Building a Dataset Search Engine with Spark and Elasticsearch

Oscar Castañeda-Villagrán

Xoom a PayPal service

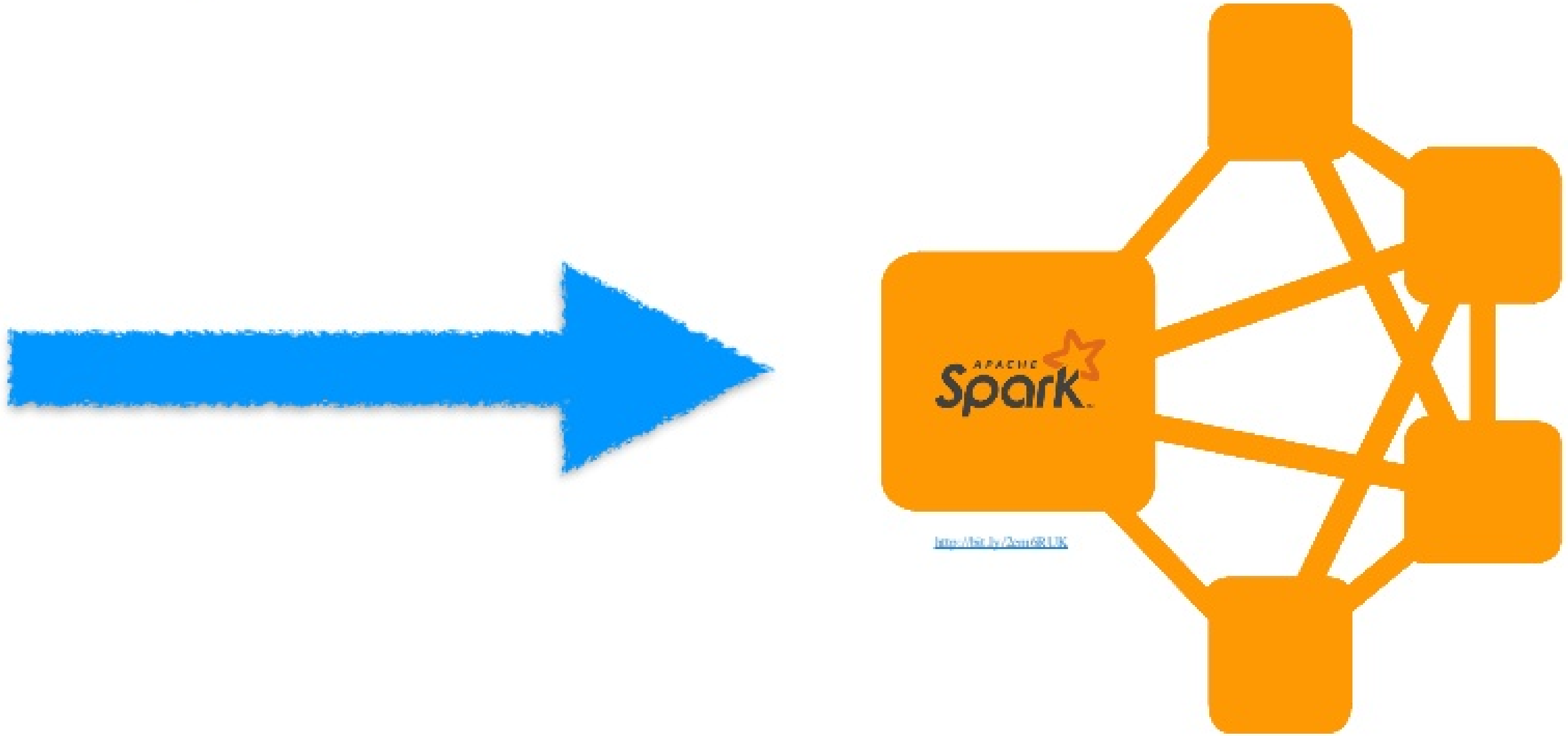
SPARK SUMMIT EAST 2017

About

- Data Scientist at Xoom a PayPal service.

- Interests:
  - Data Management,
  - Dataset Search,
  - Online Learning to Rank.

