

INDIAN INSTITUTE OF TECHNOLOGY BOMBAY

Department of Computer Science and Engineering

CS695 : TOPICS IN VIRTUALIZATION AND CLOUD COMPUTING

FaaSINETES

A FaaS Wrapper around Kubernetes



Submitted By

Swetha M (23M0756)
Chaitra Gurjar (23M0831)

Submitted on

Friday, 03 May, 2024

TABLE OF CONTENTS

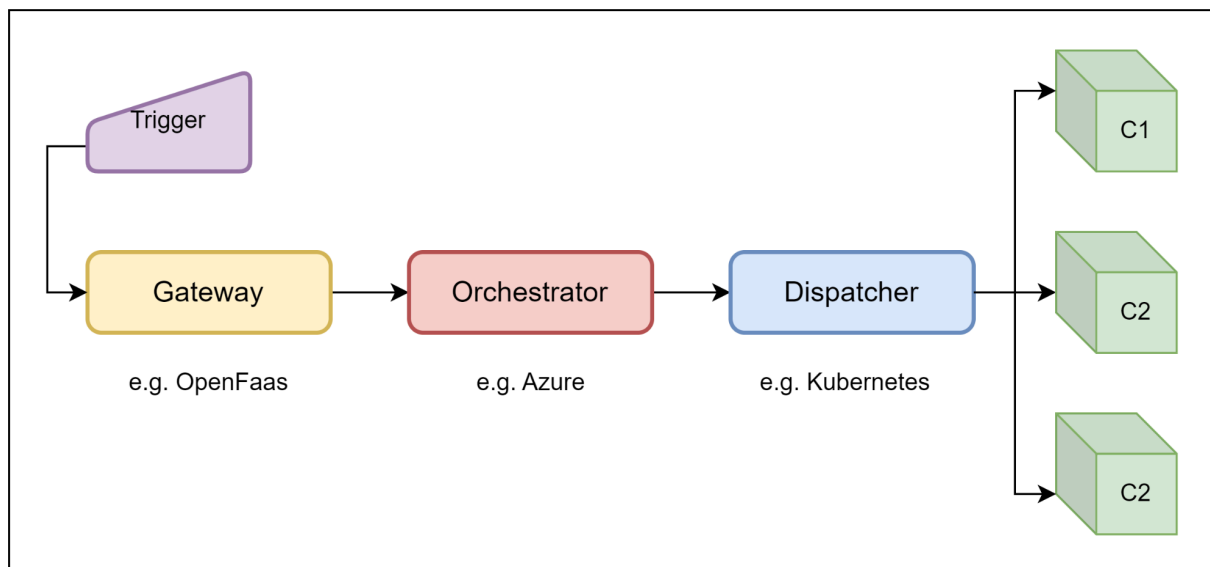
1	Introduction	3
2	Tech Stack	4
3	Approach	4
4	Testing	6
5	Metrics	8
6	Conclusion and Future Work	9
7	References	9

1 INTRODUCTION

Function as a Service. FaaS is a cloud computing paradigm in which cloud service providers oversee the execution of particular functions or code segments in response to predetermined events or triggers. Developers write code snippets in the form of functions and submit them to the providers without having to concern about the underlying technology, number of servers or the operating systems.

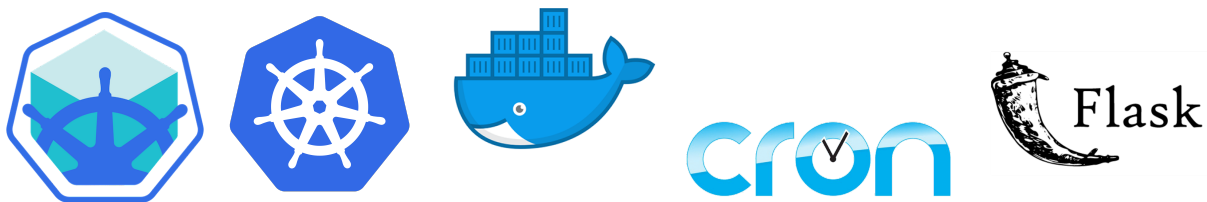
Serverless Computing. Cloud service providers use this technology to decouple deployment and development of functions. The event driven design enables scalability, flexibility and an ease of maintenance based on workload demands. The **event or trigger** is given as an input to a gateway, which is then forwarded to the orchestrator. The **orchestrator** then passes on the events to the **dispatcher**, which then starts containers or pods (known as servers) to run these functions. This is especially useful for cost intensive applications, where the provider charges an amount only for the time the server runs on the cloud.

