>>> >>> Kubernetes

Name: Henrique Tsuyoshi Yara (OPUS-software)
Date: January 16, 2023

Figure: Kubernetes logo

>>> Índice

>>> Mostrar o projeto funcionando

* Mostrar o projeto funcionando
* K9s

>>> Especificação do projeto

* Plataforma de *Capture The Flag*[1]

[1] Repositório para o CTFd

>>> Especificação do projeto

* Plataforma de *Capture The Flag*[1]
* Especificações:
    * 2 *Deployments* de *MariaDB*

[1] Repositório para o CTFd

>>> Especificação do projeto

* Plataforma de *Capture The Flag*[1]
* Especificações:
  * 2 *Deployments* de *MariaDB*
  * 2 *Deployments* de *CTFd*

[1] Repositório para o CTFd

>>> Especificação do projeto

* Plataforma de *Capture The Flag*[1]
* Especificações:
  * 2 *Deployments* de *MariaDB*
  * 2 *Deployments* de *CTFd*
  * 2 *StatefulSet* de *Redis*

[1] Repositório para o CTFd

>>> Especificação do projeto

* Plataforma de *Capture The Flag*[1]
* Especificações:
  * 2 *Deployments* de *MariaDB*
  * 2 *Deployments* de *CTFd*
  * 2 *StatefulSet* de *Redis*
  * 1 *Deployment* de *Apache*

[1] Repositório para o CTFd

>>> Especificação do projeto

* Plataforma de *Capture The Flag*[1]
* Especificações:
  * 2 *Deployments* de *MariaDB*
  * 2 *Deployments* de *CTFd*
  * 2 *StatefulSet* de *Redis*
  * 1 *Deployment* de *Apache*
  * 1 *Deployment* de *Php*

