k8s配置学习记录

环境配置

k8s版本：1.25.3 机器操作系统：ubuntu 22.04

- 配置1.25.3版本的k8s，与1.23及以前版本的一个很大不同是，1.25以后的k8默认支持containerd，不再支持docker，需要cri-docker中间件才能正常使用docker作为容器。

参考：

https://blog.csdn.net/yzh0905_/article/details/122978696

https://blog.csdn.net/SHELLCODE_8BIT/article/details/122192034

关闭swap分区

```
sed -i '/fstab/d' /etc/fstab
```

可以通过`free`查看是否已经关闭了swap分区

设置主机名

```
hostnamectl set-hostname master
hostnamectl set-hostname node1
bash
```

机器 公网IP 内网IP 主机名 机器1 47.100.177.151 172.29.239.20 master 机器2 101.132.186.239 172.29.239.21 node1

加入每个节点的ip信息

修改`/etc/hosts`文件，在最后加入集群IP信息，内网外网都行，只要能够互相ping通。

```
172.29.239.21   node1
172.29.239.20   master
```

软件安装

更新docker

```
apt-get remove docker docker-engine docker.io containerd runc
apt-get update
apt-get install ca-certificates curl gnupg lsb-release
mkdir -p /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg

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

apt-get update
apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

然后需要将docker的cgroup驱动类型改成systemd，只要在/etc/docker/daemon.json中加入以下内容：

```
{
  "registry-mirrors": [
    "https://registry.docker-cn.com"
  ],
  "exec-opts": [ "native.cgroupdriver=systemd" ]
}
```

运行以下命令重启，通过`docker info | grep Cgroup`来查看修改后的 docker cgroup 状态，发现变为systemd即为修改成功。

```
sudo systemctl daemon-reload
sudo systemctl restart docker
```

安装cri-docker

从Mirantis/cri-dockerd下载最新的cri-docker release

```
wget https://github.com/Mirantis/cri-dockerd/releases/download/v0.2.6/cri-
dockerd_0.2.6.3-0.ubuntu-jammy_amd64.deb
```

然后用dpkg安装cri-docker

```
dpkg -i cri-dockerd_0.2.6.3-0.ubuntu-jammy_amd64.deb
```

然后需要修改它的启动命令，让其使用cni插件，以及指定pause镜像

```
sed -i '/ExecStart/s/$/ --network-plugin=cni --pod-infra-container-
image=registry.aliyuncs.com\/google_containers\/pause:3.8/'
/usr/lib/systemd/system/cri-docker.service

# 然后重启cri
systemctl daemon-reload
systemctl restart cri-docker.service
systemctl status cri-docker.service
```

安装k8s（kubelet kubeadm kubectl）

```
curl https://mirrors.aliyun.com/kubernetes/apt/doc/apt-key.gpg | sudo apt-key add
-
add-apt-repository "deb https://mirrors.aliyun.com/kubernetes/apt/ kubernetes-
xenial main"
apt update
# 默认最新版本，可以通过版本号指定软件版本，`apt-cache madison kubeadm`可以查看软件版本
apt-get install -y kubelet=1.25.3-00 kubeadm=1.25.3-00 kubectl=1.25.3-00
```

`apt-cache madison kubeadm`可以查看软件版本

配置节点

初始化master节点

- 只需要在master节点上配置

