DEPARTMENT OF COMPUTER SCIENCE
THE UNIVERSITY OF WESTERN ONTARIO
Directed Study Report

A Serverless Approach to Building and Deploying Pipeline

Kun Wang
SUPERVISOR: Dr . Hanan Lutfiyya

Introduction

Cloud computing has been widely used in the technology industry, due to its low cost and high scalability. With the rapid growth of microservices architecture, serverless computing has become a hot topic. Serverless computing is a cloud computing execution model. The most advantage of this model is that as a developer, you can put most of your effort into developing the function and don't need to spend too much time worrying about the servers. The cloud provider takes care of the server part for the developers. In this project, we used FaaS(Function as a service) model to deploy a pipeline in a serverless manner.

One of the challenging parts of this project is getting familiar with different frameworks, structures, and packages and integrating them to provide new functionalities. In this project, we used Docker, Kubernetes, open source framework OpenFaas, open source asynchronous task queue Celery, open source message broker RabbitMQ, Node-RED - a low-code programming tool for event-driven applications, Redis, and some packages like pika, Pydictionary, and Googleletrans. It was a very pleasant journey to learn new software tools which are also very useful for my future career and integrate them into a project.

The contribution of this project is to implement a relatively new approach to building and deploying a pipeline. This approach is based on a serverless architecture. I will elaborate on this approach in the framework and implementation section.

Related Work

There are some platforms that provide similar services such as AWS Lambda. AWS Lambda is a serverless, event-driven computer service that allows customers to run their functions virtually any type of application without managing servers. This is a very powerful tool. However, there is a limitation of AWS Lambda. It does not provide access for developers to control the deployment of functions. In another word, as a developer, you won't be able to know how and where the functions are deployed after the deployment since lambdas are deployed in the AWS region. For our purpose in this project, we need to have control over where and how the functions are deployed since we will need to manage the pipeline. This is the main purpose of this project, which is to build a new approach to deploy a pipeline base on OpenFaaS which is very similar to the AWS Lambda. OpenFaaS also has many templates and provides a simple way for developers to deploy their functions in a serverless manner.

Design Framework

1. Overall Framework

The design of the framework for this project is shown in figure 1. There are mainly three parts of works in this project. First, the user can break down a monolithic function into serval relatively small functions. These functions can be built into Docker images and deployed to the OpenFaaS. Second, according to the original monolithic function, dependencies among these small functions can be implemented into the DAG.

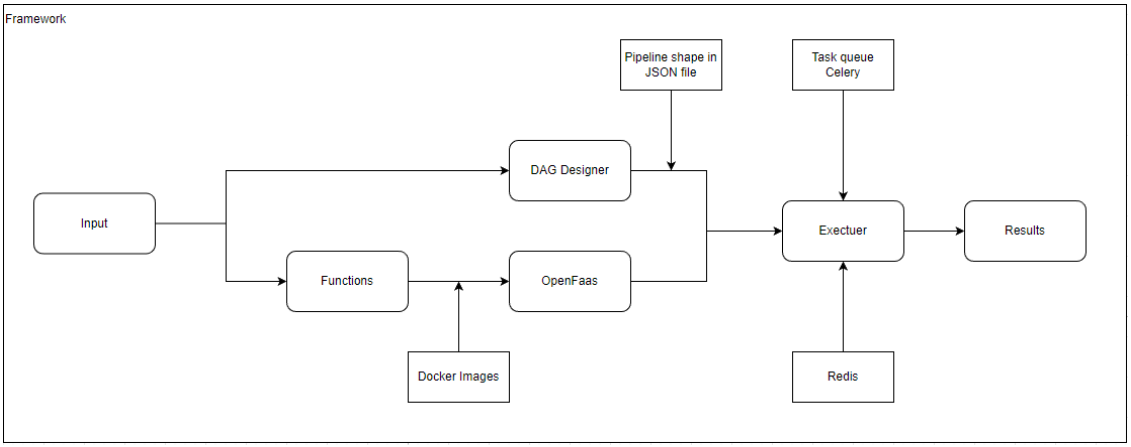

Figure 1: The Overall Framework

The pipeline shape in the DAG also specifies the direction of the input and output of each function. The DAG Designer exports a JSON file which will later be used in the Executor function. Third, the Executor function then can execute the pipeline based on the JSON file and Docker images on OpenFaas. This part involves using RabbitMQ and Celery to build a publisher and subscriber to process the pipeline asynchronously.

