# Implementing a MQTT broker provides 3 crucial business needs :

- **Insight** :
  - The ability to monitor and understand the current situation.
- **Hindsight** :
  - The ability to understand effects of an event after it has happened, although you did not understand at the time.
- **Foresight**
  - AI and Machine Learning algorithms provide the ability to predict what will happen in the future based on collected and stored information.

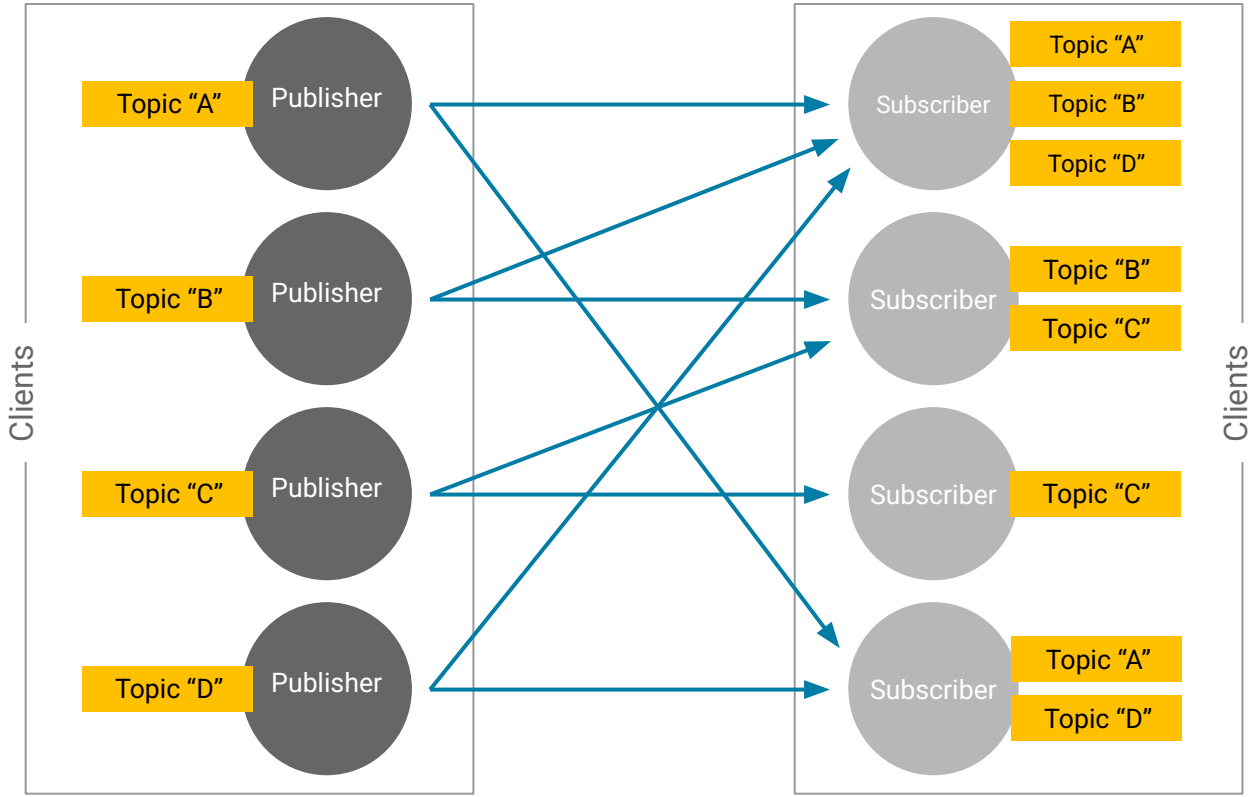
# API´s and OPU/UA: One on One Relationships

A lot of relation administration to administer

All publishers of data must “know” all Consumers; where they are and what data they produce.

Connection handling involves:

- Security
- Connection loss
- etc.



# Insert the MQTT broker :

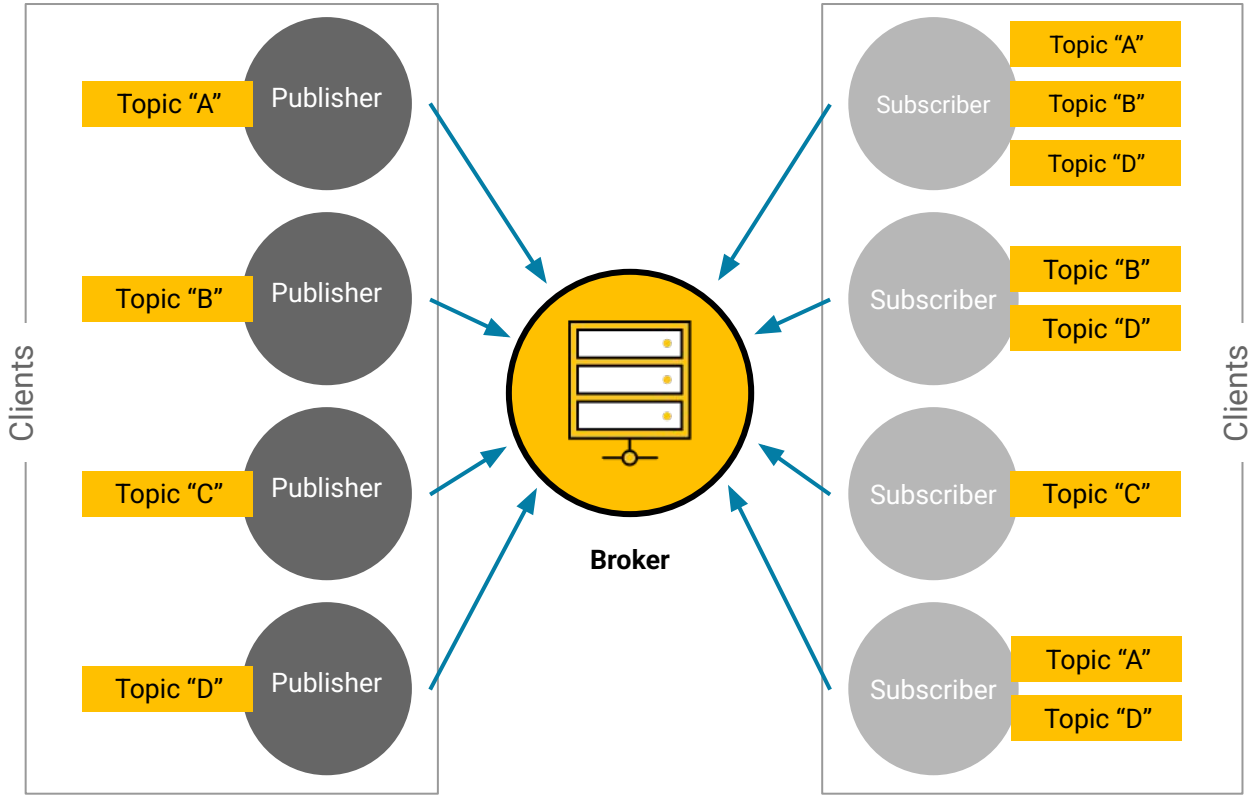
## Less administration and more benefits

