STRATIO

Kerberizing Spark

Spark Summit East

February 2017

INDEX

Jorge López-Malla Matute
jlopezm@stratio.com
@jorgelopezmalla

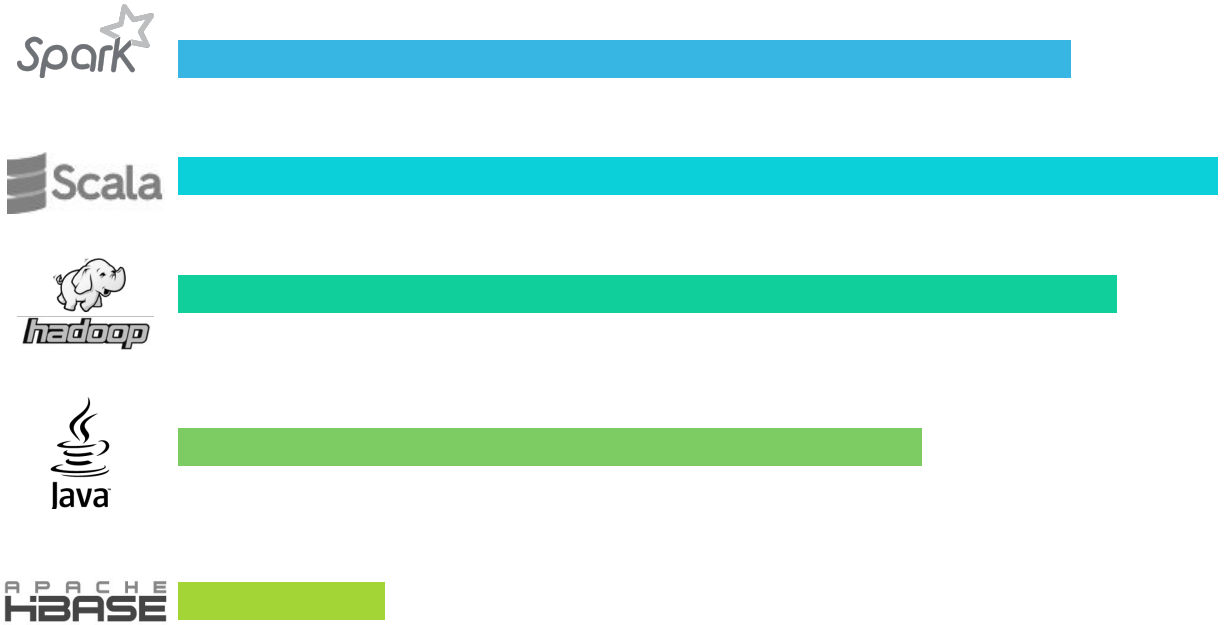
Abel Rincón Matarranz
arincon@stratio.com
@ganger85

Presentation

JORGE LÓPEZ-MALLA

After working with traditional processing methods, I started to do some R&S Big Data projects and I fell in love with the Big Data world. Currently i'm doing some awesome Big Data projects and tools at Stratio.

