kubernetes

一、容器编排工具

- docker machine
- docker compose
- docker swarm
  - docker service
  - docker stack
- kubernetes
- mesos+marathon

二、PaaS平台

- OpenShift
- Rancher

三、认识kubernetes

kubernetes

官方网址

https://kubernetes.io/

https://kubernetes.io/zh/

中文社区

http://docs.kubernetes.org.cn/

希腊语：舵手、飞行员

来自于谷歌Borg

使用golang语言开发

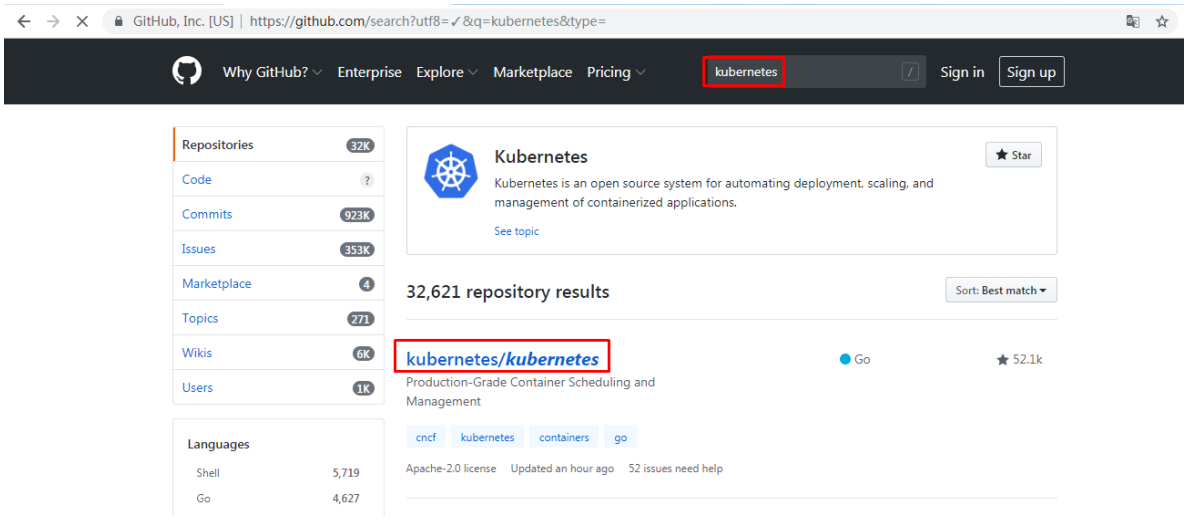
简称为k8s

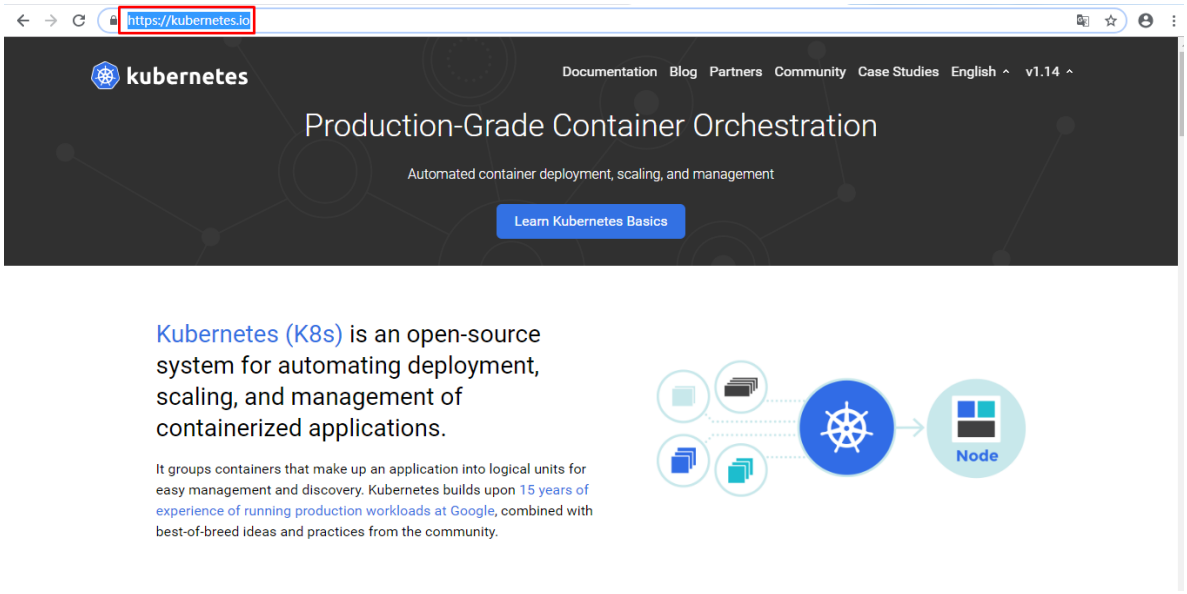
现归属于CNCF

- 云原生计算基金会
- 是一个开源软件基金会，致力于使云计算普遍性和持续性
- 官方：http://www.cncf.io

kubernetes版本

- 2014年9月第一个正式版本
- 2015年7月1.0版本正式发布
- 现在稳定版本为1.15
- 主要贡献者：Google,Redhat,Microsoft,IBM,Intel
- 代码托管github:https://github.com/kubernetes/





- 节点数支持

    - 100台
    - 现在可以支持5000台

- pod管理支持
    - 原先管理1000
    - 现管理150000

用户

- 2017年docker官方宣布原生支持kubernetes
- RedHat公司 PaaS平台 OpenShift核心是kubernetes
- Rancher平台核心是kubernetes
