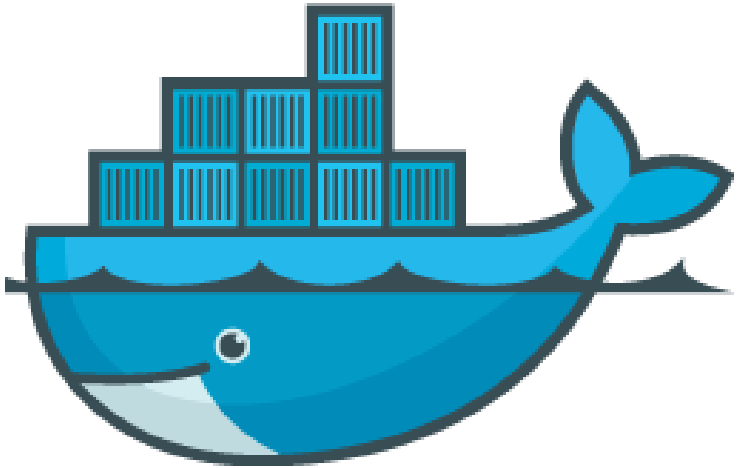


Docker

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

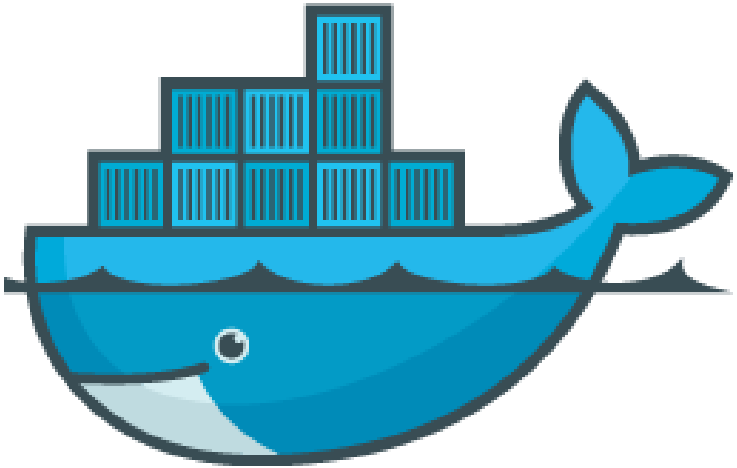
Interesting Things about me:

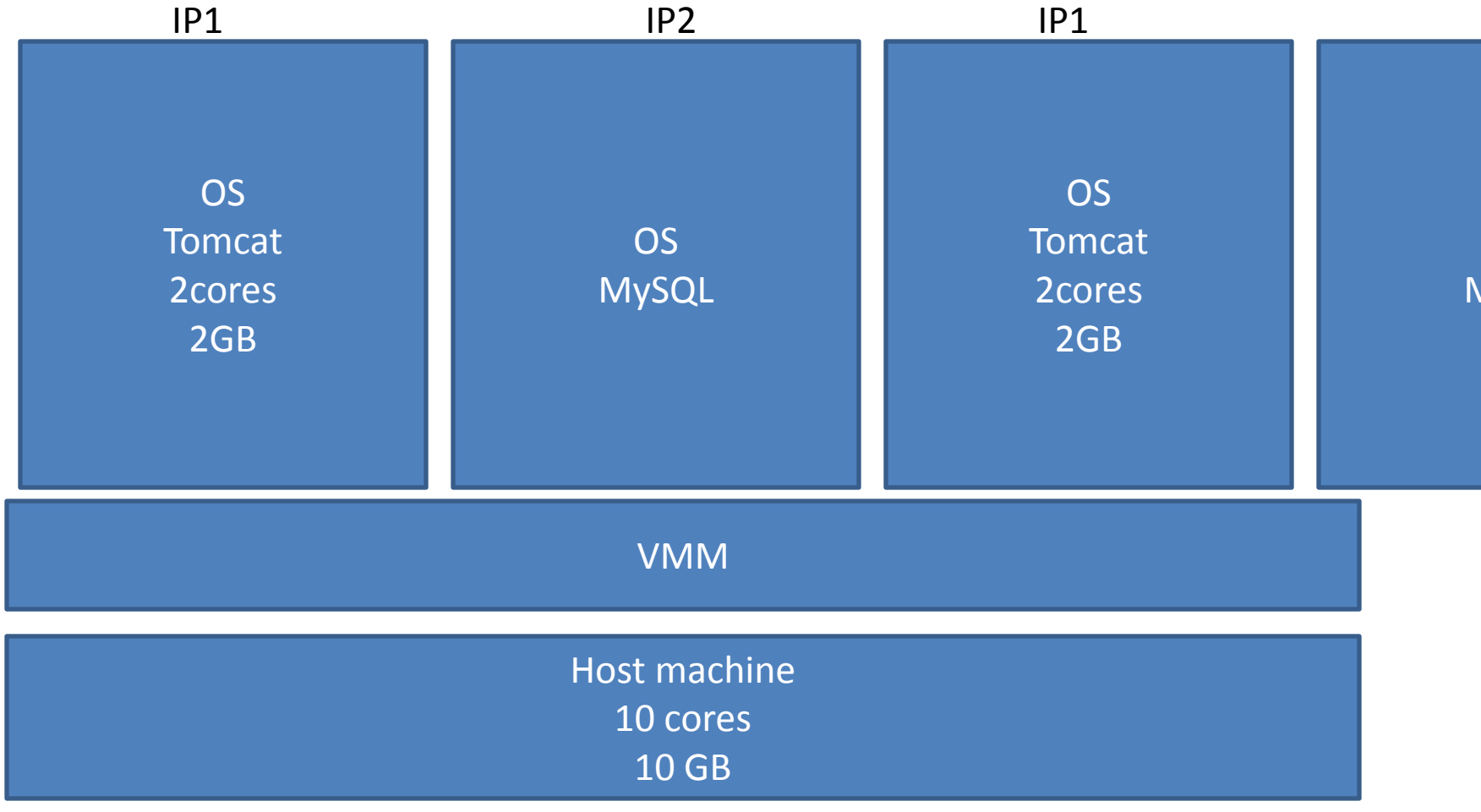
- Actually Nothing !

