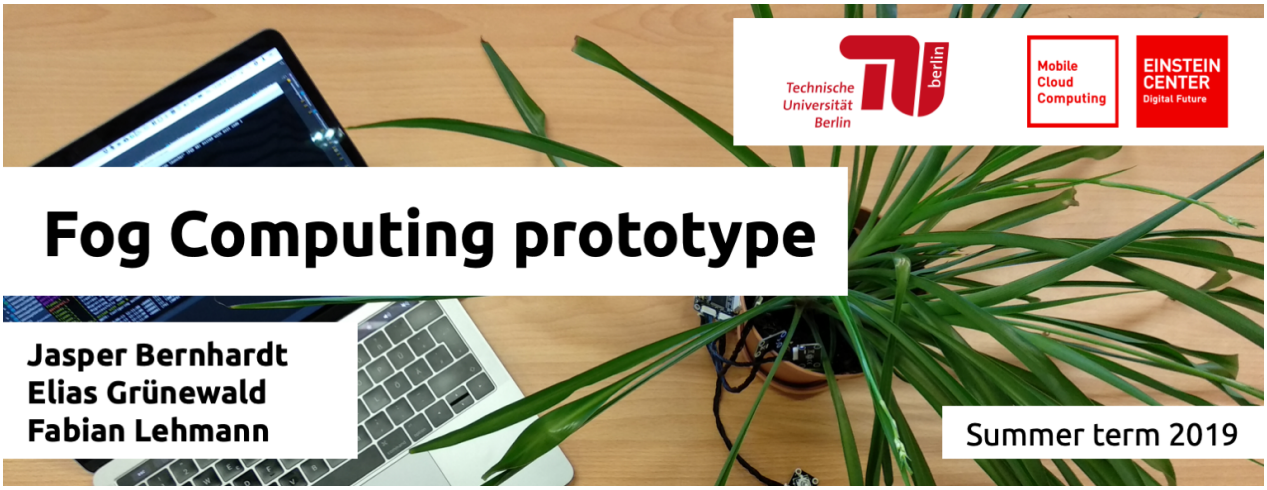


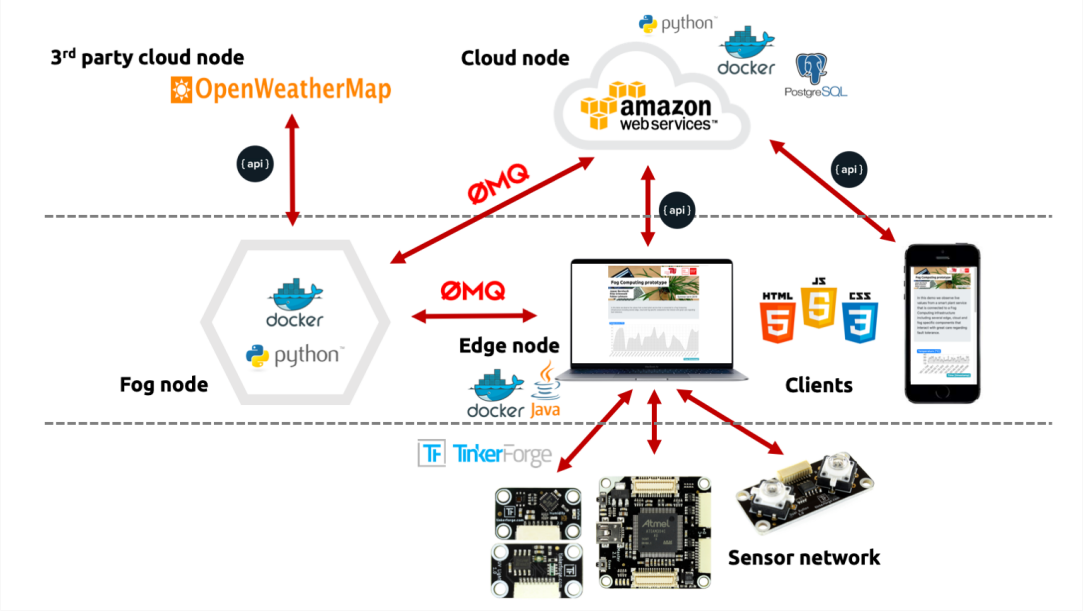
Fog Computing
Supervisors: Prof. Dr.-Ing. David Bermbach, Martin Grambow, Jonathan Hasenburg

