Ejecución de un programa Spark en un cluster

Comando `spark-submit`

- Permite lanzar programas Spark a un cluster
- Configuración (**NO** como usuario `root`, sino como usuario normal `luser`):

  1. Si la distribución no permite usar `pip`, hay que crear un entorno virtual y activarlo:

  ```
  $ python3 -m venv .venv
  $ source .venv/bin/activate
  ```

  (después instalar `pyspark` con `pip install pyspark`),

  1. Recordad que primero hay que hacer visible el script `spark-submit` si se ha instalado desde `pip`:

  ```
  $ export PATH=~/.local/bin:$PATH
  ```

  1. Y hay que ajustar el valor de la variable de entorno `HADOOP_CONF_DIR` para que apunte al directorio de configuración de Hadoop:

  ```
  $ export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
  ```

  1. Si se quiere conectar con jupyter o con visual studio code de forma remota, hay que instalar también el paquete `jupyter` con pip e iniciar el servidor jupyter. También hay que asegurarse de que el contenedor exporta el puerto 8888 (opción `-p 8888:8888` en el comando `docker run`):

  ```
  $ pip install jupyter
  $ jupyter notebook --ip=0.0.0.0
  ...
      To access the server copy and paste one of these URLs:
        http://127.0.0.1:8888/tree?token=45447...
  ```

La dirección `http` que muestra se puede usar para conectar al servidor usando Visual Studio Code seleccionando un "Existing Jupyter Server".

Las líneas 2 y 3 se pueden añadir al principio del fichero `.bashrc` para que se ejecuten automáticamente al abrir una terminal.

- Ejemplo:

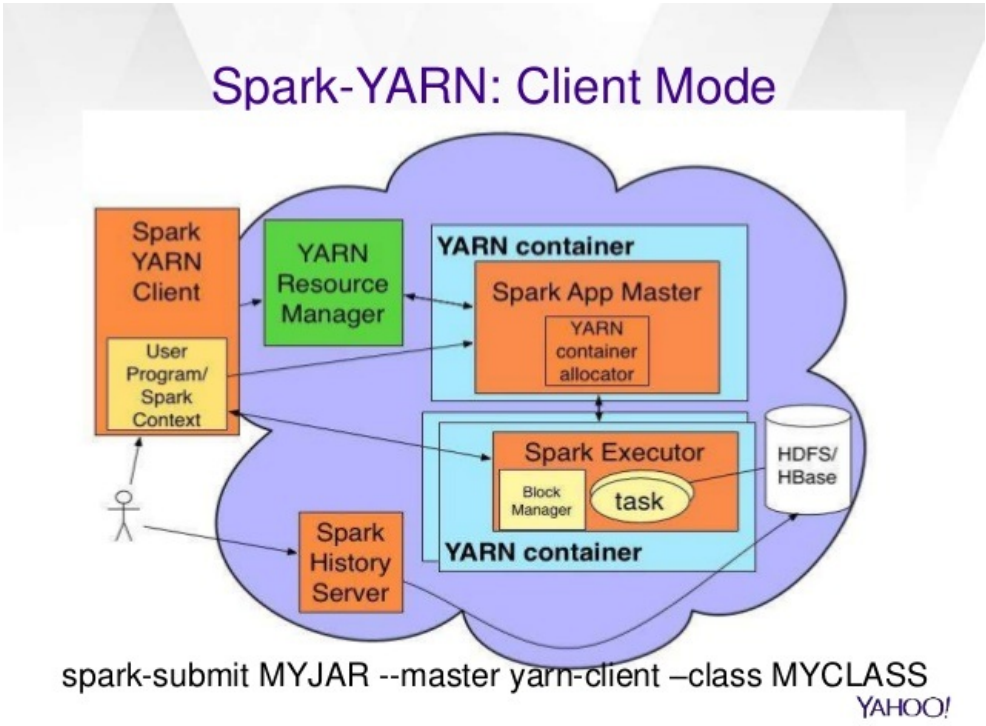
```
$ spark-submit --master yarn --deploy-mode cluster \
 --py-files otralib.zip,otrofich.py \
 --num-executors 10 --executor-cores 2 \
 mi-script.py opciones_del_script
```

