

```
>>> >>> Kubernetes
```

```
Name:  Henrique Tsuyoshi Yara (OPUS-software)
```

```
Date:  January 11, 2023
```



kubernetes

Figure: Kubernetes logo

>>> Índice

- | | |
|---------------------|--------------------------|
| 1. K8s | 5. Recursos |
| 2. K8s vs Docker | 6. Troubleshooting |
| 3. Pod vs Container | 7. Ambientes Productivos |
| 4. Cluster local | 8. Helm |
| | 9. Referencias |

>>> **Conceito**

★ O kubernetes provê:

>>> **Conceito**

- * O kubernetes provê:

- * *Service Discovery* e Balanceamento de carga

>>> Conceito

- * O kubernetes provê:
 - * *Service Discovery* e Balanceamento de carga
 - * Orquestração de armazenamento

>>> Conceito

- * O kubernetes provê:
 - * *Service Discovery* e Balanceamento de carga
 - * Orquestração de armazenamento
 - * *Rollouts* e *Rollbacks* automáticos

>>> Conceito

★ O kubernetes provê:

- ★ *Service Discovery* e Balanceamento de carga
- ★ Orquestração de armazenamento
- ★ *Rollouts* e *Rollbacks* automáticos
- ★ Controle de recursos automáticos (CPU, RAM)

>>> Conceito

* O kubernetes provê:

- * *Service Discovery* e Balanceamento de carga
- * Orquestração de armazenamento
- * *Rollouts* e *Rollbacks* automáticos
- * Controle de recursos automáticos (CPU, RAM)
- * Auto recuperação

>>> Conceito

★ O kubernetes provê:

- ★ *Service Discovery* e Balanceamento de carga
- ★ Orquestração de armazenamento
- ★ *Rollouts* e *Rollbacks* automáticos
- ★ Controle de recursos automáticos (CPU, RAM)
- ★ Auto recuperação
- ★ Gerenciamento de *Secrets* e configurações

>>> Pré-requisitos

Container runtime The container runtime is the software that is responsible for running containers. Kubernetes supports container runtimes such as containerd, CRI-O, and any other implementation of the Kubernetes CRI (Container Runtime Interface).

>>> Componentes

- * *Pod*: Consiste em um container ou um conjunto de containers

>>> Componentes

- * *Pod*: Consiste em um container ou um conjunto de containers
- * *Node*: É onde fica agrupado os *Pods* que compõe a aplicação

>>> Componentes

- * *Pod*: Consiste em um container ou um conjunto de containers
- * *Node*: É onde fica agrupado os *Pods* que compõe a aplicação
- * *Cluster*: Um conjunto de máquinas trabalhadoras (*Nodes*)

>>> Componentes

- * *Pod*: Consiste em um container ou um conjunto de containers
- * *Node*: É onde fica agrupado os *Pods* que compõe a aplicação
- * *Cluster*: Um conjunto de máquinas trabalhadoras (*Nodes*)
- * *Control Plane*: Faz decisões e gerencia o *cluster* (detectar eventos, programar *Pods*)

>>> Componentes

- * *Pod*: Consiste em um container ou um conjunto de containers
- * *Node*: É onde fica agrupado os *Pods* que compõe a aplicação
- * *Cluster*: Um conjunto de máquinas trabalhadoras (*Nodes*)
- * *Control Plane*: Faz decisões e gerencia o *cluster* (detectar eventos, programar *Pods*)
 - * Pode rodar em múltiplos nós provendo alta disponibilidade e prevenindo falhas

>>> Componentes

- * *Pod*: Consiste em um container ou um conjunto de containers
- * *Node*: É onde fica agrupado os *Pods* que compõe a aplicação
- * *Cluster*: Um conjunto de máquinas trabalhadoras (*Nodes*)
- * *Control Plane*: Faz decisões e gerencia o *cluster* (detectar eventos, programar *pods*)
 - * Pode rodar em múltiplos nós provendo alta disponibilidade e prevenindo falhas
- * *Worker*: Um conjunto de máquinas trabalhadoras (*Nodes*)

>>> Componentes

- * *Pod*: Consiste em um container ou um conjunto de containers
- * *Node*: É onde fica agrupado os *Pods* que compõe a aplicação
- * *Cluster*: Um conjunto de máquinas trabalhadoras (*Nodes*)
- * *Control Plane*: Faz decisões e gerencia o *cluster* (detectar eventos, programar *Pods*)
 - * Pode rodar em múltiplos nós provendo alta disponibilidade e prevenindo falhas
- * *Worker*: Um conjunto de máquinas trabalhadoras (*Nodes*)
 - * Roda em múltiplos nós provendo alta disponibilidade e prevenindo falhas

>>> Componentes - Control Plane

- ★ **kube-apiserver:** *front end* do *control plane*. A implementação foi feita de uma forma que é possível ser escalada horizontalmente em várias instancias e consegue balancear o trafico de rede entre essas instâncias.

>>> Componentes - Control Plane

- ★ **kube-apiserver:** *front end* do *control plane*. A implementação foi feita de uma forma que é possível ser escalada horizontalmente em várias instancias e consegue balancear o trafico de rede entre essas instâncias.
- ★ **etcd:** Possui consistência e alta disponibilidade. Armazena os dados do cluster em valores-chaves. É bom ter um *backup* para esses dados

>>> Componentes - Control Plane

- ★ **kube-apiserver:** *front end* do *control plane*. A implementação foi feita de uma forma que é possível ser escalada horizontalmente em várias instancias e consegue balancear o trafico de rede entre essas instâncias.
- ★ **etcd:** Possui consistência e alta disponibilidade. Armazena os dados do cluster em valores-chaves. É bom ter um *backup* para esses dados
- ★ **kube-scheduler:** Procura por *Pods* que não foram atribuídos à nós e atribui nós para eles (Leva em conta políticas, *hardware*, *software*, etc...)

>>> Componentes - Control Plane

- ★ **kube-controller-manager:** Roda processos *controllers*

>>> Componentes - Control Plane

- ★ **kube-controller-manager:** Roda processos *controllers*
 - ★ **Node controller:** Verificar quando um nó falha

>>> Componentes - Control Plane

- ★ **kube-controller-manager:** Roda processos *controllers*
 - ★ **Node controller:** Verificar quando um nó falha
 - ★ **Job controller:** Procuram por *Job* que acontecem apenas uma vez, criam *Pods* para executar a tarefa

>>> Componentes - Control Plane

- ★ **kube-controller-manager:** Roda processos *controllers*
 - ★ **Node controller:** Verificar quando um nó falha
 - ★ **Job controller:** Procuram por *Job* que acontecem apenas uma vez, criam *Pods* para executar a tarefa
 - ★ **EndpointSlice controller:** Popula objetos *EndpointSlice* (provê links entre Serviços e *Pods*)

>>> Componentes - Control Plane

- ★ **kube-controller-manager:** Roda processos *controllers*
 - ★ **Node controller:** Verificar quando um nó falha
 - ★ **Job controller:** Procuram por *Job* que acontecem apenas uma vez, criam *Pods* para executar a tarefa
 - ★ **EndpointSlice controller:** Popula objetos *EndpointSlice* (provê links entre Serviços e *Pods*)
 - ★ **ServiceAccount controller:** Criar *ServiceAccounts* para novos *namespaces*

