

# Kubernetes for developers

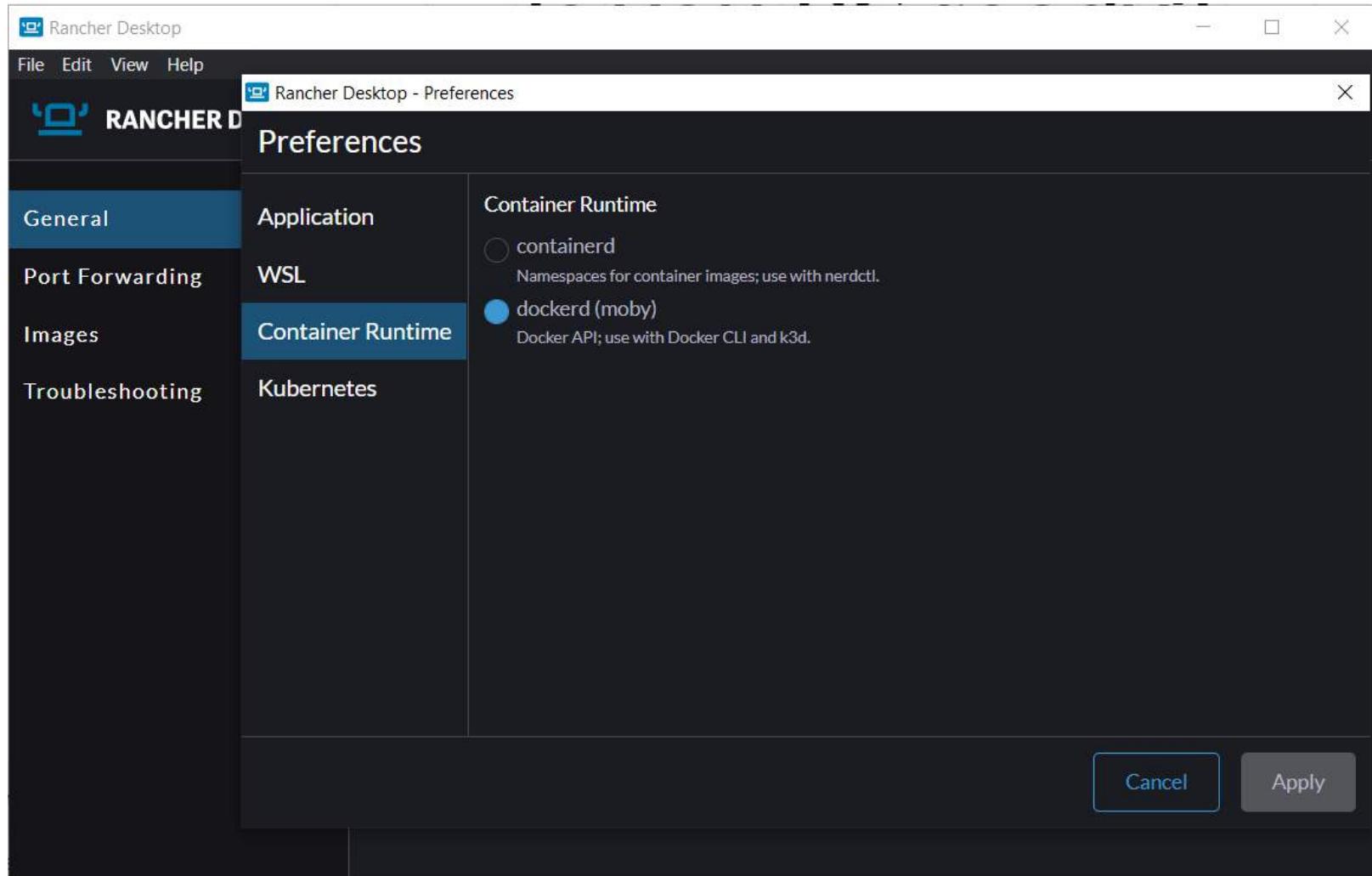
Alior 2023

# Before we begin

Organizational stuff



# Is your PC ready? (we will need this later)



# Download image

```
# docker pull  
docker pull gutek/dumpster:v1
```

# Contract

1. Ask about anything!
2. Make notes!
3. Do your exercises, and we can guarantee you will gain understanding of how things works
4. If you feel you need a break, tell us/write us!

# Introduction

# Goal of the workshop

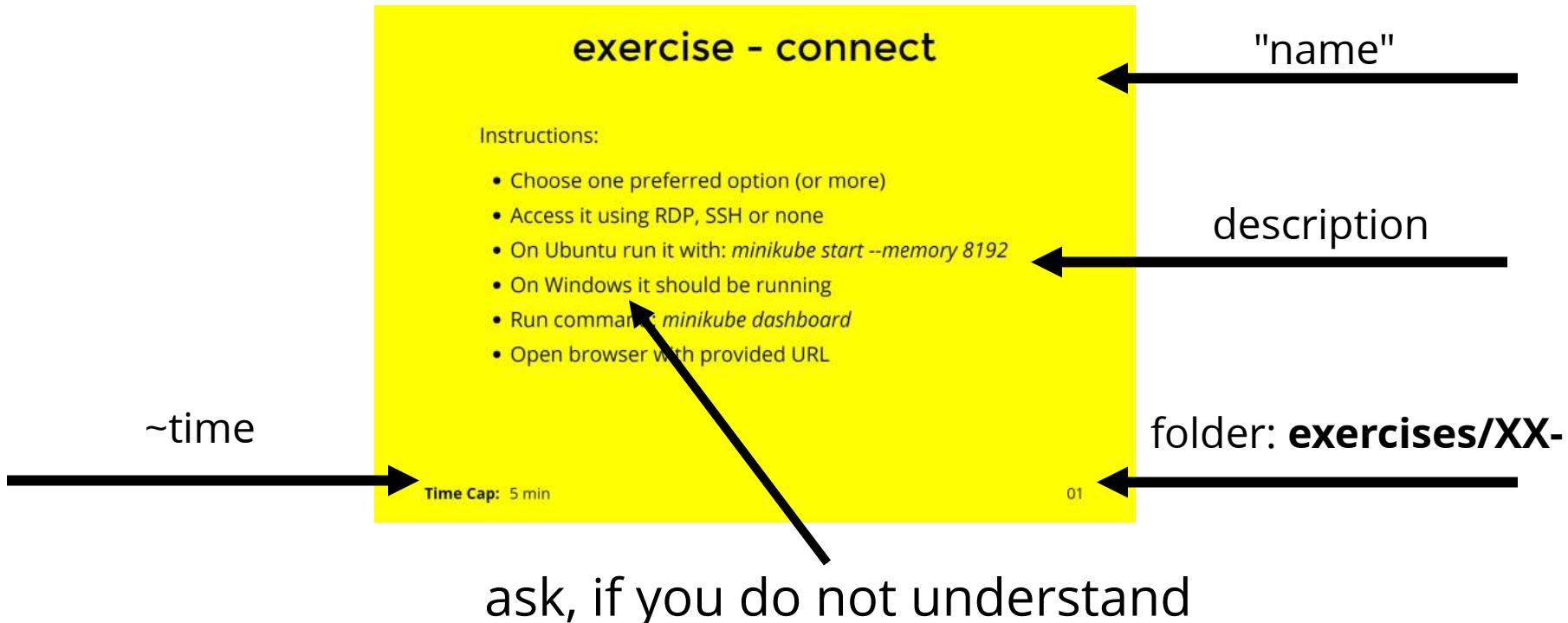
# Follow us

[https://slides.com  
/d/HIO2MAY/live](https://slides.com/d/HIO2MAY/live)

[all slides](#)

# Exercises

<https://github.com/ptrstpp950/k8s-alior>





# Big bang theory

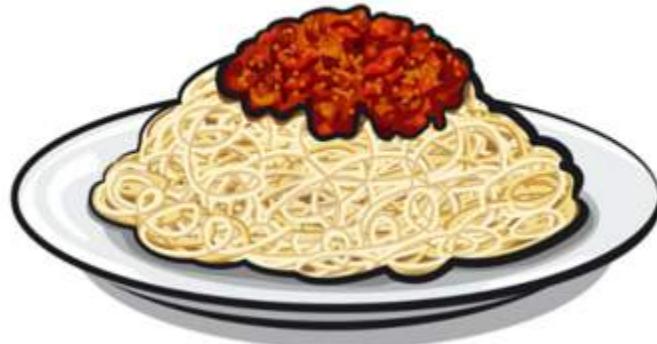
**Definitions are hard  
When someone asks for  
them, so we will create  
them together**

THE EVOLUTION OF  
**SOFTWARE ARCHITECTURE**

By @benorama

# 1990's

SPAGHETTI-ORIENTED  
ARCHITECTURE  
(aka Copy & Paste)



# 2000's

LASAGNA-ORIENTED  
ARCHITECTURE  
(aka Layered Monolith)



# 2010's

RAVIOLI-ORIENTED  
ARCHITECTURE  
(aka Microservices)



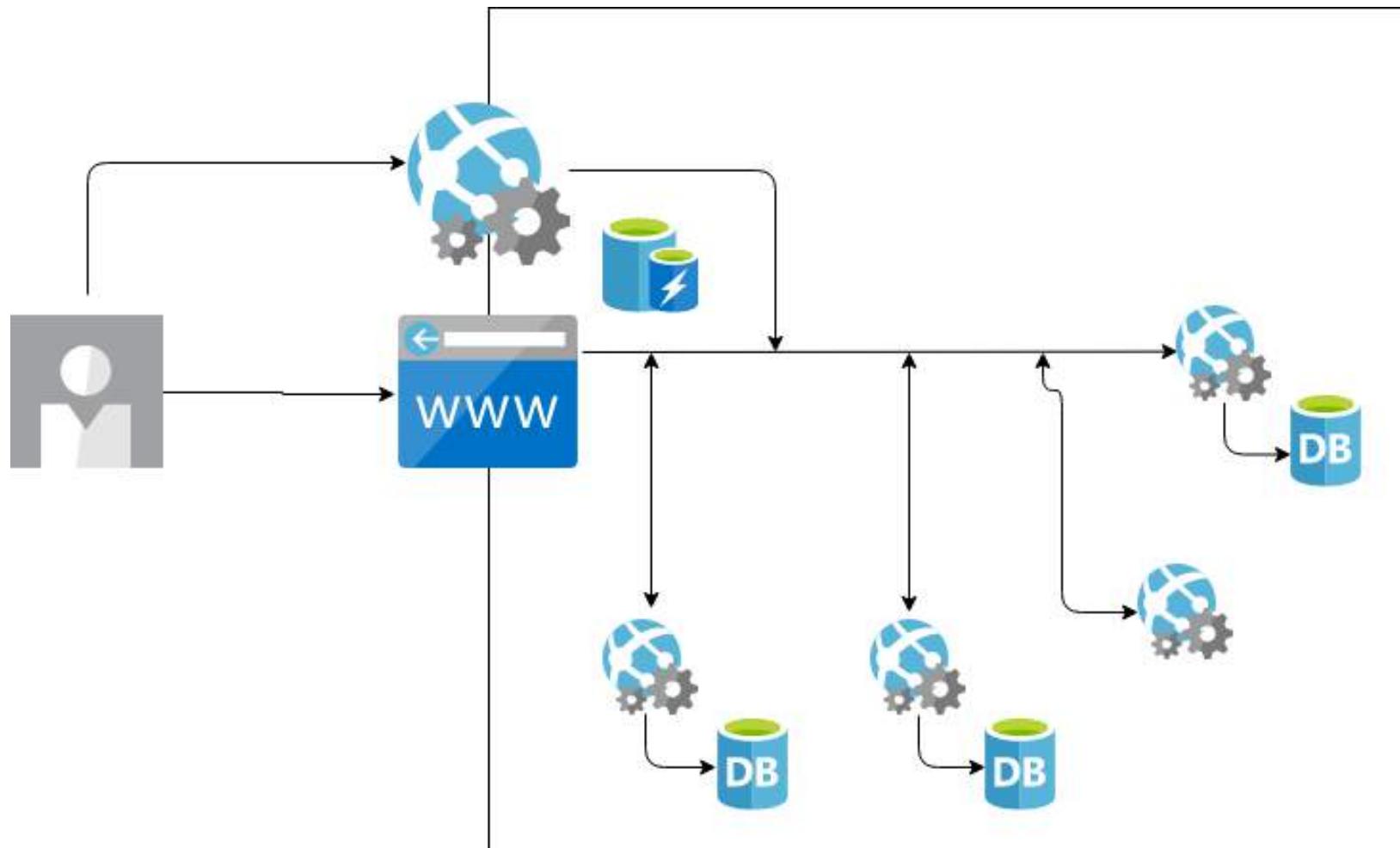
# WHAT'S NEXT?

## PROBABLY PIZZA-ORIENTED ARCHITECTURE

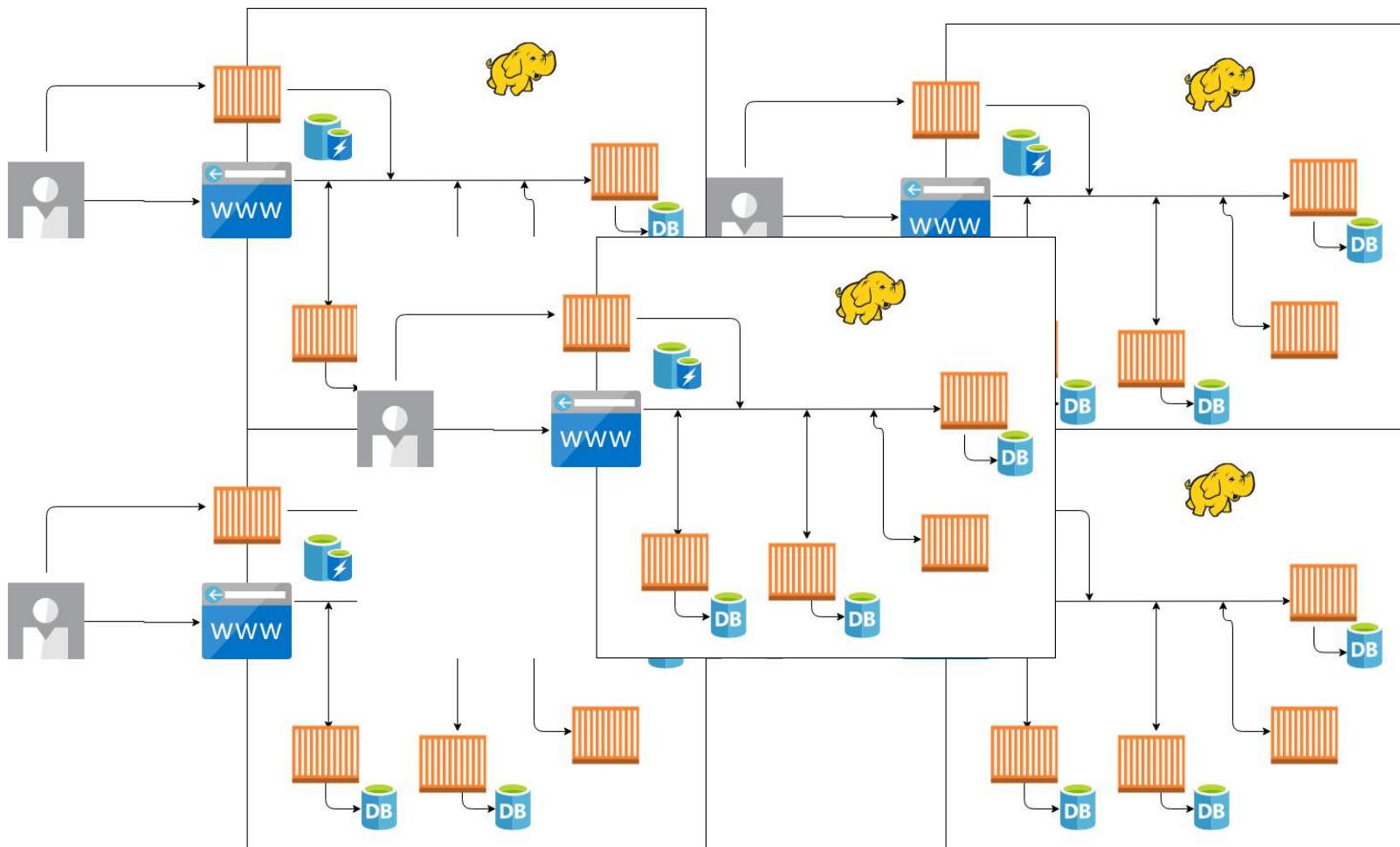
*"If you can't feed a team with two pizzas,  
it's too large. That limits a task force to five to  
seven people, depending on their appetites"*

Jeff Bezos

# A bit lasagna and ravioli



# Modern one



# Deployment



gifak.net

# orchestration (platform)

- Is it responsible for scaling?
- Is it responsible for monitoring?
- Is it responsible for deployments?
- Is it responsible for configuration?
- Is it responsible for ...?

# Kubernetes

- From Greek, meaning *helmsman* or *pilot*
- Does not limit the types of applications supported
- Does not deploy source code and does not build your application
- Does not provide application-level services
- Does not dictate logging, monitoring, or alerting solutions
- Does not provide nor mandate a configuration language/system
- Does not provide nor adopt any comprehensive machine configuration, maintenance, management, or self-healing systems

# K8s

- What's k8s?

# K8s

- Where do you know it is used?
- Where can you use it?
- Who uses it?
- ...

# K8s discussion

# WHY not K8s?

The background of the image is a dense, overlapping stack of shipping containers in a port. The containers are primarily blue, with many featuring orange and white stripes. Some have company logos like "CMA CGM", "OPDR", and "GRUPO". The perspective is from above, looking down at the sprawling expanse of cargo.

# Docker

# It's not rocket science



# Tooling...



*gifson.net*

# Deployment...



# Scaling

But docker  
have:

- Swarm
- ~~Compose~~



# Kubernetes in action



@ 1000



# How to work

# Docker Desktop

Docker is popular container platform allowing to run linux\* containers on variety of platforms including Windows and Mac.

Docker Inc. provides kubernetes cluster for local development - this cluster is hosted in... containers ;)

It's a good tool to be used on daily basis if we are working with docker too. If not, why bother.

\* to serve the greater good of humanity and the World,  
let's not talk about windows containers...

# ~~Docker~~ Rancher Desktop

Docker is popular container platform allowing to run linux\* containers on variety of platforms including Windows and Mac.

~~Docker Inc.~~ Rancher provides kubernetes cluster for local development - this cluster is hosted in... containers ;)

It's a good tool to be used on daily basis if we are working with docker too. If not, why bother.

\* to serve the greater good of humanity and the World,  
let's not talk about windows containers...

# minikube

Minikube is a tool that makes it easy to run Kubernetes locally.

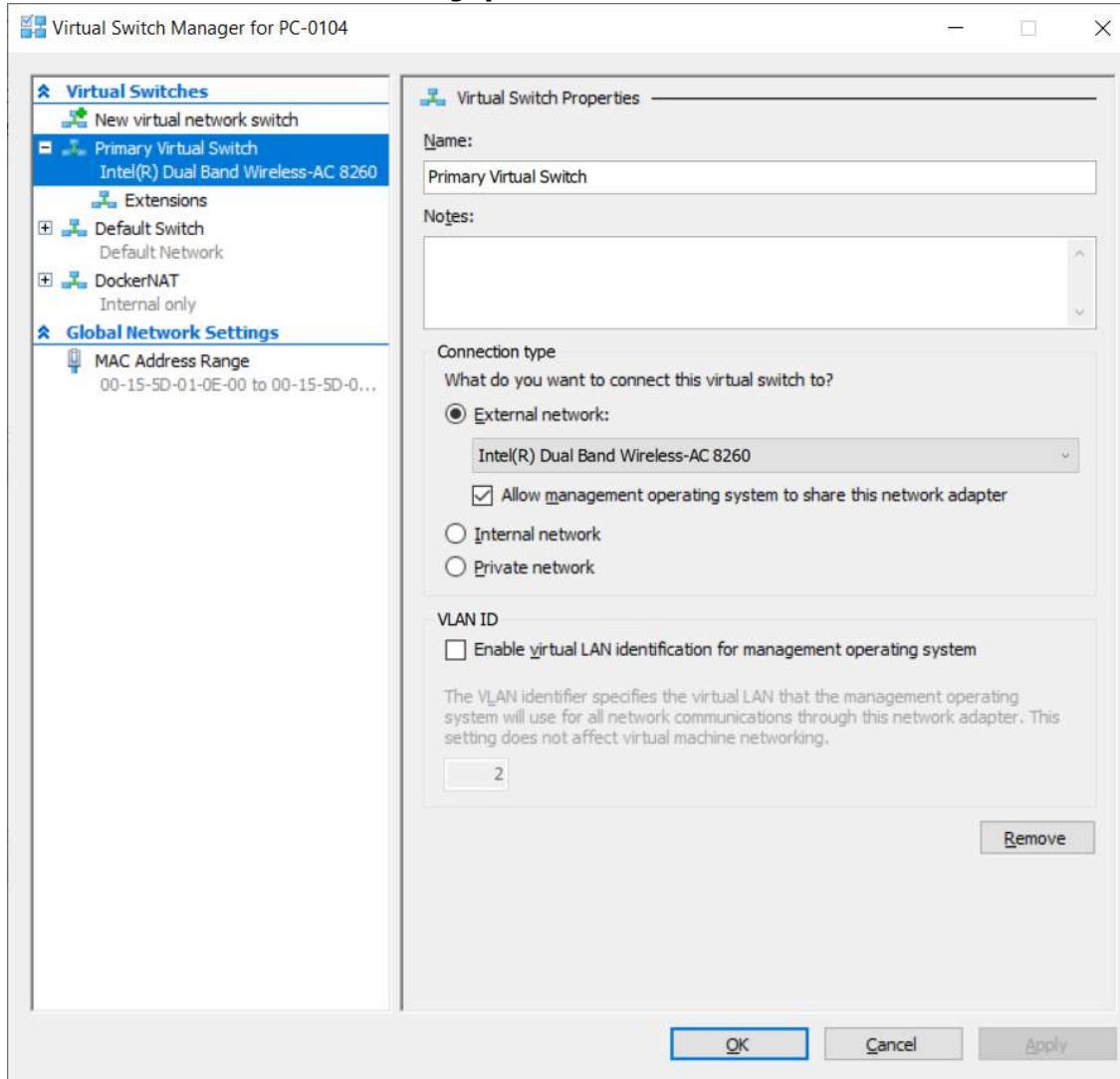
Minikube runs a single-node Kubernetes cluster inside a VM on your laptop for users looking to try out Kubernetes or develop on day-to-day basis.

# installation

```
# linux  
sudo apt minikube  
  
# mac  
brew install minikube  
  
# win chocolaty  
choco install minikube  
  
# other options  
https://github.com/kubernetes/minikube/
```

# minikube

## Windows - Hyper-V virtual switch



# minikube

## basic commands

```
# start
minikube start

# start full
minikube start --memory 8192 --vm-provider virtualbox
minikube start --vm-driver hyperv --hyperv-virtual-switch "Primary Virtual Switch" --memory
minikube start --memory 8192 --vm-provider none

# destroy
minikube delete
```

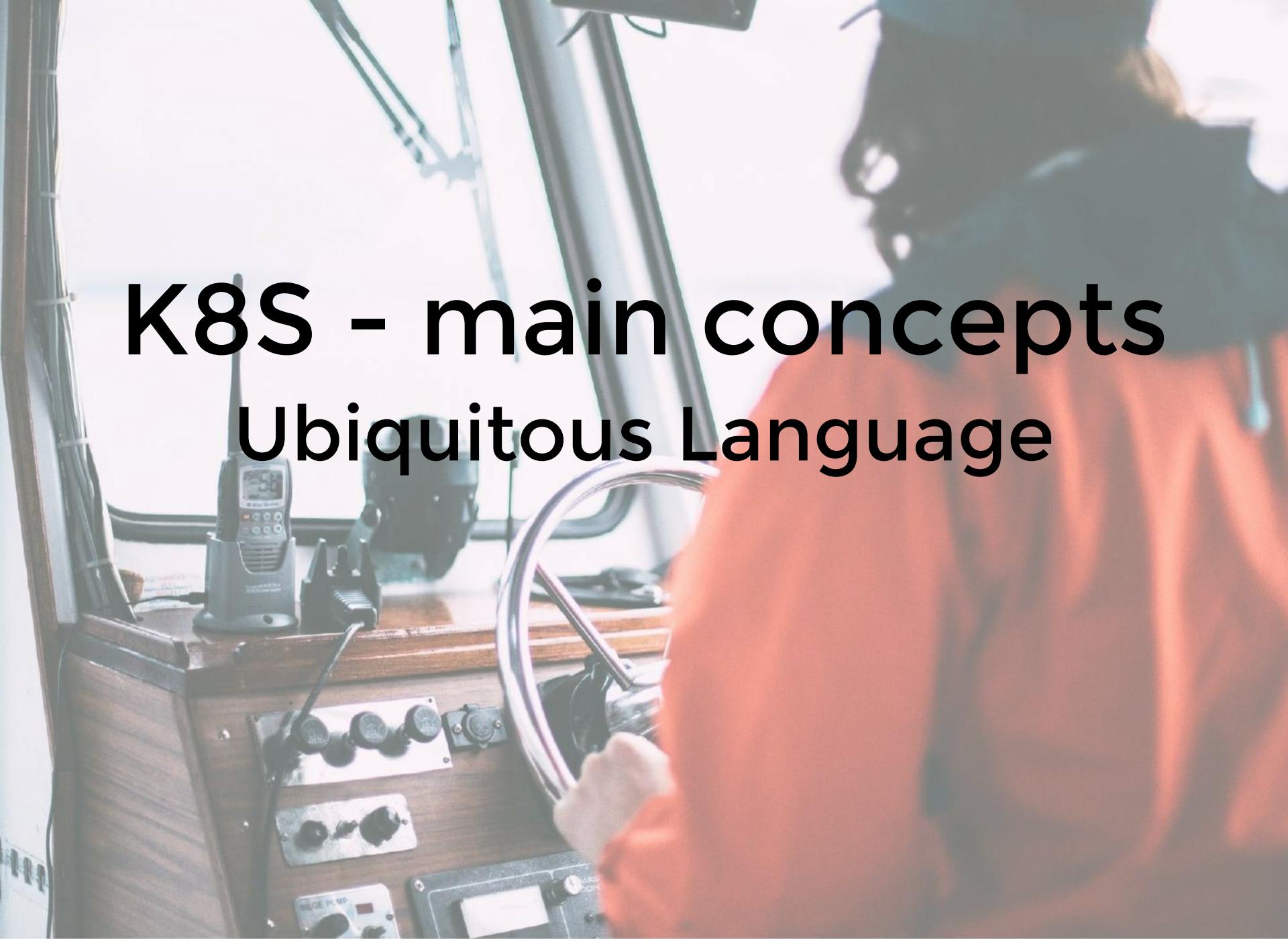
# exercise - connect

Instructions:

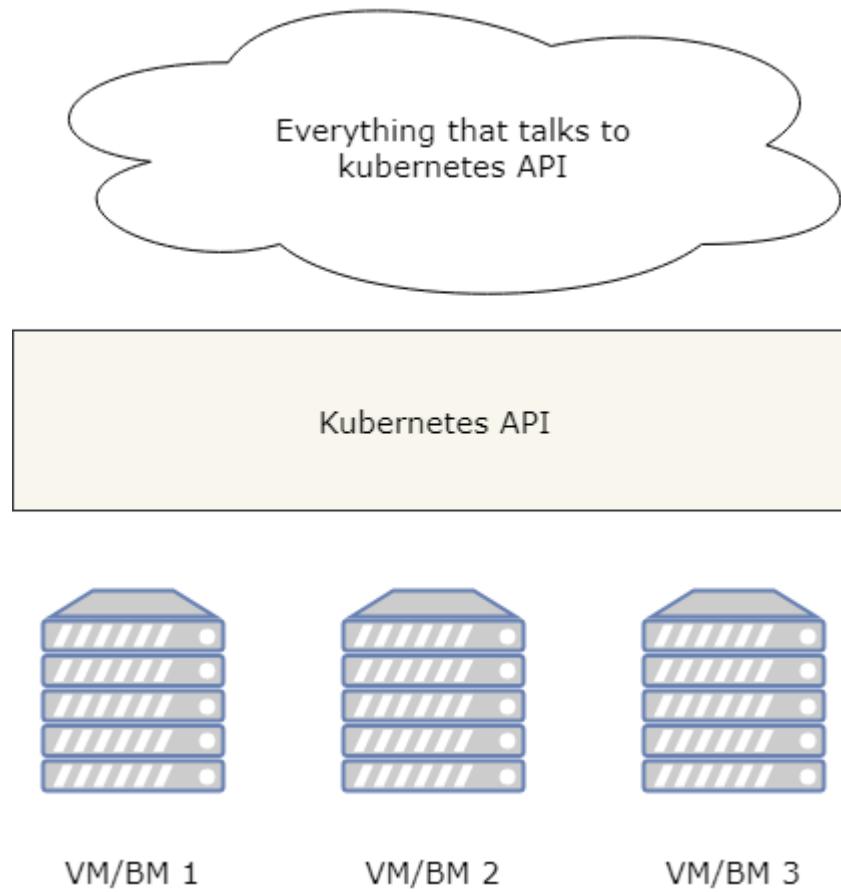
- Choose one preferred option (or more)
- if ssh - try accessing it
- minikube
  - on ubuntu run: *minikube start --memory 8192*
  - locally run: *minikube start*
  - run: *minikube status* host, kubelet and api should be running
- docker
  - Check in UI if kubernetes is running

# K8S - main concepts

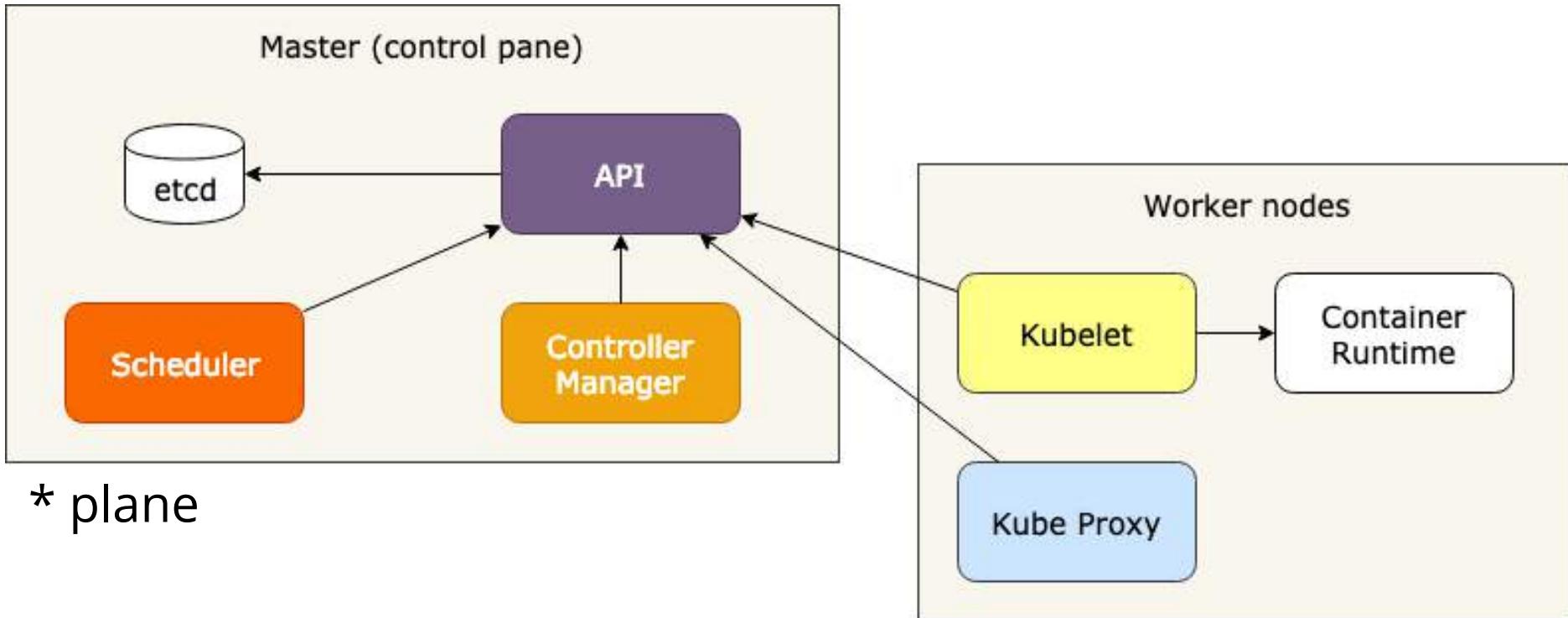
## Ubiquitous Language



# Birds-eye view



# Main blocks of K8S



# Master Node

- **API** - RESTful API used by all components to communicate
- **Scheduler** - assign nodes to deployable components
- **Controller Manager** - performs cluster-wide operations like, registering new nodes, or replicating components
- **etcd** - key-value store keeping the current state of the application and configuration. Only API talks to etcd.

