VIRTUAL CONTROL ROOM (VCR)

Collecting information. Controlling systems.

A modular platform

for traffic, event, logistics, and agricultural management

based on Zabbix

Short description

Our **Virtual Control Room (VCR)** is an **open, modular platform** designed for monitoring and **controlling** distributed systems.

The platform uses Zabbix as its monitoring and portal base. Zabbix's excellent capabilities for displaying IT equipment information have been enhanced by several business-level components from Hilbig IT in the backend and frontend.

We have also added a GraphQL API layer to facilitate translation between Zabbix and the domain of controlling IoT devices. This layer uses MQTT and an MQTT broker written in Node.js to aggregate sensor data and submit it to Zabbix. We wrote a device controller that listens for control automatisms scheduled for execution on MQTT. The controller monitors conditions such as sensor values to submit the desired commands to actuator devices, such as LED displays.

The device configuration makes heavy use of Zabbix templates and tags. The GraphQL API maps item data and delivers device sensor information, mapping tags into configuration data within our domain-specific language. This language is used by our custom front-end web components, which are implemented in TypeScript and Angular. These web components can be used as Zabbix widgets. On the other hand, we use DOM patching to add rich client applications to the Zabbix front end at runtime. For example, this could be used as an alternative, interactive replacement for icons within Zabbix network map widgets.

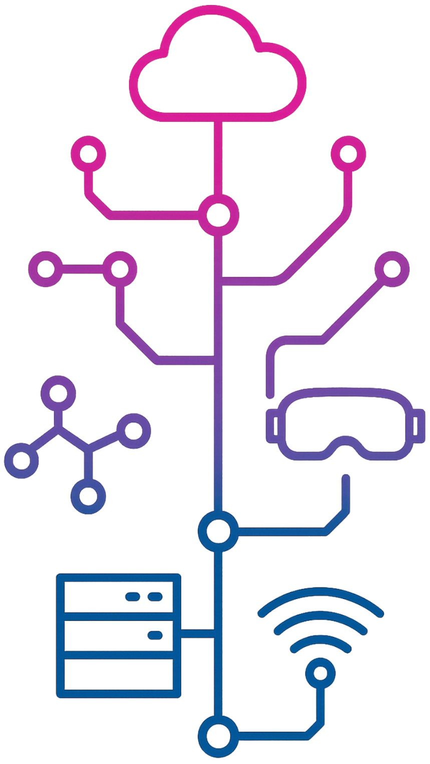
As a result, the VCR provides a **precise situational picture** that facilitates operational decisions for businesspeople, whether they are involved in traffic management, large-scale events, logistics or agriculture.

The VCR thus creates a connection between **data, processes, and decisions**.

Thanks to its open architecture, it can be easily adapted to new applications and industries.

We are technology enthusiasts and entrepreneurs.

We create solutions from bits and bytes that make a difference.



Hilbig
IT WORKS

www.hilbigit.com

ZABBIX 20 YEARS

What does the VCR do?

