**Tutorial de implantação de um cluster do kubernetes com volume persistente distribuído por meio de um cluster no *GlusterFS* e monitorado pelo *Prometheus e Grafana*.**

Versão do sistema operacional: Debian 10.6.0. 64 bits

Para este tutorial, será necessário Sete máquinas: 3 para o master, 3 para os workers e 1 para o Haproxy, totalizando 6 nodes e 1 HAproxy.

1. **Instalando e configurando o HAProxy**

Execute o seguinte comando para a instalação Haproxy:

sudo apt-get update

sudo apt-get install -y haproxy

Aplique a configuração do haproxy.cfg no arquivo “/etc/haproxy/haproxy.cfg”. Em seguida, reinicie o serviço com o comando:

Sudo systemctl restart haproxy

1. **Instalação do Docker**

Executar o seguinte comando para a instalação do Docker nas 3 máquinas workers e nas 3 master:

sudo apt-get update

sudo curl -fsSL https://get.docker.com -o get-docker.sh

sudo sh get-docker.sh

1. **Alteração dos hostnames**

Para facilitar, foi alterado o *hostname* em todos os nós*:*

Alterar o *hostname* do haproxy:

sudo hostnamectl set-hostname "haproxy"

Alterar o *hostname* do master 1:

sudo hostnamectl set-hostname "master1"

Alterar o *hostname* do master 2:

sudo hostnamectl set-hostname "master2"

Alterar o *hostname* do master 3:

sudo hostnamectl set-hostname "master3

Alterar o *hostname* do worker 1:

sudo hostnamectl set-hostname "worker1"

Alterar o *hostname* do worker 2:

sudo hostnamectl set-hostname "worker2"

Alterar o *hostname* do worker 3:

sudo hostnamectl set-hostname "worker3"

Inserir no arquivo /etc/hosts as seguintes configurações:

sudo nano /etc/hosts

192.168.0.11 master1

192.168.0.12 master2

192.168.0.13 master3

192.168.0.14 worker1

192.168.0.15 worker2

192.168.0.16 worker3

192.168.0.17 haproxy

1. **Habilitar encaminhamento de IP e desabilitar swap**

Para desabilitar o *swap* é necessário editar o arquivo “/etc/fstab”, comentando a linha responsável pela sua execução após cada reboot e então executar o comando “sudo swapoff --a" para desabilitar de imediato.

E para habilitar o encaminhamento de IP será necessário editar o arquivo “sudo nano /etc/sysctl.conf” e então adicionar a linha “net.ipv4.ip\_forward=1”.

1. **Instalação do Kubectl, Kubelet e o Kubeadm em todos os nodes.**

Para a instalação, execute os seguintes comandos:

sudo apt install -y apt-transport-https curl

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

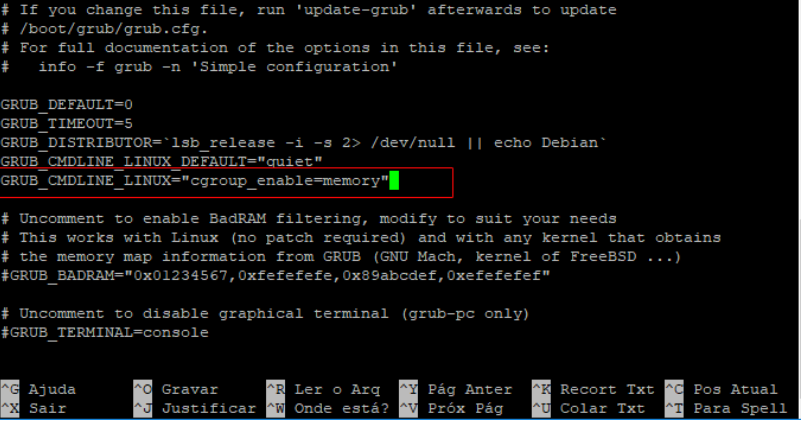
sudo apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"

sudo apt update

sudo apt install -y kubelet kubeadm kubectl

1. **Habilitando o CGROUPS.**

Por padrão, o cgroup vem por padrão desabilitado no Debian. Para isso, a necessidade de habilitar o Cgroup nos 6 nodes, editando o arquivo “/etc/default/grub” e inserindo a opção GRUB\_CMDLINE\_LINUX="cgroup\_enable=memory", atualizar o GRUB e reiniciar a máquina.



