**SMART FACTORY ACCELERATOR - THINGSBOARD**

**INTRODUCTION**

* Smart-Factory-Accelerator is a real-time analytics and predictive maintenance solution built on top of ThingsBoard and QlikSense, enabled by AI and is distributed under the Apache license.

**DESCRIPTION**

* With the emergence of Fourth Industrial Revolution, or Industry 4.0, a smart factory has turned into a reality. The development of technology to collect and analyze data from sensors in real time provides the ability to monitor and proactively resolve issues that come up in the production process. It has become possible to extremely easily keep track of the performance levels of the equipment well before it breaks down or technical failure, causing a hindrance in the overall productivity of the factory. The Smart Factory Accelerator (SFA) is designed to initiate a result oriented approach that facilitates prevention of any possible collapse in the machinery that would cause a negative impact on production. The solution aims to put in place a monitoring system that analyzes key causes of failure so that factories can make Zero Downtime a reality.
* Using The Smart Factory Accelerator, the sensors, devices, people and process become a part of the connected ecosystem, where it is employed. All the relevant data is aggregated, analyzed and thus acted upon. As modern factories are populated with complex and expensive equipment, it becomes essential to get a clear picture of what is happening in the factory without any delay. Therefore, identifying bottlenecks in processing, taking proactive steps to deal with changing situations, and increasing awareness of the operational system are key to sensor-based monitoring. The salient features of Smart Factory Accelerator are its IoT integration with ThingsBoard and real-time alerting system with alert notification to employee’s Telegram Bot as well as on device manager of ThingsBoard, providing real time updates of any machine malfunction to the operator to initiate a corrective measure.

**TECHNICAL FUNCTIONALITY**

The Smart Factory Accelerator is an **Industry 4.0** compatible solution, providing :-

\* IoT integration with ThingsBoard using MQTT publish/subscribe protocol

\* Provisioning devices, assets and define relations between them

\* Analyze incoming telemetry and trigger Real-time Alerting mechanism

\* Rule Chain Development for specific functionalities

\* Predictive maintenance and Anomaly Detection for Smart Factories using Kafka and Advanced Analytics

\* Design dynamic and responsive Dashboards using customizable widgets

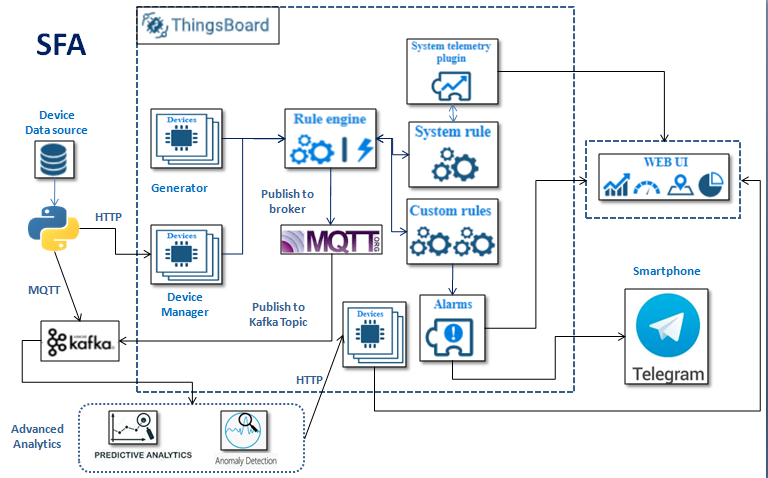
\* Managing users and assigning entities to relevant customers

\* Device Authentication and Security

\* ThingsBoard integration with Telegram Bot for alert notifications on smartphone

**DEVELOPING SMART FACTORY ACCELERATOR**

1. **ARCHITECTURE :**

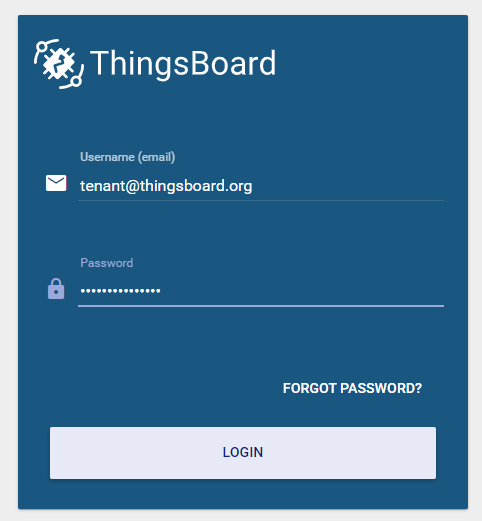


* **Real-time simulator** is run using Generator Action Rule Node of Rule Engine which generates various telemetry information like temperature, pressure, humidity, acoustic emission, rpm etc for different devices. **Also**, we run a **Real-time telemetry application** from a Python IDE, which uses **HTTP** protocol to send message payload to the ThingsBoard device.
* **ThingsBoard Device topics** consume both the types of data, and assign them to the Latest Telemetry feature of the Devices configured.
* **Rule Engine** performs the necessary action on the real-time data using various rules and sends the data for Real-time Data Analysis and Alerting. Also, helps in sending data outside ThingsBoard to external systems.
* **Data Adapter** running in background consumes MQTT data, adapts to the data schema and sends the data for further processing to Advanced Analytics services via Kafka.
* **Advanced Analytics** is done involving RUL Prediction of each engine and Anomaly Detection whose results are sent back to ThingsBoard using HTTP protocol.
* **Real-time Dashboards** showing real-time readings at regular intervals.
* **Alerts** (on Dashboard + Smartphone-Telegram Bot) are triggered when any machine parameter crosses their respective specific threshold.