2. DAG Designer

One of the three parts of this project is the DAG Designer shown in figure 2. In this part of the project, the user can draw a DAG of a pipeline. This DAG of the pipeline specifies the number of tasks, the name of the tasks, and the dependencies among the tasks. The DAG Designer is built based on Node-RED which is a low-code programming tool for event-driven applications. Node-RED is a very powerful tool that provides many different functions.

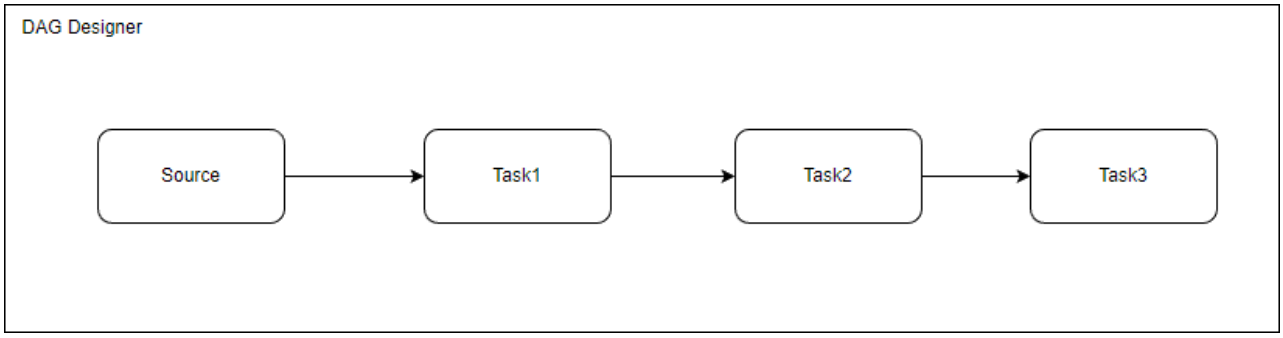

Figure 2: An example DAG

For this project, we only used it to draw a pipeline DAG and convert it to a JSON file as shown in figure 3. This JSON example shows the pipeline id, name, and status. All the tasks have their name, sequence id, input, and output. The connections indicate dependency. It shows where is the input from and where the output goes. In this way, the Executor function can execute the pipeline asynchronously. The source task contains the initial input for task 1.

Figure 3: An example of DAG in JSON

3. Function Builder

The function builder as shown in Figure 4 puts the functions into a Docker image and deploy them on the OpenFaaS. After breaking down a monolithic function into many small functions, each piece of the monolithic function needs to be deployed separately to the OpenFaaS.

It will be easier to understand this part of the project if we dig more into OpenFaaS. OpenFaaS is an open-source functions framework. It makes it easier for developers to deploy functions and microservices to Kubernetes without repetitive and template coding. Developers can package the code into a Docker image and run functions on any cloud. The user can also choices to use any programming language to build their tasks.

