

# Docker para Data Science

# Docker para Data Science

- Introducción
- Dockerización de aplicaciones R
- Dockerización de aplicaciones Python
- Uso de Jupyter dockerizado

# Docker para Data Science

- **Introducción**
- Dockerización de aplicaciones R
- Dockerización de aplicaciones Python
- Uso de Jupyter dockerizado

# Docker para Data Science

- Docker tiene las siguientes ventajas para los desarrolladores en **Data Science**:
  - Te permite tener en un **entorno local** todo el software necesario para el procesamiento de datos de forma sencilla
  - Te permite **empaquetar** tu aplicación con todo lo necesario para que se pueda ejecutar sin problemas en servidores

# Docker para Data Science

- Docker es una tecnología genérica para empaquetar y desplegar aplicaciones
- Veremos ejemplos de aplicaciones habituales en el contexto del Data Science
  - Dockerización de aplicaciones R
  - Dockerización de aplicaciones Python
  - Jupyter

# Docker para Data Science

- Todos los ejemplos que se verán en la siguientes diapositivas están en el repositorio

<https://github.com/codeurjc/docker-datascience>

- Para clonar el repositorio se puede utilizar los siguientes comandos

```
$ git clone https://github.com/codeurjc/docker-datascience.git
$ cd docker-datascience
```

# Docker para Data Science

- Introducción
- **Dockerización de aplicaciones R**
- Dockerización de aplicaciones Python
- Uso de Jupyter dockerizado

# Dockerización de R

- Al dockerizar un Script de R no dispondremos de interfaz gráfico
- La información generada deberá ser mostrada o bien por pantalla o bien guardada en ficheros en disco (PNG, JPG, PDF, TXT...)
- Deberemos instalar en la imagen las librerías necesarias para que el Script funcione



# Dockerización de R

- **Script R sin dependencias**
  - Script de R que imprime una gráfica en un fichero PNG
  - El Script no necesita dependencias adicionales
  - Dispondremos de los ficheros **main.R** y **Dockerfile**

# Ejemplo Script R sin dependencias

- main.R

```
message("Sample Script in R")
message("Creating cars and trucks arrays...")

cars <- c(1, 3, 6, 4, 9)
trucks <- c(2, 5, 4, 5, 12)

message("Creating plot using the cars and trucks arrays...")

if (!dir.exists("/tmp/output")) {
  dir.create("/tmp/output")
}

png(filename="/tmp/output/plot.png")
plot(cars, type="o", col="blue", ylim=c(0,12))
lines(trucks, type="o", pch=22, lty=2, col="red")
title(main="Autos", col.main="red", font.main=4)
dev.off()
```

# Ejemplo Script R sin dependencias

- **Dockerfile**

```
FROM r-base:4.2.2
WORKDIR /usr/src/myscript/
COPY main.R /usr/src/myscript/
CMD ["Rscript", "main.R"]
```

- **Compilación de la imagen**

```
$ docker build -t miusuario/ejemplo-1 .
```

- **Ejecutar imagen**

```
$ docker run -v $PWD:/tmp/output miusuario/ejemplo-1
```

# Ejemplo Script R con dependencias

- **Script R con dependencias**
  - Script de R que realiza una predicción
  - El Script necesita la instalación de dependencias para el entrenamiento del modelo
  - Dispondremos de los ficheros **predict.R**, **requirements.R** y **Dockerfile**

# Ejemplo Script R con dependencias

- **predict.R**

```
library(randomForest)
args <- commandArgs(T)

input <- data.frame(
  Sepal.Length = args[1],
  Sepal.Width = args[2],
  Petal.Length = args[3],
  Petal.Width = args[4]
)

if (! file.exists("/tmp/model/iris_clf_rf.rds")){
  message('Training model...')
  clf_rf <- randomForest(Species ~ ., data = iris)

  if (! dir.exists("/tmp/model")) {
    dir.create("/tmp/model")
  }

  saveRDS(clf_rf, '/tmp/model/iris_clf_rf.rds')
} else {
  message('Reading model...')
  clf_rf <- readRDS('/tmp/model/iris_clf_rf.rds')
}

predict(clf_rf, input)
```

