一、我的环境

- 操作系统: windows 10
- 虚拟机: VMware 16 Pro
- Linux: CentOS 8.3
- 内核: 4.18.0-240.el8.x86_64 (`uname -r`)
- Master: 172.20.54.226
- Docker: 19.03.5-3.el7
- Kubeadm: v1.20.1
- Kubectl: v1.20.1
- Kubelet: v1.20.1
- KubernetesDashboard: v2.0.0

二、环境准备

> 正式环境需要至少4-5台服务器方能实现有效集群, 此处用于学习, 只是搭建了个伪集群: Master、Node两个节点

1. 各节点分别执行相应命令

```
hostnamectl set-hostname master
```

```
hostnamectl set-hostname node
```

2. 所有节点执行地址映射命令

```
cat <<EOF > /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4
localhost4.localdomain4
::1         localhost localhost.localdomain localhost6
localhost6.localdomain6
172.20.54.226 master
172.20.54.108 node
EOF
```

3. 所有节点关闭防火墙

```
setenforce  0
sed -i "s/^SELINUX=enforcing/SELINUX=disabled/g" /etc/sysconfig/selinux
sed -i "s/^SELINUX=enforcing/SELINUX=disabled/g" /etc/selinux/config
sed -i "s/^SELINUX=permissive/SELINUX=disabled/g" /etc/sysconfig/selinux
sed -i "s/^SELINUX=permissive/SELINUX=disabled/g" /etc/selinux/config

systemctl disable firewalld
systemctl stop firewalld

swapoff -a
sed -i 's/.*swap.*/#&/' /etc/fstab
```

4. 所有节点加载模块

```
modprobe br_netfilter
sh -c 'echo "br_netfilter" > /etc/modules-load.d/br_netfilter.conf'
```

5. 所有节点配置内核参数

> 执行以下命令后, 使用: `ulimit -Hn` 验证结果为655360, 否则重新连接客户端

```
cat > /etc/sysctl.d/k8s.conf <<EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
sysctl -p /etc/sysctl.d/k8s.conf
echo "* soft nofile 655360" >> /etc/security/limits.conf
echo "* hard nofile 655360" >> /etc/security/limits.conf
echo "* soft nproc 655360"  >> /etc/security/limits.conf
echo "* hard nproc 655360"  >> /etc/security/limits.conf
echo "* soft  memlock  unlimited"  >> /etc/security/limits.conf
echo "* hard memlock  unlimited"  >> /etc/security/limits.conf
echo "DefaultLimitNOFILE=1024000"  >> /etc/systemd/system.conf
echo "DefaultLimitNPROC=1024000"  >> /etc/systemd/system.conf
```

6. 所有节点配置国内镜像源

```
rm -rf  /etc/yum.repos.d/*
wget -O /etc/yum.repos.d/CentOS-Base.repo
http://mirrors.aliyun.com/repo/Centos-8.repo
wget -O /etc/yum.repos.d/epel.repo  http://mirrors.aliyun.com/repo/epel-
7.repo
```

```
cat <<EOF> /etc/yum.repos.d/kubernetes.repo
[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
```

```
yum clean all && yum makecache
```

7. 所有节点安装依赖

```
yum install  -y conntrack ipvsadm ipset jq sysstat curl iptables libseccomp
bash-completion yum-utils device-mapper-persistent-data lvm2 net-tools
conntrack-tools vim libtool-ltdl
```

8. 同步各节点时间

```
yum -y install chrony
systemctl enable chronyd.service && systemctl start chronyd.service &&
systemctl status chronyd.service
chronyc sources
```

9. 配置节点互信 (至少master)

```
ssh-keygen # 全部回车
ssh-copy-id  node # 输入node节点密码
```

三、部署Docker

1. 所有节点删除旧版本

```
yum remove -y docker docker-ce docker-common docker-selinux docker-engine
```

2. 所有节点设置Docker国内镜像源

```
yum-config-manager --add-repo http://mirrors.aliyun.com/docker-
ce/linux/centos/docker-ce.repo
```