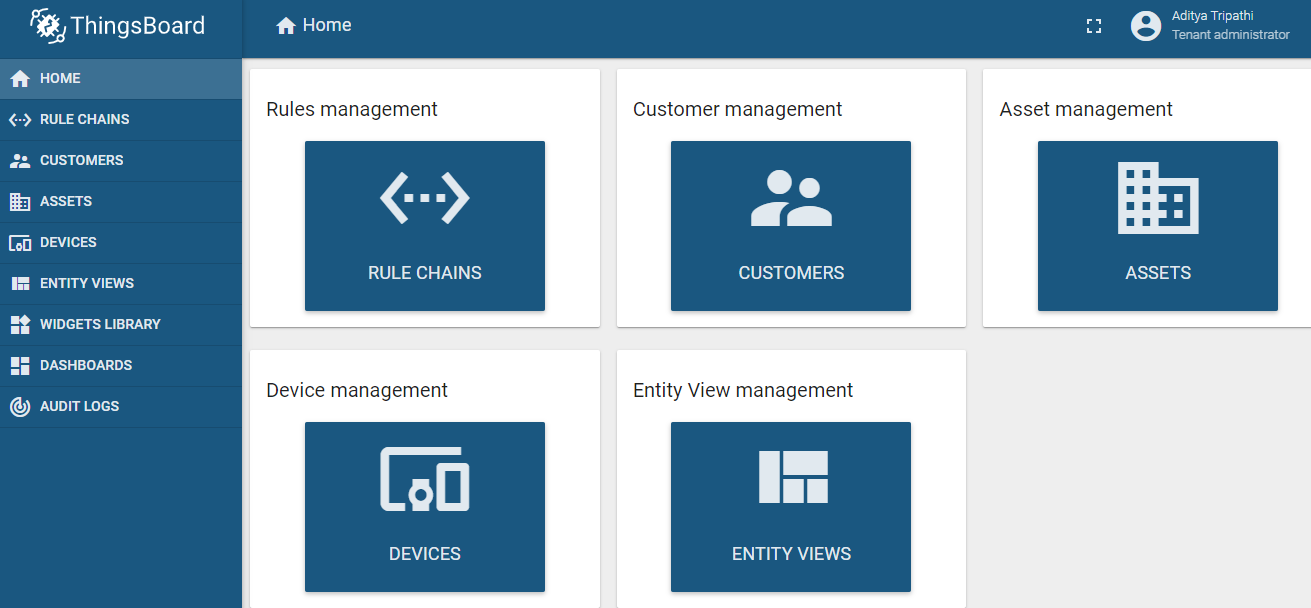
**Note: Descriptive Dashboards will be developed using QlikSense as done in QSFA.**

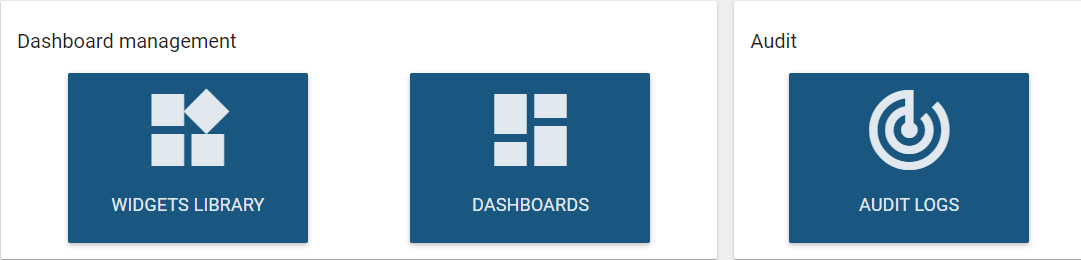
1. **IMPLEMENTATION STEPS**

* Open Command Prompt in Admin mode and start Thingsboard service after going to the directory where it is installed. Open the Web UI link that was available in the installation steps and enter Tenant credentials.

