POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Master Degree Thesis

Development, Test and Application of a framework for cloud serverless services

Thesis Supervisor

Dr. Ing. Boyang Du

Candidate

Andrea Santu

matricola: 251579

Intership Tutor

Dott. Magistrale Antonio Giordano

Academic year 2020-2021

Abstract

In the scene of services for the creation of web applications is focusing more and more towards a micro services oriented approach, moving away from structures called monolithic. The maximum representation of this is with the serverless paradigm, which since 2014 has seen an ever greater increase in its use and in its investments by the major cloud providers. Such a paradigm has found an implementation in the cloud model Functions as a Service, which uses as its main resources, plain simple functions. Serverless Framework has emerged as one the major framework that allow the usage of the homonym paradigm in a simple way, and introducing a level of abstraction regarding the underlying structure of the chosen cloud provider. Despite the functionalities introduced by Serverless, the developer has to take charge of various operations concerning indirectly the business logic of the application, with the main one being: to structure the code base, to define the various resources through the compilation of a configuration file, to define a unit testing structure, fundamental once the application complexity increases. Furthermore, based on the chosen cloud provider, the developer must find solutions to problems such as Cold start, and limitations in the creation of resources.

The Restlessness framework was born with the goal of improving the user experience of Serverless, providing a standard project and testing structure, a Command Line Interface and a local Web Interface through which is possible to completely manage the project, and with the further goal of minimizing all operations that do not concern directly the application's business logic. The framework is provided as an Open Source package, and with the possibility of extending its functionalities, through the use of addons, some of which are already present, to address common patterns, such as database access or authentication. During the framework development has been possible to test it on real applications, thus allowing to find and

correct critical issues, whose main ones were: Cold start handling, use of the non relational database mongodb, and limitations on the applications structure proposed at the beginning.

Contents

Chapter 1

Cloud services

In the early days of the web, anyone who wanted to build a web application had to buy and maintain the physical hardware required to run a server, which was a cumbersome process to undertake, especially for small businesses [1]. Then came a new paradigm for the provisioning of computing infrastructure, named Cloud Computing, and defined as:

"Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs." [2]

Cloud Computing is possible because of a technology called virtualization, which allows the creation of a simulated computer, named virtual machine, that behaves as if it were a physical computer with its own hardware. When properly implemented, this approach allows having a more efficient use of the physical hardware, as each computer is able to run many virtual machines at once. Despites the many benefits, using virtual machines still requires manual server administration, as each one simulate a full system, including the operating system and the underlying kernel.

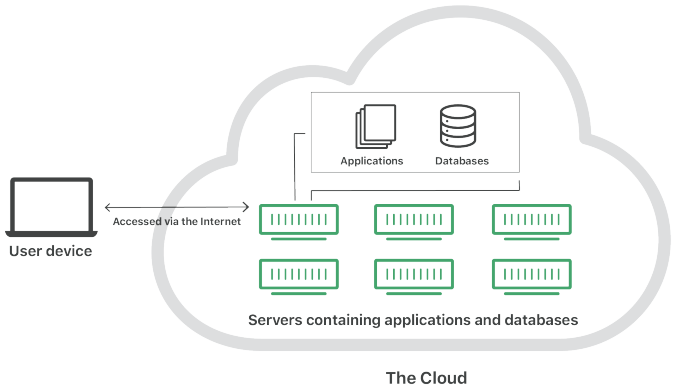


Figure 1.1. Representation of the cloud

The next technological step has been containerization, which gave the possibility of packing an application and all its dependencies, such as system libraries and system settings into a single entity called Container. With this approach a single physical machine, including the kernel, is shared by a multitude of containers. The main advantages that containerization offers, with respect to virtual machines are [3]:

- Portability: once the application is packed into a container it can be run on any host supporting that technology.
- Control and flexibility.
- Faster deploy.
- Less server administration.

With this premises about the cloud and its infrastructure is possible to outline the main models that have emerged in the context of cloud computing.

1.1 Cloud computing models

Between the various types of cloud computing architectures have emerged three main models, which are: Infrastructure as a Service, Platform as a Service and Software as a Service. Each model is characterized by an increasing level of abstraction regarding the underlying infrastructure.

