Distributed Systems and Cloud Computing: FaaS Management

Fardella Roberto - 0334186

October 16, 2023

TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Fig. 1. university logo

1 INTRODUCTION

Function-as-a-Service (FaaS) is a type of cloud computing service that allows developers to create, run and manage application packages as functions, without the need to maintain a dedicated infrastructure. FaaS is an event-driven execution model that operates within stateless containers. Functions are designed to manage server-side logic and state, leveraging the services offered by FaaS providers. The main reasons for developing FaaS management include dynamic scalability, workload balancing on local node resources, and cloud compute service. In this report, we will create a per- sonalized FaaS management system to balance the load between local resources and AWS Lambda,a serverless event-driven compute service that allows running codes for any type of back-end application or service without provisioning or managing servers. In addition, we will examine all crucial aspects of the project, from design to implementation.

2 DESCRIPTION OF THE ARCHITECTURE

The system consists of three macro components (Figure 2):

- **A client** in which users can interface to be able to invoke 3 possible functions of their choice ;
- A lightweight virtualization platform, called **Docker**, that enables the creation, dis- tribution and management of functions within containers;
- **AWS Lambda**, an event-driven serverless computing platform provided by Amazon as part of Amazon Web Services.

In the system, in addition to these three components, we have a **container and meta-data manager of the application, which is responsible for** managing the execution of functions, in support in the implementation of the offioading policy and container management, taking into consideration:

