Macau University of Science and Technology

Faculty of Innovation Engineering
School of Computer Science and Engineering
Thesis for Degree of Bachelor of Science

Synthesis and Performance Analysis of Deep Learning Model in Serverless Computing

Student Name : Wang Haocheng
Student No. : 18098537-I011-0112

Supervisor : Subrota Kumar Mondal

July 2022

Abstract

The cloud computing concept was first proposed in 2006. The Serverless computing to be a part of cloud computing become more and more popular. Serverless as a new technology distinguished from traditional cloud computing. It gives developers simple platform framework to deploy function. In the Serverless, developers can pay more attention to the functional coding. Function as a Service (FaaS) to be a main part of Serverless tells us that we simply need deploy the function code on platform - FaaS help us build an end-to-end service. We observe that `OpenFaaS` is the most commonly used FaaS (Serverless) framework.

To help developers understand the fundamentals and working principles of OpenFaaS, this thesis collates and summarizes the concepts based on the available literature. Consequently, it present the first purpose of this paper: learning more details about basic layer components of OpenFaaS. Moreover, developers may also confused about how to operate and manage OpenFaaS. The second purpose of the paper is learning operation of OpenFaaS by analyzing data process. If developers want to learn more, this meet our the third purpose: watching, comparing and analyzing the resource consumption in cold and warm start.

Especially, we fist demonstrate the basic concepts of Serverless and deep learning since analysing deep learning models with Serverless in our goal. Then, we separate OpenFaaS and explain every components. In the next part, we deploy a deep learning model and make use of `JMeter`, which is load testing tool to analyze CPU and memory consumption in the process of OpenFaaS running. Then we get result of analysis. It show how different time_out affect and control function process. We the perform the comparative analysis with the help of `Grafana` (it is a visualization tool) and `JMeter`. Consequently, it is obvious the cold start latency is longer than warm start and pods' auto scaling process in condition of high load. Besides, the paper also show that, the more pods expansion the more load affect to deployment and small scale expansion make less impact. Finally, the last part give much advice for beginners for helping them more comprehensive understanding.

Contents

List of Figures

Abbreviations

K8s	**K**ubernetes
CNCF	**C**loud **N**ative **C**omputing
SaaS	**S**oftware **a**s **a** **S**ervice
PaaS	**P**latform **a**s **a** **S**ervice
IaaS	**I**nfrastructure **a**s **a** **S**ervice
BaaS	**B**ackend **a**s **a** **S**ervice
FaaS	**F**unction **a**s **a** **S**ervice
PC	**P**ersonal **C**omputer
AWS	**A**mazon **W**eb **S**ervices
AI	**A**rtifical **I**ntelligence
ML	**M**achine **L**earning
API	**Application Programming Interface**
MLP	**M**ulti **L**ayer **P**erception
TCP	**T**ransmission **C**ontrol **P**rotocol
IP	**I**nternet **P**rotocol
UI	**U**ser **I**nterface
FTP	**F**ile **T**ransfer **P**rotocol
HPA	**H**orizontal **P**pd **A**utoscaler
CPU	**C**entral **P**rocessing **U**nit
RGB	**R**ed **G**reen **B**lue
KNN	**K** **N**earest **N**eighbors
SSH	**S**ecure **SH**ell
DNS	**D**omain **N**ame **S**ystem
CMD	**Command Line**

Chapter 1

Introduction

1.1 Background and Motivation

Since 2006 the cloud computing concept was first proposed, the cloud computing has played an increasingly important roles [16]. Nowadays, there are many company provide cloud computing services such as AWS [17], Google Cloud [18] and so on. However, the services is the primary stage of cloud computing. This cloud computing which we are using now mostly is IaaS. It is complex and waste many resource. With the development of technology, many other kinds of cloud computing are supported. These years a new concept is popular: FaaS which is a kind of Serverless technology. The Serverless computing is presented base on cloud computing and in many conditions, the former is better than the latter.

Although FaaS can use server more efficient, there is few document focus on FaaS basic components working principle. It is hard for beginners to use it comfortably. In this case, the paper based on available document summarize a completed articles about theoretical background. It introduce what is Serverless computing and the concept of deep learning model which we used in experiment. Then we give details about each components in OpenFaaS. After reading them, developers will have a specific understand of OpenFaaS.

OpenFaaS depends on many other tools. If developers do not know these tools, it will be troublesome to customize their own OpenFaaS program. The function deployment depends on docker and cluster management depends on K8s. Docker is an instance of container. It is simplified virtual machine and Kubernetes is the tool to manage

container cluster. To help them learn the underlying layers and tools, the thesis uses the specific model to explain them step by step. The explanation style is deploying a deep learning model and show the execution process from docker to show result in OpenFaaS. The reason why we choose deep learning is, it is a popular concept. Many developers use FaaS service to train deep learning models. We think this kind of explanation is attractive and interesting.

After deploying, developers may worry about how to manage and safeguard OpenFaaS system, because the data processing is hard to watch. In executing process analysis part, paper show how to realize it. We will learn the way to formulate OpenFaaS configuration. It helps developers know about underlying logic. Finally, the paper focus on `time_out` analysis, verify the details about platform rules to process data.

Although there are many service deployed on OpenFaaS platform, developers confused about extra resource consumption and latency. In order to analysis it, the performance analysis part will introduce the warm and cold start process. Cold start is the time duration of creating now pod and execute function. Warm start is the time duration about reuse the pods to execute function one more time. Then using experiment to watch system status and resource consumption to verify the auto-scaling strategy. The pods will staged expansion when the current pods load achieve CPU load according to HPA. Finally, it will achieve the status of the load of each pods is balance. Consequently, we analyze the cold and warm start latency and resource consumption. According to comparing result, it is clear about resource consumption in different phases.

Many developers want to try deploying OpenFaaS by themselves. However, they may encounter different issues, consequently we demonstrate the viable solution strategies of the common questions or issues encountered, such as how to use local image and the difference between OpenFaaS and OpenFaaS pro. We hope that our endeavor can help the community and ensure better learning and development experience.

1.2 Contributions

In the last paragraph, we talks about background and motivation about the paper. The research and contributions made in this paper will be listed here for the reader to quickly search:

1.2.1 Learn basic structure of Serverless computing

In the front of the paper (Chapter: 2), beginners will get a comprehensive and fine theoretical background of OpenFaaS. It introduce the related research what the paper base on and give a evaluation to exist works. Then in Chapter 3, the paper talks about what is Serverless computing and deep learning. Base on these basic knowledge, paper introduce OpenFaaS and give details explanations about each OpenFaaS components. These will help developers know how OpenFaaS work step by step.

1.2.2 Learn deployment process

After we introduce the system of OpenFaaS, we glad to use a specific experiment to explain the underlying working principle in Chapter 4. From the experiment, beginners can learn many details about each components about Serverless computing, deep learning and OpenFaaS components. We think the way of showing specific model is more attractive to readers. When they want customize system to realize certain function, they know how to modify it.

1.2.3 Learning system operation process

After deploying a specific model, the chapter 5 analysis system operation process. Readers will learn the data process in OpenFaaS by watching how model data is processed by Kubernetes and OpenFaaS. Then, paper show the whole deploying process to users. In the end of this chapter, paper also show the way of analysis. Giving an example to verify theory.

1.2.4 Performance analysis

In the front of chapter 6, we give the concepts which we will analysis later. The paper show the process and running status of system. By watching and comparing result, developers can see which phase and how much deployment consuming performance. we get the more cold start pods there are, the more latency of deployment. Then warm start will consume small resource and have low latency comparing to cold start. After

understanding them, developers can explore how to reduce consumption in the next step.

1.2.5 Advice to beginners

After reading the paper, many beginners want to deploy OpenFaaS by themselves. This chapter is a good tutorial to them. Paper show the troubles when the author meet in experiment here. It will helps beginners to avoid them.

1.3 Outline

Here is the outline of the paper:

1. Chapter 2 talks about the existing works which this paper bases on and give advantages and disadvantages about them.
2. Chapter 3 introduces the fundamentals and working principles of Serverless and OpenFaaS. Moreover introduce the components make up OpenFaaS and how they work.
3. Chapter 4, demonstrates the underlying components of OpenFaaS, how they work and how to deploy function step by step.
4. Chapter 5 shows how to run a deployment in OpenFaaS system and verify the time_out restriction.
5. Chapter 6 introduces some basic concepts and analyzing tools. Then according to compare result, get result of analysis.
6. In the final chapter (Chapter 7), the thesis lists a set of issues encountered while working with and concludes with a set of viable suggestions toward addressing the issues.

Chapter 2

Related work

2.1 Usage of existing works

In the Github, we can find the OpenFaaS tutorial written by OpenFaaS author [19]. It give a simple tutorial about basic function of OpenFaaS. There is another paper[20] talks about cold star and warm start cost in AWS. From these two document, we make up the OpenFaaS system. Then we customize function and deploy deep learning model. Trying to analysis warm and cold start in the system.

2.2 Advantage and disadvantage of existing work

In Github tutorial, the recommended OpenFaaS install method is arkade. In our opinion it is an automation script. However, It can not run in some PC environment. We prefer to choose helm which is platform to install applications. Although it more complex, developers can learn more knowledge.

In the tutorial, author recommend using Minikube for beginners. However, Minikube is just an alternative. If users want explore more function excepting the tutorial mentioned, they will spend more time to set Minikube. Here we recommend install k8s directly. In chapter 7, we will also mention it.

Chapter 3

Theoretical background

In this chapter, we introduce the components' concept and function in the system. It will help us understand clearly about the system.

3.1 Serverless computing

Nowadays, computers play one of the most important roles in 21st. Its powerful computing power to provide us convenient lives. Distributed computing to be a model type of computing, it has been more and more general [21]. The Serverless computing that this paper talks about is one of the distributed computing. Since the concept of cloud computing was introduce in 2006, the cloud computing has been one of the most popular productions for developers. Most of cloud computing server is "IaaS", which means Infrastructure as a Service. It allows developers get access to machine and install their own operating system and machines, but without having the security which holds of the hardware as a service. We can also understand cloud computing in this way: We rent a Tecent cloud service and install an operate. We sent command to the cloud and use it to compute without consuming our computer resource. However, it is not enough. We may waste a lot of resource when we do not use cloud computing service because it runs all time. Then we have Serverless computing. The Serverless cloud computing provide these two kinds of server: "FaaS(Function as a Service)" and "BaaS(Back-end as a Service)".

If we want to know understand “BaaS”, we also need understand what are “SaaS(Software as a Service)” and “PaaS(Platform as a Service)”. In our opinion, the “SaaS” is smaller than “BaaS” and the “BaaS” is smaller than “PaaS”. “SaaS” means the provider give a software to consumer. The consumer just pay for one bill and he can use the software and its whole platform. It just like we buy an application in Apps store. We do not need to care anything except application. “SaaS” will save consumers much money and time because they no need build operating environment and service.

The “BaaS” is bigger than “SaaS” because we need to provide front-end to the service. However, we pay more time and money to the service, we can design more function or interface which can invoke back-end data. Then “PaaS” is bigger than “BaaS” because we should develop both front-end and back-end. The “PaaS” only provide a platform which contains the operate system, dependent environment and so on to us. Finally, “IaaS” is the biggest, because it only provide us a infrastructure which means server. We need establish all things by ourselves except server.Nowadays, there are many company provide “IaaS” just like Google cloud platform, AWS and “Alibaba” cloud.

Then we are going to introduce “FaaS”. The most popular “FaaS” service is AWS Lambada at present and almost “FaaS” frame based on Kubernetes. We think “FaaS” is similar to Baas, because both of them provide back-end for us. We have explained the concept of “BaaS” before. Here we talk about “FaaS”. Just like its name “Function as a Service”, developer only need to provide a function to service, and “FaaS” service will build a completed operating environment to developer. The function means many kinds of code just like docker file, python and so on. [21]There are many advantages of “FaaS”. The first advantage is simple to deploy. Developer can only focus on designing function. Comparing to traditional front-end development and “BaaS”, design function is easier because developer do not need master skill about front-end development. Besides, developer can save much time about checking and fixing bugs due to there is no need to building complex program.

The second advantage is the “FaaS” will save time and money to manage or maintain. As we know if we want a service to work day and night, we need hire professional technical engineer to avoid operational failure. Besides, if we want to change or add new functions, we also need professional technical engineer. However, when we use “FaaS”,

we can only need someone who can design function code of "FaaS" to realize whole service managing.

The third advantage is flexible. The traditional "IaaS" or "BaaS" is developer need rent a fixed performance equipment to execute service. Developers will be faced with how to solve the problem of insufficient device performance. There will be another problem is the excess performance waste money. But the "FaaS" will not. Developer only needs to pay execute and run function code costs. How much the system cost, how much developer pays. And there is no need to worry about performance insufficient. The "FaaS" service will provide suitable performance. How much does it cost depend on how much performance it needs. In summary, it is really flexible.

There are many service using "FaaS". In our research, we use the OpenFaaS which is one of the most popular open source "FaaS". In the next part, we am going to introduce the deep learning which we deploy on OpenFaaS. We should have a general understanding about deployed model, then we can realize "FaaS" more clearly.

3.2 Deep learning

In our research, the model which we deploy is deep learning. Deep learning is not only one of the applications of "FaaS", but also most popular concept in technology area. As we know in this area, the biggest is AI(artificial intelligence). The most popular method to realize AI is ML(machine learning). And DL(deep learning) is efficient way of ML.

In general, AI can be considered as an advantage tool to analysis and process information as human. Due to the computers' high performance, its efficiency is higher than human. ML is trained by amount of data to help people do many things such as making decision and classification. And then, DL is a more advanced method to realize ML. Comparing to ML, DL need more data and higher performance machine to realize. DL using neural networks to divide a complete works into many layers and each layer can be ML. DL can do more complex information process and get smarter results. DL has many applications such as image process, language translation and autopilot. In chapter 3, I will give a specific model to make it understood better.

3.3 OpenFaaS components

3.3.1 Overview

There are some basic components make up a Serverless OpenFaaS system[22]: main function codes, programs basic environment (dockerfile), containers, OpenFaaS platform, Kubernetes and API gateway. Here We find two images of developers' and operators' views to give developers more specific explanations: (Fig.3.1 Fig.3.2)

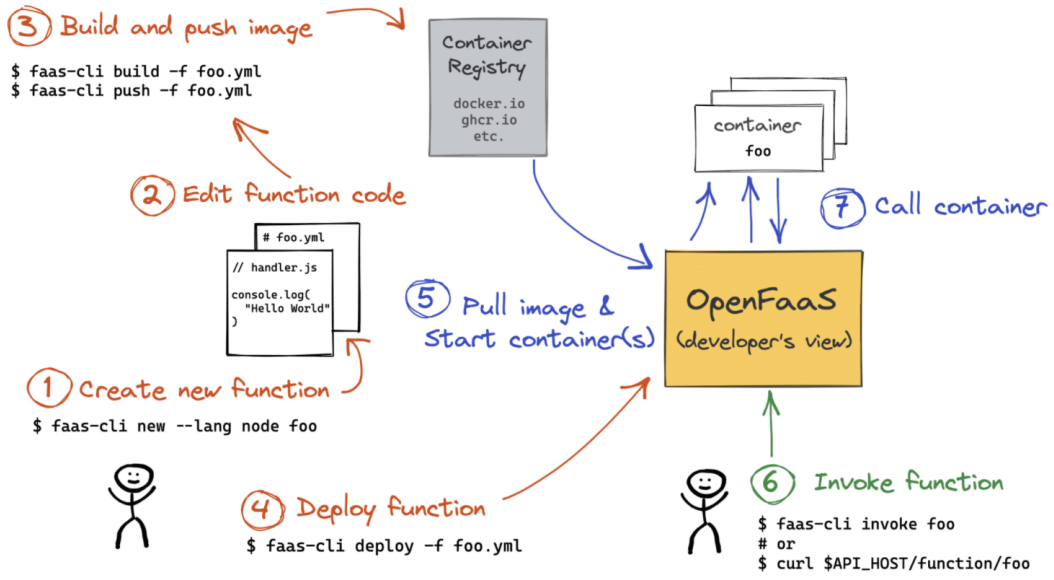

FIGURE 3.1: Developers' view[1]

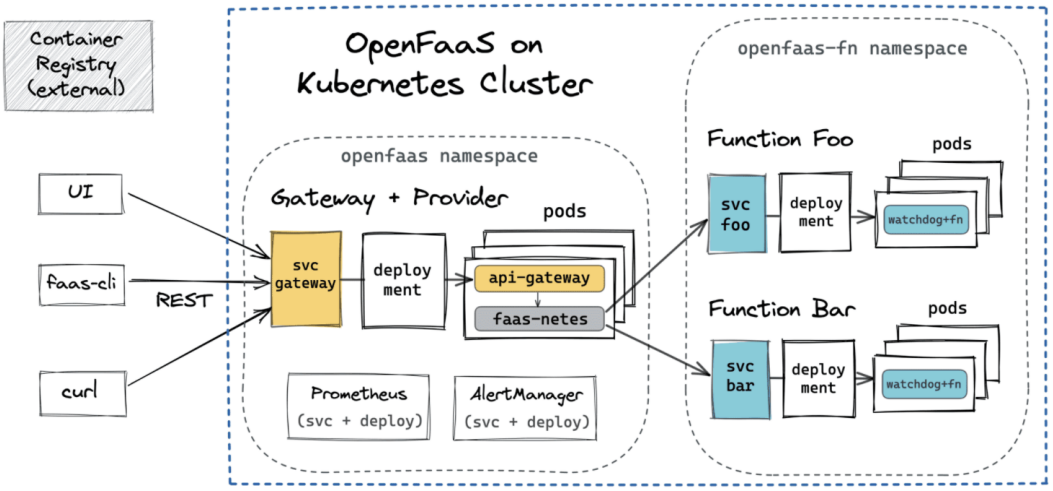

FIGURE 3.2: Operators' view[1]

Next, we will explain how these parts of components work.

3.3.2 Main function code

This part is one of the few parts that needs to be customized by ourselves. We need to write some codes to talk system what function we want to realized. Fortunately, The OpenFaaS provides many program language to developers such as python, go, and so on. In our deployed model, we use python. Python is the most popular program language to analysis data and deep learning.

3.3.3 Program basic environment

If developers have basic computer basement, their codes should be translated and run in the programs. Because different codes need different basic environment, it need to use docker to realize it. This is the reason why docker is the most important part to OpenFaaS. It not only provide dependent environment but also containers(I will introduce it later.). In first, OpenFaaS provide many program code templates. Developers can choose the templates they like. Then Developers can write some code to realize function. If developers want to realize complex function which need some dependent packages, they can add them to the file named requirements. Then developers can use operation tools named “faas-cli” to build push and deploy function. The OpenFaaS will deploy them automatically.

3.3.4 Container

After we have a “program”, we also need something to be container to run it. It just like developers buy a CD, they need a CD player to play it [23]. The program carrier is container. The concept of container is beginning from docker. We need to know what is docker, then we will realize what role does the container play in OpenFaaS. And in the next part, we will give details about this.

3.3.5 OpenFaaS platform

The “faas” service bases on the internet. Developers need a channel to connect and manage FaaS service. Developers can understand the FaaS service as a program which run in the OpenFaaS platform. This platform provide internal and external ports which will expose service to outside. After that, the developers can control FaaS service though ports.

3.3.6 API gateway

In previous, we mentioned, both “Faas” and “Baas”belong to Serverless service. Comparing to “Baas”, “Faas” is a newer concept. So we want to introduce how users interact with “Baas” to explain why “Faas” needs API gateway.

If developers use “Baas” service, they need use some tools to connect, send and receive message to back-end service. The tool’s name is API gateway. API gateway means “Application Programming Interface Gateway”. The basic function of API gateway is accept remote requests and return responses. Actually, API gateway will separate users’ requests and route them to the suitable location. After each back-end receive specific requests, they will send information back to the API gateway to realize users’ request function. The API gateway also has many other function such as intercepting request, Authentication, network speed limitation, billing and monitoring. Then we back to the “Faas” service. The difference between “Faas” and “Baas” is the way of they process back-end data. API gateway provide the same function to the “Faas” service.

In this section, we introduction each components’ function in the OpenFaaS. In the next section, we are going to give developers some details about how we deploy our function to on the OpenFaaS and shows the details about them.

Chapter 4

Details of Deployed Model

4.1 Deep learning model

4.1.1 Image classification model

[24] [25]In this paragraph, we will show a specific example to make people understand why we need deep learning. Here we deploy a image classification model. In order to consider the size of the image, here we prefer to choose size of image is 32x32. (Because we need a set of image to be deep learning model and another set of image to be text model, it is too hard to train for our normal PC.) So, the size of each image is 32x32x3 = 3072 (use RGB model). Here we use the name of image set is "CIFAR-10". This set of images contain 10 kinds of image labels, 50000 training data, 10000 test data and each size of image is 32x32.

Firstly, I use the easiest algorithm named "KNN". We divided each image as a 4x4 matrix. and each position as it grey level (as usual is 0-255). Then we use test image's grey level subtract training image's grey level and add the each part of the result. The result means the similarity between the test image and training image. Finally get 10 images which are the most similar images to the test image. In order to understand, here some explain images: (Fig.4.1 Fig.4.2)

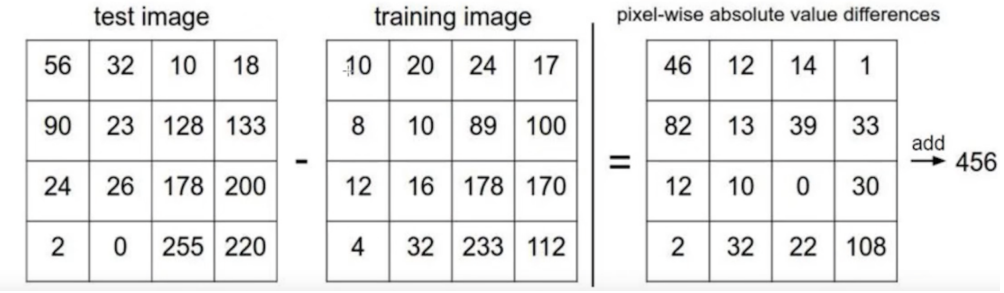

FIGURE 4.1: Matrix

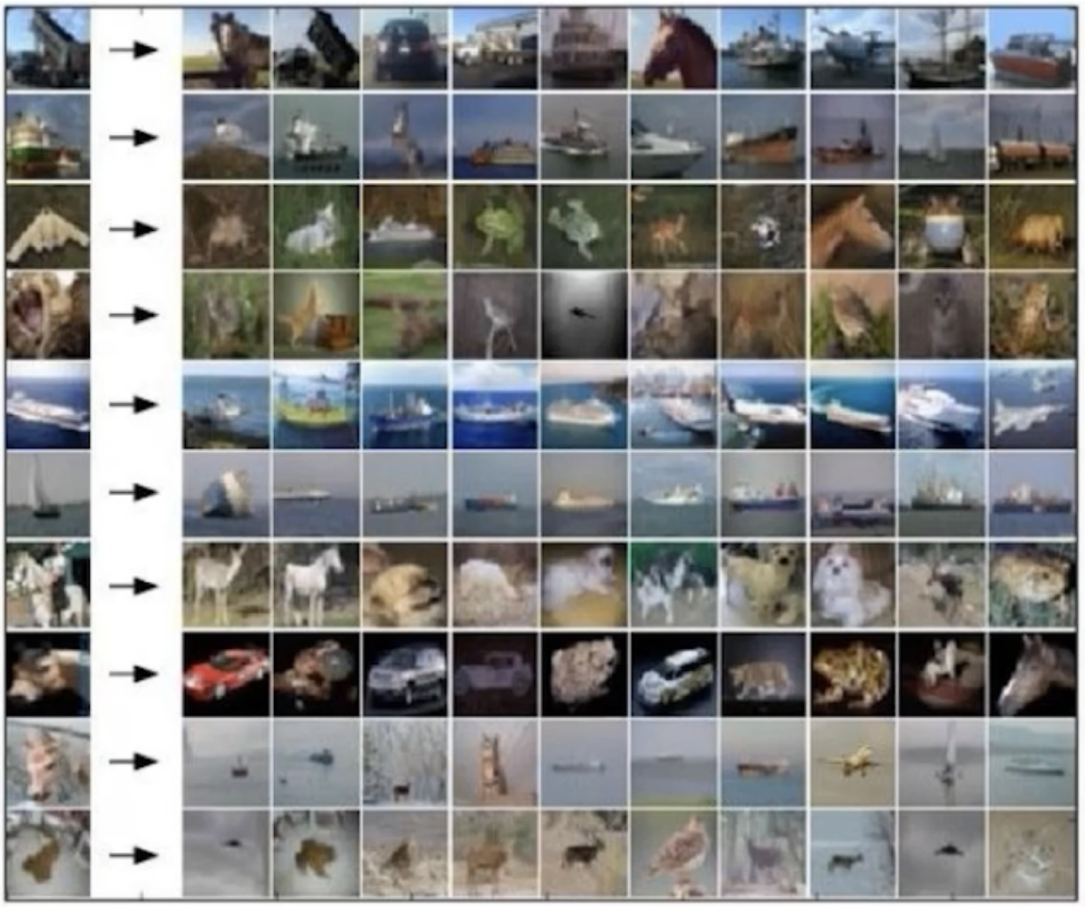

FIGURE 4.2: CIFAR-10 model[2]

Here we will show developers details about this lab in Jupyter: (Fig.4.3)

```
import pickle
import numpy as np
import os

CIFAR_DIR = "/Users/chriswong/lab4/cifar-10-batches-py"
print (os.listdir(CIFAR_DIR))
```

```
['data_batch_1', 'readme.html', 'batches.meta', 'data_batch_2', 'data_batch_5', 'test_batch', 'data_batch_4', 'data_batch_3']
```

```
with open(os.path.join(CIFAR_DIR, "data_batch_1"),'rb')as f:
    data = pickle.load(f,encoding='bytes')
    print (type(data))
    print (data.keys())
    print (type(data[b'data']))
    print (type(data[b'labels']))
    print (type(data[b'batch_label']))
    print (type(data[b'filenames']))
```

```
<class 'dict'>
dict_keys([b'batch_label', b'labels', b'data', b'filenames'])
<class 'numpy.ndarray'>
<class 'list'>
<class 'bytes'>
<class 'list'>
```

FIGURE 4.3: Details 1

In the output 1 we notice that there are 8 files in the "CIFAR-10" fold. The "data_bash_1 to data_bash_5 are the training data and the test_bash is test data. Then we open one file(Here I open data_bash_1), We find there are 4 files in it from output 2. We can also check these 4 files' type. Then we am going to find what these files contain: (Fig.4.4)

```
    print (data[b'data'].shape)
    print (data[b'data'][0:2])
    print (data[b'labels'][0:2])
    print (data[b'batch_label'])
    print (data[b'filenames'][0:2])
```

```
(10000, 3072)
[[ 59  43  50 ... 140  84  72]
 [154 126 105 ... 139 142 144]]
[6, 9]
b'training batch 1 of 5'
[b'leptodactylus_pentadactylus_s_000004.png', b'camion_s_000148.png']
```

FIGURE 4.4: Details 2

We notice that data shape is a matrix of (10000, 3072). 10000 means it contains 10000 data and 3072 means each data size is 3072(I calculated before.). Then we print fist 2 data. We can get each file specific data(Each pixel grey-level between 0-255 which I explained before.). From the web which we download "CIFAR-10", we find there are 10 kinds of image, so the fist 2 labels are kind 10 and 7(The output is from 0 to 9.). The output "b training batch 1 of 5" means it is the first training data set. Finally the last output are the names of the images of the first and second.

However, we will notice that some training images which is selected are similar, some are not similar. If we use more accurate algorithm may avoid this problem. We notice that the eighth row example's result is not good. The selected training image all contain black background. We guess the point is this. Can we classify the main body and background for each images? This will improve accuracy.

4.1.2 Model in experiment

In our research, we choose a simple model named Binary classification which is a kind of MLP(Multi-Layer Perception). This model aim to predict that radar can reflex signal through ionosphere. Because we use python to realize our function, we should announce the dependent package we used (Fig.4.5).

```
In [2]:
1 import tensorflow
2 print(tensorflow.__version__)

2.0.0

In [3]:
1 # mlp for binary classification
2 from pandas import read_csv
3 from sklearn.model_selection import train_test_split
4 from sklearn.preprocessing import LabelEncoder
5 from tensorflow.keras import Sequential
6 from tensorflow.keras.layers import Dense
```

FIGURE 4.5: Dependent package

The Tensorflow is a common package. It can help me build neural network model. Pandas which is a tool based on Numpy can store and analysis data. Here the "read_csv" is our data recorded by excel. Pandas translate our data and build model we need. The full name of "sklearn" is "Scikit-learn". The package contains Estimator and Transformer. The Estimator is used to calculate result from data. the Transformer is used to process data format such as translate three-dimensional array to two-dimensional array. Finally, "keras" is an API which helps us execute other package in back-end. It will be used with Tensorflow together in this model. In the row of"Sklearn.model_selection", we divide a training set. The next row function is I transform the result to 0 and 1.

