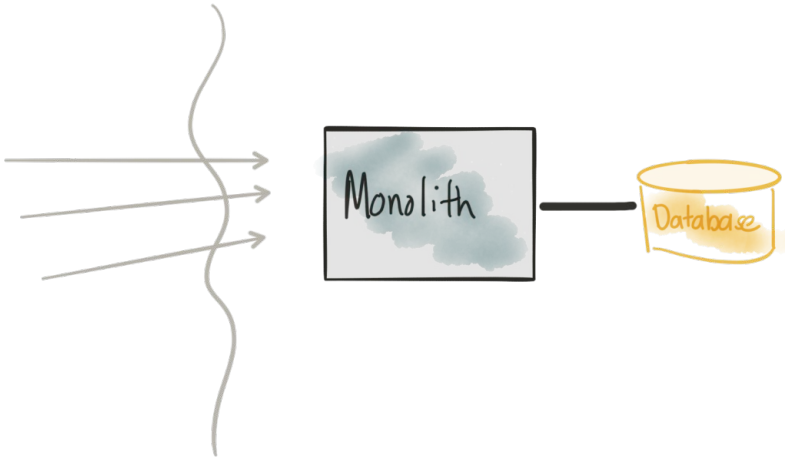
AKS: Kubernetes e Azure alla massima potenza

Alessandro Melchiori // @amelchiori

Monolith vs microservice(s)

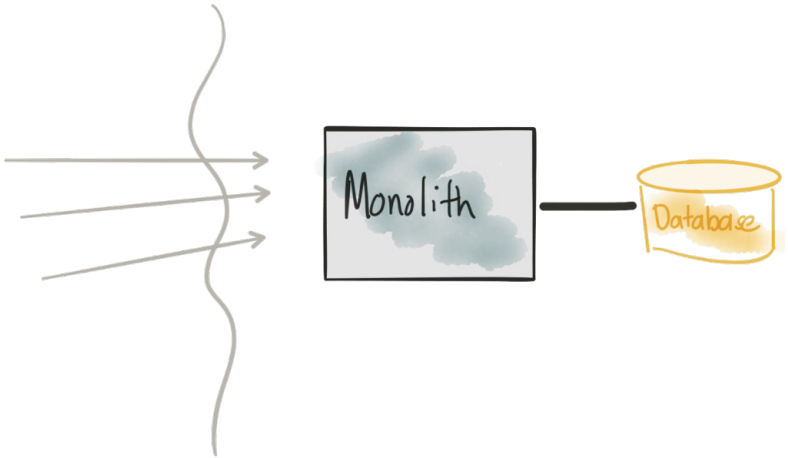
Monolith



The Good

- Monolith can be good when you are starting out
- Fewer moving parts enables easy deployment

Monolith



The “Bad”

- When your application (or company) grows, monolith begin slowing you down

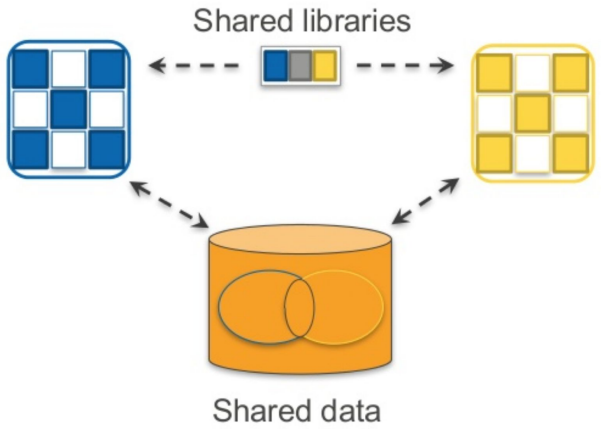
Challenges with monolithic software

Challenges with monolithic software

- Difficult to scale

Challenges with monolithic software

- Difficult to scale
- Architecture is hard to maintain and evolve
  - Es: too much software coupling



Challenges with monolithic software

- Difficult to scale
- Architecture is hard to maintain and evolve
  - Es: too much software coupling
- Long build/test/release cycle
  - Update to one functionality requires redeployment of the entire codebase

Challenges with monolithic software

- Difficult to scale
- Architecture is hard to maintain and evolve
  - Es: too much software coupling
- Long build/test/release cycle
  - Update to one functionality requires redeployment of the entire codebase
- Operations is a nightmare

Other options?

The biggest questions ever asked (some of)

- What happens after you die?

The biggest questions ever asked (some of)

- What happens after you die?
- What is life?

The biggest questions ever asked (some of)

- What happens after you die?
- What is life?
- **What is a microservice?**


Adam Sándor
@adamsand0r
Follow
Microservices are units of (independent!) deployment. If you need to deploy several of them together, then you're doing something wrong. Take a good look at your architecture and see if you can make them truly independent. If not, they probably should be in one service.
5:25 PM - 3 Oct 2018
65 Retweets 157 Likes
9
65
157


Kevin Sigmund
@ksigmund
Follow
Replying to @adamsand0r
Microservices are about logical, *not* physical boundaries. We are running multiple services inside the same process, but because they are logically independent, if the need arises, any/all of them could be deployed physically independent of the others.
2:59 AM - 5 Oct 2018
2 Likes
1
2

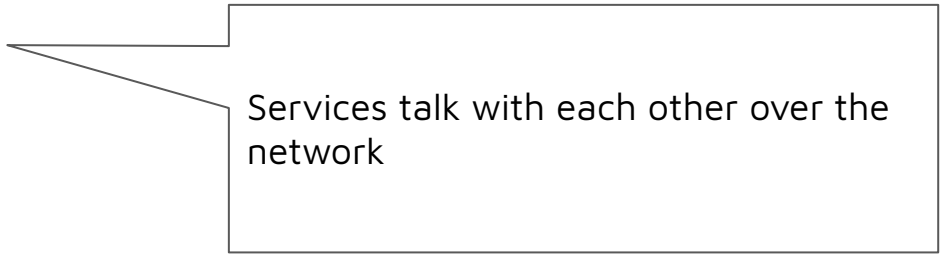
What are microservices

service -oriented
architecture composed of
loosely coupled elements
that have **bounded context**

Adrian Cockcroft

What are microservices

service -oriented
architecture composed of
loosely coupled elements
that have **bounded context**



Adrian Cockcroft

What are microservices

service -oriented
architecture composed of
loosely coupled elements
that have **bounded context**

You can update the services independently; updating one service doesn't require changing any other service