- 现国内大多数公司都可使用kubernetes进行传统IT服务转换，以实现高效管理等。

功能

- 自动装箱
- 自我修复(自愈能力)
- 水平扩展
- 服务发现
- 滚动更新
- 版本回退
- 密钥和配置管理
- 存储编排

四、kubernetes架构

kubernetes是具有中心节点的架构,也就是说有master管理节点

节点角色

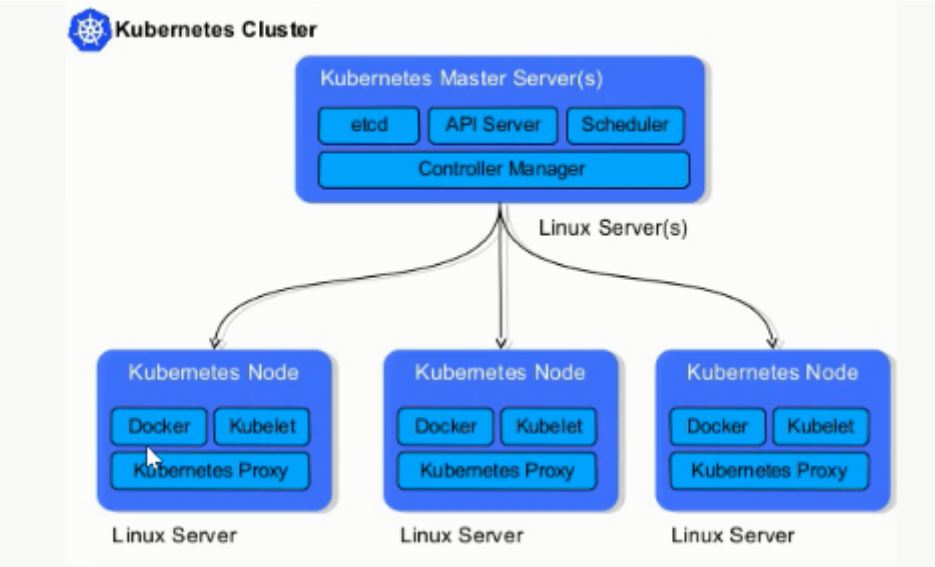
- Master Node manager
- Minion Node worker

简单叫法

Master

Node

架构图示



Master节点组件介绍

master节点是集群管理中心，它的组件可以在集群内任意节点运行，但是为了方便管理所以会在一台主机上运行Master所有组件，**并且不在此主机上运行用户容器**

Master组件包括：

- kube-scheduler

监视新创建没有分配到Node的Pod，为Pod选择一个Node

- kube-apiserver

  用于暴露kubernetes API，任何的资源请求/调用操作都是通过kube-apiserver提供的接口进行

- ETCD

  是kubernetes提供默认的存储系统，保存所有集群数据，使用时需要为etcd数据提供备份计划

- kube-controller-manager

  运行管理控制器，它们是集群中处理常规任务的后台线程

  控制器包括:

  - 节点(Node)控制器
  - 副本(Replication)控制器：负责维护系统中每个副本中的pod
  - 端点(Endpoints)控制器：填充Endpoints对象(即连接service&pods)
  - Service Account和Token控制器：为新的NameSpaces创建默认帐户访问API Token

