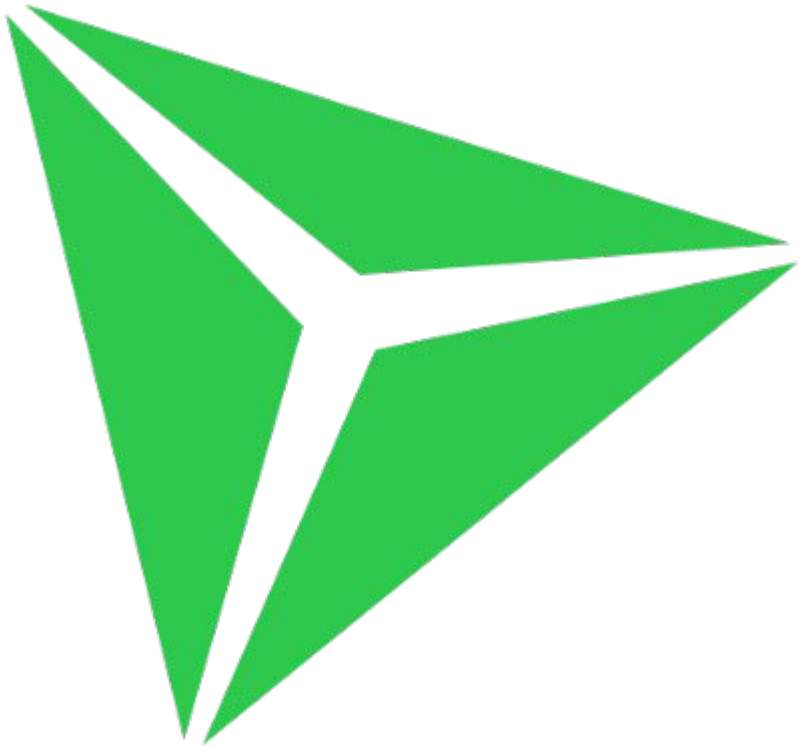
MISSION TO MARS:
EXPLORING NEW WORLDS WITH AWS IOT

Jeroen Resoort
@JeroenResoort @jdriven_nl

About me

Jeroen Resoort

JDriven

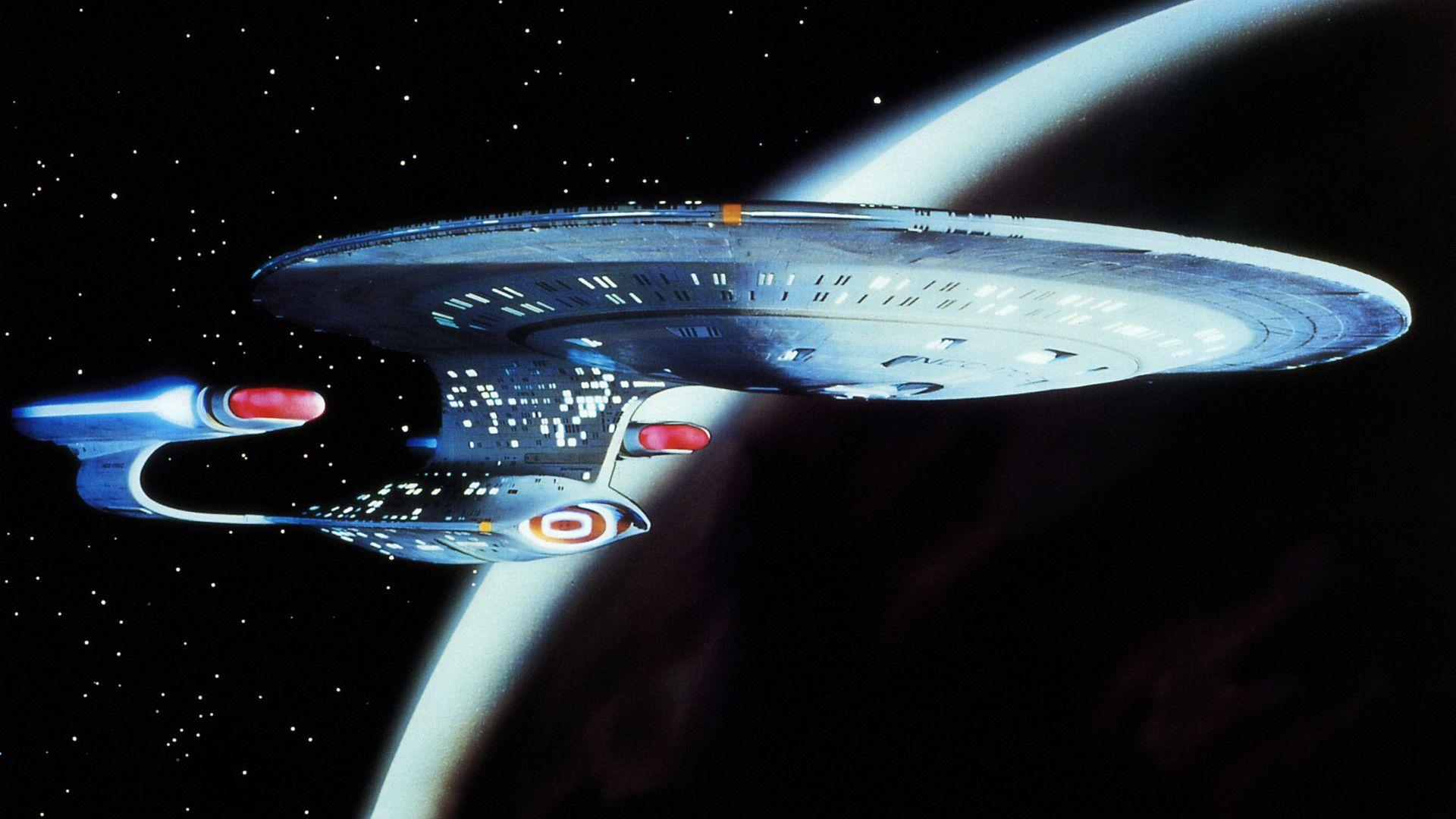


Learn new things

About this talk

- Robot
  - Hardware
  - Software
- AWS IoT platform
- Demo
- AWS IoT rules engine examples


Inspiration?

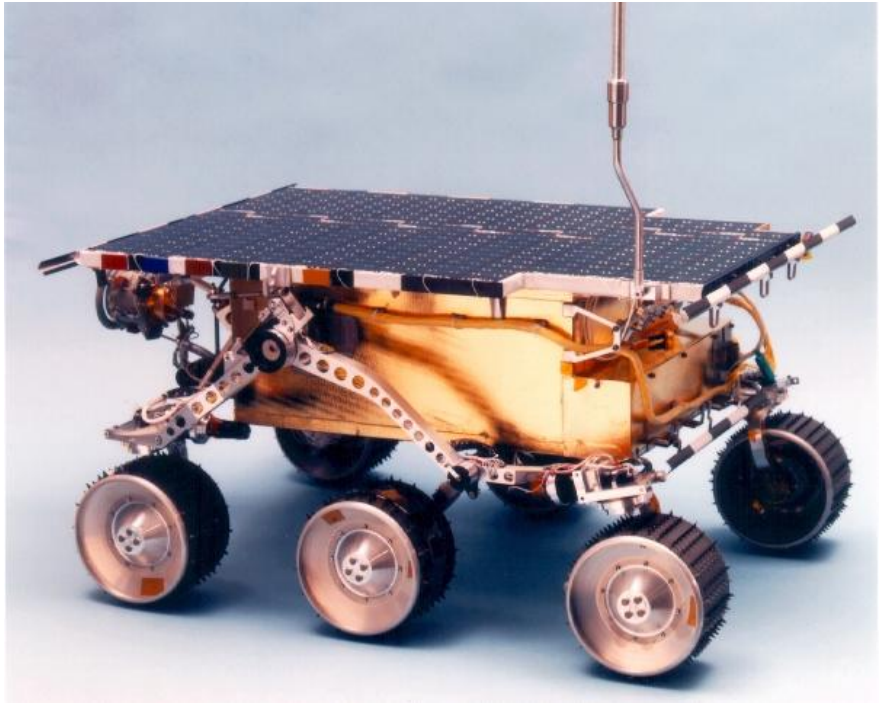


Pathfinder mission

Pathfinder landed in 1997

Sojourner Rover explored the surface of Mars for 3 months

Several other missions followed



Our own robot

What do we want it to do?

Our own robot

What do we want it to do?

- Move around
- Take pictures
- Gather data

Our own robot

What does our robot need?

Our own robot

What does our robot need?