>>> Componentes - Control Plane

- ★ **kube-controller-manager:** Roda processos *controllers*
 - ★ **Node controller:** Verificar quando um nó falha
 - ★ **Job controller:** Procuram por *Job* que acontecem apenas uma vez, criam *Pods* para executar a tarefa
 - ★ **EndpointSlice controller:** Popula objetos *EndpointSlice* (provê links entre Serviços e *Pods*)
 - ★ **ServiceAccount controller:** Criar *ServiceAccounts* para novos *namespaces*
- ★ **cloud-controller-manager:** permite conectar com a *API* do provedor de nuvem. Dessa forma o usuário vai interagir apenas com o *cluster*

>>> Componentes - Control Plane

- ★ **kube-controller-manager:** Roda processos *controllers*
 - ★ **Node controller:** Verificar quando um nó falha
 - ★ **Job controller:** Procuram por *Job* que acontecem apenas uma vez, criam *Pods* para executar a tarefa
 - ★ **EndpointSlice controller:** Popula objetos *EndpointSlice* (provê links entre Serviços e *Pods*)
 - ★ **ServiceAccount controller:** Criar *ServiceAccounts* para novos *namespaces*
- ★ **cloud-controller-manager:** permite conectar com a *API* do provedor de nuvem. Dessa forma o usuário vai interagir apenas com o *cluster*
 - ★ Pode ser escalado horizontalmente para melhorar a performance e ajudar a evitar falhas

>>> Componentes - Todos os Nodes

Estão em todos os nós, fazem manutenção nos *Pods* rodando.

- ★ **kubelet**: Garantem que os containers estão rodando em um *Pod*

>>> Componentes - Todos os Nodes

Estão em todos os nós, fazem manutenção nos *Pods* rodando.

- ★ **kubelet**: Garantem que os containers estão rodando em um *Pod*
 - ★ O **kubelet** não gerencia containers que não foram criados pelo **kubernetes**

>>> Componentes - Todos os Nodes

Estão em todos os nós, fazem manutenção nos *Pods* rodando.

- * **kubelet**: Garantem que os containers estão rodando em um *Pod*
 - * O **kubelet** não gerencia containers que não foram criados pelo **kubernetes**
- * **kube-proxy**: Um proxy que roda em todos os nós do *cluster*

>>> Componentes - Todos os Nodes

Estão em todos os nós, fazem manutenção nos *Pods* rodando.

- ★ **kubelet**: Garantem que os containers estão rodando em um *Pod*
 - ★ O **kubelet** não gerencia containers que não foram criados pelo **kubernetes**
- ★ **kube-proxy**: Um proxy que roda em todos os nós do *cluster*
 - ★ Fazem com que as regras de redes funcionem nos nós (Permitem comunicação de rede dos *Pods* dentro e fora do cluster)

>>> K8s vs Docker

- * **Docker** é uma plataforma de containerização
- * **K8s** é um orquestrador de containers

>>> **Pods vs containers**

"Pods are the smallest deployable units of computing that you can create and manage in Kubernetes." [2]

>>> Pods vs containers

"Pods are the smallest deployable units of computing that you can create and manage in Kubernetes." [2]

★ 1 Pod : 1 Container

>>> Pods vs containers

"Pods are the smallest deployable units of computing that you can create and manage in Kubernetes." [2]

- ★ 1 Pod : 1 Container

- ★ Grupo de 1 ou mais containers

>>> Pods vs containers

"Pods are the smallest deployable units of computing that you can create and manage in Kubernetes." [2]

- ★ 1 Pod : 1 Container
- ★ Grupo de 1 ou mais containers
 - ★ Compartilham armazenamento e rede

>>> Pods vs containers

"Pods are the smallest deployable units of computing that you can create and manage in Kubernetes." [2]

- ★ 1 Pod : 1 Container
- ★ Grupo de 1 ou mais containers
 - ★ Compartilham armazenamento e rede
 - ★ Especificações para rodar o container

>>> Pods vs containers

"Pods are the smallest deployable units of computing that you can create and manage in Kubernetes." [2]

- ★ 1 Pod : 1 Container
- ★ Grupo de 1 ou mais containers
 - ★ Compartilham armazenamento e rede
 - ★ Especificações para rodar o container
 - ★ Pode conter initContainers

>>> Pods vs containers

"Pods are the smallest deployable units of computing that you can create and manage in Kubernetes." [2]

- ★ 1 Pod : 1 Container
- ★ Grupo de 1 ou mais containers
 - ★ Compartilham armazenamento e rede
 - ★ Especificações para rodar o container
 - ★ Pode conter initContainers
 - ★ Pode conter containers efêmeros

>>> Containers efêmeros

Ephemeral containers:

- * Geralmente é usado para inspecionar (*Debugging*) o estado do Pod:

>>> Containers efêmeros

Ephemeral containers:

- * Geralmente é usado para inspecionar (*Debugging*) o estado do Pod:
- * Não possui portas, portanto **livenessProbe**, **readinessProbe** não são permitidos

>>> Containers efêmeros

Ephemeral containers:

- * Geralmente é usado para inspecionar (*Debugging*) o estado do Pod:
- * Não possui portas, portanto **livenessProbe**, **readinessProbe** não são permitidos
- * Recursos do pod são imutáveis, portanto não é permitido configurar recursos

>>> Containers efêmeros

Ephemeral containers:

- * Geralmente é usado para inspecionar (*Debugging*) o estado do Pod:
- * Não possui portas, portanto **livenessProbe**, **readinessProbe** não são permitidos
- * Recursos do pod são imutáveis, portanto não é permitido configurar recursos
- * Lista completa de permissões: [link](#)

>>> Pods estáticos

- ★ Gerenciados diretamente pelo *kubelet* do nó (independente do *API server*)

¹<https://anote.dev/static-pods-in-kubernetes/>

>>> Pods estáticos

- ★ Gerenciados diretamente pelo *kubelet* do nó (independente do *API server*)
- ★ O *kubelet* verifica o *Pod* e reinicia caso falhe

¹<https://anote.dev/static-pods-in-kubernetes/>

>>> Pods estáticos

- * Gerenciados diretamente pelo *kubelet* do nó (independente do *API server*)
- * O *kubelet* verifica o *Pod* e reinicia caso falhe
- * *kubelet* verifica o diretório */etc/kubernetes/manifests* por padrão e cria pods caso seja necessário

¹<https://anote.dev/static-pods-in-kubernetes/>

>>> Pods estáticos

- ★ Gerenciados diretamente pelo *kubelet* do nó (independente do *API server*)
- ★ O *kubelet* verifica o *Pod* e reinicia caso falhe
- ★ *kubelet* verifica o diretório */etc/kubernetes/manifests* por padrão e cria pods caso seja necessário
- ★ Não suporta *Containers efêmeros* e não pode referenciar outros objetos da *API* (*ConfigMap*, *Secret*, *etc...*)

¹<https://anote.dev/static-pods-in-kubernetes/>

>>> Pods estáticos

- ★ Gerenciados diretamente pelo *kubelet* do nó (independente do *API server*)
- ★ O *kubelet* verifica o *Pod* e reinicia caso falhe
- ★ *kubelet* verifica o diretório */etc/kubernetes/manifests* por padrão e cria pods caso seja necessário
- ★ Não suporta *Containers efêmeros* e não pode referenciar outros objetos da *API* (*ConfigMap*, *Secret*, *etc...*)
- ★ Onde usar?

