Green mobility data models and services for smart ecosystems

D4.3 Source Selection Building Block

Document Identification	
Contractual Delivery Date	28/02/2023
Actual Delivery Date	28/02/2023
Responsible Beneficiary	IMEC
Contributing Beneficiaries	ATOS, FIWARE
Dissemination Level	PU
Version	1.0
Total Number of Pages:	22

Keywords
Sustainable Mobility, Context Broker, Linked Data, Source Selection, Linked Data.

This document is issued within the frame and for the purpose of the GreenMov project. This project has received funding from the European Union's Innovation and Networks Executive Agency – Connecting Europe Facility (CEF) under Grant AGREEMENT No INEA/CEF/ICT/A2020/2373380 Action No: 2020-EU-IA-0281. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

Document Information

Related Activity	Activity 4	Document Reference	D4.3
Related Deliverable(s)	D4.2	Dissemination Level (*)	PU

List of Contributors	
Name	**Partner**
Julian Rojas	IMEC
Brecht Van de Vyvere	IMEC
Filip Gosselé	IMEC
Rémi Ollivier	ATOS
Alberto Abella	FIWARE

Document History			
Version	**Date**	**Change editors**	**Changes**
0.1	12/01/2023	Filip Gosselé	Starting ToC
0.12	14/02/2023	Rémi Ollivier	Add section 2.1 on Context Brokers
0.13	16/02/2023	Filip Gosselé.	Minor changes.
0.14	21/02/2023	Rémi Olivier/Filip Gosselé	Graphs updated/conclusion added.
0.2	23/02	Filip Gosselé	Content ready for review.
0.3	27/02	Rémi Ollivier	Add context broker schemas as recommended by reviewer.
0.4	28/02	Filip Gosselé	Formatting

Document History			
Version	**Date**	**Change editors**	**Changes**
0.9	28/02/2023	María Guadalupe Rodríguez	Quality Review Form
1.0	28/02/2023	Carmen Perea	FINAL VERSION TO BE SUBMITTED

Quality Control		
Role	**Who (Partner short name)**	**Approval Date**
Reviewer	Nuria Bernabé, Eduardo Ilueca (HOPU)	24/02/2023
Quality manager	María Guadalupe Rodríguez (ATOS)	28/02/2023
Project Coordinator	Carmen Perea (ATOS)	28/02/2023

Table of Contents

List of Figures

List of Acronyms

Abbreviation / acronym	Description
CR	Context Registry
CSR	Context Source Registrations
Dx.y	Deliverable number y belonging to Activity x
EC	European Commission
JSON	JavaScript serialized object notation. It is an open standard file format and data interchange format that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and arrays (or other serializable values)
JSON schema	JSON Schema specifies a JSON-based format to define the structure of JSON data for validation, documentation, and interaction control.
JSON-LD	It is a method of encoding linked data using JSON.
LDES	Linked Data Event Stream.
ODALA	ODALA is an initiative (European project) that aims to promote the use of Big Data to facilitate and speed up decision-making in public administrations. This initiative has a social focus and is designed to help the use of Smart Cities technology simply and practically.
SHACL	Shapes Constraint Language. Language for describing Resource Description Framework graphs.

Executive Summary

This document and work fit nicely at the heart of the GreenMov endeavours: as this project contributes to Green Mobility in Europe using data, it must handle massive amounts of data. Where the volume represents a first challenge, the speed or response time is key in several real time use cases too.

Activity 4 focused on the context broker or the central entity that manages the information flow. This document describes performance and solutions to optimize performance in a federated scenario. Federation is relevant where multiple context brokers are used for the same application.

By testing the impact of federation on query performance was measures and source selection was developed to reduce the query execution time.

1 Introduction

1.1 Purpose of the document

Describe how different Context Brokers scale and also how source selection could be implemented over them.

1.2 Relation to other project work

It is part of Activity 4 with context broker federation testing executed by ATOS France and is a building block that must be seen as part of the reference architecture..

1.3 Structure of the document

This document is structured in 3 major chapters.

Chapter 1 presents the introduction.

Chapter 2 presents the performance measurements and the solutions to increase the performance.

Chapter 3 presents the conclusion.

2 Basic software components

2.1 Context broker

2.1.1 Introduction to NGSI-LD

A context broker is used to connect various data sources and data consumers. It allows to share context data between different trade in a decentralized way.

The three brokers we are comparing implement the NGSI-LD API ETSI specification. The latest version is 1.6.1 published in September 2022. The GitHub page of these brokers provide the level of compliance of each broker with the specification.

NGSI-LD describes a data-model and an API using linked-data and JSON format. This standard allows users to query with filtering, create, update, and delete context information. It allows user to subscribe to context data and to receive notifications when entities are created, updated, or deleted.

In the framework of this project, brokers must also provide a temporal interface to be able to request historic data.

Examples of Context Broker we see fit: Orion-LD + Mintaka, Scorpio, Stellio.

Another metric that can be useful when choosing one broker is the load performance of each broker. So, some load tests have been made depending on the different use cases of the project.

