

Instalar y ejecutar Pyspark con docker

Introducción a Docker

Docker es una herramienta que puede empaquetar una aplicación y sus dependencias en un contenedor virtual que se puede ejecutar en cualquier servidor Linux. Esto ayuda a permitir la flexibilidad y portabilidad en donde la aplicación se puede ejecutar, ya sea en las instalaciones físicas, la nube publica, nube privada, etc.

Docker permite introducir en un contenedor todas aquellas dependencias que mi aplicación necesita para ejecutarse(Servidor de aplicaciones,base de datos..),junto con la propia aplicación.

De esta forma podemos llevar este contenedor a cualquier máquina que tenga instalado docker y la aplicación debería funcionar igual que como la ejecutas en tu máquina local.

Se ejecutará la aplicación desde el contenedor de docker y dentro de él estarán todas las dependencias a nivel de librerías para que la aplicación funcione.

Es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de Virtualización a nivel de sistema operativo.

La idea detrás de Docker es crear contenedores ligeros y portables para las aplicaciones software que puedan ejecutarse en cualquier máquina con Docker instalado, independientemente del sistema operativo que la máquina tenga por debajo, facilitando así también los despliegues

Para aquellos que no lo posean instalado, les dejo la documentación oficial para las distintas plataformas:¿Cómo instalo Docker?

<https://docs.docker.com/engine/installation/>

Entre las principales ventajas para trabajar con un entorno virtualizado podemos destacar:

- Trabajar con un entorno virtualizado te permite aislar tu proyecto de la infraestructura del SO que tengas por abajo, permitiendo probar varias configuraciones sin tener que desinstalar e instalar paquetes en tu máquina host.
- Te permite tener un entorno que puedas escalar fácilmente.

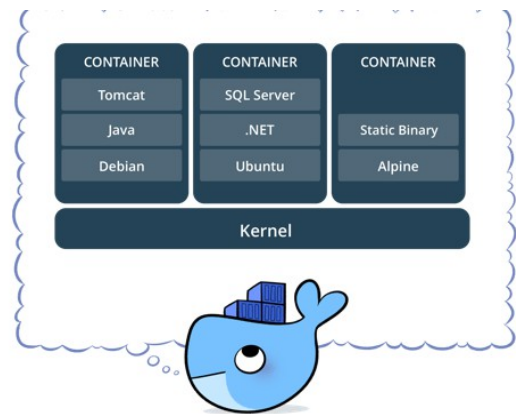
Enlaces

[https://es.wikipedia.org/wiki/Docker_\(software\)](https://es.wikipedia.org/wiki/Docker_(software))

<https://www.docker.com/what-container>

Package software into standardized units for development, shipment and deployment

A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings. Available for both Linux and Windows based apps, containerized software will always run the same, regardless of the environment. Containers isolate software from its surroundings, for example differences between development and staging environments and help reduce conflicts between teams running different software on the same infrastructure.



Definiciones

DockerFile: Archivo de configuración para crear imágenes. Se indica lo que se quiere que tenga la imagen y los comandos para instalar las herramientas

Docker Image: Captura del estado en el que se encuentra un contenedor, como una especie de plantilla

Contenedor: Instancia en ejecución de una imagen. A partir de una imagen podemos ejecutar varios contenedores.

Comandos útiles de docker

En este apartado se mencionan una serie de comandos de relativa importancia para el uso de Docker. (La mayoría de ellos deben ser precedido por el comando sudo).

\$ docker-compose build: Construye o reconstruye los servicios. Los servicios son contruidos una unica vez y luego son etiquetados como project_service. Si se modifica el Dockerfile de un servicio o el contenido del directorio de construcción se debe ejecutar de nuevo este comando para reconstruirlo.

\$ docker-compose up: Construye, inicia, y agrega los contenedores para un servicio.

\$ docker-compose start: Inicia los contenedores existentes para un servicio.

\$ docker-compose stop: Detiene los contenedores que se están ejecutando sin eliminarlos. Pueden volverse a iniciar con el comando anterior (start).