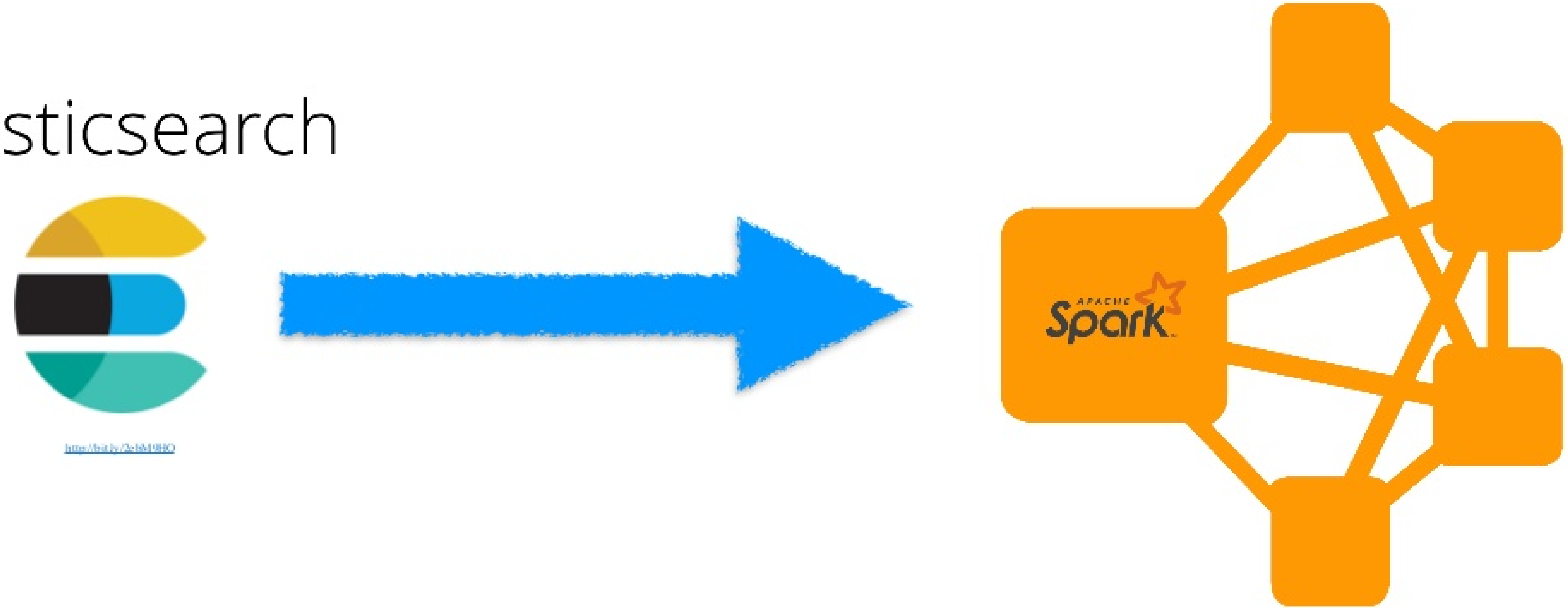
Spark cluster …



http://bit.ly/2cm6RUK

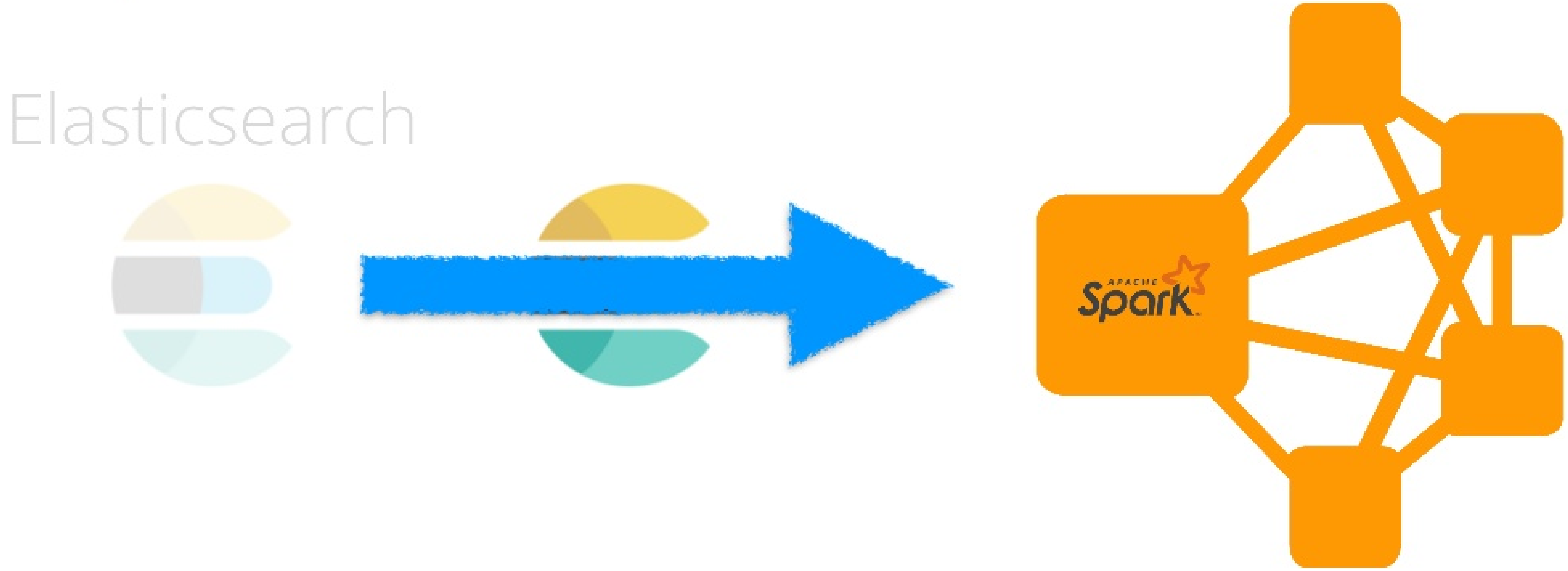
Spark cluster with …

Elasticsearch

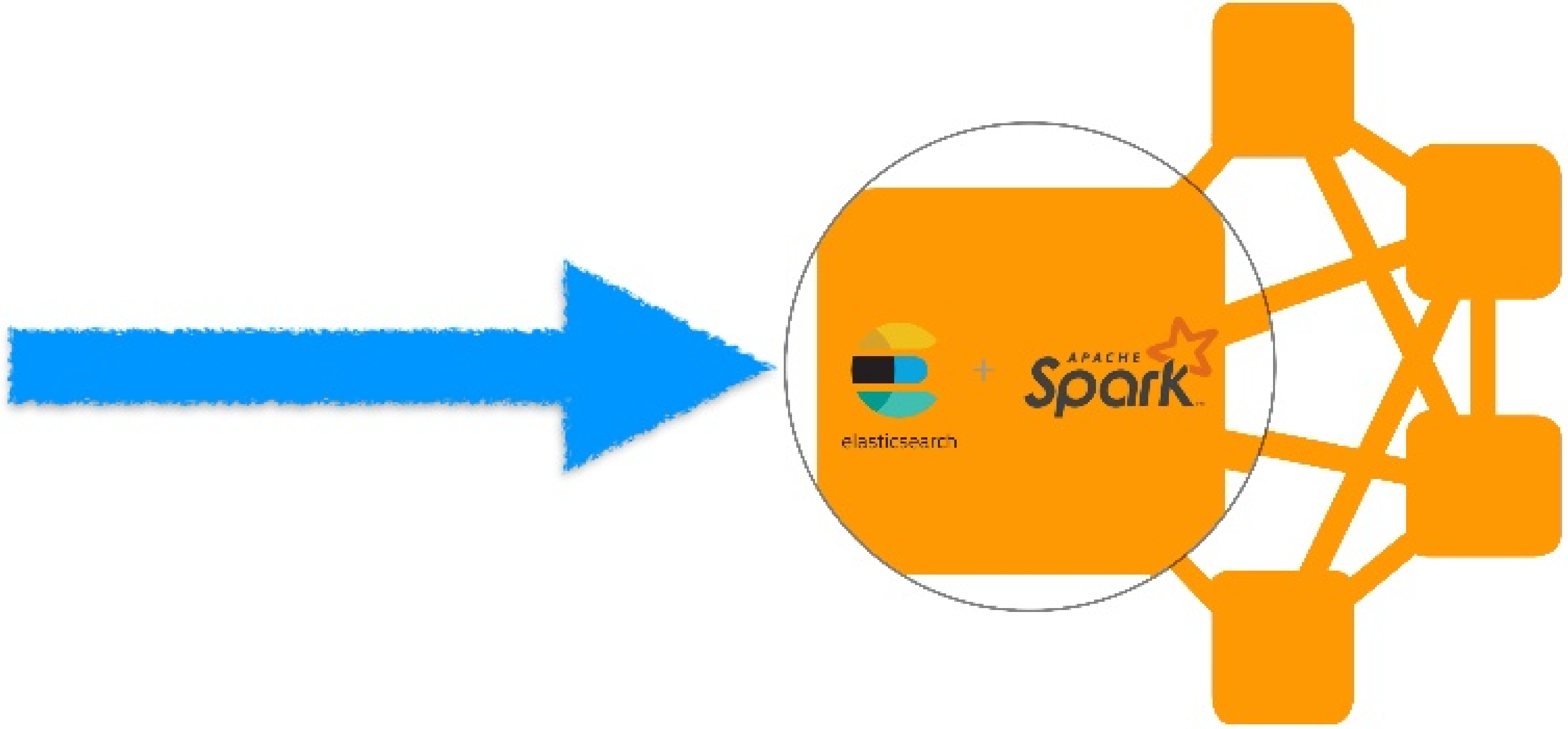


http://bit.ly/2cbM9HO

Spark cluster with Elasticsearch …

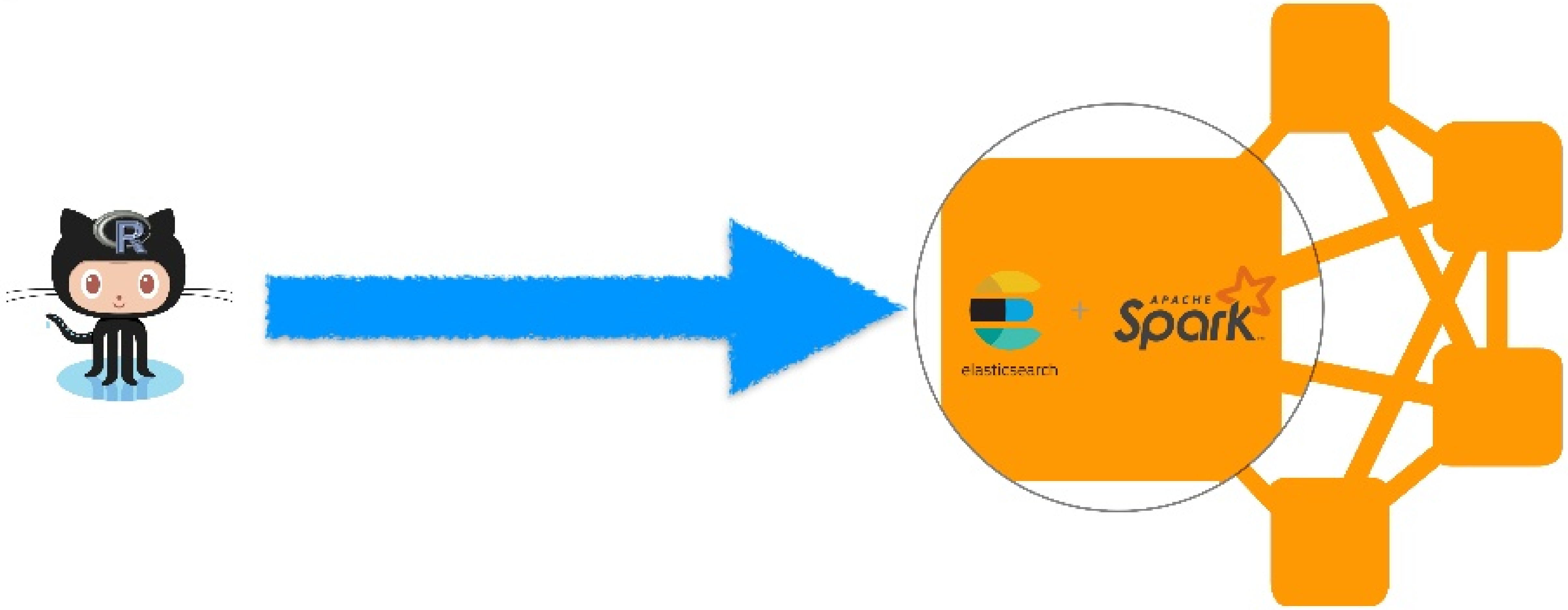


Spark cluster with Elasticsearch Inside



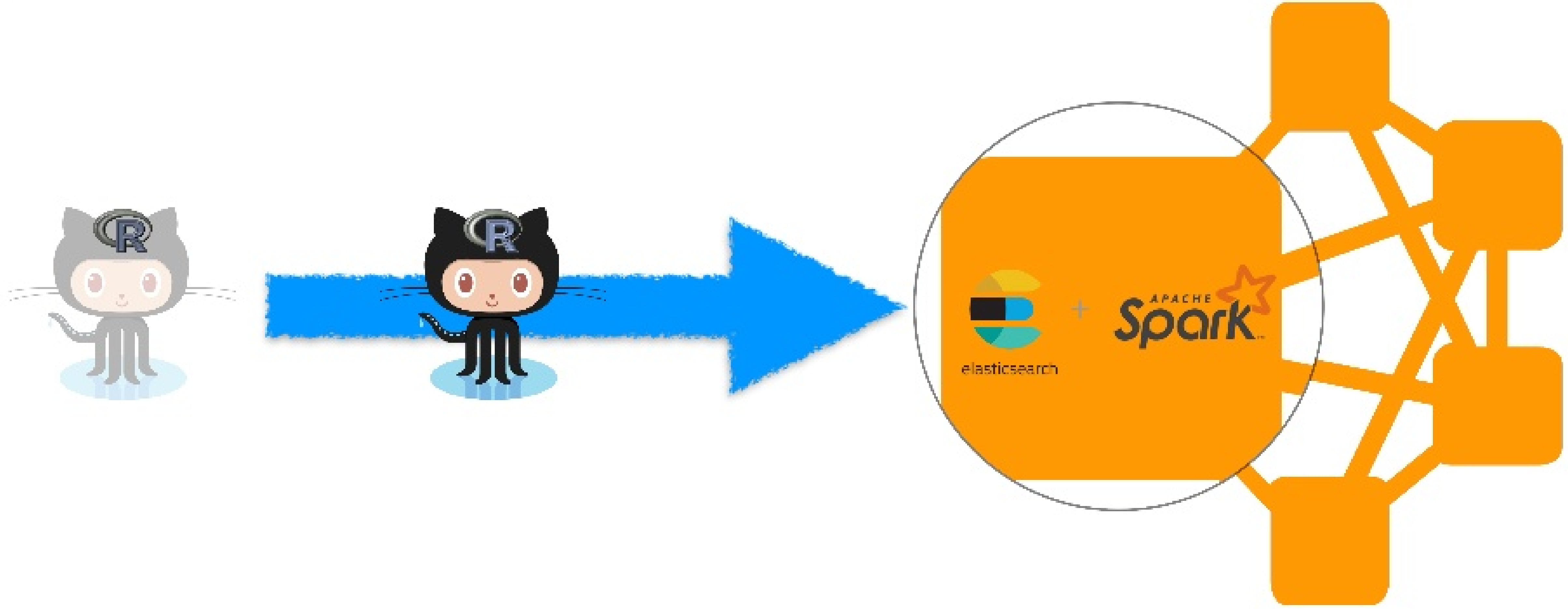
And …

Spark cluster with Elasticsearch Inside



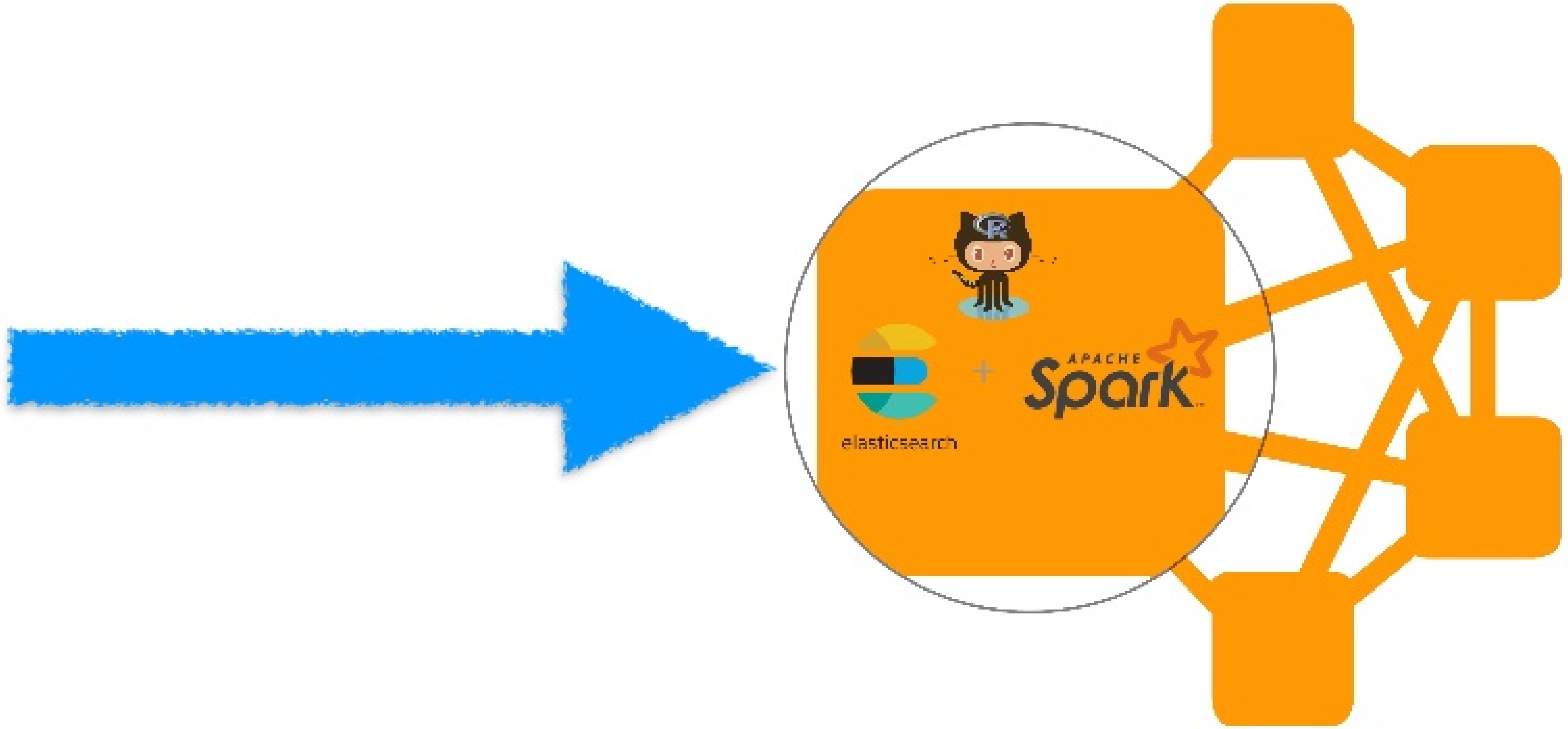
And …

Spark cluster with Elasticsearch Inside



And Indexed …

Spark cluster with Elasticsearch Inside



And Indexed RDatasets

Agenda

- Problem Statement, Overview and Motivation.
- Elasticsearch Server inside Spark Cluster.
- **Metadata extraction**. (Write to ES + Snapshot index.)
- Demo: RDataset Search Engine with Spark and Elasticsearch.
- Q&A

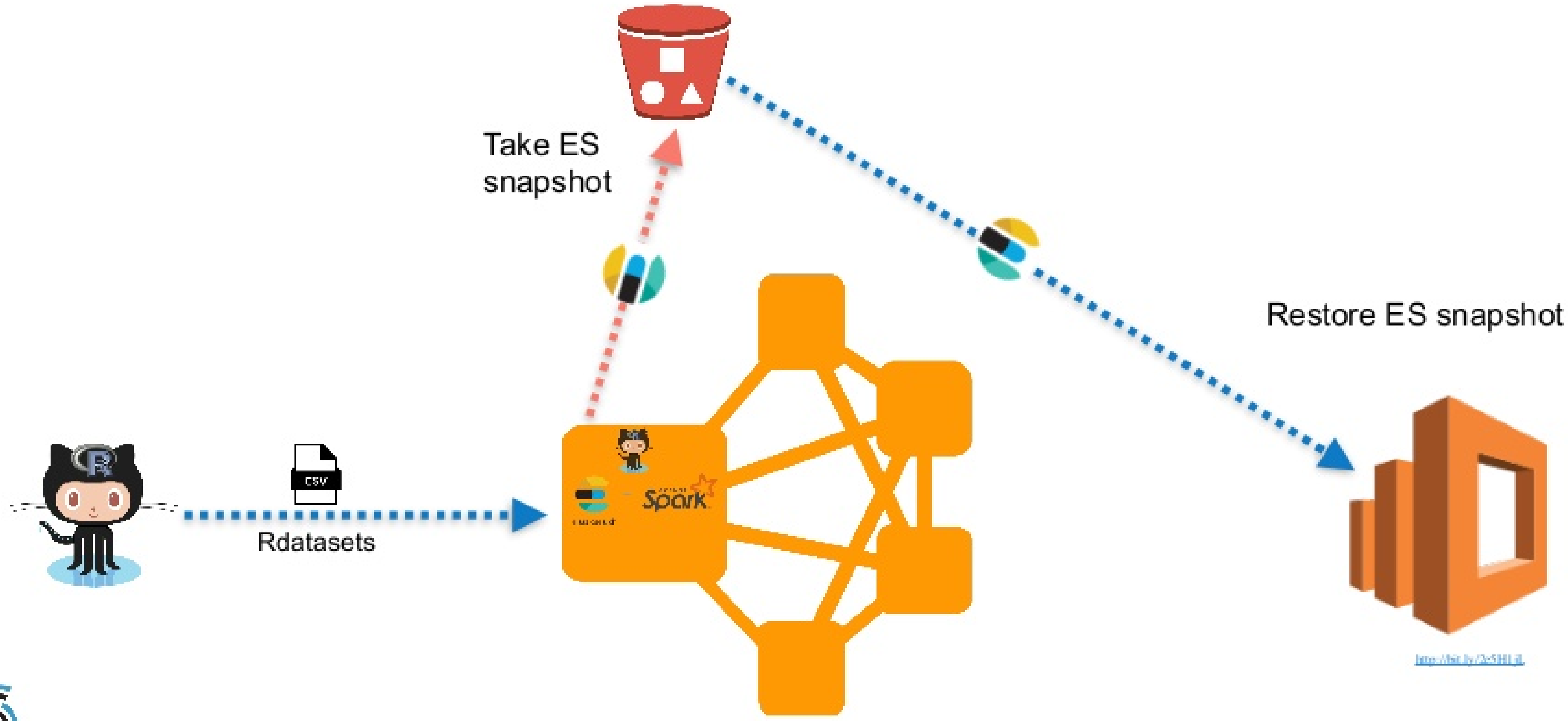
Problem Statement

- Despite Datasets being a <u>key corporate asset</u> they are generally not given the importance they deserve in terms of making them easy to find.

