Serverless technologies comparison
Large Systems project

Sean Liao and Mar Badias

December 17, 2019

Abstract

Write abstrat here

1 Introduction

Public clouds are growing, and with it comes the latest push into serverless offerings. These come in many forms depending on the abstraction level, but they can largely be grouped into: long-lived containers, short-lived containers, functions as a service.

Long-lived containers or Platform as a Service (PaaS). They represent the first generation of serverless technologies. These can be full-fledged, stateful applications, packaged in containers. The clouds will take these and run them for you on VMs. Auto-scaling and load balancing is usually offered, but fast startup times are not guaranteed. These should be considered an alternative UI to the underlying VMs, which will be reflected in the pricing model (charge for underlying VMs). Examples: AWS Elastic Container service, GCP App Engine, Azure App Services, Alibaba Container Service.

Functions as a Service (FaaS). Currently the highest level of abstraction they are the second generation of serverless technologies, developers provide their application code for the clouds to compile, package, deploy, and run. These are short-lived and stateless, an instance may be started for every request and killed after it completes. Billing is only for the time it is running serving a request. Examples: AWS Lambda, GCP Cloud Functions, Azure Functions, Alibaba Function Compute, IBM Cloud Functions, Zeit Now.

Short-lived containers or Containers as a Service (CaaS). Is the newest technologies short-lived, stateless runtimes and a similar billing model. Where they differ from FaaS is that they introduce containers, giving developers control of the execution environment, allowing them to run languages or runtimes unsupported by FaaS. Examples: AWS Fargate, GCP Cloud Run, Azure Container Instance, Alibaba Elastic Container Instance.

2 Research question

Given the amount of services of this type in the current market we defined a buisness case based on the most popular features [1] of serverless computing: dynamically managed runtimes, autoscaling, globally reachable HTTP(S) endpoint and pay only for usage. Having defined our preferences, we want to look in their profiles and answer the following question:

- **When one should be chosen over the others when all are available?**

For deciding which is preferable we will consider the following subquestions:

- **Which has better raw computer performance?** With this question we will determine which solution provides the faster performance. While some platforms do provide numbers, we aim to check if these are comparable across services. Better here would be a faster completion of tasks, and/or a corresponding reduction in costs.
- **Which one has lower platform overhead?** We plan to measure the excess of computation time introduced by the platform as it supposes extra time in client request that we wish to minimize. Lower platform overhead will be considered more preferable.
- **Which one has lower cold start latencies?** When a new instance receives its first request, the response time increases because this instance must be created. As answering the client request needs to be done as fast as possibles, lower cold start will be considered preferable.

3 Methods

We want to test the products offered by the top 5 cloud providers as of 2019: Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure, Alibaba Cloud, and IBM Cloud. Additionally we will also test Zeit, a startup in the FaaS space popular for its streamlined experience. In the table 1 we can see the products we will work with and in wich category they fit.

As mentioned in section , the the pricing model for Pass, charge for underlying VMs, does not fit in the 'Pay only for usage' defined in our Reaserch question. However GCP App Engine offers the possiblitily to scale your applications to 0 when it is not been used so it fits into our buissnes case.
For answering the reaserch questions we will measure following variables:

- Overall performance. We want to obtaint how much does it take for each services to performe a request
- Platform overhead.

Type	Platform	Product
FaaS	AWS	Lambda
FaaS	GCP	Functions
FaaS	Azure	Functions
FaaS	IBM Cloud	Functions
FaaS	Alibaba Cloud	Function Compute
FaaS	Zeit	Now
CaaS	GCP	Cloud Run
PaaS	GCP	App Engine

Table 1: Serverless services that will be tested in this reaserch.

- Compute performance.

