WIZELINE ACADEMY

SURVIVAL KIT

BIG DATA ENGINEERING WITH SPARK

WIZELINE ACADEMY

Requirements

We are very excited to have you around. Here are some tips for your information:

Prerequisites:

- Gmail: It is paramount that you have a Gmail account and that you share it with us at your earliest convenience, otherwise you will not be able to access our cloud environment. If you do not have an account, please create one here.

What we will be using:

- Technologies, frameworks, and languages you will use:
  - Spark: the available languages for this course are:
    - PySpark (Python 2)
    - Scala
  - Zeppelin: A notebook technology.
  - YARN/HDFS: This is the underlying system where information can be stored (you will not be handling this directly as google cloud handles it up for you.)
- Google Cloud: Google's web services. Feel free to go through any tutorial as preparation. See how to get started with Google Platform here.
  - Google storage (Buckets): You have access to several available buckets:
    - **input** buckets for you to read data,
    - **output** buckets for storing your assignment results.
  - Google DataProc: this is a managed cluster service where you will be running your jobs and monitoring their performance.
- Interacting with Google Cloud:
  - Google console: This is the main page, for web interaction you can go from here to see the storage and the clusters easily. With the console you will also be submitting jobs to the dataproc clusters.
  - Google cloud shell: This is a shell environment to manage your resources on the cloud. It has the extra benefit that is browser dependent so you won't have to install anything.
  - GCloud SDK: In case you hate GUIs and feel like using command line/terminal all the time you may install and use the google sdk. Just note that we will not help you with the installation.

Academy Logistics

During this four-week long course, you will be interacting with the Data Team and your classmates constantly. Here is some important information:

- **Communications:**
  - All communications will be done via Slack. You will receive an invitation before Monday to setup your account:
    - Enter the following link for the Academy slack: http://bit.ly/slackacademy
    - Once there, join the Data-Eng-Academy channel
  - You have a mentor assigned, they will be available to help you troubleshoot connection and other technical issues during your work. You should reach for you mentor before going to your teacher.
  - Teachers will also be in the channel, they are there to help you with concept questions and tips in case you want to go in depth.
- **Gmail account:**
  - The email account you provided has been enabled in Google Cloud.
  - Before class you will be able to enter the cloud console with your account and look at everything.
  - *Don't update or execute anything until the class.*
- **User name:**
  - The email you provided is your user name. If you have special characters or more than 20 characters in your email, we deleted those for usability purposes. For example: the user name for **sutdent_example99@gmail.com** is **studentexample99**
  - This user name will be used to check your exercises, homeworks and resources in Google Cloud.
  - If your student number is **studentexample99**, you can upload and see the contents in **`de-training-output-bucket-studentexample99`** and **`de-training-output-bucket-studentexample99`**; and you will connect to the Zeppelin notebook in the **`de-training-studentexample99-m`** instance.

  Note:`<user-name>` is a integer ID that will identify the resources assigned to you for the duration of the course.

Opening the Zeppelin Notebooks

During each session of the course, there are exercises to be completed. All exercises are mounted on Zeppelin, as stated earlier. Follow these instructions to connect to your Zeppelin notebook:

1) Go to the projects' console.
2) On the right side of the Google Cloud Platform toolbar, Start the cloud shell by selecting the **Start Shell** button.

A Shell window appears.

3) In the new shell window, enter the following command, replacing `<user-name>` with the ID you were provided for the course:

```
gcloud compute ssh --project data-castle-bravo --zone us-central1-a --ssh-flag="-L
            8080:localhost:8080" de-training-<user-name>-m
```

Note: The first time you connect, the tool creates the directory `[/home/user/.ssh]`. When prompted to continue, accept by entering **`(Y)`**.