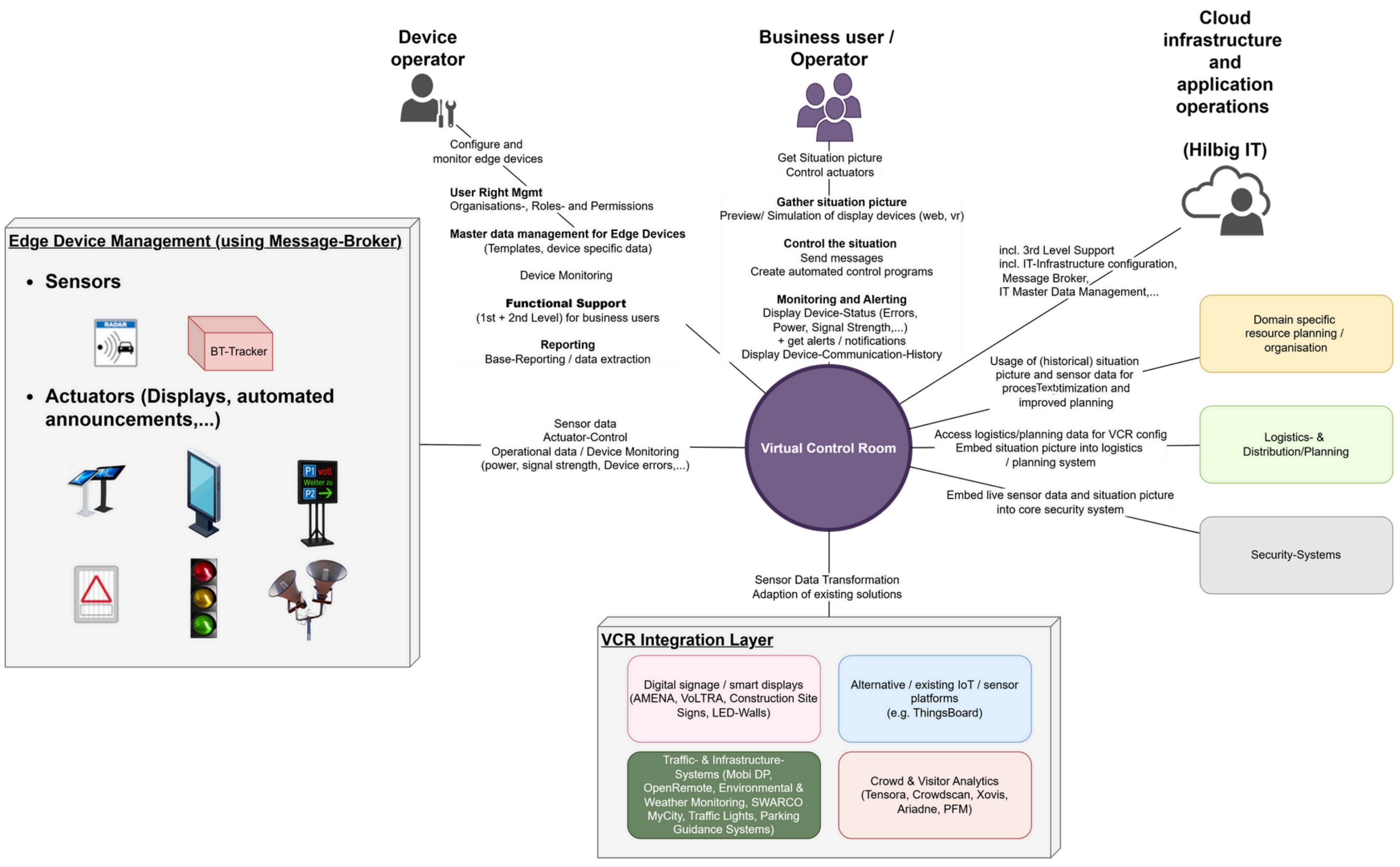

Device operator
Configure and monitor edge devices
User Right Mgmt
Organisations-, Roles- and Permissions
Master data management for Edge Devices
(Templates, device specific data)
Device Monitoring
Functional Support
(1st + 2nd Level) for business users
Reporting
Base-Reporting / data extraction
Business user / Operator
Get Situation picture
Control actuators
Gather situation picture
Preview/ Simulation of display devices (web, vr)
Control the situation
Send messages
Create automated control programs
Monitoring and Alerting
Display Device-Status (Errors, Power, Signal Strength,...)
+ get alerts / notifications
Display Device-Communication-History
Cloud infrastructure and application operations
(Hilbig IT)
incl. 3rd Level Support
incl. IT-Infrastructure configuration,
Message Broker,
IT Master Data Management,...
Edge Device Management (using Message-Broker)
Sensors
RADAR
BT-Tracker
Actuators (Displays, automated announcements,...)
P1 voll
P2
Sensor data
Actuator-Control
Operational data / Device Monitoring
(power, signal strength, Device errors,...)
Virtual Control Room
Usage of (historical) situation picture and sensor data for procesTextptimization and improved planning
Domain specific resource planning / organisation
Access logistics/planning data for VCR config
Embed situation picture into logistics / planning system
Logistics- & Distribution/Planning
Embed live sensor data and situation picture into core security system
Security-Systems
Sensor Data Transformation
Adaption of existing solutions
VCR Integration Layer
Digital signage / smart displays (AMENA, VoLTRA, Construction Site Signs, LED-Walls)
Alternative / existing IoT / sensor platforms (e.g. ThingsBoard)
Traffic- & Infrastructure-Systems (Mobi DP, OpenRemote, Environmental & Weather Monitoring, SWARCO MyCity, Traffic Lights, Parking Guidance Systems)
Crowd & Visitor Analytics (Tensora, Crowdscan, Xovis, Ariadne, PFM)

Figure: VCR System Context

Our **Virtual Control Room** encompasses the entire control loop of **data acquisition, analysis, and control**.