A client only needs to know:

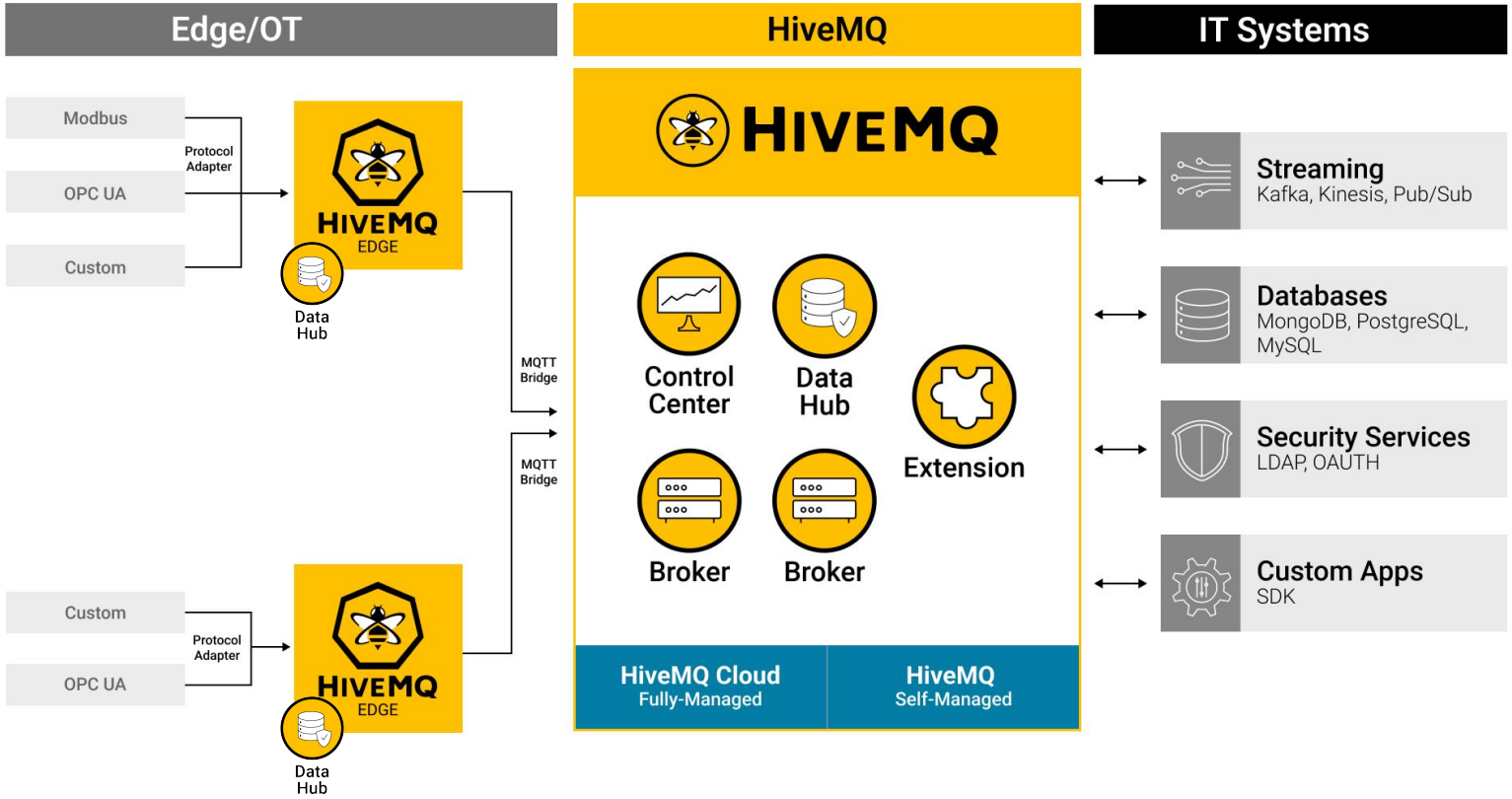
A. The broker (IP)
B. The topic to pub/sub to

Besides that the broker handles :

- Reconnects
- Disappearing Publishers (Last Will & Testament)
- Security
- and lots more



