Gateway ex Machina

Agile IoT extended with Thingsboard and ESP32

D1. System architecture and communications

Document version: v0.4

Table of contents

D1. System architecture and communications 1

Project overview **3**

Goals overview 3

D1 overview **4**

Gateway ex Machina Architecture **5**

Background 5

Architecture design approach 6

Scenario #1 - Original TB-GW as a Kura OSGi bundle 7

Challenges 7

Scenario #2 - Lightweight TB GW as a Kura OSGi bundle 8

Scenario #3 - TB-GW as a service, managed by a Kura OSGi bundle 9

Scenario #4 - TB-GW as a service, multi-container 10

Challenges 10

Scenario #5 - NodeRED as a TB-GW 11

Challenges 11

Scenario #6 - TB-GW as an AGAIL SDK app 12

Challenges 12

Scenario #7 - Kura CloudService compatible broker as TB-GW 13

Project overview

Gateway ex Machina (GxM) extends and integrates Agile IoT technologies with Thingsboard [1] and ESP32 [2] in an attempt to create the necessary software for an industrial grade robust gateway with wireless sensors that can be later used by EXM in industrial IoT projects.

Goals overview

1. Utilize Eclipse Kura and connect to MODBUS smart meter devices.
2. Integrate Eclipse Kura with Thingsboard.io and communicate telemetry from MODBUS devices on the field to the Thingsboard cloud server.
3. Enable easy remote device management and OTA software update of GW software by utilizing dockerized Kura, resin.io and other AGAIL technologies.
4. Develop an ESP32 mesh sensor network that is bridged with the GW, enabling sensor telemetry data to be forwarded to Thingsboard server.
5. Add ESP32 firmware OTA capability via GW.

[1] ThingsBoard is an open-source IoT platform for data collection, processing, visualization, and device management. https://thingsboard.io

[2] Espressif ESP32 is a low cost WiFi/Bluetooth enabled 32bit, dual core, Arduino compatible microcontroller https://www.espressif.com/en/products/hardware/esp32/overview

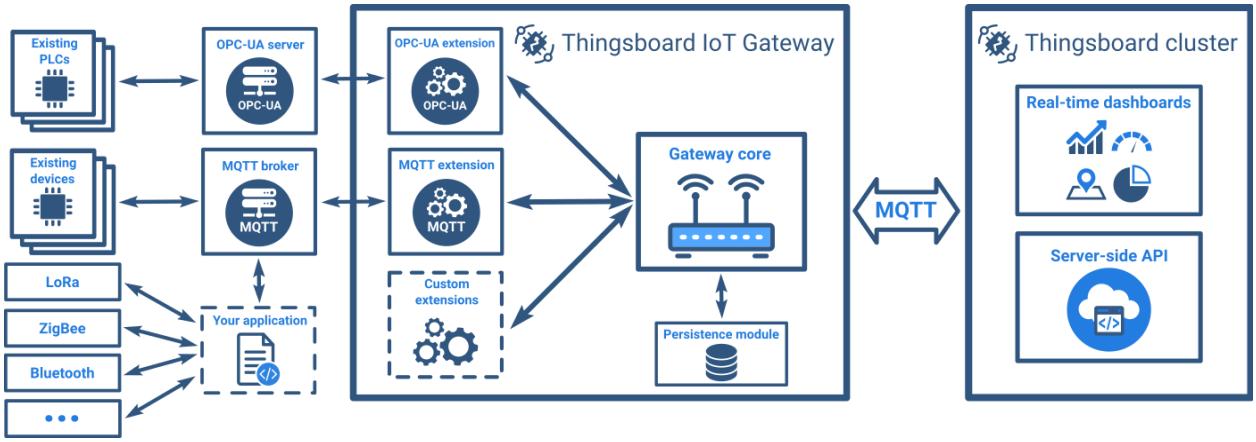
D1 overview

This document contains the architecture design, explaining the communication and integration points between all software and hardware components. This document will evolve into the documentation of the published source explaining the final integration mechanism between Eclipse Kura and Thingsboard.io

