Mục lục

[CHƯƠNG 1. ARCHITECTURE 4](#_Toc70363703)

[1.1 Overview 4](#_Toc70363704)

[1.1.1 ThingsBoard Transport 4](#_Toc70363705)

[1.1.2 ThingsBoard Core 4](#_Toc70363706)

[1.1.3 ThingsBoard Web UI 5](#_Toc70363707)

[1.1.4 ThingsBoard Rule Engine 5](#_Toc70363708)

[1.2 Monolithic and Microservices Architecture 6](#_Toc70363709)

[1.3 SQL, NoSQL, Hybrid database 7](#_Toc70363710)

[CHƯƠNG 2. ENTITIES AND RELATIONS 8](#_Toc70363711)

[2.1 Entities 8](#_Toc70363712)

[2.1.1 Tenants 8](#_Toc70363713)

[2.1.2 Customers 8](#_Toc70363714)

[2.1.3 Users 8](#_Toc70363715)

[2.1.4 Devices 8](#_Toc70363716)

[2.1.5 Assets 9](#_Toc70363717)

[2.1.6 Entity Views (EVs) 9](#_Toc70363718)

[2.1.7 Alarms 10](#_Toc70363719)

[2.1.8 Dashboard 10](#_Toc70363720)

[2.2 Relations 11](#_Toc70363721)

[CHƯƠNG 3. DATA TYPES 12](#_Toc70363722)

[3.1 Attribute 12](#_Toc70363723)

[3.1.1 Attribute names 12](#_Toc70363724)

[3.1.2 Attribute types 12](#_Toc70363725)

[3.1.3 Attribute persistence 13](#_Toc70363726)

[3.1.4 Data Query API: 13](#_Toc70363727)

[3.1.5 Data visualization: 14](#_Toc70363728)

[3.1.6 Rule Engine: 14](#_Toc70363729)

[3.2 TIME-SERIES DATA (TELEMETRY DATA) 14](#_Toc70363730)

[3.2.1 Tính năng cơ bản 14](#_Toc70363731)

[3.2.2 Data Upload API 14](#_Toc70363732)

[3.2.3 Data Visualization 14](#_Toc70363733)

[3.2.4 Data Storage 14](#_Toc70363734)

[3.2.5 Data Retention 14](#_Toc70363735)

[3.2.6 Data durability 15](#_Toc70363736)

[3.2.7 Rule Engine 15](#_Toc70363737)

[3.2.8 Data Query REST API 15](#_Toc70363738)

[3.2.9 WebSocket API 15](#_Toc70363739)

[CHƯƠNG 4. PROFILES 17](#_Toc70363740)

[4.1 Tenant Profile 17](#_Toc70363741)

[4.1.1 Entity Limits 17](#_Toc70363742)

[4.1.2 API Limit & Usage 17](#_Toc70363743)

[4.1.3 Rate Limits 17](#_Toc70363744)

[4.1.4 Processing in isolated ThingsBoard Core and Rule Engine Container 17](#_Toc70363745)

[4.2 Device Profile 18](#_Toc70363746)

[4.2.1 Rule Chain 18](#_Toc70363747)

[4.2.2 Queue Name 18](#_Toc70363748)

[4.2.3 Transport Configuration 18](#_Toc70363749)

[4.2.4 Alarm Rules 18](#_Toc70363750)

[CHƯƠNG 5. RULE ENGINE 19](#_Toc70363751)

[5.1 Các khại niệm quan trọng 19](#_Toc70363752)

[5.1.1 Rule Engine Message 19](#_Toc70363753)

[5.1.2 Rule Node 19](#_Toc70363754)

[5.1.3 Rule Node Relation 19](#_Toc70363755)

[5.1.4 Rule Chain 19](#_Toc70363756)

[5.1.5 Message Processing Result 20](#_Toc70363757)

[5.1.6 Rule Engine Queue 20](#_Toc70363758)

[5.1.7 Queue submit strategy 20](#_Toc70363759)

[5.1.8 Queue processing strategy 21](#_Toc70363760)

[5.1.9 Default queue 21](#_Toc70363761)

[5.2 Predefined Message Types 21](#_Toc70363762)

[5.3 Rule Node Types 21](#_Toc70363763)

[5.4 Architecture 21](#_Toc70363764)

[5.5 Capabilities 22](#_Toc70363765)

[CHƯƠNG 6. OTHER FEATURES 23](#_Toc70363766)

[6.1 RPC (Remote Procedure Calls) capabilities 23](#_Toc70363767)

[6.2 Device Connectivity Status 24](#_Toc70363768)

[6.3 Device Claiming 24](#_Toc70363769)

[6.4 Provisioning Devices 26](#_Toc70363770)

[6.4.1 Device Profile Configuration 26](#_Toc70363771)

[6.4.2 Provision Device API 26](#_Toc70363772)

[6.5 Administration UI 26](#_Toc70363773)

Danh mục hình ảnh

[Hình 1.1. Kiến trúc tổng quan Thingsboard 5](#_Toc70364116)

[Hình 1.2. Hệ thống phân cấp Actor 6](#_Toc70364117)

[Hình 1.3. Chế độ Clustering 7](#_Toc70364118)

[Hình 1.4. Kiến trúc Monolithic 8](#_Toc70364119)

[Hình 1.5. Kiến trúc Microservices 8](#_Toc70364120)

[Hình 2.1. Mối liên hệ giữa Admin, Tenant và Customer 9](#_Toc70364121)

[Hình 2.2. Mối liên hệ của Asset và Device 12](#_Toc70364122)

[Hình 3.1. Server-side Attribute 13](#_Toc70364123)

[Hình 3.2. Shared Attribute 14](#_Toc70364124)

[Hình 3.3. Client-side Attribute 14](#_Toc70364125)

[Hình 5.1. Ví dụ Rule Chain 20](#_Toc70364126)

[Hình 5.2. Kiến trúc Rule Engine 23](#_Toc70364127)

[Hình 6.1. Client-side RPC 24](#_Toc70364128)

[Hình 6.2. One-way server-side RPC 24](#_Toc70364129)

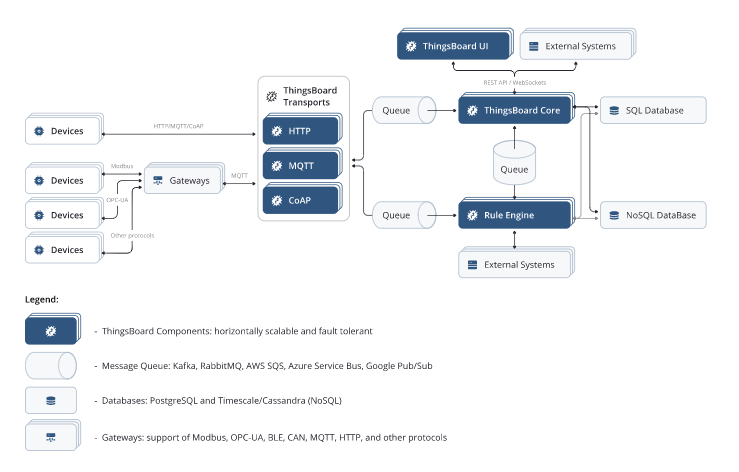
[Hình 6.3. Two-way server-side RPC 24](#_Toc70364130)

[Hình 6.4. Device-side key claiming 26](#_Toc70364131)

[Hình 6.5. Server-side key claiming 26](#_Toc70364132)

# ARCHITECTURE

## Overview



Hình 1.1. Kiến trúc tổng quan Thingsboard

### ThingsBoard Transport

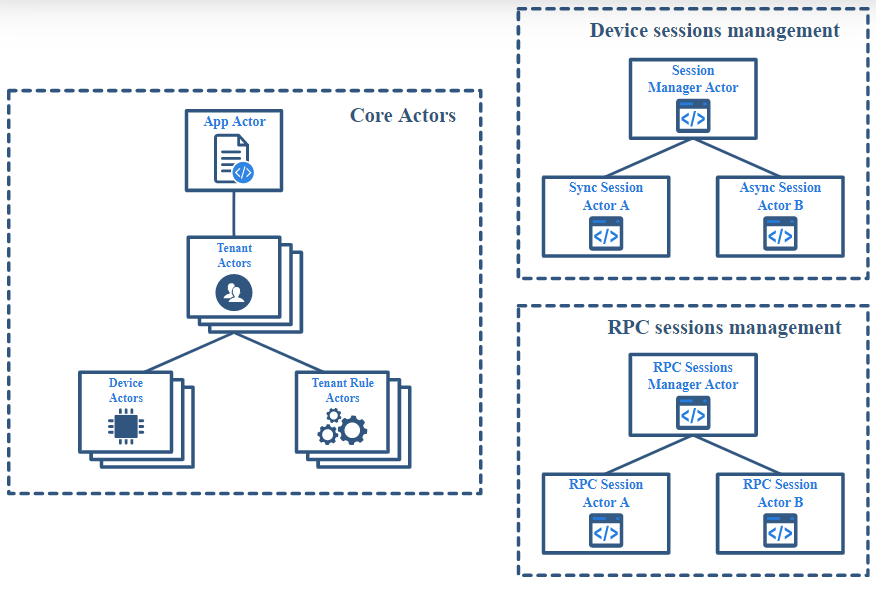
ThingsBoard cung cấp các API MQTT, HTTP, CoAP mà các *Device* được hỗ trợ. Mỗi API được cung cấp 1 server riêng và là một phần của lớp Transport của ThingsBoard. Transport của MQTT được cung cấp Gateway API. Bản tin nhận được từ *Device* sed được đẩy vào *Message Queue.* Bản tin sẽ được báo cho *Device* là đã truyền khi *Message Queue* đã ack bản tin đó.

