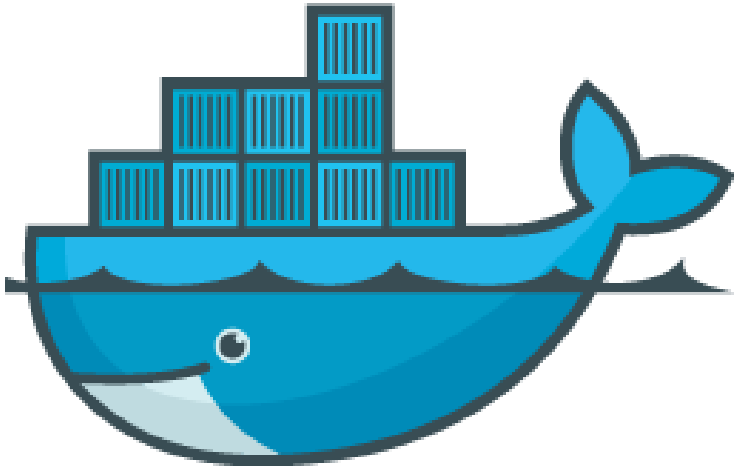


Docker

By
ADITYA PRABHAKARA

Introducing Myself

Aditya S P (sp.aditya@gmail.com)
Freelance trainer and technologist

Boring Stuff about me:

- 14+ years of experience in development and training
- Started with Java, moved to Android and now working on Big Data Technologies

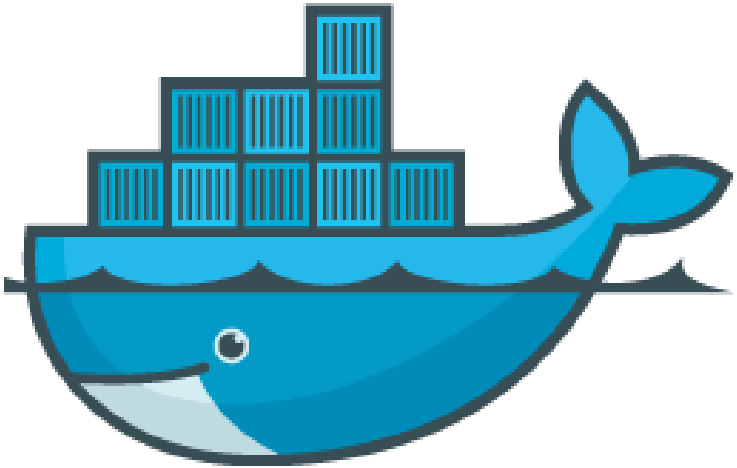
Interesting Things about me:

- Actually Nothing !

Getting to know you

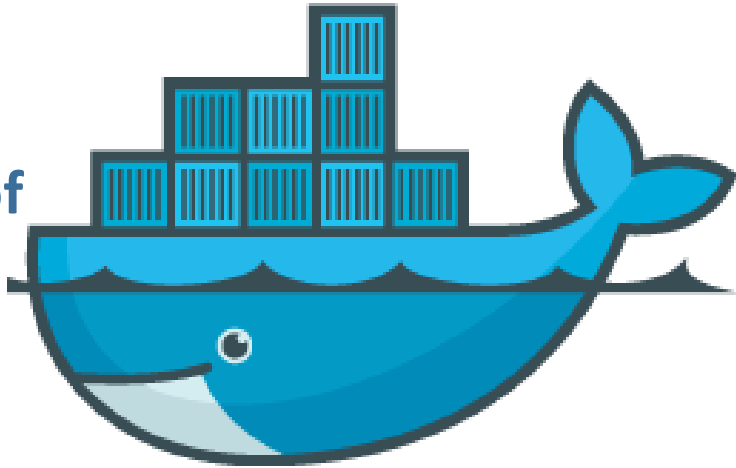
Agenda

- Introduction to DevOps
- Docker

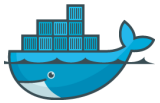


Course Objectives

- A good understanding of DevOps
- A good fundamental understanding of Docker
- Where does docker fit in the DevOps Movement
- Understanding of role of Kubernetes

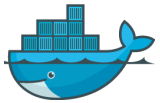


Chapter: Introduction



Docker – Why Now?

