Python Package Manager

- The Python Packaging Authority (PyPA) is a working group that maintains many of the relevant projects in Python packaging (ex: pip).
- *pip* is a recursive acronym and stands for ‘Pip Installs Packages’ or ‘Pip Installs Python’;
- The Python Package Index (PyPI) is a repository of software for the Python programming language;
- pip is a tool for installing Python packages from PyPI.
- pip is already installed if you are using Python 2 >=2.7.9 or Python 3 >=3.4, or if you are working in a Virtual Environment created by virtualenv or pyvenv.

pip

- To install pip:

```
$ curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
$ python get-pip.py
```

- To install packages:

```
$ pip install Package-name
$ pip install Package-name-1.0-py2.py3-none-any.whl
$ pip show --files Package-name
$ pip list –outdated
$ pip install --upgrade Package-name
$ pip uninstall Package-name
```

- To remove pip:

```
$ python -m pip uninstall pip setuptools
```

pip

- Potential problems:
  - There isn't coordination between different package managers.
- More details:
  - https://pip.pypa.io/en/stable/user_guide/
    - On Unix the default configuration file is: $HOME/.config/pip/pip.conf which respects the XDG_CONFIG_HOME environment variable.

pip

- Potential problems:
  - There isn't coordination between different package managers.
- More details:
  - https://pip.pypa.io/en/stable/user_guide/
    - On Unix the default configuration file is: $HOME/.config/pip/pip.conf which respects the XDG_CONFIG_HOME environment variable.

Virtual Environments

- *virtualenv* is a tool to create isolated Python environments
  - It creates a folder which contains all the necessary executables to use the packages that a Python project would need.

```
$ pip install virtualenv
$ virtualenv –version
```

- Create a virtual environment for a project:

```
$ cd my_project_folder
$ virtualenv my_project
```

- Configuring Python version

```
$ virtualenv -p /usr/bin/python2.7 my_project
$ export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python2.7 (in ~./bashrc)
```

Virtual Environments

- it needs to be activated:

```
$ source my_project/bin/activate
$ deactivate
```

  -

- To remove a virtualenv:
  - Just delete its folder.

```
$ rm -rf my_project
```

Virtual Environments

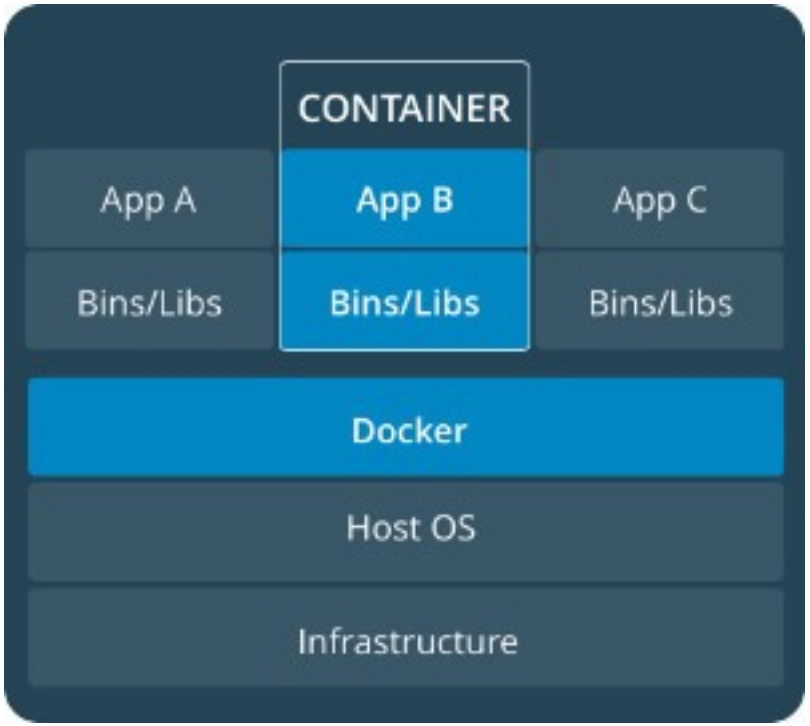
- Exam-1:
  - Install the pip and virtualenv tools.
  - Install the package scikit-image (isolated repository).
  - Compare both lists of pip tool (isolated repository vs global repository).

