Kubernetes

an introduction

What is kubernetes

" Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications. " (kubernetes.io)

What is included?

- basic overview
- some general information
- deploy an application into a cluster

What is not included?

- setting up a cluster
- more advanced deployments
- rights and role management
- persistant storage

Content

- What is kubernetes
- Architecture
- CLI - kubectl
- Deploying your application
- What else can you do ?
- Tasks
- Sources

General facts

- started 2014 by Google employees
- third iteration of container ochestrator after Borg and Omega
- very active GitHub community
  - 90.900 commits
  - 66.000 + stars
  - new release every 3 month

Architecture

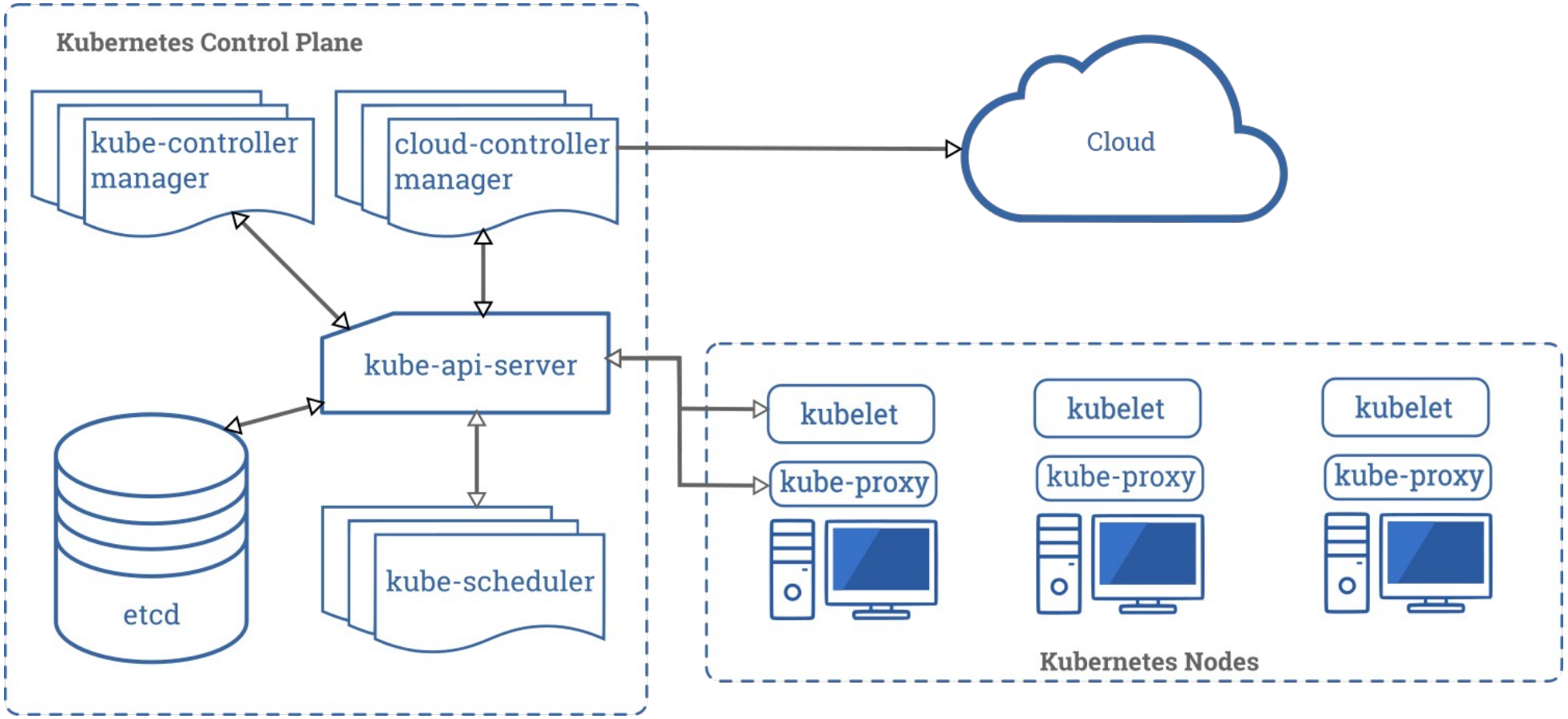
Master

- manage "ressources"
- only few server, but odd number (1,3,5)
- applications neededd:
  - apiserver
  - scheduler
  - controller-manager
  - etcd
  - (container runtime)
  - (kubelet)