Adrian Cockcroft

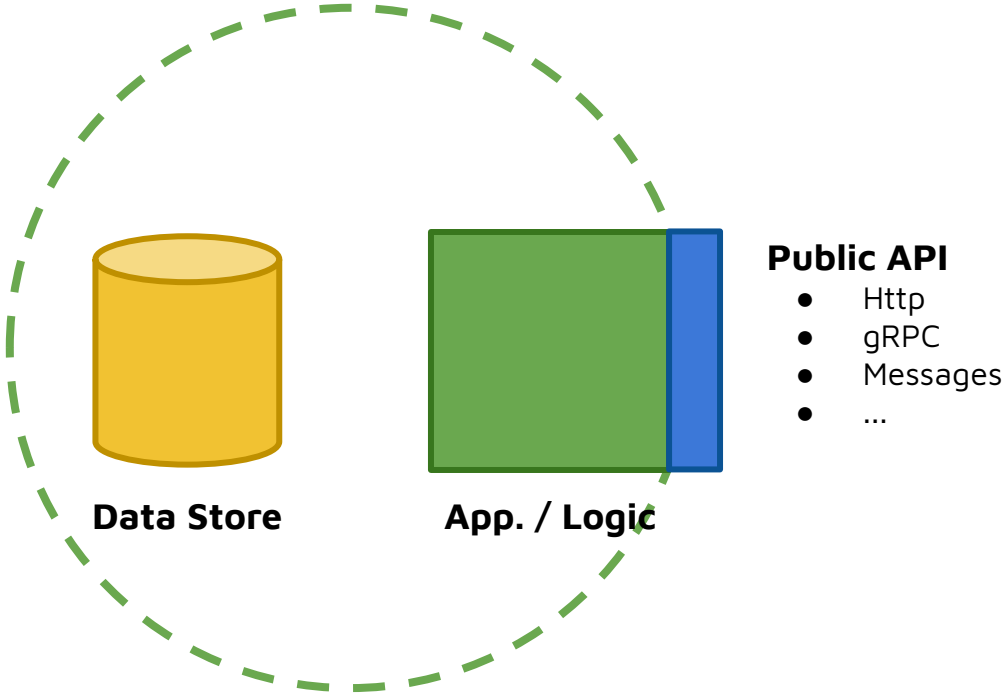
What are microservices

service -oriented
architecture composed of
loosely coupled elements
that have **bounded context**

Self-contained; you can update the code without knowing anything about the internals of other microservices

Adrian Cockcroft

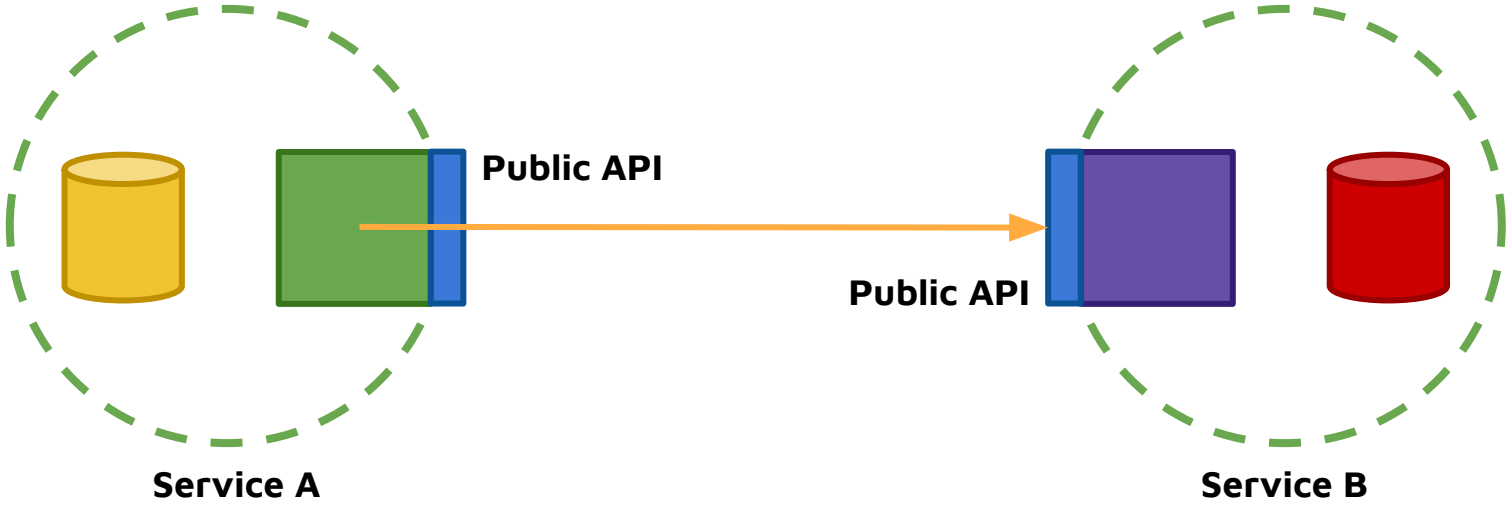
Anatomy of a (micro)service



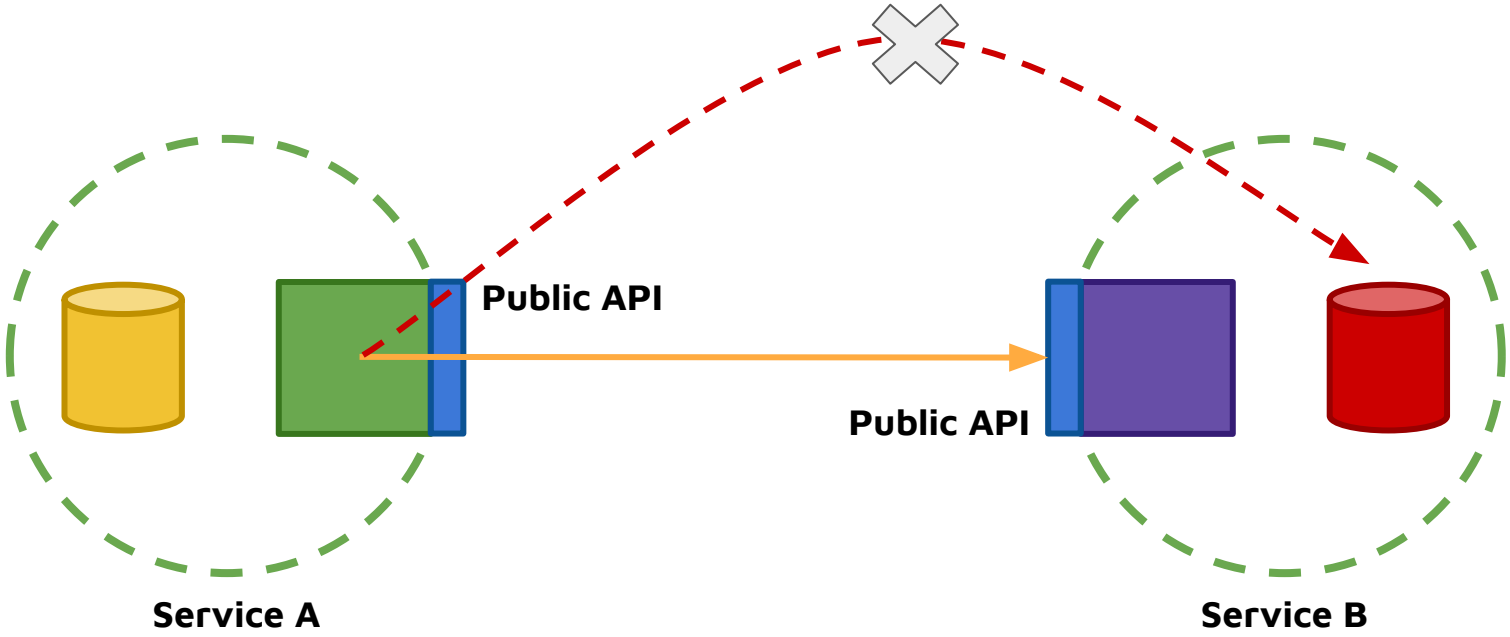
Public API

- Http
- gRPC
- Messages
- ...

