Hadoop Ecosystem

--

Carlos García Martínez

Goals

- To provide a very…
  ... **very very**...
  … (very much)...
  ... general idea.
- To have a contact with some tools
  - We will not dive into any of them. There is not enough time.
  - Further information is available in books and the internet.
  - These slides are based on the HDP and HDF Hortonworks tutorials, and the first ones of Kafka:
    - https://www.cloudera.com/tutorials/getting-started-with-hdp-sandbox.html
    - https://www.cloudera.com/tutorials/getting-started-with-cdf-sandbox.html
    - https://developer.confluent.io/learn-kafka/
    - Some modifications have been introduced

Hadoop tools we will use

- *Ambari*
- *Hive*
- *Zeppelin*
- *Nifi*
- *Kafka*

- Hortonworks Sandboxes HDP y HDF (**en AWS**)

Famous quotes

“There will be events for which I have no answer. You should go over these instructions, backtracking whenever necessary: 1) look for information (this requires some training), 2) read and try to understand the pieces of information (is this related to my problem? What are the main concepts and terms? Can I use these terms to improve my query?), and 3) adapt the solutions to your needs (perfect understanding is not always necessary)”

Famous quotes

“There will be events for which I have no answer. You should go over these instructions, backtracking whenever necessary: 1) look for information (this requires some training), 2) read and try to understand the pieces of information (is this related to my problem? What are the main concepts and terms? Can I use these terms to improve my query?), and 3) adapt the solutions to your needs (perfect understanding is not always necessary)”

– Carlos García Martínez. January, 2022

Famous quotes

“There will be events for which I have no answer. You should go over these instructions, backtracking whenever necessary: 1) look for information (this requires some training), 2) read and try to understand the pieces of information (is this related to my problem? What are the main concepts and terms? Can I use these terms to improve my query?), and 3) adapt the solutions to your needs (perfect understanding is not always necessary)”

– ***Carlos García Martínez. January, 2022***

“Patience”

Famous quotes

“There will be events for which I have no answer. You should go over these instructions, backtracking whenever necessary: 1) look for information (this requires some training), 2) read and try to understand the pieces of information (is this related to my problem? What are the main concepts and terms? Can I use these terms to improve my query?), and 3) adapt the solutions to your needs (perfect understanding is not always necessary)”

– ***Carlos García Martínez. January, 2022***

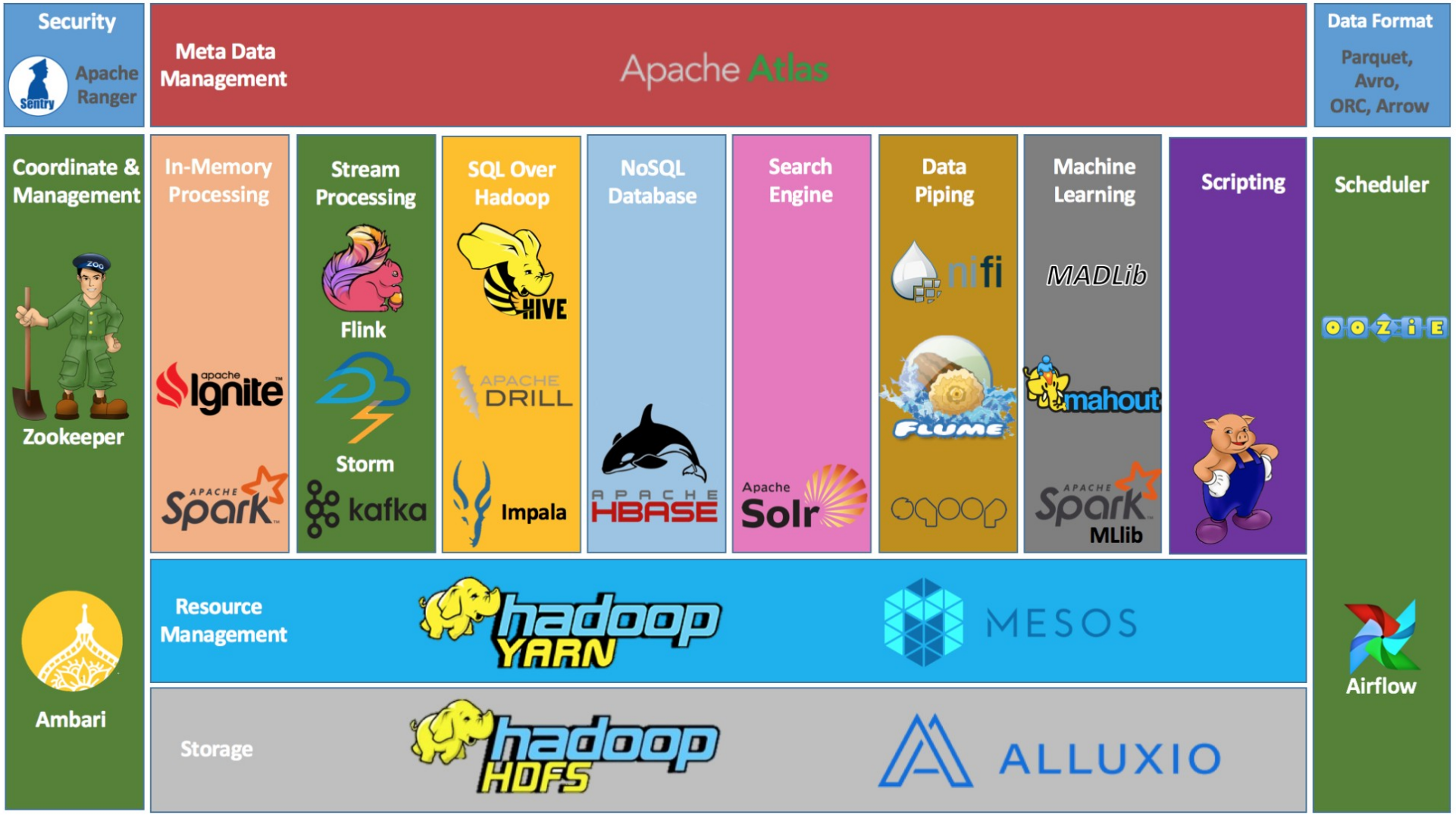
“Patience”

– ***Carlos García Martínez. January, 2022***

Hadoop Ecosystem

The Ecosystem

https://www.oreilly.com/library/view/apache-hive-essentials/9781788995092/e846ea02-6894-45c9-983a-03875076bb5b.xhtml