- Power supply
- Connectivity (internet)
- Camera
- Sensors

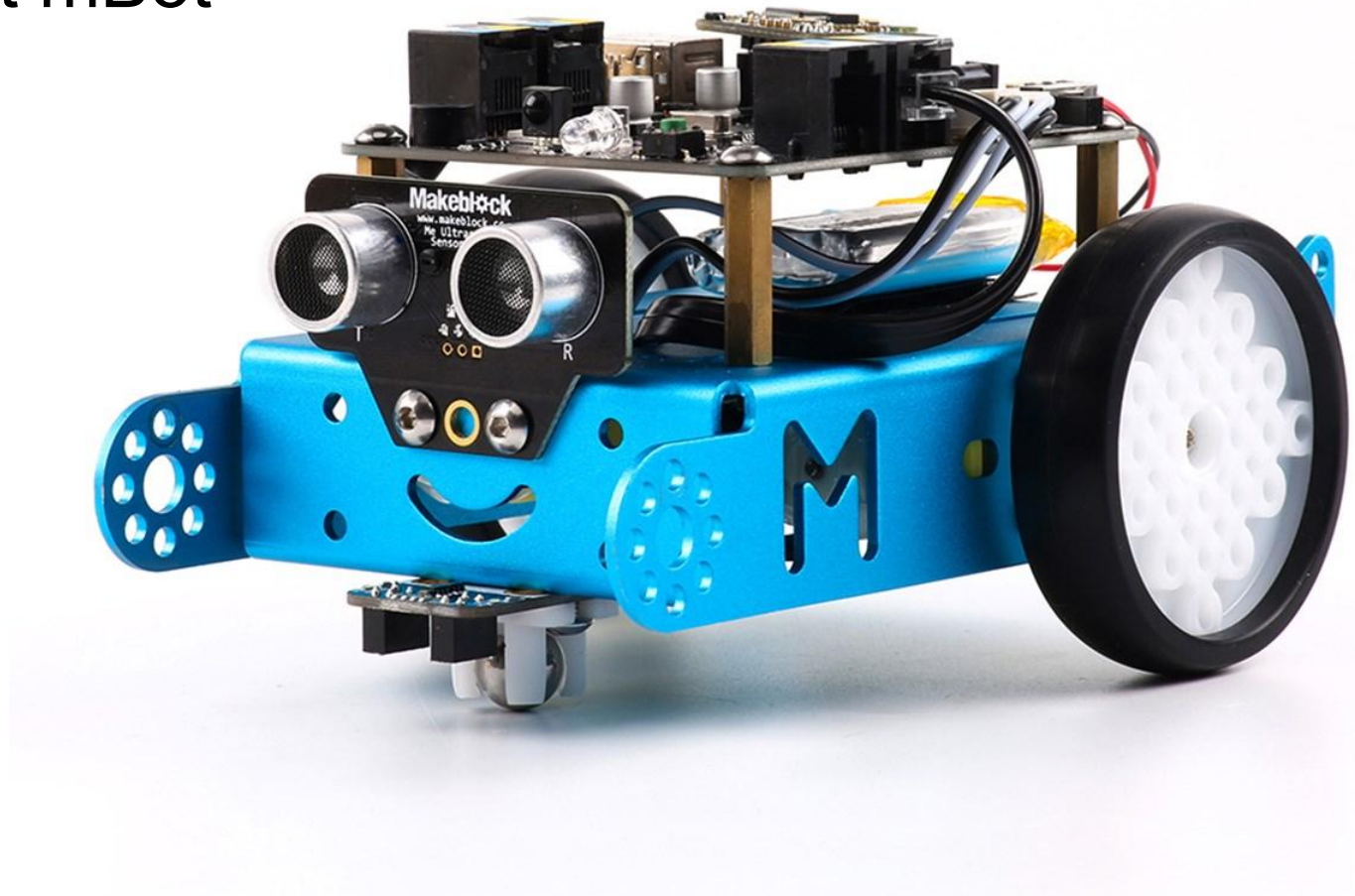
Our own robot

A lot of robots available

KickStarter project called ‘mBot’

Funded within 24 hours

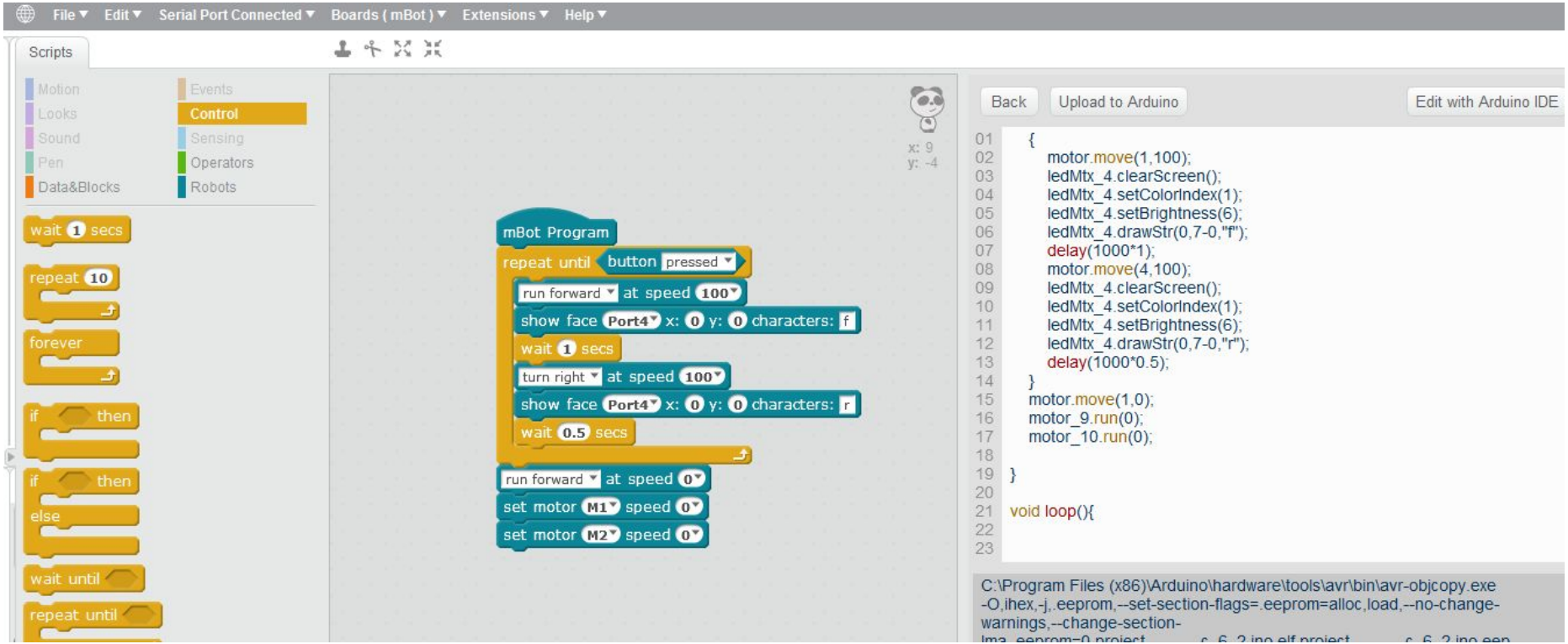
Meet mBot



mBot features

- Easy to build
- Based on arduino
- Comes with Bluetooth or 2.4GHz, infrared remote control, light sensor, leds, buttons, buzzer, line follower, ultrasonic
- Powered by AA batteries or 3.7V lithium battery

mBot features



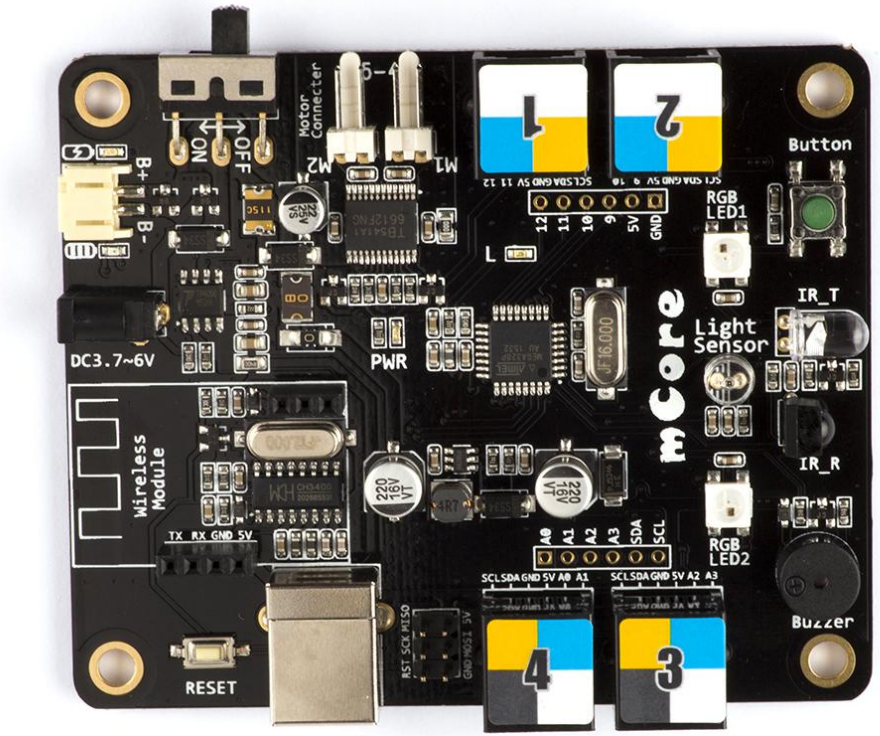
mBot @ devoxx4kids

DEVOXX™ 4KIDS Netherlands

mBot only is not enough

We also need

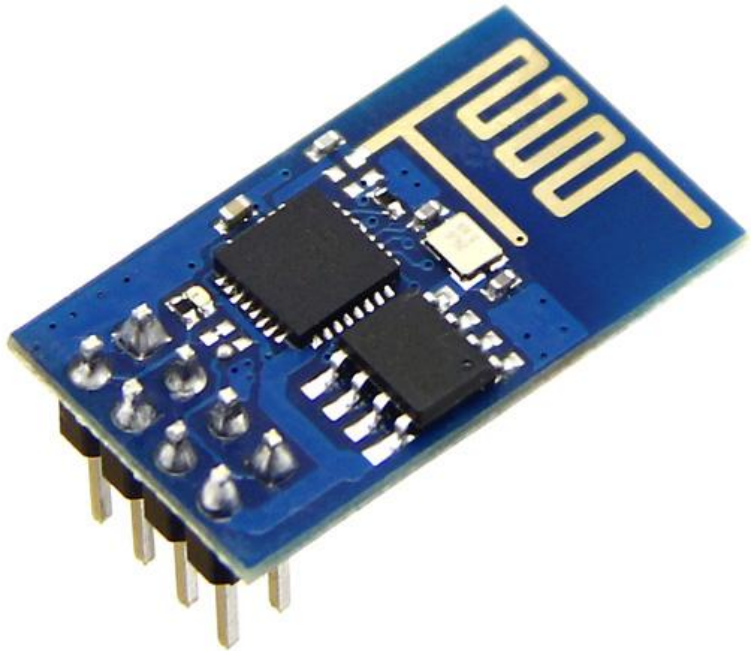
- Connectivity
- Camera
- More processing power



What about an ESP8266?

