K8s 快速搭建手册（在线）

2020年2月10日

15:57

**设备配置**

Master : 192.168.132.131 2核 4G

Node1 : 192.168.132.132 2核 3G

**环境准备**

**a、关闭防火墙：**  
$ systemctl stop firewalld  
$ systemctl disable firewalld

**b、关闭selinux：**$ sed -i 's/enforcing/disabled/' /etc/selinux/config   
$ setenforce 0

**c、关闭swap：**  
$ swapoff -a $ 临时  
$ vim /etc/fstab $ 永久

**d、添加主机名与IP对应关系（记得设置主机名）：**

hostnamectl set-hostname master

$ cat /etc/hosts  
192.168.132.131 k8s-master  
192.168.132.132 k8s-node1

**e、将桥接的IPv4流量传递到iptables的链：**$ cat > /etc/sysctl.d/k8s.conf << EOF  
net.bridge.bridge-nf-call-ip6tables = 1  
net.bridge.bridge-nf-call-iptables = 1

net.ipv4.ip\_forward = 1  
vm.swappiness = 0

EOF  
$ sysctl -p /etc/sysctl.d/k8s.conf

**f、安装必要工具：**

yum install -y ipset  
yum install -y ipvsadm  
yum install -y bind-utils

**检查docker版本（安装18.06）**

**a、卸载旧版本docker**

yum remove -y docker \

docker-client \

docker-client-latest \

docker-common \

docker-latest \

docker-latest-logrotate \

docker-logrotate \

docker-engine

**b、更新docker依赖驱动**

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

**c、安装docker-ce.repo(切换阿里云源)**

sudo yum-config-manager --add-repo <https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo>

**d、安装docker**

yum install -y docker-ce-18.06.3.ce-3.el7

**e、调整docker参数**

mkdir -p /etc/docker  
tee /etc/docker/daemon.json <<-'EOF'  
{  
 "registry-mirrors": ["<https://pcy9sknd.mirror.aliyuncs.com>"],  
 "exec-opts": ["native.cgroupdriver=systemd"]  
}  
EOF  
systemctl daemon-reload && systemctl restart docker

这里主要做两点调整：

* 配置国内镜像加速 (国内的各种原因，可能导致拉取镜像像蜗牛一样。。。)
* 修改 cgroup 驱动 (Docker 默认使用 cgroupfs，Kubernetes 官方推荐 systemd）

**安装 Kubeadm（所有主机）**

**a、设置 Kubernetes 的 yum 仓库源：**

tee /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

**b、安装kubeadm和相关工具：**

yum install -y kubelet-1.15.0 kubeadm-1.15.0 kubectl-1.15.0

**c、启动kubelet：**

systemctl start kubelet && systemctl enable kubelet

**Master主机配置**

**a、创建配置文件：**

**#vim master-init.yaml**

apiVersion: kubeadm.k8s.io/v1beta2

kind: InitConfiguration

localAPIEndpoint:

advertiseAddress: 192.168.132.131

bindPort: 6443

---

apiVersion: kubeadm.k8s.io/v1beta2

imageRepository: registry.cn-hangzhou.aliyuncs.com/google\_containers

kind: ClusterConfiguration

kubernetesVersion: v1.15.0

networking:

dnsDomain: cluster.local

podSubnet: 10.244.0.0/16

**b、kubeadm初始化：**

kubeadm init --config master-init.yaml

(结尾会生成如何添加节点的命令，记得拷贝，并按照提示创建.kube)

**c、添加fannel插件**

wget <https://raw.githubusercontent.com/coreos/flannel/62e44c867a2846fefb68bd5f178daf4da3095ccb/Documentation/kube-flannel.yml>

kubectl apply -f kube-flannel.yml

(下载不下来镜像执行:%s/quay.io/quay-mirror.qiniu.com/g)

**初始化Node节点**

**a、节点加入：**

kubeadm join 192.168.132.131:6443 --token zjt7zm.ypfjg6dqynw8i898 \

--discovery-token-ca-cert-hash sha256:1a2fa894f11ab683eddf6ddf4e4957ab2d02e3faf42d139436030aa4f0298634

（没有记录通过主机执行以下命令：sudo kubeadm token create --print-join-command --ttl=0）

**b、检查节点状态：**

kubectl get pod -n kube-system -o wide （所有pod都为ready）

kubectl get node -o wide (所有Node 状态为 Ready)

搭建k8s-dashboard

2020年2月11日

16:00

**获取配置文件**

#wget <https://raw.githubusercontent.com/kubernetes/dashboard/v1.10.1/src/deploy/recommended/kubernetes-dashboard.yaml>

