53、

Docker安装

使用xserver1节点，自行配置YUM源，安装docker服务（需要用到的包为xserver1节点/root目录下的Docker.tar.gz）。安装完服务后，将registry\_latest.tar上传到xserver1节点中并配置为私有仓库。要求启动registry容器时，将内部保存文件的目录映射到外部的/opt/registry目录，将内部的5000端口映射到外部5000端口。依次将启动registry容器的命令及返回结果、执行docker info命令的返回结果以文本形式提交到答题框。 (30)分

环境准备

# systemctl stop firewalld && systemctl disable firewalld

# iptables -t filter -F

# iptables -t filter -X

# sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/selinux/config

# reboot

[root@xserver1 ~]# cat >> /etc/sysctl.conf << EOF

net.ipv4.ip\_forward=1

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

EOF

[root@xserver1 ~]# modprobe br\_netfilter

[root@xserver1 ~]# sysctl -p

net.ipv4.ip\_forward = 1

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

[root@xserver1 ~]# mkdir /opt/docker/

[root@xserver1 ~]# tar -zxvf Docker.tar.gz -C /opt/docker/

修改yum文件

[root@xserver1 ~]# cat /etc/yum.repos.d/local.repo

[centos]

name=centos

baseurl=file:///opt/centos

enable=1

gpgcheck=0

[docker]

name=docker

baseurl=file:///opt/docker/Docker

enable=1

gpgcheck=0

[root@xserver1 ~]# yum install docker-ce

[root@xserver1 ~]# systemctl daemon-reload

[root@xserver1 ~]# systemctl restart docker

[root@xserver1 ~]# systemctl enable docker

进入/opt/docker/目录

[root@xserver1 docker]# ./image.sh

[root@xserver1 docker]# docker run -d -v /opt/registry:/var/lib/registry -p 5000:5000 --restart=always --name registry registry:latest

f66ef4b03a54588a9a32b82f621f1f0e6ebef117ab236eb0be82b8077bf93a14

[root@xserver1 docker]# docker info

Containers: 1

Running: 1

Paused: 0

Stopped: 0

Images: 11

Server Version: 18.09.6

Storage Driver: devicemapper

提交答案

54、

Dockerfile编写

使用xserver1节点，新建目录centos-jdk，将提供的jdk-8u141-linux-x64.tar.gz复制新建的目录，然后编辑Dockerfile文件，文件要求如下： 1.使用centos:latest基础镜像； 2.指定作为为xiandian； 3.新建文件夹/usr/local/java用于存放jdk文件； 4.将JDK文件复制到镜像内创建的目录并自动解压； 5.创建软连接：ln -s /usr/local/java/jdk1.8.0\_141 /usr/local/java/jdk； 6.设置环境变量如下 ENV JAVA\_HOME /usr/local/java/jdk ENV JRE\_HOME ${JAVA\_HOME}/jre ENV CLASSPATH .:${JAVA\_HOME}/lib:${JRE\_HOME}/lib ENV PATH ${JAVA\_HOME}/bin:$PATH 编写完毕后，构建名为centos-jdk的镜像，构建成功后，查看镜像列表。最后将Dockerfile的内容、构建镜像的操作命令、查看镜像列表的命令和返回的结果以文本形式提交到答题框。 (40)分

进入/opt/docker/jdk/目录

[root@xserver1 jdk]# mkdir centos-jdk

[root@xserver1 jdk]# mv jdk-8u141-linux-x64.tar.gz ./centos-jdk/

[root@xserver1 jdk]# cd centos-jdk/

[root@xserver1 centos-jdk]# vi Dockerfile

FROM centos

MAINTAINER Xiandian

RUN mkdir /usr/local/java

ADD jdk-8u141-linux-x64.tar.gz /usr/local/java/

RUN ln -s /usr/local/java/jdk1.8.0\_141 /usr/local/java/jdk

ENV JAVA\_HOME /usr/local/java/jdk

ENV JRE\_HOME ${JAVA\_HOME}/jre

ENV CLASSPATH .:${JAVA\_HOME}/lib:${JRE\_HOME}/lib

ENV PATH ${JAVA\_HOME}/bin:$PATH

[root@xserver1 centos-jdk]# docker build -t="centos-jdk" .

