



Hops



@hopshadoop



<http://github.com/hopshadoop>



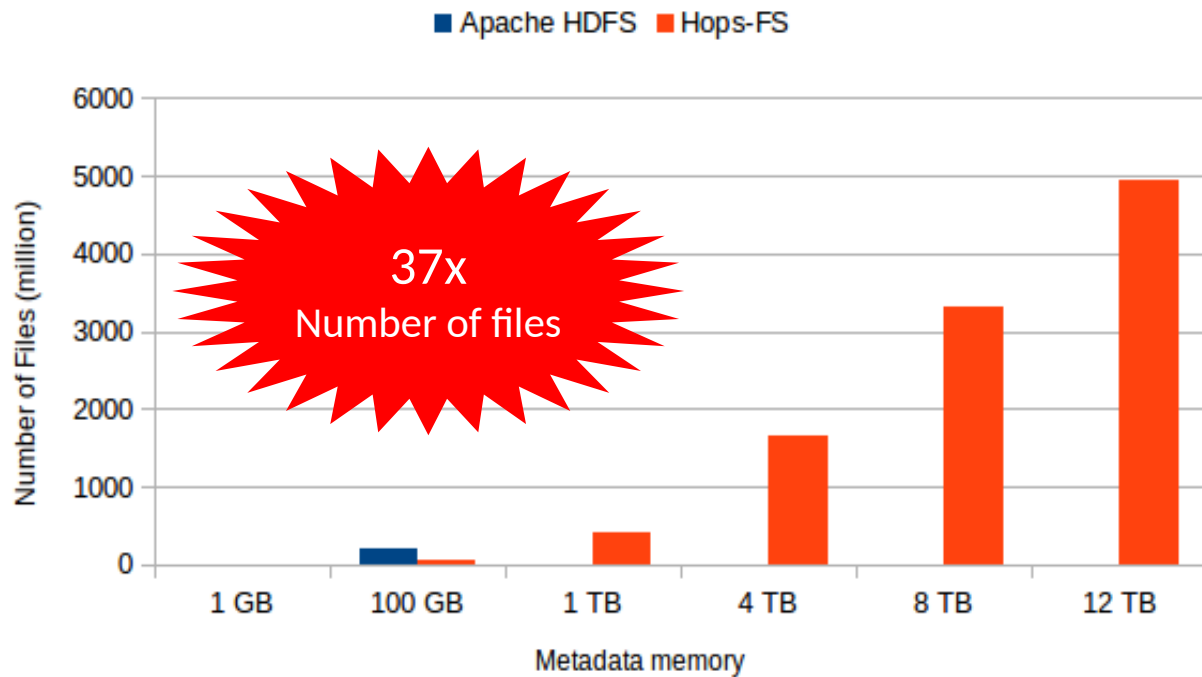
<http://www.hops.io>

# Hopsworks: Secure Spark/Flink/Kafka-as-a-Service

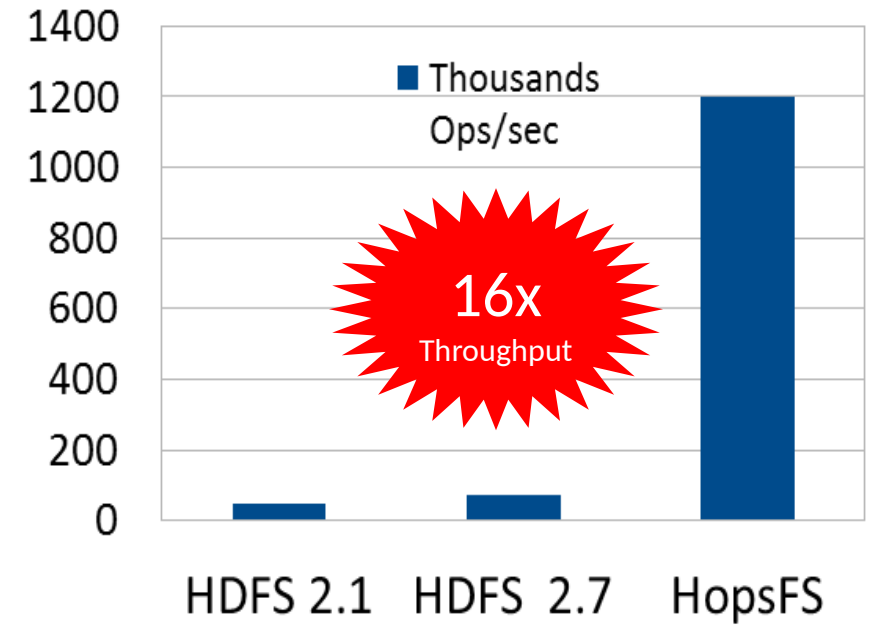
LOGICAL  
CLOCKS

Jim Dowling  
Associate Prof @ KTH  
CEO @ Logical Clocks AB

# HopsFS: Next Generation HDFS\*



Bigger



Faster

 **IEEE** Scale Challenge Winner (2017)

\*<https://www.usenix.org/conference/fast17/technical-sessions/presentation/niazi>

# Streaming-as-a-Service in Sweden

- **SICS ICE**
- Datacenter research environment
- **Hopsworks**  
Spark/Flink/Kafka/Tensorflow-as-a-Service
  - Built on HopsFS/YARN
  - >150 active users



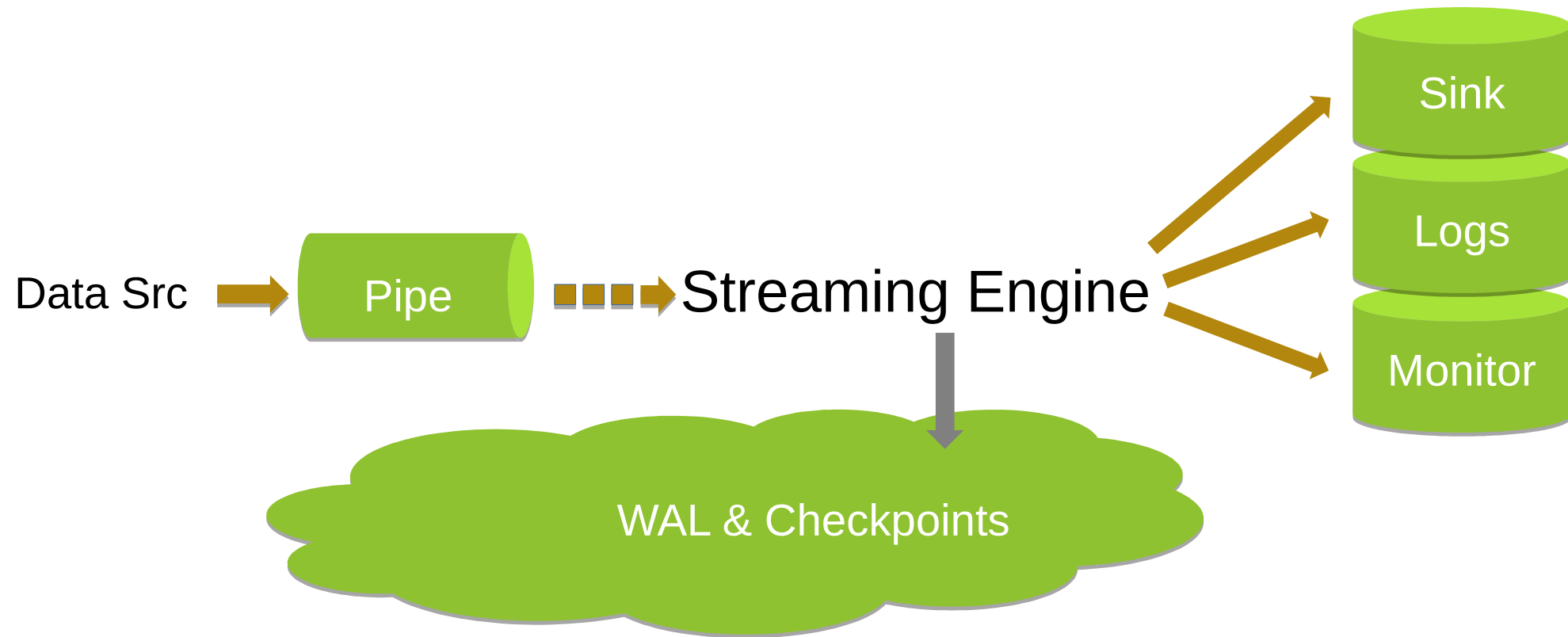
# Self-Service Streaming-as-a-Service

I want to Spark Up,  
all by myself with  
a piece of Flink.





