Project 3 Containerized or Serverless Learning Report

Group 20: Hao Sun, Joy Xu, Yang Xu, Zhecan Yang, Yinuo Yao

Solution Design

In this project, we choose "**Containerized or Serverless Learning**", and we aim to deploy an image generation machine learning model on Google Cloud using both containerized and serverless architectures. By deploying on Google Cloud and employing these two distinct methodologies, we aim to evaluate and compare their performance, scalability, and cost-effectiveness, providing valuable insights into the most suitable deployment strategy which helps us determine the best environment for model deployment.

- Containerized Approach: we use a Docker container for the Flask backend and the machine learning model, deployed on Google Cloud Run. This setup uses Flask as the API server to handle image generation requests and interact with the model.
- Serverless Approach: we deploy using Google Cloud Functions, adapting the Flask application to a serverless context. This leverages serverless architecture's scalability and cost efficiency by dynamically allocating resources.
- Image Generation Model: our model, based on Progressive GANs (PGANs), is trained on the "celebA" dataset to generate high-quality images with resolutions up to 512 x 512 pixels. PGANs are chosen for their ability to enhance image quality and resolution progressively.

Our code and detailed evaluation results are available for review on our GitHub page(https://github.com/harry-sunhao/CMPT756-Project).

Implementation

We set up a local Docker environment, deployed a pre-trained image generation model, and measured evaluation metrics like response time and latency. Through code refactoring, we adapted to a serverless architecture using Google Cloud Functions. We also created a webpage to showcase the outcomes and processing times of both the Docker and serverless methods.

We successfully fine-tuned our Docker environment for optimized performance and completed deployment on Google Cloud Run, meeting our projected specifications of 4GB memory and 1 CPU. Additionally, we expanded our service capabilities by integrating and evaluating additional image generation models. The project's region is designated as us-central1.

Results

Here is the result of our implementation. On our web page, the user can specify the number of images they want to generate by inputting a number in the provided field and clicking the "Go!" button. The results are displayed under two categories: "Serverless Image" and "Docker Image", each with their corresponding generated faces (Figure. 1).

Performance metrics are also presented for both the serverless and Docker-based image generation methods. These metrics include “Round trip time,” “Process Time,” and “Latency Time,” measured in milliseconds, allowing users to compare the efficiency and response times of serverless versus Docker approaches in image generation tasks. The generated images are visually distinct and appear to be examples of high-quality, AI-generated human faces, showcasing the capabilities of modern GANs in creating lifelike images.

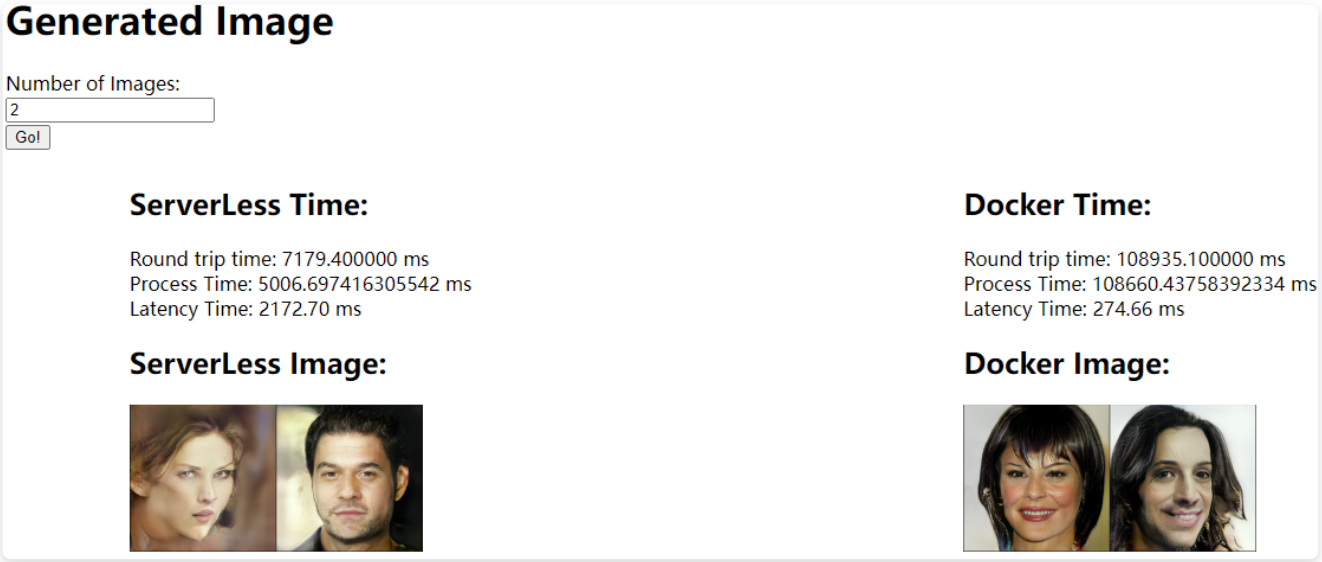

Figure 1. Results of Serverless vs. Docker

Measurements

In our evaluation, we employ Apache JMeter for load testing to assess the performance of container and serverless solutions. We simulate 100 users making image requests to our system and record the system’s response every minute for ten minutes. We focus on four metrics: response time, latency and resource usage (memory and CPU) (Figure 2) and connect time.

- Response time is how long the server takes to process a request from when it’s received to when it’s ready to reply, covering server activities like logic processing and database queries.
- Latency is the delay from sending a request to the beginning of receiving a response, mainly including network delays and the time before the server starts to send a response.
- Memory and CPU usage track the resources consumed during operation.

In our evaluation, low latency signals fast server response. High latency prompts us to check network conditions or improve server responses.

Figure 2. Evaluation Measurements

Analysis

The analysis of five images highlights the advantages of serverless architecture over Docker across multiple performance metrics:

- Latency and Response Time: Serverless demonstrates significantly better performance, with approximately a 20-fold reduction in latency and response time compared to Docker. This indicates faster image processing and better user experience.
- Memory Usage: Serverless may have a slightly larger memory footprint than Docker, but its efficient resource management ensures optimal performance without excessive overhead. This balance between resource utilization and performance is crucial for sustained operational efficiency.
- CPU Usage: Serverless architecture exhibits lower CPU usage compared to Docker, leading to more efficient resource utilization and reduced operational costs. Its ability to dynamically allocate resources based on workload demands contributes to this efficiency.
- Connect Time: Containers have a mean connection time of 58.94 ms with a standard deviation of 18.61 ms, while serverless exhibits a slightly higher mean connection time of 62.28 ms with a lower standard deviation of 14.52 ms. This suggests that serverless architecture, despite having a slightly longer average connection time, offers more consistent connectivity compared to containers.

In conclusion, when comparing serverless architecture and Docker for deploying machine learning models, serverless clearly excels in latency, response time, and resource efficiency. Additionally, a cost comparison reveals that Cloud Functions, with its significantly lower cost of $0.23 compared to $3.15 for Cloud Run, stands out as the more economical option. The scalability and ease of operation of Cloud Functions further solidify it as the optimal architecture for tasks like image generation.