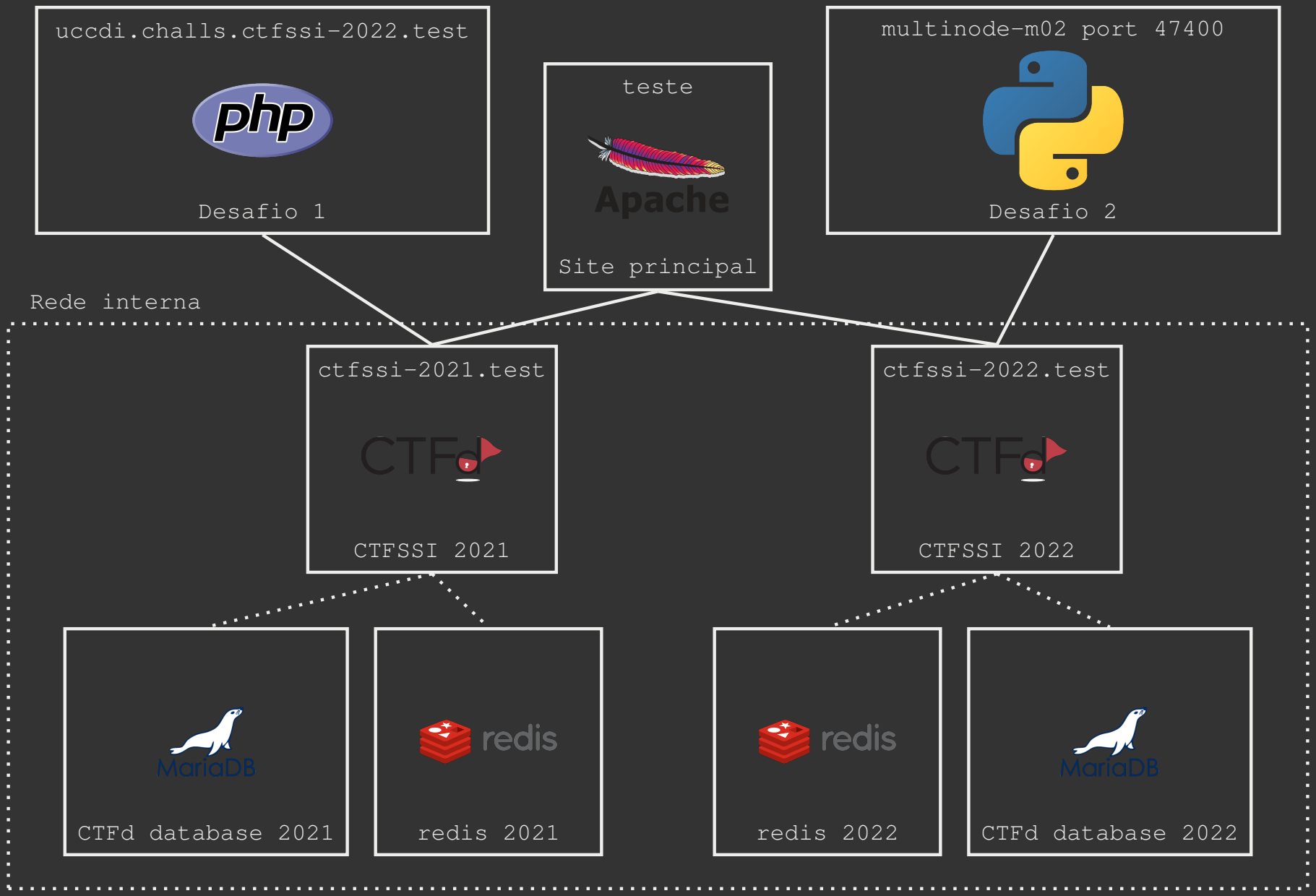
[1] Repositório para o CTFd

>>> Especificação do projeto

* Plataforma de *Capture The Flag*[1]
* Especificações:
  * 2 *Deployments* de *MariaDB*
  * 2 *Deployments* de *CTFd*
  * 2 *StatefulSet* de *Redis*
  * 1 *Deployment* de *Apache*
  * 1 *Deployment* de *Php*
  * 1 *Deployment* de *Python*

[1] Repositório para o CTFd

>>> Jornada do usuário e Rede interna



>>> 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

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

* 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

>>> 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*: Máquinas trabalhadoras que vão rodar a aplicação(*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*: Máquinas trabalhadoras que vão rodar a aplicação(*Nodes*)
  * Geralmente são usados múltiplos nós *workers* 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...)

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

>>> Componentes - *Control Plane*

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

* **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

* **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 - 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 (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

Exemplo *LivenessProbe*

```
livenessProbe:
  tcpSocket:
    port: 8080
  initialDelaySeconds: 60
  periodSeconds: 60
```

>>> *LivenessProbe, ReadinessProbe e StartUpProbe*

ReadinessProbe:

* É usado para saber quando um container está pronto para aceitar tráfico de rede.

>>> *LivenessProbe, ReadinessProbe e StartUpProbe*

ReadinessProbe:

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

>>> *LivenessProbe, ReadinessProbe e StartUpProbe*

ReadinessProbe:

- É usado para saber quando um container está pronto para aceitar tráfico 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

Exemplo *ReadinessProbe*

```
readinessProbe:
  tcpSocket:
    port: 8080
  initialDelaySeconds: 60
  periodSeconds: 60
```

>>> *LivenessProbe, ReadinessProbe e StartUpProbe*

StartupProbe:

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

>>> *LivenessProbe, ReadinessProbe e StartUpProbe*

StartupProbe:

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

>>> *LivenessProbe, ReadinessProbe e StartUpProbe*

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:

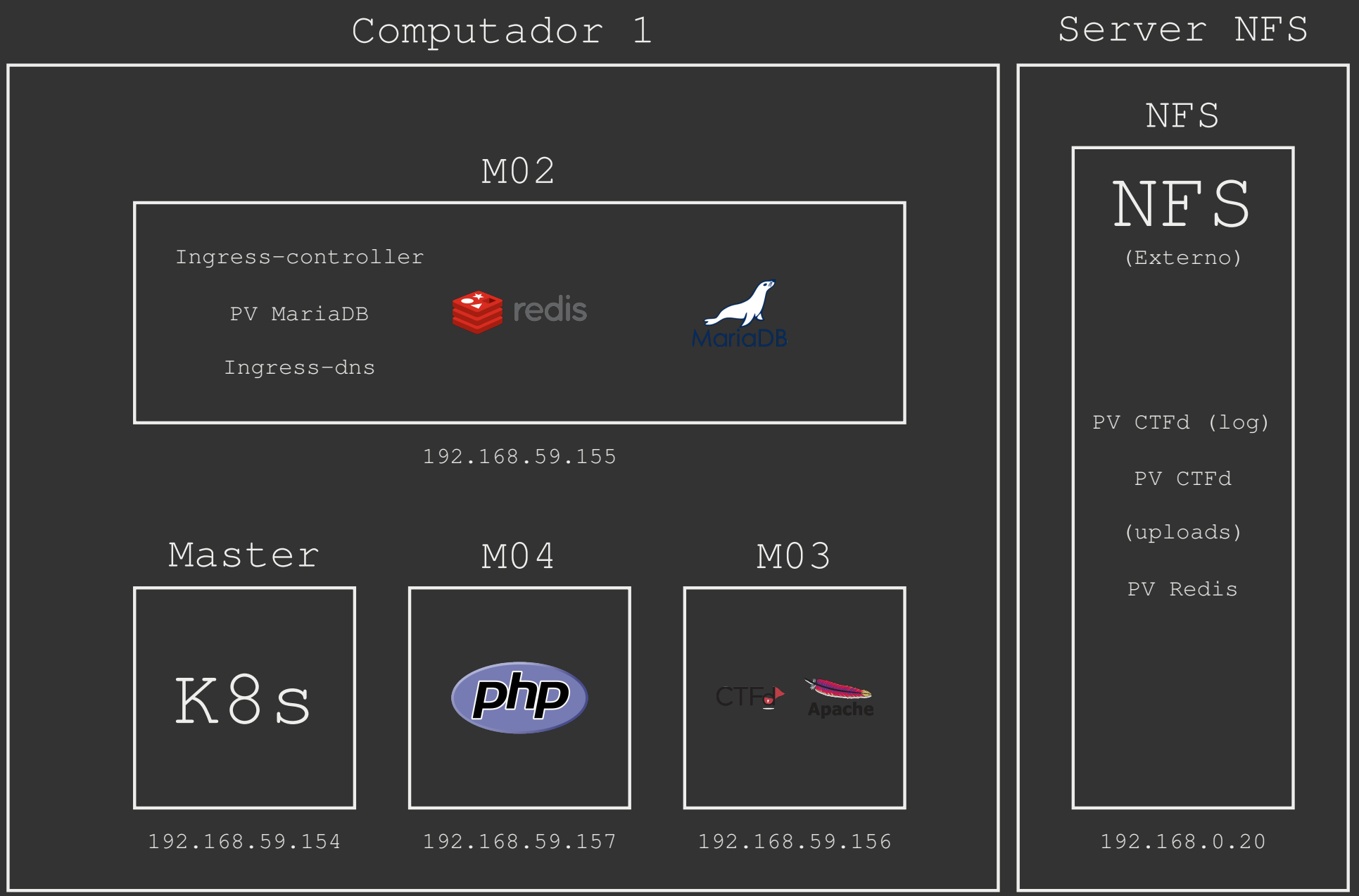
* kind (Kubernetes in Docker)
    * Cluster usando containers de docker para criar os nós.
* minikube
    * Cluster usando containeres ou máquinas virtuais.

>>> Criando o cluster com minikube

Criação do cluster utilizado na parte prática:

Comando *minikube*

```
minikube start --nodes='4' \
--profile=multinode \
--vm-driver=virtualbox
```


Computador 1
Server NFS
M02
Ingress-controller
PV MariaDB
Ingress-dns
redis
MariaDB
192.168.59.155
Master
M04
M03
K8s
php
CTFd
Apache
192.168.59.154
192.168.59.157
192.168.59.156
NFS
NFS
(Externo)
PV CTFd (log)
PV CTFd
(uploads)
PV Redis
192.168.0.20

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

[3] Kubernetes Documentation

* 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]

[3] Kubernetes Documentation

* 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

[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

[3] Kubernetes Documentation

- 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*

[3] Kubernetes Documentation

Para ver os recursos que são e não são isolados 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)
ctfd-2021,2-service
ctfd-2021,2-deployment

redis

redis-202{1,2}-statefulset
redis-202{1,2}-service

ingress-nginx

ingress-nginx-controller

site

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

database

secrets (mariaDB)
db-202{1,2}-deployment
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=='
```

* Os **Services** conseguem abstrair a parte da rede dos **Pods**

>>> Recursos - Services

* Os **Services** conseguem abstrair a parte da rede dos **Pods**
* Os **Services** cuidam da parte do balanceamento de carga e **Service Discovery**

>>> Recursos - ClusterIP

* Expõe o serviço para o *cluster* interno
* O serviço só pode ser alcançado dentro do cluster

ClusterIP do Apache

