Report on Faast-A-Faas-Framework

Claude

1 Introduction

Faast-A-Faas-Framework is a project that aims to test various cluster configurations for Function as a Service (FaaS) platforms and perform analysis using metrics such as latency, throughput, and resource utilization. This report provides a detailed overview of the project, including the cluster configurations, metrics used, workloads for testing, requirements, and instructions for setting up and running the experiments.

2 Cluster Configurations

The project considers the following cluster configurations:

2.1 Single Pod Cluster

This configuration contains a single pod with a single container deployed in a single node cluster. The following image illustrates the cluster configuration:

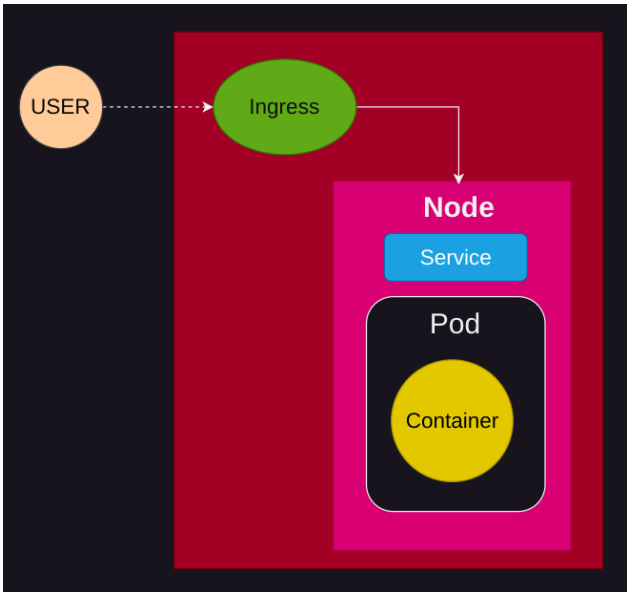

Figure 1: Single Pod Cluster

2.2 Single Pod with Multi-Container

This configuration contains a pod with multiple containers deployed in a single node cluster, with each container running the same service. It also includes a pod that runs an nginx-loadbalancer to route incoming requests to both containers of the pod running the FaaS service. The following image illustrates the cluster configuration:

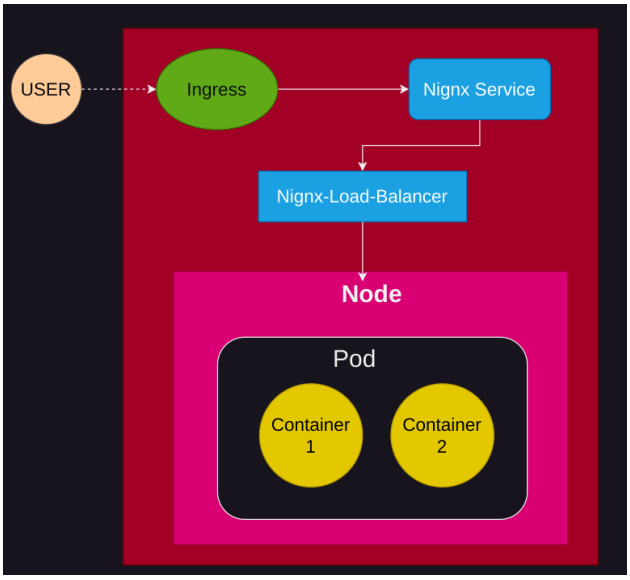

Figure 2: Single Pod with Multi-Container

2.3 Multi-Pod with Single Node

This configuration contains multiple pods, each containing one container that runs the FaaS service, deployed in a single node. It also includes a pod that runs an nginx-loadbalancer to route incoming requests to the pods running the FaaS service. The following image illustrates the cluster configuration:

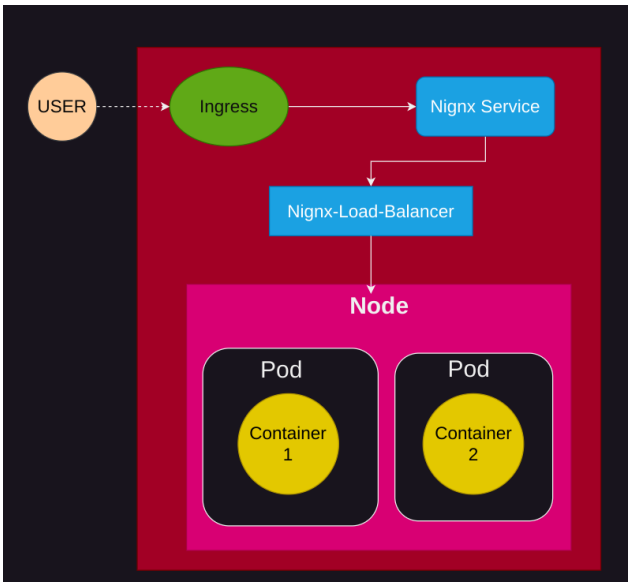

Figure 3: Multi-Pod with Single Node

2.4 Multi-Pod with Multi-Node

This configuration contains multiple pods, each containing one container that runs the FaaS service, deployed in two different nodes. It also includes a pod that runs an nginx-loadbalancer to route incoming requests to the pods running the FaaS service via their respective services. The following image illustrates the cluster

configuration:

