一、CentOS7安装K8S环境-通用

前期准备

修改hostname

```
[root@master ~]$ vi /etc/hostname # 修改hostname
[root@master ~]$ vi /etc/hosts  # 将本机IP指向hostname
[root@master ~]$ reboot -h      # 重启(可以做完全部前期准备后再重启)
```

配置主机和IP映射

```
192.168.3.233 master
192.168.3.234 worker1
192.168.3.235 worker2
192.168.3.236 worker3
192.168.3.237 worker4
192.168.3.238 worker5

## 远程拷贝到每个机器上面
scp /etc/hosts 192.168.3.237@root:/etc/hosts
```

关闭防火墙(不推荐)

```
systemctl disable firewalld
systemctl stop firewalld
```

安装Docker

```
$ wget https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo -O
/etc/yum.repos.d/docker-ce.repo
$ yum -y install docker-ce-19.03.13-3.el7
$ systemctl enable docker && systemctl start docker
$ docker --version
Server Version: 19.03.13
Kernel Version: 3.10.0-693.el7.x86_64
```

添加阿里云YUM软件源

```
$ cat > /etc/yum.repos.d/kubernetes.repo << EOF
[kubernetes]
name=Kubernetes
baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg
https://mirrors.aliyun.com/kubernetes/yum/doc/rpm-package-key.gpg
EOF
```

修改配置文件

```
vim /etc/docker/daemon.json

{
  "registry-mirrors": [
        "https://1nj0zren.mirror.aliyuncs.com",
        "https://docker.mirrors.ustc.edu.cn",
        "http://f1361db2.m.daocloud.io",
        "https://registry.docker-cn.com"
    ],
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
```

- 重新加载配置文件 `systemctl daemon-reload`
- 重启Docker `systemctl restart docker`

安装Kubernetes

添加源

由于国内网络原因, 官方文档中的地址不可用, 本文替换为阿里云镜像地址, 执行以下代码即可:

```
cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=http://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=http://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg
http://mirrors.aliyun.com/kubernetes/yum/doc/rpm-package-key.gpg
exclude=kube*
EOF
```

安装

```
yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
# 指定版本（可选） 目前用的是最新版本
yum install -y kubelet-1.18.0 kubeadm-1.18.0 kubectl-1.18.0 --disableexcludes=kubernetes

# 启动
systemctl enable kubelet && systemctl start kubelet
```

修改网络配置

```
cat <<EOF >  /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
sysctl --system
```

> 注意: 至此, 以上的全部操作, 在Worker机器上也需要执行. 注意hostname等不要相同.

二、初始化Master

生成初始化文件

A 配置文件方式

```
kubeadm config print init-defaults > kubeadm-init.yaml

# 编辑配置文件
vi kubeadm-init.yaml
```

该文件有两处需要修改:

- 将 `advertiseAddress: 1.2.3.4` 修改为本机地址
- 将 `imageRepository: k8s.gcr.io` 修改为 `imageRepository: registry.cn-hangzhou.aliyuncs.com/google_containers`

修改完毕后文件如下:

```
apiVersion: kubeadm.k8s.io/v1beta2
bootstrapTokens:
- groups:
  - system:bootstrappers:kubeadm:default-node-token
  token: abcdef.0123456789abcdef
  ttl: 24h0m0s
  usages:
  - signing
  - authentication
kind: InitConfiguration
localAPIEndpoint:
  advertiseAddress: 10.33.30.92 # 本机IP
  bindPort: 6443
nodeRegistration:
  criSocket: /var/run/dockershim.sock
  name: k8s-master
  taints:
  - effect: NoSchedule
    key: node-role.kubernetes.io/master
---
apiServer:
  timeoutForControlPlane: 4m0s
apiVersion: kubeadm.k8s.io/v1beta2
certificatesDir: /etc/kubernetes/pki
clusterName: kubernetes
controllerManager: {}
dns:
  type: CoreDNS
etcd:
  local:
    dataDir: /var/lib/etcd
imageRepository: registry.cn-hangzhou.aliyuncs.com/google_containers #镜像仓库
kind: ClusterConfiguration
kubernetesVersion: v1.19.0
networking:
  dnsDomain: cluster.local
  serviceSubnet: 10.96.0.0/12
  podSubnet: 10.244.0.0/16 # 新增Pod子网络
scheduler: {}
```

B 直接传参方式(可选)

```
kubeadm init \
    --apiserver-advertise-address=192.168.3.222 \
    --image-repository registry.cn-hangzhou.aliyuncs.com/google_containers \
    --kubernetes-version v1.18.0 \
    --service-cidr=10.1.0.0/16 \
    --pod-network-cidr=10.244.0.0/16
```

下载镜像

```
[root@master ~]$ kubeadm config images pull --config kubeadm-init.yaml
```