# Ejemplo Script R con dependencias

- requirements.R

```
install.packages("randomForest")
```

- Dockerfile

```
FROM r-base:4.2.2
WORKDIR /usr/src/myscript/
COPY requirements.R /usr/src/myscript/
RUN Rscript requirements.R
COPY predict.R /usr/src/myscript/
ENTRYPOINT [ "Rscript", "predict.R" ]
CMD [ "5.1", "3.5", "1.4", "0.2" ]
```

# Ejemplo Script R con dependencias

- **Compilación de la imagen**

```
$ docker build -t miusuario/ejemplo-2 .
```

- **Ejecución de los valores por defecto**

```
$ docker run -v $PWD:/tmp/model miusuario/ejemplo-2
```

- **Ejecución con valores de entrada diferentes**

```
$ docker run -v $PWD:/tmp/model \  
  miusuario/ejemplo-2 7.0 3.2 4.7 1.4
```

# Docker para Data Science

- Introducción
- Dockerización de aplicaciones R
- **Dockerización de aplicaciones Python**
- Uso de Jupyter dockerizado



# Dockerización de Python

- Al dockerizar un Script de Python al igual que pasaba con R no dispondremos de una interfaz gráfica
- La información generada deberá ser mostrada o bien por pantalla o bien guardada en ficheros en disco (PNG, JPG, PDF, TXT...)
- Deberemos instalar en la imagen las librerías necesarias para que el Script funcione

# Dockerización de Python

- **Script Python con dependencias**
  - Script de Python que realiza una predicción
  - El Script necesita la instalación de dependencias para el entrenamiento del modelo
  - Dispondremos de los ficheros **predict.py**, **requirements.txt** y **Dockerfile**

# Ejemplo Script Python

- **predict.py**

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
import argparse
import joblib
import os

__species = {
0: 'setosa',
1: 'versicolor',
2: 'virginica'
}

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('-i', '--input', nargs='+', type=float, action='append')
    args = parser.parse_args()

    if not os.path.exists('/tmp/model/iris_clf_rf.pkl'):
        print('Training model...')
        X, y = load_iris(return_X_y=True)
        clf_rf = RandomForestClassifier(n_estimators=10, random_state=0)
        clf_rf.fit(X, y)

        if not os.path.isdir('/tmp/model'):
            os.mkdir("/tmp/model")
            joblib.dump(clf_rf, '/tmp/model/iris_clf_rf.pkl')
        else:
            print('Reading model...')
            clf_rf = joblib.load('/tmp/model/iris_clf_rf.pkl')
            y_pred = clf_rf.predict(args.input)

    for el_x, el_y in zip(args.input, y_pred):
        print(f'Prediction for {el_x}: {__species[el_y]}')
```

# Ejemplo Script Python

ejemplo-3

- requirements.txt

```

joblib==1.2.0
numpy==1.23.5
scikit-learn==1.2.0
scipy==1.9.3
threadpoolctl==3.1.0

```

- Dockerfile

```

FROM python:3.11.1-slim-buster
WORKDIR /usr/src/myscript/
COPY requirements.txt /usr/src/myscript/
RUN pip install --no-cache-dir -r requirements.txt
COPY predict.py /usr/src/myscript/
ENTRYPOINT [ "python", "predict.py" ]
CMD [ "--input", "5.1", "3.5", "1.4", "0.2" ]

```

# Ejemplo Script Python

ejemplo-3

- **Compilación de la imagen**

```
$ docker build -t miusuario/ejemplo-3 .
```

- **Ejecución de los valores por defecto**