# Worker Node

- **Kubelet** - is responsible for everything that runs on worker node, this is the only component that needs to be executed as binary on node, all others might be deployed over kubernetes as part of kubernetes (🍺)
- **Kube Proxy** - responsible for providing network access to containers in the cluster
- **Container Runtime** - shim over different container technologies like Docker, CRI-O or containerd

# One API to rule them all

- Core of K8S is REST API
- This API is consumed by
  - Dashboard (UI) - clickable
  - kubectl (CLI) - imperative
  - and all k8s main components (scheduler, controller manager) and more

# Pre-Basics

Things we should understand to understand basics :)

# kubectl

- CLI client for K8S Api Server
- The basic tool to work with k8s from
- Allows imperative communications:
  - operations on live objects
  - no history of operations
  - no easy way to revert changes
  - we say what to do, not what state we want

# kubectl

## installation

```
# linux
sudo snap install kubectl --classic

# mac
brew install kubernetes-cli

# win chocolaty
choco install kubernetes-cli

# win powershell
Install-Script -Name install-kubectl -Scope CurrentUser -Force
install-kubectl.ps1 [-DownloadLocation <path>]

# docker desktop (mac, win)
# already contains it

# other options
# with info how to add autocomplete to shell
https://kubernetes.io/docs/tasks/tools/install-kubectl/
```

# kubectl

## verification

```
$ kubectl version -o json
{
  "clientVersion": {
    "major": "1",
    "minor": "10",
    "gitVersion": "v1.10.3",
    "gitCommit": "2bba0127d85d5a46ab4b778548be28623b32d0b0",
    "gitTreeState": "clean",
    "buildDate": "2018-05-21T09:17:39Z",
    "goVersion": "go1.9.3",
    "compiler": "gc",
    "platform": "windows/amd64"
  },
  "serverVersion": {
    "major": "1",
    "minor": "11",
    "gitVersion": "v1.11.3",
    "gitCommit": "a4529464e4629c21224b3d52edfe0ea91b072862",
    "gitTreeState": "clean",
    "buildDate": "2018-09-09T17:53:03Z",
    "goVersion": "go1.10.3",
    "compiler": "gc",
    "platform": "linux/amd64"
  }
}
```

**We need kubectl working so  
make sure that version check did  
work for you**

# yaml

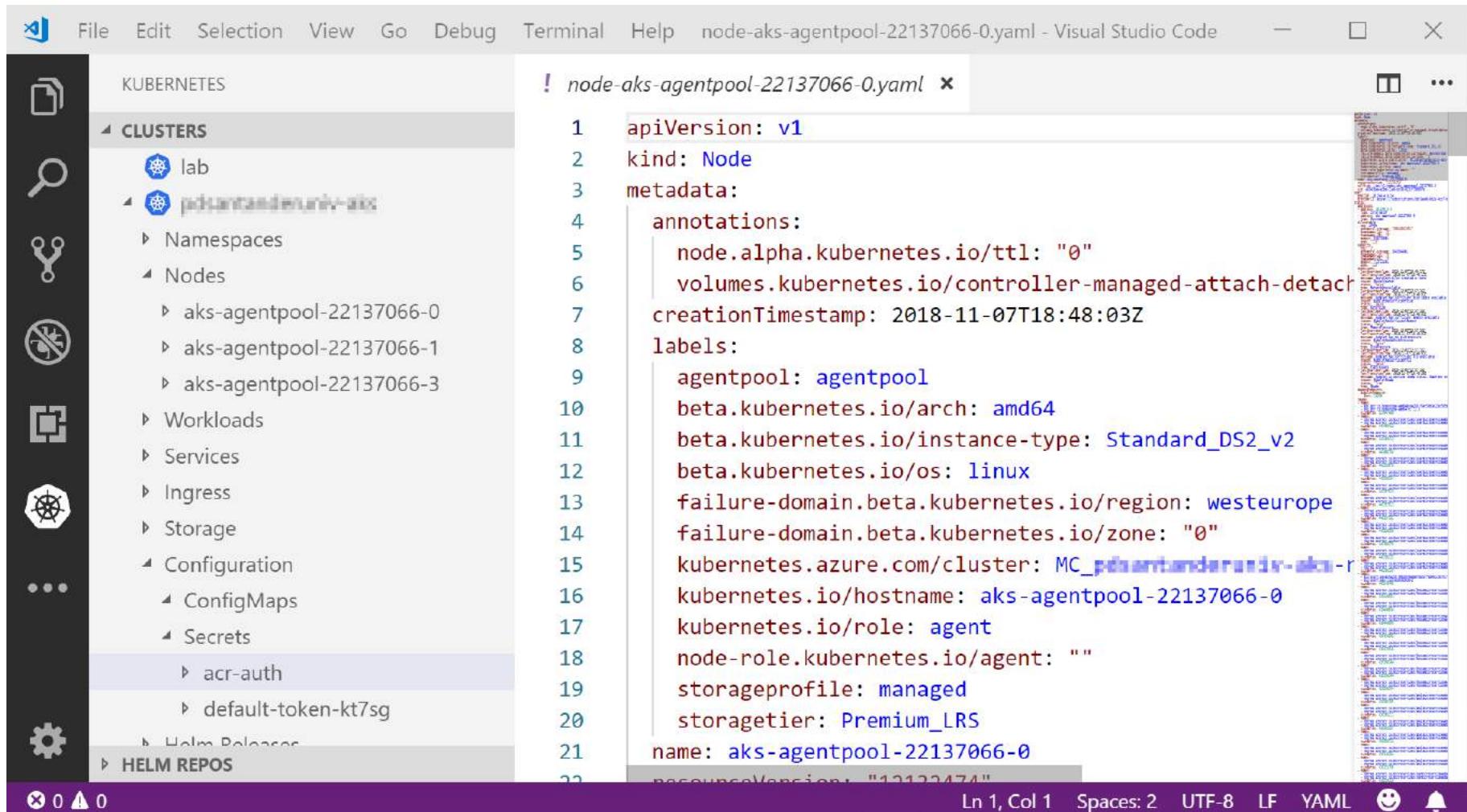
- Human-readable data serialization language
- Whitespace indentation is used to denote structure
- Tab characters are never allowed as indentation
- "Missing ;" (in yaml missing space) with yaml escalates quickly and is hard to fix
- Allows declarative communication
  - we are describing a desire state, k8s will do all steps needed to achieve this

# yaml

## tools

- Amazingly notepad is quite good
- Whatever tool suites you
- Visual Studio Code have two nice plugins:
  - [Kubernetes](#)
  - [YAML Support by Red Hat](#)

# demo - Code with plugins



The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Debug, Terminal, Help, node-aks-agentpool-22137066-0.yaml - Visual Studio Code
- Sidebar:** KUBERNETES, CLUSTERS (lab, akshanderson/aks), Namespaces, Nodes (aks-agentpool-22137066-0, aks-agentpool-22137066-1, aks-agentpool-22137066-3), Workloads, Services, Ingress, Storage, Configuration (ConfigMaps, Secrets), acr-auth, default-token-kt7sg, Helm Releases, HELM REPOS.
- Code Editor:** A YAML file titled "node-aks-agentpool-22137066-0.yaml". The content is as follows:

```
! node-aks-agentpool-22137066-0.yaml x
1 apiVersion: v1
2 kind: Node
3 metadata:
4   annotations:
5     node.alpha.kubernetes.io/ttl: "0"
6     volumes.kubernetes.io/controller-managed-attach-detach: "true"
7   creationTimestamp: 2018-11-07T18:48:03Z
8   labels:
9     agentpool: agentpool
10    beta.kubernetes.io/arch: amd64
11    beta.kubernetes.io/instance-type: Standard_DS2_v2
12    beta.kubernetes.io/os: linux
13    failure-domain.beta.kubernetes.io/region: westeurope
14    failure-domain.beta.kubernetes.io/zone: "0"
15    kubernetes.azure.com/cluster: MC_pdhantunderhand-v-aks-rg
16    kubernetes.io/hostname: aks-agentpool-22137066-0
17    kubernetes.io/role: agent
18    node-role.kubernetes.io/agent: ""
19    storageprofile: managed
20    storagetier: Premium_LRS
21    name: aks-agentpool-22137066-0
22    resourceVersion: "12122474"
```

The status bar at the bottom shows: Ln 1, Col 1 Spaces: 2 UTF-8 LF YAML. There are also icons for file operations (New, Open, Save, Find, Replace) and status indicators (01).

# VS Code Plugins

The screenshot shows a Mac OS X browser window displaying the Visual Studio Code Marketplace page for the "Kubernetes" extension. The extension icon is a blue octagon with a white steering wheel. The title is "Kubernetes" with a "Preview" badge. It's developed by "Microsoft" with 100,691 installs, 306,313 downloads, and a 5-star rating from 12 reviews. A green "Install" button is prominent, along with a link for "Trouble Installing?". Below the main card, there's a navigation bar with "Overview" (underlined), "Q & A", and "Rating & Review".

**Visual Studio Code Kubernetes Tools**

**Categories**

- Other
- Azure

**Tags**

- azure
- debugger
- k8s
- kubernetes

**Resources**

- [Repository](#)
- [License](#)
- [Changelog](#)
- [Download Extension](#)

An extension for developers building applications to run in Kubernetes clusters, and for DevOps staff troubleshooting Kubernetes applications. Features include:

- View your clusters in an explorer tree view, and drill into workloads, services, pods and nodes.
- Browse Helm repos and install charts into your Kubernetes cluster.
- Intellisense for Kubernetes resources and Helm charts and templates.
- Edit Kubernetes resource manifests and apply them to your cluster.
- Build and run containers in your cluster from Dockerfiles in your project.
- View diffs of a resource's current state against the resource manifest in your Git repo
- Easily check out the Git commit corresponding to a deployed application.

# VS Code Plugins

The screenshot shows a web browser window on the Visual Studio Code Marketplace at [marketplace.visualstudio.com](https://marketplace.visualstudio.com). The page is for the "YAML" plugin by Red Hat. The plugin has 1,058,133 installs and 4,711,744 downloads, with a rating of 4.5 stars from 18 reviews. It is listed as free. A large red circular icon on the left contains a white file icon with the letters "YML". Below the title are two buttons: a green "Install" button and a link to "Trouble Installing?". Below the main header, there are tabs for "Overview" (which is selected), "Q & A", and "Rating & Review".

**YAML Language Support by Red Hat**

Provides comprehensive YAML Language support to [Visual Studio Code](#), via the `yaml-language-server`, with built-in Kubernetes and Kedge syntax support.

**Features**

A screenshot of the VS Code interface shows a code editor with three files open: `kubernetes.yml`, `appveyor.yml`, and `to_format.yaml`. The `kubernetes.yml` file is the active tab, displaying YAML configuration for a Kubernetes API version v1. The sidebar on the right lists "User Settings" and other settings. A tooltip or callout box highlights the "User Settings" option.

**Tags**

- autocompletion
- yaml

**Resources**

- Issues
- Repository
- License
- Changelog
- Download Extensi

# VS Code Plugins

The screenshot shows a web browser window on a Mac OS X system displaying the Visual Studio Marketplace. The URL in the address bar is `marketplace.visualstudio.com`. The main content is the page for the "Docker" extension by Microsoft. The extension has 1,806,085 installs and 8,870,532 downloads, with a rating of 4.5 stars from 30 reviews. It is listed as "Free". There are "Install" and "Trouble Installing?" buttons. Below the main card, there are tabs for "Overview", "Q & A", and "Rating & Review", with "Rating & Review" being underlined. To the right, there are sections for "Categories" (Programming Languages), "Tags" (compose, dockerfile, ignore), and "Resources" (Repository, Homepage, License). The "Docker Support for Visual Studio Code" section details the extension's version (v0.5.2), number of installs (1.81M), and Azure Pipelines status (succeeded). A description explains how it integrates Docker support into VS Code. A bulleted list of features includes automatic file generation, syntax highlighting, linting, and command palette integration.

YAML - Visual Studio Marketplace   Docker - Visual Studio Marketplace

**Docker**

Microsoft | 1,806,085 installs | 8,870,532 downloads | ★★★★★ (30) | Free

[Install](#) [Trouble Installing?](#)

Overview Q & A Rating & Review

**Docker Support for Visual Studio Code**

Visual Studio Marketplace v0.5.2 installs 1.81M Azure Pipelines succeeded

The Docker extension makes it easy to build, manage and deploy containerized applications from Visual Studio Code, for example:

- Automatic Dockerfile, docker-compose.yml, and .dockerignore file generation (Press F1 and search for Docker: Add Docker files to Workspace)
- Syntax highlighting, hover tips, IntelliSense (completions) for docker-compose.yml and Dockerfile files
- Linting (errors and warnings) for Dockerfile files
- Command Palette (F1) integration for the most common Docker commands (for example docker build,

**Categories**  
Programming Languages

**Tags**  
compose dockerfile ignore

**Resources**  
Repository Homepage License

# VS Code Plugins

Visual Studio | Marketplace      Sign in      

Visual Studio Code > Other > SSH FS      New to Visual Studio Code? Get it now.

---



**SSH FS**  
Kelvin Schoofs | 45,531 installs | 136,450 downloads | ★★★★★ (43) | Free  
File system provider using SSH  
[Install](#)   [Trouble Installing?](#)

---

[Overview](#)   [Q & A](#)   [Rating & Review](#)

### SSH FS



[GitHub](#)   [VS Marketplace v1.13.0](#)   [Donate](#)   [PayPal](#)

This extension makes use of the new FileSystemProvider, added in version 1.23.0 of Visual Studio Code. It allows "mounting" a remote folder over SSH as a local Workspace folder.

### Categories

Other

### Tags

json

### Resources

[Issues](#)  
[Repository](#)  
[License](#)  
[Download Extension](#)

### Project Details

[View on GitHub](#)   [View on Marketplace](#)

**Make sure you have a way to  
write YAML files - w/ or w/o  
intellisense**

# Dumpster

our amazing application



# Dumpster

- Simple node application
- Our goal is not to *write* app, but **learn** about kubernetes
- App is containerized and available on docker hub
- Default image we will be working with  
**gutek/dumpster:v1**
- We will be playing with different tags - v1, v2 etc
- Source for v1 is in repo

# Dumpster

## endpoints

- **/data** - list files in directory /var/data
- **/env** - list all environment variables
- **/\*** - general endpoint printing version number and host name

# Dumpster

## code

```
const http = require('http');
const os = require('os');
const fs = require('fs');

const port = 8080;
const path = '/var/data'

console.log('dumpster server starting...');

const dataHandler = (request, response) => {}
const envHandler = (request, response) => {}
const generalHandler = (request, response) => {}

const requestHandler = (request, response) => {
    console.log(`received request from ...`);

    response.writeHead(200);
    if(request.url === '/data') {}

    if(request.url === '/env') { }

    console.log('handling general request');
    generalHandler(request, response);
};

const server = http.createServer(requestHandler);
server.listen(port, err => {});
```

# Dumpster

## docker

```
FROM node:11
ADD app.js /app.js

ENTRYPOINT ["node", "app.js"]
```

# PKAD

alternativly

- Instead of simple dumpster, we can use pkad, application that shows on UI a lot of kubernetes functionality
- All demos and sample code are using by default dumpster, but you can always change this to:

**poznajkubernetes/pkad**

# Dashboard

API walkthrough with definitions

# Accessing Dashboard

## minikube

```
# we can open dashboard by
minikube dashboard
👉 Enabling dashboard ...
🤔 Verifying dashboard health ...
🚀 Launching proxy ...
🤔 Verifying proxy health ...
🍕 Opening http://127.0.0.1:51160/api/v1/namespaces/
    kube-system/services/http:kubernetes-dashboard:/proxy/
    in your default browser...
```

```
# or
kubectl proxy

# and then opening url manually:
http://127.0.0.1:8001/api/v1/namespaces/
    kube-system/services/http:kubernetes-dashboard:/proxy/
```

# Accessing Dashboard

docker

```
# execute
kubectl create -f demos/00-dashboard/install.yaml

# then
kubectl proxy

# wait a moment :)

# and finally open browser (beware of spaces):
http://localhost:8001/api/v1/namespaces/
    kubernetes-dashboard/services/
https:kubernetes-dashboard:/proxy/
```

# Dashboard

 **kubernetes** Search + CREATE

≡ Overview

default ▾

Workloads

Overview

Workloads

Cron Jobs

Daemon Sets

Deployments

Jobs

Pods

Replica Sets

Replication Controllers

Stateful Sets

Discovery and Load Balancing

Ingresses

Services

Config and Storage

Config Maps

Persistent Volume Claims

Secrets

Settings

About

Workloads Statuses

100.00% Deployments

100.00% Pods

100.00% Replica Sets

Deployments

Name	Labels	Pods	Age	Images
hello-minikube	run: hello-minikube	1 / 1	9 days	k8s.gcr.io/echoserver:1.10

Pods

Name	Node	Status	Restarts	Age	⋮
hello-minikube-7c77b68cff-pzmzr	minikube	Running	1	9 days	⋮

Replica Sets

Name	Labels	Pods	Age	Images
------	--------	------	-----	--------

node-template-hach-2722621700

# demo - create pod from UI

Instructions:

- create pod with name **dumpster-pod**
- image: **gutek/dumpster:v1**
- service: **Internal**
- Port: 80 -> 8080

# Dashboard & definitions

The screenshot shows the Kubernetes Dashboard interface. On the left, there is a sidebar with navigation links: Overview, Workloads, Deployments, Jobs, PersistentVolumeClaims, ReplicationControllers, Services, Config and Storage, Config Maps, Persistent Volume Claims, Secrets, Settings, and About. The 'Overview' link is highlighted with a blue bar.

The main content area has two tabs: 'Pods' and 'Replica Sets'. A large black arrow points down to the 'Pods' table, which lists a single pod named 'hello-minikube-7c77b68cff-pzmzr'. The table columns include Name, Node, Status, Restarts, and Age. The pod status is 'Running' with 1 restart and an age of 9 days.

The 'Replica Sets' tab is also visible below the 'Pods' table.

Name	Node	Status	Restarts	Age
hello-minikube-7c77b68cff-pzmzr	minikube	Running	1	9 days

# demo - pod definition and logs

Instructions:

- open pod
- take a look at logs
- take a look at pods definition under edit button (note, we will come back to it later on)

# exercise - Access Pod

Instructions:

- use pod created during demo of name **dumpster-pod**
- pod is web application that is accessible on url path: /
- might help: port mapping is 80 -> 8080
- find a way to open application

# discussion - Access Pod

- Did you manage to access pod?
- How did you achieve that?
- How we can make pod available from Internet using ui?

let's do it together

# demo - accessing pod

- Accessing pod
  - using a proxy (pod and internal)  
`(http://localhost:8001/api/v1/namespaces/default/pods/http:POD_NAME:8080/proxy/ )`
  - using a terminal in UI

# Dashboard & definitions

A ReplicaSet:

- build on top of *replication controller*
- is a type of *Controller*, it controls Pods :)
- ensures that a specified number of pod replicas are running at any given time
- should be managed by a Deployment

Name	Labels	Pods	Age	Images
hello-minikube-7c77b08cff	pod-template-hash: 3733624799 run: hello-minikube	1 / 1	9 days	k8s.gcr.io/echoserver:1.10

# exercise - Delete POD

Instructions:

- open our app in browser (using one of the ways described before)
- check hostname
- delete **dumpster-pod-\*** pod
- keep attention on the pod name (especially on \* part) and pod age
- might required browser refresh

# Dashboard & definitions

The screenshot shows the Kubernetes Dashboard's Overview page. At the top, there are three green circular metrics indicating 100.00% for Deployments, Pods, and Replica Sets. Below these are three tabs: Overview (selected), Workloads, Cron Jobs, Daemon Sets, Deployments, Jobs, Pods, Replica Sets, and Replication Controllers. The Deployments tab is selected. A table titled 'Deployments' lists one item: 'hello-minikube'. The 'hello-minikube' row is highlighted with a red border and a black arrow points to the 'Name' column. The table has columns for Name, Labels, Pods, Age, and Images.

Name	Labels	Pods	Age	Images
hello-minikube	run: hello-minikube	1 / 1	9 days	k8s.gcr.io/echoserver:1.10

## A Deployment:

- 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.

# Dashboard & definitions

## A Service:

- is an abstraction which defines a logical set of Pods and a policy by which to access them
- the set of Pods targeted by a Service is (usually) determined by a Label Selector
- makes pod available externally (adds external access)

Overview

Name	Labels	Cluster IP	Internal endpoints	External endpoints	Age	⋮
frontend	run: hello-minikube	10.102.202.19	front-minikube:8080 TCP front-minikube:32121 TCP	-	a minute	⋮
hello-minikube	run: hello-minikube	10.100.52.103	hello-minikube:8080 TCP hello-minikube:30395 TCP	-	a day	⋮
kubernetes	component: apiserver provider: kubernetes	10.96.0.1	kubernetes:443 TCP kubernetes:0 TCP	-	a day	⋮

Workloads

Cron Jobs

Daemon Sets

Deployments

Jobs

Pods

Replica Sets

Replication Controllers

Stateful Sets

Discovery and Load Balancing

Ingresses

Services

Config and Storage

Secrets

Name	Type	Age	⋮
default-token-h7gm7	kubernetes.io/service-account-token	a day	⋮

# exercise - Scale up & down

Instructions:

- Scale up deployment
- Check number of pods and their names
- Access pod using service abstraction
- Check hostname
- Delete Pod that was shown
- Refresh your browser
- Optional: Scale down

```
while true; do curl http://...; done
#powershell
while ($true) {curl http://... -UseBasicParsing}
#powershell 6+
while ($true) {curl http://... }
```

# demo - exposing

- Exposing pod
  - Internal
  - External
  - There are two more ways, we will learn about them during a course

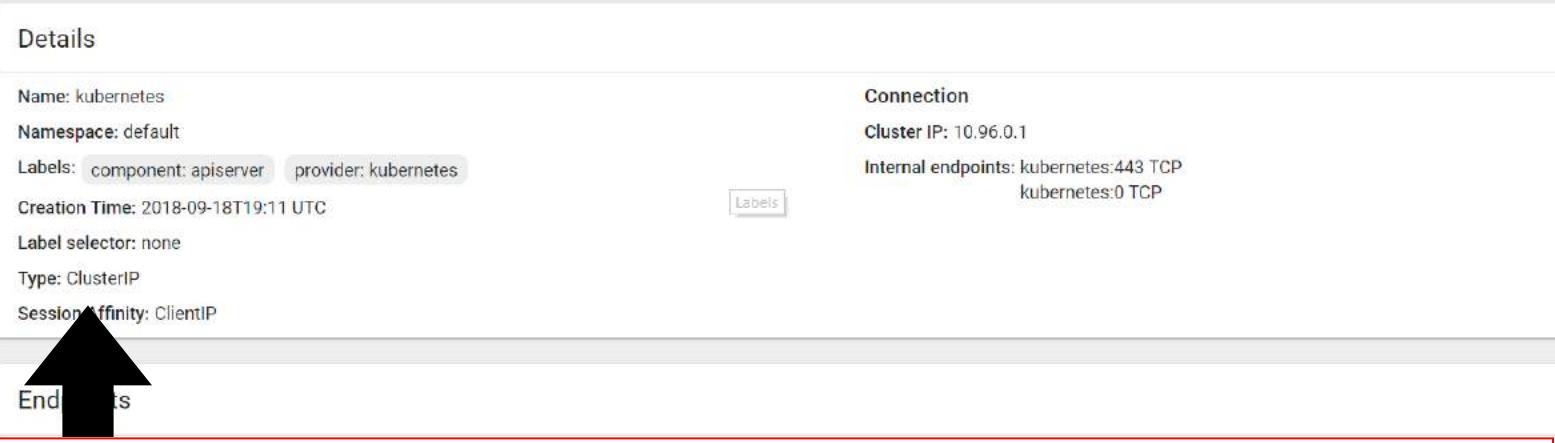
# Dashboard & definitions

The screenshot shows the Kubernetes Dashboard interface. On the left, there's a sidebar with navigation links: Cluster, Namespaces, Nodes, Persistent Volumes, Roles, Storage Classes, Namespace (set to default), Overview, and Workloads. The main area has a breadcrumb trail: Discovery and load balancing > Services > kubernetes. At the top right are buttons for + CREATE, EDIT, and DELETE. The central part displays the 'Details' for the service 'kubernetes'. It shows the Name: kubernetes, Namespace: default, Labels (component: apiserver, provider: kubernetes), Creation Time: 2018-09-18T19:11 UTC, Label selector: none, Type: ClusterIP, and Session Affinity: ClientIP. To the right, under 'Connection', it lists the Cluster IP: 10.96.0.1 and Internal endpoints: kubernetes:443 TCP, kubernetes:0 TCP. A large black arrow points upwards from the 'Workloads' section on the left towards the 'Type: ClusterIP' line in the service details.

A Service Type can be:

- ClusterIP - exposes the service on a cluster-internal IP. This is the default ServiceType
- NodePort - Exposes the service on each Node's IP at a static port (the NodePort). Access to service can be done with `http(s)://NodeExternalIP:NodePort`
- continue next slide :)

# Dashboard & definitions



The screenshot shows the Kubernetes Dashboard interface. On the left, there's a sidebar with options like Cluster, Namespaces, Nodes, Persistent Volumes, Roles, and Storage Classes. Below that is a dropdown for Namespace set to 'default'. At the top, there's a search bar and a '+ CREATE' button. The main area shows a 'Discovery and load balancing > Services > kubernetes' path. On the right, there's an 'EDIT' and a 'DELETE' button. The central part displays 'Details' for the 'kubernetes' service, including its Name, Namespace, Labels, Creation Time, Label selector, Type (ClusterIP), and Session Affinity (ClientIP). It also lists its Connection information: Cluster IP (10.96.0.1) and Internal endpoints (kubernetes:443 TCP, kubernetes:0 TCP). The 'Endpoints' section is partially visible at the bottom.

