



Introdução ao Kubernetes

Objetivos

Ao final do curso, espera-se que os alunos saibam:

- O que é o Kubernetes e como ele é usado
- Como criar um cluster simples usando o kubeadm
- Entendam alguns dos conceitos básicos do Kubernetes
- Entendam como os conceitos de *deployments* e *services* podem ajudar a gerenciar aplicativos complexos



AGENDA

1. Revisão Docker
2. O que é Kubernetes?
3. Arquitetura do K8s
4. Conceitos Importantes
5. Instalação dos Componentes
6. Configuração do K8s
7. ReplicaSet
8. Deployment
9. Service
10. Microsserviços

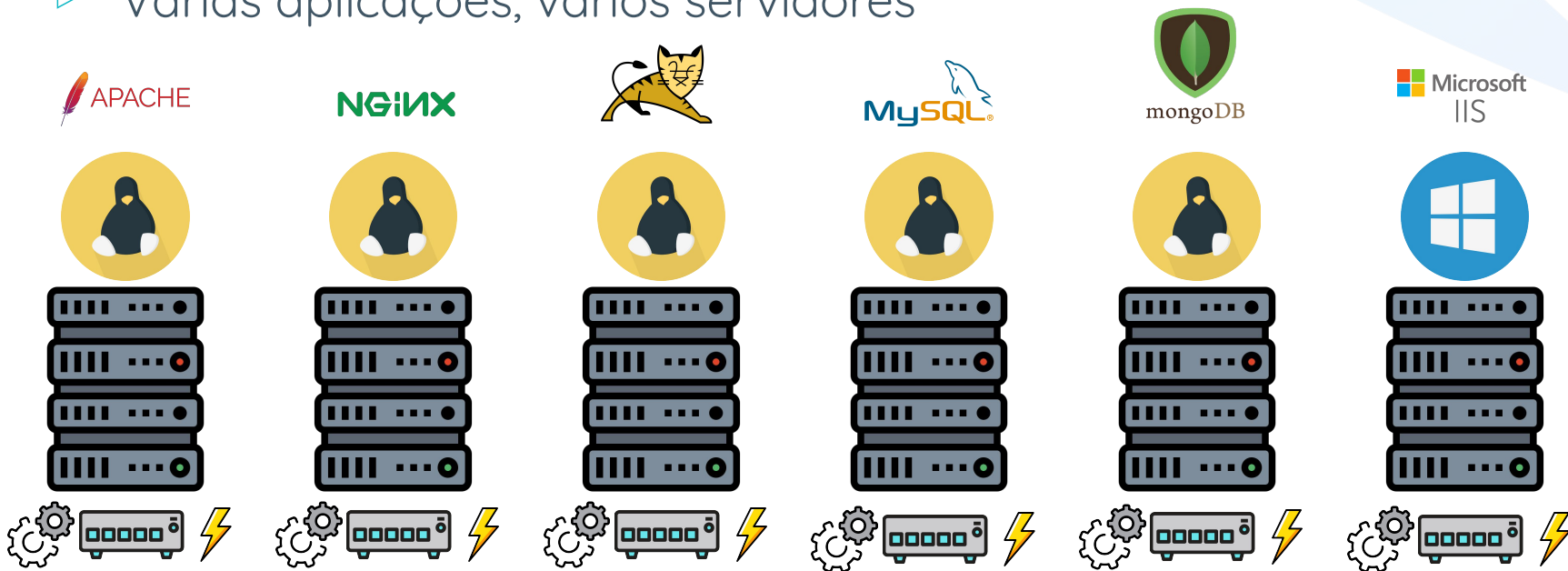
1. Revisão Docker

Como surgiram os containers?

O problema das máquinas virtuais

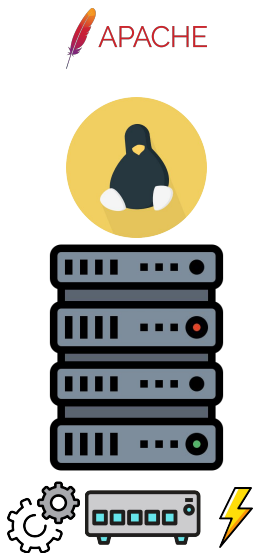
Como hospedamos aplicações antigamente?

- ▶ Várias aplicações, vários servidores



O problema das máquinas virtuais

Capacidade subutilizada!



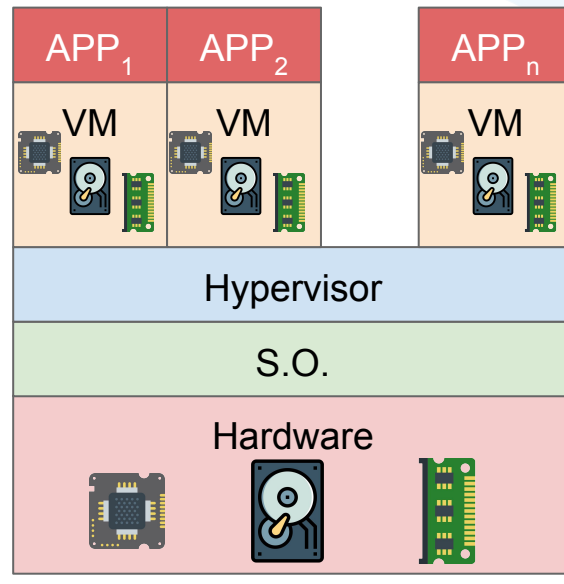
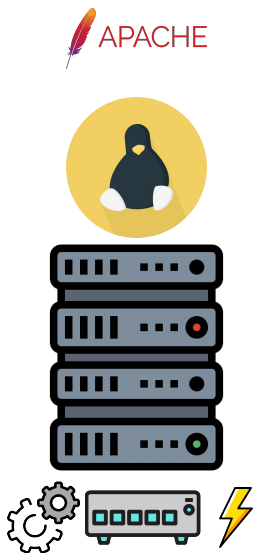
Carga