```
$ docker run -v $PWD:/tmp/model miusuario/ejemplo-3
```

- **Ejecución con valores de entrada diferentes**

```
$ docker run -v $PWD:/tmp/model \
  miusuario/ejemplo-3 --input 7.0 3.2 4.7 1.4
```

# Docker para Data Science

- Introducción
- Dockerización de aplicaciones R
- Dockerización de aplicaciones Python
- **Uso de Jupyter dockerizado**

# Jupyter dockerizado

- Jupyter es una herramienta que nos ofrece una Shell interactiva vía Web
  - Jupyter Notebook
  - JupyterLab
- Permite escribir texto en Markdown o Latex
- Permite escribir código en multitud de lenguajes

<https://jupyter.org/>



# Jupyter dockerizado

- Jupyter nos ofrece una serie de imágenes docker que podremos utilizar dependiendo de nuestras necesidades
  - jupyter/base-notebook
  - jupyter/minimal-notebook
  - jupyter/r-notebook
  - jupyter/datascience-notebook
  - jupyter/all-spark-notebook
  - ...



# Jupyter dockerizado

ejemplo-4

- Una vez decidamos la imagen que mejor se adapta a nuestras necesidades podremos lanzar Jupyter

```
$ docker run -d --rm \
  -p 8888:8888 \
  -v $PWD:/notebooks \
  jupyter/datascience-notebook start-notebook.sh \
    --NotebookApp.token='password' \
    --notebook-dir=/notebooks
```

# Jupyter dockerizado

ejemplo-4

- Acceso a Jupyter Notebook

<http://localhost:8888/>

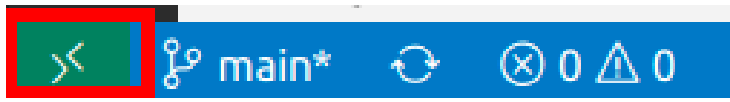
- Acceso a JupyterLab

<http://localhost:8888/lab>

# Jupyter Remote Containers

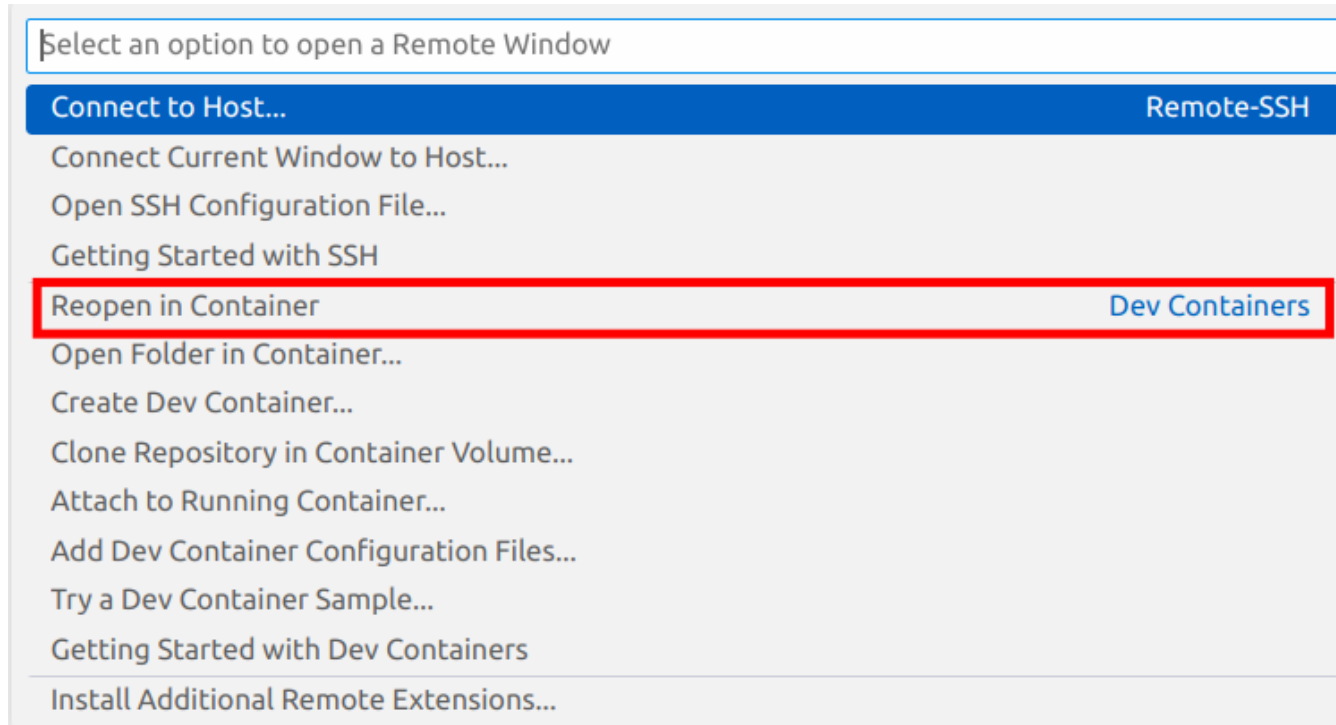
ejemplo-4

- En **VSCode** también podremos utilizar la imagen de Jupyter para hacer uso de ella con **Remote Containers**