```
apiVersion: v1
kind: Service
metadata:
  name: ctfd-site-service
  namespace: site
spec:
  type: ClusterIP
  ports:
  - name: 'http'
    port: 80
    targetPort: 80
  selector:
    ctfd.site.pod: ctfd-site-pod
```

>>> Recursos - NodePort

* Expõe o serviço em todos os Nós em uma porta estática

>>> Recursos - NodePort

* Expõe o serviço em todos os Nós em uma porta estática
* **node port** o K8s aloca uma porta com o protocolo do serviço (TCP, UDP, SCTP, etc...)

>>> Recursos - NodePort

* Expõe o serviço em todos os Nós em uma porta estática
* **node port** o K8s aloca uma porta com o protocolo do serviço (TCP, UDP, SCTP, etc...)
* Todo os nós começam a ouvir na porta especificada e redirecionar o tráfico para o *endpoint* do serviço

>>> Recursos - LoadBalancer

* Expõe o serviço usando o provedor de nuvem especificado

>>> Recursos - LoadBalancer

* Expõe o serviço usando o provedor de nuvem especificado
* O tráfico de rede externo do *Load Balancer* é redirecionado para os **Pods.** A distribuição dos requests é decidida pelo provedor

>>> Recursos - ExternalName

* Mapeia o serviço para um nome **DNS**

ExternalName uccdi