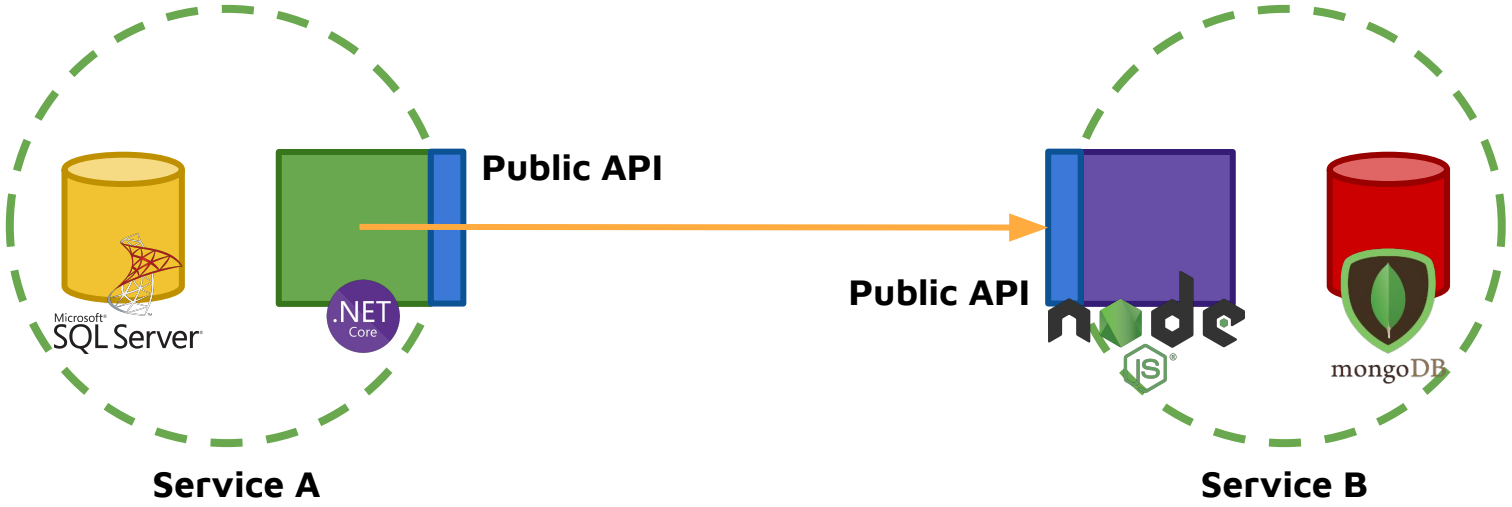
Anatomy of a (micro)service



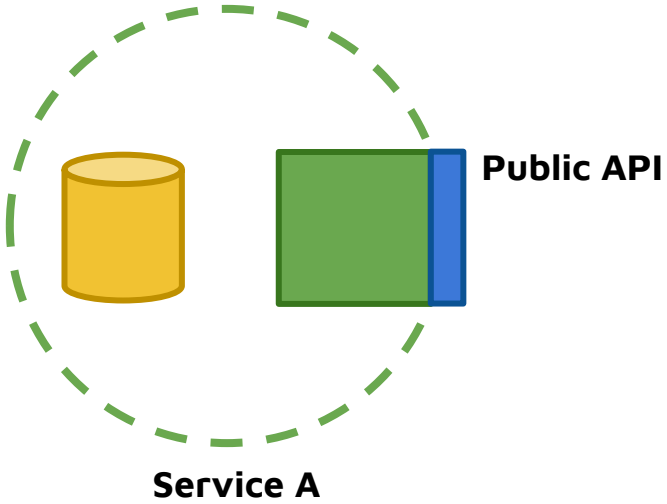
Anatomy of a (micro)service



Anatomy of a (micro)service



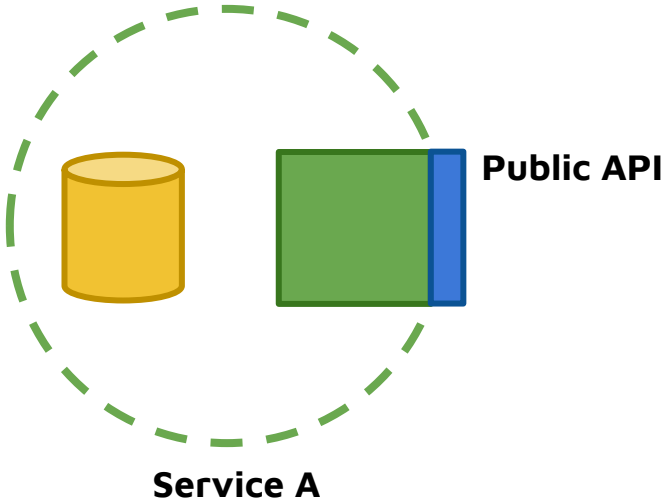
Anatomy of a (micro)service



Version 1.0.0

```
storeRestaurant(id, name)
```

Anatomy of a (micro)service



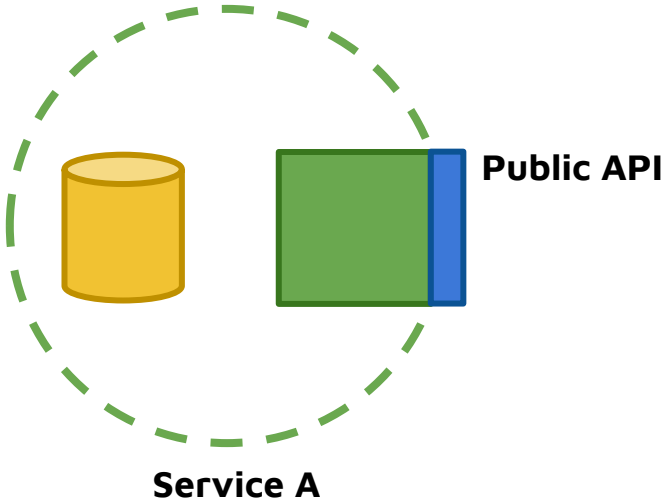
Version 1.0.0

```
storeRestaurant(id, name)
```

Version 1.1.0

```
storeRestaurant(id, name)
storeRestaurant(id, name, metadata)
addReview(restaurantId, rating, comments)
```

Anatomy of a (micro)service



Version 1.0.0

```
storeRestaurant(id, name)
```

Version 1.1.0

```
storeRestaurant(id, name)
storeRestaurant(id, name, metadata)
addReview(restaurantId, rating, comments)
```

Version 2.0.0

```
storeRestaurant(id, name, metadata)
addReview(restaurantId, rating, comments)
```

Microservice architecture

The “Good”

- An application is sum of its components
- Better fault isolation
- Components can be spread across multiple servers

Microservice architecture

The “Good”

- An application is sum of its components
- Better fault isolation
- Components can be spread across multiple servers

The “Bad”

- Many components, many moving parts
- Difficult to manage inter-communication
- Manual management can be difficult

Microservice architecture

The “Good”

- An application is sum of its components
- Better fault isolation
- Components can be spread across multiple servers

The “Bad”

- Many components, many moving parts
- Difficult to manage inter-communication
- Manual management can be difficult