Questions

- How are Datasets produced?
- Can indexing be performed when Dataset are produced?

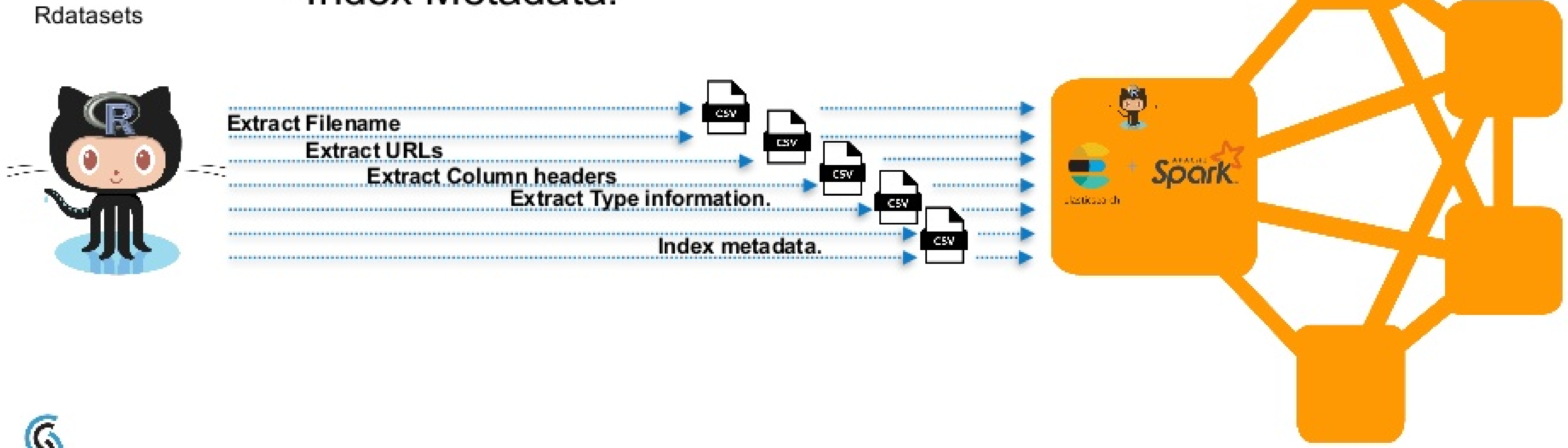
Overview


Take ES
snapshot
Restore ES snapshot
CSV
Rdatasets
Spark
http://bit.ly/2c5HLjL

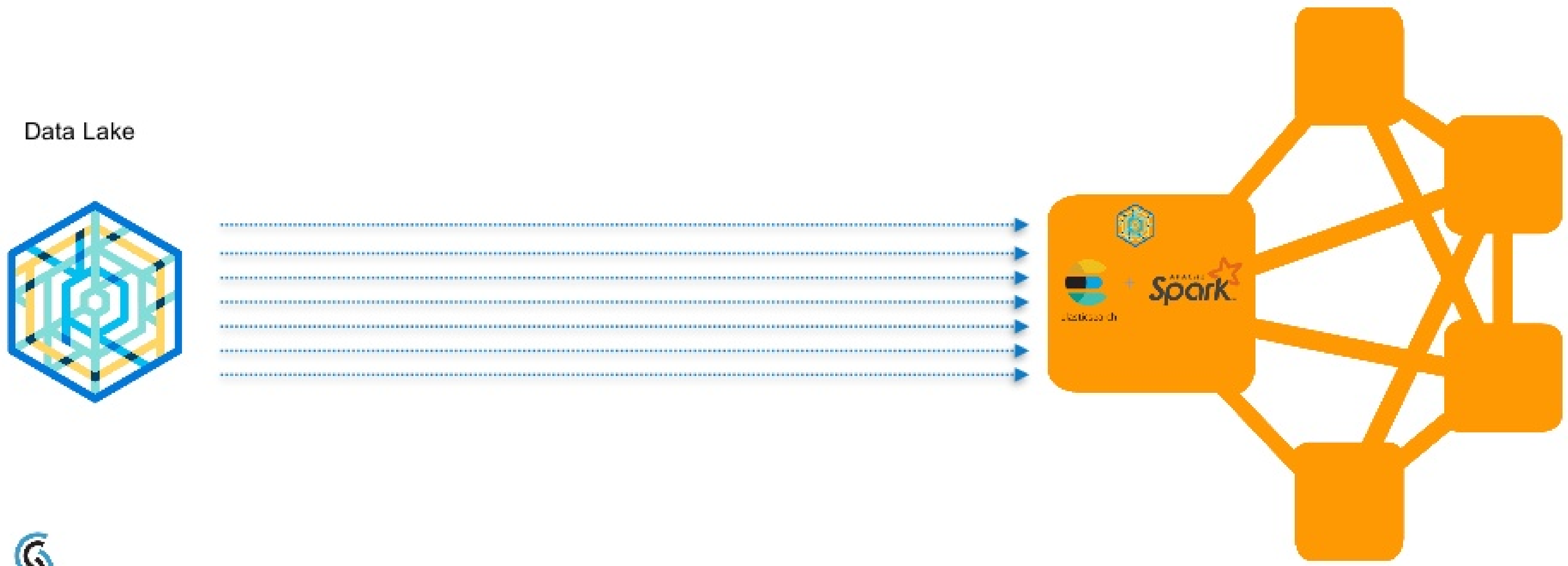
Overview: Metadata Extraction

Data Pipelines:

- Extract: Filename, URLs, Column headers.
- Extract: Type information.
- Index Metadata.