Node节点组件介绍

node节点用于运行以及维护Pod,提供kubernetes运行时环境

Node组件包括：

- kubelet
  - 负责维护容器的生命周期(创建pod，销毁pod)，同时也负责Volume(CVI)和网络(CNI)的管理
- kube-proxy
  - 通过在主机上维护网络规则并执行连接转发来实现service(Iptables/Ipvs)
  - 随时与API通信，把Service或Pod改变提交给API（不存储在Master本地，需要保存至共享存储上），保存至etcd（可做高可用集群）中，负责service实现，从内部pod至service和从外部node到service访问。
- docker

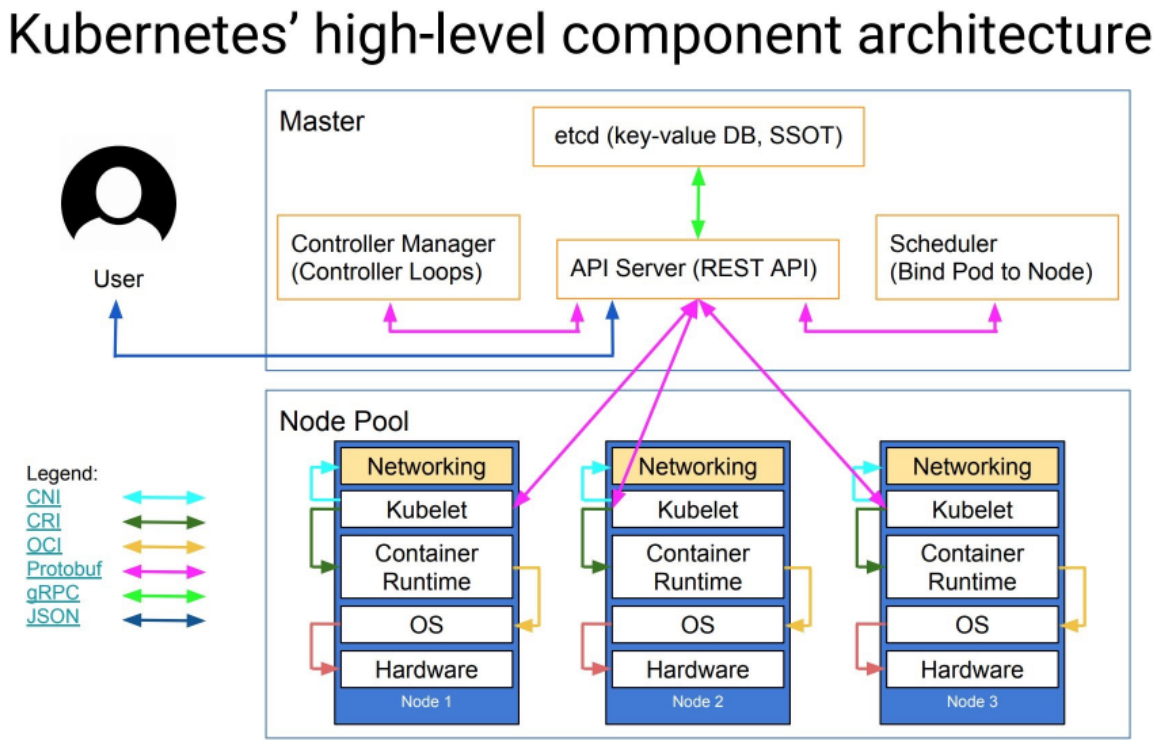
- 容器运行时(Container Runtime)
- 负责镜像管理以及Pod和容器的真正运行
- 支持docker/Rkt/Pouch/Kata等多种运行时,但我们这里只使用docker

Add-ons介绍

Add-ons是附件不是插件，插件是程序本身的一部分，附件不是程序本身一部分.

