Core Concepts

Containerization & Orchestration

Kubernetes Architecture & API. Basic Objects and Tools

SoftUni Team

Technical Trainers

Software University

https://softuni.bg

Have a Question?

sli.do

#Kubernetes

https://www.facebook.com/groups/KubernetesOctober2023

Table of Contents

1. Containerization and Orchestration
2. Kubernetes Architecture and API
3. Basic Tools
4. Basic Objects

Containerization

Containerization

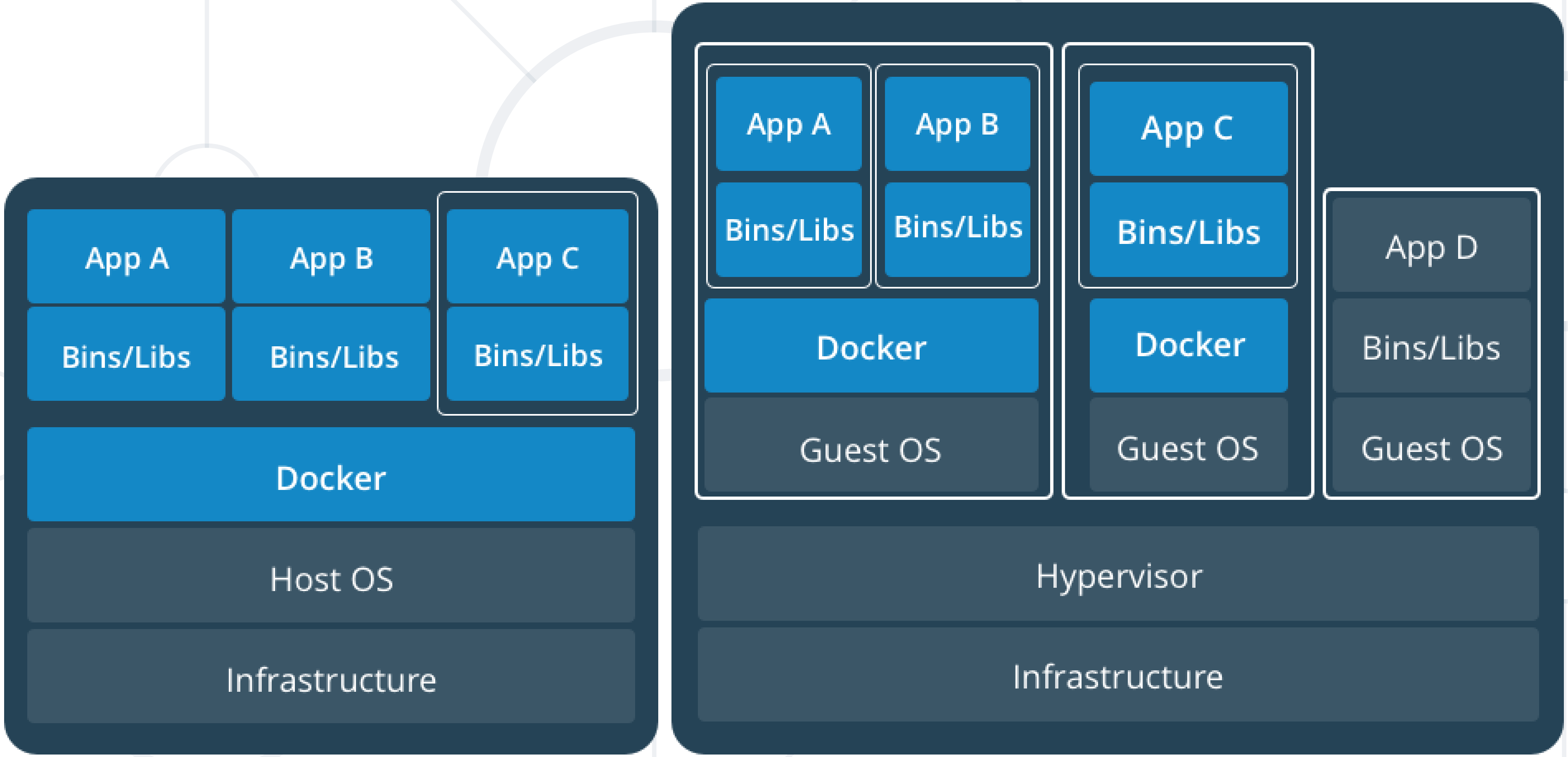
> “OS-level virtualization refers to an operating system paradigm in which the kernel allows the existence of **multiple isolated user space instances** known as **containers, zones, jails, …**”

https://en.wikipedia.org/wiki/OS-level_virtualization

Virtual Machines vs Containers

- **Virtual Machines**
  - Virtualize the hardware
  - Complete isolation
  - Complete OS installation
  - Require more resources
  - Run almost any OS
- **Containers**
  - Virtualize the OS
  - Lightweight isolation
  - Shared kernel
  - Require fewer resources
  - Run on the same OS

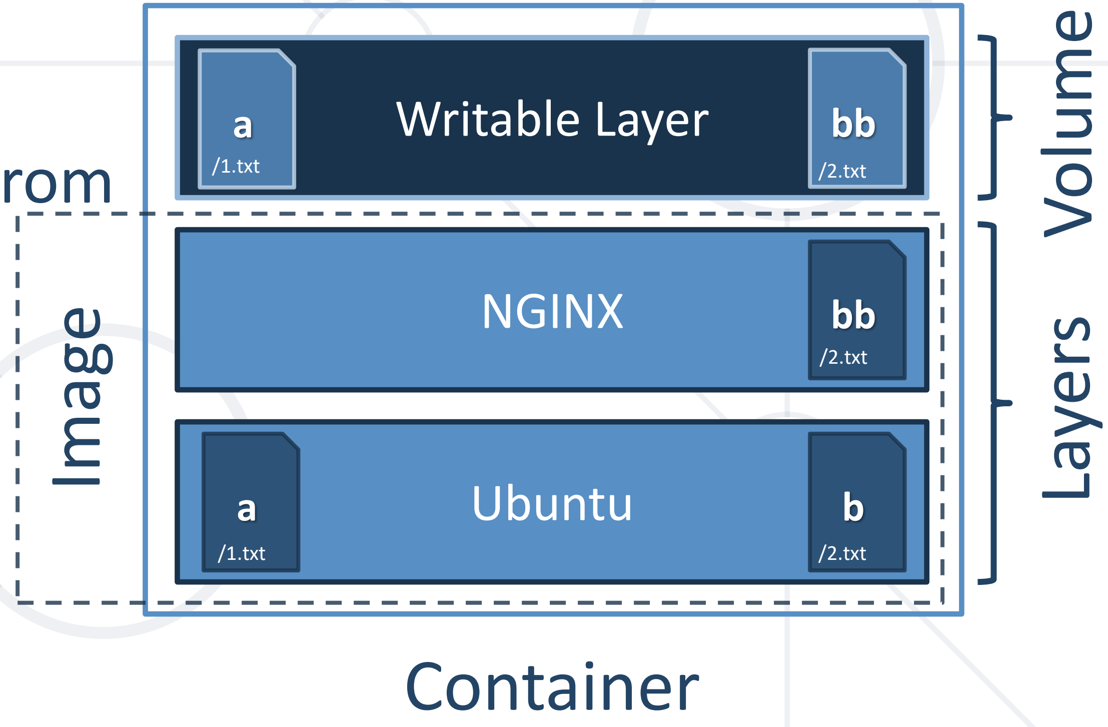
Virtual Machines and Containers



https://www.docker.com/what-container

Definitions

- **Container**
  - A runnable instance of an image. Containers are processes with much more isolation
- **Image**
  - A read-only template of a container built from layers. Images provide a way for simpler software distribution
- **Repository**
  - A collection of different versions of an image identified by tags
- **Registry**
  - A collection of repositories



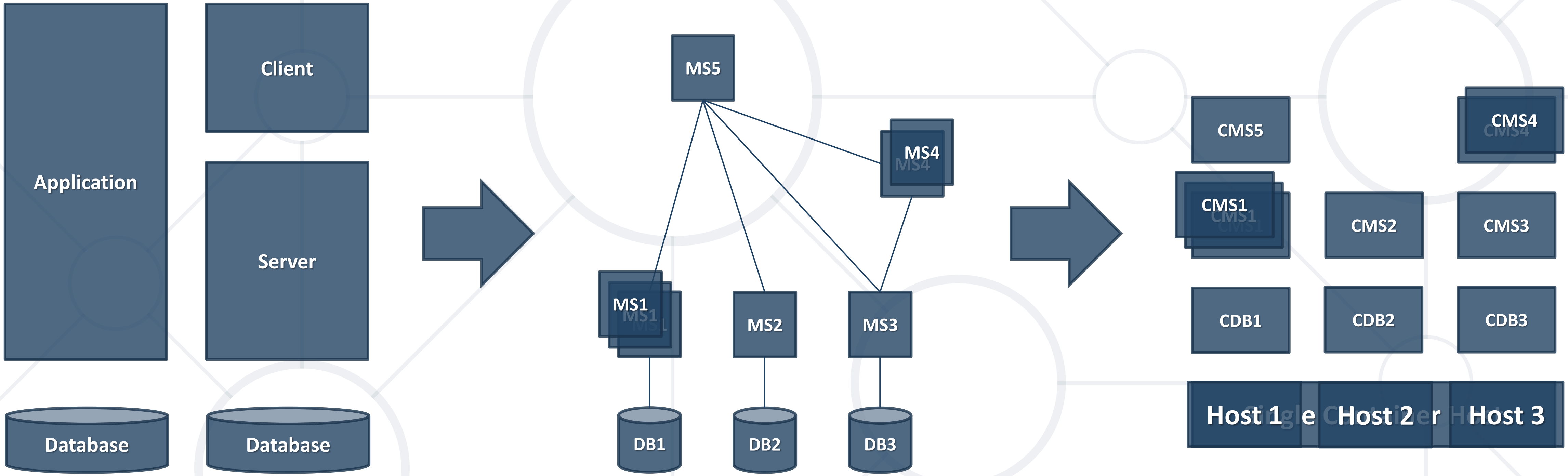
Orchestration

Application Evolution *

Monolithic Applications

Microservices

Containers



Application

Client

Server

Database

Database

MS5

MS4

MS1

MS2

MS3

DB1

DB2

DB3

CMS5

CMS4

CMS1

CMS2

CMS3

CDB1

CDB2

CDB3

Host 1

Host 2

Host 3

Microservices != Containers

New Demands*

- Workload deployment and distribution
- Resource governance
- Scalability and availability
- Automatization and management
- Internal and external communication