¹<https://anote.dev/static-pods-in-kubernetes/>

>>> Pods estáticos

- ★ Gerenciados diretamente pelo *kubelet* do nó (independente do *API server*)
- ★ O *kubelet* verifica o *Pod* e reinicia caso falhe
- ★ *kubelet* verifica o diretório */etc/kubernetes/manifests* por padrão e cria pods caso seja necessário
- ★ Não suporta *Containers efêmeros* e não pode referenciar outros objetos da *API* (*ConfigMap*, *Secret*, *etc...*)
- ★ Onde usar?
 - ★ Nos componentes do control plane¹

¹<https://anote.dev/static-pods-in-kubernetes/>

>>> *LivenessProbe, ReadinessProbe e StartUpProbe*

Os 3 são controlados pelo *kubelet*

>>> *LivenessProbe, ReadinessProbe e StartUpProbe*

Os 3 são controlados pelo *kubelet*

LivenessProbe:

- ★ Tentar fazer a aplicação ficar mais disponível removendo *bugs*

>>> *LivenessProbe, ReadinessProbe e StartUpProbe*

Os 3 são controlados pelo *kubelet*

LivenessProbe:

- ★ Tentar fazer a aplicação ficar mais disponível removendo *bugs*
 - ★ Pode perceber um *deadlock* na aplicação

ReadinessProbe:

- ★ É usado para saber quando um container está pronto para aceitar tráfego de rede.

>>> *LivenessProbe, ReadinessProbe e StartUpProbe*

Os 3 são controlados pelo *kubelet*

LivenessProbe:

- ★ Tentar fazer a aplicação ficar mais disponível removendo *bugs*
 - ★ Pode perceber um *deadlock* na aplicação

ReadinessProbe:

- ★ É usado para saber quando um container está pronto para aceitar tráfego de rede.
 - ★ Pode ser usado para controlar os pods que servem como *backend*

>>> *LivenessProbe, ReadinessProbe e StartUpProbe*

Os 3 são controlados pelo *kubelet*

LivenessProbe:

- ★ Tentar fazer a aplicação ficar mais disponível removendo *bugs*
 - ★ Pode perceber um *deadlock* na aplicação

ReadinessProbe:

- ★ É usado para saber quando um container está pronto para aceitar tráfego de rede.
 - ★ Pode ser usado para controlar os pods que servem como *backend*
 - ★ Se o pod não estiver pronto, ele é removido do *Load Balancer* do serviço

StartupProbe:

- * É usado para saber se o container da aplicação começou.

StartupProbe:

- ★ É usado para saber se o container da aplicação começou.
- ★ Pode ser configurado para desativar *liveness* e *readiness* até o *startup* tenha sucesso.

StartupProbe:

- ★ É usado para saber se o container da aplicação começou.
 - ★ Pode ser configurado para desativar *liveness* e *readiness* até o *startup* tenha sucesso.
 - ★ Dessa forma o *kubelet* não vai matar aplicações lentas antes de elas inicializarem.

>>> **Cluster Local**

★ Ambientes menores (Recursos da máquina local);

>>> Cluster Local

- * Ambientes menores (Recursos da máquina local);
- * Para aprendizado;

>>> Cluster Local

- * Ambientes menores (Recursos da máquina local);
- * Para aprendizado;
- * Não são feitos para produção

>>> Ferramentas

Ferramentas para criação de cluster **local** do
kubernetes:

>>> Ferramentas

Ferramentas para criação de cluster **local** do kubernetes:

- ★ kind (Kubernetes in Docker)
 - ★ Cluster usando containers de docker para criar os nós.

>>> Ferramentas

Ferramentas para criação de cluster **local** do kubernetes:

- * kind (Kubernetes in Docker)
 - * Cluster usando containers de docker para criar os nós.
- * kubeadm
 - * Cluster usando um container runtime.

>>> Ferramentas

Ferramentas para criação de cluster **local** do kubernetes:

- * kind (Kubernetes in Docker)
 - * Cluster usando containers de docker para criar os nós.
- * kubeadm
 - * Cluster usando um container runtime.
- * minikube
 - * Cluster usando containeres ou máquinas virtuais.

>>> Criando o cluster com minikube

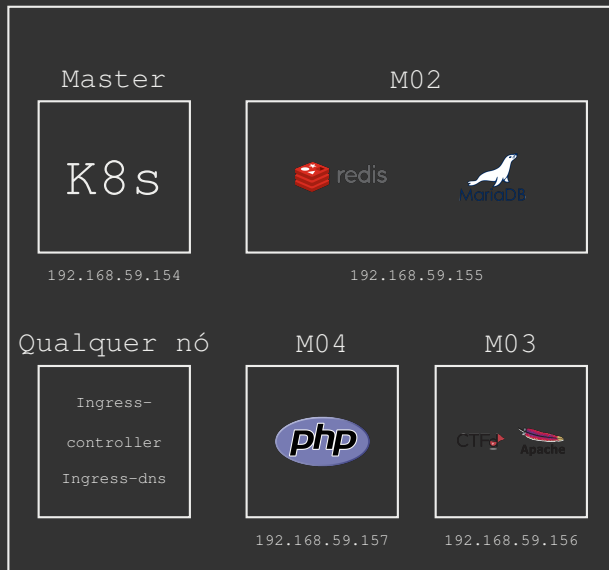
Criação do cluster utilizado na parte prática:

Comando *minikube*

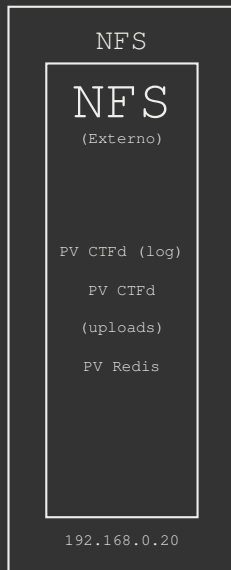
```
minikube start --nodes='4' \  
--network-plugin=cni \  
--cni=calico \  
--profile=multinode \  
--vm-driver=virtualbox \  
--kubernetes-version=v1.24.3
```

>>> Nós

Computador 1



Server NFS



>>> Recursos – Namespaces

- ★ Os *namespaces* permitem isolar grupos de recursos em um cluster

²Kubernetes Documentation

>>> Recursos - Namespaces

- ★ Os *namespaces* permitem isolar grupos de recursos em um cluster
 - ★ Os *namespaces* são uma forma de dividir o cluster para várias pessoas com limite de recursos.²

²Kubernetes Documentation

>>> Recursos - Namespaces

- ★ Os *namespaces* permitem isolar grupos de recursos em um cluster
 - ★ Os *namespaces* são uma forma de dividir o cluster para várias pessoas com limite de recursos.²
 - ★ Os administradores do cluster devem dividir os recursos

