CITY IoT

Vipuvoimaa EU:lta 2014–2020

European Union
European Regional Development Fund

Report on FIWARE Platform

Ville Heikkilä, Otto Hylli, Mikko Nurminen, and Kari Systä

June 2, 2020

6Aika · Uudenmaan liitto Nylands förbund · OULU · TAMPERE · OULUN YLIOPISTO · OAMK · Tampereen yliopisto Tampere University

Contents

1 Introduction **3**

2 Background **3**
2.1 FIWARE introduction 4
2.2 Evolution of the CityIoT FIWARE platform 5

3 Platform **6**
3.1 Platform architecture 6
3.2 Platform components 8
3.2.1 Nginx proxy server 8
3.2.2 Mongo database 9
3.2.3 Orion Context Broker 10
3.2.4 QuantumLeap 11
3.2.5 Grafana 11
3.2.6 Wirecloud 12
3.2.7 CKAN 13
3.2.8 Utilities 13
3.2.9 IoT agents 14
3.3 Access control system 14

4 Platform configuration and maintenance **15**
4.1 System requirements 16
4.2 Platform configuration 16
4.3 Platform maintenance 17
4.3.1 Data backup and restore 18
4.3.2 Updating access control permissions 18
4.3.3 SSL certificates 18
4.3.4 Memory issue with Orion 19

5 Examples of data usage in FIWARE platform **21**
5.1 Data sources and data gathering 21
5.1.1 Tampere streetlight data 22
5.1.2 Viinikka streetlight data 23
5.1.3 Electric buses 23
5.1.4 Passenger data 24
5.2 Data visualization 24
5.3 Streetlight demo 27

6 Summary **31**

A Technical details related to Grafana visualizations **32**

B Technical details related to streetlight demo **33**

1 Introduction

The goal of the CityIoT[1] project is to define a vendor independent IoT platform for SmartCity applications. The platform should support multiple data sources and several applications using the data sources. The platform should also be scalable to various usage needs, provide access control, be convenient to deploy and offer possibilities for configuration and customization to different needs.

In the beginning of the project we analyzed technical options and selected FIWARE[2] as the technical framework, because it has similar goals towards vendor independence and has already gained interest in the SmartCity community. FIWARE offers open source components for building a platform for smart applications.

This report presents the CityIoT solution for building a smart city platform with FIWARE and other open source components. The platform uses Docker containers for deployment which offers an easy way to manage the platform and for example to scale components as needed. FIWARE components are used to store data. Other components are used to provide a data registry and data visualizations. The popular Nginx web server is used as an entry point for the platform. It can offer secure HTTPS communication and access control. Alternatively access control can be also provided with FIWARE components.

This report is intended for people who are planning to deploy a FIWARE based platform. It gives them information for evaluating the CityIoT platform as an possible solution. For people who have decided to use the CityIoT platform it gives information about the platform architecture, configuration possibilities and help in deploying the platform. This report does not go into every technical detail for deploying and configuring the platform. That information is available from the platform's GitHub repository[3].

The rest of the document has been organized as follows. The first section 2 provides some required background information by giving a short overview of the FIWARE technology and history of the CityIoT platform. The next section 3 describes the CityIoT platform more in-depth. It describes the Docker based deployment system and describes each platform component. Then section 4 talks more about how the platform can be configured and maintained. Section 5 illustrates how the platform can be used by describing how the CityIoT project utilized it with data from real world smart city pilot cases. Finally section 6 presents some summarizing remarks.

2 Background

This section provides an overview of the FIWARE technology and explains the historical evolution of the CityIoT FIWARE platform.

[1] https://www.cityiot.fi/english
[2] https://www.fiware.org
[3] https://github.com/cityiot/cityiot-platform

2.1 FIWARE introduction

The core of the FIWARE project is the NGSI v2 specification[4]. It defines a context entity based data model that all FIWARE components understand. These context entities can represent various physical or logical objects such as devices, vehicles, weather observations, or buildings. What the entity represents is indicated by its type such as *Vehicle* or *WeatherObserved*. The type then determines what attributes the entity can have for representing its state. Vehicle for example has speed and location, and weather observation temperature and humidity. A NGSI v2 compatible data model can then be specified for a certain domain by defining the entity types and their attributes. The FIWARE community has already defined data models for various domains[5] such as weather, parking, and street lighting.

In addition to the data model, the NGSI v2 specification also defines an API for a context broker which can manage these entities. This includes creating, updating, deleting, and querying entities. The API also has a subscription feature which allows clients to get notifications about changes to entities.

Around this core idea of context entity management the FIWARE project then has developed various components also called general enablers (GE) which are listed in the FIWARE catalogue[6]. The CityIoT platform uses 8 of these. The 3 FIWARE access control components (Keyrock, Wilma, and AuthZForce) and their use are not covered in this document. More information about setting up a FIWARE platform using these components for access control can be found in CityIoT reference platform access control document[7]. All other listed components are covered in more detail in section 3.2 but they are all introduced shortly here by listing them under their FIWARE catalogue category:

- Core Context Management
  - **Orion context broker**: The core component of any FIWARE platform. It implements the NGSI v2 API and manages the current state of the entities.
  - **QuantumLeap**: Can store the entity history into a time series database.
- Interface with IoT, Robots and third-party systems
  - **IoT Agent for Ultralight**: A bridge between the NGSI v2 API and data model, and entity data sources that use the light weight ultralight message payload to communicate via HTTP, AMQP or MQTT transport protocols.
- Context Processing, Analysis and Visualization

[4] https://fiware.github.io/specifications/ngsiv2/stable/
[5] https://github.com/smart-data-models/data-models
[6] https://www.fiware.org/developers/catalogue/
[7] https://drive.google.com/file/d/1tmzA3I45ZV35X549piClEWw2SzV5uDdl

  - **Wirecloud**: Offers a web mashup platform for creating dashboards for end users.

- Context Data/API Management, Publication and Monetization
  - **Keyrock**: Identity management component allows management of users and their roles. Supports OAuth 2 authentication.
  - **AuthZForce**: Allows the creation of complex access rules.
  - **Wilma**: Controls access to components based on users and roles managed with Keyrock and rules defined with AuthZForce.
  - **CKAN extensions**: Enables the integration of the CKAN open data publication platform to a FIWARE based platform.

2.2 Evolution of the CityIoT FIWARE platform

In the CityIoT project the first FIWARE platform was setup at the University of Oulu to allow the CityIoT pilot programs to store the gathered data in the platform. This first FIWARE platform consisted of Orion Context Broker, STH-Comet and Mongo database. Due to some performance issues related to how STH-Comet stores the historical data, 4 separate STH-comet instances was included in the platform (3 for writing data and 1 for reading data). The platform also included Kong[8] to provide simple access control (any access requires a secret token with the request, however, there is no other limitations and for example different users can access each others data without restrictions) and Nginx to provide load balancing to the platform.

At the Tampere University, the first tests with a FIWARE platform were done with a similar setup as the platform at the University of Oulu, i.e. using STH-Comet as the component to store the history data. Since there were issues with the performance when writing new data when using STH-Comet, it was decided to test QuantumLeap, a different history component, to see if it would work better. At the time QuantumLeap was still heavily in development (version 0.4 had been released) and only part of the API had been implemented but there was no similar performance issues as when using STH-Comet. The QuantumLeap API also was more flexible than the one provided by STH-Comet, so it was decided to concentrate on using QuantumLeap instead of STH-Comet in the future FIWARE platforms.

These first test versions of the FIWARE platform contained Orion, Quantumleap, and their required databases, Mongo and Crate respectively. The visualization component Grafana was also included since it is quite easy to connect to the Crate database and can be used to make quick test visualizations for the stored data. There first FIWARE platforms deployed at the Tampere University did not have any access control mechanism integrated to them. A FIWARE platform similar to the one used in the Tampere University tests can

[8] https://konghq.com/kong/?itm_source=website&itm_medium=nav

be deployed using the Docker Compose file from the QuantumLeap code repository[9].

The final version of the CityIoT FIWARE platform was developed based on the previous experiences. The core components were still Orion and QuantumLeap but a user based access control mechanism using Nginx was also included to the platform. Also, to provide better performance and scalability multiple core components can be running at the same time. This final version is described in more detail in the next chapter 3.1.

3 Platform

This chapter describes the final version of CityIoT FIWARE platform. Section 3.1 contains description of the overall architecture of the platform including general description how it uses Docker for deployment. More detailed component specific descriptions are given in section 3.2 and section 3.3 contains information on how the access control system works on the platform.

The CityIoT FIWARE platform deployed at the Tampere University can be found at https://tlt-cityiot.rd.tuni.fi.

3.1 Platform architecture

The CityIoT FIWARE platform is deployed in a Docker swarm. A Docker swarm allows the use of Docker services as well as the use of multiple nodes. A node in Docker swarm can be either a physical machine or a virtual machine. The default configuration for the CityIoT FIWARE platform uses a single-node swarm but expanding to a distributed system with multiple nodes should not require extensive amount of extra configuration work. All the existing configuration has been done in a way that is compatible with a multi-node swarm.

A Docker service can contain either just one or multiple Docker instances of the same Docker image with identical configurations. The number of containers in a service can be easily scaled either by giving the number of wanted replicas directly (e.g. 5 containers distributed among the nodes) or connecting the number of replicas to the number of nodes in the Docker swarm (e.g. 3 containers in a 3-node swarm and 1 container in a single-node swarm). The use of Docker services gives an easy way to adjust the availability and performance of the CityIoT FIWARE platform.