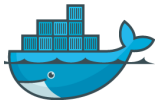
- Speed. Speed. Speed.
- Value movement dev-> test-> prod easier and faster
- Portability
- Reduce complexity of developing code for distributed systems
- Reduce complexity of deploying code to the cloud
- For a later time - Docker's founder and CTO Solomon Hykes
- https://www.youtube.com/watch?v=3N3n9FzebAA



Docker – Different Versions

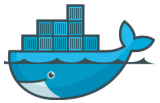
- https://www.docker.com/get-docker
- Community Edition and Enterprise Edition
- Stable and Edge

- Stable vs. Edge Cont.
- Edge (beta) released monthly,
- Stable quarterly
- Edge gets new features first, but only supported for a month
- Stable rolls in three months of Edge features



Docker – Setup

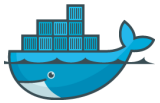
- Docker toolbox install



Docker – initial commands

- docker version
  - verified it's working
- docker info
  - most config values
- docker command line structure
- docker (options)

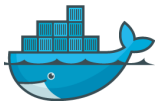
Chapter: Container



Container

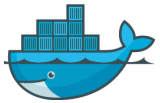
- Basic Building block
- Let us get a container running and then we will connect the dots
- Execute the command

```
docker container run nginx
```

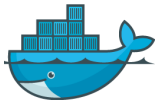


Container

- They are not really mini vms. They are processes
- They get their own logical filepath, process space
- They exit when the process stops
- Some docker container command examples
  - docker top
  - docker container ls
  - docker container stop

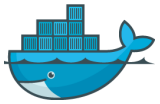


First Container Run:What just happened?



Knowing more about a Container

- docker container stats <container id>
- docker container inspect <container id>
- docker container top <container id>

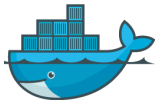


Interactive Container

- docker container run –it nginx bash
- docker container exec –it <container id>

Try this out !
“alpine” is light weight linux distribution , run an alpine container interactively

Chapter: Images



What is an image

- Application binaries
- Application dependencies
- Some meta data about what to run and how to run
- Not a full fledged OS – No kernel No drivers
- Where are these images stored?

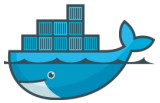
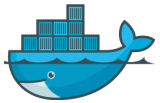


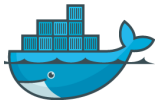
Image vs Container

- An image is an application we want to run
- A container is an instance of the image running as a process
- Multiple containers can run using the same image
- A bad analogy but helps to get the point across : an image is like a “.exe” file
  Container is application that runs when we click on that “.exe”



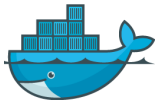
Introduction to docker hub

- What is Docker Hub
- How to find images
- How do we say an image is good!
- Versions of images
- What are official images
- Download images



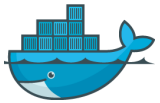
docker container run

- Look for image locally in image cache
- If nothing exists, then look in image repository
- Downloads the image related to the tag
- Creates a new container based on that image
- Provides a virtual ip on a private network inside docker engine
- Publishes a port if specified
- Starts the process in the container using the CMD in the image Dockerfile



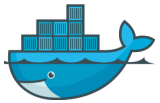
Working with images

- Pull an image
- Pull based on a tag



Images and layers

- Union file system concept
  - Layers of files and meta data
  - docker image history nginx
  - Saves space as it reuses the layers



Layered Visualization

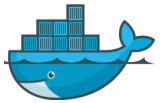
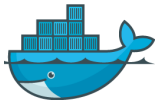