²Kubernetes Documentation

>>> Recursos – Namespaces

- ★ Os *namespaces* permitem isolar grupos de recursos em um cluster
 - ★ Os *namespaces* são uma forma de dividir o cluster para várias pessoas com limite de recursos.²
 - ★ Os administradores do cluster devem dividir os recursos
- ★ O nome dos recursos precisam ser únicos em cada namespace

²Kubernetes Documentation

>>> Recursos – Namespaces

- ★ Os *namespaces* permitem isolar grupos de recursos em um cluster
 - ★ Os *namespaces* são uma forma de dividir o cluster para várias pessoas com limite de recursos.²
 - ★ Os administradores do cluster devem dividir os recursos
- ★ O nome dos recursos precisam ser únicos em cada namespace
- ★ Não são todos os objetos que aceitam o isolamento do *namespace*

²Kubernetes Documentation

>>> Recursos - Namespaces

Para ver os recursos que estão e não são gerenciados por namespace

values.yaml

```
# In a namespace
```

```
kubectl api-resources --namespaced=true
```

```
# Not in a namespace
```

```
kubectl api-resources --namespaced=false
```


>>> Namespaces

ctfd

```
secrets (mariaDB)
```

redis

```
secrets (mariaDB)  
db-202{1,2}-statefulset  
db-202{1,2}-service
```

ingress-nginx

```
ingress-nginx-controller  
ingress-dns
```

site

```
site-hpa  
site-deployment  
site-service  
site-ingress
```

database

```
secrets (mariaDB)  
db-202{1,2}-statefulset  
db-202{1,2}-service
```

challs

```
{uccdi,xor-otp}-hpa  
{uccdi,xor-otp}-deployment  
{uccdi,xor-otp}-service  
uccdi-ingress
```

>>> Recursos - Daemonsets

- * O *DaemonSet* garante que todos os nós ou os nós selecionados rodem uma cópia de um *Pod*

>>> Recursos - Daemonsets

- * O *DaemonSet* garante que todos os nós ou os nós selecionados rodem uma cópia de um *Pod*
- * Quando nós são adicionados nos clusters, os *Pods* são adicionados nesses nós e vice versa

>>> Recursos - Daemonsets

- * O *DaemonSet* garante que todos os nós ou os nós selecionados rodem uma cópia de um *Pod*
- * Quando nós são adicionados nos clusters, os *Pods* são adicionados nesses nós e vice versa
- * Deletar o *DaemonSet* mata os *Pods* que foram criados

>>> Recursos - Daemonsets

- * O *DaemonSet* garante que todos os nós ou os nós selecionados rodem uma cópia de um *Pod*
- * Quando nós são adicionados nos clusters, os *Pods* são adicionados nesses nós e vice versa
- * Deletar o *DaemonSet* mata os *Pods* que foram criados
- * Alguns exemplos de uso:
 - * Rodar um *cluster storage daemon* em todos os nós

>>> Recursos - Daemonsets

- * O *DaemonSet* garante que todos os nós ou os nós selecionados rodem uma cópia de um *Pod*
- * Quando nós são adicionados nos clusters, os *Pods* são adicionados nesses nós e vice versa
- * Deletar o *DaemonSet* mata os *Pods* que foram criados
- * Alguns exemplos de uso:
 - * Rodar um *cluster storage daemon* em todos os nós
 - * Rodar um *logs collection daemon* em todos os nós

>>> Recursos - Daemonsets

- * O *DaemonSet* garante que todos os nós ou os nós selecionados rodem uma cópia de um *Pod*
- * Quando nós são adicionados nos clusters, os *Pods* são adicionados nesses nós e vice versa
- * Deletar o *DaemonSet* mata os *Pods* que foram criados
- * Alguns exemplos de uso:
 - * Rodar um *cluster storage daemon* em todos os nós
 - * Rodar um *logs collection daemon* em todos os nós
 - * Rodar um *node monitoring daemon* em todos os nós

>>> Recursos - ConfigMaps

- ★ O *ConfigMap* é usado para guardar dados não confidenciais em pares de chave-valor

>>> Recursos - ConfigMaps

- ★ O *ConfigMap* é usado para guardar dados não confidenciais em pares de chave-valor
- ★ Pode ser usado como:

>>> Recursos - ConfigMaps

- ★ O *ConfigMap* é usado para guardar dados não confidenciais em pares de chave-valor
- ★ Pode ser usado como:
 - ★ Variáveis de ambiente

>>> Recursos - ConfigMaps

- * O *ConfigMap* é usado para guardar dados não confidenciais em pares de chave-valor
- * Pode ser usado como:
 - * Variáveis de ambiente
 - * Argumentos de comando de linha

>>> Recursos - ConfigMaps

- * O *ConfigMap* é usado para guardar dados não confidenciais em pares de chave-valor
- * Pode ser usado como:
 - * Variáveis de ambiente
 - * Argumentos de comando de linha
 - * Arquivos de configuração em um volume

>>> Recursos - ConfigMaps

- ★ Aplicação no projeto prático:

Variáveis de ambiente

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: ctf-d-env-configmap
  namespace: ctf-d
data:
  DB_FILE: '/etc/ctf-d-db/database'
  DB_USER_FILE: '/etc/ctf-d-db/username'
  DB_PASS_FILE: '/etc/ctf-d-db/password'
  ACCESS_LOG: '-'
  ERROR_LOG: '-'
  UPLOAD_FOLDER: '/var/uploads'
  LOG_FOLDER: '/var/log/CTFd'
  REVERSE_PROXY: 'true'
```

>>> Recursos - Secrets

- * O *secret* é um objeto que contém uma pequena quantidade de dados **sensíveis**

³[https:](https://kubernetes.io/docs/concepts/security/secrets-good-practices/)

[//kubernetes.io/docs/concepts/security/secrets-good-practices/](https://kubernetes.io/docs/concepts/security/secrets-good-practices/)

>>> Recursos - Secrets

- * O *secret* é um objeto que contém uma pequena quantidade de dados **sensíveis**
- * Como senhas, tokens, chaves, etc...

³https:

//kubernetes.io/docs/concepts/security/secrets-good-practices/

>>> Recursos - Secrets

- * O *secret* é um objeto que contém uma pequena quantidade de dados **sensíveis**
- * Como senhas, tokens, chaves, etc...
- * Dessa forma não é preciso incluir dados confidenciais no seu código

³https:

//kubernetes.io/docs/concepts/security/secrets-good-practices/

>>> Recursos – Secrets

- * O *secret* é um objeto que contém uma pequena quantidade de dados **sensíveis**
- * Como senhas, tokens, chaves, etc...
- * Dessa forma não é preciso incluir dados confidenciais no seu código
- * Algumas boas práticas para manter o *secrets* seguro:³
 - * Encryption at Rest for Secrets

³https:

//kubernetes.io/docs/concepts/security/secrets-good-practices/

>>> Recursos – Secrets

- * O *secret* é um objeto que contém uma pequena quantidade de dados **sensíveis**
- * Como senhas, tokens, chaves, etc...
- * Dessa forma não é preciso incluir dados confidenciais no seu código
- * Algumas boas práticas para manter o *secrets* seguro:³
 - * Encryption at Rest for Secrets
 - * Enable or configure RBAC rules with least-privilege access to Secrets

³https:

//kubernetes.io/docs/concepts/security/secrets-good-practices/

>>> Recursos – Secrets

- * O *secret* é um objeto que contém uma pequena quantidade de dados **sensíveis**
- * Como senhas, tokens, chaves, etc...
- * Dessa forma não é preciso incluir dados confidenciais no seu código
- * Algumas boas práticas para manter o *secrets* seguro:³
 - * Encryption at Rest for Secrets
 - * Enable or configure RBAC rules with least-privilege access to Secrets
 - * Restrict Secret access to specific containers

³https:

//kubernetes.io/docs/concepts/security/secrets-good-practices/

>>> Recursos – Secrets

- * O *secret* é um objeto que contém uma pequena quantidade de dados **sensíveis**
- * Como senhas, tokens, chaves, etc...
- * Dessa forma não é preciso incluir dados confidenciais no seu código
- * Algumas boas práticas para manter o *secrets* seguro:³
 - * Encryption at Rest for Secrets
 - * Enable or configure RBAC rules with least-privilege access to Secrets
 - * Restrict Secret access to specific containers
 - * Consider using external Secret store providers

³https:

//kubernetes.io/docs/concepts/security/secrets-good-practices/

>>> Recursos – Secrets

- ★ Aplicação no projeto prático:

Volumes

