Semantix

Meetup Squad Alertas - Camus

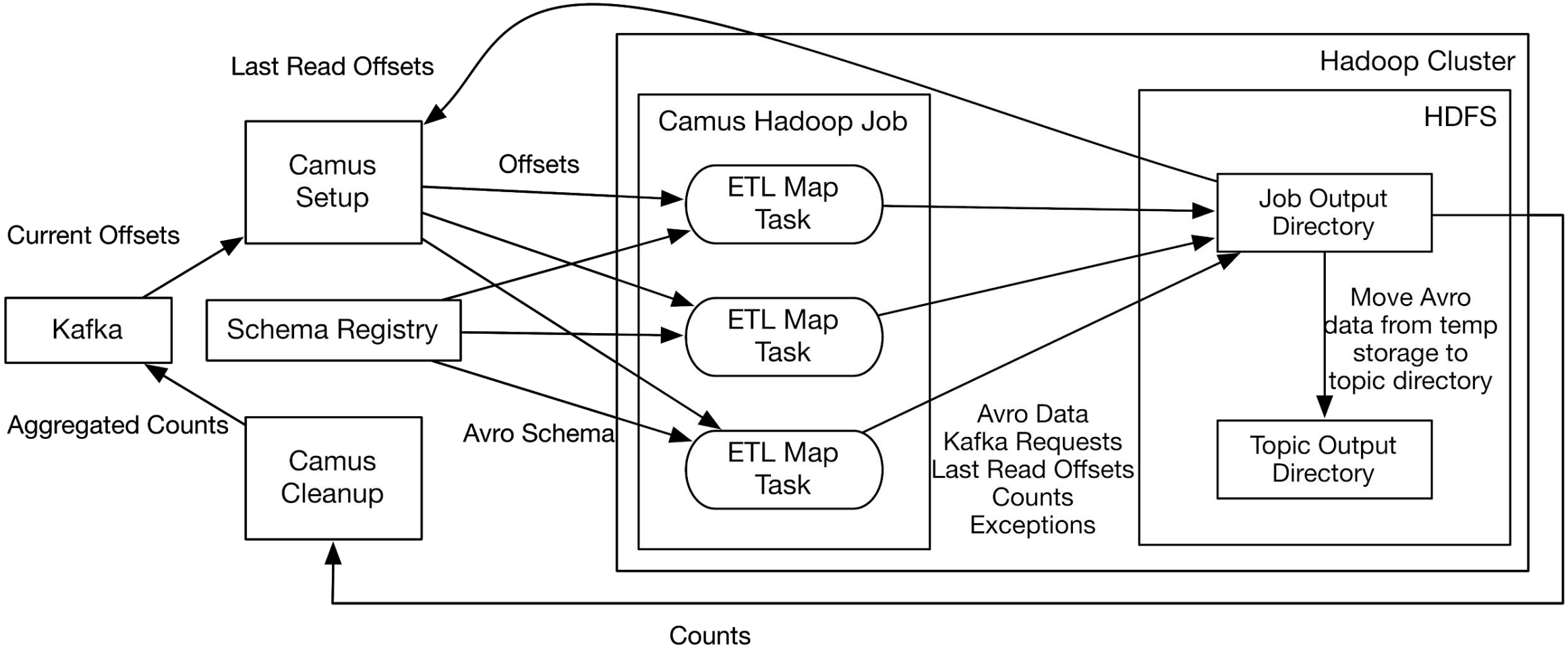
Agenda

- O que é o Camus
- Arquitetura
- Implementação
  - Avro
  - Schema Registry
  - Camus
  - Santander
- Hands-on
- Goblin