Gateway ex Machina Architecture

Background

Thingsboard project includes a powerful gateway open-source software designed to run at the edge, connecting to the Thingsboard server over MQTT, as well as to local hardware sensors.



https://github.com/thingsboard/thingsboard-gateway

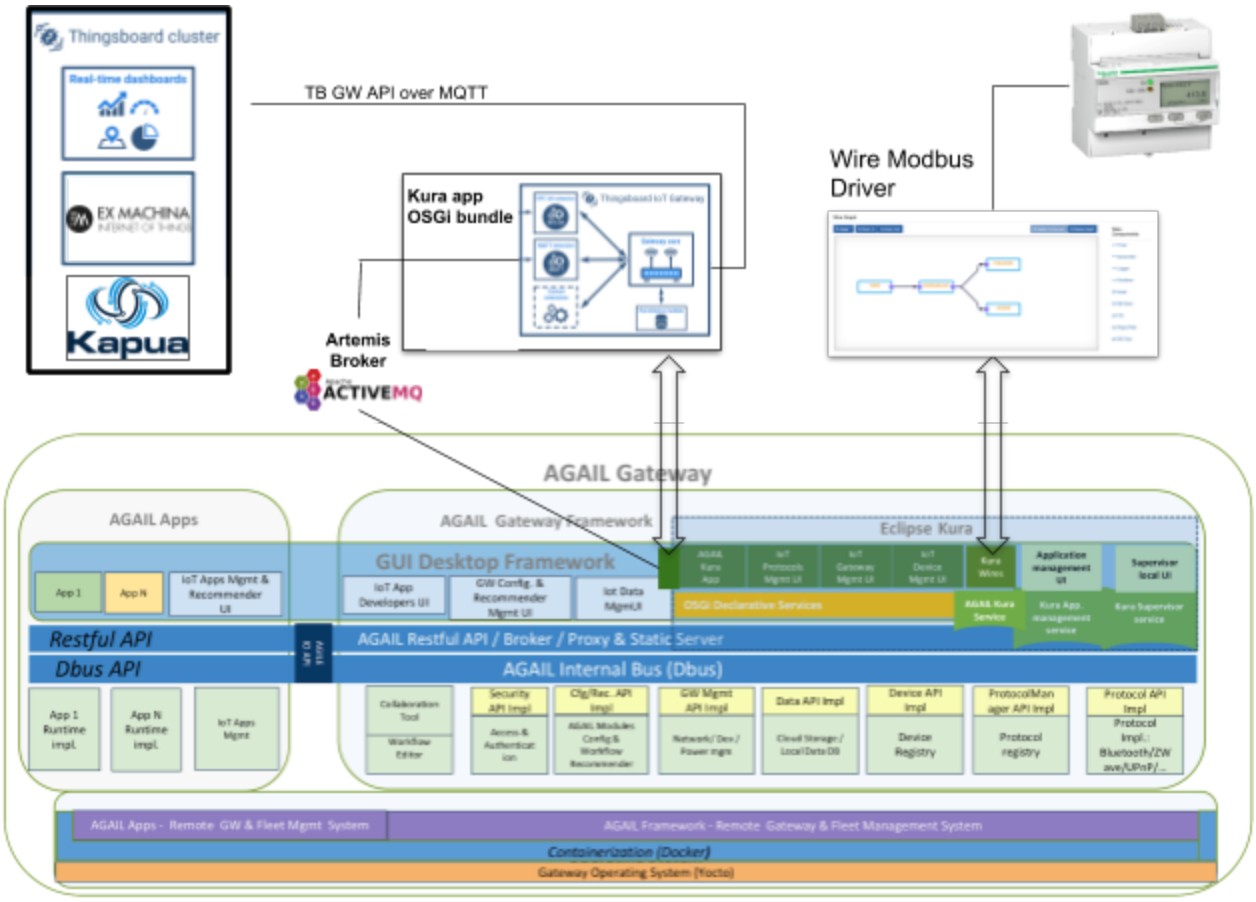
The Gateway provides simple integration APIs, and encapsulates common Thingsboard related tasks: device provisioning, local data persistence and delivery, message converters/adaptors and other. It is designed with the assumption that application developers will connect to it through an external MQTT broker or an OPC-UA server, or by implementing custom extensions that support other protocols (e.g. MODBUS). It is based on Java Spring and its monolithic nature introduces various limitations for application developers who wish to extent it with custom protocols, plus it lacks device management features such as OTA software updates, self-healing agents etc. Therefore the combination of Eclipse Kura and other Agile IoT technologies with the thingsboard gateway will bring multiple advantages to application developers.

