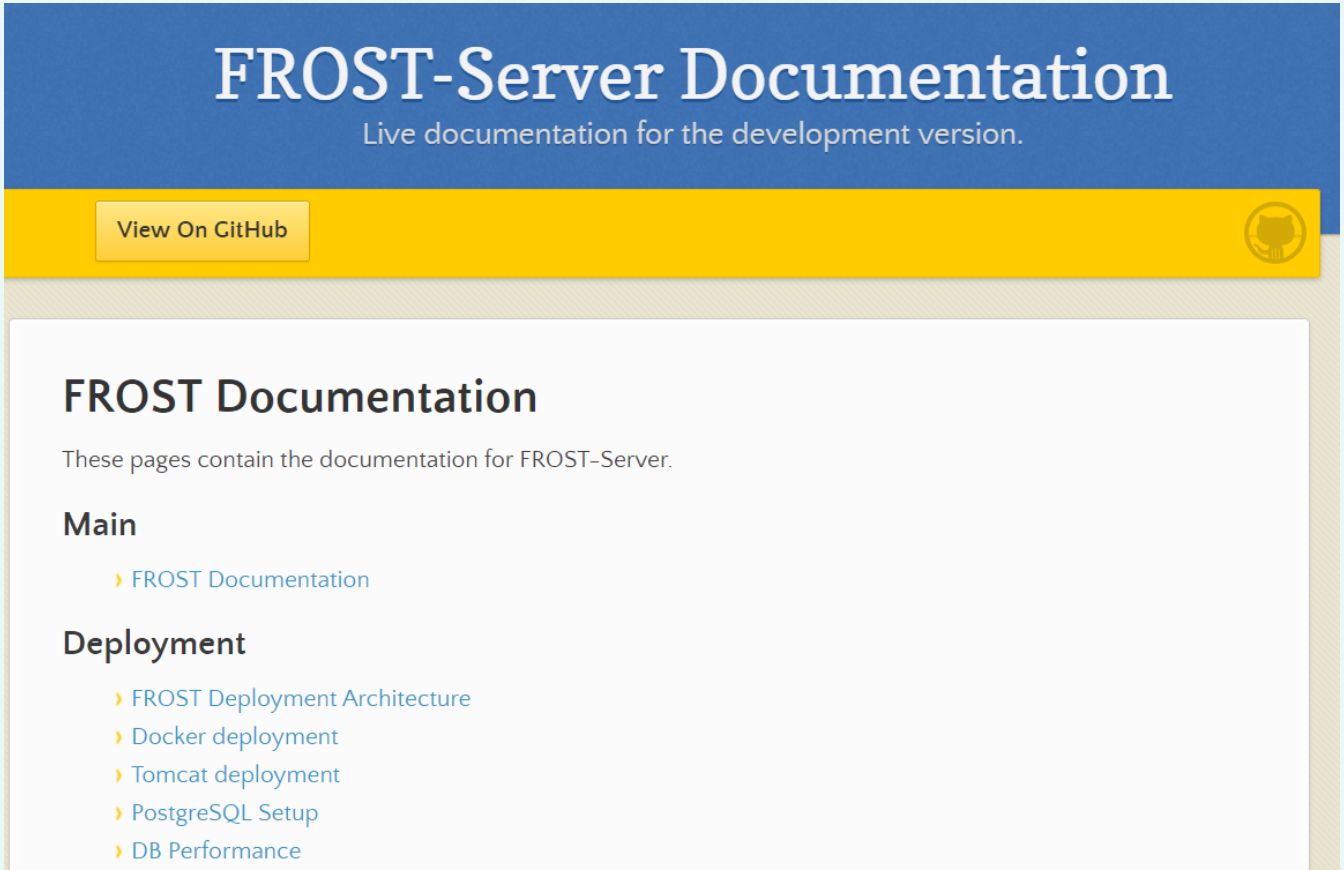
FROST Deployment

deda.next

Luca Giovannini

https://fraunhoferiosb.github.io/FROST-Server/



https://github.com/FraunhoferIOSB/FROST-Server

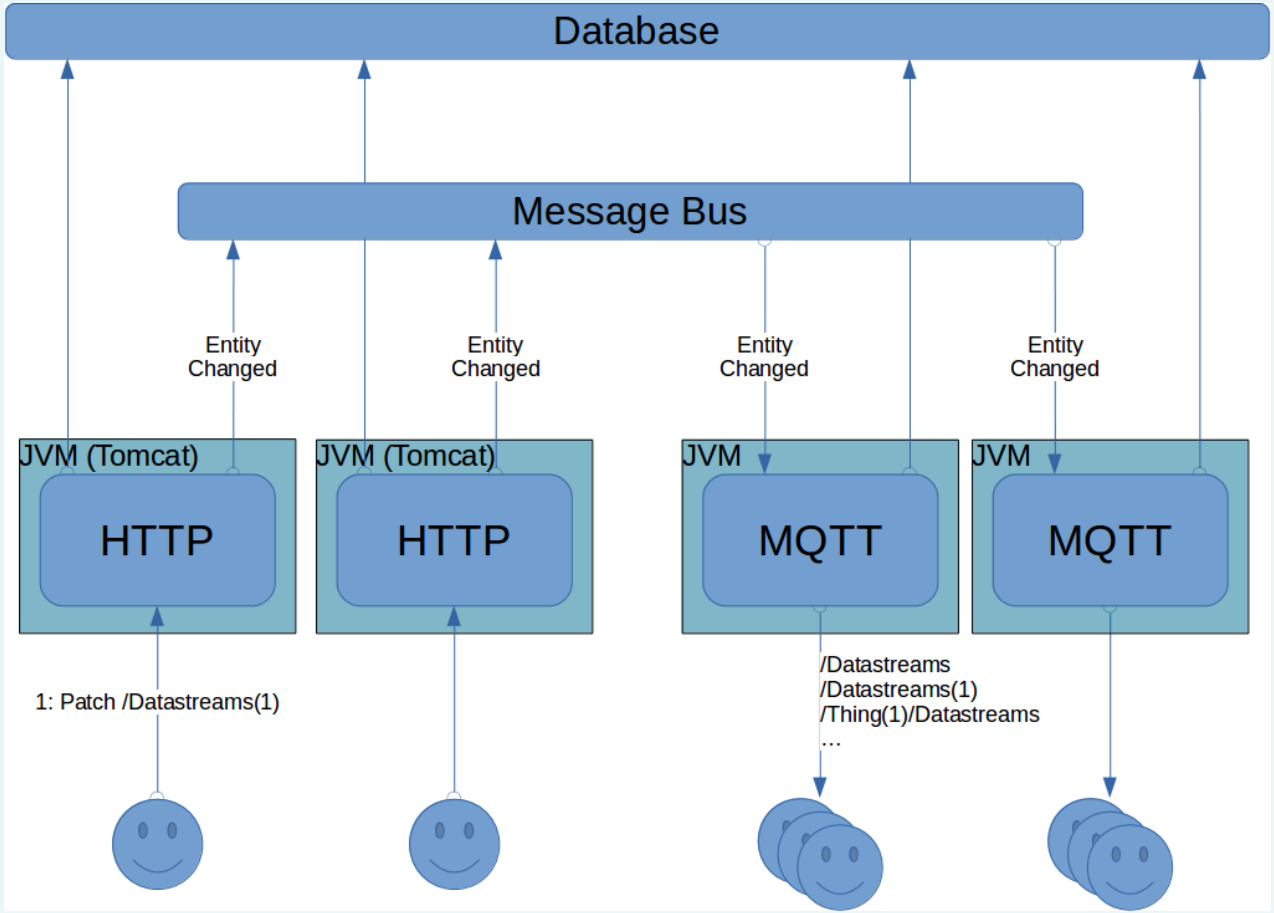
A Server implementation of the OGC SensorThings API. The **FR**aunhofer **O**pensource **S**ensor**T**hings-Server is the first complete, open-source official reference implementation of the OGC SensorThings API Part 1: Sensing 1.0. It also implements OGC SensorThings API Part 1: Sensing 1.1 and OGC SensorThings API Part 2: Tasking Core 1.1.

Downloading

Instead of compiling the server yourself, you can also download pre-built war and jar files from:

- FROST-Server.MQTTP
- FROST-Server.HTTP
- FROST-Server.MQTT

Or you can use Docker.