Getting to know you

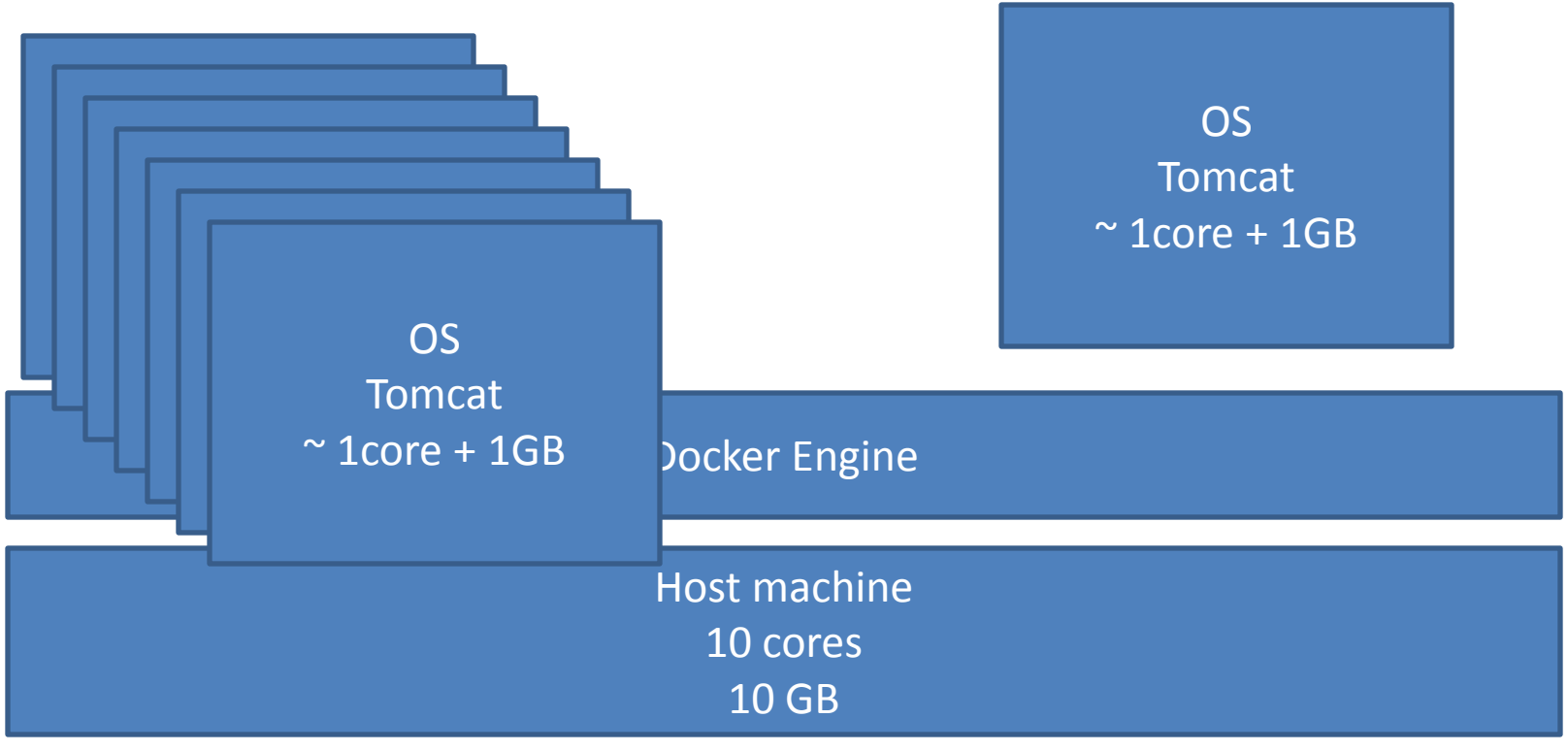
Agenda

- Introduction to DevOps
- Docker



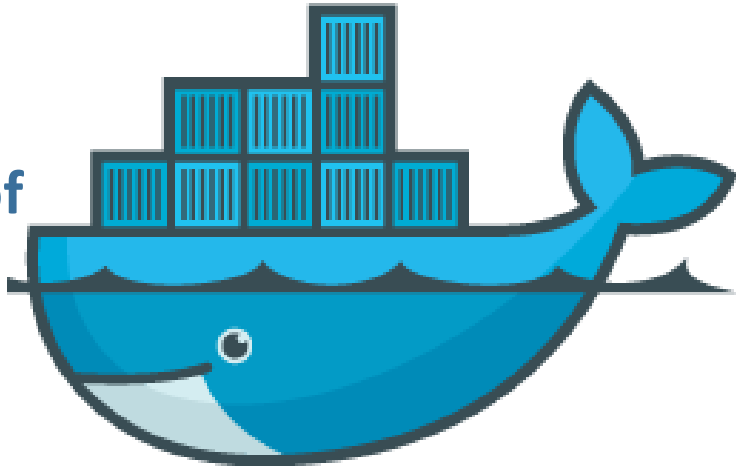

IP1
IP2
IP1
OS
Tomcat
2cores
2GB
OS
MySQL
OS
Tomcat
2cores
2GB
VMM
Host machine
10 cores
10 GB

container

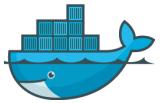

OS
Tomcat
~ 1core + 1GB
OS
Tomcat
~ 1core + 1GB
Docker Engine
Host machine
10 cores
10 GB

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

