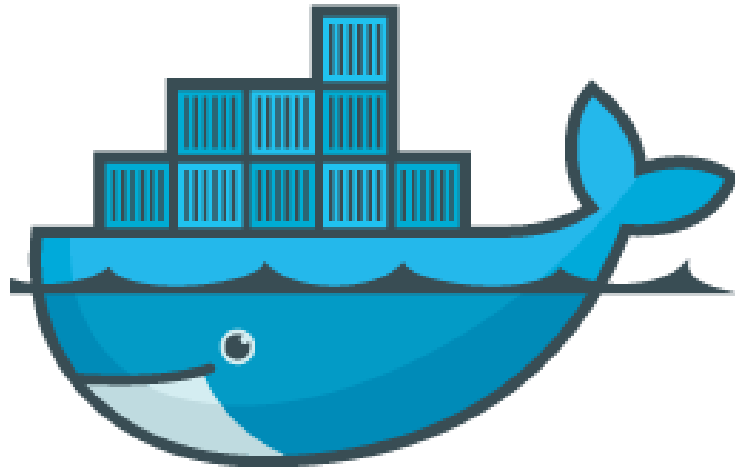


Docker



# Docker

By  
ADITYA PRABHAKARA



# Introducing Myself

**Aditya S P** ([sp.aditya@gmail.com](mailto: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 !

1. Minikube start
2. Downloads an iso
3. Talk to virtual box
4. Bring up a vm

Minikube: 192.168.99.100

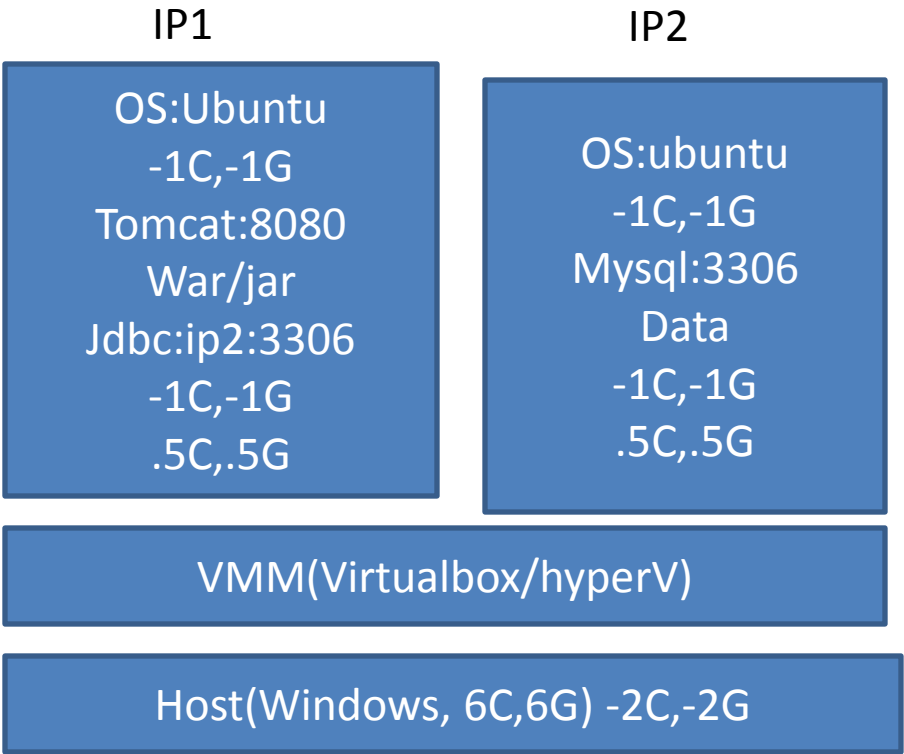
dockerd

5. Downloads local kube
6. Bring up docker and kubernetes

Virtual box(VMM)

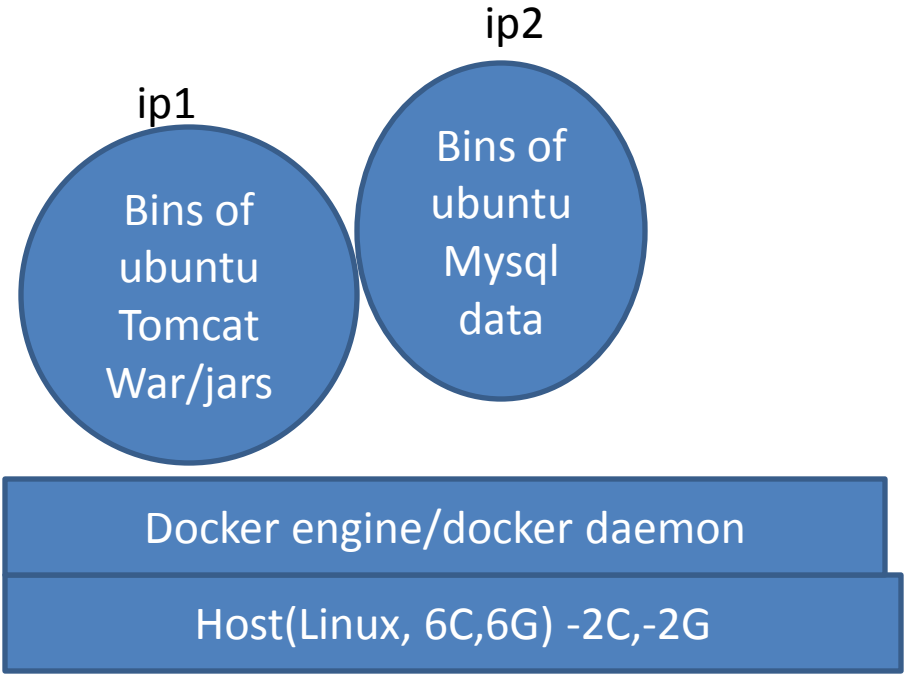
Host: window/osx

- 1. speed: mins
- 2. Sharability: snapshot/checkpoint, GBs
- 3. Repeatability: runsheets, scripts
- 4. Bang for the buck: limited,fragmented

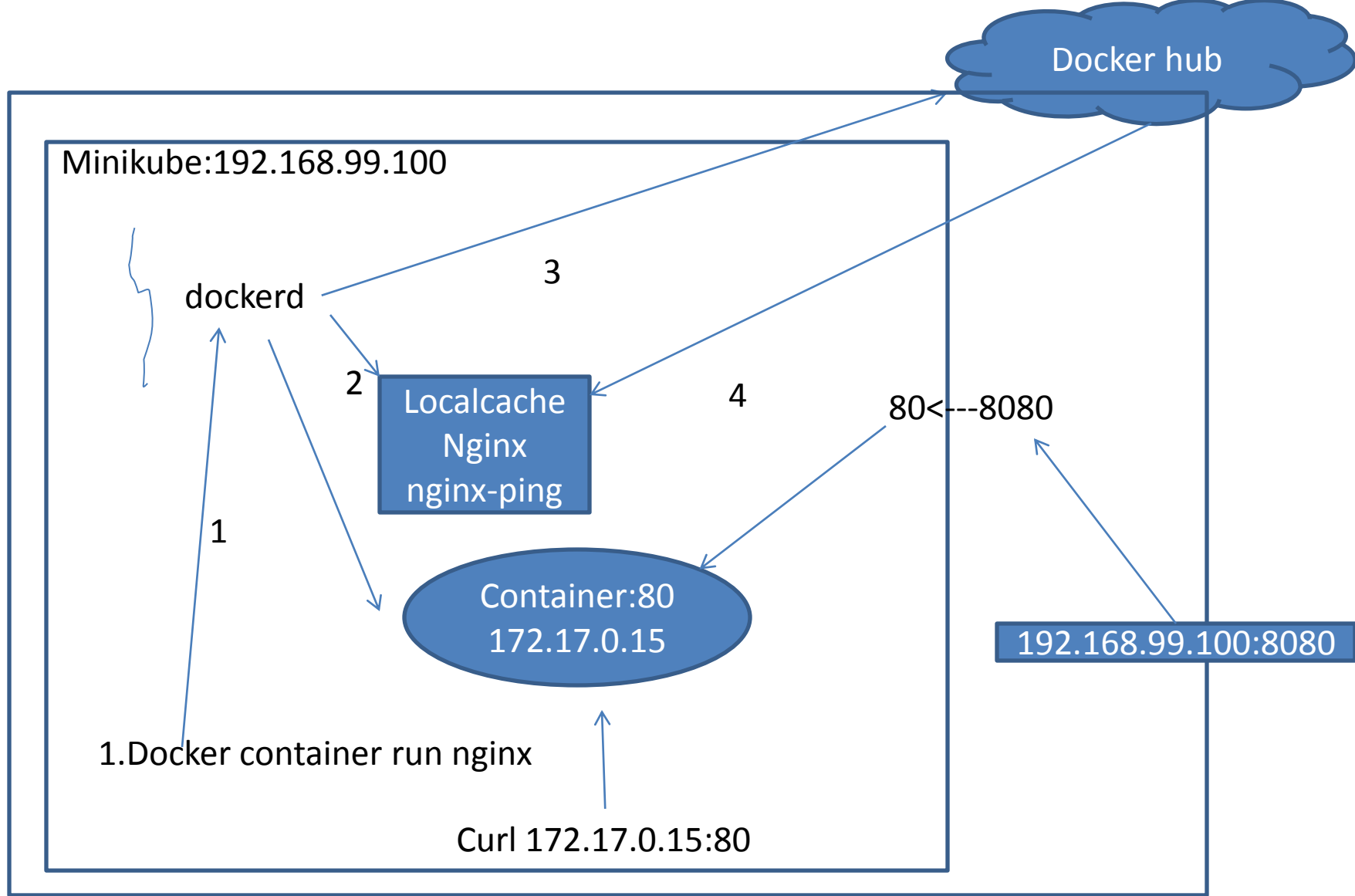


Filesystem,process space,n/w isolation

- 1. speed: ms
- 2. Sharability: image,MBs
- 3. Repeatability:dockerfile, DSL of docker
- 4. Bang for the buck: processes and defragmented

