

KUBERNETES TRAINING

HOW WE TEACH

- // We believe in **learning by doing**
- // The training is **lab driven**
- // Work together!
- // Ask questions at any time

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/2mTQr8I>

YOUR LAB ENVIRONMENT

- // You have been given an instances for use in exercises
- // Ask instructor for credentials if you don't have them already

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

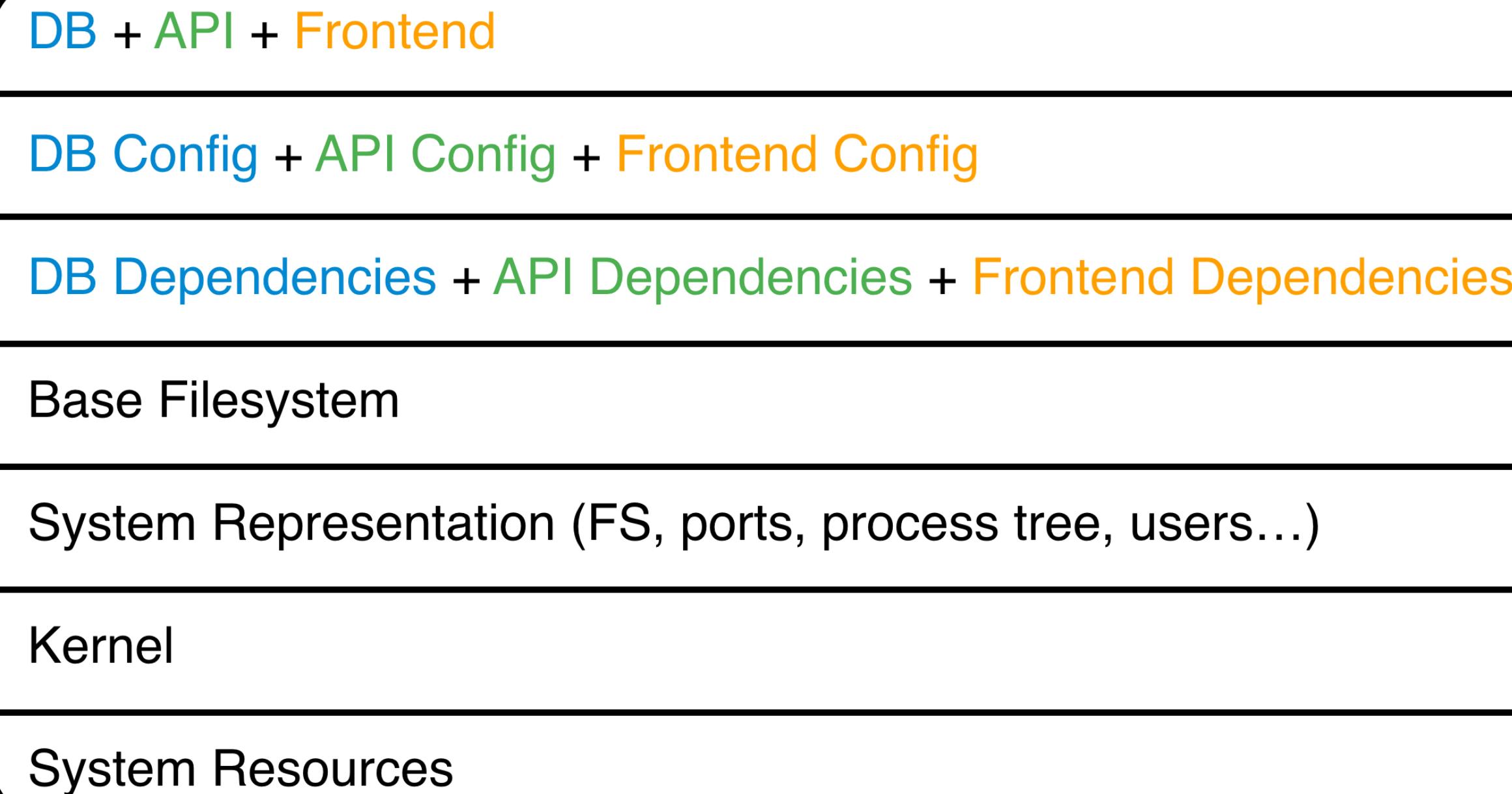
INTRODUCING DOCKER

WHAT WE WANT

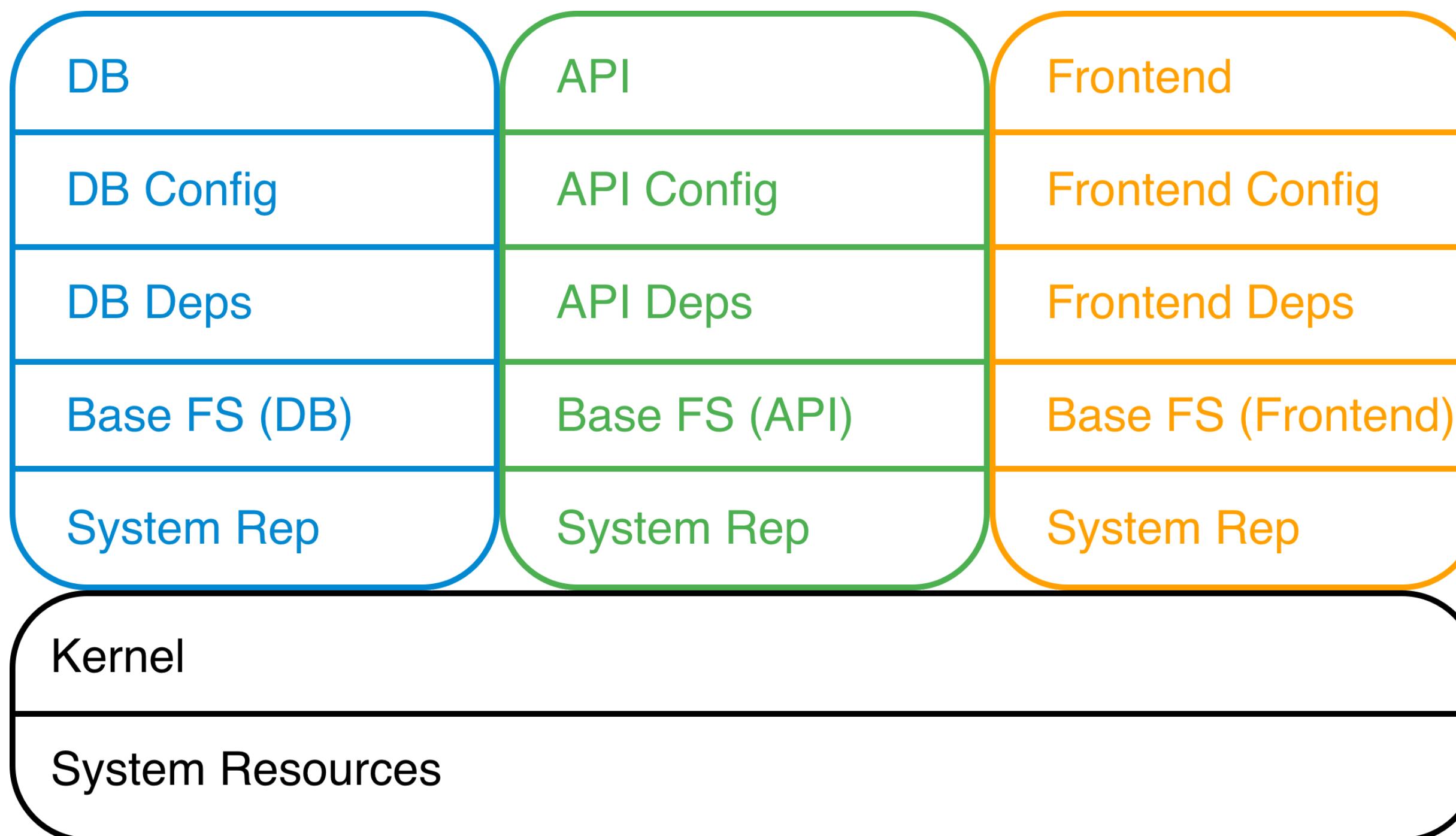
Ideal software should

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

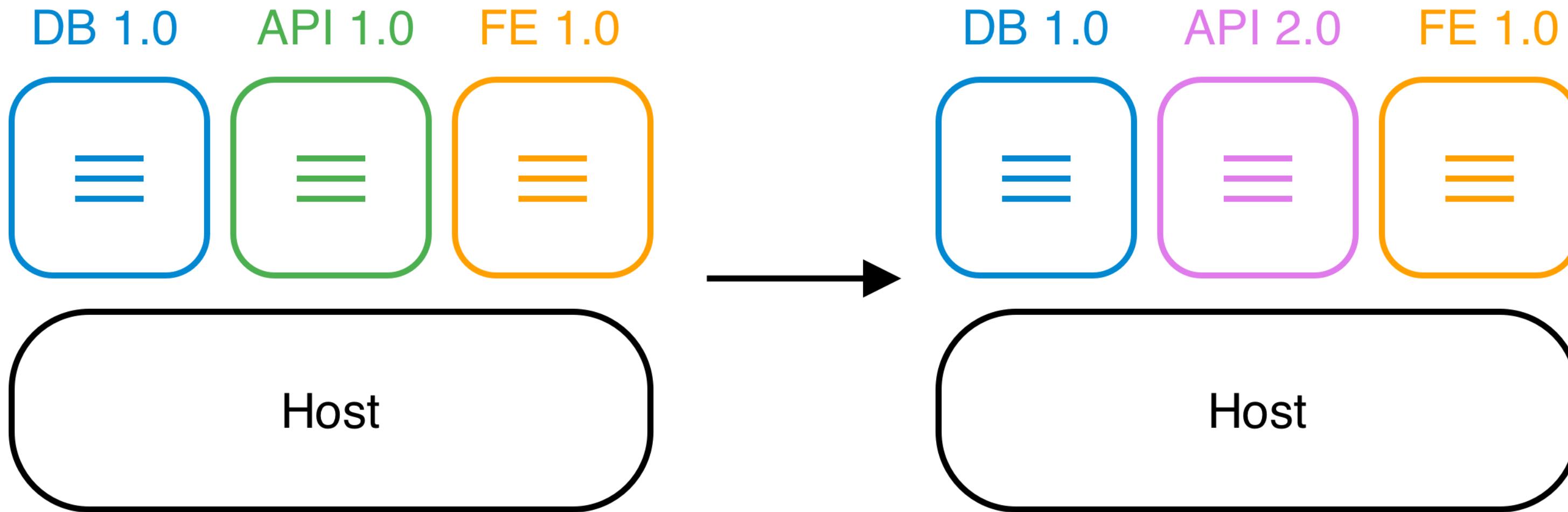
WITHOUT CONTAINERIZATION



WITH CONTAINERIZATION