Welcome Kubernetes

Kubernetes

Greek for “Helmsman” < the person who steers a ship

Kubernetes

Greek for “Helmsman” < the person who steers a ship

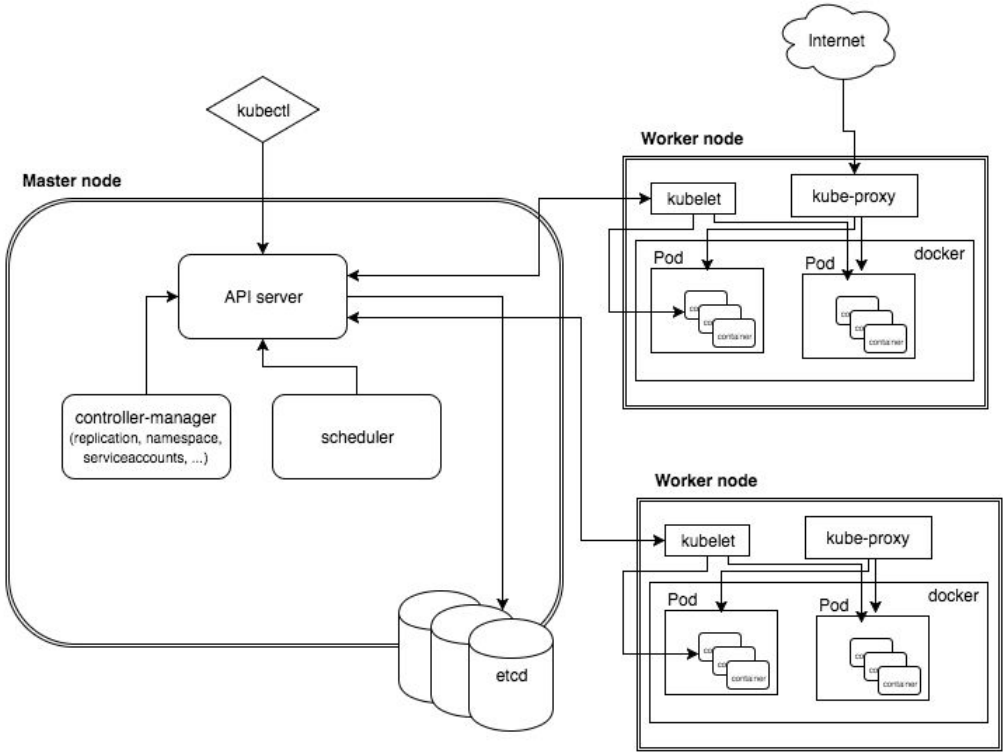
K8s

Greek for “Helmsman” < the person who steers a ship

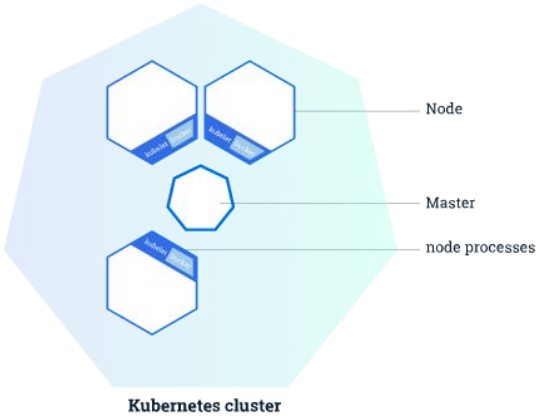
K8s: some infos

- Born in Google
- Donated to CNCF in 2014
- Open source (Apache 2.0)
- v1.0 July 2015
- Written in Go/Golang
- Code is on GitHub (where otherwise?)

K8s: the big picture view

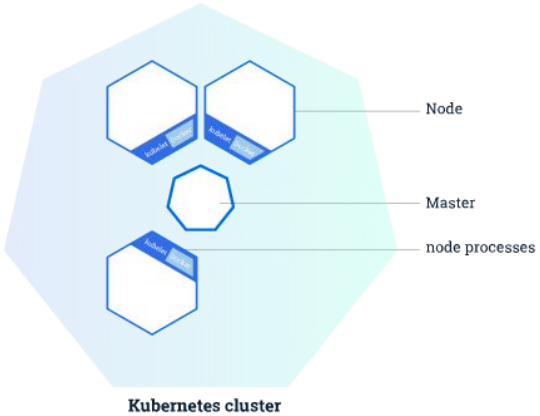


K8s: big picture view



- The **Master** is responsible for managing the cluster

K8s: big picture view

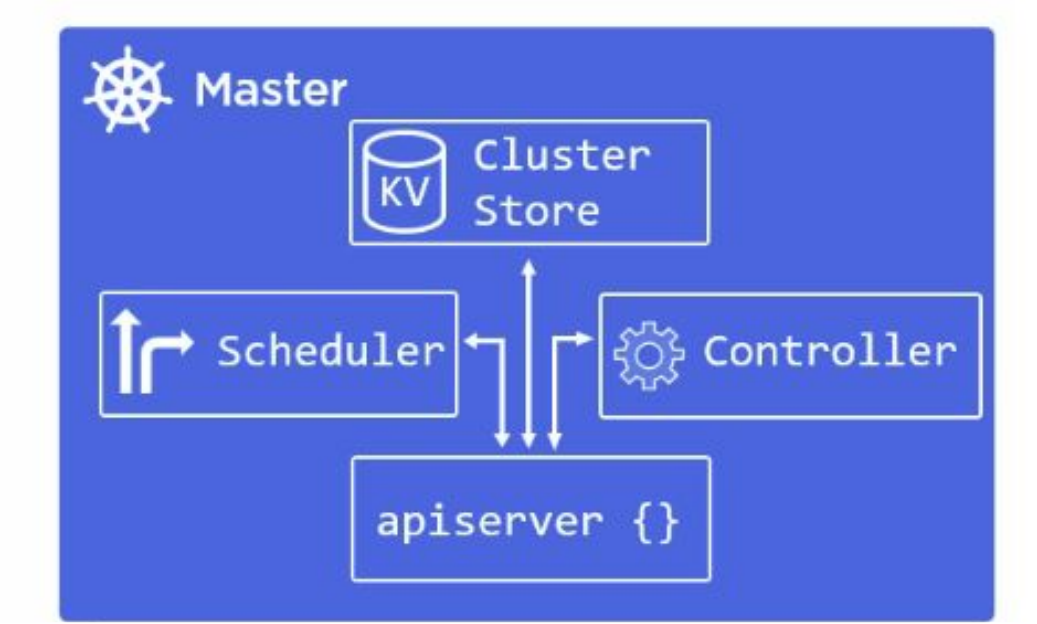


- The **Master** is responsible for managing the cluster
- A **node** is a VM or a physical computer that serves as a worker machine in a Kubernetes cluster.

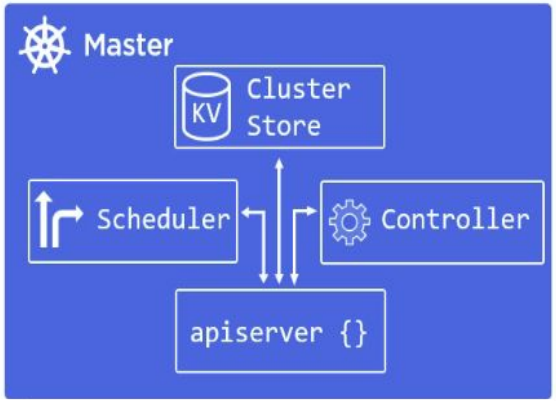
Master(s)

The K8s control plane

K8s: master components



K8s: master components

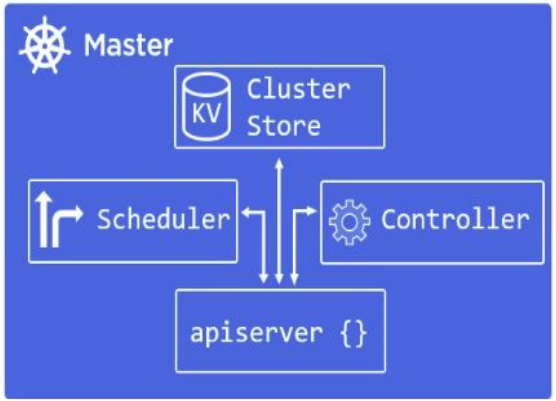


kube-apiserver

Component on the master that exposes the Kubernetes API. It is the front-end for the Kubernetes control plane.