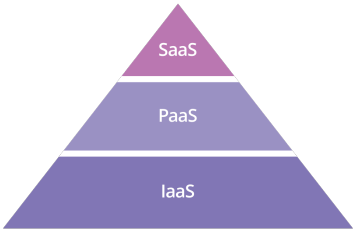


Figure 1.2. IaaS, PaaS, SaaS Pyramid

1.1.1 Infrastructure as a Service (IaaS)

Infrastructure refers to the computers and servers than run code and store data. A vendor hosts the infrastructure in data centers, referred to as the cloud, while customers access it over the Internet. This eliminates the need for customers to own and manage the physical infrastructure, so they can build and host web applications, store data or perform any kind of computing with a lot more flexibility. An advantage of this approach is scalability, as customers can add new servers on demand, every time the business needs to scale up, and the same apply also if the resources are not needed anymore. Essentially physical servers purchasing, installing, maintenance and updating operations are outsourced to the cloud provider, so customers can spend fewer resources on that and focus more on business operations, thus leading to a faster time to market. The main drawback of this approach is the cost effectiveness, as businesses needs to over-purchase resources to handle usage spikes, this leads to wasted resources [4].

1.1.2 Platform as a Service (PaaS)

This model simplify web development, from a developer perspective, as they can rely on the cloud provider for a series of services, which are vendor dependent. However some of them can be defined as core PaaS services, and those are: development tools, middleware, operating systems, database management, and infrastructure. PaaS can be accessed over any internet connection, so developers can work on the application from anywhere in the world and build it completely on the browser. This kind of simplification comes at the cost of less control over the development environment [5]. An example of this kind of services is Google with its product App Engine.

Another model has recently been added to the three main cloud computing models, named Backend as a Service (Baas). This model stands, with some differences, at the same level of PaaS, and it's suited especially for web and mobile backend development. As with PaaS, BaaS also makes the underlying server infrastructure transparent from the developer point of view, and also provides the latter with api and sdk that allow the integration of the required backend functionalities. The main functionalities already implemented by BaaS are: database management, cloud storage, user authentication, push notifications, remote updating and hosting. Thanks to these functionalities there may be a greater focus on frontend or mobile development. In conclusion BaaS provides more functionalities with respect to the PaaS model, while the latter provides more flexibility.

1.1.3 Software as a Service (SaaS)

In this model the abstraction from the underlying infrastructure is maximized. The vendor makes available a fully built cloud application to customers, through a subscription contract, so rather than purchasing the resource once there is a periodic fee. The main advantages of this model are: access from anywhere, no

need for updates or installations, scalability, as it's managed by the SaaS provider, cost savings. However there are also main disadvantages, that makes this solution not suitable in some cases: developers have no control over the vendor software, the business may become dependent on the SaaS provider (vendor lock-in), no direct control over security, this may be an issue especially for large companies [6].