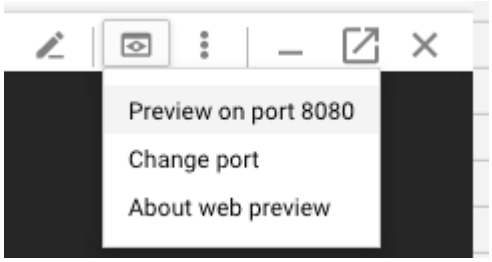
```
This tool needs to create the directory [/home/user/.ssh] before being able to
generate SSH keys.
Do you want to continue (Y/n)?
```

The following message appears:

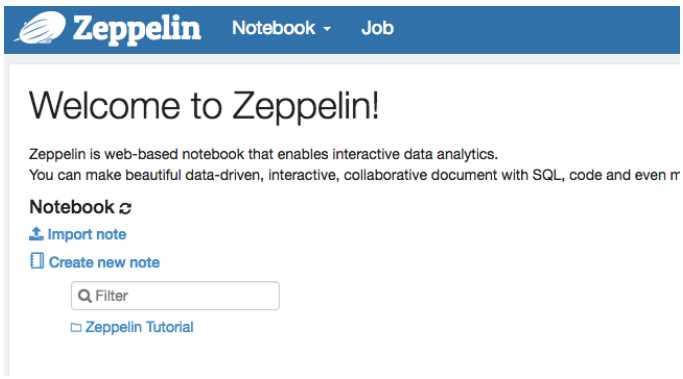
```
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
```

You can leave it blank or create a passphrase at your convenience.

4) Click the **Web Preview** icon, select **Preview on port 8080** from the list.



5) Once in zeppelin, select **Import note**.



6) Select **Add from URL**,
7) Upload the raw link from the github repository in the `de-academy-1` repository corresponding to the session.

8) Click the recently imported notebook to open it.

9) Have fun!

General Assignment Instructions

After each session of the course, there is an assignment for you to complete at home. All assignments follow a similar delivery format, which we describe in this document. Specific details for each assignment will be given to you in the last section of each presentation.

If anything seems unclear, contact us at wizeline-de-training@googlegroups.com (our general Q&A forum/mailing list) or reach out to the course instructors/mentors during office hours.

Submission Attempts and Deadlines

For each assignment, you can make as many submission attempts of your solution as you want without penalization. We are grading only the last attempt.

The deadline for each weekly submission is on Sunday at 11:59 pm. **If you can't deliver your solution on time for any reason, please reach out to the instructors.**

Self-Contained Applications

During the course sessions we will be using Zeppelin notebooks and you can complete the exercises in these same notebooks as well.

However, for all assignments, you have to create a self-contained application. To do this, follow the instructions from the official Spark website here. The following is an example of what a self-contained application looks like in Scala/Python:

```scala
/* SimpleApp.scala */
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SimpleApp {
  def main(args: Array[String]) {
    val logFile = "gs://..." // Should be some file on a Google bucket
    val conf = new SparkConf().setAppName("Simple Application")
    val sc = new SparkContext(conf)
    val logData = sc.textFile(logFile, 2).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))
  }
}
```

If you are using Scala, we recommend that you use the following content in your **`simple.sbt`** file (i.e. the file used to build your Spark program into a `JAR` file):

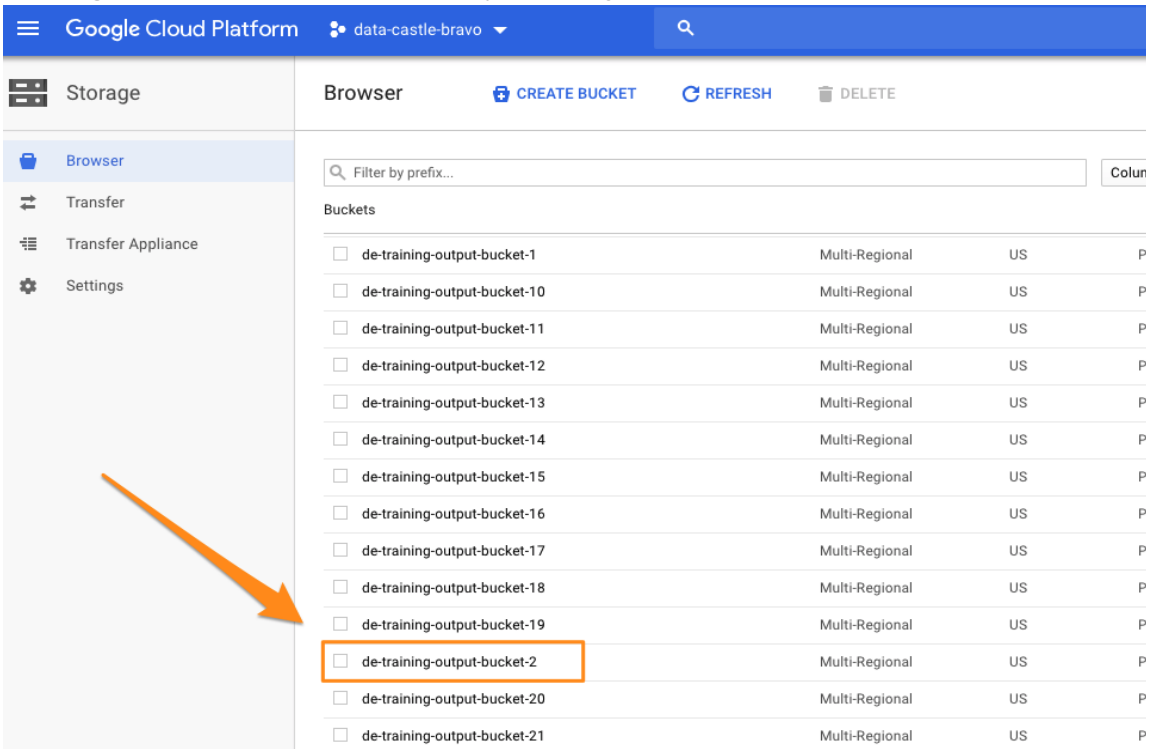
```
name := "<Assignment Name>"
version := "1.0"
scalaVersion := "2.11.0"
libraryDependencies += "org.apache.spark" %% "spark-core" % "2.2.0" % "provided"
```