有附件可以使用功能更丰富，没它并不影响实际使用，可以与主体程序很好结合起来使用

- coredns/kube-dns 负责为整个集群提供DNS服务
- Ingress Controller 为服务提供集群外部访问
- Heapster/Metries-server 提供集群资源监控(监控容器可以使用prometheus)
- Dashboard 提供GUI
- Federation 提供跨可用区的集群
- Fluentd-elasticsearch 提供集群日志采集、存储与查询



五、kubernetes核心概念

pod

pod(豌豆夹)是kubernetes调度的最小单元

pod中包含了运行的容器, 有两种主要使用方式:

- 一个pod包含一个容器
- 一个pod包含多个相联系(紧耦合)的容器

一个Pod内的容器共享一个名称空间(Net Namespace,UTS Namespace,IPC Namespace,另外3个依旧相互隔离)及存储卷(存储卷属于Pod存储资源，多个容器可以共享)

用户pod运行在node节点上,master节点不运行用户pod

pod在集群内可以访问,不能在集群外直接访问, 集群外访问pod需要借助于service

service

此服务不是真正的服务, 此服务是iptables或ipvs中的规则

能够感知pod ip地址的变化(服务发现)

先创建pod，后创建service，创建service其实就是在iptables或ipvs中添加一条规则

如果访问pod，直接访问service

Label

label标签是一组绑定到K8s资源对象上的key/value键值对

label可以附加到各种资源对象上，如Node,Pod,Service,RC等

通过给指定的资源对象捆绑一个或多个不用的label来实现多维度的资源分组管理功能，以便于灵活，方便地进行资源分配，调度，配置，部署等管理工作。

是同一个资源对象上，key不能重复，必须唯一

比如: 我们在创建pod时对pod添加一个app=nginx键值对, 那么后期管理时可以通过app这个key找到对应的pod.

Label Selector

标签选择器

在众多带有标签的pod中，找出指定标签的pod

是kubernetes核心的分组机制，通过Label Selector客户端/用户能够识别一组有共同特征或属性的资源对象。

应用场景:

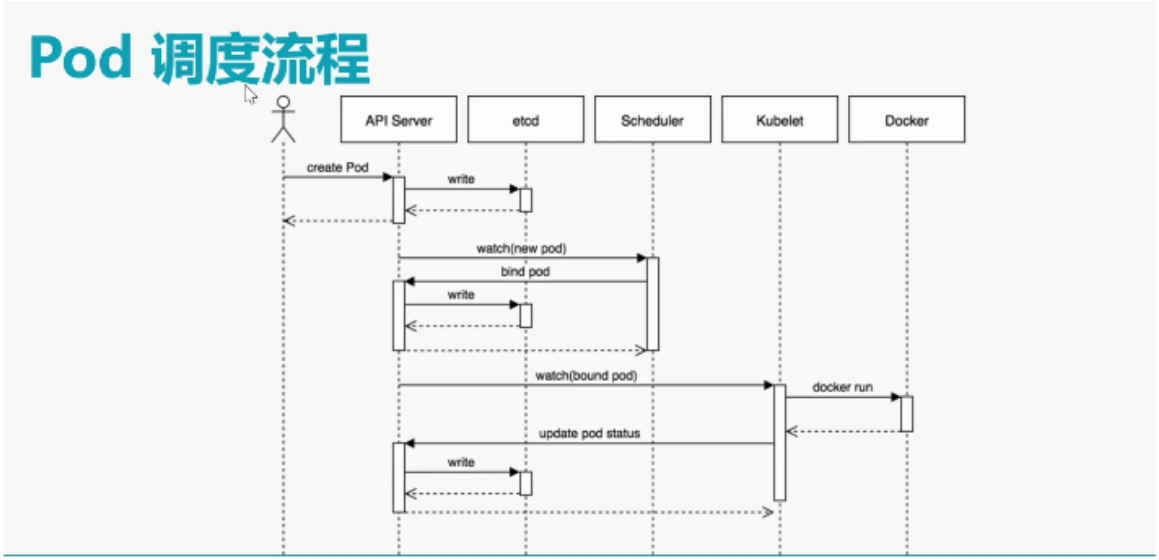
- kube-proxy是通过service上的label selector来选择pod,自动建立每个Service到对应Pod的请求转发路由表，从而实现Service的智能负载均衡机制
- 通过对某些Node定义特定的Label,并且在Pod定义文件中使用Node Selector这种标签调度策略，Kube-scheduler进程可以实现Pod定向调度的特性 (类似docker swarm里的placement)

Scheduler

调度器

提供资源调度功能，负责对每个Node节点资源监控及Pod运行资源适配

共2轮调度，一为预选，二为优选，最终哪个Node节点运行容器取决于优选算法



Replication Controllers

- 副本控制器
- 能够保证集群中运行的pod高可用
- 可以保证pod处于用户期望状态

Replication Controller Manager

- 控制器管理器
- 负责控制器监控，防止控制器出现问题，从而导致容器不可用情况的发生。
- 运行在Master节点
- 为了防止被控制器管理器挂掉，可以考虑多Master节点