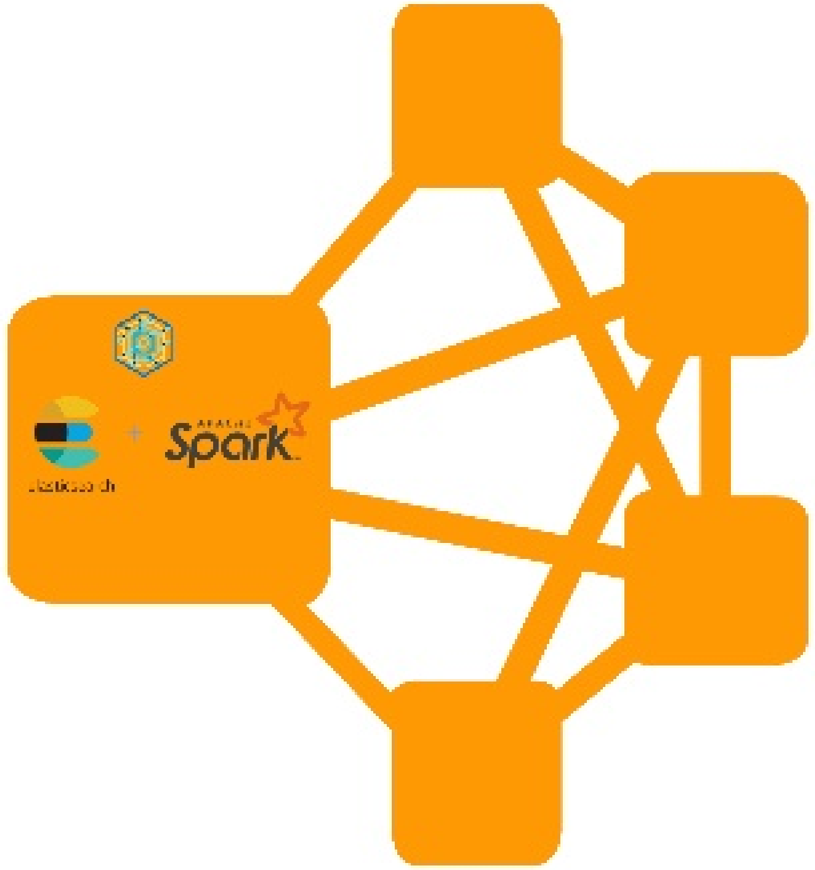


Overview



Overview

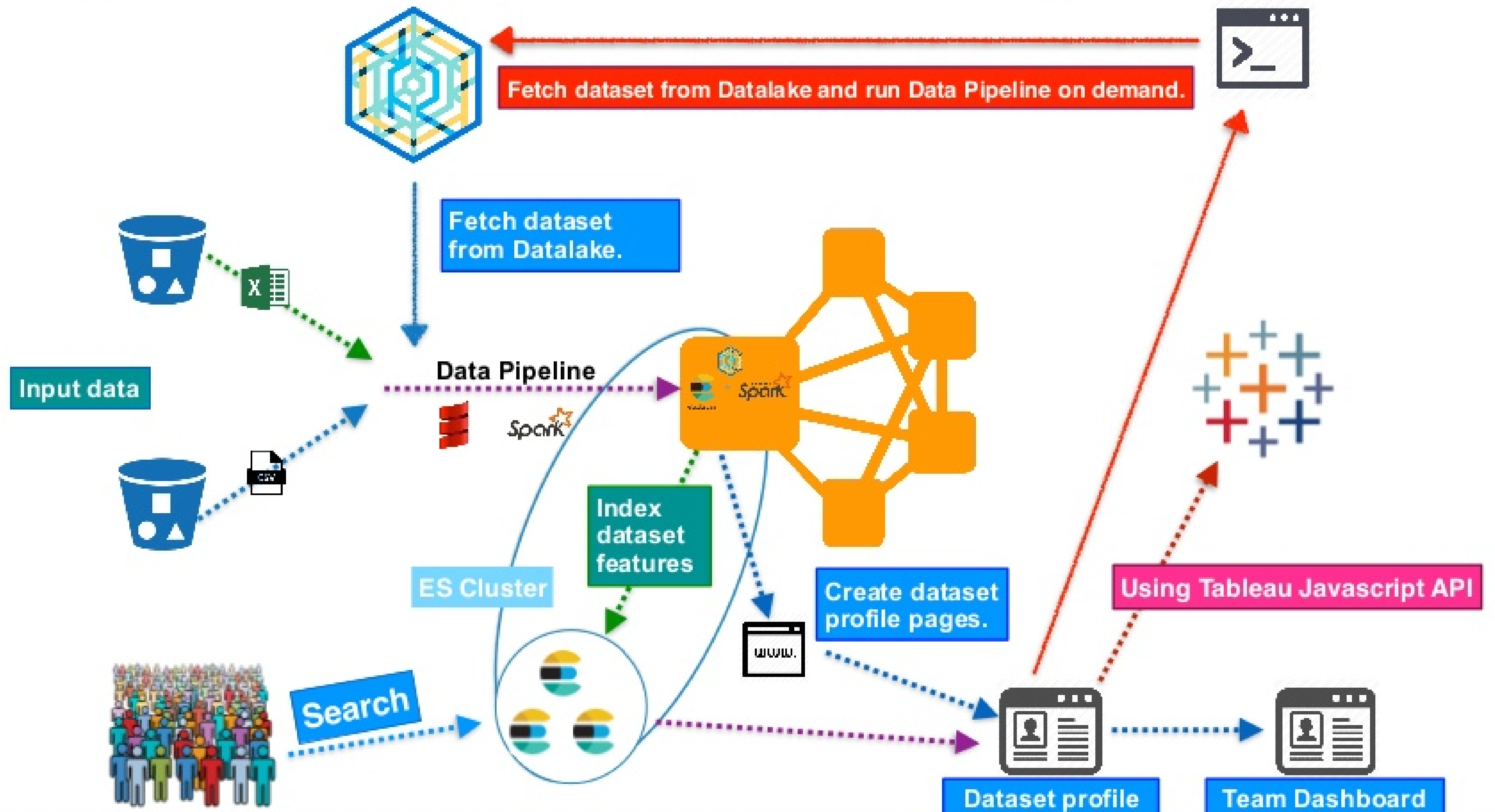
Data Lake



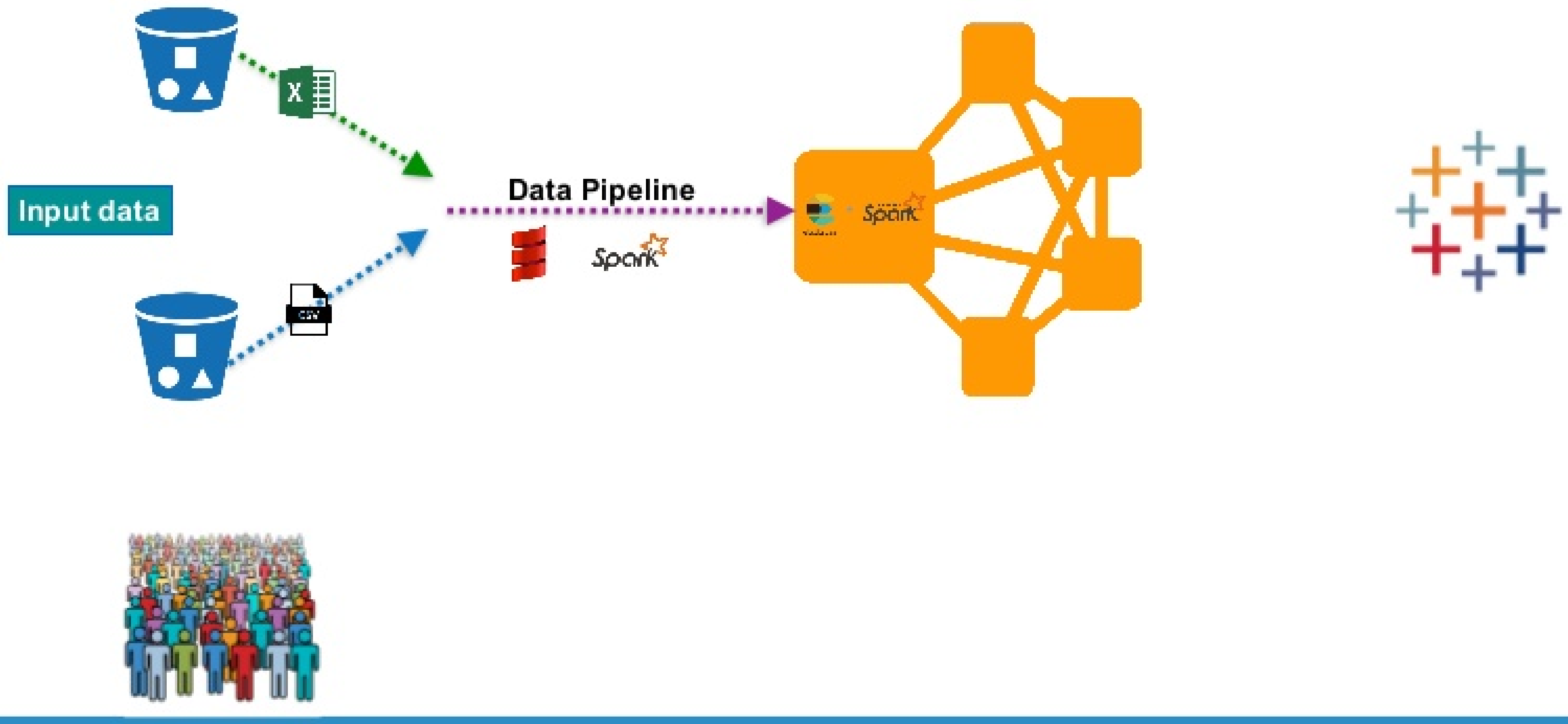
Motivation

- *Organizing and indexing* Datasets and Data Lake(s).
- Spark with Elasticsearch Inside can be used to *index datasets* that are produced as part of Spark data pipelines.
- "*Replaying Datasets*" in Data Lake(s) can be leveraged to create a dataset index (*a posteriori* vs. post hoc (Halevy et al., 2016)).
- ES snapshot is a useful means to deploy Dataset search index in production (out of Spark with Elasticsearch Inside).

Organizing and Indexing Datasets

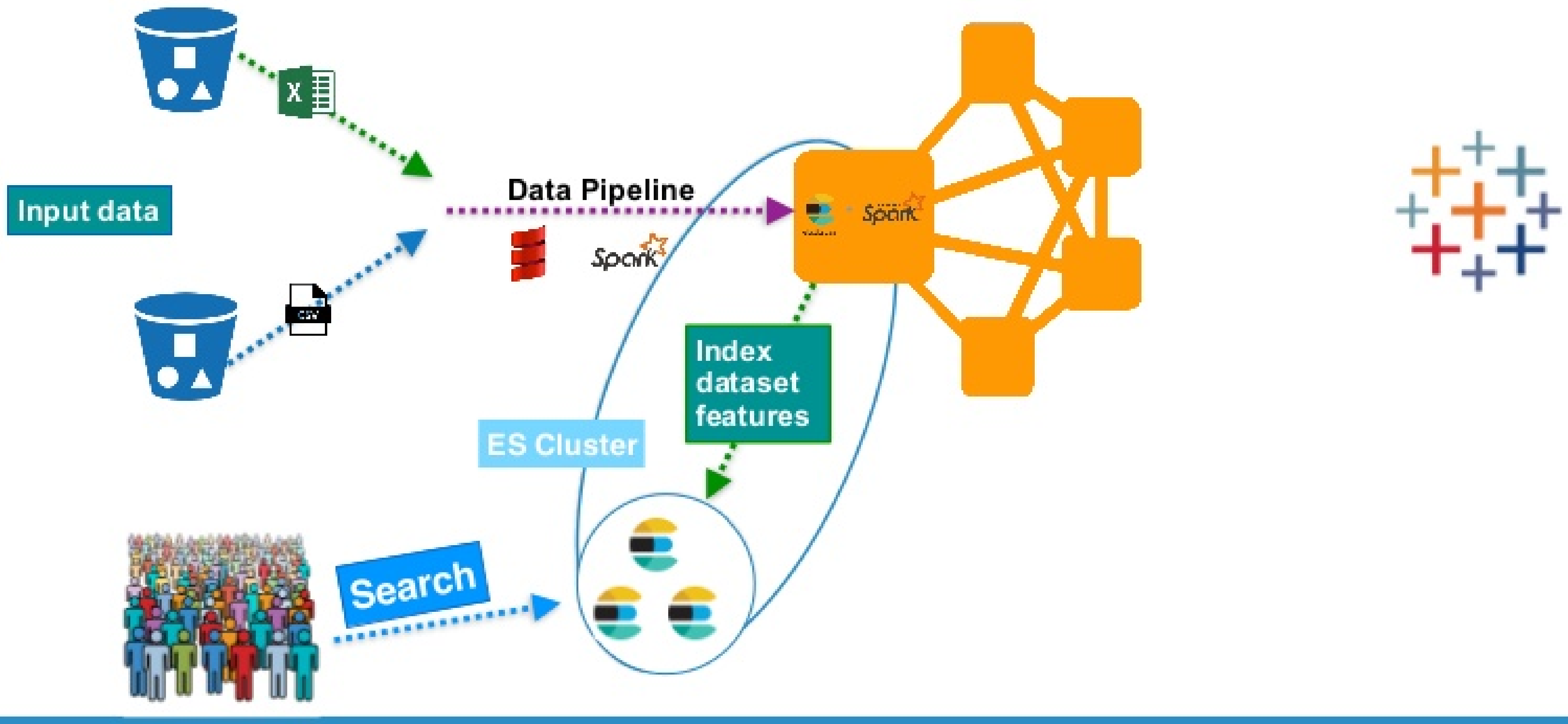

Fetch dataset from Datalake and run Data Pipeline on demand.
Fetch dataset from Datalake.
Input data
Data Pipeline
Spark
Index dataset features
ES Cluster
Search
www.
Create dataset profile pages.
Using Tableau Javascript API
Dataset profile
Team Dashboard