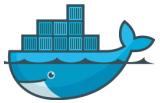


Chapter: Introduction



Docker – Why Now?

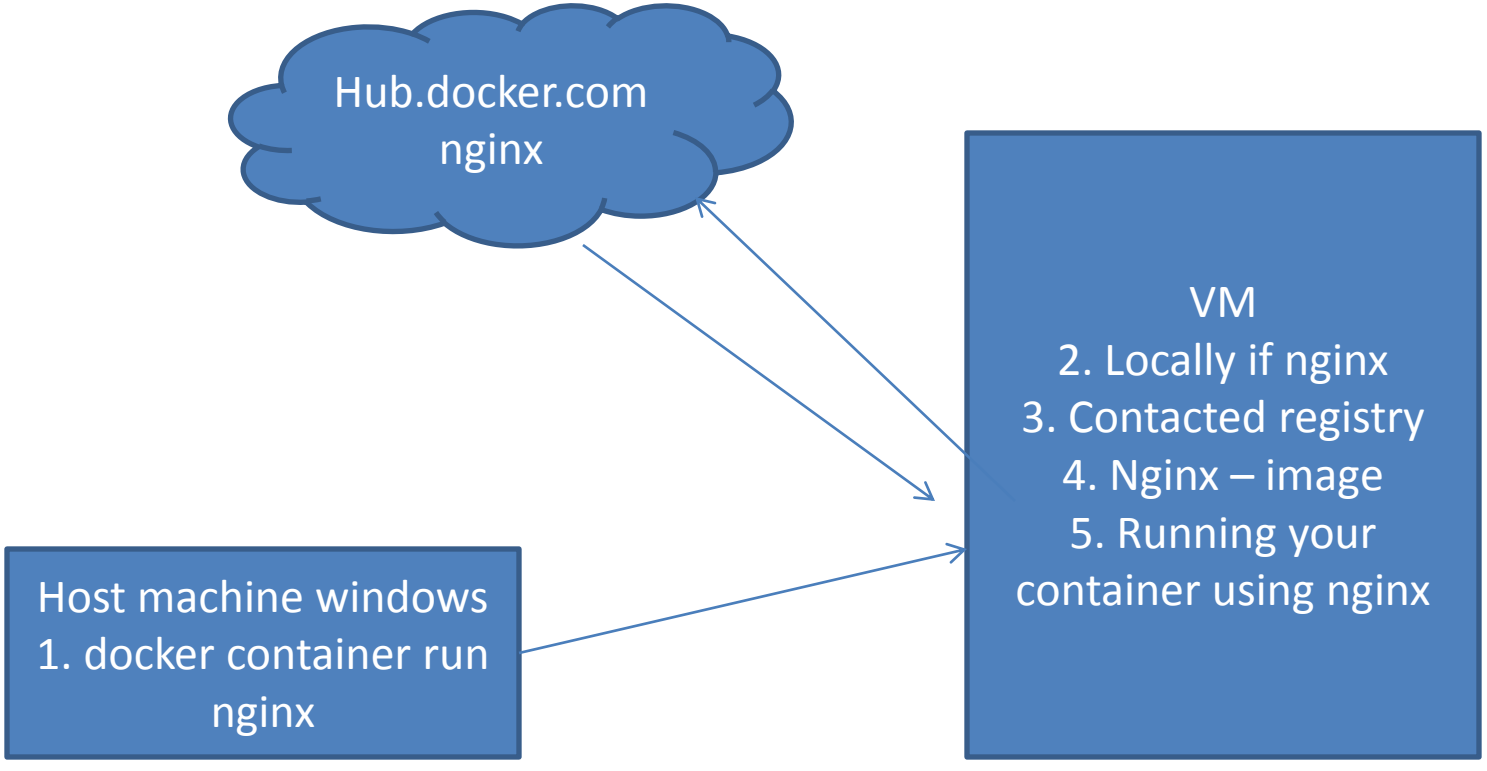
- Speed. Speed. Speed.
- Value movement dev-> test-> prod easier and faster
- Portability
- Reduce complexity of developing code for distributed systems
- Reduce complexity of deploying code to the cloud
- For a later time - Docker's founder and CTO Solomon Hykes
- https://www.youtube.com/watch?v=3N3n9FzebAA

