INSTRUMENTS FOR SOFTWARE DEVELOPMENT
IOT PLATFORM USING MICROSERVICES FOR MONITORING NUMERICAL DATA FROM DEVICES

Cătălin-Alexandru Rîpanu 341C3
Automatic Control and Computers
National University of Science and Technology POLITEHNICA Bucharest
`catalin.ripanu@stud.acs.upb.ro`
Laboratory Assistant: Andrei Damian

22 March 2024

Abstract

For this **project**, I **propose** implementing a **platform** for **collecting**, **storing**, and **visualizing numerical data** coming from a large number of **Internet of Things devices**. For its **implementation**, we will adopt a **simplified** (but efficient) architecture inspired by the operation mode of the **most well-known public cloud services** of this type. Typically, such a solution **comprises** the following components.

- **Devices** with sensors connected to the Internet
- A **message broker** (MQTT Broker)
- A **time-series database** (InfluxDB)
- An **adapter** that parses data from messages received from **IoT devices** and adds them to the database
- An interface for **data visualization** and management (Grafana).
- A **database management utility** for the MySQL database that stores information about the accounts used for **authentication** (PhpMyAdmin).
- A **Python server** implementing login functionality using Flask.
- A **database for accounts** (MySQL).
- An **API Gateway** behind which the application can be **publicly exposed** (Kong).
- **CI/CD elements** to automate the deployment process following changes to the source code (Gitlab).
- A **open-source tool** for monitoring and alerting, which collects metrics by reading them from **HTTP endpoints** of the monitored **components** (Prometheus)

Each of these **components**, except for the **sensor devices** (which will be **simulated** at the time of assignment evaluation), will appear as a **software solution** included in this project **github repository**. **InfluxDB** and **Grafana** will be used for the **database** and **data visualization** interface, respectively, and an **open-source** implementation will be chosen for the **MQTT messages broker**. The entire **solution** will be delivered as a **Docker Swarm** stack (or **Kubernetes**), where each component **appears** as a distinct service described within the **.yml file**. In-depth details regarding the **functioning** of each component are described in the following **sections**.

1 Implementation

The solution stack **contains 7 important** services, namely:

- **MQTT broker** using the latest version of its corresponding image
- **InfluxDB database** version 1.8 which **does not require any authentication logic (this feature will be visible using a login server implemented using Flask**)
- **MQTT adapter** implemented in Python, described in its own Dockerfile in its directory
- **Grafana instance** which allows visualization of data sent by the adapter through dashboards written in **JSON** and added in the directory **grafana_db/grafana_provisioning/grafana_dashboards**
- **API Server** for **authentication** and **authorization** features
- **the MySQL database** for storing accounts information
- **PhpMyAdmin** for monitoring and managing the aforementioned database

The solution can be executed using **just one command**, namely **./run.sh** (this is a script that performs the entire setup). Additionally, **deleting all local** images/volumes created by **run.sh** can be done using the command **./clean.sh** (this is a script that removes all things added by run.sh).

1.1 MQTT Broker

For the message broker, an Open-Source solution will be used (among those available with official images on hub.docker.com) that implements at least version 3.0 of the MQTT protocol. This is an **open protocol** of the publish/subscribe type, widely used in the **Internet of Things (IoT)/Machine-to-Machine (M2M)** environment due to its highly efficient resource utilization of network, CPU, and memory of connected devices.

The broker will serve as the **communication gateway to the platform for IoT devices** and is **usually** deployed as a post-authentication (and authorization) service, accepting and delivering messages only from/to authorized clients, according to an Access Control List (ACL).

This **broker** awaits potential connections from **IoT devices** on **all IP addresses of the machines on which it runs using the default TCP port (1883)**. Furthermore, it uses the configuration file **mosquitto_mqtt_vol/mosquitto.conf** to allow connections from **clients** and permits any client to publish messages on any topic and subscribe to any topic (including all topics using the wildcard character #)

Moreover, it is located in the **broker_adapter_network** along with the adapter to communicate only with it, and last but not least, the decision was made for the broker to **save all files** in the aforementioned directory to avoid losing logs that could help in a potential **failure situation**.

1.2 InfluxDB Database

The **system's database for numerical values** will be represented by an **instance of InfluxDB**. This will be exposed only within this **solution stack** and will allow, without authentication, the addition, reading, and manipulation of **time series data**. It is configured as follows:

- Data retention is **unlimited** (to store, for each time series, a history as long as needed, without automatically deleting/aggregating data with older timestamps)
- The time resolution of the data is **maximum** (storing data timestamps with a minimum precision of one second in the database)
- Data is **persistently stored** in one or more Docker volumes, so that deleting and restarting the stack does not lead to data loss

This **service** was configured using the bash script from the directory **database/init_influx_db.sh**, which utilizes **CLI** commands to create a **database** named **IoT_Devices** with infinity data retention. Similarly to above, all information sent during execution is retained in the directory **database/influx_db** by copying all files from **/var/lib/influxdb** (a path within the **Docker container**).

