Comparing Serverless Container as a Service Solutions in Public Cloud Service Providers

Vinícius Moura Barros
Department of Computing, TU Dublin, Tallaght, Ireland
X00159357@myTUDublin.ie

Introduction

Containers have been identified as the new application development paradigm by several sources and serverless architectures are the ones where the infrastructure is provisioned and managed by a third party. This research aimed to compare serverless Container as Service (CaaS) from major public Cloud Service Providers (CSPs): AWS ECS Fargate, Azure Container Instances and GCP Cloud Run. The study found answers such as whether these serverless CaaS solutions could be considered serverless, which solution has the best performance, which one is the cheapest, and defined recommendations of solutions based on the application utilisation. Alongside the study, a containerised application and tools were developed to provision, run tests, collect information and automatically plot charts on demand. This research collected data from 26,665 test cases executed between 02/10/2020 and 26/10/2020.

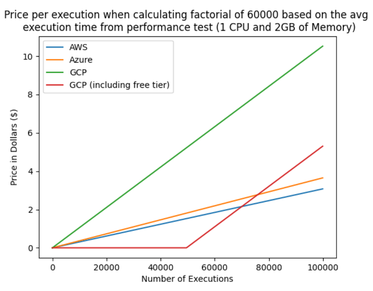
Are the solutions truly serverless?

3 pillars in the definition of serverless by Van Eyk et al. (2017): High Availability, Granular Billing and Event Driven Services.

GCP Cloud Run meets all criteria, but AWS ECS needs an extra service (Load Balancer) and Azure CI does not offer scaling.

Cheapest Solution

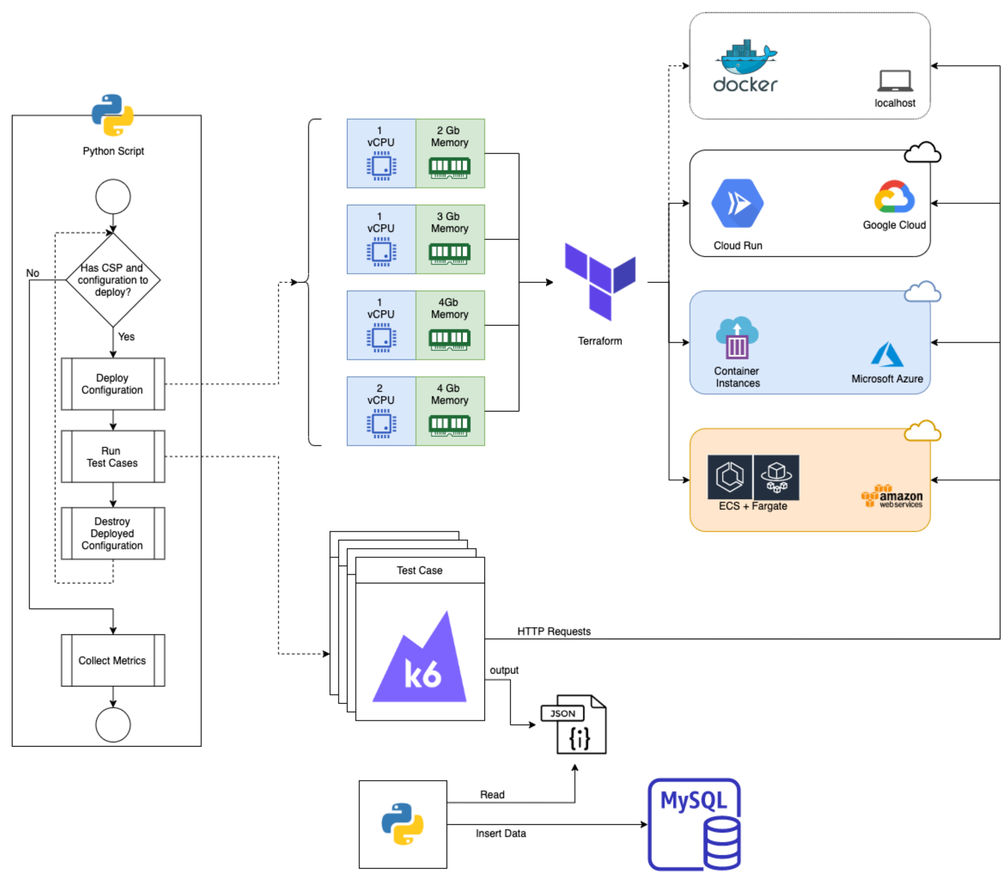
The main finding is that it depends on how many times and for how long an application needs to run. When only comparing the prices, AWS ECS Fargate is the cheapest, close followed by Azure CI. However, GCP Cloud Run offers a 50 hour free tier every month. When considering free tier and performance, in the long run, AWS ECS Fargate is the cheapest.