配置禁用Swap

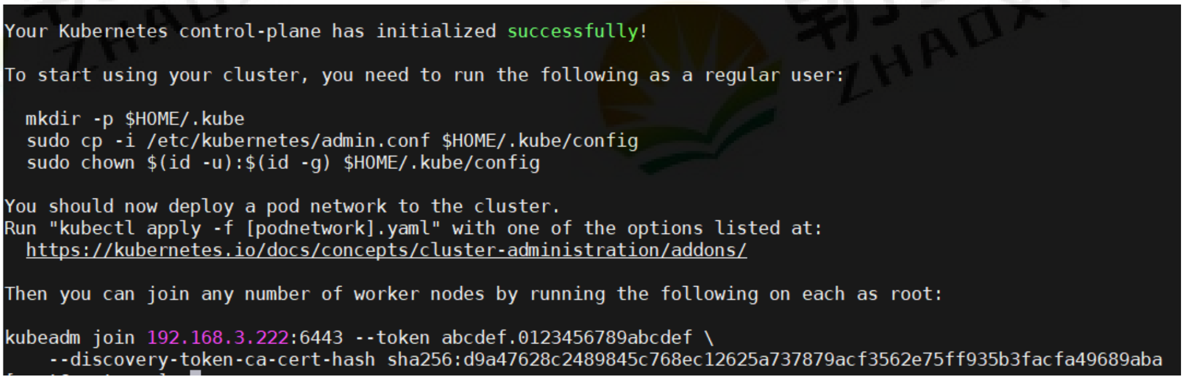
```
# 临时关闭（宿主机重启后k8s不会自动部署，需要手动关闭）
swapoff -a
```

执行初始化

```
[root@master ~]$ kubeadm init --config kubeadm-init.yaml
0825 03:43:47.245862    2166 configset.go:202] WARNING: kubeadm cannot validate
component configs for API groups [kubelet.config.k8s.io kubeproxy.config.k8s.io]
[init] Using Kubernetes version: v1.18.0
[preflight] Running pre-flight checks
error execution phase preflight: [preflight] Some fatal errors occurred:
        [ERROR NumCPU]: the number of available CPUs 1 is less than the required
2
        [ERROR Swap]: running with swap on is not supported. Please disable swap
[preflight] If you know what you are doing, you can make a check non-fatal with
`--ignore-preflight-errors=...`
To see the stack trace of this error execute with --v=5 or higher

出现端口被占用情况
[root@master ~]$ kubeadm reset
```

```
kubeadm init --config kubeadm-init.yaml --ignore-preflight-errors=Swap
```

验证是否成功



```
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

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 192.168.3.222:6443 --token abcdef.0123456789abcdef \
    --discovery-token-ca-cert-hash sha256:d9a47628c2489845c768ec12625a737879acf3562e75ff935b3facfa49689aba
```

```
# 这个是node节点需要做的
kubeadm join 192.168.3.233:6443 --token abcdef.0123456789abcdef \
    --discovery-token-ca-cert-hash
sha256:4158c5823bc89d10a310533473f506342d93ee9255c7d9331300bf5fe4251ceb
```

配置环境, 让当前用户可以执行kubectl命令

```
## 配置kubectl执行命令环境
mkdir -p $HOME/.kube
cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
chown $(id -u):$(id -g) $HOME/.kube/config

## 执行kubectl命令查看机器节点
kubectl get node
-----------------------------------------
NAME      STATUS     ROLES    AGE   VERSION
master    NotReady   master   48m   v1.18.8
```

配置网络

使用以下命令安装Calico

```
wget https://docs.projectcalico.org/manifests/calico.yaml

vi calico.yaml

## 编辑calico.yaml
    ## 修改calico.yaml文件设置指定的网卡
    # Cluster type to identify the deployment type
    - name: CLUSTER_TYPE
      value: "k8s,bgp"
    # IP automatic detection
    - name: IP_AUTODETECTION_METHOD
      value: "interface=en.*"
    # Auto-detect the BGP IP address.
    - name: IP
      value: "autodetect"
    # Enable IPIP
    - name: CALICO_IPV4POOL_IPIP
      value: "Never"
## 构建calico网络
kubectl apply -f calico.yaml
```

此时查看node信息, master的状态已经是`Ready`了.

```
[root@master ~]$ kubectl get node
NAME      STATUS     ROLES    AGE   VERSION
master    Ready    master    48m    v1.18.8
```

安装Dashboard

> 文档地址: Web UI (Dashboard)

部署Dashboard

> 文档地址: Deploying the Dashboard UI

```
[root@master ~]$ wget
https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0-
beta4/aio/deploy/recommended.yaml
## 异常
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 0.0.0.0, ::
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|0.0.0.0|:443... failed: Connection refused.
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|::|:443...
failed: Connection refused.
## 解决
解决GitHub的raw.githubusercontent.com无法连接问题
1、https://site.ip138.com/raw.Githubusercontent.com/
2、输入raw.githubusercontent.com
查询IP地址，获取到对应的IP
151.101.108.133
3、编辑/etc/hosts文件配置映射
151.101.108.133 raw.githubusercontent.com
==========================================================================
[root@master ~]$ kubectl apply -f recommended.yaml
```

部署完毕后, 执行 `kubectl get pods --all-namespaces` 查看pods状态

```
[root@master ~]$ kubectl get pods --all-namespaces | grep dashboard
NAMESPACE              NAME                                         READY
STATUS
kubernetes-dashboard   dashboard-metrics-scraper-66b49655d4-mtb4v    Running   0

kubernetes-dashboard   kubernetes-dashboard-74b4487bfc-sr162         Running   0
```

访问dashboard

这里作为演示，使用nodeport方式将dashboard服务暴露在集群外，指定使用30443端口，可自定义：

```
kubectl  patch svc kubernetes-dashboard -n kubernetes-dashboard \
-p '{"spec":{"type":"NodePort","ports":
[{"port":443,"targetPort":8443,"nodePort":30443}]}}'
```