~15%

- ▶ Muito tempo ocioso
- ▶ Muitos recursos desperdiçados

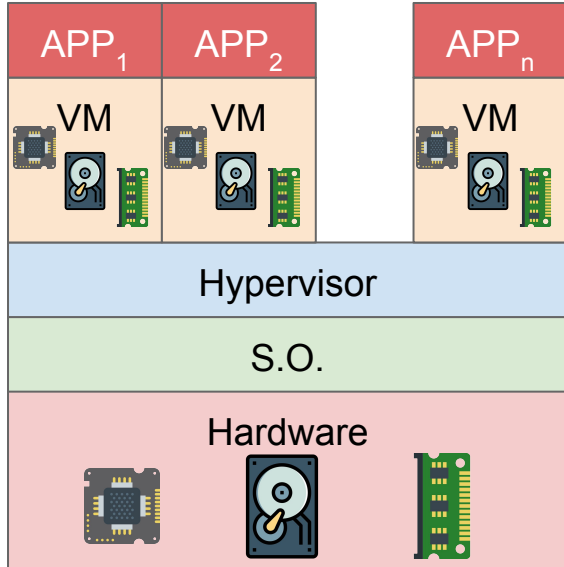
O problema das máquinas virtuais

Melhorando a situação: virtualização

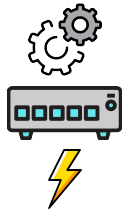


O problema das máquinas virtuais

Melhoria no uso dos recursos de infraestrutura

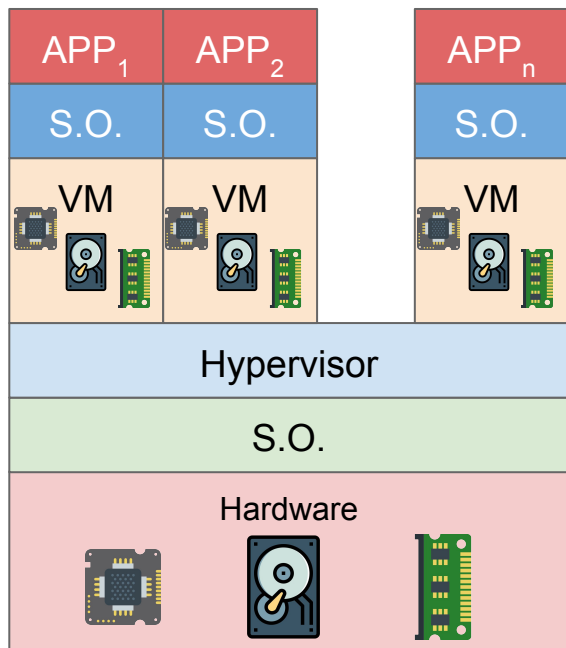


Carga



O problema das máquinas virtuais

Nem tudo são flores... O problema das VM's



1GB de RAM

10GB de
Armazenamento

% de processamento

6GB de RAM

60GB de
Armazenamento

++% de processamento

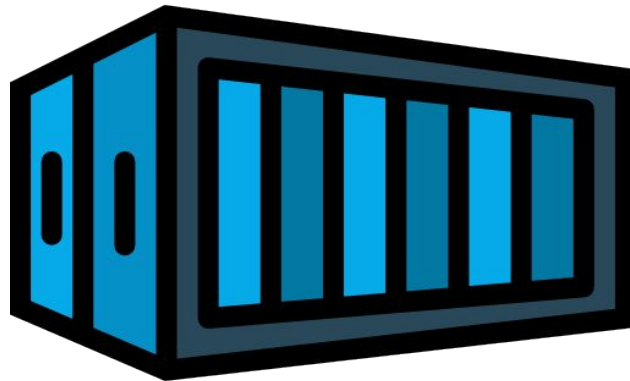
Outros custos de um S.O.

- ▶ Configuração
- ▶ Atualização
- ▶ Segurança

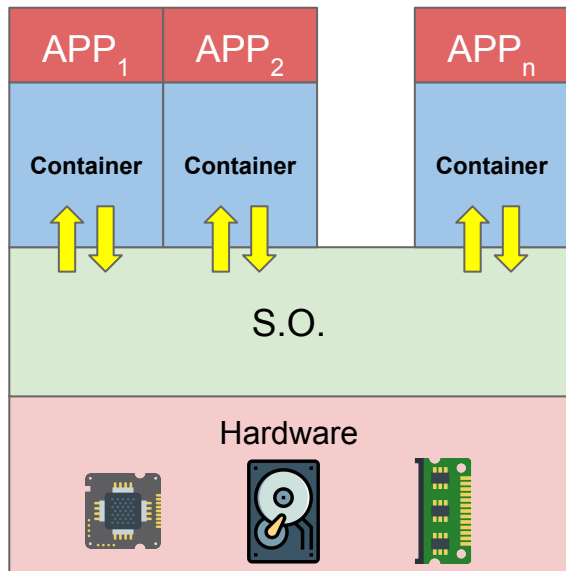
O problema das **máquinas virtuais**

Como melhorar agora?

A era dos **containers**!

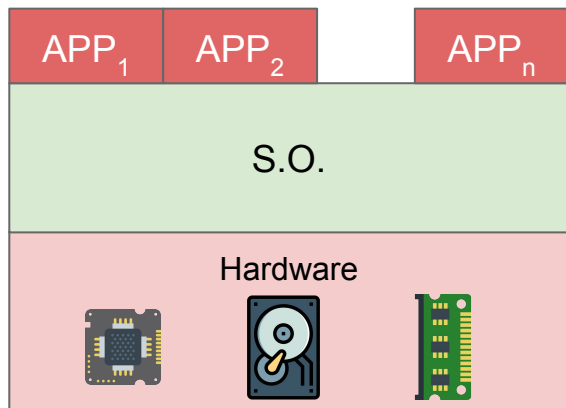


O que é um container?



- ▷ Mais leve
- ▷ Baixo custo na manutenção de múltiplos S.O.'s
- ▷ Mais rápido de subir.

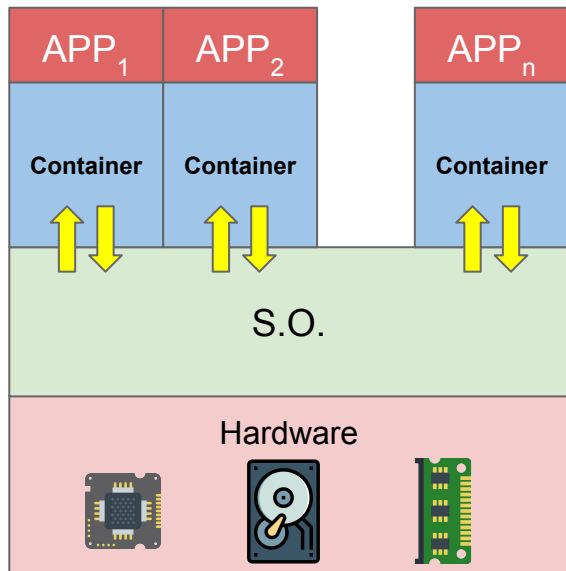
Mas por que precisamos deles?



Os problemas dessa abordagem

- ▶ Dois apps utilizando a mesma porta de rede?
- ▶ E se um app começar a consumir muito de um recurso, como a CPU?
- ▶ E se cada app precisar de uma versão específica de uma linguagem?
- ▶ E se um app congelar todo o sistema?

Utilizando containers

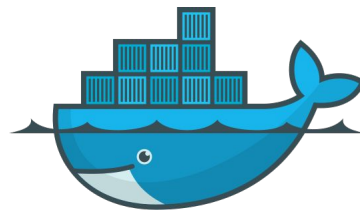
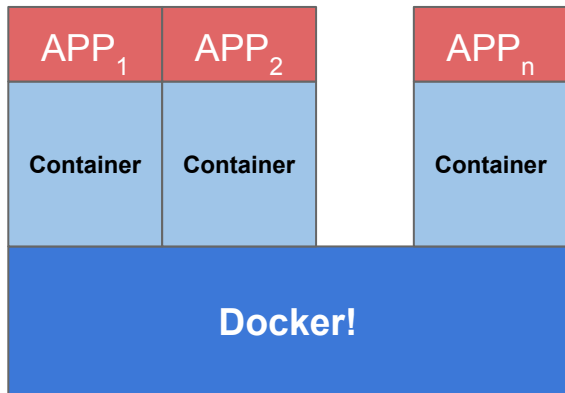


Ganhamos:

- ▶ Melhor controle sobre o uso de cada recurso (CPU, Disco, Rede...)
- ▶ Agilidade na hora de criar ou remover um container
- ▶ Maior facilidade na hora de trabalhar com diferentes versões de linguagens e bibliotecas
- ▶ Mais leves que as VM's

Docker

Tecnologias de containers para prover ferramentas modernas para lançar e executar aplicações



docker

Docker Engine



2. O que é Kubernetes?

Uma perspectiva de alto nível

Kubernetes (K8s)

- ▶ Ferramenta de código aberto para automatizar implantação, gerenciamento e escalonamento de aplicações containerizadas (i.e., **orquestração** de containers).
- ▶ Objetivo: Automatizar a infraestrutura de aplicações e facilitar seu gerenciamento.

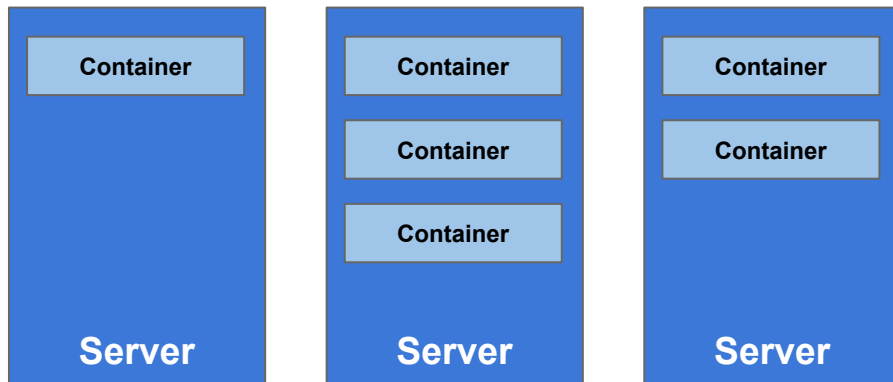


kubernetes

Orquestração

Já vimos o que são containers. Com eles podemos executar uma variedade de componentes de software em um cluster de servidores genéricos, o que ajuda a

- ▶ Prover alta disponibilidade, e
- ▶ Escalar os recursos.