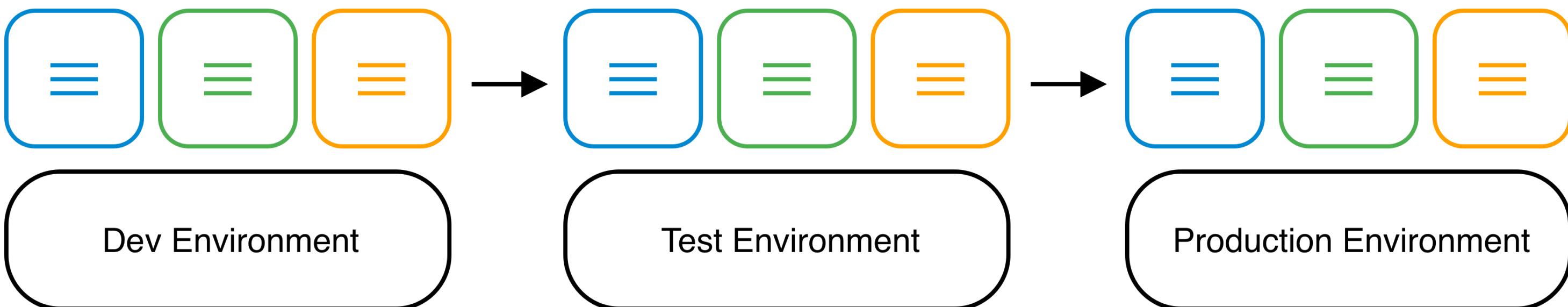


RAPID DEVELOPMENT



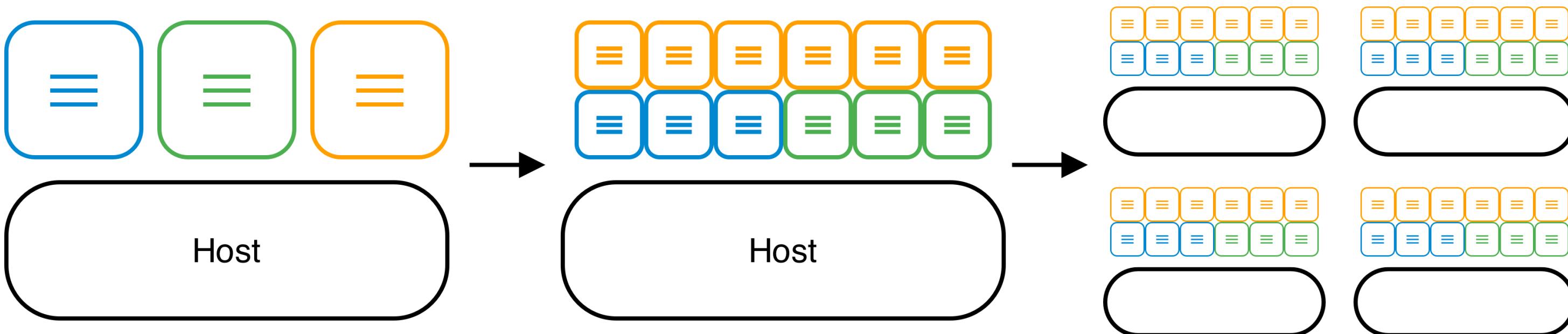
Containers can be removed and replaced with a minimum of impact on their neighbors, increasing developer choice and speed.

SMOOTH MIGRATION



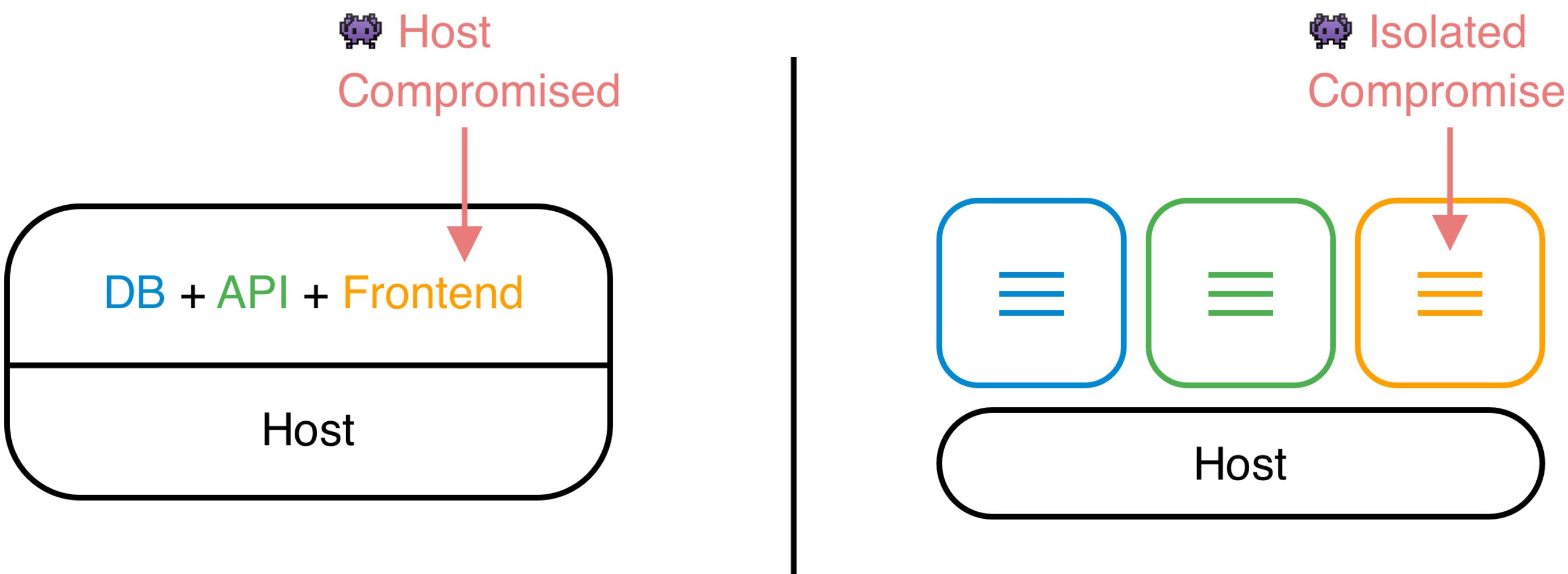
Containers carry their environment and dependencies with them, simplifying and minimizing requirements on the hosts that run them.

SIMPLE SCALE & MAINTENANCE



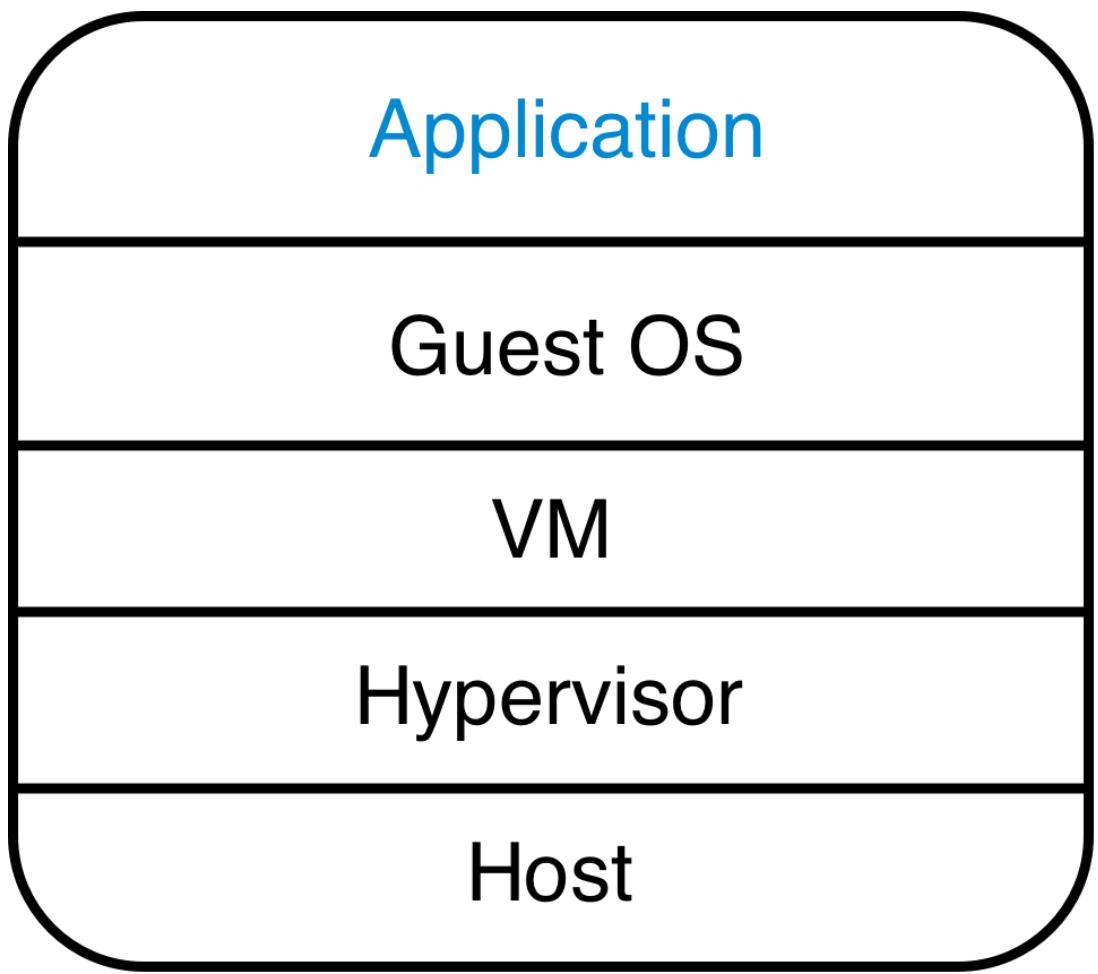
Weak coupling between containers minimizes side effects when scaling and simplifies monitoring.

SECURE BY DEFAULT



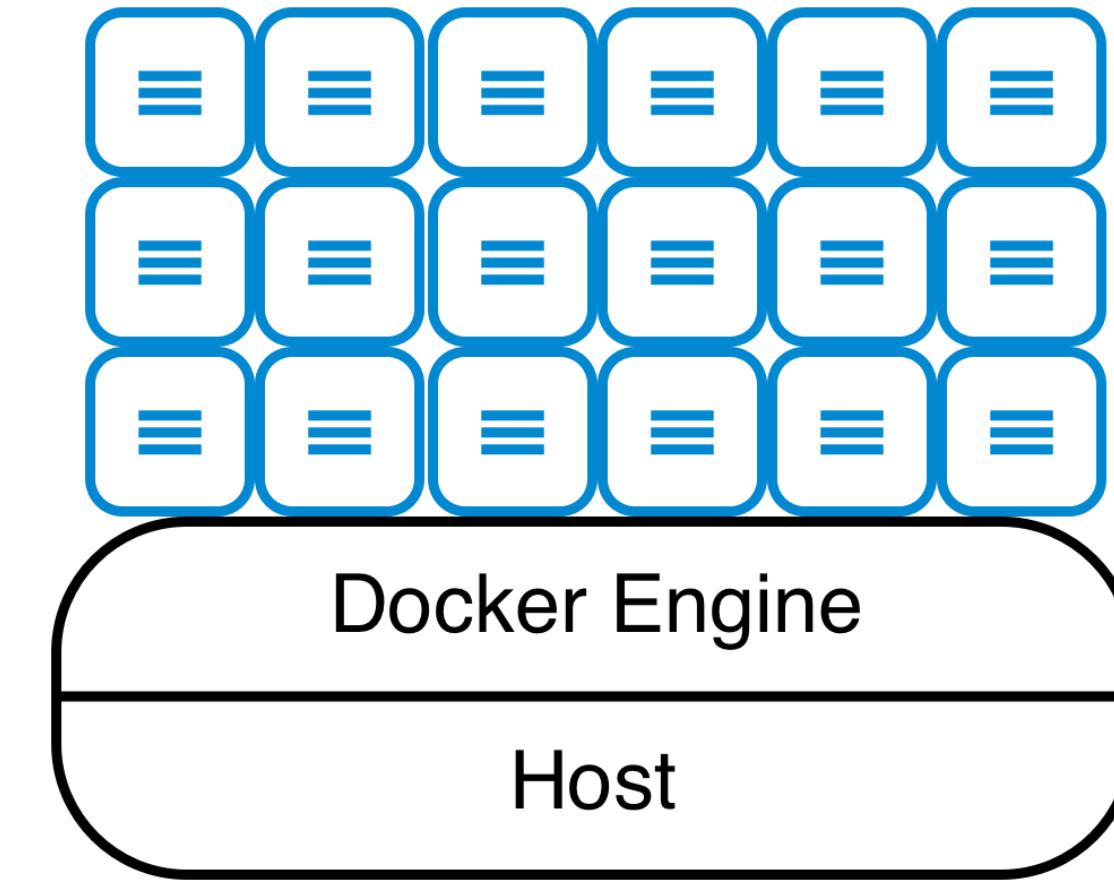
Containers have private system resources, so a compromise in one does not affect the rest.

