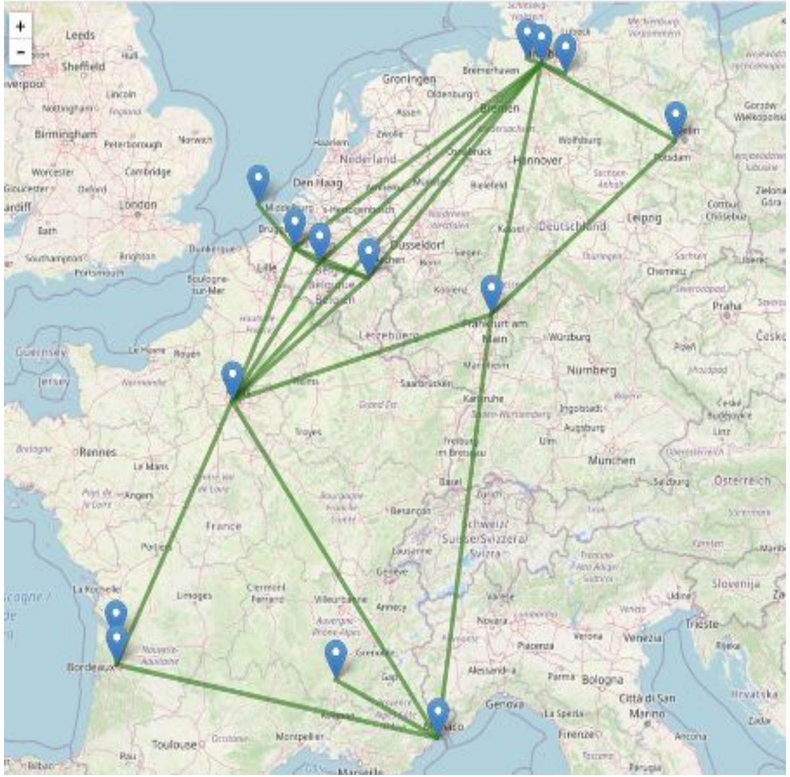
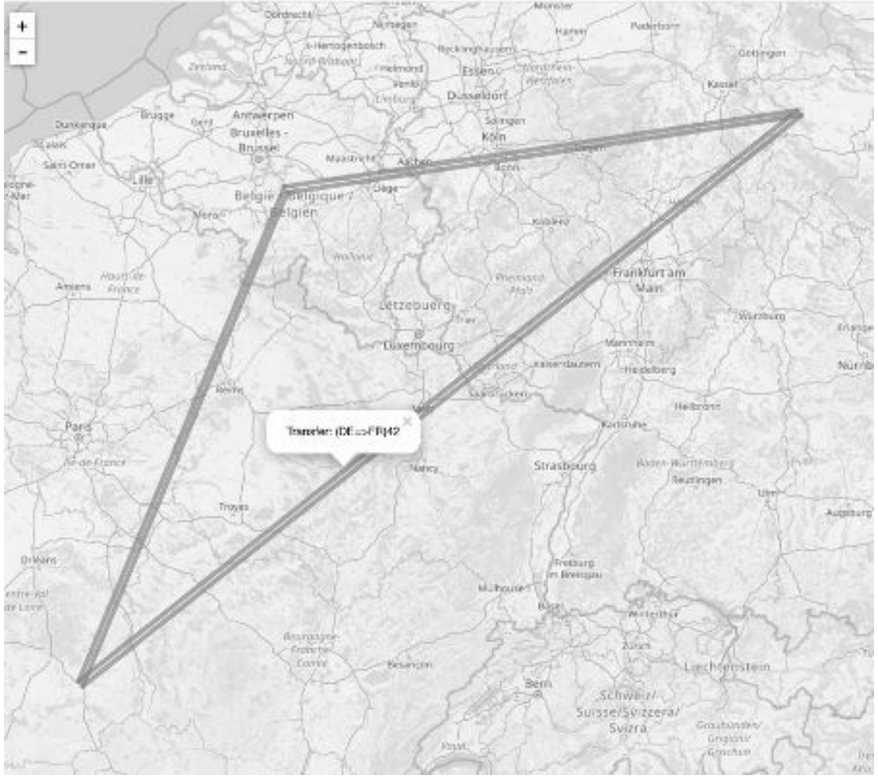
Power Grid Simulation
A Case for Cloudnative Streaming

