THE INTERNET OF THINGS

AND CAPTURING

TIME SERIES DATA

Marco Pas / @marcopas

GOAL

Learn how to **CONNECT** IoT devices,
COLLECT and **VISUALIZE** Time Series Data.

AGENDA

- The **Internet Of Things**
- How to **connect** your Things
- **Getting data** from your Things
- **Visualize the data** from your Things

THE INTERNET OF THINGS

Small Things, Big Things and Stupid Things

DEFINITION

*The **network of physical devices** and **connectivity** which enables these objects to connect and **exchange data***

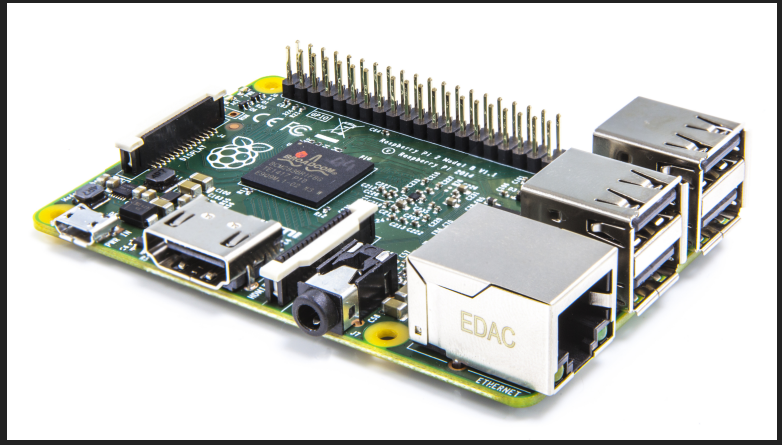
TYPICAL IOT DEVICE

Software

- Applications
- Connectivity
- Operating System

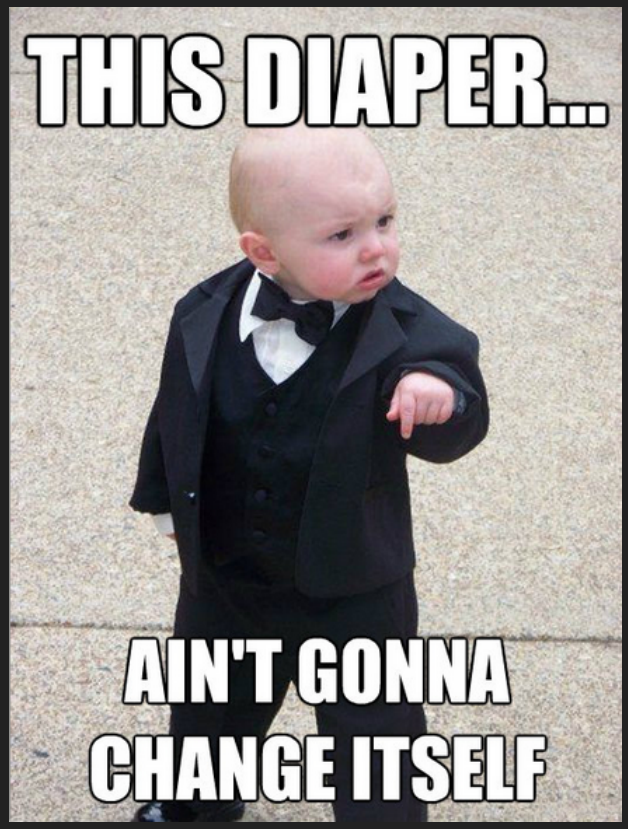
Hardware

- Device itself
- Sensors
- Connectivity modules



SOME REMARKABLE EXAMPLES

- Smart diapers
- Water bottles
- Egg minder
- Connected Spoons
- Toilet rings
- Wearable rings
- Toasters



SOME COOL EXAMPLES


AMBIENT EXPERIENCE
• Smart Home

WEARABLES

- Health Monitoring
- Assistive Technology


SMART FARMING
• Precision Farming
• Livestock Monitoring

IOT IS HARD ⚠

IoT Projects have a 75% Failure Rate

- Long completion times
- Poor quality of the data collected
- IoT integration
- Budget overruns
- Data privacy / Security


I TALK, LISTEN & LEARN!
PLUS AUTOMATIC CONTENT UPDATES
WE'LL NEVER RUN OUT OF FUN THINGS TO DO!
(SERIOUSLY.)
3-8 YEARS
Let's go on an adventure!
BEAR
2000 mAh
Fisher-Price
SMART TOY®


GROCERIES
by MasterCard
GROCERIES
by MasterCard
Search for Groceries...
freshdirect
GO SHOPPING
ShopRite
GO SHOPPING
RECENTLY BOUGHT
FreshDirect Fresh Brioche Hamburger Rolls
$5.49
FreshDirect Oven-Ready Sprouted Multigrain Baguette
$2.99
Green Seedless Grapes
$4.99/lb
Add 1 to Cart
Add 1 to Cart
Add 1 to Cart
1 item $2.79


DDOS
ATTACK
EA6
C2B
8C1
05E
060
811
8C6
B8B
B48
186
2A3
CF4
C4B
C2B
8C6
F05

When you start playing around with IOT devices do not forget about **security and data privacy**.

HOW TO CONNECT YOUR THINGS

Natively & using AWS IOT

CONNECTIVITY REQUIREMENTS

- **Lightweight** and Bandwidth Efficient
- **Simple** to implement
- **Data Agnostic**
- Continuous **Session Awareness**
- Support **Quality of Service**

CONNECTIVITY CANDIDATES

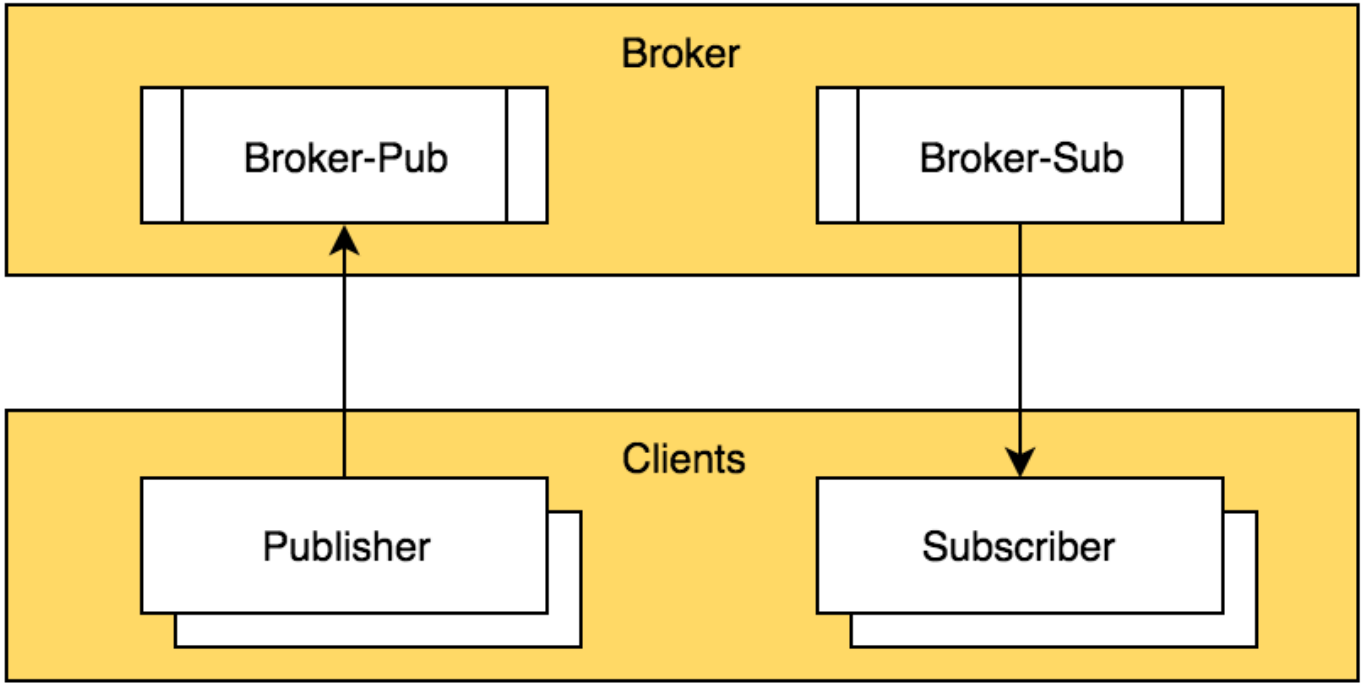
- REST/GRPC
- AMQP
- XMPP
- STOMP
- MQTT
- CoAP

