

# Welcome to the HiveMQ Partner technical training

All resources for this training can be found here: [https://github.com/hivemq/Orange\\_Workshop](https://github.com/hivemq/Orange_Workshop)

Internal resources :  Technical track

## Who is this workshop for ?

This session is aimed at our partners' technical teams. We take a closer look at the platform, its different bricks and how they work.

We will cover topics like enterprise broker deployment and security, extension configuration, DataHub policies and transformations, HiveMQ Edge and its protocol adapters.

Prerequisite :

- Have a good understanding of HiveMQ Platform.
- Have basic knowledge on kubernetes and linux.

## What is the goal of the training ?

The aim of this workshop is to train our partners' teams in the installation and advanced configuration of the HiveMQ platform.

After attending the workshop, you will be able to implement a production site with best practices on security and enterprise extensions configured.

We wish you a pleasant training

May 2025,

Anthony Olazabal  
Kamiel Straatman



## Preparations :

Please have Docker, Docker Compose, Git, and the HiveMQ CLI installed:

<https://docs.docker.com/engine/install/>

<https://docs.docker.com/compose/install/>

[HiveMQ CLI](#)

[Git CLI](#) or [GitDesktop](#)

Useful additional software :

[PgAdmin](#)

[MQTT explorer](#)

Brew 4 Mac : `/bin/bash -c "$(curl -fsSL \`  
`https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"`



## Introduction to the Container setup

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

This workshop is based on running the several needed software components, including the broker(s), in a multi-container Docker environment. This is to create an equal and easy to manage sandbox infrastructure within a range of different laptop systems.

To get the resources please run the following command in your terminal within an appropriate local directory (eg. MyDocuments) :

Unset

**On Mac / Linux :**

```
git clone https://github.com/hivemq/Orange\_Workshop.git
```

Unset

**On Windows :**

```
git config --global core.autocrlf false
```

```
git clone https://github.com/hivemq/Orange\_Workshop.git
```

**TIP:**

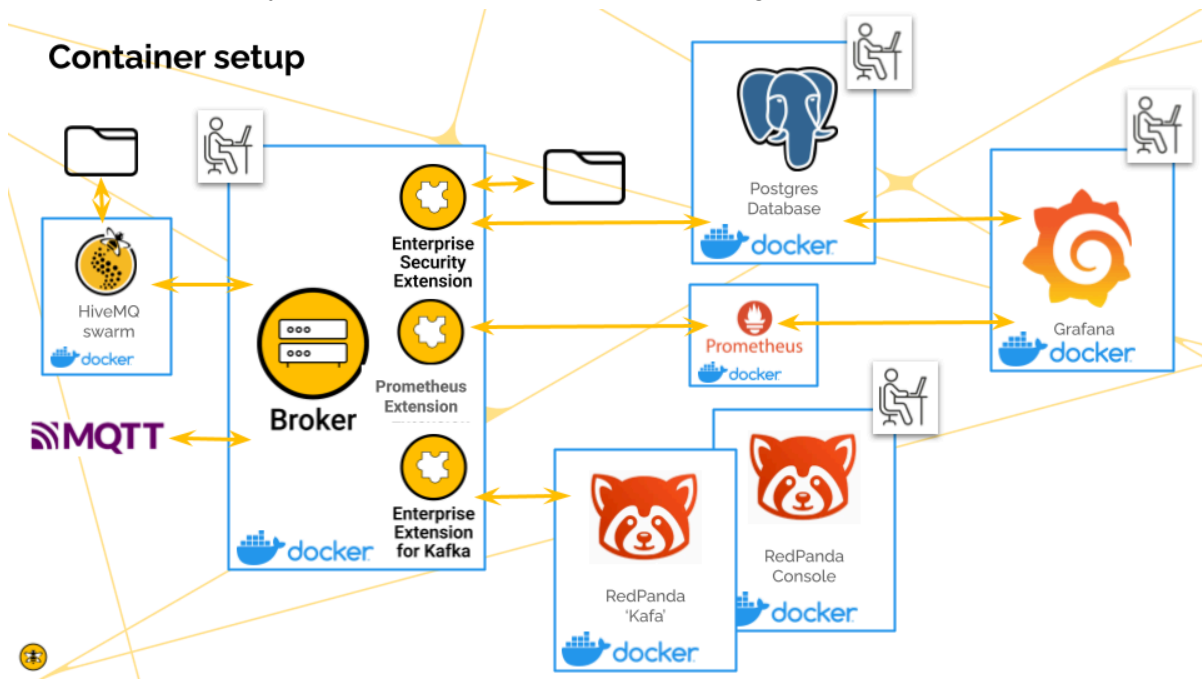
Windows (eg VScode) will add \r to the \n character by default. This usually leads to errors when building Docker (=unix based) Images. Make sure CR/LF translation is switched off.