A single common Docker network is used with all the Docker services in the platform to allow the communication between the different components. All services in the same network have access to the other services using the name and the port used by the other service. This Docker network is a private network that can only be accessed through the Nginx proxy server from outside. A more complicated approach with multiple different Docker networks (e.g. different networks for backend and frontend components) could have been used to provide more security by limiting the access between the platform components. The

[9] https://github.com/smartsdk/ngsi-timeseries-api/blob/master/docker

one network approach was used in the CityIoT FIWARE platform because of its simplicity and since any outside access to the network is limited by the use of the Nginx proxy server.

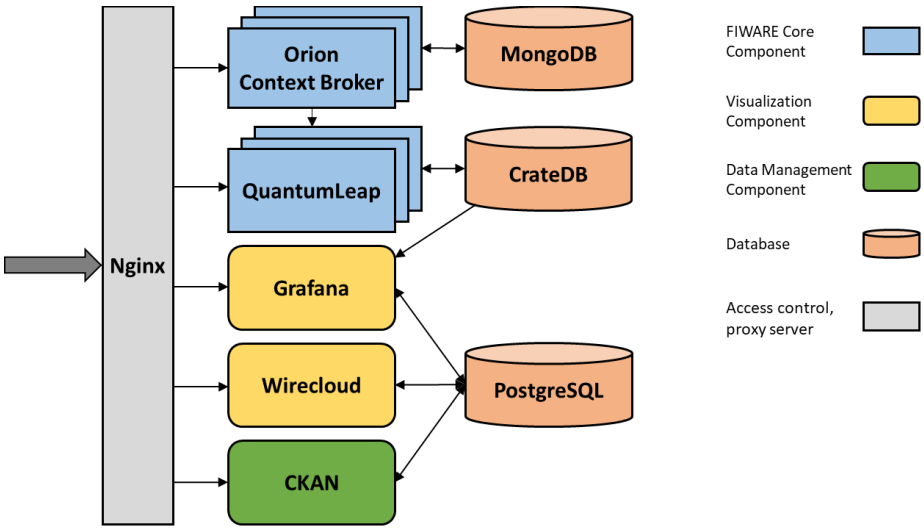

Figure 1: The FIWARE platform architecture used at Tampere University.

Figure 1 shows the overall architecture of the CityIoT FIWARE platform. All outside traffic goes through the Nginx proxy server that provides the Transport Layer Security (TLS) to the connections and forwards the queries to the proper components. Basic access security at the user and FIWARE service level for the FIWARE components Orion and QuantumLeap is provided by Nginx (more information about this access control can be found in the Nginx component description 3.2.1). Also, the Nginx server provides a cache to make repeated queries more efficient. The FIWARE components Orion and QuantumLeap are deployed with multiple replicas to provide better availability and performance. The load balancing between the different replicas of Orion and QuantumLeap is done internally by the Docker swarm manager.

The figure does not include the IoT Agent for Ultralight component but the code repository for the platform[10] does include an option to include it to the platform. If the Iot Agent is included, it would be deployed similarly to the Orion Context Broker and QuantumLeap using multiple replicas and access control provided by Nginx.

Open source visualization tools Grafana and Wirecloud as well as open source data portal tool CKAN are included in the platform. They provide tools for the users of the platform to work with the data stored in the FIWARE platform. The main databases used by the services are shown in the figure. Orion uses NoSQL database, MongoDB, to store the current context data and QuantumLeap uses SQL database, CrateDB, to store the context history. These

[10] https://github.com/cityiot/CityIoT-platform

databases have ready-made support for deployment in a multi-node platform. For the storage of user data used by Grafana, Wirecloud and CKAN, an instance of PostgreSQL server is deployed on the platform. The server contains separate PostgreSQL databases for each application. There are also some other components deployed on the platform that are not included in the diagram but that are required by the provided services, e.g. Redis service for CKAN. These other components are all included in the more detailed component descriptions in the following section 3.2.

3.2 Platform components

This section describes in more detail each of the platform components. It is divided into subsections that correspond to the Docker application stacks (the terms used in Docker documentation are either stack or application stack) that are collections of Docker services.

In the CityIoT FIWARE platform the division of services to different application stacks is mainly done to emphasize the connection some of the services have with each other. The division could have done in many different ways. During deployment (or after the system is already running) new services can be added to an existing stack or to a new stack. There is a limitation with adding services to stacks that means that all services defined in a single Compose file[11] must go to the same stack and they are also all deployed simultaneously. So, for application stacks that have optional services, the services definitions must be separated to different files.

Only the Nginx, Mongo, and Orion services are mandatory for the CityIoT FIWARE platform. The context broker and its database are required for any FIWARE platform and the Nginx is required for the CityIoT FIWARE platform in order to have the option to provide access control to the context broker. All other services are optional including the data history component, QuantumLeap. The configuration section 4.2 contains more detailed information about the options and how to select the components to be included when deploying a new platform instance.

3.2.1 Nginx proxy server

The Nginx proxy server application stack contains one Nginx proxy (version 1.15.8) service that handles all incoming traffic to the CityIoT FIWARE platform by forwarding the HTTP requests to the correct platform services. Table 1 lists the services the Nginx server can forward the queries to and the URL for each service when `<host>` is replaced by the actual host server name.

The Nginx server provides Transport Layer Security (TSL) to the connections using a Let's Encrypt certificate[12]. While Let's Encrypt was used in the CityIoT project, any other certificate provider can be used with the platform as well. The Nginx server also provides a cache to make repeated queries more

[11] https://docs.docker.com/compose/compose-file/
[12] https://letsencrypt.org/how-it-works/

Table 1: Services available through the Nginx server.

Service	URL to the service	
Orion Context Broker	`https://<host>/orion/`	3.2.3
QuantumLeap	`https://<host>/quantumleap/`	3.2.4
Grafana	`https://grafana.<host>` or `https://<host>/grafana/`	3.2.5
Wirecloud	`https://wirecloud.<host>` or `https://<host>/wirecloud/`	3.2.6
CKAN	`https://ckan.<host>` or `https://<host>/ckan/`	3.2.7

efficient. This means that a HTTP GET query with the same parameters (the same address, the same query parameters, and the same relevant header values) that is repeated within one minute will be immediately returned by Nginx (with the same response as in the previous occurrence) without forwarding it to a platform service. The Nginx service could be easily replicated to provide higher availability but on the Tampere University FIWARE platform just one Nginx service has worked well.

For Orion and QuantumLeap Nginx provides user and service based access control to the APIs. User and FIWARE service tokens are stored in Nginx configuration and they are used to determine access rights (either no access, read-only access, or both read and write access). To get access user has to provide the appropriate token in the HTTP header. The other services use their own access control systems. Access control section (3.3) provides more details about the used access control mechanisms.

The platform components that have an API, Orion and QuantumLeap, are accessed through service specific URL paths in the HTTP address given in table 1. The other platform components are served under subdomains, although service specific URL path redirection is done for user convenience.

The use of subdomains is done to allow the applications to work in a root path. Deploying applications with a web interface like Grafana, Wirecloud, or CKAN in a non-root path can be problematic unless the application has built-in support for non-root paths. With Grafana a non-root path is not a problem because of this built-in support but Wirecloud and CKAN do not have a proper support for non-root paths. For example, static assets might not be found because of a wrong URL generated by the application or a login button can direct to the wrong page.

3.2.2 Mongo database

The Mongo NoSQL database[13] (version 3.6.16) is used as the storage database by Orion Context Broker and it is mandatory in all FIWARE platforms. It is

[13] https://www.mongodb.com/

set up as a separate application stack to offer support for multi-node platforms. Other reason to separate it from the Orion application stack is the fact that other components like possible IoT-agents (3.2.9) can also use the Mongo database.

The Mongo application stack contains one Mongo controller service and one Mongo database for each node in the platform. For one-node system as in the CityIoT FIWARE platform there will be only one Mongo database that stores all the data. In a multi-node setup with n nodes, there will be n Mongo databases which are all replicas of each other. The Mongo controller service contains the logic to manage the consistency of this Mongo database replica set. The database cannot be accessed directly from outside. The direct access is restricted to the other components in the CityIoT FIWARE platform.

3.2.3 Orion Context Broker

The Orion Context Broker[14] (by default version 2.3.0) is the main FIWARE component and it is mandatory in all FIWARE platforms. It uses the Mongo database 3.2.2 and it is set up as a separate application stack to give the user freedom to choose which other components to include in their FIWARE platform and to keep more logical separation between the stacks. It is also separated from the Mongo stack, since other components like possible IoT-agents (3.2.9) can also use the Mongo database.

Orion Context Broker keeps a record of the current context i.e. the stored entities and the latest attribute values. It provides the FIWARE NGSIv2 API [15] which is a RESTful API that enables the user to perform updates, queries, or subscribe to changes on the context information. Orion also implements an additional way to logically separate the context data to FIWARE services[16] (also called tenants in some documentation). To store the evolution of the context information an extra component is needed for the platform. In CityIoT FIWARE platform this context history is handled by QuantumLeap 3.2.4.

Orion does not by itself provide any access control methods. In the CityIoT FIWARE platform the access control to the Orion is provided by Nginx using user and FIWARE service based tokens. This access control system is explained in more detail at section 3.3.

The Orion application stack contains several replicas (the default number of replicas is 5) of the Orion Context Broker to provide better availability and performance. The number of Orion services can easily be scaled up or down. More information on how to scale the services can be found at the platform repository instructions[17].