# Basic Services needed for Stream Processing



# Smoking Self-Service Streaming

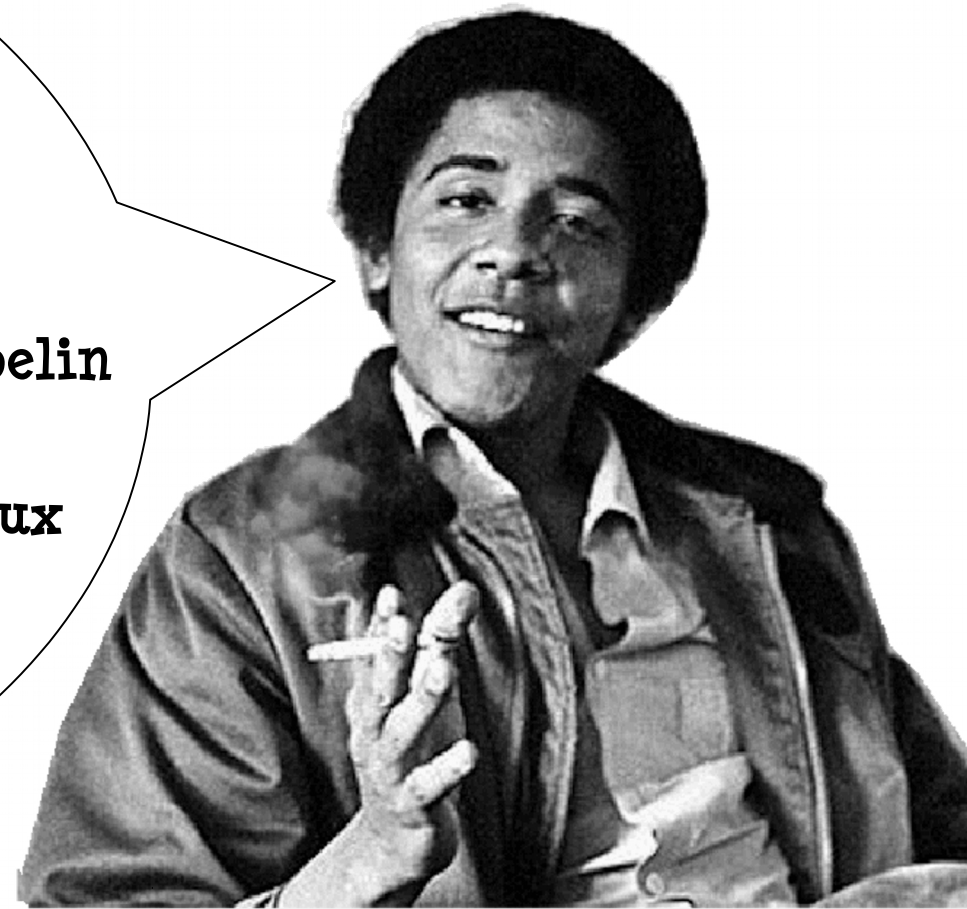
A pipe for my data: **Kafka**

A cloud for my smoke: **Hops Hadoop**

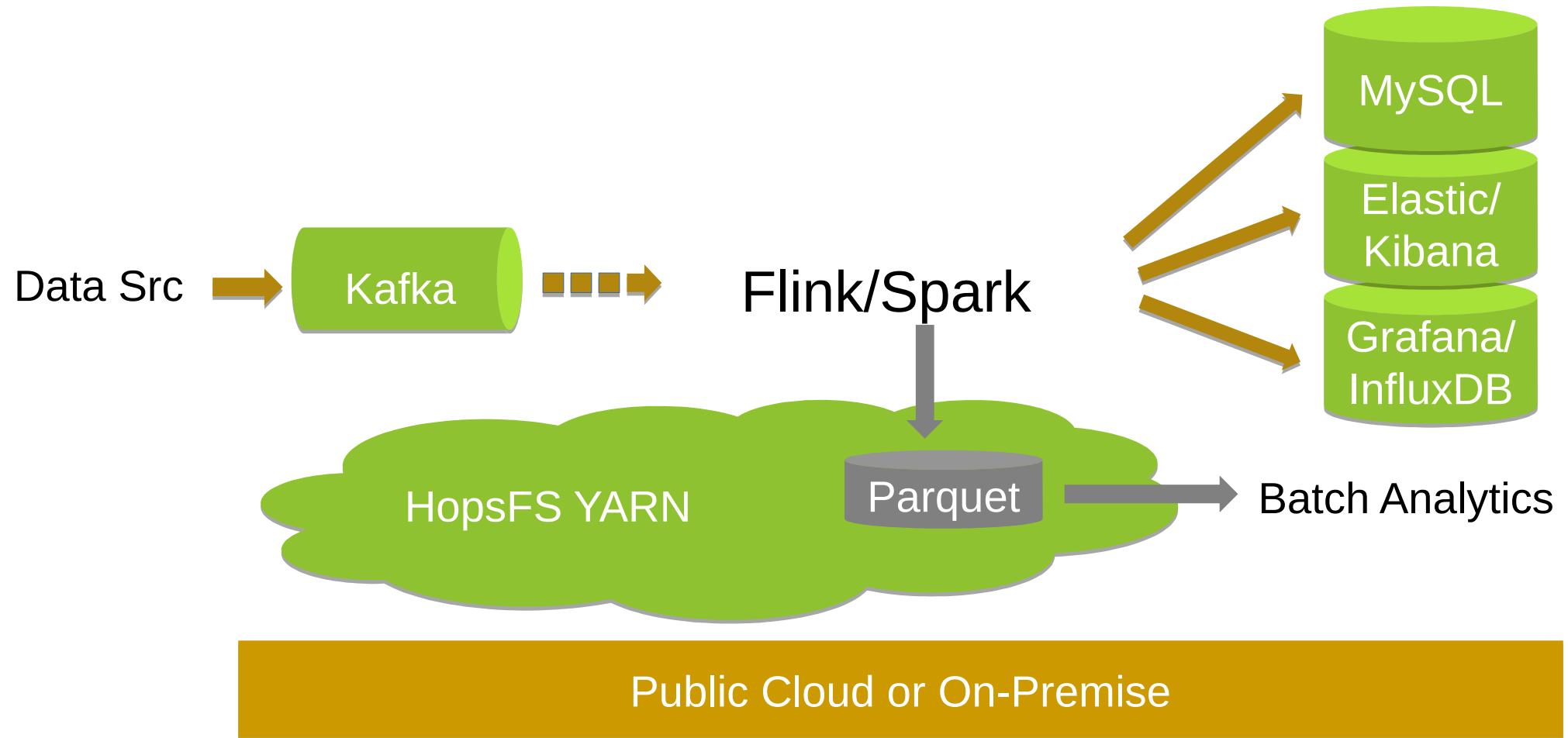
A way to roll A/B tests: **Jupyter/Zeppelin**

A lookout for trouble: **Grafana/Influx**

A log with evidence: **ELK Stack**



# Hopsworks



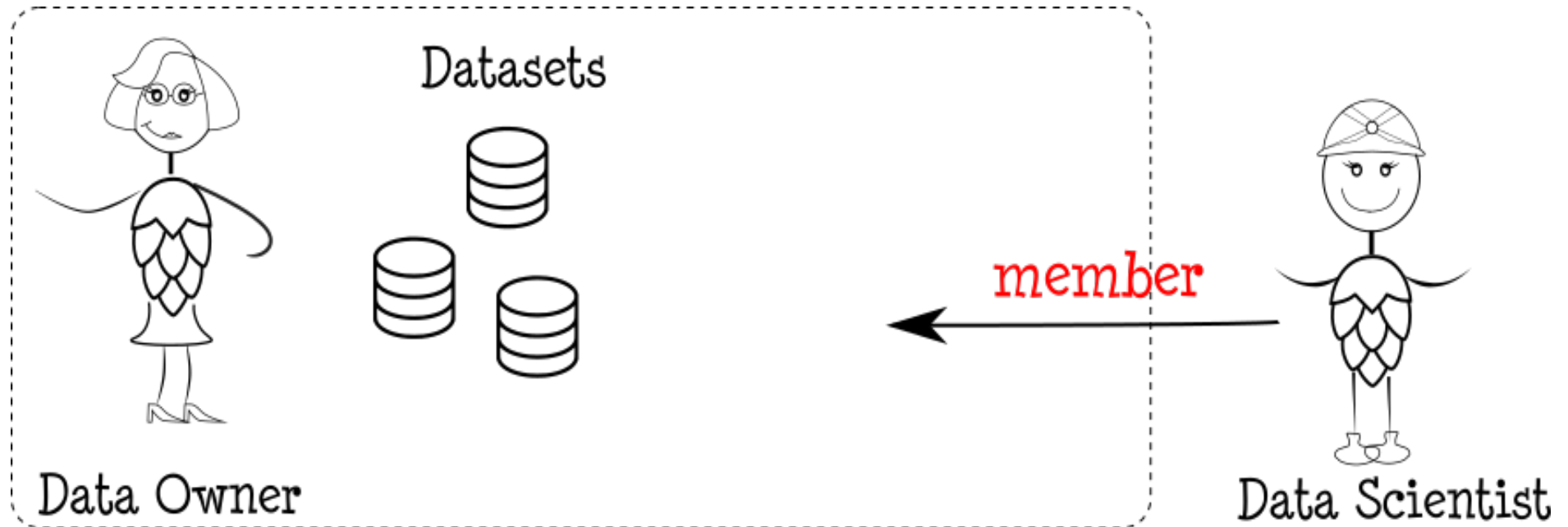
# General Data Protection Regulation (GDPR)

