gepardec

# KUBERNETES TRAINING

#WECKDENGEPARDENINDIR

gepardec

# GEPARD WHO?

gepardec

# THAT'S US

// office: vienna / linz

// size: ~ 40 people

// we do what we love..

- custom software solution
- cloud transformation
- devOps automation

Red Hat Premier Business Partner

JBoss by Red Hat

OPENSHIFT

..and we are extremly good at it

# HOW WE TEACH

// gepardec believes in **learning by doing**

// The training is **lab driven**

// Work together!

// Ask questions at any time

gepardec

# SESSION LOGISTICS

// 1 day duration

// Mostly exercises

// Regular breaks

# ASSUMED KNOWLEDGE AND REQUIREMENTS

// Familiarity with Docker

// Familiarity with Bash

// Docker Cheat Sheet: https://bit.ly/3t01s8x

// Bash Cheat Sheet: http://bit.ly/2mTQr8l

gepardec

# YOUR LAB ENVIRONMENT

// You have been given an instances for use in exercises

// Ask instructor for credentials if you don't have them already

gepardec

# COURSE LEARNING OBJECTIVES

By the end of this course, learners will be able to

// Understand the fundamental similarities and differences between docker-compose and Kubernetes

// Utilize Kubernetes orchestrators to deploy, maintain, and scale a distributed application

gepardec

# INTRODUCING DOCKER

# WHAT WE WANT

Ideal software should

// be modular and flexible (devs)
// be easy to migrate (devops)
// be easy to scale, monitor and lifecycle (ops)
// mitigate vulnerabilities (security)
// run cheap (business)

gepardec

# WITHOUT CONTAINERIZATION

DB + API + Frontend

DB Config + API Config + Frontend Config

DB Dependencies + API Dependencies + Frontend Dependencies

Base Filesystem

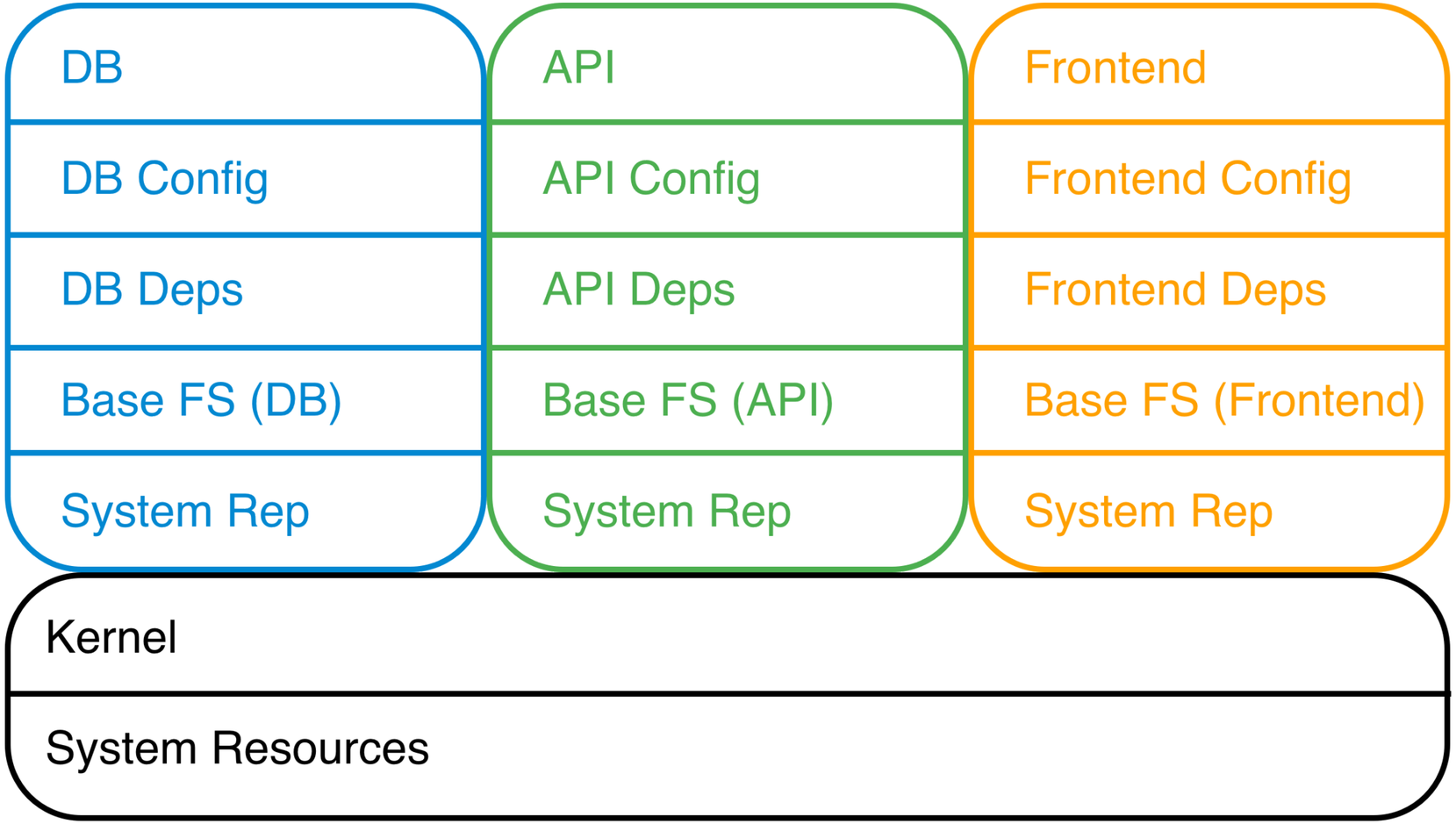
System Representation (FS, ports, process tree, users…)

Kernel

System Resources

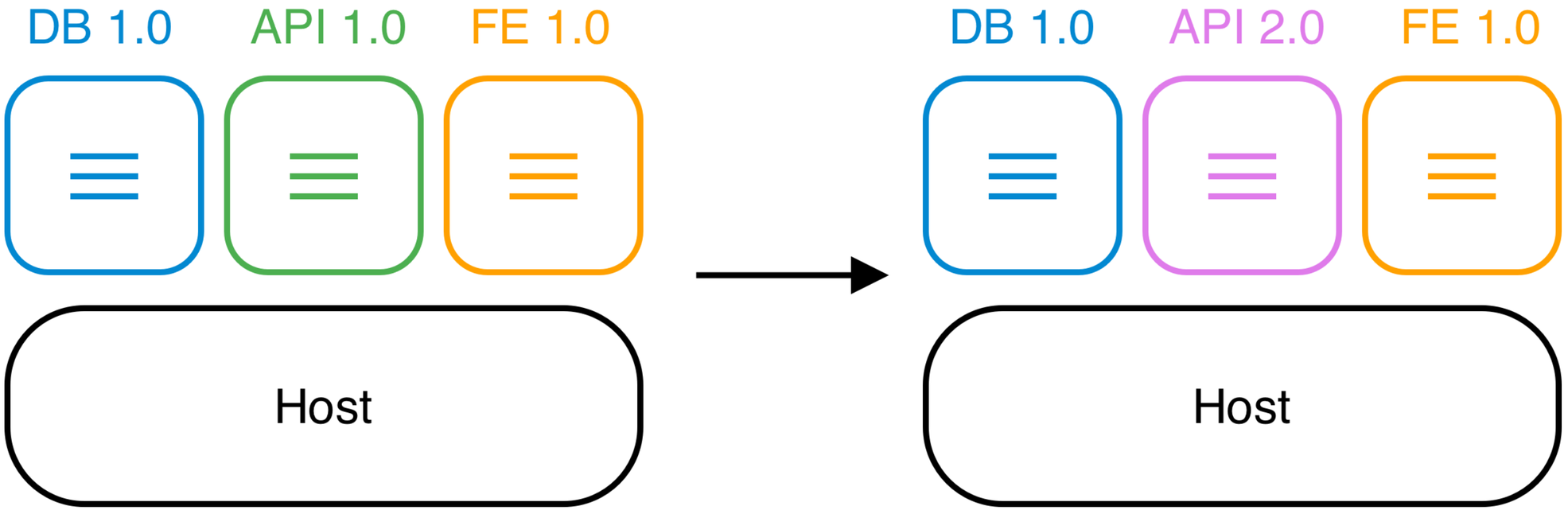
gepardec

# WITH CONTAINERIZATION



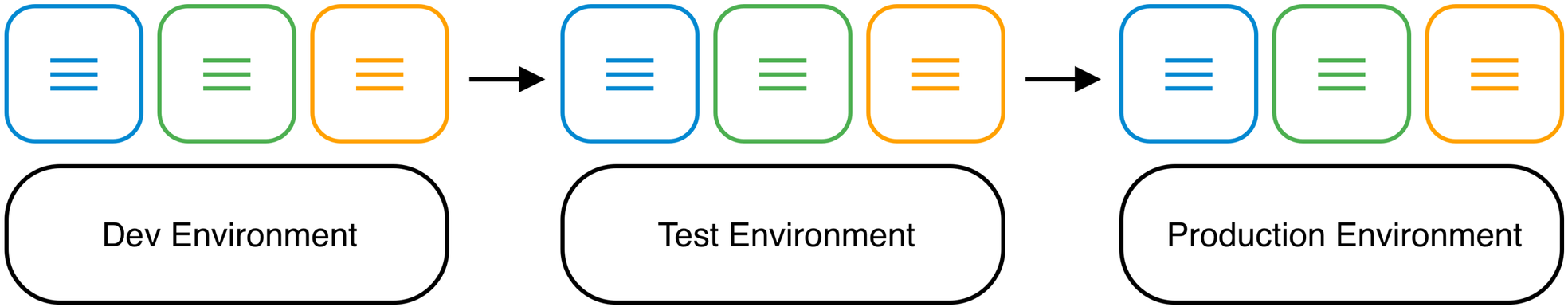
gepardec

# RAPID DEVELOPMENT




Containers can be removed and replaced with a minimum of impact on their neighbors, increasing developer choice and speed.