Microcontroller and WiFi

Cool and cheap… but... 96 KiB of data RAM



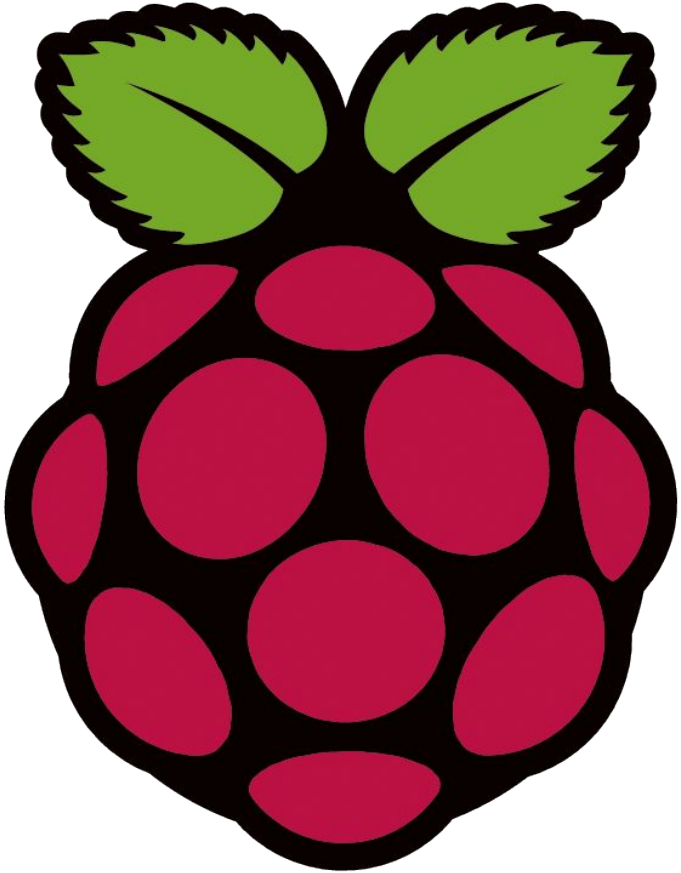
Raspberry Pi

Pi 3 has built in WiFi

Camera interface

Way more powerful

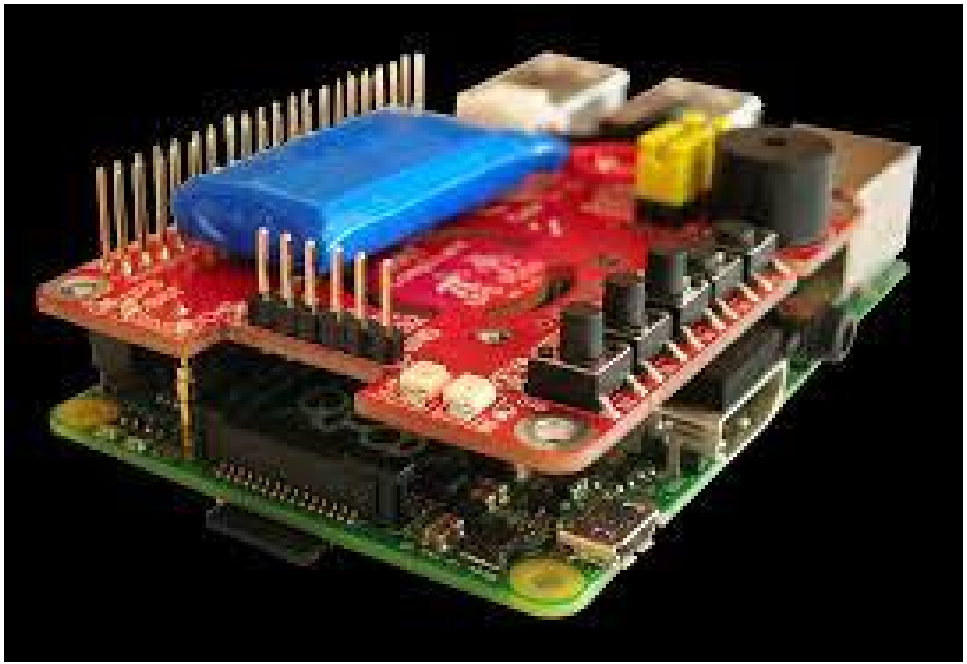
Easy to extend through GPIO header



UPS Pico

Power supply board

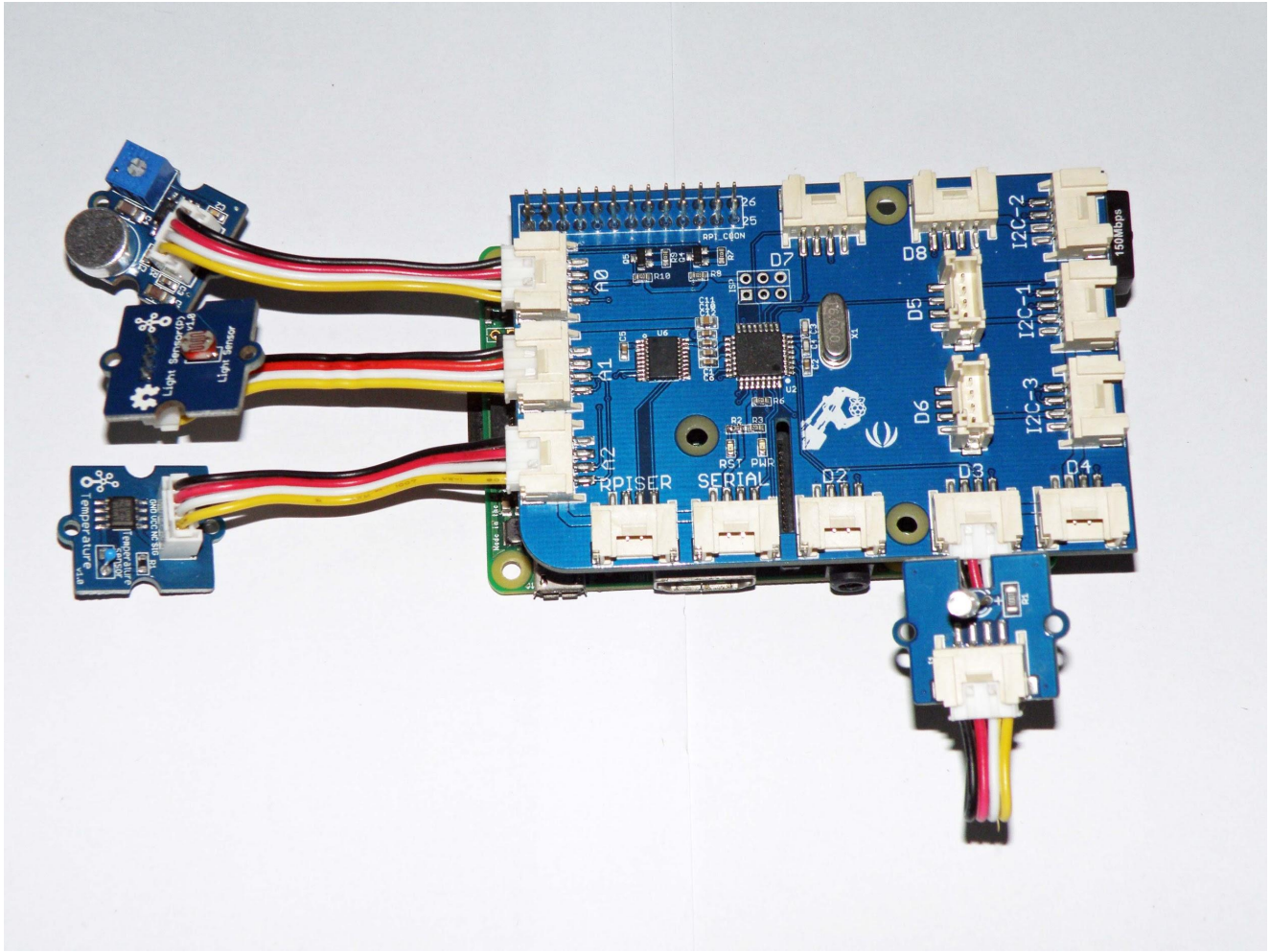
Runs for hours on 3000mAh battery



GrovePi

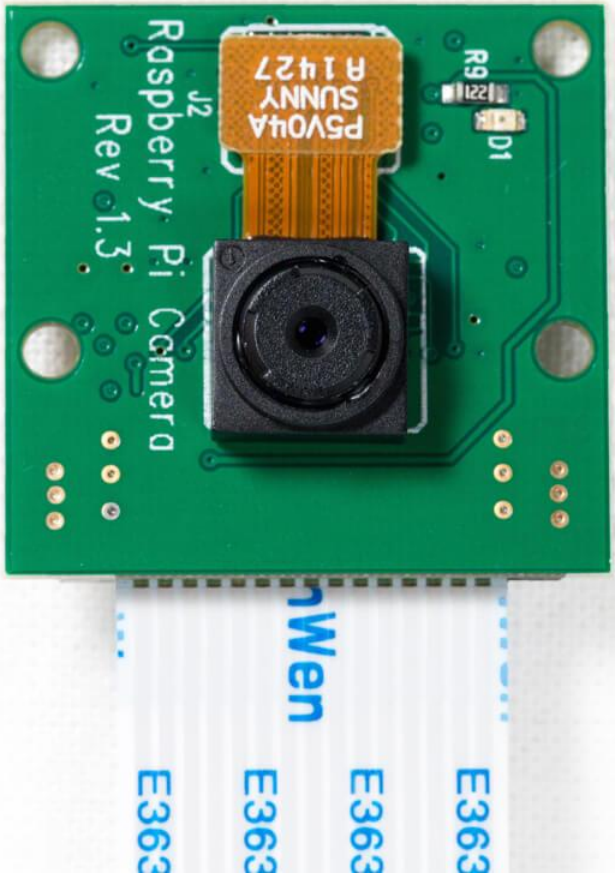
Plug-n-play

Lots of sensors available



Pi Camera

Easy to connect



Pi-Pan

Camera mount

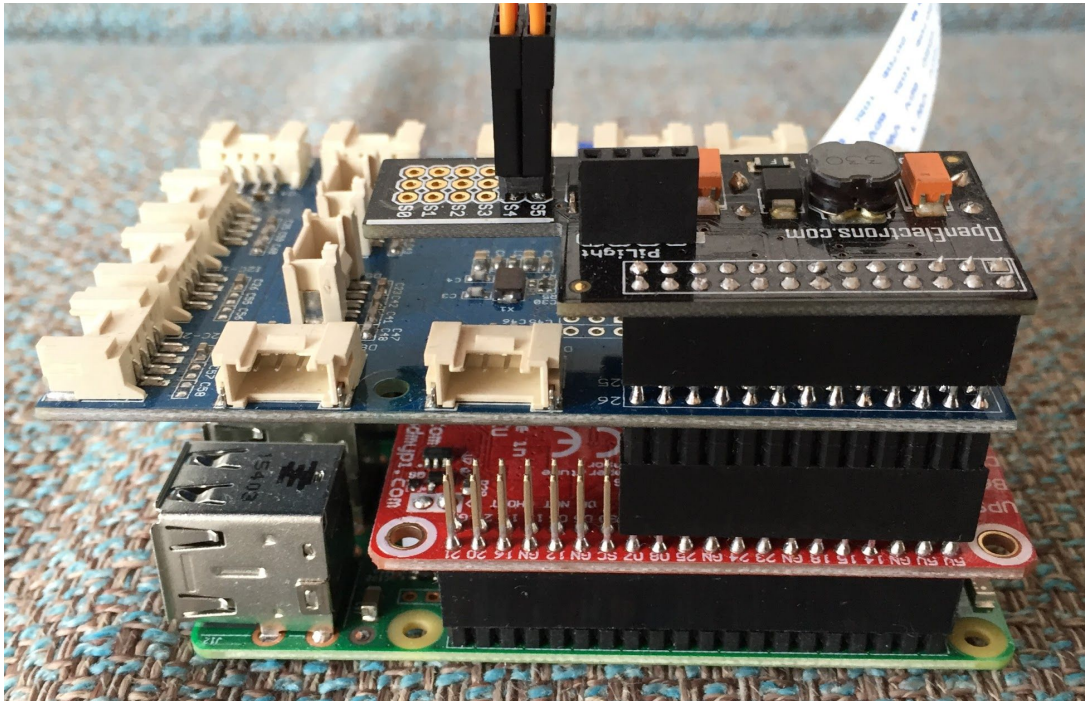
Panning and Tilting

Comes with servo controller board

Putting it all together

UPS-Pico, GrovePi and Pi-Pan controller stackable on Pi headers

Communication over i2c



Putting it all together

Raspberry Pi connects to mBot through USB

Mbot is powered through USB

USB Serial communication with mBot

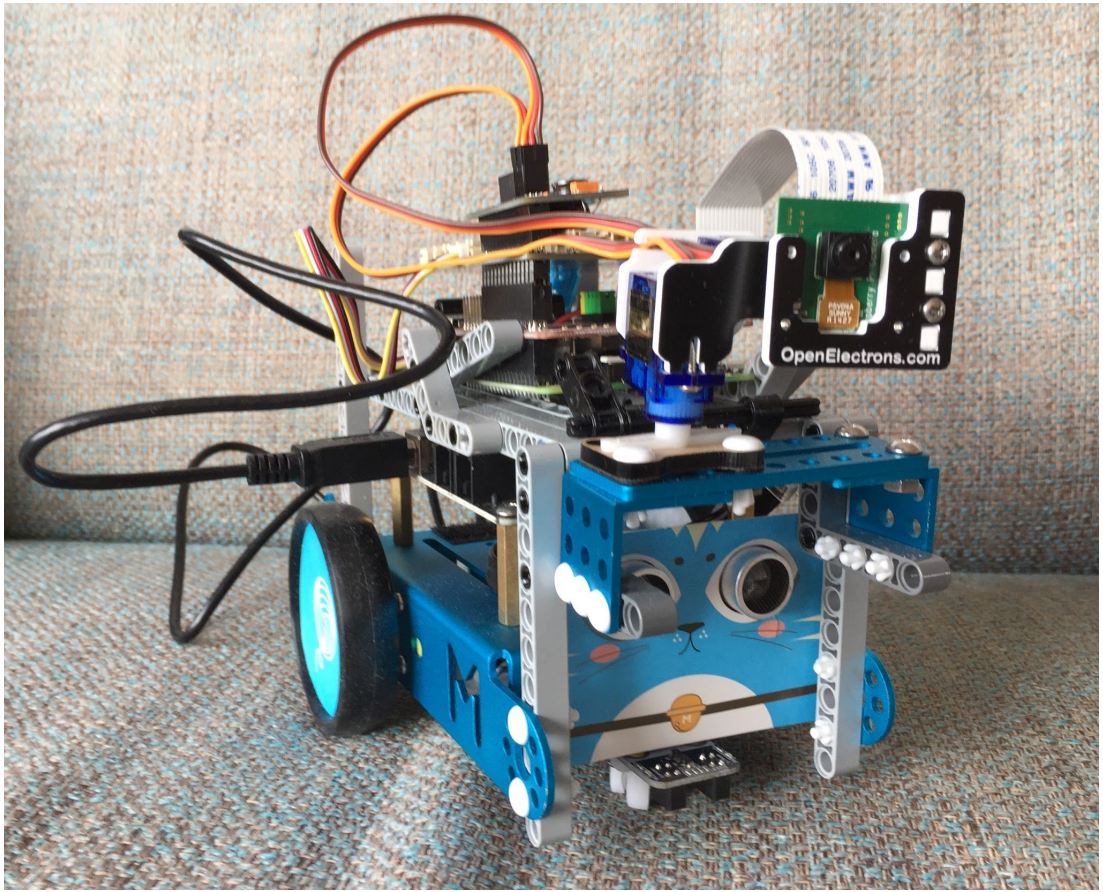
Putting it all together

But Raspberry Pi does not fit on mBot...


LEGO®

Putting it all together

Meet MarsBot



And now we need some software

Software: Python all the way

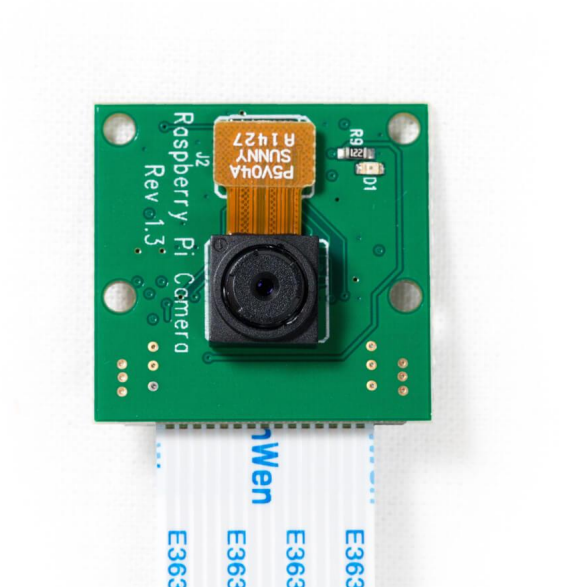
You can find a python library for everything :-)

We need to program our Pi to communicate with

- Camera
- PiPan