```
apiVersion: v1
kind: Service
metadata:
  name: ctfd-2022-chall-uccdi
  namespace: challs
spec:
  type: ExternalName
  externalName: ctfd-2022-chall-uccdi-deployment.default.svc.cluster.local
```

* **Ingress:** Gerencia acesso externo para os serviços dentro do cluster (Geralmente o protocolo **HTTP**)

[5] Outros exemplos de Ingress-Controllers

* **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

[5] Outros 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

[5] Outros 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**

[5] Outros 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

[5] Outros 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 nginx[5]

[5] Outros exemplos de Ingress-Controllers

* **Persistent Volume (PV):** é um armazenameto do cluster que foi disponibilizado pelo administrador ou dinamicamente disponibilizado usando **Storage Class**

* **Persistent Volume (PV):** é um armazenameto do cluster que foi disponibilizado pelo administrador ou dinamicamente disponibilizado usando **Storage Class**
  * É um recurso assim como um **Node**

* **Persistent Volume (PV):** é um armazenameto do cluster que foi disponibilizado pelo administrador ou dinamicamente disponibilizado usando **Storage Class**
  * É um recurso assim como um **Node**
  * Tem um ciclo de vida independente de um **Pod** que usa o **PV**

* **Persistent Volume (PV):** é um armazenameto do cluster que foi disponibilizado pelo administrador ou dinamicamente disponibilizado usando **Storage Class**
  * É um recurso assim como um **Node**
  * Tem um ciclo de vida independente de um **Pod** que usa o **PV**
  * Esse objeto pega os detalhes de implementação do armazenamento disponibilizado. (Podendo ser NFS, iSCSI, cloud, etc...)

* **Persistent Volume (PV):** é um armazenameto do cluster que foi disponibilizado pelo administrador ou dinamicamente disponibilizado usando **Storage Class**
  * É um recurso assim como um **Node**
  * Tem um ciclo de vida independente de um **Pod** que usa o **PV**
  * Esse objeto pega os detalhes de implementação do armazenamento disponibilizado. (Podendo ser NFS, iSCSI, cloud, etc...)
* **Persistent Volume Claim:** Permite um usuário consumir armazenamentos

>>> Recursos - Persistência de dados

* **Persistent Volume (PV):** é um armazenameto do cluster que foi disponibilizado pelo administrador ou dinamicamente disponibilizado usando **Storage Class**
  * É um recurso assim como um **Node**
  * Tem um ciclo de vida independente de um **Pod** que usa o **PV**
  * Esse objeto pega os detalhes de implementação do armazenamento disponibilizado. (Podendo ser NFS, iSCSI, cloud, etc...)
* **Persistent Volume Claim:** Permite um usuário consumir armazenamentos
  * É comum o usuário precisar de **PVs** com várias propriedades, tamanhos e acessos

Provisionamento:

* **Estático**: O administrador cria um número de **PVs** e vão ficar disponíveis para consumo

Provisionamento:

* **Estático**: O administrador cria um número de **PVs** e vão ficar disponíveis para consumo
* **Dinâmico**: Quando nenhum **PV** estático supre as necessidades do **PVC**, o cluster dinâmicamente provê um volume especialmente para o **PVC**

Provisionamento:

* **Estático**: O administrador cria um número de **PVs** e vão ficar disponíveis para consumo
* **Dinâmico**: Quando nenhum **PV** estático supre as necessidades do **PVC**, o cluster dinâmicamente provê um volume especialmente para o **PVC**
  * O **PVC** precisa pedir por uma **Storage Class** que foi configurada para a alocação dinâmica acontecer

* Binding:
  * O **PVC** só vai juntar com um **PV** se o **PV** foi provisionado dinâmicamente.

>>> Recursos - Lifecycle

* Binding:
  * O **PVC** só vai juntar com um **PV** se o **PV** foi provisionado dinâmicamente.
  * Caso não seja provisionado dinâmicamente vai pegar o **PV** com pelo menos a capacidade requerida pelo **PVC**

>>> Recursos - Lifecycle

* Binding:
  * O **PVC** só vai juntar com um **PV** se o **PV** foi provisionado dinâmicamente.
  * Caso não seja provisionado dinâmicamente vai pegar o **PV** com pelo menos a capacidade requerida pelo **PVC**
  * Um **PVC** só vai se juntar com um **PV** e vice versa

>>> Recursos - Lifecycle

* Binding:
  * O **PVC** só vai juntar com um **PV** se o **PV** foi provisionado dinâmicamente.
  * Caso não seja provisionado dinâmicamente vai pegar o **PV** com pelo menos a capacidade requerida pelo **PVC**
  * Um **PVC** só vai se juntar com um **PV** e vice versa
* Using:

>>> Recursos - Lifecycle

* Binding:
  * O **PVC** só vai juntar com um **PV** se o **PV** foi provisionado dinâmicamente.
  * Caso não seja provisionado dinâmicamente vai pegar o **PV** com pelo menos a capacidade requerida pelo **PVC**
  * Um **PVC** só vai se juntar com um **PV** e vice versa
* Using:
  * Então os **Pods** usam os **Claims** como volumes

>>> Recursos - Lifecycle

* Binding:
  * O **PVC** só vai juntar com um **PV** se o **PV** foi provisionado dinâmicamente.
  * Caso não seja provisionado dinâmicamente vai pegar o **PV** com pelo menos a capacidade requerida pelo **PVC**
  * Um **PVC** só vai se juntar com um **PV** e vice versa
* Using:
  * Então os **Pods** usam os **Claims** como volumes
  * O **PV** então perntece ao usuário até ele parar de usar

* Storage Object in Use Protection

* Storage Object in Use Protection
  * Ele garante que **PVC** ativos em um **Pod** e um **PV** não podem ser removido do sistema

* Storage Object in Use Protection
  * Ele garante que **PVC** ativos em um **Pod** e um **PV** não podem ser removido do sistema
  * Só vai ser removido quando nenhum **Pod** usar o **PVC**

* Storage Object in Use Protection
  * Ele garante que **PVC** ativos em um **Pod** e um **PV** não podem ser removido do sistema
  * Só vai ser removido quando nenhum **Pod** usar o **PVC**
