>>> >>> Kubernetes

Name: Henrique Tsuyoshi Yara (OPUS-software)
Date: January 11, 2023

Figure: Kubernetes logo

>>> Índice

>>> Título

Lista:

* a;
* b;

>>> 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)

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

* 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

* **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 performace 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)

>>> Diferenças

Kubernetes vs Docker (diferenças, em que casos é melhor utilizá-los) [1]

[1] IBM Kubernetes vs Docker: Why not both?

>>> 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 (indepentente do *API server*)

[2] https://anote.dev/static-pods-in-kubernetes/

>>> Pods estáticos

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

[2] https://anote.dev/static-pods-in-kubernetes/

>>> Pods estáticos

* Gerenciados diretamente pelo *kubelet* do nó (indepentente 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

[2] https://anote.dev/static-pods-in-kubernetes/

>>> Pods estáticos

* Gerenciados diretamente pelo *kubelet* do nó (indepentente 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...)*

[2] https://anote.dev/static-pods-in-kubernetes/

>>> Pods estáticos

* Gerenciados diretamente pelo *kubelet* do nó (indepentente 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?

[2] https://anote.dev/static-pods-in-kubernetes/

>>> Pods estáticos

* Gerenciados diretamente pelo *kubelet* do nó (indepentente 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 plane[2]

[2] 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 tráfico 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 tráfico de rede.
  * Pode ser usado para controler 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 tráfico de rede.
  - Pode ser usado para controler os pods que servem como *backend*
  - Se o pod não estiver pronto, ele é removido do *Load Balancer* do serviço

StartupProbe:

* É usado para sabe se o container da aplicação começou.

StartupProbe:

* É usado para sabe se o container da aplicação começou.
    * Pode ser configurado para desativar *liveness* e *readness* até o *startup* tenha sucesso.

StartupProbe:

* É usado para sabe se o container da aplicação começou.
  * Pode ser configurado para desativar *liveness* e *readness* 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.

