基本对象概念

Pod：

Pod是最小部署单元，一个Pod有一个或多个容器组成，Pod中容器共享存储和网络，在同一台Docker主机上运行

Service：

Service一个应用服务抽象，定义了Pod逻辑集合和访问这个Pod集合的策略

Service代理Pod集合对外表现是为一个访问入口，分配一个集群IP地址，来自这个IP的请求将负载均衡转发后端Pod中的容器

Service通过Lable Selector选择一组Pod提供服务

Volume：

数据卷，共享Pod中容器使用的数据

Namespace：

命名空间将对象逻辑上分配到不同的Namespace，可以是不同的项目，用户等区分管理，并设定控制策略，从而实现多租户。命名空间也称为虚拟集群。

Lable：

标签用于区分对象（如：Pod，Service），键/值对存在；每个对象可以有多个标签，通过标签关联对象

基于基本对象更高层次抽象：

ReplicaSet：

下一代Replication Controller。确保任何给定时间指定的Pod副本数量，并提供声明式更新等功能。RC与RS唯一区别就是lable selector支持不同，RS支持新的基于集合的标签，RC仅支持基于等式的标签

Deployment：

Deployment式一个更高层次的API对象，它管理ReplicaSets和Pod，并提供声明式更新等功能。

官方建议使用Deployment管理ReplicaSets，而不是直接使用ReplicaSets，这就意味着可能永远不需要直接操作ReplicaSet对象

StatefulSet：

StatefulSet适合持久性的应用程序，有唯一的网络标识符（IP），持久存储，有序的部署，扩展，删除和滚动更新

DaemonSet：

DaemonSet确保所有节点运行同一个Pod。当节点加入Kubernetes集群中，Pod会被调度到该节点上运行，当节点从集群中移除时，DaemonSet的Pod会被删除。删除DaemonSet会清理它所有创建的Pod。

Job：

一次性任务，运行完成后Pod销毁，不再重新启动新容器。还可以任务定时运行。

系统架构及组件功能

Etcd Cluster

kubectl

Node1

Node3

Node2

master

kube-controller-manager

kube-scheduler

kube-apiserver

kubelet

kube-proxy

Docker Engine

Docker Engine

kube-proxy

kubelet

kubelet

kube-proxy

Docker Engine

**Master组件**：

**kube-apiserver**：

Kubernetes API，集群的统一入口，各组件协调者。以HTTP API提供接口服务，所有对象资源的增删改查和监听操作都交给APIServer处理后再提交给Btcd存储。

**kube-controller-manager**:

处理集群中常规后台任务，一个资源对应一个控制器，而ControllerManager就是负责管理这些控制器的

**kube-scheduler**：

根据调度算法为新创建的Pod选择一个Node节点

**Node组件：**

**kubelet**：

kubelet是master在Node节点上的Agent，管理本机运行容器的生命周期，比如创建容器，Pod挂载数据卷，下载secret，获取容器和节点状态等工作。kubelet将每个Pod转换成一组容器。

**kube-proxy**：

在Node节点上实现Pod网络代理，维护网络规则和四层负载均衡工作

**docker or rocket/rkt**

运行容器

第三方服务：

**etcd**：

分布式键值存储系统。用于保持集群状态，比如：Pod，Service等对象信息。

**集群部署**

1. 环境规划
2. 安装Docker
3. 自签TLS证书
4. 部署Etcd集群
5. 部署Flannel网络
6. 创建Node节点kubeconfig文件
7. 获取K8S二进制包
8. 运行Master组件
9. 运行Node组件
10. 查询集群状态
11. 启动一个测试示例
12. 部署Web UI（Dashboard）

|  |  |
| --- | --- |
| 软件 | 版本 |
| Linux操作系统 |  |
| Kubernetes |  |
| Docker |  |
| Etcd |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 角色 | IP | 组件 | 推荐配置 |
| Master | 192.168.56.101 | kube-scheduler  kube-apiserver  kube-controller-manager | CPU 2核2G |
| Node01 | 192.168.56.102 | kubelet  kube-proxy  docker  flannel  etcd |
| Node02 | 192.168.56.103 | kubelet  kube-proxy  docker  flannel  etcd |