Sending build context to Docker daemon 185.5MB

Step 1/9 : FROM centos

---> 0f3e07c0138f

Step 2/9 : MAINTAINER Xiandian

---> Running in 3736d578c313

Removing intermediate container 3736d578c313

---> fa0c6d886381

Step 3/9 : RUN mkdir /usr/local/java

---> Running in 195c61df8e62

Removing intermediate container 195c61df8e62

---> ce91748992ab

Step 4/9 : ADD jdk-8u141-linux-x64.tar.gz /usr/local/java/

---> 7d70136331de

Step 5/9 : RUN ln -s /usr/local/java/jdk1.8.0\_141 /usr/local/java/jdk

---> Running in 9cdb402a35d4

Removing intermediate container 9cdb402a35d4

---> 68192956906e

Step 6/9 : ENV JAVA\_HOME /usr/local/java/jdk

---> Running in 8630213a4780

Removing intermediate container 8630213a4780

---> 12c69d704c93

Step 7/9 : ENV JRE\_HOME ${JAVA\_HOME}/jre

---> Running in 685acf52138e

Removing intermediate container 685acf52138e

---> ca7fd5191219

Step 8/9 : ENV CLASSPATH .:${JAVA\_HOME}/lib:${JRE\_HOME}/lib

---> Running in f4896e85cf4b

Removing intermediate container f4896e85cf4b

---> 9e526796f817

Step 9/9 : ENV PATH ${JAVA\_HOME}/bin:$PATH

---> Running in 3e4e73f1f462

Removing intermediate container 3e4e73f1f462

---> 77540f9e264c

Successfully built 77540f9e264c

Successfully tagged centos-jdk:latest

[root@xserver1 centos-jdk]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

centos-jdk latest 77540f9e264c 53 seconds ago 596MB

httpd latest d3017f59d5e2 12 months ago 165MB

busybox latest 020584afccce 12 months ago 1.22MB

nginx latest 540a289bab6c 12 months ago 126MB

redis alpine 6f63d037b592 12 months ago 29.3MB

python 3.7-alpine b11d2a09763f 12 months ago 98.8MB

<none> <none> 4cda95efb0e4 13 months ago 80.6MB

centos latest 0f3e07c0138f 13 months ago 220MB

registry latest f32a97de94e1 20 months ago 25.8MB

swarm latest ff454b4a0e84 2 years ago 12.7MB

httpd 2.2.32 c51e86ea30d1 3 years ago 171MB

httpd 2.2.31 c8a7fb36e3ab 3 years ago 170MB

提交答案

55、

部署K8S集群

使用xserver1、xserver2节点，自行配置好网络，安装好docker-ce。部署K8S 集群，不要求部署kubernetes-dashboard。部署K8S平台完成后，在主节点使用命令依次检查集群状态、Pods状态、各节点的状态。最后将检查状态的命令及返回结果以文本形式提交到答题框。 (50)分

#### 1.基础环境配置

##### （1）配置YUM源

所有节点将提供的压缩包K8S.tar.gz上传至/root目录并解压。

# tar -zxvf K8S.tar.gz

所有节点配置本地YUM源。

# cat /etc/yum.repod.s/local.repo

[kubernetes]

name=kubernetes

baseurl=file:///root/Kubernetes

gpgcheck=0

enabled=1

##### （2）升级系统内核

所有节点升级系统内核。

# yum upgrade -y

##### （3）配置主机映射

所有节点，修改/etc/hosts文件。

# cat /etc/hosts

127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4

::1 localhost localhost.localdomain localhost6 localhost6.localdomain6

192.168.100.11 xserver1

192.168.100.12 xserver2

##### （4）配置防火墙及SELinux

所有节点配置防火墙及SELinux。

# systemctl stop firewalld && systemctl disable firewalld

# iptables -F

# iptables -X

# iptables -Z

# /usr/sbin/iptables-save

# sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/selinux/config

# reboot

##### （5）关闭Swap

Kubernetes的想法是将实例紧密包装到尽可能接近100%。所有的部署应该与CPU和内存限制固定在一起。所以如果调度程序发送一个Pod到一台机器，它不应该使用交换。设计者不想交换，因为它会减慢速度。所以关闭Swap主要是为了性能考虑。

