Spark 101 for Scala People

Murray Todd Williams (murraytodd@gmail.com)

11 June 2020 - Austin Scala Enthusiasts Meetup

http://www.murraywilliams.com/2020/06/spark-101-for-scala-users/

about me

- Working for Accenture for 12 years
- First exposure to Scala via Spark about 4 years ago
- Currently wrestling with learning Cats and understanding Applicatives—far from an expert, but a passionate life-long learner…

Agenda

- How easy it is to start playing with Spark (hands-on demo)
- Getting started (environments, prototyping, setting up SBT)
- What's going on under the hood
- Spark RDDs vs DataFrames (and Datasets) i.e. Spark vs Spark SQL
- Partitioning

Getting Started

- Choose Cloudera vs Hortonworks flavor
- Choose cloud environment
- Provision and install
- Get data into environment
- Access Spark
- …

Getting Started

- ~~Choose Cloudera vs Hortonworks flavor~~
- ~~Choose cloud environment~~
- ~~Provision and install~~
- ~~Get data into environment~~
- ~~Access Spark~~

Run locally without Hadoop

Four Easy Approaches

Download and run (Mac or Linux)
Unpack tgz
cd bin
./spark-shell

Docker Spark Image

```
docker run --name spark -v $HOME/spark/data:/root -p 4040:4040 -it mesosphere/spark bin/spark-shell
```

Docker Zeppelin Image

```
docker run --name zeppelin -p 8080:8080 -p 4040:4040 -v $HOME/spark/data:/data -v \
       $HOME/spark/logs:/logs -v $HOME/spark/notebook:/notebook -e ZEPPELIN_NOTEBOOK_DIR='/notebook' \
       -e ZEPPELIN_LOG_DIR='/logs' -e ZEPPELIN_INT_JAVA_OPTS="-Dspark.driver.memory=4G" \
       -e ZEPPELIN_INTP_MEM="-Xmx4g" -d apache/zeppelin:0.9.0 /zeppelin/bin/zeppelin.sh
```

Dependencies.scala :

```
import sbt._

object Dependencies {

  val sparkVersion = "2.4.6"

  lazy val scalaTest = "org.scalatest" %% "scalatest" % "3.0.8"
  lazy val sparkCore = "org.apache.spark" %% "spark-core" % sparkVersion
  lazy val sparkSQL = "org.apache.spark" %% "spark-sql" % sparkVersion
}
```