In VScode settings.json file select this for LF only:

```
{  "files.eol": "\n", }
```



This clone will render you with infra-as-code for the following setup of containers:



Basically you will have a single node broker container that is able to interface with a Postgres database, a Redpanda Kafka system and a prometheus monitor database. The Postgres- as well as the Prometheus-database can be queried by the Grafana visualisation system.

In the front-end a HiveMQ swarm container is foreseen that can generate MQTT traffic according to need.

Now, in the downloaded Dockerfile, you can selectively en/disable extensions with entries starting around line 60:

```
# Default allow all extension, set this to false to disable it
ENV HIVEMQ_ALLOW_ALL_CLIENTS=true
# Enable ESE, set this to false to disable it
ENV HIVEMQ_ENABLE_ESE=false
# Enable Kafka extension, set this to false to disable it
ENV HIVEMQ_ENABLE_KAFKA=false
# Enable REST API default value
ENV HIVEMQ_REST_API_ENABLED=true
```

Please start initially with the above settings. Later in these sessions you can enable selective extensions one by one.



## Use it / Start :

MAC Linux : to start please `cd` into the cloned `Orange_Workshop` directory and run the following commands:

```
Unset
export HIVEHQ_VERSION=4.38.0
export REDPANDA_VERSION=24.2.7
export REDPANDA_CONSOLE_VERSION=2.7.2
./build.sh
docker-compose up -d --build --force-recreate
```

Windows: to start please open a Powershell, `cd` into the cloned `Orange_Workshop` directory and run the following commands:

```
Unset
$env:HIVEHQ_VERSION="4.38.0"
$env:REDPANDA_VERSION="24.2.7"
$env:REDPANDA_CONSOLE_VERSION="2.7.2"
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
./build.ps1
docker-compose up -d --build --force-recreate
```

## Test by connecting to the broker GUI

The HiveMQ GUI for a local or dockerized broker can be accessed on <http://127.0.0.1> and by default on port 8080. The default credentials are **admin** and as password **hivemq**.

Please test: open your browser and go to page <http://127.0.0.1:8080>. Familiarise yourself with the layout and don't forget to check out our new CC 2.0 interface. Repeat review when you test with MQTT commands in the next session.



## Using HiveMQ and review tooling and GUI

### **Sending and receiving a MQTT message**

A broker is not able to send MQTT packages by itself so we need to have some MQTT client software (or devices) to do so. A number of options to do this are available to your liking.

### **HiveMQ CLI both available for Mac and Windows.**

Downloadable at <https://www.hivemq.com/blog/mqtt-cli/>

We recommend this CLI because it provides a compact command line interface (CLI) for MQTT 3.1.1 and MQTT 5 clients that supports interactive command modes. It also supports access to the HiveMQ API in an easy to access way.

Please put the executable in one of your PATH directories.

### **Good GUI-oriented, Mac and Windows based alternatives are:**

MQTT.fx (payed): <https://www.softblade.de/>

MQTT explorer (free) : <https://mqtt-explorer.com/>

Except for the HiveMQ specific API interaction these GUI based tools can be used during this course.

Please test your basic browser connectivity over mqtt by subscribing to, and subsequent publishing to a known topic. This can be your local broker but also try your neighbors IP address.

### **Example:**

test with the command : `mqtt test -h 127.0.0.1`

ping-pong:

`mqtt sub -h 127.0.0.1 -t "#"` (this command will 'wait' for input, try the -J option as well)

And in another window:

`mqtt pub -h 127.0.0.1 -t "sensor/temp" -m "20.4 degrees"`

*(copy and paste may miscopy the " char, be aware)*

As an option use the IP address of your neighbour to replace the local 127.0.0.1 address to send and receive messages from remote brokers.