2.1.2 Testing and performance

To make the tests, a Kubernetes environment hosted on OVHCloud [1] has been used. The cluster characteristics were: 2 CPU, 8 GB of RAM and 50 GB of storage. The test tool that has been chosen is Hyperfoil. It is a web benchmark tool, licensed under Apache. This tool allows the simulations of a lot of virtual users that can request the system totally independently.

Four benchmarks have been performed with this first Kubernetes environment:

- The first benchmark was about the number of updates a broker can support per second depending on the number of entities in the broker via POST "/ngsi-ld/v1/entities/<ENTITY_ID>/attrs".
- The second benchmark was about the number of GET requests a broker can handle per second retrieving all entities in the broker via GET "/ngsi-ld/v1/entities?type=AirQualityObserved".
- The third benchmark was about the number of GET requests a broker can handle per second when retrieving one single entity via GET "/ngsi-ld/v1/entities?type=AirQualityObserved&id=<ENTITY_ID>".
- The fourth benchmark was about the number of GET requests, a broker can support per second for historic data depending on the number of updates via GET "/ngsi-ld/v1/temporal/entities/<ENTITY_ID>". For each entity update, 5 attributes of the entity were updated. There was only one entity into the broker.

Finally, one last benchmark has been performed with a bigger Cluster Environment to test if horizontal scaling could improve the results. A Kubernetes environment hosted on OVHCloud has been used with 2 nodes with 4 CPU, 15GB of RAM and 100GB of storage. The choice has been made to make again the fourth benchmark describe above in order to compare the results.

In all these tests, "AirQualityObserved" entities have been used. Results are presented for each broker in the next sections.

Results describing the evolution of the amount of updates per seconds according to the number of entities inside the broker:

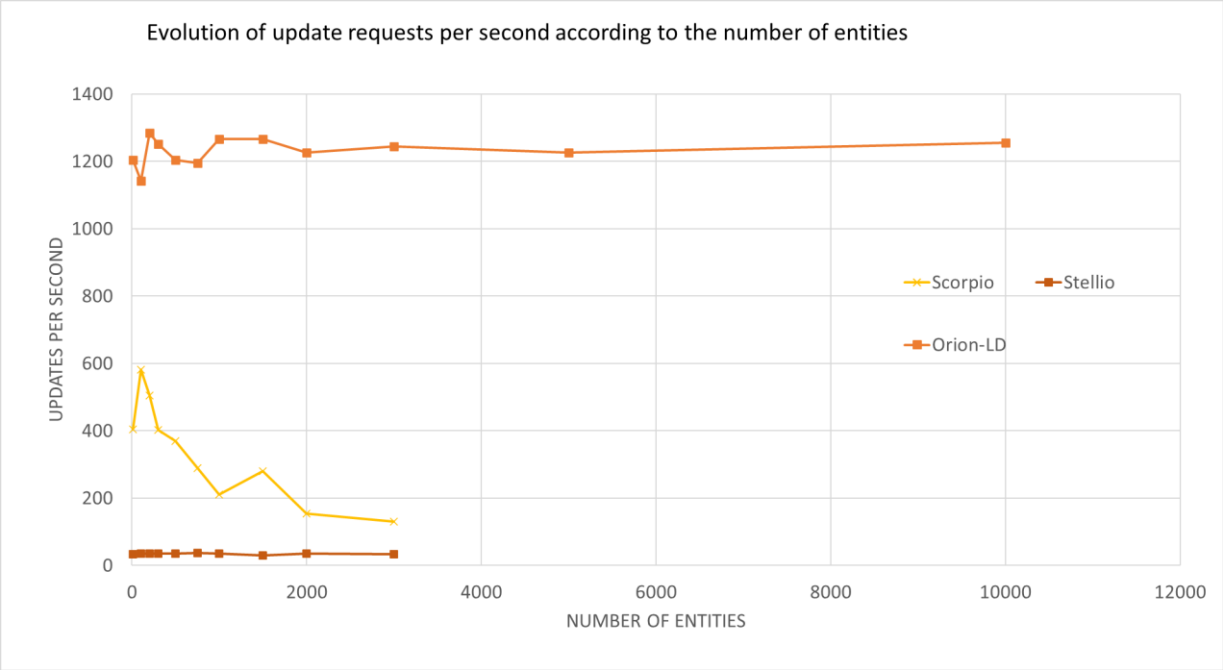

Figure 1: Evolution of update rate according to the number of entities.

