

使用kubeadm安装部署kubernetes集群:

前提:

- 1、各节点时间同步;
- 2、各节点主机名称解析: dns OR hosts;
- 3、各节点iptables及firewalld服务被disable;

一、设置各节点安装程序包

1、生成yum仓库配置

先获取docker-ce的配置仓库配置文件:

```
# wget https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo -O
/etc/yum.repos.d/
```

生成kubernetes的yum仓库配置文件/etc/yum.repos.d/kubernetes.repo, 内容如下:

```
[kubernetes]
name=Kubernetes
baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86_64/
gpgcheck=0
gpgkey=https://mirrors.aliyun.com/kubernetes/yum/doc/rpm-package-key.gpg
enabled=1
```

2、安装相关的程序包

```
# yum install docker-ce kubelet kubeadm kubectl
```

二、初始化主节点

1、配置docker Unit File中的Environment变量, 定义其HTTPS_PROXY, 或者事先导入所需要的镜像文件; 这里采用第二种方式:

```
# systemctl start docker.service
# docker load master-component-imgs.gz
```

2、编辑kubelet的配置文件/etc/sysconfig/kubelet, 设置其忽略Swap启用的状态错误, 内容如下:

```
KUBELET_EXTRA_ARGS="--fail-swap-on=false"
```

```
KUBE_PROXY_MODE=ipvs
```

```
ip_vs, ip_vs_rr, ip_vs_wrr, ip_vs_sh, nf_conntrack_ipv4
```

3、设定docker和kubelet开机自启动:

```
# systemctl enable docker kubelet
```

4、初始化master节点:

```
# kubeadm init --kubernetes-version=v1.11.1 --pod-network-cidr=10.244.0.0/16 service-
cidr=10.96.0.0/12 --ignore-preflight-errors=Swap
```

注意: 请记录最后的kubeadm join命令的全部内容。

5、初始化kubectl

```
# mkdir ~/.kube
# cp /etc/kubernetes/admin.conf ~/.kube/
```

测试:

```
# kubectl get componentstatus
# kubectl get nodes
```

6、添加flannel网络附件

```
# kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-
flannel.yml
```

7、验证master节点已经就绪

```
# kubectl get nodes
```

三、添加节点到集群中 (以下前四步在要添加的节点上运行, 最后一步在master上运行)

1、配置docker Unit File中的Environment变量，定义其HTTPS_PROXY，或者事先导入所需要的镜像文件；这里采用第二种方式，相关文件的获取路径为ftp://172.20.0.1/pub/Sources/7.x86_64/kubernetes/：

```
# systemctl start docker.service
# docker load node-component-imgs.gz
```

2、编辑kubelet的配置文件/etc/sysconfig/kubelet，设置其忽略Swap启用的状态错误，内容如下：
KUBELET_EXTRA_ARGS="--fail-swap-on=false"

3、设定docker和kubelet开机自启动：
systemctl enable docker kubelet

4、将节点加入第二步中创建的master的集群中，要使用第二步的第4小步记录的kubeadm join命令，而且要额外附加"--ignore-preflight-errors=Swap"选项；

5、待加入完成后，在设置了kubect1的节点上验证节点的就绪状态：
kubect1 get nodes

资源配置清单：

自主式Pod资源

资源的清单格式：

一级字段：apiVersion(group/version), kind, metadata(name,namespace,labels,annotations, ...), spec, status (只读)

Pod资源：

spec.containers <[]object>

```
- name <string>
  image <string>
  imagePullPolicy <string>
    Always, Never, IfNotPresent
```

修改镜像中的默认应用：

command, args

<https://kubernetes.io/docs/tasks/inject-data-application/define-command-argument-container/>

标签：

key=value

key: 字母、数字、_、-、.

value: 可以为空，只能字母或数字开头及结尾，中间可使用

标签选择器：

等值关系：=, ==, !=

集合关系：

KEY in (VALUE1,VALUE2,...)

KEY notin (VALUE1,VALUE2,...)

