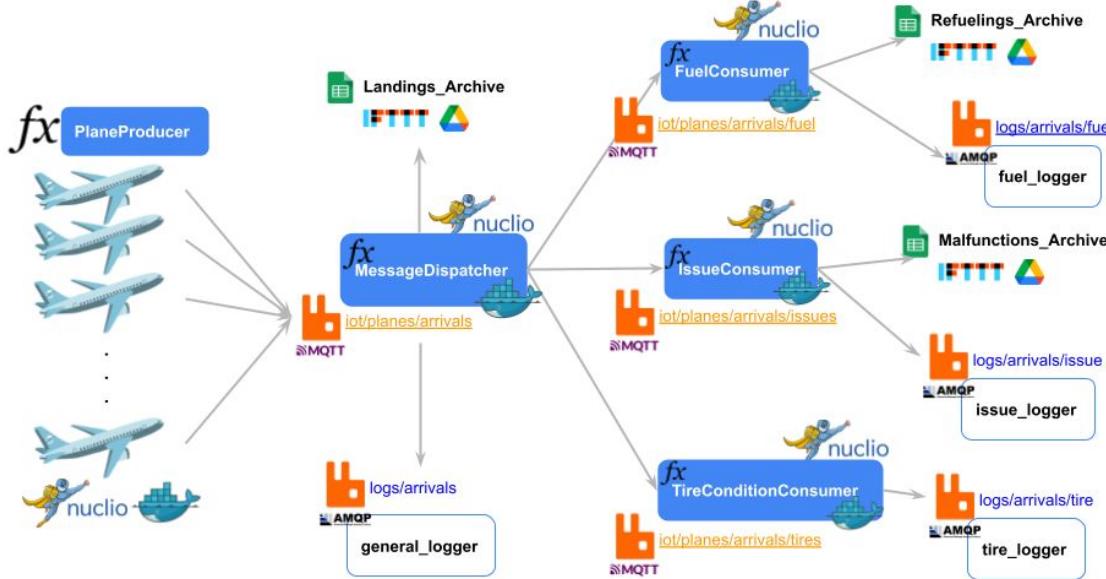


Serverless Computing for IoT

2019/2020



Plane-System



Prof:

Vittorio Scarano
Carmine Spagnuolo
Matteo D'Auria

Student:

Antonio Giulio 0522500732

Plane-System

Contents

1

Introduction

2

Tools

3

Scenario

4

Planes Abstraction

5

Architecture

6

IFTTT

7

How to Run

A wide-angle photograph of an airport at dusk or night. A large, modern passenger airplane is parked on the tarmac in the foreground, its body angled towards the left. The aircraft has a dark tail and a light-colored fuselage. In the background, a long, modern airport terminal building with a curved glass facade is visible. A jet bridge extends from the terminal towards the plane. Several tall, illuminated airport lights stand on poles along the tarmac. The sky is a vibrant orange and yellow, suggesting sunset or sunrise. A small, solid orange rectangular box is positioned in the upper right corner of the image.

Introduction

INTRODUCTION



The **IoT** industry is revolutionizing the
Aerospace & Defence sector.



AEROSPACE & DEFENCE

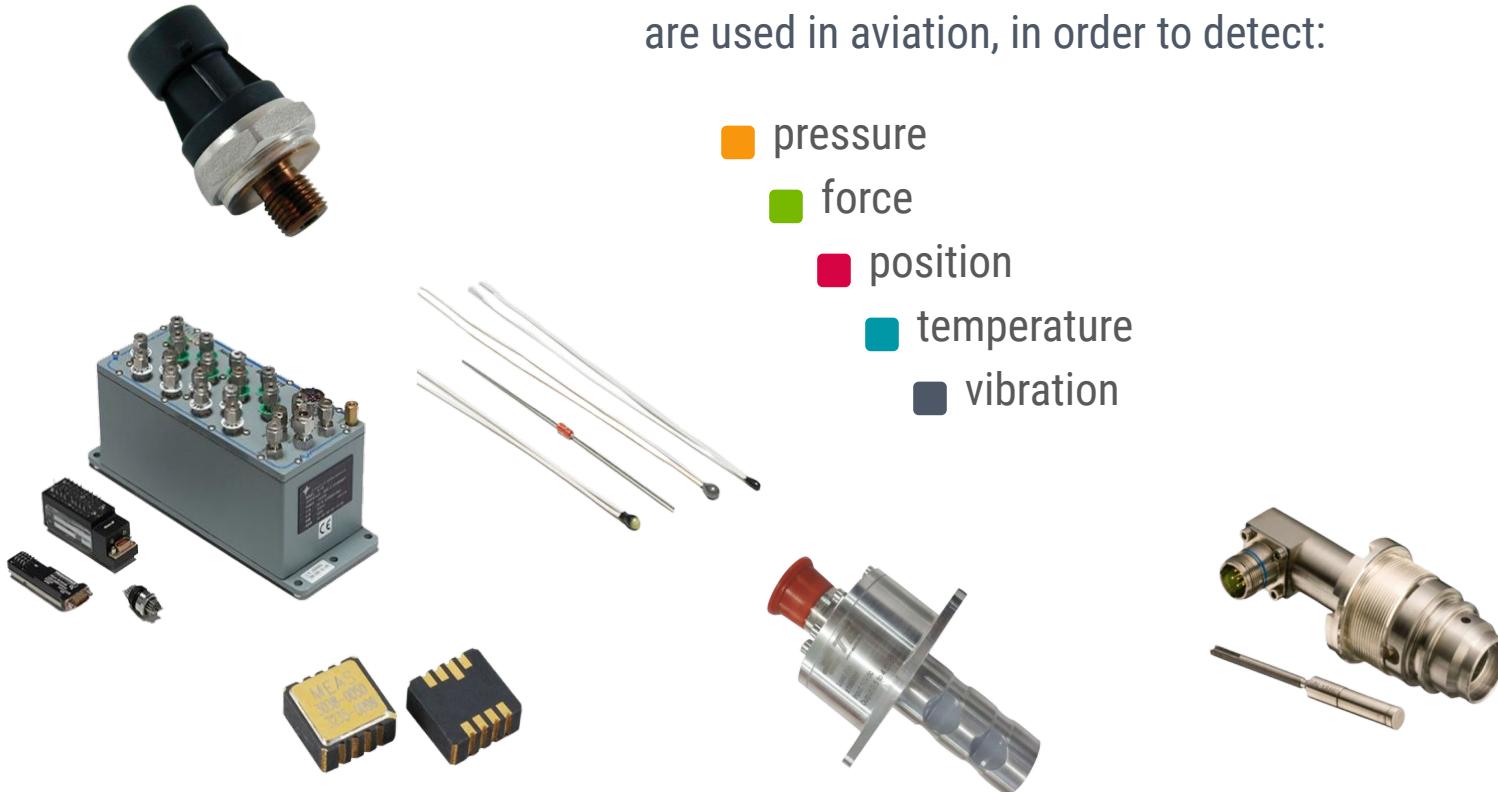
The IoT industry is revolutionizing the
Aerospace & Defence sector.

Last year were invested over **91.5**
billion dollars.



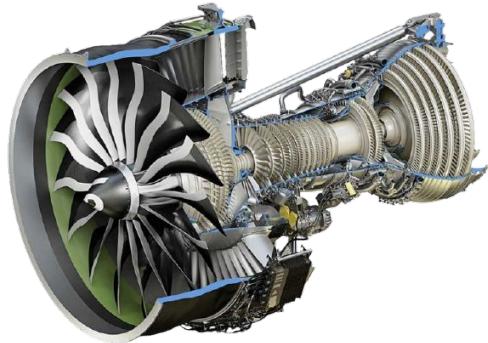
A multitude of small **sensors** and **actuators** are used in aviation, in order to detect:

- pressure
- force
- position
- temperature
- vibration



INTRODUCTION

These sensors are applied in different parts of the aircraft, offering a **constant data provisioning** that aims to improve **efficiency**, **safety**, **control** and **costs**!