Orquestração

Perguntas:

- ▶ Como garantir que instâncias de um mesmo software sejam espalhadas entre diferentes servidores para atingir alta disponibilidade?
- ▶ Como implantar uma nova versão do software e garantir que ele será atualizado em todo o cluster?
- ▶ Como escalar o software para lidar com aumento da demanda?

c

3. Arquitetura do K8s

Visão geral dos componentes

Orquestração

Perguntas:

- ▶ Como garantir que instâncias de um mesmo software sejam espalhadas entre diferentes servidores para atingir alta disponibilidade?
- ▶ Como implantar uma nova versão do software e garantir que ele será atualizado em todo o cluster?
- ▶ Como escalar o software para lidar com aumento da demanda?

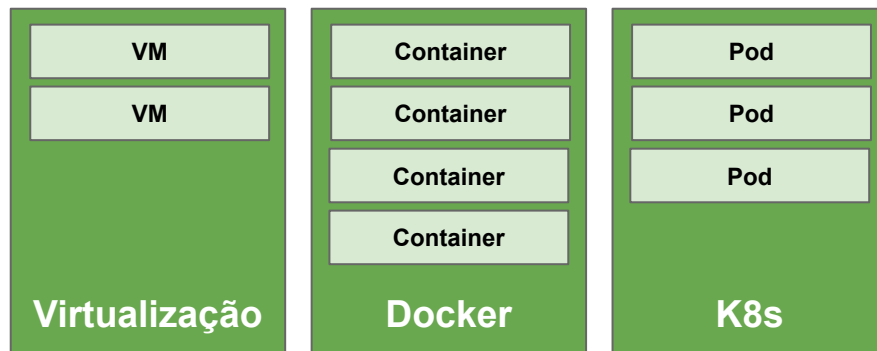
Resposta:

- ▶ Manualmente ou com **K8s**

Orquestrar tarefas de gerenciamento!
É isso que o **K8s** faz!



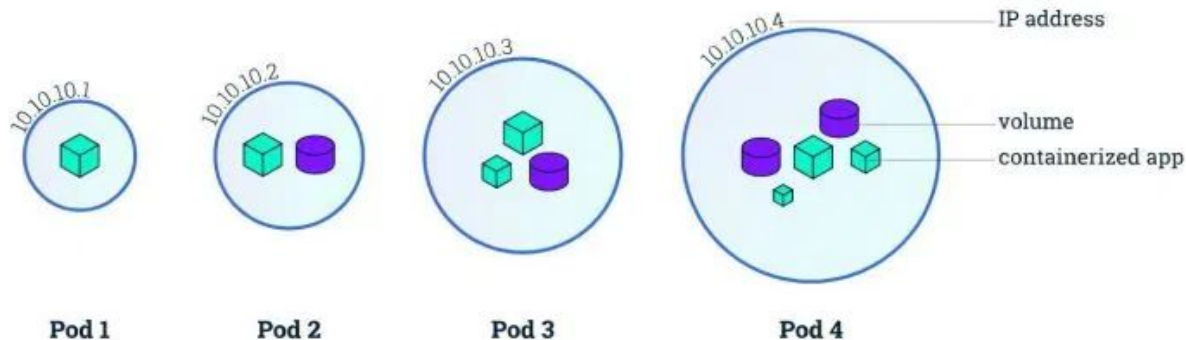
Antes de tudo...



Pod é a menor e mais básica estrutura do K8s

Pod

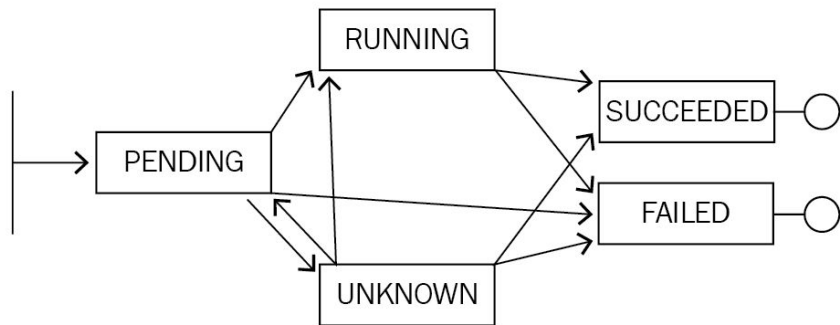
É a menor e mais básica estrutura do K8s, que consiste de um ou mais containers, recursos de armazenamento, e um único endereço IP na rede do cluster K8s.



Ciclo de vida de um Pod

Pending: O Pod foi aceito pelo K8s, mas um ou mais containers ainda não foram criados. Isso inclui tempo de escalonamento e tempo de download da imagem.

Running: O Pod foi alocado em um node e todos os containers foram criados. Pelo menos um container deve estar em execução, ou no processo de (re)inicialização.

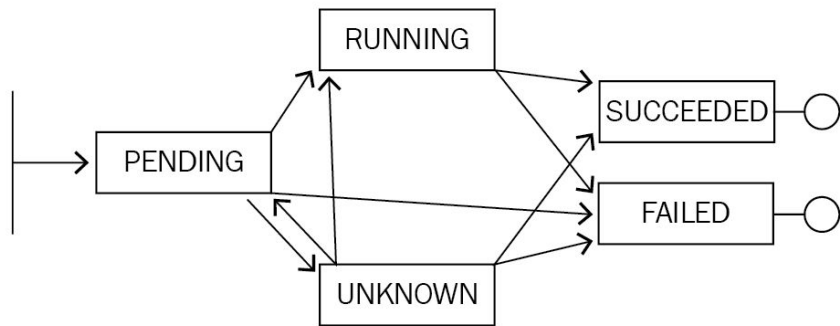


Ciclo de vida de um Pod

Succeeded: Os containers do Pod terminaram com sucesso.

Failed: Os Containers do Pod terminaram, e pelo menos um deles com falha (status != zero ou foi terminado pelo sistema).

Unknown: Por alguma razão, o estado do Pod não pode ser obtido. Tipicamente por causa de erro de comunicação entre o node e o Pod.

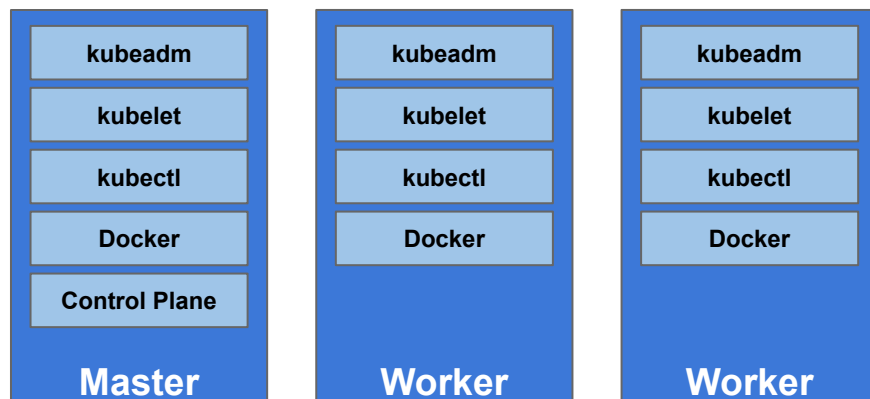


Arquitetura do K8s

O K8s pode ser instalado em modo single node ou cluster

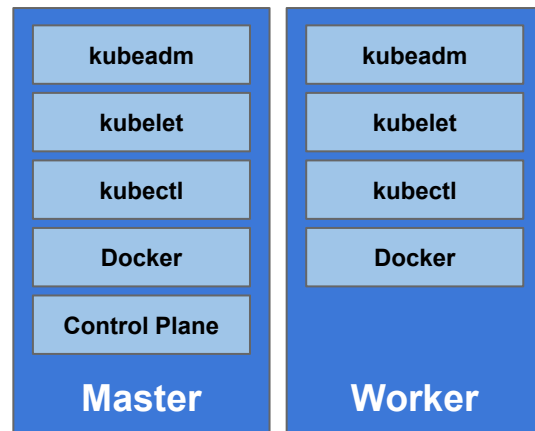
O cluster é formado por nós master e worker