### ThingsBoard Core

Thành phần xử lý các hoạt động của *REST API* và *WebSocket*, đồng thời chịu trách nhiệm lưu dữ liệu ngày tháng về các hoạt động và giám sát trạng thái của *Device*.

Hệ thống *Actor* được sử dụng để cho phép việc xử lý đồng thời các bản tin từ *Device*. Một vài hoạt động của các loại *Actor* như sau

* *App Actor:* quản lý các *Tenant Actor,* hoạt động nhất thời từ *Actor* này luôn được giữ trong bộ nhớ.
* *Tenant Actor:* quản lý các *Tenant Device* và *Rule Chain Actor*, hoạt động nhất thời từ *Actor* này luôn được giữ trong bộ nhớ.
* *Device Actor:* duy trì trạng thái của *Device*. Bộ nhớ đệm của *Attribute* của *Device* được giữ để duy trì hiệu suất hoạt động. *Actor* này được tạo ra khi xử lý bản tin đầu tiên từ *Device*, và kết thúc khi không có bản tin từ *Device* trong 1 khoảng thời gian nhất định.
* *Rule Chain Actor:* xử lý các bản tin đến và phân chia chúng cho *Rule Node Actor,* hoạt động nhất thời của *Actor* này luôn được lưu trong bô nhớ.
* *Rule Node Actor:* xử lý các bản tin đến và gửi kết quả cho *Rule Chain Actor,* hoạt động nhất thời của *Actor* này luôn được lưu trong bộ nhớ.



Hình 1.2. Hệ thống phân cấp Actor

### ThingsBoard Web UI

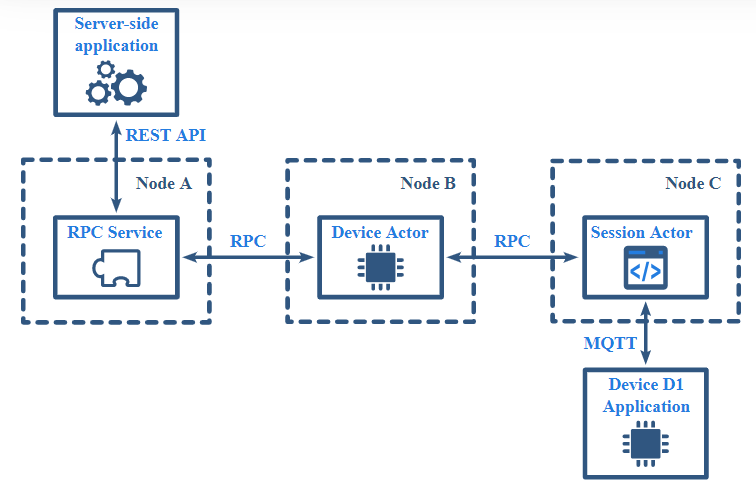
ThingsBoard cung cấp các thành phần sử dụng *Express.js Framework* làm *Web UI* tĩnh. Các thành phần này không được đánh dấu và không cần điều chỉnh nhiều. *Web UI* tĩnh bao gồm các gói ứng dụng, một khi được sử dụng, chúng sẽ dùng *REST API, WebSocket API* được hỗ trợ bởi ThingsBoard Core.

### ThingsBoard Rule Engine

Thành phần chịu trách nhiệm cho việc xử lý các bản tin đến, sử dụng các *Actor* (được nêu phần trước). Các nút *Rule Engine* có thể tham gia vào mạng *Cluster*, với mỗi nút chịu trách nhiệm cho 1 phần các bản tin đến.

Chế độ *Clustering*:

* *Service Discovery* (sử dụng Zookeeper): tất cả các nút trong ThingsBoard đều được định danh và đăng ký như 1 thành phần không bền vững trong Zookeeper và có bộ nhớ đệm để xác định các nút liên quan.
* *Consistent Hashing:* bản tin từ *Device A* sẽ được nhận ở một nút nào đó và được chuyển đi theo *Device ID*, cần có sự qui định trước cho mạng, tuy nhiên sẽ cung cấp khả năng xử lý các bản tin ở các *Device Actor* tương ứng trong một mạng nhất định, và sẽ có một vài ưu điểm: nâng cao khả năng xử lý bộ đệm, tránh sử dụng điều kiện với bản tin, cho phép gọi từ *Server-side API* thông qua *Device ID*.



Hình 1.3. Chế độ Clustering

*Rule Engine* chuyển các dữ liệu đến vào *Queue* và chỉ ack bản tin sau khi bản tin đó đã được xử lý xong, có nhiều cách để kiểm soát thứ tự, việc xử lý bản tin và tiêu chí ack của bản tin.

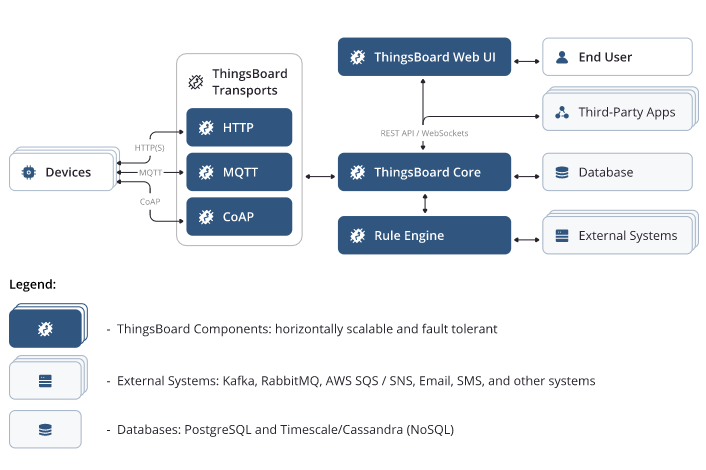
*Message Queue:* Với các *Queue* này, ThingsBoard có thể sử dụng *Back-pressure* (quan trọng khi bộ nhớ đạt đỉnh) và *Load Balancing*, được cung cấp *Abstraction Layer* để duy trì *Topic* và *Topic Partition* (vách ngăn). Mỗi *Topic* có thể có nhiều *Topic Partition. Producer* qui định sử dụng *Partition* nào dựa vào *Entity ID*. *Consumer* sử dụng Zookeeper và thuật toán để xác định danh sách các *Partition* mà *Consumer* đó kết nối tới. Các *Topic* được ThingsBoard sử dụng:

* Tb\_transport.api.requests: gửi các lệnh API để cho phép gửi dữ liệu từ *Transport* lên *Core*.
* Tb\_transport.api.responses: nhận kết quả xác nhận từ *Core* đến *Transport*.
* Tb\_core: đưa bản tin từ *Transport* hoặc *Rule Engine* lên *Core.*
* Tb\_rule\_engine: đưa bản tin từ *Transport* hoặc Core đến *Rule Engine*.

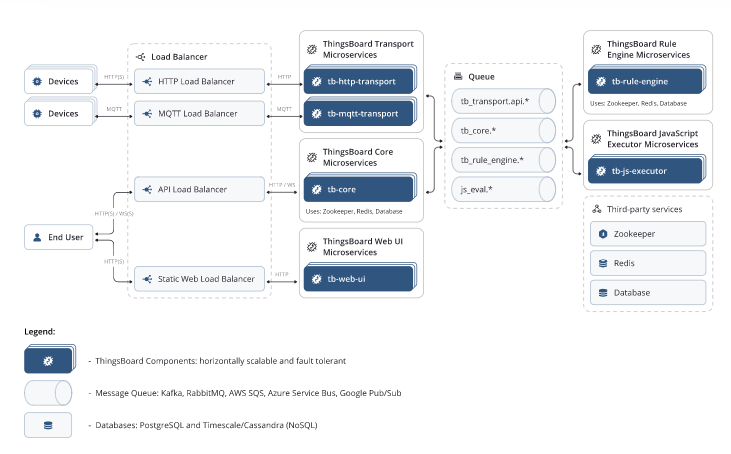
*Rule Engine* có 2 chế độ hoạt động: *Shared* và *Isolated.* Trong chế độ *Shared,* *Rule Engine* xử lý bản tin từ nhiều *Tenant*, trong chế độ *Isolated*, thì xử lý từ 1 Tenant.