Submission

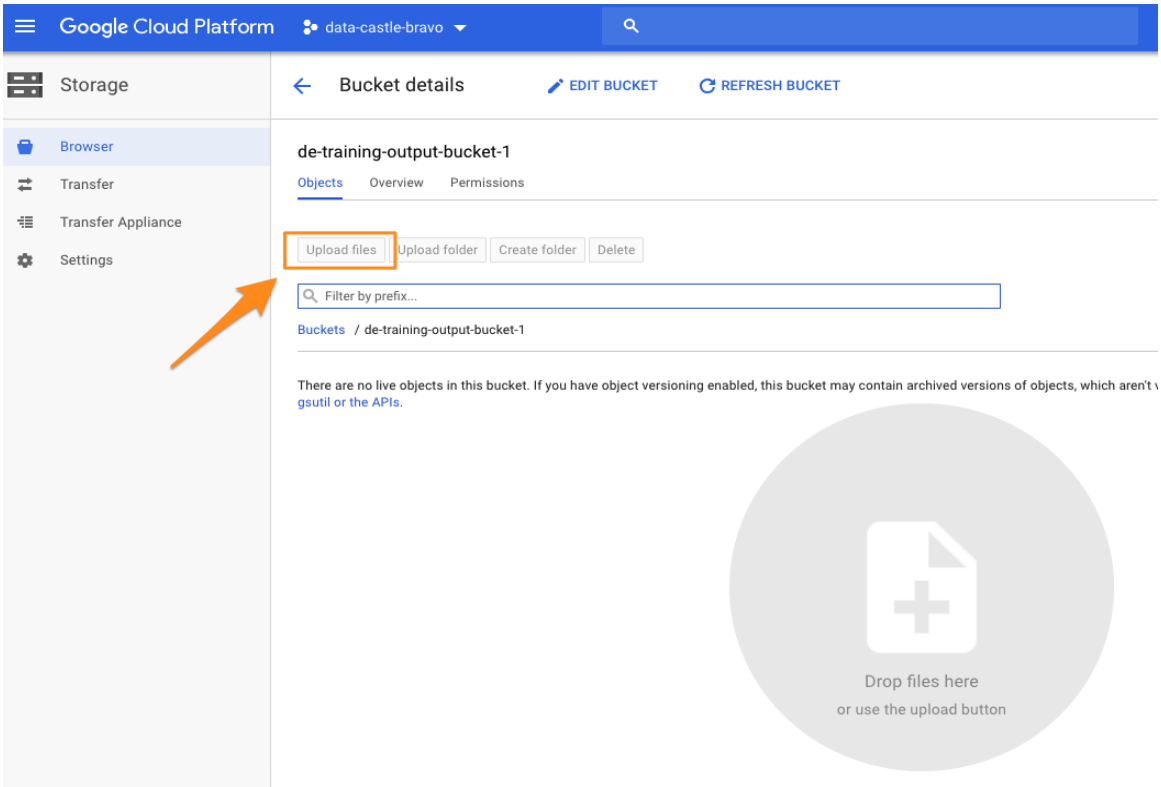
To submit your assignment as a Spark job:

- Create a `JAR` file if using **Scala**, or
- Create a `*.py` file if using **Python**

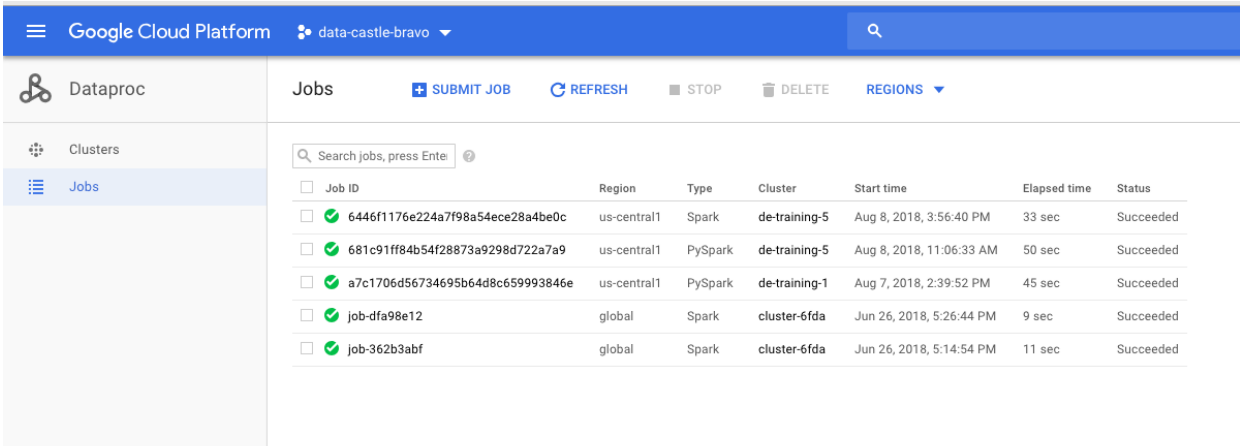
The next step it is to upload your JAR or .py file to your assigned bucket. Do this task going to "**Storage > Browser"** and then click your assigned bucket:



Then click on the **"Upload file"** button, or just Drag and Drop the file to your browser



