AWS

SUMMIT

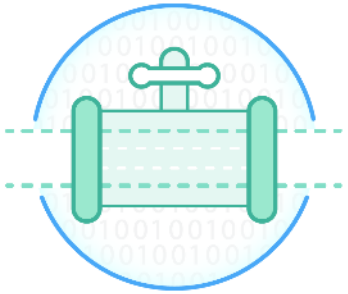
Analyzing Streaming Data in Real-Time with Amazon Kinesis Analytics

Dr. Steffen Hausmann, Solutions Architect, AWS

May 18, 2017

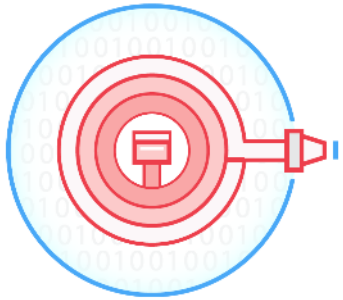
amazon web services

Amazon Kinesis makes it easy to work with real-time streaming data



Amazon Kinesis Streams

- For technical developers
- Collect and stream data for ordered, replayable, real-time processing



Amazon Kinesis Firehose

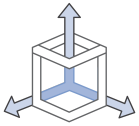
- For all developers, data scientists
- Easily load massive volumes of streaming data into Amazon S3, Amazon Redshift, Amazon Elasticsearch Service

Amazon Kinesis Analytics

- For all developers, data scientists
- Easily analyze data streams using standard SQL queries

Amazon Kinesis Analytics

Easy to use



Automatic elasticity

Real-time processing

Pay for only what you use

Standard SQL for analytics

Connect to streaming source

- Streaming data sources include Amazon Kinesis Firehose or Amazon Kinesis Streams
- Input formats include JSON, .csv, variable column, or unstructured text
- Each input has a schema; schema is inferred, but you can edit
- Reference data sources (S3) for data enrichment

Write SQL code

- Build streaming applications with one-to-many SQL statements
- Robust SQL support and advanced analytic functions
- Extensions to the SQL standard to work seamlessly with streaming data
- Support for at-least-once processing semantics

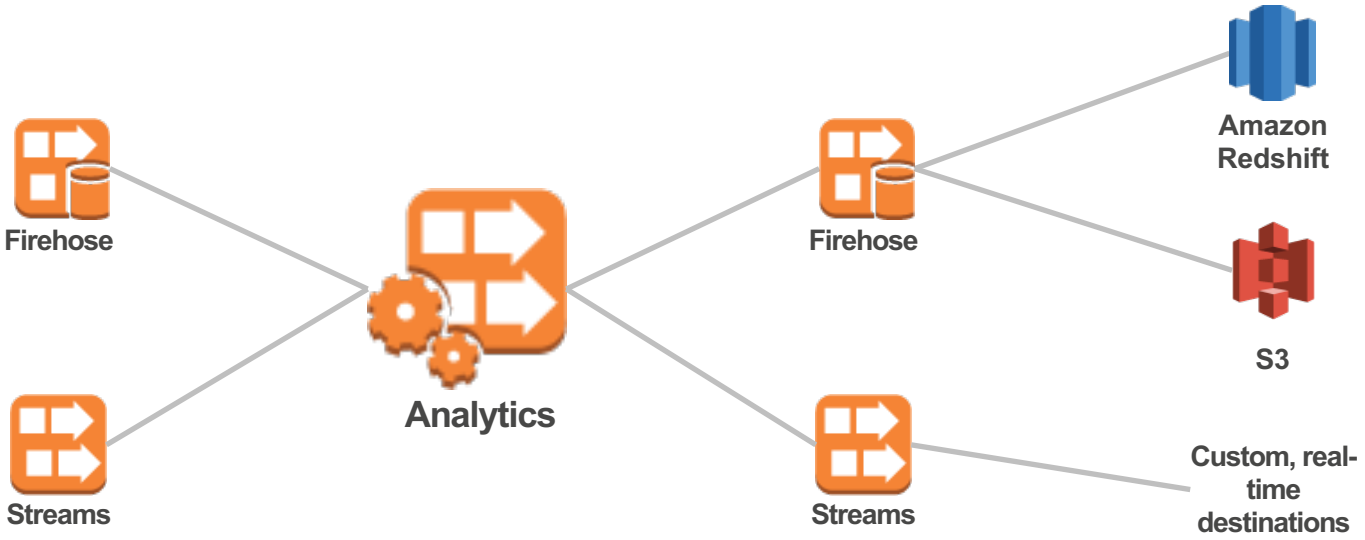
Continuously deliver SQL results

- Send processed data to multiple destinations
  - S3, Amazon Redshift, Amazon ES (through Firehose)
  - Streams (with AWS Lambda integration for custom destinations)
- End-to-end processing speed as low as sub-second
- Separation of processing and data delivery

What are common uses for Amazon Kinesis Analytics?