Version 1.0
(prep for meetup)

Power Grid Simulation:
5 plants, 9 stations in 3 regions





Primary Goals:

- Enriche raw data with Geo-Context
- Aggregate data based on Geo-Context
- Provide results in GeoJson format

Data Flow

KSQL App

Queries
- station metadata linked to segment via station ID

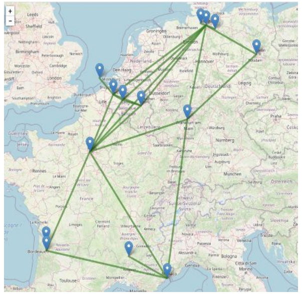
Grid Simulation

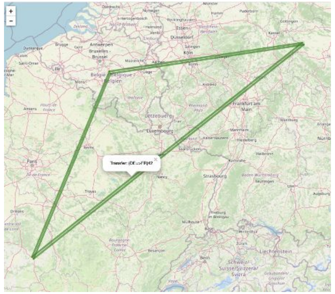
- network structure from file
- random flow values for each segment
- station metadata linked to segment via station ID

topics

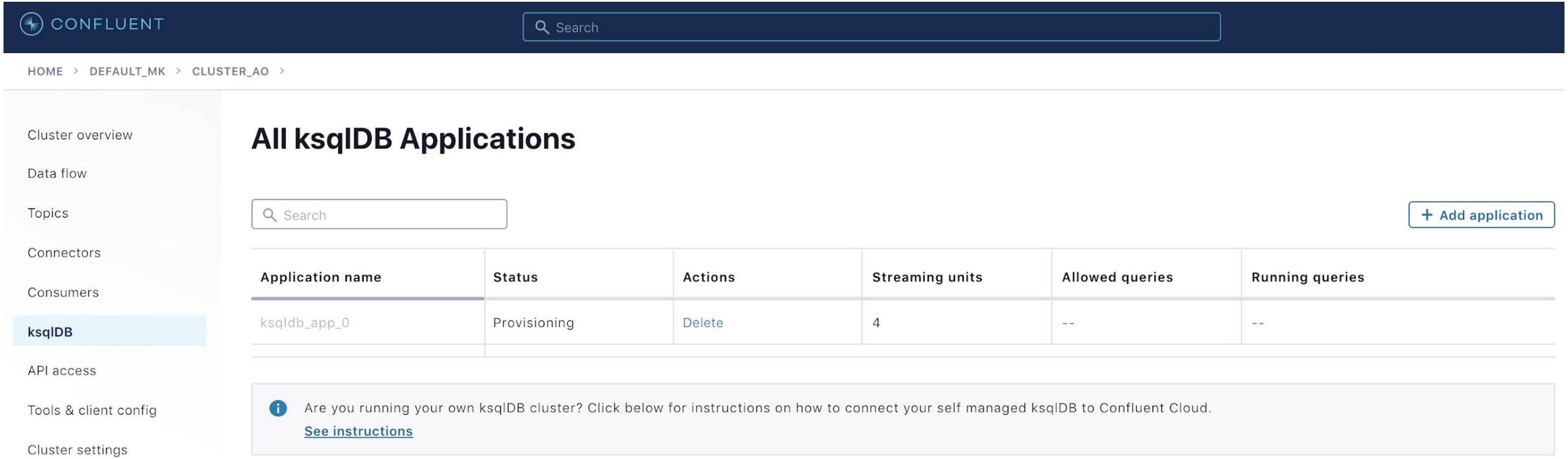
Kafka

Client App





Create a KSQL Instance



Details of KSQL

```
ccloud ksql app list
```

```
✘ mkampf@MirkoKapfMBP152  ~  ccloud ksql app list

      Id       |     Name      | Topic Prefix |   Kafka    | Storage |                         Endpoint                          | Status
+--------------+---------------+--------------+------------+---------+-----------------------------------------------------------+--------+
  lksqlc-28pgq | ksqldb_app_0  | pksqlc-57w02 | lkc-jwgvw  |     500 | https://pksqlc-57w02.us-east1.gcp.confluent.cloud:443 | UP
```