Database
Message Bus
Entity Changed
Entity Changed
Entity Changed
Entity Changed
JVM (Tomcat)
HTTP
JVM (Tomcat)
HTTP
JVM
MQTT
JVM
MQTT
1: Patch /Datastreams(1)
/Datastreams
/Datastreams(1)
/Thing(1)/Datastreams
...

https://github.com/FraunhoferIOSB/FROST-Server

Step 01: Deploy WAR su Tomcat

- Install Tomcat
- Download FROST MQTTP war from https://repo1.maven.org/maven2/de/fraunhofer/iosb/ilt/FROST-Server/FROST-Server.MQTTP/
- Get postgresql.jar and postgis-jdbc.jar libraries and put them in *tomcat/lib* folder

Tested with	version
Apache Tomcat	9.0.24
FROST MQTTP	1.13.0
postgresql.jar	42.2.16
postgis-jdbc.jar	2.4.0

Step 02: Configure PostgreSQL DB

deda.next x EDIAQI

- Create a new *user = sensorthings* and *pwd = ChangeMe*
- Create a new database with *name = sensorthings* and *owner = sensorthings*
- CREATE EXTENSION "uuid-ossp"
- CREATE EXTENSION "postgis"
- CREATE EXTENSION "timescaledb" (optional)
- (install extension libraries if not already available)

Tested with	version
postgreSQL	12.11
postgis	3.0.3
uuid-ossp	1.1
timescaledb	2.3.1

Step 03: Configure FROST Server

- Using configuration file: *../tomcat/webapps/FROST-Server/META-INF/context.xml*
  - Set the JDBC resource with url, username and password;
  - Set «serviceRootUrl», «defaultTop» and «maxTop» parameter, if needed;
  - Check the «persistenceManagerImplementationClass», for ID type
  - Check the «idGenerationMode», for ID management

Tested with	values
serviceRootUrl	https://iot.comune.fe.it/FROST-Server
defaultTop / maxTop	5000 / 10000
persistenceManagerImplementationClass	PostgresPersistenceManagerString
idGenerationMode	ServerAndClientGenerated

Step 04: Create DB schema

- Go to <FROST-ServerURL> / DatabaseStatus and «do update»

Servlet DatabaseStatus at /FROST-Server

Checking Database status.

Do Update

Back...

de.fraunhofer.iosb.ilt.frostserver.persistence.pgjooq.imp.PostgresPersistenceManagerString

```
-- *********************************************************************
-- Update Database Script
-- *********************************************************************
-- Change Log: liquibase/tablesString.xml
-- Ran at: 7/20/23, 11:46 PM
-- Against: sensorthings@jdbc:postgresql://localhost:5432/sensorthings
-- Liquibase version: 3.10.2
-- *********************************************************************

SET SEARCH_PATH TO public;

-- Lock Database
UPDATE public.databasechangeloglock SET LOCKED = TRUE, LOCKEDBY = '38008_iot.comune.fe.it (10.50.1.8)', LOCKGRANTED = '2023-07-20 23:46:51.

SET SEARCH_PATH TO public;

SET SEARCH_PATH TO public;

-- Release Database Lock
SET SEARCH_PATH TO public;
```

