Docker: an overview

Data Science Retreat

Jacopo Farina

The problem of the state

You try a library or a different version of Python, and:

- You cannot easily reconstruct what exactly is installed now and communicate it

- Different versions of the same thing are installed, and sometimes one or the other is used

- A newer version of a software X has been released, if you install X on a new machine, it's not the same you used in development, and may break

The problem of the state

You developed a software, (e.g. a web application) depending on a few libraries and maybe requiring databases and frameworks installed on the system

- How can you install it on a new machine and be reasonably sure it will work the same way?

- Can you make it easy for someone else to run it?

  - Or for yourself in the future...

No module named '_gdal' when importing Shapely first #3779

jacopofar commented 29 days ago

hello,
I'm using the latest stable from GDAL and Shapely on macOS, with Python 3.9.
When importing GDAL or Shapely separately they work, but when using both I get an error only when using a specific order.
Seems the same issue as #2957 but on macOS.

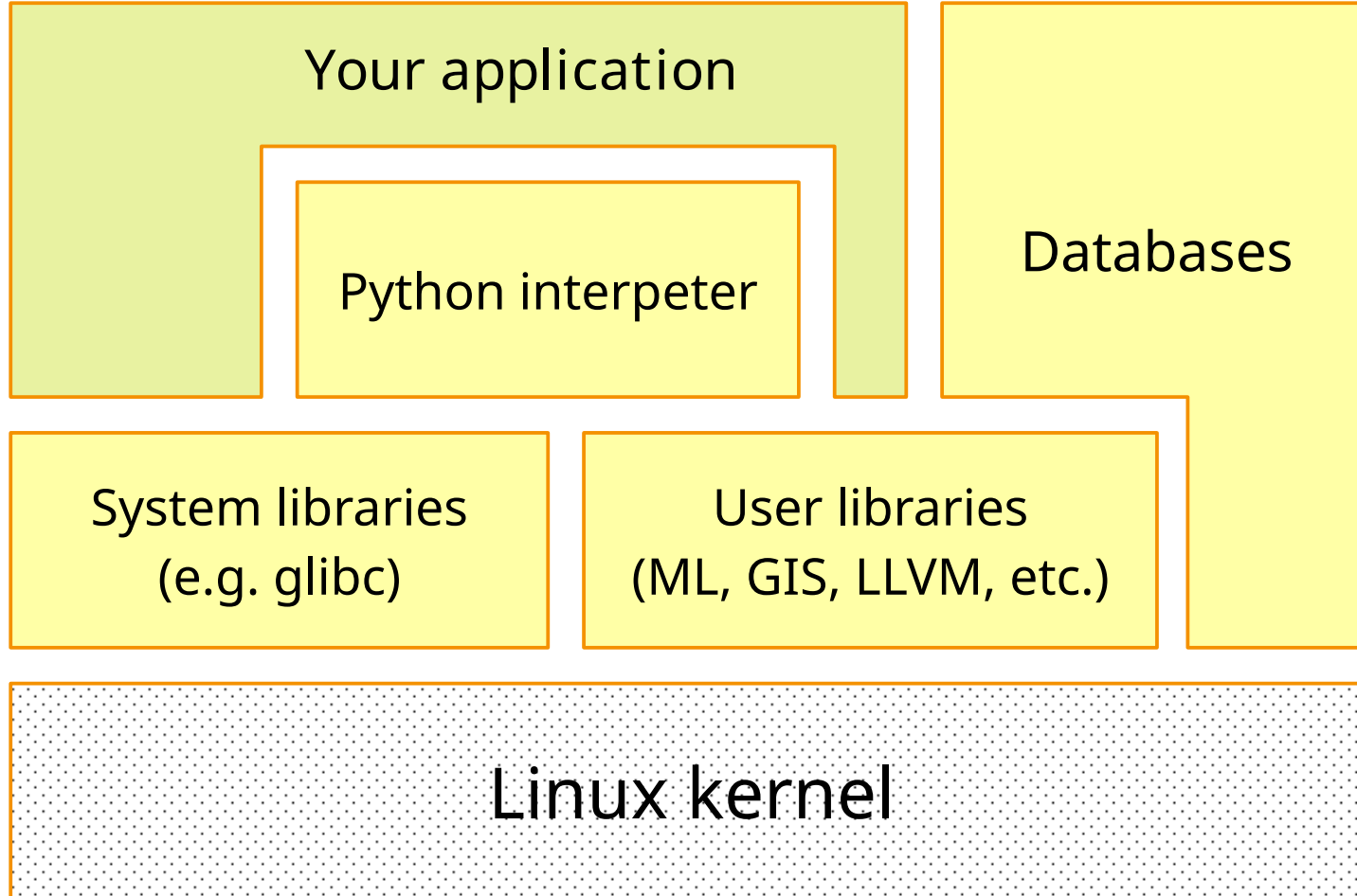## Expected behavior and actual behavior.