Define ACLs for KSQL

```
ccloud service-account list
```

```
✘ mkampf@MirkoKapfMBP152  ~  ccloud service-account list
   Id    |         Name         |              Description
+--------+----------------------+------------------------------------+
  122239 | KPING_Demo2          | KPING_Demo2
  138337 | KSQL.lksqlc-28pgq    | SA for KSQL w/ ID lksqlc-28pgq
         |                      | and Name ksqldb_app_0
```

```
ccloud kafka acl create --allow --service-account  138337 --operation READ
--operation WRITE --operation CREATE --topic '*' --cluster lkc-jwgvw
```

KSQL App needs privileges to access topics:

```
# IDs used in this setup:
#
KSQLDB-APP    : lksqlc-28pgq
KAFKA-CLUSTER : lkc-jwgvw

#############################################################################################
# How to define acls for the KSQL-DB app?
##############################################################################################

ccloud ksql app configure-acls lksqlc-28pgq grid-link-flow-data --cluster lkc-jwgvw
ccloud ksql app configure-acls lksqlc-28pgq grid-plants --cluster lkc-jwgvw
ccloud ksql app configure-acls lksqlc-28pgq grid-regions --cluster lkc-jwgvw
ccloud ksql app configure-acls lksqlc-28pgq grid-stations --cluster lkc-jwgvw
ccloud ksql app configure-acls lksqlc-28pgq grid-static-links --cluster lkc-jwgvw

ccloud kafka acl create --allow --service-account 138337 --operation READ --operation WRITE --operation CREATE --topic '*' --cluster lkc-jwgvw
```

Create topics (1)

Which cluster can I use?

```
✘ mkampf@MirkoKapfMBP152 ~ ccloud kafka cluster list
      Id      |    Name     | Type  | Provider |  Region   | Availability | Status
+-------------+-------------+-------+----------+-----------+--------------+--------+
   lkc-jwgvw  | cluster_AO  | BASIC | gcp      | us-east1  | single-zone  | UP
```

Which topics exist?

```
mkampf@MirkoKapfMBP152 ~ ccloud kafka topic list --cluster lkc-jwgvw
                                    Name
+--------------------------------------------------------------------------------+
  MY_TEMP_TABLE
  RESULT_TABLE
  _confluent-ksql-pksqlc-57w02_command_topic
  _confluent-ksql-pksqlc-mvvy2_command_topic
  _confluent-ksql-pksqlc-mvvy2transient_5145757274248306066_1603991824641-KsqlTopic-Reduce-changelog
```

Create and Delete topics (2)

```
#############################################################################################
#  Which topics are used for simulation data?
#############################################################################################

# Clean up
ccloud kafka topic delete grid-regions --cluster lkc-jwgvw
ccloud kafka topic delete grid-stations --cluster lkc-jwgvw
ccloud kafka topic delete grid-plants --cluster lkc-jwgvw
ccloud kafka topic delete grid-link-flow-data --cluster lkc-jwgvw
ccloud kafka topic delete grid-static-links --cluster lkc-jwgvw

# Create
ccloud kafka topic create grid-regions --cluster lkc-jwgvw --partitions 1
ccloud kafka topic create grid-stations --cluster lkc-jwgvw --partitions 1
ccloud kafka topic create grid-plants --cluster lkc-jwgvw --partitions 1
ccloud kafka topic create grid-link-flow-data --cluster lkc-jwgvw --partitions 1
ccloud kafka topic create grid-grid-static-links --cluster lkc-jwgvw --partitions 1
```