APPLICATION DENSITY



Virtual Machines

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



Containers

INTRODUCTION TO DOCKER COMPOSE

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

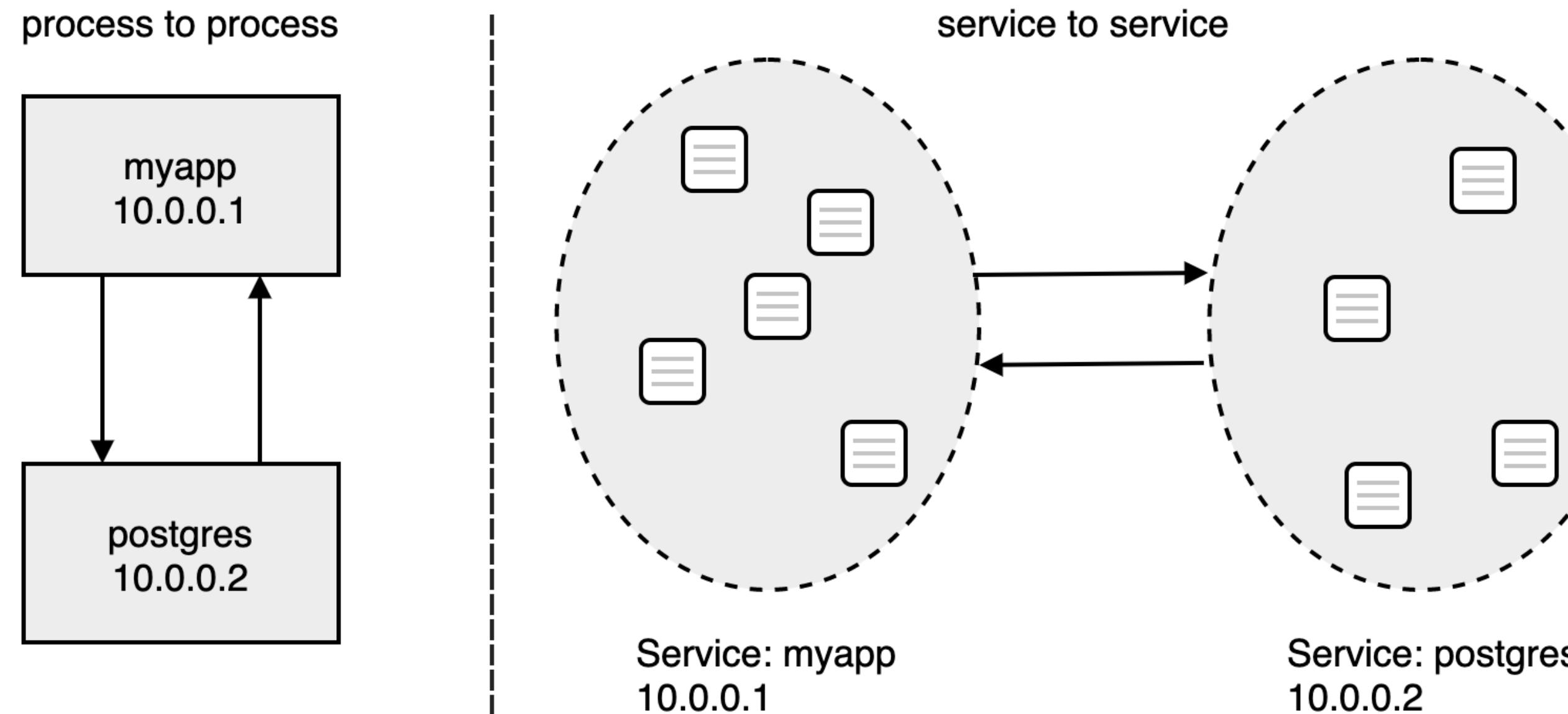
// Applications consisting of one or more containers across one or more nodes

// 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.

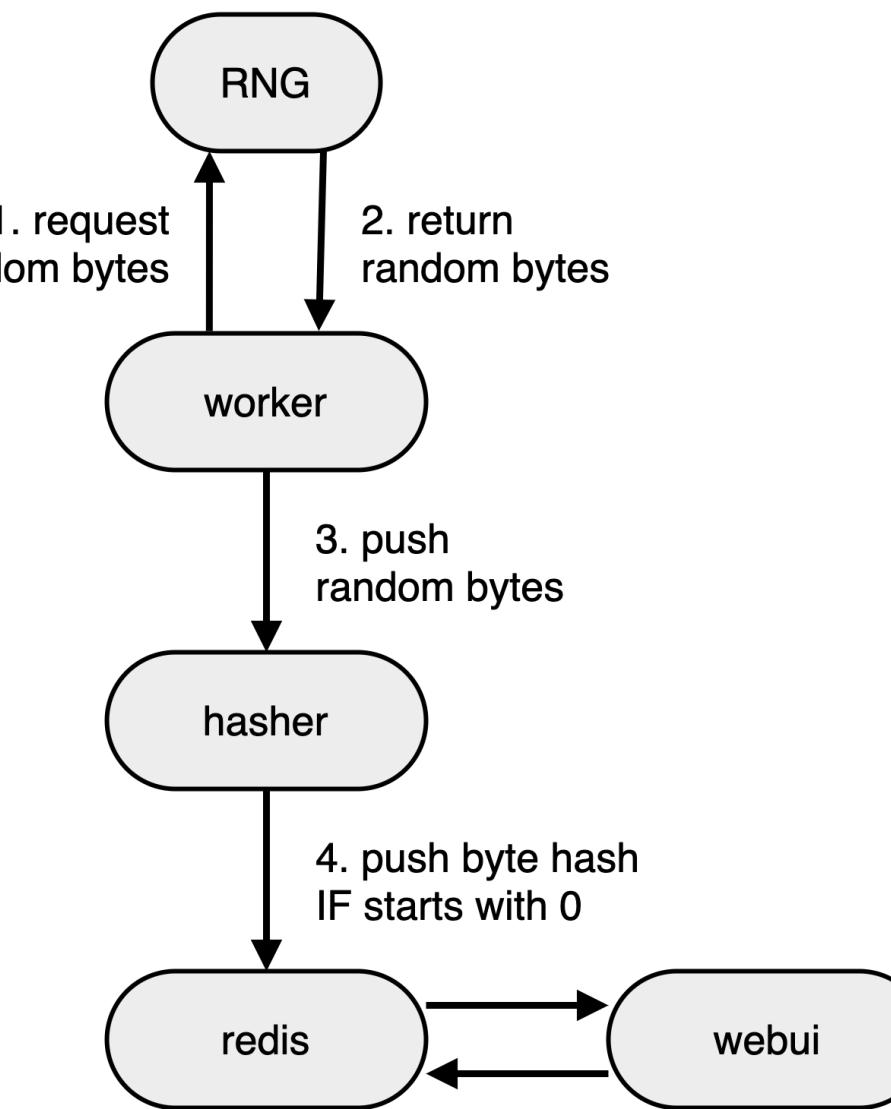
OUR APPLICATION: DOCKERCOINS



(DockerCoins 2016 logo courtesy of [@XtICnsIt](#) and [@ndelooft](#). 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.

DOCKER COMPOSE TAKEAWAYS

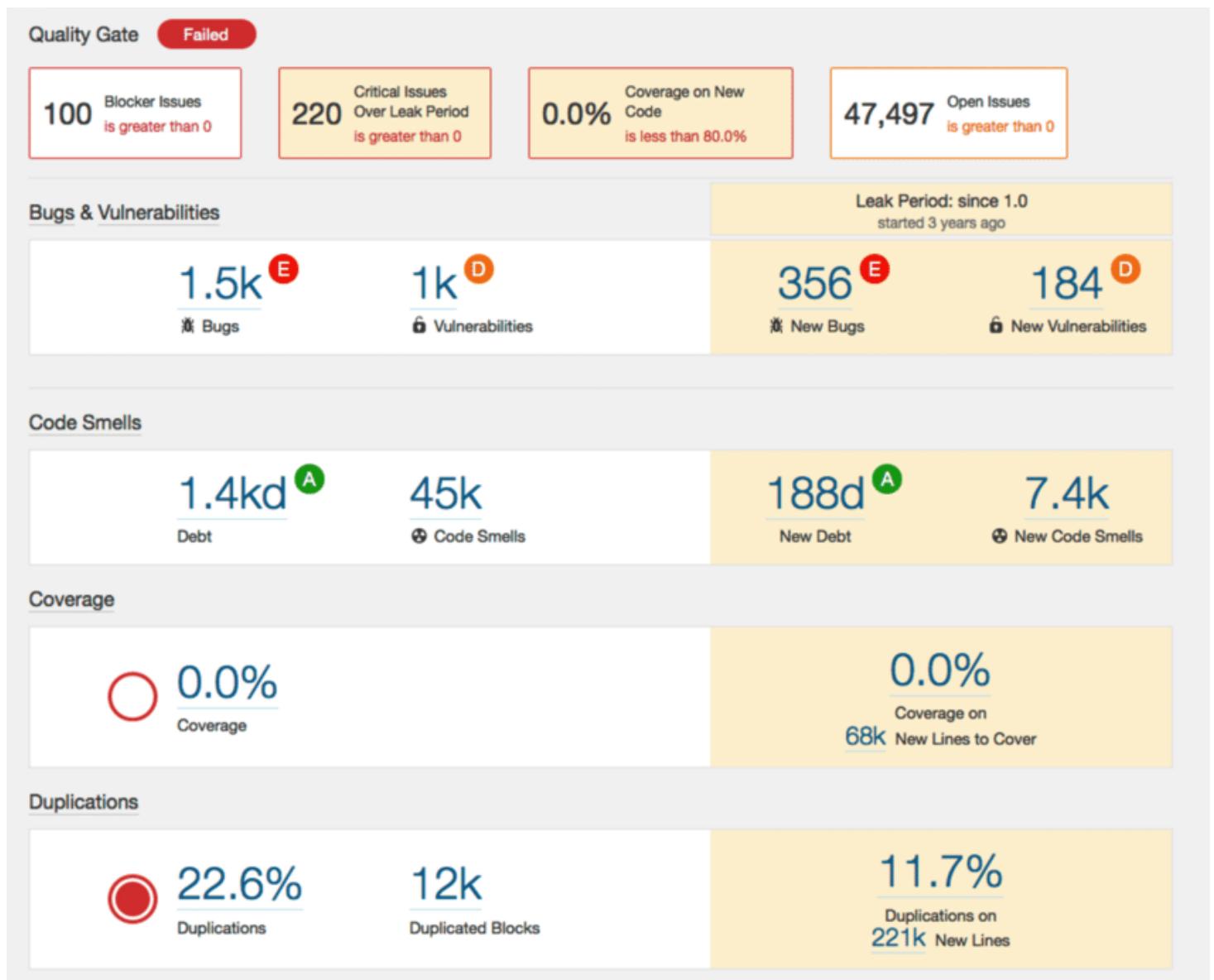
- // Docker Compose makes single node orchestration easy
- // Compose services makes scaling applications easy
- // Bottleneck identification important
- // Syntactically: `docker-compose.yml` + API

FURTHER READING

- // Docker compose examples: <http://dockr.ly/1FL2VQ6>
- // Overview of docker-compose CLI: <http://dockr.ly/2wtQIZT>
- // `docker-compose.yaml` reference: <http://dockr.ly/2iHUpex>
- // Docker Compose and Windows: <http://bit.ly/2watrqk>

WRAP UP DOCKER-COMPOSE - SONARQUBE

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 its 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

```
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:
  sonarqube_bundled-plugins:
  postgresql:
  postgresql_data:
```

INTRODUCTION TO KUBERNETES

DISCUSSION: ANY APP, ANYWHERE?

