K8s实践

使用kubeadm安装并部署k8s集群

> Q1：请记录所有安装步骤的指令，并简要描述其含义

我的安装流程如下：

1. 安装docker

   `curl -fsSL https://get.docker.com | bash -s docker --mirror Aliyun`

2. 安装kubeadm

   由于某些原因无法下载gpg文件，上网找到了如下指令，通过阿里云镜像源下载

   ```
   sudo apt-get update && sudo apt-get install -y apt-transport-https curl
   sudo curl -s https://mirrors.aliyun.com/kubernetes/apt/doc/apt-key.gpg |
   sudo apt-key add -
   sudo tee /etc/apt/sources.list.d/kubernetes.list <<-'EOF'
   deb https://mirrors.aliyun.com/kubernetes/apt kubernetes-xenial main
   EOF
   sudo apt-get update
   ```

   执行之后，就可以通过apt-get下载kubeadm了：

   ```
   sudo apt-get install -y kubeadm
   ```

3. 由于docker采用的是cgroupfs而k8s采用的是systemd，为了统一，将两者cgroup-driver均设置为systemd。具体做法我是通过查阅资料发现并参考了这篇文章：[Docker中的Cgroup Driver:Cgroupfs 与 Systemd]

4. 通过`kubeadm init`指令配置master node

   ```
   kubeadm init --image-
   repository='registry.cnhangzhou.aliyuncs.com/google_containers'
   ```

   设置环境变量：

   ```
   export KUBECONFIG=/etc/kubernetes/admin.conf
   ```

5. 安装CNI插件weave：

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

   k8s使用的coredns依赖于CNI插件，安装完weave之后，通过指令`kubectl get pod -n kube-system -o wide`能发现coredns正常运行

```
NAME                              READY   STATUS    RESTARTS   AGE   IP
      NODE     NOMINATED NODE   READINESS GATES
coredns-65c54cc984-nqlvs          1/1     Running   0          16m
10.32.0.2      master   <none>           <none>
coredns-65c54cc984-ww9lf          1/1     Running   0          16m
10.32.0.3      master   <none>           <none>
etcd-master                       1/1     Running   6          16m
192.168.1.12   master   <none>           <none>
kube-apiserver-master             1/1     Running   6          16m
192.168.1.12   master   <none>           <none>
kube-controller-manager-master    1/1     Running   13         16m
192.168.1.12   master   <none>           <none>
kube-proxy-6wjcv                  1/1     Running   0          11m
192.168.1.14   node1    <none>           <none>
kube-proxy-rccdg                  1/1     Running   0          16m
192.168.1.12   master   <none>           <none>
kube-scheduler-master             1/1     Running   14         16m
192.168.1.12   master   <none>           <none>
weave-net-vhtm2                   2/2     Running   0          33s
192.168.1.12   master   <none>           <none>
weave-net-zhqsw                   2/2     Running   0          33s
192.168.1.14   node1    <none>           <none>
```

6. 配置worker

   按照同样的方式下载docker和kubeadm，并使用kubeadm init执行后提供的`kubeadm join`指令：

```
kubeadm join 192.168.1.12:6443 --token 7tzbg7.znbnshrpm5vgsrsz \
--discovery-token-ca-cert-hash
sha256:ad15fde7f9efdbf2c9333e493ea074d6adf806a155262ab409a1be2c25d6662d
```

到目前为止，完成了基本的配置，通过`kubectl get nodes`发现节点均已处于Ready状态。

```
NAME     STATUS   ROLES                  AGE   VERSION
master   Ready    control-plane,master   22m   v1.23.6
node1    Ready    <none>                 16m   v1.23.6
```

一些问题

1. token默认是24小时过期，可以通过`kubeadm token list`查看token。若token过期，可以使用`kubeadm token create --print-join-command`生成token
2. 要保证master的`6443`端口没有被占用，并且允许外部访问。如果采用的是云主机，需要涉及到安全组的设置：

