Internet Of Things

Smart Production Control And Anomaly Detection

Department of Information Engineering
M.Sc in Artificial Intelligence and Data Engineering

Fabio Cognata, Luca Di Giacomo

Academic Year 2024/2025

Code Available as GitHub Repository

INDEX

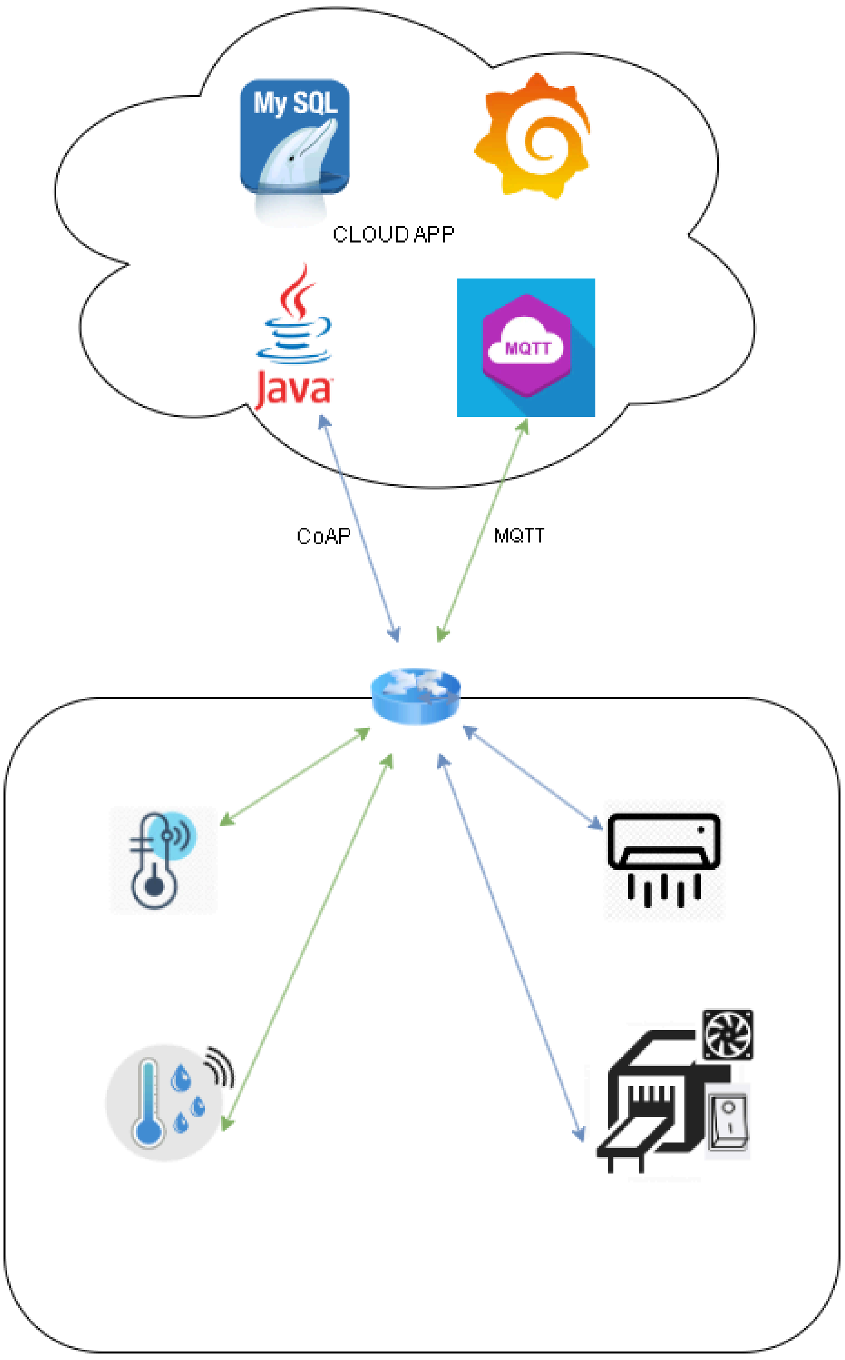
1. Application

The idea behind the application is to use sensors and actuators to monitor and automatically handle critique situations during Industrial processes involving production machines (using paper production machines as reference for which humidity temperature and produced quantities are used as metrics for qualifying the activity level and anomaly assessment of the machine). It will also provide automatic handling of the external environment where machines are located using a Conditioner to adjust the external temperature.

The system interacts with a cloud application based on Java and MySQL to notify and store data received from sensors as well as anomalies occurring on the single machine.

It also provides the possibility to control actuation from a User Interface using specific commands for changing the actuation behavior and a web interface exploiting Grafana to visualize machines' history.

1.1. System Architecture



1.2. Environment Description

In accordance with the project guidelines, the COAP protocol is employed by the actuator nodes, while the MQTT protocol is mandated for the sensor nodes. This protocol was adhered to in the project, enabling sensors to communicate via MQTT and actuators via COAP.
The technologies utilized in the project are:

- Zorin OS 16.3 based on Ubuntu 22.04 LTS
- Contiki-NG v4.7 on Docker
- C language for sensors
- Java language for cloud application
- MySQL database
- Californium (Coap Java library)
- Paho (MQTT Java library)
- HikariCP for a "zero-overhead" production-ready JDBC connection pool
- Grafana for data visualization

During the testing phase, the Cooja simulator was used. Once the program's functionality was validated within the simulator, the deployment phase commenced on the sensor devices, specifically targeting the nRF52840 dongle boards.

The virtual machine simulates the cloud environment by exposing the Java server, MySQL database and Grafana web interface while sensors and actuators are deployed in Contiki-NG container communicating with the Virtual Machine's environment by exposing the rpl-border-router port 60001 using Contiki's tunslip6 tool.

2. MQTT - based sensors

The idea for MQTT-based sensors has been to use them only for readings and communications with the MQTT broker. In line with the project specifications those sensors have been limited to use MQTT only and provide different generating functions to simulate readings. Each MQTT sensor connects to the MQTT broker and publishes messages on its own topic but also subscribes to the actuator state topic to receive updates on it by the cloud application and acts accordingly.

For the Broker Mosquitto has been used as broker service, providing also a simple CLI to test both publish and subscribe methods over topics.

To keep messages as simple as possible JSON has been chosen as encoding method, allowing human readability as well as parsing simplicity. The process responsible for publishing uses the etimer struct available in contiki's libraries to periodically send the new reading in a json-formatted message over its MQTT topic and keeps the old value to progressly update it accordingly to the actuator's state.