常见的pod控制器

- Deployment 声明式更新控制器，只能管理无状态应用，使用较多
- ReplicaSet 副本集控制器,不直接使用.结合其它控制器一起使用。比如一个pod有2个或多个副本
- StatefulSet 有状态副本集

- DaemonSet 在所有Node(包括master)上都运行一个副本,比如可以用到类似filebeat收集日志,监控等场景
- Job 运行作业任务, 对于不需要一直处于运行状态的，我们称之为job，如果运行过程中出现中段，其将被再次启动，直至本次任务结束。如果拷贝一个文件这种job
- CronJob 周期执行任务作业任务

像httpd,nginx这种运行静态页面为无状态的pod

象php,tomcat,mysql等这种有连接的为有状态的pod

pod类型

有控制器的pod

- 有控制器管理的Pod，例如：处于守护进程状态应用，必须处于高可用运行状态，可以使用Pod控制器
- 有Replication Controllers（副本控制器），主要进行监控并时刻保持用户定义指定量Pod副本（运行相同应用的Pod，我们可以把它们叫副本）存在。
- 可以使用Replication Controllers实现pod滚动更新(可临时允许大于用户期望量，先创建一个新版本再关闭一个旧版本),回滚。

无控制器的pod

- 自主式Pod：创建Pod,通过API 进行调度（Scheduler）指定Node创建，如果Node出现故障则此类型Pod将不复存在。无法实现全局调度。

核心概念之间的关系与回顾

- pod

  最小调度或管理单元

- service