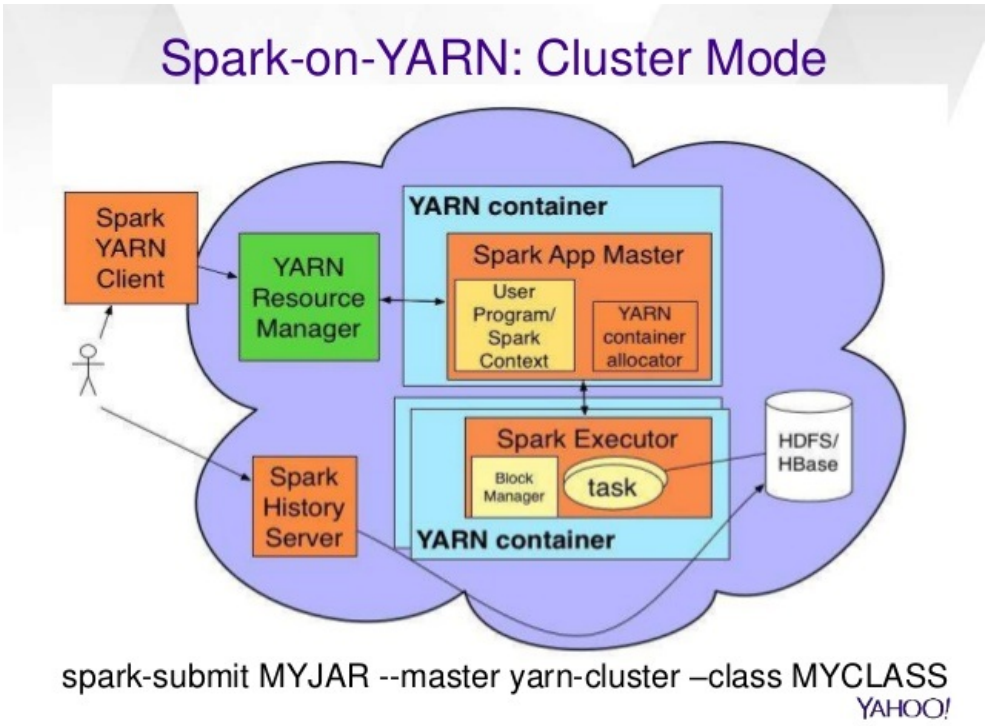
Opciones de `spark-submit`

- `master`: cluster manager a usar (opciones: `yarn`, `mesos://host:port`, `spark://host:port`, `local[n]`)
- `deploy-mode`: dos modos de despliegue
  - `client`: ejecuta el driver en el nodo local
  - `cluster`: ejecuta el driver en un nodo del cluster
- `class`: clase a ejecutar (Java o Scala)
- `name`: nombre de la aplicación (se muestra en el Spark web)
- `jars`: ficheros jar a añadir al classpath (Java o Scala)
- `py-files`: archivos a añadir al PYTHONPATH (`.py`, `.zip`, `.egg`)
- `files`: ficheros de datos para la aplicación
- `executor-memory`: memoria total de cada ejecutor
- `driver-memory`: memoria del proceso driver

Para más opciones: `spark-submit --help`

In [ ]:
```
%%sh
spark-submit --help
```





Fuente: Spark-on-YARN: Empower Spark Applications on Hadoop Cluster

---

Parámetros de configuración

Diversos parámetros ajustables en tiempo de ejecución

- En el script

```
from pyspark.sql import SparkSession
from pyspark import SparkContext
# Creamos un objeto SparkSession (o lo obtenemos si ya está creado)
spark: SparkSession = SparkSession \
  .builder \
  .appName("Mi aplicacion") \
  .master("local[2]") \ # Cluster manager modo local con 2 hilos
  .getOrCreate()
# Obtenemos el SparkContext
sc: SparkContext = spark.sparkContext
```

- Mediante flags en el `spark-submit`

```
$ spark-submit --master local[2] --name "Mi apli" mi-script.py
```

- Mediante un fichero de propiedades:

```
$ cat config.conf
spark.master     local[2]
spark.app.name   "Mi apli"
spark.ui.port 4040
$ spark-submit --properties-file config.conf mi-script.py
```

Más info: http://spark.apache.org/docs/latest/configuration.html#spark-properties

Ejemplo de ejecución de un script Python

In [ ]:
```
%%writefile "/tmp/miscript.py"
# -*- coding: utf-8; -*-
from pyspark.sql import SparkSession
from pyspark.sql.functions import sum,col

def main():
    spark: SparkSession = SparkSession \
      .builder \
      .appName("Mi script Python") \
      .getOrCreate()

    # Cambio la verbosidad para reducir el número de
    # mensajes por pantalla
    spark.sparkContext.setLogLevel("FATAL")

    df1 = spark.range(2, 10000000, 2)
    df2 = spark.range(2, 10000000, 4)
    step1 = df1.repartition(5)
    step12 = df2.repartition(6)
    step2 = step1.selectExpr("id * 5 as id")
    step3 = step2.join(step12, ["id"])
    step4 = step3.select(sum(col("id")))

    # step4 es un dataframe con una única fila
    # que es un objeto Row.
    # Con collect() obtengo la fila como una lista
    # me quedo con el primer elemento (Row) de la lista
    # y lo convierto a un diccionario Python
    salida = step4.collect()[0].asDict()['sum(id)']
    print("Resultado final = {0}".format(salida))

if __name__ == "__main__":
    main()
```

In [ ]:
```
%%sh
cat /tmp/miscript.py
```

In [ ]:
```
%%sh
spark-submit --master local[8] /tmp/miscript.py
```

Lo siguiente lanza la aplicación en el clúster. El modo de deploy se ha dejado en local, por lo que el máster se ejecuta en local y producirá la misma salida que el de arriba.

In [ ]:
```
%%sh
# Para lanzarlo en el clúster:
#spark-submit --master yarn /tmp/miscript.py
```

Si se ejecuta con el deployment en el clúster no se muestran los resultados, y hay que acceder a los logs en la web del YARN, o bien con un comando para ver los logs de YARN:

```
$ yarn logs -applicationId <application_id>
```

(donde `<application_id>` es el ID de la aplicación que se muestra en la salida de `spark-submit`).

In [ ]:
```
%%sh
#spark-submit --master yarn --deploy-mode cluster /tmp/miscript.py
```