**配置文件修改**

#vim kubernetes-dashboard.yaml

:%s/k8s.gcr.io/registry.cn-hangzhou.aliyuncs.com\/google\_containers/g

**修改模式为nodeport**

spec下添加type: NodePort

targetPort下添加nodePort: 30001(可修改你需要的端口)

创建服务帐户和集群角色绑定

# 创建服务帐户  
cat > dashboard\_service\_account\_admin.yaml << EOF  
apiVersion: v1  
kind: ServiceAccount  
metadata:  
 name: admin-user  
 namespace: kube-system  
EOF  
kubectl apply -f dashboard\_service\_account\_admin.yaml

# 创建集群角色绑定  
cat > dashboard\_cluster\_role\_binding\_admin.yaml << EOF  
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  
EOF  
kubectl apply -f dashboard\_cluster\_role\_binding\_admin.yaml

获取token

kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep admin-user | awk '{print $1}') > admin-token.yaml && cat admin-token.yaml

eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJhZG1pbi11c2VyLXRva2VuLWw1bnpyIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZXJ2aWNlLWFjY291bnQubmFtZSI6ImFkbWluLXVzZXIiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC51aWQiOiJkYmIzMjM3Ni03Yjc5LTQ1MDItOTVmMy1iMmQxMzg2NDBmOTkiLCJzdWIiOiJzeXN0ZW06c2VydmljZWFjY291bnQ6a3ViZS1zeXN0ZW06YWRtaW4tdXNlciJ9.hkmgCOg9jLcWSkRvOGZ7tjUuZn11Mbq5I87JJcP5Z-c7Pne4Jsw\_bRbBDXFCrtZOD8IxIdRdnI3SWAQpQuiuu9bpNRwwzLuHDQrcFxhJugBTF\_xetXmrB2nuSXeYl1r1QMxPZrWNa7dDP2F9M9E1Zz63KPL1dw7ITiVcLfvyJXBzs\_xn9BbBpbNH1kcx3YmxjRUtSmhuegBok\_2IeAfd35e8hl82tttIjA73PFpACGnm8Y2Zrqnb0A4RNasBPxxlQZeW\_CAppEM\_xaHNFjq5yP3te8qszcyO4uQvq4DLYFXnmmJCjPJYV3IeW21BE355Uc9nHs0q6SStfsv6vVFwag

K8s内部服务发现

2020年2月21日

10:28

service\_name: 服务的名称

namespace：资源对象所在名称空间

domain：提供的域名后缀，比如默认的 cluster.local

在 pod 中通过 service\_name.namespace.svc.domain 来访问kubernetes集群中的服务，如果 pod 和 service 在同一个 namespace 中，可以直接使用 service\_name。

搭建EFK日志系统

2020年2月11日

16:31

**3. 部署Elasticsearch 集群**

使用3个 Elasticsearch Pod 来避免高可用下多节点集群中出现的“脑裂”问题

**创建一个名为 logging 的 namespace**

kubectl create namespace logging

**创建建一个名为 elasticsearch 的无头服务**

elasticsearch-svc.yaml

kind: Service  
apiVersion: v1  
metadata:  
 name: elasticsearch  
 namespace: logging  
 labels:  
 app: elasticsearch  
spec:  
 selector:  
 app: elasticsearch  
 clusterIP: None  
 ports:  
 - port: 9200  
 name: rest  
 - port: 9300  
 name: inter-node

创建服务资源对象

$ kubectl create -f elasticsearch-svc.yaml  
$ kubectl get svc -n logging  
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE  
elasticsearch ClusterIP None <none> 9200/TCP,9300/TCP 13d

**创建ES持久化存储**

创建[StorageClass](https://www.jianshu.com/p/aca71580a7c8)

创建elasticsearch-storageclass.yaml

apiVersion: storage.k8s.io/v1  
kind: StorageClass  
metadata:  
 name: es-data-db  
provisioner: fuseim.pri/ifs

创建服务资源对象

$ kubectl create -f elasticsearch-storageclass.yaml   
$ kubectl get storageclass  
NAME PROVISIONER AGE  
es-data-db fuseim.pri/ifs 13d

**使用StatefulSet 创建Es Pod**

Kubernetes StatefulSet 允许我们为 Pod 分配一个稳定的标识和持久化存储，Elasticsearch 需要稳定的存储来保证 Pod 在重新调度或者重启后的数据依然不变，所以需要使用 StatefulSet 来管理 Pod。

elasticsearch-statefulset.yaml

apiVersion: apps/v1  
kind: StatefulSet #定义了名为 es-cluster 的 StatefulSet 对象  
metadata:  
 name: es-cluster  
 namespace: logging  
spec:  
 serviceName: elasticsearch # 和前面创建的 Service 相关联，这可以确保使用以下 DNS 地址访问 StatefulSet 中的每一个 Pod：es-cluster-[0,1,2].elasticsearch.logging.svc.cluster.local，其中[0,1,2]对应于已分配的 Pod 序号。  
 replicas: 3 #3个副本  
 selector:  
 matchLabels:  
 app: elasticsearch #设置匹配标签为app=elasticsearch  
 template:  
 metadata:  
 labels:  
 app: elasticsearch  
 spec: #定义Pod模板  
 containers:  
 - name: elasticsearch  
 image: elasticsearch:7.4.2