}

Container Orchestration

* Not an exhaustive list

Kubernetes Got You Covered*

- Runs a cluster of hosts
- Schedules containers to run on different hosts
- Facilitates the communication between the containers
- Provides and controls access to/from outside world
- Tracks and optimizes the resource usage

* Not an exhaustive list

Kubernetes Origin

- Born out of projects like **Borg** and **Omega** at **Google**
- Written in **Go**
- Donated to **CNCF** in **2014**
- Open source, licensed under **Apache 2.0**
- **Version 1.0** came into existence in **July 2015**. Current is **1.28.2**
- **κυβερνήτης** in Greek means **Helmsman** – s.o. who steers the ship
- Can be seen often as **k8s** (K$\overbrace{\cancel{\text{ubernete}}}^{8}$s)

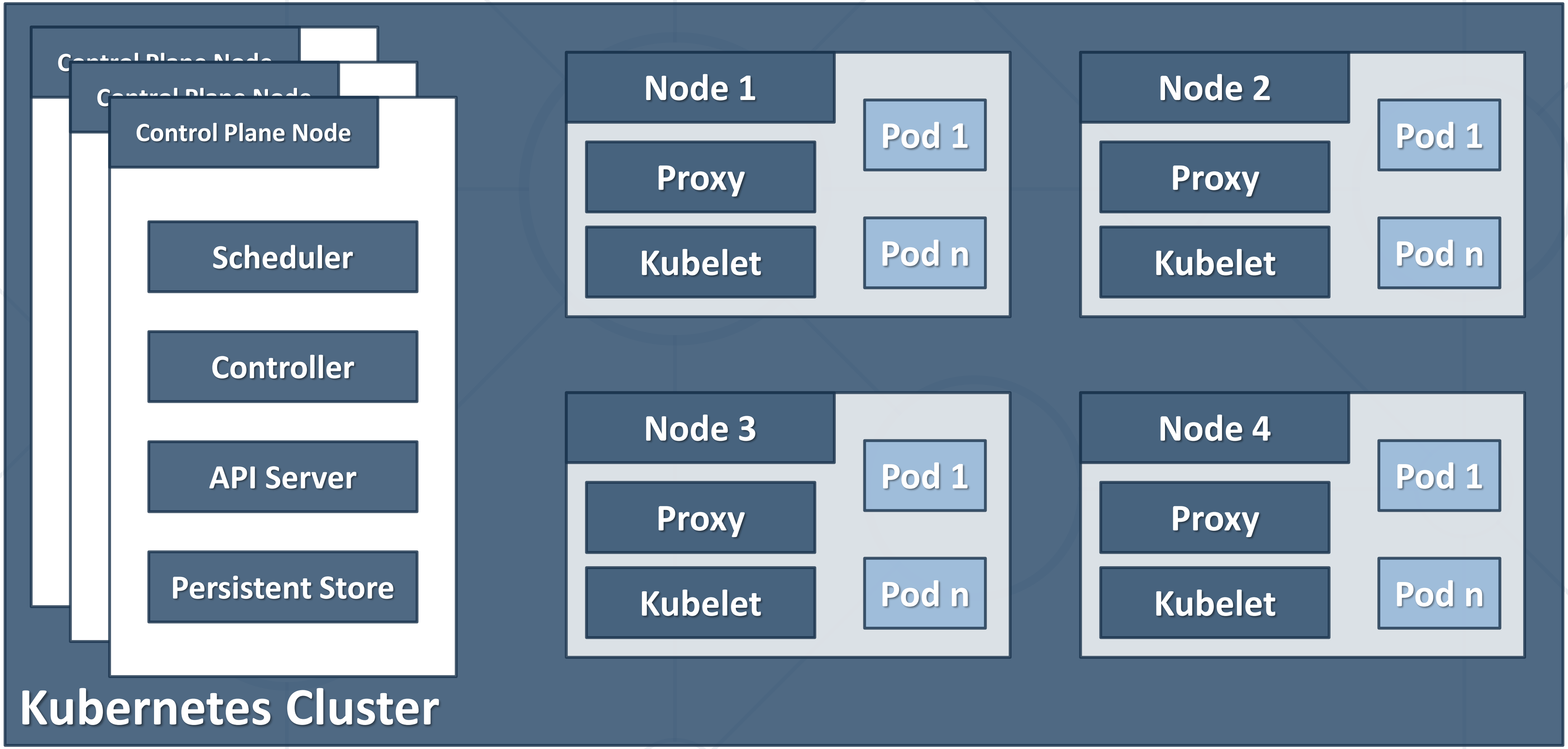
Other Solutions*

- Docker Swarm
- HashiCorp Nomad
- Apache Mesos + Marathon

* Not an exhaustive list

Kubernetes Architecture

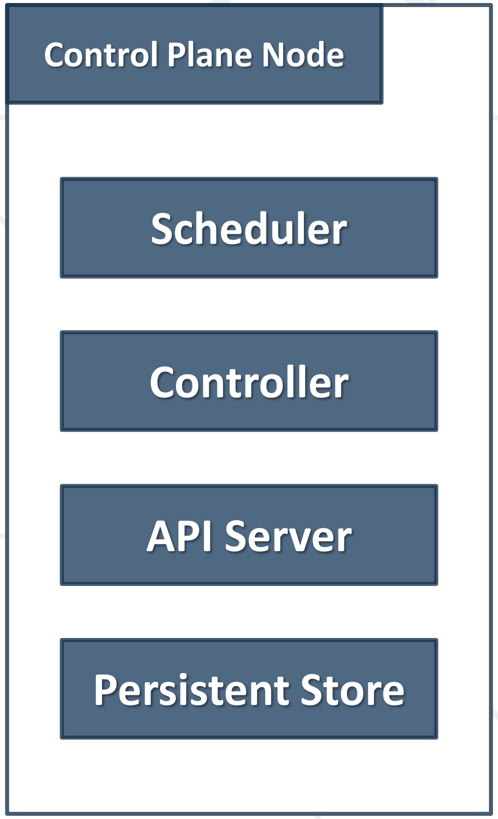
Architecture Overview *



* Not all components are shown

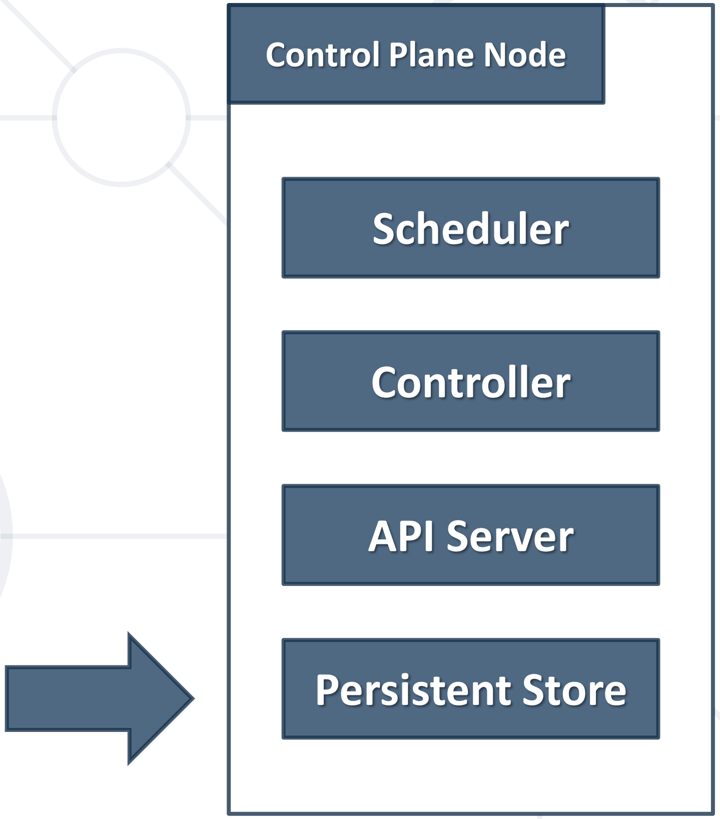
Control Plane (master) Nodes

- Responsible for **managing** the cluster
- Typically, **more than one** is installed
- In HA mode one node is the **Leader**
- It is **work-free** (this can be changed)
- Components running on master are also known as **Control Plane**
- Can be reached via **CLI** (kubectl), **APIs**, or **Dashboard**



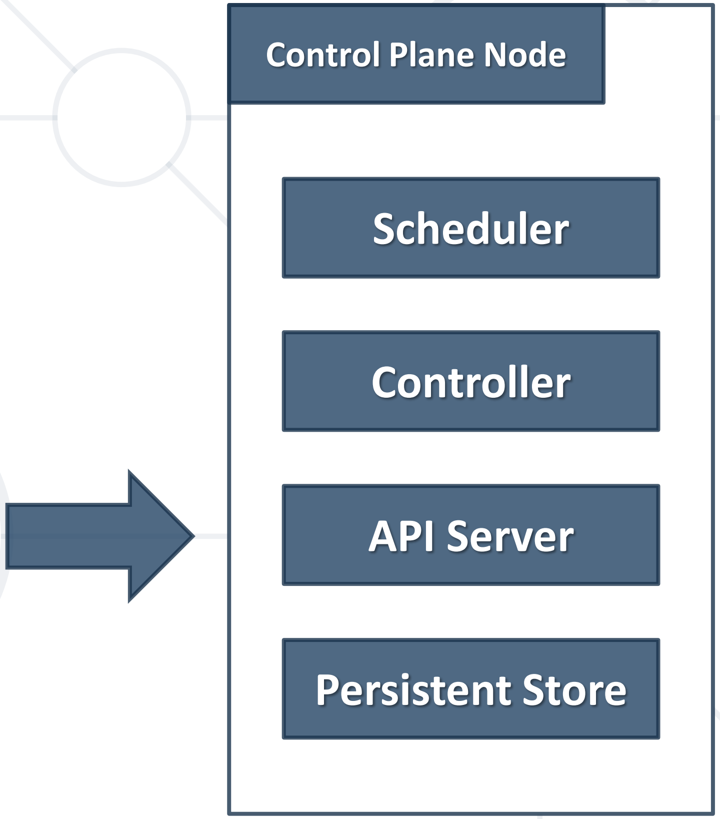
Control Plane Nodes: Persistent Store

- Based on **etcd**
- **Persistent** storage
- Cluster **state** and **configuration**
- **Distributed** and **consistent**
- Provides single **source of truth**
- Can be installed **externally**