KEY

!KEY

许多资源支持内嵌字段定义其使用的标签选择器：

matchLabels: 直接给定键值

matchExpressions: 基于给定的表达式来定义使用标签选择器，{key:"KEY", operator:"OPERATOR", values:[VAL1,VAL2,...]}

操作符：

In, NotIn: values字段的值必须为非空列表；

Exists, NotExists: values字段的值必须为空列表；

nodeSelector <map[string]string>

节点标签选择器，

nodeName <string>

annotations:

与label不同的地方在于，它不能用于挑选资源对象，仅用于为对象提供“元数据”。

Pod的生命周期：

状态: Pending, Running, Failed, Succeeded, Unknown

创建Pod:

Pod生命周期中的重要行为:

初始化容器

容器探测:

liveness

readiness

restartPolicy:

Always, OnFailure, Never. Default to Always.

探针类型有三种:

ExecAction、TCPSocketAction、HTTPGetAction

```
apiVersion: v1
kind: Pod
metadata:
  name: liveness-exec-pod
  namespace: default
spec:
  containers:
  - name: liveness-exec-container
    image: busybox:latest
    imagePullPolicy: IfNotPresent
    command: ["/bin/sh", "-c", "touch /tmp/healthy; sleep 30; rm -f /tmp/healthy; sleep 3600"]
    livenessProbe:
      exec:
        command: ["test", "-e", "/tmp/healthy"]
      initialDelaySeconds: 1
      periodSeconds: 3
```

```
apiVersion: v1
kind: Pod
metadata:
  name: readiness-httpget-pod
  namespace: default
spec:
  containers:
  - name: readiness-httpget-container
    image: ikubernetes/myapp:v1
    imagePullPolicy: IfNotPresent
    ports:
    - name: http
      containerPort: 80
    readinessProbe:
      httpGet:
        port: http
        path: /index.html
      initialDelaySeconds: 1
      periodSeconds: 3
```

回顾: Pod

apiVersion, kind, metadata, spec, status (只读)

```
spec:
  containers
  nodeSelector
  nodeName
  restartPolicy:
    Always, Never, OnFailure

  containers:
    name
```

```

image
imagePullPolicy: Always、Never、IfNotPresent
ports:
  name
  containerPort
livenessProbe
readinessProbe
lifecycle

```

```

ExecAction: exec
TCPSocketAction: tcpSocket
HTTPGetAction: httpGet

```

Pod控制器:

```

ReplicationController:
ReplicaSet:
Deployment:
DaemonSet:
Job:
Cronjob:
StatefulSet

```

```

TPR: Third Party Resources, 1.2+, 1.7
CDR: Custom Defined Resources, 1.8+

```

Operator:

ReplicaSet控制器示例:

```

apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: myapp
      release: canary
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        release: canary
        environment: qa
    spec:
      containers:
        - name: myapp-container
          image: ikubernetes/myapp:v1
          ports:
            - name: http
              containerPort: 80

```

Deployment控制器示例:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deploy
  namespace: default
spec:
  replicas: 3
  selector:

```

```

matchLabels:
  app: myapp
  release: canary
template:
  metadata:
    labels:
      app: myapp
      release: canary
  spec:
    containers:
    - name: myapp
      image: ikubernetes/myapp:v2
      ports:
      - name: http
        containerPort: 80

```

DaemonSet控制器示例:

```

apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: filebeat-ds
  namespace: default
spec:
  selector:
    matchLabels:
      app: filebeat
      release: stable
  template:
    metadata:
      labels:
        app: filebeat
        release: stable
    spec:
      containers:
      - name: filebeat
        image: ikubernetes/filebeat:5.6.5-alpine
        env:
        - name: REDIS_HOST
          value: redis.default.svc.cluster.local
        - name: REDIS_LOG_LEVEL
          value: info

```

Service

工作模式: userspace, iptables, ipvs
 userspace: 1.1-
 iptables: 1.10-
 ipvs: 1.11+

类型:

ExternalName, ClusterIP, NodePort, and LoadBalancer

资源记录:

SVC_NAME.NS_NAME.DOMAIN.LTD.

svc.cluster.local.

redis.default.svc.cluster.local.

配置容器化应用的方式:

1、自定义命令行参数;
 args: []

2、把配置文件直接焙进镜像;

3、环境变量

- (1) Cloud Native的应用程序一般可直接通过环境变量加载配置;
- (2) 通过entrypoint脚本来预处理变量为配置文件中的配置信息;

4、存储卷

CoreOS: Operator

StatefulSet:

cattle, pet

PetSet -> StatefulSet

- 1、稳定且惟一的网络标识符;
- 2、稳定且持久的存储;
- 3、有序、平滑地部署和扩展;
- 4、有序、平滑地删除和终止;
- 5、有序的滚动更新;

三个组件: headless service、StatefulSet、volumeClaimTemplate