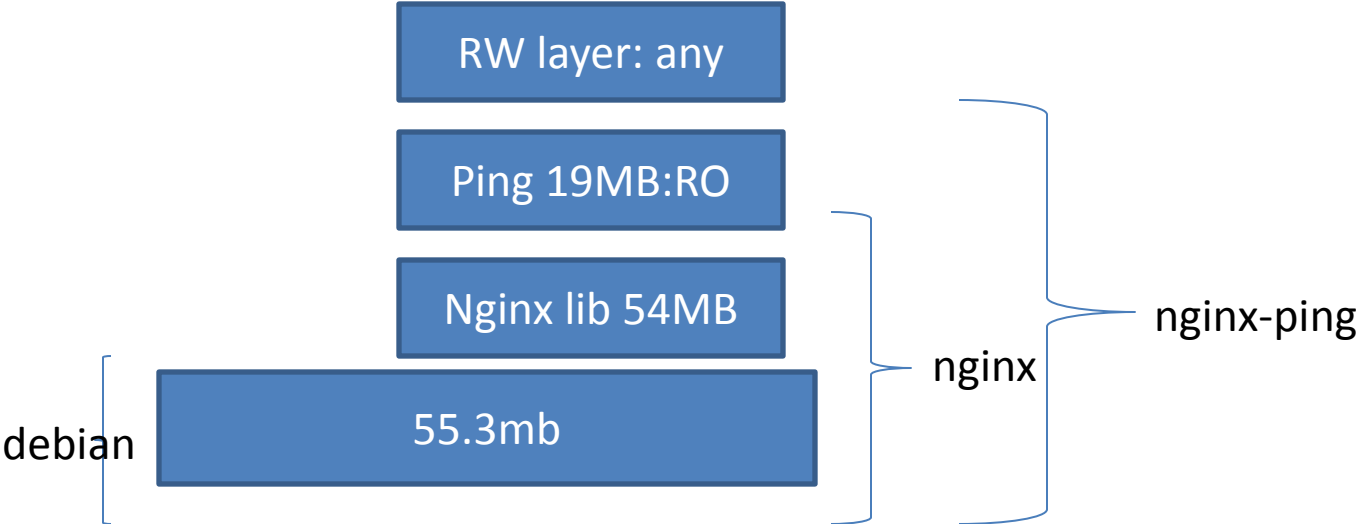


Filesystem,process space,n/w isolation

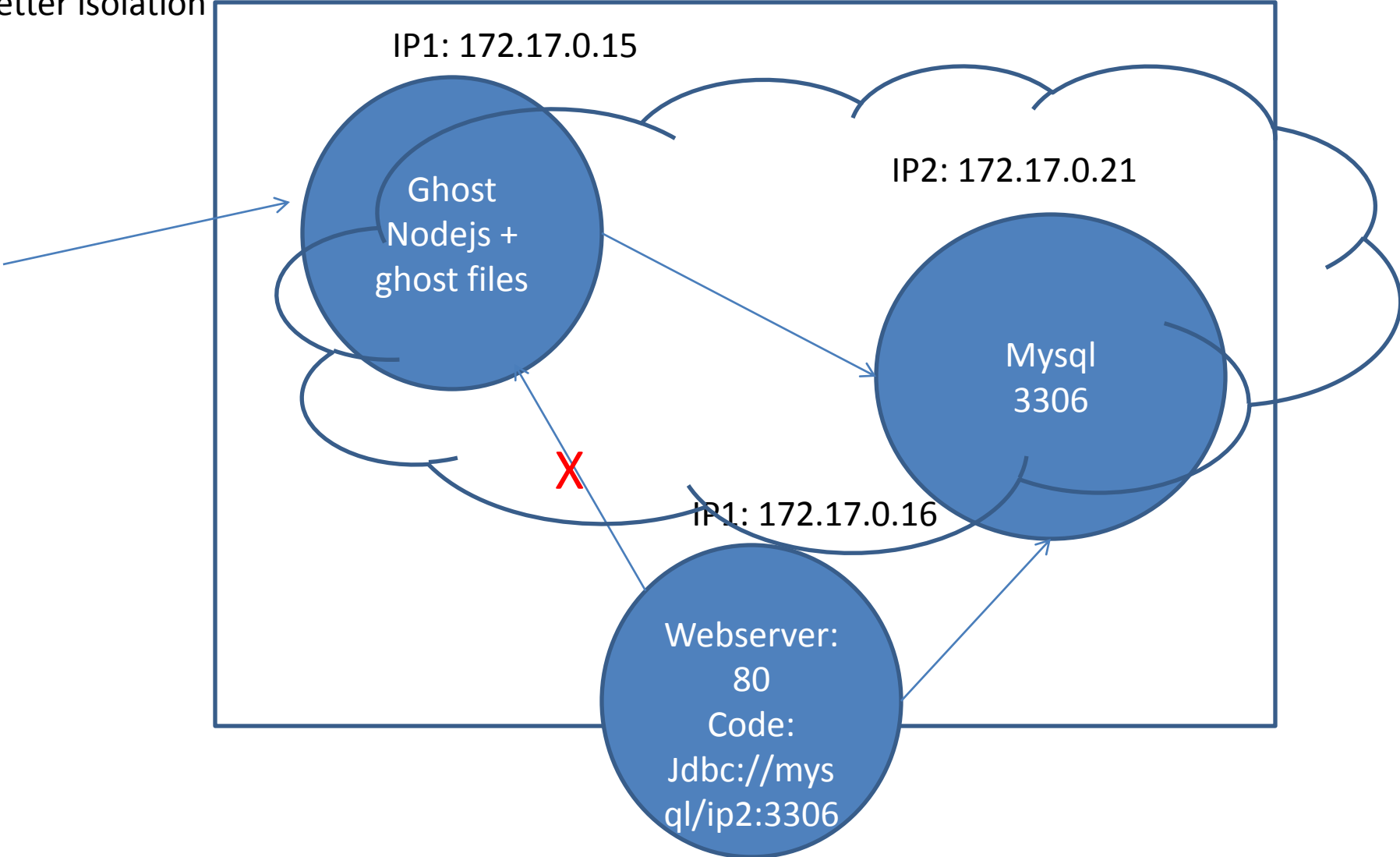


UFS: Unified File System

Nginx-ping



- 1. Dns name resolution
- 2. Better isolation



Bridge: 172.17.0.0/16

w2-w3-net: 172.19.0.0/16

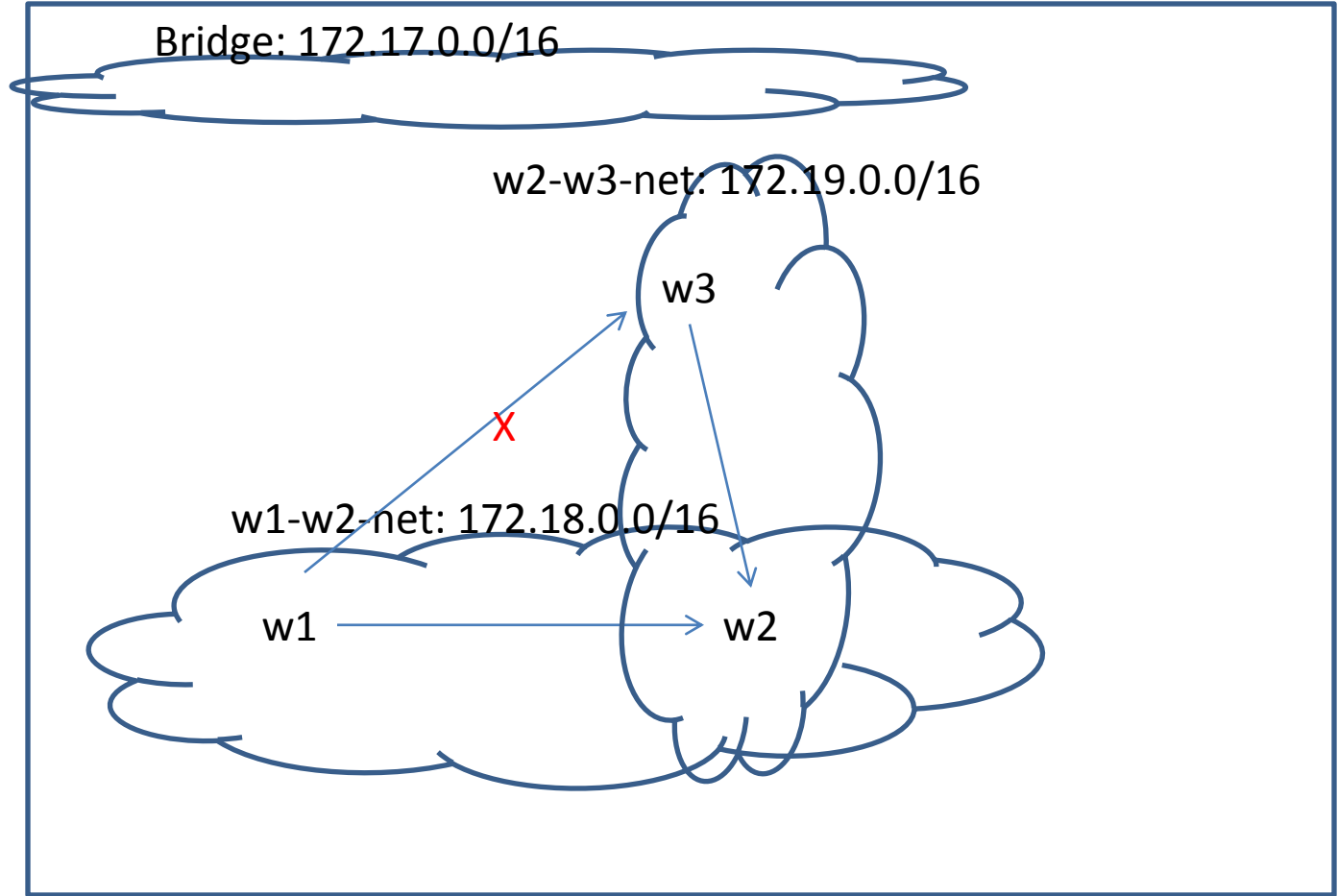
w3

X

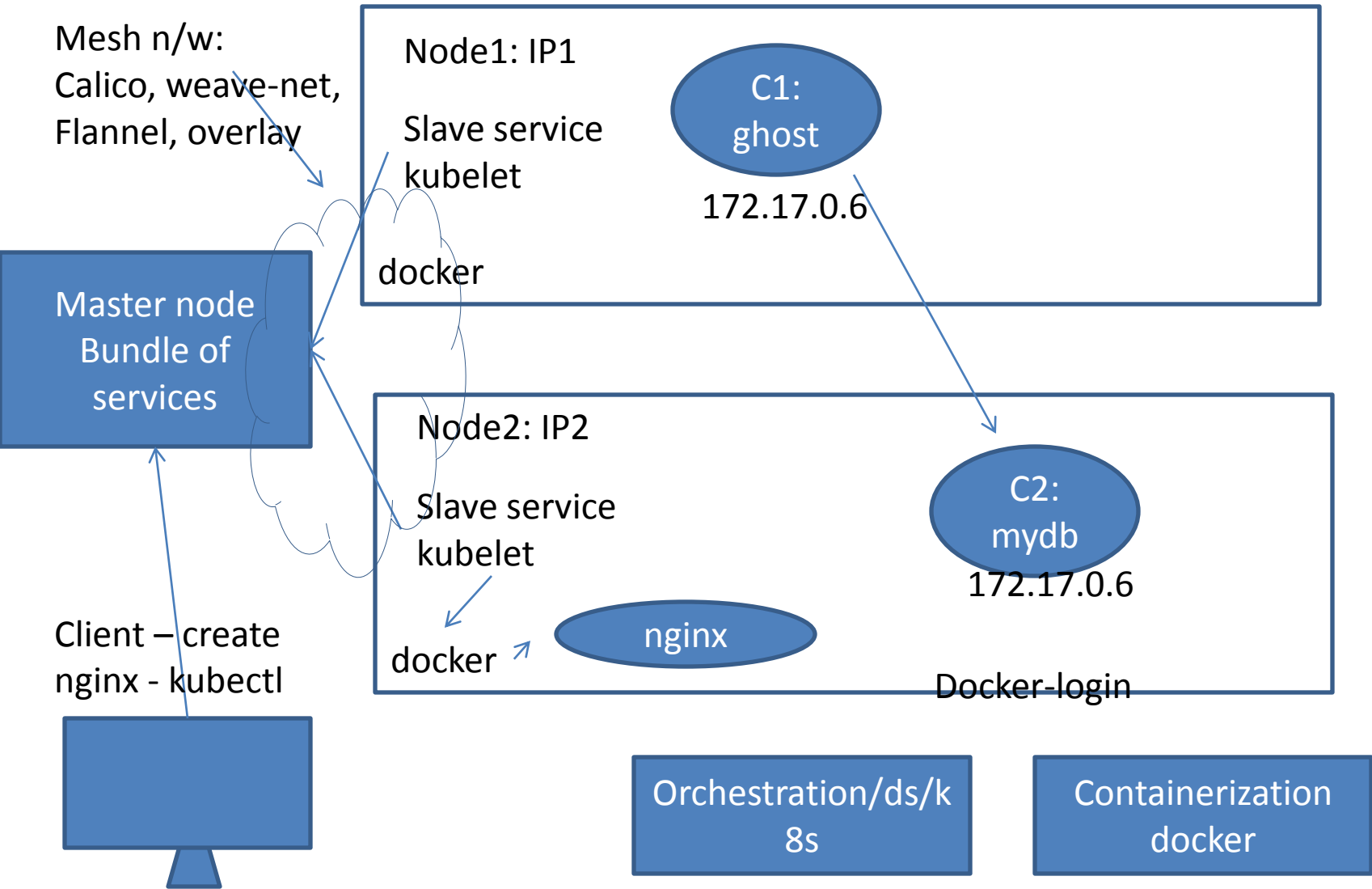
w1-w2-net: 172.18.0.0/16

w1

w2







User → IP3:80 → reverselookup → IP1:8080

User → IP3:80 → reverselookup → IP2:8080

lb.com → ip3

IP3:80

Loadbalancer  
Ha-proxy,  
nginx

Backend:

IP1:8080

IP2:8080

IP4:8080

lb.com:80

Ip3:80

Ip1:8080

server/  
Apache/tomc  
at

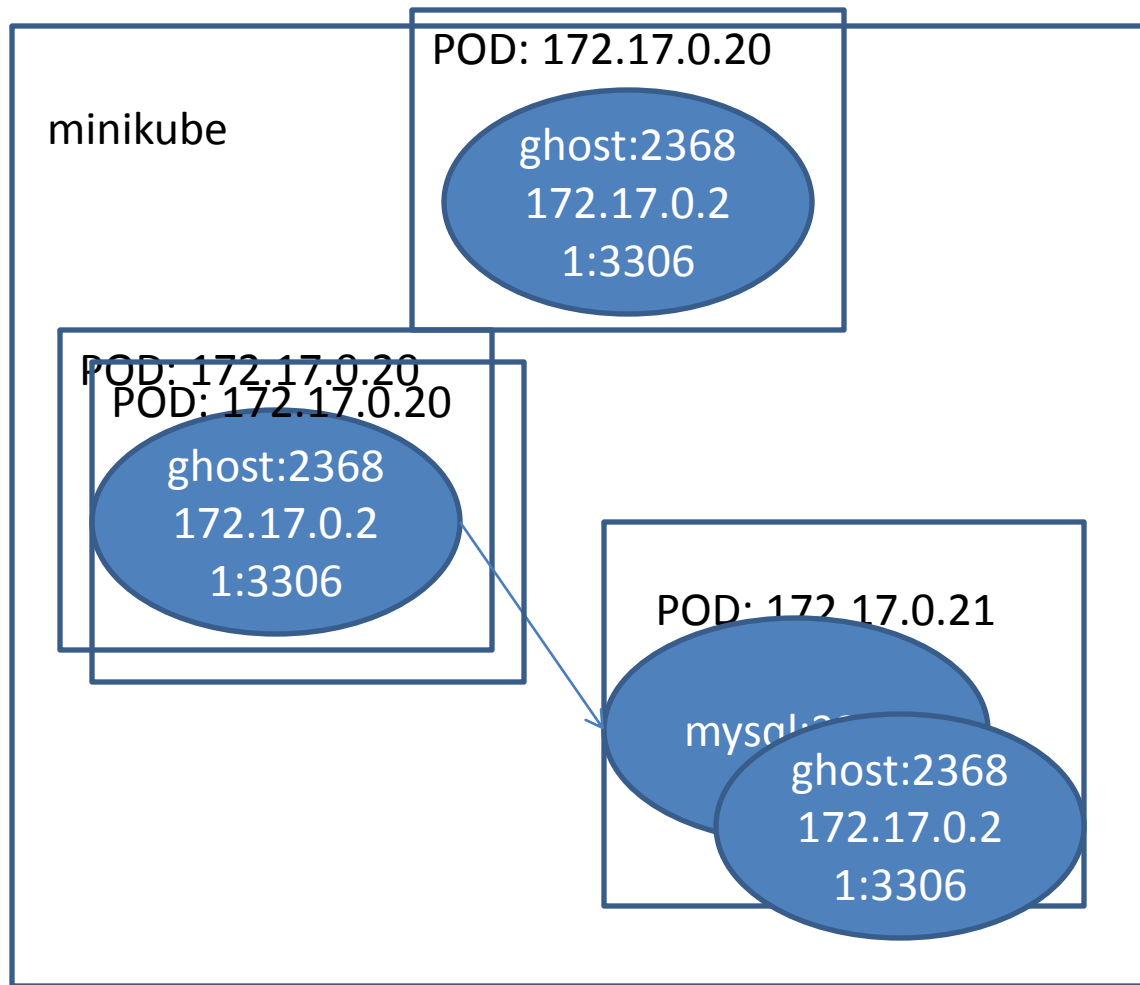
IP1:8080

server/  
Apache/tomc  
at

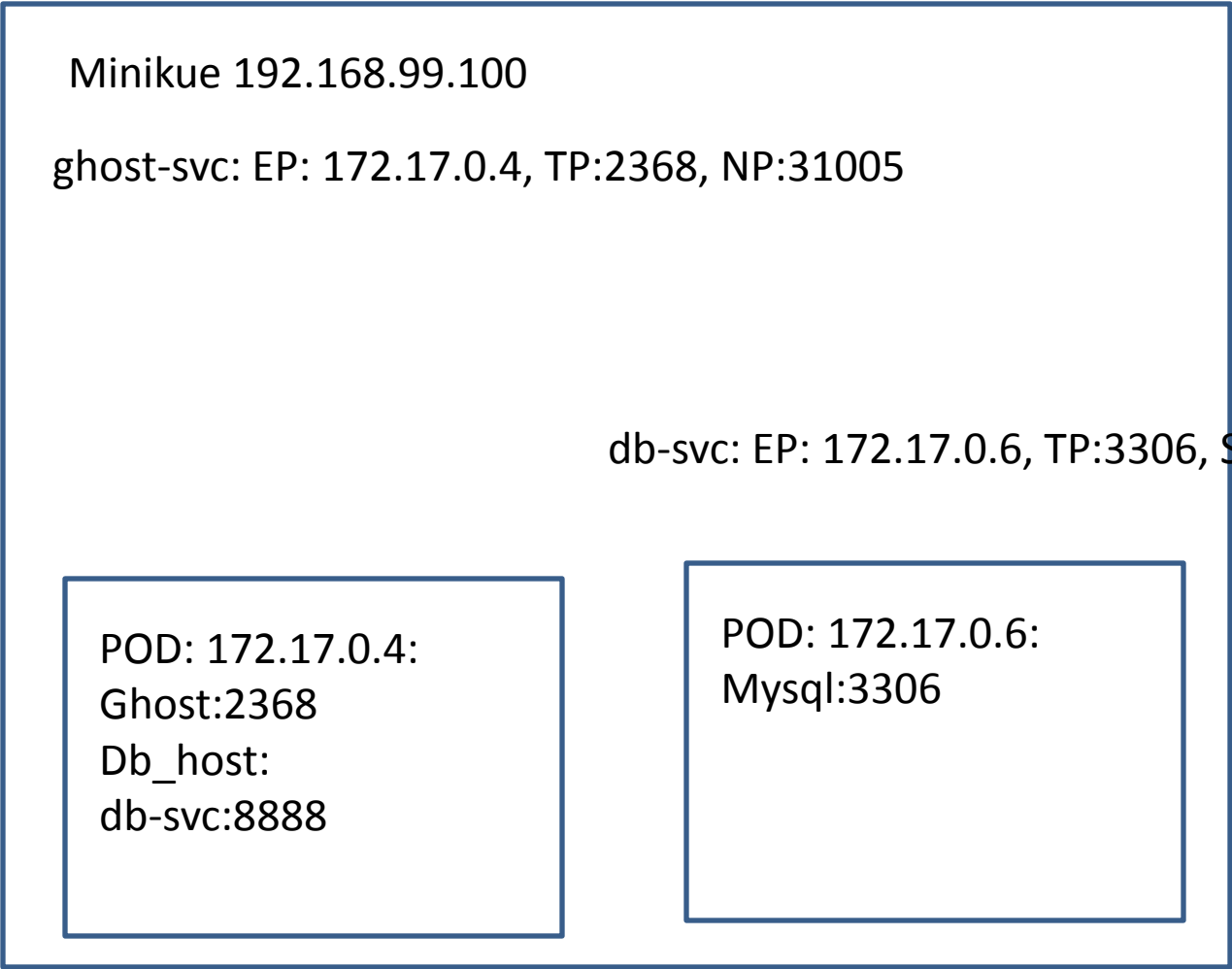
IP2:8080

server/  
Apache/tomc  
at

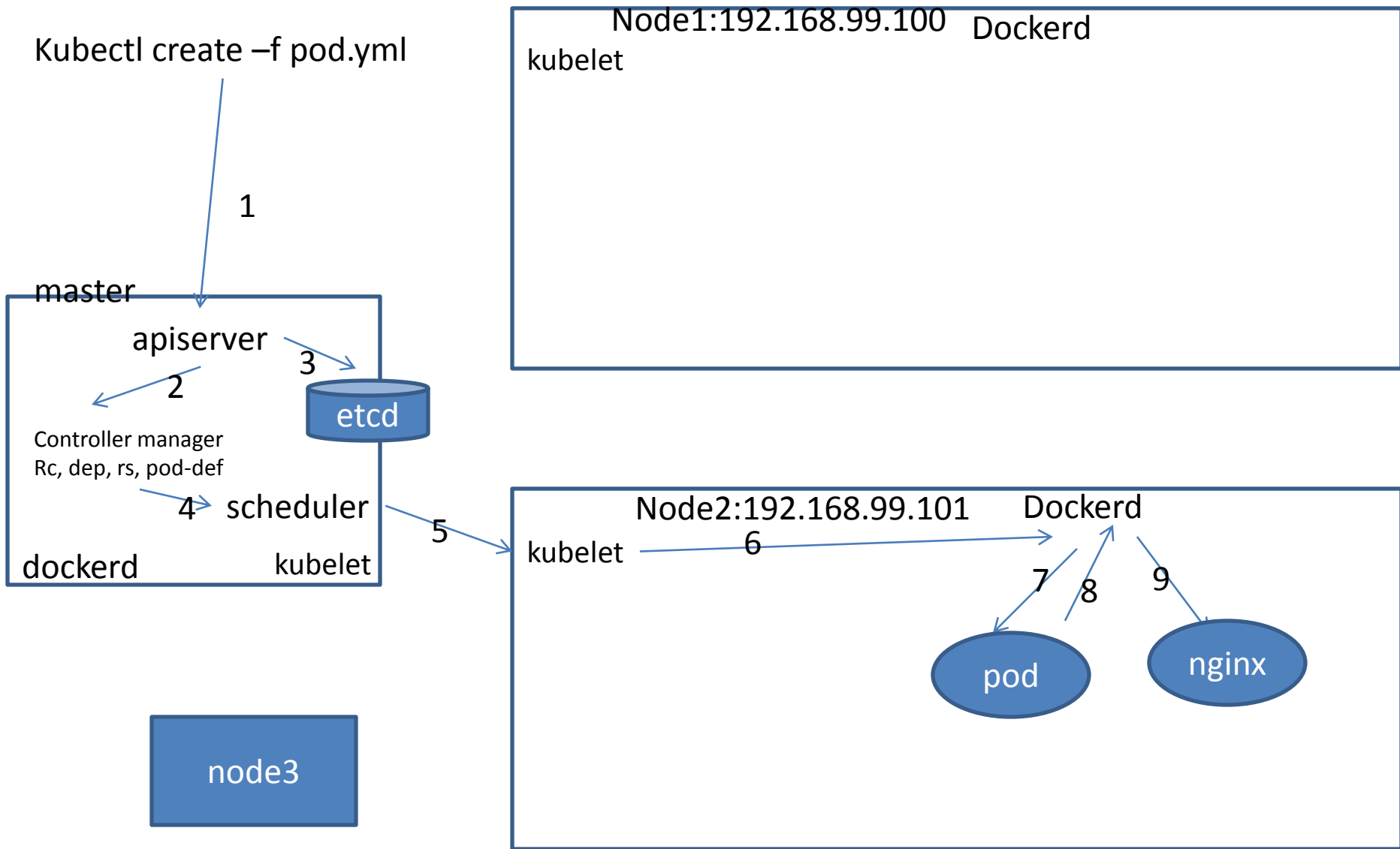
IP4:8080



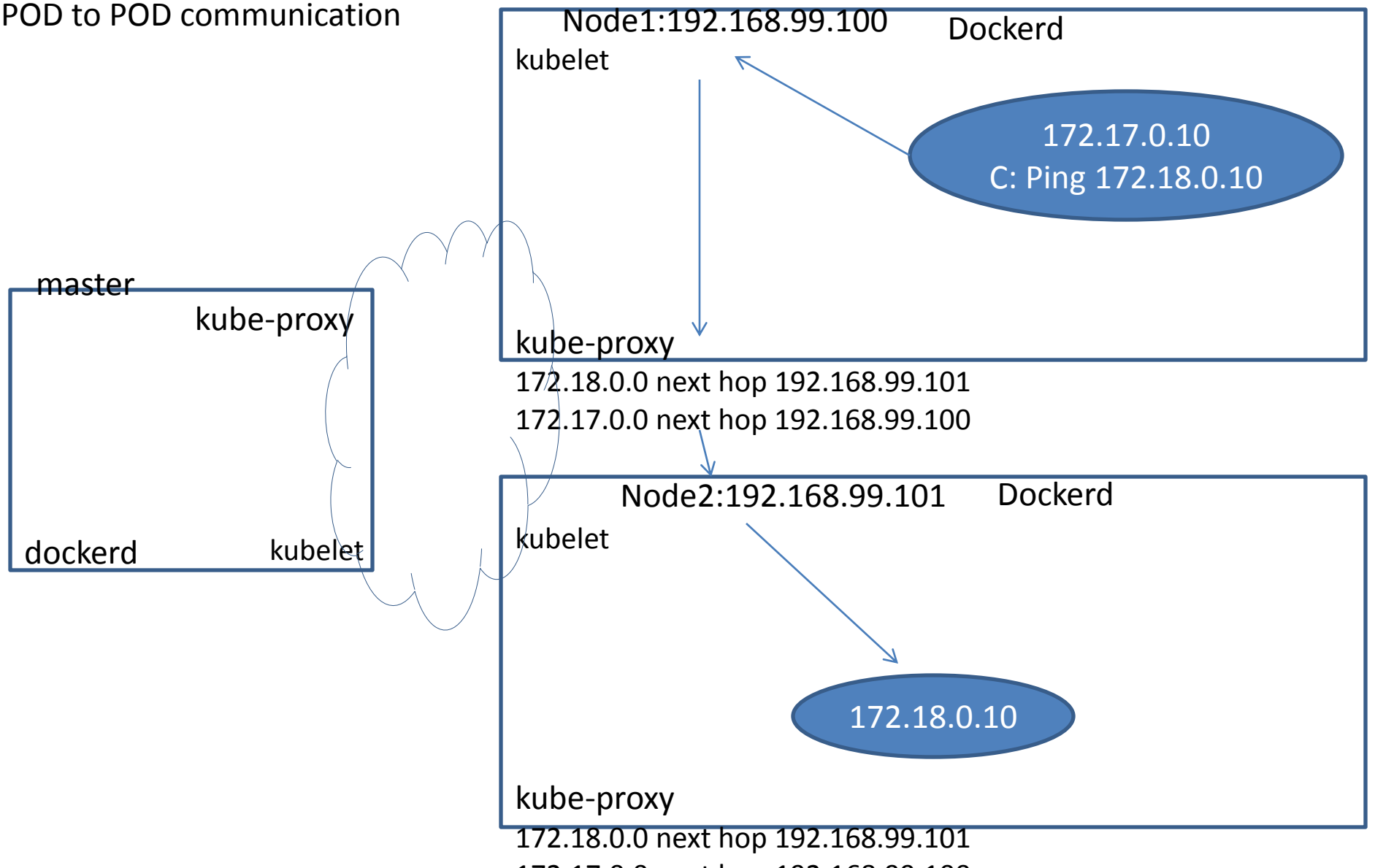
lb.com:80  
lp3:80  
lp1:8080  
  
db-svc:8888  
10.96.244.198:8888  