In next step, we read and transport our data to the model (Fig.4.6).

```
# load the dataset
df = read_csv('/Users/chriswong/Documents/毕业论文/Lesson50-ionosphere.csv', header=None)
df
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9	...	25	26	27	28	29	30	31
0	1	0	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.00000	0.03760	...	-0.51171	0.41078	-0.46168	0.21266	-0.34090	0.42267	-0.54487
1	1	0	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000	-0.04549	...	-0.26569	-0.20468	-0.18401	-0.19040	-0.11593	-0.16626	-0.06288
2	1	0	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965	0.01198	...	-0.40220	0.58984	-0.22145	0.43100	-0.17365	0.60436	-0.24180
3	1	0	1.00000	-0.45161	1.00000	1.00000	0.71216	-1.00000	0.00000	0.00000	...	0.90695	0.51613	1.00000	1.00000	-0.20099	0.25682	1.00000
4	1	0	1.00000	-0.02401	0.94140	0.06531	0.92106	-0.23255	0.77152	-0.16399	...	-0.65158	0.13290	-0.53206	0.02431	-0.62197	-0.05707	-0.59573
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
346	1	0	0.83508	0.08298	0.73739	-0.14706	0.84349	-0.05567	0.90441	-0.04622	...	-0.04202	0.83479	0.00123	1.00000	0.12815	0.86660	-0.10714
347	1	0	0.95113	0.00419	0.95183	-0.02723	0.93438	-0.01920	0.94590	0.01606	...	0.01361	0.93522	0.04925	0.93159	0.08168	0.94066	-0.00035
348	1	0	0.94701	-0.00034	0.93207	-0.03227	0.95177	-0.03431	0.95584	0.02446	...	0.03193	0.92489	0.02542	0.92120	0.02242	0.92459	0.00442
349	1	0	0.90608	-0.01657	0.98122	-0.01989	0.95691	-0.03646	0.85746	0.00110	...	-0.02099	0.89147	-0.07760	0.82983	-0.17238	0.96022	-0.03757
350	1	0	0.84710	0.13533	0.73638	-0.06151	0.87873	0.08260	0.88928	-0.09139	...	-0.15114	0.81147	-0.04822	0.78207	-0.00703	0.75747	-0.06678

351 rows × 35 columns

FIGURE 4.6: Data

From the image we can notice that there are 350 sets of data. Each column represent different variable which affect the result. And in the last column, it give a result represent it can reflex signal from ionosphere or not (Fig.4.7).

In [4]:
```
# split into input and output columns
X, y = df.values[:, :-1], df.values[:, -1]
```

In [5]:
```
# ensure all data are floating point values
X = X.astype('float32')
```

In [6]:
```
X
```

Out[6]:
```
array([[ 1.     ,  0.     ,  0.99539, ..., -0.54487,  0.18641, -0.453  ],
       [ 1.     ,  0.     ,  1.     , ..., -0.06288, -0.13738, -0.02447],
       [ 1.     ,  0.     ,  1.     , ..., -0.2418 ,  0.56045, -0.38238],
       ...,
       [ 1.     ,  0.     ,  0.94701, ...,  0.00442,  0.92697, -0.00577],
       [ 1.     ,  0.     ,  0.90608, ..., -0.03757,  0.87403, -0.16243],
       [ 1.     ,  0.     ,  0.8471 , ..., -0.06678,  0.85764, -0.06151]],
      dtype=float32)
```

FIGURE 4.7: Transform data

Then in this part, we convert data to the matrix except the last column and convert the data to the float. We define the data to X and the result to y (Fig.4.8).

```
In [7]:  # encode strings to integer
         y = LabelEncoder().fit_transform(y)

In [8]:  y

Out[8]: array([1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
               1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
               1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
               1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
               1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
               0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
               0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
               0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
               0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
               0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
               0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
               0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
               1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
               1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
               1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
               1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

FIGURE 4.8: Transform data

In this part the result of good and bad are converted to 0 and 1. They are stored in one-dimensional matrix (Fig.4.9).

```
In [9]:  # split into train and test datasets
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)
         print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

(235, 34) (116, 34) (235,) (116,)

In [10]: # determine the number of input features
         n_features = X_train.shape[1]

In [11]: n_features

Out[11]: 34
```

FIGURE 4.9: Transform data

Here we divide the data according to a certain proportion: (Fig.4.10)

In [12]:
```
# define model
model = Sequential()
model.add(Dense(10, activation='relu', kernel_initializer='he_normal', input_shape=(n_features,)))
model.add(Dense(8, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(1, activation='sigmoid'))
# compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [13]:
```
# fit the model
model.fit(X_train, y_train, epochs=150, batch_size=32, verbose=1)
```

```
Train on 235 samples
Epoch 1/150
235/235 [==============================] - 1s 3ms/sample - loss: 0.7096 - accuracy: 0.4383
Epoch 2/150
235/235 [==============================] - 0s 67us/sample - loss: 0.6722 - accuracy: 0.4128
Epoch 3/150
235/235 [==============================] - 0s 67us/sample - loss: 0.6577 - accuracy: 0.5489
Epoch 4/150
235/235 [==============================] - 0s 77us/sample - loss: 0.6499 - accuracy: 0.7149
Epoch 5/150
235/235 [==============================] - 0s 57us/sample - loss: 0.6418 - accuracy: 0.7234
```

FIGURE 4.10: Adjustment parameters, build and train the model

Here we define neural network parameters. Then build and train the data model (Fig.4.11).

In [14]:
```
# evaluate the model
loss, acc = model.evaluate(X_test, y_test, verbose=1)
print('Test Accuracy: %.3f' % acc)

```

FIGURE 4.11: Evaluate model

After training, we evaluate the model and here is evaluation result: (Fig.4.12)

```
==============================================================================
==============================================================================
==============================================================================
oss: 0.3330 - accuracy: 0.8966
Test Accuracy: 0.897
```

FIGURE 4.12: Evaluate result

Finally we put test data in the model and output a result: (Fig.4.13)

In [7]:
```
# make a prediction
row = [1,0,0.99539,-0.05889,0.85243,0.02306,0.83398,-0.37708,1,0.03760,0.85243,-0.17755,0.59755,-0.44945,0.60536,-0
yhat = model.predict([row])
print('Predicted: %.3f' % yhat)
if yhat >= 1/2:
    yhat = 'G'
else:
    yhat = 'B'
print('Predicteds:', yhat)
```

```
Predicted: 0.740
Predicteds: G
```

FIGURE 4.13: Predict data

4.2 Deploy function in OpenFaaS

In the last part, we deploy our model in the Jupyter notebook. It means our machine learning model is successful. In this part, I change some code to make the function code run on the OpenFaaS platform. OpenFaaS provide a template to developers. The templates define a main frame and it invoke the details code which locate another folder (Fig.4.14).

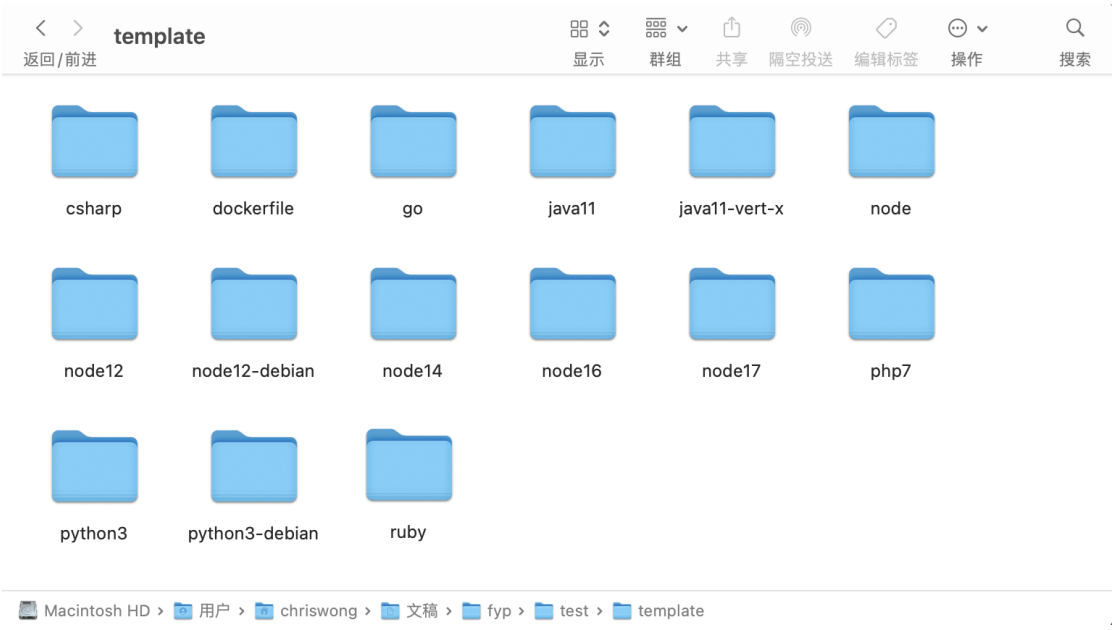

FIGURE 4.14: Example

Here is our model folder. There are many program language template provided to developers. Developers will see that there is a folder named "python3-debian". This is not default template, it needs to be added by ourselves. OpenFaaS are used to multiple uses. And these default template are not available for all areas. Actually, developers can define which model they want to use. Then, the specified model will create a `Dockerfile` automatically (we mentioned that the OpenFaaS based on docker in last section.). Here we want to use some paragraph to introduce docker to help developers understand more details about OpenFaaS.

4.2.1 Docker

Docker [26] is a containerization technology. It is lighter than virtual machine. Traditional virtual machine divides a part of the hardware form the host for itself. And it

needs independent operate system to be a interface which connected to divided hardware. But it will waste much resource not only the resource of virtual machine can not be full used, but also independent operate system will use a part of system resource. As we know, a completed operate system contains interface and kernel. Here we give developers an image to explain it: (Fig.4.15)

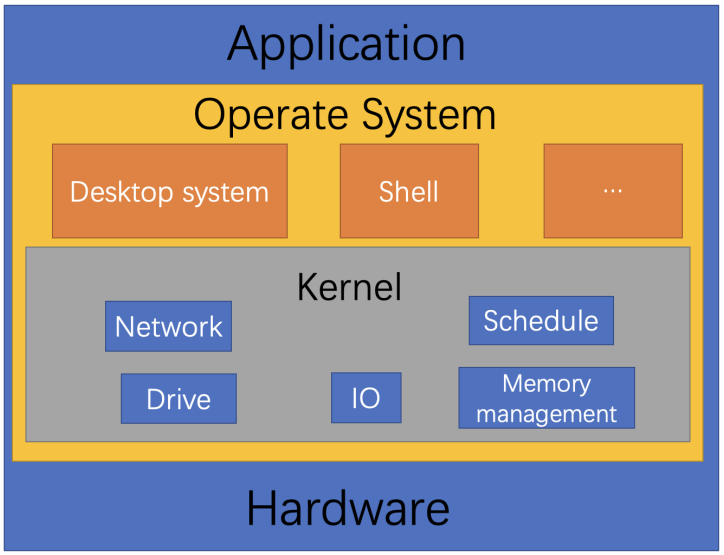

FIGURE 4.15: OS relationship

However, docker will not use the whole operate system. All of the docker container use the kernel of the host and they save much resource (Fig.4.16).

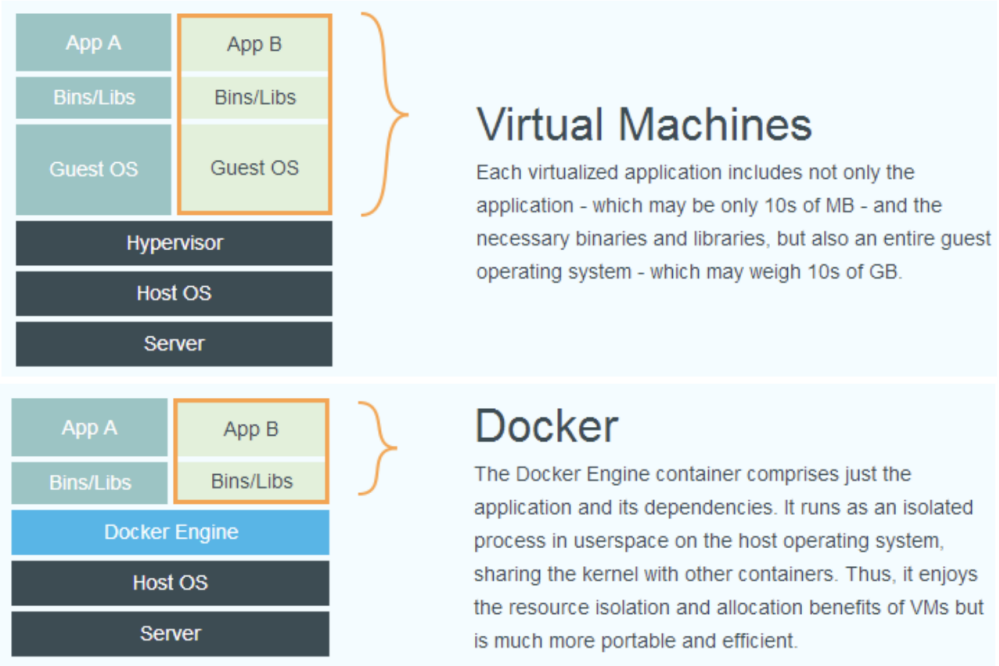

FIGURE 4.16: Docker and Virtual machine [3]

Besides, there is another unique concept named image. Developers can treat images as a light operate system. In the docker folder, developers can see a text file named "dockerfile". Developers can customize and operate docker images in this file. There are many layers in "dockerfile" and each layers is a step to operate images. Usually, the first step to write dockerfile is pulling a basic image from "dockerhub". Developers need to modify basic images or pull other dependency package to realize function.

After customizing docker image, developer can execute it in container. Different form traditional operate system, it is hard to modify the container running status directly. Developer should can change it through docker image and restart it in container to achieve goal.

We would like to note that it is unconditionally impossible to manage containers manually. Kubernetes, the most popular container orchestration platform jumps into the place [27].

4.2.2 Kubernetes

In the company which provide "faas" service to earn money, they will install many "faas" container to the cloud server and these container are called cluster [28] [29]. But how to manage cluster easily? The operators need check and modify the containers one by one? Of course not. This will need many words if they do that. The Kubernetes are designed for manage cluster and here are the function:

1) Service discovery and load balancing:
   Kubernetes can expose containers through OpenFaaS platform ports, and if there is a lot of traffic entering the container, Kubernetes can load balance and distribute network traffic, making deployments stable.

2) Storage orchestration:
   Developers can mount their own storage system to the service automatically through Kubernetes, such as local storage, public cloud providers, etc.

3) automatic deployment and rollback:

   Developers can use Kubernetes to describe the desired state of a deployed container, making each containers to be controlled rate. If the container more than limitation, it will be kill or change by Kubernetes.

4) automatically complete binning calculations:

   Developers can use Kubernetes to specify CPU and memory for each container. Then Kubernetes will manage other containers to use the remaining resources.

5) self-healing:

   Kubernetes restarts, kills or replaces failed containers that do not respond developers' check.

6) Key and configuration management:

   Kubernetes allows developers to store and manage sensitive information such as passwords, OAuth tokens and SSH keys. Developers can deploy and update keys and application configuration without rebuilding container images, and without exposing keys in stack configuration.

4.2.2.1 Kubernetes architecture

In the last paragraph, we know why we need Kubernetes. Kubernetes can deploy and control all of containers although they are not in same machine. Kubernetes divide the whole system in order to distribute them and realize users' demand efficiently (Fig.4.17).

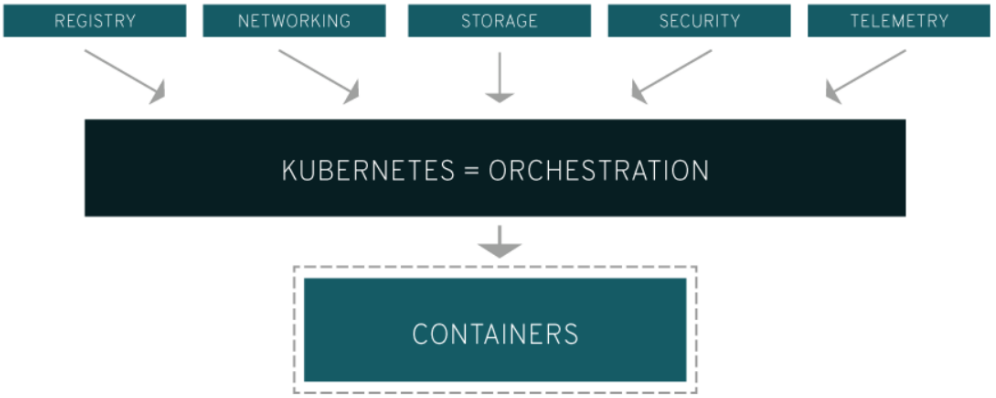

FIGURE 4.17: Kubernetes [4]

Then, we are going to explain each components and how they work together.

Pods: Pod is the smallest indivisible unit in Kubernetes. If developers create a container by Kubernetes, it will create a pod and containers will be created in pod. Each pod contain many containers. The containers in one pod share memory, volumes network and interface. These containers provide different service and they must be used together (Fig.4.18).

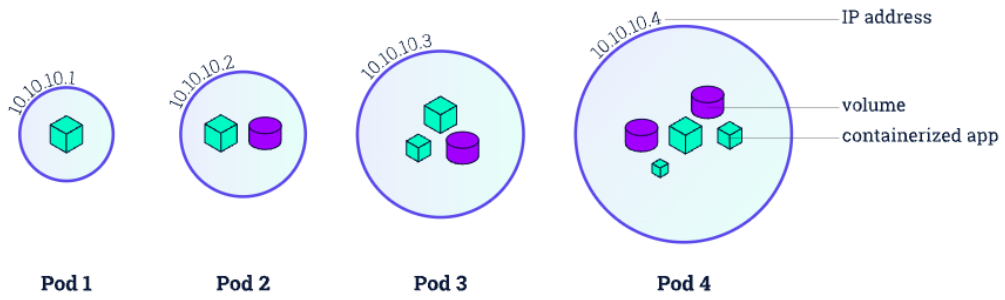

FIGURE 4.18: Kubernetes pods [5]

Node: Mostly, one node is deployed in an independent environment or virtual machine. There are many pods deployed in one node. Developers can treat node as the host of pods. Developers can define each pods' hardware just like assembling computer. Besides, one node can be replaced by other node just like developers can work both on laptop and PC. Then we will show developers the structure of the node: (Fig.4.19)

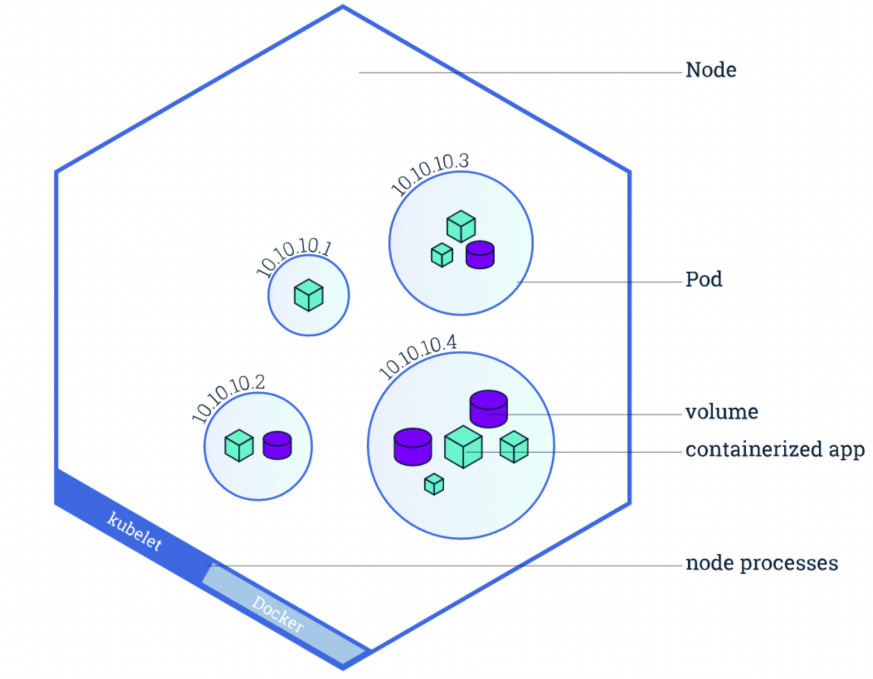

FIGURE 4.19: Kubernetes node [5]

Developers can control these node by the tool named Kubectl which is the Kubernetes official tool. If developers are beginner they can also use "minikuke" to deploy a single node on the local environment. Minikube is a docker image, but Kubernerts not. And there are some restriction on Minikube because it is in the same level to other images. For example, if developers want to use local image they need to set Minikube environment and use docker daemon to build image, otherwise developer can only use cloud image to build image in Minikube.

Cluster: Many nodes make up a cluster. Usually, developers can treat a cluster as a whole thing and do not need to care about the single node in the cluster. If a program is deployed in a cluster, the cluster will distribute assignment to each node automatically. If one node can not run program, the other node in the same node will substitute it. Moreover the nodes in one cluster not only in one machine but also in other different which are connected with network. The cluster can be controlled by kubectl, too (Fig.4.20).

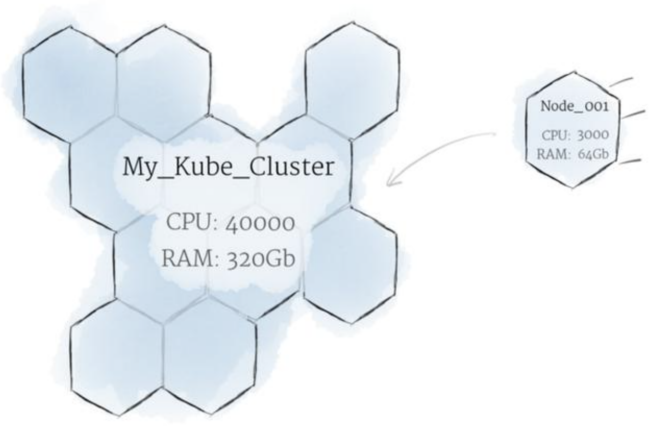

FIGURE 4.20: Kubernetes cluster

Namespace: In a commercial environment, different user always deploy different service in a shared cluster. These service may affect each other and make server slow. In order to solve it, developers need namespace to divide different servers in one cluster. There are many nodes in one namespace and they can use one certain rule which avoid different cluster affect each other.

4.2.3 Docker in OpenFaaS

In the last part, we introduce what is docker and how to use it. However, developers need more steps to use docker in OpenFaaS(If developers deploy it by manual control.). We mentioned that OpenFaaS base on docker. Mostly, developers do not need to care about docker, although it actually use docker. In the OpenFaaS folder, developers will notice there is a text file named "handler.py". They can write the code there, because it is a function what the main invoke. If developers want to use more complex model, they can find the main code in the template which they use. Here are our function code and main code and developers will notice that the function named "handle" are used by main code(the 19 rows) (Fig.4.21 Fig.4.22).

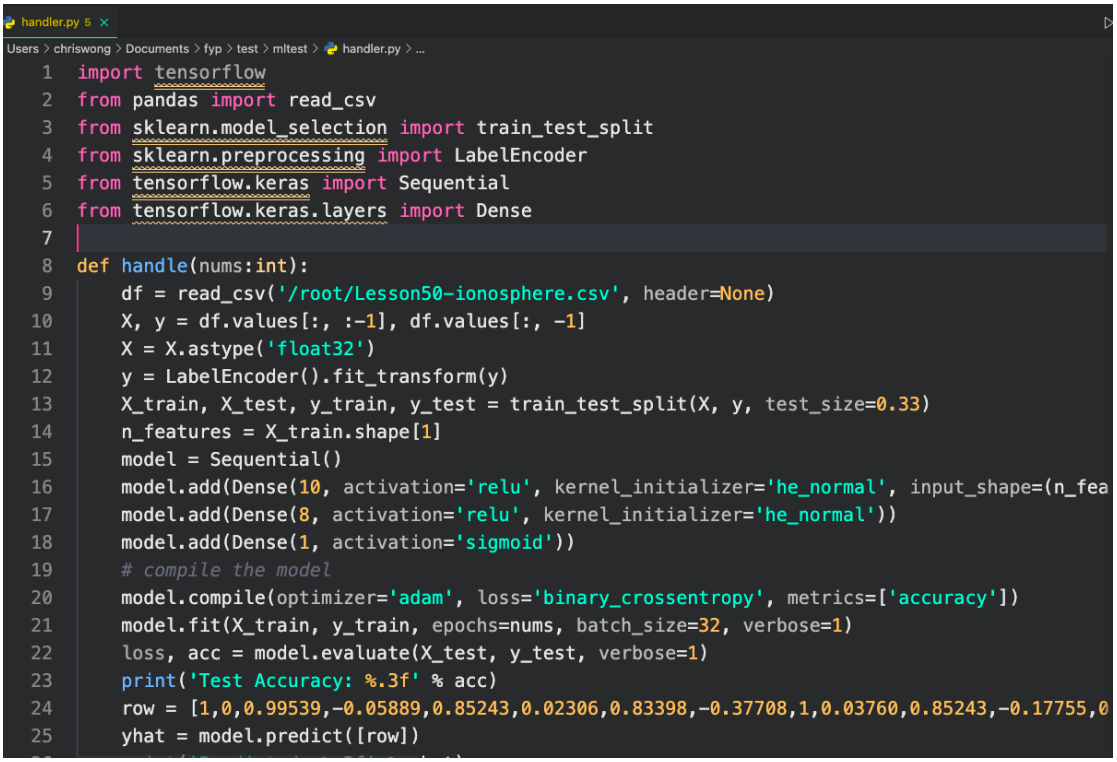

FIGURE 4.21: Function code in OpenFaaS folder

index.py

handler.py 5 index.py

Users > chriswong > Documents > fyp > test > template > python3-debian > index.py > ...

```
1  # Copyright (c) Alex Ellis 2017. All rights reserved.
2  # Licensed under the MIT license. See LICENSE file in the project root for full license infor
3
4  import sys
5  from function import handler
6  import os
7  def get_stdin():
8      buf = ""
9      while(True):
10         line = sys.stdin.readline()
11         buf += line
12         if line=="":
13             break
14     return buf
15
16 if(__name__ == "__main__"):
17     apc = get_stdin()
18     nums = int(apc)
19     handler.handle(nums)
20     #ret = handler.handle(st)
21     #if ret != None:
22         #print(ret)
```

FIGURE 4.22: Main code in template folder

In our model, we write function code in "handler". we want users choice how much training data set they want. So we change the main code to make the program read the users input.

In this subsection, there are other two question developers should pay attention. The first is developers can not add copy training data to container directly. The better is write the copy command in dockerfile. Then modify the user permission to root. The second is developers also need to care about the default docker operate system of the OpenFaaS. Usually, docker just use the kernel of the host. But OpenFaaS need a interface to invoke docker. Then the docker will be add a base operate system and the default operate system is alpine. Mostly, developer do not need to care about this, because OpenFaaS develop many program language. However, we deploy "tensorflow" in our model. From the searching, we learn that the Tensorflow is not supported in alpine operate system. Therefore, after we choice a programming language, we need to specify the base operate system image for docker. And here we choose "debian" because Tensorflow can be great supported by it.

4.2.4 OpenFaaS setting

In the final step, OpenFaaS needs the interface to expose it service and add some configuration files to complete the service.

4.2.4.1 YML file

Developers will find two YML files. One is in OpenFaaS folder the other is in template folder. Same as before, the file in template relate to docker. This file tells the program which programming language template should be used and which file is main code (Fig.4.23).

```
1 language: python3-debian
2 fprocess: python3 index.py
```

FIGURE 4.23: YML template

Then the other YML file in OpenFaaS means network configure. It define the pod name and where is the configuration file. Mostly, the default image hub is docker hub. It specify the address where user upload and pull images. But the docker hub has many restriction, we use "Aliyun" cloud image server here. Developer can also specify many other parameter here such as function time out of gateway and ask Kubernetes use local images. We will give developers details about this part later (Fig.4.24).

