Docker

Jitesh Agarwal

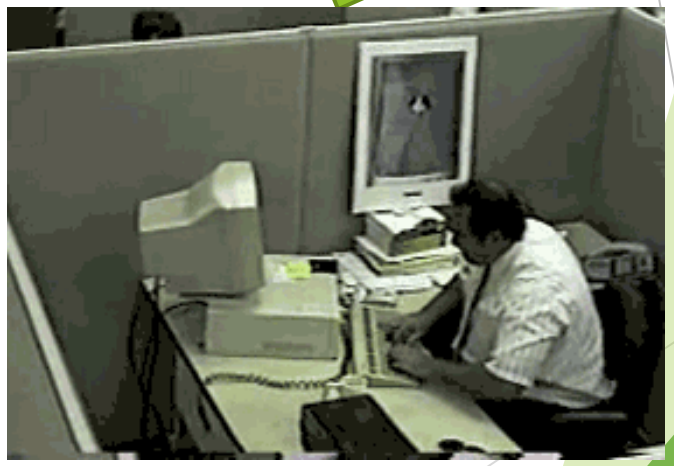
Agenda

- Day - 1
  - Application Architecture
  - Virtualization
  - Containerization
  - Docker
  - Benefits
  - Installation
  - Hands-on
- Day - 2
  - Docker Architecture
  - Docker Registry
  - Docker Engine
  - Docker Images
  - Docker Containers
  - Hands-on
- Day - 3
  - Docker Volume
  - Docker Port
  - Docker Network
  - Dockerfile
  - Docker-compose
  - Hands-on

User Concern

It’s Working for Me……

Why Not for Me????????



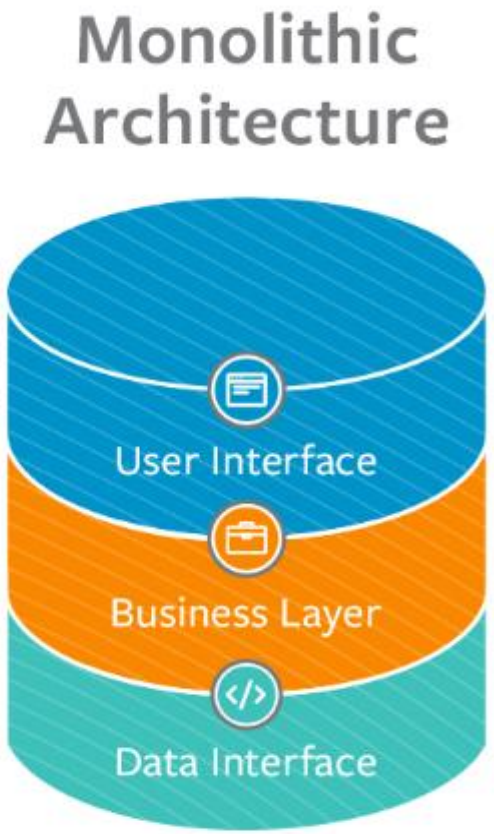
Why Not Working

- Environment is not Same
  - Hardware dependency
  - OS dependency
- Software Dependency
- Missing Environment Variable
- Not started as it should be
- Missing required files
- Application Version mismatch

Type of Application

- Monolithic Application
- Microservices (Modular) Application

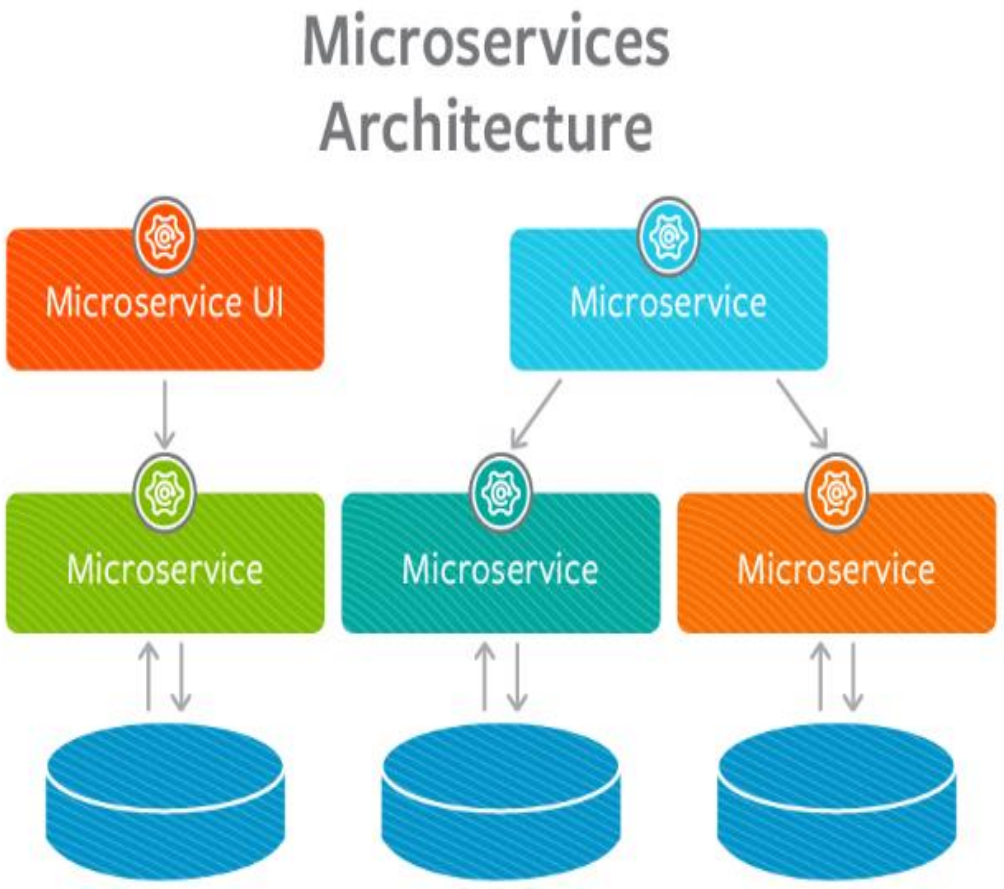
Monolithic Application



- Pros:
  - Easy to Develop & Test
  - Easy to Deploy
  - Easy to Manage
  - Easy to Debug
  - Easy to Scale horizontally

- Cons:
  - Reliability
  - Difficult to manage big application
  - Deploying changes & updates
  - Scaling
  - **Slowdown**

