Open APIs for Open Minds

Access to short term context history using Comet

Germán Toro del Valle

Technology Specialist at Telefónica I+D (http://tid.es/en)

LinkedIn: https://www.linkedin.com/in/gtorodelvalle

Email: german.torodelvalle@telefonica.com

Twitter: @gtorodelvalle

FIWARE

Agenda

1. Introduction
2. Architecture
3. Data schemas & pre-aggregation
4. API
5. Installation
6. Configuration
7. Running
8. References

FIWARE

1. Introduction

1. Introduction

Why? (memory matters…)

- The **Context Broker** only stores the latest attribute values:
  - Event-driven action-oriented paradigm
- The **Short Time Historic** adds memory into the equation:
  - Continuous improvement paradigm
- Code name:
  - **Comet**

1. Introduction

What? (data evolution in time…)

- **Time series database**:
  - Optimized to deal with **values indexed in time**
  - **Raw** data vs. **Aggregated** data
  - Basic aggregation concepts:
    - **Range**
    - **Resolution**
    - Origin
    - Offset

1. Introduction

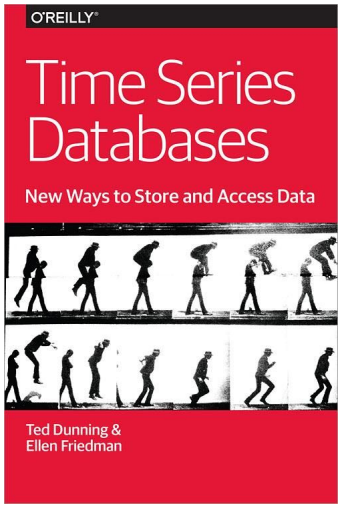
How? (the "best" solution…)

- **Best technical solution**:

InfluxDB

OpenTSDB

MAPR



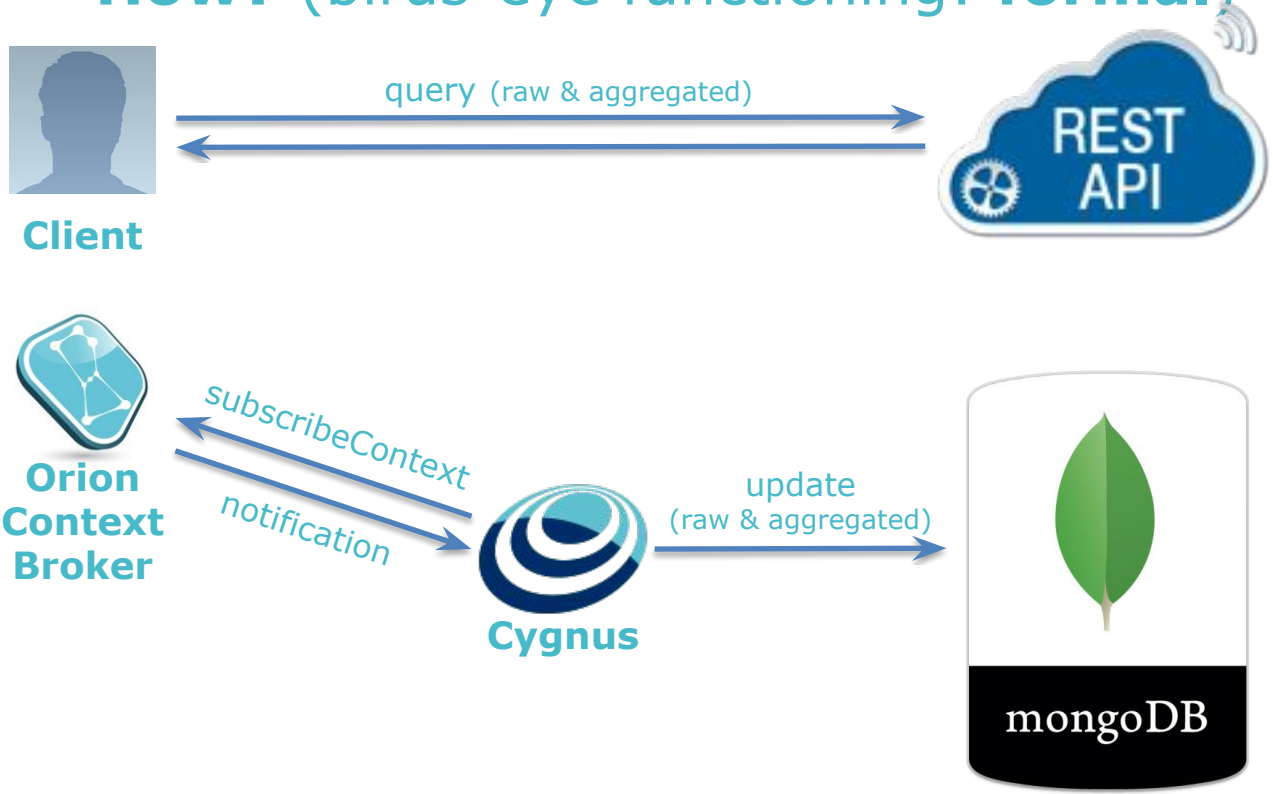
1. Introduction

How? (... is not always the "best")