Chair Mobile Cloud Computing
Faculty IV - Electrical Engineering and Computer Science
Technical University of Berlin

Abstract

In this project we observe live values with a smart plant service that is connected to a Fog Computing infrastructure including several edge, cloud and fog specific components that interact with great care regarding fault tolerance. The setup is easily extendable and built upon state-of-the-art technologies.

Architecture



Architecture

Components

In the following, the different components of the Fog Computing infrastructure are described briefly.

Fog

The Fog is a Python Programm that heavily utilizes ZeroMQ and multiple `asyncio` tasks to process incoming messages and cache messages in case of connection issues. These routines are restarted in case of unexpected crashes. Mainly these tasks handle the internal queues for processing new control message for the Edge and uploading data to the Cloud. Additionally, one async-task is started per connected Edge device and published control messages periodically. Lastly one routine forwards requests for new edge ids to the cloud. Incoming messages from the Edge are only validate for a correct message format and then copied into internal queues to process them asynchronously. Upload of reports to the Cloud are uploaded in bulks periodically. The Fog is build with Docker and only has one Python-Interpreter (one CPU core) to process messages. If deployed on machines with multiple cores the fog might need a TCP Load Balancer.

To start the fog use `docker-compose up --build`. These are all settings available via environment variables for the Fog Container.

ENV Variable	Default	Description
IS_DEBUG_LOGGING	`'False'`	Enables a more verbose output
CONTROL_MESSAGE_TICK_RATE	`4`	Tickrate of Fog -> Edge Control Message in Hz
CONTROL_MESSAGE_TICKER_UPDATE_TIMEOUT	`300000`	Timout in Milliseconds
EDGE_RECEIVER_LISTEN_PORT	`5555`	TCP Port for the Edge Receiver
EDGE_CONTROLLER_PORT	`5556`	TCP Port tor the Edge Controller Message
EDGE_ID_GENERATOR_LISTEN_PORT	`5557`	TCP Port for the Edge ID Relay
CLOUD_UPLOAD_URL	`'tcp://cloud:5558'`	Upstream URL for the Report Messages
EDGE_ID_UPSTREAM_URL	`'tcp://cloud:5559'`	Upstream URL for the Edge ID Relay
EDGE_RECEIVER_MAX_QUEUE_LENGTH	`10000`	ZeroMQ Max Queue Length for the Edge Receiver
EDGE_ID_RELAY_MAX_QUEUE_LENGTH	`10000`	ZeroMQ Max Queue Length for the Edge ID Relay
INTERNAL_MESSAGE_CACHE_MAX_QUEUE_LENGTH	`100000`	ZeroMQ Max Queue Length for Caches
CLOUD_SUBMIT_TIMEOUT	`60000`	Timout for Report Messages to Upstream
CLOUD_ID_RELAY_CLOUD_TIMEOUT	`5000`	Timout for Edge ID Relayed Messages to Upstream
WEATHER_LIMIT_HOURS_IN_FUTURE	`6`	Hours in forecast to consider for calculating Water & Light status
WEATHER_FORCE_ALWAYS_WATER_RETURN_VALUE	`-1`	Debug value to override the calculated value for a static; `-1` disables the override
WEATHER_FORCE_ALWAYS_LIGHT_RETURN_VALUE	`-1`	Debug value to override the calculated value for a static; `-1` disables the override
WEATHER_CITY	`'Berlin'`	City to use for the weather forecast; See OpenWeatherMap

Cloud

The Cloud is build with an REST API and an sperate ZeroMQ component to accept messages from the Fog and persisting them for a frontend. To start all cloud based components use `docker-compose up --build`.

