Serverless Computing for IoT

Project Homeworks Lab.

ISISLab - Università degli Studi di Salerno

- Prof. Vittorio Scarano
- Carmine Spagnuolo, PhD
- Matteo D'Auria, PhD Student

Labs organization

- We talk (asynchronously) on the
  - Discord ISISLab community 

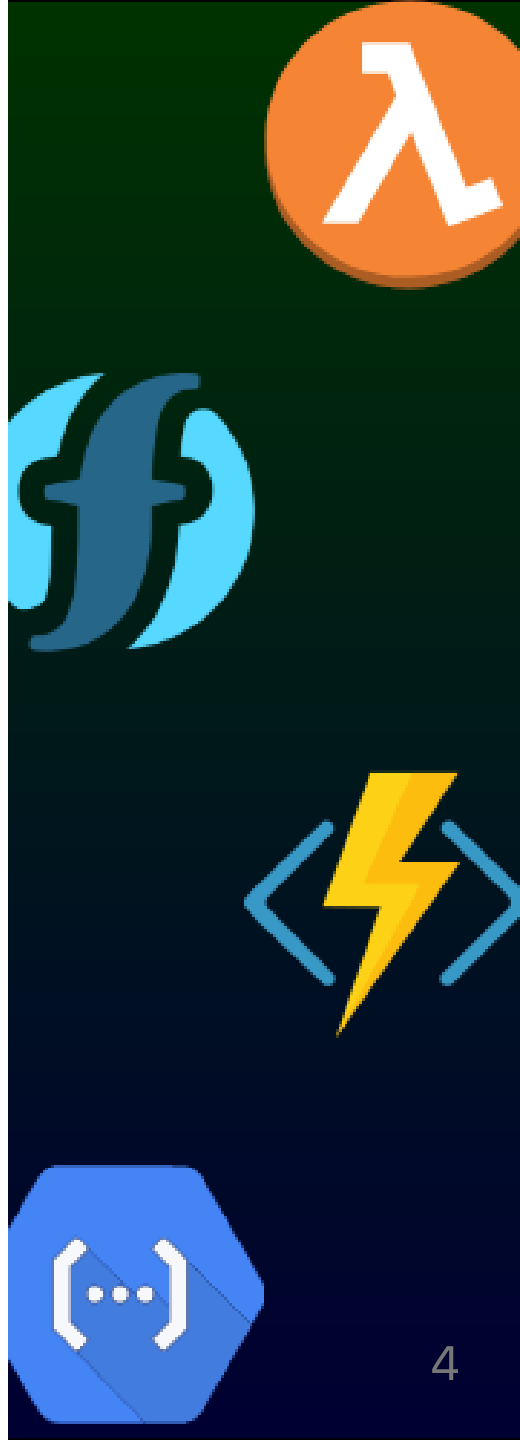
  - channel CLASSES `#sciot`
- or you can schedule an online meeting (synchronously) during the official course hours (on the same platform).
- please mention us `@Carmine Spagnuolo#6098 and @Matteo D'Auria#1998`.

Serverless Computing

- Serverless computing is a cloud-computing execution model in which the cloud provider acts as the server, dynamically managing the allocation of machine resources.

Function as a Service

- Cloud computing services that allow customers to:
  - develop,
  - run,
  - and manage application functionalities
- Advantage: no complexity of building and maintaining the hardware/software infrastructure.



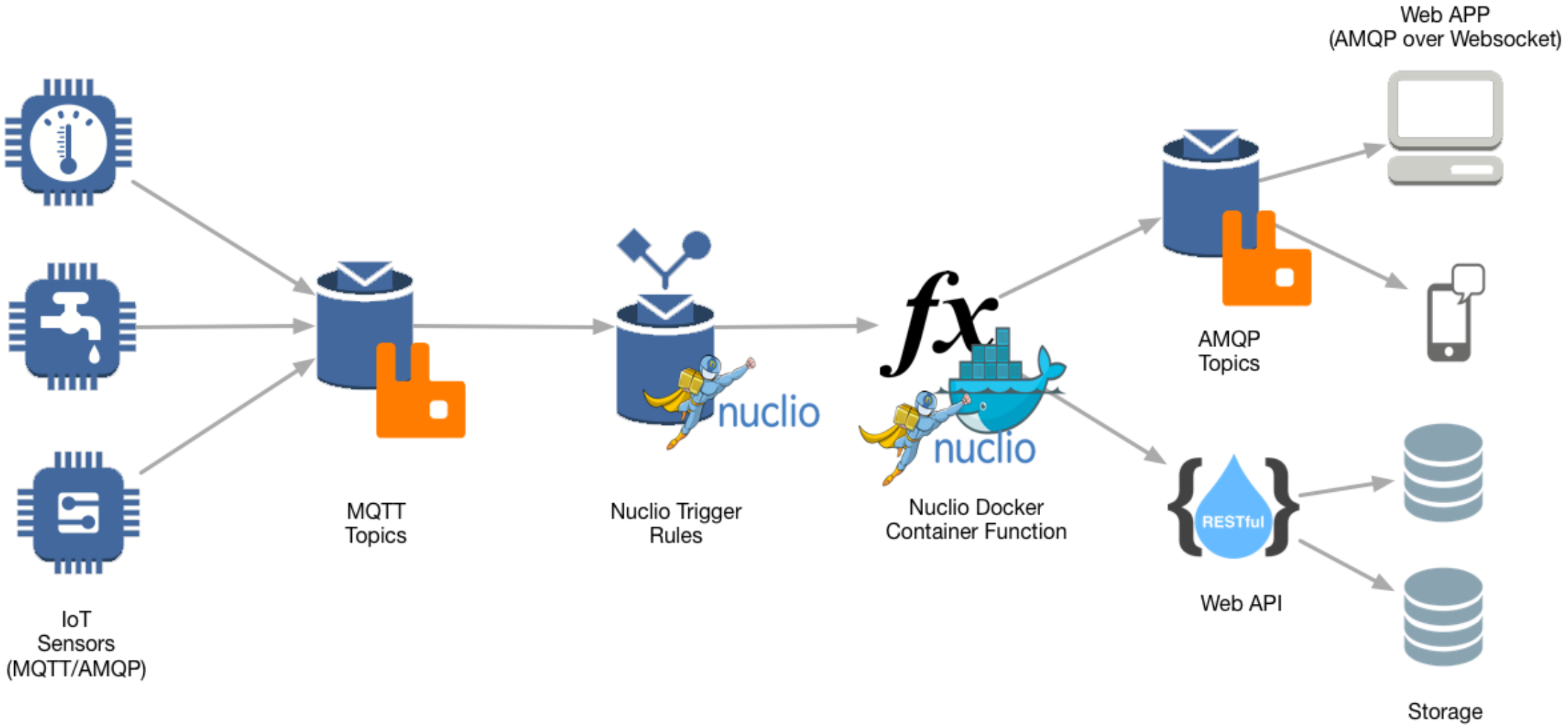
The Internet of Thing (IoT)

- A network of physical devices, vehicles, home appliances, and other items embedded with electronics, software, sensors, actuators, and connectivity, which enables these things to connect, collect, and exchange data.
- Course goal:
    - efficiently collect and elaborate data, produce new data as the answer to particular conditions computed from the data received.
    - integrate different cloud services to build an IoT application by using only open-source software.

Lesson Objectives

- *Design* a computing architecture, based on open-source software, that allows the users to exploit the Function-as-service model in the context of IoT.
- The system must allow to *deploy* functions that are *trigged* by events generated from small devices such as sensors and mobile (IoT devices)
- Use the *message-passing* communication paradigm, in particular on dedicated protocols as AMQP or MQTT.