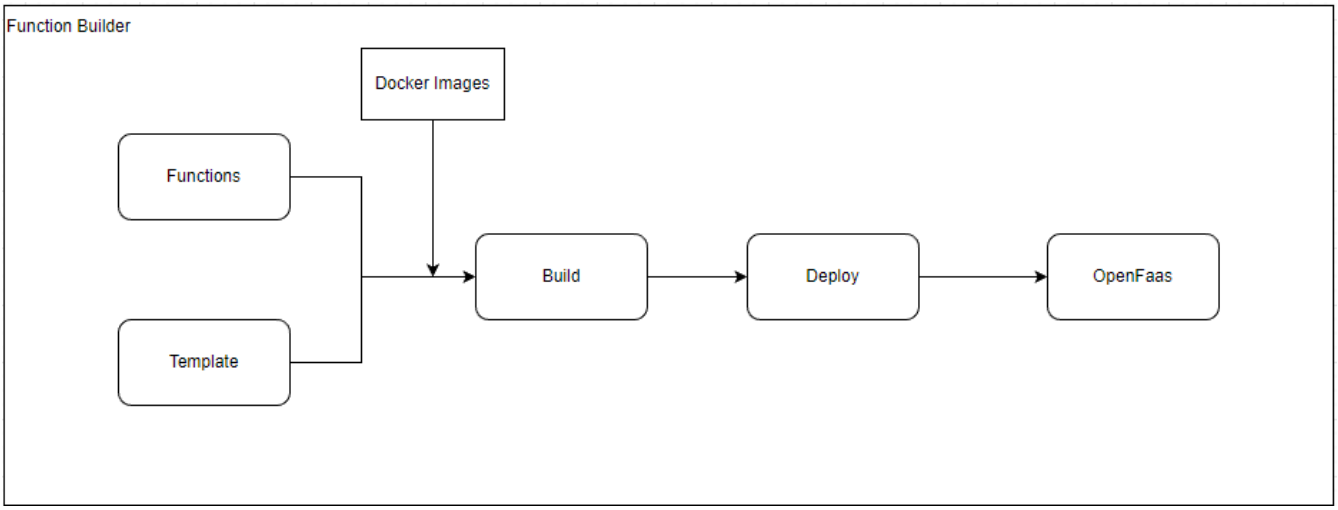

Figure 4: A diagram shows the Function Builder

There are totally three layers on the OpenFaas as shown in Figure 5. The uppermost layer is CI/GitOps Layer. OpenFaas allows the user to run HTTP microservices. However, in this project, during the development and deployment, it is done manually using faas-cli. For example, faas-cli deploy or faas-cli up are used to deploy the functions. The command line faas-cli build is used to build the image; faas-cli push is used to push the image to the registry.

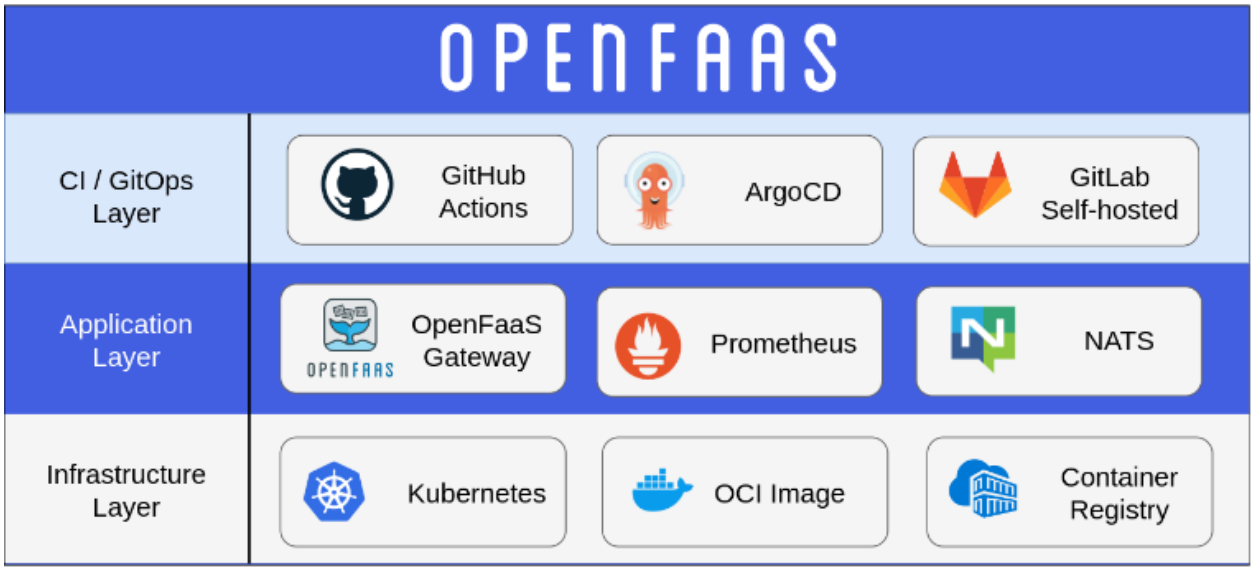

Figure 5: OpenFaas Stack

The second layer is the Application Layer which has three options (OpenFaas Gateway, Prometheus, and NATS). This project only uses the OpenFaas Gateway. As shown