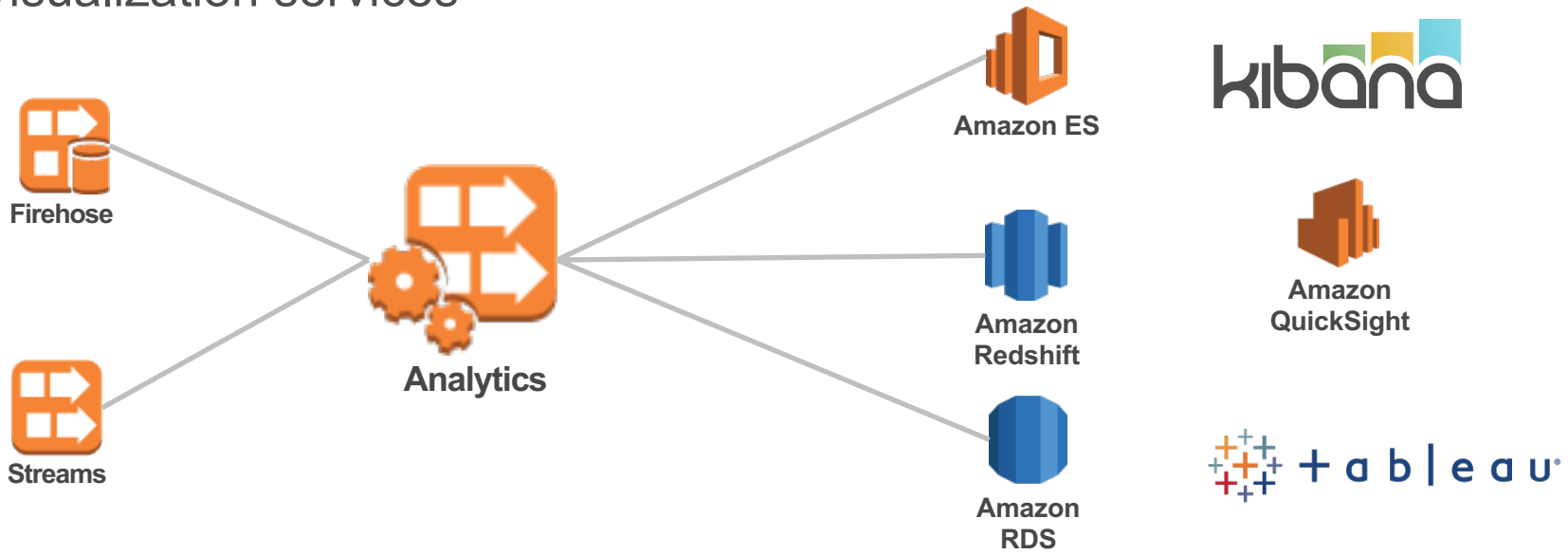
Generate time series analytics

- Compute key performance indicators over time periods
- Combine with static or historical data in S3 or Amazon Redshift



