Location Analytics – Real-Time Geofencing using Kafka

Berlin Buzzwords 2019
Guido Schmutz
(guido.schmutz@trivadis.com)

gschmutz

http://guidoschmutz.wordpress.com

Agenda

1. Introduction & Motivation
2. Implementing using KSQL
3. Implementing using Tile38
4. Visualization using ArcadiaData
5. Summary

Guido Schmutz

Working at Trivadis for more than 22 years

Oracle Groundbreaker Ambassador & Oracle ACE Director

Consultant, Trainer, Software Architect for Java, AWS, Azure, Oracle Cloud, SOA and Big Data / Fast Data

Platform Architect & Head of Trivadis Architecture Board

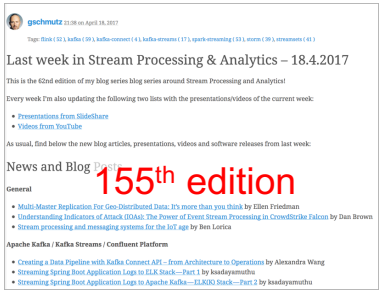
More than 30 years of software development experience

Contact: guido.schmutz@trivadis.com

Blog: http://guidoschmutz.wordpress.com

Slideshare: http://www.slideshare.net/gschmutz

Twitter: gschmutz



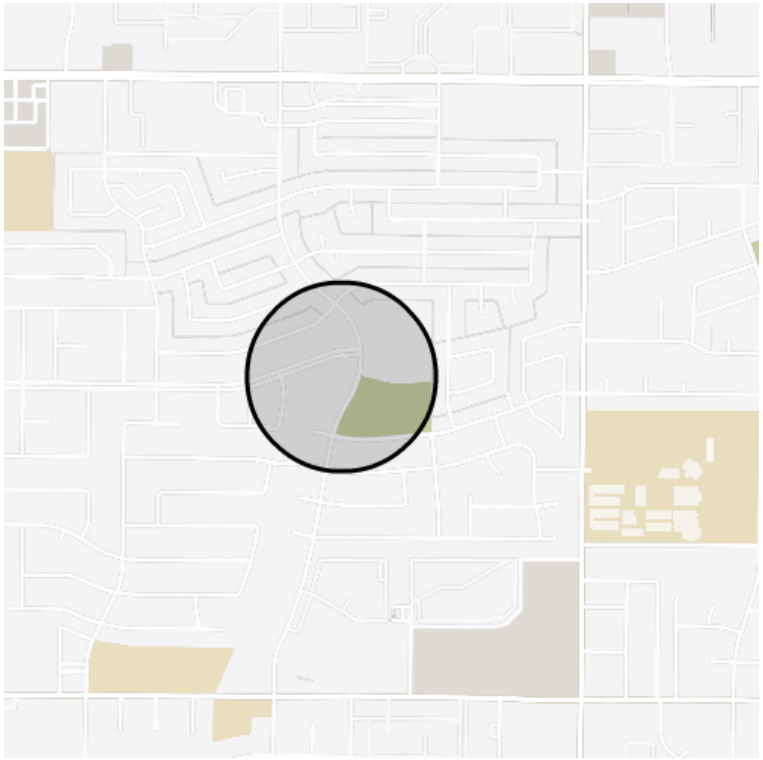
Introduction

Geofencing – What is it?

the use of GPS or RFID technology to create a virtual geographic boundary, enabling software to trigger a response when a object/device **enters** or **leaves** a particular area

Possible Events

- OUTSIDE
- INSIDE
- ENTER
- EXIT



Source: https://tile38.com

Geofencing – What can we do with it?

- **On-Demand and Delivery Service**s - assign orders to an area's designated service provider
- **On-Demand Transportation** - track Electronic Transportation Devices and their distance from charging stations
- **Transportation Management** - track flow of people using public transport systems
- **Commercial Real Estate** - Identify how many people drive or walk by a specific location
- **Retail Shopper Guidance** - Guide customer to a specific product once they are in your store
- **Property Security** - Open or lock doors as individuals with designated devices approach or leave a building or vehicle.
- **Property Control** - restrict vehicles to be operational only inside a geofenced area – like drones or construction equipment

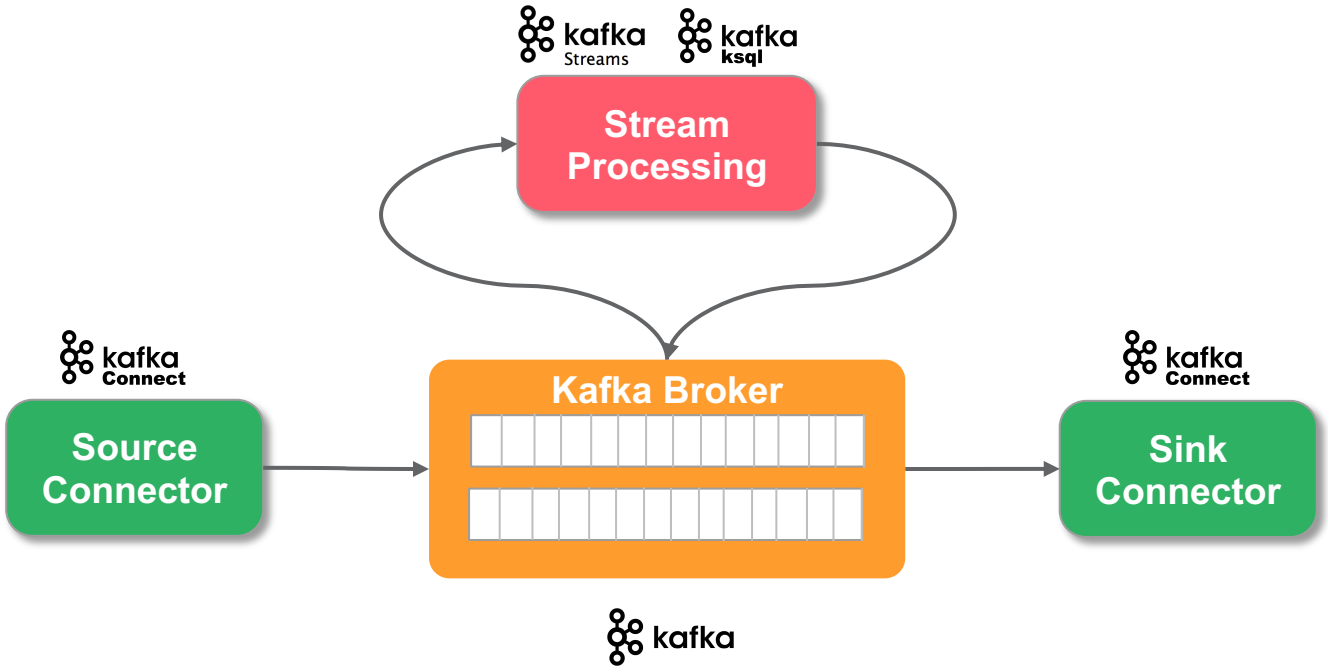
Geo-Processing

Well-known text (**WKT**) is a text markup language for representing vector geometry objects on a map