  iptables或ipvs规则，与pod之间有关系，用户通过service访问pod

- label

  为了给kubernetes资源对象打上标识，此为标签

- label selector

  标签选择器，可以对资源对象进行分组

- replication controller

  控制器，副本控制器，时刻保持pod数量到达用户的期望值

- replication controller manager

  master节点中运行副本控制器管理器,用于监视RC

- scheduler

  接受aip-server访问，实现pod在某台kubernetes node上运行

- DNS

  通过DNS解决集群内资源名称，达到访问资源目的

六、服务分层架构

- LAMP 一台服务台安装 单服务架构
- 1LNMP 十台服务器安装 分层服务架构
- LNMP 使用容器进行分层，叫微服务架构

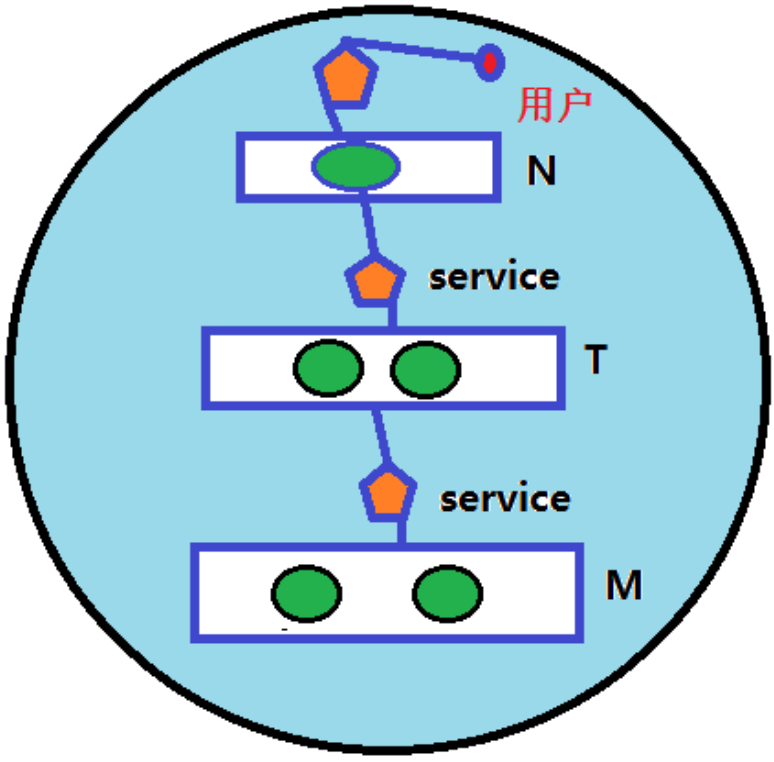
微服务架构组件之间的关系

> 使用kubernetes集群运行NMP或NMT

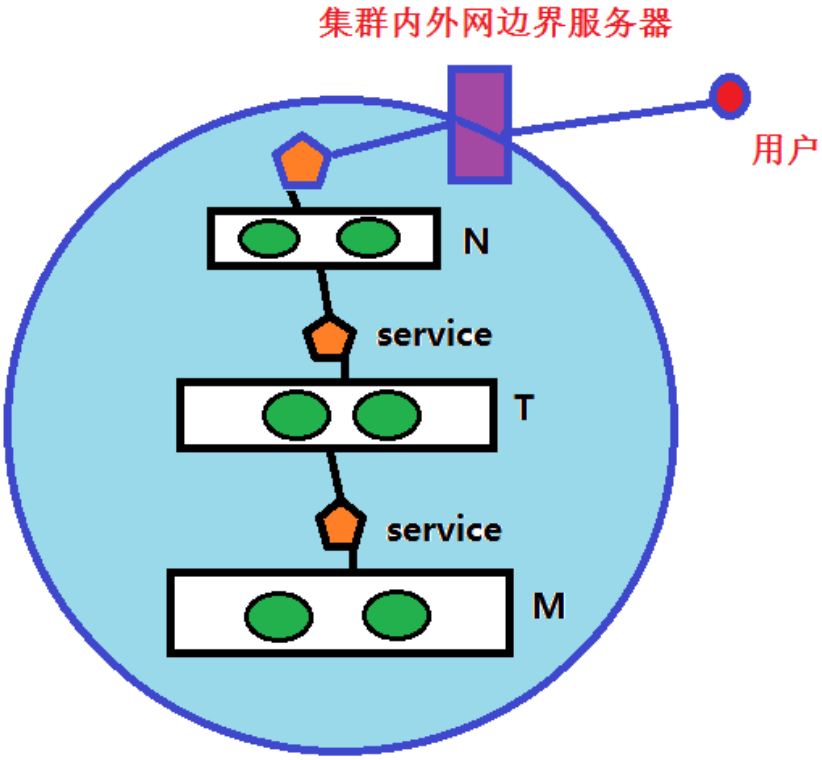
- 可以把Kubernetes集群看做是一个机房（IDC）
- 人访问N前面的service
- N访问T前面的service
- T访问M前面的service

微服务架构图示

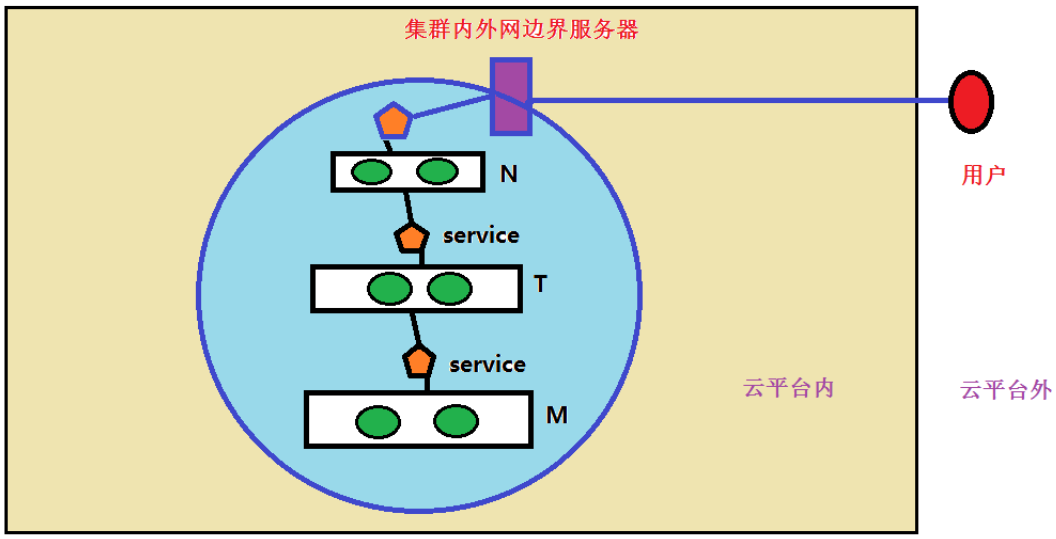
问题: 集群内部用户怎么访问?



问题: 集群外部用户如何访问?



问题: 云平台上运行kubernetes集群, 云平台外面的用户如何访问?



