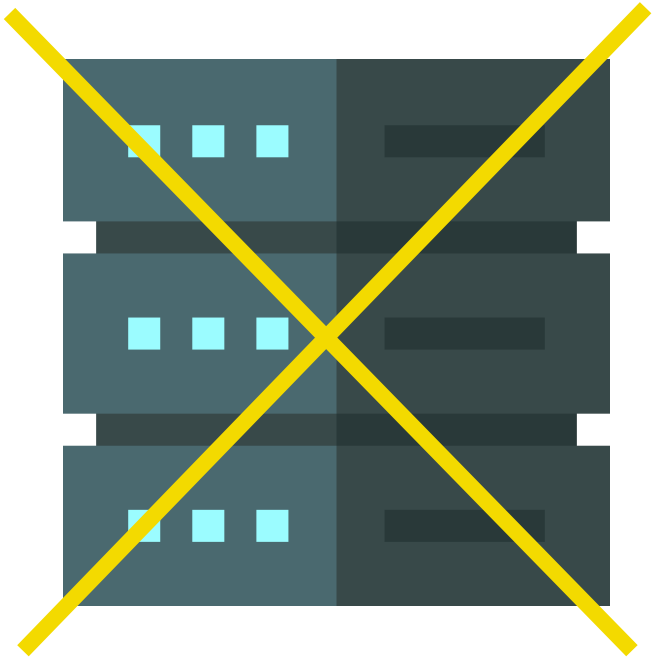


HOTEL SERVICE MANAGEMENT

Serverless Computing for IoT-Project

PROBLEM DESCRIPTION

The basic problem that this project aims to solve is the **management of all kind of accesses in a building**.

- It can be initially applied to a hotel building scenario, but it can be potentially extended to all buildings that need access management.

«SERVERLESS»

Serverless Computing is a cloud computing model in which code is run as a service without the need for the user to maintain or create the underlying infrastructure.

This doesn’t mean that serverless architecture doesn’t require servers, but instead that a third party is managing these servers instead of the developers.

01

PROJECT REQUISITES

Details and requisites of the project

PROJECT REQUISITES

In a hotel scenario, in most cases, there are many services available to guests such as spa, games room, gym, and so on. However each guest cannot access each service for free, but each service has its cost.

Games Room

Room with all kind of games: poker, pool...

Gym

Gym room and specialized training rooms

Spa

Mineral bath with health-giving properties

Restaurant

Sea and land food restaurant

Laundry

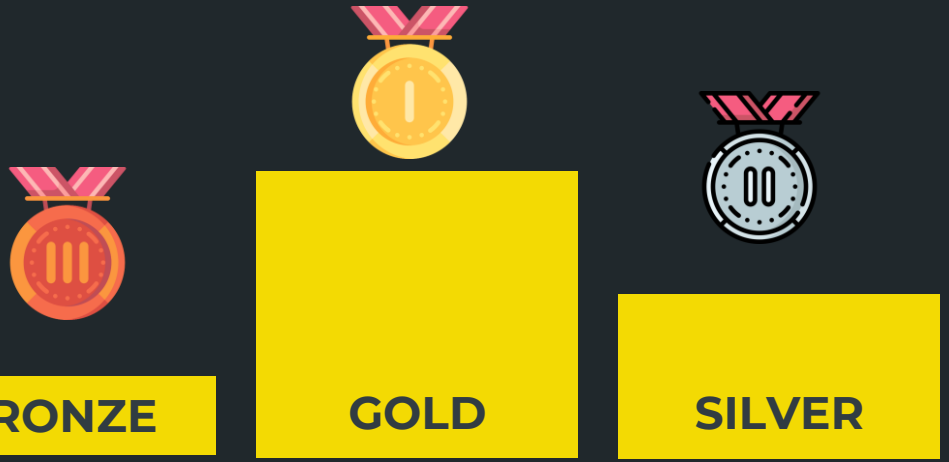
Clothes can be washed and ironed.

Massages

Experienced people with years of experience

ROLES

To manage the access to these services some roles are assigned to the guests. When a guest pays for the services he wants, a bracelet is issued to the guest. The bracelet will represent the access pass to the services purchased by the guest.



BRONZE

The access is allowed only in the games room.

GOLD

The access is allowed for every service unlimited.

SILVER

The access is allowed for every service, but for a limited number of times.