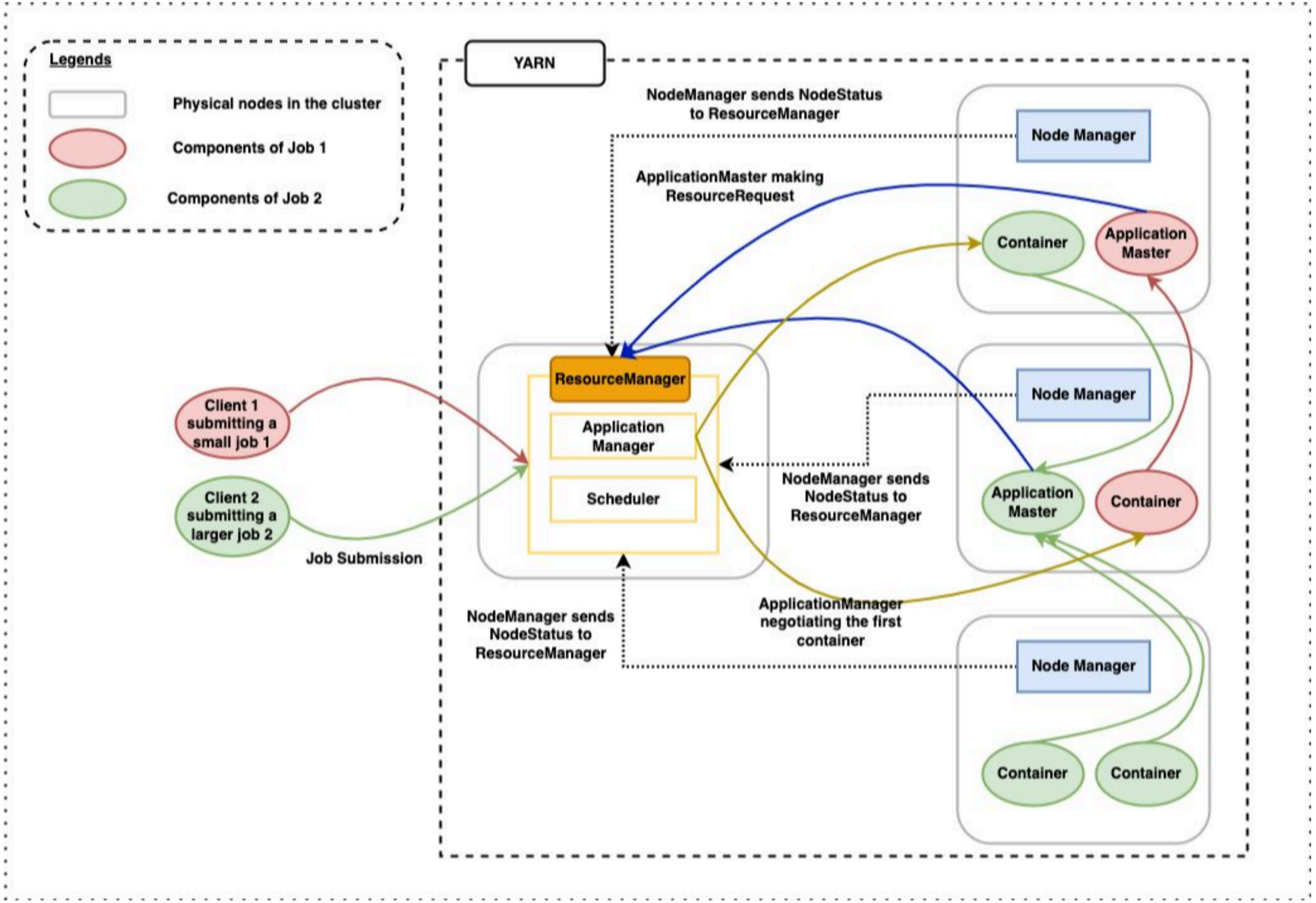


The ecosystem

https://www.cloudera.com/tutorials/getting-started-with-hdp-sandbox/1.html

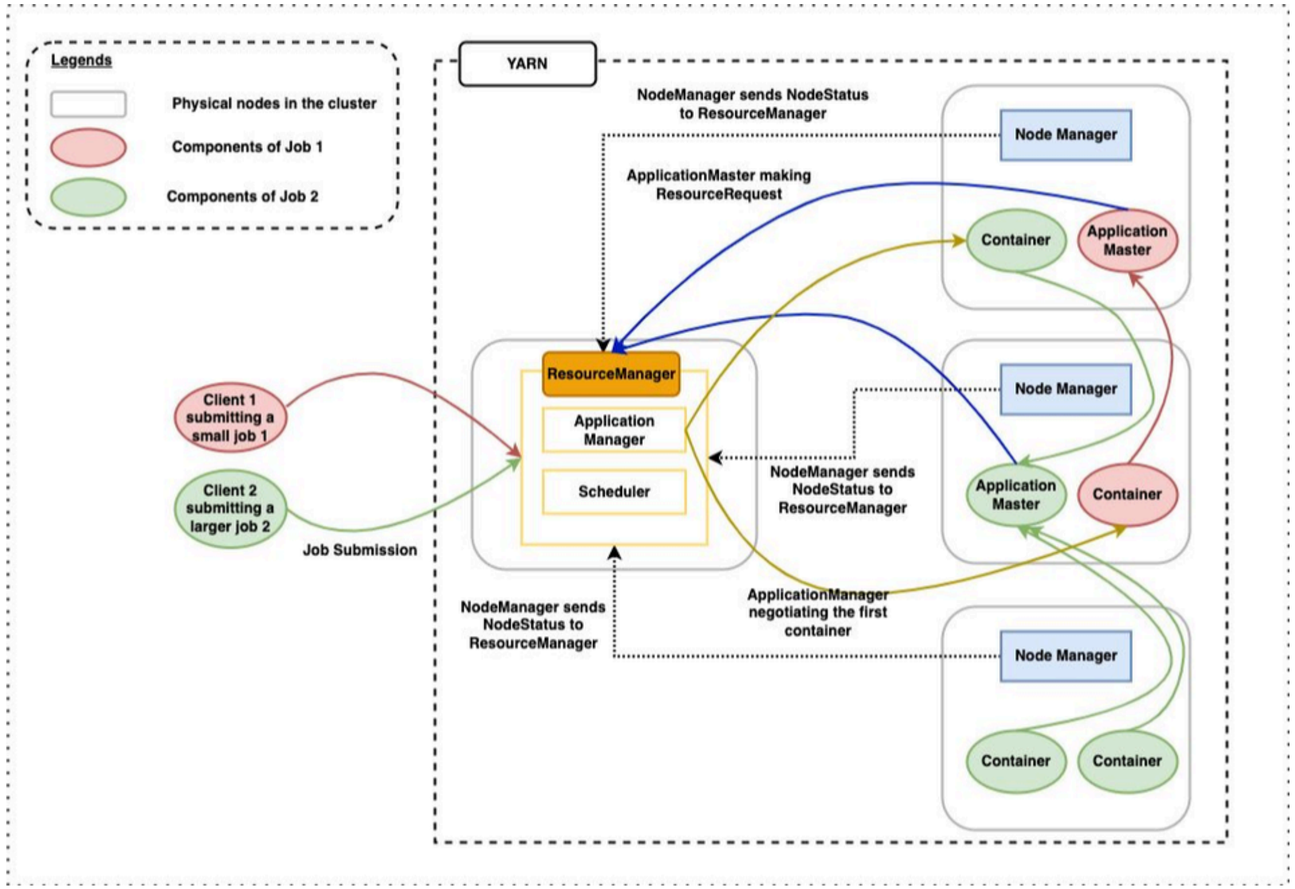
- Base Apache Hadoop:
  - **Hadoop Common** - contains libraries and utilities needed by other Hadoop modules.
  - Hadoop Distributed File System (**HDFS**) – a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster.
  - Hadoop **YARN** – a **resource-management** platform responsible for managing computing resources in clusters and using them for scheduling of users' applications.
  - Hadoop **MapReduce** – a programming model for large scale data processing.
- Data Management:
  - Apache Hadoop **YARN** – Part of the core Hadoop project, YARN is a next-generation framework for Hadoop data processing **extending MapReduce capabilities by supporting non-MapReduce workloads** associated with other programming models.
  - HDFS – Hadoop Distributed File System (HDFS) is a Java-based file system that provides scalable and reliable data storage that is designed to span large clusters of commodity servers.
- Data Access:
  - Apache **Hive** – Built on the MapReduce framework, Hive is a data warehouse that enables easy data summarization and ad-hoc queries via an **SQL**-like interface for large datasets stored in HDFS.
  - Apache **Pig** – A platform for processing and **analyzing large data sets**. Pig consists of a high-level language (Pig Latin) for expressing data analysis programs paired with the MapReduce framework for processing these programs.
  - **MapReduce** – MapReduce is a framework for writing applications that process large amounts of structured and unstructured data in parallel across a cluster of thousands of machines, in a reliable and fault-tolerant manner.
  - Apache **Spark** – Spark is ideal for **in-memory data processing**. It allows data scientists to implement fast, iterative algorithms for advanced analytics such as clustering and classification of datasets.
  - Apache **Storm** – Storm is a **distributed real-time computation** system for processing fast, large streams of data adding reliable real-time data processing capabilities to Apache Hadoop 2.x
  - Apache **HBase** – A column-oriented **NoSQL** data storage system that provides random real-time read/write access to big data for user applications.
  - Apache **Tez** – Tez **generalizes the MapReduce** paradigm to a more powerful framework for executing a complex DAG (directed acyclic graph) of tasks for near real-time big data processing.
  - Apache **Kafka** – Kafka is a fast and scalable **publish-subscribe messaging system** that is often used in place of traditional message brokers because of its higher throughput, replication, and fault tolerance.
  - Apache **HCatalog** – A table and metadata management service that provides a centralized way for data processing systems to **understand the structure and location of the data** stored within Apache Hadoop.
  - Apache **Slider** – A framework for deployment of long-running data access applications in Hadoop. **Slider leverages YARN's resource management capabilities** to deploy those applications, to manage their lifecycles and scale them up or down.
  - Apache **Solr** – Solr is the open source platform for searches of data stored in Hadoop. Solr enables powerful **full-text search** and near real-time indexing on many of the world's largest Internet sites.
  - Apache **Mahout** – Mahout provides scalable **machine learning** algorithms for Hadoop which aids with data science for clustering, classification and batch based collaborative filtering.
  - Apache **Accumulo** – Accumulo is a high performance **data storage and retrieval** system with cell-level access control. It is a scalable implementation of Google's Big Table design that works on top of Apache Hadoop and Apache ZooKeeper.
- Data Governance and Integration
  - **Workflow Management** – Workflow Manager allows you to easily **create and schedule workflows and monitor workflow jobs**. It is based on the Apache **Oozie** workflow engine that allows users to connect and automate the execution of big data processing tasks into a defined workflow.
  - Apache **Flume** – Flume allows you to efficiently aggregate and **move large amounts of log data** from many different sources to Hadoop.
  - Apache **Sqoop** – Sqoop is a tool that **speeds and eases movement of data in and out of Hadoop**. It provides a reliable parallel load for various, popular enterprise data sources.
- Security
  - Apache **Knox** – The Knox Gateway ("Knox") provides a **single point of authentication and access** for Apache Hadoop services in a cluster. The goal of the project is to simplify Hadoop security for users who access the cluster data and execute jobs, and for operators who control access to the cluster.
  - Apache **Ranger** – Apache Ranger delivers a **comprehensive approach to security for a Hadoop** cluster. It provides central security policy administration across the core enterprise security requirements of authorization, accounting and data protection.
- Operations:
  - Apache **Ambari** – An open source **installation lifecycle management**, administration and monitoring system for Apache Hadoop clusters.
  - Apache **Oozie** – Oozie Java Web application used to s**chedule Apache Hadoop jobs**. Oozie combines multiple jobs sequentially into one logical unit of work.
  - Apache **ZooKeeper** – A highly available system for **coordinating distributed processes**. Distributed applications use ZooKeeper to store and mediate updates to important configuration information.