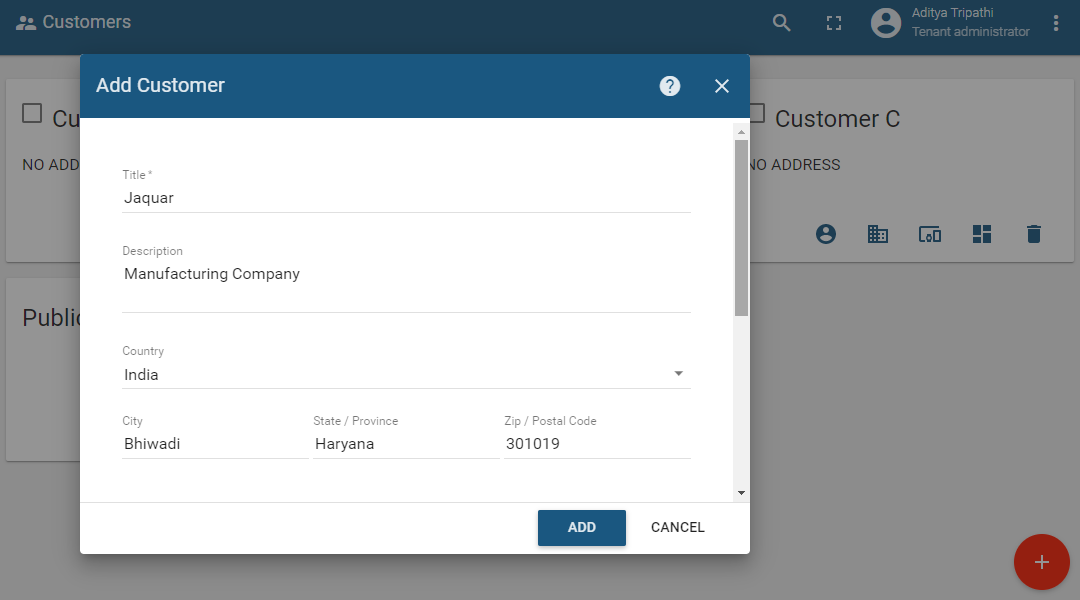


* When logged in, you’ll see the ThingsBoard Tenant Home Page.