MQ TELEMETRY TRANSPORT

MQTT is a Client Server
publish/subscribe messaging
transport protocol.

Standardized under OASIS.

MQTT PARTS


Broker
Broker-Pub
Broker-Sub
Clients
Publisher
Subscriber

MQTT TOOLING

Brokers

- **Mosquitto**
- HiveMQ
- ActiveMQ
- RabbitMQ
- emqttd
- **AWS IOT**
- ...

Clients

- **Paho**
- **Spring Integration**
- ...

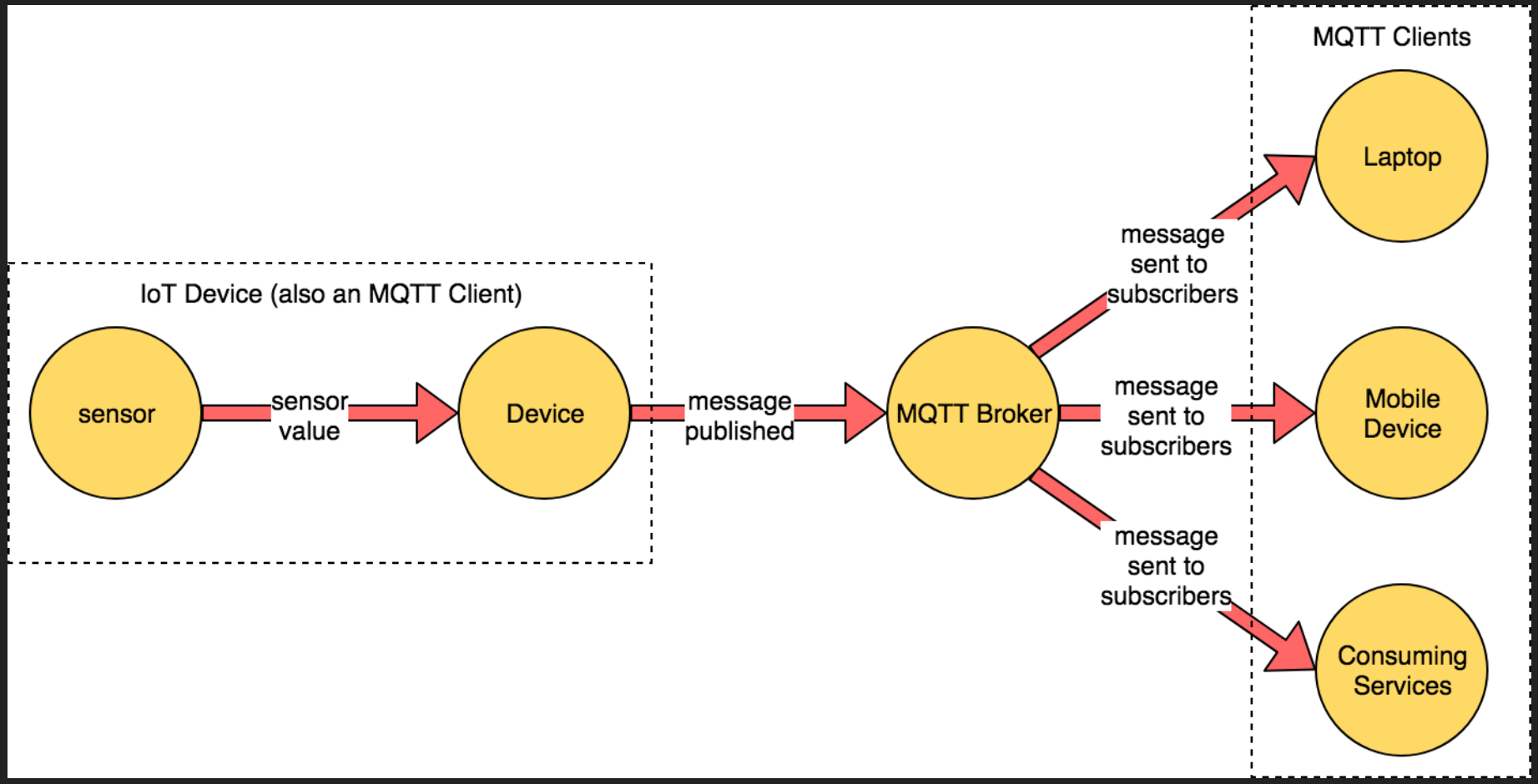
Tools

- MQTT.fx
- MyMQTT
- MQTT Lens
- **MQTTBox**
- ...