```
apiVersion: v1
kind: Secret
metadata:
  name: ctfd-db-2022
  namespace: database
type: Opaque
data:
  database: 'Y3RmZGRiCg=='
  username: 'Y3RmZHVzZXIK'
  password: 'Y3RmZHBhc3MK'
  root-pass: 'Y3RmZHJvb3RwYXNzCg=='
```

```
>>> Recursos - Services
```

Secrets

>>> Recursos - Ingress

- ★ **Ingress:** Gerencia acesso externo para os serviços dentro do cluster (Geralmente o protocolo **HTTP**)
- ★ **Ingress Controllers:** É necessário para que o as regras o **Ingress** funcionem

⁴Outros exemplos de Ingress-Controllers

>>> Recursos – Ingress

- ★ **Ingress:** Gerencia acesso externo para os serviços dentro do cluster (Geralmente o protocolo **HTTP**)
 - ★ Provê balanceamento de carga, hosts virtuais e pode ser um terminal SSL/TLS
- ★ **Ingress Controllers:** É necessário para que o as regras o **Ingress** funcionem
- ★ Alguns exemplos são os **Ingress Controllers** da AWS, GCE e nginx⁴

⁴Outros exemplos de Ingress-Controllers

>>> Recursos - Ingress

- ★ **Ingress:** Gerencia acesso externo para os serviços dentro do cluster (Geralmente o protocolo **HTTP**)
 - ★ Provê balanceamento de carga, hosts virtuais e pode ser um terminal SSL/TLS
 - ★ O ingress não expõe portas ou protocolos
- ★ **Ingress Controllers:** É necessário para que o as regras o **Ingress** funcionem
- ★ Alguns exemplos são os **Ingress Controllers** da AWS, GCE e nginx⁴

⁴Outros exemplos de Ingress-Controllers

>>> Recursos – Ingress

- ★ **Ingress:** Gerencia acesso externo para os serviços dentro do cluster (Geralmente o protocolo **HTTP**)
 - ★ Provê balanceamento de carga, hosts virtuais e pode ser um terminal SSL/TLS
 - ★ O ingress não expõe portas ou protocolos
 - ★ Serviços que não são **HTTP** ou **HTTPS** são expostos usando **Serviços Nodeport** ou **LoadBalancer**
- ★ **Ingress Controllers:** É necessário para que o as regras o **Ingress** funcionem
- ★ Alguns exemplos são os **Ingress Controllers** da AWS, GCE e nginx⁴

⁴Outros exemplos de Ingress-Controllers

>>> Recursos – Persistência de dados

Secrets

>>> Recursos - Jobs e CronJobs

- * **Jobs:** Cria um ou mais *Pods* para executar uma tarefa

>>> Recursos - Jobs e CronJobs

- ★ **Jobs:** Cria um ou mais *Pods* para executar uma tarefa
 - ★ Continua reexecutando os *Pods* até que o número especificado de tentativas dele termine (*backoff failure policy*)

>>> Recursos - Jobs e CronJobs

- * **Jobs:** Cria um ou mais *Pods* para executar uma tarefa
 - * Continua reexecutando os *Pods* até que o número especificado de tentativas dele termine (*backoff failure policy*)
 - * Assim que um *Pod* completa com sucesso, o *Job* continua a executar até o número de sucessos seja atingido

>>> Recursos - Jobs e CronJobs

- * **Jobs:** Cria um ou mais *Pods* para executar uma tarefa
 - * Continua reexecutando os *Pods* até que o número especificado de tentativas dele termine (*backoff failure policy*)
 - * Assim que um *Pod* completa com sucesso, o *Job* continua a executar até o número de sucessos seja atingido
 - * Suspende/Apagar um **Job** vai apagar os *Pods* criados.

>>> Recursos - Jobs e CronJobs

- * **Jobs:** Cria um ou mais *Pods* para executar uma tarefa
 - * Continua reexecutando os *Pods* até que o número especificado de tentativas dele termine (*backoff failure policy*)
 - * Assim que um *Pod* completa com sucesso, o *Job* continua a executar até o número de sucessos seja atingido
 - * Suspende/Apagar um **Job** vai apagar os *Pods* criados.
 - * Quando um **Job** termina os *Pods* e o **Job** em si não são apagados

>>> Recursos - Jobs e CronJobs

- * **Jobs:** Cria um ou mais *Pods* para executar uma tarefa
 - * Continua reexecutando os *Pods* até que o número especificado de tentativas dele termine (*backoff failure policy*)
 - * Assim que um *Pod* completa com sucesso, o *Job* continua a executar até o número de sucessos seja atingido
 - * Suspende/Apagar um **Job** vai apagar os *Pods* criados.
 - * Quando um **Job** termina os *Pods* e o **Job** em si não são apagados
 - * É possível rodar **Jobs** em paralelo

>>> Recursos - Jobs e CronJobs

- * **Jobs:** Cria um ou mais *Pods* para executar uma tarefa
 - * Continua reexecutando os *Pods* até que o número especificado de tentativas dele termine (*backoff failure policy*)
 - * Assim que um *Pod* completa com sucesso, o *Job* continua a executar até o número de sucessos seja atingido
 - * Suspende/Apagar um **Job** vai apagar os *Pods* criados.
 - * Quando um **Job** termina os *Pods* e o **Job** em si não são apagados
 - * É possível rodar **Jobs** em paralelo
- * **Cronjobs:** Executa uma tarefa periodicamente em um determinado cronograma, escrito no formato Cron.
 - * O fuso horário para o contêiner executando o **kube-controller-manager** determina o fuso horário que o **CronJob** utiliza

>>> Recursos - Jobs e CronJobs

- * **Jobs:** Cria um ou mais *Pods* para executar uma tarefa
 - * Continua reexecutando os *Pods* até que o número especificado de tentativas dele termine (*backoff failure policy*)
 - * Assim que um *Pod* completa com sucesso, o *Job* continua a executar até o número de sucessos seja atingido
 - * Suspende/Apagar um **Job** vai apagar os *Pods* criados.
 - * Quando um **Job** termina os *Pods* e o **Job** em si não são apagados
 - * É possível rodar **Jobs** em paralelo
- * **Cronjobs:** Executa uma tarefa periodicamente em um determinado cronograma, escrito no formato Cron.
 - * O fuso horário para o contêiner executando o **kube-controller-manager** determina o fuso horário que o **CronJob** utiliza
 - * O nome não pode ter mais que 52 caracteres

>>> Recursos - Jobs e CronJobs

- * **Jobs:** Cria um ou mais *Pods* para executar uma tarefa
 - * Continua reexecutando os *Pods* até que o número especificado de tentativas dele termine (*backoff failure policy*)
 - * Assim que um *Pod* completa com sucesso, o *Job* continua a executar até o número de sucessos seja atingido
 - * Suspende/Apagar um **Job** vai apagar os *Pods* criados.
 - * Quando um **Job** termina os *Pods* e o **Job** em si não são apagados
 - * É possível rodar **Jobs** em paralelo
- * **Cronjobs:** Executa uma tarefa periodicamente em um determinado cronograma, escrito no formato Cron.
 - * O fuso horário para o contêiner executando o **kube-controller-manager** determina o fuso horário que o **CronJob** utiliza
 - * O nome não pode ter mais que 52 caracteres
 - * Podem fazer backup, enviar emails, etc...

