Kubernetes - Material de Apoio, Matheus Luis Oliveira da Silva

▼ Definição do projeto

Tópicos de Estudo:

1. Kubernetes (conceito, pré-requisitos, componentes)
2. Kubernetes vs Docker (diferenças, em que casos é melhor utilizá-los)
3. Pods vs Container
4. Criação local do cluster kubernetes (exemplo: kind, minikube, etc) e uso do kubectl
5. Recursos (conceito e prática: comandos e aplicação de yaml)
    - Namespaces
    - Deployments
    - Configmaps
    - Secrets
    - Services
    - ReplicaSet
    - Ingress
    - Persistência de dados (Storageclass - SC, Persistent Volume Claim - PVC, Persistent Volume - PV)
    - Daemonsets - conceito
    - Job e Cronjobs - conceito
    - Statefulsets - conceito
    - HPA - conceito
    - RBAC (roles, service accounts, users) - conceito

- CRD - conceito

6. Troubleshooting (describe, logs, events e exec)
7. Ambientes produtivos: setup de alta disponibilidade, scaling, quais cuidados tomar, etc
8. Uso do Helm para a instalação de pacotes

Entregáveis (prazo máximo estipulado: 2 meses):

Deploy de uma solução que utiliza no mínimo um frontend (com ou sem backend) e um banco de dados, utilizando todos os recursos do Kubernetes estudados (exceto itens "*conceito"*), com arquivos no formato "yaml", com alguns requisitos:

Os dados, quando utilizados pelos pods, precisam ser persistentes (usar SC, PVC e PV)

Os valores de configmaps precisam ser consumidos pelos pods como variáveis de ambientes

Os valores de secrets precisam ser consumidos pelos pods através arquivos em volumes

Deploy de uma aplicação qualquer via Helm passando parâmetros. A aplicação dos parâmetros deverá ser evidenciada dentro do kubernetes. Utilizar uma aplicação existente.

▼ Referências:

- https://www.udemy.com/course/certified-kubernetes-application-developer/
- https://kubernetes.io/pt-br/
-
- https://www.youtube.com/watch?v=01qcYSck1c4
- https://github.com/kubernetes-sigs/nfs-subdir-external-provisioner/blob/master/deploy/class.yaml
- https://cursos.alura.com.br/loginForm?urlAfterLogin=https://cursos.alura.com.br/course/kubernetes-deployments-volumes-escalabilidade/task/80511
- https://www.youtube.com/watch?v=RGSeeN-o-kQ

▼ Kubernetes

▼ Características:

- Desenvolvida pela google
- Orquestrador de containers OpenSource
- Gerencia aplicações containerizadas em diferentes ambientes de deploy
- Resolve o problema da superpopulação de containers na nova política da quebra do monolito em microsserviços
- Alta avaliabilidade e baixo downtime
- Promove escalabilidade e performance
- Mecanismos de backup de grande eficiência

▼ Arquitetura:

- Cluster, composto por várias máquinas
- Temos ao menos um Master Node, virtual ou físico.
- Worker nodes:
  - As aplicações rodam nos worker nodes.
  - Possui um runtime para o container
  - kubelet: o processo kubelet é como um processo de conexão do node com todo o cluster, viabilizando toda a comunicação dentro do cluster.
  - K-Proxy: Permite a comunicação e assegura as diretrizes para comunicação entre os serviços
- Master node:
  - Fica responsável por rodar todos os principais processos do kubernetes.
  - sem acesso ao master node, todo o acesso a cluster será perdido, portanto a importância da replicação e do backup do master node;
  - Possui um runtime para o container
  - Etcd: uma unidade de armazenamento que guarda todo o estado do cluster, assim como cada um de seus nodes, e cada comportamento interno dentro desses.
  - Scheduler: redireciona processamento para cada node, baseado na disponibilidade de recursos;
  - API server: é um container que roda o processo responsável por responder requisições dos clientes do cluster kubernetes, como uma UI, uma API ou uma CLI - Kubectl
  - Controller Manager: realiza a supervisão do funcionamento do cluster, Node-controller + replication-controller;
- ETCD:
  - É uma Key-value store
  - Facilita o armazenamento de dados em pares, inovando o armazenamento em tabelas, com mais versatilidade.