1. **Capture and Understand**
   The VCR collects telemetry and sensor data—from radar, Bluetooth, or LiDAR sensors, GPS trackers, and battery monitors. This information is reliably transmitted via the **standardized IoT protocol MQTT** (e.g. with **RabbitMQ** as broker).

   The dashboard—built on the open-source system **Zabbix**—enables an **individually configurable display** in map, table, or diagram form.

2. **Manage and Secure**
   The integrated **Edge Device Management** is used to **manage** field devices. This enables monitoring and alerting based upon both sensor data and operational data like states, firmware versions, connectivity, or battery power. Role and rights concepts, logging, and alerting ensure security and traceability.

   Field devices can be sensors or actuators like traffic light systems, mobile LED panels, or others. The devices can either implement our simple protocol, or they are integrated using our **VCR Integration Layer,** which is based upon the open-source integration solution Apache Camel. It translates existing protocols and adapts our solution to use information from other IoT platforms.

3. **Control and Automate**
   Controls can be performed manually via the dashboards or **automatically through defined rules and workflows**.
   For example, warning signs can be activated, traffic flows can be adjusted, or display panels can be controlled in real time.
   Feedback from the devices is automatically compared with target values, allowing immediate detection of disruptions or failures.

Unique Features: Flexibility, Openness, and Possibility for Integration with Your Existing Systems

The VCR features open **APIs/interfaces** and focuses on the **integration of existing systems** through the **VCR Integration Layer**. Operation can occur in the **cloud or on-premise**. Even hybrid models are possible. It can be supplemented with **3D or VR interactions** if needed.

It is continuously developed in **research and pilot projects** to explore new applications and technologies. The core source code is **open source**. The supplementary components are made available transparently and in full. **Vendor lock-in is avoided** as self-operation is possible if desired.

How does it work? And how is Zabbix used?