Image and push

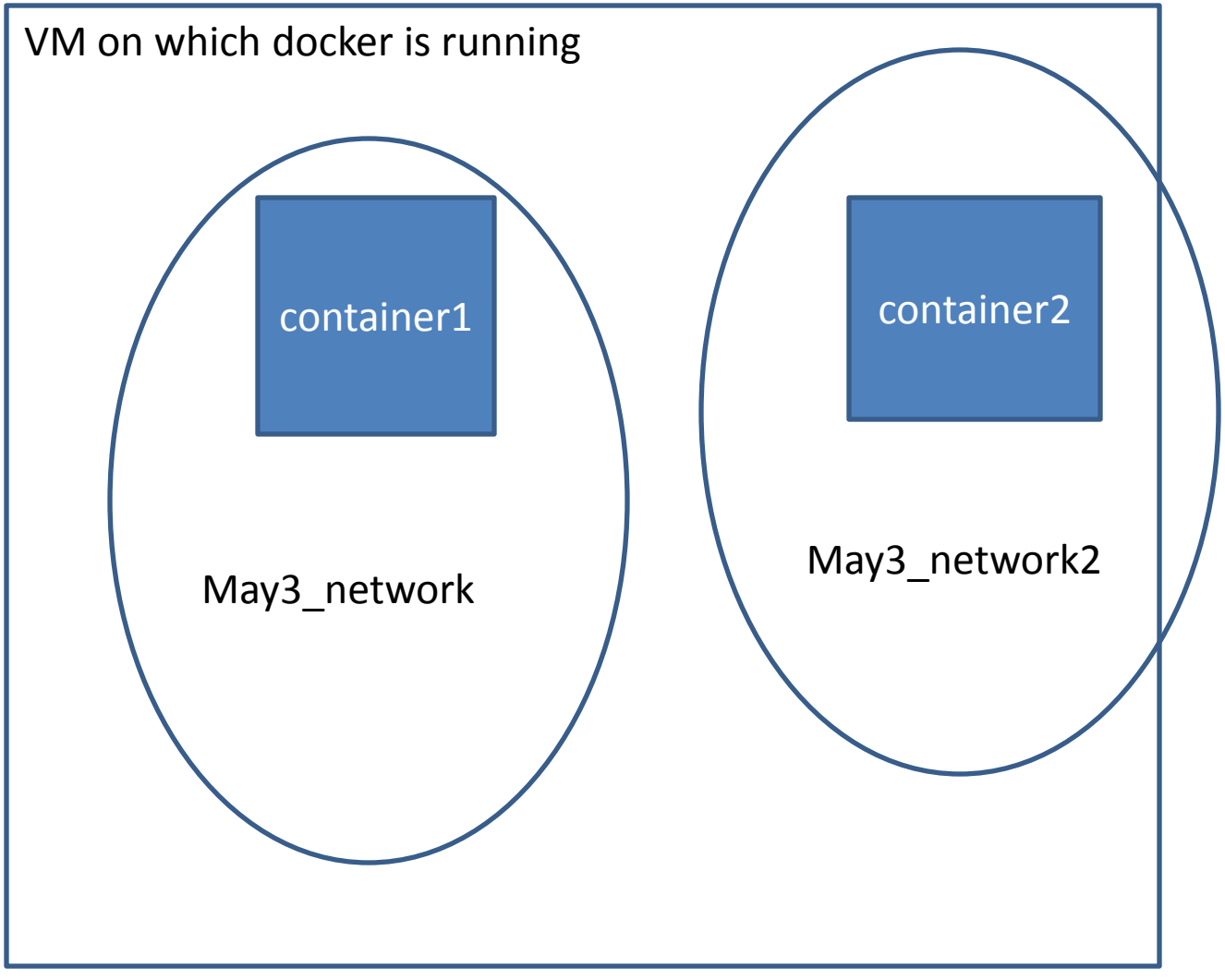
- An image has no real name as such
- It is uniquely identified through user/image:tag
- I can retag an existing image and push to my repository
- Only official images do not have username every other image has a user id behind it

