Preparation for the Workshop

https://github.com/alegeno92/docker-workshop/

- Go to “Requirements”
- Follow instructions in prep for the workshop:
  - Important: Need Xcode for Mac

arm

AIOT Dev Summit

Innovate. Learn. Experience.

Containers in an Iot World

How Docker Can Make It Easier to Deploy Your Application on an Edge Device / Maurizio Caporali, Alessandro Genovese and Jenny Fong

arm #ArmDevSummit

Speakers

Alessandro Genovese
IoT and UDOO
Software Engineer
SECO

Maurizio Caporali
IoT and UDOO Product
Manager
SECO

Jenny Fong
Sr. Director Product
Marketing
DOCKER

Introduction to Docker for Embedded Developers

Jenny Fong

Developers Love Docker!

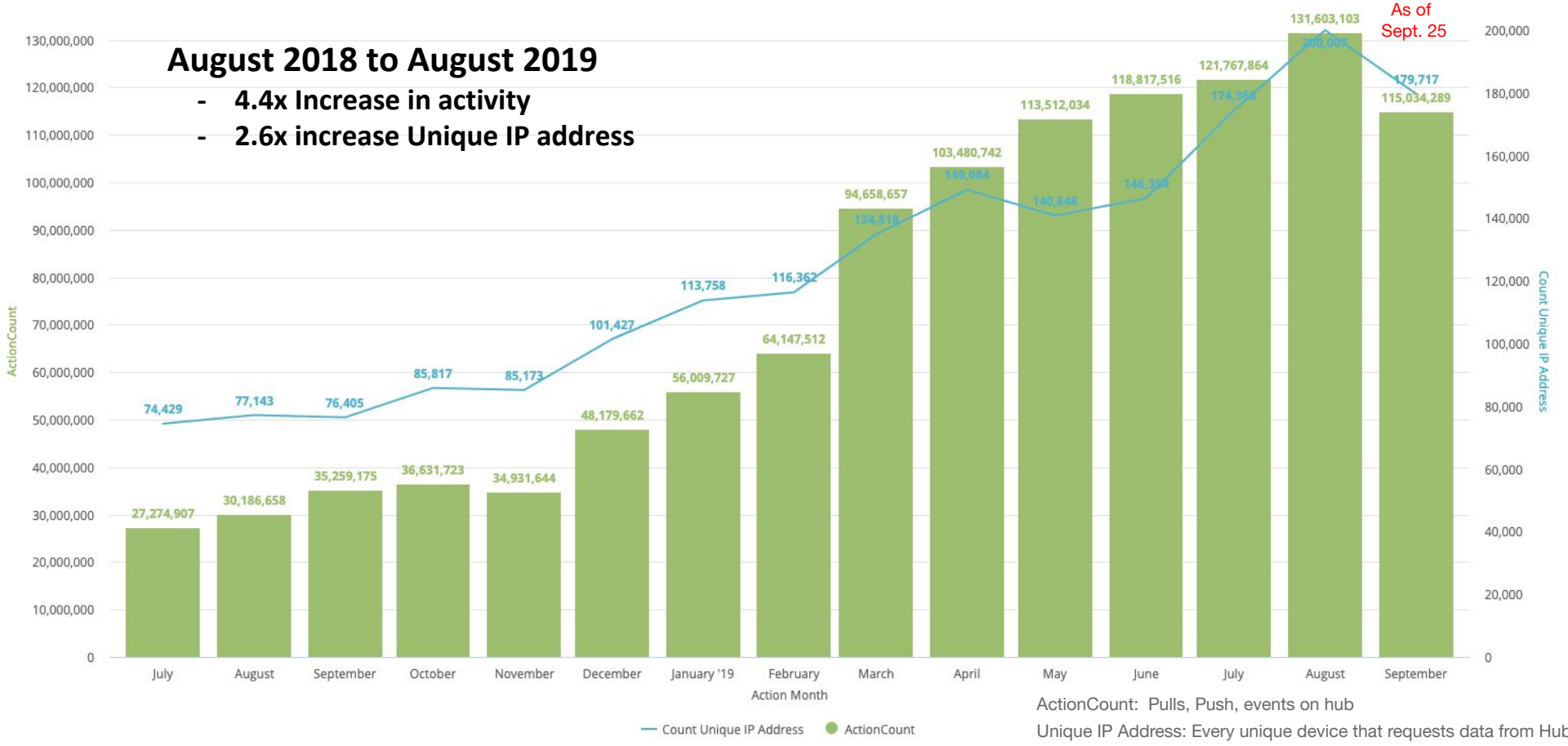
Most **Wanted**
Platform

Most **Loved**
Platform

Most **Used**
Platform

stack**overflow**

Docker Hub Activity and IP address Count - last 15 months



ActionCount: Pulls, Push, events on hub

Unique IP Address: Every unique device that requests data from Hub

Alternatives for embedded software development

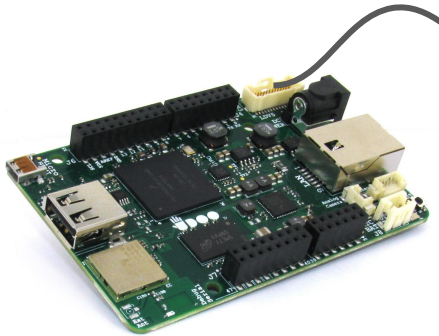
Docker improves on cross-compilation and target compilation

	Cross-compile	Target Build	Docker
Complexity	 Different environment leads to more porting errors	 Same environment	 Consistent environment
At scale deployment	 Medium, must reconfigure each target environment	 Slow	 Simple, must setup docker on each target
Build time	 Quickest, host PC speeds	 Very slow, must reconfigure environment and app on each device	 OK: Host PC with QEMU Good: Using remote Arm build farm

Docker's Edge/IoT Developer Platform

Embedded developers become cloud-native in minutes!

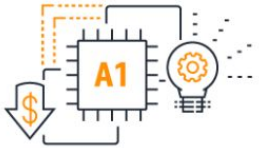
1. Embedded developer loop



2. Inner Loop - Emulation environment

Docker Desktop

3. Native build on remote server



4. CI/CD pipeline

Docker Trusted Registry

Docker Enterprise Platform for Secure Edge Development

Key Features

Application Templates

Start new projects quickly, with all the required Docker configuration files

- SecOps builds templates for your Dev teams
- GUI and CLI

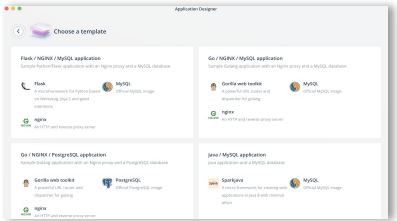
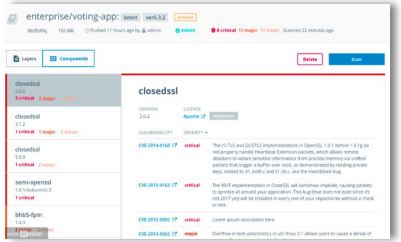


Image Scanning & Signing

Know what's inside your containers, track provenance

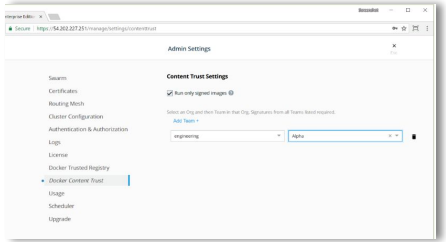
- Binary-level scans against NIST CVEs
- Add digital signatures to approve content