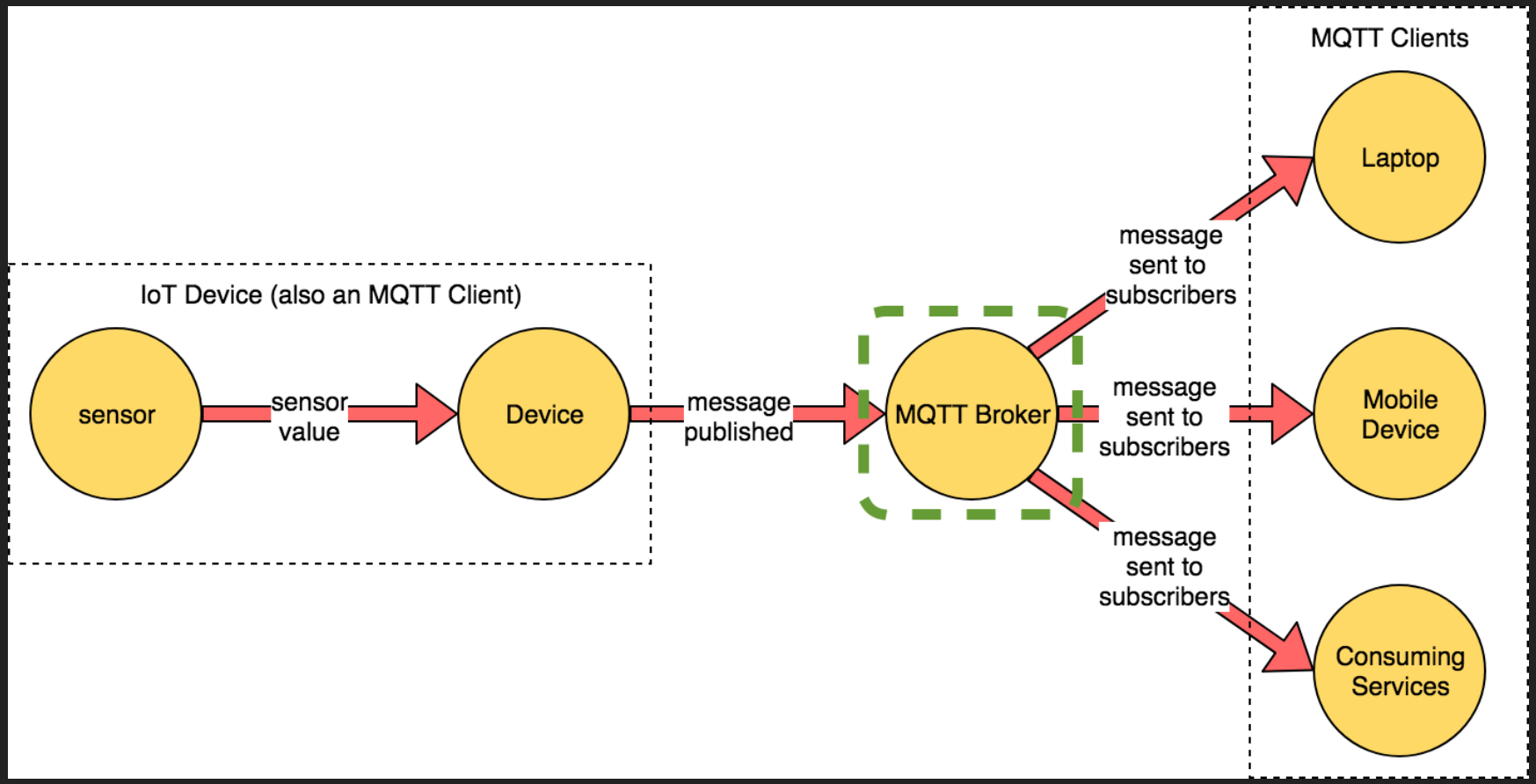
IOT & MQTT

NATIVELY

DEMO - OVERVIEW



DEMO - RUN BROKER & CONNECT CLIENT



```
// file: docker-compose.yml
version: "3"
services:

  mosquitto:
    image: eclipse-mosquitto:1.4.12
    container_name: mosquitto
    ports:
      - 1883:1883       # MQTT port
      - 9001:9001       # MQTT websocket port
    volumes:
      - $PWD/../../mosquitto/config/mosquitto.conf:/mosquitto/config/mosquitto.conf
      - $PWD/../../mosquitto/data:/mosquitto/data
```

DEMO

RUN BROKER & CONNECT CLIENT

MQTT CONNECT

What	Description
ClientId	Unique identifier of each client
Username/Password	Authenticating/Authorization
Last Will Message	Notify other clients, when a client disconnects ungracefully

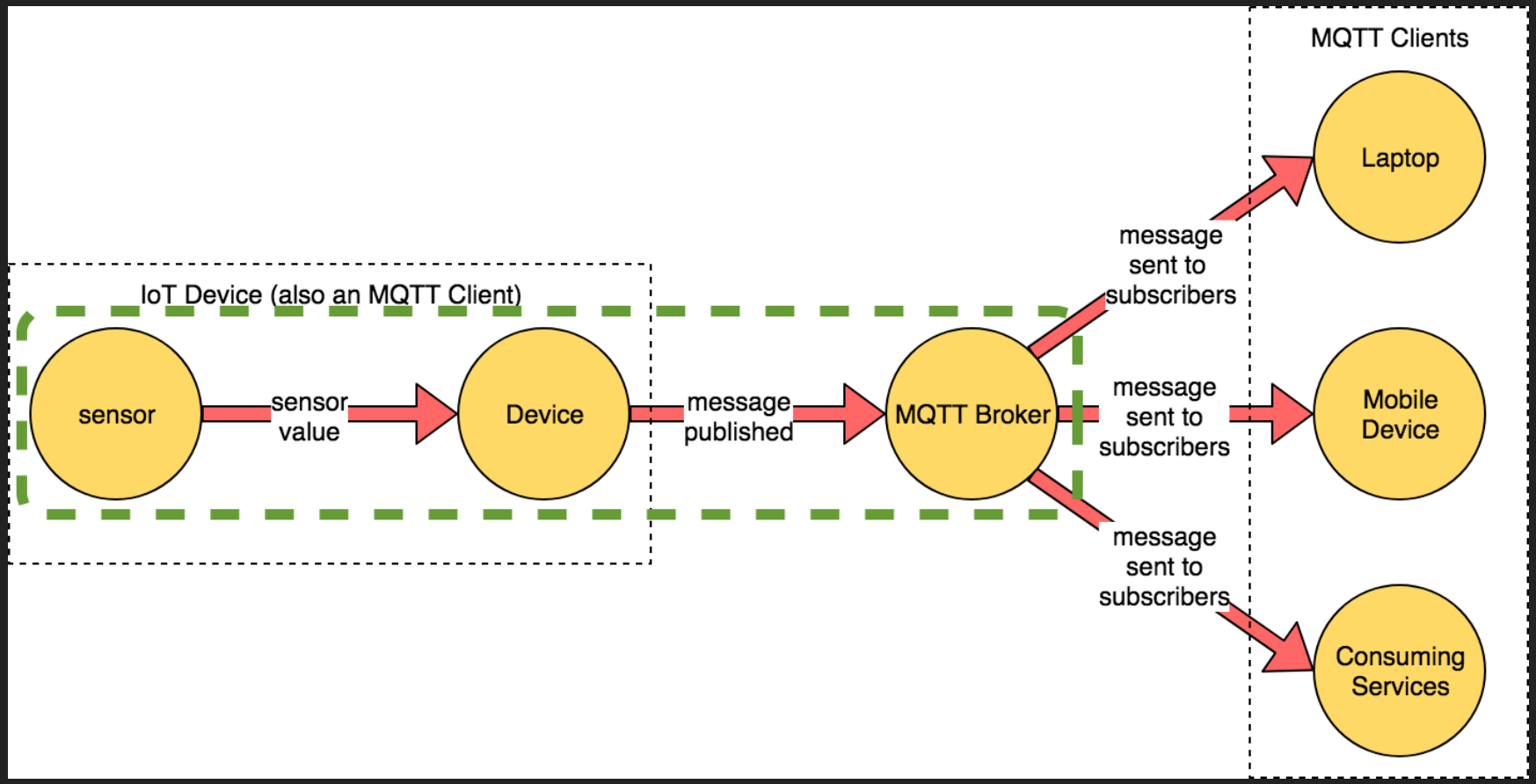
MQTT CONNECT EXAMPLE

```
String clientId = MqttClient.generateClientId();
MqttClient client = new MqttClient("tcp://localhost:1883", clientId);

MqttConnectOptions connOpts = new MqttConnectOptions();
connOpts.setUserName("foo");
connOpts.setPassword("bar");

client.connect(connOpts);
```