- **Collateral aspects** to take into consideration:
  - **Risk**:
    - Maturity
    - Performance
    - Expertise
  - **Flexibility** (future evolution)
  - Current **inversions**

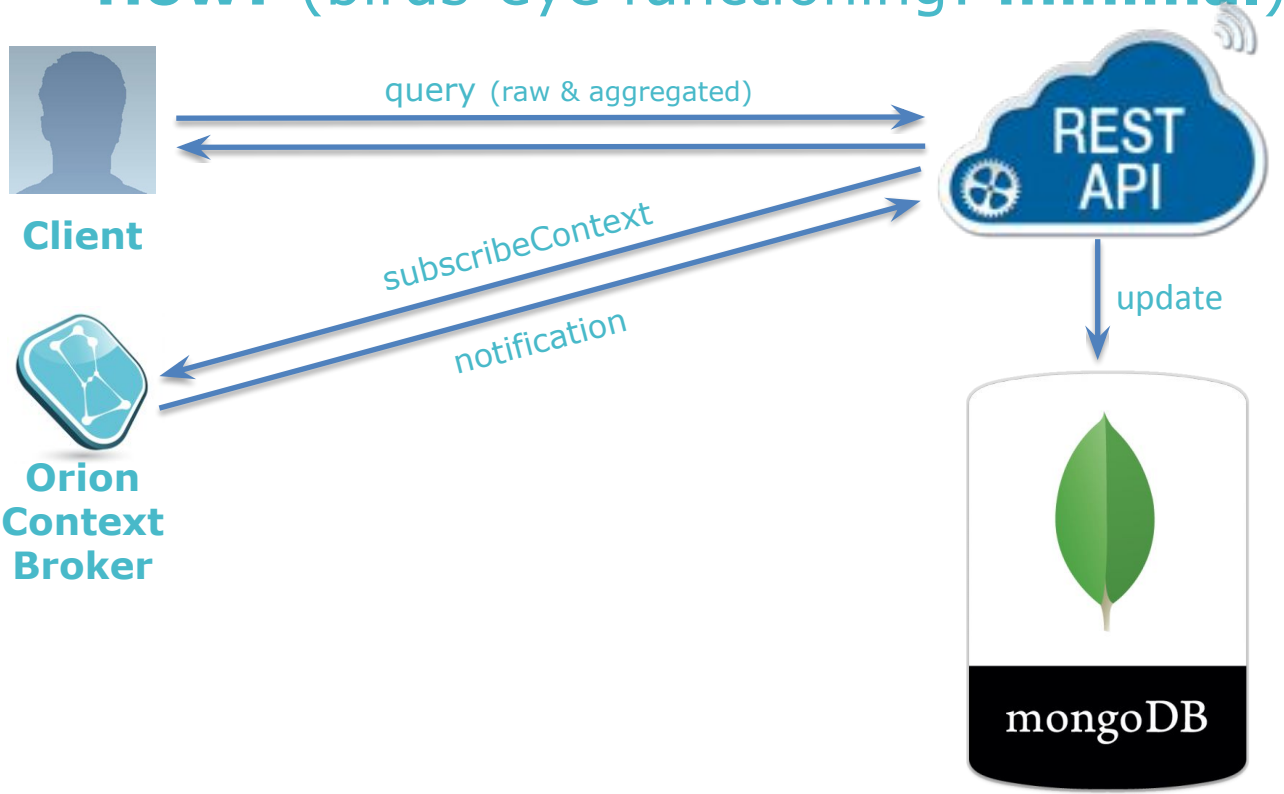
1. Introduction

How? (birds-eye functioning: **formal**)



1. Introduction

How? (birds-eye functioning: **minimal**)



2. Architecture

2. Architecture

Node application

- Database:
  - MongoDB
    - Optimized for time series functioning (specific schemas + pre-aggregation)
- Web server:
  - hapi
    - Exposing the component API
      - Data registration
      - Data retrieval
      - Data deletion
      - Log level

