Containers

for

Scientific Computing and Data Science

Tommaso Buvoli

September 29, 2020

Common Software Problems



Common Software Problems

Each project may require many software dependencies.

Common Software Problems

Each project may require many software dependencies.

- No reproducibility across machines or across time.

Common Software Problems

Each project may require many software dependencies.

- No reproducibility across machines or across time.
- Packages end up littering your hard drive for years even after you've completed a project.

Common Software Problems

Each project may require many software dependencies.

- No reproducibility across machines or across time.
- Packages end up littering your hard drive for years even after you've completed a project.
- Certain software can also be difficult to install.

Why Containers?

Why Containers?

1. **Containers are reproducible.**

Why Containers?

1. **Containers are reproducible**.

    - You can save a compute environment and replicate it on any machine at a later date.

Why Containers?

1. **Containers are reproducible**.

    - You can save a compute environment and replicate it on any machine at a later date.
    - You can share containers across teams.

Why Containers?

1. **Containers are reproducible**.

    - You can save a compute environment and replicate it on any machine at a later date.
    - You can share containers across teams.

2. **Containers bundle your software dependencies.**

Why Containers?

1. **Containers are reproducible**.

    - You can save a compute environment and replicate it on any machine at a later date.
    - You can share containers across teams.

2. **Containers bundle your software dependencies.**

    - No clutter on your local machine.

Why Containers?

1. **Containers are reproducible**.

    - You can save a compute environment and replicate it on any machine at a later date.
    - You can share containers across teams.

2. **Containers bundle your software dependencies.**

    - No clutter on your local machine.
    - Save your containers online and reuse them at a later date.

A Computer in a Computer

A Computer in a Computer

- Containers let you run a “*computer in a computer.*”

A Computer in a Computer

- Containers let you run a “*computer in a computer.*”
- Similar in spirit to virtual machines.

Container Basics

Container Basics

1. A **container image** describes the Linux operating system and the software installed in it.

Container Basics

1. A **container image** describes the Linux operating system and the software installed in it.
   - You can think of an image as a description for a virtual computer.

Container Basics

1. A **container image** describes the Linux operating system and the software installed in it.
    - You can think of an image as a description for a virtual computer.

2. A **container** is a collection of processes that run from an image.

Container Basics

1. A **container image** describes the Linux operating system and the software installed in it.

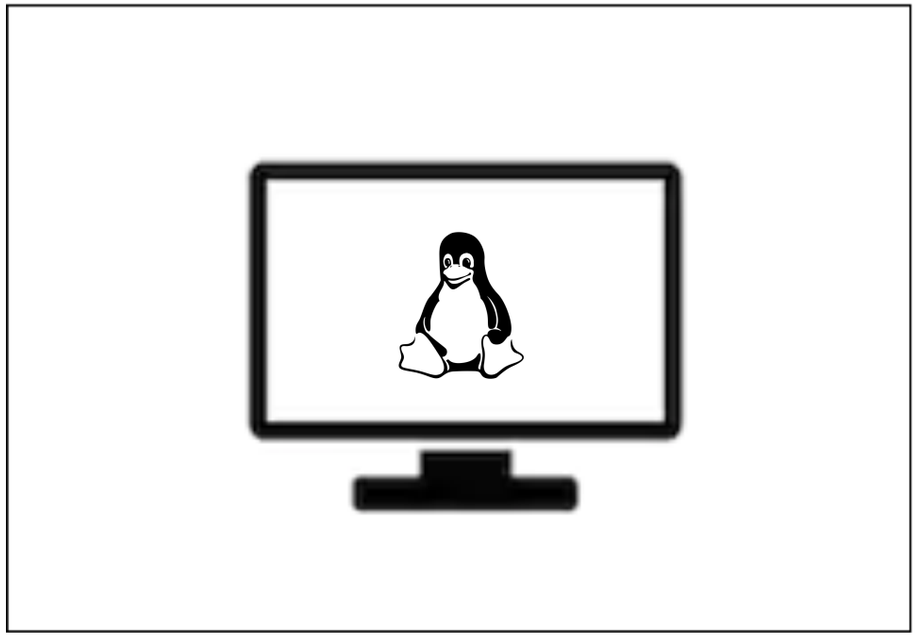
    - You can think of an image as a description for a virtual computer.