Secure Engine Runtime

Run only trusted content at the edge

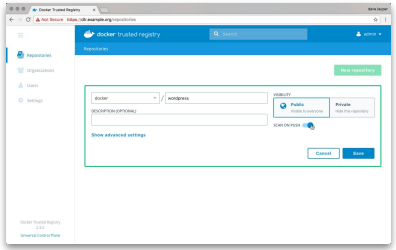
- Prevent unsigned images from being deployed
- Leverage hardware-based security



Automated Pipelines

Rapidly deliver new applications to devices

- Secure collaboration through Docker Trusted Registry and Docker Hub
- Webhooks integrate with your chosen IDE and CI tools



Introducing PARSEC

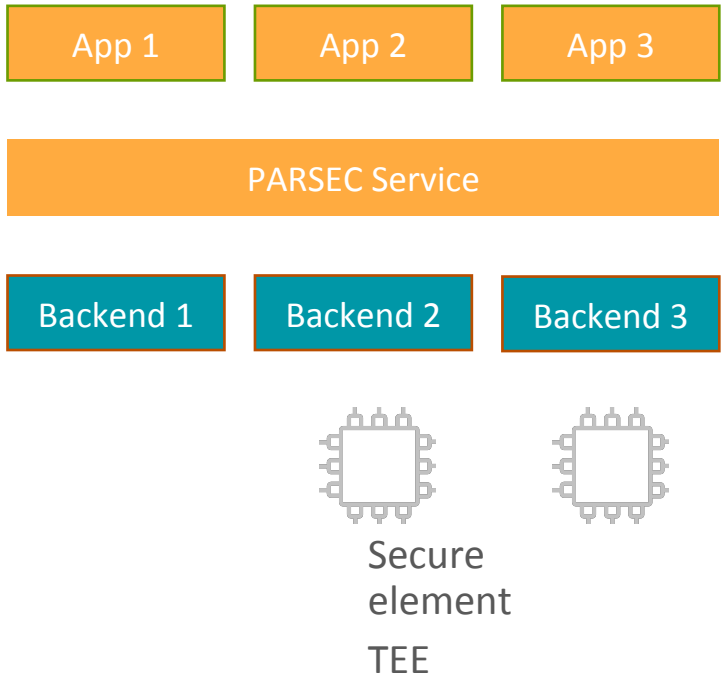
Platform AbstRaction for SECurity

Microservice abstracts hardware security

Surfaces a modern, simple, multi-language friendly, Open Source API that is common to IoT endpoints

Multiple back-ends: use the best capabilities of a given platform

Developing collaboratively and in the open



Problem statement and workshop flow – Docker/SECO

Maurizio Caporali & Jenny Fong

Problem Statement

Developing complex embedded applications for a PoC or a project in a few days.

Project

Our project is:

- Application: People Counter device to control the passage of people in a shopping center, with the possibility of accessing the real time counter and sending the data to the cloud.
- Hardware: Arm SBC based on NXP 1GHz ARM® Cortex-A9 with 1GB of RAM with Yocto OS.

What do we need?

Base Applications:
Python + Libraries
OpenCV
C++ Compilers
CMake
Paho MQTT Client Libraries
Redis (maybe)
Mosquitto MQTT Broker

Specific Applications:
People Counter
Cloud Connector
Telemetry Agent
Restful Server + Dashboard

Standard Approach

Activities and Timing estimation for the project using a standard approach:

- Compiling the Base Applications on the Arm SBC with Yocto OS - **TIME: 5 Days**
- Developing or porting the Specific Applications - **TIME: 2 Days**
- Write Yocto recipes - **TIME: 4 Days**
- Tests - **TIME: 1 Days**

The Workshop Flow

What we're going to do:

- Take a hands on approach with the UDOO hardware Kit
- Define the Docker Container structure for the project (Docker Approach)
- Startup the Arm board and test the Docker Engine runs
- Build Containers from scratch
- Connect many Docker containers together using Docker Compose tool
- Deploy on UDOO NEO (Linux Ubuntu)
- Deploy on an Industrial IoT Gateway with EDGEHOG OS (Yocto based): Seco SBC-C23, Seco SBC-C61

Intro to UDOO – SECO boards

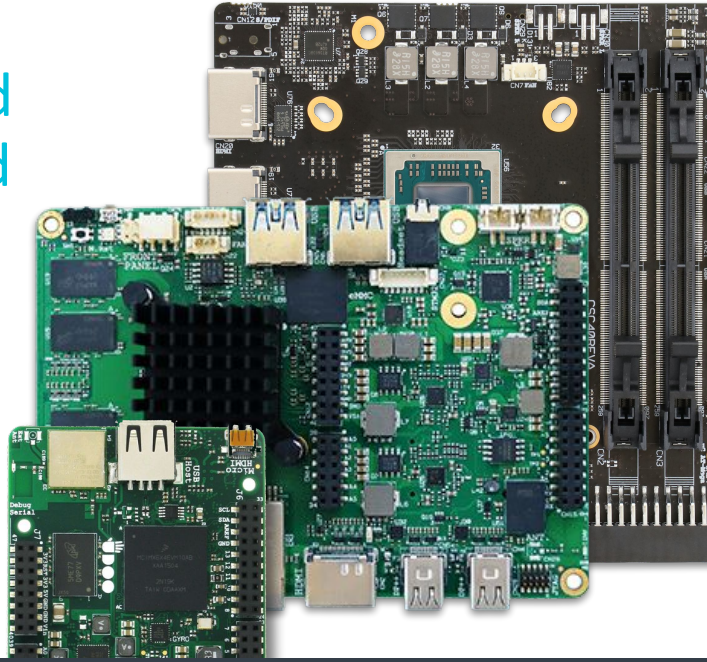
Maurizio Caporali

UDOO

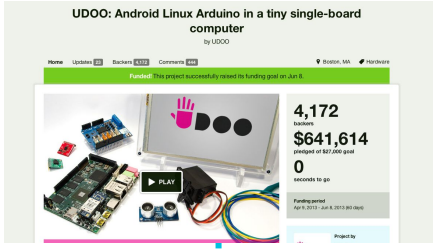
What is UDOO?

UDOO is a family of open source embedded boards that merges physical computing and programming.

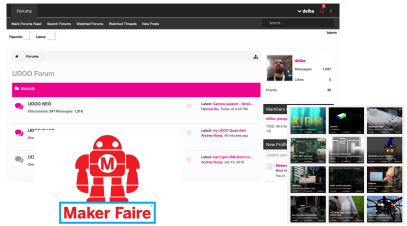
- UDOO is a co-founded project by SECO, embedded manufacturer, and AIDILAB, interaction design firm.
- Four Kickstarter Campaigns raising 2.3M+ with 13k+ backers, and 150k+ boards sold from 2014



1st Board Launched on Kickstarter:
UDOO QUAD/DUAL



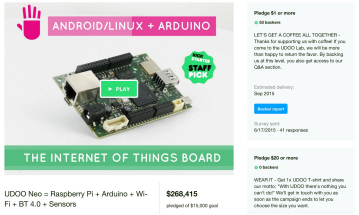
The start of the hottest events in tech



2013/2014

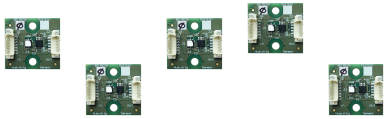
Raised $1 Million on Kickstarter

2nd Board Launched on Kickstarter:
UDOO NEO



2015

Further product range expansion: **BRICKS**



3rd Board launched on Kickstarter:
UDOO X86

4,245 backers pledged $800,211 to help bring this project to life.

2016

4th Board launched on Kickstarter:
UDOO BOLT

1,447 backers pledged $635,769 to help bring this project to life.