- ▶ O master executa os componentes do plano de controle
- ▶ O worker é quem geralmente executa os containers



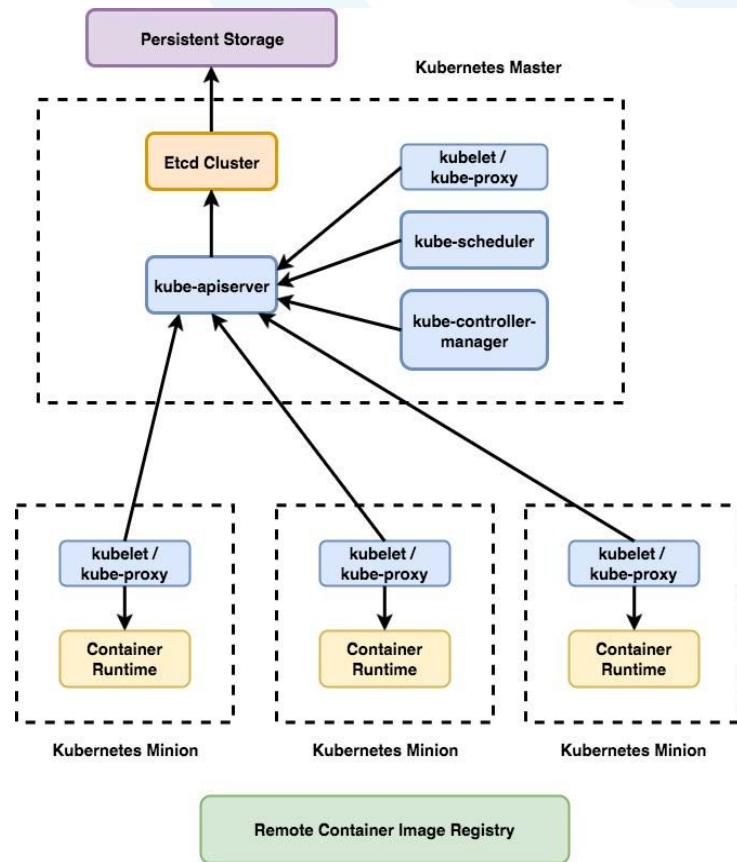
Arquitetura do K8s

- **kubeadm**: ferramenta que automatiza grande parte do processo de criação do cluster.
- **kubelet**: componente essencial do K8s que lida com a execução de pods. Atua como um agente em cada node, intermediando as trocas de mensagens entre API server e Docker runtime.
- **kubectrl**: CLI de interação com o K8s cluster.



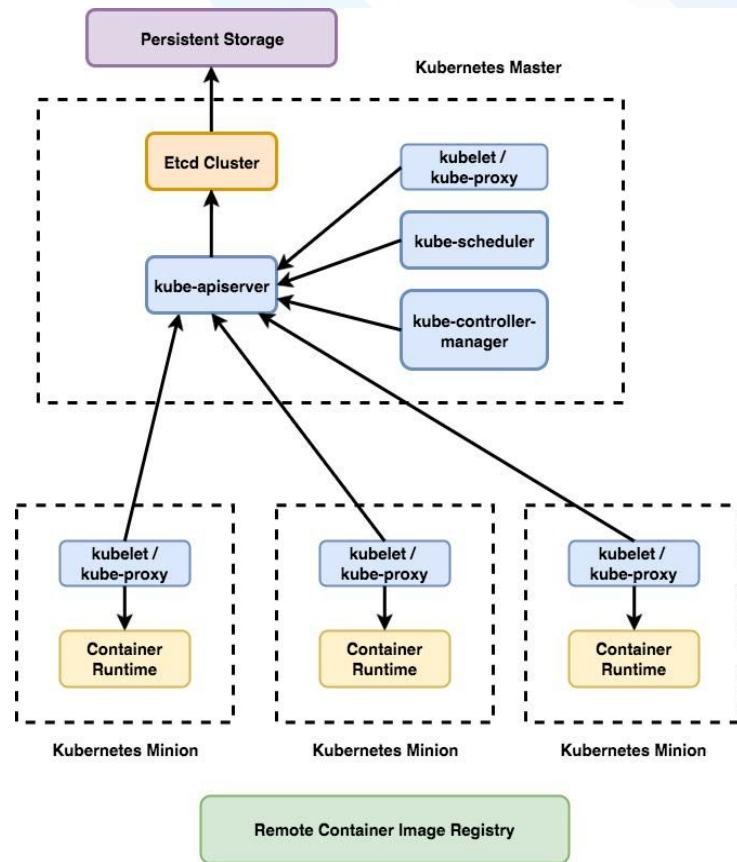
Arquitetura do K8s

- ▶ **etcd**: provê um sistema distribuído e compartilhado para armazenar o estado do cluster
 - ▶ Chave-valor
- ▶ **kube-apiserver**: serve a API do K8s
 - ▶ Baseada em REST
- ▶ **kube-controller-manager**: pacote com diversos componentes de controle



Arquitetura do K8s

- ▶ **kube-scheduler**: escalona os Pods para serem executados nos nodes
- ▶ **kube-proxy**: trata da comunicação entre nodes, adicionando regras ao firewall



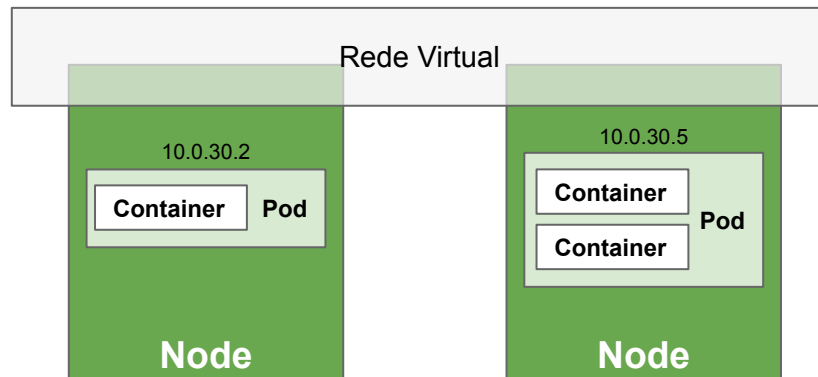
Rede no K8s

O modelo de redes do K8s envolve a criação de redes virtuais no cluster.

Cada Pod do cluster tem um endereço IP único e pode se comunicar com qualquer outro Pod do cluster, mesmo aqueles que são executados em outros nós.

Plugins:

- ▶ Flannel
- ▶ Weave





c

4.

Conceitos importantes do K8s

Como as coisas funcionam...

