Python 101

Structuring a python project, Dependency management using virtualenv and docker

Contents

Laying out a Python Project

According to **Kenneth Reitz** (*developer of Requests and PSF Member*)

- ➔ README.rst
- ➔ LICESE
- ➔ MANIFEST.in
- ➔ setup.py
- ➔ requirements
- ➔ project_package
- ➔ tests_package
- ➔ docs_package

Laying out a Python Project extended

If your project involves using C extensions as well.Seperate the C and Python code by putting C code in a folder called **/lib** and python code in a folder called **/src**

- ➔ README.rst
- ➔ LICENSE
- ➔ MANIFEST.in
- ➔ lib (.c and .h files)
- ➔ src (modules and packages)
- ➔ test_package
- ➔ docs_package

Laying out a Python Project - The Right Way

contd..

https://github.com/requests/requests

For reference see requests package by Kenneth Reitz.

Dependency Management

When we are developing Python Projects more often than not we will end up using package that are not there in our python's standard library.

The packages we use are called dependencies of our project. Our project should have least possible dependencies as having many dependencies makes our project prone to errors.

In order to make our project interoperable we need to ship in the libraries that we have used along with our project.

Virtualenv

It is a tool to create **isolated python environments**.

A Python environment means a set of available libraries.

Do **sudo apt install virtualenv**

Create a new environment named ENV - virtualenv ENV

By default it creates environment for Python2.7 To create a Python3.6 environment do -

virtualenv --python=python3.6 ENV

Virtualenv removing an environment

Removing an environment.

When we create an environment a corresponding file is created at home in linux. Just remove that file.

rm -r ENV_NAME

Virtualenv activating and deactivating environments

source is a utility in linux that is used in order to execute commands from a bash file.

There exists a file named activate in each environment's bin folder which contains code for activating that environment. So we can activate an environment using source command as-

~$ source activate ENV/bin/activate

The result is **(ENV) ~$** ← which represents the current environment is ENV.

In order to deactivate an environment just type **deactivate**

pip freeze and pip install -r

pip freeze outputs the names and versions of the external packages that are installed in current environment.

We can use the output generated by this in order to keep track of **requirements** of our project's environment in a file named requirements.txt

In order to install all of the external packages written inside the requirements file we can use **pip3 install -r requirements.txt** here -r stands for requirements file ie. install packages given in requirements file named requirements.txt.

Docker - A primer

Docker is a containerization tool. It bundles all of the requirements of our projects irrespective of what the requirements are (ie. not only python specific, eg. MongoDB might be a dependency of our project) into a container which can run on any system.

Thus it makes dependency management in a project much easier.

Containers - A container is a lightweight standalone, stateless, executable package of a piece of software that includes everything needed to run it.

Stateless meaning that none of the data in a container persists. It wipes it all each time it restarts.

Docker - A primer contd..

Containers vs Virtual Machines - A VM runs on a system much like a container does. But the VM is not specific to a piece of software. Its literally a whole new system within a system. VMs are generally hard to setup and much much harder to manage upon that due to mismanaged Hypervisors the resource usage is also significantly higher than containers.

CaaS or Containers as a service thus became an alternative due to relatively harder management of VMs.

So instead of bothering about installing everything separately we need to just install one thing that is Docker. The rest of it will be managed by Docker.

Docker - A primer contd..

sudo apt install docker.io ← Install docker

As docker needs access to system resources in order to prepare containers it needs superuser priviledges. So we need to run each docker command as **sudo.** But an alternative approach is to add our user to the docker usergroup. Where docker usergroup has the required permissions for functioning docker.

sudo usermod -aG docker ${USER} ← Add user to docker

Further we will learn how to, **create, run, stop, attach and build** a container.

Docker - A primer contd..

There are a lot of containers people have built already and have stored on DockerHub much like people have created packages and stored in on PyPI.

We can use those containers on our system using **create.**

- **docker create -it ubuntu bash** ← it means create a ubuntu container and open an interface to it and run bash on the interface.
- **docker ps -a** ← see history of containers that have run or are running.

Let the id of created container be 12345678

- **docker start 12345678** ← run the container with id 12345678

Docker - A primer contd..

- **docker attach 12345678** ← attach our terminal's io to the io of container with id 12345678. Now using this terminal we can run any command inside the container.
- **docker rm 12345678** ← remove the container with id 12345678

Volumes in docker - As we know containers are stateless ie. they wipe data on each run. What if we needed some persisting data? Its quite natural that we might. Like what if the container is of a Database? We will need some persisting data in that case.