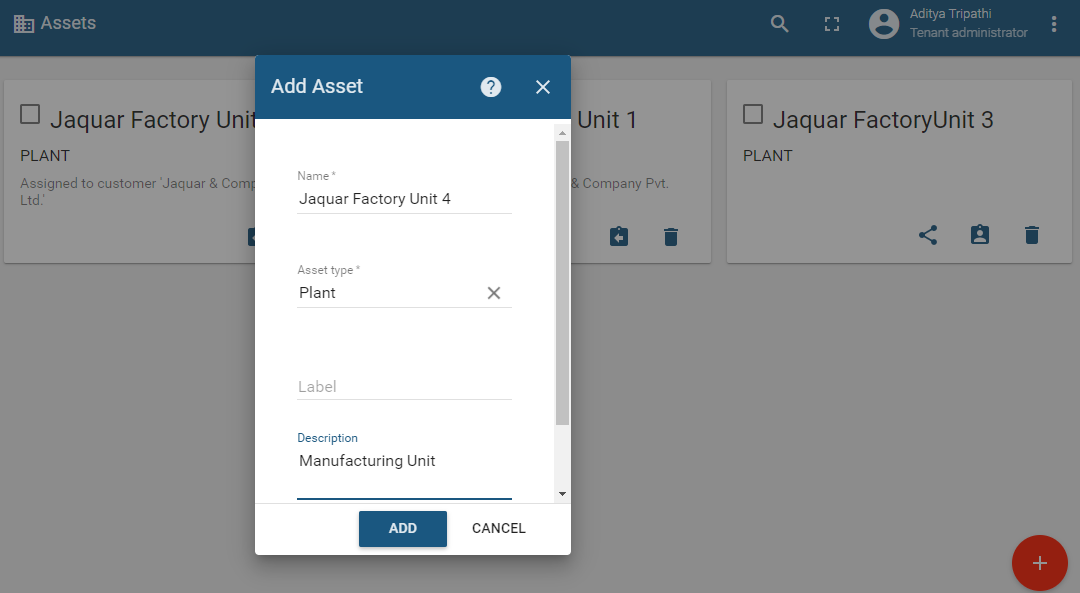




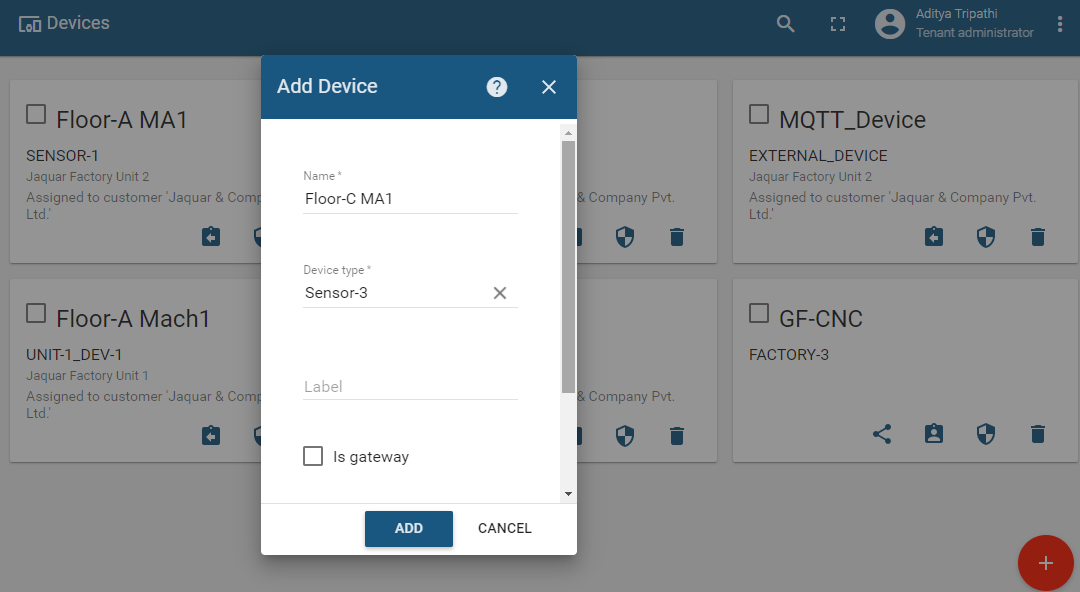
* **Customer Management:** Create customer by clicking on the ‘+’ icon in the bottom-right of the platform, and Add new Customer, say ‘**Jaquar**’.



* After Customer is created, then go to **Assets** section (on left-pane) to create Assets.

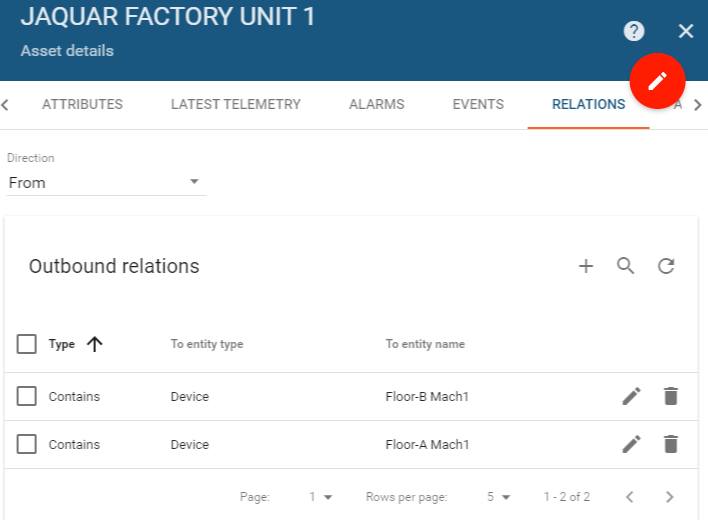


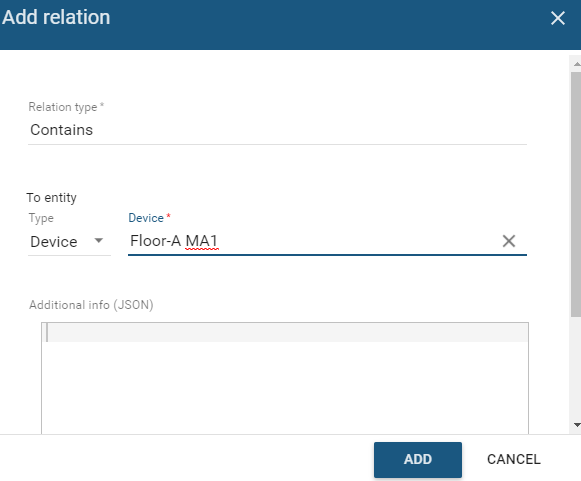
* After Asset gets created, go to **Devices** section (on left-pane).

****

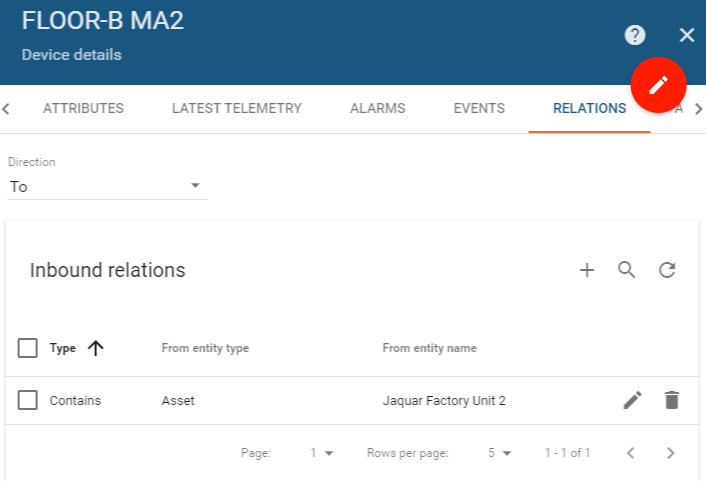
* After creating **Device**, go back to **Assets** and click on the Asset you created and **follow** the necessary **steps** to create **relation** between Assets and Devices:

{Asset-name} **>>** {Relations} **>>** {Direction: **From**} **>>** {click on **‘+’** of **Outbound relations** section to include devices} **>>** {Choose Relation Type : **Contains**} **>>** {Choose Entity Type : **Device**} **>>** {Choose **Device-name**} **>>** Click **Add** button

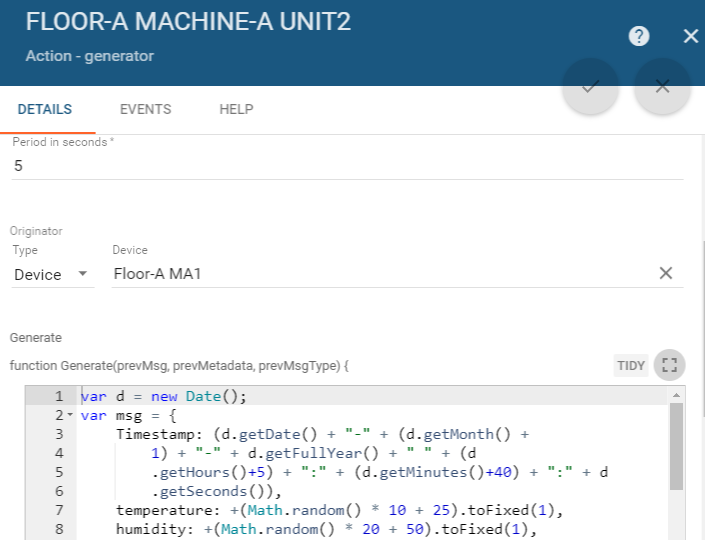
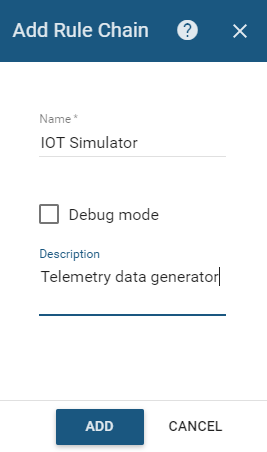


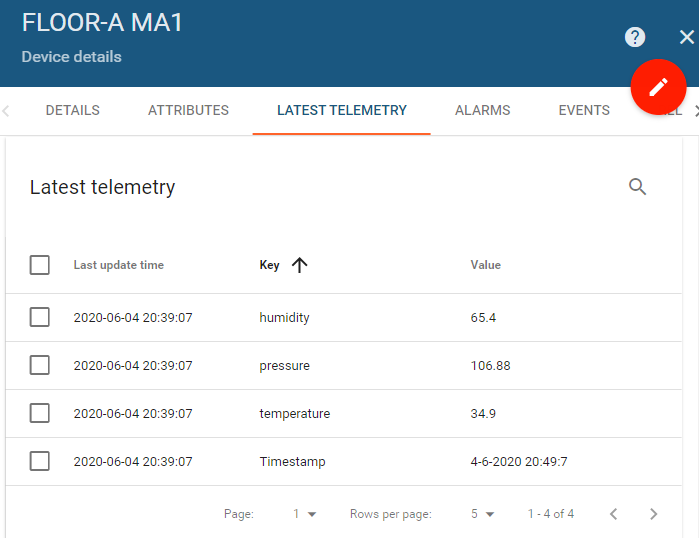


* To check if a relation has been established or not you can go to the Device section and choose the Device. Go to **Relations**, Select **Direction** as **‘To’** , and confirm as required.



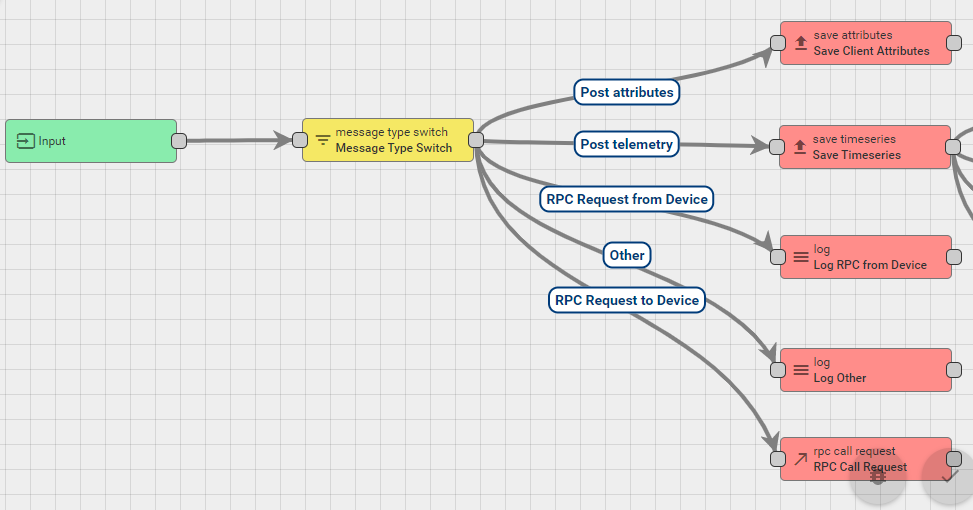
* Now since we are working on **IOT simulator**, it’s time to **add Telemetry** information in our device. Go to **Rule Engine** section, and create new Rule Chain for IOT data Generator. You may now notice Telemetry data incoming in your device by checking in the Latest Telemetry section.