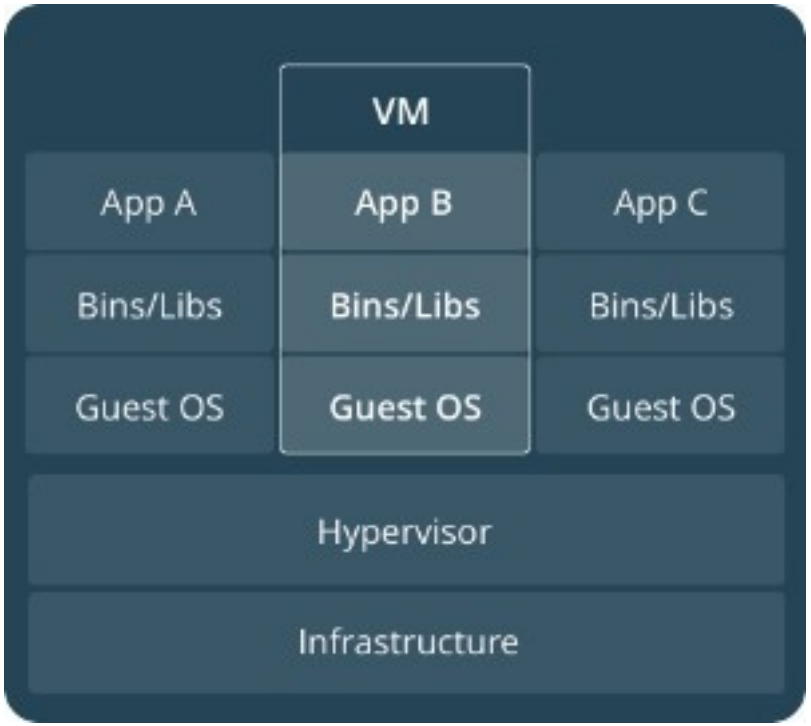
Container

- A **Docker** container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings
- Docker elements:
  - Dockerfile Defines what goes on in the environment inside your container:
  - Image: All environment already built;
  - Container: Images become containers when they run on Docker Engine.
- A registry is a collection of repositories, and a repository is a collection of images — sort of like a GitHub repository, except the code is already built.
  - An account on a registry can create many repositories (ex: Docker's public registry).

Container

- Docker:
  - It runs a discrete process.
  - It runs natively on Linux and shares the kernel of the host machine with other containers.





Container

- Set up the system:

```
$ sudo apt-get update
```

  - Install packages to allow apt to use a repository over HTTPS:

```
$ sudo apt-get install apt-transport-https ca-certificates curl \
  software-properties-common
```

  - Add Docker’s official GPG key:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

  - Set up the stable repository:

```
$ sudo add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable"
```

Container

- Install docker-CE:

```
$ sudo apt-get update
$ sudo apt-get install docker-ce
```

- The Docker daemon starts automatically:

```
$ sudo docker run hello-world
```

- Uninstall the Docker CE package:

```
$ sudo apt-get purge docker-ce
```

  - To delete all images, containers, and volumes:

```
$ sudo rm -rf /var/lib/docker
```

Container

- Dockerfile defines what goes on in the environment inside your container:
  - Applications, network interfaces, hard disk drives.

```
# Use an official Python runtime as a parent image
FROM python:2.7-slim
# Set the working directory to /app
WORKDIR /app
# Copy the current directory contents into the container at /app
ADD . /app
# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt
# Make port 80 available to the world outside this container
EXPOSE 80
# Define environment variable
ENV NAME World
# Run app.py when the container launches
CMD ["python", "app.py"]
```

Container

- Build an image:

```
$ docker build -t my-name .
```

- Start a container:

```
$ docker run -p 4000:80 my-image

$ docker run -d -p 4000:80 my-image
```

- Others parameter:

```
$ docker run --runtime=nvidia -e NVIDIA_VISIBLE_DEVICES=3 --cpus=5
--device=/dev/nvidia3
```

- Working with volume:

```
VOLUME ["/data"]
```

Container

- Exam-2:
  - Make a Dockerfile of example, build it and start a new container.

Container

- Exam-3:
  - Make Dockerfile with: Python3, pip, jupyter, matplotlib and pandas.
  - Configure a shared folder.
  - Make a bulid stage and start a container.