ZeroMQ

The Cloud ZeroMQ is written in Python. It is build around ZeroMQ to accept messages from Fog clients. It is build similar to the Fog using `asyncio` for multiple loops to work off the Queue. Messages to the REST API are submitted in bulks periodically.

These are all settings available via environment variables fore Cloud Container.

ENV Variable	Default	Description
IS_DEBUG_LOGGING	`'False'`	Enables a more verbose output
DATABASE_REST_URL	`'http://postgrest:3000/'`	Upstream URL for the Database REST Service
FOG_RECEIVER_LISTEN_PORT	`5558`	TCP Port for the Report Message from Fog
ID_FOG_RECEIVER_LISTEN_PORT	`5559`	TCP Port for the Edge ID Requests
FOG_RECEIVER_MAX_QUEUE_LENGTH	`10000`	ZeroMQ Max Queue Length for the Report Message from Fog
ID_RECEIVER_MAX_QUEUE_LENGTH	`10000`	ZeroMQ Max Queue Length for the Edge ID Requests
INTERNAL_MESSAGE_CACHE_MAX_QUEUE_LENGTH	`100000`	ZeroMQ Max Queue Length for Caches

REST

A simple REST API build with PostgREST 5.2 and utilizing PostreSQL 11 as Datastore for the Application. It only does validation of the message and restricts access to only `GET` and `POST` with relevant params. Basic filtering and Ordering are supported for the frontend. Migrations for PostgreSQL are applied with the FlyWay 6 Beta to have versioned database schema.

The API exposes 4 relevant endpoints:

- `GET` and `POST` `http://<server>/sensor` with the following model

```
{
  "id": "string",
  "edge_id": "string",
  "temperature": "integer",
  "temperature_sensor_id": "string",
  "humidity": "integer",
  "humidity_sensor_id": "string",
  "uv": "integer",
  "uv_sensor_id": "string",
  "started_at": "string",
  "stopped_at": "string"
}
```

- `GET` and `POST` `http://<server>/edge_id`

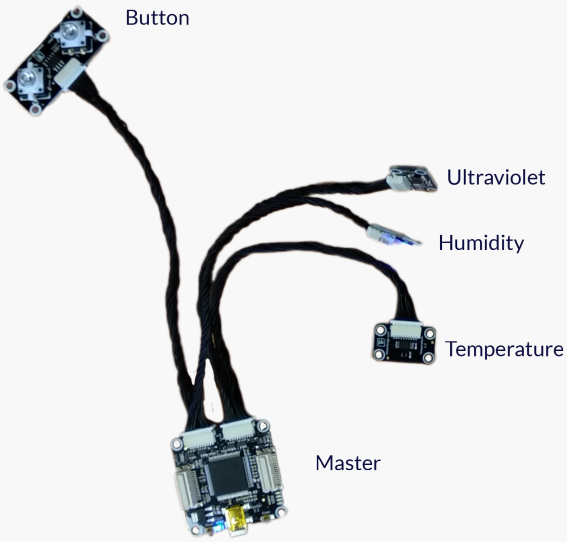
```
{
  "id": "integer",
  "plant": "string"
}
```

Edge

The Edge Client is a light weight Maven Java programm. It collects data from sensors and sends it to the fog. The application also communicates with the user and gives feedback about the current values. The edge handles different sensors:

- *1..n* humidity sensors
- *0..n* temperature sensors
- *0..n* ultraviolet sensors
- *1* button

A sample setup looks like this:



Sensors

Overview

To start the java project run:

```
install brickd https://www.tinkerforge.com/de/doc/Software/Brickd.html#brickd
install brickv https://www.tinkerforge.com/de/doc/Software/Brickv.html#brickv
cd edge\edge.client\
mvn clean package
java -jar -jar target\edge.client.jar 18.185.92.86 myplant vTy (h,HF1) (u,xkb)
(t,EKx)
```

Firstly, one has to define the address of the fog node. There after the name of the supervised plant, for example: "myplant", "vTy" is an example ID for the button and the tuples in the end present the sensors. "h" is humidity, "t" - temperature, "u" - ultraviolet and the second param the ID.