   配置之后，应该能通过`telnet master_host_ip 6443`连接。

> Q2 在两个节点上分别使用`ps aux | grep kube`列出所有和k8s相关的进程，记录其输出，并简要说明各个进程的作用

Master节点

`ps aux | grep kube`结果：

```
root        4127  1.1  0.6 11214780 51280 ?      Ssl  16:40   0:17 etcd --
advertise-client-urls=https://192.168.1.12:2379 --cert-
file=/etc/kubernetes/pki/etcd/server.crt --client-cert-auth=true --data-
dir=/var/lib/etcd --initial-advertise-peer-urls=https://192.168.1.12:2380 --
initial-cluster=master=https://192.168.1.12:2380 --key-
file=/etc/kubernetes/pki/etcd/server.key --listen-client-
urls=https://127.0.0.1:2379,https://192.168.1.12:2379 --listen-metrics-
urls=http://127.0.0.1:2381 --listen-peer-urls=https://192.168.1.12:2380 --
name=master --peer-cert-file=/etc/kubernetes/pki/etcd/peer.crt --peer-client-
cert-auth=true --peer-key-file=/etc/kubernetes/pki/etcd/peer.key --peer-trusted-
ca-file=/etc/kubernetes/pki/etcd/ca.crt --snapshot-count=10000 --trusted-ca-
file=/etc/kubernetes/pki/etcd/ca.crt
...
```

经过整理，得到下表：

由于理解有限，一些东西不知道如何准确表达，因而部分作用描述参考了网络资料。

进程	作用
etcd	存储各种元数据和状态。
kube-apiserver	控制面最主要的组件，负责和其他的组件进行交互，是控制面的中心点。
kube-scheduler	负责将创建的pod调度到某个节点上。
kube-controller-manager	作为集群内部的管理控制中心，负责集群内的Node，Pod副本，服务端点（endpoint），命名空间（namespace）等的管理。
kubelet	处理Master节点下发到本节点的任务，管理Pod和其中的容器。
kube-proxy	维护节点上的网络规则，实现了Kubernetes Service 概念的一部分 。它的作用是使发往 Service 的流量（通过ClusterIP和端口）负载均衡到正确的后端Pod。

Worker节点

worker节点不包含控制面的内容，所以只有`kubelet`和`kube-proxy`，具体作用如上所示。

> Q3: 在两个节点中分别使用`docker ps`显示所有正常运行的docker容器，记录其输出，并简要说明各个容器所包含的k8s组件，以及哪些k8s组件未运行在容器中

Master节点

调用`docker ps`之后获得的结果：

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	NAMES
2fe31c041f8d	4c0375452406	"/usr/local/bin/kube..."	45 minutes ago	Up 45 minutes	k8s_kube-proxy_kube-proxy-j85j9_kube-system_a187e252-97b0-45de-9207-e19cdd922422_0
a243ddce286d	registry.cn-hangzhou.aliyuncs.com/google_containers/pause:3.6	"/pause"	45 minutes ago	Up 45 minutes	k8s_POD_kube-proxy-j85j9_kube-system_a187e252-97b0-45de-9207-e19cdd922422_0
dd856da5c01a	df7b72818ad2	"kube-controller-man..."	45 minutes ago	Up 45 minutes	k8s_kube-controller-manager_kube-controller-manager-master_kube-system_27cf30a1c2a4a64564f85953423a4afb_5
e0f696107940	595f327f224a	"kube-scheduler --au..."	45 minutes ago	Up 45 minutes	k8s_kube-scheduler_kube-scheduler-master_kube-system_c8875b03a070436a9c6ca77dac5b9520_5
2314768bf1bf	8fa62c12256d	"kube-apiserver --ad..."	45 minutes ago	Up 45 minutes	k8s_kube-apiserver_kube-apiserver-master_kube-system_fef0d6d8c1ff3a71585c7ad5a30ab5d3_5
987b3b32b6a2	25f8c7f3da61	"etcd --advertise-cl..."	45 minutes ago	Up 45 minutes	k8s_etcd_etcd-master_kube-system_76a59c1cb93f43093d9910303f2ef924_5
b5ed2dacc331	registry.cn-hangzhou.aliyuncs.com/google_containers/pause:3.6	"/pause"	45 minutes ago	Up 45 minutes	k8s_POD_kube-scheduler-master_kube-system_c8875b03a070436a9c6ca77dac5b9520_0
9a4daa466d7c	registry.cn-hangzhou.aliyuncs.com/google_containers/pause:3.6	"/pause"	45 minutes ago	Up 45 minutes	k8s_POD_kube-controller-manager-master_kube-system_27cf30a1c2a4a64564f85953423a4afb_0
b18b393d984a	registry.cn-hangzhou.aliyuncs.com/google_containers/pause:3.6	"/pause"	45 minutes ago	Up 45 minutes	k8s_POD_kube-apiserver-master_kube-system_fef0d6d8c1ff3a71585c7ad5a30ab5d3_0
9b60c8bafd50	registry.cn-hangzhou.aliyuncs.com/google_containers/pause:3.6	"/pause"	45 minutes ago	Up 45 minutes	k8s_POD_etcd-master_kube-system_76a59c1cb93f43093d9910303f2ef924_0