YARN



https://aws.amazon.com/es/blogs/big-data/configure-hadoop-yarn-capacityscheduler-on-amazon-emr-on-amazon-ec2-for-multi-tenant-heterogeneous-workloads/

Flujo de trabajo en YARN



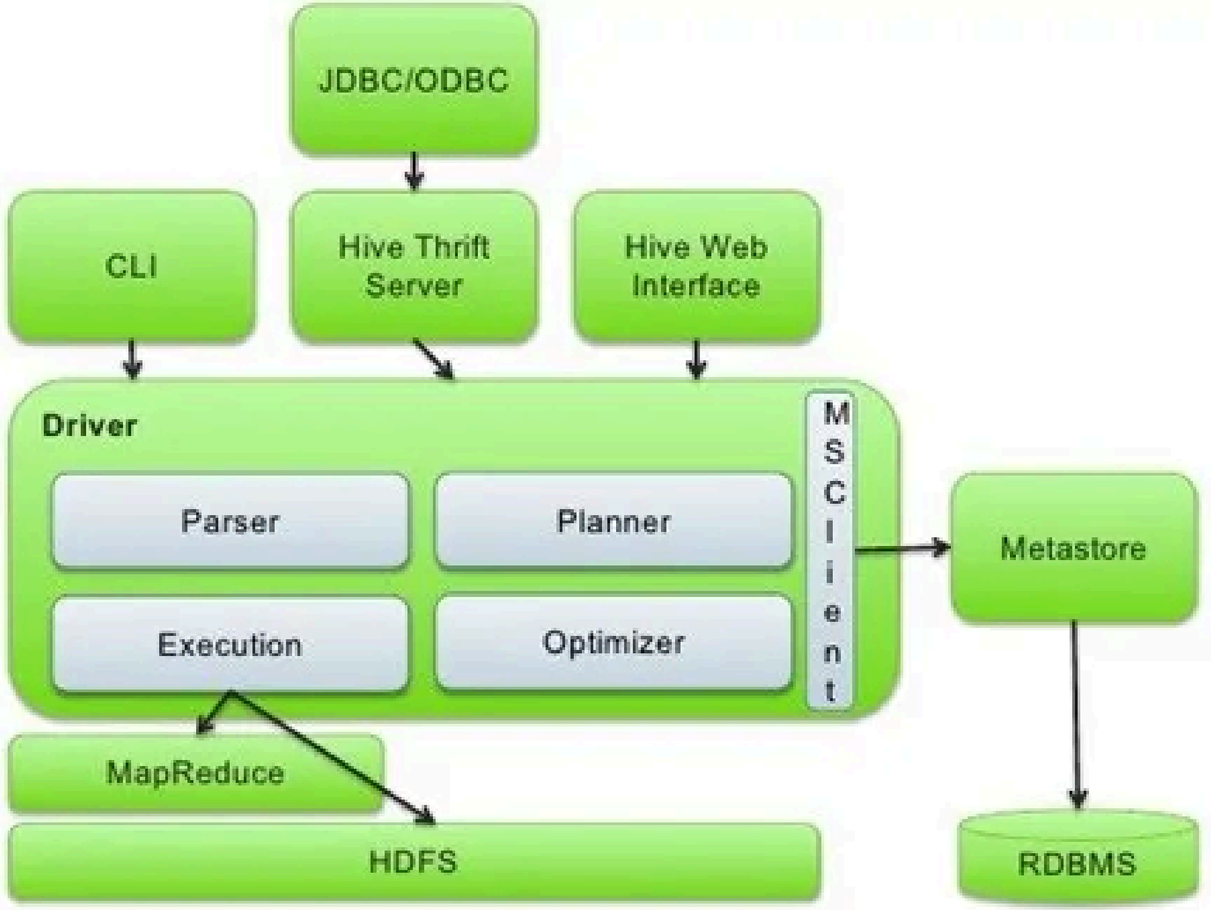
1. Un cliente envía su aplicación al ResourceManager.
2. El ApplicationsManager crea un contenedor inicial para ejecutar el ApplicationMaster.
3. El ApplicationMaster negocia recursos con el ResourceManager.
4. El ResourceManager asigna contenedores en los nodos adecuados.
5. El NodeManager supervisa los contenedores mientras ejecutan las tareas.
6. El ApplicationMaster coordina las tareas y devuelve los resultados al cliente.