**Ostrich Day: 2018-05-25**

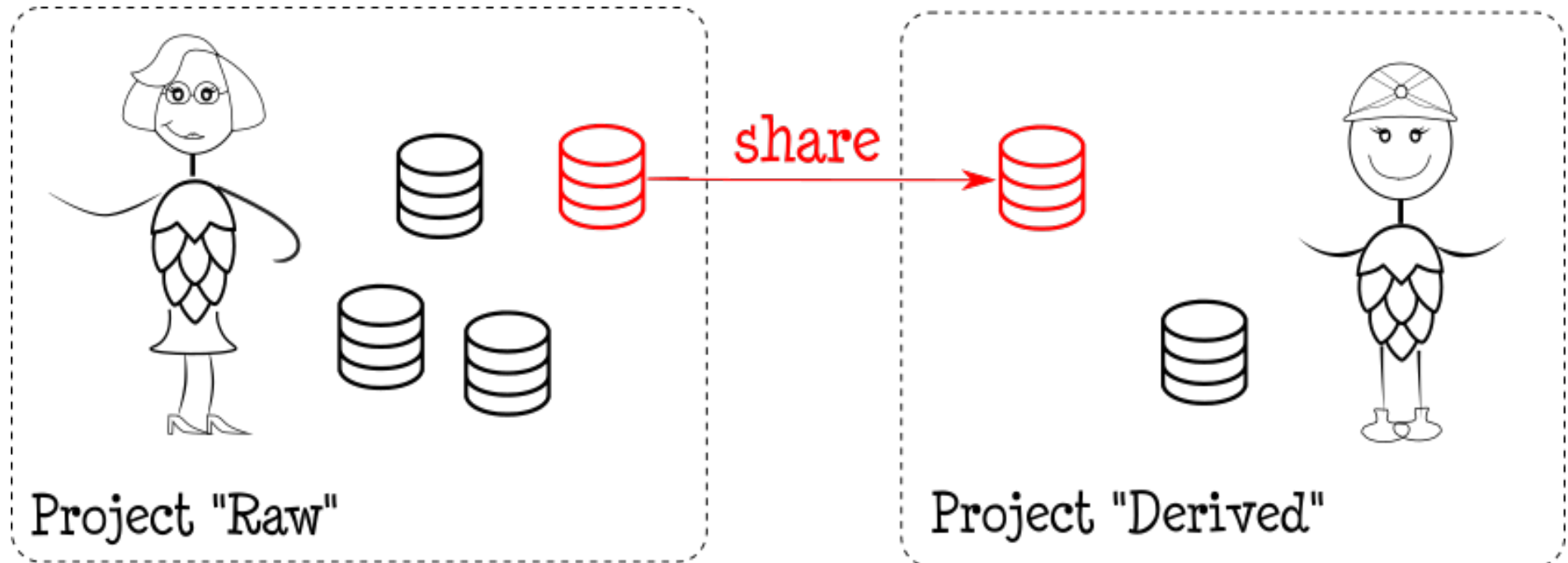


<http://www.computerweekly.com/news/450295538/D-Day-for-GDPR-is-25-May-2018>

# Manage Projects like GitHub

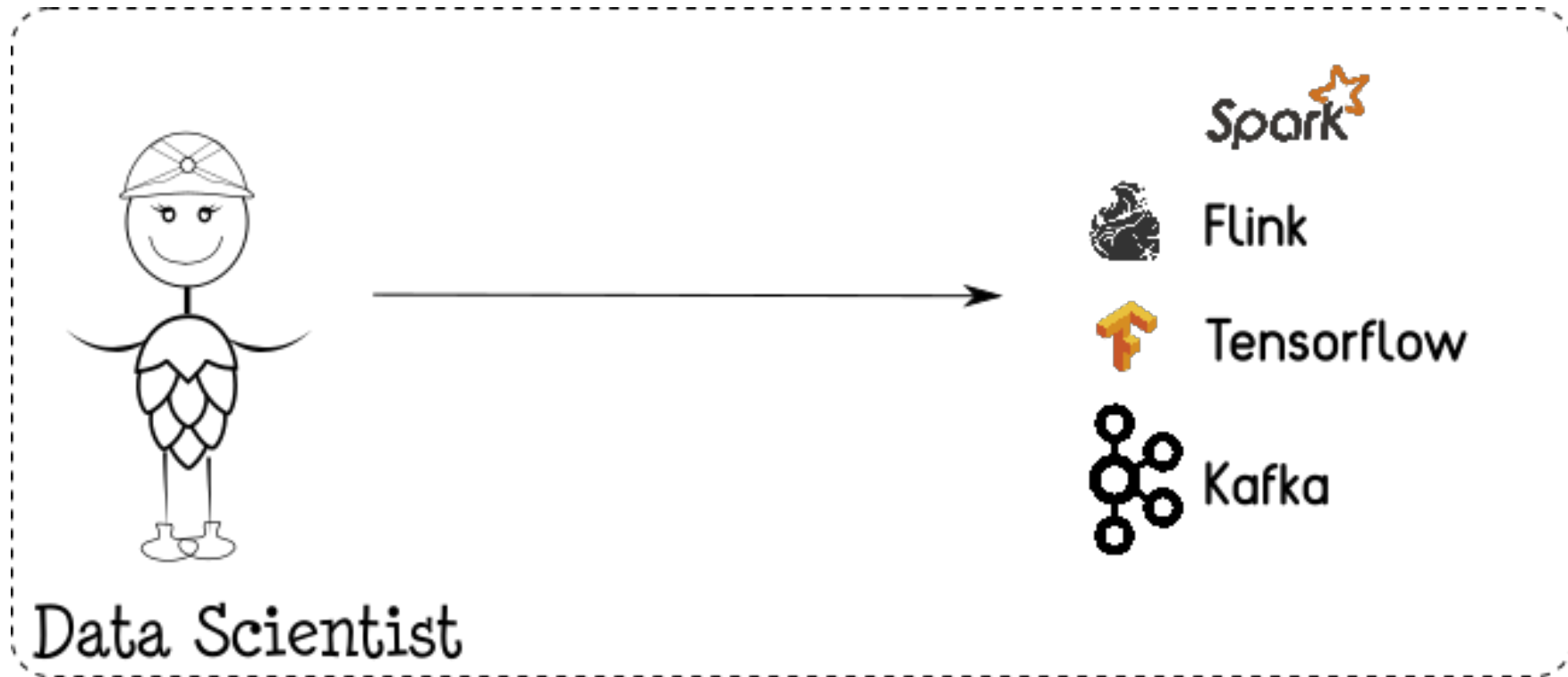


# Share like in Dropbox

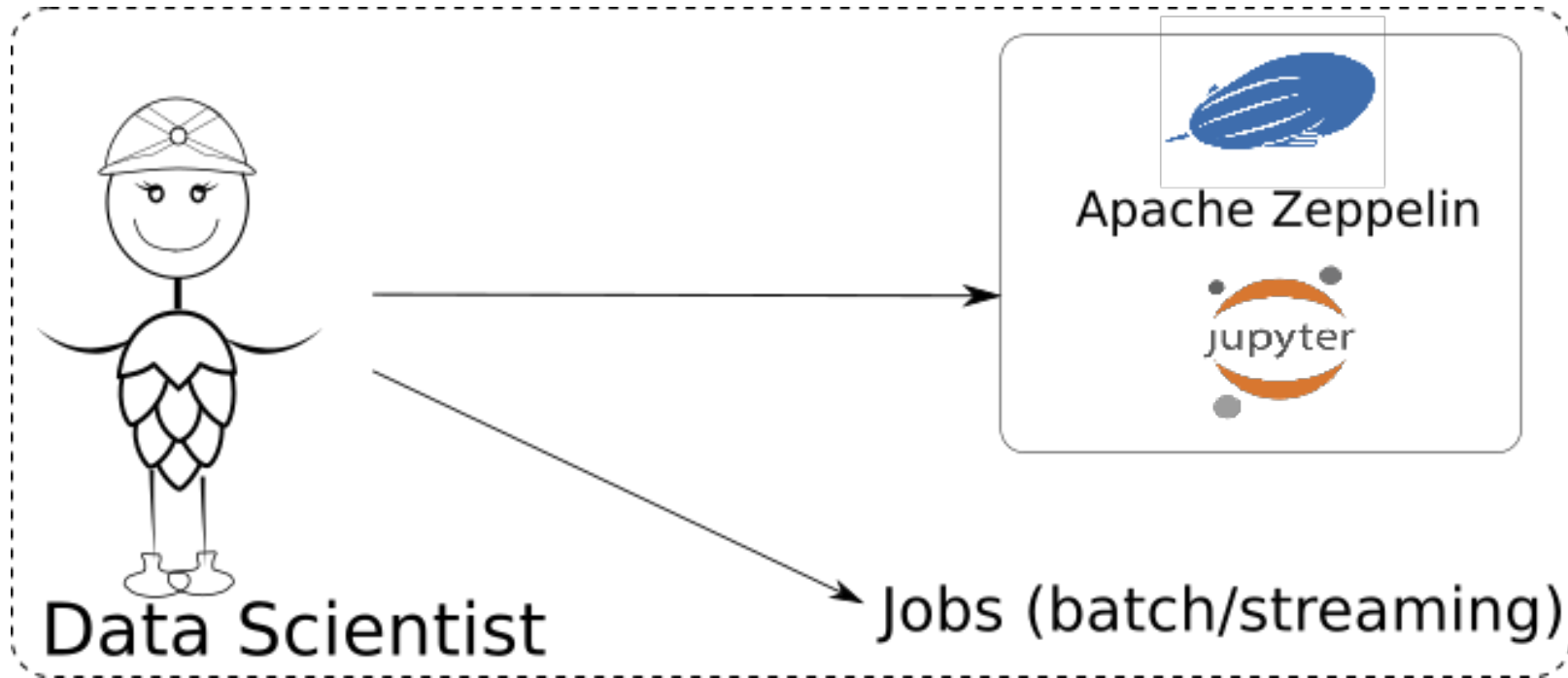


Share any Data Source/Sink: HDFS Datasets, Kafka Topics, etc

# Only Modern Data Parallel Platforms

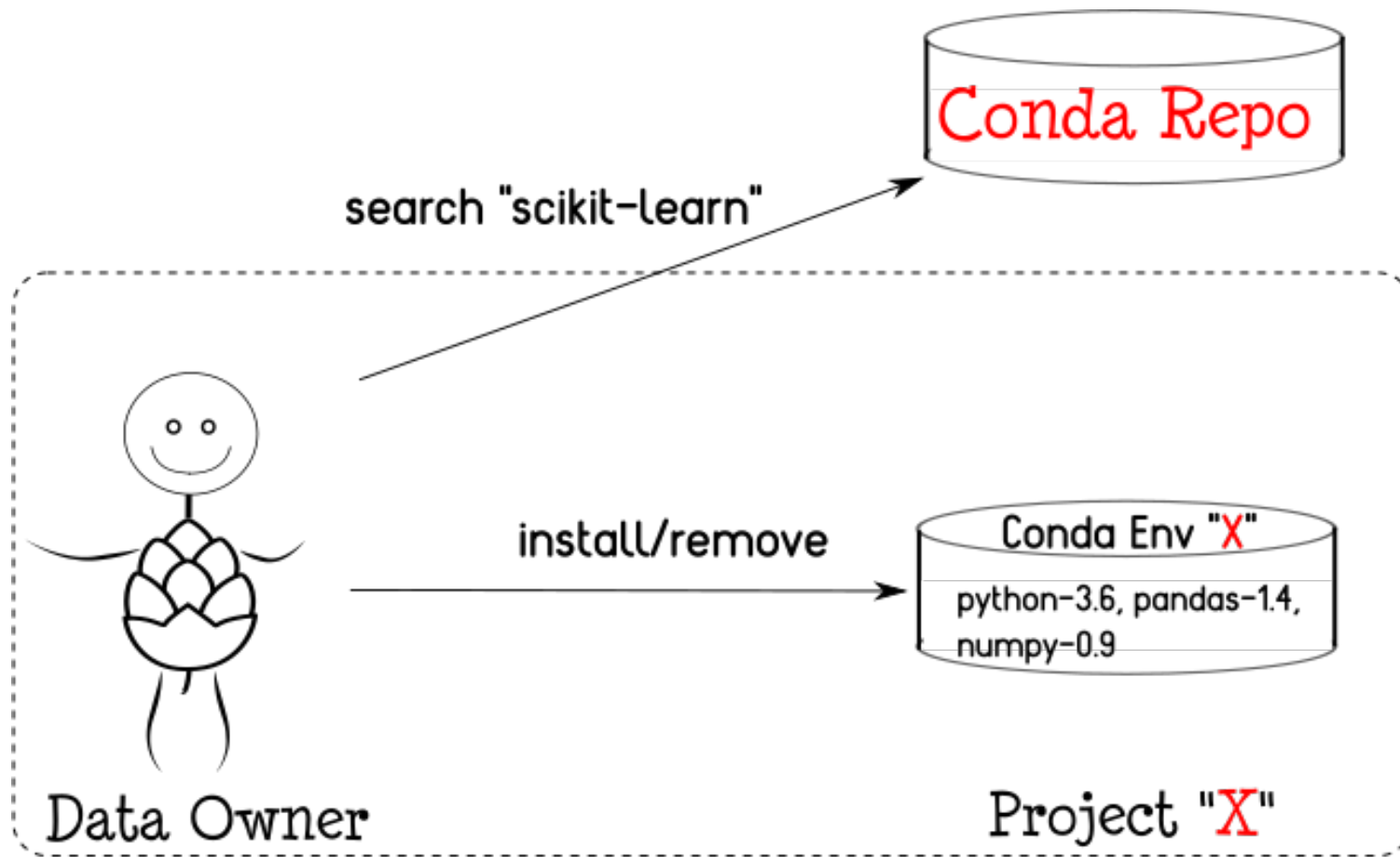