Run simulation tool to generate context data …

```java
public static void main(String[] args) throws Exception {

    powerPlants = CSVFileRepository.getPowerPlantsFromRepository();
    stations = CSVFileRepository.getStationsFromRepository();
    regions = CSVFileRepository.getRegionsFromRepository();
    segments = GridDataProvider.getSegments();

    /**
     *   This is the network layer which represents the reality (simulation setup).
     */
    GeoJSONExporter.generateGrid();

    /**
     *   This is the network layer which represents the analysis results.
     */
    GeoJSONExporter.generateRegionLinks();

    /**
     *   Provide data via topics in Confluent cloud.
     */
    TopicGroupTool.configureProducer(appID);

    TopicGroupTool.storeRegionContextData( regions );
    TopicGroupTool.storeStationContextData( stations );
    TopicGroupTool.storePlantContextData( powerPlants );
    TopicGroupTool.storeLinkContextData( gridLinks );

    System.out.println( "> Now we have to define the streams and tables in KSQLDB. " );
```

Define Tables and Streams over Input Data

```sql
-- table with location data about regions (region centers):
CREATE TABLE regionTable (
     rowkey VARCHAR PRIMARY KEY,
     type STRING,
     id INTEGER,
     name STRING,
     lat DOUBLE,
     lon DOUBLE,
     country STRING,
     production DOUBLE,
     consumption DOUBLE,
     imports DOUBLE,
     exports DOUBLE
) WITH (
     KAFKA_TOPIC = 'grid-regions',
     VALUE_FORMAT = 'JSON'
);
```

```sql
-- table with location data about powerPlants:
CREATE TABLE plantsTable (
     rowkey VARCHAR PRIMARY KEY,
     type STRING,
     id STRING,
     name STRING,
     lat DOUBLE,
     lon DOUBLE,
     country STRING
) WITH (
     KAFKA_TOPIC = 'grid-plants',
     VALUE_FORMAT = 'JSON'
);
```

```sql
-- stream with data points from raw_dp topic
CREATE STREAM gridLinkFlowDataStream (
     rowkey VARCHAR KEY,
     ts BIGINT,
     flow DOUBLE
) WITH (
     KAFKA_TOPIC = 'grid-link-flow-data',
     VALUE_FORMAT = 'JSON'
);
```

```sql
CREATE TABLE stationTable (
     rowkey VARCHAR PRIMARY KEY,
     type STRING,
     id STRING,
     name STRING,
     lat DOUBLE,
     lon DOUBLE,
     country STRING
) WITH (
     KAFKA_TOPIC = 'grid-stations',
     VALUE_FORMAT = 'JSON'
);
```

```sql
-- table with link metadata :
CREATE TABLE linkTable (
     rowkey VARCHAR PRIMARY KEY,
     id STRING,
     regionContextTag STRING,
     asGeoJSON STRING
) WITH (
     KAFKA_TOPIC = 'grid-static-links',
     VALUE_FORMAT = 'JSON'
);
```

Enrich data points (from links) with context data

```
CREATE STREAM gridflowEnrichedSampleStream2
WITH (
     KAFKA_TOPIC = 'grid-flow-enriched-sample-stream2',
     VALUE_FORMAT = 'JSON',
     PARTITIONS = 1
)
AS SELECT
 r.rowkey as rowkey,
 r.ts,
 r.flow,
 l.id,
 l.regionContextTag,
 l.asGeoJSON,
 SPLIT(l.regionContextTag, '->')[1] AS source,
 SPLIT(l.regionContextTag, '->')[2] AS target
FROM gridLinkFlowDataStream r
  JOIN linkTable l ON r.rowkey = l.rowkey
EMIT CHANGES;
```

Aggregate the results in a topic

```
#
# Filter only traffic which is cross-border
#
Select * from gridflowEnrichedSampleStream2 where source != target emit changes;


Create table crossRegionFlows
WITH (
     KAFKA_TOPIC = 'grid-cross-region-flows',
     VALUE_FORMAT = 'JSON',
     PARTITIONS = 1
)
as Select REGIONCONTEXTTAG, sum(flow) from gridflowEnrichedSampleStream2
where source != target
group by REGIONCONTEXTTAG;
```

Results (1):

```
#
#  Aggregation over time for total flows since beginning ...
#
Create table crossRegionFlowStream3
WITH (
     KAFKA_TOPIC = 'grid-cross-region-flow3',
     VALUE_FORMAT = 'JSON',
     PARTITIONS = 1
)
AS Select
 REGIONCONTEXTTAG, sum(flow) as cross_region_flow,
 TOPK(SOURCE,1)[1] as source_region,
 TOPK(TARGET,1)[1] as target_region,
 TOPK(ASGEOJSON,1)[1] as GEOJSON
from gridflowEnrichedSampleStream2
where source != target
group by REGIONCONTEXTTAG
emit changes;

Select * from crossRegionFlowStream3 emit changes;
```

The result is updated every time a new data point arrives.

