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. Antonio Giordano

Academic year 2020-2021

Abstract

todo

brief description of the thesis

Acknowledgements

todo

Contents

Chapter 1

Introduction

1.1 Cloud computing models

Between the various types of cloud computing architectures, in the last few years have emerged three main models, through which to develop web applications. These are IaaS, PaaS e SaaS. Each of the models is characterized by an increasing level of abstraction regarding the underlying infrastructure.

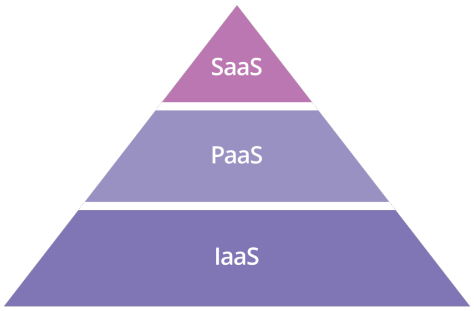

Figure 1.1. IaaS, PaaS, Saas pyramid

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 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.

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.

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.

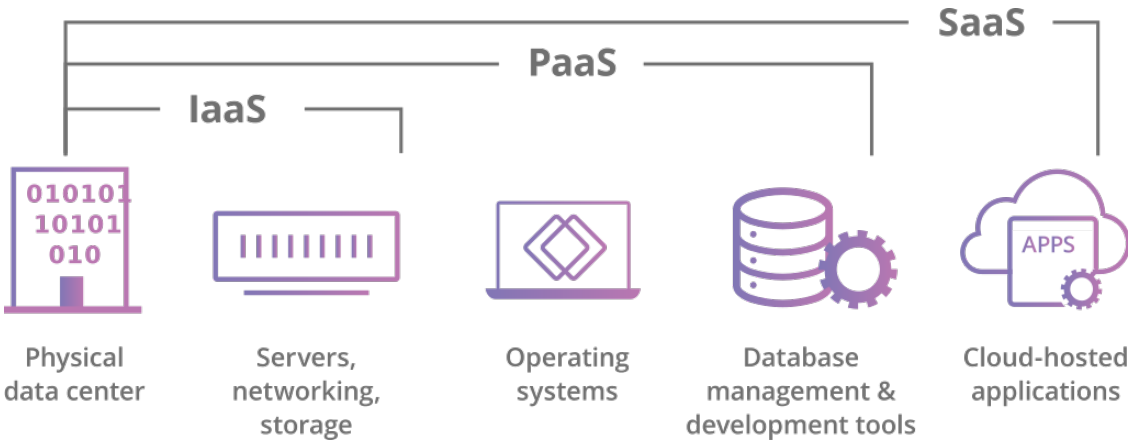

Figure 1.2. IaaS, PaaS, Saas diagram

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.

1.2 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 1.3.

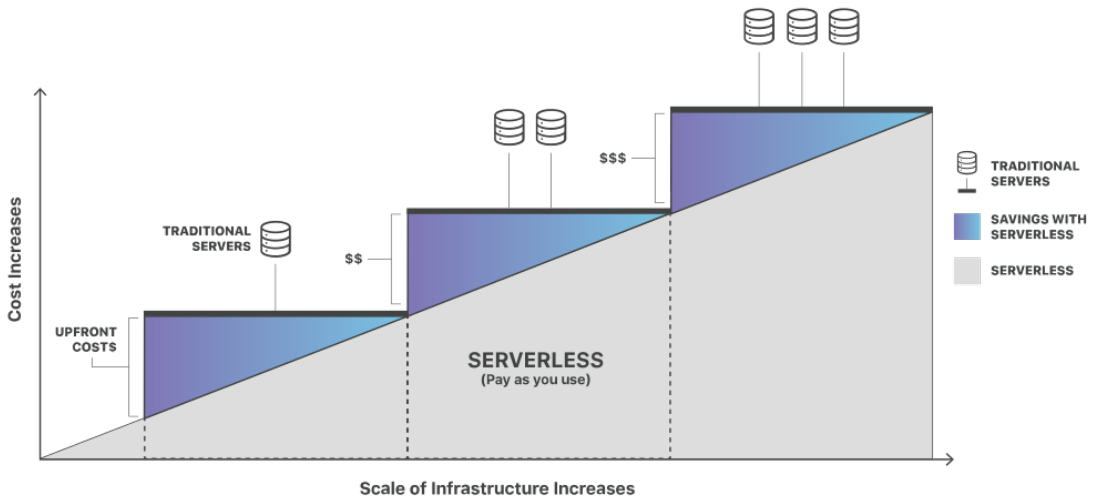

Figure 1.3. Cost Benefits of Serverless

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 (1.4), since the code is organized as a set of independent functions from the beginning.