▼ componentes do kubernetes:

▼ POD’s:

- A menor unidade do cluster que busca abstrair a complexidade de um container, sendo, assim, apenas uma layer superior do mesmo na qual será implementada dada aplicação.
- Uma vez que a complexidade é abstraída, não se trabalha mais diretamente com tecnologias de containerização, como docker.
- Usualmente se trabalha com apenas uma aplicação por POD, exceto quando temos serviços auxiliares que façam sentido estarem mais próximos da aplicação principal, dentro do mesmo POD.

- Cada POD recebe um IP específico efêmero, que se altera a cada reinstância do POD.
- Um container dentro de um POD receberá uma porta para acesso a ele, aberta a partir do IP do POD.

▼ Namespaces:

- São usados pelo kubernetes para organizar os recursos de um cluster
- Divide o cluster em pequenos clusters virtualizados, cada um com seu nome e seus recursos especificos
- Os namespaces padrões geralmente são:
  - kube-system: contem os processos do sistema do kubernetes e pods de alto gerenciamento do cluster
  - kube-public: possui informações publicas a respeito do cluster
  - kube-node-lease: monitora a “saúde” dos nós do cluster
  - default: namespace no qual por padrão os recursos são criados
  - kubectl api-resources —namespaced=false

▼ Services e Network:

▼ Serviços

- Um serviço não é efêmero, tendo seu IP persistido.
- Se um POD cai, o serviço permanece intacto, podendo receber e ser conectado a nova instância do POD.
- Prove DNS para um ou mais pods e suportam load balancer
- External service: abre comunicação para serviços externos
- Internal service: preserva a não acessibilidade de aplicações sensiveis

▼ Tipos:

- Cluster IP:
  - clusterIP confere ao service uma rede de comunicação interna ao cluster, acessos externos não são permitidos.
- NodePort:
  - o service NodePort permite a comunicação interna da mesma forma que o ClusterIP;
  - possui um IP para dentro do cluster nativamente.
  - Permite também comunicação externa
- LoadBalancer:
  - Utilizam os lb nativos do serviço de cloud
  - Também são clusterIP e NodPorts

▼ Network:

- Com a configuração do cluster KinD, as portas do localhost da maquina que o hospeda foram abertas para o cluster, assim, ele pode ser acessado via localhost e não pelo ip do nó específico.
- Cada nó do cluster recebe um IP interno e possui um IP externo também, que no caso é o ip do container que está rodando no runtime Docker.
  - Interno: 10.x.x.x
  - Externo: 172.x.x.x
- Por padrão o kubernetes implementa um serviço de DNS nativo.
  - Possui um deploy para rodar o serviço chamado core DNS
  - Esse, por sua vez, possui um serviço chamado kube-dns
- O padrão completo da atribuição do dns segue o seguinte modelo:

  Hostname.Namespace.Type.Root → IP Address

  frontend.default.svc.cluster.local →10.96.72.216
- Ingress:
  - Intermediador de requests que redireciona uma call vinda de um determinado dominio, para o seu respectivo serviço.
  - Necessita de um ingress controller, um POD, para operar.
  - HostNetowork: Com a hostnetwork habilitada os pods vão utilizar o namespace de rede do host, podendo ser acessados via interfaces e endereço do host
- Nip.io
  - O Nip.io é um serviço open source que redireciona qualquer requisição no formato
  - **<anything>[.-]<IP Address>.nip.io**
  - Para o IP Address correspondente

▼ Configuração de Pods:

- configMap: contém toda a configuração de urls, banco de dados, ou qualquer outro serviço, usado apenas para dados não confidenciais.
- secrets: similar ao configMap, porém para dados sensíveis, guardado em um arquivo encriptado, geralmente por outras ferramentas externas ao kubernetes.
  - Ambos podem ser acessados via variaveis de ambiente, arquivos de propriedades ou volumes.
- InitContainers: servem para pré-estabelecer alguma configuração, algum serviço, ou alguma ação antes que o container da aplicação seja executado.