Arquivo de manifesto/especificação

Manifestos podem ser escritos usando YAML ou JSON, mas YAML é mais utilizado porque é mais fácil de ser lido por humanos, além da possibilidade de adicionar comentários.

```
apiVersion: v1
kind: Pod
metadata:
  name: kuard
spec:
  containers:
  - image: gcr.io/kuar-demo/kuard-amd64:blue
    name: kuard
    ports:
    - containerPort: 8080
      name: http
      protocol: TCP
```

Além de Pods

Diversos recursos podem ser manipulados:

- ▶ Pods
- ▶ Nodes
- ▶ Deployment
- ▶ ReplicaSet
- ▶ Service

Eles são considerados objetos no K8s



Kubectl e as maneiras de interação

Há duas maneiras básicas de interagir com o Kubernetes:

- ▶ Imperativa: através de diversos parâmetros do kubectl
 - ▶ Diz ao K8S o que fazer
 - ▶ Boa para usar quando se está aprendendo, para fazer experimentos interativos ou debugar serviços em produção.
- ▶ Declarativa: escrevendo manifestos e os usando com o comando *kubectl apply*.
 - ▶ Diz ao K8s o que você quer
 - ▶ Melhor para implantar serviços de maneira a facilitar a reprodutibilidade.
 - ▶ Recomendada para gerenciar aplicações K8s em produção

Kubectl e as maneiras de interação

Abordagem Imperativa:

- ▶ Kubectl Get, Describe, Delete podem ser usados com quaisquer recursos:

- ▶ Pods

```
kubectl get pods
```

- ▶ Nodes

```
kubectl get nodes
```

- ▶ Deployment

```
kubectl get services
```

- ▶ ReplicaSet

```
kubectl describe pod <Nome do Pod>
```

- ▶ Service

```
kubectl describe service <Nome do Service>
```

```
kubectl delete pod <Nome do Pod>
```

```
kubectl delete deployment <Nome do Deployment>
```

Kubectl e as maneiras de interação

Abordagem Imperativa:

- ▶ Kubectl Create
- ▶ Kubectl Run
- ▶ Kubectl Scale
- ▶ Kubectl Expose
- ▶ Kubectl Exec
- ▶ Kubectl Copy
- ▶ Kubectl Logs



Kubectl e as maneiras de interação

Abordagem Declarativa:

- ▶ `kubectl apply -f <arquivo.yaml>`
- ▶ `kubectl apply -f <arquivo1.yaml> -f <arquivo2.yaml>`
- ▶ `kubectl apply -f <folder>/`

Em caso de mudanças no arquivo, *kubectl apply* atualiza os recursos

Criando um Pod

```
kubectl run nginx --generator=run-pod/v1 --image=nginx
```

```
cat << EOF | kubectl create -f -  
apiVersion: v1  
kind: Pod  
metadata:  
  name: nginx-pod  
spec:  
  containers:  
  - name: nginx-container  
    image: nginx  
EOF
```

Exportando manifesto

1. Salva manifesto de um Pod

```
kubect! get pod my-pod -o yaml > my-pod.yaml
```

2. Salva manifesto sem informações específicas do cluster

```
kubect! get pod my-pod -o yaml --export > my-pod.yaml
```


Namespace

K8s usa namespaces para organizar objetos no cluster através de uma divisão lógica (como se fosse uma pasta).

Por padrão, kubectl interage com o namespace padrão (default).

Para usar um namespace específico (diferente do padrão), pode-se usar a flag `--namespace=<nome>`, ou ainda `-n <nome>`.

Para interagir com todos os namespaces, pode-se passar a flag `--all-namespaces` para o comando.

Namespace

1. Criar namespace

```
kubectl create namespace dev  
kubectl create namespace prod
```

2. Listar namespaces

```
kubectl get namespaces
```

3. Remover namespace

```
kubectl delete namespace dev
```



Namespace

4. Filtrar Pods por namespace (opção 1)

```
kubectl get pods --namespace=teste
```

5. Filtrar Pods por namespace (opção 2)

```
kubectl get pods -n teste
```

6. Listar Pods de todos os namespaces

```
kubectl get pods --all-namespaces
```



Labels

Um Label é um par chave-valor do tipo string.
Todos os recursos/objetos K8s podem ser rotulados.

1. Equality-based requirement

environment = production

tier != frontend

2. Set-based requirement

environment in (production, qa)

tier notin (frontend, backend)

Labels

Um Label é um par chave-valor do tipo string.
Todos os recursos/objetos K8s podem ser rotulados.

1. Mostrar labels dos recursos:

```
kubectl get pods --show-labels
```

2. Deletar Pods que têm label run=myapp

```
kubectl delete pods -l environment=production,tier=frontend
```

```
kubectl get pods -l 'environment in (production),tier in (frontend)'
```

3. Atribuir label

```
kubectl label deployment nginx-deployment tier=dev
```

5. Instalação dos componentes do K8s

No sistema operacional Ubuntu

K8s singlenode

Microk8s

<https://microk8s.io/>

Minikube

<https://github.com/kubernetes/minikube>



Instalação de cluster K8s no Ubuntu

- ▶ Master Node
- ▶ Worker Node 1
- ▶ Worker Node 2



Instalação de cluster K8s no Ubuntu

Instalar Docker em todos os Nodes

Adicionar chave GPG

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Adicionar repositório do Docker

```
sudo add-apt-repository \
    "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

Atualizar lista de pacotes

```
sudo apt-get update
```

Listar as versões do Docker que estão disponíveis no repositório

```
sudo apt list docker-ce -a
```

Instalação de cluster K8s no Ubuntu

Instalar versão específica do Docker

```
sudo apt-get install -y docker-ce=18.06.1~ce~3-0~ubuntu
```

Impedir que o Docker atualize para a versão mais recente

```
sudo apt-mark hold docker-ce
```

Configurar o daemon

```
cat > /etc/docker/daemon.json <<EOF
```

```
{ "exec-opts": ["native.cgroupdriver=systemd"],
```

```
  "log-driver": "json-file",
```

```
  "log-opts": {
```

```
    "max-size": "100m"},
```

```
  "storage-driver": "overlay2"}  
EOF
```

Instalação de cluster K8s no Ubuntu

Configurar o daemon

```
mkdir -p /etc/systemd/system/docker.service.d
```

Reiniciar o docker.

```
systemctl daemon-reload
```

```
systemctl restart docker
```

Verificar a versão instalada

```
sudo docker version
```



Instalação de cluster K8s no Ubuntu

Instalar K8s em todos os Nodes

Adicionar chave GPG

```
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
```

Adicionar repositório do Kubernetes

```
cat << EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list
```

```
deb https://apt.kubernetes.io/ kubernetes-xenial main
```

```
EOF
```

Atualizar lista de pacotes

```
sudo apt-get update
```

Listar as versões do K8s que estão disponíveis no repositório

```
sudo apt list kubelet -a
```

Instalação de cluster K8s no Ubuntu

Instalar versões específicas das ferramentas

```
sudo apt-get install -y kubelet=1.12.7-00 kubeadm=1.12.7-00 kubectl=1.12.7-00
```

Impedir que atualizações das ferramentas sejam instaladas

```
sudo apt-mark hold kubelet kubeadm kubectl
```

Verificar a versão instalada

```
sudo kubeadm version
```

Desabilitar o Swap nos nós do Kubernetes

Desabilitar swap imediatamente

```
sudo swapoff -a
```

Atualizar fstab para que o swap permaneça desabilitado após o reboot

```
sudo sed -i 's/^(\s*)swap\s/(\s*)$/#1/g' /etc/fstab
```

c

6. Configuração do K8s

Criando o cluster

Minikube no Kataconda

Kataconda é um ambiente interativo para auxiliar em estudos e treinamentos, onde criam-se ambientes virtuais acessíveis através do browser.

Precisa ter conta Docker ou Github.

<https://www.katacoda.com/courses/kubernetes/launch-single-node-cluster>

Cluster no Kataconda

Kataconda é um ambiente interativo para auxiliar em estudos e treinamentos, onde criam-se ambientes virtuais acessíveis através do browser.

Precisa ter conta Docker ou Github.

<https://www.katacoda.com/courses/kubernetes/getting-started-with-kubeadm>

Configurando o Cluster K8s

1. Iniciar o Master Node (ao fim, salvar o comando *kubeadm join*)

```
kubeadm init --apiserver-advertise-address $(hostname -i)
```

2. Setar arquivo de configuração para poder interagir com o cluster

```
sudo cp /etc/kubernetes/admin.conf $HOME/  
sudo chown $(id -u):$(id -g) $HOME/admin.conf  
export KUBECONFIG=$HOME/admin.conf
```

3. Configurar a rede do K8s com o Plugin Weave

```
kubectly apply -n kube-system -f \  
"https://cloud.weave.works/k8s/net?k8s-version=$(kubectly version | base64 |tr -d '\n')"
```

Configurando o Cluster K8s

4. Listar os Nodes

