UNIL | Université de Lausanne

Week 9 - Big-Scale Analytics

Introduction to Docker

Docker is a set of Platform as a Service (PaaS) products that uses OS-level virtualisation to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating system kernel and therefore use fewer resources than virtual machines (definition from Wikipedia). The software that hosts the containers is called **Docker Engine.**
Docker can package an application and its dependencies in a virtual container that can run on any Linux server. This will provide **flexibility** and **portability** enabling the application to be run in various locations, wether on-premises (e.g, your computer), in a public cloud, or in a private cloud. Therefore, you don't need to worry about dependencies if you want to run your application in different locations. Moreover Docker containers are lightweight and they use fewer resources than virtual machine, so you can run several containers on a single server simultaneously.

Installing Docker Desktop

Please refer to the Docker documentation to see how to install Docker Desktop. Docker Desktop is an easy-to-install application for your Mac or Windows environment that enables you to build and share containerized applications and microservices. Docker Desktop includes Docker Engine, Docker CLI client, Docker Compose, Notary, Kubernetes, and Credential Helper.

Running a Docker image

In this section we will see how to run a docker image. Docker images are templates which are used to create Docker containers. They contain all the files, dependencies, etc required to run a container. Container is a running instance of an image. You can find many different pre-built Docker images in Docker Hub. In today's lab we are going to run a Docker image which will run a Python script.
Open the terminal in your computer (cmd in windows) and run the following command. You will be able to see all of the docker images you have on your system:

```
docker images
```

Next, download the Docker image from the following link:
https://drive.switch.ch/index.php/s/Jc2vjxsBDMM0GHl

Load the downloaded image, which is a ".tar" file, using the following command:

```
docker load < "image_name"
```

Now if you run "`docker images`" again, you will see the new image in the list. Next you will start a container based on this image by running the following:

```
docker run "image_name"
```

You will see that as the container starts, the python script will be executed and you will see the result in your terminal page. The script will randomly select a Wikipedia article and returns the title and the summary of the article. As you can see, the container stops as soon as the execution of the Python script is finished. Can you explain why this happens?

You can also see the list of docker containers running on your computer:

```
docker ps
```

And if you want to see the list of all containers, whether running or stopped, run:

```
docker ps -a
```

For deleting a container simply run: And for deleting images, run:

```
docker rm "container_id"
docker rmi "image_name"
```

Running a Docker image in an interactive mode

Follow this 12 minutes tutorial video that explains how to pull an image from Docker Hub and run it in interactive mode.

Building a Docker image

Docker images are templates that contain all the files, dependencies, etc required to run a container. The main step in building an image is creating a "`dockerfile`". Docker can build images automatically by reading the instructions from a "`dockerfile`". A "`dockerfile`" is a text document that contains all the commands a user could call on the command line to assemble an image. Using "`docker build`" users can create an automated build that executes several command-line instructions in succession.

When dockerizing Python applications, another important text file to create is the "`requirements.txt`" file. It contains all the dependencies (i.e, Python libraries) that are required by the application and should be installed.

In what follows, we will give you a walk-through of dockerizing a Python Flask application.

Dockerizing a Flask application

Before starting, you need to create a new directory that will contain your docker files. Create a new directory with a name of your choice and copy the Flask app files from week 9 to this directory. We are going to dockerize the Flask app from week 7.

Important note: make sure that inside the application.py script, for `app.run()` method you have set the host to be 0.0.0.0 (example below)

```
if __name__ == '__main__':
    app.run(debug = True, host='0.0.0.0')
```

This is a necessity, as 0.0.0.0 is a wildcard IP address that will match any possible incoming port on the host machine.

The next step is to create a dockerfile. Create a new file named **dockerfile** (note that this file **should not have any extension**) and edit it with an editor of your choice. Here's a view of how this file should look like in our case:

```
FROM ubuntu:18.04

MAINTAINER Your Name “your_email”

RUN apt-get update -y && \
    apt-get install -y python3-pip python3-dev

COPY ./requirements.txt /app/requirements.txt

WORKDIR /app

RUN pip3 install --upgrade pip
RUN pip3 install -r requirements.txt
RUN python3 -m spacy download en

COPY . /app
EXPOSE 5000

ENTRYPOINT [ "python3" ]

CMD [ "application.py" ]
```

Let’s go over this dockerfile line by line:

1. We base our image on ubuntu version 18.04. This means that when building the image, docker will pull the image of this version of ubuntu from Docker Hub.
2. `MAINTAINER` sets the author field of the image. This will be useful if you want to push your image to Docker Hub later.
3. The command coming after `RUN` will be executed in a shell. This means that Python3 and pip3 will be installed on ubuntu (the parent image)
4. Copy the “requirements.txt” file from the local machine to “/app” directory in the ubuntu image.
5. Set the working directory in ubuntu to be “/app”. This is equivalent to when you do “cd /app” in the terminal of your machine.
6. Update pip3
7. Install the libraries inside “requirement.txt” file.
8. Install English language data for spacy.
9. Copy all the files in the current directory of your machine (the directory in which dockerfile is) to the “/app” directory in ubuntu. This will copy the Python scripts to “/app” directory.
10. The `EXPOSE` instructions informs Docker that the container listens on the specified network ports at runtime. The port 5000 is the one used by Flask
11. `ENTRYPOINT` configures the container to run as an executable.Here we want to execute a Python3 script.
12. Finally, the last line will execute the “application.py” script

Note that the last two line will internally run `python3 application.py` inside the docker container.

It only remains to create the “requirements.txt” file. We need the following libraries for our application. Note that the versions are specified after the “==” sign.

```
Flask==1.1.2
joblib==0.17.0
numpy==1.19.2
scipy==1.5.2
scikit-learn==0.23.2
pandas==1.1.3
spacy==2.3.2
```

Now you are ready to build your image. You can do it by simply running the following in the terminal:

```
docker build -t flask-text-classification-app:latest .
```

What comes before “:” is the name of the image and what comes after it specifies that this is the latest version of this image. The “.” Means that we want to build the docker file in the current directory. After the build completed, you can run the container:

```
docker run -it -d -p 5000:5000 flask-text-classification-app
```

The “-it” option means that the container runs in interactive mode, “-d” means the container will run in background, and “-p” means that container’s port 5000 is published to host’s port 5000.

Now you container is running and you can open the Flask web app from your browser. To open the web app just type the address: “127.0.0.1:5000” in your browser.

Recommended tutorial to watch: watch this tutorial video which explains more in details some of the above-mentioned concepts.