查看暴露的service,已修改为nodeport类型：

```
# kubectl -n kubernetes-dashboard get svc
NAME                         TYPE        CLUSTER-IP       EXTERNAL-IP    PORT(S)
      AGE
dashboard-metrics-scraper    ClusterIP   10.102.18.37     <none>         8000/TCP
        69s
kubernetes-dashboard         NodePort    10.110.118.188   <none>
 443:30443/TCP    69s
```

修改Service(可以使用这个方式操作)

```
vim ~/recommended.yaml
```

```
kind: Service
apiVersion: v1
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kubernetes-dashboard
spec:
  type: NodePort
  ports:
    - port: 443
      targetPort: 8443
      nodePort: 30443
  selector:
    k8s-app: kubernetes-dashboard
```

创建用户

> 文档地址: [Creating sample user]

创建一个用于登录Dashboard的用户. 创建文件 `dashboard-adminuser.yaml` 内容如下:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin-user
  namespace: kube-system
```

执行命令

```
kubectl apply -f dashboard-adminuser.yaml
```

生成证书（证书登录）

官方文档中提供了登录1.7.X以上版本的登录方式, 但并不清晰, 笔者没有完全按照该文档的方式进行操作.

```
[root@master ~]$ grep 'client-certificate-data' ~/.kube/config | head -n 1 | awk
'{print $2}' | base64 -d >> kubecfg.crt
[root@master ~]$ grep 'client-key-data' ~/.kube/config | head -n 1 | awk '{print
$2}' | base64 -d >> kubecfg.key
[root@master ~]$ openssl pkcs12 -export -clcerts -inkey kubecfg.key -in
kubecfg.crt -out kubecfg.p12 -name "kubernetes-client"
```

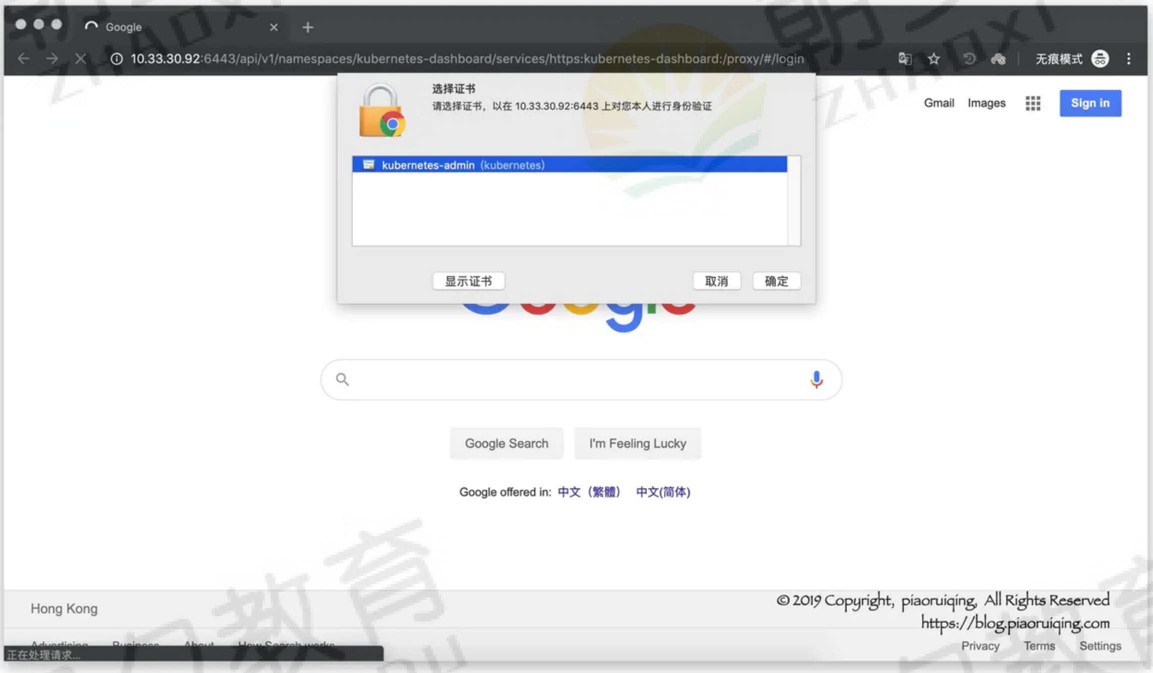
第三条命令生成证书时会提示输入密码, 可以直接两次回车跳过.

`kubecfg.p12` 即需要导入客户端机器的证书. 将证书拷贝到客户端机器上, 导入即可.