[14] https://fiware-orion.readthedocs.io/en/master/index.html
[15] http://fiware.github.io/specifications/ngsiv2/stable/
[16] https://fiware-orion.readthedocs.io/en/master/user/multitenancy
[17] https://github.com/cityiot/CityIoT-platform#scaling-the-components

3.2.4 QuantumLeap

The QuantumLeap application stack contains the component and database that can be used to store the FIWARE NGSIv2 data as time series for later use. The stack contains several replicas (the default number of replicas is 3) of QuantumLeap[18] (by default version 0.7.5) service and one Crate database[19] (version 3.3.5) for each node in the FIWARE platform. The replication of QuantumLeap service is done to provide better availability and performance. The number of QuantumLeap services can easily be scaled up or down. QuantumLeap has an optional geocoding feature, i.e. determining location coordinates based on street addresses, that requires a Redis service which is included in the utilities application stack.

QuantumLeap provides an API[20] to access the stored data. The API is similar to the FIWARE NGSIv2 API used by Orion. This API offers operations to read either entity or entity type specific data. Either raw data or aggregated data (for numerical values) can be queried (e.g. average value per hour). In addition the API provides way to send data to QuantumLeap. The normal way to store FIWARE NGSIv2 data with QuantumLeap is to make a subscription to Orion context broker where the `/notify` endpoint of the QuantumLeap API is given as the target address for any notification created based on the subscription parameters. It is also possible to manually send data to QuantumLeap using this same `/notify` endpoint. More discussions on how to manage the FIWARE data gathering can be found on the Collecting data to FIWARE document[21].

For single-node system as in the CityIoT platform there will be only one Crate database that stores all the data. In a multi-node setup with n nodes, there will be n Crate databases which are all replicas of each other. The database cannot be accessed directly from outside. The direct access is restricted to the other components in the CityIoT FIWARE platform. It should be noted that all user management utilities are only available in the enterprise version of the Crate database. Support for Timescale database for which user management does not require a paid-version is included in QuantumLeap but at the time of writing only Crate database works with the provided API.

QuantumLeap does not by itself provide any access control methods. In the CityIoT FIWARE platform the access control to the QuantumLeap is provided by Nginx using user and FIWARE service based tokens. This access control system is explained in more detail at section 3.3.

3.2.5 Grafana

Grafana[22] is an open source software for time series analytics and visualization that works in a web browser. Grafana uses plugins to support different databases and various visualization objects (e.g. tables, graphs, maps) that can be used

[18] https://quantumleap.readthedocs.io/en/latest/
[19] https://crate.io/products/cratedb/
[20] https://app.swaggerhub.com/apis/smartsdk/ngsi-tsdb/0.7
[21] https://drive.google.com/file/d/16k77MirUt_AM16j5jN5y4X6gVY43kFYK
[22] https://grafana.com/grafana/

in the user created dashboards. It has integrated support for querying data directly from the Crate database that the history component QuantumLeap uses. With it a user can create, for example, dashboards using the history data stored in the CityIoT FIWARE platform. Grafana also has its own user and access control system. While Grafana works well with QuantumLeap's Crate database, there is no direct support for getting data from Orion's Mongo database to Grafana. Also, there does not exist an easy way to provide FIWARE NGSIv2 data from Orion API to a Grafana dashboard. There exist a couple of third party MongoDB plugins for Grafana but they have not been tested during the CityIoT project.

The Grafana application stack contains one Grafana (version 6.5.3) dashboard service and it is optional in the CityIoT FIWARE platform. Grafana is put to its own application stack to emphasize that it is a separate application from the other main platform components. When using the Grafana service, the utilities application stack is mandatory because Grafana uses the PostgreSQL database to store the user information and the definitions for the created dashboards.

3.2.6 Wirecloud

Wirecloud[23] is a FIWARE Generic Enabler that provides a web mashup platform that can be used to develop operational dashboards. Wirecloud uses widgets written in JavaScript that users can add to their dashboards. These widgets can be connected together in flexible ways. For example there could be a general purpose graph widget to visualize data and another widget to receive data from a data source. If the received data is not in a format the graph widget understands, an adapter could be made between the data source and graph widgets. Any FIWARE stored data that is visualized in Wirecloud dashboards is queried through the Orion or QuantumLeap API. Wirecloud does not use a direct database connection like Grafana does with QuantumLeap's database. This means that any limitations in the API (like the limited number of results for one query) has to be taken into account when working with Wirecloud, unlike with Grafana and its direct database access. Wirecloud also has its own user and access control system.

The Wirecloud application stack is optional and it contains one Wirecloud (version 1.3) dashboard service. Wirecloud is put to its own application stack to emphasize that it is a separate application from the other main platform components. When using the Wirecloud service, the utilities application stack is mandatory because Wirecloud uses the PostgreSQL database to store the user information and the definitions for the created dashboards. Wirecloud can also make use of the Elasticsearch and memcached services provided in the utilities application stack.

[23] https://wirecloud.readthedocs.io/en/stable/

3.2.7 CKAN

CKAN[24] is an open source data management system for a web browser that can be used to publish and share data. The FIWARE CKAN extensions[25] provide support for publication of data sets matching the FIWARE NGSIv2 format. With FIWARE CKAN extensions comes also a preview functionality that can be used to preview data from the Orion Context Broker. Similarly to Wirecloud any queries made for the FIWARE data for the preview functionality are done using the FIWARE NGSIv2 API provided by Orion and there is no direct database connections made. CKAN also has its own user management system for access control.

The CKAN application stack is optional and it contains CKAN (version 2.8) with FIWARE extensions service. The stack contains also a specifically for CKAN configured version of Apache Solr[26] service as well as a CKAN datapusher[27] service that are both used by CKAN. In addition when using the CKAN service, the utilities application stack is mandatory because CKAN uses the PostgreSQL database for storing the user information as well as the Redis service provided by the utilities application stack.

CKAN provides also an API that can be used to create data sets without using the browser interface. Using this API, the CityIoT project created a script tool that can create data sets to CKAN where the data found in the Orion Context Broker is divided into sets based on the FIWARE service and service path. Using this tool also required direct access to the Mongo database used by Orion since the NGSIv2 API does not provide the service or service path information. However, using the tool, previews for any data stored in the CityIoT FIWARE platform can be shared in a more user friendly way without a lot of manual work. The tool is available at the Github repository[28].

3.2.8 Utilities

The utilities application stack contains databases and utilities that can be used by other platform components. The services in this stack are all using the official Docker images available at Docker Hub[29]. Utility services that have been modified to work specifically with some other service are not included in this stack. Instead, they are part of the application stack containing that other service. Example of this is the CKAN version of Apache Solr that is included in the CKAN application stack. This application stack is only mandatory if at least one of the services is needed for the use of some other service in the deployed FIWARE platform.

List of services included in the utilities application stack:

[24] https://ckan.org/
[25] https://fiware-ckan-extensions.rtfd.io/
[26] https://lucene.apache.org/solr/
[27] https://docs.ckan.org/projects/datapusher/en/latest/
[28] https://github.com/cityiot/orion_to_ckan
[29] https://hub.docker.com/

- PostgreSQL database[30] (version 9.6.16) for storing the user data for Grafana, Wirecloud, and CKAN.
- Redis[31] service (version 5.0.7) for the use of QuantumLeap and CKAN. By QuantumLeap this is only needed for optional feature of geocoding, i.e. converting street addresses to geographical coordinates in the cases when only street addresses and no coordinates are given.
- Elasticsearch[32] service (version 2.4) for the use of Wirecloud.
- Memcached[33] service (version 1.5.12) for the use of Wirecloud.

3.2.9 IoT agents

The FIWARE catalogue mentioned in the introductory section 2.1 contains several IoT agents that can be used to make it simpler for the user to provide data to the Orion Context Broker. The architecture diagram 1 does not show any IoT agents but the CityIoT FIWARE platform has a ready-made support for the FIWARE IoT-agent for Ultralight 2.0[34] as an optional component of the platform. The agent is designed to be a bridge between Ultralight and the NGSI interface of a context broker and supports AMQP, HTTP and MQTT transport protocols.

If used, the Ultralight 2.0 IoT Agent is deployed similarly to the Orion Context Broker with 2 replicated instances by default. The access control for the configuration API provided by the agent is also provided similarly to Orion by using the user and FIWARE service based access control system maintained by Nginx. The agent uses the same Mongo database as Orion to store the user provided configurations.

3.3 Access control system

There are three different kinds of access control systems in use in the CityIoT FIWARE platform. Only the first two (Nginx and web browser interface) are considered in this document. The third system using the FIWARE access control components is described in CityIoT reference platform access control document[35] which has also more detailed descriptions about the implementation of the Nginx based access control system.

1. **Access control provided by Nginx for Orion, QuantumLeap, and IoT agents.**
   The Nginx configuration provides user and FIWARE service based access control for Orion API, QuantumLeap API, or the Iot agent API.

[30] https://www.postgresql.org/docs/9.6/index.html
[31] https://redis.io/
[32] https://www.elastic.co/elasticsearch/
[33] https://memcached.org/
[34] https://fiware-iotagent-ul.readthedocs.io/en/latest/
[35] https://drive.google.com/file/d/1tmzA3I45ZV35X549piClEWw2SzV5uDdl

Each user is specified which FIWARE services they can access. The access types are given for each FIWARE service separately and they can be either read-only, i.e. only GET operations are allowed, or full access, i.e. no restrictions on the HTTP operations. Each user is given their own token which they must use when accessing the platform.
When updating the access rights, the Nginx configuration files need to be manually edited and the Nginx proxy server restarted for the edits to go into an effect.