# Workflow/Jobs and Notebook Support





# Custom Python Environments with Conda

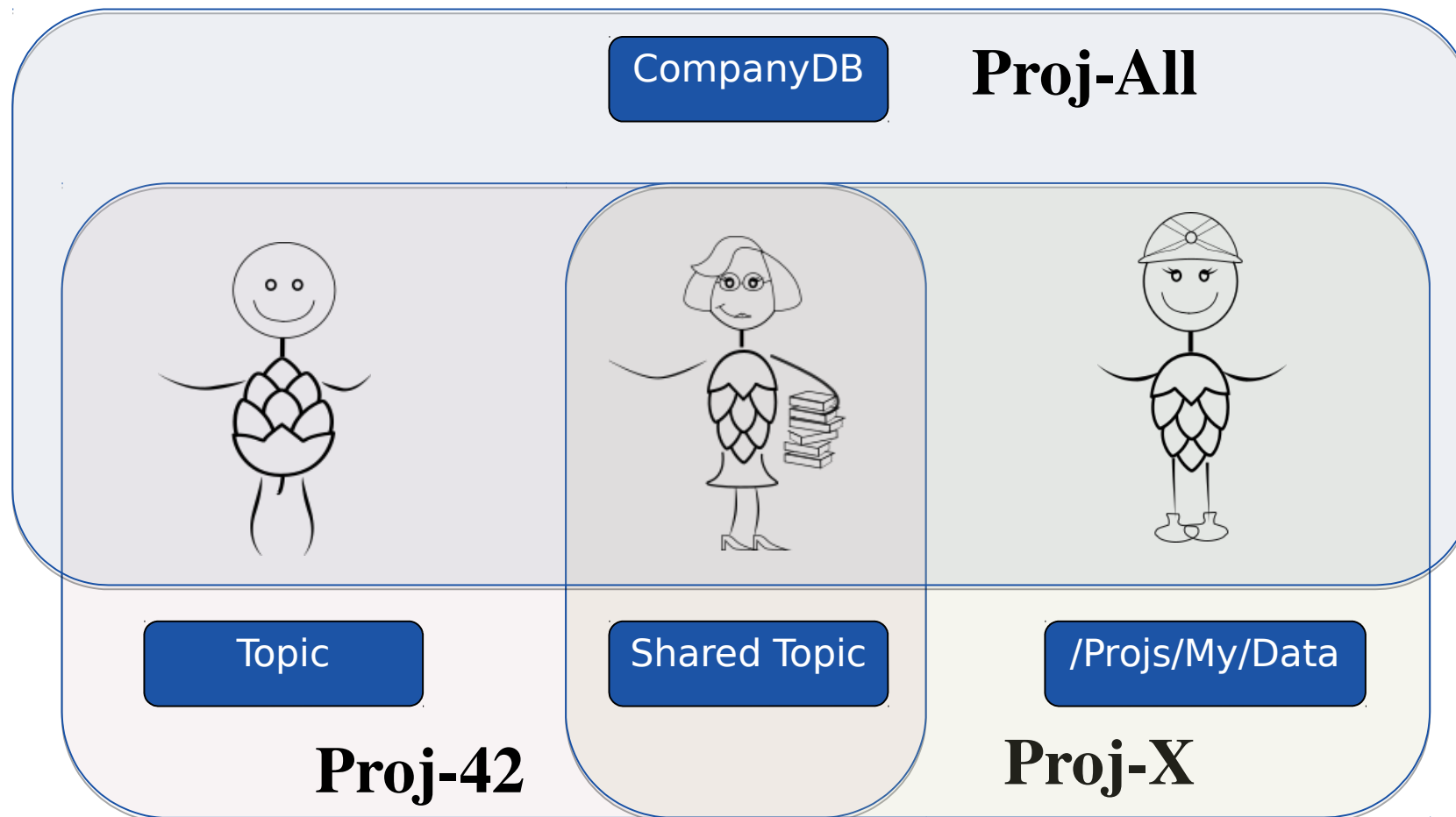


Python libraries are usable by Spark/Tensorflow

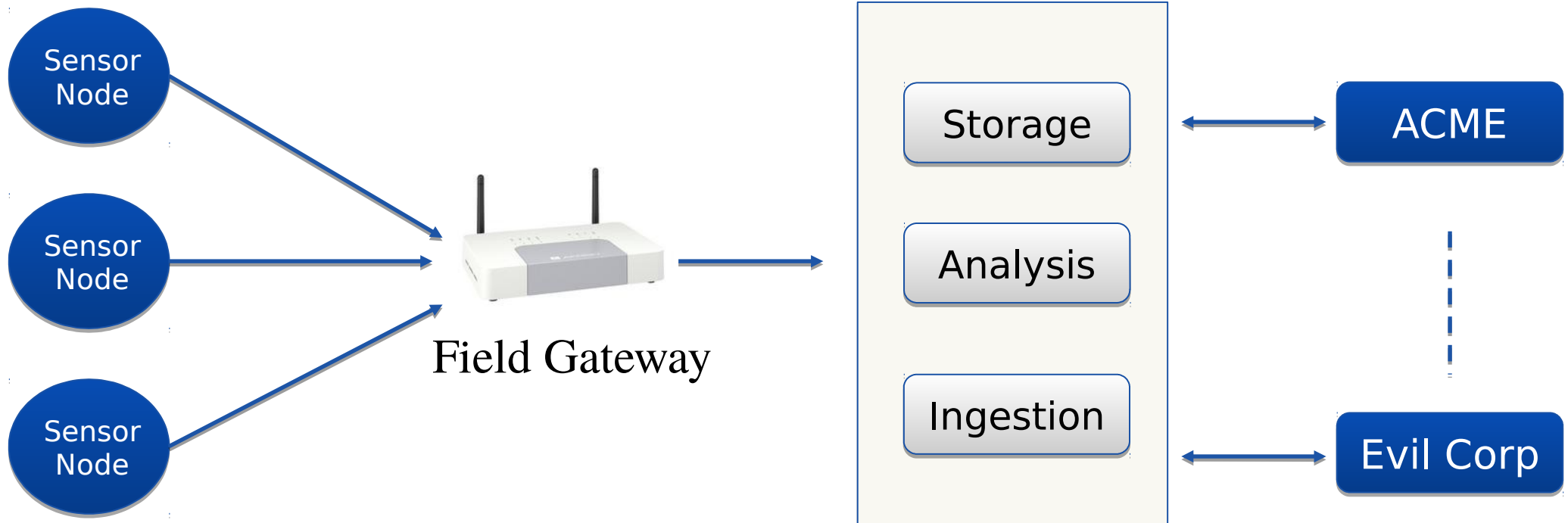
# Privacy-by-Design with Projects, Data, Users

# Projects for Software-as-a-Service

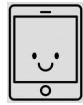
A **Project** is a Grouping of **Users** and **Data**



