ДЗ: Подходы к развертыванию

Подготовка

В этом ДЗ через kubeadm мы поднимем кластер версии 1.23 и обновим его

Создание нод для кластера

- В YC создайте 4 ноды с образом Ubuntu 20.04 LTS:
  - master - 1 экземпляр (intel ice lake, 2vCPU, 8 GB RAM)
  - worker - 3 экземпляра (intel ice lake, 2vCPU, 8 GB RAM)

Подготовка машин

Отключите на машинах swap
sudo sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab

sudo swapoff -a

Включаем маршрутизацию

```
cat > /etc/sysctl.d/99-kubernetes-cri.conf <<eof net.bridge.bridge-nf-call-
iptables="1" net.ipv4.ip_forward="1" net.bridge.bridge-nf-call-ip6tables="1" eof=""
sysctl="" --system="" <="" code="">></eof>
```

Загрузим br_netfilter и позволим iptables видеть трафик.

```
sudo modprobe overlay

sudo modprobe br_netfilter

sudo tee /etc/sysctl.d/kubernetes.conf<<EOF

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

net.ipv4.ip_forward = 1

EOF

sysctl --system
```

Установим Containerd

```
cat <<EOF | sudo tee /etc/modules-load.d/containerd.conf
overlay
br_netfilter
EOF
sudo modprobe overlay
sudo modprobe br_netfilter
# Setup required sysctl params, these persist across reboots.
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
# Apply sysctl params without reboot
sudo sysctl --system
#Install and configure containerd
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
sudo apt update -y
sudo apt install -y containerd.io
sudo mkdir -p /etc/containerd
containerd config default | sudo tee /etc/containerd/config.toml
#Start containerd
sudo systemctl restart containerd
sudo systemctl enable containerd
```

Установим kubectl, kubeadm, kubelet

Установим версию 1.23, данные команды необходимо выполнить на всех нодах.

```
apt-get update && apt-get install -y apt-transport-https curl


curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list
sudo apt update -y
sudo apt -y install vim git curl wget kubelet=1.23.0-00 kubeadm=1.23.0-00 kubectl=1.23.0-00
sudo apt-mark hold kubelet kubeadm kubectl
sudo kubeadm config images pull --cri-socket /run/containerd/containerd.sock --kubernetes-version v1.23.0
```

Создание кластера

Создадим настроим мастер ноду при помощи kubeadm, для этого на ней выполним:

```
sudo kubeadm init   --pod-network-cidr=10.244.0.0/16   --upload-certs --kubernetes-
version=v1.23.0 --ignore-preflight-errors=Mem  --cri-socket /run/containerd/containerd.sock
```

В выводе будут:

- команда для копирования конфига `kubectl`
- сообщение о том, что необходимо установить сетевой плагин
- команда для присоединения worker ноды

Копируем конфиг kubectl

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

Проверяем:

```
kubectl get nodes
```

Установим сетевой плагин

После инициализации кластера `kubeadm` требуется сетевой плагин для сетевой связанности между подами.

Документация

.

В этом ДЗ в качестве примера мы установим Flannel, Вы можете установить, любой другой.

```
kubectl apply -f https://github.com/coreos/flannel/raw/master/Documentation/kube-flannel.yml
```

Подключаем worker-ноды

Установите на worker ноды docker, включите маргрутизацию, выключите swap, установите kubeadm, kubelet, kubectl и выполните `kubeadm join` на worker нодах.

```
# Пример команды для присоеденения к кластеру
kubeadm join --token <token> <master-ip>:<master-port> --discovery-token-ca-cert-
hash sha256:<hash></hash></master-port></master-ip></token>
```

Если вывод команды потерялся, токены можно посмотреть командой

```
kubeadm token list
```

Получить хеш

```
openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -pubin -outform
der 2>/dev/null | \
   openssl dgst -sha256 -hex | sed 's/^.* //'
```

Просмотр нод кластера

Таким образом, мы подключили ноды к кластеру Псмотрим, что у нас получилось командой `kubectl get nodes`.