02 USE CASE

Tipical use case

USE CASE (ACCESS SERVICE REQUEST)

01

A guest wants to access the games room, so he puts his bracelet on the reader sensor near the games room door.

02

A function checks the role of the guest who made the access request and store the request on database.

03

If that role is allowed to access that particular service, then a message is showed on the screen near the service door:
“**access allowed**" or "**access not allowed**".

03 SOLUTION DESCRIPTION

General description of the solution, its architecture and its components

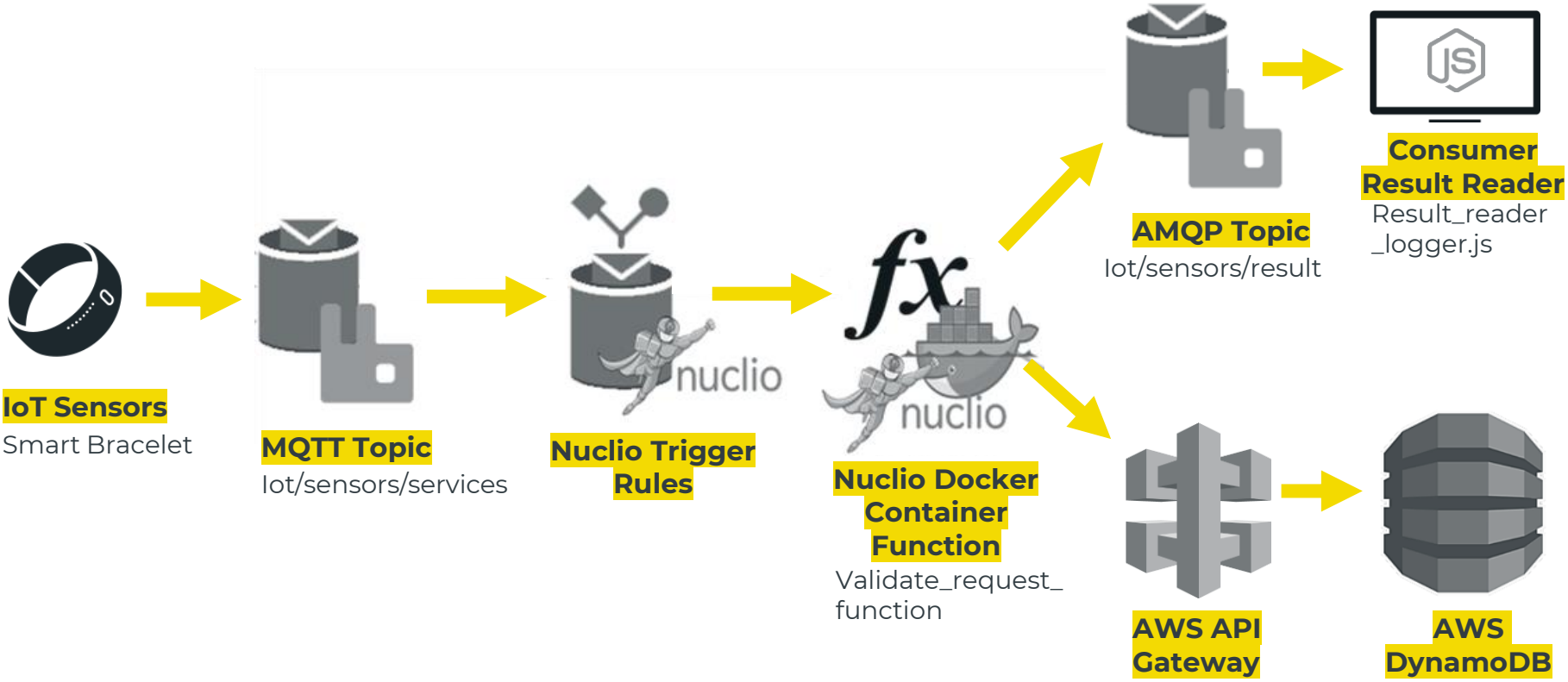
IOT SENSORS EMULATION

- The serverless functions are trigged by events generated from small devices such as sensors and mobile (IoT devices), commonly these devices communicates using message-passing, in particular on dedicated protocols as **AMQP** or **MQTT**.
- In this project to emulate the IoT sensors the events could be generated either by another function or from a MQTT client(mobile app).
- Anyway it would be easy to test using a real device that allows to send messages on MQTT or AMQP topic).