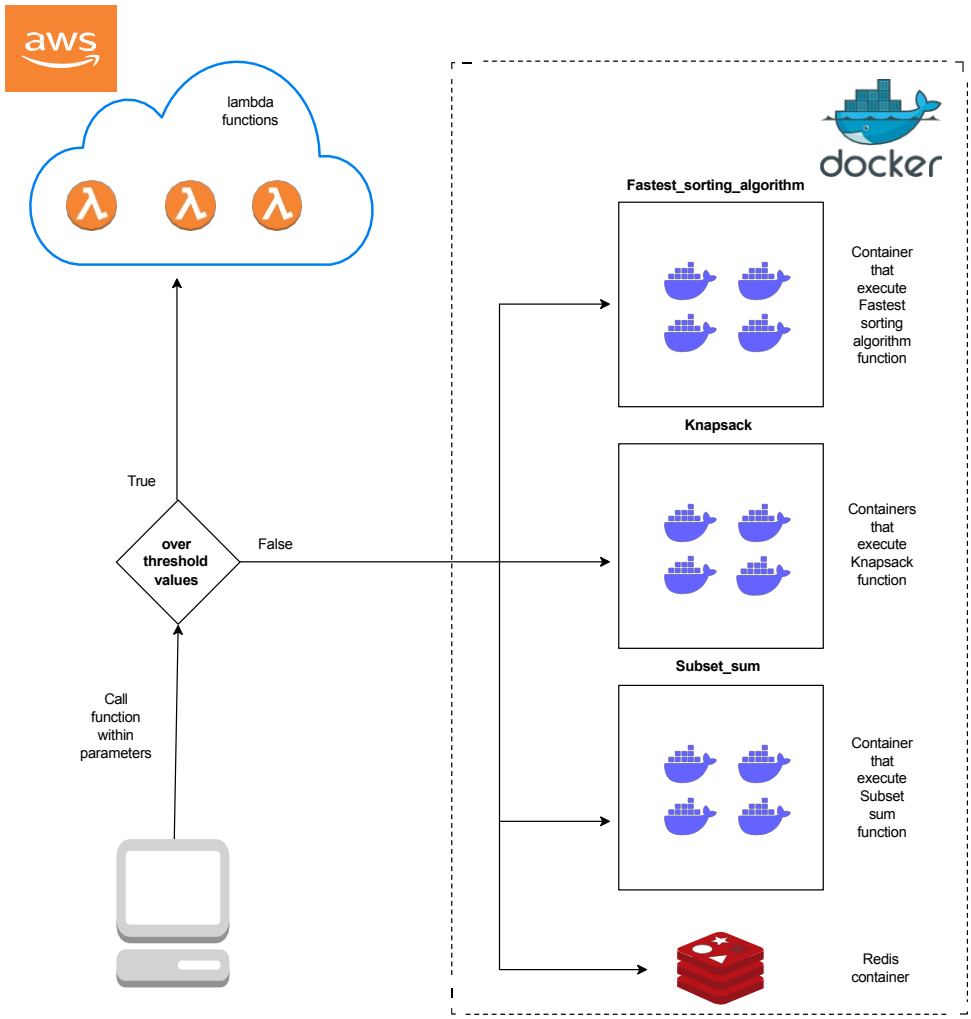

Fig. 2. ArchiteNura of FaaS management.

- The cold start problem;
- Simultaneous calls of the same function.

2.1 Functions

2.1.1 ***Fastest sorting algorithm.*** is proposed to compare three well-known sorting algorithms: "Bubble Sort," "Selection Sort," and "Merge Sort." The goal is to determine which of these algorithms is the fastest for sorting an array of integers. To do this, we implemented a function called findFastestSortingAlgorithm. The goal of the Backpack Problem is to determine which objects to select so as to maximize the total value of the objects within the maximum capacity of the backpack.

2.1.2 ***Knapsack.*** The "Knapsack Problem" is a classic combined optimization problem. In this problem, a selector must decide which items to choose from a set of items, each with a specific value and a specific weight, in order to maximize the total value of the selected items without exceeding a predetermined weight limit.

2.1.3 ***Subset sum.*** The "Subset Sum Problem" is another combinatorial optimization problem in which the objective is to determine whether there is a subset of elements from a given set such that the sum of the values of these elements is equal to a specific target value. In other words, the problem is to find a subset of the elements such that their sum is equal to a desired target value.

2.2 Offloading policy

The threshold-based offioading policy is based on a set of threshold values, which are taken from a configuration file called "config.json." The "config.json" file is accessed periodically by a thread at Runtime, which is then possible to change its values making FaaS Management more flexible and trying to reduce the degree of application-dependency, depending on the necessary needs arising from the application usage context.

Threshold values can be used to determine when a serverless function should be offioading on the AWS Lambda cloud service. For example, aas set of criteria that have been used to do offioading are based on system resources within the Docker environment . A thread at Runtime uses threshold values to monitor running serverless functions, comparing them with corresponding metrics that estre periodically from the Redis cache. When a serverless function exceeds a threshold value, the thread will set a flag to the value True, redirecting the next requests from the client to the AWS Lambda service.

2.3 Cold start

Cold start is a problem that occurs when a container is started for the first time and takes some time to load all the necessary resources. This can cause a delay in the application's response time. To address this problem, we have imple- mented a solution that keeps the container idle for a specified period of time before removing it. The time period is defined in the config.json configuration file. A thread periodically monitors the config.json file. If the timeout has been reached, the thread compares the values of CPU and memory used and the number of active containers against a set of threshold values. If one or more values exceed the threshold, the thread removes the inactive container. This solution reduces the amount of system resources used by idle containers, improving application performance.

3 IMPLEMENTATION & TECHNOLOGIES USED

3.1 Docker

Deployment of the functions, written in the Go language, was done through docker containers.

3.1.1 ***Dockerfile.*** the image that will later be used to create a container, is built using a multistage Dockerfile. A multistage Dockerfile is a Dockerfile that uses at least one base image to build the final image. This can be useful to reduce the size of the final image and to improve the security of the Dockerfile. The first image is an official Go image, which is then used as the base image to build the final one called base-debian11.

3.1.2 ***Use of Redis.*** In the context of our project, we identified several crucial needs to ensure the performance, scalability, and reliability of our system. A key compo- nent in realizing these goals was the implementation of Redis, a high-performance in-memory data storage system. Below, we will discuss in detail how we used Redis in two important aspects of our project.

- **Pub/Sub Messaging Broker:** Redis is known for its pub/sub messaging capabilities. It is useful for propagating parameters to containers in a serverless or FaaS architecture. You can publish messages containing parameters and subscribe containers or functions interested in those messages. This allows asynchronous communication between different parts of your system.

