

Open APIs  
for Open  
Minds

# Building your own IoT platform using FIWARE GEIs

José Manuel Cantera Fonseca

Technological Expert. Data Chapter.

[josemanuel.canterafonseca@telefonica.com](mailto:josemanuel.canterafonseca@telefonica.com)

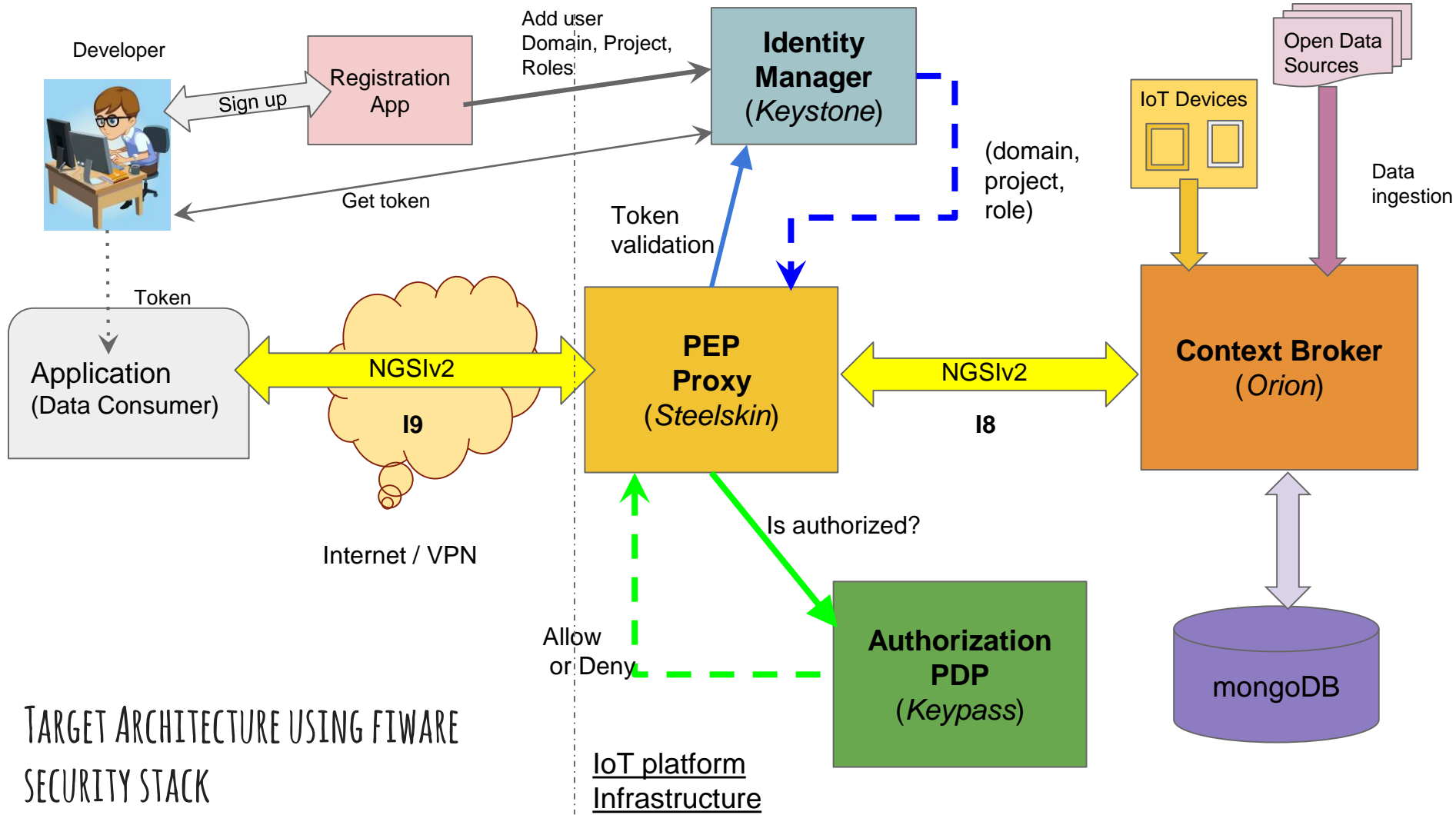
# INTRODUCTION

## Talk Objectives

Illustrate how a secured IoT platform instance can be implemented using FIWARE GEis on a container-based environment (**Docker**)