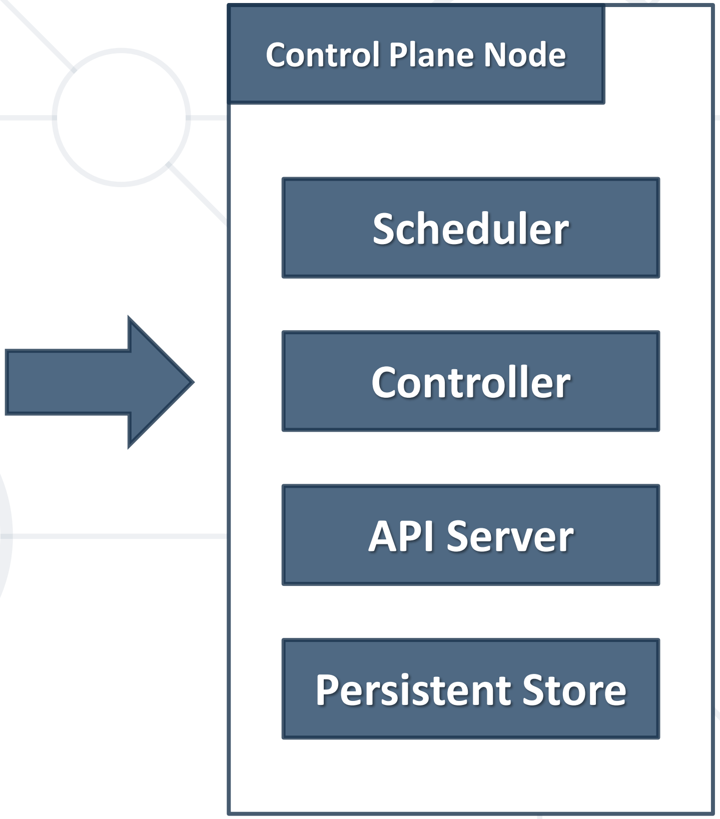
Control Plane Nodes: API Server

- Exposes the **Kubernetes API (REST)**
- **Front-end** for the control plane
- **Administrative** tasks
- Consumes **JSON** via **Manifest files (YAML)**



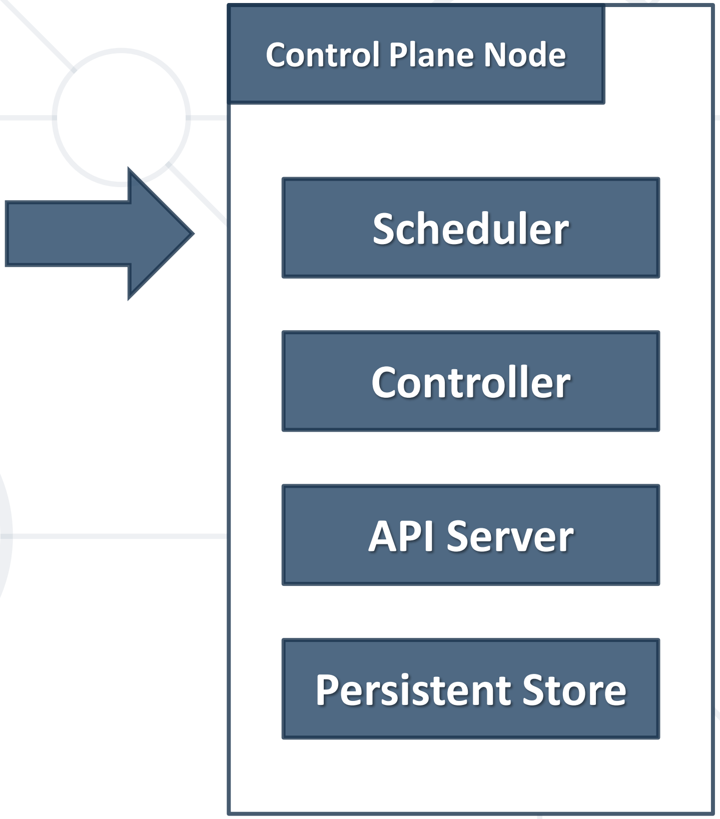
Control Plane Nodes: Controller

- Executes **control loops**
- Responsible for other controllers
  - Node controller
  - Endpoints controller
  - Namespace controller, etc.
- Watches for **changes**
- Maintains the **desired state**



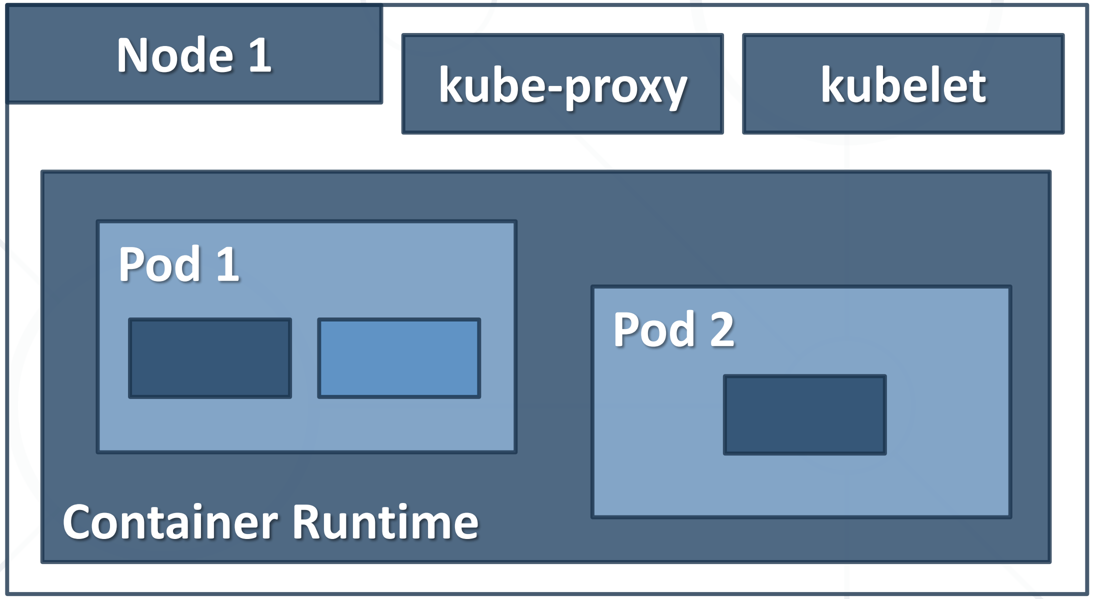
Control Plane Nodes: Scheduler

- **Listens** API Server for new work
- **Assigns work** to nodes



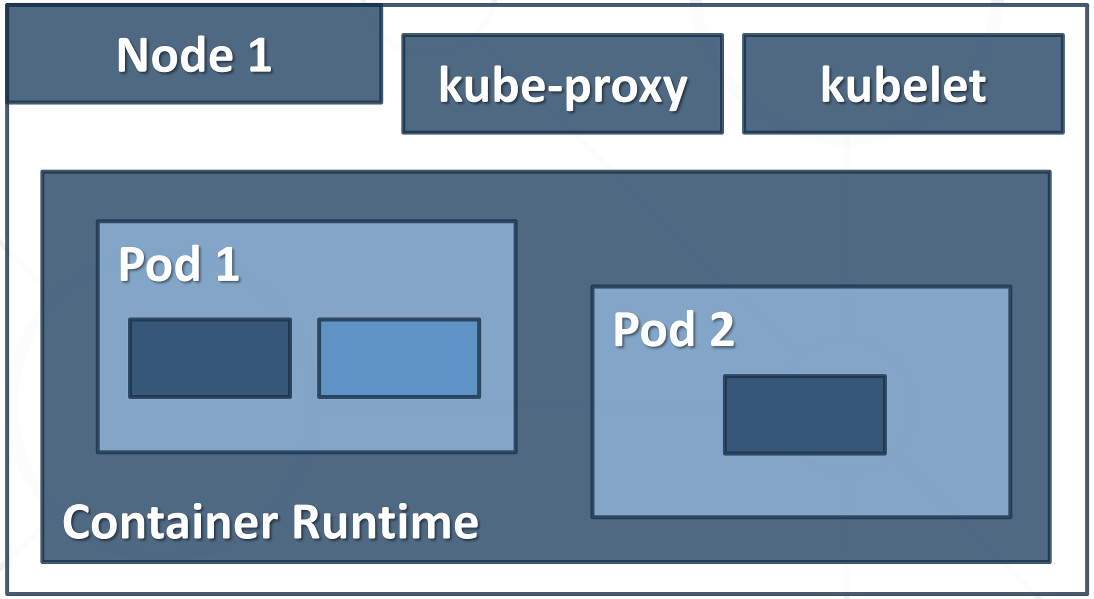
(Worker) Nodes

- **kubelet**
  - Communicates with the control plane
- **Container runtime**
  - containerd, CRI-O, etc.
- **kube-proxy**
  - Network proxy



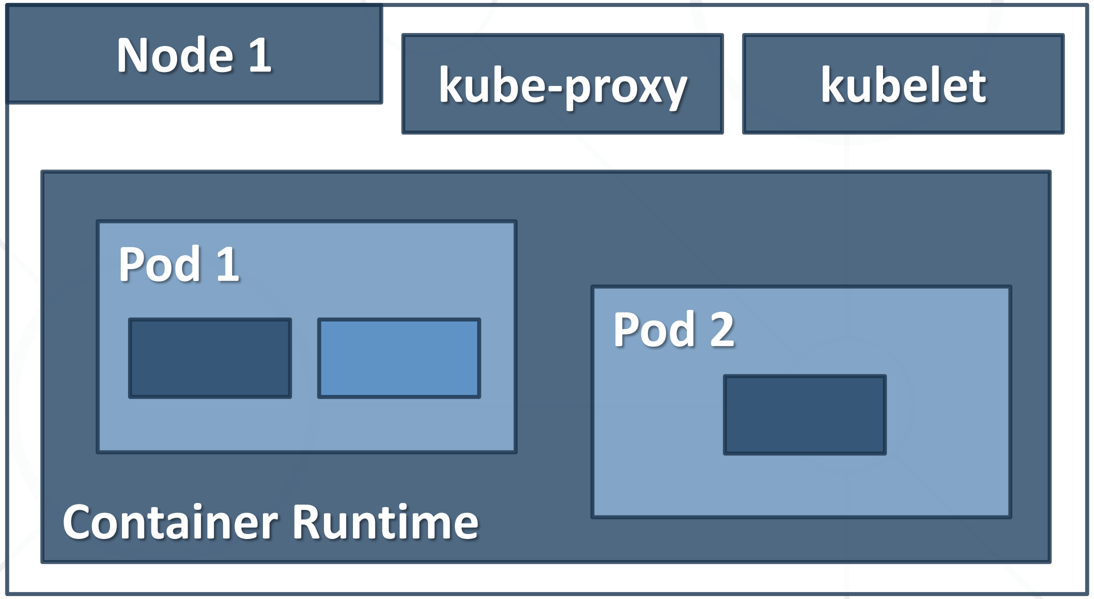
(Worker) Nodes: kubelet

- Main Kubernetes agent
- Registers node in the cluster
- Listens to the API Server
- Creates pods
- Reports back to the control plane
- Exposes endpoint on :10255
    - /spec
    - /healthz
    - /pods