Triggers. Triggers are stimuli to a gateway which execute the corresponding functions. These may include database, HTTP, timer or any other events defined by the client. In this project we have provisioned **two triggers : HTTP and timers.**



2 TECH STACK

- **Minikube** : This is an open source tool for developers to run single node Kubernetes clusters within a local system.
- **kubectl** : This is a command line tool to easily interact with any Kubernetes cluster and deploy applications on it.
- **Docker** and **DockerHub** : Docker enables users to abstract applications inside a container in a reproducible environment. We use DockerHub to host such container applications as repositories which can be pulled into a server.
- **crontab** : This is a command line utility in Linux like systems to schedule events at certain time periods. We use this to trigger timer events.
- **Flask** : This is a flexible web framework for Python to build web apps and APIs for them. We create a HTTP listener using flask and routes to listen for HTTP triggers corresponding to functions registered.



3 APPROACH

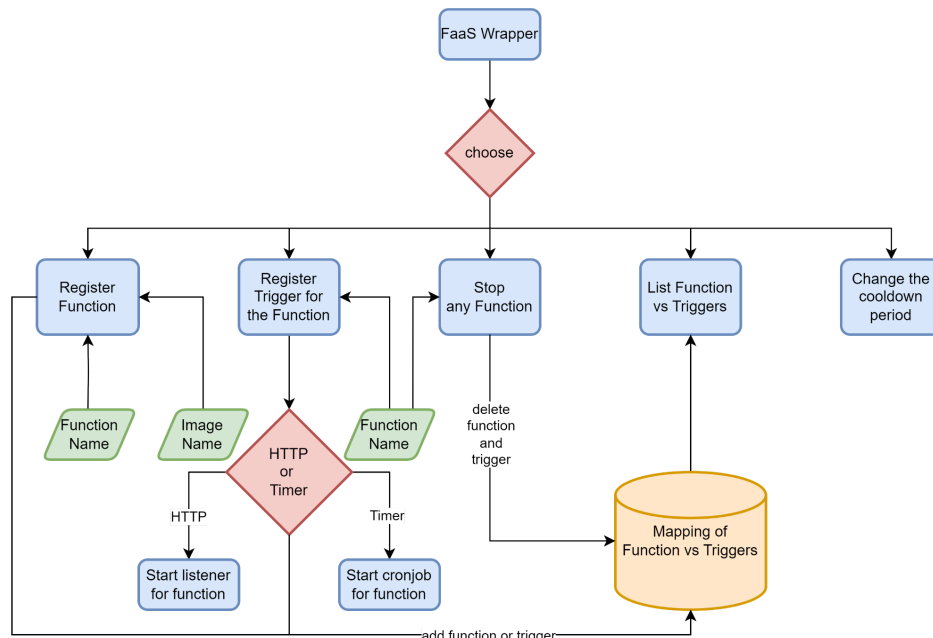
FaaS Wrapper.

- 1) The FaaS wrapper is a Python script which consists of five features as mentioned in the figure. All deployment activities are started using this script.
- 2) The '**register function**' is used by the client to name their function and pull a docker image associated with it.
- 3) The '**register trigger**' takes the function name and trigger type as its input and dispatches the appropriate trigger for it.