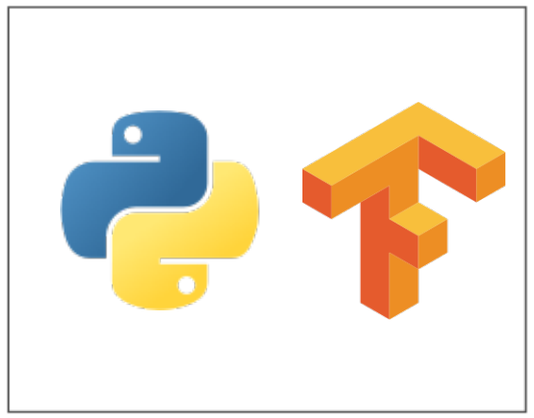
2. A **container** is a collection of processes that run from an image.

    - Multiple containers can be run from the same image.

Containers

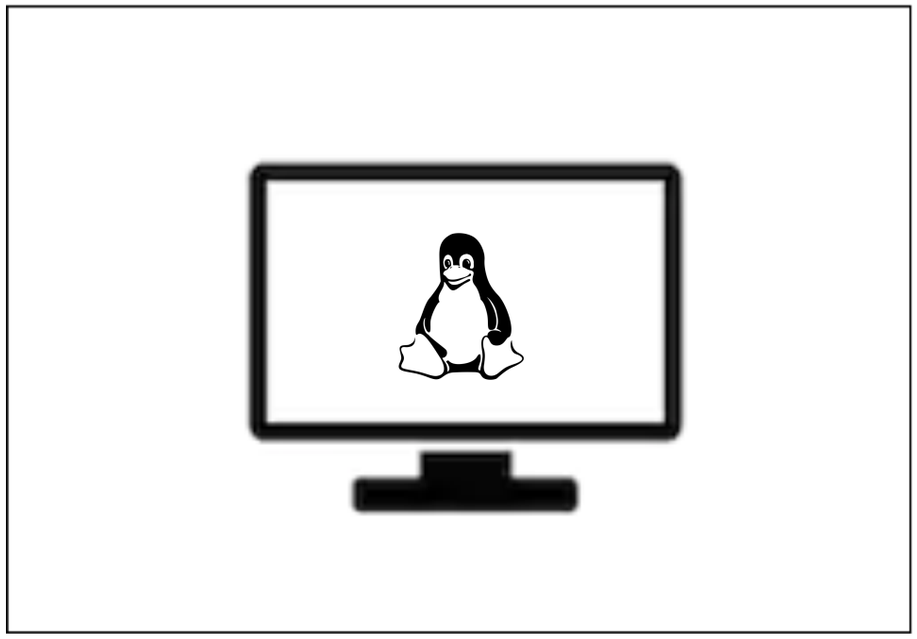
Image

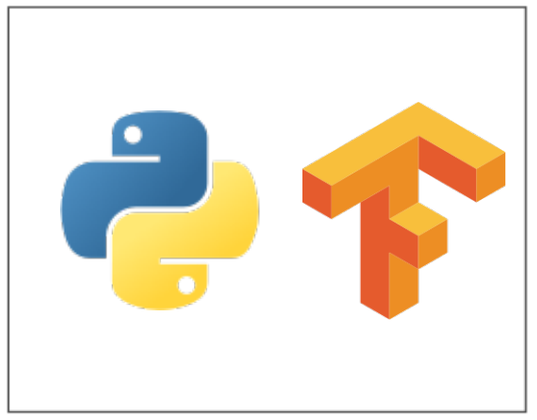