sudo nano /etc/default/grub

sudo update-grub

sudo reboot

1. **Iniciando o cluster kubernetes**

No master1, execute o seguinte comando para iniciar o cluster:

sudo kubeadm init --control-plane-endpoint “haproxy:6443” --upload-certs

Após a iniciação do máster, salve os dois “*join*” que apareceu na saída para que os nodes possam ingressar ao cluster.

Em seguida, será necessário a instalação de um componente de rede para a comunicação entre os nodes no node master:

sudo kubectl apply -f https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')

Depois, inclua os nodes restante dentro do cluster Kubernetes, com o *join* respectivos e fornecidos. Abaixo, consta os comandos para inclusão dos nodes no nosso cenário.

Incluindo o master 2:

tierry@master2:~ sudo kubeadm join 192.168.0.17:6443 --token kwrjs6.s03y3gh8oumgsh6v --discovery-token-ca-cert-hash sha256:91e662d773b6775e530a356534eb36de828df637c48b9ad7b5d266ec43523870 --control-plane --certificate-key 5da0f39004a76508041b412fb5e3faf3ed0f045d1ffd17f170996e2c84b240cc

Incluindo o master 3:

tierry@master3:~ sudo kubeadm join 192.168.0.17:6443 --token kwrjs6.s03y3gh8oumgsh6v --discovery-token-ca-cert-hash sha256:91e662d773b6775e530a356534eb36de828df637c48b9ad7b5d266ec43523870 --control-plane --certificate-key 5da0f39004a76508041b412fb5e3faf3ed0f045d1ffd17f170996e2c84b240cc

Incluindo o worker 1:

tierry@worker1:~ sudo kubeadm join 192.168.0.17:6443 --token kalond.o4x4codx4rs5rol2 --discovery-token-ca-cert-hash sha256:91e662d773b6775e530a356534eb36de828df637c48b9ad7b5d266ec43523870

Incluindo o worker 2:

tierry@worker2:~ sudo kubeadm join 192.168.0.17:6443 --token kalond.o4x4codx4rs5rol2 --discovery-token-ca-cert-hash sha256:91e662d773b6775e530a356534eb36de828df637c48b9ad7b5d266ec43523870

Incluindo o worker 3:

tierry@worker3:~ sudo kubeadm join 192.168.0.17:6443 --token kalond.o4x4codx4rs5rol2 --discovery-token-ca-cert-hash sha256:91e662d773b6775e530a356534eb36de828df637c48b9ad7b5d266ec43523870

E para que possamos interagir com o *Kubernetes* utilizando o usuário non-root do *Linux*, execute os seguintes comandos nos 3 master:

sudo mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

Comando para verificar o funcionamento dos nodes:

tierry@master:~$ kubectl get nodes

NAME STATUS ROLES AGE VERSION

master1 Ready master 18m v1.19.3

master2 Ready master 18m v1.19.3

master3 Ready master 18m v1.19.3

worker1 Ready <none> 3m58s v1.19.3

worker2 Ready <none> 3m58s v1.19.3

worker3 Ready <none> 3m58s v1.19.3

Agora podemos ver que o cluster está pronto para a sua utilização.