```
>>> Recursos - StatefulSet
```

Secrets

>>> Recursos - HPA

Horizontal Pod Autoscaling

- ★ Automaticamente atualiza recursos (como *Deployment* ou *StatefulSet*), tem o objetivo de escalar automaticamente de acordo com a demanda

>>> Recursos - HPA

Horizontal Pod Autoscaling

- * Automaticamente atualiza recursos (como *Deployment* ou *StatefulSet*), tem o objetivo de escalar automaticamente de acordo com a demanda
- * Ou seja, mais *pods* **iguais** vão ser criados

Horizontal Pod Autoscaling

- * Automaticamente atualiza recursos (como *Deployment* ou *StatefulSet*), tem o objetivo de escalar automaticamente de acordo com a demanda
- * Ou seja, mais *Pods* **iguais** vão ser criados
- * Quando a carga da aplicação abaixar os *Pods* vão se reduzir até a quantidade mínima

>>> Recursos - HPA

Horizontal Pod Autoscaling

- * Automaticamente atualiza recursos (como *Deployment* ou *StatefulSet*), tem o objetivo de escalar automaticamente de acordo com a demanda
- * Ou seja, mais *Pods* **iguais** vão ser criados
- * Quando a carga da aplicação abaixar os *Pods* vão se reduzir até a quantidade mínima
- * Não se aplica a objetos que não podem ser escalados (Exemplo: **DaemonSet**)

Horizontal Pod Autoscaling

- * Automaticamente atualiza recursos (como *Deployment* ou *StatefulSet*), tem o objetivo de escalar automaticamente de acordo com a demanda
- * Ou seja, mais *Pods* **iguais** vão ser criados
- * Quando a carga da aplicação abaixar os *Pods* vão se reduzir até a quantidade mínima
- * Não se aplica a objetos que não podem ser escalados (Exemplo: **DaemonSet**)
- * Periodicamente o **HPA** ajusta os **Pods** do seu objeto observando métricas (CPU, memória ou métricas customizadas)

>>> **Recursos - RBAC**

Daemonsets - conceito

>>> Recursos - CRD

Daemonsets - conceito

>>> **Describe**

É possível ver mais informações sobre determinado recurso ou grupo de recursos

```
ctfd-db-2021-secret:
  Type: Secret (a volume populated by a Secret)
  SecretName: ctfd-db-2021
  Optional: false
kube-api-access-nthvj:
  Type: Projected (a volume that contains injected data from multiple sources)
  TokenExpirationSeconds: 3607
  ConfigMapName: kube-root-ca.crt
  ConfigMapOptional: <nil>
  DownwardAPI: true
QoS Class: BestEffort
Node-Selectors: <none>
Tolerations:
  node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
  node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
  type=db

Events:
  Type      Reason      Age      From      Message
  ----      -
Warning    FailedScheduling  58s (x4 over 65s)  default-scheduler  0/4 nodes are available: 4 pod has unbound immediate PersistentVolumeClaims.
preemption: 0/4 nodes are available: 4 Preemption is not helpful for scheduling.
```

Figure: Motivo de falha do banco de dados

>>> **Logs**

Possibilita ver a causa de erro dentro do container:

```
tsuyoshi 📁 opus 📁 final 歩<k logs -n ctfd ctfd-2021-deployment-56c9d757b7-db8n8 -c wait-database
Server:      10.96.0.10
Address 1: 10.96.0.10

nslookup: can't resolve 'ctfd-db-2021-service.database.svc.cluster.local'
waiting for db
nslookup: can't resolve 'ctfd-db-2021-service.database.svc.cluster.local'
Server:      10.96.0.10
Address 1: 10.96.0.10

waiting for db
Server:      10.96.0.10
Address 1: 10.96.0.10

nslookup: can't resolve 'ctfd-db-2021-service.database.svc.cluster.local'
waiting for db
Server:      10.96.0.10
Address 1: 10.96.0.10
```

Figure: Log do *initContainer*

>>> **exec**

Verificar *Mounts, secrets, env, conexão, etc...*:

```
△tsuyoshi 📁 opus 📁 final 歩<k exec -ti -n ctfd ctfd-2021-deployment-78cf9d7fc6-6rjmg -- bash
Defaulted container "ctfd-2021" out of: ctfd-2021, wait-database (init)
ctfd@ctfd-2021-deployment-78cf9d7fc6-6rjmg:/opt/CTFd$ env
KUBERNETES_SERVICE_PORT_HTTPS=443
REDIS_URL=redis://ctfd-redis-master.redis.svc.cluster.local:6379
CTFD_2022_SERVICE_PORT_8000_TCP_PORT=8000
KUBERNETES_SERVICE_PORT=443
CTFD_2022_SERVICE_PORT_8000_TCP_PROTO=tcp
HOSTNAME=ctfd-2021-deployment-78cf9d7fc6-6rjmg
PYTHON_VERSION=3.9.16
CTFD_2021_SERVICE_PORT_8000_TCP_PROTO=tcp
CTFD_2022_SERVICE_HOST=10.98.92.139
ERROR_LOG=-
CTFD_2021_SERVICE_PORT=tcp://10.106.127.63:8000
PWD=/opt/CTFd
PYTHON_SETUPTOOLS_VERSION=58.1.0
CTFD_2022_SERVICE_PORT_8000_TCP=tcp://10.98.92.139:8000
CTFD_2021_SERVICE_PORT_8000_TCP_ADDR=10.106.127.63
UPLOAD_FOLDER=/var/uploads
CTFD_2021_SERVICE_PORT_8000_TCP_PORT=8000
```

Figure: Verificar variáveis de ambiente