172.17.0.6:3306



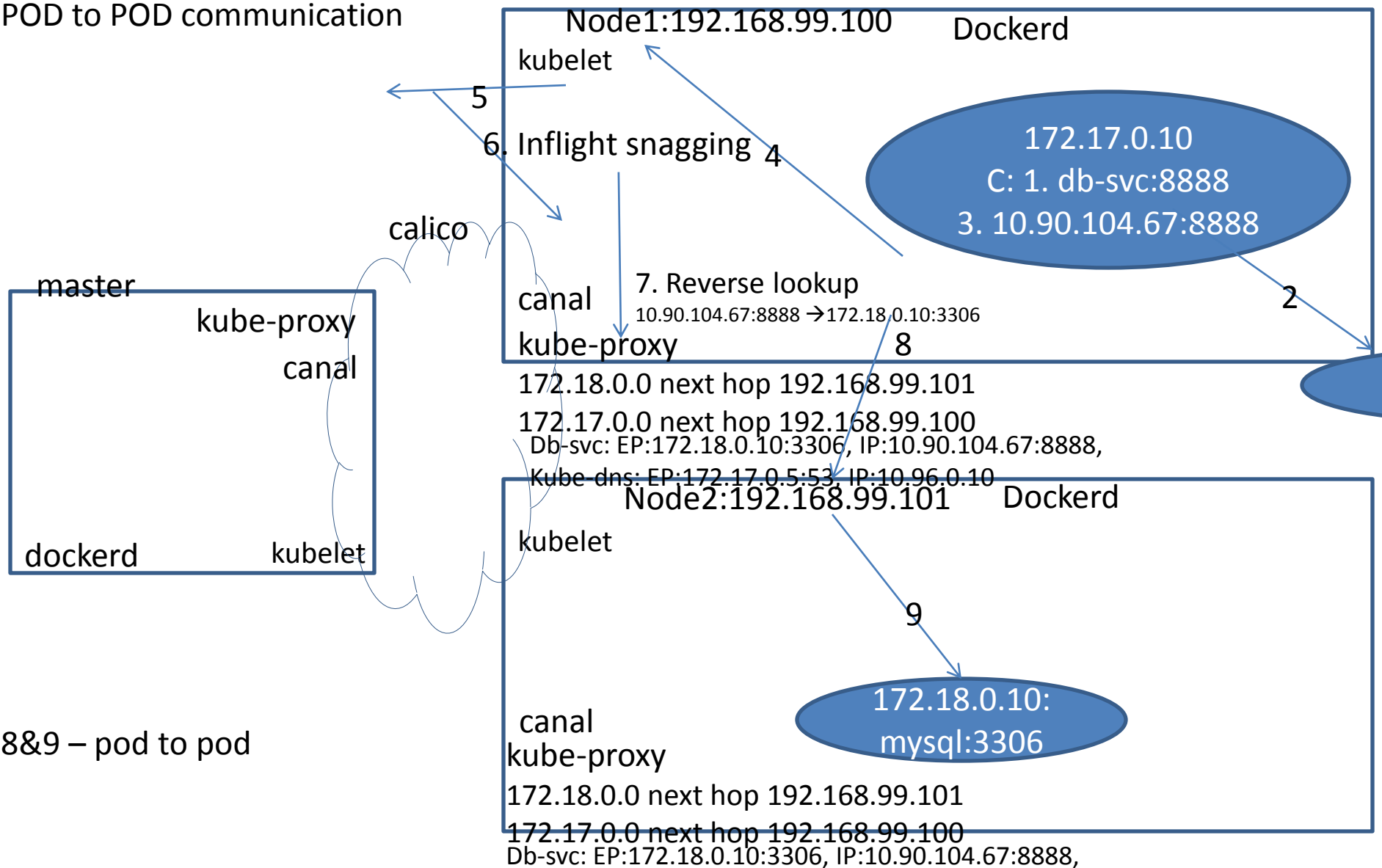
192.168.99.100:31005



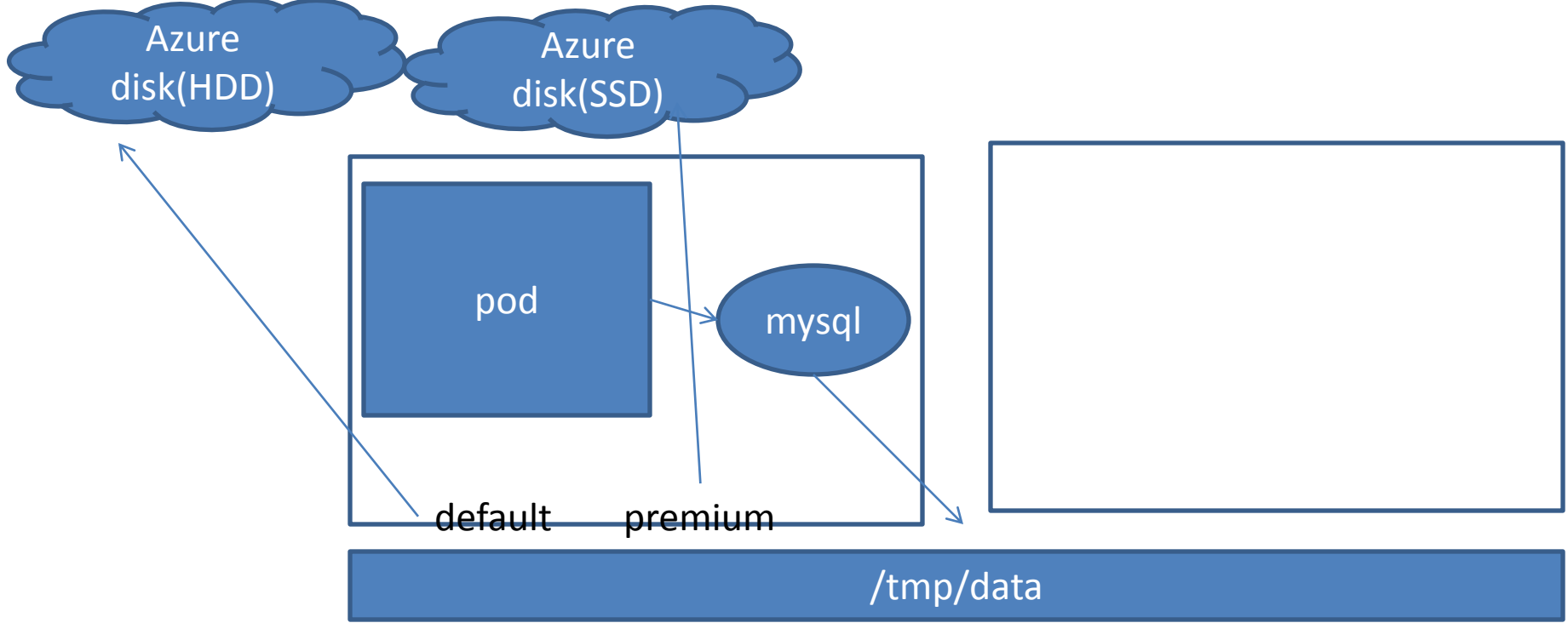
POD to POD communication



POD to POD communication



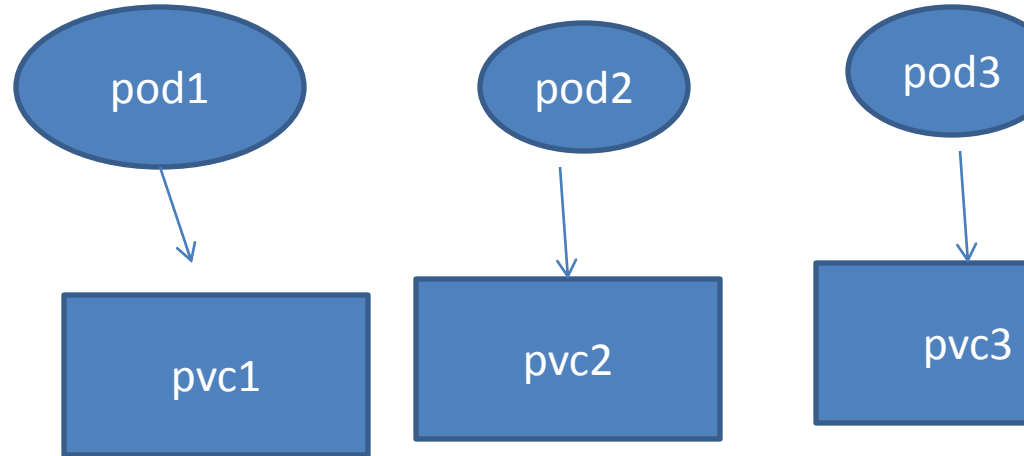
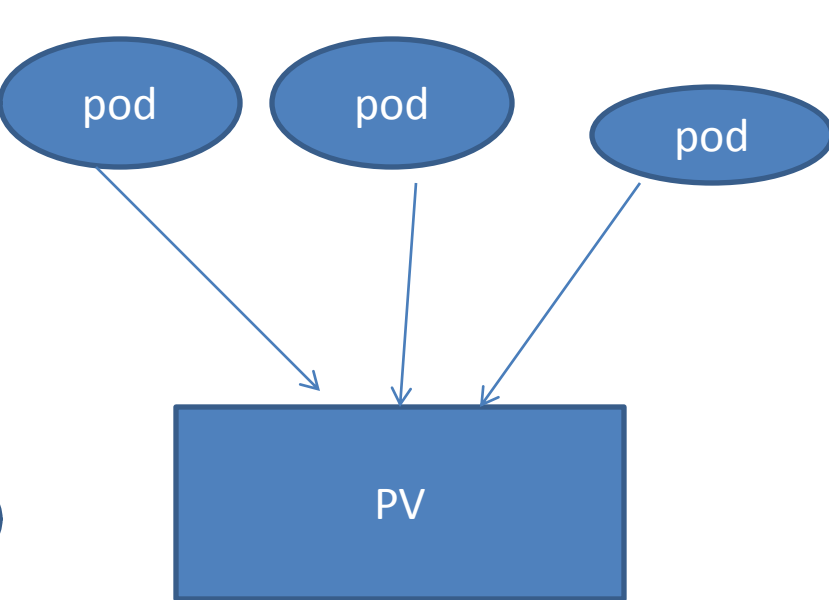
8&9 – pod to pod





1. Each pod to have its own volume
2. Reliable naming and bind to the same vol
3. Headless service –hs
4. Nodeprot np-svc