A **Service Type** can be:

- LoadBalancer - Exposes the service externally using a **cloud provider's** load balancer
- 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

# Dashboard & definitions

## A ConfigMap:

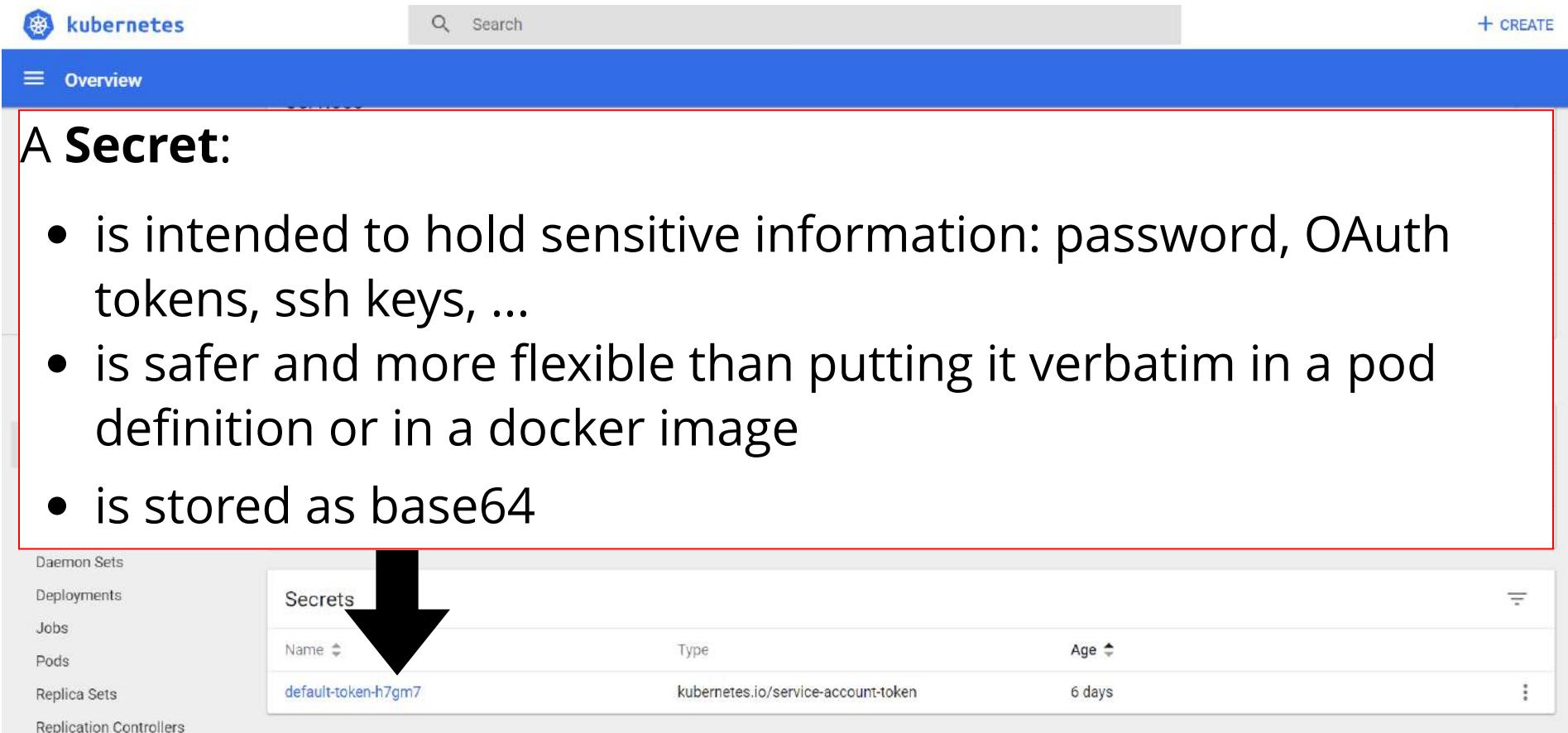
- binds configuration files, command-line arguments, environment variables, port numbers, and other configuration artifacts to your Pods' containers and system components at runtime.
- is useful for storing and sharing *non-sensitive*, unencrypted configuration information

The screenshot shows a Kubernetes dashboard interface. On the left, there is a sidebar with navigation links: Overview (which is currently selected), Workloads, Cron Jobs, Daemon Sets, Deployments, Jobs, Pods, Replica Sets, and Replication Controllers. The main area displays two tables. The top table is titled 'Config Maps' and has columns for Name, Labels, and Age. It shows one entry: 'simple-cm' with an age of '22 seconds'. A large black arrow points from the text 'A ConfigMap:' in the previous slide to the 'simple-cm' entry in this table. The bottom table is titled 'Secrets' and has columns for Name, Type, and Age. It shows one entry: 'default-token-h7gm7' with a type of 'kubernetes.io/service-account-token' and an age of '6 days'.

Name	Labels	Age
simple-cm	-	22 seconds

Name	Type	Age
default-token-h7gm7	kubernetes.io/service-account-token	6 days

# Dashboard & definitions



A Secret:

- is intended to hold sensitive information: password, OAuth tokens, ssh keys, ...
- is safer and more flexible than putting it verbatim in a pod definition or in a docker image
- is stored as base64

The screenshot shows the Kubernetes Dashboard with a red box highlighting the "Secrets" section. A large black arrow points down to the "default-token-h7gm7" row in the table.

Name	Type	Age	⋮
default-token-h7gm7	kubernetes.io/service-account-token	6 days	⋮

- Daemon Sets
- Deployments
- Jobs
- Pods
- Replica Sets
- Replication Controllers

# exercise - discover a secret

Instructions:

- Try to find value of **token** secret
- When you are ready raise your hand

# Dashboard & definitions

The screenshot shows the Kubernetes dashboard interface. On the left, there's a sidebar with navigation links: Cluster, Namespaces, Nodes, Persistent Volumes, Roles, Storage Classes, and Namespace (set to default). The main area has a blue header bar with 'Discovery and load balancing > Services > hello-minikube'. Below the header, there's a search bar and a 'CREATE' button. The main content area is titled 'Details' and contains the following information for the service 'hello-minikube':

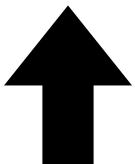
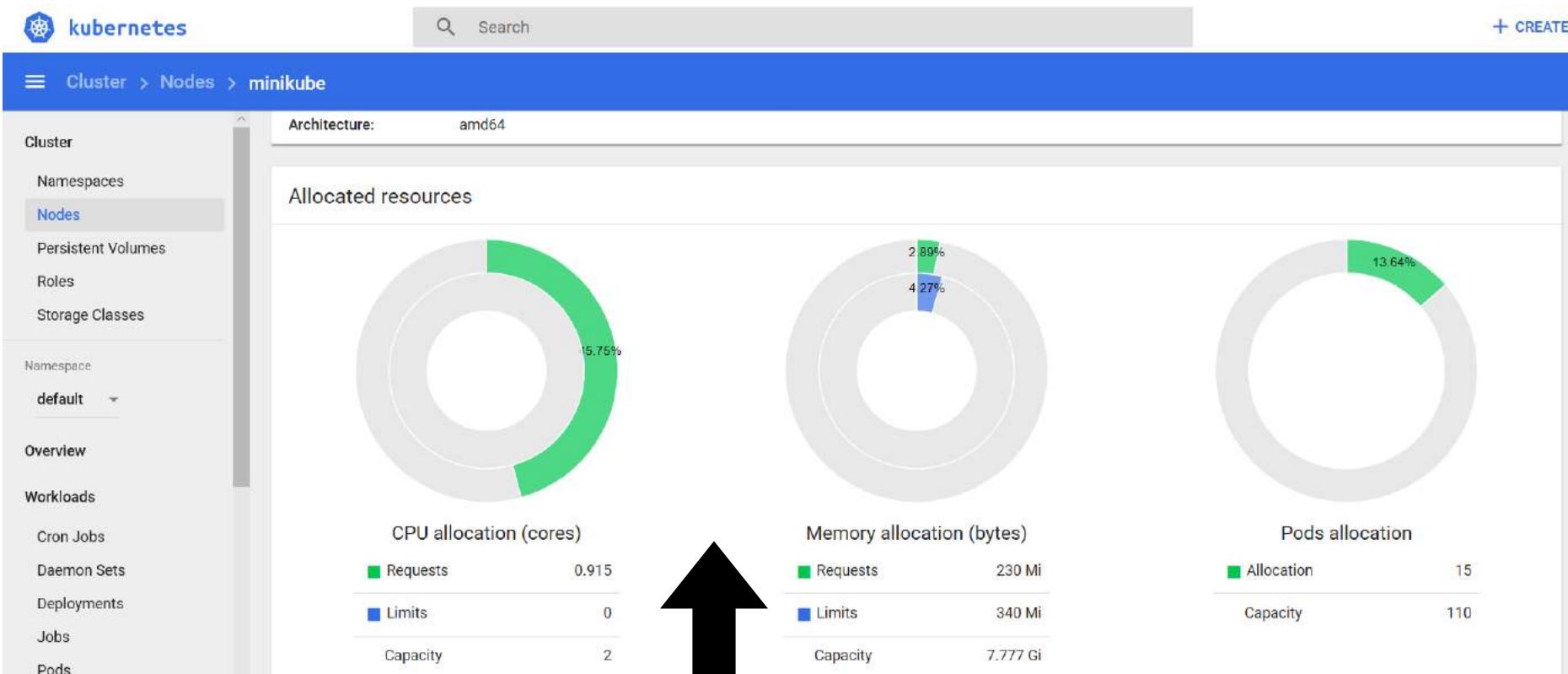
Details		Connection
Name:	hello-minikube	Cluster IP: 10.100.52.103
Namespace:	default	Internal endpoints: hello-minikube:8080 TCP hello-minikube:30395 TCP
Labels:	run: hello-minikube	
Creation Time:	2018-09-18T19:16 UTC	
Label selector:	run: hello-minikube	
Type:	NodePort	
Session Affinity:	None	

A large black arrow points upwards from the 'Labels' row towards the 'A Label:' section below.

## A Label:

- is key/value pairs that are attached to objects, such as pods, services, ...
- do not directly imply semantics
- is human friendly description used for search, grouping, etc

# Dashboard & definitions



Bare metal underhood :

- CPU stats
- memory stats
- allocations

# dashboard - quick trick

```
# execute in shell
kubectl edit svc kubernetes-dashboard -n kubernetes-dashboard

# !!!"type: " is important in below replace!!
# replace -> type: ClusterIP
# with -> type: NodePort

# execute
kubectl get svc kubernetes-dashboard -n kubernetes-dashboard
NAME           TYPE      ... PORT(S)
kubernetes-dashboard   NodePort   ... 80:31611/TCP

# remember port 31611 (your's might be different)
# if you have docker, open http://127.0.0.1:31611
# if you have minikube
minikube ip

# then open http://IP_FROM_ABOVE:31611
```

# Common commands



kubectl

commands

option

**Open shell/cmd and play along :)**

# kubectl config

- allows working with more than one k8s clusters

```
# get list of clusters
$ kubectl config get-contexts
CURRENT   NAME                   CLUSTER          AUTHINFO
          lab                  lab
*         aaa-aks              aaav-aks        kube-admin-local
          u4k8scluster        u4k8scluster  clusterUser_aaa-aks-rg_aaa-aks
                                         clusterUser_unity4_u4k8scluster

# set cluster as current
$ kubectl config use-context lab
Switched to context "lab".

# get ~/.kube/config file with some "redacted" values
$ kubectl config view
```

# kubectl cluster-info

- information about cluster services and roles

```
$ kubectl cluster-info
Kubernetes master is running at https://www
Heapster is running at https://www/api/v1/namespaces/kube-system/services/heapster/proxy
KubeDNS is running at https://www/api/v1/namespaces/kube-system/services/kube-dns:prox
kubernetes-dashboard is running at https://www/api/v1/namespaces/kube-system/services/kube
Metrics-server is running at https://www/api/v1/namespaces/kube-system/services/https:metr
```

To further debug and diagnose cluster problems, use '`kubectl cluster-info dump`'.

# kubectl explain

- explain what specific resource is
- nice help, not really useful later on

```
# what is node
kubectl explain node

# what is pod
kubectl explain pod
kubectl explain pod.spec
kubectl explain pod.spec.containers
kubectl explain pod.spec.containers.command

# what is service
kubectl explain service

# what is RESOURCE_NAME
kubectl explain RESOURCE_NAME
```

# kubectl get|describe

- display details of resource

```
# get status information about all nodes
$ kubectl get node
NAME                  STATUS   ROLES      AGE      VERSION
NODE_NAME-0           Ready    agent      25d     v1.11.3
NODE_NAME-1           Ready    agent      25d     v1.11.3
NODE_NAME-3           Ready    agent      25d     v1.11.3

# get status information about single node
$ kubectl get node NODE_NAME
NAME                  STATUS   ROLES      AGE      VERSION
NODE_NAME             Ready    agent      25d     v1.11.3

# get status information about single RESOURCE (pod, service...)
$ kubectl get RESOURCE RESOURCE_NAME

# get details about node
$ kubectl describe node NAME

# get node yaml representation
$ kubectl get node NAME -o yaml
```

# demo - get yaml for UI pod

Instructions:

- using kubectl
- and describe
- get yaml for pod deployed

# Basics

Things we should know

**Employees Must Wash  
Hands Before Returning  
to Work**



# Cleanup

## how to clean up our mess

```
# list all resources in cluster
kubectl get all

# remove everything that deployment had created including:
# replica set and pods
kubectl delete deployments NAME

# remove pod of specific name
kubectl delete pod NAME

# remove service
kubectl delete service NAME

# remove all pods
kubectl delete pod --all

# remove all deployments
kubectl delete deployments --all

# hint: aliases
alias kdel='kubectl delete po,svc,deploy,pvc,pv,cm,secret,rs --all'
kdel

# powershell
function kube-delete-all {kubectl delete pod,service,cm,deployments --all}
```

# exercise - clean up

Instructions:

- Clean your instance
- Use commands from prev slide (*kubectl delete ...*)
- Try do it step by step without *--all* argument
- Hint: Do you remember:
  - `kubectl get deployments`
  - `kubectl get pods`

# discussion - clean up

- Do you feel powerful?
- Is it dangerous?
- did you deleted *kubernetes* service? if not, why?
- *Can I do it at home?*

# Pod

let's look at a structure from the dashboard

```
# /seed/dumpster-pod-extract.yaml

kind: Pod
apiVersion: v1
metadata:
  name: dumpster-6d6f7864d9-tzr8j
  generateName: dumpster-6d6f7864d9-
  namespace: default
  selfLink: /api/v1/namespaces/default/pods/dumpster-6d6f7864d9-tzr8j
  uid: 028fee83-140b-4c95-80ca-9e04990d6d52
  resourceVersion: '86671'
  creationTimestamp: '2021-02-23T20:25:10Z'
  labels:
    k8s-app: dumpster
    pod-template-hash: 6d6f7864d9
  ownerReferences:
    - apiVersion: apps/v1
      kind: ReplicaSet
      name: dumpster-6d6f7864d9
      uid: 8e0a3762-e4f8-477d-865a-121d6fac6cde
      controller: true
      blockOwnerDeletion: true
  managedFields:
    - manager: kube-controller-manager
      operation: Update
      apiVersion: v1
      time: '2021-02-23T20:25:10Z'
      fieldsType: FieldsV1
```

# Pod

## yaml structure

```
apiVersion: v1
kind: Pod
metadata:
  name: POD_NAME
  labels:
    key: value
spec:
  containers:
    - name: CONTAINER_NAME
      image: CONTAINER_IMAGE
      ports:
        - containerPort: 80
```

- **apiVersion** - RESTful verison of API (part of path)
- **kind** - object type
- **metadata** - information about the object
- **labels** - key-value object identification
- **spec** - desire state of the object
- **containers** - containers specification
- **name** - name of the container
- **image** - docker/rkt image to use
- **ports** - ports that container exposes, completely OPTIONAL!

# Pod

## deployment

```
# create pod from file
kubectl create -f pod.yaml

# create pod and save base configuration so it can be used later by apply
kubectl create -f pod.yaml --save-config=true

# create pod or update if exists
kubectl apply -f pod.yaml

# replace pod
kubectl replace -f pod.yaml

# get the current pod definition
kubectl get pod NAME -o yaml|json
```

# exercise - create and get yaml

Instructions:

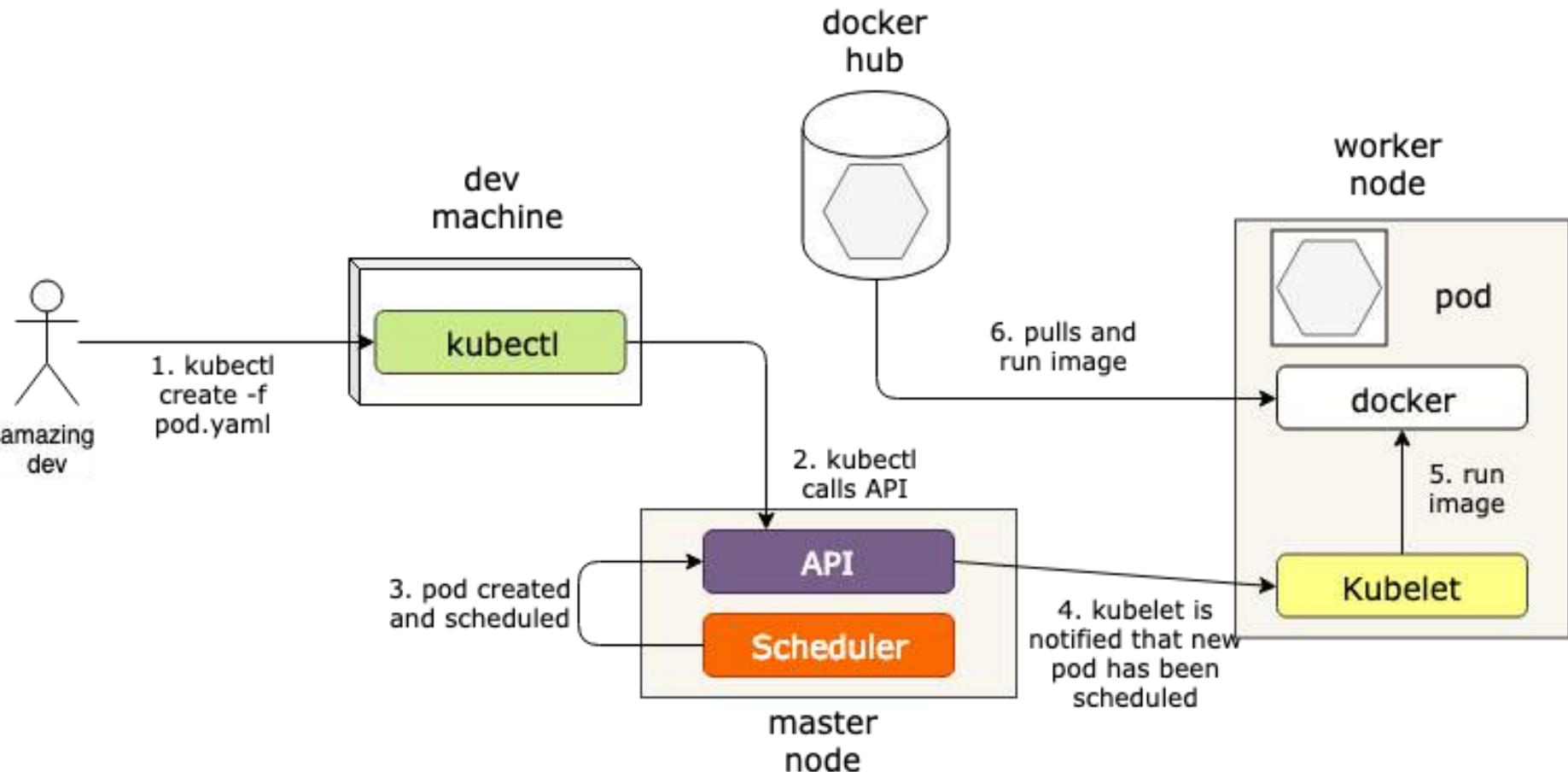
- open and view **pod.yaml**
- deploy pod **pod.yaml** using **create** command
- use *kubectl* to get yaml of deployed pod
- open yaml and compare it to **pod.yaml**

# discussion - create and get yaml

- What differences are in files?
- Why these files vary?

# Pod

what exactly happens when *create* is executed?



# exercise - apply and get yaml

Instructions:

- deploy pod **pod.yaml** using **apply**
- answer: what was your intention?

# discussion - apply and get yaml

- Where you able to **apply** pod?
- What issue did you encounter?
- How you would like it to behave in this situation?

# Pod

## delete

```
# deletes pod described in file  
$ kubectl delete -f pod.yaml
```

```
# delete pod of name  
$ kubectl delete pod POD_NAME
```

# exercise - compare yaml

Instructions:

- clear environment using delete cmd
- deploy **pod.yaml** using **create**
- get yaml of deployed pod - save it under **create-pod.yaml**
- delete pod (whatever option you choose)
- deploy pod **pod.yaml** using **apply**
- get yaml of deployed pod - save it under **apply-pod.yaml**
- compare two yaml's - the create and apply one
- what's the difference?

# discussion - compare yaml

- What options did you use to delete pod?
  - Did you use downloaded yaml file to delete?
- What are the differences between two files?

# Pod

## how to access pod from the command line

```
# allowing access to the pod by url http://127.0.0.1:8888
kubectl port-forward pod/NAME 8888:pod_container_port

# accessing container in pod/running app in container
kubectl exec -it POD_NAME bash
kubectl exec -it POD_NAME -c container_name bash
kubectl exec POD_NAME -- cmd -p1 -p2
kubectl exec POD_NAME cmd

# and
kubectl proxy
```

# exercise - access pod

Instructions:

- access pod using all three options
- with each option, get result from query / on pod app

# discussion - access pod

- Did you manage to do all three options?
- What options had been most difficult?
- What options do you prefer?

# Pod

## best practices

- never use **:latest** in image specification
- pod's are rarely used as single deployment units
- provide **containerPort** even thought is not needed
- always specify **label**
- add **annotations**
- always specify a **container name**
- when using **create** always add **--save-config=true**  
*(nice explanation why and when to use it)*

# Pods are ephemeral



# Pets VS. cattle



# Labels

a way to describe resource

- Used across k8s
- One resource can have multiple labels
- Labels are used in selectors to find all resources that meet criteria, for instance:
  - find frontend of version 1
  - find all pods responsible for handling back-end requests

# Labels

## example

```
apiVersion: v1
kind: RESOURCE_TYPE
metadata:
  name: MY_RESOURCE_NAME
  labels:
    version: 1.0.0.0
    app: new-facebook
    tier: backend
```

---

```
apiVersion: v1
kind: RESOURCE_TYPE
metadata:
  name: MY_RESOURCE_NAME
  labels:
    version: 1.0.0.0
    app: new-facebook
    tier: frontend
```

# Labels

## kubectl

```
# get all pods with label tier of value backend
kubectl get pods -l tier=backend

# get all resources with label tier of value frontend
kubectl get all -l tier=frontend

# delete all pods with label app of value api
kubectl delete pods -l app=api

# delete all pods and services with label app of value api
kubectl delete pods,services -l app=api

# get all pods and services with label app
kubectl get pods,services -l app

# get all pods and services with label app and display its value in column
kubectl get pods,services -L app

# display labels while listing resources
kubectl get pods --show-labels
```

# exercise - add labels to pod

Instructions:

- add following labels to **pod.yaml**:
  - version, app and tier - whatever value
  - and **workshop=true**
- apply **pod.yaml**
- check if labels had been applied using kubectl
- remove all resources with **workshop=true** label

# discussion - add lables to pod

- for what we can use labels?
- did your label worked? did you needed to change it?
- did you manage to remove all resource of given label?

# demo - adding labels manually

- We can also add labels manually when needed
- For instance, we want to remove all except one pod having a specific label
- Or we want to remove label that pod have so we can investigate it and let kubernetes handle rest for us

```
# adding
kubectl label pod NAME key1=value1 key2=value2

# changing
kubectl label pod NAME key1=newValue --overwrite
```

from now on you can clean your  
k8s by deleting everything under  
**workshop=true**  
label

# Services

exposing pods

- Services allow exposing pods that match specific label selectors to:
  - internal network - **ClusterIP**
  - publicly available port number - **NodePort**
  - publicly available IP address - **LoadBalancer**
- Also allows:
  - mapping external service as "local" one - **ExternalName**

# Services

## template

```
apiVersion: v1
kind: Service
metadata:
  name: name-svc
  labels:
    tier: backend
    app: api
spec:
  type: TYPE
  selector:
    pod-label-key: pod-label-val
  ports:
  - name: http
    port: 8080
    targetPort: 80
```

- **kind** - we are working with service
- **spec** - desire state of the object
- **type** - type of the service - ClusterIP (default), NodePort, LoadBalancer, ExternalName
- **selector** - label selector of pods that should be exposed by given service
- **ports** - mapping between service port and pod ports

# Services

## ports

```
ports:  
- name: http  
  protocol: TCP  
  port: 8080  
  targetPort: 80|api
```

- **name** - name that we can give to current port, needed if more than one port defined
- **protocol** - protocol used in communication: TCP (default), UDP & SCTP
- **port** - port number on service to which client (user) connects to
- **targetPort** - port on the pod to which connects to. we can use names instead of numbers allowing us to dynamically change ports on pods or to have different port on each pod

# Services

## kubectl

```
# get services
```

```
kubectl get service
```

```
kubectl get services
```

```
kubectl get svc
```

NAMESPACE	NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
default	kubernetes	ClusterIP	10.0.0.1	<none>	443/TCP	97d
default	my-svc	NodePort	10.0.18.147	<none>	8080:31238/TCP	97d
default	my-second-svc	NodePort	10.0.45.201	<none>	8080:32245/TCP	97d

```
# removes service
```

```
kubectl delete svc NAME
```

```
# create/update service
```

```
kubectl apply -f service.yaml
```

# exercise - ClusterIP

Instructions:

- apply **pod.yaml**
- try to create ClusterIP service for **pod.yaml**
- you can use **template.yaml** to help with structure
- get internal IP address of deployed service
- try to access service

# discussion - ClusterIP

- what are the differences between service and pod names?
- what ClusterIP is doing for you that you could not achieve without it?
- were you able to access ClusterIP service?

# demo - accessing ClusterIP

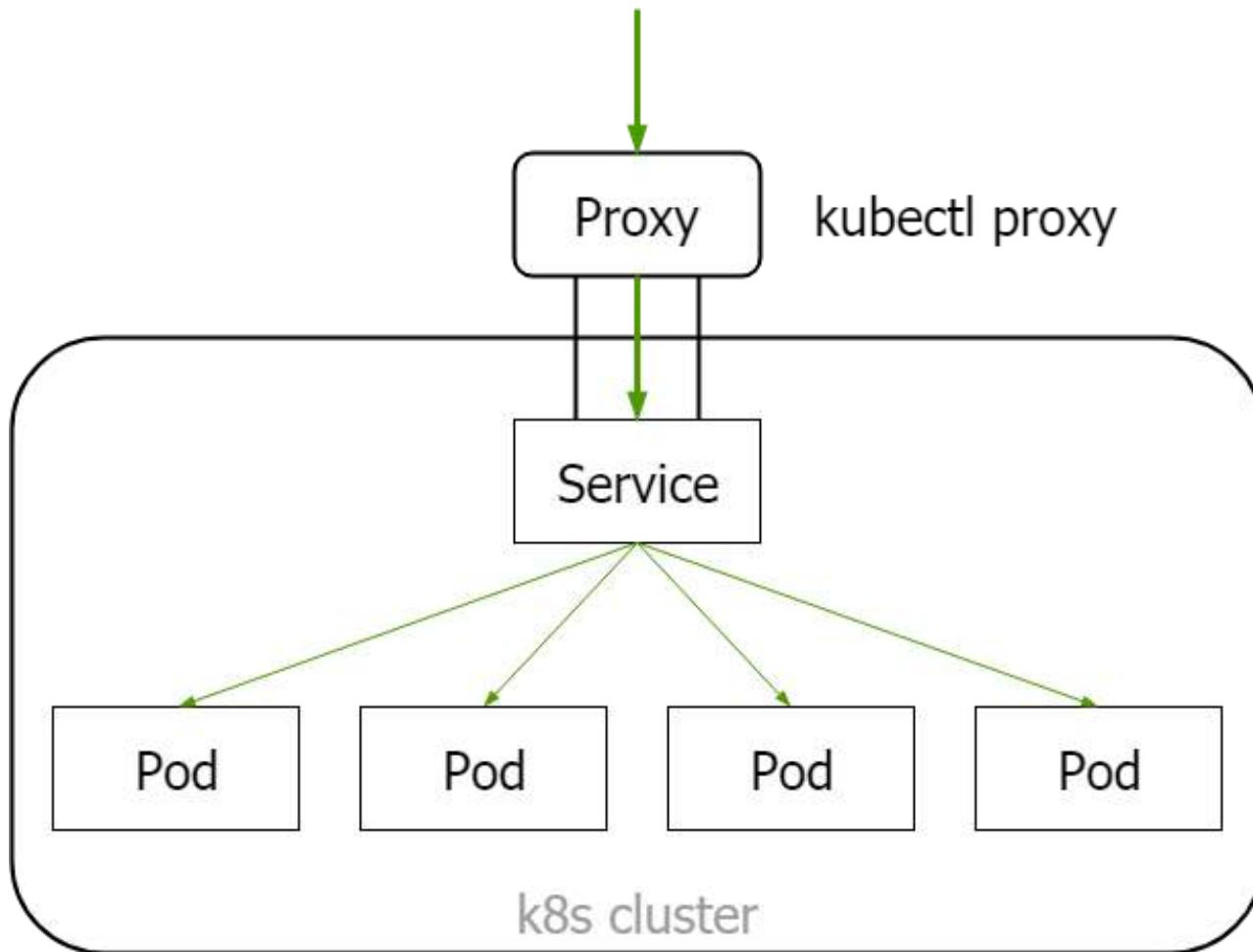
We have three options to access ClusterIP service

- *kubectl port-forward*
- *kubectl proxy*
- *kubectl exec POD\_NAME -- curl http://service\_name*

# Services

ClusterIP

Traffic/Requests



# exercise - ping ClusterIP

Instructions:

- access pod that ClusterIP expose as service
- try to *ping* ClusterIP IP address
- try to *curl* ClusterIP IP address

# discussion - ping ClusterIP

- Were you able to *ping* ClusterIP?
- Were you able to *curl* ClusterIP?
- Do you have an idea why this happens?