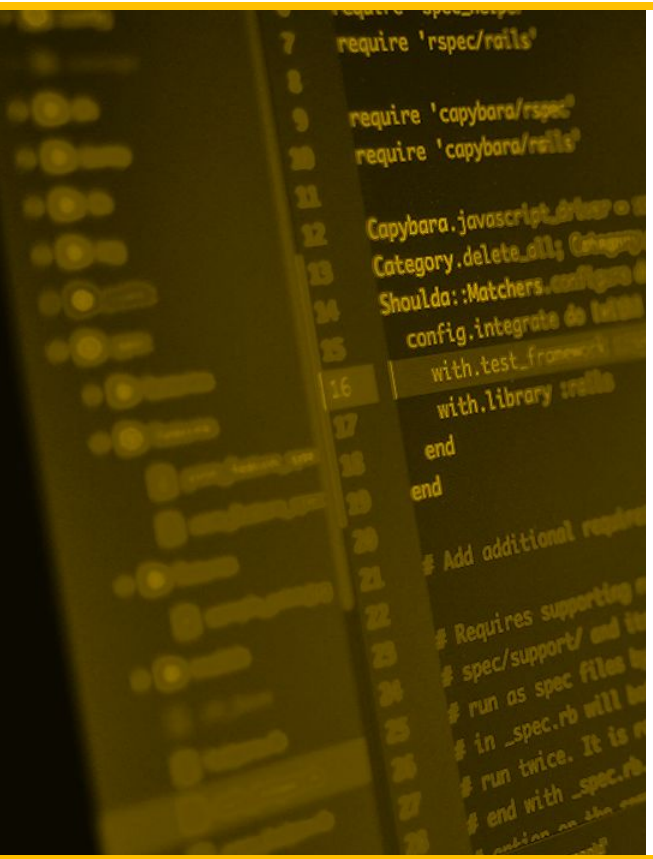
# Edge & Enterprise MQTT Platform



# HiveMQ with ESE, Swarm, Datahub, Kafka and Grafana

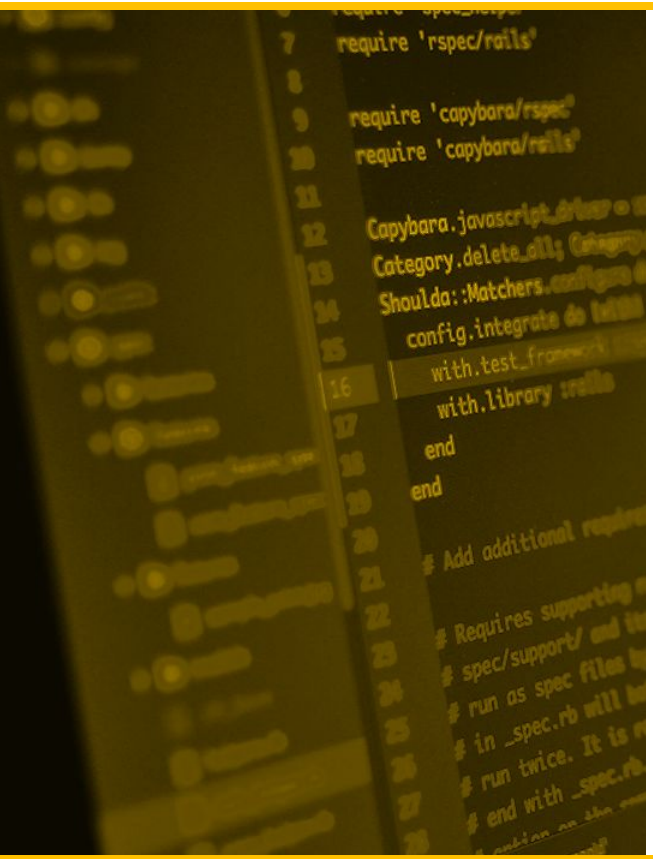
HO1 - A  dockerized hands-on session

# Directory Structure



```
Hivemq /
├── bin (run.sh/bat)
├── conf
│   └── examples
│       ├── configuration
│       └── logging
├── data
│   └── < STAY AWAY >
├── extensions
│   ├── hivemq-allow-all-extension - DISABLED
│   ├── hivemq-amazon-kinesis-extension
│   │   └── conf
│   ..
│   ├── hivemq-bridge-extension
│   │   └── conf
│   └── hivemq-data-lake-extension
│       └── conf
├── license
├── log (hivemq.log)
└── tools
    ├── hivemq-swarm
    │   └── scenario
    └── mqtt-cli
```