容器包含了如下k8s组件：

- kube-proxy
- kube controller manager
- kube-scheduler
- kube-apiserver
- etcd

以及他们配套的infra容器(k8s默认为k8s.gcr.io/pause镜像)

Worker节点

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	NAMES
255bca2632d3	registry.cn-hangzhou.aliyuncs.com/google_containers/kube-proxy	"/usr/local/bin/kube..."	57 minutes ago	Up 57 minutes	k8s_kube-proxy_kube-proxy-d2lkd_kube-system_9b7ddf4c-2e00-4904-ab47-3d6ea10376b5_0
d9985bc654f3	registry.cn-hangzhou.aliyuncs.com/google_containers/pause:3.6	"/pause"	57 minutes ago	Up 57 minutes	k8s_POD_kube-proxy-d2lkd_kube-system_9b7ddf4c-2e00-4904-ab47-3d6ea10376b5_0

worker节点只有kube-proxy。

kubelet没有运行在容器里

部署Pod

> Q4: 请采用声明式接口对Pod进行部署，并将部署所需的yml文件记录在实践文档中

采用声明式部署方式：`kubectl apply -f ./pod.yaml`

该文件内容如下所示：

```
apiVersion: v1
kind: Pod
metadata:
```

```yaml
  name: pod
  labels:
    app: myApp
spec:
  restartPolicy: Always
  containers:
    - name: viewer
      image: dplsming/nginx-fileserver:1.0
      ports:
        - containerPort: 80
          hostPort: 8888
      volumeMounts:
        - name: volume
          mountPath: /usr/share/nginx/html/files
    - name: downloader
      image: dplsming/aria2ng-downloader:1.0
      ports:
        - containerPort: 6800
          hostPort: 6800
        - containerPort: 6880
          hostPort: 6880
      volumeMounts:
        - name: volume
          mountPath: /data
  volumes:
    - name: volume
      hostPath:
        path: /pod
```

> Q5: 请在worker节点上，在部署Pod的前后分别采用`docker ps`查看所有运行中的容器并对比两者的区别。请将创建该Pod所创建的全部新容器列举出来，并一一解释其作用

worker节点上多出了如下三个容器：

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	NAMES
a7fae79530a1	dplsming/aria2ng-downloader	"/conf-copy/start.sh"	About an hour ago	Up About an hour	k8s_downloader_pod_default_05260c21-c524-41c7-82e7-ea61f09f4f2f_0
c17c77e1b546	dplsming/nginx-fileserver	"/docker-entrypoint...."	About an hour ago	Up About an hour	k8s_viewer_pod_default_05260c21-c524-41c7-82e7-ea61f09f4f2f_0
23f2f68378cc	registry.cn-hangzhou.aliyuncs.com/google_containers/pause:3.6	"/pause"	About an hour ago	Up About an hour	k8s_POD_pod_default_05260c21-c524-41c7-82e7-ea61f09f4f2f_0

作用解释：

容器	作用
k8s_downloader_pod_default_05260c21-c524-41c7-82e7-ea61f09f4f2f_0	对应于pod.yaml中指定的downloader容器
k8s_viewer_pod_default_05260c21-c524-41c7-82e7-ea61f09f4f2f_0	对应于pod.yaml中指定的viewer容器
k8s_POD_pod_default_05260c21-c524-41c7-82e7-ea61f09f4f2f_0	infra容器，k8s默认采用pause镜像。该容器是一个轻量级的容器，用于为所有用户指定的pod中的容器提供网络命名空间。

对于pause容器，我们可以通过以下方式验证这一点：

使用指令`docker inspect k8s_viewer_pod_default_05260c21-c524-41c7-82e7-ea61f09f4f2f_0 | grep NetworkMode`