DEMO - PUBLISH & SUBSCRIBE



DEMO - PUBLISH & SUBSCRIBE

```
// Publish a message
$ mosquitto_pub -t "myhome/livingroom/temperature" -m '0.1' # publish a message
$ mosquitto_pub -t "myhome/livingroom/temperature" -l       # publish by line

// Start a subscriber
$ mosquitto_sub -v -t "myhome/livingroom/temperature"
$ mosquitto_sub -v -t "myhome/+/temperature"               # using + wildcard
$ mosquitto_sub -v -t "myhome/#"                           # using # wildcards
```

DEMO

PUBLISH & SUBSCRIBE

MQTT MESSAGE

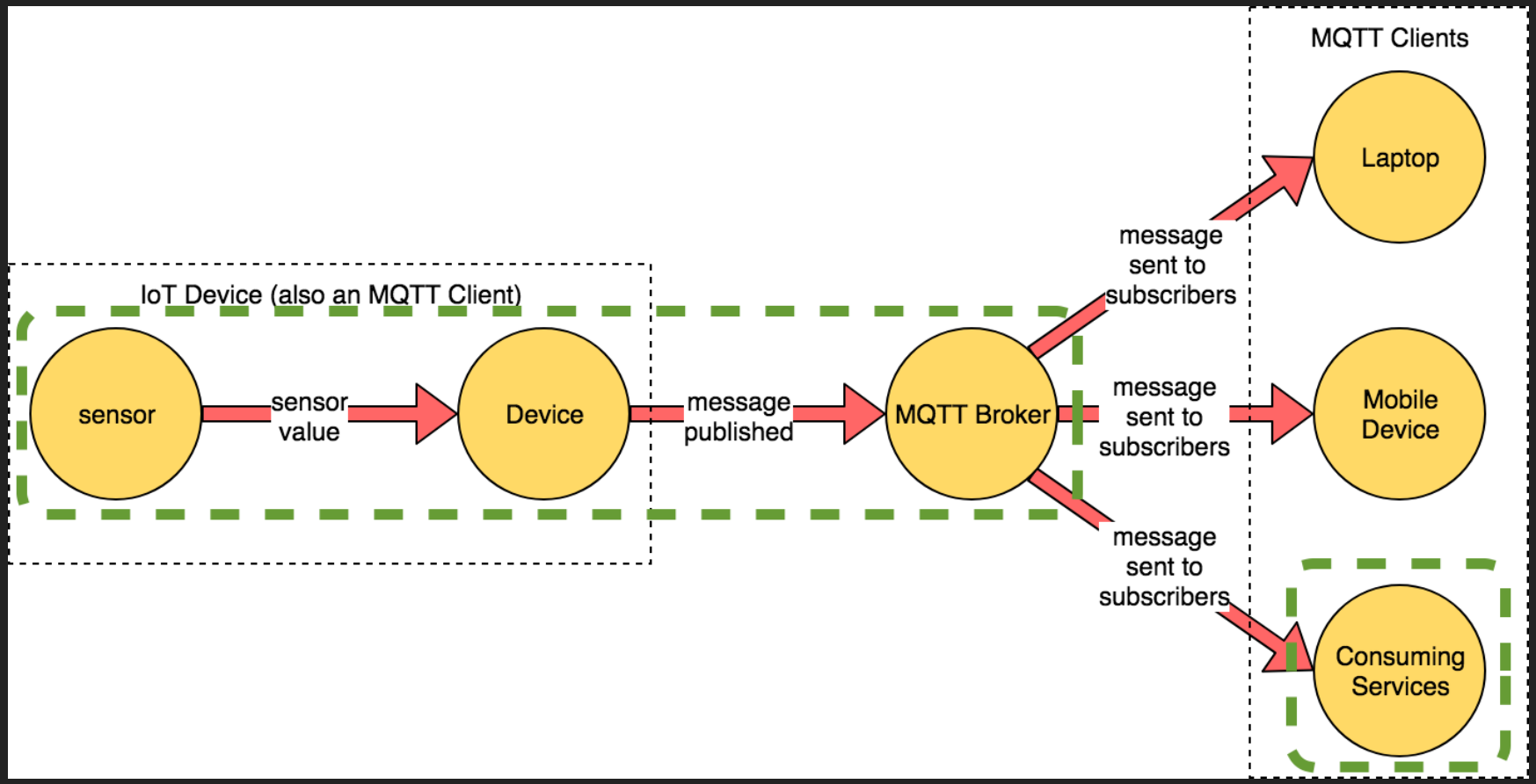
What	Description
Payload	Data agnostic payload (images, texts, any binary data)
Topicname	A simple string, hierarchically structured
Retainflag	Retain last message if no subscribers
QoS	The quality level of this message (0/1/2)

MQTT PUBLISH EXAMPLE

```
String clientId = MqttClient.generateClientId();
MqttClient client = new MqttClient("tcp://localhost:1883", clientId);
MqttConnectOptions connOpts = new MqttConnectOptions();
connOpts.setUserName("foo");
connOpts.setPassword("bar");
client.connect(connOpts);

// publish
MqttMessage message = new MqttMessage();
message.setPayload("Hello world from Java".getBytes());
client.publish("iot_data", message);
client.disconnect();
```

DEMO - SPRINGBOOT/GRAILS AND MQTT

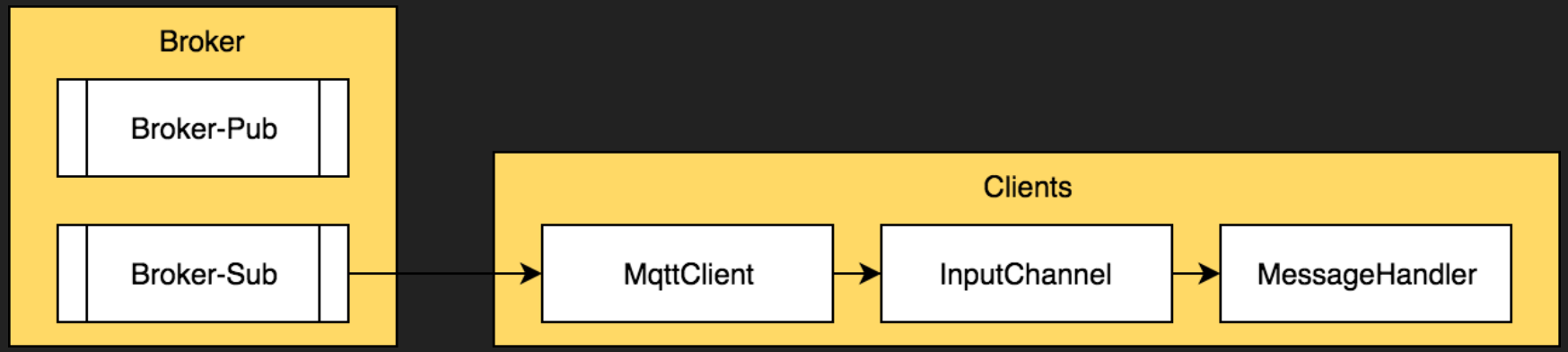


SPRINGBOOT/GRAILS AND MQTT

It is just as simple as adding dependencies on:

```
// file:build.gradle
compile "org.springframework.boot:spring-boot-starter-integration"
compile "org.springframework.integration:spring-integration-mqtt"
```