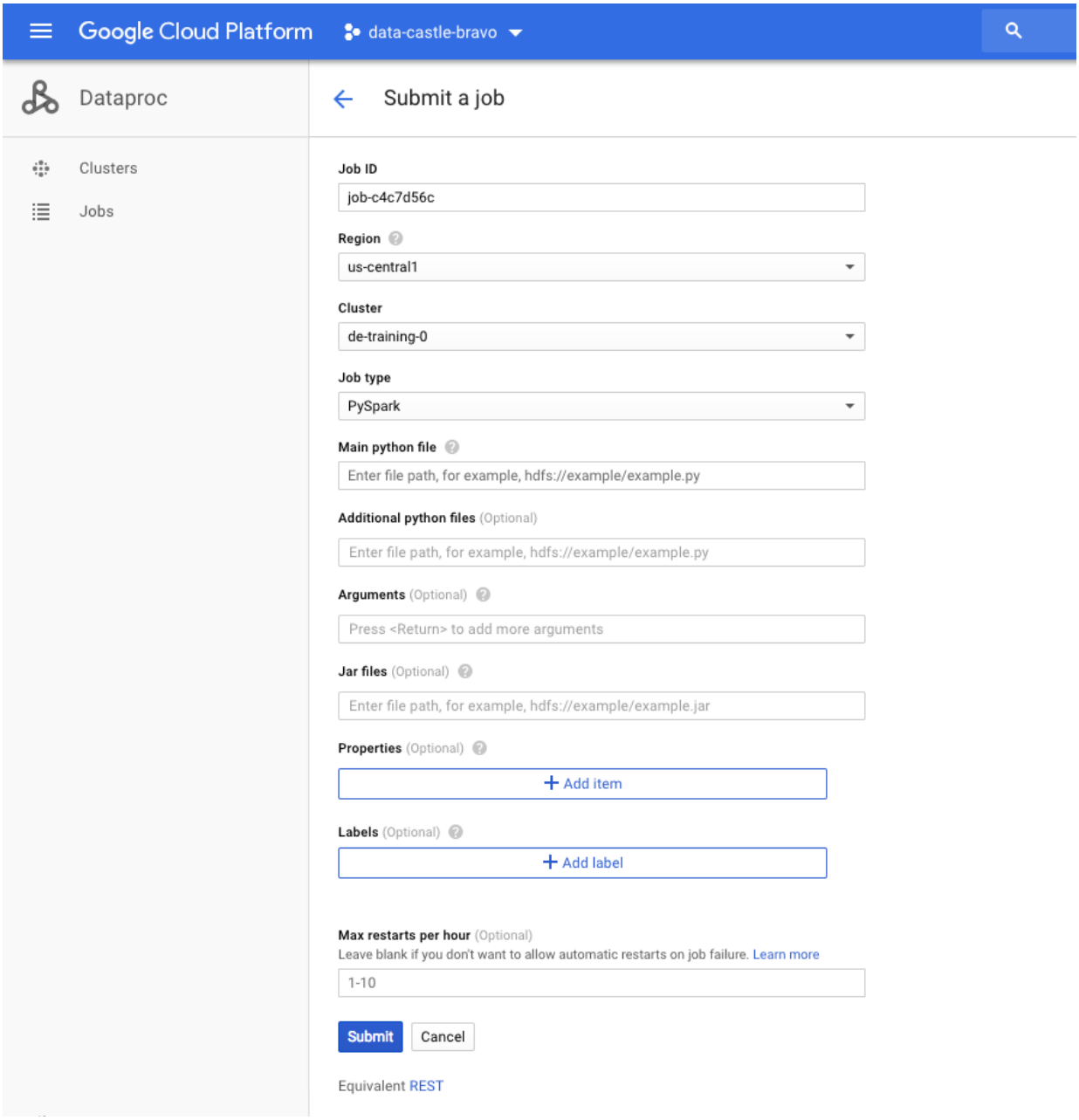
Then, submit the files to your assigned cluster by going to "**DataProc > Jobs**" in the web interface:



The interface shows quite a few options, but the crucial ones are **Region** (you won't see your cluster if you don't select the proper region, so be sure to check the "Clusters" tab to see in which region is your assigned cluster), **Cluster**, **Job Type**, and **Jar Files** (for Scala) **/ Main Python File** (for PySpark).

Notice that to specify your JAR or Python file, just put the name of your bucket and the file name such as: **`gs://de-training-output-bucket-<user-name>/SimpleAll.jar`**

Note: The interface includes built in help that describes each of the elements and actions.



The ability to include `zip` files is because Python treats them as directories from which to import modules and functions. (See this post for more details.)

Input Datasets

You will find all input datasets for your assignments in the **`gs://de-training-input-bucket`** bucket. For example, the "Alimazon" dataset is located in **`gs://de-training-input-bucket/alimazon`**

During your first session, you will have seen examples of how to read files from the buckets, but you can also refer to the **Input and Output Functions** notebook, provided at the beginning of the course in your “Survival Kit” package.

Output Locations

In your self-contained application, you will need to write the result to your assigned output bucket, which takes the form: **`gs://de-training-output-bucket-<user-name>/assignment-<session-id>.csv`** where **`<user-name>`** is a integer ID provided at the beginning of the course, and **`<session-id>`** is an increasing integer ID (e.g. for the first assignment it is **`0`**, for the second one it is **`1`**, etc.)

When writing the output, be sure to write it as a single file and in **`csv`** format. There is an example of **Writing an Output as CSV** in the **Input and Output Functions** notebook.

Glossary

Session 1, Week 1

- **Apache Nutch**: An open source web crawler that needed a compute engine to process the vast amounts of data that crawling the web implies. As a solution to that problem, *Hadoop* (the most commonly used solution for Big Data processing prior to Spark) was initially implemented in that project and later on moved to its own subproject.

- **Big Data:** Data sets that are so big (bigger than what you can fit in a single machine) and complex that traditional data-processing application software are inadequate to deal with them. The term is also used in the industry to encompass a broad range of technologies, methodologies and practices for the exploitation of such data sets.

