# Serverless technologies comparison
## Large Systems project

Sean Liao and Mar Badias

December 18, 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) 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 and 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.

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, globally reachable HTTP(S) endpoint, pay only for usage and autoscaling. Having defined these characteristics, we selected the products that provide them 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 also included Zeit, a startup in the FaaS space popular for its streamlined experience. The products selected are:

- AWS Lambda (FaaS)
- GCP Functions (FaaS)
- GCP Cloud Run (CaaS)
- GCP App Engine (PaaS)
- Azure Functions (FaaS)
- IBM Cloud Functions (FaaS)
- Alibaba Cloud Function Compute (FaaS)
- Zeit Now (FaaS)

As mentioned in section before, the the pricing model for PaaS (charge for underlying

VMs) does not fit in the 'Pay only for usage' defined in our buisness case. 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.
Our reaserch will measure and analiyse its CPU performance, overhead and specially their capacity to scale when the workload increases. Scaling entails creating more instances, known as cold starts, and assigning new resources between them without influencing others' instances performance.

## 2 Related work

Serverless technologies have been subject of previous research. We can find good examples of PaaS analysis in [2] and [**?**], where the reaserchers analize Google App Enginee performance with CPU-intensive applications. CaaS 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 analized but from a functionality point of view but oversimplificating it and not taking into account the performance [**?**][**?**][**?**].

FaaS on its own also has been subject of previous research. An excellent example is [3] where multiple serverless providers are continuously been benchmarked. Another example of a comparison of Faas providers is [**?**]. In [**?**] the researches dicuss the advantages of using cloud services and AWS Lambada for systems that require higher resilance. Finally, is necessary to mention [**?**], which fouses in its performance evaluation and also benchmarking the data transfer to storage and its lifetime of the major cloud functions providers.

Aldought serverless performance has been heavily analized, to the best of our knowledge their scaling and its consequences to performance and overhead have not been comprehensively analysed yet, which motivates our research.

## 3 Research question

Having defined our buisness case, we needed to 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.

- **Does scaling out affect the performance of the services?** When the workload increases, serverless platforms should create more instances and assign more resources to them to provide an stable performance to the client. The stablest performance and overhead when incresing the workoad would be considered better.

# 4 Methods

As mentioned in section 4, we want to test the products offered by the top cloud providers as of 2019.
In the table 1 we can see the products we will work with and in wich category they fit.// A detailed description

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.

- 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

| Type | Platform | Product | Web Framework | Instance Size | Pyhon runtime | Location of the DC |
|---|---|---|---|---|---|---|
| FaaS | AWS | Lambda | Custom | 128 MB | 3.8 | London, UK |
| FaaS | GCP | Functions | Flask | 128 MB | 3.7 | St. Ghislain, BE |
| FaaS | Azure | Functions | Custom | 128 MB | 3.7 | NL |
| FaaS | IBM Cloud | Functions | OpenWhisk | 128 MB | 3.7 | London, UK |
| FaaS | Alibaba Cloud | Function Compute | Custom | 128 MB | 3.6 | Frankfurt, DE |
| FaaS | Zeit | Now | Python http.server | 128 MB | 3.6 | Brussels, BE |
| CaaS | GCP | Cloud Run | Python http.server | 128 MB | 3.8 | St. Ghislain, BE |
| PaaS | GCP | App Engine | Flask | 128 MB | 3.8 | St. Ghislain, BE |

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

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.

# 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).

## 6 Discussion

### 6.1 Scaling

From Table 2 we observed elevated failure rates for Alibaba Function Compute. Their documentation states they have a limit of 50 instances per service, and even though our client places a strict limit of 50 inflight requests at any point in time, their scheduler might not be keeping up.
Also from Table 2 we can see the average number of unique instances per test cycle. Ideally, this number would be 50, to match the number of parallel requests we are sending.
On the low end, Azure Functions is notable for only starting 4 instances. This may be from a documented limitation of only starting at most 1 instance per second with HTTP triggers. Their load balancers still hold the requests in queue, but they appear to be reluctant to start new instances even as sufficient time has passed.
At the high end, we configured GCP Cloud Run to accept up to 5 concurrent requests per instance, expecting it to utilise the available memory and CPU more efficiently. Unfortunately we are unsure if this negatively impacted their scheduler as we observed a less efficient reuse of instances and greater variability in the number of instances started.

### 6.2 Warm vs Cold Starts

An often cited concern of using truly on-demand compute services is the cold start, when the platforms have to start up a new instance to handle increases in load. This is such a concern that some some platforms have specific features built to keep your instances warm, such as AWS Provisioned Concurrency and Azure Functions Premium Plan.
Our results in Figure 2 show that, as expected, in most cases, single concurrency cold starts are slower than warm starts and the start times are highly consistent. Surprisingly, cold start times are much more variable at higher concurrency levels, with the exception of AWS Lambda which appears unaffected.
What is unexpected is in Figure 3, that that cold starts affect CPU performance. These instances consistently perform worse on a cold start, even as they are reused for future requests.

### 6.3 Technologies

how does this explain about performance.

### 6.4 Research Questions

(answer research questions)

## References

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

[2] Prodan R. Sperk M. Ostermann S. Evaluating High-Performance Computing on Google App Engine. IEEE Software (Volume: 29, Issue: 2, March-April 2012).

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