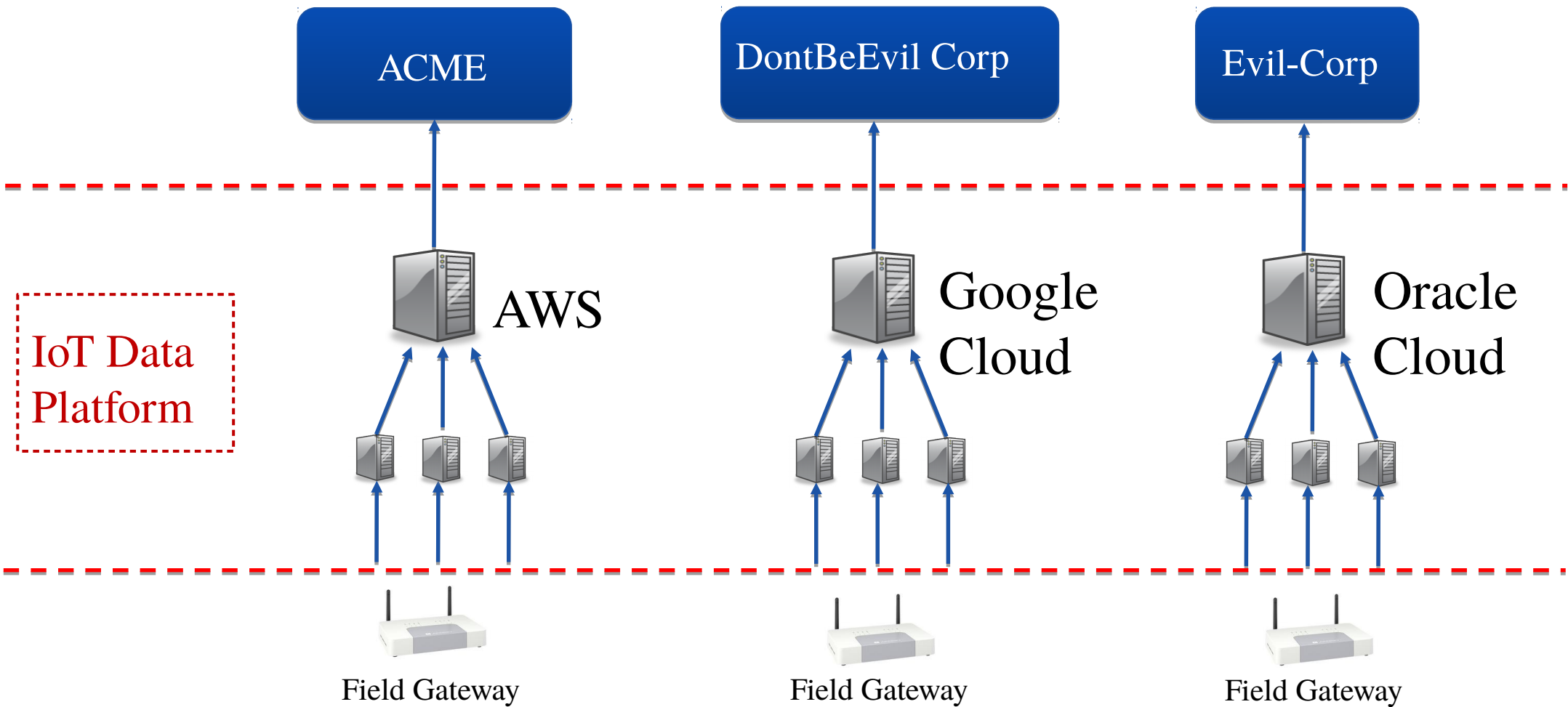
# SaaS IoT Data Platform



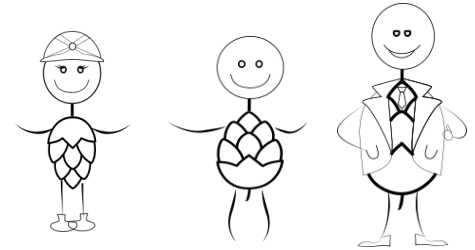
# Multi-Cloud SaaS IoT Data Platform



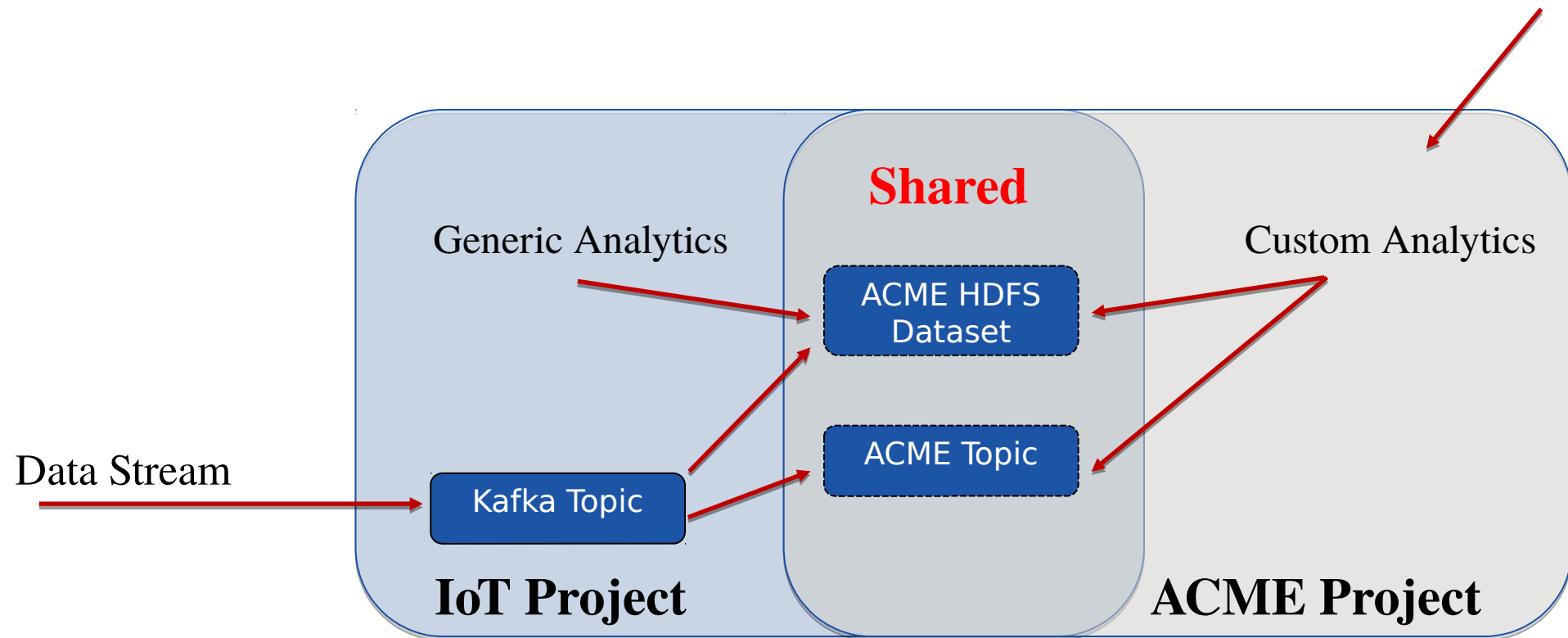
User Apps control IoT Devices



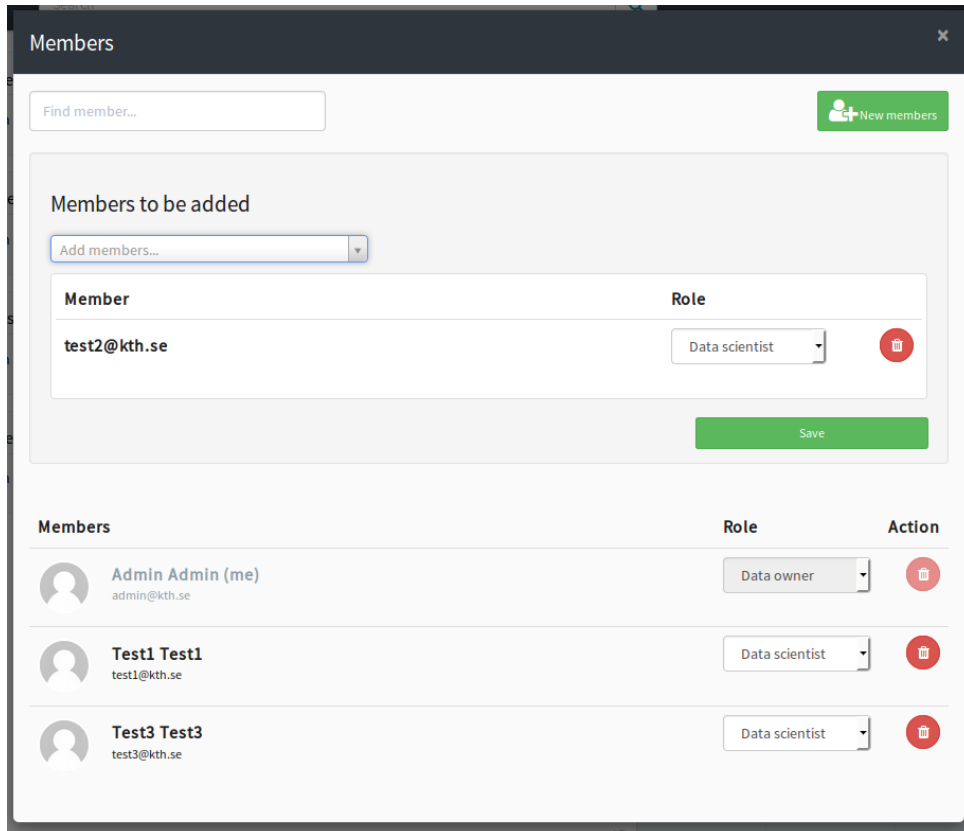
# SaaS IoT Platform: Project per Customer



ACME manage membership



# Project Roles



- Data Owner Privileges
  - Import/Export data
  - Manage Membership
  - Share DataSets, Topics
- Data Scientist Privileges
  - Write and Run code

**We delegate administration of privileges to users**

# Project Quotas

- Per-Project quotas
  - Storage in HDFS
  - CPU in YARN (Uber-style Pricing)
- Sharing is not Copying
  - Datasets/Topics

SURGE PRICING

✘

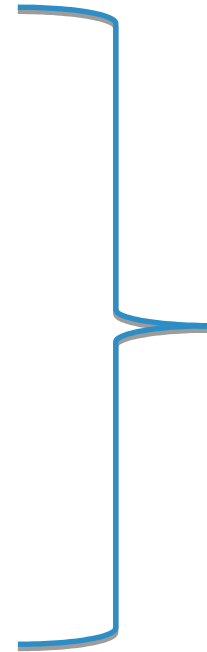
Demand is off the charts! Rates have increased to get more Ubers on the road.





# Tooling for Streaming

- Hops Hadoop
- Apache Kafka
- ELK Stack
- Grafana/InfluxDB
- Jupyter/Apache Zeppelin



**Hopsworks**  
Self-Service

# Kafka Self-Service UI

The screenshot displays the HopsWorks Kafka Self-Service UI. The interface includes a dark sidebar on the left with navigation options: producer, Zeppelin, Jobs, Jobs History, Kafka, Data Sets, Settings, Members, and Metadata Designer. At the bottom of the sidebar, a cluster utilization bar shows 20% usage. The main content area is titled 'Topics' and shows '1 of 10 topics in use' with a 'New Topic +' button. The selected topic is 'hellotopic', which has a schema of 'kafka-schema (1)'. Below this, there are tabs for 'ACL', 'Share', 'Advanced', and 'Remove'. The ACL tab is active, showing a table of permissions:

Project	UserEmail	Permission	Operation	Host	Role	Remove	Edit
producer	admin@kth.se	allow	*	*	*		
consumer	tkak@kth.se	allow	*	*	*		

Below the ACL table, there is a table showing partition details:

Partition id	Partition leader	Partition replicas	Insync replicas
1	10.0.2.15	["10.0.2.15"]	["10.0.2.15"]
0	10.0.2.15	["10.0.2.15"]	["10.0.2.15"]

On the right side of the screenshot, there is a green callout box with the text 'Manage & Share' and a bulleted list:

- Topics
- ACLs
- Avro Schemas

# Tuning Kafka/Spark Streaming

- Key Kafka Tuning Parameters
  - Number of Topics
  - Number of Partitions per Topic
- Spark Streaming Tuning Considerations
  - Match # of Executors to the # of Partitions
  - Ensure balanced data across partitions

# Realtime Logs

- YARN aggregates logs on job completion
  - No good to us for Streaming
- Collect logs and make them searchable in real-time using Logstash, Elasticsearch, and Kibana
  - Log4j auto-configured to write to Logstash

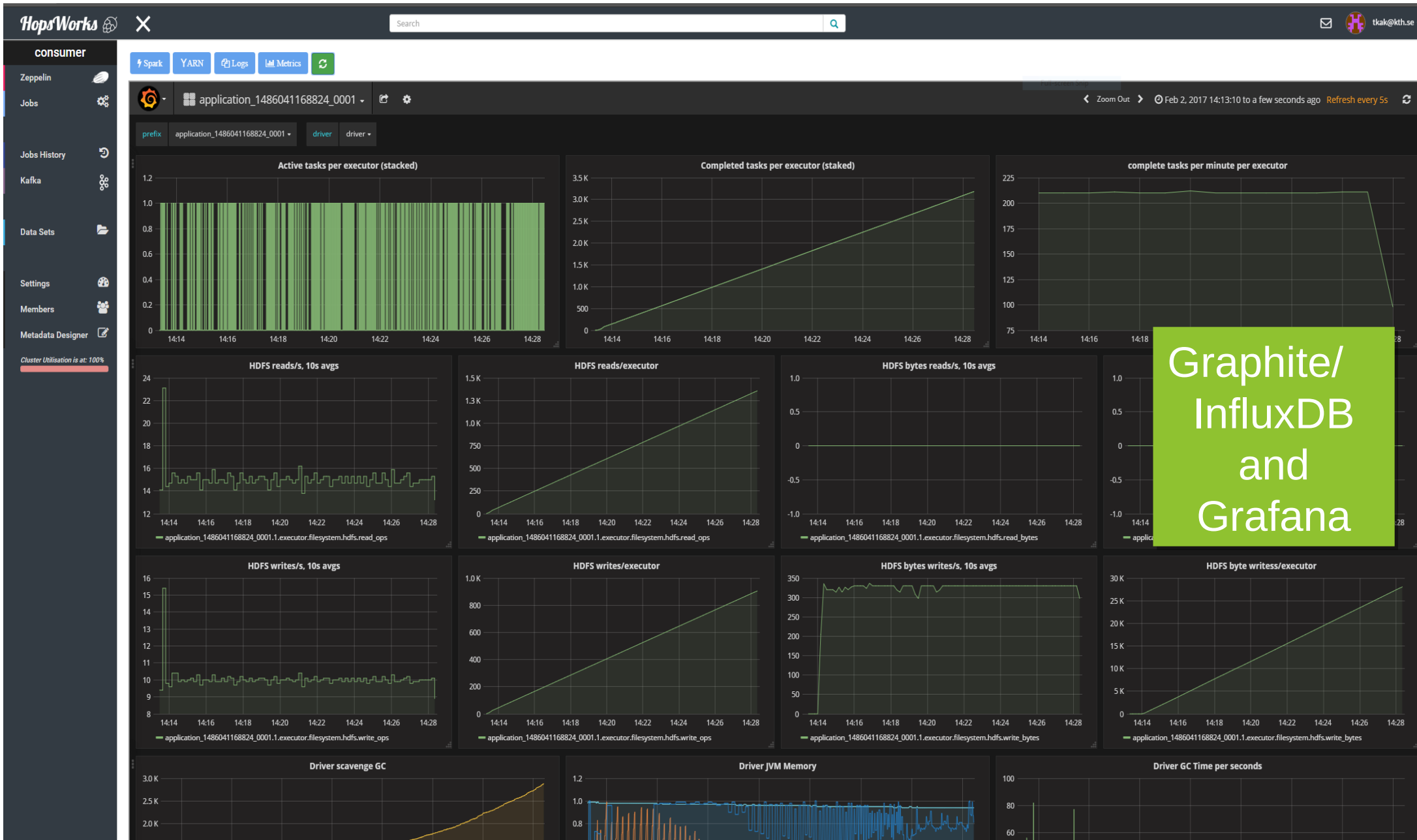
<http://mkuthan.github.io/blog/2016/09/30/spark-streaming-on-yarn/>