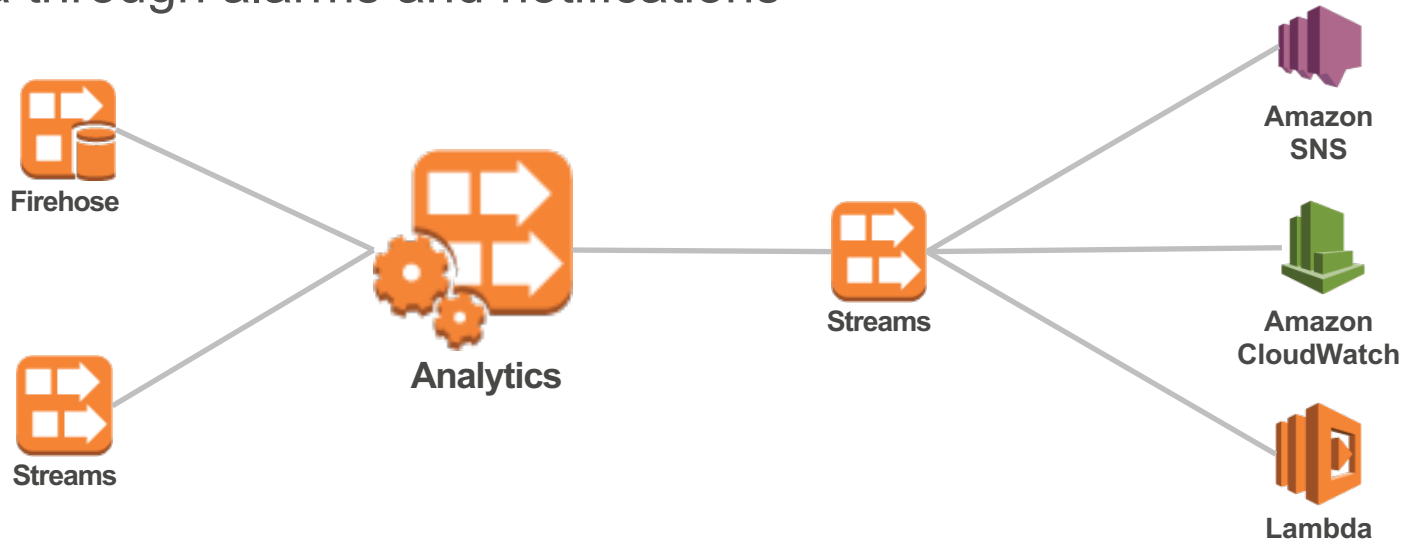
Feed real-time dashboards

- Validate and transform raw data, and then process to calculate meaningful statistics
- Send processed data downstream for visualization in BI and visualization services



Create real-time alarms and notifications

- Build sequences of events from the stream, like user sessions in a clickstream or app behavior through logs
- Identify events (or a series of events) of interest, and react to the data through alarms and notifications



Example: Bundesliga Tweet Analysis

Example Scenario Requirements

Data to capture

- Filter for soccer-related tweets
- Total number of tweets per hour that contain hashtags for soccer teams
- Top 5 mentioned teams names per hour

Output Requirements

- Filtered tweets are saved to Amazon S3
- Hourly aggregate count is saved to Amazon ES
- Full team name of top 5 hashtags are saved to Amazon ES

Why use Amazon Kinesis Analytics for this solution?