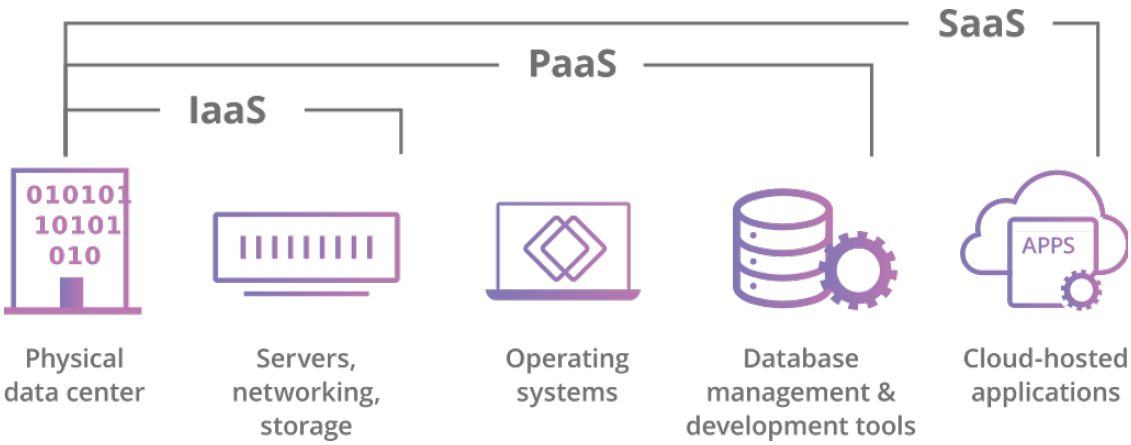


Figure 1.3. IaaS, PaaS, SaaS diagram

1.2 Serverless paradigm

The downsides of the previously described approaches varies from the control on the infrastructure and on the software, to scalability problems, to end with cost and resources utilization effectiveness. With the aim of solving these problems, the major providers started investing on a new cloud computing model, named Function as a Service (FaaS) and based on the serverless paradigm. Such a paradigm is based on providing backend services on an as-used basis, with the cloud provider allowing to develop and deploy small piece of code without the developer having to deal with the underlying infrastructure. So despite the terminology, serverless does not means without servers, as they are of course still required, but they are transparent to developers, which can focus on smaller pieces of code. With this model, rather

than over purchase the resources, to ensure correct functionality in all workload situations, as happens in the IaaS model, the vendor charges for the actual usage, as the service is auto-scaling. Thanks to this approach consumer costs will be fine grained as shown in 1.4.

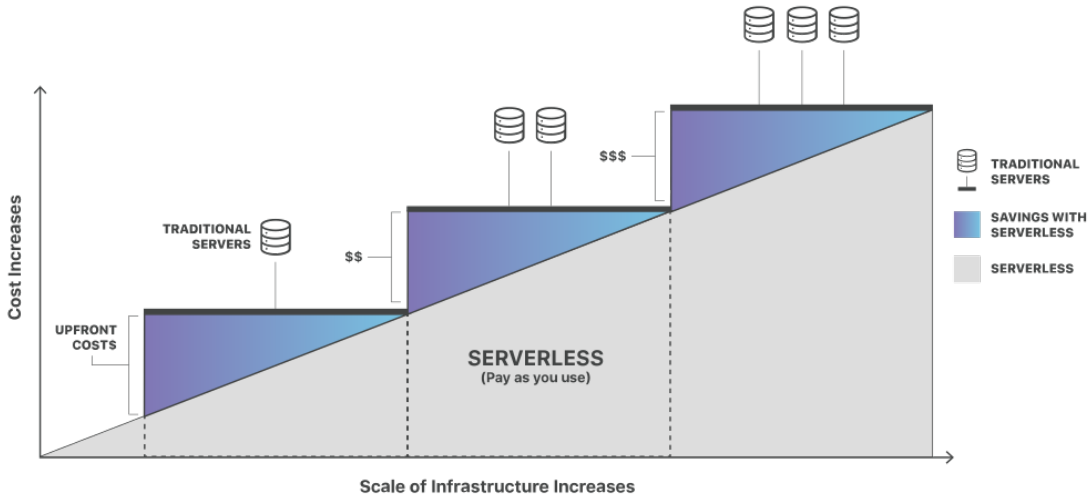


Figure 1.4. Cost Benefits of Serverless

Being the underlying infrastructure transparent for the developer, you get the advantage of a simpler software development process, and this advantage characterize also the PaaS model. Furthermore being the service auto-scaling, is possible to obtain a virtually unlimited scaling capacity, as it happens in the IaaS model, where the limit is the cloud provider availability.

An implementation of the serverless paradigm is the cloud model named Function as a Service (FaaS), which allows developers to write and update pieces of code on the fly, typically a single function. Such code is then executed in response to an event, usually an api call, but other options are possible, so it executed regardless of the events, and this lead to the previously described benefit regarding scalability and cost effectiveness. Furthermore, through this model turns out to be more efficient to implement web applications using the modular approach of the micro services architecture (1.5), since the code is organized as a set of independent

functions from the beginning.