Results for benchmark number 2, about the number of queries brokers can handle when retrieving all entities depending on the number of entities:

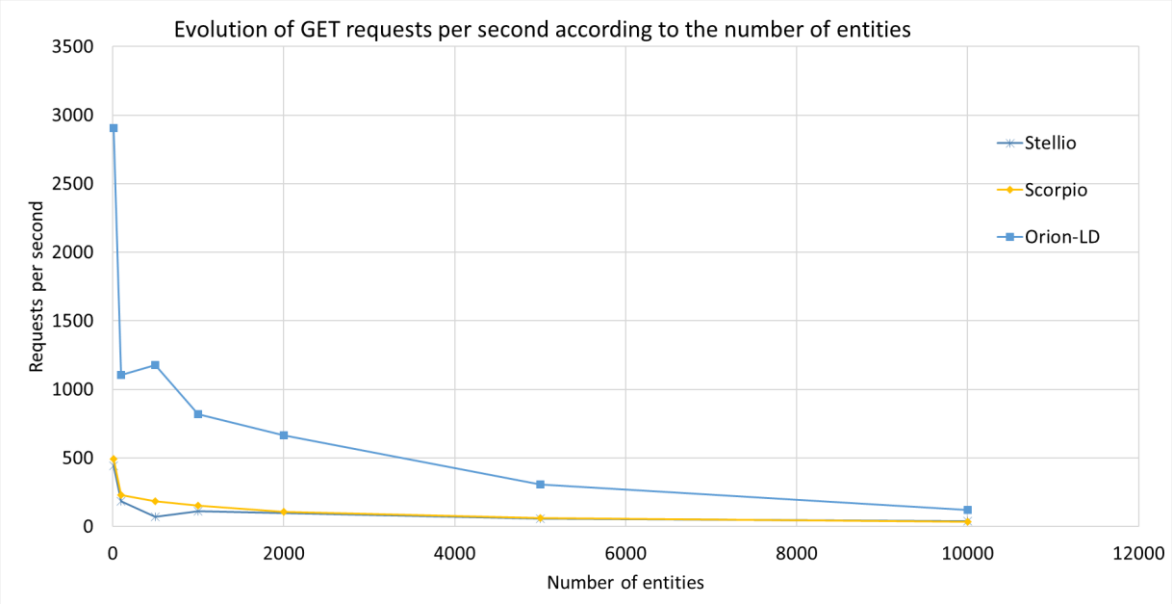

Figure 2: Evolution of query rate for all entities according to the number of entities

Graph describing the number of queries each broker can support when retrieving one single entity depending on the number of entities:

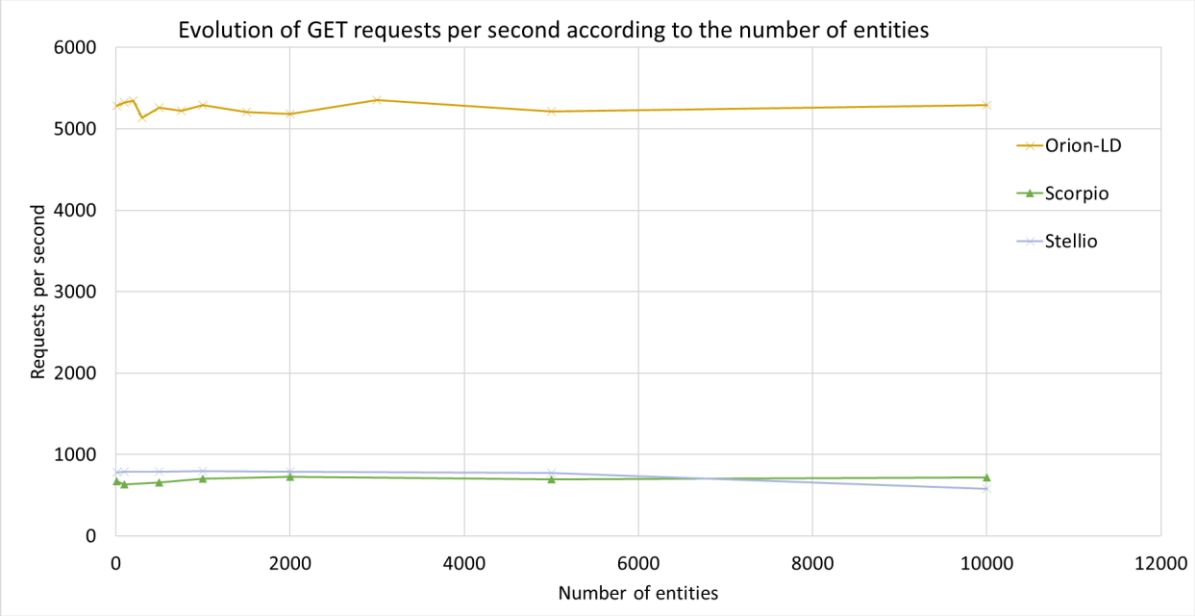

Figure 3: Evolution of queries rate for one entity according to the number of entities

Results for benchmark number 4, describing the evolution of queries per seconds on temporal interface according to the number of updates:

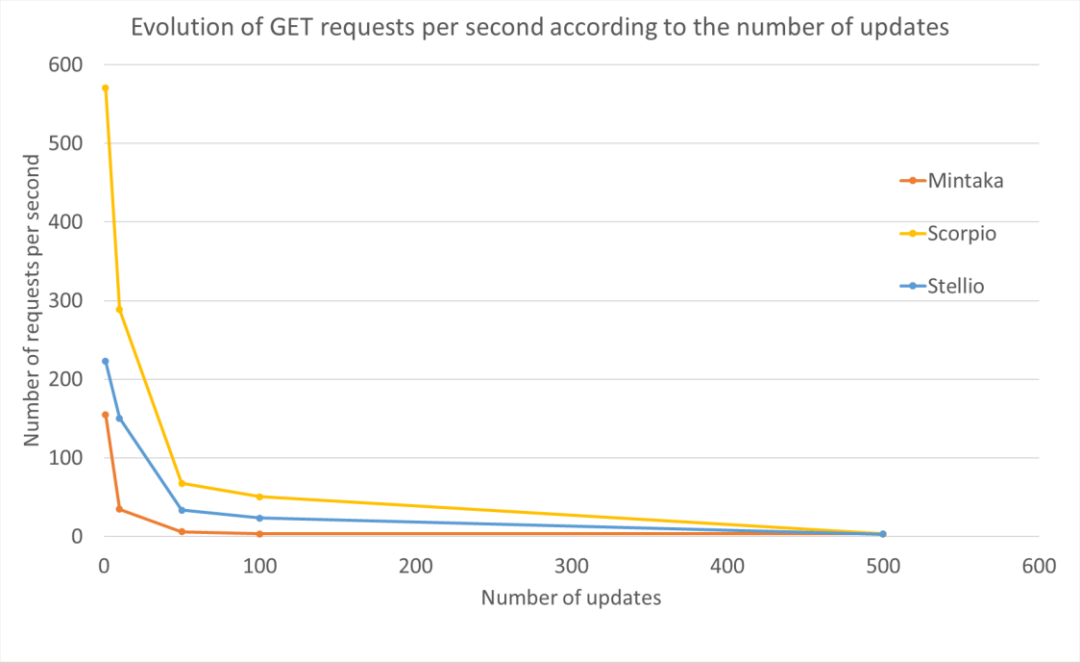

Figure 4: Evolution of queries rate according to the number of updates

The graph below corresponds to the same benchmark than the one above but with a Kubernetes cluster with more resources.

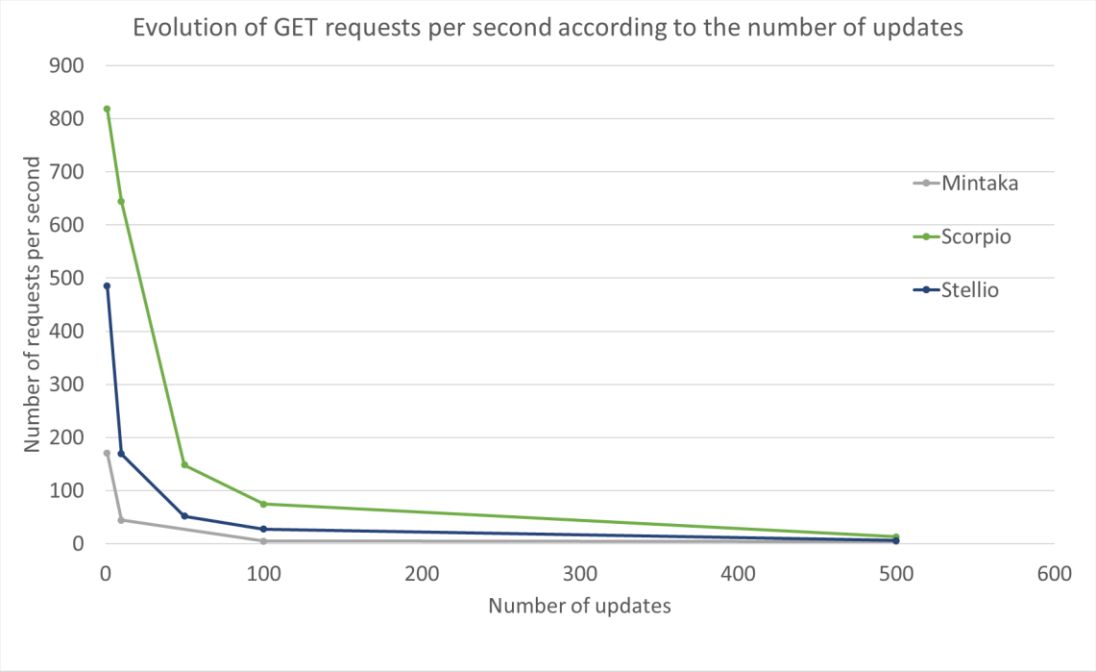

Figure 5: Evolution of query rate according to the number of updates with a bigger Kubernetes cluster.