Open-source Architecture



Architecture Components

- Ubuntu 18.04 LTS 

- Application containers engine Docker and Docker Compose 

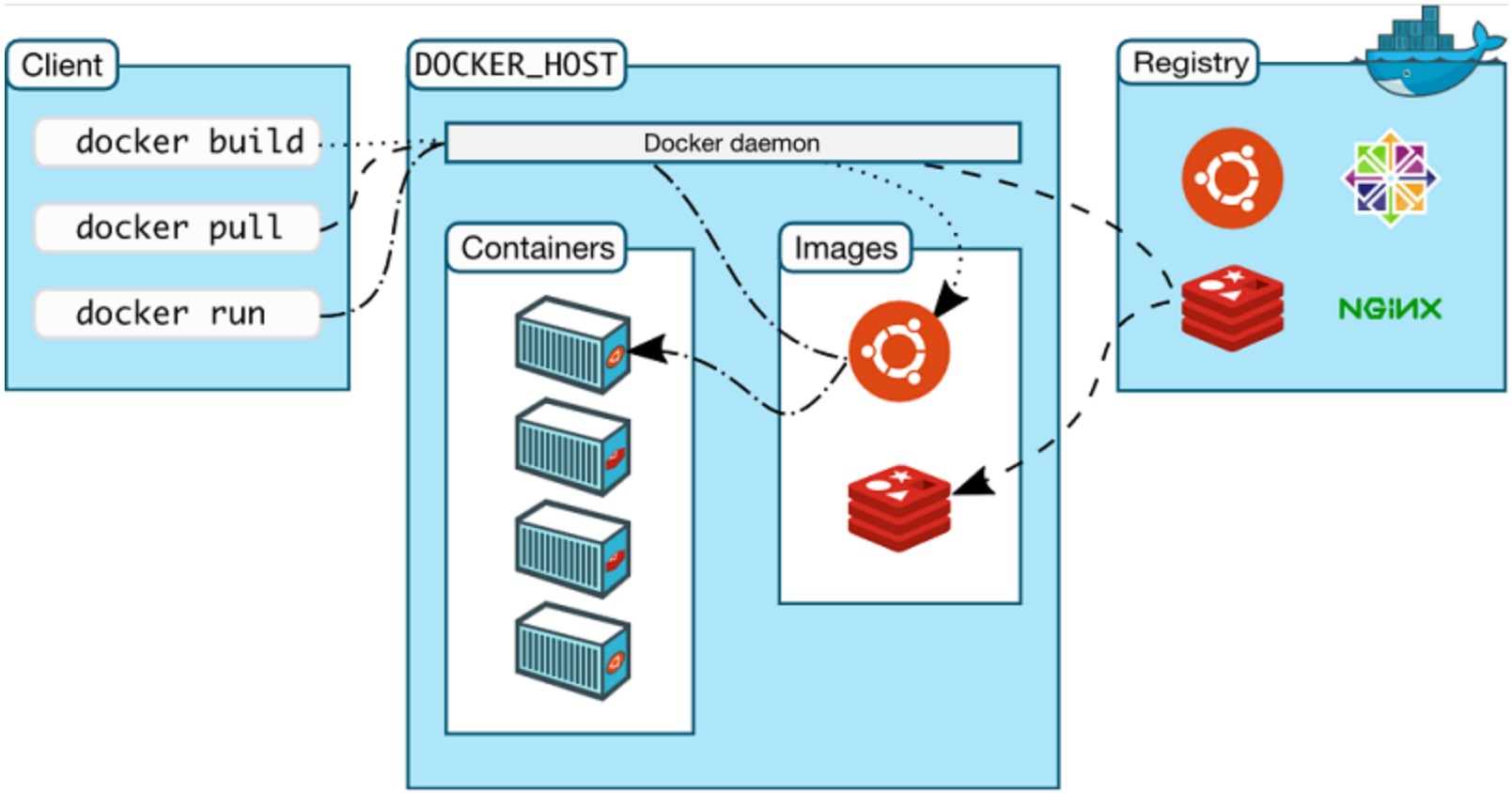
- Serverless computing provider Nuclio 

- AMQP and MQTT message broker RabbitMQ 

- JavaScript Application runtime Node.js 

Docker

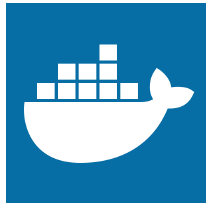
- Docker is a tool designed to make it easier to create, deploy, and run applications by using containers.



Docker Installation

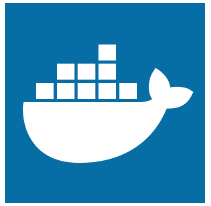
Install Docker using the Docker CE installation guide

```
$ sudo apt-get update
$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    software-properties-common
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
$ sudo apt-key fingerprint 0EBFCD88
$ sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"
$ sudo apt-get update
$ sudo apt-get install docker-ce
```

Docker IMPORTANT FIX 

- Ubuntu 18.04 changed to use systemd-resolved to generate /etc/resolv.conf.
- Now by default it uses a local DNS cache 127.0.0.53.
- It will not work inside a container, so Docker will default to Google's 8.8.8.8 DNS server, which may break for people behind a firewall.

```
sudo ln -sf /run/systemd/resolve/resolv.conf /etc/resolv.conf
```

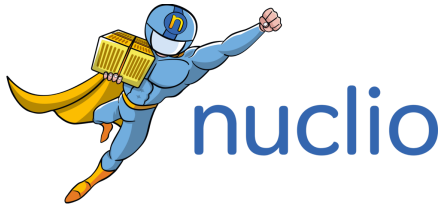
Docker Compose 

- Compose is a tool for defining and running multi-container Docker applications.
- With Compose, you use a YAML file to configure your application's services.
- Install Docker Compose using the Docker Compose installation guide.

```
$ sudo curl -L "https://github.com/docker/compose/releases/download/
1.22.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
$ sudo chmod +x /usr/local/bin/docker-compose
```

Nuclio  nuclio

- Nuclio is a new "serverless" project, derived from Iguazio's elastic data life-cycle management service for high-performance events and data processing.
- The simplest way to explore Nuclio is to run its graphical user interface (GUI) of the Nuclio dashboard.
- The Nuclio documentation is available at this link.

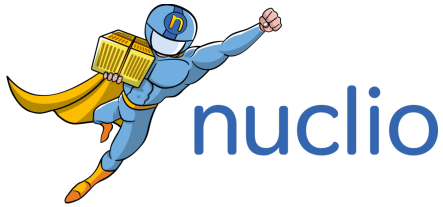
Nuclio 

- Start Nuclio using a docker container.

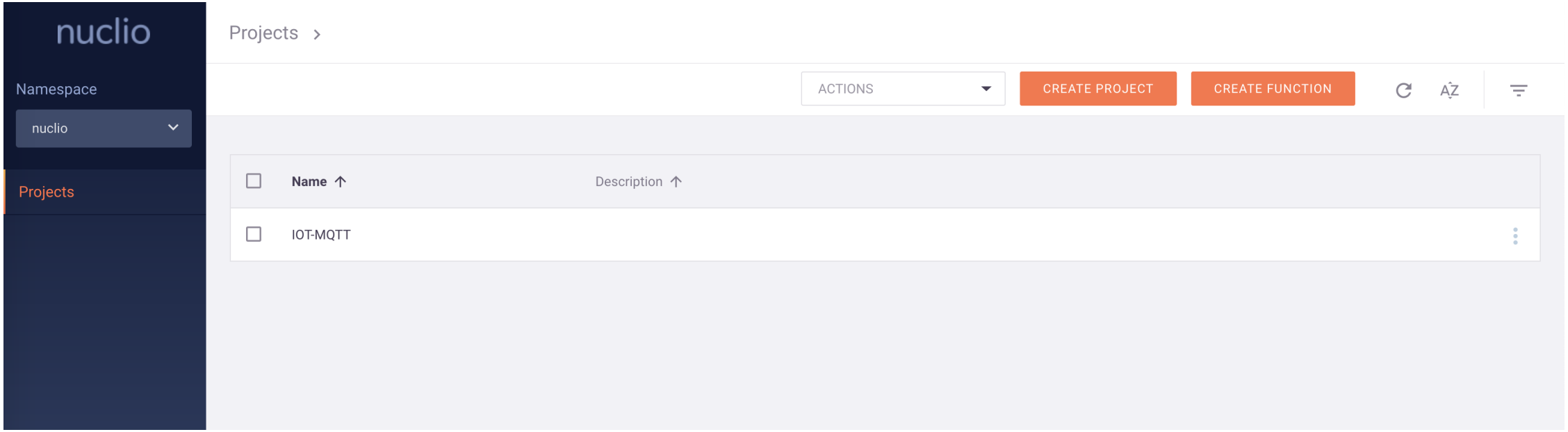
```
$ docker run -p 8070:8070 -v /var/run/docker.sock:/var/run/
docker.sock -v /tmp:/tmp nuclio/dashboard:stable-amd64
```

- Browse to http://localhost:8070, create a project, and add a function.
- When run outside of an orchestration platform (for example, Kubernetes or Swarm), the dashboard will simply deploy to the local Docker daemon.
- Nuclio provide also the nctcl application client that allows to basically execute the same operation of Nuclio dashboard.