SPRING INTEGRATION AND MQTT



MQTT CLIENT

```
// MqttClient
@Bean
DefaultMqttPahoClientFactory mqttClientFactory() {
    DefaultMqttPahoClientFactory factory = new DefaultMqttPahoClientFactory()
    defaultMqttPahoClientFactory.setServerURIs("tcp://localhost:1883")
    return factory
}
```

INPUTCHANNEL

```
// InputChannel
@Bean
MessageChannel mqttInputChannel() {
    return new DirectChannel()
}

@Bean
MessageProducerSupport mqttInbound() {
    MqttPahoMessageDrivenChannelAdapter adapter =
        new MqttPahoMessageDrivenChannelAdapter(
            UUID.randomUUID().toString(),
            mqttClientFactory(),
            "myhome/livingroom/temperature"         // subscribe to topic
    );
    // some code intentionally omitted
    adapter.setOutputChannel(mqttInputChannel())    // send to channel
    return adapter
}
```

MESSAGEHANDLER

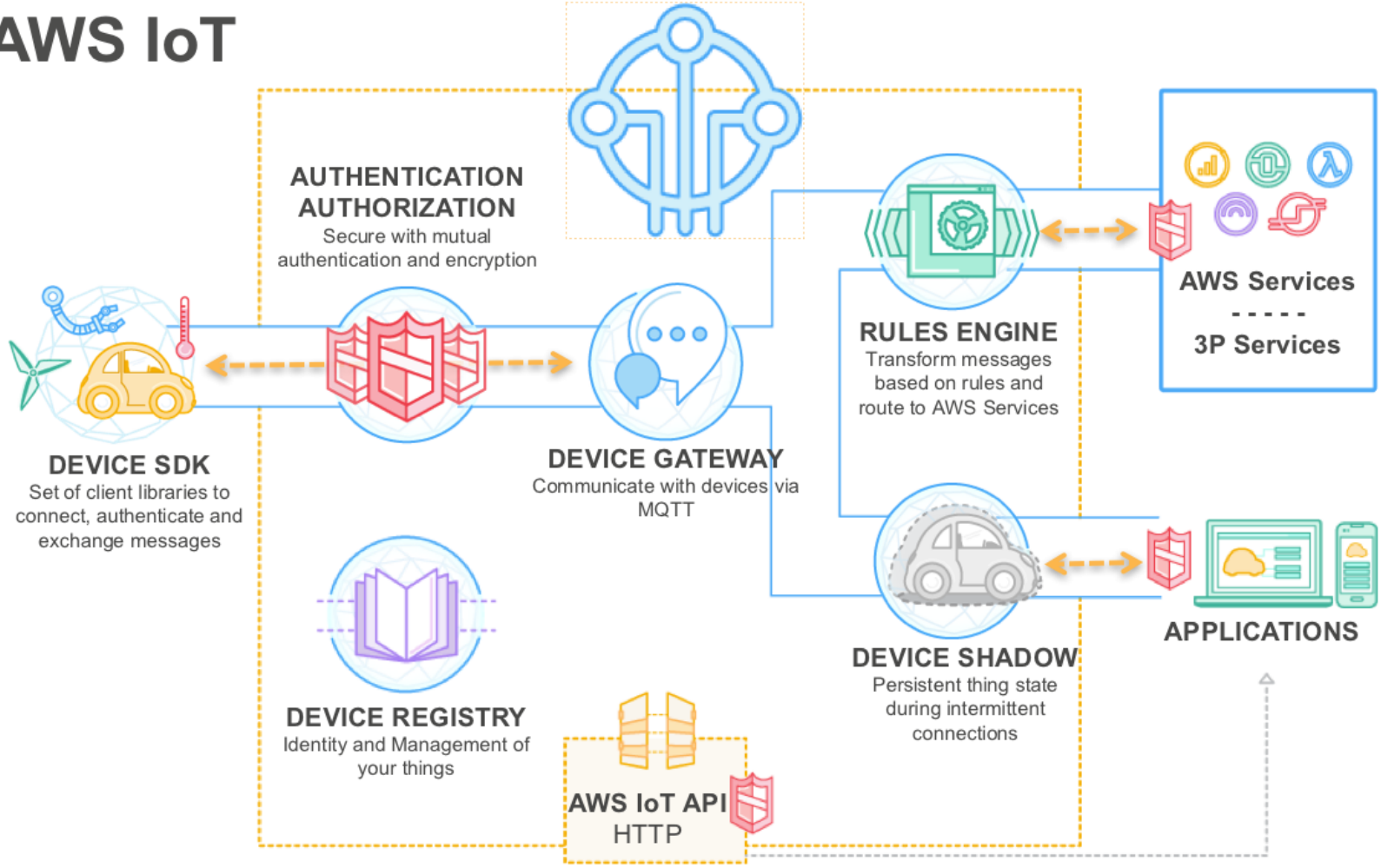
```
// MessageHandler
@Bean
@ServiceActivator(inputChannel = "mqttInputChannel")
MessageHandler stringHandler() {
    return new MessageHandler() {
        @Override
        void handleMessage(Message<?> message) throws MessagingException {
            println message.payload.toString()
        }
    }
}
```

DEMO

SPRINGBOOT/GRAILS AND MQTT

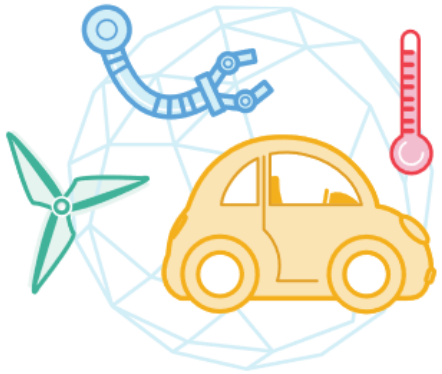
AMAZON AWS IOT

AWS IoT


AUTHENTICATION
AUTHORIZATION
Secure with mutual authentication and encryption
RULES ENGINE
Transform messages based on rules and route to AWS Services
AWS Services
- - - - -
3P Services
DEVICE SDK
Set of client libraries to connect, authenticate and exchange messages
DEVICE GATEWAY
Communicate with devices via MQTT
DEVICE SHADOW
Persistent thing state during intermittent connections
APPLICATIONS
DEVICE REGISTRY
Identity and Management of your things
AWS IoT API
HTTP

AWS IoT

- Monitor
- Onboard
- **Manage**
  - **Things**
  - Types
- Greengrass
- Secure
- Act
- Test
- Software
- Settings
- Learn



You don't have any things yet

A thing is the representation of a device in the cloud.