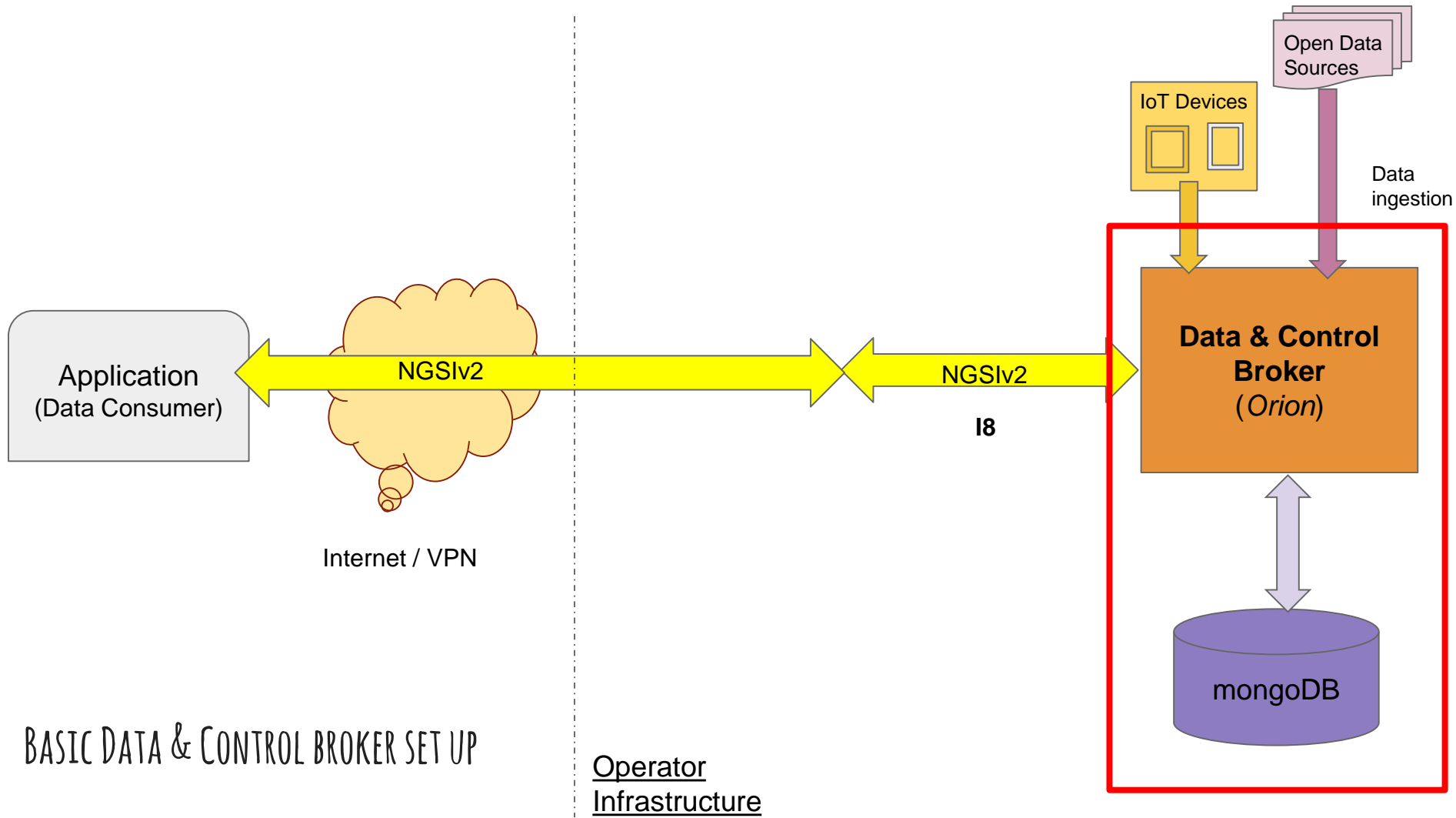
Understand how to **set up** a security layer on top of a Context Broker

Learn how to **configure** all the components at the different layers



# Getting started

A basic context broker set up



# STEP 1.- MONGODB (I)



**mongoDB** is the NoSQL database used to store context data

mongoDB is properly packaged as a docker container

Use **mongoDB 3.2** docker container

Prepare a folder to store mongoDB data

Ex. \$HOME/data/mongo

Run mongoDB

```
$ docker run --name mongo -v $HOME/data/mongo:/data/db -d  
-h mongo -p 27017:27017 mongo:3.2
```