Nuclio

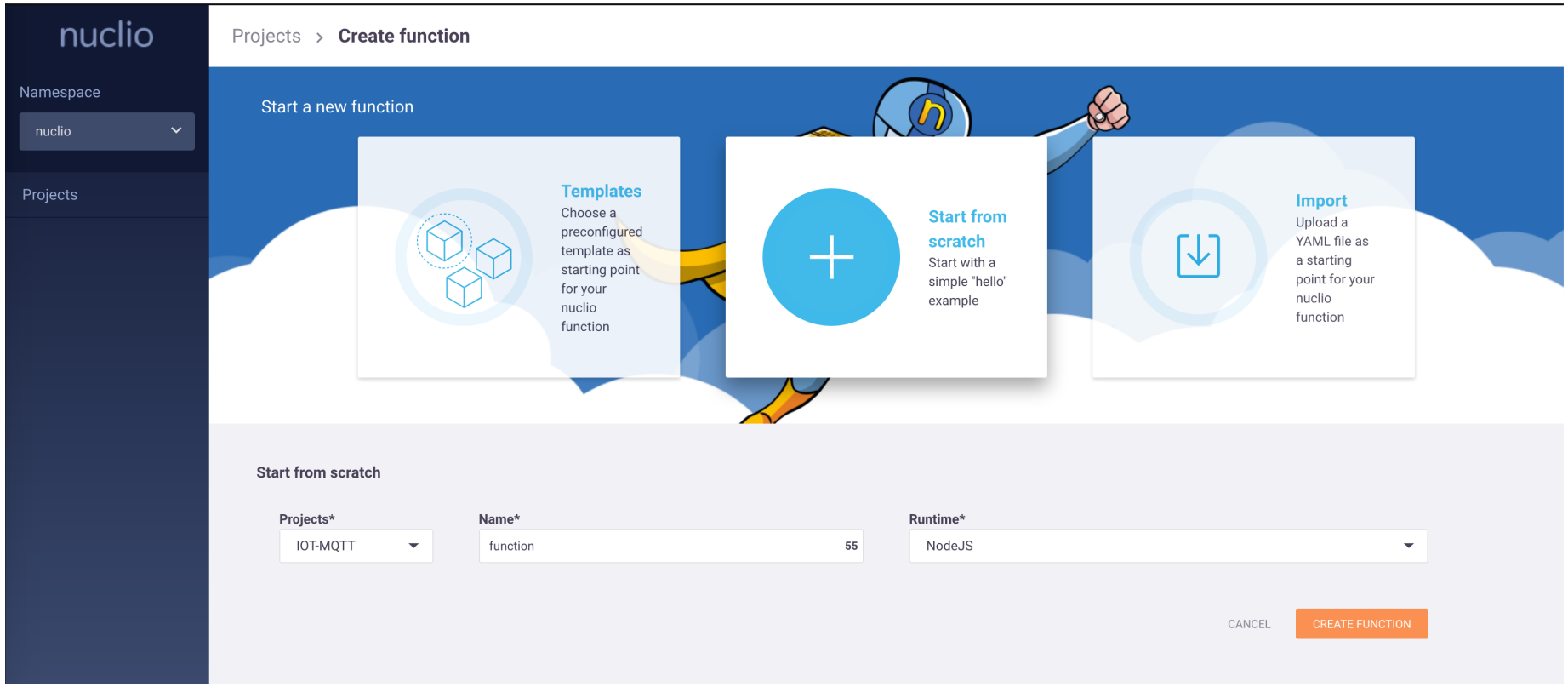


How to deploy functions in Nuclio using the Nuclio dashboard:



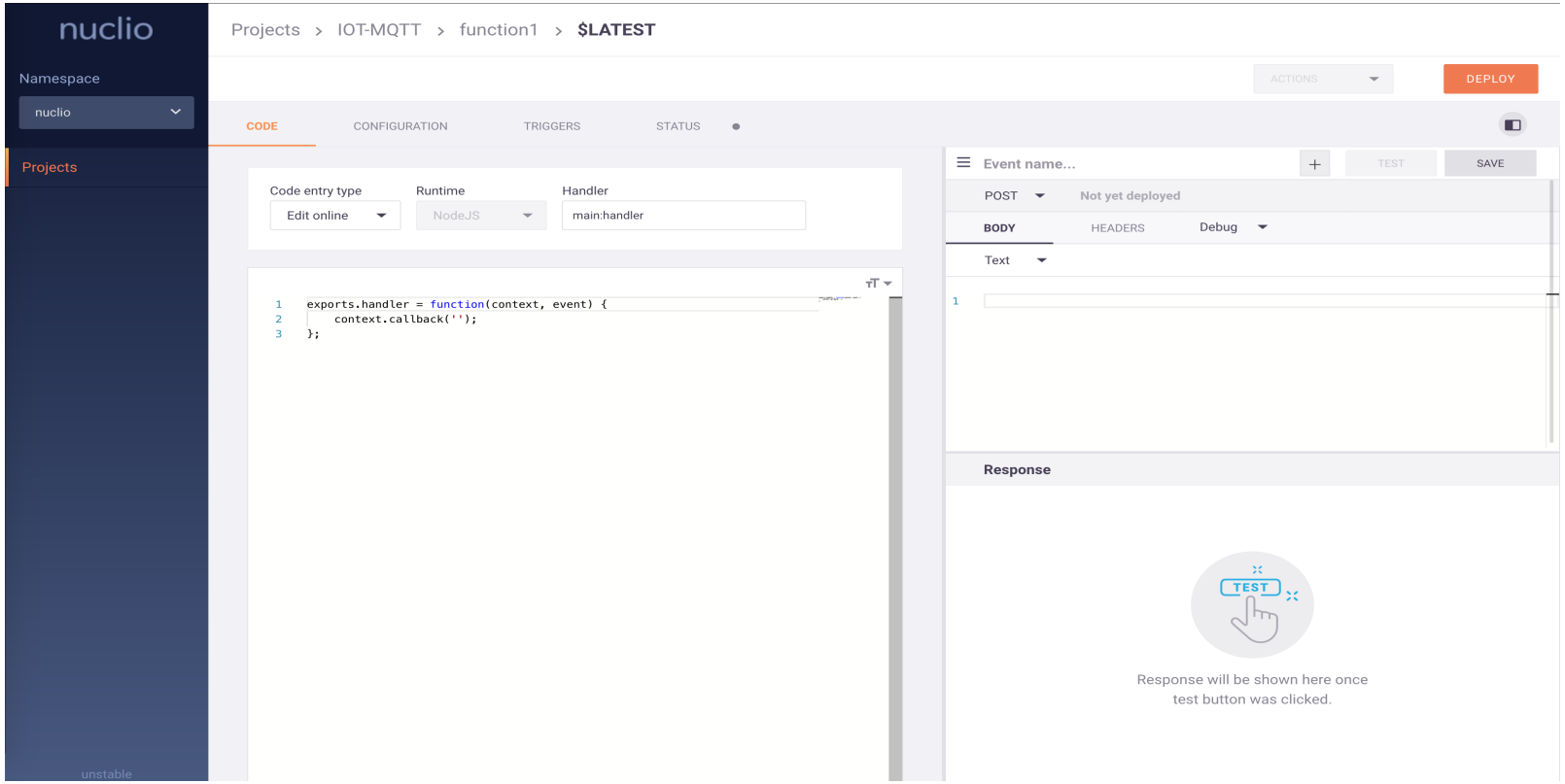
Nuclio

How to deploy functions in Nuclio using the Nuclio dashboard:

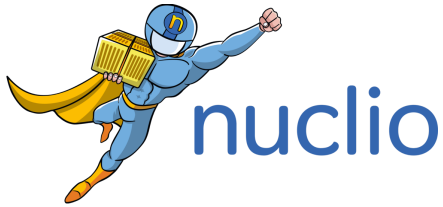


Nuclio

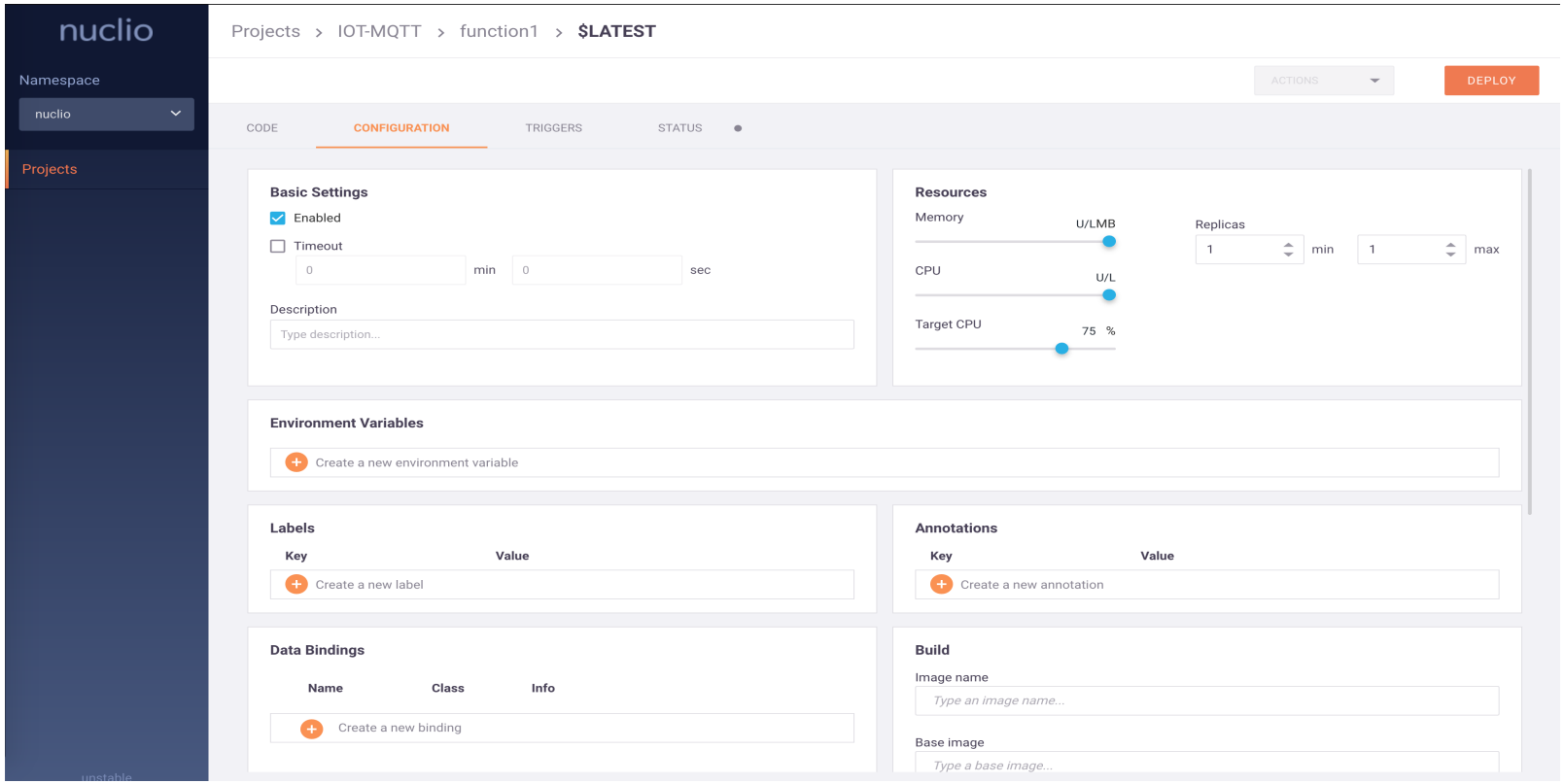
How to deploy functions in Nuclio using the Nuclio dashboard:



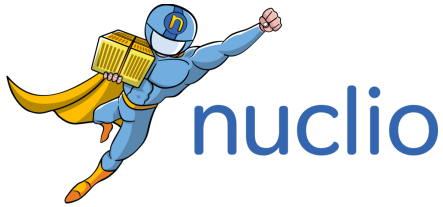
Nuclio



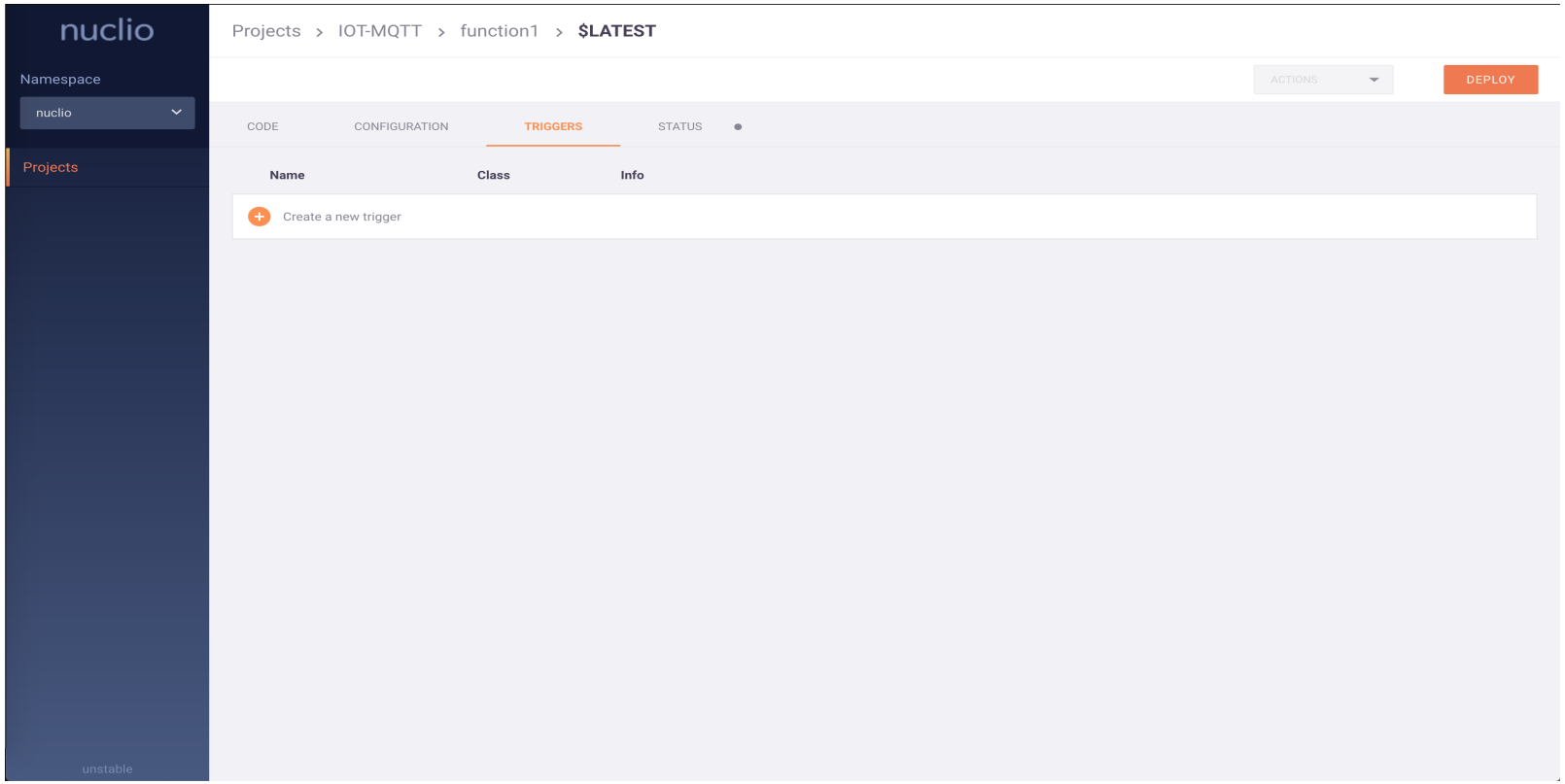
How to deploy functions in Nuclio using the Nuclio dashboard:



Nuclio

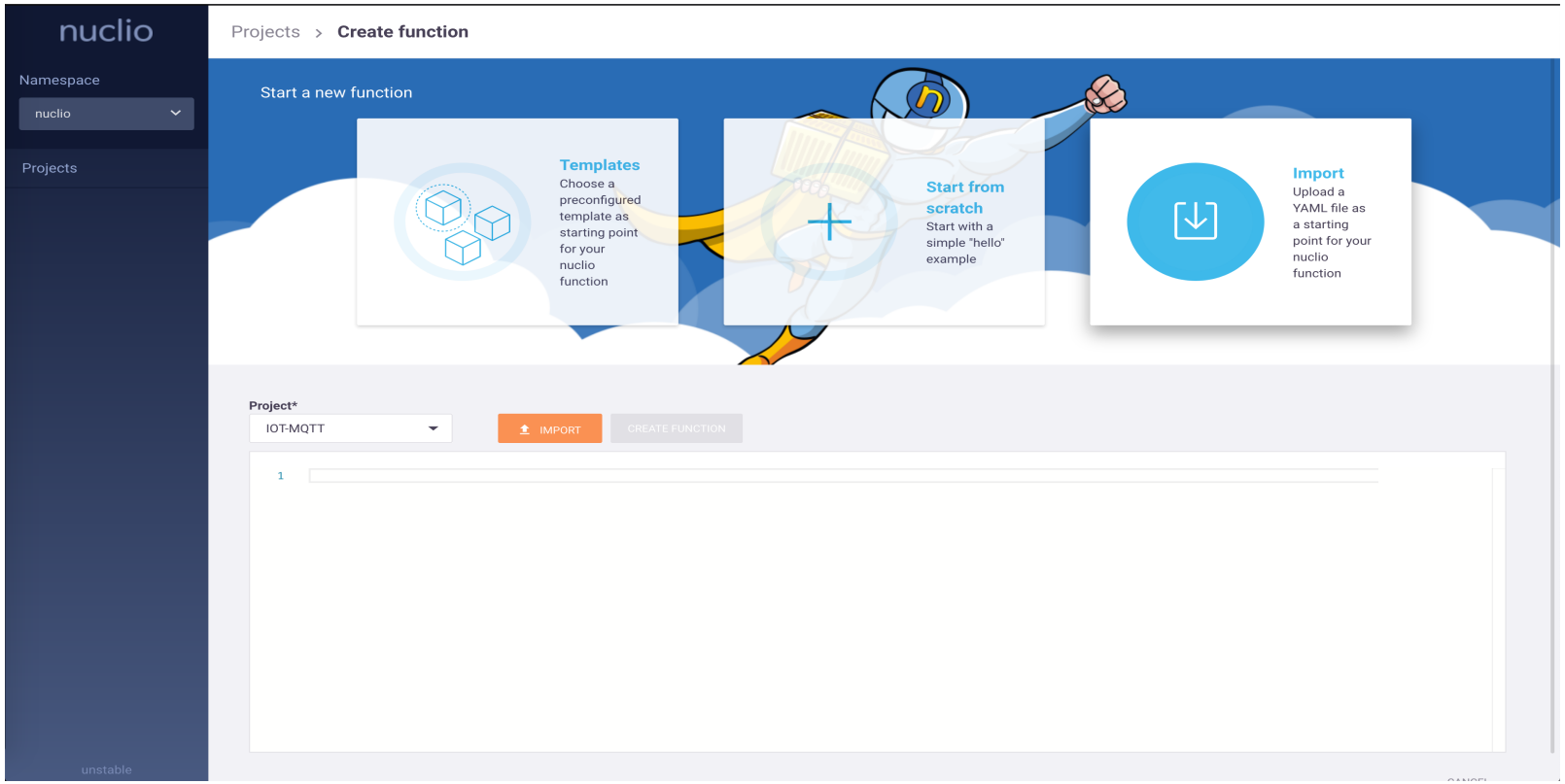


How to deploy functions in Nuclio using the Nuclio dashboard:



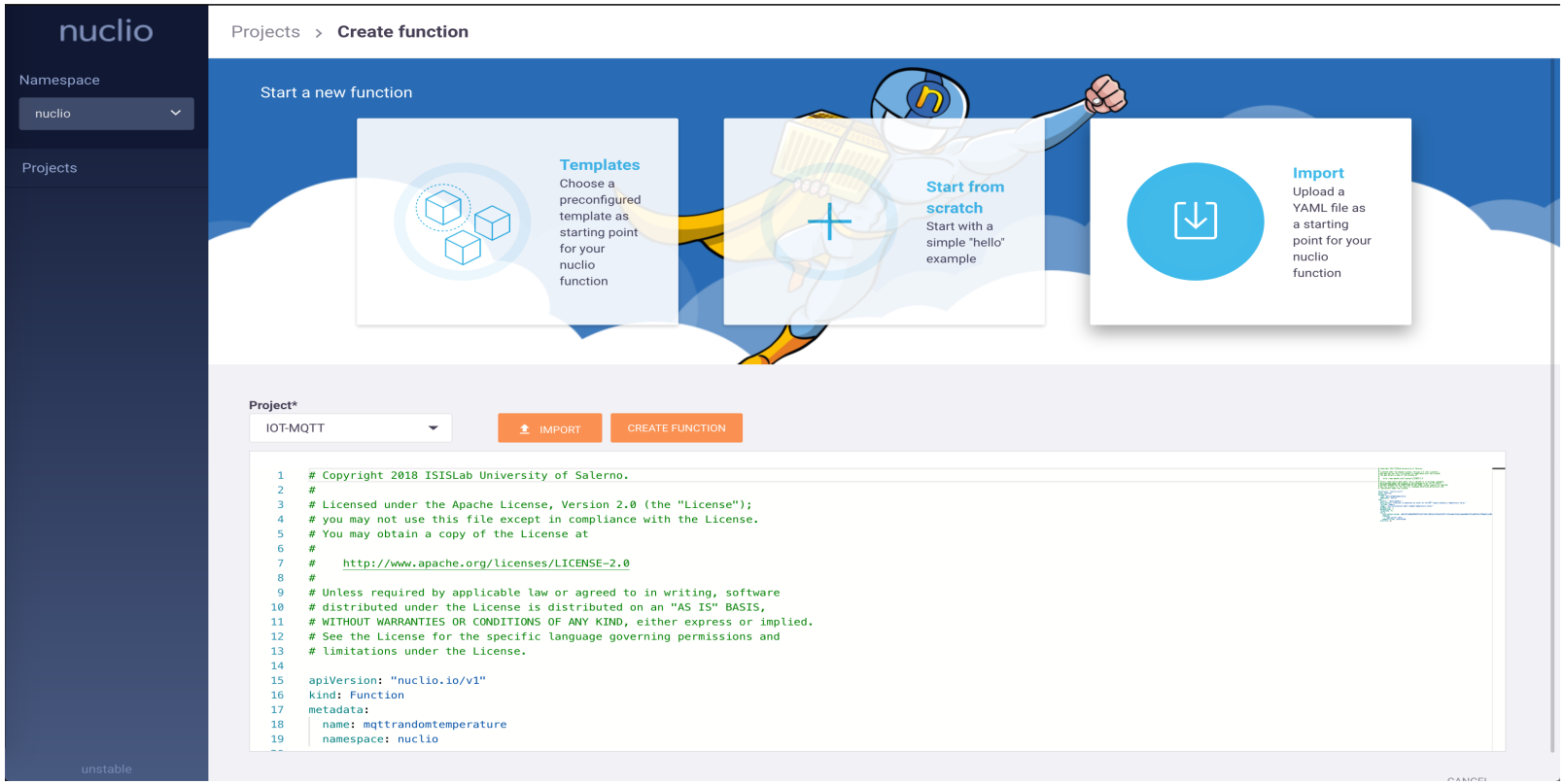
Nuclio

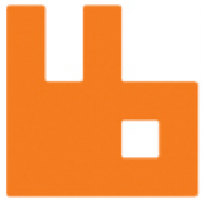
How to deploy functions in Nuclio using the Nuclio dashboard and .yaml file:



Nuclio

How to deploy functions in Nuclio using the Nuclio dashboard and .yaml file:



RabbitMQ 

- RabbitMQ is lightweight and easy to deploy on premises and in the cloud. It supports multiple messaging protocols. RabbitMQ can be deployed in distributed and federated configurations to meet high-scale, high-availability requirements.
- Start RabbitMQ instance with MQTT enabled using docker.

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

- Browse to http://localhost:9000, and login using username: guest and password: guest, to access to the RabbitMQ managment, where is possible to visualize the message queues and the broker status.

AMQP and MQTT JS clients 

- There are different libraries for many languages for interacting with protocol AMQP and MQTT you can use what you want. We use:
  - AMQP 0-9-1 for Javascript AMQP.
  - MQTT.js for JavaScript MQTT.

We used a general purpose MQTT client for Android.