resources:  
 limits:  
 cpu: 1000m  
 requests:  
 cpu: 100m  
 ports:  
 - containerPort: 9200  
 name: rest  
 protocol: TCP  
 - containerPort: 9300  
 name: inter-node  
 protocol: TCP  
 volumeMounts: #声明数据持久化目录  
 - name: data  
 mountPath: /usr/share/elasticsearch/data  
 env: #定义变量  
 - name: cluster.name #Elasticsearch 集群的名称  
 value: k8s-logs  
 - name: node.name   
 valueFrom:  
 fieldRef:  
 fieldPath: metadata.name

- name: discovery.seed\_hosts # 旧版本使用 discovery.zen.ping.unicast.hosts  
 value: "elasticsearch" # Disvocery Service  
 - name: cluster.initial\_master\_nodes # 初始化的 master 节点，旧版本相关配置 discovery.zen.minimum\_master\_nodes  
 value: "es-cluster-0" # 根据副本数和name配置

- name: ES\_JAVA\_OPTS #JVM限制  
 value: "-Xms512m -Xmx512m"  
 initContainers: #环境初始化容器  
 - name: fix-permissions  
 image: busybox  
 command: ["sh", "-c", "chown -R 1000:1000 /usr/share/elasticsearch/data"]   
 securityContext:  
 privileged: true  
 volumeMounts:  
 - name: data  
 mountPath: /usr/share/elasticsearch/data  
 - name: increase-vm-max-map  
 image: busybox  
 command: ["sysctl", "-w", "vm.max\_map\_count=262144"]  
 securityContext:  
 privileged: true  
 - name: increase-fd-ulimit  
 image: busybox  
 command: ["sh", "-c", "ulimit -n 65536"]  
 securityContext:  
 privileged: true  
 volumeClaimTemplates: #定义持久化模板  
 - metadata:  
 name: data  
 labels:  
 app: elasticsearch  
 spec:  
 accessModes: [ "ReadWriteOnce" ] #设置访问模式，这意味着它只能被 mount 到单个节点上进行读写  
 storageClassName: es-data-db #使用Storagerclass自动创建PV  
 resources:  
 requests:  
 storage: 100Gi #容量

创建服务资源对象

$ kubectl create -f elasticsearch-statefulset.yaml  
statefulset.apps/es-cluster created  
$ kubectl get pods -n logging  
NAME READY STATUS RESTARTS AGE  
es-cluster-0 1/1 Running 0 20h  
es-cluster-1 1/1 Running 0 20h  
es-cluster-2 1/1 Running 0 20h

验证ES服务是否正常

将本地端口9200转发到 Elasticsearch 节点（如es-cluster-0）对应的端口：

$ kubectl port-forward es-cluster-0 9200:9200 --namespace=logging  
Forwarding from 127.0.0.1:9200 -> 9200  
Forwarding from [::1]:9200 -> 9200

其他终端请求es

$ curl <http://localhost:9200/_cluster/state?pretty>

**4. 部署Kibana服务**

kibana.yaml

apiVersion: v1

kind: Service

metadata:

name: kibana

namespace: logging

labels:

app: kibana

spec:

ports:

- port: 5601

type: NodePort

selector:

app: kibana

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: kibana

namespace: logging

labels:

app: kibana

spec:

selector:

matchLabels:

app: kibana

template:

metadata:

labels:

app: kibana

spec:

containers:

- name: kibana

image: 1ssqq1lxr/kibana-oss:7.2.0

resources:

limits:

cpu: 1000m

requests:

cpu: 100m

env:

- name: ELASTICSEARCH\_URL

value: http://<你的主机IP>:9200

ports:

- containerPort: 5601

使用 kubectl 工具创建：

$ kubectl create -f kibana.yaml

service/kibana created

deployment.apps/kibana created

$ kubectl get pods --namespace=logging

NAME READY STATUS RESTARTS AGE

es-cluster-0 1/1 Running 1 13d

es-cluster-1 1/1 Running 1 13d

es-cluster-2 1/1 Running 1 13d

kibana-7fc9f8c964-nnzx4 1/1 Running 1 13d

$ kubectl get svc --namespace=logging

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

elasticsearch ClusterIP None <none> 9200/TCP,9300/TCP 13d

kibana NodePort 10.111.51.138 <none> 5601:31284/TCP 13d

在浏览器中打开http://<任意节点IP>:31284

**5. 部署Fluent**

使用DasemonSet 控制器来部署 Fluentd 应用，确保在集群中的每个节点上始终运行一个 Fluentd 容器

新建fluentd-configmap.yaml

kind: ConfigMap

apiVersion: v1

metadata:

name: fluentd-config

namespace: logging

labels:

addonmanager.kubernetes.io/mode: Reconcile

data:

system.conf: |-

<system>

root\_dir /tmp/fluentd-buffers/

</system>

containers.input.conf: |-

<source>

@id fluentd-containers.log #表示引用该日志源的唯一标识符，该标识可用于进一步过滤和路由结构化日志数据

@type tail #Fluentd 内置的指令，tail表示 Fluentd 从上次读取的位置通过 tail 不断获取数据，另外一个是http表示通过一个 GET 请求来收集数据。

path /var/log/containers/\*.log #tail类型下的特定参数，告诉 Fluentd 采集/var/log/containers目录下的所有日志，这是 docker 在 Kubernetes 节点上用来存储运行容器 stdout 输出日志数据的目录。

pos\_file /var/log/es-containers.log.pos #检查点，如果 Fluentd 程序重新启动了，它将使用此文件中的位置来恢复日志数据收集。

time\_format %Y-%m-%dT%H:%M:%S.%NZ

localtime

tag raw.kubernetes.\* #用来将日志源与目标或者过滤器匹配的自定义字符串，Fluentd 匹配源/目标标签来路由日志数据

format json

read\_from\_head true

</source>

<match raw.kubernetes.\*\*>

@id raw.kubernetes

@type detect\_exceptions

remove\_tag\_prefix raw

message log

stream stream

multiline\_flush\_interval 5

max\_bytes 500000

max\_lines 1000

</match>

system.input.conf: |-

<source>

@id journald-docker

@type systemd

filters [{ "\_SYSTEMD\_UNIT": "docker.service" }]

<storage>

@type local

persistent true

</storage>

read\_from\_head true

tag docker

</source>

<source>

@id journald-kubelet

@type systemd

filters [{ "\_SYSTEMD\_UNIT": "kubelet.service" }]

<storage>

@type local

persistent true

</storage>

read\_from\_head true

tag kubelet

</source>

forward.input.conf: |-

<source>

@type forward

</source>

output.conf: |-

<filter kubernetes.\*\*>

@type kubernetes\_metadata

</filter>

<match \*\*> #标识一个目标标签，后面是一个匹配日志源的正则表达式，我们这里想要捕获所有的日志并将它们发送给 Elasticsearch，所以需要配置成\*\*。

@id elasticsearch

@type elasticsearch #输出到 Elasticsearch

@log\_level info #要捕获的日志级别 INFO级别及以上

include\_tag\_key true

host elasticsearch

port 9200

logstash\_format true #Elasticsearch 服务对日志数据构建反向索引进行搜索，将 logstash\_format 设置为true，Fluentd 将会以 logstash 格式来转发结构化的日志数据。

request\_timeout 30s

<buffer> # Fluentd 允许在目标不可用时进行缓存，比如，如果网络出现故障或者 Elasticsearch 不可用的时候。缓冲区配置也有助于降低磁盘的 IO。

@type file

path /var/log/fluentd-buffers/kubernetes.system.buffer

flush\_mode interval

retry\_type exponential\_backoff

flush\_thread\_count 2

flush\_interval 5s

retry\_forever

retry\_max\_interval 30

chunk\_limit\_size 2M

queue\_limit\_length 8

overflow\_action block

</buffer>

</match>

配置 docker 容器日志目录以及 docker、kubelet 应用的日志的收集，收集到数据经过处理后发送到 elasticsearch:9200 服务

新建一个 fluentd-daemonset.yaml

apiVersion: v1

kind: ServiceAccount

metadata:

name: fluentd-es

namespace: logging

labels:

k8s-app: fluentd-es

kubernetes.io/cluster-service: "true"

addonmanager.kubernetes.io/mode: Reconcile

---

kind: ClusterRole

apiVersion: rbac.authorization.k8s.io/v1

metadata:

name: fluentd-es

labels:

k8s-app: fluentd-es

kubernetes.io/cluster-service: "true"

addonmanager.kubernetes.io/mode: Reconcile

rules:

- apiGroups:

- ""

resources:

- "namespaces"

- "pods"

verbs:

- "get"

- "watch"

- "list"

---

kind: ClusterRoleBinding

apiVersion: rbac.authorization.k8s.io/v1

metadata:

name: fluentd-es

labels:

k8s-app: fluentd-es

kubernetes.io/cluster-service: "true"

addonmanager.kubernetes.io/mode: Reconcile

subjects:

- kind: ServiceAccount

name: fluentd-es

namespace: logging

apiGroup: ""

roleRef:

kind: ClusterRole

name: fluentd-es

apiGroup: ""

---

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: fluentd-es

namespace: logging

labels:

k8s-app: fluentd-es

version: v2.0.4

kubernetes.io/cluster-service: "true"

addonmanager.kubernetes.io/mode: Reconcile

spec:

selector:

matchLabels:

k8s-app: fluentd-es

version: v2.0.4

template:

metadata:

labels:

k8s-app: fluentd-es

kubernetes.io/cluster-service: "true"

version: v2.0.4

annotations:

scheduler.alpha.kubernetes.io/critical-pod: ''

spec:

serviceAccountName: fluentd-es

containers:

- name: fluentd-es

image: cnych/fluentd-elasticsearch:v2.0.4

env:

- name: FLUENTD\_ARGS

value: --no-supervisor -q

resources:

limits:

memory: 500Mi

requests:

cpu: 100m

memory: 200Mi

volumeMounts:

- name: varlog

mountPath: /var/log

- name: varlibdockercontainers

mountPath: /var/lib/docker/containers

readOnly: true

- name: config-volume

mountPath: /etc/fluent/config.d

nodeSelector: #节点选择

beta.kubernetes.io/fluentd-ds-ready: "true" #节点需有这个标签才会部署收集

tolerations: #添加容忍

- key: node-role.kubernetes.io/master

operator: Exists

effect: NoSchedule

terminationGracePeriodSeconds: 30

volumes:

- name: varlog

hostPath:

path: /var/log

- name: varlibdockercontainers

hostPath:

path: /var/lib/docker/containers

- name: config-volume

configMap:

name: fluentd-config

为节点添加标签

$ kubectl label nodes k8s-node01 beta.kubernetes.io/fluentd-ds-ready=true

$ kubectl get nodes --show-labels

NAME STATUS ROLES AGE VERSION LABELS

k8s-node01 Ready <none> 18d v1.15.3 beta.kubernetes.io/arch=amd64,beta.kubernetes.io/fluentd-ds-ready=true,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=k8s-node01,kubernetes.io/os=linux

kubesphere Ready master 20d v1.15.3 beta.kubernetes.io/arch=amd64,beta.kubernetes.io/fluentd-ds-ready=true,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=kubesphere,kubernetes.io/os=linux,node-role.kubernetes.io/master=

部署资源对象

$ kubectl create -f fluentd-configmap.yaml

configmap "fluentd-config" created

$ kubectl create -f fluentd-daemonset.yaml

serviceaccount "fluentd-es" created

clusterrole.rbac.authorization.k8s.io "fluentd-es" created

clusterrolebinding.rbac.authorization.k8s.io "fluentd-es" created

daemonset.apps "fluentd-es" created

$ kubectl get pods -n logging

NAME READY STATUS RESTARTS AGE

es-cluster-0 1/1 Running 1 18d

es-cluster-1 1/1 Running 1 18d

es-cluster-2 1/1 Running 1 18d

fluentd-es-442jk 1/1 Running 0 4d22h

fluentd-es-knp2k 1/1 Running 0 4d22h

kibana-7fc9f8c964-nnzx4 1/1 Running 1 18d

$ kubectl get svc -n logging

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

elasticsearch ClusterIP None <none> 9200/TCP,9300/TCP 18d

kibana NodePort 10.111.51.138 <none> 5601:31284/TCP 18d

访问Kibana页面添加索引

mysql持久化集群

2020年2月13日

11:20

**1、搭建nfs存储**

为了方便操作，我直接在master上搭建nfs存储。

[root@master ~]# yum -y install nfs-utils  
[root@master ~]# systemctl enable rpcbind  
[root@master lv]# mkdir -p /nfsdata/mysql  
[root@master ~]# vim /etc/exports  
/nfsdata \*(rw,sync,no\_root\_squash)  
[root@master ~]# systemctl start nfs-server  
[root@master ~]# systemctl enable nfs-server  
[root@master ~]# showmount -e  
Export list for master:  
/nfsdata \*

**2、创建PV**

[root@master lv]# vim mysql-pv.yaml #编写pv的yaml文件  
kind: PersistentVolume  
apiVersion: v1  
metadata:  
 name: mysql-pv  
spec:  
 accessModes:  
 - ReadWriteOnce #访问模式定义为只能以读写的方式挂载到单个节点  
 capacity:  
 storage: 1Gi  
 persistentVolumeReclaimPolicy: Retain  
 storageClassName: nfs  
 nfs:  
 path: /nfsdata/mysql  
 server: 192.168.20.6  
[root@master lv]# kubectl apply -f mysql-pv.yaml #执行yaml文件