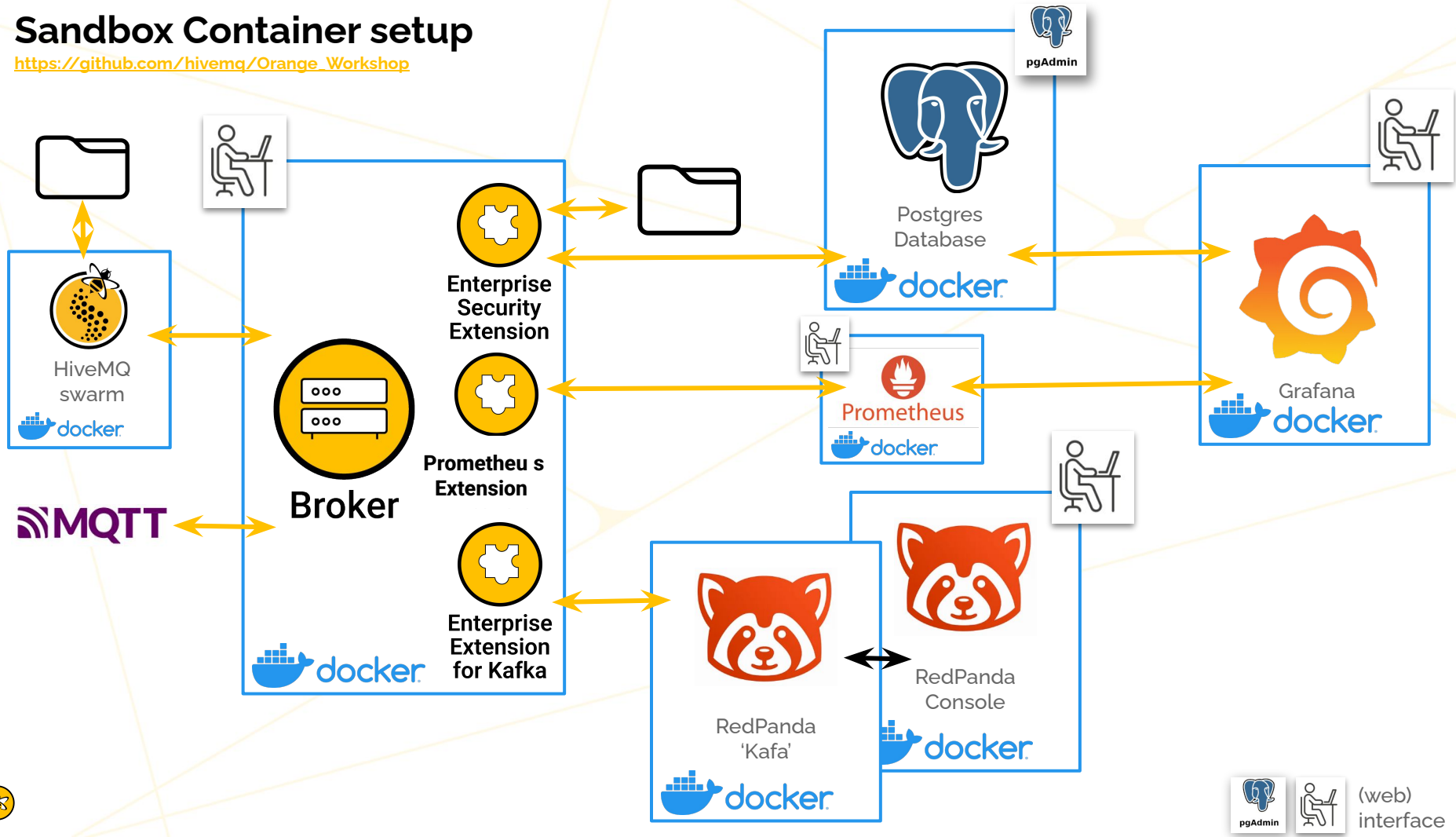
# Extension Structure




- Can be placed (or is already present) in the `extensions` directory.
- Connections and extensions are individually licensed. HiveMQ comes with a 25 connections and a 5 hr time-bombed ext set.
- All extensions are `DISABLED` with a flag.
- Default the `Allow-All` ‘security’ extension is enabled.
- All extensions are `HOT` reloadable. (Re)set the `DISABLED` flag.
- The `<ext.>/conf` directory is containing extension specific XML-based config (and sometimes support-) files.
- Check your <`HiveMQ>/log/hivemq.log` file.

# Sandbox Container setup

https://github.com/hivemq/Orange_Workshop



# HO-2 Enterprise Security Extension (ESE)

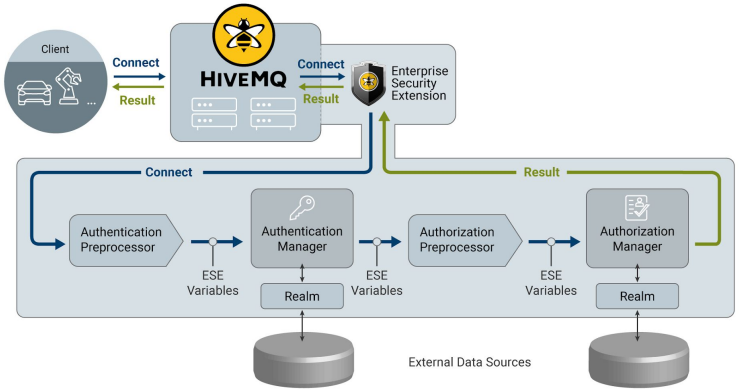
## File based RBAC realm

```
<enterprise-security-extension
xmlns:xsi="http://www.w3.org/2001/XMLS.-instance>

 <realms>
       <file-realm>
            <name>file-realm</name>
            <enabled>true</enabled>
            <configuration>
<file-path>conf/ese-file-realm.xml</file-path>
            </configuration>
       </file-realm>
   </realms>

   <pipelines>
       <listener-pipeline listener="ALL">
            <file-authentication-manager>
                <realm>file-realm</realm>
            </file-authentication-manager>
            <file-authorization-manager>
                <realm>file-realm</realm>
            </file-authorization-manager>
   </pipelines>
</enterprise-security-extension>
```

XML




```
<user>
   <name>mqtt-user-1</name>
   <password>mqtt-password-1</password>
   <permissions>
       <permission>
           <topic>#</topic>
           <qos>ALL</qos>
           <activity>ALL</activity>
           <retain>ALL</retain>
           <shared-subscription>ALL</shared-subscription>
           <shared-group>group-1</shared-group>
       </permission>
   </permissions>
</user>
```

ese-file-realm.xml

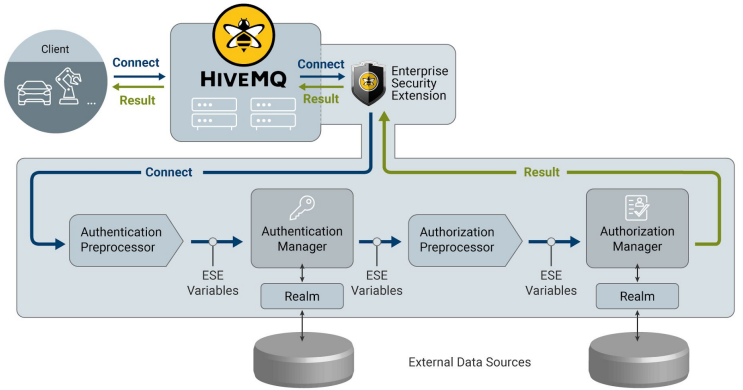
# H)2 - Enterprise Security Extension (ESE)