After performing a thorough evaluation of Eclipse Kura, Resin.io and the Eclipse AGAIL project, a number of architecture approaches were identified, which are described below.

Candidate solution architectures and approaches

Thingsboard gateway (TB-GW) is an active project, where new features are constantly added, and it is important to ensure seamless upgradability with future versions of the gateway. Therefore a loose coupled approach is preferred versus one that requires major refactoring of the TB-GW source code. A number of architecture scenarios has been explored with proof of concept implementations in order to conclude on the final architecture that will be used for the public version of the software.

Scenario #1 - Original TB-GW as a Kura OSGi bundle

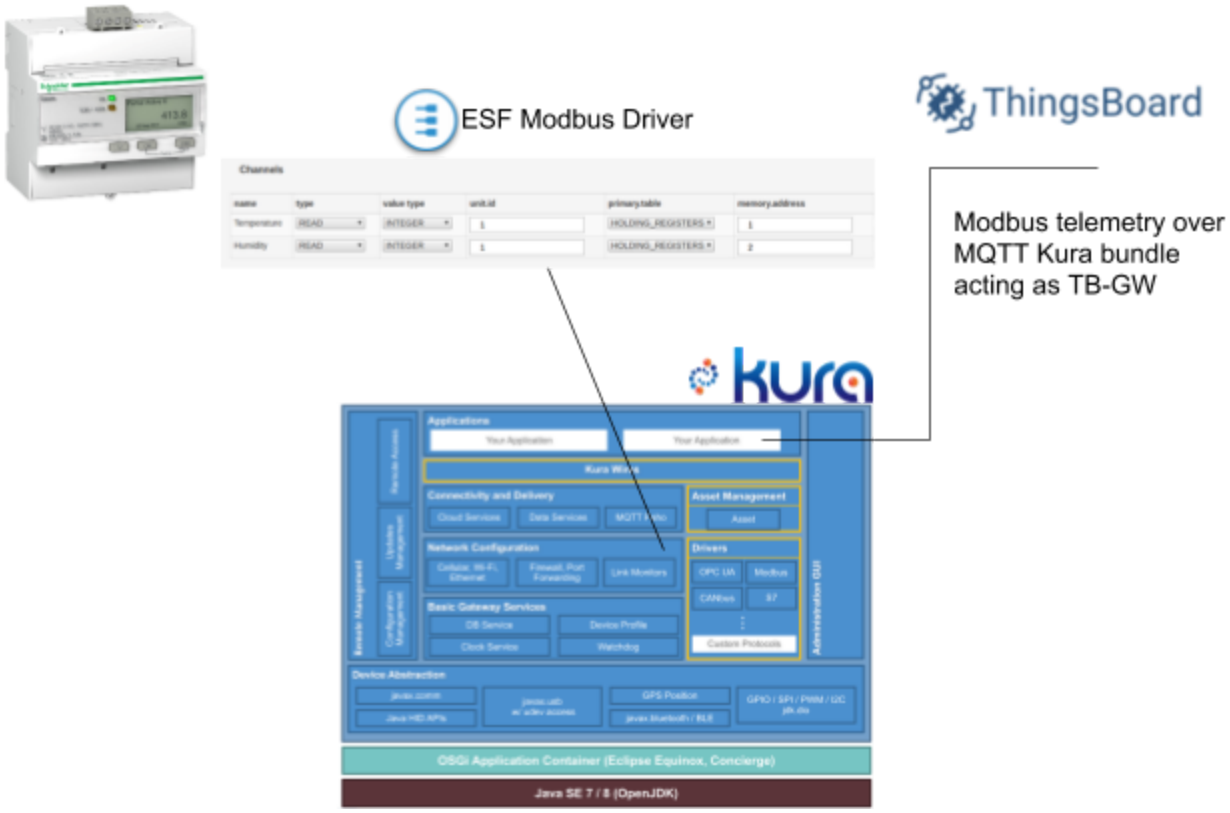


This is the original idea/proposal scenario, where the existing TB-GW software is wrapped as an OSGi bundle and becomes fully managed by Kura. Telemetry data integration with Kura will