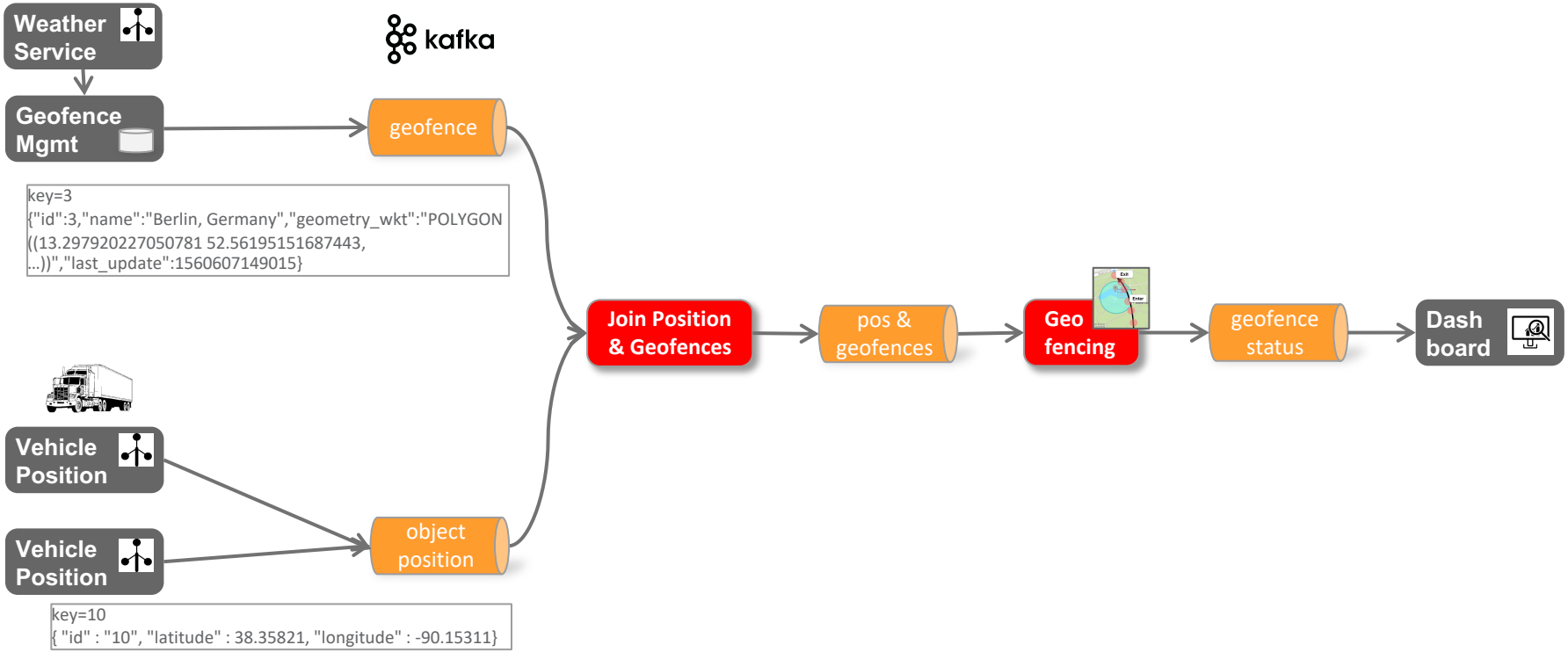
Type	Examples	
Point		`POINT (30 10)`
LineString		`LINESTRING (30 10, 10 30, 40 40)`
Polygon		`POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))`
		`POLYGON ((35 10, 45 45, 15 40, 10 20, 35 10), (20 30, 35 35, 30 20, 20 30))`

GeoTools is a free software **GIS toolkit** for developing standards compliant solutions

Apache Kafka – A Streaming Platform



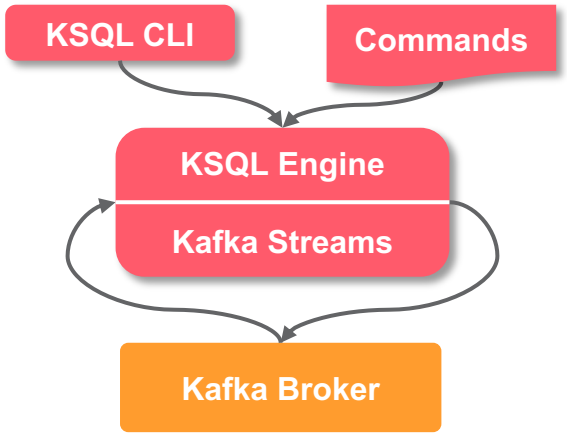
High Level Overview of Use Case



Implementing using KSQL

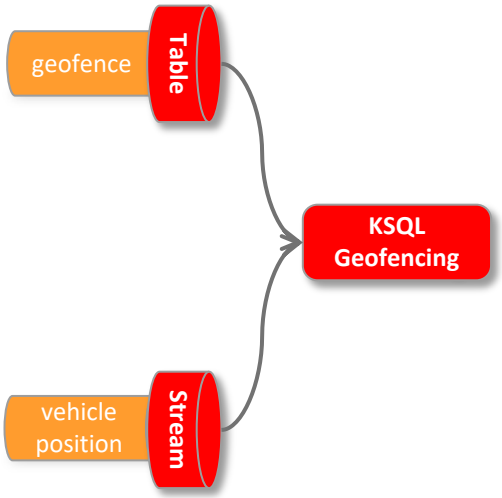
KSQL - Overview

- Stream Processing with zero coding using SQL-like language
- Stream and Table as first-class citizens



- Stream
  - unbounded sequence of structured data ("facts")
  - Facts in a stream are **immutable**
- Table
  - collected state of a stream
  - Latest value for each key in a stream
  - Facts in a table are **mutable**

KSQL – Streams and Tabless




```
CREATE TABLE geo_fence_t
       (id BIGINT,
       name VARCHAR,
       geometry_wkt VARCHAR)
WITH (KAFKA_TOPIC='geo_fence',
       VALUE_FORMAT='JSON',
       KEY = 'id');
```

```
CREATE STREAM vehicle_position_s
   (id VARCHAR,
    latitude DOUBLE,
    longitude DOUBLE)
  WITH (KAFKA_TOPIC='vehicle_position',
        VALUE_FORMAT='DELIMITED');
```

How to determine ”inside” or “outside” geofence?

