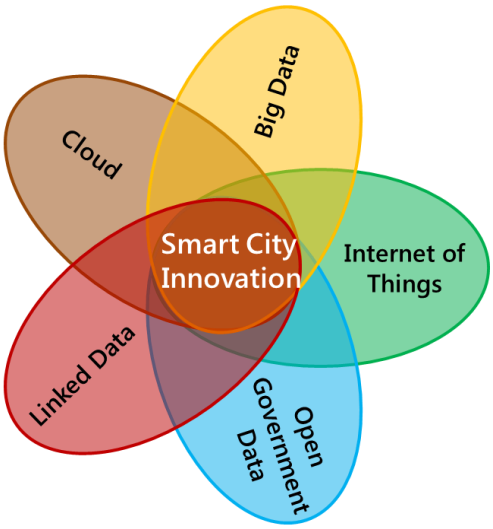
Horizon 2020
European Union Funding
for Research & Innovation

CPaaS.io
City Platform as a Service – Integrated and Open

NICT

H2020-EUJ-02-2016
H2020 Grant Agreement Number 723076
NICT Management Number 18302

CPaaS.io Deployment Guide



FIWARE-Based Toolbox: Installation and Configuration

Everton Luís Berz
NEC Laboratories Europe, Heidelberg, Germany
evertonluis.berz@neclab.eu

Juan Antonio Martínez Navarro
Odin Solutions, Murcia, Spain
jamartinez@odins.es

Table of Contents

1 Introduction

CPaaS.io[1] is a joint R&D project between Europe and Japan whose main goal is to develop such a platform that can be federated to support regional or even global applications, and that forms the basis for a smart city data infrastructure. The project defines two implementation architectures, one based on FIWARE[2] and another one based on u2[3]. Both architectures have been instantiated as a toolbox in Europe and Japan, respectively. This white paper presents a guide to retrieve, install and configure the main components of the FIWARE-based toolbox.

Figure 1 shows an overview of all components described in the next sections. This deployment diagram depicts a static view of the run-time configuration of processing nodes and the components that run on those nodes. The distribution of nodes and components in this schema can be enhanced to provide better performance and scalability, depending on the use case scenario.