Docker useful commands 

- Docker container ID: `sudo docker ps -a`
  - displays all deployed containers, remember that functions are Docker containers, the IMAGE field is the function name and version, while the CONTAINER ID is the ID of the function. In this view, it is also possible to see the listening port for the function in the field PORTS.
- Docker Logs: `sudo docker logs CONTAINER ID`
  - displays the STDOUT and STDERR of the associated container.
- Docker Kill: `sudo docker kill CONTAINER ID`
  - kills the associated container.
- Docker Remove: `sudo docker rm CONTAINER ID`
  - removes the associated container.

MQTT Temperature Example

Application Goals 

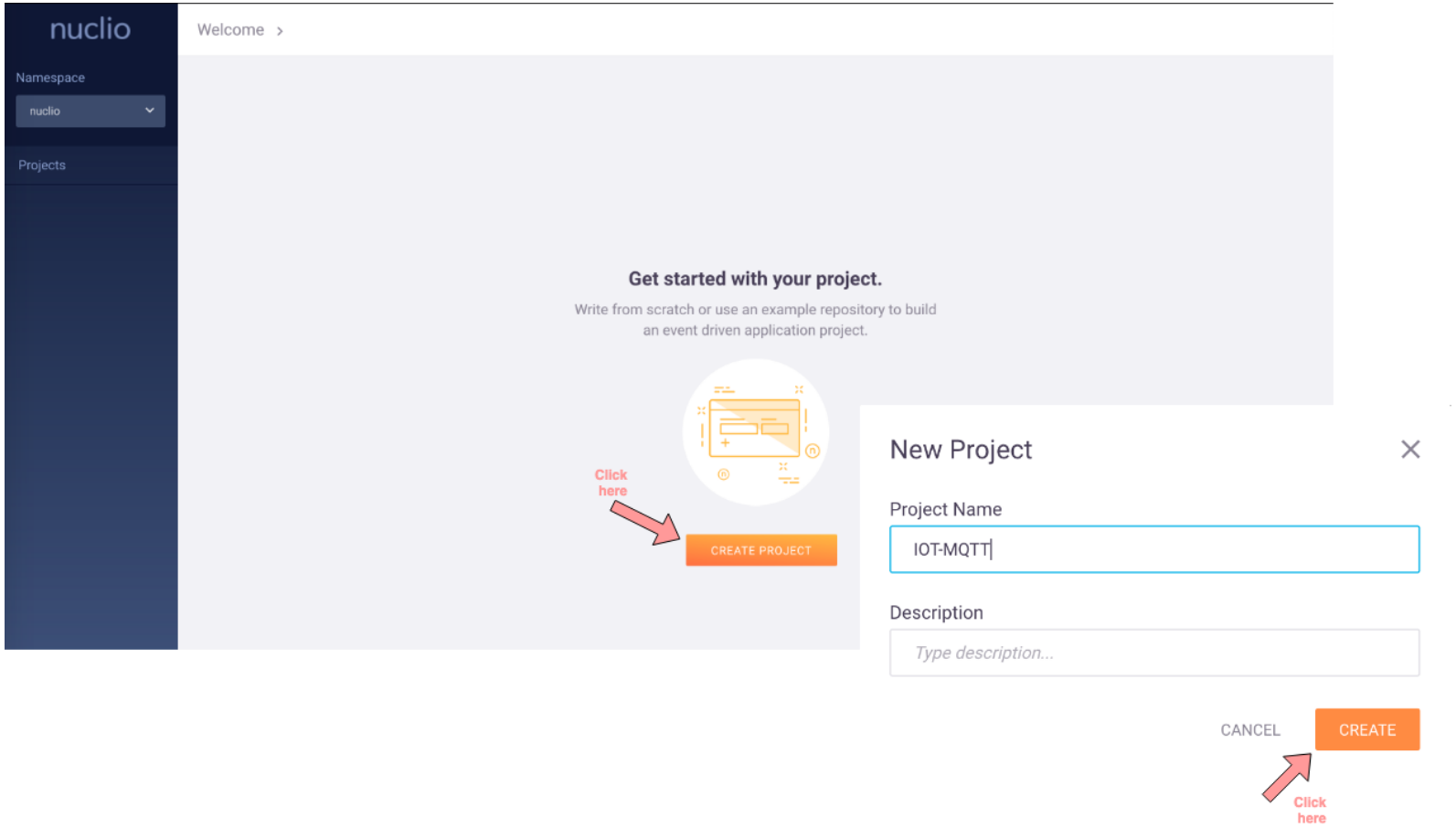
The temperature example aims to demonstrate the potential of the suggested architecture to collect data from IoT sensors and logging this data on an external data manager.

Scenario:

- We have several sensors that detect the temperature.
- These sensors send these temperatures through the use of the MQTT protocol.
- These temperatures are processed in a Serverless way.

First Step

- The first step to do is access to the Nuclio dashboard and create a new project named IOT-MQTT.

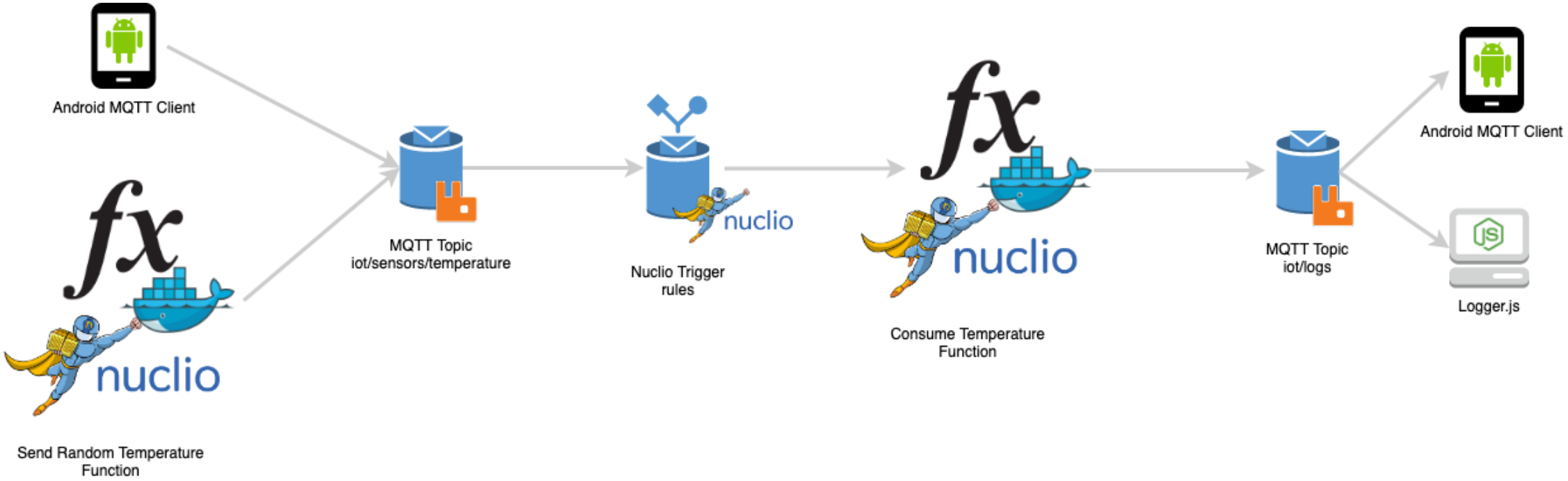


MQTT Temperature Example

The application is composed by four functions:

- Temperature Consume Function triggered by a new MQTT message on the topic "iot/sensors/temperature".
- Send Random Temperature Function sends a new temperature value on the MQTT on the topic "iot/sensors/temperature".
- Logger logs the invocation of the consume function, this function is in waiting for new messages on the queue AMQP "iot/logs". It is a JavaScript function for Node.js and is executed on an external machine.
- IoT Client a general purpose Android MQTT Client.