## Monolithic and Microservices Architecture

Kiến trúc *Monolithic* được sử dụng nhiều do yêu cầu ít về khả năng hỗ trợ, kiến thức và phần cứng để áp dụng, và việc bảo trì đơn giản. Nếu cần sử dụng đến rất nhiều *Device* thì nên sử dụng *Microservices* với việc triển khai và nhiều tình huống phức tạp nên cần việc phân chia nhỏ để bảo vệ hệ thống khỏi: lượng thông tin vượt quá dự đoán, việc cấu hình sai *Rule Chain*, 1 *Device* kết nối quá nhiều thiết bị cùng lúc do lỗi và nhiều tình huống khác.



Hình 1.4. Kiến trúc Monolithic



Hình 1.5. Kiến trúc Microservices

## SQL, NoSQL, Hybrid database

ThingsBoard sử dụng *Database* để lưu trữ *Entity* và *Telemetry Data*.

* SQL: Lưu trữ tất cả dữ liệu và Enity lên SQL. ThingsBoard hỗ trợ sử dụng PostgreSQL.
* NoSQL: Lưu trữ tất cả dữ liệu và Entity lên noSQL. ThingsBoard hỗ trợ sử udngj với Cassandra.
* Hybrid (PostgreSQL + noSQL): Lưu trữ Entity lên PostgreSQL và dữ liệu lên noSQL. ThingsBoard hỗ trợ cả Cassandra và TimescaleDB)

# ENTITIES AND RELATIONS

## Entities

### Tenants

*Tenant* được coi như là 1 *business-entity* riêng biệt, là 1 cá nhân hoặc tổ chức sở hữu hoặc sản xuất *Device* hoặc *Asset*. *Tenant* có thể có nhiều *Tenant Administrator* và hàng triệu *Customer*, *Device* và *Asset.*

ThingsBoard hỗ trợ *Multitenacy*, có thể sử dụng *ThingsBoard Tenant* như 1 *Entity* riêng biệt như 1 cá nhân hay tổ chức sở hữu thiết bị.

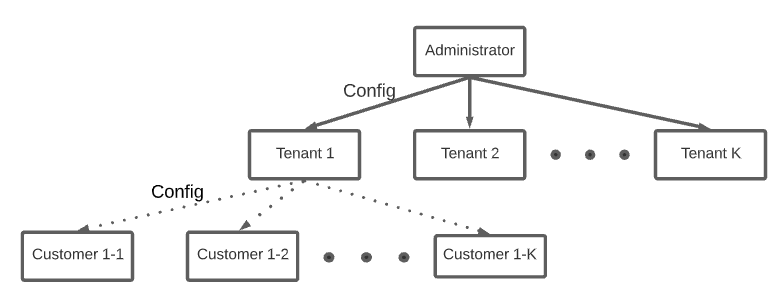
*System administrator* có thể tạo ra các *Tenant Entity* và có thể tạo ra các *User* có *Tenant administrator* đối với mỗi *Tenant*.

*Tenant administrator* có thể thực hiện các thao tác sau (đều có thể làm được sử dụng *Rest API*):

* Cấp phép và quản lý *Device*
* Cấp phép và quản lý *Asset*
* Tạo và quản lý *Customer*
* Tạo và quản lý *Dashboard*
* Cấu hình *Rule Engine*
* Thêm hoặc chỉnh các *Widget* ban đầu sử dụng *Widget Library*.

### Customers

*Customer* là 1 *business-entity* riêng biệt, có thể có nhiều *User* và hàng triệu *User, Device, Asset.*



Hình 2.1. Mối liên hệ giữa Admin, Tenant và Customer

### Users

*User* được phép truy cập *Dashboard* và quản lý *Entity*

### Devices

*Entity* cơ bản có thể lấy dữ liệu từ xa và sử dụng các lệnh RPC như cảm biến, thiết bị chấp hành, khóa đóng cắt. ThingsBoard cho phép các tính năng quản lý sử dụng *Web UI* và *REST API:*

* Thêm và xóa các *Device*: cho phép thêm hoặc xóa các *Device* khỏi ThingsBoard
* Quản lý chứng nhận của *Device:* để có thể gửi các hỗ trợ về *Access Token* và chứng chỉ X.509 dựa trên chứng nhận
* Lấy ID của *Device:* có thể copy ID của thiết bị sử dụng nút bấm trên màn hình.
* Kết nối *Device* với *Customer*: để *Customer* có thể lấy dữ liệu trên *Device* sử dụng *REST API* và *Web UI*.
* Truy cập *Device Attribute*, *Device Telemetry*, *Device Alarm, Device Event*
* Quản lý các liên kết của *Device*

### Assets

Tóm lược *Entity* liên quan đến các *Device* và *Asset khác.* ThingsBoard hỗ trợ các tính năng quản lý *Asset* sử dụng *Web UI* và *REST API :*

* Thêm và xóa các *Asset*: cho phép thêm hoặc xóa các *Asset* khỏi ThingsBoard
* Lấy ID của *Asset:* có thể copy ID của thiết bị sử dụng nút bấm trên màn hình.
* Kết nối *Asset* với *Customer*: để *Customer* có thể lấy dữ liệu trên *Asset* sử dụng *REST API* và *Web UI*.
* Truy cập *Asset Attribute*, *Asset Alarm, Asset Event*
* Quản lý các liên kết của *Asset*

### Entity Views (EVs)

Tương tự như với cơ sở dữ liệu SQL, ThingsBoard EVs giới hạn lượng thông tin *Customer* xem được, về *Device* hay *Asset Telemetry* hoặc *Attribute.* *Tenant Administrator* có thể tạo ra nhiều EVs cho mỗi *Device, Asset* đối với mỗi *Customer.* Hỗ trợ các trường hợp:

* Chia sẻ một vài dữ liệu *Device, Asset* với nhiều *Customer* đống thời.
* Cho phép *Customer* cụ thế có thể xem được dữ liệu thu được (dữ liệu cảm biến), những giấu một vài thông tin như mức pin, lỗi hệ thống, …
* Mô hình *Device-as-a-Service­* (DaaS), mà dữ liệu được thu vào thời điểm khác, thuộc về *Customer* khác.

EVs bao gồm các thông tin về:

* *Tenantld*: thể hiện liên kết đến chủ của giao diện
* *Customerld*: thể hiện liên kết đến *Customer* truy cập vào giao diện
* *Entityld:* thể hiện liên kết đến *Device, Asset* chỉ định
* *Name and Type:* Trường *Entity* của ThingsBoard được sử dụng để hiện thị và tìm kiếm
* *Start and end Time:* thể hiện khoảng thời gian được sử dụng để hạn chế truy cập đến *Device Telemetry* chỉ định, *Customer* sẽ không thể truy cập vào ngoài khoảng thời gian này.
* *Time series keys:* danh sách các khóa dữ liệu mà có thể truy cập bởi người xem
* *Attribute keys:* danh sách các tên *Attribute* mà có thể truy cập bởi người xem

*Time series data view:* Dữ liệu này được lưu trữ trong cơ sở dữ liệu của *Entity* được chỉ định. Các dữ liệu này không thể bị nhân bản ở các *Entity View* khác. Khi 1 người dùng truy cập vào *Dashboard* hoặc sử dụng *REST API* để gọi giúp *Entity View ID,* các hành động sau được thực hiện:

* Yêu cầu dấu thời gian *Start and End* được xác thực và điều chỉnh để vừa với thời gian *Start and End* của *Entity View ID*. Nếu *Dashboard* muốn lấy dữ liệu của 1 năm nhưng EV chỉ cho phép truy cập dữ liệu 6 tháng thì sẽ không thể lấy được.
* Yêu cầu *Time series data view* được xác thực và điều chỉnh dựa theo *Time series data view* được cấp phép trong *Entity View*. Nếu *Dashboard* cố gắng lấy khóa *telemetry* mà nó không được truy cập ở một giao diện chỉ định thì sẽ không vào được.

*Attributes View: Entity View* tự động copy các *Attributes* từ *Entity* được chỉ định mỗi lần lưu hoặc cập nhật *Entity View*. Để bảo đảm hoạt động, *Entity Attribute* không được nhân bản đến *Entity View* đối với mỗi sự thay đổi về *Attribute*. Việc tự động nhân bản có thế được cho phép khi cài đặt luật của nút “copy to view” trong dãy luật và kết nối “Post attributes” với tin nhắn “Attributes Updated” trong luật của nút.