Worker

- run workload
- typically much more server than master
- applications needed:
  - container runtime
  - kubelet

Architecture



kubectl

- CLI for interacting with cluster via apiserver
- available for ...
  - Windows (Download)
  - Linux (Download)
  - Mac (Download)
- or follow install instructions from official docs

kubectl - useful commands

Create "something" in your cluster

```
$ kubectl apply -f <resource>.yaml
```

Delete "something" in your cluster

```
$ kubectl delete (pod <name>/-f <resource>.yaml)
```

Show all applications (pods) in your namespace

```
$ kubectl get pods (-o wide)
```

Inspect an application (pod)

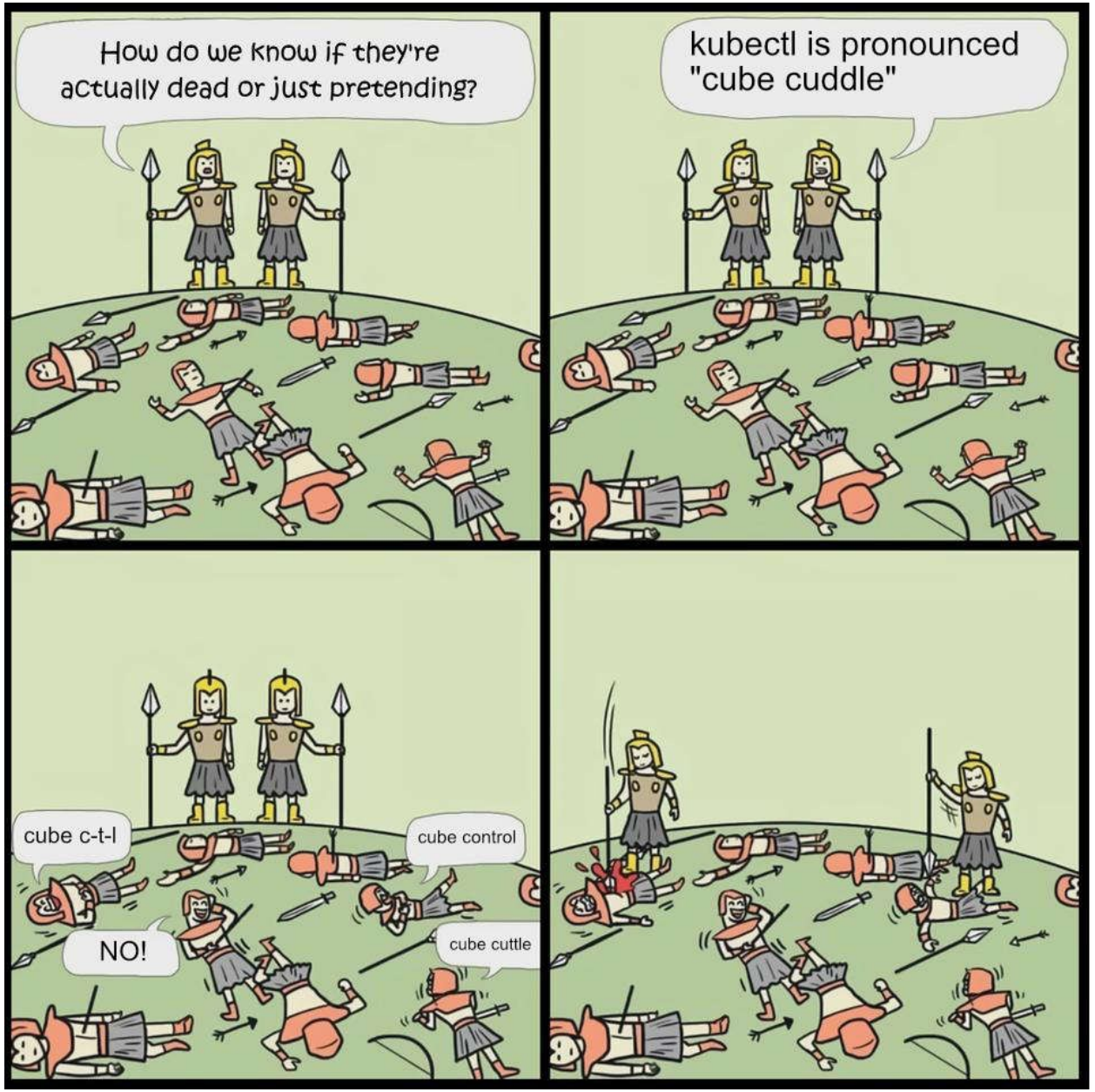
```
$ kubectl describe pod <name>
```

```
$ kubectl logs pod <name> (<container>)
```