Organizing and Indexing Datasets

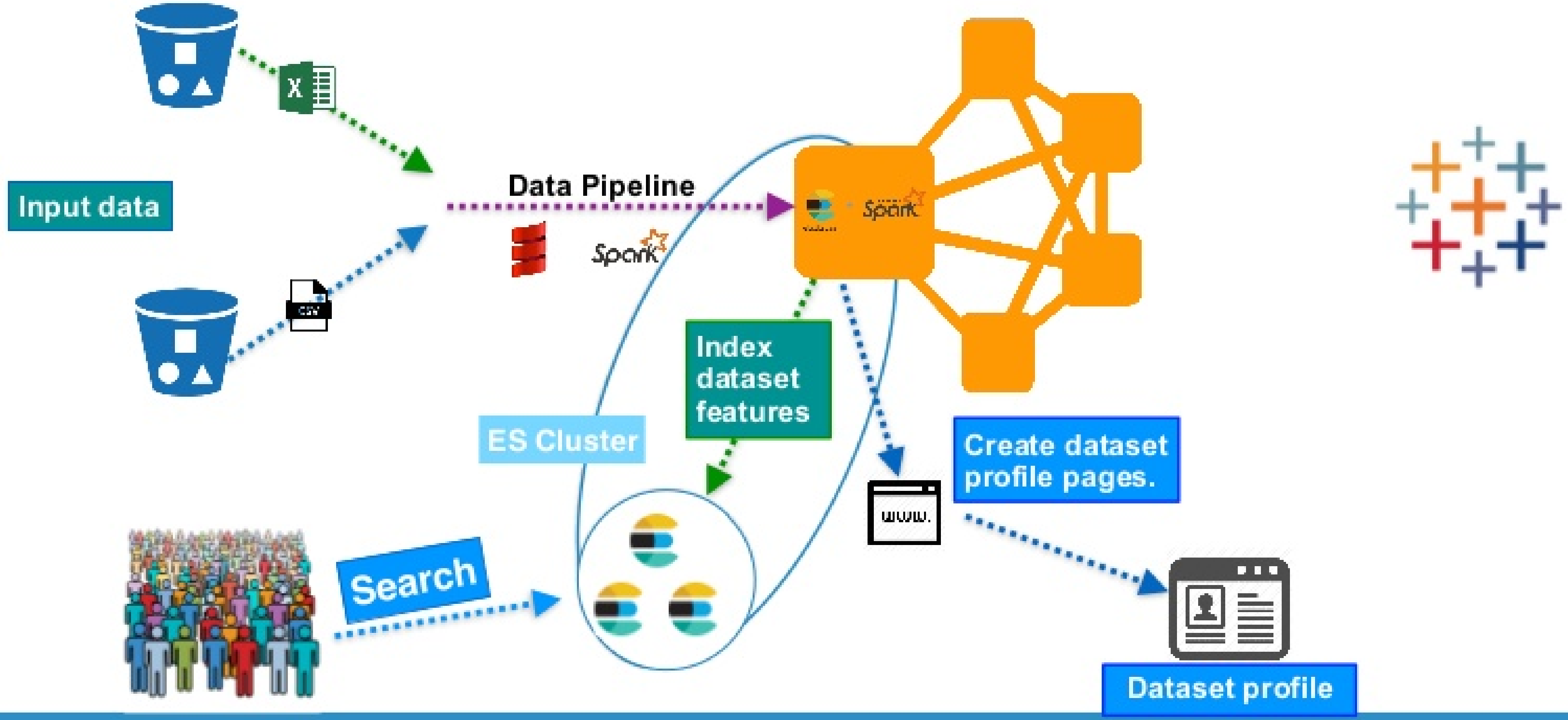


SPARK SUMMIT EAST 2017

Organizing and Indexing Datasets

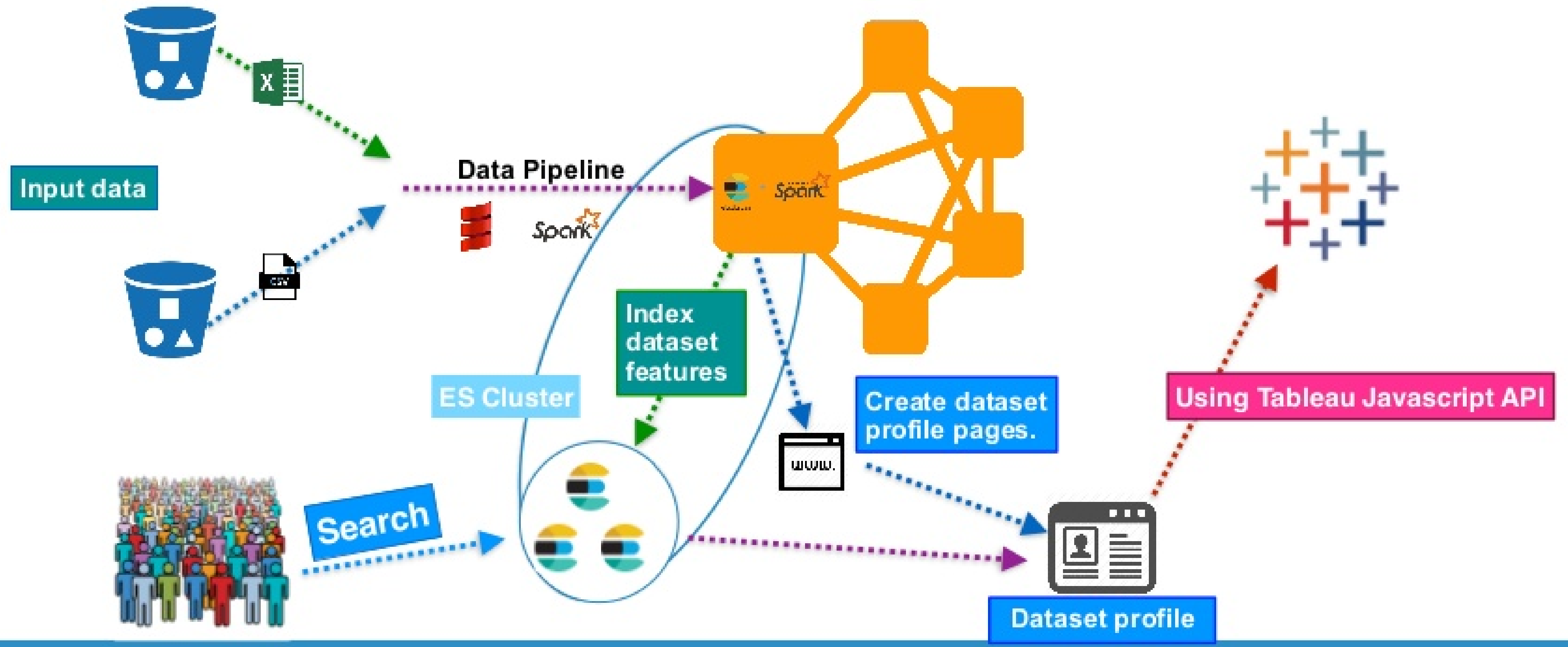


Organizing and Indexing Datasets



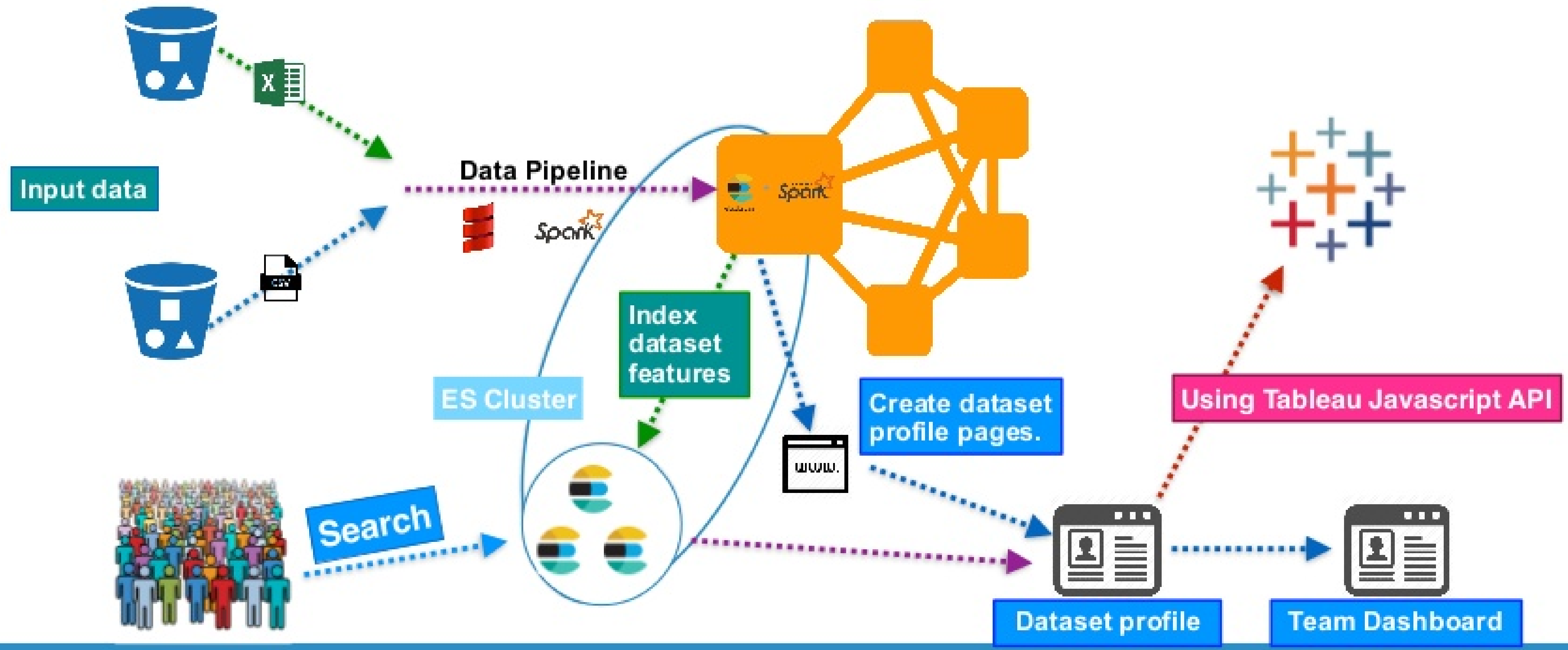
SPARK SUMMIT EAST 2017

Organizing and Indexing Datasets

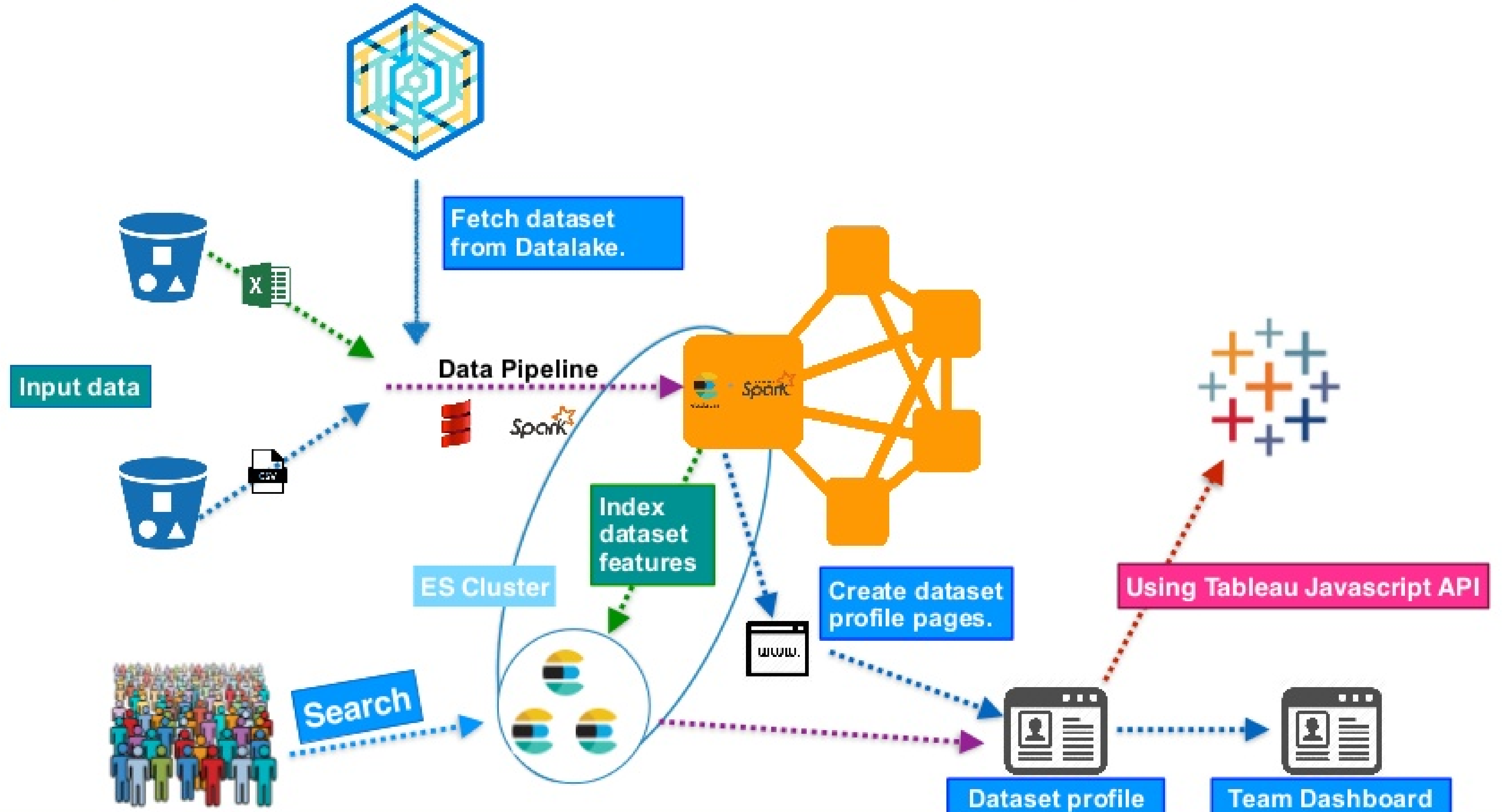


SPARK SUMMIT EAST 2017

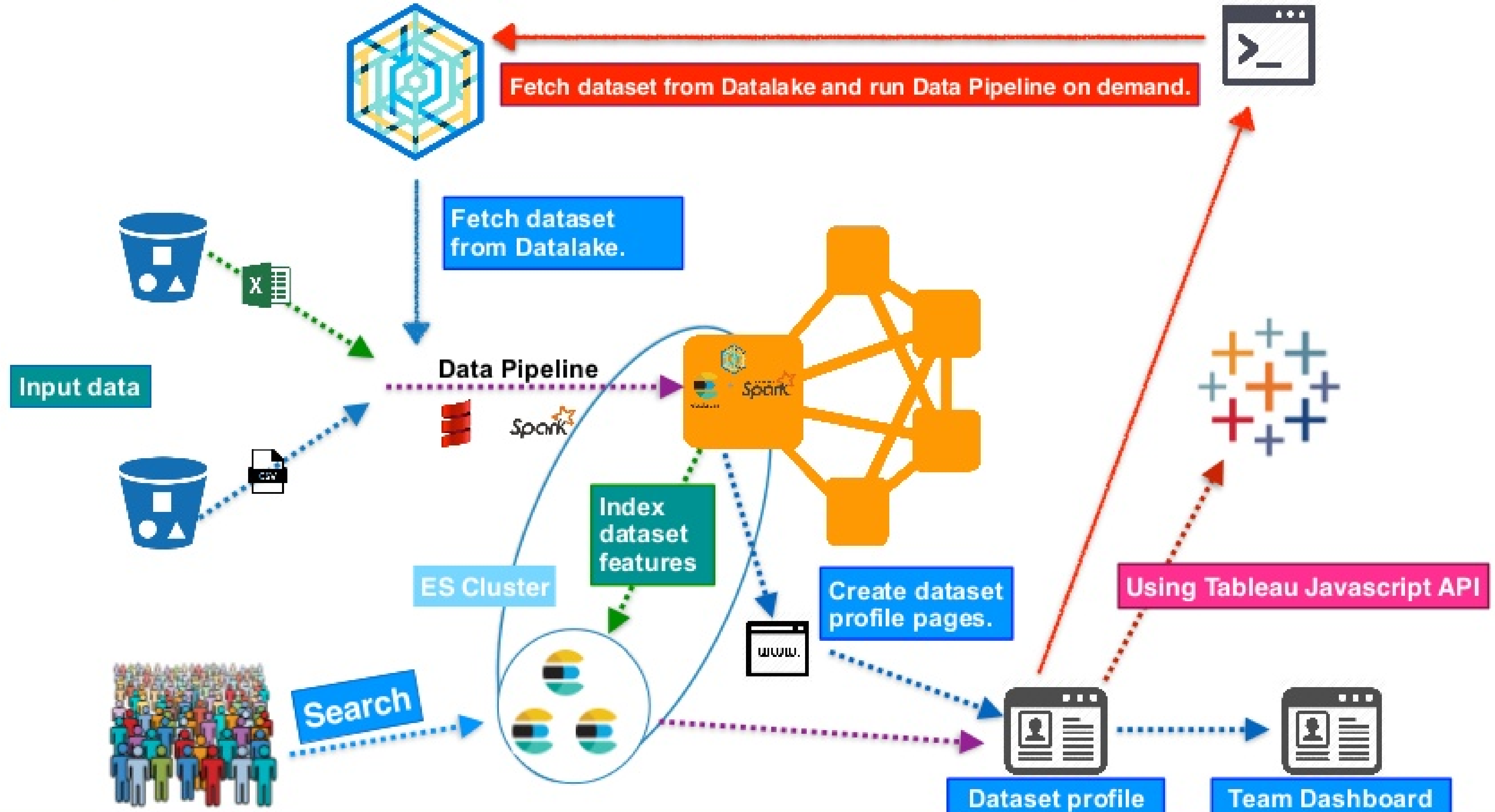
Organizing and Indexing Datasets

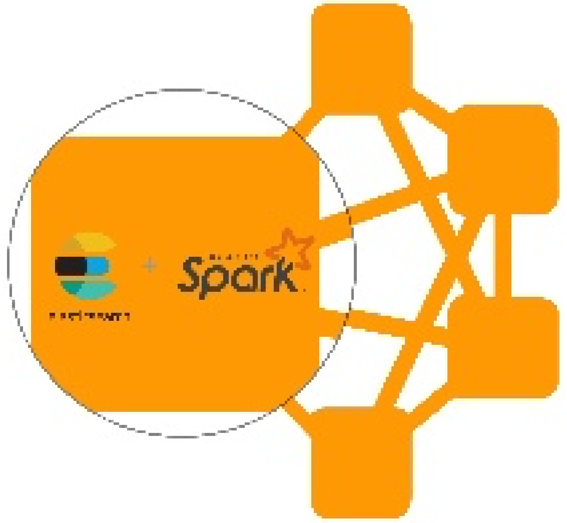


Organizing and Indexing Datasets



Organizing and Indexing Datasets


Fetch dataset from Datalake and run Data Pipeline on demand.
Fetch dataset from Datalake.
Input data
X
CSV
Data Pipeline
Spark
Spark
Index dataset features
ES Cluster
Create dataset profile pages.
www.
Using Tableau Javascript API
Search
Dataset profile
Team Dashboard



How do you run Elasticsearch inside Spark Cluster?