# exercise - Env Vars ClusterIP

Instructions:

- clear all pods and services
- apply **pod.yaml**
- Create ClusterIP service for **pod.yaml**, you can use **template.yaml** to help with structure
- execute */env* endpoint of exposed pod (by doing this over ClusterIP or pod) - save result
- delete and create pod again (do not delete service)
- compare results of */env* endpoint

# discussion - Env Vars ClusterIP

- What did you notice?
- What are differences between `/env` endpoints?
- Have you notice something *interesting*?

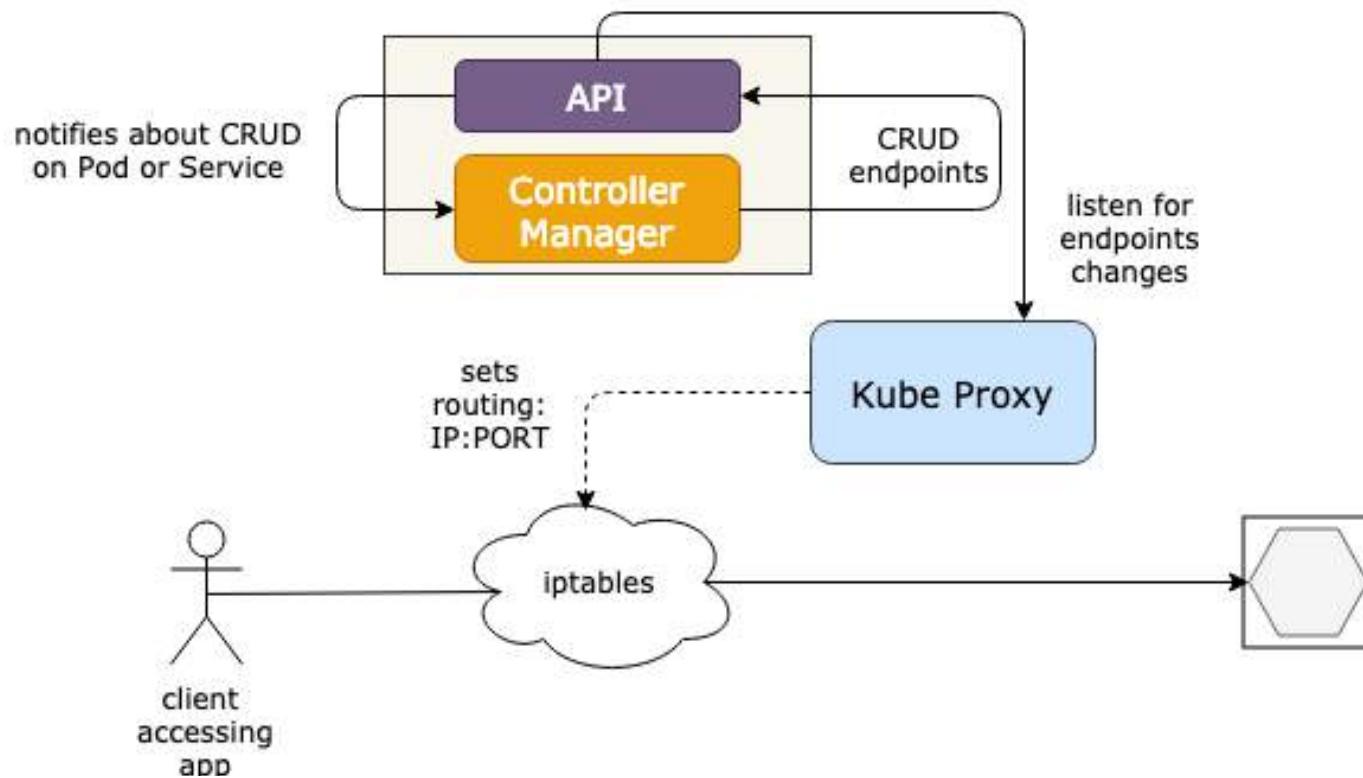
# exercise - ClusterIP pods

Instructions:

- clear all pods and services and apply **pod-one.yaml**
- create ClusterIP service of name *dumpster*
- execute *kubectl describe svc dumpster* and save results
- apply **pod-two.yaml**
- execute *kubectl describe svc dumpster* and compare results with previous one
- what are endpoints?
- can you figure this out?

# discussion - ClusterIP pods

- What was different?
- What are endpoints?



# Services

## ClusterIP

### Pros

- Internally visible service by IP and DNS
- Base for all service types

### Cons

- Not available externally
- Should not be exposed publicly - don't depend on kubectl proxy

# Services

## ClusterIP

When to use it:

- Internal dashboard
- Internal proxy to external service - i.e. all microservices call this service to get to external service
- giving a *name* to set of pods that act as a backend

# exercise - NodePort

Instructions:

- apply **pod.yaml**
- create NodePort service for **pod.yaml**
- do it manually - if you have issues, take a look at template from prev exercise
- get yaml of deployed service, notice differences between your's yaml and deployed service
- access exposed NodePort service

# discussion - NodePort

- did you specify nodePort in settings?
- what are differences between ClusterIP and NodePort?
- how did you access your's service?
- when do you think this is a good solution?

# demo - accessing NodePort

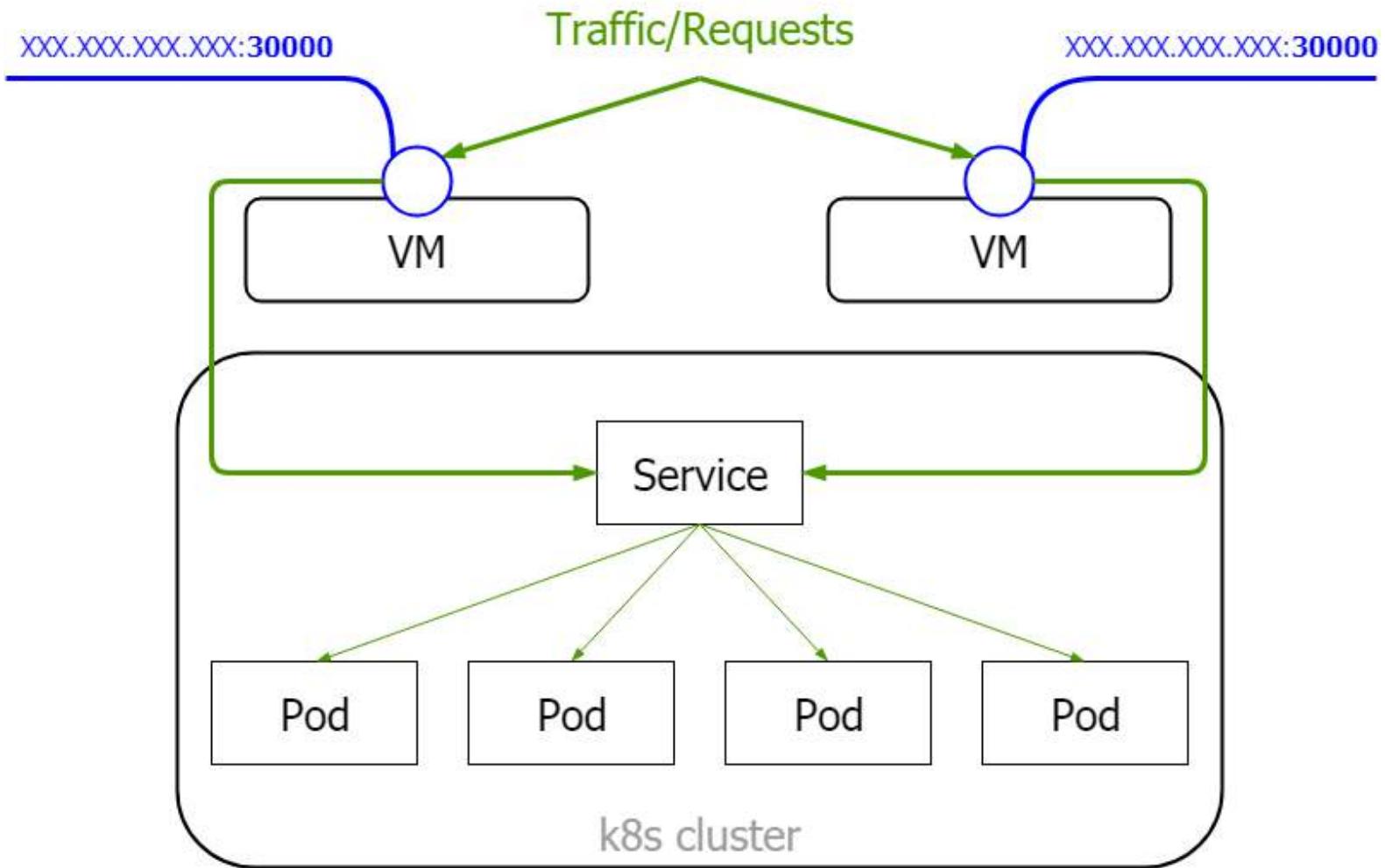
```
# get services
kubectl get services
NAME            TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)           AGE
dumpster        NodePort    10.101.205.44   <none>          80:30111/TCP    5h51m
kubernetes      ClusterIP  10.96.0.1       <none>          443/TCP         4d20h

# on minikube
minikube ip
192.168.99.101

# open http://192.168.99.101:30111
# on docker open http://localhost:30111
```

# Services

## NodePort



based on Ahmet Alp Balkan drawings

# Services

## NodePort

### Pros

- Service available from all nodes
- One Service per port

### Cons

- One Service per port
- Limited number of available ports ([30 000 - 32 767](#))
- Address depends on node IP - if that changes, all clients needs to be updated

# Services

## NodePort

When to use it:

- for development purpose
- as a part of bigger solution - for instance, most cloud services required service of type NodePort when using ingress

# Services

## LoadBalancer

- LoadBalancer is not part of k8s
- K8s provides an interface that specific provider should implement
- LoadBalancer is different depending on the cloud we are using and how we do a setup on-prem
- When we request LoadBalancer from k8s, K8s ask the provider to create one externally

# exercise - LoadBalancer

Instructions:

- apply **pod.yaml**
- create LoadBalancer service for **pod.yaml**
- you can use **template.yaml** to help with structure
- access exposed LoadBalancer

# discussion - LoadBalancer

- Did you deployed LoadBalancer?
- How did you know it was deployed?
- Where you able to access LoadBalancer?
- How did you do it?

# demo - accessing LoadBalancer

Most important: **LoadBalancer -> NodePort -> ClusterIP**, so NodePort is ClusterIP, and LoadBalancer is NodePort and ClusterIP

```
# get services
kubectl get services
NAME      TYPE            CLUSTER-IP      EXTERNAL-IP      PORT(S)        AGE
dumpster   LoadBalancer    10.98.49.32    <pending>       80:30803/TCP  4s

# 30803 its NodePort so we have access service same
# was as we did with NodePort

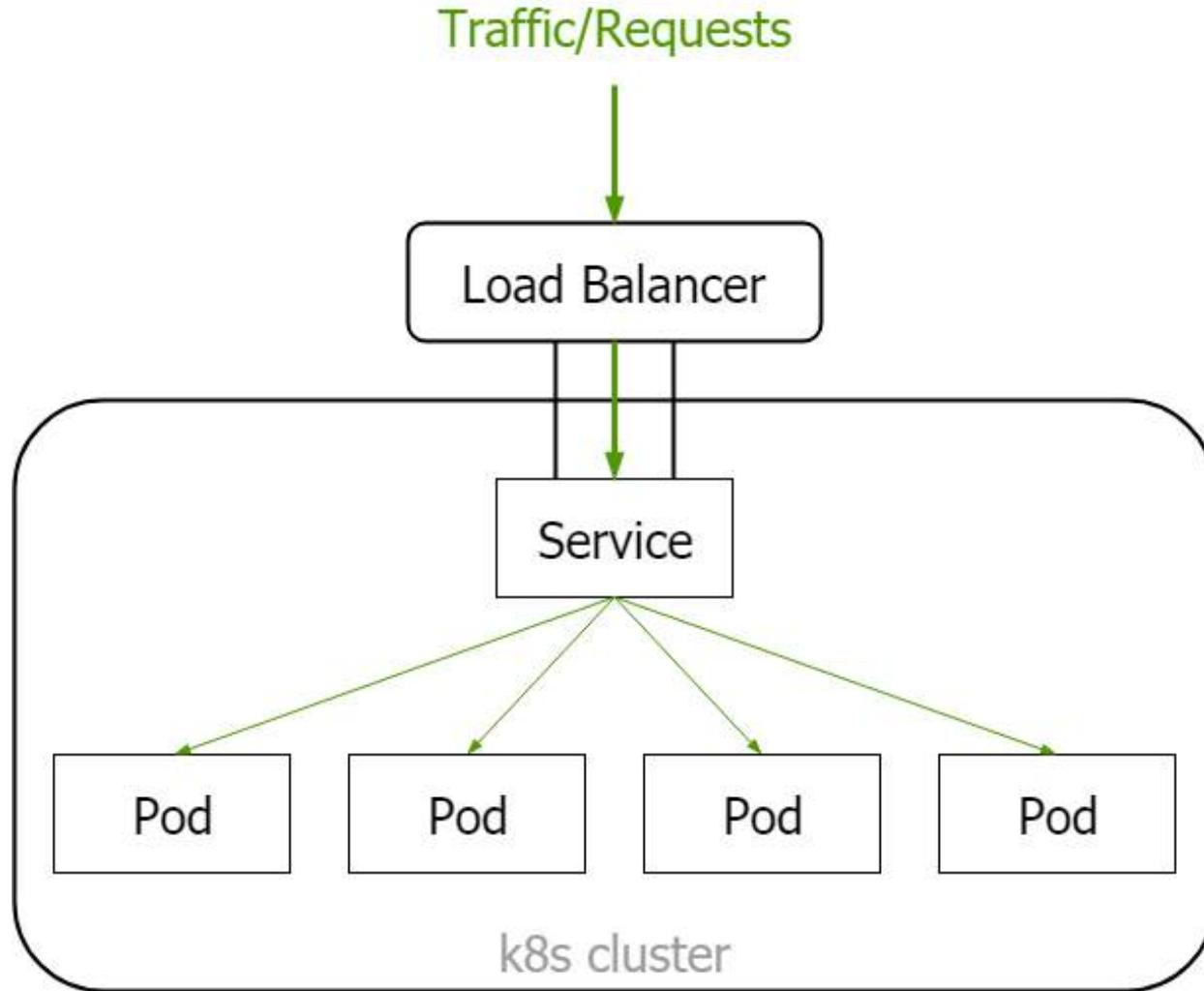
# to make it work in minikube, in new shell tab execute:
minikube tunnel

kubectl get services dumpster
NAME      TYPE            CLUSTER-IP      EXTERNAL-IP      PORT(S)        AGE
dumpster   LoadBalancer    10.98.49.32    10.98.49.32    80:30803/TCP  108s

# now you can access your's service: http://10.98.49.32
```

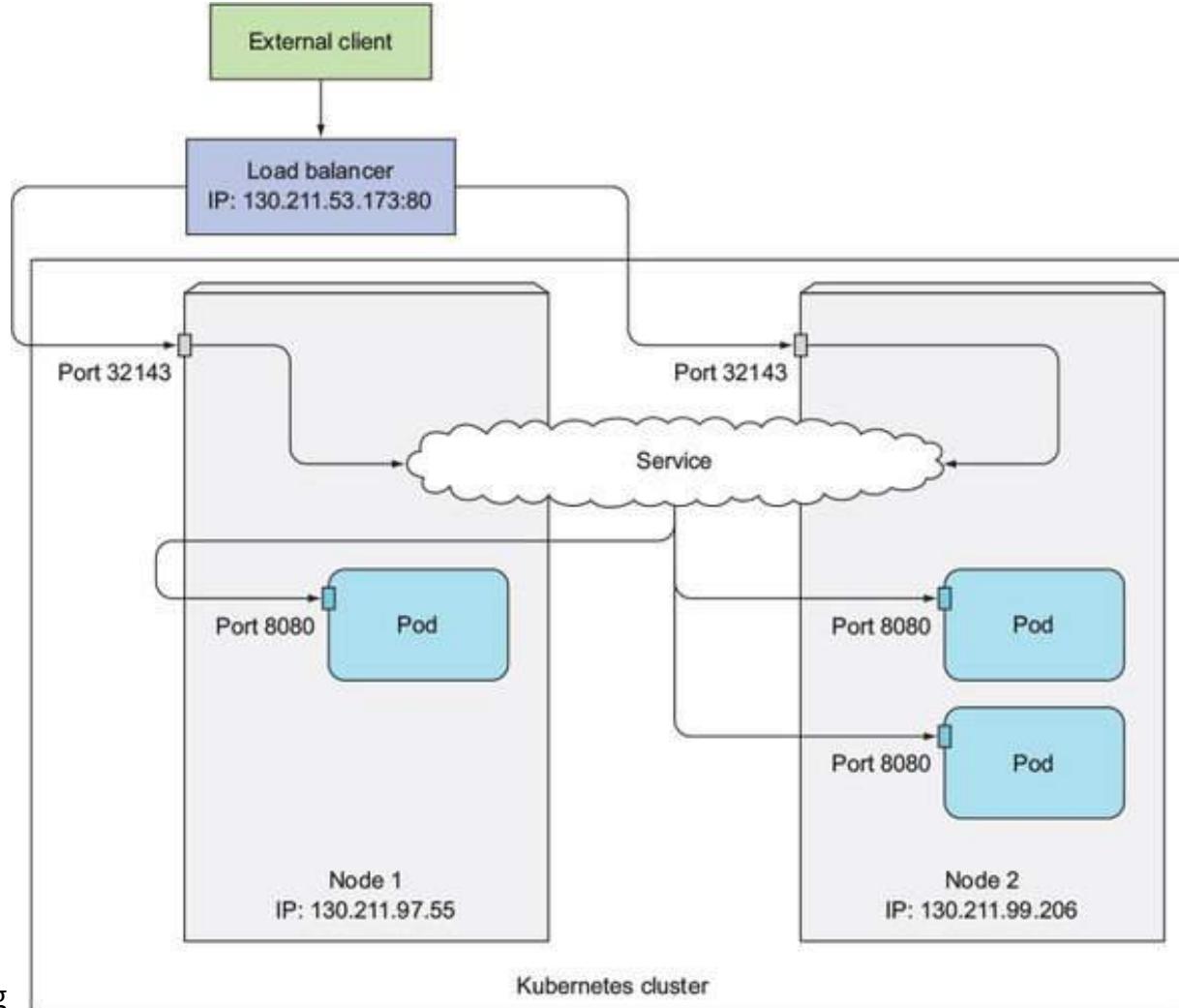
# Services

LoadBalancer



# Services

## LoadBalancer



# Services

## Load Balancer

### Pros

- "Cloud" native - on-prem available
- No filtering - all traffic is routed to service

### Cons

- One Load Balancer per Service!
- Might be expensive
- No filtering - all traffic is routed to service

# Services

## Load Balancer

when to use it:

- When we want to expose service externally
- Define IP/dns for our service

# Services

## Load Balancer

how it's normally done?

- If we have all the money in the world, we can create as many LoadBalancers as we want
- Generally, we have one LoadBalancer per our solution
- LoadBalancer is used with Ingress
- Ingress is responsible for managing k8s routes, and external LoadBalancer manage access to ingress
- So when we will talk about Ingress? **soooon :)**

when in doubt, remember you  
can always clean everything:

**workshop=true**

An aerial nighttime photograph of a bustling shipping port. The foreground and middle ground are dominated by a massive stack of shipping containers in various colors, including red, blue, white, and yellow. These containers are organized into several levels of storage. In the background, large industrial gantry cranes with bright lights are visible, some with their booms extended over the containers. The overall scene conveys a sense of high-volume global trade and logistics.

# Deployments

# Deployments

a way to create MANY pods ;)

- Allows to manage process of deploying scalable solution to k8s
- Declaratively manage **ReplicaSet**
- Allows by default two kind of deployments:
  - **recreate** - replace existing version with new version, called Recreate in k8s
  - **ramped** - slowly replace version 1 with version 2, default in k8s and called RollingUpdate.
- if our deployment update fails, the previous version keep's running!

# Deployments

## template

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: DEP_NAME
spec:
  replicas: 3
  selector:
    matchLabels:
      app: app_name
  template:
    metadata:
      labels:
        app: app_name
    spec:
      containers:
        - name: pod_name
          image: gutek/dumpster:v1
          ports:
            - containerPort: 8080
            ...
...
```

- **replicas** - number of "template" instances
- **selector** - how do we know how many instances there are
- **template** - pod yaml w/o apiVersion and kind. you can have only one template per deployment

# Deployments

## kubectl

```
# create/update deployment
kubectl apply -f deployment.yaml

# create/update deployment and keep track of operations
kubectl apply -f deployment.yaml --record=true

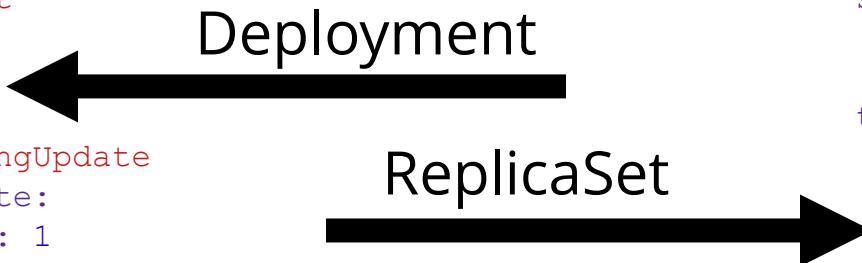
# get all deployments
kubectl get deployments
kubectl get deployment
kubectl get deploy

# get replica sets
kubectl get rs
```

# Deployments vs ReplicaSet

- There is no **vs**
- Deployments manage **ReplicaSets**
- **ReplicaSets** manage pods replications
- There is **Replication Controller vs ReplicaSet**

```
apiVersion: apps/v1
kind: Deployment
metadata:
spec:
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge : 1
      maxUnavailable: 1
```



```
replicas: 3
selector:
  matchLabels:
    app: app_name
template:
  metadata:
    labels:
      app: app_name
spec:
  containers:
  - name: pod_name
    image: gutek/dumpster:0
    ports:
    - containerPort: 80
  ...
```

# Deployments

## naming

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dumpster-deploy
spec:
  replicas: 3
  selector:
    matchLabels:
      app: dumpster-naming-01
  template:
    metadata:
      labels:
        app: dumpster-naming-01
    spec:
      containers:
        - name: dumpster-naming-containe
          image: gutek/dumpster:v1
```

```
kubectl get all
NAME
pod/dumpster-deploy-585cdbd6cf-7zm27
pod/dumpster-deploy-585cdbd6cf-nkbhv
pod/dumpster-deploy-585cdbd6cf-w6cg8
```

```
NAME
deployment.apps/dumpster-deploy
NAME
replicaset.apps/dumpster-deploy-585cdbd6cf
```

- **metadata.name** - a name of deployment and base name for **ReplicaSet** and **Pod**
- **dumpster-naming-01** - it's just a label
- **dumpster-naming-container** - it's just a name of the container in Pod(s), each separate Pod might have same name of container
- **585cdbd6cf** - hash of spec.template, for instance execute *kubectl get pods --show-labels*
- **585cdbd6cf-7zm27** - random generated, we can achieve this with *generateName*

# exercise - deploy pods

Instructions:

- modify & use **deploy.yaml**
- make sure that deploy creates **3 replicas**
- deploy should work on **workshop=true** and **app=api** label
- make sure that 3 instances have been deployed
- download deployed yaml and check deployment strategy
- access web application deployed by this **deploy.yaml**

# discussion - deploy pods

- did you manage to deploy 3 instances?
- were you able to access web application?
  - how?
  - why not?
- what deployment strategy had been set?