# STEP 1.- MONGODB (II)



```
$ docker ps -a to list running containers
```

```
aa075751485b          mongo:3.2             "/entrypoint.sh mongo" 5
  seconds ago         Up 4 seconds          0.0.0.0:27017->27017/tcp
  mongo
```

run mongo client application to check everything is ok

You might need to install it

<https://docs.mongodb.com/v3.2/tutorial/install-mongodb-on-ubuntu/>

```
$ apt-get install mongodb-org-shell
```

```
$ mongo> show dbs
```

Or you can directly connect to the container

```
$ docker exec -it mongo bash
```

# STEP 2.- ORION CONTEXT BROKER (I)



Orion is

an implementation of the “Data and Context Broker”

An open source project hosted by the FIWARE OSS

<https://github.com/fiware/context.Orion> (License Affero GPL v3.0)

properly packaged as a docker container running on CentOS 6

Orion uses **mongoDB** as the data storage

For running Orion ...

```
$ docker run --name orion -d -p 1026:1026 --link mongo -h orion fiware/orion:1.4.1 -dbhost  
mongo
```

```
$ docker ps -a
```



## STEP 2.- ORION CONTEXT BROKER (II)



```
$ curl -s localhost:1026/version | python -mjson.tool
```

```
{
  "orion" : {
    "version" : "1.4.1",
    "uptime" : "0 d, 0 h, 1 m, 17 s",
    "git_hash" : "905d5fa58ace7fa4f14330ddc982b41cf9b30be6",
    "compile_time" : "Mon Oct 10 15:06:02 UTC 2016",
    "compiled_by" : "root",
    "compiled_in" : "b99744612d0b"
  }
}
```

```
$ mongo > show dbs > use orion
```

Now a DB named “orion” should appear if everything is ok

```
db.getCollectionNames()
```

```
[ "entities" ]
```

```
curl -s localhost:1026/v2/entities | python -mjson.tool
```

# STEP 3.- LET'S ADD SOME ENTITIES TO ORION (I)



```
$ curl localhost:1026/v2/entities -s -S --header 'Content-Type: application/json' -d @- <<EOF
{
  "id": "WeatherObserved-6789",
  "type": "WeatherObserved",
  "temperature": {
    "value": 23,
    "type": "Number"
  },
  "barometricPressure": {
    "value": 720,
    "type": "Number"
  },
  "dateObserved": {
    "value": "2016-10-18T11:08:20.228Z",
    "type": "DateTime"
  },
  "source": {
    "value": "http://www.aemet.es",
    "type": "URL"
  }
}
EOF
```

# STEP 3.- LET'S ADD SOME ENTITIES TO ORION (II)



```
$ curl localhost:1026/v2/entities?options=keyValues | python -mjson.tool
```

```
[
  {
    "barometricPressure": 720,
    "dateObserved": "2016-10-18T11:08:20.00Z",
    "id": "WeatherObserved-6789",
    "source": "http://www.aemet.es",
    "temperature": 23,
    "type": "WeatherObserved"
  }
]
```

```
$ mongo
```

```
> use orion
```

```
switched to db orion
```

```
> db.entities.find({})
```

```
{ "_id" : { "id" : "WeatherObserved-6789", "type" : "WeatherObserved", "servicePath" : "/" }, "attrNames" : [
"temperature", "barometricPressure", "dateObserved", "source" ], "attrs" : { "temperature" : { "type" :
"Number", "creDate" : 1476789680, "modDate" : 1476789680, "value" : 23, "mdNames" : [ ] },
"barometricPressure" : { "type" : "Number", "creDate" : 1476789680, "modDate" : 1476789680, "value" : 720,
"mdNames" : [ ] }, "dateObserved" : { "type" : "DateTime", "creDate" : 1476789680, "modDate" : 1476789680,
"value" : 1476788900, "mdNames" : [ ] }, "source" : { "type" : "URL", "creDate" : 1476789680, "modDate" :
1476789680, "value" : "http://www.aemet.es", "mdNames" : [ ] } }, "creDate" : 1476789680, "modDate" :
1476789680 }
```

# STEP 4.- MULTITENANCY (I)



Orion Context Broker is multitenant

Logical databases **isolated**, each one containing data from different organizations or domains

Tenant is a “**service**” in FIWARE terminology.