not take place internally via the OSGi event bus, but externally (localhost) via local MQTT broker where both Kura's MQTT CloudService and TB-GW will connect. By integrating telemetry over MQTT the MQTT payload mapping capability of the TB-GW is retained. While progressing with the implementation of this architecture it became apparent that the TB-GW software will require substantial modifications in order to be converted in to an OSGi bundle therefore alternative approaches were further investigated.

Challenges

Since the project proposal , TB-GW has evolved substantially. In TB-GW v1.4 snapshot there are multiple configuration files that can be remotely managed by the TB server, via an HTTP service. As a result, wrapping the original TB-GW in a Kura OSGi bundle, became more complex and requires substantial refactoring making it less future proof, thus alternative architectures were sought.

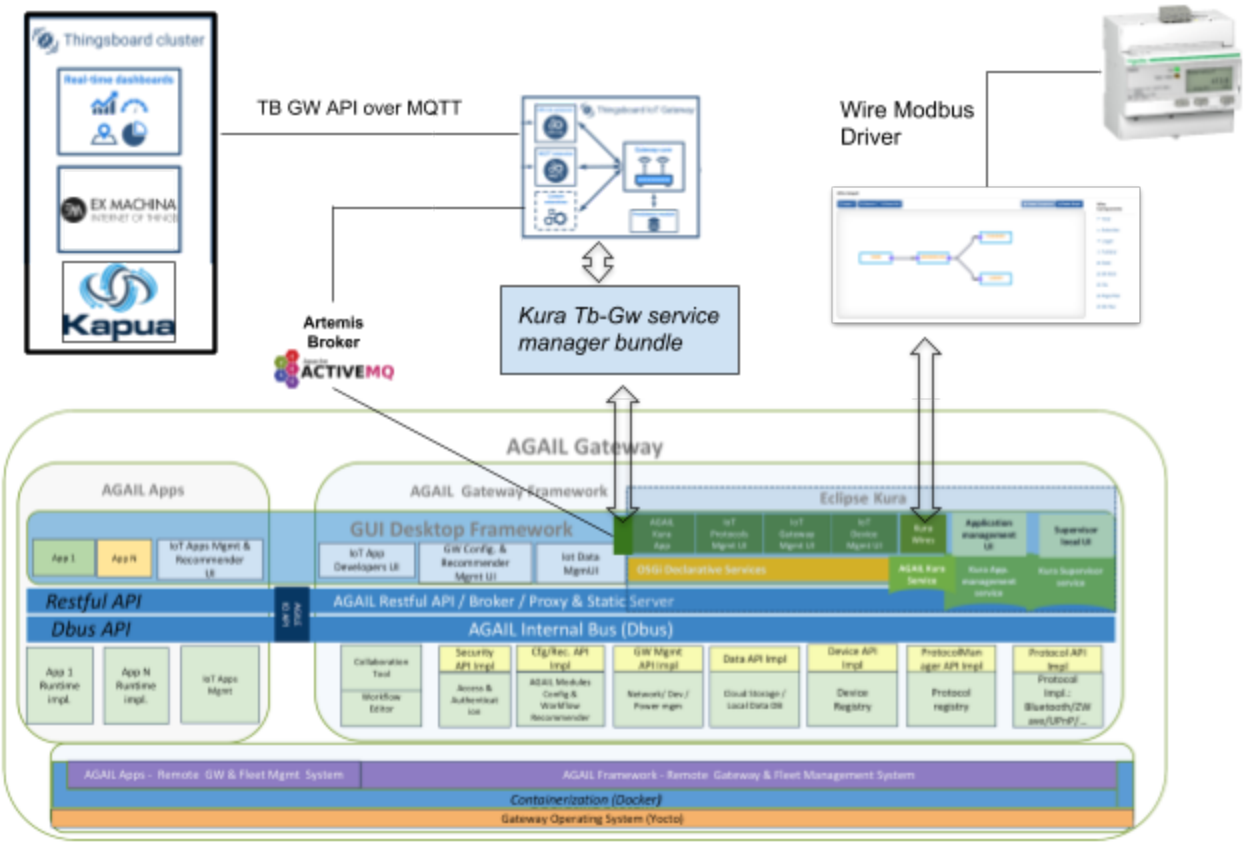
Scenario #2 - Lightweight TB GW as a Kura OSGi bundle