▼ Criação de Pods:

- ReplicaSets:
    - Componente responsável pela responsibilidade a quedas de pods. Quando um pod cai, ele verifica o número de replicas desejado para aquele pod, e reinicia um pod.
    - O serviço, por sua vez, permanece referenciando todas as replicas, uma vez que elas continuam com o mesmo identificador
- Deployments:
    - são uma segunda camada de abstração, agora abstraindo os pods, uma vez que instancia varias replicas de pods, bem como as gerencia;
    - O desenvolvedor, então, comumente trabalhará com deployments de seus pods, e não diretamente com os pods.
    - É um conjunto que engloba os replicasets, ou seja, todo deployment é um replicaset.
    - A diferença principal é o controle de versão, semelhante ao git, referente aos pods os quais ele configura.
- StatefulSets:
    - Similar aos deployments, entretanto usado para aplicações stateFull ou banco de dados, ou seja, visa evitar a inconsistencia de dados possivelmente promovida pelas replicas, como um possível duplo acesso simultaneo ao banco, por exemplo.
    - Com um statefulset é possível definir um persistent volume claim padrão para o pod, ou seja, com a ligação em cadeia, com apenas um statefulSet, todos os persistent volumes de um container que esta dentro de um pod serão criados automaticamente.
    - Por padrão vai utilizar a storage class default do sistema

▼ Volumes:

- novamente, como em docker, é o componente usado para persistencia de dados dos pods, localmente ou em nuvem.
- Todo o gerenciamento de armazenamento de dados não tem nenhuma relação com kubernetes, ficando a cargo do desenvolvedor.
- Persistent Volume:
    - É uma declaração de atribuição de volume, a qual fica aguardando a sua ligação com um pod para aplicar suas especificações.
    - Fará, quando solicitado, uma montagem em algum disco alocado no host
    - Caso o pod venha a falhar, ele pode ser executado novamente e a montagem e o vínculo não serão perdidos.
        - HostPath:

            - um mount é dependente do pod o qual está ligado, caso o container caia ele vai persistir, porém caso o pod morra, ele é descartado
            - Entretanto, o conteúdo do volume permanece, porém o que se perde é o vínculo
            - Muitos Problemas de segurança, não é muito utilizado.
        - Local:
            - Melhor opção ao se comparar com o hostPath
            - Mais seguro e restrito.
            - No mais, comportamento semelhante ao hostpath para armazenamento local.
- Persistent Volume Claim:
    - Permite a conexão e acesso de um pod a um persistent volume
    - Caso o pv ja tenha sido criado, a conexão ocorre via matching dos componentes do arquivo yaml
    - Caso nenhum pv tenha sido criado, o pvc cria automaticamente um pv para o pod
- StorageClass:
    - Automatiza a criação de persistentVolumes sob a demanda de persistent volume claims que sejam ligados a ela.
    - Também cria os discos dos pvolumes automaticamente.
    - Para cada provisioner, o driver tem um comportamento específico, que, a priori não permite encontrar a montagem física do volume.
    - A storage class padrão provisiona a montagem dos pvs automaticamente, caso a padrão não seja utilizada, tudo deve ser configurado manualmente;
    - Storage local NÃO GARANTE um funcionamento padrão das funções de armazenamento do kubernetes, gerando comportamentos muito inconsistentes, então para evitar isso, procura-se um proviosionamento externo de armazenamento
    - provisionamento externo de armazenamento em host local:
        - NFS server
        - NFS provisioner
    - Access mode não garante a restrição de montagem localmente em um ou múltiplos nodes, serve apenas para o match entre o pv e o pvc, já em provedores externos, cada um tem compatilibidade com um modo de acesso, o qual espera-se que funcione como restrição.