I expect to be able to import and usa GDAL and Shapely in any order.

Steps to reproduce the problem.

```
# create the virtualenv
python3 -m venv .venv
source .venv/bin/activate

gdal-config --version
# output: 3.2.2

python3 -m pip install numpy==1.20.2
python3 -m pip install --no-binary=Shapely Shapely==1.7.1
```

What is Docker

- Runs a Linux process in an isolated and controlled way

- Separates applications from Infrastructure

- Can be used during development and/or production

- Allows reproducible build and execution (no „works on my machine")

- Available for macOS and Windows using a virtual machine

- (There is also a native Docker for Windows, but is not that common)

Daemon and client

- The daemon (dockerd) is the process that manages all docker resources
    - Also known as *engine*
    - On macOS and Windows, it runs in a Linux VM
- The Docker client (docker) is the primary way that Docker users interact with the Daemon
    - Can run directly on non-Linux system
    - Communicate with the engine via HTTP
- In our case, the daemon and the client will be on the same machine: our laptops!

Images and containers

• An image is a template to create containers to run. Most importantly, it contains all the files visible to the running app (including libraries and intepreters)

• Public images are on *hub.docker.com*, including official ones

• A container is a running image, you can have multiple containers from the same image

• Every container is based on one and only one image, but an image can be executed in many different containers
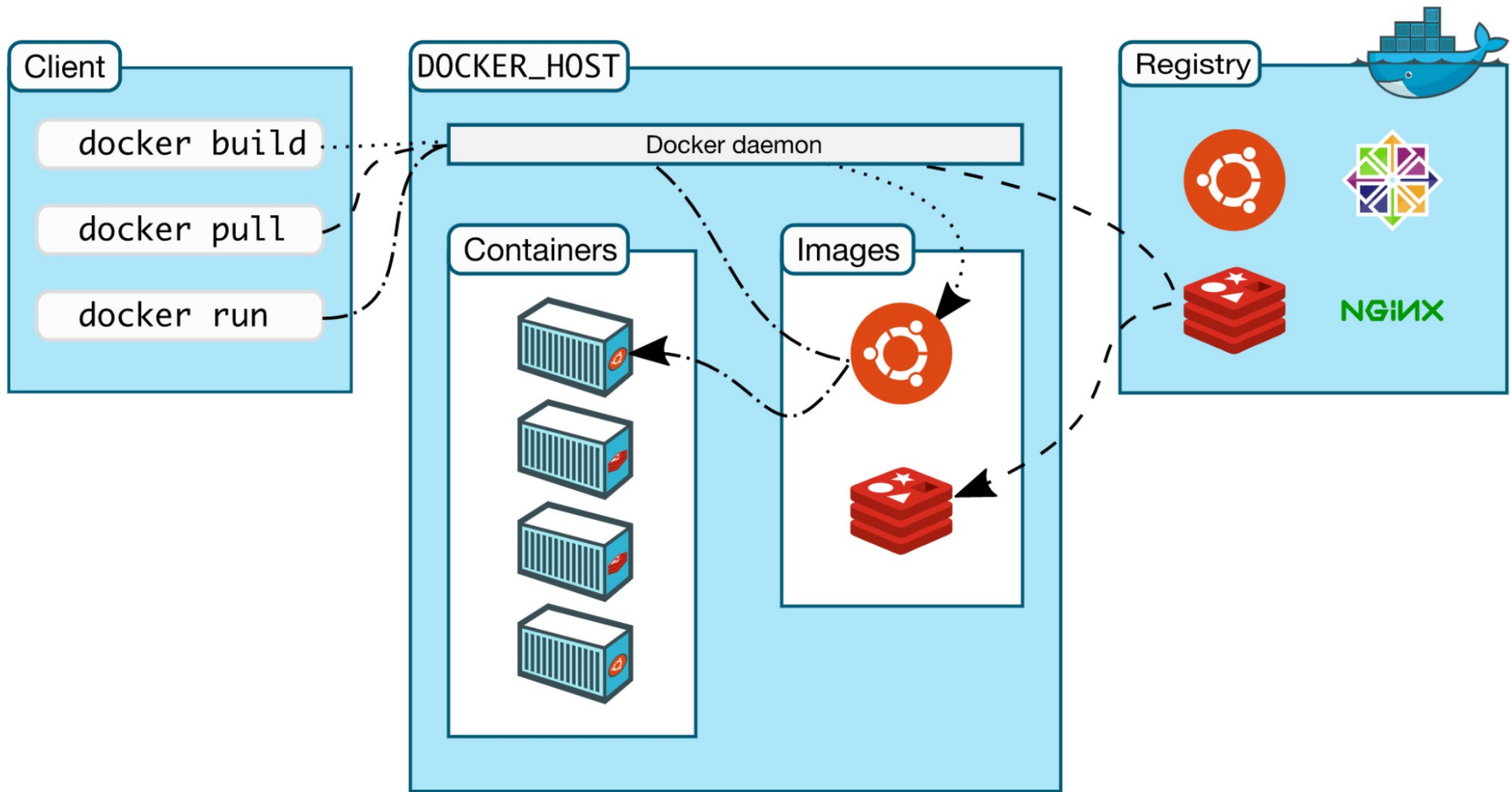
DockerCon Live is coming this May 27th! Sign up to learn more.