```
pod_name.service_name.ns_name.svc.cluster.local
myapp-0.myapp.default.svc.cluster.local
```

客户端-->API server

```
user: username, uid
group:
extra:
```

API

Request path

http://172.20.0.70:6443/apis/apps/v1/namespaces/default/deployments/myapp-deploy/

HTTP request verb:

get, post, put, delete

API requests verb:

get, list, create, update, patch, watch, proxy, redirect, delete, deletecollection

Resource:

Subresource:

Namespace

API group

Object URL:

/apis/<GROUP>/<VERSION>/namespaces/<NAMESPACE_NAME>/<KIND>[/OBJECT_ID]/

授权插件: Node, ABAC, RBAC, Webhook

RBAC: Role-based AC

角色 (role)

许可 (permission)

Kubernetes: 认证、授权

API server:

subject --> action --> object

认证: token, tls, user/password

账号: UserAccount, ServiceAccount

授权: RBAC

role, rolebinding

clusterrole, clusterrolebinding

rolebinding, clusterrolebinding:

subject:

user

group

```
serviceaccount
```

```
role:
```

```
role, clusterrole
```

```
object:
```

```
resource group
```

```
resource
```

```
non-resource url
```

```
action: get, list, watch, patch, delete, deletecollection, ...
```

Dashboard:

1、部署:

```
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```

2、将Service改为NodePort

```
kubectl patch svc kubernetes-dashboard -p '{"spec":{"type":"NodePort"}}' -n kube-system
```

3、认证:

认证时的账号必须为ServiceAccount: 被dashboard pod拿来由kubernetes进行认证;

token:

- (1) 创建ServiceAccount, 根据其管理目标, 使用rolebinding或clusterrolebinding绑定至合理role或clusterrole;
- (2) 获取到此ServiceAccount的secret, 查看secret的详细信息, 其中就有token;

kubeconfig: 把ServiceAccount的token封装为kubeconfig文件

- (1) 创建ServiceAccount, 根据其管理目标, 使用rolebinding或clusterrolebinding绑定至合理role或clusterrole;
- (2)

```
kubectl get secret | awk '/^ServiceAccount/{print $1}'
```



```
KUBE_TOKEN=$(kubectl get secret SERVICEACCOUNT_SECRET_NAME -o jsonpath={.data.token} | base64 -d)
```

(3) 生成kubeconfig文件

```
kubectl config set-cluster --kubeconfig=/PATH/TO/SOMEFILE
```

```
kubectl config set-credentials NAME --token=$KUBE_TOKEN --kubeconfig=/PATH/TO/SOMEFILE
```

```
kubectl config set-context
```

```
kubectl config use-context
```

kubernetes集群的管理方式:

- 1、命令式: create, run, expose, delete, edit, ...
- 2、命令式配置文件: create -f /PATH/TO/RESOURCE_CONFIGURATION_FILE, delete -f, replace -f
- 3、声明式配置文件: apply -f, patch,

Kubernetes网络通信:

- (1) 容器间通信: 同一个Pod内的多个容器间的通信, lo
- (2) Pod通信: Pod IP <--> Pod IP
- (3) Pod与服务通信: PodIP <--> ClusterIP
- (4) Service与集群外部客户端的通信;

CNI:

```
flannel
```

```
calico
```

```
cannel
```

```
kube-router
```

```
...
```

解决方案:

虚拟网桥

多路复用: MacVLAN

硬件交换: SR-IOV

```
kubelet, /etc/cni/net.d/
```

flannel:

支持多种后端:

VxLAN

(1) vxlan

(2) Directrouting

host-gw: Host Gateway

UDP:

flannel的配置参数:

Network: flannel使用的CIDR格式的网络地址, 用于为Pod配置网络功能;

10.244.0.0/16 ->

master: 10.244.0.0/24

node01: 10.244.1.0/24

...

node255: 10.244.255.0/24

10.0.0.0/8

10.0.0.0/24

...

