Exploiting the Challenges of the Adoption of Serverless Computing

Author: Leandro Rodrigues da Silva
Advisor: Alfredo Goldman
Co-advisors: João Francisco Lino Daniel, Anderson Andrei da Silva

1. Introduction

Serverless is a paradigm that is becoming more popular in the software engineering community in the last years, as an alternative to the conventional server-based cloud solutions. In this paradigm, the developer no longer needs to worry about scaling or managing resources and can focus on the business domain problems of the application, while the cloud platform keeps responsible for dealing with these challenges.

Also, serverless computing offers a low cost host service, since it has a cost-by-execution policy. That means the cost of maintaining an application is no longer tied to the time it’s kept up, but to the amount of executions used. This allows companies that are living an early stage of development to deliver their systems with low cost.

Despite the benefits, Serverless still has some challenges to overcome, such as testing, benchmarking, documenting patterns, task scheduling, etc. In this work we aim to explore these challenges and run experiments and surveys with the goal of deeply understanding these gaps and, finally, deliver a practical solution.

2. Background

Over the years, the way we architect software has been changing. Since the problems to be solved become more and more complex, different approaches were adopted. Some well known models of building complex systems are:

2.1 Server Architecture

The server architecture is one of the most used architectures through the web together with the “client-server model”.

In this architectural style, the developers are responsible for both producing code related to the domain of the application and to the infrastructure behind it. When the application is done and deployed, it will run nonstop even when it's not being used.

This architectural style has the benefit of usually delivering responses with low latency, since the server it's always up to respond. This approach brings some

disadvantages, such as: high cost, need of maintaining code unrelated to the core business of the application and also a limited scalability [1].

2.2 Cloud computing

Cloud computing is the delivery of computing services and resources - such as storage, networking, analytics, etc – over a network,being a private one or over the internet. Currently, the following models are the main in the classification of cloud computing [2]:

On-premises - in this model, the company maintains and manages the resources that build the physical infrastructure below the Operating System. Developers are responsible for managing hardware, allocating and scaling resources, configuring Operating Systems, etc. This kind of model brings some advantages, such as having full control over the infrastructure, which can lead to an environment that makes it easier to conduct compliance policies. On the other hand, this kind of model incurs some drawbacks, such as high costs, since it will need a fully prepared space to allocate this infrastructure, and also specialized employees that will be responsible for maintaining it.

Infrastructure as a Service (IaaS) - In this kind of model, a service provider manages the infrastructure and offers access to resources on top of it over the Internet. The developers can access it through an Application Programming Interface (API) . All the complexity of managing hardware and space is abstracted by the providers, such as AWS, Azure and Google Cloud. They are responsible for dealing with these challenges and the developers can keep focused on working in the core of the application.

The usage of cloud computing has advantage sas the following:

- **Reducing costs**: cloud platforms reduce some of the efforts of providing resources, such as installing, configuring and transporting hardware. This leads to lower costs to the final consumer in comparison with the traditional way of building infrastructure.
- **Improving productivity of developers:** after eliminating these challenges, the developers are free to work in solving the problems of the application, instead of dealing with problems related to managing the infrastructure.
- **Providing high scalability:** cloud platforms provide high scalability through its services. The services are built-in with the possibility of horizontal and vertical scalability in just a few clicks [3].

Platform as a Service (PaaS) - In this model, the third-party provides both the framework on which engineers can build applications. It differs from Infrastructure as Code in the way that it abstracts some components (like storage and network services) to deliver an easier to use environment.

2.2 Serverless Architecture

Serverless is a cloud-native execution model for building and running applications without server management. In this model, the cloud platform is responsible for dealing with provisioning, managing and maintaining the infrastructure on behalf of the server, so that the developers are responsible for plugging the business code into this structure.

In this execution model, a concept commonly used is **Function as a Service (FaaS).** FaaS is a type of cloud service that allows developers to deploy small applications (or parts of a bigger one). The cloud platform receives a package containing the code and wraps it into a stateless container, abstracting the logic of allocating resources, scheduling tasks, setting up protocols and mainly, setting up a server. In services like this - such as done by AWS Lambda - the cost of maintaining an application is tied to how many executions a function receives [3].