dep-svc:EP:3



Pod1.hs

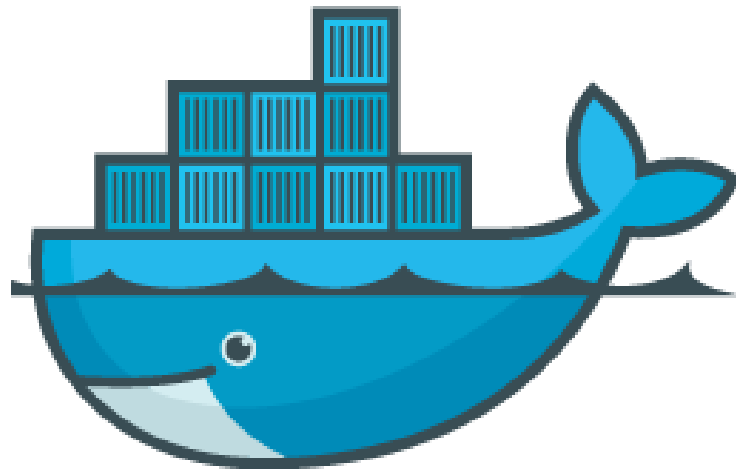




# Getting to know you

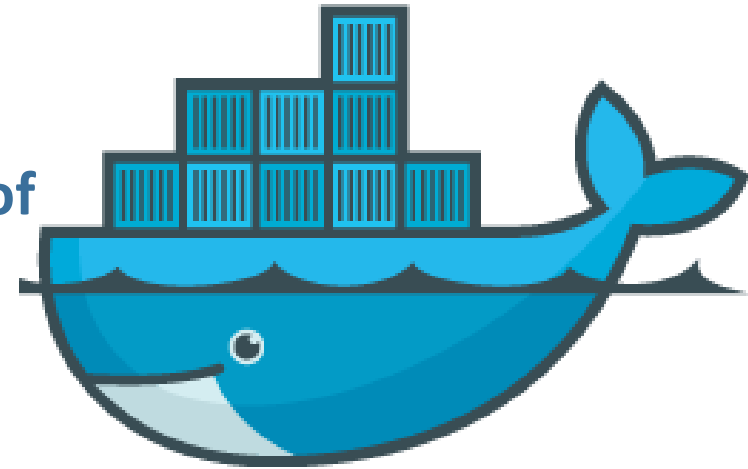
# Agenda

- 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

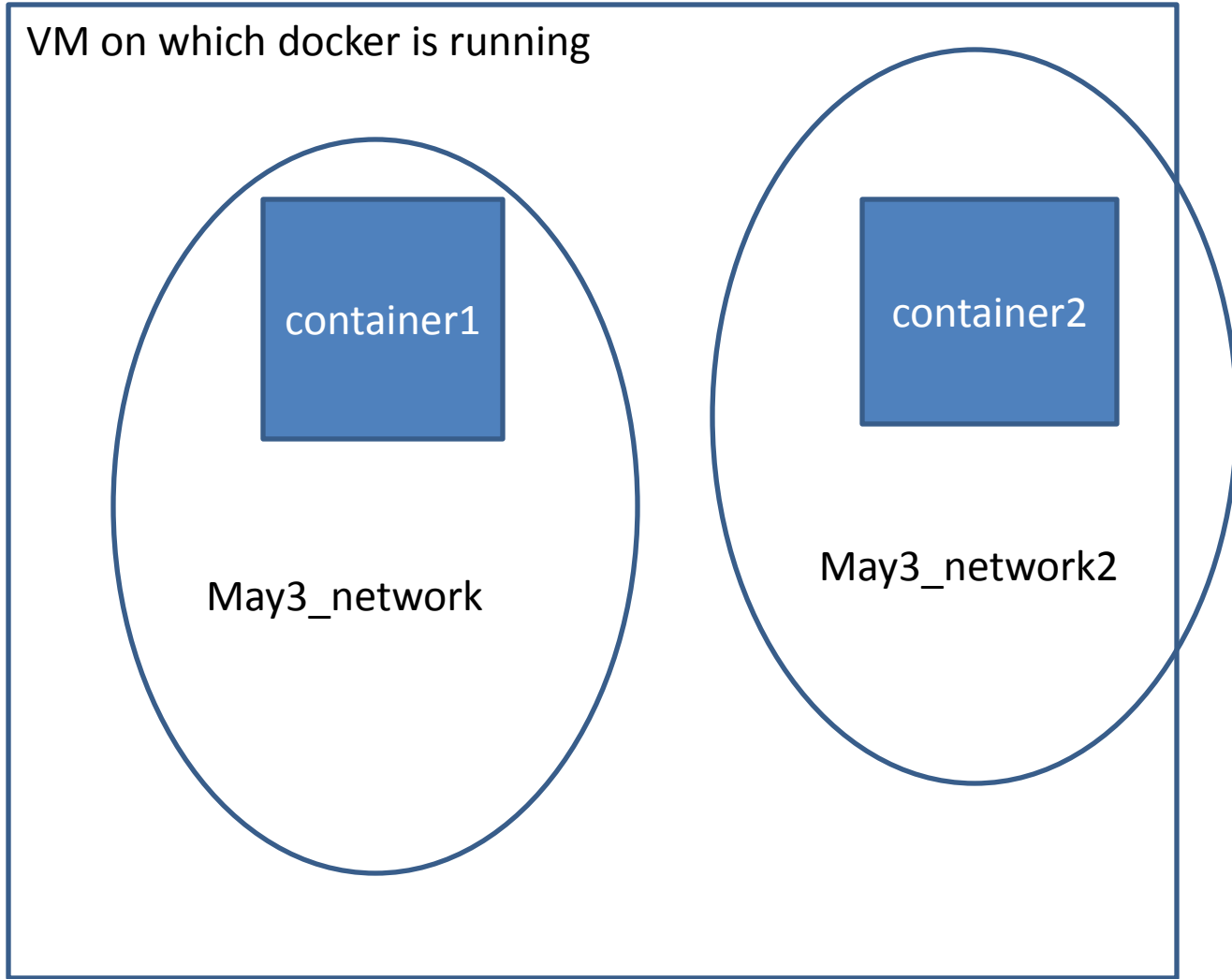
VM on which docker is running

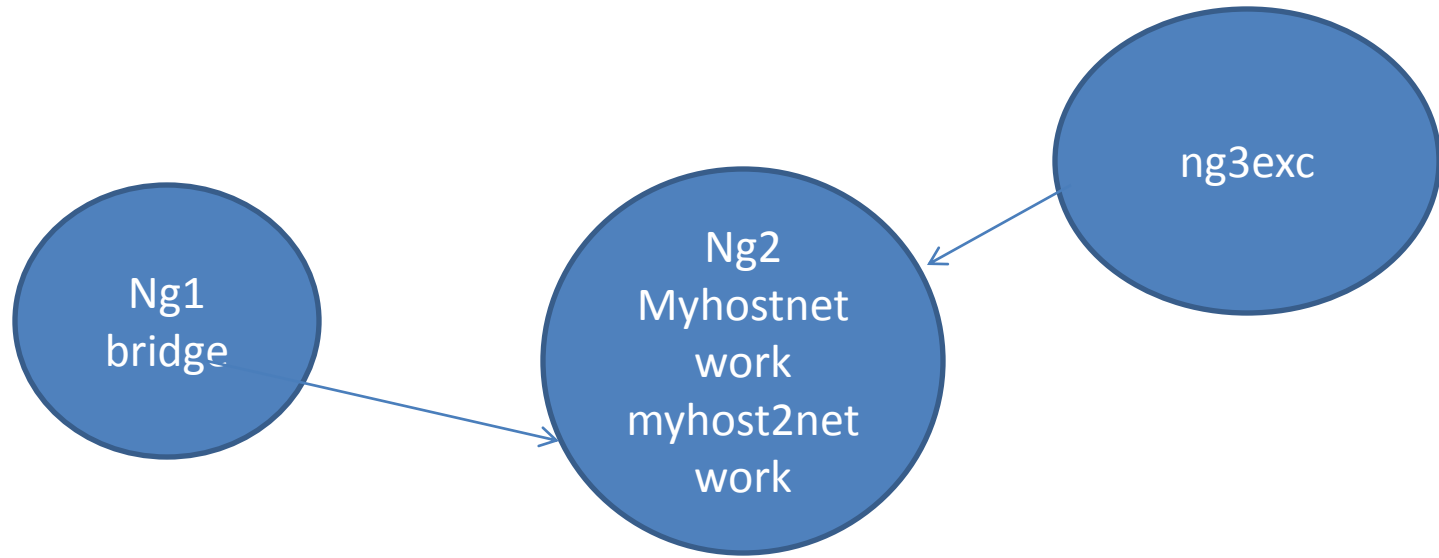
container1

May3\_network

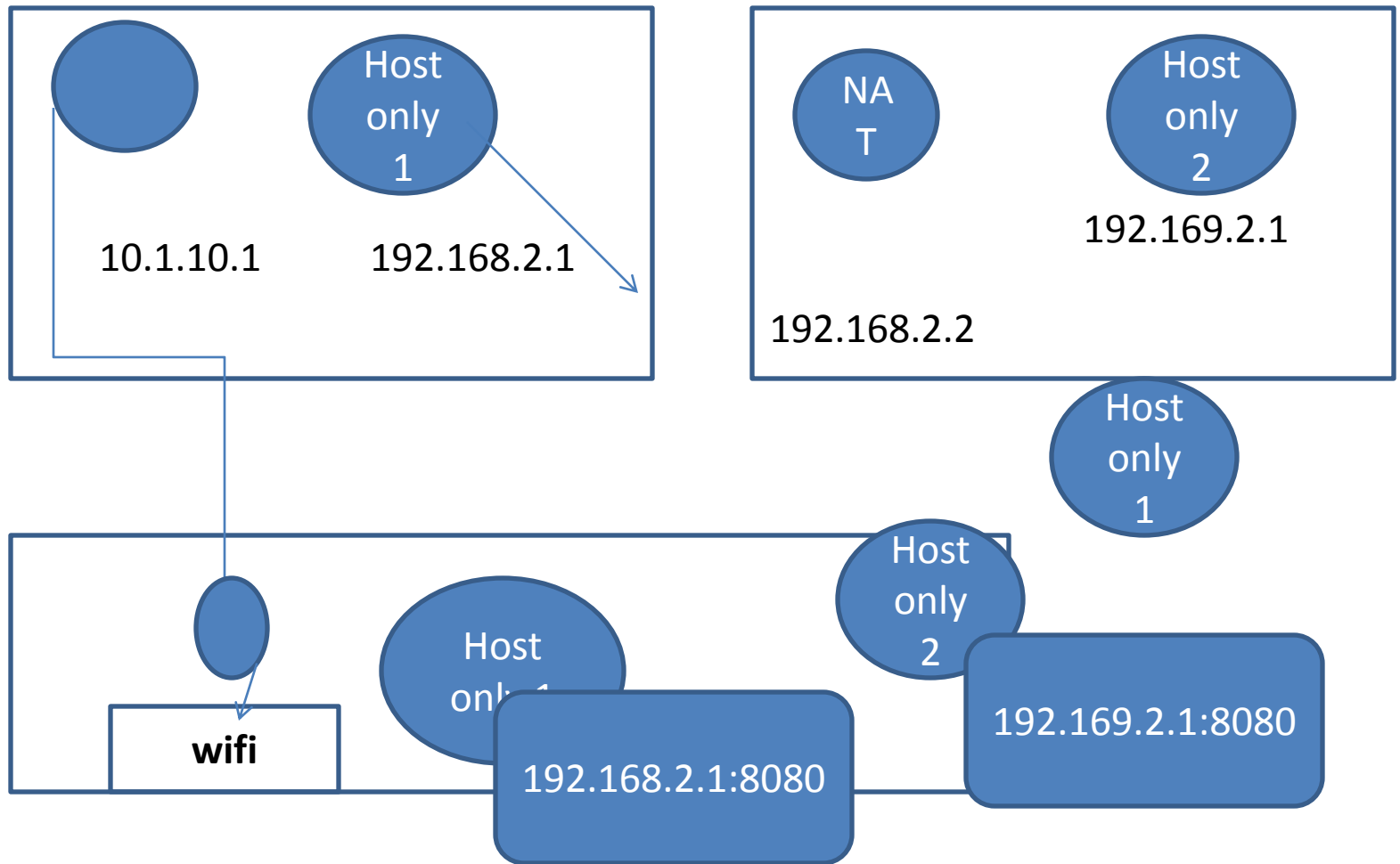
container2

May3\_network2





Link ng2 myhost2network ng1





# 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

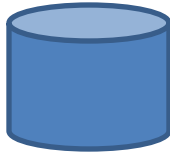
- Maps a host file or directory to a container file or directory
- Basically just two locations pointing to the same file
- Skips UFS and host files overwrite any in container
- Not a Dockerfile code. It has to be mentioned during the container run

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 tool docker-compose used for local dev/test automation with YAML files



# docker-compose.yml

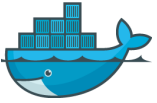
- Its own versions, 1,2,2.1,3,3.1
- YAML file can be used with docker-compose command for local docker sutomation
- docker-compose.yml is default name but can be changed



# 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



# 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



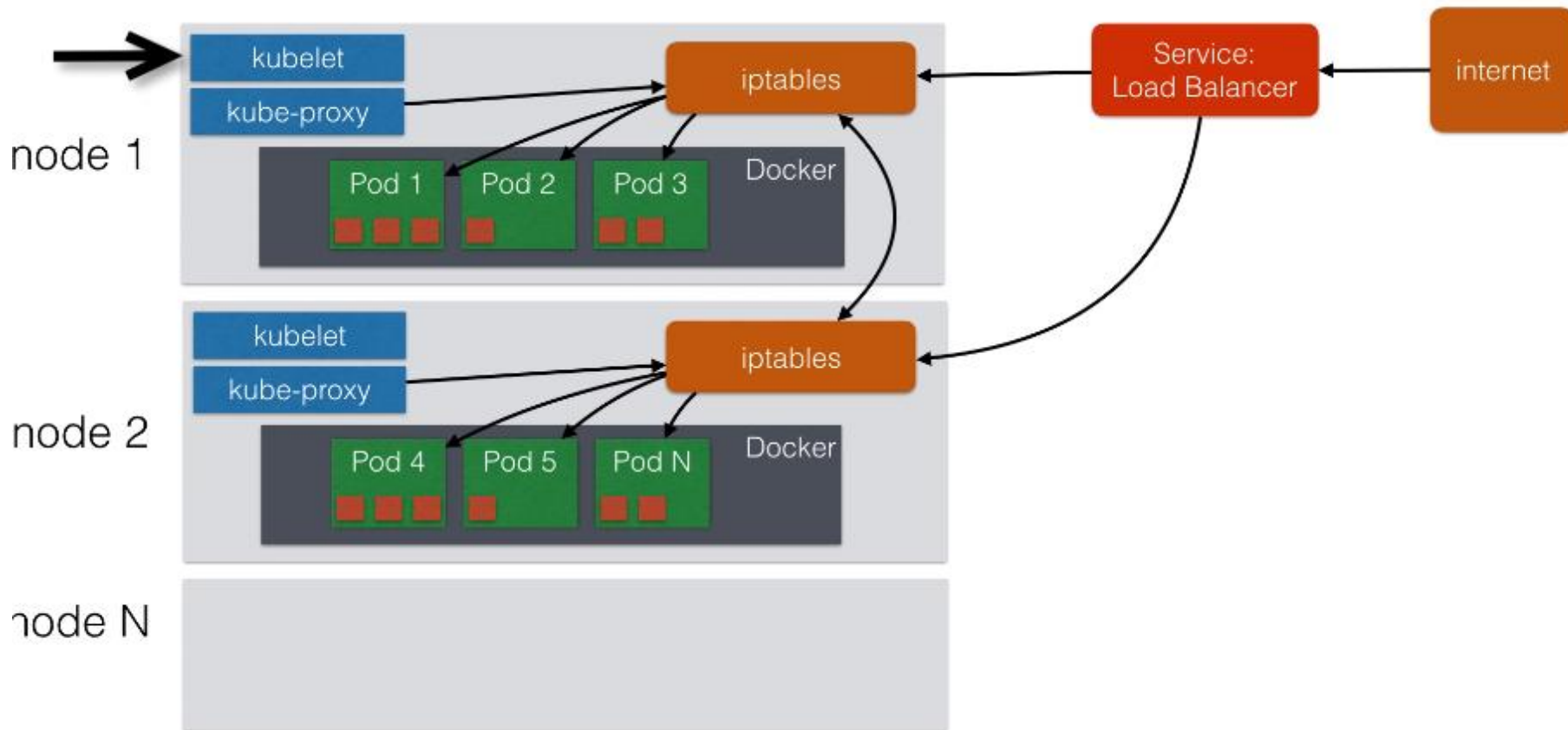


# Kubernetes

- 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



# Kubernetes – arch overview





# 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



# Kubernetes - Pod

- A pod is an application running on Kubernetes
- A pod can contain one or more tightly coupled containers
- The app can communicate easily with each other using their local port numbers



