

CPS Humidity Homework

Janos Benjamin Antal

January 14, 2018

Prerequisites

To build the application on Linux Mint 18.3 the following steps are needed:

- Install required packages with apt:

```
- libboost-all-dev
- libcurl4-openssl-dev
- libssl-dev
- uuid-dev
- rapidjson-dev
- build-essential
- cmake
- g++
- git
```

Use the following command:

```
$ sudo apt install libboost-all-dev libcurl4-openssl-dev
libssl-dev uuid-dev rapidjson-dev build-essential cmake g
++ git
```

- Install [cURLpp](#) with the steps described [here](#):

```
$ sudo apt-get remove libcurlpp0
$ mkdir curlppbuild
$ cd curlppbuild
$ git clone https://github.com/jpbarrette/curlpp.git
$ cd curlpp
$ cmake .
$ sudo make install
```

- Install AzureIoT C SDK from [source](#). After building the SDK use `sudo make install` to copy the header and the lib files to the system include path.
- Download and install [RTI Connex DDS 5.3](#)
- Set the `NDDSHOME` environment variable to the root of RTI Connex DDS, e.g.: `/opt/rti_connex_dds-5.3.0`

Build

1. Clone git repository

```
$ git clone https://github.com/antaljanosbenjamin/  
cps_homework.git
```

2. Generate source code from .idl files

```
$ cd cps_homework  
$ $NDDSHOME/bin/rtiddsgen -language C++11 -stl -d DDS/Config  
/common -replace idl_files/Config.idl  
$ $NDDSHOME/bin/rtiddsgen -language C++11 -stl -d DDS/  
Decision/common -replace idl_files/Decision.idl  
$ $NDDSHOME/bin/rtiddsgen -language C++11 -stl -d DDS/  
Schedule/common -replace idl_files/Schedule.idl  
$ $NDDSHOME/bin/rtiddsgen -language C++11 -stl -d DDS/  
Humidity/common -replace idl_files/UvegHaz.idl  
$ $NDDSHOME/bin/rtiddsgen -language C++11 -stl -d DDS/  
Weather/common -replace idl_files/Weather.idl
```

3. Build

```
$ mkdir build  
$ cd build  
$ cmake ..  
$ make [-j 8]
```

Run demo

1. Start humidity publisher

```
$ ./cps_main h <humidityDataFilePath>
```

As result of this command the application will read the data file and start to publish a humidity value every 4 minutes. The file shall contains a humidity value per line. Each humidity value is a decimal number. See example [file](#).

2. Start IoTEdge

```
$ ./cps_main e <weatherApiKey> <azureConnectionString> <
  scheduleFilePath>
```

The meaning of parameteres are the following:

- **weatherApiKey**: an API key for <http://api.airvisual.com>
- **azureConnectionString**: the connection string of the device used Azure IoT Hub
- **scheduleFilePath**: path to a CSV file which stores the schedules time intervals. See example [file](#).

The IoTEdge module is responsible for communication with Azure IoT Hub, the weather information system and also to send schedule information through DDS topic.

3. Start humidity controller

```
$ ./cps_main c
```

The controller receives the required informations and sensor values and also make decisions based on the collected data. It also sends the decision input and output to the IoTEdge in order to store them in the cloud.

System architecture

The system consists three modules:

- IoTEdge
- Controller
- Publisher

The publisher module exists only for testing purposes, so this documentation doesn't contains it's details.

IoTEdge

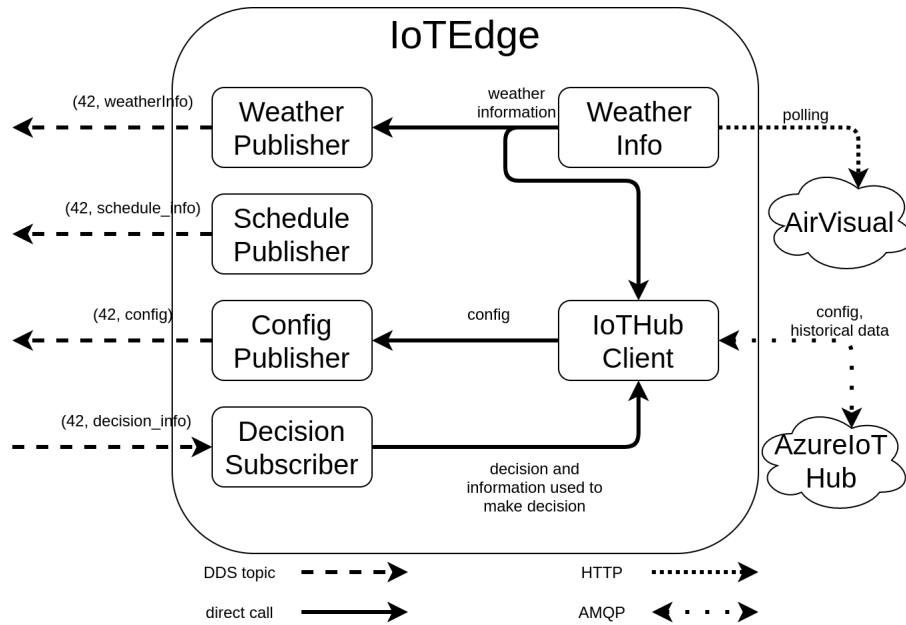


Figure 1: Architecture of IoT Edge

As the figure 1 shows, the IoTEdge translates the messages received from the cloud and the weather forecast API to DDS messages and vica versa. The DDS Interface Description Language exactly describe the contents of the used DDS messages:

```
struct Config {
    double maxTemperature;
    unsigned long maxPollution;
    unsigned long minHumidity;
    unsigned long maxHumidity;
};

struct Schedule {
```

```

        boolean scheduled;
        unsigned long until;
        unsigned long sentTS;
};

struct UvegHaz{
    string ID; //@key
    double Value;
    long TimeStamp;
};

struct Weather {
    double temperature;
    unsigned long tempTS;
    unsigned long pollution;
    unsigned long pollTS;
};

enum Decision { CLOSE, OPEN };

struct DecisionInfo {
    unsigned long decisionTS;
    Decision decision;
    Config config;
    Weather lastWeather;
    Schedule lastSchedule;
    UvegHaz lastHumidity;
};

```

The most complex part of the IoTEdge module is the IoTHub Client. It is responsible for translating and communicating between the IoTEdge and AzureIoT Hub. It sends the received sensor, weather and decision data to the cloud, and also receives config message from it. A config message contains the maximum outside temperature, the maximum outside pollution and the desired minimum and maximum temperature in the classroom. For example:

```

{
    "maximumTemperature" : 50,
    "maximumPollution" : 55,
    "minimumHumidity" : 42,
    "maximumHumidity" : 46
}

```

- maximumTemperature: a temperature value in Celsius. If the outside temperature reaches this limit, the controller logic mustn't decide to open the window.
- maximumPollution: a pollution value in Air Quality Index. If the outside pollution reaches this limit, the controller logic mustn't decide to open the window.

- minimumHumidity, maximumHumidity: the desired range of humidity expressed as relative humidity. The goal of the system is to keep humidity within this range.