  * Um **PV** só vai ser deletado quando não estiver junto com nenhum **PVC**

>>> Recursos - Lifecycle

* Reclaiming:
    *

>>> Recursos - Lifecycle

* Reclaiming:
    * Retain

        *

>>> Recursos - Lifecycle

* Reclaiming:
    * Retain
        * Depois de um **PVC** for deletado, o **PV** mantém seus dados e fica indisponível para outros **PVC**
        *

>>> Recursos - Lifecycle

* Reclaiming:
    * Retain
        * Depois de um **PVC** for deletado, o **PV** mantém seus dados e fica indisponível para outros **PVC**
        * O administrador precisa alocar ele manualmente
        *

>>> Recursos - Lifecycle

* Reclaiming:
    * Retain
        * Depois de um **PVC** for deletado, o **PV** mantém seus dados e fica indisponível para outros **PVC**
        * O administrador precisa alocar ele manualmente
    * Delete
        *

* Reclaiming:
    * Retain
        * Depois de um **PVC** for deletado, o **PV** mantém seus dados e fica indisponível para outros **PVC**
        * O administrador precisa alocar ele manualmente
    * Delete
        * Remove o **PV** e o armazenamento externo (AWS EBS, GCE PD, etc...)
        *

* Reclaiming:
    * Retain
        * Depois de um **PVC** for deletado, o **PV** mantém seus dados e fica indisponível para outros **PVC**
        * O administrador precisa alocar ele manualmente
    * Delete
        * Remove o **PV** e o armazenamento externo (AWS EBS, GCE PD, etc...)
        *

>>> 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...

>>> Recursos - StatefulSet

* O **StatefulSet** é um objeto da *API* que é usado para gerenciar aplicações com estados

>>> Recursos - StatefulSet

* O **StatefulSet** é um objeto da *API* que é usado para gerenciar aplicações com estados
* Gerencia e escala os **Pods** de forma previsível (Nomes únicos e com ordenados)

>>> Recursos - StatefulSet

* O **StatefulSet** é um objeto da *API* que é usado para gerenciar aplicações com estados
* Gerencia e escala os **Pods** de forma previsível (Nomes únicos e com ordenados)
* Diferente do **Deployment** o **StatefulSet** mantém um indentificador para cada **Pod**

>>> Recursos - StatefulSet

* O **StatefulSet** é um objeto da *API* que é usado para gerenciar aplicações com estados
* Gerencia e escala os **Pods** de forma previsível (Nomes únicos e com ordenados)
* Diferente do **Deployment** o **StatefulSet** mantém um indentificador para cada **Pod**
* Os **Pods** são criados da mesma especificação, mas não podem ser trocados

>>> Recursos - StatefulSet

* O **StatefulSet** é um objeto da *API* que é usado para gerenciar aplicações com estados
* Gerencia e escala os **Pods** de forma previsível (Nomes únicos e com ordenados)
* Diferente do **Deployment** o **StatefulSet** mantém um indentificador para cada **Pod**
* Os **Pods** são criados da mesma especificação, mas não podem ser trocados
* Os indentificadores dos **Pods** facilitam na hora de relacionar um **Pod** com seu **Volume**

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

>>> 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

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)

* **Role-based access control (RBAC)** é um método de regular acesso para o computador ou recursos na rede baseado nos seus papéis

* **Role-based access control (RBAC)** é um método de regular acesso para o computador ou recursos na rede baseado nos seus papéis
* O **RBAC** usa a *API* **rbac.authorization.k8s.io** para autorizar as decisões

* **Role-based access control (RBAC)** é um método de regular acesso para o computador ou recursos na rede baseado nos seus papéis
* O **RBAC** usa a *API* **rbac.authorization.k8s.io** para autorizar as decisões
* É possível configurar as políticas dinamicamentes pela API do kuberntes

>>> Recursos - RBAC

* RBAC Role

* **RBAC Role**
    * Permissões dentro de um **namespace**

* **RBAC Role**
    * Permissões dentro de um **namespace**
* **RBAC Cluster Role**
    * Permissões que se aplicam em todos os **namespaces**

* **RBAC Role**
  * Permissões dentro de um **namespace**
* **RBAC Cluster Role**
  * Permissões que se aplicam em todos os **namespaces**
* As regras do **RBAC Role** e do **RBAC ClusterRole** são sempre aditivas

* **Custom Resources** são extensões da *API* do kubernetes
* **Custom Resources Definition** é uma forma simples de criar um **Custom Resources**

Calico exemplo

```
apiVersion: projectcalico.org/v3
kind: GlobalNetworkPolicy
metadata:
  name: default-deny
spec:
  selector: projectcalico.org/namespace != "kube-system"
  types:
  - Ingress
  - Egress
```

>>> 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 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...:*

```
⚠tsuyoshi  opus  final 歩く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_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 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

>>> 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*

>>> *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

>>> *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
* **Autorização:** **RBAC** ou **ABAC**

>>> *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
* **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
  * É possível usar **Kerberos** ou **LDAP** para uma autenticação mais avançada
* **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
  * É possível usar **Kerberos** ou **LDAP** para uma autenticação mais avançada
* **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
  * É possível usar **Kerberos** ou **LDAP** para uma autenticação mais avançada
* **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 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

"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

"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).