2018

UDOO

The UDOO Community

150k+ UDOO Boards sold all around the world.

- UDOO Boards are considered one of the best professional hacking SBC.
- Google, Amazon, IBM, engineers use the UDOO boards.

Updated daily
37k+ likes

Updated daily
1800+ followers

Updated daily
4.7k+ followers

Updated daily
500+ members

1.6M+ visualizations
7.3k+ registered users

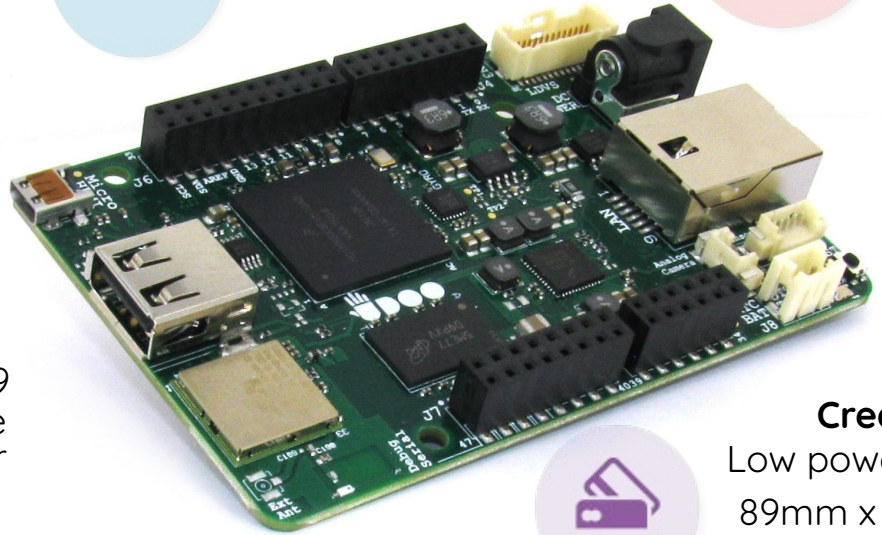
28k+ total posts
12.2k+ members

UDOO

The UDOO NEO

Born to be wireless
Wi-Fi 802.11 b/g/n
Bluetooth 4.0 BLE

9 Axis sensors
Accelerometer
Gyroscope
Magnetometer

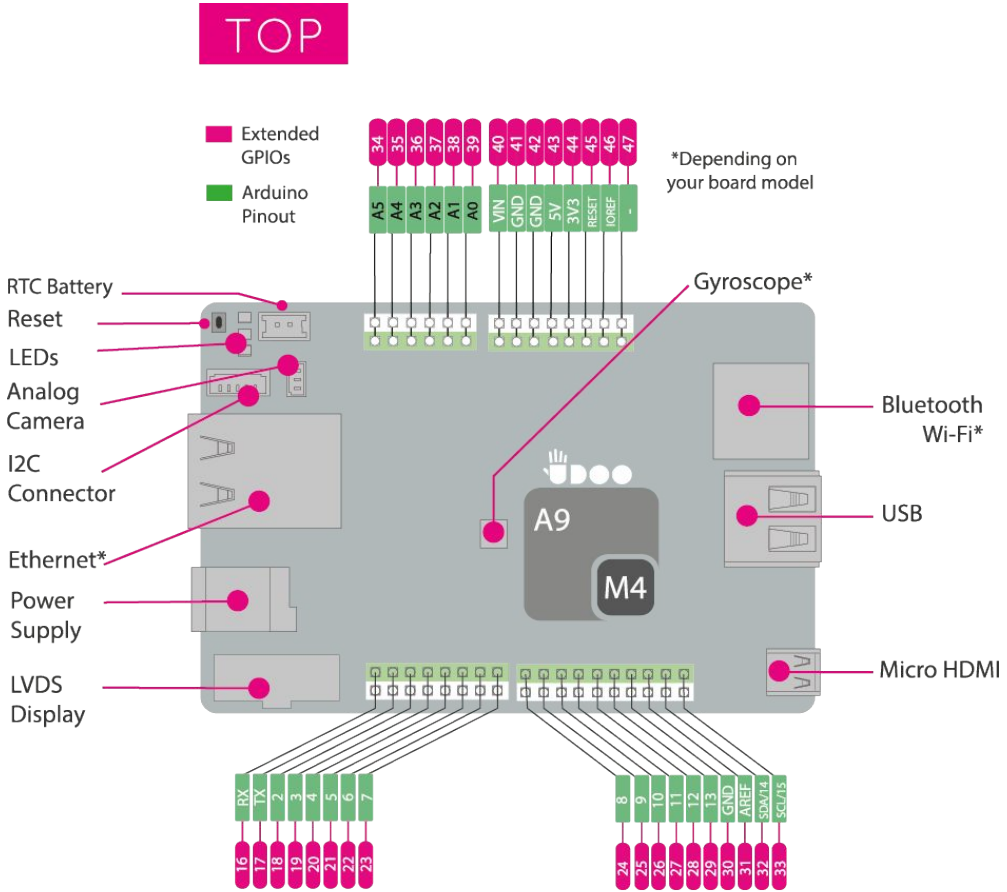


Powerful 1GHz Cortex-A9
+M4 I/O realtime
co-processor

Credit-card sized
Low power consumption board
89mm x 59mm (3.50” x 2.32”)