As mentioned above, there are lot of open challenges to be tackled in the area of developing complex systems for the cloud, specifically using serverless architecture. So, we plan to exploit this area and propose some goals in this work.

3. Proposal

The goal of this work is to explore the current challenges of building systems using serverless architectures. As of the time being, we do not have a well-defined problem to attack. Nonetheless, we're set to explore the aforesaid context in order to better understand the relevance of its contribution opportunities. Yet, under this context we have the following points of interest:

- **Documentation of design patterns** - Currently, there is no centralized source, not even deep study of documentation about design patterns related to serverless architectures, so there is a huge gap when finding literature that can guide the process of building this kind of system.

- **Tools for developing in open source serverless platforms** - There are lots of tools to help developing in the biggest serverless services (AWS Lambda, Azure Functions and Google Cloud Functions), but there is a huge gap when we talk about tools for open source platforms (e.g OpenWhisk).

- **Scheduling** - Finding the best way of scheduling tasks inside the container of a serverless platform is still a problem. There are various possible ways of scheduling jobs to run inside these containers and finding ways of optimizing the time of execution/cost relation is still a challenge for engineers that build these platforms.

3.1 Methodology

The universe of serverless computing still has lots of challenges in different scopes. As a relatively new technology, there are open gaps in development, patterns documentation, optimization, benchmarking, etc [5]. Our immediate goal is to explore the context under the perspective of a practitioner. To do so, we plan to follow some experiments to find it and discover the contributions we will have made by the end of the project. The main ideas to conduct these experiments are:

1. **Dive deep into the community challenges** - we also plan to take a closer look at the problems that the open source community is facing nowadays. This will be done by looking for possible gaps documented inside these open source repositories.
2. **Conduct surveys** - we expect to conduct a survey with practitioners that have daily contact with these technologies and find possible gaps that can be solved in this work.
3. **Running actual tests on platforms** - we plan to explore open source serverless platforms in order to understand some of the problems they present. The following platforms may be investigated:
    - AWS Lambda
    - OpenFaaS
    - OpenWhisk

4. **Simulations** - another approach will be to simulate real behaviors found during the exploitation. With this in hand, we can study specific scenarios in isolated environments. To do so, tools like Batsim [6] and SimGrid [7] will be considered.

And after theses studies, we plan to:

5. **Work on a solution to solve the problems we found in the previous steps.**
6. **Work on monograph and presentation**

3.2 Expected Results Resultados esperados

For this immediate goal, we expect to have a deep understanding of the problems the serverles community is facing. With this in hand, we can use it as a base to pursue a relevant contribution as the final goal of this work.

4. Schedule

We divided the work in the following tasks:

1. Dive deep into the community challenge
2. Conduct surveys
3. Running actual tests on platforms
4. Simulations

5. Decide and work on a solution.
6. Work on monograph and presentation

Task	Apr	May	Jun	Jul	Aug	Sept	Oct	Nov	Dec
1	x	x	x						
2		x	x	x					
3			x	x	x	x	x	x	
4				x	x	x	x	x	
5					x	x	x	x	
6						x	x	x	x

5. Bibliography

[1] Justin Garrison and Kris Nova. Cloud Native Infrastructure, 2018 - Patterns for Scalable Infrastructure and Applications in a Dynamic Environment
[2] Redhat, 2018 -Types of Cloud computing. https://www.redhat.com/en/topics/cloud-computing/public-cloud-vs-private-cloud-and-hybrid-cloud#cloud-services
[3] Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems, 2017 - Martin Kleppmann
[4] Hossein Shafiei, Ahmad Khonsari, Payam Mousavi, 2021 - Serverless Computing: A Survey of Opportunities, Challenges, and Applications
[5] Batsim. https://batsim.readthedocs.io/en/latest/
[6] SimGrid. https://simgrid.org/