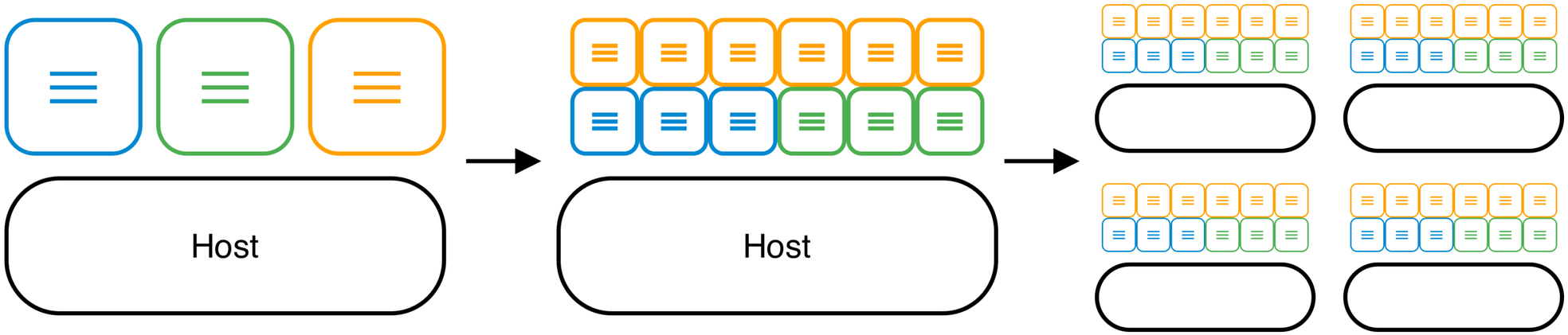
# SMOOTH MIGRATION




Containers carry their environment and dependencies with them, simplifying and minimizing requirements on the hosts that run them.

gepardec

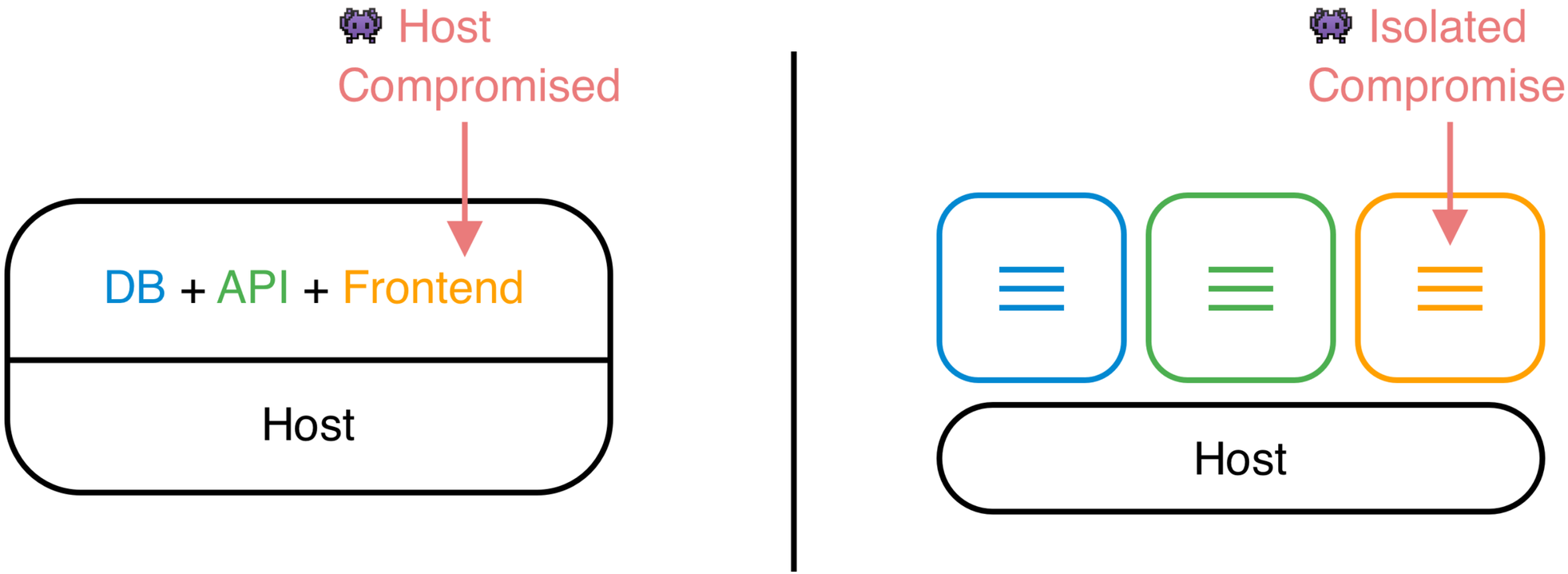
# SIMPLE SCALE & MAINTENANCE




Weak coupling between containers minimizes side effects when scaling and simplifies monitoring.

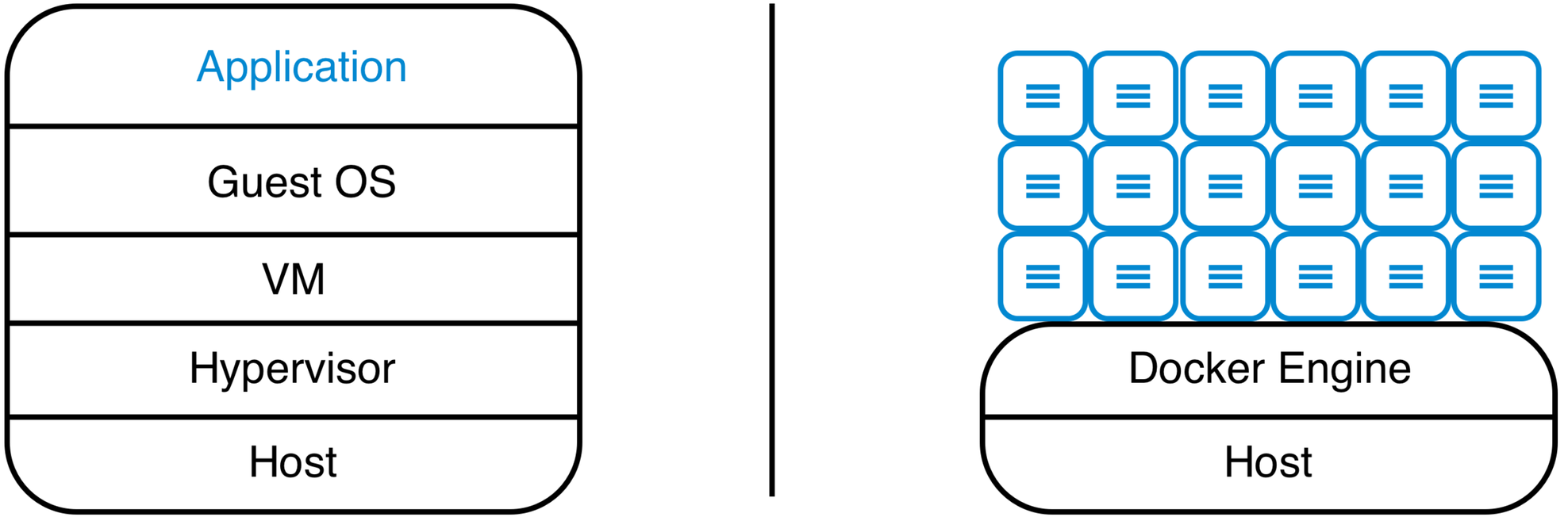
gepardec

# SECURE BY DEFAULT




Containers have private system resources, so a compromise in one does not affect the rest.

# APPLICATION DENSITY




Virtual Machines

Containers

Containers save datacenter costs by running many more application instances than virtual machines can on the same physical hosts.

gepardec

# INTRODUCTION TO DOCKER COMPOSE

gepardec

# DISCUSSION: PROCESSES VS. APPLICATIONS

Containers and images describe individual processes. What will we need to describe entire applications?

# LEARNING OBJECTIVES

By the end of this module, learners will be able to

// Design scalable Docker services
// Leverage Docker's built in service discovery mechanism
// Write a compose file describing an application

# DISTRIBUTED APPLICATION ARCHITECTURE

// Applications consisting of one or more containers across one or more nodes

// Docker Compose facilitates multi-container design **on a single node**

# DOCKER SERVICES

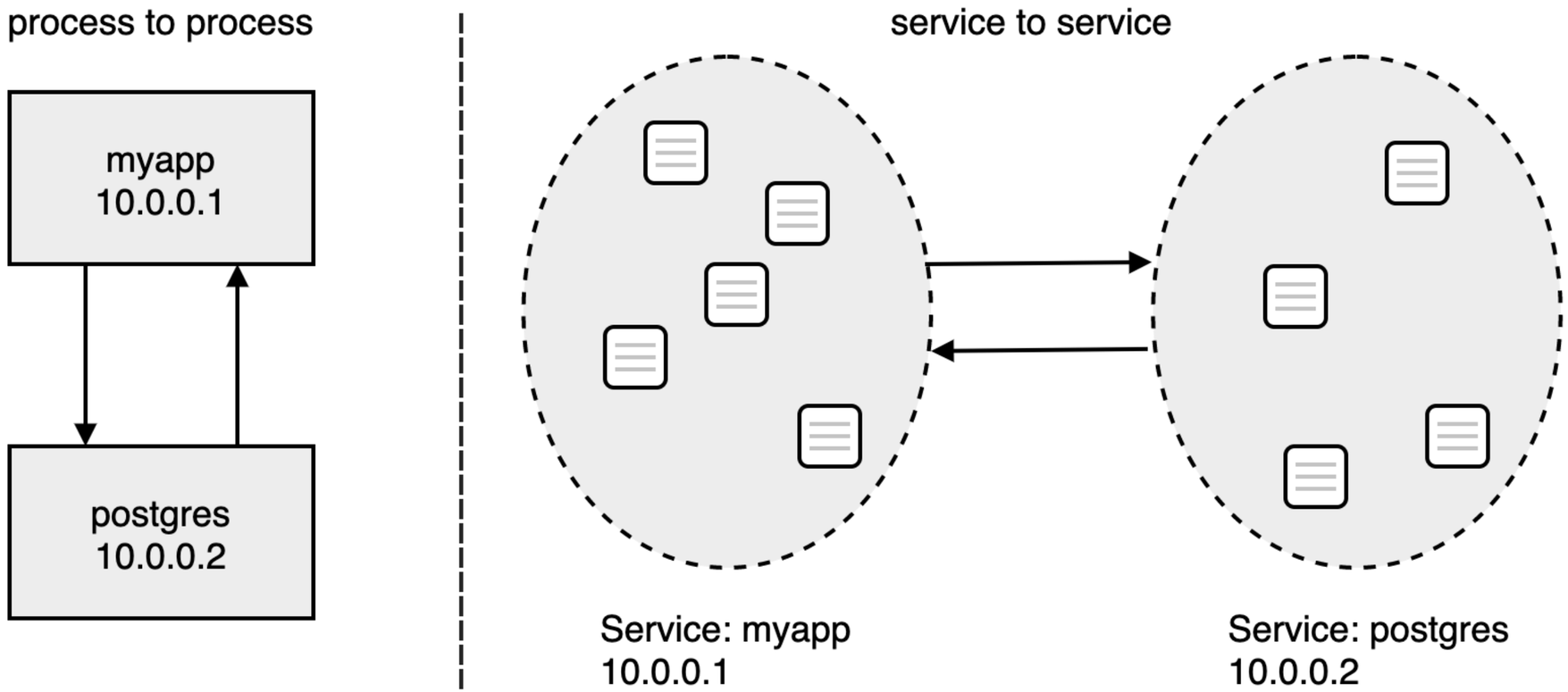
// Goal: declare and (re)configure many similar containers all at once

// Goal: scale apps by adding containers seamlessly

// A **service** defines the **desired state** of a group of identically configured containers