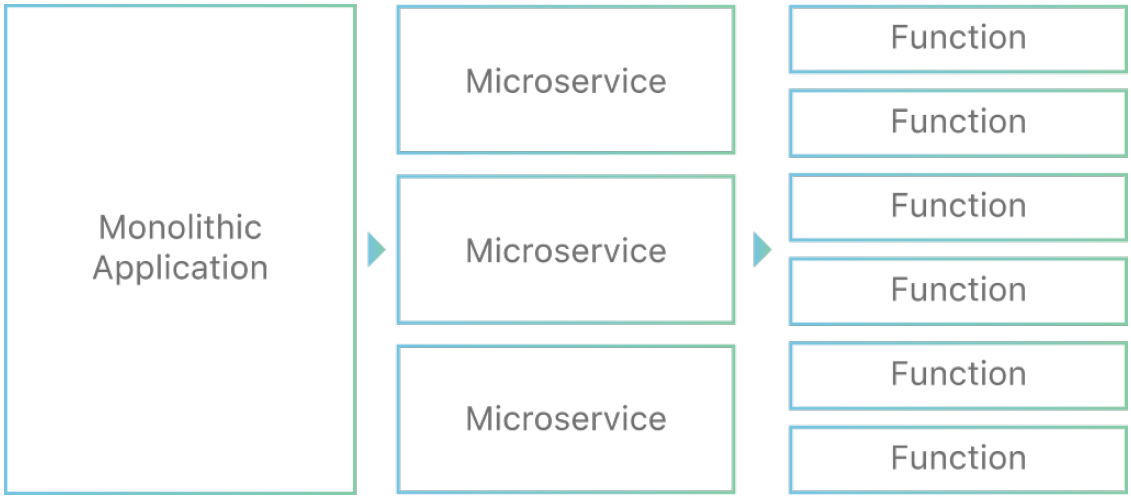


Figure 1.5. Monolithic to Micro services application

So the main advantages of the FaaS model are: improved developer velocity, built-in scalability and cost efficiency. As each approach, there are also drawbacks, in this case developers have less control on the system, and an increased complexity when it comes to test the application in a local environment.

The first cloud provider to move into the FaaS director has been Amazon, with the introduction of aws lambda in 2014, followed by microsoft and google, with azure function and cloud function respectively in 2016.

1.3 Serverless Framework

Shortly after the release of the service Aws lambda functions, has been introduced, in 2015, the Serverless framework, with the main objective of making development, deploy and troubleshoot serverless applications with the least possible overhead. The framework consists of an open source Command Line Interface and a hosted dashboard, that combined provide developers with serverless application lifecycle management. Serverless supports all runtime provided by Aws, corresponding to

the most popular programming languages such as: Node.js, Python, Ruby, Java, Go, .Net, and others are on development.

Although the serverless framework, given the number of cloud providers supported, aim to be platform agnostic, the following examples will be based on the Aws provider and on the Node.js programming language.

The main work units of the framework, according to the FaaS model, are the functions. Each function is responsible for a single job, and although is possible to perform multiple tasks using a single function, it's not recommended as stated by the design principle Separation of concerns [7]. Each function is executed only when triggered by and Event, which can be of different type, such as: http api request, scheduled execution and image or file upload. Once the developer has defined the function and the events associated to it, the framework take care of creating the necessary resources on the provider platform.

The framework introduces the concept of Services as unit of organization. Each service has one or more functions associated to it and an application can then be composed by multiple services. This structure reflects the modular approach of the micro services architecture described previously. Finally various applications are grouped under an organization (1.6)

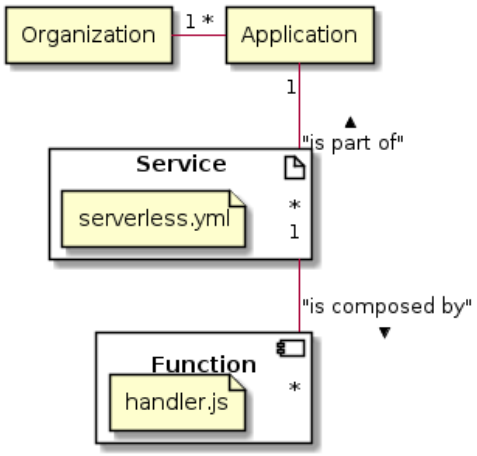


Figure 1.6. Serverless framework resources scheme

A service is described by a file, located at the root directory of the project, and composed in the format Yaml or Json. Below is a simple serverless.yml file (listing 1.1), it defines the service users, which contains just a function, responsible of creating a user. The handler field specify the path to the function code, in this case the framework will search for a handler.js file, exporting a usersCreate function, as show on listing 1.2.

```
org: my-company-org
app: chat-app
service: users
provider:
    name: aws
    runtime: nodejs12.x
functions:
    usersCreate:
        handler: handler.usersCreate
        events:
            - http: post users/create
```