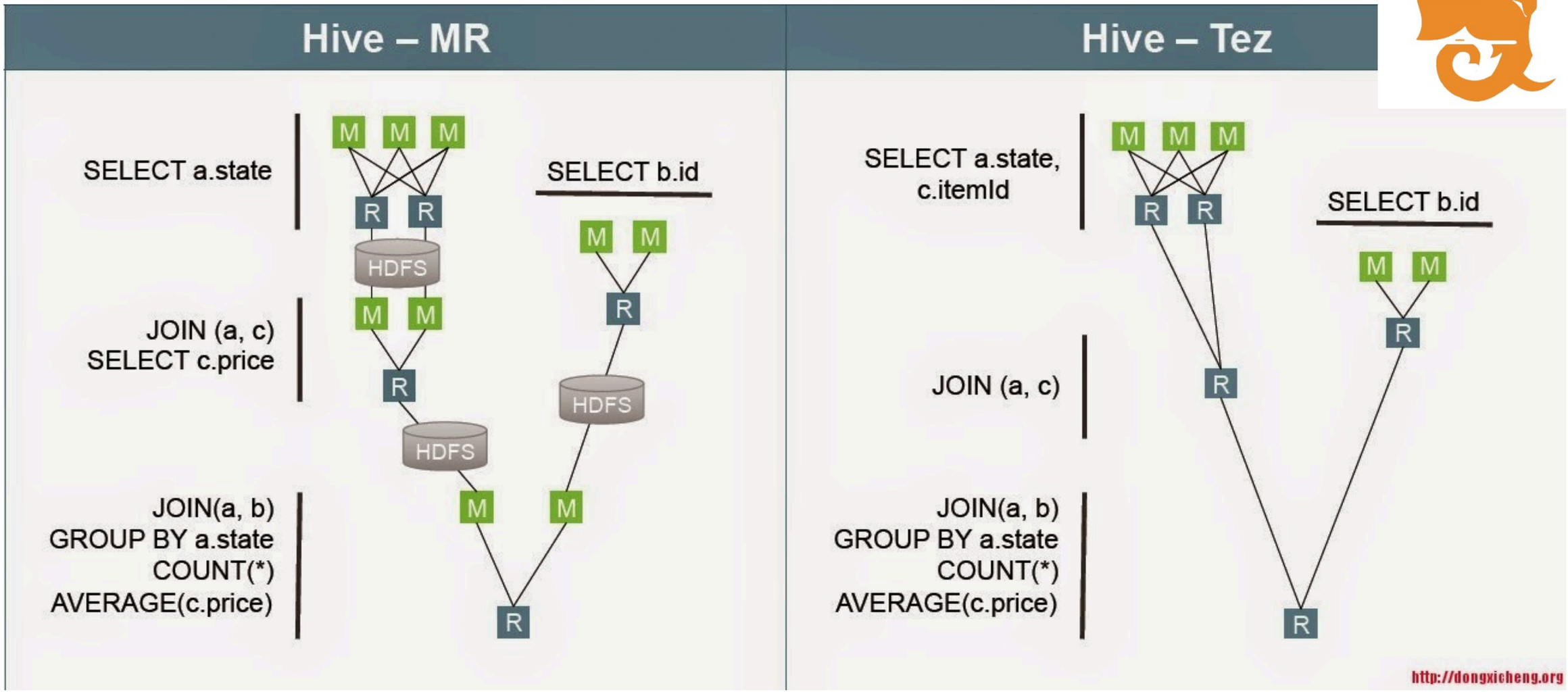
https://aws.amazon.com/es/blogs/big-data/configure-hadoop-yarn-capacityscheduler-on-amazon-emr-on-amazon-ec2-for-multi-tenant-heterogeneous-workloads/

Ventajas de YARN

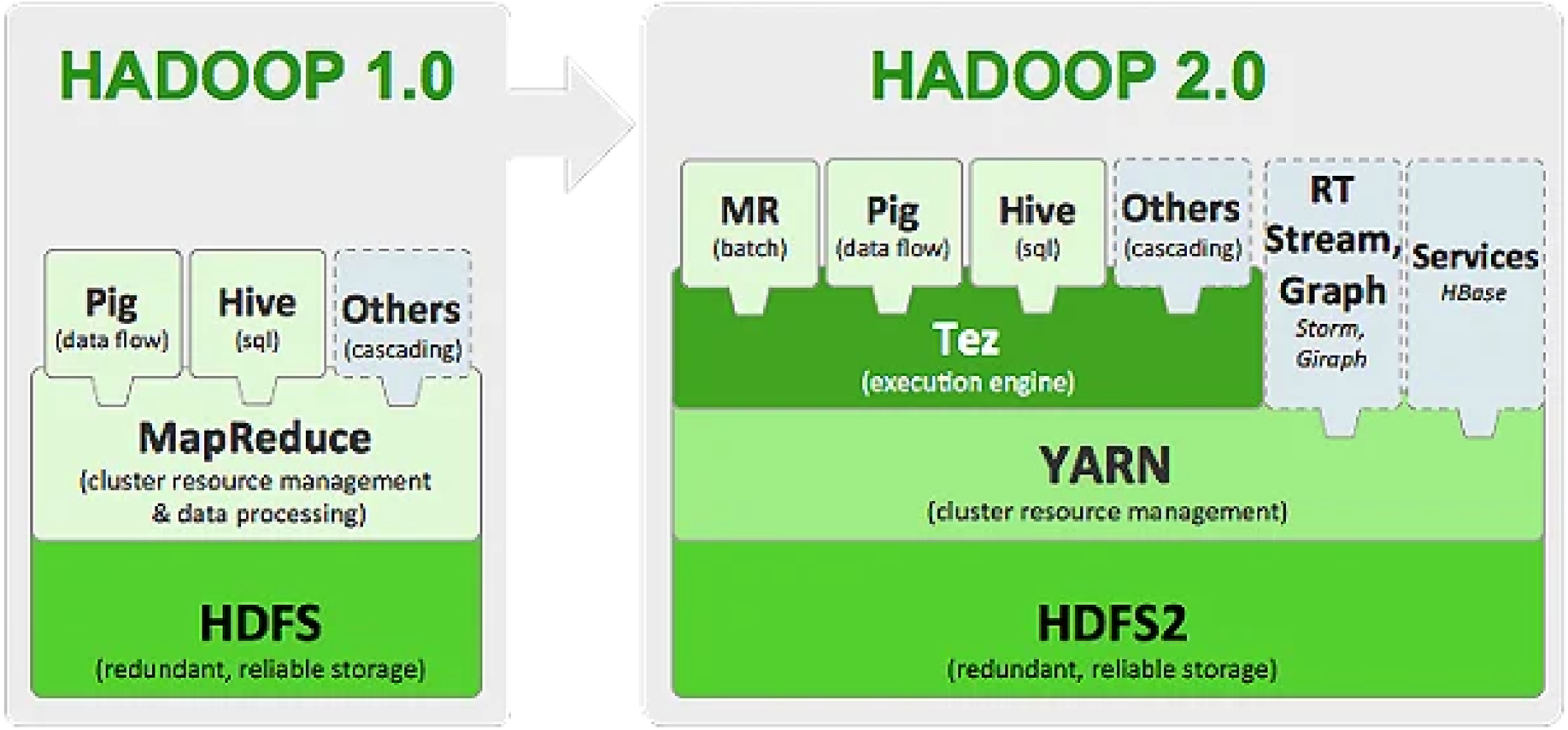
- Escalabilidad: Maneja miles de nodos y aplicaciones simultáneamente.
- Flexibilidad: Permite ejecutar múltiples tipos de frameworks y aplicaciones, no solo MapReduce.
- Utilización Eficiente de Recursos: Asigna recursos dinámicamente según las necesidades de las aplicaciones.
- Modularidad: La separación del manejo de recursos y las aplicaciones permite una arquitectura más limpia y extensible.

Apache Hive Architecture





https://ganesansenthilvel.blogspot.com/2014/10/apache-tez.html



https://www.infoq.com/articles/apache-tez-saha-murthy/

```sql
SELECT categoria, region, SUM(cantidad * precio) AS total_ventas
FROM ventas
WHERE region = 'Norte'
GROUP BY categoria, region;
```

1. Map task:
    a. Filtrar y transformar
    b. Genera pares clave-valor (categoría, región) para la siguiente fase.
2. Reduce: Se suman los resultados

```sql
SELECT categoria, region, SUM(cantidad * precio) AS total_ventas
FROM ventas
WHERE region = 'Norte'
GROUP BY categoria, region;
```

1. Map task:
    a. Filtrar y transformar
    b. Genera pares clave-valor (categoría, región) para la siguiente fase.
2. Reduce: Se suman los resultados

```sql
SELECT r.nombre_region, v.categoria, SUM(v.cantidad * v.precio) AS ingreso_total,
       SUM(v.cantidad) AS productos_vendidos
FROM ventas v
JOIN regiones r ON v.region_id = r.region_id
WHERE v.precio > 50
GROUP BY r.nombre_region, v.categoria
HAVING SUM(v.cantidad * v.precio) > 1000
ORDER BY ingreso_total DESC;
```

1. Map task:
    a. Filtrar y transformar en tablas separadas
    b. Genera pares clave-valor (region_id: {...})
2. Reduce: Se combinan los resultados (JOIN)
3. Map: Se agrupan (GROUP BY) ({region, cat}: {...})
4. Reduce: Se filtra (HAVING)
5. Se ordena (quizás fase reduce)

Laboratory sessions!