### Alarms

ThingsBoard cung cấp khả năng tạo và quản lý các *Alarm* liên quan đến các *Entity*. Các khái niệm cơ bản:

1. Originator: điểm xảy ra báo động
2. Type: để xác định nguyên nhân của báo động
3. Severity: mức độ của báo động
4. Life Cycle: *Alarm* có thể hoạt động hoặc bị loại bỏ. Khi mặc định, thời gian start và end là như nhau. Nếu điều kiện kích hoạt *Alarm* lặp lại, thì platform sẽ cập nhật thời gian end và ThingsBoard có thể xóa *Alarm* khi điều kiện xóa xuất hiện. *User* cũng có thể tắt *Alarm* thủ công. ThingsBoard cũng theo dõi xem có người biết *Alarm* được kích hoạt không thông qua *Dashboard Widget* hoặc 1 tab về *Entity*, có 4 loại được thể hiện:

* ACTIVE\_UNACK: *Alarm* chưa xóa và chưa được ack
* ACTIVE\_ACK: *Alarm* chưa xóa nhưng được ack
* CLEARED\_UNACK: *Alarm* đã xóa và chưa được ack
* CLEARED\_ACK: *Alarm* đã xóa và được ack

1. Uniqueness: ThingsBoard nhận dạng *Alarm* thông qua sự kết hợp của các khái niệm trên. Tại 1 thời điểm, chỉ có 1 *Alarm* được kích hoạt có các đặc điểm nhận dạng riêng biệt.
2. Propagation: Để làm đơn giản hóa hàng chờ của dữ liệu và cải thiện thời gian chờ, ThingsBoard cho phép các *Alarm* được tạo ra có cần hiển thị trên *Entity* chủ không, đồng thời cũng có thể qui định mỗi quan hệ giữa với *Alarm* của *Entity* nguồn với các *Entity* chủ.

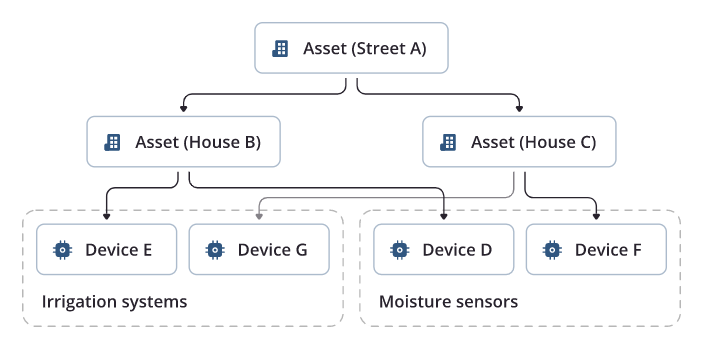
### Dashboard

Cho phép hiển thị dữ liệu và khả năng điều khiển *Device* khác qua giao diện

* *Entity Aliase*: xác định *Entities* nào được hiện thị trên Dashboard, mỗi Aliase có thể có 1 hoặc nhiều thiết bị.
* *Entity Filter:* xác định danh sách các *Enitity* được qui định bởi Aliase.
* *Widget:*
* Cung cấp các chức năng hiển thị, điều khiển, báo động nhất định, bao gồm 5 loại: time-series (dùng cho *Time-series* data là chính), latest values (sử dụng được với cả *Attribute* và *Time-series data*), control (cho phép gửi lệnh RCP), alarms (báo động), static (hiện thị dữ liệu tính như mặt bằng tòa nhà, thông tin công ty)
* *Dialog and Settings:* Data settings, Data Sources, Data keys, Widget Time Window, Alarm Filter, Basic settings(Title, Style, Legend, Display, Action).
* *Data key:* xác định *Attribute, Time-series* và trường *Entity* mà sử dụng trong *Widget*, kết hợp với một khóa. Danh sách các *Attribute key* là danh sách của các *Attribute client, server, shared* của thiết bị hoặc *Entities* khác. Danh sách các *Time-series key* phụ thuộc vào dữ liệu thiết bị gửi lên ThingsBoard hoặc được lưu trữ bởi *Rule Engine* hoặc *REST API*. Danh sách trường *Entity* phụ thuộc vào loại *Entity* và có thể mở rộng.
* *Widget Action:* Action Types, Action Source, Special Action Settings
* *Time window:*
* Thể hiện khoảng thời gian và hàm tổng hợp sẽ được sử dụng để lấy thời gian, có thể sử dụng với *Time-series Widget* (lấy dữ liệu trong khoảng thời gian qui định)*, Alarm Widget* (lấy các *Alarm* được tạo ra trong khoảng thời gian đó).
* Có thể hoạt động ở 2 chế độ: chế độ thời gian thực (liên tục cập nhật và hiển thị dữ liệu), chế độ lịch sử (chỉ lấy dữ liệu ở thời gian khởi tạo và không được cập nhật)

## Relations

Xác định các mối liên hệ giữa 2 *Entities* trong cùng 1 *Tenant,* bao gồm loại: *Contains, Manages, Supports.*



Hình 2.2. Mối liên hệ của Asset và Device

# DATA TYPES

## Attribute

ThingsBoard cung cấp khả năng tạo và quản lý các “Custom Attribute”. Các “Attribute” này được lưu trong cơ sở dữ liệu và có thể được sử dụng cho hình ảnh hóa và xử lý dữ liệu.

“Attribute” được sử dụng như khóa giá trị để kết nối. Tính linh hoạt và đơn giản của khóa giá trị này cho phép việc áp dụng đơn giản và liền mạch đối với phần lớn các thiết bị IoT trên thị trường. Khóa này luôn là 1 "string" và là tên “Attribute”, trong khi giá trị của “Attribute” có thể là "string", "boolean", "double", "integer” hoặc “JSON”

### Attribute names

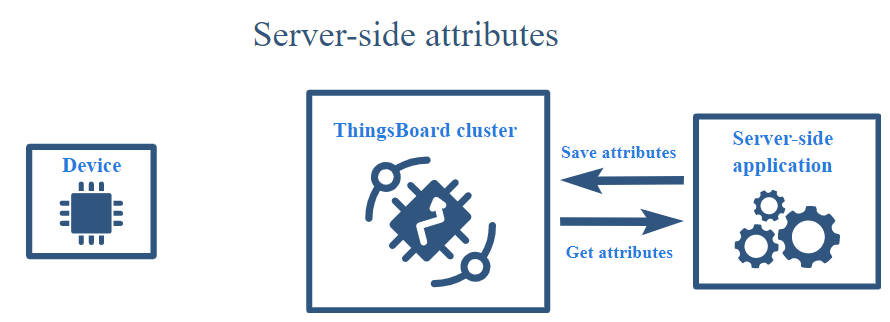
Có thể sử dụng bất kỳ loại tên nào, nhưng gợi ý sử dụng theo CamelCase để việc đặt tên cho các hàm JS để xử lý và hình ảnh hóa dữ liệu.

### Attribute types

Có 3 loại Attribute:

#### Loại Server

Loại Attribute này được hỗ trợ bởi mọi *Entity,* loại này có thể được điều chỉnh thông qua *Administration* *UI* hoặc *REST API. Device* không thể truy cập các *Attribute* này



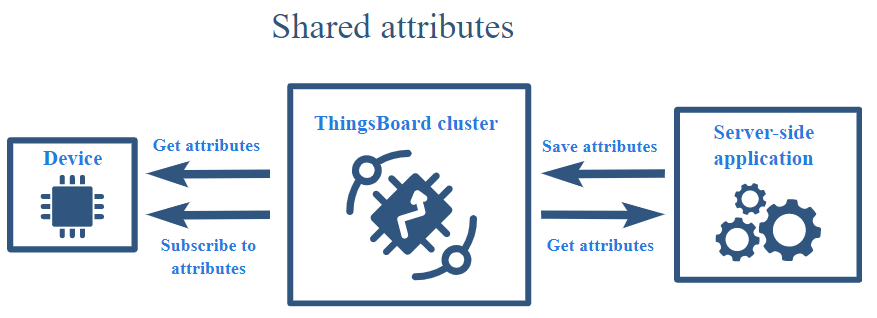
Hình 3.1. Server-side Attribute

Trong hệ thoogns giám sát trong tòa nhà:

* Các giá trị về kinh độ, vĩ độ, địa chỉ của các thiết bị là *Server Attribute*. Các *Attribute* này được đưa lên *Map Widget* để có thể hiện thỉ vị trí.
* *FloorPlanImage* có chứa URL cho hình ảnh và có thể được hiển thị trên *Map Widget* hình ảnh.
* *MaxTemperatureThreshold* và *TemperatureAlarmEnabled* có thể được sử dụng để cấu hình các *Alarm* cho một vài *Device, Asset*.

#### Loại Shared

Tương tự loại Server nhưng *Device* có thể lấy được hoặc đăng ký để lấy các *Attribute* này, Nếu sử dụng MQTT thì sẽ nhận dữ liệu thời gian thực, nếu sử dụng HPTT hoặc các giao thức đăng ký – nhận dữ liệu khác thì sẽ nhận được các giá trị theo chu kỳ nhất định.