Containers

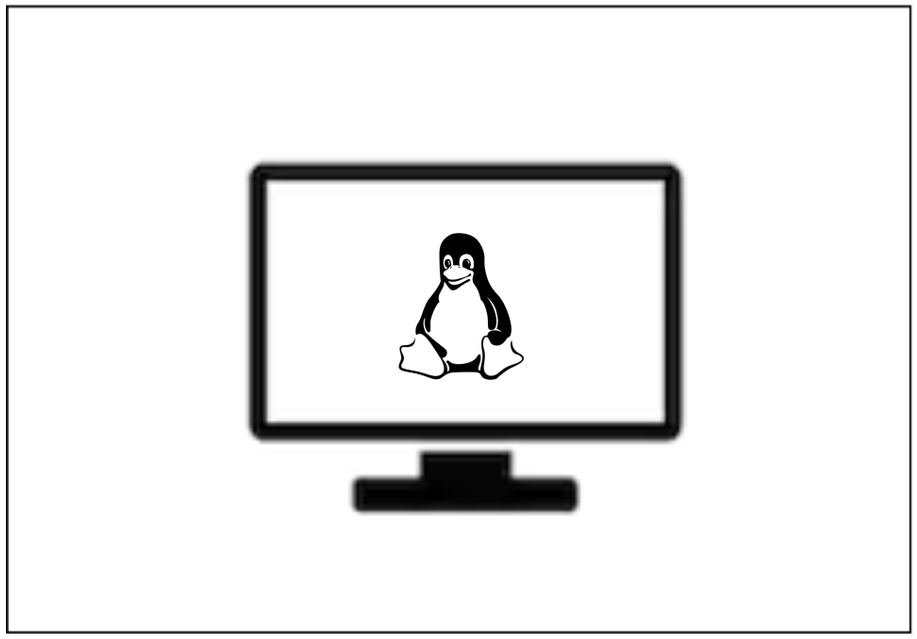
Image





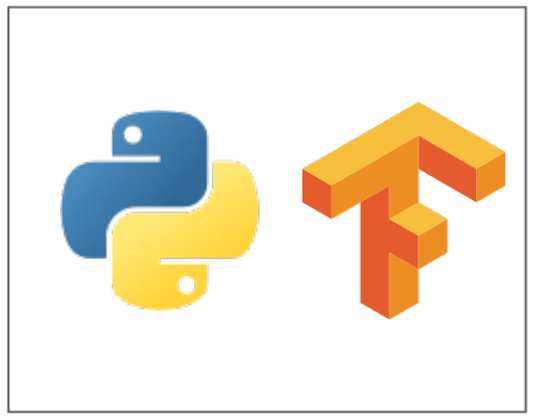
Remark: images are labeled as name:tag (e.g. myimage:v10)

Image



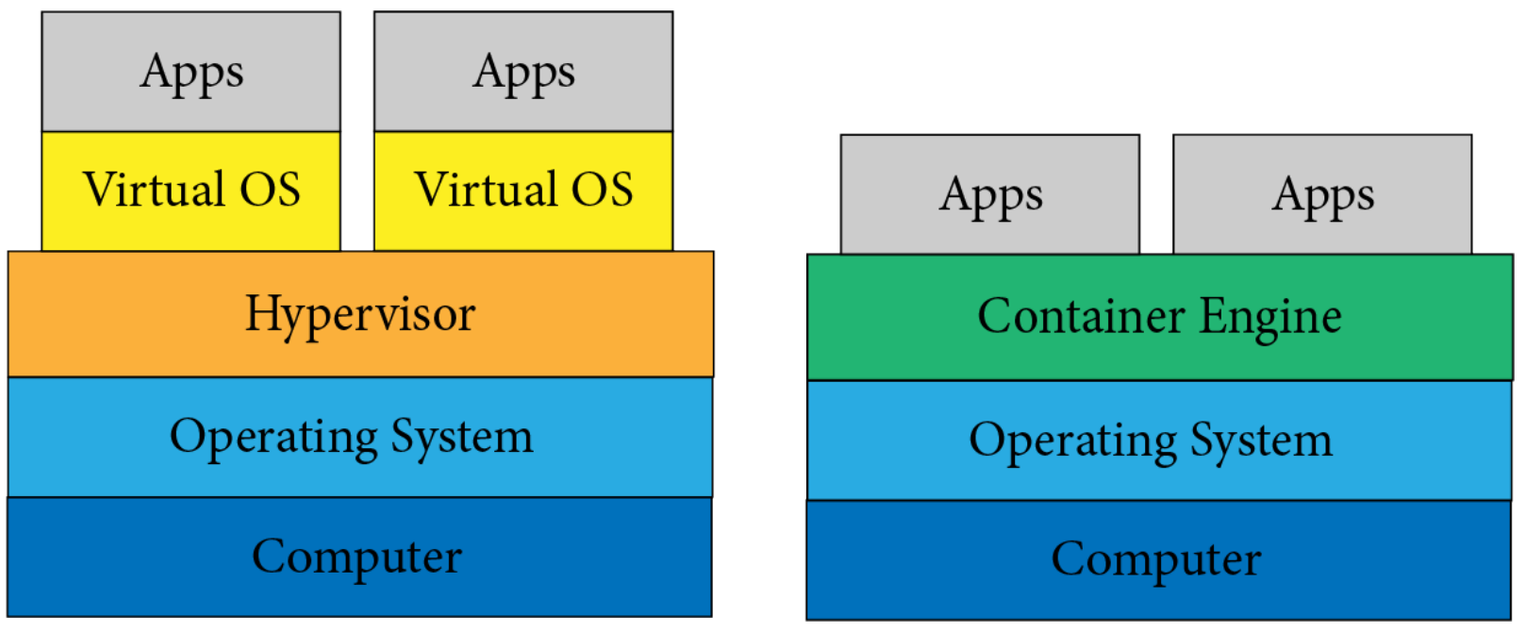
Remark: images are labeled as name:tag (e.g. myimage:v10)