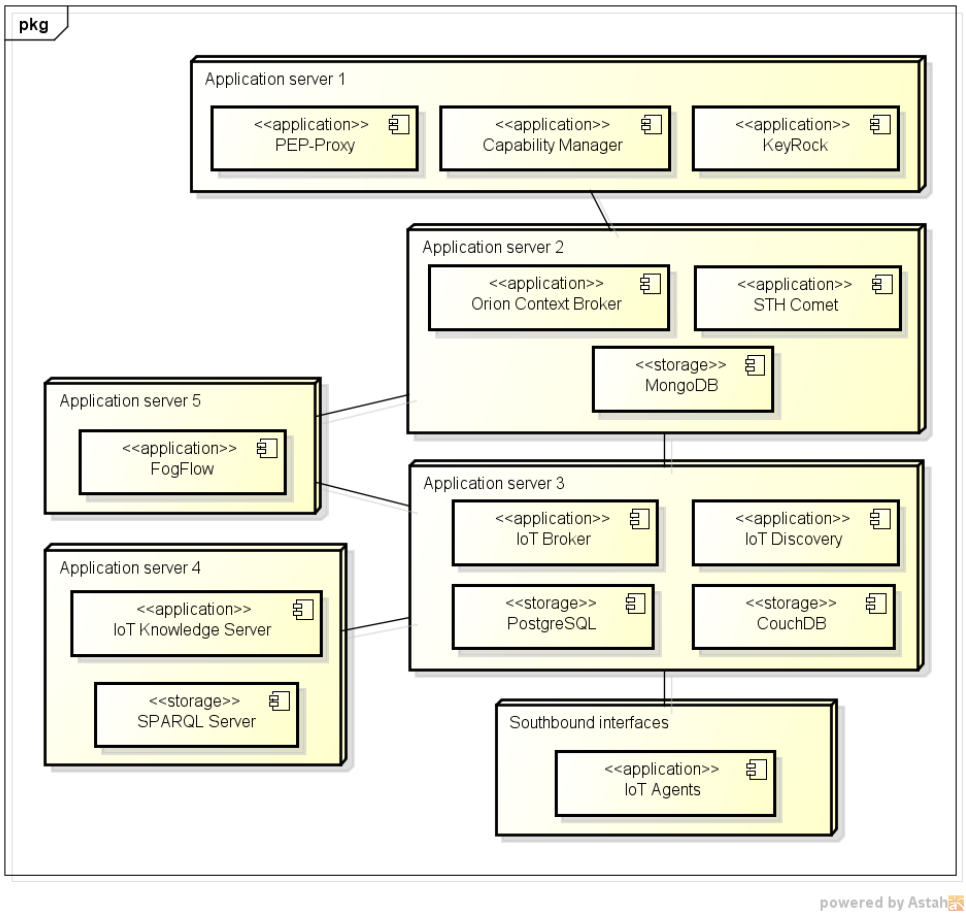


Figure 1. Deployment diagram of FIWARE-based toolbox.

2 Orion Context Broker

The Orion Context Broker [1] is an implementation of the Publish/Subscribe Context Broker GE, providing the NGSI9 and NGSI10 interfaces. Orion allows you to manage the entire lifecycle of context information including updates, queries, registrations and subscriptions. Orion is an NGSIv2 server implementation to manage context information and its availability. Using the Orion Context Broker, you are able to create

[1] "City Platform as a Service - Integrated and Open". http://www.cpaas.io
[2] FIWARE is a curated framework of open source platform components. https://www.fiware.org
[3] https://www.tron.org/wp-content/uploads/2016/06/Open-IoT-Platform-and-IoT-Engine.pdf

context elements and manage them through updates and queries. In order to install Orion, first check the following requirements:

- Operating system: CentOS/RedHat. Recommended version is 7.x.
- Database: MongoDB. Recommended version is 3.6.
- RPM dependencies (some of them are EPEL packages, see http://fedoraproject.org/wiki/EPEL):
    - libstdc++, boost-thread, boost-filesystem, gnutls, libgcrypt, libcurl, openssl, logrotate and libuuid.

Figure 2 presents the steps to download, install and start Orion in a Linux server. First, the FIWARE repository is included in the system and then Orion package is installed. Systemd can be used to initialize the broker. Finally, a sanity check can be issued to verify the broker status and version information.

```
$ sudo wget -P /etc/yum.repos.d/
https://nexus.lab.fiware.org/repository/raw/public/repositories/el/7/x86_
64/fiware-release.repo
$ sudo yum install contextBroker
$ /etc/init.d/contextBroker start
$ curl --header 'Accept: application/json' localhost:1026/version
{
    "orion": {
        "version": "1.7.0",
        "uptime": "10 d, 5 h, 25 m, 30 s",
        "git_hash": "e544780eb64a4a2557c1f51dde070b8d82b86c49",
        "compile_time": "Wed Feb 8 13:30:24 CET 2017",
        "compiled_by": "fermin",
        "compiled_in": "centollo"
    }
}
```

Figure 2. Orion Context Broker installation and initialization procedures.

3 IoT Broker and ConfMan

The IoT Broker [2] is specified as a lightweight and scalable middleware component that separates IoT applications from the underlying device installations. It offers a single point of contact to the user, hiding the complexity of the multi-provider nature of the Internet of Things. IoT Broker collects and aggregates information about thousands of real-world objects on behalf of the user.

The NEC Configuration Management or NEC ConfMan [3] is an implementation of the FIWARE IoT Discovery Generic Enabler. This implementation is specifically designed to interwork with the IoT Broker reference implementation, serving as the registry of FIWARE NGSI context providers. ConfMan is responsible for discovering the availability of context.

IoT Broker and ConfMan have the following requirements:

- Processor: 1 CPU 1.2 GHZ
- RAM: 1 GB
- Disk space:50 MB
- Java 7
- Operating System: 32 or 64-bit version Windows or Linux

Both IoT Broker and ConfMan can be installed together through a docker image. As a docker container does not persist data by default, CouchDB and PostgreSQL need to be installed in the docker host (or another machine). Please follow the steps in Figure 3 to install and configure CouchDB and PostgreSQL in an Ubuntu operating system.

```
$ sudo apt install couchdb
$ sudo apt install postgresql

$ sudo su - postgres
$ psql -c ALTER USER postgres PASSWORD '<password>'

$ echo -e 'host\tall\tall\t<docker_ip>/32\ttrust' >>
/etc/postgresql/9.5/main/pg_hba.conf
$ sed -i "s/^port/listen_addresses = '\*'\nport/g"
/etc/postgresql/9.5/main/postgresql.conf

$ sudo service postgresql restart
```

Figure 3. Steps required to provide permanent storage to IoT Broker and ConfMan.