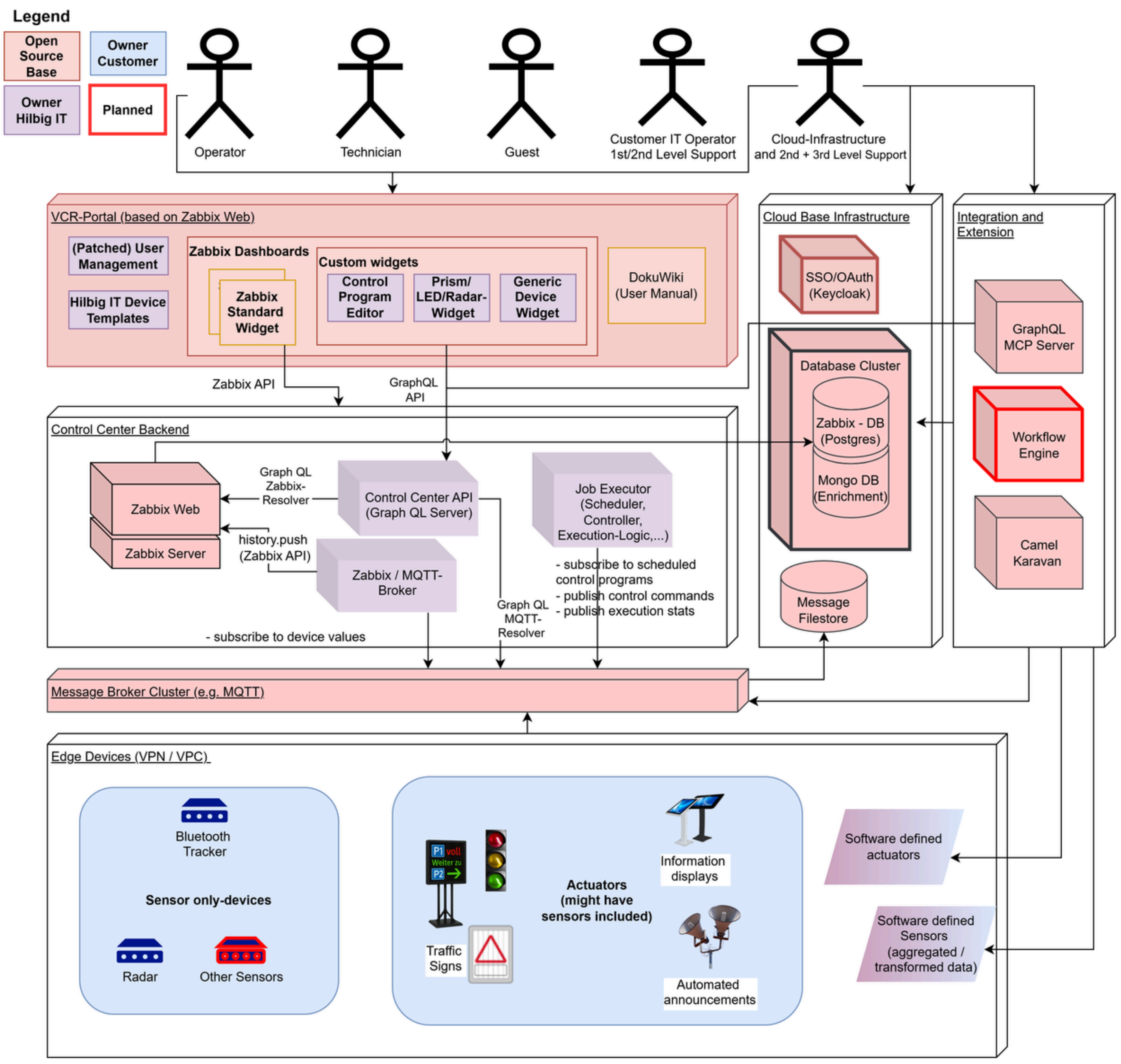


Figure: VCR Component Model

We used Zabbix and other open-source components as a baseline and added the following components and modifications:

1. **Hilbig IT Device Templates**
   Zabbix templates are used to represent a specific type of device. Within the templates, both the device metadata and the mapping to MQTT values are represented in a way that can be mapped to an object model by the GraphQL API (see “Appendix 1: IoT Device Templates-Sample” for details)

2. **(Patched) User Management**
   We use the Zabbix cookie to authenticate against our GraphQL API. The token is forwarded to the Zabbix API as soon as a GraphQL Zabbix-Resolver is concerned. We implemented a special “queryUserPermissions” operation that queries access to all template groups having a “Permission/” prefix. This convention can be used to define and query front-end-level permissions within our Angular applications (Control Program Editor and Device Widgets).

   We had to apply a minimally invasive patch to the Zabbix API to implement the “Customer IT Operator” role. This role requires the permission to assign users to groups and roles, but our customers must not become super users. We solved this by patching the Zabbix API source code—each time the login user’s role name has a prefix (separated by a token that can be set within the environment settings), the “query groups” and “query roles”—Zabbix API Operations will get an injected filter only delivering roles and groups starting with this exact prefix. E.g. if a user belongs to the role “Customer1 Admin” this user will only be able to see roles and groups starting with “Customer1“—even if the user is Super Admin. Therefore this user will only be able to assign users to “customer roles” and not to super admin roles which are restricted to be used by the “Cloud Infrastructure and Technical Operation”-staff.

3. **Control Program Editor**
   We built a small Zabbix widget, which is a wrapper around selectable and configurable HTML tags. This enables us to embed our own Angular web components—one of them is used to define control programs, which can automate device commands based upon sensor or other conditions. Those control programs will be stored as JSON text values within a specific Zabbix Trapper Item belonging to a “Controller”-host, which will be automatically created as soon as a user stores a control program (one controller host will be created per host group). If a control program is in the “ACTIVE” state, it will be scheduled for execution on a specific MQTT topic and interpreted by our Job Executor.