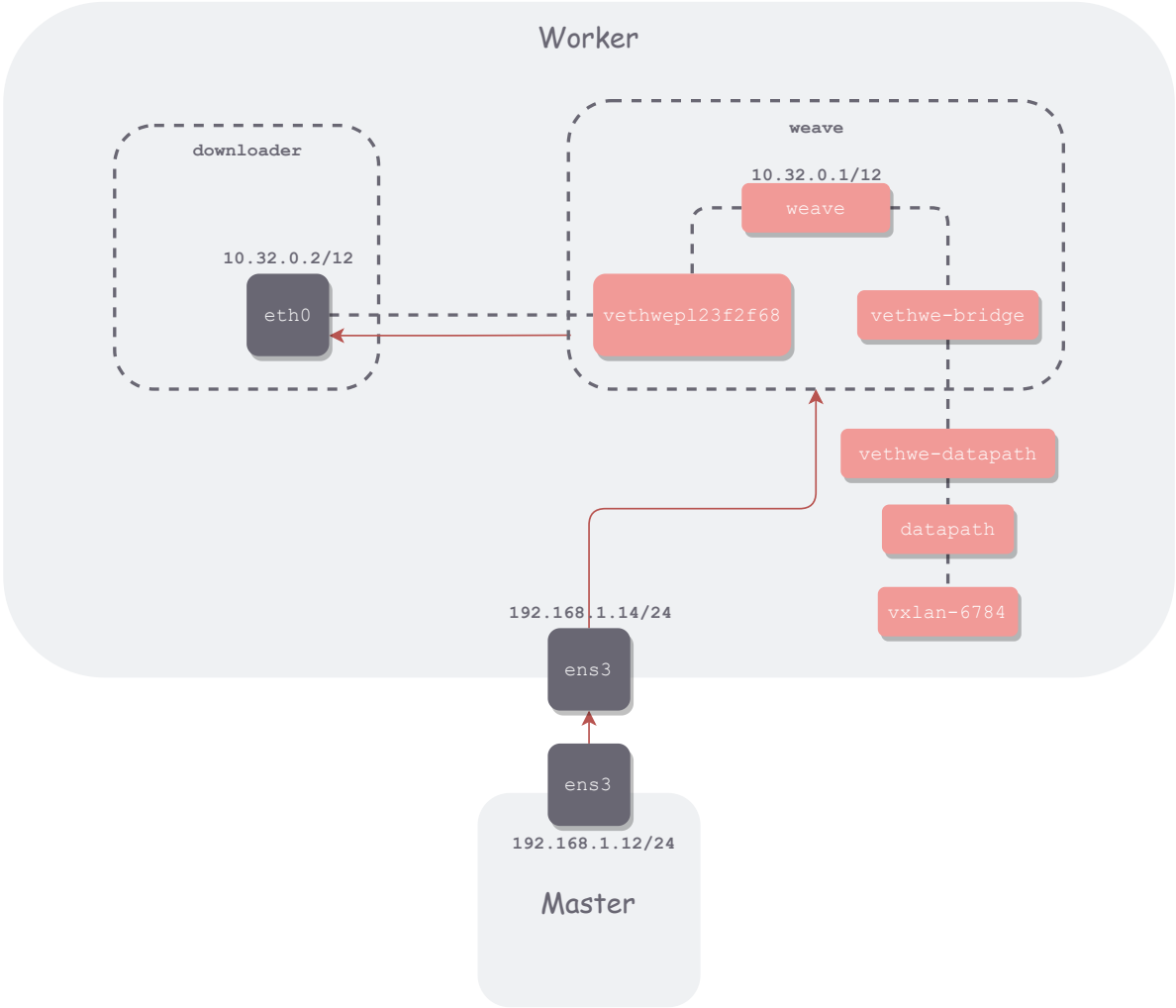
得到如下输出：

```
"NetworkMode":
"container:23f2f68378cc2b4f42d9f01845882040e7390b84736c7e05196fafb55f1fed72",
```

其中`container:23f2f68378cc2b4f42d9f01845882040e7390b84736c7e05196fafb55f1fed72`指代的就是`k8s_POD_pod_default_05260c21-c524-41c7-82e7-ea61f09f4f2f_0`容器(pause)。

> Q6: 请结合博客 https://blog.51cto.com/u_15069443/4043930 的内容，将容器中的veth与host机器上的veth匹配起来，并采用`ip link`和`ip addr`指令找到位于host机器中的所有网络设备及其之间的关系。结合两者的输出，试绘制出worker节点中涉及新部署Pod的所有网络设备和其网络结构，并在图中标注出从master节点中使用cluster ip访问位于worker节点中的Pod的网络路径