in Figure 6, the OpenFaaS API Gateway provides a route into the functions. It also has a built-in UI shown in Figure 7 which can be used to deploy user-customized functions. In our case, the user-customized functions are these tasks in the pipeline. After we deploy the function of tasks on OpenFaaS, the API Gateway will create one-to-many containers. The user can invoke the function and get the return results through the built-in UI in which pytran function translates the word golden retriever to Japanese.

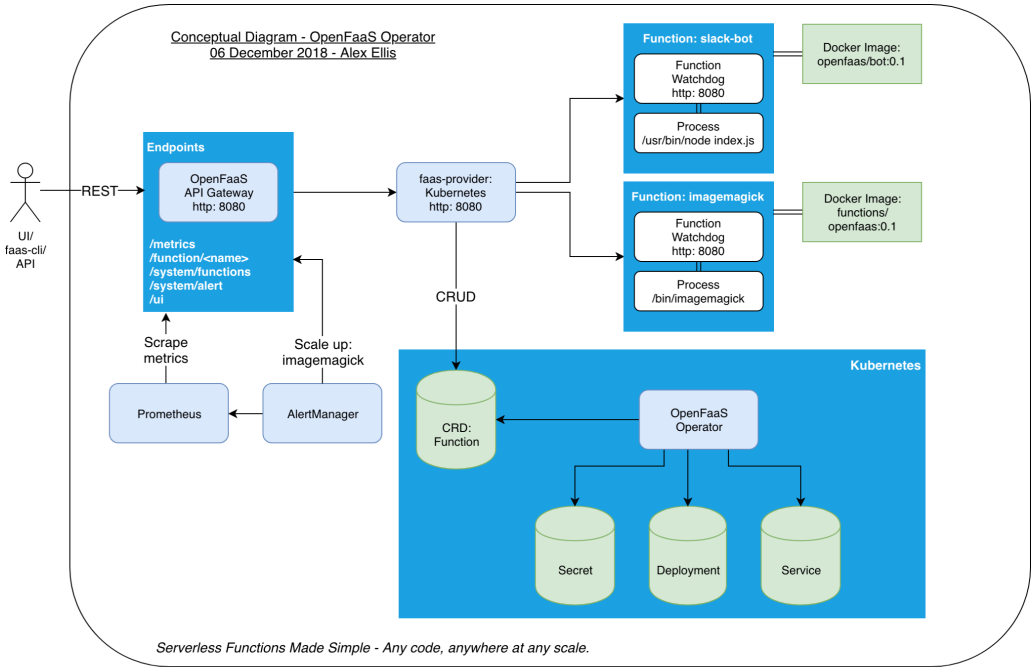

Figure 6: Conceptual Diagram - OpenFaaS Operator

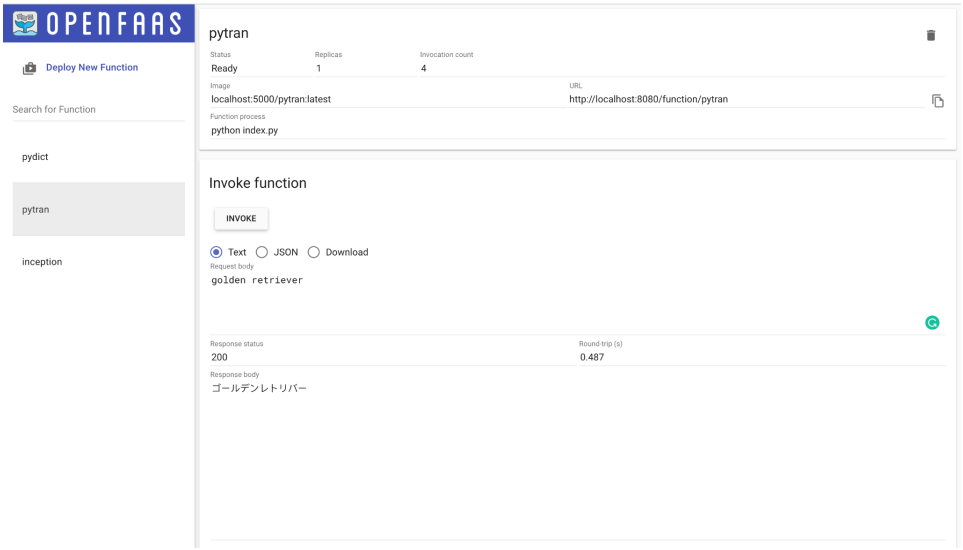

Figure 7: OpenFaaS Gateway UI

4. DAG Executor

There are mainly three stacks used in DAG Executor as shown in figure 8. They are Redis, Celery, and RabbitMQ. Redis is implemented to read the DAG shape which is expatiated in a JSON file and store the return results. Celery as mentioned before is an open-source task queue that is used as a mechanism to distribute work across threads or machines. Celery communicates through messages which means it