docker hub 🔍 Search for great content (e.g., mysql)

Explore   Pricing   Sign In   **Sign Up**

Explore > ubuntu

ubuntu ☆

Docker Official Images

Ubuntu is a Debian-based Linux operating system based on free software.

⬇ **1B+**

Container   Linux   IBM Z   PowerPC 64 LE   x86-64   ARM   386   ARM 64   Base Images   Operating Systems

Official Image

Copy and paste to pull this image

```
docker pull ubuntu
```

View Available Tags

Description   Reviews   Tags

Quick reference

Google Chrome

Example

Let's say we want to check how this code behaves in different Python versions:

```python
print(f'this is {3+2}')
```

installing Python 3.5 and 3.9 on the same machine can be messy.

It's a good use case for Docker

Step 0: find the image details

We need to find the image name and tag on the docker hub.

The complete name of an image is

{docker hub username}/{image}:{tag}

- If the tag is omitted, it is „latest" by default
- When an image is official, you can omit the username

Step 1: download the image

Use:

- docker pull to download an image

- docker images to list the downloaded images

- docker rmi to delete an image

Notice that images are versioned and have an hash.

Step 2: run a container

Use:

- docker run to start a container from an image
  - -it to run it in interactive mode
  - --rm to delete upon exit
  - --name to give it a name (if not it uses a random one)
- docker ps to see the running containers
- docker stop / kill / rm to delete the containers
- docker inspect to see all the details (extremely useful)
- docker rm $(docker ps -aq) can delete all stopped containers to save space

Exercise: play with the FS

- Create a container with:

    docker run -it --name mypython python:3.9

- Use docker ps in another window to see the running container

- Use docker exec to start bash in this container, and write to a file

  - e.g. docker exec -it mypython bash

- Can you read the file from python?

- What happens if you start another container, is the file there?

- Use docker ps -a to see stopped containers, docker rm to eliminate them

The filesystem

- From „inside" a container, you see the filesystem from the image

- A container can write files, but changes are only for that container and are ephemeral

  - This is by design, to make every process reproducible

- If you need to configure/install components not provided by the image, you need to create your own image (we are going to see this)

- However, you can mount volumes to process data from the outside

See the output

We use -it to create an interactive terminal, so that we can read and write on the standard input/output

We use instead -d to detach it and run in background

It is still possible to read the output using docker logs.

Useful: docker logs -f, (like tail -f for files), shows the live log (f is for follow)

Environment variables

A container does not see the system environment variables

- It receives the environment variables defined in the image

- Further environment variables can be passed using the -e flag when running a container

  - This is the recommended way to pass the configuration (tokens, connection strings, etc.)

Networking

A container is isolated also in terms of networking

- A server running in a container is not visible on the host, unless you pass the -p flag to expose a specific port.

  - A container by default has its own IP address.

- You can also expose services on different host ports (e.g.: -p 3000:80), useful to run different instances of the same type of service at the same time

- You can also keep a container offline, create networks of containers and much more

Exercise: run a database

- Find the image for Postgres, see the instructions to run it

- It will run in the background (becase of the -d flag)

- Run a client inside the same container using docker exec

  - Doing so requires no password, only the user (psql -U postgres)

- Try connecting from another postgres container using the link function

Network - linking

Often your app requires services (e.g. a database), and you want to connect to a container from another.

In this case, the --link flag can help:

docker run --name somemysql \

  -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mariadb:latest

docker run -it --link somemysql python:3.9

Now the second container can use „somemysql" as an hostname to connect to the first

Docker compose

When you need to start multiple containers together, a tool called docker compose can help.

It uses a YAML file to get the definition of the containers to run, and how to link them.

In short it replaces a list of long commands with a config file.

It also takes care of aggregating the logs and shut down the containers in a single step.

https://docs.docker.com/compose/

Kubernetes

Companies with a complex setup often use kubernetes to manage a cluster and the deployment on it.