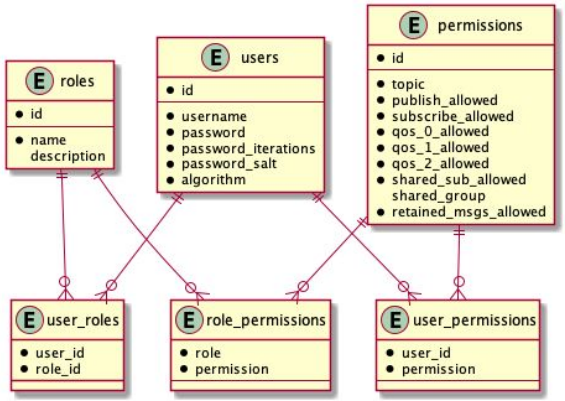
## Postgres RBAC realm

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<enterprise-security-extension

xmlns:xsi="http://www.w3.org/2001/XMLSchema"
      version="1">
   <realms>
      <!-- a postgresql db-->
      <sql-realm>
         <name>postgres-backend</name>
         <enabled>true</enabled>
         <configuration>
            <db-type>POSTGRES</db-type>
            <db-name>hivemq_ESE</db-name>
            <db-host>postgres_db</db-host>
            <db-port>5432</db-port>
            <db-username>hivemq</db-username>
            <db-password>secret</db-password>
         </configuration>
      </sql-realm>
   </realms>
```

XML



# HO2 - Authorization of MQTT Clients

- Granular and flexible permission definitions using topic filters that can be reused for role and user permissions.
- **Filters** are built by the following expressions:
  - Preserved MQTT variables named **mqtt-clientid**, **mqtt-username** and **mqtt-password**
  - Use of general purpose variables e.g. **string-1**

  Examples:

  All topics with wildcard: `#`
  Strict topic structure: `ohio/factory-01/machine-01/temperature`
  Dynamic with MQTT Variables: `+/${mqtt-username}/commands/#`
  Dynamic with General Purpose Variables: `${string-1}/${mqtt-clientid}/commands/+`

- **Permission** that can be set:
  - **Method (Sub / Pub):** Enable publishing and subscription rights individually
  - **QoS:** Enable QoS levels individually
  - **Retained Messages:** Allow to send retained messages
  - **Shared Subscription:** Enable shared subscription rights and define allowed shared subscription group name

# HO3 - Postgres Extension

## Uni-directional / Timescale, CrateDB etc.

```xml
<hivemq-postgresql-extension
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <postgresqls>
        <postgresql>
            <id>my-postgresql-id</id>
            <host>postgres_db</host>
            <port>5432</port>
            <database>hivemq_ESE</database>
            <username>hivemq</username>
            <password>secret</password>
        </postgresql>
    </postgresqls>

    <mqtt-to-postgresql-routes>
        <mqtt-to-postgresql-route>
            <id>my-mqtt-to-postgresql-route-template</id>
            <postgresql-id>my-postgresql-id</postgresql-id>
            <mqtt-topic-filters>
                <mqtt-topic-filter>Temp-TS/#</mqtt-topic-filter>
            </mqtt-topic-filters>
            <processor>
    <statement-template>conf/statement.sql</statement-template>
            </processor>
        </mqtt-to-postgresql-route>
    </mqtt-to-postgresql-routes>

</hivemq-postgresql-extension>
```

XML

```sql
INSERT INTO tempdata (sensorid,isotime, unixtime,
temperature)
SELECT
    json_data->>'SensorID' AS sensorid,
    (json_data->>'isotime'):: timestamp AS isotime,
    (json_data->>'unixtime')::numeric AS isotime,
    (json_data->>'temperature')::numeric AS temperature
/* casting from text to numic value ! */
FROM (
    VALUES
        ( ${mqtt-payload-utf8}::jsonb)
) AS input(json_data);
```

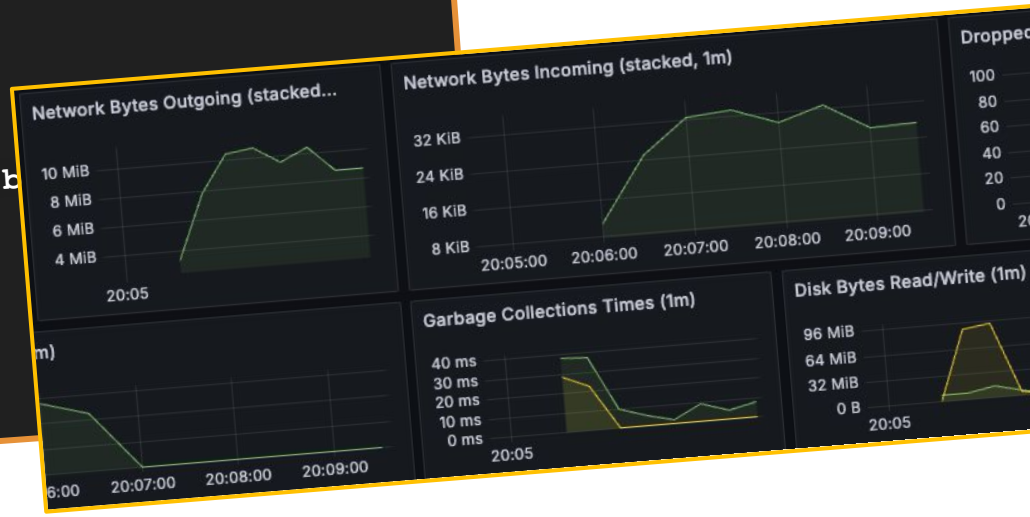
statement.sql

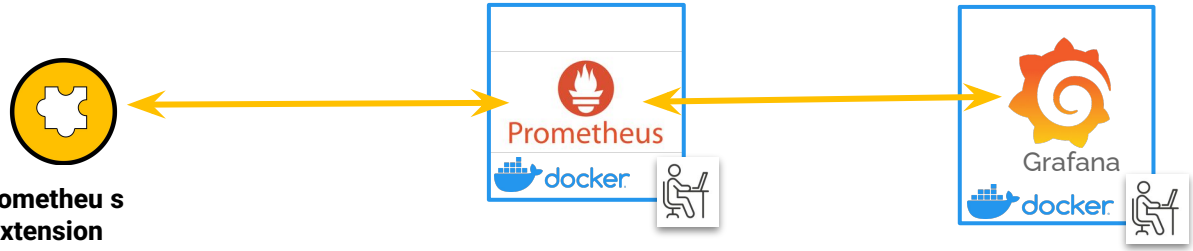
# HO4 - Prometheus Extension

## 'User added'