Step 04 B: Fix bug in ver1.13.0

deda.next x EDIAQI

```
-- FUNCTION: public.datastreams_update_delete()

-- DROP FUNCTION public.datastreams_update_delete();

CREATE OR REPLACE FUNCTION public.datastreams_update_delete()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
declare
"DS_ROW" "DATASTREAMS"%rowtype;
"MDS_ROW" "MULTI_DATASTREAMS"%rowtype;
begin

if (OLD."DATASTREAM_ID" is not null)
then
    select * into "DS_ROW" from "DATASTREAMS" where "DATASTREAMS"."ID"=OLD."DATASTREAM_ID";

    if (OLD."PHENOMENON_TIME_START" = "DS_ROW"."PHENOMENON_TIME_START"
        or coalesce(OLD."PHENOMENON_TIME_END", OLD."PHENOMENON_TIME_START") = "DS_ROW"."PHENOMENON_TIME_END")
    then
        update "DATASTREAMS"
            set "PHENOMENON_TIME_START" = (select min("PHENOMENON_TIME_START") from "OBSERVATIONS" where "OBSERVATIONS"."DATASTREAM_ID" = "DS_ROW"."ID")
            where "DATASTREAMS"."ID" = "DS_ROW"."ID";
        update "DATASTREAMS"
            set "PHENOMENON_TIME_END" = (select max(coalesce("PHENOMENON_TIME_END", "PHENOMENON_TIME_START")) from "OBSERVATIONS" where "OBSERVATIONS"."DATASTREAM_ID" = "DS_ROW"."ID")
            where "DATASTREAMS"."ID" = "DS_ROW"."ID";
    end if;

    if (OLD."RESULT_TIME" = "DS_ROW"."RESULT_TIME_START")
    then
        update "DATASTREAMS"
            set "RESULT_TIME_START" = (select min("RESULT_TIME") from "OBSERVATIONS" where "OBSERVATIONS"."DATASTREAM_ID" = "DS_ROW"."ID")
            where "DATASTREAMS"."ID" = "DS_ROW"."ID";
    end if;
    if (OLD."RESULT_TIME" = "DS_ROW"."RESULT_TIME_END")
    then
        update "DATASTREAMS"
            set "RESULT_TIME_END" = (select max("RESULT_TIME") from "OBSERVATIONS" where "OBSERVATIONS"."DATASTREAM_ID" = "DS_ROW"."ID")
            where "DATASTREAMS"."ID" = "DS_ROW"."ID";
    end if;

end if;

if (OLD."MULTI_DATASTREAM_ID" is not null)
then
    select * into "MDS_ROW" from "MULTI_DATASTREAMS" where "MULTI_DATASTREAMS"."ID"=OLD."MULTI_DATASTREAM_ID";

    if (OLD."PHENOMENON_TIME_START" = "DS_ROW"."PHENOMENON_TIME_START"
        or coalesce(OLD."PHENOMENON_TIME_END", OLD."PHENOMENON_TIME_START") = "MDS_ROW"."PHENOMENON_TIME_END")
    then
        update "MULTI_DATASTREAMS"
            set "PHENOMENON_TIME_START" = (select min("PHENOMENON_TIME_START") from "OBSERVATIONS" where "OBSERVATIONS"."MULTI_DATASTREAM_ID" = "MDS_ROW"."ID")
            where "MULTI_DATASTREAMS"."ID" = "MDS_ROW"."ID";
        update "MULTI_DATASTREAMS"
            set "PHENOMENON_TIME_END" = (select max(coalesce("PHENOMENON_TIME_END", "PHENOMENON_TIME_START")) from "OBSERVATIONS" where "OBSERVATIONS"."MULTI_DATASTREAM_ID" = "MDS_ROW"."ID")
            where "DATASTREAMS"."ID" = "MDS_ROW"."ID";
    end if;

    if (OLD."RESULT_TIME" = "MDS_ROW"."RESULT_TIME_START")
    then
        update "MULTI_DATASTREAMS"
            set "RESULT_TIME_START" = (select min("RESULT_TIME") from "OBSERVATIONS" where "OBSERVATIONS"."MULTI_DATASTREAM_ID" = "MDS_ROW"."ID")
            where "MULTI_DATASTREAMS"."ID" = "MDS_ROW"."ID";
    end if;
    if (OLD."RESULT_TIME" = "MDS_ROW"."RESULT_TIME_END")
    then
        update "MULTI_DATASTREAMS"
            set "RESULT_TIME_END" = (select max("RESULT_TIME") from "OBSERVATIONS" where "OBSERVATIONS"."MULTI_DATASTREAM_ID" = "MDS_ROW"."ID")
            where "MULTI_DATASTREAMS"."ID" = "MDS_ROW"."ID";
    end if;

end if;

return NULL;
end
$BODY$;

ALTER FUNCTION public.datastreams_update_delete()
    OWNER TO sensorthings;
```

Trigger function datastreams_update_delete()

Step 04 C: Activate Timescale

```
ALTER TABLE public."OBSERVATIONS" drop constraint "OBSERVATIONS_PKEY";
SELECT create_hypertable('"OBSERVATIONS"','PHENOMENON_TIME_START');
CREATE INDEX "OBSERVATIONS_ID" ON public."OBSERVATIONS"("ID");
```

Remark 1:

Timescale creates hypertables «splitting» tables along their temporal dimention, but cannot guarantee cross-tables uniqueness constraints;

The solution is to remove the uniqueness constraint from the observation ID but keeping the index (the sequential ID generator will guarantee uniqueness nonetheless);

Remark 2:

Transformation of table in hypertables is best done when the OBSERVATIONS table is still empty;

Step 05: Configure proxy and security

- If needed:
  - configure URL proxies
  - and SSL certificates

WARNING: avoid providing external access to «database status and update» page

Tested with	implementation
Proxy server	Apache Webserver
SSL certificate	Let's Encrypt

Improvements in new 2.x versions

deda.next x EDIAQI

Extendable Data Model and pluggable APIs

FROST-Server implements the SensorThings API data model and API, but is not limited to these. Using plugins the data model can be extended or even completely replaced depending on your specific requirements. APIs and result formats can also be added using plugins. By default, FROST-Server comes with experimental plugins for the OData 4.0 and 4.01 APIs and for CSV and GeoJSON result formats.

Fine-grained Authorisation

Authorisation rules can either be simple with Read, Create, Update and Delete on a service level, or they can be very fine-grained with, for instance, certain users being able to read or create only Observations in Datastreams of certain Things.