Containers



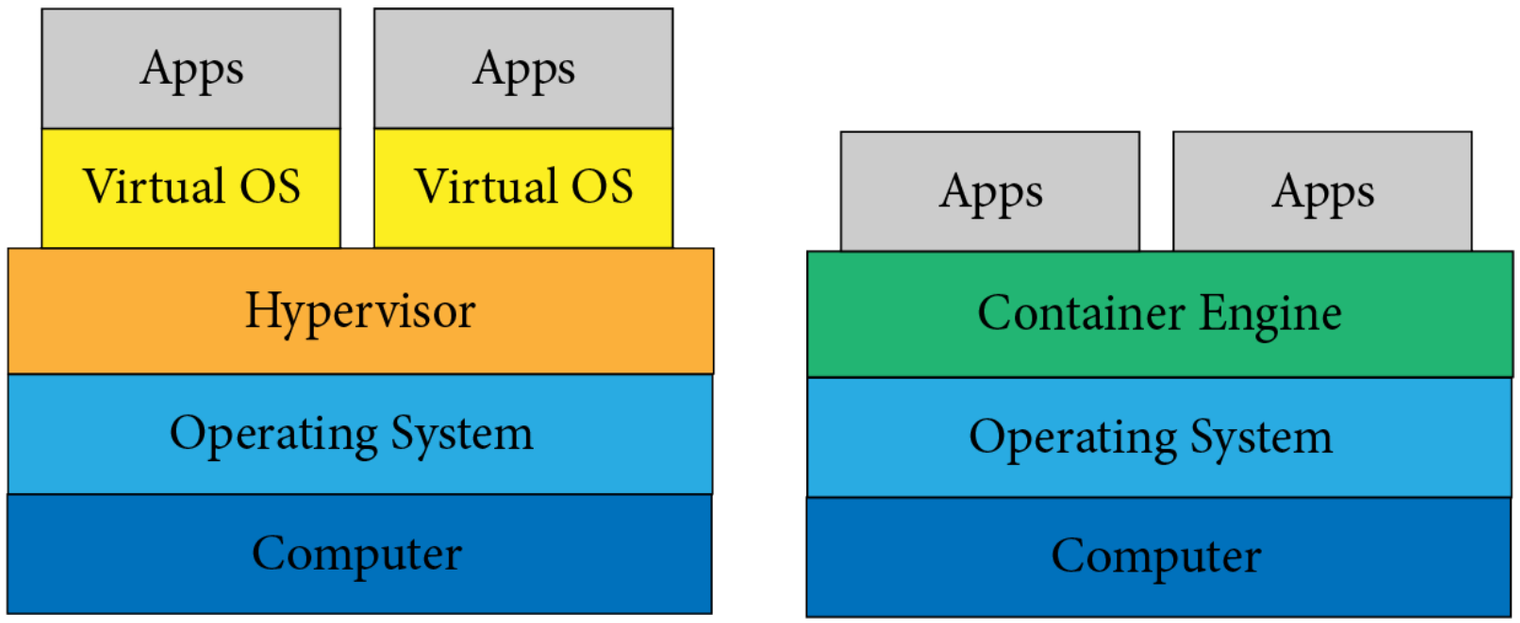
Remark: Containers are ephemeral.The image is never changed.