- GrovePi
- mBot

Software: Controlling the camera

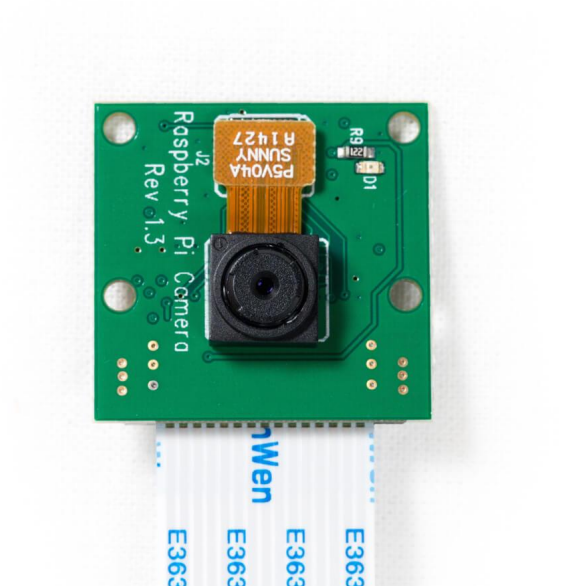
import picamera



Software: Controlling the camera

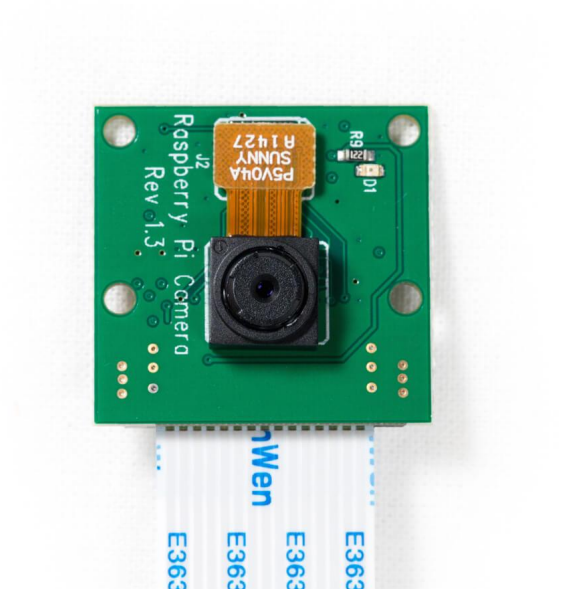
import picamera

camera = picamera.PiCamera()
camera.hflip = True
camera.vflip = True
camera.resolution = (800, 600)



Software: Controlling the camera

```
import picamera

camera = picamera.PiCamera()
camera.hflip = True
camera.vflip = True
camera.resolution = (800, 600)

camera.capture('marsbot-camera.jpg')
```



Software: Controlling the Pi-Pan

import pipan

Software: Controlling the Pi-Pan

```
import pipan

pan = pipan.PiPan()
```

Software: Controlling the Pi-Pan

```
import pipan

pan = pipan.PiPan()
```

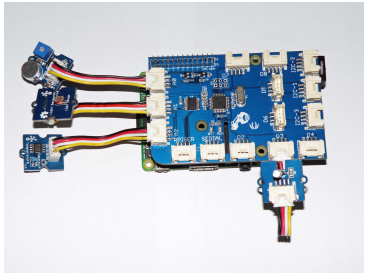
pan.neutral_pan()
pan.neutral_tilt()

Software: Controlling the Pi-Pan

```
import pipan

pan = pipan.PiPan()

pan.neutral_pan()
pan.neutral_tilt()

pan.do_pan(120)
pan.do_tilt(170)
```

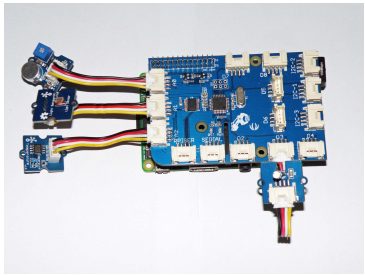
Software: Getting data from temperature sensor

```
from grovepi import *
```



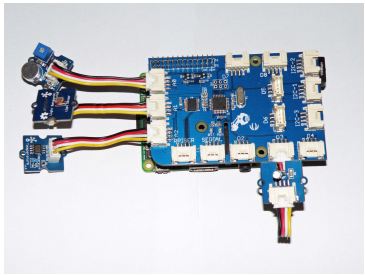
Software: Getting data from temperature sensor

```
from grovepi import *

dht_sensor_port = 7  # Connect the DHt sensor to port 7
```