```
version: 1.0
provider:
  name: openfaas
  gateway: http://127.0.0.1:8080
functions:
  mltest:
    lang: python3-debian
    handler: ./mltest
    image: registry.cn-shenzhen.aliyuncs.com/whclab/whc1:va
```

FIGURE 4.24: YML in OpenFaaS

4.2.4.2 Requirement file

We mentioned that the program need some dependency package to realize function. Here developers can add the dependency package. In dockerfile, developers can write install code to install these packages which have wrote in requirement file (Fig.4.25).

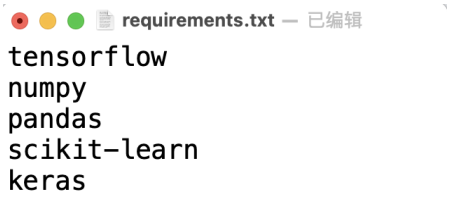

FIGURE 4.25: Requirement file

4.2.5 Function deployment process

After we introduce components, it is easy to deploy function in OpenFaaS now. After developers install all components, the first step is login OpenFaaS gateway. Then use port forward command to expose port and set users' name and password. The next step is pull available template of OpenFaaS. After writing function code and set all parameters, developers can use command of "faas-cli up" to deploy all service automatically. But, developers need to know, this command integrate 3 command: "faas-cli build", "faas-cli push" and "faas-cli deploy".(They all base on docker build push and deploy. This is the reason why developers will find a folder named "build" in OpenFaaS folder.) These 3 command means The OpenFaaS will build program according to developers' folder. Then it push it to address which they have specified(We will give developers more strategy details about using local images in chapter 6.1). Finally, it will deploy the program to the OpenFaaS platform. Here is our running result of running our deployment on OpenFaaS: (Fig.4.26 Fig.4.27)

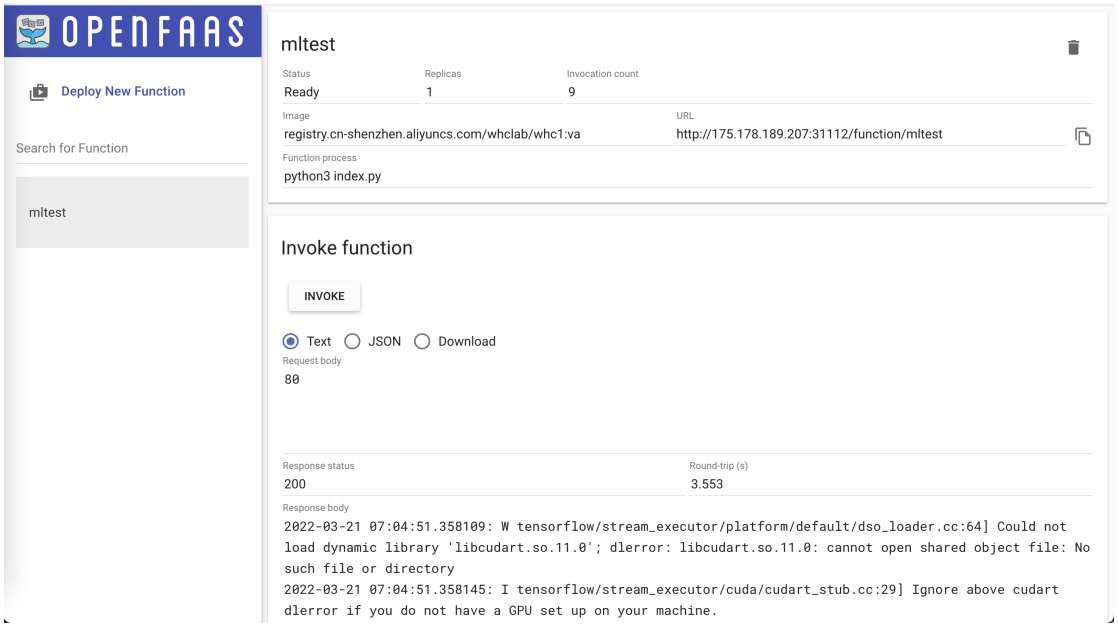

FIGURE 4.26: Deployment result 1

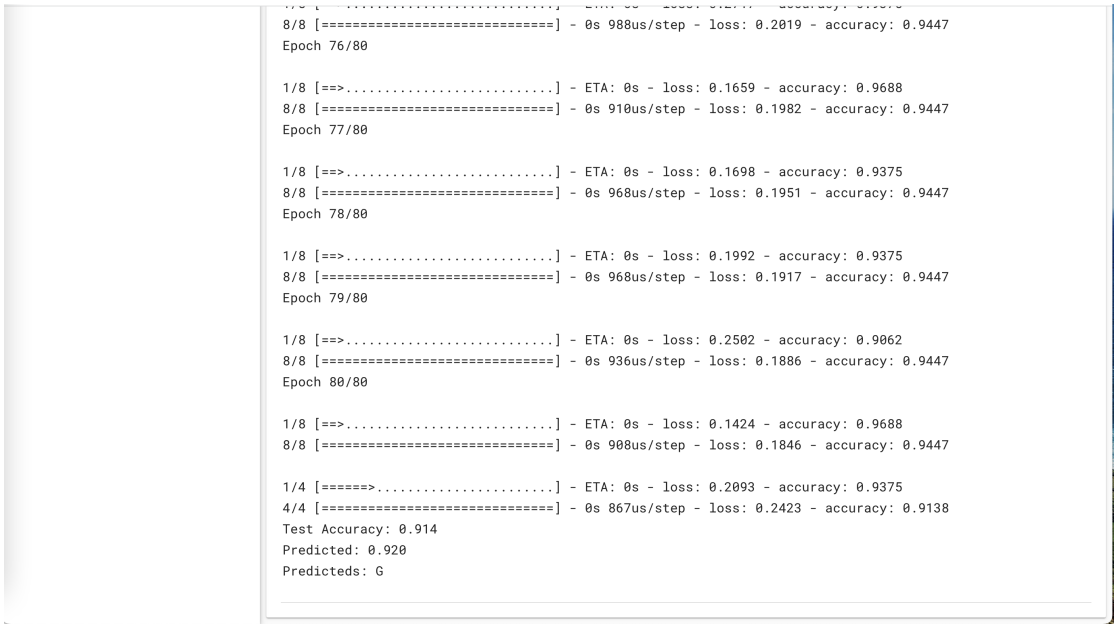

FIGURE 4.27: Deployment result 2

Chapter 5

Executing Process Analysis

5.1 Why performance analysis

The aim to develop OpenFaaS is proving smart and flexible service to user. It is important to developers to analysis performance because it will save much cost for company. Then, analyzing performance of OpenFaaS is necessary for developers. Usually, we can check running logs to analyze performance. But before that, we need to know the running process of the function from start to finish.

5.2 Function running process

This part we are going to give details about how API invoke developers' code (Fig.5.1).

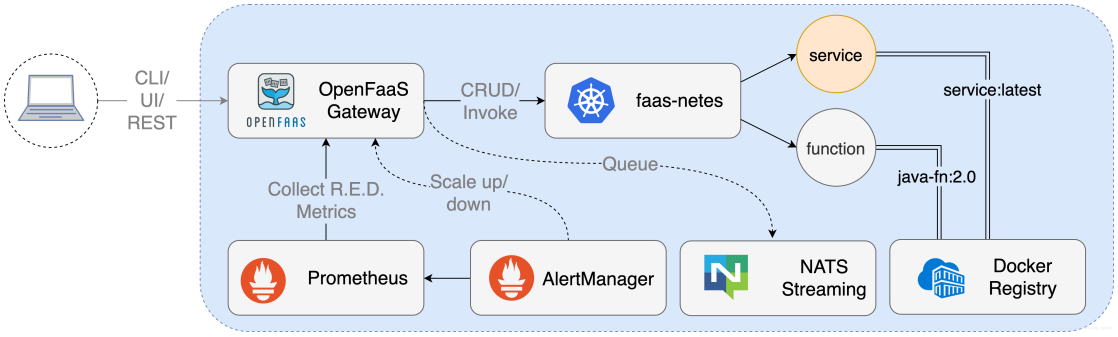

FIGURE 5.1: OpenFaaS service on Kubernetes [6]

From the image developers notice that "faas-netes" provides service in the environment of Kubernetes. How faas-netes handler function? There is an image to explain it: (Fig.5.2)

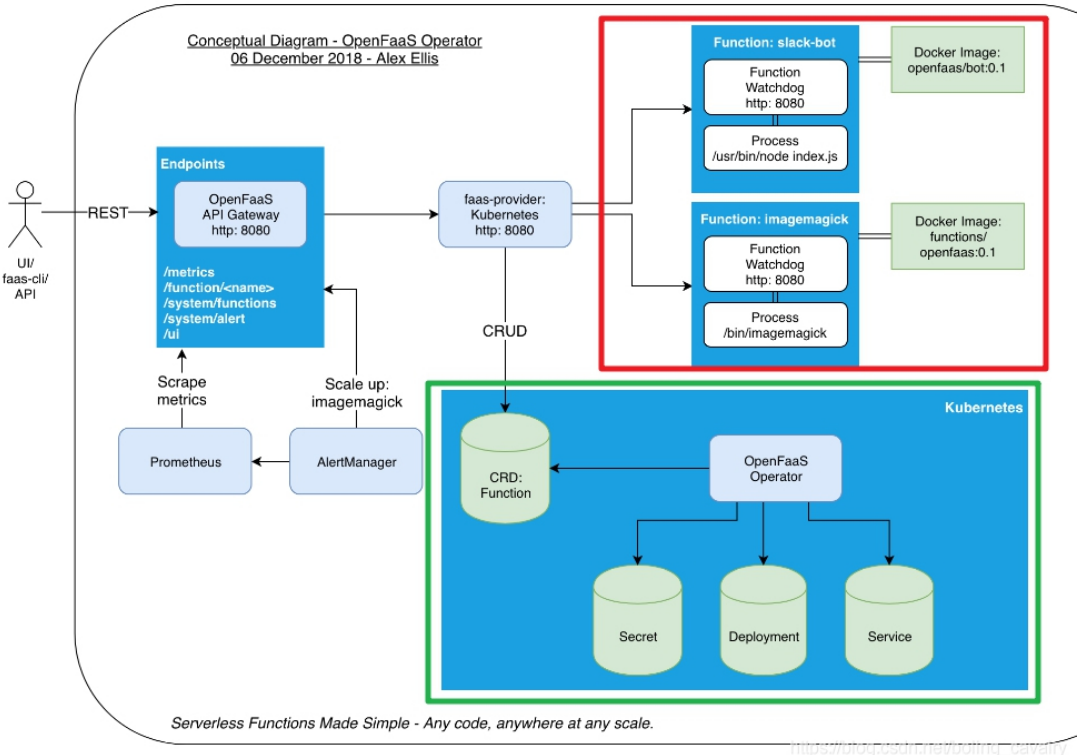

FIGURE 5.2: FaaS-netes [7]

From the image, developers will notice that the uses' request will translate to the port of "OpenFaaS API Gateway". If the function has been published, it will be finished in the process of green frame. If it is a new function, it will be handled with red frame. Now we focus on red frame. The upper blue frame will create a file according to chosen template. Then the file will be translated to the port of 8080 in watchdog and the watchdog will create a process according to file. In another way to explain it. We mentioned that, when developers use faas-cli, it will create a "build" folder automatically. This is a docker image which contain watchdog. The watchdog monitor port of 8080. When it receive request, it will do a fork and transform the parameter to forked process. When the process has been finished the result to the watchdog (Fig.5.3) [30].

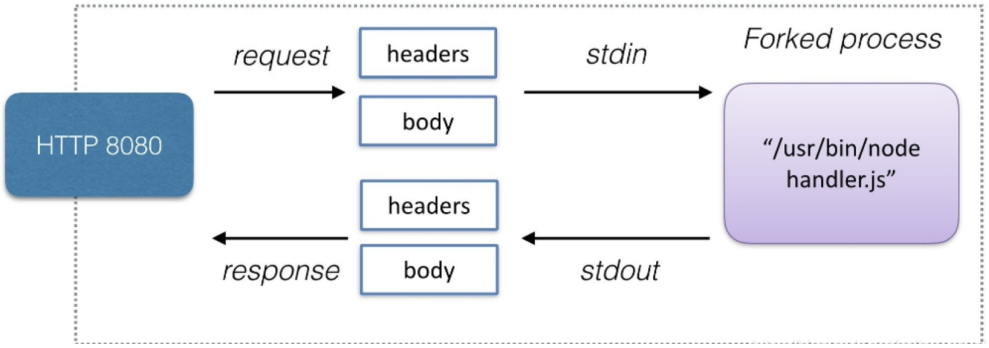

FIGURE 5.3: Watchdog [8]

If developers want to verify it, they can see the dockerfile. The watchdog is basic image.When the container start, watchdog will execute (Fig.5.4).

```
1 FROM --platform=${TARGETPLATFORM:-linux/amd64} ghcr.io/openfaas/classic-watchdog:0.2.0 as watchdog
2 FROM --platform=${TARGETPLATFORM:-linux/amd64} python:3
3
4 ARG TARGETPLATFORM
5 ARG BUILDPLATFORM
```

FIGURE 5.4: Watchdog code

5.3 Analysis of Kubernetes

In Chapter 4.2.2, we have given a general introduction to Kubernetes. In this section, we will introduce more details and give a experiment to verify it.

5.3.1 Kubernetes data process

In last paragraph, we mentioned that Kubernetes is one of the components of OpenFaaS. However, Kubernetes is also a completed system. Developers will find all components function in Kubernetes official website. In summary, after developers have deployed the Kubernetes, there is a completed cluster. Then we are going to introduce all components in Kubernetes. In the beginning, here is a summary image of Kubernetes' components: (Fig.5.5)

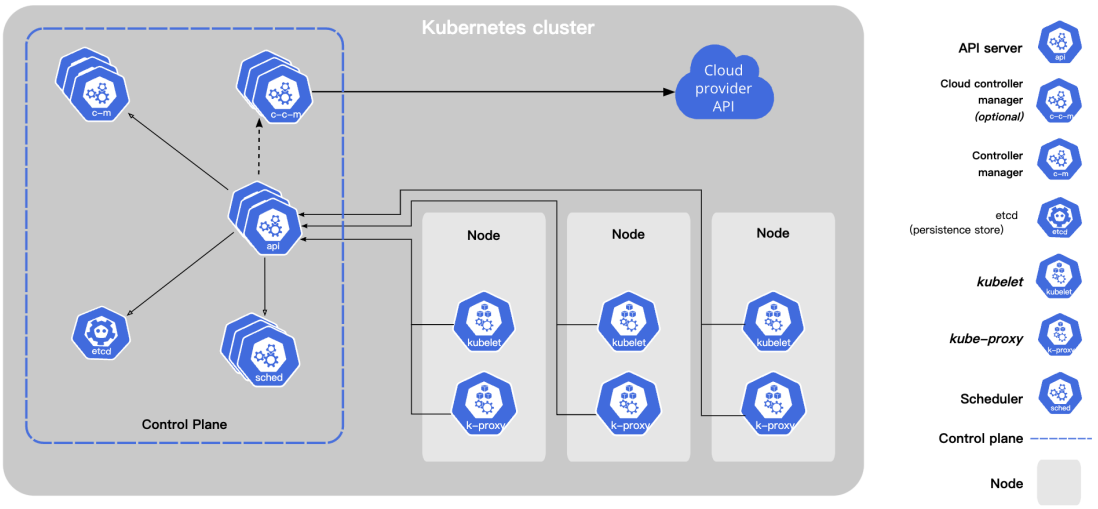

FIGURE 5.5: Kubernetes components [9]

In Kubernetes official website, it divide Kubernetes to 3 components:

5.3.1.1 Control plane components

From image, there is a blue rectangle. It means control plane contains 5 components. Control plane contains can run in any node in cluster. It makes global decisions about the cluster. Besides, it detect and respond to cluster events.

1) Kube-apiserver:

   In any system, it need provide an API to developers to invoke system. The Kube-apiserver realize this function. It exposes and define interfaces and developers will use other tools to connect Kube-apiserver.

2) Etcd:

   Etcd is a database for Kubernetes. It can collect save and backup data.

3) Kube-scheduler :

   Kube-scheduler according to resource requirement and hardware or software restrict to create and monitor pods. Then make pods running on nodes.

4) Kube-controller-manager :

   Kube-controler-manager includes node-controller, job controller, end points controller and service account & token controllers. Each controller is a separate process. It control pods which are running.

5) Cloud-controller-manager :

   Cloud-controller-manager include node controller, route controller and service controller. Its principle is similar to Kube-controller-manager. The cloud-controller-manager makes system connect to the cloud serer API. According to it, developers can control system by cloud server interface.

5.3.1.2 Control components

Node control components run on every node, maintaining running pods and providing the Kubernetes run environment. Node components contains 2 part.

1) Kubelet:

   As we know, Kubernetes is a control system which manages containers. There is one or more containers running in pods. The kubelet send command and manage these containers which created by Kubernetes.

2) Kube-proxy:

   The function of Kube-proxy is as his name saying. the Kube-proxy is the internet proxy in each node. There are many pods in one node. these node need connect to each other. And node needs to connect other nodes or cluster. Kube-proxy provide internet configuration to the node, helping it realize communication function.

5.3.1.3 Addons

Kubernetes in addition to the necessary support components, many other function need to be realized by Kuberneters addons such as Kubernetes DNS, Kubernetes dashboard and so on.

Kubernetes DNS provides DNS service to each pods. Generally, pods will succeed DNS from nodes. Kubernetes DNS service can modify each pods DNS. There is another addon named ingress. Generally, if developers want expose service to internet, they need to expose node. But ingress addon can only expose a single service to internet.

5.3.2 Kubernetes object

What is Kubernetes object? In summary, it is a persistent entity to express the ideal state of cluster. It just like a text file which records specific configuration of cluster. And cluster will try to realize this text file. Almost every Kubernetes object contains two nested object fields. One is object "spec", the other is object "status". In the object "spec", it describes the basic configuration of cluster when it is created. In object "status", it describes the current status, and the system will try to adjustment status to reach the ideal status.

There is a ".yaml" text file records the description of the object. When kubectl want to invoke this description file, it will transform it to JSON file, in according to adapted to API. Developer can apply it by command:"kubectl apply -f (name).ymal".

5.3.3 Verification

As we mentioned before, Kubernetes will manage the containers which are created by itself. In cluster, Kubernetes makes pods to achieve requirement as much as possible. In ".yaml" file, we have set the number of pods is 2 in the node. Here developers can notice that there are two pods have been deployed (Fig.5.6).

```
root@VM-8-10-ubuntu:~# kubectl get pods -n openfaas-fn
NAME                          READY   STATUS    RESTARTS   AGE
mltest-demo-69f8f88ff9-2t7sq   1/1     Running   0          6d4h
mltest-demo-69f8f88ff9-95qw4   1/1     Running   1          6d4h
root@VM-8-10-ubuntu:~#
```

FIGURE 5.6: Deployed pods

In next step, we will delete the pods but not modify the number of pods in "YML" file. Because Kubernetes will deploy pods according to configuration file, developers will notice that the number of pods will be 2 again (Fig.5.7).

```
root@VM-8-10-ubuntu:~# kubectl delete pod mltest-demo-69f8f88ff9-95qw4 -n openfa
as-fn
pod "mltest-demo-69f8f88ff9-95qw4" deleted

root@VM-8-10-ubuntu:~# kubectl get pods -n openfaas-fn
NAME                          READY   STATUS    RESTARTS   AGE
mltest-demo-69f8f88ff9-2t7sq   1/1     Running   0          6d4h
mltest-demo-69f8f88ff9-cksgr   1/1     Running   0          39s
```

FIGURE 5.7: Delete pod result

Developers will notice that the second pod name is changed and the creation time is earlier, so the theory established. If developers want delete deployment, they can use "Kubectl delete deployment" or "faas-cli delete" to realize(Openfaas is in a deeper level, so both "kubectl" and "faas-cli" can realize it).

5.4 Analysis of OpenFaaS

5.4.1 Timeout concept

In the beginning, we want to introduce a concept named timeout. This concept is mostly used in many network protocol such as TCP/IP. This protocol avoid network congestion and avoid crash. For example, there is a request lost cost much time and the server wait for it. It will cause the server can not process other request. Timeout mechanism will give up the request which is timeout to make network system run successfully.

Then, in OpenFaaS system, developers also need the timeout to deal the error status. They can can set "exec_timeout, write_timeout and read_timeout". Here we will show developers details about these.

From the http network transmission, we will notice that read timeout and write timeout. Literally, read and write timeout belong to transmission phase and exec timeout belongs to program execution. Because we just focus on timeout, we are going to introduce these concept (Fig.5.8).

1) Request headers:
   Request headers contains many attributes about the request.The common request headers are accept, cookie, reference and Cache-Control. They tell terminal what attribution about the request.

2) Request body:
   Request body contains the content about what the request want to send to terminal.

3) Response:
   Response is the message about feedback. The response usually is result.

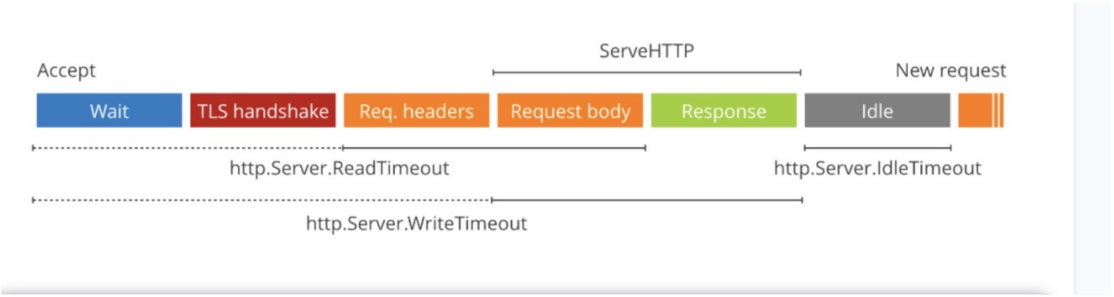

FIGURE 5.8: Http network transmission

From the image, we notice that read_Timeout contain request headers and request body, and write_Timeout contains request body and response. But where is execution phase? It begin location is between request body and response and end location is the end of response (Fig.5.9).

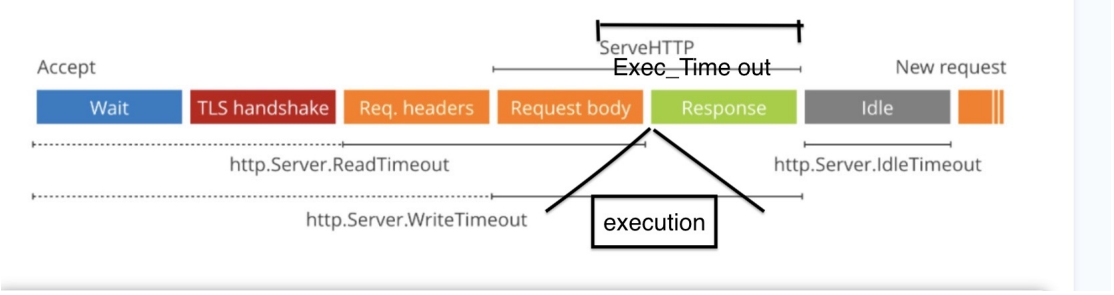

FIGURE 5.9: Execution Timeout

The execution timeout contains container execution time and the time about it transport execution value to transmission phase.

5.4.2 Analysis with logs

Kubernetes provide "kubectl" which is the official command tool to developers. Firstly, developers can check "description" of the pods. When developers fail to deploy the function, they can find basic information it may hint where has problem (Fig.5.10).

FIGURE 5.10: Pod description

Then, if developers want to check execution time details. they can use "kubectl logs" to check it (Fig.5.11).

```
root@VM-8-10-ubuntu:~# kubectl logs -n openfaas-fn mltest-dev-5fc47f9755-td5fz
2022/03/23 09:24:32 Version: 0.2.0      SHA: 56bf6aac54deb3863a690f5fc03a2a38e7d9e6e
f
2022/03/23 09:24:32 Timeouts: read: 5s write: 8s hard: 8s health: 8s.
2022/03/23 09:24:32 Listening on port: 8080
2022/03/23 09:24:32 Writing lock-file to: /tmp/.lock
2022/03/23 09:24:32 Metrics listening on port: 8081
2022/03/23 09:24:51 Forking fprocess.
2022/03/23 09:24:58 Wrote 14605 Bytes - Duration: 6.754017s
```

FIGURE 5.11: Pod logs

5.4.3 Testing timeout problem

Now, we have known the timeout mechanism and how to check execution time [31]. We are going to test it. Here we wrote a timeout code function in "handler.py" (Fig.5.12).

```python
from time import sleep
import os

def handle(nums:int):
    sleep_duration = int(os.getenv("sleep_duration"))
    preSleep = "Starting to sleep for %d" % sleep_duration
    sleep(sleep_duration)  # Sleep for a number of seconds
    postSleep = "Finished the sleep"
```

FIGURE 5.12: Sleep code function

Then we can define sleep time and invoke function in YML file. We design our sleep time in the execution phase, so it will affect write and exec_timeout (Fig.5.13).

