Contents

[Contents 3](#_Toc9512906)

[1 Setting up the database & front-end components 4](#_Toc9512907)

[1.1 Setting up the development and testing environment 4](#_Toc9512908)

[1.1.1 Preparations 4](#_Toc9512909)

[1.1.1.1 The virtual environments 4](#_Toc9512910)

[1.1.2 Setup 5](#_Toc9512911)

[1.1.3 Finishing up 9](#_Toc9512912)

[1.2 Setting up the production environment 9](#_Toc9512913)

[1.2.1 Preparations 9](#_Toc9512914)

[1.2.1.1 Basic setup of the VM 10](#_Toc9512915)

[1.2.1.2 Nginx 10](#_Toc9512916)

[1.2.1.3 Create project structure 11](#_Toc9512917)

[1.2.1.4 Python 12](#_Toc9512918)

[1.2.1.5 The virtual environments 12](#_Toc9512919)

[1.2.1.6 MongoDB 13](#_Toc9512920)

[1.2.2 Setup 14](#_Toc9512921)

[1.2.2.1 Configuration files 14](#_Toc9512922)

[1.2.2.2 Installing services 14](#_Toc9512923)

[1.2.3 Finishing up 15](#_Toc9512924)

[2 Database workings 16](#_Toc9512925)

[2.1 Configuration 16](#_Toc9512926)

[2.2 Receiving data 16](#_Toc9512927)

[2.3 Expanding 16](#_Toc9512928)

[3 Front-end workings 17](#_Toc9512929)

[3.1 Configuration 17](#_Toc9512930)

[3.2 Reading data 17](#_Toc9512931)

[3.3 Serving webpages and data 17](#_Toc9512932)

[3.3.1 General 17](#_Toc9512933)

[3.3.2 Dashboard 18](#_Toc9512934)

[3.3.3 Detail 18](#_Toc9512935)

[3.3.4 History 18](#_Toc9512936)

[3.4 Expanding 18](#_Toc9512937)

[3.4.1 Database 18](#_Toc9512938)

[3.4.2 Webpages 19](#_Toc9512939)

[Sources 20](#_Toc9512940)

# Setting up the database & front-end components

The database and front-end components of this project will run on a virtual machine running Debian 9.8 as an operating system. For reconstructions of this project other Linux or even Windows distributions can be used, but this documentation will focus on a Debian production environment. The development and testing environment used in this part of the documentation is a Windows machine.

## Setting up the development and testing environment

Before deploying software to a production server it’s a good idea to make sure everything works as it should. To do that, setting up a development / testing environment on a local machine is a good idea. The development / testing environment will in this case run on a Windows machine and consist out of a MongoDB instance and a Django webserver.

All the examples in this documentation are based on the code, files and configurations for this project which are provided on GitHub at <https://github.com/Senglide/datacenter-monitoring>, so make sure this project folder is present on the local machine.

### Preparations

For this project Python 3.7 or higher and MongoDB Community Server 4.0 or higher are required, so start with downloading and installing these two components first. Suitable installation packages can be found on <https://www.python.org/downloads/> and <https://www.mongodb.com/download-center/community> respectively. Once installed the MongoDB server will run automatically, and since this is a development server no further configuration or security is required.

Before it’s possible to start developing code and testing it out there are some Python requirements that are still to be fulfilled. To prevent faulty code from affecting system resources and processes any Python code and modules are best run in a virtual environment. Create a virtual environment for the database component and one for the front-end component by opening a command prompt in the project folder and executing the following commands.



#### The virtual environments

The database component and the front-end component require different Python modules to work as intended. All these modules can be installed with pip in the appropriate virtual environment. Since this setup process is using the Jimma University network, which uses a proxy server to connect to the outside world, all pip commands will be suffixed with ‘--proxy http://10.140.5.19:8080’. To prevent any timeout errors because of eventual network issues the default timeout setting will also be increased to 120 seconds with ‘--timeout 120’. All the commands to install the required modules are prescribed below.

**Database virtual environment:**

**Front-end virtual environment:**



### Setup

Once the virtual environments and Python modules are installed some additional setup is required to run the project on the local machine. The configuration files from the GitHub project are set up to work in a production environment. To get everything running for the development / testing environment there are a few changes to be made.

**Database component:**

The local machine won’t be receiving any data from sensors in the data center, so to make development and testing easier a test script is provided which generates data for the front-end. On the local machine, change datacenter-monitoring/database/code/db\_writer.py to the code below.