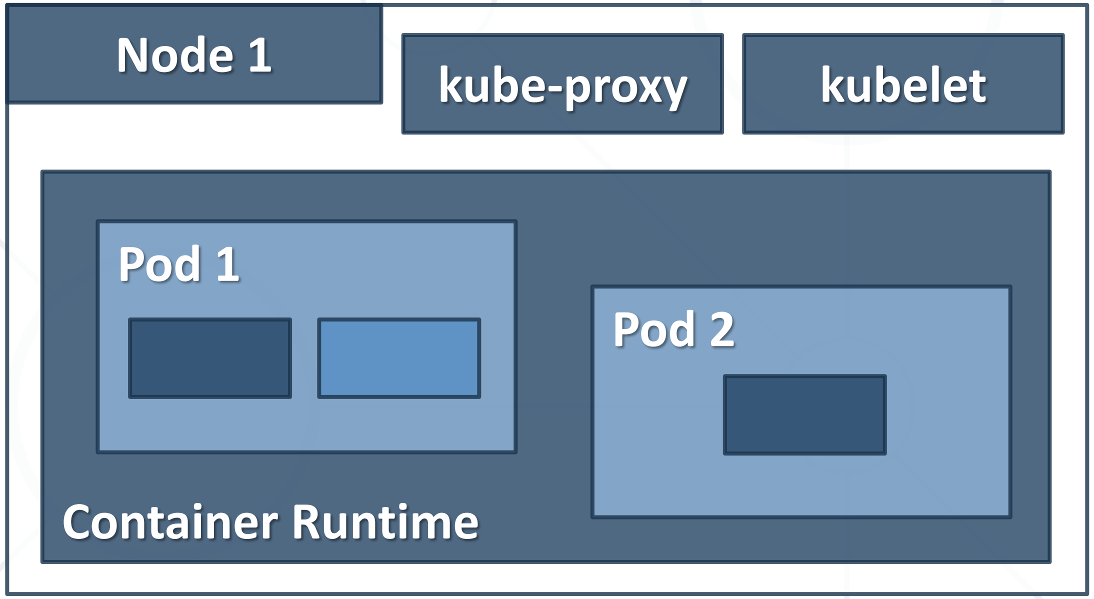
(Worker) Nodes: Container Runtime

- Container management
  - **Pulling** images
  - **Starting** and **stopping**
- It is **pluggable**



(Worker) Nodes: kube-proxy

- Provides the **networking**
- Each pod has its **own address**
- All containers in a pod share the **same IP** address
- Offers **load balancing** across all pods in a **service**



Work with Kubernetes

Kubernetes Distributions

- A software package that provides a pre-built version of Kubernetes
- Most distributions also offer installation tools or additional software integrations
- On-premise
  - **KinD, Minikube, MicroK8s, K3s, k0s, OpenShift, VMware Tanzu …**
- Cloud-based
  - **Azure Kubernetes Services (AKS), Elastic Container Service for Kubernetes (EKS), Google Kubernetes Engine (GKE), …**
- Usually, cloud versions are a few versions behind

Installation Scenarios and Tools

- Installation methods
  - **Localhost** (for test and development)
  - **On-Premise** (VMs, Bare Metal)
  - **Cloud** (Hosted Solutions, Turnkey Solutions, Bare Metal)
- Configurations
  - **All-in-One Single Node** and different **Multi Node** options
- Installation tools
  - Test/development - **KinD**, **Minikube**, etc.
  - Production - **kubeadm**, **KubeSpray**, **Kops**, etc.

kind

- Easiest way to test and start with Kubernetes
- **kind** stands for Kubernetes in Docker
- So, it requires **Docker** to be installed and configured

https://kind.sigs.k8s.io/docs/

minikube

- Easiest and recommended way for a **local all-in-one cluster**
- Requirements
  - **kubectl**
  - **Hypervisor** (VirtualBox, Hyper-V, KVM, xhyve, VMware Fusion)
  - **VT-x/AMD-v** enabled
  - Internet connection on first run
- Supports **Linux**, **macOS**, and **Windows**
- Provides **docker-machine**-like experience, but for **Kubernetes**

https://minikube.sigs.k8s.io/docs/

kubectl (1)

- Controls Kubernetes clusters
- Expects a file named **config** in the **$HOME/.kube** directory
- Other files can be specified by setting the **KUBECONFIG** environment variable or by setting the **--kubeconfig** flag
- The syntax is

```
kubectl [command] [TYPE] [NAME] [flags]
```

https://kubernetes.io/docs/reference/kubectl/overview/

kubectl (2)

- Where **command** is the operation (**run**, **get**, etc.) and **type** is the resource (**pod**, **service**, etc.). Note that **name** is case-sensitive
- Its version should +/- 1 minor version compared to the cluster. For example, with kubectl version 1.22 we can work with clusters version 1.21, 1.22, and 1.23

https://kubernetes.io/docs/reference/kubectl/overview/

Dashboard

- A web-based Kubernetes user interface
- Deployment of containerized applications to a cluster
- Troubleshooting containerized application
- Managing the cluster resources

https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/

Practice

Live Exercise in Class (Lab)

Basic Kubernetes Objects 101

Namespaces. Pods. Services

Objects Overview

- Kubernetes objects are persistent entities
- They are used to represent the state of the cluster
- An object is a "**record of intent**". Once created, the Kubernetes system will constantly work to ensure that object exists
- Almost every object includes two nested object fields
  - **Spec** provides a description of the characteristics (**desired state**)
  - **Status** describes the **current state** of the object
- They include **Pods**, **Services**, **Namespaces**, **Volumes**, etc.

https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/

Objects Management

- **Imperative commands**
  - Commands are invoked against live objects. We directly state what should be done. Good for development or test and for one-off tasks
- **Imperative object configuration**
  - Operations are specified together with at least one file, which contains the definition of target object(s). Can be used in production
- **Declarative object configuration**
  - Operates with local configuration files but the actions are not stated explicitly. Can work with files and folders

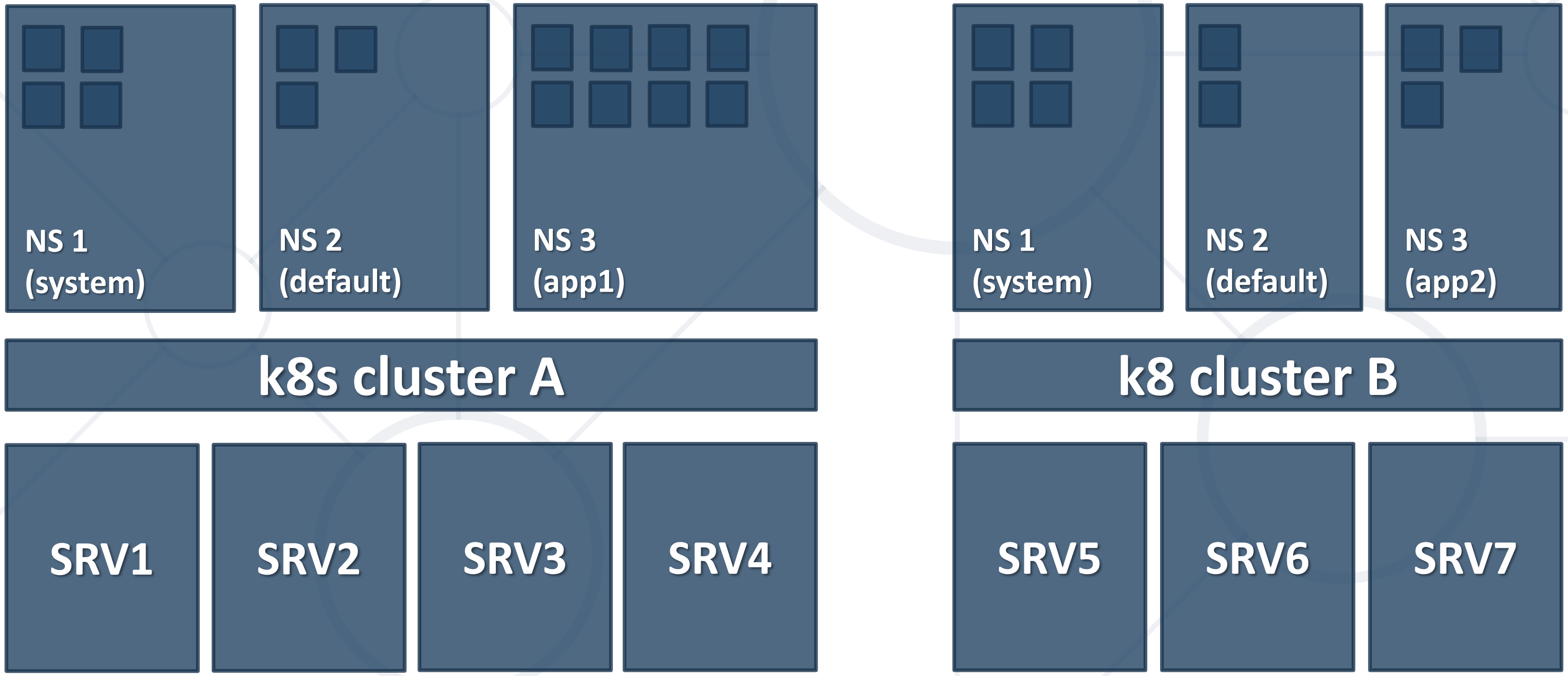
https://kubernetes.io/docs/concepts/overview/working-with-objects/object-management/