INTRODUCTION

The critical aspect is that these flight data can be analyzed **in real-time**, the engineers and **ground-crews** can diagnose any malfunction and arrange maintenance in order to **minimize downtime!**

© BRANISLAVMILIC.COM/PHOTOS



INTRODUCTION

The critical aspect is that these flight data can be analyzed in **real-time**, the engineers and **ground-crews** can diagnose any malfunction and arrange maintenance in order to **minimize downtime!**

less Downtime



INTRODUCTION

The critical aspect is that these flight data can be analyzed in **real-time**, the engineers and **ground-crews** can diagnose any malfunction and arrange maintenance in order to **minimize downtime!**

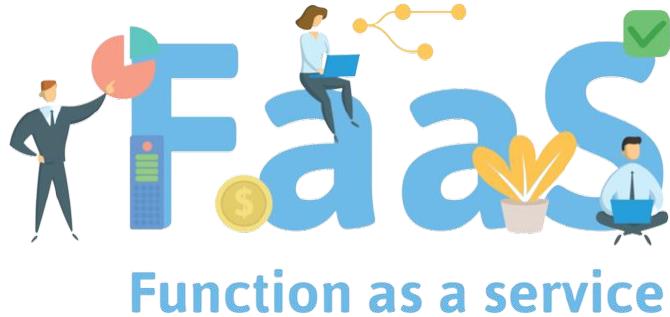
less Downtime



more



© BRANISLAVMILIC.COM/PHOTOS



It consists in a computing architecture based on open-source software that exploits the **Function as a Service** model in the context of IoT.

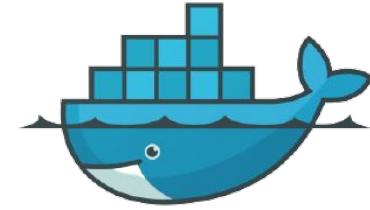
This project focuses on the processing of these data on **ground-control**.



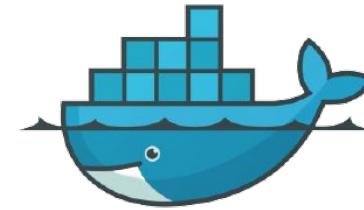
Tools



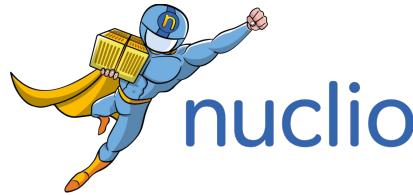
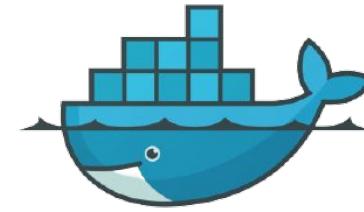
- Application containers engine **Docker** and **Docker Compose**.



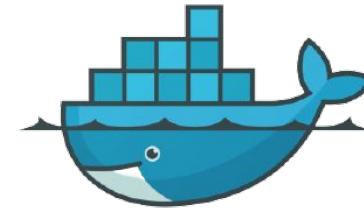
- Application containers engine **Docker** and **Docker Compose**.
- Serverless computing provider **Nuclo**.



- Application containers engine **Docker** and **Docker Compose**.
- Serverless computing provider **Nuclo**.
- MQTT and AMQP message broker **RabbitMQ**.



- Application containers engine **Docker** and **Docker Compose**.



- Serverless computing provider **Nuclio**.

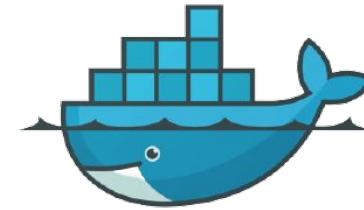
- MQTT and AMQP message broker **RabbitMQ**.



- JavaScript Application runtime **Node.js**



- Application containers engine **Docker** and **Docker Compose**.



- Serverless computing provider **Nuclio**.

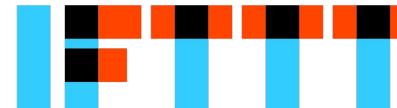
- MQTT and AMQP message broker **RabbitMQ**.



- JavaScript Application runtime **Node.js**



- Applets creation and management service **IFTTT**.



A photograph showing the rear view of several Emirates Airline aircraft tails parked at a gate. The aircraft have the characteristic red, white, and green stripes on their tails. In the background, a tall, modern control tower stands against a clear sky. A large teal rectangular box is overlaid on the right side of the image, containing the word "Scenario" in white capital letters.

Scenario

SCENARIO

The system simulates the **approach and landing** of airplanes at the airport.



SCENARIO

The system simulates the **approach and landing** of airplanes at the airport.



- Each airplane has sensors that produce information regarding the **fuel level**, the **condition of tires** and various **malfunctions**.



The system simulates the **approach and landing** of airplanes at the airport.

- Each airplane has sensors that produce information regarding the **fuel level**, the **condition of tires** and various **malfunctions**.
- An approaching airplane sends these data using the MQTT protocol, which are processed by different functions in a **serverless** way, and subsequently logged and stored.

SCENARIO

In this way the Ground Crew can prepare the exact amount of the necessary fuel, it already knows what are the tires that need maintenance and any damage that the aircraft suffered.



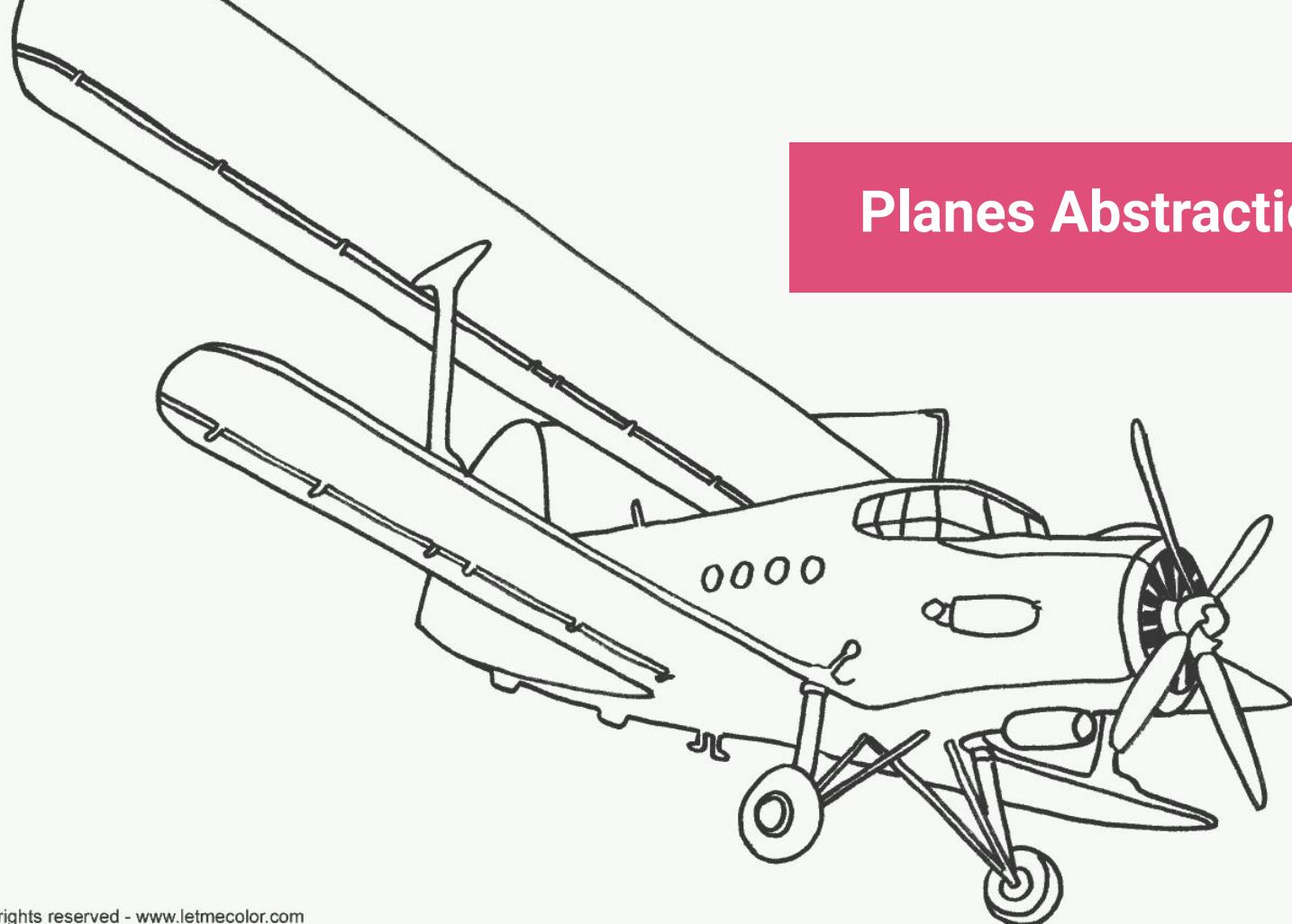
SCENARIO

In this way the Ground Crew can prepare the exact amount of the necessary fuel, it already knows what are the tires that need maintenance and any damage that the aircraft suffered.