O que é o Camus

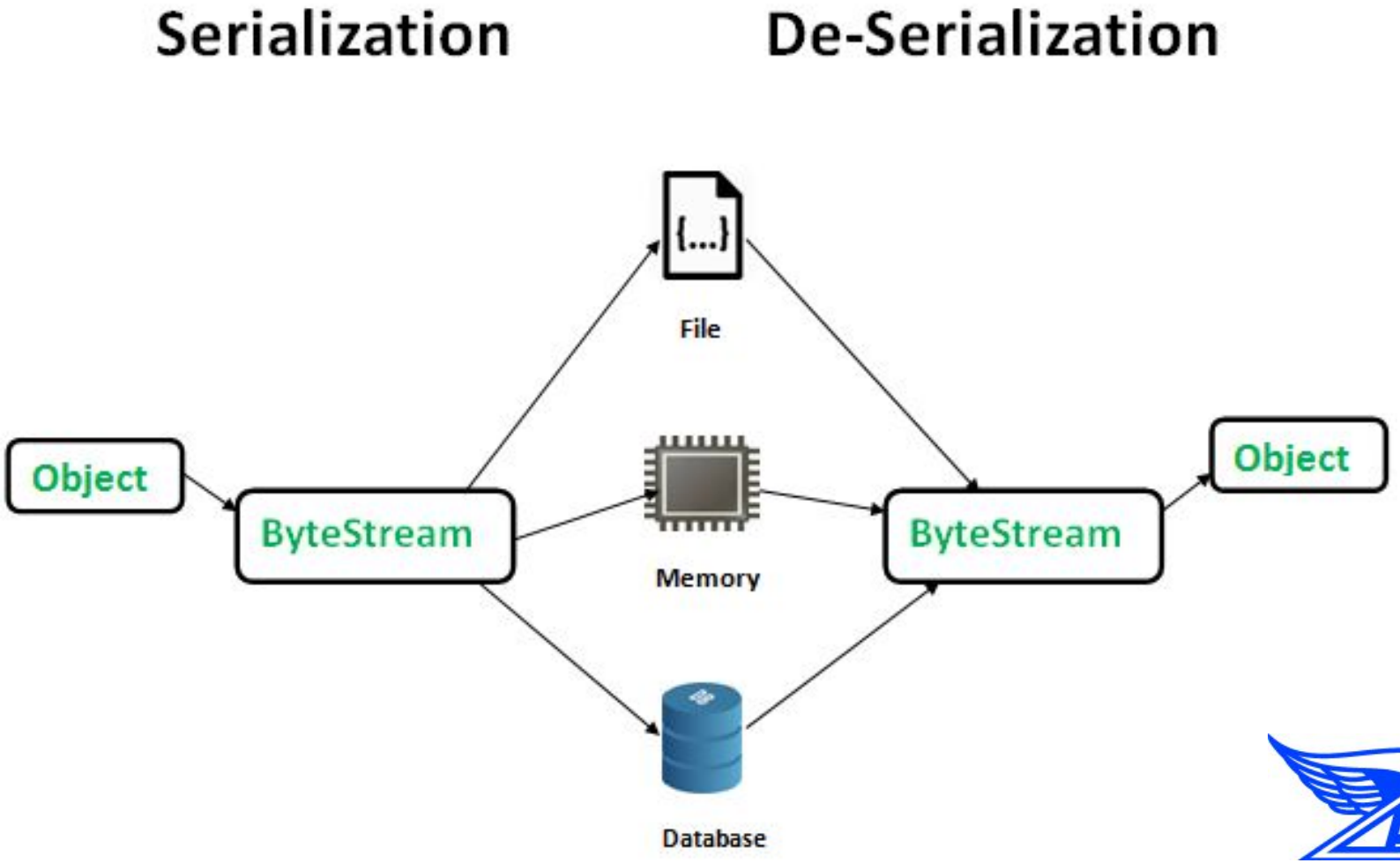
Camus is a simple MapReduce job developed by LinkedIn to load data from Kafka into HDFS. It is capable of incrementally copying data from Kafka into HDFS such that every run of the MapReduce job picks up where the previous run left off. At LinkedIn, Camus is used to load billions of messages per day from Kafka into HDFS.

Arquitetura



Implementação

Porque Avro?