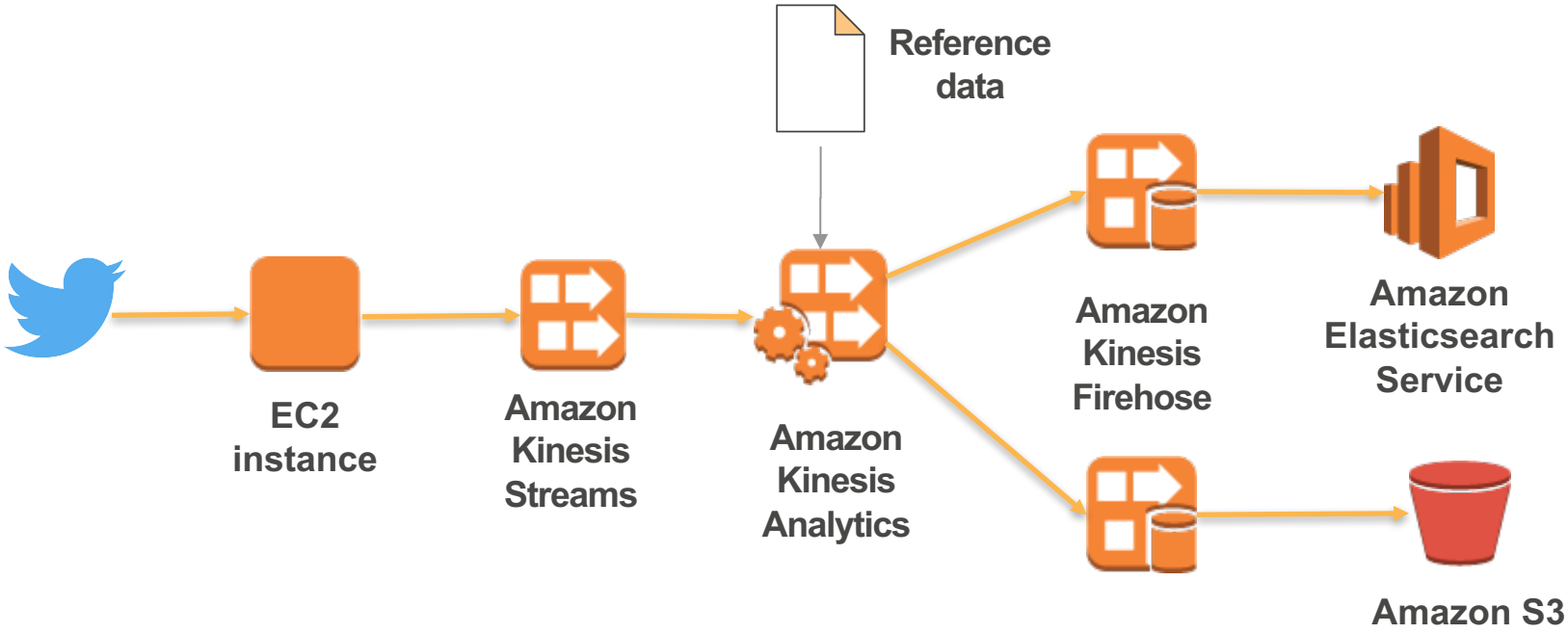
Challenges

- Twitter stream can be noisy
- Tweet structure is complex, with several levels of nested JSON
- soccer-related tweet volume is cyclical

With Amazon Kinesis Analytics:

- Easily filter out unwanted tweets
- Normalize tweet schema for simple SQL queries
- Automatically scale to meet demand

End-to-End Architecture



How is streaming data accessed with SQL?

STREAM

- Analogous to a TABLE
- Represents continuous data flow

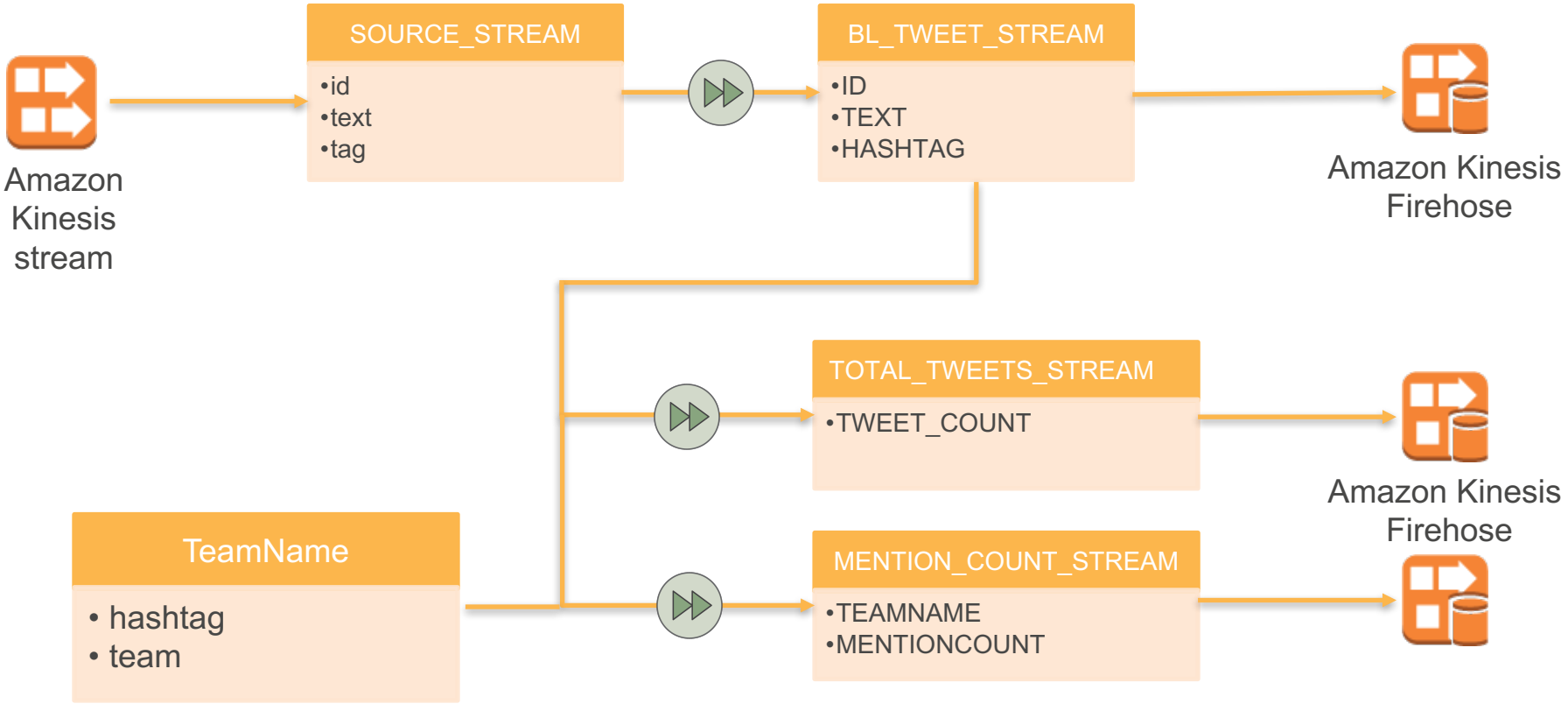
```
CREATE OR REPLACE STREAM "BL_TWEET_STREAM" (
ID BIGINT, TWEET_TEXT VARCHAR(140), HASHTAG VARCHAR(140));
```