// Docker provides **transparent service discovery** for Services

# SERVICE DISCOVERY



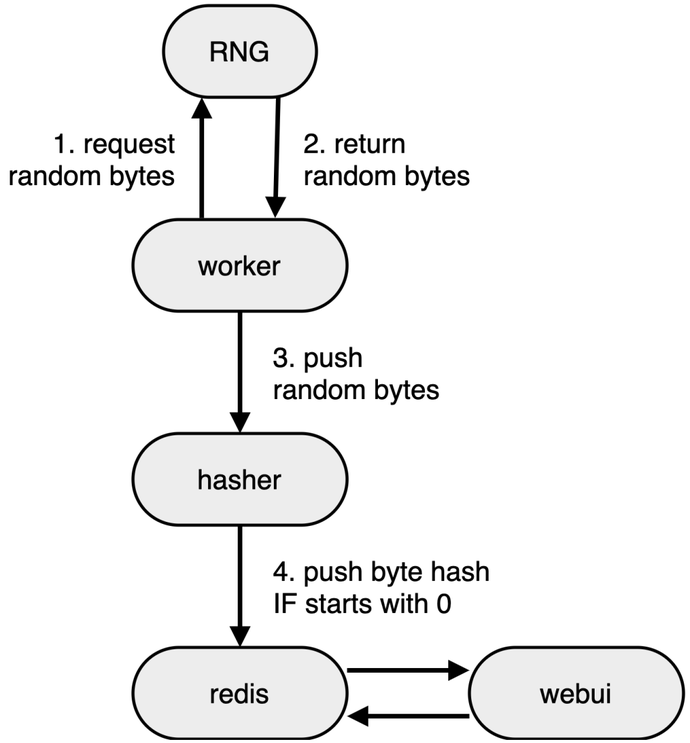

Services are assigned a Virtual IP which spreads traffic out across the underlying containers automatically.

# OUR APPLICATION: DOCKERCOINS


(DockerCoins 2016 logo courtesy of @XtlCnslt and @ndeloof. Thanks!)

// It is a DockerCoin miner! 💰🐳📦🚢

// Dockercoins consists of 5 services working together:



# INSTRUCTOR DEMO: DOCKER COMPOSE

See the demo

// Docker Compose

In the Exercises book.

# EXERCISE: COMPOSE APPS

Work through

// Starting a Compose App

// Scaling a Compose App

in the Exercises book.

gepardec

# DOCKER COMPOSE TAKEAWAYS

// Docker Compose makes single node orchestration easy

// Compose services makes scaling applications easy

// Bottleneck identification important

// Syntactically: `docker-compose.yml` + API

gepardec

# FURTHER READING

// Docker compose examples: http://dockr.ly/1FL2VQ6

// Overview of docker-compose CLI: http://dockr.ly/2wtQlZT

// `docker-compose.yaml` reference: http://dockr.ly/2iHUpeX

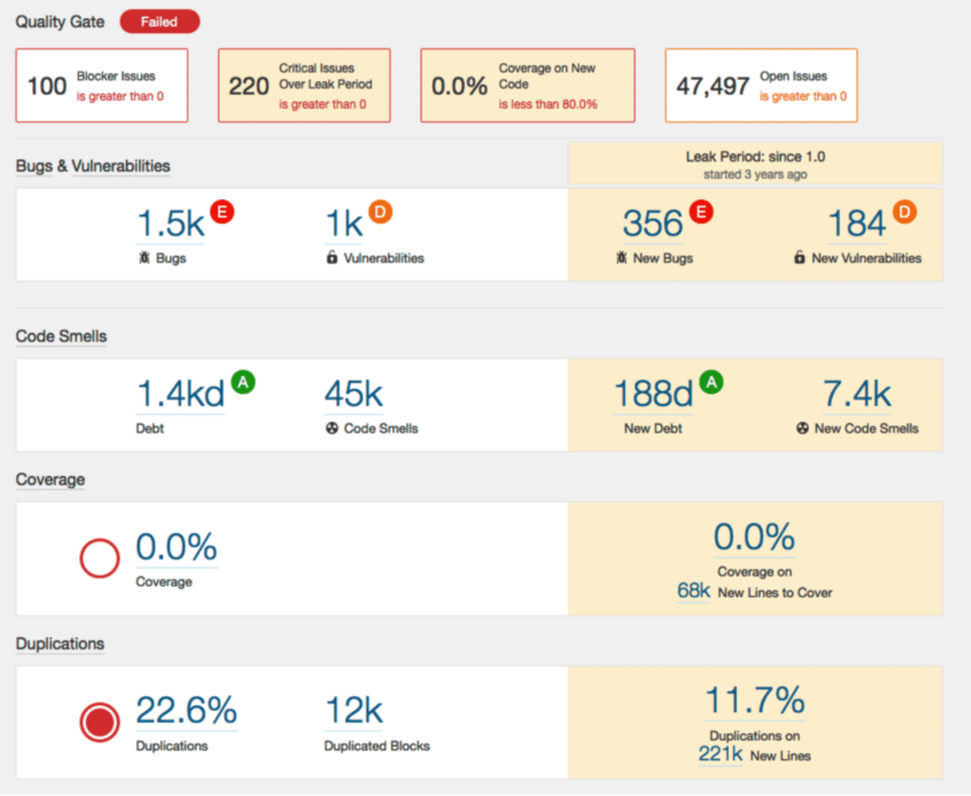
// Docker Compose and Windows: http://bit.ly/2watrqk

gepardec

# WRAP UP DOCKER-COMPOSE - SONARQUBE

gepardec

# SONARQUBE



# EXERCISE INSTRUCTIONS

// Setup a Sonarqube server that listens on port 9000

// Connect it to a persistent database
i.e. if you , docker rm –f‘ your Sonarqube container and run a new one, no data is lost

// Use postgresql and persist it‘s data on the host filesystem using volumes

// Verify e.g. by creating a user via Sonarqube UI, remove the container and run a new one – is the user still present?

// Check that Sonarqube is really using your postgresql database

// **Hint**: use docker-compose

# SOLUTION

gepardec

```
version: "2"
services:
  sonarqube:
    image: sonarqube
    ports:
      - "9000:9000"
    networks:
      - sonarnet
    environment:
      - SONARQUBE_JDBC_URL=jdbc:postgresql://db:5432/sonar
    volumes:
      - sonarqube_conf:/opt/sonarqube/conf
      - sonarqube_data:/opt/sonarqube/data
      - sonarqube_extensions:/opt/sonarqube/extensions
      - sonarqube_bundled-plugins:/opt/sonarqube/lib/bundled-plugins

  db:
    image: postgres
    networks:
      - sonarnet
    environment:
      - POSTGRES_USER=sonar
      - POSTGRES_PASSWORD=sonar
    volumes:
      - postgresql:/var/lib/postgresql
      - postgresql_data:/var/lib/postgresql/datanetworks:
    sonarnet:
      driver: bridge

volumes:
  sonarqube_conf:
  sonarqube_data:
  sonarqube_extensions:
```

gepardec

# INTRODUCTION TO KUBERNETES

gepardec

# DISCUSSION: ANY APP, ANYWHERE?

Containers are portable. What does this imply about the best ways to manage a containerized data center?

gepardec

# LEARNING OBJECTIVES

By the end of this module, learners will be able to

// Understand the components and roles of Kubernetes masters and nodes
// Identify and explain the core Kubernetes objects (Pod, ReplicaSet, Deployment, Service, Volumes)
// Provision configuration via configMaps and secrets
// Explore the Kubernetes networking model

# ORCHESTRATOR GOALS

**Top-line goal:** operate a datacenter like a **pool of compute resources** (not individual machines). This requires

// Add / remove compute resources securely and easily

// Schedule containers across the cluster transparently

// Streamline container-to-container communication (service discovery, load balancing and routing)

# PETS VERSUS LIVESTOCK

// Kubernetes reschedules exited containers automatically

// When a container becomes unhealthy, **kill it and get a new one**.