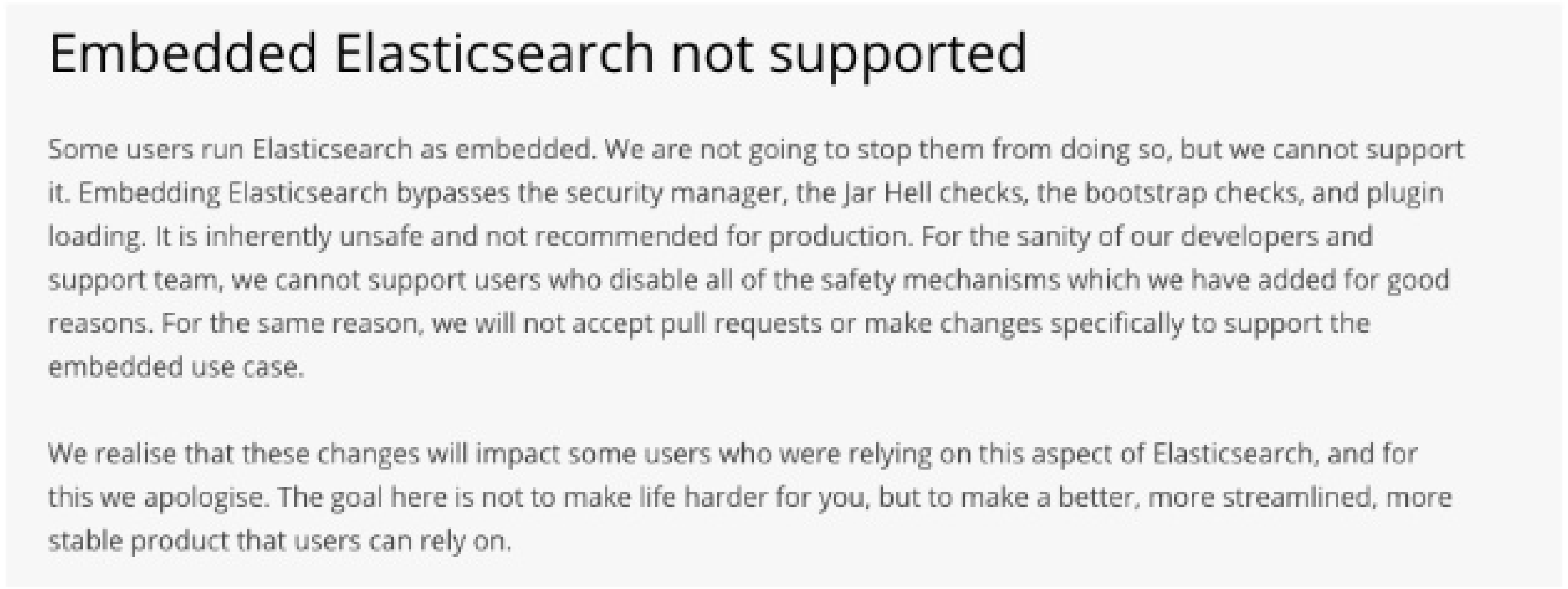
Preliminary Notes

- *Embedded* Elasticsearch 5 not supported.

  More information:

  - https://www.elastic.co/blog/elasticsearch-the-server
  - https://github.com/elastic/elasticsearch/issues/19903

http://bit.ly/2ehM9HO

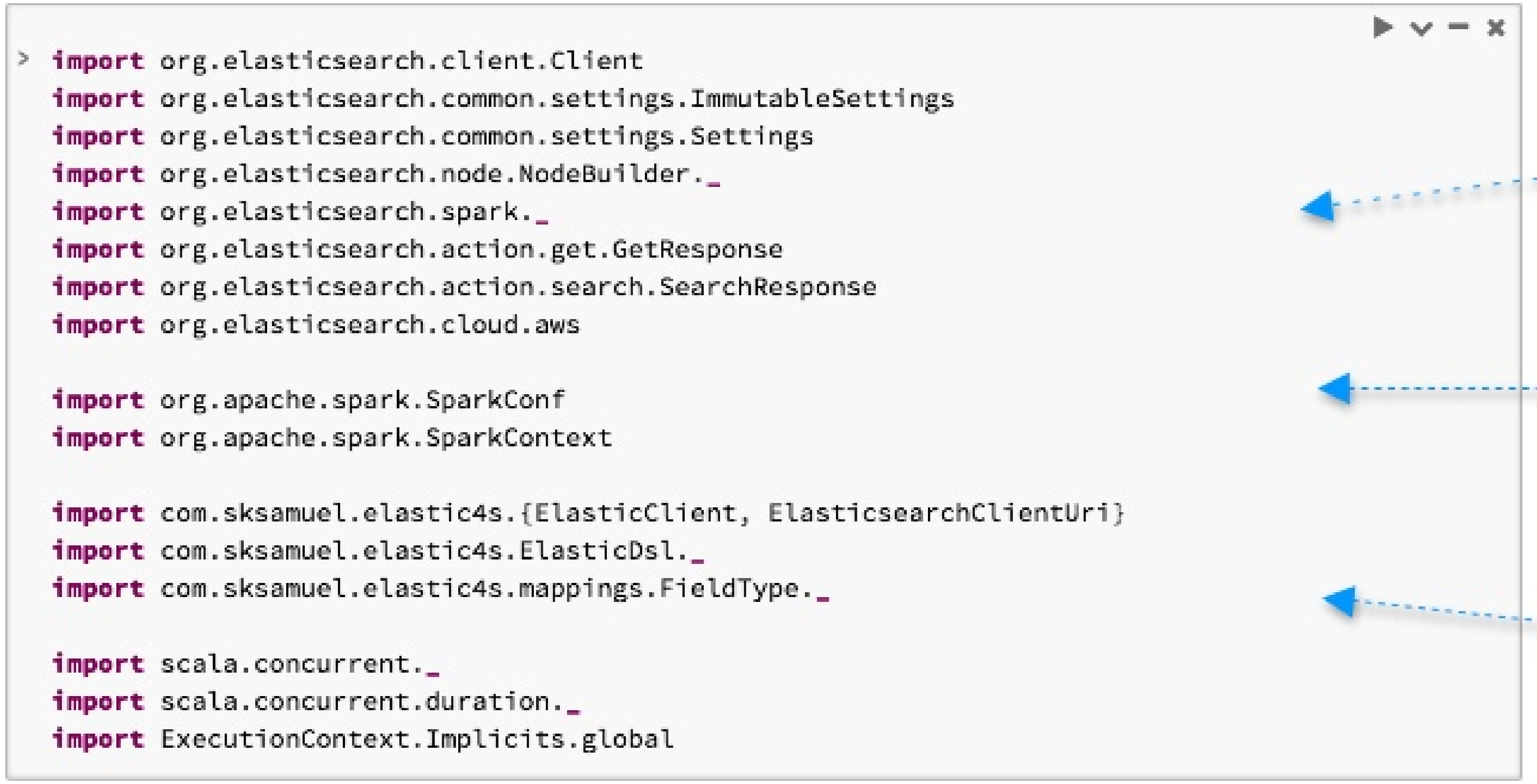


Embedded Elasticsearch not supported

Some users run Elasticsearch as embedded. We are not going to stop them from doing so, but we cannot support it. Embedding Elasticsearch bypasses the security manager, the Jar Hell checks, the bootstrap checks, and plugin loading. It is inherently unsafe and not recommended for production. For the sanity of our developers and support team, we cannot support users who disable all of the safety mechanisms which we have added for good reasons. For the same reason, we will not accept pull requests or make changes specifically to support the embedded use case.

We realise that these changes will impact some users who were relying on this aspect of Elasticsearch, and for this we apologise. The goal here is not to make life harder for you, but to make a better, more streamlined, more stable product that users can rely on.

Imports



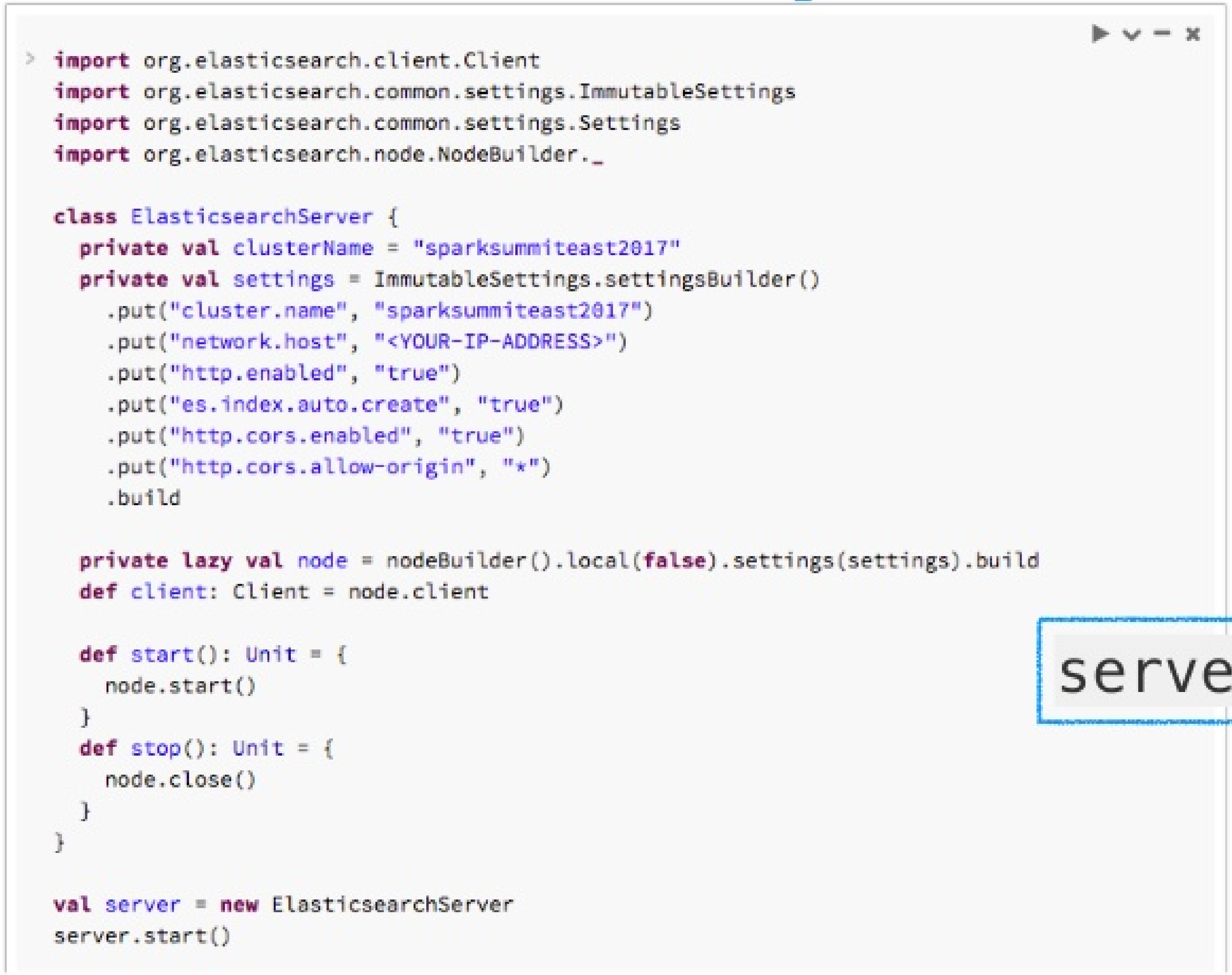
```
> import org.elasticsearch.client.Client
  import org.elasticsearch.common.settings.ImmutableSettings
  import org.elasticsearch.common.settings.Settings
  import org.elasticsearch.node.NodeBuilder._
  import org.elasticsearch.spark._
  import org.elasticsearch.action.get.GetResponse
  import org.elasticsearch.action.search.SearchResponse
  import org.elasticsearch.cloud.aws

  import org.apache.spark.SparkConf
  import org.apache.spark.SparkContext

  import com.sksamuel.elastic4s.{ElasticClient, ElasticsearchClientUri}
  import com.sksamuel.elastic4s.ElasticDsl._
  import com.sksamuel.elastic4s.mappings.FieldType._

  import scala.concurrent._
  import scala.concurrent.duration._
  import ExecutionContext.Implicits.global
```