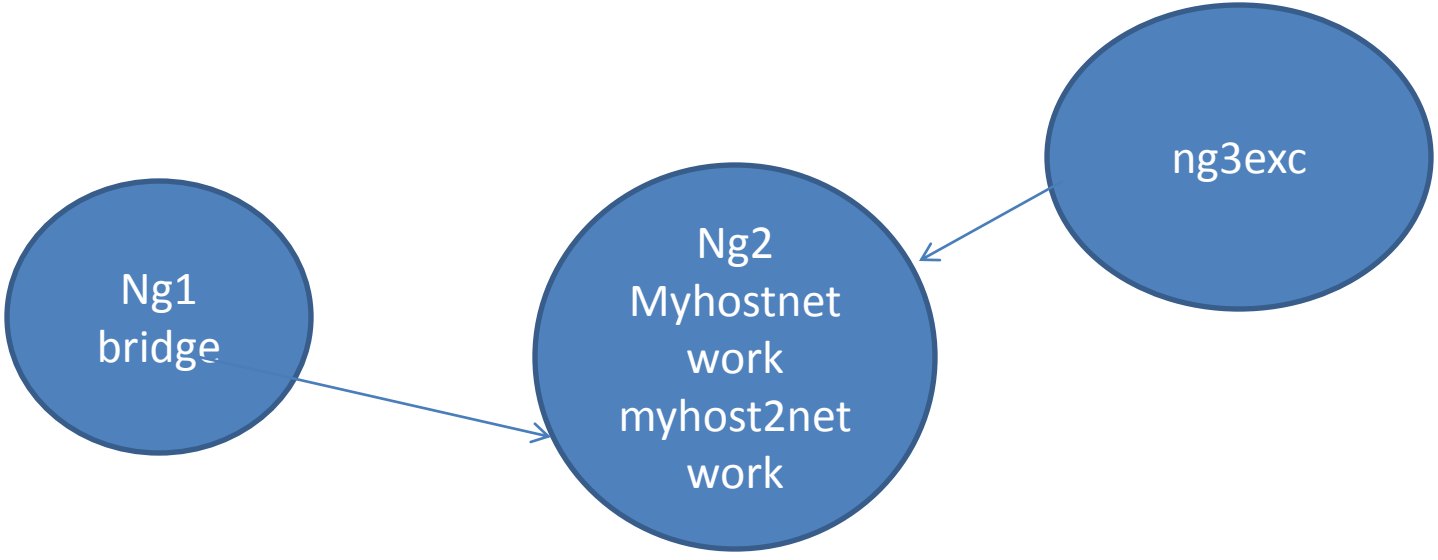
Chapter: Networking



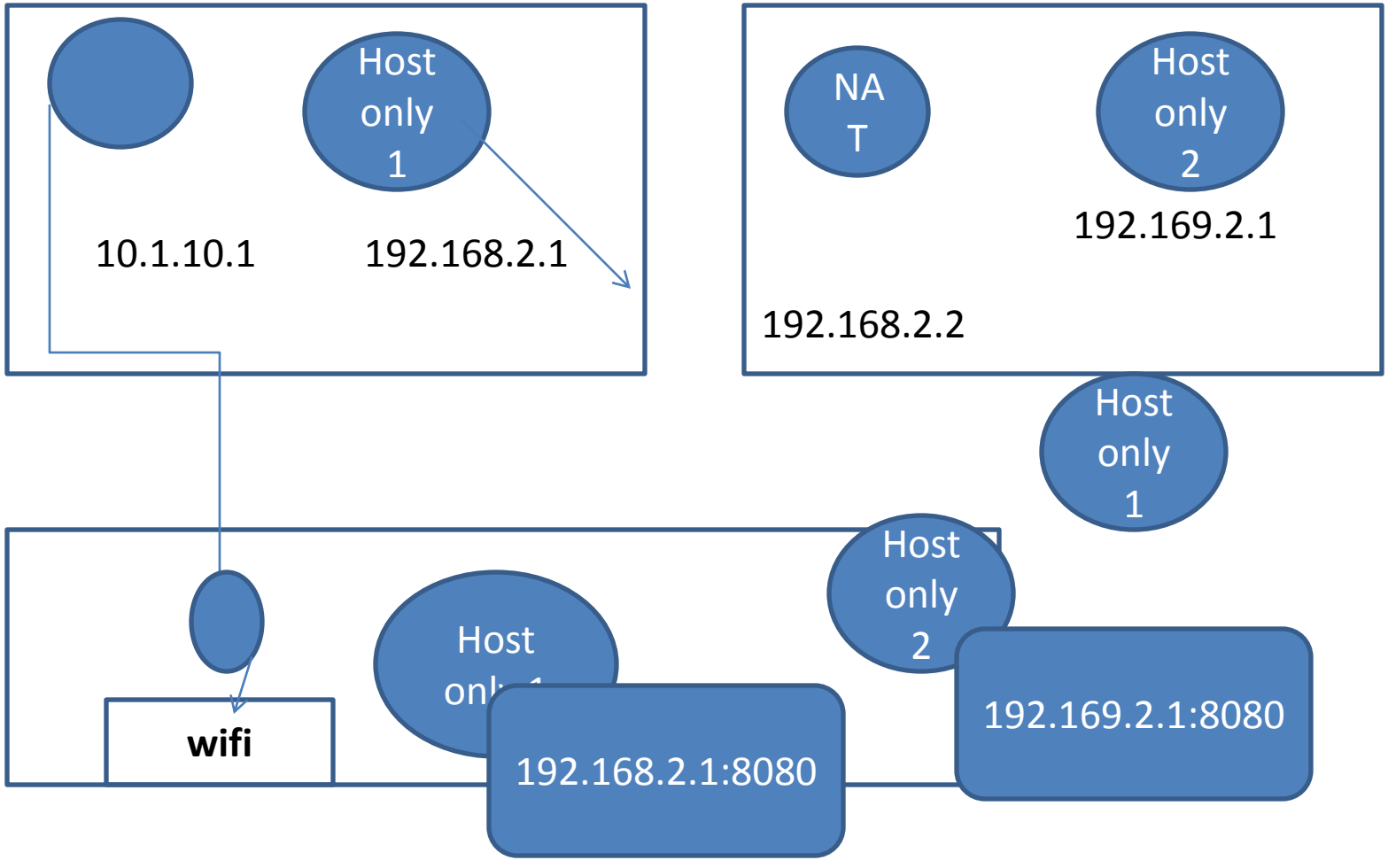
Container Network

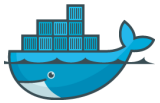
- An image has no real name as such
- Each container connected to a private virtual network "bridge"
- Each virtual network routes through NAT firewall on host IP
- All containers on a virtual network can talk to each other without -p
- Best practice is to create a new virtual network for each app:
  - network “my_weblayer" for mysql and php/apache containers
  - network “my_mongo_rest" for mongo and nodejs containers


VM on which docker is running
container1
May3_network
container2
May3_network2



Link ng2 myhost2network ng1

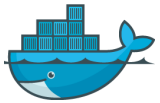

Host only 1
10.1.10.1
192.168.2.1
NA T
Host only 2
192.169.2.1
192.168.2.2
Host only 1
Host only 2
Host
wifi
192.168.2.1:8080
192.169.2.1:8080



Docker network commands

- docker network ls
- docker network inspect bridge
  - Check the containers running
  - Check the ip address
- docker network create my_own_network