Para que os comandos do Kubectl possa auto completar no *bash,* há necessidade executar os seguintes comandos nos 6 nodes:

sudo echo 'source <(kubectl completion bash)' >>~/.bashrc

sudo source .bashrc

1. **Criando container do banco de dados Mysql**

Crie o banco de dados com o comando abaixo

sudo docker run --name moodle-mysql -v /data/docker/datadir:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=password -d --restart unless-stopped -p 3306:3306 mysql:5.7.32

1. **Instalação do *GlusterFS***

Com o objetivo de obter um volume persistente para o cluster do Kubernetes, foi instalado o GlusterFS nos 3 *master*. Para a instalação do cluster distribuído do GlusterFS, o seguinte procedimento foi efetuado.

Execute os comandos no 3 master:

sudo add-apt-repository ppa:gluster/glusterfs-7

sudo apt update

sudo apt install glusterfs-server

sudo systemctl start glusterd.service

sudo systemctl enable glusterd.service

Executar no *master1*:

sudo gluster peer probe master2

sudo gluster peer probe master3

Execute os comandos no 3 worker:

sudo add-apt-repository ppa:gluster/glusterfs-7

sudo apt update

sudo apt install -y glusterfs-client

Verifique o status do gluster

tierry@master1:~$ sudo gluster peer status

[sudo] password for tierry:

Number of Peers: 2

Hostname: master2

Uuid: 78f095ce-69ec-49c0-b19f-4768b6b12b6e

State: Peer in Cluster (Connected)

Hostname: master3

Uuid: 949a2f7d-91d8-4c8a-9cdd-d8be3647a5a5

State: Peer in Cluster (Connected)

Criar o diretório nos 3 master:

sudo mkdir -p /data/glusterfs/brick1/gv0

Criação do cluster replicado no *master1*:

tierry@master1:~$ sudo gluster volume create gv0 replica 3 master1:/data/glusterfs/brick1/gv0 master2:/data/glusterfs/brick1/gv0 master3:/data/glusterfs/brick1/gv0 force

Iniciar o volume:

tierry@master1:~$ sudo gluster volume start gv0

volume start: gv0: success

Verificação do status do volume:

tierry@master1:~/projeto-yaml$ sudo gluster volume info

Volume Name: gv0

Type: Replicate

Volume ID: 53ccd784-3d22-4dff-b5f9-f774a1116ca6

Status: Started

Snapshot Count: 0

Number of Bricks: 1 x 3 = 3

Transport-type: tcp

Bricks:

Brick1: master1:/data/glusterfs/brick1/gv0

Brick2: master2:/data/glusterfs/brick1/gv0

Brick3: master3:/data/glusterfs/brick1/gv0

Options Reconfigured:

performance.client-io-threads: off

nfs.disable: on

storage.fips-mode-rchecksum: on

transport.address-family: inet

1. **Criação do volume persistente utilizando o cluster do *GlusterFS***

Após a configuração do cluster GlusterFS, os yaml a seguir possibilitam a criação de um volume persistente para armazenar os dados da aplicação. A sua configuração consiste em definir os IPs disponíveis do cluster GlusterFS para ser utilizado no Kubernetes por meio dos arquivos de tipo “Endpoints” e “Service”. Então o yaml de tipo “PersistentVolume” trata de especificar o tipo de plugin a ser utilizado que no caso foi o “GlusterFS”, o nome do volume e seu tamanho. Por fim, o “PersistentVolumeClaim” permite que os pods dos Deployments possam montar suas pastas neste volume persistente de forma que os arquivos armazenados pelo mesmo possam ser acessados por outros pods de outros workers. Importante mencionar que a montagem das pastas são realizadas pelos workers dos PODs, ou seja, é necessário que as máquinas workers possuam o cliente do GlusterFS instalado para que possam ser montados por um dos IPs configurados pelo yaml do tipo Endpoints.