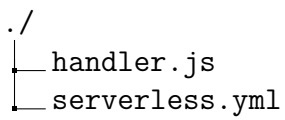
Listing 1.1. Simple serverless.yml file

```
async function usersCreate(event, context) {
    const user = {
        name: 'sample_name',
        surname: 'sample_surname'
    }
    await mockDb.createUser(user)
    return {
        statusCode: 200,
        body: JSON.stringify({user})
```

```
    }
}
```

Listing 1.2. Simple handler function

Figure 1.7. Simple Serverless project structure



Serverless is flexible and does not force a fixed structure of the project, that task is up to the developer. Defined that structure, the service can be deployed using the Serverless CLI, on the chosen provider, as shown on listing 1.3.

```
$ serverless deploy
Serverless: Stack update finished...
Service Information
service: users
stage: dev
region: us-east-1
stack: users-dev
resources: 12
api keys:
  None
endpoints:
  POST - https://.../dev/users/create
functions:
  usersCreate: users-dev-usersCreate
layers:
  None
```

Listing 1.3. Deploy command

The deploy command creates the necessary aws resources, in this case they are: a lambda function corresponding to the usersCreate function and an api gateway to handle http requests. It is then possible to test the newly created resource by making requests to the url returned by the CLI, specifying the resource path /users/create. It is possible to invoke online functions also directly from the CLI, specifying the identifier of the function used in the serverless.yml file, as shown on listing 1.4

```
$ serverless invoke -f usersCreate
{
    "statusCode": 200,
    "body": "{\"user\":{\"name\":\"sample_name\", ...}}"
}
```

Listing 1.4. Invoke command

The development and deploy process shown for a service with a single function remains the same as the service complexity grows, in particular it is possible to modify and deploy a single function at a time, since each function has its own resource associated. This process gets along with the previously described micro services architecture.

1.3.1 Advantages

The main advantages of using the Serverless framework are:

- Provider agnostic: the framework aims to be independent from the chosen cloud provider, thus avoiding vendor lock-in. In practice this feature is not achieved completely, as the configuration file serverless.yml may be different

across providers. However the main structure remains the same, and that simplify providers migration.

- Simplified development: the CLI commands simplify the development process, from the deploy from the testing of the deployed functions.
- Extensible: is possible to develop plugins that integrate with the CLI commands lifecycle, increasing their functionalities.
- Dashboard: the hosted dashboard allow monitoring and tracing of the deployed functions and services.

1.3.2 Disadvantages

The main advantages of using the Serverless framework and the Serverless paradigm are:

- Compilation of the configuration file may become tedious as the project grows.
- The framework is extremely flexible regarding the project structure and that is an advantage, however this can also be a drawback as it's up to the developer to find a suitable structure, and this means less time spent on business related tasks.
- Unit testing: it is possible to test a deployed function easily, however for big projects, where it's necessary to test a lot of functions, this may become cumbersome.
- Resource threshold: for projects created with Aws, a single serverless.yml file may create up to 200 resources, and if exceeded the deploy operation fails. Since each function is responsible for the creation of about 10 resources, is very easy to exceed this limit. The only solution so solve this problem is to split the functions across multiple services, hence different serverless.yml configuration files.

- Cold start: inherent overhead of the current implementation of the serverless paradigm. Since each function is executed only in response to an event, a certain amount of time is required for resources initialization.

1.4 Conclusions

Each cloud model presented has its own strength and drawbacks, depending on the needs of the wanted goal. Favouring as selection criteria, solutions that present major advantages in terms of scalability, cost efficiency and speed of development, has been decided to favour the Serverless option. The main cloud providers offering this kind of service, as previously stated, are: Aws, with its Lambda service, Microsoft, with Azure Functions, and Google, with Cloud Functions. Each provider offer different configurations, with different pricing, based on memory, CPU, and execution time as parameters, as shown on 1.1. In the literature there are several documents comparing the various services side by side exhaustively [8]. For the project subject of this document has been chosen Aws as the main provider, as the most mature platform meeting the project's needs. In particular it providers the following advantages with respect to the competitors [8]:

- Cold start (1.2)
- Overall maturity
- Performance consistency
- Scalability

	AWS	**Azure**	**Google**
Memory (MB)	64 * k (k = 2, 3, ..., 24)	1536	128 * k (k = 1, 2, 4, 8, 16)
CPU	Proportional to Memory	Unknown	Proportional to Memory
Language	Python Nodejs Java, and others	Nodejs Python, and others	Nodejs
Runtime OS	Amazon Linux	Windows 10	Debian 8
Local disk (MB)	512	500	> 512
Run native code	Yes	Yes	Yes
Timeout (second)	300	600	540
Billing factor	Execution time, Allocated memory	Execution time, Consumed memory	Execution time, Allocated memory, Allocated CPU

