts the number of pods based on predefined thresholds, ensuring efficient resource usage and consistent performance.

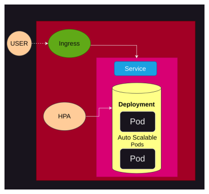

Figure 5: Horizontal Pod Autoscaler

- **Beautiful Loops:** This workload runs a for loop performing simple computations, providing insight into the performance of the FaaS service under repetitive tasks.
- **Large Lorem Ipsum Generator:** This workload generates a large block of Lorem Ipsum text and returns it as a response, testing the service's ability to handle data-heavy tasks and manage memory usage efficiently.

These workloads were chosen to provide a diverse set of scenarios for testing the FaaS service. The results of latency, throughput, and resource utilization were measured and compared across different cluster configurations.

In summary, these metrics help assess the FaaS platform's performance and resource usage under varying conditions, providing valuable insights for optimizing serverless applications.

4 Workloads for Testing

The project uses two types of simple stateless workloads to test various cluster environments:

4.1 Beautiful Loops

This workload runs a for loop performing some simple computations.

4.2 Random Weird Text

This workload generates some random text and returns it as a response.

5 Requirements

To run the experiments, the following tools need to be installed:

- docker
- kubectl
- minikube
- helm

The Vertical Pod Autoscaler (VPA) is a Kubernetes feature that adjusts the resource limits and requests of a pod based on its actual usage. In this configuration, a single pod is deployed with VPA enabled in a single-node cluster. The VPA monitors the pod's resource usage and dynamically adjusts its resource requests and limits to ensure optimal performance and resource efficiency.

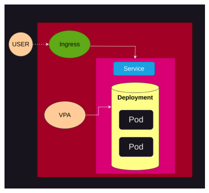

Figure 6: Vertical Pod Autoscaler

6 Running Instructions for Setting up a Cluster Environment

To set up the environment for running cluster configurations, run the following command:

```
bash setup.sh
```

The supported app-types based on the cluster configurations defined above are:

```
single-pod  two-pod-same-node  two-pod-diff-node  hpa  vpa  two-container
```

```
bash deploy_app.sh <app_name> <app_type> <docker_image_name> <python-app-file> <requirements-
file> <port> <map_url>
```

7 Generating Analysis Results

To generate analysis results, follow these steps:

1. Set up the 'metrics-server' REST-API by running the following command in two different terminal windows:

   ```
   minikube dashboard --port=20000
   ```

2. Run the following command to perform the analysis for different cluster configurations and generate logs for response-time and resource utilization:

   ```
   bash src/analysis/perform_analysis.sh <host> <url> <app-type> <app-name>
   ```

   This will generate logs for response-time and resource utilization based on the defined workload. The file names will be in the following format:

   ```
   <logs-dir>/<app-name>-<app-type>-response_time.csv
   <logs-dir>/<app-name>-<app-type>-resource_usage.csv
   ```

3. Execute the following Python file to generate plots for the analysis:

   ```
     python3 analysis/get_plot_from_log.py
     usage: script to generate plot from log files [-h] --app-type APP_TYPE
                                                   [--response-log RESPONSE_LOG]
                                                   [--resources-log RESOURCES_LOG] --output-
   folder
                                                  OUTPUT_FOLDER --app-name APP_NAME
   ```

```
   options:
     -h, --help            show this help message and exit
     --app-type APP_TYPE   single-pod/two-pod-same-node/two-pod-diff-node/hpa/vpa/two-
container
     --response-log RESPONSE_LOG
     --resources-log RESOURCES_LOG
     --output-folder OUTPUT_FOLDER
     --app-name APP_NAME
```