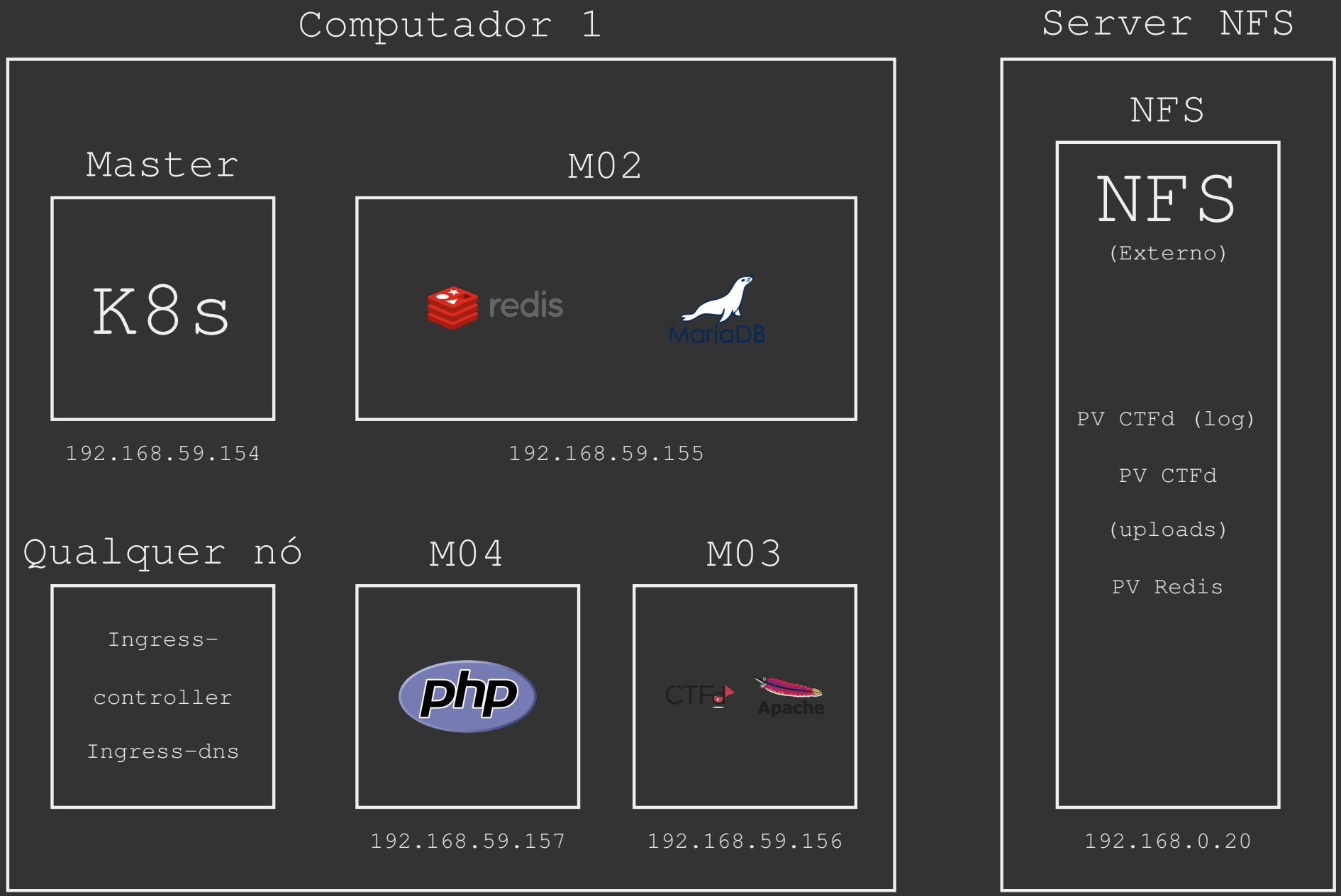
>>> Criando o cluster com minikube

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
Master
K8s
192.168.59.154
M02
redis
MariaDB
192.168.59.155
Qualquer nó
Ingress-
controller
Ingress-dns
M04
php
192.168.59.157
M03
CTFd
Apache
192.168.59.156
Server NFS
NFS
NFS
(Externo)
PV CTFd (log)
PV CTFd
(uploads)
PV Redis
192.168.0.20

>>> Recursos - Namespaces

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

Recursos (conceito e prática: comandos e aplicação de yaml) Namespaces Deployments Daemonsets - conceito Configmaps Secrets Services Ingress Persistência de dados (Storageclass - SC, Persistent Volume Claim - PVC, Persistent Volume - PV) Job e Cronjobs - conceito Statefulsets - conceito HPA - conceito RBAC (roles, service accounts, users) - conceito CRD - ~~conceito~~

[3] Kubernetes 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.[3]

Recursos (conceito e prática: comandos e aplicação de yaml) Namespaces Deployments Daemonsets - conceito Configmaps Secrets Services Ingress Persistência de dados (Storageclass - SC, Persistent Volume Claim - PVC, Persistent Volume - PV) Job e Cronjobs - conceito Statefulsets - conceito HPA - conceito RBAC (roles, service accounts, users) - conceito CRD - ~~conceito~~

[3] Kubernetes 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.[3]
  * Os administradores do cluster devem dividir os recursos

Recursos (conceito e prática: comandos e aplicação de yaml) Namespaces Deployments Daemonsets - conceito Configmaps Secrets Services Ingress Persistência de dados (Storageclass - SC, Persistent Volume Claim - PVC, Persistent Volume - PV) Job e Cronjobs - conceito Statefulsets - conceito HPA - conceito RBAC (roles, service accounts, users) - conceito CRD - ~~conceito~~

[3] Kubernetes 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.[3]
  * Os administradores do cluster devem dividir os recursos
* O nome dos recursos precisam ser únicos em cada namespace

Recursos (conceito e prática: comandos e aplicação de yaml) Namespaces Deployments Daemonsets - conceito Configmaps Secrets Services Ingress Persistência de dados (Storageclass - SC, Persistent Volume Claim - PVC, Persistent Volume - PV) Job e Cronjobs - conceito Statefulsets - conceito HPA - conceito RBAC (roles, service accounts, users) - conceito CRD - ~~conceito~~

