Docker

Welcome

Making Intellectuals Meet Excellence

Safe Harbour

- All images used are taken with consensus for this education purpose and does not associate with any obligation of Intellectual Property.

SAR INFOTECH
Making Intellectuals Meet Excellence

Day 2

- ENTRYPOINT
- Volume
- More Docker Commands
- Networking and Ports

ENTRYPOINT

- --entrypoint or in Dockerfile
- Two Forms
  - ENTRYPOINT Command , PARAM1
  - ENTRYPOINT [“Cmd”, “PARAM1”]

  - Entry point treats Container as Executable
  - Default values for entry point can come via CMD
  - CMD can be overridden

  - FROM ubuntu
  - ENTRYPOINT [“/bin/bash”]

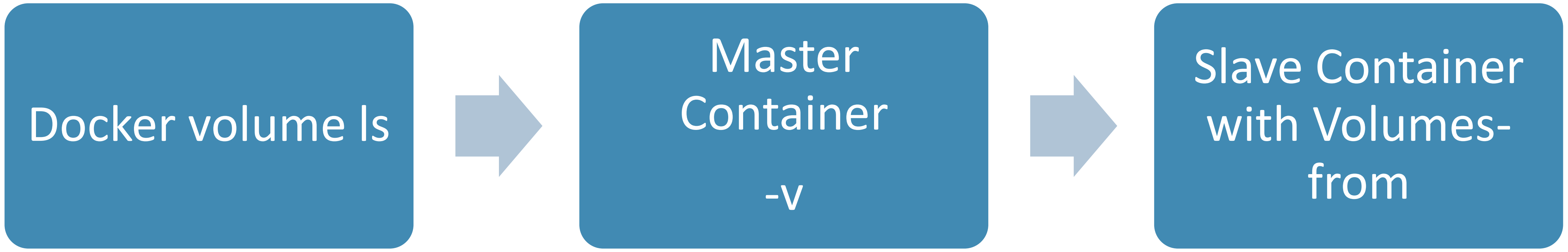
SAR INFOTECH
Making Intellectuals Meet Excellence

VOLUMES

- A volume is a specially-designated directory within one or more containers that bypasses the Union File System. Volumes are designed to persist data, independent of the container's life cycle.
- Docker therefore never automatically delete volumes when you remove a container, nor will it "garbage collect" volumes that are no longer referenced by a container
- Types
  - Host (FS)
  - Named (Disk)
  - Anonymous
- Driver
  - Flocker
  - local

SAR INFOTECH
Making Intellectuals Meet Excellence

Volume Creation – Type 1



```
Docker run –it –name mastervol-data – v bddata:/bddata ubuntu /bin/bash
```

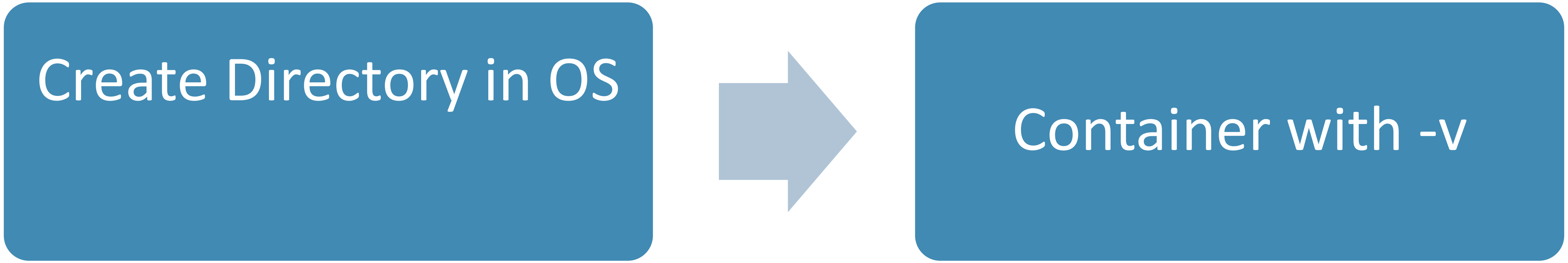
```
# cd data
```

Create files and directories.

```
Docker inspect bddata
```

```
Docker run –it –name slave_from_master –volumes-from mastervol-data  ubuntu /bin/bash
```

Volume Creation – Type 2



```
docker run -dit --name devmtest1 -v ~/data:/app ubuntu /bin/bash
```

VOLUME CREATION – TYPE3

- Dockerfile
  - FROM ubuntu
  - RUN mkdir /data
  - RUN mkdir /newdir
  - VOLUME /data
- Docker build –t
- Docker exec –it

More Docker Commands…

- Docker diff <<container ID>
  - A, C, D
- Docker logs <<container ID>>
- Docker export –o a.tar <<container ID>>
- Docker kill <<container id>>
- Docker commit –mpc <<containerid>> <<new imageid>>
- Docker attach <<container>>
- Docker images [-q]
- Docker save <image> as <Tar>
- Docker load < [Tar File]

More Docker Commands

- Docker create –it fedora bash
    - Docker start
    - Docker exec –it << CID>> bash

PORTS

- Container Centric Communication
- Inter Container Communication (between Container)
- Intra Container Communication (Outside Docker)

- EXPOSING and PUBLISHING PORTS
  - **Exposing ports is a way of documenting which ports are used, but does not actually map or open any ports. Exposing ports is optional**.