Next, change datacenter-monitoring/database/test/test.py to the code below, and run it to start populating the database with test data. Make sure on line 8 to change the path to the code folder to the right path corresponding with the folder location on your machine before running the test script.

To run the test script, open a command prompt in the project folder, activate the virtual environment and execute it.



**Front-end component:**

To get the front-end component running on the local machine open datacenter-monitoring/frontend/frontend/settings.py, remove the import for configparser on line 14 and change the databases section starting on line 76 to the code below.

to run the webserver on the local machine open a command prompt in the project folder, activate the virtual environment, and run the local development server. If it gets run for the first time, don’t forget to make the necessary migrations for Django.



### Finishing up

Once everything on the local machine is running it is possible to start developing and making changes to the project to tailor it to your needs. The exact workings of the database and front-end components of this project will be described and explained later in the documentation, so read that chapter to understand how everything works and make efficient changes.

## Setting up the production environment

Once the project development is finalized it can be moved to a production environment after some minor tweaks to configurations and services. The environment has the same requirements as the development / testing environment but because it is moved from a Windows machine to a Debian machine there are some modifications to the setup process.

### Preparations

For this project Python 3.7 or higher and MongoDB Community Server 4.0 or higher are required. At the time of writing, the Debian package list does not include these versions of the software packages so manual installations are required. The production environment also needs almost the same Python virtual environments as the development / testing environment. The package list is a little bit different, but this will described when setting up the virtual environments.

#### Basic setup of the VM

Before the installation of all the components basic setup of the virtual machine is required. Make sure you are logged in as root, install sudo and add your user to the sudo group. Don’t forget to replace ‘mis’ with your own username.



Next, install a firewall for basic overall protection of the system and add exceptions for the services that will be running on the VM, like SSH and HTTP.



Also install Git, and configure it to work with the JU network proxy.



#### Nginx

This project uses Daphne, Uwsgi and Nginx as webservers to host the front-end component. Daphne will handle the asynchronous web sockets that provide the alarm functionality. Uwsgi will act as a translator between the Python based Django backend and the web based Nginx. Nginx is configured as a reverse proxy to negotiate the different type of web requests to the right underlying webserver. Daphne and Uwsgi will be installed as Python packages in the virtual environment section. To get Nginx up and running the following should be done:



#### Create project structure

To make the project feel like a native Linux program and make it easy for other people to find everything, it is best to move all the project files to appropriate places. Follow the instructions below to install everything in the right place.



#### Python

Since the Python version provided by Debian is 3.5 and the project requires version 3.7 or higher Python will have to be manually upgraded. To do this, first remove Python 3 completely from the system. Next, install all the dependencies that Python requires and lastly acquire the Python 3.7 source code from <https://www.python.org/downloads/source>, transfer it to the VM and install it.



#### The virtual environments

To set up the virtual environments on the VM, first install pip, then create the virtual environments and install all the required dependencies.



#### MongoDB

Just like the latest version of Python, the latest version of MongoDB is also not in the package list of Debian so it has to be installed manually. This is possible by adding the public key to MongoDB Community Server to the sources list and then updating it. Download the public key from <https://www.mongodb.org/static/pgp/server-4.0.asc> and make sure it’s present on the VM.



Once MongoDB is installed and running it should be configured with users and authentication. Make sure to replace all usernames and passwords with the correct usernames and passwords for your project.



### Setup

With all the software installed and all the files in place there are still some configurations to be set. For security reasons there are no login credentials or keys stored in any of the Python scripts. These are kept in configuration files which can be read by the Python scripts, but these configurations still need to be generated.

It is also a good idea to configure the project as a service which starts automatically at system startup. This has the advantage that if the VM goes down for some reason, then the project will start op automatically when the VM comes online again. Since we are using Nginx, which runs as an automatic service, the reverse proxy will already restart automatically. For MongoDB the automatic service is also already enabled. For the Python script that receives the data and the other webservers however this will still have to be configured.

#### Configuration files

To create the database configuration edit /opt/monitoring/database/config/create\_db\_config.py and fill in the database login credentials and connections settings for the project. Then run the script, and move the generated configuration file to /etc/monitoring



Do the same thing for the web configuration for which the creation file is under /var/www/frontend/config.



#### Installing services

To run the data receiving script and webservers as services, move /opt/monitoring/database/config/receive\_data.service, /var/www/frontend/config/run\_daphne.service and /var/www/frontend/config/run\_uwsgi.service to /etc/systemd/system/, activate them and start them.