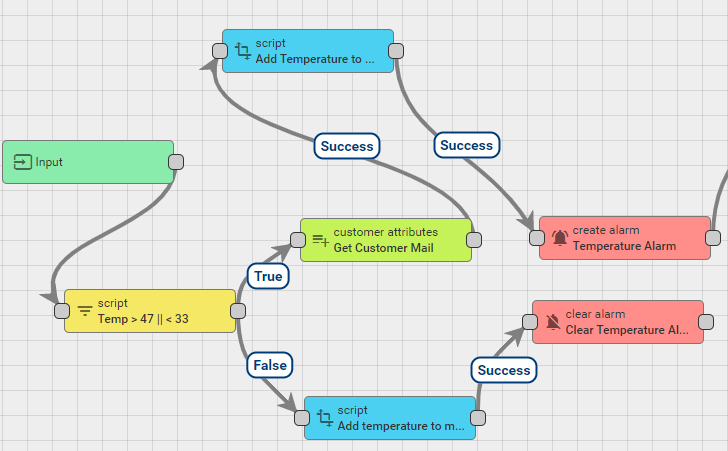


* After creation of **Generator**, send data to **Root Rule Chain** which identifies **type of Telemetry message request** and saves in internal database.

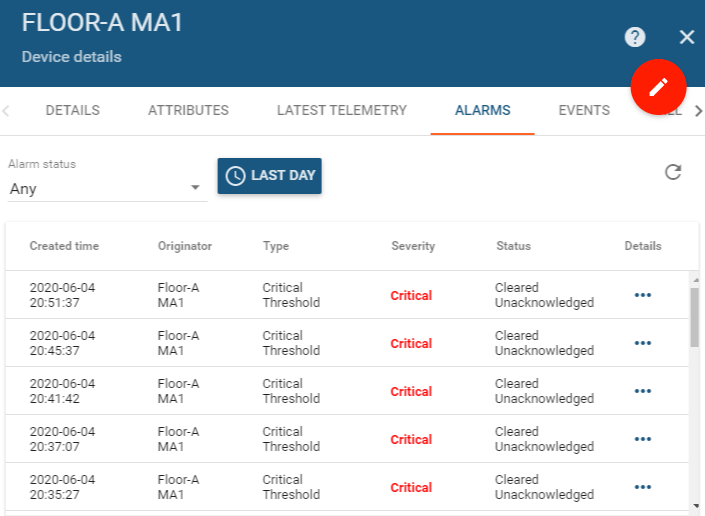


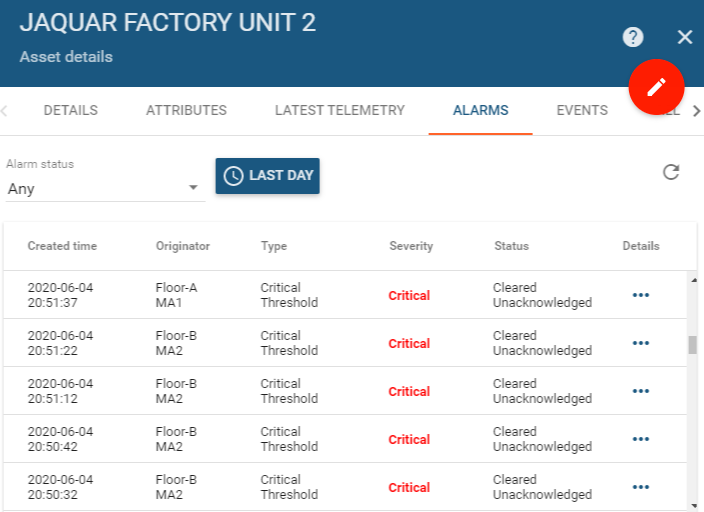


* **Working with Alarms**: We will now create **Alarms** which send **alert notifications** whenever Telemetry values go **out of range**. For this, we will first create a **Filter script** node which will verify the telemetry conditions if they are in interval or not. If they are inside interval then, Alarm is cleared (**False**), otherwise Alarm is created (**True** – since we are creating Alarms).



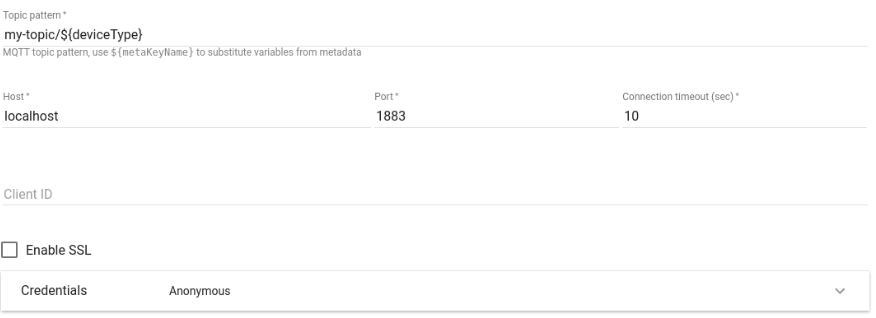
* We can check alerts in our **Devices’/Assets’** Alarms section.