**Congratulations**, you're able to send and receive MQTT messages.



# The Enterprise ESE extension

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

Now, in your `Dockerfile` you can enable the ESE extension. Rerun a `docker-compose up -d --build --force-recreate` and the config files will be included in the running image.

```
# Default allow all extension, set this to false to disable it
ENV HIVEMQ_ALLOW_ALL_CLIENTS=false
# Enable ESE, set this to false to disable it
ENV HIVEMQ_ENABLE_ESE=false
# Enable Kafka extension, set this to false to disable it
ENV HIVEMQ_ENABLE_KAFKA=false
# Enable REST API default value
ENV HIVEMQ_REST_API_ENABLED=true
```

First test with the above settings. Adapt your `Dockerfile` accordingly and rerun a `docker-compose up -d --build --force-recreate`

You can test with the `mqtt test -h 127.0.0.1` command. What do you notice ? Also check the logging in the dockerfile.

Now enable the ESE extension and Rerun a `docker-compose up -d --build --force-recreate`

```
# Default allow all extension, set this to false to disable it
ENV HIVEMQ_ALLOW_ALL_CLIENTS=false
# Enable ESE, set this to false to disable it
ENV HIVEMQ_ENABLE_ESE=true
# Enable Kafka extension, set this to false to disable it
ENV HIVEMQ_ENABLE_KAFKA=false
# Enable REST API default value
ENV HIVEMQ_REST_API_ENABLED=true
```

Please check your `exe-resources/config.xml` extension config file. We provided two examples for this file; one based on the file realm (with accompanying `ese-file-realm.xml` data) and one based on the PostgreSQL database backend. By default the file realm is used as `config.xml`.



## Test

Depending on the realm, different users/password combinations may be active.

Please check the realm data sources ( `ese-file-real.xml` for the file realm, `init.sql` (Line > 446) for the database realm)

```
<users>
  <user>
    <name>mqtt-user-1</name>
    <password>mqtt-password-1</password>
    <permissions>
      <permission>
        <topic>#</topic>
        <qos>ALL</qos>
        <activity>ALL</activity>
      </permission>
    </permissions>
  </user>
```

File realm test :

```
mqtt test -u mqtt-user-1 -pw mqtt-password-1
```

Now copy the database realm template file (`config.xml.postgr-realm`) to the `resources-ese/config.xml` and rerun a `docker-compose up -d --build --force-recreate`

Below an example of entries that are held in the database:

```
insert into public.users
(username, password, password_iterations, password_salt, algorithm)
values
('backendservice', 'wtUo2dri+ttHG..AQULg==', 100, 'Nv6NU9XY7tvHdSGaKmNTow==', 'SHA512'),
('frontendclient', 'ZHg/rNJellBH.Z8ps1Q==', 100, 'JhpW27QU9WfIaG6FJT5MkQ==', 'SHA512'),
('superuser', 'nOgr9rAKm/LqxAt8o.AbVK6Q==', 100, 'wxw+3diCV4bWXQHb6LLniA==', 'SHA512');

insert into public.permissions
(id, topic, publish_allowed, subscribe_allowed, qos_0_allowed, qos_1_allowed, qos_2_allowed,
retained_msgs_allowed, shared_sub_allowed, shared_group)
values
(1, 'topic/+status', false, true, true, true, true, false, false, ''),
(2, 'topic/${mqtt-clientid}/status', true, false, true, true, true, true, false, ''),
(3, '#', true, true, true, true, true, true, true, '');

insert into public.roles
(id, name, description)
values
(1, 'backendservice', 'only allowed to subscribe to topics'),
(2, 'frontendclients', 'only allowed to publish to topics'),
(3, 'superuser', 'is allowed to do everything');
```





Database realm test:

```
mqtt test -u superuser -pw supersecurepassword
```

Or do the ping-pong test (in multiple windows or over multiple laptops):

```
mqtt sub -u superuser -pw supersecurepassword -t "#"
```

```
mqtt pub -u superuser -pw supersecurepassword -t topic-3/test -m testmsg
```

This concludes the ESE session.



# The Postgres DB extension

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

Now, in your `Dockerfile` you can enable the Postgres extension and optionally disable ESE.

Rerun a `docker-compose up -d --build --force-recreate` and the config files will be included in the running image. Also check the logging on the console of the broker. No errors must be seen here.

```
# Default allow all extension, set this to false to disable it
ENV HIVEMQ_ALLOW_ALL_CLIENTS=true

# Enable ESE, set this to false to disable it
ENV HIVEMQ_ENABLE_ESE=false

# Enable Postgres extension, set this to false to disable it
ENV HIVEMQ_ENABLE_POSTGRES=true
```

Please check the files `config.xml` and `statement.sql` in the postgres resources-postgres directory. As stated a topic filter is applied :

