> 讲师：阿良（微信 init1024）
>
> DevOps实战学院：**www.ctnrs.com**
>
> 概述：
>
> - 2个小时
> - 17道题（每道题分数按照操作的难度变化，1~8分不等）
> - 66分通过
> - 考试中可查询kubernetes.io官网文档
>
> 命令行补全：source <(kubectl completion bash)
>
> 切换集群：kubectl config use-context k8s

1、使用kubeadm搭建一个K8s集群

1、安装Docker（考试环境已装好）

```
curl -fsSL http://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg | sudo apt-key add -

vi /etc/apt/sources.list # 添加一行
deb [arch=amd64] http://mirrors.aliyun.com/docker-ce/linux/ubuntu xenial stable

apt-get update
apt-get install docker-ce -y

systemctl start docker
systemctl enable docker
```

2、配置kubernetes源

```
apt-get update && apt-get install -y apt-transport-https
curl https://mirrors.aliyun.com/kubernetes/apt/doc/apt-key.gpg | apt-key add -

cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
deb https://mirrors.aliyun.com/kubernetes/apt/ kubernetes-xenial main
EOF
```

3、安装kubelet、kubeadm和kubectl

```
apt-get update
apt-get install -y kubelet kubeadm kubectl
```

指定版本：apt-get install -y kubelet=1.19.0-00 kubeadm=1.19.0-00 kubectl=1.19.0-00

4、配置文件引导Master

```
$ vi kubeadm.conf
apiVersion: kubeadm.k8s.io/v1beta2
kind: ClusterConfiguration
kubernetesVersion: v1.18.0
imageRepository: registry.aliyuncs.com/google_containers
networking:
  podSubnet: 10.244.0.0/16
  serviceSubnet: 10.96.0.0/12

$ kubeadm init --config kubeadm.conf --ignore-preflight-errors=all
```

5、向集群添加新节点，执行kubeadm init输出的kubeadm join命令

6、部署CNI网络

```
kubectl apply -f https://docs.projectcalico.org/v3.11/manifests/calico.yaml
```

参考文档：**https://kubernetes.io/zh/docs/setup/production-environment/tools/kubeadm/install-kubeadm**

2、新建命名空间，在该命名空间中创建一个pod

- 命名空间名称：cka
- pod名称：pod-01
- 镜像：nginx

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-01
  namespace: cka
  labels:
    app: myapp
spec:
  containers:
  - name: nginx
    image: nginx
```

参考文档：**https://kubernetes.io/zh/docs/concepts/workloads/pods/**

3、创建一个deployment并暴露Service

- 名称：aliang-666
- 镜像：nginx

命令行：

```
kubectl create deployment aliang-666 --image=nginx
kubectl expose deployment aliang-666 --port=80 --target-port=80
```

或者使用YAML创建：

```
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: aliang-666
  labels:
     app: nginx
spec:
  replicas: 3
  selector:
     matchLabels:
        app: nginx
  template:
     metadata:
        labels:
           app: nginx
     spec:
        containers:
        - name: nginx
           image: nginx
           ports:
           - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: aliang-666
spec:
  selector:
     app: nginx
  ports:
     - protocol: TCP
        port: 80
        targetPort: 80
```

参考文档1：**https://kubernetes.io/docs/concepts/workloads/controllers/deployment/**

参考文档2：**https://kubernetes.io/docs/concepts/services-networking/service/**

4、列出命名空间下指定标签pod

- 命名空间名称：kube-system
- 标签：k8s-app=kube-dns

```
kubectl get pods -n kube-system -l k8s-app=kube-dns
```

5、查看pod日志，并将日志中Error的行记录到指定文件

- pod名称：web
- 文件：/opt/web-log

```
kubectl logs web | grep error > /opt/web-log
```

6、查看指定标签使用cpu最高的pod，并记录到到指定文件

- 标签：app=web
- 文件：/opt/cpu

```
kubectl top pods -l app=web --sort-by="cpu" > /opt/cpu
```

7、在节点上配置kubelet托管启动一个pod

- 节点：k8s-node1
- pod名称：web
- 镜像：nginx