2. **User and access control managed by a web browser interface.**
Grafana, Wirecloud, CKAN all provide their own user and access control management systems. For each application the main page of the application is freely available to all users without user account. An admin account (with account username and password that can be chosen by the deployer of the platform) is created for each application during the platform deployment. Using these admin accounts it is possible to manage all needed access rights through the browser interface of each application.

3. **FIWARE access control provided components Wilma, Keyrock, and AuthZForce.**
This is an alternative to the access control system provided by Nginx and it is based on the use of OAuth 2.0 tokens. It is described in detail in CityIoT reference platform access control document.

Additionally there is a special `/notify` endpoint provided by the Nginx. This endpoint can be used to allow the use of a separate token when receiving NGSI data from an external source. The Nginx forwards any queries made to this endpoint to the Orion Context Broker as a notification message about new data. In the external source the token might be stored in such a way that it might be visible to other users and thus could allow other users unwanted access to the FIWARE platform. With this endpoint and using a separate token in the external source, any unwanted access attempts of this kind are limited to sending notification messages. While this is not a "perfect" solution, it is still better than allowing unrestricted access with tokens stored on external sources. More information on how to use this endpoint can be found at the repository documentation[36].

4 Platform configuration and maintenance

This chapter contains the general information about the system requirements 4.1 and on what options there are when deploying a new CityIoT FIWARE platform instance 4.2. Also some general information related to the maintenance of a deployed FIWARE platform are included in the last section 4.3 of this chapter.

[36] https://github.com/cityiot/CityIoT-platform#setting-the-nginx-access-control-permission-step-4

For more detailed instructions on how to actually deploy the platform see the GitHub repository[37].

4.1 System requirements

The platform has been tested on Ubuntu 18.04.3 LTS operating system.

Required software:

- Bash Shell[38]
- Docker Engine[39]
- Docker Compose[40]

On the Tampere University instance of the CityIoT FIWARE platform, Bash Shell version was 4.4.20, the Docker Engine version was 19.03.5, and Docker Compose version was 1.25.1.

Other operating systems which support the use of Docker containers and running Bash scripts, should be compatible with CityIoT FIWARE platform.

4.2 Platform configuration

When setting up a new CityIoT FIWARE platform instance, there exists a few different ways the instance can be deployed. Also, some of the components are optional and can be left out from the deployed FIWARE platform. This section lists the available deployment and component composition options. The instructions on how to modify the platform configuration to accomplish the wanted deployment configuration can be found in the documentation at the GitHub repository.

The overall platform deployment options are:

- **Secure version** where the incoming and outgoing network traffic is secured with HTTPS protocol (using verified certificates) and the data usage services Grafana, Wirecloud, and CKAN are served on their own subdomains (e.g. `https://grafana.<host>`). This is the default and recommended version for any production environment. It requires that the server ports 80 and 443 are open to the users of the platform.
- **HTTP version** where there is no Transport Layer Security used for the network traffic but the data usage services Grafana, Wirecloud, and CKAN are served on their own subdomains. This version can be used in a development server, since it is quicker to setup since it does not require SSL certificates and it only requires the port 80 to be open to the users of the platform.

[37] https://github.com/cityiot/CityIoT-platform#deployment-instructions
[38] https://www.gnu.org/software/bash/
[39] https://docs.docker.com/engine/install/ubuntu/
[40] https://docs.docker.com/compose/install/

- **Localhost version** where there is no Transport Layer Security used for the network traffic and the data usage services Grafana, Wirecloud, and CKAN are served on dedicated ports, i.e. Grafana would be in `http://<host>:3000` instead of `http://grafana.<host>`. This version can be used for quick tests for example on a local computer. It requires the port 80 to be open for the FIWARE components Orion and QuantumLeap and ports 3000, 8000, and 5000 open for Grafana, Wirecloud, and CKAN respectively.

The optional components that can be excluded when deploying the platform are collected in the following list. For QuantumLeap there exists an extra option for whether to use the geocoding using Redis with QuantumLeap. For Grafana, Wirecloud, and CKAN there are extra options for choosing the admin username and password.

- QuantumLeap
- IoT Agent for Ultralight 2.0
- Grafana
- Wirecloud
- CKAN

In addition to the platform deployment options and the component selection options, the host name is an option that the the deployer must choose before generating the commands and rules. The number of replicated components for Orion, QuantumLeap, and IoT agent is also chosen before the deployment of the platform. However, the number of replicas can also be scaled up or down later on while the platform is running.

To make the configuration and deployment of the platform as easy as possible the code repository includes Bash scripts that do the necessary work when changing the options. The scripts generate and modify the various configuration files used by the platform according to the selections made by the user. The modified configuration files include some of the Docker Compose files, Nginx configuration files, and environmental variable files. After the configuration files have been modified, the application stacks for the platform can be deployed by running the platform start script.

4.3 Platform maintenance

This section contains information related to the maintenance of the platform. Any deployed system will always require some kind of maintenance. With the CityIoT FIWARE platform there is no automatic maintenance system included but the Docker swarm system can in most cases restart any components that might have crashed. However, some occasional manual checking on whether all the platform components are working properly is required from platform maintainer to ensure that the platform continues working without problems.

Data backup procedure is described in section 4.3.1 and the procedure of updating access control permissions in section 4.3.2. Both procedures require some manual work from the maintainer. SSL certificates require periodic renewal and section 4.3.3 describes the simple manual process that was used with the Tampere University instance of the CityIoT platform. A memory issue with Orion caused some problems on Tampere University instance and the the issue is described and the how it was circumvented in section 4.3.4.

4.3.1 Data backup and restore

The code repository contains data backup script (for Bash Shell) that allow making backups of the data in the databases used in the FIWARE platform: MongoDB (used by Orion), CrateDB (used by QuantumLeap) and PostgreSQL (used by Grafana, Wirecloud, and CKAN). The script also backs up the used platform configuration settings.

For the historical data in CrateDB, it is possible to only backup the new data received since last backup. Since, the other databases are a lot smaller in size, full backup are always used with them. Instructions on how to use the backup script can be found in the code repository.[41]

The repository also contains data restore scripts. They can be used to restore the data backed up by the backup scripts for all three database types. While using the data restore script for restoring the historical data to the CrateDB, it can happen that some data rows becomes duplicated in the database. This can happen if the data is restored from two files that contain the same data in both files. The code repository also contains a helper script that can be used to remove any duplicate rows from CrateDB.

4.3.2 Updating access control permissions

The process of updating the access control permissions for the Orion API and QuantumLeap API must be done manually by the platform maintainer. It requires manually updating two text files that contain the tokens for the users and the user specific permissions. To deploy the updated permissions to the platform requires restarting the Nginx proxy component. Detailed instructions on how to do the access control updates can be found on the code repository.[42]

4.3.3 SSL certificates

The CityIoT FIWARE platform deployed at the Tampere University used a free SSL certificate provided by Let's Encrypt[43] to allow secure HTTPS connections to the platform. These free certificates are only valid for 90 days, so they need to be renewed regularly. There exists a simple manual procedure that

[41] https://github.com/cityiot/CityIoT-platform#backing-and-restoring-data
[42] https://github.com/cityiot/CityIoT-platform#updating-nginx-access-control-permissions
[43] https://letsencrypt.org/

can be followed to get the SSL certificate.[44] The procedure involves several manual steps but the website contains all the required instructions to complete each step. However, some technical knowledge is assumed by the website. The procedure is the same for getting a new certificate as well as renewing an old one.

For the CityIoT FIWARE platform the code repository has detailed instructions[45] on how to confirm that the host site is actually owned by the maintainer, which is the last step before Let's Encrypt can issue the certificate. The maintainer only needs to edit the instructed Nginx configuration files with the values given during the certification process. Since, a single SSL certificate can have many aliases, all of the used subdomains in the CityIoT FIWARE platform can be covered with a single certificate.

4.3.4 Memory issue with Orion

During the project it was noticed that for the Tampere University instance all the CityIoT FIWARE platform containers in the Docker swarm restarted periodically. This happened once or twice a week. When researching the phenomena it was found out that the Orion Context Broker containers kept reserving more and more memory and never releasing it. This then resulted the whole platform eventually running out of memory which caused all the Docker containers in the swarm to be forcefully closed. Most of the time the swarm manager did manage to restart all the containers successfully and this issue only caused a short downtime for the platform.

As a solution for this issue, a system was developed that closed the Orion containers once a day in a controlled manner, i.e. only one of the replicated Orion containers was put offline at a time. This meant that there was no downtime in the Orion Context Broker service and total memory reserved by the Orion containers was kept at manageable levels. After this fix no container restarts was noticed in this platform instance. Figure 2 shows the FIWARE platform total memory usage (red color indicates the reserved memory) for a time period both before and after the memory fix was put in place for a total of 35 days. For the second half of the time period the Orion containers have been restarted once a day and the total memory usage has not been at very high levels in that time. Figure 3 shows the last 13 days of the same time period and it can be seen that the average memory usage has stayed relatively constant for the whole time.

It is possible that this memory issue was somehow caused by some settings in the virtual machine in which the CityIoT FIWARE platform was run. When there was less data coming in to the platform, the issue was a lot less noticeable.