[3] Kubernetes 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.[3]
  - 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*

Recursos (conceito e prática: comandos e aplicação de yaml) Namespaces Deployments Daemonsets - conceito Configmaps Secrets Services Ingress Persistência de dados (Storageclass - SC, Persistent Volume Claim - PVC, Persistent Volume - PV) Job e Cronjobs - conceito Statefulsets - conceito HPA - conceito RBAC (roles, service accounts, users) - conceito CRD - ~~conceito~~

[3] Kubernetes Documentation

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

* Aplicação no projeto prático:

Variáveis de ambiente

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: ctfd-env-configmap
  namespace: ctfd
data:
  DB_FILE: '/etc/ctfd-db/database'
  DB_USER_FILE: '/etc/ctfd-db/username'
  DB_PASS_FILE: '/etc/ctfd-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**

[4] 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...

[4] 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...
* Dessa forma não é preciso incluir dados confidenciais no seu código

[4] 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...
* Dessa forma não é preciso incluir dados confidenciais no seu código
* Algumas boas práticas para manter o *secrets* seguro:[4]
  * Encryption at Rest for Secrets

[4] 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...
* Dessa forma não é preciso incluir dados confidenciais no seu código
* Algumas boas práticas para manter o *secrets* seguro:[4]
  * Encryption at Rest for Secrets
  * Enable or configure RBAC rules with least-privilege access to Secrets

[4] 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...
* Dessa forma não é preciso incluir dados confidenciais no seu código
* Algumas boas práticas para manter o *secrets* seguro:[4]
  * Encryption at Rest for Secrets
  * Enable or configure RBAC rules with least-privilege access to Secrets
  * Restrict Secret access to specific containers

[4] 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...
* Dessa forma não é preciso incluir dados confidenciais no seu código
* Algumas boas práticas para manter o *secrets* seguro:[4]
  * 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

---

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

* 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=='
```

Secrets

>>> Recursos - Ingress

Secrets

>>> 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
  * Suspender/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
  * Suspender/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
  * Suspender/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
  * Suspender/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
  * Suspender/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
  * Suspender/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...

Secrets

>>> Recursos - HPA

Horizontal Pod Autoscaling In Kubernetes, a HorizontalPodAutoscaler automatically updates a workload resource (such as a Deployment or StatefulSet), with the aim of automatically scaling the workload to match demand.

Horizontal scaling means that the response to increased load is to deploy more Pods. This is different from vertical scaling, which for Kubernetes would mean assigning more resources (for example: memory or CPU) to the Pods that are already running for the workload.

If the load decreases, and the number of Pods is above the configured minimum, the HorizontalPodAutoscaler instructs the workload resource (the Deployment, StatefulSet, or other similar resource) to scale back down.

Horizontal pod autoscaling does not apply to objects that can’t be scaled (for example: a DaemonSet.)

The HorizontalPodAutoscaler is implemented as a

Daemonsets - conceito

Daemonsets - conceito

>>> Describe

É possível ver mais inform

```
  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 ava
ilable: 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 歩く 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 *initContiner*