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

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.**

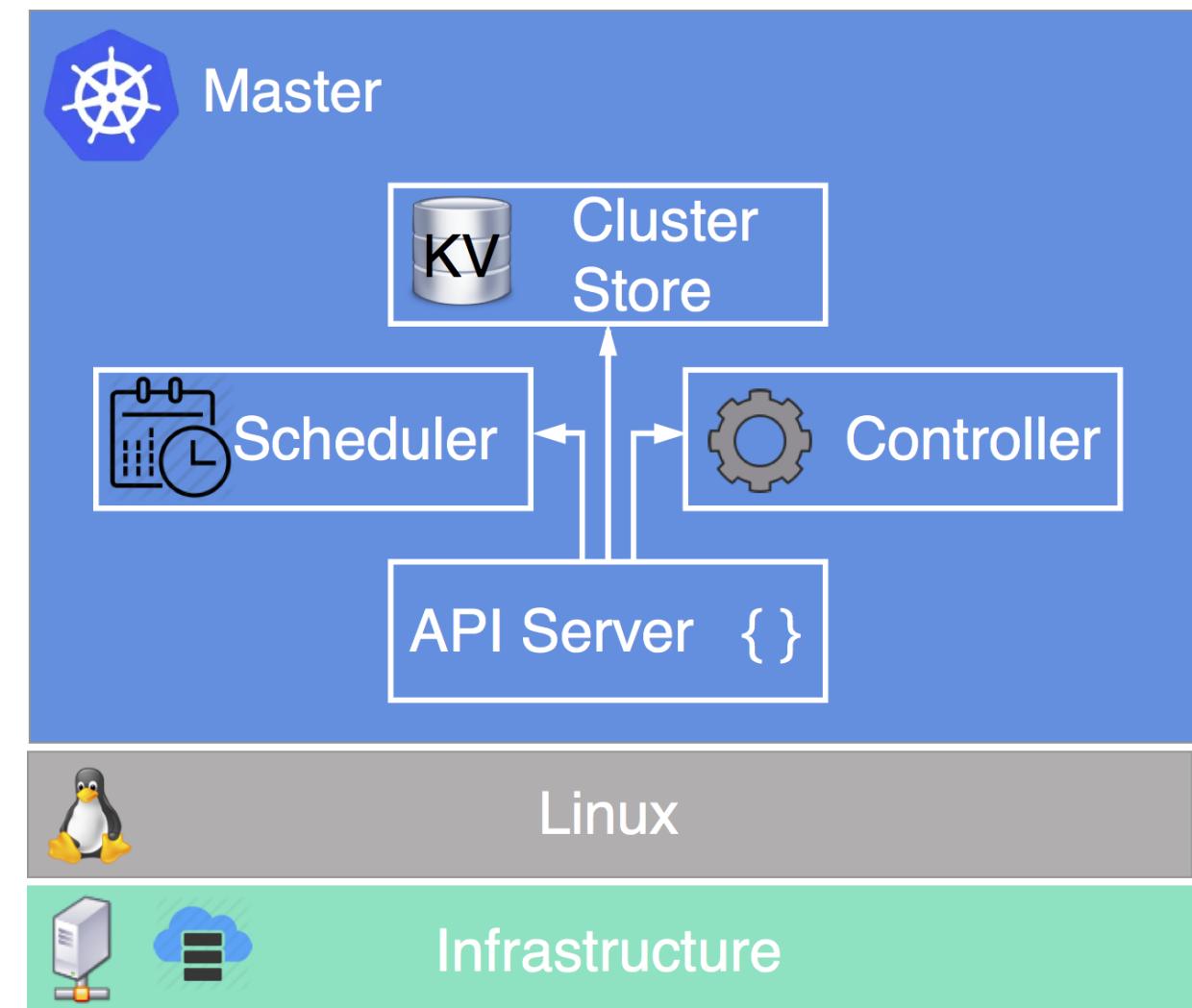


Dog photo [jeffreyw](#); Livestock photo [Paul Asman, Jill Lenoble](#); images CC-BY 2.0

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

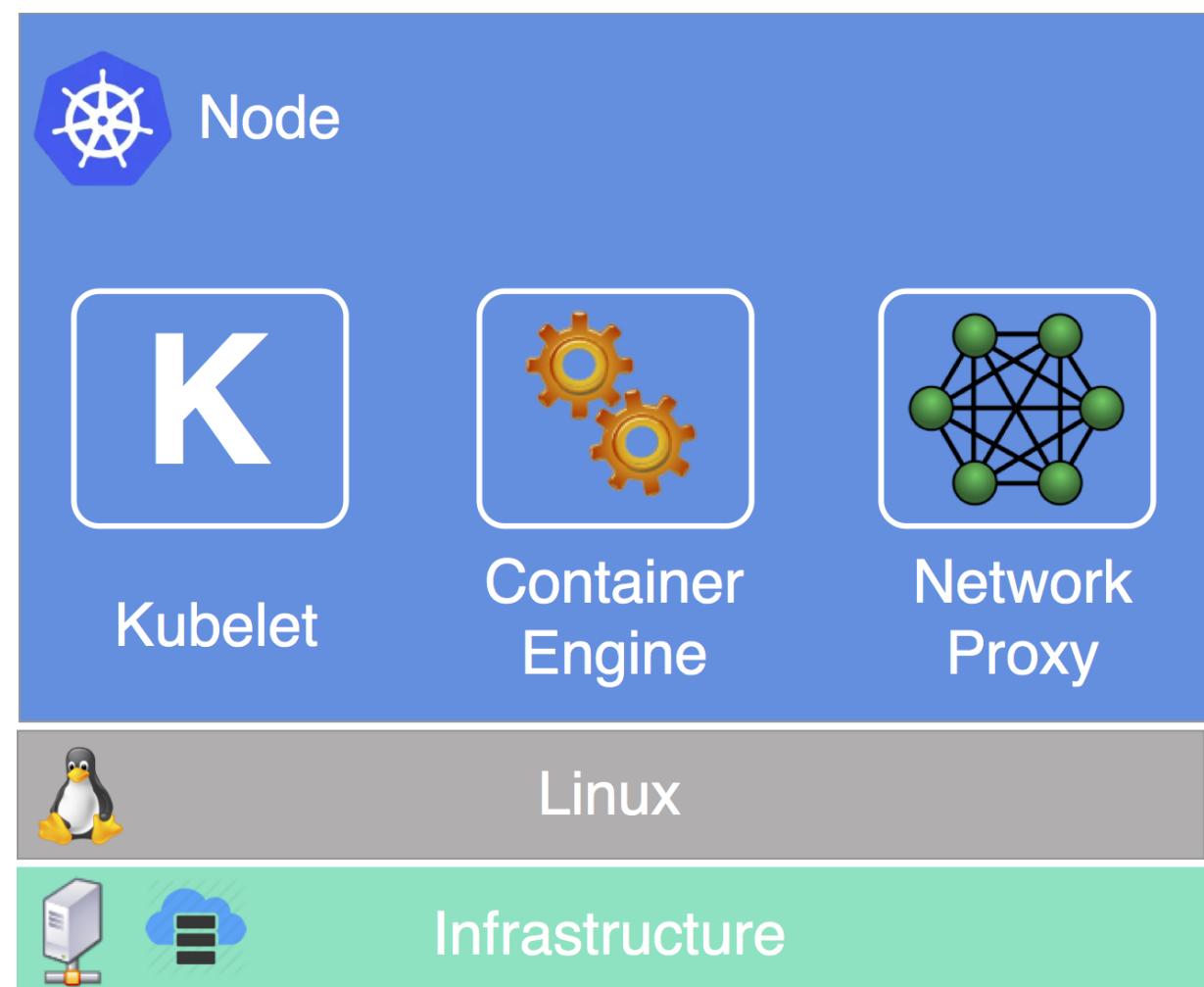


KUBERNETES NODE

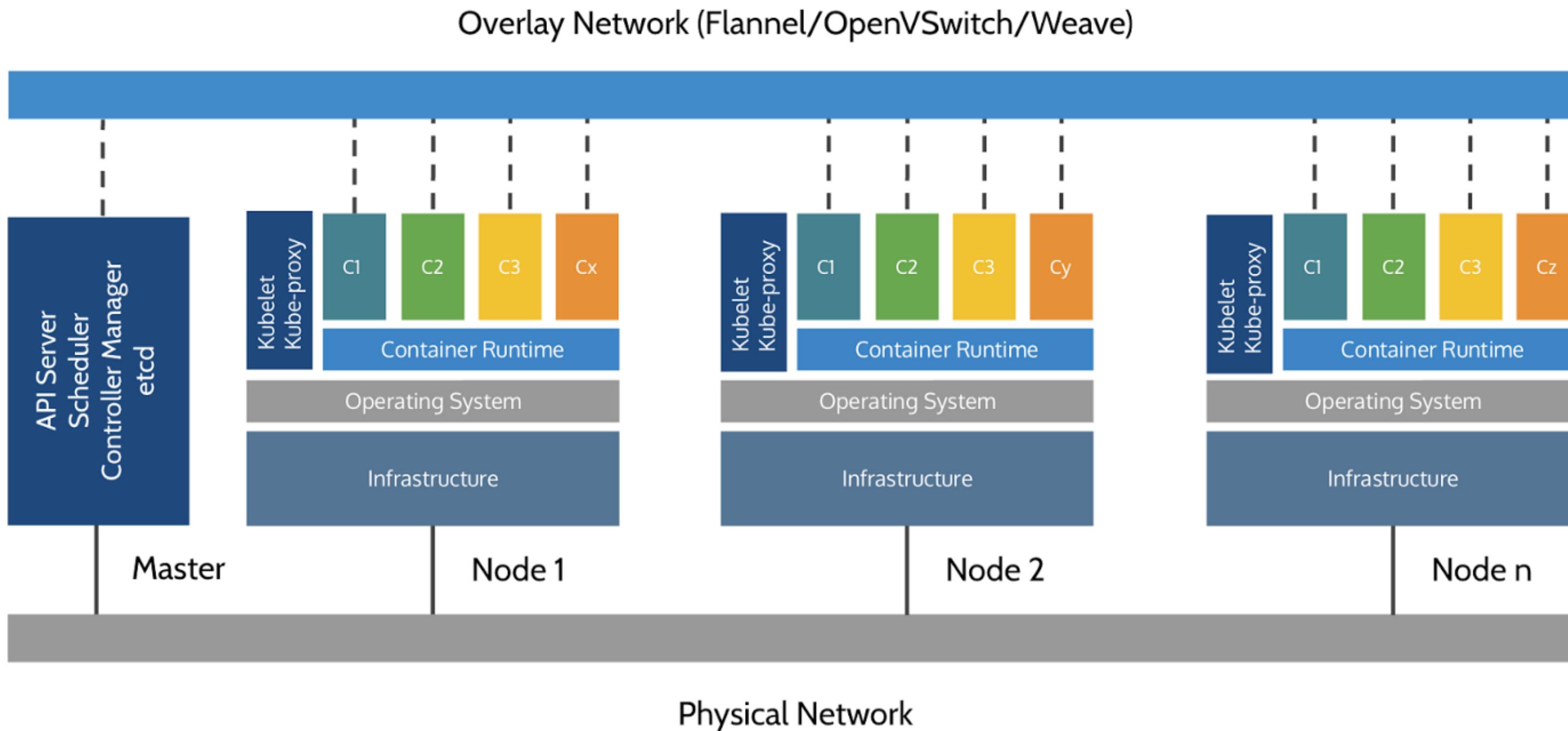
// **Kubelet**: Kubernetes Agent

// **Container Engine**: Host Containers

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



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/2kdI1qQ>
- // Kubernetes the Hard Way: <http://bit.ly/29Dq4wC>