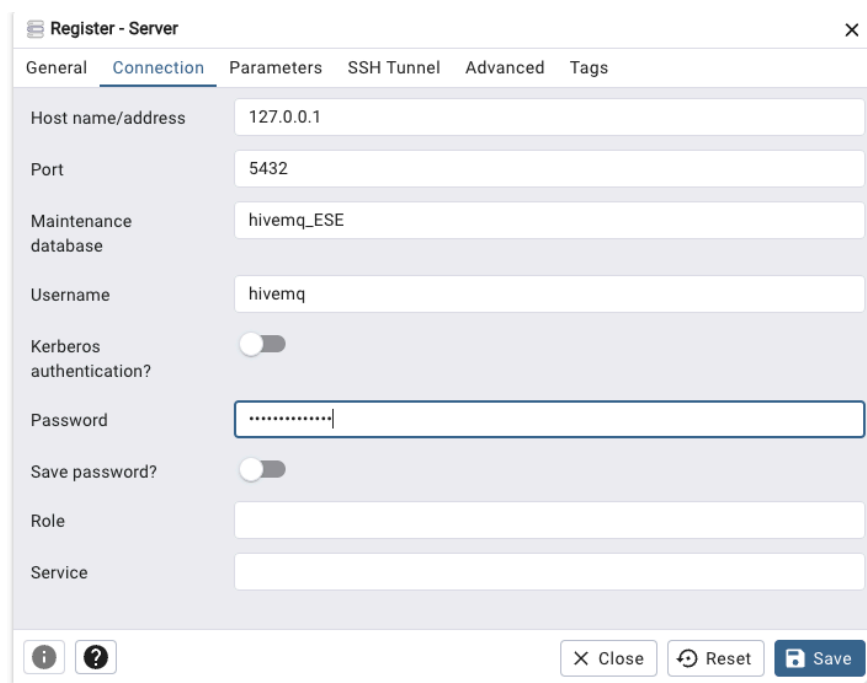
```
<mqtt-topic-filters>
  <mqtt-topic-filter>Temp-TS/#</mqtt-topic-filter>
</mqtt-topic-filters>
<processor>
  <statement-template>conf/statement.sql</statement-template>
</processor>
```

And a SQL insert statement is called:

```
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);
```



Please test by connecting your **PGadmin** tool to the Postgress container  
( database : hivemq\_ESE / username : hivemq / Password : secretpassword)



The screenshot shows the 'Register - Server' dialog box in PGAdmin, with the 'Connection' tab selected. The fields are as follows:

Field	Value
Host name/address	127.0.0.1
Port	5432
Maintenance database	hivemq_ESE
Username	hivemq
Kerberos authentication?	<input type="checkbox"/>
Password	.....
Save password?	<input type="checkbox"/>
Role	
Service	

At the bottom, there are buttons for 'Close', 'Reset', and 'Save', along with information and help icons.

Now keep an eye on the 'tempdata' table. This can be viewed under  
[Databases/hivemq\\_ESE/Schemas/public/Tables/tempdata](#) (Rightclick 'View data').

When publishing a valid json file to the broker the data on a specific topic will be ingested by the database insertion script

Test by sending a valid JSON file and refreshing the database table contents:

```
mqtt pub -t Temp-TS//test --message-file=tempdata.json
```

This concludes the Postgres database extension section.



# The Prometheus extension

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

In your `Dockerfile` the prometheus extension is enabled by default. By default this extension comes with a properties file containing

```
ip=0.0.0.0

# The port where the servlet will work on
port=9399

# The path for the servlet which gets called by prometheus
# <ip>:<port> and <metric_path>
# For example 127.0.0.1:9399/metrics
metric_path=/metrics
```

Please check the metrics coming in into the Prometheus timeseries database at

<http://localhost:9090/query>

The already running Grafana docker instances will have a preloaded HiveMQ visualisation dashboard available at (admin/admin):