```
# 检查是否启用
cat /var/lib/kubelet/config.yaml
staticPodPath: /etc/kubernetes/manifests

vi /etc/kubernetes/manifests/web.yaml
apiVersion: v1
kind: Pod
metadata:
  name: web
  namespace:
  labels:
     app: myapp
spec:
  containers:
  - name: nginx
     image: nginx
```

注：如果不生效，检查配置文件是否启用该功能

8、向pod中添加一个init容器，init容器创建一个空文件，如果该空文件没有被检测到，pod就退出

- pod名称：web

```
apiVersion: v1
kind: Pod
metadata:
  name: web
spec:
  initContainers:
  - name: init
     image: nginx
     command:
     - touch
     - /opt/test
```

```
    volumeMounts:
    - name: data
      mountPath: /opt
  containers:
  - name: nginx
    image: nginx
    volumeMounts:
    - name: data
      mountPath: /opt
    livenessProbe:
      exec:
        command:
        - cat
        - /opt/test
  restartPolicy: Never
  volumes:
  - name: data
    emptyDir: {}
```

9、创建一个deployment 副本数 3，然后滚动更新镜像版本，并记录这个更新记录，最后再回滚到上一个版本

- 名称：nginx
- 镜像版本：1.16
- 更新镜像版本：1.17

```
kubectl create deployment web --image=nginx:1.16
kubectl set image deployment web nginx=nginx:1.17 --record
kubectl rollout history deploy web     # 查看版本记录
kubectl rollout undo deployment web    # 回滚到上一个版本
kubectl rollout undo deployment web --to-revision=1    # 也可以回滚到指定版本
```

10、给web deployment扩容副本数为3

```
kubectl scale deployment web --replicas=3
```

11、创建一个pod，其中运行着nginx、redis、memcached、consul 4个容器

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: nginx
```

```
    image: nginx
  - name: redis
    image: redis
  - name: memcache
    image: memcached
  - name: consul
    image: consul
```

12、生成一个deployment yaml文件保存到/opt/deploy.yaml

- 名称：web
- 标签：app_env_stage=dev

```
kubectl create deployment web --image=nginx --dry-run=client -o yaml > /opt/deploy.yaml

# 再修改标签
apiVersion: apps/v1
kind: Deployment
metadata:
  name: java-demo
spec:
  replicas: 3
  selector:
    matchLabels:
      app_env_stage=dev
  template:
    metadata:
      labels:
        app_env_stage=dev
    spec:
      containers:
      - name: nginx
        image: nginx
```

13、创建一个pod，分配到指定标签node上

- pod名称：web
- 镜像：nginx
- node标签：disk=ssd

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
     image: nginx
  nodeSelector:
     disk: ssd
```

参考文档：https://kubernetes.io/docs/concepts/configuration/assign-pod-node/

14、确保在每个节点上运行一个pod

- 名称：filebeat
- 镜像：elastic/filebeat:7.3.2

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: filebeat
  namespace: kube-system
spec:
  selector:
     matchLabels:
        name: filebeat
  template:
     metadata:
        labels:
           name: filebeat
     spec:
        containers:
        - name: log
           image: elastic/filebeat:7.3.2
```

参考文档：https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/

15、查看集群中状态为ready的node数量，不包含被打了NodeSchedule污点的节点，并将结果写到/opt/node.txt

```
kubectl describe node $(kubectl get nodes|grep Ready|awk '{print $1}') |grep Taint|grep -vc NoSchedule > /opt/node.txt
```

16、设置成node不能调度，并使已被调度的pod重新调度

```
kubectl cordon <节点名称>
kubectl drain node1 --ignore-daemonsets
```

17、给一个pod创建service，并可以通过ClusterIP访问

- 名称：web-service
- pod名称：web-pod
- 容器端口：80

```
kubectl expose pod web-pod --port=80 --target-port=80 --name=web-service --type=NodePort
kubectl get svc web-service
curl CLUSTER-IP
```