**关闭selinux**

安装Docker

#移除旧版本

sudo yum remove docker \

docker-client \

docker-client-latest \

docker-common \

docker-latest \

docker-latest-logrotate \

docker-logrotate \

docker-engine

#安装Docker

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

sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

sudo yum-config-manager --enable docker-ce-nightly

sudo yum-config-manager --enable docker-ce-test

sudo yum-config-manager --disable docker-ce-nightly

sudo yum install -y docker-ce docker-ce-cli containerd.io

sudo systemctl enable docker

sudo systemctl start docker

sudo docker run hello-world

#Uninstall Docker Engine

sudo yum remove docker-ce

#To remove all images, containers, volumes

sudo rm -rf /var/lib/docker

集群部署 – 自签TLS证书

|  |  |
| --- | --- |
| 组件 | 使用的证书 |
| etcd | ca.pem, server.pem, server-key.pem |
| kube-apiserver | ca.pem, server.pem, server-key.pem |
| kubelet | ca.pem, ca-key.pem |
| kube-proxy | ca.pem, kube-proxy.pem, kube-proxy-key.pem |
| kubectl | ca.pem, admin.pem, admin-key.pem |

安装证书生成工具cfssl：

wget <https://pkg.cfssl.org/R1.2/cfssl_linux-amd64>

wget <https://pkg.cfssl.org/R1.2/cfssljson_linux-amd64>

wget <https://pkg.cfssl.org/R1.2/cfssl-certinfo_linux-amd64>

chmod +x cfssl\_linux-amd64 cfssljson\_linux-amd64 cfssl-certinfo\_linux-amd64

mv cfssl\_linux-amd64 /usr/local/bin/cfssl

mv cfssljson\_linux-amd64 /usr/local/bin/cfssljson

mv cfssl-certinfo\_linux-amd64 /usr/local/bin/cfssl-certinfo

wget <https://github.com/etcd-io/etcd/releases/download/v3.2.28/etcd-v3.2.28-linux-amd64.tar.gz>

将etcd, etcdctl 🡪 /opt/kubernetes/bin

内网环境可以不使用证书。尽量都使用

cat > ca-config.json << EOF

{

"signing":{

"default":{

"expiry":"87600h"

},

"profiles":{

"kubernetes":{

"expiry":"87600h",

"usages":[

"signing",

"key encipherment",

"server auth",

"client auth"

]

}

}

}

}

EOF

cat > ca-csr.json << EOF

{

"CN":"kubernetes",

"key":{

"algo":"rsa",

"size":2048

},

"names":[

{

"C":"CN",

"L":"Beijing",

"ST":"Beijing",

"O":"k8s",

"OU":"System"

}

]

}

EOF

**$ cfssl gencert -initca ca-csr.json | cfssljson -bare ca -**

cat > server-csr.json <<EOF

{

"CN":"kubernetes",

"hosts":[

"127.0.0.1",

"192.168.56.101",

"192.168.56.102",

"192.168.56.103"

],

"key":{

"algo":"rsa",

"size":2048

},

"names":[

{

"C":"CN",

"L":"Beijing",

"ST":"Beijing",

"O":"k8s",

"OU":"System"

}

]

}

EOF

**$ cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=kubernetes server-csr.json | cfssljson -bare server**

cat > admin-csr.json <<EOF

{

"CN":"admin",

"hosts":[],

"key":{

"algo":"rsa",

"size":2048

},

"names":[

{

"C":"CN",

"L":"Beijing",

"ST":"Beijing",

"O":"System:masters",

"OU":"System"

}

]

}

EOF

**$ cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=kubernetes admin-csr.json | cfssljson -bare admin**

cat >> **kube-proxy-csr.json** <<EOF

{

"CN":"system:kube-proxy",

"hosts":[],

"key":{

"algo":"rsa",

"size":2048

},

"names":[{

"C":"CN",

"L":"Beijing",

"ST":"Beijing",

"O":"k8s",

"OU":"System"

}]

}

EOF

**$ cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=kubernetes kube-proxy-csr.json | cfssljson -bare kube-proxy**

将生成的\*.pem文件，移到/opt/kubernetes/ssl目录下

**# vim /opt/kubernetes/cfg/etcd**

#[Member]

ETCD\_NAME="etcd01"

ETCD\_DATA\_DIR="/var/lib/etcd/default.etcd"

ETCD\_LISTEN\_PEER\_URLS="http://192.168.56.101:2380"

ETCD\_LISTEN\_CLIENT\_URLS="http://192.168.56.101:2379"

#[Clustering]"

ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS="http://192.168.56.101:2380"

ETCD\_ADVERTISE\_CLIENT\_URLS="http://192.168.56.101:2379"

ETCD\_INITIAL\_CLUSTER="etcd01=http://192.168.56.101:2380,etcd02=http://192.168.56.102:2380,etcd03=http://192.168.56.103:2380"

ETCD\_INITIAL\_CLUSTER\_TOKEN="etcd-cluster"

ETCD\_INITIAL\_CLUSTER\_STATE="new"