Microservices Application

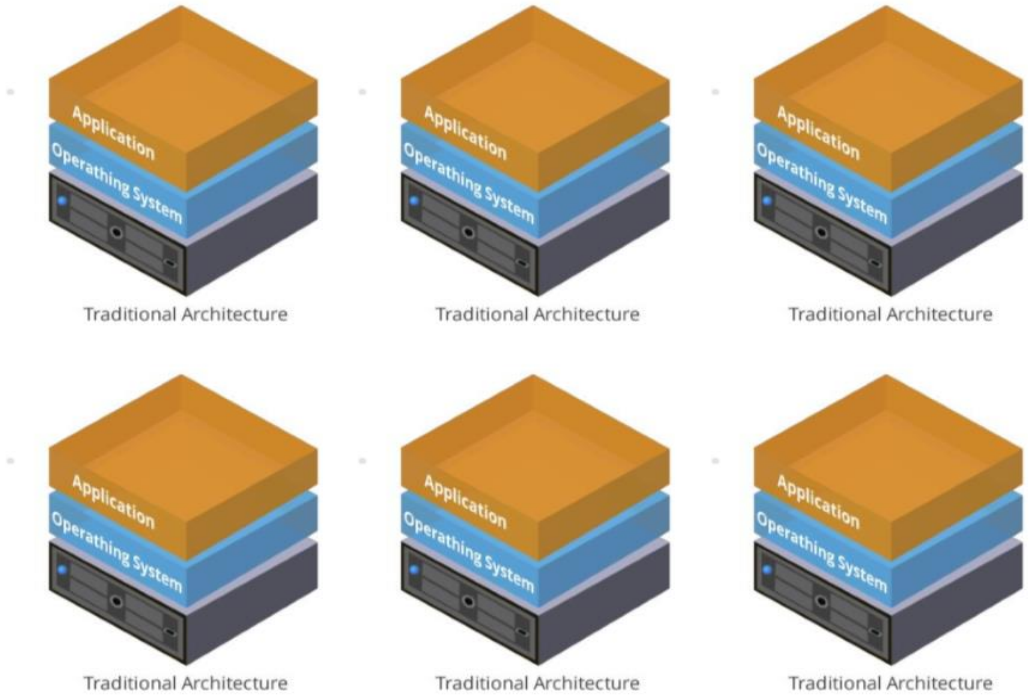


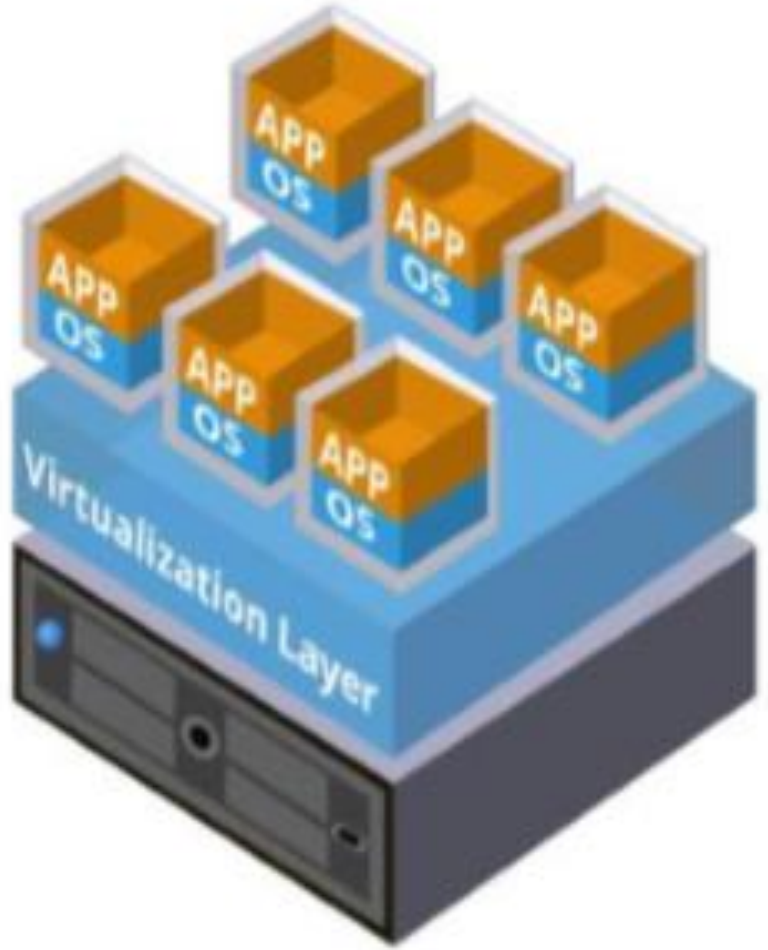
- Pros:
  - Managing complexity
  - Focused on modules
  - Flexible to accommodate new tech
  - Scalable
  - Focus on input and output
  - Continuous Deployment
  - Performance
- Cons:
  - Complexity due to Distributed Architecture
  - Difficult to test
  - Deployment is difficult

Why Virtualization

- Physical Machine
  - 1:1 usage
  - 20-30% Utilization
  - High cost of ownership
- Virtual Machine
  - 1:3, 1:5, 1:10 usage
  - 80-90% utilization
  - Low cost of ownership
  - Easy to maintain

Virtualization Architecture

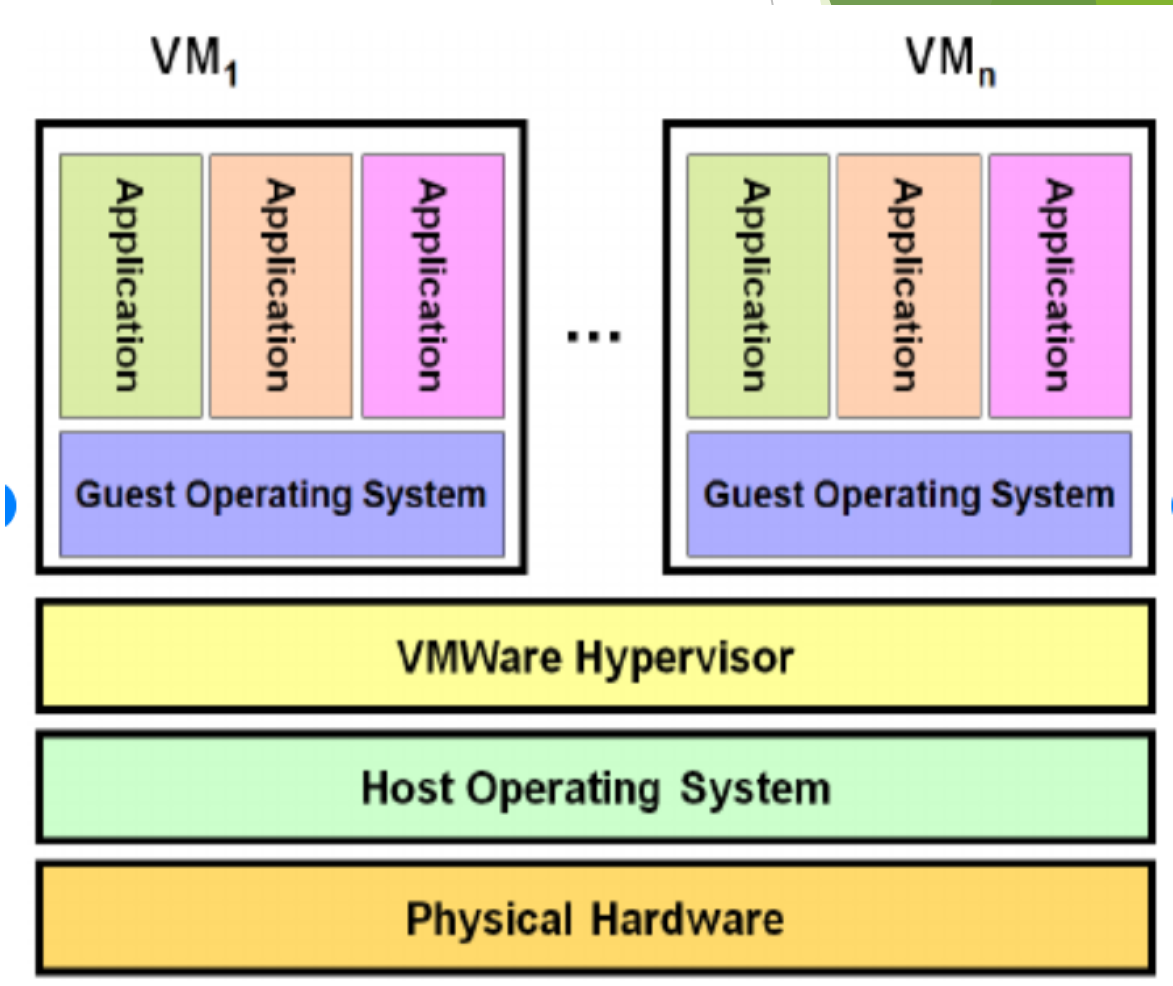




Virtual Architecture

Virtualization Benefit

- Easy to Manage Hardware Resource
  - Infrastructure
- Energy Saving
- Cost Effective
- Maximum Utilization of available resources
- Allocation of resources as per requirement
- Install what you need