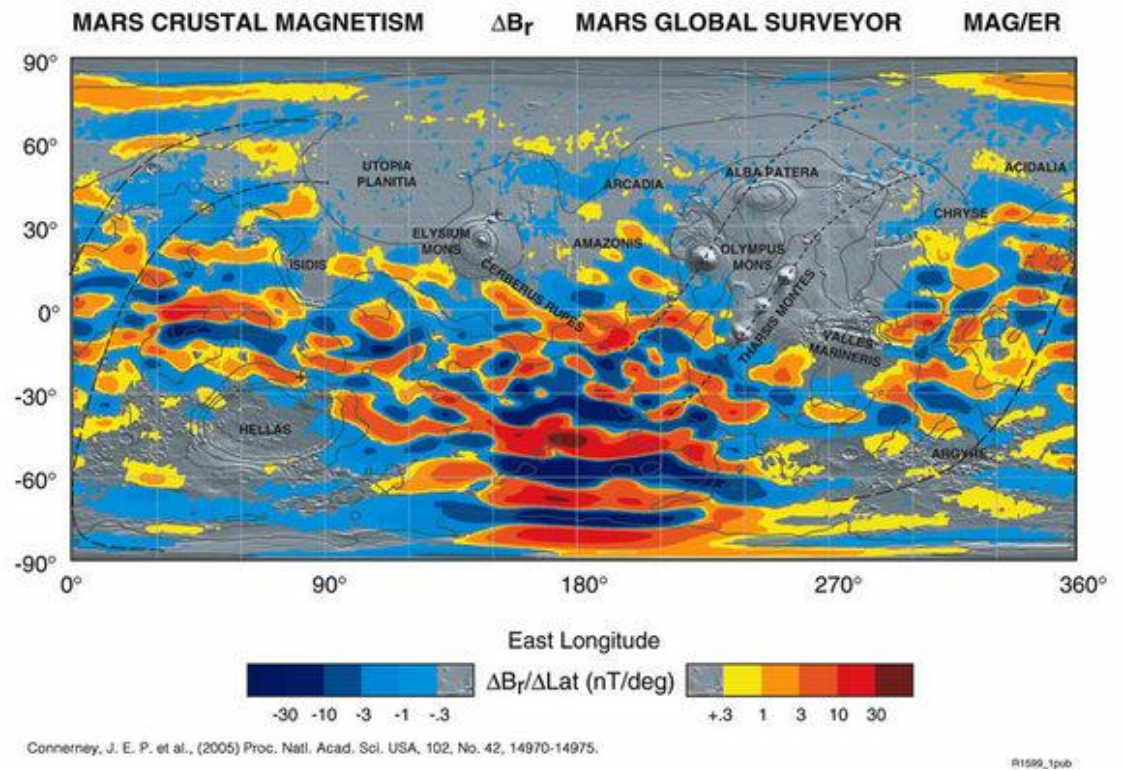
Software: Getting data from temperature sensor

```
from grovepi import *

dht_sensor_port = 7    # Connect the DHt sensor to port 7

while True:
    try:
        [ temp,hum ] = dht(dht_sensor_port, 0)
        print "temp =", temp, "C\thumidity =", hum,"%"
```



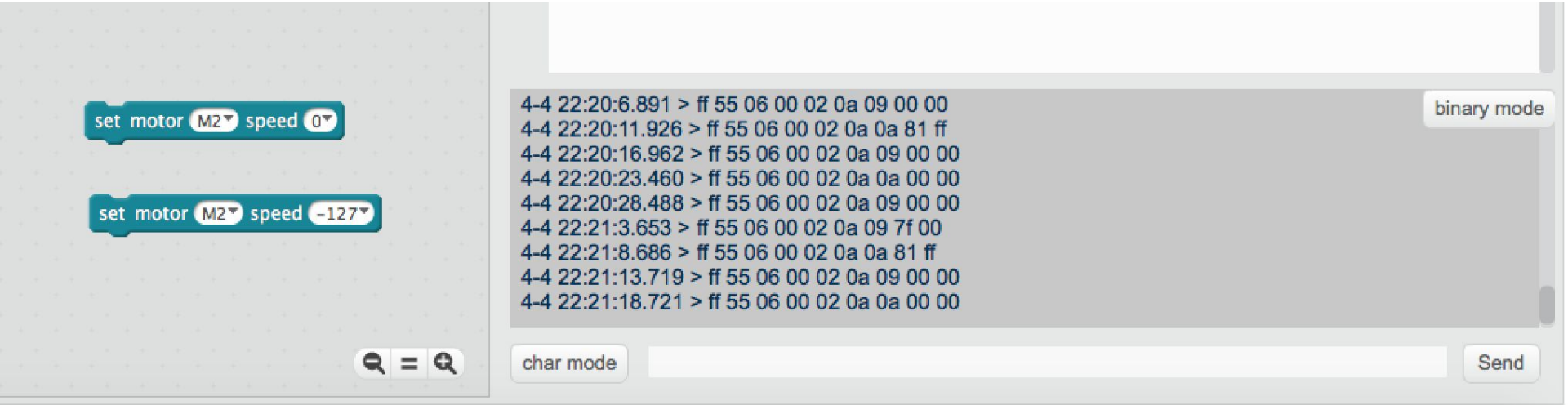
Software: Getting data from compas sensor

Oops…



Software: Controlling the mBot

Sending commands over serial connection



Software: Controlling the mBot

```
import serial
import binascii
import time
```

Software: Controlling the mBot

```
import serial
import binascii
import time

ser = serial.Serial('/dev/ttyUSB0', 115200)
```

Software: Controlling the mBot

```
import serial
import binascii
import time

ser = serial.Serial('/dev/ttyUSB0', 115200)

motor1_on  = binascii.unhexlify('ff550600020a0981ff') # half speed forward
motor1_off = binascii.unhexlify('ff550600020a090100')
motor1_rev = binascii.unhexlify('ff550600020a097f00') # half speed reverse
motor2_on  = binascii.unhexlify('ff550600020a0a7f00')
motor2_off = binascii.unhexlify('ff550600020a0a0000')
motor2_rev = binascii.unhexlify('ff550600020a0a81ff')
```

Software: Controlling the mBot

```
ser.write(motor1_on)
ser.write(motor2_on)
time.sleep(1)
ser.write(motor1_off)
ser.write(motor2_off)
```

Now we have

- a robot
- software running on the robot

But we need more...


MISSION CONTROL CENTER
esa
PAYLOAD DEPLOY/ RETRIEVAL (PDRS)

Amazon Web Services

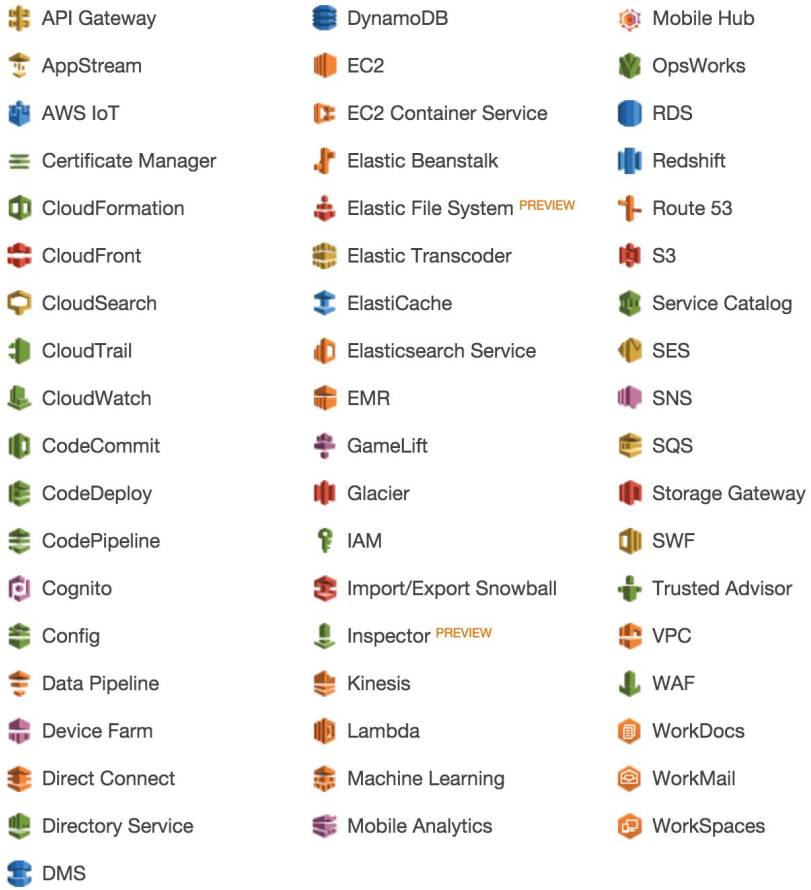
One of the biggest cloud services providers