Learn more | **Register a thing**

AWS IOT - RULES

Select an action

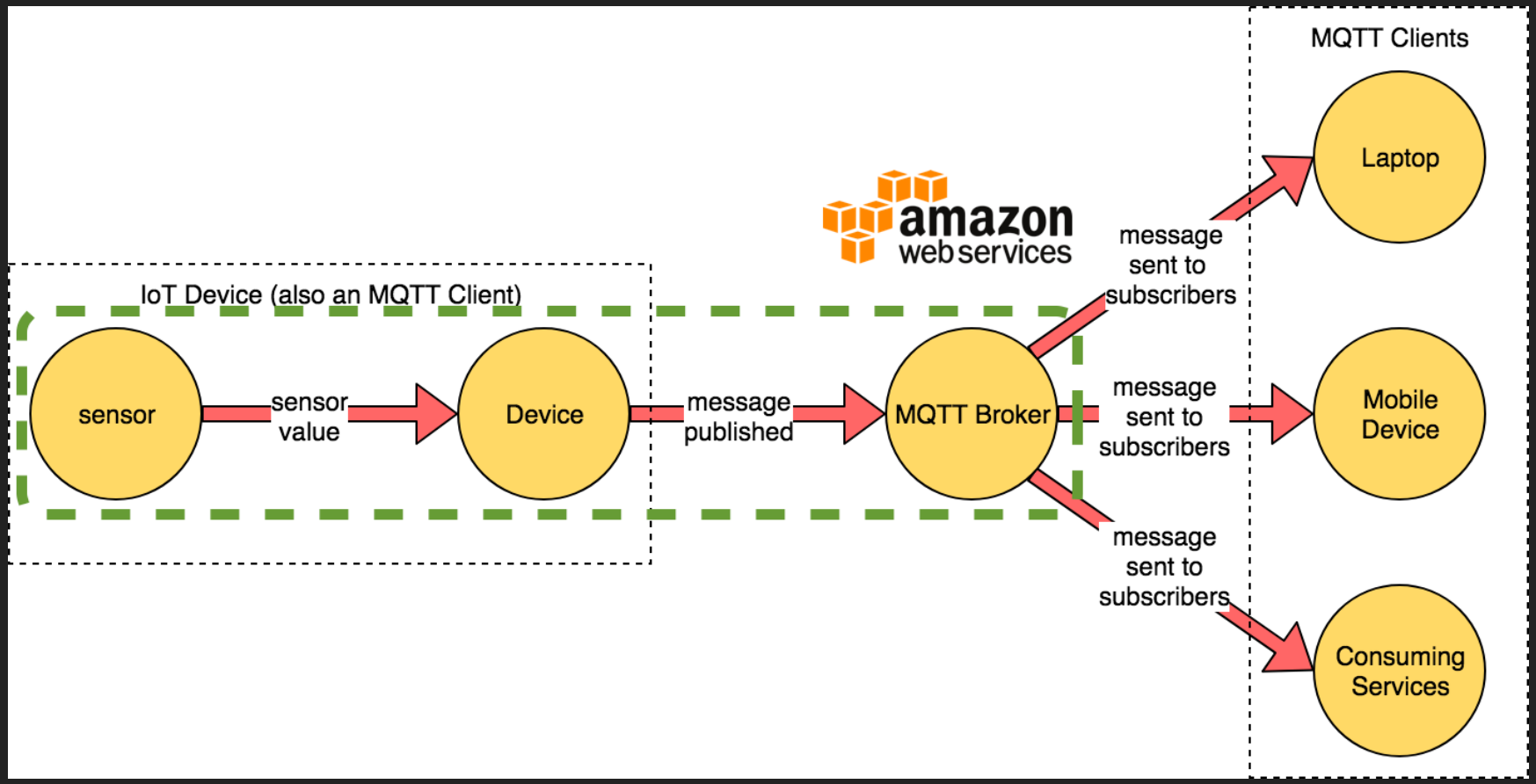
Select an action.

-  Insert a message into a DynamoDB table
  DYNAMODB
-  Split message into multiple columns of a database table (DynamoDBv2)
  DYNAMODBV2
-  Invoke a Lambda function passing the message data
  LAMBDA
-  Send a message as an SNS push notification
  SNS
-  Send a message to an SQS queue
  SQS
-  Sends messages to an Amazon Kinesis Stream
  AMAZON KINESIS

AWS IOT OFFERINGS

- IoT Core
- Device Management
- GreenGrass
- IoT Analytics
- Amazon FreeRTOS
- IoT 1-Click
- IoT Button

DEMO - AWS IOT - PUBLISH & SUBSCRIBE



CONNECT TO AWS IOT

Requirements

- **Register your thing** inside AWS IoT
- Generate certificate for your thing
- Attach policy to the certificate

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:*",
      "Resource": "arn:aws:iot:<region>:<arnId>:*"
    }
  ]
}
```

AWS IOT - PUBLISH & SUBSCRIBE

Publish

```
mosquitto_pub --cafile aws-iot-rootCA.pem --cert <device-certificate>.pem.crt
  --key <private-key>.pem.key  -h <aws-iot-endpoint> -p 8883
  -t <topicName> -m "Hello from Mosquitto"
```

Subscribe