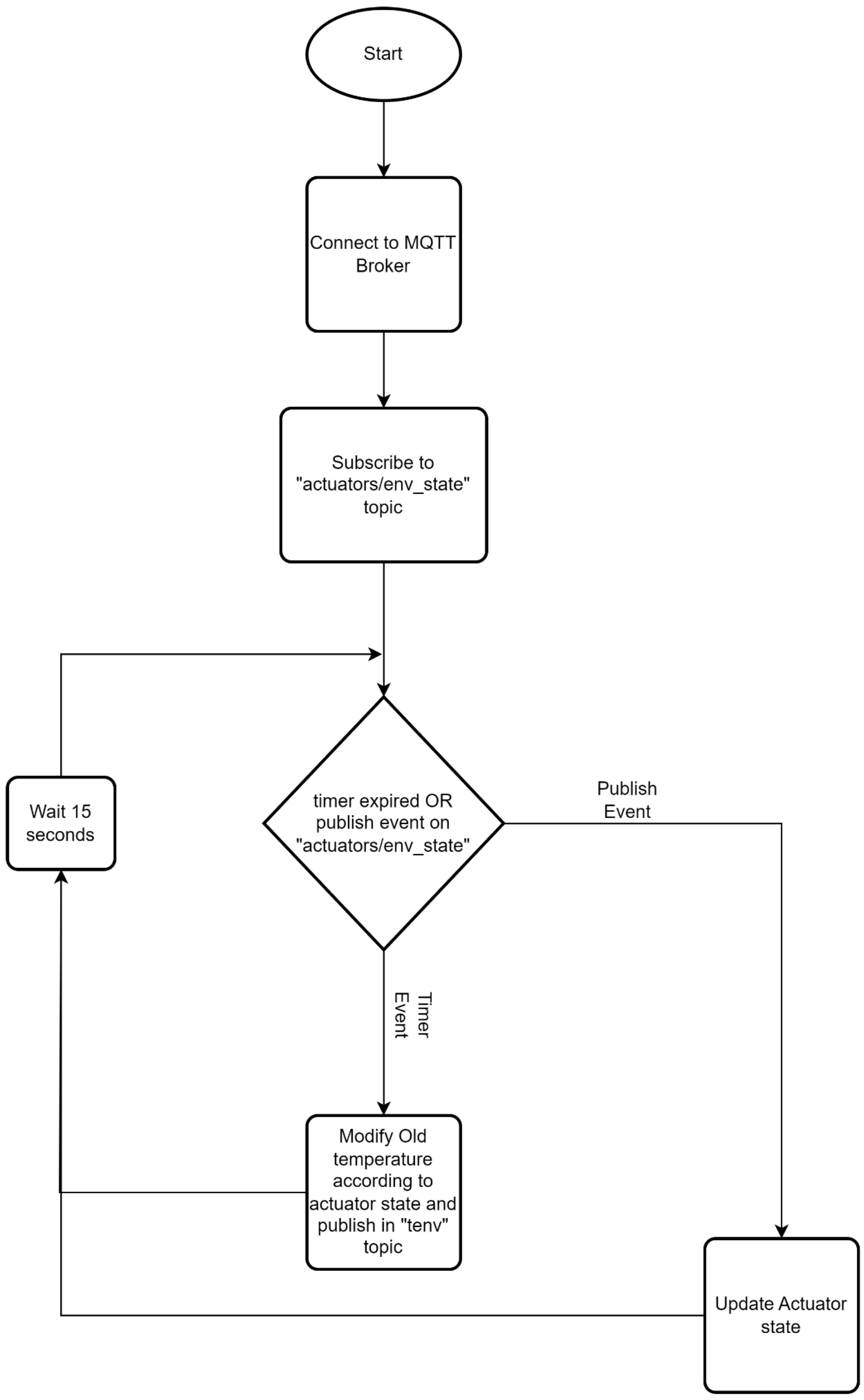
2.1. Progressive update of the readings

MQTT nodes implement different functions to generate new data that are based on the last generated value to progressively update it according to the actuator state. When the java application publishes a new state for the actuator the MQTT sensor receives it and adjusts its state to use the relative generation function.

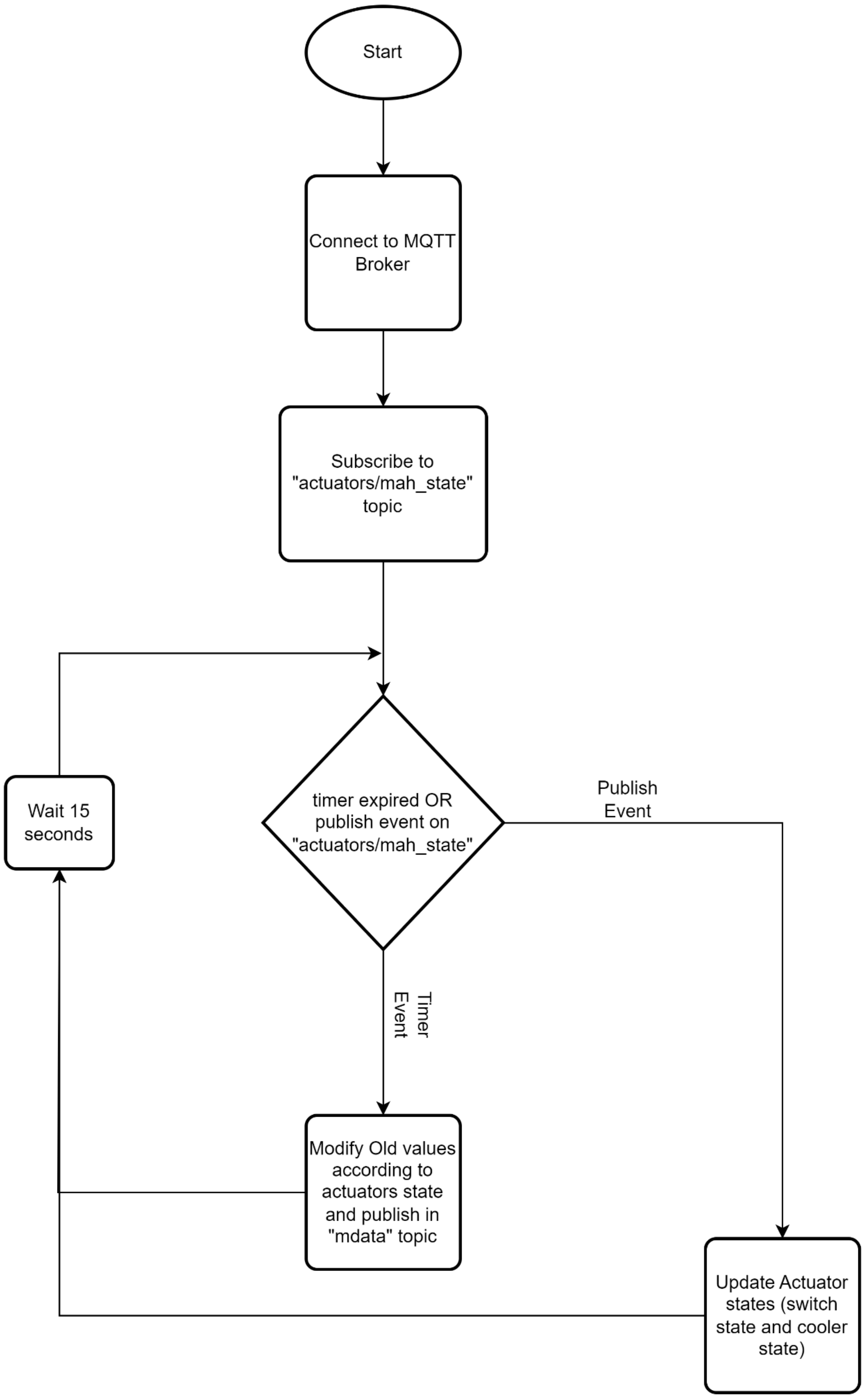
- Environment Temperature
    - OFF → f(old) = old + random_increment in [-2,2]
    - Heating → f(old) = old + random_increment in [0, 5]
    - Cooling → f(old) = old + random_increment in [-5, 0]
- Machine Temperature
    - OFF → f(old) = old + random_increment in [1, 5]
    - Medium → f(old) = old + random_increment in [0, 2]
    - High → f(old) = old + random_increment in [-3, 1]
- Machine Humidity
    - f(t) = (t > 60 ? 10 : 30) + random_increment in [-5, 5]
- Machine Outputs Produced
    - simple random number generation in [0, 50]

In the specific case of Machine’s sensors, the sensing phase is disabled as the switch state turns to **off** to simulate the machine disconnection from the power source.

2.2. Environment Sensing


Start
Connect to MQTT Broker
Subscribe to "actuators/env_state" topic
Wait 15 seconds
timer expired OR publish event on "actuators/env_state"
Publish Event
Timer Event
Modify Old temperature according to actuator state and publish in "tenv" topic
Update Actuator state

2.3. Machine Sensing (Temperature + Humidity + outputs/minute)


Start
Connect to MQTT Broker
Subscribe to "actuators/mah_state" topic
Wait 15 seconds
timer expired OR publish event on "actuators/mah_state"
Publish Event
Timer Event
Modify Old values according to actuators state and publish in "mdata" topic
Update Actuator states (switch state and cooler state)

3. CoAP - based nodes

In the project two CoAP nodes have been developed, one for the environment actuation simulating an air conditioner with Heating and Cooling modes, and one for the machine actuation composed by a physical switch connecting/disconnecting it from the power source, and a cooling system which can be activated in two levels of activity: “medium” and “high”. Switch functions as a security measure for when anomalies are detected (temperature too high, or produced output sensor count too low or malfunctioning).

Both nodes register to the cloud application before activating their resources, once they receive the ACK for the registration, they will start listening on the activated resources (res_envtemp for environment node; res_mahswitch and res_mahtemp for machine switch and temperature respectively). All of the resources provide a POST method for accepting external commands that when succeeds, changes the state of the actuator.

3.1. LEDs usage

LEDs have been used to notify in real time the state of the actuator.

ENVIRONMENT	
STATE	ACTIVE LED
off	None
heating	Purple
cooling	Blue

MACHINE		
Switch State	Cooler State	ACTIVE LED
off	/	Red
on	off	None
on	medium	Cyan
on	high	Blue

The switch OFF state means that the machine has been forcibly shutdown and detached from the power source and Cooler state becomes irrelevant