使用如下指令查看docker容器中的veth：`docker exec -it container_id/container_name ip addr`



上图即为worker节点的相关网络结构。其中红色实心箭头代表了从master节点中使用cluster ip访问位于worker节点中的Pod的网络路径。

weave通过自身的router完成了访问的转发。

部署Service

> Q7: 请采用声明式接口对Service进行部署，并将部署所需的yml文件记录在实践文档中

采用声明式接口：`kubectl apply -f service.yaml`

所采用的yaml文件：

```
apiVersion: v1
kind: Service
metadata:
  name: service
spec:
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
    name: port1
  - port: 6800
    targetPort: 6800
    protocol: TCP
    name: port2
  - port: 6880
    targetPort: 6880
    protocol: TCP
    name: port3
  selector:
    app: myApp
```

> Q8: 请在master节点中使用`iptables-save`指令输出所有的iptables规则，将其中与Service访问相关的iptable规则记录在实践文档中，并解释网络流量是如何采用基于iptables的方式被从对Service的cluster IP的访问定向到实际的Pod中的

通过指令`kubectl get service service`可以得到有关新创建的service的信息：

```
NAME      TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
service   ClusterIP   10.111.55.101   <none>        80/TCP,6800/TCP,6880/TCP
4m42s
```

通过指令`iptables-save`得到如下相关信息(与service无关的信息已剔除)

```
-A KUBE-SERVICES -d 10.111.55.101/32 -p tcp -m comment --comment
"default/service:port1 cluster IP" -m tcp --dport 80 -j KUBE-SVC-
N4AQN24LQINGUND7
-A KUBE-SERVICES -d 10.111.55.101/32 -p tcp -m comment --comment
"default/service:port2 cluster IP" -m tcp --dport 6800 -j KUBE-SVC-
UVN4MDSDRVPTA7FJ
-A KUBE-SERVICES -d 10.111.55.101/32 -p tcp -m comment --comment
"default/service:port3 cluster IP" -m tcp --dport 6880 -j KUBE-SVC-
S5VZYIT6AGQLBSGJ
```

这三条和刚才创建的service直接相关，其中10.111.55.101就是ClusterIp。

由iptables的指令说明：

指令参数	作用
-j	转发动作
-m	指定模块
--dport	目标端口

先看对`clusterIp:80`的访问，该访问被转发到`KUBE-SVC-N4AQN24LQINGUND7`

查看相关记录：

```
-A KUBE-SVC-N4AQN24LQINGUND7 -m comment --comment "default/service:port1" -j
KUBE-SEP-RVDZAUGKJFSTBZP7
```

在做简单注释之后，继续被转发到`KUBE-SEP-RVDZAUGKJFSTBZP7`

相关记录：

```
-A KUBE-SEP-RVDZAUGKJFSTBZP7 -s 10.32.0.2/32 -m comment --comment
"default/service:port1" -j KUBE-MARK-MASQ
-A KUBE-SEP-RVDZAUGKJFSTBZP7 -p tcp -m comment --comment "default/service:port1"
-m tcp -j DNAT --to-destination 10.32.0.2:80
```

最终通过DNAT(Destination Network Address Translation)转发到目标地址：10.32.0.2:80，这正是Pod的Ip地址。

对于`6800`端口：

```
-A KUBE-SERVICES -d 10.111.55.101/32 -p tcp -m comment --comment
"default/service:port2 cluster IP" -m tcp --dport 6800 -j KUBE-SVC-
UVN4MDSDRVPTA7FJ
-A KUBE-SVC-UVN4MDSDRVPTA7FJ -m comment --comment "default/service:port2" -j
KUBE-SEP-DWR2IHM6UD4ETWNV
-A KUBE-SEP-DWR2IHM6UD4ETWNV -s 10.32.0.2/32 -m comment --comment
"default/service:port2" -j KUBE-MARK-MASQ
-A KUBE-SEP-DWR2IHM6UD4ETWNV -p tcp -m comment --comment "default/service:port2"
-m tcp -j DNAT --to-destination 10.32.0.2:6800
```