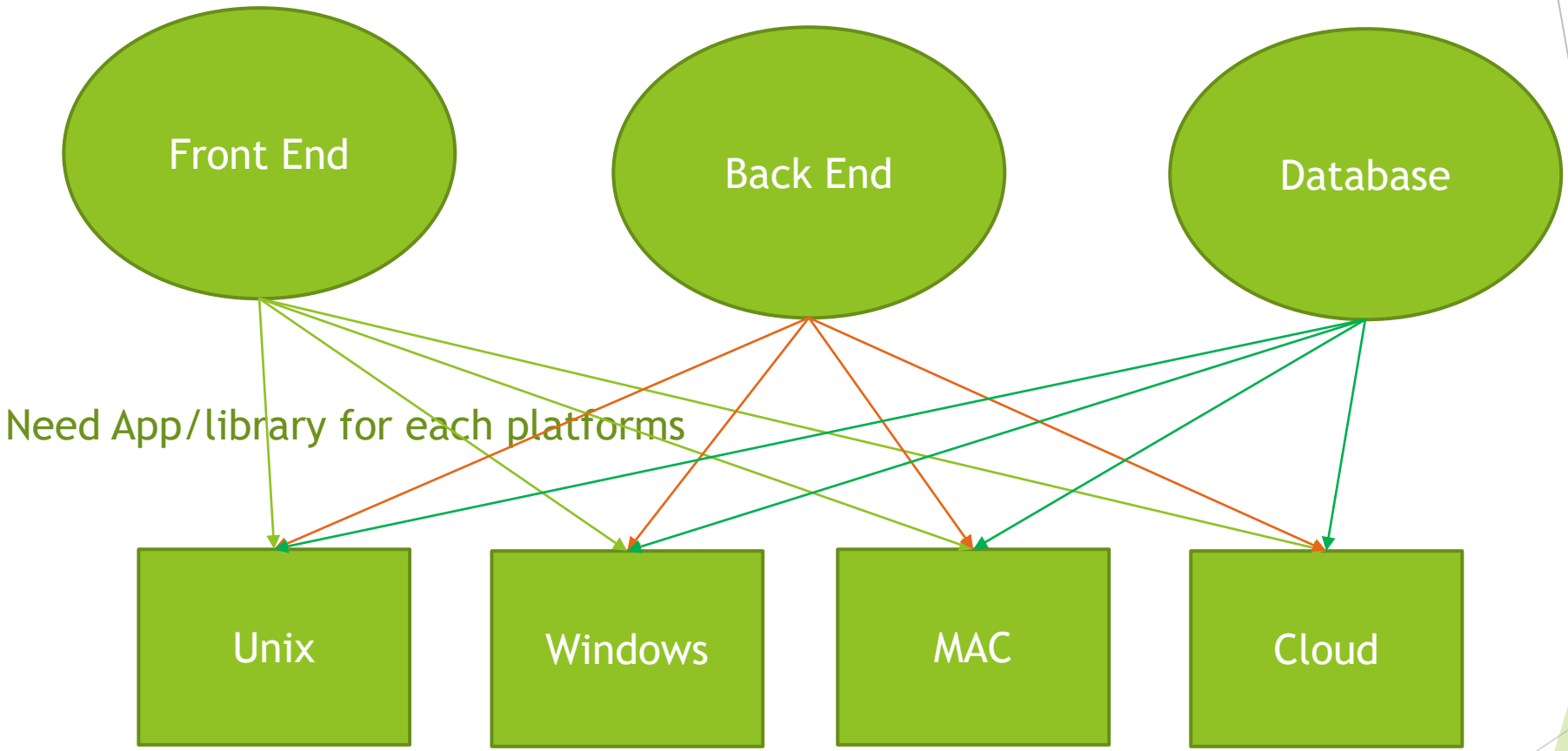


Problem with Virtualization

- High Booting time - each VMs has it’s own booting time
- Not Efficient uses of available resources
  - RAM Memory
  - Physical Memory
  - CPU
- Difficult to share data between different VMs
- Not able to create new VMs even resources are not fully used.

- Containerization Solve these issues with many more advantages.

Software Application Stack



Containers in Surrounding

Why Container

Some Items are big

Some Items are Small

Some Items are hard

Some Items are fragile

Some Items have Shape

Some Items have no Shape

Some are liquid

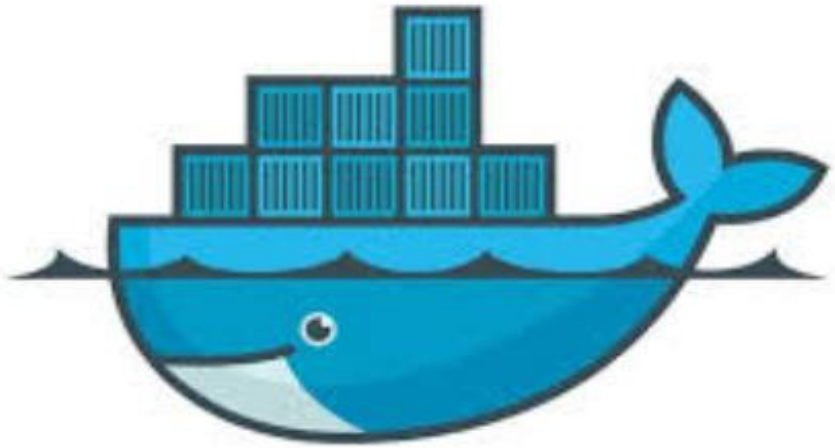
Why Container

Software Container

- Container contains everything required to run a module
  - Software application
  - Operation System
  - Libraries
  - Environment variable
  - Etc…
- Container can be placed anywhere

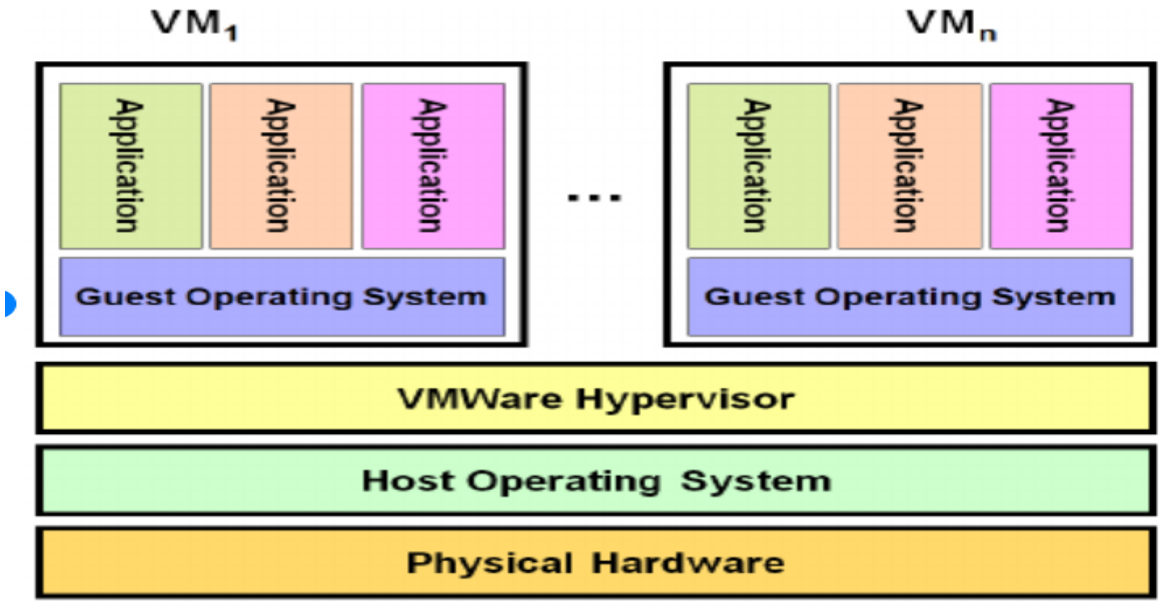
Container

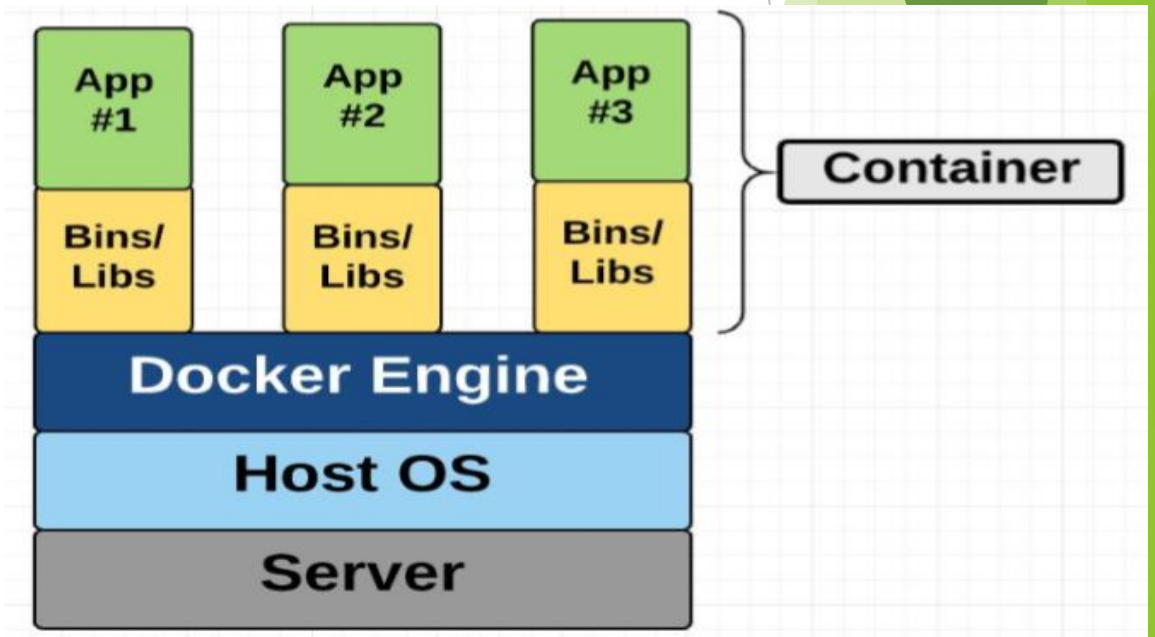
- Definition of Container – Containerization is an approach to software development in which an application or service, its dependencies, and its configuration are packaged together as a container image.
- containerized application can be tested as a unit
- deployed as a container image instance to the host operating system (OS)



Containers Vs Virtual Machine

- Virtual machine is virtualization on hardware however Containerization is on Operating system
- Efficient use of available hardware resources
- No dependency on Operating System
- Fast booting time
- Easy data sharing across multiple containers





Advantages of Containers

- Quick Startup
- Isolation
- Agility
- Scalability
- Portability
- Maintainability
- Versioning

Container Platforms

- Docker
- CRI-O
- rktlet

Installing Docker

https://docs.docker.com/

https://docs.docker.com/engine/install/centos/

- sudo yum install -y yum-utils
- sudo yum-config-manager \
  --add-repo \
  https://download.docker.com/linux/centos/docker-ce.repo
- sudo yum --nobest install docker-ce docker-ce-cli containerd.io
- sudo systemctl start docker
- docker version

Docker hub Demo

What is Docker

- Docker is a platform for developing, Shipping & Running application using an open-Source Container based technology
- OS based Virtualization
- Run anywhere – Physical, Virtual machine or Cloud
- Run anything – Any type of application can run in docker container
- Run in any number – Any number of container can be created using a docker image