HDP

Server setup (what I tried) 1/4

- **Warning!!**
  - Shall you do this, you will incur personal expenses.
  - These steps are provided just in case you would like to try them on your own responsability. Ask the teacher.
- **THESE STEPS HAVE ALREADY BEEN APPLIED FOR YOU (Coutesy of Carlos García Martínez)**
- Launch AWS (North Virginia)
  - (If you have a computer with ~32GB RAM, perhaps you can install HDP and HDF there. You would not need AWS)
  - Amazon Linux
  - m4.x2large (8 cores, 32GB RAM)
  - Edit storage: 60GB (SSD)
  - Labels: nothing to do
  - Security groups: default. All outbound traffic should be allowed. **RESTRICT the inbound traffic to your IP**.
  - Launch
- **THIS IS IMPORTANT:**
  - You should stop your EC2 instance, otherwise the expenses will keep on growing.
  - You can keep your storage volume at lower prices. To stop incurring expenses, you have to remove it, but you will lost your progress

Server setup 2/4

- THESE STEPS HAVE ALREADY BEEN APPLIED FOR YOU (Coutesy of Carlos García Martínez)
- HDP Install (https://github.com/qge/hdp)
  - Connect to your EC2 instance with ssh (instrucctions in the connection section):
    - `ssh -i <key.pem file> ec2-user@<url_aws_server>`
  - `sudo yum intall -y git`
  - `sudo yum install -y nano`
  - `git clone https://github.com/qge/hdp.git`
  - `cd hdp`
  - `bash install_docker.sh` (This lasts a few minutes)
  - Logout and login
  - `docker info` to check that docker works
  - `cd hdp/HDP_3.0.1`
  - `sed -i 's/$proxyVersion"/$proxyVersion"; docker image tag hortonworks\/sandbox-proxy:1.0 hortonworks\/sandbox-proxy-hdp:1.0/g' docker-deploy-hdp30.sh`
  - `sed -i "s/sandbox-proxy/sandbox-proxy-hdp/g" assets/generate-proxy-deploy-script.sh`
  - `bash docker-deploy-hdp30.sh` (This lasts longer: ~10 minutes)
    - In two occasions, I had to select the docker.io registry

Server setup 3/4

- THESE STEPS HAVE ALREADY BEEN APPLIED FOR YOU (Coutesy of Carlos García Martínez)
- HDF install:
  - Download the docker bundle: https://www.cloudera.com/downloads/hortonworks-sandbox.html
  - Copy the file into your EC2 instance (you need the url of your instance):
    - `scp -i <key.pem file> HDF_3.1.1_deploy-scripts_180624d542a25.zip ec2-user@<url_aws_server>:~/`
  - Connect to your EC2 instance with ssh (instrucctions in the connection section):
    - `ssh -i <key.pem file> ec2-user@<url_aws_server>`
  - It is better to stop the HDP dockers. You have to locate their IDs: `docker stop <ids>`. `sandbox-proxy-hdp`, in particular
  - `mkdir hdf`
  - `mv HDF_3.1.1_deploy-scripts_180624d542a25.zip hdf; cd hdf`
  - `unzip HDF_3.1.1_deploy-scripts_180624d542a25.zip`
  - `sed -i 's/$proxyVersion"/$proxyVersion"; docker image tag hortonworks\/sandbox-proxy:1.0 hortonworks\/sandbox-proxy-hdf:1.0/g' docker-deploy-hdf311.sh`
  - `sed -i "s/sandbox-proxy/sandbox-proxy-hdf/g" assets/generate-proxy-deploy-script.sh`
  - `bash docker-deploy-hdf31.sh` (This lasts a few minutes)
    - It should be possible to have the HDP and HDF servers in parallel, but this would require to modify the proxy configurations. I have not done this.
    - Another possibility is to install both of them, but stop and start the corresponding proxys. For instance: `docker stop sandbox-proxy-hdp; docker start sandbox-proxy-hdf`
- Set the urls of the HDP and HDF servers in your EC2 instance
  - `sudo nano /etc/hosts`
    - Add this information: `127.0.0.1 sandbox-hdp.hortonworks.com sandbox-hdf.hortonworks.com`

Server setup 3

- THESE STEPS HAVE ALREADY BEEN APPLIED FOR YOU (Courtesy of Carlos García Martínez)
- Set the admin passwords
  - (https://www.cloudera.com/tutorials/learning-the-ropes-of-the-hdp-sandbox.html)
  - `ssh root@sandbox-hdp.hortonworks.com -p 2222`
    - Change password: hadoop → p1i2n3e4
  - `(sudo) ambari-admin-password-reset`
    - p1i2n3e4
  - logout
  - `ssh root@sandbox-hdf.hortonworks.com -p 2202`
    - Change passworkd: hadoop → p1i2n3e4
  - `(sudo) ambari-admin-password-reset`
    - p1i2n3e4

How to connec to HDP/HDF

- **You shall do this everytime you start your AWS EC2 instance (even for the course sessions)**
- Start docker
  - `ssh` the EC2 instance
  - `sudo systemctl restart docker`
  - `docker start sandbox-hdp sandbox-hdf sandbox-proxy-(hdp|hdf)`
- Optional: You can add the EC2 IP in your personal computer for the following address (be aware that this ip changes every time the EC2 instance is initiated)
  - `sudo nano /etc/hosts`
    - Add the following information:

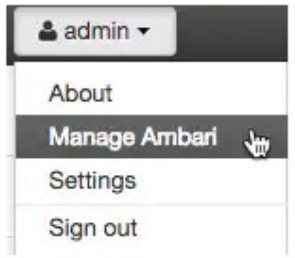
      ```
      <ip_aws_server> sandbox-hdp.hortonworks.com sandbox-hdf.hortonworks.com
      ```
  - Alternatively, you can write the EC2 url in your browser

Ambari (HDP)

- Open **Ambari** with admin credentials http://sandbox-hdp.hortonworks.com:8080/
- If you started the EC2 instance a few minutes ago, the services will be being deployed. You can check that at:
  - / Start All Services / sandbox-hdp.hortonworks.com
- In case any error happened, you could check the logs here
- You can restart them in any case:
  - Services ... / Start all
- You can set some services in Maintenance mode, or even remove them:
  - Click on the service / Actions / Turn On Maintenance Mode
  - This prevents the service to start
  - **Exercise**: Stop all the services. Set Kafka into maintenance. Start all the services. Call the teacher.

