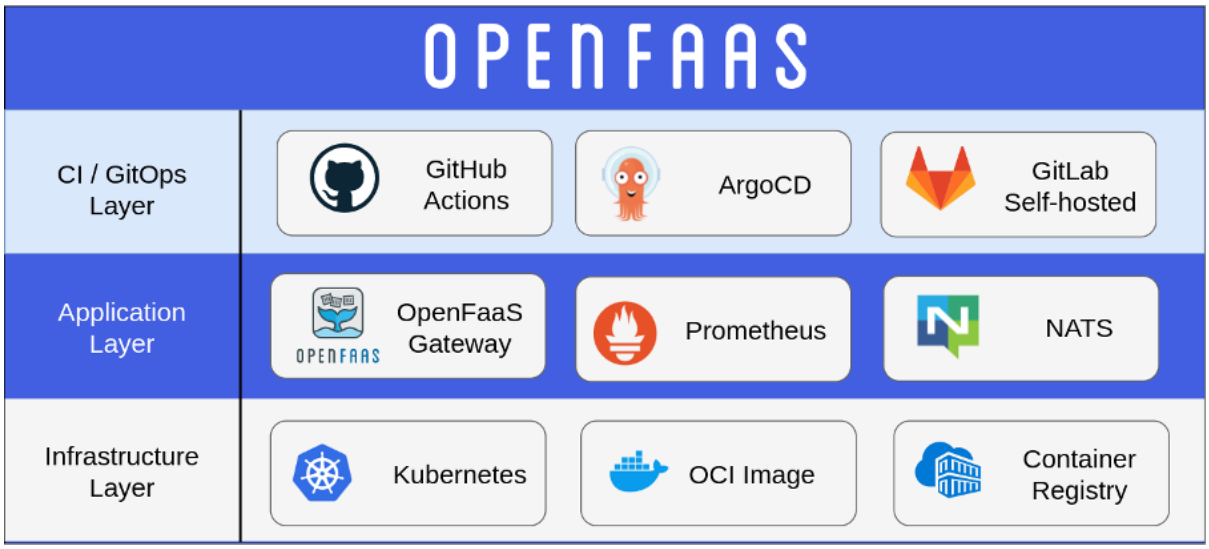
Deployment and Optimization of Machine Learning Models in a Serverless Architecture

By Ngai Lai Yeung and Zhang Zi Di

1. Introduction

This project focuses on deploying a CNN-based model for CIFAR-10 image classification task using OpenFaaS. OpenFaaS is an open-source framework that allows developers to easily deploy and manage serverless functions in a variety of environments.

It provides a powerful and flexible environment for building serverless applications, enabling developers to easily create and manage functions while focusing on their core business logic. Its open-source nature and active community contribute to its growing popularity in the serverless ecosystem. The following figure shows the structure of the OpenFaaS [1]



2. System Design and Implementation

The system contains design of the OpenFaaS and the CNN model for image classification:

- **2.1 OpenFaaS structure design:**
    - Infrastructure Layer: Local Kubernetes cluster (simulated via KinD).
    - Application Layer: OpenFaaS Gateway
    - CI / GitOps Layer: None (Developed locally)

- **2.2 CNN Model design:**

  - Target : Image classification task for CIFAR-10 dataset

  - Architecture:
    - Convolutional layer with 32 filters (ReLU activation)
    - Dropout layer
    - Convolutional layer with 32 filters (ReLU activation)
    - MaxPooling layer (2x2)
    - Convolutional layer with 64 filters (ReLU activation)
    - Dropout layer
    - Convolutional layer with 64 filters (ReLU activation)
    - MaxPooling layer (2x2)
    - Fully connected layer

  This CNN architecture is used for image classification tasks, leveraging multiple convolutional layers, activation functions, and pooling layers to extract and learn features from input images.

  It is designed to effectively learn hierarchical patterns from images, starting from low-level features (like edges) and progressing to high-level abstractions.

  The use of dropout layers, and max pooling layers combine is to minimizing the risk of overfitting.

- **2.3 Deployment Workflow**

  - Modifications to the Dockerfile to accommodate Conda and PyTorch installation.

  - Modifications to the Handler.py to handle the request.