Docker Components

- Core Components
  - Docker Daemon 

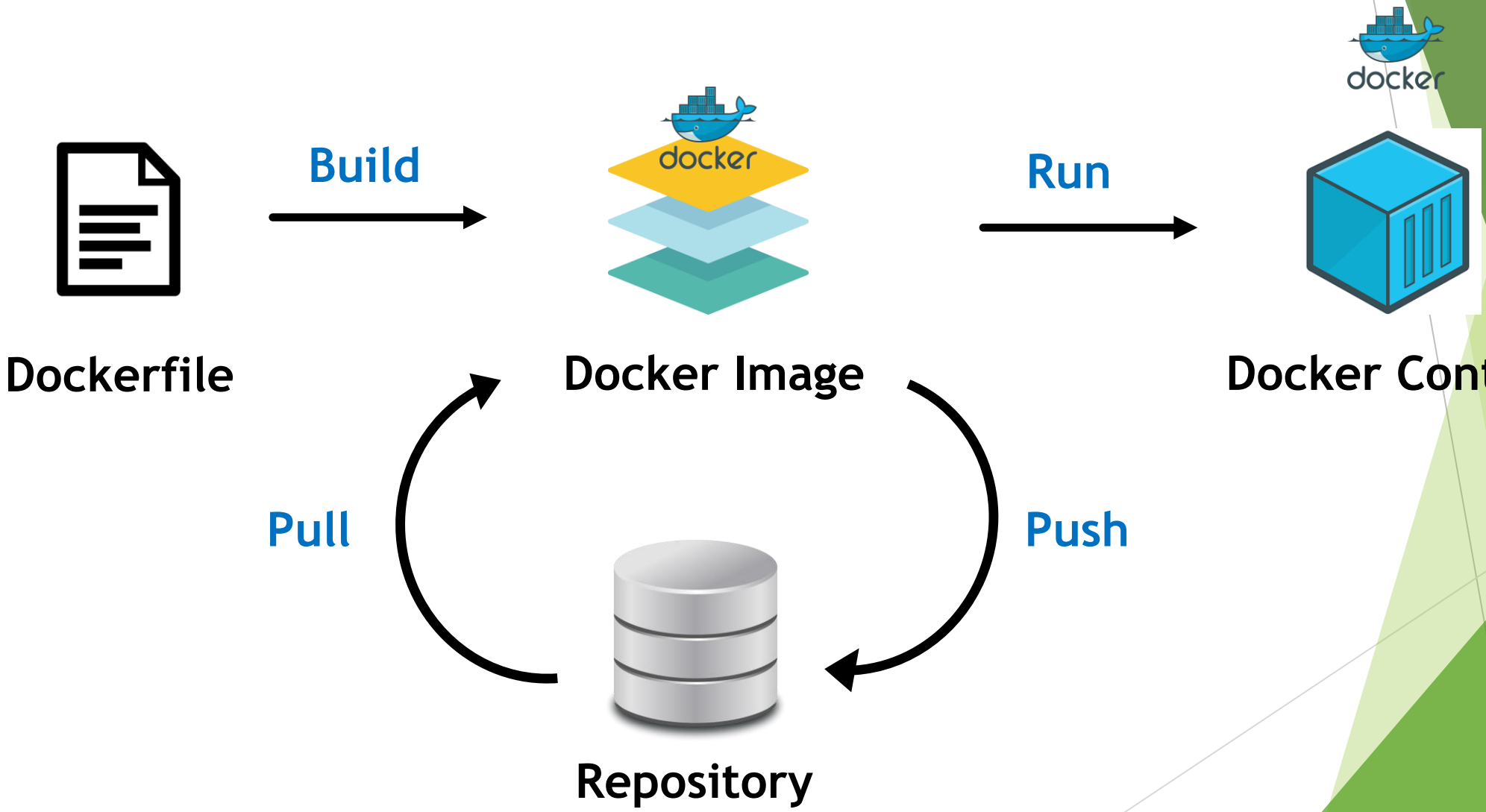
    Docker engine which runs on host machine and responsible for running/ managing docker system.
  - Docker Client 

    Docker client is platform using which user interact with docker daemon.

- Workflow Components
  - Docker Image : Template which holds environment and application
  - Docker Container: Run time instance of image – Start, Stop, Run, Delete

- Docker Registry
  - Public or Private repository for storing images

Docker Workflow



Running some script on a new PC

1) Install OS
2) Install programs
3) Copy files
4) Execute the script

Dockerfile

Dockerfile

```
FROM ubuntu

RUN apt install python3

COPY script.py /app

CMD python3 script.py
```

PC

1) Install OS
2) Install programs
3) Copy files
4) Execute script

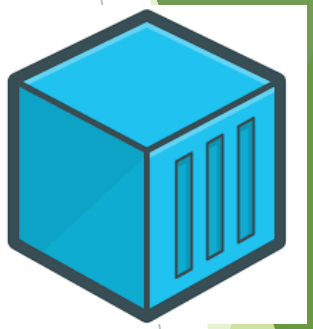
Docker Workflow

Dockerfile

Build

Docker Image

Run



Docker Container

```
FROM ubuntu

RUN apt install python3

COPY script.py /app

CMD python3 script.py
```

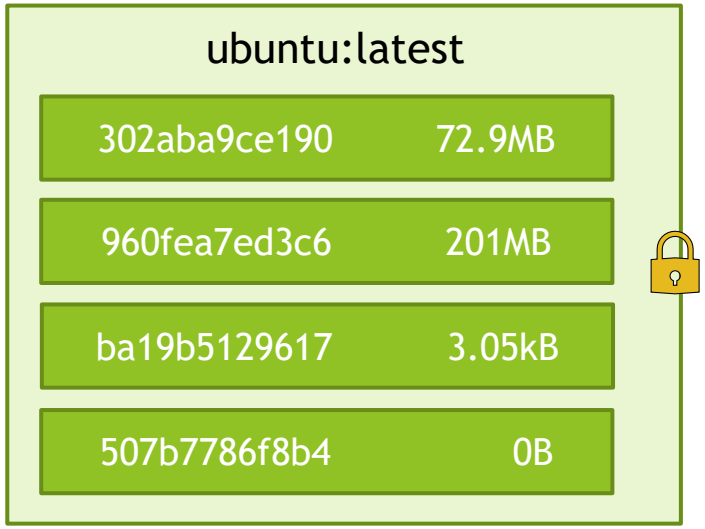


Image Layers (R/O)