Implementação

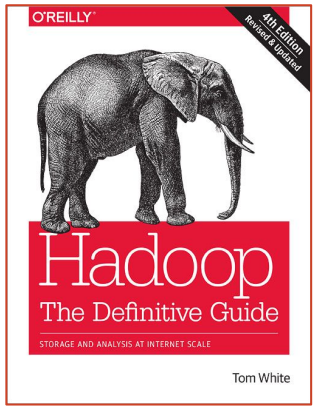
Porque Avro?

Serialization

Serialization is the process of turning structured objects into a byte stream for transmission over a network or for writing to persistent storage. *Deserialization* is the reverse process of turning a byte stream back into a series of structured objects.

Serialization is used in two quite distinct areas of distributed data processing: for interprocess communication and for persistent storage.

In Hadoop, interprocess communication between nodes in the system is implemented using *remote procedure calls* (RPCs). The RPC protocol uses serialization to render the message into a binary stream to be sent to the remote node, which then deserializes the binary stream into the original message. In general, it is desirable that an RPC seriali-



AVRO™

Implementação

Porque Avro?

Apache Avro[1] is a language-neutral data serialization system. The project was created by Doug Cutting (the creator of Hadoop) to address the major downside of Hadoop Writables: lack of language portability. Having a data format that can be processed by many languages (currently C, C++, C#, Java, JavaScript, Perl, PHP, Python, and Ruby) makes it easier to share datasets with a wider audience than one tied to a single language. It is also more future-proof, allowing data to potentially outlive the language used to read and write it.

Implementação

Defining a schema

Avro schemas are defined using JSON. Schemas are composed of primitive types (`null`, `boolean`, `int`, `long`, `float`, `double`, `bytes`, and `string`) and complex types (`record`, `enum`, `array`, `map`, `union`, and `fixed`). You can learn more about Avro schemas and types from the specification, but for now let's start with a simple schema example, *user.avsc*:

```
{"namespace": "example.avro",
 "type": "record",
 "name": "User",
 "fields": [
     {"name": "name", "type": "string"},
     {"name": "favorite_number",  "type": ["int", "null"]},
     {"name": "favorite_color", "type": ["string", "null"]}
 ]
}
```

Implementação

```
{
  "namespace":
  "type": "record",
  "name": "sdgdownloadhue",
  "doc": "Avro schema to download hue",
  "fields": [
    { "name": "appId",       "type": "string" },
    { "name": "category",    "type": {"type": "enum", "name": "options_method", "symbols": ["METRIC", "SECURITY", "TELEMETRY", "APPLICATION"]} },
    { "name": "watermark",   "type": ["string","null"] },
    { "name": "ipCollector", "type": ["string","null"] },
    { "name": "timestamp",   "type": ["string","null"] },
    { "name": "cluster",     "type": ["string","null"], "default": "Cluster DCN" },
    { "name": "usuario",     "type": "string" },
    { "name": "datahora",    "type": "string" },
    { "name": "clienteip",   "type": "string" },
    { "name": "ferramenta",  "type": ["string","null"], "default": "hue" },
    { "name": "extraInfo","type": {"type": "map", "values": "string"} }
  ]
}
```

Implementação

Schema Registry

The Schema Registry is the answer to this problem: it is a server that runs in your infrastructure (close to your Kafka brokers) and that stores your schemas (including all their versions). When you send Avro messages to Kafka, the messages contain an identifier of a schema stored in the Schema Registry.