\$ docker ps -a: Muestra información de todos los contenedores que existen actualmente y en que estado se encuentran, ademas de otra información adicional.

\$ docker rm <Contanider ID>: Este comando permitirá eliminar un contenedor.

\$ docker images: Muestra información acerca de cada una de las imágenes que se encuentran en nuestra maquina (nombre, id, espacio que ocupa, el tiempo que transcurrió desde que fue creada).

\$ docker rmi <Image ID>: Este comando permitirá eliminar una imagen.

Referencias

Documentación oficial Docker.

<https://docs.docker.com/>

Información básica de instalación Docker-engine:

<https://docs.docker.com/engine/installation/linux/ubuntu/linux/>

Información básica de instalacion Docker-compose:

<https://docs.docker.com/compose/install/>

Trabajando docker con pyspark

Para trabajar con pyspark y docker tenemos varias alternativas.

La primera es usar el proyecto que se encuentra en la cuenta de github [PySpark_docker](https://github.com/joseerlang/PySpark_docker) que te va a permitir crear un entorno virtualizado con Spark 2.1 sobre hadoop 2.7

https://github.com/joseerlang/PySpark_docker

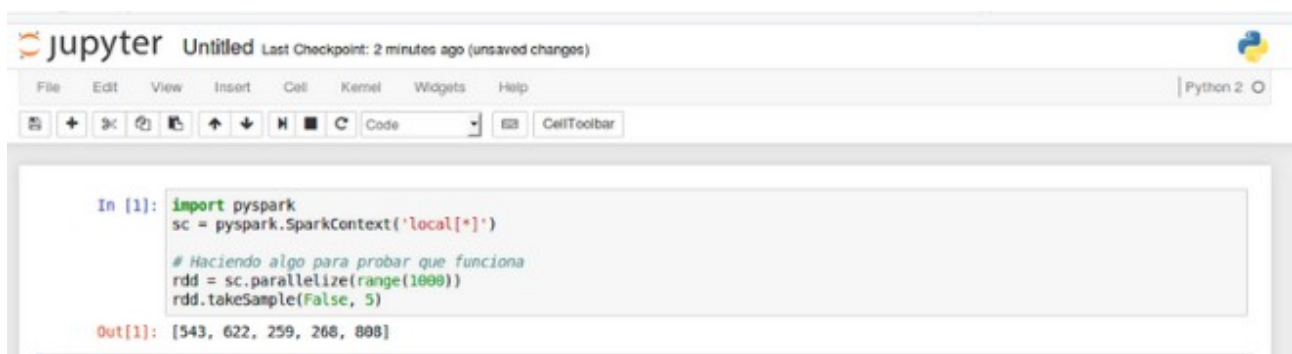
Una vez que tengas el repositorio sincronizado en tu entorno de trabajo y **docker** y **docker-compose** instalado, puedes ejecutar el siguiente comando:

docker-compose up

Este comando arrancará la creación de una imagen que a partir de java8 incluirá tanto las librerías de python como spark en la versión 2.1 sobre hadoop 2.7. Si todo va bien, al cabo de un par de minutos deberías ser capaz de ver en el shell de tu pc algo parecido a esto:

Como puedes ver, obtenemos una url que lanzará jupyter en modo standalone y configurado para utilizar python y spark.

Si accedes a la url que te proporciona el log de la imagen que hemos creado




Para aquellos que quieran experimentar con PySpark y les parezca algo complicado el proceso de instalación, pueden hacer uso de la imagen de Docker ofrecida por [Jupyter](https://hub.docker.com/r/jupyter/pyspark-notebook/) disponible en su repositorio oficial. Esta imagen nos ofrece un entorno con todo preparado y adicionalmente nos provee de una interfaz web para la interacción de forma dinámica con PySpark.