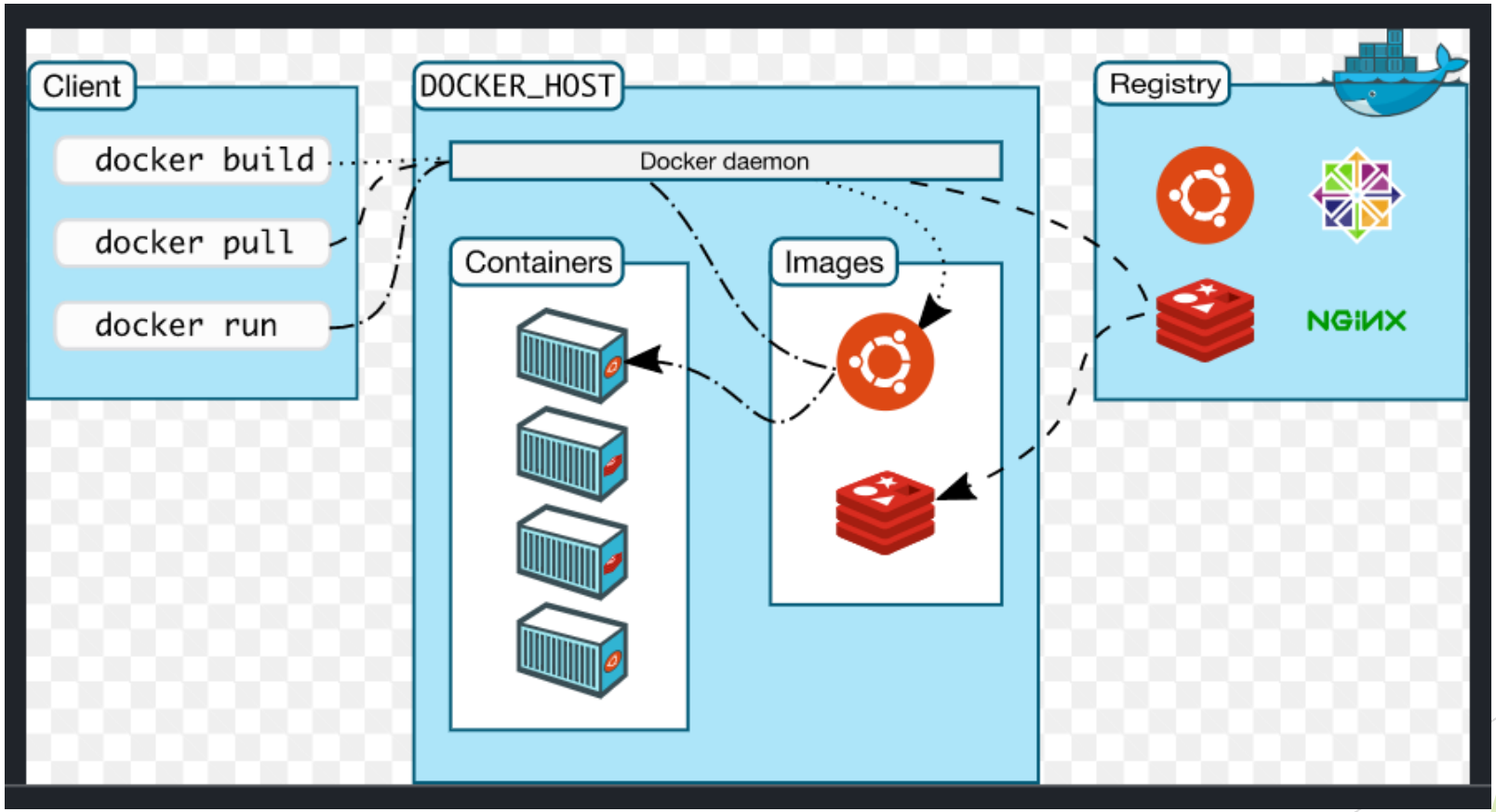
Container Layer

Docker info

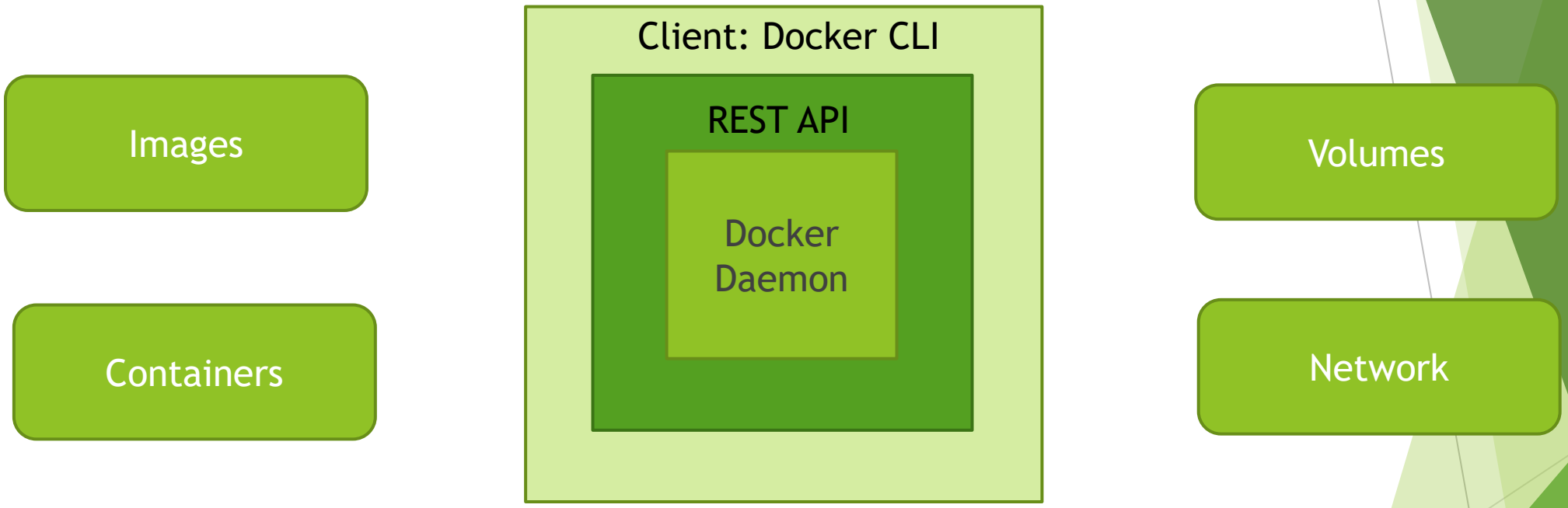
Docker Root Dir (default): /var/lib/docker

```
.
├── builder
├── buildkit
├── containers # container writtable layer
├── image
├── network
├── overlay2 # docker image layers
├── plugins
├── runtimes
├── swarm
├── tmp
├── trust
└── volumes
```

Docker Workflow



Docker Runtime Component



Docker image

- Images are files
- Comprised of multiple layer
- Contains environment
- Contains application
- Contains startup details

Images Basic Command

- List images available in local environment
  docker image ls
  docker images
- Getting image to local environment
  docker image pull <image-name>
- Tagging image
  docker image tag <old-name> <new-name>
- Publishing image to repository
  docker image push <image-name>
- Removing image
  docker image rm <image-name>
  docker rmi <image-name>
- Removing unused image
  docker image prune
- Saving image as tar
  docker image save <image-name> -o <output.tar>
- Loading image from tar
  docker image load --input <input.tar>
- Getting details about an image
  docker image inspect <image-name>
- History of an image
  docker image history <image-name>

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

Docker Container

- Running an image
  docker run <image-name>
- List all the running containers
  docker ps
- List all containers
  docker ps –a
- List all only container ID of all containers
  docker ps –aq
- Removing a containers
  docker rm <container-id>
  docker rm <container-name>
- Removing a running container forcefully
  docker rm –f <container-id>
  docker rm –f <container-name>

Docker Container

- Removing all containers
  docker rm –f `docker ps –aq`
- Running Container with giving container name
  docker run –-name <container-name> <image-name>
- Running Container in interactive mode
  docker run –it <image-name> /bin/bash
- Creating image from container
  docker commit –m”commit-message” <container-id> <new-image-name>
- Stopping and Starting container
  docker stop <container-id>
  docker start <container-id>
- Collecting container logs
  docker logs <container-id>

Docker Container

- Executing commands containers
  docker exec <container-id> <command>
- Attaching to a running container
  docker attach <container-name>
- Attaching to a running container in interactive mode
  docker exec –it <container-name> /bin/bash
- Copying files from Container
  docker cp <container-id>:<filename> <destination>

Docker Workflow

Build

```
docker build -t Name:Tag .
```

Run

```
docker -it -d image
docker create -it image
```

Dockerfile

Docker Image

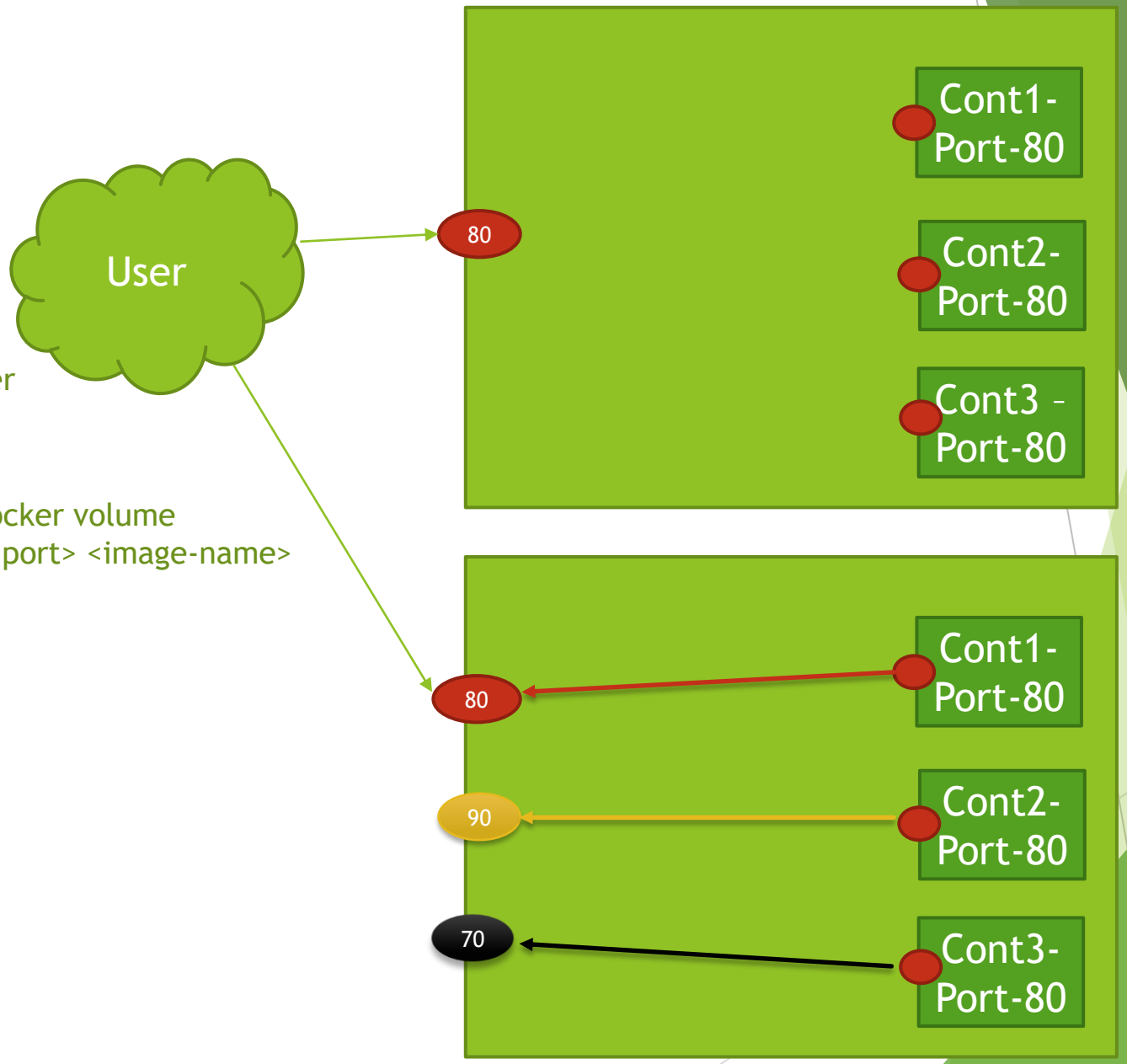
```
docker pull Name:Tag
docker push Name:Tag
docker tag old new

docker image ls
docker rmi Name:Tag
```