Only one standard UDF for geo processing in KSQL: GEO_DISTANCE

Implement custom UDF using functionality from GeoTools Java library

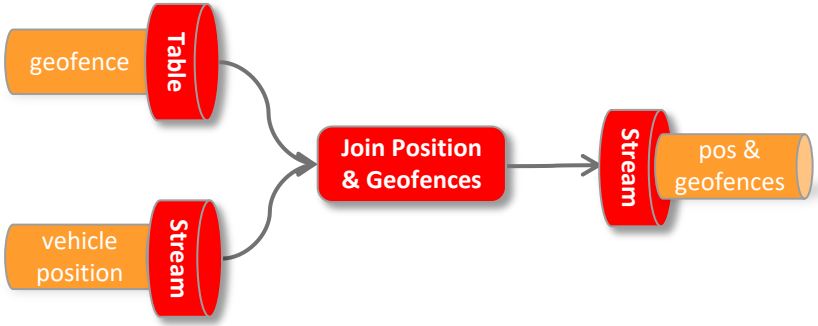
```
public String geo_fence(final double latitude, final double longitude,
                        final String geometryWKT){ .. }

public List<String> geo_fence_bulk(final double latitude
        , final double longitude, List<String> idGeometryListWKT) { .. }
```

```
ksql> SELECT geo_fence(latitude, longitude, ' POLYGON ((13.297920227050781
52.56195151687443, 13.2440185546875 52.530216577830124, ...))')
FROM test_geo_udf_s;

52.4497 | 13.3096 | OUTSIDE
52.4556 | 13.3178 | INSIDE
```

Custom UDF to determine if Point is inside a geometry

```
@Udf(description = "determines if a lat/long is inside or outside the
                    geometry passed as the 3rd parameter as WKT encoded ...")
public String geo_fence(final double latitude, final double longitude,
                        final String geometryWKT) {
  String status = "";
  GeometryFactory geometryFactory = JTSFactoryFinder.getGeometryFactory();
  WKTReader reader = new WKTReader(geometryFactory);

  Polygon polygon = (Polygon) reader.read(geometryWKT);
  Coordinate coord = new Coordinate(longitude, latitude);
  Point point = geometryFactory.createPoint(coord);
  if (point.within(polygon)) {
      status = "INSIDE";
  } else {
      status = "OUTSIDE";
  }
  return status;
}
```

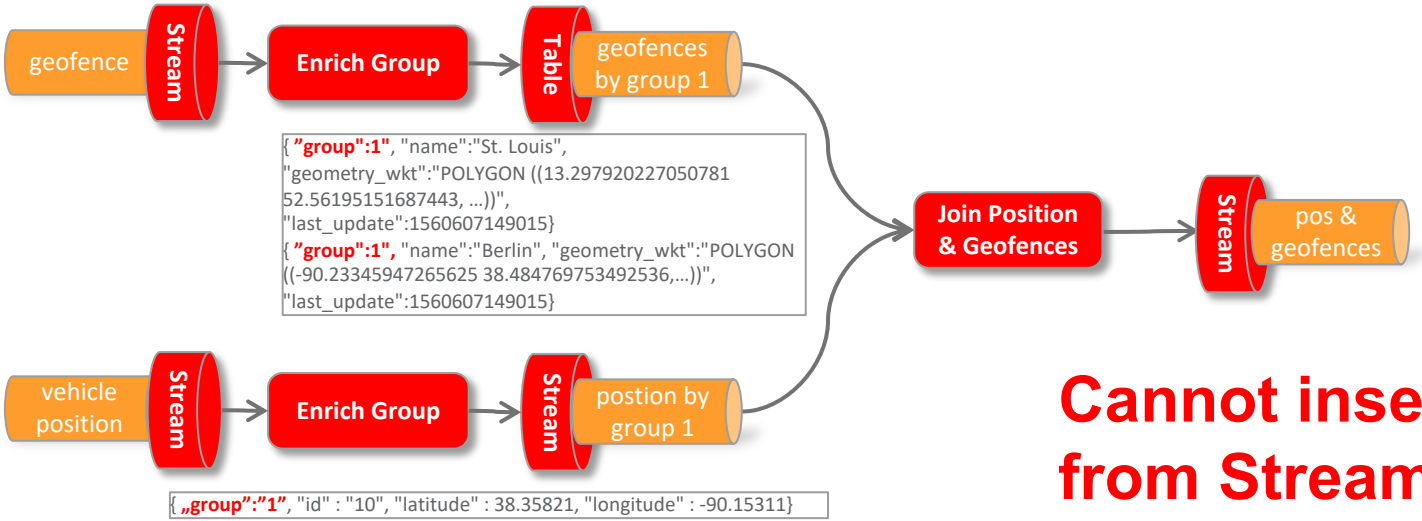
1) Using Cross Join



There is no Cross Join in KSQL!

```
CREATE STREAM vp_join_gf_s
AS
SELECT vp.id, vp.latitude, vp.longitude,
       gf.geometry_wkt
FROM vehicle_position_s AS vp
CROSS JOIN geo_fence_t AS gf
```