   The permissions for our Control Program Editor are derived from a call to the GraphQL operation “queryPermissions” and may therefore be defined by using the Zabbix default mechanism of assigning host group permissions to one of the user’s user groups.

   We had to apply a small patch to the Zabbix source code to enable the bootstrapping of our Angular web components, which could be applied without modifications from Zabbix Version 6.4 to 7.4 thanks to the clean and stable Zabbix code provided by the Zabbix Team—thanks, great work!

4. **Prism/LED/Radar-Widget and Generic Device Widget**
   We wanted to embed live previews of LED display devices showing operational data like voltage etc., and on top a specific picture depending on the received data. The previews should be embeddable into a network map, as we liked the network map functionality by 100 %. We wanted to avoid heavy patching of Zabbix Network Map Code and decided to implement our preview pictures as Angular web components using an SVG template instead of HTML. In order to display our components at positions specified by the user with Zabbix standard means (as “Images”) we wrote a little JavaScript hook that patches the DOM of the Zabbix website and manipulates all SVG-NetworkMap-children of type “Image”, replacing them by calls to our own Angular web components. To make this work, we only had to apply a small patch to the Zabbix PHP code, which enables our component to receive additional data like the hostid, which is normally not passed within the network map SVG.

5. **Zabbix/MQTT-Broker**
   Unfortunately, we found the MQTT plugin provided by Zabbix to be buggy, which is a well-known issue within the Zabbix community. Therefore, we reimplemented a simple version of it in Node.js.

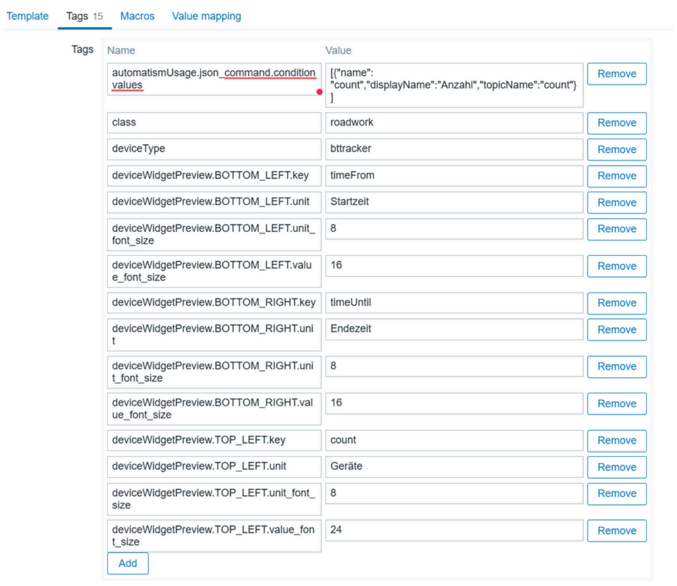
6. **Control Center API (GraphQL Server)**
   We wanted to establish a domain specific language on top of the Zabbix Data Model and implemented a GraphQL API. Next to VCR-specific operations this provides data extraction features, which can on the one hand be used for historization and reporting purpose. On the other hand we use it for migrating data including user groups and roles from one VCR instance to the other, which is required for supporting customer specific staging requirements and running multiple tenants on completely separated servers.
7. **GraphQL MCP Server and Workflow Engine for AI Agent integration**
   We use the Apollo MCP Server and our API in order to allow Agent based analysis and operation auf our Solution. Planned: Usage of a workflow engine and combination with MCP Based AI Agent in order to enable enhanced workflow control supperted

8. **Job Executor**
   Our job executor listens for control automatisms scheduled for execution on a predefined MQTT topic and then executes commands after subscribing to and checking the values of the appropriate sensor conditions.

9. **Software defined actuators and sensors**
   We started a cooperation with a German university of technology in order to build a new device for tracking locations and counting devices based upon Bluetooth technology. As we started with an existing open-source project delivering raw data to our MQTT, we needed a protocol conversion and introduced Apache Camel and Apache Karavan. Meanwhile, this is also used to get aggregated information from multiple devices or from e.g. counting devices occurring within a time interval.