PersistentVolume

apiVersion: v1

kind: PersistentVolume

metadata:

# The name of the PV, which is referenced in pod definitions or displayed in various oc volume commands.

name: gluster-pv

spec:

capacity:

# The amount of storage allocated to this volume.

storage: 3Gi

accessModes:

# labels to match a PV and a PVC. They currently do not define any form of access control.

- ReadWriteMany

# The glusterfs plug-in defines the volume type being used

glusterfs:

endpoints: glusterfs-cluster

# Gluster volume name, preceded by /

path: /gv0

readOnly: false

# volume reclaim policy indicates that the volume will be preserved after the pods accessing it terminate.

# Accepted values include Retain, Delete, and Recycle.

persistentVolumeReclaimPolicy: Retain

PersistentVolumeClaim

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: gluster-claim

spec:

accessModes:

- ReadWriteMany

resources:

requests:

storage: 3Gi

Endpoints

apiVersion: v1

kind: Endpoints

metadata:

name: glusterfs-cluster

subsets:

- addresses:

- ip: 192.168.0.11

ports:

- port: 1

- addresses:

- ip: 192.168.0.12

ports:

- port: 1

- addresses:

- ip: 192.168.0.13

ports:

- port: 1

Service

apiVersion: v1

kind: Service

metadata:

name: glusterfs-cluster

spec:

ports:

- port: 1

1. **Configuração e instalação da aplicação no kubernetes.**

Para a aplicação, foi utilizada a última imagem disponibilizada no docker hub, a seguir será apresentado o yaml da aplicação do tipo *Deployment*.

apiVersion: apps/v1

kind: Deployment

metadata:

name: moodle

labels:

app: moodle

spec:

selector:

matchLabels:

app: moodle

strategy:

type: Recreate

template:

metadata:

labels:

app: moodle

spec:

containers:

- name: moodle

image: moodlehq/moodle-php-apache:7.4-buster

volumeMounts:

- mountPath: "/var/www/"

name: gluster-vol

readOnly: false

ports:

- containerPort: 80

resources:

requests:

memory: "256Mi"

cpu: "250m"

limits:

memory: "512Mi"

cpu: "500m"

volumes:

- name: gluster-vol

persistentVolumeClaim:

claimName: gluster-claim

Comando para criar o deployment da aplicação:

kubectl apply -f moodle-deployment.yaml

Após a criação do deployment, será necessário baixar o código do moodle e mover para a pasta do cluster do *GlusterFS*, onde será mapeado para a pasta “/var/www/” da aplicação. Sugestão para o procedimento:

Obtendo a imagem por meio do git:

git clone https://github.com/moodle/moodle.git

Para mover os arquivos para dentro do cluster do *GlusterFS*, execute os seguintes comandos:

sudo mount -t glusterfs master1:/gv0 /mnt

E então, mover a pasta “moodle” que acabou de ser clonada do *git* para a pasta “/mnt”:

mv moodle /mnt/

Para a instalação do moodle, é necessário acessar a aplicação por meio do navegador, neste trabalho foi necessário inserir um registro no hots do Windows, para que possa ser possível acessar a aplicação pelo DNS “master1/moodle”. Mas antes de iniciar a instalação, será necessário alterar o dono do arquivo moodle apenas uma vez acessando o POD. Execute os seguintes passos para que isso seja possível:

kubectl apply -f moodle-deployment.yaml

Após iniciar a aplicação, descubra o nome do container pelo comando:

kubectl get pods

Acessando o container, criando um arquivo “data” exigido pela aplicação e alterando o dono da pasta “moodle” e do arquivo “moodledata”:

host $ Kubectl exec -it <nome-container> -- bash

pod # mkdir /var/www/html/moodledata

pod # chown www-data.www-data /var/www/html/moodle –R

pod # chown www-data.www-data /var/www/html/ moodledata -R

Após isso, é só prosseguir para a instalação por meio do navegador no endereço “http://master1/moodle”.