To get reliable data one has to place the sensors directly on the plant. The humidity sensor should touch the potting soil. The ultraviolet sensor should be aligned to the sun.

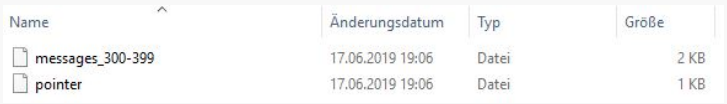
Plant setup

Afterward you are supposed to connect the sensors to the master brick and the master brick to a (cheap) edge device. For example a Raspberry Pi with speakers or a Mac Book Pro (not a common edge device).

Overview

Persistence

Dealing with energy outages or other problems, where we can't send all messages we build a modul which persists all messages on the disk until they are send. Therefor it creates small files up to 100 messages. Each message is appended to such a file. A separate pointer file stores the last processed message. When all messages of a file are processed, the file is deleted to free up space.



Message files

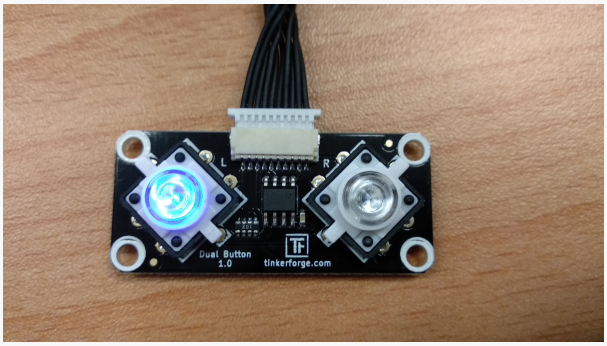
Message format

To communicate between fog and edge we try to reduce the message size as much as possible. That's why we don't send simple json messages.
For example, we didn't send the start end end timestamp of a measurement. Instead we send the start end the smaller difference. Also we don't send whether it needs water or sun seperatly, we combine it to one variable using simple math. needs water = {0,1}, needs sun can be {0,1,2} (less, same, more). Our result is *3 * water + sun*

Human interaction

To make our programm accessible to all we build different ways of communication.

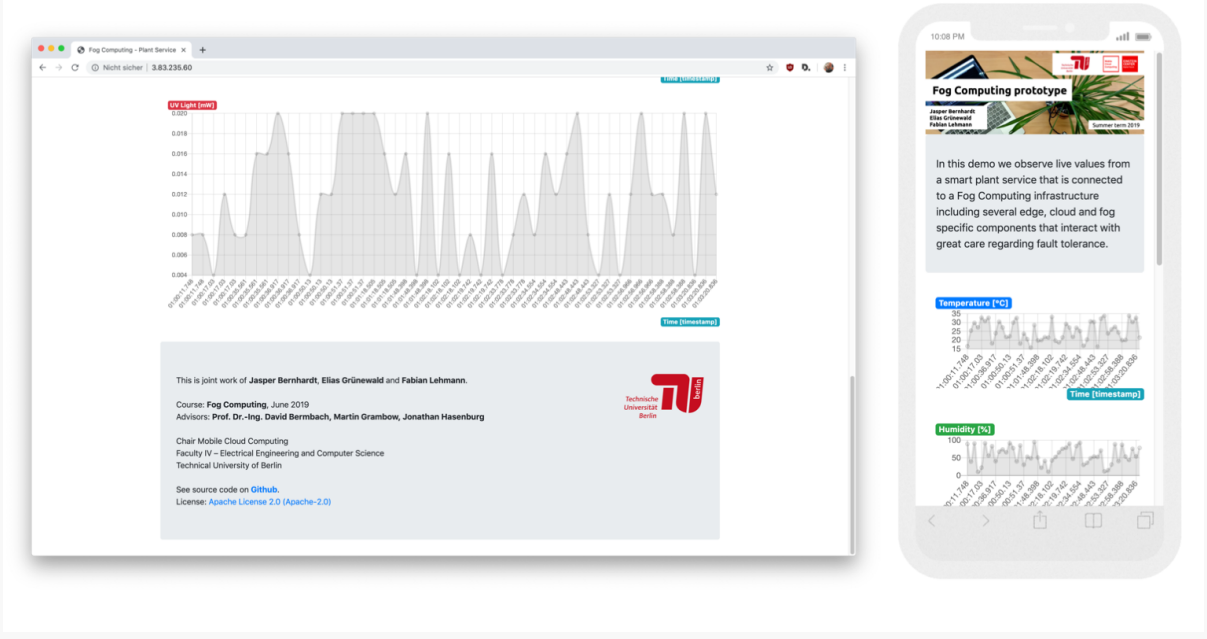
button	state	meaning
left	off	don't need water
left	on	needs water
left	blinking	needs water, based on a local calculation
left	press	reads out loud the current values
right	off	needs more sun
right	on	no sun chance
right	blinking	to much sun
right	press	reads out loud the needs of the plant