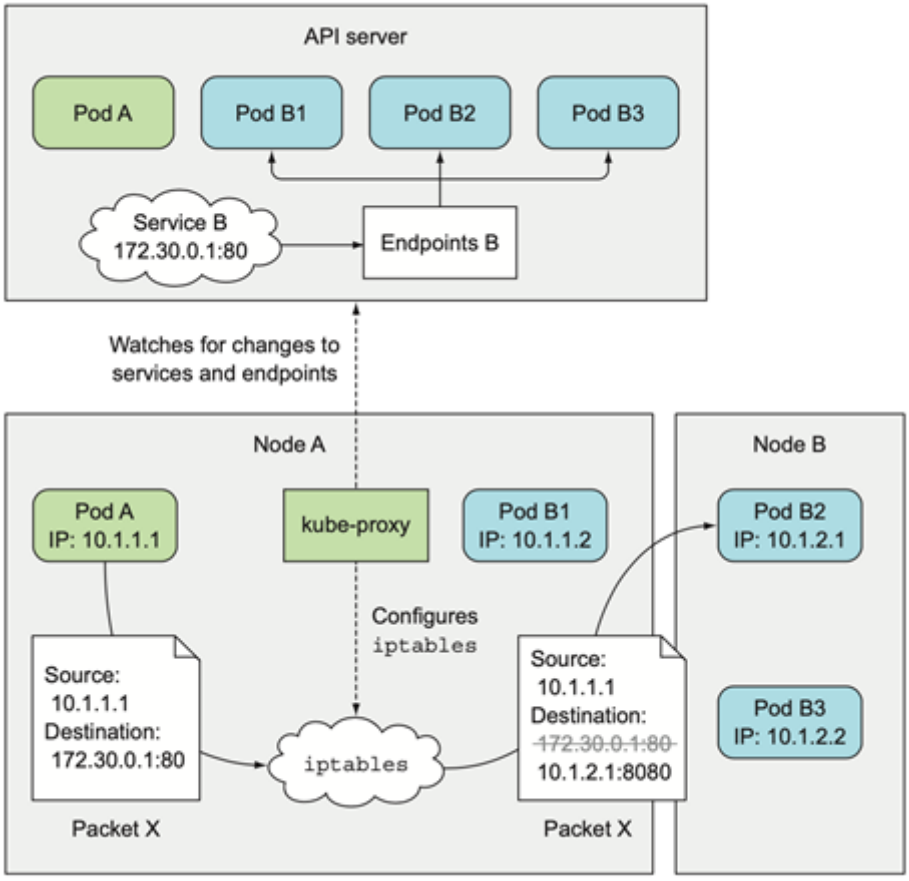
对于`6880`端口：

```
-A KUBE-SERVICES -d 10.111.55.101/32 -p tcp -m comment --comment
"default/service:port3 cluster IP" -m tcp --dport 6880 -j KUBE-SVC-
S5VZYIT6AGQLBSGJ
-A KUBE-SVC-S5VZYIT6AGQLBSGJ -m comment --comment "default/service:port3" -j
KUBE-SEP-OEXVTZY32XMIVGYO
-A KUBE-SEP-OEXVTZY32XMIVGYO -s 10.32.0.2/32 -m comment --comment
"default/service:port3" -j KUBE-MARK-MASQ
-A KUBE-SEP-OEXVTZY32XMIVGYO -p tcp -m comment --comment "default/service:port3"
-m tcp -j DNAT --to-destination 10.32.0.2:6880
```

> Q9: kube-proxy组件在整个service的定义与实现过程中起到了什么作用？请自行查找资料，并解释在iptables模式下，kube-proxy的功能

kube-proxy是Kubernetes的核心组件，部署在每个Node节点上，它是实现Kubernetes Service的通信与负载均衡机制的重要组件; kube-proxy负责为Pod创建代理服务，从apiserver获取所有service信息，并根据service信息创建代理服务，实现service到Pod的请求路由和转发，从而实现K8s层级的虚拟转发网络。

在iptables模式下，kube-proxy为service后端的每个Pod创建对应的iptables规则，直接将发向Cluster IP的请求重定向到一个Pod IP。



使用Deployment为Pod创建备份

> Q10: 请采用声明式接口对Deployment进行部署，并将Deployment所需要的yml文件记录在文档中

所使用的的yaml文件：

```yaml
kind: Deployment
apiVersion: apps/v1
metadata:
  name: deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myApp
  template:
    metadata:
      labels:
        app: myApp
    spec:
      containers:
        - name: viewer
          image: dplsming/nginx-fileserver:1.0
```

```
          ports:
            - containerPort: 80
          volumeMounts:
            - name: volume
              mountPath: /usr/share/nginx/html/files
        - name: downloader
          image: dplsming/aria2ng-downloader:1.0
          ports:
            - containerPort: 6800
            - containerPort: 6880
          volumeMounts:
            - name: volume
              mountPath: /data
      volumes:
        - name: volume
          hostPath:
            path: /pod
```