* Sending **telemetry** data to **external MQTT broker:** Here, we can publish incoming message payload to the topic of the configured MQTT broker with QoS **AT\_LEAST\_ONCE**.

**Configuration:**



* **Topic pattern** - can be a static string, or pattern that is resolved using Message Metadata properties. For example ${deviceType}.
* **Host** - MQTT broker host.
* **Port** - MQTT broker port.
* **Connection timeout** - timeout in seconds for connecting to MQTT broker.
* **Client ID** - optional client identifier used for connecting to MQTT broker. If not specified, default generated clientId will be used.
* **SSL Enable/Disable** - enable/disable secure communication.
* **Credentials** - MQTT connection credentials. Can be either Anonymous, Basic or PEM.

Different **Authentication credentials** are supported for external MQTT broker:

* **Anonymous** - no authentication
* **Basic** - username\password pair is used for authenticating
* **PEM** - PEM certificates are used for Authentication

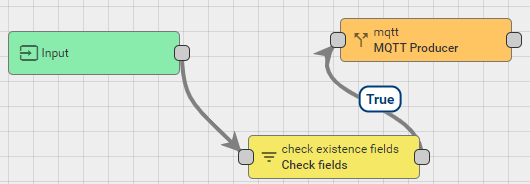
If **PEM credentials** type is selected, the following configuration should be provided:

* CA certificate file
* Certificate file
* Private key file
* Private key password

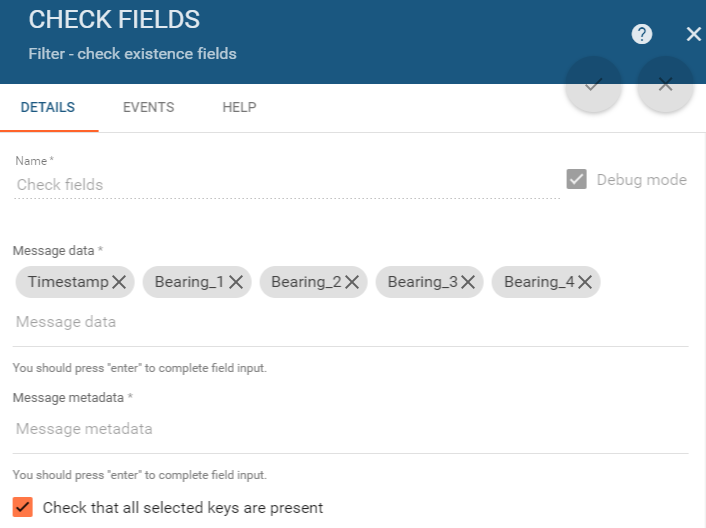
**Published body** - Node will send full Message payload to the MQTT topic. If required, Rule Chain can be configured to use chain of Transformation Nodes for sending correct Payload to the MQTT broker.

In case of successful message publishing, original Message will be passed to the next nodes via **Success** chain, otherwise **Failure** chain is used.

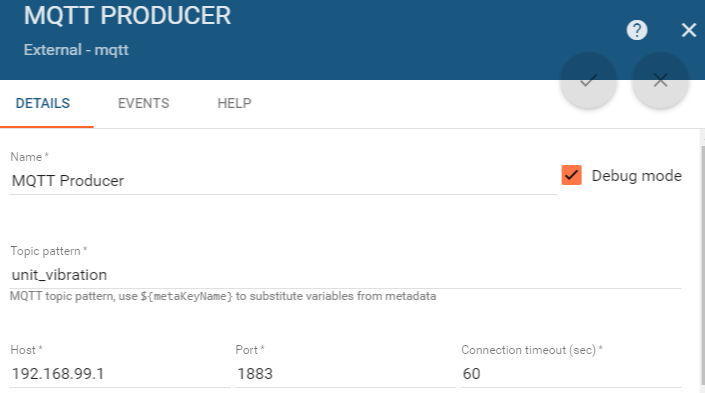
In our case, we will select the fields from **payload message** and **metadata** that we require for our analysis, using **‘Check Existence Fields’** rule node and then if condition holds **True**, we connect to the **external MQTT broker** rule node.