Methodology

At a high level, the main methodology consisted in developing a reusable and automated way of provision infrastructure, running tests, collecting data and display results in a friendly way.

The processes are mostly controlled by Python scripts. The main script is responsible for holding information about the configuration for different combinations of CPU (1 or 2 cores) and Memory (1, 2, 3 or 4GB) that are going to be tested. At every hour on the dot, the script provisions the serverless CaaS infrastructure in each CSP using Terraform, then runs a series of K6 test cases, which outputs results in a shared folder, and finally it destroys the infrastructure created, also using Terraform.



The second script runs every 30 minutes, collecting and normalising the data generated by the K6 tests before storing it in a MySQL database.

Once the data is in the MySQL database, another Python web application, using Flask and Matplotlib, is used to read and plot charts on demand. In total, there are 70 plots being generated by automated scripts, 60 related to performance, 8 related to pricing and 2 related to comparison between weekdays and weekends.

Performance Results

Hello World – Endpoint

- Similar and fast performance in all serverless CaaS (< 0.02ms)
- Best avg. application execution time: CGP Cloud Run

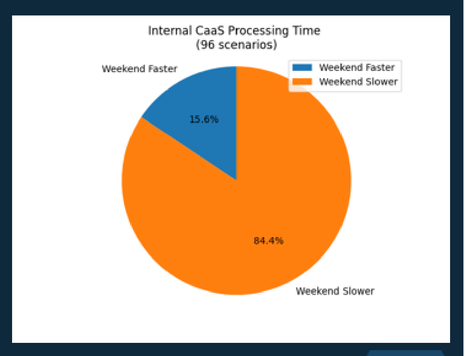
High CPU – Endpoint

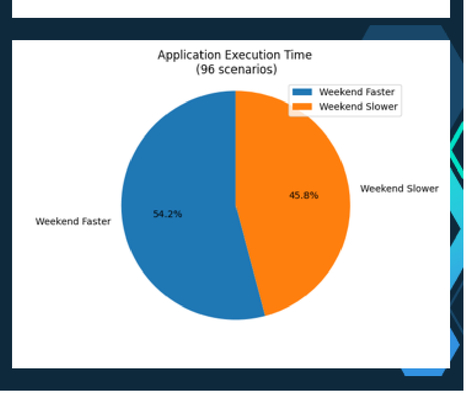
- Fargate avg. time always below baseline
- Factorial of 60,000 Fargate was 38% faster than GCP Cloud Run and 12% faster than Azure CI
- Fargate had the smallest variation between min, avg. and max processing times

Internal CaaS Processing Time

- Azure Container Instances had the best time in all studied scenarios

Weekend vs Weekday





GCP Cloud Run

AWS ECS Fargate

Azure CI

Conclusions and Future Work

The main conclusion is that no single solution is the best in every aspect studied by the research. Therefore, knowing well how an application works and its demands is crucial before choosing a serverless CaaS to host it. GCP Cloud run is recommended for an almost empty application. AWS ECS Fargate is recommended for a High CPU utilisation, and Azure CI is recommended for a faster internal processing time. In relation to pricing, GCP has a generous free tier that can benefit projects willing to try using a servereless CaaS, but for applications with larger and constant demands, AWS is the cheapest. Future work would include analysing auto-scaling, vendor lock-in implications, easy of use and cold starts. Future researchers could take advantage of the tools created in this project which are publicly available at `https://github.com/viniciusbarros/msc-caas-comparison`.