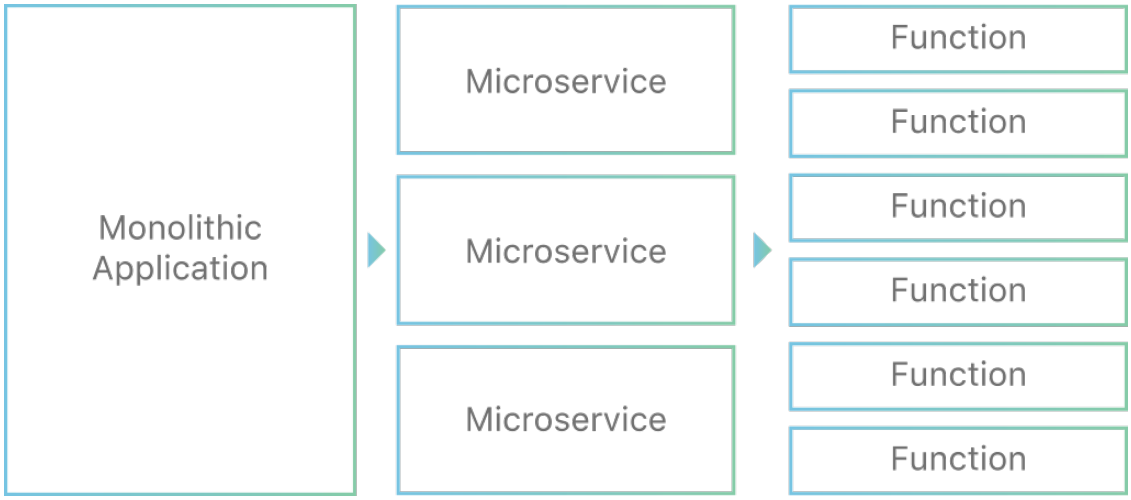

Figure 1.4. Monolithic to Micro services application

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.

Although the serverless framework, given the number of cloud providers supported, aim to be platform agnostic, this document will consider primarily the Aws provider, for both the usage of the serverless framework and the subsequent development of the Restlessness framework. This choice is due to the maturity of the platform with respect to what the competitors. 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. This document will focus on the Node.js runtime along with the typescript 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. Each function is executed only when triggered by and Event, there are a lot of events, 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 a web application can then be composed by multiple services. This structure reflects the modular approach of the micro services architecture described previously.

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 , 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

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, as show in 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, as shown in 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 in 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 The idea behind Restlessness

The open source framework named Restlessness was born with the goal of improving the developer experience of the Serverless framework, by addressing its encountered problems. The framework is composed by a Command Line Interface and a frontend application with an associated web server running locally. In particular the main functionalities that the framework aims to provide are:

- Creation of a new project, through the CLI, based on the typescript language and with a standard structure.
- A local Web Interface that allow creating and managing project resources, functions, with their associated events, and models.
- The creation of a standard unit testing structure for each function, and based on the jest library.
- A standard validation structure for function's input, based on the yup library.
- Deploy of multiple services with a single CLI command, to deal with the resource threshold limitation of Aws.

By addressing those points the framework aims to give developers the tools to focus on writing business code rather than spend time on boundary problems, that are important, but there may be the risk of solving the same problems multiple times (reinventing the wheel), which may be avoided.

1.5 Related Works

@TODO

Are there other similar framework? What are the differences? Why use restlessness instead?

1.6 Tools

Several tools were used during the development of the project, varying from the ones supporting code development, to the organizational ones. Below is a list of them:

GitHub Version control platform based on the Git system. It has been used for the development, organization and management of the main project *Restlessness*, as well as the drafting of this document. Both are available for consultation:

- Restlessness: `https://www.github.com/getapper/restlessness`
- Thesis: `https://www.github.com/androsanta/Thesis`

Slack Business communication platform. It has been used for team communication to achieve a more direct and private interaction with respect to what *GitHub* offers.

CircleCi Continuous integration platform. It has been used for testing and deploying operations for all restlessness packages.

Serverless Open source framework that simplify the development and deployment of applications based on the serverless architecture, on the best known compute services.

Aws Amazon Web Services. Platform for cloud computing services. It has been used for deployment and testing of application created with the *Restlessness* framework.

Node.js Open source Javascript runtime environment that allow Javascript code execution outside a web browser.

Npm The *Node.js* package manager. It has been used for project's dependencies and for publishing of all *Restlessness* packages.

Typescript It extends the Javascript language by adding type definitions, and a transpiler that generates code that runs anywhere Javascript runs, from the browser to *Node.js*. It has been the main programming language used.

WebStorm Javascript IDE from JetBrains. It has been used for all code development.

Chapter 2

Restlessness

Detailed description of the project's structure with description of main components/packages (maybe a class diagram could show easily the structure)

micro-services structure of the project created by the framework handling of environment files

2.1 Project created by the framework

todo
description of a project created using the framework

2.2 Core

todo

2.3 Cli

todo

2.4 Backend

todo

2.5 Frontend

todo

2.6 Testing

todo

handling of tests for the framework itself and for the project created by the framework

magari partire dalle funzionalità più che fare una documentazione sterile quindi creazione del progetto, sviluppo, testing, deploying....

Chapter 3

Restlessness Extensions

the framework has been designed with extensibility in mind

some auth and dao extensions are already provided

3.1 Authentication

auth-jwt and auth-cognito packages

3.2 Database Access Object

dao-mongo package (plus plugin to create a database proxy)

3.2.1 Database Proxy

todo

show data about number of connections until crash in normal situation and when using the proxy

Chapter 4

Development

4.1 Github

Useful tools provided by GitHub (projects to handle tasks) Roadmap, development process/flow (issues, pull requests...)

4.2 Continuous Integration

circle-ci

parlare anche di npm (insieme a ci)

Chapter 5

Application

Application to real projects with emerged problems

5.1 FGA covid school api

brief description of the project
Problems arised by using restlessness (mainly the need for a database proxy and a solution for cold start, i.e. warmup plugin)

5.2 Gbsweb Claranominis api

brief description of the project
Problems arised by using restlessness (mainly necessary migration to micro-services structure)

rimuovere claranominis e parlare delle 2 problematiche solo per Spazio alla scuola parlare anche dell'attenzione mediatica e del numero di utenti (per mostrare le capacità di scalabilità di sls)

Chapter 6

Deployment

6.1 Aws

todo

detailed description of what resources are created when deploying

6.2 Why Aws

Chapter 7

Conclusions

todo?? cosa si potrebbe fare per migliorare il framework -> integrazioni con altri db cross platform in futuro? includere gli altri cloud providers per renderlo agnostico testing -> non c'è ancora nulla per gli integration test ricavare info dalla dashboard,