Through the use of these systems it has been shown that the **downtime is reduced by around 30-40%**.



Planes Abstraction



PLANES ABSTRACTION

To abstract and manage airplanes I created a JavaScript class called **planes-generator**.

To abstract and manage airplanes I created a JavaScript class called **planes-generator**.

Each instance of airplane has:

- **model:** randomly chosen from a set of models, precisely among the 7 most produced model in the world!



Boeing-747



AirBus-A320



Boeing-777



Douglas-DC-9



AirBus-A330



AirBus-A300



Boeing-757

To abstract and manage airplanes I created a JavaScript class called **planes-generator**.

Each instance of airplane has:

- **model**: randomly chosen from a set of models, precisely among the 7 most produced model in the world!
- **name**: alphanumeric string made up of 5 randomly chosen characters.

To abstract and manage airplanes I created a JavaScript class called **planes-generator**.

Each instance of airplane has:

- **model**: randomly chosen from a set of models, precisely among the 7 most produced model in the world!
- **name**: alphanumeric string made up of 5 randomly chosen characters.
- **capacity**: the exact tank capacity of the selected aircraft model.

To abstract and manage airplanes I created a JavaScript class called **planes-generator**.

Each instance of airplane has:

- **model**: randomly chosen from a set of models, precisely among the 7 most produced model in the world!
- **name**: alphanumeric string made up of 5 randomly chosen characters.
- **capacity**: the exact tank capacity of the selected aircraft model.
- **fuelLevel**: current fuel percentage.

To abstract and manage airplanes I created a JavaScript class called **planes-generator**.

Each instance of airplane has:

- **model**: randomly chosen from a set of models, precisely among the 7 most produced model in the world!
- **name**: alphanumeric string made up of 5 randomly chosen characters.
- **capacity**: the exact tank capacity of the selected aircraft model.
- **fuelLevel**: current fuel percentage.
- **tag**: indicates if the airplane is arriving or has landed.

To abstract and manage airplanes I created a JavaScript class called **planes-generator**.

Each instance of airplane has:

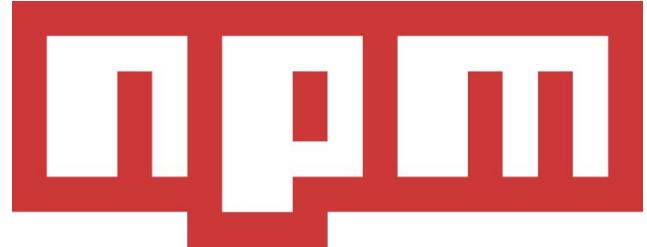
- **model**: randomly chosen from a set of models, precisely among the 7 most produced model in the world!
- **name**: alphanumeric string made up of 5 randomly chosen characters.
- **capacity**: the exact tank capacity of the selected aircraft model.
- **fuelLevel**: current fuel percentage.
- **tag**: indicates if the airplane is arriving or has landed.
- **undercarriages**: exact number for model selected.

To abstract and manage airplanes I created a JavaScript class called **planes-generator**.

Each instance of airplane has:

- **model**: randomly chosen from a set of models, precisely among the 7 most produced model in the world!
- **name**: alphanumeric string made up of 5 randomly chosen characters.
- **capacity**: the exact tank capacity of the selected aircraft model.
- **fuelLevel**: current fuel percentage.
- **tag**: indicates if the airplane is arriving or has landed.
- **undercarriages**: exact number for model selected.
- **tires**: exact number for model selected.

To make this class available across all Nuclio functions,
I published it on my public **NPM** profile.



To make this class available across all Nuclio functions,
I published it on my public **NPM** profile.

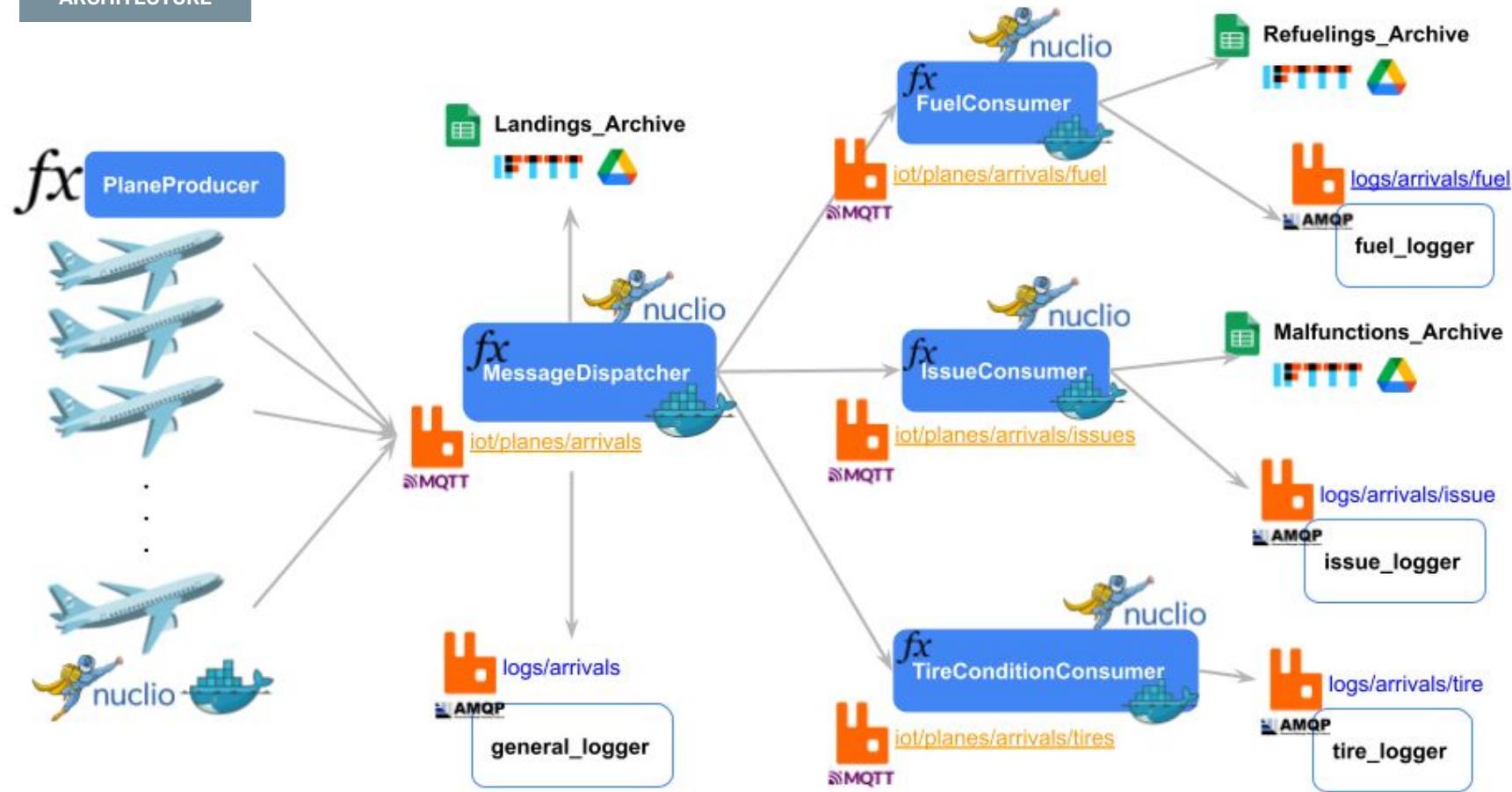


In this way anyone who uses Node.js can easily exploit the code by
simply installing it by writing:

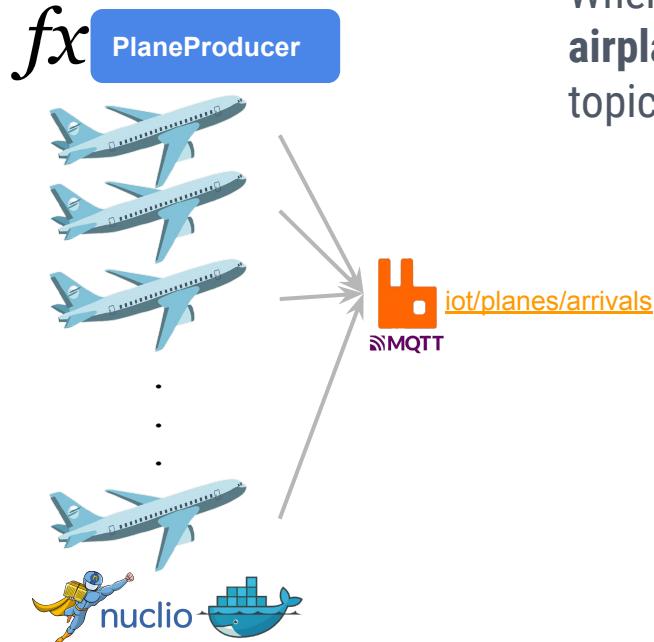
`npm i planes-generator`

Here the NPM page: [planes-generator](#)

ARCHITECTURE

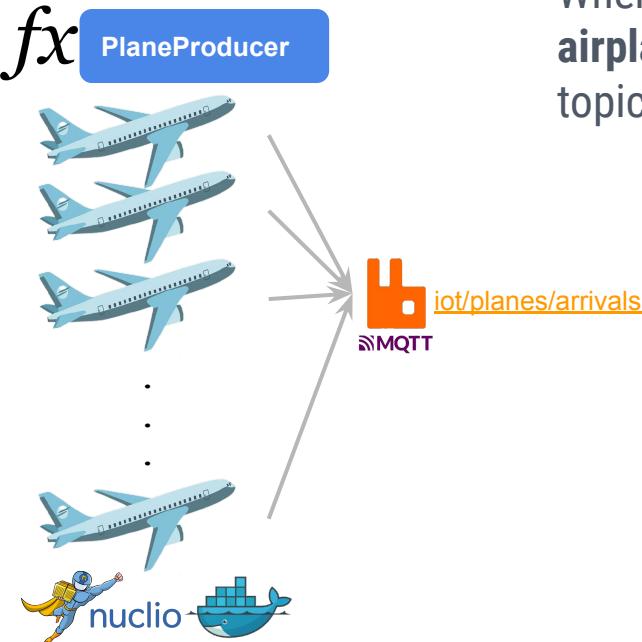


Plane Producer



Whenever this function is invoked, it creates a **new airplane instance** and then sends the data on the MQTT topic **iot/planes/arrivals**.

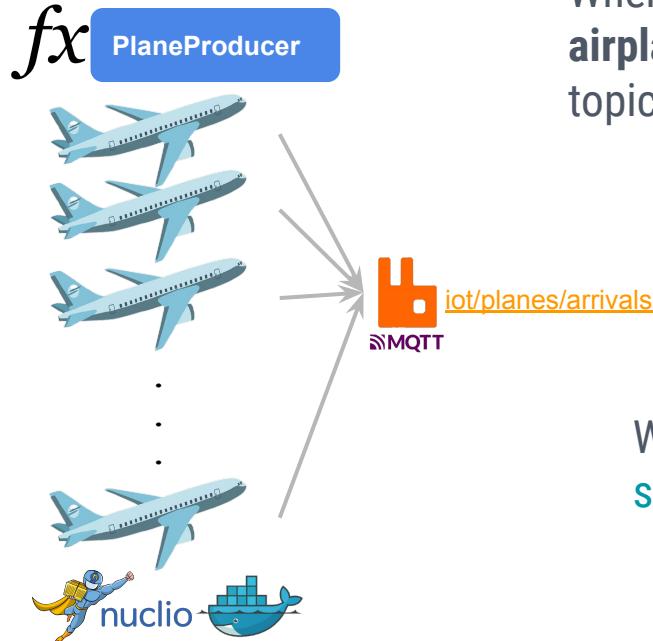
Plane Producer



Whenever this function is invoked, it creates a **new airplane instance** and then sends the data on the MQTT topic **iot/planes/arrivals**.

To simulate the **approach** and then **landing**, the data are sent every 5 seconds for a minute with a decreasing fuel level that indicates the fuel consumption.

Plane Producer

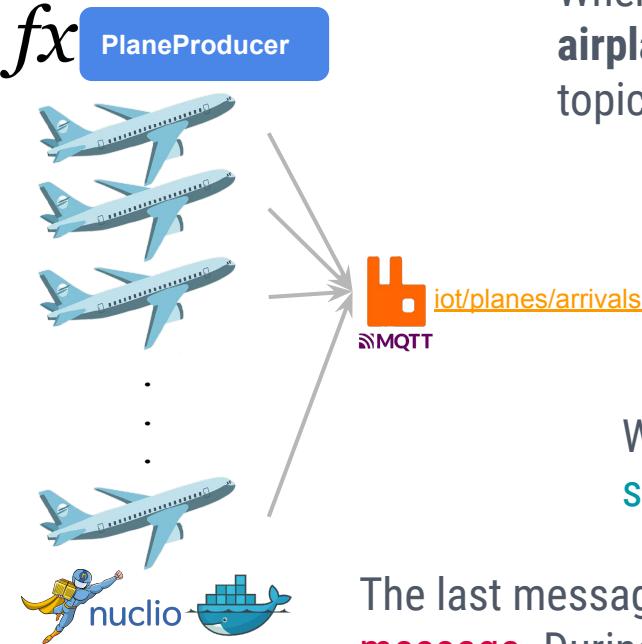


Whenever this function is invoked, it creates a **new airplane instance** and then sends the data on the MQTT topic **iot/planes/arrivals**.

To simulate the **approach** and then **landing**, the data are sent every 5 seconds for a minute with a decreasing fuel level that indicates the fuel consumption.

We can iterate these sendings exploiting **setTimeout()**, **setInterval()** and **clearInterval()** functions.

Plane Producer



Whenever this function is invoked, it creates a **new airplane instance** and then sends the data on the MQTT topic **iot/planes/arrivals**.

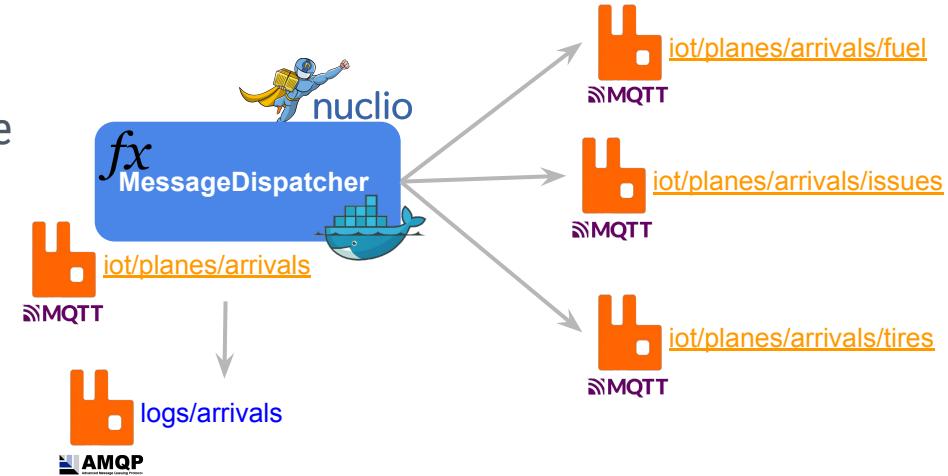
To simulate the **approach** and then **landing**, the data are sent every 5 seconds for a minute with a decreasing fuel level that indicates the fuel consumption.

We can iterate these sendings exploiting **setTimeout()**, **setInterval()** and **clearInterval()** functions.