3. 所有节点安装新版容器

```
dnf install --allowerasing http://mirrors.aliyun.com/docker-
ce/linux/centos/8/x86_64/stable/Packages/containerd.io-1.4.3-
3.1.el8.x86_64.rpm
```

4. 所有节点安装指定版本Docker

```
yum install -y docker-ce-19.03.5-3.el7.x86_64
```

5. 所有节点配置加速器和存放路径

```
mkdir -p /etc/docker/
vim /etc/docker/daemon.json
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "registry-mirrors": ["https://q2hy3fzi.mirror.aliyuncs.com"],
  "graph": "/tol/docker-data"
}
systemctl restart docker
```

6. 所有节点启动Docker

```
systemctl daemon-reload && systemctl restart docker && systemctl enable
docker && systemctl status docker
```

7. 所有节点安装并启动k8s部署工具

```
yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
systemctl enable kubelet && systemctl start kubelet
```

四、部署Kubeadm

1. 所有节点生成部署工具默认配置

```
kubeadm config print init-defaults > kubeadm.conf
sed -i "s/imageRepository: .*/imageRepository:
registry.aliyuncs.com\/google_containers/g" kubeadm.conf
```

2. 所有节点安装指定版本Kubernetes

```
sed -i "s/kubernetesVersion: .*/kubernetesVersion: v1.20.0/g" kubeadm.conf
kubeadm config images pull --config kubeadm.conf
```

3. 所有节点防屏蔽

> 如果不打tag变成k8s.gcr.io，那么后面用kubeadm安装会出现问题:
>
> 因为 kubeadm里面只认 google自身的模式, 我们执行下面命令即可完成tag标识更换

```
vim tag.sh
```

```
#!/bin/bash

newtag=k8s.gcr.io
for i in $(docker images | grep -v TAG |awk '{print $1 ":" $2}')
do
   image=$(echo $i | awk -F '/' '{print $3}')
   docker tag $i $newtag/$image
   docker rmi $i
done
```

```
bash tag.sh
```

五、部署Master

1. 使用部署工具初始化Master节点

> 记得将172.20.54.226改成自己Master的IP

```
kubeadm init --kubernetes-version=v1.20.0 --pod-network-cidr=172.22.0.0/16 -
-apiserver-advertise-address=172.20.54.226
```

2. 测试验证

```
mkdir -p /root/.kube
cp /etc/kubernetes/admin.conf /root/.kube/config
```

> 获取pods列表命令: 其中coredns pod还处于`Pending`状态 (此处贴的是最终效果)

```
# kubectl get pods --all-namespaces
NAMESPACE     NAME                                    READY   STATUS
RESTARTS   AGE
kube-system   calico-kube-controllers-7854b85cf7-5fnv8     1/1     Running
1          4h
kube-system   calico-node-tk4xs                           1/1     Running
1          4h
kube-system   calico-node-tpwtk                           1/1     Running
4          4h
kube-system   coredns-74ff55c5b-9pvr6                     1/1     Running
1          5h25m
kube-system   coredns-74ff55c5b-kfvsb                     1/1     Running
1          5h25m
kube-system   dashboard-metrics-scraper-5587f78f94-88b8h   1/1     Running
0          57m
kube-system   etcd-master                                 1/1     Running
6          5h25m
kube-system   kube-apiserver-master                       1/1     Running
7          5h25m
kube-system   kube-controller-manager-master              1/1     Running
7          4h46m
kube-system   kube-proxy-hvq8c                            1/1     Running
1          4h48m
kube-system   kube-proxy-xmltp                            1/1     Running
4          5h25m
kube-system   kube-scheduler-master                       1/1     Running
8          4h45m
kube-system   kubernetes-dashboard-68486945bd-k4rp8       1/1     Running
0          58m
```

3. 查看集群状态

```
# kubectl get cs
etcd-0               Healthy   {"health":"true"}
controller-manager   Healthy   ok
scheduler            Healthy   ok
```