3. Data schemas & pre-aggregation

3. Data schemas & pre-aggregation

- Although the STH stores the evolution of (raw) data (i.e., attributes values) in time, its real power comes from the storage of **aggregated data**
- The STH should be able to respond to queries such as:
  - Give me the **maximum** temperature of this room during the last month (range) aggregated by day (resolution)
  - Give me the **mean** temperature of this room today (range) aggregated by hour or even minute (resolution)
  - Give me the **standard deviation** of the temperature of this room this last year (range) aggregated by day (resolution)
  - Give me the **number of times** the air conditioner of this room was switched on or off last Monday (range) aggregated by hour

3. Data schemas & pre-aggregation

- Calculating the aggregated data needed to respond to the previous queries from the raw data would be **highly inefficient** and **repetitive**
- Solution:
  - Data **schemas** & **pre-aggregation**

- Numeric pre-aggregation:

attribute:001 -> 333 (on Valentine's day)

```
{
  "_id": {
    "attrName": "attribute:001",
    "attrType": "Number",
    "origin": ISODate("2016-02-01T00:00:00Z"),
    "resolution": "day"
  },
  "points": [
    …,
    {
      "offset": 14,
      "samples": 3,
      "sum": 666,
      "sum2": 172494,
      "min": 111,
      "max": 333
    },
    …
  ]
}
```

3. Data schemas & pre-aggregation

- Calculating the aggregated data needed to respond to the previous queries from the raw data would be **highly inefficient** and **repetitive**
- Solution:
  - Data **schemas** & **pre-aggregation**

- Textual pre-aggregation:

attribute:001 -> "ON" (on Valentine's day)

```
{
  "_id": {
    "attrName": "attribute:001",
    "attrType": "Text",
    "origin": ISODate("2016-02-01T00:00:00Z"),
    "resolution": "day"
  },
  "points": [
    …,
    {
      "offset" : 14,
      "samples" : 3,
      "occur": {
        "ON": 2,
        "OFF": 1
      }
    },
    …
  ]
}
```

4. API

4. API

Raw data retrieval (NGSI v1): pagination

GET /STH/v1/contextEntities
/type/**<entityType>**
/id/**<entityId>**
/attributes/**<attrName>**
?**hLimit**=10
&**hOffset**=0
&dateFrom=2016-01-01T00:00:00.000Z
&dateTo=2016-03-30T23:59:59.999Z
HTTP/1.1
Host <sth-host>:<sth-port>
Fiware-Service: testservice
Fiware-ServicePath: /testservicepath
X-Auth-Token: ABC...

4. API

Raw data retrieval (NGSI v1): pagination

GET /STH/v1/contextEntities
/type/**<entityType>**
/id/**<entityId>**
/attributes/**<attrName>**
?**lastN**=10
&dateFrom=2016-01-01T00:00:00.000Z
&dateTo=2016-03-30T23:59:59.999Z
HTTP/1.1
Host <sth-host>:<sth-port>
Fiware-Service: testservice
Fiware-ServicePath: /testservicepath
X-Auth-Token: ABC...

4. API

Raw data retrieval (NGSI v1): response


```
{
  "contextResponses": [
    {
      "contextElement": {
        "attributes": [
          {
            "name": "attribute:001",
            "values": [
          {
            "recvTime": "2016-02-14T13:43:33.306Z",
            "attrValue": "111"
          },
          {
            "recvTime": "2016-02-14T16:55:23.617Z",
            "attrValue": "222"
          },
          ...
```

4. API

Raw data retrieval (NGSI v1): response


```
            ...
            {
              "recvTime": "2016-02-14T21:35:11.339Z",
              "attrValue": "333"
            }
          ]
          "id": "Entity:001",
          "isPattern": false,
          "isPattern": "Entity",
        },
        "statusCode": {
          "code": "200",
          "reasonPhrase": "OK"
        }
      }
    ]
}
```

4. API

Aggregated data retrieval (NGSI v1)

GET /STH/v1/contextEntities
/type/**<entityType>**
/id/**<entityId>**
/attributes/**<attrName>**
?**aggrMethod**=sum
&**aggrPeriod**=day
&dateFrom=2016-01-01T00:00:00.000Z
&dateTo=2016-03-30T23:59:59.999Z
HTTP/1.1
Host <sth-host>:<sth-port>
Fiware-Service: testservice
Fiware-ServicePath: /testservicepath
X-Auth-Token: ABC...