Nuclio Function that generate the event

MQTT Client

ARCHITECTURE



RABBIT MQ: OPEN SOURCE MESSAGE-BROKER

RabbitMQ is an open-source message-broker software that supports various communication protocols, AMQP and MQTT.

In our architecture we use both mentioned protocols:

MQTT

MQTT to publish a message on a topic in order to trigger a Nuclio Function

AMQP

AMQP to send the result of the access request on a topic in order to be consumed and showed

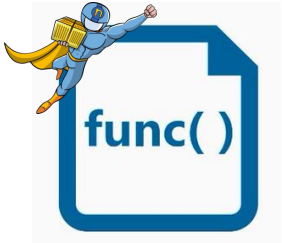
NUCLIO

Nuclio is an open source and managed serverless platform used to minimize development and maintenance overhead and automate the deployment of data-science based applications

In our architecture we use Nuclio for:

Nuclio Trigger Rules

A Nuclio trigger rule ignite the execution of a Nuclio serverless function



Nuclio Docker Container Function

DYNAMO DB

It is a fast and flexible **NoSQL** database service for any scale.

This DB choice is made because:

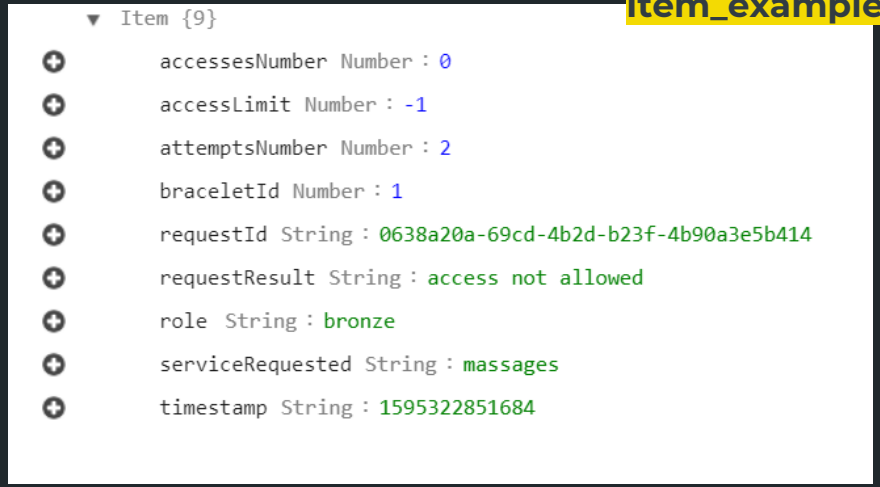
1. No particular structure or relations are needed for our application
2. No server to manage: DynamoDB is serverless with no servers to provision, patch, or manage and no software to install, maintain, or operate.

DYNAMO DB TABLE

hotel_service_management

requestId	timestamp	braceletId	role	Access Limit	Attempts Number	Accesses Number	Service Requested	Request Result

Item_example



AWS API GATEWAY

Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale.

We use it in our architecture as “front door” for serverless functions to access data in DynamoDB.

API KEY

To secure the APIs, in order to make them accessible only to the applications allowed and not publicly we use **API KEYs**, created in AWS and inserted in each request header.

Without the API Key a «forbidden error message» will be given.

CONSUMER APP JS

A NodeJS app is developed to **consume** the result published on a topic and **show** it on screen (in reality the result message would be showed on a screen near the hotel service door)

04 CODE IN DETAILS

Highlights of
implementation

Nuclio Trigger Rules

```
exports.handler = function(context, event) {
    var _event = JSON.parse(JSON.stringify(event));
    var _data = bin2string(_event.body.data);

    var extractedStrings = _data.split("-");
    var braceletId = parseInt(extractedStrings[0]);
    var serviceRequested = extractedStrings[1];

    if(check_wrong_service(serviceRequested)){
        //The requested service is wrong: it is not present in th
    e list of services
        send_result(braceletId + "-"+ serviceRequested + "-
    errorService");
    }else{
        //Ther requested service is one of the avalaible services
        check_requisites(braceletId, serviceRequested);
    }
```

Take and convert the message read on topic : (e.g. “1-gym”)

Check if there is an error in the service requested name

If the requested service exists, then check the requisites of the guest ->

VALIDATE_REQUEST_FUNCTION.JS (I)

```
function check_requisites(braceletId, serviceRequested){
```

...GET REQUEST: take from DB the last request made from guest with given braceletID

```
//Handling error inexistent ID in the DB
 if(body.Count == 0){
     //The bracelet ID that has issued the request is not registered into DB
     send_result(braceletId + "-"+ serviceRequested + "-errorBracelet");
       }else{

          //bracelet ID existent in DB
          lastRequest = body.Items[0];

          //check role requisites
```

Take the role of given braceletId and check if the guest
who made the request is allowed to access the service

...