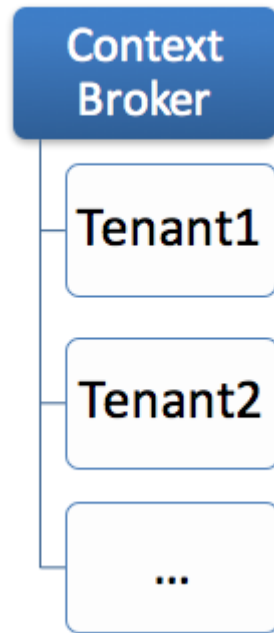
Aka a “Domain” in OpenStack terminology

A tenant can be composed by multiple child sub-tenants

“**subservice**” in FIWARE terminology

Aka a “Project” in OpenStack terminology

Example



# STEP 4.- MULTITENANCY (II)



The way to address tenants are HTTP headers

**Fiware-Service** : <<Tenant\_Name>>

**Fiware-Servicepath**: <<Subservice\_Name>>

Subtenants follow a hierarchical structure and there is a default subtenant, root one ('/')

Example:

Fiware-service: weather

Fiware-servicepath: /Spain

A pair (**service, subservice**) is used for security

# STEP 4 .- MULTITENANCY (III)



- Creating an entity in a (tenant , subtenant)

```
TENANT="Fiware-Service:$1"
```

```
SUBSERVICE="Fiware-ServicePath:/$2"
```

```
curl localhost:1026/v2/entities --header $TENANT --header $SUBSERVICE -s -S --header  
'Content-Type: application/json' -d @- <<EOF
```

```
{  
  "id": "WeatherObserved-6789",  
  "type": "WeatherObserved",  
  "temperature": {  
    "value": 23,  
    "type": "Number"  
  },  
}
```

```
.....
```

```
}
```

```
EOF
```



# STEP 4 .- MULTITENANCY (IV)

```
$mongo
```

```
> show dbs
```

```
local            0.000GB
orion             0.000GB
orion-example_a  0.000GB
orion-london     0.000GB
orion-weather    0.000GB
```