2.1.2.1 Orion-LD + Mintaka

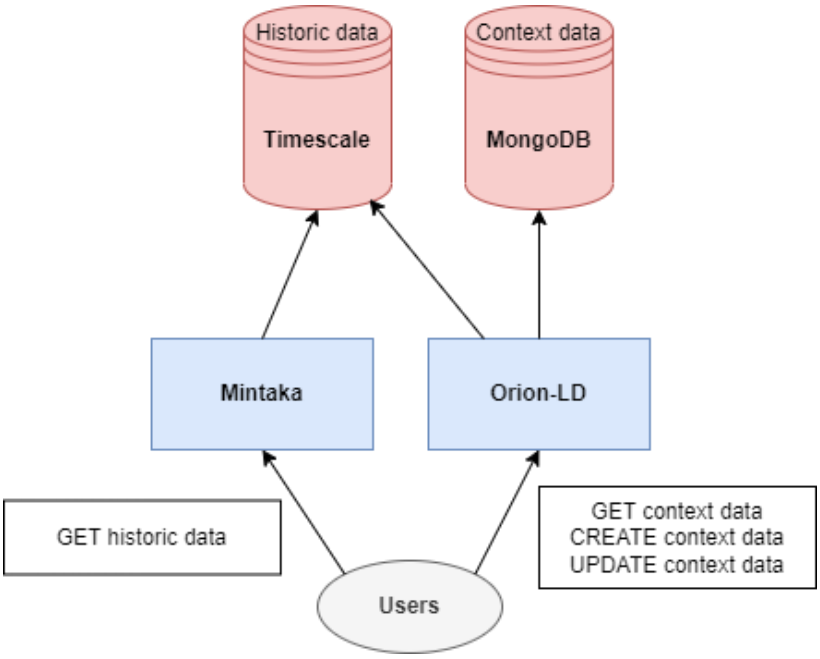

Figure 6: Orion-LD architecture

Orion-LD provides a NGSI-LD API for context data stored in MongoDB; it also provides a temporal API called Mintaka. Temporal data are stored in a Timescale database. These tests have been made with the version 1.1.1 of Orion-LD and 0.5.15 of Mintaka. For these tests, the option ORIONLD_MONGO_ID_INDEX has been set to true to automatically create an index on "_id.id" into MongoDB.

According to the figure 1, Orion-LD can handle around 1.200 updates per seconds whatever the number of entities inside Orion-LD.

According to the figure 2 relative to the number of queries for all entities, the Orion-LD performance decreases with the number of entities increasing. Orion-LD can manage almost 3.000 queries with 10 entities into the broker but less than 150 queries with 10.000 entities into the broker.

The figure 3 shows the results for GET context data. Orion-LD can handle between 5.000 and 6.000 GET request per second for context data whatever the number of entities in the broker.

When retrieving historic data on figure 4, Mintaka can handle more than 150 queries per second with a small number of updates but this result drops quickly with the number of updates increasing.

Now, this result can be compared to results obtained with a Kubernetes cluster with more resources. On figure 5, it can be observed that Mintaka provides a slightly better performance with more resources.

2.1.2.2 Scorpio

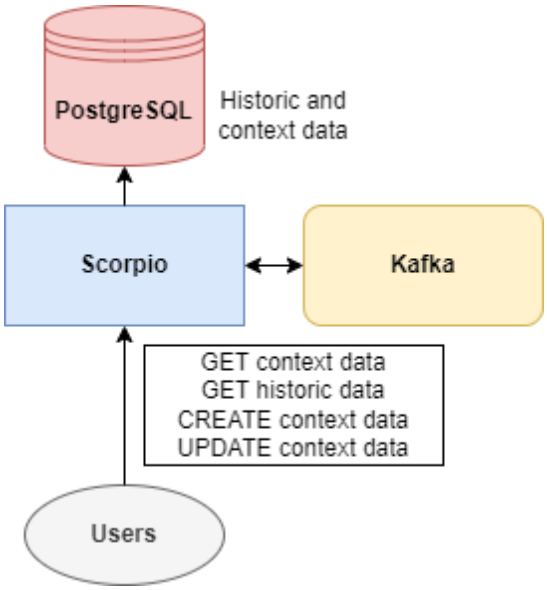

Figure 7: Scorpio Architecture

Scorpio uses PostgreSQL to store context and historic data, it also provides both API for context and historic data. Version scorpio-aaio-no-eureka_2.1.22 of Scorpio has been tested and some PostgreSQL options have been tested to increase the number of workers but without significantly improving Scorpio performance.

Results on figure 1 indicate that Scorpio can handle between more than 600 updates per seconds with a small number of entities but performance decreases with the number of entities increasing.

Regarding the performance for GET requests on the context API, Scorpio performance is decreasing with the number of entities increasing in the broker. Indeed, it can handle almost 500 GET per seconds with a small number of entities while it can handle less than 50 GET per seconds when retrieving 10.000 entities.

The third benchmark results shows that Scorpio can handle between 650 and 750 queries per seconds when retrieving one single entity whatever the number of entities inside the broker.

The performance for GET requests on the temporal API are presented on figure 4. Scorpio performance is decreasing with the number of entities increasing in the broker. Indeed, with only one update, Scorpio can handle more than 500 GET per seconds. However, with more than 500 updates it can reach less than 10 GET per second.

According to figure 5, we can see that Scorpio has slightly better results when tests are made on a Kubernetes cluster with more resources.