It's based on YAML definitions like docker compose, it deploys one or more container on one or more machines, the user only needs to specify what they want, not the implementation details.

Relatively easy to use, but quite hard to maintain. Handles:

- Restarting and health monitoring

- Load balancing

- Cronjobs

- Services

- Node eviction / scaling

It's possible to try it using minikube

Building an image

So far, we used images from the docker hub. There are many for the most common open source tools and libraries, but what if you want to create one with your own code?

That way, you can allow the user (or yourself) to run it using a single command, without worrying about they having different versions of the dependencies.

It can also be useful during development, to quickly build and test some code in a reproducible way.

Building an image

Docker allows you to build an image, that can be then published on the Docker Hub or private registries.

It's possible to snapshot a container FS to get an image, but not recommended because it's much harder to maintain and rebuild.

Instead, the traditional way is to define a Dockerfile.

This file is basically a recipe to build an image, a list of steps which are executed in sequence

Dockerfile: an example

```
1    FROM ubuntu:20.04
2
3    RUN apt-get -y update && \
4        apt-get -y install wget
5
6    COPY hello.sh /opt/hello.sh
7
8    RUN chmod +x /opt/hello.sh
9
10   CMD ["bash", "/opt/hello.sh"]
11
```

Building an image

Exercise: Try to build a simple image yourself

Three tips from experience:

- Put the dockerfile in the root of your project, or the code subfolder if any

- Use a .dockerignore file to ignore files, same logic and syntax as .gitignore

- Use intermediate images to troubleshoot the build process

  - NOTE: docker shows the id of each intermediate image if you build with the environment variable DOCKER_BUILDKIT=0

Exercise: dockerize a Flask app

Let's create a small Flask app that generates a Numpy array and return the median from a GET endpoint, and then dockerize it.

First let's focus on having the app run outside Docker.

Suggestion: create a new folder for this, with this structure to avoid confusion
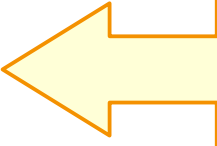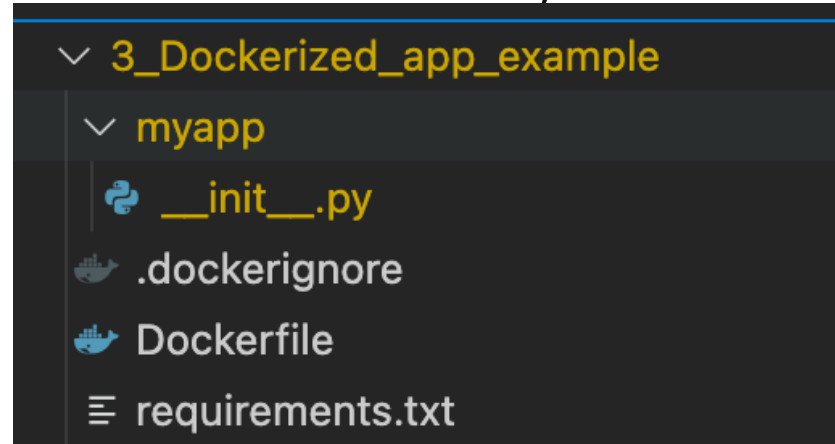
Exercise: dockerize a Flask app

You can write your own logic, here's a minimal example

```python
1   import numpy as np
2   from flask import Flask
3
4   app = Flask(__name__)
5
6   @app.route('/')
7   def random_median():
8       a = np.random.random(12)
9       return str(np.median(a))
```

Exercise: dockerize a Flask app

To run it (without Docker) you should:

- Create a virtualenv with python3 -m venv .venv

- Activate it with source .venv/bin/activate

- Install flask and numpy with pip

- Write the dependencies in the *requirements.txt* to pin them (e.g. *numpy==1.20.3*)

- Run it with FLASK_APP=myapp:app flask run --host 0.0.0.0

- Visit http://localhost:5000 to see the result

Or use Conda, if you prefer

Exercise: dockerize a Flask app

Now we can replicate the setup in a Dockerfile, to build an image with all the logic.

Try to write the Dockerfile yourself

Tips:

- For a starting image you can use python:3.9

- You don't need the virtualenv, install on the whole container because the isolation is already provided by Docker. Use the requirements.txt

- You can run an arbitrary command with docker run, not just the default one specified in the Dockerfile CMD. Useful to run bash and get a shell for troubleshooting

- The --host flag for Flask is not always needed. Use docker exec and then curl to call the app from inside the container

- Bonus: try to use *gunicorn* to run the app, once you can run it using only flask.