4. API

Aggregated data retrieval (NGSI v1)


```
{
  "contextResponses": [
    {
      "contextElement": {
        "attributes": [
          {
            "name": "attribute:001",
            "values": [
              {
                "_id": {
                  "attrName": "attribute:001",
                  "origin": "2016-02-14T00:00:00.000Z",
                  "resolution": "day"
                },
                "points": [
                  {
                    "offset": 14,
                    "samples": 3,
                    "sum": 666
                  }
                ]
              }
              ...
```

4. API

Aggregated data retrieval (NGSI v1)


```
            ...
            ]
          }
        ],
        "id": "Entity:001",
        "isPattern": false,
        "type": "Entity"
      },
      "statusCode": {
        "code": "200",
        "reasonPhrase": "OK"
      }
    }
  ]
}
```

4. API

Attribute data removal (NGSI v1)

DELETE /STH/v1/contextEntities
/type/**<entityType>**
/id/**<entityId>**
/attributes/**<attrName>**
HTTP/1.1
Host <sth-host>:<sth-port>
Fiware-Service: testservice
Fiware-ServicePath: /testservicepath
X-Auth-Token: ABC...

4. API

Entity data removal (NGSI v1)

DELETE /STH/v1/contextEntities
/type/**<entityType>**
/id/**<entityId>**
HTTP/1.1
Host <sth-host>:<sth-port>
Fiware-Service: testservice
Fiware-ServicePath: /testservicepath
X-Auth-Token: ABC...

4. API

Service path data removal (NGSI v1)

DELETE /STH/v1/contextEntities
HTTP/1.1
Host <sth-host>:<sth-port>
Fiware-Service: testservice
Fiware-ServicePath: /testservicepath
X-Auth-Token: ABC...

4. API

Log level retrieval (NGSI v1)

GET /admin/log
HTTP/1.1
Host <sth-host>:<sth-port>
Fiware-Service: testservice
Fiware-ServicePath: /testservicepath
X-Auth-Token: ABC...

4. API

Log level update (NGSI v1)

PUT /admin/log
?**level**=debug
HTTP/1.1
Host <sth-host>:<sth-port>
Fiware-Service: testservice
Fiware-ServicePath: /testservicepath
X-Auth-Token: ABC...

5. Installation

5. Installation

Nice and simple :)

1. From Github:

```
> git clone
https://github.com/telefonicaid/fiware-sth
-comet
```

```
> npm install
```

2. From Docker:

```
> docker pull fiware/sth-comet
```

```
> docker run -t -i fiware/sth-comet
/bin/bash
```

6. Configuration

6. Configuration

Via environment variables and config.js

- The STH can be configured via enviroment variables or via the `config.js` file
- Both are equivalent although environment variables take precedence over the `config.js` file

6. Configuration

Via environment variables and config.js

- Environment variables:
  - `STH_HOST`: The host where the STH server will be started. Optional. Default value: "localhost".
  - `STH_PORT`: The port where the STH server will be listening. Optional. Default value: "8666".
  - `FILTER_OUT_EMPTY`: A flag indicating if the empty results should be removed from the response. Optional. Default value: "true".
  - `DEFAULT_SERVICE`: The service to be used if not sent in the Orion Context Broker notifications. Optional. Default value: "testservice".
  - `DEFAULT_SERVICE_PATH`: The service path to be used if not sent in the Orion Context Broker notifications. Optional. Default value: "/testservicepath".

6. Configuration

Via environment variables and config.js

- Environment variables:
  - `DATA_MODEL`: The STH component supports 3 alternative data models when storing the raw and aggregated data into the database: 1) one collection per attribute, 2) one collection per entity and 3) one collection per service path. The possible values are: "collection-per-attribute", "collection-per-entity" and "collection-per-service-path" respectively. Default value: "collection-per-entity".
  - `DB_USERNAME`: The username to use for the database connection. Optional. Default value: "".
  - `DB_PASSWORD`: The password to use for the database connection. Optional. Default value: "".
  - `DB_URI`: The URI to use for the database connection. This does not include the 'mongo://' protocol part (see a couple of examples below). Optional. Default value: "localhost:27017".

6. Configuration

Via environment variables and config.js

- Environment variables:
  - `REPLICA_SET`: The name of the replica set to connect to, if any. Default value: "".
  - `DB_PREFIX`: The prefix to be added to the service for the creation of the databases. Optional. Default value: "sth_".
  - `COLLECTION_PREFIX`: The prefix to be added to the collections in the databases. Optional. Default value: "sth_".
  - `POOL_SIZE`: The default MongoDB pool size of database connections. Optional. Default value: "5".
  - `WRITE_CONCERN`: The write concern policy to apply when writing data to the MongoDB database. Default value: "1".
  - `SHOULD_STORE`: Flag indicating if the raw and/or aggregated data should be persisted. Valid values are: "only-raw", "only-aggregated" and "both". Default value: "both".