The last message published for each execution is the **landing message**. During the landing procedure, tire issues and other casual issues are generated.

Message Dispatcher

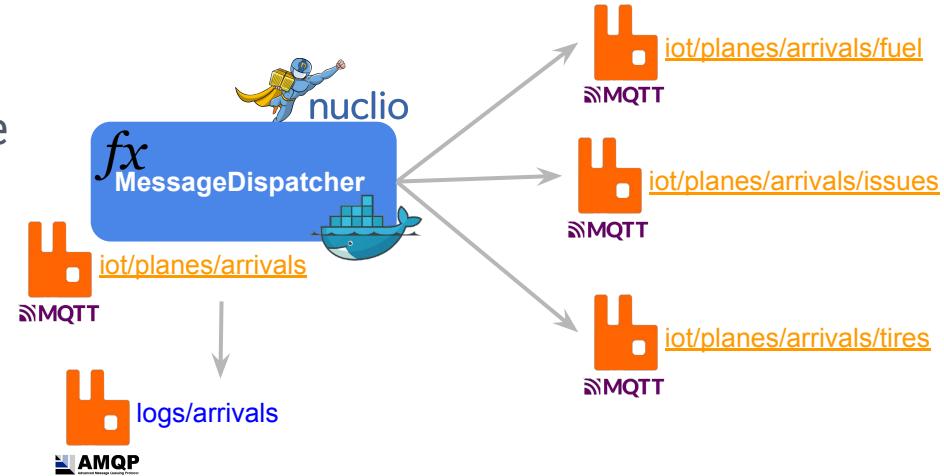
This function is triggered by a new MQTT message on the topic **iot/planes/arrivals** containing the plane information produced by Plane Producer.



Message Dispatcher

This function is triggered by a new MQTT message on the topic **iot/planes/arrivals** containing the plane information produced by Plane Producer.

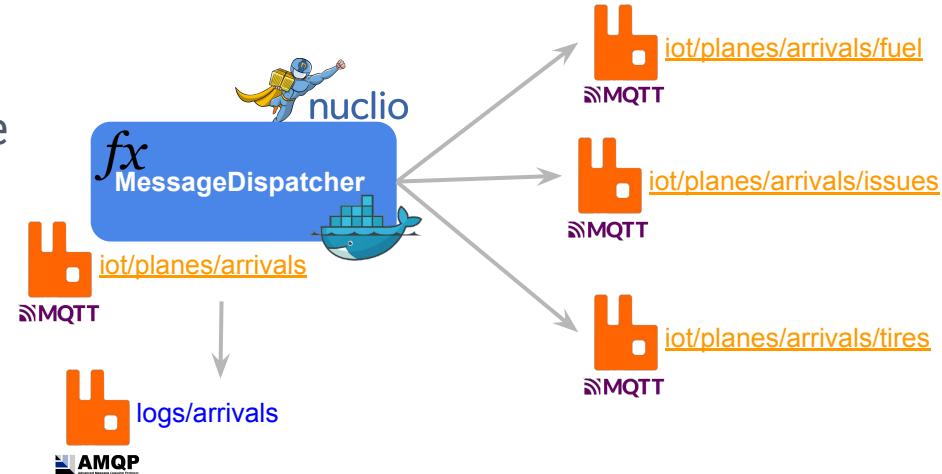
For each message received sends a feedback on the AMQP channel **logs/arrivals** with the plane name, model and current fuel_level.



Message Dispatcher

This function is triggered by a new MQTT message on the topic **iot/planes/arrivals** containing the plane information produced by Plane Producer.

For each message received sends a feedback on the AMQP channel **logs/arrivals** with the plane name, model and current fuel_level.

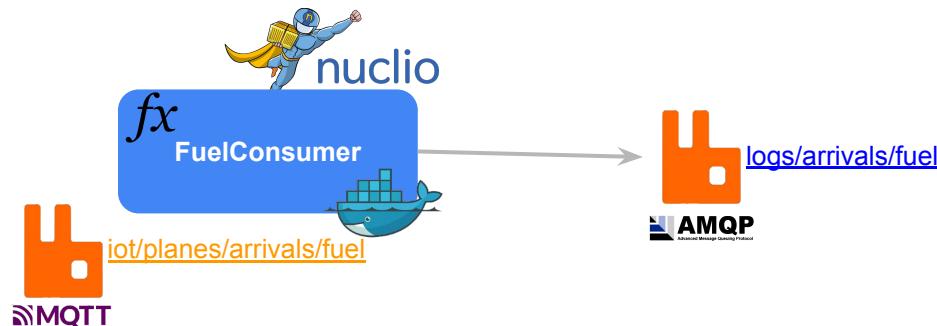


Only when the plane **lands** the Message Dispatcher publishes his data on 3 different MQTT topics, a sort of **forwarding**:

- **iot/planes/arrivals/fuel**
- **iot/planes/arrivals/issues**
- **iot/planes/arrivals/tires**

Fuel Consumer

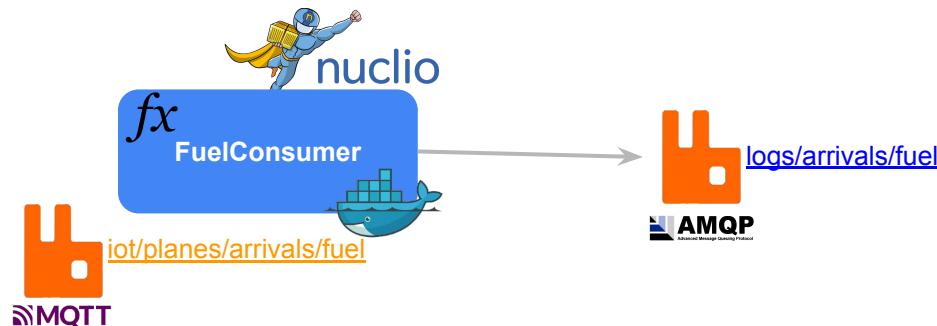
This function is triggered by a new MQTT message on the topic **iot/planes/arrivals/fuel** containing the plane information forwarded by the Message Dispatcher function.



Fuel Consumer

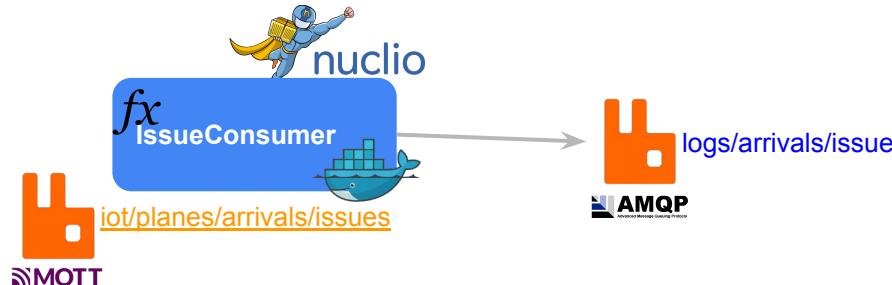
This function is triggered by a new MQTT message on the topic **iot/planes/arrivals/fuel** containing the plane information forwarded by the Message Dispatcher function.

The Fuel Consumer deals with **calculating the exact amount of fuel needed to refuel the aircraft**, then publish this information on the AMQ channel **logs/arrivals/fuel**.



Issue Consumer

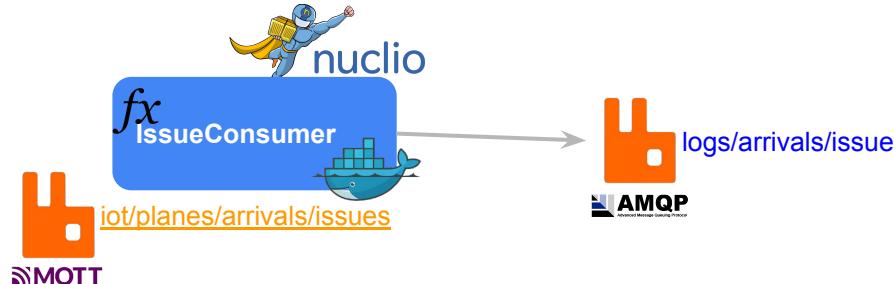
This function is triggered by a new MQTT message on the topic **iot/planes/arrivals/issues** containing the plane information forwarded by the Message Dispatcher function.