### Finishing up

Before Django can communicate with the database, the Django project models need to be migrated to the database. Luckily, Django provides a helpful script for this.



Django will be looking for static files of the web project in /var/www/frontend/static, but this location does not exist yet. Go to /var/www/frontend, collect the static files which will be moved to /home/mis/static because the collection script does not run with administrator privileges, and move this folder to the right location.



To finish up, reboot the system and make sure every service is running.



# Database workings

As described before the database for this project is a MongoDB instance. In the setup from part one of this documentation there are three users configured who have access to this database: an administrative user, a user who can only read and a user who can read and write. This part of the documentation will describe how the database functions and is accessed.

## Configuration

The configuration files for the database part of the project can be found in the project structure under /database/config/. The create\_db\_config.py script takes all the login credentials for the database for one user and uses configparser to turn them into a useable configuration file for the script that writes to the database. The receive\_data.service file is a service configuration file that can be used to run the receiving process as a service on the virtual machine.

## Receiving data

The files that handle receiving data from the sensors and writing it to the database can be found in the project structure under database/code/. The central script in this process is receive\_data.py. This script acts as an MQTT client that connects to the MQTT broker that provides the data coming from the sensors. This connection is secured with TLS encryption. More information about the connection and the security can be found in the IOT documentation.

The receive\_data.py script connects to the MQTT broker and will try to keep the connection open forever. If the broker goes offline, the client will keep listening until the broker comes online again and then reconnect. If the script receives a message, the message will first be converted to a JSON compatible string in the ‘on\_message’ function. Then all sensor readings in the message will be formatted in the ‘process\_reading’ function. This function uses the reading\_class.py script as a template to construct a reading. Depending on the data contained in the reading it will define the collection the reading has to go to and send it off to the db\_writer.py script.

The db\_writer.py script uses a Python package called Pymongo to connect and write to the database. The login credentials for this process are provided by the dbConfig.ini file which can be generated with the create\_db\_config.py script from the config folder.

Another function the ‘process\_reading’ function has is to keep track of status of the alarm values for the smoke and movement detectors. It uses the first values it receives to update the system to the current status of the alarm signals, and after that it checks for changes in the signal. This initial setting of the values and the updating when changes occur is done by simply sending a http post request to the front-end. To send these requests the function uses the Python package called Requests.

## Expanding

To expand the database part of the this project the only thing that needs to happen is to add more sensors, and make sure they send the trough the MQTT broker in a correct way as described in the IOT documentation. The nature of MongoDB allows it to dynamically generate new collections, so the scripts will take care of that without any adjustment.

# Front-end workings

The front-end part of this project is a Django web application visualizing the data generated and stored by the system. It requires regular web hosting for most of the files. Uwsgi is used as a translator between the Python based Django project and most of the web requests. Some of the web requests generated by the project are long polling http requests which are basically open web sockets. These cannot be handled by Uwsgi. For these requests the project uses Daphne. To make out which requests go to which handler, Nginx is used. It acts as a reverse proxy that supervises the web requests coming in. The front-end is hosted in the JU local network and can be reached via the following link: http://10.140.5.36/dashboard.

## Configuration

The configuration files for the database part of the project can be found in the project structure under /frontend/config/. The create\_web\_config.py script takes all the login credentials for the database for one user and uses configparser to turn them into a useable configuration file for the script that reads data from the database. The run\_daphne.service and run\_uwsgi.service files are service configuration files that can be used to run the web servers as a service on the virtual machine.

## Reading data

The configuration of Django to work with MongoDB happens in /frontend/frontend/settings.py. This file uses a Python package Djongo, which in turn uses Pymongo, to create a direct connection between Django and the database. The login credentials are read from the configuration file created by the create\_web\_config.py script. Once the connection is established, it will try to get data based on the models described in /frontend/dashboard/models.py.

## Serving webpages and data

The available webpages and web requests of the project are defined in /frontend/dashboard/views.py. To links to reach these resources are defined in /frontend/dashboard/urls.py. Django translates incoming requests between these two to serve the correct data.

### General

There are some general JavaScript files that get used through the entire project which can be found in /frontend/dashboard/static/dashboard/js/general/. In this folder settings.js is located which is a central configuration file for the web project. It contains general settings like which server racks are active and which types of sensors are used but also some default settings and options for the web pages Django serves.