```
>>> events
```

- * Os eventos são um relatório de um evento que ocorreu dentro do *cluster*


```
>>> events
```

- * Os eventos são um relatório de um evento que ocorreu dentro do *cluster*
- * Os eventos são informativos e informações adicionais

>>> events

- ★ Os eventos são um relatório de um evento que ocorreu dentro do *cluster*
- ★ Os eventos são informativos e informações adicionais

```
tsuyoshi opus final 書<k get events
LAST SEEN   TYPE      REASON
11m         Warning   FailedGetScale
4m16s       Warning   FailedGetScale
4s          Warning   FailedGetScale
11m         Warning   FailedGetScale
4m18s       Warning   FailedGetScale
4s          Warning   FailedGetScale
13m         Warning   FailedGetScale
4m57s       Warning   FailedGetScale
14s         Warning   FailedGetScale
37m         Normal    NodeHasSufficientMemory
37m         Normal    NodeHasNoDiskPressure
37m         Normal    RegisteredNode
37m         Normal    Starting
35m         Normal    NodeHasSufficientMemory
35m         Normal    NodeHasNoDiskPressure
35m         Normal    RegisteredNode
35m         Normal    Starting
34m         Normal    NodeHasSufficientMemory
34m         Normal    NodeHasNoDiskPressure
34m         Normal    RegisteredNode
34m         Normal    Starting
39m         Normal    NodeHasSufficientMemory
39m         Normal    NodeHasNoDiskPressure
39m         Normal    NodeHasSufficientPID
39m         Normal    Starting

OBJECT
horizontalpodautoscaler/ctfd-2021-chall-xor-otp-hpa
horizontalpodautoscaler/ctfd-2021-chall-xor-otp-hpa
horizontalpodautoscaler/ctfd-2021-chall-xor-otp-hpa
horizontalpodautoscaler/ctfd-2022-chall-uccdl-hpa
horizontalpodautoscaler/ctfd-2022-chall-uccdl-hpa
horizontalpodautoscaler/ctfd-2022-chall-uccdl-hpa
horizontalpodautoscaler/ctfd-site-hpa
horizontalpodautoscaler/ctfd-site-hpa
horizontalpodautoscaler/ctfd-site-hpa
horizontalpodautoscaler/ctfd-site-hpa

MESSAGE
deployments/scale.apps "ctfd-2021-chall-xor-otp-deployment" not found
deployments/scale.apps "ctfd-2021-chall-xor-otp-deployment" not found
deployments/scale.apps "ctfd-2021-chall-xor-otp-deployment" not found
deployments/scale.apps "ctfd-2022-chall-uccdl-deployment" not found
deployments/scale.apps "ctfd-2022-chall-uccdl-deployment" not found
deployments/scale.apps "ctfd-2022-chall-uccdl-deployment" not found
deployments/scale.apps "ctfd-site-deployment" not found
deployments/scale.apps "ctfd-site-deployment" not found
deployments/scale.apps "ctfd-site-deployment" not found
Node multinode-m02 status is now: NodeHasSufficientMemory
Node multinode-m02 status is now: NodeHasNoDiskPressure
Node multinode-m02 event: Registered Node multinode-m02 in Controller

Node multinode-m03 status is now: NodeHasSufficientMemory
Node multinode-m03 status is now: NodeHasNoDiskPressure
Node multinode-m03 event: Registered Node multinode-m03 in Controller

Node multinode-m04 status is now: NodeHasSufficientMemory
Node multinode-m04 status is now: NodeHasNoDiskPressure
Node multinode-m04 event: Registered Node multinode-m04 in Controller