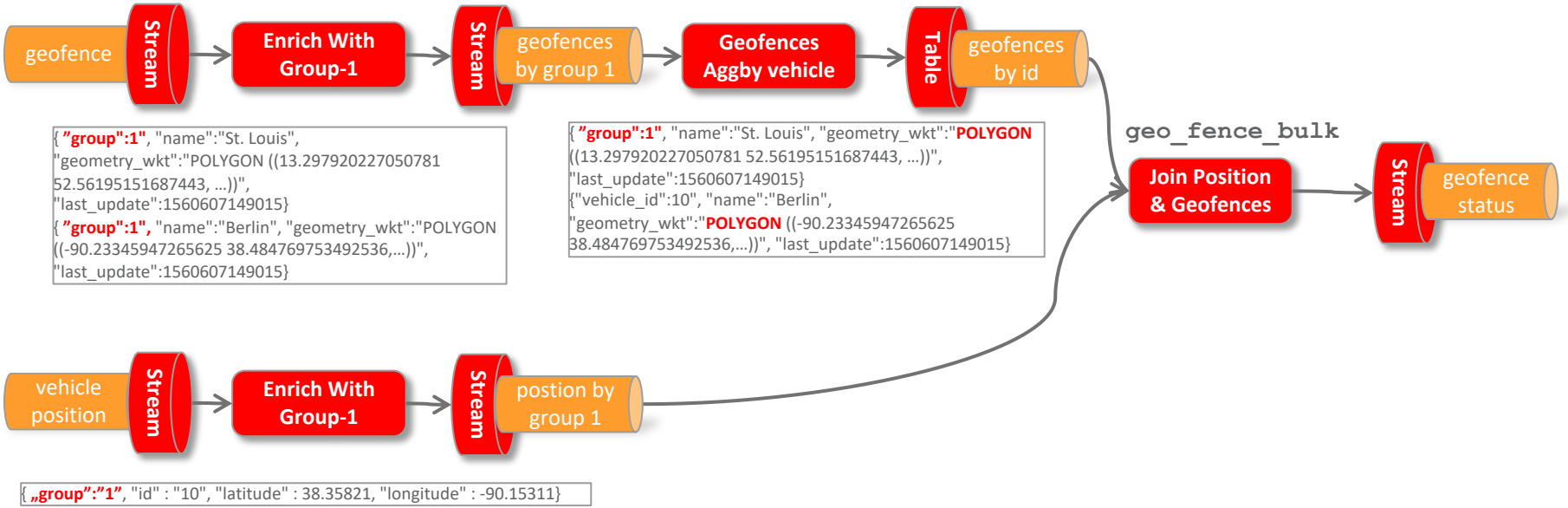
2) INNER Join



Cannot insert into Table from Stream

```
>INSERT INTO geo_fence_t
>SELECT '1' AS group_id, geof.id, …
>FROM geo_fence_s geof;
INSERT INTO can only be used to insert into
a stream. A02_GEO_FENCE_T is a table.
```

3) Geofences aggregated in one group



Scalable	high	medium	**low**
Latency	low	**medium**	high
“Code Smell”	**low**	medium	high

3) Geofences aggregated in one group

```
CREATE TABLE a03_geo_fence_aggby_group_t
AS
SELECT group_id
,   collect_set(id + ':' + geometry_wkt) AS id_geometry_wkt_list
FROM a03_geo_fence_by_group_s       geof
GROUP BY group_id;
```

```
CREATE STREAM a03_vehicle_position_by_group_s
AS
SELECT '1' group_id, vehp.id, vehp.latitude, vehp.longitude
FROM vehicle_position_s vehp
PARTITION BY group_id;
```

3) Geofences aggregated in one group

```
CREATE STREAM a03_geo_fence_status_s
AS
SELECT vehp.id, vehp.latitude, vehp.longitude,
       geo_fence_bulk(vehp.latitude, vehp.longitude,
                      geofaggid_geometry_wkt_list) AS geofence_status
FROM a03_vehicle_position_by_group_s vehp
LEFT JOIN a03_geo_fence_aggby_group_t geofagg
ON vehp.group_id = geofagg.group_id;
```

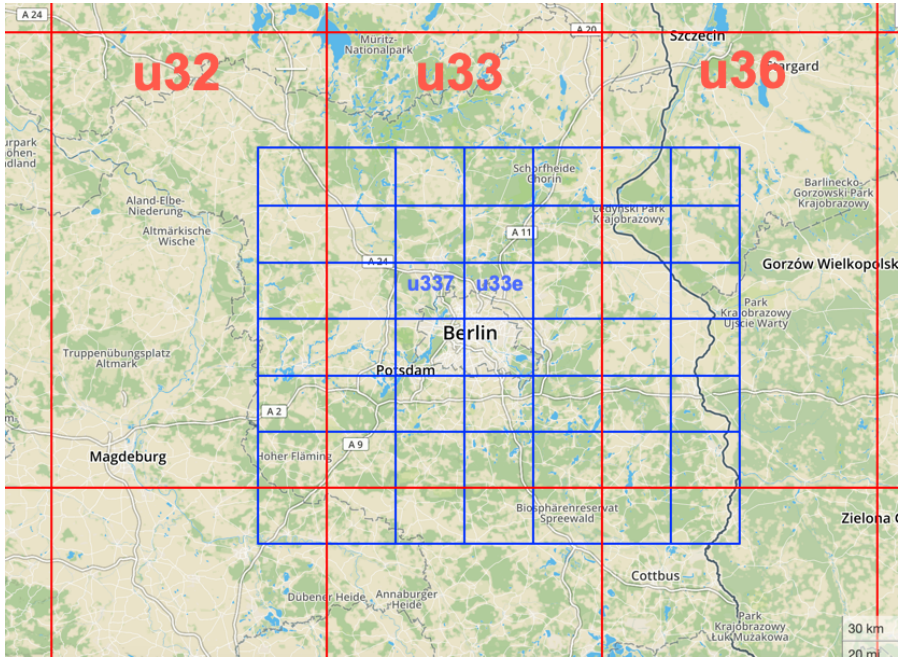
```
ksql> SELECT * FROM a03_geo_fence_status_s;

46 | 52.47546 | 13.34851 | [1:OUTSIDE, 3:INSIDE]
46 | 52.47521 | 13.34881 | [1:OUTSIDE, 3:INSIDE]
...
```

As many as there are geo-fences

Geo Hash for a better distribution

Geohash is a geocoding which encodes a geographic location into a short string of letters and digits