▼ HPA:

- Horizontal Pod Autoscaler

- Como o proprio nome diz, é o componente do kubernetes que faz a escala da criação de pods automaticamente de acordo com a necessidade de uso de recursos.

▼ Helm:

- Gerenciador de pacotes para o kubernetes
- São armazenados em helm charts, que são conjuntos de arquivos yaml.
- Assim como o docker, possui repositorios privados e públicos

▼ Daemonsets

Para serviços Web em geral - Deployments

Databeses - stateful

Jobs e schedule taks -cronjobs

Nodes - daemonSet

- Assim, um daemonset pode ser usado para configurar todos os nós de um cluster simultaneamente, atribuindo a eles componentes de armazenamento, monitoramento, etc.
- Com um daemonSet é possível declarar a atribuição de um só pod para rodar em todos os nodes disponíveis.
- Caso um serviço padrão seja atribuído a um daemonSet, o serviço abrirá conexão com o conjunto de pods inteiro com apenas um IP, servindo como uma espécie de redirecionador de requisições para cada nó aleatóriamente, ou até mesmo com um loadbalancer;
- O serviço pode ser declarado como headless, e, assim, abrirá conexão com cada pod individualmente, cada um com seu endereço de IP.

▼ Job e Cronjobs

- Um Job tem uma rotina com começo meio e fim.
- Permite execução de rotinas “paralelas”
- Permite a definição do número de falhas toleráveis, número de sucessos desejados, entre muitas outras.
- Para completar a sua tarefa, o job criará constantemente novos containers até que a tarefa seja concluída ou algum limite seja excedido.
- Cronjobs, nada mais são do que jobs agendados, facilitando o planejamento de rotinas periódicas automaticamente no cluster, executando jobs de tempos em tempos pré-definidos.

▼ RBAC (roles, service accounts, users)

- CA Certificates:
  - O kubernetes não reconhece users nativamente, e, para ter esse efeito, ele se utiliza de CA certificates, com os quais ele autentificará o usuário que estiver acessando o cluster.

        - Os certificados ficam geralmente na pasta /etc/kubernetes/pki o nó chamado control-plane;
        - A partir desse ponto, é preciso criar um certificado para o usuário, o que é feito a partir de um container auxiliar com a ferramenta openssl, para gerar o certificado desejado, assiná-lo e configurá-lo
        - https://github.com/marcel-dempers/docker-development-youtube-series/tree/master/kubernetes/rbac
    - Cubeconfig file:
        - Com o certificado pronto, é preciso adicioná-lo num arquivo de configuração do cluster, o cubeconfig.
        - Com esse arquivo, os usuários poderão ter acesso a determinado cluster o qual ele faz referência, desde que haja um certificado anexado a ele.
        - Geralmente o configfile padrão está contido na pasta .kube do usuário padrão do host local.
        - O administrador que possui esse arquivo, pode, então, criar um novo configfile para disponibilizar para o seu colega desenvolvedor, provendo acesso ao servidor kubernetes e a um cluster especifico, num namespace específico e com autorizações para funcionalidades especificas, definidas essas num role.
    - Roles:
        - Temos users e groups, cada qual qual seus roles, os quais são atribuídos via rolebindings.
        - Geralmente são usados a nivel de namespaces, por exemplo, uma permissão de leitura para um grupo específico a um namespace específico.
        - Para permissões a nível de cluster, existem os clusterroles e clusterbindings.
    - ServiceAccount:
        - ServiceAccounts, por sua vez, são utilizados para atribuir a uma aplicação, a um pod, permissões de acesso ao cluster e seus componentes, com um procedimento bem semelhante a autorização de um usuário.