Exposição da aplicação na porta 80 através do IP dos três hosts do tipo "LoadBalancer". Yaml da do serviço:

apiVersion: v1

kind: Service

metadata:

name: frontend

labels:

app: moodle

spec:

type: LoadBalancer

externalIPs:

- 192.168.0.11

- 192.168.0.12

- 192.168.0.13

ports:

- port: 80

selector:

app: moodle

Após expor a aplicação, será necessário um yaml para escalonar a aplicação baseado no uso do processamento e/ou de memória. Então como apresentado no *yaml* a seguir, foi definido que o *deployment* irá iniciar com 2 *pods* e conforme a necessidade ele cria automaticamente até 5. Neste caso, as métricas definidas indicam que a partir de 70% de uso ou de memória ou de processador, o *kubernetes* realiza o escalonamento.

apiVersion: autoscaling/v2beta2

kind: HorizontalPodAutoscaler

metadata:

name: moodle-hpa

spec:

maxReplicas: 5

minReplicas: 2

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: moodle

metrics:

- type: Resource

resource:

name: cpu

target:

type: Utilization

averageUtilization: 70

- type: Resource

resource:

name: memory

target:

type: Utilization

averageUtilization: 70

Para que o "HorizontalPodAutoscaler" possa funcionar corretamente, é necessário adicionar o "*Metrics Server*" ao cluster, com isso é possível obter as informações de utilização de processador e memória através do comando "kubectl top nodes/pods". A seguir o yaml para o *metrics*.

---

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

name: system:aggregated-metrics-reader

labels:

rbac.authorization.k8s.io/aggregate-to-view: "true"

rbac.authorization.k8s.io/aggregate-to-edit: "true"

rbac.authorization.k8s.io/aggregate-to-admin: "true"

rules:

- apiGroups: ["metrics.k8s.io"]

resources: ["pods", "nodes"]

verbs: ["get", "list", "watch"]

---

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: metrics-server:system:auth-delegator

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: system:auth-delegator

subjects:

- kind: ServiceAccount

name: metrics-server

namespace: kube-system

---

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: metrics-server-auth-reader

namespace: kube-system

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: Role

name: extension-apiserver-authentication-reader

subjects:

- kind: ServiceAccount

name: metrics-server

namespace: kube-system

---

apiVersion: apiregistration.k8s.io/v1beta1

kind: APIService

metadata:

name: v1beta1.metrics.k8s.io

spec:

service:

name: metrics-server

namespace: kube-system

group: metrics.k8s.io

version: v1beta1

insecureSkipTLSVerify: true

groupPriorityMinimum: 100

versionPriority: 100

---

apiVersion: v1

kind: ServiceAccount

metadata:

name: metrics-server

namespace: kube-system

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: metrics-server

namespace: kube-system

labels:

k8s-app: metrics-server

spec:

selector:

matchLabels:

k8s-app: metrics-server

template:

metadata:

name: metrics-server

labels:

k8s-app: metrics-server

spec:

serviceAccountName: metrics-server

volumes:

# mount in tmp so we can safely use from-scratch images and/or read-only containers

- name: tmp-dir

emptyDir: {}

containers:

- name: metrics-server

image: k8s.gcr.io/metrics-server/metrics-server:v0.3.7

imagePullPolicy: IfNotPresent

args:

- --cert-dir=/tmp

- --secure-port=4443

- --kubelet-insecure-tls

- --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname

ports:

- name: main-port

containerPort: 4443

protocol: TCP

securityContext:

readOnlyRootFilesystem: true

runAsNonRoot: true

runAsUser: 1000

volumeMounts:

- name: tmp-dir

mountPath: /tmp

nodeSelector:

kubernetes.io/os: linux

---

apiVersion: v1

kind: Service

metadata:

name: metrics-server

namespace: kube-system

labels:

kubernetes.io/name: "Metrics-server"

kubernetes.io/cluster-service: "true"

spec:

selector:

k8s-app: metrics-server

ports:

- port: 443

protocol: TCP

targetPort: main-port

---

---

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

name: system:metrics-server

rules:

- apiGroups:

- ""

resources:

- pods

- nodes

- nodes/stats

- namespaces

- configmaps

verbs:

- get

- list

- watch

---

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: system:metrics-server

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: system:metrics-server

subjects:

- kind: ServiceAccount

name: metrics-server

namespace: kube-system