http://bit.ly/2ebM9HO

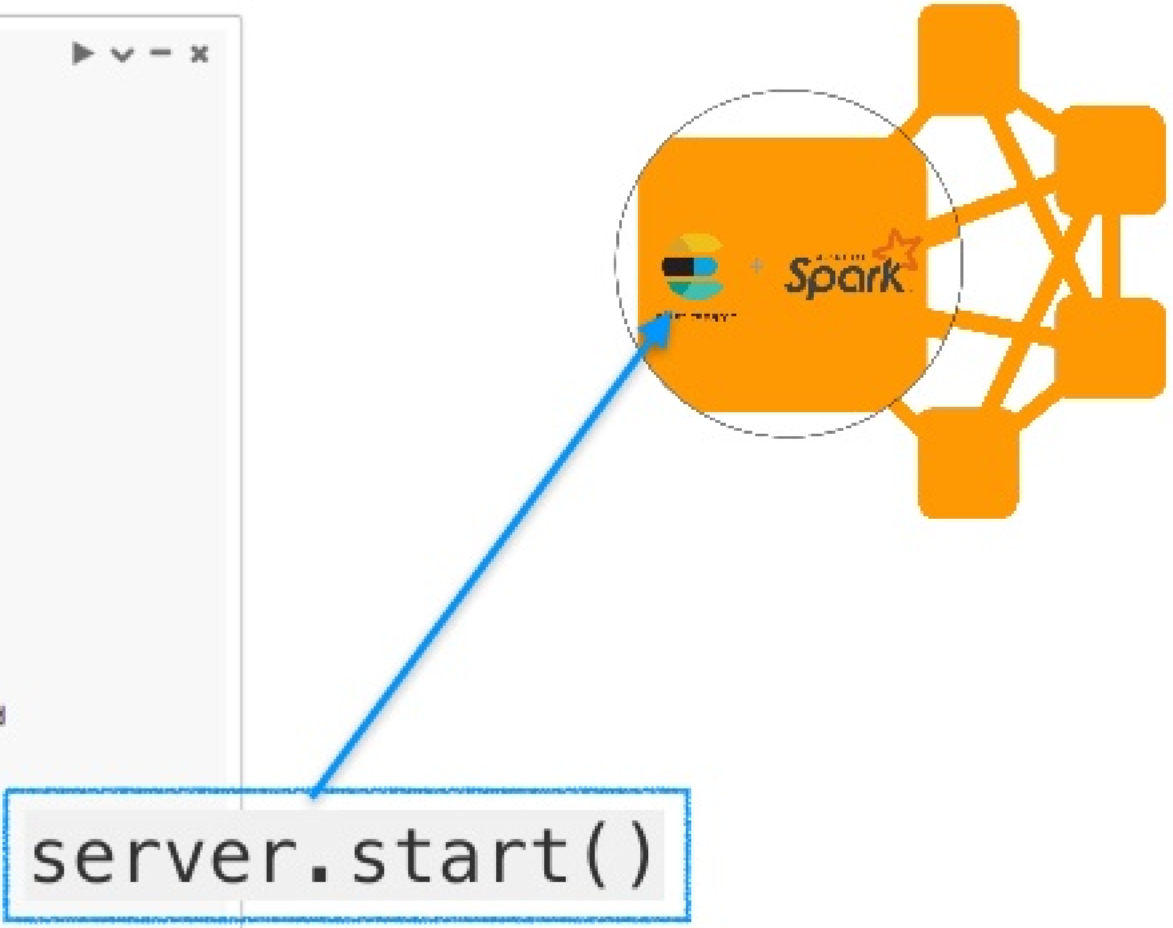
http://bit.ly/2dl0oEq

http://bit.ly/2efalb4

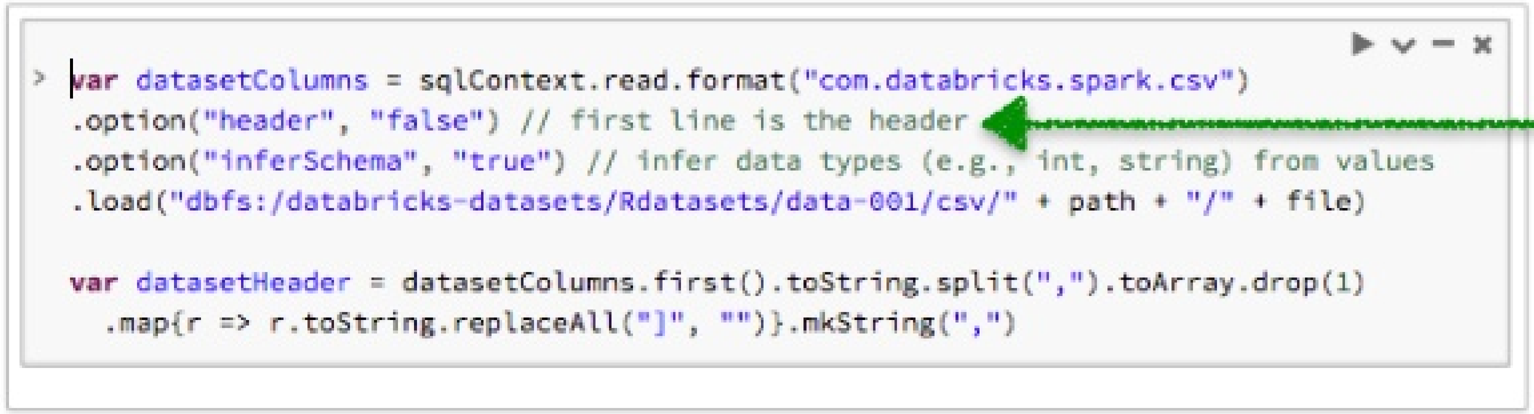
Setup Local ES



```
import org.elasticsearch.client.Client
import org.elasticsearch.common.settings.ImmutableSettings
import org.elasticsearch.common.settings.Settings
import org.elasticsearch.node.NodeBuilder._

class ElasticsearchServer {
  private val clusterName = "sparksummiteast2017"
  private val settings = ImmutableSettings.settingsBuilder()
    .put("cluster.name", "sparksummiteast2017")
    .put("network.host", "<YOUR-IP-ADDRESS>")
    .put("http.enabled", "true")
    .put("es.index.auto.create", "true")
    .put("http.cors.enabled", "true")
    .put("http.cors.allow-origin", "*")
    .build

  private lazy val node = nodeBuilder().local(false).settings(settings).build
  def client: Client = node.client

  def start(): Unit = {
    node.start()
  }
  def stop(): Unit = {
    node.close()
  }
}

val server = new ElasticsearchServer
server.start()
```



Metadata Extraction

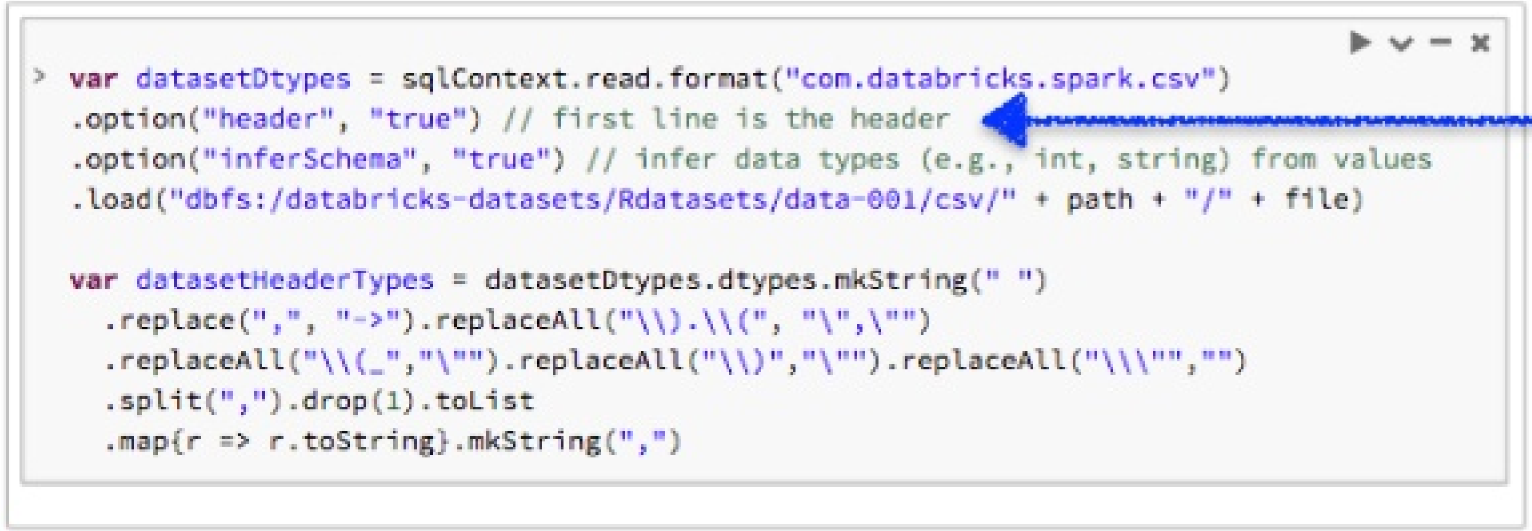


```
> var datasetColumns = sqlContext.read.format("com.databricks.spark.csv")
  .option("header", "false") // first line is the header
  .option("inferSchema", "true") // infer data types (e.g., int, string) from values
  .load("dbfs:/databricks-datasets/Rdatasets/data-001/csv/" + path + "/" + file)

  var datasetHeader = datasetColumns.first().toString.split(",").toArray.drop(1)
    .map{r => r.toString.replaceAll("]", "")}.mkString(",")
```

E.g. do not interpret first line as header

- Only first row
- Split by comma
- Remove additional characters
- Extract column-header names.

Result = <u>Comma separated list of column-header names.</u>



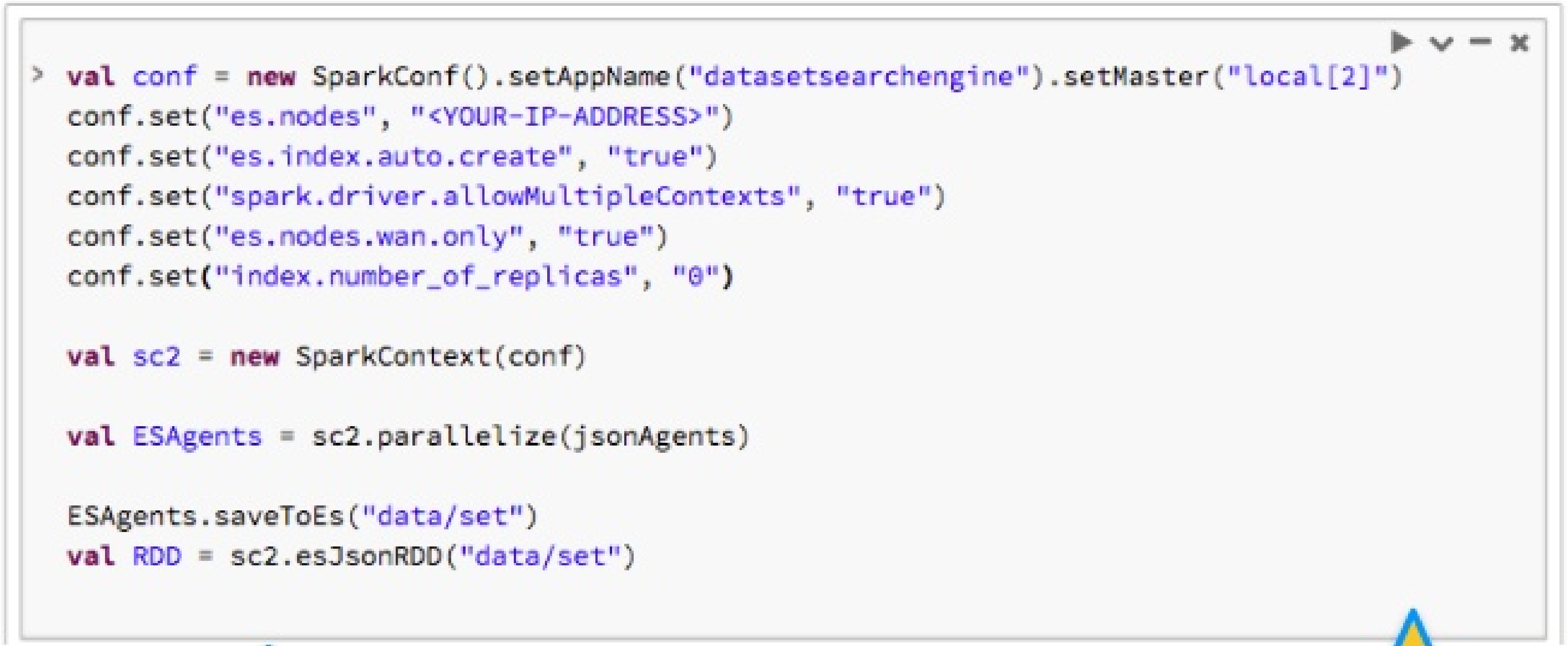
```
> var datasetDtypes = sqlContext.read.format("com.databricks.spark.csv")
  .option("header", "true") // first line is the header
  .option("inferSchema", "true") // infer data types (e.g., int, string) from values
  .load("dbfs:/databricks-datasets/Rdatasets/data-001/csv/" + path + "/" + file)

  var datasetHeaderTypes = datasetDtypes.dtypes.mkString(" ")
    .replace(",", "->").replaceAll("\\).\\(", "\",\"")
    .replaceAll("\\(_","\"").replaceAll("\\)","\"").replaceAll("\\\"","")
    .split(",").drop(1).toList
    .map{r => r.toString}.mkString(",")
```

E.g. do interpret first line as header

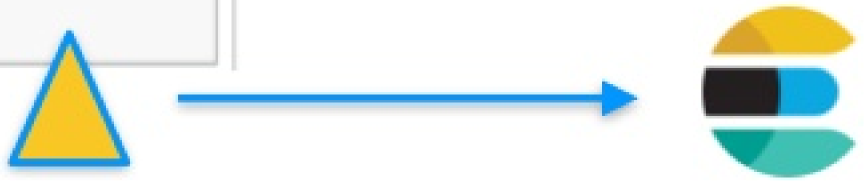
- run ‘dtypes’ on Data Frame
- Split by comma
- Remove additional characters
- Extract column-header types.