Dog photo jeffreyw; Livestock photo Paul Asman, Jill Lenoble; images CC-BY 2.0

# KUBERNETES MASTER

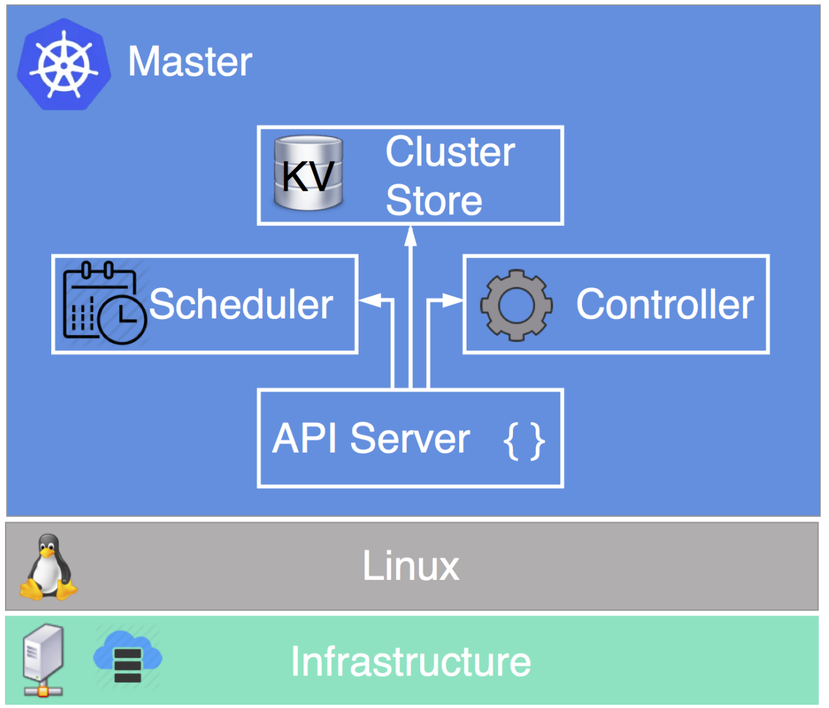
Important Components

// **API Server:** Frontend into Kubernetes control plane

// **Cluster Store:** Config and state of cluster

// **Controller Manager:** Assert desired state

// **Scheduler:** Assigns workload to nodes

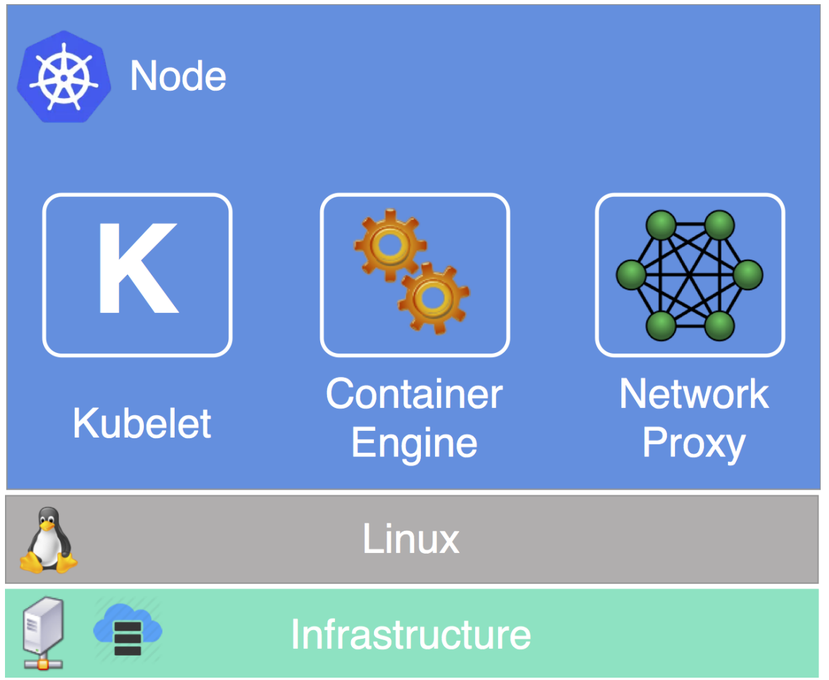


# KUBERNETES NODE

// **Kubelet:** Kubernetes Agent

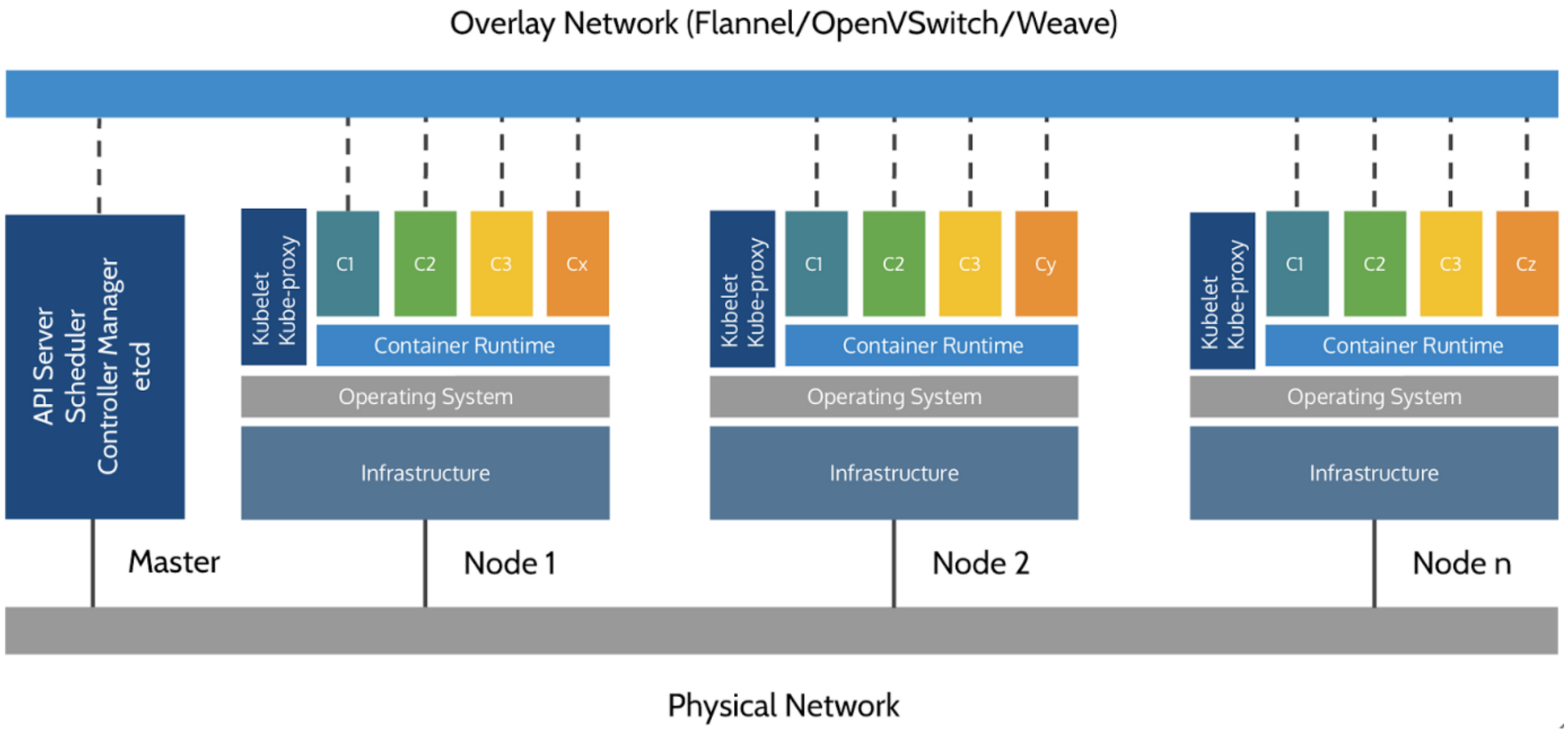
// **Container Engine:** Host Containers

// **Network Proxy:** Networking & Load Balancing



gepardec

# ARCHITECTURE



# INSTRUCTOR DEMO: KUBERNETES BASICS

See the demo

// Kubernetes Basics

In the Exercises book.

# EXERCISE: INSTALLING KUBERNETES

Work through

// Installing Kubernetes

in the Exercises book.

# FURTHER READING

// Docker & Kubernetes: https://www.docker.com/kubernetes

// Official Kubernetes Docs: https://kubernetes.io/docs

// Tutorials: http://bit.ly/2yLGn61

// Interactive Tutorials: https://bit.ly/2rdwlVZ

// Understanding Kubernetes Networking: http://bit.ly/2kdl1qQ

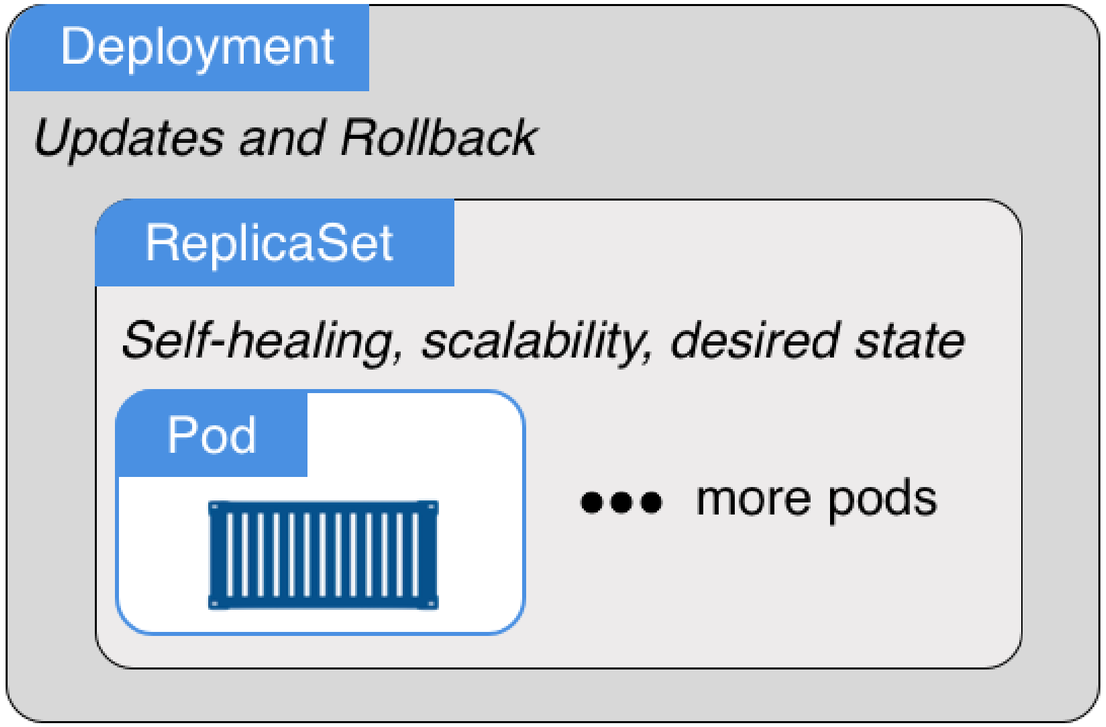
// Kubernetes the Hard Way: http://bit.ly/29Dq4wC