**# vim /usr/lib/systemd/system/etcd.service**

[Unit]

Description=Etcd Server

After=network.target

After=network-online.target

Wants=network-online.target

[Service]

Type=notify

EnvironmentFile=-/opt/kubernetes/cfg/etcd

ExecStart=/opt/kubernetes/bin/etcd \

--name=${ETCD\_NAME} \

--data-dir=${ETCD\_DATA\_DIR} \

--listen-peer-urls=${ETCD\_LISTEN\_PEER\_URLS} \

--listen-client-urls=${ETCD\_LISTEN\_CLIENT\_URLS},http://127.0.0.1:2379 \

--advertise-client-urls=${ETCD\_ADVERTISE\_CLIENT\_URLS} \

--initial-advertise-peer-urls=${ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS} \

--initial-cluster=${ETCD\_INITIAL\_CLUSTER} \

--initial-cluster-token=${ETCD\_INITIAL\_CLUSTER} \

--initial-cluster-state=new \

--cert-file=/opt/kubernetes/ssl/server.pem \

--key-file=/opt/kubernetes/ssl/server-key.pem \

--peer-cert-file=/opt/kubernetes/ssl/server.pem \

--peer-key-file=/opt/kubernetes/ssl/server-key.pem \

--trusted-ca-file=/opt/kubernetes/ssl/ca.pem \

--peer-trusted-ca-file=/opt/kubernetes/ssl/ca.pem

Restart=on-failure

LimitNOFILE=65536

[Install]

WantedBy=multi-user.target

ssh-keygen : 生成公钥私钥

ssh-copy-id root@192.168.56.102 : 将密钥发送到node

#发送文件到Nodes

scp /opt/kubernetes/bin/ [root@192.168.56.102:/opt/kubernetes/bin/](mailto:root@192.168.56.102:/opt/kubernetes/bin/)

scp /opt/kubernetes/cfg/ root@192.168.56.102:/opt/kubernetes/cfg/

scp /opt/kubernetes/ssl/ root@192.168.56.102:/opt/kubernetes/ssl/

scp /usr/lib/systemd/system/etcd root@192.168.56.102: /usr/lib/systemd/system/etcd

systemctl start etcd

systemctl enable etcd

echo "PATH=\$PATH:/opt/kubernetes/bin">>/etc/profile

source /etc/profile

#检测ETCD连通性

etcdctl --ca-file=ca.pem --cert-file=server.pem --key-file=server-key.pem --endpoints="http://192.168.56.101:2379,http://192.168.56.102:2379,http://192.168.56.103:2379" cluster-health

**集群部署 – 部署Flannel网络**

**Overlay Networks** : 覆盖网络，在基础网络上叠加得一种虚拟网络技术模式，该网络中得主机通过虚拟链路连接起来。

**VXLAN**：将源数据包封装到UDP中，并使用基础网络得IP/MAC作为外层报文头进行封装，然后在以太网上传输，到达目的地后由隧道端点解封装并将数据发送给目标地址。

**Flannel**：是Overlay网络得一种，也是将源数据包封装在另一种网络包里面进行路由转发和通信，目前已支持UDP，VXLAN，AWS VPC和GCE路由等数据转发方式。

**多主机容器网络通信其它主流方案**：隧道方案（Weave，OpenvSwitch），路由方案（Calico）等。

Container1

Container2

Container3

Container4

VTEP

Container1

Container2

Container3

Container4

eth0

VXLAN Tunnel

eth0

VTEP

节点1

Dockers daemon

节点1

Dockers daemon

VXLAN网络

wget <https://github.com/coreos/flannel/releases/download/v0.11.0/flannel-v0.11.0-linux-amd64.tar.gz>

tar –zxvf flannel-v0.11.0-linux-amd64.tar.gz

#将flanneld, mk-docker-opts.sh 发送到nodes上

scp flanneld mk-docker-opts.sh root@192.168.56.102:/opt/kubernetes/bin/

**$ vim /opt/kubernetes/cfg/flanneld**

FLANNEL\_OPTIONS="--etcd-endpoints=http://192.168.56.101:2379,http://192.168.56.102:2379,http://192.168.56.103:2379 -etcd-cafile=/opt/kubernetes/ssl/ca.pem -etcd-certfile=/opt/kubernetes/ssl/server.pem -etcd-keyfile=/opt/kubernetes/ssl/server-key.pem"

**$ vim /usr/lib/systemd/system/flanneld.service**

[Unit]

Description=Flanneld overlay address etcd agent

After=network-online.target network.target

Before=docker.service

[Service]