SKILLS



Presentation

ABEL RINCÓN MATARRANZ

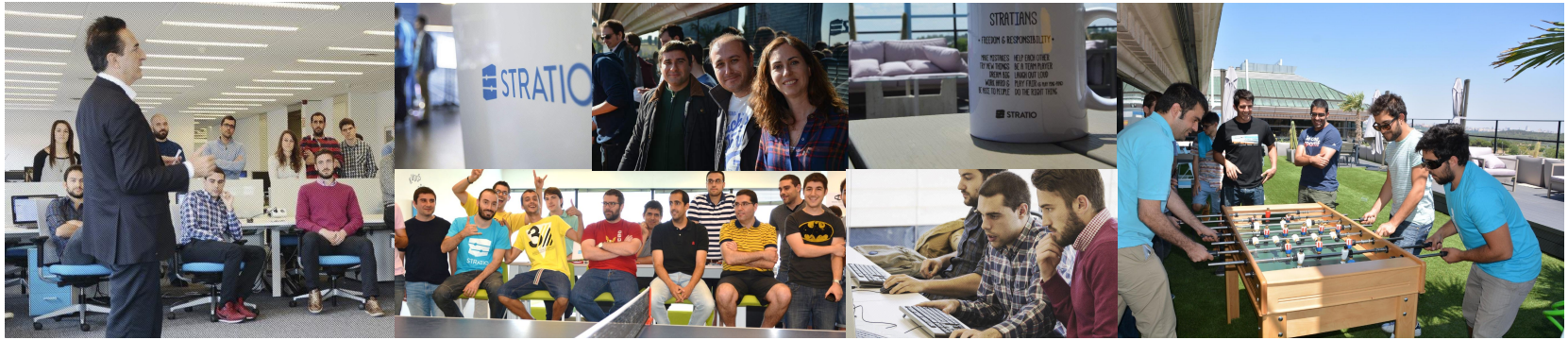
SKILLS

Our company

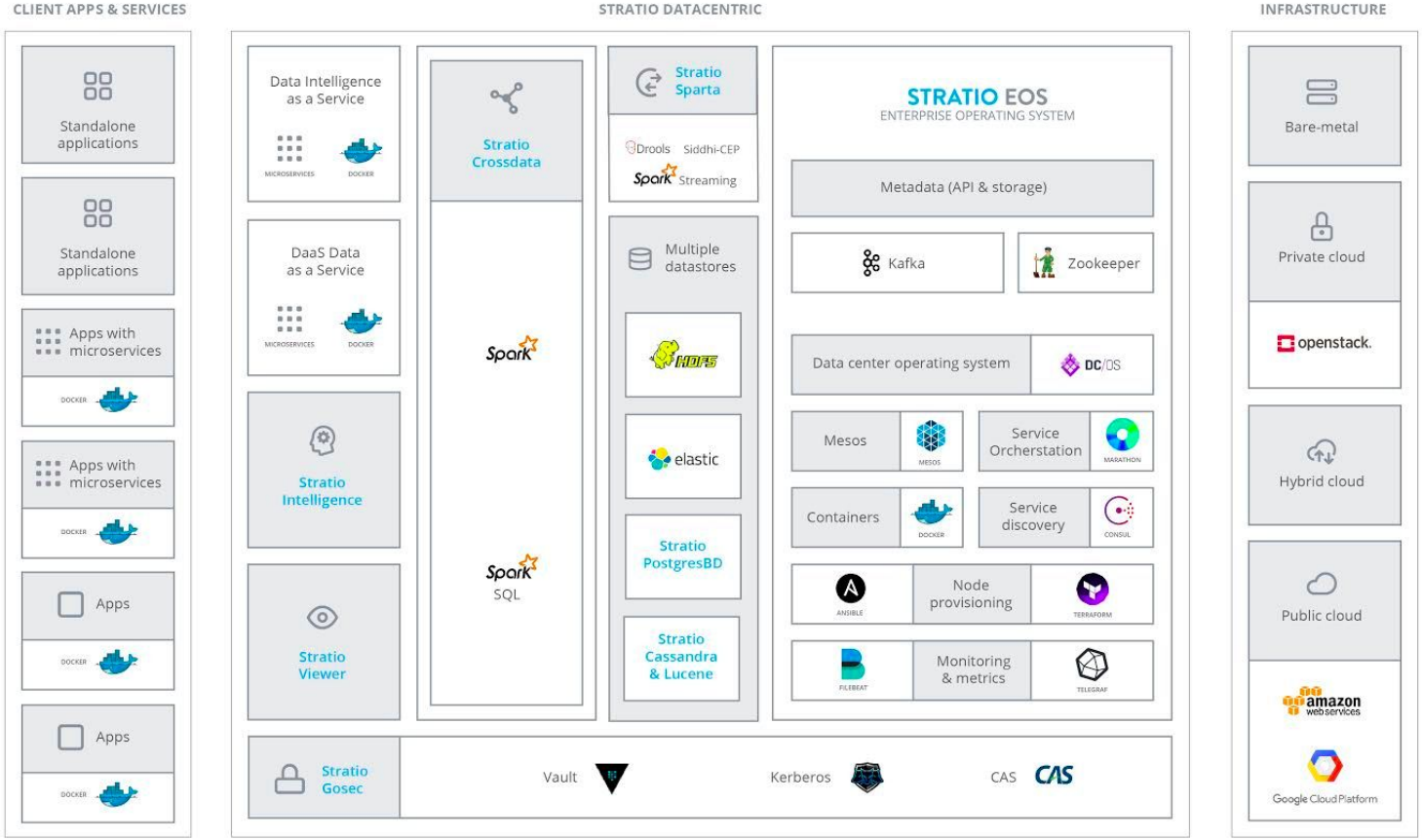
Stratio accompanies businesses on their journey through complete Digital Transformation. Through hard work and creativity, our ambition is to reinvent companies around their data so they can compete in a changed world.

Our solution is a transformational product that uses third generation Big Data technologies to execute the most comprehensive form of Digital Transformation, ensuring scalability, maximum flexibility and adaptation to new markets.