gepardec

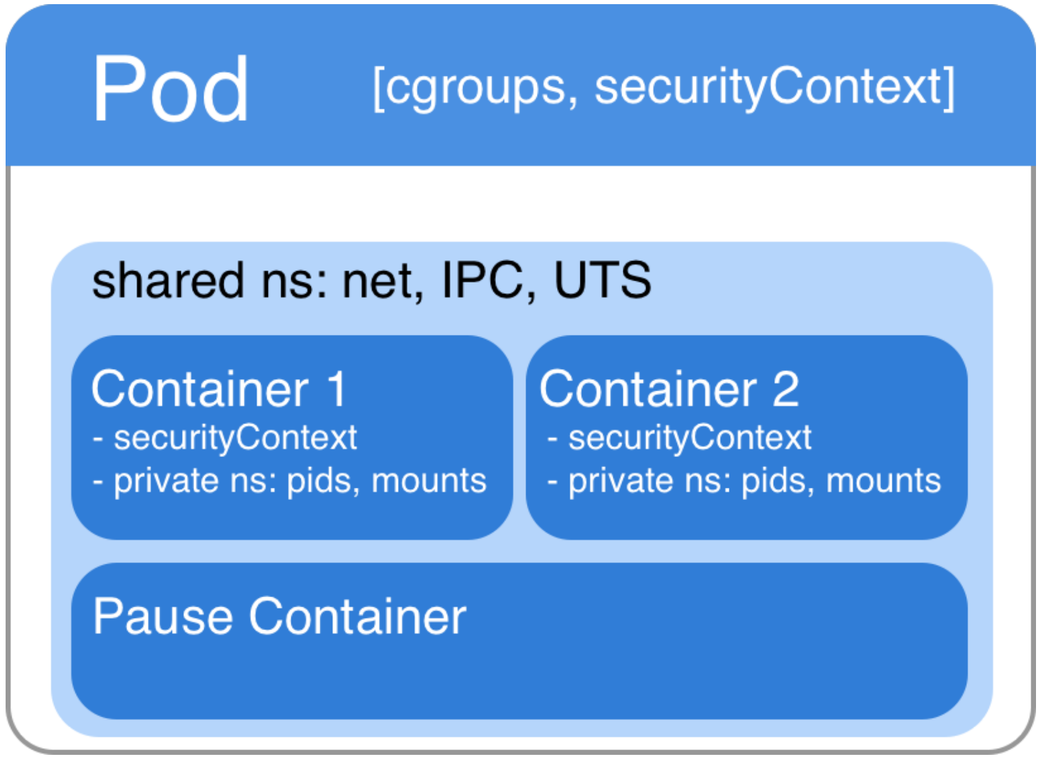
# KUBERNETES RESSOURCES

gepardec

# KUBERNETES ORCHESTRATION OBJECTS

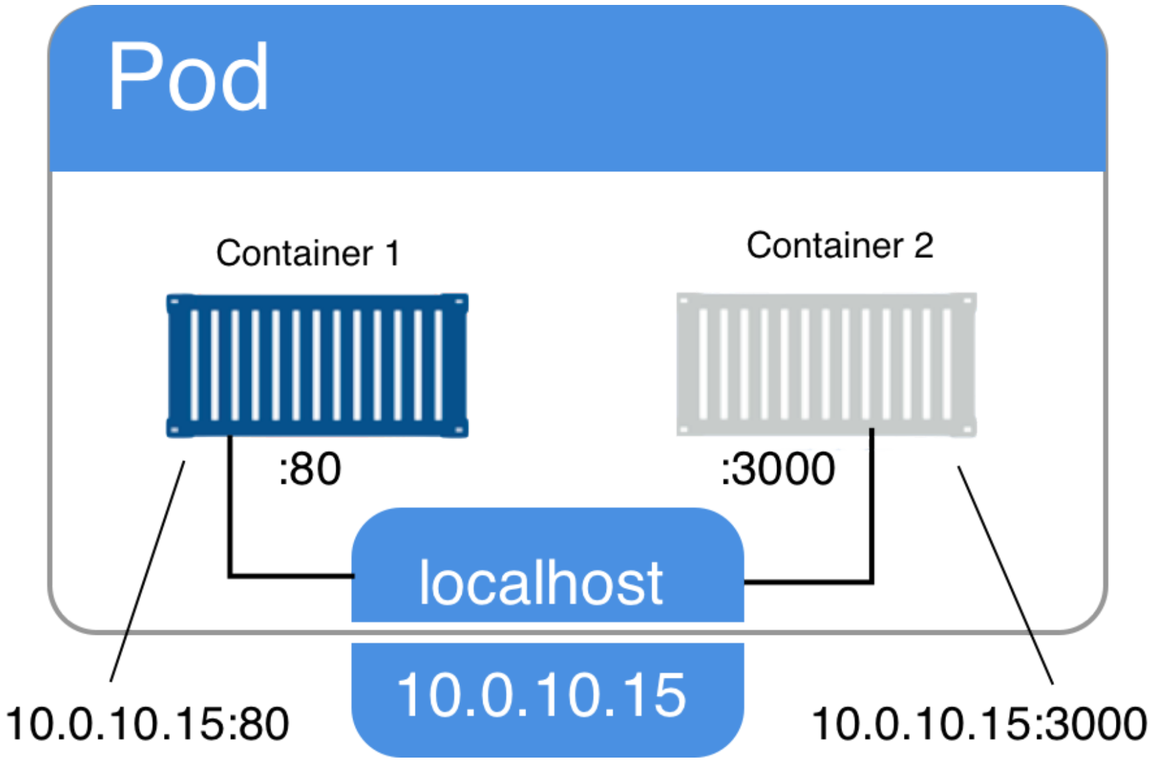


# PODS




Logical host supporting interacting processes

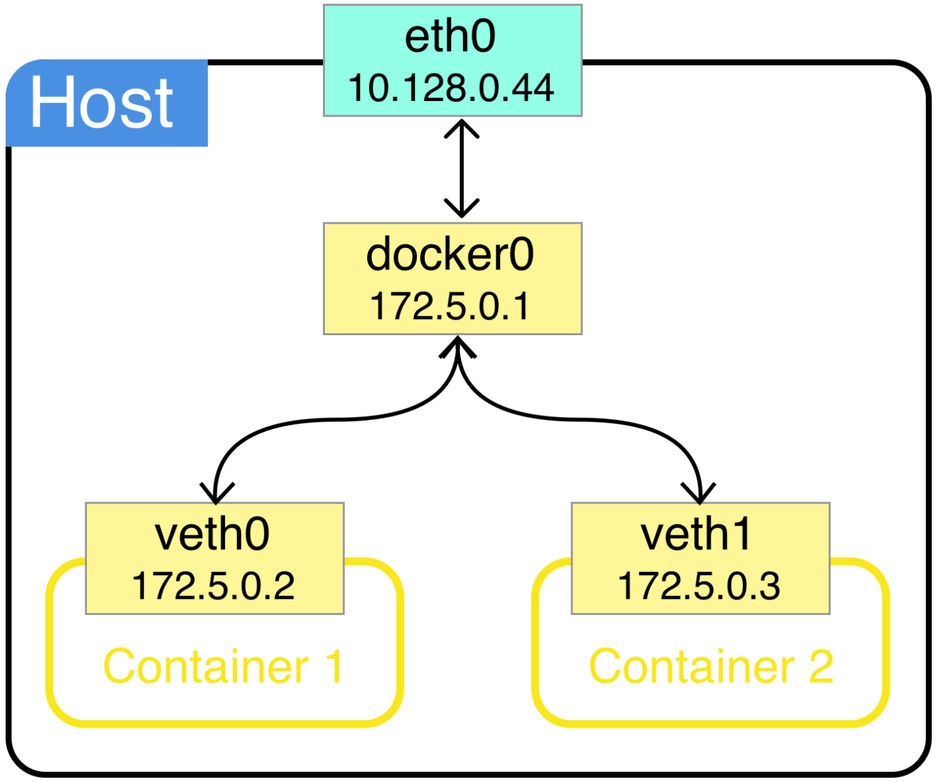
# PODS



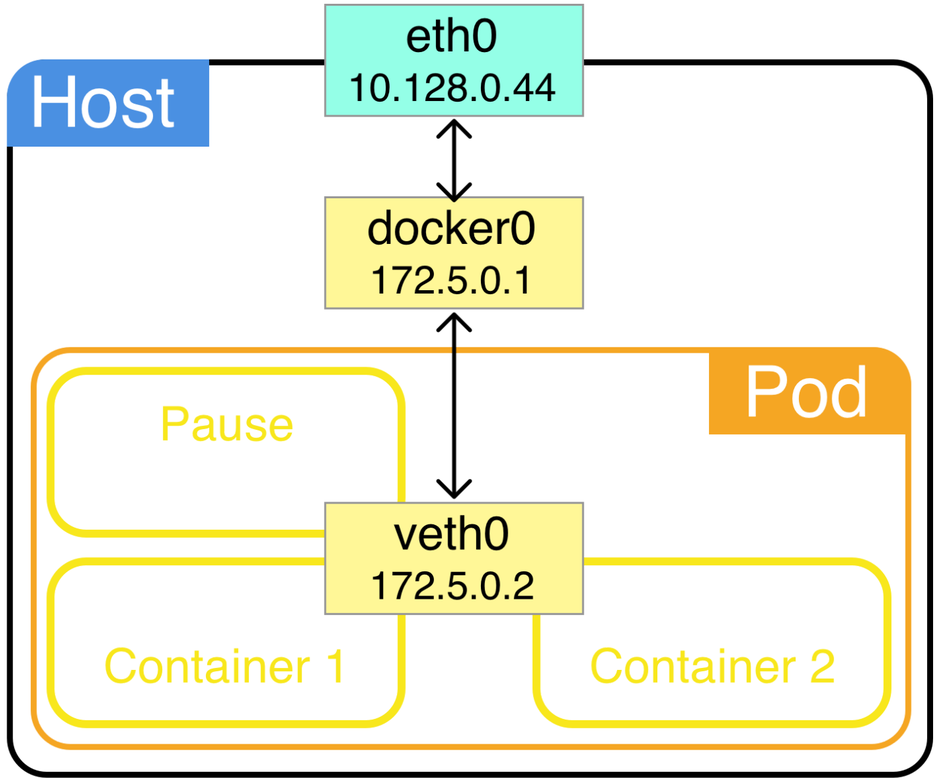

// Use `localhost` for intra-pod communication

// All containers in a pod share same IP

# POD NETWORKING

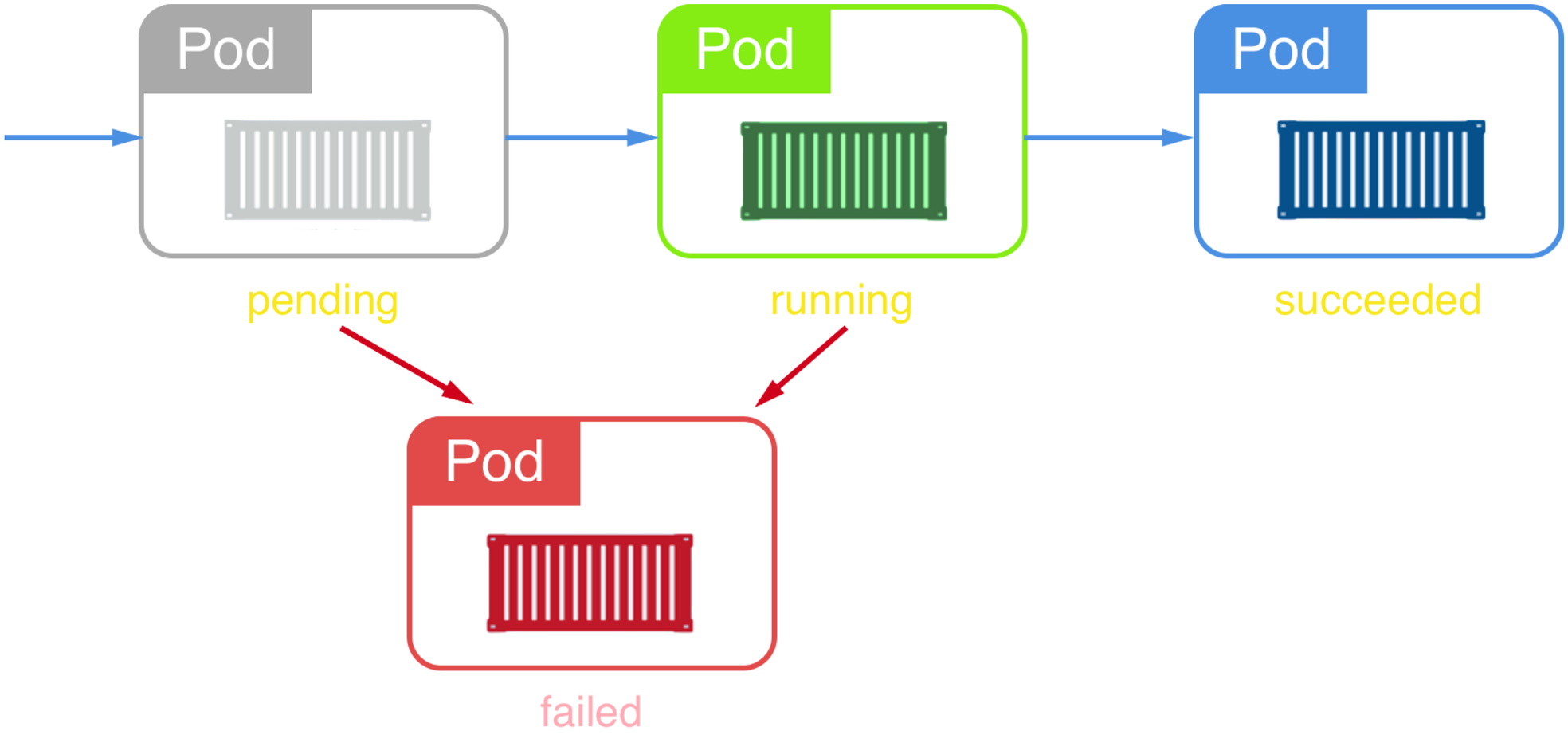



Docker Bridge Network

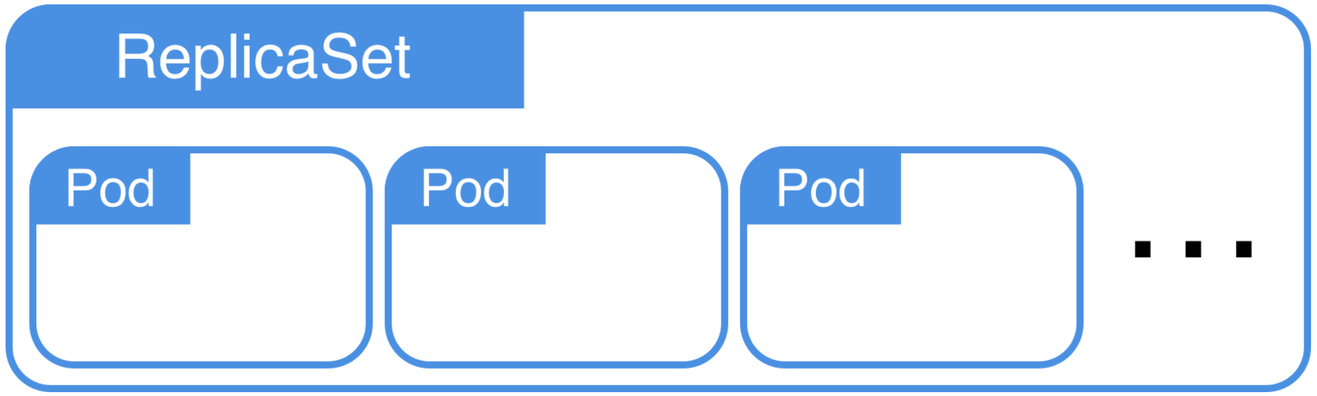



Kubernetes Network

# POD LIFECYCLE



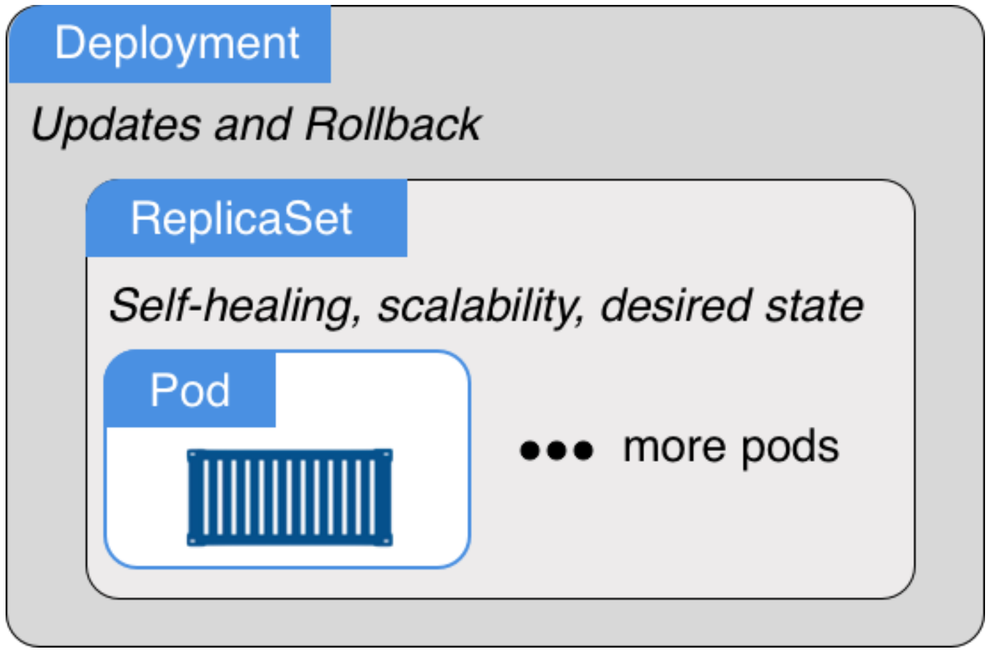
# REPLICASET




// Scaling

// Keep-alive

# DEPLOYMENT




// Build on top of **ReplicaSets**

// Add configurable Updates and Rollback

// Older Versions of ReplicaSets stick around for easy Rollback

# EXERCISE: KUBERNETES ORCHESTRATION

Work through

// Kubernetes Orchestration

in the Exercises book.

# FURTHER READING

// Docker & Kubernetes: https://www.docker.com/kubernetes

// Official Kubernetes Docs: https://kubernetes.io/docs

// Tutorials: http://bit.ly/2yLGn61

// Interactive Tutorials: https://bit.ly/2rdwlVZ

// Understanding Kubernetes Networking: http://bit.ly/2kdl1qQ

// Kubernetes the Hard Way: http://bit.ly/29Dq4wC

gepardec

# CONFIGURATION & SECRETS

# CONFIGURATION

Applications typically need environment-specific config:

// Environment variables
// Configuration files