- PRIVATE and PUBLIC PORTS

SAR INFOTECH
Making Intellectuals Meet Excellence

PORTS STRATEGY

- EXPOSE
  - EXPOSE private_port
  - EXPOSE private-port:public-port
  - EXPOSE 7000 7001 7002

- PUBLISH –p
  - Host:container_port
    - 7000:7000
    - -p 7000:7000 –p 7001:7001
  - Host-range-port:container-range port
    - 7001-7010:7001-7010

SAR INFOTECH
Making Intellectuals Meet Excellence

Publishing Ports through CLI

- docker run -d -p 9090:80 -t --name ngin nginx
- Docker-machine ip
- Docker exec –it ngin bash
- #apt-get update
- #apt-get install –y curl

DockerFile Porting

- Try the same in docker file posting using EXPOSE
  - FROM nginx
  - RUN apt-get update
  - RUN apt-get install –y curl
  - EXPOSE 9090
  - ENTRYPOINT [“curl http://192.168.99.100:9090”]

- Docker build –t newimgng .
- Docker run -it newimgng bash
- Curl http://192.168.99.100:9090

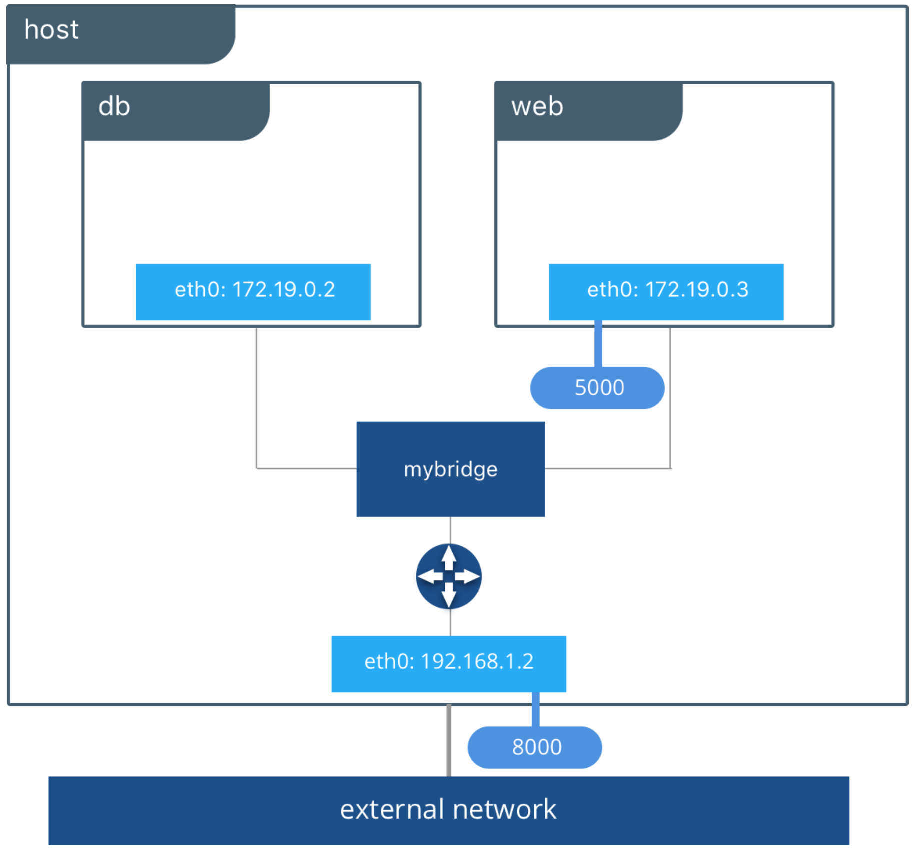
Lets try Apache now !

```
docker run -t -d --name webserver -p 80:80 ubuntu /bin/bash

docker exec –it webserver bash
```

CNM

- CNM
- Drivers - **bridge**, **overlay** and **macvlan.**
- **Bridge**
    - Creates a private network internal to the host so containers can communicate. External access is granted by exposing ports to containers. Docker secures the network by managing rules that block connectivity between different Docker networks.
    - Behind the scenes, the Docker Engine creates the necessary Linux bridges, internal interfaces, iptables rules, and host routes to make this connectivity possible



SAR INFOTECH
Making Intellectuals Meet Excellence

Bridges…

The built-in Docker `overlay` network driver radically simplifies many of the complexities in multi-host networking. It is a **swarm scope** driver, which means that it operates across an entire Swarm or UCP cluster rather than individual hosts. With the `overlay` driver, multi-host networks are first-class citizens inside Docker without external provisioning or components. IPAM, service discovery, multi-host connectivity, encryption, and load balancing are built right in.

Newest built-in network driver and offers several unique characteristics. It’s a very lightweight driver, because rather than using any Linux bridging or port mapping, it connects container interfaces directly to host interfaces macvlan is a local scope network driver which is configured per-host. As a result, there are stricter dependencies between MACVLAN and external networks, which is both a constraint and an advantage that is different from overlay or bridge.

Commands for Docker network

- Docker network ls
- Docker network inspect bridge
  - Options , config ,Subnet, Gateway
- Docker run –dit –name alp1 alpine ash
- Docker run –dit –name alp2 alpine ash
- Docker container ls
- Docker network inspect bridge
  - Docker attach alp2

  Ping –c2 (alp2)

SAR INFOTECH
Making Intellectuals Meet Excellence