```
environment:
  sleep_duration: 3
  read_timeout: "5s"
  write_timeout: "8s"
  exec_timeout: "8s"
```

FIGURE 5.13: Sleep and Timeout setting

Next part we are going to verify our guess. The first step, we should know the time of our function execution time when we use 50 set of data to test, without sleeping (Fig.5.14).

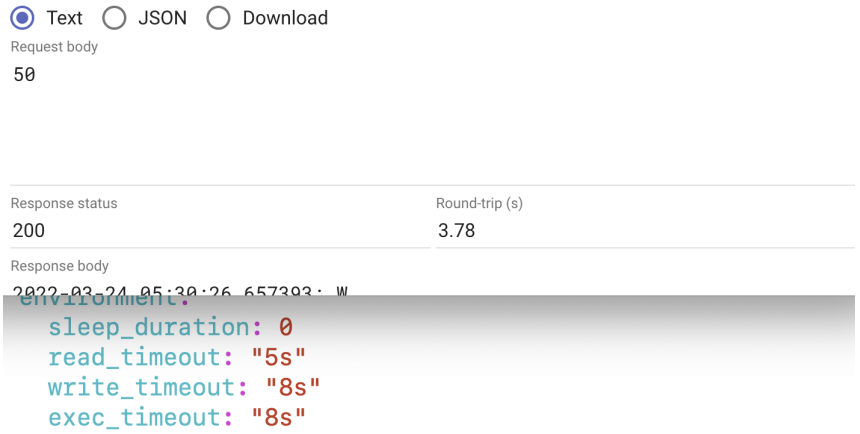

FIGURE 5.14: Time without sleeping

Then, if we set sleep time is 3 second with the same test data: (Fig.5.15)

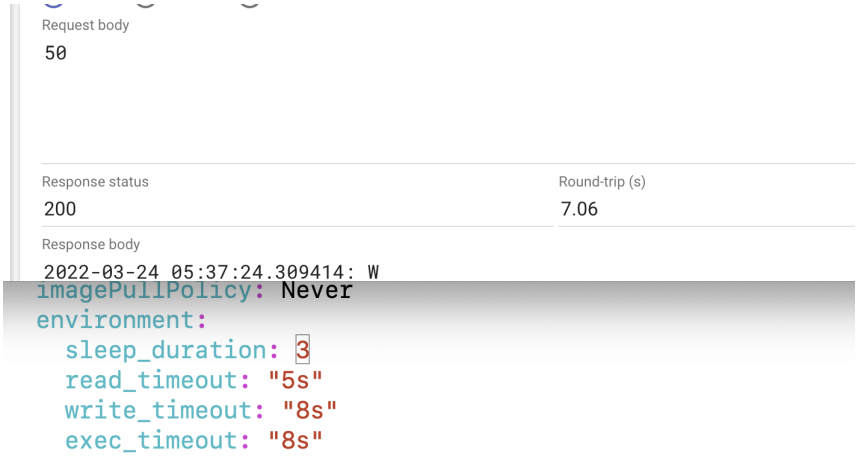

FIGURE 5.15: 3s sleeping time

Comparing these two time, the first time 3.68s add 3s approximately equal to 7.06s. What about change read timeout?

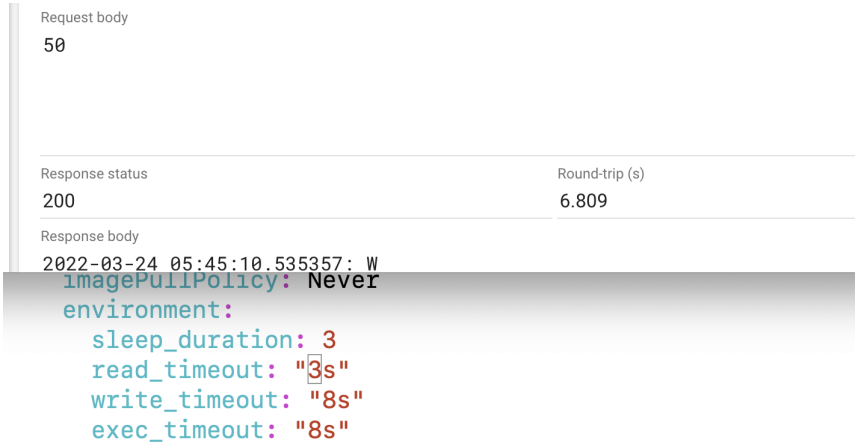

FIGURE 5.16: Change read timeout

From the image we can notice that sleep time will not lead to fail about read timeout. However, what about changing write or execution timeout (Fig.5.17 Fig.5.18)?

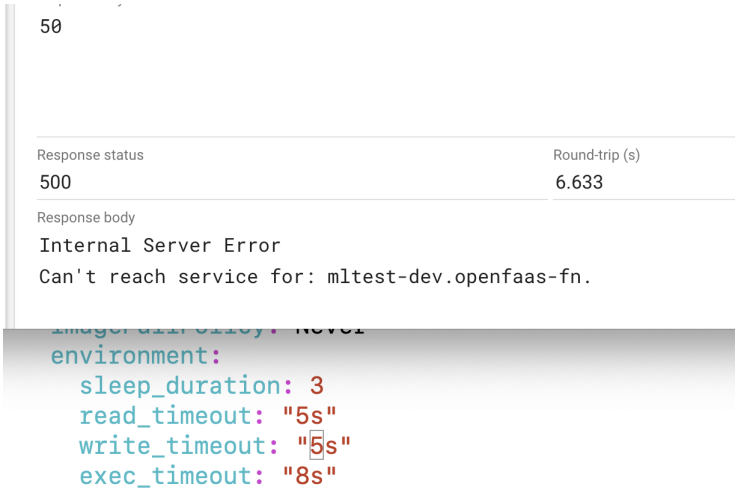

FIGURE 5.17: Change write timeout

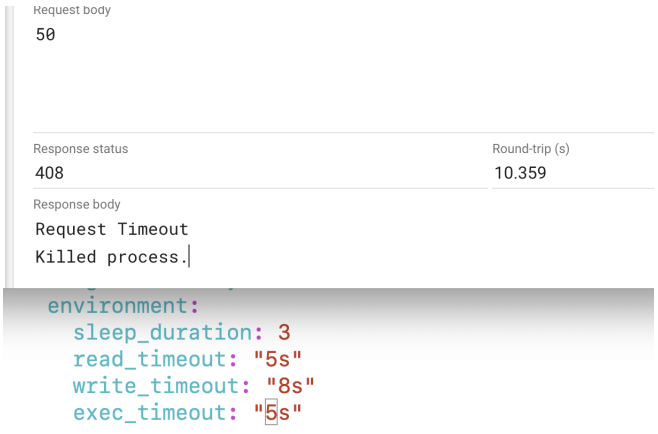

FIGURE 5.18: Change execution timeout

We find that not only shorten write_timeout but also shorten execution time will lead to fail. One is can not get return information and the other is timeout. These confirm our suspicions.

Chapter 6

Performance analysis

In this paragraph, we will do some analysis about performance. In the begin, we are going to introduce basic concept about scaling.

6.1 Concept

6.1.1 Kubernetes scaling strategy

In the Kubernetes official document, developers will notice that it introduce the horizontal pod autoscaler. It is a tool which define rule and control scaling according to CPU load.

6.1.2 OpenFaaS scaling strategy

In Kubernetes, there is a function named scaling [32]. OpenFaaS bases on Kubernetes, and it has the scale characteristic [22]. What is scale? In summary, it expand or reduce the pods in one node according to the certain rule. Scaling is one of the characteristic of OpenFaaS. Comparing to the traditional IaaS or BaaS, OpenFaaS just call system resource when it is invoked but not a service running all the time. Every time the OpenFaaS system is invoked, the pods expand by one. When the pods stop to be invoked, OpenFaaS system will recycle resources. Generally, the rule of scaling is horizontal pod autoscaling in OpenFaaS. Comparing to vertical pod autosacling which will increase the

numbers of the pods but not pods' resources, the horizontal pod autoscaling just increase the number of the pods. The horizontal pod autoscaler is implemented as a Kubernetes API resource and a controller. The resource determines the behavior of the controller. The horizontal pod autoscaling controller, running within the Kubernetes control plane, periodically adjusts the desired scale of its target (for example, a Deployment) to match observed metrics such as average CPU utilization, average memory utilization, or any other custom metric their specify (Fig.6.1).

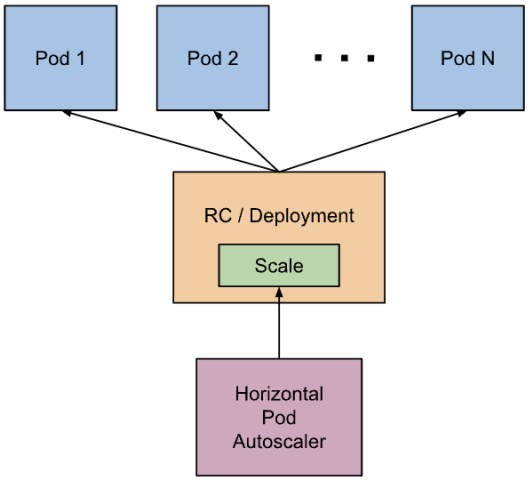

FIGURE 6.1: Horizontal pod auto-scaling [10]

6.1.3 Cold start and warm start

We are going to do an experiment to research cold and warm start [33]. First of all, we are going to introduce what is cold and warm start.

Generally, the concepts of cold and warm are used in applications. However, the concepts are popular in Serverless platform. As we know, if developers want to run a application, they need a PC or mobile phone to execute it. If the computer or mobile phone is closed, the first step is PC startup, then the application will be executed. For another example, if the developers want to run some programs in a new isolated environment, the virtual machine is a good idea. The first step also create a new virtual environment. Then they can install the program to realize it. Now we come back to Serverless platform. We mentioned that the OpenFaaS is dependent on docker. Docker provide containers

to OpenFaaS to run. Exactly, OpenFaaS function is running on docker container. The docker container is similar to PC or virtual machine in Serverless.

6.1.3.1 Cold start

In the condition of the developers want to run a program, but they do not have containers to execute it, they need create a new container and deploy function on it. The process of creating containers deploying function and executing function is cold start. The cold start apply to deploy a new function on container or container is freed because of long time no use.

6.1.3.2 Warm start

As we know, The OpenFaaS platform is elastic what we have introduced scaling before. If the container is long time no used, it will be release. However, the function is used continuously, the container will not be release. In this condition, there is no need to recreate container. This process is warm start. The obvious difference is warm start is more quickly than cold start.

6.2 Analyzing tools

If we want to analyzing performance, the first step is know about the performance analysis tools. From the data which is collected by these tools, we will have clear view of the changing to machine performance.

6.2.1 Prometheus and Grafana

In the beginning, if we want to analysis pods' performance, we need to know how to monitor performance. The tool named Prometheus can help us. Prometheus is a set of open source system monitoring and alarm system. Besides, according to Grafana, it provide many pods' data for developers just like dash board, it has high performance to process large amount of data in many pods. It not rely on distributed storage, a single node can work directly [34]. Here is the initial UI of Prometheus (Fig.6.2).

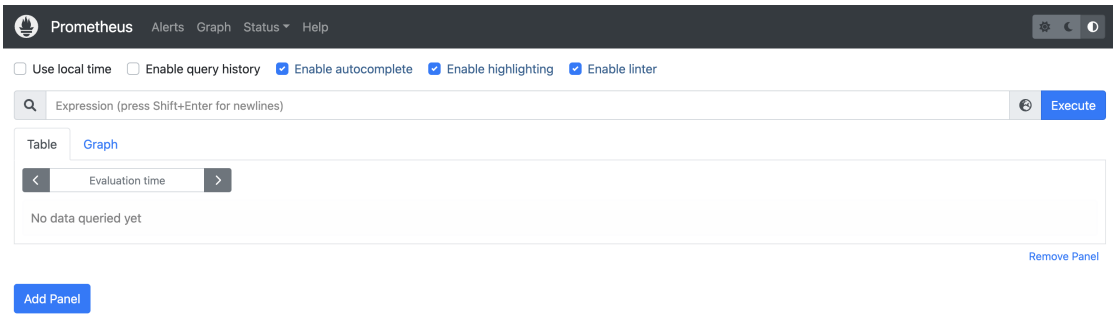

FIGURE 6.2: Prometheus UI

After we have "dashboard", we also need a interface to collect and transport data form node to dashboard. The interface is "Node expoter". The data from node exporter is not friendly for human. Prometheus can transform the data more friendly. Here we will show the data from node export and mount node export to Prometheus (Fig.6.3 Fig.6.4 Fig.6.5).

```
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 2.5147e-05
go_gc_duration_seconds{quantile="0.25"} 8.6142e-05
go_gc_duration_seconds{quantile="0.5"} 0.000113955
go_gc_duration_seconds{quantile="0.75"} 0.000141607
go_gc_duration_seconds{quantile="1"} 0.055541817
go_gc_duration_seconds_sum 2.62604796
go_gc_duration_seconds_count 9305
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 8
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.17.3"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 4.646984e+06
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 2.1247693832e+10
# HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket hash table.
# TYPE go_memstats_buck_hash_sys_bytes gauge
go_memstats_buck_hash_sys_bytes 1.857783e+06
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 2.12412658e+08
# HELP go_memstats_gc_cpu_fraction The fraction of this program's available CPU time used by the GC since the program started.
# TYPE go_memstats_gc_cpu_fraction gauge
go_memstats_gc_cpu_fraction 0.00010513481743484654
# HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system metadata.
# TYPE go_memstats_gc_sys_bytes gauge
go_memstats_gc_sys_bytes 5.349128e+06
# HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and still in use.
# TYPE go_memstats_heap_alloc_bytes gauge
go_memstats_heap_alloc_bytes 4.646984e+06
# HELP go_memstats_heap_idle_bytes Number of heap bytes waiting to be used.
# TYPE go_memstats_heap_idle_bytes gauge
go_memstats_heap_idle_bytes 5.48864e+06
# HELP go_memstats_heap_inuse_bytes Number of heap bytes that are in use.
# TYPE go_memstats_heap_inuse_bytes gauge
go_memstats_heap_inuse_bytes 6.47168e+06
# HELP go_memstats_heap_objects Number of allocated objects.
# TYPE go_memstats_heap_objects gauge
go_memstats_heap_objects 29315
# HELP go_memstats_heap_released_bytes Number of heap bytes released to OS.
# TYPE go_memstats_heap_released_bytes gauge
go_memstats_heap_released_bytes 4.349952e+06
# HELP go_memstats_heap_sys_bytes Number of heap bytes obtained from system.
# TYPE go_memstats_heap_sys_bytes gauge
```

FIGURE 6.3: Node export

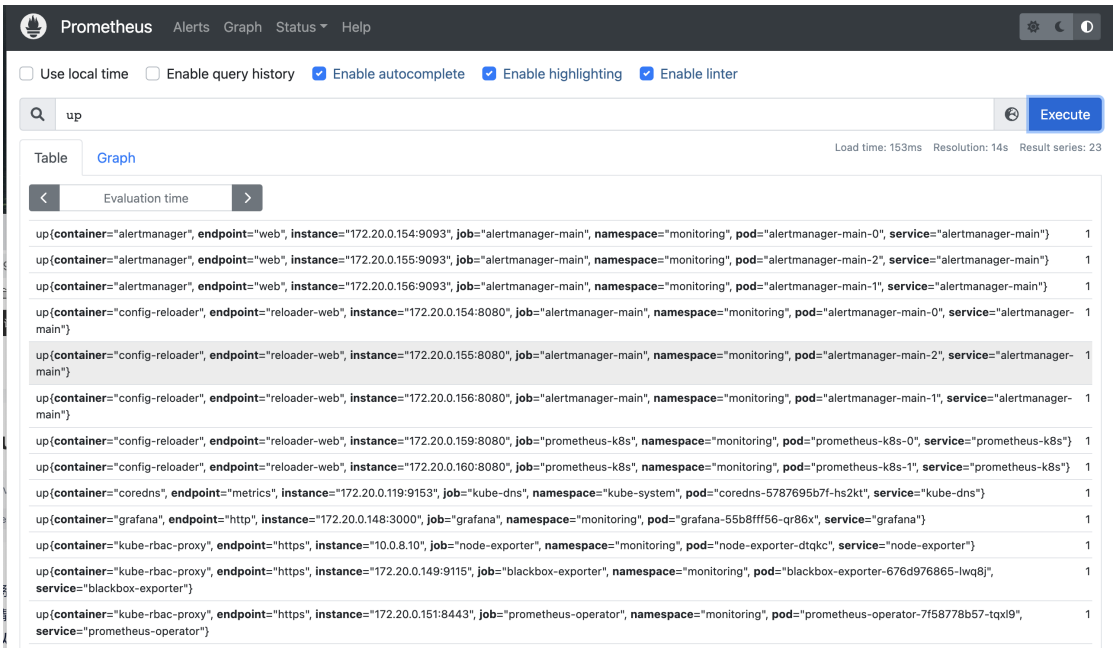

FIGURE 6.4: Node export data on Prometheus

In the next step, we can use Grafana to analysis pod clearly. Developers can import dash board template from Grafana website. After we enter Prometheus IP to Grafana, we can start to watch the status of node(Fig.6.5).

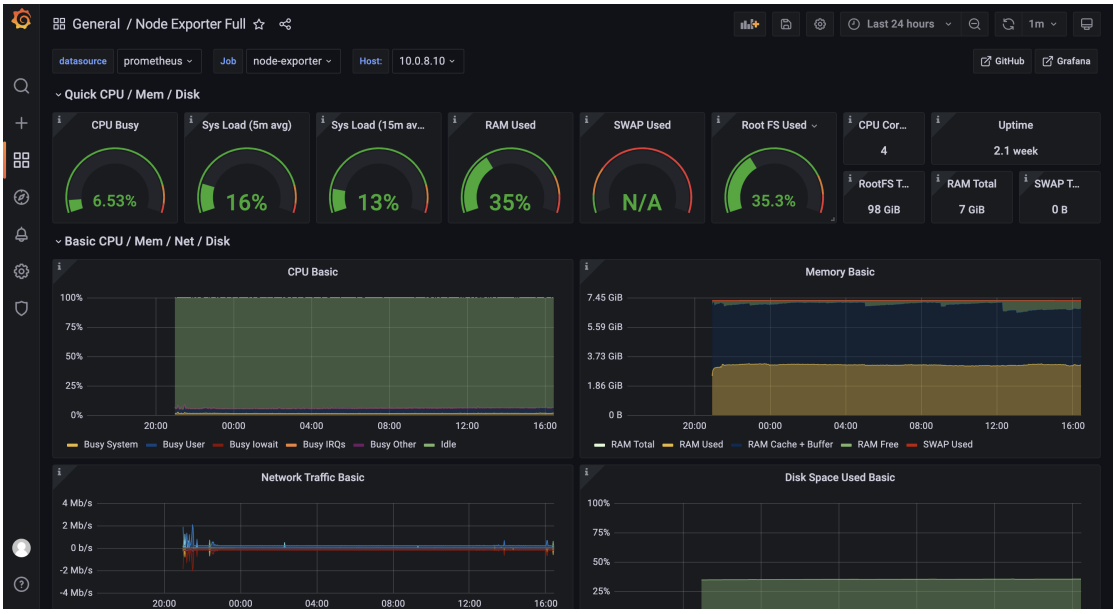

FIGURE 6.5: Grafana dash board

Usually, developers use these tools to monitor host performance. However, we need monitor Kubernetes cluster in this experiment. We install Kubernetes version of Prometheus

and point export to node. In addition, we also set the right interface to each component.

In order to distinguish monitoring tool and OpenFaaS, we use another namespace and install all components in this namespace. Here we want to check Prometheus directly, we also need port forwarding. Because Prometheus interface is intranet, we need forward it and visit it by host IP. But if developers just want to use Grafana but not visit Prometheus directly, there is no need to use Prometheus port forwarding. Just connect Prometheus to Grafana in intranet and forward Grafana service is OK. The port forwarding command is(Fig.6.6):

```
# Forward to deployment
kubectl port-forward deployment/redis-master 6379:6379

# Forward to replicaSet
kubectl port-forward rs/redis-master 6379:6379

# Forward to service
kubectl port-forward svc/redis-master 6379:6379
```

FIGURE 6.6: Port forwarding [11]

Finally, we can check the service again(Fig.6.7):

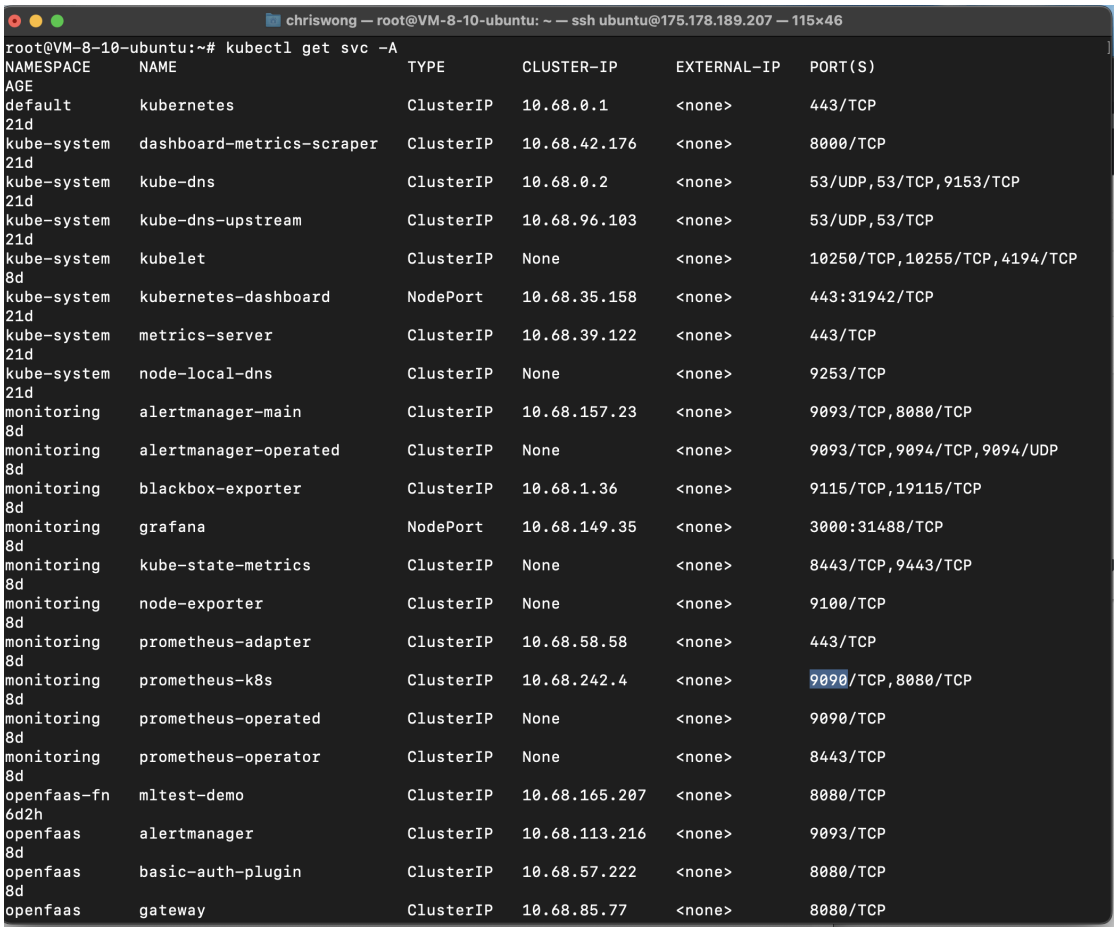

FIGURE 6.7: Check service

In the end of this paragraph, we are glad to summarize how these tools make up to do analysing. Developers will see all the components which we introduce before in this image (Fig.6.8):

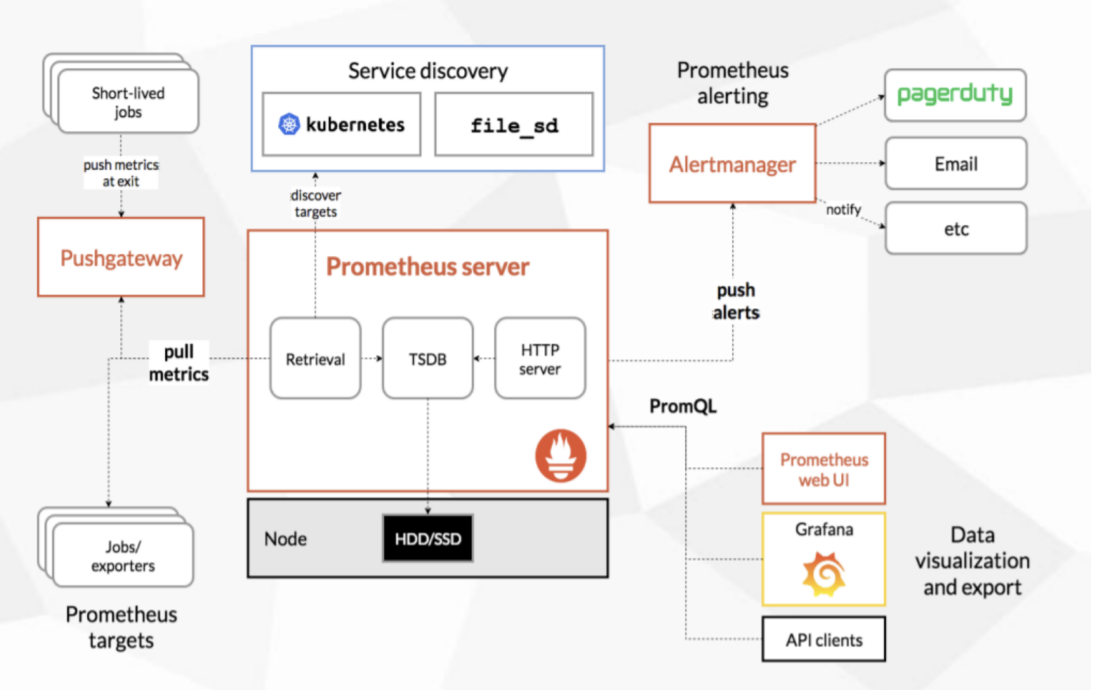

FIGURE 6.8: Analysing process [12]

The Figure 6.8 sketches all working component of Prometheus. We summarize the whole process in the following four steps:

1) Prometheus server pull metrics from configured jobs or exporters regularly. Apart from that, it can also receive metrics from Push-gateway.

2) The Prometheus server stores the collected metrics locally and runs the defined "alert.rules" to record new time series or push alerts to Alert-manager.

3) Alert-manager processes the received alerts according to the configuration file and issues alerts.

4) In a graphical interface, visualize the collected data.

6.2.2 Metrics

In section 6.2.1, we install node exporter. It can collect node data from Prometheus and transform them to Grafana. However, what about there are many pods in one node? How to collect the certain pod data? Here we need to introduce the components named metrics. It is important for this experiment because it collect data from Prometheus and

translate data to Grafana. Developers can find more details on Github. In summary, the Prometheus has many exports which can expose it collected data. Generally, Prometheus only has basic metrics to collect data through export. It is not enough for our experiment. Here we install another component named "kube-state-metrics" which aim to collect pod auto-scaling status. The tutorial is easy to find on Github, we are not going to introduce it. After installed, developers can find the components' pods is running (Fig.6.9):