needs a message broker. We use RabbitMQ as the message broker in this project. After the Redis receives the initial input and name of the first function, it passes to the Celery. Celery then invokes the function which is deployed on the OpenFaaS. The output of the function is from the OpenFaaS and the message that contains the output of the previous tasks is published through RabbitMQ. After the subscriber receives the message, it sends it to Redis. Redis stores the result and passes data to Celery to execute the next task.

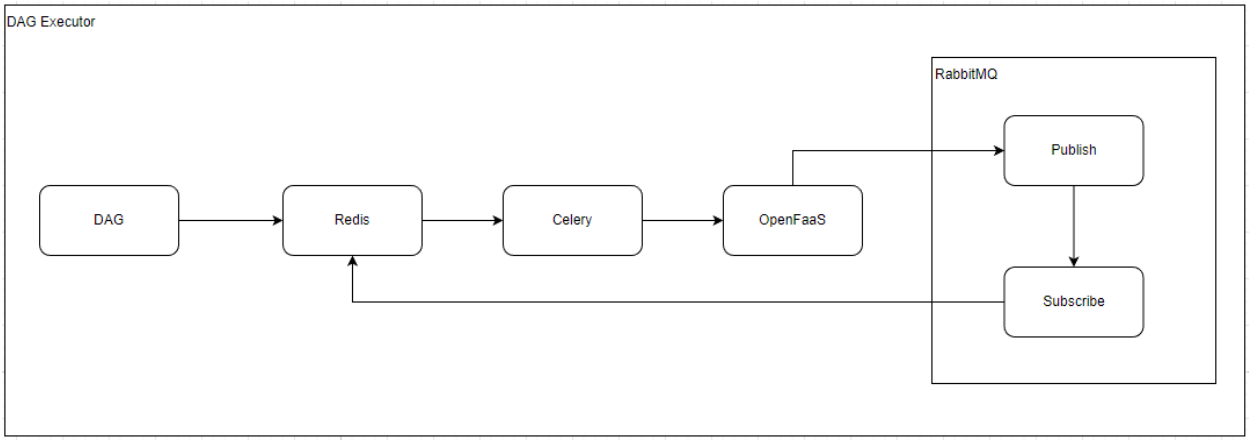

Figure 8: DAG Executor

Implementation

1. Use Case

Sometimes it could be hard to tell a dog's breed even for a dog lover like me. In this project, the application we built can recognize and interpret a dog's image. The user only needs to input an URL of a dog image to the application. The application can show the breed of the dog, a couple of sentences to describe the dog and translate the dog's bread into other languages other than English in case the user does not know English. This application is shown in figure 9.