<http://localhost:3000/>

This concludes the Prometheus / Grafana section



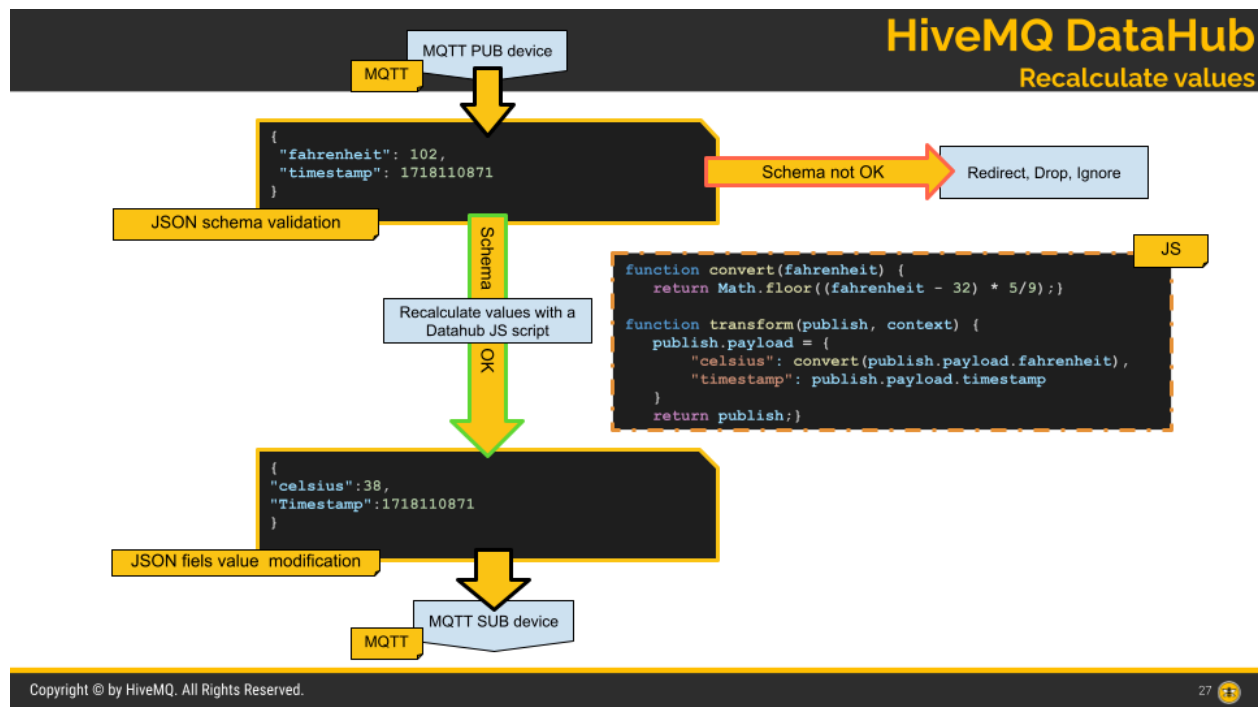
# Datahub

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

The HiveMQ Data Hub provides mechanisms to define how MQTT payload-data and MQTT client-behavior is handled within the HiveMQ broker. Data Validation in the HiveMQ Data Hub allows you to implement declarative policies that check whether your data sources are sending data in the data format you expect and provides the ability to even change it on the fly.

Data modification allows Datahub to perform alteration/correction of the actual MQTT payload. A truly unique feature.

We are working here with an example where a device is sending data in JSON format with the temperature in Fahrenheit. This is an actual case where the vendor was not allowed to change firmware because of FCC regulations. The EU based consumer of the data wanted to have all data normalized to degrees celsius. HiveMQ Datahub is able to do this at scale.



## Setup Datahub

**Ho6 -> Hands-On:** Configure a policy and data transformation in your platform. Con't

In this case we will extend a simple JSON structure not containing timestamps into one that will contain these.

Open the GUI for the local broker (<http://127.0.0.01> / hivemq/ admin) and select on the left menu 'Datahub/Schemas'. In the selected screen enable the 5 hour trial version for datahub on top of your screen or use the API:

```
(C:\Windows\System32\curl -X POST localhost:8888/api/v1/data-hub/management/start-trial
```

Now upload the two selected JSON schema definitions (schema-for.., schema-from..)

```
mqtt hivemq schema create --id schema-from-device --type json --file schema-from-device.json
```

```
mqtt hivemq schema create --id schema-for-fan --type json --file schema-for-fan.json
```

or copy paste their content in the schema definition fields. <sup>1</sup>

Now do the same for the javascript:

```
mqtt hivemq script create --id=fahrenheit-to-celsius --file=script.js --type=transformation
```

or copy paste their content in the schema definition fields.

The policy can be manually created with the data policy entry on the left menu or can be uploaded over the API:

```
mqtt hivemq data-policy create --file policy.json
```

Or define it interactively in the GUI:

**Matching Topic Filter** : factory/#

### Schema Validators

All Of - Schema with id *schema-from-device* and version *latest*

### On Success Pipeline:

- Operation - **Deserialize** Schema with id *schema-from-device*
- Operation - **Script** - fahrenheit-to-celsius - latest : Link to Scrip
- Operation - **Serdes** - Serialize id *schema-for-fan*

### On Failure Pipeline:

operation- Drop Message with Reason String: Your client \${clientId} sent invalid data according to the schema: \${validationResult}.