Docker - A primer contd..

We know that we can persist data on our base system by creating files and directories.

What if we could map a directory to a directory inside the container? This way all the content on our base system inside that directory could be accessed by container which would persist as the content actually resides on our base system instead of the containers. This is achieved using volumes.

- **docker create -it -v ~/some_directory_base:/some_directory ubuntu bash** ← create a container with ubuntu and start bash in it and map the some_directory_base at home to some_directory at root in container.

Docker - A primer contd..

We know that we can persist data on our base system by creating files and directories.

What if we could map a directory to a directory inside the container? This way all the content on our base system inside that directory could be accessed by container which would persist as the content actually resides on our base system instead of the containers. This is achieved using volumes.

- **docker create -it -v ~/some_directory_base:/some_directory ubuntu bash** ← create a container with ubuntu and start bash in it and map the some_directory_base at home to some_directory at root in container.

Docker - A primer contd..

Mapping ports - Every service runs on a port in our systems. Eg. 443 for https, 80 for http, 27017 for MongoDB, 6379 for Redis, 5432 for PostgreSQL, and 3306 for MySQL etc.

So if a service is running inside our container with which we want to communicate then we need to be able to map the ports of that container to the ports of our base system to be able to use those services on our base systems.

We can achieve this using this command -

- **docker create -it -p port_of_base:port_of_container ubuntu bash**

Docker - A primer contd..

Last thing left here is that if we want to just give our application to someone we can not. Because we will have to tell the person create a container do this and that. What if we could automate even that process ?

We use **Dockerfile** for this purpose. Dockerfile defines what steps to perform in order to build an image.

Docker - A primer contd..

FROM base_image_to_use

MAINTAINER “Name of the creator”

RUN commands_to_run_on_creation

VOLUME volume_to_mount_from_base

EXPOSE port_to_expose

WORKDIR directory_path_in_which_to_work_on_startup_of_container

CMD command_to_run_on_container_startup

Docker - A primer contd..

```
FROM ubuntu

VOLUME /content

RUN apt update

RUN apt install -y python3

EXPOSE 8000

WORKDIR /content

CMD python3 -m http.server
```

Docker - A primer contd..

Building the container image using the dockerfile-

- **docker build . -t demo_image:version1** ← Build a docker image using the Dockerfile available in current directory with the name as demo_image and tag as version1.
- **docker create -it -v ~/content_to_host:/content -p 8000:8000 demo_image:version1** ← Create a container based on image docker_image:version1 and and map content_to_host folder on base to it's content folder and map the port 8000 of base to its port 8000

Docker - A primer contd..

Building the container image using the dockerfile-

- **docker build . -t demo_image:version1** ← Build a docker image using the Dockerfile available in current directory with the name as demo_image and tag as version1.
- **docker images** ← list all of the locally available images
- **docker rmi 12345678** ← remove the image with image id as **12345678**
- **docker create -it -v ~/content_to_host:/content -p 8000:8000 demo_image:version1** ← Create a container based on image docker_image:version1 and and map content_to_host folder on base to it's content folder and map the port 8000 of base to its port 8000

Docker - A primer contd..

A practical of installing and using mongodb within seconds (depending on internet connection ofcourse).

https://hub.docker.com/_/mongo/

docker create --name local_mongo -v ~/datadir:/data/db -p 27017:27017 -d mongo ← Save data in /datadir map port of mongodb to 27017.

load on browser http://localhost:27017 it says -

It looks like you are trying to access MongoDB over HTTP on the native driver port. ← it means Mongodb is accessible and running successfully.

Docker - A primer interacting with mongodb using python

We just saw the error when we tried to access the mongodb through http. It said It looks like you are trying to access MongoDB over HTTP on the **native driver** port. So we need some sort of native driver to access mongodb. PyMongo is that native driver

We can install PyMongo using following command-

pip3 install pymongo

Docker - A primer interacting with mongodb using python

```
from pymongo import MongoClient

conn = MongoClient('localhost', 27017)

db = conn.test_database ← if it doesn't exist then it will be created.

col = db.test_collection ← if it doesn't exist then it will be created as well.

col.insert_one({"key": "value", "k2": "v2"})

col.find_one({"key": "value"})
```