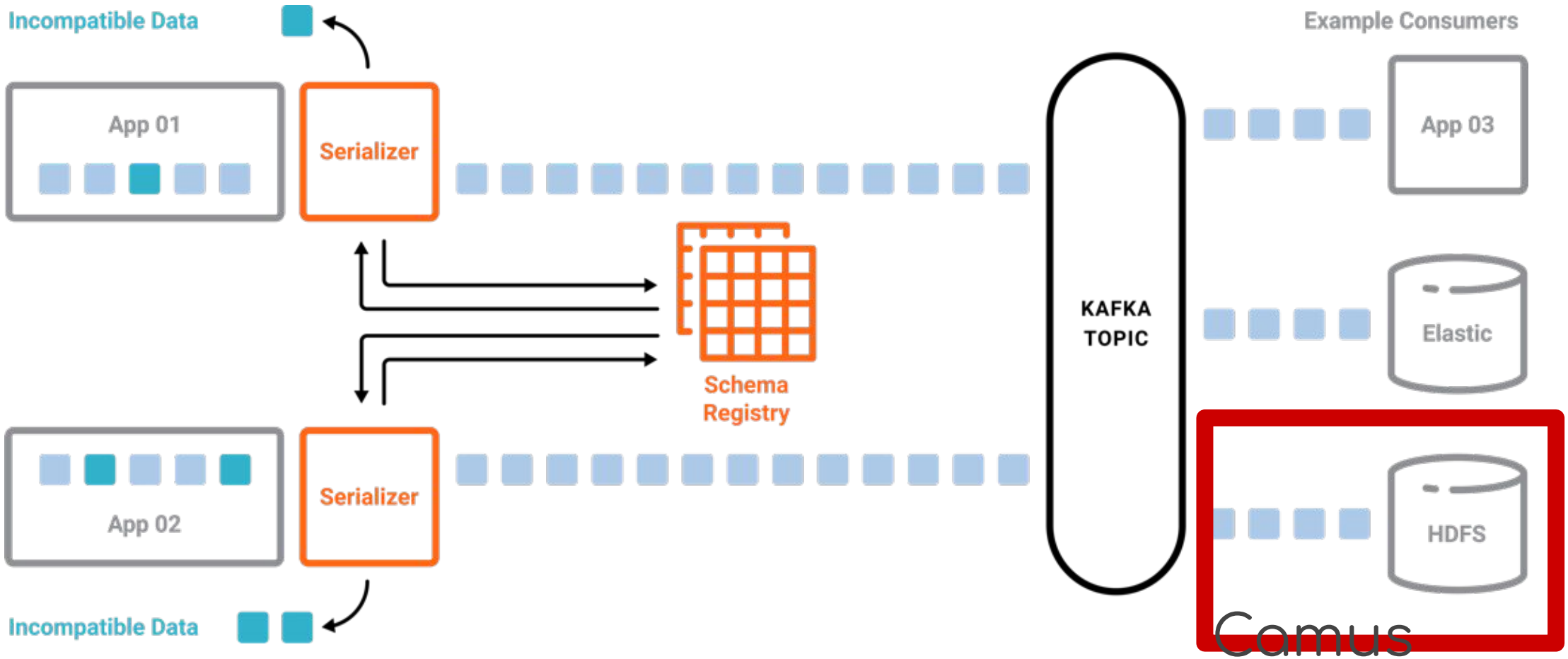
A library allows you to serialize and deserialize Avro messages, and to interact transparently with the Schema Registry:

- When sending a message, the serializer will make sure the schema is registered, get its ID, or register a new version of the schema for you (this can be disabled by setting `auto.register.schemas` to `false`).
- When reading a message, the deserializer will find the ID of the schema in the message, and fetch the schema from the Schema Registry to deserialize the Avro data.

Both the Schema Registry and the library are under the Confluent umbrella: open source but not part of the Apache project. This means you will want to use the Confluent distribution to use the Schema Registry, not the Apache distribution.

Implementação

Schema Registry



Implementação

register_schema.py

Raw

```python
import os
import sys

import requests

schema_registry_url = sys.argv[1]
topic = sys.argv[2]
schema_file = sys.argv[3]

aboslute_path_to_schema = os.path.join(os.getcwd(), schema_file)

print("Schema Registry URL: " + schema_registry_url)
print("Topic: " + topic)
print("Schema file: " + schema_file)
print

with open(aboslute_path_to_schema, 'r') as content_file:
    schema = content_file.read()

payload = "{ \"schema\": \"" \
          + schema.replace("\"", "\\\"").replace("\t", "").replace("\n", "") \
          + "\" }"

url = schema_registry_url + "/subjects/" + topic + "-value/versions"
headers = {"Content-Type": "application/vnd.schemaregistry.v1+json"}

r = requests.post(url, headers=headers, data=payload)
if r.status_code == requests.codes.ok:
    print("Success")
else:
    r.raise_for_status()
```