To take full advantage of Kura's modular architecture, we decided to create a new, lightweight Java TB gateway OSGi bundle that is not based on the original TB-GW Java Spring source, but instead is utilizing Eclipse Paho MQTT library for direct communication with TB server over the TB gateway API. Doing so we avoid the complexities in refactoring the original TB-GW Java Spring app into an OSGi bundle, which require the handling of external configuration files within the Kura environment.

Documentation and source code published on github (Deliverable D2):
https://github.com/exmgr/Kura-Thingsboard-Bundle

Scenario #3 - TB-GW as a service, managed by a Kura OSGi bundle



This architecture aims to utilize the original TB-GW deployed as intended by its authors, i.e. as a service, for maximum future proofing. For Kura to be able to manage an external service, a new custom Kura app / OSGi bundle could be created, that will provide an external managing ability for TB-GW service.

A first step towards this approach is to have the TB-GW service running as normal and then install a custom Kura bundle from which the user can start/stop it using Kura's web ui.

TB-GW is then configured to subscribe to Kura's built-in Artemis MQTT broker to listen for devices publishing telemetry data which it then transforms to a Thingsboard compatible JSON format and forwards it the Thingsboard platform. As a final step in the setup process, a CloudService must be set up that will be used by the Kura Wires graphs to publish telemetry to the local ArtemisMQTT service. The user can then add his devices as Assets in Kura, configure their data inputs as channels, and set up Kura Wire graphs with a simple Timer -> Asset -> Publisher flow to publish their data to Thingsboard.

More information on this secnario can be found here:
https://github.com/exmgr/Kura-Tb-Gateway-Manager

Scenario #4 - TB-GW as a service, multi-container

Building on scenario #3 for maximum flexibility on OTA software updates for the GW software, an additional (to the one running Kura) docker container could be utilized just for running the TB-GW service. Resin.io supports already multi-containers[3] therefore this architecture would be useful in the later phases of the project where OTA via resin.io would be introduced.

Challenges

We faced major performance issues while implementing this scenario. We used the latest version of AgileIoT image (v0.3.5) based on raspbian stretch with a dockerized version of the Thingsboard Gateway (v1.4) from the official repository. This resulted in a very sluggish operation for all the components involved rendering the whole system unusable at times. Possible solutions could include some kind of optimization we could apply that would improve the system's performance (e.g. ResinOS).

Furthermore, the ports set up to be exposed by Kura's docker container (port 1883 for MQTT and 5002 for telnet) were not accessible from the host and thus Thingsboard gateway was unable to subscribe to Kura's Artemis MQTT broker.