遇到的问题：

由于需要在一个worker节点上部署三个POD备份，可能会造成端口冲突，所以在pod template中，不应该指定hostPort，此时k8s会自动选择一个随机的端口使用，如此一来解决了端口冲突导致无法部署的问题。

service.yaml也做了相应修改：

```
apiVersion: v1
kind: Service
metadata:
  name: service
spec:
  type: ClusterIP
  ports:
  - port: 80
    protocol: TCP
    name: port1
  - port: 6800
    protocol: TCP
    name: port2
  - port: 6880
    protocol: TCP
    name: port3
  selector:
    app: myApp
```

> Q11: 请在master节点中使用`iptables-save`指令输出所有的iptables规则，将其中与Service访问相关的iptable规则记录在实践文档中，并解释网络流量是如何采用基于iptable的方式被从对Service的cluster ip的访问，以随机且负载均衡的方式，定向到Deployment所创建的实际的Pod中的

由于部署deployment之后，重新启动了service，所以其虚拟IP发生了变化，目前为`10.106.221.241/32`

由指令`iptables-save`，我们得到了如下相关规则：

```
-A KUBE-SERVICES -d 10.106.221.241/32 -p tcp -m comment --comment
"default/service:port1 cluster IP" -m tcp --dport 80 -j KUBE-SVC-
N4AQN24LQINGUND7
-A KUBE-SERVICES -d 10.106.221.241/32 -p tcp -m comment --comment
"default/service:port2 cluster IP" -m tcp --dport 6800 -j KUBE-SVC-
UVN4MDSDRVPTA7FJ
-A KUBE-SERVICES -d 10.106.221.241/32 -p tcp -m comment --comment
"default/service:port3 cluster IP" -m tcp --dport 6880 -j KUBE-SVC-
S5VZYIT6AGQLBSGJ
```

与Service版块中的讨论类似，我们以80端口为例进行说明，其余端口的逻辑是类似的，不必赘述。

通向该端口(`clusterIp:80`)的网络流量会被转发到`KUBE-SVC-N4AQN24LQINGUND7`

有三条关于`KUBE-SVC-N4AQN24LQINGUND7`的规则

```
-A KUBE-SVC-N4AQN24LQINGUND7 -m comment --comment "default/service:port1" -m
statistic --mode random --probability 0.33333333349 -j KUBE-SEP-LQIGZSLMK2LBCNBP
-A KUBE-SVC-N4AQN24LQINGUND7 -m comment --comment "default/service:port1" -m
statistic --mode random --probability 0.50000000000 -j KUBE-SEP-KDXLKT7NB4FKDRHC
-A KUBE-SVC-N4AQN24LQINGUND7 -m comment --comment "default/service:port1" -j
KUBE-SEP-WS3QRLXHGRL46REP
```

显然，上述的三条规则是在做负载均衡，通过一定的概率进行网络流量的随机转发，让我们继续跟踪这一转发链。

对于`KUBE-SEP-LQIGZSLMK2LBCNBP`：

```
-A KUBE-SEP-LQIGZSLMK2LBCNBP -s 10.32.0.3/32 -m comment --comment
"default/service:port1" -j KUBE-MARK-MASQ
-A KUBE-SEP-LQIGZSLMK2LBCNBP -p tcp -m comment --comment "default/service:port1"
-m tcp -j DNAT --to-destination 10.32.0.3:80
```

对于`KUBE-SEP-KDXLKT7NB4FKDRHC`

```
-A KUBE-SEP-KDXLKT7NB4FKDRHC -s 10.32.0.4/32 -m comment --comment
"default/service:port1" -j KUBE-MARK-MASQ
-A KUBE-SEP-KDXLKT7NB4FKDRHC -p tcp -m comment --comment "default/service:port1"
-m tcp -j DNAT --to-destination 10.32.0.4:80
```

对于`KUBE-SEP-WS3QRLXHGRL46REP`

```
-A KUBE-SEP-WS3QRLXHGRL46REP -s 10.32.0.5/32 -m comment --comment
"default/service:port1" -j KUBE-MARK-MASQ
-A KUBE-SEP-WS3QRLXHGRL46REP -p tcp -m comment --comment "default/service:port1"
-m tcp -j DNAT --to-destination 10.32.0.5:80
```