- **Conviva**: A company founded by Ion Stoica, UC Berkeley professor at the time, and also co-founder of Databricks. As part of the video analytical tools built by the company, they supported ad-hoc querying and were using MySQL at that time, but they knew it wasn't good enough and started having talks with the founders of Spark to request a framework that supported such querying at big scale.

- **Cluster Manager**: An external service that provides abstractions for CPU, memory, storage, job scheduling, and other compute resources in a distributed cluster.

- **Databricks**: A company founded by some of the creators of Apache Spark that aims to help clients with cloud-based big data processing using Spark. Databricks grew out of the AMPLab project at University of California, Berkeley that was involved in making Apache Spark.

- **Dataframes**: They are a distributed collection of data organized into named columns. It is conceptually equivalent to a table in a relational database, but with richer optimizations under the hood. They build upon *RDDs* (the core underlying data structure in Spark, providing a higher-level data structure for programmers to work with) providing a higher-level API, with operations that are very familiar for those acquainted with SQL or Hive.

- **Datasets**: They are a type-safe version of Spark's structured API for Java and Scala. They are an abstraction that builds upon *RDDs* (the core underlying data structure in Spark, providing a higher-level data structure for programmers to work with).

- **Driver Program**: An application that creates a Spark Context for executing one or more jobs in the *Spark Cluster*. It is the process running the `main()` function of your application, too.

- **Executor/Worker:** A process in charge of running individual tasks in a given Spark job. They are launched at the beginning of a Spark application and typically run for the entire lifetime of an application.

- **Functional Programming:** See Functional Programming Intro

- **Hadoop:** An open-source framework for storing data and running applications on clusters of commodity hardware. It was the first framework to gain notoriety for its ability to run Big Data computations reliably and relatively efficiently without the need for specialized hardware. Prior to Spark's creation, it was the most commonly used solution for Big Data processing.

- **Kubernetes**: An open-source container-orchestration system for automating deployment, scaling and management of containerized applications. It was originally designed by Google and is now maintained by the Cloud Native Computing Foundation. As of version 2.3, Spark can run on clusters managed by Kubernetes. This feature makes use of the native Kubernetes scheduler that has been added to Spark.

- **Low-Level Computing Model**: This refers to programming languages or general computing models that require a lot of low-level details which are not necessarily relevant to the core problem being solved. For example, C can be considered a low-level programming language compared to Python, since it forces its programmers to think of many low-level details which may not be essential to the problem at hand (e.g. memory allocation, implementation of basic data structures, etc.

- **Higher-Level Computing Model**: This refers to programming languages or general computing models that tend to abstract away low-level details and help the programmer focus on higher-level concerns. Examples of languages that can be considered in this category are Python and Scala (as opposed to languages like C and C++).

- **Mesos**: A distributed systems kernel. It abstracts away CPU, memory, storage, and other compute resources away from machines (physical or virtual, enabling fault-tolerant and elastic distributed systems to easily be built and run effectively.

- **RDDs**: **R**esilient **D**istributed **D**atasets are an immutable, distributed collection of elements of your data, partitioned across nodes in your cluster that can be operated in parallel with a low-level API. The term *resilient* refers to the fact that they can recover from node failures in your cluster, preventing data loss.

- **Spark**: An open-source unified analytics engine for Big Data built in Scala. It's known for its speed, ease of use, generality and integration with a myriad tools in the Big Data ecosystem.

- **Spark Application**: A user program built on Spark. It consists of a *Driver* program and *Executors* on the cluster.

- **Spark Context**: An object in a Spark application whose responsibilities include setting up internal services and establishing a connection to the Spark execution environment. It acts as the master of your Spark application.

- **Spark Job**: A parallel computation consisting of multiple tasks that get spawned in response to a Spark action (e.g. `save`, `collect`); you'll see the term used in the *Driver's* logs.

- **Spark Streaming**: One of the components of Spark that allows for Big Data processing in real-time, using special data structures called *DStreams*.

- **Spark Graphx**: One of the components of Spark that focuses on Big Data processing of graphs, including the implementation of well-known algorithms like PageRank, the computation of strongly connected components, etc.