Offtoptic

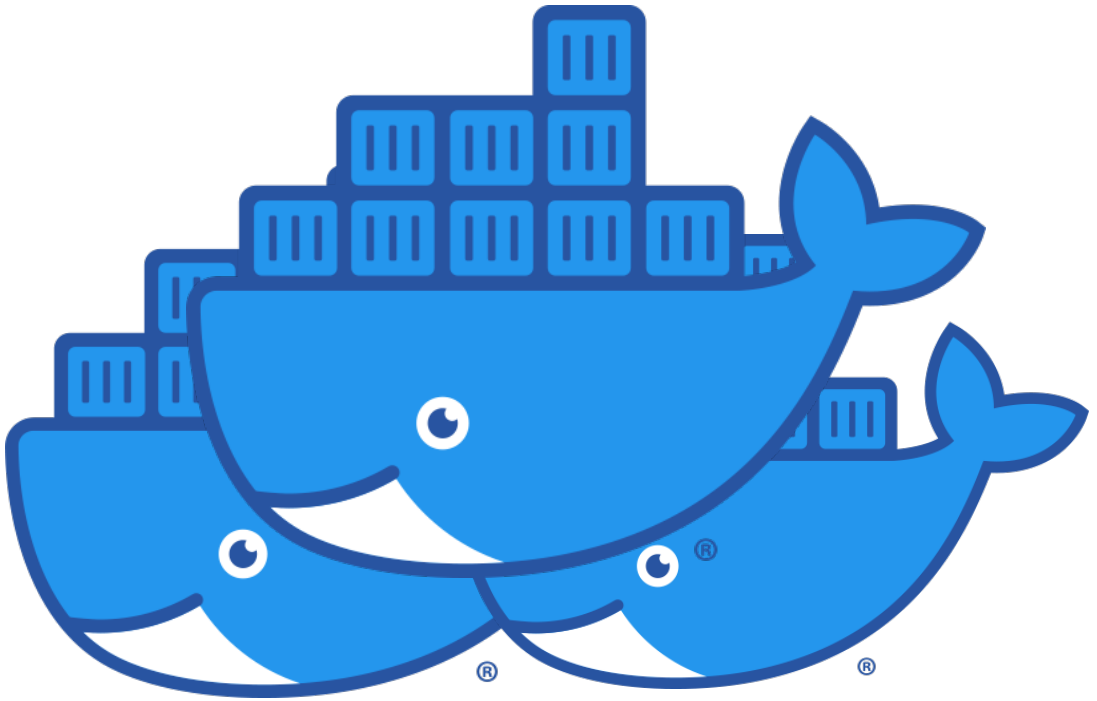
kubectl - pronunciation

- kube-c-t-l ?
- kube-control ?
- kube-cuddle ?
- kube-cuttle ?
- ...