If you have any questions or would like a demonstration of our product, get in touch: **contact@stratio.com**



Our product


CLIENT APPS & SERVICES
Standalone applications
Standalone applications
Apps with microservices
Apps with microservices
Apps
Apps
STRATIO DATACENTRIC
Data Intelligence as a Service
DaaS Data as a Service
Stratio Intelligence
Stratio Viewer
Stratio Crossdata
Spark
Spark SQL
Stratio Sparta
Drools
Siddhi-CEP
Spark Streaming
Multiple datastores
HDFS
elastic
Stratio PostgresBD
Stratio Cassandra & Lucene
STRATIO EOS
ENTERPRISE OPERATING SYSTEM
Metadata (API & storage)
Kafka
Zookeeper
Data center operating system
DC/OS
Mesos
Service Orcherstation
Containers
Service discovery
Node provisioning
Monitoring & metrics
Stratio Gosec
Vault
Kerberos
CAS
INFRASTRUCTURE
Bare-metal
Private cloud
openstack.
Hybrid cloud
Public cloud
amazon web services
Google Cloud Platform

1 Kerberos

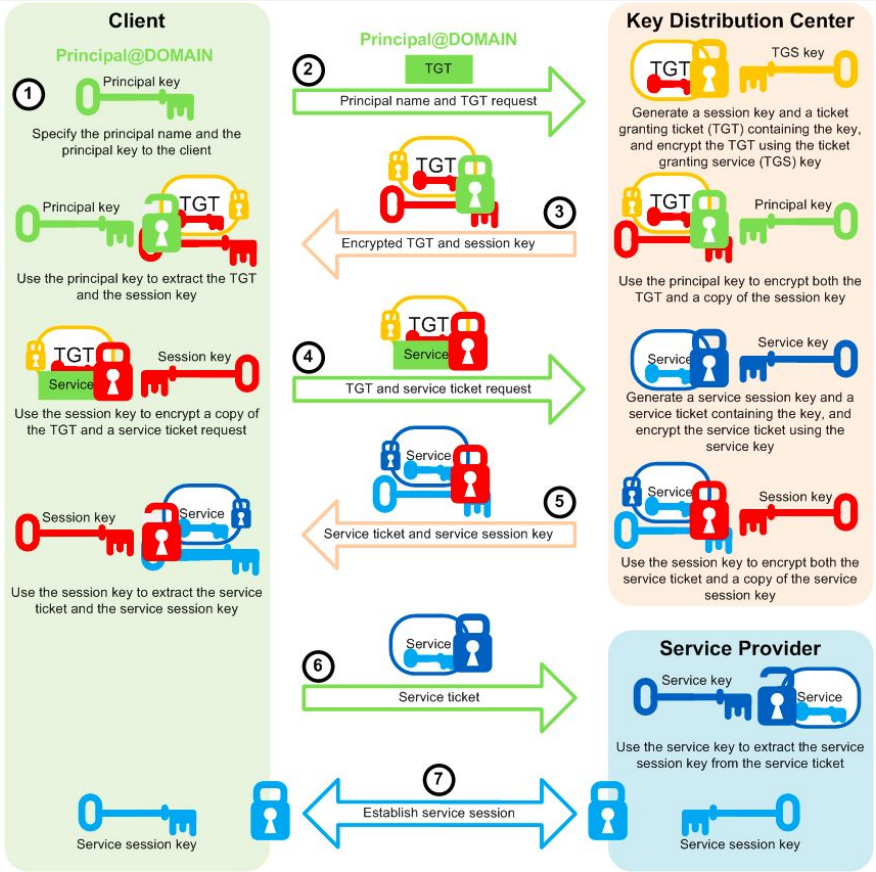
Kerberos

- What is Kerberos
  - Authentication protocol / standard / service
    - Safe
    - Single-sign-on
    - Trust based
    - Mutual authentication