```
monitoring    alertmanager-main-2                  2/2     Running   0     23d
monitoring    blackbox-exporter-676d976865-lwq8j   3/3     Running   0     23d
monitoring    grafana-55b8fff56-qr86x              1/1     Running   0     23d
monitoring    kube-state-metrics-5d6885d89-qpzvw   3/3     Running   0     21h
monitoring    node-exporter-hn4lx                  2/2     Running   0     14d
```

FIGURE 6.9: Kube-state-metrics

Then developers can watch the pod running status on Prometheus and Grafana (Fig.6.10). There are also some precautions, we will introduce them on chapter 7.

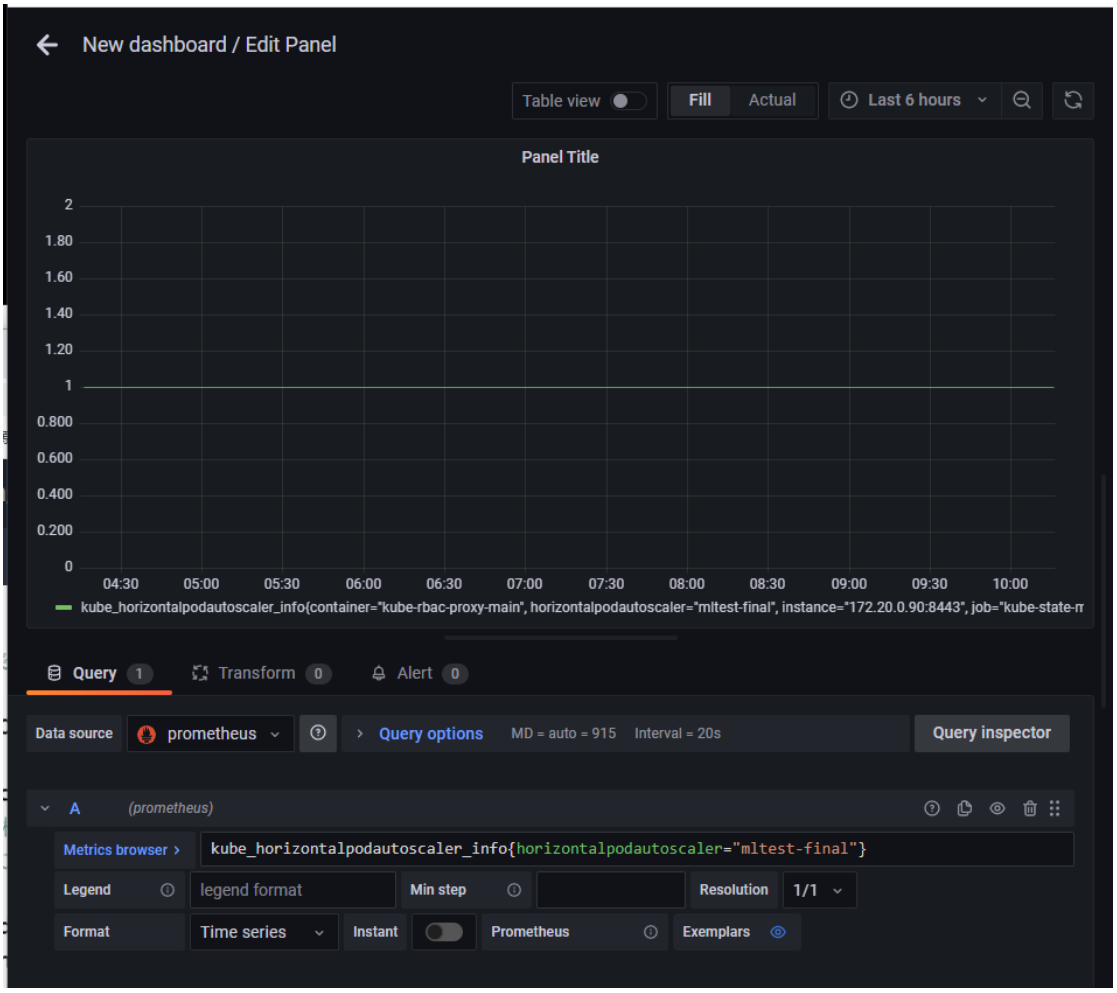

FIGURE 6.10: Pod running status

6.2.3 JMeter

The using of JMeter[1] which is developed by Apache is do a stress testing to web applications. It is based on Java, and now it is used in more fields to give a stress test such as static file, FTP server and so on. Here we need it to simulate a stress test about the function which are widely used by customs in OpenFaaS platform.

After developers installing Java, they can download JMeter in its official website. The initial UI is here (Fig.6.11):

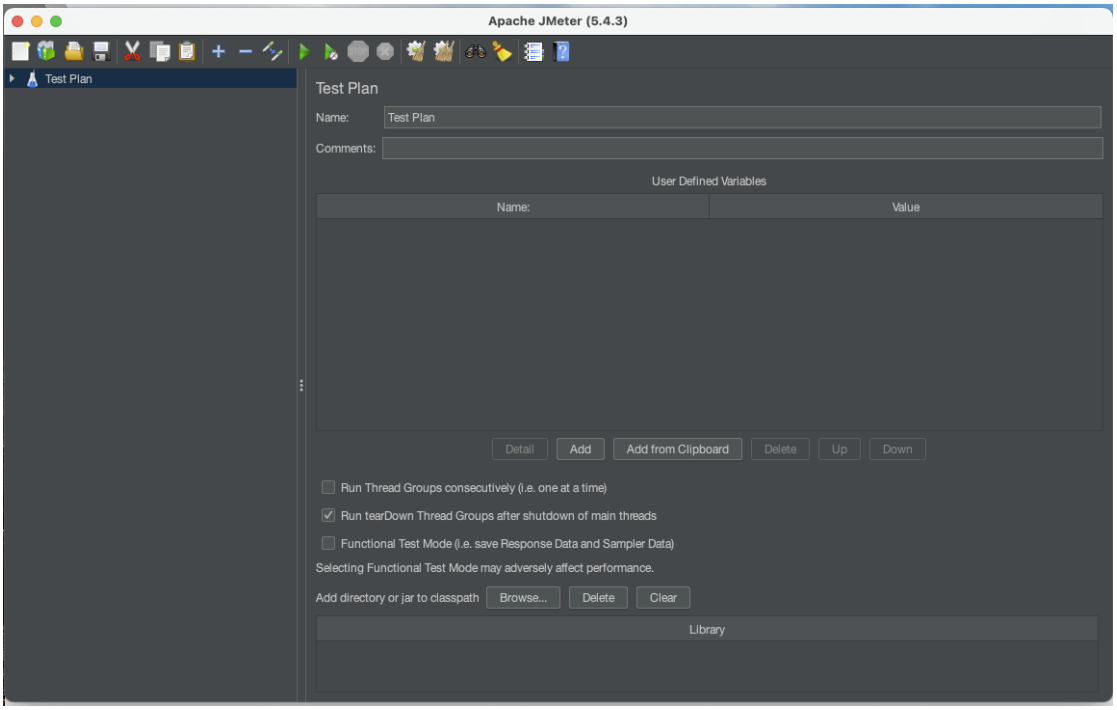

FIGURE 6.11: JMeter initial UI

Firstly, developers need to add a thread group to give a preliminary request parameter to JMeter (Fig.6.12).

[1] JMeter https://jmeter.apache.org/

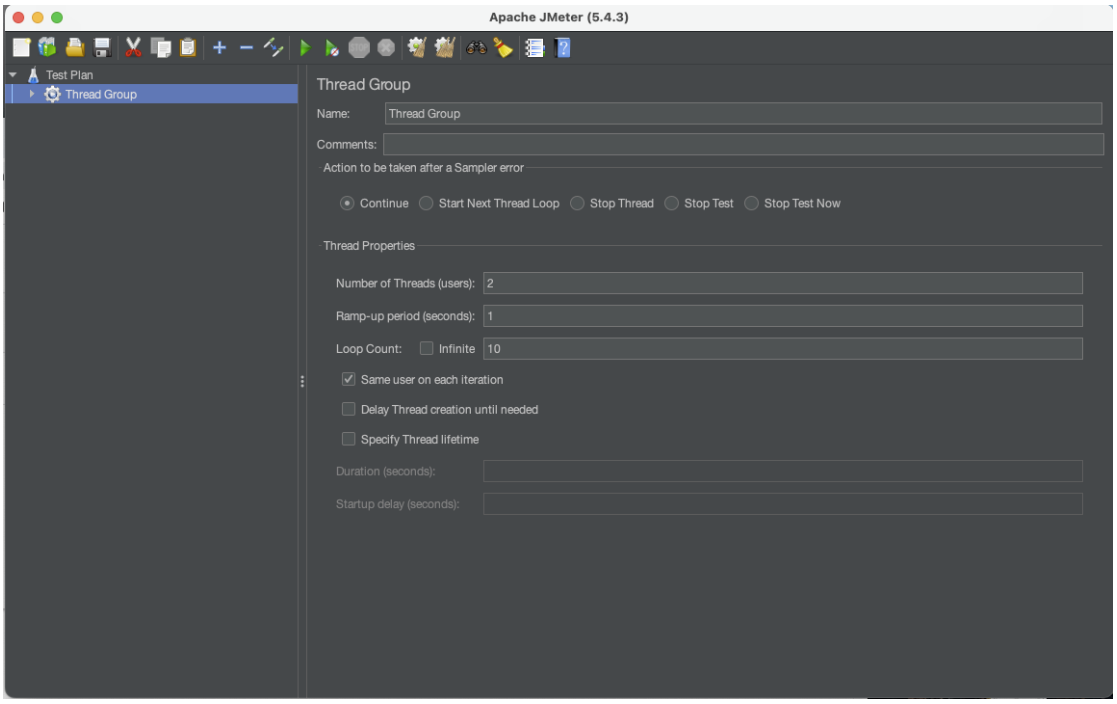

FIGURE 6.12: JMeter thread group

Then the next step developers need to choose a request type. Generally, the beginners may confused about what means about these parameters. Then we will give developers more details about the knowledge about web request.

In the last paragraph, we mention that OpenaFaaS UI. Because we need sent request continuously to web, we can find some information in web. (Fig.6.13)

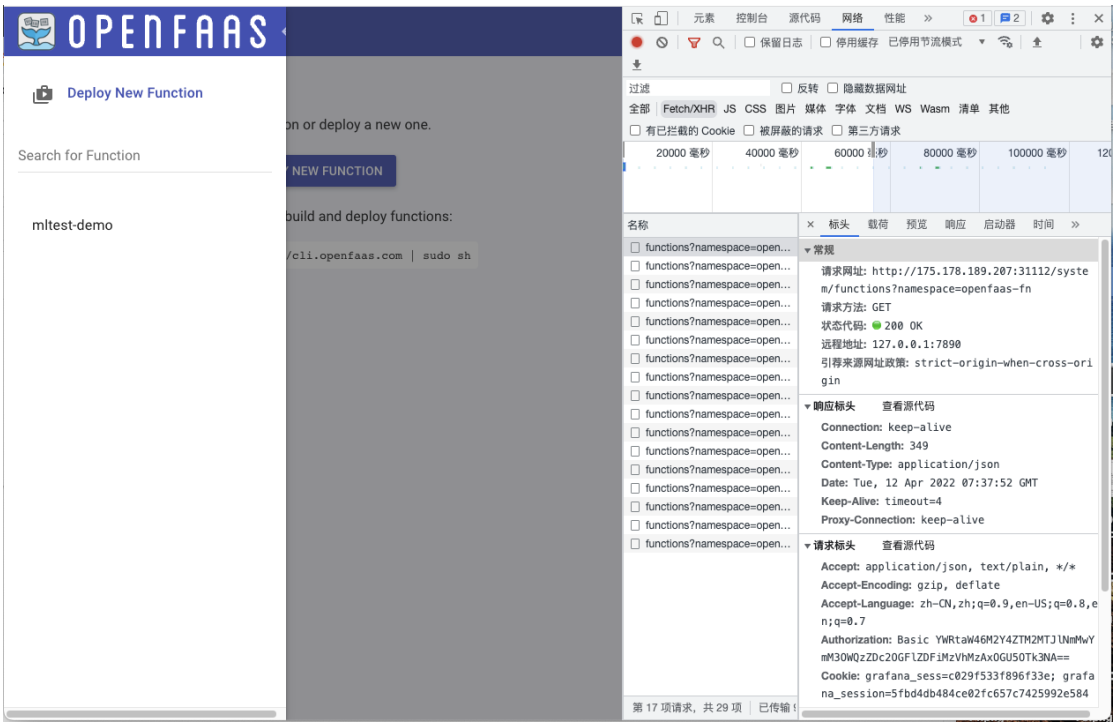

FIGURE 6.13: Web information

Here we choose internet and fetch/XDR to check the response of website. After choosing one of the response, we can find the options named header and select it. From the request header, we can find out what kinds of requests needs in this web. Here developers will notice that the type of request is http, so I choose http in JMeter. Besides, we can find other parameters here, such as username, password and key (request parameter). However, developers can not find these type of parameter here, because we not set up and we can skip it. we introduced our deployed function before. Users need to input a parameter to be the number of machine training set. The parameter is request body here. Then we will input a number to be the request body here (Fig.6.14).

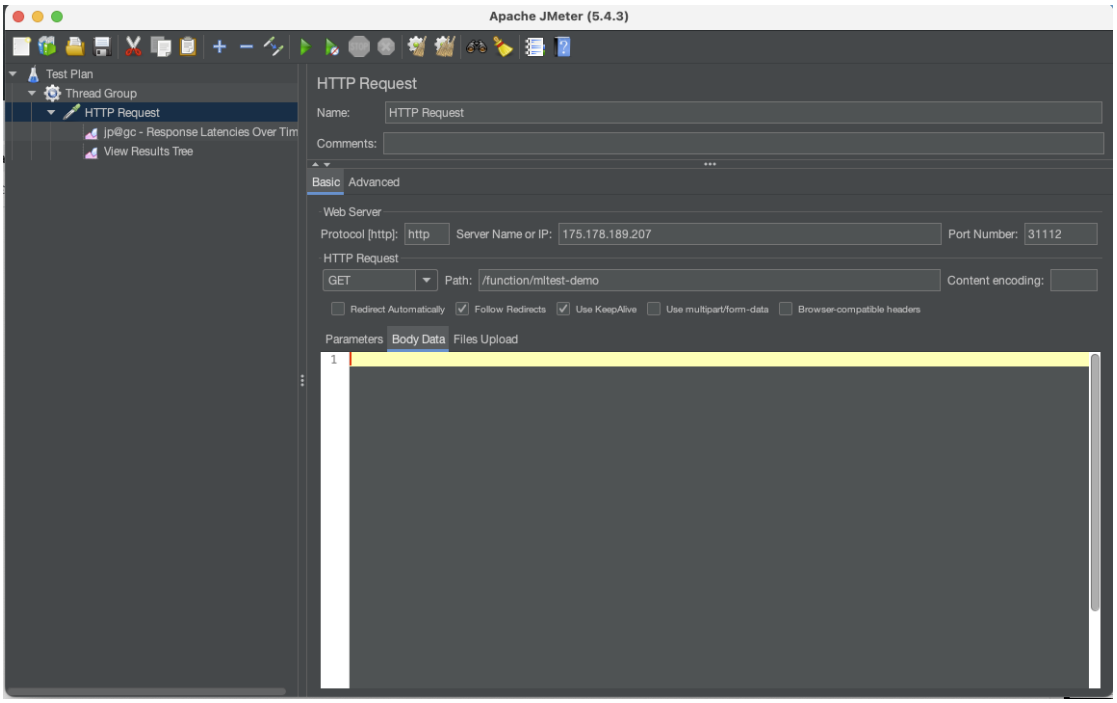

FIGURE 6.14: Http request parameter

After that, developers also need input basic web server parameters. Developers can notice we have finished it here. It is easy to find all of them in web. By the way, we invoke function in local host of OpenFaaS (using node IP). It is important to expose node to host. Because we have introduce it before, we are not going to talk about it here. Then developers can click green triangle to run JMeter. After finishing it, developers can use monitor named "View Result Tree" to check the details about request process (Fig.6.15).

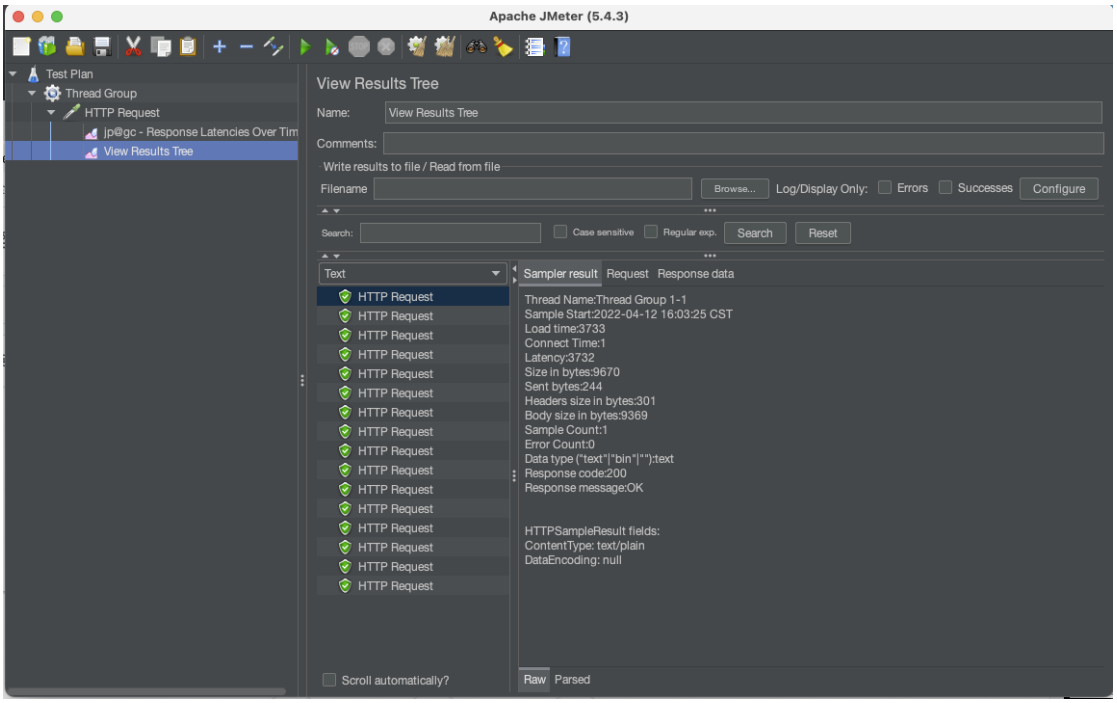

FIGURE 6.15: Request process

Because the default monitor tools is not enough, developers need download plugins managers in official website. There are many monitor there. We install response delay here. It can monitor the time between JMeter send request and JMeter receiver response.

6.2.4 cAdvisor

Developers may confuse about why we need `cAdvisor` except Granafa and Prometheus. cAdvisor is a monitor tool which is dedicated to monitoring node. Comparing to Prometheus which monitor host, it is more suitable for our experiment because there is no interference about the host. We can get more precise data about node resource. Besides, Grafana can also collect data from cAdvisor. Generally, we are not used to using Grafana to watch data from cAdvisor because cAdvisor has provided graph UI to users. Then, we do not prepare to introduce how to install it. It depends on docker. It is easy to pull image from docker hub. Here is UI of cAdvisor. (Fig.6.16)

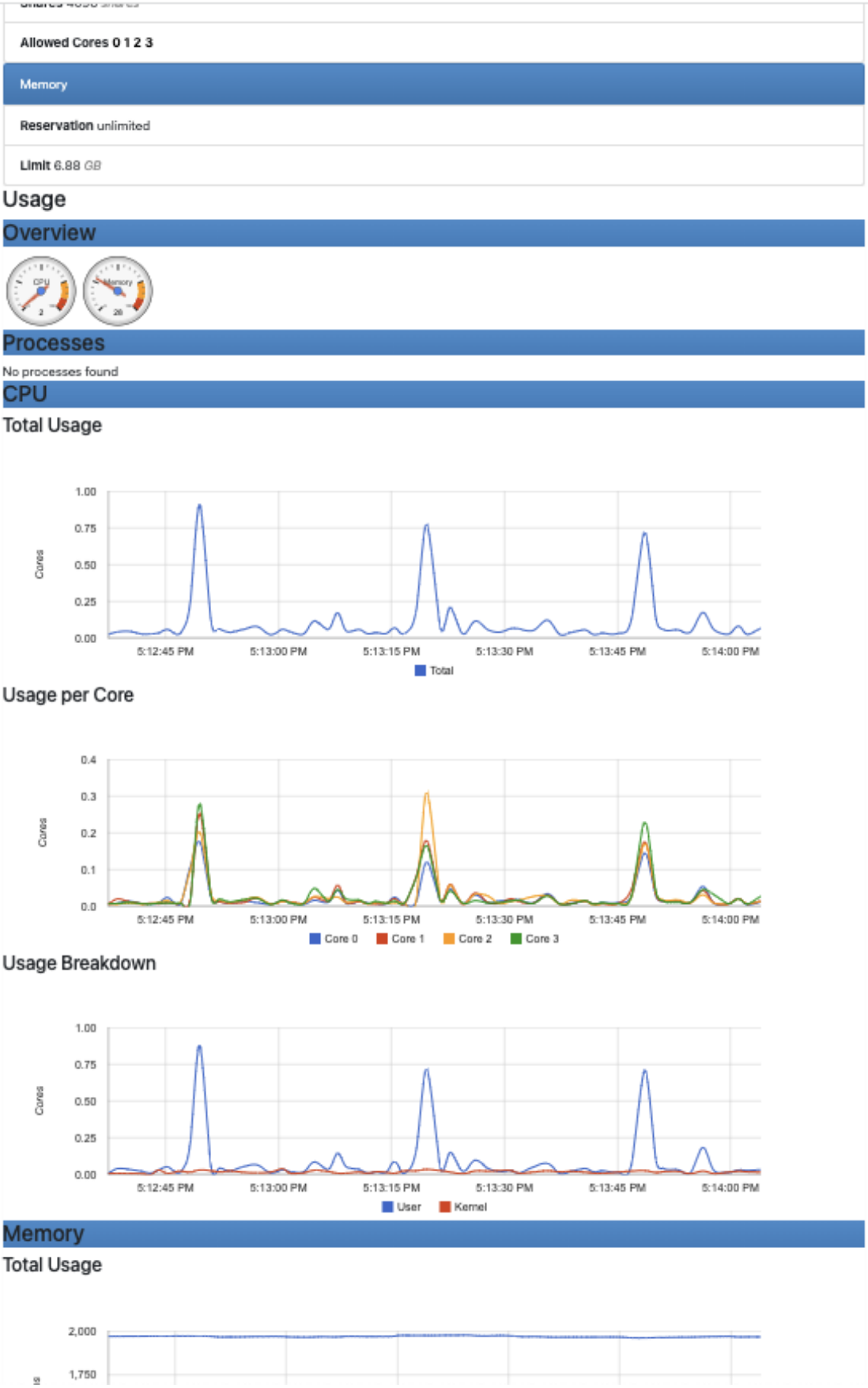

FIGURE 6.16: cAdviosr UI

6.3 Experiment

As we know, OpenFaaS bases on Kubernetes. Both Kubernetes and OpenFaaS have the function of autoscaling. The OpenFaaS writes scaling configuration on deployment and Kubernetes writes it on HPA. Developers need to cautious that only set one restriction is OK. Now, we are going to introduce and realise them one by one.

6.3.1 Scaling by Kuberetes

In Kubernetes official website, it introduce a tool for controlling scaling named Horizontal Pod Autoscaler (HPA). It will scale pod according to container CPU load. Follow the steps of official website, the first step is creating a HPA controller (Fig.6.17).

```
root@VM-8-10-ubuntu:~/test# kubectl autoscale deployment mltest-final --cpu-percent=20 --min=2 --max=6
Error from server (NotFound): deployments.apps "mltest-final" not found
root@VM-8-10-ubuntu:~/test# kubectl autoscale deployment mltest-final -n openfaas-fn --cpu-percent=20 -
-min=2 --max=6
horizontalpodautoscaler.autoscaling/mltest-final autoscaled
```

FIGURE 6.17: Creating HPA controller

Then, developers can check and modify the configuration by using "kubectl -n OpenFaaS-fn edit HPA (deployment name)" (Fig.6.18):

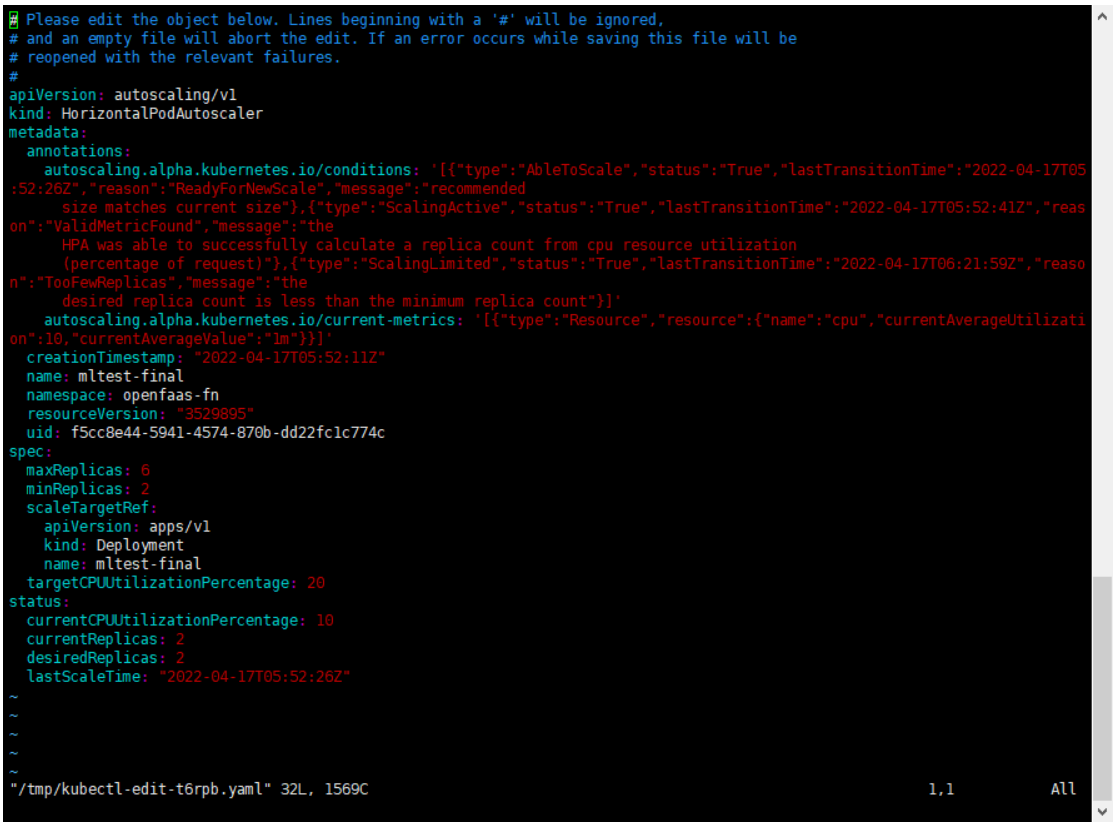

FIGURE 6.18: HPA configuration

By the way, it can not modify it by creating HPA again. It is only available to modify the existing HPA configuration.

Before next step experiment, we need check the number of pods (Fig.6.19):

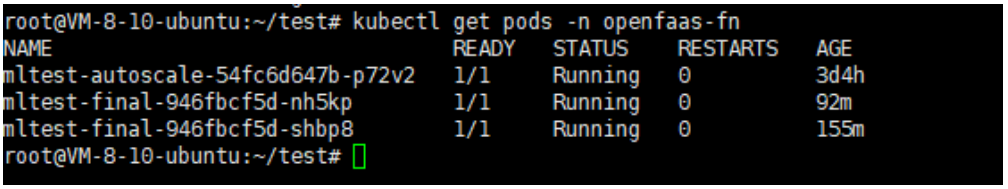

FIGURE 6.19: Ensure status

We confirm our deployment is running correctly (The name of deployment in this experiment is mltest-final and the minimum pods number is 2 which we set). Then, I invoke the function about a high load request:

Firstly, developers will notice the containers are increasing (Fig.6.20).

```
root@VM-8-10-ubuntu:~/test# kubectl get pods -n openfaas-fn
NAME                                READY   STATUS              RESTARTS   AGE
mltest-autoscale-54fc6d647b-p72v2   1/1     Running             0          3d4h
mltest-final-946fbcf5d-9fbb8        0/1     ContainerCreating   0          0s
mltest-final-946fbcf5d-bvzj9        0/1     ContainerCreating   0          0s
mltest-final-946fbcf5d-nh5kp        1/1     Running             0          110m
mltest-final-946fbcf5d-shbp8        1/1     Running             0          173m
```

FIGURE 6.20: Container increase

Second, the first phrase pods are running. The next two pods are creating (Fig.6.21):

```
root@VM-8-10-ubuntu:~/test# kubectl get pods -n openfaas-fn
NAME                                READY   STATUS    RESTARTS   AGE
mltest-autoscale-54fc6d647b-p72v2   1/1     Running   0          3d4h
mltest-final-946fbcf5d-79xsf        0/1     Running   0          3s
mltest-final-946fbcf5d-9bl56        0/1     Running   0          3s
mltest-final-946fbcf5d-9fbb8        1/1     Running   0          18s
mltest-final-946fbcf5d-bvzj9        1/1     Running   0          18s
mltest-final-946fbcf5d-nh5kp        1/1     Running   0          110m
mltest-final-946fbcf5d-shbp8        1/1     Running   0          173m
```

FIGURE 6.21: Container continue increase

Finally we notice that all the pods are running and the load is very high (Fig.6.22 Fig.6.23):

```
root@VM-8-10-ubuntu:~/test# kubectl get pods -n openfaas-fn
NAME                                READY   STATUS    RESTARTS   AGE
mltest-autoscale-54fc6d647b-p72v2   1/1     Running   0          3d4h
mltest-final-946fbcf5d-79xsf        1/1     Running   0          49s
mltest-final-946fbcf5d-9bl56        1/1     Running   0          49s
mltest-final-946fbcf5d-9fbb8        1/1     Running   0          64s
mltest-final-946fbcf5d-bvzj9        1/1     Running   0          64s
mltest-final-946fbcf5d-nh5kp        1/1     Running   0          111m
mltest-final-946fbcf5d-shbp8        1/1     Running   0          174m
```

FIGURE 6.22: Running final status

```
root@VM-8-10-ubuntu:~/test# kubectl get hpa -n openfaas-fn
NAME              REFERENCE                     TARGETS         MINPODS   MAXPODS   REPLICAS   AGE
mltest-autoscale  Deployment/mltest-autoscale   10%/20%         1         5         1          3d20h
mltest-demo       Deployment/mltest-demo        <unknown>/2%    1         10        0          3d21h
mltest-final      Deployment/mltest-final       5830%/20%       2         6         6          111m
root@VM-8-10-ubuntu:~/test#
```

FIGURE 6.23: Deployment load

The load of deployment is very high. Generally if the load over 20 percent, it will expansion. Because we set the biggest number of pods is 6, the pods number achieve 6.

Then we stop request. After a while, HPA detect the load is low, and it zoom back to 2 (Fig.6.24):

```
root@VM-8-10-ubuntu:~/test# kubectl get pods -n openfaas-fn
NAME                                READY   STATUS    RESTARTS   AGE
mltest-autoscale-54fc6d647b-p72v2   1/1     Running   0          3d4h
mltest-final-946fbcf5d-79xsf        1/1     Running   0          3m25s
mltest-final-946fbcf5d-9bl56        1/1     Running   0          3m25s
mltest-final-946fbcf5d-9fbb8        1/1     Running   0          3m40s
mltest-final-946fbcf5d-bvzj9        1/1     Running   0          3m40s
mltest-final-946fbcf5d-nh5kp        1/1     Running   0          114m
mltest-final-946fbcf5d-shbp8        1/1     Running   0          177m
root@VM-8-10-ubuntu:~/test# kubectl get pods -n openfaas-fn
NAME                                READY   STATUS    RESTARTS   AGE
mltest-autoscale-54fc6d647b-p72v2   1/1     Running   0          3d5h
mltest-final-946fbcf5d-nh5kp        1/1     Running   0          132m
mltest-final-946fbcf5d-shbp8        1/1     Running   0          3h15m
root@VM-8-10-ubuntu:~/test#
```

FIGURE 6.24: Finish load status

6.3.2 OpenFaaS scaling

Here we display a new simple scaling model which is not restricted by Kubernetes. It is allowed to scale pod number to 0 in Kubernetes. First, we scale pod number to 0 (Fig.6.25).

```
root@VM-8-10-ubuntu:~# kubectl get deploy mltest-dev -n openfaas-fn
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
mltest-dev   1/1     1            1           13h
root@VM-8-10-ubuntu:~# kubectl scale deploy --replicas=0 mltest-dev -n openfaas-
fn
deployment.apps/mltest-dev scaled
root@VM-8-10-ubuntu:~# kubectl get deploy mltest-dev -n openfaas-fn
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
mltest-dev   0/0     0            0           13h
root@VM-8-10-ubuntu:~#
```

FIGURE 6.25: Scale to 0

Then we invoke our machine learning model. Developers notice that it return result and the number of the pod is 1 (Fig.6.26).

```
root@VM-8-10-ubuntu:~# echo -n "50" | faas-cli invoke mltest-dev -gateway 175.17
8.189.207:31112
2022-03-28 04:30:10.460484: W tensorflow/stream_executor/platform/default/dso_lo
ader.cc:64] Could not load dynamic library 'libcudart.so.11.0'; dlerror: libcuda
rt.so.11.0: cannot open shared object file: No such file or directory
2022-03-28 04:30:10.460523: I tensorflow/stream_executor/cuda/cudart_stub.cc:29]
 Ignore above cudart dlerror if you do not have a GPU set up on your machine.
9149
4/4 [==============================] - 0s 820us/step - loss: 0.4028 - accuracy:
0.8362
Test Accuracy: 0.836
Predicted: 0.833
Predicteds: G
root@VM-8-10-ubuntu:~# kubectl get deploy mltest-dev -n openfaas-fn
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
mltest-dev   1/1     1            1           14h
```

FIGURE 6.26: Invoke and scale

As we mentioned before, the OpenFaaS will scale according to the server load. Developers can set a certain parameter before deployment is deployed (In consideration of different versions of OpenFaaS, it is hard to modify parameter after it is deployed. We will explain it in the end of the paper). Here we will show developers how to scale deploy. In official document of OpenFaaS, it shows some labels of auto-scaling and developers can add these labels in shell (Fig.6.27).

Label	Description	Default
`com.openfaas.scale.max`	The maximum number of replicas to scale to.	`20`
`com.openfaas.scale.min`	The minimum number of replicas to scale to.	`1`
`com.openfaas.scale.zero`	Whether to scale to zero.	`false`
`com.openfaas.scale.zero-duration`	Idle duration before scaling to zero	`15m`
`com.openfaas.scale.target`	Target load per replica for scaling	`50`
`com.openfaas.scale.target-proportion`	Proportion as a float of the target i.e. 1.0 = 100% of target	`0.90`
`com.openfaas.scale.type`	Scaling mode of `rps`, `capacity`, `cpu`	`rps`

FIGURE 6.27: The labels of auto-scaling [13]

After deploying it successfully, we can check it by using "kubectl edit deploy mltest-demo -n OpenFaaS-fn" (Fig.6.28):

```
selector:
  matchLabels:
    faas_function: mltest-demo
strategy:
  rollingUpdate:
    maxSurge: 1
    maxUnavailable: 0
  type: RollingUpdate
template:
  metadata:
    annotations:
      prometheus.io.scrape: "false"
    creationTimestamp: null
    labels:
      com.openfaas.scale.max: "10"
      com.openfaas.scale.min: "4"
      com.openfaas.scale.target: "5"
      com.openfaas.scale.target-proportion: "0.20"
      com.openfaas.scale.type: cpu
      com.openfaas.scale.zero: "true"
      faas_function: mltest-demo
      uid: "301550677"
                                                    39,1          Top
```

FIGURE 6.28: Deployment set up

From the image, we notice that the setting of the biggest number of deployment is 10 and the minimum is 4. We also allow deployment scaling to 0. The server use CPU and the proportion is 0.2. The proportion is calculated by these two functions:

1) The current load is used to calculate the new number of replicas:
$$desired = \text{current replicas} \times \frac{\text{current load}}{\text{target load per replica} \times \text{current replicas}}$$

2) The target-proportion flag can be used to adjust how early or late scaling occurs:
$$desired = \text{current replicas} \times \frac{\text{current load}}{\text{target load per replica} \times \text{current replicas} \times \text{target proportion}}$$

After setup is completed, we need to invoke pods to realize scaling. Make it run function continuously6.29.