VALIDATE_REQUEST_FUNCTION.JS (2)

```
if(reqResult){
                    resultString = "access allowed";
                    updatedAccessNumber = parseInt(lastR
    equest.accessesNumber.N) + 1;
                }else{
                    resultString = "access not allowed";
                    updatedAccessNumber = parseInt(lastR
    equest.accessesNumber.N);
                }

            //Save request to DB for further analytics

send_result(braceletId + "-"+ serviceRequested + "
    " + resultString);
```

Set the result of the access request depending on the role requisites of the guest

Publish on topic using AMQP, the result of the request to be consumed by another serverless function that shows the result

VALIDATE_REQUEST_FUNCTION.JS (3)

05 HOW TO EXECUTE: USAGE PATH

Installing and compiling
code to run the project

NUCLIO

Start Nuclio using a Docker Container and run its graphical user interface where you can create a project and add functions

```
$ docker run -p 8070:8070 -v /var/run/docker.sock:/var/run/docker.sock -v
/tmp:/tmp nuclio/dashboard:stable-amd64
```

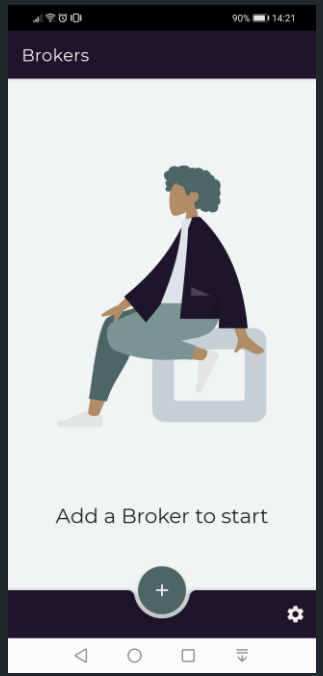
RABBIT MQ

Start RabbitMQ (message-broker) instance with MQTT enabled using Docker Container.

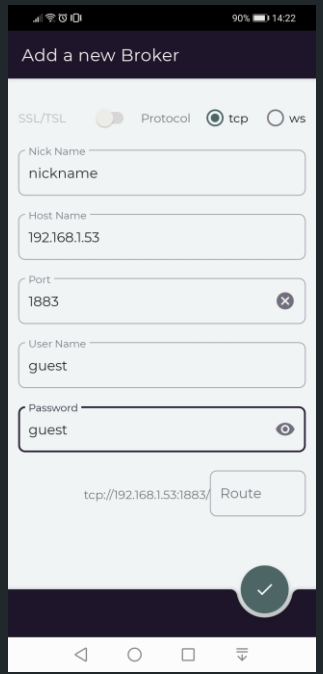
```
$ docker run -p 9000:15672  -p 1883:1883 -p 5672:5672  cyrilix/rabbitmq-mqtt
```

ANDROID MQTT CLIENTT

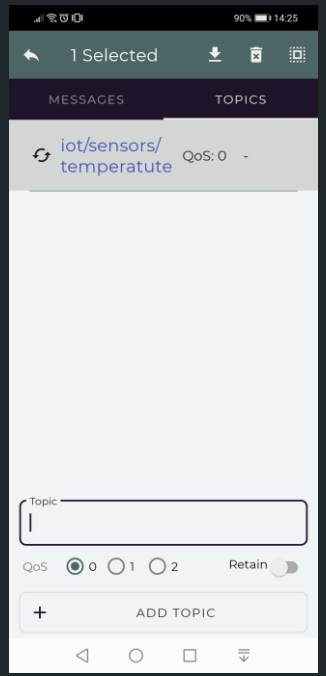
MQTIZER Android MQTT Client that simulates the event of an IoT Device that trigger the serverless function sending a message on topic using MQTT