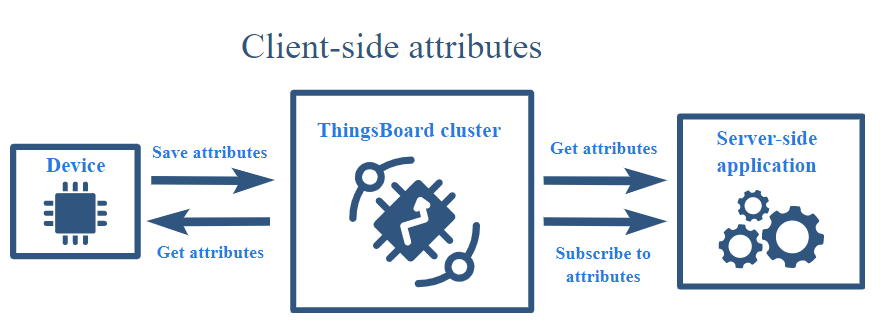
Hình 3.2. Shared Attribute

Trong hệ thống giám sát trong tòa nhà:

* *TargetFirmwareVersion Attribute* có thể lưu trữ các bản cập nhất của *Device*.
* *MaxTemperature* *Attribute* có thể tự động kích hoạt HVAC nếu phòng trở nên quá nóng.

#### Loại Client

Tương tự loại Shared nhưng cho phép gửi dữ liệu từ *Device* lên platform.



Hình 3.3. Client-side Attribute

Trong hệ thống giám sát trong tòa nhà:

* *CurrentFirmwareVersion Attribute* báo cáo bản cập nhật sẽ được cài đặt của *Device*
* *CurrentConfiguration Attribute* báo cáo bản cập nhật hiện tại của *Device*
* *CurrentState* hủy bỏ và khôi phục bản cập nhật hiện tại thông qua mạng, nếu thiết bị không có vùng chứa ổn định.

Lấy dữ liệu từ *Client Attribute* thông qua REST API.

### Attribute persistence

ThingsBoard lưu trữ các giá trị và thời gian điều chỉnh gần nhất trong cơ sở dữ liệu SQL, cho phép sử dụng lọc *Entity* trên *Dashboard.*

### Data Query API:

* REST API sử dụng để lấy lại dữ liệu *Attribute* cũ.
* API cho các ứng dụng: MQTT¸CoAP, HTTP.

### Data visualization:

* Hiện thị các dữ liệu của các *Attribute* lên *Dashboard,* có thể sử dụng *Widget*
* Hiện thị được nhiều *Attribute* cùng lúc, tuy nhiên giá trị trước của 1 Attribute sẽ bị loại bỏ khi có giá trị mới.

### Rule Engine:

* Xử lý các dòng dữ liệu và sự kiện
* Tạo *Alarm* dựa trên các giá trị của *Attribute*
* Điều chỉnh các dữ liệu trước khi chúng được đưa lên cơ sở dữ liệu
* Phản ứng với các thay đổi từ *Attribute Server*
* Lấy giá trị *Attribute* để phân tích dòng dữ liệu từ *Device*

## TIME-SERIES DATA (TELEMETRY DATA)

### Tính năng cơ bản

1. Thư thập dữ liệu từ các giao thức và ứng dụng
2. Lưu trữ dữ liệu vào cơ sở dữ liệu SQL (PostgreSQL) hoặc noSQL (Cassandra hoặc Timescale)
3. *Query* giá trị gần nhất hoặc những giá trị trong một khoảng thời gian nhất định.
4. *Subcribe* việc cập nhật dữ liệu sử dụng *WebSocket* để hiện thị hoặc phân tích thời gian thực.
5. Hiển thị dữ liệu sử dụng *Widget* và *Dashboard*.
6. Lọc và phân tích dữ liệu sử dụng *Rule Engine*
7. Tạo *Alarm* dựa trên các dữ liệu thu được
8. Chuyển tiếp dữ liệu đến các thiết bị ngoại vi sử dụng *External Rule Node*

### Data Upload API

* Sử dụng các giao thức đã được xây dựng: MQTT API, HTTP API, CoAP API.

### Data Visualization

* Hiện thị các dữ liệu của các Attribute lên Dashboard, có thể sử dụng Widget.
* Hiển thị các dữ liệu cũ và giá trị theo thời gian thực.

### Data Storage

* Cấu hình để lưu trữ dữ liệu trong cơ sở dữ liệu SQL hoặc noSQL hoặc Hybrid (kết hợp)
* SQL sử dụng được với lượng dữ liệu nhỏ, khoảng 5000 dữ liệu/s, sử dụng Cassandra có số lượng dữ liệu nhiều hơn.

### Data Retention

* Cassandra hỗ trợ giá trị TTL (có thể điều chỉnh với ‘TS\_KV\_TTL’ tron ‘Save Timeseries’) cho mỗi hàng thêm vào, với giá trị tối đa của TTL là 5 năm.
* VD có thể lưu trữ dữ liệu chưa qua xử lý trong 3 tháng, đồng thời lưu trữ dự liệu đã tổng hợp trong 3 năm.

### Data durability

* *Device* gửi *Time-series Data* lên ThingsBoard sẽ nhận được phản hồi khi dữ liệu này đã được lưu trữ trong *Queue* của *Rule Engine* dành cho *Device Profile* cụ thể.
* *Tenant* có thể cấu hình các kỹ thuật xử lý cho *Queue*, xử lý lại hoặc không quan tâm đến cái lỗi trong việc xử lý dữ liệu.

### Rule Engine

* Xử lý các dòng dữ liệu và sự kiện
* Tạo *Alarm* dựa trên các giá trị của *Attribute*
* Điều chỉnh các dữ liệu trước khi chúng được đưa lên cơ sở dữ liệu
* Tính toán sai lệch giữa giá trị trước và sau
* Lấy giá trị *Time-series* trước đó để phân tích dòng dữ liệu từ *Device*
* Lấy giá trị *Attribute* để phân tích dòng dữ liệu từ *Device*
* Tổng hợp dữ liệu đối với các *Asset* liên quan nhau

### Data Query REST API

Lấy khóa của *Time-series data* cho *Entity* được chỉ định

http(s)://host:port/api/plugins/telemetry/{entityType}/{entityId}/keys/timeseries

Lấy dữ liệu gần nhất của *Time-series data* cho *Entity* được chỉ định

http(s)://host:port/api/plugins/telemetry/{entityType}/{entityId}/values/timeseries?keys=key1,key2,key3

Lấy dữ liệu cũ của *Time-series data* cho *Entity* được chỉ định

http(s)://host:port/api/plugins/telemetry/{entityType}/{entityId}/values/timeseries?keys=key1,key2,key3&startTs=1479735870785&endTs=1479735871858&interval=60000&limit=100&agg=AVG

Các giá trị hỗ trợ như sau:

* keys – danh sách các khóa để lấy
* startTs – thời gian bắt đầu việc lấy dữ liệu
* endTs – thời gian kết thức việc lấy dữ liệu
* interval – thời gian tổng hợp các hàm
* agg – các hàm tổng hợp như SUM, MIN, MAX, AVG, COUNT, NONE.
* limit – số lượng dữ liệu được trả về hoặc khoảng thời gian xử lý giới hạn

ThingsBoard sẽ sử dụng *startTs, endTs, interval* để nhận diện các ngăn tổng hợp hay các *Query* và đưa *Query* không đồng bộ lên *Dashboard*.

### WebSocket API

WebSocket được sử dụng bởi ThingsBoard Web UI, nhân bản các chức năng REST API và cung cấp khả năng nhận dữ liệu của *Device*.

ws(s)://host:port/api/ws/plugins/telemetry?token=$JWT\_TOKEN

Các lệnh để nhận các dữ liệu cập nhật:

* cmdld – ID lệnh độc nhất
* entityType – loại *Entity* độc nhất : TENANT, CUSTOMER, USER, DASHBOARD, ASSET, DEVICE, ASSET, DEVICE, ALARM.
* entityld – nhận dạng *Entity* độc nhất
* keys – danh sách được chia bởi dấu phẩy của các khóa dữ liệu
* timeWindow – lấy thông tin của khoảng thời gian lấy dữ liệu, theo ms, dữ liệu lấy được sẽ nằm trong khoảng [now – timeWindow, now]
* startTs – thời gian bắt đầu lấy dữ liệu cũ, theo ms
* endTs – thời gian kết thúc lấy dữ liệu cũ, theo ms

VD: Trong <https://thingsboard.io/docs/user-guide/telemetry/> (thay các host:port thành 127.0.0.1:8080)

# PROFILES

## Tenant Profile

### Entity Limits

Cài đặt này cho phép *System Administrator* có thể điều chỉnh số lượng *Entity* mà mỗi *Tenant* có thể tạo được.

* ThingsBoard Community: cho phép giới hạn *Device, Asset, Customer, User, Dashboard, Rule Chain.*
* ThingsBoard Professional: cho phép giới hạn các *Entity* trên, đồng thời cùng bao gồm cả: *Integration, Converter, Scheduler* (bộ lập lịch)

### API Limit & Usage

Cấu hình số lượng tối đa các bản tin, gọi API trong mỗi khoảng thời gian đối với mỗi *Tenant*. ThingsBoard liên tục thu và xử lý *API Usage* đối với từng phần tử về *Transport, Rule Engine, JS Function, Telemetry Persistence, Email, SMS*. Hệ thống sẽ vô hiệu hóa phần tử nào vượt quá ngưỡng tối đa, khi *API Usage* bị quá ngưỡng nào đó (thường 80%) thì *Tenant Administrator* sẽ được thông báo.