Length	Area width x height
1	5,009.4km x 4,992.6km
2	1,252.3km x 624.1km
3	156.5km x 156km
4	39.1km x 19.5km
12	3.7cm x 1.9cm



http://geohash.gofreerange.com/

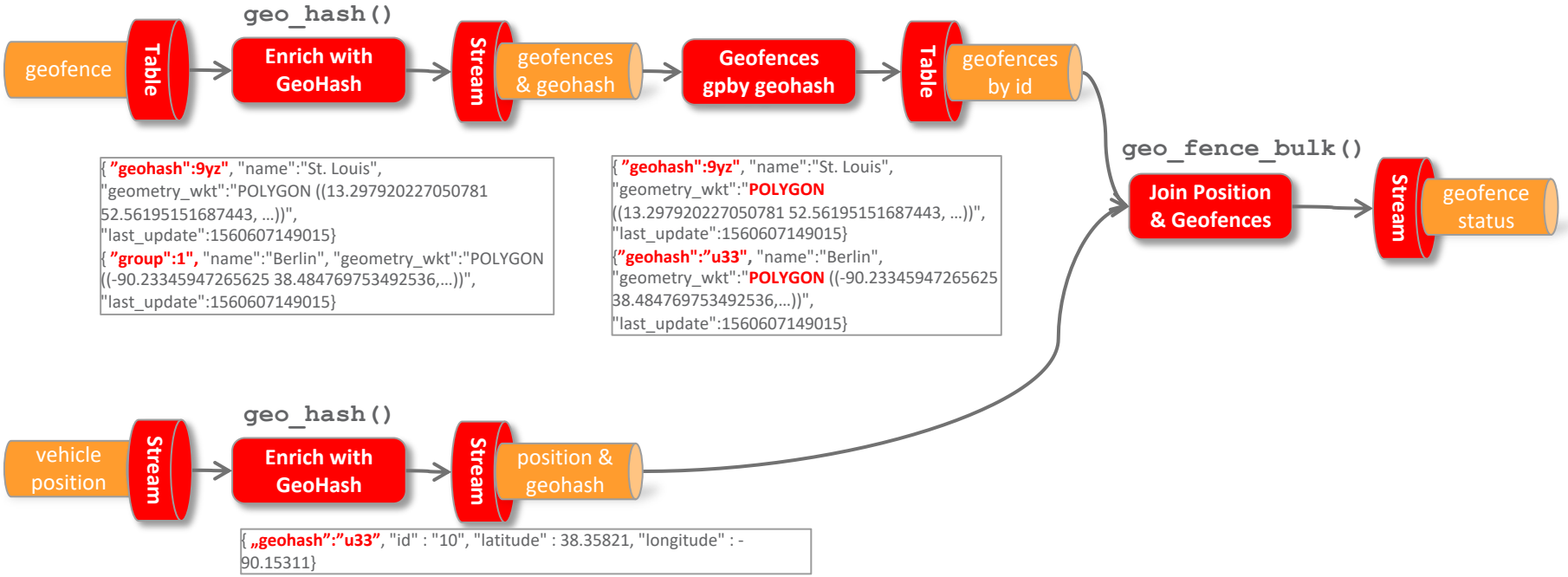
Geo Hash Custom UDF

```
public String geohash(final double latitude,
                      final double longitude, int length)
public List<String> neighbours(String geohash)
public String adjacentHash(String geohash, String directionString)
public List<String> coverBoundingBox(String geometryWKT, int length)
```

```
ksql> SELECT latitude, longitude, geo_hash(latitude, longitude, 3)
>FROM test_geo_udf_s;
38.484769753492536 | -90.23345947265625 | 9yz
```

```
ksql> SELECT geometry_wkt, geo_hash(geometry_wkt, 5)
>FROM test_geo_udf_s;
POLYGON ((-90.23345947265625 38.484769753492536, -90.25886535644531
38.47455675836861, ...)) | [9yzf6, 9yzf7, 9yzfd, 9yzfe, 9yzff, 9yzfg, 9yzfk,
9yzfs, 9yzfu]
```

4) Geofences aggregated by GeoHash



Scalable	high	medium	low
Latency	low	medium	high
"Code Smell"	low	medium	high

4) Geofences aggregated by GeoHash

```
CREATE STREAM a04_geo_fence_by_geohash_s
AS
SELECT geo_hash(geometry_wkt, 3)[0] geo_hash, id, name, geometry_wkt
FROM a04_geo_fence_s
PARTITION by geo_hash;
```

```
INSERT INTO a04_geo_fence_by_geohash_s
SELECT geo_hash(geometry_wkt, 3)[1] geo_hash, id, name, geometry_wkt
FROM a04_geo_fence_s
WHERE geo_hash(geometry_wkt, 3)[1] IS NOT NULL
PARTITION BY geo_hash;s
```

```
INSERT INTO a04_geo_fence_by_geohash_s
SELECT ...
```

There is no explode() functionality in KSQL!

https://github.com/confluentinc/ksql/issues/527

4) Geofences aggregated by GeoHash

```
CREATE TABLE a04_geo_fence_by_geohash_t
AS
SELECT geo_hash,
   COLLECT_SET(id + ':' + geometry_wkt) AS id_geometry_wkt_list,
   COLLECT_SET(id) id_list
FROM a04_geo_fence_by_geohash_s
GROUP BY geo_hash;
```

```
CREATE STREAM a04_vehicle_position_by_geohash_s
AS
SELECT vp.id, vp.latitude, vp.longitude,
       geo_hash(vp.latitude, vp.longitude, 3) geo_hash
FROM vehicle_position_s vp
PARTITION BY geo_hash;
```