# Resource Monitoring/Alerting

- `$SPARK_HOME/conf/metrics.properties`
  - Different sinks supported
    - JMX, **Graphite**, Servlet/JSON, CSV, Console, Slf4j
- `StructuredQueryListener`
  - Send query progress to a Kafka topic for inspection
- `StreamingQueryListener`
  - Asynchronous Monitoring of all queries for a Spark session

# Resource Monitoring/Alerting



# Zeppelin Notebooks


The screenshot displays the HopsWorks interface for Zeppelin Notebooks. At the top, there is a search bar and a user profile for 'admin@kth.se'. A sidebar on the left contains navigation options: Zeppelin, Jobs, Jobs History, Kafka, Data Sets, Settings, Members, and Metadata Designer. The main area shows a 'Create New Notebook' button and a 'Goto Zeppelin' button. Below these are three notebook cards: one with a Zeppelin logo, one with a code icon labeled 'ff', and one with a plus sign labeled 'Create New Notebook'. On the right, a panel titled 'INTERPRETERS' shows a list of interpreters: flink Interpreter (running), angular Interpreter (stopped), livy Interpreter (stopped), spark Interpreter (stopped), and md Interpreter (stopped). The central notebook window shows a 'Load Data Into Table' task with three SQL queries and their visualizations:


- Query 1:** `select age, count(1) value from bank group by age order by age`. The histogram shows the distribution of ages, with a peak around 48.
- Query 2:** `select age, job, count(1) value from bank where age < ${maxAge=30} group by age, job order by age`. The histogram shows the distribution of jobs for ages under 30.
- Query 3:** `select age, count(1) value from bank where marital=${marital=single, single|divorced|married} group by age`. The histogram shows the distribution of ages for a specific marital status.

**i** Running a paragraph in a notebook will automatically start the necessary interpreters for that job.




# Jupyter Notebooks

**HopsWorks** X Search 

**Demo**  **Jupyter** Demo Last Checkpoint: 2 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help PySpark

 Code CellToolbar

```
In [5]: df = spark.read.csv('hdfs://192.168.56.101:8020/Projects/Demo/DemoData/mcs_201611.csv', \
                             sep=';', schema=schema, ignoreLeadingWhiteSpace=True, \
                             ignoreTrailingWhiteSpace=True, \
                             timestampFormat='yyyy-MM-dd HH:mm:ss.SSS')
```

```
In [6]: df.count()
85275962
```

```
In [7]: df.createOrReplaceTempView("FlowData")
```

```
In [9]: %%sql -o query1]
SELECT Timestamp, Ds_Reference, Detector_Number, Flow_In, Average_Speed FROM FlowData WHERE Status == 3
```

Type: **Table** Pie Scatter Line Area Bar

Encoding:

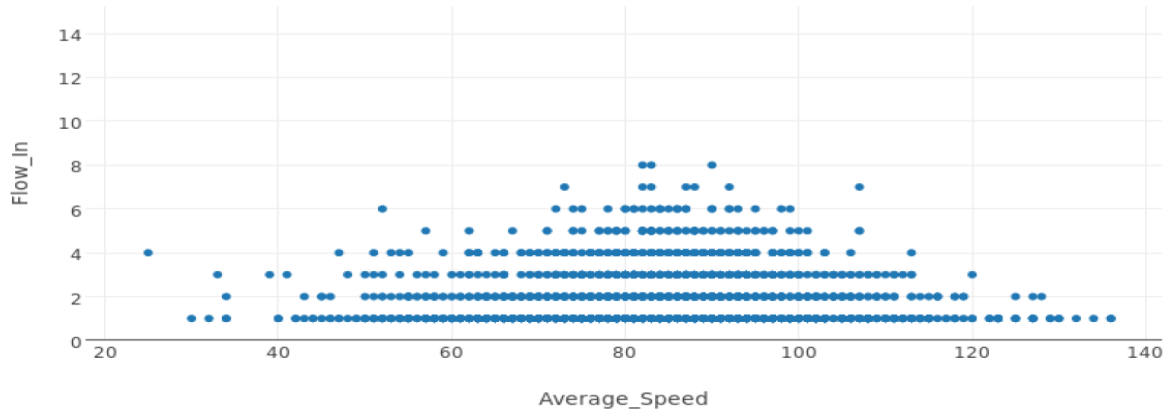
X: Average\_Speed

Y: Flow\_In

Func: -

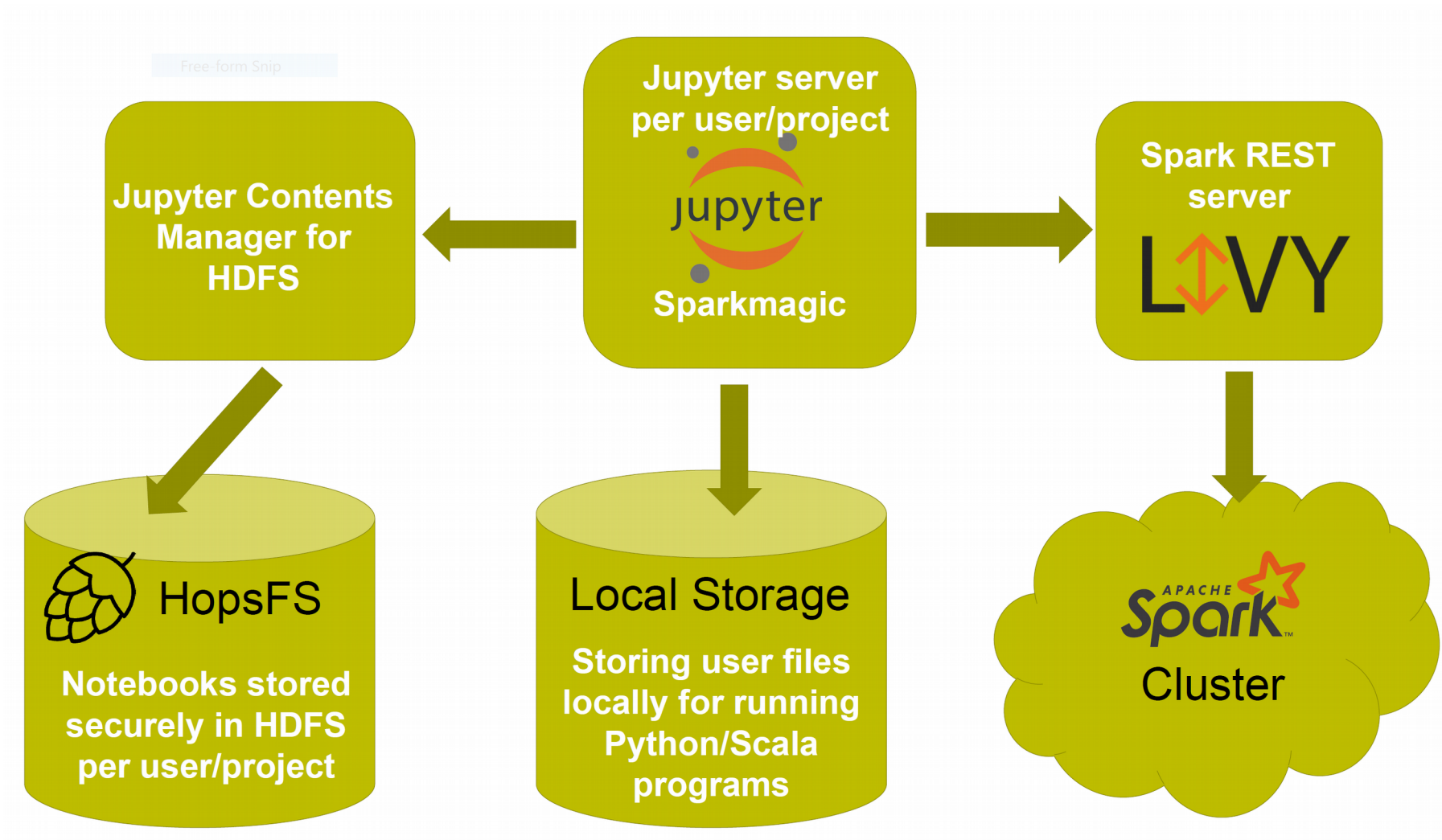
Log scale X

Log scale Y



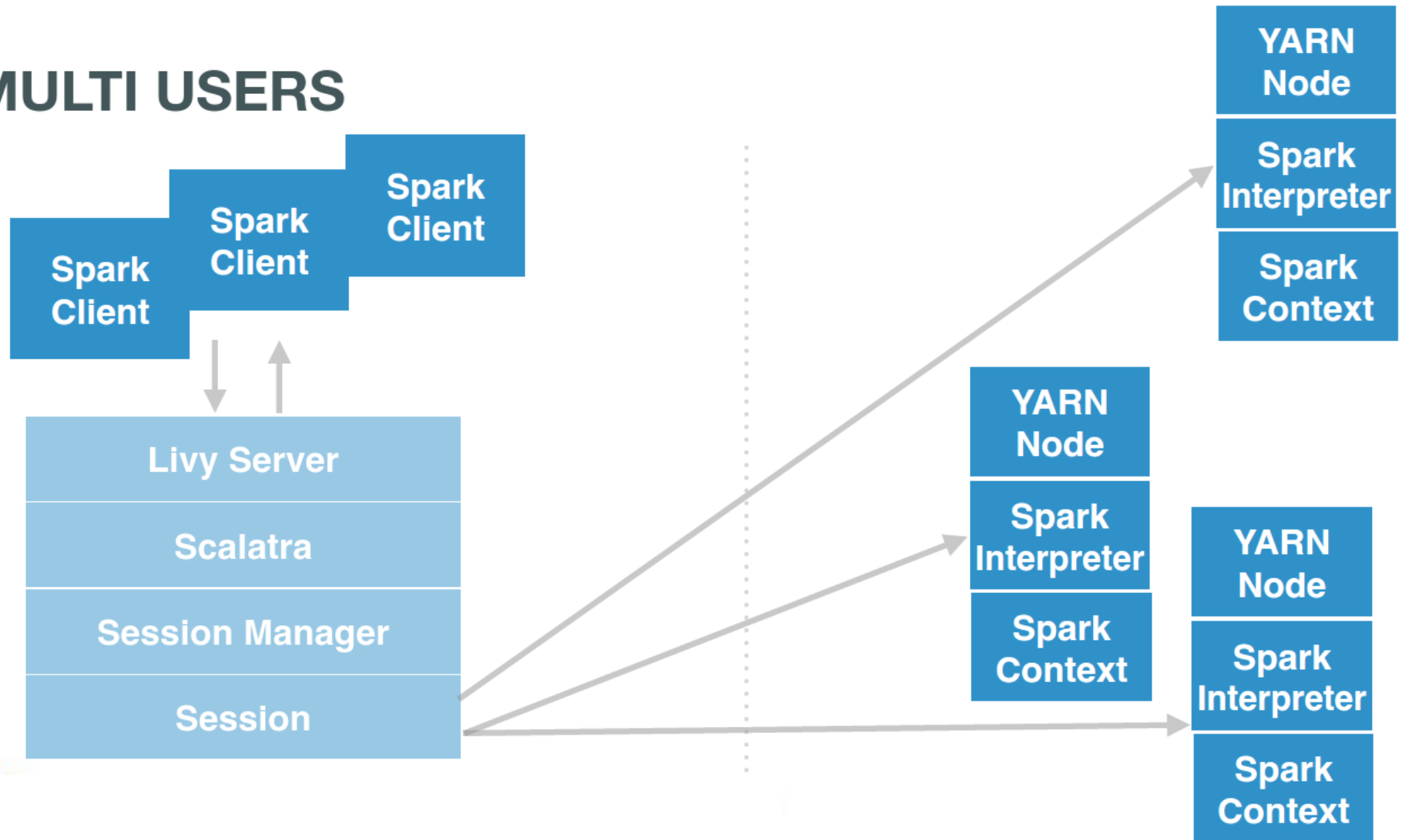
The scatter plot displays the relationship between Average\_Speed (X-axis, ranging from 20 to 140) and Flow\_In (Y-axis, ranging from 0 to 14). The data points are blue dots, showing a dense distribution of points across the range of Average\_Speed, with Flow\_In values generally between 1 and 8. There are some outliers with higher Flow\_In values, up to 14, scattered across the Average\_Speed range.

# Jupyter Support



# Livy REST Server for Spark

## MULTI USERS



# Basic Structured Spark Streaming Program

- Query
- Input source
- Output sink
  - Mode: append/complete/update
- Trigger Interval
- Checkpoint Location

```
val cloudtrailEvents = ...

val streamingETLQuery =
  cloudtrailEvents
    .withColumn("date",
      $"timestamp".cast("date"))
    .writeStream
    .trigger(ProcessingTime
      ("10 seconds"))
    .format("parquet")
    .partitionBy("date")
    .option("path", "/cloudtrail")
    .option("checkpointLocation",
      "/cloudtrail.checkpoint/")
    .start()
```

# Secure Structured Spark Streaming App

- Real Streaming Apps are more complex:
  - Credentials, Endpoints
  - monitoring.properties
  - How to shutdown gracefully
- The **HopsUtil API** hides this complexity.

# HopsUtil simplifies Secure Spark/Kafka

```
Properties props = new Properties();
props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, brokerList);
props.put(SCHEMA_REGISTRY_URL, restApp.restConnect);
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
org.apache.kafka.common.serialization.StringSerializer.class);
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
io.confluent.kafka.serializers.KafkaAvroSerializer.class);
props.put("producer.type", "sync");
props.put("serializer.class", "kafka.serializer.StringEncoder");
props.put("request.required.acks", "1");
props.put("ssl.keystore.location", "/var/ssl/kafka.client.keystore.jks"
)
props.put("ssl.keystore.password", "test1234")
props.put("ssl.key.password", "test1234")
ProducerConfig config = new ProducerConfig(props);
String userSchema =
"{\"namespace\": \"example.avro\", \"type\": \"record\", \"name\": \"U
ser\", \" +
                \"fields\":
[{\\"name\": \"name\", \"type\": \"string\"}]}";
Schema.Parser parser = new Schema.Parser();
Schema schema = parser.parse(userSchema);
GenericRecord avroRecord = new GenericData.Record(schema);
avroRecord.put("name", "testUser");
Producer<String, String> producer = new Producer<String,
String>(config);
ProducerRecord<String, Object> message = new
ProducerRecord<>("topicName", avroRecord );
producer.send(data);
```

**SparkProducer producer =  
HopsUtil.getSparkProducer();**



<https://github.com/hopshadoop/hops-kafka-examples>

# Kafka Producer in HopsWorks

```
SparkConf sparkConf = new SparkConf().setAppName(HopsUtil.getJobName());  
JavaSparkContext jsc = new JavaSparkContext(sparkConf);  
...  
SparkProducer producer = HopsUtil.getSparkProducer();  
...  
producer.produce(message);  
...  
HopsUtil.shutdownGracefully(jsc);
```

# Streaming Consumer in HopsWorks

```
SparkConf sparkConf = new SparkConf().setAppName(HopsUtil.getJobName());
JavaSparkContext jsc = new JavaSparkContext(sparkConf);
...
DataStreamReader dsr = HopsUtil.getSparkConsumer().getKafkaDataStreamReader();
Dataset<Row> lines = dsr.load();
...
StreamingQuery queryFile = logEntries.writeStream()
...
HopsUtil.shutdownGracefully(queryFile);
```