MQTT Temperature Example



MQTT Temperature Example

Temperature Consume Function

Temperature Consume Function

```
var amqp = require('amqplib');
        var FUNCTION_NAME = "mqttconsume";
        function send_feedback(msg){
            var q = 'iot/logs';
            amqp.connect('amqp://guest:guest@172.16.15.52:5672').then(function(conn) {
                return conn.createChannel().then(function(ch) {
                    var ok = ch.assertQueue(q, {durable: false});
                    return ok.then(function(_qok) {
                    ch.sendToQueue(q, Buffer.from(msg));
                    console.log(" [x] Sent '%s'", msg);
                    return ch.close();
                    });
                }).finally(function() {
                        conn.close();
                    });
            }).catch(console.warn);
        }
```

Temperature Consume Function

```
function bin2string(array){
  var result = "";
  for(var i = 0; i < array.length; ++i){
    result+= (String.fromCharCode(array[i]));
  }
  return result;
}

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

    context.callback("feedback "+_data);

    console.log("TRIGGER "+_data);
    send_feedback("Invoked Function MQTT: "+FUNCTION_NAME+" received "+_data);
};
```

Temperature Consume Function

- We deploy using the Nuclio's Docker compose specifics:
  - *.yaml* file that declares all functions specifications and source code.
  - function code (*JavaScript*) is encoded in base64 and copied in the attribute "functionSourceCode".
  - triggered with MQTT events on topic `iot/sensors/temperature`.
- We use the *amqplib* to send feedback to a log server:
  - "commands" attribute ➡️ `npm install amqplib`.
- ⚠️ Change the user, password and IP for the log server and for the MQTT trigger (yaml file `url` field):

```
...
amqp.connect('amqp://guest:guest@172.16.15.52:5672').then(function(conn) ...
```

Temperature Consume Function

```yaml
apiVersion: "nuclio.io/v1"
kind: Function
metadata:
  name: mqttconsume
  namespace: nuclio
spec:
  handler: "main:handler"
  description: "Function the is ..."
  runtime: nodejs
  image: "nuclio/processor-mqttconsume:latest"
  minReplicas: 1
  maxReplicas: 1
  targetCPU: 75
  triggers:
    myMqttTrigger:
      kind: "mqtt"
      url: "guest:guest@172.16.15.52:1883"
      attributes:
          subscriptions:
          - topic: iot/sensors/temperature
            qos: 0
  build:
    functionSourceCode: <base64 of the code>
    commands:
      - 'npm install amqplib'
    codeEntryType: sourceCode
  platform: {}
```

Temperature Consume Function

- From Nuclio dashboard create a new project `IOT-MQTT`.
- Create a new function `Consume`
    - select import from yaml, and load the file `iot/mqtt/temperature/amqpconsume.yaml`;
    - press Deploy (see the Error log).
- You can build your function manually by pasting the code, setting the MQTT trigger, and adding the *amqplib* in the install commands:
    - ⚠️ MQTT trigger must be ➡️ `user:passowrd@IP:1883`;
    - don't fill the user and password text field.

MQTT Temperature Example

Send Random Temperature Function

Send Random Temperature Function

```
var mqtt = require('mqtt'), url = require('url');

var mqtt_url = url.parse(process.env.CLOUDAMQP_MQTT_URL || 'mqtt://guest:guest@172.16.15.52:1883');
var auth = (mqtt_url.auth || ':').split(':');
var url = "mqtt://" + mqtt_url.host;

var options = {
  port: mqtt_url.port,
  clientId: 'mqttjs_' + Math.random().toString(16).substr(2, 8),
  username: auth[0],
  password: auth[1],
};
...
```

- ⚠️ Change the user, password and IP for the `mqtt_url` variable:

```
var mqtt_url = url.parse(process.env.CLOUDAMQP_MQTT_URL || 'mqtt://guest:guest@172.16.15.52:1883');
```

Send Random Temperature Function

```
...
exports.handler = function(context, event) {
    var client = mqtt.connect(url, options);

    client.on('connect', function() {
        client.publish('iot/sensors/temperature', (Math.floor(Math.random()*30)).toString(), function() {
                    client.end();
                    context.callback('MQTT Message Sent');
                } );

        });

};
```

Send Random Temperature Function

```
apiVersion: "nuclio.io/v1"
kind: Function
metadata:
  name: mqttrandomtemperature
  namespace: nuclio
spec:
  handler: "main:handler"
  description: "Function to generate an event on the MQTT queue sending a temperature value."
  runtime: nodejs
  image: "nuclio/processor-mqtt-random-temperature:latest"
  minReplicas: 1
  maxReplicas: 1
  targetCPU: 75
  build:
    functionSourceCode: <base64 of the code>
    commands:
      - 'npm install mqtt'
    codeEntryType: sourceCode
  platform: {}
```

Send Random Temperature Function deploy

- Create a new function `Produce`
  - select import from yaml, and load the file `iot/mqtt/temperature/amqpevent.yaml`;
  - press Deploy (see the Error log).
- You can build your function manually by pasting the code, setting the MQTT trigger and adding the *amqplib* in the install commands.

Invoke Send Random Temperature Function

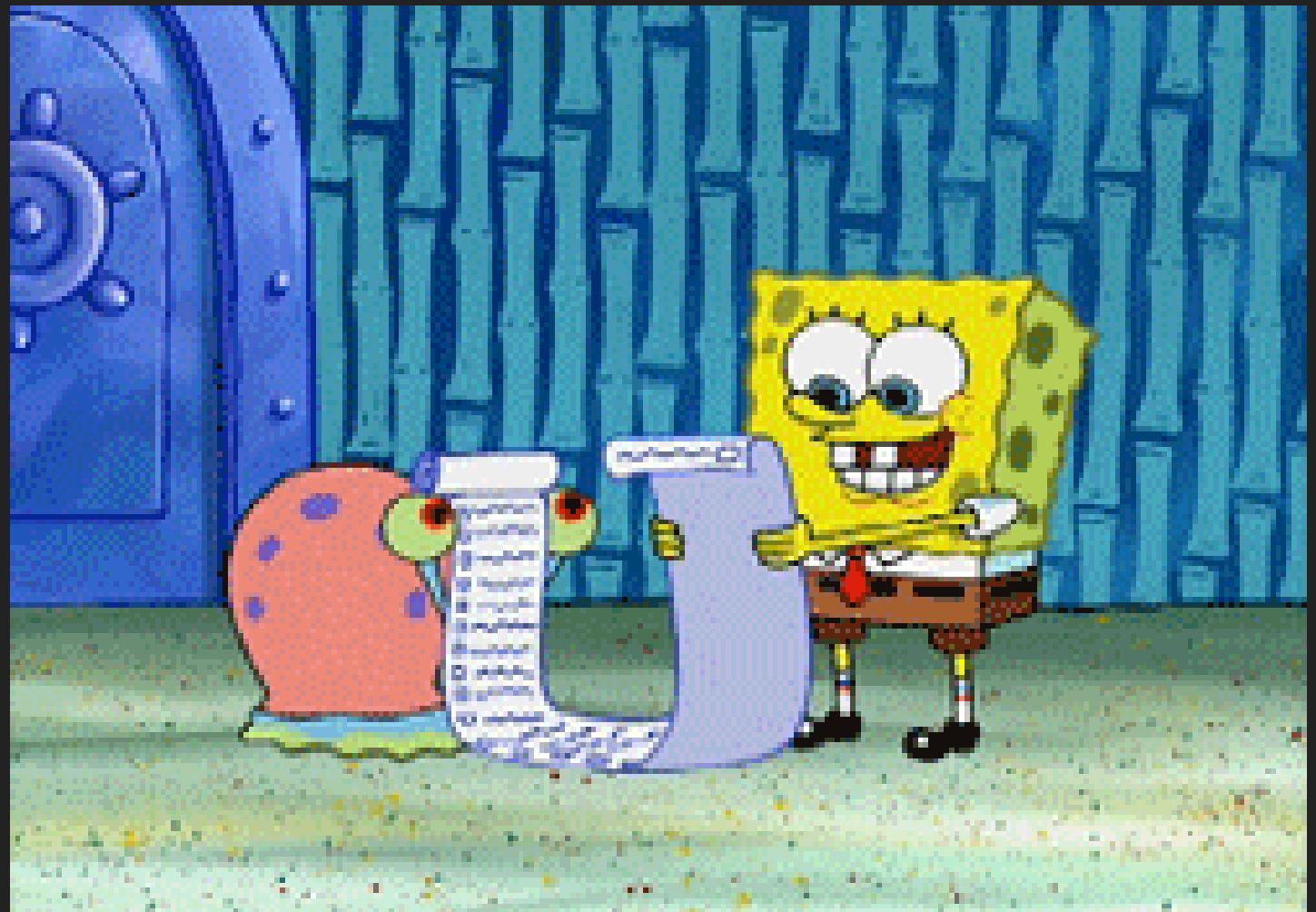
- For invoking the function is possible to press the button `TEST` in the dashboard.
- Or invoke the function by generating an *HTTP event* using the command line tool curl

```
curl -X POST -H "Content-Type: application/text"  http://localhost:39823
```

Notice: each Nuclio' function is identified by the serving port, you can see the serving port in the dashboard (change the port in the url http://localhost:PORT).