**3、创建PVC**

[root@master lv]# vim mysql-pvc.yaml #编写pvc的yaml文件  
kind: PersistentVolumeClaim  
apiVersion: v1  
metadata:  
 name: mysql-pvc  
spec:  
 accessModes:  
 - ReadWriteOnce  
 resources:  
 requests:  
 storage: 1Gi  
 storageClassName: nfs #这里指定关联的PV名称  
[root@master lv]# kubectl apply -f mysql-pvc.yaml #执行yaml文件

**4、确认pv及pvc的状态**

至此，PV和PVC的状态类应该都是Bound。

[root@master lv]# kubectl get pv   
NAME CAPACITY ACCESS MODES RECLAIM POLICY STATUS CLAIM STORAGECLASS REASON AGE  
mysql-pv 1Gi RWO Retain Bound default/mysql-pvc nfs 5m8s  
[root@master lv]# kubectl get pvc  
NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE  
mysql-pvc Bound mysql-pv 1Gi RWO nfs 60s

**5、创建pod+svc（service）**

这个pod是提供的MySQL服务，并将其映射到宿主机，可以做和client端通信。

这里我将pod和svc的yaml文件写在同一个文件中。

[root@master lv]# vim mysql-pod.yaml #编写pod的yaml文件

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: mysql  
spec:  
 selector:  
 matchLabels:  
 app: mysql  
 template:  
 metadata:  
 labels:  
 app: mysql  
 spec:  
 containers:  
 - name: mysql  
 image: mysql:5.7  
 env: #以下是设置MySQL数据库的密码  
 - name: MYSQL\_ROOT\_PASSWORD  
 value: "123456"  
 ports:  
 - containerPort: 3306  
 volumeMounts:  
 - name: mysql-persistent-storage  
 mountPath: /var/lib/mysql #MySQL容器的数据都是存在这个目录的，要对这个目录做数据持久化  
 volumes:  
 - name: mysql-persistent-storage  
 persistentVolumeClaim:  
 claimName: mysql-pvc #指定pvc的名称

--- #以下是创建svc的yaml文件  
apiVersion: v1  
kind: Service  
metadata:  
 name: mysql  
spec:  
 type: NodePort  
 ports:  
 - port: 3306  
 targetPort: 3306  
 nodePort: 31111  
 selector:  
 app: mysql  
 selector:  
 app: mysql  
[root@master lv]# kubectl apply -f mysql-pod.yaml #执行yaml文件  
[root@master lv]# kubectl get pod #确定pod运行正常  
NAME READY STATUS RESTARTS AGE  
mysql-6d898f8bcb-lhqxb 1/1 Running 0 51s

tomcat封装docker镜像

2020年2月14日

12:03

创建dockerfile

FROM centos #（在这里使用官方的centos:latest作为tomcat的基础镜像)

MAINTAINER ztkj\_zwj

ENV REFRESHED\_AT 2019-03-05 （这个环境变量用来表明该镜像模板的最后更新时间）

#切换镜像目录，进入/usr目录

WORKDIR /usr

#在/usr/下创建jdk目录,用来存放jdk文件

RUN mkdir jdk

#在/usr/下创建tomcat目录，用来存放tomcat

RUN mkdir tomcat

#将宿主机的jdk目录下的文件拷至镜像的/usr/jdk目录下，

#注意：要使用相对路径也就是当前路径，需与Dockerfile同路径

ADD jdk1.8.0\_151 /usr/jdk/

#将宿主机的tomcat目录下的文件拷至镜像的/usr/tomcat目录下

ADD apache-tomcat-9.0.6 /usr/tomcat/

#设置环境变量

ENV JAVA\_HOME=/usr/jdk

ENV JRE\_HOME=$JAVA\_HOME/jre

ENV CLASSPATH=.:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar:$JRE\_HOME/lib:$CLASSPATH

ENV PATH=/sbin:$JAVA\_HOME/bin:$PATH

#公开端口

EXPOSE 8080

#设置启动命令

ENTRYPOINT ["/usr/tomcat/bin/catalina.sh","run"]

创建镜像带标签：

docker build -t datamanage-tomcat:1.0.0 .【最后有个点代表当前路径】

数据管理组件k8s部署

2020年2月14日

12:07

**一、部署redis**

**1.创建redis-confmap.yaml**

apiVersion: v1

kind: ConfigMap

metadata:

name: redis-conf

data:

redis.conf: |

bind 0.0.0.0

requirepass 123456

**2.创建redis-rc.yaml**

apiVersion: apps/v1

kind: Deployment

metadata:

name: redis

spec:

replicas: 1

selector:

matchLabels:

app: redis

template:

metadata:

labels:

app: redis

spec:

containers:

- name: redis

image: redis

command: ["/bin/sh","-c","exec redis-server /etc/redis/redis.conf"]

resources:

requests:

cpu: 100m #限制cpu的数量为0.1个

memory: 100Mi #限制内存为100M

ports:

- containerPort: 6379

volumeMounts:

- mountPath: /etc/redis

name: redis-config

volumes:

- name: redis-config

configMap:

name: redis-conf

**3.创建redis-svc.yaml**

apiVersion: v1

kind: Service

metadata:

name: redis

spec:

type: NodePort

ports:

- port: 6379

nodePort: 30003

selector:

app: redis

3.执行创建命令

Kubectl apply -f redis-confmap.yaml

Kubectl apply -f redis-rc.yaml

Kubectl apply -f redis-svc.yaml

4.测试redis

kubectl exec -it <你的redis pod> -- redis-cli -h 127.0.0.1 -p 6379 -a 123456

**二、部署rabbitmq**

**1.创建yaml**

apiVersion: extensions/v1beta1  
kind: Deployment  
metadata:  
 annotations:  
 fabric8.io/iconUrl: <https://raw.githubusercontent.com/docker-library/docs/81187b7b50f5af5bdb64d75882f4d9c782ad52c3/rabbitmq/logo.png>  
 labels:  
 app: rabbitmq  
 provider: rabbit  
 version: 3.6.11-management  
 group: com.rabbit  
 name: rabbitmq  
spec:  
 replicas: 1  
 selector:  
 matchLabels:  
 app: rabbitmq  
 provider: rabbit  
 group: com.rabbit  
 template:  
 metadata:  
 annotations:  
 fabric8.io/iconUrl: <https://raw.githubusercontent.com/docker-library/docs/81187b7b50f5af5bdb64d75882f4d9c782ad52c3/rabbitmq/logo.png>  
 labels:  
 app: rabbitmq  
 provider: rabbit  
 version: 3.6.11-management  
 group: com.rabbit  
 spec:  
 containers:  
 - env:  
 - name: RABBITMQ\_DEFAULT\_USER  
 value: "guest"  
 - name: RABBITMQ\_DEFAULT\_PASS  
 value: "guest"  
 image: rabbitmq:3.6.11-management  
 imagePullPolicy: IfNotPresent  
 name: rabbitmq  
 ports:  
 - containerPort: 15672  
 name: manager  
 - containerPort: 5672  
 name: broker  
---  
apiVersion: v1  
kind: List  
items:  
- apiVersion: v1  
 kind: Service  
 metadata:  
 annotations:  
 api.service.kubernetes.io/path: /  
 fabric8.io/iconUrl: <https://raw.githubusercontent.com/docker-library/docs/81187b7b50f5af5bdb64d75882f4d9c782ad52c3/rabbitmq/logo.png>  
 labels:  
 expose: "true"  
 app: rabbitmq  
 provider: rabbit  
 version: 3.6.11-management  
 group: com.rabbit  
 name: mqadmin  
 spec:

type: NodePort

ports:  
 - name: http  
 port: 15672  
 protocol: TCP  
 targetPort: 15672

nodePort: 31672

selector:  
 app: rabbitmq  
 provider: rabbit  
 group: com.rabbit  
- apiVersion: v1  
 kind: Service  
 metadata:  
 annotations:  
 api.service.kubernetes.io/path: /  
 fabric8.io/iconUrl: <https://raw.githubusercontent.com/docker-library/docs/81187b7b50f5af5bdb64d75882f4d9c782ad52c3/rabbitmq/logo.png>  
 labels:  
 expose: "true"  
 app: rabbitmq  
 provider: rabbit  
 version: 3.6.11-management  
 group: com.rabbit  
 name: rabbitmq  
 spec:  
 ports:  
 - name: http  
 port: 5672  
 protocol: TCP  
 targetPort: 5672  
 selector:  
 app: rabbitmq  
 provider: rabbit  
 group: com.rabbit

**2.创建应用**

kubectl apply -f rabbitmq.yaml

**三、部署neo4j**

**1.创建neo4j-storageclass.yaml**

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

name: neo4j-db

provisioner: fuseim.pri/ifs

**2.创建Statefulset.yaml**

apiVersion: "apps/v1beta1"

kind: StatefulSet

metadata:

name: neo4j-core

spec:

serviceName: neo4j

replicas: 1

template:

metadata:

labels:

app: neo4j

component: core

spec:

containers:

- name: neo4j

image: "neo4j:3.3.2-enterprise"

imagePullPolicy: "IfNotPresent"

env:

- name: NEO4J\_dbms\_mode

value: CORE

- name: NUMBER\_OF\_CORES

value: "3"

- name: NEO4J\_dbms\_security\_auth\_\_enabled

value: "false"

- name: NEO4J\_causal\_\_clustering\_discovery\_\_type

value: DNS

- name: NEO4J\_causal\_\_clustering\_initial\_\_discovery\_\_members

value: "neo4j.default.svc.cluster.local:5000"