```
kubectl get nodes
```

5. Listar os Pods do namespace kube-system

```
kubectl get pod -n kube-system
```

6. Adicionar os Worker Nodes ao cluster (usar o comando salvo após configurar o Master Node)

```
kubeadm join 172.17.0.8:6443 --token <TOKEN> \  
--discovery-token-ca-cert-hash <SHA256>
```

Configurando o Cluster K8s

7. Listar os Nodes continuamente

```
kubectl get nodes -w
```

8. Mostrar detalhes do Node com nome master

```
kubectl describe node master
```

9. Mostrar detalhes do Node com nome node01

```
kubectl describe node node01
```



Configurando o Cluster K8s

10. Listar o status dos componentes

```
kubectl get componentstatuses
```

11. Listar os Pods de todos os namespaces

```
kubectl get pod --all-namespaces
```

12. Mostrar algumas informações do cluster

```
kubectl cluster-info
```

13. Permitir que o node master execute Pods

```
kubectl taint node <MASTER> node-role.kubernetes.io/master:NoSchedule-
```

Outros comandos úteis

```
kubectl logs deployment/myapp
```

```
kubectl -n kube-system logs -f POD_NAME
```

```
kubectl exec POD_NAME -it sh
```

```
kubectl cp POD_NAME:<FILE> <LOCAL FILE>
```

```
kubectl top pod POD_NAME --containers
```

```
kubectl -n my-ns delete pod,svc --all
```

```
kubectl edit svc/docker-registry
```



7. ReplicaSet

Gerenciando um conjunto de Pods

ReplicaSet

É uma maneira de manter estável um conjunto de Pods em execução, garantindo a **disponibilidade** de um número específico de Pods idênticos.

```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: kuard
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: kuard
        version: "2"
    spec:
      containers:
        - name: kuard
          image: "gcr.io/kuar-demo/kuard-amd64:green"
```

ReplicaSet

1. Criar ReplicaSet

```
kubectl create -f busybox-rs.yaml
```

2. Listar ReplicaSets e Pods

```
kubectl get rs
```

```
kubectl get pods -o wide
```

3. Mostrar detalhes do ReplicaSet

```
kubectl describe rs/busybox
```

busybox-rs.yaml:

```
apiVersion: apps/v1
```

```
kind: ReplicaSet
```

```
metadata:
```

```
  name: busybox
```

```
  labels:
```

```
    app: busybox
```

```
spec:
```

```
  replicas: 2
```

```
  selector:
```

```
    matchLabels:
```

```
      tier: busybox
```

```
  template:
```

```
    metadata:
```

```
      labels:
```

```
        tier: busybox
```

```
    spec:
```

```
      containers:
```

```
        - name: busybox
```

```
          image: radial/busyboxplus:curl
```


ReplicaSet

4. Escalar um ReplicaSet

```
kubectl scale --replicas 3 rs busybox
```

```
kubectl scale --replicas 1 rs busybox
```

5. Deletar um ReplicaSet

```
kubectl delete rs busybox
```

6. Deletar todos os Pods e ReplicaSets

```
kubectl delete pod,rs --all
```

busybox-rs.yaml:

```
apiVersion: apps/v1
```

```
kind: ReplicaSet
```

```
metadata:
```

```
  name: busybox
```

```
  labels:
```

```
    app: busybox
```

```
spec:
```

```
  replicas: 2
```

```
  selector:
```

```
    matchLabels:
```

```
      tier: busybox
```

```
  template:
```

```
    metadata:
```

```
      labels:
```

```
        tier: busybox
```

```
    spec:
```

```
      containers:
```

```
        - name: busybox
```

```
          image: radial/busyboxplus:curl
```

ReplicaSet

ReplicaSet é raramente utilizado diretamente, pois existe um conceito de mais alto nível, Deployment, que gerencia ReplicaSets e provê uma maneira de atualizar os Pods, além de outras features.

DO NOT USE

c

8. Deployment

Como automatizar a inicialização de múltiplos Pods?

Deployment

É uma maneira de automatizar o gerenciamento de Pods.

Ele permite especificar um estado desejado para um conjunto de Pods e o cluster vai constantemente trabalhar para manter o estado desejado.

Todo Deployment cria um ReplicaSet.

Deployment

Vantagens:

- ▶ Escalabilidade: com um Deployment, pode-se especificar o número de réplicas desejado e o Deployment vai criar ou remover Pods até alcançar o número desejado.
- ▶ Atualizações: é possível alterar a imagem de um container para uma nova versão e o Deployment vai gradualmente substituir os containers para a nova versão (evita downtime).
- ▶ Self-healing: se um dos Pods for acidentalmente destruído, o Deployment vai imediatamente iniciar um novo Pod para substituí-lo.

Criando um Deployment

```
cat <<EOF | kubectl create -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.15.4
          ports:
            - containerPort: 80
EOF
```

Deployment: Demo

1. Criar Deployment

```
kubectl create deployment http-deployment --image=nginx
```

2. Listar Pods e Deployments

```
kubectl get pods
```

```
kubectl get deployments
```

3. Aumentar a quantidade de réplicas

```
kubectl scale --replicas 3 deployment http-deployment
```

Deployment: Demo

4. Diminuir a quantidade de réplicas

```
kubectl scale --replicas 2 deployment http-deployment
```

5. Listar os deployments

```
kubectl get deployments
```

6. Mostrar detalhes de um deployment

```
kubectl describe deployment http-deployment
```



Deployment - Nova versão da app

1. Criar Deployment

```
kubectl create deployment nginx-deployment --image=nginx:1.7.9
```

2. Listar Pods, Deployments e ReplicaSets

```
kubectl get pods
```

```
kubectl get deployments
```

```
kubectl get rs
```

3. Escalar o Deployment

```
kubectl scale --replicas 3 deployment nginx-deployment
```

Deployment - Nova versão da app

4. Atualizar o Deployment com uma nova versão da aplicação

```
kubectl set image deployment/nginx-deployment nginx=nginx:1.91 --record
```

5. Mostrar detalhes do Deployment

```
kubectl describe deployment nginx-deployment
```

6. Verificar status da atualização

```
kubectl rollout status deployment.v1.apps/nginx-deployment
```

7. Listar Deployments

```
kubectl get deployments
```

Deployment - Nova versão da app

8. Em caso de erro, desfazer a atualização

```
kubectl rollout undo deployment.v1.apps/nginx-deployment
```

9. Mostrar detalhes do Deployment

```
kubectl describe deployment nginx-deployment
```

10. Atualizar o Deployment com a versão correta da aplicação

```
kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1 --record
```

11. Verificar status da atualização