  - Abrimos el proyecto **ejemplo-4** y pulsamos



# Jupyter Remote Containers

- Seleccionamos “Reopen in Container”










# Jupyter Remote Containers

- Utilizamos la imagen “Jupyter Data Science Notebooks community”

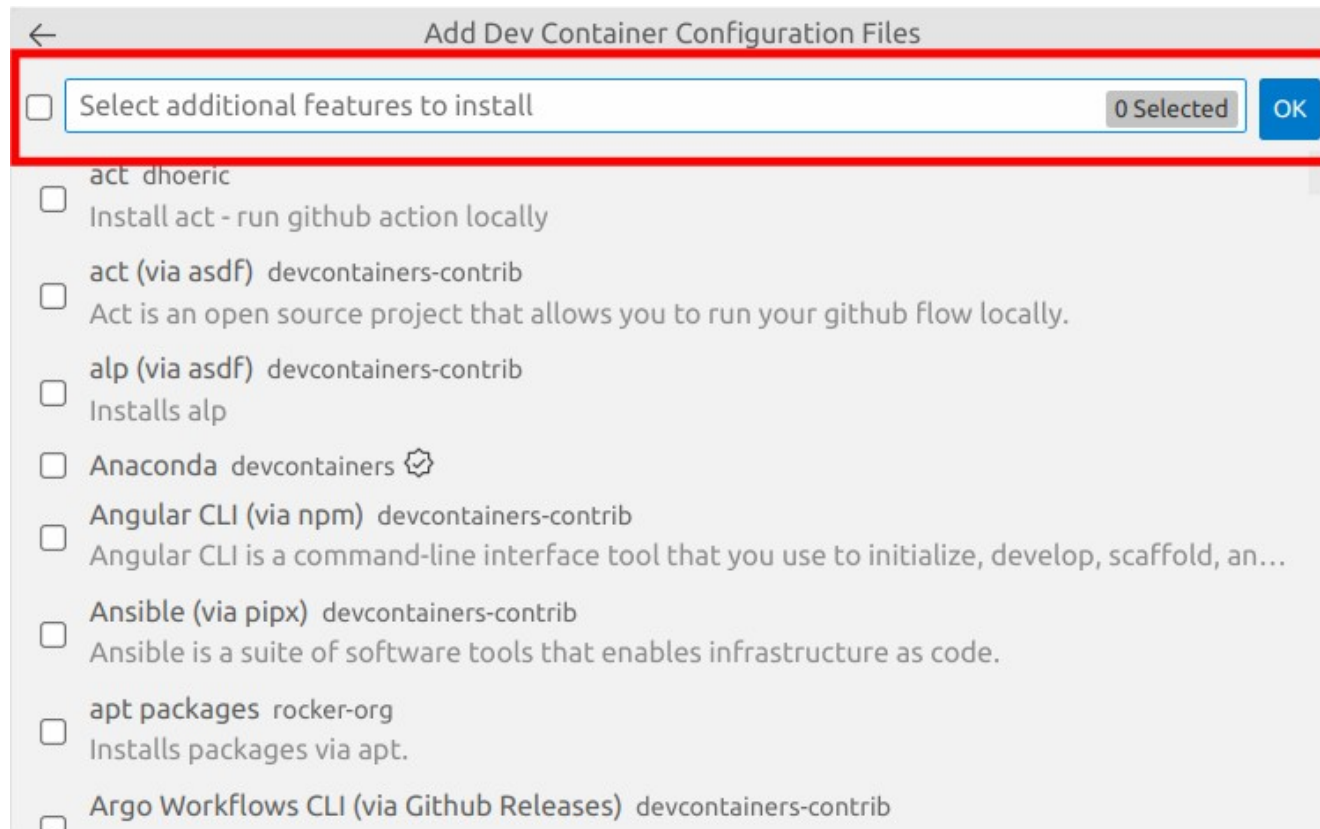
Add Dev Container Configuration Files

Select a container configuration template

- Default Linux Universal devcontainers   
Use or extend the new Ubuntu-based default, large, multi-language universal container for Git...
- Docker in Docker devcontainers   
Create child containers \_inside\_ a container, independent from the host's docker instance. Inst...
- Docker outside of Docker devcontainers   
Access your host's Docker install from inside a dev container. Installs Docker extension in the c...
- Docker outside of Docker Compose devcontainers   
Access your host's Docker install from inside a container when using Docker Compose. Installs ...
- Jupyter Data Science Notebooks community**   
Use Jupyter Data Science Notebooks with Python, R, Julia, and more.
- Kubernetes - Local Configuration devcontainers   
Access a local (or remote) Kubernetes cluster from inside a dev container using your local con...
- Kubernetes - Minikube-in-Docker devcontainers   
Access an embedded minikube instance or remote a Kubernetes cluster from inside a dev cont...

# Jupyter Remote Containers

- No añadimos nada adicional y pulsamos "OK"



# Jupyter Remote Containers

- Una vez cargado el entorno podremos
  - Ejecutar comandos dentro del contenedor desde la consola de VSCode