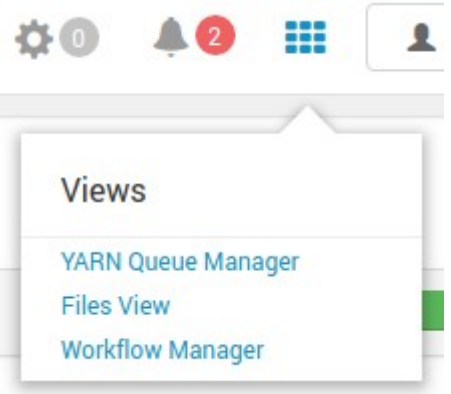
Ambari (HDP)

- Dashboard shows some information
- Select Manage Ambari
- Revise the available users at Users
  - You can edit the privileges.
  - **Exercise**: Make `raj_ops` ambari administrator and call the teacher



Ambari (HDP)

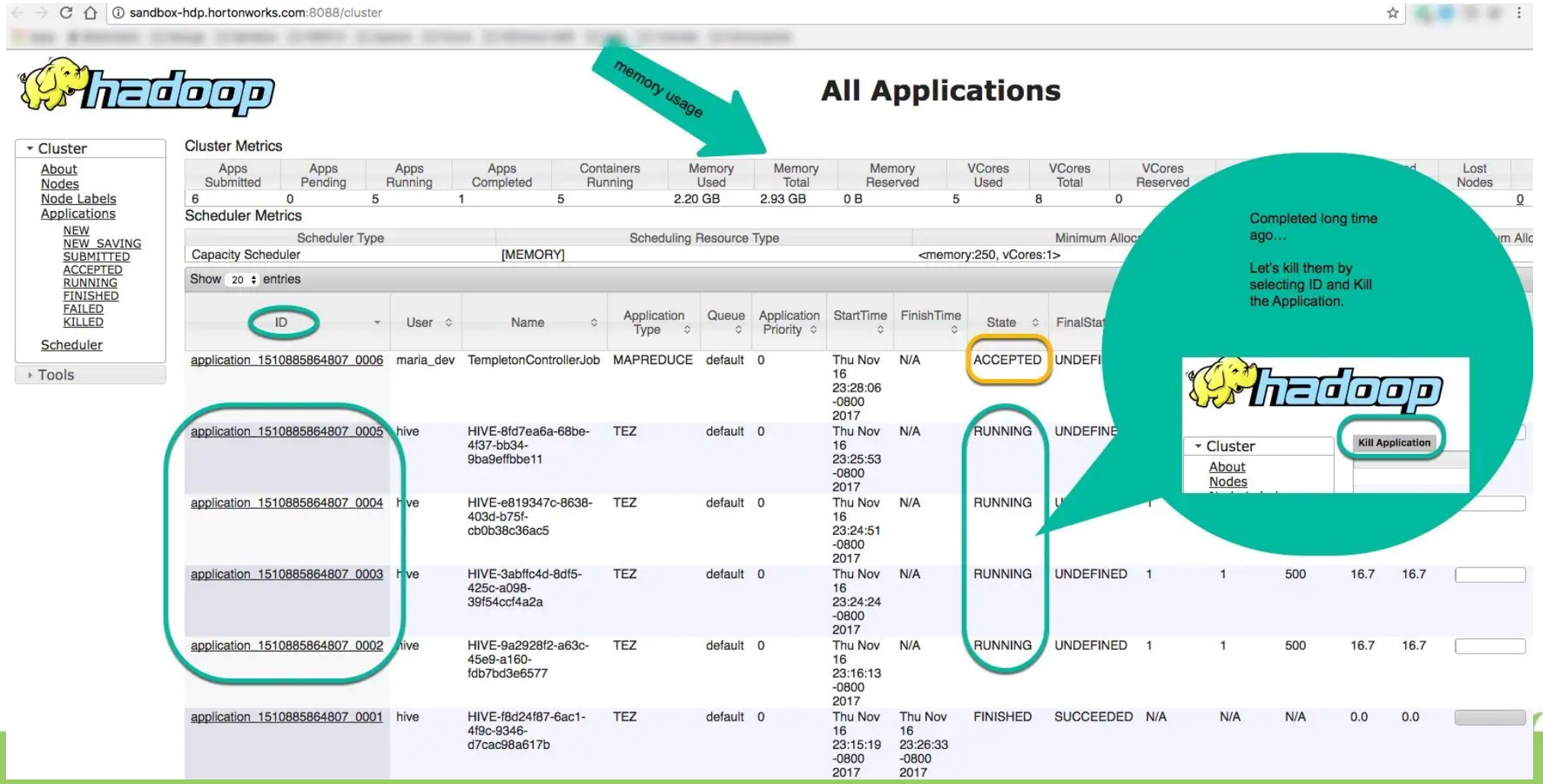
- Go back to the Dashboard
- In Views, select **YARN Queue Manager** and have a look. Do not change anything
- You could add new services at
  - Services / … / Add Service



Resource Manager (port: 8088)

https://www.cloudera.com/tutorials/learning-the-ropes-of-the-hdp-sandbox.html#appendix-b-troubleshoot

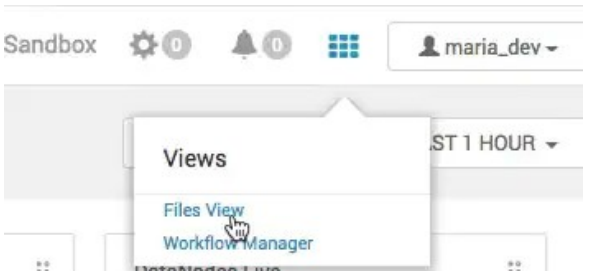
- You can kill tasks: (You will need to during this session)



Load data into HDFS (HDP)

https://www.cloudera.com/tutorials/getting-started-with-hdp-sandbox/2.html

- Download the file `Geolocation.zip` from moodle and unzip it
- Load the CSV files into HDFS
  - Log into **Ambari** as `maria_dev` (pass: maria_dev)
  - Go to Files View
  - Go to `/tmp/`
  - Create the directory `data` and go into it
  - Upload the files there (`geolocation.csv` and `trucks.csv`)
  - Go to `/tmp/`
  - Select the `data` directory and click permissions
  - Allow all the options
  - Call the teacher



HIVE 1/7

https://www.cloudera.com/tutorials/getting-started-with-hdp-sandbox/3.html

- Log into Ambari as `maria_dev`
- Select **Data Analytics Studio UI,** and then, the link underneath Quick Links
- Select Database
- Select the option to add a new Table Upload Tabla
- Indicate that the table has got a header and the file is in HDFS
- Call the teacher
- Write the path `/tmp/data/geolocation.csv` and select Preview and Create
- Wait some time, until the list of tables is updated, or click on Database if the system seems to be stuck. Call the teacher
- Repite previous steps for the file `trucks.csv`
- Have look at the Resource Manager (port: 8088). There should be a *RUNNING* process. If the system seems to be stuck, kill that process.
- Tables are in ORC format. In the Queries section, you should be able to see the executed instructions for creating the tables (well, they may not appear)
- In the Database section, you should be able to see the tables
- Call the teacher

HIVE 2/7

https://www.cloudera.com/tutorials/getting-started-with-hdp-sandbox/3.html

- You can execute sql queries at Compose
- Try the following queries:
  - `select * from trucks limit 10;` (call the teacher)
  - `show tables;`
  - `describe geolocation;`
  - `show create table geolocation;`

HIVE 3/7 (Beeline)

https://www.cloudera.com/tutorials/getting-started-with-hdp-sandbox/3.html

- Log into your EC2 instance with ssh
  - `ssh -i <key.pem file> ec2-user@<aws_url>`
- Log into the HDP server:
  - `ssh root@sandbox-hdp.hortonworks.com -p 2222`
- Start **Beeline hive**:
  - `beeline -u jdbc:hive2://sandbox-hdp.hortonworks.com:10000 -n hive`
- Grant privileges for `maria_dev`:
  - `grant all on database foodmart to user maria_dev;`
  - `grant all on database default to user maria_dev;`
  - `!quit`
- Start Beeline as `maria_dev`
  - `beeline -u jdbc:hive2://sandbox-hdp.hortonworks.com:10000 -n maria_dev`
- Execute the following queries:
  - `show databases;`
  - `!tables`
  - `show tables`
  - `select * from foodmart.customer limit 10;`
  - `select * from trucks;`
  - `select a.customer_id, count(product_id) from foodmart.customer a, foodmart.sales_fact_1997 b where a.customer_id == b.customer_id group by a.customer_id having count(product_id) > 100;`
  - `select max(subquery.compras) from (select count(product_id) compras from foodmart.customer a, foodmart.sales_fact_1997 b where a.customer_id == b.customer_id group by a.customer_id) subquery;`
  - `select a2.lname, a2.fname, a2.customer_id, count(b2.product_id) from foodmart.customer a2, foodmart.sales_fact_1997 b2 where a2.customer_id == b2.customer_id group by a2.customer_id, a2.lname, a2.fname having count(b2.product_id) == (select max(subquery.compras) from (select count(product_id) compras from foodmart.customer a, foodmart.sales_fact_1997 b where a.customer_id == b.customer_id group by a.customer_id) subquery);`
  - `!help`
  - `!describe trucks`
  - `!quit`
- **Exercise**: Raise your hand and comment that you observed and seems interesting