PUMP

- Continuous INSERT query
- Inserts data from one in-application stream to another

```
CREATE OR REPLACE PUMP "BL_TWEET_PUMP" AS
       INSERT INTO "BL_TWEET_STREAM"
           SELECT STREAM * FROM . . .
```

Kinesis Analytics Application Overview



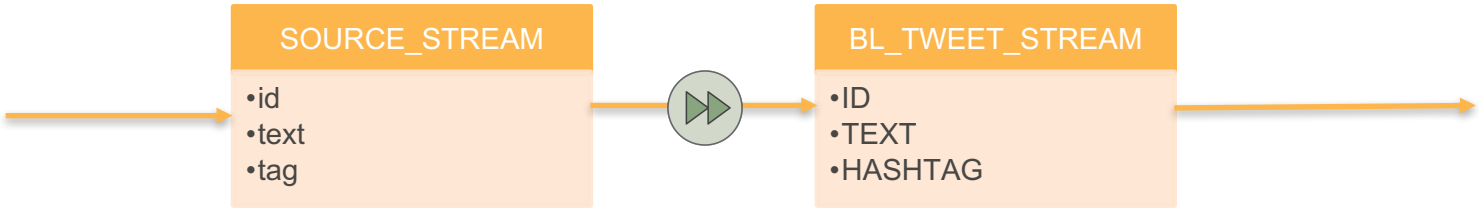
How are tweets mapped to a schema?


```
{
  "id": 795296435386388500,
  "text": "#FCB Spiel heute Abend! #bl",
  "created_at": "11-06-2016 16:07:00",
  "tags": [{
    "tag": ”FCB"
    }, {
    "tag": ”bl"
    }]
}
```

id	text	created_at	tag
795…	#FCB…	11-06-2016…	FCB
795…	#FCB…	11-06-2016…	bl

Source data for
Amazon Kinesis Analytics

How do we filter unwanted tweets?



Use PUMP to insert filtered data into STREAM

```
CREATE OR REPLACE PUMP ”BL_TWEET_PUMP" AS
    INSERT INTO ”BL_TWEET_STREAM"
        SELECT STREAM "id", "text", LOWER("tag")
        FROM "SOURCE_STREAM"
        WHERE LOWER("tag") NOT IN (‘bl’,‘bundesliga’);
```