Namespaces

- Kubernetes supports multiple virtual clusters
- These virtual clusters are called **namespaces**
- Namespaces provide a **scope for names**
- Names of resources need to be **unique** within a namespace
- Namespaces **cannot be nested** inside one another
- Each Kubernetes resource can **only be in one** namespace
- Most Kubernetes resources are in some namespace
- Namespace resources are not themselves (and others such as **nodes**) in a namespace
- Deleting a Namespace will clean up everything under it

https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

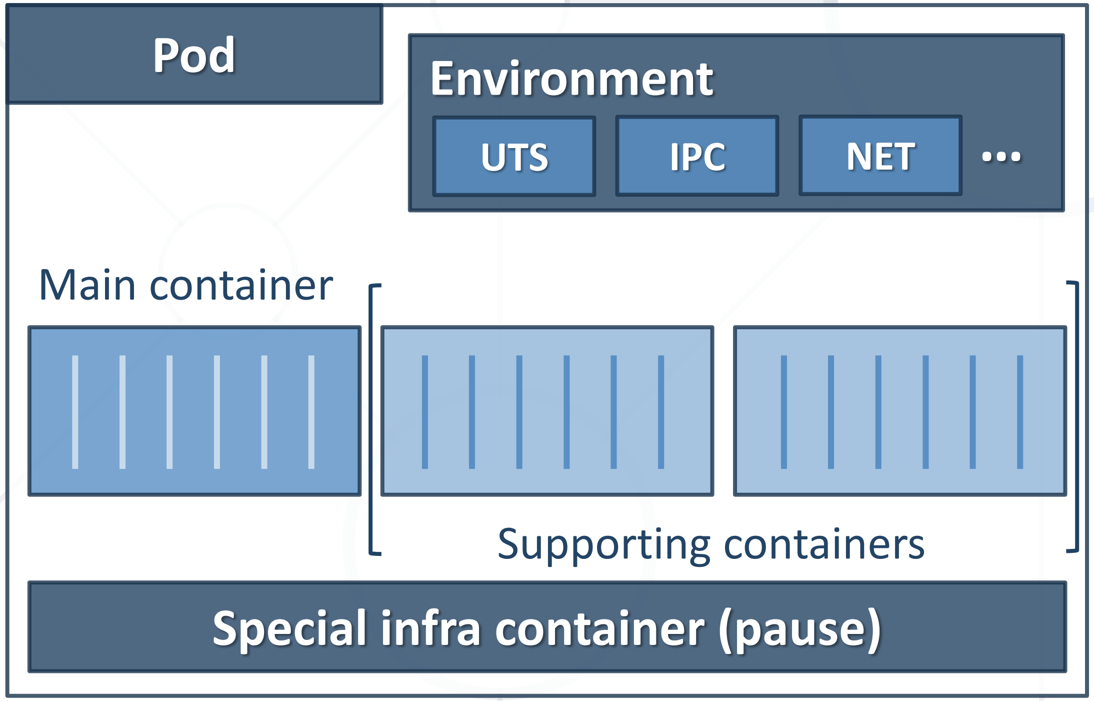
Namespaces vs Clusters vs Data Centers



Namespaces divide a k8s cluster to virtual clusters

k8s cluster abstracts the datacenter

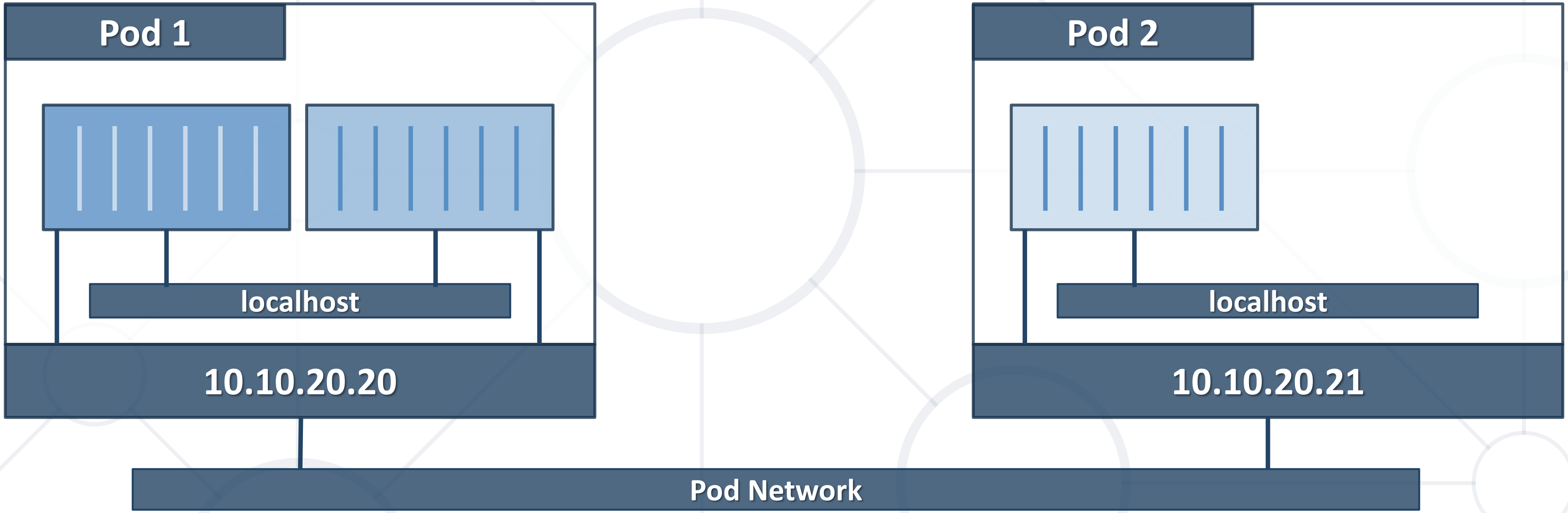
Pods (1)



- Smallest **unit of scheduling**
- **Scheduled** on nodes
- **One** or **more** containers
- Containers **share** the pod **environment**
- **Deployed as one** and on **one node.** It is **atomic**
- Created via **manifest files**

https://kubernetes.io/docs/concepts/workloads/pods/

Pods (2)



- Each pod has a **unique IP** address
- **Inter-pod** communication is via a **pod network**
- **Intra-pod** communication is via **localhost** and **port**

Pod Manifest *

```
apiVersion: v1
kind: Pod
metadata:
  name: appa-pod
spec:
  containers:
  - name: appa-container
    image: shekeriev/k8s-appa:v1
    ports:
    - containerPort: 80
```

* Working but very simple one

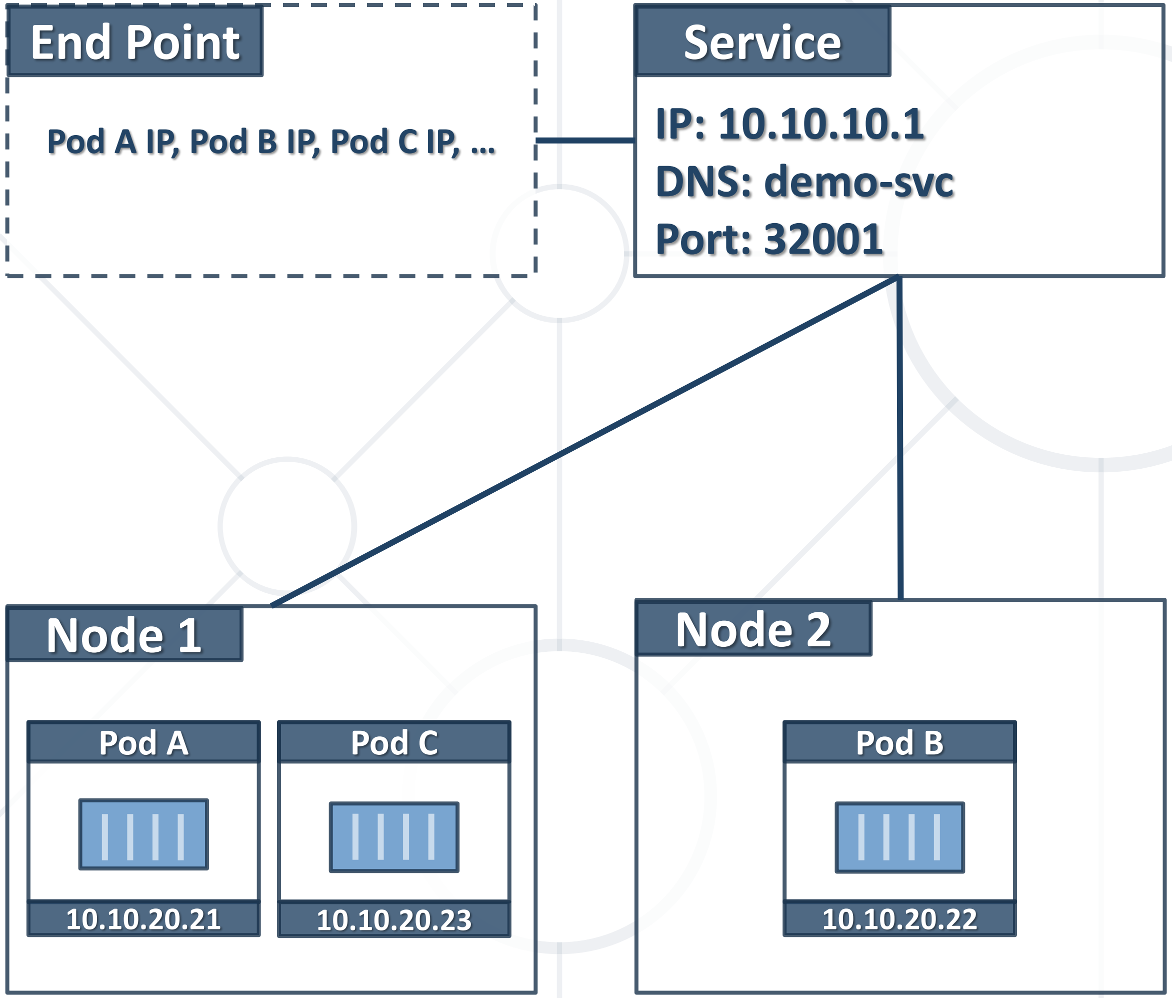
Labels and Annotations

- **Labels**
  - Key-value pairs attached to objects
  - Each object may have multiple labels
  - Each label may be attached to multiple objects
  - Used to identify and group sets of objects
  - Used with **label selectors** to select a group of objects

Apply to **annotations** as well

- **Annotations**
  - Key-value pairs attached to objects
  - Used to store additional information (metadata) like description, creator, etc.