六、部署Node

1. 下载安装镜像

```
docker pull registry.aliyuncs.com/google_containers/kube-proxy:v1.13.0
docker pull registry.aliyuncs.com/google_containers/pause:3.1
docker pull calico/node:v3.1.4
docker pull calico/cni:v3.1.4
docker pull calico/typha:v3.1.4

docker tag registry.aliyuncs.com/google_containers/kube-proxy:v1.13.0
k8s.gcr.io/kubeproxy:v1.13.0
docker tag registry.aliyuncs.com/google_containers/pause:3.1
k8s.gcr.io/pause:3.1
docker tag calico/node:v3.1.4 quay.io/calico/node:v3.1.4
docker tag calico/cni:v3.1.4 quay.io/calico/cni:v3.1.4
docker tag calico/typha:v3.1.4 quay.io/calico/typha:v3.1.4
```

```
docker rmi registry.aliyuncs.com/google_containers/kube-proxy:v1.13.0
docker rmi registry.aliyuncs.com/google_containers/pause:3.1
docker rmi calico/node:v3.1.4
docker rmi calico/cni:v3.1.4
docker rmi calico/typha:v3.1.4
```

2. 在Master节点上获取加入集群的命令

> 符号``内的命令即可在所有Node节点执行(下一步)

```
# kubeadm token create --print-join-command
`kubeadm join 172.20.54.226:6443 --token 16l83a.e1tpcgkmze0i3fuy --
discovery-token-ca-cert-hash
sha256:233a3d9c6c0ed466642c08293e0bf2bb217359d414d3ccb0bf25afa1c00b7ca3`
```

3. 在Node节点上执行获取到的: 加入集群命令

```
# kubeadm join 172.20.54.226:6443 --token 16l83a.e1tpcgkmze0i3fuy --
discovery-token-ca-cert-hash
sha256:233a3d9c6c0ed466642c08293e0bf2bb217359d414d3ccb0bf25afa1c00b7ca3
```

七、部署Calico

> 该项操作只需要在Master节点上操作

1. 下载安装标识calico镜像

```
docker pull calico/node:v3.1.4
docker pull calico/cni:v3.1.4
docker pull calico/typha:v3.1.4
docker tag calico/node:v3.1.4 quay.io/calico/node:v3.1.4
docker tag calico/cni:v3.1.4 quay.io/calico/cni:v3.1.4
docker tag calico/typha:v3.1.4 quay.io/calico/typha:v3.1.4
docker rmi calico/node:v3.1.4
docker rmi calico/cni:v3.1.4
docker rmi calico/typha:v3.1.4
```

2. 安装权限配置

```
curl https://docs.projectcalico.org/v3.1/getting-
started/kubernetes/installation/hosted/rbac-kdd.yaml -O
kubectl apply -f rbac-kdd.yaml
```

3. 修改calico配置

```
curl https://docs.projectcalico.org/v3.1/getting-
started/kubernetes/installation/hosted/kubernetes-datastore/policy-
only/1.7/calico.yaml -O
```

> 修改typha_service_name: calico-typha

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: calico-config
  namespace: kube-system
data:
  typha_service_name: "calico-typha" # 原none
```

> 修改apiVersion: apps/v1、 replicas: 1等

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: calico-typha
  namespace: kube-system
  labels:
    k8s-app: calico-typha
spec:
  replicas: 1
  revisionHistoryLimit: 2
```

```
- name: CALICO_IPV4POOL_CIDR
  value: "172.22.0.0/16"
```

```
- name: CALICO_NETWORKING_BACKEND
  value: "bird"
```

4. 部署calico网络

```
# kubectl apply -f calico.yaml
# kubectl get pods --all-namespaces
# kubectl get nodes
NAME     STATUS   ROLES                  AGE     VERSION
master   Ready    control-plane,master   5h43m   v1.20.1
node     Ready    <none>                 5h5m    v1.20.1
```

八、部署Dashboard

1. 下载镜像

```
docker pull kubernetesui/dashboard:v2.0.0
docker pull kubernetesui/metrics-scraper:v1.0.4
```