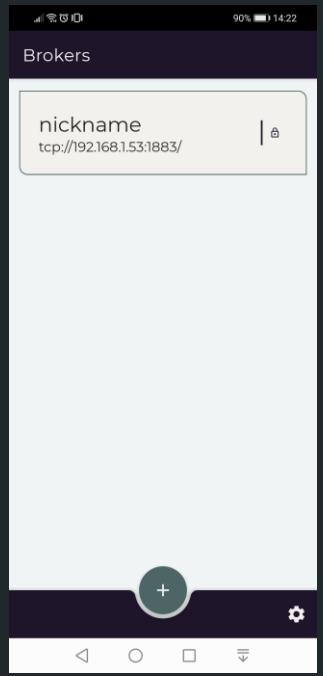
1



2



3

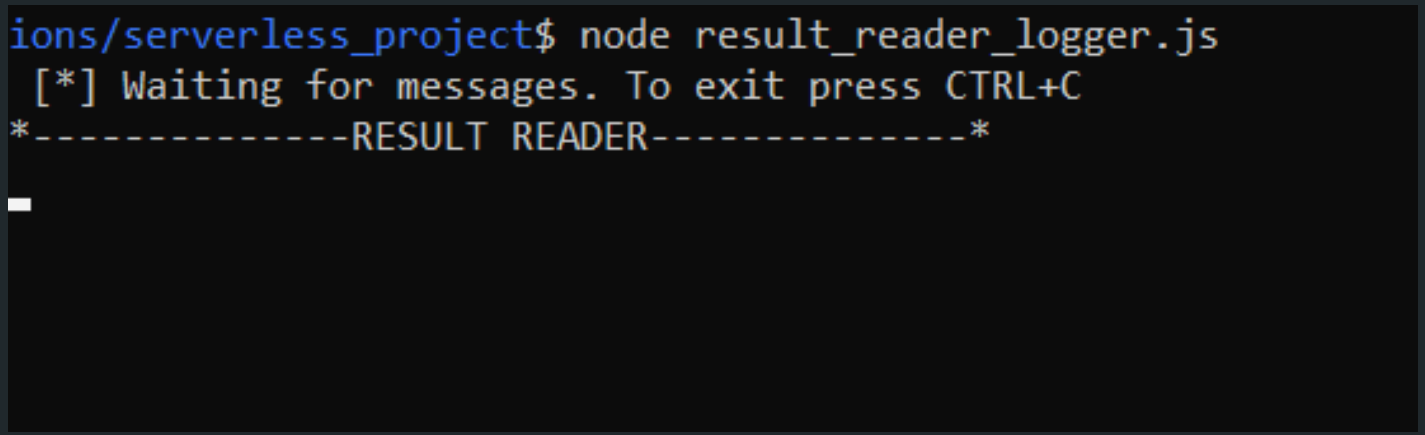


4

CONSUMER JS APP

Node JS application that represents the reader of the access request result.

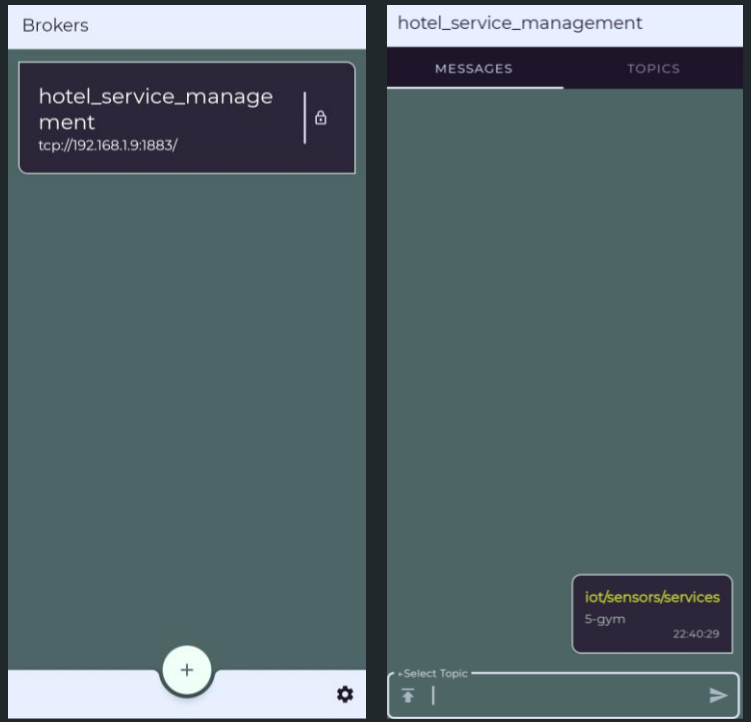
```
$ npm install amqplib
$ node result_reader_logger.js
```