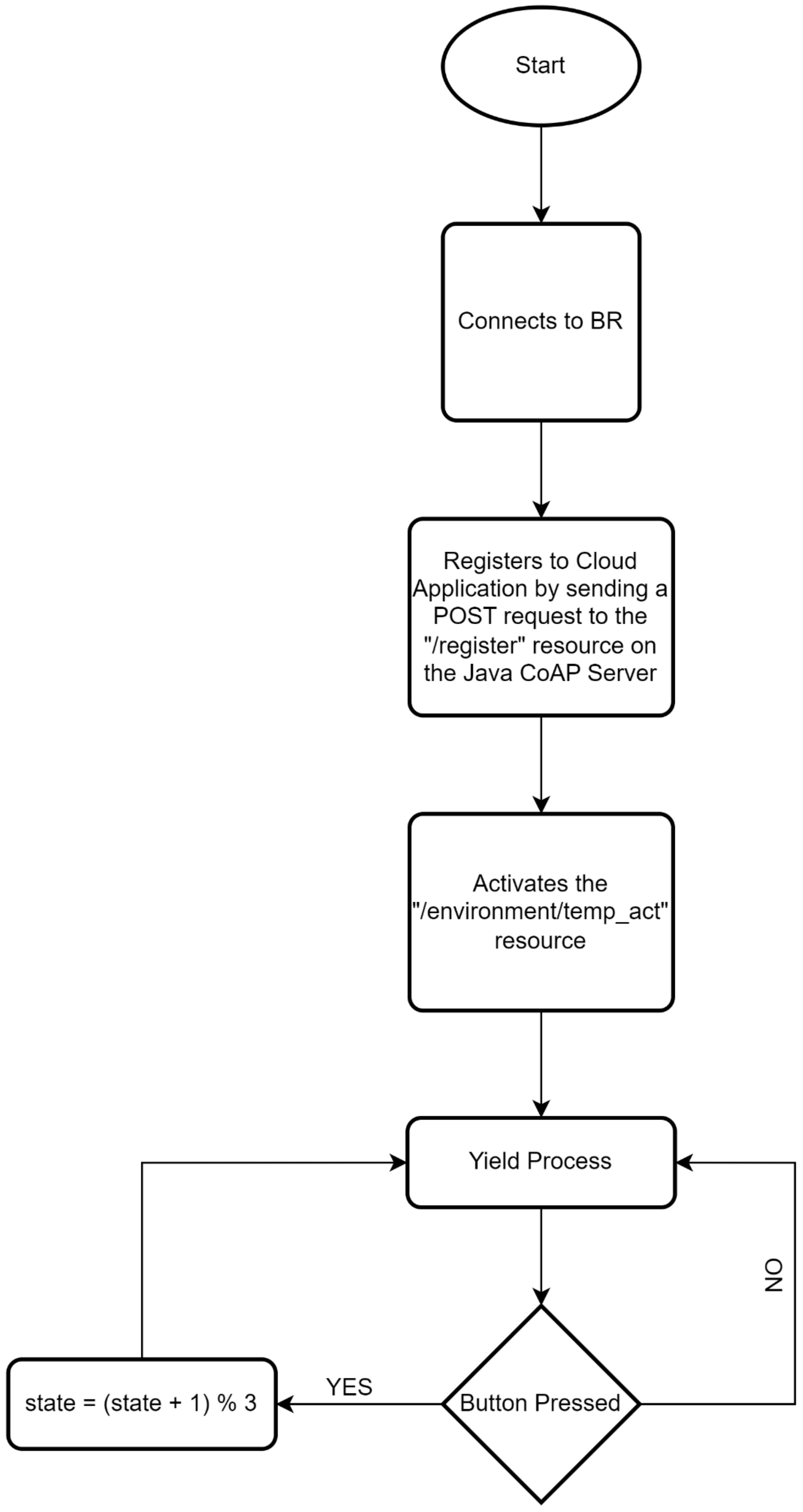
3.2. Buttons usage

Machine		**Environment**
Press	Hold (> 3s)	Press
alternate cooler states	enable/disable switch	alternate conditioner states

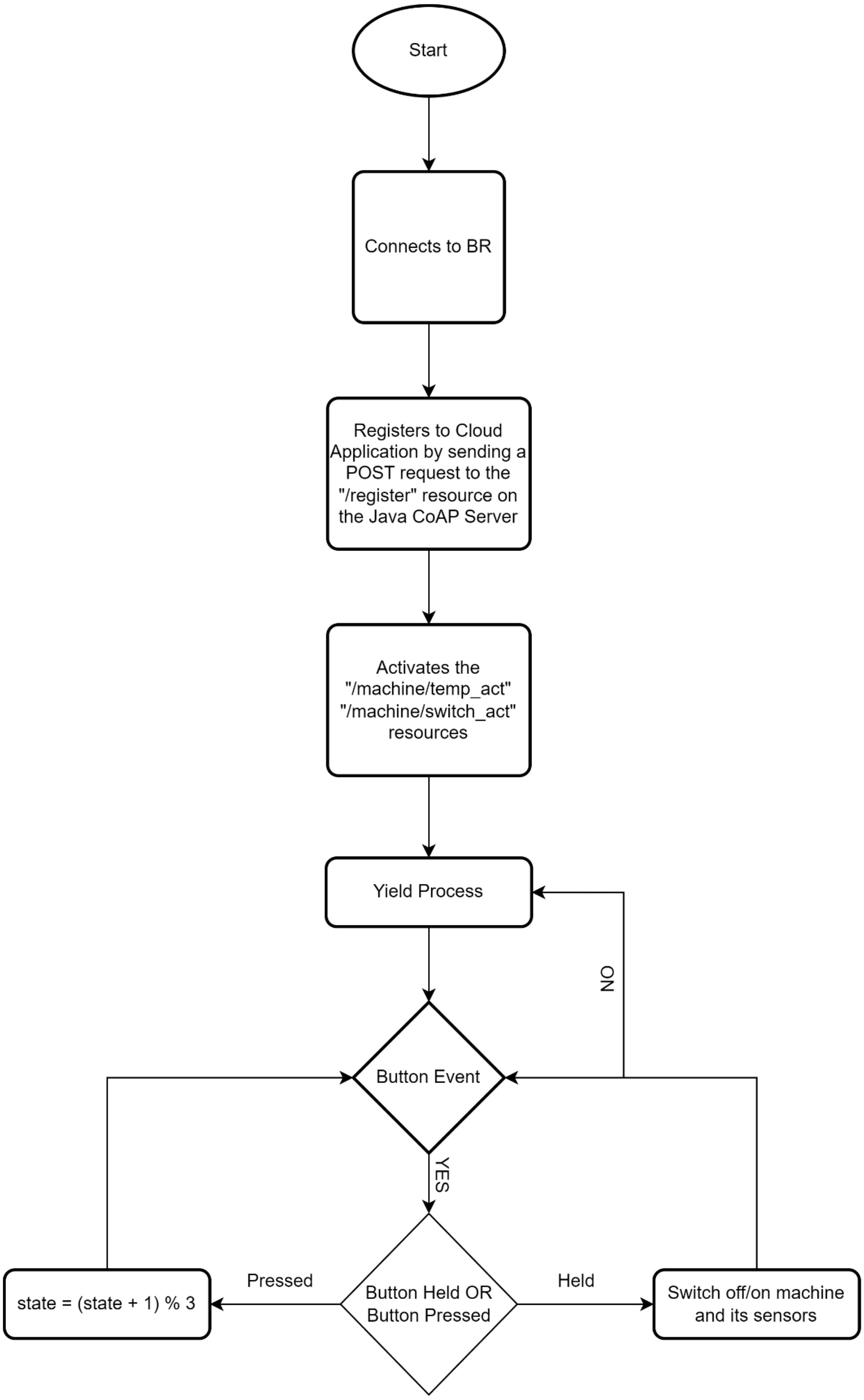
Cooler and Conditioner states are encoded as integers (0, 1, 2) each describing the relative state for the actuator (off; medium; high for machine cooler and off; heating; cooling for the environment conditioner). The alternation consists in applying the (*old_state + 1) % 3*.

Machine switch can be changed by holding the button, if the machine was shutted off, it will force a reset otherwise it will force a shutdown.

3.3. Environment Node


Start
Connects to BR
Registers to Cloud Application by sending a POST request to the "/register" resource on the Java CoAP Server
Activates the "/environment/temp_act" resource
Yield Process
Button Pressed
NO
YES
state = (state + 1) % 3