2. 部署权限服务

> vim dashboard-rbac.yaml

```
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
```

```
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kube-system
rules:
  - apiGroups: [""]
    resources: ["secrets"]
    resourceNames: ["kubernetes-dashboard-key-holder", "kubernetes-
dashboard-certs", "kubernetes-dashboard-csrf"]
    verbs: ["get", "update", "delete"]
  - apiGroups: [""]
    resources: ["configmaps"]
    resourceNames: ["kubernetes-dashboard-settings"]
    verbs: ["get", "update"]
  - apiGroups: [""]
    resources: ["services"]
    resourceNames: ["heapster", "dashboard-metrics-scraper"]
    verbs: ["proxy"]
  - apiGroups: [""]
    resources: ["services/proxy"]
    resourceNames: ["heapster", "http:heapster:", "https:heapster:",
"dashboard-metrics-scraper", "http:dashboard-metrics-scraper"]
    verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
rules:
  - apiGroups: ["metrics.k8s.io"]
    resources: ["pods", "nodes"]
    verbs: ["get", "list", "watch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kube-system
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: kubernetes-dashboard
subjects:
  - kind: ServiceAccount
    name: kubernetes-dashboard
    namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
```

```
metadata:
  name: kubernetes-dashboard
  namespace: kube-system
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: kubernetes-dashboard
subjects:
  - kind: ServiceAccount
    name: kubernetes-dashboard
    namespace: kube-system
```

```
# kubectl apply -f dashboard-rbac.yaml
```

3. 部署密钥服务

> vim dashboard-configmap-secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard-certs
  namespace: kube-system
type: Opaque
---
apiVersion: v1
kind: Secret
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard-csrf
  namespace: kube-system
type: Opaque
data:
  csrf: ""
---
apiVersion: v1
kind: Secret
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard-key-holder
  namespace: kube-system
type: Opaque
---
kind: ConfigMap
apiVersion: v1
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard-settings
  namespace: kube-system
```

```
# kubectl apply -f dashboard-configmap-secret.yaml
```

4. 部署控制台服务

> vim dashboard-deploy.yaml

```
## Dashboard Service
kind: Service
apiVersion: v1
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kube-system
spec:
  type: NodePort
  ports:
    - port: 443
      nodePort: 30001
      targetPort: 8443
  selector:
    k8s-app: kubernetes-dashboard
---
## Dashboard Deployment
kind: Deployment
apiVersion: apps/v1
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kube-system
spec:
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      k8s-app: kubernetes-dashboard
  template:
    metadata:
      labels:
        k8s-app: kubernetes-dashboard
    spec:
      serviceAccountName: kubernetes-dashboard
      containers:
        - name: kubernetes-dashboard
          image: kubernetesui/dashboard:v2.0.0
          securityContext:
            allowPrivilegeEscalation: false
            readOnlyRootFilesystem: true
            runAsUser: 1001
            runAsGroup: 2001
          ports:
            - containerPort: 8443
              protocol: TCP
          args:
            - --auto-generate-certificates
            - --namespace=kube-system          #设置为当前部署的Namespace
```

```
          resources:
            limits:
              cpu: 1000m
              memory: 512Mi
            requests:
              cpu: 1000m
              memory: 512Mi
          livenessProbe:
            httpGet:
              scheme: HTTPS
              path: /
              port: 8443
            initialDelaySeconds: 30
            timeoutSeconds: 30
          volumeMounts:
            - name: kubernetes-dashboard-certs
              mountPath: /certs
            - name: tmp-volume
              mountPath: /tmp
            - name: localtime
              readOnly: true
              mountPath: /etc/localtime
      volumes:
        - name: kubernetes-dashboard-certs
          secret:
            secretName: kubernetes-dashboard-certs
        - name: tmp-volume
          emptyDir: {}
        - name: localtime
          hostPath:
            type: File
            path: /etc/localtime
      tolerations:
        - key: node-role.kubernetes.io/master
          effect: NoSchedule
```

```
# kubectl apply -f dashboard-deploy.yaml
```

5. 部署指标服务

> vim dashboard-metrics.yaml