It is designed to scale horizontally

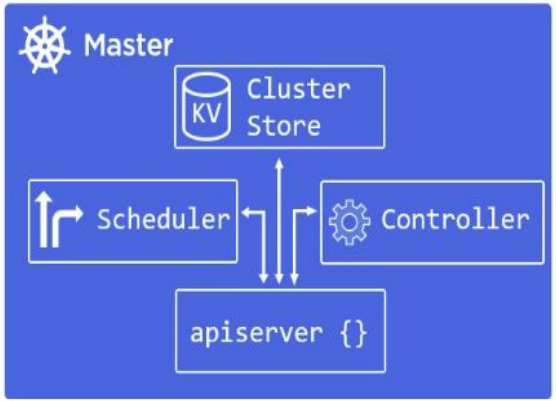
K8s: master components



etcd

Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data.

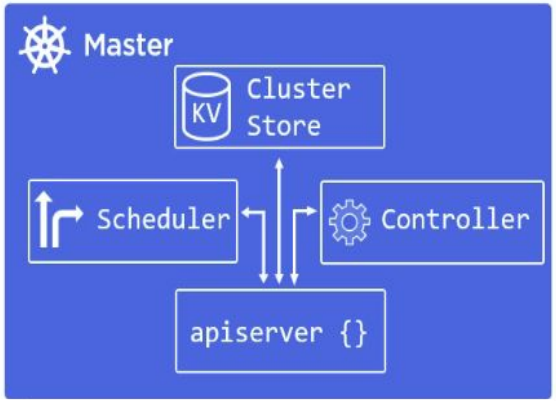
K8s: master components



kube-scheduler

Component on the master that watches newly created pods that have no node assigned, and selects a node for them to run on.

K8s: master components



kube-controller-manager

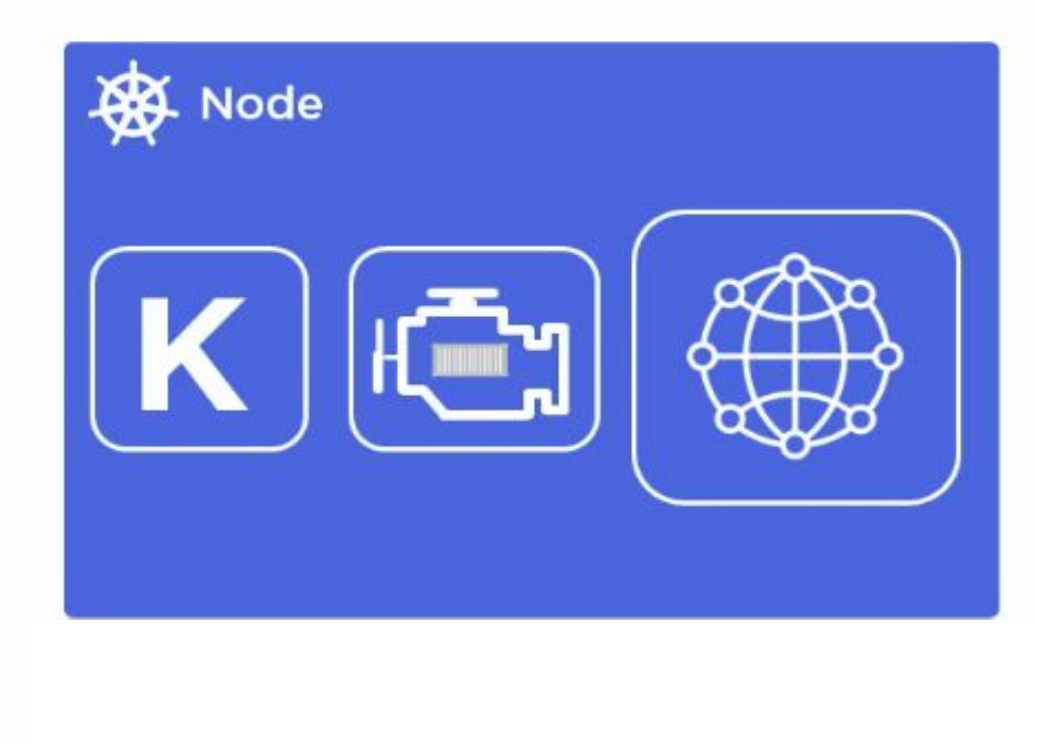
Component on the master that runs controllers:

- Node controller
- Replication controller
- Endpoints controller
- Service Account & Token controller

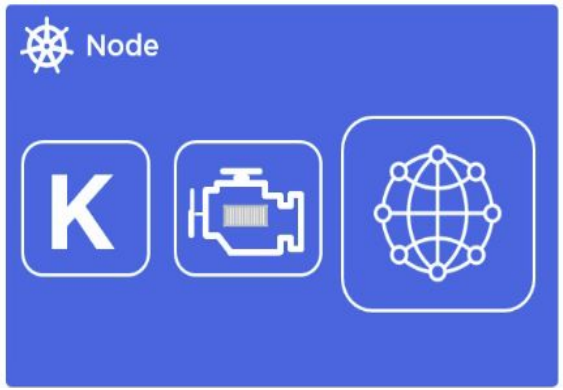
Node(s)

The K8s workers

K8s: node components



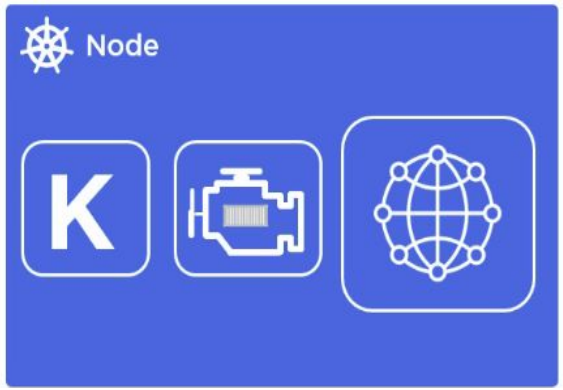
K8s: node components



kubelet

An agent that runs on each node in the cluster. It makes sure that containers are running in a pod.

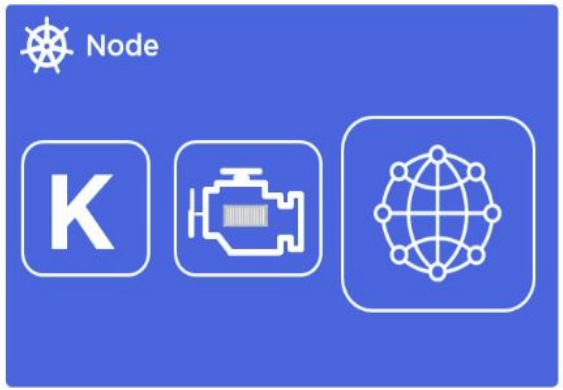
K8s: node components



kube-proxy

It is like the network brain of the node. It is a network proxy which reflects Kubernetes networking services on each node.