3.4. Machine Node

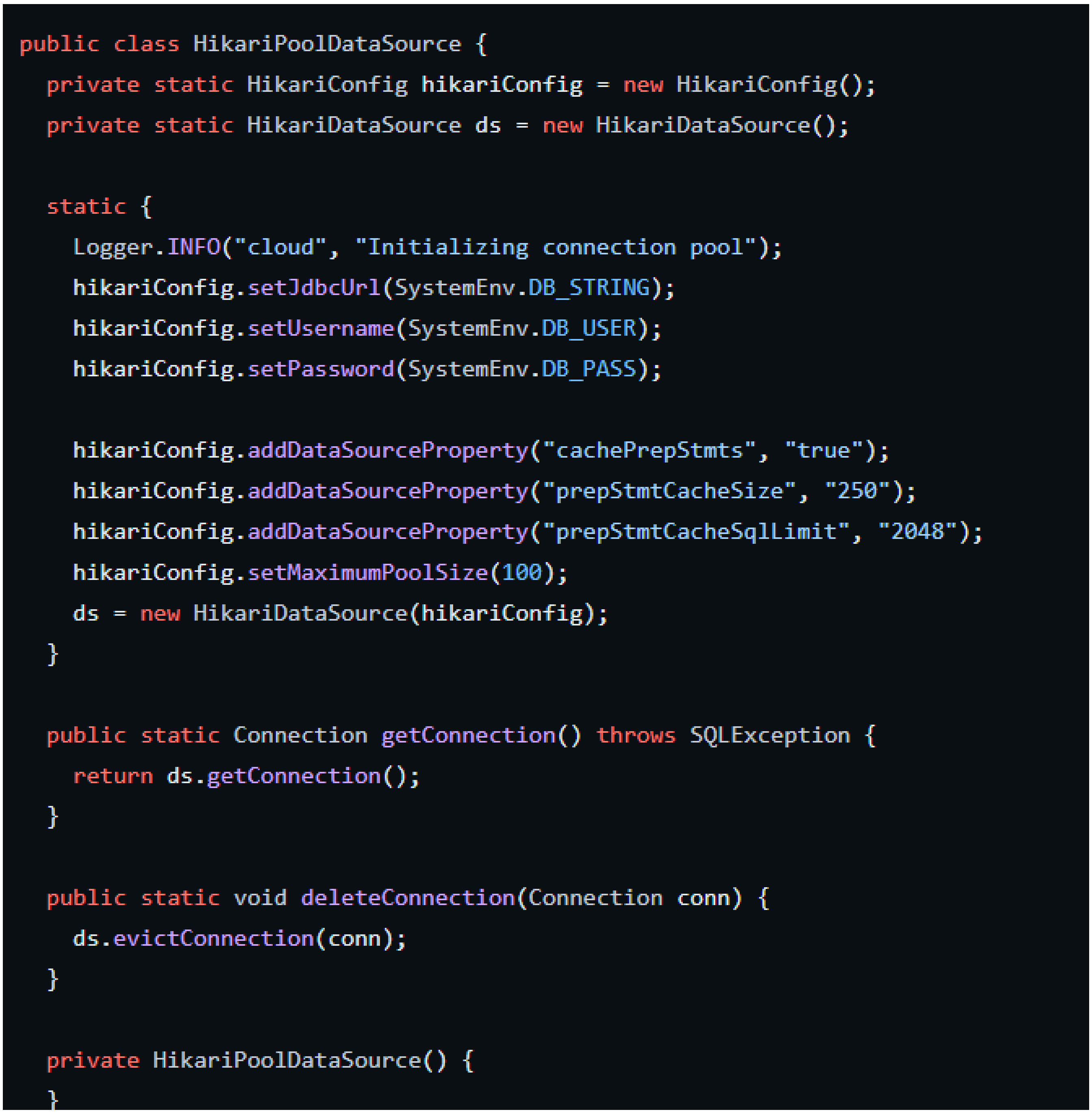

Start
Connects to BR
Registers to Cloud Application by sending a POST request to the "/register" resource on the Java CoAP Server
Activates the "/machine/temp_act" "/machine/switch_act" resources
Yield Process
NO
Button Event
YES
Button Held OR Button Pressed
Pressed
Held
state = (state + 1) % 3
Switch off/on machine and its sensors

4. Java-based Cloud Application

The Cloud Application mainly focuses on several aspects, such as collecting data from the iot devices in order to save them permanently and handle different actions.
It's encharged also to handle the MySQL connections through hikariCP for data storing, establish the connection with the MQTT broker allowing also the MQTT sensors to be notified when a CoAP actuator is working via MQTT broker topic where the cloud application can publish to.

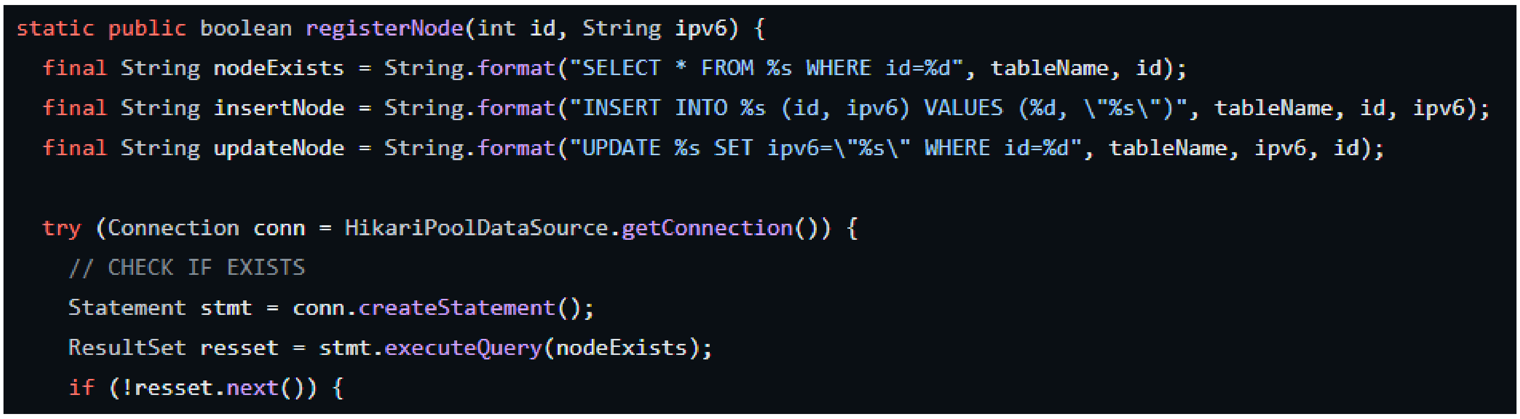
4.1. MySQL Connection pool using Hikari

To allow multi-thread connections to the database, HikariCP has been used to create a connection pool from which to parallelize connections to MySQL for each requesting thread



```java
public class HikariPoolDataSource {
  private static HikariConfig hikariConfig = new HikariConfig();
  private static HikariDataSource ds = new HikariDataSource();

  static {
    Logger.INFO("cloud", "Initializing connection pool");
    hikariConfig.setJdbcUrl(SystemEnv.DB_STRING);
    hikariConfig.setUsername(SystemEnv.DB_USER);
    hikariConfig.setPassword(SystemEnv.DB_PASS);

    hikariConfig.addDataSourceProperty("cachePrepStmts", "true");
    hikariConfig.addDataSourceProperty("prepStmtCacheSize", "250");
    hikariConfig.addDataSourceProperty("prepStmtCacheSqlLimit", "2048");
    hikariConfig.setMaximumPoolSize(100);
    ds = new HikariDataSource(hikariConfig);
  }

  public static Connection getConnection() throws SQLException {
    return ds.getConnection();
  }

  public static void deleteConnection(Connection conn) {
    ds.evictConnection(conn);
  }

  private HikariPoolDataSource() {
  }
```

Extract of the code 1: Config/HikariPoolDataSource.java - Cache has been enabled with 2048bytes of size and max pool size has been set to 100 to correctly handle multiple messages rapidly arriving from nodes.



```java
static public boolean registerNode(int id, String ipv6) {
  final String nodeExists = String.format("SELECT * FROM %s WHERE id=%d", tableName, id);
  final String insertNode = String.format("INSERT INTO %s (id, ipv6) VALUES (%d, \"%s\")", tableName, id, ipv6);
  final String updateNode = String.format("UPDATE %s SET ipv6=\"%s\" WHERE id=%d", tableName, ipv6, id);

  try (Connection conn = HikariPoolDataSource.getConnection()) {
    // CHECK IF EXISTS
    Statement stmt = conn.createStatement();
    ResultSet resset = stmt.executeQuery(nodeExists);
    if (!resset.next()) {
```

Extract of code 2: DAOs/RegistryDAO.java - registerNode function where CoAP nodes are inserted in the nodes_registry table, here a MySQL connection has been extracted from the available connections in Hikari's connection pool and is used inside the try-catch block

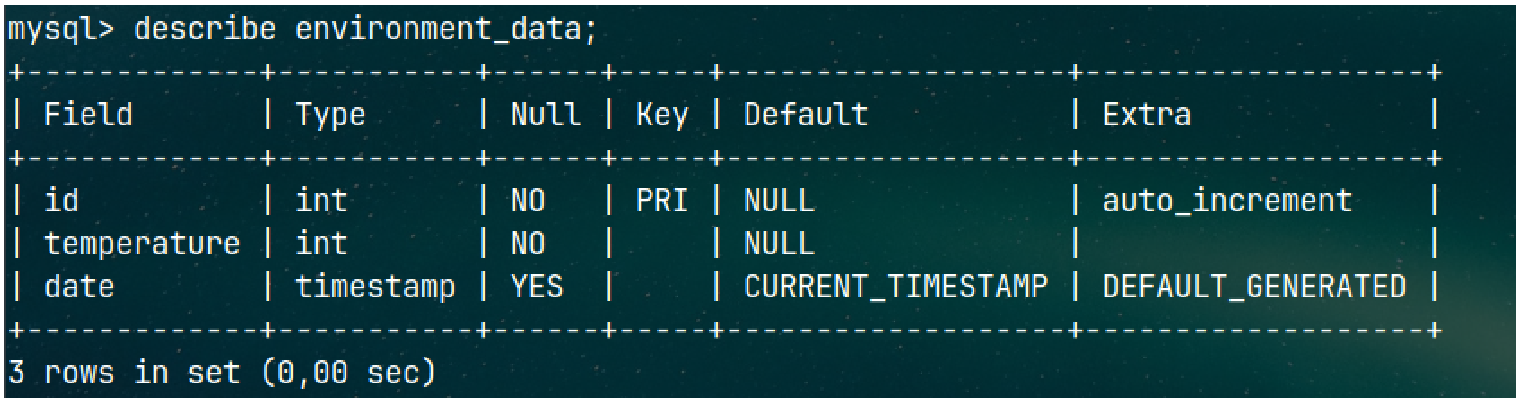
4.2. MySQL Models

To model MySQL tables in the java application DAOs have been used allowing to create a class providing the functionalities exploiting Hikari connections to interact with the database completed by model classes providing the instances parameters to store data correctly.