Issue Consumer

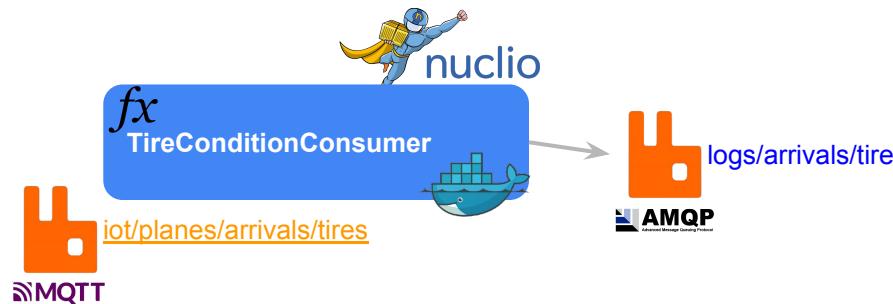
This function is triggered by a new MQTT message on the topic **iot/planes/arrivals/issues** containing the plane information forwarded by the Message Dispatcher function.

The Issue Consumer deals with updating the AMQP channel **logs/arrivals/issue** with information concerning the possible **malfunctions** detected during the previous flight.



Tire Condition Consumer

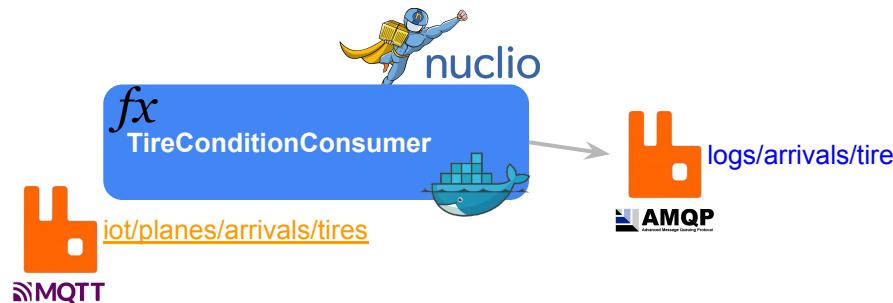
This function is triggered by a new MQTT message on the topic **iot/planes/arrivals/tires** containing the plane information forwarded by the Message Dispatcher function.



Tire Condition Consumer

This function is triggered by a new MQTT message on the topic **iot/planes/arrivals/tires** containing the plane information forwarded by the Message Dispatcher function.

The Tire Condition Consumer deals with updating the AMQP channel **logs/arrivals/tires** with information concerning **which tires** and **which undercarriage** need immediate maintenance

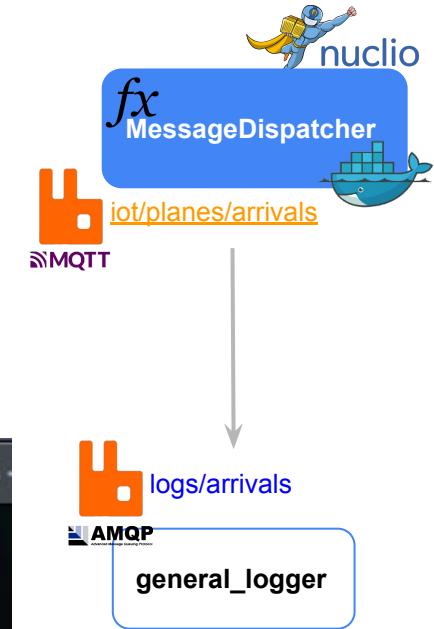


Loggers

4 loggers, each associated with a consumer function and listening on a AMQP dedicated channel.

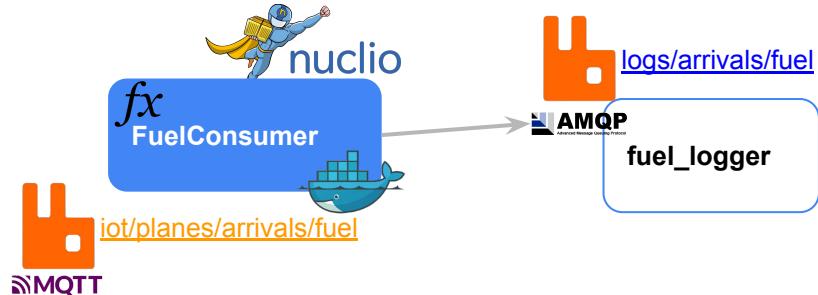
General_logger listening on **logs/arrivals**

```
[x] Received 'INCOMING' -----> Boeing-757 Hh1lK 79 _LANDED_
[x] Received 'INCOMING' -----> Airbus-A330 6hcwE 71 _LANDED_
[x] Received 'INCOMING' -----> Boeing-777 ycEOU 32 ARRIVING
[x] Received 'INCOMING' -----> Boeing-777 ycEOU 31 ARRIVING
[x] Received 'INCOMING' -----> Boeing-777 ycEOU 30 ARRIVING
[x] Received 'INCOMING' -----> Boeing-777 ycEOU 29 ARRIVING
[x] Received 'INCOMING' -----> Boeing-777 ycEOU 28 ARRIVING
[x] Received 'INCOMING' -----> Boeing-777 ycEOU 27 ARRIVING
[x] Received 'INCOMING' -----> Boeing-777 ycEOU 26 ARRIVING
[x] Received 'INCOMING' -----> Boeing-777 ycEOU 25 ARRIVING
[x] Received 'INCOMING' -----> Boeing-777 ycEOU 24 ARRIVING
[x] Received 'INCOMING' -----> Boeing-777 ycEOU 23 ARRIVING
[x] Received 'INCOMING' -----> Boeing-777 ycEOU 22 ARRIVING
[x] Received 'INCOMING' -----> Boeing-777 ycEOU 21 ARRIVING
[x] Received 'INCOMING' -----> Boeing-777 ycEOU 21 _LANDED_
[x] Received 'INCOMING' -----> Airbus-A330 3ubwT 15 ARRIVING
```



Loggers

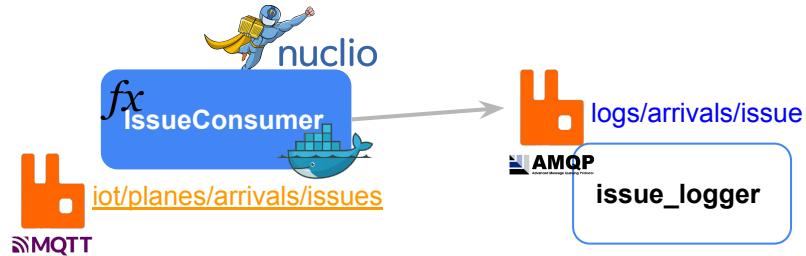
Fuel_logger listening on [logs/arrivals/fuel](#)



```
loggers — node fuel_logger.js — 134x35
[x] Received 'Airbus-A300 Wg3tJ is _LANDED_ with    40% of fuel. Fuel needing computed: 40896 liters!'
[x] Received 'Boeing-777 r6JKZ is _LANDED_ with    42% of fuel. Fuel needing computed: 105144.14 liters!'
[x] Received 'Boeing-757 sPM04 is _LANDED_ with    28% of fuel. Fuel needing computed: 30729.6 liters!'
[x] Received 'Boeing-747 wVB49 is _LANDED_ with    81% of fuel. Fuel needing computed: 41196.56 liters!'
[x] Received 'Airbus-A320 QqqG6 is _LANDED_ with    11% of fuel. Fuel needing computed: 21234.510000000002 liters!'
[x] Received 'Airbus-A330 ukQE9 is _LANDED_ with    87% of fuel. Fuel needing computed: 18081.7 liters!'
[x] Received 'Airbus-A300 1xF0d is _LANDED_ with    52% of fuel. Fuel needing computed: 32716.8 liters!'
[x] Received 'Douglas-DC-9 PHM3K is _LANDED_ with    45% of fuel. Fuel needing computed: 7659.3 liters!'
[x] Received 'Boeing-777 gA1WR is _LANDED_ with    15% of fuel. Fuel needing computed: 154090.55 liters!'
```