There will be as many databases as tenants available

"orion" is the DB which stores data in the default tenant

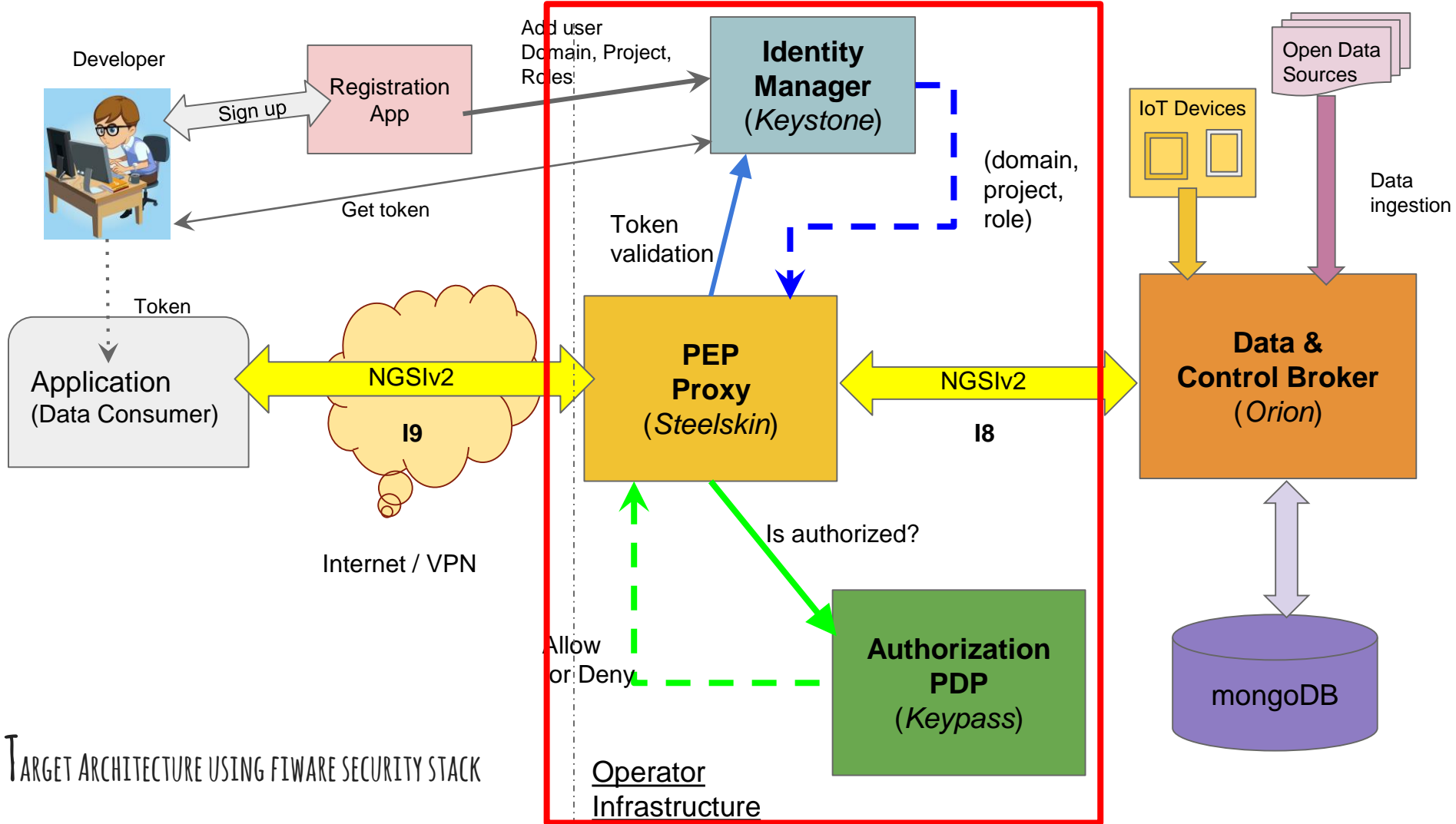
DB name is "orion-" + <<tenant\_name>>

To query data of a tenant just issue regular NGSIV2 requests using *Fiware-Service* and *Fiware-Servicepath* headers

# Deeping dive

Adding a security layer to the data & control broker





TARGET ARCHITECTURE USING FIWARE SECURITY STACK

# STEP 4.- SECURITY STACK - PREPARATION



Security stack uses **MySQL** to store configuration data

```
$ docker run --name mysql -d -p 3306:3306 -h mysql -v $HOME/data/mysql:/var/lib/mysql -e "MYSQL_ROOT_PASSWORD=gsma" -e "MYSQL_DATABASE=keypass" -e "MYSQL_USER=keypass" -e "MYSQL_PASSWORD=keypass" mysql:5.5
```

Check that everything is ok. Above command creates a database named "keypass" used later.

```
$ docker exec -it mysql bash
```

```
mysql --user=root --password=gsma
```

```
mysql> show databases;
```

```
+-----+
| Database          |
+-----+
| information_schema |
| keypass           |
| mysql             |
| performance_schema |
+-----+
```

```
4 rows in set (0.00 sec)
```

# STEP 4.1 .- KEYSTONE (I)



Keystone is an open source project hosted by OpenStack OSS Community

<https://github.com/openstack/keystone>

(Apache 2.0 license)

Keystone is an *Identity Manager* service capable of storing information about domains, project, users, groups or roles

Keystone is in charge of generating tokens which can be used to get access to services requiring credentials

For this exercise we will be using a keystone image

# STEP 4.1 .- KEYSTONE (II)



## Running keystone

```
$ docker run --name keystone -d -p 5001:5001 --link mysql -h keystone
telefonicaid/fiware-keystone-spasword -dbhost mysql -default_pwd 4pass1w0rd -mysql_pwd
gsma
```

### Sanity check operations

```
$ docker logs keystone
```

```
$ docker exec -it keystone bash (to open a shell session on the container)
```

```
$ curl -s -S http://localhost:5001/v3 | python -mjson.tool
```

Once we have a keystone instance up and running different REST requests can be issued

<http://developer.openstack.org/api-ref/identity/v3/index.html>

# STEP 4.1 .- KEYSTONE (II-B)



Checking keystone has created its database properly

```
$ docker exec -it mysql bash
```

```
root@mysql:/# mysql --user=root --password=gsma
```

```
mysql> show databases;
```

```
+-----+
| Database          |
+-----+
| information_schema |
| keypass           |
| keystone          |
| mysql             |
| performance_schema |
+-----+
```

```
mysql> use keystone;
```

```
show tables;
```

```
+-----+
| Tables_in_keystone |
+-----+
| assignment          |
| credential          |
| domain              |
| endpoint            |
| group               |
| migrate_version    |
| policy              |
| project             |
| region              |
| role                |
| service             |
| spassword           |
| token               |
| trust               |
| trust_role          |
| user                |
| user_group_membership |
+-----+
```