Table 1.1. Cloud providers configuration [8]

Provider-Memory	Median	Min	Max	STD
AWS-128	265.21	189.87	7048.42	354.43
AWS-1536	250.07	187.97	5368.31	273.63
Google-128	493.04	268.5	2803.8	345.8
Google-2048	110.77	52.66	1407.76	124.3
Azure	3640.02	431.58	45772.06	5110.12

Table 1.2. Cloud providers Cold start (in ms) [8]

Bibliography

[1] What is serverless https://www.cloudflare.com/learning/serverless/what-is-serverless

[2] A Break in the Clouds: Towards a Cloud Definition ....

[3] What is the cloud https://www.cloudflare.com/learning/cloud/what-is-the-cloud/

[4] What is IaaS https://www.cloudflare.com/learning/cloud/what-is-iaas

[5] What is PaaS https://www.cloudflare.com/learning/serverless/glossary/platform-as-a-service-paas

[6] What is SaaS https://www.cloudflare.com/learning/cloud/what-is-saas

[7] Dijkstra, Edsger W (1982). "On the role of scientific thought". Selected writings on Computing: A Personal Perspective. New York, NY, USA: Springer-Verlag. pp. 60–66. ISBN 0-387-90652-5. https://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD447.html

[8] Liang Wang, UW-Madison; Mengyuan Li and Yinqian Zhang, The Ohio State University, Thomas Ristenpart, Cornell Tech; Michael Swift, UW-Madison "Peeking Behind the Curtains of Serverless Platforms" ISBN 978-1-939133-02-1 https://www.usenix.org/conference/atc18/presentation/wang-liang

[9] What is JavaScript https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

[10] Node.js https://nodejs.org/en

[11] Node.js Event Loop https://nodejs.dev/learn/the-nodejs-event-loop

[12] Node.js on Backend https://www.netguru.com/blog/node-js-backend

[13] Node.js Event Loop https://www.geeksforgeeks.org/node-js-event-loop/

[14] TypeScript https://www.typescriptlang.org/

[15] TypeScript: What is it and when is it useful? https://medium.com/front-end-weekly/typescript-what-is-it-when-is-it-useful-c4c41b5c4ae7

[16] Eric Elliott (26 June 2014). Programming JavaScript Applications: Robust Web Architecture with Node, HTML5, and Modern JS Libraries. "O'Reilly Media, Inc.". pp. 87–. ISBN 978-1-4919-5027-2. https://books.google.com/books?id=jUfnAwAAQBAJ&pg=PA87

[17] JSON data interchange syntax ISO https://www.iso.org/standard/71616.html

[18] Npm Scoped packages https://docs.npmjs.com/creating-and-publishing-an-organization-scoped-package

[19] Mastering Issues https://guides.github.com/features/issues

[20] From Monolith to Monorepo https://medium.com/@brockreece/from-monolith-to-monorepo-19d78ffe9175

[21] Agile Manifesto https://agilemanifesto.org/iso/it/principles.html

[22] L. Chen, "Continuous Delivery: Huge Benefits, but Challenges Too," in IEEE Software, vol. 32, no. 2, pp. 50-54, Mar.-Apr. 2015, doi: 10.1109/MS.2015.27. https://ieeexplore.ieee.org/document/7006384

[23] CircleCi Documentation https://circleci.com/docs/2.0/about-circleci

[24] API Gateway Lambda Authorizers https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-use-lambda-authorizer.html

[25] Serverless pros and cons https://hackernoon.com/what-is-serverless-architecture-what-are-its-pros-and-cons-cc4b804022e9

[26] Serverless Framework Aws Guide https://www.serverless.com/framework/docs/providers/aws/guide/intro

[27] Data Access Object Pattern https://www.oracle.com/java/technologies/dataaccessobject.html

[28] Spazio alla scuola https://www.fondazioneagnelli.it/2020/07/17/spazio-alla-scuola

[29] Aws Lambda Environment https://docs.aws.amazon.com/lambda/latest/dg/runtimes-context.html