LED button

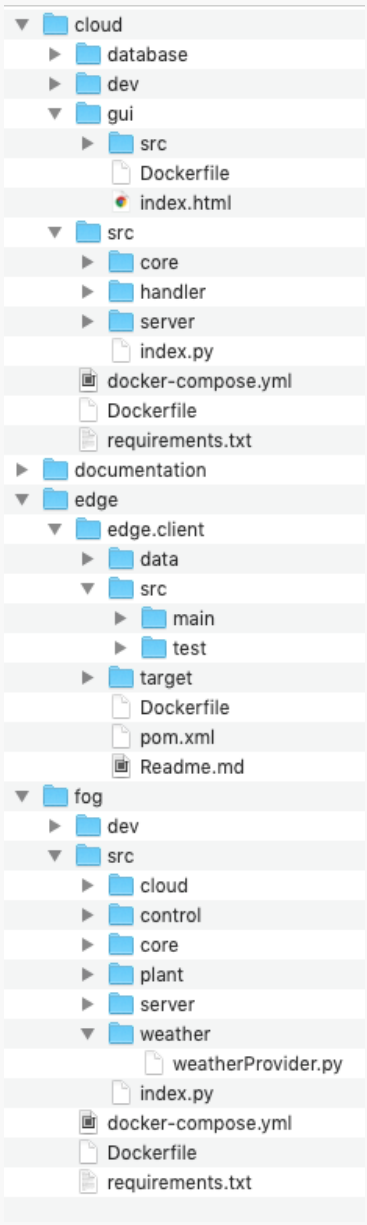
Graphical User Interface/Frontend

In order to display historic but also current data, a Graphical User Interface was built upon standard web technologies such as HTML, CSS and JavaScript. Standard libraries were included such as *JQuery*, *Bootstrap*, and *Chart.js*. The user interface is served with an *nginx* webserver and the deployment is *Docker*ized. The page is responsive and renders correctly on different devices. The frontend supports different query parameters such as `sensor`, `limit`, `time` and `retry`. Example: http://localhost/?sensor=2&limit=50&time=1000&retry=1000



Frontend displayed on Desktop (left) and mobile (right) device

Code structure



Filesystem tree

Demo video

Please refer to https://www.youtube.com/watch?v=1XjiOtrxI_4 for a minimal demo of the plant setup. Under https://www.youtu.be/7Vdtk6EMx8c a short demo of the frontend can be seen.

License

The software is licensed under GPLv3 and is published as Open Source Software under

https://github.com/fog-computing-tu-berlin/prototype