Containers vs Virtual Machines



Containers vs Virtual Machines

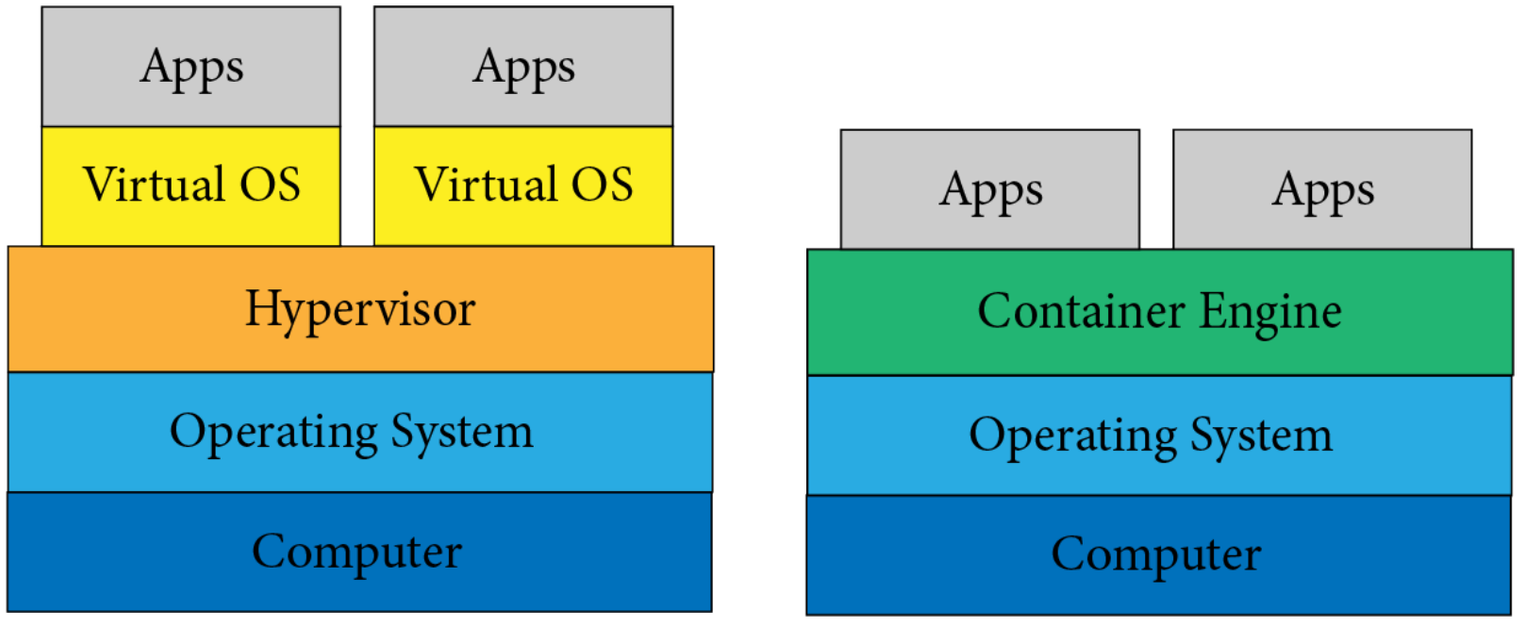
1. Containers are faster to start than virtual machines; no hypervisor, the kernel is shared with host the OS.



Containers vs Virtual Machines

1. Containers are faster to start than virtual machines; no hypervisor, the kernel is shared with host the OS.

2. It is much simpler to share and distribute container images than VM images.



Three Container Engines

Docker is the most popular container engine and runs on Mac, Windows, and Linux.

Podman is a new, open-source container engine that runs on Linux.

Singularity is a container engine that was designed for HPC systems.

Container Orchestrators

Orchestrators manage multiple containers (advanced topic)

Compose is a simple orchestrator that ships with Docker.

Kubernetes is a much more sophisticated orchestrator aimed at deploying complex containerized apps.

Today’s Goals

Today’s Goals

- Learn how to **pull images** from Docker Hub.

Today's Goals

- Learn how to **pull images** from Docker Hub.

- Learn how to **start a container** interactively.

Today's Goals

- Learn how to **pull images** from Docker Hub.

- Learn how to **start a container** interactively.

- Learn how to **build your own container images.**

Pulling Images

Pulling Images

Docker’s official repository of container images:

Pulling Images

Docker’s official repository of container images:

Docker Hub (https://hub.docker.com/)

Pulling Images

Docker’s official repository of container images:

Docker Hub (https://hub.docker.com/)

You can find a variety of images for:

Pulling Images

Docker's official repository of container images:

Docker Hub (https://hub.docker.com/)

You can find a variety of images for:

- **Linux distributions** (e.g. Ubuntu, Fedora, Manjaro, Clear Linux)

Pulling Images

Docker's official repository of container images:

Docker Hub (https://hub.docker.com/)

You can find a variety of images for:

- **Linux distributions** (e.g. Ubuntu, Fedora, Manjaro, Clear Linux)

- **Programming languages** (e.g. Python, Julia, c, C++, Fortran)

Pulling Images

Docker's official repository of container images:

Docker Hub (https://hub.docker.com/)

You can find a variety of images for:

- **Linux distributions** (e.g. Ubuntu, Fedora, Manjaro, Clear Linux)
- **Programming languages** (e.g. Python, Julia, c, C++, Fortran)
- **Software** (Jupyter Lab, Tensor Flow, Paraview)

```
$ docker pull fedora:latest
```

$ docker pull fedora:latest

pulls the image named fedora from docker hub onto your local computer

Starting a Container

Starting a Container

- To start a container use **docker run.**

Starting a Container

- To start a container use **docker run.**
- The format for the command is:

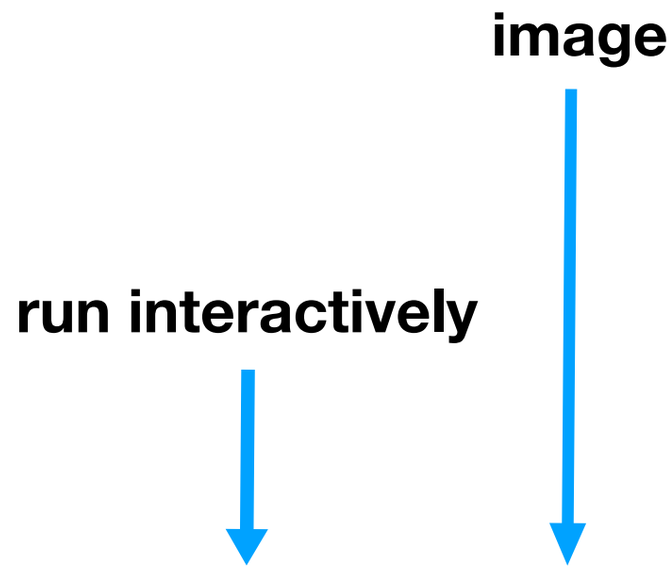
```
$ docker run [FLAGS] image:tag command
```

https://docs.docker.com/engine/reference/run/

$ docker run -ti fedora:latest bash

image

```
$ docker run -ti fedora:latest bash
```




```
$ docker run -ti fedora:latest bash
```

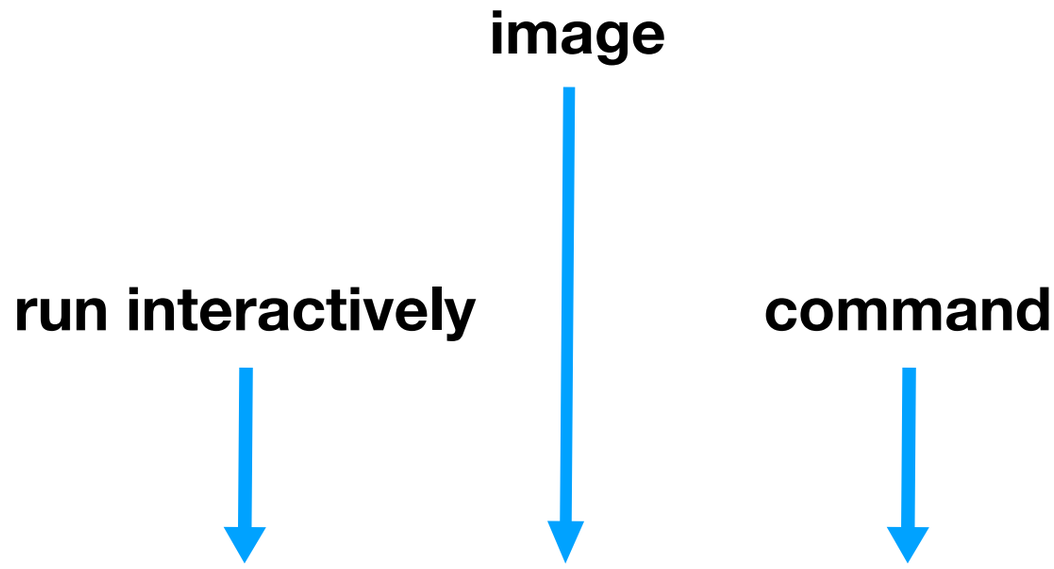
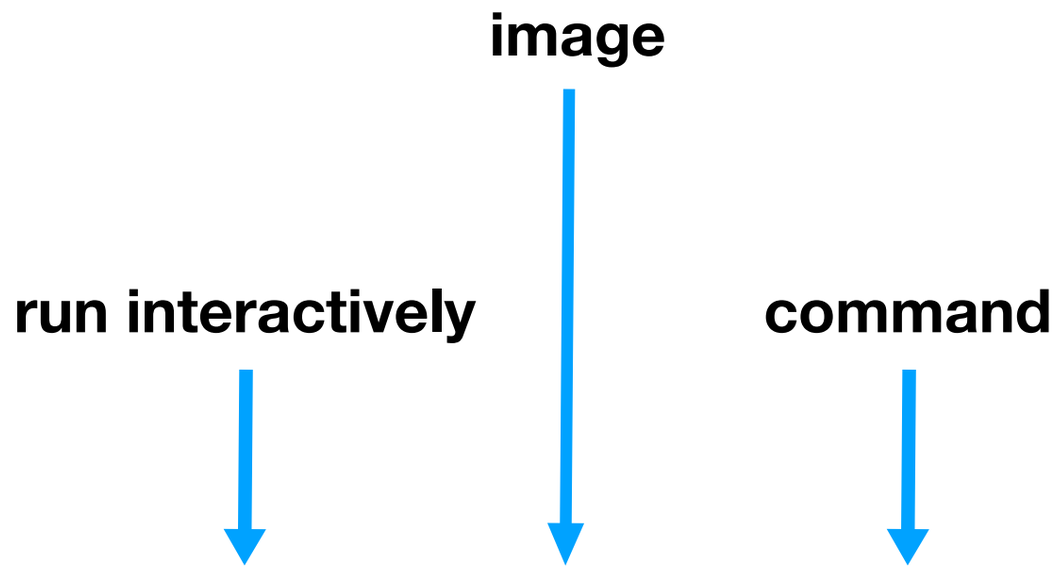
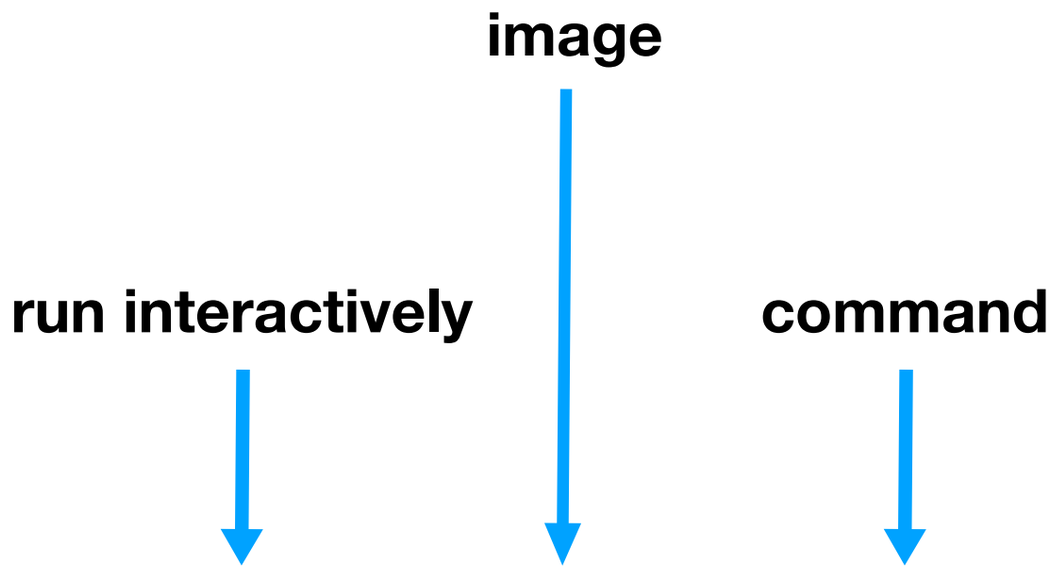