```
root@VM-8-10-ubuntu:~/test# kubectl get svc -n openfaas-fn
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
mltest-demo   ClusterIP   10.68.165.207   <none>        8080/TCP   3d1h
root@VM-8-10-ubuntu:~/test# for i in {0..1000}; do  echo -n "50" | faas-cli invoke mltest-de
mo -g 10.68.165.207:8080; done;
```

FIGURE 6.29: Invoke pods

When it running I use another terminal to monitor its status (Fig.6.30):

```
root@VM-8-10-ubuntu:~/test# kubectl get pods -n openfaas-fn
NAME                              READY   STATUS    RESTARTS   AGE
mltest-demo-6d59655c6b-7tg5t   1/1     Running   0          4m43s
mltest-demo-6d59655c6b-c62nj   1/1     Running   0          4m50s
mltest-demo-6d59655c6b-g8tjd   1/1     Running   0          4m46s
root@VM-8-10-ubuntu:~/test#
```

FIGURE 6.30: Invoke pods

Finally, we check the server by "kubectl edit deploy mltest-demo -n OpenFaaS-fn" again (Fig.6.31):

```
spec:
  progressDeadlineSeconds: 600
  replicas: 3
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      faas_function: mltest-demo
```

FIGURE 6.31: Pods replication

The number of replications is 3 actually. By the way, here we use OpenFaaS pro. It is ineffective to control scaling by using CPU in OpenFaaS-CE. we will give more details on chapter 7.

6.4 Analysis of CPU, Memory, and Response Time

In this chapter, we are going to analyzing CPU, memory consumption and response delay. In order to research cold and warm start, we divide experiment to two parts. It is easy to analysis warm start. We can just set certain number pods but not run any function. Then we make it run a function continuously and watch the data graph. Then when we analyse cold start, we can scale pods to 0 and set the minimum scale number. Watching data graph to analyse [35].

6.4.1 CPU and memory consumption

We mentioned that we installed node-exporter and cAdvisor. Here we make pods cold start and warm start. Because there is deviation here, we are going to calculate average result of two tools. Finally, we will compare them.

6.4.1.1 Cold start

Firstly, I set the scaling number is 5. Then I scale the pods to 0 (Fig.6.32).

```
  uid: 85056d75-eca6-46a6-a426-3aa21cba63e0
spec:
  progressDeadlineSeconds: 600
  replicas: 5
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      faas_function: mltest-demo
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
    type: RollingUpdate
  template:
    metadata:
      annotations:
        prometheus.io.scrape: "false"
      creationTimestamp: null
      labels:
        com.openfaas.scale.min: "5"
        com.openfaas.scale.max: "6"
        faas_function: mltest-demo
        uid: "925455461"
      name: mltest-demo
```

FIGURE 6.32: Cold start scaling

6.4.2 Warm and cold start latency

In order to watch warm start latency, we need to start all of the container first (6.33).

```
root@VM-8-10-ubuntu:~# kubectl get pods -n openfaas-fn
NAME                                READY   STATUS              RESTARTS   AGE
mltest-autoscale-54fc6d647b-p72v2   1/1     Running             0          6d
mltest-final-946fbcf5d-82nm5        0/1     ContainerCreating   0          2s
mltest-final-946fbcf5d-gkdf7        0/1     ContainerCreating   0          2s
mltest-final-946fbcf5d-glw4q        0/1     ContainerCreating   0          2s
mltest-final-946fbcf5d-mp77w        0/1     ContainerCreating   0          2s
mltest-final-946fbcf5d-shbp8        1/1     Running             0          2d22h
mltest-final-946fbcf5d-v6jv2        0/1     ContainerCreating   0          2s
root@VM-8-10-ubuntu:~#
```

FIGURE 6.33: Warm start preparing

In next step, we use JMeter to send request and watch latency (Fig.6.34).

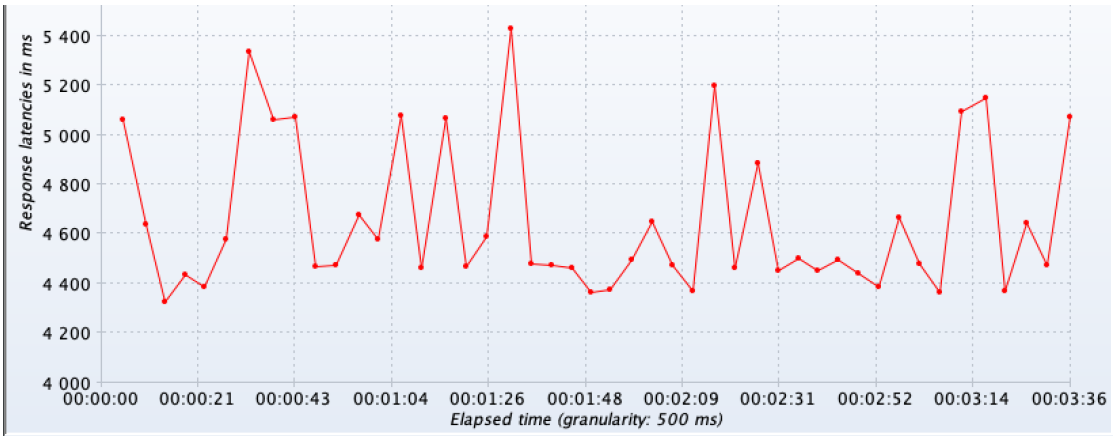

FIGURE 6.34: Warm start response latency

Then we check memory of each pods we used (Fig.6.35).

```
root@VM-8-10-ubuntu:~# kubectl top pods -n openfaas-fn
NAME                                CPU(cores)   MEMORY(bytes)
mltest-autoscale-54fc6d647b-p72v2   1m           21Mi
mltest-final-946fbcf5d-5tngr        1m           12Mi
mltest-final-946fbcf5d-mzl7r        1m           17Mi
mltest-final-946fbcf5d-shbp8        322m         91Mi
mltest-final-946fbcf5d-shvr9        473m         25Mi
mltest-final-946fbcf5d-x42vb        1m           17Mi
mltest-final-946fbcf5d-zwpq8        1m           12Mi
```

FIGURE 6.35: Used memory of each pods

Then we wait HPA control the scaling to initial status. We may need a few minutes. Here is the HPA latency illustrate we found in official document "You should note that HPA is designed to react slowly to changes in traffic, both for scaling up and for scaling down. In some instances you may wait 20 minutes for all your Pods to scale back down to a normal level after the load has stopped."

Here we wait a few minutes for the initial status (Fig.6.36):

```
root@VM-8-10-ubuntu:~# kubectl get pods -n openfaas-fn
NAME                                READY   STATUS        RESTARTS   AGE
mltest-autoscale-54fc6d647b-p72v2   1/1     Running       0          6d3h
mltest-final-946fbcf5d-mzl7r        0/1     Terminating   0          12m
mltest-final-946fbcf5d-shbp8        1/1     Running       0          3d1h
mltest-final-946fbcf5d-shvr9        0/1     Terminating   0          24m
```

FIGURE 6.36: HPA control deployment restore to initial status

In next step, we will watch cold start latency. After the deployment resuming to initial status, we check the HPA and modify the max auto-scaling number same to warm start number. There is one thing need to be attention here. The cold start process is:

1) When the CPU load reach to the load boundary, It will expansion a part of pods and the initial continue execute function by warm start and new pods execute cold start.

2) When the new pods finish cold start it will execute function by warm start and initial pods continue warm start.

3) when the CPU load reach load boundary, it will create some other pods and repeat above process until reach the max pods restriction.

Here we use JMeter request a high load to OpenFaaS and watch the time of pods' number variety in Grafana. Then compare the time of request latency in JMeter. Above that, we can analysis which is cold start latency and which is warm start latency(Developers need cautious that set the Grafana time to browser time.). Here is JMeter request (Fig.6.37):

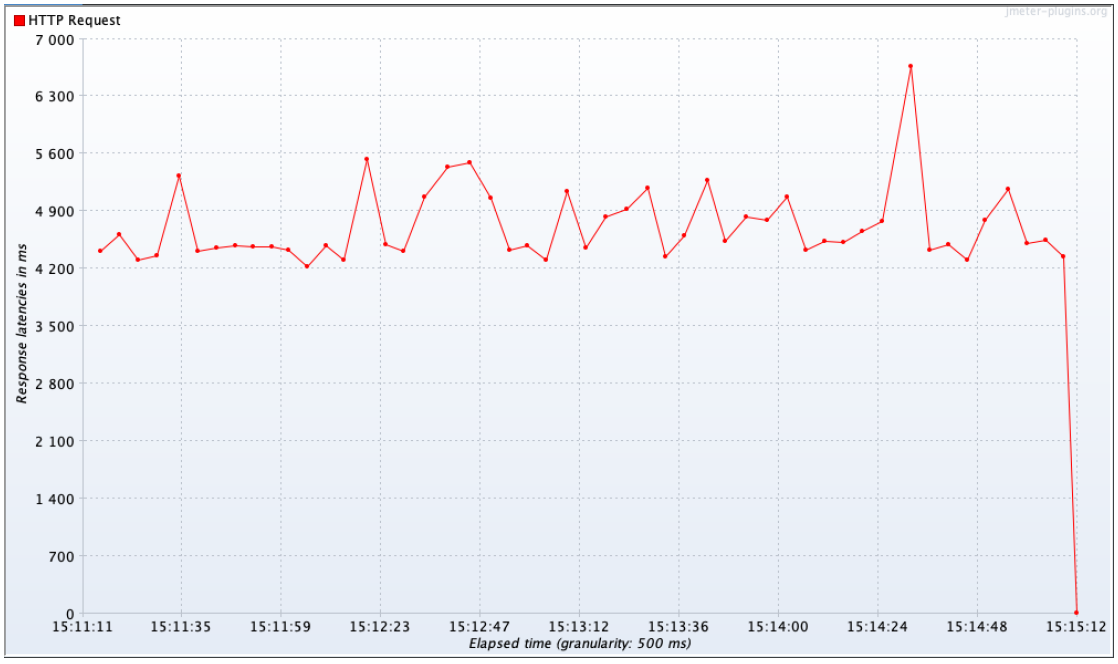

FIGURE 6.37: Cold start scaling in JMeter

And here is the record in grafana (Fig.6.38):

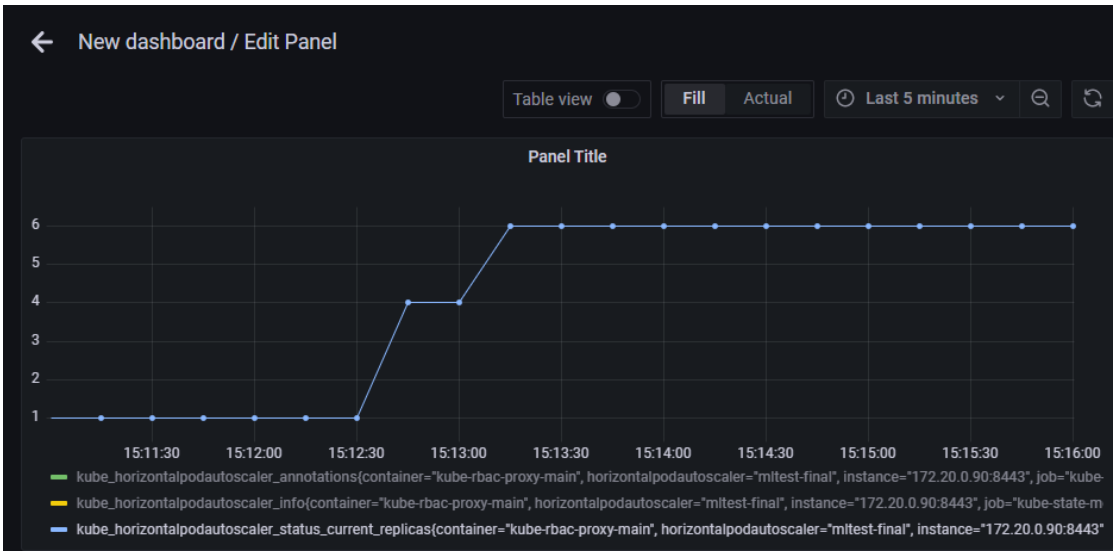

FIGURE 6.38: Cold start scaling in Grafana

From these two image, we find that the pod expansion two times. The first time in 15:12:45 and the second time in 15:13:15. Then we compare the JMeter, there is a relative high latency in 15:12:45. However, the latency is relative low in 15:13:45.

In order to ensure accuracy, we are going to do test one more time. This time we prepare to modify the max number of auto-scaling to 10. Here is the Grafana record of 10 auto-scaling (Fig.6.39).

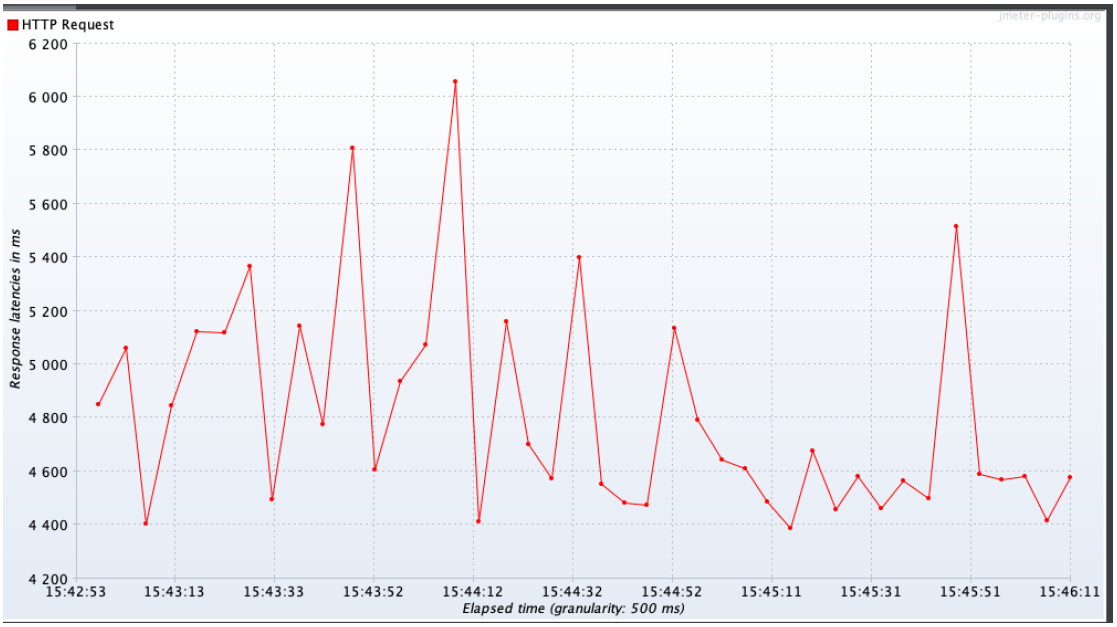

FIGURE 6.39: Cold start scaling in JMeter 2

Here is auto-scaling record in JMeter (Fig.6.40):

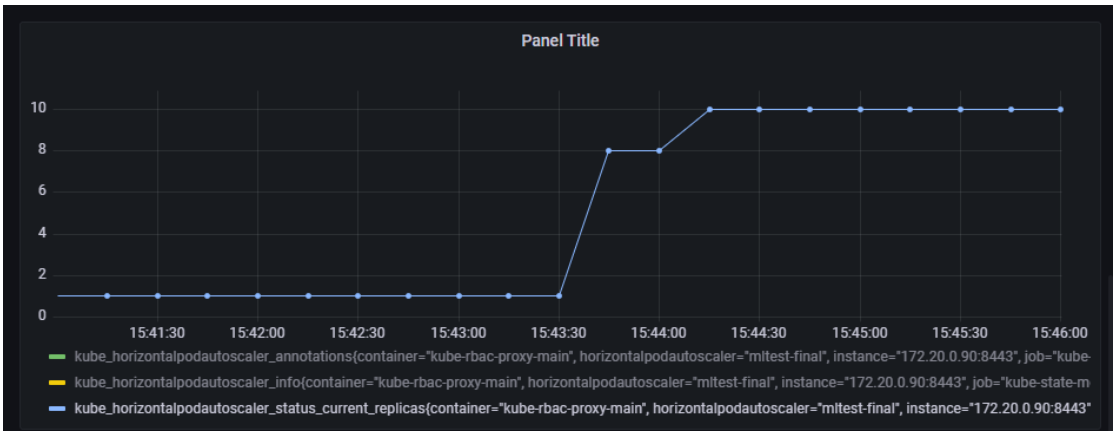

FIGURE 6.40: Cold start scaling in Grafana 2

From the image of Grafana, we notice that the time of first scaling is 15:43:45. The second scaling time is 15:44:15. Comparing two record in this experiment, the first scaling has a relative high latency. The second scaling has a relative low latency.

Here we are going to do another comparing experiment about different CPU load restriction of each pod. We modify back the number of max scaling to 6. Then change CPU load percent from 20 to 40. Here is the record in JMeter (Fig.6.41):

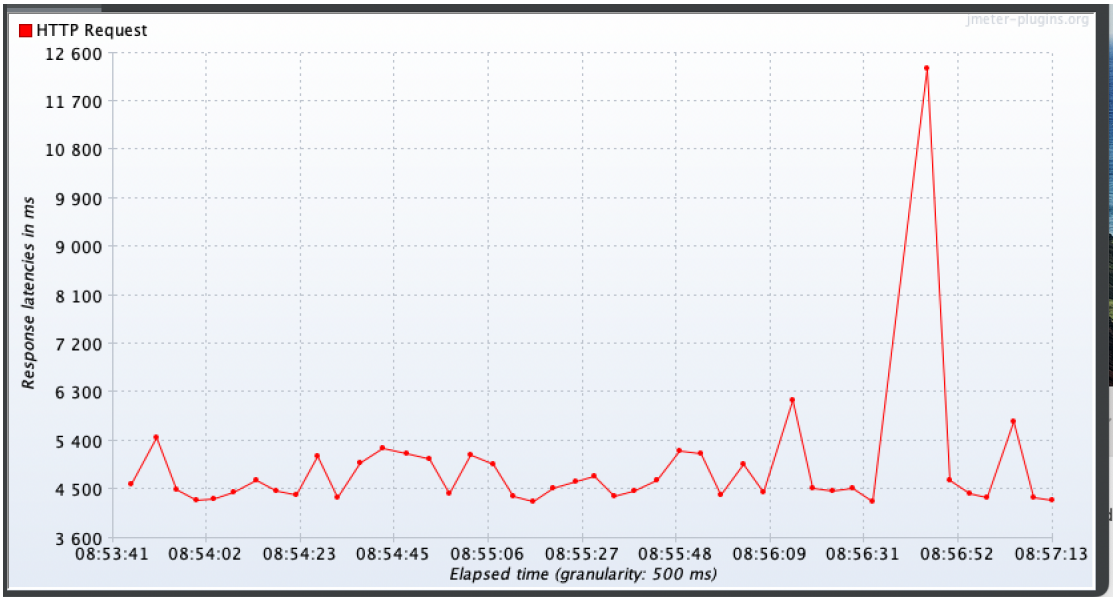

FIGURE 6.41: Cold start scaling in JMeter 3

Then, here is record in Grafana (Fig.6.42):

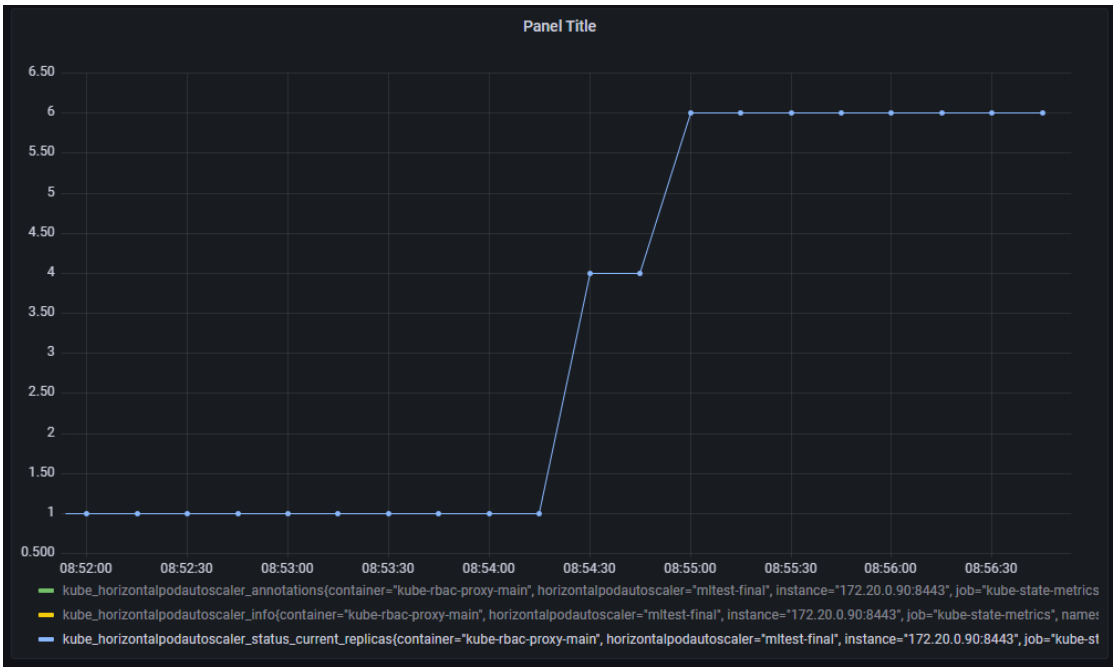

FIGURE 6.42: Cold start scaling in Grafana 3

From the image of Grafana, we notice that the first expansion at 8:54:45. The second time of expansion in 8:55:00. Comparing to record in JMeter, these 2 time are in the peak of relative time but the peak not higher than other peaks.

We have done 3 experiment so far. We notice that the latency more pods' expansion the higher latency. Then the higher latency accompanied by the lower CPU load restrict of each pod.

6.4.3 CPU and memory load analysis

In this part, we are going to comparing CPU and memory load to pod scaling time in Grafana.

First of all, we change load restrict back to 20 percent. From the image we notice that the pod zoom to 1 and we can just watch one pod's load in deployment of "mltest-final" (Fig.6.43).

FIGURE 6.43: CPU load and pods' number initial data

Then we request to OpenFaaS by JMeter infinitely. Here is record in JMeter (Fig.6.44):

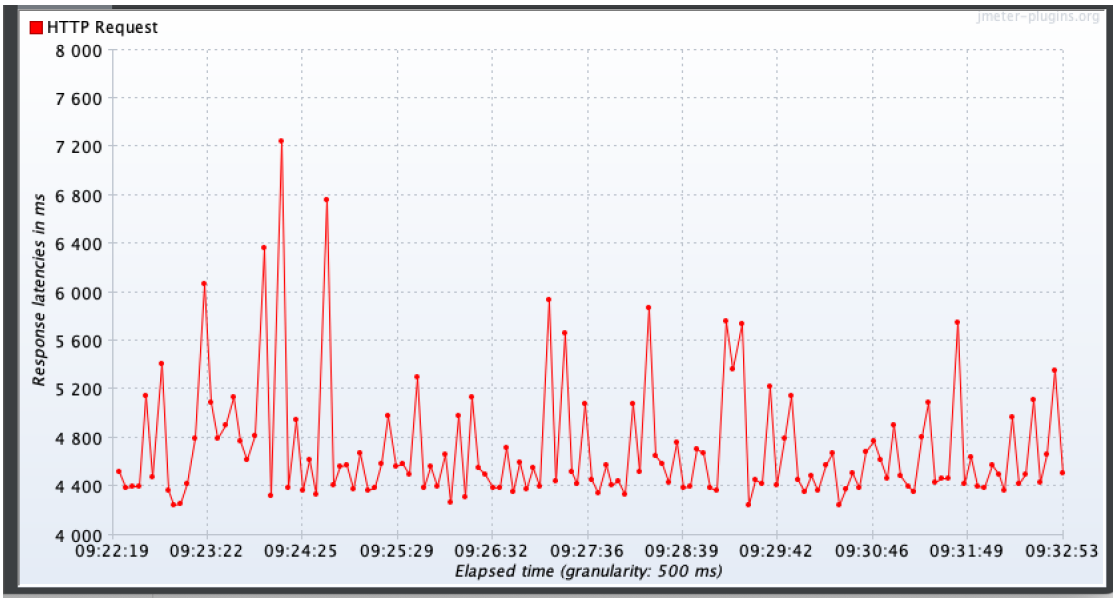

FIGURE 6.44: JMeter record in CPU test.

After that, we wait for 15 minutes. Here are the CPU load and pods' scaling record (Fig.6.45):

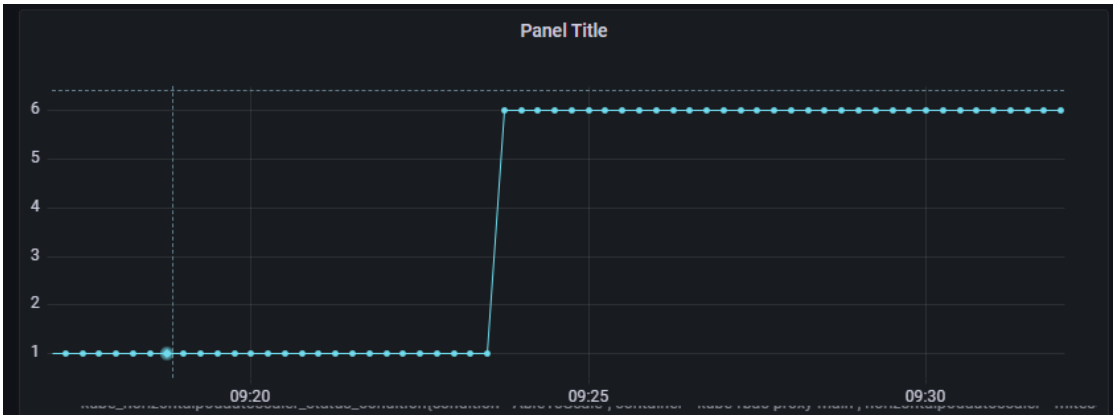

FIGURE 6.45: Scaling record

This is scaling record, we can notice that it expansion around 9:24. Then, we are going to see the CPU load record (Fig.6.46 Fig.6.47 Fig.6.48 Fig.6.49 Fig.6.50 Fig.6.51):

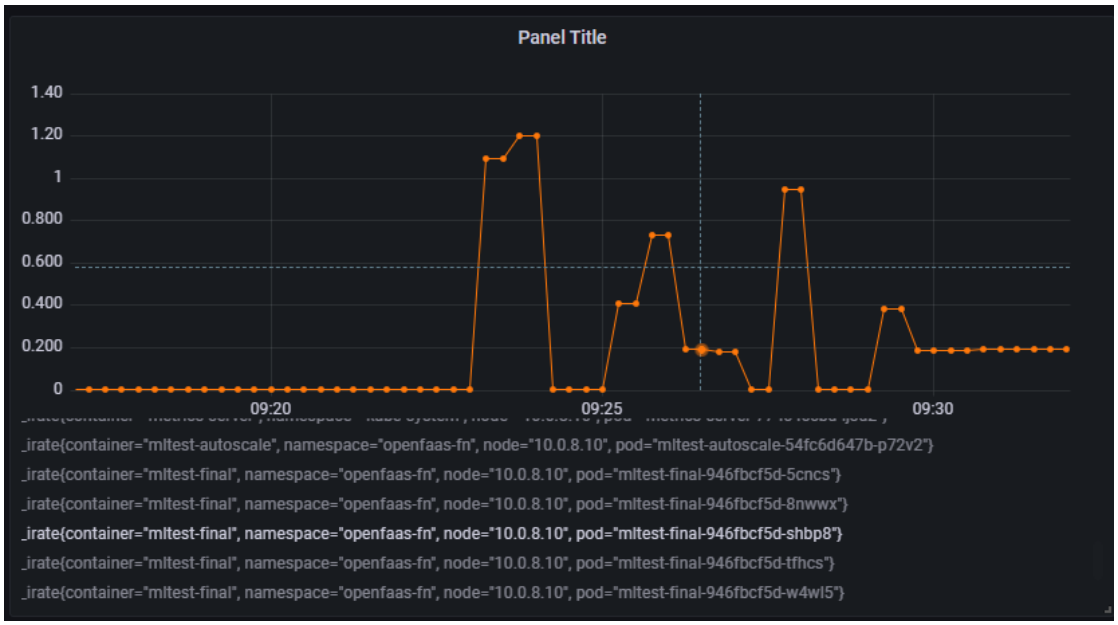

FIGURE 6.46: CPU load record 1

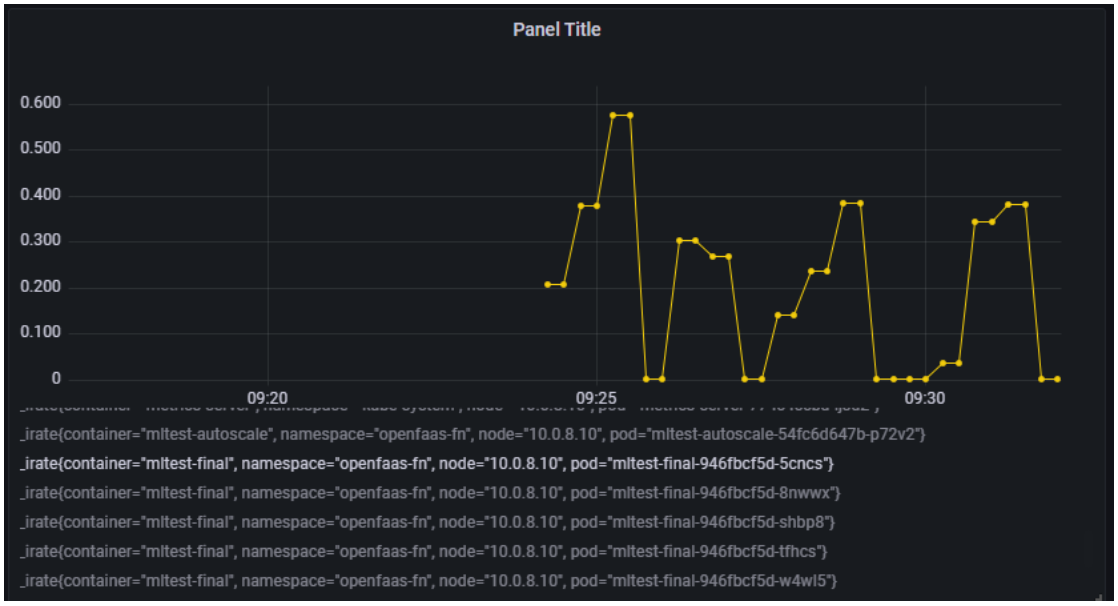

FIGURE 6.47: CPU load record 2

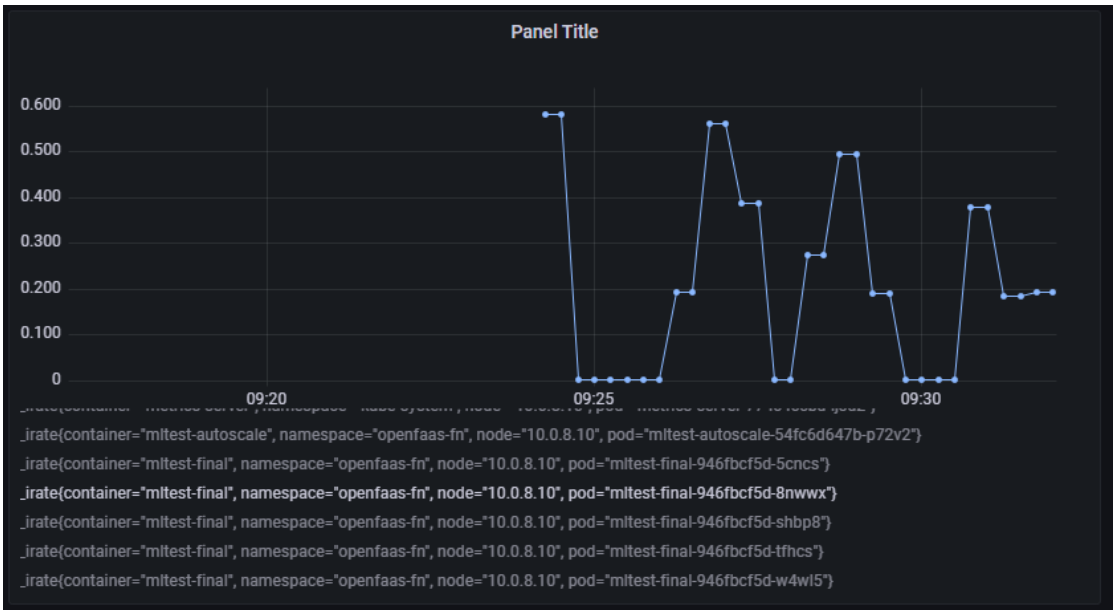

FIGURE 6.48: CPU load record 3

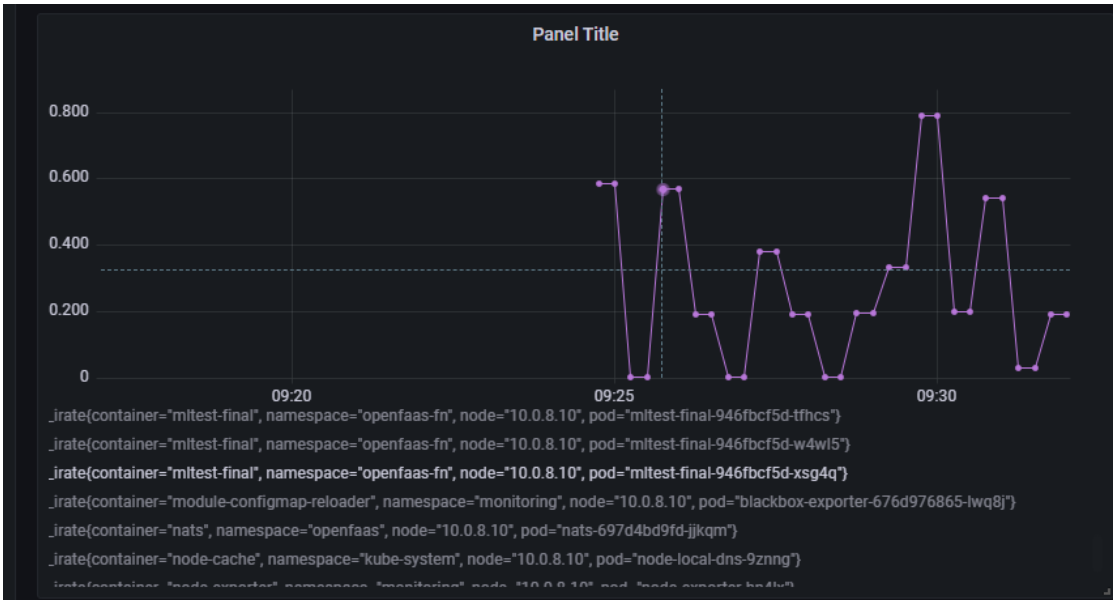

FIGURE 6.49: CPU load record 4

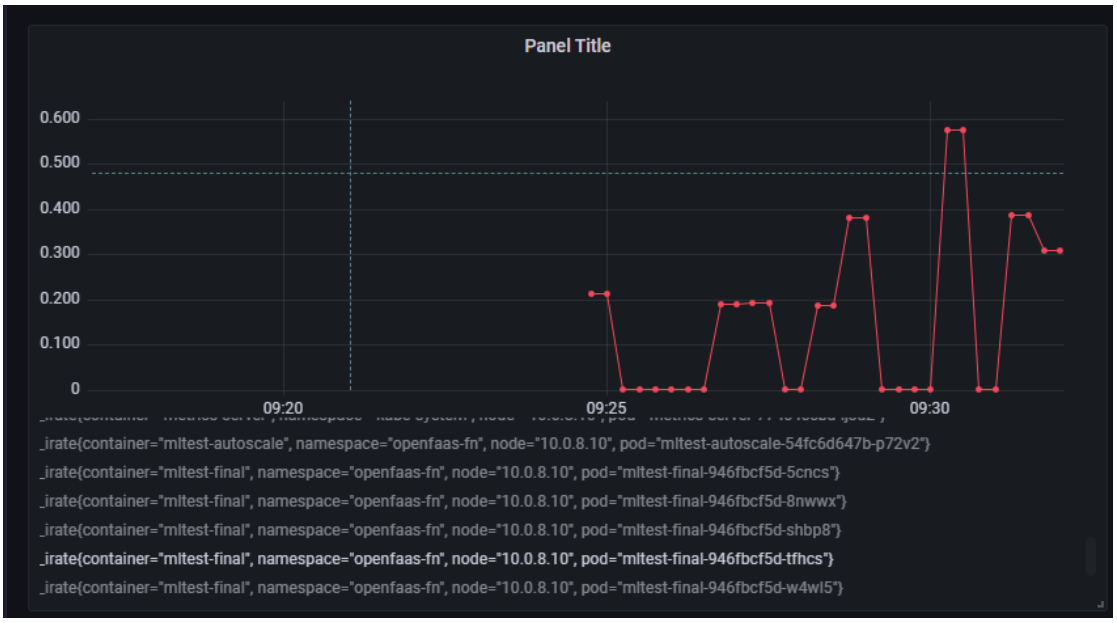

FIGURE 6.50: CPU load record 5

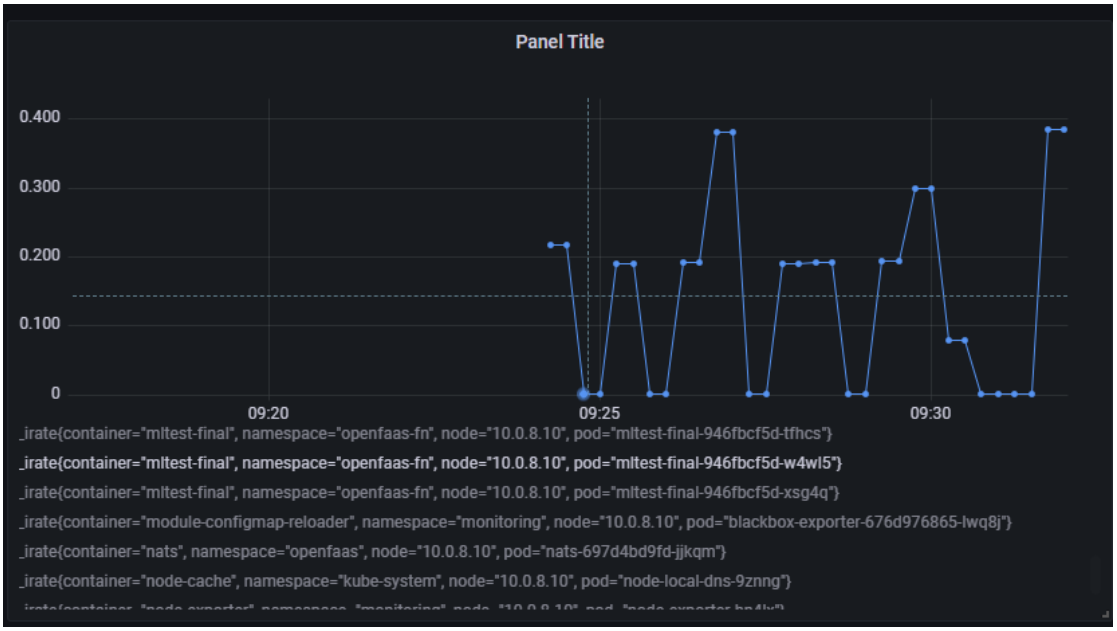

FIGURE 6.51: CPU load record 6

Here we notice there are 6 pods because we set the max expansion number is 6. These record include 1 initial pod (the first one) and 5 new scaling pods. In the first image (the initial pod), the CPU load rise from 0% t0 1.2% at 9:23. Other pods are created at 9:24 and CPU load rise. The peak of peak load around 0.5. After the CPU load reach the peak, it will decline and rise up again. Then keep repeating this process. However, the initial pod peak will decline which is same to other pods' peak.

The result which is shown to us correspond our imagination:

1) When we give a high load request to OpenFaaS, the initial pod will process it, so there is high load of this pod.

2) The high load exceed its load boundary, so the deployment expand. We can notice that the expansion time is same to new pod creating time. It shows us which pods are new created.

3) When the new pods are created, the CPU load is not very high in initial pod because other pods will help it finish mission. We notice that the initial pods' CPU load will decline after scaling. Every pods' CPU load all close to 0.6 because all pods finish mission together. Why the CPU load is 0.6 but not 0.2 (Our configuration is 0.2)? The season our max pod number is 6. It can not expend more pod. Each pod need 0.6 CPU load to finish mission.

4) From the image, we find that the load curve is not stationary. It means when it process function, it will warm start and the CPU load is high. When one function finish, it the pod will not work and wait to process next function. At this time, the CPU load is close to 0. This process will be repeated until the request stop.

Chapter 7

Issues and Addressing Strategy

7.1 Using local images

As we mentioned before, the OpenFaaS bases on Kubernetes and Kubernetes bases on docker. Developers can treat docker as a basic virtual machine to be a container. If we use default OpenFaaS configuration such as "faas-cli up", it will upload our local images which have been built to docker hub. And then, the OpenFaaS will use our docker images which is in docker hub. In most scenarios, it is convenient for developers, because most FaaS system bases on distributed system or a lot of nodes. Both these two kinds of containers have their own IP. In this case, using docker hub is the best choice to deploy deployment to every containers. But for beginners, they need to deploy them on their local environment first. Besides, it is also not convenient for developers modify one container. We need to learn how to use local images but not images on docker hub.

7.1.1 Using Minikube

First of all, we need to know what is Minikube. Minikube is a simplified version of Kubernetes. Different to Kubernetes, Minikube is an image of docker actually. For beginners, they do not need to install Kubernetes, they can just use docker which they have installed to realize. Besides, beginners not need to control a cluster. Deploying a single node is OK for them. Minikube can only deploy a single node [36].

For Minikube, the best choice is using cloud images such as docker hub. The reason is Minikube to be an image, it is in the same management level with other images. Uploading and downloading images is easier for Minikube because it avoid management level problem. Here we will explain why it is difficult to control same management docker image for Minikube: (Fig.7.1)

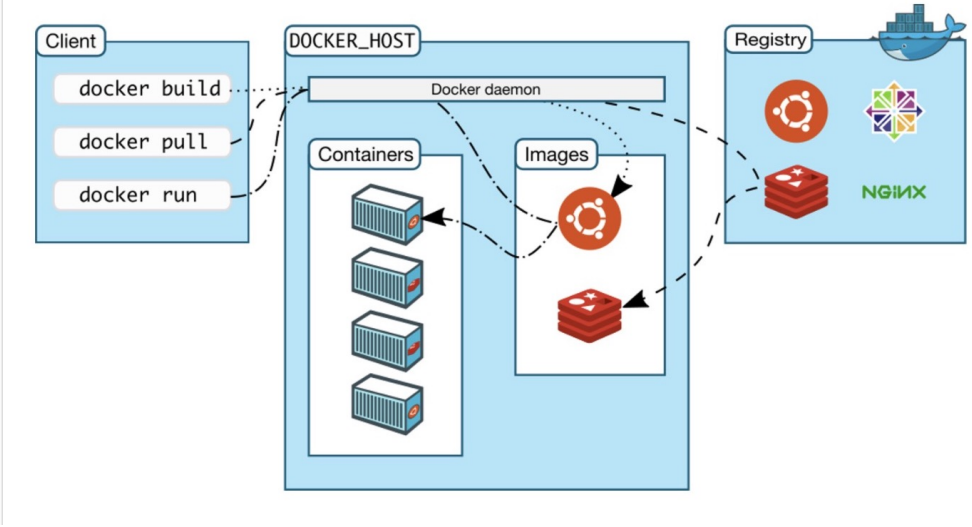

FIGURE 7.1: Docker working process [14]

In this image we notice that users control docker host by using client. Exactly, users send commands to docker daemon and docker daemon transform commends to different parts. If there are not required images in local environment, docker daemon will download image form docker registry to local environment. Then docker daemon will put the images in corresponding containers. However, the local images are usually in host. Kubernetes will order docker to complete all thing but Minikube can not, because they are in same level, deployment is not in Minikube containers.

But in some conditions, developer will meet many problems by using Minikube. If developers just using docker hub by free, it will limit their usage count. The transfer speed is also not good for docker hub. One solution is using local images. First we need to set environment variable by using “eval $(Minikube docker-env)”. After that, we need to change imagePullPolicy to “IfNotPresent” or Never in “.yml” file, otherwise it will pull image from internet. Finally, developers can use “build”, “deploy” command to use local images.

There is another method avoid using docker hub. Developers can use other cloud image service to replace docker hub. In ".yml" file, developer can replace their docker hub username to other cloud image service address.

7.1.2 Using Kubernetes

Kubernetes is most widely used [27]. After developers installing Kubernetes environment, there is no need to enter Minikube environment. They can operate docker directly in host environment. But when we using Kubernetes, we meet a bug about the YML file in host can not control deployment YML file in docker container (We hope it can be solved in subsequent versions). In order to complete it, we modify the actually deployment in container:(Fig.7.2 Fig.7.3)

