KUBERNETES

Fernando C. Menardi

Obtendo Docker e Ferramentas do Kubernetes

O kubernetes utiliza o docker, sendo assim, é necessário instalar o docker em todas as instâncias (master e nodes), seguindo os comandos a seguir (exclusivos para o ubuntu).

Com privilégios de superusuário (root):

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key
add -

add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"

apt-get update

apt-get install -y docker-ce=18.06.1~ce~3-0~ubuntu

apt-mark hold docker-ce
```

No "apt-get" é especificada uma versão do docker. Esta versão é compatível com outros serviços que vão ser instalados e utilizados com o kubernetes, podendo não ser a versão atualizada. **É necessário verificar compatibilidade e manter uma versão estável e atualizada do docker**, caso prossiga com uma instalação para fins de aplicação direta.

Para verificar se tudo foi instalado corretamente utilize o comando:

```
docker version
```

Agora, para facilitar processos futuros, prosseguimos com a instalação de três ferramentas adicionais.

- ✓ **Kubeadm**: Automatiza o processo de construção do cluster.
- ✓ **Kubelet**: Componente essencial do kubernetes, responsável por lidar com os containers que estão rodando em um node (todo server que estiver rodando containers precisa do Kublet.
- ✓ **Kubectl**: Terminal de controle para interagir com o cluster quando o mesmo estiver rodando.

Para instalar, execute os comandos a seguir em todos os três servidores:

```
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo
apt-key add -

cat << EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF

apt-get update

sudo apt-get install -y kubelet=1.12.7-00 kubeadm=1.12.7-00
kubectl=1.12.7-00

sudo apt-mark hold kubelet kubeadm kubectl
```

Para verificar se a instalação foi concluída corretamente:

```
docker version
```

Inicializando Cluster

O Processo de bootstrap e inicialização do cluster devem ser feitos no host "MASTER" executando o seguinte comando:

```
sudo kubeadm init --pod-network-cidr=10.244.0.0/16
```

A flag "pod-network-cidr" especifica o alcance dos endereços IP para a rede de "Pods".

O conceito de "Pods" pode ser esclarecido aqui:

1. https://medium.com/google-cloud/understanding-kubernetes-networking-pods-7117dd28727

Ao final do resultado retornado pelo comando "kubeam init" é exibido um comando "kubeadm join" contendo um token e uma hash. Esse comando é utilizado para que os nodes possam se conectar ao master, guarde-o.

```
sudo kubeadm join $some_ip:6443 --token $some_token --discovery-token-
ca-cert-hash $some_hash
```

O comando a seguir aplica as configurações locais (master).

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

Para verificar se o cluster está respondendo, execute:

```
kubectl version
```

Nesse ponto, é preciso verificar atentamente a resposta do sistema. A versão do cliente e do servidor deve ser exibida na tela, isso indica que o cliente e o servidor conseguem se comunicar e foram configurados corretamente.

Agora, execute nos nodes o comando “kubeadm join” que foi retornado pelo comando “kubeadm init” anteriormente.

```
Client Version: version.Info{Major:"1", Minor:"12",
GitVersion:"v1.12.2",
GitCommit:"17c77c7898218073f14c8d573582e8d2313dc740",
GitTreeState:"clean", BuildDate:"2018-10-24T06:54:59Z",
GoVersion:"go1.10.4", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"12",
GitVersion:"v1.12.2",
GitCommit:"17c77c7898218073f14c8d573582e8d2313dc740",
GitTreeState:"clean", BuildDate:"2018-10-24T06:43:59Z",
GoVersion:"go1.10.4", Compiler:"gc", Platform:"linux/amd64"}
```

Caso tudo esteja certo, uma mensagem “This node has joined the cluster” irá ser exibida.

Verifique se todos os nodes se conectaram ao cluster execuando o comando a seguir no master:

```
kubectl get nodes
```

A resposta deve ser como a exibida a seguir e deve apresentar info de todos os nodes conectados.

```
NAME                      STATUS     ROLES    AGE     VERSION
wboyd1c.mylabserver.com   NotReady   master   5m17s   v1.12.2
wboyd2c.mylabserver.com   NotReady   <none>   53s     v1.12.2
wboyd3c.mylabserver.com   NotReady   <none>   31s     v1.12.2
```

Para obter mais informações sobre um node especifico, use o comando:

```
kubectl describe node $node_name
```

Cluster Networking

O status de todos os nodes apresentados pelo comando "kubectl get nodes" é "NotReady", para mudar isso, precisamos de um plugin de networking chamado "flammel" e configurar o iptables para permitir tráfego do "guest" para o "host" durante a virtualização. Tudo isso pode ser feito executando os seguintes comandos:

```
echo "net.bridge.bridge-nf-call-iptables=1" | sudo tee -a
/etc/sysctl.conf
sudo sysctl -p

kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/bc79dd1505b0c8681ece4
de4c0d86c5cd2643275/Documentation/kube-flannel.yml
```

Agora, rodando novamente o comando "kubectl get nodes", o status de todos os nodes deve ter sido alterado para "Ready".

Namespaces e Pods

O "pod" é a unidade básica do kubernetes. Cada "pod" possui pelo menos um container (normalmente apenas um) e seu próprio endereço IP. O Kubernetes escala os pods para rodarem nos servidores presentes no cluster.

O "namespace" é um cluster virtual. Ele oferece um meio de dividir os recursos de um cluster para vários usuários.

O Kubernetes inicializa com três "namespaces" padrão:

- **default**: para objetos que não possuem um "namespace".
- **kube-system**: para objetos que foram criados pelo kubernetes.
- **kube-public**: criado automaticamente e pode ser lido por todos os usuários por padrão.

Criar/Deletar um "namespace":

```
Kubectl create/delete namespace <namespace>
```

Listar "namespace":

```
Kubectl get namespaces
```

Informacional adicional sobre um "namespace":

```
Kubectl describe namespace <namespace>
```

Para criar um pod, existem duas possibilidades:

```
Kubectl create deployment <name> --image=<image>
```

Com STDIN:

```
cat << EOF | kubectl create -f -
> apiVersion: v1
> kind: Pod
> metadata:
>   name: nginx
> spec:
>   containers:
>   - name: nginx
>     image: nginx
> EOF
```

Para ver os pods ativos:

```
Kubectl get pods
```

Para obter informações adicionais sobre um pod:

```
Kubectl describe pod <pod name>
```

Para deletar um pod:

```
Kubectl delete pod <pod name>
```

Para mais detalhes sobre como criar uma imagem para docker utilizando uma aplicação pronta:

1. https://cloud.google.com/kubernetes-engine/docs/tutorials/hello-app

2. https://www.linode.com/docs/kubernetes/deploy-container-image-to-kubernetes/

Para lançar um pod/deployment de forma mais efetiva, é possível utilizar arquivos YAML. É possível descrever múltiplos lançamentos diferentes, facilitando e agilizando a criação/manutenção de pods e deployments.

Para criar um pod utilizando yaml:

```
kubectl create -f <pod name>.yaml
```

Da mesma forma, para criar um deployment:

```
kubectl create -f <deployment name>.yaml
```

Para criar um Pod/Deplyment sob um namespace:

```
kubectl –n <custom-namespace> create -f <deployment name>.yaml
```

OBS: Enquanto o pod roda um determinado número de containers, o Deployment é responsável por rodar uma série de pods idênticos, monitorando o estado de cada um deles e realizando atualizações caso necessário. Levando em conta a produtividade, o Deployment é mais efetivo.

Para mais detalhes sobre como lançar pods e deployments a partir de arquivos YAML:

1. https://www.mirantis.com/blog/introduction-to-yaml-creating-a-kubernetes-deployment/

Kubernetes API

Requisições HTTP RESTful:

- GET – Retorna um recurso ou uma lista de recursos.
- POST – Cria um novo recurso.
- PUT – Usado para atualizar um recurso manualmente.
- PATCH – Modifica seletivamente um campo especifico de um recurso.
- DELETE – Deleta um recurso.

Para interagir com a API é necessário ativar o proxy do kubectl:

```
Kubectl proxy –port=8080 &
```

Para utilizar a API sem proxy, é necessário utilizar o token e passar o mesmo durante a requisição.

Para obter o token:

```
Kubectl describe secret
```

Para fazer a requisição (usando curl, por exemplo):

```
curl $APISERVER/api –header “Authorization: Bearer $TOKEN” --
insecure
```

Escalonamento

O kubernetes oferece a possibilidade de escalar componentes individuais de um microserviço de forma independente, fazendo com que recursos não sejam desperdiçados. Isso pode ser muito útil no caso de um dos pods no cluster precisar de mais recursos ou tiver mais tráfego de usuários que outro pod.

É possível realizar esse escalonamento de forma manual ou automática:

1. **Escalonamento Manual:**

```
Kubectl scale deployment/<deployment> --replicas=<number> -n
<namespace>
```

--replicas: Número de pods que devem ser mantidos em execução.

2. **Auto escalonamento:**

O Kubernetes implementa o *Horizontal Pod Autoescaler*, ou "HPA". O "HPA" é implementado como um recurso de API do kubernetes que controla e escalona automaticamente o número de pods baseado no monitoramento do uso de CPU (ou outros critérios).

Criar um HPA para um deployment, considerando o uso de 65% da CPU :

```
Kubectl autoscale deployment <DEPLOYMENT> -n <NAMESPACE> --min
2 --max 6 --cpu-percent 65
```

Listar HPAs:

```
Kubectl get hpa –n <NAMESPACE>
```

Deletar HPA:

```
Kubectl delete hpa –n <NAMESPACE> <HPA>
```