[44] https://gethttpsforfree.com/
[45] https://github.com/cityiot/CityIoT-platform#updating-ssl-certificate

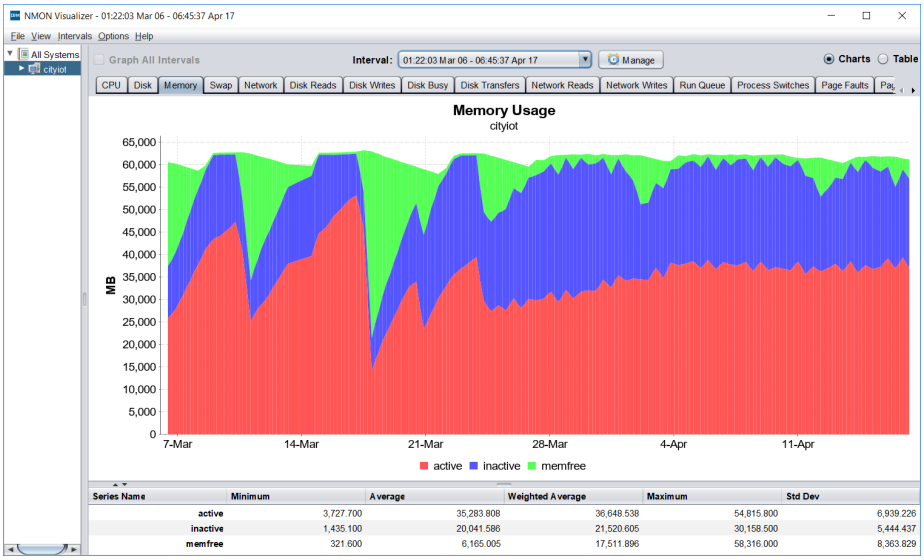

Figure 2: The FIWARE platform memory usage before and after the Orion memory issue fix was but in place.

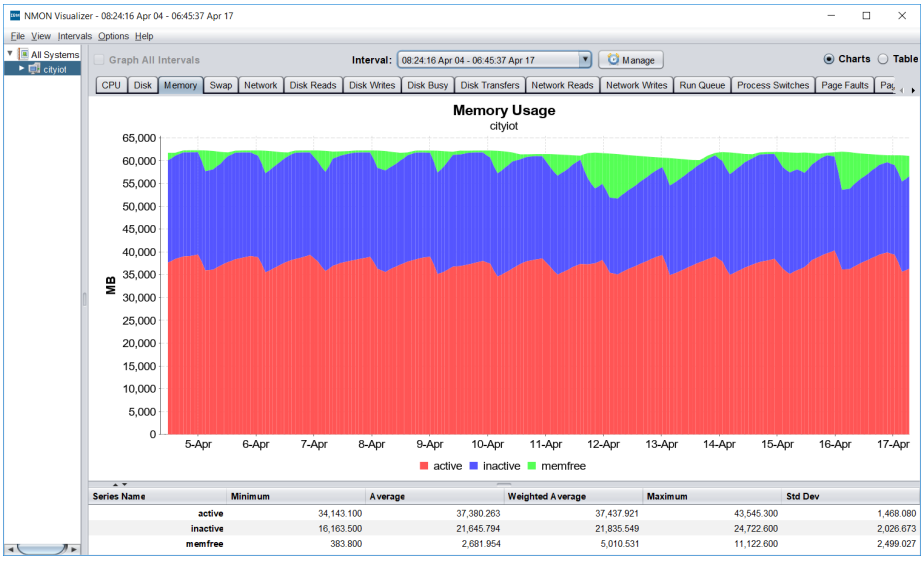

Figure 3: The FIWARE platform memory usage after the memory issue fix.

5 Examples of data usage in FIWARE platform

This chapter describes what kind of data was used with the FIWARE platform during the CityIoT project at the Tampere University. This document concentrates mostly on the data volume and velocity. More comprehensive description about the data gathering process is included in the data gathering document Collecting data to FIWARE[46] that also contains experiences on the data sources used with a FIWARE at the University of Oulu during the CityIoT project. However, a short description including data flow diagrams for the data sources used at the Tampere University are included in this chapter. Section 5.1 describes the data sources. Section 5.2 shows how data in the platform can be visualized. In section 5.3 a demo web application is discussed, which combined data from two different streetlight data sources.

5.1 Data sources and data gathering

There were four data sources from which data was fetched to the CityIoT FIWARE platform. Table 2 shows these data sources and their descriptions.

Table 2: Data sources for Tampere University FIWARE platform.

Data source name	Description	Start time	End time
Tampere streetlights	Streetlight group specific data covering the entire Tampere area	01.10.2018	04.05.2020
Viinikka streetlights	Streetlight specific data covering the Viinikka district	11.04.2019	continues
Electric buses	Various data from 5 electric buses	01.01.2019	continues
Passenger data	Passenger information from one bus line	20.02.2019	30.04.2019

From the data sources presented in table 2 the streetlight data source covering the entire Tampere area is described in section 5.1.1 and the streetlight data source offering data about the smart streetlights at the Viinikka area in Tampere is described in section 5.1.2. The data sources related to the electric buses are described in two sections: in section 5.1.3 for the electricity, speed and location data, and in section 5.1.4 for the passenger data.

[46] https://drive.google.com/file/d/16k77MirUt_AMl6j5jN5y4X6gVY43kFYK

Table 3: The used data in numbers.

Data source	Entities	Total static attrs.	Total dynamic attrs.	Measurement interval	Values per day (on average)
Tampere streetlights	587[47]	1 023	910[48]	6 min – several hours	7 264
Viinikka streetlights	413[49]	4 847	3 212[50]	1 min – 1 hour	605 527[51]
Electric buses	5[52]	25	90	1 s – 15 s	1 042 911
Passenger data	71[53]	350	404	1 day	404
Total	**1 076**	**6 245**	**4 616**	**1 s – 1 day**	**1 656 106 (19,2 / s)**

Table 3 lists the numbers related to the volume and velocity of the data used in the CityIoT FIWARE platform. The entities column gives the total number of different FIWARE entities for the data source (the footnotes give the total number of entities for each entity type). The total static attributes column gives the total number of entity attributes that either do not change or change seldom for each data source. And, the total dynamic attributes column gives the total number of entity attributes that can change more often. From the values per day column it can be seen that the electric bus and the Viinikka streetlights data sources dominated the gathered data in terms of data volume.

5.1.1 Tampere streetlight data

The city of Tampere receives data about the entire streetlight control system once a day by email as CSV formatted files. The city of Tampere personnel then pushes this data to an SQL database in an Azure cloud. Streetlight adapter program fetches the data from Azure and combines and modifies it to an NGSI data model that is based on the data models given in FIWARE Data Models. After the modification the adapter program sends the data to Orion and time series data is then saved by QuantumLeap through the use of the subscription system of Orion. This process of getting the data to the CityIoT FIWARE platform is shown as a diagram in figure 4.

The used data model and the process of collecting the data can found in two documents: Streetlight data model [54] and Streetlight - Lessons learned [55].

[47] 326 streetlight groups, 252 door sensor devices, 3 light sensor devices, 3 weather observations, 3 control cabinets
[48] The 3-phase values for current and voltage are counted as single attributes
[49] 400 streetlights, 6 weather observations, 6 control cabinets, 1 switching group
[50] The 3 angles for the streetlight pole angle values are counted together as single attribute
[51] Dominated by the pole angle values (74 %)
[52] 5 Vehicle
[53] 1 GtfsRoute, 2 GtfsShapes, 2 GtfsTrips, 33 GtfsStops, 33 GtfsStopTimes
[54] https://drive.google.com/file/d/1h4yaGK1GsM329IalvioyhR5gPPhbNOvF
[55] https://drive.google.com/file/d/1cJKOBaMMV5v--ehv_bVIDmbo8vgsURZK

5.1.2 Viinikka streetlight data

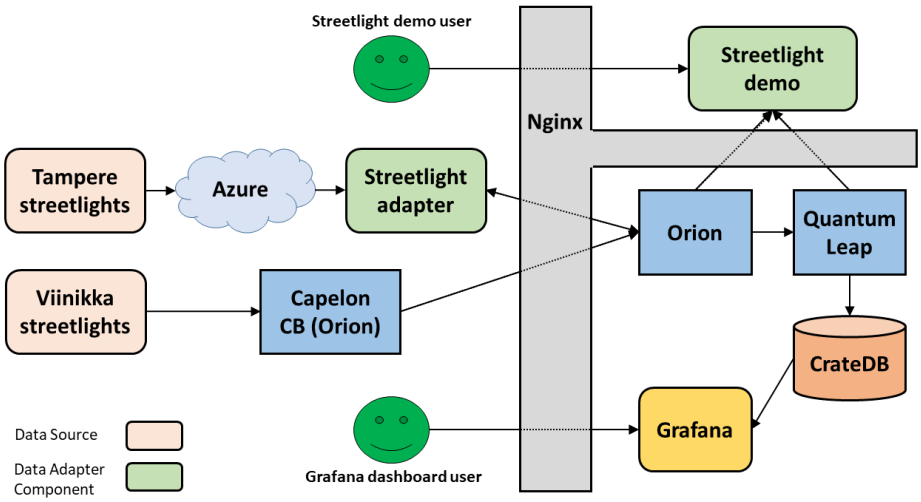

Figure 4: Components for handling the streetlight data.