▼ CRD

- Um resource é um endpoint na API do kubernetes que guarda uma coleção de APIObjects
- API/v1 por exemplo é um endpoint group
- um Deployment é um recurso, um api object
- Custom Resources Definition, então, é autoexplicativo, ou seja, o usuario do cluster kubernetes pode definir seus próprios recursos com seus próprios objetos e, inclusive propriedades.
- https://www.youtube.com/watch?v=u1X5Rf7fWwM

▼ Kind

▼ Configuração

- Instalação:

```
curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.17.0/kind-linux-amd64
chmod +x ./kind
sudo mv ./kind /usr/local/bin/kind
```

▼ Primeiros Comandos

- `kind create cluster`: cria um cluster baseado numa imagem padrão de node
  - `--help`: manual
  - `--image`: especifica outra imagem
  - `--wait`: bloqueia até que o control plane
  - `--name`: altera nome do cluster
  - `--config`: adiciona um arquivo de configuração para o cluster

  ▼ Arquivo de configuração base:

  - kind: Cluster
    apiVersion: kind.x-k8s.io/v1alpha4
    name: clusterKind
    nodes:
    - role: control-plane
        #image: OTHERIMAGE
        #freatureGates:
          #FEATURE: true|false
        extraPortMappings:
      - containerPort: 80
        hostPort: 80
        listenAddress: "0.0.0.0"
        protocol: tcp
    - role: worker
    - role: worker
    - role: worker
- `kind get clusters`: lista clusters criados
- `kind delete cluster --name CLUSTERNAME`: deleta o cluster alvo
- `kind export logs ./somedir`
- `kind load docker-image IMG_NAME`