```
~$ scp root@192.168.3.222:/root/.kube/kubecfg.p12 ./
```

- 需要注意的是: 若生成证书时跳过了密码, 导入时提示填写密码直接回车即可, 不要纠结密码哪来的 (ﾟ▽ﾟ)/

此时我们可以登录面板了, 访问地址: `https://192.168.3.222:30443`, 登录时会提示选择证书, 确认后会提示输入当前用户名密码(注意是电脑的用户名密码).



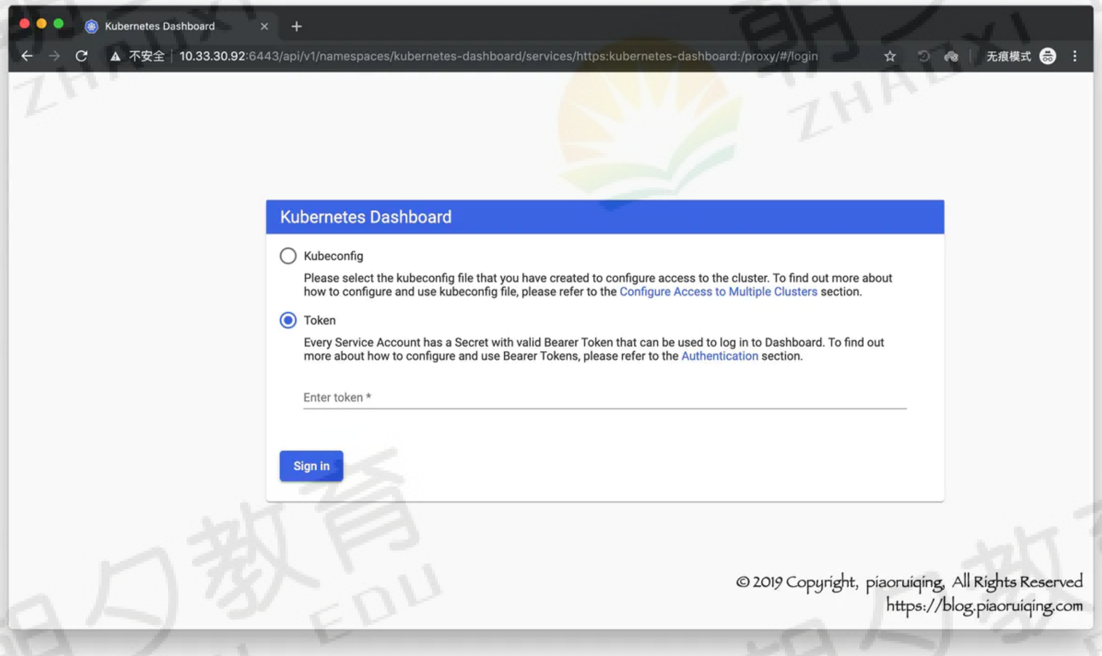
Google
10.33.30.92:6443/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/#/login
无痕模式
Gmail
Images
Sign in
Google Search
I'm Feeling Lucky
Google offered in: 中文（繁體） 中文(简体)
Hong Kong
Privacy
Terms
Settings
正在处理请求...

选择证书

请选择证书，以在 10.33.30.92:6443 上对您本人进行身份验证

kubernetes-admin (kubernetes)

显示证书 取消 确定



Kubernetes Dashboard
不安全 | 10.33.30.92:6443/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/#/login
无痕模式

Kubernetes Dashboard

○ Kubeconfig

Please select the kubeconfig file that you have created to configure access to the cluster. To find out more about how to configure and use kubeconfig file, please refer to the Configure Access to Multiple Clusters section.

● Token

Every Service Account has a Secret with valid Bearer Token that can be used to log in to Dashboard. To find out more about how to configure and use Bearer Tokens, please refer to the Authentication section.

Enter token *

Sign in

登录（Token登录）

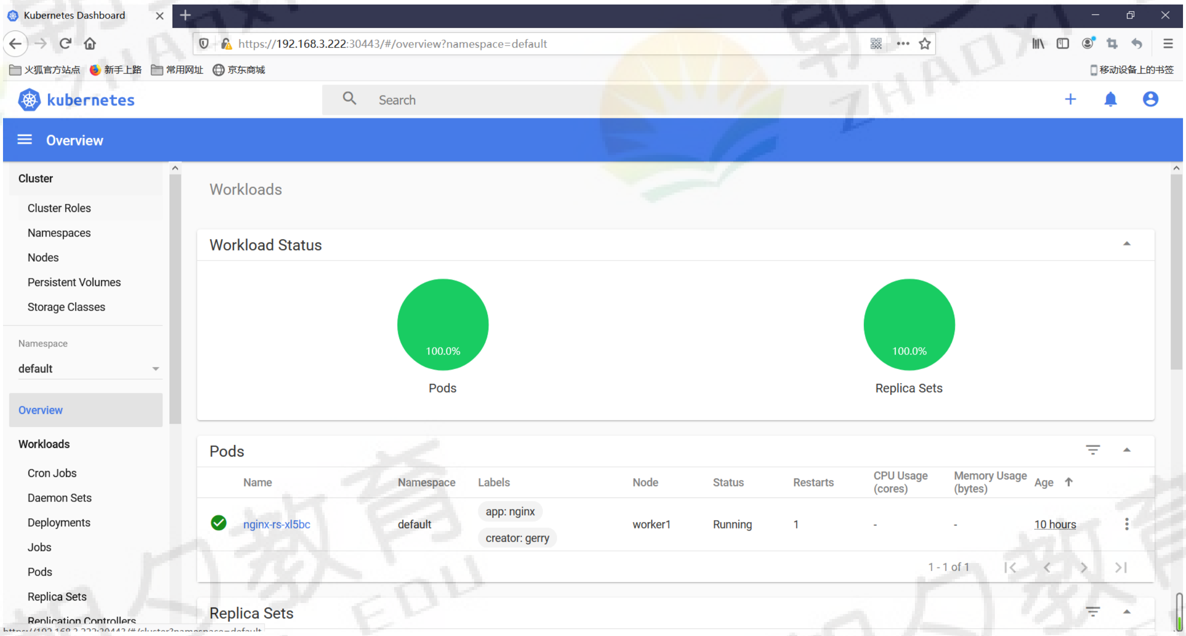
> 文档地址:Bearer Token

执行`kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep admin-user | awk '{print $1}')`，获取Token.