```
kubeadm init \
  --apiserver-advertise-address=172.29.239.20 \
  --image-repository registry.aliyuncs.com/google_containers \
  --kubernetes-version v1.25.3 \
  --service-cidr=10.96.0.0/12 \
  --pod-network-cidr=10.244.0.0/16 \
  --ignore-preflight-errors=all \
  --cri-socket unix:///var/run/cri-dockerd.sock
```

常用参数定义：

- apiserver-advertise-address: k8s 中的主要服务apiserver的部署地址，填自己的管理节点 ip
- image-repository: 拉取的 docker 镜像源，因为初始化的时候kubeadm会去拉 k8s 的很多组件来进行部署，所以需要指定国内镜像源，下不然会拉取不到镜像。
- pod-network-cidr: 这个是 k8s 采用的节点网络，因为我们将要使用flannel作为 k8s 的网络，所以这里填10.244.0.0/16就好

- kubernetes-version: 这个是用来指定你要部署的 k8s 版本的，一般不用填，不过如果初始化过程中出现了因为版本不对导致的安装错误的话，可以用这个参数手动指定。
- ignore-preflight-errors: 忽略初始化时遇到的错误，比如说我想忽略 cpu 数量不够 2 核引起的错误，就可以用--ignore-preflight-errors=CpuNum。错误名称在初始化错误时会给出来。

如果说某次执行kubeadm init初始化k8s集群失败了，在下一次执行kubeadm init初始化语句之前，先执行kubeadm reset命令。这个命令的作用是重置节点，可以把这个命令理解为：上一次kubeadm init初始化集群操作失败了，该命令清理了之前的失败环境。会删去/etc/kubernetes/*.conf内容

问题：

1. Found multiple CRI endpoints on the host. Please define which one do you wish to use by setting the 'criSocket' field in the kubeadm configuration file: unix:///var/run/containerd/containerd.sock, unix:///var/run/cri-dockerd.sock To see the stack trace of this error execute with --v=5 or higher 解决：kubernetes1.24版本，添加--cri-socket选项指定一个CRI，这里指定cri-dockered
2. [wait-control-plane] Waiting for the kubelet to boot up the control plane as static Pods from directory "/etc/kubernetes/manifests". This can take up to 4m0s [kubelet-check] Initial timeout of 40s passed. 解决：apiserver-advertise-address里ip地址输错了

返回信息：

```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

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

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as
root:

kubeadm join 172.29.239.20:6443 --token 0npvkb.r9petu7nnlezdkmk \
        --discovery-token-ca-cert-hash
sha256:60417b84c7ccc4505488223a11da61e0ecbc50ca901c930f0d8b8d7eaad90896
```

根据要求拷贝认证文件：

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

```
sudo chown $(id -u):$(id -g) $HOME/.kube/config
# or if root:
export KUBECONFIG=/etc/kubernetes/admin.conf
```

工作节点加入集群

```
kubeadm join 172.29.239.20:6443 --token 0npvkb.r9petu7nnlezdkmk \
        --discovery-token-ca-cert-hash
sha256:60417b84c7ccc4505488223a11da61e0ecbc50ca901c930f0d8b8d7eaad90896 --cri-
socket unix:///var/run/cri-dockerd.sock
```

在master节点run`kubectl get nodes`可以看到当前状态

部署pod network

这里使用flannel

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

把master节点也转换为工作节点

k8s默认master节点只做调度工作，如果也想要作为node使用，需要额外执行以下命令使其作为工作节点：

```
kubectl taint nodes --all node-role.kubernetes.io/control-plane-
```

- 旧`node-role.kubernetes.io/master`标签和污点键已被弃用，将替换为`node-role.kubernetes.io/control-plane`，它们在过渡期内都有效。
- `kubectl describe nodes master | grep Taint` 确认污点状态被更改
- 恢复成 Master Only 状态:

  ```
  kubectl taint node --all node-role.kubernetes.io/control-plane="":NoSchedule
  ```

K8基本使用

参考： https://www.jianshu.com/p/8d60ce1587e1 https://www.jianshu.com/p/116ce601a60f

基本概念

k8s 是经典的一对多模型，有一个主要的管理节点master和许多的工作节点slaver。当然，k8s 也可以配置多个管理节点，拥有两个以上的管理节点被称为 高可用。k8s 包括了许多的组件，每个组件都是单运行在一个

docker容器中，然后通过自己规划的虚拟网络相互访问。

k8s把数量众多的服务器重新抽象为一个统一的资源池，增加新的服务器对运维人员来说，只是增加自资源池的可用量。不仅如此，k8s 把所有能用的东西都抽象成了资源的概念，从而提供了一套更统一，更简洁的管理方式。

什么是pod? pod的概念其实和docker中的容器非常相似。他是 k8s中的最小工作单位。可以把 pod 理解成一个一个的小机器人，而 k8s抽象出来的大资源池就是他们的工厂。在使用过程中，pod将一个或多个docker容器封装成一个统一的整体进行管理并对外提供服务。

在我们的项目中为什么需要K8S

横向的扩展，一个csv文件对应一个任务，可以同时起多个任务，由k8做资源调配。

资源类型

- Pod： Pod是一个或多个容器的组合，这些容器共享存储、网络和命名空间，以及如何运行的规范。
- ReplicaSet： ReplicaSet是Replication Controller升级版。Replication Controller的作用是确保Pod以指定的副本个数运行。 ReplicaSet和Replication Controller之间的唯一区别是对选择器支持。Replication Controller只支持基于等式的selector（env=dev或environment!=qa），但ReplicaSet还支持新的，基于集合的selector（version in (v1.0,v2.0)或env notin (dev, qa)）。
- Deployment： Deployment用于管理Pod、ReplicaSet，可实现滚动升级和回滚应用、扩容和缩容。
- Service k8s的Service定义了一个服务的访问入口地址，前端的应用通过这个入口地址访问其背后的一组由Pod副本组成的集群实例，来自外部的访问请求被负载均衡到后端的各个容器应用上。Service与其后端Pod副本集群之间则是通过Label Selector实现关联。

基本操作

1. kubectl get列出资源

例如： `kubectl get pod -n kube-system`**查看k8s所有pod** // `-n`参数指定了要查看哪个命名空间下的pod。例如kube-system命名空间下有k8s运行本身需要的所有pod 。

每列意思：

- NAME：第一列是 pod 的名字，k8s 可以为 pod 随机分配一个五位数的后缀。
- READY：第二列是 pod 中已经就绪的 docker 容器的数量，上文中我们提到了，pod 封装了一个或多个 docker 容器。在这里，1/1的含义为就绪1个容器/共计1个容器。
- STATUS：第三列是 pod 的当前状态，下面是一些常见的状态： 状态名 含义 Running 运行中 Error 异常，无法提供服务 Pending 准备中，暂时无法提供服务 Terminaling 结束中，即将被移除 Unknown 未知状态，多发生于节点宕机 PullImageBackOff 镜像拉取失败
- RESTART：k8s 可以自动重启 pod，这一行就是标记了 pod 一共重启了多少次。

2. kubectl describe 查看某一资源的具体信息

`kubectl describe pod <pod名> -n <命名空间>` 例如： `kubectl describe pod kube-scheduler-master -n kube-system` 包括基本信息（运行状态，所在节点etc），内部镜像信息（当 pod 出现ImagePullBackOff错误的时候就可以查看该字段确认拉取的什么镜像），**事件**（当 pod 的状态不是Running时，从这里可以看到出现问题详细原因）

3. kubectl logs 查看日志

`kubectl logs <pod名>` 通过添加`-f`参数可以持续查看日志 例如： `kubectl logs -f -n kube-system kube-proxy-65zlj`

4. kubectl create 创建资源

详见下一节部署应用

5. kubectl delete 删除

`kubectl delete <资源类型> <资源名>`

部署应用

打包docker镜像

这里打包的是一个简单的带读取文件数据的C++程序

参考： https://blog.csdn.net/dreamflyly/article/details/128163267

https://www.yingsoo.com/news/servers/45607.html

1. 下载需要用到的基础镜像 docker自带的ubuntu镜像没有gcc和g++，需要通过`docker search`寻找所需镜像自行下载。这里： `docker pull codenvy/cpp_gcc`
2. 在代码同目录下编写dockerfile