KUBERNETES RESSOURCES

KUBERNETES ORCHESTRATION OBJECTS

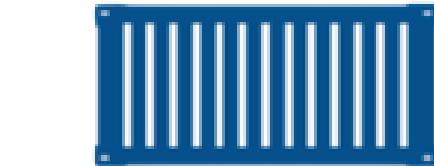
Deployment

Updates and Rollback

ReplicaSet

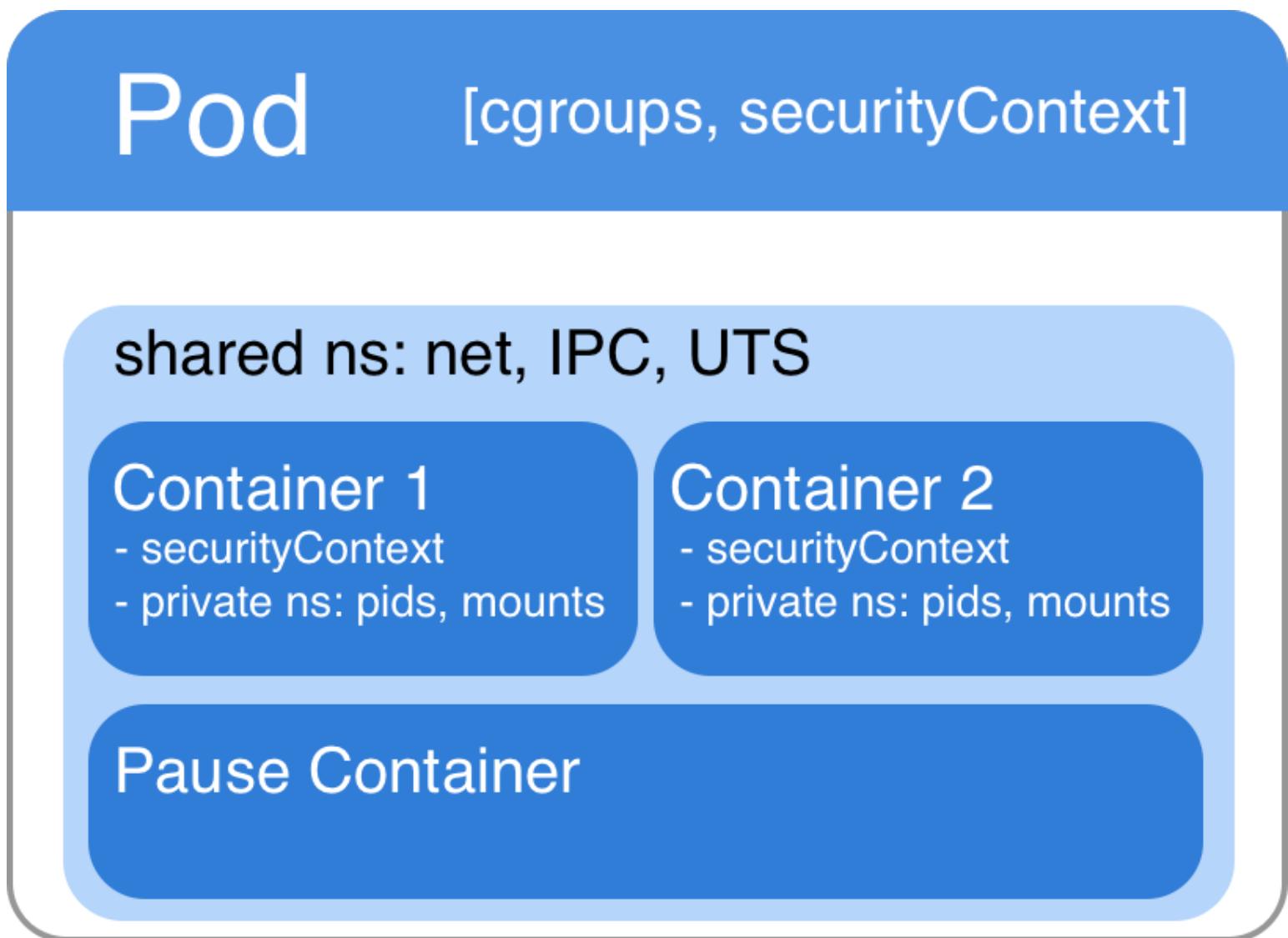
Self-healing, scalability, desired state