6. Configuration

Via environment variables and config.js

- Environment variables:
  - `REPLICA_SET`: The name of the replica set to connect to, if any. Default value: "".
  - `DB_PREFIX`: The prefix to be added to the service for the creation of the databases. Optional. Default value: "sth_".
  - `COLLECTION_PREFIX`: The prefix to be added to the collections in the databases. Optional. Default value: "sth_".
  - `POOL_SIZE`: The default MongoDB pool size of database connections. Optional. Default value: "5".
  - `WRITE_CONCERN`: The write concern policy to apply when writing data to the MongoDB database. Default value: "1".
  - `SHOULD_STORE`: Flag indicating if the raw and/or aggregated data should be persisted. Valid values are: "only-raw", "only-aggregated" and "both". Default value: "both".

6. Configuration

Via environment variables and config.js

- Environment variables:
  - `TRUNCATION_EXPIRE_AFTER_SECONDS` : Data from the raw and aggregated data collections will be removed if older than the value specified in seconds. In case of raw data the reference time is the one stored in the recvTime property whereas in the case of the aggregated data the reference of time is the one stored in the _id.origin property. Set the value to 0 not to apply this time-based truncation policy. Default value: "0".

6. Configuration

Via environment variables and config.js

- Environment variables:
  - `TRUNCATION_SIZE`: The oldest raw data (according to insertion time) will be removed if the size of the raw data collection gets bigger than the value specified in bytes. Set the value to 0 not to apply this truncation policy. Take into consideration than the "size" configuration parameter is mandatory in case size collection truncation is desired as required by MongoDB. Default value: "0". Notice that this configuration parameter does not affect the aggregated data collections since MongoDB does not currently support updating documents in capped collections which increase the size of the documents. Notice also that in case of the raw data, the size-based truncation policy takes precedence over the TTL one. Default value: "0".

6. Configuration

Via environment variables and config.js

- Environment variables:
  - `TRUNCATION_MAX`: The oldest raw data (according to insertion time) will be removed if the number of documents in the raw data collections goes beyond the specified value. Set the value to 0 not to apply this truncation policy. Notice that this configuration parameter does not affect the aggregated data collections since MongoDB does not currently support updating documents in capped collections which increase the size of the documents. Default value: "0".
  - `IGNORE_BLANK_SPACES` : Attribute values to one or more blank spaces should be ignored and not processed either as raw data or for the aggregated computations. Default value: "true".

6. Configuration

Via environment variables and config.js

- Environment variables:
  - `NAME_MAPPING`: The mapping mechanism provided consists on a 1 to 1 mapping between services, service paths, entity ids and types and attribute names via a mapping configuration file applied for the generation of the collection names. The `NAME_MAPPING` value is an object including 2 properties: 1) `enabled` which, as its name states, enables or disables the mapping mechanism (default value: "true") and 2) `configFile` which is a relative or absolute path to the mapping configuration file (default value: "./name-mapping.json").

6. Configuration

Via environment variables and config.js

- Environment variables:
  - `NAME_ENCODING`: The encoding criteria consists on: 1) encode the forbidden characters using an escaping character (`x`) and the numerical Unicode code for each character (for instance, the `$` character will be encoded as `x0024`), 2) database and collection names already using the above encoding must be escaped prepending another `x` (for instance, the text `x002a` will be encoded as `xx002a`), 3) the uppercase characters included in database names will be encoded using the mechanism stated in 1), 4) collection names starting with `system.` will be encoded as `xsystem.` and 5) the name separator character (`xffff`) is not decoded. It is important to note that the encoding mechanism also applies in case a mapping has been accomplished over the resulting or new element. Default value: "true".

6. Configuration

Via environment variables and config.js

- Environment variables:
  - `LOGOPS_LEVEL`: The log level to use. Possible values are: "DEBUG", "INFO", "WARN", "ERROR" and "FATAL". Default value: "INFO".
  - `LOGOPS_FORMAT`: The log format to use. Possible values are: "json" (writes logs as JSON), "dev" (for development, used when the `NODE_ENV` variable is set to "development"). Default value: "json".
  - `PROOF_OF_LIFE_INTERVAL`: The time in seconds between proof of life logging messages informing that the server is up and running normally. Default value: "60".

7. Running

7. Running

Even nicer and simpler :)

```
fiware-sth-comet> ./bin/sth
```

8. References

8. References

1. **Github repository**:
   - https://github.com/telefonicaid/fiware-sth-comet

2. **Documentation at ReadTheDocs:**
   - https://fiware-sth-comet.readthedocs.io/en/latest/

Thank you!

http://fiware.org
Follow @FIWARE on Twitter