# Hops Roadmap

# Dela - A Global Ecosystem for Datasets



milli



Search results

3 Results found for: milli

**DATASET**  
millimetre

OWNER  
**Admin Admin**  
DATE CREATED  
2/23/17 7:44 AM

SIZE  
0.2KB



**PUBLIC DATASET**  
millionsong

OWNER  
**Admin Admin**  
DATE CREATED  
2/23/17 8:06 AM

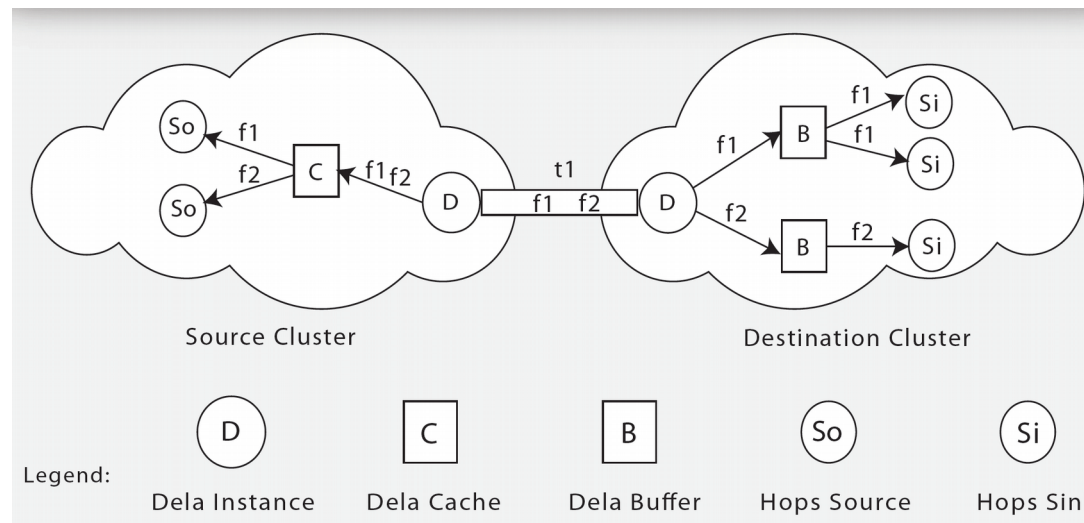
SIZE  
0.4KB



**PUBLIC DATASET**  
millionthings

OWNER  
**Admin Admin**  
DATE CREATED  
2/23/17 7:43 AM

SIZE  
0.4KB

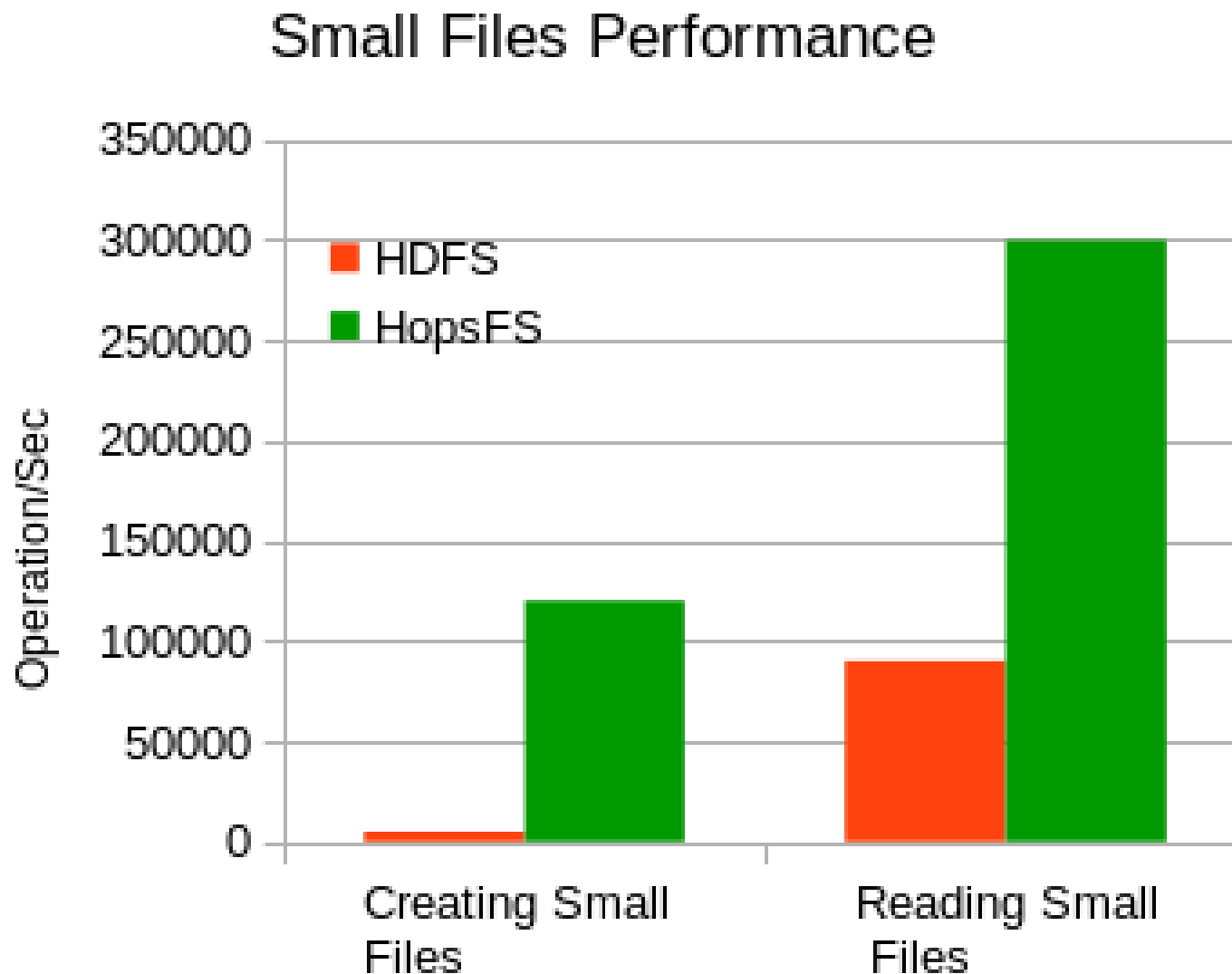


# Distributed Tensorflow on YARN



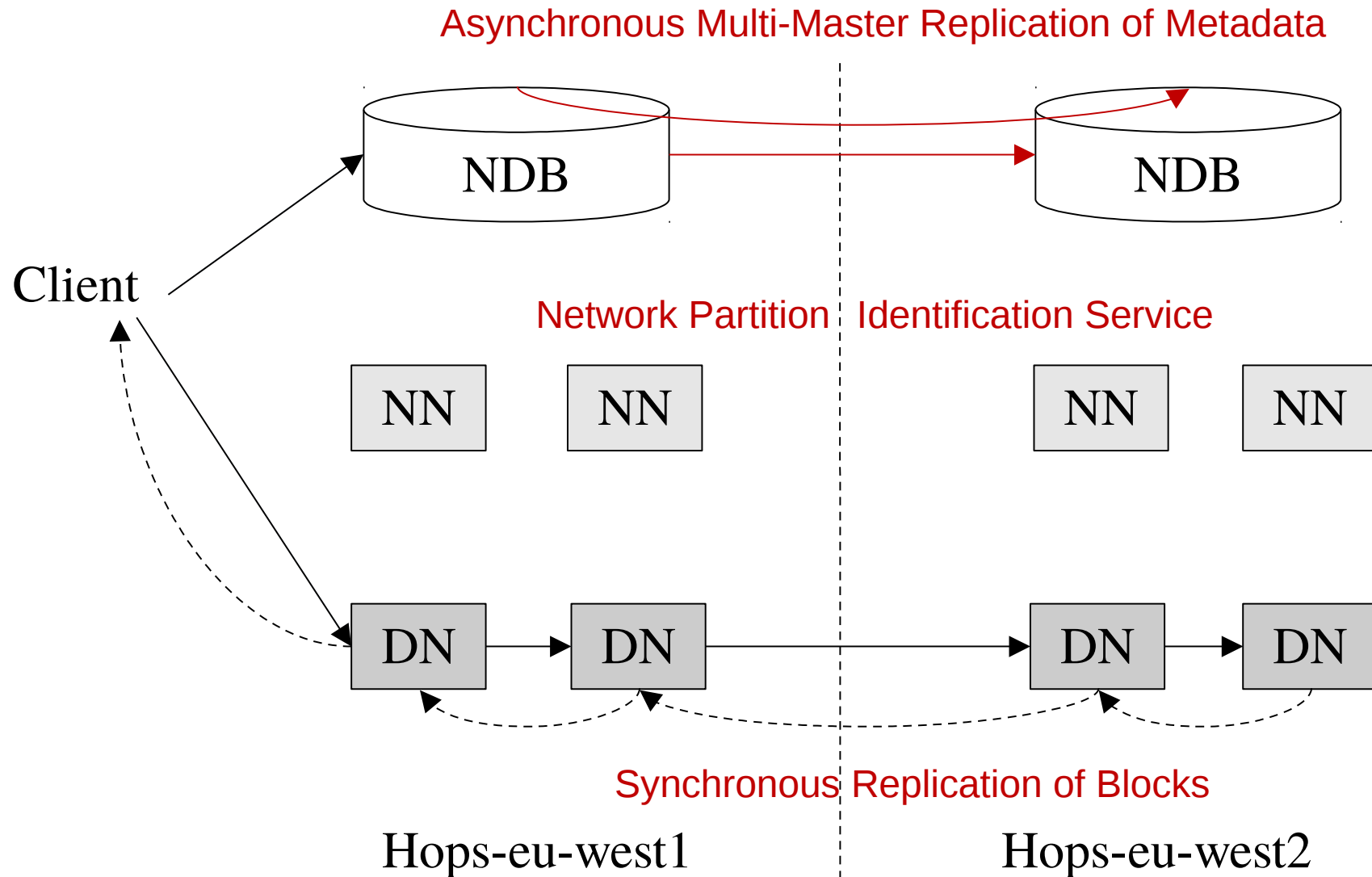
1. GPU-as-a-resource in Hops-YARN
2. Tensorflow-on-Spark
3. Native Tensorflow-on-YARN with Infiniband Support

# HopsFS Small Files Performance (Early Results)



30 namenodes/datanodes and 6 NDB nodes were used. Small file size was 4 KB. HopsFs files were stored on Intel 750 Series SSDs

# Multi-Data-Center HopsFS



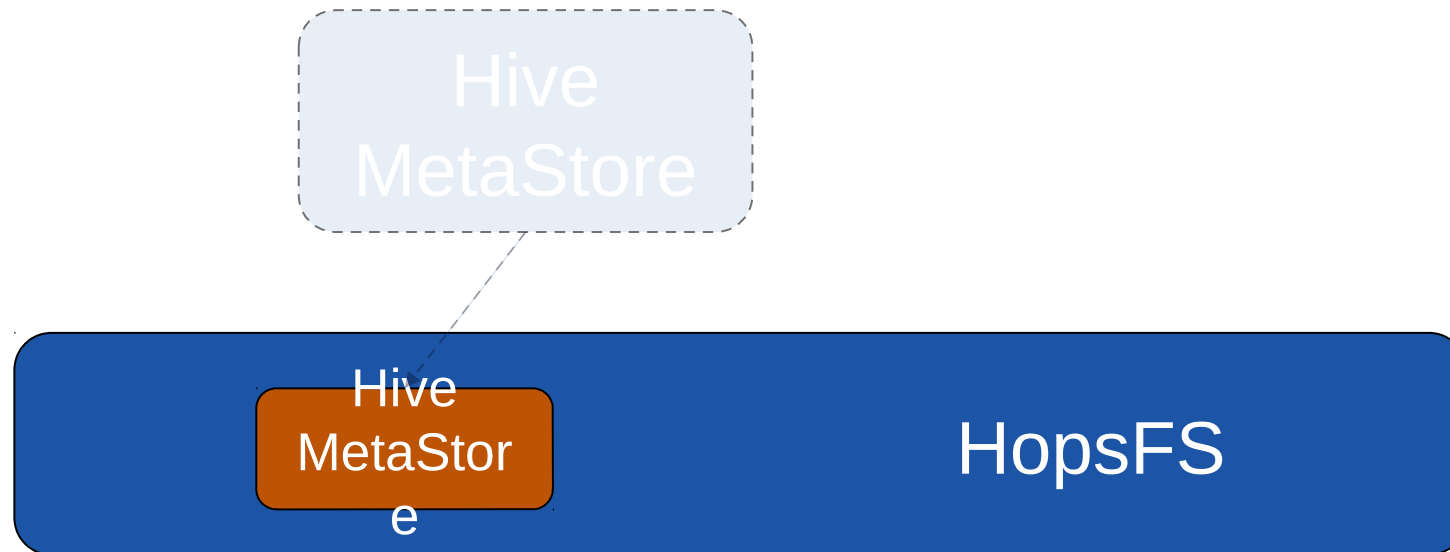
# Hive Metastore is Moving in with HopsFS

Hive  
MetaStore

A diagram illustrating the migration of Hive Metastore. It features two rounded rectangular boxes. The top box is orange and contains the text 'Hive MetaStore'. The bottom box is blue and contains the text 'HopsFS'. The blue box is significantly wider than the orange box, suggesting a larger or more integrated environment for the metastore.

HopsFS

# Hive Metastore is Moving in with HopsFS



# Strongly Consistent Hive Metadata

1.

```
0: jdbc:hive2://localhost:9084> show tables;
+-----+
| tab_name |
+-----+
| web_sales |
| web_site |
+-----+
2 rows selected (0.141 seconds)
0: jdbc:hive2://localhost:9084>
```



2.

```
$ hdfs dfs -rm -r /user/glassfish/2/web_sales
```



3.

```
0: jdbc:hive2://localhost:9084> show tables;
+-----+
| tab_name |
+-----+
| web_site |
+-----+
1 row selected (0.155 seconds)
0: jdbc:hive2://localhost:9084>
```

Removing the HDFS backing directory removes the Table from the Hive Metastore



# Summary

- Europe's Only Hadoop Distribution – Hops Hadoop
  - Fully Open-Source
- Hops supports larger/faster Hadoop Clusters
  - More scalable, tinker-friendly, and fully open-source.
- Hopsworks is a new Data Platform built on HopsFS with first-class support for Streaming
  - Spark or Flink

# Hops Heads

## Active:

Jim Dowling, Seif Haridi, Tor Björn Minde, Gautier Berthou, Salman Niazi, Mahmoud Ismail, Theofilos Kakantousis, Ermias Gebremeskel, Antonios Kouzoupis, Alex Ormenisan, Roberto Bampi, Fabio Buso, Fanti Machmount Al Samisti, Braulio Grana, Zahin Azher Rashid, Robin Andersson, ArunaKumari Yedurupaka, Tobias Johansson, August Bonds, Filotas Siskos.

## Alumni:

Vasileios Giannokostas, Johan Svedlund Nordström, Rizvi Hasan, Paul Mälzer, Bram Leenders, Juan Roca, Misganu Dessalegn, K "Sri" Srijeyanthan, Jude D'Souza, Alberto Lorente, Andre Moré, Ali Gholami, Davis Jaunzems, Stig Viaene, Hooman Peiro, Evangelos Savvidis, Steffen Grohsschmiedt, Qi Qi, Gayana Chandrasekara, Nikolaos Stanogias, Daniel Bali, Ioannis Kerkinos, Peter Buechler, Pushparaj Motamari, Hamid Afzali, Wasif Malik, Lalith Suresh, Mariano Valles, Ying Lieu.





Hops

# Thank You.

Follow us: @hopshadoop

Star us: <http://github.com/hopshadoop/hopsworks>

Join us: <http://www.hops.io>