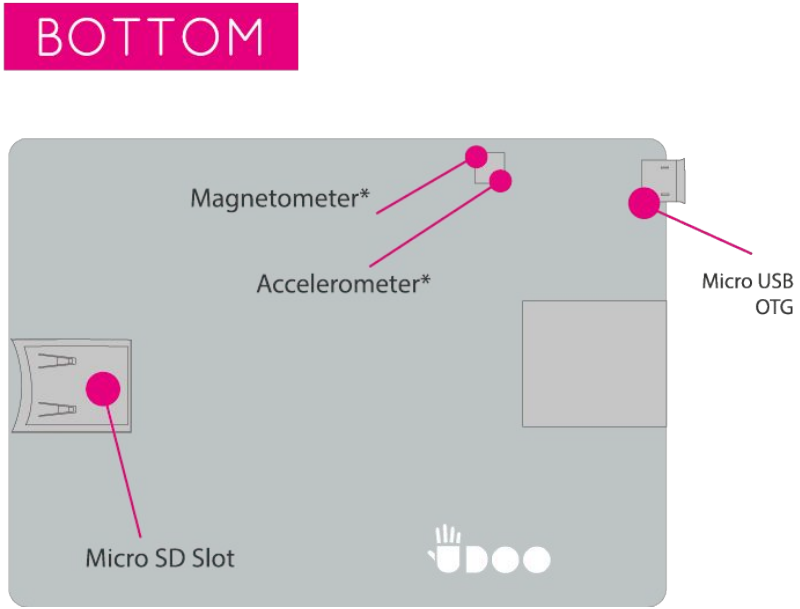
UDOO

The UDOO NEO



UDOO

The UDOO NEO



Hands-on – SECO

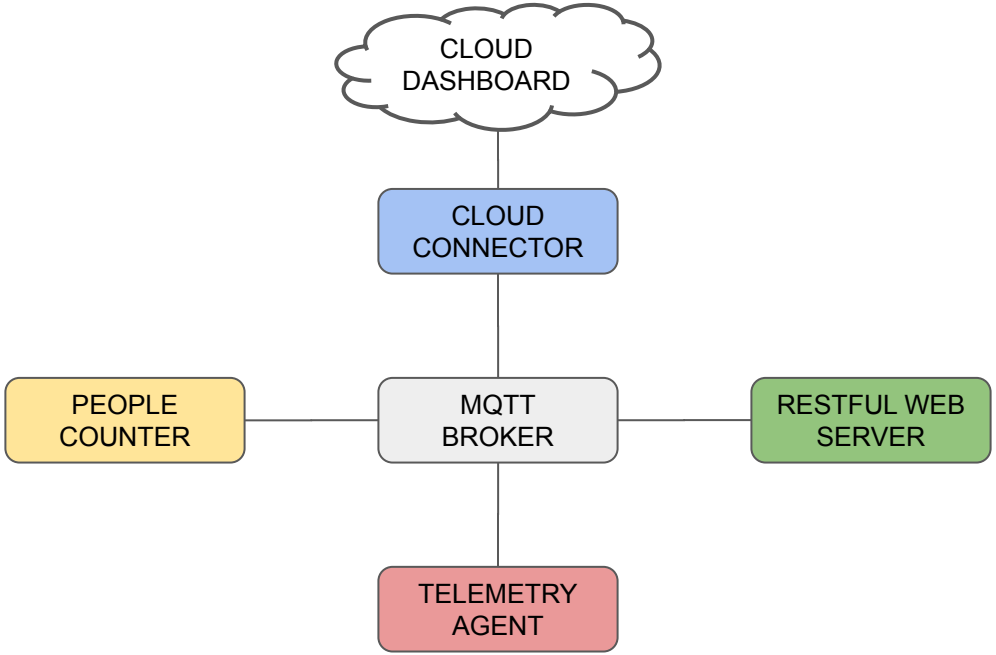
Alessandro Genovese

Workshop Flow

What are we going to do:

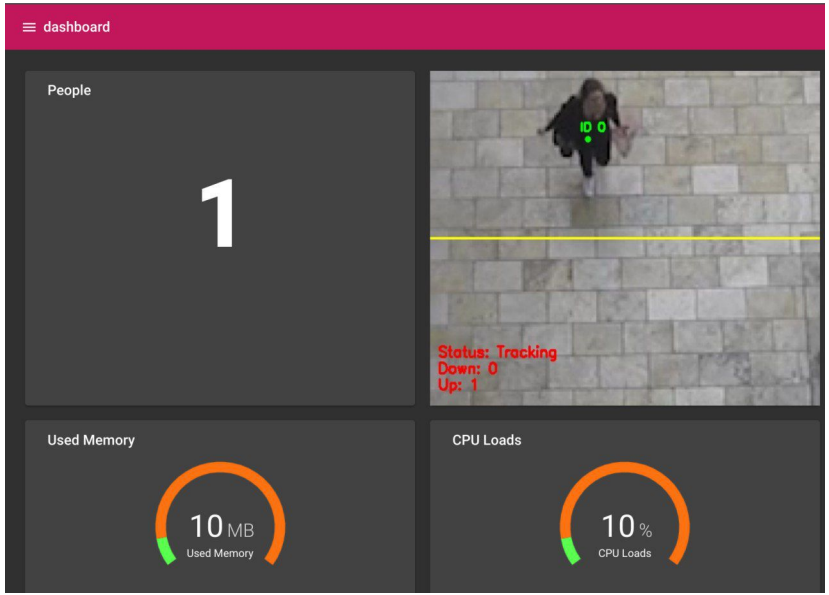
- **Define the Docker Container structure for the project (Docker Approach)**
- Test Docker Engine runs on the Laptop
- Build Python Container from scratch
- Build C++ Container from scratch
- Use Buildx to create containers for different platform and Push them on DockerHub
- Startup the Arm board and test the Docker Engine runs
- Connect many Docker containers together using Docker Compose
- Deploy on UDOO NEO (Linux Ubuntu)
- Deploy on an Industrial IoT Gateway with EDGEHOG OS (Yocto based): Seco SBC-C23, Seco SBC-C61

The Docker Container Approach



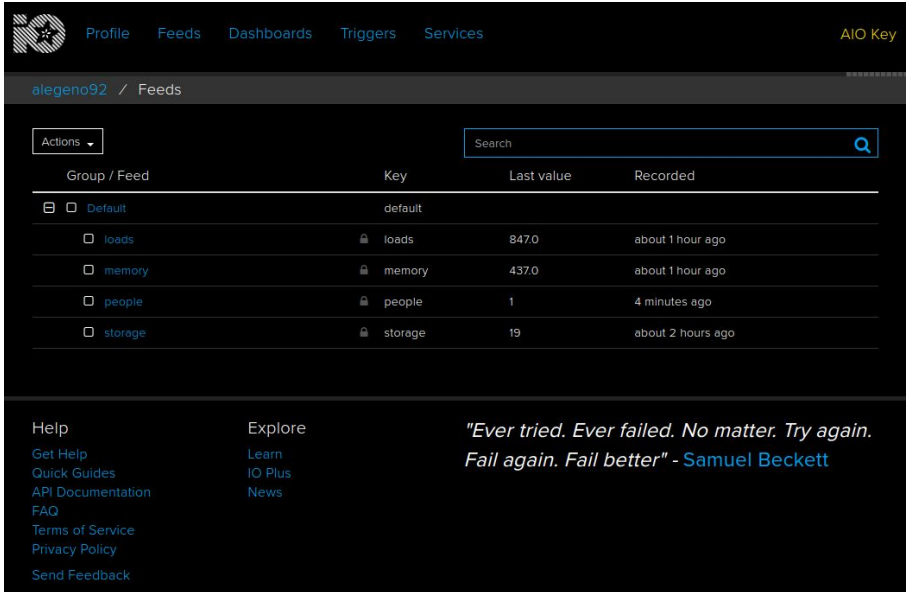
What we are going to implement

Restful Server - Dashboard

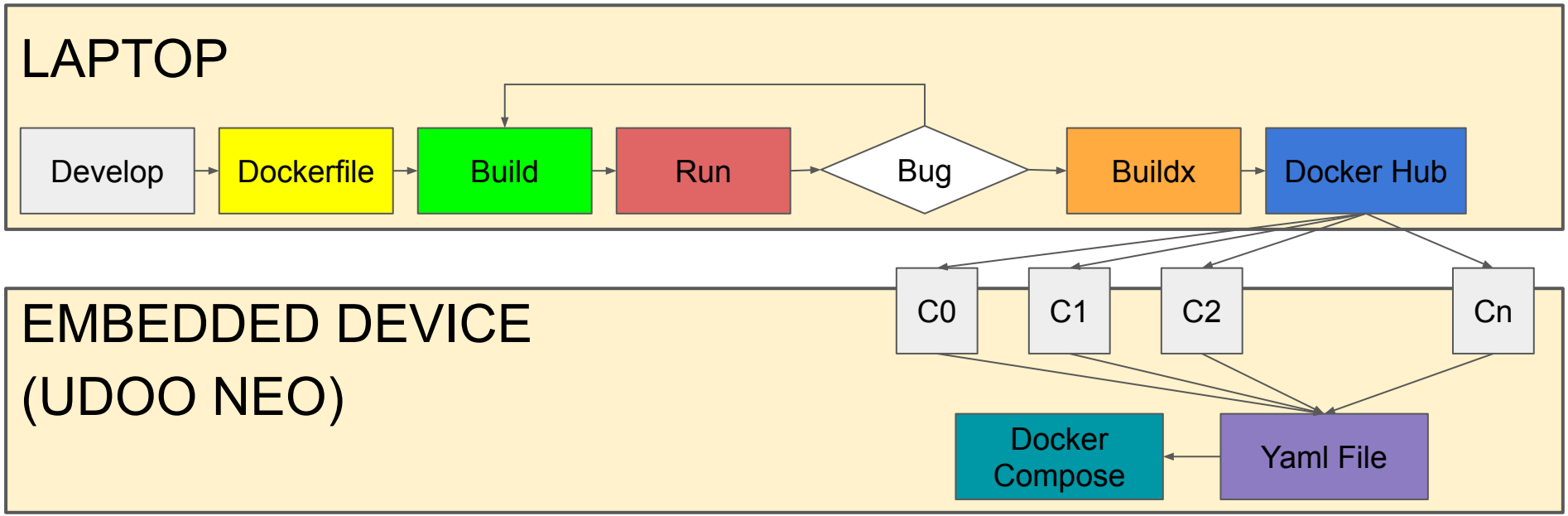


What we are going to implement

Adafruit IO - Cloud Dashboard



Development Workflow



Workshop Flow

What are we going to do:

- ~~Define the Docker Container structure for the project (Docker Approach)~~
- **Test Docker Engine runs on the Laptop**
- Build Python Container from scratch
- Build C++ Container from scratch
- Use Buildx to create containers for different platform and Push them on DockerHub
- Startup the Arm board and test the Docker Engine runs
- Connect many Docker containers together using Docker Compose
- Deploy on UDOO NEO (Linux Ubuntu)
- Deploy on an Industrial IoT Gateway with EDGEHOG OS (Yocto based): Seco SBC-C23, Seco SBC-C61

Docker

Installation

Check that **docker engine** and **docker-compose** are installed locally

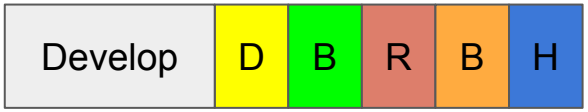
```
docker --version
docker-compose --version
```

Workshop Flow

What are we going to do:

- ~~Define the Docker Container structure for the project (Docker Approach)~~
- ~~Test Docker Engine runs on the Laptop~~
- **Build Python Container from scratch**
- Build C++ Container from scratch
- Use Buildx to create containers for different platform and Push them on DockerHub
- Startup the Arm board and test the Docker Engine runs
- Connect many Docker containers together using Docker Compose
- Deploy on UDOO NEO (Linux Ubuntu)
- Deploy on an Industrial IoT Gateway with EDGEHOG OS (Yocto based): Seco SBC-C23, Seco SBC-C61

Docker



Develop

Clone the cloud connector app from Github.

```
git clone https://github.com/alegeno92/cloud-connector.git
cd cloud-connector
```

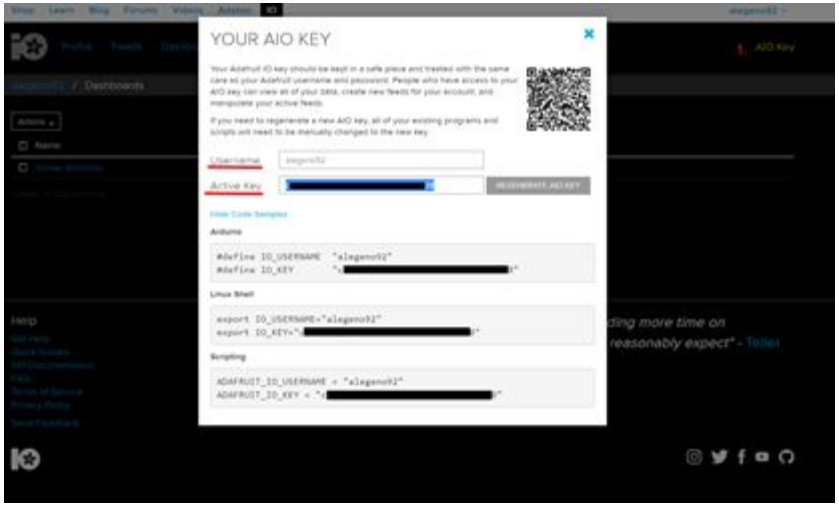
Docker

Develop	D	B	R	B	H

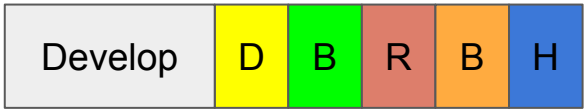
Develop - Send data to Adafruit IO cloud

Register to Adafruit IO: https://accounts.adafruit.com/users/sign_up

Generate a new Auth Token:



Docker



Develop - Send data to Adafruit IO cloud

Edit the file config.json in order to see the data online:

```
{
  "AIO": {
    "username": "",
    "key": ""
  },
  "mock_client": true
...
}
```

Docker

Dockerfile

Create the Dockerfile in the root of the project

```
FROM python:3-alpine
RUN pip install --no-cache-dir paho-mqtt adafruit-io
RUN mkdir /app
WORKDIR /app
COPY *.py /app/
CMD ["python", "-u", "/app/main.py" , "/config/config.json"]
```

Docker

Build

First of all, build the container:

```
docker build . -t cloud-connector
```

Check if everything works fine:

```
docker image ls
```

Docker

Run

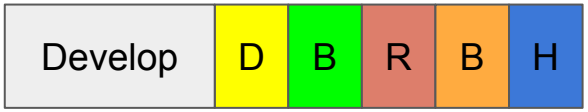

```
docker run -v <directory_path_of_config.json>:/config --network=host cloud-connector
```

Workshop Flow

What are we going to do:

- ~~Define the Docker Container structure for the project (Docker Approach)~~
- ~~Test Docker Engine runs on the Laptop~~
- ~~Build Python Container from scratch~~
- **Build C++ Container from scratch**
- Use Buildx to create containers for different platform and Push them on DockerHub
- Startup the Arm board and test the Docker Engine runs
- Connect many Docker containers together using Docker Compose
- Deploy on UDOO NEO (Linux Ubuntu)
- Deploy on an Industrial IoT Gateway with EDGEHOG OS (Yocto based): Seco SBC-C23, Seco SBC-C61

Docker



Develop - A more complex application: the Telemetry Agent

Clone the cloud connector app from Github.

```
cd ~
git clone https://github.com/alegeno92/agent.git
cd agent
```

Docker

Dockerfile - A more complex application: the Telemetry Agent

Dockerizing the telemetry agent: a C++ application that depends on many libraries (e.g. Paho MQTT C and C++ libraries). We use a smart approach.

Split the Dockerfile in two (or more):

- Build:
  - Build tools: cmake, gcc, dev libraries, …
  - Build dependencies
  - Build the application
- Runtime:
  - Binary app
  - Shared libraries

Docker

Dockerfile

First of all, start building the Paho MQTT C library:

```
FROM alpine:latest AS paho-c
RUN apk add --no-cache cmake g++ make
RUN apk add --no-cache git libressl-dev
RUN git clone https://github.com/eclipse/paho.mqtt.c.git
WORKDIR paho.mqtt.c
RUN git checkout v1.3.1
RUN cmake -Bbuild -H. -DPAHO_WITH_SSL=ON -DPAHO_ENABLE_TESTING=OFF
RUN cmake --build build/ --target install
```

Docker

Dockerfile

Next, starting from the previous containere build the Paho MQTT C++ library:

```
FROM paho-c AS paho-cpp
WORKDIR /
RUN apk add --no-cache git libressl-dev
RUN apk add --no-cache --upgrade bash
RUN git clone https://github.com/eclipse/paho.mqtt.cpp.git
WORKDIR paho.mqtt.cpp
RUN cmake -Bbuild -H. -DPAHO_BUILD_DOCUMENTATION=FALSE -DPAHO_BUILD_SAMPLES=FALSE
RUN cmake --build build/ --target install
```