MQTT Temperature Example

Logger Server



Logger Server

```
var amqp = require('amqplib');
amqp.connect('amqp://guest:guest@172.16.15.52:5672').then(function(conn) {
  process.once('SIGINT', function() { conn.close(); });
  return conn.createChannel().then(function(ch) {
    var ok = ch.assertQueue('iot/logs', {durable: false});
    ok = ok.then(function(_qok) {
      return ch.consume('iot/logs', function(msg) {
        console.log(" [x] Received '%s'", msg.content.toString());
      }, {noAck: true});
    });
    return ok.then(function(_consumeOk) {
      console.log(' [*] Waiting for messages. To exit press CTRL+C');
    });
  });
}).catch(console.warn);
```

- ⚠️ Change the user, password and IP for the `mqtt_url` variable:

```
...
amqp.connect('amqp://guest:guest@172.16.15.52:5672').then..
```

Execute the Logger

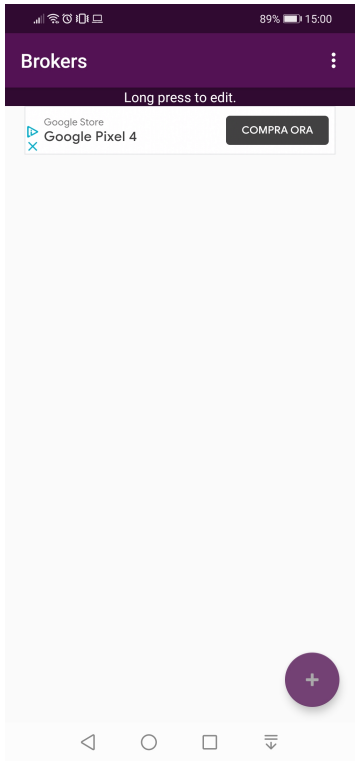
Running the server using the Node.js runtime.

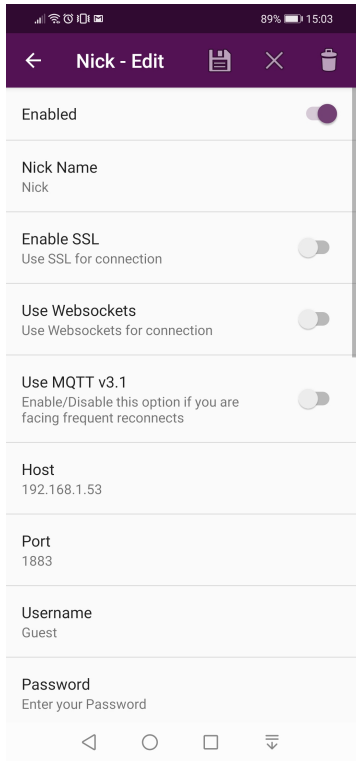
```
$ npm install amqplib
$ node logger.js
```

Android IoT Client (1)

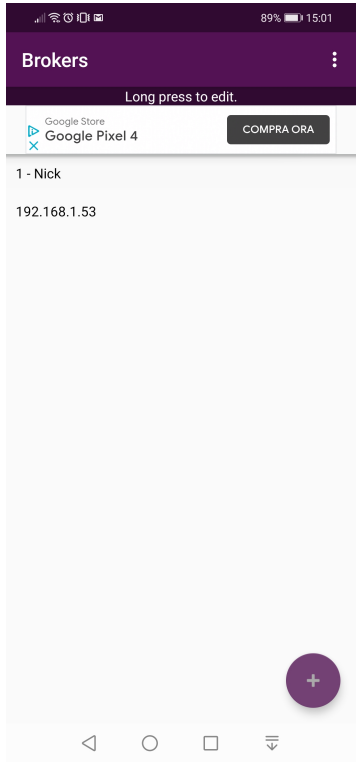
- MQTT Client.
- In this app, you can connect to the RabbitMQ broker using the protocol MQTT (create a new connection to the IP where the RabbitMQ is running).
- After creating the connection, you can easily send or receive values on some topics.

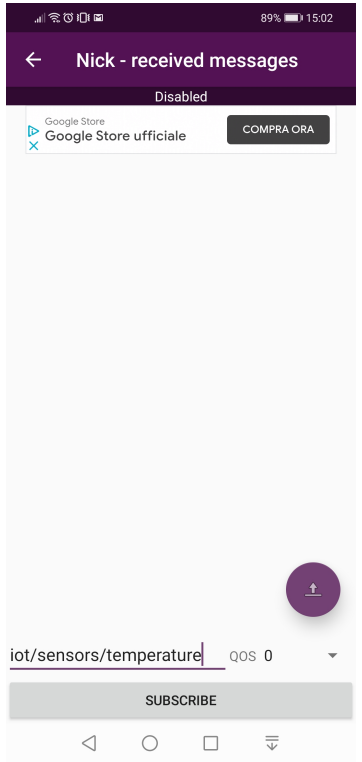
Android IoT Client (1) - Connection

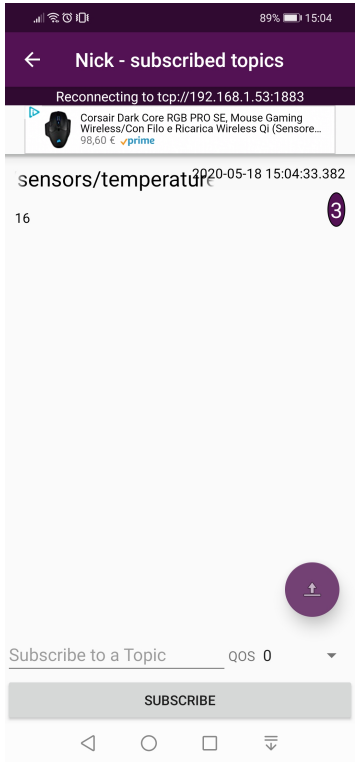




Android IoT Client (1) - Subscribe and send/receive message



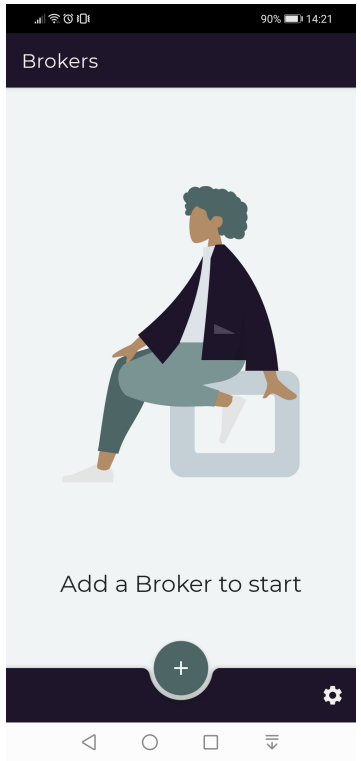


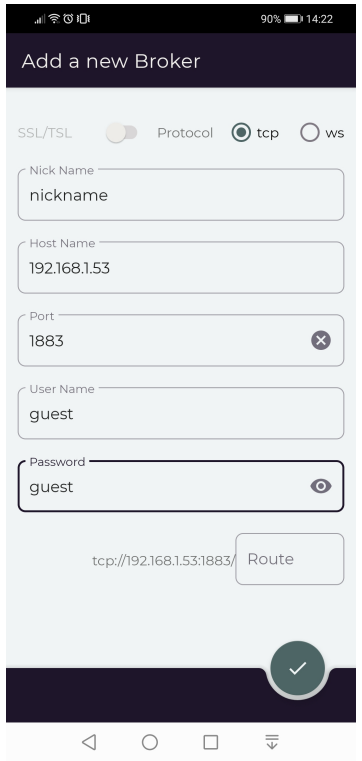


Android IoT Client (2) 🌟

- MQTIZER - Free MQTT Client.
- In this app, you can connect to the RabbitMQ broker using the protocol MQTT (create a new connection to the IP where the RabbitMQ is running).
- After creating the connection, you can easily send or receive values on some topics.

Android IoT Client (2) 🌟 - Connection





Android IoT Client (2) ✴ - Subscription and send/receive message