```
# The ip where the servlet will be hosted
ip=0.0.0.0
# The port where the servlet will work on
port=9399
# The path for the servlet which gets called b
# <ip>:<port> and <metric_path>
# For example 127.0.0.1:9399/metrics
metric_path=/metrics
```

prometheusConfiguration.properties



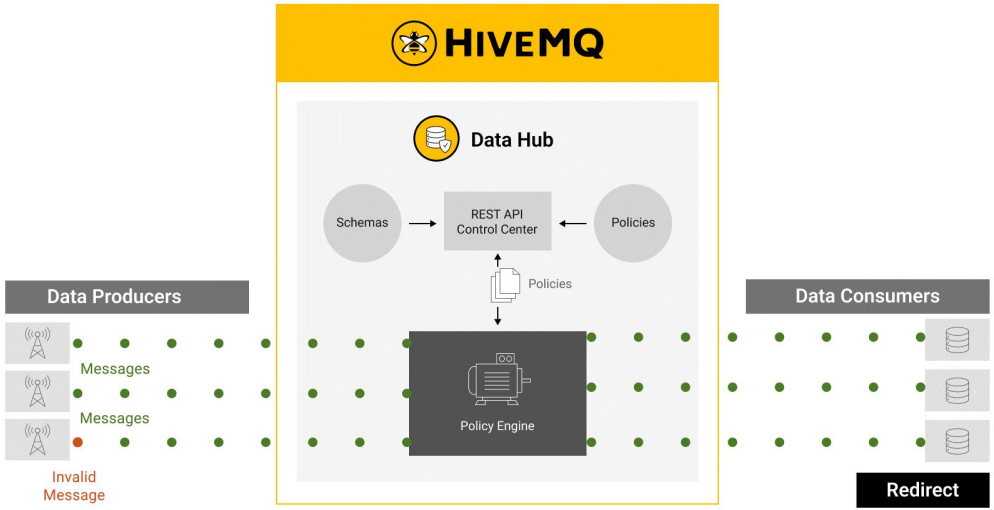



# HO6 - HiveMQ DataHub


Assure the quality of MQTT data:
Validate, contextualize and transform

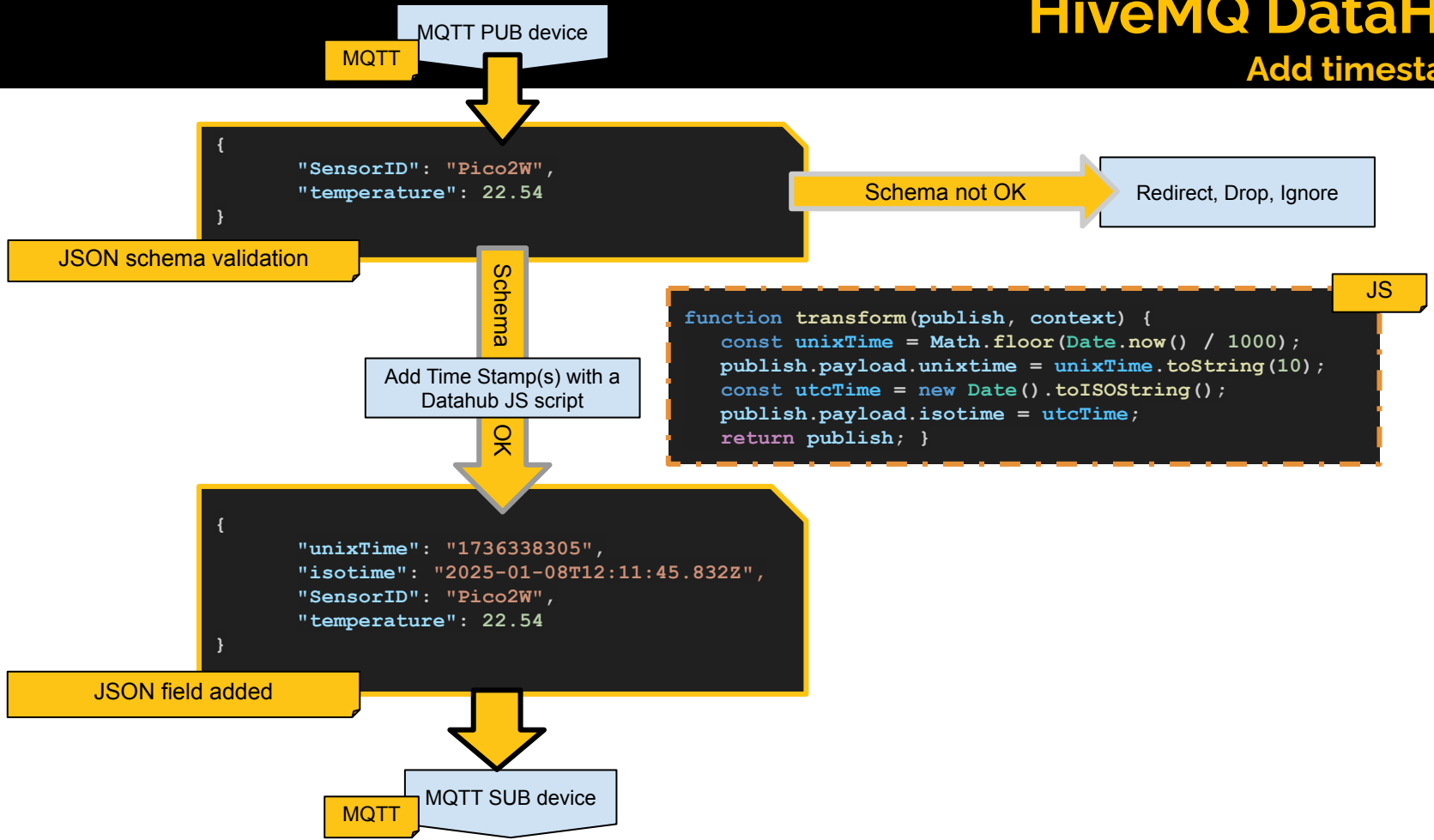
# DataHub: How does it work?

Devices send data to the broker in a data format that is agreed upon by data producers and consumers and the broker enforces that format ensuring the consumer always gets the data in the expected format.



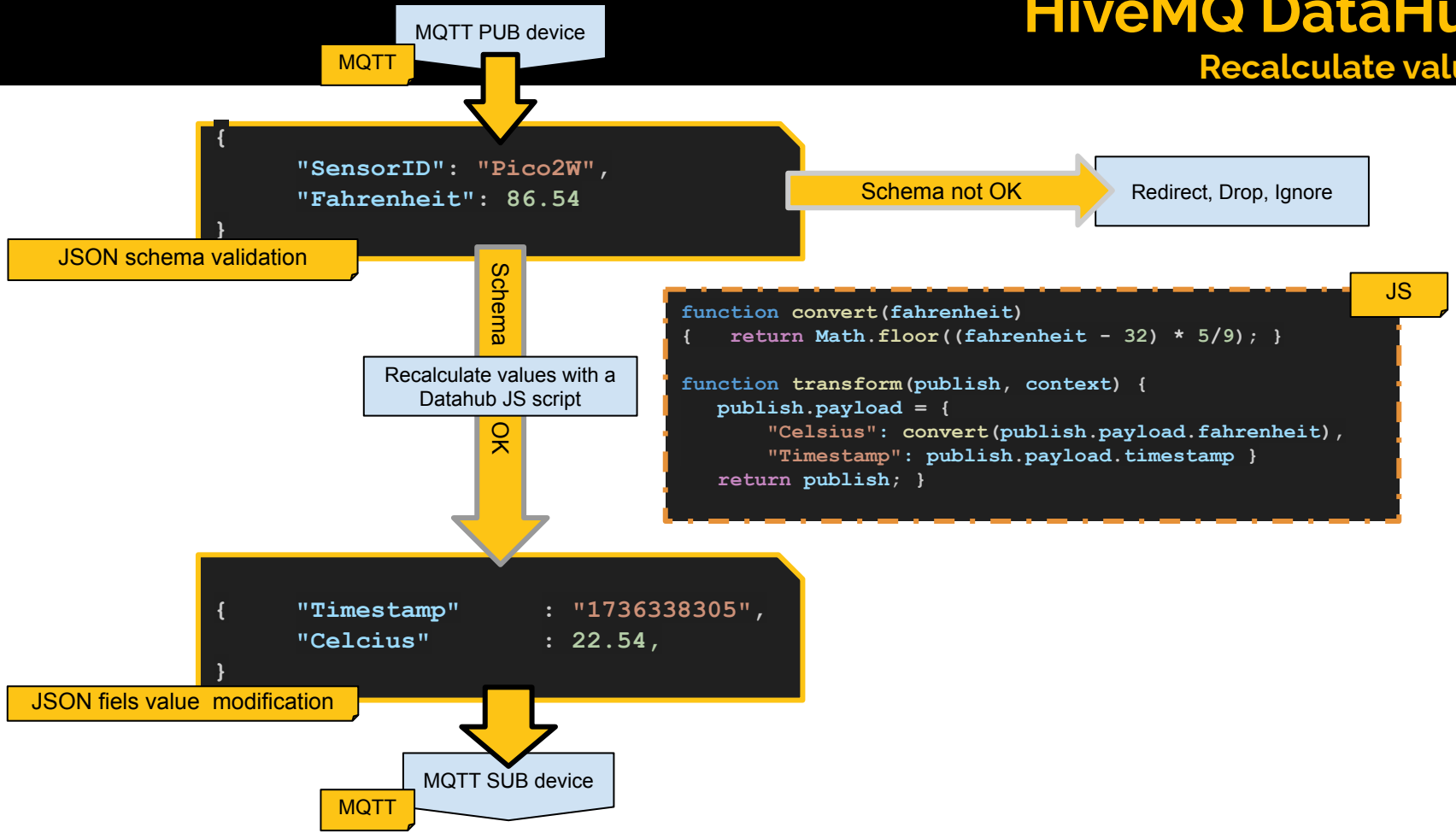
# HiveMQ DataHub

## Add timestamps



# HiveMQ DataHub

## Recalculate values



# HO6b - Kafka Extension

## Bi-directional / Azure Eventhub and Confluent !