Deploying

your application



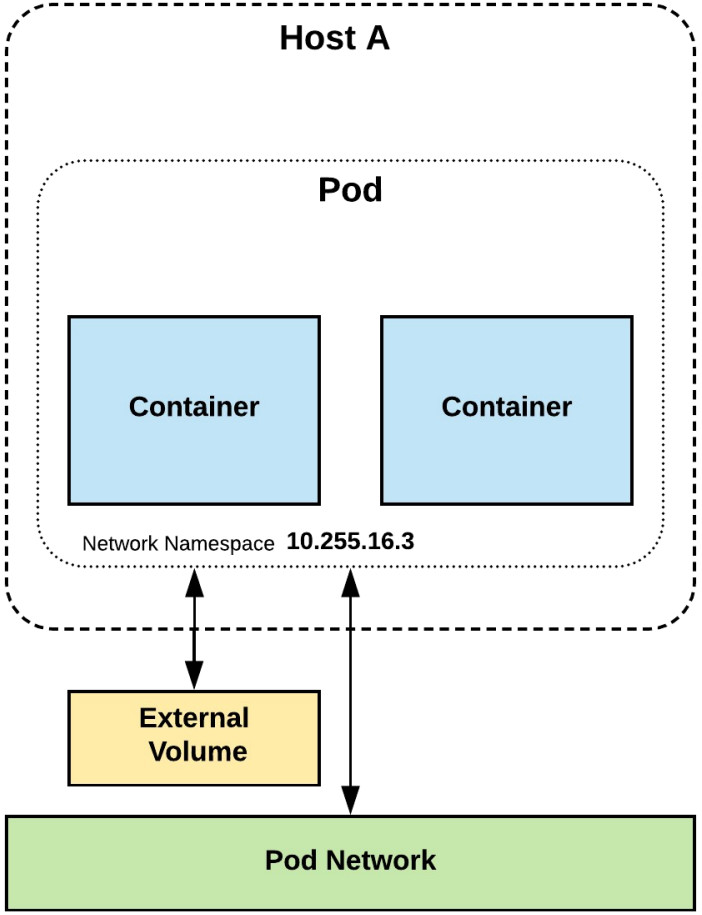
Terminology

- Pods
- Labels
- Deployments
- Services
- Ingress
- Volumes
- ConfigMaps / Secrets
- Many more ...

Pods - the actual containers ?

- smallest unit inside a kubernetes cluster
- consits of one or multiple containers
- share volumes, network, rights, ...

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: hello-world-kubernetes-frontend
  labels:
    app: hello-world-kubernetes
    stage: frontend
spec:
  containers:
  - name: nginx
    image:
docker.pkg.github.com/mm53/hello-world-
kubernetes/frontend:latest
    ports:
    - containerPort: 8080
```



pod vs. docker-compose

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: wordpress-example

spec:
  containers:
    - name: wordpress
      image: wordpress
      ports:
      - containerPort: 8080

    - name: mysql
      image: mysql
      volumeMounts:
        - name: db-data
          mountPath: /var/lib/mysql/data

  volumes:
    - name: db-data
      emptyDir: {}


# replications are not possible on pod level

# all pods are in the same network
```

```yaml
version: "3.8"

services:
  wordpress:
    image: wordpress
    ports:
      - "8080:80"
    networks:
      - overlay
    deploy:
      mode: replicated
      replicas: 2
      endpoint_mode: vip

  mysql:
    image: mysql
    volumes:
       - db-data:/var/lib/mysql/data
    networks:
       - overlay
    deploy:
      mode: replicated
      replicas: 2
      endpoint_mode: dnsrr

volumes:
  db-data:

networks:
  overlay:
```

Deployments - the better option

- Pods aren't durable on their own
- → recommend to use controllers (usually Deployments)
- manage number of replicas of your application through ReplicaSets
- manage update strategie and rollback


```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-example
spec:
  replicas: 3
  revisionHistoryLimit: 3
  selector:
    matchLabels:
      app: hello-world-kubernetes
      stage: frontend
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  template:
    metadata:
      name: hello-world-kubernetes-frontend
      labels:
        app: hello-world-kubernetes
        stage: frontend
    spec:
      containers:
      - name:frontend
        image:
docker.pkg.github.com/mm53/hello-world-
kubernetes/frontend:latest
        ports:
        - containerPort: 8080
```

Services - connecting your pods

- select pods based on labels
- loadbalance requests between pods
- 4 different types:
  - ClusterIP
  - NodePort
  - LoadBalancer
  - ExternalName
- provide a unique IP inside the cluster
- provide a DNS name to access app
  <service-name>.<namespace>.svc.cluster.local

```yaml
apiVersion: v1
kind: Service
metadata:
  name: hello-world
spec:
  type: NodePort
  selector:
    app: hello-world-kubernetes
    stage: frontend
  ports:
  - protocol: TCP
    port: 8080
    targetPort: 80
```

Exposing over http - ingresses

- routes HTTP or HTTPS traffic from outside the custer to services within the cluster
- routing based on
  - URL-path
  - virtual hosts
- SSL termination or passthrough
- more options depending on Ingress Controller (e.g. Nginx, Traefik, Gloo, ...)