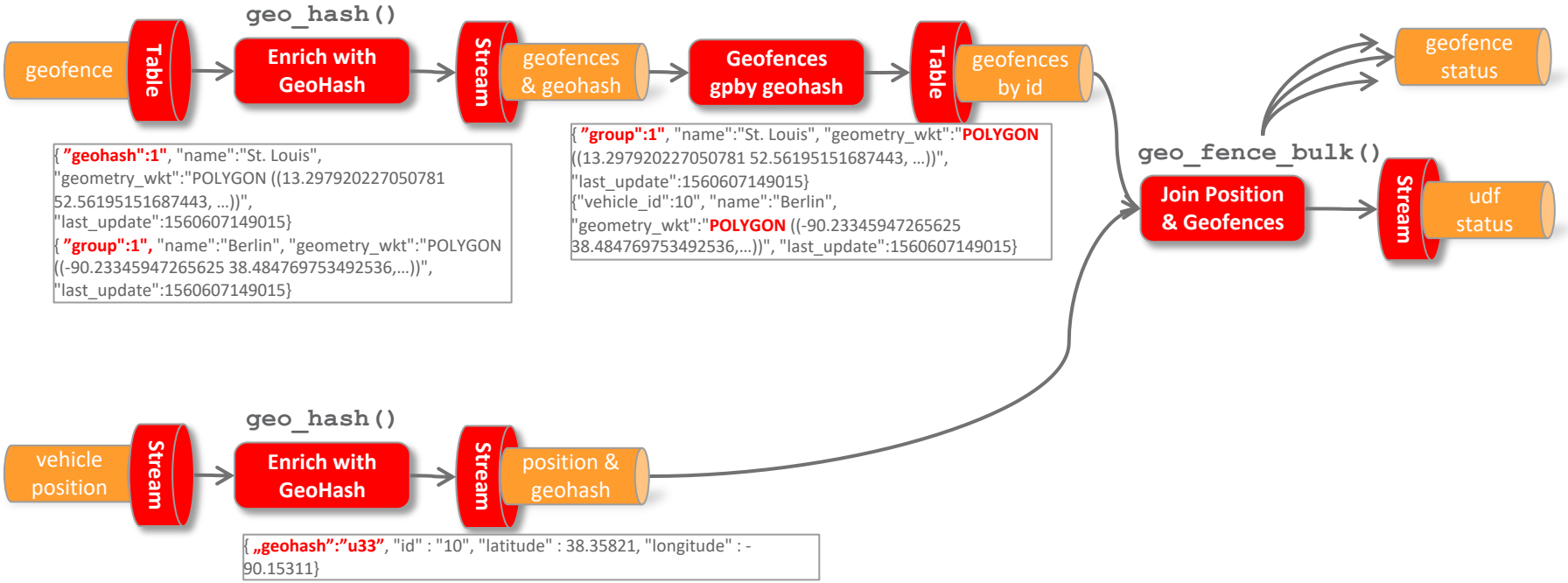
4) Geofences aggregated by GeoHash

```
CREATE STREAM a04_geo_fence_status_s
AS
SELECT vp.geo_hash, vp.id, vp.latitude, vp.longitude,
       geo_fence_bulk (vp.latitude, vp.longitude, gf.id_geometry_wkt_list)
                    AS fence_status
FROM a04_vehicle_position_by_geohash_s vp \
LEFT JOIN a04_geo_fence_by_geohash_t gf \
ON (vp.geo_hash = gf.geo_hash);
```

```
ksql> SELECT * FROM a04_geo_fence_status_s;

u33 | 46 | 52.3906 | 13.1599 | [3:OUTSIDE]
u33 | 46 | 52.3906 | 13.1599 | [3:OUTSIDE]
9yz | 12 | 38.34409 | -90.15034 | [2:OUTSIDE, 1:OUTSIDE]
...
```

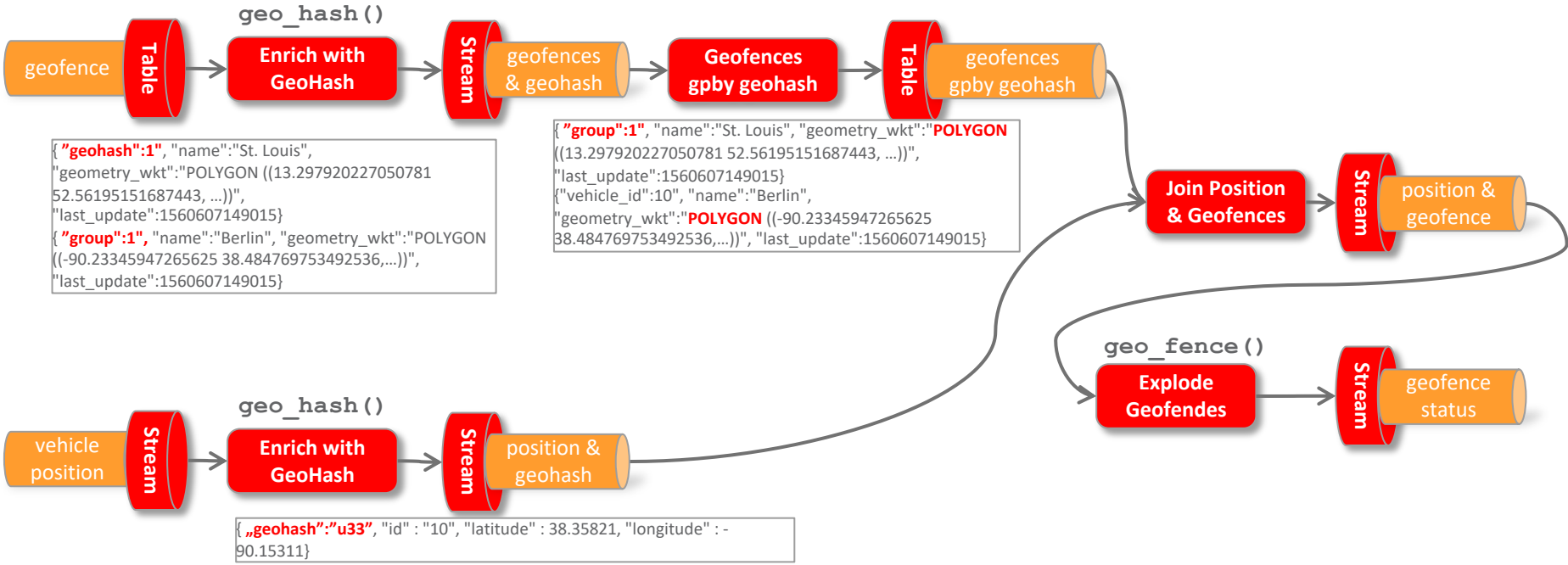
As many as there are geo-fences in geohash

4a) Geofences aggregated by GeoHash



Scalable	high	medium	low
Latency	low	medium	high
"Code Smell"	low	medium	high

4b) Geofences aggregated by GeoHash



Scalable	high	medium	low
Latency	low	medium	high
"Code Smell"	low	medium	high

4b) Geofences aggregated by GeoHash

```
CREATE STREAM a04b_geofence_udf_status_s
AS
SELECT id, latitude, longitude, id_list[0] AS geofence_id,
    geo_fence(latitude, longitude, geometry_wkt_list[0]) AS geofence_status
FROM a04_vehicle_position_by_geohash_s vp \
LEFT JOIN a04_geo_fence_by_geohash_t gf \
ON (vp.geo_hash = gf.geo_hash);
```

```
INSERT INTO a04b_geofence_udf_status_s
SELECT id, latitude, longitude, id_list[1] geofence_id,
    geo_fence(latitude, longitude, geometry_wkt_list[1]) AS geofence_status
FROM a04_vehicle_position_by_geohash_s vp \
LEFT JOIN a04_geo_fence_by_geohash_t gf \
ON (vp.geo_hash = gf.geo_hash)
WHERE id_list[1] IS NOT NULL;
```