Docker

Dockerfile

Next, let's build our project:

```
FROM paho-cpp AS build-container
RUN apk add --no-cache zlib-dev
COPY . /home/agent
WORKDIR /home/agent
RUN if [ \-d "./build" ] ; then rm -rvf build  && mkdir build ; else mkdir build ; fi
WORKDIR build
RUN cmake ..
RUN cmake --build . --target agent --
```

Docker

Dockerfile

At the end, the cunning idea: starting from a vanilla alpine container, copy on it only the built libraries and the necessary files:

```
FROM alpine:latest
RUN apk add --no-cache zlib libressl libstdc++
COPY --from=build-container /paho.mqtt.c/build/src/* /usr/lib/
COPY --from=build-container /paho.mqtt.cpp/build/src/* /usr/lib/
COPY --from=build-container /home/agent/build/bin/agent /
CMD ["/agent", "/config/config.json"]
```

Docker

Build

Build the container：

```
docker build -t agent .
```

Check if everything works fine:

```
docker image ls
```

Docker

Run

Run the container：

```
docker run -v /home/alessandro/Projects/agent/config:/config/ --network=host agent
```

Workshop Flow

What are we going to do:

- ~~Define the Docker Container structure for the project (Docker Approach)~~
- ~~Test Docker Engine runs on the Laptop~~
- ~~Build Python Container from scratch~~
- ~~Build C++ Container from scratch~~
- **Use Buildx to create containers for different platform and Push them on DockerHub**
- Startup the Arm board and test the Docker Engine runs
- Connect many Docker containers together using Docker Compose
- Deploy on UDOO NEO (Linux Ubuntu)
- Deploy on an Industrial IoT Gateway with EDGEHOG OS (Yocto based): Seco SBC-C23, Seco SBC-C61

Docker Buildx

New CLI plugin that extends the docker command with the full support of the features provided by Moby BuildKit builder toolkit.

Supports multi-architecture builds for both x86 and Arm (one build, multiple images)

Using the Buildx feature to create images for different kind of architectures at the same time. (refs. https://docs.docker.com/buildx/working-with-buildx/)

Docker Buildx

Create a new Buildx builder

Create our builder

```
docker buildx create --name arm_builder
```

List the builders

```
docker buildx ls
```

Starts the builder

```
docker buildx inspect --bootstrap
```

Use the builder

```
docker buildx use arm_builder
```

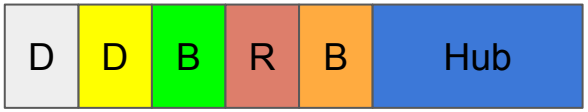
Docker Buildx

Build the Telemetry Agent using buildx

Build the containers for the needed platforms:

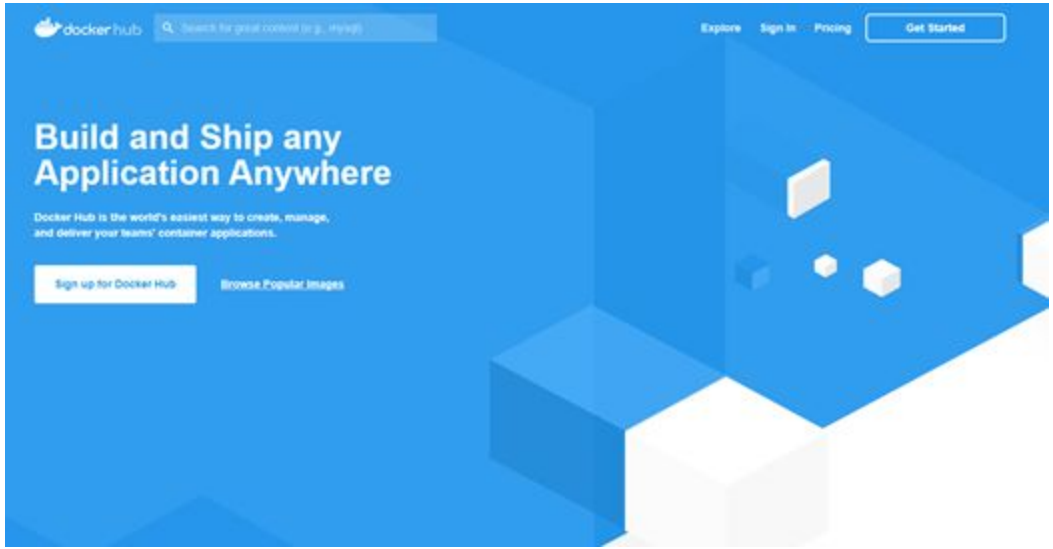
```
docker buildx build -t alegeno92/agent  --platform linux/arm64,linux/arm/v7 --load .
```

DockerHub

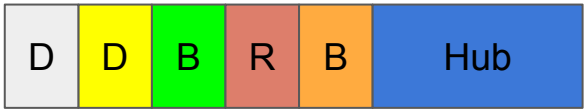


Uploading an image to DockerHub

Create an account to DockerHub: [hub.docker.com]



DockerHub



Login to DockerHub

- Login to DockerHub via cli:

```
docker login
```

Docker Buildx

Build the REST Server using Buildx and push on DockerHub

Build the agent for different platforms and push them on DockerHub:

```
docker buildx build -t alegeno92/agent --platform linux/amd64,linux/arm/v7   --push .
```

Workshop Flow

What are we going to do:

- ~~Define the Docker Container structure for the project (Docker Approach)~~
- ~~Test Docker Engine runs on the Laptop~~
- ~~Build Python Container from scratch~~
- ~~Build C++ Container from scratch~~
- ~~Use Buildx to create containers for different platform and Push them on DockerHub~~
- **Startup the Arm board and test the Docker Engine runs**
- Connect many Docker containers together using Docker Compose
- Deploy on UDOO NEO (Linux Ubuntu)
- Deploy on an Industrial IoT Gateway with EDGEHOG OS (Yocto based): Seco SBC-C23, Seco SBC-C61

UDOO NEO SSH connection

A first condition to establish a SSH connection with your UDOO Neo is to have previously completed the tutorial about the USB Direct Connection.

https://github.com/alegeno92/docker-workshop/blob/master/SSH_connection.md

A second condition is to download and install an SSH Client for your system (if not):

- Windows: Putty
- Linux: Shell
- Mac: Shell

UDOO NEO SSH connection

Open an SSH connection to the board

In your laptop run:

```
ssh udooer@192.168.7.2  (password: udooer)
cd ~
```

Workshop Flow

What are we going to do:

- ~~Define the Docker Container structure for the project (Docker Approach)~~
- ~~Test Docker Engine runs on the Laptop~~
- ~~Build Python Container from scratch~~
- ~~Build C++ Container from scratch~~
- ~~Use Buildx to create containers for different platform and Push them on DockerHub~~
- ~~Startup the Arm board and test the Docker Engine runs~~
- **Run many Docker Containers together using Docker Compose**
- Deploy on UDOO NEO (Linux Ubuntu)
- Deploy on an Industrial IoT Gateway with EDGEHOG OS (Yocto based): Seco SBC-C23, Seco SBC-C61

Docker Compose

A tool to run many docker containers together

- Compose is a tool for defining and running multi-container Docker applications.
- It uses a YAML file to configure your application's services.
- With a single command, you create and start all the services from your configuration.