2.1.2.3 Stellio

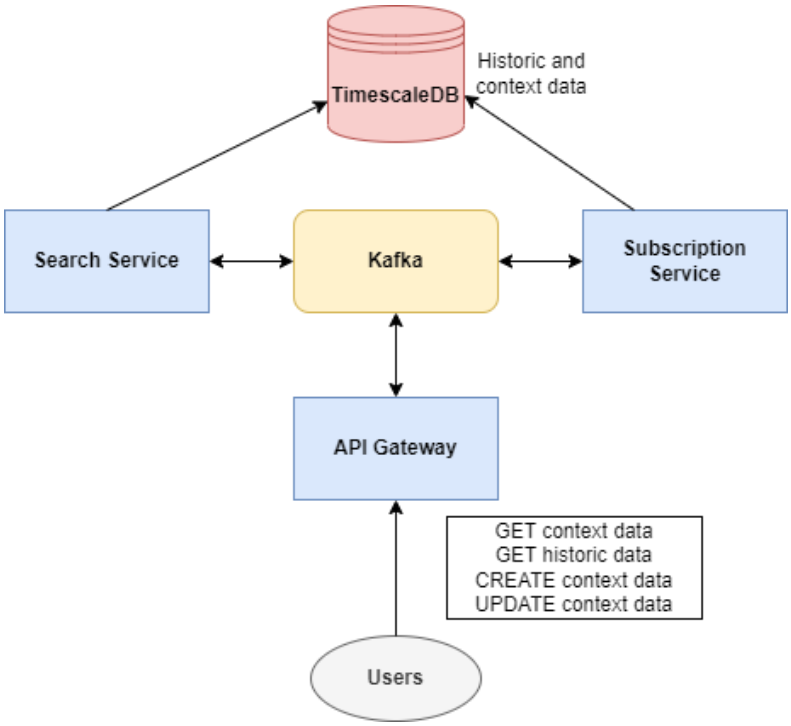

Figure 8: Stellio architecture

Stellio is a context broker developed by EGM. Stellio provides an API to query context and historic data that are stored in TimescaleDB.

Results were obtained without modification in database configuration and with 6 replicas of the search-service and api-gateway, which are micro-services part of the Stellio broker.

The first benchmark results indicate that Stellio has a very stable performance with the number of entities increasing when updating entities. According to figure 1, Stellio can handle around 250 updates per seconds whether for one entity or for 3.000.

The second benchmark about GET requests Stellio can handle to retrieve entities demonstrate that Stellio performance decrease with the number of entities increasing. Stellio can handle almost 450 queries with 10 entities into the broker but less than 50 queries with 10.000 entities.

The third benchmark indicates that Stellio has a stable performance when it stands to retrieve one single entity inside the broker. Indeed, Stellio can handle around 800 queries per seconds while the number of entities vary from 1 to 5.000, but this amount drops to 600 queries per seconds with 10.000 entities into the broker.

The figure 4 shows the results for the fourth benchmark about the number of queries Stellio can handle on the temporal interface depending on the number of updates made on the single entity inside the broker. The graph indicates that Stellio performance is decreasing with the number of updates increasing. Stellio can handle between 200 and 250 query per seconds with only one update while it can handle less than 10 query per second with more than 500 updates.

When we compare previous graph with results obtained with more resources, we can see that results are much better for a small number of updates but only slightly better when the number of updates is higher than 50.

2.1.3 Conclusion

These results show that the three brokers benchmarked have the same performance characteristic. When updating entities and when querying one single entity they have stable performance with the number of entities increasing. When querying all entities or querying all historic data of an entity, performance drops with the number of entities or the number of updates increasing. This behaviour is quite normal because more the larger the response size, the longer the request takes.

With these results, Orion-LD seems to be the context broker that can handle the most updates per seconds whatever the number of entities that are into the broker. Furthermore, Orion-LD is also the best broker when it stands to query context API to retrieve entities. However, when retrieving historic data Orion-LD has not the best results, Scorpio and Stellio can handle more queries.

However, these results can only be indicative because the difficulty when it stands to compare brokers performance is that there is a lot of parameters that can affect the results:

- First the size of the cluster where tests are made. The last benchmark was about testing to scale horizontally the Kubernetes cluster. Results shows that it increases slightly the performance, but more tests should be done with other benchmark to validate this trend. Furthermore, other databases configuration should be tested with different sizes of cluster.
- Then some customization on databases have been made for Orion-LD and Scorpio but more tests should be carried out to try new configurations (indexing databases, partitioning, increasing the number of workers). Furthermore, the best configuration will necessarily be different depending on the project. Indeed, the number of entities, the frequency of updates, the number of queries and the cluster resources are some inputs that must be considered when configuring the broker.
- Finally, database performance can be improved by creating databases clusters, something that has not been tested during GreenMov project.

According to this, the choice of the broker will necessarily depend on the project and on the use cases of the project.

2.2 NGSI LDES

The NGSI-LDES component acts as a scalable data synchronization/exchange interface that publishes one or more Linked Data Event Streams (LDES) from NGSI-LD compliant context brokers. LDES defines a hypermedia-based Web API that provides access to immutable and semantically annotated data subsets containing the (historical) state of a certain data collection. For example, it can be used to read the latest state of a bicycle sharing station but also allows to query for previous states of that same station, as reported in the past.