# Kubernetes - Pod

Command	Description
<code>kubectl get pod</code>	Get information about all running pods
<code>kubectl describe pod &lt;pod&gt;</code>	Describe one pod
<code>kubectl expose pod &lt;pod&gt; --port=444 --name=frontend</code>	Expose the port of a pod (creates a new service)
<code>kubectl port-forward &lt;pod&gt; 8080</code>	Port forward the exposed pod port to your local machine
<code>kubectl attach &lt;podname&gt; -i</code>	Attach to the pod
<code>kubectl exec &lt;pod&gt; -- command</code>	Execute a command on the pod
<code>kubectl label pods &lt;pod&gt; mylabel=awesome</code>	Add a new label to a pod
<code>kubectl run -i --tty busybox --image=busybox --restart=Never -- sh</code>	Run a shell in a pod - very useful for debugging



# Kubernetes - Deployments

Command	Description
<b>kubectl get deployments</b>	Get information on current deployments
<b>kubectl get rs</b>	Get information about the replica sets
<b>kubectl get pods --show-labels</b>	get pods, and also show labels attached to those pods
<b>kubectl rollout status</b> deployment/helloworld-deployment	Get deployment status
<b>kubectl set image</b> deployment/helloworld-deployment k8s-demo=k8s-demo:2	Run k8s-demo with the image label version 2
<b>kubectl edit</b> deployment/helloworld-deployment	Edit the deployment object
<b>kubectl rollout status</b> deployment/helloworld-deployment	Get the status of the rollout
<b>kubectl rollout history</b> deployment/helloworld-deployment	Get the rollout history
<b>kubectl rollout undo</b> deployment/helloworld-deployment	Rollback to previous version
<b>kubectl rollout undo</b> deployment/helloworld-deployment --to-revision=n	Rollback to any version version