```
kubectl rollout status deployment.v1.apps/nginx-deployment
```

9. Service

O que fazer quando entidades externas quiserem acessar os Pods?

Service

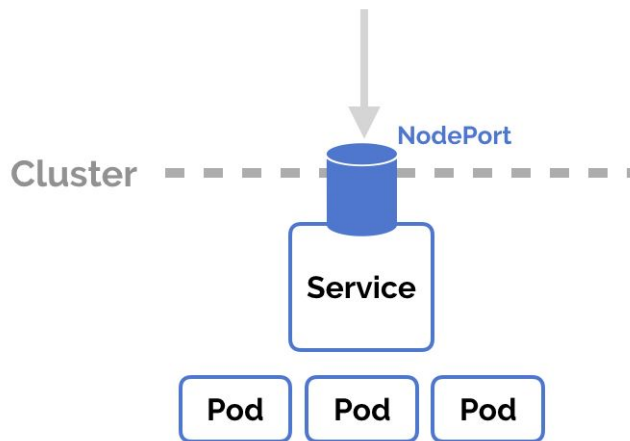
Motivação:

- ▶ Pods podem ser criados e destruídos constantemente por um Deployment
- ▶ Uma vez que cada Pod recebe um IP diferente, os Pods em execução em um dado momento podem ser diferentes dos Pods em um momento posterior.
- ▶ Como manter o acesso aos serviços providos pelos Pods?
 - ▶ Ex: Como “frontend” Pods mantém a informação sobre os “backend” Pods se os IPs dos Pods mudam repentinamente?

Service

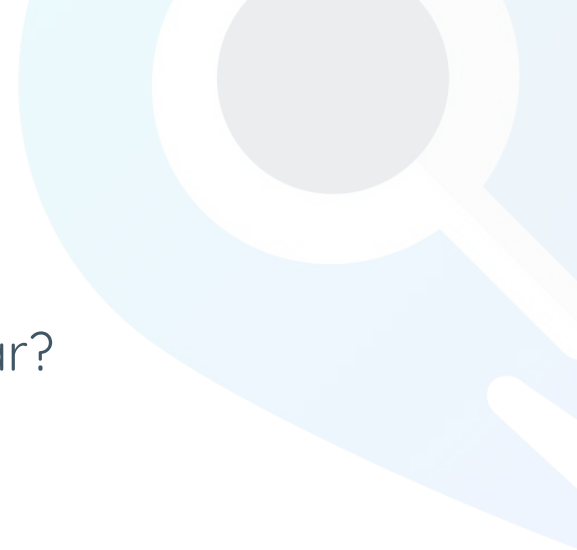
O Service cria uma camada de abstração acima do conjunto de Pods réplicas, permitindo acesso dinâmico a um grupo de Pods.

Assim, pode-se prover acesso ininterrupto e dinâmico a qualquer réplica, além de balanceamento de carga.



Service

Como o Service sabe em quais Pods deve atuar?



Service

Como o Service sabe em quais Pods deve atuar?

- ▶ **Selector:** define o label dos Pods afetados pelo Service

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```


Service

Alguns tipos de Services:

- ▶ **ClusterIP:** expõe o Service apenas para a rede interna do cluster. Esse é o ServiceType padrão.
- ▶ **NodePort:** expõe o Service em cada Node com uma porta estática (NodePort), permitindo acesso externo através do endereço <IP do Node>:<NodePort>.
- ▶ **ExternalName:** Mapeia o Service para um nome/endereço (e.g. foo.bar.example.com).
- ▶ **LoadBalancer:** expõe o Service externamente através de um balanceador de carga de um provedor de nuvem. Internamente, cria NodePort, pra onde o balanceador de carga externo roteia.

Service: Demo do NodePort

1. Criar 2 Pods com mesmo label

```
cat << EOF | kubectl create -f -  
apiVersion: v1  
kind: Pod  
metadata:  
  name: nginx-pod-1  
  labels:  
    app: nginx  
spec:  
  containers:  
    - name: nginx-container  
      image: nginx  
EOF
```

```
cat << EOF | kubectl create -f -  
apiVersion: v1  
kind: Pod  
metadata:  
  name: nginx-pod-2  
  labels:  
    app: nginx  
spec:  
  containers:  
    - name: nginx-container  
      image: nginx  
EOF
```

Service: Demo do NodePort

2. Listar Pods com mesmo label

```
kubectl get pods -l app=nginx -o wide
```

```
kubectl get pods --selector=app=http-deployment
```

3. Acessar os pods

```
curl <IP do Pod>
```

4. Criar Service

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

nginx-service.yaml:

```
kind: Service
apiVersion: v1
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
      nodePort: 30080
      type: NodePort
```

Service: Demo do NodePort

5. Listar os serviços

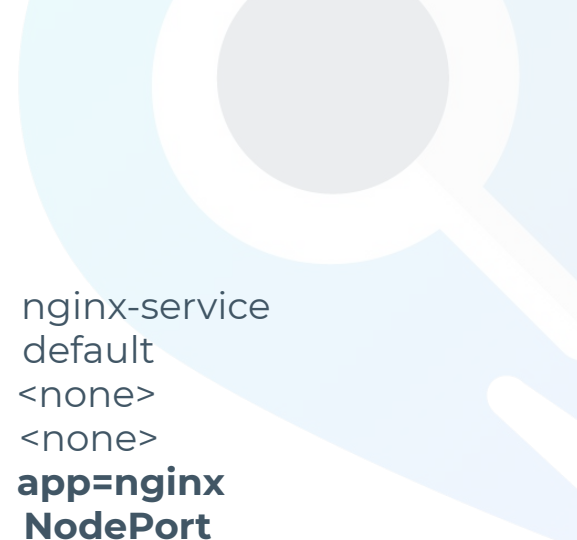
```
kubectl get services
```

6. Mostrar detalhes do service