image
run interactively
command
$ docker run -ti fedora:latest bash




```
$ docker run -ti fedora:latest bash
```

runs the command “bash” interactively using the image fedora:latest



$ docker run -ti fedora:latest bash

runs the command “bash” interactively using the image fedora:latest

Let’s verify that containers are ephemeral.

```
$ docker run -ti ubuntu:latest bash
```

It’s easy to try new Linux distributions.

Modifying Local Files

Modifying Local Files

- By default, containers **cannot access** your local files.

Modifying Local Files

- By default, containers **cannot access** your local files.

- We can grant access by creating a bind mount:

```
$ docker run -v /path/on/host:/path/on/container …
```

Modifying Local Files

- By default, containers **cannot access** your local files.

- We can grant access by creating a bind mount:

```
$ docker run -v /path/on/host:/path/on/container …
```

- This mounts the directory */path/on/host* on the **host**
  to */path/on/container* in the **container.**

```
$ docker run -v $(pwd):/files -ti fedora:latest bash
```

binds current directory to /files

```
$ docker run -v $(pwd):/files -ti fedora:latest bash
```

binds current directory to /files

```
$ docker run -v $(pwd):/files -ti fedora:latest bash
```

runs the command “bash” interactively using the image fedora:latest

we now also have access to the current directory

binds current directory to /files

```
$ docker run -v $(pwd):/files -ti fedora:latest bash
```

runs the command “bash” interactively using the image fedora:latest
we now also have access to the current directory

Let’s try creating a file. It will also be on the host now.