400 streetlights at Viinikka pilot area send data through wireless network to Orion Context Broker at a FIWARE platform maintained by Capelon. This data is fetched to the CityIoT FIWARE platform through subscriptions made to the Capelon's FIWARE platform. Every time streetlight data is updated on Capelon's platform, it sends a notification to all subscribers. On the CityIoT FIWARE the data is then forwarded to Orion by Nginx using a special `/notify` endpoint to avoid publishing a full token on the Capelon FIWARE platform. The data model used is based on the FIWARE streetlight data model[56]. The process of getting the data to the CityIoT FIWARE platform is shown as a diagram in figure 4.

5.1.3 Electric buses

Measurements from five buses are collected to Wapice IoT-Ticket by Wapice proprietary hardware. The bus data adapter fetches the data using IoT-Ticket API, modifies it to the NGSIv2 data model, and sends it to the CityIoT FIWARE platform. The time series data is sent in 1 minute batches directly to QuantumLeap due to the volume of the data and the problems described in the linked document. Orion is also updated with the latest values of the data batch. The used data model is based on FIWARE Vehicle data model. The process of getting the data to the CityIoT FIWARE platform is shows as a diagram in figure 5.

The used data model and the process of collecting the data is found in two

[56] https://github.com/smart-data-models/dataModel.Streetlighting

documents: Tampere electric bus data model[57] and Converting and transferring data from another IoT platform to FIWARE: case Electric bus[58].

5.1.4 Passenger data

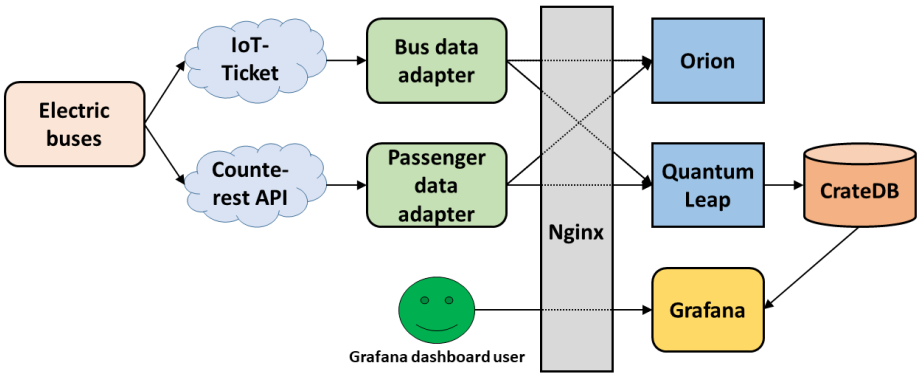

Figure 5: Components for handling electric bus and the passenger data.

Bus passenger data was available for a single route from an API provided by Counterest. The passenger data adapter fetched the data from the API, modified it to NGSI data model, and sent it to FIWARE using the same transfer method as with the electric bus data case. The used data model is based on the FIWARE Urban Mobility data model which in turn is based on the General Transit Feed Specification.

The data collection process for the passenger data can be found at the Storing Bus Passenger Analytics Data into FIWARE document[59].

5.2 Data visualization

A number of Grafana dashboards have been made by the CityIoT project to visualize the gathered data. To give examples of the types of visualizations that can be useful, four examples are shown with screenshots in this section. As mentioned in the Grafana component section 3.2.5, Grafana connects directly to the Crate database used by QuantumLeap. Technical details related to the experience of creating the Grafana dashboards can be found in appendix A.

In the upper graph of the screenshot of figure 6 the green color shows the outside illuminance and the blue the average current on the Viinikka streetlights and the red the average streetlight group current calculated from the entire Tampere area. The blue and red lines are shown in different scales so a direct comparisons of the values are not too useful in this graph. However, it can

[57] https://github.com/cityiot/electric-bus-data-collector/blob/master/spec.md
[58] https://drive.google.com/file/d/1Ct7Rws7NmUY7YbZfEqbWB0515YZHJJqW
[59] https://drive.google.com/file/d/1dgxj_BS5c02d8eiTZucsUVHwCK049Nh5

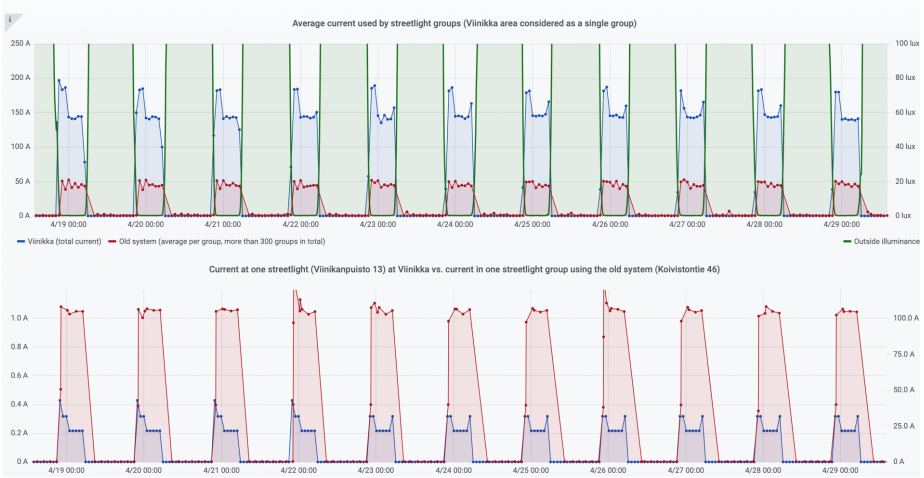

Figure 6: Visualization of average electric current in streetlights using Grafana.

be seen from the graph that streetlights are on when its dark outside, and off during day time. There is some delay seen for the streetlight groups (red line) when switching off but that can be mostly explained by the sparse data points. The lower graph shows the current for one individual Viinikka streetlight and for one streetlight group.

Figure 7: Visualization of pole angle drift of two streetlights using Grafana.

Streetlight poles' angle can drift when compared to their normal near-vertical angle due to heavy winds, ground erosion, or even accidents where for example a car hits a pole. Viinikka area smart streetlights transmit pole drift data. Figure 7 shows pole angle drift data from two of the smart streetlights. The violet color shows the average Z angle of the pole, the maximum Z angle is shown in blue

color, the minimum with green color. The values are calculated from 10 minute time intervals. The variation of the pole angle might seem dramatic from the graphs, but typically the variations are within a couple degrees. This visualization is useful for city personnel tasked with the maintenance of streetlights, as sudden dramatic changes in pole angle can be easily detected and studied.

Figure 8: Visualization of speed and power, location, battery charge, and door status for a Tampere City electric bus using Grafana.

In figure 8 Grafana visualization for one Tampere city electric bus is shown. The topmost graph panel shows the speed (blue line) and power consumption (yellow line) for the bus, the two values correspond well with each other. I.e., after the power goes up, the speed soon follows. Since the scale for the power goes from -100 kW to 300 kw, it can be seen that the bus charges it batteries when it is breaking. The map view panel on the bottom left shows the locations of the bus at the selected time interval, and the route the bus drives can be clearly seen as a red line. When viewing the dashboard through Grafana, the exact location on the map can be easily seen for any selected time. The bottom right the graph panel displays the energy charge of the bus batteries, and the status of the bus doors. Batteries' charge level (blue line) naturally decreases while driving, and doors are of course closed when driving (the red dots on the bottom of the graph). The red dots at middle level of the graph indicate that the bus doors are closing, while red dots at the top of the graph mark the times when at least one of the bus doors was open.

Figure 9 shows a Grafana visualization with two graph panels visualizing passenger data about a single bus route in Tampere. The panel on top shows the total daily passengers, and the panel on bottom shows the daily average occupancy rate. The data is divided such that data from driving the bus route one way is shown with a blue line and driving the bus route the other way is shown with a red line. The graph reveal that the fetched data sets are somewhat incomplete, as can be seen from the missing data points. When there

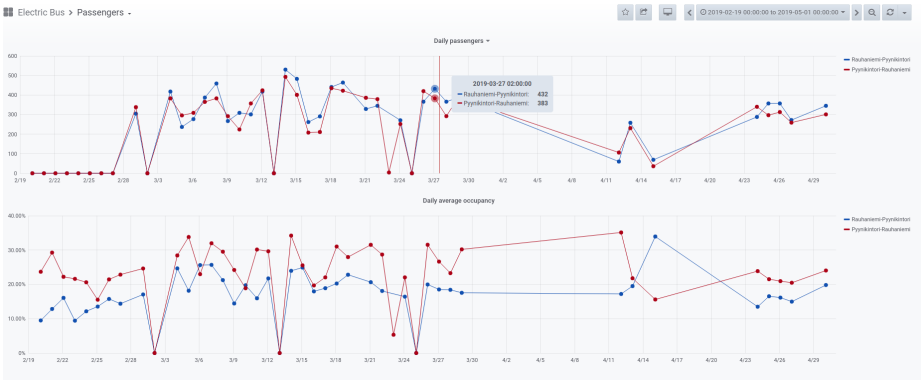

Figure 9: Visualization of total daily passengers and average occupancy rate for a Tampere City electric bus using Grafana.

are missing data points, this leads to possibly misleading output from other analysis programs. Also the visualization of the data can give the viewer a wrong impression, as the line is drawn between the existing sequential data points, without considering the missing data points in between. Data sets with erroneous or missing data points are a common problem that has to be resolved when analyzing the data, and visualizations can help detect these problems.