18、 任意名称创建deployment和service，然后使用busybox容器nslookup解析service

```
kubectl create deployment nginx-dns --image=nginx
kubectl expose deployment nginx-dns --name=nginx-dns --port=80

kubectl run bs-dns --image=busybox:1.28.4 busybox sleep 36000
kubectl exec -it bs-dns -- nslookup nginx-dns
```

19、列出命名空间下某个service关联的所有pod，并将pod名称写到/opt/pod.txt文件中（使用标签筛选）

- 命名空间：default
- service名称：web

```
先查看service用的标签选择器：
kubectl get service web -o yaml
或者
kubectl get svc java-demo -o jsonpath='{.spec.selector}'

kubectl get pods -l app=web -o name > /opt/pod.txt
```

> DevOps实战学院：**www.ctnrs.com**

20、创建一个secret，并创建2个pod，pod1挂载该secret，路径为/etc/foo，pod2使用环境变量引用该secret，该变量的环境变量名为ABC

- secret名称：mysecret
- pod1名称：pod-volume-secret

- pod2名称：pod-env-secret

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  password: MWYyZDF1MmU2N2Rm
```

password值经过base64编码。

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-volume-secret
spec:
  containers:
  - name: nginx
     image: nginx
     volumeMounts:
     - name: foo
        mountPath: "/etc/foo"
  volumes:
  - name: foo
     secret:
        secretName: mysecret
---
apiVersion: v1
kind: Pod
metadata:
  name: pod-env-secret
spec:
  containers:
  - name: nginx
     image: nginx
     env:
        - name: SECRET_PASSWORD
           valueFrom:
              secretKeyRef:
                 name: mysecret
                 key: password
```

参考文档：**https://kubernetes.io/docs/concepts/configuration/secret/**

21、 创建一个Pod使用PV自动供给

- 容量：5Gi
- 访问模式：ReadWriteMany

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: test-claim
spec:
```

```
  storageClassName: "managed-nfs-storage"
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
---
apiVersion: v1
kind: Pod
metadata:
  name: web
spec:
  containers:
  - name: web
    image: nginx
    volumeMounts:
      - name: data
        mountPath: "/mnt"
  volumes:
    - name: data
      persistentVolumeClaim:
        claimName: test-claim
```

参考文档：https://kubernetes.io/docs/concepts/storage/persistent-volumes/

22、创建一个pod并挂载数据卷，不可以用持久卷

- 卷来源：emptyDir、hostPath任意
- 挂载路径：/data/redis

```
apiVersion: v1
kind: Pod
metadata:
  name: no-persistent-redis
spec:
  containers:
  - name: redis
    image: redis
    volumeMounts:
    - name: cache
      mountPath: /data/redis
  volumes:
  - name: cache
    emptyDir: {}
```

23、将pv按照名称、容量排序，并保存到/opt/pv文件

```
kubectl get pv --sort-by=.metadata.name > /opt/pv
kubectl get pv --sort-by=.spec.capacity.storage > /opt/pv
```

24、Bootstrap Token方式增加一台Node（二进制）

1、确保kube-apiserver配置文件已启用Bootstrap Token（--enable-bootstrap-token-auth=true）

2、使用Secret存储Bootstrap Token

3、创建RBAC角色绑定，允许 kubelet tls bootstrap 创建 CSR 请求

4、kubelet配置Bootstrap kubeconfig文件

5、kubectl get csr && kubectl certificate approve xxx

参考文档1：**https://kubernetes.io/docs/reference/access-authn-authz/bootstrap-tokens/**

参考文档2：**https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet-tls-bootstrapping/**

25、Etcd数据库备份与恢复（kubeadm）

备份：

```
ETCDCTL_API=3 etcdctl \
snapshot save snap.db \
--endpoints=https://127.0.0.1:2379 \
--cacert=/etc/kubernetes/pki/etcd/ca.crt \
--cert=/etc/kubernetes/pki/etcd/server.crt \
--key=/etc/kubernetes/pki/etcd/server.key
```

恢复：