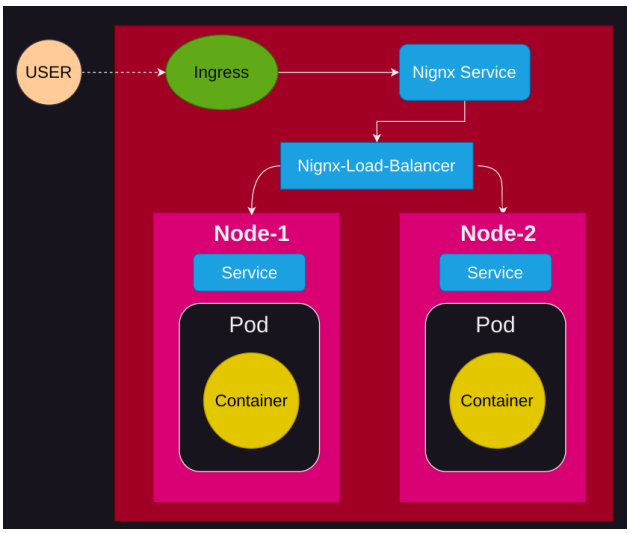

Figure 4: Multi-Pod with Multi-Node

2.5 Horizontal Pod Autoscaler

This configuration contains multiple replica sets of pods (initially containing a single pod) deployed in a single node. The Horizontal Pod Autoscaler (HPA) scales out when the resource requirements go beyond the set limits and adjusts the number of pods accordingly. The following image illustrates the cluster configuration:

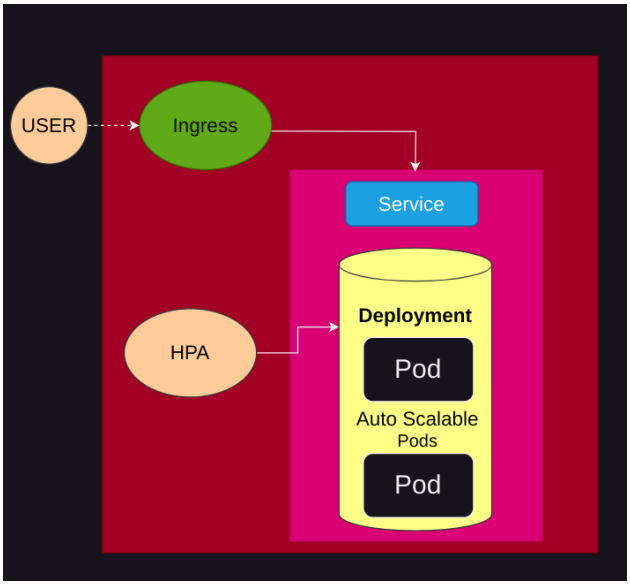

Figure 5: Horizontal Pod Autoscaler

2.6 Vertical Pod Autoscaler

This configuration contains a single pod enabled with Vertical Pod Autoscaler (VPA) deployed in a single node. The VPA scales up when the resource requirements go beyond the set limits and adjusts the pod resources accordingly. The following image illustrates the cluster configuration:

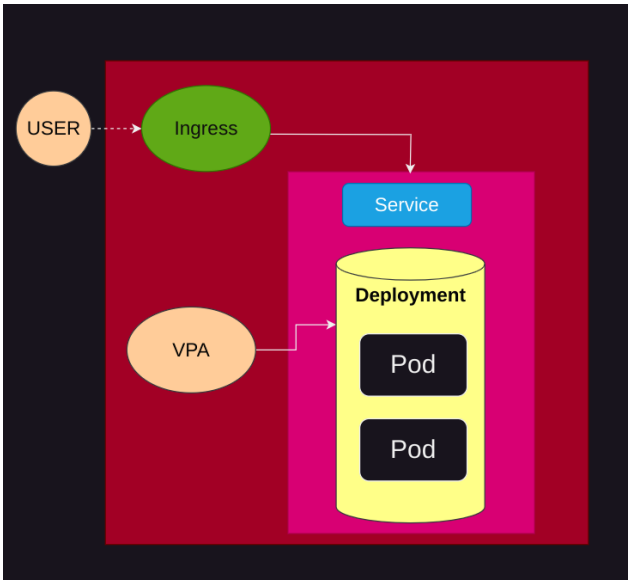

Figure 6: Vertical Pod Autoscaler

3 Metrics

The project measures the following metrics:

3.1 Latency and Throughput

The latency and throughput of the FaaS service are measured using the ‘wrk‘ tool. The analysis is performed for different cluster configurations, and the results are compared. The following image shows the latency and throughput comparison for different cluster configurations:

Figure 7: Latency and Throughput Comparison

3.2 Resource Utilization (CPU + Mem)

The CPU and memory utilization of the FaaS service are measured using the ‘metrics-server‘ API of Kubernetes. The analysis is performed for different cluster configurations, and the results are compared.

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

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
```

To set up the requirements for running clusters, run the following command with the appropriate ‘¡app$_type >$ ‘ :

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

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
```