5.3 Streetlight demo

The streetlight demo web application combines two different sources of streetlight data, one for the whole city of Tampere 5.1.1, and one for the Viinikka area 5.1.2. The demo application fetches the data using the Orion and QuantumLeap APIs and calculates, for example, the hourly energy consumption for each streetlight entity. This calculated data is shown to the user as visual and textual information in the pages of the demo application.

The main motivation for creating the streetlight demo was to demonstrate power of data integration, i.e. two data different data sources are a lot easier handle together when they conform to the same data format, as well as to demonstrate to the city of Tampere the benefits of the smart streetlight system at Viinikka compared to the old system in use at Tampere. Also, creating the demo helped project personnel to gather experience in using the FIWARE platform. Technical details related to the streetlight demo and the usage of the Orion and QuantumLeap APIs withing the streetlight demo context can be found at appendix B.

The source code for the demo is available at the GitHub repository[60] and a running instance of the demo is available at the same server as the CityIoT FIWARE platform[61]. Accessing it requires a username and a password, which the reader can obtain from the Tampere University's CityIoT personnel whose

[60] https://github.com/cityiot/streetlight-demo
[61] https://tlt-cityiot.rd.tuni.fi/streetlight

contact information can be found from CityIoT web page[62]. Below are some screenshots taken from the application.

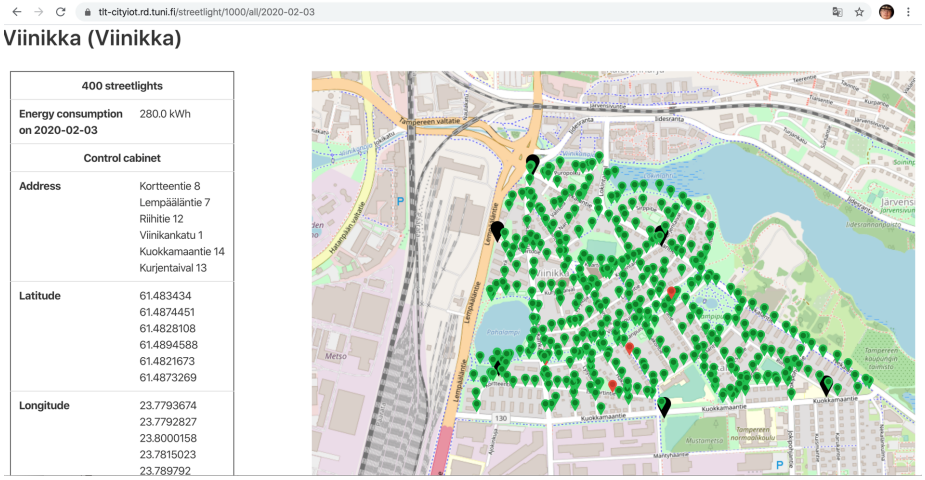

Figure 10: Web app showing general information about Viinikka area streetlights

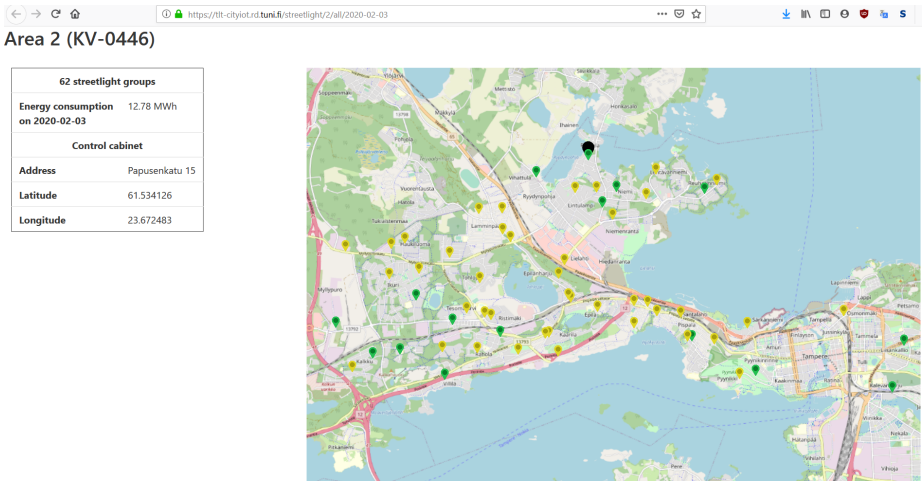

Figure 11: Area 2

Figure 10 shows the demo application page when the user is viewing information about all Viinikka area smart streetlights on a chosen day. Map panel shows the location of the streetlights with smaller location indicator icons, as well as its status with the color of the icon. Green color indicates that the streetlight was sending data on the specified and functioning, red means that

[62] https://www.cityiot.fi/english#contact

there was a problem with the streetlight and no data was received for the chosen day. Black location icons are used for the control cabinets. Text panel on the left side of the map panel shows that there are 400 streetlights in the Viinikka area, which have consumed 280 kWh of energy on the selected day. Control cabinets in the area, as well as their map coordinates are also shown.

Figure 11 shows the demo application page rendered when the user is viewing information about another area called Area 2 which consists of 62 streetlight groups. Whereas Viinikka area smart streetlights can be controlled individually, the older streetlights in Area 2 have been grouped into streetlight groups, all groups managed by a single control cabinet. In this case the smaller location indicator icons represent the streetlight groups, larger blacker icons again reserved for the control cabinet. Compared to Viinikka area visualization, there are many more icons that are yellow, which indicates that for those streetlight groups less than half of the expected amount of data was received for the chosen day.

Page that demo application users are shown when viewing information about a single Viinikka smart streetlight can be seen in figure 12. In this case the user has chosen to view information about a streetlight located at Riihitie 4 on the 6th November 2019. The top left text panel presents information which includes the streetlights name, the control are the streetlight is in, as well as the address the streetlight is closest to, and its map coordinates. Map panel in top center shows the streetlights location. Graph panel on top right shows the hourly energy consumption of the streetlight for the hours of the user-chosen day. The graph shows that the streetlight seems to be working correctly, consuming more energy on the evening and in the morning when there is expected to be more traffic. On the middle of the page there is a date selection form, which the user can use to choose other days to inspect. The bottom of the page shows textual information about when the streetlight switched on and off. It can be seen that the streetlamp has switched on and off within the expected time range. From the "Warning flags" box it can be seen that the streetlamp has there are no warnings: the streetlamp sent all its data, and switched on and off as expected.

Streetlight groups in other areas of Tampere are analogous to the single streetlights in Viinikka from the previous figure 12. An example of a page for a single streetlight group is show in figure 13. It gives user the same information as the for one streetlight, but we can see that the energy consumption is much higher, as there are a number of streetlights in the group. In this case there have been multiple missing data points (but still less than half of the expected data points were missing as is indicated by the "Warning flags" text box) and some of the values have been estimated based on the nearest actual values. In the energy graph the hours that have been marked with bright blue bars had no missing data and hours that are marked with bright red bars had no data available. However, despite the missing data the demo application was able to determine that lights in the streetlight group switched on and off during the expected time range as indicated by the textually presented light data.

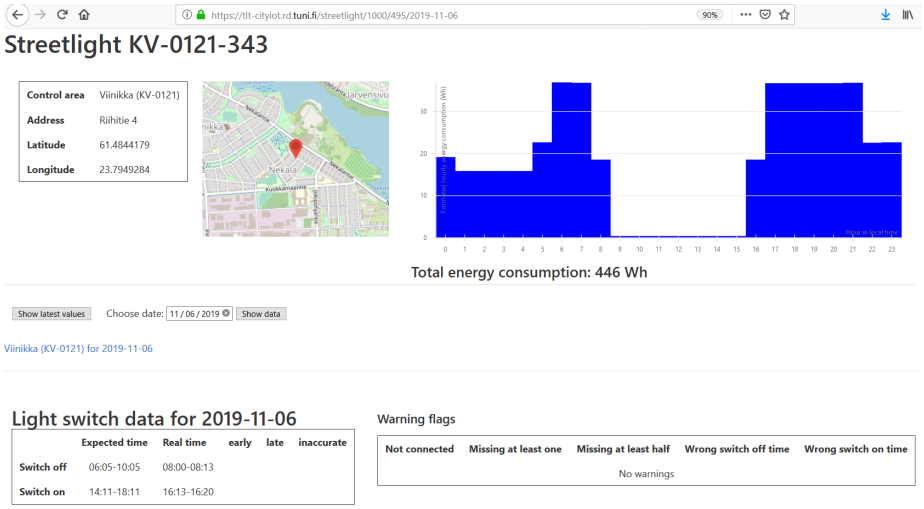

Figure 12: A single Viinikka streetlight

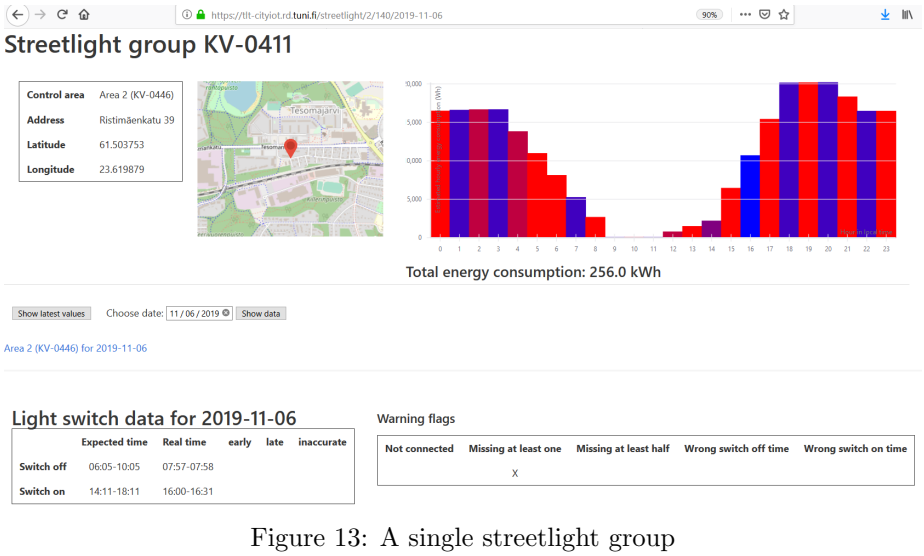

Figure 13: A single streetlight group

6 Summary

This report introduced the CityIoT project's solution for an open source, vendor independent data integration platform for smart cities.