- **Spark ML/MLlib**: One of the components of Spark that focuses on Big Data processing of some of the most common machine learning algorithms, including implementations of classifiers, regressors, topic modeling, etc.

- **Task**: A unit of work that will be sent to one *Executor*.

- **Zeppelin Notebooks**: Web-based notebook that enables data-driven, interactive data analytics and collaborative documents with SQL, Scala and more. They support a variety of interpreters, including Spark and PySpark.

Session 2, Week 1

- **Fault tolerance**: The property that enables a system to continue operating properly in the event of failure (or one or more faults within) of some of its components. There are varying levels of fault tolerance that can be accomplished; the ones we will be focusing on this session will be spark's application fault tolerance for distributed computing, and distributed storage fault tolerance for the computer devices.

- **Distributed Storage:** A distributed data store is a computer network where information is stored on more than one node. Spark does not include distributed storage, it is an abstraction that can interact with the different ones already existent for big data such as Amazon S3, HDFS. (Note that we are referring to the baseline file systems that are available, distributed databases (SQL and NoSQL are themselves built to exploit said distributed).

- **File system:** Controls how data is stored and retrieved. A file system is a necessary part for the computation, and as long as you have a hard disk drive, any network with information to store, a local system or a distributed system, you will have a file system on varying levels of abstraction. Hence you may or may not need to deal with it depending on how you use the system.

- **Node**: In this context, we are referring to a computer within a computer network. It can be a master node, a worker node or a backup node.

- **Computer network:** Is a digital telecommunications network which allows nodes to share resources. In computer networks, computing devices exchange data with each other using connections (data links) between nodes.

- **Data / data set / file:** In this context we are referring to data as a file, table, or a document that contains information you are interested in processing.

- **Data chunk:** Is one piece of the data set that has been stored in a distributed storage. The logic for the chunking or partitioning depends on the distributed storage being used, typically it is 64 MB for Hadoop. Click the following links to see what this means on distributed storage, on spark.

- **Data partition:** As the data chunk, is what the data set is being split on for distributed storage and processing. Spark partitioning tends to follow the predefined partition (if reading from the distributed storage), or have its own rules for distribution after a grouping activity happens.

- **Master Node:** Computational unit (Program or designated node), that contains the information and logical daemos for handling the distributed task. For HDFS the master is called Name Node, in Spark.

- **HDFS:** It is a distributed file system that handles large data sets running on commodity hardware. It is one of the two main descriptors of hadoop, and has ever since been adopted for many distributed frameworks as the file system foundations

- **YARN:** The framework that splits the includes a global resource manager (who ultimately handles resources among applications), and the application master (who negotiates the resources for each application with the resource manager and the name node).

- **Amazon S3:** A cloud distributed solution for storage offered by amazon. It started as the main rival of hadoop (developed by yahoo! members). It is a similarly distributed storage system on the cloud that, as google cloud, offers different levels of availability and scalability. These can be already managed, reducing the burden that existed with hadoop hdfs clusters.

- **Embarrassingly parallel task** (or program): When the parallelization of the work is trivial, thus requiring little to no effort to perform this. These are the most seeked units for parallelization, and the aim is to re-describe any task to include as many embarrassingly parallel steps as possible.

- **Matrix**: Collection of numbers ordered in columns and rows (table of values), where their position as well as the number contained are relevant to the analysis.

- **Matrix multiplication**: If **A** is an $n \times m$ matrix and **B** is an $m \times p$ matrix, their matrix product **AB** is an $n \times p$ matrix, in which the m entries across a row of **A** are multiplied with the m entries down a column of **B** and summed to produce an entry of **AB**. When two linear maps are represented by matrices, then the matrix product represents the composition of the two maps.

- **Block matrix**: A type of matrix optimized in spark, created with RDDs, that supports additions and multiplications by default.

- **Directional Acyclic Graph (DAG)**: In Spark, a job is associated with a chain of RDD dependencies organized in a direct acyclic graph (DAG).

- **Task scheduler**: Spark component that communicates with the DAG and the worker nodes. It is useful to pass information, and has the ability to re-execute a failed task, or communicate to the dag DAG the ultimate failure.

- **Cluster manager**: The daemon outside spark that handles the cluster, and balances it, depending on the way spark has been set up (Mesos, kubernetes, YARN, etc...) it will have different characteristics.