Kerberos key concepts

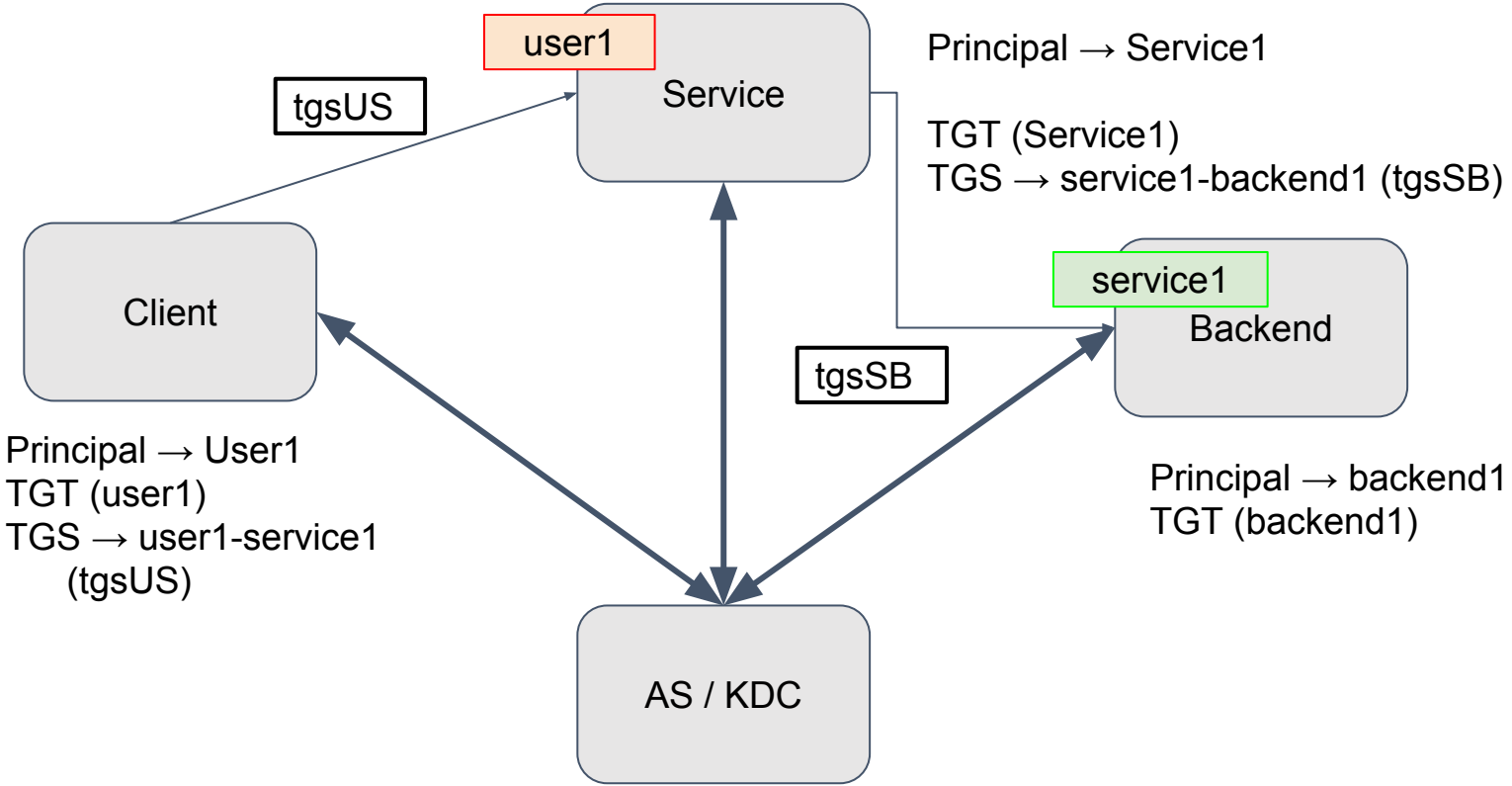
- Client/Server → Do you need an explanation???
- Principal → Identify a unique client or service
- Realm → Identify a environment, company, domain ...
  - DEMO.EAST.SUMMIT.SPARK.ORG
- KDC → Actor who manages the tickets
- TGT → Ticket which has the client session
- TGS → Ticket which has the client-service session