Appendix 1: IoT Device Templates-Sample

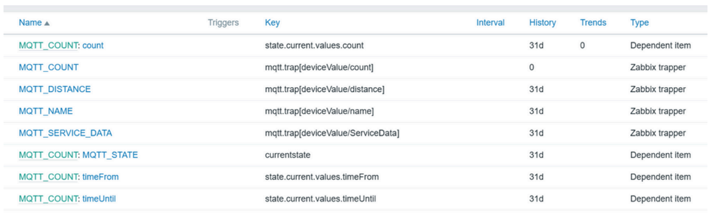
We make intensive use of Zabbix Templates in order to represent a specific type of device. The device metadata is stored as Tags, the data is represented by items. We use JSON-Path-like tag / item names in order to map this data to an object model by the GraphQL-API, as illustrated within the following example



Tags

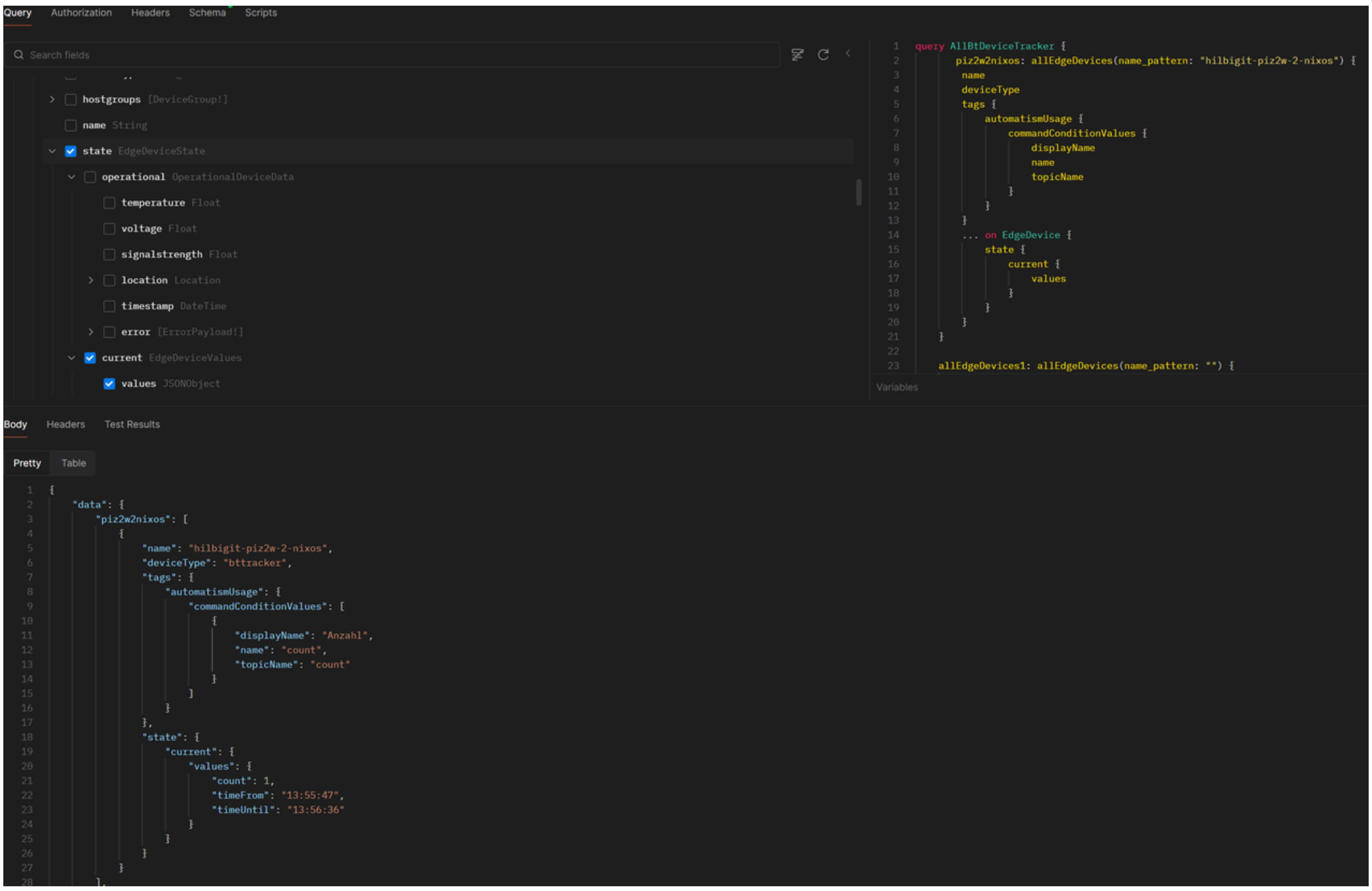
Name	Value	
automatismUsage.json_command.condition values	[{"name": "count","displayName":"Anzahl","topicName":"count"}]	Remove
class	roadwork	Remove
deviceType	bttracker	Remove
deviceWidgetPreview.BOTTOM_LEFT.key	timeFrom	Remove
deviceWidgetPreview.BOTTOM_LEFT.unit	Startzeit	Remove
deviceWidgetPreview.BOTTOM_LEFT.unit_font_size	8	Remove
deviceWidgetPreview.BOTTOM_LEFT.value_font_size	16	Remove
deviceWidgetPreview.BOTTOM_RIGHT.key	timeUntil	Remove
deviceWidgetPreview.BOTTOM_RIGHT.unit	Endezeit	Remove
deviceWidgetPreview.BOTTOM_RIGHT.unit_font_size	8	Remove
deviceWidgetPreview.BOTTOM_RIGHT.value_font_size	16	Remove
deviceWidgetPreview.TOP_LEFT.key	count	Remove
deviceWidgetPreview.TOP_LEFT.unit	Geräte	Remove
deviceWidgetPreview.TOP_LEFT.unit_font_size	8	Remove
deviceWidgetPreview.TOP_LEFT.value_font_size	24	Remove