After having all requirements installed, just pull the docker image from Docker Hub to retrieve the software components. All necessary configurations can be done through parameters in the docker container initialization. Figure 4. IoT Broker and ConfMan installation and initialization steps.Figure 4 shows how to pull and run IoT Broker and ConfMan. Integration with Orion Broker is done through the parameter "iotbroker_consumers_ngsiv1_orion". The parameter "iotbroker_knowledgebaseaddress" enables the IoT Knowledge Server (section 4) integration.

```
$ docker pull fiware/iotbroker:standalone-dev

$ nohup sudo docker run -t -p 8065:8065 -p 8060:8060 -p 9442:9442 \
fiware/iotbroker:standalone-dev \
-p iotbroker_semantic="enabled" \
-p iotbroker_historicalagent="enabled" \
-p confman_semantic="enabled" \
-p iotbroker_entitycomposer="enabled" \
-p iotbroker_exposedAddress=<external_ip> \
-p iotbroker_embeddedagent_couchdbhost="<internal_ip>" \
-p confman_couchdbipandport='http://<internal_ip>:5984' \
-p confman_postgresurl='<internal_ip>' \
-p iotbroker_knowledgebaseaddress='http://<internal_ip>' \
-p confman_knowledgebaseaddress='http://<internal_ip>' \
-p iotbroker_consumers_ngsiv1_orion="http://<orion_ip>:1026/v1/" \
-p iotbroker_loglevel="DEBUG" \
-p iotbroker_embeddedagent_localagentid="agent1" \
-p iotbroker_embeddedagent_historicallyTimestampContextElement="true" >>
/tmp/iotbroker.log &

$ curl http://<external_ip>:8065/sanityCheck
...
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<sanityCheck>
  <name>IoT Broker GE</name>
  <type>Sanity Check</type>
  <version>Version: 6.1.0.SNAPSHOT</version>
</sanityCheck>
```

Figure 4. IoT Broker and ConfMan installation and initialization steps.

4 IoT Knowledge Server

IoT Knowledge Server [4] adds semantic information and reasoning into NGSI messages. It serves semantic knowledge such as entity subtypes or supertypes and provides high-level access to the semantic ontologies via query/subscription functionalities.

The only dependency of IoT Knowledge Server is Apache Jena Fuseki, a SPARQL server used to provide the protocol engine for RDF query and storage systems. In order to install Fuseki, first download the binary distribution from https://jena.apache.org/download/. IoT Knowledge Server was tested with Fuseki 2.6.0. After decompress, run "startFuseki-asDaemon.sh" to start the service.

```
Download IoT Knowledge Server source:
$ git clone https://github.com/Aeronbroker/NECIoTKnowledge.git
$ cd NECIoTKnowledge

Open SystemParameters.txt file and set the configurations as you
wish.

Compile and install binaries:
$ mvn clean install

Run Knowledge Server:
$ nohup mvn spring-boot:run > knowledgeserver.log 2>&1 &
```

Figure 5. Steps to retrieve, compile, configure and run IoT Knowledge server.

You can upload ontologies to the SPARQL server through Fuseki page http://localhost:3030/. First, create a new dataset named "\Subscription" and then upload any other ontology. There are ontologies included in NGSI_Sparql_Examples folder from IoT Knowledge Server. After the SPARQL server is installed and configured, please follow the procedures from Figure 5 to install and run IoT Knowledge Server.

5 FogFlow

FogFlow [5][6] is a distributed execution framework to support dynamic processing flows over cloud and edges. It can dynamically and automatically composite multiple NGSI-based data processing tasks to form high level IoT services, and then orchestrate and optimize the deployment of those services within a shared cloud-edge environment, with regards to the availability, locality, and mobility of IoT devices.