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

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 infomações adicionais

>>> events

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

```
tsuyoshi 💻 opus 📁 final 書く k get events
LAST SEEN   TYPE      REASON                   OBJECT                                          MESSAGE
11m         Warning   FailedGetScale           horizontalpodautoscaler/ctfd-2021-chall-xor-otp-hpa   deployments/scale.apps "ctfd-2021-chall-xor-otp-deployment" not found
4m16s       Warning   FailedGetScale           horizontalpodautoscaler/ctfd-2021-chall-xor-otp-hpa   deployments/scale.apps "ctfd-2021-chall-xor-otp-deployment" not found
4s          Warning   FailedGetScale           horizontalpodautoscaler/ctfd-2021-chall-xor-otp-hpa   deployments/scale.apps "ctfd-2021-chall-xor-otp-deployment" not found
11m         Warning   FailedGetScale           horizontalpodautoscaler/ctfd-2022-chall-uccdi-hpa    deployments/scale.apps "ctfd-2022-chall-uccdi-deployment" not found
4m18s       Warning   FailedGetScale           horizontalpodautoscaler/ctfd-2022-chall-uccdi-hpa    deployments/scale.apps "ctfd-2022-chall-uccdi-deployment" not found
4s          Warning   FailedGetScale           horizontalpodautoscaler/ctfd-2022-chall-uccdi-hpa    deployments/scale.apps "ctfd-2022-chall-uccdi-deployment" not found
13m         Warning   FailedGetScale           horizontalpodautoscaler/ctfd-site-hpa               deployments/scale.apps "ctfd-site-deployment" not found
4m57s       Warning   FailedGetScale           horizontalpodautoscaler/ctfd-site-hpa               deployments/scale.apps "ctfd-site-deployment" not found
14s         Warning   FailedGetScale           horizontalpodautoscaler/ctfd-site-hpa               deployments/scale.apps "ctfd-site-deployment" not found
37m         Normal    NodeHasSufficientMemory  node/multinode-m02                                  Node multinode-m02 status is now: NodeHasSufficientMemory
37m         Normal    NodeHasNoDiskPressure    node/multinode-m02                                  Node multinode-m02 status is now: NodeHasNoDiskPressure
37m         Normal    RegisteredNode           node/multinode-m02                                  Node multinode-m02 event: Registered Node multinode-m02 in Controller
37m         Normal    Starting                 node/multinode-m02
35m         Normal    NodeHasSufficientMemory  node/multinode-m03                                  Node multinode-m03 status is now: NodeHasSufficientMemory
35m         Normal    NodeHasNoDiskPressure    node/multinode-m03                                  Node multinode-m03 status is now: NodeHasNoDiskPressure
35m         Normal    RegisteredNode           node/multinode-m03                                  Node multinode-m03 event: Registered Node multinode-m03 in Controller
35m         Normal    Starting                 node/multinode-m03
34m         Normal    NodeHasSufficientMemory  node/multinode-m04                                  Node multinode-m04 status is now: NodeHasSufficientMemory
34m         Normal    NodeHasNoDiskPressure    node/multinode-m04                                  Node multinode-m04 status is now: NodeHasNoDiskPressure
34m         Normal    RegisteredNode           node/multinode-m04                                  Node multinode-m04 event: Registered Node multinode-m04 in Controller
34m         Normal    Starting                 node/multinode-m04
38m         Normal    NodeHasSufficientMemory  node/multinode                                      Node multinode status is now: NodeHasSufficientMemory
38m         Normal    NodeHasNoDiskPressure    node/multinode                                      Node multinode status is now: NodeHasNoDiskPressure
38m         Normal    NodeHasSufficientPID     node/multinode                                      Node multinode status is now: NodeHasSufficientPID
38m         Normal    Starting                 node/multinode                                      Starting kubelet.
```

Figure: Eventos

>>> Considerações

Ambientes produtivos: setup de alta disponibilidade, scaling, quais cuidados tomar, etc

>>> Disponibilidade e Escalabilidade

Para criar um *cluster* de alta disponibilidade:

* Separar *control plane* do nó dos *workers*
* Replicar os componentes do *control plane* em multiplos nós
* Balancear o tráfico 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*

>>> Segurança e controle de acesso

Security and access management: You have full admin privileges on your own Kubernetes learning cluster. But shared clusters with important workloads, and more than one or two users, require a more refined approach to who and what can access cluster resources. You can use role-based access control (RBAC) and other security mechanisms to make sure that users and workloads can get access to the resources they need, while keeping workloads, and the cluster itself, secure. You can set limits on the resources that users and workloads can access by managing policies and container resources.