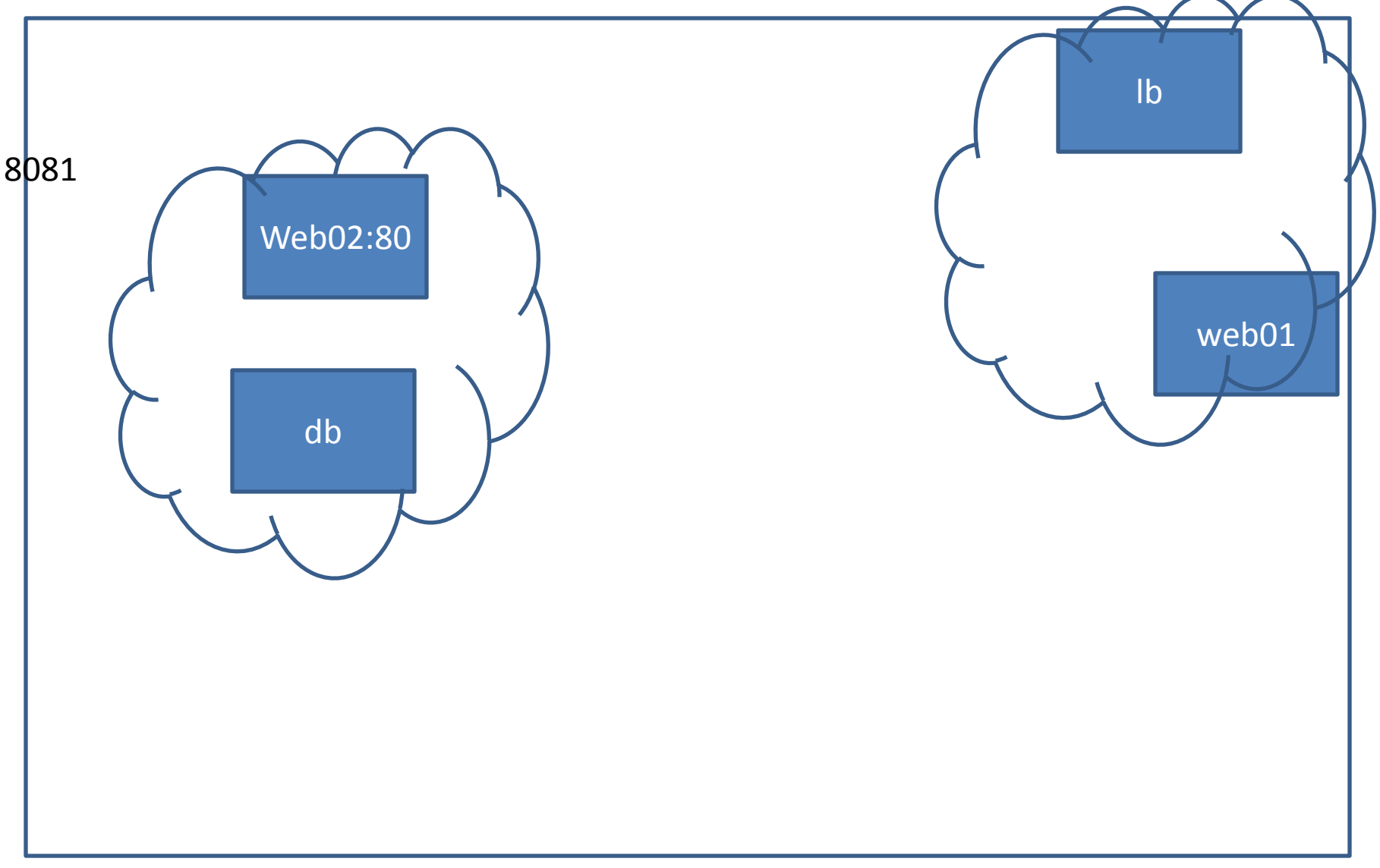


Docker – Different Versions

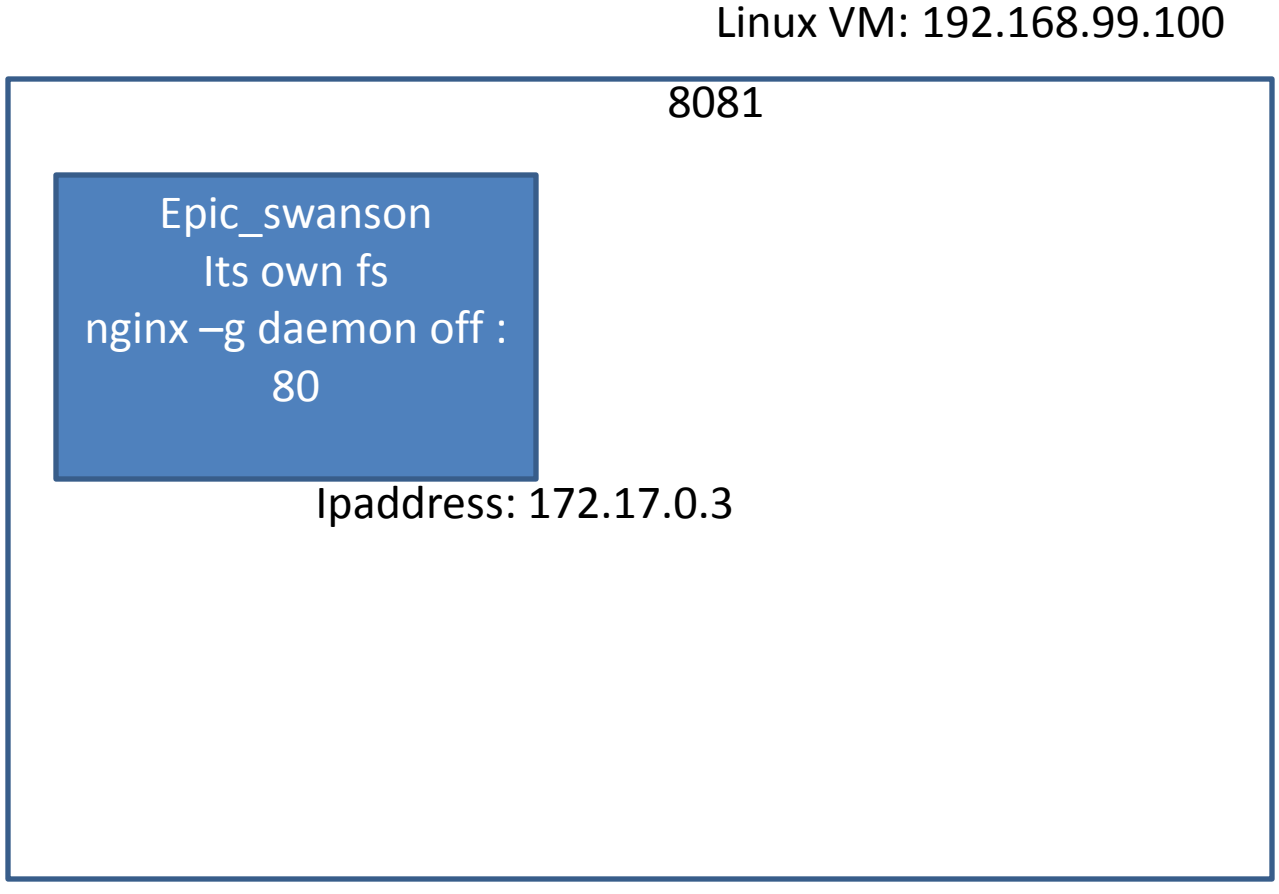
- https://www.docker.com/get-docker
- Community Edition and Enterprise Edition
- Stable and Edge