```
FROM codenvy/cpp_gcc
RUN mkdir /home/user/hello_world
ADD hello_world.cpp /home/user/hello_world
WORKDIR /home/user/hello_world
RUN g++ hello_world.cpp -o hello
CMD ["./hello"]
```

FROM:该目标镜像使用的基础镜像，基础镜像有且只能有一个 **RUN:容器构建时需要的命令，** 后面带有的就是我们常见的linux命令。 比如这里： `RUN mkdir /home/user/hello_world`该命令就是在基础镜像的/home/user 路径下新建一个 hello_world 文件夹 `RUN g++ hello_world.cpp -o hello`就是使用g++编译，生成执行文件 **ADD**:把宿主机下的文件拷贝到镜像，ADD相比COPY带有解压缩功能 **WORKDIR**:指定默认工作目录 **CMD：指定容器创建完成后第一个运行的命令。** 这里就是运行可执行文件

3. 创建镜像

`docker build -f ./hello-dockerfile -t my_hello_docker:v1.0 .`

部署docker

1. 用yaml文件部署应用

参考： https://blog.csdn.net/dreamflyly/article/details/128063318

`kubectl create -f hello.yaml`

2. 通过命令直接创建

参考： https://blog.csdn.net/dreamflyly/article/details/128061177

k8s 为一些常用的资源提供了简易创建的方法，比如说service、namespace、deployment等，这些方法可以使用`kubectl create <资源类型> <资源名>`的方式创建。 这里执行： `kubectl create deploy my-hello --image=my_hello_docker:v1.0`

- 使用`kubectl get deploy`可以查看创建是否成功
- 使用`kubectl create -h`命令查看哪些资源可以快速生成

k8s中的数据持久化

参考：

https://www.cnblogs.com/zjq-blogs/p/14067656.html

https://blog.csdn.net/weixin_44024436/article/details/122545092

https://blog.51cto.com/u_14306186/2522109

> K8S启动的pod属于容器，当pod删除或者重建，原容器中的数据将丢失，这将是严重的安全事故。 在 Docker 中就有数据卷的概念，当容器删除时，数据也一起会被删除，想要持久化使用数据，需要把主机上的目录挂载到 Docker 中去，在 K8S 中，数据卷是通过 Pod 实现持久化的，如果 Pod 删除，数据卷也会一起删除，k8s 的数据卷是 docker 数据卷的扩展，K8S 适配各种存储系统，包括本地存储 EmptyDir，HostPath， 网络存储（NFS，GlusterFS，PV/PVC）等。

在我们的项目中，读取原数据和最终结果的存储都需要用到数据持久化。这里把机器2上的/Data目录作为挂载目录，也就是持久化存储的目录。

持久化配置的流程是：配置NFS共享文件系统，保证每台机器都能访问同一个目录 -> 在k8中配置PV，以这个目录作为一个持久卷的根节点 -> 在k8中配置PVC，创建一个对持久卷的消费者，和PV绑定 -> pod声明时指定需要的PVC

配置NFS共享文件系统

安装NFS服务端

安装：

```
apt-get install -y nfs-kernel-server
```

给要共享的文件目录增加读写权限：

```
chmod a+rw /root/Data
```

配置NFS服务目录

打开文件 `vi /etc/exports`，在尾部新增一行，内容如下

```
/root/Data *(rw,sync,no_subtree_check)
```

- /usr/local/kubernetes/volumes：作为服务目录向客户端开放
- *：表示任何 IP 都可以访问
- rw：读写权限
- sync：同步权限
- no_subtree_check：表示如果输出目录是一个子目录，NFS 服务器不检查其父目录的权限

`/etc/init.d/nfs-kernel-server restart`重启服务，让配置生效。

安装NFS客户端

//主要只是为了验证文件是否可以正常上传