The DAOs allow to provide some abstract APIs hiding the complexity of the classic CRUD operations, so isolating the application layer from the persistence layer.
Within our application, three DAOs were used:

- Environment and Machine → handle MQTT data storaging for both environment and machine,
- Registry → Stores IPv6 of the CoAP nodes implementing the actuators.

4.2.1. Environment DAO



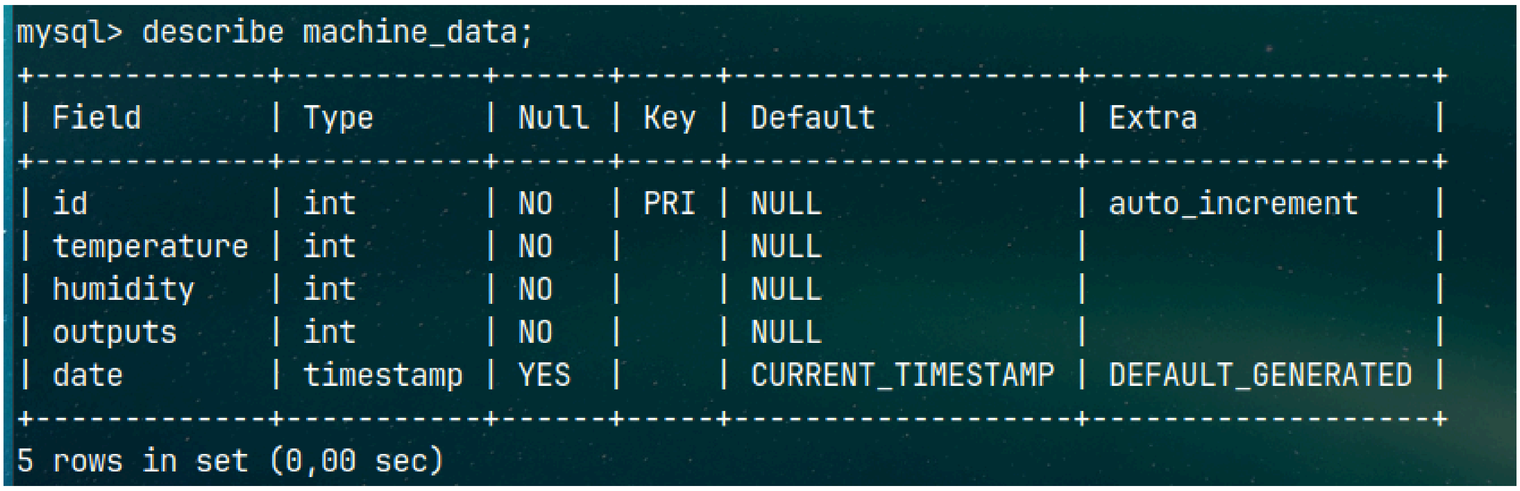
```
mysql> describe environment_data;
+-------------+-----------+------+-----+-------------------+-------------------+
| Field       | Type      | Null | Key | Default           | Extra             |
+-------------+-----------+------+-----+-------------------+-------------------+
| id          | int       | NO   | PRI | NULL              | auto_increment    |
| temperature | int       | NO   |     | NULL              |                   |
| date        | timestamp | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+-------------+-----------+------+-----+-------------------+-------------------+
3 rows in set (0,00 sec)
```

Data received from the Environment sensors are modeled in the database through a table composed of:

- **id:** uniquely identifyies the data received and also offers a quick way to sort it by reception date
- **temperature:** it is the reading performed by the environment sensor
- **date:** timestamp in which the data has been received, used to better organize the data in Grafana interface

4.2.2. Machine DAO

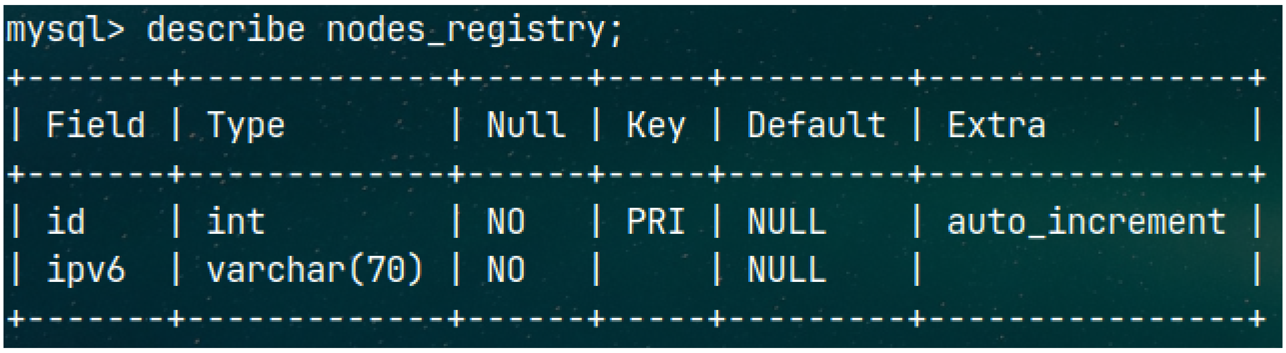
Very similarly to Environment DAO it is described by the sensor readings (temperature, humidity, outputs/min) an id and the timestamp:



```
mysql> describe machine_data;
+-------------+-----------+------+-----+-------------------+-------------------+
| Field       | Type      | Null | Key | Default           | Extra             |
+-------------+-----------+------+-----+-------------------+-------------------+
| id          | int       | NO   | PRI | NULL              | auto_increment    |
| temperature | int       | NO   |     | NULL              |                   |
| humidity    | int       | NO   |     | NULL              |                   |
| outputs     | int       | NO   |     | NULL              |                   |
| date        | timestamp | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+-------------+-----------+------+-----+-------------------+-------------------+
5 rows in set (0,00 sec)
```

4.2.3. Registry DAO

Works as internal registry, records the updated IPv6 of each CoAP node for then using the entry to send CoAP requests when needed (commands)

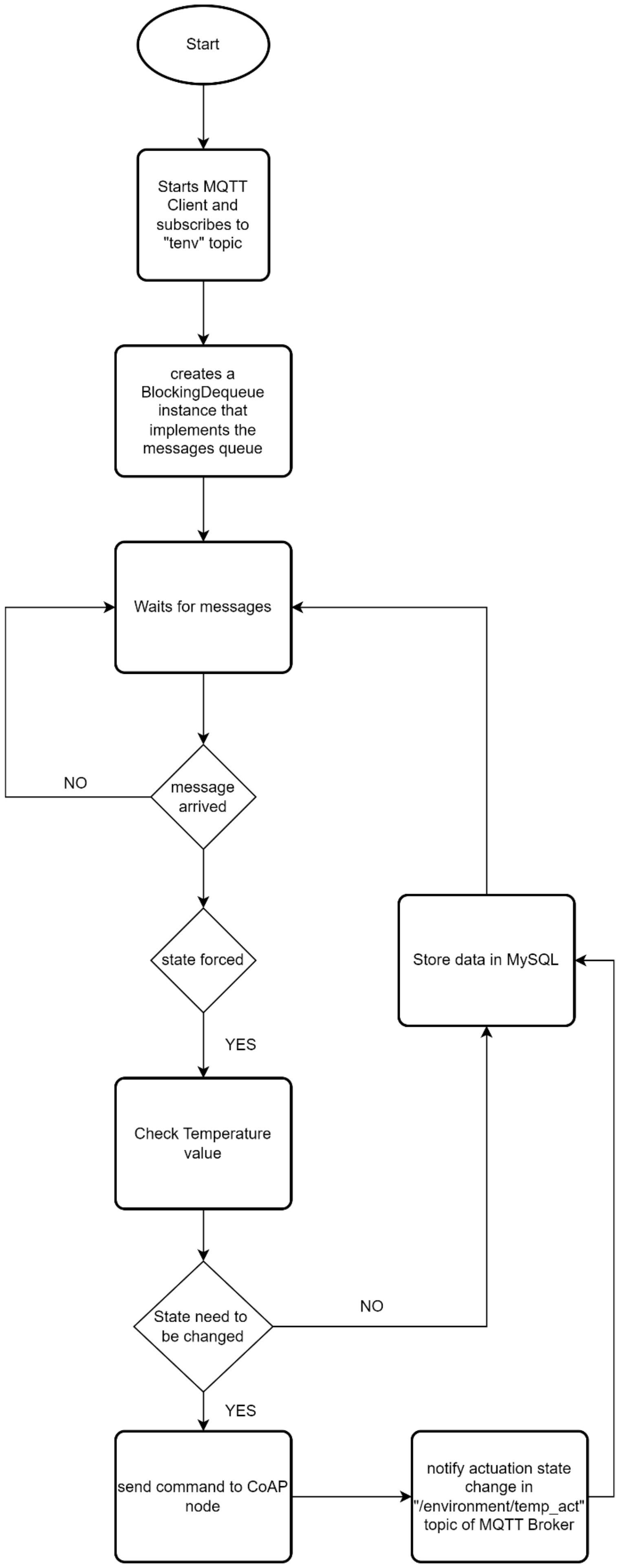


```
mysql> describe nodes_registry;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int         | NO   | PRI | NULL    | auto_increment |
| ipv6  | varchar(70) | NO   |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
```

The id identifies the sensor node, it is hardly coded into the sensor which sends its identification number to the Java Application to be recognized and updated if its IP changes in the network.