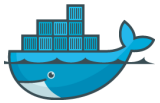
Chapter: Docker File



Docker Building Images

- Dockerfile basics
- FROM (base image)
- ENV (environment variable)
- RUN (any arbitrary shell command)
- EXPOSE (open port from container to virtual network)
- CMD (command to run when container starts)
- docker image build (create image from Dockerfile)

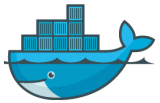
Chapter: Persistent Data



Container lifetime and data

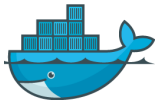
- Containers are usually meant to be immutable and ephemeral
- Immutable == unchanging
- Ephemeral == temporary or throwable
- Immutable infra – only redeploy containers

- Currently data is present as long as the container is not destroyed
- Persistent data can be achieved by two ways
  - 1. Volume
  - 2. Bind Mounts



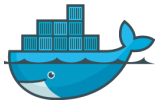
Volume and Bind Mounts

- Volumes : special location outside of container UFS
- Bind Mounts :
  - Sharing or
  - Link container path to host path



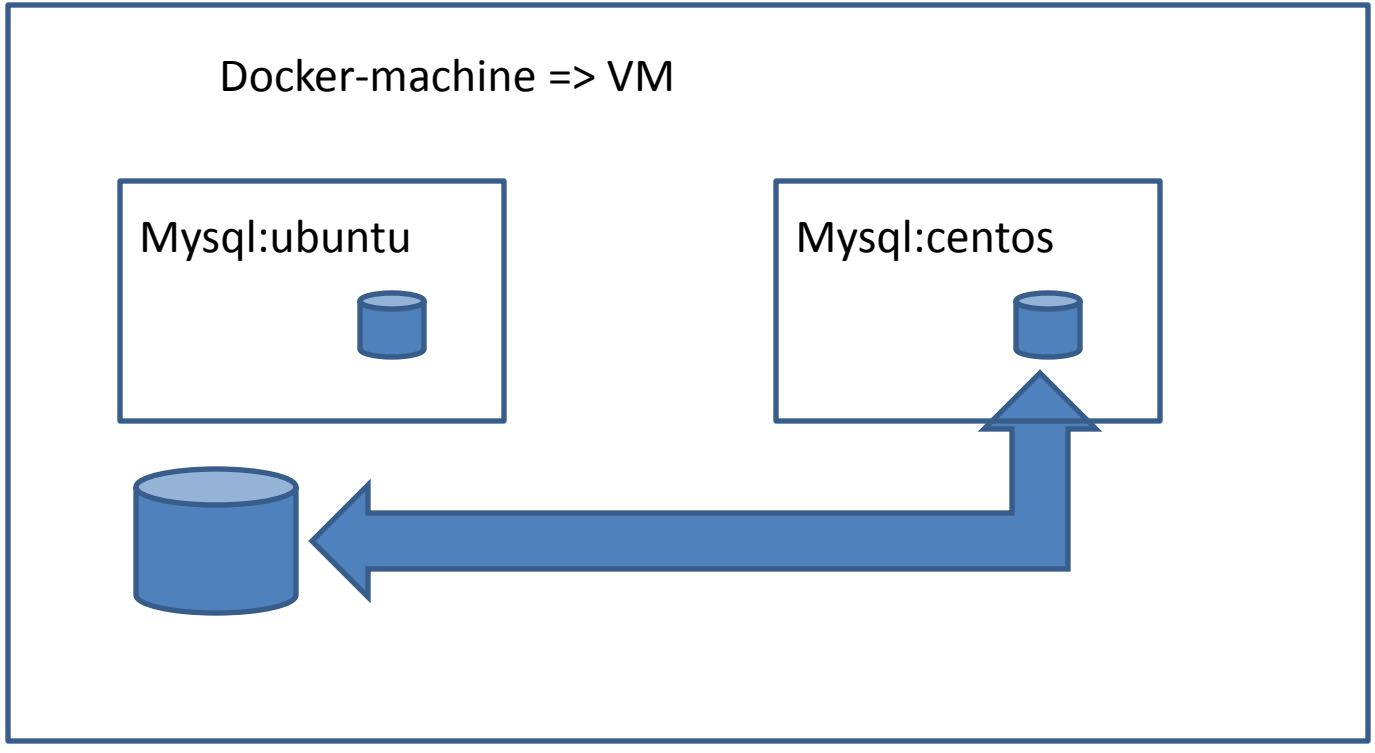
Volume

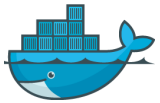
- VOLUME command in the Dockerfile
- Override with docker run –v /path/in/container
- Bypasses the Union File System and stores in the alt location on host
- Includes its own management commands under docker volume
- Connect to none, one or multiple containers at once
- Not subject to commit, save or export commands
- They have a unique id. But if you assign a name its then a named volume



Bind mounting