---

<sup>1</sup> This is an excellent json to schema convertor :

<https://www.liquid-technologies.com/online-json-to-schema-converter>



Now test :

```
mqtt pub -h 127.0.0.1 -t factory/test -m '{"fahrenheit": 102, "timestamp": 1718110871}'
```

Output of `mqtt sub -h 127.0.0.1 -t "#"` should be like:

```
{"celsius":38, "timestamp":1718110871}
```

Or use a MQTT client of your choice. Subscribe to “#” and see what happens when you send some valid test data. Please also monitor your brokers log screen.

This concludes the Datahub section.



# Putting it all together :

## DataHub, Database, swarm and visualisation

Ho6a -> **Putting it all together** :DataHub, Database and visualisation

Now enable both Postgres and API access. Disable the rest for clarity:

```
# Default allow all extension, set this to false to disable it
ENV HIVEMQ_ALLOW_ALL_CLIENTS=true
# Enable ESE, set this to false to disable it
ENV HIVEMQ_ENABLE_ESE=false
# Enable Kafka extension, set this to false to disable it
ENV HIVEMQ_ENABLE_KAFKA=false
# Enable REST API default value
ENV HIVEMQ_REST_API_ENABLED=true
```

- ☐ Windows: Use the GUI as in previous Daaahub topic
- ☐ MAC / Linux: Use the mqtt Hivemq API feature as below:

First enable datahub in testmode over the API:

```
(C:\Windows\System32\)curl -X POST
localhost:8888/api/v1/data-hub/management/start-trial
```

Upload schema's scripts and policies with the HiveMQ CLI tool:

```
mqtt hivemq schema create --id=mytemp-in-schema \
--file=./resources-datahub/mytemp-in-schema.json --type=json
```

```
mqtt hivemq schema create --id=mytemp-out-schema \
--file=./resources-datahub/mytemp-out-schema.json --type=json
```

```
mqtt hivemq script create --id=add_timestamp \
--file=./resources-datahub/add_timestamp.js --type=transformation
```

```
mqtt hivemq data-policy create --file=./resources-datahub/add_ts_policy.json
```





Test by sending simplified JSON:

```
mqtt pub -t temp/test --message-file=./resources-datahub/mytemp.json
```

Or start a swarm container (config in local resources-swarm directory) by executing

```
docker compose start swarm
```

Check the grafana temperature dashboard, already preconfigured, here:

<http://localhost:3000/d/be9o4k0uzfnk0e/temp-dashboard> or keep an eye on both the temp table in the database and the broker logging.

This concludes the 'all together section'.



# The Kafka extension

## Ho6b \_> The Kafka extention

Now, in your `Dockerfile` you can enable the kafka extension along with ESE. Rerun a `docker-compose up -d --build --force-recreate` and the config files will be included in the running image. Check the brokers console for the correct loading of the "HiveMQ Enterprise Extension for Kafka" and for possible errors. Fix is necessary.

```
# Default allow all extension, set this to false to disable it
ENV HIVEMQ_ALLOW_ALL_CLIENTS=true
# Enable ESE, set this to false to disable it
ENV HIVEMQ_ENABLE_ESE=true
# Enable Kafka extension, set this to false to disable it
ENV HIVEMQ_ENABLE_KAFKA=true
# Enable REST API default value
ENV HIVEMQ_REST_API_ENABLED=true
```

Please review the config files that are present in the local 'resources-kafka' directory:

```
<mqtt-to-kafka-mappings>
  <mqtt-to-kafka-mapping>
    <id>mapping01</id>
    <cluster-id>cluster01</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>
```

The predefined kafka config files will filter on the 'to-kafka' topic and will push it towards the kafka topic kafka-topic.



You can also send MQTT data towards the Kafka propagation topic (ESE is active so you need some credentials here):

Unset

```
mqtt pub -u superuser -pw supersecurepassword -t \
to-kafka/test -m my1stKafkaMsg
```

See in Redpanda console the MQTT sent message appear in the correct `kafka-topic` topic by accessing the red panda console on <http://localhost:8090/topics/>

This concludes the Kafka section.



## Overview:

### Day 1:

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

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

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

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

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

### Day 2:

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

**Ho6a -> Putting it all together :**DataHub, Database and visualisation

**Ho6b -> The Kafka extension**

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