```
[root@k8s-master .kube]$ kubectl -n kube-system describe secret $(kubectl -n
kube-system get secret | grep admin-user | awk '{print $1}')
Name:         admin-user-token-dhhkb
Namespace:    kube-system
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: admin-user
              kubernetes.io/service-account.uid: b20d1143-ce94-4379-9e14-
8f80f06d8479

Type:  kubernetes.io/service-account-token
```

```
Data
====
ca.crt:     1025 bytes
namespace:  11 bytes
token:eyJhbGciOiJSUzI1NiIsImtpZCI6Illlz3poR2xVUm9jTGlqVXlYMFVaVDZ1VTVHQTVHXzNOMm
5qQlJnRkoOdEEifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5
pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsImt1YmVybmV0ZXMuaW8vc2V
ydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJhZG1pbi11c2VyLXRva2VuLXFiZzI5Iiwia3ViZXJuZXR
lcy5pby9zZXJ2aWNlYWNjb3VudC9zZXJ2aWNlLWFjY291bnQubmFtZSI6ImFkbWluLXVzZXIiLCJrdWJ
lcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC51aWQiOiI3YjEyN2UyMy0zMmI
OLTRmNzQtYTBkZC0OYWU3MzBlM2QyMmYiLCJzdWIiOiJzeXN0ZW06c2VydmljZWFjY291bnQ6a3ViZS1
zeXN0ZW06YWRtaW4tdXNlciJ9.QmodxUrJWyfqqodaWMjuhj5MsIslIZ0YhoZTnmdGXC6nWCCUb8SG_B
nddHA_zBcmyid03Mv4u3tAjUyLVx9UJ-841z3DWImpAR1AaMMyWJ-
QGLPYvJR7ddNF3TxZrWjCfT042MTxBSs1MTY-
XivBGOWf_O4nCPebORSR_lp9Ym9hjvRcYLJbWxUSEbTrnCkKR2Nh67jSgO1KpuLEPzm_93FkOXbQHtCb
PWhwsw0fOdjONYx9GMzTjLnBQCt_4M5kkQ5NwGRbrs3R9mL7x3mLcAzr72EQBbAz0lIH0iyLRFKhffNj
_xuDsP-Ar9NcvFNHVSsDoVqFsoX4QV8XQIfc9w
```

输入https://192.168.3.233:30443，复制该Token到登录页, 点击登录即可, 效果如下:



	Name	Namespace	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Age ↑
✓	nginx-rs-xl5bc	default	app: nginx creator: gerry	worker1	Running	1	-	-	10 hours

1 - 1 of 1

三、添加Worker节点

重复执行前面的通用操作， 初始化一个Worker机器.

执行如下命令将Worker加入集群:

```
# 临时关闭（宿主机重启后k8s不会自动部署，需要手动关闭）
swapoff -a

kubeadm join 192.168.3.233:6443 --token 5sn8j0.vhokv4v08xwrobbm     --discovery-
token-ca-cert-hash
sha256:4158c5823bc89d10a310533473f506342d93ee9255c7d9331300bf5fe4251ceb
```

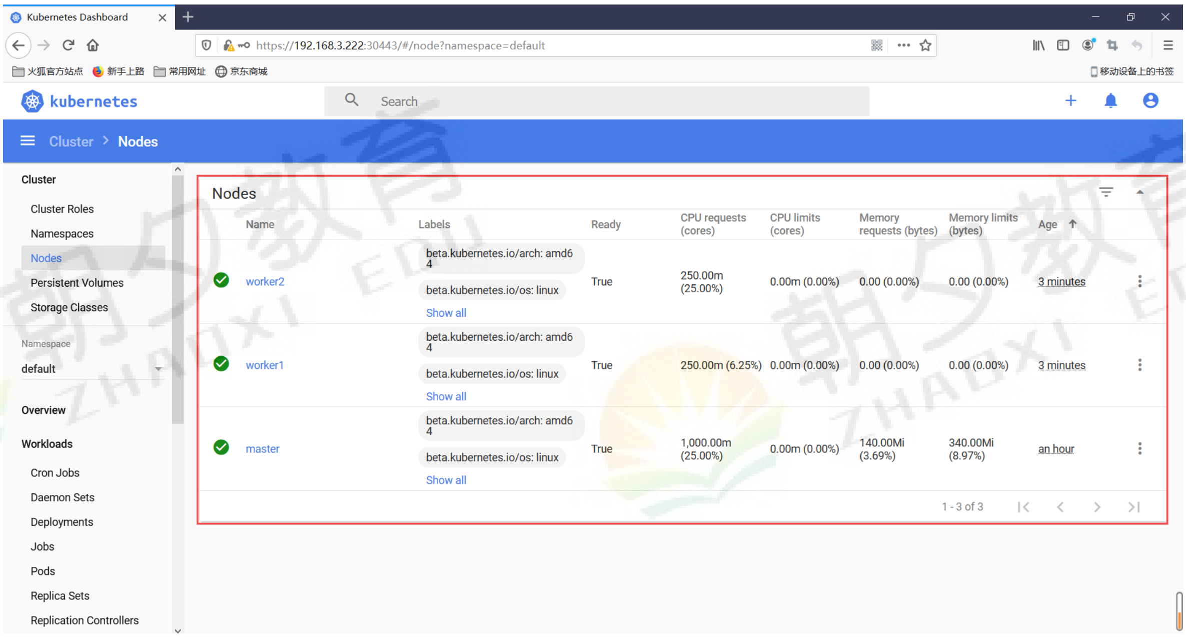
- 注意: 此处的秘钥是初始化Master后生成的, 参考前文.