Docker Container

```
docker start ContainerID
docker stop ContainerID
docker restart ContainerID

docker exec -it ID command

docker ps -a

docker rm ID

docker logs ID

docker inspect ID
```

Docker Port

- Why????
- How??
  Mapping host port with container
- Running Docker container with docker volume
  docker run -p<host-port>:<cont-port> <image-name>



Docker Port

```
docker run -itd --rm -p 8081:80 nginx:latest

57dd72c3c1f4        nginx:latest        "/docker-entrypoint.…."   0.0.0.0:8081->80/tcp    funny_cori


 "NetworkSettings": {
            "Bridge": "",
            "SandboxID": "d5a075f5fc5576785be4213003b48dd51d255afbc56868b23eedeb9f44034778",
            "HairpinMode": false,
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "Ports": {
                "80/tcp": [
                    {
                        "HostIp": "0.0.0.0",
                        "HostPort": "8081"
                    }
                ]
            },
```

Docker Volumes

By default, all files created inside a container are stored on a writable container layer.

- The data doesn't persist when that container no longer exists, and it can be difficult to get the data out of the container if another process needs it.
- A container's writable layer is tightly coupled to the host machine where the container is running. You can't easily move the data somewhere else.
- Writing into a container's writable layer requires a storage driver to manage the filesystem. The storage driver provides a union filesystem, using the Linux kernel. This extra abstraction reduces performance as compared to using *data volumes*, which write directly to the host filesystem.

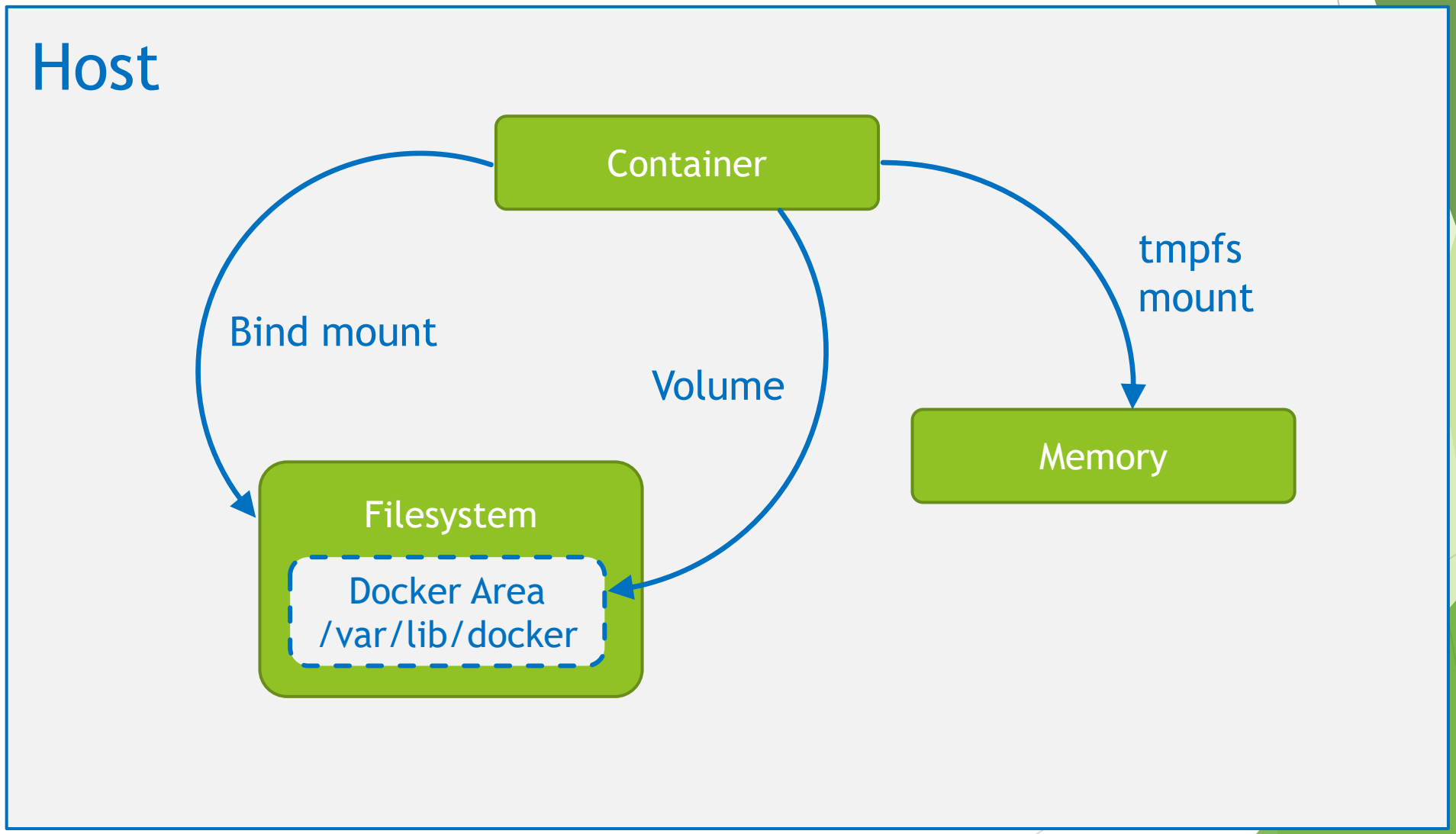
Docker Volumes

- **Volumes** are stored in a part of the host filesystem which is managed by Docker (/var/lib/docker/volumes/ on Linux). Non-Docker processes should not modify this part of the filesystem. Volumes are the best way to persist data in Docker.

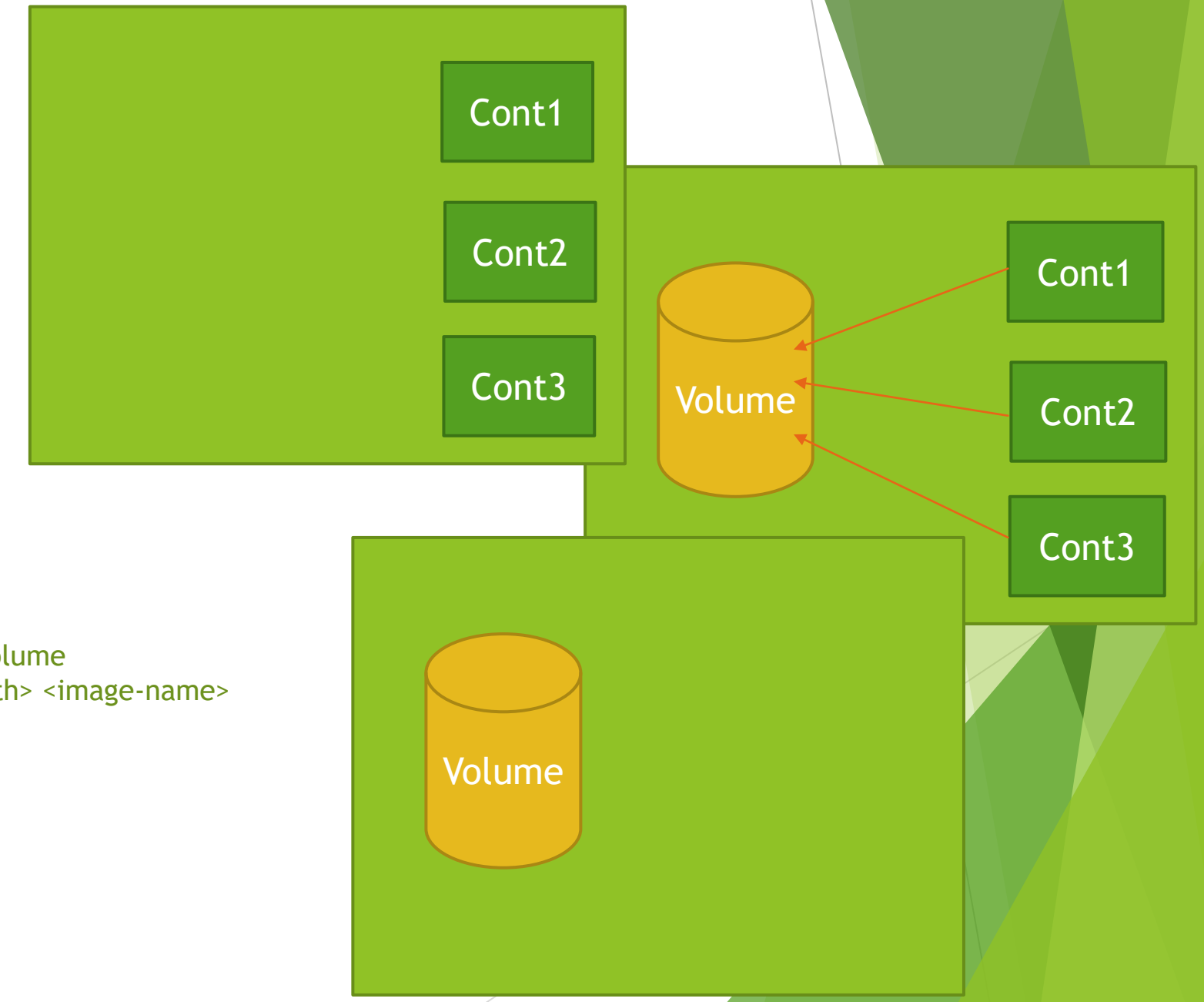
- **Bind mounts** may be stored *anywhere* on the host system. They may even be important system files or directories. Non-Docker processes on the Docker host or a Docker container can modify them at any time.