# Deployments

available strategies - RollingUpdate (default)

```
apiVersion: apps/v1
kind: Deployment
...
spec:
  replicas: 3

  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge : 1
      maxUnavailable: 1

  selector:
    matchLabels:
      app: app_name
template:
  ...
...
```

- **strategy.type** - defines the current strategy type selected. RollingUpdate is the default one, other option is Recreate
- **rollingUpdate.maxUnavailable** - how many pods might be unavailable in the current moment
- **rollingUpdate.maxSurge** - how many pods might be created over a total number of replicas

# Deployments

## available strategies - Recreate

```
apiVersion: apps/v1
kind: Deployment
...
spec:
  replicas: 3

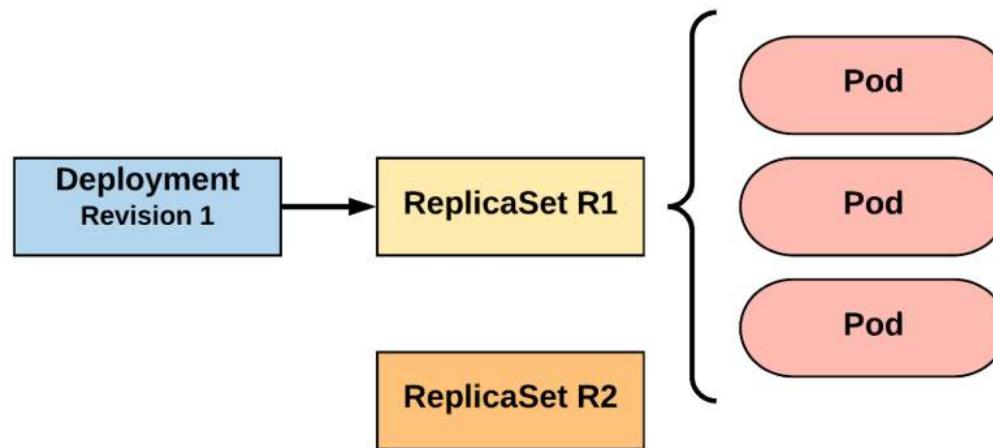
  strategy:
    type: Recreate

  selector:
    matchLabels:
      app: app_name
template:
  ...

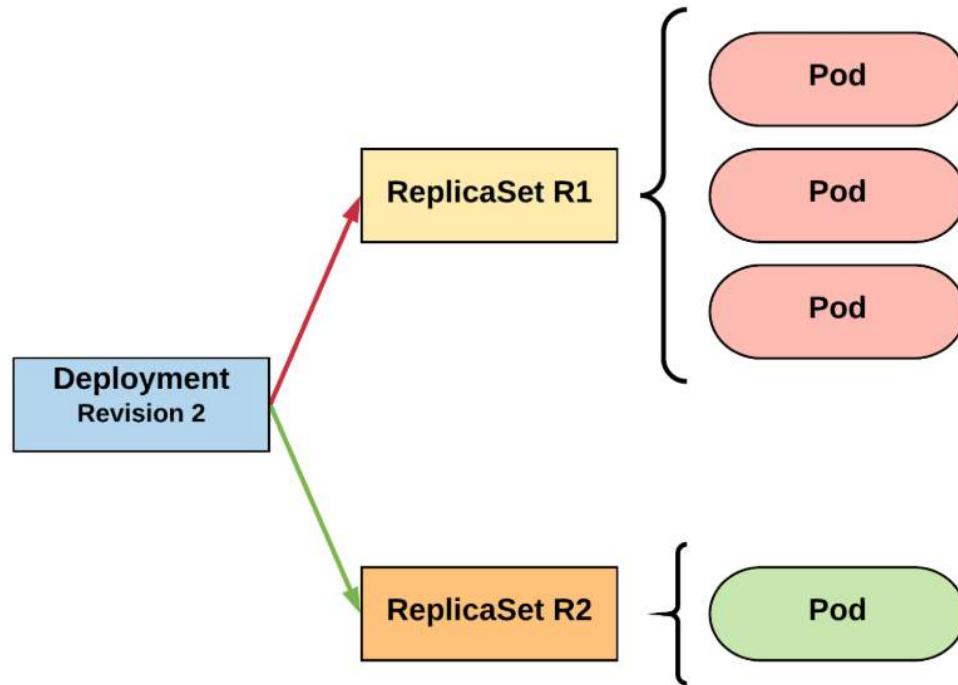
```

- **strategy.type** - when recreate is selected all current pods are removed before new one are installed

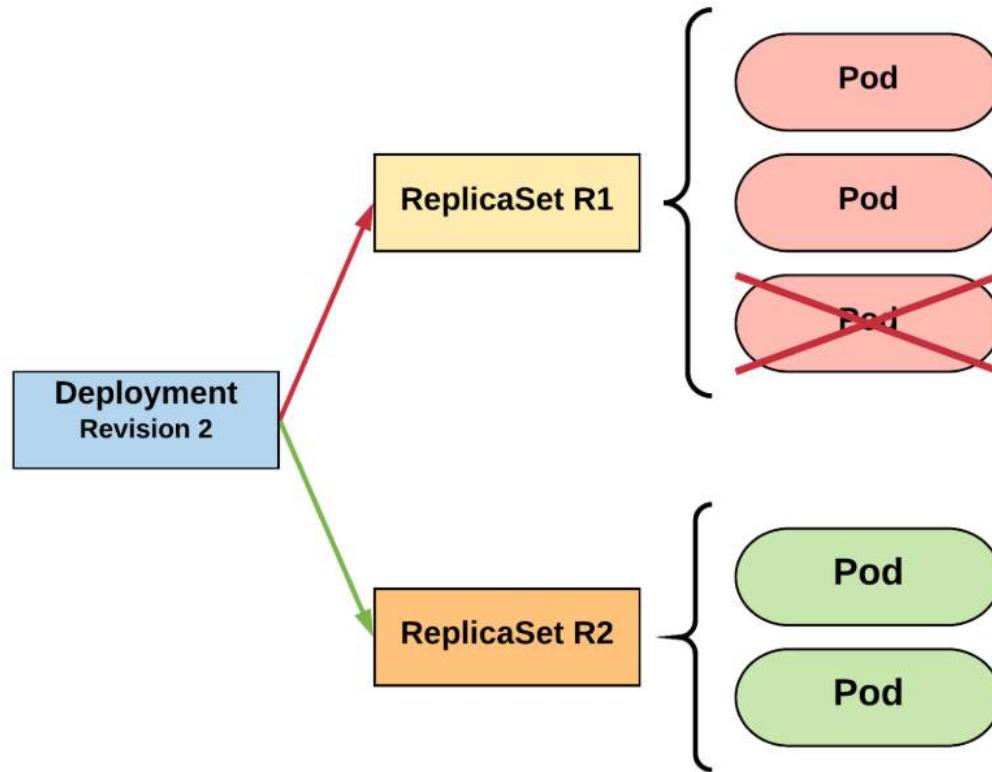
# RollingUpdate



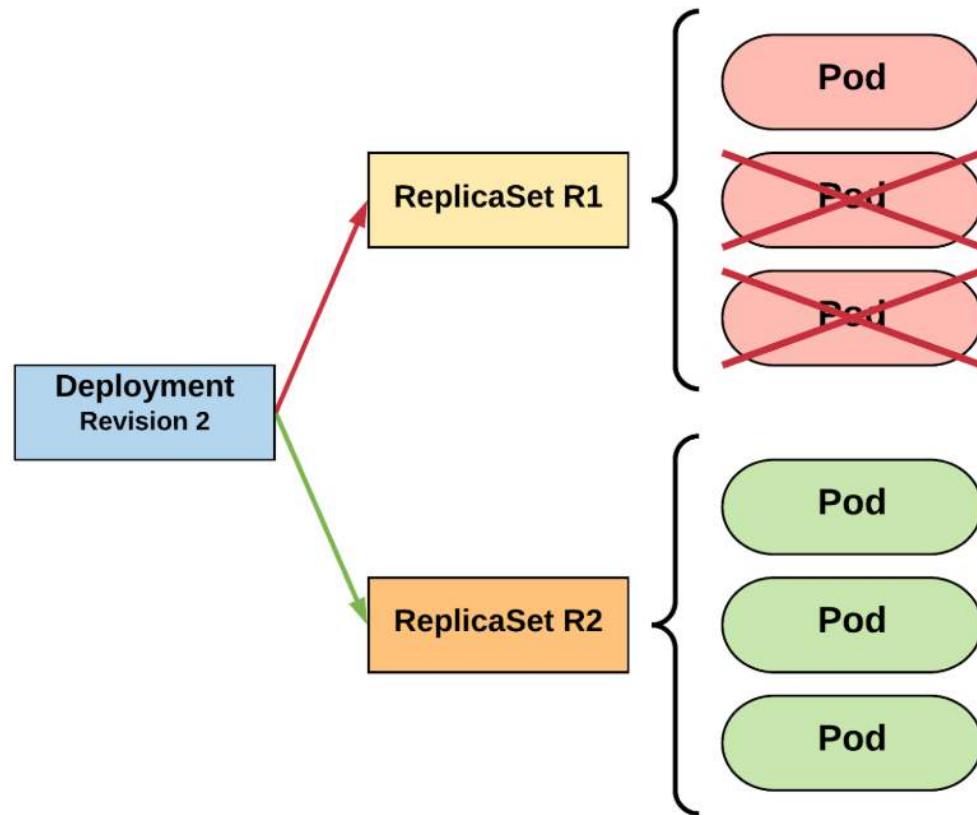
# RollingUpdate



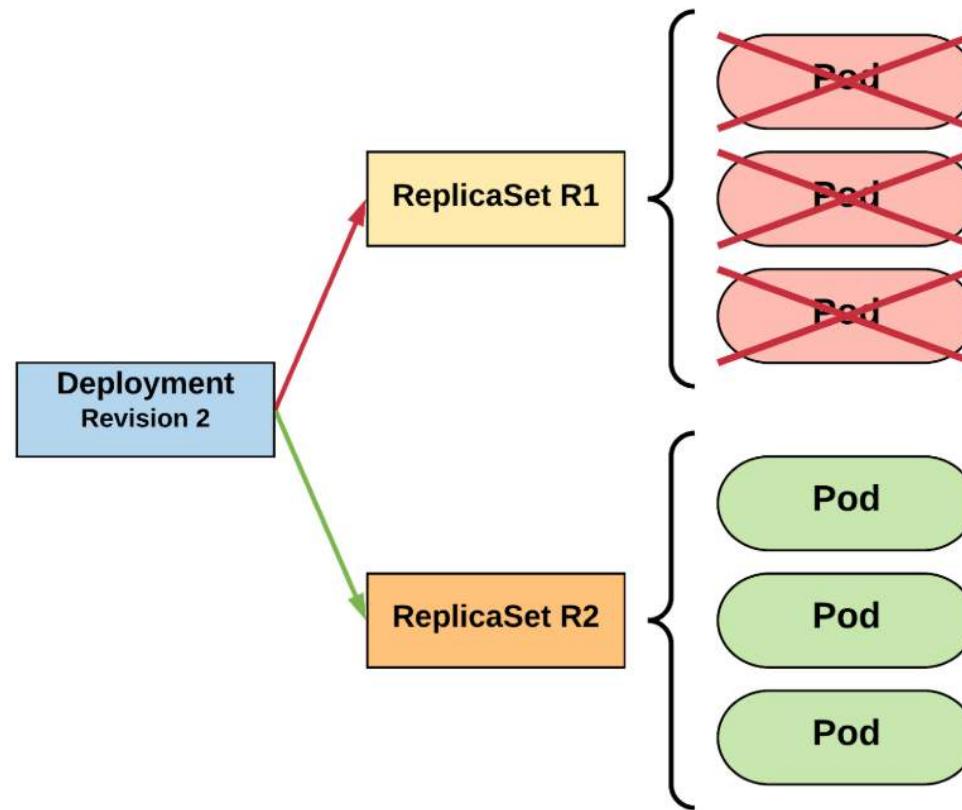
# RollingUpdate



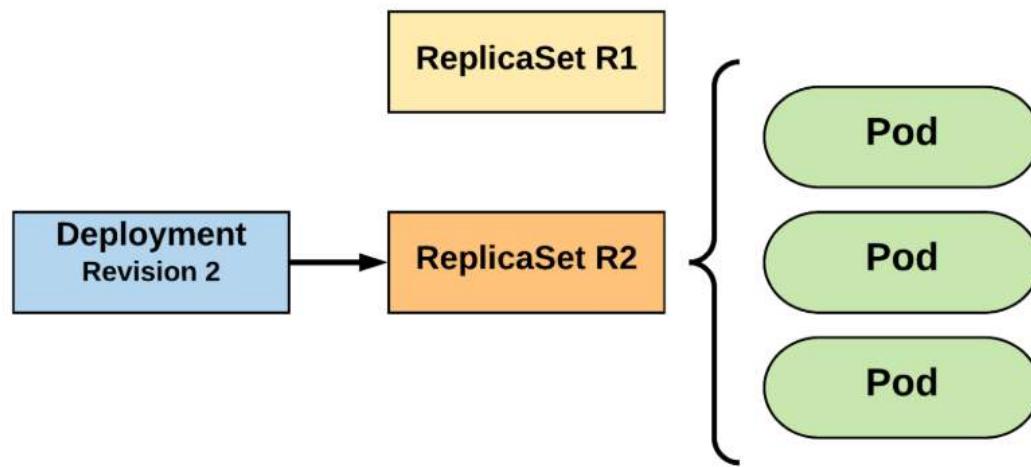
# RollingUpdate



# RollingUpdate



# RollingUpdate



# exercise - deployment order

Instructions:

- create two files: **service.yaml** and **deploy.yaml**
- in **service.yaml** define **NodePort** exposing pods that have **app=api** label
- in **deploy.yaml** create deployment for `gutek/dumpster:v1` image that will have labels defined in **service.yaml**
- write down assumptions: what needs to be deployed first
- verify your assumptions

# discussion - deployment order

- What were your assumptions?
- Were you right?
- Does order matter?

# demo - RollingUpdate

- create service
- run in while curl
- create deployment
- create new deployment

```
while true; do curl http://...; done
#powershell
while ($true) {curl http://... -UseBasicParsing}
#powershell 6+
while ($true) {sleep 1; curl http://... }
```

# Deployment strategies

When it comes to production, a ramped or blue/green deployment is usually a good fit, but proper testing of the new platform is necessary.

Blue/green and shadow strategies have more impact on the budget as it requires double resource capacity. If the application lacks in tests or if there is little confidence about the impact/stability of the software, then a canary, a/b testing or shadow release can be used.

If your business requires testing of a new feature amongst a specific pool of users that can be filtered depending on some parameters like geolocation, language, operating system or browser features, then you may want to use the a/b testing technique.



Strategy	ZERO DOWNTIME	REAL TRAFFIC TESTING	TARGETED USERS	CLOUD COST	ROLLBACK DURATION	NEGATIVE IMPACT ON USER	COMPLEXITY OF SETUP
<b>RECREATE</b> version A is terminated then version B is rolled out	✗	✗	✗	■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■
<b>RAMPED</b> version B is slowly rolled out and replacing version A	✓	✗	✗	■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■
<b>BLUE/GREEN</b> version B is released alongside version A, then the traffic is switched to version B	✓	✗	✗	■ ■ ■	□ □ □	■ ■ ■	■ ■ ■
<b>CANARY</b> version B is released to a subset of users, then proceed to a full rollout	✓	✓	✗	■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■
<b>A/B TESTING</b> version B is released to a subset of users under specific condition	✓	✓	✓	■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■
<b>SHADOW</b> version B receives real world traffic alongside version A and doesn't impact the response	✓	✓	✗	■ ■ ■	□ □ □	□ □ □	■ ■ ■

<https://github.com/ContainerSolutions/k8s-deployment-strategies>

# discussion - deployment strategies

- How to create different deployment types?
- What can you use?

# Deployments

## managing deployments from kubectl

```
# return status of current deployment
# this waits if the status is not finished
$ kubectl rollout status deployment/NAME

# get a history of rollout's
$ kubectl rollout history deployment/NAME

# get history details of rollout N
$ kubectl rollout history deployment/NAME --revision=N

# undo rollout
$ kubectl rollout undo deployment/NAME

# restart
$ kubectl rollout restart deployment/NAME
```

# exercise - fix deployment

Instructions (use **kubectl** only):

- deploy **deploy.yaml** (hint: use --record=true)
- deploy **new-deploy.yaml**
- check deployment status
- check what has changed (hint: --revision)
- in editor fix and re-deploy **new-deploy.yaml**

# discussion - fix deployment

- what have you used to see what has changed?
- what has changed?
- is there other way of fixing this rather than editing yaml?
  - which way is better?

# Deployments

when do we know that our pod is alive?

```
...
spec:
  containers:
    - name: NAME
      image: gutek/dumpster:0.1
      livenessProbe:
        httpGet:
          path: /path
          port: 80
        initialDelaySeconds: 5
        periodSeconds: 5
        timeoutSeconds: 1
        failureThreshold: 3
```

- **livenessProbe** - defines when POD is alive
- **httpGet** - verify that container is running by executing following **path** on given **port**
- **failureThreshold** - how many times we should times try before give up, default 3
- **periodSeconds** - checks again after Ns, default 10s
- **initialDelaySeconds** - wait Ns before checking
- **timeoutSeconds** - finish request after Ns, default 1s

# exercise - liveliness

Instructions:

- deploy **dep.yaml**
- modify **dep.yaml** to include **livenessProbe**
  - set **httpGet:path** to **/test** url
- re-deploy **dep.yaml** with changes
- confirm that deployment did succeeded/failed

# discussion - liveliness

- Did deployment succeed?
- How did you know if deployment did succeed/failed?
- Change url to **/healthz** and deploy again

# Deployments

when do we know that our pod is ready/deployed?

```
...
spec:
  containers:
    - name: NAME
      image: gutek/dumpster:0.1
      readinessProbe:
        httpGet:
          path: /healthz
          port: 80
        initialDelaySeconds: 5
        periodSeconds: 5
        timeoutSeconds: 1
        failureThreshold: 3
```

- **readinessProbe** - defines when POD is ready to receive requests
- **httpGet** - verify that container is running by executing following **path** on given **port**
- **failureThreshold** - how many times we should times try before give up, default 3
- **periodSeconds** - checks again after Ns, default 10s
- **initialDelaySeconds** - wait Ns before checking
- **timeoutSeconds** - finish request after Ns, default 1s

clean up reminder :)

**workshop=true**

label

# exercise - readiness

Instructions:

- modify **dep.yaml** to include **readiness** probe
  - set **httpGet.path** to **/healthz** url
- deploy **dep.yaml**, see what happens
- play around with **failureThreshold**, **periodSeconds**,  
**initialDelaySeconds** and **timeoutSeconds** to find best  
and working solution

# discussion - readiness

- What happened after first deployment?
- Did you found perfect set of parameters that works in this case?
- What values have you used?

# Deployments

liveliness & readiness best practices

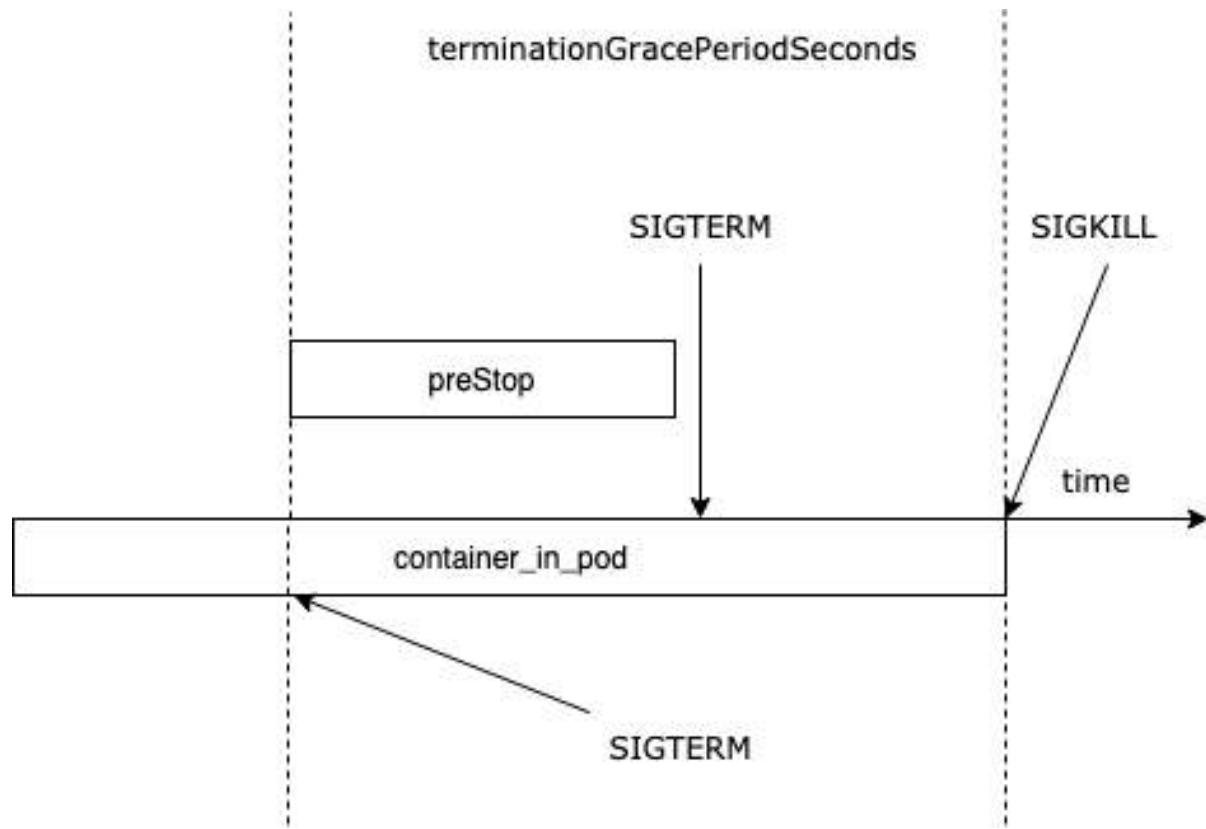
- Always specify
- If you do not have special endpoint, try access your's default one
- It have impact on *deployment* time, however it makes sure that your services are running as expected
- Sometimes being more explicit helps when and if k8s change defaults

# Deployments

help with zero-downtime

our app might need extra time to shutdown gracefully

```
...
spec:
  containers:
    - name: NAME
      image: gutek/dumpster:0.1
    ...
  lifecycle:
    preStop:
      exec:
        command:
          - sh
          - -c
          - "sleep 10"
```



# Deployments

## best practices

- Deployments are good for **stateless services**
- The more details you add to yaml, the better control of deployment process you will have
- Start with **1 replica!** extend if needed
- Use labels
- ~~Remember to add --record~~
- Monitor deployments - you will not get an alert that old replica set is still working and new had failed
- add liveness and readiness probes

**Tasks**



# Jobs

fire and forget

- All system have "tasks"
- Units of work that needs to be done on schedule
- Mostly processing thing

# Jobs

## template

```
apiVersion: batch/v1
kind: Job
metadata:
  name: some-name
spec:
  template:
    ...

```

- **spec.template** - pod template
- rest is self explanatory

# Jobs

## kubectl

```
# get cron jobs
$ kubectl get cronjob

# on cron schedule new job is created
$ kubectl get jobs --watch

# get pods created by jobs
$ kubectl get pods

# get job result
$ kubectl logs pod-name
```

# demo - job

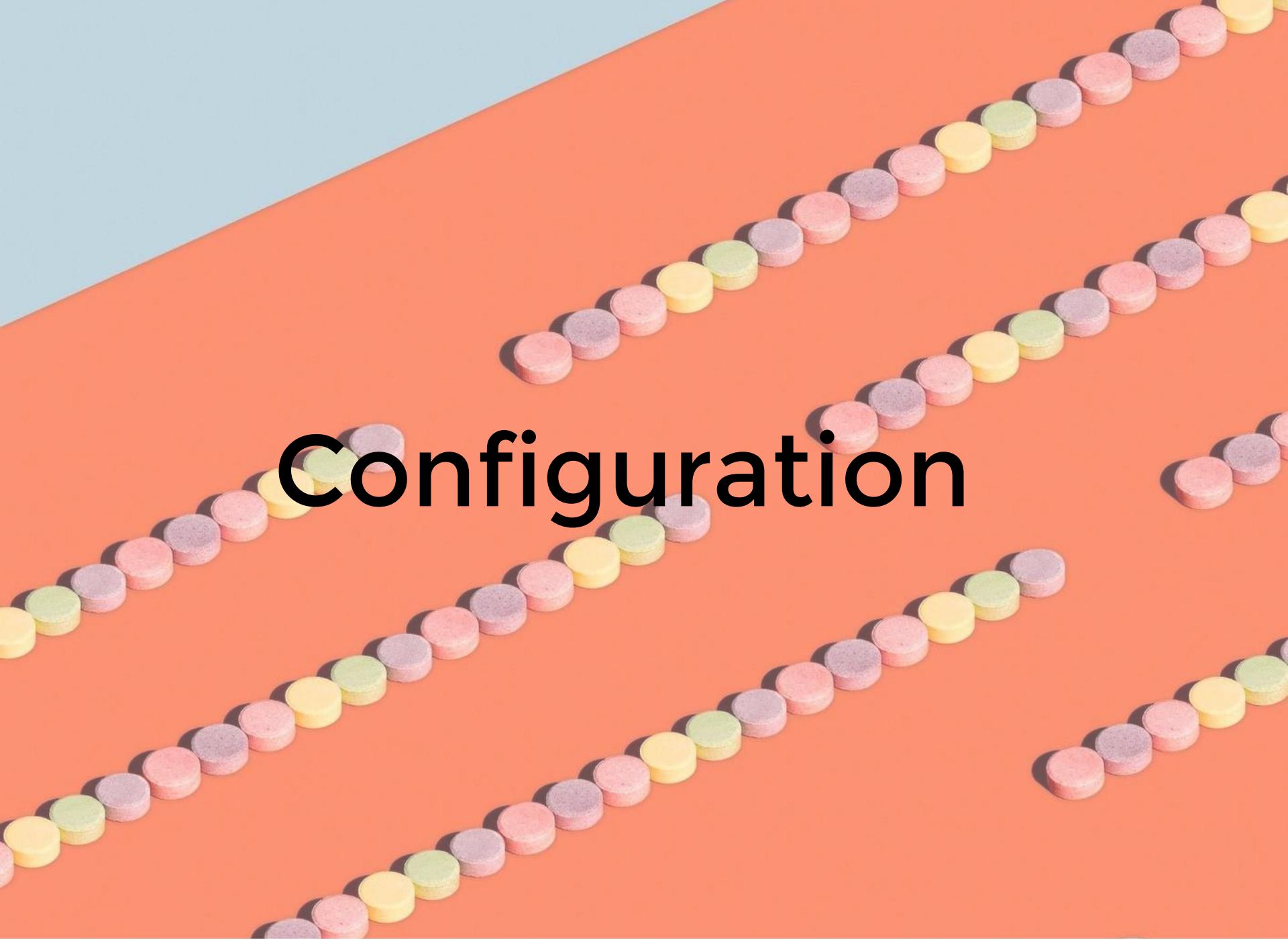
- execute single job
- check status of the job
- check log of the job
- clean up

# CronJobs

## template

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        metadata:
          labels:
            app: hello-cronpod
        spec:
          containers:
            - name: hello
              image: busybox
              args:
                - /bin/sh
                - -c
                - date; echo Hello World
  restartPolicy: OnFailure
```

- **schedule** - cron schedule of job
- **jobTemplate** - exactly the same as in **Job**
- **job.Template.spec.template** - pod template
- rest is self explanatory



# Configuration

# Namespaces

group application

- Virtual cluster
- Allows to separate applications
- i.e. namespace per env, namespace per team
- Allows scoping resource names - two namespaces can have resources of the same name
- Add complexity to all queries

# Namespaces

## kubectl

```
# get all namespaces
$ kubectl get namespace

# get pods from namespace A
$ kubectl get pods --namespace=A
$ kubectl get pods -n=A

$ get all pods from all namespaces
$ kubectl get pods --all-namespaces

$ get all from all namespaces
$ kubectl get all --all-namespaces
```

# Namespace

## template

```
apiVersion: v1
kind: Namespace
metadata:
  name: our-name
---
```

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  namespace: our-name
...
```

- **name**- name of our namespace
- **metadata.namespace**- name of the namespace where given object should be placed

# exercise - Namespace

Instructions:

- create namespace **k8s**
- deploy pod to newly created namespace
- list pod in the new namespace
- delete all pods under the new namespace

# discussion - Namespace

- is it doable?
- how did you query all pods?
- how we could leverage namespaces?
- is there a better way so we do not need to set namespace all the time?
- what will happen if you delete namespace?