HIVE 4/7 (Configuration)

https://www.cloudera.com/tutorials/getting-started-with-hdp-sandbox/3.html

- Log into Ambari as admin
- Select **Hive** on the left
- Have a look at Configs and Settings tabs
- Originally, everyting in HADOOP was configured with XML files. Ambari makes things easier
- **Exercise**:
  - Look for the Vectorization properties.
  - Tell the teacher if they are checked or unchecked

HIVE 5/7

https://www.cloudera.com/tutorials/getting-started-with-hdp-sandbox/3.html

- Create the table `truckmileate` with the following command in beeline:
  - `CREATE TABLE truckmileage STORED AS ORC AS SELECT truckid, driverid, rdate, miles, gas, miles / gas mpg FROM trucks LATERAL VIEW stack(54, 'jun13',jun13_miles,jun13_gas,'may13',may13_miles,may13_gas,'apr13',apr13_miles,apr13_gas,'mar13',mar13_miles,mar13_gas,'feb13',feb13_miles,feb13_gas,'jan13',jan13_miles,jan13_gas,'dec12',dec12_miles,dec12_gas,'nov12',nov12_miles,nov12_gas,'oct12',oct12_miles,oct12_gas,'sep12',sep12_miles,sep12_gas,'aug12',aug12_miles,aug12_gas,'jul12',jul12_miles,jul12_gas,'jun12',jun12_miles,jun12_gas,'may12',may12_miles,may12_gas,'apr12',apr12_miles,apr12_gas,'mar12',mar12_miles,mar12_gas,'feb12',feb12_miles,feb12_gas,'jan12',jan12_miles,jan12_gas,'dec11',dec11_miles,dec11_gas,'nov11',nov11_miles,nov11_gas,'oct11',oct11_miles,oct11_gas,'sep11',sep11_miles,sep11_gas,'aug11',aug11_miles,aug11_gas,'jul11',jul11_miles,jul11_gas,'jun11',jun11_miles,jun11_gas,'may11',may11_miles,may11_gas,'apr11',apr11_miles,apr11_gas,'mar11',mar11_miles,mar11_gas,'feb11',feb11_miles,feb11_gas,'jan11',jan11_miles,jan11_gas,'dec10',dec10_miles,dec10_gas,'nov10',nov10_miles,nov10_gas,'oct10',oct10_miles,oct10_gas,'sep10',sep10_miles,sep10_gas,'aug10',aug10_miles,aug10_gas,'jul10',jul10_miles,jul10_gas,'jun10',jun10_miles,jun10_gas,'may10',may10_miles,may10_gas,'apr10',apr10_miles,apr10_gas,'mar10',mar10_miles,mar10_gas,'feb10',feb10_miles,feb10_gas,'jan10',jan10_miles,jan10_gas,'dec09',dec09_miles,dec09_gas,'nov09',nov09_miles,nov09_gas,'oct09',oct09_miles,oct09_gas,'sep09',sep09_miles,sep09_gas,'aug09',aug09_miles,aug09_gas,'jul09',jul09_miles,jul09_gas,'jun09',jun09_miles,jun09_gas,'may09',may09_miles,may09_gas,'apr09',apr09_miles,apr09_gas,'mar09',mar09_miles,mar09_gas,'feb09',feb09_miles,feb09_gas,'jan09',jan09_miles,jan09_gas) dummyalias AS rdate, miles, gas;`
  - In case the system seems to be stuck, kill the corresponding process at the Resource Manager (port: 8088)
- Check the result:
  - `select * from truckmileage limit 100;`
- Call the teacher

HIVE 6/7

https://www.cloudera.com/tutorials/getting-started-with-hdp-sandbox/3.html

- In **Data Analytics Studio** / Compose, write the following query:
  - `select truckid, avg(mpg) avgmpg FROM truckmileage GROUP BY truckid;`
- Save the query (Save as) with the name `average-mpg`
- You should be able to locate it in the Save tab
- Execute the query. If the system seems stuck, kill the corresponding process at Resource Manager (port: 8088)
- Select Visual Explain
- **Exercise**: Raise your hand and tell the teacher what you see
- Go to Queries, to the executed query (if it is not there, select another one)
- **Exercise**: Try to locate how much time took the query

HIVE 7/7

https://www.cloudera.com/tutorials/getting-started-with-hdp-sandbox/3.html

- Create the table `avgmileage` with two columns, the truck id and the average of its miles (this is the previous query; You may, perhaps, have to kill the process at Resource Manager)
  - `CREATE TABLE avgmileage STORED AS ORC`
    `AS`
    `SELECT truckid, …`
- Show its contents:
  - `SELECT * FROM avgmileage LIMIT 100;`
  - Call the teacher
- **Exercise**: Create the table `DriverMileage` with the dirver id (as `driverid`) and the sum of miles (as `totmiles`)
- **Exercise**: Show its contents in Data Analytics Studio. Raise your hand and compare the results with that of other students for driver `A47`
- **Exercise**: Export the result into HDFS with Export Data
  - path: `/tmp/data/drivermileage`
- Go to that file in Files View and grant all the privileges to everybody

Zeppeling I 1/7

https://www.cloudera.com/tutorials/getting-started-with-hdp-sandbox/4.html

- Check that Spark2 y Zeppeling Notebook are active
- Go to the **Zeppeling** GUI
- Select Notebook / Create new note and name it as `Compute Riskfactor`
- Interpreter: `spark2`
- You should not have any riskfa `riskfactor` table at this point (you can check that in Data Analytics Studio)
- Create a source cell with the following code, and run it:

```
%spark2
/* Conector Hive a la base de datos*/
val hiveContext = new
org.apache.spark.sql.SparkSession.Builder().getOrCreate()
```

Zeppeling I 2/7

https://www.cloudera.com/tutorials/getting-started-with-hdp-sandbox/4.html

- Create the following source cell:
  - ```
    %spark2
    /**
     * Let us first see what temporary views are already
     existent on our Sandbox */
    hiveContext.sql("SHOW TABLES").show()
    ```
- At this point, you should see nothing, due to there is not any temporal view


    ```

Zeppeling I 3/7

https://www.cloudera.com/tutorials/getting-started-with-hdp-sandbox/4.html

- Create a source cell and run it:
  - ```
    %spark2
    val geoLocationDataFrame = spark.read.format("csv").option("header",
    "true").load("hdfs:///tmp/data/geolocation.csv")

    /**
     * Now that we have the data loaded into a DataFrame, we can register a
    temporary view.
     */
    geoLocationDataFrame.createOrReplaceTempView("geolocation")
    ```
- The first instruction creates a RDD object, which is a query view.


    ```

Zeppeling I 4/7

https://www.cloudera.com/tutorials/getting-started-with-hdp-sandbox/4.html

- Show its contents with the following code in another cell:
  - ```
    %spark2
    hiveContext.sql("SELECT * FROM geolocation LIMIT 15").show()
    ```
  - Call the teacher
- Create another cell with the following code:
  - ```
    %spark2
    hiveContext.sql("DESCRIBE geolocation").show()
    ```