Должно получиться что-то подобное:

```
NAME                STATUS   ROLES    AGE     VERSION
master-instance-1   Ready    master   4h33m   v1.23.0
worker-instance-1   Ready    <none>   4h31m   v1.23.0
worker-instance-2   Ready    <none>   4h23m   v1.23.0
worker-instance-3   Ready    <none>   4h20m   v1.23.0</none></none></none>
```

Запуск нагрузки

Для демонстрации работы кластера запустим nginx

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 4
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.17.2
        ports:
        - containerPort: 80
```

```
kubectl apply -f deployment.yaml
```

Обновление кластера

Так как кластер мы разворачивали с помощью kubeadm, то и производить обновление будем с помощью него.

Обновлять ноды будем по очереди.

Обновление мастера

Допускается, отставание версий worker-нод от master, но не наоборот. Поэтому обновление будем начинать с нее master-нода у нас версии 1.23.0

Обновление пакетов

```
apt update
apt-cache madison kubeadm
apt-mark unhold kubeadm && \
apt-get update && apt-get install -y kubeadm=1.24.x-00
&& \
apt-mark hold kubeadm
kubeadm upgrade plan
sudo kubeadm upgrade apply v1.24.x
```

Проверка

`kubectl get nodes` все ноды должны быть готовы, какая версия у мастер ноды? Почему? Какая версия у Api сервера, какая у kubelet?

Пример вывода:

```
NAME                STATUS   ROLES    AGE     VERSION
master-1            Ready    master   8m5s    v1.24.0
worker-instance-1   Ready    <none>   4m54s   v1.23.0
worker-instance-2   Ready    <none>   4m51s   v1.23.0
worker-instance-3   Ready    <none>   4m49s   v1.23.0</none></none></none>
```

Обновим остальные компоненты кластера

Обновление компонентов кластера (API-server, kube-proxy, controller-manager)

```
# просмотр изменений, которые собирает сделать kubeadm
kubeadm upgrade plan
```

```
# применение изменений
kubeadm upgrade apply
```

Проверка

```
kubeadm version
kubelet --version
kubectl version
kubectl describe pod <Ваш под с API сервером> -n kube-system
```

Вывод worker-нод из планирования

Первым делом, мы сливаем всю нагрузку с ноды и выводим ее из планирования:

```
kubectl drain worker-instance-1
```

```
node/worker-instance-1 cordoned
error: unable to drain node "worker-instance-1", aborting command...

There are pending nodes to be drained:
 worker-instance-1
error: cannot delete DaemonSet-managed Pods (use --ignore-daemonsets to ignore):
kube-system/calico-node-8sm6s, kube-system/kube-proxy-q97k7
```

Вывод worker-нод из планирования

`kubectl drain` убирает всю нагрузку, кроме DaemonSet, поэтому мы явно должны сказать, что уведомлены об этом

```
kubectl drain worker-instance-1 --ignore-daemonsets
```

kubectl drain возвращает управление только тогда, когда все поды выведены с ноды

```
node/worker-instance-1 already cordoned
WARNING: ignoring DaemonSet-managed Pods: kube-system/calico-node-8sm6s, kube-
system/kube-proxy-q97k7
evicting pod "nginx-deployment-6dd86d77d-m95f6"
evicting pod "nginx-deployment-6dd86d77d-7cwvr"
pod/nginx-deployment-6dd86d77d-m95f6 evicted
pod/nginx-deployment-6dd86d77d-7cwvr evicted
node/worker-instance-1 evicted
```

Обновление статуса worker-нод

Когда мы вывели ноду на обслуживание, к статусу добавилась строчка `SchedulingDisabled`