▼ KubeCtl

▼ Configuração

- Instalação:

```
curl -LO "https://dl.k8s.io/release/$( curl -L -s https://dl.k8s.io/release/stable.txt )/bin/linux/amd64/kubectl"
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

▼ Comandos:

Gerenciamento do Cluster:

Cluster contexts:

- `kubeclt config get-contexts`
- `kubectl config use-context CONTEXTNAME`
- `kubectl cluster-info --context CONTEXTNAME`

Namespaces:

- `kubectl create namespace NAMENAMESPACE`
- `kubectl config set-context --current --namespace= NAMESPACENAME`

Configura o metrics server para HPA`s:

- `kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yam l`
- `hostNetwork: true`
- `--kubelet-insecure-tls=true`

Sobre os componentes do ingress:

- `kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/kind/deploy.yaml`

Criação de pods:

- `kubectl run PODNAME --image=IMAGE:VERSION`
- `kubectl edit pod PODNAME`
- `kubectl apply -f FILENAME`

Gerenciamento de pods:

- `kubectl get svc|secret|configmap|pod|deployments -o wide –watch`
- `kubectl delete deployment|pod DEPLOYMENTNAME|PODNAME`
- `kubectl describe pod|service|deploy COMPONENTNAME`
- `kubectl describe endpoints SERVICENAME`
- `kubectl patch pvc pvc_name -p '{"metadata":{"finalizers":null}}'`
- `kubectl logs PODNAME`
- `sudo kubectl get events --field-selector involvedObject.name = hpa-frontend`
- `kubectl exec -it PODNAME --container CONTAINERNAME -- bash`

Gerenciamento de Deployments:

- `kubectl rollout hisotry deployment DEPLOYNAME`
- `kubectl apply -f .\FILE --record #realiza alteração e grava no historico`
- `kubectl annotate deployment DEPLOYNAME kubernetes.io/change-cause="texto de anotação"`
- `kubectl rollout undo deployment DEPLOYNAME --to-revision=x`

- `kubectl edit deploy DEPLOYNAME --namespace NAMENAMESPACE`

Gerenciamento dos nodes:

- `kubectl label nodes worker-2.example.com color=blue`
- `kubectl get nodes --show-labels`

▼ Helm exemplos

- `helm repo add metrics-server https://kubernetes-sigs.github.io/metrics-server/`
- `helm upgrade --install metrics-server metrics-server/metrics-server --set hostNetwork.enabled=true -f utils/valuesMetricsServer.yaml`

- `helm repo add nfs-subdir-external-provisioner https://kubernetes-sigs.github.io/nfs-subdir-external-provisioner/`
- `helm install nfs-subdir-external-provisioner nfs-subdir-external-provisioner/nfs-subdir-external-provisioner --set nfs.server=192.168.65.2 --set nfs.path=/srv/nfs/k8sdata`

- `helm list`
- `helm delete`
- `helm delete metrics-server`

▼ NFS Server Exemplo

- sudo apt install nfs-kernel-server
- create a directory to share
- sudo chown nobody: /directory
- edit /etc/exports file
- sudo service rpcbind restart
- sudo service nfs-kernel-server start
- sudo exportfs -rav
- sudo showmount -e localhost
- apt install -y nfs-common
- mount -t nfs 192.168.65.2:/srv/nfs/k8sdata (srv para dados para serviços)
- mount | grep k8sdata

▼ Explicação a respeito da conectividade entre o serviço de storage no wsl com o container docker

Para entender o que está acontecendo de fato é preciso analisar toda a infraestrutura de rede da arquitetura de virtualização que está implementada na maquina.

O windows possui um ip para a internet, no caso na interface de wifi, com ip `192.168.15.55`, verificado com `ipconfig`

O docker desktop necessita de um ip para acessar o host, que no caso possui como o DNS `host.docker.internal` e, coincidentemente, usa como numero de ip o mesmo ip 192.168.15.55 da interface de wifi do windows verificado no arquivo `C:\Windows\System32\drivers\etc/hosts` no windows (que foi adicionado pelo docker desktop) e no `/etc/hosts` no linux, que herda o mesmo endereço por conta da virtualização.

▼ Para o docker desktop com a host.docker.internal habilitada:

Ao entrar na rede docker, cada node receberá um ip para resolução de DNS, que pode ser encontrado no arquivo `/etc/resolv.conf`, que, ao observar a faixa de rede, é a mesma do endereço da rede interna do docker→maquina host, verificado com `ping host.docker.internal`, assim, espera-se então que ao iniciar um nó, o runtime docker atribui um endereço para o servidor baseado na rede da interface de internet do windows.

Assim, o nó virtualizado consegue se conectar com exposições de montagem de armazenamentos ou serviços da maquina que o hospeda, como no servidor nfs por exemplo.

Ainda há uma analise a ser feita, pois ao observar os endereços do nó com ip addr, vemos o seu ip interno, o mesmo verificado com `kubectl get nodes -o wide.` e ao analisar as rotas de rede com o `ip route` dentro do container vemos que o endereço 172.18.0.1 é o endereço default, o qual provavelmente está relacionado com o host também. que fornece a rede para os ips locais de cada container docker.