Obviously, to limit communication with other containers, this service is located in these networks **adaptor_influx_network** and **influx_grafana_network**, so that it can communicate only with the **Adapter** and **Grafana instance**.

1.3 MQTT Adapter

This component **will persistently** connect to the **MQTT broker**, subscribe to **all messages** sent through it (using **the wildcard topic** - #), and will **insert them** into the database, according to the specifications in **Section 1.1**. For this component, processing **MQTT messages** will be event-based (when they reach the broker), and sending data to the **TSDB** will be done as quickly as **possible**.

The adapter component **needs to display** appropriate logging messages to facilitate **tracking of the added data**. These messages will be displayed so that they are visible using the docker service logs command for the corresponding service, **only if the DEBUG_DATA_FLOW** environment variable has the value "true". The absence of this environment variable should not affect the program's functionality, however, debug messages **will not be generated** if it is missing.

This service was implemented in **Python** using the **paho.mqtt.client** package and is located in the **adapter** directory. Additionally, the corresponding **adapter image** can be built using the **Dockerfile** located in the same directory. For the design of this entity, an **OOP implementation** coded in the source **adapter.py** was preferred, utilizing an **MQTT client** to connect to the **broker** and an **InfluxDB client** to communicate with the **database** (the adapter is responsible for adding processed information to this database, not the clients transmitting data to the broker). To **filter out messages** with **incorrectly formatted topics**, a **regular expression** was used to **reject them** (using the **match** function from the **re** package).

1.4 Grafana Instance

For **data visualization**, I will use an instance of the latest version of Grafana (Grafana 5). This will connect to the **Influx database** and allow **graphical visualization** of the data. Within the scope of the project, it will expose a specific web interface (through port 80 of all IP addresses of the machines it runs on), allowing a user authenticated with the **username assistant** and the **password grafanaIDP2023** to view and edit two predefined dashboards.

The data is from the stack's database and the dashboards were written in **JSON** and are located in the directory **grafana_db/grafana_provisioning/grafana_dashboards** (in **grafana_db/grafana_provisioning** there are also **several .yml files** used by Grafana to load these **dashboards** and **set the data source**).

1.4.1 Dashboard for visualizing IoT data from UPB location

The **first predefined** dashboard will include **2 panels** (a graph and a table) for all **datasets** generated by all devices in the **UPB location**. A row in the table will contain **all measurements** taken at a specific moment in time (after applying the **grouping policy**). In case the devices measure at different times, a row **may be incomplete**. Below are the **construction details** of the dashboard:

- **Dashboard** name: UPB IoT Data
- Format of **time series** names: DEVICE.METRIC (example: RPi_1.TEMP)
- Display **period**: last 6 hours
- Data **grouping interval**: 1 second
- Data **grouping policy**: arithmetic mean
- Automatic **dashboard refresh period**: 30 seconds

1.4.2 Dashboard for monitoring battery levels

The **second dashboard** will provide an overview of **battery levels** for all devices that have sent data to the platform in the **last 48 hours**. It will consist of **2 panels** (a graph similar to that in the **UPB IoT Data dashboard** and a table with **statistical aggregations**). As a convention, I will consider that, if capable, all devices **report the battery level** as a value of a key **named BAT** within the data dictionary. The construction details of the dashboard are as follows:

- **Dashboard** name: Battery Dashboard
- Format of **time series** names: DEVICE_NAME (example: RPi_1)
- Data **grouping interval**: 1 second
- Data **grouping policy**: arithmetic mean
- Automatic **dashboard refresh period**: 30 minutes
- Table header for **the statistical table**:

- **Time Series** (DEVICE_NAME)
- **Current Value** (last available)
- **Minimum** Value, **Maximum** Value, **Average** Value

By using **regular expressions** in the **JSON sources**, the **2 dashboards** are **able** to adhere to the **requirements of a practical application** (such as adding **newly connected devices** and **new metrics** that appear).

1.5 Login API Server

This server plays a **crucial role** in the system as it is responsible for managing communication with the **Kong API Gateway**. Its primary task is to handle requests from clients, particularly devices, for storing numeric data, such as **time series**, into the **Influx database**.

Moreover, the server interacts with the **MySQL database** to ensure that the devices initiating these operations **possess the requisite permissions**. This verification process is crucial for maintaining the integrity and security of the system, ensuring that only **authorized devices** can execute **read** and **write** operations.

1.6 PhpMyAdmin

This utility, commonly referred to as **PhpMyAdmin**, serves the purpose of providing a graphical interface for **visualizing and monitoring** the MySQL database responsible for managing the accounts generated by the **devices**. Its functionality allows users to interact with the **database efficiently**.

Additionally, it's worth noting that this service can be accessed via **port 8081**. This means that users can connect to PhpMyAdmin through their web browsers using the specified port number, enabling them to utilize its **features seamlessly**.

1.7 MySQL Database

This **service** has been meticulously configured to maintain a **persistent storage** of all information pertinent to the accounts utilized for **authentication** and **authorization** purposes by devices that connect to the infrastructure.