```
有关 token 的过期时间是24小时
certificate-key 过期时间是2小时

如果是不记得，请执行以下命令获取
1. 在master节点执行kubeadm token list获取token（注意查看是否过期）

2. 如果没有--discovery-token-ca-cert-hash值，也可以通过以下命令获取
openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -pubin -outform der 2>/dev/null | openssl dgst -sha256 -hex | sed 's/^.* //'

如果是过期了，需要重新生成
# 如果是添加 worker 节点，不需要执行这一步，直接使用上面返回的 join 命令加入集群。
1. 执行kubeadm token create --print-join-command，重新生成，重新生成基础的 join 命令（对于添加 master 节点还需要重新生成certificate-key，见下一步）

# 添加 master 节点：用上面第1步生成的 join 命令和第2步生成的--certificate-key 值拼接起来执行
2. 使用 kubeadm init phase upload-certs --experimental-upload-certs 重新生成certificate-key
```

添加完毕后, 在Master上查看节点状态:

```
[root@k8s-master ~]$ kubectl get node
NAME      STATUS   ROLES    AGE   VERSION
master    Ready    master   98m   v1.18.0
worker1   Ready    <none>   78s   v1.18.8
worker2   Ready    <none>   28s   v1.18.8
```

在面板上也可查看:



Nodes

Name	Labels	Ready	CPU requests (cores)	CPU limits (cores)	Memory requests (bytes)	Memory limits (bytes)	Age
worker2	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/os: linux Show all	True	250.00m (25.00%)	0.00m (0.00%)	0.00 (0.00%)	0.00 (0.00%)	3 minutes
worker1	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/os: linux Show all	True	250.00m (6.25%)	0.00m (0.00%)	0.00 (0.00%)	0.00 (0.00%)	3 minutes
master	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/os: linux Show all	True	1,000.00m (25.00%)	0.00m (0.00%)	140.00Mi (3.69%)	340.00Mi (8.97%)	an hour

1 - 3 of 3

四、各种实际操作

部署配置准备

1 Pod配置 nginx-pods.yml（单独创建Pod）

kubectl apply -f nginx-pods.yml

kubectl delete pods --all

```
apiVersion: v1
kind: Pod
metadata:
  name: mynginx-pod1
  labels:
    app: mynginx
    env: test
spec:
  containers:
  - name: nginx
    image: mynginx
    imagePullPolicy: IfNotPresent
    ports:
    - name: http
      containerPort: 80
---
apiVersion: v1
kind: Pod
metadata:
  name: mynginx-pod2
  labels:
    app: mynginx
    env: test
    creator: gerry
spec:
  containers:
  - name: mginx
    image: mynginx:v1
    imagePullPolicy: IfNotPresent
    ports:
    - name: http
      containerPort: 80
```

2 RS配置 nginx-rs.yml(通过副本创建Pod)

kubectl apply -f nginx-rs.yml

kubectl delete -n default replicaset mynginx-rs

ReplicaSet

```
apiversion: apps/v1    #api版本定义 //extendsion/v1bate1
kind: ReplicaSet       #定义资源类型为ReplicaSet
metadata:              #元数据定义
  name: replicaset-example
spec:                  #ReplicaSet的规格定义
  replicas: 2          #定义副本数量为2个
  selector:            #标签选择器，定义匹配pod的标签
    matchLabels:
      app: nginx
  template:            #pod的模板定义
    metadata:          #pod的元数据定义
      labels:          #定义pod的标签，需要和上面定义的标签一致，也可以多出其他标签
```

```
        app: nginx
    spec:              #pod的规格定义
      containers:      #容器定义
      - name: nginx    #容器名称
        image: nginx   #容器镜像
```

Example:

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: mynginx-rs
  labels:
    app: mynginx
    env: test
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mynginx
      env: test
  template:
    metadata:
      labels:
        app: mynginx
        env: test
        creator: gerry
    spec:
      containers:
      - name: nginx
        image: nginx
        imagePullPolicy: IfNotPresent
        ports:
        - name: http
          containerPort: 80
```

3 Service配置 nginx-service.yml

kubectl apply -f nginx-service.yml

```
apiVersion: v1
kind: Service
metadata:
 name: nginx-service
 labels:
    app: mynginx
    env: test
spec:
 type: NodePort
 ports:
 - port: 80
   targetPort: 80
   nodePort: 30001
 selector:
   app: mynginx
   env: test
```

开始部署

```
kubectl apply -f nginx-pod.yml
kubectl apply -f nginx-rc.yml
kubectl apply -f nginx-service.yml
```

4 基于Deployment部署(官方推荐方式)