Pod



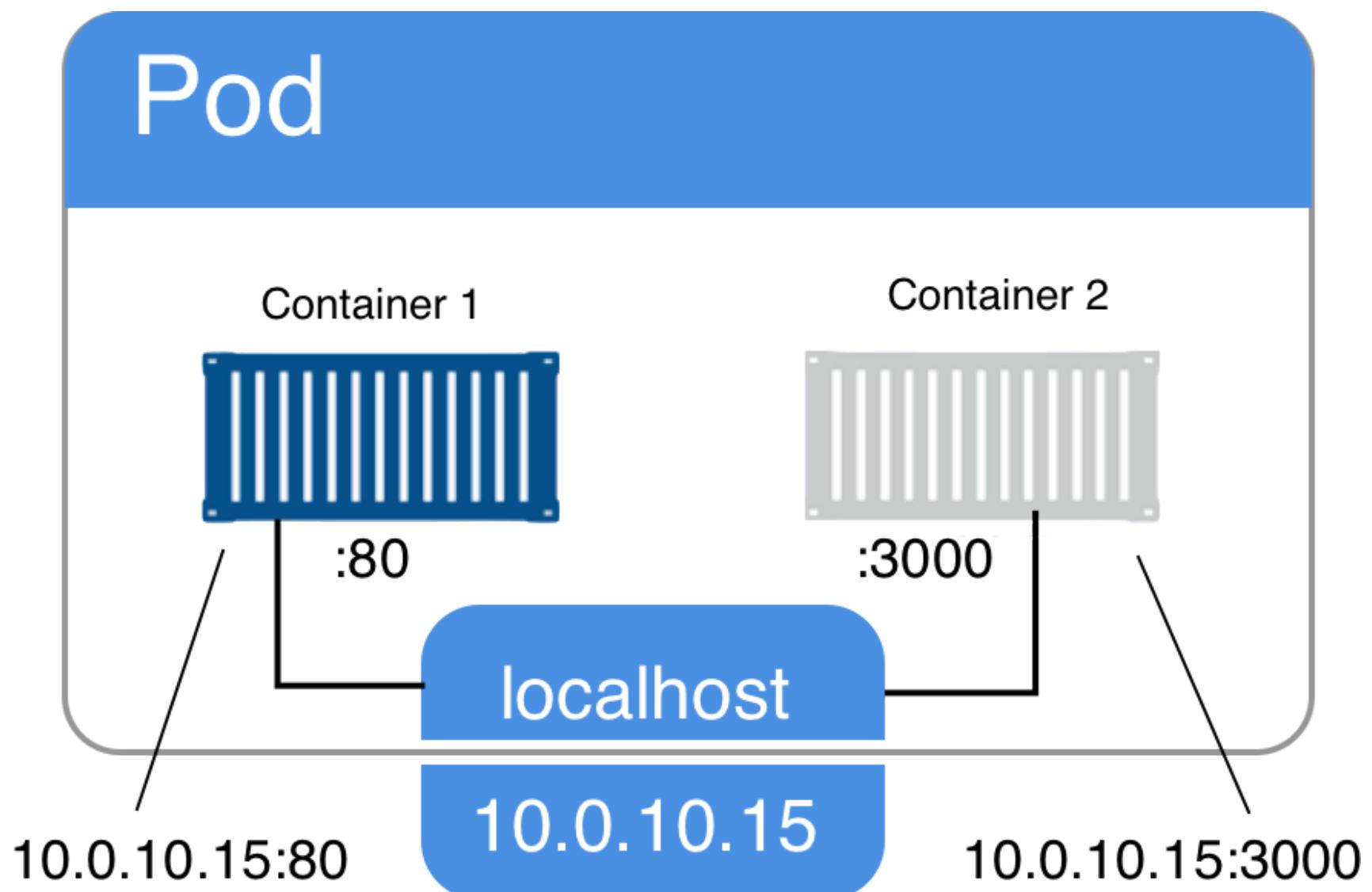
••• more pods

PODS



Logical host supporting interacting processes

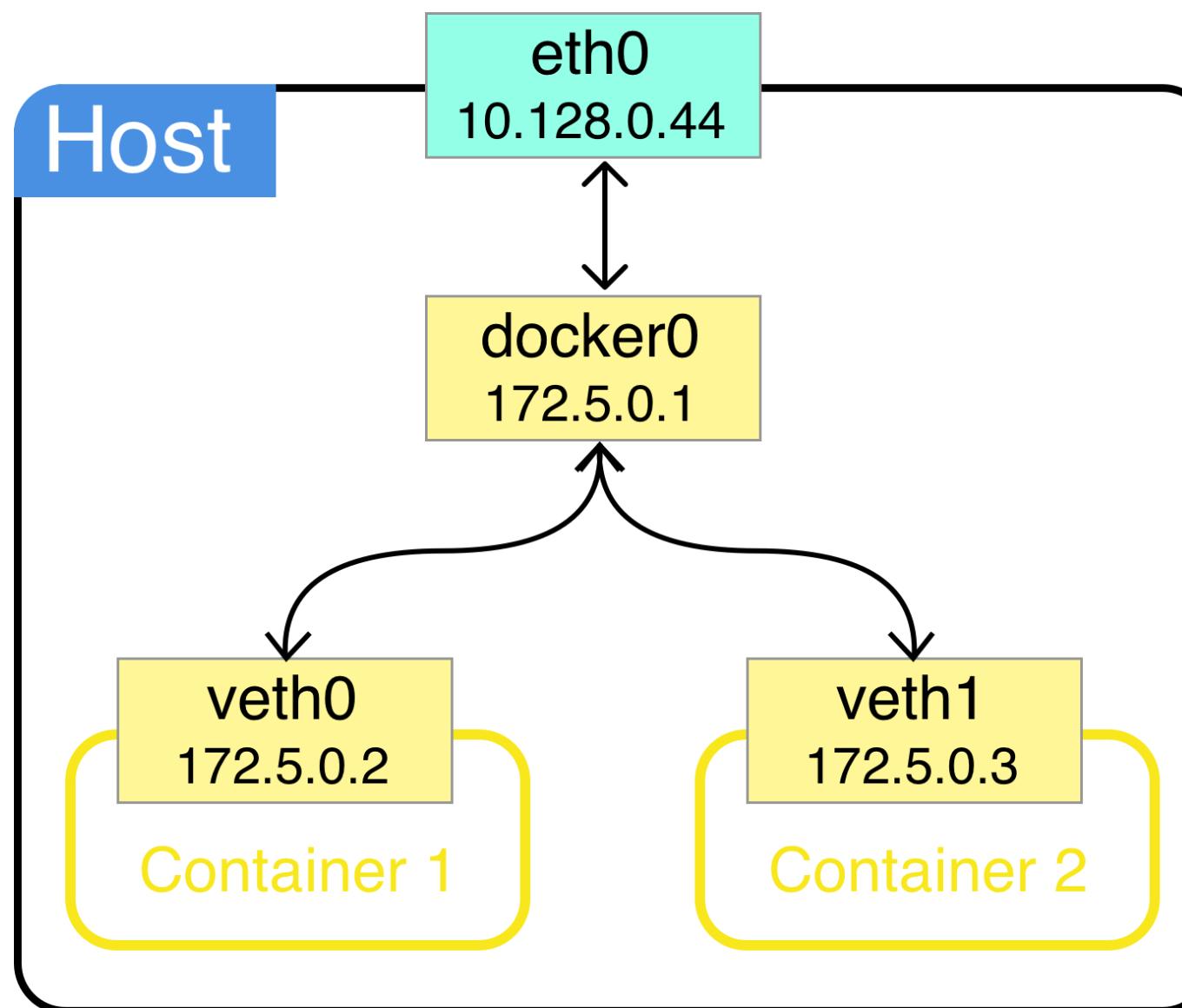
PODS



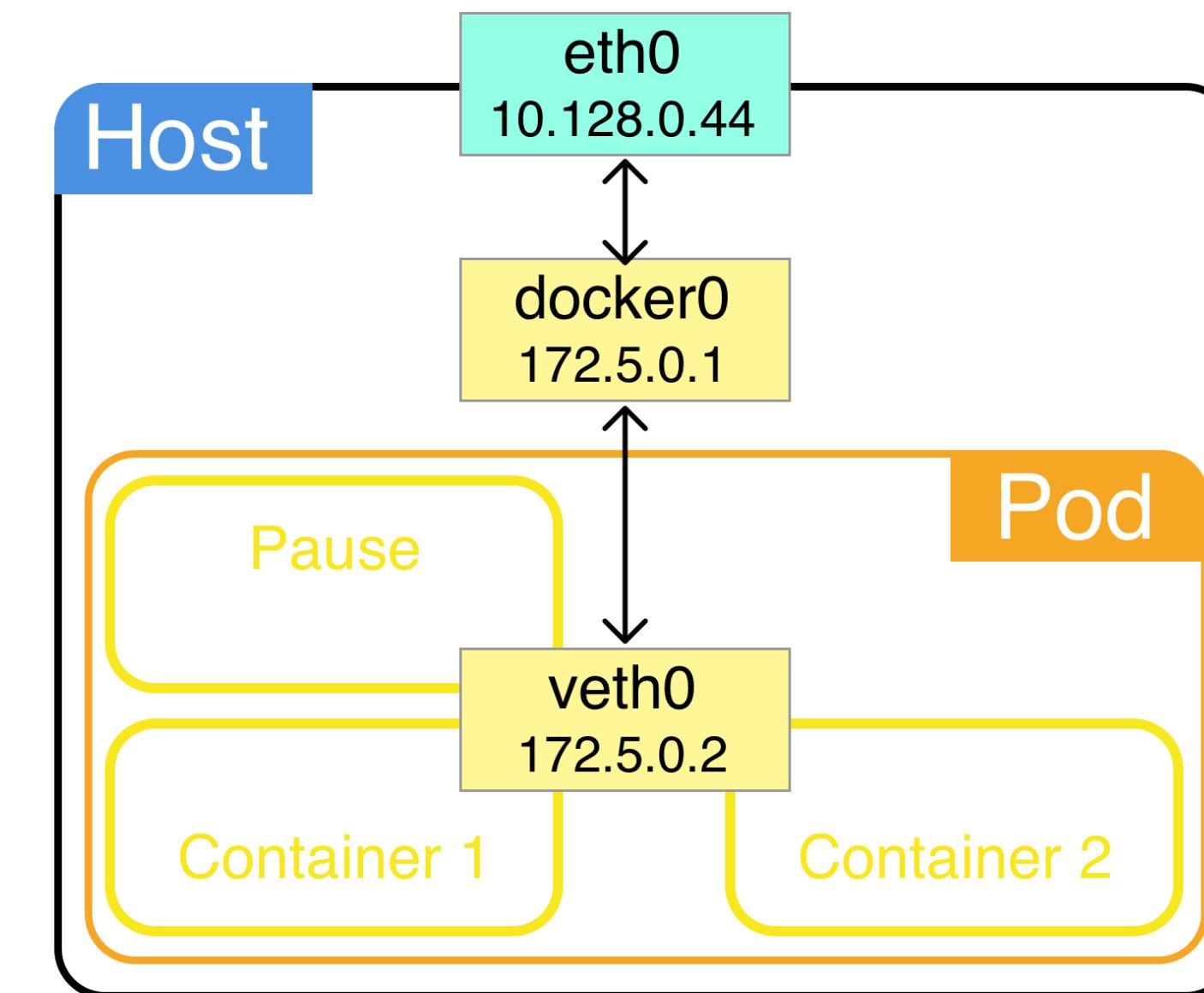
// Use `localhost` for intra-pod communication

// All containers in a pod share same IP

POD NETWORKING

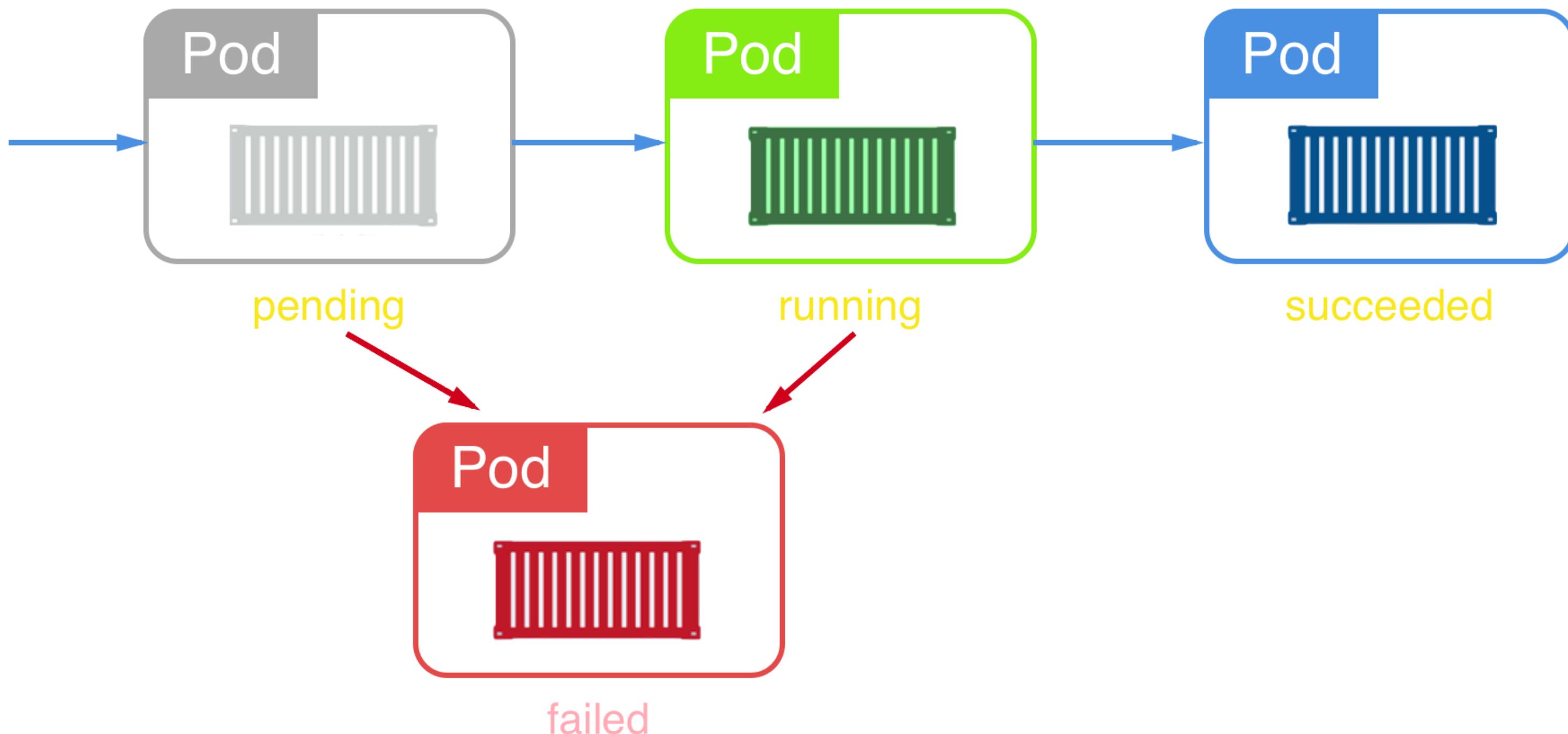


Docker Bridge Network

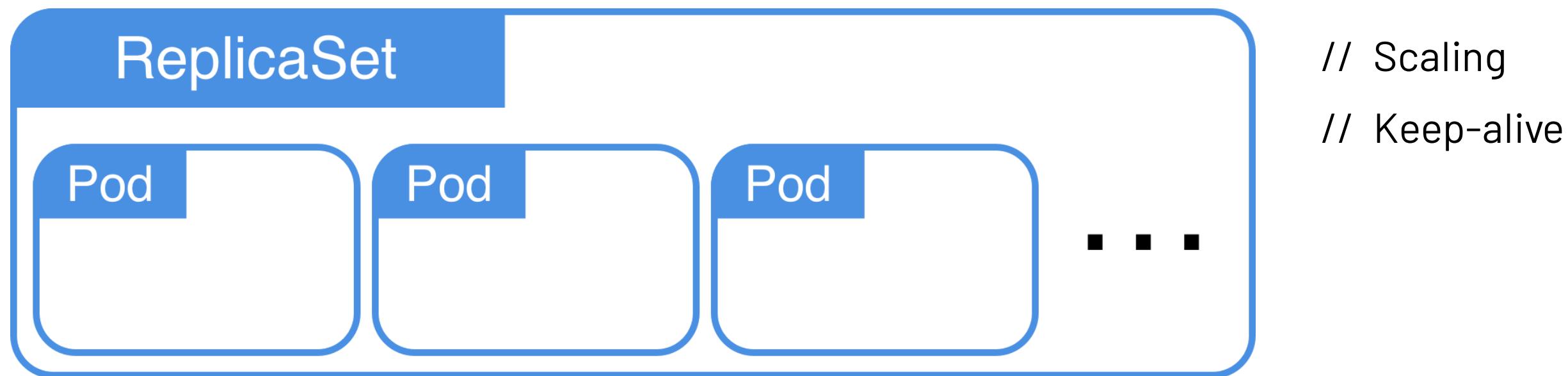


Kubernetes Network

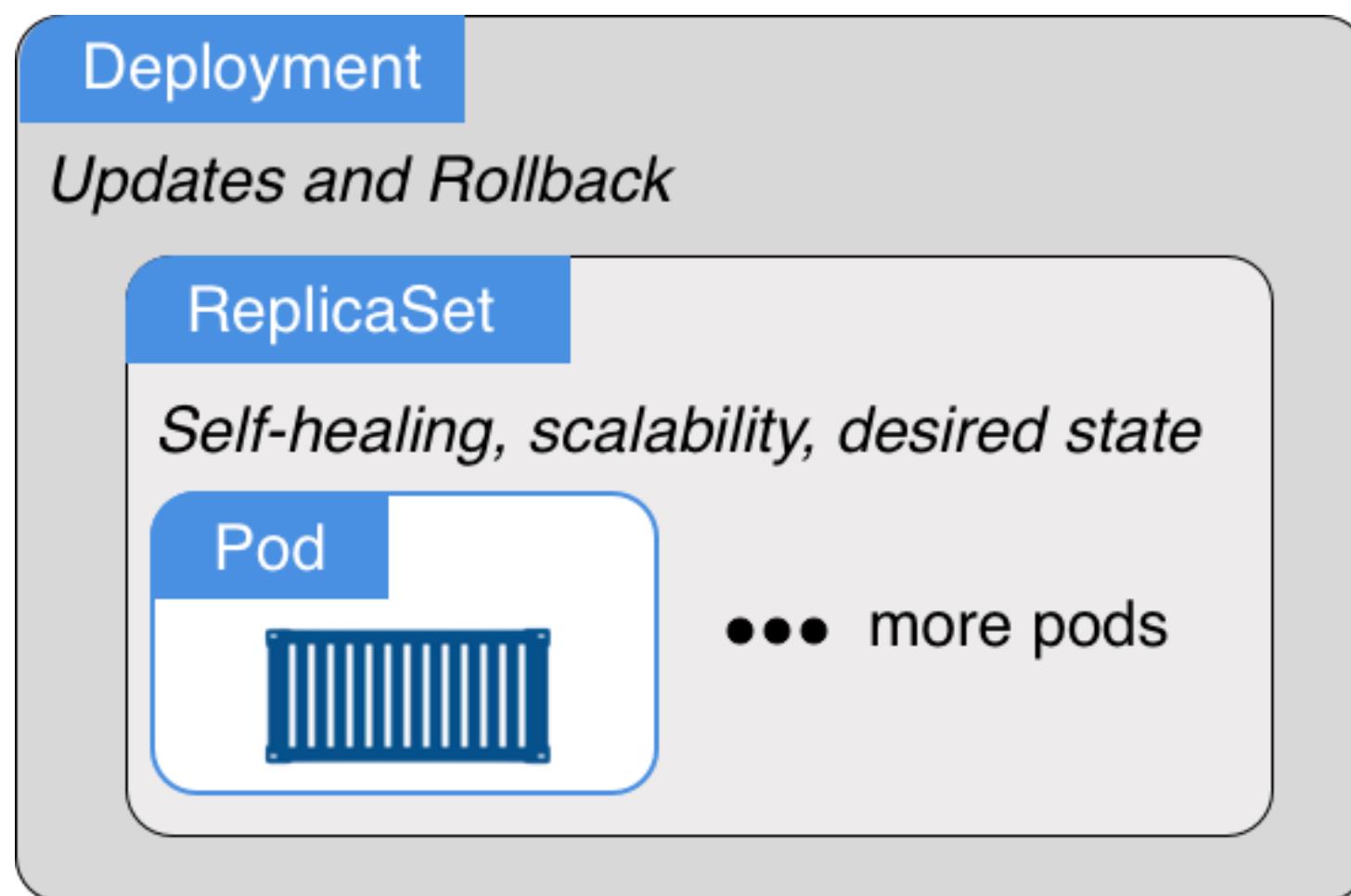
POD LIFECYCLE



REPLICASET



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/2kdI1qQ>
- // Kubernetes the Hard Way: <http://bit.ly/29Dq4wC>

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.