* *Transport Message:* các bản tin *Device* gửi lên Server, có thể là *Telemetry, Attribute, RPC call, …*
* *Transport Data Point :* số của cặp *Key-Value* mà bản tin về *Telemetry, Attribute* chứa, giới hạn 1 *Data Point* là 512 ký tự
* *Rule Engine Execution*: các kết quả của *Rule Node* trong 1 *Tenant*, xử lý 1 bản tin có thể sử dụng đến nhiều *Execution* này.
* *JavaScript Execution*: các kết quả của hàm nào đó được xác định bởi *Tenant Administrator.*
* *Data Point storage days:* được tính với tất cả *Telemetry Data Point* trong cơ sở dữ liệu, thông số TTL sẽ được sử dụng để qui định số lượng và *Tenant Administrator* sẽ gửi giá trị này thông qua một yêu cầu Post.
* *Email sent:* số lượng email được gửi đi từ *Rule Engine* sử dụng SMTP.
* *SMS sent:* số lượng SMS được gửi đi từ *Rule Engine* sử dụng SMS.

### Rate Limits

Cấu hình số lượng tối đa các yêu cầu được xử lý đối với 1 *Device* hay cho tất cả *Device* đối với 1 *Tenant*, bao gồm cả giá trị và khoảng thời gian giới hạn.

### Processing in isolated ThingsBoard Core and Rule Engine Container

Cấu hình này mặc định là được tắt, chỉ có ích khi triện khai kiến trúc *Microservices*, việc điều chỉnh sai có thể dẫn đến việc xử lý sai các bản tin tới.

Với cấu hình này, mỗi *Tenant* sẽ có 1 *Core* và *Rule Engine* riêng để xử lý bản tin. Ở chế độ mặc định, bản tin được gửi lên cùng *Queue* và được lấy ra để xử lý trên cùng 1 *Rule Engine* nên cần ít tài nguyên hơn.

Còn chế độ còn lại, khi bản tin được đưa lên *Queue*, mỗi *Tenant* sẽ lấy bản tin liên quan và xử lý trên 1 *Rule Engine* riêng biệt, làm tốn nhiều tài nguyên hơn, nhưng cung cấp khả năng tính toán riêng biệt cho từng *Tenant*.

## Device Profile

*Tenant* *Administrator* có thể thiết lập cùng 1 cài đặt cho nhiều *Device*, mỗi *Device* chỉ có duy nhất 1 *Device Profile* trong 1 thời điểm.

### Rule Chain

“Root Rule Chain” là mặc định để xử lý tất cả các bản tin và sự kiện, tuy nhiên có càng nhiều loại thiết bị thì càng phức tạp nên “Root Rule Chain” thường chỉ dùng để gửi bản tin sang các *Rule Chain* khác – có thể được tạo bởi người dùng.

### Queue Name

“Main” là *Queue* mặc định. Tuy nhiên, khi có quá nhiều thiết bị sẽ cần nhiều *Queue* để lưu trữ nên cần phải tách biệt các loại *Queue* khác nhau đối với từng chức năng. Cài đặt này có thể được thực hiện ở *Device Profile* trong thingsboard.yml

### Transport Configuration

ThingsBoard hỗ trợ 3 loại Transport: *Default,* *MQTT* và *CoAP*

* *Default Transport:* Sử dụng các API của CoAP, HTTP, MQTT để kết nối với *Device*, không có cấu hình nào cho loại Transport này.
* *MQTT Transport:* Cho phép sử dụng các cấu hình MQTT nâng cao bao gồm *Device Topic Filter* (cho phép kết nối với bất kỳ *Device* nào gửi bằng JSON hoặc Protobuf), *Device Payload* (cho phép sử dụng *Protocal Buffer* để gửi dữ liệu một cách tuần tự nhưng giảm bớt được lượng dự liệu cần gửi đi, thay vì JSON như mặc định)
* *CoAp Transport:* Cho phép sử dụng các cấu hình CoAP nâng cao bao gồm *Device Type* (gồm loại mặc định sử dụng API, đồng thời cho phép truyền đi sử dụng Protobuf và loại *Efento NB-IoT* hỗ trợ đối với các cảm biến về nhiệt đô, độ ẩm, áp suất không khí, …)

### Alarm Rules

Sử dụng *Rule Engine* đối với từng *Rule Chain* của *Device Profile* để cài đặt cho *Alarm.*

* *Alarm Type:* loại báo động
* *Create Condition*: bao gồm các thông số về *Severity, Key Filter, Condition Type, Schedule, Detail.*
* *Clear Condition:* qui định điều kiện để hủy báo động
* *Advanced settings:* qui định tính tuần tự của báo động liên quan đến các *Entity.*

# RULE ENGINE

## Các khại niệm quan trọng

### Rule Engine Message

Là cấu trúc đữ liệu nối tiếp và không đổi, thể hiện rất nhiều loại bản tin trong hệ thống. VD như:

* Các cập nhật về *Telemtry, Attribute* và *RPC call* từ *Device*
* Các sự kiện xoay quanh sự tồn tại của các *Entity*: tạo, cập nhật, xóa, gắn vào và gỡ ra *Customer*.
* Các sự kiện về trạng thái của sự kiện: kết nối, ngừng kết nối, hoạt động, không hoạt động.
* Các sự kiện khác của hệ thống

Bao gồm các thông tin sau

* ID bản tin: dựa theo thời gian, các mã nhận dạng đặc thù
* Nguồn của bản tin: *Device, Asset,* mã nhận dạng *Entity*.
* Loại bản tin
* Dữ liệu bản tin mang : file JSON với các dữ liệu
* Meta-data : Danh sách của các giá trị cặp của các dữ liệu được thêm vào bản tin.

### Rule Node

Là đơn vị logic chính của *Rule Engine*, xử lý từng bản tin đến một và tạo ra 1 hoặc nhiều bản tin, có thể lọc, bổ sung, biến đổi các bản tin đến, thực hiện các hành vi và giao tiếp với hệ thống ngoài.

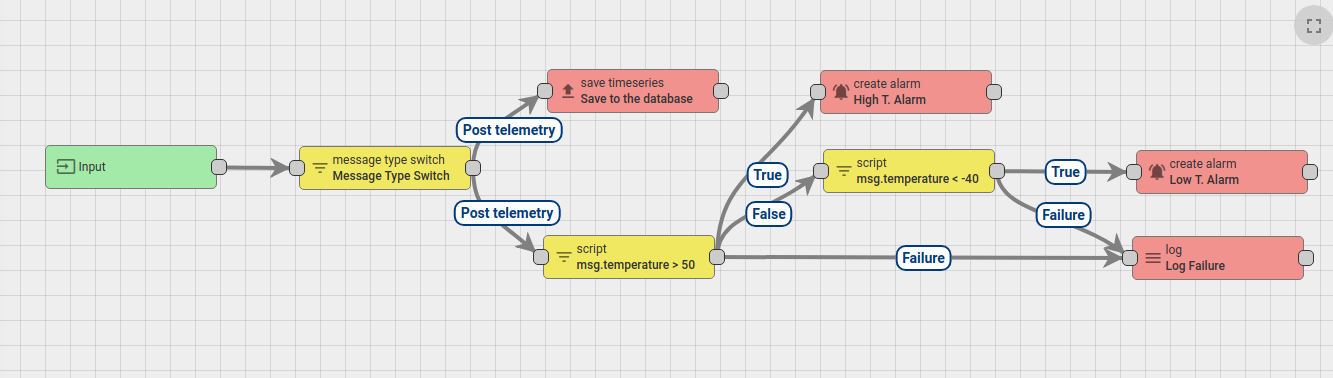
### Rule Node Relation

Các *Rule Node* có thể liên quan đến nhau, mỗi mối liên quan (*Relation)* này có 1 nhãn được sử dụng để thể hiện về mặt logic. Khi các *Rule Node* tạo ra các bản tin gửi đi, chúng luôn được xác định loại *Relation* được sử dụng để chuyển bản tin đến Node tiếp theo.

*Rule Node* *Relation* cơ bản thường là “Success”hoặc “Failure”. *Rule Node* thể hiện hoạt động logic có thể sử dụng “True” hoặc “False” Một vài *Rule Node* có thể có các *Relations* loại khác như "Post Telemetry”, “Attributes Updated”, “Entity Created”.

### Rule Chain

*Rule Chain* là nhóm *Rule Node* và các *Relation* của chúng. VD :



Hình 5.1. Ví dụ Rule Chain

* Lưu dữ liệu vào cơ sở dữ liệu
* So sánh nhiệt độ với 50 độ, nếu lớn hơn sẽ bật *Alarm*
* So sánh nhiệt độ với -40 độ, nếu nhỏ hơn sẽ bật *Alarm*
* Nếu bị lỗi thì sẽ hiện thị ra hàm log

### Message Processing Result

Có 3 kết quả có thể xảy ra trong việc xử lý bản tin : "Success", "Failure", "Timeout". "Success" khi *Rule Node* cuối cùng được hoàn thành, "Failure" khi không có *Rule Node* nào xử lý được trạng thái xảy ra, "Timeout" khi thời gian xử lý vượt quá giới hạn thời gian đặt trước.