所有节点关闭Swap。

# swapoff -a

# sed -i "s/\/dev\/mapper\/centos-swap/\#\/dev\/mapper\/centos-swap/g" /etc/fstab

##### （5）配置时间同步

所有节点安装chrony服务。

# yum install -y chrony

Xserve1节点修改/etc/chrony.conf文件，注释默认NTP服务器，指定上游公共NTP服务器，并允许其他节点同步时间。

[root@xserver1~]# sed -i 's/^server/#&/' /etc/chrony.conf

[root@ xserver1 ~]# cat >> /etc/chrony.conf << EOF

local stratum 10

server master iburst

allow all

EOF

xserver1节点重启chronyd服务并设为开机启动，开启网络时间同步功能。

[root@ xserver1 ~]# systemctl enable chronyd && systemctl restart chronyd

[root@ xserver1~]# timedatectl set-ntp true

Node节点修改/etc/chrony.conf文件，指定内网Master节点为上游NTP服务器，重启服务并设为开机启动。

[root@ xserver2~]# sed -i 's/^server/#&/' /etc/chrony.conf

[root@ xserver2 ~]# echo server 192.168.100.11 iburst >> /etc/chrony.conf //IP为master节点地址

[root@ xserver2 ~]# systemctl enable chronyd && systemctl restart chronyd

所有节点执行chronyc sources命令，查询结果中如果存在以“^\*”开头的行，即说明已经同步成功。

# chronyc sources

210 Number of sources = 1

MS Name/IP address Stratum Poll Reach LastRx Last sample

==================================================================

^\* xserver1 10 6 77 7 +13ns[-2644ns] +/- 13us

##### （6）配置路由转发

RHEL/CentOS7上的一些用户报告了由于iptables被绕过而导致流量路由不正确的问题，所以需要在各节点开启路由转发。

所有节点创建/etc/sysctl.d/K8S.conf文件，添加如下内容。

# cat << EOF | tee /etc/sysctl.d/K8S.conf

net.ipv4.ip\_forward = 1

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

EOF

# modprobe br\_netfilter

# sysctl -p /etc/sysctl.d/K8S.conf

net.ipv4.ip\_forward = 1

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

##### （7）配置IPVS

由于IPVS已经加入到了内核主干，所以需要加载以下内核模块以便为kube-proxy开启IPVS功能。

在所有节点执行以下操作。

# cat > /etc/sysconfig/modules/ipvs.modules <<EOF

#!/bin/bash

modprobe -- ip\_vs

modprobe -- ip\_vs\_rr

modprobe -- ip\_vs\_wrr

modprobe -- ip\_vs\_sh

modprobe -- nf\_conntrack\_ipv4

EOF

# chmod 755 /etc/sysconfig/modules/ipvs.modules

# bash /etc/sysconfig/modules/ipvs.modules && lsmod | grep -e ip\_vs -e nf\_conntrack\_ipv4

上面脚本创建了/etc/sysconfig/modules/ipvs.modules文件，保证在节点重启后能自动加载所需模块。使用lsmod | grep -e ip\_vs -e nf\_conntrack\_ipv4命令查看是否已经正确加载所需的内核模块。

# lsmod | grep -e ip\_vs -e nf\_conntrack\_ipv4

nf\_conntrack\_ipv4 15053 0

nf\_defrag\_ipv4 12729 1 nf\_conntrack\_ipv4

ip\_vs\_sh 12688 0

ip\_vs\_wrr 12697 0

ip\_vs\_rr 12600 0

ip\_vs 145497 6 ip\_vs\_rr,ip\_vs\_sh,ip\_vs\_wrr

nf\_conntrack 139224 2 ip\_vs,nf\_conntrack\_ipv4

libcrc32c 12644 3 xfs,ip\_vs,nf\_conntrack

所有节点安装ipset软件包。

# yum install ipset ipvsadm -y

##### （8）安装Docker

Kubernetes默认的容器运行时仍然是Docker，使用的是Kubelet中内置dockershim CRI实现。需要注意的是，在Kubernetes1.14的版本中，支持的版本有1.13.1、17.03、17.06、17.0918.06和18.09，案例统一使用Docker 18.09版本。