To allow for efficient and scalable data access, an LDES defines a certain logical data structure that ensures that historical data records, which will not change any further, are served as immutable resources (in terms of HTTP

caching), lowering the cost for serving historical data and thus improving scalability. Multiple logical data structures can be implemented for an LDES, (e.g., linked lists, b-trees, skip lists, etc.). The choice is made at design time and largely depends on the subjacent data sources and their querying capabilities. In the context of GreenMov, a b-tree like hierarchical data structure was chosen, which is supported by the temporal querying interfaces of NGSI-LD context brokers. Each node in the tree represents aggregated time windows of a certain granularity (e.g., week, day, hour, etc.). Such design allows the LDES to behave as a virtualized view over the data that is hosted in a context broker, i.e., the data does not need to be duplicated anywhere else, and also lowers the cost for historical queries, since historical fragments only need to be request to the context broker once and be served from cache onwards.

In terms of data content, the NGSI-LDES module produces an independent LDES stream for every entity type that can be found within a NGSI-LD context broker. Each LDES will continuously produce versioned members (as in LDES/TREE notation) which will contain links to the respective (versioned) entities which are defined in correspondence to the specific (smart) data model used by the context broker.

Additionally, the NGSI-LDES component leverages the NGSI-LD types of interface of a context broker to automatically generate a compliant DCAT metadata catalog. NGSI-LDES is available as open source on GitHub[2].”

2.3 Coverage index.

Due to time limitations, the coverage index has not been implemented in GreenMov. In future work, the geospatial coverage should be added to datasets in the DCAT description of NGSI-LDES.

2.4 Source selection building block

We implemented a source selection library to compare source selection on specific types of entities with certain properties from a Context Registry (CR) with sources that expose a DCAT catalogue. A CR source selection client is responsible for handling a CR by querying the context source registration list with query parameters type and attributes. A DCAT source selection client is responsible for handling a source containing DCAT information by using the Comunica query engine. Comunica was chosen thanks to its ability to query over heterogeneous Linked Data interfaces. The source selection query is transformed into a SPARQL query and used as input for the engine to query over the DCAT source. The library is open source available on Github[3]:. Drawing 1 gives an example of searching for datasets that contain resource reports entities that in turn, have a location property.

The source selection library is used in a demonstrator (Fig. 10) with the following process:

1. The user selects the type of entity and properties it wants to retrieve.
2. The consumer selects which CRs or DCAT catalogs need to be contacted.
3. Depending on (2), the source selection library runs a CR or DCAT source selection client.
4. The clients output the relevant sources that meet the requirements of 1.
5. The relevant sources and the required time to perform the source selection are listed for the user.

```
PREFIX tree: <https://w3id.org/tree#>
PREFIX dcat: <http://www.w3.org/ns/dcat#>
PREFIX sh: <https://www.w3.org/ns/shacl#>
PREFIX pth: <https://purl.eu/ns/mobility/passenger-transport-hubs#>
PREFIX ngsi-ld: <https://uri.etsi.org/ngsi-ld/>
SELECT DISTINCT ?source
WHERE {
   ?dataset a dcat:Dataset ;
      tree:shape ?shape .
   ?service dcat:servesdataset ?dataset ;
      dcat:endpointURL ?source .
   ?shape a sh:NodeShape ;
      sh:targetClass pth:ResourceReport ;
      sh:property [ sh:path ngsi-ld:location ] .
}
}
```

Figure 9: Drawing 1- Example of SPARQL query to select datasets that contain entities of type ResourceReport and have a location property.