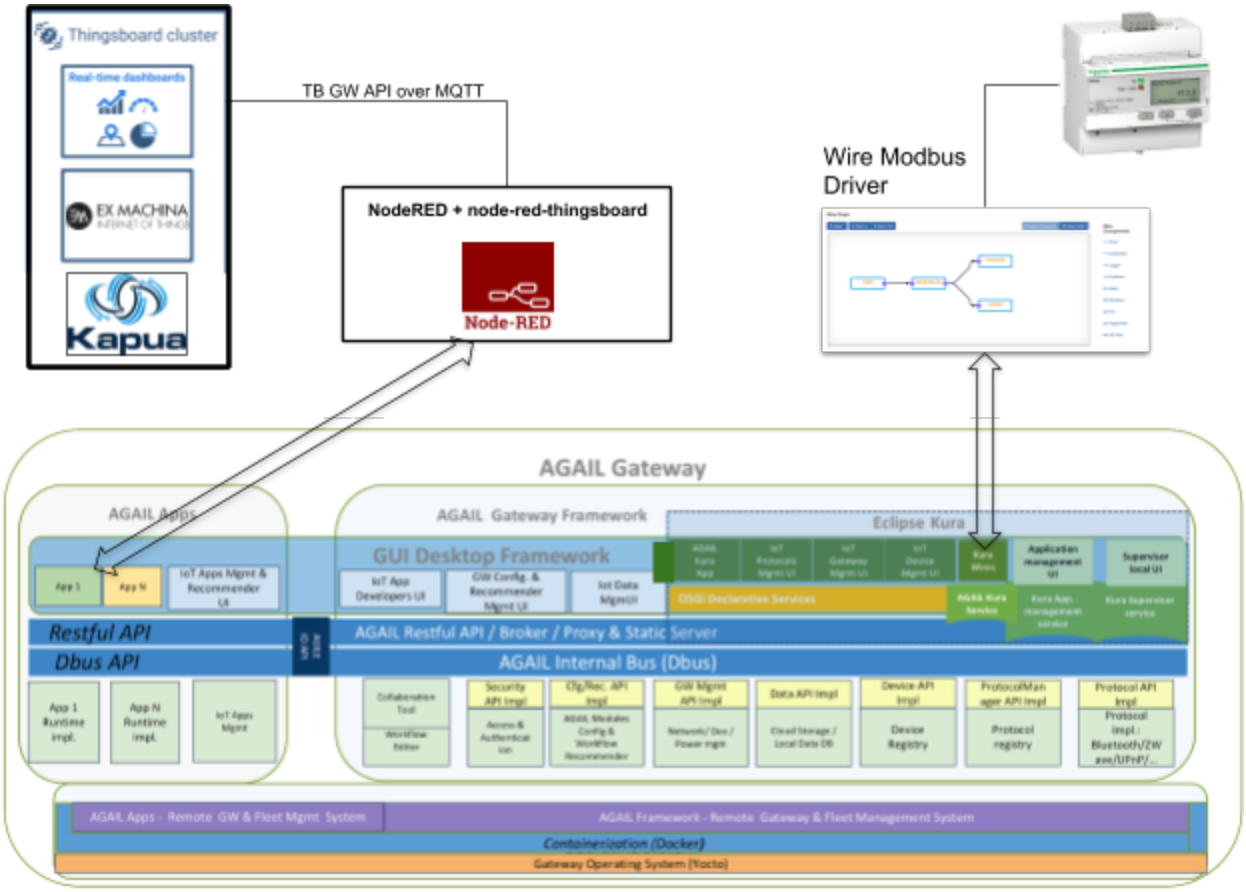
We will investigate this further at future phases of the project.

https://github.com/exmgr/Dockerized-Thingsboard-Gateway

[3] https://docs.resin.io/learn/develop/multicontainer/

Scenario #5 - NodeRED as a TB-GW

TB-GW could be completely replaced by an equivalent software as long as its core functionality is maintained. An open-source **node-red-thingsboard** [4] node implementation is already published that could be improved and combined with the existing NodeRED server of the Eclipse AGAIL. This architecture is trying to build on the Agile IoT “Rapid prototyping using the Developer UI” concept.



Challenges

We could not find a way to read Kura telemetry data from AgileIoT image (v0.3.5) bundled Node-RED.