Loggers

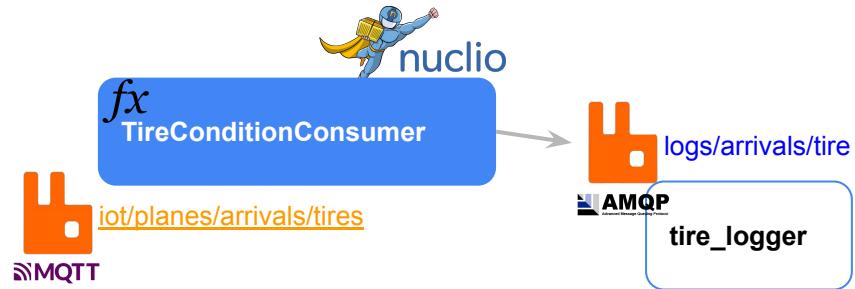
Issue_logger listening on **logs/arrivals/issues**



```
[x] Received 'Airbus-A300 Wg3tJ is _LANDED_! The sensors encountered a Possible left engine malfunction'  
[x] Received 'Boeing-777 r6JKZ is _LANDED_! The sensors encountered a Possible damage to the tail'  
[x] Received 'Boeing-757 sPM04 is _LANDED_! The sensors encountered a Possible depressurization of the hold'  
[x] Received 'Boeing-747 wVB49 is _LANDED_! The sensors encountered a Possible damage to the left flaps'  
[x] Received 'Airbus-A320 QqqG6 is _LANDED_! The sensors encountered a Possible refrigerant leak'  
[x] Received 'Airbus-A330 ukQE9 is _LANDED_! The sensors encountered a Possible damage to the fuselage'  
[x] Received 'Airbus-A300 1xF0d is _LANDED_! The sensors encountered a Possible damage to the left wing'  
[x] Received 'Douglas-DC-9 PHM3K is _LANDED_! The sensors encountered a Possible damage to the right flaps'  
[x] Received 'Boeing-777 gA1WR is _LANDED_! The sensors encountered a Possible damage to the right flaps'  
[x] Received 'Douglas-DC-9 mU7EM is _LANDED_! The sensors encountered a Possible damage to the right flaps'  
[x] Received 'Airbus-A300 7A1Sk is _LANDED_! The sensors encountered a Possible damage to the left flaps'  
[x] Received 'Airbus-A330 mhnoQ is _LANDED_! The sensors encountered a Possible depressurization of the hold'
```

Loggers

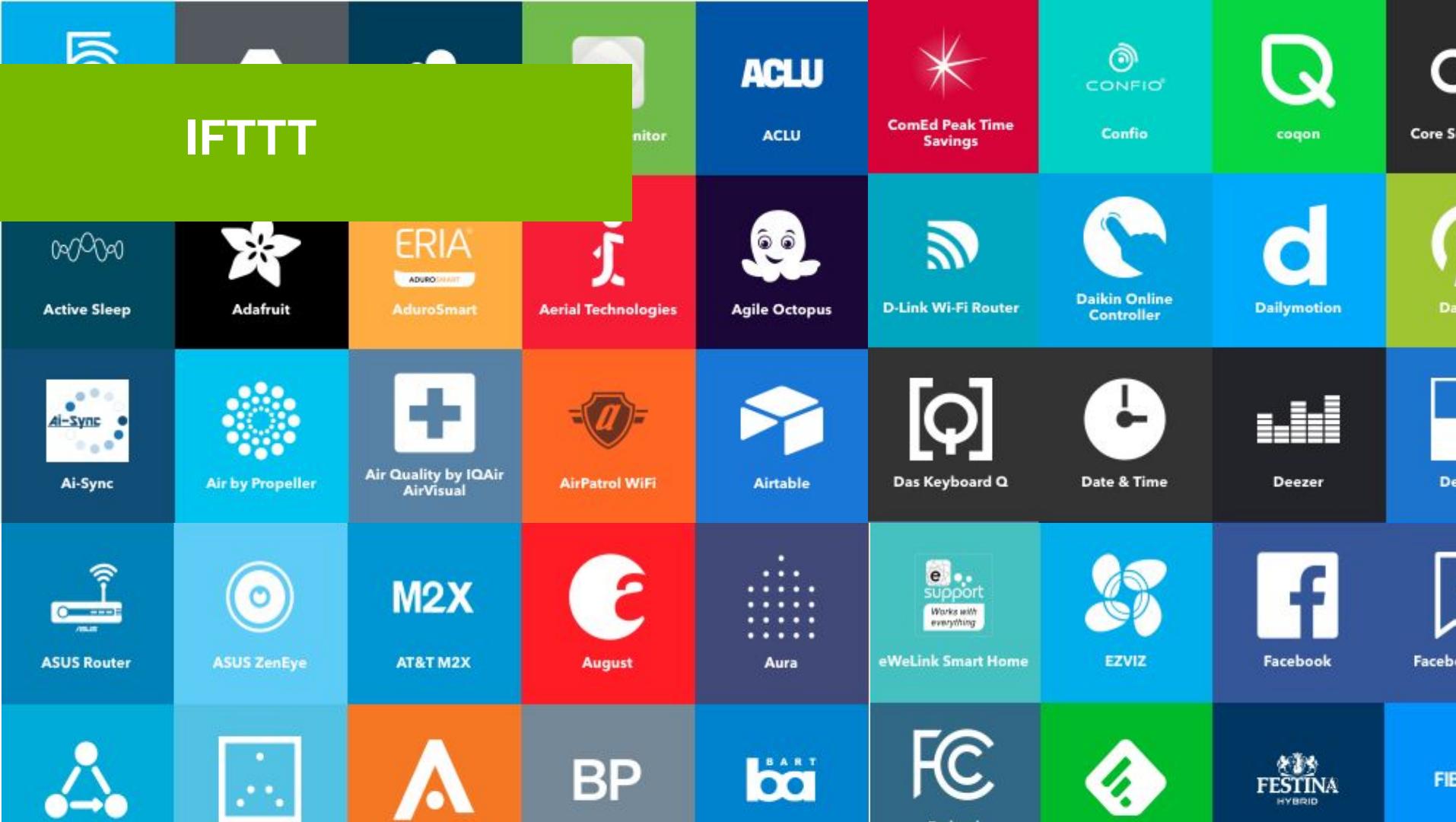
Tire_logger listening on **logs/arrivals/tires**



```

loggers — node tire_logger.js — 134x33
[x] Received 'Boeing-757 sPM04 is _LANDED_! Manteinance undercarriage n: 3! Pressure drop detected at tire n: 6 '
[x] Received 'Boeing-747 wVB49 is _LANDED_! Manteinance undercarriage n: 4! Pressure drop detected at tire n: 2 '
[x] Received 'Airbus-A320 QqqG6 is _LANDED_! Manteinace undercarriage n: 1! Pressure drop detected at tire n: 2 '
[x] Received 'Airbus-A330 ukQE9 is _LANDED_! Manteinace undercarriage n: 2! Pressure drop detected at tire n: 5 '
[x] Received 'Airbus-A300 1xF0d is _LANDED_! Manteinace undercarriage n: 3! Pressure drop detected at tire n: 9 '
[x] Received 'Douglas-DC-9 PHM3K is _LANDED_! Manteinace undercarriage n: 1! Pressure drop detected at tire n: 3 '
[x] Received 'Boeing-777 gA1WR is _LANDED_! Manteinace undercarriage n: 2! Pressure drop detected at tire n: 8 '
[x] Received 'Douglas-DC-9 mU7EM is _LANDED_! Manteinace undercarriage n: 3! Pressure drop detected at tire n: 3 '
[x] Received 'Airbus-A300 7ALSK is _LANDED_! Manteinace undercarriage n: 3! Pressure drop detected at tire n: 1 '
[x] Received 'Airbus-A330 mhnoQ is _LANDED_! Manteinace undercarriage n: 1! Pressure drop detected at tire n: 2 '
[x] Received 'Airbus-A300 PNkOP is _LANDED_! Manteinace undercarriage n: 1! Pressure drop detected at tire n: 1 '
[x] Received 'Airbus-A320 4aseA is _LANDED_! Manteinace undercarriage n: 1! Pressure drop detected at tire n: 3 '
[x] Received 'Douglas-DC-9 aTRWP is _LANDED_! Manteinace undercarriage n: 3! Pressure drop detected at tire n: 2 '
[x] Received 'Airbus-A330 kUuzO is _LANDED_! Manteinace undercarriage n: 3! Pressure drop detected at tire n: 3 '

```



IFTTT



If This Then That is a service that allows a user to program a **response** to events of various kinds. The programs, called **Applets** are simple and created graphically.