```
1、先暂停kube-apiserver和etcd容器
mv /etc/kubernetes/manifests /etc/kubernetes/manifests.bak
mv /var/lib/etcd/ /var/lib/etcd.bak
2、恢复
ETCDCTL_API=3 etcdctl \
snapshot restore snap.db \
--data-dir=/var/lib/etcd
3、启动kube-apiserver和etcd容器
mv /etc/kubernetes/manifests.bak /etc/kubernetes/manifests
```

> 注：考试使用的k8s集群的版本不同，命令可能会有一些不同，可以使用etcdctl --endpoints -h 命令查看

参考文档：**https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/**

26、给定一个Kubernetes集群，排查管理节点组件存在问题

```
kubectl get cs
systemctl start xxx
systemctl enable xxx
```

27、工作节点 NotReady状态怎么解决?

```
ssh k8s-node1
systemctl start kubelet
systemctl enable kubelet
```

28、升级管理节点kubelet ,kubectl 组件由1.18 升级为1.19 ， 工作节点不升级

升级管理节点:

```
1、查找最新版本号
yum list --showduplicates kubeadm --disableexcludes=kubernetes
2、升级kubeadm
yum install -y kubeadm-1.19.3-0 --disableexcludes=kubernetes
3、驱逐node上的pod，且不可调度
kubectl drain k8s-master --ignore-daemonsets
4、检查集群是否可以升级，并获取可以升级的版本
kubeadm upgrade plan
5、执行升级
kubeadm upgrade apply v1.19.3
6、取消不可调度
kubectl uncordon k8s-master
7、升级kubelet和kubectl
yum install -y kubelet-1.19.3-0 kubectl-1.19.3-0 --disableexcludes=kubernetes
8、重启kubelet
systemctl daemon-reload
systemctl restart kubelet
9、验证
kubectl get node
```

参考文档: https://kubernetes.io/zh/docs/tasks/administer-cluster/kubeadm/kubeadm-upgrade/

29、创建一个ingress

- 域名：example.ctnrs.com
- service名称：web
- service端口：80

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: java-demo
spec:
  rules:
  - host: example.ctnrs.com
     http:
        paths:
        - path: /
           pathType: Prefix
```

```
          backend:
             service:
                name: web
                port:
                   number: 80
```

参考文档：https://kubernetes.io/zh/docs/concepts/services-networking/ingress/

30、Pod创建一个边车容器读取业务容器日志

```
apiVersion: v1
kind: Pod
metadata:
  name: log-counter
spec:
  containers:
  - name: web
     image: busybox
     command: ["/bin/sh","-c","for i in {1..100};do echo $i >> /var/log/access.log;sleep
1;done"]
     volumeMounts:
        - name: varlog
           mountPath: /var/log
  - name: log
     image: busybox
     command: ["/bin/sh","-c","tail -f /var/log/access.log"]
     volumeMounts:
        - name: varlog
           mountPath: /var/log
  volumes:
  - name: varlog
     emptyDir: {}
```

参考文档：https://kubernetes.io/docs/concepts/cluster-administration/logging/

31、创建一个clusterrole，关联到一个服务账号

```
# 创建用户
$ kubectl create serviceaccount dashboard-admin -n kube-system
# 用户授权
$ kubectl create clusterrolebinding dashboard-admin --clusterrole=cluster-admin --
serviceaccount=kube-system:dashboard-admin
```

32、default命名空间下所有pod可以互相访问，也可以访问其他命名空间Pod，但其他命名空间不能访问default命名空间Pod

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-from-other-namespaces
  namespace: default
spec:
  podSelector: {}
  policyTypes:

  - Ingress
     ingress:
  - from:
     - podSelector: {}
```

podSelector: {}：default命名空间下所有Pod

from.podSelector: {}：如果未配置具体的规则，默认不允许

参考文档： **https://kubernetes.io/docs/concepts/services-networking/network-policies/**

> DevOps实战学院： **www.ctnrs.com**