Huge number of cloud services

Available around the globe

AWS IoT as a messaging platform for your IoT devices

Connect AWS IoT to other Amazon services



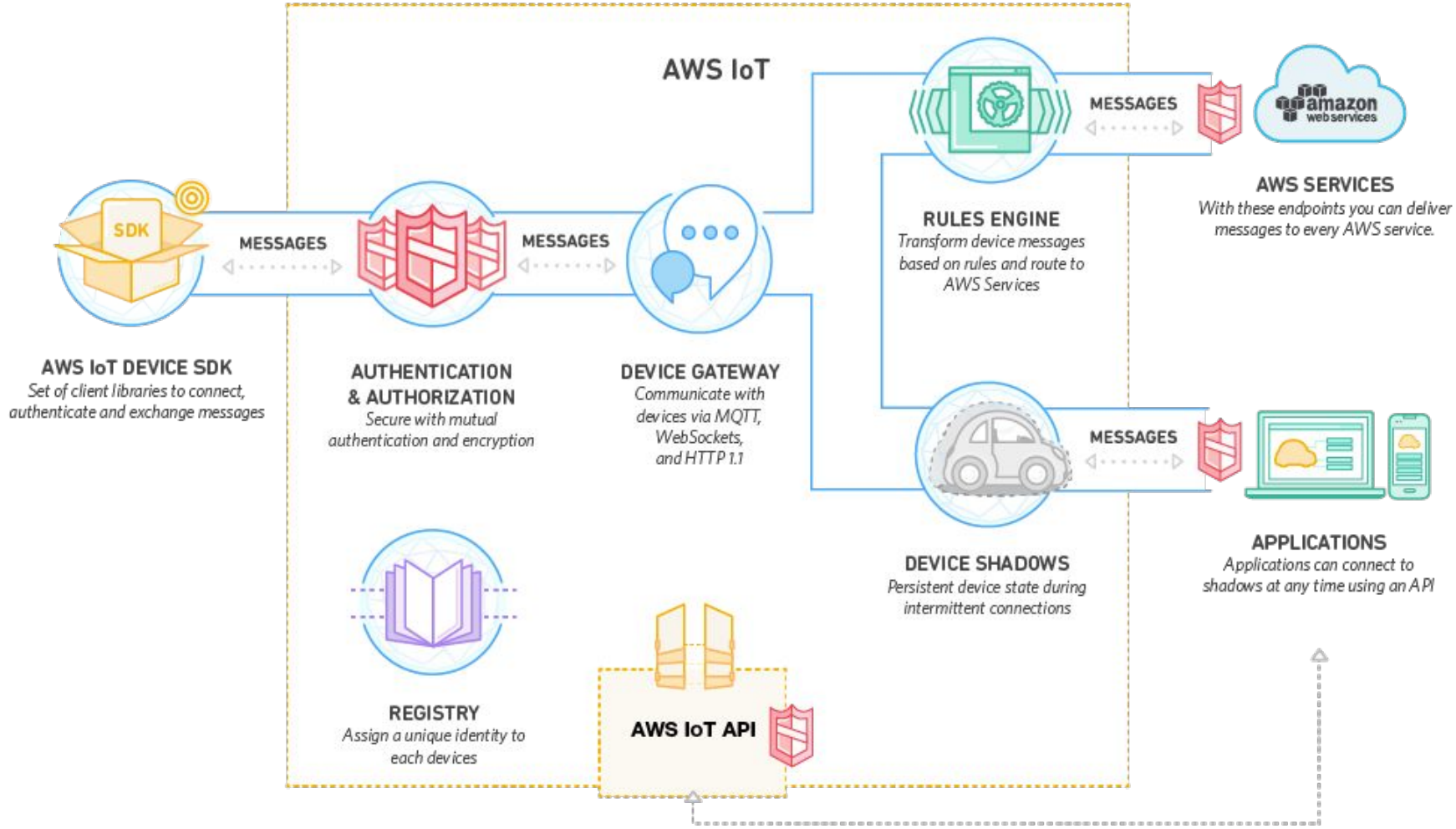
AWS IoT

Secure communication with your devices

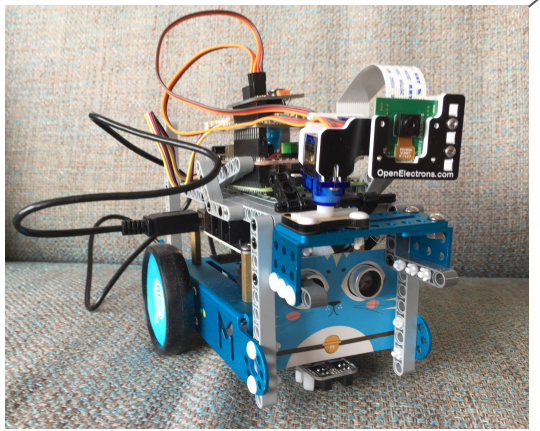
Messaging based on MQTT

Rules engine for routing and transforming messages, and connecting to other Amazon services

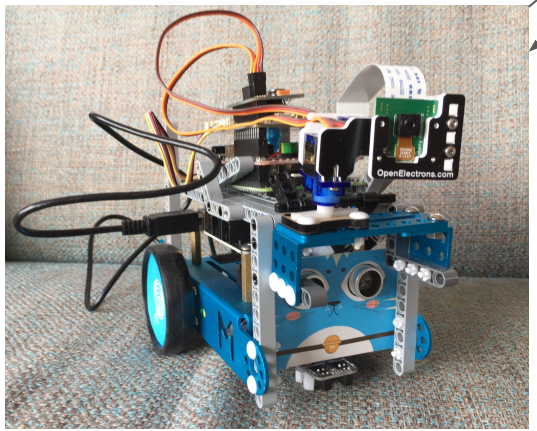
Device Shadow for persisting state and keeping it available when your device is offline


AWS IoT
SDK
MESSAGES
MESSAGES
MESSAGES
MESSAGES
amazon web services
AWS IoT DEVICE SDK
Set of client libraries to connect, authenticate and exchange messages
AUTHENTICATION & AUTHORIZATION
Secure with mutual authentication and encryption
DEVICE GATEWAY
Communicate with devices via MQTT, WebSockets, and HTTP 1.1
RULES ENGINE
Transform device messages based on rules and route to AWS Services
AWS SERVICES
With these endpoints you can deliver messages to every AWS service.
DEVICE SHADOWS
Persistent device state during intermittent connections
APPLICATIONS
Applications can connect to shadows at any time using an API
REGISTRY
Assign a unique identity to each devices
AWS IoT API

Connecting MarsBot to AWS IoT



Connecting MarsBot to AWS IoT



Software: Setting up a connection with AWS IoT

Using Eclipse paho

https://eclipse.org/paho/

Software: Setting up a connection with AWS IoT

```
import paho.mqtt.client as paho
import os
import socket
import ssl
```

Software: Setting up a connection with AWS IoT

```
awshost = "A2BKF6WMC3MQMP.iot.eu-west-1.amazonaws.com"
awsport = 8883
clientId = "marsbot"
thingName = "marsbot"
caPath = "aws-iot-rootCA.crt"
certPath = "cert.pem"
keyPath = "privkey.pem"
```

Software: Setting up a connection with AWS IoT

```
awshost = "A2BKF6WMC3MQMP.iot.eu-west-1.amazonaws.com"
awsport = 8883
clientId = "marsbot"
thingName = "marsbot"
caPath = "aws-iot-rootCA.crt"
certPath = "cert.pem"
keyPath = "privkey.pem"

mqttc = paho.Client()
mqttc.tls_set(caPath, certfile=certPath, keyfile=keyPath, cert_reqs=ssl.CERT_REQUIRED,
tls_version=ssl.PROTOCOL_TLSv1_2, ciphers=None)
mqttc.connect(awshost, awsport, keepalive=60)
```

Software: Subscribing to an MQTT topic

```
mqttc.on_connect = on_connect
mqttc.on_message = on_message
mqttc.loop_forever()
```

Software: Subscribing to an MQTT topic

```
mqttc.on_connect = on_connect
mqttc.on_message = on_message
mqttc.loop_forever()

def on_connect(client, userdata, flags, rc):
    print("Connection returned result: " + str(rc) )
    # Subscribing in on_connect() means that if we lose the connection and
    # reconnect then subscriptions will be renewed.
    client.subscribe("#" , 1 )
```

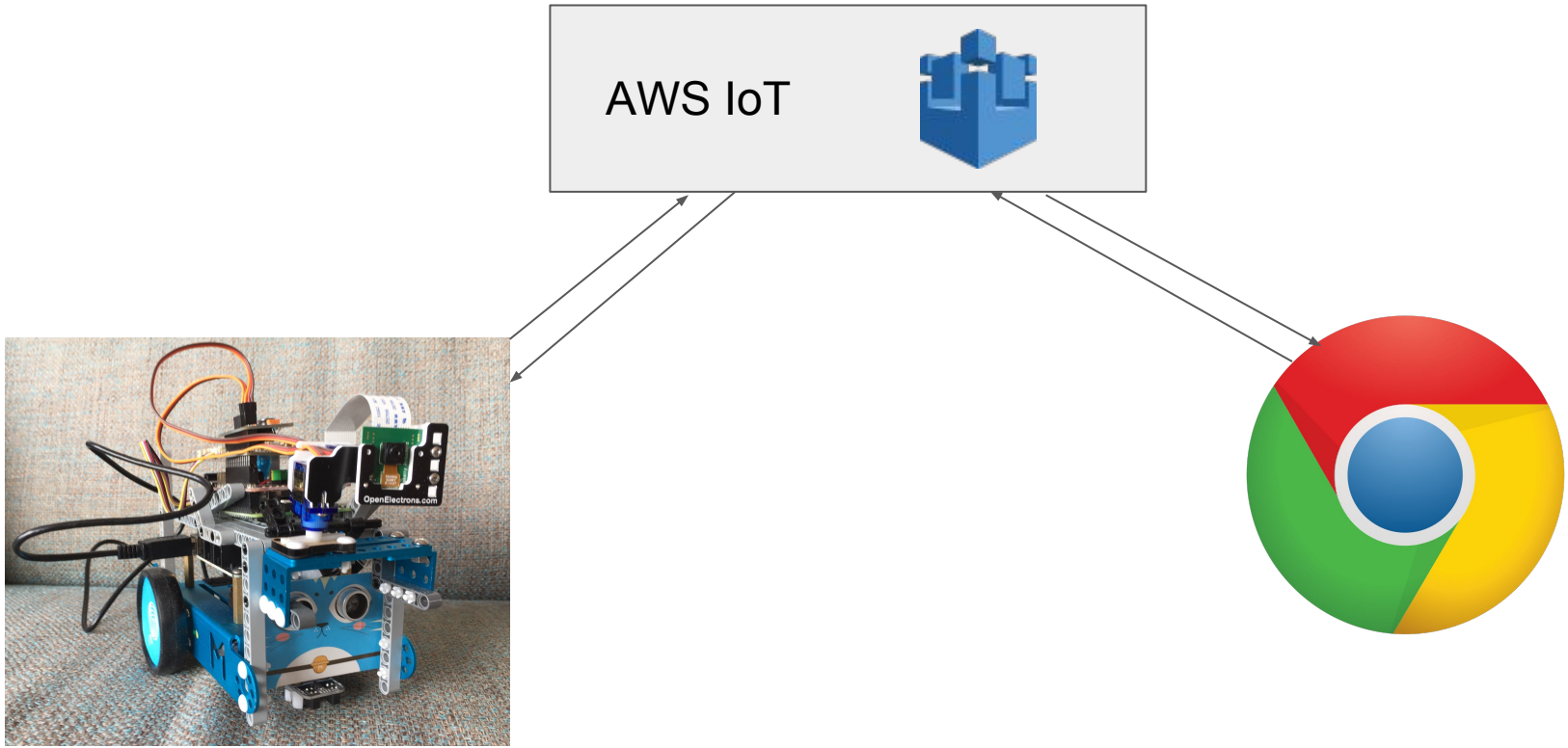
Software: Responding to messages

```
def on_message(client, userdata, msg):
    topic = str(msg.topic);
    command = str(msg.payload);
    print("topic: "+topic)
    print("payload: "+command)
    if topic == 'marsbot/mbot':
        if command == 'fwd':
            print("moving forward")
            forward()
        elif command == 'left':
            ...
```