Implementação

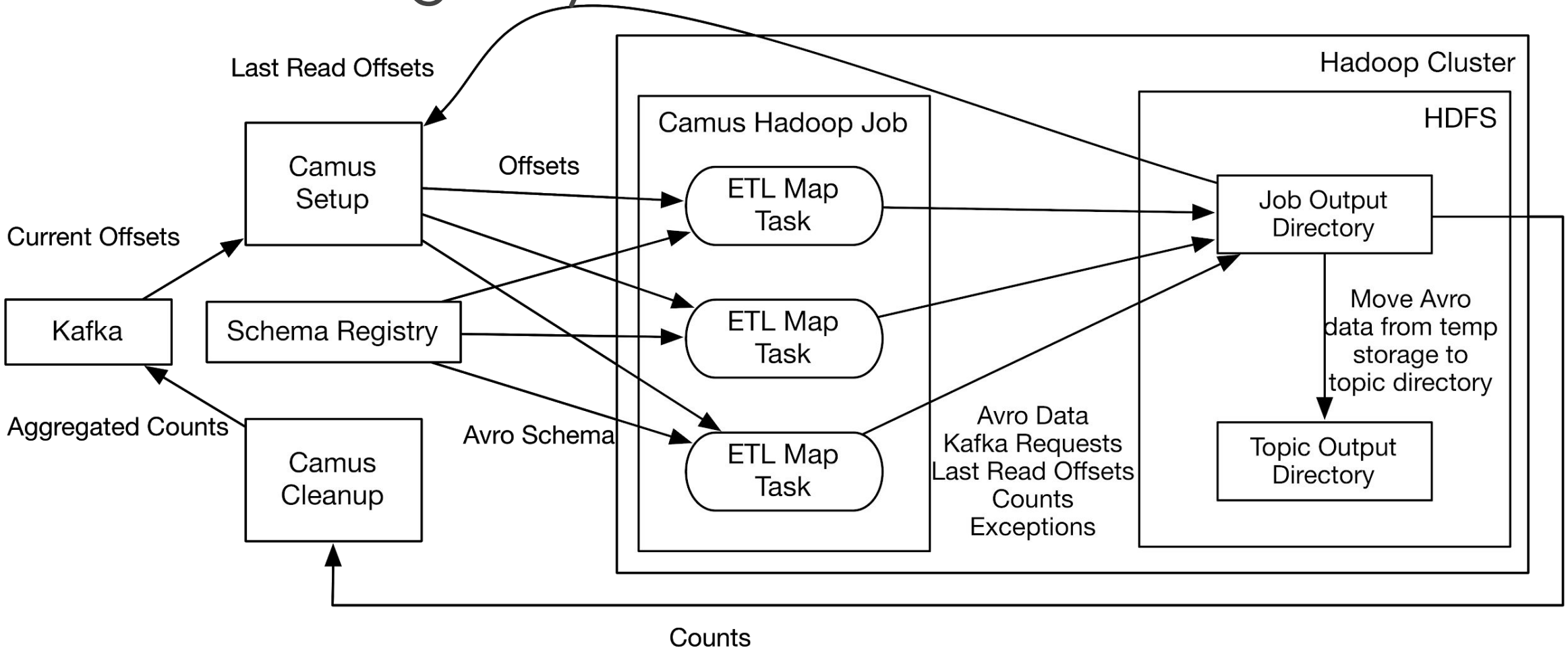
```
$ python src/main/resources/register_schema.py http://localhost:8081 persons-avro src/main/resources/person.avsc
Schema Registry URL: http://localhost:8081
Topic: persons-avro
Schema file: src/main/resources/person.avsc

Success
```

```
semantix@semantix:~/meetups/camus$ curl http://localhost:8081/subjects/downloadhue-value/versions/1 && echo
{"subject":"downloadhue-value","version":1,"id":1,"schema":"{\"type\":\"record\",\"name\":\"sdgdownloadhue\",\"namespace\":\"br.com.produban.openbus.model.avro\",\"doc\":\"Avro schema to download hue\",\"fields\":[{\"name\":\"appId\",\"type\":\"string\"},{\"name\":\"category\",\"type\":{\"type\":\"enum\",\"name\":\"options_method\",\"symbols\":[\"METRIC\",\"SECURITY\",\"TELEMETRY\",\"APPLICATION\"]}},{\"name\":\"watermark\",\"type\":[\"string\",\"null\"]},{\"name\":\"ipCollector\",\"type\":[\"string\",\"null\"]},{\"name\":\"timestamp\",\"type\":[\"string\",\"null\"]},{\"name\":\"cluster\",\"type\":[\"string\",\"null\"],\"default\":\"Cluster DCN\"},{\"name\":\"usuario\",\"type\":\"string\"},{\"name\":\"datahora\",\"type\":\"string\"},{\"name\":\"clienteip\",\"type\":\"string\"},{\"name\":\"ferramenta\",\"type\":[\"string\",\"null\"],\"default\":\"hue\"},{\"name\":\"extraInfo\",\"type\":{\"type\":\"map\",\"values\":\"string\"}}]}"}
semantix@semantix:~/meetups/camus$
```