06 RUNNING EXAMPLE

Screenshots of typical use case execution

RUNNING EXAMPLE (1)



RabbitMQ

(5-gym) → Iot/sensors/services

Nuclio function is triggered «validate_request_function»

RUNNING EXAMPLE (2)

(access allowed)

↓

Iot/sensors/result

Result_reader_logger.js

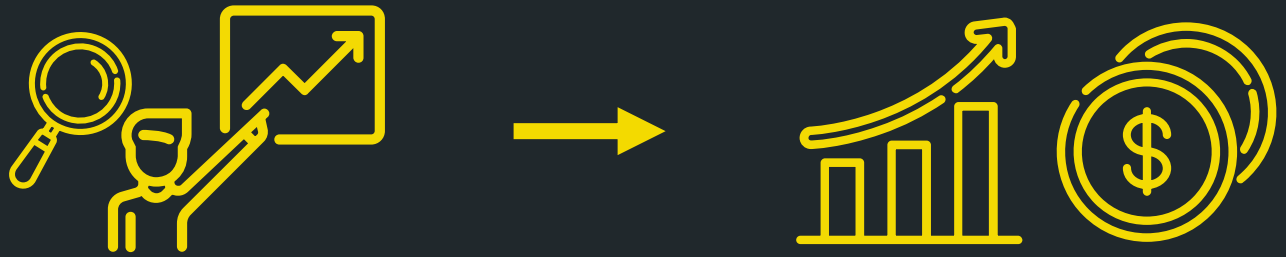
```
 [*] Waiting for messages. To exit press CTRL+C
*--------------RESULT READER--------------*


#Request of bracelet ID:  5
#Service requested:  gym
#Result:  access allowed
```

07 ANALYTICS

Analytics on data collected into database

ANALYTICS



Analytics is the analysis of data or statistics. It is used for the discovery, interpretation, and communication of meaningful patterns in data. Organizations may apply analytics to business data to describe, predict, and improve business performance.

ANALYTICS_ROLE

An example of data analytics: retrieve from DB the requests made by guests with a particular role (e.g. Bronze)

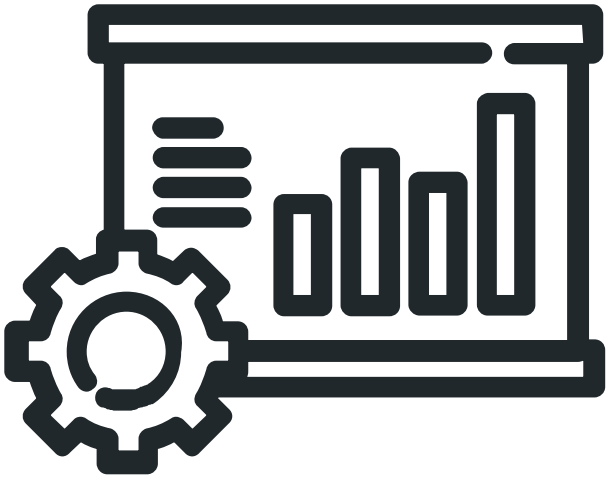
AttemptsNumber
Number of times a user has tried to access a hotel service

AccessesNumber
Number of times a user has effectively accessed a hotel service

If we find that a user has tried many times to access a hotel service **unsuccessful,** namely the number of attempts of this user is higher than normal in comparison with number of accesses this could be an insight that this user wants to obtain a higher level role to access more hotel services.

Promotional ads should be presented to this user in order to increase the potential profit.

OTHER POSSIBLE ANALYTICS FUNCTIONS



Access Frequence

Analyze all the services accessed in order to identify the access frequence and then boost the resources for the services more accessed

Peak/Low usage Periods

Analyze the dates of the requests to notice if there are periods of peak or low usage in order to decide in what period increase the advertising campaign

07 CONCLUSIONS

Conclusions and serverless benefits in this architecture

WHY SERVERLESS ARCHITECTURE

00 No maintenance of infrastructure

01

Rapid development and deployment

02

Ease of use

03

Enhanced Scalability

04

Lower Cost

THANKS

Nicola De Cristofaro
nico.de.cristofaro97@gmail.com
n.decristofaro2@studenti.unisa.it

Nicola De Cristofaro