[4] https://github.com/flix90/node-red-thingsboard

Scenario #6 - TB-GW as an AGAIL SDK app

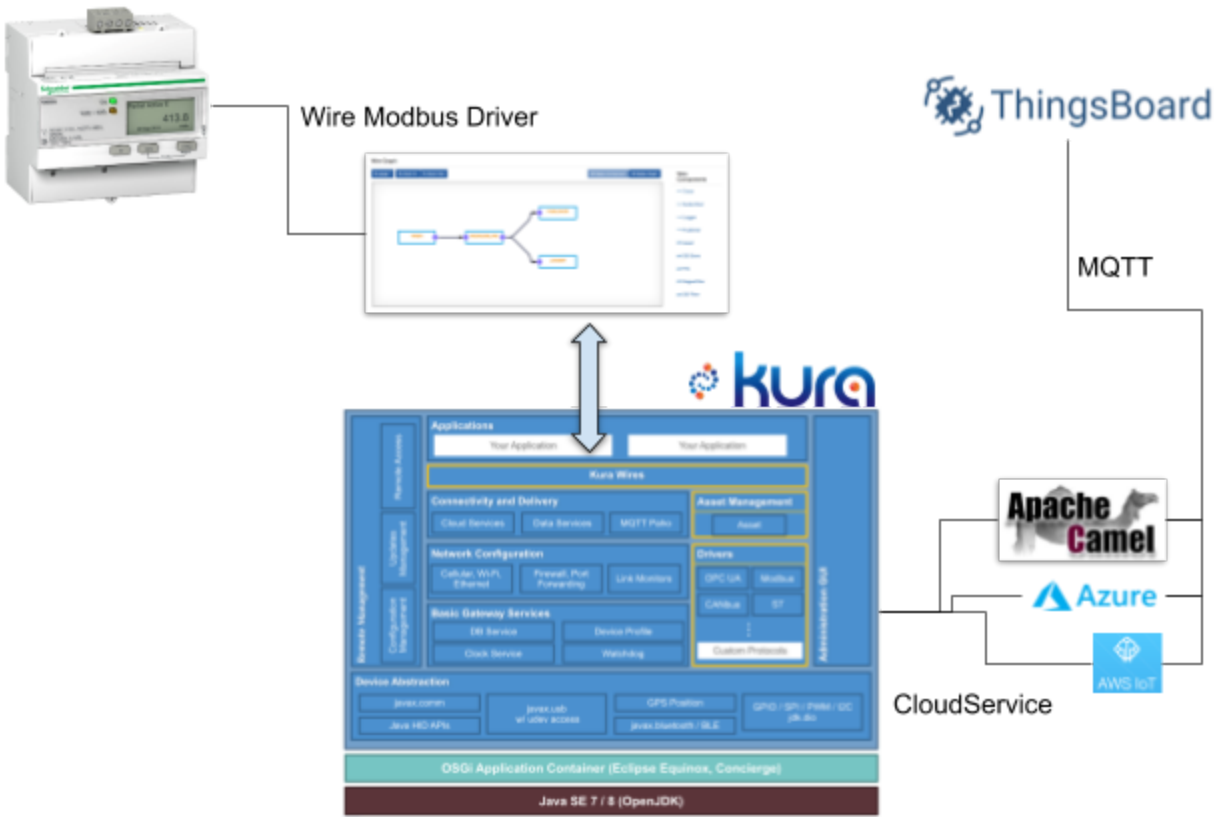
A straight forward scenario is to re-implemented from scratch the TB-GW in node.js as an AGAIL SDK native application. In this scenario an additional custom Agile protocol driver for modbus should be built using one of the readily available modbus libraries. This way the app could use Agile's devices concept to provide the user with an easy to use configuration interface that is built-in in Agile (os.js based Agile-OS web interface).

Challenges

It seems Eclipse AGAIL is still in its infancy, therefore it is too risky, at this point in time, to build a new implementation from scratch of TB-GW in node.js with AGAIL SDK. Moreover, building a new protocol implementation for AGAIL poses a challenge, as there is little information available for external parties and the relevant wiki pages are not yet completed
(e.g. http://agile-iot.eu/wiki/index.php?title=Dev_Protocol)

Scenario #7 - Kura CloudService compatible broker as TB-GW

This is an alternative architecture approach which further deviates from the original goals of this project, therefore will not be further explored, however for reasons of completeness it is briefly documented bellow. Cloudservice telemetry message payloads between Kura and TB are not compatible therefore some kind of mapping has to take place. Currently TB server community edition (v1.4) does not support alternative payloads, therefore this problem has to be solved on the gateway side or is some kind of middle message broker. Apache Camel, could be used for such transformation.



Conclusion

After comparing the cons and pros of each scenario we concluded that the best approach for our needs is either scenario #2 or #3. We have implemented and published both approaches and we will continue to evaluate them as we progress in the project, in order to decide on the final one that efficiently addressing the project's goals.