Zeppeling I 5/7

https://www.cloudera.com/tutorials/getting-started-with-hdp-sandbox/4.html

- The following code loads the information in `drivermileage`. In this case, we indicate the column types:
  - ```
    %spark2
    /**
     * The SQL Types library allows us to define the data types of our schema
     */
    import org.apache.spark.sql.types._
    /**
     * Recall from the previous tutorial section that the driverid schema only has
    two relations: driverid (a String), and totmiles (a Double).
     */
    val drivermileageSchema = new StructType().add("driverid",StringType,true).add("totmiles",DoubleType,true)

    val drivermileageDataFrame = spark.read.format("csv").option("header",
    "true").schema(drivermileageSchema)load("hdfs:///tmp/data/drivermileage.csv")

    drivermileageDataFrame.createOrReplaceTempView("drivermileage")
    ```
- **Exercise**: Create another source cell with this code
  - ```
    %spark2

    hiveContext.sql("select * from drivermileage").show()
    ```
- Call the teacher

Zeppeling I 6/7

https://www.cloudera.com/tutorials/getting-started-with-hdp-sandbox/4.html

- Create the following cells:
  -
```
%spark2
val geolocation_temp1 = hiveContext.sql("SELECT driverid, COUNT(driverid) occurance
from geolocation WHERE event!='normal' GROUP BY driverid")

/**
 * Show RDD
 */
geolocation_temp1.show(10)
geolocation_temp1.createOrReplaceTempView("geolocation_temp1")
hiveContext.sql("SHOW TABLES").show()
```
  -
```
%spark2
hiveContext.sql("SELECT * FROM geolocation_temp1 LIMIT 15").show()
```
  -
```
%spark2
val joined = hiveContext.sql("select a.driverid,a.occurance,b.totmiles from
geolocation_temp1 a,drivermileage b where a.driverid=b.driverid")

joined.createOrReplaceTempView("joined")
hiveContext.sql("SELECT * FROM joined LIMIT 10").show()
```
- **Exercise**: Call the teacher and tell him the result

Zeppeling I 7/7

https://www.cloudera.com/tutorials/getting-started-with-hdp-sandbox/4.html

- Compute the risk factor as:
  - ```
    %spark2
    val risk_factor_spark = hiveContext.sql("SELECT driverid,
    occurance, totmiles, totmiles/occurance riskfactor FROM
    joined")

    risk_factor_spark.createOrReplaceTempView("risk_factor_spark"
    )
    risk_factor_spark.show(10)
    ```
- Save the results with the following code:
  - ```
    %spark2
    risk_factor_spark.coalesce(1).write.option("header","true").csv("hd
    fs:///tmp/data/riskfactor.csv")
    ```
- Call the teacher

Zeppelin Reports 1/4

- Go to **Zeppelin**
- Create a new notebook with name `Driver Risk Factor`
- Load previous data:
  - ```
    %spark2
    val hiveContext = new org.apache.spark.sql.SparkSession.Builder().getOrCreate()
    val riskFactorDataFrame = spark.read.format("csv").option("header",
    "true").load("hdfs:///tmp/data/riskfactor.csv")

    riskFactorDataFrame.createOrReplaceTempView("riskfactor")
    hiveContext.sql("SELECT * FROM riskfactor LIMIT 15").show()
    ```
- Show them with:
  - ```
    %sql
    /* Lo anterior utiliza un conector jdbc a Hive */
    SELECT * FROM riskfactor
    ```

Zeppelin Reports 2/4

- Explore the different graphs options.
- Call the teacher

Zeppelin Reports 3/4

- Select Settings and Advanced chart…
- Move `driverid` to keys and `riskfactor` to values (SUM)
- Explore other options.

Zeppelin Reports 4/4

- Run the following code:
  - ```
    %sql
    SELECT a.driverid, a.riskfactor, b.city, b.state
    FROM riskfactor a, geolocation b where a.driverid=b.driverid
    ```
- Select the scatterplot.
- Move `a.driverid` to the X axis, `a.riskfactor` to Y axis, and `b.city` to the Group by field.
- **Exercise**: raise your hand and tell the teacher what you can deduce from that graph.


    ```

Kafka

Kafka

- **Apache Kafka** is a distributed event streaming system used for real-time data pipelines, integration, and analytics.
  - to collect, process, store, and integrate data at scale
- **Events**: {notification and state} (what and when) {key/value}
- **Topics**: Log of events ~ Table (partitions)
- **Kafka core**: Brokers, Producers and Consumers
- **Kafka ecosystem**
  - Kafka Connect: to connect to other systems (Source and Sink connectors)
  - Kafka Schema Registry: to register different versions of data schemas.
  - Kafka Streams: stream processing
  - KsqlDB: stream processing in SQL
- Learn Kafka: https://developer.confluent.io/learn-kafka/

Kafka

- A Kafka server manages **Topics** (tables) and the reading and writing therein.
- Messages with the same key go to the same partition, in the incoming order
  - For instance, if there events for a client of yours, and you use the client id as key, these events would go to the same topic partition
- Infrastructure
  - **Brokers** (computers), manage partitions and receives reading and writing requests
  - **Partitions** have associated several copies (leader and follower replicas)
  - Communications with the Kafka server are managed by **Producers** or **Consumers**, which are applications that you can program.
  - **Producers** write in topics, which involves managing the connection pooling, network buffering, partitioning
  - **Consumers** read from topics. They can be replicated and gathered up into groups. This means that the same application could have several consumers reading from the same topic, and combine the information as required.

Write a producer and a consumer I/II Kafka Python

- Download **examples** `ccloud_lib.py, consumer.py, producer.py` y `requirements.txt` for python from https://github.com/confluentinc/examples/tree/7.0.1-post/clients/cloud/python
- Teacher must: **Launch the Kafka cluster** (https://confluent.cloud/login)
  - Environment / Basic / … / Launch
  - Clients / Python / Create Kafka cluster API key / Save in `python.config`) and provide the file to the students.
- Save the configuration file in the directory with the downloaded examples, with name **`python.config`**
- Teacher should: Show the window of topic `test1` of the cluster
- **Execute**:
  - `virtualenv ccloud-venv`
  - `source ./ccloud-venv/bin/activate`
  - `pip install -r requirements.txt`
- Open file `producer.py` and try to understand it
- Execute: **`python producer.py -f python.config -t test1`**
- Open file `consumer.py` and try to understand it
- Execute: **`python consumer.py -f python.config -t test1`**
- Finally, to close the environment, execute `deactivate`

Write a producer & a consumer II/II Kafka Python

- **Exercise 1**: Modify producer and consumer:
  - The producer must send messages to topic **`bda_messages`** with key `<your name>` and value whatever the user writes as input. For particular messages, write "`To <whoever>: <message>`" (read from standard input until reading `exit`)
  - The consumer must show all the messages coming into the topic. Important, they are character strings, not JSON objects
- **Exercise 2**: Combine consumer and producer to have a message application (use `.upper()` to ignore capitalization):
  - Build an application with previous producer and consumer. In case it detects a message sent "`To <your name>:`", it should copy the message into a topic `for_users` with key `<your name>`. In case it detects a message speaking about you "`To ….: ….<your name>….`", it should copy the message into a topic `about_users` with key `<your name>`.
  - Write a second consumer that shows the messages coming into the topic `for_users` with key `<your name>`
  - Write a third consumer that shows the messages coming into the topic `about_users` with key `<your name>`
- All this could have been done easier with Kafka Streams (just to let you know)
- Close the cluster: Cluster overview / Cluster settings / Delete cluster

Confluent Platform Demo

https://docs.confluent.io/platform/current/tutorials/cp-demo/docs/overview.html

- Init the *Gitpod* at *Deploy Confluent Platform Environment* *(chrome, no chromium)*
- Open a terminal and run: `./scripts/start.sh`
- Wait for the services to be active
- Open the following port in Remote Explorer: `5601` (Kibana)
- Play around. For instance, how many changes were made during the last half hour? Which domain registers most of these changes?
- Open Remote Explorer at port `9021` (http://)
- Open a Confluent Platform with `superUser` / `superUser`
- Go to Kafka Raleigh / Brokers and try to understand
- Go to Topics / `wikipedia.parsed` and try to understand. How many partitions are there?
- Check the messages of the Topic
- Go to Topics / `wikipedia.parsed.count-by-domain`
- Go to Connect / `connect1`, try to understand and check the configuration of `elasticsearch-…`
- Go to ksqlDB / `wikipedia` / Flow and try to understand
- Go to ksqlDB / `wikipedia` / Running queries
- Go to Consumers / `_confluent-ksql-ksql-clusterquery_CSAS_WIKIPEDIABOT_5` and try to understand
- **Exercise:** Try to make ElasticSearch (Kibana) show the number of modifications of non-bots.