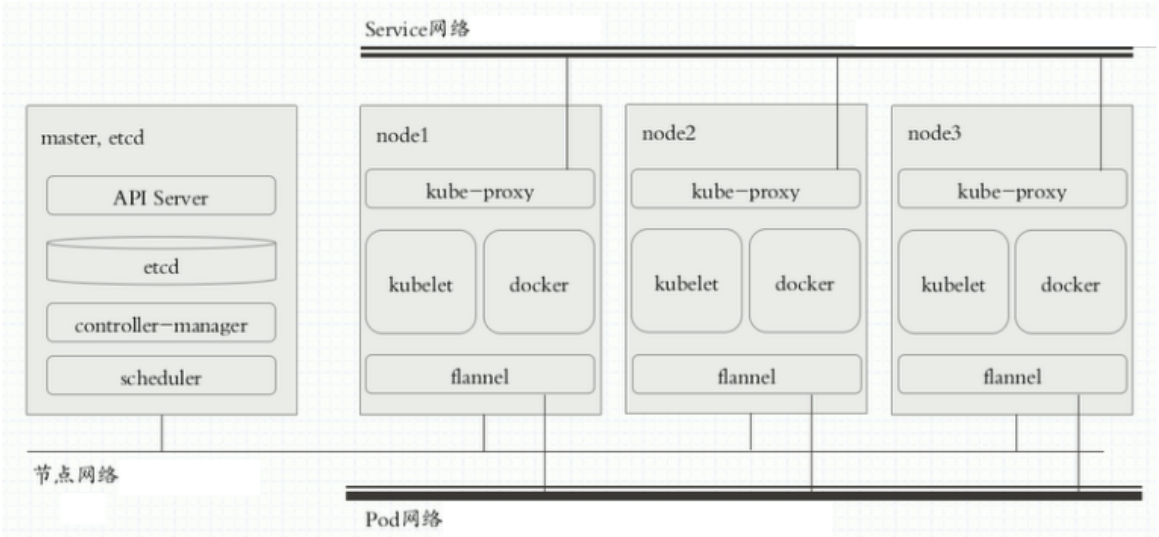
七、kubernetes集群中的网络

kubernetes对网络要求

- pod必须拥有独立的IP
- 所有的pod可通过IP通信，但是不可使用NAT
- 所有的kubernetes Node，可以直接访问pod

网络分层

- 节点网络 服务器节点之间的通信
- 集群网络 Overlay
- Pod网络 Overlay



CNI

- CNI(container network interface)为容器网络接口
- 以插件方式使用
- 为pod提供网络

kubernetes网络解决方案

网络主要考虑以下两个方面:

- 网络配置：给Pod、Service提供IP地址
- 网络策略：通过提供网络策略添加网络规则，实现网络隔离（多租户场景下非常有必要）

网络插件选择

- flannel 仅支持网络配置
- calico 支持网络配置及网络策略，三层隧道网络
- canel 使用calico提供网络策略，使用flannel提供网络配置

Kubernetes 网络方案

	Overlay	L3 Routing	Underlay
描述	把二层报文封装在IP报文之上进行传输	通过三层路由的方式向IP报文传输到目的宿主机	直接使用底层网络的IP，与宿主机在同一个网络里进行通讯
网络要求	底：IP可达	二层可达或BGP可达	二层可达/交换机支持
性能	中：封包、拆包	高：路由转发	高：几乎没有损耗
IP关系	虚拟IP	虚拟IP	物理IP
集群外访问	Ingress/NodePort	Ingress/NodePort	Ingress/NodePort
访问控制	Network Policy	Network Policy	Iptables/外部网络
静态IP	不支持	不支持	支持
场景	对性能要求不高的；网络环境不灵活的	大多数场景	对性能要求高的；需要和现有业务直接通信；需要静态IP
开源产品	Flannel-vxlan,openshift-sdn, cisco-contiv	Calico, flannel-hostgw	Macvlan/Ipvlan

kubernetes集群中三类通信

- 同一Pod内容器间能通信,依靠lo通信
- 各Pod之间通信
  - 各Node之间相同应用Pod通信可以使用Overlay Network（二层报文或三层隧道报文）进行通信
- Pod与Service之间通信
  - 把Pod网关直接指向本地网桥，而Service是非实体组件，是一组保存在iptables或ipvs中的规则,可以由本地TCP/IP协议栈直接实现

Pod与Service通信，如果是跨主机的，那么将会使用至上条提到的Overlay Network网络，当然Service与Pod会不断发生变化的，如果发生变化将由kube-proxy直接API注册并存储至etcd中，不用担心pod找不到service