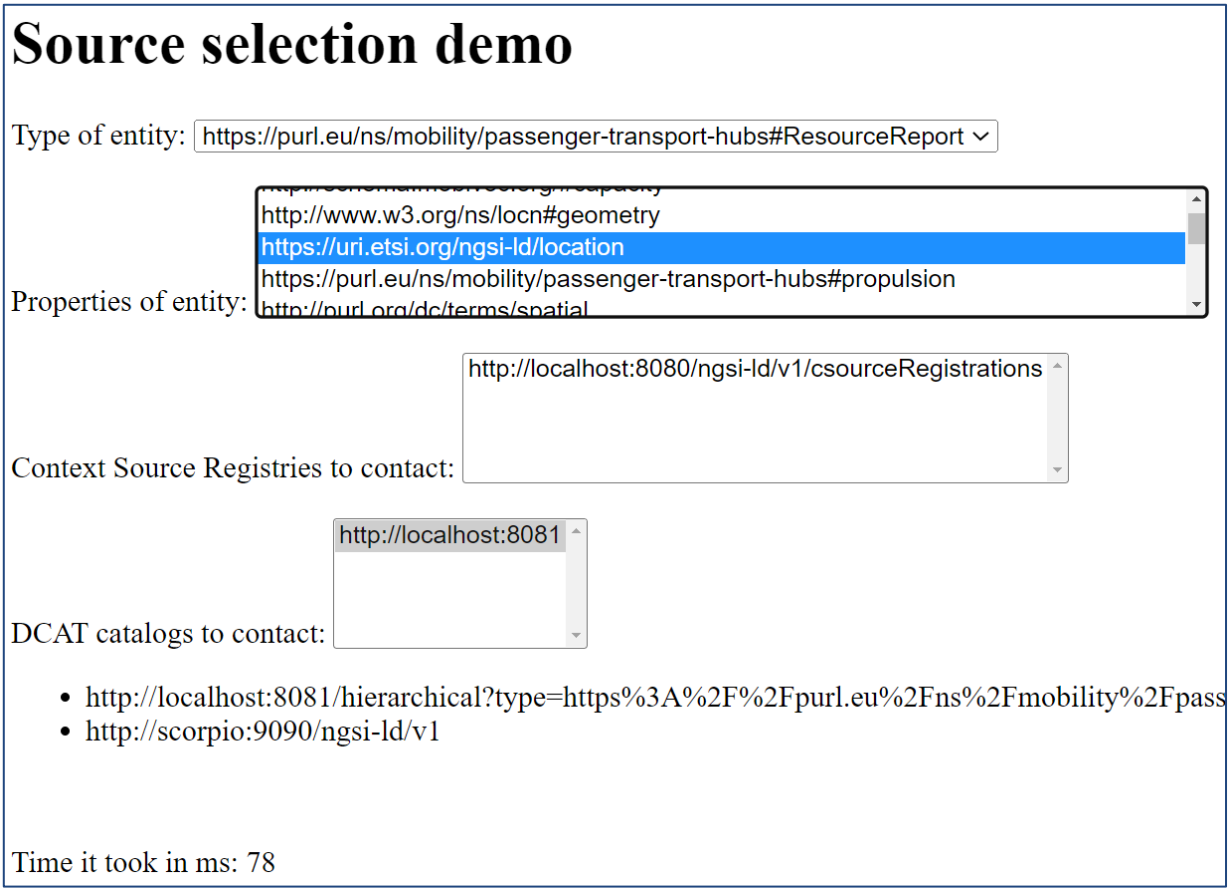

Figure 10 Source selection demonstrator where relevant sources can be retrieved based on the type of entity and properties

Figure 10 shows a screenshot of the implemented demo user interface that makes use of the source selection library to find relevant data sources for given entity types and properties. In the demo, the uses chooses an entity type (oslo-passenger :ResourceReport[1]) and the ngsild:location property. It also provides the address of a Context Source Registry and the URL of a reference DCAT catalog. With these input parameters, the source selection library is able to query both the Context Source Registry API and the DCAT catalog using the SPARQL template query shown in Drawing 1. The resulting selected sources are shown in the bottom of Figure 6. The source code of the demo is also available as part of the source selection
available as part of the source selection Github repository.[3]

2.5 Connection with open data portals

This is a feature under discussion in the use cases. Anyhow information about the connector to transfer the data from the services to open data portals based on CKAN is included.

The CKAN Extension[4]. Publishing and consuming open data is a keystone for the development of applications and the creation of an innovation ecosystem. CKAN is one of the most extended Open Data publication platforms and is becoming the de-facto standard for data publication in Europe. Moreover, CKAN is an open source platform which means it can be easily adapted and expanded.

The CKAN Extension integrates CKAN solution with the FIWARE platform, enabling the right-time context information served by a FIWARE Context Broker and to be published as a dataset resource, making it easier to be discovered and consumed as Open Data content. Additionally, this extension allows the integration with FIWARE Security in order to enrich the access control and enable explicit acceptance of data terms and conditions, usage accounting, or data monetization.

[1] "oslo-passengers" is a prefix abbreviation for the base URI https://purl.eu/mobility/passenger-transport-hubs#

3 Conclusions

Testing of federated has confirmed that high performance cannot be taken for granted:

When one single entity is involved, they have stable performance with the number of entities increasing. When querying all entities or querying all historic data of an entity, performance drops with the number of entities or the number of updates increasing.

With these results, Orion-LD seems to be the context broker that can handle the most updates per seconds regardless of the number of entities present in the broker. Furthermore, Orion-LD is also the best broker when it stands to query context API to retrieve entities. However, when retrieving historic data Orion-LD has not the best results, Scorpio and Stellio can handle more queries.

However, these results can only be indicative because the difficulty when it stands to compare brokers performance is that there is a lot of parameters that can affect the results: the size of the cluster, database configuration and database clustering.

According to this, the choice of the broker will necessarily depend on the project and on the use cases of the project.

As alternative and complement LDES with inherent fragmentation has been tested for high performance use cases. Source selection has been deployed as prototype to direct a query to the right context broker without having to retrieve entities from all brokers and hitting the performance issues described above.

At the same time it remains a key message from Greenmov that LDES and NGSI-LD are compatible when building service using linked data and especially when AI is enabled.

4 References

[1] https://www.ovhcloud.com/en/

[2] https://github.com/TREEcg/ngsi-ldes

[3] https://github.com/TREEcg/ngsi-ld-source-selection

[4] https://github.com/conwetlab/FIWARE-CKAN-Extensions