Implementing using Tile38

Tile38

https://tile38.com

Open Source Geospatial Database & Geofencing Server

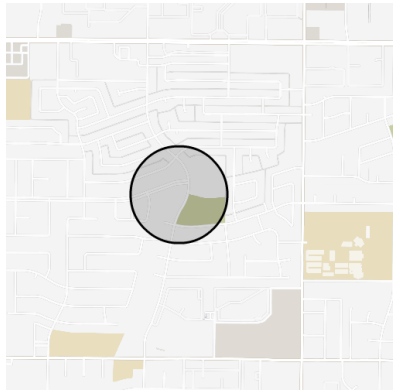
Real Time Geofencing

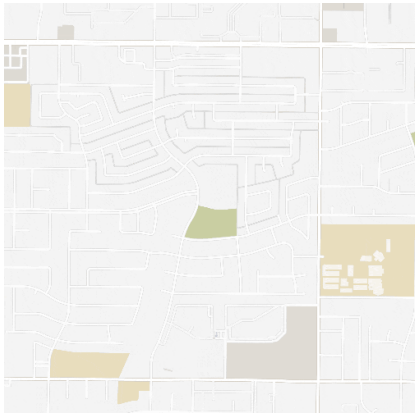
Roaming Geofencing

Fast Spatial Indices

Plugable Event Notifications

NATS, Apache Kafka, Amazon SQS, gRPC, redis, RabbitMQ, MQTT.org, HTTP/2





Tile38 – How does it work?

```
> SETCHAN berlin WITHIN vehicle FENCE OBJECT
{"type":"Polygon","coordinates":[[[13.297920227050781,52.56195151687443],[1
3.2440185546875,52.530216577830124],[13.267364501953125,52.45998421679598],
[13.35113525390625,52.44826791583386],[13.405036926269531,52.44952338289473
],[13.501167297363281,52.47148826410652], ...]]}

> SUBSCRIBE berlin
{"ok":true,"command":"subscribe","channel":"berlin","num":1,"elapsed":"5.85
µs"}

.
.
.

{"command":"set","group":"5d07581689807d000193ac33","detect":"outside","hoo
k":"berlin","key":"vehicle","time":"2019-06-
17T09:06:30.624923584Z","id":"10","object":{"type":"Point","coordinates":[1
3.3096,52.4497]}}
```

```
SET vehicle 10 POINT 52.4497 13.3096
```

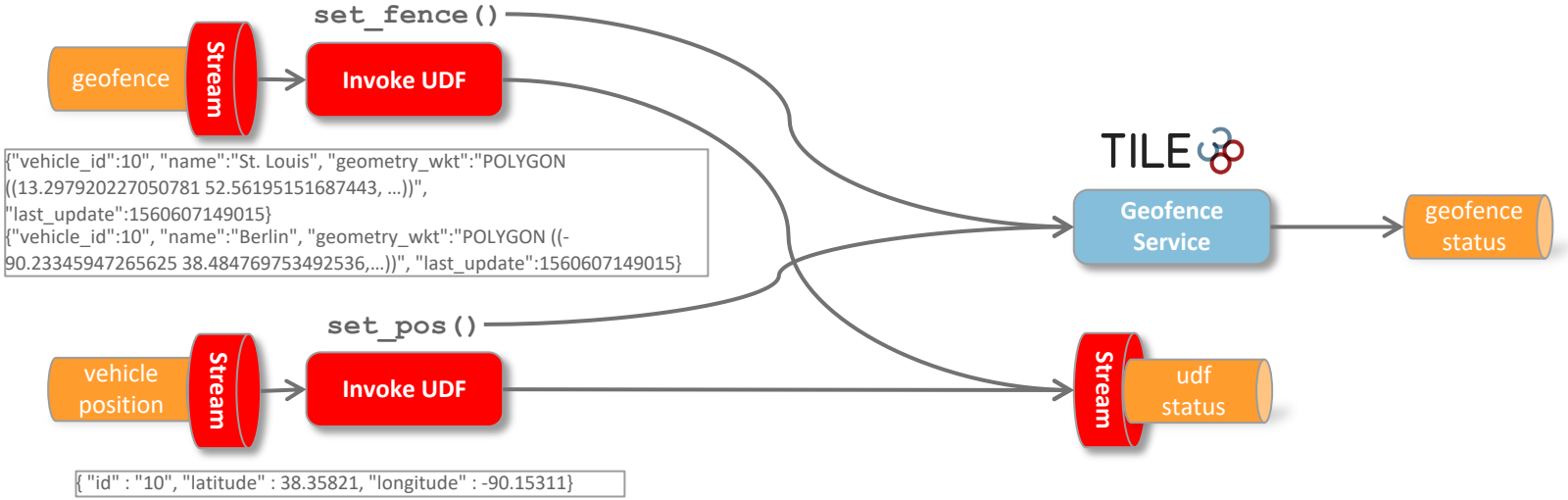
Tile38 – How does it work?

```
> SETHOOK berlin_hook kafka://broker-1:9092/tile38_geofence_status WITHIN
vehicle FENCE OBJECT
{"type":"Polygon","coordinates":[[[13.297920227050781,52.56195151687443],[1
3.2440185546875,52.530216577830124],[13.267364501953125,52.45998421679598],
[13.35113525390625,52.44826791583386],[13.405036926269531,52.44952338289473
],[13.501167297363281,52.47148826410652], ...]]}
```

```
SET vehicle 10 POINT 52.4497 13.3096
```