It shows the total sum over all values from the past.

Results (2):

This gives us a total flow value per region-pair per time interval.

```
#
#  Aggregation for time windows for actual flows ...
#
Create table crossRegionFlowStream4
WITH (
     KAFKA_TOPIC = 'grid-cross-region-flow4',
     VALUE_FORMAT = 'JSON',
     PARTITIONS = 1
)
AS Select
  REGIONCONTEXTTAG, sum(flow) as cross_region_flow,
  TOPK(SOURCE,1)[1] as source_region,
  TOPK(TARGET,1)[1] as target_region,
  TOPK(ASGEOJSON,1)[1] as GEOJSON
from gridflowEnrichedSampleStream2
WINDOW TUMBLING (SIZE 1 MINUTE)
where source != target
group by REGIONCONTEXTTAG
emit changes;

####################################################
#
# This is the result of the processing
#
####################################################
Select * from crossRegionFlowStream4 emit changes;
```

Run simulation for continuous data generation ...

```
simulateFlow();

System.out.println( "> Show GeoJSON data in browser: https://utahemre.github.io/geojsontest.html " );
```

```
[ITERATION] -> 7
PowerFlowSample{id='1ST12345-2ST12345', ts=1605189726931, flow=100.97549130324805}
PowerFlowSample{id='1ST12345-3ST12345', ts=1605189726931, flow=109.35624063754196}
PowerFlowSample{id='2ST12345-3ST12345', ts=1605189726931, flow=96.51855065838487}
PowerFlowSample{id='4ST12345-5ST12345', ts=1605189726931, flow=93.46840320850201}
PowerFlowSample{id='4ST12345-6ST12345', ts=1605189726931, flow=90.56437339903474}
PowerFlowSample{id='5ST12345-7ST12345', ts=1605189726931, flow=106.56615950518284}
PowerFlowSample{id='5ST12345-9ST12345', ts=1605189726931, flow=98.0096729951033}
PowerFlowSample{id='6ST12345-1ST12345', ts=1605189726931, flow=91.51766289437131}
PowerFlowSample{id='6ST12345-2ST12345', ts=1605189726931, flow=108.73703970530246}
PowerFlowSample{id='6ST12345-3ST12345', ts=1605189726931, flow=98.31443556059729}
PowerFlowSample{id='6ST12345-5ST12345', ts=1605189726931, flow=106.63149033122792}
PowerFlowSample{id='6ST12345-7ST12345', ts=1605189726931, flow=103.75719151450966}
PowerFlowSample{id='7ST12345-1ST12345', ts=1605189726931, flow=103.08443717578004}
PowerFlowSample{id='7ST12345-2ST12345', ts=1605189726931, flow=101.43170138126155}
PowerFlowSample{id='7ST12345-3ST12345', ts=1605189726931, flow=101.54492941402502}
PowerFlowSample{id='7ST12345-8ST12345', ts=1605189726931, flow=102.17189016549506}
PowerFlowSample{id='7ST12345-9ST12345', ts=1605189726931, flow=92.7827947966908}
PowerFlowSample{id='8ST12345-9ST12345', ts=1605189726931, flow=94.39865044304355}
PowerFlowSample{id='PP1-6ST12345', ts=1605189726931, flow=3.2228806460864265}
PowerFlowSample{id='PP2-6ST12345', ts=1605189726931, flow=4.098471694868678}
PowerFlowSample{id='PP3-8ST12345', ts=1605189726931, flow=5.080566534024371}
PowerFlowSample{id='PP4-9ST12345', ts=1605189726931, flow=3.850661362880845}
PowerFlowSample{id='PP5-1ST12345', ts=1605189726931, flow=4.313673473469986}
PowerFlowSample{id='PP1-6ST12345', ts=1605189726931, flow=3.122703038406629}
PowerFlowSample{id='PP1-6ST12345', ts=1605189726931, flow=2.8053574276822935}
PowerFlowSample{id='PP2-6ST12345', ts=1605189726931, flow=3.921778081870191}
PowerFlowSample{id='PP2-6ST12345', ts=1605189726931, flow=3.7364255062248737}
PowerFlowSample{id='PP3-8ST12345', ts=1605189726931, flow=5.15437252799277}
PowerFlowSample{id='PP3-8ST12345', ts=1605189726931, flow=4.633760649802475}
PowerFlowSample{id='PP4-9ST12345', ts=1605189726931, flow=3.7666391254177083}
PowerFlowSample{id='PP4-9ST12345', ts=1605189726931, flow=3.7936950075000766}
PowerFlowSample{id='PP5-1ST12345', ts=1605189726931, flow=4.159726968127969}
PowerFlowSample{id='PP5-1ST12345', ts=1605189726931, flow=3.717482317782453}
[-------------------]

Export-Import : 150.0 :: 100.0 => 50.0
Prod-Cons     : 600.0 :: 550.0 => 50.0
Excess        : 0.0
```