To maintain a **secure** and **efficient** communication environment between containers, it is strategically deployed solely within the designated networks: the **app_network**, which houses the **login server**, and the **interface_network**, where the **database management utility resides**. This **segmentation** ensures that **communication** is restricted to the **necessary components**, enhancing overall system **security** and **performance**.

2 Communication within the application

According to the description, the entire application will expose only 3 ports: port 1883 for the **MQTT broker**, port 80 for the web interface of the visualization module **Grafana** and port 8081 for the **PhpMyAdmin** utility.

All **backend-to-backend communications** will be carried out through private IPs within the Docker Swarm stack (or Kubernetes).

To limit the impact of various **cyberattacks** on the application and to **strengthen its security** (hardening), multiple Docker networks will be used to restrict communication between containers that **do not typically** require such interaction.

Thus:

- The **MQTT broker** will communicate only with the adapter that adds information to the **Influx database**
- The **Influx database** will communicate only with the adapter and the visualization interface (Grafana)
- The **adapter** will communicate only with the MQTT broker and the **Influx database**
- The **visualization interface** (Grafana) will communicate only with the **Influx database**

All tasks aimed at designing and completing the project will be **fulfilled by me**.

Github **link**: here.

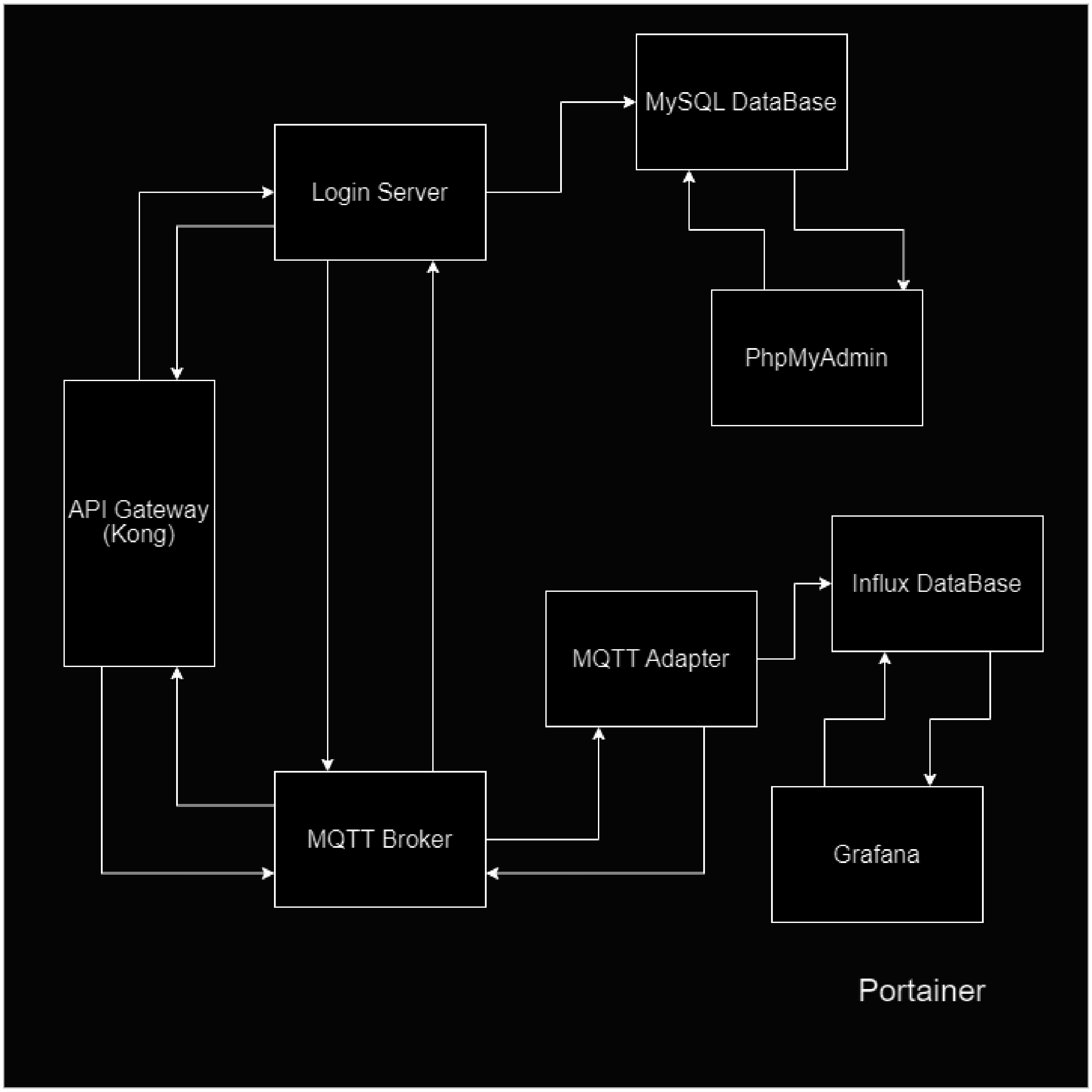

Figure 1: Internet Of Things platform architecture