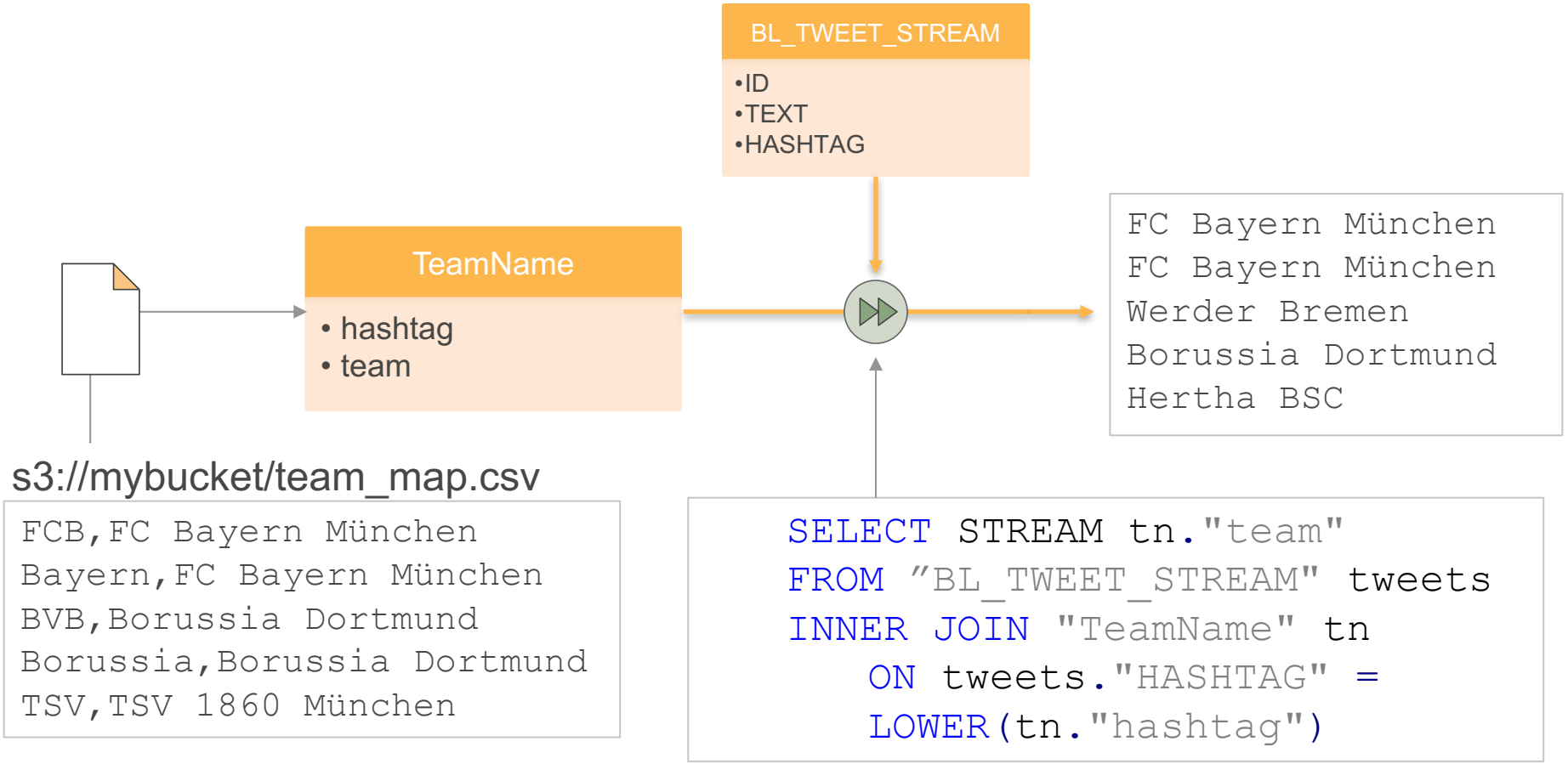
How do we get team name from the hashtag?