We simulate the power transfer on grid segments using a “white noise model”.

Each iteration represents a measurement cycle which would provide a measured data point per grid segment per time interval.

GeoJSON

GeoJSON is a format for encoding a variety of geographic data structures.

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [125.6, 10.1]
  },
  "properties": {
    "name": "Dinagat Islands"
  }
}
```

GeoJSON supports the following geometry types: `Point`, `LineString`, `Polygon`, `MultiPoint`, `MultiLineString`, and `MultiPolygon`. Geometric objects with additional properties are `Feature` objects. Sets of features are contained by `FeatureCollection` objects.

The GeoJSON Specification (RFC 7946)

In 2015, the Internet Engineering Task Force (IETF), in conjunction with the original specification authors, formed a GeoJSON WG to standardize GeoJSON. RFC 7946 was published in August 2016 and is the new standard specification of the GeoJSON format, replacing the 2008 GeoJSON specification.

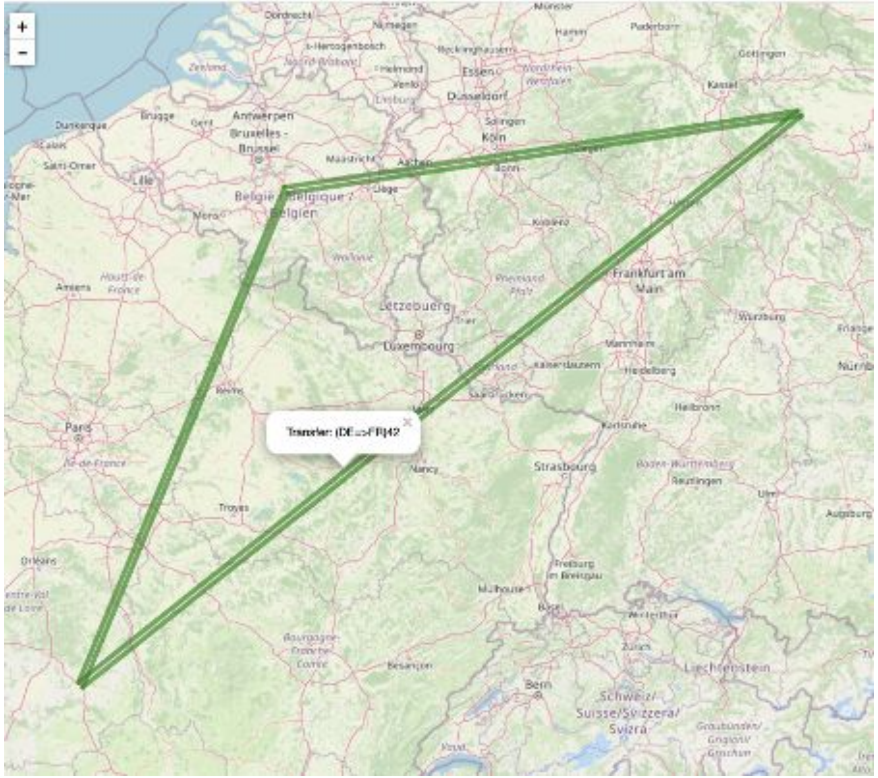
Structure of output: calculated results + template