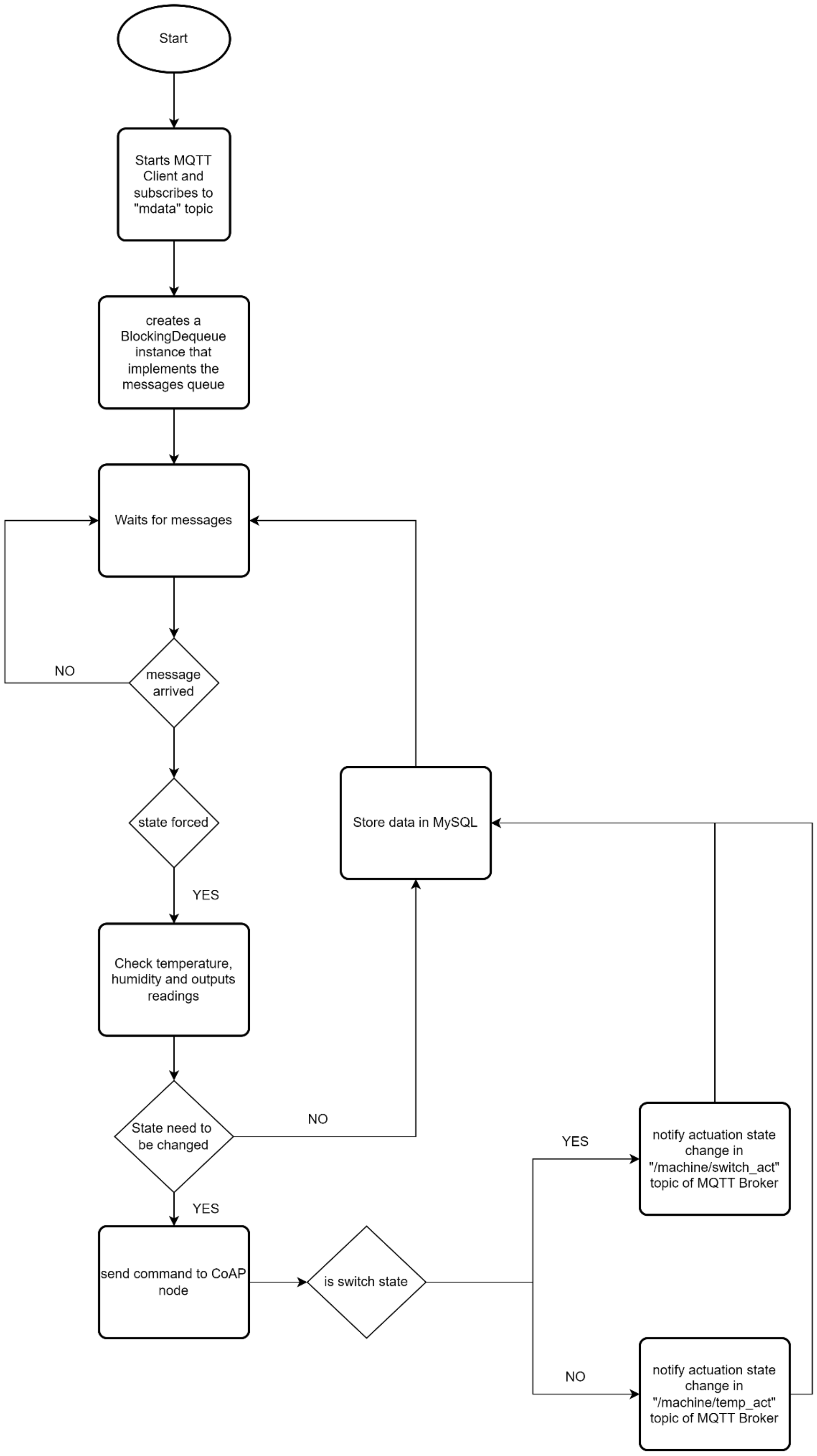
4.3. Environment Controller

Runnable object that executes all the necessary analysis on received data from MQTT Environment sensor and coordinates CoAP and Application states accordingly.


Start
Starts MQTT Client and subscribes to "tenv" topic
creates a BlockingDequeue instance that implements the messages queue
Waits for messages
NO
message arrived
state forced
YES
Check Temperature value
State need to be changed
NO
YES
Store data in MySQL
send command to CoAP node
notify actuation state change in "/environment/temp_act" topic of MQTT Broker

4.4. Machine Controller

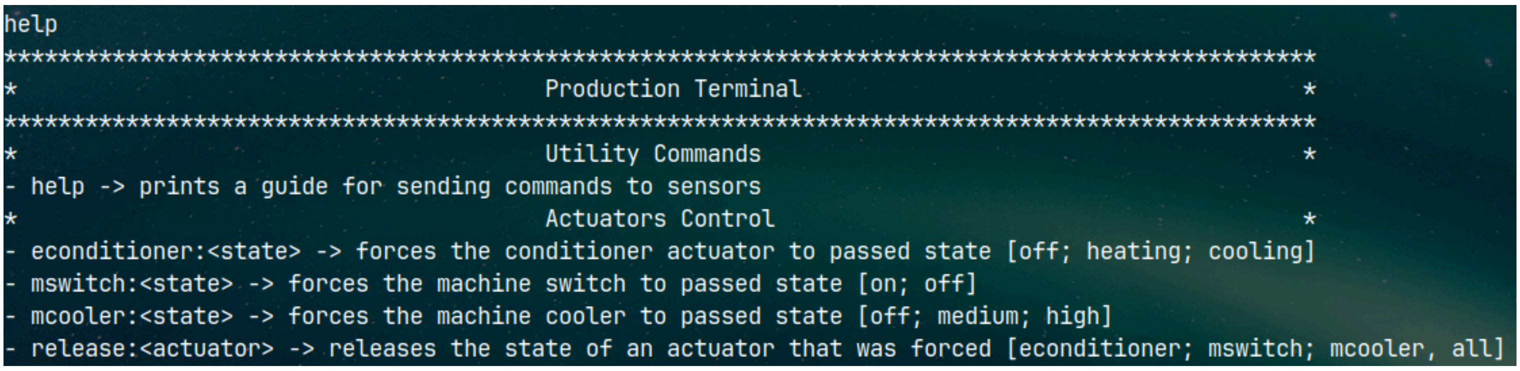
Runnable object that executes all the necessary analysis on received data from MQTT Machine sensor and coordinates CoAP and Application states accordingly.


Start
Starts MQTT Client and subscribes to "mdata" topic
creates a BlockingDequeue instance that implements the messages queue
Waits for messages
message arrived
NO
state forced
YES
Check temperature, humidity and outputs readings
State need to be changed
NO
YES
send command to CoAP node
is switch state
YES
NO
notify actuation state change in "/machine/switch_act" topic of MQTT Broker
notify actuation state change in "/machine/temp_act" topic of MQTT Broker
Store data in MySQL

4.5. Remote Application Controller

It implements a CLI that allows the user to send direct commands to the various actuators enforcing a state on them that cannot be changed until the resource isn't released. Commands provided by the application in the current state:

- **help** → prints all the commands available in the application
- **econditioner:<state>** → changes the environment conditioner state to either [off, heating, cooling]
- **mswitch:<state>** → either forces a shutdown when <state> is *off* or a reboot when it is *on*
- **mcooler:<state>** → changes the machine's cooler state to either [off, medium, high]
- **release:<resource>** → releases the specified resource [econditioner, mswitch, mcooler, all]