所有节点安装Docker，启动Docker引擎并设置开机自启。

# yum install -y yum-utils device-mapper-persistent-data lvm2

# yum install docker-ce -y

# mkdir -p /etc/docker

# tee /etc/docker/daemon.json <<-'EOF'

{

"exec-opts": ["native.cgroupdriver=systemd"]

}

EOF

# systemctl daemon-reload

# systemctl restart docker

# systemctl enable docker

# docker info |grep Cgroup

Cgroup Driver: system

#### 2.安装Kubernetes集群

##### （1）安装工具

Kubelet负责与其他节点集群通信，并进行本节点Pod和容器生命周期的管理。Kubeadm是Kubernetes的自动化部署工具，降低了部署难度，提高效率。Kubectl是Kubernetes集群命令行管理工具。

所有节点安装Kubernetes工具并启动Kubelet。

# yum install -y kubelet-1.14.1 kubeadm-1.14.1 kubectl-1.14.1

# systemctl enable kubelet && systemctl start kubelet

// 此时启动不成功正常，后面初始化的时候会变成功

##### （2）初始化Kubernetes集群

登录xserver1节点，初始化Kubernetes集群。

[root@xserver1~]# ./kubernetes\_base.sh

[root@xserver1 ~]# kubeadm init --apiserver-advertise-address 192.168.100.11 --kubernetes-version="v1.14.1" --pod-network-cidr=10.16.0.0/16 --image-repository=registry.aliyuncs.com/google\_containers

[init] Using Kubernetes version: v1.14.1

[preflight] Running pre-flight checks

[preflight] Pulling images required for setting up a Kubernetes cluster

[preflight] This might take a minute or two, depending on the speed of your internet connection

[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'

[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"

[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"

[kubelet-start] Activating the kubelet service

[certs] Using certificateDir folder "/etc/kubernetes/pki"

[certs] Generating "ca" certificate and key

[certs] Generating "apiserver" certificate and key

[certs] apiserver serving cert is signed for DNS names [master kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 10.18.4.33]

[certs] Generating "apiserver-kubelet-client" certificate and key

[certs] Generating "front-proxy-ca" certificate and key

[certs] Generating "front-proxy-client" certificate and key

[certs] Generating "etcd/ca" certificate and key

[certs] Generating "etcd/healthcheck-client" certificate and key

[certs] Generating "apiserver-etcd-client" certificate and key

[certs] Generating "etcd/server" certificate and key

[certs] etcd/server serving cert is signed for DNS names [master localhost] and IPs [10.18.4.33 127.0.0.1 ::1]

[certs] Generating "etcd/peer" certificate and key

[certs] etcd/peer serving cert is signed for DNS names [master localhost] and IPs [10.18.4.33 127.0.0.1 ::1]

[certs] Generating "sa" key and public key

[kubeconfig] Using kubeconfig folder "/etc/kubernetes"

[kubeconfig] Writing "admin.conf" kubeconfig file

[kubeconfig] Writing "kubelet.conf" kubeconfig file

[kubeconfig] Writing "controller-manager.conf" kubeconfig file

[kubeconfig] Writing "scheduler.conf" kubeconfig file

[control-plane] Using manifest folder "/etc/kubernetes/manifests"

[control-plane] Creating static Pod manifest for "kube-apiserver"

[control-plane] Creating static Pod manifest for "kube-controller-manager"

[control-plane] Creating static Pod manifest for "kube-scheduler"

[etcd] Creating static Pod manifest for local etcd in "/etc/kubernetes/manifests"

[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

[apiclient] All control plane components are healthy after 25.502670 seconds

[upload-config] storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace

[kubelet] Creating a ConfigMap "kubelet-config-1.14" in namespace kube-system with the configuration for the kubelets in the cluster

[upload-certs] Skipping phase. Please see --experimental-upload-certs

[mark-control-plane] Marking the node master as control-plane by adding the label "node-role.kubernetes.io/master=''"

[mark-control-plane] Marking the node master as control-plane by adding the taints [node-role.kubernetes.io/master:NoSchedule]

[bootstrap-token] Using token: i9k9ou.ujf3blolfnet221b

[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles

[bootstrap-token] configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate credentials

[bootstrap-token] configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token