- Create CSV file with hashtag to team name map in S3
- Configure Amazon Kinesis Analytics application to import file as reference data
- Reference data appears as a table
- Join streaming data on reference data

s3://mybucket/team_map.csv

```
hashtag,team
FCB,FC Bayern München
Bayern,FC Bayern München
BVB,Borussia Dortmund
Borussia,Borussia Dortmund
TSV,TSV 1860 München
```

Use Reference Data in Query

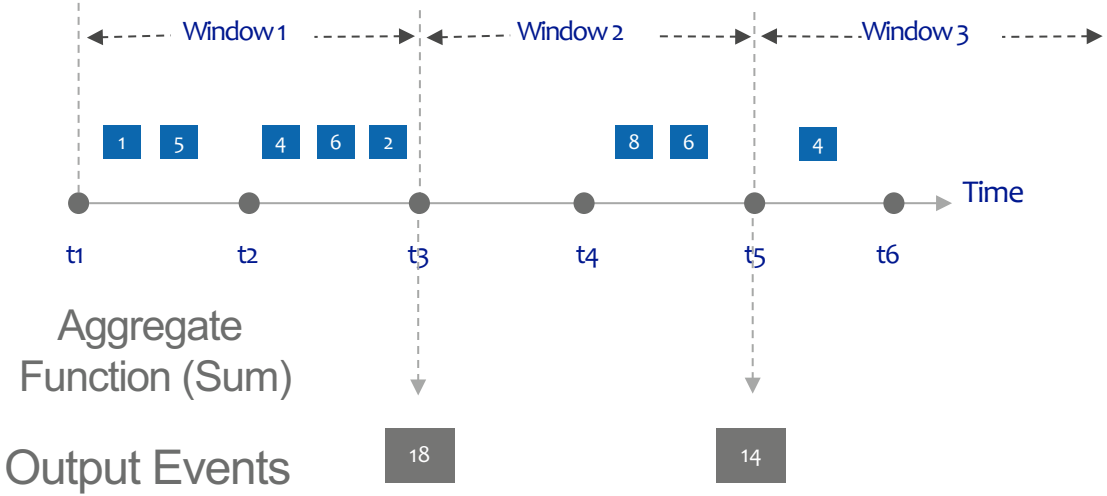


How do we aggregate streaming data?

- A common requirement in streaming analytics is to perform set-based operation(s) (`count, average, max, min`,..) over events that arrive within a specified period of time
- Cannot simply aggregate over an entire table like typical static database
- How do we define a subset in a potentially infinite stream?
- Windowing functions!

Windowing Concepts

- Windows can be **tumbling** or **sliding**
- Windows are fixed length



Output record will have the timestamp of the end of the window

How do we aggregate team mentions per hour?