```
root@VM-8-10-ubuntu:/home/ubuntu# kubectl edit deploy mltest-dev -n openfaas-fn
deployment.apps/mltest-dev edited
root@VM-8-10-ubuntu:/home/ubuntu# kubectl get po -n openfaas-fn
NAME                          READY   STATUS        RESTARTS   AGE
```

FIGURE 7.2: Controlling actually deployment

```
chriswong — root@VM-8-10-ubuntu: ~ — ssh ubuntu@175.178.189.207 — 80×24
    spec:
      containers:
      - env:
        - name: exec_timeout
          value: 5s
        - name: fprocess
          value: python3 index.py
        - name: read_timeout
          value: 5s
        - name: sleep_duration
          value: "0"
        - name: write_timeout
          value: 8s
        image: demo1/mltest:latest
        imagePullPolicy: IfNotPresent
        livenessProbe:
          failureThreshold: 3
          httpGet:
            path: /_/health
            port: 8080
            scheme: HTTP
          initialDelaySeconds: 2
          periodSeconds: 2
                                                              50,1          45%
```

FIGURE 7.3: Modify pulling strategy

Developers will notice that I change "imagePullPolicy" to "IfNotPresent". Then, developers can save the configuration file, it will be deployed.

7.2 Operating in docker

We mentioned that the essence of Kubernetes is a tool to manage docker containers. What about operate docker container directly without Kubernetes? Developers can treat docker container as virtual machine. There are three popular ways to operate docker container directly.

7.2.1 Use "docker attach" enter containers

This method means we can use developers' CMD tool to be a client of docker. First, developers can use "docker ps" to check the docker containers ID(Fig.7.4).

```
[ubuntu@VM-8-10-ubuntu:~$ sudo -i
[root@VM-8-10-ubuntu:~# docker ps
CONTAINER ID        IMAGE                                  COMMAND                  CREATED             STATUS              PORTS               NAMES
c45db977ab86        ccr.ccs.tencentyun.com/library/busybox   "fwatchdog"              46 hours ago        Up 46 hours                             k8s_mltest-demo_mltest-demo-69f8f88ff9-2t7sq_openfaas-fn_2e924e7-afda-4f04-b12d-0156a1cc5a2d_0
dac3e34d37a1        easzlab/pause-amd64:3.2                "/pause"                 46 hours ago        Up 46 hours                             k8s_POD_mltest-demo-69f8f88ff9-2t7sq_openfaas-fn_2e924e7-afda-4f04-b12d-0156a1cc5a2d_0
8dac811f790f        1628f4aaa3925                          "fwatchdog"              46 hours ago        Up 46 hours                             k8s_mltest-demo_mltest-demo-69f8f88ff9-95qw4_openfaas-fn_75e8dea8-7375-499e-a2a2-448f19aab894_0
ef25229b8852        easzlab/pause-amd64:3.2                "/pause"                 46 hours ago        Up 46 hours                             k8s_POD_mltest-demo-69f8f88ff9-95qw4_openfaas-fn_75e8dea8-7375-499e-a2a2-448f19aab894_0
d70b0e4b498c        ghcr.io/openfaas/faas-netes            "./faas-netes -opera…"   8 days ago          Up 8 days                               k8s_faas-netes_gateway-77f7d67cbc-6zw72_openfaas_619330f5-1ff4-4276-a7e2-4a841bf5e86b_0
9c95e4babb1a        prom/alertmanager                      "alertmanager --conf…"   8 days ago          Up 8 days                               k8s_alertmanager_alertmanager-c4f4d8fb8-n2g6r_openfaas_cf1607f8-8d88-4428-92c7-30000488ab80_0
d88fabcd054a        ghcr.io/openfaas/gateway               "./gateway"              8 days ago          Up 8 days                               k8s_gateway_gateway-77f7d67cbc-6zw72_openfaas_619330f5-1ff4-4276-a7e2-4a841bf5e86b_0
eec81e66e6c5        prom/prometheus                        "prometheus --config…"   8 days ago          Up 8 days                               k8s_prometheus_prometheus-59879547ff-p2pzz_openfaas_aeb4007d-9e8a-40d9-8ee6-07a930d504bb_0
ffefb0ee6968        ghcr.io/openfaas/queue-worker          "./app"                  8 days ago          Up 8 days                               k8s_queue-worker_queue-worker-757994c7d-cmh27_openfaas_97ab1d89-a077-4d91-85d7-6ffbbb539f18_0
56ee7d727cf5        nats-streaming                         "/nats-streaming-ser…"   8 days ago          Up 8 days                               k8s_nats_nats-697d4bd9fd-jjkqm_openfaas_b6300d09-24f9-4225-a7cd-93c32e07fc36_0
1128875c6936        ghcr.io/openfaas/basic-auth            "./handler"              8 days ago          Up 8 days                               k8s_basic-auth-plugin_basic-auth-plugin-98cd974cc-4v8ct_openfaas_9eb557fc-ac29-48e5-85b8-eb221bf0b07e_0
606a80e21c67        easzlab/pause-amd64:3.2                "/pause"                 8 days ago          Up 8 days                               k8s_POD_alertmanager-c4f4d8fb8-n2g6r_openfaas_cf1607f8-8d88-4428-92c7-30000488ab80_0
21c3d68ac04e        easzlab/pause-amd64:3.2                "/pause"                 8 days ago          Up 8 days                               k8s_POD_gateway-77f7d67cbc-6zw72_openfaas_619330f5-1ff4-4276-a7e2-4a841bf5e86b_0
b777f97b2f3b        easzlab/pause-amd64:3.2                "/pause"                 8 days ago          Up 8 days                               k8s_POD_prometheus-5987954ff-p2pzz_openfaas_aeb4007d-9e8a-40d9-8ee6-07a930d504bb_0
5ba0613126 1f       easzlab/pause-amd64:3.2                "/pause"                 8 days ago          Up 8 days                               k8s_POD_nats-697d4bd9fd-jjkqm_openfaas_b6300d09-24f9-4225-a7cd-93c32e07fc36_0
bcc29a497772        easzlab/pause-amd64:3.2                "/pause"                 8 days ago          Up 8 days                               k8s_POD_basic-auth-plugin-98cd974cc-4v8ct_openfaas_9eb557fc-ac29-48e5-85b8-eb221bf0b07e_0
89a14d47e13e        easzlab/pause-amd64:3.2                "/pause"                 8 days ago          Up 8 days                               k8s_POD_queue-worker-757994c7d-cmh27_openfaas_97ab1d89-a077-4d91-85d7-6ffbbb539f18_0
f072f29816de        29589495df8d                           "/usr/local/bin/kube…"   8 days ago          Up 8 days                               k8s_kube-rbac-proxy-self_kube-state-metrics-5d6885d89-4q777_monitoring_4c93d218-c79e-44fa-b1d1-f8881dde8599_0
2fc475356126        29589495df8d                           "/usr/local/bin/kube…"   8 days ago          Up 8 days                               k8s_kube-rbac-proxy-main_kube-state-metrics-5d6885d89-4q777_monitoring_4c93d218-c79e-44fa-b1d1-f8881dde8599_0
3811974ab863        f56b05ba1b94                           "/bin/prometheus-con…"   8 days ago          Up 8 days                               k8s_config-reloader_prometheus-k8s-1_monitoring_708daff0-d1c4-4efc-893e-265d530d028c_0
7d864a1921dd        e3cf894a63f5                           "/bin/prometheus --w…"   8 days ago          Up 8 days                               k8s_prometheus_prometheus-k8s-1_monitoring_708daff0-d1c4-4efc-893e-265d530d028c_0
3abffa59a21a        easzlab/pause-amd64:3.2                "/pause"                 8 days ago          Up 8 days                               k8s_POD_prometheus-k8s-1_monitoring_708daff0-d1c4-4efc-893e-265d530d028c_0
f57bcb1b07ea        f56b05ba1b94                           "/bin/prometheus-con…"   8 days ago          Up 8 days                               k8s_config-reloader_prometheus-k8s-0_monitoring_9ea586d0-a401-43b5-a9ff-0514ddf55c71_0
ab1bfc8eda77        e3cf894a63f5                           "/bin/prometheus --w…"   8 days ago          Up 8 days                               k8s_prometheus_prometheus-k8s-0_monitoring_9ea586d0-a401-43b5-a9ff-0514ddf55c71_0
51a18190b1be        easzlab/pause-amd64:3.2                "/pause"                 8 days ago          Up 8 days                               k8s_POD_prometheus-k8s-0_monitoring_9ea586d0-a401-43b5-a9ff-0514ddf55c71_0
51c17f51a807        f56b05ba1b94                           "/bin/prometheus-con…"   8 days ago          Up 8 days                               k8s_config-reloader_alertmanager-main-2_monitoring_d4cea671-e7a8-4866-b28a-187d3fb4de6d_0
4fa986faa9b6        f56b05ba1b94                           "/bin/prometheus-con…"   8 days ago          Up 8 days                               k8s_config-reloader_alertmanager-main-1_monitoring_7359513c-12ff-434d-b03c-2fd4f2011b7f_0
29c128864758        ba2b418f427c                           "/bin/alertmanager -…"   8 days ago          Up 8 days                               k8s_alertmanager_alertmanager-main-2_monitoring_d4cea671-e7a8-4866-b28a-187d3fb4de6d_0
3390e1ecb43ec       ba2b418f427c                           "/bin/alertmanager -…"   8 days ago          Up 8 days                               k8s_alertmanager_alertmanager-main-1_monitoring_7359513c-12ff-434d-b03c-2fd4f2011b7f_0
2b1ff3650cc9        f56b05ba1b94                           "/bin/prometheus-con…"   8 days ago          Up 8 days                               k8s_config-reloader_alertmanager-main-0_monitoring_69bb4a53-3f34-4d50-8f7d-4507a510cb21_0
50cb82780e0c        ba2b418f427c                           "/bin/alertmanager -…"   8 days ago          Up 8 days                               k8s_alertmanager_alertmanager-main-0_monitoring_69bb4a53-3f34-4d50-8f7d-4507a510cb21_0
ad6e09d9bcfa        29589495df8d                           "/usr/local/bin/kube…"   8 days ago          Up 8 days                               k8s_kube-rbac-proxy_node-exporter-dtqkc_monitoring_78bc399c-ba83-4b48-b2a5-16fd5b1a5c0a_0
955c3e97a826        easzlab/pause-amd64:3.2                "/pause"                 8 days ago          Up 8 days                               k8s_alertmanager_alertmanager-main-2_monitoring_d4cea671-e7a8-4866-b28a-187d3fb4de6d_0
c9d1b917398f        easzlab/pause-amd64:3.2                "/pause"                 8 days ago          Up 8 days                               k8s_POD_alertmanager-main-1_monitoring_7359513c-12ff-434d-b03c-2fd4f2011b7f_0
f0bf1834c7ad        easzlab/pause-amd64:3.2                "/pause"                 8 days ago          Up 8 days                               k8s_POD_alertmanager-main-0_monitoring_69bb4a53-3f34-4d50-8f7d-4507a510cb21_0
1306aae3d6fb        1dbe0e931976                           "/bin/node_exporter …"   8 days ago          Up 8 days                               k8s_node-exporter_node-exporter-dtqkc_monitoring_78bc399c-ba83-4b48-b2a5-16fd5b1a5c0a_0
9464bccd6479        easzlab/pause-amd64:3.2                "/pause"                 8 days ago          Up 8 days                               k8s_POD_node-exporter-dtqkc_monitoring_78bc399c-ba83-4b48-b2a5-16fd5b1a5c0a_0
c8097f14a75c        29589495df8d                           "/usr/local/bin/kube…"   8 days ago          Up 8 days                               k8s_kube-rbac-proxy_prometheus-operator-7f5877bb57-tqx19_monitoring_db255f07-ca7b-4e1c-a0ad-ce21eaa1bf3c_0
556d639650a0        29589495df8d                           "/usr/local/bin/kube…"   8 days ago          Up 8 days                               k8s_kube-rbac-proxy_blackbox-exporter-676d976865-lwq8j_monitoring_cc54a1a8-2216-468c-b7f2-3801a3308df0_0
15ccacd49589        bd70f8b1b534                           "/bin/operator --kub…"   8 days ago          Up 8 days                               k8s_prometheus-operator_prometheus-operator-7f5877bb57-tqx19_monitoring_db255f07-ca7b-4e1c-a0ad-ce21eaa1bf3c_0
d32b80e2022d        easzlab/pause-amd64:3.2                "/pause"                 8 days ago          Up 8 days                               k8s_POD_prometheus-adapter-6cf5d8bfcf-zzg8d_monitoring_86cb819e-ca17-4463-87c1-c30ef5d8e785_0
4dee8bb32ed8        easzlab/pause-amd64:3.2                "/pause"                 8 days ago          Up 8 days                               k8s_POD_prometheus-adapter-6cf5d8bfcf-989r5_monitoring_eb9c31db-574d-4e20-953c-2ff8cc15b74b_0
5c52b5304a44        d771cc9785a1                           "/configmap-reload -…"   8 days ago          Up 8 days                               k8s_module-configmap-reloader_blackbox-exporter-676d976865-lwq8j_monitoring_cc54a1a8-2216-468c-b7f2-3801a3308df0_0
af6c77b18926        easzlab/pause-amd64:3.2                "/pause"                 8 days ago          Up 8 days                               k8s_POD_prometheus-operator-7f5877bb57-tqx19_monitoring_db255f07-ca7b-4e1c-a0ad-ce21eaa1bf3c_0
5e10cdff1d41        206d202cf866                           "/bin/blackbox_expor…"   8 days ago          Up 8 days                               k8s_blackbox-exporter_blackbox-exporter-676d976865-lwq8j_monitoring_cc54a1a8-2216-468c-b7f2-3801a3308df0_0
74f77811d6bb        easzlab/pause-amd64:3.2                "/pause"                 8 days ago          Up 8 days                               k8s_POD_kube-state-metrics-5d6885d89-4q777_monitoring_4c93d218-c79e-44fa-b1d1-f8881dde8599_0
fc482acc d7b0       bdc63515e8db                           "/run.sh"                8 days ago          Up 8 days                               k8s_grafana_grafana-55b8ff756-qr86x_monitoring_b11a8c4a-0e15-4dd5-99af-e8f8347ac020_0
b1c06eea3086        easzlab/pause-amd64:3.2                "/pause"                 8 days ago          Up 8 days                               k8s_POD_blackbox-exporter-676d976865-lwq8j_monitoring_cc54a1a8-2216-468c-b7f2-3801a3308df0_0
2bc0ea2c2b25        easzlab/pause-amd64:3.2                "/pause"                 8 days ago          Up 8 days                               k8s_POD_grafana-55b8ff756-qr86x_monitoring_b11a8c4a-0e15-4dd5-99af-e8f8347ac020_0
7f226932 3eef       e708f4bb69e3                           "/opt/bin/flanneld -…"   8 days ago          Up 8 days                               k8s_kube-flannel_kube-flannel-ds-amd64-9jwj6_kube-system_577693 6-4351-4377-922a-67cdffa11b34_1
e5f5275c219b        9dd718864ce6                           "/metrics-server --c…"   8 days ago          Up 8 days                               k8s_metrics-server_metrics-server-8568cf894b-8fvll_kube-system_6eda5f43-4c2e-4596-8fa6-6fb0f938ab31_1
01c09b1be6d1        48d79e554db6                           "/metrics-sidecar"       8 days ago          Up 8 days                               k8s_dashboard-metrics-scraper_dashboard-metrics-scraper-79c5968bdc-qswhl_kube-system_fc4e74d6-f646-4263-bd8a-30a61c361af3_1
b99c6df8cc8a        9a07b5b4bfac                           "/dashboard --insecu…"   8 days ago          Up 8 days                               k8s_kubernetes-dashboard_kubernetes-dashboard-c4c6564dd-7x2hv_kube-system_8a2979d3-8bdf-4d38-a5c3-fd2c613d37b1_1
175810aefd2b        90f0d984ec9a                           "/node-cache -locali…"   8 days ago          Up 8 days                               k8s_node-cache_node-local-dns-9znng_kube-system_d6d33cd7-5860-4634-8c20-60b4216cfc84_1
33a47add9720        0a6cfbf7b0b6                           "/coredns -conf /etc…"   8 days ago          Up 8 days                               k8s_coredns_coredns-5787695b7f-hs2kt_kube-system_57dcf3ef-0997-47fc-a5e5-0000cd39fc2a_1
dea534d628eb        easzlab/pause-amd64:3.2                "/pause"                 8 days ago          Up 8 days                               k8s_POD_kubernetes-dashboard_kubernetes-dashboard-c4c6564dd-7x2hv_kube-system_577693 6-4351-4377-922a-67cdffa11b34_1
7febd7dd9d82        easzlab/pause-amd64:3.2                "/pause"                 8 days ago          Up 8 days                               k8s_POD_dashboard-metrics-scraper-79c5968bdc-qswhl_kube-system_fc4e74d6-f646-4263-bd8a-30a61c361af3_1
591f7626178 9       easzlab/pause-amd64:3.2                "/pause"                 8 days ago          Up 8 days                               k8s_POD_metrics-server-8568cf894b-8fvll_kube-system_6eda5f43-4c2e-4596-8fa6-6fb0f938ab31_1
6d408fcf4d0         easzlab/pause-amd64:3.2                "/pause"                 8 days ago          Up 8 days                               k8s_POD_kubernetes-dashboard-c4c6564dd-7x2hv_kube-system_8a2979d3-8bdf-4d38-a5c3-fd2c613d37b1_1
79356a97fb69        easzlab/pause-amd64:3.2                "/pause"                 8 days ago          Up 8 days                               k8s_POD_node-local-dns-9znng_kube-system_d6d33cd7-5860-4634-8c20-60b4216cfc84_1
2e70123cb3ae        easzlab/pause-amd64:3.2                "/pause"                 8 days ago          Up 8 days                               k8s_POD_coredns-5787695b7f-hs2kt_kube-system_57dcf3ef-0997-47fc-a5e5-0000cd39fc2a_1
72f5ba897e0a        easzlab/kubeasz:3.0.0                  "sleep 36000"            2 weeks ago         Up About an hour                        kubeasz
root@VM-8-10-ubuntu:~#
```

FIGURE 7.4: Docker containers ID

Then, developers can find the containers they want to operate. Second, they can use "docker attach" to enter containers. But it is not good for developers. First is developers can not use two or more CMD to operate containers. If developers open two CMD windows, the output is synchronous in two windows. Second, if they exit containers, the containers will stop. We need to restart and docker Id will be changed. From the image we can notice that the times of restart is 1, because we use "docker attach": (Fig.7.5)