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: hello-world-ingress
spec:
  rules:
  - http:
      paths:
      - path: /
        backend:
          serviceName: hello-world
          servicePort: 80
---
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: hello-world-ingress
spec:
  backend:
    serviceName: hello-world
    servicePort: 80
```

ConfigMaps - decouple configurations

- allows to store "data" wihtin the cluster
- multiple ways to inject into pod :
  - mounted as file
  - in environement variable
  - in commands
- → used for config or other static files

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: hello-world-greeting
data:
  greeting.html: |
  <div class=""greeting>
    Now you will see another
message ... <br/>
  </div>
```

- also encrypted possible as Secret
- → used for passwords, certificates, ...

ConfigMaps - usage

Create from file :

```
$ kubectl create configmap
  <name> --from-file=<file>
```

Write yourself :

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: hello-world-config
data:
  BACKEND: hello-world-
backend.default.svc.cluster.local
```

```
...
  containers:
    - name: frontend
      ...
      env:
        - name: BACKEND_URL
          valueFrom:
            configMapKeyRef:
              name: hello-world-config
              key: BACKEND_URL
      volumeMounts:
        - name: templates
          mountPath: /var/www/templates/additional

  volumes:
    - name: templates
      configMap:
        name: hello-world-greeting

        # optional, if not present all files (key-

          value-pairs) will be added
        items:
          - key: greeting.html
            path: greeting.html
```

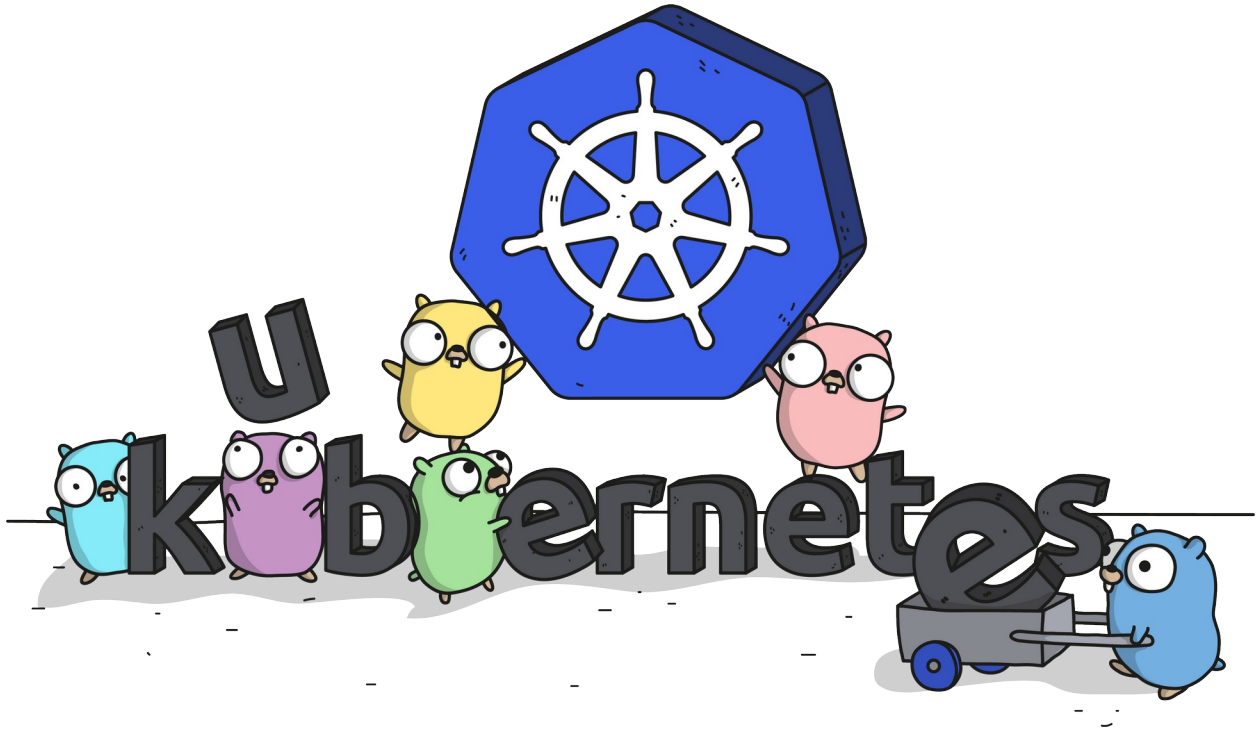
Whats next ?


Only the
tip of the iceberg

Kubernetes can do even more

- advanced container settings
- using differnt ways to deploy pods (DeamonSets, StatefulSets, ...)
- group applications using namespaces
- manage rights using role-based-authentification
- using persistant volumes to store data
- create your own api ressources and operators
- secure incluster communication using service meshes
- and much more ...

Questions ?



Start hacking ...

Getting started

Use online playground ...

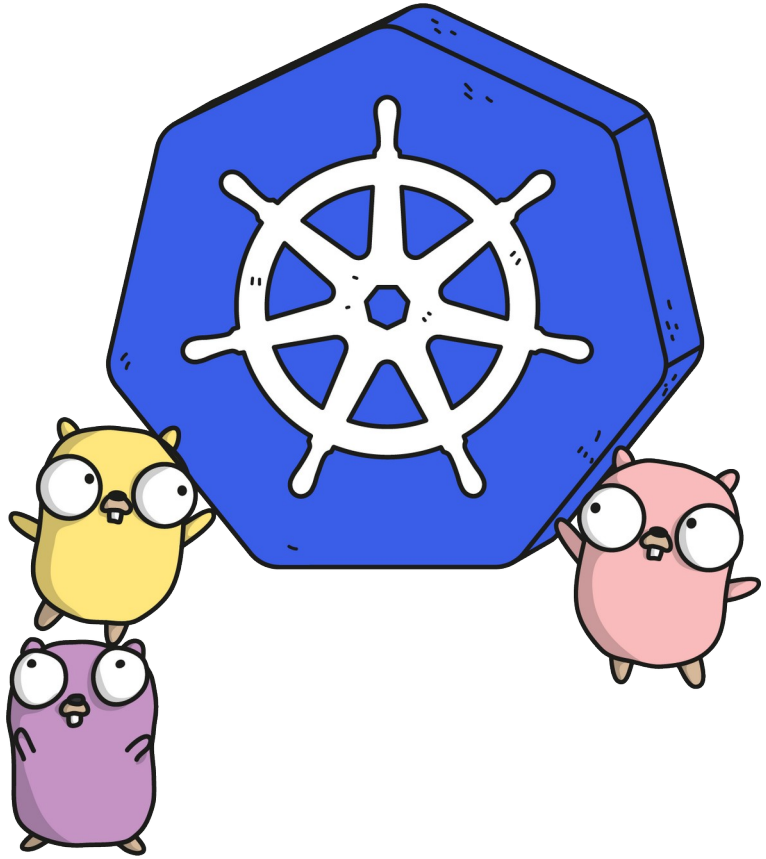
... or setup your own "cluster" with ...

... Minikube ...

... Kind ...

... kubeadm ...

... and deploy your own apps



- Tutorial : GitHub
- official online tutorial (includes some other aspects)

Sources

This presentation is based on:

- the official documentation on kubernetes.io
- Introduction to Kubernetes Workshop (full-day) by Bob Killen and Jeff Sica (July 17, 2018)
- kubernetes repo on GitHub
- experiences from work
- docker-compose documentation for example snippet

Sources - images

- kubernetes logo : GitHub
- cluster overview : kubernetes.io
- kubectl logo : GitHub
- kubectl image : Twitter
- docker logo : docker.com
- pod and deployment overview : Introduction to Kubernetes Workshop (full-day)
- kubernetes gopher : GitHub
- iceberg : Pixabay