▼ Para o docker nativo no wsl:

Para o docker nativo no wsl, a conexão com o localhost do wsl é feita através da interface docker0 criada ao baixar o docker direto no linux, sem utilizar a host.internal.docker

essa interface vai conectar o nfs client do node até a montagem aberta no localhost da maquina host.

▼ Comandos troubleshooting interno

Comandos de dentro do nó control-plane:

- `docker exec -it IDCONTAINER bash`
- `crictl -r unix:///run/containerd/containerd.sock ps`
- `journalctl -ur kubelet`
- `watch ps aux`
- `cd /var/log`
- `crictl -r unix:///run/containerd/containerd.sock logs IDCONTAINER`

▼ Criação do projeto passo a passo:

- `sudo service docker start`

Iniciar e parar os containers do cluster:

▼ Roteiro para apresentação:

Kubernetes:

conceito

pré-requisitos

- `docker start $(docker ps -a | grep -oP '(?<=\s)clusterbackup[^\s]+$')`
- `docker container stop $(docker container ls -q --filter name=clusterbackup*)`
- `kubeclt config get-contexts`
- `kubectl config use-context CONTEXTNAME`

Criação de cluster com criptografia de banco e namespaces:

- `kind create cluster --config cluster-config.yaml`
- Baixar ferramentas para alterar api-server e verificar criptografia no control-plane
  - `apt-get install bsdmainutils -y`
  - `apt install etcd-client -y`
  - `apt-get install vim -y`
  - `vi /etc/kubernetes/manifests/kube-apiserver.yaml`
- `sudo kubectl apply -f namespaces.yaml`

Adição de serviços, metrics server, nfs-client, ingress-controller:

- `helm upgrade --install metrics-server metrics-server/metrics-server --set hostNetwork.enabled=true -f utils/valuesMetricsServer.yaml`
- `helm install nfs-subdir-external-provisioner nfs-subdir-external-provisioner/nfs-subdir-external-provisioner --set nfs.server=172.17.0.1 --set nfs.path=/srv/nfs/k8sdata`
- `kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/kind/deploy.yaml`

Criação dos configmaps, secrets e services e pvc's:

- `kubectl apply -f services/.`
- `kubectl apply -f pvc/.`
- `kubectl apply -f configmaps/.`

Kubernetes vs Docker

Pods vs Container

▼ Componentes: Aplicação + Helm

- Criptografia do banco etcd
  - kubectl create secret generic secret1 -n default --from-literal=mykey=mydata
  - ETCDCTL_API=3 etcdctl\ --cacert=/etc/kubernetes/pki/etcd/c --cert=/etc/kubernetes/pki/etcd/ser --key=/etc/kubernetes/pki/etcd/ser get /registry/secrets/default/secret1 | hexdump -C
  - Mostrar as montagens do encryption.yaml
  - Mostrar secrets criptografados
    - `ETCDCTL_API=3 etcdctl --cacert=/etc/kubernetes/pki/etcd --cert=/etc/kubernetes/pki/etcd/s --key=/etc/kubernetes/pki/etcd/se get /registry/secrets/backend/secre backend | hexdump -C`
  - Mostrar os container rodando de dentro do nó
    - `crictl -r unix:///run/containerd/containe ps -a`
  - mostrar o api-server
    - `vi /etc/kubernetes/manifests/kube-apiserver.yaml`

- `kubectl apply -f secrets/.`

Criação dos pods do sistema:

- `kubectl apply -f backend-statefulset.yaml`
- `kubectl apply -f db-mysql-deployment.yaml`
- `kubectl apply -f frontend-deployment.yaml`
- `kubectl apply -f frontend-hpa.yaml`

Criação do ingress config:

- `kubectl apply -f utils/ingress-config.yaml`

- Namespaces
- Configmaps e Secrets
  - Mostrar variaveis de ambiente no statefulset do backend
  - Mostrar os secrets como volumes no pod do my-sql
  - Falar sobre a diferença na segurança dessas duas abordagens
- ReplicaSet, Deployments e Statefulsets
- Nfs-client, SC, PVC e StatefulSet
  - mostrar montagem no host wsl
  - mostrar a storage class criada pelo helm e como funciona a estrutura por trás das montagens externas
  - `cat /etc/exports` no wsl
  - `ipconfig` no windows
  - `C:\Windows\System32\drivers\etc/hos` no windows
  - `/etc/hosts` no linux
  - `/etc/resolv.conf` nos nodes
- Metrics-server e HPA
  - mostrar pods e suas funcionalidades
  - Rodar `bash stress.sh` e mostrar hpa em ação
  - `sudo kubectl get hpa -n frontend –watch`
  - `sudo kubectl get pods -n frontend –watch`
- services e ingress
  - mostrar pods e suas funcionalidades
  - mostrar o config utilizando o nip.io e o motivo de alterar o host

        - mostrar os external names para acessar serviços de fora do namespace

- ▼ Conceitos
    - Daemonsets
    - Job e Cronjobs
    - RBAC (roles, service accounts, users)
    - CRD

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