Kerberos Workflow

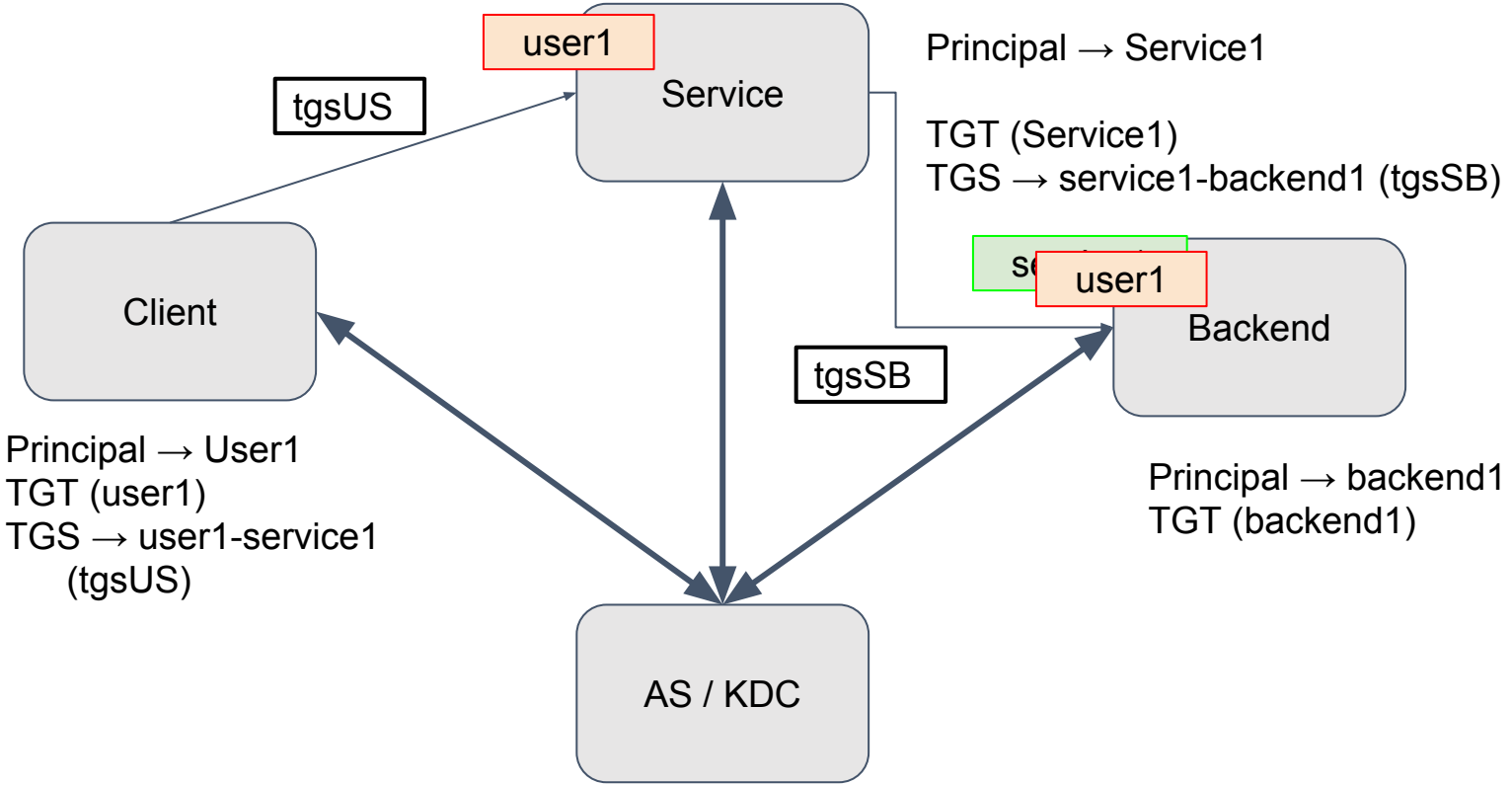


1. Client retrieve principal and secret
2. Client performs a TGT request
3. KDC returns TGT
4. Client request a TGS with the TGT
5. KDC returns the TGS
6. Client request a service session using the TGS
7. Service establish a secure connection directly with the client