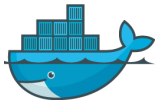
- Stable vs. Edge Cont.
- Edge (beta) released monthly,
- Stable quarterly
- Edge gets new features first, but only supported for a month
- Stable rolls in three months of Edge features


Hub.docker.com
nginx
VM
2. Locally if nginx
3. Contacted registry
4. Nginx – image
5. Running your container using nginx
Host machine windows
1. docker container run nginx


8081
Web02:80
db
lb
web01

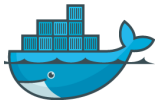
Windows :


Linux VM: 192.168.99.100
8081
Epic_swanson
Its own fs
nginx –g daemon off :
80
Ipaddress: 172.17.0.3



Docker – Setup

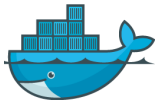
- Docker toolbox install



Docker – initial commands

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

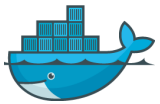
Chapter: Container



Container

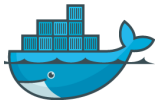
- Basic Building block
- Let us get a container running and then we will connect the dots
- Execute the command

```
docker container run nginx
```

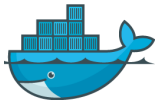


Container

- 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

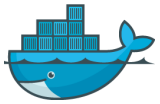


First Container Run:What just happened?



Knowing more about a Container

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

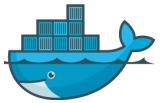


Interactive Container

- 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?

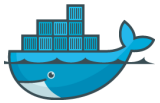
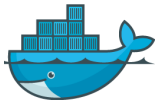


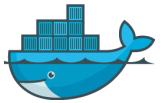
Image vs Container

- 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”



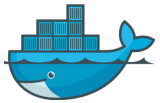
Introduction to docker hub

- 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



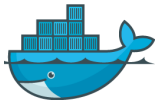
docker container run

- 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



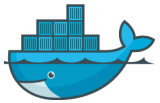
Working with images

- 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

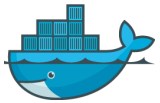
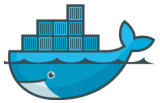


Image and push

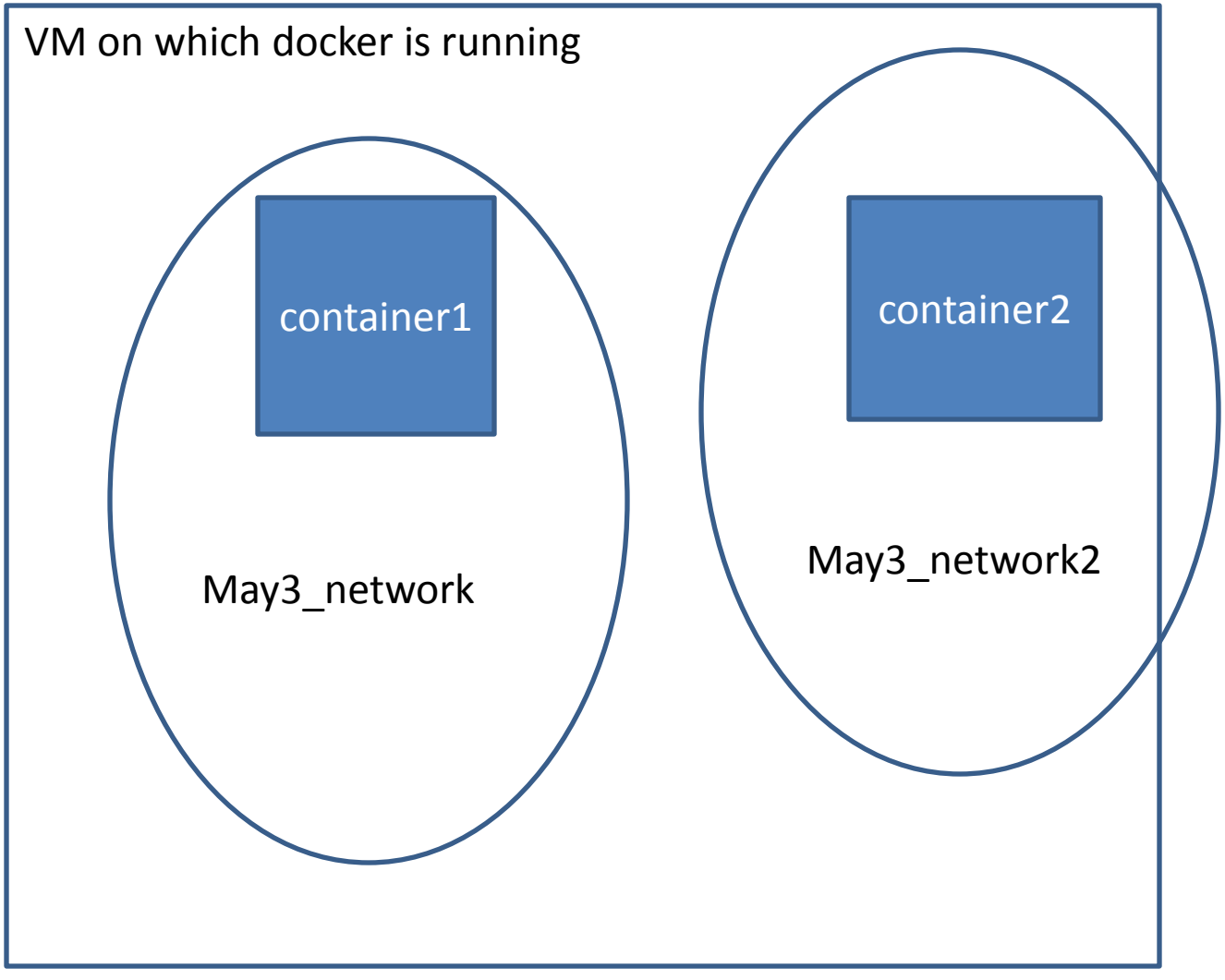
- An image has no real name as such
- It is uniquely identified through user/image:tag
- I can retag an existing image and push to my repository
- Only official images do not have username every other image has a user id behind it