The platform was developed using open source software. FIWARE open source IoT components provide the core of the platform. Open source databases MongoDB, CrateDB, and PostgreSQL provide data persistence. Nginx server is used as a proxy and for access control, and serving static data. The platform is convenient to deploy with the help of the provided tools and to scale using the scalability features of Docker swarm. The platform can be customized to various needs by allowing the platform provider to choose the components they require for their own use when deploying it.

The platform is suitable for different smart city pilots as illustrated by the presented use cases. The integration platform can be used to store data from heterogeneous data sources. The data on the platform is modelled using open data models and can be accessed using openly defined APIs. Access control can be applied to the platform to restrict users access to the data and to make the platform more secure.

We hope that people interested in a complete FIWARE based platform with ready made configurations for access control, secure HTTPS connections and Docker based multi-host deployment, will find the CityIoT platform useful for their own use cases.

A Technical details related to Grafana visualizations

Grafana connects directly to the Crate database used by QuantumLeap. Setting up the Crate database as a data source in Grafana is straightforward and can be done by following the instructions in the QuantumLeap documentation[63].

Creating dashboards such as shown in the section 5.2, requires some familiarity with SQL databases and also some knowledge on how QuantumLeap uses the Crate database. As the first required piece of information, the timestamps for each data row are stored in a table column called `time_index`. And as the second required piece of information, QuantumLeap creates a separate table for each entity type and for each FIWARE service. For example, for data whose entity type is `Vehicle` and for which the FIWARE service `transport` is used, QuantumLeap puts the data in Crate table `mttransport.etvehicle`. With this background knowledge making simple diagrams with Grafana using QuantumLeap data should be quite simple.

Some care is needed when constructing the queries that fetch the data from the Crate database to the Grafana interface. This is because Grafana does very little checking on what kind of queries are send to the database. If the database contain a lot of data, it is not hard to make mistake, and write a query whose results are too large for the server running Grafana to handle. This can crash or at least slow down the database. It should also be kept in mind that the free version of Crate does not provide any user management functionalities and thus Grafana gets admin privileges to the database. If QuantumLeap and Grafana are going to be used in an environment where there are multiple users who are creating dashboards and thus making database queries, it is advisable to switch from the Crate database to the Timescale database. Timescale is also supported by QuantumLeap as a backend database but as of writing the QuantumLeap API does not yet work with Timescale. However, with Timescale the access of the Grafana can be limited by views and other mechanisms, and thus it offers a lot more security to the data.

[63] https://quantumleap.readthedocs.io/en/latest/admin/grafana#configuring-the-datasource-for-cratedb

B Technical details related to streetlight demo

This section considers the usage experience of the Orion and QuantumLeap APIs withing the context of developing the streetlight demo described in section 5.3 using streetlight data from two different data sources: 5.1.1 and 5.1.2. It should be noted that the used versions of Orion and Quantumleap were 2.3.0 and 0.7.5 respectively. Since both components are actively developed, any problems that were noted might be fixed in a newer version.

The demo fetches all the streetlight entities from Orion for the two data sources as well as any illuminance measurement entities that are provided. Since the data source covering the entire Tampere area was only updated once a day and did not allow showing new data in real time, the approach of the demo application regarding the data retrieval was to only fetch the data from FIWARE that user of the demo specifically asks for. This meant that the Orion subscription system was not used with the Viinikka data even though it would have allowed showing real time updates of the Viinikka streetlight to the demo user. Any query result is stored in the internal database of the demo application, so that there is no need to fetch the same history data from QuantumLeap more than once.

B.1 Orion API

Fetching the entity data using the FIWARE NGSI v2 API provided by Orion works as described by the API documentation[64]. The queries have a maximum limit of 1000 results, and this has to be taken into account when dealing with data sources with a lots of different entities. However, the API allows the use of reasonably complex queries that can be used to filter the results and possible avoid the issue of reaching this limitation. Regarding this results limit, the default value for it is 20. The limit can raised up to the 1000 with the query parameter `limit` and and the query parameter `offset` can be used to paginate the results. However, this low default limit can result to use cases where the end user might miss some of the available data and only consider the first 20 entities.

It should noted that the multi tenancy feature[65] with the FIWARE service and FIWARE service paths is an Orion specific addition to the FIWARE NGSI v2 API. Also, the Orion API does not provide any direct way check the to which service or service path a particular entity belongs to. This means that when using these services and service paths, the end user might need some extra knowledge that is not directly available through Orion.

B.2 QuantumLeap API

With the demo, quite a lot of work had to be done with QuantumLeap when considering how to fetch all the required history data in as efficient manner as

[64] http://telefonicaid.github.io/fiware-orion/api/v2/stable/
[65] https://fiware-orion.readthedocs.io/en/master/user/multitenancy/

possible. The QuantumLeap API[66] allows fetching either entity specific data that can contain multiple attributes, or entity type specific data for one attribute. Both of these types of queries were used with demo. The entity type queries were found to be slightly more efficient with regards to the query time but the results are given in a more complicated structure. Also, when there are two or more attributes that are needed for several entities of the same type, running the entity type specific query for each attribute separately reduces the efficiency advantage over the entity specific queries.

The main complication with API queries and how the query results were handled was related to the fact the even though both data sources provided for example the current and voltage, the data types for these values were different. The Viinikka data concerns individual streetlights and the provided electrical values are given as single-phase values, i.e. as real numbers. On the other hand, the other data concerns streetlight groups and the electrical values were provided as 3-phase values, i.e. as JSON objects with the real values for each phase given separately. This meant that both of these data types had to be supported by the demo application. Other data that was available on both data sources like the location data or the outside illuminance data was provided in the same format from both sources. And thus, this part of the data could be handled the same way regardless of the used data source.

The 3-phase electrical data and any data stored as JSON objects in general brings out another complication when the data is queried through the QuantumLeap API. For numerical data the API offers queries retuning aggregated values, e.g. the hourly average values or the daily maximum values for a given time period. Using these aggregated queries is a lot more efficient than fetching all data points and calculating aggregated values externally. However, there is no support for aggregated queries for JSON data. So, it is not possible to query for example the hourly averages of the current for the 3-phase data. Since the demo application needs separate hourly averages for each of the 3 phases of the current, the application has to fetch all the data points to do the aggregation inside the application. While this is less efficient it also means that there might be a lot more results and thus a lot more network traffic is needed to get the wanted results.

As with the Orion API also with the QuantumLeap API the maximum number of results received for a single query is limited. The QuantumLeap API has a maximum limit of 10 000 data points per query. While this might be enough for a wide variety of use cases and especially when using the aggregated queries, the use cases with the streetlight demo often resulted in queries where this limit was broken. As with the Orion API there the received results from the QuantumLeap API can be paginated using query parameter `offset` but unlike the Orion API there is no easy way to check the exact number of results for the used query. This means that the end user needs to keep making new queries while increasing the offset parameter until the received result is empty.

[66] https://app.swaggerhub.com/apis/smartsdk/ngsi-tsdb/0.7

B.3 Observations related to the data

Any data analysis program needs to handle possible errors in the data. This was also the case with the streetlight demo. The Viinikka data source worked quite consistently but there were occasional time periods where some of the expected data was missing. For any missing data the streetlight demo application tries to use simple linear interpolation. If the time period is too long no interpolation is tried. While the missing data points were not a big problem with the Viinikka data, the data contained some erroneous values. There were for example some current values that were more than 100 larger than normal, i.e. the values were something that were not physically possible. These erroneous values were filtered out whenever they were received and found out.

There were less clearly erroneous values received from the other streetlight data source, however, there were a lot of missing data points. This made, for example, trying to estimate the total daily energy consumption quite difficult. The same simple linear interpolation was used to try to fill out any missing values. With the 3-phase values this interpolation was done for each phase separately. Unfortunately, in several cases the missing time periods involved the time periods when the streetlights were supposed to go off or on. In addition for several of the streetlight groups the voltage values were received very rarely, sometimes only once day. Both of these made any estimation even more inaccurate. However, some estimation could still be done as is demonstrated in the screenshots of the streetlight demo section 5.3.

B.4 Summary

Since the actual data with the two data sources was somewhat different (single-phase data versus 3-phase data), it was not possible to implement the functionalities of the streetlight demo in a totally data source independent way. However, the data fetching part could be done in a similar manner with both data sources. And apart from the data differences, the results could be handled in similar way in cases. The demo application does demonstrate that when the data is available in a similar format regardless of the data source location, any application development and data integration is made a lot easier than in a more common case where each data source has its own format. This does require that any data provider follows the commonly used data modeling rules. In the FIWARE case this means that any data is modelled using the FIWARE data models[67].

[67] https://github.com/smart-data-models