Docker Compose

Download the Docker compose repo

```
ssh udooer@192.168.7.2  (password: udooer)
cd ~
git clone https://github.com/alegeno92/compose.git (if there isn't)
cd compose
```

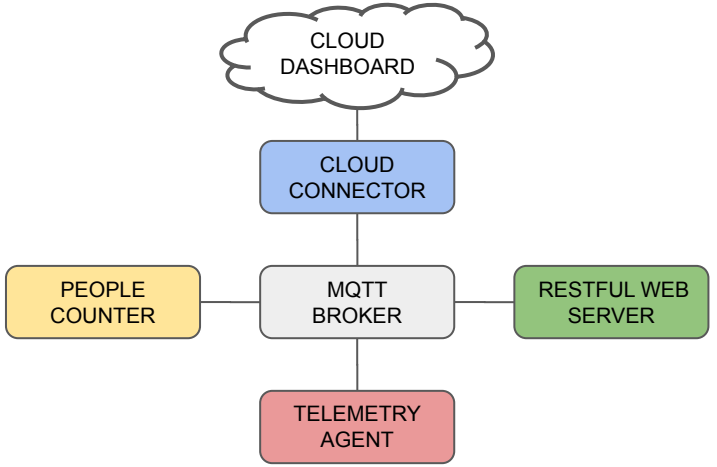
Docker Compose

C YAML

The YAML file

```
version: '3'
services:
  server:
    image: alegeno92/server:latest
    ports:
      - "8000:8000"
      - "8086:8086"
    depends_on:
      - "mosquitto"
      - "people-counter"
    volumes:
      - ./configurations/server:/config/
```

RESTFUL WEB SERVER

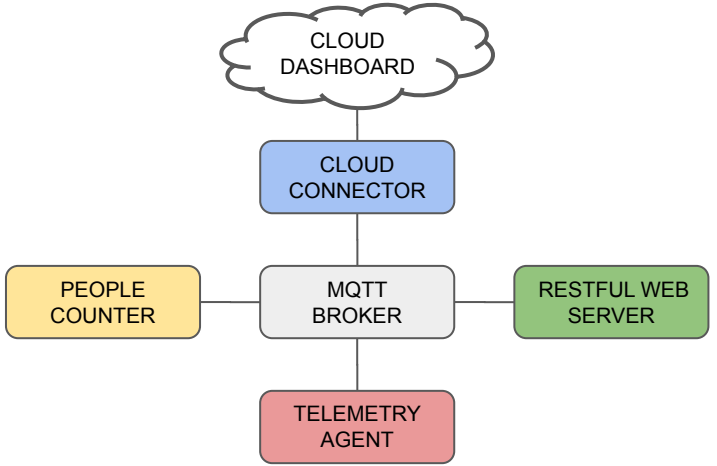


Docker

C YAML

The YAML file

```
....
agent:                                          TELEMETRY AGENT
 image: alegeno92/agent:latest
 hostname: agent
 depends_on:
   - "mosquitto"
 volumes:
   - ./configurations/agent:/config/
....
```

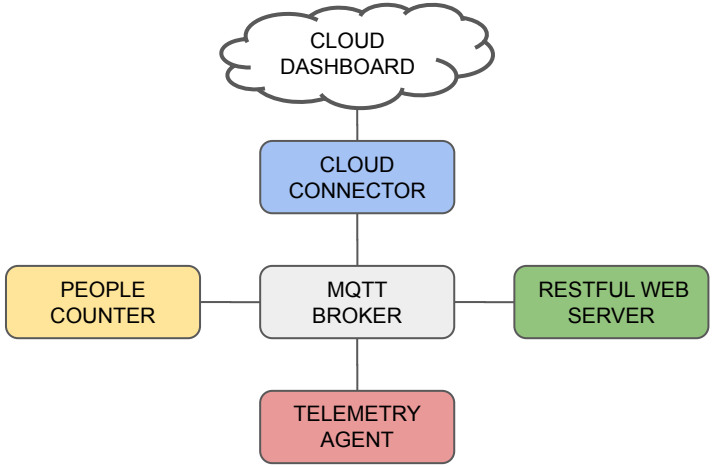


Docker

C YAML

The YAML file

```
....
cloud-connector:                                CLOUD CONNECTOR
  image: alegeno92/cloud-connector:latest
  hostname: cloudconnector
  depends_on:
    - "mosquitto"
  volumes:
    - ./configurations/cloud-connector/:/config
  environment:
    - PYTHONUNBUFFERED=1
....
```



Docker

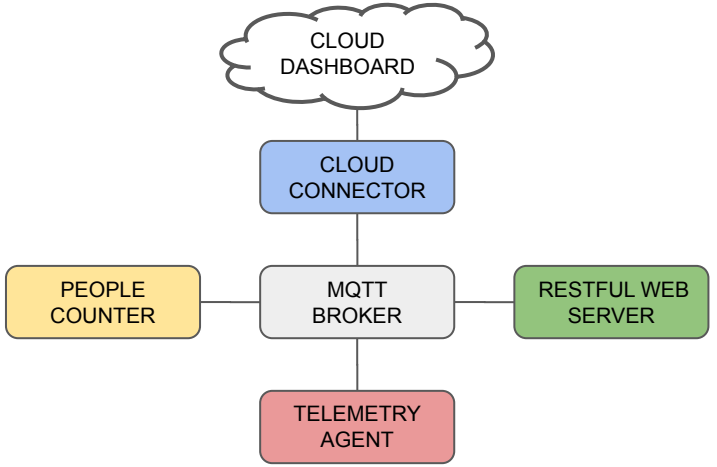
C YAML

The YAML file

….

```
people-counter:                                    PEOPLE COUNTER
 image: alegeno92/people-counter:latest
 depends_on:
   - "mosquitto"
 expose:
   - "8084"
 ports:
   - "8084:8084"
 volumes:
   - ./configurations/people-counter/:/config
 environment:
   - PYTHONUNBUFFERED=1
```

….



Workshop Flow

What are we going to do:

- ~~Define the Docker Container structure for the project (Docker Approach)~~
- ~~Test Docker Engine runs on the Laptop~~
- ~~Build Python Container from scratch~~
- ~~Build C++ Container from scratch~~
- ~~Use Buildx to create containers for different platform and Push them on DockerHub~~
- ~~Startup the Arm board and test the Docker Engine runs~~
- ~~Connect many Docker containers together using Docker Compose~~
- **Deploy on UDOO NEO (Linux Ubuntu)**
- Deploy on an Industrial IoT Gateway with EDGEHOG OS (Yocto based): Seco SBC-C23, Seco SBC-C61

Docker Compose

Testing the App on the UDOO NEO

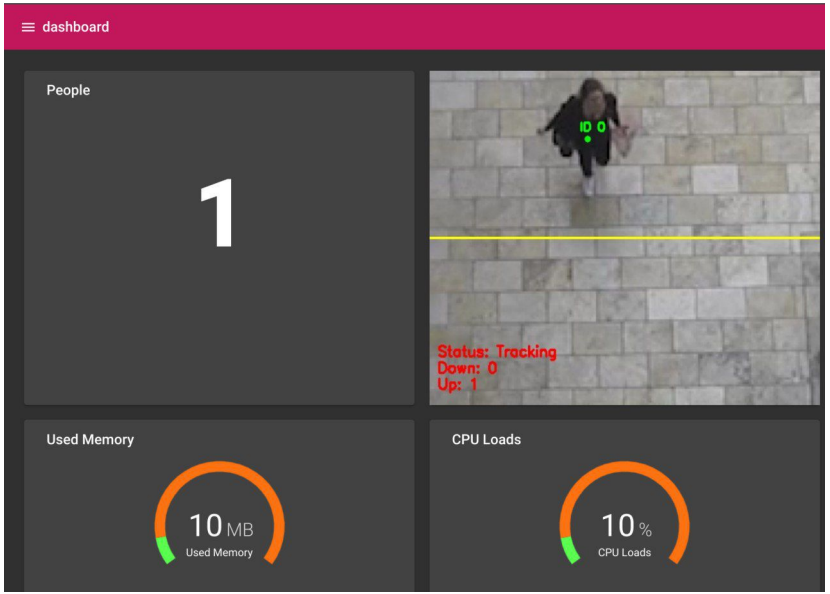
Launch the Docker Compose command:

```
docker-compose -f "docker-compose.yml" up
```