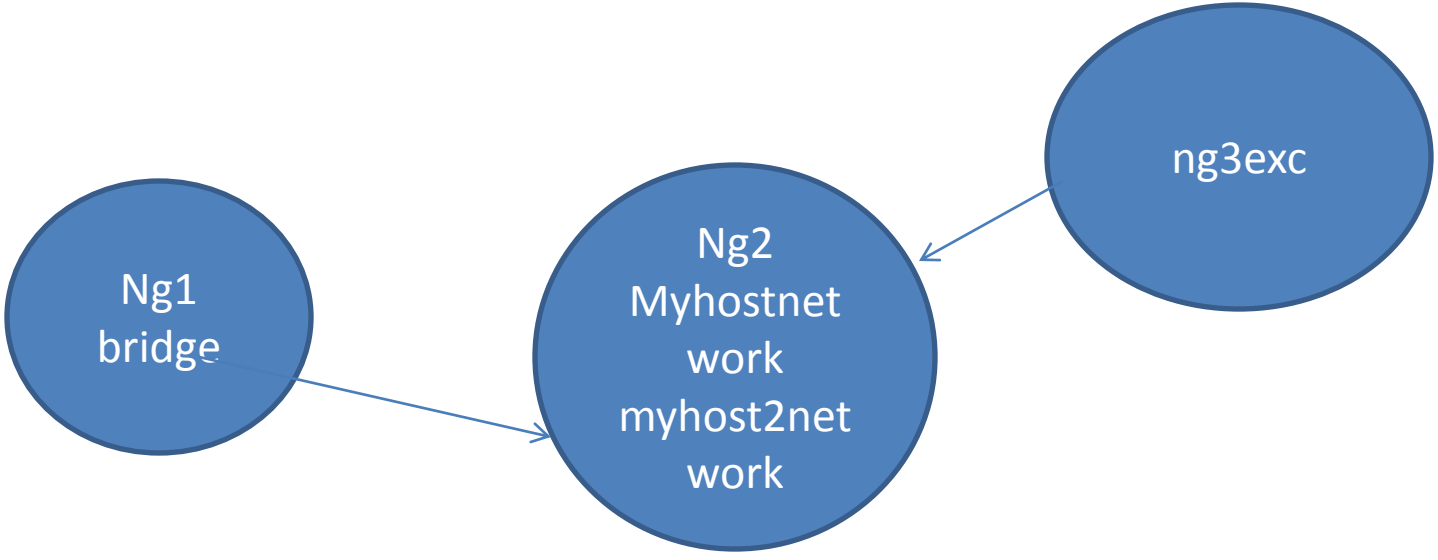
Chapter: Networking



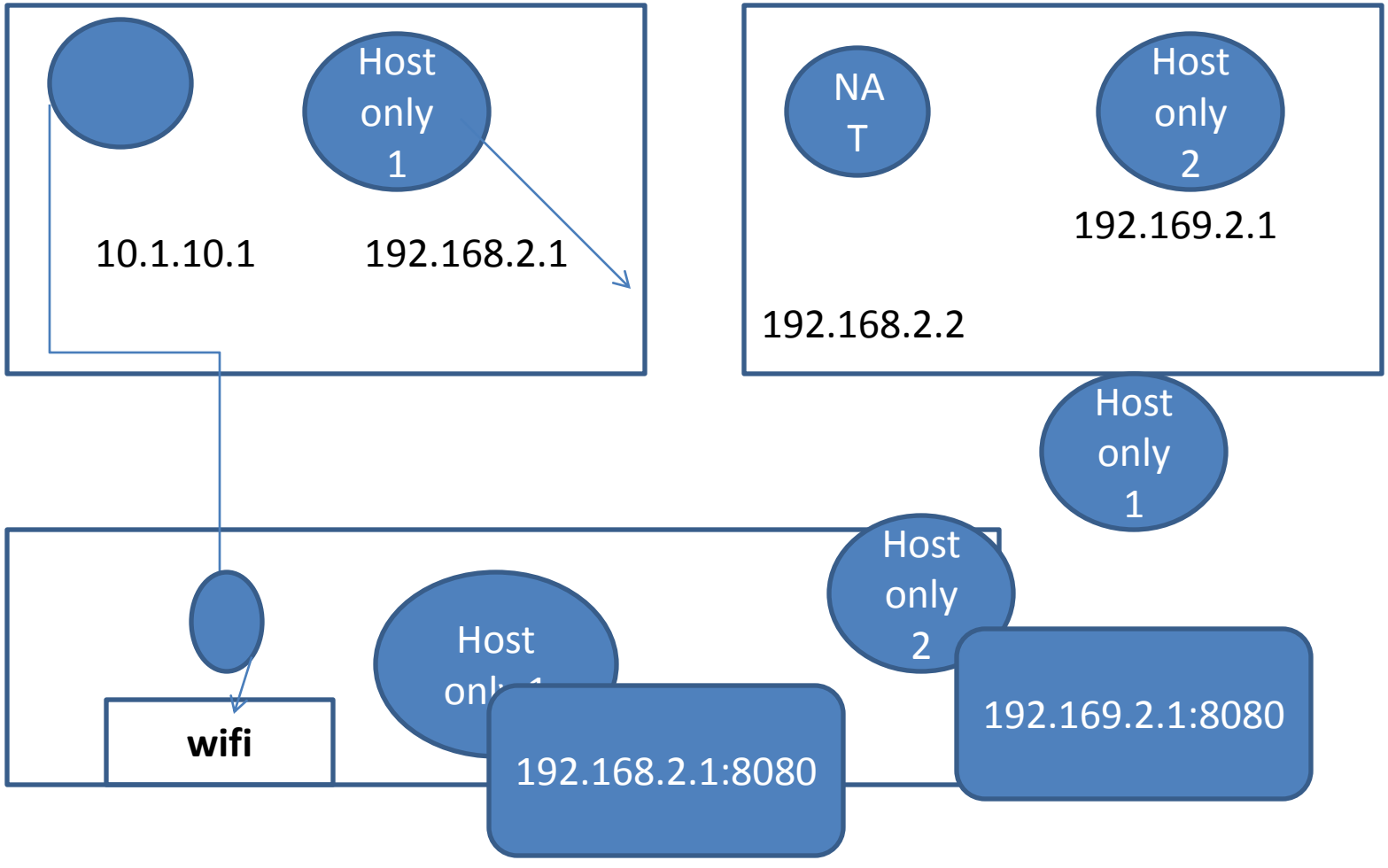
Container Network

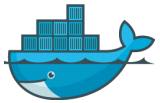
- An image has no real name as such
- Each container connected to a private virtual network "bridge"
- Each virtual network routes through NAT firewall on host IP
- All containers on a virtual network can talk to each other without -p
- Best practice is to create a new virtual network for each app:
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

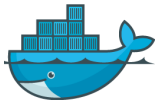

Host only 1
10.1.10.1
192.168.2.1
NA T
Host only 2
192.169.2.1
192.168.2.2
Host only 1
Host only 2
Host
wifi
192.168.2.1:8080
192.169.2.1:8080



Docker network commands

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

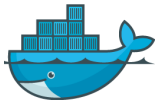
Chapter: Docker File



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