```
$ python test_parameters.py
```

# Jupyter Remote Containers

ejemplo-4

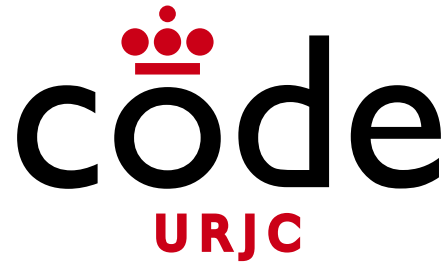
- Conectarnos a Jupyter Notebook y JupyterLab  
<http://localhost:8888/>      <http://localhost:8888/lab>
- Para saber el Token por defecto ejecutamos en la consola de VSCode

```
$ jupyter server list
```

- Copiamos el token y lo utilizamos para conectarnos

```
jovyan → /workspaces/docker-datascience/ejemplo-4 (main x) $ jupyter server list
Currently running servers:
http://270f2504bbc8:8888/?token=c7956a639b1f7fd3333d6641d67560ac4486f0095fe654db :: /home/jovyan
```





©2022 Óscar Soto Sánchez

Algunos derechos reservados

Este documento se distribuye bajo la licencia  
"Atribución-CompartirIgual 4.0 Internacional"  
de Creative Commons Disponible en  
<https://creativecommons.org/licenses/by-sa/4.0/deed.es>