In order to deploy FogFlow, first install Docker and Docker Compose[4], then download the FogFlow deployment script and configuration file from the following URL's, respectively:

- https://raw.githubusercontent.com/smartfog/fogflow/master/docker/core/docker-compose.yml
- https://raw.githubusercontent.com/smartfog/fogflow/master/docker/core/config.json

You can use the default setting for a simple test, but you need to change the following addresses according to your own environment:

- webportal_ip: this is the IP address to access the FogFlow web portal provided by Task Designer. It must be accessible from outside by user's browser;
- coreservice_ip: it is used by all edge nodes to access the FogFlow core services, including Discovery, Broker(Cloud), and RabbitMQ;
- external_hostip: this is the same as coreservice_ip, for the cloud part of FogFlow;
- internal_hostip is the IP of your default docker bridge, which is the "docker0" network interface on your host.

Next, pull the docker images of all FogFlow components using "docker-compose pull". Finally, use the docker-compose tool to start the FogFlow system: "docker-compose up –d". Open the link

[4] https://docs.docker.com/compose/

"http://webportal_ip" in your browser to check the status of all FogFlow running components in the cloud.

5.1 Integration with Orion

The first way to integrate FogFlow with Orion is to consider the broker as destination of any context information generated by a FogFlow IoT service. In this case a NGSI subscription must be issued by an external application or FogFlow Dashboard to ask FogFlow to forward the requested context updates to a specified Orion Broker. Figure 6 shows the JavaScript code to integrate FogFlow with Orion. A JavaScript NGSI library, including the NGSI10Client referred in the example, can be found at https://github.com/smartfog/fogflow/tree/master/designer/public/lib/ngsi.

```
function subscribeFogFlow(entityType, orionBroker)
{
    var subscribeCtxReq = {};
    subscribeCtxReq.entities = [{type: entityType, isPattern: true}];
    subscribeCtxReq.reference =  'http://' + orionBroker + '/v2';

    client.subscribeContext4Orion(subscribeCtxReq).then(
function(subscriptionId) {
        console.log(subscriptionId);
        ngsiproxy.reportSubID(subscriptionId);
    }).catch(function(error) {
        console.log('failed to subscribe context');
    });
}

// client to interact with IoT Broker
var client = new NGSI10Client(config.brokerURL);

subscribeFogFlow('<entity_type>', '<orion_ip>:<orion_port>');
// entityType: the type of context entities to be pushed to Orion Broker
// orionBroker: the URL of your running Orion Broker
```

Figure 6. JavaScript code example of FogFlow integration with Orion Context Broker.

6 STH Comet

The Short Time Historic (STH, aka. Comet) [7] is a component of the FIWARE ecosystem in charge of managing (storing and retrieving) historical raw and aggregated time series information about the evolution in time of context data (i.e., entity attribute values) registered in an Orion Context Broker instance. Comet requires MongoDB, npm and node.js packages, which can be easily installed through apt or yum in Linux distros. In order to retrieve, install and run Comet, please follow the procedures described in Figure 7.

In CPaaS.io project, Comet has been enhanced to handle metadata values. At the time of this writing, a pull request is under review to merge this implementation to STH Comet. Alternatively, it is possible to retrieve Comet from a fork that provides this feature at the following repository: https://github.com/evertonberz/fiware-sth-comet.git

```
$ git clone https://github.com/telefonicaid/fiware-sth-comet.git
$ cd fiware-sth-comet/
$ npm install

To run STH Comet, just run:
$ ./bin/sth
```

Figure 7. Steps to download, install and run STH Comet.

6.1 Integration with Orion

In order to start consuming data from Orion and store in Comet historical storage, one or more subscriptions to Orion are needed. Figure 8 shows how to subscribe to Orion and get data being written to Comet. The duration can be changed in case you want to store historical data for a short period of time.

```
curl -X POST \
  http://<orion_ip>:<orion_port>/v1/subscribeContext \
  -H 'Accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "entities": [
    {
      "type": "<entity_type>",
      "isPattern": "true",
      "id": ".*"
    }
  ],
  "attributes": [],
  "reference": "http://<sthcomet_ip>:<sthcomet_port>/notify",
  "duration": "P99Y",
  "notifyConditions": [
    {
      "type": "ONCHANGE",
      "condValues": []
    }
  ],
  "throttling": "PT5S"
}'
```

Figure 8. STH Comet Integration with Orion Context Broker.

7 Security Components

This section presents the deployment details for each security component in CPaaS.io platform. All security components are available in a single Docker image in CPaaS.io internal git repository.