- **tmpfs** mounts are stored in the host system's memory only, and are never written to the host system's filesystem.

Docker Volumes



Docker Volume

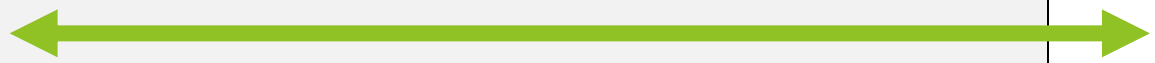


- Create Docker volume
  docker volume create <volume-name>
- Details of docker volume
  docker volume inspect <volume-name>
- Running Docker container with docker volume
  docker run -v<volume-name>:<cont-path> <image-name>
- Removing volume
  docker volume rm <volume-name>

Docker Volumes

Host Machine

```
docker run -itd -v NamedVol:/tmp/vol1 -v /tmp/vol2 alpine

/var/lib/docker/volumes

├── d5472c3139694b14d26337b300f0a20259b1d6a1b11e06441570422146d07b40
│   └── _data
│       └── fajl2
├── metadata.db
└── NamedVol
    └── _data
        └── fajl1
```



Container

```
/tmp

├── vol1
│   └── fajl1.txt
└── vol2
    └── fajl2.txt
```

Docker Volumes

docker inspect ContainerID


```
"Mounts": [
        {
            "Type": "volume",
            "Name": "NamedVol",
            "Source": "/var/lib/docker/volumes/NamedVol/_data",
            "Destination": "/tmp/vol1",
            "Driver": "local",
            "Mode": "z",
            "RW": true,
            "Propagation": ""
        },
        {
            "Type": "volume",
            "Name": "d5472c3139694b14d26337b300f0a20259b1d6a1b11e06441570422146d07b40",
            "Source": "/var/lib/docker/volumes/d5472c3139694b14d26337b300f0a20259b1d6a1b11e06441570422146d07b40/_data",
            "Destination": "/tmp/vol2",
            "Driver": "local",
            "Mode": "",
            "RW": true,
            "Propagation": ""
        }
```

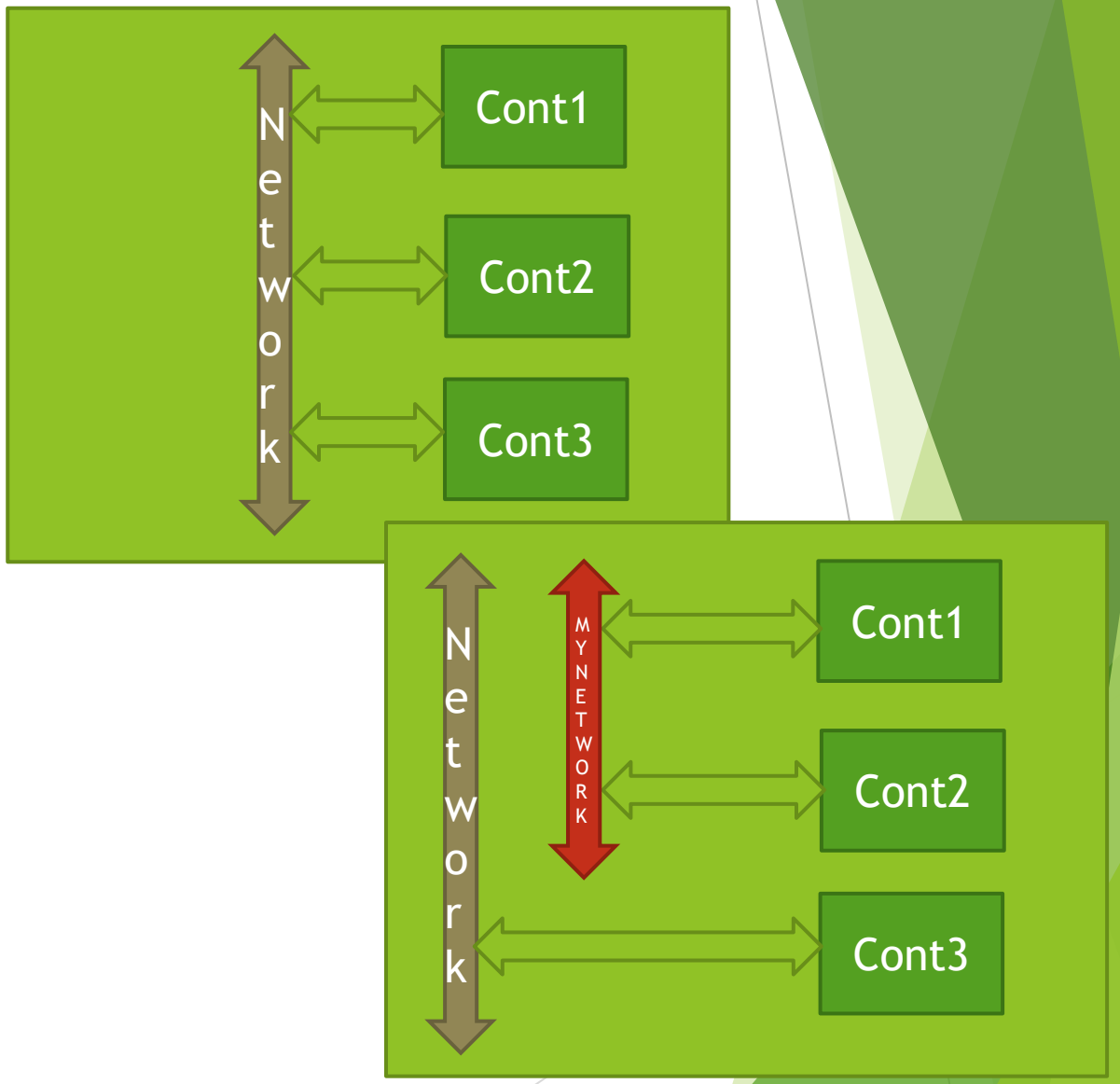
Docker Volumes (Bind Mounts)

docker run -itd -v NamedVol:/tmp/vol1 -v /opt/test:/tmp/vol2 alpine


```
"Mounts": [
        {
            "Type": "volume",
            "Name": "NamedVol",
            "Source": "/var/lib/docker/volumes/NamedVol/_data",
            "Destination": "/tmp/vol1",
            "Driver": "local",
            "Mode": "z",
            "RW": true,
            "Propagation": ""
        },
        {
            "Type": "bind",
            "Source": "/opt/test",
            "Destination": "/tmp/vol2",
            "Mode": "",
            "RW": true,
            "Propagation": "rprivate"
        }
```

Docker Network

- Why????
- How??
  Create a Network
  Run container with network
- Create Docker network
  docker network create -d bridge <network-name>
- Details of docker network
  docker network inspect <network-name>
- Running Docker container with docker volume
  docker run -network <network-name> <image-name>
- Removing a network
  docker network rm <network-name>