Production cluster setup In a production-quality Kubernetes cluster, the control plane manages the cluster from services that can be spread across multiple computers in different ways. Each worker node, however, represents a single entity that is configured to run Kubernetes pods.
Production control plane The simplest Kubernetes cluster has the entire control plane and worker node services running on the same machine. You can grow that environment by adding worker nodes, as reflected in the diagram illustrated in Kubernetes Components. If the cluster is meant to be available for a short period of time, or can be discarded if something goes seriously wrong, this might meet your needs.
If you need a more permanent, highly available cluster, however, you should consider ways of extending the control plane. By design, one-machine control plane services running on a single machine

are not highly available. If keeping the cluster up and running and ensuring that it can be repaired if something goes wrong is important, consider these steps:
Choose deployment tools: You can deploy a control plane using tools such as kubeadm, kops, and kubespray. See Installing Kubernetes with deployment tools to learn tips for production-quality deployments using each of those deployment methods. Different Container Runtimes are available to use with your deployments. Manage certificates: Secure communications between control plane services are implemented using certificates. Certificates are automatically generated during deployment or you can generate them using your own certificate authority. See PKI certificates and requirements for details. Configure load balancer for apiserver: Configure a load balancer to distribute external API requests to

the apiserver service instances running on different nodes. See Create an External Load Balancer for details. Separate and backup etcd service: The etcd services can either run on the same machines as other control plane services or run on separate machines, for extra security and availability. Because etcd stores cluster configuration data, backing up the etcd database should be done regularly to ensure that you can repair that database if needed. See the etcd FAQ for details on configuring and using etcd. See Operating etcd clusters for Kubernetes and Set up a High Availability etcd cluster with kubeadm for details. Create multiple control plane systems: For high availability, the control plane should not be limited to a single machine. If the control plane services are run by an init service (such as systemd), each service should run on at least three machines. However, running control plane services as

pods in Kubernetes ensures that the replicated number of services that you request will always be available. The scheduler should be fault tolerant, but not highly available. Some deployment tools set up Raft consensus algorithm to do leader election of Kubernetes services. If the primary goes away, another service elects itself and take over. Span multiple zones: If keeping your cluster available at all times is critical, consider creating a cluster that runs across multiple data centers, referred to as zones in cloud environments. Groups of zones are referred to as regions. By spreading a cluster across multiple zones in the same region, it can improve the chances that your cluster will continue to function even if one zone becomes unavailable. See Running in multiple zones for details. Manage on-going features: If you plan to keep your cluster over time, there are tasks you need to do to maintain

its health and security. For example, if you installed with kubeadm, there are instructions to help you with Certificate Management and Upgrading kubeadm clusters. See Administer a Cluster for a longer list of Kubernetes administrative tasks. To learn about available options when you run control plane services, see kube-apiserver, kube-controller-manager, and kube-scheduler component pages. For highly available control plane examples, see Options for Highly Available topology, Creating Highly Available clusters with kubeadm, and Operating etcd clusters for Kubernetes. See Backing up an etcd cluster for information on making an etcd backup plan.
Production worker nodes Production-quality workloads need to be resilient and anything they rely on needs to be resilient (such as CoreDNS). Whether you manage your own control plane or have a cloud provider do it

for you, you still need to consider how you want to manage your worker nodes (also referred to simply as nodes).
Configure nodes: Nodes can be physical or virtual machines. If you want to create and manage your own nodes, you can install a supported operating system, then add and run the appropriate Node services. Consider: The demands of your workloads when you set up nodes by having appropriate memory, CPU, and disk speed and storage capacity available. Whether generic computer systems will do or you have workloads that need GPU processors, Windows nodes, or VM isolation. Validate nodes: See Valid node setup for information on how to ensure that a node meets the requirements to join a Kubernetes cluster. Add nodes to the cluster: If you are managing your own cluster you can add nodes by setting up your own machines and either adding them manually or having

them register themselves to the cluster's apiserver. See the Nodes section for information on how to set up Kubernetes to add nodes in these ways. Scale nodes: Have a plan for expanding the capacity your cluster will eventually need. See Considerations for large clusters to help determine how many nodes you need, based on the number of pods and containers you need to run. If you are managing nodes yourself, this can mean purchasing and installing your own physical equipment. Autoscale nodes: Most cloud providers support Cluster Autoscaler to replace unhealthy nodes or grow and shrink the number of nodes as demand requires. See the Frequently Asked Questions for how the autoscaler works and Deployment for how it is implemented by different cloud providers. For on-premises, there are some virtualization platforms that can be scripted to spin up new nodes based on demand. Set up node health

checks: For important workloads, you want to make sure that the nodes and pods running on those nodes are healthy. Using the Node Problem Detector daemon, you can ensure your nodes are healthy. Production user management In production, you may be moving from a model where you or a small group of people are accessing the cluster to where there may potentially be dozens or hundreds of people. In a learning environment or platform prototype, you might have a single administrative account for everything you do. In production, you will want more accounts with different levels of access to different namespaces. Taking on a production-quality cluster means deciding how you want to selectively allow access by other users. In particular, you need to select strategies for validating the identities of those who try to access your cluster (authentication) and deciding if they have permissions to do what they are asking