Result = <u>Comma separated list of column-header types.</u>

Writing to local ES

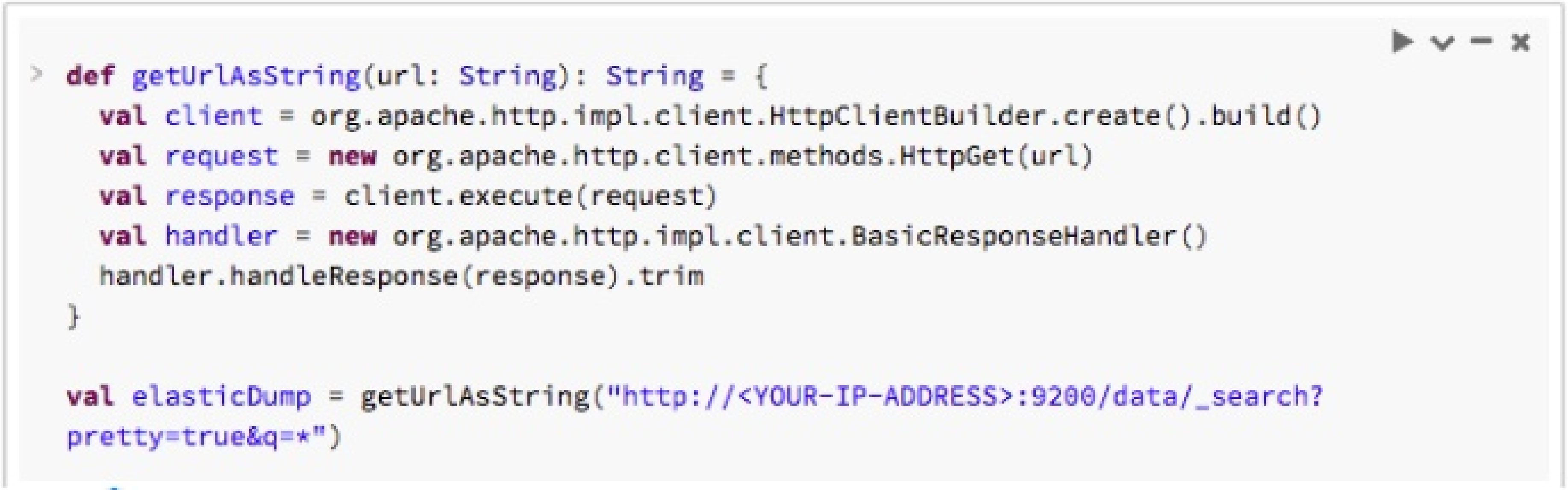


```
> val conf = new SparkConf().setAppName("datasetsearchengine").setMaster("local[2]")
  conf.set("es.nodes", "<YOUR-IP-ADDRESS>")
  conf.set("es.index.auto.create", "true")
  conf.set("spark.driver.allowMultipleContexts", "true")
  conf.set("es.nodes.wan.only", "true")
  conf.set("index.number_of_replicas", "0")

  val sc2 = new SparkContext(conf)

  val ESAgents = sc2.parallelize(jsonAgents)

  ESAgents.saveToEs("data/set")
  val RDD = sc2.esJsonRDD("data/set")
```



```
saveToEs("data/set")
```

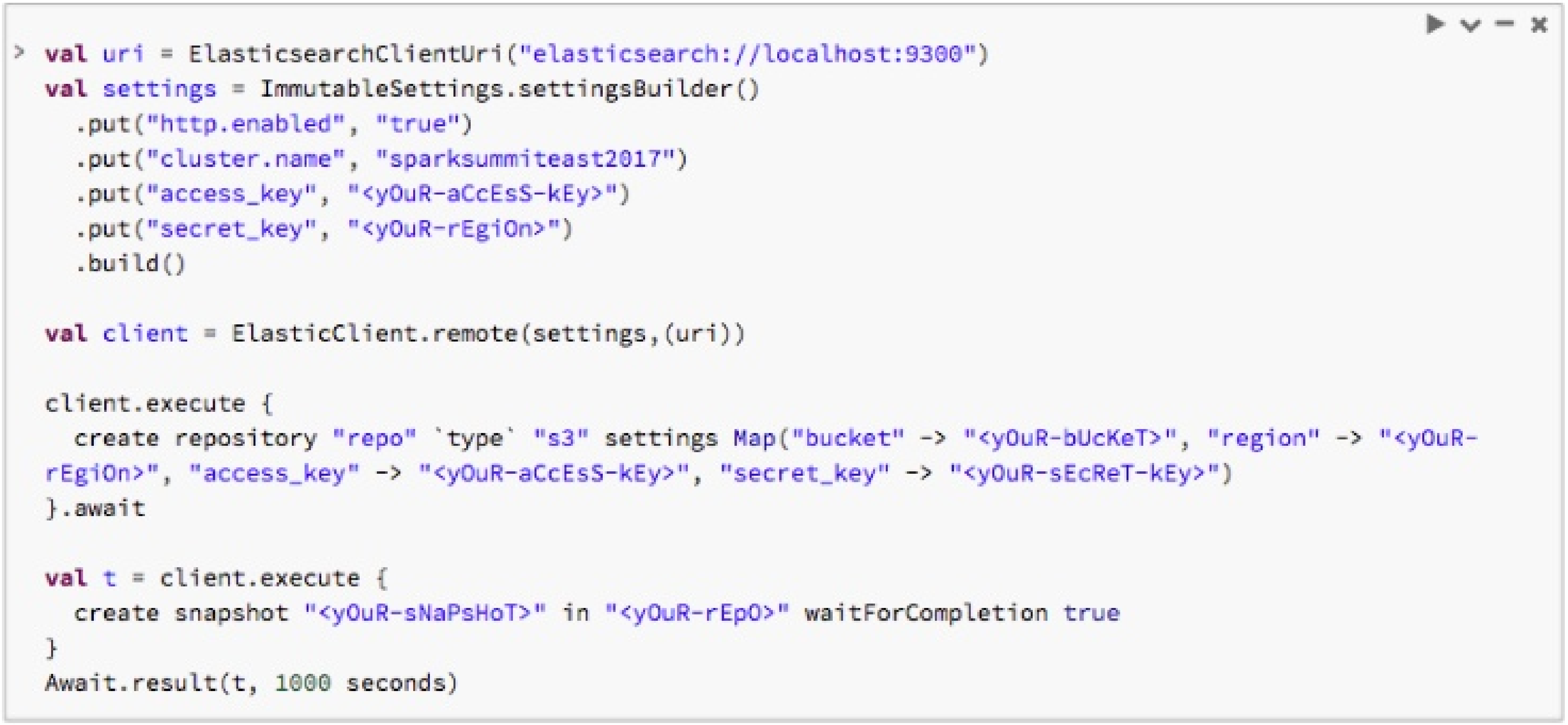
Check results on local ES



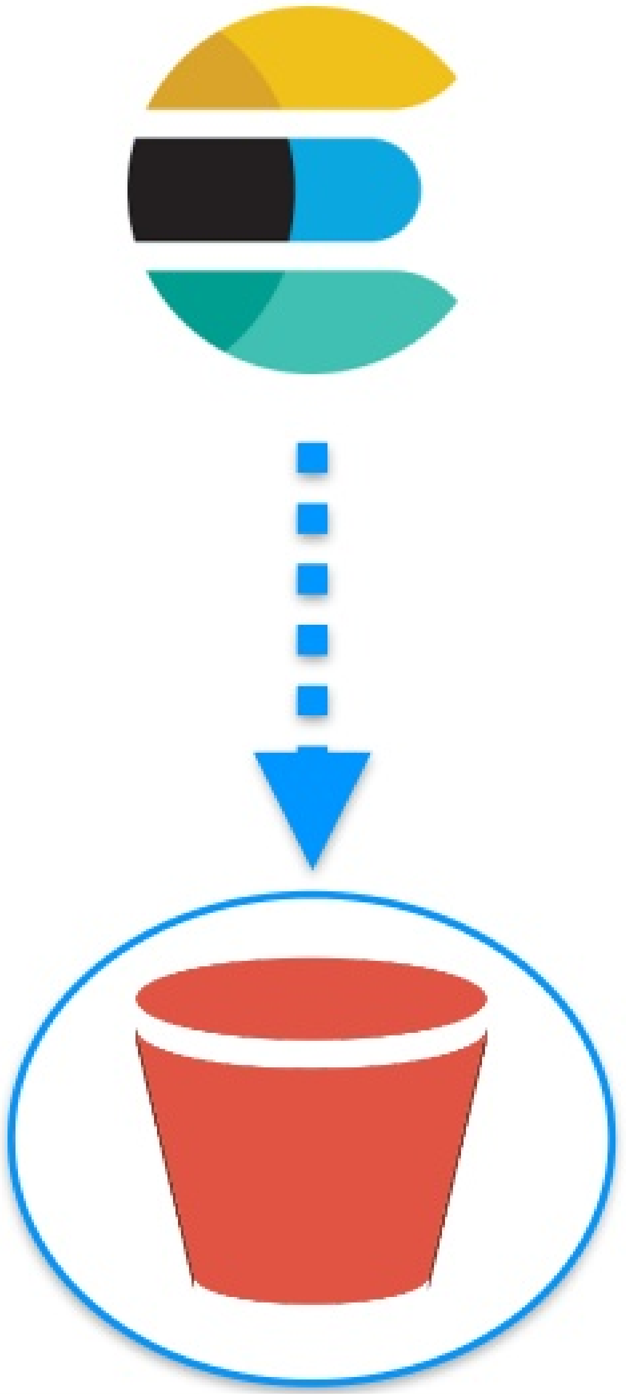
```
> def getUrlAsString(url: String): String = {
    val client = org.apache.http.impl.client.HttpClientBuilder.create().build()
    val request = new org.apache.http.client.methods.HttpGet(url)
    val response = client.execute(request)
    val handler = new org.apache.http.impl.client.BasicResponseHandler()
    handler.handleResponse(response).trim
  }

  val elasticDump = getUrlAsString("http://<YOUR-IP-ADDRESS>:9200/data/_search?
  pretty=true&q=*")
```

```
getUrlAsString("http://<YOUR-IP-ADDRESS>:9200/_cat/indicies?v")
```

Snapshot to S3



```
> val uri = ElasticsearchClientUri("elasticsearch://localhost:9300")
  val settings = ImmutableSettings.settingsBuilder()
    .put("http.enabled", "true")
    .put("cluster.name", "sparksummiteast2017")
    .put("access_key", "<yOuR-aCcEsS-kEy>")
    .put("secret_key", "<yOuR-rEgiOn>")
    .build()

  val client = ElasticClient.remote(settings,(uri))

  client.execute {
    create repository "repo" `type` "s3" settings Map("bucket" -> "<yOuR-bUcKeT>", "region" -> "<yOuR-
  rEgiOn>", "access_key" -> "<yOuR-aCcEsS-kEy>", "secret_key" -> "<yOuR-sEcReT-kEy>")
  }.await

  val t = client.execute {
    create snapshot "<yOuR-sNaPsHoT>" in "<yOuR-rEpO>" waitForCompletion true
  }
  Await.result(t, 1000 seconds)
```



Demo!

A posteriori vs. Post-hoc

- Halevy et al (2016) advocate finding data in a *post-hoc* manner by collecting and aggregating metadata *after* datasets are created or updated.

- I propose an *a posteriori* approach where metadata is *generated* and indexed using Spark as part of running pipelines.

Pros: A posteriori (1)

- Gathering metadata *and* recovering dataset semantics does not require separate processing.
- Dataset normalization to a single “dataset” concept becomes feasible.
- More granular metrics available to evaluate metadata regeneration.

Pros: A posteriori (2)

- Granular (per-pipeline) access to recompute dataset importance based on feedback from user interaction.
    - E.g. Datasets with increased user interaction call for more in-depth metadata extraction.
- Granular (per-pipeline) access to status metadata.

Cons: A posteriori (2)

- Looking at trees instead of the forest.
- Cluster-type (or higher order) analyses limited in-pipeline (tunnel vision).
- Need to replay indexing pipeline when things change (per data pipeline).

What have we seen?

- How to create ES Server inside Spark Cluster.
- How to write to Elasticsearch index + snapshot to S3.
- How to extract metadata inside data pipelines.
- Demo: RDataset Search Engine with Spark and Elasticsearch.

Next Steps (1)

- **Replicate on embedded Solr** using spark-solr plugin to read/write from/to Spark RDD's instead of Elasticsearch.
- **Naïvely implement Online Learning to Rank (OLTR) on Dataset index** using Solr Online Learning to Rank Plugin (Hofmann, 2013) [1].
- **Describe Datasets in a structured schema.org way** using Data Catalog Vocabulary [2].

[1] https://bitbucket.org/ilps/solr-online-learning-to-rank-plug-in
[2] http://www.w3.org/TR/vocab-dcat/

Next Steps (2)

- **Develop methods to extract relations among Datasets and rank those using OLTR** (interleave (Hofmann, 2013) or multileave (Schuth, 2016)) *in place of human relevance judgements* (as with featureRank in (Balakrishnan, et al. 2015)).

- **Build a knowledge graph** and use GraphX to extract insights. (Useful e.g. for column concept determination (Deng et al. 2013)).

- **Build topic models based on structured Datasets** using Glint to perform scalable topic model extraction in Spark (Jagerman and Eickhoff, 2016) [1].

[1] https://spark-summit.org/eu-2016/events/glint-an-asynchronous-parameter-server-for-spark/

References

- Alon Y. Halevy, Flip Korn, Natalya Fridman Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. Goods: Organizing google's datasets. In Fatmañzcan, Georgia Koutrika, and Sam Madden, editors, Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016, pages 795–806. ACM, 2016. ISBN 978-1-4503-3531-7. doi: http://doi.acm.org/10.1145/2882903.2903730.
- Katja Hofmann. Fast and Reliable Online Learning to Rank for Information Retrieval. PhD thesis, Informatics Institute, University of Amsterdam, May 2013.
- Rolf Jagerman and Carsten Eickhoff. Web-scale topic models in spark: An asynchronous parameter server. CoRR, abs/ 1605.07422, 2016. URL http://arxiv.org/abs/1605.07422.
- Dong Deng, Yu Jiang, Guoliang Li, Jian Li, and Cong Yu. Scalable column concept de- termination for web tables using large knowledge bases. PVLDB, 6(13):1606–1617, 2013. doi: http://www.vldb.org/pvldb/vol6/p1606-li.pdf.
- Anne Schuth, Harrie Oosterhuis, Shimon Whiteson, and Maarten de Rijke. Multileave gradient descent for fast online learning to rank. In *WSDM 2016: The 9th International Conference on Web Search and Data Mining*, pages 457-466. ACM, February 2016.
- Sreeram Balakrishnan, Alon Y. Halevy, Boulos Harb, Hongrae Lee, Jayant Madhavan, Afshin Rostamizadeh, Warren Shen, Kenneth Wilder, Fei Wu 0003, and Cong Yu. Ap- plying webtables in practice. In CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings. www.cidrdb.org, 2015.

SPARK SUMMIT EAST 2017

Q&A

SPARK SUMMIT
EAST 2017

Thank You.

Email: ocastaneda@paypal.com
Twitter: @oscar_castaneda

SPARK SUMMIT EAST 2017