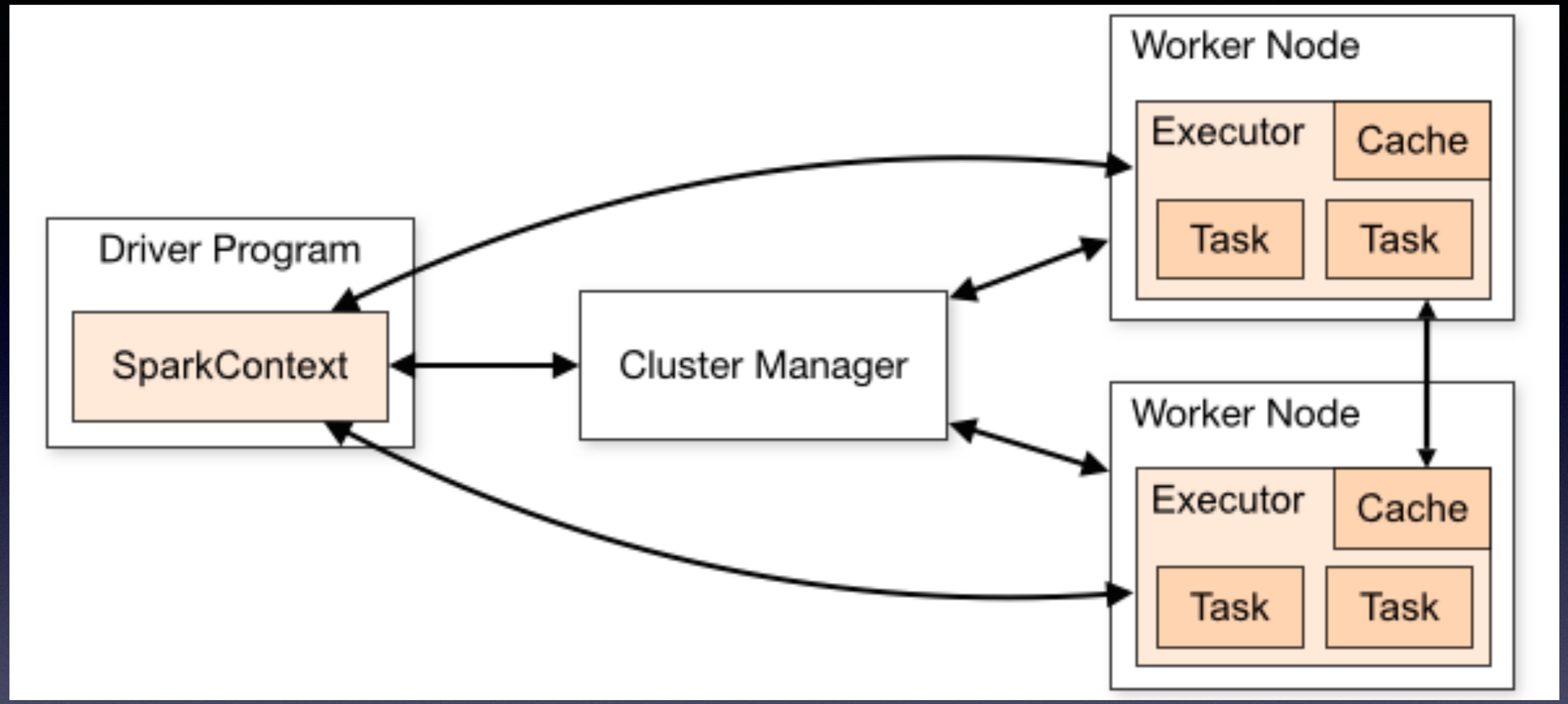
build.sbt :

```
import Dependencies._

ThisBuild / scalaVersion     := "2.12.11"
ThisBuild / version          := "0.1.0-SNAPSHOT"
ThisBuild / organization     := "com.example"
ThisBuild / organizationName := "Scala Meetup"

lazy val root = (project in file("."))
  .settings(
    name := "spark-base",
    libraryDependencies ++= Seq( scalaTest % Test, sparkCore, sparkSQL)
  )
```

sbt console :

```
import org.apache.spark.sql.SparkSession

val spark = SparkSession.builder().master(“local").getOrCreate
val sc = spark.SparkContext
```

DEMO



Under the hood

Designed to run massively parallel tasks

- You’re not creating objects (e.g. lists, maps) but rather assembling strategies broken into stages
- Everything is incredibly lazy
- If you’re working with RDDs (more on that in a moment), it feels like you’re just manipulating Scala structures.

Spark vs Spark SQL

Original Spark	**RDDs** Resilient Distributed Datasets	• Original way of working • Resembles a Scala collection
Spark SQL (v1.5+) Became the preferred paradigm in v2.0+	**DataFrames / Datasets**	• SparkSQL (typed) • Resembles a Scala collection
	Spark Tables & SQL	• Untyped (SQL in Text) • Spark managed tables

- SparkSQL can be much faster due to the fact that SQL operations can be optimized
- It’s pretty trivial to jump between RDDs and (SparkSQL) DataFrames and back*

HANDS-ON DEMO

Partitioning

- In order to do parallel work, the data needs to be split into partitions
- Spark/Hadoop data files will often have scores of partitions
- Partitioning is important for joins so each worker/executer can guarantee all records for a particular field (e.g. customer) are local to a single partition file
- If you run out of memory, make more (smaller) partitions to break up the problem into tiny pieces