- 4) Trigger **dispatch** implies that a deployment and a service is created for a function in a pod on minikube, which is triggered using the event registered before.
- 5) Any function can be **stopped** by providing its name. All deployments, services in pods, and the function vs trigger mapping is deleted in this step.
- 6) The wrapper can also persistently **list mapping of a function to trigger**.
- 7) The **cooldown period** for this wrapper is the time for which a deployment runs in a pod. After this time, the function is stopped in a serverless setting. This period can be changed.

Dispatcher (trigger_dispatch in our code).

- 1) If the user chooses an HTTP trigger, we provide a URL to stimulate the function. Clicking on this URL executes the trigger dispatch script.
- 2) If the user chooses a timer trigger, we start a cronjob with the frequency provided by the user. The cronjob runs the trigger dispatch script every 'x' minute.
- 3) The trigger dispatch script creates a deployment and a service for the function for which it was triggered.
- 4) It then exposes the port from the pod and forwards it to the localhost for a local view.
- 5) It waits for a 'cooldown period' after which the function is stopped and all pods related to it are deleted.



4 TESTING

We created two containerized applications for testing - one for HTTP trigger and another for timer trigger.

- 1) Fortune Teller - This is a simple web application to display fortunes upon clicking a button. We used an HTTP trigger to start this website on some port on the localhost.

```
Enter the function name
onion
Enter the image name
chaitragurjar/test-flask

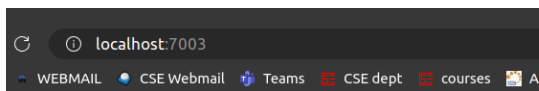
=====
Enter:
1 to Register a function,
2 to Register a Trigger,
3 to List Functions and Triggers,
4 To stop a function
5 to change cooldown period
6 to exit
=====

2
Enter the function name
onion
Enter the trigger name: 1. HTTP, 2. CronJob
1
HTTP Trigger re Follow link (ctrl+click) ly
Trigger using: http://localhost:5000/trigger/onion

=====
Enter:
1 to Register a function,
2 to Register a Trigger,
3 to List Functions and Triggers,
4 To stop a function
5 to change cooldown period
6 to exit
=====
```

```
localhost:5000/trigger/onion

1 {
2   "image": "chaitragurjar/test-flask",
3   "port": "7003",
4   "trigger_value": "HTTP"
5 }
```



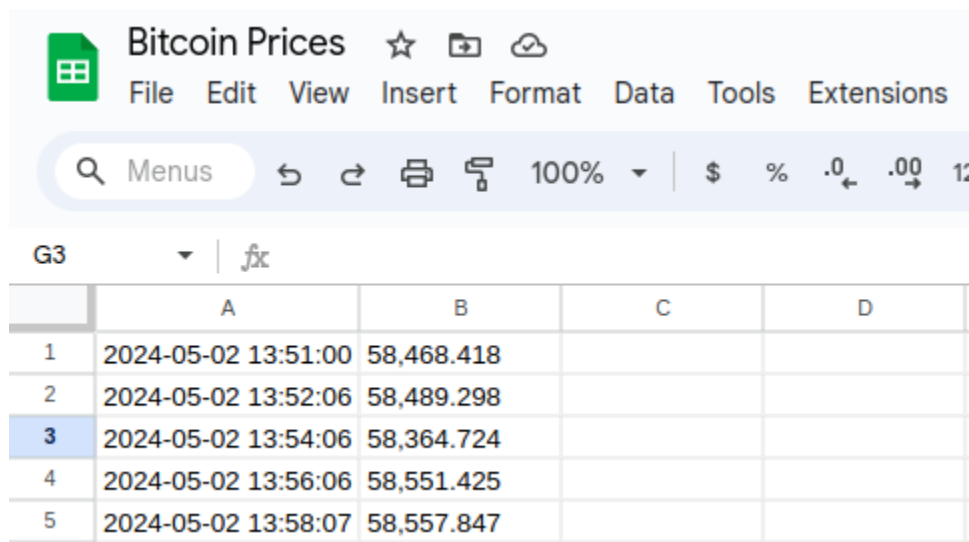
What is your fortune? Beware the one behind you.