```
kubectl get nodes -o wide
```

```
NAME                STATUS                     ROLES    AGE    VERSION  INTERNAL-IP
EXTERNAL-IP   OS-IMAGE                       KERNEL-VERSION
master-instance-1   Ready                      master   107m   v1.24.0  10.128.0.8
<none>        Debian GNU/Linux 9 (stretch)   4.9.0-9-amd64
worker-instance-1   Ready,SchedulingDisabled   <none>   105m   v1.23.0  10.128.0.6
<none>        Debian GNU/Linux 9 (stretch)   4.9.0-9-amd64
worker-instance-2   Ready                      <none>   101m   v1.23.0  10.128.0.7
<none>        Debian GNU/Linux 9 (stretch)   4.9.0-9-amd64
</none></none></none></none>
```

Обновление worker-нод

На worker-ноде выполняем

```
apt-mark unhold kubeadm && \
apt-get update && apt-get install -y kubeadm=1.24.x-00
&& \
apt-mark hold kubeadm
sudo kubeadm upgrade node
apt-mark unhold kubelet kubectl && \
apt-get update && apt-get install -y kubelet=1.24.x-00
kubectl=1.24.x-00 && \
apt-mark hold kubelet kubectl
sudo systemctl daemon-reload
sudo systemctl restart kubelet
```

Просмотр обновления

После обновления `kubectl` показывает новую версию, и статус `SchedulingDisabled`

```
kubectl get nodes
NAME                STATUS                        ROLES    AGE    VERSION
master-instance-1   Ready                         master   128m   v1.24.0
worker-instance-1   Ready,SchedulingDisabled      <none>   126m   v1.24.0
worker-instance-2   Ready                         <none>   122m   v1.23.0
worker-instance-3   Ready                         <none>   122m   v1.23.0</none></none>
</none>
```

Возвращение ноды в планирование

Командой `kubectl uncordon worker-instance-1` возвращаем ноду обратно в планирование нагрузки

Задание

Мы обновили одну из двух нод.

Задание:

- обновите оставшиеся ноды при помощи kubeadm

Автоматическое развертывание кластеров

В данном задании ради демонстрации механики обновления мы вручную развернули и обновили кластер с одной master-нодой.

Но развертывать большие кластера подобным способом не удобно. Поэтому мы рассмотрим инструмент для автоматического развертывания кластеров kubespray.

Kubespray - это Ansible playbook для установки Kubernetes.

Для его использования достаточно иметь SSH-доступ на машины, поэтому не важно как они были созданы (Cloud, Bare metal).

Установка kubespray

Пре-реквизиты:

- Python и pip на локальной машине
- SSH доступ на все ноды кластера

```
# получение kubespray
git clone https://github.com/kubernetes-sigs/kubespray.git
# установка зависимостей
sudo pip install -r requirements.txt
# копирование примера конфига в отдельную директорию
cp -rfp inventory/sample inventory/mycluster
```

Добавьте адреса машин кластера в конфиг kubespray `inventory/mycluster/inventory.ini`:

```
# в блоке all мы описывем все машины (master и worker)
# для мастер нод мы указывем переменную etcd_member_name
[all]
node1 ansible_host=95.54.0.12 etcd_member_name=etcd1
node2 ansible_host=95.54.0.13
node3 ansible_host=95.54.0.14
node4 ansible_host=95.54.0.15

# в блоке kube-master мы указывем master-ноды
[kube-master]
node1

# в блоке etcd ноды, где будет установлен etcd
# если мы хотим HA кластер, то etcd устанавливается отдельно от API-server
[etcd]
node1

# в блоке kube-node описываем worker-ноды
[kube-node]
node2
node3
node4

# в блоке k8s-cluster:children соединяем kube-master и kube-node
[k8s-cluster:children]
kube-master
kube-node
```

Установка кластера

После редактирования конфига можно устанавливать кластер:

```
ansible-playbook -i inventory/mycluster/inventory.ini --become --become-user=root \
--user=${SSH_USERNAME} --key-file=${SSH_PRIVATE_KEY} cluster.yml
```

Задание со 

Выполните установку кластера с 3 master-нодами и 2 worker-нодами, можно использовать kubeadm или любой другой способ установки kubernetes.

В README.MD и в описании PR покажите вывод команды `kubectl get nodes` и опишите выбранный способ установки.