- Mapsa host file or directory to a container file or directory
- Basically just two locations pointing to the same file
- Skips UFS and host files overwrite any in container
- Not a Dockerfile code. It has to be mentioned during the container run


Docker-machine => VM
Mysql:ubuntu
Mysql:centos



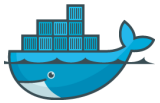
Volume and Bind Mounts

- docker container run -d --name mysql -e MYSQL_ALLOW_EMPTY_PASSWORD=True mysql
- docker volume ls
- docker volume inspect
- docker container run -d

--name mysql -e MYSQL_ALLOW_EMPTY_PASSWORD=True

-v mysql-db:/var/lib/mysql mysql

Chapter: Docker Compose

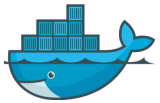


Why?

- Configure relationships between containers
- Save our docker container run settings in easy to read file
- Create one-liner developer environment startups
- Comprised of
  - A YAML formatted file that describes
    - Containers
    - Networks
    - Volumes
  - A CLI took docker-compose used for local dev/test automation with YAML files

docker-compose.yml

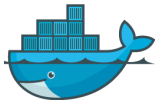
- Its own versions, 1,2,2.1,3,3.1
- YAML file can be used with docker-compose command for local docker sutomation
- docker-compose.yml is default name but can be changed



docker-compose CLI

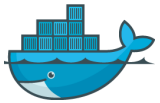
- CLI tool comes with docker (has to be downloaded for linux)
- Not really production grade but ideal for dev and test
- Two most common commands
  - docker-compose up
  - docker-compose down
- Very easy for developer onboarding