Get Fortune

Get Fortune

- 2) Bitcoin Price Tracker - This is a Python application to track the price of Bitcoin and write it on a hosted source (like google sheets). We used a timer trigger to start this application every hour to track the prices.

```
=====
1
Enter the function name
potato
Enter the image name
chaitragurjar/price-tracker
=====
Enter:
1 to Register a function,
2 to Register a Trigger,
3 to List Functions and Triggers,
4 To stop a function
5 to change cooldown period
6 to exit
=====
2
Enter the function name
potato
Enter the trigger name: 1. HTTP, 2. CronJob
2
Enter frequency in min
2
bash create_cronjob.sh "*/* * * * *" potato chaitragurjar/price-tracker 7004 30 &
Creating cronjob for */2 * * * *
CronJob created successfully
```



The screenshot shows a Google Sheet titled "Bitcoin Prices" with a menu bar (File, Edit, View, Insert, Format, Data, Tools, Extensions) and a toolbar with search, undo, redo, print, copy, paste, zoom (100%), and currency symbols (\$, %, .0, .00, 1). The sheet contains a table with 5 rows of data. The first column is labeled "G3" and the second column is labeled "fx". The table has columns A, B, C, and D. The data in column B represents Bitcoin prices.

	A	B	C	D
1	2024-05-02 13:51:00	58,468.418		
2	2024-05-02 13:52:06	58,489.298		
3	2024-05-02 13:54:06	58,364.724		
4	2024-05-02 13:56:06	58,551.425		
5	2024-05-02 13:58:07	58,557.847		

5 METRICS

Main goal of serverless is to reduce the expenses on a serverful setting which wastes resources even when the application is not required. Hence we provision serverless functions on demand. One primary metric would be to measure the amount of resources saved by switching to a serverless setting. This would proportionally save capital on serverful architecture.

A disadvantage of Serverless is the cold start problem. Another important metric would be the amount of time a client has to wait before the function becomes available. The resources saved come at the cost of availability.

Metric 1: Resource saved - Serverful vs Serverless

We assume that we track the price of bitcoin using our application every 1 hour for 1 week. The cooldown period for the server is 5 minutes.

Total time taken for serverless $= 7 * 24 * 5 \text{ minutes} = 840 \text{ minutes}$

Total time for serverful $= 7 * 24 * 60 \text{ minutes} = 10,080 \text{ minutes}$

Now assuming that a service provider charges ₹ 0.01 per minute.

Cost for serverless application $= 840 * 0.01 = ₹ 8.4$

Cost for serverful application $= 10080 * 0.01 = ₹ 100.8$

Hence, for a cooldown period of 5 minutes, cost is saved by 12 times.

6 CONCLUSION AND FUTURE WORK

In this project, we have focused on two triggers - HTTP and timer. We can also include database changes, events queues and external triggers. We also noticed a cold start time period for starting a server in a pod. This can be reduced by thinking of some optimizations and evaluating performance vs overload. We can look after scalability by maintaining replica sets and elastic resources. We can further enhance client usability by allowing them to directly submit code files instead of pulling images from repositories. Certain challenges were faced by us during this project like - initial setup of the tech stack, considering port forwarding to localhost, exposing IP addresses etc. All in all, we enjoyed working on this project, gaining multiple skill sets in the process and discovering many interesting technologies.

7 REFERENCES

[1] Kubeless vs Fission :

<https://medium.com/@natefonseka/kubeless-vs-fission-the-kubernetes-serverless-match-up-41f66611f54d>

[2] Kubectl :

<https://kubernetes.io/docs/reference/kubectl/>

[3] Minikube Quick Start :

<https://minikube.sigs.k8s.io/docs/start/>

[4] Serverless Computing and FaaS :

<https://medium.com/swlh/everything-you-need-to-know-about-serverless-architecture-5cdc97e48c09>