# STEP 4.1 .- KEYSTONE (III)



Remember :

Fiware-Service → **Domain** in Keystone

Fiware-Servicepath → **Project** in Keystone

A developer will register in Keystone as *user in a domain*  
<-> Developer can get access to the data offered by the corresponding FIWARE service (tenant)

We will later show how this works in practice

# STEP 4.1 .- KEYSTONE (IV)



List all domains

```
curl -s -S --header x-auth-token:4pass1w0rd http://localhost:5001/v3/domains/ | python -m json.tool
```

```
{
  "domains": [
    {
      "enabled": true,
      "id": "8b883aaa740e4d75b91095eaa550b35c",
      "links": {
        "self": "http://localhost:5001/v3/domains/8b883aaa740e4d75b91095eaa550b35c"
      },
      "name": "admin_domain"
    },
    {
      "description": "Owns users and tenants (i.e. projects) available on Identity API v2.",
      "enabled": true,
      "id": "default",
      "links": {
        "self": "http://localhost:5001/v3/domains/default"
      },
      "name": "Default"
    }
  ]
}
```

# STEP 4.1 .- KEYSTONE (V)



List all users

```
curl -s -S --header x-auth-token:4pass1w0rd http://localhost:5001/v3/users/ | python -m json.tool
```

```
"users": [  
  {  
    "description": "Cloud service",  
    "domain_id": "8b883aaa740e4d75b91095eaa550b35c",  
    "enabled": true,  
    "id": "02cd3dbb6ceb48eb92588c7885bbcc1f",  
    "links": {  
      "self": "http://localhost:5001/v3/users/02cd3dbb6ceb48eb92588c7885bbcc1f"  
    },  
    "name": "pep"  
  },  
  {  
    "description": "Cloud administrator",  
    "domain_id": "8b883aaa740e4d75b91095eaa550b35c",  
    "enabled": true,  
    "id": "177cf5a4d12e4f85b7b65cbcac6d9697",  
    "links": {  
      "self": "http://localhost:5001/v3/users/177cf5a4d12e4f85b7b65cbcac6d9697"  
    },  
    "name": "cloud_admin"  
  }  
]
```



# STEP 4.1 .- KEYSTONE (VI)



Get a token for the *cloud\_admin* user

```
curl localhost:5001/v3/auth/tokens -s -S --header 'Content-Type: application/json' -d @- <<EOF
{ "auth": {
  "identity": {
    "methods": ["password"],
    "password": {
      "user": {
        "name": "cloud_admin",
        "domain": { "name": "admin_domain" },
        "password": "4pass1w0rd"
      }
    }
  }
}
}
}
EOF
```

HTTP/1.1 201 Created

X-Subject-Token: 19e200834a3f4e149c7f4033a003a8f4

## STEP 4.2 .- KEYPASS .- AUTH PDP (I)

*keypass* is an implementation of the FIWARE Authorization PDP (*Policy Decision Point*)

*Keypass* is an open source project hosted at

<https://github.com/telefonicaid/fiware-keypass>

License is Apache 2.0

It complies with XACML (eXtensible Access Control Markup Language) v3.0.

It provides an API to get authorization decisions based on authorization policies

# STEP 4.2 .- KEYPASS .- AUTH PDP (II)



## Running keypass

```
$ docker run --name keypass -d -p 7070:7070 -h keypass --link mysql telefonicaiot/fiware-keypass -dbhost mysql
```

```
$ docker logs keypass
```

```
$ curl --header 'Fiware-Service: dummy' localhost:7070/version
```

1.2.1

## STEP 4.3 .- STEELSKIN .- PEP PROXY (I)



Steelskin is an implementation of the FIWARE PEP (*Policy Enforcement Point*)

Steelskin is an open source project hosted at

<https://github.com/telefonicaid/fiware-pep-steelskin>

License is Affero GPL 3.0

A proxy which ensures that only authorized users are able to perform requests against the Data & Control Broker

<https://github.com/telefonicaid/fiware-pep-steelskin#-rules-to-determine-the-context-broker-action-from-the-request>