# ConfigMaps

common configuration

- Reusable configuration
- Many ways of using:
  - as a mounted file/volume
  - as environmental variables

# ConfigMaps

## kubectl

```
# get all config maps
$ kubectl get cm
$ kubectl get configmap

# create/update config map from file
$ kubectl apply -f cm.yaml

# create config map from arguments
$ kubectl create cm CM_NAME --from-literal=key_01=val_01 --from-literal=key_02=val_02

# get config map info
$ kubectl describe configmaps CM_NAME
```

# ConfigMaps

## template

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-name
data:
  KEY_01: VAL1
  KEY_02: VAL2
```

- **data** - key value list of configurations

# Pod

## passing parameters

```
...
spec:
  containers:
    - name: NAME
      image: gutek/dumpster:v1
      env:
        - name: app_version
          value: v1
        - name: app_hostname
          value: NAME
```

- **env**- list of environmental variables
- **name/value** - key/value list of environmental variables passed to pod

# exercise - Pod evn

Instructions:

- do clean-up
- add env parameters to **pod-and-service.yaml**:
  - month: december
  - channel: stable
- deploy **pod-and-service.yaml** using apply
- open deployed service **pod-env** url in browser
- check under /env if env parameter exists

# discussion - Pod env

- how this can be used?
- what its missing?
- what improvements would you add to env to make it usable?
- is there any other way to access pods env?

# Pod

## passing parameters from config map

```
...
spec:
  containers:
    - name: NAME
      image: gutek/dumpster:0.1
      envFrom:
        - configMapRef:
            name: CM_NAME
      env:
        - name: app_version
          valueFrom:
            configMapKeyRef:
              name: CM_NAME
              key: key01
```

- **envFrom.configMapRef** - add all env variables from config map of **name**.  
Config map needs to exist prior pod creation
- **valueFrom.configMapKeyRef** - takes value from **key** located in the config map of **name**

# exercise - config map with env

Instructions:

- clean up
- create config map with two keys and values
- add reference to all keys and values to **pod-and-service.yaml** (hint: use *valueFrom.configMapKeyRef*)
- deploy **config map** using **apply**
- deploy pod **pod-and-service.yaml** using **apply**
- verify that env variables had been created by opening **pod-cm-env** service

# discussion - config map with env

- why would you use **valueFrom**?
- for what you can use config maps as env variables?

# ConfigMaps

## best practices

- Extract common configuration to file
- Inject extra information about env
- if config is not pod specific, add it to config map - easier to manage

# Secrets

common configuration in base64

- Works similar to ConfigMaps
- Allow storing and re-using key value pair
- Value is not encrypted, is in plain text
- Value is base64 encoded
- Where configMap was used we can change that to secret:
  - configMapKeyRef -> secretKeyRef
  - configMapRef -> secretRef
  - configMap -> secret
  - name -> secretName

# Secrets

## kubectl

```
# get all secrets
$ kubectl get secrets

# create/update secret from file
$ kubectl apply -f secret.yaml

# create secret from arguments
$ kubectl create secret generic SECRET_NAME --from-literal=key_01=val_01 --from-literal=key_02=val_02

# get config map info
$ kubectl describe secret SECRET_NAME
Name:           SECRET_NAME
Namespace:      default
Labels:         <none>
Annotations:   <none>

Type:          Opaque

Data
=====
key_02: 6 bytes
key_01: 6 bytes
```

# Secrets

## template

```
apiVersion: v1
kind: Secret
metadata:
  name: env-secrets
data:
  DB_PASSWORD: cDZbUGVXeU5e0ZW
  REDIS_PASSWORD: AAZbUGVXeU5e0Z
```

- **data** - key value list of configurations, whereas value is in base64

# Secrets

## in pods

```
...
spec:
  containers:
    - name: dumpster-container
      image: gutek/dumpster:0.5
      ports:
        - containerPort: 80
    envFrom:
      - secretRef:
          name: local-secret
    env:
      - name: app_version
        valueFrom:
          secretKeyRef:
            name: local-secret
            key: key01
    volumeMounts:
      - name: config-volume
        mountPath: /etc/config
```

- **envFrom.secretRef** - add all env variables from secret of **name**. Secret needs to exist prior pod creation
- **valueFrom.secretKeyRef** - takes value from **key** located in the secret of **name**

# Secrets

## best practices

- For dev/test env only
- Use whenever there is something *like* password
- Make sure its not checked in into repro
- SECRETS **are not secure**, encrypted value stores!
- Only protection is RBAC

# Volumes



# Volumes

## types

- There are a lot of **types**, few basics are:
- **emptyDir** - lives as long as pod does, allow sharing data between containers
- **configMap** - allow hosting config map key/values as file/content
- **secret** - same as configMap
- **nfs** - Network Folder Share mounted into pod, we need nfs server
- **downwardAPI** - expose pod properties to container including all metadata from pod definition etc.
- **projected** - mount configMap, secret, downwardAPI into single folder
- **hostPath** - node file system mounted into pod
- **azureDisk, azureFile, awsElasticBlockStore, gcePersistentDisk** - cloud storage
- **persistentVolume/Claim** - abstraction over cloud and other storage's, allows claiming PersistentVolume resource

# demo - empty dir

```
...
spec:
  containers:
    - name: dumpster-container
      image: gutek/dumpster:v1
      ports:
        - containerPort: 8080
    volumeMounts:
      - name: empty-volume
        mountPath: /var/data
  volumes:
    - name: empty-volume
      emptyDir: {}
```

- **volumeMounts**

- **name** - name of the mount to be used
- **mountPath** - local (container) path to be used to which **name** will be mounted, it "hides" anything that is present under that path

- **volumes**

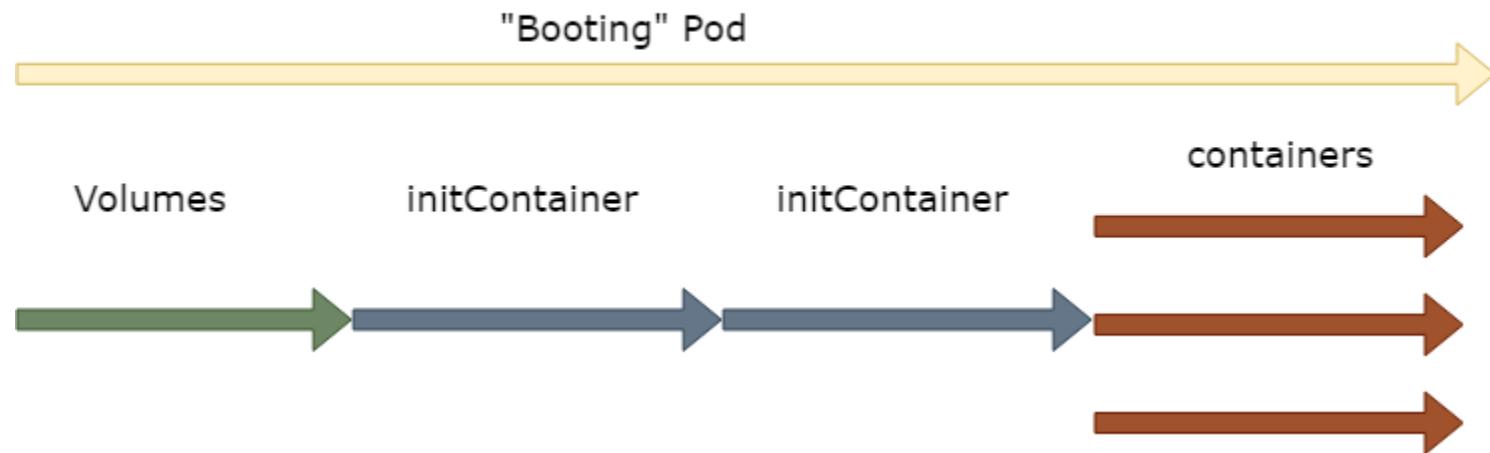
- **name** - name of the volume to be used in pod
- **emptyDir** - empty directory

# note - gitRepo

```
...
spec:
  initContainers:
    - name: git-container
      image: ...
      args:
        - /bin/sh
        - -c
        - git clone https://github.com/kubernetes/kubernets /local/
  volumeMounts:
    - name: empty-volume
      mountPath: /local/path
  containers:
    - name: dumpster-container
      image: gutek/dumpster:v1
      ports:
        - containerPort: 8080
      volumeMounts:
        - name: empty-volume
          mountPath: /var/data
  volumes:
    - name: empty-volume
      emptyDir: {}
```

- is deprecated
- we can achieve same thing with emptyDir
- **initContainers**

# initContainers



# demo - config map as volume

```
...
spec:
  containers:
    - name: dumpster-container
      image: gutek/dumpster:0.5
      ports:
        - containerPort: 80
    volumeMounts:
      - name: config-volume
        mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        name: local-cm
```

- **volumeMounts**

- **name** - name of the mount to be used
- **mountPath** - local (container) path to be used to which **name** will be mounted, it "hides" anything that is present under that path

- **volumes**

- **name** - name of the volume to be used in
- **configMap** - volume comes from config map of **name** local-cm

# demo - secret as volume

```
...
spec:
  containers:
    - name: dumpster-container
      image: gutek/dumpster:0.5
      ports:
        - containerPort: 80
      volumeMounts:
        - name: config-volume
          mountPath: /etc/config
    - name: config-volume
      secret:
        secretName: local-secret
```

- **volumeMounts**

- **name** - name of the mount to be used
- **mountPath** - local (container) path to be used to which **name** will be mounted, it "hides" anything that is present under that path

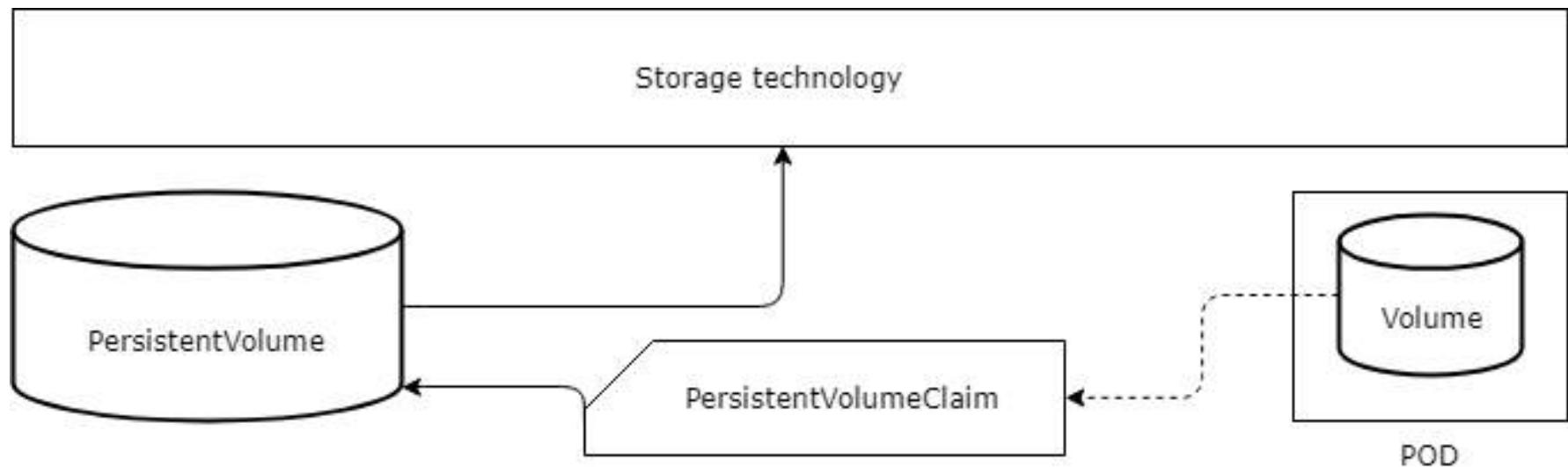
- **volumes**

- **name** - name of the volume to be used in pod
- **secret** - volume comes from secret of **name** local-cm

**emptyDir, configMap,  
secret, hostPath** contains data as  
long as Pod/Node works. They **do  
not sync** data back!

# PersistentVolume/Claims

decoupling storage technology



# PersistentVolume

## template

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
    - ReadOnlyMany
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: /tmp/mypath
```

- **capacity.storage** - size of the storage
- **accessMode** - how this volume can be mounted in nodes
- **persistentVolumeReclaimPolicy** - what should be done with volume when pod is not using it any more? **Retain** - leave it with content, after its release from claim. But its not usable, to use it again, you need to delete and create same one again. **Recycle** - clears content and make it available to re-use. **Delete** - deletes volume and data.
- **hostPath** - type of storage

# PersistentVolumeClaim

## template

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  volumeMode: Filesystem| raw
  resource:
    requests:
      storage: 1Gi
  accessModes:
    - ReadWriteOnce
    - ReadOnlyMany
  storageClassName: slow
  selector:
    ...
```

- **volumeMode** - can be on PV too, they need to match, introduced in 1.13 k8s
- **resource.requests.storage** - how big storage we need
- **accessModes** - same as in PV
- **storageClassName** - specify which provisioner should be used to provision PV. PV and PVC needs to match, we can leave this empty
- **selector** - same as in Deployments, filter PV by labels

# Using PersistentVolumeClaim in Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-volume
spec:
  containers:
  - name: fc-container
    image: gutek/dumpster:v1
    volumeMounts:
    - name: data
      mountPath: /var/data
  volumes:
  - name: data
    persistentVolumeClaim:
      claimName: my-pvc
```

# demo - Simple PVC on Azure

- use AKS
- apply pvc.yaml
- apply pod.yaml
- create file at /mnt/azure
- destroy pod & recreate it
- find disk with
- More <https://docs.microsoft.com/en-us/azure/aks/azure-files-dynamic-pv>

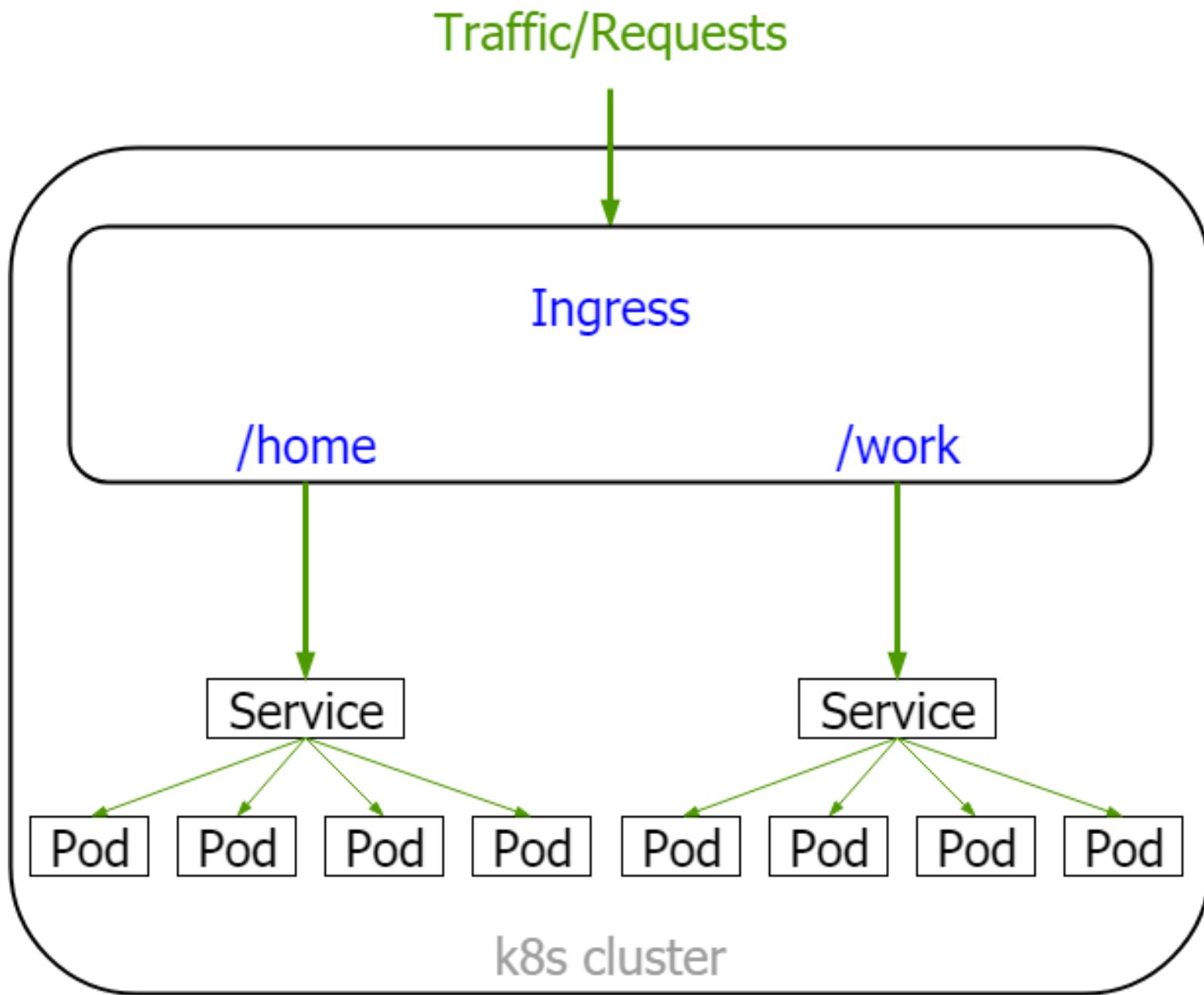
A photograph of a yellow-walled courtyard. The floor is made of colorful, patterned tiles. A traditional archway leads to another room. A small lantern hangs on the wall.

Ingress

# Ingress

/'ɪng्रɛs/

*The action or fact of going in or entering;  
the capacity or right of entrance.*



based on Ahmet Alp Balkan drawings

# Ingress

## installation

```
# for minikube
minikube addons enable ingress

# for docker for desktop
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/cont:

# verify
kubectl get po -n ingress-nginx -w

# popular - the rest of the world :)

# add repo chart
helm repo add nginx-stable https://helm.nginx.com/stable
helm repo update

# install simple but ....
helm install my-release nginx-stable/nginx-ingress

# install complicated
kubectl create ns ingress-nginx
helm install my-release nginx-stable/nginx-ingress --namespace ingress-nginx --se
```

# Attention: Ingress

```
apiVersion: extensions/v1beta1
```

Changed into:

```
apiVersion: networking.k8s.io/v1
```

# Ingress

## default configuration

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: dumpster-ing
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/enable-cors: "true"
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
spec:
  rules:
    - http:
        paths:
          - path: /01
            backend:
              serviceName: dumpster-01-svc
              servicePort: 8080

          - path: /02
            backend:
              serviceName: dumpster-02-svc
              servicePort: 8080

          - path: /
            backend:
              serviceName: dumpster-03-svc
              servicePort: 8080
```

# Ingress

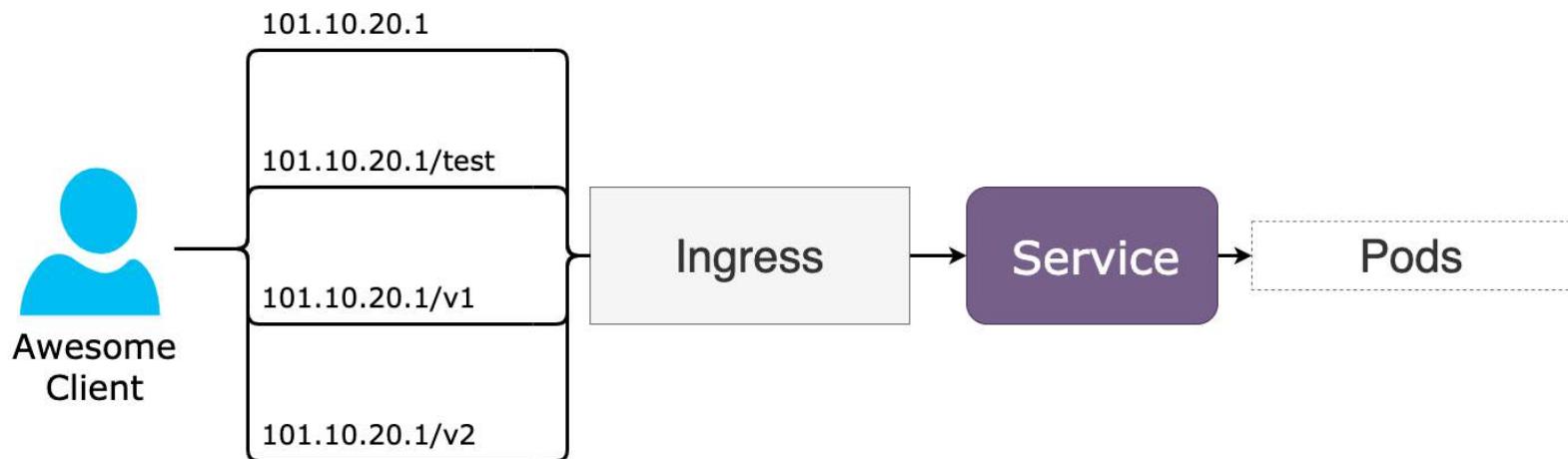
using external load balancer

- best way to do this is during installation
- after installation we need to manually update dynamically named service
- there is no guarantee that this will work
- best solution:

```
$ helm install stable/nginx-ingress --set controller.service.loadBalancerIP="8.8  
,controller.service.externalTrafficPolicy="Local"
```

# Ingress

Default backend



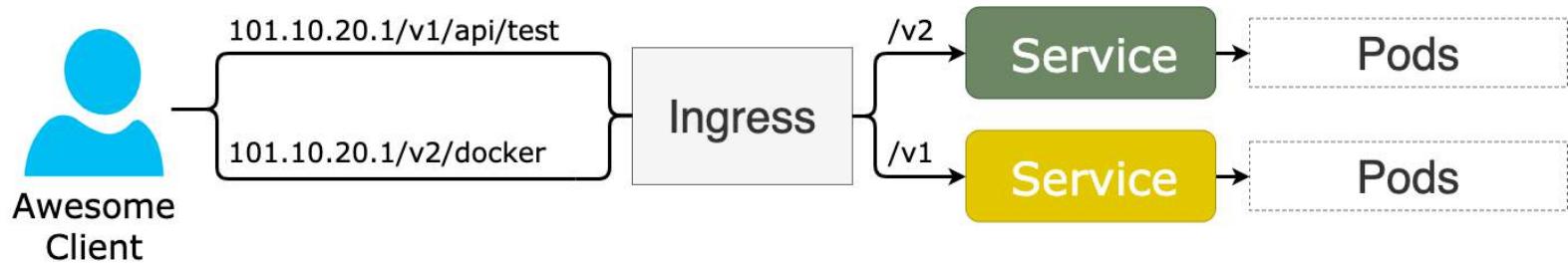
# Ingress

## default backend

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: all-paths
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  rules:
  - http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: servicename
          port:
            number: 8080
```

# Ingress

Fanout routing



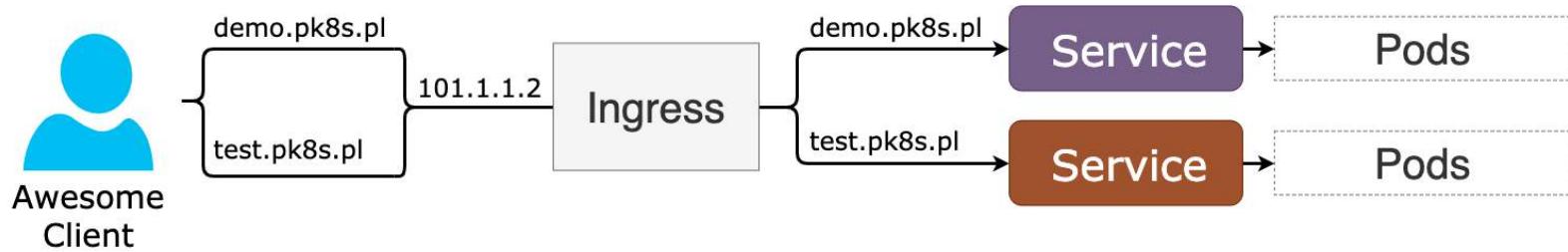
# Ingress

## Fanout routing

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: fanout
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  rules:
  - http:
      paths:
      - pathType: Prefix
        path: "/api/v1"
        backend:
          service:
            name: svc-v1
            port:
              number: 8080
      - pathType: Prefix
        path: "/api/v2"
        backend:
          service:
            name: svc-v2
            port:
              number: 8080
```

# Ingress

Name based virtual hosting



# Ingress

## Name based virtual hosting

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: virtual-hosting
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  rules:
  - host: v1.k8s.buzz
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: echo-v2
            port:
              number: 8080
  - host: v2.k8s.buzz
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: echo-v2
            port:
              number: 8080
```

# Ingress

## Rewrite target

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: rewrite-target
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    kubernetes.io/ingress.class: "nginx"
spec:
  rules:
  - host: v1.k8s.buzz
    http:
      paths:
      - pathType: Prefix
        path: "/app"
        backend:
          service:
            name: app
          port:
            number: 8080
```

# exercise - ingress

Instructions:

- create 3 deployments (v1, v2 & v3 dumpster version) and 3 services for deployments
- create ingress that will map paths based on deployment name
- use ingress base file: **ingress.yaml**
- validate that you can open pages for each of 3 deployments

# discussion - ingress

- were you able to create 3 distinct ingress paths?
- did they worked?
- why ingress can be useful?

# Imperative

few operations that might be useful



# kubectl run

- runs particular images (pod)
- creates deployment for managing pod
- will be removed in next versions of K8S

```
# create deployment with pod and one replicaset
kubectl run NAME --image gutek/dumpster:v1

# create 5 replicas
kubectl run NAME --image gutek/dumpster:v2 --replicas=5

# run pod that will be removed once done
kubectl run -it NAME --image=some/img --rm --restart=Never -- curl http://onet

# we can use different generators
# only pod
kubectl run NAME --image=some/img --generator=run-pod/v1
# deployment
kubectl run NAME --image=some/img --generator=extensions/v1beta1
```

- good for quick demos

# kubectl set

- updates particular property of the resource

```
# List the environment variables defined on all pods
kubectl set env pods --all --list

# Update all containers in all replication controllers in the project to have ENV_VAR=VALUE_1
kubectl set env deploy --all ENV_VAR=VALUE_1

# Set a deployment's container image to 'nginx:1.9.1'
kubectl set image deployment/DEPLOYMENT_NAME CONTAINER_NAME=nginx:1.9.1
```

- whenever you do this, always! update yaml to have that changes included
- rule: **don't do it on production...** unless your boss asks you to do this ;)

# exercise - update image

Instructions:

- run image **gutek/dumpster:v1**
- describe deployment to get:
  - deployment name
  - container name
- update image to **gutek/dumpster:v2** (hint: use values from previous point)
- check pod image using describe command

# discussion - update image

- Did it fill right to update image manually?
- What could go wrong?
- Did you know about --dry-run functionality?

# kubectl scale

- manually scale instances of pod manage by ReplicaSet and/or deployment

```
# scale to 3 replicas deployment of name DEPLOYMENT_NAME
$ kubectl scale --replicas=3 deployment/DEPLOYMENT_NAME

# scale to 3 replicas ReplicaSet of name RS_NAME
$ kubectl scale --replicas=3 rs/RS_NAME
```

- when needed can save ass
- always analyze why you needed to use *scale* manually

[https://www.youtube.com/embed/kOa\\_llowQ1c?  
start=1022&enablejsapi=1](https://www.youtube.com/embed/kOa_llowQ1c?start=1022&enablejsapi=1)

[https://youtu.be/kOa\\_llowQ1c?t=1022](https://youtu.be/kOa_llowQ1c?t=1022)

# kubectl expose

- expose resource (pod, deployment, replicates etc.) as new service

```
# create service of type NodePort
# for deployment hello-minikube
# and name it front-minikube
kubectl expose deployment hello-minikube --type=NodePort --name=front-minikube

# create service of type NodePort
# for pod POD_NAME
# and name it POD_NAME_svc
kubectl expose pod POD_NAME --type=NodePort --name=POD_NAME_svc
```

# exercise - run and expose

Instructions:

- use only **kubectl**
- run **gutek/dumpster:v1** with 6 replicas
- **expose** it as service
- make sure you can access service
- change image to **gutek/dumpster:v2**
- open browser and check deployed version

# discussion - run and expose

- how easy was to crate deployment without yaml?
- was it easy to update/set properties?
- how did you solve issue with exposing deployment as service?



# Advanced

Good to know things

# Limits & autoscaling

# Preparation

few things...

```
# install metric server on Docker for Desktop
cd demos\00-hpa
k apply -f ./components.yaml

# verify it with:
kubectl get po -n kube-system -l k8s-app=metrics-server

# and open dashboard

# check available addons
$ minikube addons list

# adding heapster addon on to minikube so we have metrics
# old one -> $ minikube addons enable heapster

minikube addons enable metrics-server
```

# Pod

## resource limit

```
apiVersion: v1
kind: Pod
...
spec:
  containers:
  - name: dumpster-containers
    image: gutek/dumpster:0.1
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

- **resources.requests** - we needs this **memory** and **cpu** to run this pod
- **resources.limits** - we maximum will take this **memory** and **cpu** utilization

# exercise - resource limit

Instructions:

- play with resource requests in **deploy.yaml**
- try to utilize 50% of memory
- try to utilize 60% of cpu
- once 50% of memory is utilized, change **replicas** to 7 and see what happens

# discussion - resource limit

- What was the status of the pod that could no be created?
- Were you able to set proper parameters so deployment took exactly 50% of memory and 60% of cpu?
- How did yo achieve it?