// Non-sensitive info (ports, usernames, endpoints)
// Sensitive info (passwords, access tokens, keys)

Config should be **decoupled** from pod definition and **portable** across the cluster.

gepardec

# CONFIGMAPS

// Collections of key/value pairs, or text files

// Provisioned to containers via env vars or volume mounts

// Appropriate for low/no security config

gepardec

# SECRETS

// Defined and provisioned similarly to configMaps (env vars or volume mounts)

// Intended for secure info:

- Provisioned in a tmpfs, never written to disk

// **Warning**: secrets are recoverable with `kubectl get secrets` from masters, and potentially with `docker container inspect` from host workers

# FURTHER READING

// Docker & Kubernetes: https://www.docker.com/kubernetes

// Official Kubernetes Docs: https://kubernetes.io/docs

// Tutorials: http://bit.ly/2yLGn61

// Interactive Tutorials: https://bit.ly/2rdwlVZ

// Understanding Kubernetes Networking: http://bit.ly/2kdl1qQ

// Kubernetes the Hard Way: http://bit.ly/29Dq4wC

gepardec

# VOLUMES

gepardec

# STORAGE VOLUMES

// Volumes: same lifecycle as pod (compare to persistent Docker volumes)

// PersistentVolumes: 'immortal' volume (similar to Docker)

// Storage backends exposed by **container storage interface** drivers

# STORAGE BACKEND

Responsible for:

// Managing writing on Disks / HDD / Directories
// Manage dynamic provisioning / sizing of volumes
// Manage replication, failover and backup