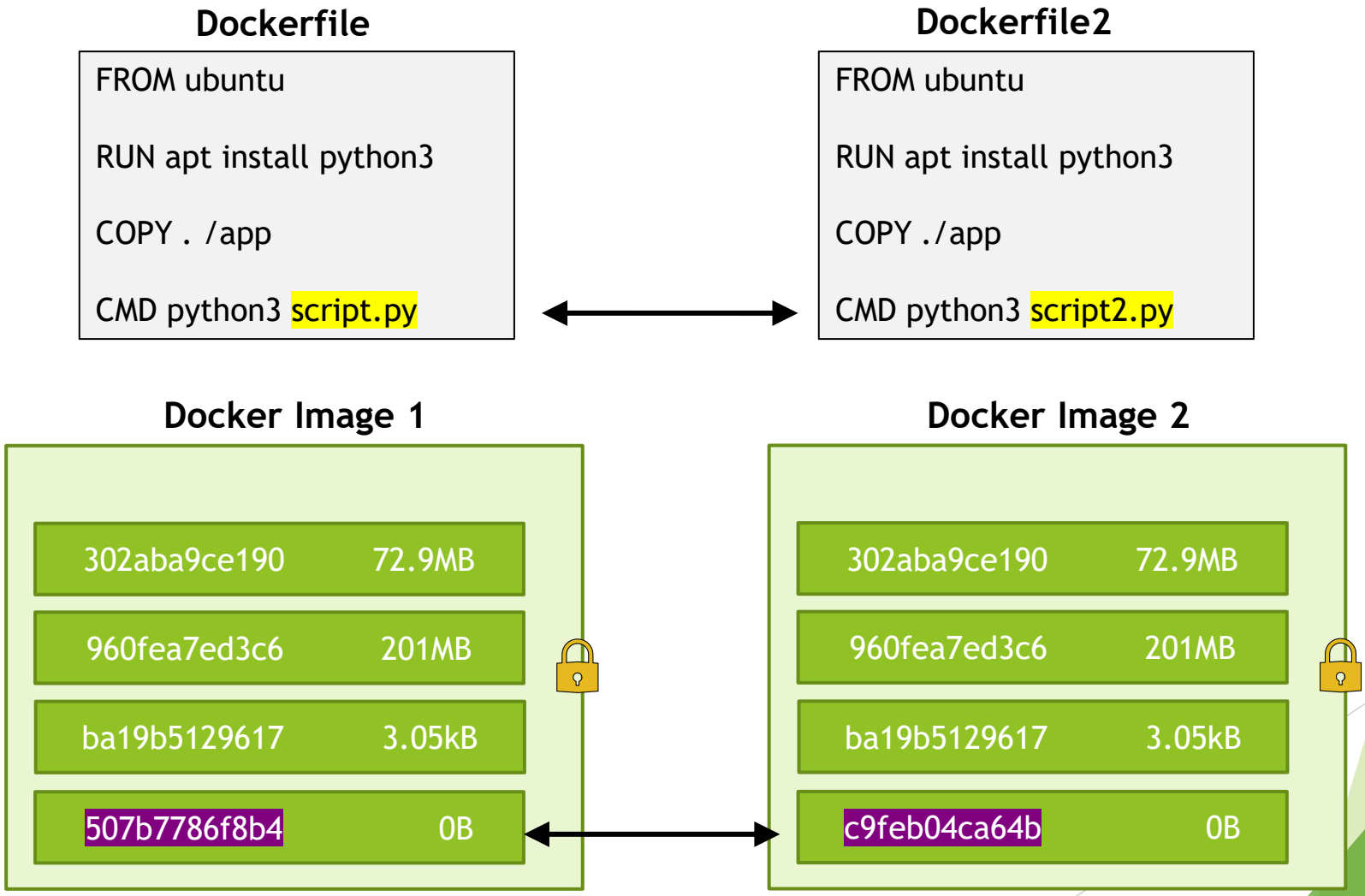
Create Docker image

- Docker Commit
- Dockerfile

Dockerfile

- Docker Commit
- Dockerfile
  - **FROM**: Sets the base image
  - **LABEL**: Sets the labels to the images – maintainer, company, product details, etc…
  - **RUN**: Executes commands inside the container
  - **ENV**: Sets an environment variable
  - **COPY**: Copies new files or directories into the filesystem of the container
  - **ADD**: Copies new files, directories or remote file URLs into the filesystem of the container
  - **USER**: Sets the username or UID to use when running an image
  - **CMD**: Default first command to execute
  - **ENTRYPOINT**: Allows you to configure a container that will run as an executable
  - **WORKDIR**: Sets the working directory for any RUN, CMD, ENTRYPOINT, COPY, and ADD commands
  - **EXPOSE**: Informs Docker that the container listens on the specified network port at runtime.
  - **VOLUME**: Creates a mount point and marks it as holding externally mounted volumes from native host or other containers

Dockerfile


Dockerfile
FROM ubuntu
RUN apt install python3
COPY . /app
CMD python3 script.py
Dockerfile2
FROM ubuntu
RUN apt install python3
COPY ./app
CMD python3 script2.py
Docker Image 1
302aba9ce190 72.9MB
960fea7ed3c6 201MB
ba19b5129617 3.05kB
507b7786f8b4 0B
Docker Image 2
302aba9ce190 72.9MB
960fea7ed3c6 201MB
ba19b5129617 3.05kB
c9feb04ca64b 0B

Dockerfile

```
docker inspect hanged:0.1 | grep -A 5 Layers
        "Layers": [
            "sha256:0f7b3ff8b310adb0c38fa8108967e51e3431bc4b7ce350de93839eeffcefd34c",
            "sha256:df6317dc62aa3451b931ac54bfec511306d04d3f72ca20bb6ef7541d0ae362b4",
            "sha256:007a341794883b3349cb24301c2c9737cc8cdf00f503cc8422a3cd4b33944e3c",
            "sha256:386bc3a70d2b5334c7c7867cf8cda36ca43782cb862b10919944661718f83e9f"
        ]

docker inspect hanged:0.2 | grep -A 5 Layers
        "Layers": [
            "sha256:0f7b3ff8b310adb0c38fa8108967e51e3431bc4b7ce350de93839eeffcefd34c",
            "sha256:df6317dc62aa3451b931ac54bfec511306d04d3f72ca20bb6ef7541d0ae362b4",
            "sha256:007a341794883b3349cb24301c2c9737cc8cdf00f503cc8422a3cd4b33944e3c",
            "sha256:97d156e5424e0c96ac0c7e9ea98d18be3035bed3a2d8820e45c08156de455cd6"
        ]
```

Dockerfile

docker history hanged:0.1

IMAGE	CREATED	CREATED BY	SIZE	COMMENT
c9feb04ca641	8 seconds ago	/bin/sh -c #(nop) CMD ["python3" "hanged.py…	0B	
e3725269e308	3 minutes ago	/bin/sh -c #(nop) COPY dir:fe241d92c22b6e8de…	3.72kB	
66edb2b5997d	6 minutes ago	/bin/sh -c #(nop) WORKDIR /usr/src/app	0B	
c13c58ae7ed7	6 minutes ago	/bin/sh -c export http_proxy=http://10.215.4…	201MB	
302aba9ce190	2 weeks ago	/bin/sh -c #(nop) CMD ["/bin/sh"]	0B	
<missing>	2 weeks ago	/bin/sh -c #(nop) ADD file:6b081cabb4b256ee0…	5.61MB	

docker history hanged:0.2

IMAGE	CREATED	CREATED BY	SIZE	COMMENT
a53fb853ed24	3 minutes ago	/bin/sh -c #(nop) CMD ["python3" "sum.py"]	0B	
e3725269e308	3 minutes ago	/bin/sh -c #(nop) COPY dir:fe241d92c22b6e8de…	3.72kB	
66edb2b5997d	6 minutes ago	/bin/sh -c #(nop) WORKDIR /usr/src/app	0B	
c13c58ae7ed7	6 minutes ago	/bin/sh -c export http_proxy=http://10.215.4…	201MB	
302aba9ce190	2 weeks ago	/bin/sh -c #(nop) CMD ["/bin/sh"]	0B	
<missing>	2 weeks ago	/bin/sh -c #(nop) ADD file:6b081cabb4b256ee0…	5.61MB	

Container orchestration

- Docker Swarm
- Docker Compose
- Kubernetes
- Openshift
- Many more ----

Docker-Compose

- Need configuration file
  - Services
  - Environment
  - Volumes
  - Port
- Deploying Services
  - docker-compose up –f <filename>
- Staring/Stopping Services
  - Docker-compose start/stop
- Removing services
  - Docker-compose rm
- Scaling services
  - Docker-compose up scale <service-name>=2

curl -L https://github.com/docker/compose/releases/download/1.20.1/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose

Docker-Compose

Docker command

```
docker run -itd nginx:latest
```

docker-compose equivalent

```
version: "3.3"
services:
  webserver:
    image: nginx:latest
```

Docker-Compose

Docker command

```
docker run -itd --rm -p 8080:80 -v/home/ptdadmin/website:/usr/share/nginx/html nginx:latest
```

docker-compose equivalent

```
version: "3.3"
services:
  nginx:
    image: nginx:latest
    ports:
      - "8080:80"
    volumes:
      - /home/ptdadmin/website:/usr/share/nginx/html
```

Docker-Compose

Multiple services (dependencies)

Volumes needs to be defined first, bind mounts do not

```
version: "3.3"
services:
  db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    ports:
      - "8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
volumes:
  db_data: {}
```

Any Query

Thank you