The actual endpoint used by applications to get access to

## STEP 4.3 .- STEELSKIN .- PEP PROXY (II)

Running:

```
$ docker run -d --name pep -p 1027:1026 --link orion --link keystone --link keypass -e  
LOG_LEVEL=DEBUG -e AUTHENTICATION_HOST=keystone -e AUTHENTICATION_PORT=5001 -e ACCESS_HOST=keypass  
-e ACCESS_PORT=7070 -e TARGET_HOST=orion -e TARGET_PORT=1026 -e PROXY_USERNAME=pep -e  
PROXY_PASSWORD=4pass1w0rd telefonicaiot/fiware-pep-steelskin
```

```
$ docker logs pep
```

```
$ docker exec -it pep bash → $ curl localhost:11211/version
```

```
{ "version": "1.2.0-next", "port":1026 }
```

# STEP 4.3 .- STEELSKIN .- PEP PROXY (III)



Remember: Given an HTTP Request (**x-auth-token**, **fiware-service**, **fiware-servicepath**)

First PEP queries *keystone* to validate the auth token and obtain (user, domain, role in project)

Then, PEP queries *keypass* to obtain the authorization policies for the role in question

A match between subject policies and the requested operation is done

If the requested operation is allowed, the HTTP request is forwarded to the Data & Control Broker

If not a non-authorized error is raised

```
curl localhost:1027/v2/entities
```

```
{
```

# STEP 5.- USING THEM ALL TOGETHER

```
$ docker ps -a
```

| CONTAINER ID  | IMAGE                                   | COMMAND                | CREATED        |
|---------------|---|------------------------|----------------|
| d18f7dbe7f75  | telefonicaid/fiware-pep-steelskin       | "/bin/sh -c bin/pepPr" | 16 minutes ago |
| Up 16 minutes | 11211/tcp, 0.0.0.0:1027->1026/tcp       | pep                    |                |
| 7e1853f0e2c4  | telefonicaid/fiware-keypass             | "/opt/keypass/keypass" | 45 minutes ago |
| Up 45 minutes | 0.0.0.0:7070-7071->7070-7071/tcp        | keypass                |                |
| c4f6ab6c390f  | telefonicaid/fiware-keystone-spasword   | "/opt/keystone/keysto" | About an hour  |
| ago           | Up About an hour 0.0.0.0:5001->5001/tcp | keystone               |                |
| 5bf5d7e8b284  | mysql:5.5                               | "docker-entrypoint.sh" | 18 hours ago   |
| Up 18 hours   | 0.0.0.0:3307->3306/tcp                  | mysql                  |                |
| 6c63cee20ae2  | fiware/orion:1.4.1                      | "/usr/bin/contextBrok" | 24 hours ago   |
| Up 24 hours   | 0.0.0.0:1026->1026/tcp                  | orion                  |                |
| 4f1d9298fb70  | mongo:3.2                               | "/entrypoint.sh mongo" | 24 hours ago   |
| Up 24 hours   | 0.0.0.0:27017->27017/tcp                | mongo                  |                |

# STEP 5.1 .- ORCHESTRATOR TO THE RESCUE

Manual provision of configurations of the three security components can be cumbersome

TEF has developed an open source project (named *orchestrator*) that helps to provide security configurations

<https://github.com/telefonicaid/orchestrator>

License is Affero GPL 3.0

It can be instantiated as a service but there are some useful scripts which can be used

<https://github.com/telefonicaid/orchestrator/blob/master/SCRIPTS.md>



# STEP 5.2 .- CONFIGURING A SERVICE (TENANT)

```
$ git clone https://github.com/telefonicaid/orchestrator
$ cd orchestrator
$ pip install -r requirements.txt
$ export PYTHONPATH=$PYTHONPATH:$HOME/gsma/orchestrator/src
cd $HOME/gsma/orchestrator/src
./orchestrator/commands/createNewService.py http localhost 5001 admin_domain cloud_admin
4pass1w0rd weatherdata "Weather Data" weather_admin weather_admin_PWD http localhost 7070
Checking that everything went ok
./orchestrator/commands/printServices.py http localhost 5001 admin_domain cloud_admin
4pass1w0rd
```

# STEP 5.3 .- CONFIGURING A SUB-SERVICE

```
$ export PYTHONPATH=$PYTHONPATH:$HOME/gsma/orchestrator/src
cd $HOME/gsma/orchestrator/src
```

```
./orchestrator/commands/createNewSubService.py http localhost 5001 weatherdata
weather_admin weather_admin_PWD "Spain" "Weather in Spain"
```