```
## Dashboard Metrics Service
kind: Service
apiVersion: v1
metadata:
  labels:
    k8s-app: dashboard-metrics-scraper
  name: dashboard-metrics-scraper
  namespace: kube-system
spec:
  ports:
    - port: 8000
      targetPort: 8000
  selector:
    k8s-app: dashboard-metrics-scraper
```

```yaml
---
## Dashboard Metrics Deployment
kind: Deployment
apiVersion: apps/v1
metadata:
  labels:
    k8s-app: dashboard-metrics-scraper
  name: dashboard-metrics-scraper
  namespace: kube-system
spec:
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      k8s-app: dashboard-metrics-scraper
  template:
    metadata:
      labels:
        k8s-app: dashboard-metrics-scraper
      annotations:
        seccomp.security.alpha.kubernetes.io/pod: 'runtime/default'
    spec:
      serviceAccountName: kubernetes-dashboard
      containers:
        - name: dashboard-metrics-scraper
          image: kubernetesui/metrics-scraper:v1.0.4
          securityContext:
            allowPrivilegeEscalation: false
            readOnlyRootFilesystem: true
            runAsUser: 1001
            runAsGroup: 2001
          ports:
            - containerPort: 8000
              protocol: TCP
          resources:
            limits:
              cpu: 1000m
              memory: 512Mi
            requests:
              cpu: 1000m
              memory: 512Mi
          livenessProbe:
            httpGet:
              scheme: HTTP
              path: /
              port: 8000
            initialDelaySeconds: 30
            timeoutSeconds: 30
          volumeMounts:
          - mountPath: /tmp
            name: tmp-volume
          - name: localtime
            readOnly: true
            mountPath: /etc/localtime
      volumes:
        - name: tmp-volume
          emptyDir: {}
        - name: localtime
```

```
          hostPath:
            type: File
            path: /etc/localtime
      nodeSelector:
        "beta.kubernetes.io/os": linux
      tolerations:
        - key: node-role.kubernetes.io/master
          effect: NoSchedule
```

```
# kubectl apply -f dashboard-metrics.yaml
```

6. 部署认证服务

> vim dashboard-token.yaml

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: admin
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
roleRef:
  kind: ClusterRole
  name: cluster-admin
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: ServiceAccount
  name: admin
  namespace: kube-system
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin
  namespace: kube-system
  labels:
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
```

```
# kubectl apply -f dashboard-token.yaml
```

7. 登陆
    1. 访问链接: https://master:30001/
    2. 获取令牌: 执行以下命令

```
kubectl describe secret/$(kubectl get secret -n kube-system |grep
admin|awk '{print $1}') -n kube-system
```

```txt
eyJhbGciOiJSUzI1NiIsImtpZCI6Ikp2bV9pZmNIROxqLUxRREd3QlRzNU1pdnBkYnMxTXRlWG15alBidWOxNTAifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW5OIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJhZG1pbi10b2tlbi1zandkdiIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW5OLm5hbWUiOiJhZG1pbiIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW5OLnVpZCI6IjUxOTAxNmFkLTU3YjEtNDkzYS04ZGZiLTM2Mzg3NTIwODgwNiIsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDprdWJlLXN5c3RlbTphZG1pbiJ9.I4voTZHn83jPe7apabqOtTjsBuj0uEbkgQGu1fl2tAbbpocg89NjN-DrTkyrETa7qDVp2bmXCHbIbiJU64xlfifCgNFgOOHnWqvuMgztYnYMUpbYSRuQVumn-WCDsIxBnfK-lIbhdSGZZVS66PK4Rwlf4hQHdE_3oclzBYnoz_i11xoFaDDUhhSLxmIDuBA-HoR-n_LJRDtJEqD7VmCTiDkUECxVpIM2oQtVb-nLxuBQg7M7rsbdWFsp5MJ7f-AdRBFgszEQaezBCt4kf0Uuakl6AC_0fDGjwEo04M12Md5Q7JOkyUNKgPbw0S3p8rxuw07I_LBipTIW8Sznll_wzw
```