### Rule Engine Queue

*Rule Engine* truy cập vào *Queue* để nhận bản tin mới, luôn có 1 topic chính để tạo đường vào cho các bản tin đến, có thể tạo các topic mới sử dụng "Checkpoint" Node – chúng sẽ tự ack các bản tin trong topic hiện tại. *Queue* bao gồm các thông tin :

* name – biến tĩnh và dùng để đăng nhập
* topic – để tạo và nhận bản tin
* poll-interval – thời gian theo ms chờ bản tin nếu không có bản tin mới nào được chuyển đến
* partitions – số lượng ngăn của *Queue,* dùng để tăng số lượng bản tin có thể xử lý song song
* pack-poccessing-timeout – khoảng thời gian theo ms để xử lý bản tin
* submit-strategy – định nghĩa logic và thứ tự đưa bản tin vào *Rule Engine*
* processing-strategy – định nghĩa logic ack của bản tin

### Queue submit strategy

*Rule Engine* liên tục đưa bản tin với topic nhất định và khi danh sách bản tin được trả về, “TbMsgPackProcessingContext” và sẽ được đưa vào *Rule Chain*:

* BURST – các bản tin được đưa vào *Rule Chain* đúng thứ tự chúng được gửi đến
* BATCH – các bản tin được nhóm lại sử dụng 1 giá trị “queue.rule-engine.queues[queue index].batch-size”, nhóm mới không được đưa vào khi nhóm trước chưa được ack
* SEQUENTIAL\_BY\_ORIGINATOR – các bản tin được thực hiện tuần tự, khi bản tin từ 1 *Entity* nguồn cụ thể chưa được ack thì bản tin tiếp theo của *Entity* nguồn đó sẽ không được đưa vào.
* SEQUENTIAL\_BY\_TENANT – các bản tin được đưa vào 1 cách tuần tự trong cùng *Tenant*.
* SEQUENTIAL – các bản tin được đưa vào 1 cách tuần tự, nếu bản tin trước chưa được ack thì bản tin sau sẽ không được đưa vào

VD:Trong <https://thingsboard.io/docs/user-guide/rule-engine-2-5/tutorials/queues-for-synchronization/>

### Queue processing strategy

Các *Strategy* giúp kiểm soát việc xử lý lại các bản tin lỗi hoặc quá thời gian xử lý, có 5 loại *Strategy* có thể sử dụng:

* SKIP\_ALL\_FAILURES – bỏ qua tất cả failure và timeout
* RETRY\_ALL – xử lý lại toàn bộ bản tin được gửi đến
* RETRY\_FAILED – xử lý lại các bản tin failure
* RETRY\_TIMED\_OUT – xử lý lại các bản tin timeout
* RETRY\_FAILED\_AND\_TIMED\_OUT – xử lý các bản tin timeout và failure

Mỗi *Strategy* sẽ bao gồm các thông số có thể điều chỉnh: *retries* (qui định số lần thử lại, 0 là không giới hạn), *failure-percentage* (bỏ qua việc thử lại nếu lỗi nhỏ hơn X%), *pause-between-retries* (thời gian đợi tính theo giây trước khi thử lại)

VD:Trong <https://thingsboard.io/docs/user-guide/rule-engine-2-5/tutorials/queues-for-message-reprocessing/>

### Default queue

Có 3 loại *Queue* cơ bản: Main, HighPriority và SequentialByOriginator. Chúng được chia theo *submit strategy, processing strategy*.

* Main mặc định sẽ bỏ qua toàn bộ các bản tin lỗi, tuy nhiên có thể điều chỉnh lại được phù hợp với yêu cầu
* HighPriority sử dụng để chuyển các *Alarm* và các bước xử lý quan trọng khác, có thể sẽ xử lý lại liên tục khi có lỗi để bản tin toàn vẹn.
* SequentialByOriginator giúp bản tin được xử lý theo đúng thứ tự.

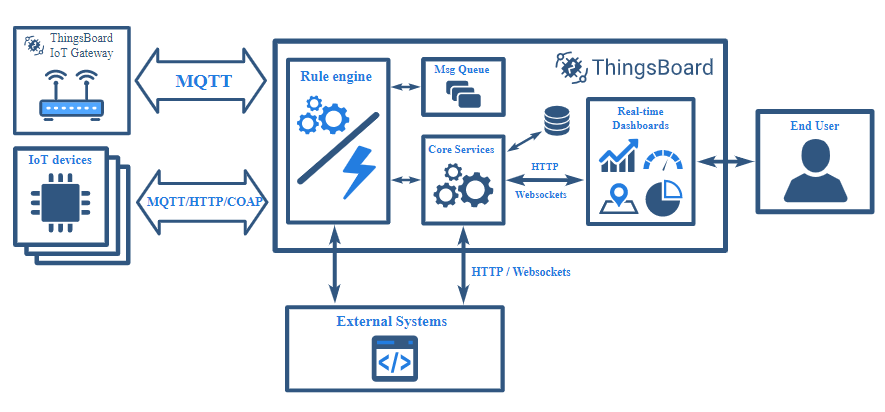
## Predefined Message Types

( <https://thingsboard.io/docs/user-guide/rule-engine-2-0/overview/> )

## Rule Node Types

* Filter Nodes sử dụng trong lọc và truyển tiếp bản tin
* Enrichment Nodes sử dụng để cập nhật meta-data đối với các bản tin đến
* Transformation Nodes sử dụng để đổi các trường của bản tin
* Action Nodes thực hiện các hoạt động dựa trên bản tin đến
* External Nodes kết nối với các hệ thống bên ngoài

## Architecture



Hình 5.2. Kiến trúc Rule Engine

## Capabilities

ThingsBoard hỗ trợ các tính năng nhất định để người dùng có thể điều chỉnh Rule Engine đơn giản hơn:

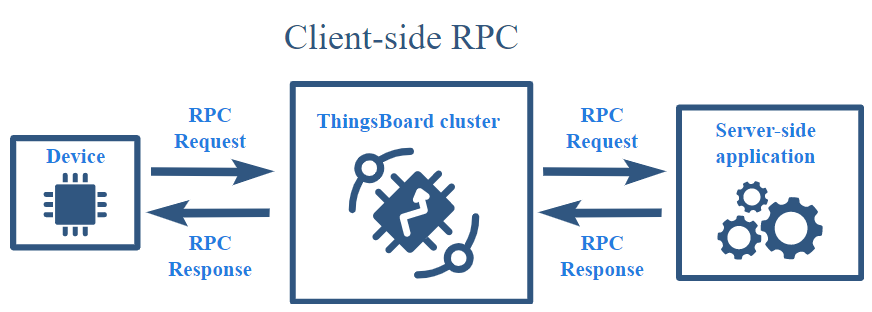
* *Configuration:* điều chỉnh trên giao diện sử dụng các hàm Test JavaScript
* *Statistics:* được cài đặt sẵn với từng *Tenant*
* *Debugging*: kiểm tra các bản tin đến và đi đối với từng *Rule Node*
* *Import/Export:* có thể chuyển *Rule Chain* thành dạng JSON
* *Custom REST API calls*

# OTHER FEATURES

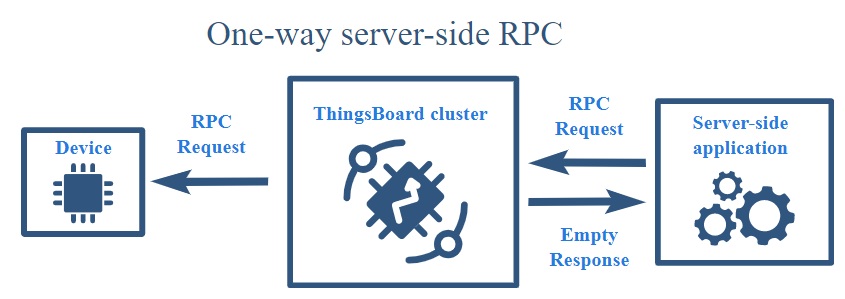
## RPC (Remote Procedure Calls) capabilities

Cho phép *Server Application* gửi lệnh đến *Device* và nhận kết quả trả lại kết quả tương ứng với lệnh đó từ *Device*, hoặc trước khi trả kết quả, *Device* sẽ thực hiện vài thuật toán. Một vài chủ đề cần được lưu ý: loại RPC, các ứng dụng cơ bản, RPC của *Server* và *Client*, RPC *Widget*.

