Instalação com Kubeadm

Existem diversas formar de criar o seu cluster Kubernetes, aqui, o objetivo vai ser criar o seu cluster de forma on-premisse utilizando o kubeadm.

O kubeadm é uma ferramenta com o objetivo de facilitar a criação de um cluster Kubernetes padrão, que segue todos os requisitos de um cluster certificado. Ou seja, você vai ter o básico de um cluster Kubernetes validado pela Cloud Native Computing Foundation. Mas você também vai usar o kubeadm pra alguns processos de manutenção do cluster, como renovação de certificados e atualizações do cluster.

Você pode utilizar o kubeadm em qualquer abordagem on-premisse de uso do Kubernetes, seja máquinas virtuais, máquinas baremetal e até mesmo Raspberry Pi.

Setup do Ambiente

Aqui eu vou mostrar como criar um cluster Kubernetes utilizando 3 máquinas, uma máquina vai ter o papel de Control Plane e as outras duas de Worker Nodes. Lembrando que dessa forma eu não estou criando um cluster com alta disponibilidade ou HA. Pois eu tenho apenas um control plane e caso ele fique fora do ar, o cluster vai ficar inoperável. Então utiliza esse setup em ambientes de estudo, teste, desenvolvimento e caso você não preciso de alta disponibilidade, homologação. NUNCA utilize em PRODUÇÃO

Requisitos da Instalação

Abaixo segue os requisitos mínimos pra cada máquina:

- Máquina Linux (aqui no caso vou utilizar Ubuntu 20.04)
- 2 GB de memória RAM
- 2 CPUs
- Conexão de rede entre as máquinas
- Hostname, endereço MAC e product_uuid únicos pra cada nó.
- Swap desabilitado

Além da conexão de rede entre as máquinas, é importante garantir que as portas utilizadas pelo Kubernetes estejam abertas. Segue abaixo a tabela com as portas para as máquinas que atuam como control plane e as máquinas que atuam como worker node:

Portas para o control plane

Protocolo	Range de Porta	Uso	Quem consome
TCP	6443	Kubernetes API server	Todos
TCP	2379-2380	etcd server client API	kube-apiserver, etcd
TCP	10250	Kubelet API	Self, Control plane
TCP	10259	kube-scheduler	Self
TCP	10257	kube-controller-manager	Self

Portas para o worker node

Protocolo	Range de Porta	Uso	Quem consome
TCP	10250	Kubernetes API server	Self, Control plane
TCP	30000-32767	NodePort Services	Todos

Instalação

Agora vamos pro passo a passo da instalação.

Container Runtime (Containerd)

O primeiro passo, é instalar em TODAS as máquinas o container runtime, ou seja, quem vai executar os containers solicitados pelo kubelet. Aqui o container runtime utilizado é o Containerd, mas você também pode usar o Docker e o CRI-O.

Antes de instalar o Containerd, é preciso habilitar alguns módulos do kernel e configurar os parâmetros do sysctl

Instalação dos módulos do kernel

```
cat <<EOF | sudo tee /etc/modules-load.d/containerd.conf overlay
br_netfilter EOF sudo modprobe overlay sudo modprobe br_netfilter
```

Configuração dos parâmetros do sysctl

```
# Configuração dos parâmetros do sysctl, fica mantido mesmo com reebot da
máquina. cat <<EOF | sudo tee /etc/sysctl.d/99-kubernetes-cri.conf
net.bridge.bridge-nf-call-iptables = 1 net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1 EOF # Aplica as definições do
sysctl sem reiniciar a máquina sudo sysctl --system
```

Agora sim, podemos instalar e configurar o Container

Instalação

OBS: A partir da versão 1.26 do Kubernetes, foi removido o suporte ao CRI v1alpha2 e ao Containerd 1.5. E até o momento que escrevo esse guia, o repositório oficial do Ubuntu não tem o Containerd 1.6, então precisamos usar o repositório do Docker pra instalar o ContainerD.

Kubernetes v1.26: Electrifying

Authors: Kubernetes 1.26 Release Team It's with immense joy that we announce the release of Kubernetes v1.26! This release

https://kubernetes.io/blog/2022/12/09/kubernetes-v1-26-r...

Adicionando o repositório do Docker

```
# Instalação de pré requisitos sudo apt update && \ sudo apt install \ ca-certificates \ curl \ gnupg \ lsb-release -y # Adicionando a chave GPG sudo mkdir -p /etc/apt/keyrings curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \ sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg # Configurando o repositório echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" \ | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null # Atualizando o repositório sudo apt-get update
```

```
sudo apt update && sudo apt install -y containerd.io -y
```

Configuração padrão do Containerd

```
sudo mkdir -p /etc/containerd && containerd config default | sudo tee /etc/containerd/config.toml
```

Alterar o arquivo de configuração pra configurar o systemd cgroup driver.