CONFIGMAPS

- // Collections of key/value pairs, or text files
- // Provisioned to containers via env vars or volume mounts
- // Appropriate for low/no security config

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/2kdI1qQ>
- // Kubernetes the Hard Way: <http://bit.ly/29Dq4wC>

VOLUMES

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
```



EXERCISE: CONFIGURATION, SECRETS & VOLUMES

Work through

// Provisioning Kube Config

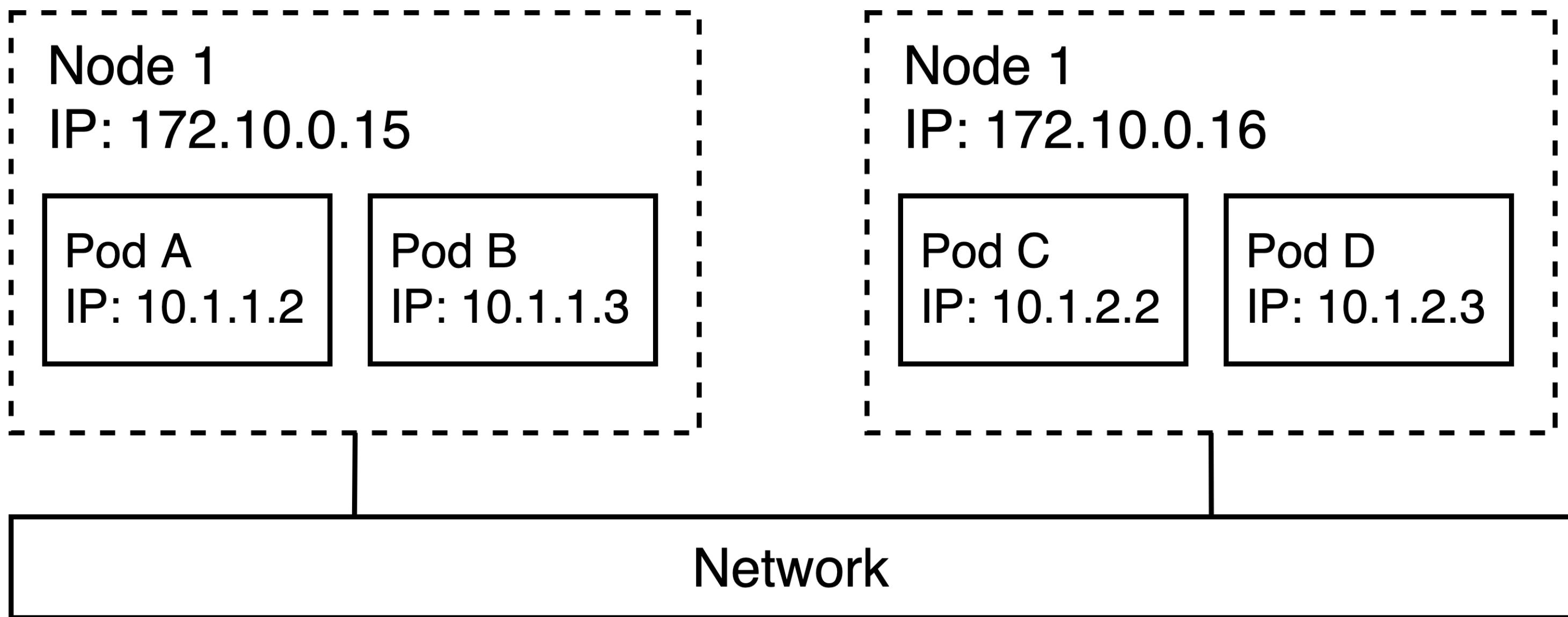
in the Exercises book.

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/2kdI1qQ>
- // Kubernetes the Hard Way: <http://bit.ly/29Dq4wC>

KUBERNETES NETWORKING

KUBERNETES NETWORK MODEL



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

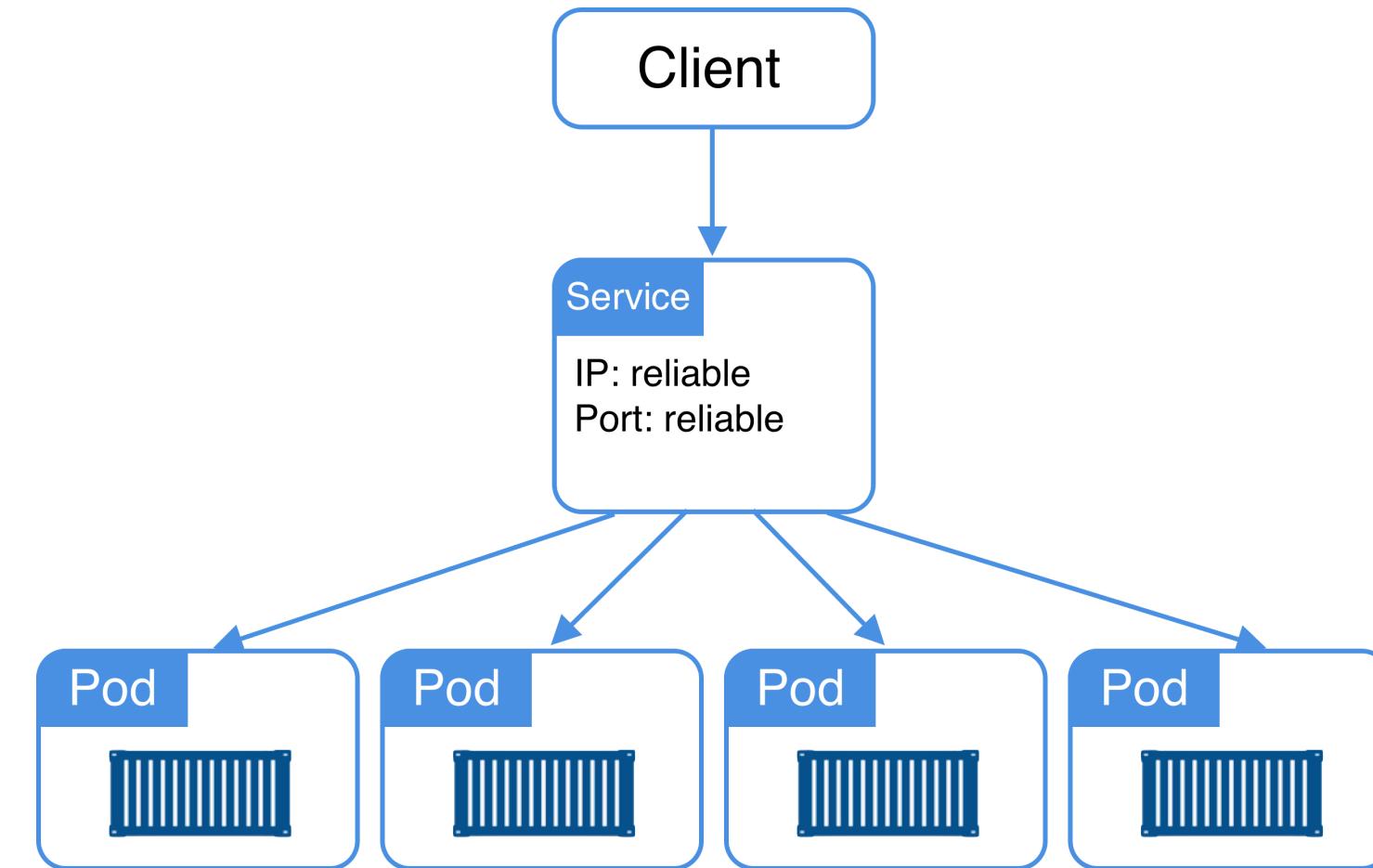
SERVICE

Problem:

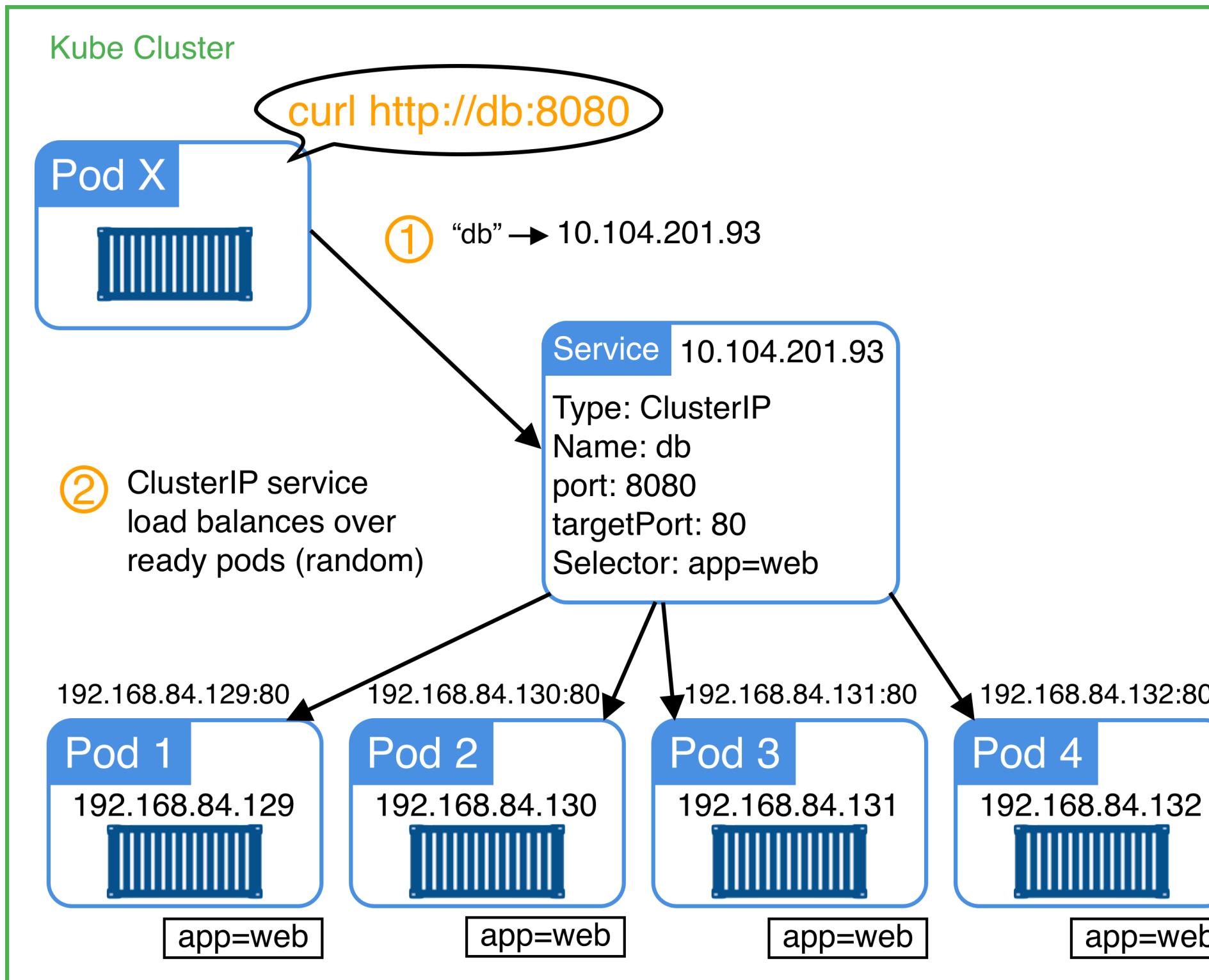
- // Pods are mortal
- // Pods are never resurrected
- // Pod IP Addr cannot be relied on

Solution:

- // Service defines:
 - Logical set of Pods
 - Policy how to access them



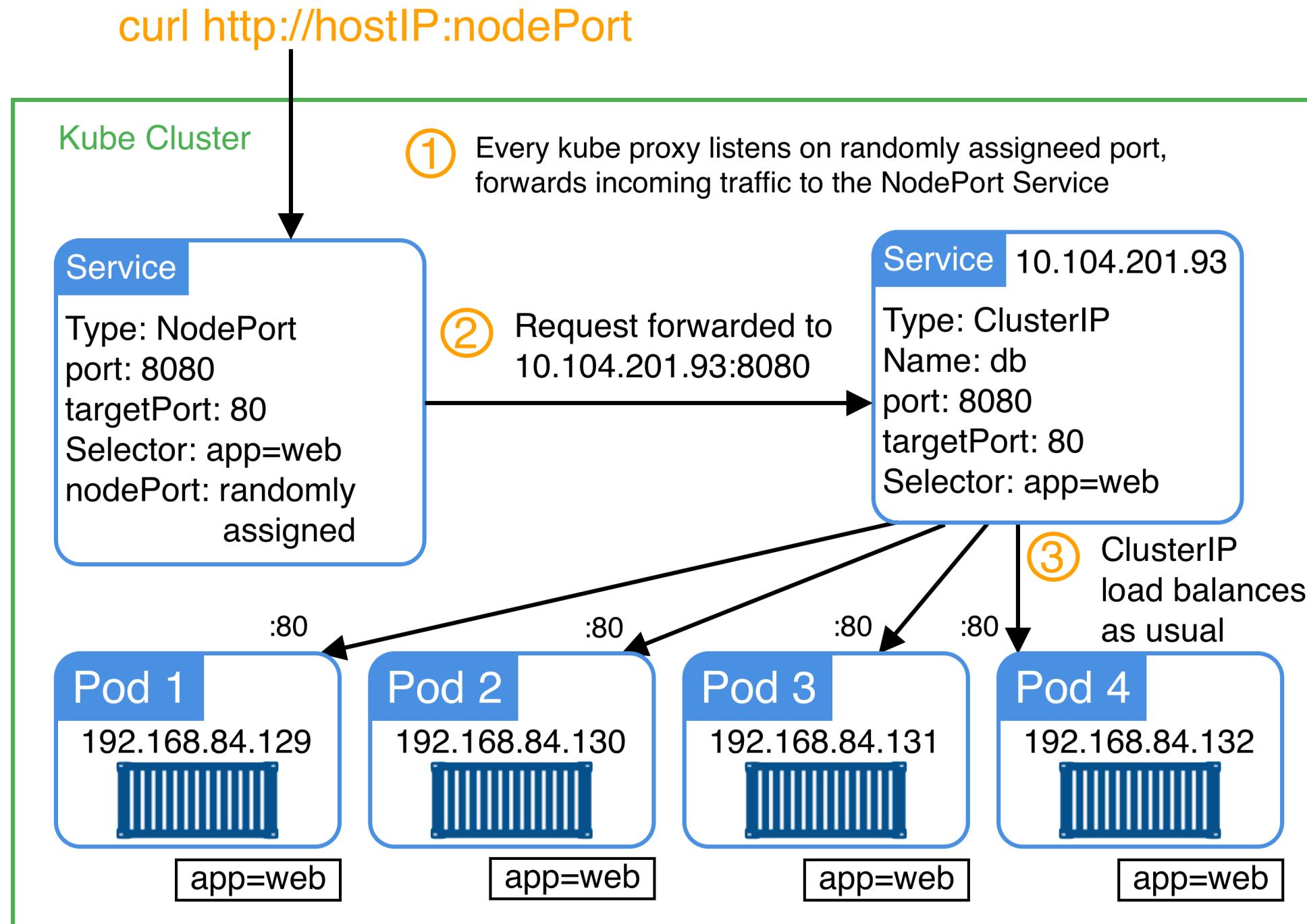
CLUSTERIP SERVICES



// 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