10.255.255.0/24

SubnetLen: 把Network切分子网供各节点使用时, 使用多长的掩码进行切分, 默认为24位;

SubnetMin: 10.244.10.0/24

SubnetMax: 10.244.100.0/24

Backend: vxlan, host-gw, udp

vxlan:

网络策略:

名称空间:

拒绝所有出站, 入站;

放行所有出站目标本名称空间内的所Pod;

调度器:

预选策略:

CheckNodeCondition:

GeneralPredicates

HostName: 检查Pod对象是否定义了pod.spec.hostname,

PodFitsHostPorts: pods.spec.containers.ports.hostPort

MatchNodeSelector: pods.spec.nodeSelector

PodFitsResources: 检查Pod的资源需求是否能被节点所满足;

NoDiskConflict: 检查Pod依赖的存储卷是否能满足需求;

PodToleratesNodeTaints: 检查Pod上的spec.tolerations可容忍的污点是否完全包含节点上的污点;

PodToleratesNodeNoExecuteTaints:

CheckNodeLabelPresence:

CheckServiceAffinity:

MaxEBSVolumeCount

MaxGCEPDVolumeCount

MaxAzureDiskVolumeCount

CheckVolumeBinding:

NoVolumeZoneConflict:

CheckNodeMemoryPressure

CheckNodePIDPressure

CheckNodeDiskPressure

MatchInterPodAffinity

优先函数:

LeastRequested:

(cpu((capacity-sum(requested))*10/capacity)+memory((capacity-sum(requested))*10/capacity))/2

BalancedResourceAllocation:

CPU和内存资源被占用率相近的胜出；

NodePreferAvoidPods:

节点注解信息 “scheduler.alpha.kubernetes.io/preferAvoidPods”

TaintToleration: 将Pod对象的spec.tolerations列表项与节点的taints列表项进行匹配度检查，匹配条目越，得分越低；

SeletorSpreading:

InterPodAffinity:

NodeAffinity:

MostRequested:

NodeLabel:

ImageLocality: 根据满足当前Pod对象需求的已有镜像的体积大小之和

节点选择器: nodeSelector, nodeName

节点亲和调度: nodeAffinity

taint的effect定义对Pod排斥效果:

NoSchedule: 仅影响调度过程，对现存的Pod对象不产生影响；

NoExecute: 既影响调度过程，也影响现在的Pod对象；不容忍的Pod对象将被驱逐；

PreferNoSchedule:

容器的资源需求，资源限制

requests: 需求，最低保障；

limits: 限制，硬限制；

CPU:

1颗逻辑CPU

1=1000,millicores

500m=0.5CPU

内存:

E、P、T、G、M、K

Ei、Pi

QoS:

Guaranteed: 每个容器

同时设置CPU和内存的requests和limits.

cpu.limits=cpu.requests

memory.limits=memory.request

Burstable:

至少有一个容器设置CPU或内存资源的requests属性

BestEffort: 没有任何一个容器设置了requests或limits属性；最低优先级；

资源指标: metrics-server

自定义指标: prometheus, k8s-prometheus-adapter

新一代架构:

核心指标流水线: 由kubelet、metrics-server以及由API server提供的api组成；CPU累积使用率、内存实时使用率、Pod的资源占用率及容器的磁盘占用率；

监控流水线: 用于从系统收集各种指标数据并提供终端用户、存储系统以及HPA，它们包含核心指标及许多非核心指标。非核心指标本身不能被k8s所解析，

metrics-server: API server

Helm:

核心术语:

Chart: 一个helm程序包;

Repository: Charts仓库, https/http服务器;

Release: 特定的Chart部署于目标集群上的一个实例;

Chart -> Config -> Release

程序架构:

操作 helm: 客户端, 管理本地的Chart仓库, 管理Chart, 与Tiller服务器交互, 发送Chart, 实例安装、查询、卸载等

Tiller: 服务端, 接收helm发来的Charts与Config, 合并生成release;

RBAC配置文件示例:

<https://github.com/helm/helm/blob/master/docs/rbac.md>

官方可用的Chart列表:

<https://hub.kubeapps.com/>

helm常用命令:

release管理:

install

delete

upgrade/rollback

list

history: release的历史信息;

status: 获取release状态信息;

chart管理:

create

fetch

get

inspect

package

verify

incubator

<http://storage.googleapis.com/kubernetes-charts-incubator>

ELK:

E: elasticsearch

L: logstash

master

replicas

image

{{.master.replicas.image}}

