Prep:

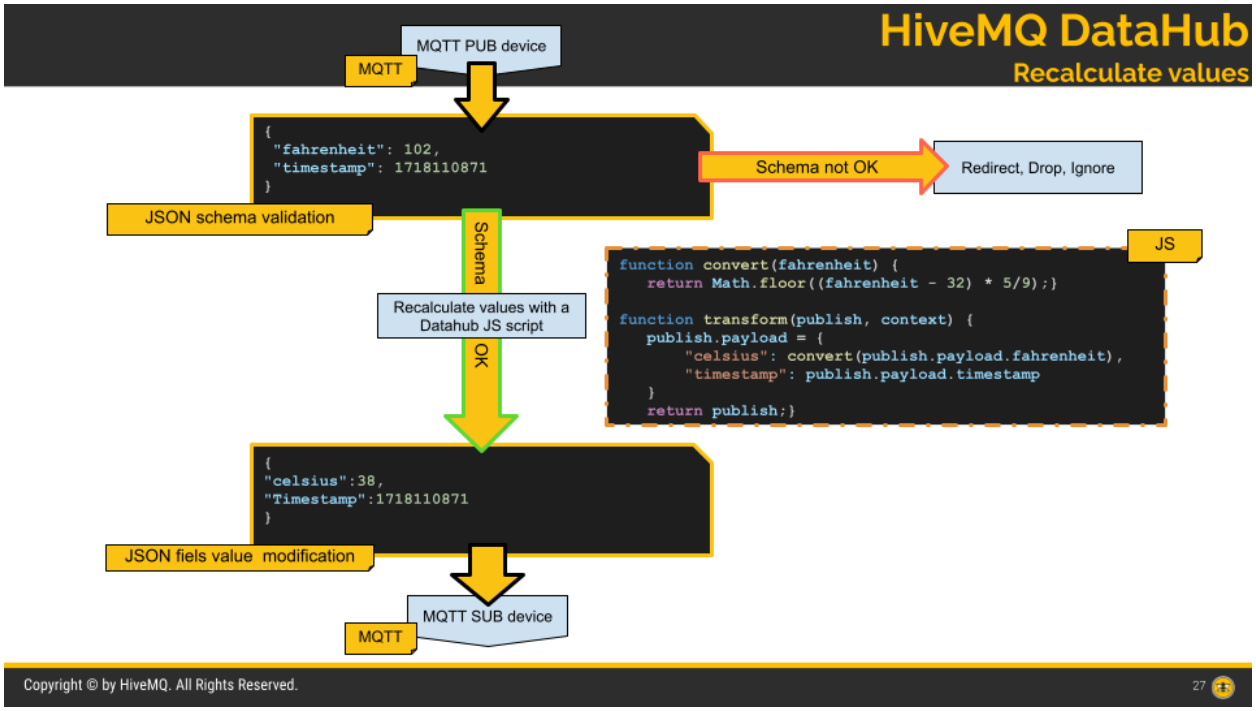
In the Dockerfile you can selectively en/disable extensions:

```
# 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 with the above settings. Later in these sessions you can enable selective extensions one by one.

Datahub

The HiveMQ Data Hub provides mechanisms to define how MQTT data and MQTT client behavior are handled in 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.



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 trail for datahub.

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.

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 s*chema-for-fan*

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

Now test :

```
mqtt pub -t factory/test  -m "{\"fahrenheit\": 102,\"timestamp\": 1718110871}"
```

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

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