Implementação

Schema Registry



Hands-on


```
wget http://packages.confluent.io/archive/2.0/confluent-2.0.0-2.11.7.zip
unzip confluent-2.0.0-2.11.7.zip
cd confluent-2.0.0
```

Hands-on

- Start Services

```
./bin/zookeeper-server-start ./etc/kafka/zookeeper.properties
./bin/kafka-server-start ./etc/kafka/server.properties
./bin/schema-registry-start ./etc/schema-registry/schema-registry.properties
```

Hands-on

- Criar Tópicos

```
./bin/kafka-topics --create --zookeeper localhost:2181 --replication-factor 1
--partitions 1 --topic test

./bin/kafka-topics --create --zookeeper localhost:2181 --replication-factor 1
--partitions 1 --topic alertas

./bin/kafka-topics --list --zookeeper localhost:2181
```

Hands-on

- Hadoop

https://hadoop.apache.org/docs/r3.1.1/hadoop-project-dist/hadoop-common/SingleCluster.html#Download

Hands-on

- Registrar Schema

```
python3 register_schema.py http://localhost:8081 person_alertas
person.avsc
```

```
{
  "namespace": "br.com.xyz",
  "type": "record",
  "name": "person",
  "doc": "person schema",
  "fields": [
    { "name": "name",       "type": "string" },
    { "name": "age",  "type": "string" }
  ]
}
```

Hands-on

- Teste schema

```
curl http://localhost:8081/subjects/person/versions/1 && echo
```

Hands-on

- Geração de Mensagens Avro - Producer

```
./bin/kafka-avro-console-producer \
        --broker-list localhost:9092 --topic test \
        --property
value.schema='{"type":"record","name":"person","fields":[{"name":"name","typ
e":"string"}, {"name":"age","type":"integer"}]}'

> {"name": "Quenaz", “age”: 32}
> {"name": "Marcos", “age”: 16}
> {"name": "Renata", “age”: 22}
> {"name": "Guilherme", “age”: 48}
```

Hands-on

- Consumo das Mensagens

```
./bin/kafka-avro-console-consumer --topic test --zookeeper localhost:2181 \
        --from-beginning
```

Hands-on

- Camus

```
bin/camus-run -P etc/camus/camus.properties
```

etc/camus/camus.properties

Goblin


Google Analytics
Microsoft Dynamics CRM
Couchbase
AVRO
TERADATA
Microsoft SQL S


Obrigado