- we will get the ServerUID which is the instance UID - time of the day we execute it
-using images that their size are somehow ok. diferent formats but they follow a normal distribution, which makes their okok.
We want to use image processing as our test workload, a real world [**?**] use case. Specifically we will be testing image resizing (thumbnail creation). We aim to use the same code for all platforms (excluding API adapters) and our choice of language is Python because is one of the few languages that all platforms support.
Image processing was selected as it represents a common workload that, without hardware accelerators, relies heavily on CPU performance. It also does not require access to external resources, such as databases, which while also a common workload, introduce too much variability.
We will send images as HTTP requests to the various platforms to be resized. The code we deploy on the platforms will be responsible for both resizing and measuring the time it takes to do so. This will form the basis for our calculations of compute performance. We will additionally measure roundtrip time for requests from the client, this, minus the computation time will be used to calculate the platform overhead. The same experiment will be performed for obtaining the cold start latencies but with different timing to ensure that the instance is created when receiving the request.
For each service, which config did we used and why
Our plan is to spread out testing over a week, to even out variability from running at different times. Specifically we want to test hourly over a week for compute and overhead at single and 50 concurrent requests and repeat it 10 times, to cover the different concurrency guarantees of different products, and every 3 hours for cold start times (to allow time for the function to be evicted from local caches, additional testing required).
At the end of the experiments, the billing of each service will be compared too.

4 Related work

Cloud FaaS performance has been subject of previous research. An excellent example is [2] where multiple serverless providers are continuously been benchmarked. Their points

of comparison will be used in this project for measuring and comparing them short-lived containers. Another example of an excellent comparison of Faas providers is [**?**]. Cloud short lived containers have also been a topic of study, comparing them to long live containers or performance test among of different provides. We can find examples of this in [**?**] and [**?**]. Comparison between Faas and short-lived containers has also been studied but from a functionality point of view, oversimplificating it and not taking into account the performance [**?**][**?**][**?**].
Up to our knowledge, the performance differences between Faas and short live containers have not been comprehensively analysed yet, which motivates our research. READ THIS AND SEE WHAT CAN U INCLUDE HERE `https://sci-hub.se/10.1002/cpe.4792`

5 Results

After performing the mentioned experiments we obtained the following results:

5.1 Overall performance

The table shows. Parallel instances say they are parallel duirng a single test cycle.
Overall performence does not change during time (time of the day does not matter)

5.2 Overhead

In figure XX we can find one plot for each service we have analized. They show the overhead of single vs muntiple and the other cold vs warm.

5.3 Compute performance

In figure XX we can see strictly the time of resizing the image in each test(multi vs single).

5.4 Price to performance

6 Discussion

Say: alibaba has a low succes rate so its results bla bla Overall performence does not change during time (time of the day does not matter)

6.1 Overhead

- cold is higher (as expected) because it has to iniciate the instance.
AWS landa load balancer is better because no differenec un single vs munlti Multi is slower because load balancer can be a bottleneck. if the graf increases at the end is bad because it means that the difference in overhead between request is high: the overherad is not stable. overhead is not stable in gpc-fun. What might affect this overhead? -¿ alibab at the end warm jumps

6.2 Computer performance

Something about CPU freq and how green and purple line very togethrt-¿ provide a good isolation of resources gpc-run results expected because it can handle 5 req at the same time. IMB and alibaba -bad isolation. if the lines grows, the platform gives(flat) or not consitency to the request (not very important here.)

6.3 Scaling

-alibaba succes rate is the lowest because it has to many request, do not scales well. Does it invalidate results? gpc-fun there are outlayers.
- azure just creates 4 instances. -aws is very consistent. for both overhead and cpu time/servertime2 - (of the table) -¿ say we are sending 50 paralel req. Some of them gives us 50 instances but gcp-run gives us many instances but the standard desviation is really high which it means its very insconsitency giving instances??? - if its lower then 50 means they are not doing it one to one. If there is more than 50, they are not good at reusing instances. gcp-run -gcp-run may have been affected by us, when telling it that just 5 req per instance. -GCP-APP-enginee it scales depending on cpu utilitzation and it looks like it consistencly handles 2 req per instance.

6.4 Price to performance

6.5 Technologies

how does this explain about performance.

6.6 Research Questions

(answer research questions)

References

[1] CloudFlare. What Is Serverless Computing? — Serverless Definition. `https://www.cloudflare.com/learning/serverless/what-is-serverless/`.

[2] Bernd Strehl. Serverless Benchmark. `https://serverless-benchmark.com/`.