```
<?xml version="1.0" encoding="UTF-8" ?>
<kafka-configuration
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="config.xsd">
    <kafka-clusters>
        <kafka-cluster>
            <id>cluster01</id>

<bootstrap-servers>redpanda-0:9092</bootstrap-servers>
        </kafka-cluster>
    </kafka-clusters>
```

XML 1/2

```
    <mqtt-to-kafka-mappings>
        <mqtt-to-kafka-mapping>
            <id>mapping 01</id>
            <cluster-id>cluster 01</cluster-id>
            <mqtt-topic-filters>

<mqtt-topic-filter>to-kafka/#</mqtt-topic-filter>
            </mqtt-topic-filters>
            <kafka-topic>kafka-topic</kafka-topic>
        </mqtt-to-kafka-mapping>
    </mqtt-to-kafka-mappings>

    <kafka-to-mqtt-mappings>
        <kafka-to-mqtt-mapping>
            <id>mapping02</id>
            <cluster-id>cluster 01</cluster-id>
            <kafka-topics>
                <kafka-topic>test</kafka-topic>

<kafka-topic-pattern>test-(.)*</kafka-topic-pattern>
            </kafka-topics>
        </kafka-to-mqtt-mapping>
    </kafka-to-mqtt-mappings>
</kafka-configuration>
```

XML 2/2

# HO6a -HiveMQ Swarm

## Distributed, fully configurable MQTT load testing

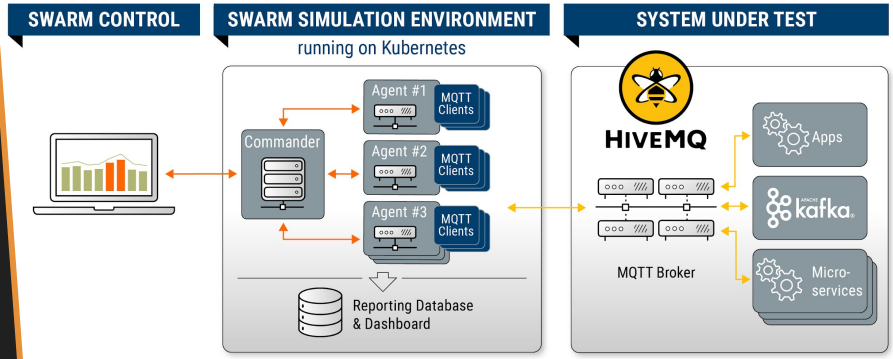
```xml
<brokers>
    <broker id="b1">
        <address>hivemq_broker</address>
        <port>1883</port>
    </broker>
</brokers>