# Kubernetes – Replication Controller

- If application is stateless it can be easily horizontally scaled
  - Stateless => doesn't write local files or keep local sessions
  - Databases for instance are stateful and can't be split over multiple instances
- Web applications can be made stateless
  - Session management needs to be done outside the container
- Scaling can be done using replication controller
- Replication controller will ensure a specified number of pod replicas will run at all times
- Pods created with rc will automatically be replaced if they fail
- Using rc is also a way to make sure at least one pod is running by setting the replica to 1



# Kubernetes – RS

- Deployments make use of replication set
  - RS => next gen RC
  - Supports a new selector that can do selection based on filtering according to a set of values





# Kubernetes – Deployments

- A deployment declaration in k8s allows you to do app deployments and updates
- Define the state of the application
- K8s will make sure the cluster matches the desired state
- Just using RC or RS makes the deployment cumbersome
- With a Deployment you can
  - Create a deployment (deploy an app)
  - Update a deployment (deploy a new version)
  - Do rolling updates ( zero downtime)
  - Roll back to previous version
  - Pause or resume



# Kubernetes – Services

- Pods are disposable entities. They come and go
  - RC and RS will create/terminate pods during scaling operations
- Pods should never be accessed directly but always through a service
- A service is the logical bridge between the pods and other services or end-users



# Kubernetes – Services

- Creating a service will create an endpoint for the pods
  - A cluster ip: A virtual IP address only reachable from within the cluster
  - A NodePort: a port that is the same on each node that is also reachable externally
  - A load balancer: created by the cloud provider that will route external traffic to every node on the NodePort (99246.20)

By default a service can run between ports 30000-32767 but this can be changed by Adding `--service-node-port-range` argument to the kube-apiserver

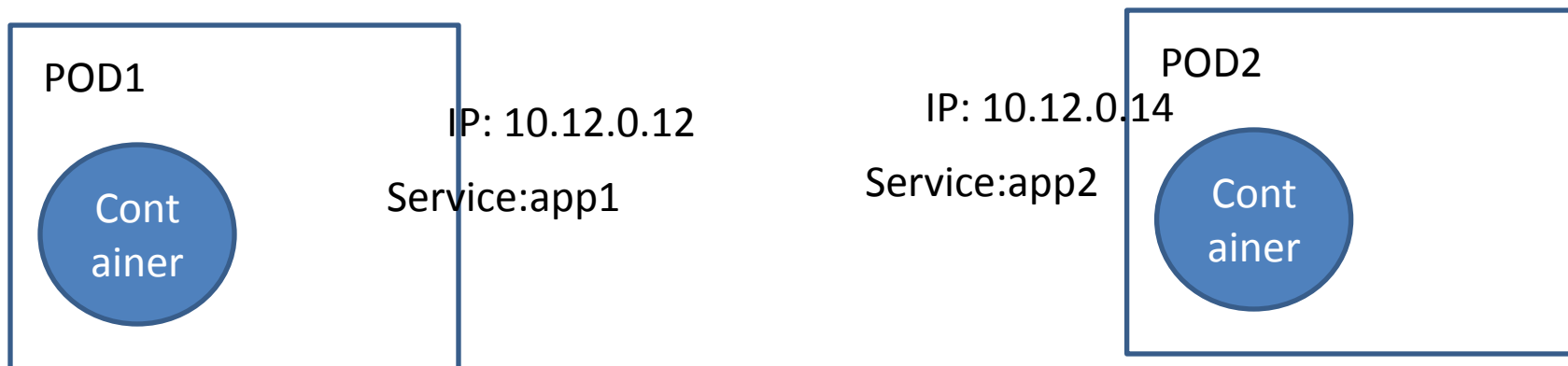


# Kubernetes – Services Discovery

- DNS is a built-in service launched automatically using the addon manager
- DNS service can be used within pods to find other services running on the same cluster
- Multiple containers within 1 pod don't need this service as they can contact directly
- A container in the same pod can connect using localhost:port
- We require a Service definition to make the DNS work



# Kubernetes – DNS working



App1-service has address 10.12.0.12

App2-service has address 10.12.0.14

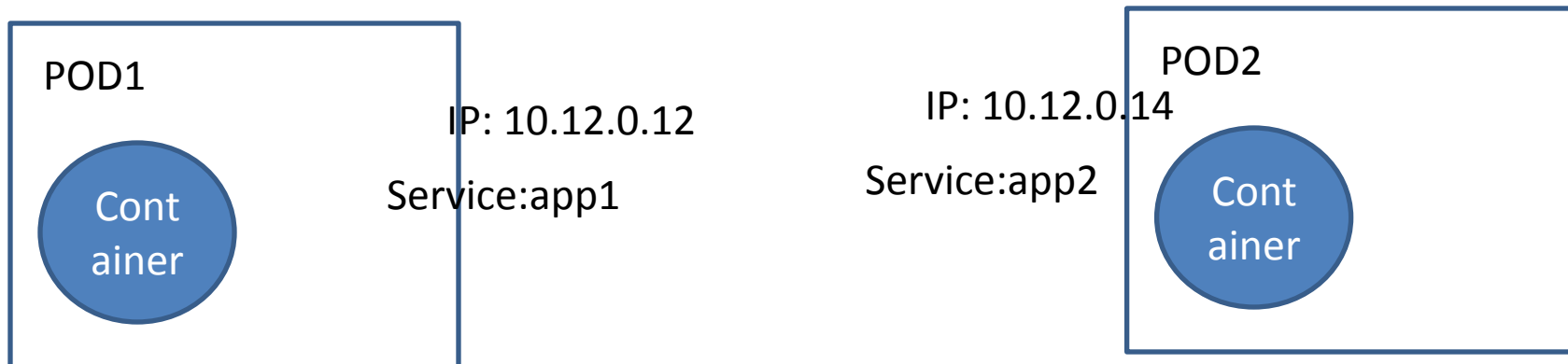
app2-service gets a fqdn as app2-service.default.svc.cluster.local

app1-service gets a fqdn as app1-service.default.svc.cluster.local

“default” is the namespace . Pods can be launched in different namespaces



# Kubernetes – DNS working



/etc/resolv.conf on pods has a way to reach services

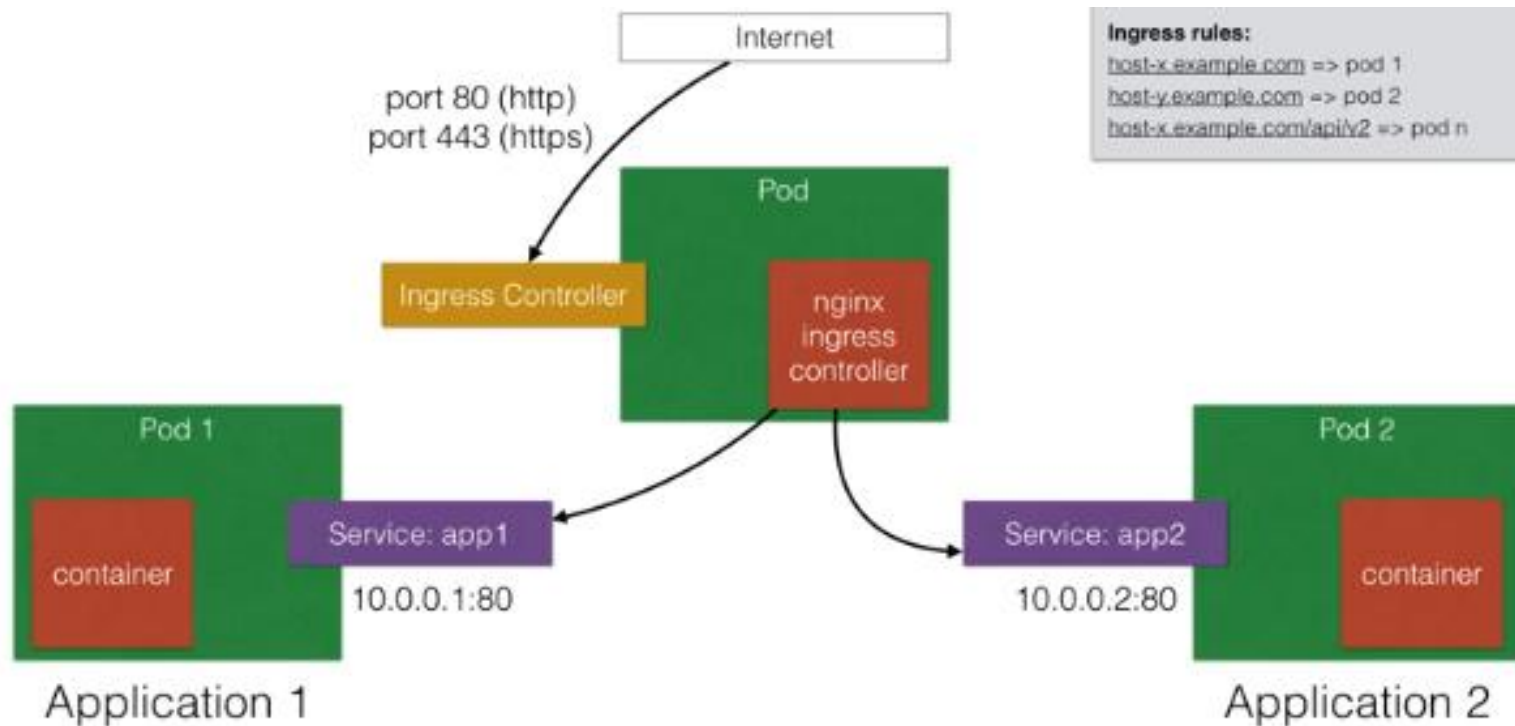


# Kubernetes – Ingress

- Ingress is a solution that allows inbound connections to the cluster
- An alternative to external loadbalancer on nodeports
  - Ingress allows you to easily expose services that need to be accessible from outside
- We can run our own ingress controller within the kubernetes cluster
- There are default ingress controllers available or we can write our own