Examples: Rook-Ceph, Longhorn, EBS, AzureDisk

gepardec

# PERSISTENT VOLUME CLAIM

// "Request" for a persistent volume

// Used by Pods to get access to a persistent volume

// ReadWriteOnce: Can only be mounted once

// ReadWriteMany: Can be mounted from multiple pods

gepardec

# PERSISTENT VOLUME CLAIM: EXAMPLE

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce #or ReadWriteMany
  resources:
    requests:
      storage: 8Gi
  storageClassName: longhorn
```

gepardec

# STATEFULSET

// Like Deployments

// Can hold a template for a PersistentVolumeClaim

// Each replica can have their own persistent volume

// Used for: Databases, Stateful Apps

# EXERCISE: KUBERNETES VOLUMES

Work through

// Kubernetes Volumes

in the Exercises book.

# FURTHER READING

// Docker & Kubernetes: https://www.docker.com/kubernetes

// Official Kubernetes Docs: https://kubernetes.io/docs

// Tutorials: http://bit.ly/2yLGn61

// Interactive Tutorials: https://bit.ly/2rdwlVZ

// Understanding Kubernetes Networking: http://bit.ly/2kdl1qQ

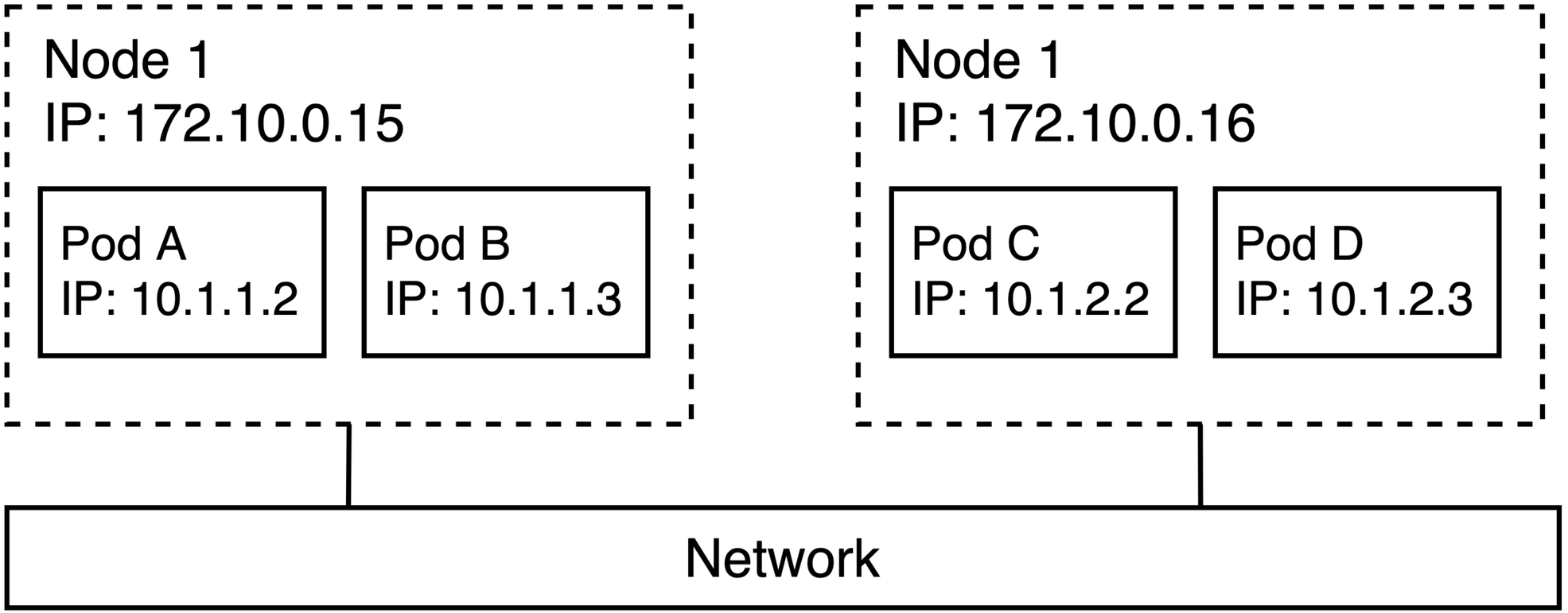
// Kubernetes the Hard Way: http://bit.ly/29Dq4wC

gepardec

# KUBERNETES NETWORKING

# KUBERNETES NETWORK MODEL

gepardec




Requirements

// Pod <—> Pod without NAT

// Node <—> Pod without NAT

// Pod's peers find it at the same IP it finds itself

// Creates a **flat network**, like VMs

# SERVICE

Problem:

// Pods are mortal

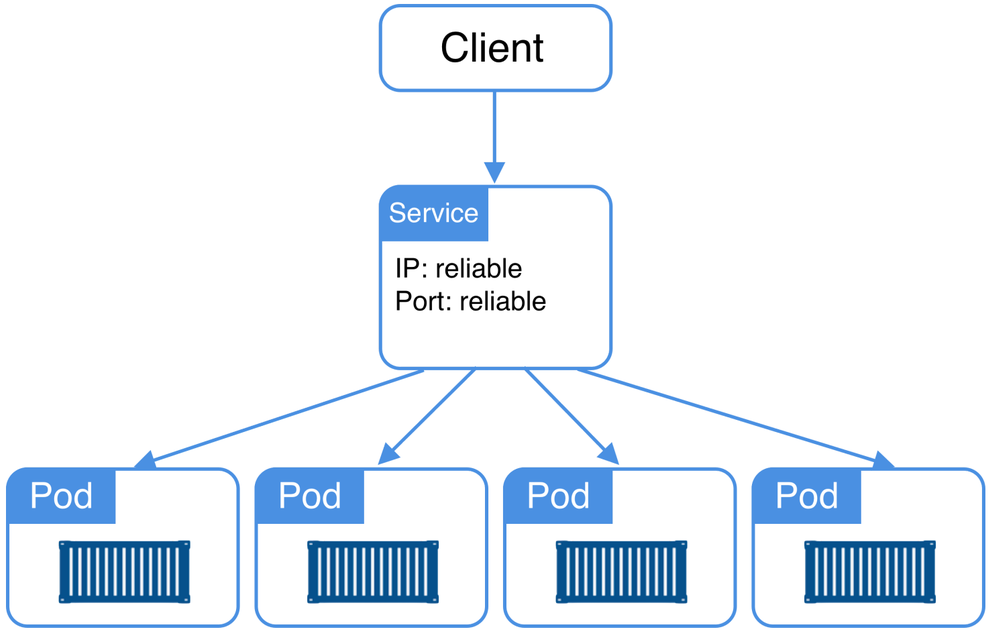
// Pods are never resurrected

// Pod IP Addr cannot be relied on

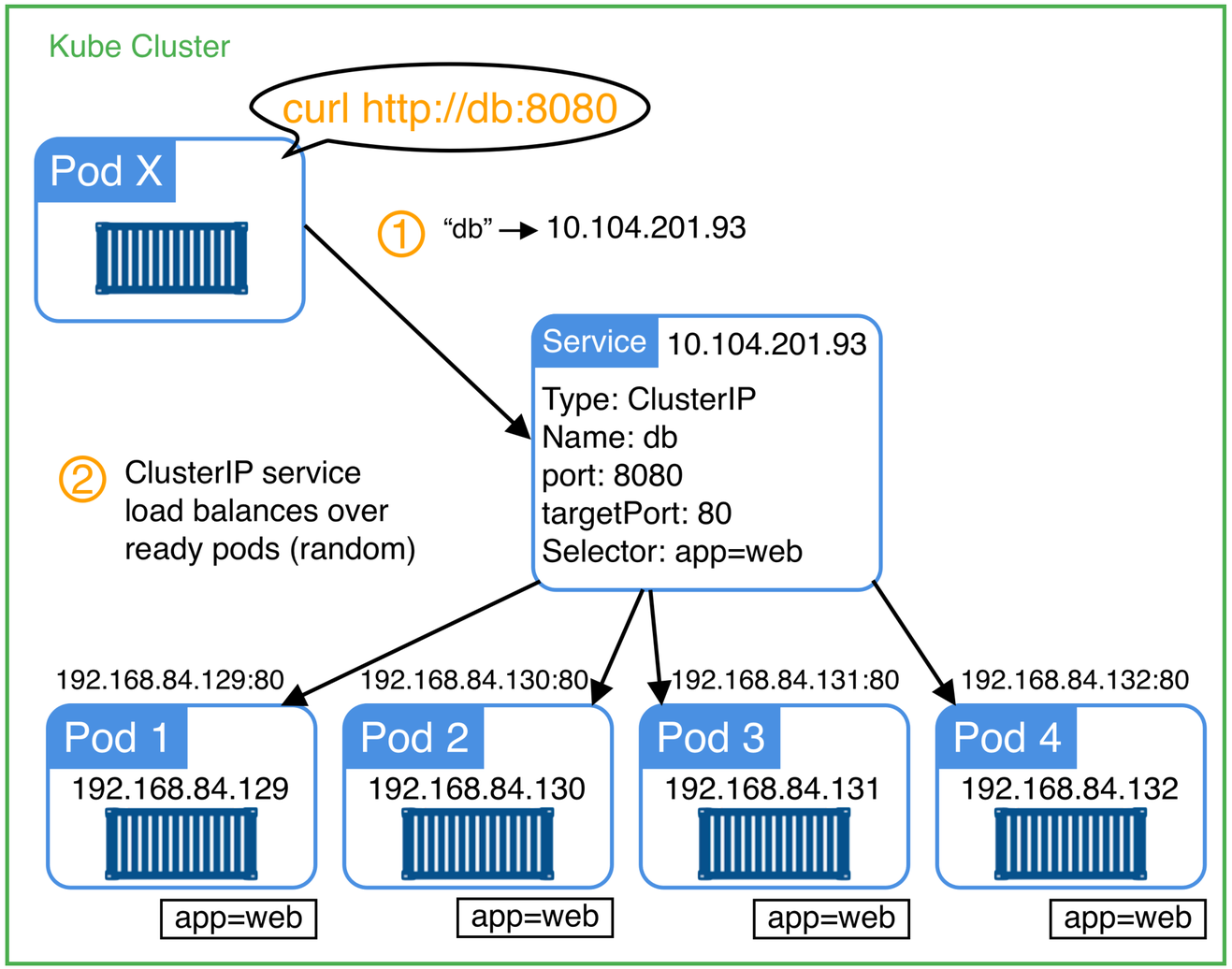
Solution:

// Service defines:

- Logical set of Pods
- Policy how to access them



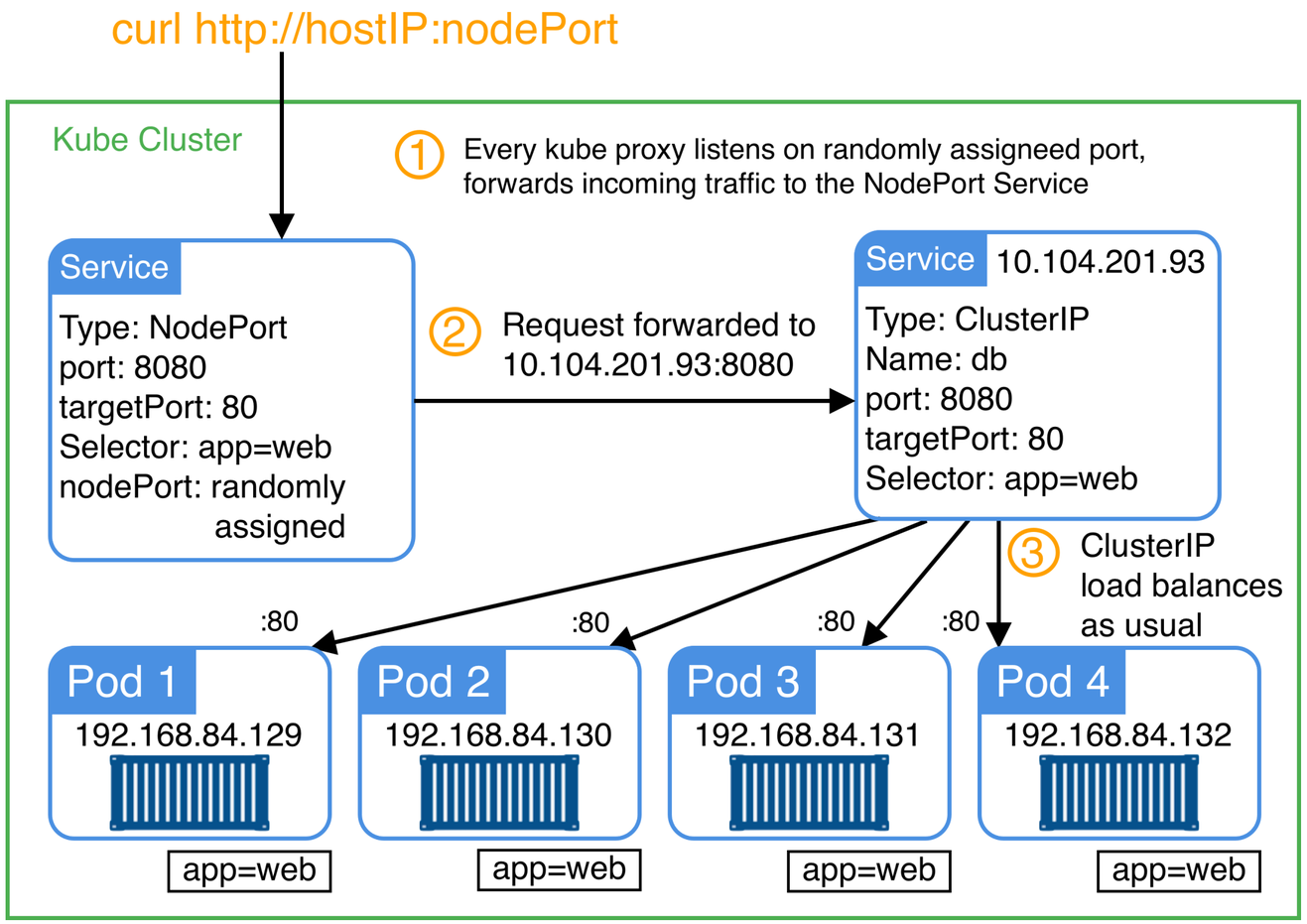
# CLUSTERIP SERVICES




// **Usecase:**

- Cluster internal origin
- Stateless destination
- Similar to Swarm VIP

# NODEPORT SERVICES




// **Usecase:**

- Cluster external origin
- Stateless destination
- Similar to Swarm L4 mesh

# KUBERNETES LABEL SELECTORS

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: nginx
  ports:
  - port: 8000
    targetPort: 80
```