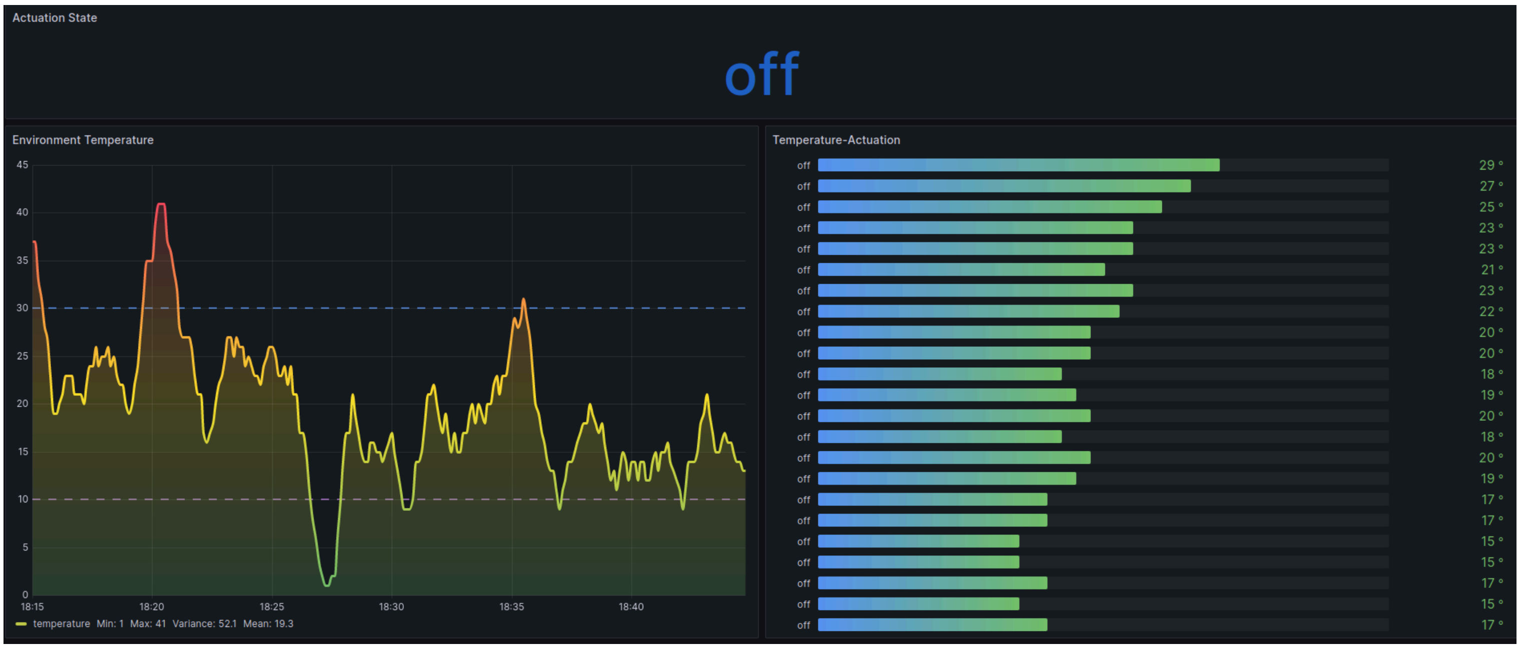


```
help
*********************************************************************************************
*                                       Production Terminal                                 *
*********************************************************************************************
*                                       Utility Commands                                    *
- help -> prints a guide for sending commands to sensors
*                                       Actuators Control                                   *
- econditioner:<state> -> forces the conditioner actuator to passed state [off; heating; cooling]
- mswitch:<state> -> forces the machine switch to passed state [on; off]
- mcooler:<state> -> forces the machine cooler to passed state [off; medium; high]
- release:<actuator> -> releases the state of an actuator that was forced [econditioner; mswitch; mcooler, all]
```

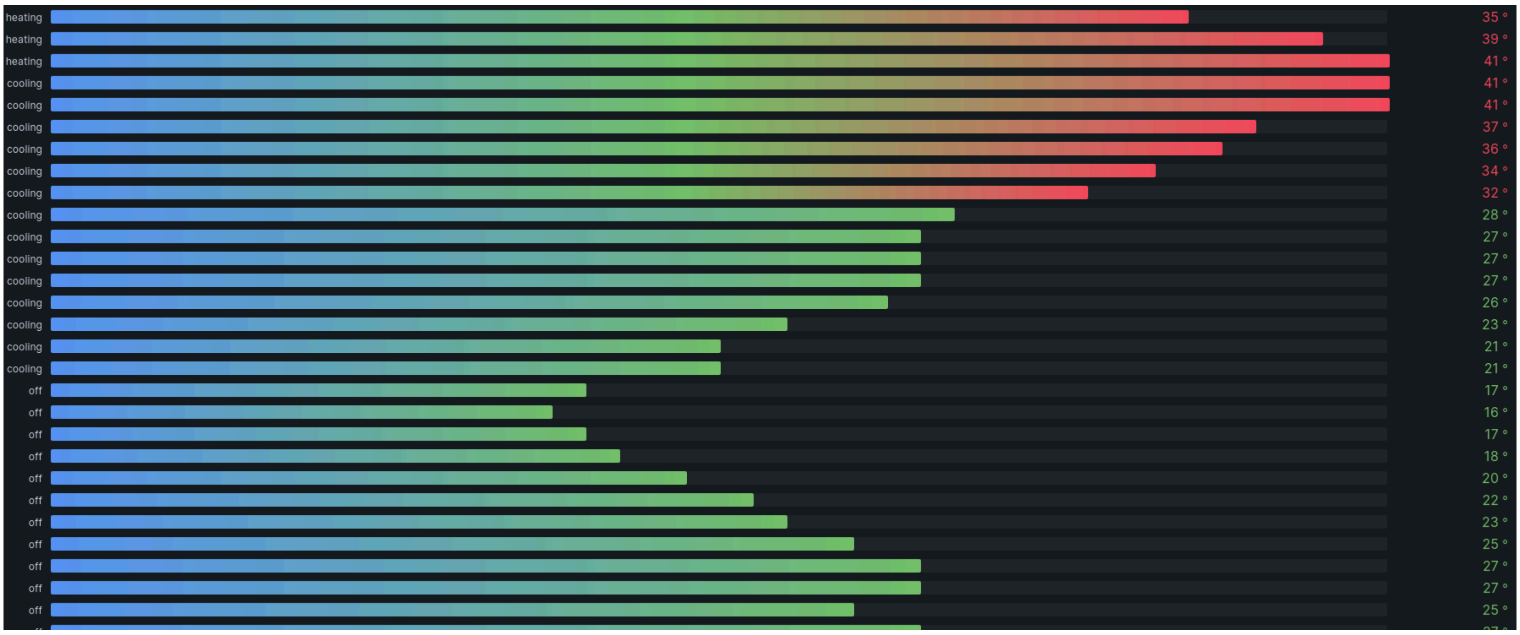
5. Grafana for MySQL Data Visualization

Grafana has been connected to MySQL Database in order to visualize soft in-real-time snapshots of sensor readings both Environmental data and Machine data. Grafana allows the creation of multiple dashboards, so it has been decided to create a dashboard for environmental control, and one for Machinery control.