[bootstrap-token] configured RBAC rules to allow certificate rotation for all node client certificates in the cluster

[bootstrap-token] creating the "cluster-info" ConfigMap in the "kube-public" namespace

[addons] Applied essential addon: CoreDNS

[addons] Applied essential addon: kube-proxy

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 10.18.4.33:6443 --token i9k9ou.ujf3blolfnet221b \

--discovery-token-ca-cert-hash sha256:a0402e0899cf798b72adfe9d29ae2e9c20d5c62e06a6cc6e46c93371436919dc

[root@ xserver1 ~]# mkdir -p $HOME/.kube

[root@ xserver1 ~]# sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

[root@ xserver1 ~]# sudo chown $(id -u):$(id -g) $HOME/.kube/config

检查集群状态。

[root@ xserver1 ~]# kubectl get cs

NAME STATUS MESSAGE ERROR

scheduler Healthy ok

controller-manager Healthy ok

etcd-0 Healthy {"health":"true"}

##### （3）配置Kubernetes网络

登录xserver1节点，部署flannel网络。

[root@ xserver1 ~]# kubectl apply -f yaml/kube-flannel.yml

[root@ xserver1 ~]# kubectl get pods -n kube-system

NAME READY STATUS RESTARTS AGE

coredns-8686dcc4fd-v88br 0/1 Running 0 4m42s

coredns-8686dcc4fd-xf28r 0/1 Running 0 4m42s

etcd-master 1/1 Running 0 3m51s

kube-apiserver-master 1/1 Running 0 3m46s

kube-controller-manager-master 1/1 Running 0 3m48s

kube-flannel-ds-amd64-6hf4w 1/1 Running 0 24s

kube-proxy-r7njz 1/1 Running 0 4m42s

kube-scheduler-master 1/1 Running 0 3m37s

##### （4）xserve2节点加入集群

登录xserver2节点，使用kubeadm join命令将xserver2节点加入集群。

[root@ xserver1 ~]# ./kubernetes\_base.sh

[root@ xserver2 ~]# kubeadm join 192.168.100.11:6443 --token qf4lef.d83xqvv00l1zces9 --discovery-token-ca-cert-hash sha256:ec7c7db41a13958891222b2605065564999d124b43c8b02a3b32a6b2ca1a1c6c

[preflight] Running pre-flight checks

[preflight] Reading configuration from the cluster...

[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'

[kubelet-start] Downloading configuration for the kubelet from the "kubelet-config-1.14" ConfigMap in the kube-system namespace

[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"

[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"

[kubelet-start] Activating the kubelet service

[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:

\* Certificate signing request was sent to apiserver and a response was received.

\* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

登录xserver1节点，检查各节点状态。

[root@ xserver1 ~]# kubectl get nodes

NAME STATUS ROLES AGE VERSION

xserver1 Ready xserver1 4m53s v1.14.1

xserver2 Ready <none> 13s v1.14.1

##### （5）安装Dashboard

使用kubectl create命令安装Dashboard。

[root@ xserver1 ~]# kubectl create -f yaml/kubernetes-dashboard.yaml

创建管理员。

[root@ xserver1~]# kubectl create -f yaml/dashboard-adminuser.yaml

serviceaccount/kubernetes-dashboard-admin created

clusterrolebinding.rbac.authorization.K8S.io/kubernetes-dashboard-admin created

检查所有Pod状态。

[root@ xserver1 ~]# kubectl get pods -n kube-system

NAME READY STATUS RESTARTS AGE

coredns-8686dcc4fd-8jqzh 1/1 Running 0 11m

coredns-8686dcc4fd-dkbhw 1/1 Running 0 11m

etcd-master 1/1 Running 0 11m

kube-apiserver-master 1/1 Running 0 11m

kube-controller-manager-master 1/1 Running 0 11m

kube-flannel-ds-amd64-49ssg 1/1 Running 0 7m56s

kube-flannel-ds-amd64-rt5j8 1/1 Running 0 7m56s

kube-proxy-frz2q 1/1 Running 0 11m

kube-proxy-xzq4t 1/1 Running 0 11m

kube-scheduler-master 1/1 Running 0 11m

kubernetes-dashboard-5f7b999d65-djgxj 1/1 Running 0 11m

提交答案