Checking that everything went ok

```
./orchestrator/commands/printSubServices.py http localhost 5001 weatherdata weather_admin
weather_admin_PWD
```

Now we have a pair (Fiware-Service, Fiware-Servicepath) → (**'weatherdata'**, **'/Spain'**)

We can check that we can get access to data

1/ obtain a token for the `'weather_admin'` user Ex. `'5bb5c6e310814b93a01d74385fe52bef'`

2/ issue a GET request through the PEP proxy

```
curl localhost:1027/v2/entities --header 'Fiware-Service:weatherdata' --header 'Fiware-Servicepath:
```

# STEP 5.4 .- ADDING A DEVELOPER WITH CONSUMER PERMISSIONS

```
$ ./orchestrator/commands/createNewServiceUser.py http localhost 5001 weatherdata  
weather_admin weather_admin_PWD developer1 developer1_PWD
```

Checking that everything went ok

```
./orchestrator/commands/printServiceUsers.py http localhost 5001 weatherdata weather_admin  
weather_admin_PWD
```

Now we need to assign the role "SubServiceCustomer" to the user 'developer1'

```
./orchestrator/commands/assignRoleSubServiceUser.py http localhost 5001 weatherdata Spain  
weather_admin weather_admin_PWD SubServiceCustomer developer1
```

Checking that everything went ok

```
./orchestrator/commands/listSubServiceRoleAssignments.py http localhost 5001 weatherdata  
weather_admin weather_admin_PWD Spain True
```

Now 'developer1' is able to query weather data on the sub-service 'Spain'. However he cannot provide data as his role is 'SubServiceCustomer'

# STEP 5.5 .- GETTING ACCESS TO DATA WITH 'DEVELOPER1' (I)

First of all a token must be obtained . Then :

```
$ curl -s -S localhost:1027/v2/entities --header 'Fiware-service:weatherdata' --header  
'Fiware-servicepath:/Spain' --header 'x-auth-token:36a1d0558612473da438c93d74d4aefc' |  
python -mjson.tool
```

```
[  
  {  
    "barometricPressure": {  
      "metadata": {},  
      "type": "Number",  
      "value": 720  
    },  
    "dateObserved": {  
      "metadata": {},  
      "type": "DateTime",  
      "value": "2016-10-18T11:08:20.00Z"  
    },  
    "id": "WeatherObserved-6789",  
    "type": "WeatherObserved"  
  }  
]
```

# STEP 5.5 .- GETTING ACCESS TO DATA WITH 'DEVELOPER1' (II)

An attempt to create a new entity on (weatherdata,/Spain) will fail

```
curl localhost:1027/v2/entities -s -S --header 'Content-Type: application/json' --header 'Fiware-Service:weatherdata' \
--header 'Fiware-servicepath:/Spain' --header 'x-auth-token:36a1d0558612473da438c93d74d4aefc' -d @- <<EOF
{
  "id": "WeatherObserved-4567",
  ...
}

EOF

{
  "name": "ACCESS_DENIED",
  "message": "The user does not have the appropriate permissions to access the selected action"
}
```

Developer will need to be assigned the role '[SubServiceAdmin](#)' in order to be able to post new data

# WHAT'S HAPPENING BEHIND THE SCENES

A set of predefined policies have been pre-populated to the 'keypass' database

```
docker exec -it mysql mysql --user=keypass --password=keypass
```

```
mysql> use keypass; select policy from Policies;
```

Relevant policies are

<https://github.com/telefonicaid/orchestrator/blob/master/src/orchestrator/core/policies/policy-orion-customer.xml>

<https://github.com/telefonicaid/orchestrator/blob/master/src/orchestrator/core/policies/policy-orion-admin.xml>

# AND FINALLY ...

Remember to remove old docker containers (exited)

```
$ docker rm <container>
```

You should **only expose** the IdM (for tokens) and the PEP Proxy (**ports**) to the developer

Ensure the mounted volumes for database data have **enough space** for the data to be stored

Remember, you can open a shell session on a container

```
$ docker exec -it <<container_name> bash
```

And then get access to the logs, databases, local services, ...

Please use **different passwords** than the used during this

# Questions



| Thank you!

<http://fiware.org>

Follow @FIWARE on Twitter