Running Scripts

Running Scripts

We can run a script from inside the container interactively:

```
$ docker run -v "$(pwd)":/files -ti fedora:latest bash
[root@c092da88cd6c] $ python3 /files/hello-world.py
```

Running Scripts

We can run a script from inside the container interactively:

```
$ docker run -v "$(pwd)":/files -ti fedora:latest bash
[root@c092da88cd6c] $ python3 /files/hello-world.py
```

We can also run it non-interactively:

Running Scripts

We can run a script from inside the container interactively:

```
$ docker run -v "$(pwd)":/files -ti fedora:latest bash
[root@c092da88cd6c] $ python3 /files/hello-world.py
```

We can also run it non-interactively:

```
docker run -v "$(pwd)":/files -ti fedora:latest python3 /files/hello-world.py
```

Running Scripts

We can run a script from inside the container interactively:

```
$ docker run -v "$(pwd)":/files -ti fedora:latest bash
[root@c092da88cd6c] $ python3 /files/hello-world.py
```

We can also run it non-interactively:

```
docker run -v "$(pwd)":/files -ti fedora:latest python3 /files/hello-world.py
```

Python does not need to be installed on the host!

Building Custom Images

Building Custom Images

- To build a container image use **docker build.**

Building Custom Images

- To build a container image use **docker build.**
- You describe the container in a **Dockerfile.**

Building Custom Images

- To build a container image use **docker build.**
- You describe the container in a **Dockerfile.**
- The format for the command is:

```
$ docker build [FLAGS] -t=image:tag .
```

https://docs.docker.com/engine/reference/build/

Two Build Commands

- `FROM image:tag`
  Allows you to start from an existing container image.

- `RUN command`
  Runs a command during the image build stage.

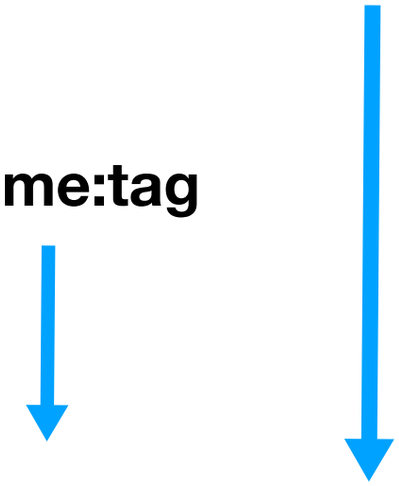
A Very Simple Dockerfile

```
FROM fedora:latest
RUN dnf -y install pip
```

```
# Starts from fedora image
# install pip
```

The build context; must contain the Dockerfile.

The image name:tag



```
$ docker build -t myfedora:pip .
```

this command will build the image specified by the Dockerfile in the cwd

Let’s discuss layers while our image builds.

Layers

Each line in a Dockerfile creates a new *layer*.

If you don't use the `--no-cache` flag when building, then previously computed layers will be reused.

```
FROM fedora:latest                  # starts from fedora image
RUN dnf -y install pip              # install pip
RUN pip install matplotlib          # install matplotlib
```

Let's try running an interactive bash session in our new image.

Additional Useful Docker Commands

Container Management

- Stop a container: `docker stop ID`
- Remove a container: `docker rm ID`
- List running a container: `docker ps`

Image Management

- Push an image to Dockerhub: `docker push image:tag`
- Remove an Image: `docker rmi image:tag`

Documentation

https://docs.docker.com/engine/reference/commandline/docker/

Docker Compose

Encode your settings into a yml file

Docker Compose

Encode your settings into a yml file

- If you use Docker, the command to start an interactive terminal is:

```
$ docker run -ti -v $(pwd):/root/files myfedora:pip bash
```

Docker Compose

Encode your settings into a yml file

- If you use Docker, the command to start an interactive terminal is:

```
$ docker run -ti -v $(pwd):/root/files myfedora:pip bash
```

- This is not a simple command.

Docker Compose

Encode your settings into a yml file

- If you use Docker, the command to start an interactive terminal is:

```
$ docker run -ti -v $(pwd):/root/files myfedora:pip bash
```

- This is not a simple command.
- Using Docker Compose, we can encode this into a file.

Docker Compose

Encode your settings into a yml file

- If you use Docker, the command to start an interactive terminal is:

```
$ docker run -ti -v $(pwd):/root/files myfedora:pip bash
```

- This is not a simple command.
- Using Docker Compose, we can encode this into a file.
- We can run the whole command with `docker-compose run`.

A Simple Compose File

docker-compose.yml

```
version: "3.8"
services:
  bash:
    image: myfedora:pip
    volumes:
       - .:/root/files
    command: bash
```

$ docker-compose run bash

If you find containers interesting…
Here is an even simpler tool

container job runner - a simple way to develop and run code in containers.

Local Development

```
$ cjr shell --here                    # start an interactive shell in local directory
$ cjr jupyter:start                   # start Jupyter Lab or notebook
```

Jobs

```
$ cjr job:start --resource=my-server  python myscript.py      # run remote job
$ cjr job:start --resource=localhost  python myscript.py      # run local job
```

website: https://container-job-runner.github.io/ **github:** https://github.com/container-job-runner/cjr

Thanks for your attention!