- **Cache System:** Redis is ideal for in-memory data storage. It allows fast access and reduces the load on permanent storage systems. This is especially useful for performance monitoring and container optimization. In particular, we have used Redis as a cache system for the following purposes:

  – **Cold Start support:** reduces container startup time. A container remains idle for a predefined period of time, which is read from a JSON configuration file. This time period is then compared to the timestamp of the container where it switches to the "exited" state, i.e., when the function execution ends;

  – **storage of system metrics:** it has been impo- pleted as a caching system to store system metrics in real time, avoiding the use of a traditional database and allowing rapid access to data during the execution of our serverless appli- cations, while also reducing the load on the system;

  – **Support for the adopted offloading policy**: threshold values are placed within this cache. These threshold values are periodically read from a JSON file. If they undergo changes, these values are stored again in the Redis cache, ensuring an efficient mechanism for dynamic threshold management within its serverless environment.

3.1.3 ***Libraries.***

- The boto3 library, Amazon Web Services (AWS) software development kit (SDK) for Python. Boto3 enables Python developers to write software that uses AWS services;
- The client was developed in Python with the **TKintker** graphical user interface library

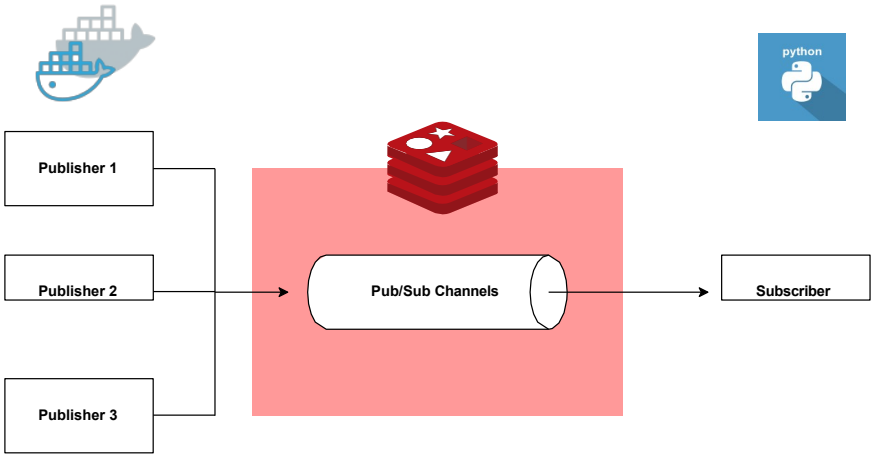

Fig. 3. return values returned by paNern pub/sub

- Docker-py, Python library for interaction with the Docker Engine API. Docker Engine is software for creating, managing and deploying containers
- The Redis library for Python, redis-py. While the Redis library for Go is go-redis.

3.1.4 ***AWS Lambda.*** AWS Lambda is a serverless service that allows you to run code without having to manage servers or infrastructure. Lambda functions are designed to run quickly and efficiently, and expire automatically after a specified timeout.

Using AWS Lambda for load balancing has the following advantages:

- It reduces the load on the server. Lambda can be used to perform functions that require a lot of time or resources in order to free up server resources and improve application performance;
- Improves application performance. Lambda can be used to execute functions quickly and efficiently, improving application performance.
- It is scalable. Lambda can scale automatically to meet demand, ensuring that functions are always available.

4 CONFIGURATION

4.1 json configuration file

Separating data from code using a JSON file is an effective way to improve the maintainability, flexibility, portability and scalability of the system. a **JSON configuration file** divided into four key categories was used:

- **Dockerfile configuration:** dockerfile paths for each component in the system;

- **Redis Server Configuration:** contains the address and port of the Redis server used in the project;
- **Threshold values;**
- **Channels for subscription and publication (pub/sub) used for communication between system components.**

4.2 AWS Lambda Credentials

In order to properly offioading on aws Lambda, the .aws/credentials file must be configured initially.
AWS credentials are valid for 4 hours, after which the AWS labora- tory must be restarted again and the new keys and token must be taken: it goes without saying that you cannot use the application consecutively for more than 4 hours.