Services



- Provide reliable network endpoint
  - IP address
  - DNS name
  - Port
- Expose pods to the outside world
- Use **end point** object to track pods
- Use **label selectors** to do their magic

https://kubernetes.io/docs/concepts/services-networking/service/

Service Types

- **ClusterIP** exposes the Service on a **cluster-internal IP**
  - This way the Service will be only reachable from within the cluster
  - **This is the default**
- **NodePort** exposes the Service on each Node's IP at a static port specified by the NodePort
  - A ClusterIP Service, to which the NodePort Service routes, is automatically created
  - We can contact the NodePort Service, from outside the cluster, by requesting ***<NodeIP>:<NodePort>***
  - Default range is between **30000** and **32767**

Service Types

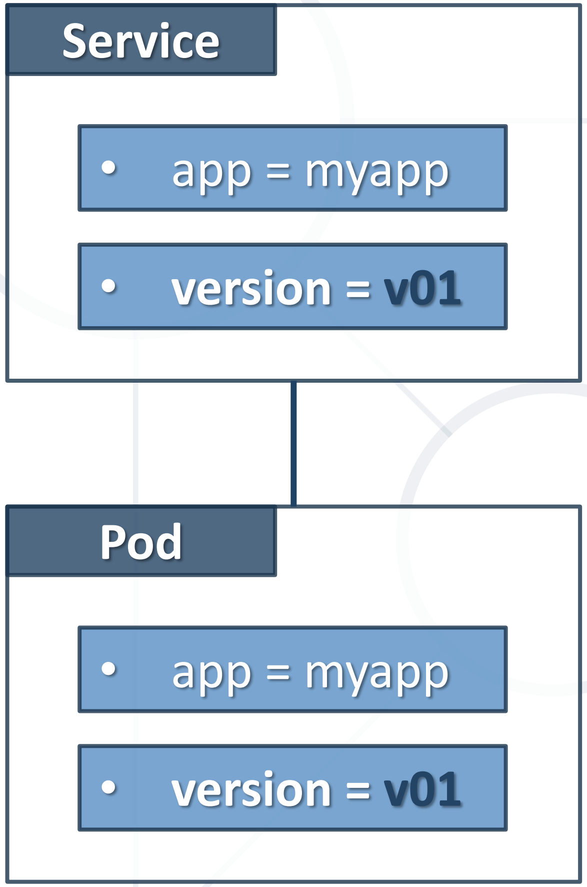
- **LoadBalancer** exposes the Service externally using a cloud provider's load balancer
  - NodePort and ClusterIP Services, to which the external load balancer routes, are automatically created
- **ExternalName** maps the Service to the contents of the **externalName** field (e.g. foo.bar.example.com), by returning a **CNAME** record with its value
  - No proxying of any kind is set up

Service Manifest *

```yaml
apiVersion: v1
kind: Service
metadata:
  name: appa-svc
  labels:
    app: appa
spec:
  type: NodePort
  ports:
  - port: 80
    nodePort: 30001
    protocol: TCP
  selector:
    app: appa
```

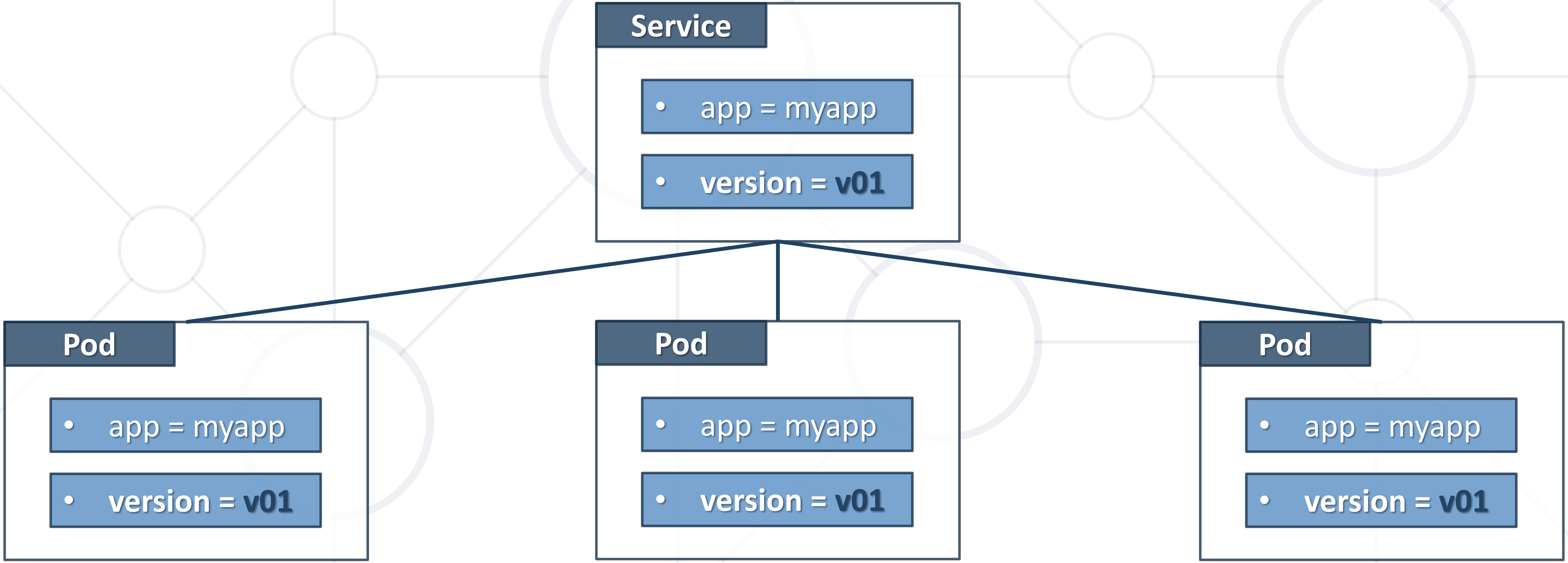
* Working but very simple one

Services in Action



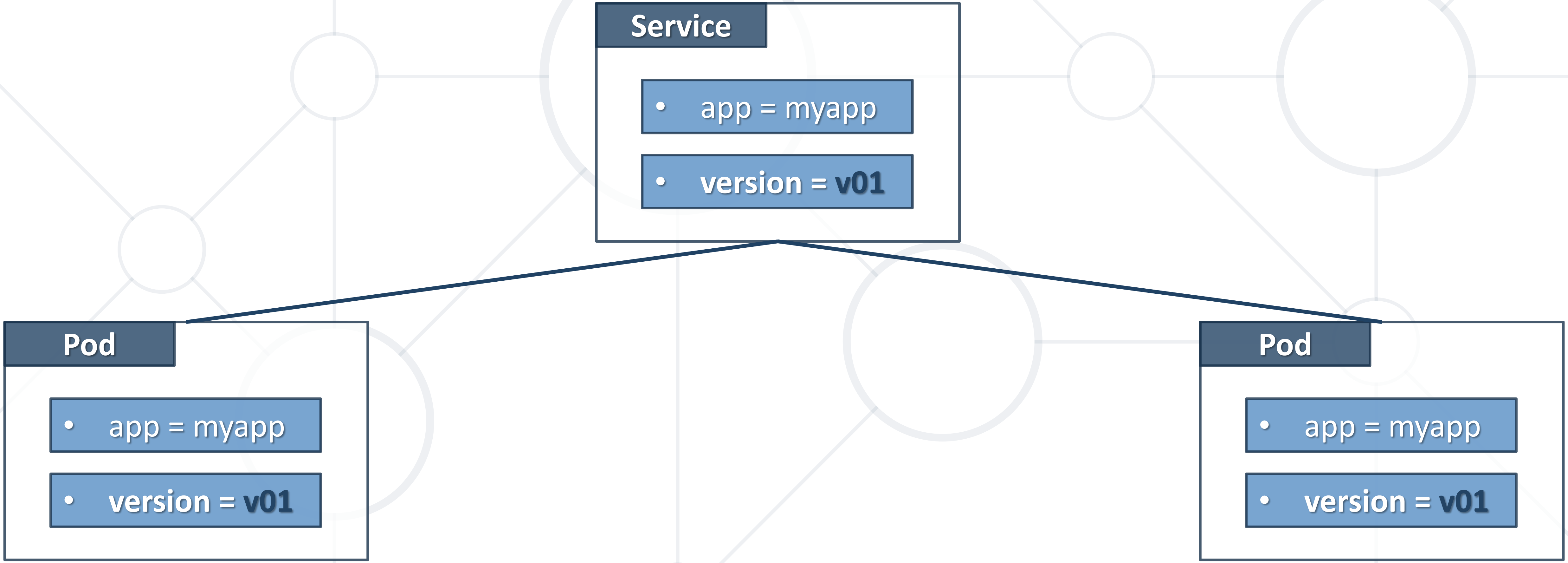
Initial deployment – one pod with version = v01

Services in Action (Scale Out)



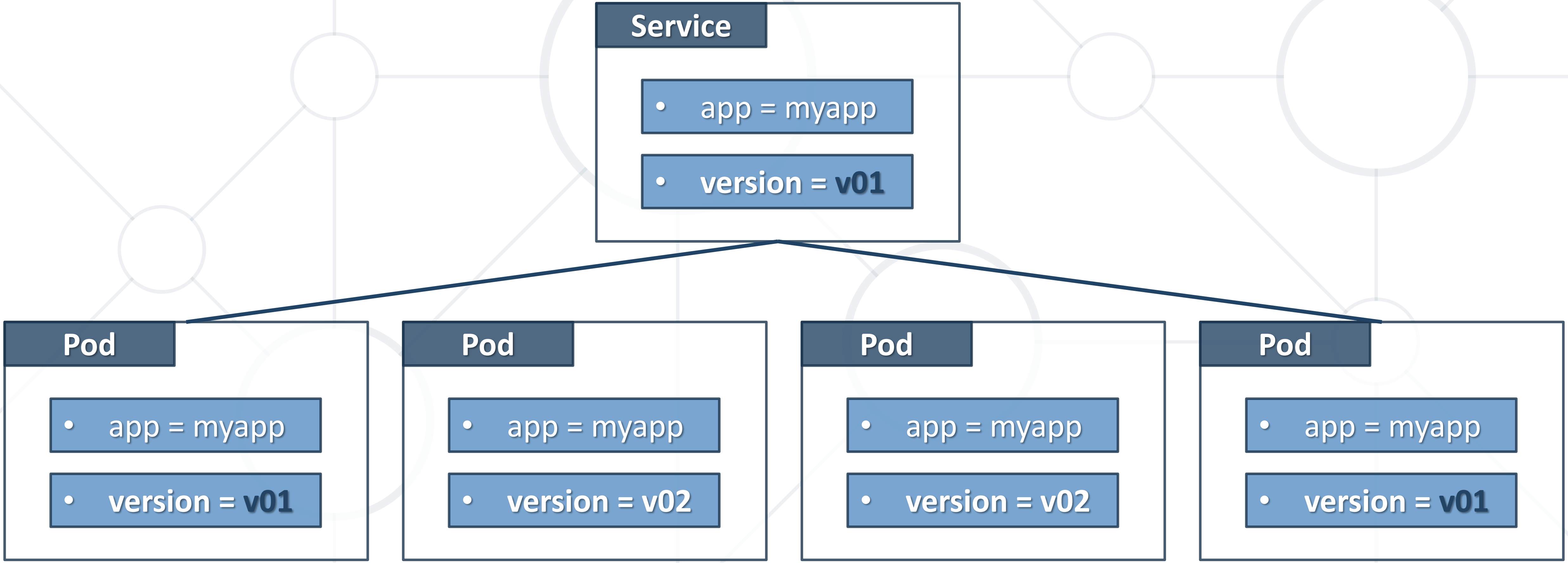
Scale out – two more pods with version = v01