gepardec

# NETWORK POLICIES

// Network policies **control traffic**

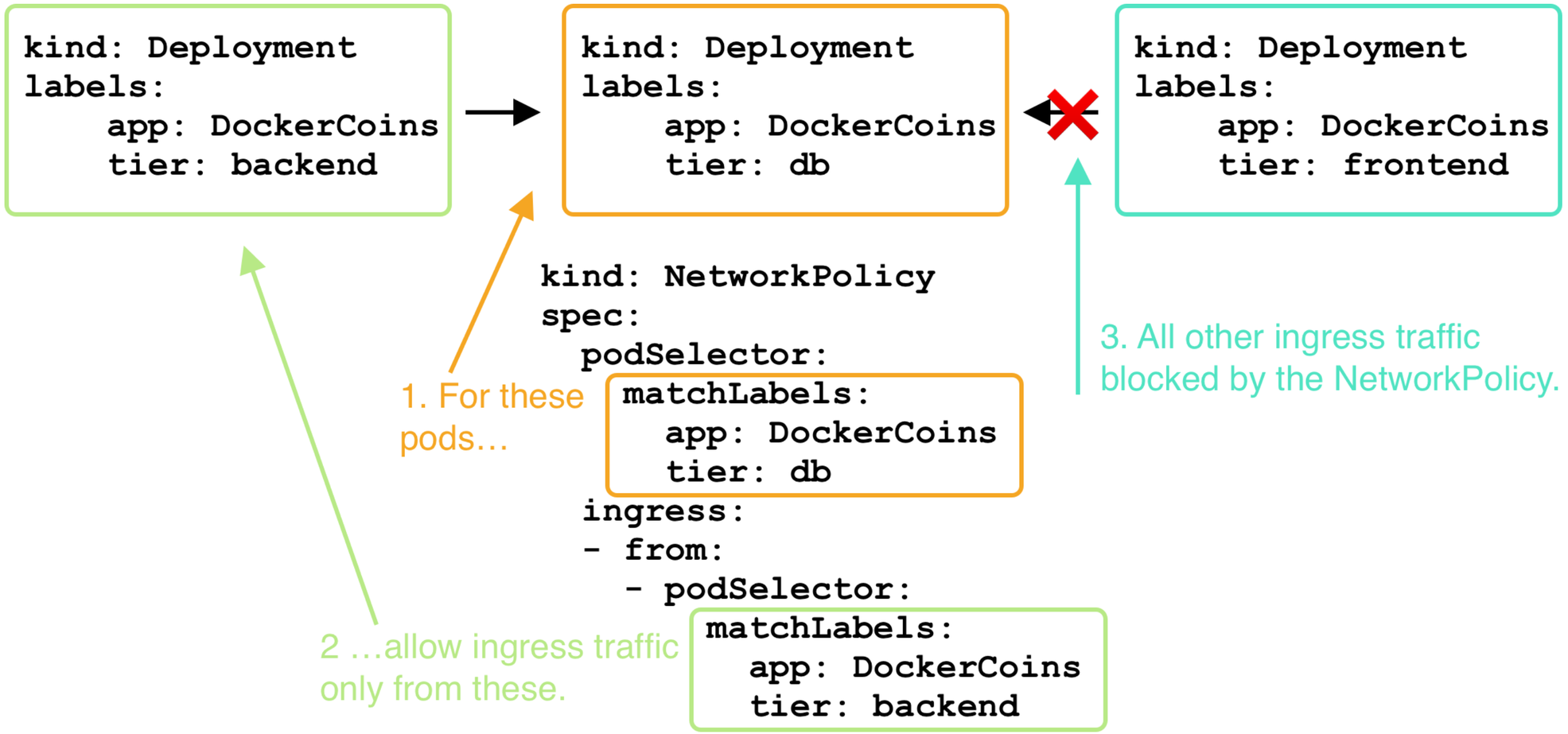
// Traffic allowed by default

// Traffic denied if network policy exist but no rule allows it

// Independent **ingress** & **egress** rules

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: ...
  namespace: ...
  ...
spec:
  podSelector: ...
  ingress:
  - ...
  - ...
  egress:
  - ...
  - ...
```

# SAMPLE NETWORK POLICIES



gepardec

# KUBERNETES NETWORKING PLANES

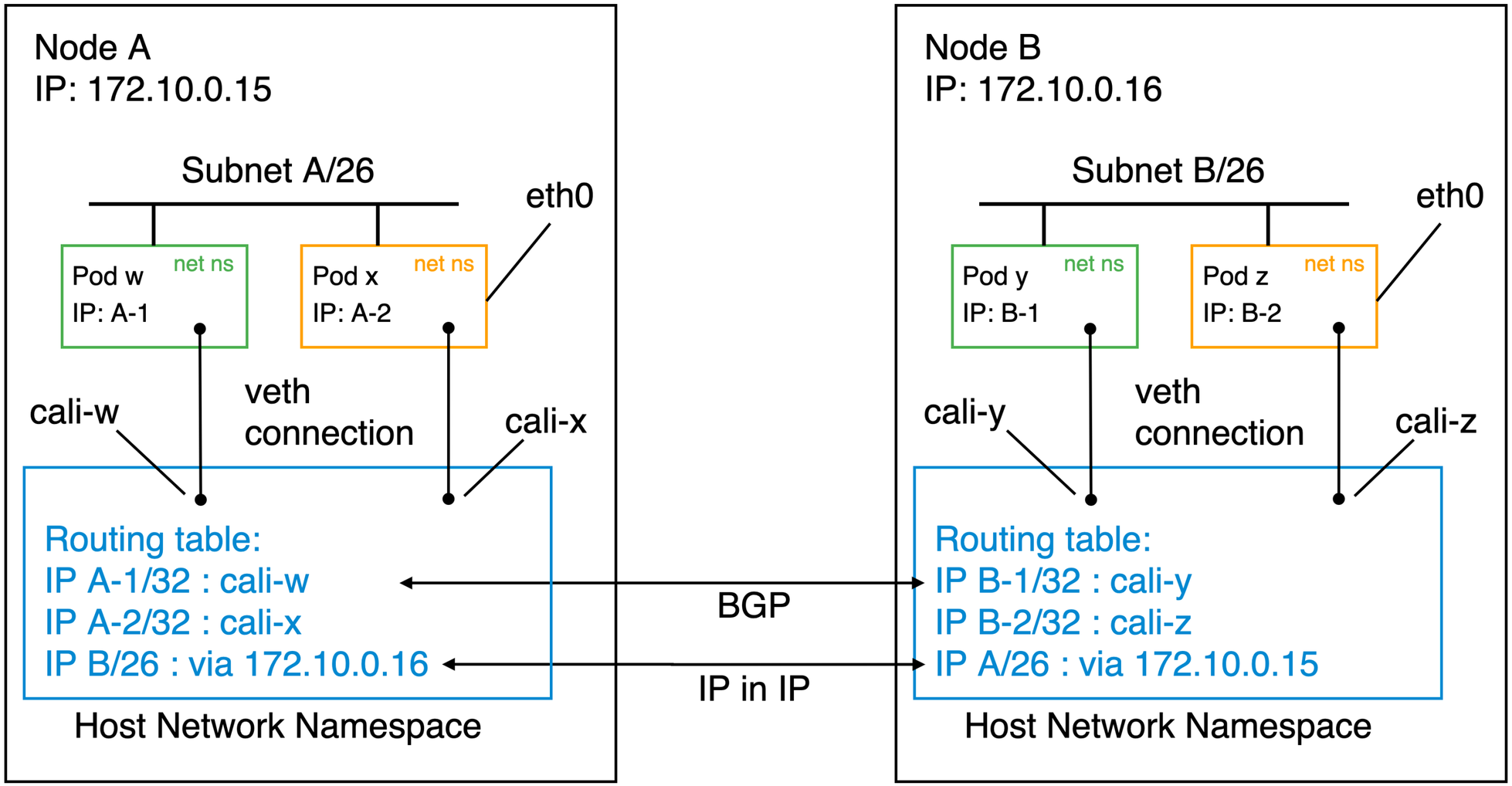
// Management:

- Master to Master: etcd Raft
- Master to Node: apiserver (TCP 6443) <-> kubelet (TCP 10250)

// Data & Control:

- BYO networking
- See Cluster DNS, http://bit.ly/2DMmdyt

# CALICO


Node A
IP: 172.10.0.15
Subnet A/26
eth0
Pod w
IP: A-1
net ns
Pod x
IP: A-2
net ns
cali-w
veth
connection
cali-x
Routing table:
IP A-1/32 : cali-w
IP A-2/32 : cali-x
IP B/26 : via 172.10.0.16
Host Network Namespace
BGP
IP in IP
Node B
IP: 172.10.0.16
Subnet B/26
eth0
Pod y
IP: B-1
net ns
Pod z
IP: B-2
net ns
cali-y
veth
connection
cali-z
Routing table:
IP B-1/32 : cali-y
IP B-2/32 : cali-z
IP A/26 : via 172.10.0.15
Host Network Namespace

# EXERCISE: KUBERNETES NETWORKING

Work through

// Kubernetes Networking

In the Exercises book.

gepardec

# KUBERNETES TAKEAWAYS

// Kubernetes provides more flexibility in its orchestration objects at the cost of more config

# FURTHER READING

// Docker & Kubernetes: https://www.docker.com/kubernetes

// Official Kubernetes Docs: https://kubernetes.io/docs

// Tutorials: http://bit.ly/2yLGn61

// Interactive Tutorials: https://bit.ly/2rdwlVZ

// Understanding Kubernetes Networking: http://bit.ly/2kdl1qQ

// Kubernetes the Hard Way: http://bit.ly/29Dq4wC

gepardec

# INGRESS

gepardec

# KUBERNETES INGRESS

// Exposes Services to the outside world

// Acts as Layer 7 LoadBalancer

// Manages domain and path based routing

// Always routes to services

// Manages tls encryption

gepardec

# INGRESS CONTROLLER

// Software that accomplishes ingress routing

// Needs to be installed on top of Kubernetes

// Examples: Nginx, Trafeik, HAProxy

gepardec

# INGRESS EXAMPLE

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
spec:
  rules:
  - host: "app.foo.com" #hostname
    http:
      paths:
      - path: /foobar #path
        pathType: Prefix
        backend:
          service:
            name: test #servicename
            port:
              number: 80 #port of service
```

# EXERCISE: KUBERNETES INGRESS

Work through

// Kubernetes Ingress

in the Exercises book.

# FURTHER READING

// Docker & Kubernetes: https://www.docker.com/kubernetes

// Official Kubernetes Docs: https://kubernetes.io/docs

// Tutorials: http://bit.ly/2yLGn61

// Interactive Tutorials: https://bit.ly/2rdwlVZ

// Understanding Kubernetes Networking: http://bit.ly/2kdl1qQ

// Kubernetes the Hard Way: http://bit.ly/29Dq4wC

gepardec

# FUNDAMENTAL ORCHESTRATION TAKEAWAYS

// Distributed Application Architecture orchestrates one or more containers across one or more nodes

// Orchestrators abstract away the differences between processes and between nodes

// Orchestrators enhance scalability and stability

gepardec

# WRAP UP KUBERNETES

# EXERCISE: KUBERNETES WRAPUP

Work through

// Kubernetes Wrapup