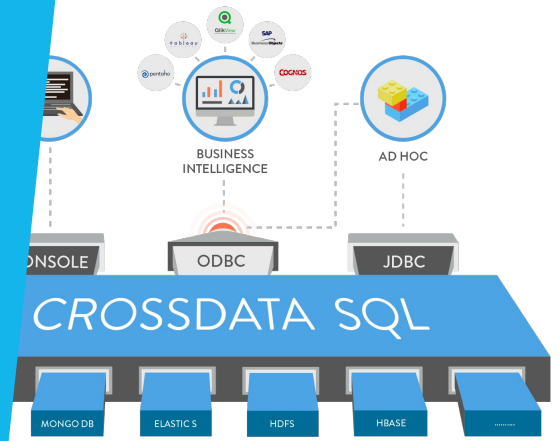
Kerberos workflow 2



Kerberos workflow - Impersonation



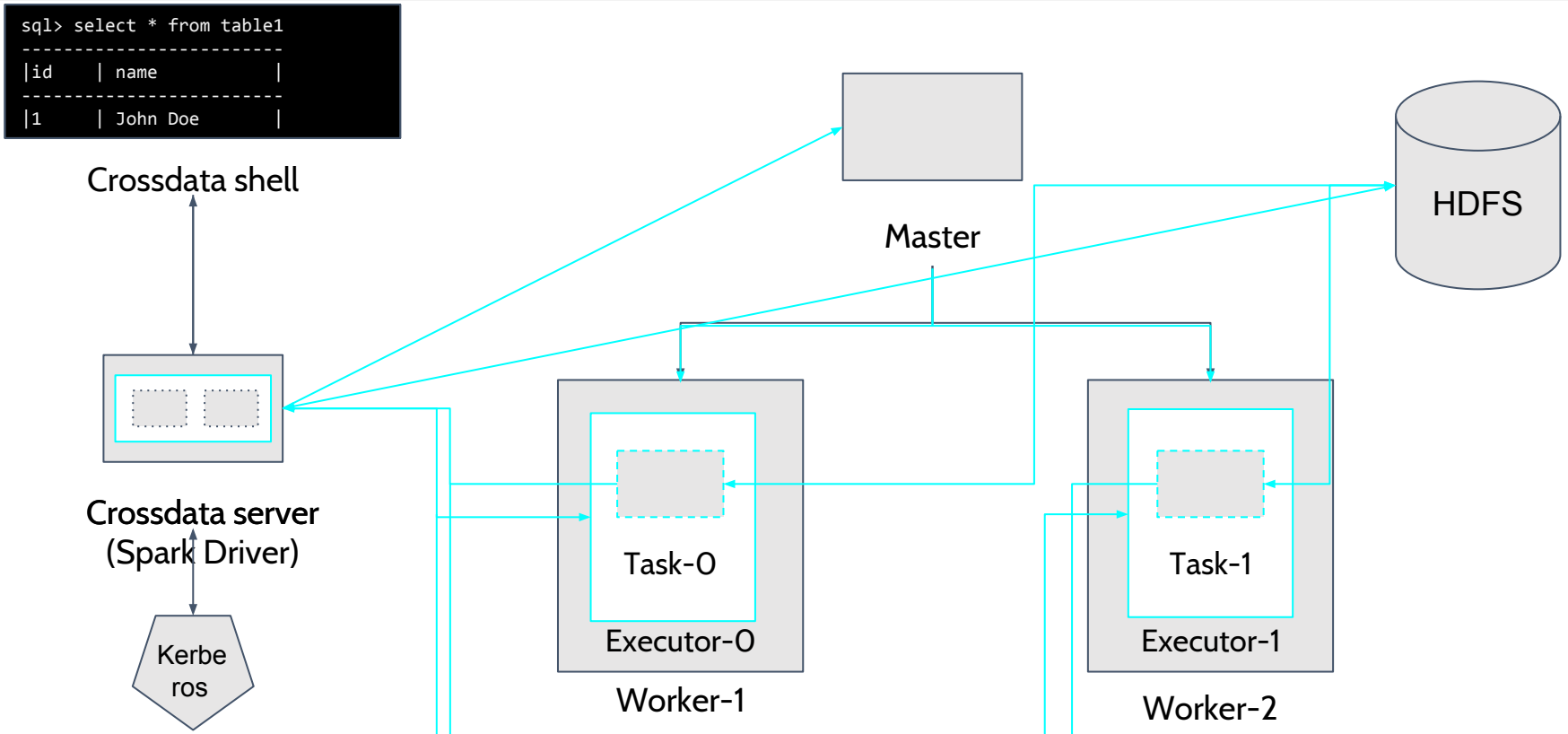
2 Use Case


BUSINESS
INTELLIGENCE
AD HOC
ONSOLE
ODBC
JDBC
CROSSDATA SQL
MONGO DB
ELASTIC S
HDFS
HBASE

Use Case

- Stratio Crossdata is a distributed framework and a fast and general-purpose computing system powered by Apache Spark
- Can be used both as library and as a server.
- Crossdata Server: Provides a multi-user environment to SparkSQL, giving a reliable architecture with high-availability and scalability out of the box
- To do so it use both native queries and Spark
- Crossdata Server had a unique long time SparkContext to execute all its Sparks queries
- Crossdata can use YARN, Mesos and Standalone as a resource manager

Crossdata as Server



Crossdata in production

- Projects in production needs runtime impersonation to be compliance the AAA(Authorization, Authentication and Audit) at the storage.
- Crossdata allows several users per execution
- Neither of the Sparks resource managers allows us to impersonate in runtime.
- Evenmore Standalone as resource manager does not provide any Kerberos feature.

3 Stratio Solution

Prerequirement

- Keytab have to be accessible in all the cluster machines
- Keytab must provide proxy grants
- Hadoop client configuration located in the cluster
- Each user, both proxy and real, must have a home in HDFS

Introduction

- Spark access to the storage system both in the Driver and the Executors.
- In the Driver side both Spark Core and SparkSQL will access to the storage system.
- Executors will always access via Task.
- As Streaming use the same classes than SparkCore or SparkSQL the same solution will be usable by Streaming jobs

KerberosUser (Utils)

```scala
object KerberosUser extends Logging with UserCache {

def setProxyUser(user: String): Unit = proxyUser = Option(user)

def getUserByName(name: Option[String]): Option[UserGroupInformation] = {
 if (getConfiguration.isDefined) {
   userFromKeyTab(name)
 }
 else None
}
private def userFromKeyTab(proxyUser: Option[String]): Option[UserGroupInformation] = {
 if (realUser.isDefined) realUser.get.checkTGTAndReloginFromKeytab()
 (realUser, proxyUser) match {
   case (Some(_), Some(proxy)) => users.get(proxy).orElse(loginProxyUser(proxy))
   case (Some(_), None) => realUser
   case (None, None) => None
 }
}
private lazy val getConfiguration: Option[(String, String)] = {
 val principal = env.conf.getOption("spark.executor.kerberos.principal")
 val keytab = env.conf.getOption("spark.executor.kerberos.keytab")
 (principal, keytab) match {
   case (Some(p), Some(k)) => Option(p, k)
   case _ => None
 }
}
```

setting proxy user (Global)

Public Method retrieve user

Choose between real or proxy user

Configuration

Wrappers (Utils)