```
kubectl describe service nginx-service
```

7. Acessar o Service

```
curl <IP do Node>:30080
```

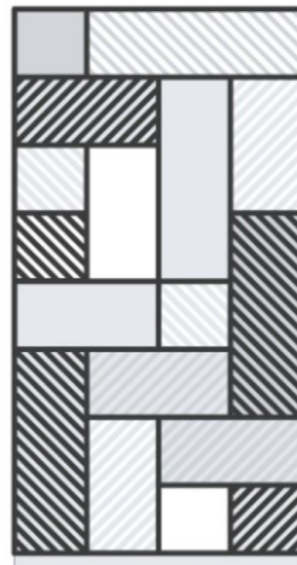
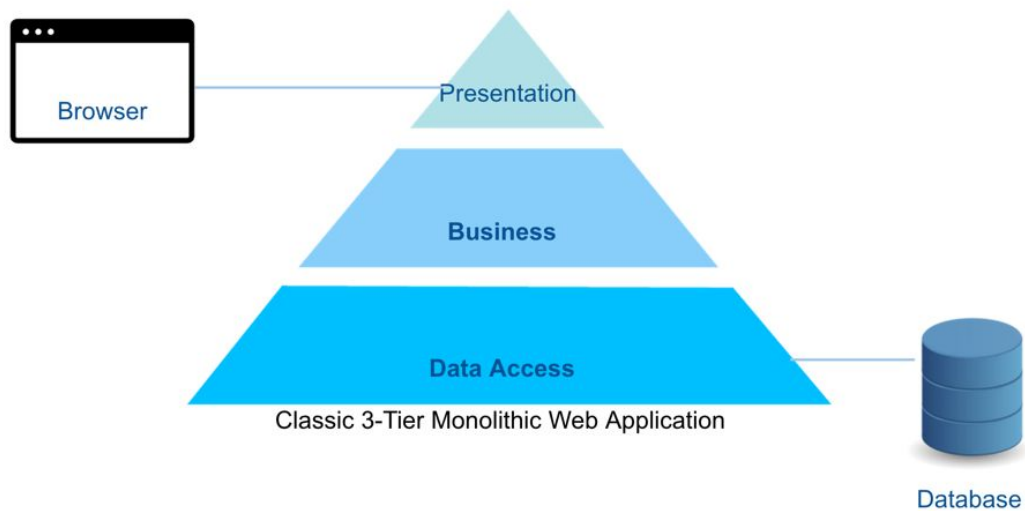


Name:	nginx-service
Namespace:	default
Labels:	<none>
Annotations:	<none>
Selector:	app=nginx
Type:	NodePort
IP:	<IP Service>
Port:	<unset> 80/TCP
TargetPort:	80/TCP
NodePort:	<unset> 30080/TCP
Endpoints:	<IP Pod 1>:80,<IP Pod 2>:80
Session Affi.:	None
Ext. Traffic P.:	Cluster
Events:	<none>

10. Microservices

Cenário ideal para utilizar o K8s

Aplicações Monolíticas

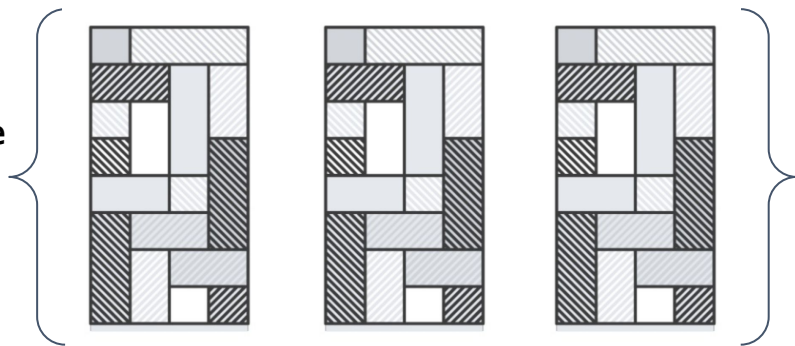


Aplicações Monolíticas

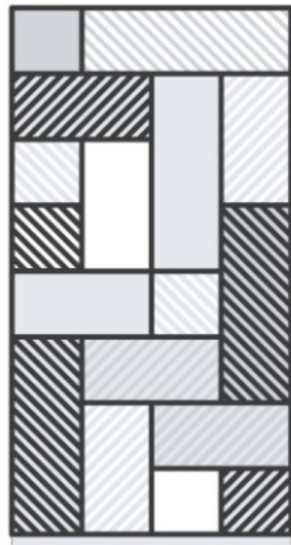
Desafios

- ▶ Dificuldade de escalar
- ▶ Arquitetura difícil de manter e evoluir

Escalabilidade
Horizontal



Escalabilidade
Vertical



88 Arquitetura de Microsserviços

Abordagem que desenvolve um aplicativo único como uma suíte de pequenos serviços, cada um executando seu próprio processo e se comunicando através de mecanismos leves.

Os serviços funcionam através de mecanismos de deploy independentes totalmente automatizados.

Há o mínimo possível de gerenciamento centralizado dos serviços, que podem ser escritos em diferentes linguagens de programação e utilizam diferentes tecnologias de armazenamento de dados.

Arquitetura de Microserviços

Decompõe a aplicação por funções básicas, onde cada função é denominada um serviço e pode ser criada e implantada de maneira independente.

Cada serviço individual pode funcionar ou falhar sem comprometer os demais.

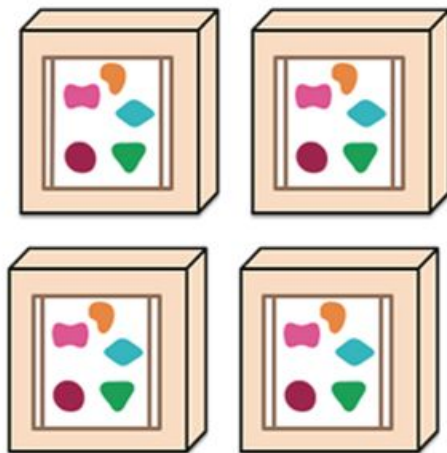
O uso de microserviços é uma das melhores maneiras de demonstrar o quão valioso é gerenciar containers com o K8s.

Monolítica x Microserviços

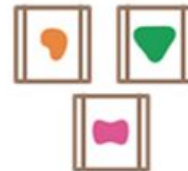
Um aplicativo monolítico tem todas as suas funcionalidades em um único processo...



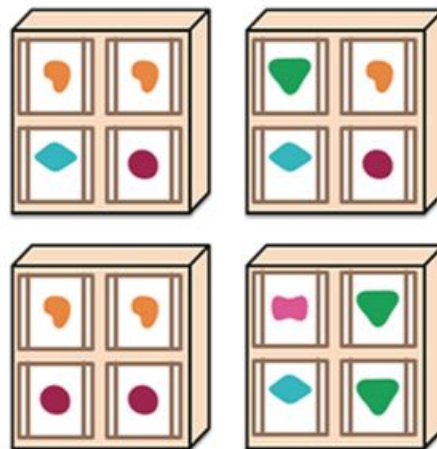
...e escala replicando o monolito em servidores múltiplos



A arquitetura de micro-serviços coloca cada elemento de funcionalidade em um serviço separado...



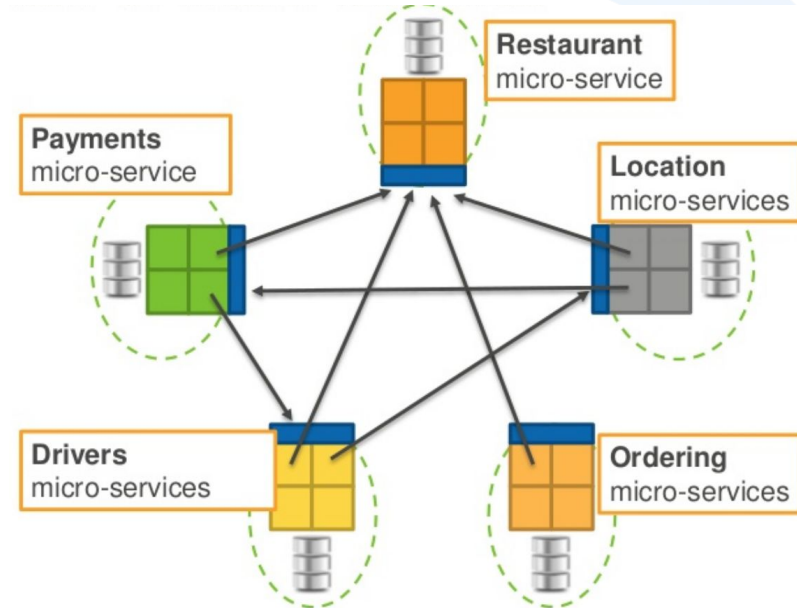
...e escala distribuindo os serviços entre os servidores, replicando por demanda.



Ecossistema de microsserviços

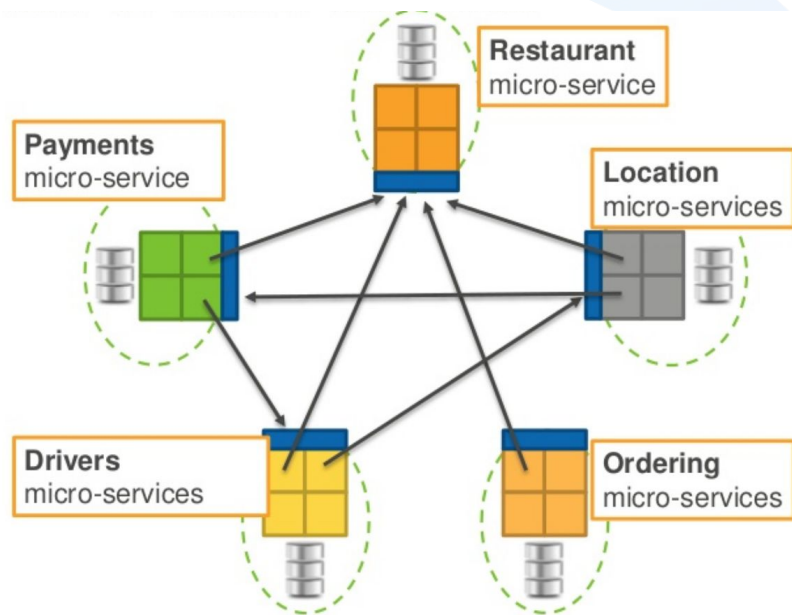
Cada um desses microsserviços pode ser escalado diferentemente e independentemente.

Assim, se necessário, é possível alocar mais recursos para um determinado microsserviço.



Algumas das vantagens

- ▶ Escalabilidade
- ▶ Mais fácil de atualizar partes da aplicação
- ▶ Confiabilidade e disponibilidade: uma parte da app pode parar, mas outras não
- ▶ Usar ferramentas, linguagens e frameworks ideais para o trabalho.



Deploy de aplicação Robot Shop App: Demo

1. Clonar o repositório do Git:

```
cd ~; git clone https://github.com/linuxacademy/robot-shop.git
```

2. Criar um namespace e fazer deploy da aplicação

```
kubectl create namespace robot-shop
```

```
kubectl -n robot-shop create -f ~/robot-shop/K8s/descriptors/
```

3. Acompanhar a mudança de status dos Pods

```
kubectl get pods -n robot-shop -w
```

4. Acessar o frontend da aplicação

```
http://<IP do Node Master>:30080
```

OBRIGADO!

Dúvidas?

- ▶ Prof. Paulo A L Rego
 - ▶ paulo@dc.ufc.br