Type=notify

EnvironmentFile=-/opt/kubernetes/cfg/flanneld

ExecStart=/opt/kubernetes/bin/flanneld --ip-masq $FLANNEL\_OPTIONS

ExecStartPost=/opt/kubernetes/bin/mk-docker-opts.sh -k DOCKER\_NETWORK\_OPTIONS -d /run/flannel/subnet.env

Restart=on-failure

[Install]

WantedBy=multi-user.target

#设置网段

etcdctl --ca-file=ca.pem --cert-file=server.pem --key-file=server-key.pem --endpoints="http://192.168.56.101:2379,http://192.168.56.102:2379,http://192.168.56.103:2379" set /coreos.com/network/config '{ "Network":"172.17.0.0/16","Backend":{"Type":"vxlan"} }' { "Network":"172.17.0.0/16","Backend":{"Type":"vxlan"} }

systemctl start flanneld

systemctl enable flanneld

$ vim /usr/lib/systemd/system/docker.service

添加： EnvironmentFile=/run/flannel/subnet.env

添加param：

ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock **$DOCKER\_NETWORK\_OPTIONS**

#查看所有子网

$ etcdctl --ca-file=ca.pem --cert-file=server.pem --key-file=server-key.pem --endpoints="http://192.168.56.101:2379,http://192.168.56.102:2379,http://192.168.56.103:2379" ls /coreos.com/network/subnets

#获取指定Key

$ etcdctl --ca-file=ca.pem --cert-file=server.pem --key-file=server-key.pem --endpoints="http://192.168.56.101:2379,http://192.168.56.102:2379,http://192.168.56.103:2379" get /coreos.com/network/subnets/172.17.23.0-24

**集群部署 – 创建Node节点kubeconfig文件**

wget <https://dl.k8s.io/v1.15.6/kubernetes-server-linux-amd64.tar.gz>

**创建TLS Bootstrapping Token && 创建kubelet kubeconfig &&创建kube-proxy kubeconfig**

#!/usr/bin/env bash

#create TLS Bootstrapping Token

export BOOTSTRAP\_TOKEN=$(head -c 16 /dev/urandom | od -An -t x | tr -d ' ')

cat >/opt/kubernetes/cfg/token.csv <<EOF

${BOOTSTRAP\_TOKEN},kubelet-bootstrap,10001,"system:kubelet-bootstrap"

EOF

#create kubelet bootstrapping kubeconfig

export KUBE\_APISERVER="http://192.168.56.101:6443"

#configure cluster parameter

kubectl config set-cluster kubernetes \

--certificate-authority=/opt/kubernetes/ssl/ca.pem \

--embed-certs=true \

--server=${KUBE\_APISERVER} \

--kubeconfig=/opt/kubernetes/cfg/bootstrap.kubeconfig

#configure client ceritifcate parameter

kubectl config set-credentials kubelet-bootstrap \

--token=${BOOTSTRAP\_TOKEN} \

--kubeconfig=/opt/kubernetes/cfg/bootstrap.kubeconfig

#configure updown parameter

kubectl config set-context default \

--cluster=kubernetes \

--user=kubelet-bootstrap \

--kubeconfig=/opt/kubernetes/cfg/bootstrap.kubeconfig

#configure updown default value

kubectl config use-context default --kubeconfig=/opt/kubernetes/cfg/bootstrap.kubeconfig

#create kube-proxy kubeconfig file

kubectl config set-cluster kubernetes \

--certificate-authority=/opt/kubernetes/ssl/ca.pem \

--embed-certs=true \

--server=\${KUBE\_APISERVER} \

--kubeconfig=/opt/kubernetes/cfg/kube-proxy.kubeconfig

kubectl config set-credentials kube-proxy \

--client-certificate=/opt/kubernetes/ssl/kube-proxy.pem \

--client-key=/opt/kubernetes/ssl/kube-proxy-key.pem \

--embed-certs=true \

--kubeconfig=/opt/kubernetes/cfg/kube-proxy.kubeconfig

kubectl config set-context default \

--cluster=kubernetes \

--user=kube-proxy \

--kubeconfig=/opt/kubernetes/cfg/kube-proxy.kubeconfig

kubectl config use-context default --kubeconfig=/opt/kubernetes/cfg/kube-proxy.kubeconfig

**生成的bootstrap.kubeconfig && kube-proxy.kubeconfig 复制到Node节点上**

**部署Node节点**

#部署Node节点前，在Master节点中创建 节点访问权限

$ kubectl create clusterrolebinding kubelet-bootstrap --clusterrole=system:node-bootstrapper --user=kubelet-bootstrap

ElasticSearch:

wget <https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-7.4.2-linux-x86_64.tar.gz>