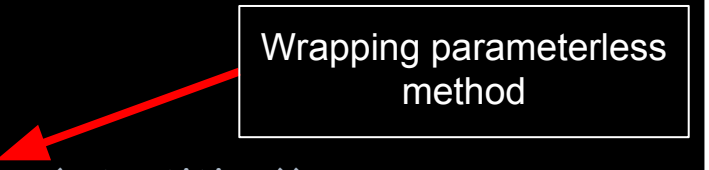
```scala
def executeSecure[U, T](proxyUser: Option[String],
                        funct: (U => T),
                        inputParameters: U): T = {
 KerberosUser.getUserByName(proxyUser) match {
   case Some(user) => {
     user.doAs(new PrivilegedExceptionAction[T]() {
       @throws(classOf[Exception])
       def run: T = {
         funct(inputParameters)
       }
     })
   }
   case None => {
     funct(inputParameters)
   }
 }
}
```

```scala
def executeSecure[T](exe: ExecutionWrp[T]): T = {
 KerberosUser.getUser match {
   case Some(user) => {
     user.doAs(new PrivilegedExceptionAction[T]() {
       @throws(classOf[Exception])
       def run: T = {
         exe.value
       }
     })
   }
   case None => exe.value
 }
}

class ExecutionWrp[T](wrp: => T) {
 lazy val value: T = wrp
}
```

Driver Side

```
abstract class RDD[T: ClassTag](
   @transient private var _sc: SparkContext,
   @transient private var deps: Seq[Dependency[_]]
 ) extends Serializable with Logging {
...
/**
* Get the array of partitions of this RDD, taking into account whether the
* RDD is checkpointed or not.
*/
final def partitions: Array[Partition] = {
 checkpointRDD.map(_.partitions).getOrElse {
   if (partitions_ == null) {
     partitions_ = KerberosFunction.executeSecure(new ExecutionWrp(getPartitions))
     partitions_.zipWithIndex.foreach { case (partition, index) =>
       require(partition.index == index,
         s"partitions($index).partition == ${partition.index}, but it should equal $index")
     }
   }
   partitions_
 }
}
```



Driver Side

```
class PairRDDFunctions[K, V](self: RDD[(K, V)])
   (implicit kt: ClassTag[K], vt: ClassTag[V], ord: Ordering[K] = null)
...
def saveAsHadoopDataset(conf: JobConf): Unit = self.withScope {
 // Rename this as hadoopConf internally to avoid shadowing (see SPARK-2038).

 val internalSave: (JobConf => Unit) = (conf: JobConf) => {
   val hadoopConf = conf
   val outputFormatInstance = hadoopConf.getOutputFormat
   val keyClass = hadoopConf.getOutputKeyClass
   val valueClass = hadoopConf.getOutputValueClass

    ...
   val writeToFile = (context: TaskContext, iter: Iterator[(K, V)]) => {
    ....
   }
   self.context.runJob(self, writeToFile)
   writer.commitJob()
 }
KerberosFunction.executeSecure(internalSave, conf)
}
```

Hadoop Datastore RDD save function

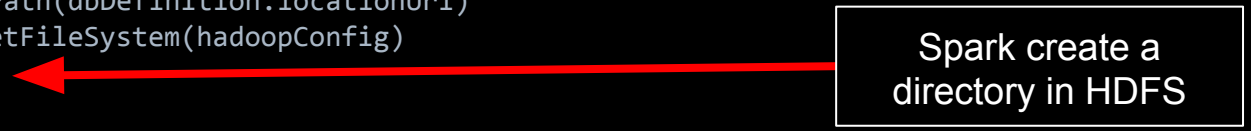
Inside wrapper function that will run in the cluster

Kerberos authentified save function

Driver Side

```
class InMemoryCatalog(
   conf: SparkConf = new SparkConf,
   hadoopConfig: Configuration = new Configuration)

override def createDatabase(
   dbDefinition: CatalogDatabase,
   ignoreIfExists: Boolean): Unit = synchronized {
 def inner: Unit = {
       ...
       val location = new Path(dbDefinition.locationUri)
       val fs = location.getFileSystem(hadoopConfig)
       fs.mkdirs(location)
     } catch {
       case e: IOException =>
         throw new SparkException(s"Unable to create database ${dbDefinition.name} as failed " +
           s"to create its directory ${dbDefinition.locationUri}", e)
     }
     catalog.put(dbDefinition.name, new DatabaseDesc(dbDefinition))
   }
 }
 KerberosFunction.executeSecure(KerberosUser.principal, new ExecutionWrp(inner))
}
```



Driver Side

```
* Interface used to load a [[Dataset]] from external storage systems (e.g. file systems,
...
class DataFrameReader private[sql](sparkSession: SparkSession) extends Logging {

...

def load(): DataFrame = {
 load(Seq.empty: _*) // force invocation of `load(...varargs...)`
}
...
def load(paths: String*): DataFrame = {
 val proxyuser = extraOptions.get("user")
 if (proxyuser.isDefined) KerberosUser.setProxyUser(proxyuser.get)
 val dataSource = KerberosFunction.executeSecure(proxyuser,
     DataSource.apply,
     sparkSession,
     source,
     paths, userSpecifiedSchema, Seq.empty, None, extraOptions.toMap)

 val baseRelation = KerberosFunction.executeSecure(proxyuser, dataSource.resolveRelation, false)

 KerberosFunction.executeSecure(proxyuser, sparkSession.baseRelationToDataFrame, baseRelation)
}
```