部署MySQL

```
apiVersion: apps/v1                             # apiserver的版本
kind: Deployment                                # 副本控制器deployment，管理pod和RS
metadata:
  name: mysqld                                  # deployment的名称，全局唯一
spec:
  replicas: 3                                   # Pod副本期待数量
  selector:
    matchLabels:                                # 定义RS的标签
      app: mysql                                # 符合目标的Pod拥有此标签
  strategy:                                     # 定义升级的策略
    type: RollingUpdate                         # 滚动升级，逐步替换的策略
  template:                                     # 根据此模板创建Pod的副本（实例）
    metadata:
      labels:
        app: mysql                              # Pod副本的标签，对应RS的Selector
    spec:
      nodeName: mysp1                            # 指定pod运行在的node
      containers:                               # Pod里容器的定义部分
        - name: mysql                           # 容器的名称
          image: mysql:5.7                      # 容器对应的docker镜像
          volumeMounts:                         # 容器内挂载点的定义部分
            - name: time-zone                   # 容器内挂载点名称
              mountPath: /etc/localtime         # 容器内挂载点路径，可以是文件或目录
            - name: mysql-data
              mountPath: /var/lib/mysql         # 容器内mysql的数据目录
            - name: mysql-logs
              mountPath: /var/log/mysql         # 容器内mysql的日志目录
          ports:
            - containerPort: 3306               # 容器暴露的端口号
          env:                                  # 写入到容器内的环境容量
            - name: MYSQL_ROOT_PASSWORD         # 定义了一个mysql的root密码的变量
              value: "123"
      volumes:                                  # 本地需要挂载到容器里的数据卷定义部分
        - name: time-zone                       # 数据卷名称，需要与容器内挂载点名称一
致
          hostPath:
            path: /etc/localtime                # 挂载到容器里的路径，将localtime文
件挂载到容器里，可让容器使用本地的时区
        - name: mysql-data
          hostPath:
            path: /data/mysql/data              # 本地存放mysql数据的目录
        - name: mysql-logs
          hostPath:
            path: /data/mysql/logs              # 本地存入mysql日志的目录
```

```yaml
apiversion: apps/v1
kind: Deployment
metadata:
  name: auth-db-mysql-deployment
  labels:
    app: auth-db-mysql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: auth-db-mysql
  template:
    metadata:
      labels:
        app: auth-db-mysql
        env: testing
    spec:
      containers:
      - name: auth-db-mysql
        image: mysql:5.7
        ports:
        - name: mysql-port
          containerPort: 3306
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: root
        - name: MYSQL_DATABASE
          value: auth_db
---
apiversion: v1
kind: Service
metadata:
  name: mysql-service
spec:
  type: NodePort
  ports:
    - port: 3306
      targetPort: 3306
      nodePort: 30006
  selector:
    app: auth-db-mysql
    env: testing
```

五、harber私有仓库搭建

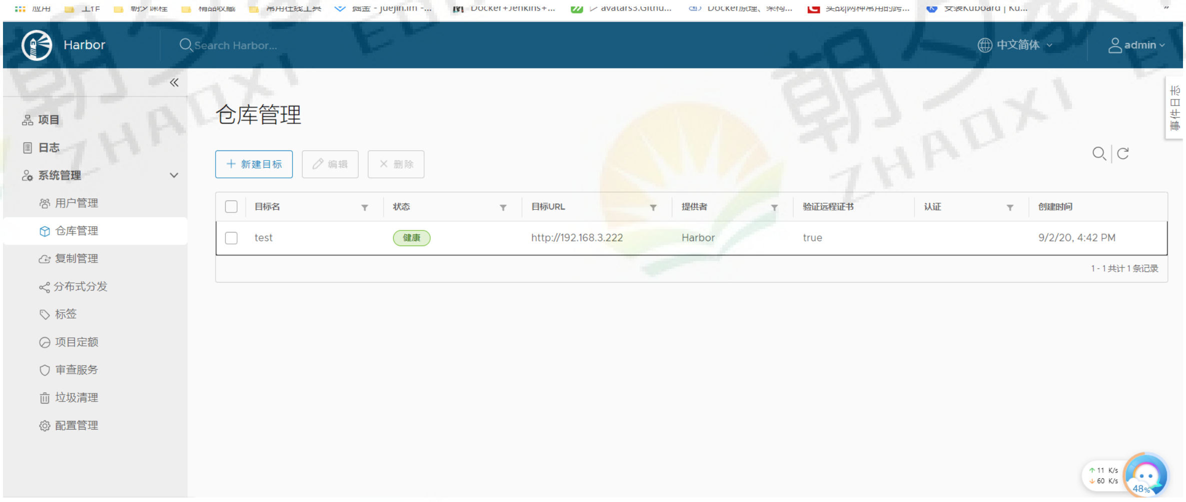
管理界面部署

Hbarber下载

- 下载下来之后解压缩，目录下会有harbor.yml.tmpl，把文件重命名为harbor.yml,就是Harbor的配置文件了。

```
# 配置Harbor，只改动了两处

hostname: 192.168.3.222 # 改为自己本机IP
http:
  port: 80
#https: # 注释掉下面三行，不注释也可以，但要配置certificate和private_key
  #port: 443
  #certificate: /your/certificate/path
  #private_key: /your/private/key/path
harbor_admin_password: Harbor12345
```

- 配置Harbor
  配置完成之后执行./install.sh。Harbor就回根据当前目录下的docker-compose.yml下载依赖的镜像
- 启动Harbor
  安装完成之后，打开浏览器访问你上边配置的hostname属性，就能看到Harbor的界面了。



仓库管理

目标名	状态	目标URL	提供者	验证远程证书	认证	创建时间
test	健康	http://192.168.3.222	Harbor	true		9/2/20, 4:42 PM