> Sem isso o Containerd não gerencia corretamente os recursos computacionais e vai reiniciar em loop

```
sed -i 's/SystemdCgroup = false/SystemdCgroup = true/g' /etc/containerd/config.toml
```

Agora é preciso reiniciar o container

```
sudo systemctl restart containerd
```

Instalação do kubeadm, kubelet and kubectl

Agora que eu tenho o container runtime instalado em todas as máquinas, chegou a hora de instalar o kubeadm, o kubelet e o kubectl. Então vamos seguir as etapas e executar esses passos em TODAS AS MÁQUINAS.

Atualizo os pacotes necessários pra instalação

```
sudo apt-get update && \ sudo apt-get install -y apt-transport-https ca-certificates curl
```

Download da chave pública do Google Cloud

```
curl -fsSL https://dl.k8s.io/apt/doc/apt-key.gpg | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-archive-keyring.gpg
```

Adiciono o repositório apt do Kubernetes

```
echo "deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list
```

Atualização do repositório apt e instalação das ferramentas

```
sudo apt-get update && \ sudo apt-get install -y kubelet kubeadm kubectl
```

Agora eu garanto que eles não sejam atualizados automaticamente.

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

Iniciando o cluster Kubernetes

Agora que todos os elementos estão instalados, tá na hora de iniciar o cluster Kubernetes, então eu vou executar o comando de inicialização do cluster. Esse comando, você vai executar APENAS NA MÁQUINA QUE VAI SER O CONTROL PLANE !!!

Comando de inicialização

```
kubeadm init
```

Você também pode incluir alguns parâmetros:

--apiserver-cert-extra-sans ⇒ Inclui o IP ou domínio como acesso válido no certificado do kube-api. Se você tem mais de 1 adaptador de rede no cluster (um interno e um externo por exemplo) é importante que você utilize.

--apiserver-advertise-address ⇒ Define o adaptador de rede que vai ser responsável por se comunicar com o cluster.

--pod-network-cidr ⇒

Estágios da inicialização

Durante o processo de inicialização do cluster, alguns estágios são executados:

preflight ⇒ Valida o sistema e verfica se é possível fazer a instalação. Ele pode exibir alertas ou erros, no caso de erro, ele sai da inicialização.

certs ⇒ Gera uma autoridade de certificação auto assinada para cada componente do Kubernetes. Isso garante a segurança na comunicação com o cluster.

kubeconfig ⇒ Gera o kubeconfig no diretório /etc/kubernetes e os arquivos utilizados pelo kubelet, controller-manager e o scheduler pra conectar no api-server.

kubelet-start, control-plane e etcd ⇒ Configura o kubelet pra executar os pods com o api-server, controller-manager, o scheduller e o etcd. Depois inicia o kubelet.

mark-control-plane ⇒ Aplica labels e tains no control plane pra garantir que não vai ser executado nenhum pod dentro dele.

addons ⇒ Adiciona o CoreDNS e o kube-proxy

Iniciado o cluster, é preciso copiar as configurações de acesso do cluster para o kubectl

Configurando o kubectl

```
mkdir -p $HOME/.kube sudo cp -i /etc/kubernetes/admin.conf
$HOME/.kube/config sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Agora que o próximo passo é incluir os worker nodes no cluster, pra isso no output de inicialização do cluster já aparece o comando kubeadm join pra executar nos worker nodes, mas se você perder ou precisar do comando de novo, é só executar no control plane o comando token create

Gerando o comando join e executando nos nodes

```
kubeadm token create --print-join-command
```

```
kubeadm join 159.223.123.99:6443 --token 4qefmj.lj9hx9atef5a9xnj --
discovery-token-ca-cert-hash
sha256:7f72c6d435aba7d320661741df4c1d3b8830414057e0c13d0ba1fa84ef4e4306
```

Agora, se você executar o kubectl get nodes, vai ver que o control plane e os nodes não estão prontos, pra resolver isso, é preciso instalar o Container Network Interface ou CNI, aqui eu vou usar o Weave Net

Instalação do CNI

```
kubectl apply -f
https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-
daemonset-k8s.yaml
```

IMPORTANTE: Se você estiver utilizando firewall, deve ser liberada as portas TCP 6783 e UDP 6783/6784 para o Weave Net

Teste de instalação

Pra saber se está tudo funcionando, vamos fazer o deploy de algo e ver se tudo dá certo.

```
apiVersion: apps/v1 kind: Deployment metadata: name: nginx spec: selector:
matchLabels: app: nginx template: metadata: labels: app: nginx spec:
containers: - name: nginx image: nginx ports: - containerPort: 80 ---
apiVersion: v1 kind: Service metadata: name: nginx spec: selector: app:
nginx ports: - port: 80 targetPort: 80 nodePort: 30000 type: NodePort
```

Agora sim, você tem o cluster Kubernetes instalado e funcionando. Lembrando que você só deve usar esse setup em testes e NUNCA EM PRODUÇÃO !!!