**Check Existence Fields Configuration:**

****

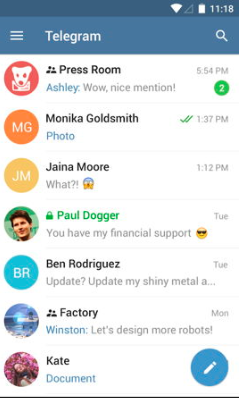
**External MQTT broker**

****

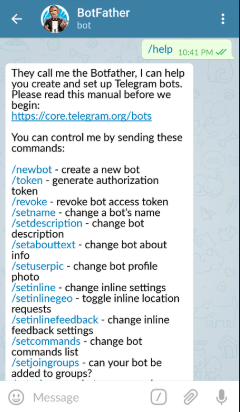
* **Sending Notification Alerts to Telegram User on Smartphone:**

**Steps Followed** –

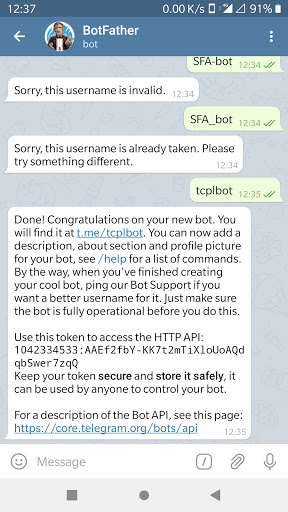
* 1. Install **Telegram** on your Smartphone

****

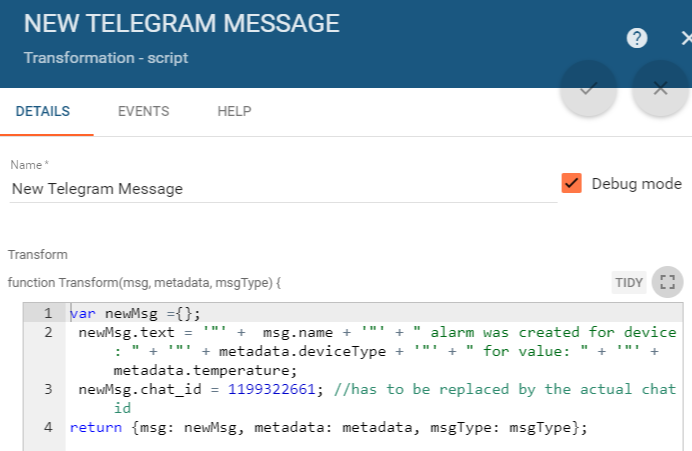
* 1. Search for **BotFather** (Telegram Bot)
  2. Send message to **BotFather** to **create new Bot**

****

* 1. **Create new username** for the new bot. An **authentication token** is generated which will be used in the **REST API Call endpoint URL** feature.



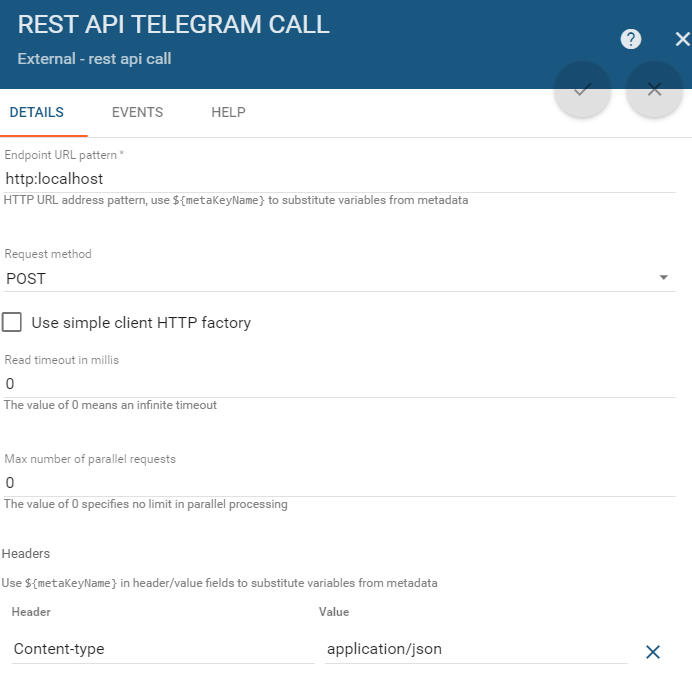
* 1. Go to **ThingsBoard Rule Engine** and add a **script** which contains the message body to be sent (**message name, device name, sensor value**).



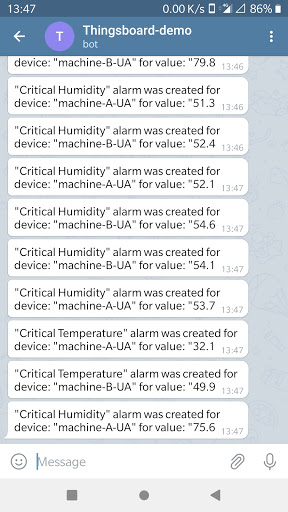
* 1. Get the ‘**Chat ID**’ of Telegram bot by sending a message to the bot. Refresh the web browser URL link of the bot for viewing the message schema, and search for ‘**id**’ key inside ‘**chat**’ parent key.



* 1. In Rule Engine, add external-type Rule node of ‘**REST API Call**’ after the script node and enter the **endpoint URL** containing the **authentication token**, with **Request type** as **POST** and **header** as ‘**content-type**’ with value ‘**application/json**’. (**Authentication URL has been omitted for security reasons**)



* 1. Check **notifications** on your smartphone in **Telegram app**.



***(Dashboard implementation covered in separate document)***