1 - 1 共计 1 条记录

私有仓库的使用

- 登录私有仓库

```
[root@master harbor]# docker login 192.168.3.222
Username: admin
Password:
Error response from daemon: Get https://192.168.3.222/v2/: dial tcp
192.168.3.222:443: connect: connection refused
[xxxxxxxxxx [root@master harbor]# docker login 192.168.3.222Username:
adminPassword:Error response from daemon: Get https://192.168.3.222/v2/:
dial tcp 192.168.3.222:443: connect: connection refused[docker login
192.168.3.222bash
## 解决上面问题 需要配置本地仓库，在/etc/docker/daemon.json文件
{
    "insecure-registries": ["192.168.3.222"]
}
---------------------------------------------------------------------
systemctl daemon-reload
systemctl restart docker
--------------------------------------------
# 进入harbor根目录执行
docker-compose stop
```

```
docker-compose start
```

- 完成镜像推送
  - 在管理界面创建一个项目
  - 在把镜像进行重新打标签

    ```
    #镜像打标签
    [root@centos7 ~]#docker tag 镜像名:标签 私服地址/仓库项目名/镜像名:标签
    #推送到私服
    [root@centos7 ~]#docker push  私服地址/仓库项目名/镜像名：标签
    #从私服拉取镜像
    [root@centos7 ~]#docker pull 私服地址/仓库项目名/镜像名：标签
    ```

- 客户端链接私服仓库并下载镜像

  ```
  在/etc/docker/daemon.json文件添加私服仓库地址
  {
      "insecure-registries": ["192.168.3.222"]
  }

  root@lyg:~# systemctl daemon-reload
  root@lyg:~# systemctl restart docker

  docker login 192.168.3.222
  docker pull 私服仓库地址/项目名/镜像名:标签
  ```

六、附录:

1.Kubectl基本操作

查询所有的集群节点

```
kubectl get pods
```

查询所有命名空间下面的pod

```
kubectl get pods --all-namespaces
```

2.ReplicaSet清单

可以通过以下命令查看ReplicaSet资源清单规则:

```
kubectl explain rs
kubectl explain rs.spec
kubectl explain rs.spec.template
```

创建`ReplicaSet`资源:

```
kubectl apply -f nginx-pods.yml
kubectl apply -f mynginx-rs.yaml
```

查看`ReplicaSet`和`Pod`信息:

```
kubectl get rs -o wide
kubectl describe rs mynginx-rs
kubectl get pod -o wide --show-labels
```

验证这些`Pod`的所有者引用是否为`mynginx-rs`，查看其中一个`Pod`的Yaml:

```
kubectl get pod mynginx-rs-jrv2t -o yaml
```

查看某个Pod的详细信息

```
 kubectl describe pod -n 命名空间名 pod的名称
 eg: kubectl describe pod -n kubernetes-dashboard dashboard-metrics-scraper-66b49655d4-lt6qd
```

删除某个指定的Pod

```
kubectl delete pod -n nginxspace mynginx-rs-jrv2t
```

执行以下命令将`Pod`副本收缩/扩容至3个:

```
kubectl scale deployment net5-eleven-deployment  --replicas=3

kubectl scale ReplicaSet net5-rs  --replicas=5
kubectl scale ReplicaSet net5-rs  --replicas=2

kubectl set image deployment/net5-eleven-deployment net5-eleven=registry.cn-hangzhou.aliyuncs.com/clay_core/netcore:net5demovip-2 --record=true
```

3.卸载部署程序

```
kubectl delete deployments --all
kubectl delete services/svc --all
kubectl get pods
kubectl get services
kubectl get deployments
```

4.完整卸载K8S

```
kubectl delete node --all # 删除所有的节点
kubeadm reset # 重置kubeadm
modprobe -r ipip
lsmod
rm -rf ~/.kube/
rm -rf /etc/kubernetes/
rm -rf /etc/systemd/system/kubelet.service.d
rm -rf /etc/systemd/system/kubelet.service
rm -rf /usr/bin/kube*
rm -rf /etc/cni
rm -rf /opt/cni
rm -rf /var/lib/etcd
rm -rf /var/etcd
yum remove kube*
```

```
## 启动所有停止的Docker容器
docker start $(docker ps -a | awk '{ print $1}' | tail -n +2)

关闭实例，自动启动---关闭docker进程，无效---关闭虚拟机生效
#关闭docker进程
systemctl stop docker
systemctl start docker
```

七、K8S滚动发布

Deployment为Pod和Replica Set提供声明式更新，并维持期望状态。

```
spec:
  ...
  minReadySeconds: 100  # 这里需要估一个比较合理的值，从容器启动到应用正常提供服务
  strategy:  # k8s 默认的 strategy 就是 RollingUpdate， 这里写明出来可以调节细节参数
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1  # 更新时允许最大激增的容器数，默认 replicas 的 1/4 向上取整
      maxUnavailable: 0  # 更新时允许最大 unavailable 容器数，默认 replicas 的 1/4 向下取整
```

命令行用patch修改配置
kubectl patch deployment nginx-test -p ‘{“spec”:{“minReadySeconds”:30}}’ -n test

修改镜像并打记录，便于回滚指定版本
kubectl set image deployment/nginx-test nginx=nginx:1.15 --record=true --namespace=test

查看发布历史
kubectl rollout history deployment/nginx-test -n test

回滚上一版本
kubectl rollout undo deployment/nginx-test -n test

回滚指定版本
kubectl rollout undo deployment nginx-test --to-revision=13 -n test

将资源标记为暂停
kubectl rollout pause deployment/nginx-test -n test

查看资源的状态
kubectl rollout status deployment/nginx-test -n test

恢复已暂停的资源
kubectl rollout resume deployment/nginx-test -n test