The first security component to be installed is KeyRock. KeyRock is a FIWARE component responsible for Identity Management. Therefore, it provides user attributes representation and validates if they are correct. KeyRock is written in Python and no extra configuration is required. Optionally, a HTTPS-Proxy can be installed to provide HTTPS support. The HTTPS-Proxy is a servlet which is deployed using the .war file in the webapps directory of Tomcat.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
   <entry key="KEYROCK_IP">KEYROCK_IP</entry>
   <entry key="KEYROCK_PORT">443</entry>
   <entry key="KEYROCK_HTTPS">true</entry>
   <entry key="CA_CERT">PATH_TO_CA_CERT_FILE</entry>
   <entry key="KEYROCK_ADMIN_TOKEN">KEYROCK_ADMIN_TOKEN</entry>
   <entry key="PDP_URL">https://EXTERNAL_IP:8443/XACMLServlet/</entry>
   <entry key="SERVER_KEY_STORE_LOCATION_PATH">PATH_TO_KEYSTORE</entry>
</properties>
```

Figure 9. Capability Manager configuration file with placeholders for KeyRock integration.

The second component is Capability Manager, which is the entity responsible for receiving authorization requests, processing them and generating the Capability Tokens associated to such authorization in case of a positive answer. This component is a servlet that must be installed in Tomcat, so once the .war file is deployed in the right directory the whole deployment takes place. Capability Manager includes a configuration file called "constants.xml" with the information presented in Figure 9.

Finally, PEP-Proxy receives the requests from Orion and IoT Broker in order to enforce the authorization policies. PEP-Proxy is responsible for handling the requests for accessing to the information stored in Orion or IoT Broker. This project is written in Node.js and the configuration can be done in the "config.json" file, which is presented in Figure 10. In order to run PEP-Proxy, just run "npm start" in the root folder of the project.

```
{
  "ProxyPrivKey": "certs/server-priv-rsa.pem",
  "ProxyPubKey": "certs/server-pub-rsa.pem",
  "ProxyCert": "certs/server-cert.crt",
  "CapManagerCert": "certs/sociotal_pub.pem",
  "NGSIServerAddress": "<orion_or_iotbroker_ip>",
  "NGSIServerPort": <orion_or_iotbroker_port>,
  "NGSIServerProtocol": "http",
  "NGSIRootPath": "/v1",
  "PEPPort": 1027,
  "CPABEMasterKey": "cpabe/master_key",
  "CPABEPublicKey": "cpabe/pub_key",
  "MaxConnections": 1
}
```

Figure 10. PEP-Proxy configuration and Orion/IoT Broker integration.

8 Conclusion

This white paper presented deployment guidelines for the FIWARE-based toolbox from CPaaS.io project. The instructions were validated in our test environment. Some configurations can change whether a distinct operating system or package version is chosen. A deployment schema has also been suggested in order to facilitate the adoption of the platform. Please note that most of the software presented is open-source and updates are often released, thus it is important to keep in touch with the FIWARE community and to follow the GIT repositories described in the References section.

9 References

[1] FIWARE-Orion. Accessed December 2018. https://fiware-orion.readthedocs.io/en/latest/.

[2] Aeronbroker/Aeron: Aeron is an Internet-of-Things middleware based on the OMA NGSI 9/10 standard. Accessed December 2018. https://github.com/Aeronbroker/Aeron.

[3] FIWARE IoT Discovery GE implementation by NEC. Accessed December 2018. https://github.com/Aeronbroker/NEConfMan.

[4] Aeronbroker/NECIoTKnowledge: Knowledge Base server for enabling semantic functionalities. Accessed December 2018. https://github.com/Aeronbroker/NECIoTKnowledge.

[5] B. Cheng, G. Solmaz, F. Cirillo, E. Kovacs, K. Terasawa and A. Kitazawa, "FogFlow: Easy Programming of IoT Services Over Cloud and Edges for Smart Cities", in IEEE Internet of Things Journal, Aug. 2017.

[6] FogFlow v2.0 Documentation. Accessed December 2018. https://fogflow.readthedocs.io/en/latest/.

[7] FIWARE-STH-Comet. Accessed December 2018. https://fiware-sth-comet.readthedocs.io/en/latest/