Method for load data from sources without path

get user from dataset options

obtaining baseRelation from datasource

Driver Side

```
* Interface used to write a [[Dataset]] from external storage systems (e.g. file systems,
...
class DataFrameWriter[T] private[sql](ds: Dataset[T]) {

...

/**
* Saves the content of the [[DataFrame]] as the specified table.
...
def save(): Unit = {
 assertNotBucketed("save")
...

 val maybeUser = extraOptions.get("user")
 def innerWrite(modeData: (SaveMode, DataFrame)): Unit = {
   val (mode, data) = modeData
   dataSource.write(mode, data)
 }
 if (maybeUser.isDefined) KerberosUser.setProxyUser(maybeUser.get)

 KerberosFunction.executeSecure(maybeUser, innerWrite, (mode, df))
}
```

Method for save data in external sources

get user from dataset options

Wrapping save execution

Driver Side

```
class DAGScheduler(...){
...
...
KerberosUser.getMaybeUser match {
 case Some(user) => properties.setProperty("user", user)
 case _ =>
}
...
val tasks: Seq[Task[_]] = try {
stage match {
 case stage: ShuffleMapStage =>
   partitionsToCompute.map { id =>
    ...
     new ShuffleMapTask(stage.id, stage.latestInfo.attemptId,
       taskBinary, part, locs, stage.latestInfo.taskMetrics, properties)
    ...
     new ResultTask(stage.id, stage.latestInfo.attemptId,
       taskBinary, part, locs, id, properties, stage.latestInfo.taskMetrics)

   }
...
```

Executor Side

```
private[spark] abstract class Task[T](
   val stageId: Int,
   val stageAttemptId: Int,
   val partitionId: Int,
   // The default value is only used in tests.
   val metrics: TaskMetrics = TaskMetrics.registered,
   @transient var localProperties: Properties = new Properties) extends Serializable {

...
final def run(
...

try {
 val proxyUser =
   Option(Executor.taskDeserializationProps.get().getProperty("user"))
 KerberosFunction.executeSecure(proxyUser, runTask, context)
} catch {
...

def runTask(context: TaskContext): T
```

properties load in Driver side

get proxy user and wrapped execution

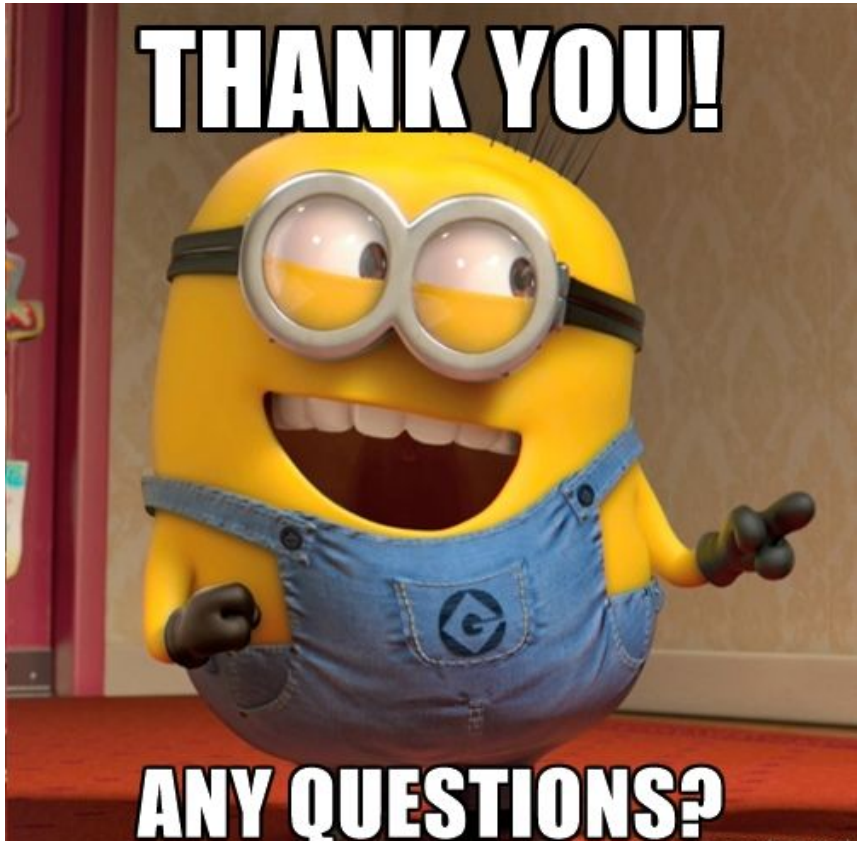
method implemented by Task subclasses

Demo time

Next Steps

- Merge this code in Apache Spark (SPARK-16788)
- Pluggable authorization
- Pluggable secret management (why always use Hadoop delegation tokens?)
- Distributed cache.
- ...

Q & A



WE ARE HIRING

people@stratio.com


@StratioBD


STRATIO
BIG DATA
A CHILD'S PLAY