通过`kubectl describe pod`指令我们可以获取指定pod的ip，不难发现与上述三个规则最终的目标ip是一致的。

对于另外两个端口的所有相关规则如下所示：

`6800`端口：

```
-A KUBE-SVC-UVN4MDSDRVPTA7FJ -m comment --comment "default/service:port2" -m
statistic --mode random --probability 0.33333333349 -j KUBE-SEP-2654F7UDVXHOXRLK
-A KUBE-SVC-UVN4MDSDRVPTA7FJ -m comment --comment "default/service:port2" -m
statistic --mode random --probability 0.50000000000 -j KUBE-SEP-G2SVSVOLNB4OWBS3
-A KUBE-SVC-UVN4MDSDRVPTA7FJ -m comment --comment "default/service:port2" -j
KUBE-SEP-CB4CBGXRS4PY6HL4

-A KUBE-SEP-2654F7UDVXHOXRLK -s 10.32.0.3/32 -m comment --comment
"default/service:port2" -j KUBE-MARK-MASQ
-A KUBE-SEP-2654F7UDVXHOXRLK -p tcp -m comment --comment "default/service:port2"
-m tcp -j DNAT --to-destination 10.32.0.3:6800

-A KUBE-SEP-G2SVSVOLNB4OWBS3 -s 10.32.0.4/32 -m comment --comment
"default/service:port2" -j KUBE-MARK-MASQ
-A KUBE-SEP-G2SVSVOLNB4OWBS3 -p tcp -m comment --comment "default/service:port2"
-m tcp -j DNAT --to-destination 10.32.0.4:6800

-A KUBE-SEP-CB4CBGXRS4PY6HL4 -s 10.32.0.5/32 -m comment --comment
"default/service:port2" -j KUBE-MARK-MASQ
-A KUBE-SEP-CB4CBGXRS4PY6HL4 -p tcp -m comment --comment "default/service:port2"
-m tcp -j DNAT --to-destination 10.32.0.5:6800
```

`6880`端口：

```
-A KUBE-SVC-S5VZYIT6AGQLBSGJ -m comment --comment "default/service:port3" -m
statistic --mode random --probability 0.33333333349 -j KUBE-SEP-WO4NWKN6XHEW4PWA
-A KUBE-SVC-S5VZYIT6AGQLBSGJ -m comment --comment "default/service:port3" -m
statistic --mode random --probability 0.50000000000 -j KUBE-SEP-CMIEQRTH76PLMOGM
-A KUBE-SVC-S5VZYIT6AGQLBSGJ -m comment --comment "default/service:port3" -j
KUBE-SEP-FVQWCGYG45DUNJUE

-A KUBE-SEP-WO4NWKN6XHEW4PWA -s 10.32.0.3/32 -m comment --comment
"default/service:port3" -j KUBE-MARK-MASQ
-A KUBE-SEP-WO4NWKN6XHEW4PWA -p tcp -m comment --comment "default/service:port3"
-m tcp -j DNAT --to-destination 10.32.0.3:6880

-A KUBE-SEP-CMIEQRTH76PLMOGM -s 10.32.0.4/32 -m comment --comment
"default/service:port3" -j KUBE-MARK-MASQ
-A KUBE-SEP-CMIEQRTH76PLMOGM -p tcp -m comment --comment "default/service:port3"
-m tcp -j DNAT --to-destination 10.32.0.4:6880

-A KUBE-SEP-FVQWCGYG45DUNJUE -s 10.32.0.5/32 -m comment --comment
"default/service:port3" -j KUBE-MARK-MASQ
-A KUBE-SEP-FVQWCGYG45DUNJUE -p tcp -m comment --comment "default/service:port3"
-m tcp -j DNAT --to-destination 10.32.0.5:6880
```

> Q12: 在该使用Deployment的部署方式下，不同Pod之间的文件是否共享？该情况会在实际使用文件下载与共享服务时产生怎样的实际效果与问题？应如何解决这一问题？

由于挂载的是同一个node的同一个本地文件，所以不同pod之间实际上是共享文件的。

由于共享文件，所以可能产生一些资源竞争问题，可能导致对文件的读或者写出现意料之外的结果。

解决方案：每个Pod分配一个UUID，在以该UUID为名字的文件夹内进行读写，类似于用namespace将不同Pod进行隔离。读取文件时，如果Pod自身所属的文件夹内没有该文件，则去其他的Pod所属的文件夹内寻找并读取。