<clientGroups>
    <clientGroup id="cg1">
        <clientIdPattern>A[0-9]{4}</clientIdPattern>
        <count>4</count>
    </clientGroup>
</clientGroups>

<topicGroups>
    <topicGroup id="tg1">
        <topicNamePattern>temp/subtopic-[0-9]</topicNamePattern>
        <count>3</count>
    </topicGroup>
</topicGroups>

<stages>
    <stage id="s1">
        <lifeCycle id="publish" clientGroup="cg1">
            <connect broker="b1"/>
            <publish topicGroup="tg1" count="100" message=' {
"SensorID": "CMD-Line",  "temperature": 22.54 } '  rate="1/5s"/>
        </lifeCycle>
    </stage>
</stages>
```

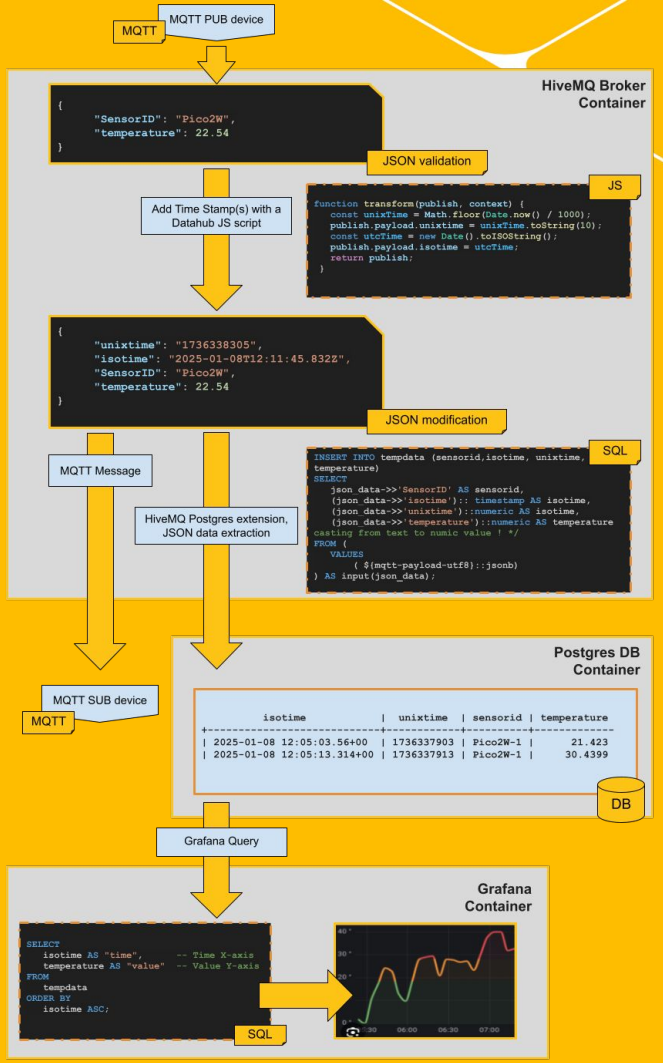
XML



# **HO6a - Usecase:** End-2-End dataprocessing and visualisation

Starring …

- HiveMQ Datahub
- HiveMQ Postgres ext.
- Postgres
- Grafana



# Day 1 : Foundations of HiveMQ

## Morning
## (8am to 12am)

**Mod 1 -> Overview of HiveMQ:** Understanding HiveMQ as an enterprise-ready platform.

**Mod 2 -> Key Features and Architecture:** Discuss the core features of HiveMQ, including scalability, security, and compliance with MQTT standards, extensions. Deep dive into the architecture of HiveMQ, focusing on its scalability and enterprise-grade security concepts.

**Ho1 -> Hands-On:** Basic configuration and setup of the Enterprise Broker.

**Mod 3 -> Security Configuration:** Configuring HiveMQ for enterprise-grade security, including authentication and authorization mechanisms.

**Ho2 -> Hands-On:** Add the Enterprise Security Extension to your deployment.

## Afternoon
## (2pm to 7pm)

**Mod 4 -> Enterprise Extensions:** Detailed look at extensions for Streaming, Databases, and Data lakes.

**Ho3 -> Hands-On:** Add the Enterprise Database extension for PostgreSQL to your deployment.

**Mod 5 -> Monitoring and Observability:** Setting up monitoring for HiveMQ deployments.

**Ho4 -> Hands-On:** Add Prometheus extension to your deployment, collect the metrics and visualize with Grafana.

**(Bonus) Mod 6 -> Performance Tuning:** Techniques for optimizing HiveMQ performance in high-load scenarios and how to use Swarm to test it.

**(Bonus) Ho5 -> Hands-On:** Extend your deployment to a cluster and run a load test.

# Day 2 : Data Hub and Edge

## Morning
## (8am to 12pm)

**Mod 7 -> HiveMQ Data Hub:** Understanding Data Hub and its modules for data transformation and validation.

**Ho6 -> Hands-On:** Configure a policy and data transformation in your platform.

**Mod 8 -> Introduction to HiveMQ Edge:** Understanding HiveMQ Edge and the differences between the open source edition and the enterprise edition. Overview of pre-built protocol adapters (Beckhoff ADS, Ethernet IP, HTTP(s), OPC UA, ModBus, Siemens S7, File Input, Simulation, BACnet).

**Ho7 -> Hands-On:** Setup of HiveMQ Edge, Configuration of an OPC-UA protocol adapter with northbound and southbound mappings.

## Afternoon
## (2pm to 7pm)

**Mod 9 -> Our SDKs:** Quick look at extensions and protocol adapter SDK and there capabilities.

**Recap:** Summary of the key points covered in the workshop.

**Q&A Session:** Open floor for questions and discussions.

**Brainstorming :** Share your challenges