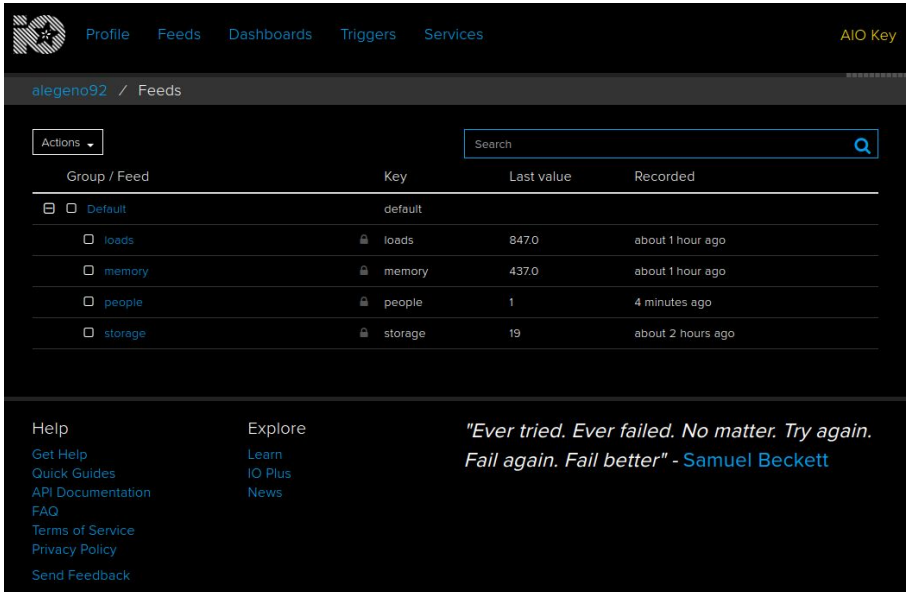
And the magic happens!! :D

The final outcome

192.168.7.2:8000



Adafruit IO



Workshop Flow

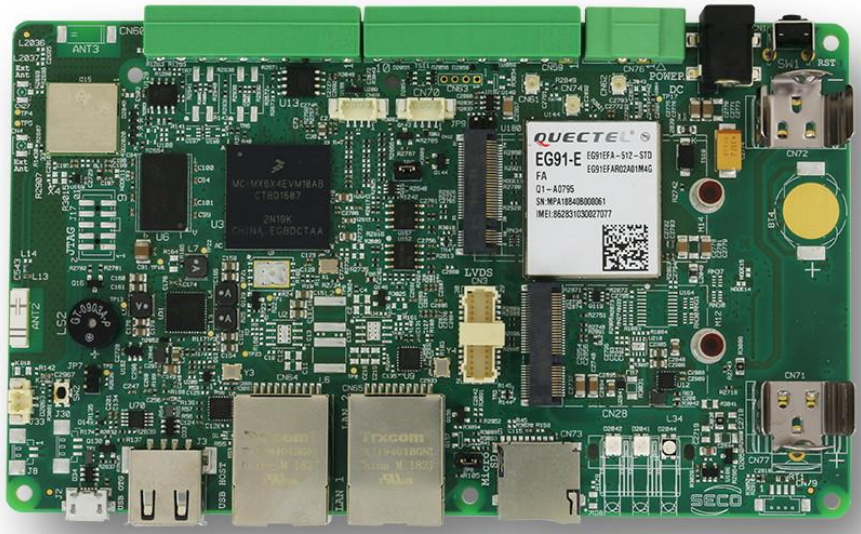
What are we going to do:

- ~~Define the Docker Container structure for the project (Docker Approach)~~
- ~~Test Docker Engine runs on the Laptop~~
- ~~Build Python Container from scratch~~
- ~~Build C++ Container from scratch~~
- ~~Use Buildx to create containers for different platform and Push them on DockerHub~~
- ~~Startup the Arm board and test the Docker Engine runs~~
- ~~Connect many Docker containers together using Docker Compose~~
- ~~Deploy on UDOO NEO (Linux Ubuntu)~~
- **Deploy on an Industrial IoT Gateway with EDGEHOG OS (Yocto based): Seco SBC-C23, Seco SBC-C61**

The final outcome

Moving on the industrial case

SECO SBC C23 Industrial Board based on the same processor of the UDOO NEO and running EDGEHOG OS (Yocto).



Docker

Compose	Y

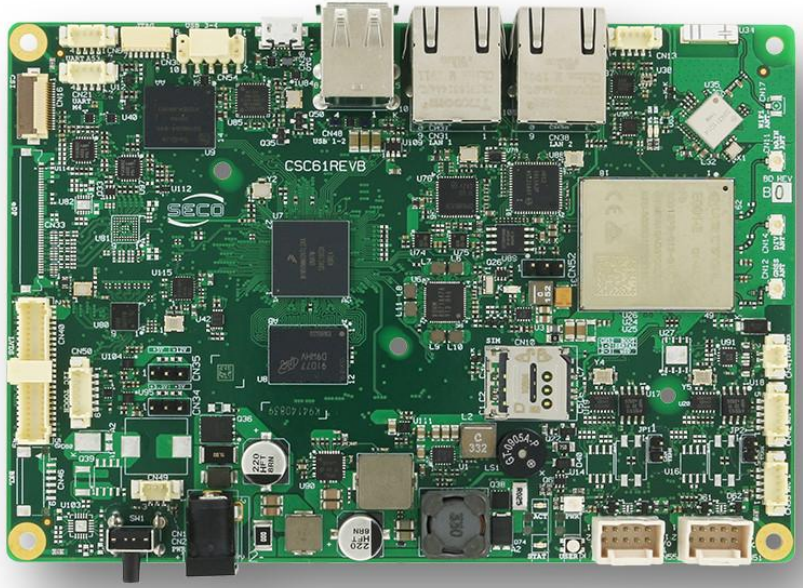
Testing on industrial board

Copy Docker Compose and launch the same command on the SECO SBC C23

The final outcome

I need to scale performance

SECO SBC C61 Industrial Board based on i.MX 8M Mini Quad – Full featured, 4x Cortex®- A53 cores up to 1.8GHz and running EDGEHOG OS (Yocto).



Docker

Compose	Y

Testing on industrial board

As before, copy Docker Compose and launch the same command on the SECO SBC C61

Conclusions

PROs

Docker is a smart solution for the development of complex applications on an ARM embedded device

Buildx is a very powerful tool (easy cross compilation)

Application Configurations is the same for different platforms

Useful for CI and CD: update made by download the updated container

Good Performance wise: Docker is not a Virtual Machine, the application runs natively (thanks to cgroup feature of Linux Kernel)

High security wise: each container runs inside a sandbox. Who runs it decides the accessible resources

CONS

Heavy disk utilization: each containers contains a replica of an entire file system

Questions?

Thank you

Maurizio Caporali, Alessandro Genovese, Jenny Fong

Useful Links

Something to think about

Workshop Resources:

https://github.com/alegeno92/docker-workshop.git

www.docker.com

www.udoo.org

www.seco.com