1. Modelling device meta data as Zabbix Tags



Name ▲	Triggers	Key	Interval	History	Trends	Type
MQTT_COUNT: count		state.current.values.count		31d	0	Dependent item
MQTT_COUNT		mqtt.trap[deviceValue/count]		0		Zabbix trapper
MQTT_DISTANCE		mqtt.trap[deviceValue/distance]		31d		Zabbix trapper
MQTT_NAME		mqtt.trap[deviceValue/name]		31d		Zabbix trapper
MQTT_SERVICE_DATA		mqtt.trap[deviceValue/ServiceData]		31d		Zabbix trapper
MQTT_COUNT: MQTT_STATE		currentstate		31d		Dependent item
MQTT_COUNT: timeFrom		state.current.values.timeFrom		31d		Dependent item
MQTT_COUNT: timeUntil		state.current.values.timeUntil		31d		Dependent item

2. Modelling device values as Zabbix Items



```
> ☐ hostgroups [DeviceGroup!]
  ☐ name String
v ☑ state EdgeDeviceState
  v ☐ operational OperationalDeviceData
      ☐ temperature Float
      ☐ voltage Float
      ☐ signalstrength Float
    > ☐ location Location
      ☐ timestamp DateTime
    > ☐ error [ErrorPayload!]
  v ☑ current EdgeDeviceValues
      ☑ values JSONObject
```

```
query AllBtDeviceTracker {
    piz2w2nixos: allEdgeDevices(name_pattern: "hilbigit-piz2w-2-nixos") {
     name
     deviceType
     tags {
         automatismUsage {
             commandConditionValues {
                 displayName
                 name
                 topicName
             }
         }
     }
     ... on EdgeDevice {
         state {
             current {
                 values
             }
         }
     }
    }

    allEdgeDevices1: allEdgeDevices(name_pattern: "") {
```

```
{
    "data": {
        "piz2w2nixos": [
            {
                "name": "hilbigit-piz2w-2-nixos",
                "deviceType": "bttracker",
                "tags": {
                    "automatismUsage": {
                        "commandConditionValues": [
                            {
                                "displayName": "Anzahl",
                                "name": "count",
                                "topicName": "count"
                            }
                        ]
                    }
                },
                "state": {
                    "current": {
                        "values": {
                            "count": 1,
                            "timeFrom": "13:55:47",
                            "timeUntil": "13:56:36"
                        }
                    }
                }
            }
        ],
```

3. Receiving device meta and values mapped to GraphQL - Objects