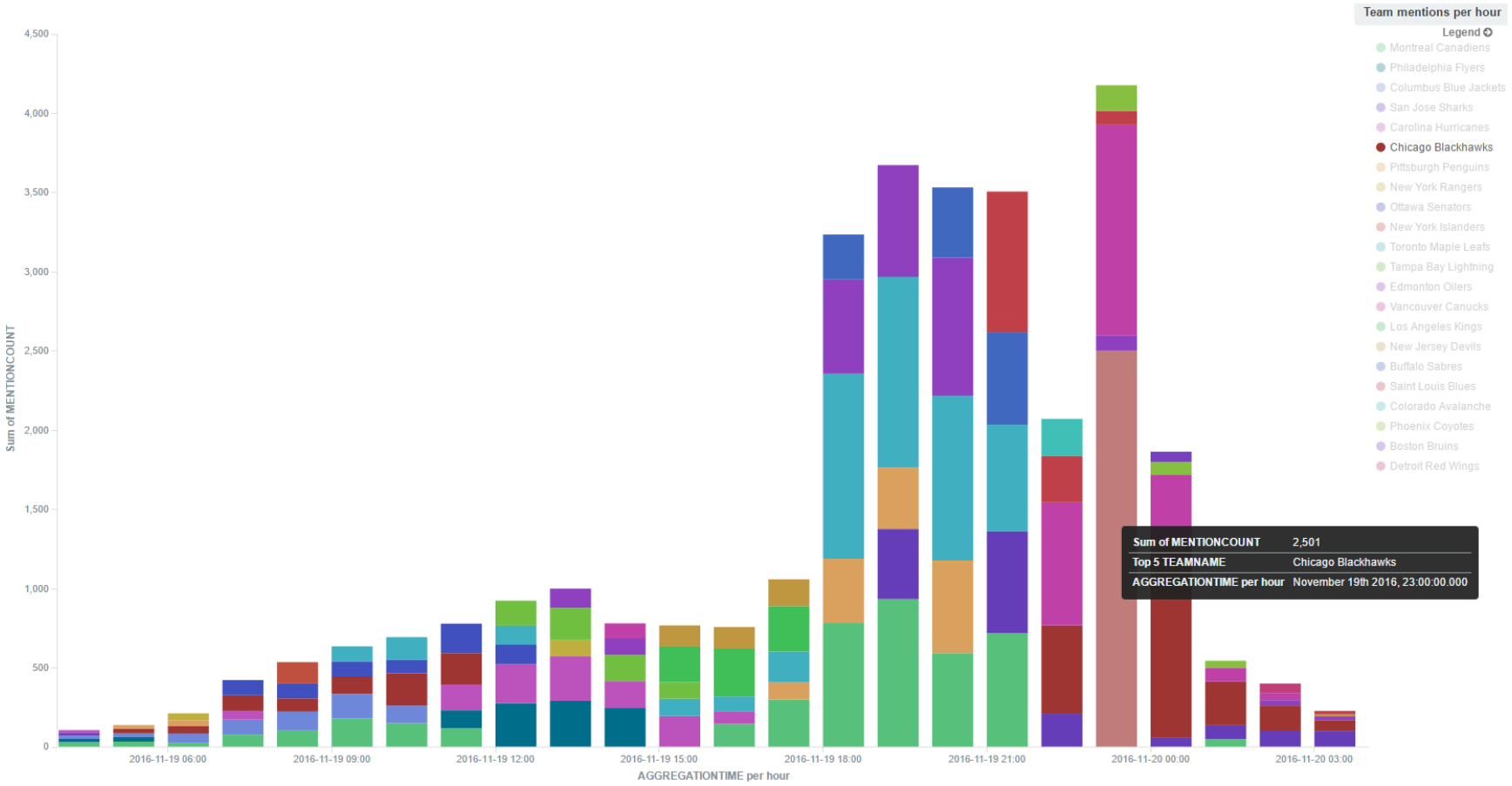
- Use `TOP_K_ITEMS_TUMBLING` function
- Pass cursor to team name stream
- Define window size of 3600 seconds

```
INSERT INTO "MENTION_COUNT_STREAM"
 SELECT STREAM *
    FROM TABLE(TOP_K_ITEMS_TUMBLING(
      CURSOR(SELECT STREAM tn."team"... ),
      'teamname', -- name of column to aggregate
      5, -- number of top items
      3600 -- tumbling window size in seconds
      ));
```

Output to Amazon Kinesis Firehose

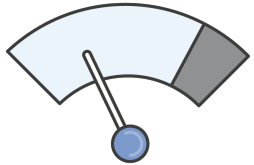

```
{
    "aggregationtime": "2016-11-06T14:42:03.335",
    "teamname": "Hertha BSC",
    "mentioncount": 604
}
```

Visualize Results with Kibana



Amazon Kinesis Analytics Best Practices

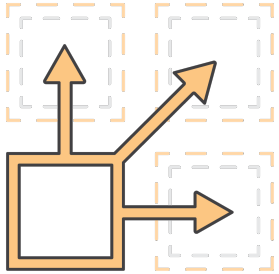
Managing Applications



Set up Cloudwatch Alarms

- `MillisBehindLatest` metric tracks how far behind the application is from the source
- Alarm on `MillisBehindLatest` metric. Consider triggering when 1-hour behind, on a 1-minute average. Adjust accordingly for applications with lower end-to-end processing needs.

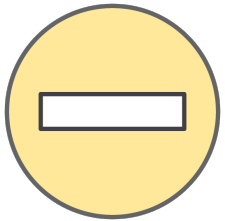
Managing Applications



Increase input parallelism to improve performance

- By default, a single source in-application stream is created
- If application is not keeping up with input stream, consider increasing input parallelism to create multiple source in-application streams

Managing Applications



Limit number of applications reading from same source

- Avoid `ReadProvisionedThroughputExceeded` exceptions
- For an Amazon Kinesis Streams source, limit to 2 total applications
- For an Amazon Kinesis Firehose source, limit to 1 application

Defining Input Schema

- Review and adequately test inferred input schema
- Manually update schema to handle nested JSON with greater than 2 levels of depth
- Use SQL functions in your application for unstructured data

Authoring Application Code

- Avoid time-based windows greater than one hour
- Keep window sizes small during development
- Use smaller SQL queries, with multiple in-application streams, rather than a single, large query

AWS

SUMMIT

Thank you!

amazon web services