If This Then That is a service that allows a user to program a **response** to events of various kinds. The programs, called **Applets** are simple and created graphically.

It employs the following concepts:

- **Services:** are the basic building blocks of IFTTT. They describe a series of data from a certain **web service**. Each service has a particular set of **triggers** and actions.



If This Then That is a service that allows a user to program a **response** to events of various kinds. The programs, called **Applets** are simple and created graphically.

It employs the following concepts:

- **Services:** are the basic building blocks of IFTTT. They describe a series of data from a certain **web service**. Each service has a particular set of **triggers** and actions.
- **Triggers:** are the “this” part of an applet. They are the items that trigger the action.



If This Then That is a service that allows a user to program a **response** to events of various kinds. The programs, called **Applets** are simple and created graphically.

It employs the following concepts:

- **Services:** are the basic building blocks of IFTTT. They describe a series of data from a certain **web service**. Each service has a particular set of **triggers** and actions.
- **Triggers:** are the “**this**” part of an applet. They are the items that trigger the action.
- **Actions:** are the “**that**” part of an applet. They are the output that results from the input of the trigger.

For the Plane-System I created **three** applets using the following services offered by IFTTT:



Webhooks

Receive a web request

This trigger fires every time the Maker service receives a web request to notify it of an event.

For the Plane-System I created **three** applets using the following services offered by IFTTT:



Webhooks

Receive a web request

This trigger fires every time the Maker service receives a web request to notify it of an event.



Google Sheets

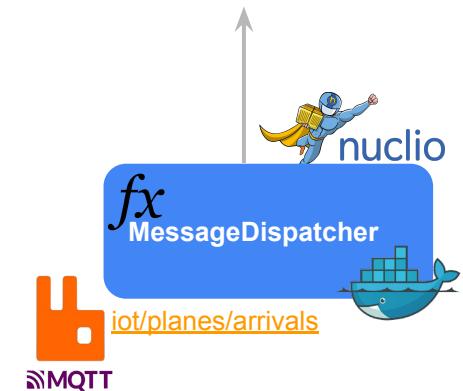
Add row to spreadsheet

This action will add a single row to the bottom of the first worksheet of a spreadsheet you specify. Note: a new spreadsheet is created after 2000 rows.

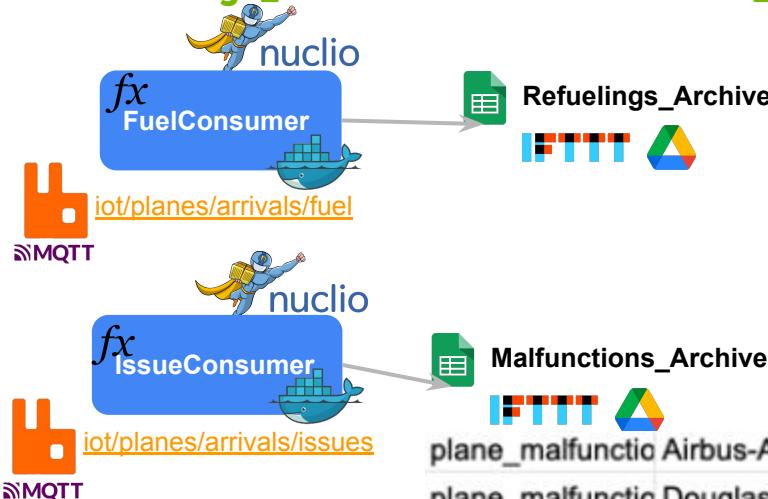
The first applet is called “**plane_landed**”.

It is connected to Message Dispatcher, every time a plane lands, the applet is invoked and updates an excel file on my **Google Drive** which acts as a **Landing_Archive**.

46	plane_landed	Airbus-A320 - 64wX9	1/2/2021	10:59
47	plane_landed	Douglas-DC-9 - c8Dgx	1/2/2021	11:0
48	plane_landed	Boeing-757 - O6uVO	1/2/2021	11:0
49	plane_landed	Airbus-A320 - kg0xl	1/2/2021	11:0
50	plane_landed	Douglas-DC-9 - pKWX7	1/2/2021	11:0
51	plane_landed	Airbus-A320 - JUpF6	8/3/2021	12:27
52	plane_landed	Airbus-A300 - 8n6lo	8/3/2021	12:27
53	plane_landed	Boeing-747 - 4yp3L	8/3/2021	12:27
54	plane_landed	Airbus-A330 - X5Gvu	8/3/2021	12:27



In the same way the other two applets (**plane_refueling** and **plane_malfunction**) are connected one with Fuel Consumer function, and the other with Issue Consumer function, in order to create **Refuelings_Archive** and a **Malfunction_Archive**.



plane_refueling	Airbus-A330 - 6hcwE	2/2/2021 - 15:33	40336.1 liters
plane_refueling	Boeing-777 - ycE0U	2/2/2021 - 15:34	143213.57 liters
plane_refueling	Airbus-A330 - 3Uhwt	2/2/2021 - 15:52	132135.5 liters
plane_refueling	Boeing-777 - 3BTB7	2/2/2021 - 15:52	106956.97 liters
plane_refueling	Airbus-A300 - ubnFM	2/2/2021 - 15:52	41577.6 liters
plane_refueling	Douglas-DC-9 - SHwNI	2/2/2021 - 15:52	6684.48 liters
plane_refueling	Airbus-A300 - 8n6lo	8/3/2021 - 12:27	28627.2 liters

plane_malfunction	Airbus-A320 - kgoxl	1/2/2021 - 11:0	Possible damage to the tail
plane_malfunction	Douglas-DC-9 - pKWX7	1/2/2021 - 11:0	Possible damage to the right wing
plane_malfunction	Boeing-757 - O6uVO	1/2/2021 - 11:0	Possible damage to the fuselage
plane_malfunction	Airbus-A300 - 8n6lo	8/3/2021 - 12:27	Possible depressurization of the hold
plane_malfunction	Airbus-A320 - JUpF6	8/3/2021 - 12:27	Possible depressurization of the hold
plane_malfunction	Boeing-747 - 4yp3L	8/3/2021 - 12:27	Possible damage to the fuselage
plane_malfunction	Airbus-A330 - X5Gvu	8/3/2021 - 12:27	Possible damage to the left wing

To integrate these applet in my code I have exploited the **NPM** package **ifttt-webhooks-channel** (installable with npm i ifttt-webhooks-channel) that provide a simple **post** method to trigger the webhook as follow:

```
// invoke the webhook
ifttt.post('plane_landed', [currentPlane.model + " - " + currentPlane.name,
                           date.getDate() + "/" + (date.getMonth()+1) + "/" + date.getFullYear(),
                           (date.getHours()+1) + ":" + date.getMinutes()]);
```

How to Run



1. Install Docker and Docker Compose
2. Start Nuclio through Docker: `$ docker run -p 8070:8070 -v /var/run/docker.sock:/var/run/docker.sock -v /tmp:/tmp nuclio/dashboard:stable-amd64`
then browse to <http://localhost:8070> to use Nuclio Dashboard.
3. Start RabbitMQ through Docker: `$ docker run -p 9000:15672 -p 1883:1883 -p 5672:5672 cyriliq/rabbitmq-mqtt`
then browse to <http://localhost:9000> and login using username:guest and password:guest, to access to RabbitMQ management, where is possible to visualize the message queues and the broker status
4. Replace all IP addresses in the code with your own, also in the .yaml files and in logger functions.
Then updates the **functionSourceCode** field in all .yaml files with the base64 encoding of the functions.
5. Browse to Nuclio Dashboard and create a new project uploading producer and consumers functions, the deploy all.
6. Start all loggers with node logger_name.js in the terminal.
7. Test the **planeProducer** function and observe messages on loggers.
8. If you want, create your applets with IFTTT and update the fields in the code.



Thank you for listening