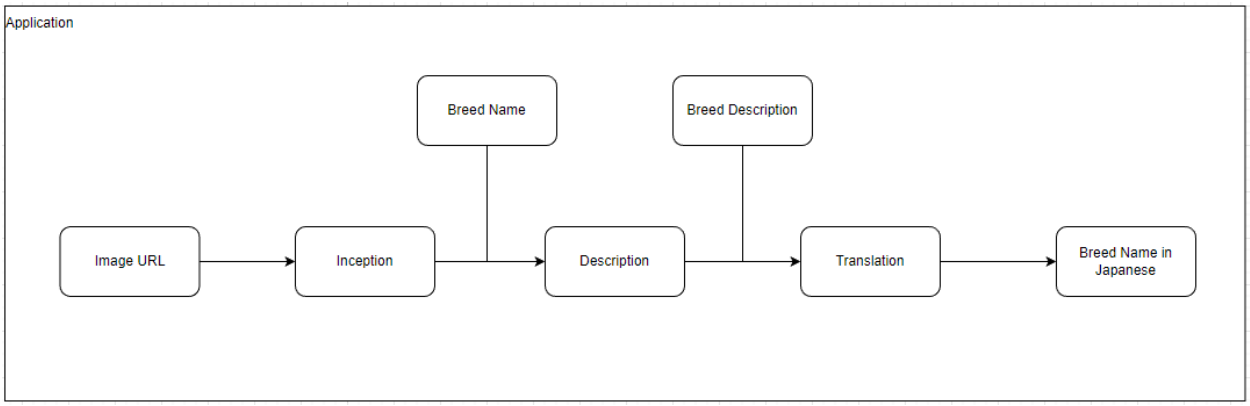

Figure 9: The application framework

2. Inception Function

The inception function is one of the OpenFaaS official functions. It is a forked version of an example project from the TensorFlow Tutorials and has been re-packaged as an OpenFaaS serverless function. This function uses a pre-train model called Inception v3. The original model is an image classification model and is trained on a Convolutional Neural Network with many layers and a complicated structure. According to the paper, it takes weeks to train on a monster computer with 8 Tesla K40 GPUs. Therefore, it is impossible to train it on my own computer. However, we can download the pre-trained model, deploy it on the OpenFaaS, and use it as an image classifier function.

3. Description Function

The inception function outputs a list of breeds' names with probability. The description function takes the breed's name on the top which has the largest probability and outputs a brief description of it. The description function is built based on the PyDictionary package. It helps to get meanings and synonyms of words.

4. Translation Function

The last function is the translation function. It receives the breed's name from the previous function and then translates it into any language. We use Japanese in this use case. The Translator package which is from Googleletrans is used in this function.

Evaluation

1. Deployment Environment

To deploy this project, there are some tools and packages that need to be installed first. The project is built on Python 3 environment. The user has to have Docker and Kubernetes installed on the machine. This project currently can only work on the local machine. Therefore, we also need to install KinD which provides a shell script to create a Kubernetes cluster along with a local Docker registry. To deploy functions on OpenFaaS, the user also needs to port-forward the OpenFaas gateway service to localhost.

2. Performance

Figure 10 shows the results of the use case we used in this project. The “received” shows the output of every function. The whole program takes about one minute to run and get the result.

Figure 10: Use case result

Conclusion

The application of this project is to predict the dog breed from a dog image and translate the breed name into Japanese. We implemented a serverless approach to building and deploying this application into a pipeline.

Acknowledgment

I would like to express my gratitude and thanks to professor Hanan Lutfiyya for inspiring my interest in this area. Special thanks to Mr. Muhamed Alarbi for helping me build this application.

References

https://docs.celeryq.dev/en/stable/getting-started/introduction.html
https://www.rabbitmq.com/
https://www.openfaas.com/
https://github.com/faas-and-furious/inception-function
https://yankee.dev/serverless-function-openfaas-kubernetes-locally
https://arxiv.org/pdf/1512.00567v3.pdf