```
mosquitto_sub --cafile aws-iot-rootCA.pem --cert <device-certificate>.pem.crt
  --key <private-key>.pem.key  -h <aws-iot-endpoint> -p 8883
  -t <topicName>
```

DEMO

AWS IOT - PUBLISH & SUBSCRIBE

GETTING DATA FROM YOUR THINGS

Large amounts of time stamped data

TIME STAMPED DATA

Applications rely on a form of data that

measures how things change over time.

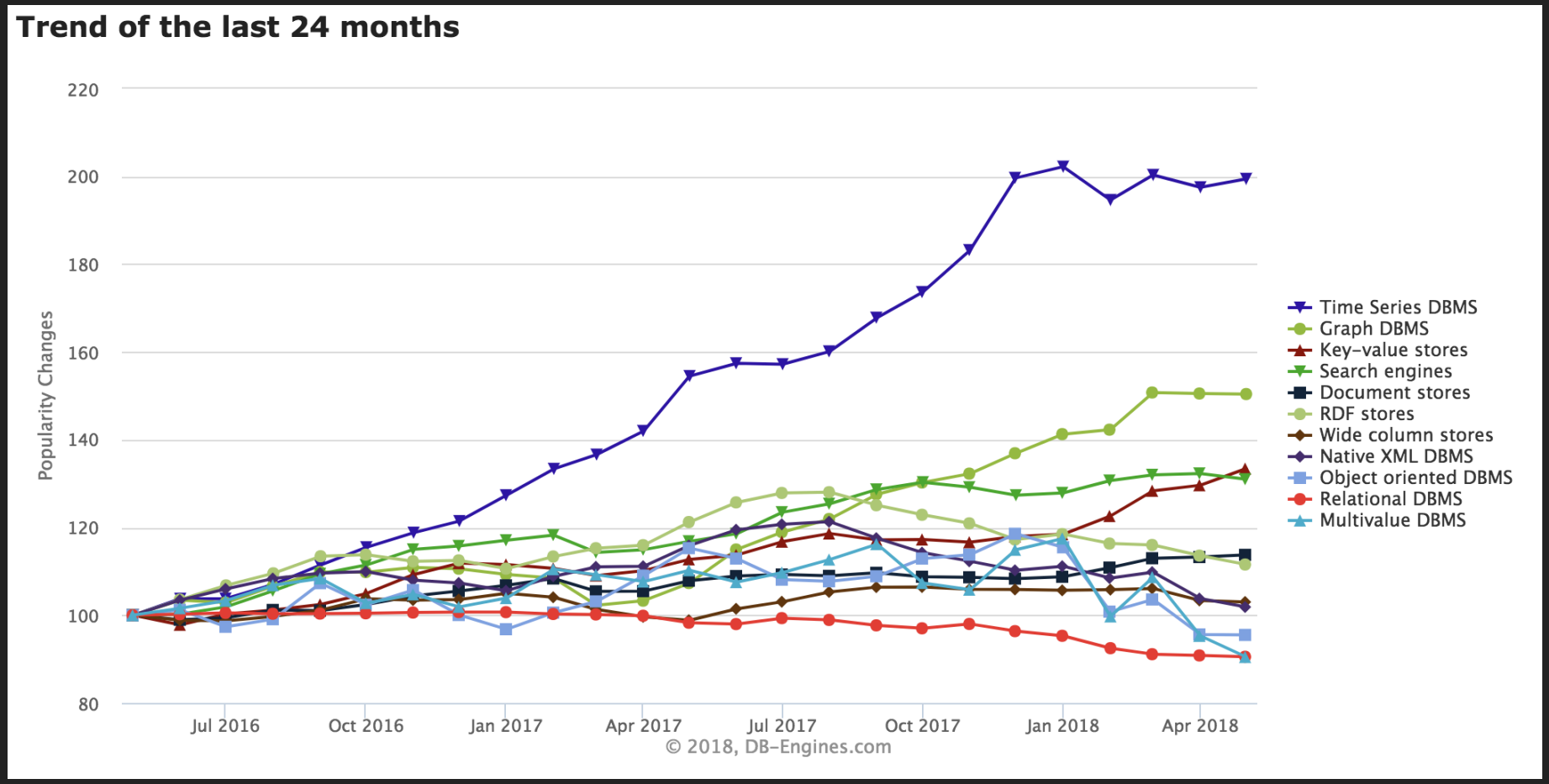
Where time isn’t just a metric, but a primary axis!

TIME SERIES DATA

A time-series is a sequence of data points consisting of successive measurements made **over a time interval**

[timestamp] [metadata/tags]
[fields+values]

DATABASE TRENDS



TIME SERIES DATABASE

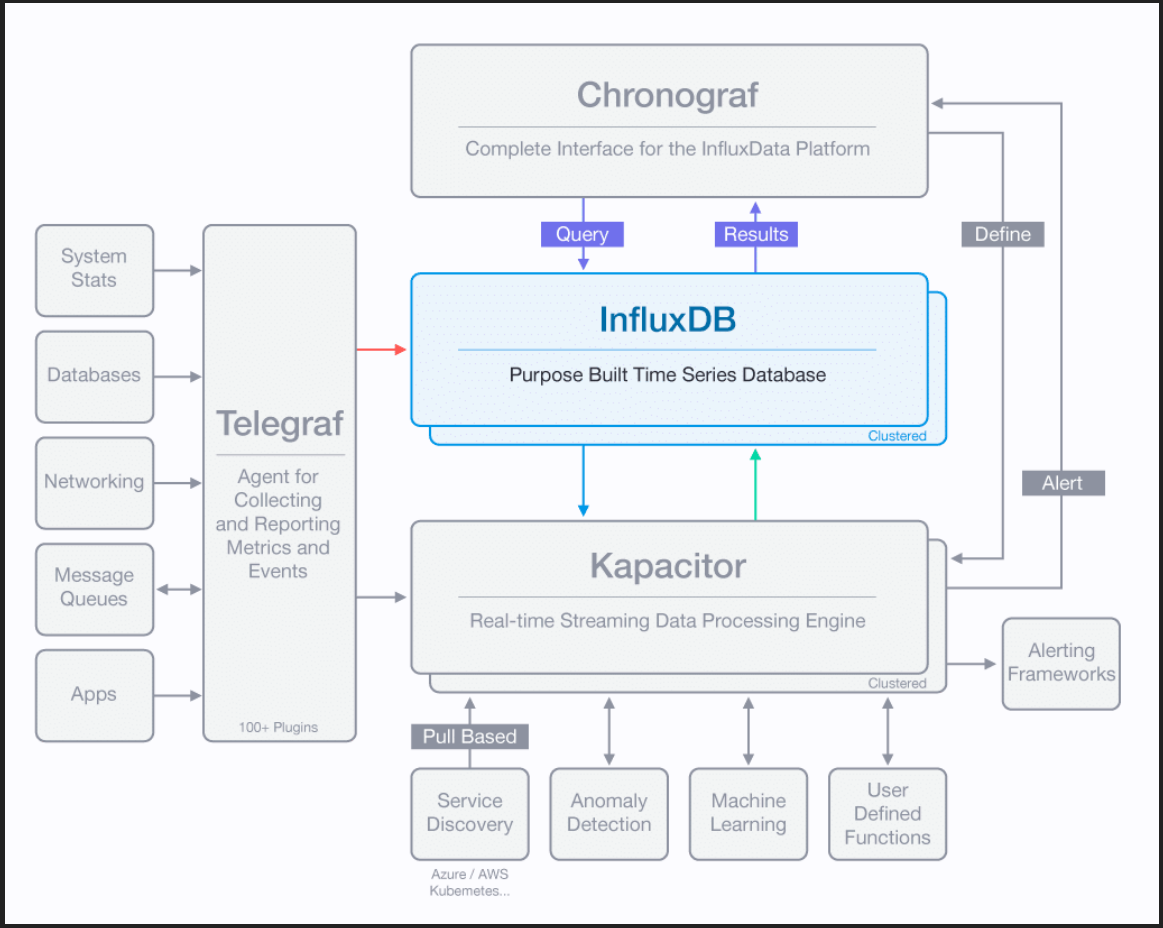
- Scale
- Usability

TIMESERIES DATABASE RANKING

23 systems in ranking, May 2018

Rank May 2018	Rank Apr 2018	Rank May 2017	DBMS	Database Model	Score May 2018	Score Apr 2018	Score May 2017
1.	1.	1.	InfluxDB	Time Series DBMS	11.00	+0.24	+3.05
2.	2.	↑ 5.	Kdb+	Multi-model	3.07	-0.01	+1.50
3.	3.	↓ 2.	RRDtool	Time Series DBMS	2.68	-0.07	-0.34
4.	4.	↓ 3.	Graphite	Time Series DBMS	2.26	+0.07	+0.25
5.	5.	↓ 4.	OpenTSDB	Time Series DBMS	1.62	-0.08	-0.05
6.	↑ 7.	↑ 8.	Prometheus	Time Series DBMS	1.12	+0.07	+0.64
7.	↓ 6.	↓ 6.	Druid	Time Series DBMS	1.01	-0.05	+0.07
8.	8.	↓ 7.	KairosDB	Time Series DBMS	0.43	-0.01	-0.08
9.	9.	9.	eXtremeDB	Multi-model	0.31	-0.01	-0.02
10.	10.	↑ 11.	Riak TS	Time Series DBMS	0.27	-0.00	+0.06

INFLUXDATA PRODUCTS



INTRODUCING

- Open source
- Written in Go
- Easy to use
- Automated data retention policy
- Schemaless
- Client libraries available
- Support for large amounts of data

DATA STRUCTURE

- **Measurement**, name of the measurement
- **Tags**, metadata for the measurement
- **Fields**, values for the measurement
- **Timestamp**, primary index is always time

```
// example:

[measurement],[tags] [fields] [timestamp]

weather_sensor,crop=blueberries,region=north temp=50.1 1472515200000000000
weather_sensor,crop=blueberries,region=midwest temp=49.8 1472515200000000000
```

QUERY LANGUAGE

- SQL Like
- CLI & HTTP-Api for read and writes
- Continuous Queries
- Operators & Mathematical Functions
- Automated data retention policies

```
// example:

SELECT MEAN("temp") FROM "weather_sensor" WHERE region = 'north'
```

DATA EXPLORATION

```
// GENERAL
SHOW DATABASES
SHOW SERIES
SHOW USERS

// SELECT
SELECT (*) FROM "wheather_sensor" GROUP BY region
SELECT (*) FROM "wheather_sensor" GROUP BY time(10m)
SELECT MEAN("temp") FROM "wheather_sensor" GROUP BY time(10m),region
SELECT MEAN("temp") FROM "wheather_sensor" GROUP BY time(10m),*
SELECT MEAN("temp") FROM "wheather_sensor" GROUP BY time(10m),* fill(none)

// INTO
SELECT MEAN("temp") INTO "grouped_data" FROM "wheather_sensor" GROUP BY time(10m)
```

WRITING DATA TO INFLUXDB

Manually using CLI or HTTP-API

```
INSERT weather_sensor,crop=blueberries,region=north temp=50.1
```

```
INSERT weather_sensor,crop=blueberries,region=north temp=50.1 1472515200000000000
```

or using client libraries
(Python, Java, Go, Elixir, JavaScript, .Net, ...)

INFLUXDB JAVA CLIENT

- uses InfluxDB HTTP-API
- Support batch operations
- Write / Query
- QueryResult mapper to POJO

```
influxDB.write(Point.measurement("cpu")
    .time(System.currentTimeMillis(), TimeUnit.MILLISECONDS)
    .addField("idle", 90L)
    .addField("user", 9L)
    .addField("system", 1L)
    .build());
```

```
Query query = new Query("SELECT idle FROM cpu", dbName);
```

SPRINGBOOT/GRAILS AND INFLUXDB

It is just as simple as adding dependencies on:

```
// file: build.gradle

compile "com.github.miwurster:spring-data-influxdb:1.6"
compile "org.influxdb:influxdb-java:2.9"
```

Result **DefaultInfluxDBTemplate** which can be configured using `application.yml`

WRITE DATA

```
class InfluxDBWriterService {

    @Autowired
    DefaultInfluxDBTemplate defaultInfluxDBTemplate     // get the template

    def writeToInfluxDB(json) {
        Point point = Point.measurement("temperature")  // create a point
                .time(System.currentTimeMillis(), TimeUnit.MILLISECONDS)
                .tag("location", json.location)
                .addField("temperature", new Double(json.temperature))
                .build()


        defaultInfluxDBTemplate.write(point)            // write a point to InfluxDB
    }
}
```

QUERY DATA TO POJO

```
InfluxDBResultMapper resultMapper = new InfluxDBResultMapper(); // threadsafe
Query query = new Query("SELECT * FROM cpu", defaultInfluxDbTemplate.getDatabase())
QueryResult queryResult = influxDB.query(query)
List<Cpu> cpuList = resultMapper.toPOJO(queryResult, Cpu.class)
```

```
@Measurement(name = "cpu")
public class Cpu {
    @Column(name = "time")
    private Instant time;
    @Column(name = "host", tag = true)
    private String hostname;
    // some code ommitted intentionally
}
```

DEMO

INSERT DATA USING GRAILS/SPRINGBOOT

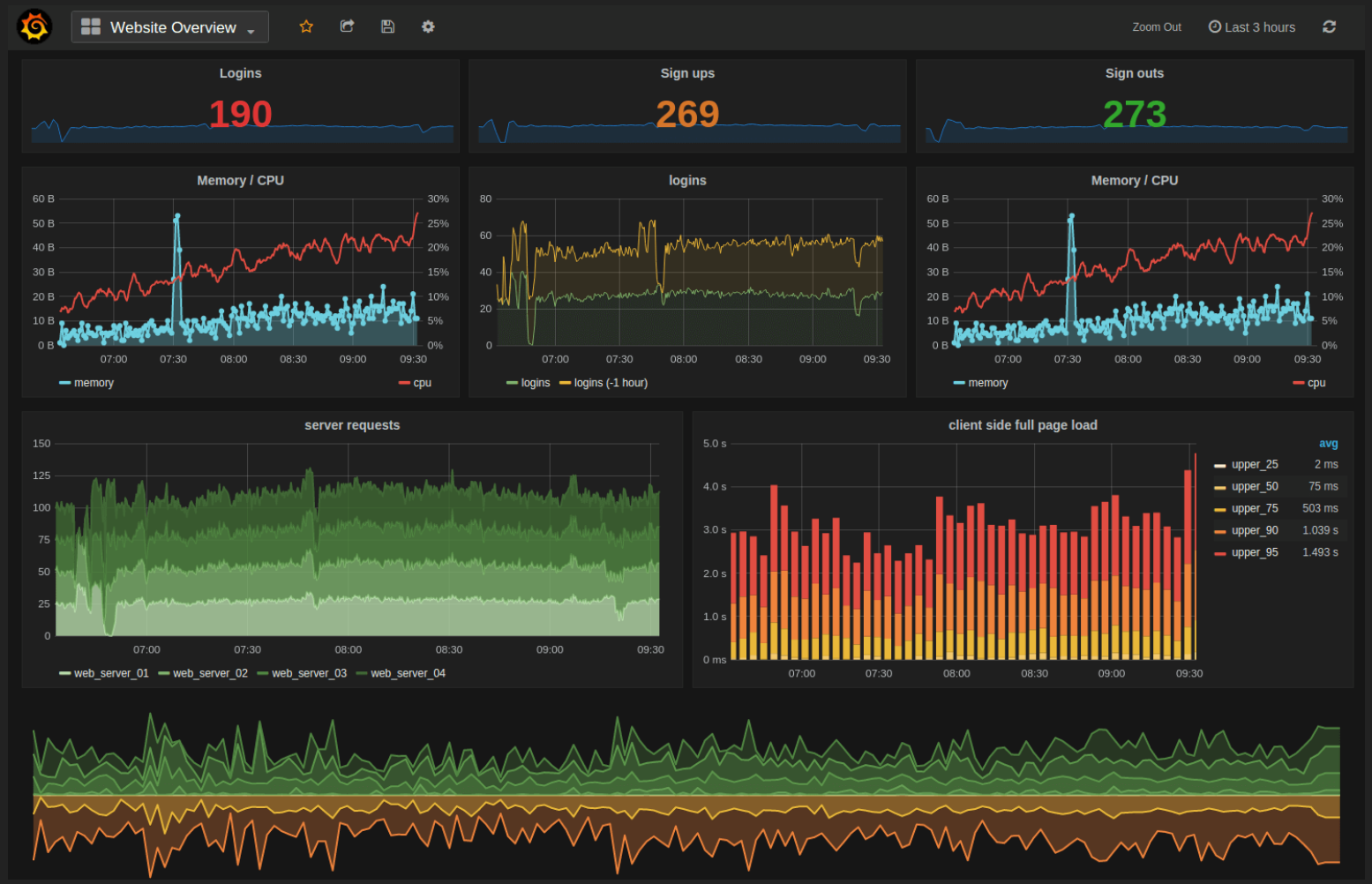
VISUALIZE THE DATA

FROM YOUR THINGS

Pretty pictures

VISUALIZATION OPTIONS

GRAFANA DASHBOARD



DEMO

GRAFANA - INFLUXDB

THANK YOU

https://github.com/mpas/the-internet-of-things-and-capturing-time-series-data