```
apt-get install -y nfs-common
```

创建 NFS 客户端挂载目录：

```
mkdir -p /root/nfs
```

将 NFS 服务器的目录挂载到 NFS 客户端的目录：

```
mount 172.29.239.21:/root/Data /root/nfs
```

这里nfs服务器设置在machine2上 使用 `df` 命令查看挂载信息 测试文件上传：

```
ip addr > /root/zyy/nfs_try/try.txt
```

取消挂载：

```
umount /root/zyy/nfs_try
```

k8s配置PV\PVC

> **Persistent Volume（PV）** 是集群之中的一块网络存储。跟 Node 一样，也是集群的资源。PV 跟 Volume (卷) 类似，不过会有独立于 Pod 的生命周期。这一 API 对象包含了存储的实现细节，例如 NFS、iSCSI 或者其他的云提供商的存储系统。

> **Persistent Volume Claim (PVC)** 是用户的一个请求。跟 Pod 类似，Pod 消费 Node 的资源，PVC 消费 PV 的资源。Pod 能够申请特定的资源（CPU 和内存）；Claim 能够请求特定的尺寸和访问模式（例如可以加载一个读写，以及多个只读实例）。

定义PV

这里持久卷实现的方式是NFS yaml配置文件：

```
apiVersion: v1
   kind: PersistentVolume
   metadata:
     name: nfs-pv-mysql
   spec:
     # 设置容量
     capacity:
       storage: 5Gi
     # 访问模式
     accessModes:
       # 该卷能够以读写模式被多个节点同时加载
       - ReadWriteMany
     # 回收策略，这里是基础擦除 `rm-rf/thevolume/*`
     persistentVolumeReclaimPolicy: Recycle
     nfs:
       # NFS 服务端配置的路径
       path: "/root/Data"
       # NFS 服务端地址
       server: 172.29.239.21
       readOnly: false
```

部署：

```
kubectl create -f /root/zyy/nfs_try/nfs-pv.yml
```

删除：

```
kubectl delete -f /root/zyy/nfs_try/nfs-pv.yml
```

定义PVC

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-pvc-orderbook
spec:
  accessModes:
```

```
  # 需要使用和 PV 一致的访问模式
  - ReadWriteMany
  # 按需分配资源
  resources:
      requests:
        storage: 5Gi
```

部署:

```
kubectl create -f /root/zyy/nfs_try/nfs-pvc.yml
```

在创建pod资源时声明共享卷

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: hello-nfs
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hello-nfs
  template:
    metadata:
      labels:
        app: hello-nfs
    spec:
      containers:
      - name: my-hello-program
        image: my_hello_docker:v3.0
        volumeMounts:
        - mountPath: "/nfs"    #容器的挂载点,也就是在容器里访问PV的路径
          name: statics-datadir                          #被挂载卷的名称
      volumes:
      - name: statics-datadir  #共享存储卷名称,把下面的PVC声明一个卷叫做statics-
datadir, 再把这个卷挂载到上面的容器目录
        persistentVolumeClaim:   #类型是PVC
          claimName: nfs-pvc-orderbook  #指定要绑定的PVC, 前面已经创建好了
```

```
kubectl create -f hello_nfs.yaml
```

调试

进入pod调试代码

如果 容器镜像 包含调试程序， 比如从 Linux 和 Windows 操作系统基础镜像构建的镜像，可以使用 `kubectl exec` 命令 在特定的容器中运行一些命令： 比如运行一个连接到终端的shell：

```
kubectl exec -it hello-nfs-59dc6ff6c4-jxrxb -- sh
```

之后就可以在里面运行gdb调试可执行文件了

项目的大概思路：把读取部分，存储部分，计算部分放在不同docker里部署在一个pod中

未解决的问题： 如果组件之间用消息中间件进行通信，是不是只能起在一台机器上？ k8s在其中的作用好像只能是横向扩展？就是相互之间没有关联的任务可以一起做，但比如计算同一张表中的order book，这样的任务是没法通过k8s加速的。