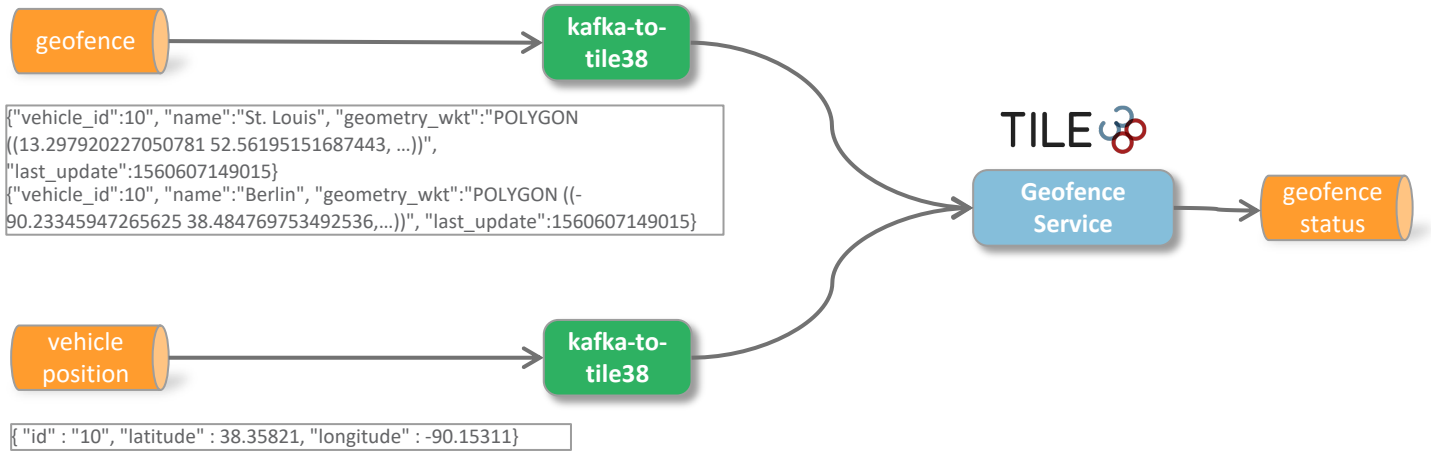
```
bigdata@bigdata:~$ kafkacat -b localhost -t tile38_geofence_status
% Auto-selecting Consumer mode (use -P or -C to override)
{"command":"set","group":"5d07581689807d000193ac34","detect":"outside","hoo
k":"berlin_hook","key":"vehicle","time":"2019-06-
17T09:12:00.488599119Z","id":"10","object":{"type":"Point","coordinates":[1
3.3096,52.4497]}}
```

1) Enrich with GeoFences – aggregated by geohash



Scalable	**high**	medium	low
Latency	**low**	medium	high
“Code Smell”	low	**medium**	high

2) Using Custom Kafka Connector for Tile38



Scalable	**high**	medium	low
Latency	**low**	medium	high
"Code Smell"	**low**	medium	high

2) Using Custom Kafka Connector for Tile38

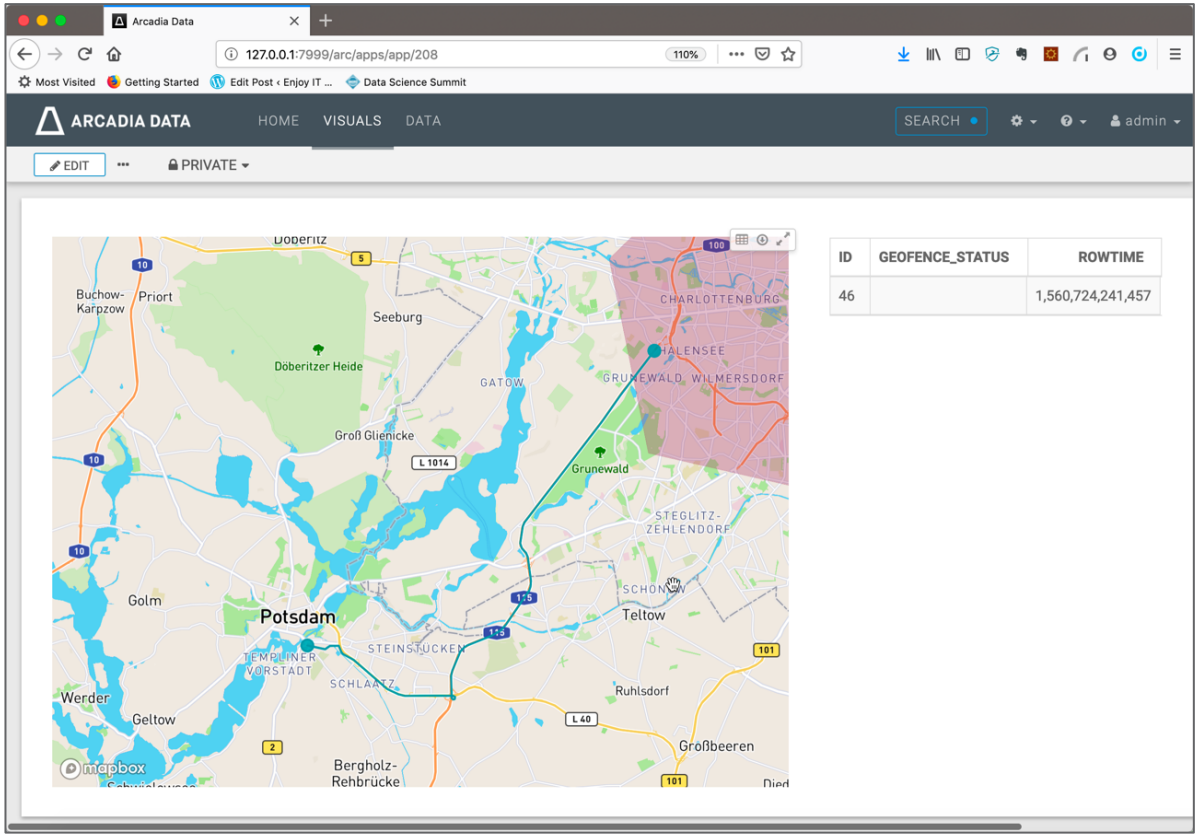
```
curl -X PUT \
  /api/kafka-connect-1/connectors/Tile38SinkConnector/config \
  -H 'Content-Type: application/json' \
  -H 'Accept: application/json' \
  -d '{
  "connector.class":
"com.trivadis.geofence.kafka.connect.Tile38SinkConnector",
  "topics": "vehicle_position",
  "tasks.max": "1",
  "tile38.key": "vehicle",
  "tile38.operation": "SET",
  "tile38.hosts": "tile38:9851"
}'
```

Currently only supports SET command

Visualization using Arcadia Data

Arcadia Data

https://www.arcadiadata.com/



Summary

Outlook

- Geo Fencing is doable using Kafka and KSQL
- KSQL is similar to SQL, but don’t think relational
- UDF and UDAF’s is a powerful way to extend KSQL
- Use Geo Hahes to partition work

- Outlook
  - Performance Tests
  - Cleanup code of UDFs and UDAFs
  - Implement Kafka Source Connector for Tile 38

Technology on its own won't help you.

You need to know how to use it properly.