**Criar um cluster**

k3d cluster create NAME

k3d cluster list

k3d cluster create meucluster --no-lb (sem o load balancer)

k3d cluster delete

k3d cluster create meucluster --servers 3 --agents 3

k3d cluster create meucluster --servers 1 --agents 2

k3d cluster create meucluster --servers 1 --agents 1 -p "8080:30000@loadbalancer"

k3d cluster create meucluster --servers 1 --agents 1 -p "8080:30000@agent:0" --no-lb

k3d cluster create meucluster --servers 3 --agents 3 -p "8080:30000@loadbalancer" -p "81:30001@loadbalancer" -p "8001:30002@loadbalancer" (modelo)

k3d cluster create meucluster -p "8080:30000@loadbalancer" -p "81:30001@loadbalancer" -p "8001:30002@loadbalancer" -p "5432:30003@loadbalancer"

k3d cluster create meucluster --servers 1 --agents 1 -p "8080:30000@loadbalancer" -p "81:30001@loadbalancer" -p "8001:30002@loadbalancer" -p "5432:30003@loadbalancer"

------------------------------------------------------------------------------------------------------------------------------------------------

docker build -t msergiost/apilist:v2.3 .

docker push msergiost/apilist:v2.3

kubectl create secret generic db-apilist-secret --from-literal=DB\_HOST=db-service.nsdatabase.svc.cluster.local --from-literal=DB\_USER=postgres --from-literal=DB\_PASSWORD=1234

kubectl create secret generic db-apilist-secret -n nsapilist --from-literal=DB\_HOST=db-service.nsdatabase.svc.cluster.local --from-literal=DB\_USER=postgres --from-literal=DB\_PASSWORD=1234

------------------------------------------------------------------------------------------------------------------------------------------------

kubectl apply -f deploy.yaml

**POD(como se fosse uma máquina virtual)**

Pode ter vários conteiners dentro dele. Todos possuem o mesmo endereço de IP como se estivessem dentro de uma máquina. O sistema de arquivos pode ser compartilhado entre os containers

Para escalar o pods é que são duplicados, portanto, dentro de cada pod o ideal é ter apenas um app



Desta maneira os pods tem que ter somente um app

Para escalar as aplicações tem que ficar em pods diferentes.





Dentro do POD pode haver containers auxiliares, por exemplo para coleta de logs.



kubectl api-resources - lista todos os objetos que posso criar no cluster kubenetes



Criar arquivo manifest

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: meupod**

**spec:**

**containers:**

**- name: web**

**image: kubedevio/web-color:blue**

**ports:**

**- name: http**

**containerPort: 80**

**protocol: TCP**

kubectl apply -f deployPods.yaml

------------------------------------------------------------------------------------------------------------------------------------------------

**ReplicaSet**

É um objeto que garante que o número especificado de réplicas de Pods esteja sempre em execução. Mantem a escalabilidade e disponibilidade das Apps.



**Funções:**

monitora o estado dos pods que ele gerencia

mantem o número de pods desejados

cria pods conforme modelo definido

distribui os pods pelo cluster

monitora continuamente os pods

aumenta ou diminui o numero e pods

O ReplicaSet é gerenciado por um deployment

**Labels**



**Deployments**







Funções

escalabilidade

resiliencia

atualizações

rollback

kubectl rollout history deployment meudeployment

kubectl rollout undo deploy meudeployment && watch 'kubectl get deploy,pod,rs'

**Service**



**Tipos de services**

**ClusterIp**

Funciona internamente no cluster

**NodePort**



**LoadBalancer**

**ExternalName**



Pode acessar um banco de dados fora do cluster

kubectl api-resources - lista todos os objetos que posso criar no cluster kubenetes

kubectl cluster-info

kubectl get nodes

kubectl get all

kubectl get pod

kubectl describe pod meupod

kubectl get pods -o wide

kubectl get replicaset -o wide

kubectl delete pod meupod

kubectl delete -f deploy.yaml

kubectl apply -f deployReplicaset.yaml && watch 'kubectl get rs,po'

kubectl rollout history deployment meudeployment

kubectl rollout undo deploy meudeployment && watch 'kubectl get deploy,pod,rs'

kubectl get endpoints

kubectl get endpointslice

kubectl run prompt --rm -it --image ubuntu -- /bin/bash

apt update && apt install curl --yes

cat ~/.kube/config

**Retorna o arquivo manifest**

kubectl get deployments -o yaml

kubectl get deploy meudeployment -o yaml > meudeployment.yaml

kubectl logs exec

kubectl get pod

**Recursos**

**Image Pull Police**

Política para baixar a image, pode ser:

* Always
* If not present
* Never

kubectl get pod

kubectl get pod apilist-7f6cdbdf9f-h7l6n -o yaml

**Restart Police**

Somente o container que estiver com erro dentro do pod será reiniciado.

Pode ser as opções:

* Always (Deployment sempre aceita com Always)
* OnFailure
* Never

apiVersion: v1

kind: Pod

metadata:

name: exec

labels:

app: exec

spec:

containers:

- name: exec

image: busybox

command: # => EntryPoint

- echo

args: #=> CMD

- "--->>>>>>>>>>>>>> Hello World"

restartPolicy: Never

kubectl exec -it exec -- /bin/sh

kubectl exec -it exec2 -- /bin/sh

**Self Healing**

**Probes**

* **livenessProbe**: Verifica se o container dentro do pod está vivo e funcionando. Se falhar, o Kubernetes reinicia o container.
* **readinessProbe**: Verifica se o container está pronto para receber tráfego. Se falhar, o Kubernetes remove o pod dos endpoints do serviço, impedindo que receba novas conexões.
* **startupProbe**: Verifica se a aplicação dentro do container inicializou corretamente. Ele desativa livenessProbe e readinessProbe até ter sucesso. Se falhar, o Kubernetes reinicia o container.

1. startupProbe é executado primeiro. Se configurado, ele garante que a aplicação tenha iniciado antes que as sondas de liveness e readiness comecem a ser usadas.
2. Após o sucesso do startupProbe (ou se não estiver configurado), o livenessProbe e o readinessProbe entram em ação para monitorar o estado contínuo do container.
3. livenessProbe foca na saúde interna, enquanto readinessProbe foca na capacidade de servir tráfego.
4. Falhas no livenessProbe levam a reinicializações para tentar corrigir um estado não saudável.
5. Falhas no readinessProbe removem o pod do serviço para evitar o envio de tráfego para uma instância não pronta, sem necessariamente reiniciá-la.

**Repositórios Privados**

Criar secrets

kubectl create secret docker-registry docker-auth --docker-server=https://index.docker.io/v1 --docker-username=msergiost --docker-email=msergiost@hotmail.com --docker-password=123456

kubectl get secret

kubectl get secret docker-auth -o yaml

**Namespace**

kubectl get namespace

kubectl get all -n kube-system

kubectl apply -f deployPostgre.yaml -n homologacao

kubectl delete namespace homologacao

kubectl get -n homologacao all

kubectl get svc -n kube-system

kubectl get svc -n homologacao

[http://service.producao.svc.cluster.local](http://service.producao.svc.cluster.local/)