# ResourceQuota

- Allows specify desire limits
- It can be applied globally or locally
- It needs to be turned on --enable-admission-plugins

# ResourceQuota

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: sample
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
    configmaps: "10"
    persistentvolumeclaims: "4"
    pods: "4"
    replicationcontrollers: "20"
    secrets: "10"
    services: "10"
    services.loadbalancers: "2"
```

- **spec.hard** defines list of quotas for current namespace

# LimitRange

- Allows specify desire limits of possible resources usage per pod
- Also allows specify ratio aspect of the resources used in Pod
- It needs to be turned on --enable-admission-plugins

# LimitRange

```
apiVersion: v1
kind: LimitRange
metadata:
  name: test-limit
spec:
  limits:
    - maxLimitRequestRatio:
        memory: 10
        type: Pod
    - max:
        cpu: "800m"
        memory: "1Gi"
      min:
        cpu: "100m"
        memory: "99Mi"
      default:
        cpu: "700m"
        memory: "900Mi"
      defaultRequest:
        cpu: "110m"
        memory: "111Mi"
      type: Container
    - type: PersistentVolumeClaim
      max:
        storage: 2Gi
      min:
        storage: 1Gi
```

- **limits** defines limits of resources for pod or container

# Autoscaling

how to scale based on resources?

- Currently deployments does not scale automatically - only manually
- During heavy load the only way of scale is to execute *kubectl scale*
- Autoscaling gives as tools to manage scaling depending on resource usage

# Autoscaling

## kubectl - imperatively

```
# auto scal to max 10 replicas when cpu utilization is at around 10%
# this is autoscaling/v1
$ kubectl autoscale deploy deployment-name --min=3 --max=10 --cpu-percent=10

# get all autoscalers
$ kubectl get hpa

# describe autoscaler
$ kubectl describe hpa hpa-name
```

# Autoscaling

## template

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

- **apiVersion** - autoscaling/v1 allows only scale on
- **targetCPUUtilizationPercentage**, beta allows scaling on multiple metrics

**to make it WORK pods needs to define resources**

# Autoscaling

## template

```
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: dumpster-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: dumpster-dep
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: memory
      targetAverageUtilization: 10
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 10
  - type: Pods
    pods:
      metricName: packects-per-secon
      targetAverageValue: 1k
```

### **MAIN THING TO TAKE FROM THIS TEMPLATE**

autoscaling/v1 allows only scale on cpu utilization %

autoscaling/v2beta1 allows to scale on multiple metrics

**to make it WORK pods needs to define  
resources**

# demo - autoscaling

## Instructions:

- set resources limit/requests and deploy **deploy.yaml**
- create autoscaler using kubectl (cpu resource needs to be defined) or template.yaml
- set minimum replicas to 5, maximum to 15
- in new shell tab execute:

```
$ kubectl run -it load-generator --image=busybox /bin/sh
```

Hit enter **for command** prompt

```
$ while true; do wget -q -O- http://dumpster-svc.default.svc.cluster.local; done
```

- see if it autoscaled, if not play around with metrics and redeploy

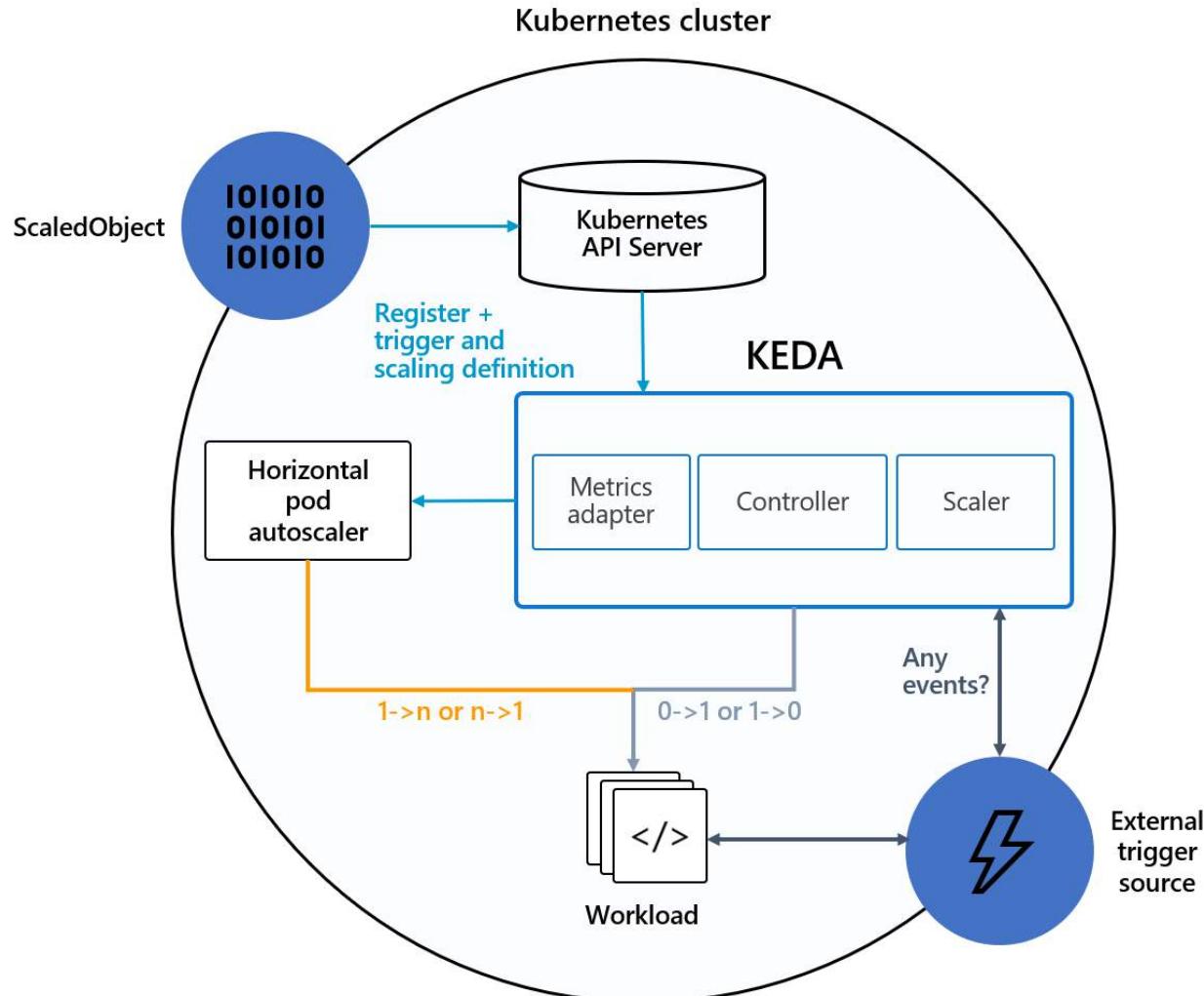
# discussion - autoscaling

- Did you notice autoscaler in work?
- how long did it take to make autoscaling scale?

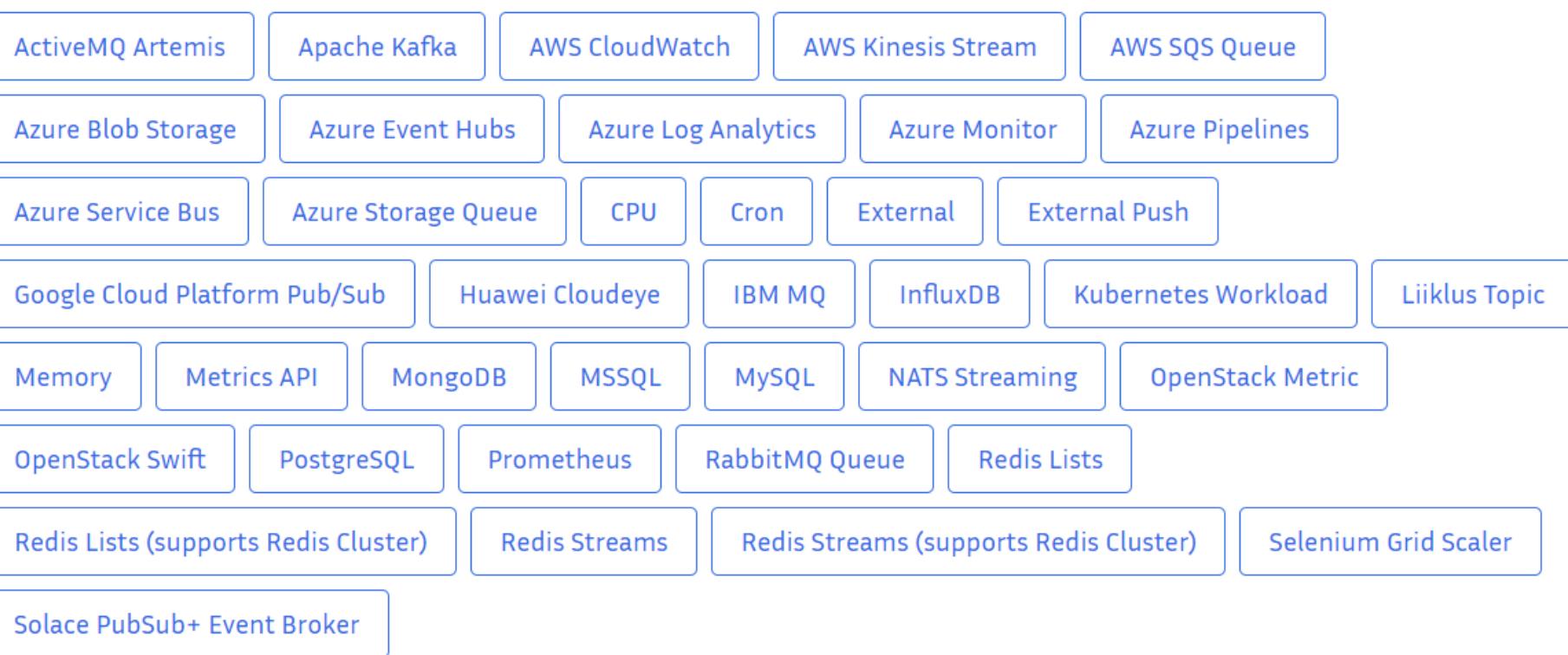
# KEDA - advanced autoscaling

- Kubernetes Event Driven Autoscaler
- When normal metrics are not enough

# KEDA



# KEDA - autoscalers



# Vertical Pod Autoscaler

- "Automagically" adjusts the CPU and memory reservations
- Free space/resources for other pods
- Extension to K8s - separate repo

<https://github.com/kubernetes/autoscaler/tree/master/vertical-pod-autoscaler>

- On AKS as an extention

# VPA - components

- Recommender - monitors current and past resource consumption and recommend new values for CPU and memory requests
- Updater - it checks if managed pods have correct resources set and kill them if not, so they can be recreated with updated requests
- Admission Plugin - it sets the correct resource requests on new pods (created or recreated by their controller due to Updater's activity).

# Before demo

```
1 az feature register \
2     --namespace Microsoft.ContainerService \
3     --name AKS-VPAPreview
4
5 az aks update \
6     --resource-group poznajkubernetes \
7     --name szkolenie-aks \
8     --enable-vpa
```

```
piotr@Azure:~$ az aks update \
--resource-group poznajkubernetes \
--name szkolenie-aks \
--enable-vpa
Argument '--enable-vpa' is in preview and under development. Reference and support levels: https://aka.ms/CLI\_refstatus
The behavior of this command has been altered by the following extension: aks-preview
(BadRequest) Vertical Pod Autoscaler is not supported for the current cluster version. Please upgrade the cluster to version 1.24.0 or above.
Code: BadRequest
Message: Vertical Pod Autoscaler is not supported for the current cluster version. Please upgrade the cluster to version 1.24.0 or above.
```

# demo - VPA

## Instructions:

- installation + verification (check readme)

```
# create namespace and switch to it
k create ns vpa-demo
kubens vpa-demo

# deploy hamster app with VPA
k apply -f \
https://raw.githubusercontent.com/kubernetes\
/autoscaler/master/vertical-pod-autoscaler/examples/hamster.yaml

# run describe
k describe pod hamster-<TAB><TAB>

# check pods with
k get --watch pods -l app=hamster

# wait a few minutes and run describe again on different pod
k describe pod hamster-<TAB><TAB>
```

# VPA - known limitations

- Pod can be recreated on different node
- VPA cannot guarantee that pods it evicts or deletes will be successfully recreated. This can be partly addressed using Cluster autoscaler
- VPA does not evict pods which are not run under a controller
- **VPA shouldn't be used with HPA on CPU and memory** (custom metrics are ok)
- VPA performance has not been tested in large clusters.
- VPA reacts to most OOM events, but not in all situations.
- VPA recommendation might exceed available resources

# DemonSets & StatefulSets

# DNS

how things are resolved

**my-svc.my-namespace.svc.cluster.local**

dumpster-svc.default.svc.cluster.local

**pod-ip.my-namespace.pod.cluster.local**

10-10-10-10.default.pod.cluster.local

[more options](#)

# DeamonSet

- ensures that each node have a running copy of pod
- we can filter-out nodes that should not have pods - if needed

# DeamonSet

## template

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: dumpster-ds
  namespace: kube-system
  labels:
    workshop: "true"
    app: dumpser
spec:
  selector:
    matchLabels:
      app: dumpster-ds-pos
  template:
    metadata:
      labels:
        app: dumpster-ds-pos
        workshop: "true"
    spec:
      tolerations:
        - key: node-role.kubernetes.io/master
          effect: NoSchedule
      containers:
        - name: dumpster-ds-pod-container
          image: gutek/dumpster:0.1
      terminationGracePeriodSeconds: 30
```

# DeamonSet

kubectl

```
# get deamon sets  
kubectl get ds  
  
# delete deamon set  
kubectl delete ds ds-name
```

# demo - deamon set

Instructions:

- deploy **ds.yaml**
- check where pods are deployed

# StatefulSet

- if we need predictable name of pods so we can use it in predictable way
- we need a predictable way of upgrading
- predictable way of destroying pods
- use "new kind of service"

# StatefulSet

## headless service

```
apiVersion: v1
kind: Service
metadata:
  name: dumpster-ss-svc
  labels:
    app: dumpster
    workshop: "true"
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: dumpster-ss-pod
```

# StatefulSet

## template

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: dumpster-ss
  # no lables!!!!!
spec:
  selector:
    matchLabels:
      app: dumpster-ss-pod
  serviceName: "dumpster-ss-svc"
  replicas: 3
  template:
    metadata:
      labels:
        app: dumpster-ss-pod
  spec:
    terminationGracePeriodSeconds: 10
    containers:
      - name: dumpster-ss-pod-container
        image: gutek/dumpster:0.1
        ports:
          - containerPort: 80
```

# StatefulSet

## kubectl

```
# get stateful sets
$ kubectl get statefulset

# delete stateful set
$ kubectl delete statefulset ss-name
```

# demo - stateful set

Instructions:

- deploy **ss.yaml**
- how to access pods?

# StatefulSet

dns

**pod-NUMBER.service-  
name.namespace.svc.cluster.local**

dumpster-ss-2.dumpster-ss-  
svc.default.svc.cluster.local

# Affinity

# Affinity & Anti-Affinity

- Defines how pod can be scheduled on node
- Defined how pod can't be scheduled on node
- Allows to specify nodes on which pod's should be deployed

# Node Selector

```
apiVersion: v1
kind: Pod
metadata:
  name: testpod
  labels:
    env: test
spec:
  containers:
  - name: pkad
    image: poznajkubernetes/pkad
    imagePullPolicy: IfNotPresent
  nodeSelector:
    disktype: ssd
```

- **nodeSelector** - label selector of node to select

# Node Name

```
apiVersion: v1
kind: Pod
metadata:
  name: testpod
  labels:
    env: test
spec:
  containers:
  - name: pkad
    image: poznajkubernetes/pkad
    imagePullPolicy: IfNotPresent
  nodeName: node-01
```

- **nodeName** - select node by name

# Node Affinity

- **requiredDuringSchedulingIgnoredDuringExecution**
  - hard limit, it needs to be fullfill before pod can be scheduled
- **preferredDuringSchedulingIgnoredDuringExecution**
  - soft limit, it would be good if the requirements can be fulfilled

# Node Affinity

```
apiVersion: v1
kind: Pod
metadata:
  name: pkad-node-affinity
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: storage
                operator: In
                values:
                  - ssd
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 10
          preference:
            matchExpressions:
              - key: env
                operator: In
                values:
                  - test
  containers:
    - name: pkad
      image: poznajkubernetes/pkad
```

# Pod Affinity/Ani-Affinity

```
apiVersion: apps/v1
kind: Pod
metadata:
  name: app
  labels:
    app: web-store
spec:
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
              - key: app
                operator: In
                values:
                  - web-store
            topologyKey: "kubernetes.io/hostname"
      podAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
                - key: app
                  operator: In
                  values:
                    - store
            topologyKey: "kubernetes.io/hostname"
  containers:
    - name: web-app
      image: poznaikubernetes/pkad
```

# Design Patterns

multi container systems



# Sidecar

container

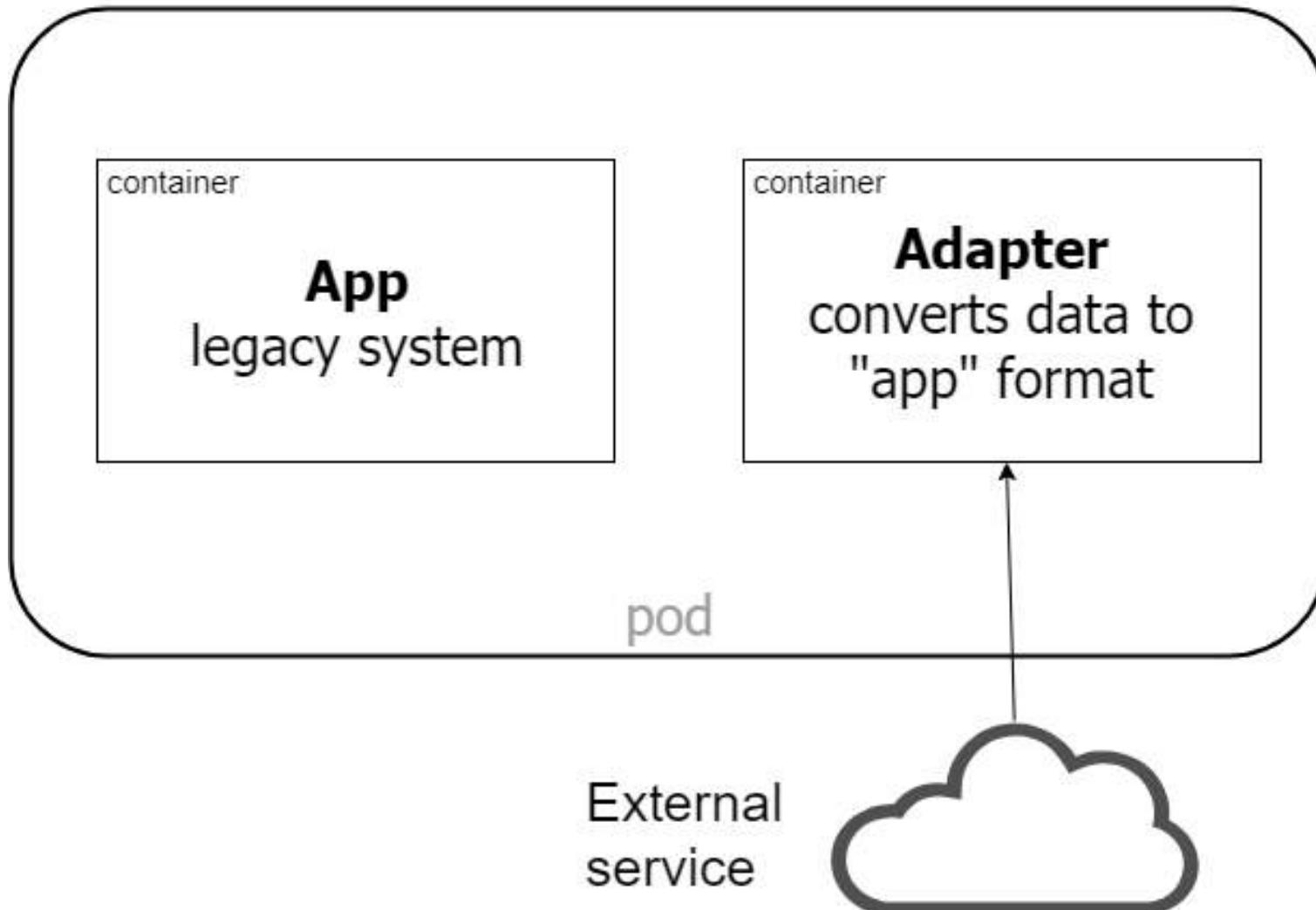
**App**  
writes logs file

container

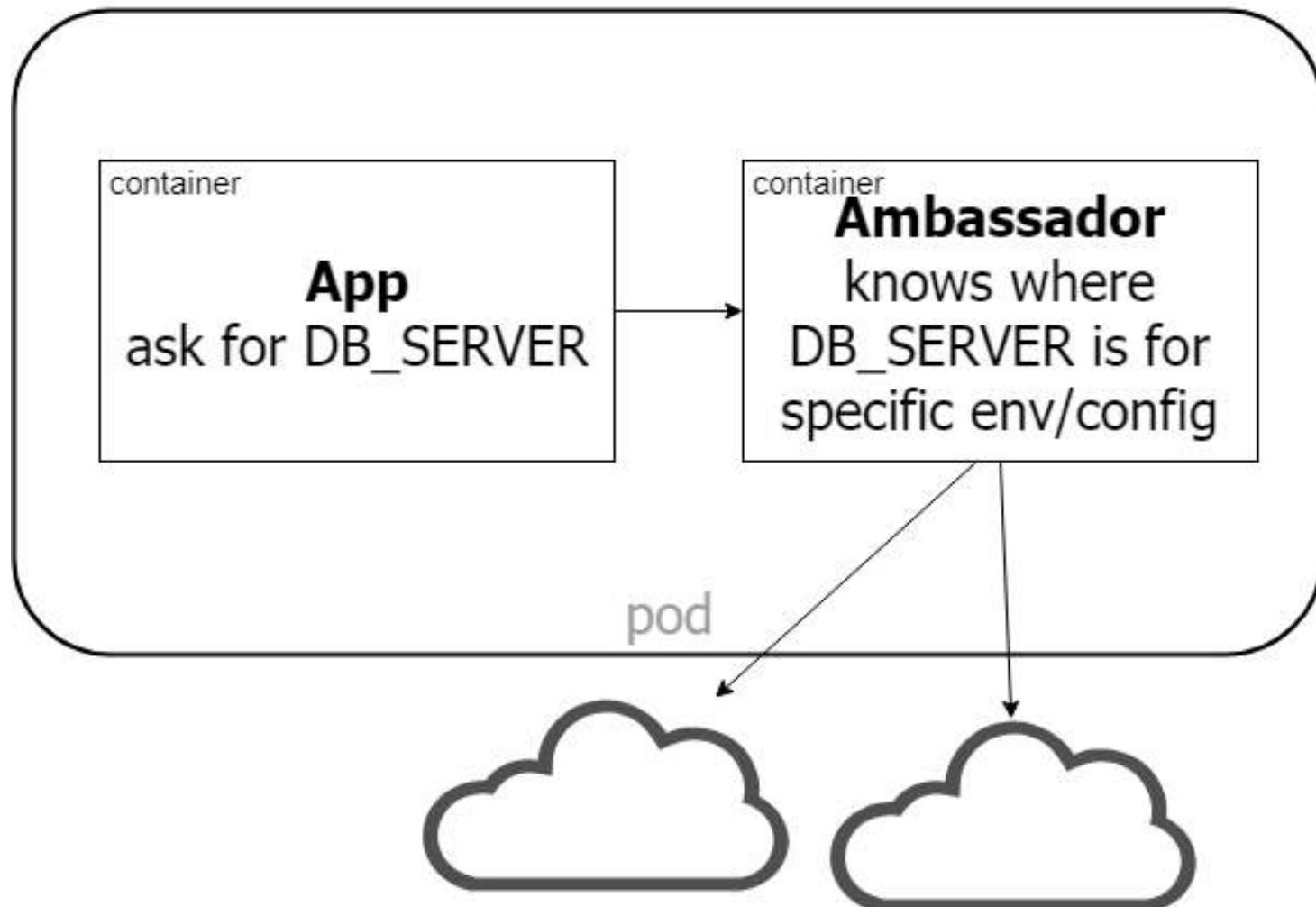
**Sidecar**  
sends logs to ES

pod with shared file system

# Adapter



# Ambassador



A large, weathered wooden ship's wheel is mounted on a metal stand on a boat deck. The wheel has a dark wood finish with metal bands and four spokes. The background shows a blue railing, a white building, and palm trees under a cloudy sky.

helm  
vs  
kustomize

# problem

- multiple env
- differences:
  - "load"
  - "power"
  - domains
  - namespaces
  - ....

# How to solve it?

- copy per environment
- "tokens" in files
- tools:
  - helm
  - kustomize

# Theory

- Template engine -> Helm
- Overlay engine -> Kustomize

# helm

- Package manager for k8s
- Allows installing packages as well as packaging our own solution as "deployable package"
- helm installation is simple as kubectl - binary file

```
# check
$ helm version

# linux
$ sudo snap install helm --classic

# mac
brew install kubernetes-helm

# win
choco install kubernetes-helm

# or by visual studio code kubernetes plugin
```

# chart

```
./Chart.yaml
./values.yaml
./templates/NOTES.txt
./templates/_helpers.tpl
./templates/deployment.yaml
./templates/hpa.yaml
./templates/ingress.yaml
./templates/service.yaml
./templates/serviceaccount.yaml
./templates/tests
./templates/tests/test-connection.yaml
```

# service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: {{ include "poznajkubernetes-helm-example.fullname" . }}
  labels:
    {{- include "poznajkubernetes-helm-example.labels" . | nindent 4 }}
spec:
  type: {{ .Values.service.type }}
  ports:
    - port: {{ .Values.service.port }}
      targetPort: http
      protocol: TCP
      name: http
  selector:
    {{- include "poznajkubernetes-helm-example.selectorLabels" . | nindent 4 }}
```

# helm

## basic commands

```
# initialize helm usage
helm repo add stable https://kubernetes-charts.storage.googleapis.com/

# refresh repo
helm repo update

# search for a package ingress
helm search repo ingress

# list all installed packages
$ helm list

# list all even deleted pacakges
$ helm list --all

# uninstall package but keep it locally
$ help uninstall NAME

# install package
$ help install channel/package-name
```

# helm

- Package manager for k8s
- Allows installing packages as well as packaging our own solution as "deployable package"
- helm installation is simple as kubectl - binary file

# helm

## Pros

- Powerful
- Like any other package manager (apt-get, mvn, npm)
- A huge chart library

## Cons

- Another abstraction level
- Learning curve
- Produces spaghetti

# kustomize

- Built-in in kubectl
- Overlay engine
- K.I.S.S

# demo -kustomize

- build base dir
- build overlay dir
- kubectl apply -k

# kustomize

## Pros

- Native support
- Easy to understand
- Minimal change

## Cons

- Primitive
- It breaks DRY rule

A photograph of two security cameras mounted on a dark, textured wall. One camera is positioned higher than the other. They are connected by a network of black cables and a central junction box. The background shows a brick ledge at the top.

# Monitoring & debugging

# Monitoring & Debugging

- some monitoring and debugging tools depends on k8s environment
- some are provided by kubectl and dashboard
- some needs extra installations i.e. from helm

# Monitoring & Debugging

## kubectl

```
# return information about current state of object
# including latest events
$ kubectl describe resource name

# wide return all resources - status of resources
$ kubectl get all -o wide

# get latest logs from pod
$ kubectl logs pod-name

# get all logs from all pods in deployment
$ kubectl logs deployment/deployment-name

# events happening in the system
$ kubectl get events --sort-by=.metadata.creationTimestamp

# checking pods/modes most used
$ kubectl top nod
$ kubectl top pod

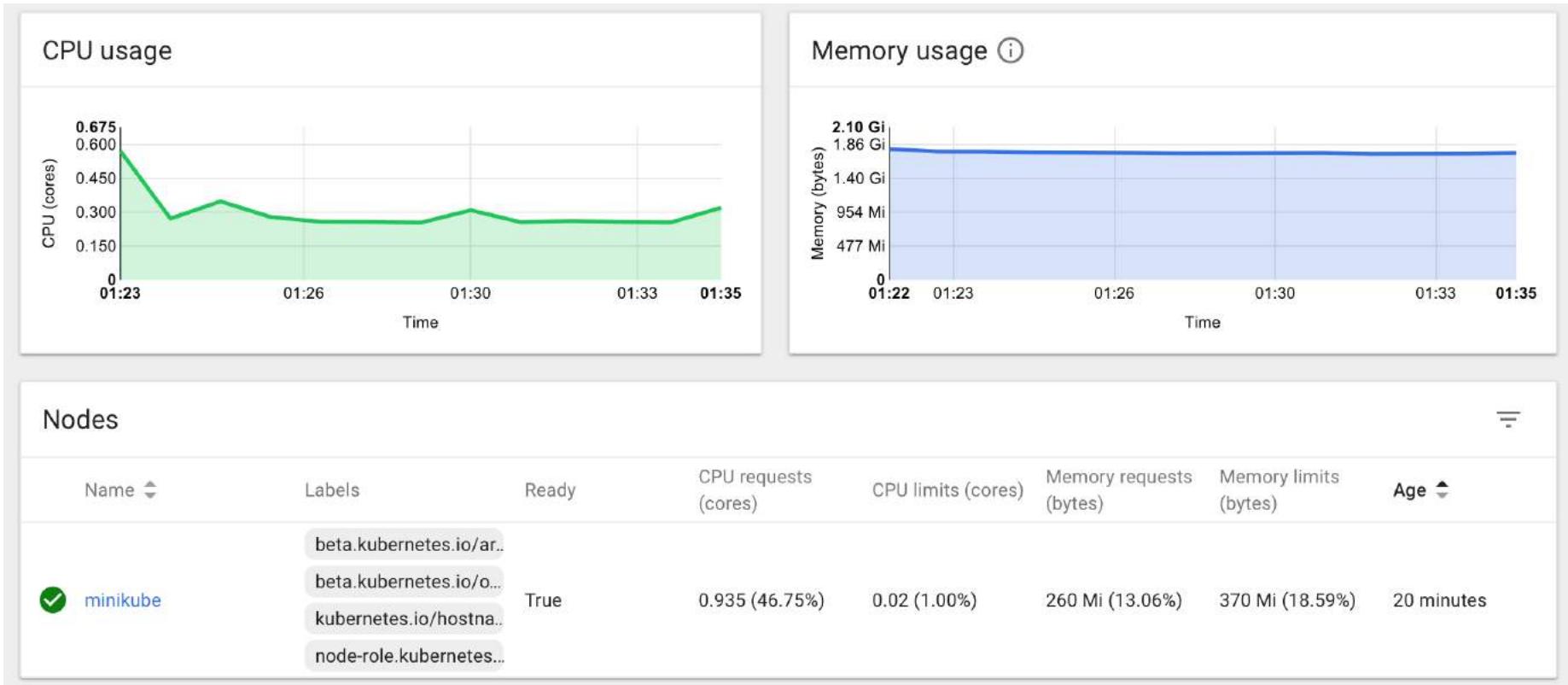
# rollout deployment
$ kubectl rollout ...

# exec command on pod
$ kubectl exec pod-name -i -t /bin/sh

# attach to running container
$ kubectl attach pod-name
```

# Monitoring & Debugging

## dashboard



# exercise - check log

Instructions:

- create pod with image **gutek/dumpster:v1**
- deploy it
- execute / endpoint
- check logs

# discussion - check log

- did you were able to find out what had gone wrong?
- when if this would happen on app start, would log still worked?