K8s: node components



Container runtime

It’s the software that is responsible for running containers. Kubernetes supports several runtimes: Docker, rkt, runc and any OCI runtime-spec implementation.

K8s objects

K8s objects overview

Kubernetes contains a number of *abstractions* that represent the state of your system: deployed containerized applications and workloads, their associated network and disk resources, and other information about what your cluster is doing.

These abstractions are represented by ***objects*** in the Kubernetes API

K8s objects

Basic Kubernetes objects:

- Pod
- Service
- Volume
- Namespace

K8s objects

Basic Kubernetes objects:

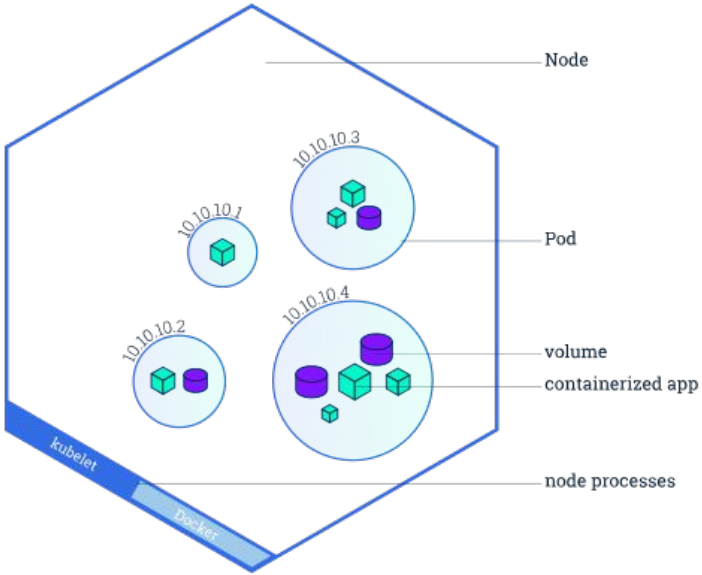
- Pod
- Service
- Volume
- Namespace

Higher-level abstraction (controllers):

- ReplicaSet
- Deployment
- StatefulSet
- DaemonSet
- Job

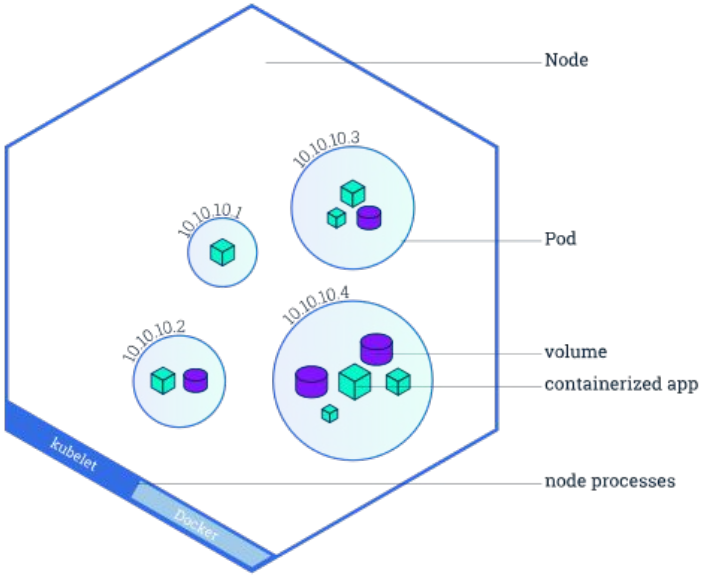
Pods, Services and Deployment

Pod overview



- Is the basic building block of Kubernetes
- Represents a running process on the cluster
- Consists of either a single container or a small number of containers that are tightly coupled and that share resources