Services in Action (Scale In)



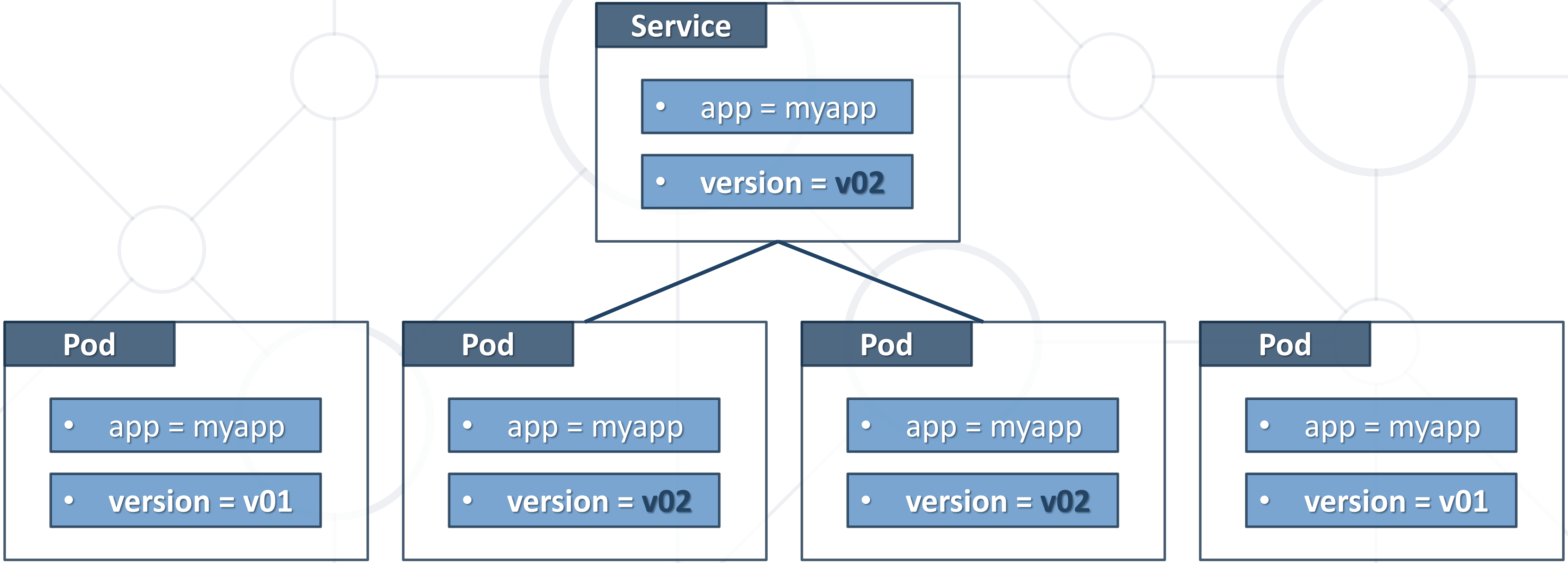
Scale in – remove one pod and end up with two pods with version = v01

Services in Action (App Update)



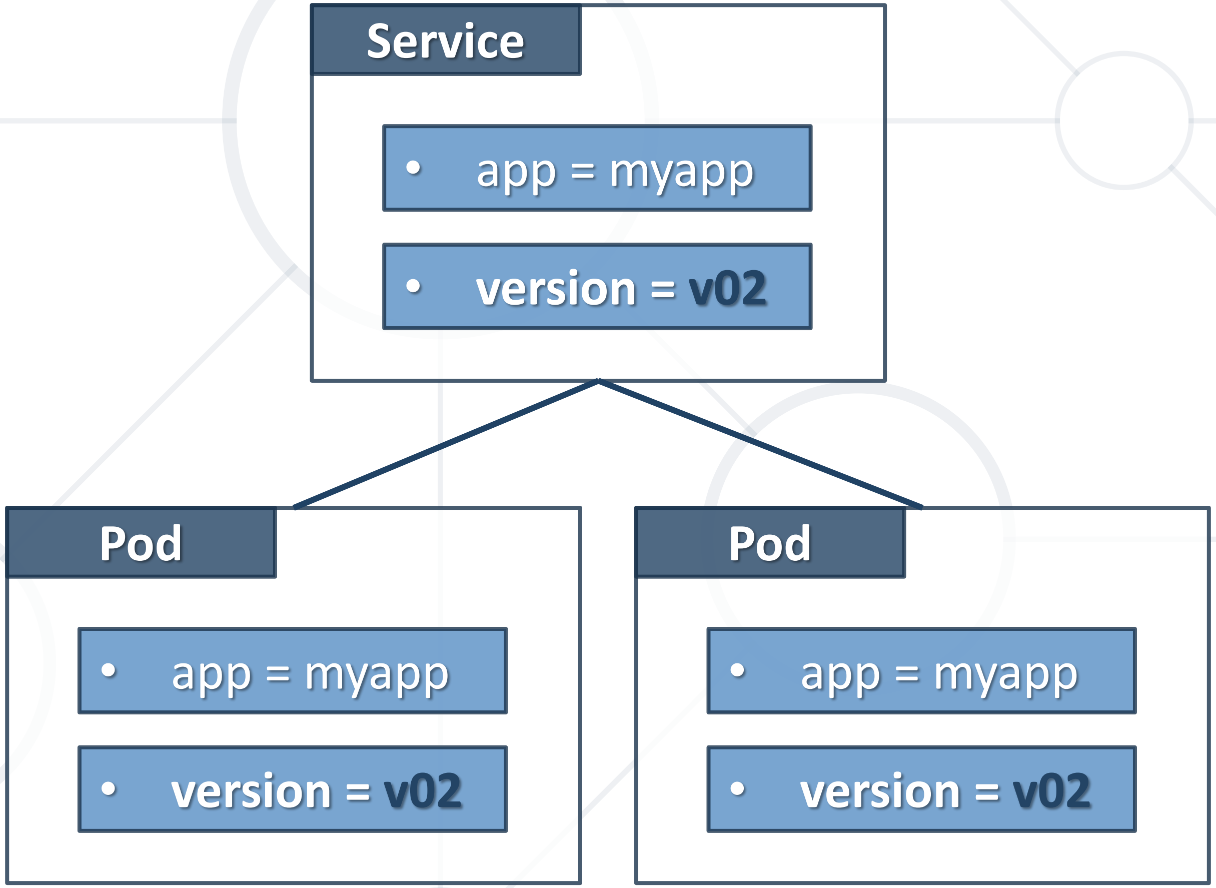
Next step – add two more pods with version = v02

Services in Action (App Update)



Next step – we update the service to look for version = v02

Services in Action (App Update)



Service

- app = myapp
- **version = v02**

Pod

- app = myapp
- **version = v02**

Pod

- app = myapp
- **version = v02**

Finally, all pods with version = v01 are destroyed

Practice

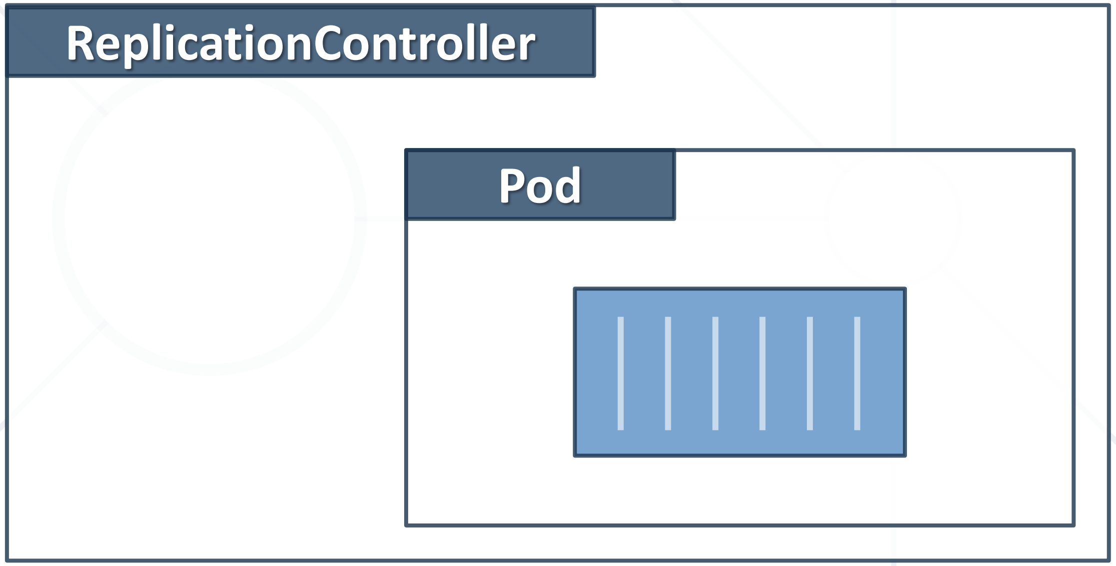
Live Exercise in Class (Lab)