```
####################################################
#
# This is an example record from our result
#
####################################################
[
  {
    "REGIONCONTEXTTAG": "FR->BE",
    "WINDOWSTART": 1605189720000,
    "WINDOWEND": 1605189780000,
    "CROSS_REGION_FLOW": 2997.9365865358313,
    "SOURCE_REGION": "FR",
    "TARGET_REGION": "BE",
    "GEOJSON": "{ \"type\": \"Feature\", \"geometry\": { \"type\": \"LineString\", \"coordinates\" :
  }
]
```

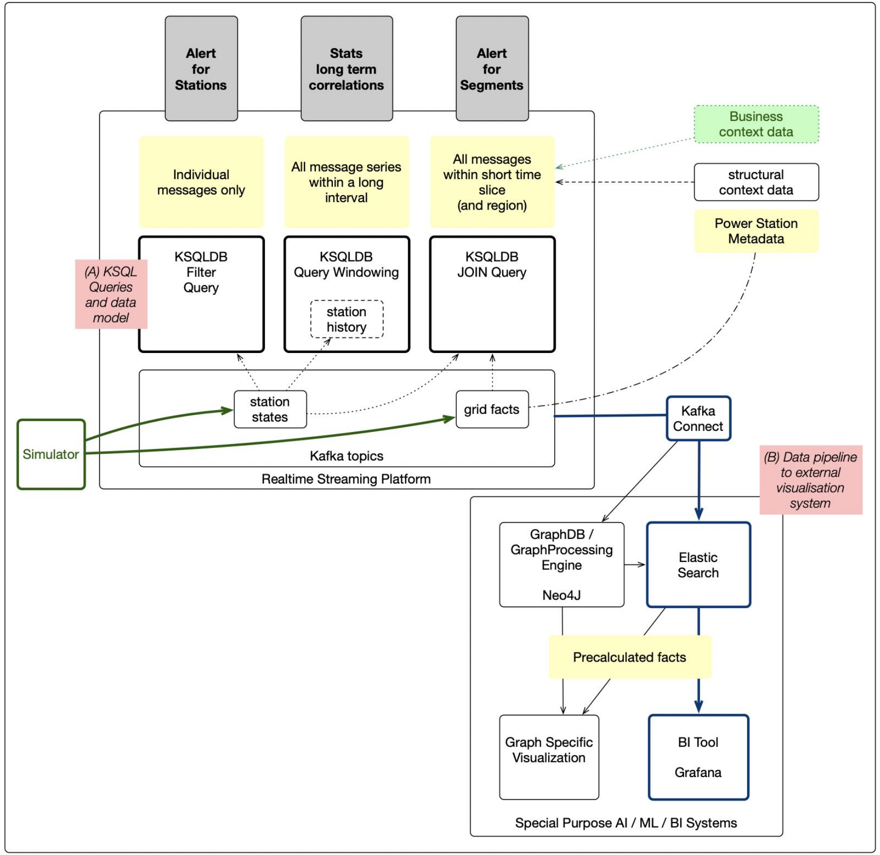
Example record for GeoJSON reference data

ID String => ROWKEY, *LOC* String


```
{
 "type": "Feature",
 "geometry": {
   "type": "Point",
   "coordinates": [51.3114,11.8542]
 },
 "properties": {
   "name": "Station 1"
   "id": "1STE1234"
 }
}
```



Recap: **Solution Overview**



Thank You

Next Steps: Query Management via CI/CD pipeline

For this we need a dedicated KSQL-server instance / cluster.

Deployment can be done:

- via pre-packaged KSQL script in Docker image (to implement)
- via REST-API and Maven goals (to implement)

CONFLUENT

Open Questions

???