# External tools

## grafana

```
# used to be so simple...
minikube addons open heapster

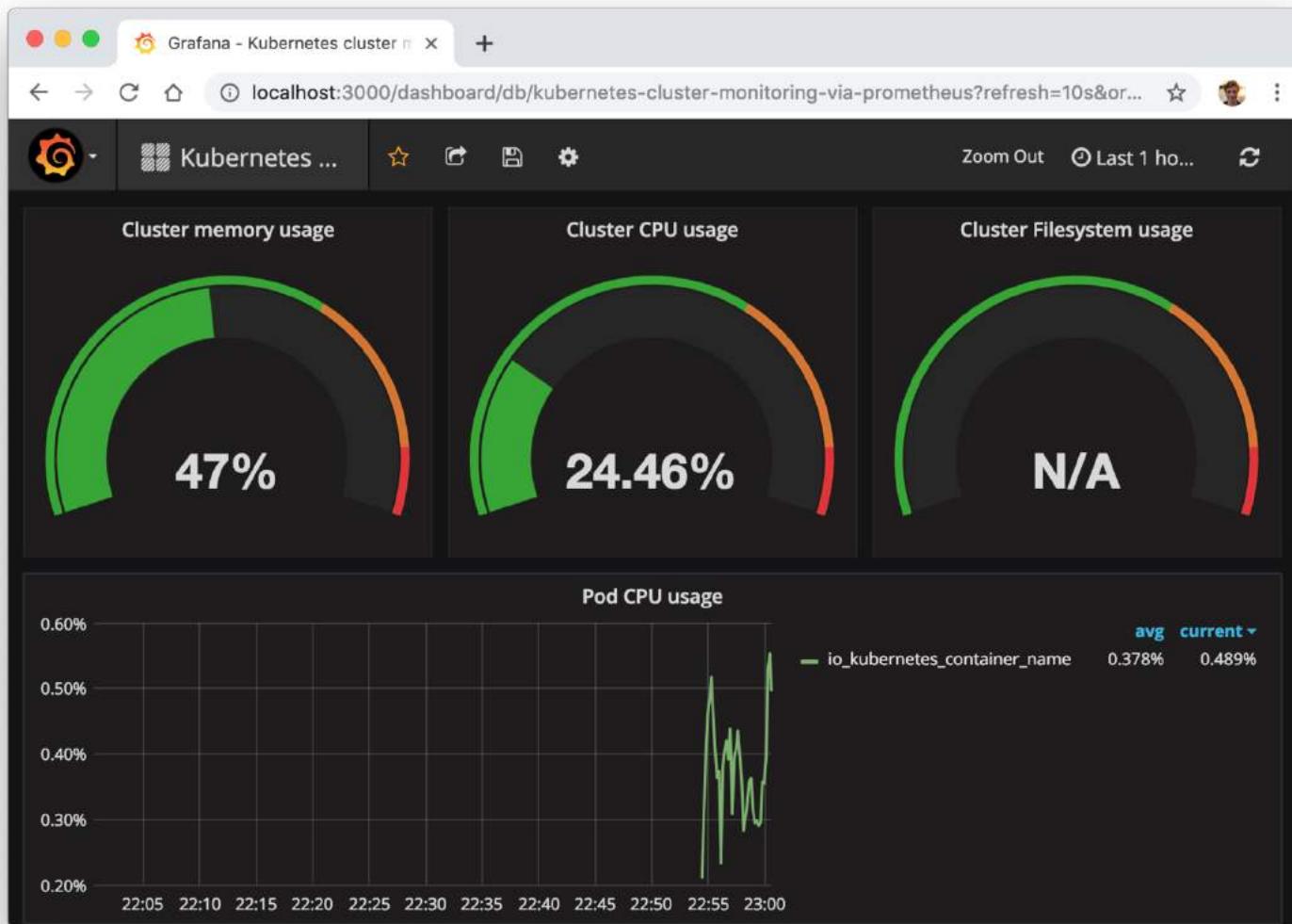
# now:
kubectl apply -f https://raw.githubusercontent.com/giantswarm/kubernetes-prometheus/0.1.0/deployment.yaml

kubectl get pods -n monitoring -l app=grafana,component=core
kubectl port-forward -n monitoring POD_FROM_PREV 3000:3000

# localhost:3000/login admin:admin
```

# External tools

grafana



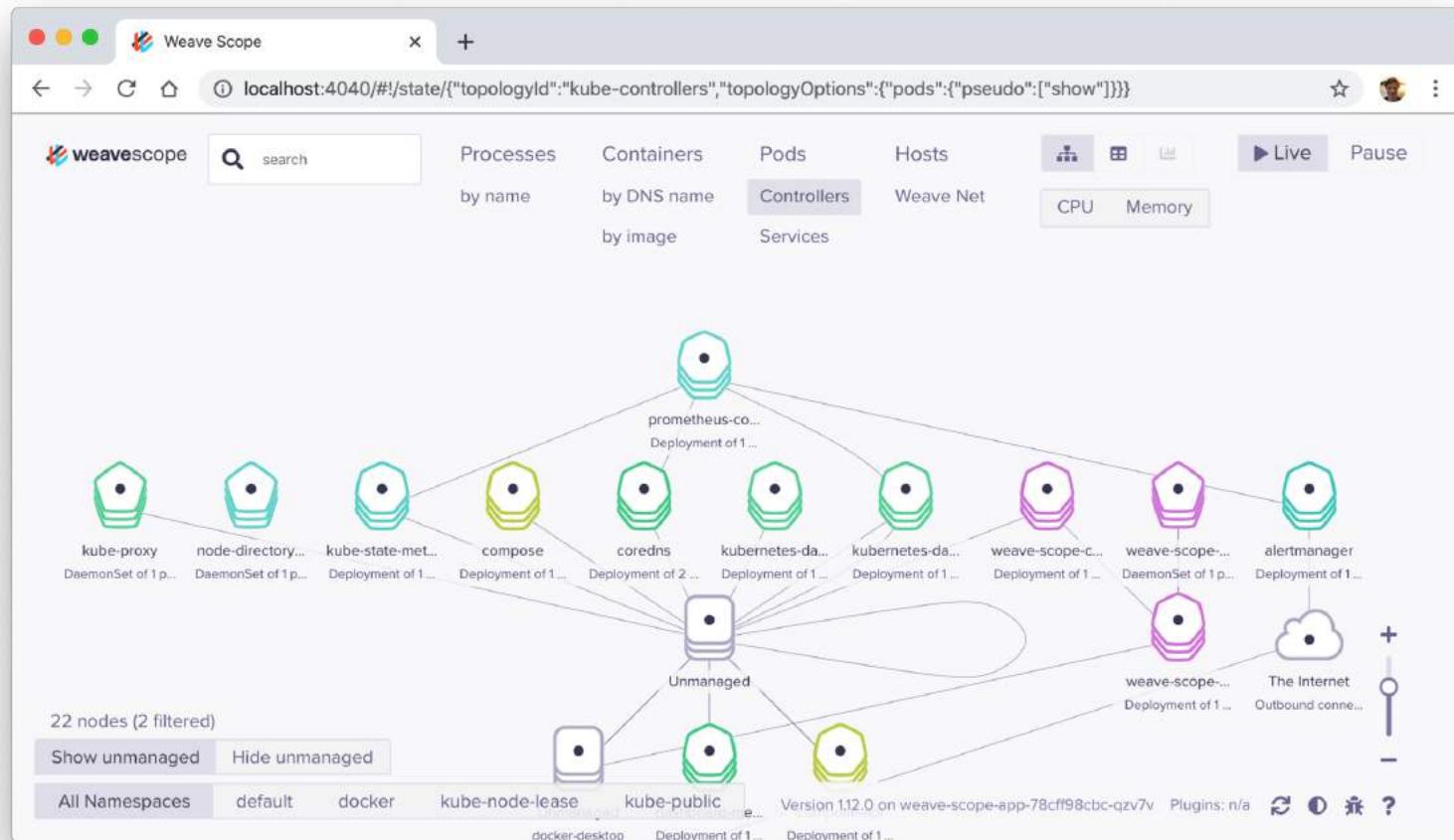
# External tools

## wave

```
kubectl apply \
-f "https://cloud.weave.works/k8s/scope.yaml?k8s-version=$(kubectl version | \
kubectl port-forward -n weave \
"$(kubectl get -n weave pod --selector=weave-scope-component=app -o jsonpath=
# localhost:4040
```

# External tools

wave



# External tools

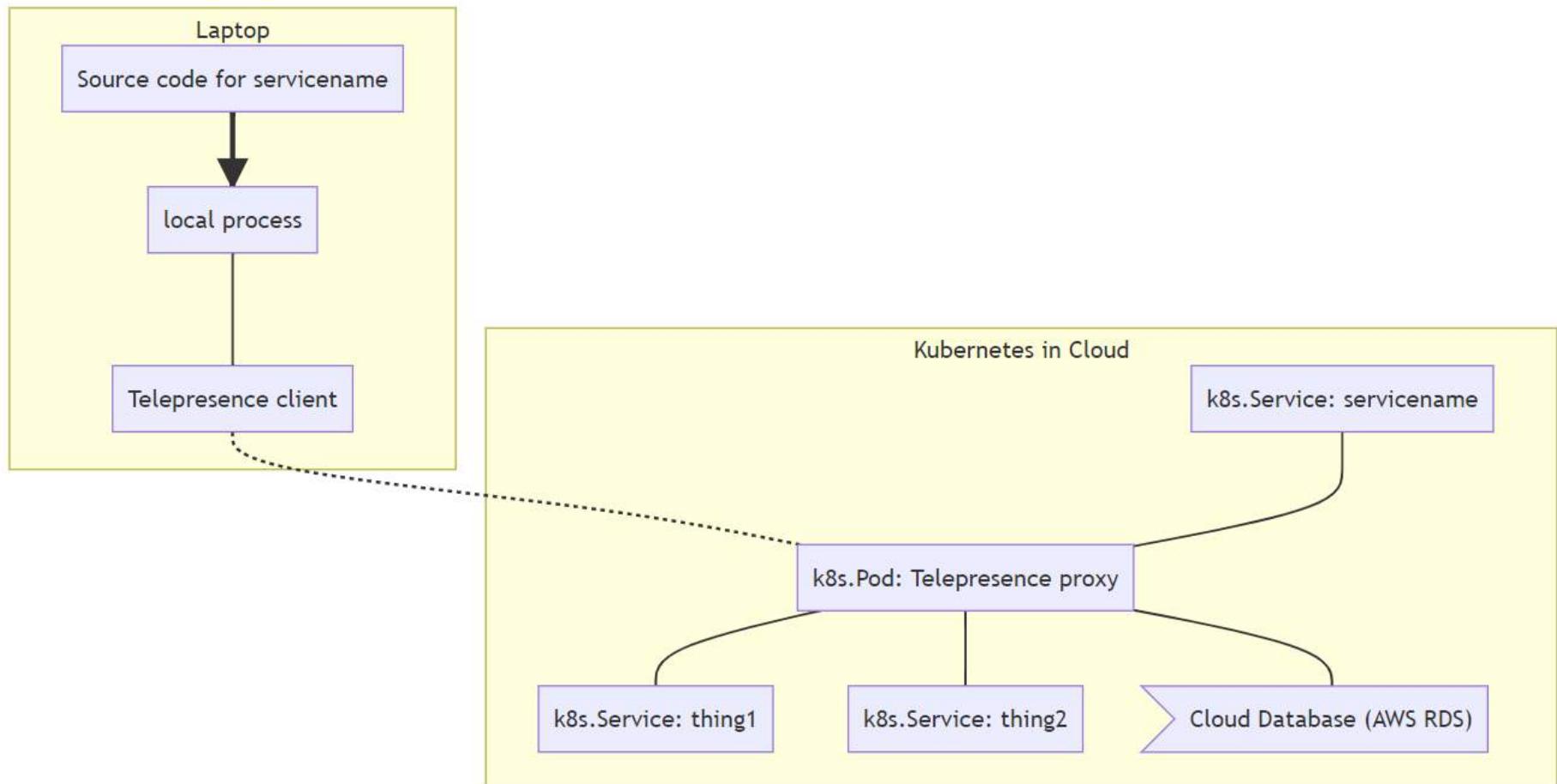
## debugging

Popular tools:

- Telepresence
- Bridge to Kubernetes

# External tools

## debugging





Tools for devs

# Autocomplete

```
# Poweshell
Import-Module PSKubectlCompletion
Set-Alias k -Value kubectl
Register-KubectlCompletion

# Bash
source <(kubectl completion bash)
```

# kubectx & kubens

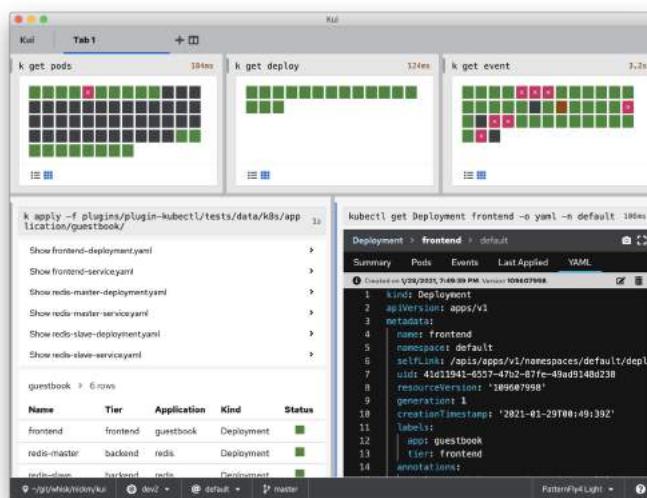
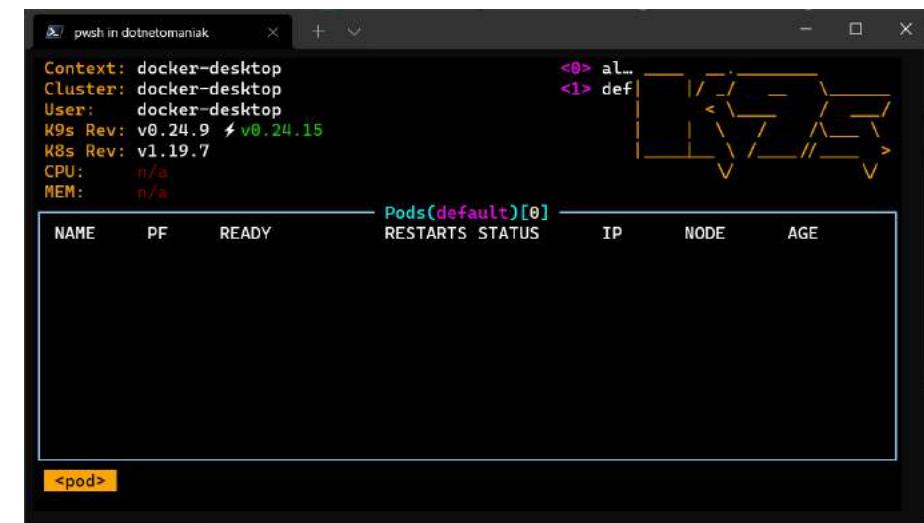
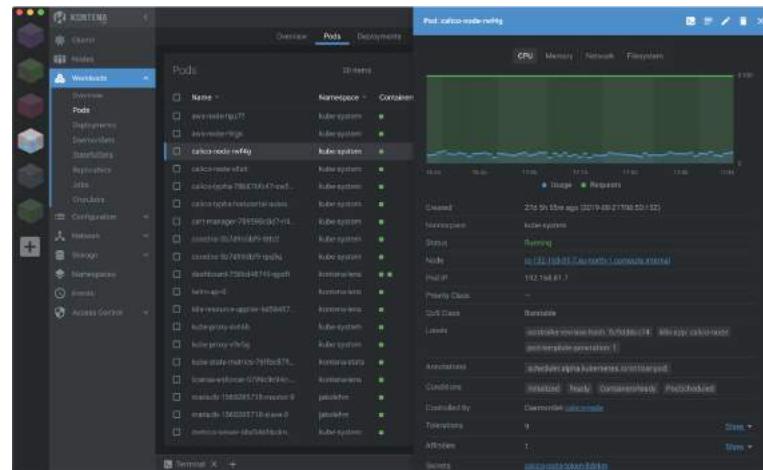
```
➜ ~ kubectx
coffee
minikube
test
➜ ~ kubectx coffee
Switched to context "coffee".
➜ ~ █
```

```
➜ ~ kubens
default
kube-public
kube-system
➜ ~ kubens kube-system
Context "test" modified.
Active namespace is "kube-system".
➜ ~ kubectl get█
```

# oh-my-....

- oh-my-zsh
- oh-my-bash
- oh-my-posh + magic tooltips

# IDE: Lens/k9s/kui



**PRIVATE**

Container Registry

# demo - private registry

upload image to private registry

- Login into private registry with docker

*docker login devszkolak8s.azurecr.io -u USER -p PASS*

- Tag image

*docker tag gutek/dumpster:v1*

*devszkolak8s.azurecr.io/YOUR\_NAME/dumpster:v1*

- Push image

*docker push*

*devszkolak8s.azurecr.io/YOUR\_NAME/dumpster:v1*

# demo - private registry p.2

register private registry in K8S

Command

```
kubectl create secret docker-registry regcred  
--docker-server=devszkolak8s.azurecr.io  
--docker-username=USER  
--docker-password=PASS  
--docker-email=email@email.com
```

# demo - private registry p.3

deploy pod

- deploy pod with image from private registry

containers:

```
- name: private-reg-container  
  image: <your-private-image>
```

imagePullSecrets:

```
- name: regcred
```

# discussion - private container registry

- Secrets can be useful :)
- Easy
- ...

The background of the image is a dense, intricate network of industrial piping. The pipes are primarily made of a dark, possibly copper or steel material, with some sections appearing to be flexible hoses. They are arranged in a complex web, with many pipes running horizontally across the frame and others forming vertical or diagonal paths. The lighting is dramatic, with strong highlights and shadows that emphasize the metallic texture and depth of the pipes.

# CI/CD

making a pipeline

# CI /CD - build

1. Build your image
2. Tag it with build number:
  - {build\_number}
  - develop-{build\_number}
3. Push it to registry
4. After build export == build artifacts:
  - tag
  - deployment.yaml
  - docker image should be in registry

# CI /CD - release

1. Extract artifacts (image tag + deployment.yaml)
2. Replace image tag in deployment.yaml
3. Validate YAML (e.g. kubeval)
4. Set variables+configs+secrets if needed
5. Run apply

*kubectl apply -f deployment.yml*

6. Verify deployment with

*kubectl rollout status deployment NAME\_FROM\_YML*

7. If deployment doesn't succeed in time-period revert rollout and mark release as failed:

*kubectl rollout undo deployment NAME\_FROM\_YML*

# demos - CI /CD - build

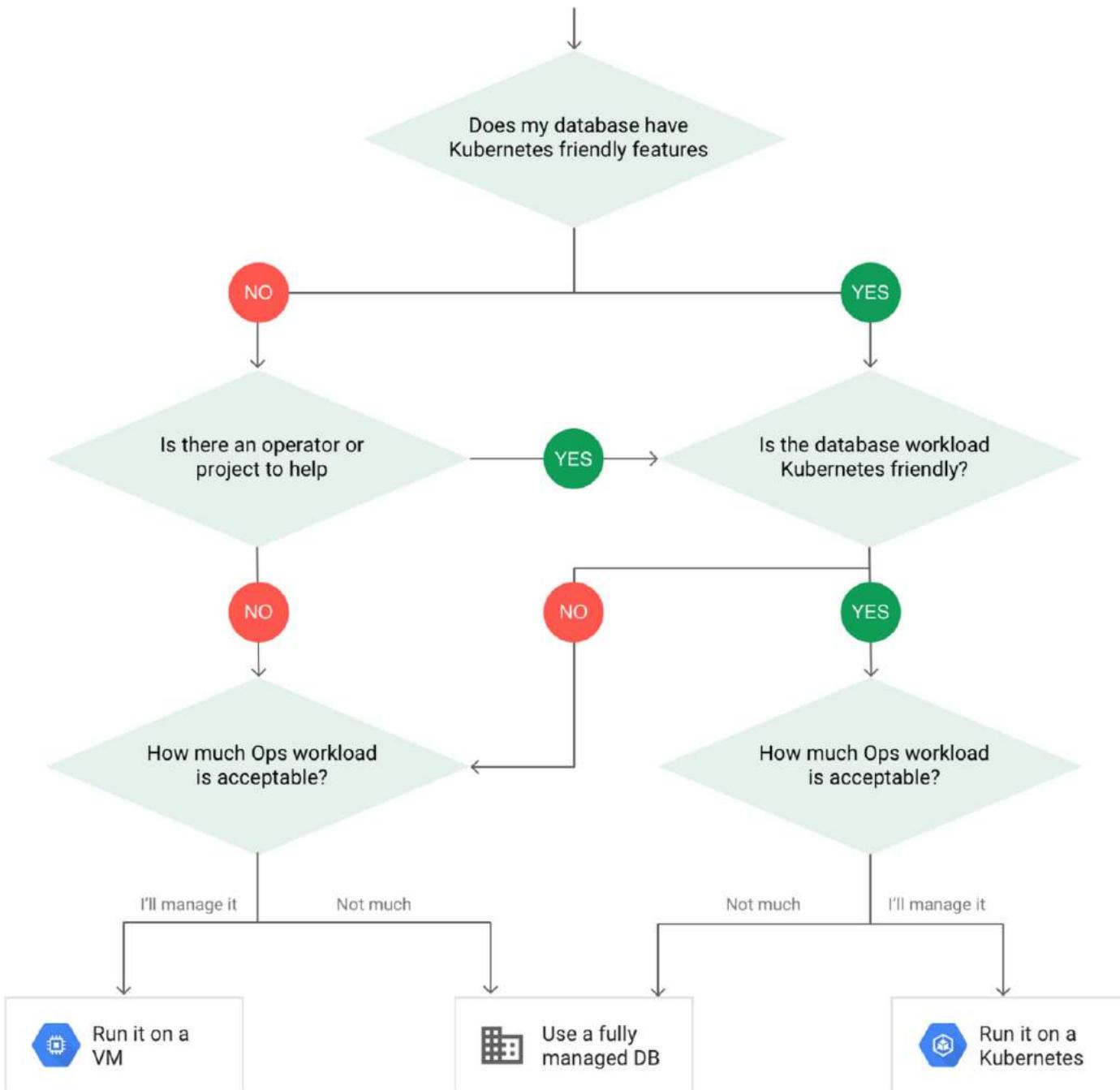
[https://dev.azure.com/poznajkubernetes/\\_git/pkad?path=%2Fazure-pipelines.yml](https://dev.azure.com/poznajkubernetes/_git/pkad?path=%2Fazure-pipelines.yml)

<https://gitlab.com/poznajKubernetes/pkad-gitlab/-/blob/master/.gitlab-ci.yml>



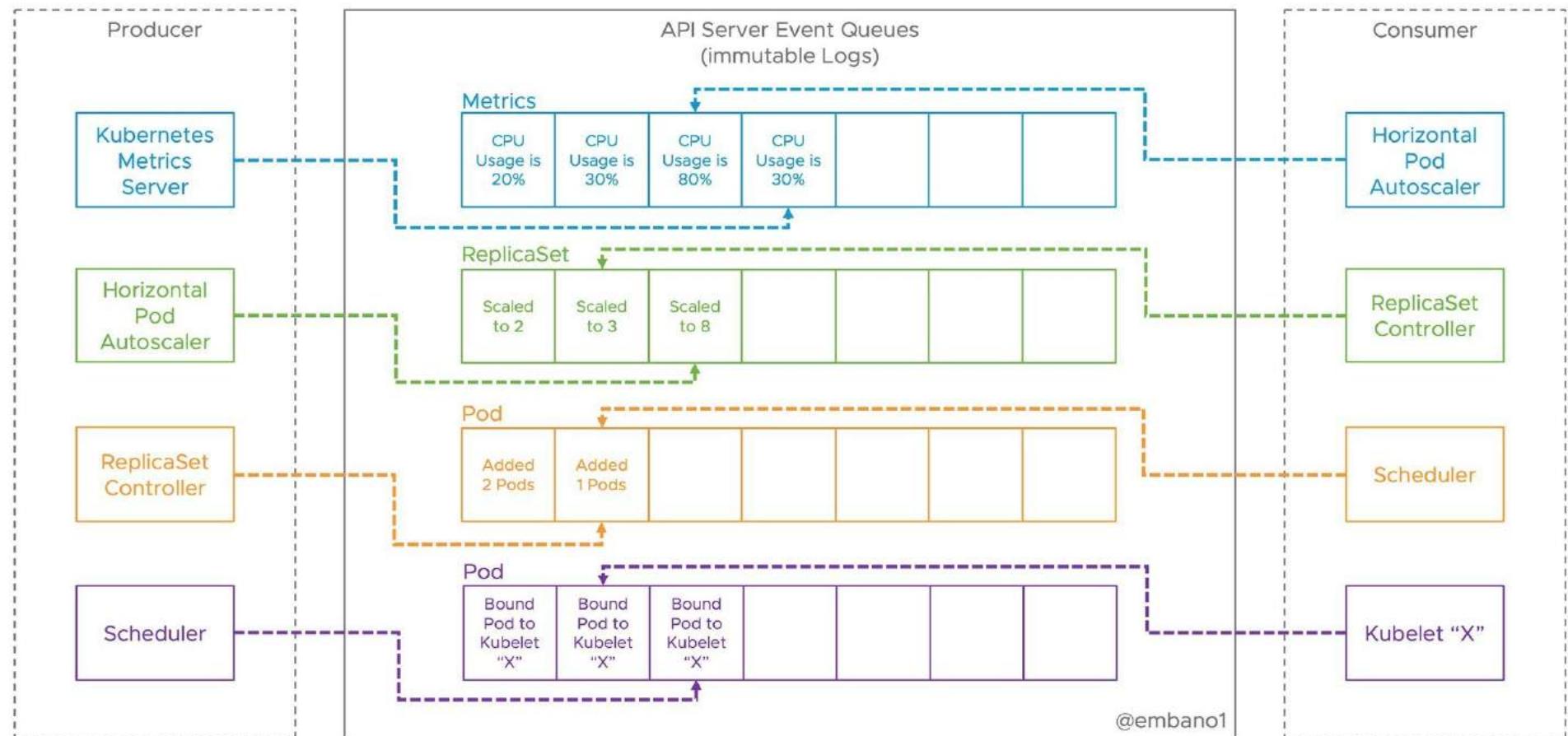
# Databases on K8s

To run or not to run



An aerial photograph of a city at night, looking down from a high altitude. The city is densely packed with buildings of various heights, illuminated by streetlights and building lights. In the center of the frame is a prominent skyscraper with a distinctive circular observation deck or antenna at the top, which is brightly lit. The surrounding buildings are mostly dark, creating a strong contrast with the central light source.

Different view



source



# CKAD

## what is needed

- Core Concepts:
  - Understand Kubernetes API primitives
  - Create and configure basic Pods
- Multi-Container Pods
  - Understand Multi-Container Pod design patterns (e.g. ambassador, adapter, sidecar)
- Pod Design
  - Understand how to use Labels, Selectors, and Annotations
  - Understand Deployments and how to perform rolling updates
  - Understand Deployments and how to perform rollbacks
  - Understand Jobs and CronJobs
- State Persistence
  - Understand PersistentVolumeClaims for storage
- Configuration
  - Understand ConfigMaps
  - **Understand SecurityContexts**
  - Define an application's resource requirements
  - Create & consume Secrets
  - **Understand ServiceAccounts**
- Observability
  - Understand LivenessProbes and ReadinessProbes
  - Understand container logging
  - Understand how to monitor applications in Kubernetes
  - Understand debugging in Kubernetes
- Services & Networking
  - Understand Services
  - **Demonstrate basic understanding of NetworkPolicies**

The background of the image is a wide, calm body of water, likely the sea or a large lake, with gentle ripples across its surface. The horizon is visible in the distance, where the water meets a clear, pale blue sky. There are no clouds, birds, or other elements in the background.

# Summary

# Survey

[https://forms.gle/RsTgCg1NBDp  
C2okg9](https://forms.gle/RsTgCg1NBDpC2okg9)

After workshop

# Resources

- Kubernetes documentation
- Kubernetes słownik
- and everything that is on following slides



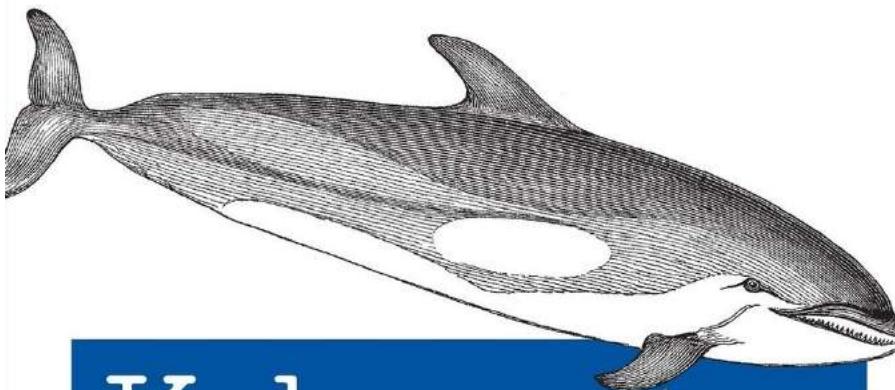
# Kubernetes IN ACTION

Marko Lukša

MANNING

Amazon, Manning

O'REILLY®



# Kubernetes Up & Running

DIVE INTO THE FUTURE OF INFRASTRUCTURE

Kelsey Hightower,  
Brendan Burns & Joe Beda

Amazon

O'REILLY®



# Kubernetes Cookbook

BUILDING CLOUD NATIVE APPLICATIONS

Sébastien Goasguen & Michael Hausenblas

Amazon

O'REILLY®



# Designing Distributed Systems

PATTERNS AND PARADIGMS FOR SCALABLE, RELIABLE SERVICES

Brendan Burns

Amazon (\$\$\$), Microsoft (Free)

<https://www.youtube.com/embed/sfQir0mij5g?enablejsapi=1>

Playlist



## Introduction to Kubernetes

Want to learn Kubernetes? Get an in-depth primer on this powerful system for managing containerized applications.



edX free course

# Piotr Stapp

- twitter: @ptrstpp950
- stapp.space
- piotr.stapp@gmail.com

# Jakub Gutkowski

- twitter: @gutek
- GutkowskiJakub
- kuba@gutek.pl