Basic Kubernetes Objects 102

Replication Controllers. Replica Sets. Deployments

Replication Controllers

- **Higher** level workload
- Looks after **pod** or **set of pods**
- **Scale** out/in **pods**
- Sets **Desired State**
- Rarely used these days



https://kubernetes.io/docs/concepts/workloads/controllers/replicationcontroller/

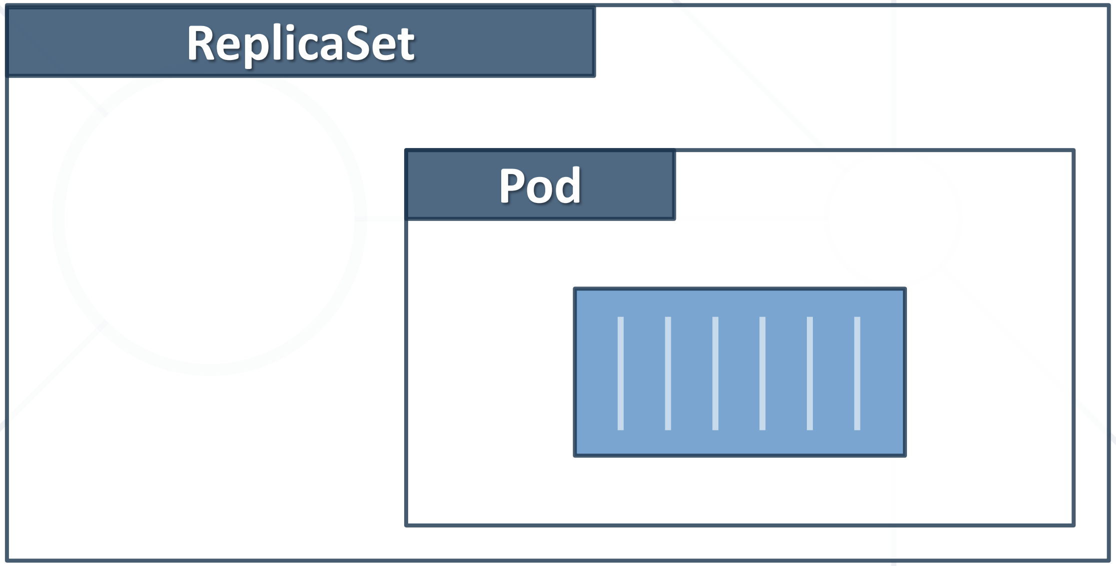
Replication Controller Manifest *

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: appa-rc
spec:
  replicas: 3
  selector:
    app: appa
  template:
    … [POD definition] …
```

* Partial one but with the important parts included (except the pod definition)

Replica Sets

- **Higher** level workload
- Looks after **pod** or **set of pods**
- **Scale** out/in **pods**
- Sets **Desired State**
- Preferred over ***Replication Controllers***
- Rarely used alone by itself



https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/

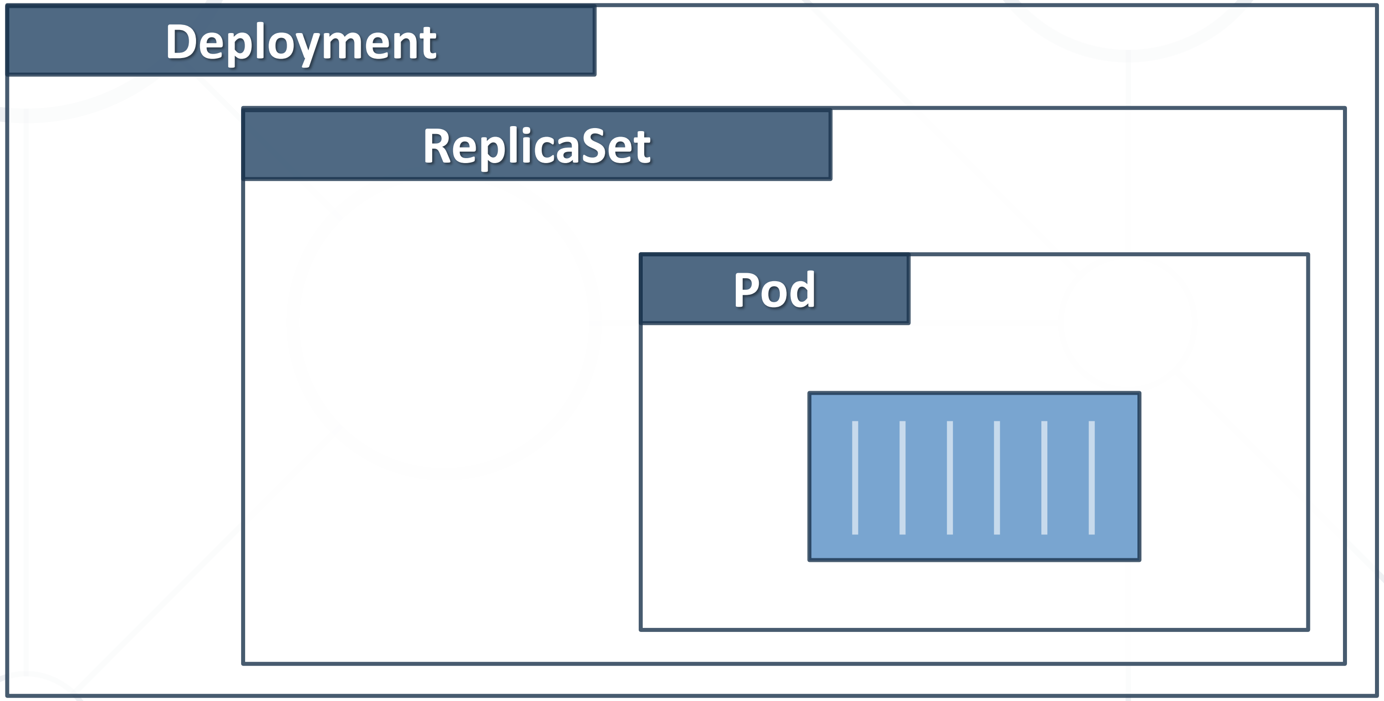
Replica Set Manifest *

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: appa-rs
spec:
  replicas: 3
  selector:
    matchLabels:
      app: appa
  template:
     … [POD definition] …
```

* Partial one but with the important parts included (except the pod definition)

Deployments

- **Even higher-level** workload
- Simplifies **updates** and **rollbacks**
- **Declarative** and **imperative** approach
- Self-**documenting**
- Suitable for **versioning**



https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

Deployment Manifest *

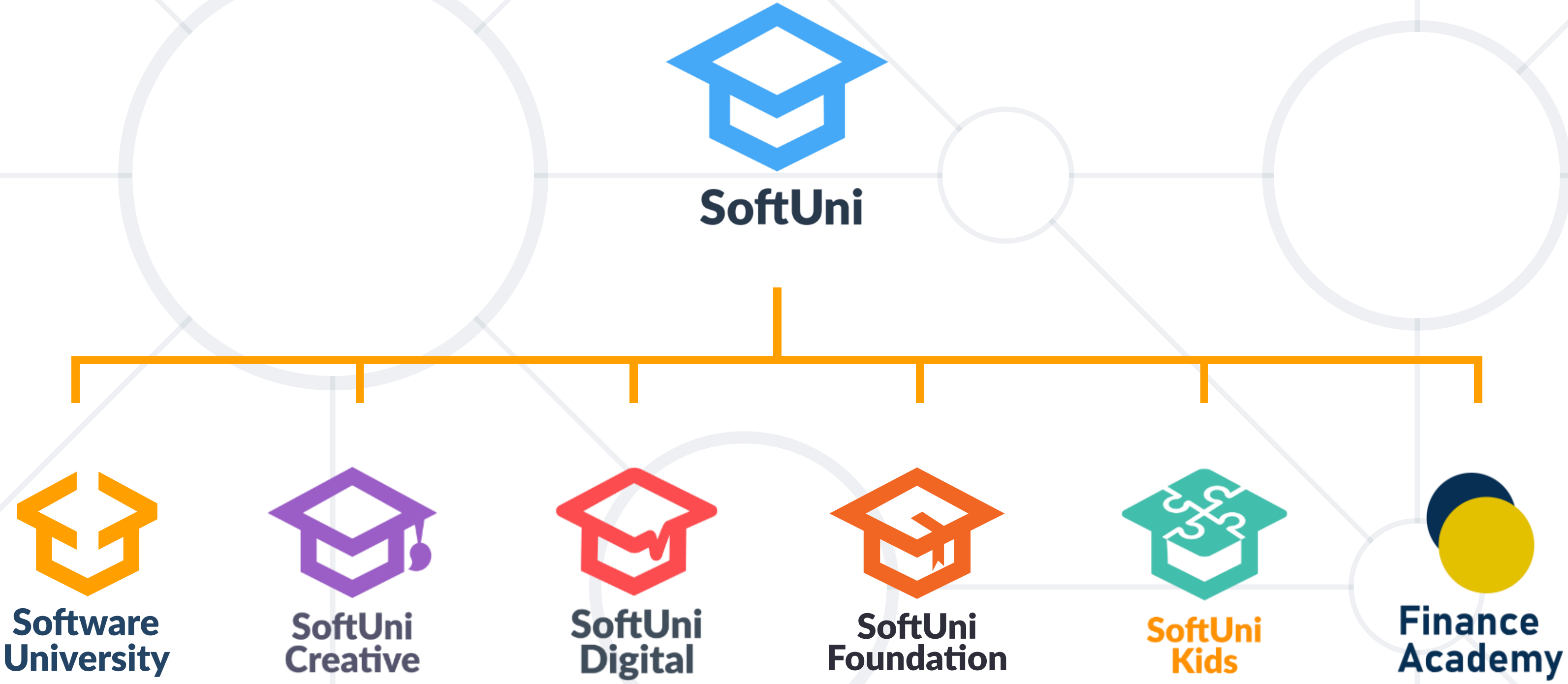
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: appa-deploy
spec:
  replicas: 3
  selector:
    matchLabels:
      app: appa
  minReadySeconds: 15        # optional, default 0
  strategy:                  # the whole block can be skipped
    type: RollingUpdate      # strategy to replace old pods, defaults to RollingUpdate
    rollingUpdate:
      maxUnavailable: 1      # maximum number of unavailable pods, defaults to 25%
      maxSurge: 1            # maximum number of pods that can be created in excess, defaults to 25%
  template:
     … [POD definition] …
```

* Partial one but with the important parts included (except the pod definition)

Practice

Live Exercise in Class (Lab)

Questions?



SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

Finance Academy

SoftUni Diamond Partners

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

Flutter™ International

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

BOSCH

Postbank
Решения за твоето утре

PHAR VISION

SmartIT

DXC TECHNOLOGY

createX

License

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

Software University

FORUM