Software: Publishing data to an MQTT topic

mqttc.publish('topic', payload=mydata, qos=0, retain=False)

Connecting your web client to AWS IoT



Connecting your web client to AWS IoT

Sending and receive MQTT messages

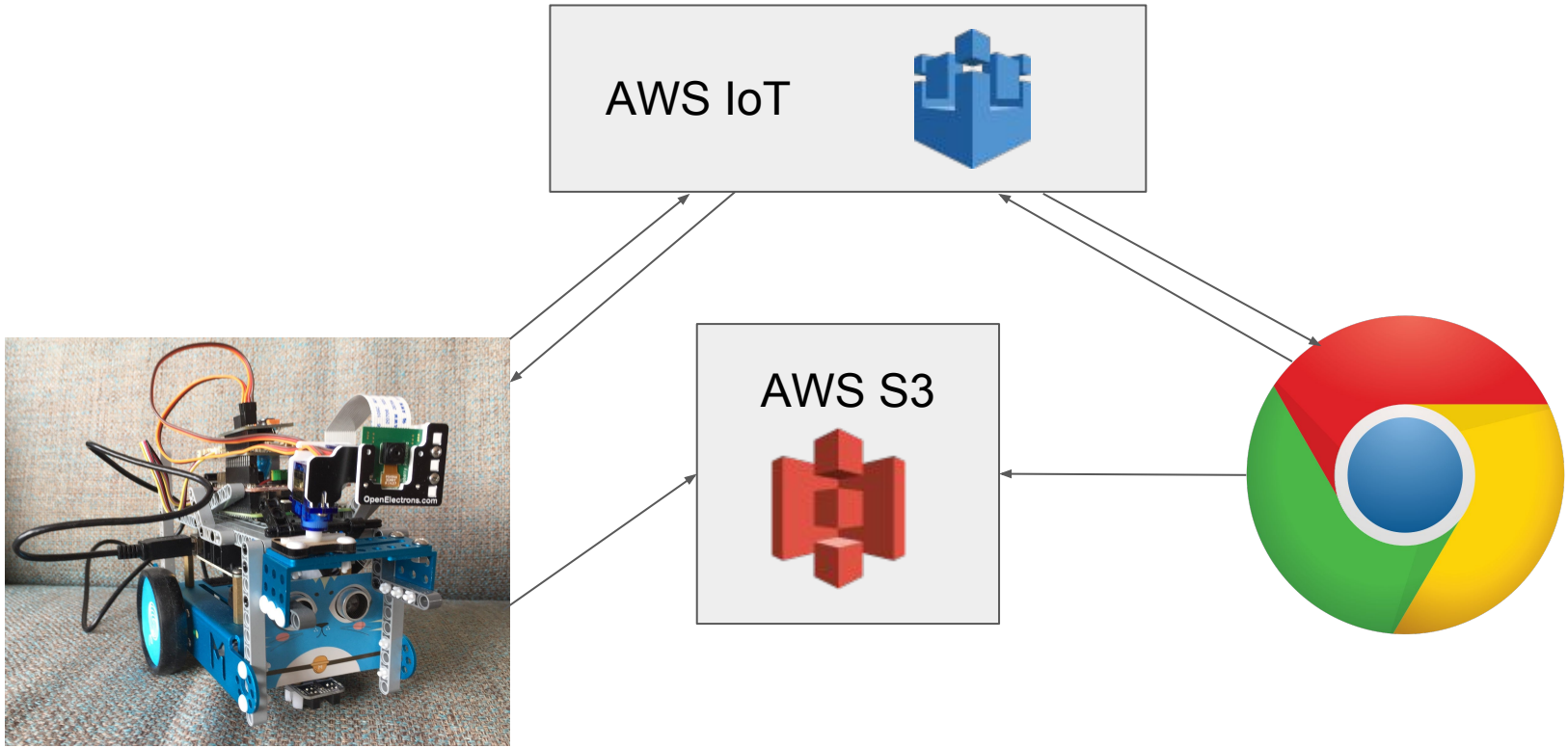
Using Eclipse Paho javascript client

Using Websockets

Javascript

Very similar to the python client...

Sending Pictures to AWS



Software: Sharing an image on S3

```
import boto3
import uuid
```

Software: Sharing an image on S3

```
import boto3
import uuid

camera.capture('marsbot-camera.jpg')
bucket_name = 'marsbot-bucket'
object_key = 'marsbot-camera-{}.jpg'.format(uuid.uuid4())
s3 = boto3.resource('s3')
s3.Bucket(bucket_name).upload_file('marsbot-camera.jpg', object_key)
url = s3client.generate_presigned_url('get_object', {'Bucket': bucket_name, 'Key': object_key})
mqttc.publish('marsbot/camera/reply', payload=url, qos=0, retain=False)
```

Demo time!

Rules engine

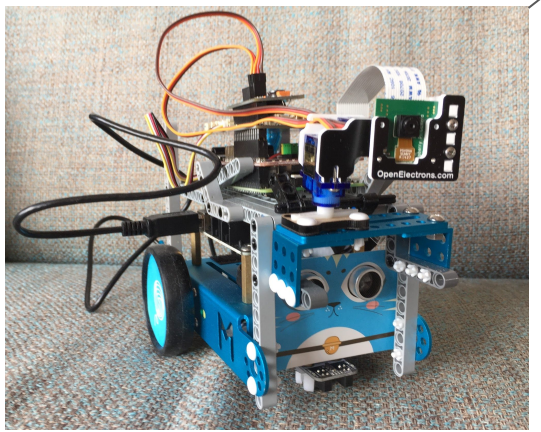
SQL-like syntax for filtering messages

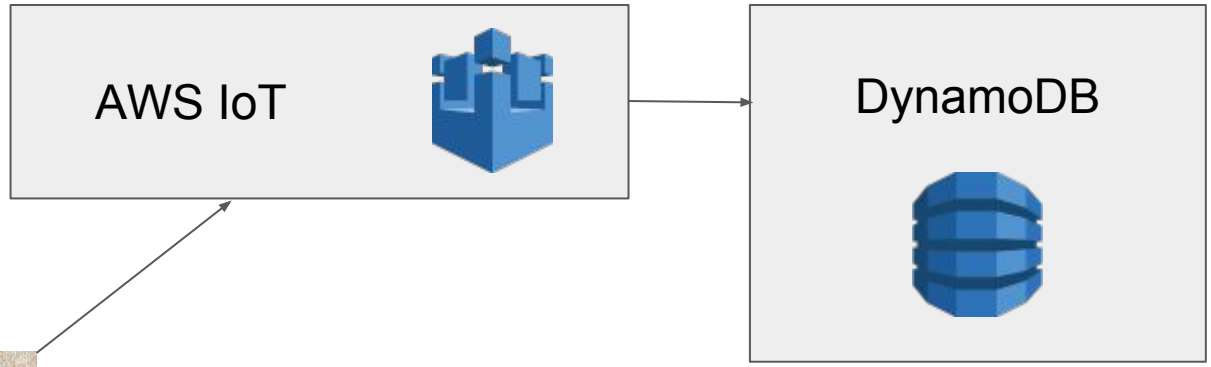
```
SELECT * FROM 'marsbot/sensor/temp' WHERE temp > 30
```

Connect to other services

cloudwatchAlarm to change a CloudWatch alarm.
cloudwatchMetric to capture a CloudWatch metric.
dynamoDB to write data to a DynamoDB database.
elasticsearch to write data to a Amazon Elasticsearch Service domain.
kinesis to write data to a Amazon Kinesis stream.
lambda to invoke a Lambda function.
s3 to write data to a Amazon S3 bucket.
sns to write data as a push notification.
firehose to write data to an Amazon Kinesis Firehose stream.
sqs to write data to an SQS queue.
republish to republish the message on another MQTT topic.

Rules engine example - Connecting to DynamoDB





Rules engine example - Connecting to DynamoDB

```
{
    "rule": {
        "ruleDisabled": false,
        "sql": "SELECT * AS message FROM 'marsbot/sensor/temp'",
        "description": "rule for dynamoDB",
        "actions": [{
            "dynamoDB": {
                "hashKeyField": "key",
                "roleArn": "arn:aws:iam::123456789012:role/aws_iot_dynamoDB",
                 "tableName": "my_ddb_table",
                 "hashKeyValue": "${topic()}",
                 "rangeKeyValue": "${timestamp()}",
                 "rangeKeyField": "timestamp"
             }
        }]
    }
}
```