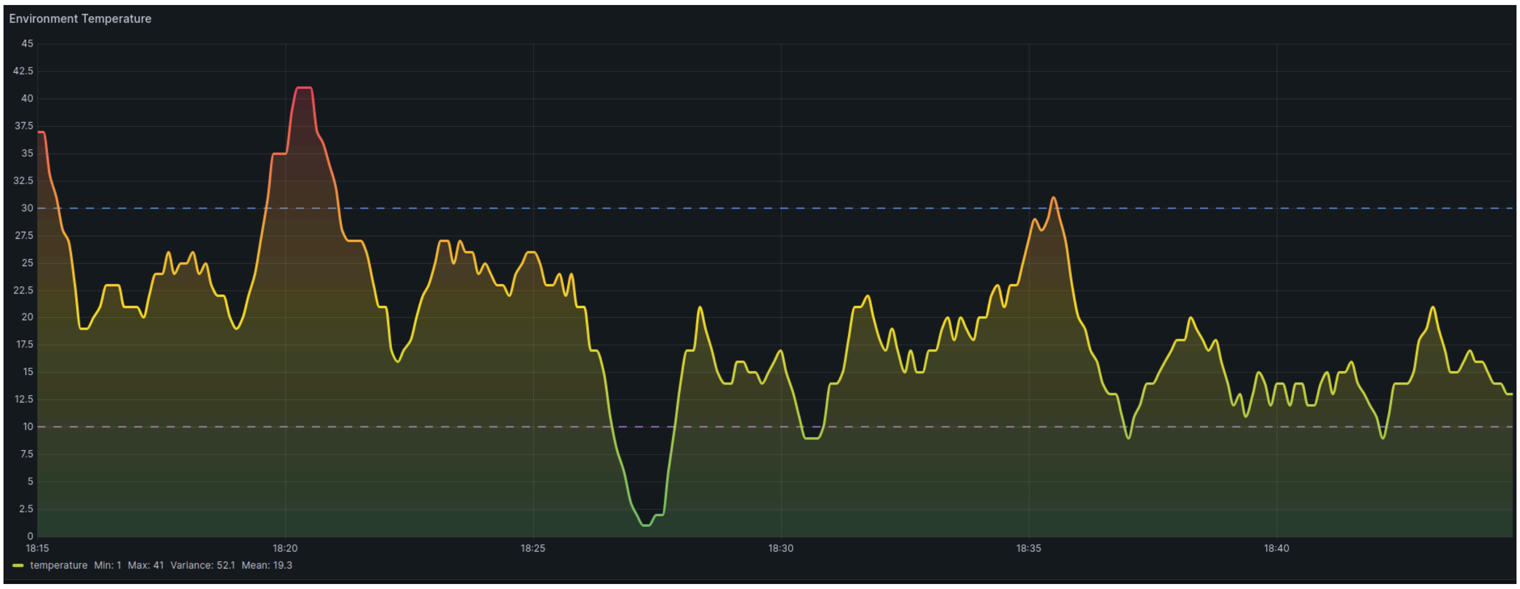
5.1. Environment Dashboard



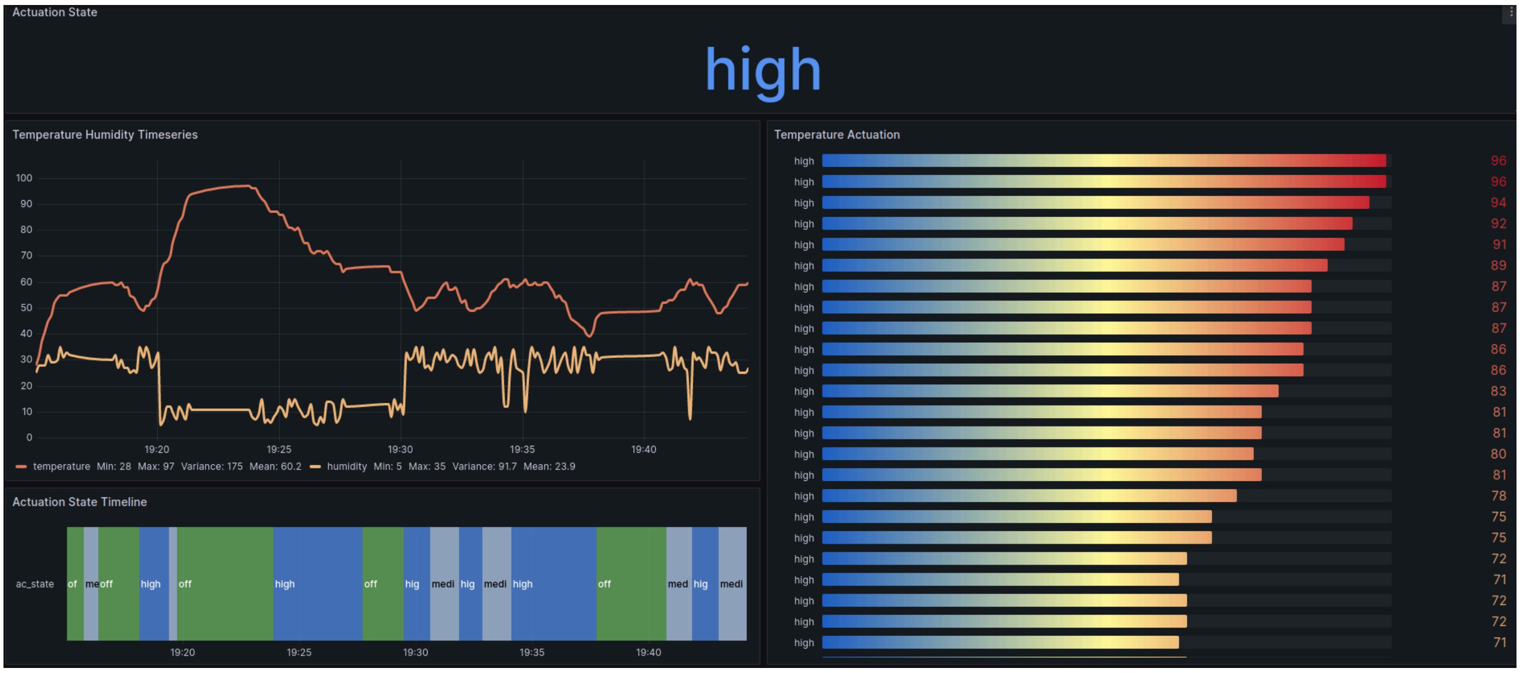
Temperature to Actuation Charts



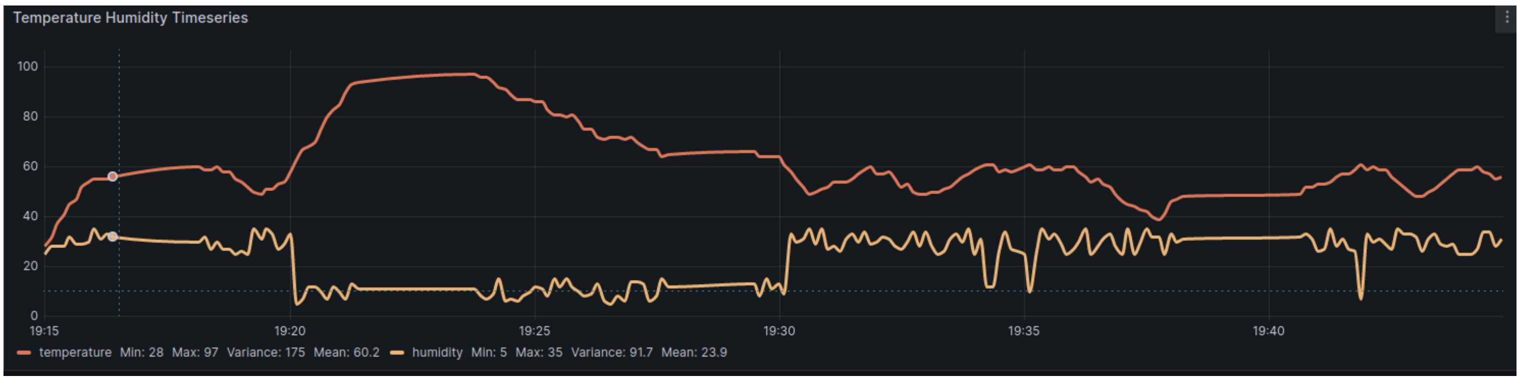
Temperature Timeseries



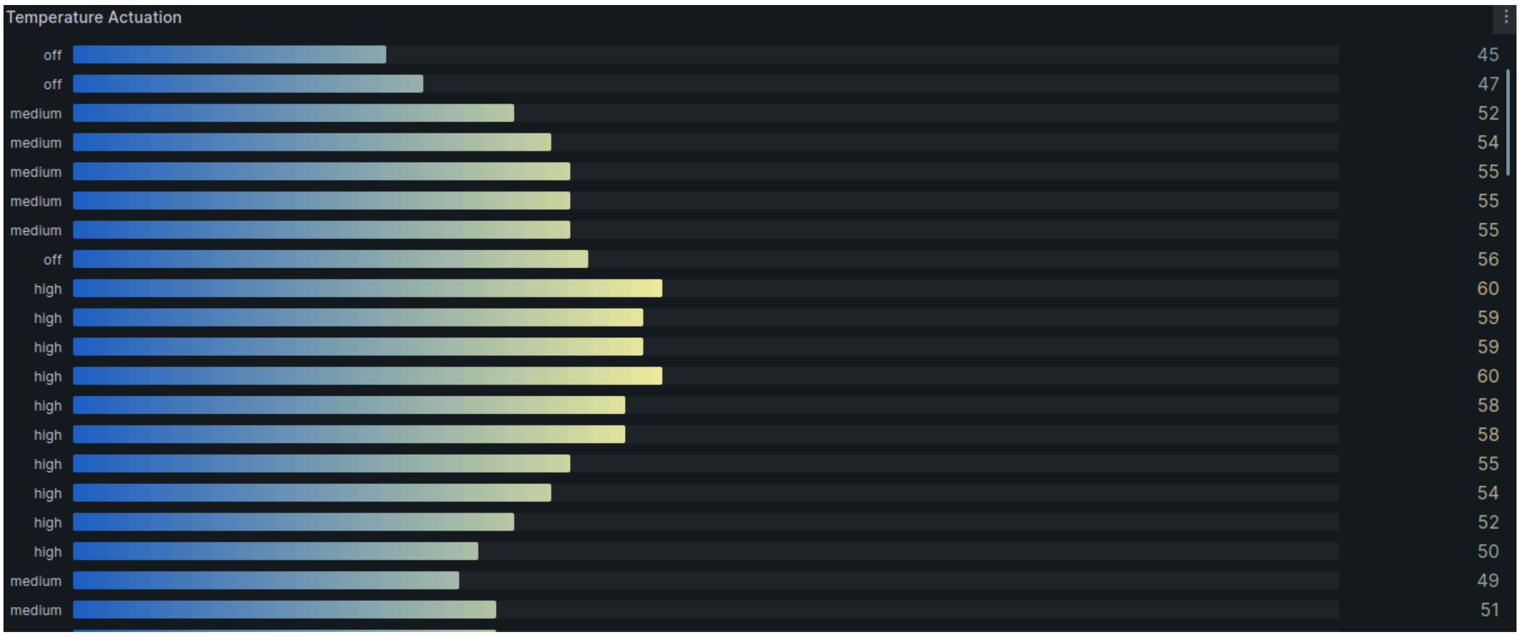
5.2. Machine Charts



Temperature Humidity Time Series



Temperature Actuation Chart


Temperature Actuation
off 45
off 47
medium 52
medium 54
medium 55
medium 55
medium 55
off 56
high 60
high 59
high 59
high 60
high 58
high 58
high 55
high 54
high 52
high 50
medium 49
medium 51