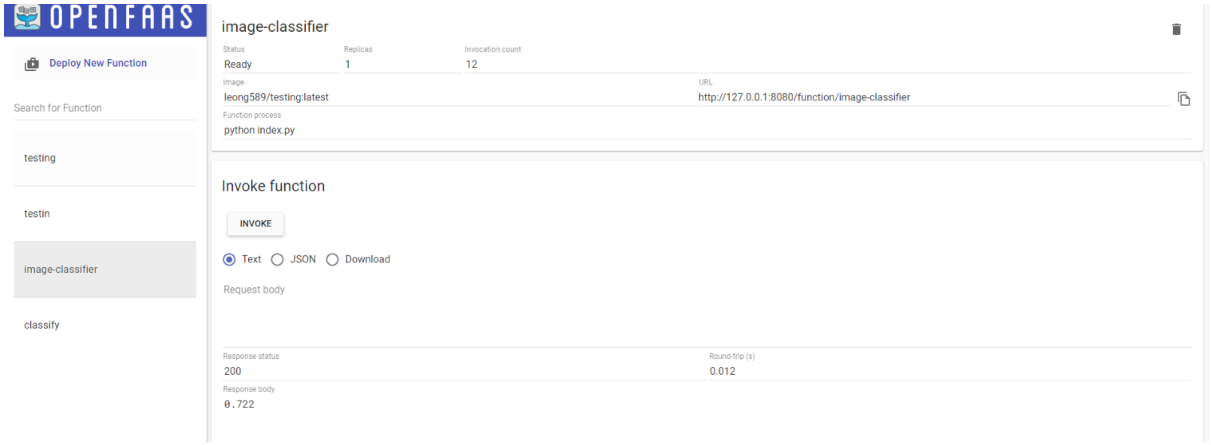
  - The model code is containerized as a Docker image and deployed using faas-cli.
    - faas-cli build -f template.yml

3. Experiments and Results

As we have installed OpenFaaS locally, we can open the OpenFaaS portal, which has been port-forwarded to 127.0.0.1:8080/ui/, directly. We can navigate to the Functions tab and invoke the function. Alternatively, we can simulate a web request by accessing http://127.0.0.1:8080/function/image-classifier.

Our function behaves as follows: when there is a request, whether it is a GET or POST request, it will return the test accuracy of the CNN model. The accuracy is 72.2% in 3 epoch.

The following figure is a demonstration of our function:



4. Challenges

- **4.1 Dependency Issues**:
  - **Challenge**: PyTorch installation caused conflicts while using pip.
  - **Solution**: We modified the Dockerfile to install Conda during the image-building process. This allows us to manage dependencies using Conda, effectively resolving version conflicts.
- **4.2 Slow Docker Image Push**:
  - **Challenge**: Each debugging iteration requires building and pushing a new image. This process consumes a lot of time (approximately 15 minutes, even with caching) because we include the dataset needed to train the CNN model during the image building process.
  - **Solution**: Wait. It slows down our development and debug process.
- **4.3 Limited Backend Development Experience**:
  - **Challenge**: We only have two members, and both of us lack familiarity with handling HTTP requests in Flask and OpenFaaS. Specifically, OpenFaaS utilizes Flask as its backend routing technique.
  - **Solution**: Focused on implementing a basic response method to achieve core functionality.
- **4.4 Limitations of Serverless Architecture**:

- **Challenge**: OpenFaaS is not suitable for long-running tasks due to connection timeouts. I attempted to have OpenFaaS train the model while simultaneously receiving a request and then respond with the model's results. However, the connection was lost, resulting in an internal server error.

- **Solution**: Avoid model training tasks while handling the request, but print the result directly.

5. Discussion

- **5.1 Advantages of OpenFaaS**:

  - **Improved developer velocity**: With OpenFaaS, developers can bypass the need to build a traditional web server (e.g., using Flask for web request handling and routing). This allows for a greater focus on application logic without worrying about compatibility across different platforms.

  - **High scalability:** OpenFaaS offers significant scalability, effectively managing spikes in traffic and scaling down resources when idle.[1]

  - **Cost-efficient:** The server provider can automatically scales based on demand. This leads to minimal resource consumption during idle periods, making it a cost-effective solution for running applications.

- **5.2 Disadvantages of OpenFaaS**:

  - **Less system control:** Installing additional system applications can be challenging, as it requires modifying the Dockerfile to include them during the image build process. This can complicate management and debugging. See Challenge 4.1 .

  - **Complex testing:** Debugging in OpenFaaS cannot be performed in a local environment. Developers must build, push the image, and deploy it to OpenFaaS every time they want to test changes, which can be cumbersome. See Challenge 4.2 .

  - **Low stability:** OpenFaaS is not well-suited for stable, long-duration tasks. It struggles with applications that require persistent states or ongoing processes, such as model training, making it less ideal for certain use cases. See Challenge 4.4 .

6. Conclusion

Serverless architectures like OpenFaaS offer significant advantages in terms of scalability and developer velocity, making them well-suited for lightweight, short-duration tasks such as model inference. However, it is not suitable for long-duration tasks such as model training.

Additionally, although it provides a framework that can bypass the need to build traditional web servers, it does not include any support or templates for machine learning models. This makes the environment very difficult to set up and involves large and complex debugging steps.

To conclude, OpenFaaS does not seem to be the best choice in our case.

Appendix

- **GitHub Repository**:
  - Link: https://github.com/COMP4651-24fall/project-group-25

Reference:

[1] O. Ltd, “Home,” *OpenFaaS - Serverless Functions Made Simple*. https://www.openfaas.com/