```
root@VM-8-10-ubuntu:~# kubectl get pods -n openfaas-fn
NAME                             READY   STATUS    RESTARTS   AGE
mltest-demo-69f8f88ff9-2t7sq   1/1     Running   0          45h
mltest-demo-69f8f88ff9-95qw4   1/1     Running   1          45h
root@VM-8-10-ubuntu:~#
```

FIGURE 7.5: Container restart times

7.2.2 Use SSH login container

In order to avoid the problem that developers only can operate by one CMD window, developers can install ssh service in containers. It seems to avoid problem by using "docker attach", but it brings more trouble.

If developers want to use SSH to login, they need to set username and password when they create containers. But what about developers want to renew password? They need to rebuild, redeploy and restart containers. Moreover, developers also need to sure that these operations will not destroy password. It is a big problem about the file which records password is destroyed. Finally, if the container collapse, it may not available to use SSH service. Developers also need another way to operate containers.

7.2.3 Use "docker exec" enter container

This is the most popular way to operate containers. We can use "docker exec –help" to check how to use it (Fig.7.6).

```
root@VM-8-10-ubuntu:~# docker exec --help

Usage:  docker exec [OPTIONS] CONTAINER COMMAND [ARG...]

Run a command in a running container

Options:
  -d, --detach               Detached mode: run command in the background
      --detach-keys string   Override the key sequence for detaching a container
  -e, --env list             Set environment variables
  -i, --interactive          Keep STDIN open even if not attached
      --privileged           Give extended privileges to the command
  -t, --tty                  Allocate a pseudo-TTY
  -u, --user string          Username or UID (format: <name|uid>[:<group|gid>])
  -w, --workdir string       Working directory inside the container
```

FIGURE 7.6: Docker exec help

Usually, developers use "docker exec -it (container id) /bin/bash" to enter containers. Here we want to give more details about "/bin/bash".

Because I use Ubuntu to finish experiment, "/bin/bash" is more fit Ubuntu. This command tells operate system using shell to run this command in linux. "sh" may more popular in other linux version. For example, "sh" means open "POSIX" of bash. "/bin/sh" is equal to "/bin/bash –posix". We add "/bin/bash" here means we want to use shell to operate containers.

7.3 OpenFaas and OpenFaas Pro

7.3.1 Modify configuration by labels

In the part of 5.1, we mentioned that there are two version of the OpenFaaS. In the official document, it compare these two versions: (Fig.7.7 Fig.7.8)

Features

Description	OpenFaaS CE	OpenFaaS Pro	OpenFaaS Enterprise
Dashboard	Basic, legacy UI portal (in code-freeze)	Dashboard with metrics, logs and multiple namespace support	As per Pro
Metrics	Basic HTTP invocation metrics	Plus advanced CPU/RAM usage metrics	As per Pro
Autoscaling granularity	Single rule for all functions	Custom per functions	As per Pro
Autoscaling strategy	RPS-only	CPU utilization, Capacity (inflight requests) or RPS	As per Pro
Authentication	Shared token with every user	Sign-On with OIDC Okta/Auth0	Custom Single Sign-On with your IdP
Scale to Zero	Not supported	Custom rule per function	As per Pro
Custom Kubernetes service account	N/a	Supported per function	As per Pro

FIGURE 7.7: Feature contrast [15]

Durability and reliability

Description	OpenFaaS CE	OpenFaaS Pro	OpenFaaS Enterprise
Health checks	Not supported	Custom HTTP path and intervals per function	As per Pro
Retry failed invocations	Not supported	Retry certain HTTP codes with a back-off	As per Pro
GDPR	Sensitive data printed in logs	Sensitive data omitted from logs	As per Pro
Grafana Dashboard	N/a	Supplied with advanced metrics in private repository	As per Pro
Secure isolation with Kata containers or gVisor	N/a	Supported via a runtimeClass	As per Pro

FIGURE 7.8: Durability and reliability contrast [15]

The official description of OpenFaaS Pro is OpenFaaS Pro is a commercially licensed distribution of OpenFaaS with additional features, configurations and commercial support from the founders. The reason why we introduce these two reason is developers are hard to modify developments' parameters. For example, Developers can not modify the scaling scope of pods after the developments have been developed. In the 5.1 experiment, we set the scale range from 10 to 4. Now, we deploy another deployment named "mltest-demo" (Fig.7.9).

```
root@VM-8-10-ubuntu:~/test# faas-cli deploy -f mltest.yml \  --label com.openfaa
s.scale.min=2
Deploying: mltest-demo.
WARNING! You are not using an encrypted connection to the gateway, consider usin
g HTTPS.

Deployed. 202 Accepted.
URL: http://10.68.168.215:8080/function/mltest-demo
```

FIGURE 7.9: Version test

And then we check the deployment's configuration (Fig.7.10).

```
metadata:
  annotations:
    prometheus.io.scrape: "false"
  creationTimestamp: null
  labels:
    com.openfaas.scale.min: "2"
    faas_function: mltest-demo
    uid: "632002394"
  name: mltest-demo
```

FIGURE 7.10: deployment's configuration

In next step, we are going to add max scale configuration (Fig.7.11).

```
root@VM-8-10-ubuntu:~/test# faas-cli deploy -f mltest.yml \  --label com.openfaa
s.scale.min=2 \ --label com.openfaas.scale.max=5
Deploying: mltest-demo.
WARNING! You are not using an encrypted connection to the gateway, consider usin
g HTTPS.

Deployed. 202 Accepted.
URL: http://10.68.168.215:8080/function/mltest-demo
```

FIGURE 7.11: Configuration changing

However, When we check the development configuration again, we notice that it is fail (Fig.7.12).

```
metadata:
  annotations:
    prometheus.io.scrape: "false"
  creationTimestamp: null
  labels:
    com.openfaas.scale.min: "2"
    faas_function: mltest-demo
    uid: "925455461"
  name: mltest-demo
```

FIGURE 7.12: Check configuration

It proves the official document. The version of OpenFaaS Pro can modify the deployment configuration easier.

7.3.2 Different kinds of restriction by labels

From the OpenFaaS official website, developers will notice there is comparison about OpenFaaS-CE and OpenFaaS Pro: (Fig.7.13)

Features

Description	OpenFaaS CE	OpenFaaS Pro	OpenFaaS Enterprise
Dashboard	Basic, legacy UI portal (in code-freeze)	Dashboard with metrics, logs and multiple namespace support	As per Pro
Metrics	Basic HTTP invocation metrics	Plus advanced CPU/RAM usage metrics	As per Pro
Autoscaling granularity	Single rule for all functions	Custom per functions	As per Pro
Autoscaling strategy	RPS-only	CPU utilization, Capacity (inflight requests) or RPS	As per Pro
Authentication	Shared token with every user	Sign-On with OIDC Okta/Auth0	Custom Single Sign-On with your IdP
Scale to Zero	Not supported	Custom rule per function	As per Pro
Custom Kubernetes service account	N/a	Supported per function	As per Pro

FIGURE 7.13: Scaling strategy restriction between two versions [15]

From the image, we can notice that developers can only restrict request times to control scaling in OpenFaaS-CE. If developers use OpenFaaS-CE, it needs to be attended that the labels of CPU is ineffective.

7.4 Prometheus rules

In Grafana, we download a template named Grafana Cloud Billing. However, it is not able to collect data. From searching, we find that it is important to change the Prometheus rules in configuration file about this template, because this template can calculate the cost of cloud server. There are also many others data which need set by developers. Developers can change the configuration under this direction address: (Fig.7.14)

```
root@VM-8-10-ubuntu:~/kube-prometheus/manifests# vim nodeExporter-p
rometheusRule.yaml
```

FIGURE 7.14: Prometheus configuration direction address

7.4.1 Prometheus-k8s export forwarding

In chapter 6, we mentioned the Prometheus export forwarding. It is used to monitor host though Prometheus, so it is not enough for our experiment. Exactly, the Prometheus-K8s is a component which is installed in node to monitor other pods running status, so we can not forwarding directly. In details, we need to forward it from pod to host. Then, if we want to access it by external port, we also need forward it from host to external interface. Developers can set it by using "kubectl edit service" (Fig.7.15)

```
  creationTimestamp: "2022-03-27T12:51:48Z"
  labels:
    app.kubernetes.io/component: prometheus
    app.kubernetes.io/instance: k8s
    app.kubernetes.io/name: prometheus
    app.kubernetes.io/part-of: kube-prometheus
    app.kubernetes.io/version: 2.34.0
  name: prometheus-k8s
  namespace: monitoring
  resourceVersion: "3726678"
  uid: f8b1ca37-6a63-4dd7-b760-29c2727fd369
spec:
  clusterIP: 10.68.242.4
  clusterIPs:
  - 10.68.242.4
  externalTrafficPolicy: Cluster
  ports:
  - name: web
    nodePort: 30871
    port: 9090
    protocol: TCP
    targetPort: web
  - name: reloader-web
    nodePort: 31405
    port: 8080
    protocol: TCP
    targetPort: reloader-web
  selector:
    app.kubernetes.io/component: prometheus
    app.kubernetes.io/instance: k8s
    app.kubernetes.io/name: prometheus
    app.kubernetes.io/part-of: kube-prometheus
```

FIGURE 7.15: Kubectl edit service

7.4.2 Python version trouble

In Python2, we can use "print" function directly. But in python 3 we need add () to the text.

In python 3, the function "cPickle" change to "pickle". We also need to care about the class of data when we use 'pickle.load' function. The default decoding function of 'pickle.load' is "ASCII". But the data which we use is not "ASCII". We need to change it to byte. Here are the error we met and how to solve it: (Fig.7.16 Fig.7.17)

```
<ipython-input-3-d6922f440bef> in __init__(self, filenames, need_shuffle)
      9         all_labels = []
     10         for filename in filenames:
---> 11             data, labels = load_data(filename)
     12             for item, label in zip(data, labels):
     13                 if label in [0, 1]:
```

FIGURE 7.16: Deep learning 1

```
with open(os.path.join(CIFAR_DIR, "data_batch_1"),'rb')as f:
    data = pickle.load(f,encoding='bytes')
    print (type(data))
    print (data.keys())
    print (type(data[b'data']))
    print (type(data[b'labels']))
    print (type(data[b'batch_label']))
    print (type(data[b'filenames']))
```

FIGURE 7.17: Deep learning 2

And others' function not need encoding such as load_data. (The image code is error output: (Fig.7.18))

```
        for filename in filenames:
            data, labels = load_data(filename, encoding='bytes')
            for item, label in zip(data, labels):
                if label in [0, 1]:
                    all_data.append(item)
                    all_labels.append(label)
        self._data = np.vstack(all_data)
        self._labels = np.hstack(all_labels)
```

FIGURE 7.18: Error output

In the Tensorflow of version 2, it has strict requirement to data type. Before we train the data model, we need to translate data to the array of Numpy.

```
# evaluate the model
error = model.evaluate(X_test, y_test, verbose=0)
print('MSE: %.3f, RMSE: %.3f' % (error, sqrt(error)))
# make a prediction
row = [0.00632,18.00,2.310,0,0.5380,6.5750,65.20,4.0900,1,296.0,15.30,
yhat = model.predict([row])
print('Predicted: %.3f' % yhat)
```

```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to
find data adapter that can handle input: <class 'pandas.core.frame.DataFr
ame'>, <class 'NoneType'>
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to
find data adapter that can handle input: <class 'pandas.core.frame.DataFr
ame'>, <class 'NoneType'>
MSE: 71.035, RMSE: 8.428
Predicted: 27.961
```

FIGURE 7.19: Error output 2

Here is the warning when I run code: (Fig.7.19)

We can check the type of data is "Pandas data frame": (Fig.7.20)

```
In [69]: type (X_test)
Out[69]: pandas.core.frame.DataFrame
```

FIGURE 7.20: Pandas data frame

We translate the data to Numpy model, there is no error: (Fig.7.21)

```
In [70]: X_test = np.array(X_test)
         y_test = np.array(y_test)
         X_train = np.array(X_train)
         y_train = np.array(y_train)
```

```
model.fit(X_train, y_train, epochs=150, batch_size=32, verbose=0)
# evaluate the model
error = model.evaluate(X_test, y_test, verbose=0)
print('MSE: %.3f, RMSE: %.3f' % (error, sqrt(error)))
# make a prediction
row = [0.00632,18.00,2.310,0,0.5380,6.5750,65.20,4.0900,1,296.0,15.30,
yhat = model.predict([row])
print('Predicted: %.3f' % yhat)
```

```
MSE: 41.629, RMSE: 6.452
Predicted: 29.000
```

FIGURE 7.21: Correct code

Chapter 8

Conclusion and future work

In the thesis, we explain the details of each OpenFaaS components. We explore simple deep learning knowledge. Use a data set to train a model which is used to predict accuracy. Deploying this deployment on OpenFaaS and using experiment to make readers learn how to operate system. Moreover, the paper introduce the data processing flow. It makes people understand data processing rules which avoid people get into trouble when they use OpenFaaS themselves. Finally, the paper comparing running time and system running status in many different situation. From the result, we get the resource consumption and latency in different phrase. Listing places which will effect performance and give advice to optimize it. However, there are also many exploration which we can do in the future. Here is the list about them:

1. The OpenFaaS system we deployed just on local host. As we know, the OpenFaaS system is designed for distributed systems. In next step, we are going to try deploy them in many host or creating many containers which connect to each other just in one host. Watching and analyzing how data process in that distribute systems.

2. The paper only give advice about where we can analysis system process efficiency, but we can not give details about how to do it in specific operations. In next step, we can analysis running logs or other method to analysis what operation can shorten latency and reduce CPU and memory consumption.

3. OpenFaaS system are supported in these years. If developers want to use it, they need follow the frame and rules of OpenFaaS. In this condition, many system or

application can not be built on OpenFaaS. In the future, we can find a convenient way to help developers conversion the platform they used to OpenFaaS platform.

Bibliography

[1] Ivan Velichko. Kubernetes concepts. https://iximiuz.com/en/posts/openfaas-case-study//, 2021.

[2] Alex Krizhevsky. Cifar-10. https://www.cs.toronto.edu/~kriz/cifar.html, 2009.

[3] unixfbi.com. Vm and docker. https://https://blog.51cto.com/shijianfeng/2915015, 2016.

[4] stardsd. Kubernetes orchestration. https://blog.51cto.com/sddai/3134384, 2022.

[5] Kubernetes.io. Kubernetes pods and node. https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/, 2022.

[6] xinchen. Openfaas service. https://blog.csdn.net/boling_cavalry/article/details/109805296, 2022.

[7] xinchen. Faas-netes. https://blog.csdn.net/boling_cavalry/article/details/109971608, 2022.

[8] Material for MkDocs. Watchdog. https://docs.openfaas.com/architecture/watchdog/, 2022.

[9] Kubernetes official websit. Kubernetes cluster concepts. https://kubernetes.io/zh/docs/concepts/overview/components/, 2022.

[10] Kubernetes official website. Horizontal. https://kubernetes.io/zh/docs/tasks/run-application/horizontal-pod-autoscale/, 2022.

[11] Kubernetes official website. svc-exporter. https://kubernetes.io/zh/docs/tasks/access-application-cluster/port-forward-access-application-cluster/, 2022.

[12] CNCF Member Blog. prometheus. https://www.cncf.io/blog/2020/04/24/prometheus-and-grafana-the-perfect-combo/, 2022.

[13] OpenFaaS official website. auto-scaling. https://docs.openfaas.com/architecture/autoscaling/, 2022.

[14] Docker official website. Docker daemon. https://docs.docker.com/get-started/overview/, 2022.

[15] OpenFaaS official website. Openfaas pro. https://docs.openfaas.com/openfaas-pro/introduction/, 2022.

[16] Christian Posta. Faas vs. microservices. 2022. URL https://dzone.com/articles/faas-vs-microservices.

[17] Sourav Mukherjee. Benefits of aws in modern cloud. *Available at SSRN 3415956*, 2019.

[18] Peter Garraghan, Paul Townend, and Jie Xu. An analysis of the server characteristics and resource utilization in google cloud. In *2013 IEEE International Conference on Cloud Engineering (IC2E)*, pages 124–131. IEEE, 2013.

[19] Openfaas workshop. https://github.com/openfaas/workshop, 2022.

[20] Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. Serving deep learning models in a serverless platform. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 257–262. IEEE, 2018.

[21] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, et al. Serverless computing: Current trends and open problems. In *Research advances in cloud computing*, pages 1–20. Springer, 2017.

[22] Sunil Kumar Mohanty, Gopika Premsankar, Mario Di Francesco, et al. An evaluation of open source serverless computing frameworks. In *CloudCom*, pages 115–120, 2018.

[23] Lele Ma, Shanhe Yi, and Qun Li. Efficient service handoff across edge servers via docker container migration. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, pages 1–13, 2017.

[24] Nicole Rusk. Deep learning. *Nature Methods*, 13(1):35–35, 2016.

[25] Shervin Minaee, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu, and Jianfeng Gao. Deep learning–based text classification: a comprehensive review. *ACM Computing Surveys (CSUR)*, 54(3):1–40, 2021.

[26] Carl Boettiger. An introduction to docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1):71–79, 2015.

[27] Subrota Kumar Mondal, Rui Pan, HM Kabir, Tan Tian, and Hong-Ning Dai. Kubernetes in it administration and serverless computing: An empirical study and research challenges. *The Journal of Supercomputing*, 78(2):2937–2987, 2022.

[28] Kubernetes concepts. https://kubernetes.io/docs/concepts/, 2020.

[29] David Bernstein. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84, 2014.

[30] James Turnbull. *Monitoring with Prometheus*. Turnbull Press, 2018.

[31] Junfeng Li, Sameer G Kulkarni, KK Ramakrishnan, and Dan Li. Understanding open source serverless platforms: Design considerations and performance. In *Proceedings of the 5th international workshop on serverless computing*, pages 37–42, 2019.

[32] David Balla, Csaba Simon, and Markosz Maliosz. Adaptive scaling of kubernetes pods. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–5. IEEE, 2020.

[33] Paulo Silva, Daniel Fireman, and Thiago Emmanuel Pereira. Prebaking functions to warm the serverless cold start. In *Proceedings of the 21st International Middleware Conference*, pages 1–13, 2020.

[34] Mainak Chakraborty and Ajit Pratap Kundan. Grafana. In *Monitoring Cloud-Native Applications*, pages 187–240. Springer, 2021.

[35] Bayo Erinle. *Performance testing with JMeter 2.9*. Packt Publishing Ltd, 2013.

[36] Ruchika Muddinagiri, Shubham Ambavane, and Simran Bayas. Self-hosted kubernetes: deploying docker containers locally with minikube. In *2019 International Conference on Innovative Trends and Advances in Engineering and Technology (ICITAET)*, pages 239–243. IEEE, 2019.

Acknowledgements

This paper is written by Haocheng Wang who is a senior student in Macao University of Science and Technology. The tutor is Subrota Kumar Mondal. In the last year in MUST, I am glad to be the student of my tutor. He teach me how to do a completed research and give me many advice about how to finish a perfect paper. At first, I think the thesis is too difficult for me. In the second half of the thesis, I try to stop research many times, but the tutor always give me advice and try to pus me do more research. Finally, I have done this excellent work which is more than I expected. Many thanks for him to spend much time to teach me. I also grateful to the university. There is just one month to graduate for me. Looking back four years I spent in college, I become more knowledgeable and mature. I met many teachers who teach me a lot. When I ask them question about courses during office time, they teach me patiently. These all make me unforgettable. Sincerely hope them reach a higher level and get more achievement in academic. Finally, the best wish to me about achieving more in the next stage of study.

Resume

Haocheng Wang
Macau University of Science and Technology
Avenida Wai Long, Taipa, Macau
+86-13016319829
whc.must@gmail.com

DATE OF BIRTH:
August 29, 2000

EDUCATION:
SEPTEMBER 2018 - PRESENT
Bachelor of Science in Computer Technology and Application at Macau University of Science and Technology, Macau S.A.R. (Expected date of graduation, July 2022)

RESEARCH EXPERIENCE

FEB.2020 – JUN.2020
Cisco lab network deployment
Learn the various ways about deploy local area network and wide area network. Learn and practice how to set each network sever and know about their working principle. Practice to set a network company network facility and implement to provide network sever to the cities.

SEP.2020 – DEC.2020
Image processing algorithm development
Learn the process of the signal modulate and processing.
Know about the basement of modifying and processing image.
Learn some basic method of processing image and finding its disadvantage.
Creating some new method to develop basic imaging process algorithm and implement.

Circuit Analysis and Simulation
Calculate and employ the common electronic component.
Do the experiment about electronic equipment and analysis signal.
Design the MUC and deploy it in life scene.

JUL. 2021 – SEP. 2021

Cryptography and Cybersecurity and Blockchains Research Seminar(CMU program)

Gained a solid background in the basics of cryptography and cyber security.
Understood why all the classical ciphers were broken and the philosophy of modern cryptography.
Studied SSL and HTTPS protocols used widely by every web browser for secure browsing today.
Studied blockchains and cryptocurrencies, including how Bitcoin mining works.

SEP. 2021 – NOV. 2021

Develop Applet in WeChat to Reserve Seats in the Library

Learned all of the required programming skills by myself.
Know about the whole process about the program developing.
Familiar with the process of how to get information from app users.
Learn how to mining users requirement and develop appropriate functions to them.
Learn how to choice develop ways to realize users' requirement quickly and well.

NOV.2021 – APR.2022

Serverless deep learning and analysis

Know about the cloud computing structure and containerization technology.
Learn the basic algorithm about deep learning and employ it on the Serverless platform.
Do deep learning model which deployed on the Serverless platform performance and research how to improve it.

INTERNSHIP EXPERIENCE:

Harbin Hytera Technology Co. Ltd.
Intern at Department of Hardware Development

JUL. 2020 – AUG. 2020

Understood of the product development process and basic hardware development operations.
Completed the design and fabrication of small-scale communication hardware.
Troubleshoot and came up with suitable solutions step by step through data material and knowledge of circuits.

Zhuhai Jafron Biotechnology Co. Ltd.
Intern at Software development

APR.2022-NOW

Operation and maintenance of the sales platform.
WeChat applet function development.

EXTRACURRICULAR ACTIVITIES:

2019 Vice President of the Dance Federation of the Macau University of Science and Technology Students' Union

2019 Awarded as the Advanced Open Water Diver(PADI)

2018 Warm Winter Education Support Activities of Macau University of Science and Technology

2022 **Paper published,** Applications of the Decentralized Finance (DeFi) on the Ethereum, 2021 International Conference on Picture Processing, Robotics and Artificial Intelligence