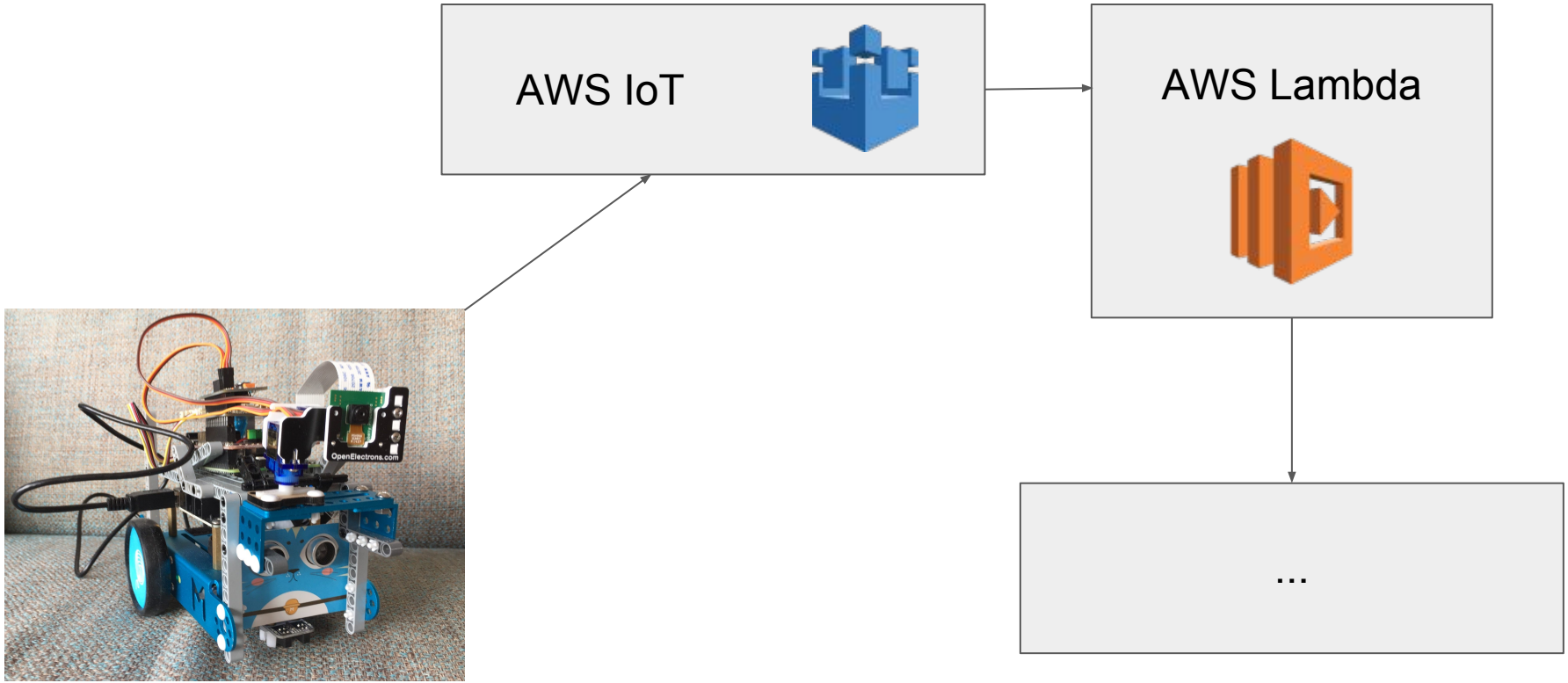
Rules engine example - Connecting to Lambda

Execute code directly on AWS infrastructure

No need to manage your own servers or environments

Java, Python, NodeJS

Rules engine example - Connecting to Lambda

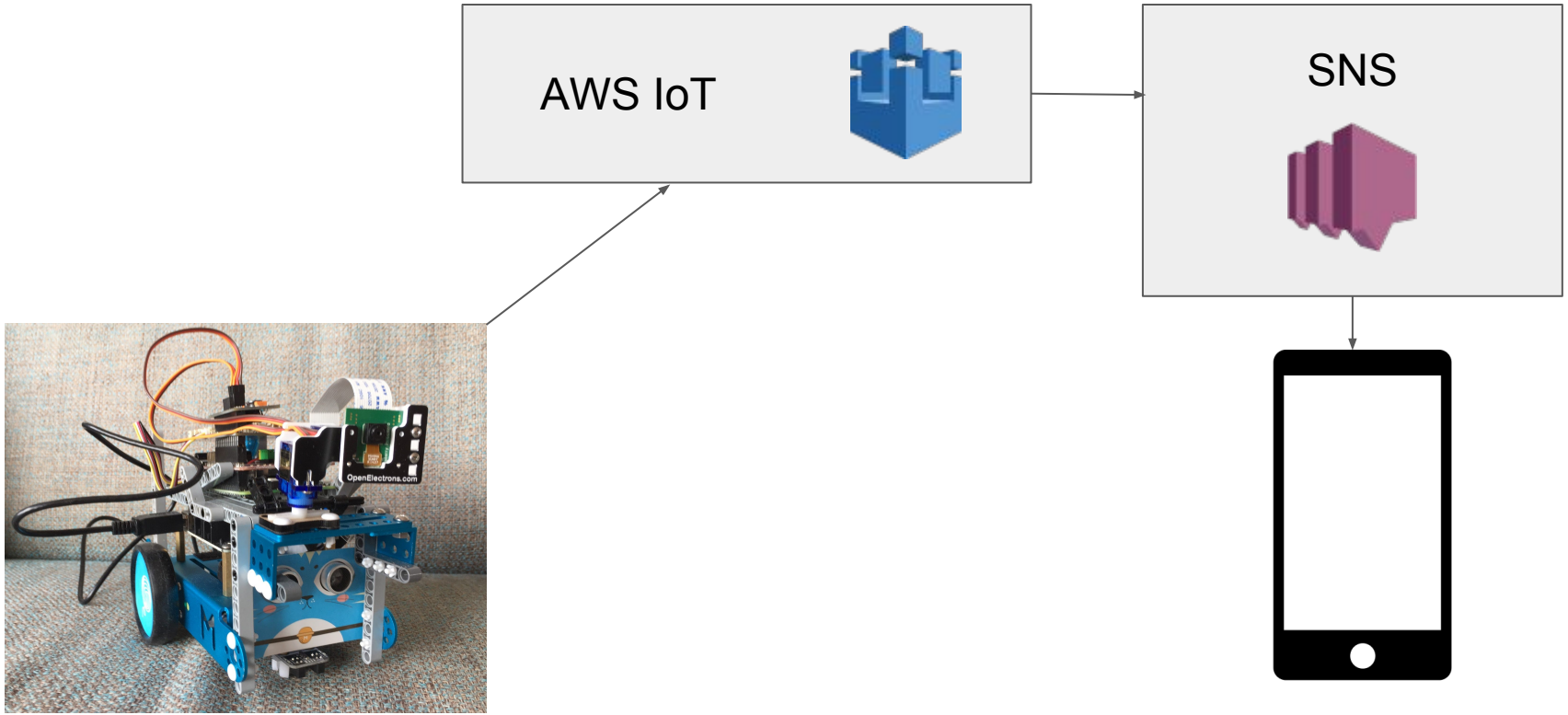


Rules engine example - Connecting to AWS SNS

Send small messages to:

- HTTP endpoints
- Mobile phone as SMS
- Email
- AWS Lambda

Rules engine example - Connecting to AWS SNS



Recap

Robots are cool :-)

mBot is a great platform to start with

A Raspberry Pi has all the capabilities you need

Writing Python code is easy, grabbing it from internet is even more easy

Recap

Amazon's IoT platform enables you to get started with IoT without running your own server

MQTT is a lightweight messaging framework, ideal for IoT applications

Using the rules engine, you can easily connect to other Amazon services

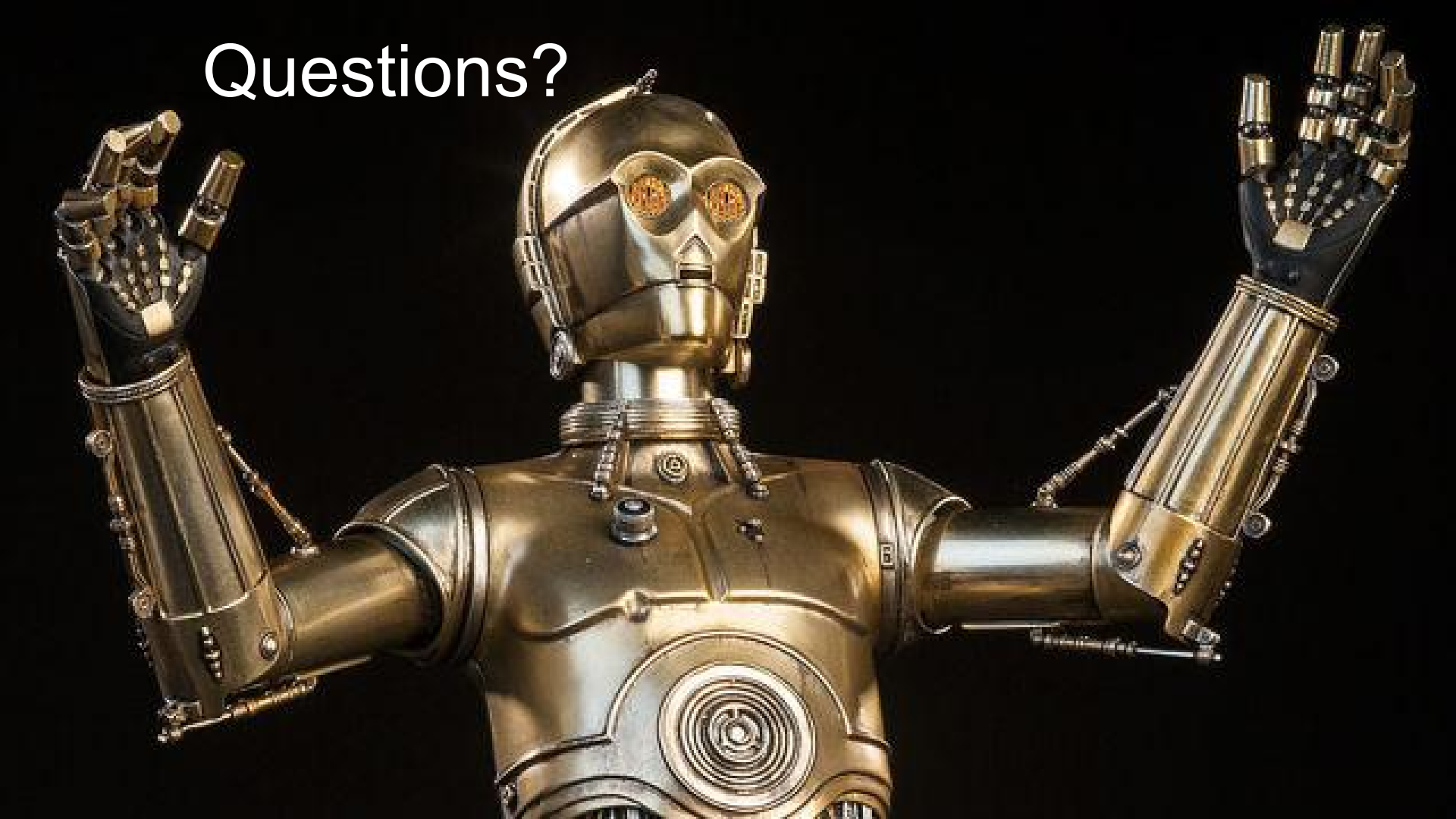
Finally

Twitter: **@JeroenResoort**

Blog: **http://blog.jdriven.com/author/jeroen-resoort/**

See my blog post for useful links and a shopping list
http://blog.jdriven.com/2016/04/mission-mars/

Questions?



MISSION TO MARS:
EXPLORING NEW WORLDS WITH AWS IOT

Jeroen Resoort
@JeroenResoort @jdriven_nl