## Kubernetes – Ingress

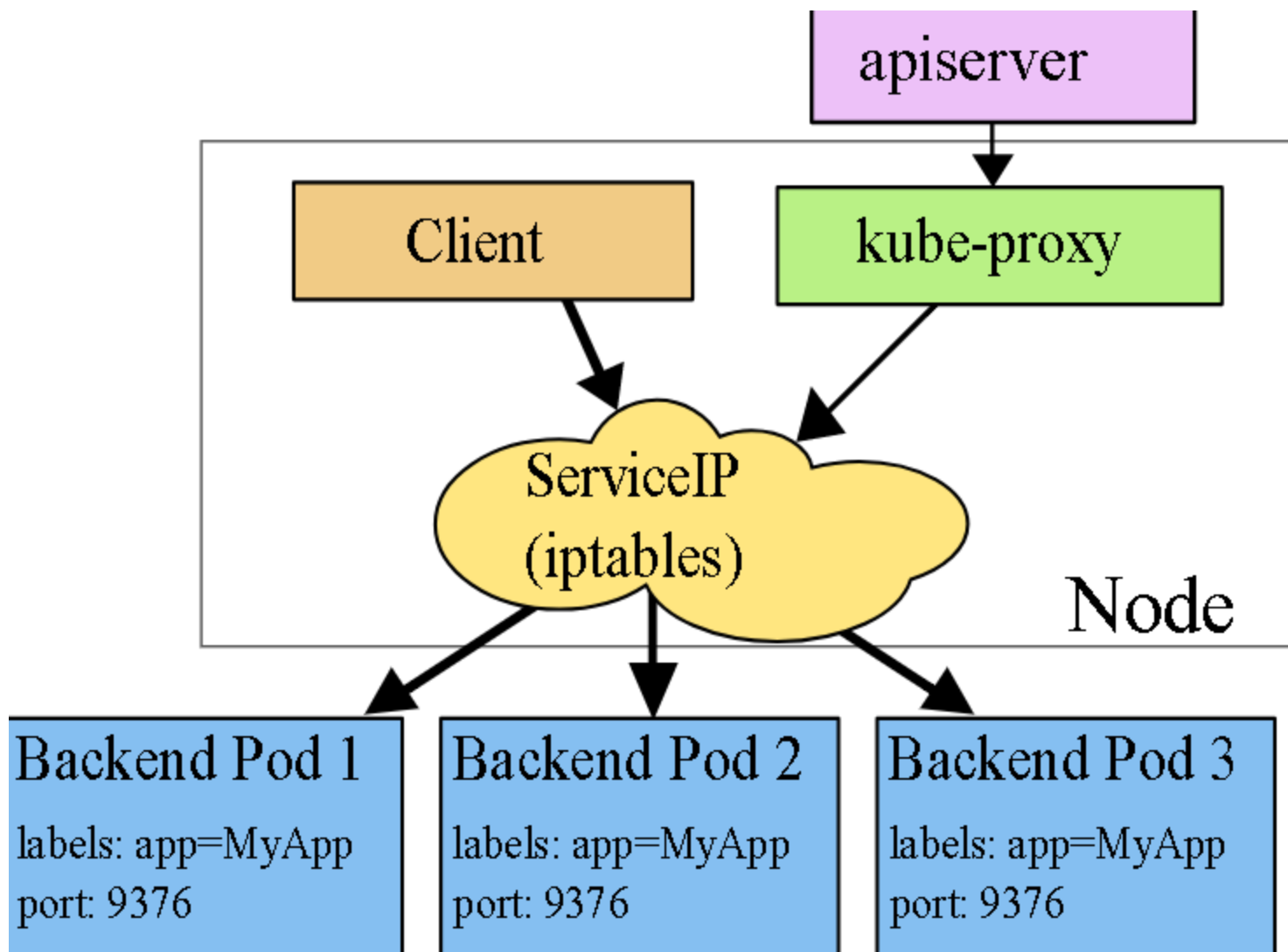






# Kubernetes – Ingress

- Ingress is a solution that allows inbound connections to the cluster
- An alternative to external loadbalancer on nodeports
  - Ingress allows you to easily expose services that need to be accessible from outside
- We can run our own ingress controller within the kubernetes cluster
- There are default ingress controllers available or we can write our own





# Docker 12 factor

**12factor.net**



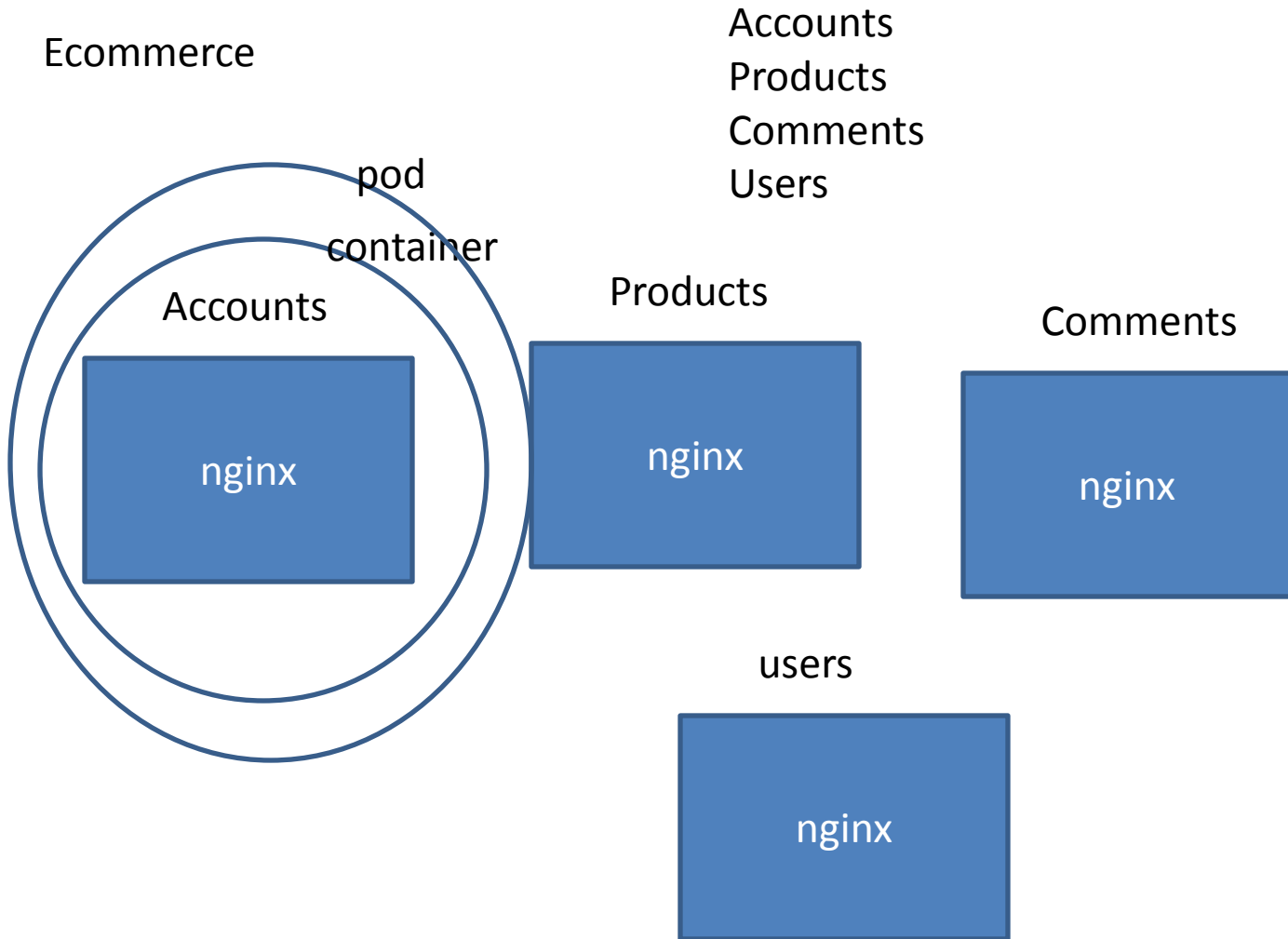
# DockerDevOps

**Implementing Docker is not DevOps**

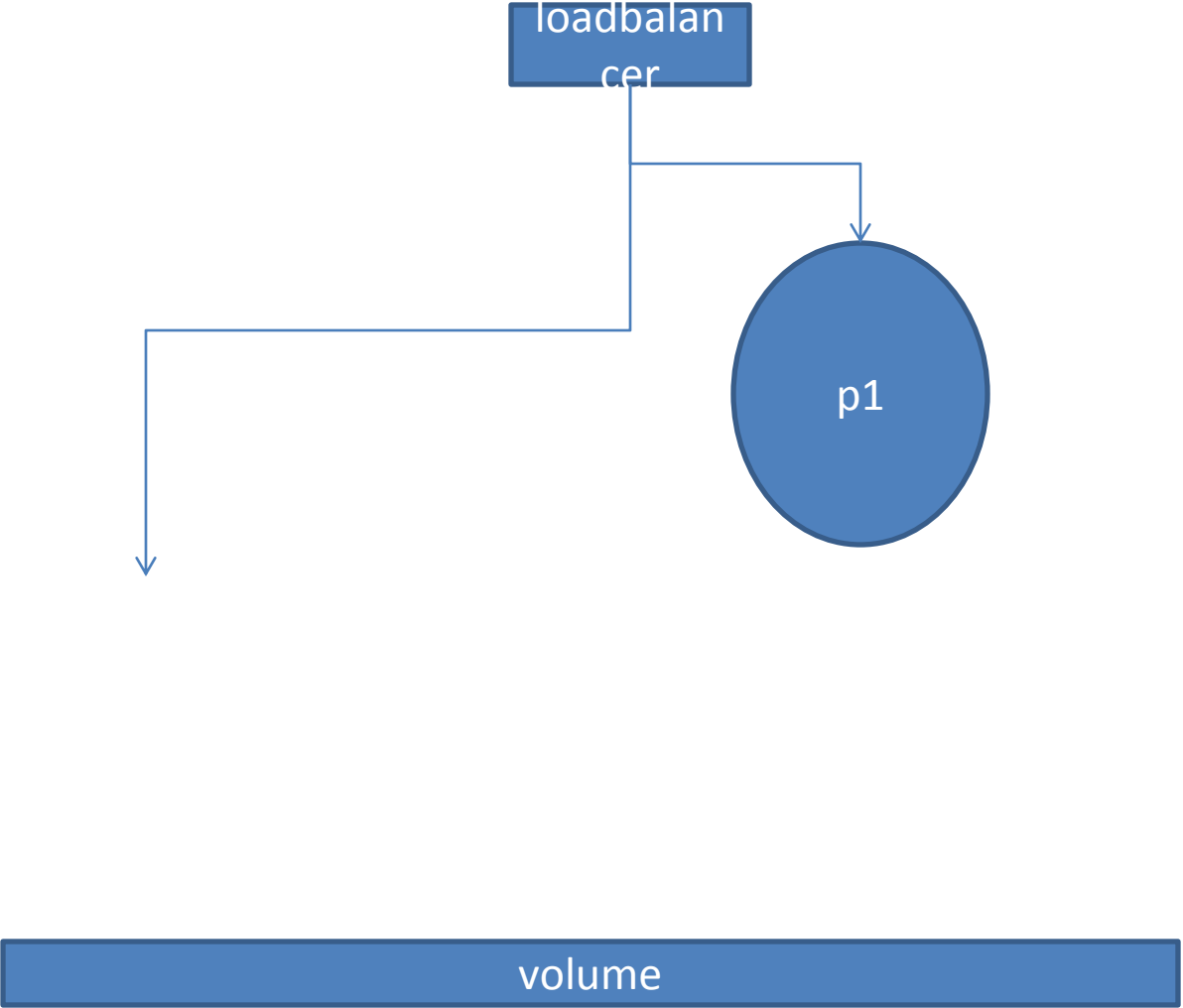
**Docker sits with the rest of automation and enables the DevOps Process**

**You will still require Cloud or VM provisioning**

**Use Docker with Agile's iterative style of development**



Accounts  
Products  
Comments  
Users





# Docker - downside

**Docker is not a fix all**

**Bare metal is always faster than VM is faster than container**

**Patterns of development have begun and takes time to evolve and mature**

**Best used for REST Api's and stateless machines**

**Persistent data storage is complicated**



## CE VS EE

Docker open source

Docker CE

DIY  
Stable and Edge  
Unsupported

Docker EE

Supported by a network of companies viz  
Microsoft, Alibaba, IBM etc

Can get your product Docker certified  
Docker datacenter  
Docker security scanning – vulnerability  
Container management - registry  
Run certified images





Kubernetes master  
Apiserver  
Control plane  
Join token  
Mesh network

Node1  
Docker  
  
Nfs mounts

Node2  
Private registry  
Docker  
Nfs mounts