Node multinode status is now: NodeHasSufficientMemory
Node multinode status is now: NodeHasNoDiskPressure
Node multinode status is now: NodeHasSufficientPID
Starting kubelet.
```

Figure: Eventos

>>> Disponibilidade e Escalabilidade

Para criar um *cluster* de alta disponibilidade:

- ★ Separar *control plane* do nó dos *workers*
- ★ Replicar os componentes do *control plane* em múltiplos nós
- ★ Balancear o tráfego de rede na *API* do *cluster*
- ★ Ter nós *workers* suficientes ou ter nós que conseguem ficar disponíveis rapidamente

Escalabilidade

- ★ Ser capaz de receber as demandas que a aplicação recebe normalmente
- ★ Ser capaz de aguentar as demandas em eventos especiais ou datas comemorativas
- ★ Planejar como escalar ou reduzir os recursos do *control plane* e dos *workers*

>>> *Control plane* em produção

- ★ Gerenciar certificados

- ★

>>> *Control plane* em produção

- ★ Gerenciar certificados

- ★ **Alta disponibilidade:** Estender o *control plane*

- ★

>>> *Control plane* em produção

- * Gerenciar certificados
- * **Alta disponibilidade:** Estender o *control plane*
 - * *Load Balancer* para o **apiserver**

*

>>> *Control plane* em produção

- * Gerenciar certificados
- * **Alta disponibilidade:** Estender o *control plane*
 - * *Load Balancer* para o **apiserver**
 - * *Backup* do **etcd**

*

>>> *Control plane* em produção

- * Gerenciar certificados
- * **Alta disponibilidade:** Estender o *control plane*
 - * *Load Balancer* para o **apiserver**
 - * Backup do **etcd**
 - * Múltiplos *control planes* em máquinas separadas

*

>>> *Control plane* em produção

- ★ Gerenciar certificados
- ★ **Alta disponibilidade:** Estender o *control plane*
 - ★ *Load Balancer* para o **apiserver**
 - ★ Backup do **etcd**
 - ★ Múltiplos *control planes* em máquinas separadas
 - ★ Espalhar por múltiplos *data centers*. (Continua disponível mesmo que uma zona fique fora do ar)

★

>>> *Control plane* em produção

- ★ Gerenciar certificados
- ★ **Alta disponibilidade:** Estender o *control plane*
 - ★ *Load Balancer* para o **apiserver**
 - ★ Backup do **etcd**
 - ★ Múltiplos *control planes* em máquinas separadas
 - ★ Espalhar por múltiplos *data centers*. (Continua disponível mesmo que uma zona fique fora do ar)
 - ★ Gerenciar *cluster*: Certificados, fazer *upgrade*, etc...

★

>>> *Worker Nodes* em produção

- ★ Gerenciar os recursos corretamente (Colocar a quantidade apropriada de Memória, CPU e armazenamento nos nós)

>>> *Worker Nodes* em produção

- ★ Gerenciar os recursos corretamente (Colocar a quantidade apropriada de Memória, CPU e armazenamento nos nós)
- ★ Validar se os nós tem os recursos necessários para entrar no cluster

>>> *Worker Nodes* em produção

- ★ Gerenciar os recursos corretamente (Colocar a quantidade apropriada de Memória, CPU e armazenamento nos nós)
- ★ Validar se os nós tem os recursos necessários para entrar no cluster
- ★ Escalabilidade dos nós

>>> *Worker Nodes* em produção

- ★ Gerenciar os recursos corretamente (Colocar a quantidade apropriada de Memória, CPU e armazenamento nos nós)
- ★ Validar se os nós tem os recursos necessários para entrar no cluster
- ★ Escalabilidade dos nós
- ★ *Health checks* para verificar se os nós estão saudáveis

>>> **Worker Nodes** em produção

- ★ Gerenciar os recursos corretamente (Colocar a quantidade apropriada de Memória, CPU e armazenamento nos nós)
- ★ Validar se os nós tem os recursos necessários para entrar no cluster
- ★ Escalabilidade dos nós
- ★ *Health checks* para verificar se os nós estão saudáveis
 - ★ **Node Problem Detector:** Pode ser rodado como um *daemon* ou como um *daemonset*

>>> **Worker Nodes** em produção

- ★ Gerenciar os recursos corretamente (Colocar a quantidade apropriada de Memória, CPU e armazenamento nos nós)
- ★ Validar se os nós tem os recursos necessários para entrar no cluster
- ★ Escalabilidade dos nós
- ★ *Health checks* para verificar se os nós estão saudáveis
 - ★ **Node Problem Detector:** Pode ser rodado como um *daemon* ou como um *daemonset*
 - ★ Usam vários *daemons* para coletar informações e reportam essas condições usando as condições do nó ou eventos.

>>> **Usuários no ambiente de produção**

- * **Autenticação:** Autenticar os usuários do **apiserver** com certificados, tokens, proxy de autenticação ou HTTP
 - * É possível usar **Kerberos** ou **LDAP** para uma autenticação mais avançada

>>> **Usuários no ambiente de produção**

- * **Autenticação:** Autenticar os usuários do **apiserver** com certificados, tokens, proxy de autenticação ou HTTP
 - * É possível usar **Kerberos** ou **LDAP** para uma autenticação mais avançada

>>> *Usuários no ambiente de produção*

- ★ **Autenticação:** Autenticar os usuários do **apiserver** com certificados, tokens, proxy de autenticação ou HTTP
- ★ **Autorização:** RBAC ou ABAC

>>> **Usuários no ambiente de produção**

- ★ **Autenticação:** Autenticar os usuários do **apiserver** com certificados, tokens, proxy de autenticação ou HTTP
- ★ **Autorização:** RBAC ou ABAC
 - ★ **RBAC (*Role-based access control*):** Permissões para namespaces específicos (*Role* ou para o *cluster* inteiro). Com **RoleBindings** e **ClusterBindings** é possível relacionar com usuários específicos

>>> **Usuários no ambiente de produção**

- ★ **Autenticação:** Autenticar os usuários do **apiserver** com certificados, tokens, proxy de autenticação ou HTTP
- ★ **Autorização: RBAC ou ABAC**
 - ★ **RBAC (*Role-based access control*):** Permissões para namespaces específicos (*Role* ou para o *cluster* inteiro). Com **RoleBindings** e **ClusterBindings** é possível relacionar com usuários específicos
 - ★ Criar certificados para usuários
CertificateSigningRequest

>>> **Usuários no ambiente de produção**

- * **Autenticação:** Autenticar os usuários do **apiserver** com certificados, tokens, proxy de autenticação ou HTTP
- * **Autorização: RBAC ou ABAC**
 - * **RBAC (*Role-based access control*):** Permissões para namespaces específicos (*Role* ou para o *cluster* inteiro). Com **RoleBindings** e **ClusterBindings** é possível relacionar com usuários específicos
 - * Criar certificados para usuários
CertificateSigningRequest
 - * **ABAC (*Attribute-based access control*):** Criar políticas baseadas em atributos do *cluster*.

>>> **Usuários no ambiente de produção**

- ★ **Autenticação:** Autenticar os usuários do **apiserver** com certificados, tokens, proxy de autenticação ou HTTP
- ★ **Autorização: RBAC ou ABAC**
 - ★ **RBAC (*Role-based access control*):** Permissões para namespaces específicos (*Role* ou para o *cluster* inteiro). Com **RoleBindings** e **ClusterBindings** é possível relacionar com usuários específicos
 - ★ Criar certificados para usuários
CertificateSigningRequest
 - ★ **ABAC (*Attribute-based access control*):** Criar políticas baseadas em atributos do *cluster*.
 - ★ Usuários podem acessar determinados tipos de atributos (*pod*, *namespaces*, *apiGroups*)

>>> Helm

"Helm helps you manage Kubernetes applications --
Helm Charts help you define, install, and upgrade
even the most complex Kubernetes application." [1]

>>> Helm

"Helm helps you manage Kubernetes applications -- Helm Charts help you define, install, and upgrade even the most complex Kubernetes application." [1]

★ Instalar charts

>>> Helm

"Helm helps you manage Kubernetes applications -- Helm Charts help you define, install, and upgrade even the most complex Kubernetes application." [1]

- * Instalar charts

- * Criar charts

```
>>> Helm
```

```
"Helm helps you manage Kubernetes applications --  
Helm Charts help you define, install, and upgrade  
even the most complex Kubernetes application." [1]
```

- * Instalar charts
- * Criar charts
- * Remover charts

```
>>> Helm
```

```
"Helm helps you manage Kubernetes applications --  
Helm Charts help you define, install, and upgrade  
even the most complex Kubernetes application." [1]
```

- * Instalar charts
- * Criar charts
- * Remover charts
- * Escolher repositórios

>>> Charts

"Charts are easy to create, version, share, and
publish -- so start using Helm and stop the
copy-and-paste." [1]

>>> Charts

"Charts are easy to create, version, share, and
publish -- so start using Helm and stop the
copy-and-paste." [1]

★ Criar

```
>>> Charts
```

```
"Charts are easy to create, version, share, and  
publish -- so start using Helm and stop the  
copy-and-paste." [1]
```

```
★ Criar
```

```
★ Versionar
```

>>> Charts

"Charts are easy to create, version, share, and publish -- so start using Helm and stop the copy-and-paste." [1]

- ★ Criar
- ★ Versionar
- ★ Compartilhar

>>> Charts

"Charts are easy to create, version, share, and publish -- so start using Helm and stop the copy-and-paste." [1]

- ★ Criar
- ★ Versionar
- ★ Compartilhar
- ★ Publicar

>>> **Helm**

Instalar o *Redis*

```
helm install ctfd-redis -f ./redis/values.yaml \  
-n redis \  
bitnami/redis
```

Instalar o *redis* no namespace **redis** com os valores **./redis/values.yaml** do repositório **<https://charts.bitnami.com/bitnami>**

>>> **Helm**

Como foi utilizado no projeto:

values.yaml

```
architecture: standalone
auth:
  enabled: false
master:
  tolerations:
    - key: "type"
      operator: "Equal"
      value: "db"
securityContext:
  enabled: true
  fsGroup: 2000
  runAsUser: 0
```

>>> **Helm**

values.yaml

```
master:
  persistence:
    enabled: true
    storageClass: "nfs"
    accessModes:
      - ReadWriteOnce
    size: 8Gi
    matchLabels:
      name: ctfd-redis-pv
```

>>> Referencias

- [1] Helm Authors. *Helm*. URL: <https://helm.sh/> (visited on 01/12/2023).
- [2] The Kubernetes Authors. *Kubernetes Documentation*. URL: <https://kubernetes.io/docs/home/> (visited on 01/12/2023).