```
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
```

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

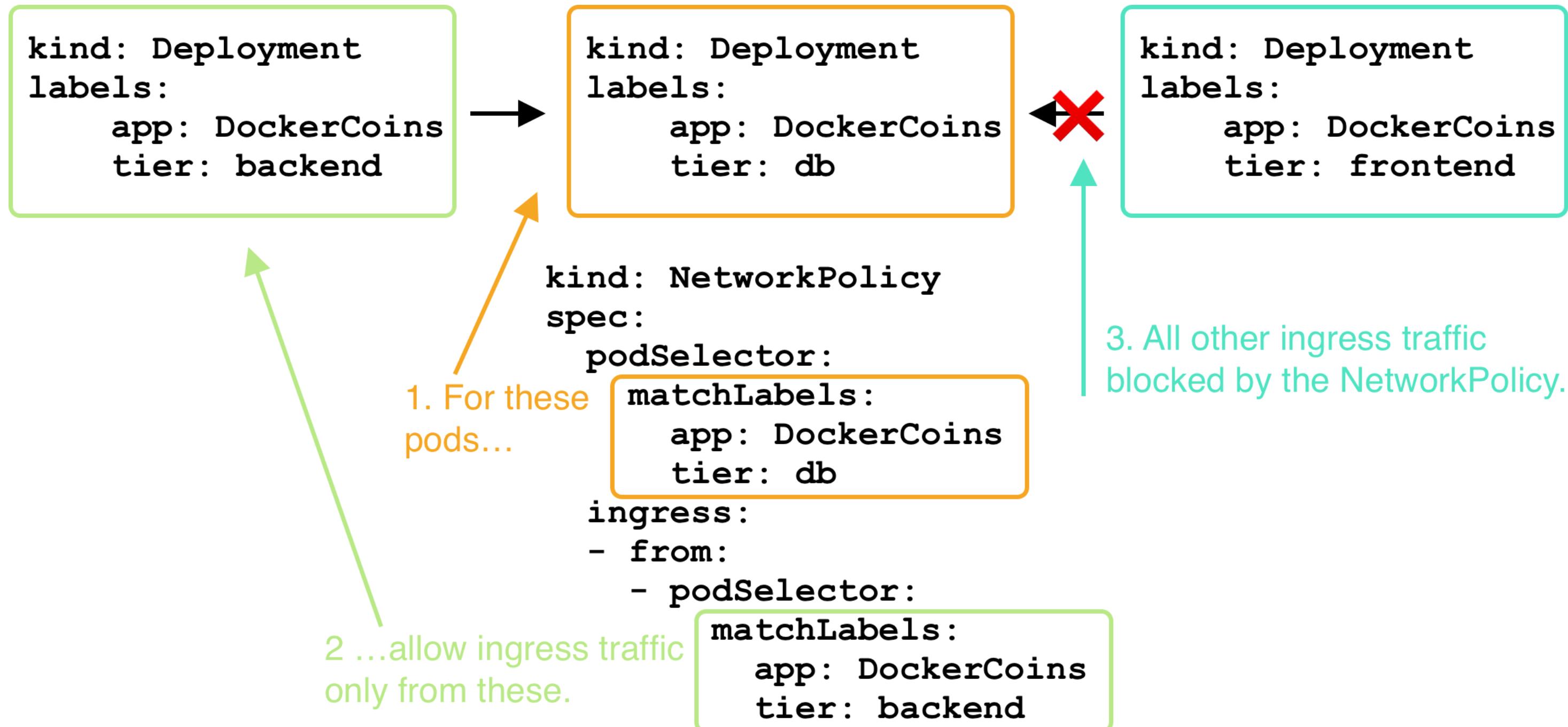


NETWORK POLICIES

```
// 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



KUBERNETES NETWORKING PLANES

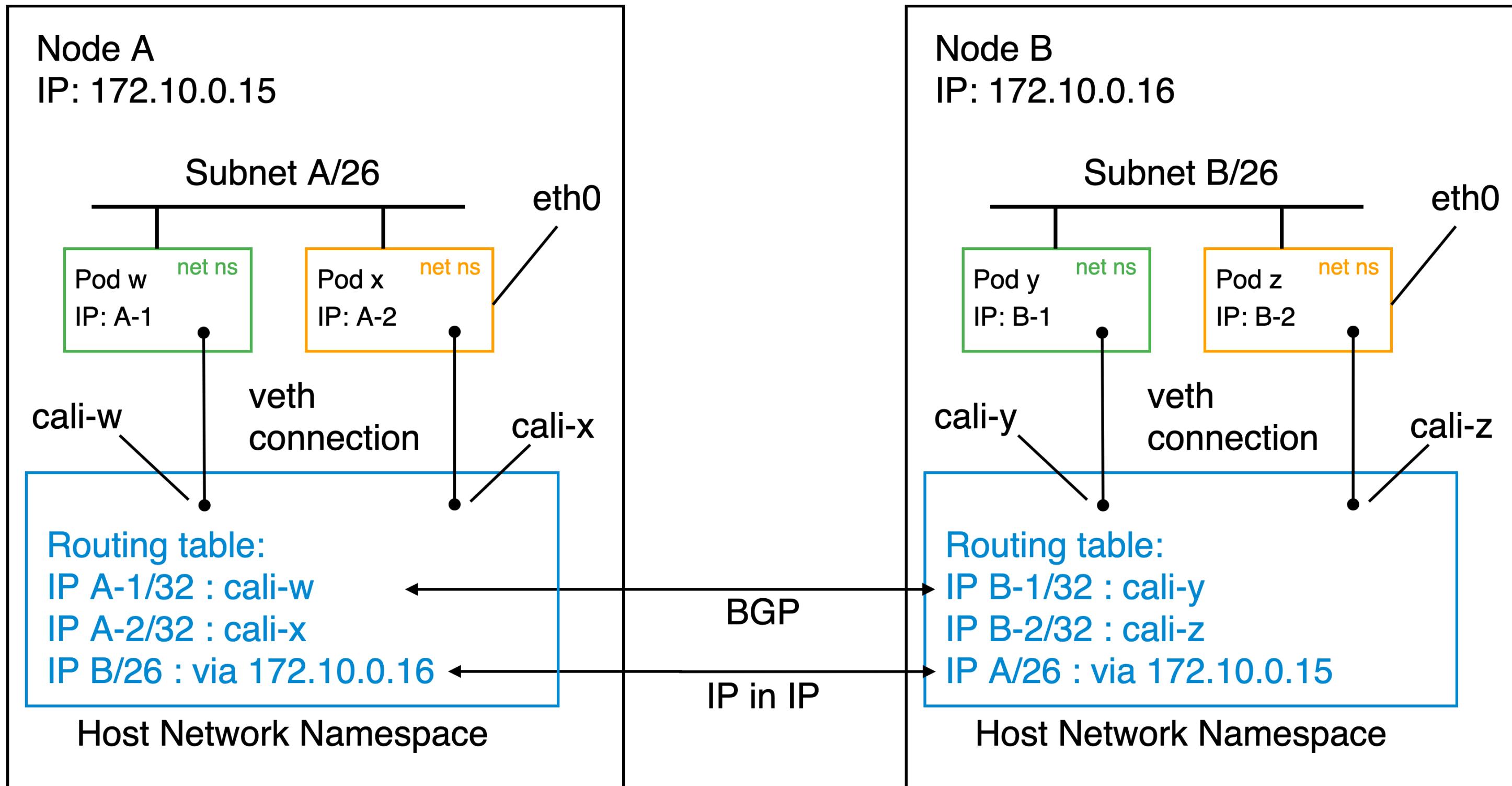
// Management:

- 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





EXERCISE: KUBERNETES NETWORKING

Work through

// Kubernetes Networking

In the Exercises book.

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/2kdI1qQ>
- // Kubernetes the Hard Way: <http://bit.ly/29Dq4wC>

KUBERNETES INGRESS

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/2kdI1qQ>
- // Kubernetes the Hard Way: <http://bit.ly/29Dq4wC>

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

WRAP UP KUBERNETES

A

EXERCISE INSTRUCTIONS

// A

// B

// C

// D

// E

// **Hint:** use ABCDEF

SOLUTION

YAML RESSOURCE