Another file found here is graph.js. This is a template file that can generate graphs if you provide it with an available html div and data. The file will create a graph based on the ‘sensor\_value’ and ‘datetime’ attributes of the objects in the data and will draw it in the provided div. To use it in practice create a graph instance providing the id of the div where to draw it, a string containing the racks it has to draw data lines for, and a sensor type. After, set the data attribute, which should be an object containing arrays of readings, and call the ‘createGraph’ function. This function will create a graph and draw a data line for each array provided. If you want to update the graph. Reset the data attribute and call ‘updateGraph’.

Graph.js in turn is used by gridcell.js. When supplied with a graph gridcell.js can deliver a html string to draw a section containing the graph and the necessary controls.

The styling for the entire web project is handled by css. All the css files can be found under /frontend/dashboard/static/dashboard/styles/.

### Dashboard

The dashboard html file can be found in the project structure under /frontend/dashboard/templates/dashboard/index.html and it uses the general JavaScript files together with its own JavaScript files found in /frontend/dashboard/static/dashboard/js/index/.

The dashboard contains a list of grid cells, which it is constructed from. It uses a timer to update all the graphs. The settings for how the graphs should be updated are stored in the refreshSettings and connectionBlocks variables. These settings can be changed by browsing to the dashboard and clicking the cogs icon in the dashboard area.

The dashboard also keeps track of the current alarms statuses for the movement and smoke detectors. When loaded, it will poll the server for the current status with the ‘getCurrentAlarm’ function and set it in the dashboard. From there, a web socket which acts as a long polling http request will be opened, which will listen for alarm changes from the server.

### Detail

The dashboard html file can be found in the project structure under /frontend/dashboard/templates/dashboard/detail.html and it uses the general JavaScript files together with its own JavaScript files found in /frontend/dashboard/static/dashboard/js/detail/.

The web page is mostly constructed by functions in detail.js because it relies on incoming data. It will draw a graph and data table based on a date, rack and sensor type. How the data is displayed can be changed by accessing the settings through the cogs icon.

### History

The dashboard html file can be found in the project structure under /frontend/dashboard/templates/dashboard/history.html and it uses the general JavaScript files together with its own JavaScript files found in /frontend/dashboard/static/dashboard/js/history/.

The history page of this project does as its name implies: it shows a history of what has been going on in the data center. You can define be history by a time range or by how far you want to go back in the future. The mechanics behind this webpage are based on the comparison of datetime timestamps. The data are presented by grid cells who in turn contain a graph.

## Expanding

Expanding the web part of this project is pretty straightforward. The only thing that should not be forgotten is that whenever something changes to update the files in /var/www/frontend and to run ‘collectstatic’ and update the static files on the virtual machine in /var/www/frontend/static/. More information on the ‘collectstatic’ command can be found in section 1.2.3 of this documentation.

### Database

If anything is to be changed to which data is stored and how it is stored this should be done in /frontend/dashboard/models.py. Right now this file is configured for 10 racks and smoke and movement data. If any new racks are added to the project they should also be added in models.py. If any new data fields are added to sensor values this should also be added in models.py. To finalize the changes they have to be migrated to the database. This can be done by activating the front-end virtual environment and running ‘manage.py makemigrations’ and ‘manage.py migrate’. The script runs interactively to notify and fix any migration problems.

### Webpages

To add a web page, in the back-end the only thing that needs to happen is add a view in /frontend/dashboard/views.py to handle the incoming requests and add a link in /frontend/dashboard/urls.py to link the web address to the right view. The view has to link back to a template file in /frontend/dashboard/templates/dashboard/ which in turn can use css and JavaScript to become a functional web page.

# Sources

D3 JS. (2018). *D3 Wiki*. Opgehaald van Github: https://github.com/d3/d3/wiki

Django Software Foundation. (2018). *Django Channels Docs*. Opgehaald van Read The Docs: https://channels.readthedocs.io/en/latest/index.html

Django Software Foundation. (2019). *Django Documentation*. Opgehaald van Djangoproject: https://docs.djangoproject.com/en/2.2/

Fanout. (2019). *Django-eventstream*. Opgehaald van Github: https://github.com/fanout/django-eventstream

MongoDB, Inc. (2017). *PyMongo 3.5.1 Documentation*. Opgehaald van MongoDB: https://api.mongodb.com/python/3.5.1/

MongoDB, Inc. (2019). *The MongoDB 4.0 Manual*. Opgehaald van MongoDB: https://docs.mongodb.com/manual/

Nesdis. (2019). *Djongo*. Opgehaald van Github: https://github.com/nesdis/djongo

The uWSGI project. (2016). *uWSGI Docs*. Opgehaald van Read The Docs: https://uwsgi-docs.readthedocs.io/en/latest/index.html