(authorization):
Authentication: The apiserver can authenticate users using client certificates, bearer tokens, an authenticating proxy, or HTTP basic auth. You can choose which authentication methods you want to use. Using plugins, the apiserver can leverage your organization's existing authentication methods, such as LDAP or Kerberos. See Authentication for a description of these different methods of authenticating Kubernetes users. Authorization: When you set out to authorize your regular users, you will probably choose between RBAC and ABAC authorization. See Authorization Overview to review different modes for authorizing user accounts (as well as service account access to your cluster):
Role-based access control (RBAC): Lets you assign access to your cluster by allowing specific sets of permissions to authenticated users. Permissions can

be assigned for a specific namespace (Role) or across the entire cluster (ClusterRole). Then using RoleBindings and ClusterRoleBindings, those permissions can be attached to particular users.
Attribute-based access control (ABAC): Lets you create policies based on resource attributes in the cluster and will allow or deny access based on those attributes. Each line of a policy file identifies versioning properties (apiVersion and kind) and a map of spec properties to match the subject (user or group), resource property, non-resource property (/version or /apis), and readonly. See Examples for details. As someone setting up authentication and authorization on your production Kubernetes cluster, here are some things to consider:
Set the authorization mode: When the Kubernetes API server (kube-apiserver) starts, the supported authentication modes must be set using the

-authorization-mode flag. For example, that flag in the kube-adminserver.yaml file (in /etc/kubernetes/manifests) could be set to Node,RBAC. This would allow Node and RBAC authorization for authenticated requests. Create user certificates and role bindings (RBAC): If you are using RBAC authorization, users can create a CertificateSigningRequest (CSR) that can be signed by the cluster CA. Then you can bind Roles and ClusterRoles to each user. See Certificate Signing Requests for details. Create policies that combine attributes (ABAC): If you are using ABAC authorization, you can assign combinations of attributes to form policies to authorize selected users or groups to access particular resources (such as a pod), namespace, or apiGroup. For more information, see Examples. Consider Admission Controllers: Additional forms of authorization for

requests that can come in through the API server include Webhook Token Authentication. Webhooks and other special authorization types need to be enabled by adding Admission Controllers to the API server.
Set limits on workload resources Demands from production workloads can cause pressure both inside and outside of the Kubernetes control plane. Consider these items when setting up for the needs of your cluster's workloads:
Set namespace limits: Set per-namespace quotas on things like memory and CPU. See Manage Memory, CPU, and API Resources for details. You can also set Hierarchical Namespaces for inheriting limits.
Prepare for DNS demand: If you expect workloads to massively scale up, your DNS service must be ready to scale up as well. See Autoscale the DNS service in a Cluster. Create additional service accounts: User accounts determine what users can do on a cluster,

while a service account defines pod access within a particular namespace. By default, a pod takes on the default service account from its namespace. See Managing Service Accounts for information on creating a new service account. For example, you might want to: Add secrets that a pod could use to pull images from a particular container registry. See Configure Service Accounts for Pods for an example. Assign RBAC permissions to a service account. See ServiceAccount permissions for details. What's next Decide if you want to build your own production Kubernetes or obtain one from available Turnkey Cloud Solutions or Kubernetes Partners. If you choose to build your own cluster, plan how you want to handle certificates and set up high availability for features such as etcd and the API server. Choose from kubeadm, kops or Kubespray deployment methods. Configure user management by determining your

Authentication and Authorization methods. Prepare for application workloads by setting up resource limits, DNS autoscaling and service accounts.

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

"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 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 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 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

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**

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
```

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).