Docker



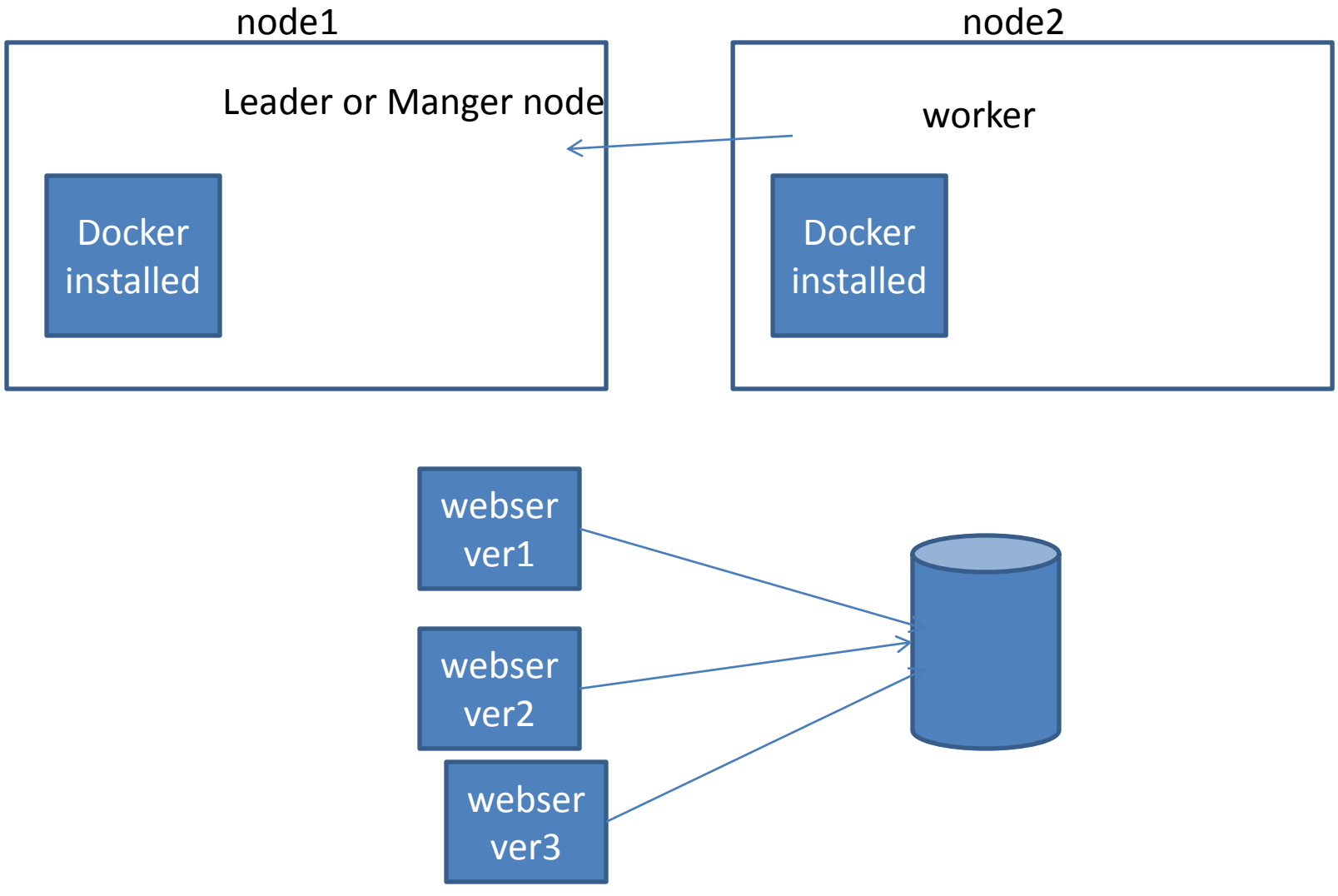
```
version: '3.1'

services:
  servicename: nginx
    image: nginx
    volumes:
        - .:/usr/share/nginx/html
    ports:
        - '8095:80'
```

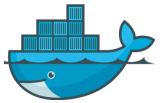
Chapter: Swarm - Introduction



Swarm

- Automate container lifecycle
- Scale out/in/up/down
- Recreate containers if they fail –resilience
- Blue/green deploy
- Cross-node virtual networks
- Run containers on trusted servers
- Ability to store secrets, keys, passwords

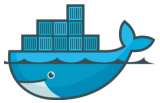

node1
Leader or Manger node
Docker installed
node2
worker
Docker installed
webser ver1
webser ver2
webser ver3



Swarm Mode

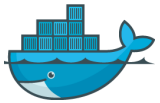
- Clustering solution built inside docker
- Not enabled by default
- New commands once enabled
  - Docker swarm
  - Docker node
  - Docker service
  - Docker stack
  - Docker secret
- docker swarm init => to enable swarm

Chapter: Kubernetes



Kubernetes

- Opensource orchestration system for Docker containers
- Schedule containers on a cluster of machines
- Run multiple containers
- Run long running services
- Kubernetes will manage the state of these containers
  - Start on specific nodes
  - Restart a container when it gets killed
- Can manage one to 1000’s of nodes



Running Kubernetes

- Minikube – is a tool that makes it easy to run Kubernetes locally
- Minikube runs a single node k-cluster inside a linux vm
- Its aimed for dev and testing