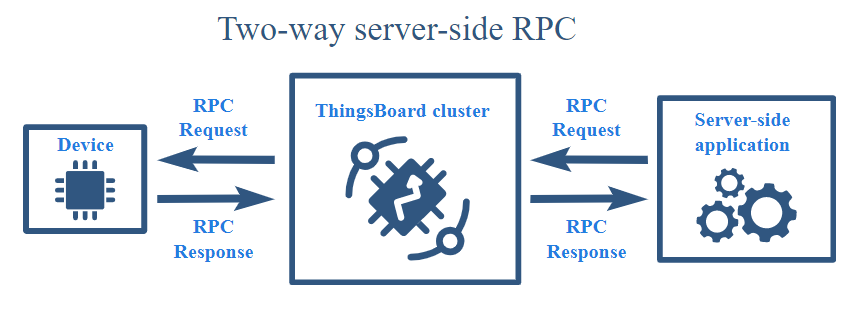
* Loại RPC:



Hình 6.1. Client-side RPC



Hình 6.2. One-way server-side RPC



Hình 6.3. Two-way server-side RPC

* Device RPC API: ThingsBoard cung cấp các API giúp gửi và nhận các lệnh RPC từ ứng dụng chạy trên điện thoại, hỗ trợ với từng loại giao thức MQTT, CoAP, HTTP.
* Server-side RPC API: Để có thể gửi được gửi 1 yêu câu RPC cần gửi HTTP POST lên URL sau:

http(s)://host:port/api/plugins/rpc/{callType}/{deviceId}

Với: callType = oneway (twoway); deviceID = Device ID được gửi đến.

curl -v -X POST -d @set-gpio-request.json http://localhost:8080/api/plugins/rpc/twoway/$DEVICE\_ID \

--header "Content-Type:application/json" \

--header "X-Authorization: $JWT\_TOKEN"

File JSON:

{

"method": "setGpio",

"params": {

"pin": "23",

"value": 1

}

}

Với: method = tên hàm, JSON string; params = thông số của hàm, JSON object.

Để có thể thực hiện yêu cầu này, cần phải thay $JWT\_TOKEN của TENANT\_ADMIN hoặc CUSTOMER\_USER sở hữu mã nhận dạng thiết bị $DEVICE\_ID.

* RPC Rule Nodes: RPC reply, RPC request.

## Device Connectivity Status

Các sự kiện được hỗ trợ khi giám giát hoạt động của thiết bị

* Connect event: được kích hoạt khi có thiết bị kết nối lên ThingsBoard (hoạt động với cả MQTT và HTTP)
* Disconnect event: được kích hoạt khi có thiết bị ngắt kết nối
* Activity event: kích hoạt khi có dữ liệu được gửi lên hoặc lệnh RPC
* Inactivity event: kích hoạt khi thiết bị không hoạt động trong một khoảng thời gian.

Các trạng thái để duy trì các *Server-side Attribute*

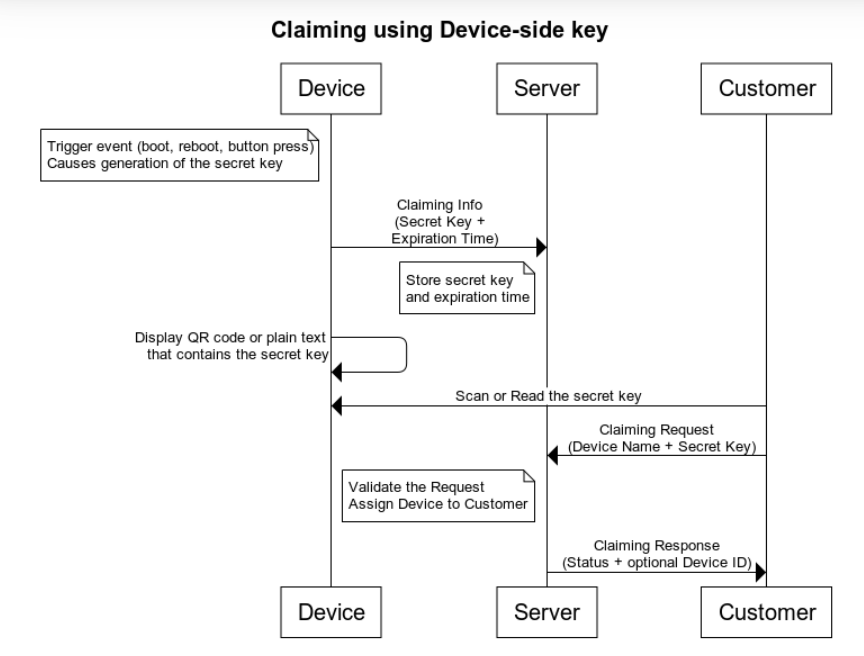
* active: trạng thái thiết bị
* lastConnectTime: thời gian cuối cùng thiết bị được kết nối (theo ms)
* lastDisconnectTime: thời gian cuối cùng thiết bị ngắt kết nối (theo ms)
* lastActivityTime: thời gian cuối cùng gửi dữ liệu của thiết bị (theo ms)
* inactivityAlarmTime: thời gian cuối cùng sự kiện inactivity được kích hoạt (theo ms)

## Device Claiming

Giúp cho *Customer* có thể nhận thiết bị của họ sở hữu thông quét QR hoặc các kỹ thuật tương tự khác khi họ nhận được thiết bị.

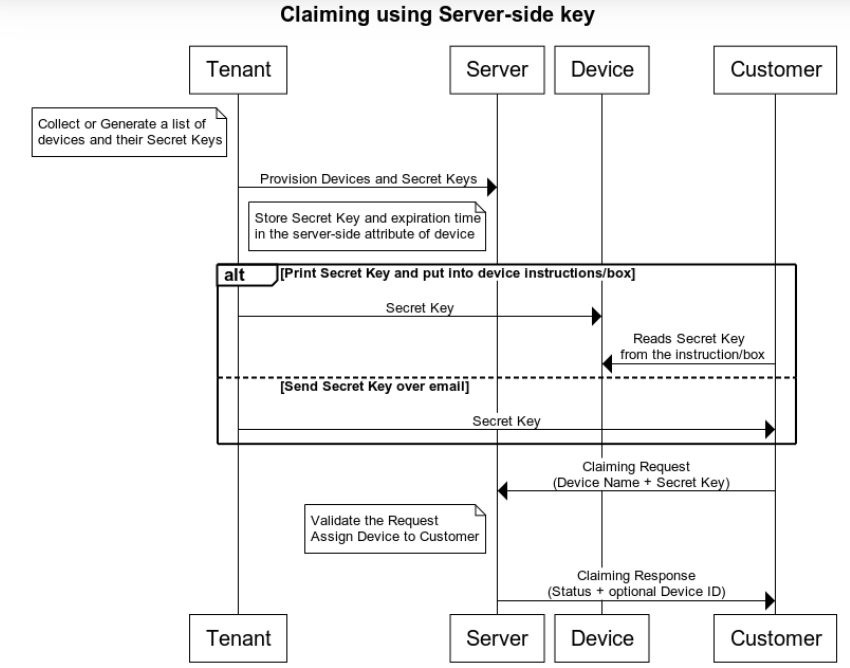
Có 2 cách cung cấp *Secret Key*:

* *Device-side key*: bao gồm *expirationTime* thể hiện thời gian hết hạn, thiết bị phải gửi yêu cầu lên ThingsBoard rồi *Customer* mới có thể nhận thiết bị của họ thông qua *Widget* nhất định.



Hình 6.4. Device-side key claiming

* *Server-side key:* bao gồm *claimingData* và không cần *Customer* phải gửi yêu cầu lên ThingsBoard.



Hình 6.5. Server-side key claiming

Các khái niệm cơ bản:

* Device Claiming Permissions in PE: cần sự cho phép ở *Device* và *Claim Device*
* Device Claiming Widget: dùng để tìm *Device*, đặt tên *Device* và *Secret Key* và một vài cài đặt khác.
* Device Claiming API Request: để yêu cầu việc nhận thiết bị
* Device Reclaiming API Request: gửi lại yêu cầu việc nhận thiết bị.

## Provisioning Devices

Cho phép thiết bị tự kết nối lên ThingsBoard bằng cách sử dụng các chứng chỉ riêng biệt hoặc yêu cầu cung cấp chứng chỉ riêng biệt cho thiết bị. Có 2 cách:

* Cho phép tạo thiết bị mới.
* Kiểm tra các thiết bị đã *Provision* trước đó.

### Device Profile Configuration

Cho phép tính năng Provisioning của thiết bị, thu thập Provision Key, Secret Key

* Tạo *Device Profile* mới hoặc dùng cũ
* Tạo tên cho *Device Profile*
* Chọn cách Provisioning và copy Key.

### Provision Device API

* MQTT Device API: Sử dụng chứng chỉ tạo từ ThingsBoard server hoặc Access Token hoặc Basic MQTT Credentials hoặc X.509 Certificate.
* HTTP Device API: Sử dụng chứng chỉ tạo bởi ThingsBoard server hoặc Access Token
* CoAP Device API: Sử dụng chứng chỉ tạo bởi ThingsBoard server hoặc Access Token.

## Administration UI

Tạo mới, xóa bỏ hoặc điều chỉnh các *Entity*

* *Tenant*
* *Customer*
* *User*
* *Asset*
* *Device*
* *Rule Chain*
* *Dashboard*

Cài đặt thêm các tính năng như SMS và Mail