Pod phases

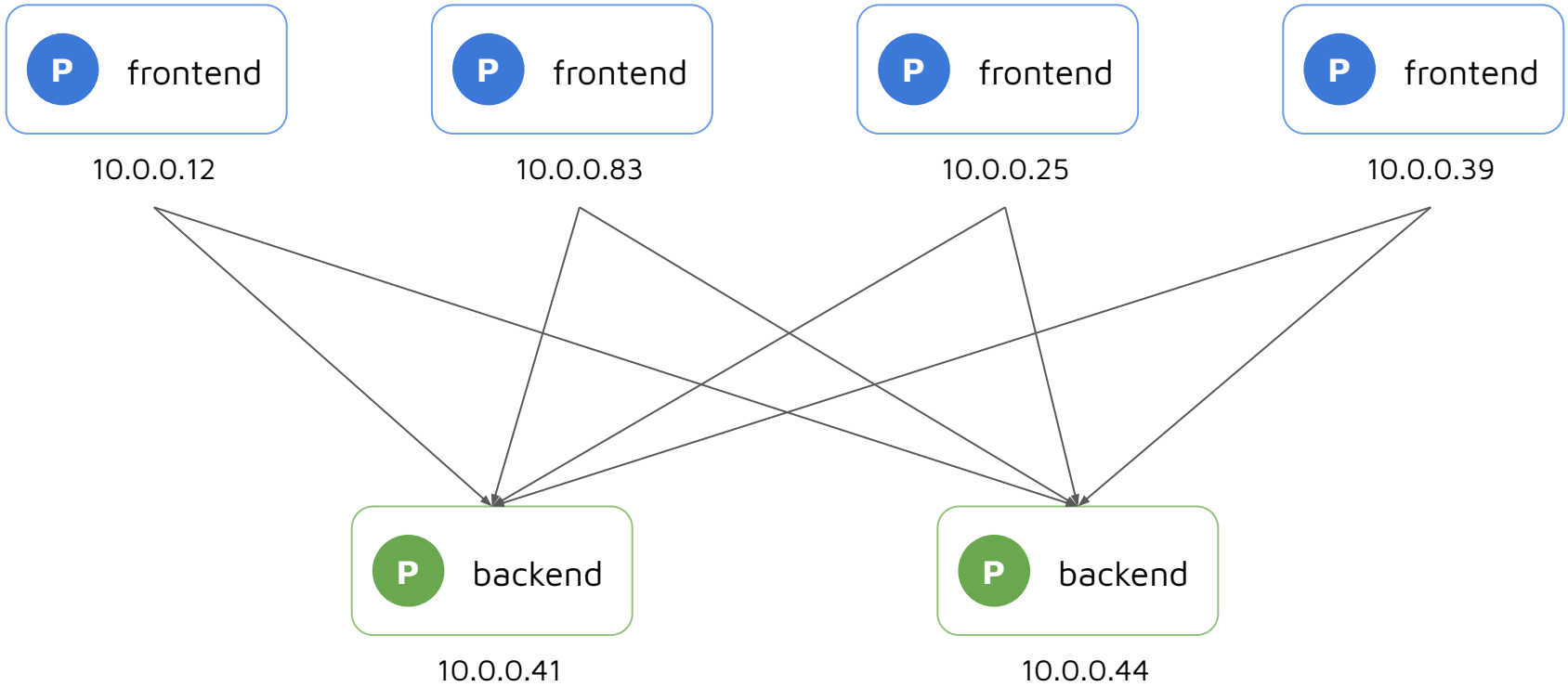


Pods are mortal

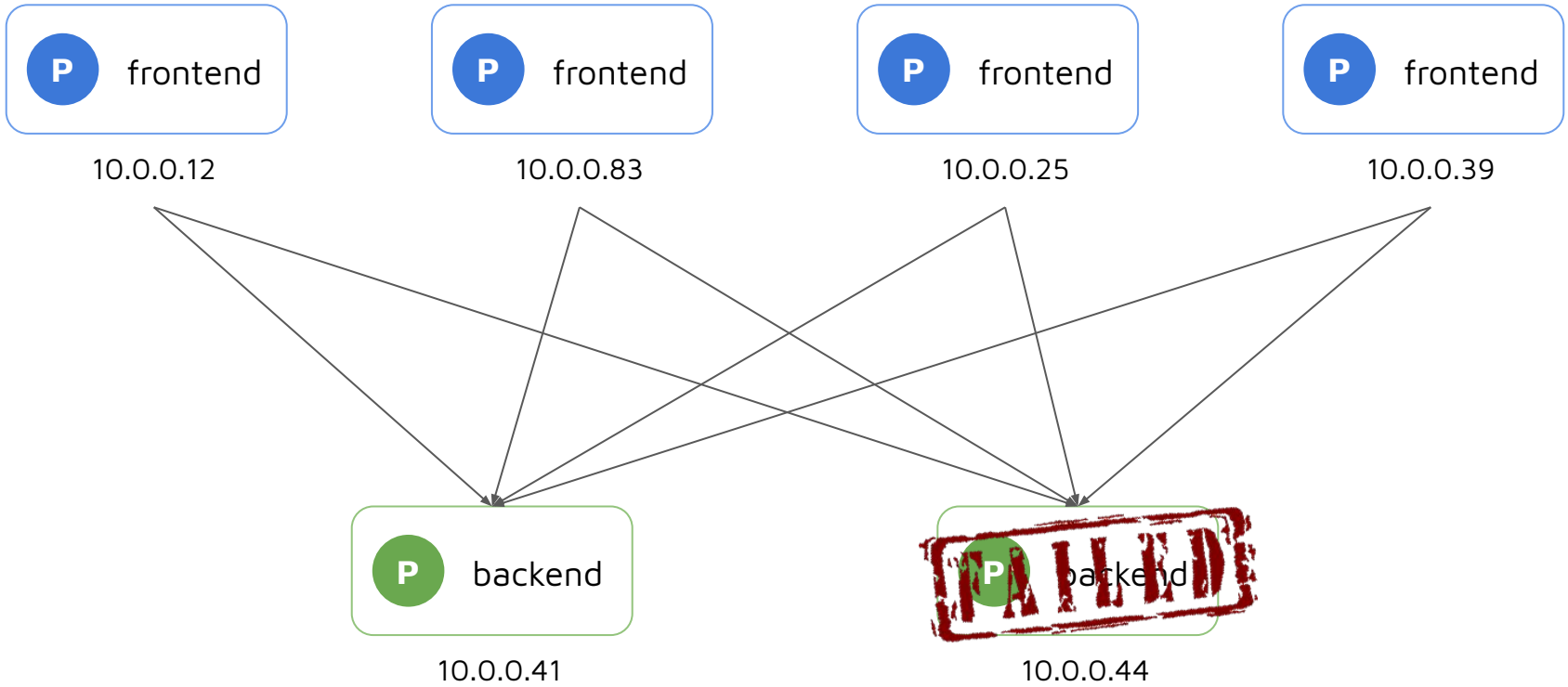
The phase of a Pod is a simple, high-level summary of where the Pod is in its lifecycle:

- Pending
- Running
- Succeeded
- Failed
- Unknown

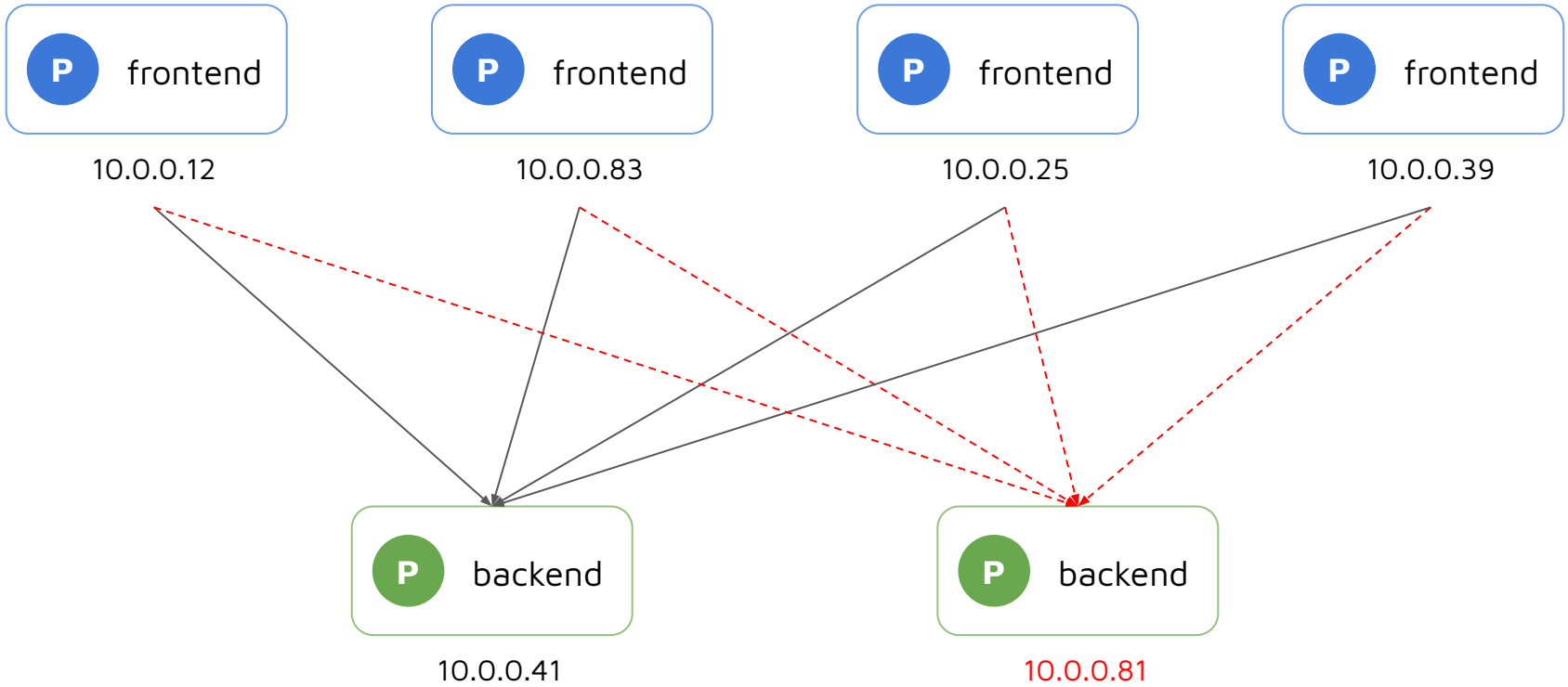
Service overview



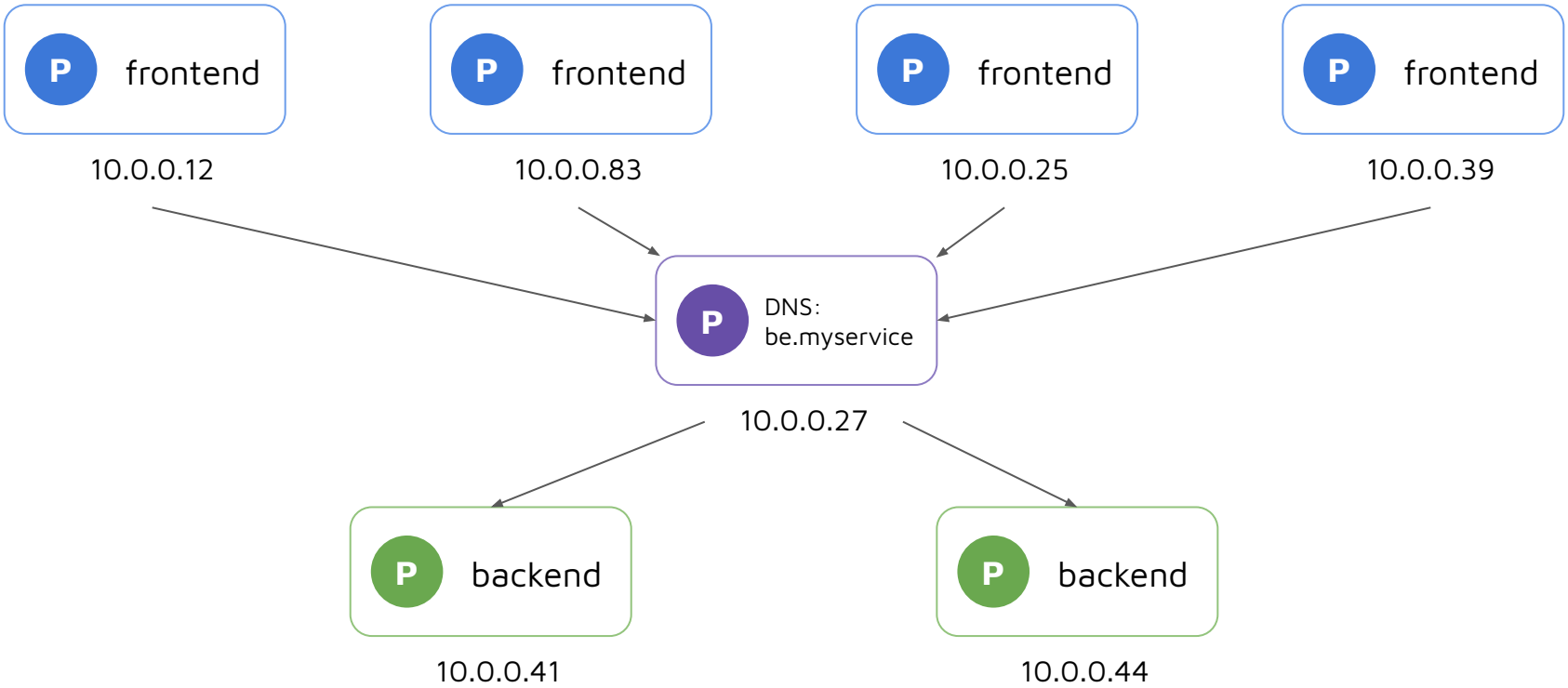
Service overview



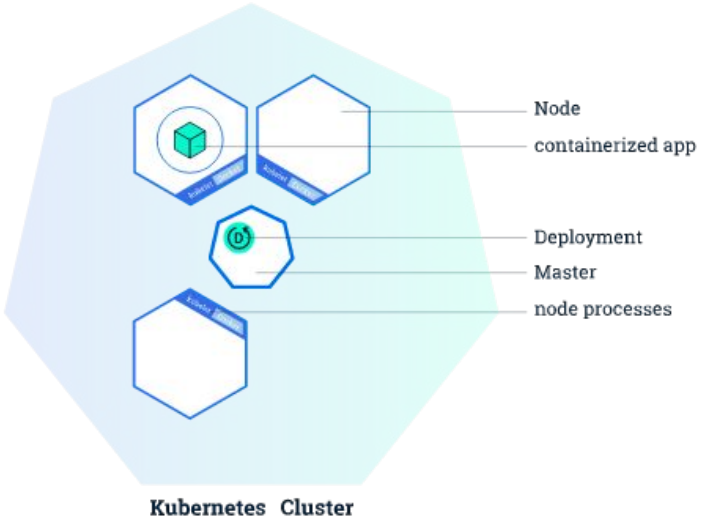
Service overview



Service overview



Deployment overview



- It provides declarative updates for Pods and ReplicaSets.
- You describe a *desired state* in a Deployment object, and the Deployment controller changes the actual state to the desired state at a controlled rate.

Declarative model

&

Desired state

Management techniques

The **kubectl** command-line tool supports several different ways to create and manage Kubernetes objects:

- Imperative commands
- Imperative object configuration
- Declarative object configuration

Imperative commands

The simplest way to get started or to run a one-off task in a cluster.

```
kubectl run nginx --image nginx
```

Imperative commands

Pro:

- Commands are simple, easy to learn and easy to remember.
- Commands require only a single step to make changes to the cluster

Cons:

- Commands do not integrate with change review processes.
- Commands do not provide an audit trail associated with changes.

Imperative object configuration

In imperative object configuration, the kubectl command specifies the operation (create, replace, etc.), optional flags and at least one file name.

The file specified must contain a full definition of the object in YAML or JSON format.

```
kubectl create -f nginx.yaml
```

Imperative object configuration

Pro:

- Object configuration can be stored in a source control system such as Git (vs. imperative commands)
- It's simpler and easier to understand (vs. declarative object configuration)

Cons:

- Object configuration requires basic understanding of the object schema (vs. imparative commands)
- It works best on files, not directories (vs. declarative object configuration)
- Updates to live objects must be reflected in configuration files, or they will be lost during the next replacement (vs. declarative object configuration)

Declarative object configuration

Using declarative object configuration, a user operates on object configuration files stored locally, however the user does not define the operations to be taken on the files.

Create, update, and delete operations are automatically detected per-object by kubectl.

```
kubectl apply -f configs/
```

Declarative object configuration

Pro:

- Changes made directly to live objects are retained, even if they are not merged back into the configuration files
- It has better support for operating on directories and automatically detecting operation types per-object

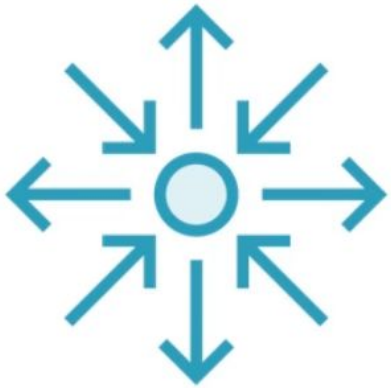
Cons:

- Declarative object configuration is harder to debug

K8s + Azure = AKS

Self-hosting K8s cluster

Manually install master and worker nodes



Need to consider master HA, adding additional worker nodes, patching, updates, ...

Azure Kubernetes Service

- Simplifies deployment, management and operations of K8s
- Makes it quick and easy to deploy and manage containerized applications without container orchestration expertise
- Eliminates the burden of ongoing operations and maintenance by provisioning, upgrading and scaling resources on demand

Alessandro Melchiori

Software developer @ CodicePlastico
alessandro@codiceplastico.com
@amelchiori