<https://hub.docker.com/r/jupyter/pyspark-notebook/>

Para bajarse la imagen de Jupyter habría que ejecutar el comando:

docker pull jupyter/pyspark-notebook



[Explore](#)
[Help](#)
[Sign up](#)
[Sign in](#)

PUBLIC REPOSITORY

jupyter/pyspark-notebook

Last pushed: 10 days ago

[Repo Info](#)
[Tags](#)

Short Description

Jupyter Notebook Python, Spark, Mesos Stack from <https://github.com/jupyter/docker-stacks>

Docker Pull Command

`docker pull jupyter/pyspark-notebook`

Full Description

Jupyter Notebook Python, Spark, Mesos Stack

An image from the [jupyter/docker-stacks](#) project, a set of opinionated stacks of ready-to-run Jupyter applications in Docker.

See the [pyspark-notebook README](#) and [Dockerfile](#) for information about the contents and options of this particular image. Also, see the [docker-stacks README](#) for details about the project including how these






Owner

 * jupyter

En el repositorio de github podemos encontrar lo que contiene esta imagen

<https://github.com/jupyter/docker-stacks>

En concreto podemos ver que hay una carpeta llamada pyspark-notebook

 datascience-notebook	Merge pull request #387 from stolho/master
 examples	Update docker-compose example README.
 internal	add tensorflow-notebook in the diagram
 minimal-notebook	Added description of NB_GID to README.md files
 pyspark-notebook	Added description of NB_GID to README.md files

Dentro de esta carpeta podemos encontrar el fichero DockerFile que contiene toda la configuración para levantar una imagen con python y pyspark

<https://github.com/jupyter/docker-stacks/blob/master/pyspark-notebook/Dockerfile>

```
ENV APACHE_SPARK_VERSION 2.1.0
ENV HADOOP_VERSION 2.7

# Temporarily add jessie backports to get openjdk 8, but then remove that source
RUN echo 'deb http://cdn-fastly.deb.debian.org/debian jessie-backports main' > /etc/apt/sources.list.d/jessie-backports.list && \
    apt-get -y update && \
    apt-get install --no-install-recommends -t jessie-backports -y openjdk-8-jre-headless ca-certificates-java && \
    rm /etc/apt/sources.list.d/jessie-backports.list && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*
RUN cd /tmp && \
    wget -q http://d3kbcqa49mib13.cloudfront.net/spark-${APACHE_SPARK_VERSION}-bin-hadoop${HADOOP_VERSION}.tgz && \
    echo "3fc94096ae34f9a1a148d37e5ed640a7e5de1812f1f2ecd715d92bbf2901e895cf4b93e6d8ee0d64debb5df7c56d673c0a36e5fc49503ec0f4507eb0edf96" \
    tar xzf spark-${APACHE_SPARK_VERSION}-bin-hadoop${HADOOP_VERSION}.tgz -C /usr/local && \
    rm spark-${APACHE_SPARK_VERSION}-bin-hadoop${HADOOP_VERSION}.tgz
RUN cd /usr/local && ln -s spark-${APACHE_SPARK_VERSION}-bin-hadoop${HADOOP_VERSION} spark
```

En el fichero README de esta carpeta podemos ver todo lo que contiene esta imagen.

Jupyter Notebook Python, Spark, Mesos Stack

What it Gives You

- Jupyter Notebook 5.0.x
- Conda Python 3.x and Python 2.7.x environments
- pyspark, pandas, matplotlib, scipy, seaborn, scikit-learn pre-installed
- Spark 2.1.0 with Hadoop 2.7 for use in local mode or to connect to a cluster of Spark workers
- Mesos client 0.25 binary that can communicate with a Mesos master
- Unprivileged user `jovyan` (uid=1000, configurable, see options) in group `users` (gid=100) with ownership over `/home/jovyan` and `/opt/conda`
- `tini` as the container entrypoint and `start-notebook.sh` as the default command
- A `start-singleuser.sh` script useful for running a single-user instance of the Notebook server, as required by JupyterHub
- A `start.sh` script useful for running alternative commands in the container (e.g. `ipython`, `jupyter kernelgateway`, `jupyter lab`)
- Options for a self-signed HTTPS certificate and passwordless `sudo`

Para ejecutar esta imagen de docker habría que ejecutar el comando:

`docker run -it --rm -p 8888:8888 jupyter/pyspark-notebook`