- name: NEO4J\_ACCEPT\_LICENSE\_AGREEMENT

value: "yes"

command:

- "/bin/bash"

- "-ecx"

- |

export NEO4J\_dbms\_connectors\_default\_\_advertised\_\_address=$(hostname -f)

export NEO4J\_causal\_\_clustering\_discovery\_\_advertised\_\_address=$(hostname -f):5000

export NEO4J\_causal\_\_clustering\_transaction\_\_advertised\_\_address=$(hostname -f):6000

export NEO4J\_causal\_\_clustering\_raft\_\_advertised\_\_address=$(hostname -f):7000

exec /docker-entrypoint.sh "neo4j"

ports:

- containerPort: 5000

name: discovery

- containerPort: 7000

name: raft

- containerPort: 6000

name: tx

- containerPort: 7474

name: browser

- containerPort: 7687

name: bolt

- containerPort: 6362

name: backup

securityContext:

privileged: true

volumeMounts:

- name: datadir

mountPath: "/data"

volumeClaimTemplates:

- metadata:

name: datadir

spec:

accessModes:

- ReadWriteOnce

storageClassName: "neo4j-pvc"

resources:

requests:

storage: "10Gi"

**3.创建neo4j-svc.yaml**

apiVersion: v1

kind: Service

metadata:

name: neo4j

labels:

app: neo4j

component: core

spec:

clusterIP: None

ports:

- port: 7474

targetPort: 7474

name: browser

- port: 6362

targetPort: 6362

name: backup

selector:

app: neo4j

component: core

**4.创建应用**

Kubectl apply -f .

**四、部署nginx**

**1.创建configMap**

kubectl create configmap nginx-conf --from-file=./nginx.conf

**2.创建nginx-rc.yaml**

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 creationTimestamp: null  
 labels:  
 run: nginx  
 name: nginx  
spec:  
 replicas: 1  
 selector:  
 matchLabels:  
 run: nginx  
 strategy: {}  
 template:  
 metadata:  
 creationTimestamp: null  
 labels:  
 run: nginx  
 spec:  
 containers:  
 - image: nginx  
 name: nginx  
 resources: {}  
 volumeMounts:  
 - mountPath: /etc/nginx/nginx.conf  
 name: nginx-conf  
 subPath: nginx.conf  
 volumes:  
 - configMap:  
 name: nginx-conf  
 name: nginx-conf

**3.创建nginx-svc.yaml**

apiVersion: v1

kind: Service

metadata:

name: nginx

spec:

selector:

app: nginx

type: NodePort

ports:

- protocol: TCP

port: 10008

targetPort: 80

nodePort: 30008

**4.创建应用**

Kubectl apply -f nginx-rc.yaml

Kubectl apply -f nginx-svc.yaml

**五、部署springCloud**

**1.创建dockfile**

#基础镜像，如果本地仓库没有，会从远程仓库拉取

FROM openjdk:8-jdk-alpine

#容器中创建目录

RUN mkdir -p /usr/local/jar

#编译后的jar包copy到容器中创建到目录内

COPY target/<包名>.jar <需要的jar包>

#指定容器启动时要执行的命令

ENTRYPOINT ["java","-jar","<需要的jar包>"]

**2.生成docker image**

docker build -t <镜像名> .

**3.创建相应的yaml**

apiVersion: v1

kind: Service

metadata:

name: dockertest

namespace: default

labels:

app: dockertest

spec:

type: NodePort

ports:

- port: <所需端口>

nodePort: <所需端口> #service对外开放端口

selector:

app: dockertest

---

apiVersion: apps/v1

kind: Deployment #对象类型

metadata:

name: dockertest #名称

labels:

app: dockertest #标注

spec:

replicas: 1 #运行容器的副本数，修改这里可以快速修改分布式节点数量

selector:

matchLabels:

app: dockertest

template:

metadata:

labels:

app: dockertest

spec:

containers: #docker容器的配置

- name: dockertest

image: <镜像名>

imagePullPolicy: IfNotPresent #pull镜像时机，

ports:

- containerPort: <所需端口> #容器对外开放端口

**4.创建应用**

kubectl create -f <相应yaml>

在 pod 中通过 service\_name.namespace.svc.domain 来访问kubernetes集群中的服务，如果 pod 和 service 在同一个 namespace 中，可以直接使用 service\_name

追加初始化检测，测试容器是否存在，例如：

initContainers:

- name: init-redis

image: busybox:1.31

command: ['sh', '-c', 'until nslookup redis-server; do echo waiting for redis; sleep 2; done;']

- name: init-mysql

image: busybox:1.31

command: ['sh', '-c', 'until nslookup mysql-server; do echo waiting for mysql; sleep 2; done;']