八、kubernetes集群中的证书

- etcd集群间通信可以使用https,需要一套证书
- etcd集群为其客户端（API Server）提供服务，需要一套证书
- API Server需要向其客户端（用户访问 kubectl）提供服务，需要一套证书
- API Server与Node上kubelet通信，需要一套证书
- API Server与Node上kube-proxy通信，需要一套证书

以上为集群内各服务间访问证书

- 外网用户访问集群服务，需要一套证书，可以在域名所有服务商申请（www.aliyun.com）

九、kubernetes集群部署工具

源码包部署介绍

1,获取源码包

2, 部署在各节点中

3, 启动服务

master

- api-server
- etcd
- scheduler
- controller manager

node

- kubelet
- kube-proxy
- docker

生成证书

- http
- https

kubeadm介绍

- 安装软件 kubelet kube-proxy kubeadm kubectl
- 初始化集群
- 添加node到集群中
- 证书自动生成
- 集群管理系统是以容器方式存在，容器运行在master
- 容器镜像是谷歌提供
    - 阿里云下载容器镜像
    - 谷歌下载

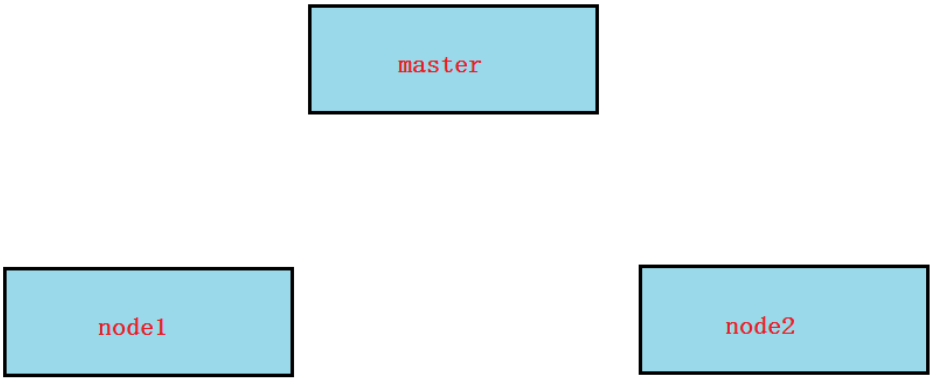
了解其它部署方式:

minikube 单机简化安装

kubeasz 支持多主 参考: https://github.com/easzlab/kubeasz

十、kubeadm部署kubernetes集群

准备环境



三台2G+内存4核CPU的centos7.6, 单网卡

1, **所有节点**主机名及绑定

```
127.0.0.1   localhost localhost.localdomain localhost4
localhost4.localdomain4
::1         localhost localhost.localdomain localhost6
localhost6.localdomain6
192.168.122.1   daniel.cluster.com
192.168.122.11  master
192.168.122.12  node1
192.168.122.13  node2
```

2, **所有节点**关闭selinux

3, **所有节点**关闭firewalld,安装iptables服务,并保存为空规则

```
# systemctl stop firewalld
# systemctl disable firewalld

# yum install iptables-services -y
# systemctl restart iptables
# systemctl enable iptables

# iptables -F
# iptables -F -t nat
# iptables -F -t mangle
# iptables -F -t raw

# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

# service iptables save
iptables: Saving firewall rules to
/etc/sysconfig/iptables:[  OK  ]
```

3, **所有节点**时间同步

4, **所有节点**准备yum源(在centos默认源的基础上再加上以下两个yum源)

```
# vim /etc/yum.repos.d/kubernetes.repo
[k8s]
name=k8s
baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/ku
bernetes-el7-x86_64/
enabled=1
gpgcheck=0
```

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

5, **所有节点**关闭swap(kubernetes1.8开始不关闭swap无法启动)

```
# swapoff -a

打开fstab文件将swap那一行注释保存
# vim /etc/fstab
UUID=38182b36-9be4-45f8-9b3f-f9b3605fcdf0 /
        xfs     defaults        0 0
UUID=6b69e04f-4a85-4612-b1c0-24939fd84962 /boot
        xfs     defaults        0 0
#UUID=9ba6a188-d8e1-4983-9abe-ba4a29b1d138 swap
         swap    defaults        0 0
```

6, RHEL7和CentOS7有由于iptables被绕过而导致流量路由不正确的问题,需要**所有节点**做如下操作:

```
# cat > /etc/sysctl.d/k8s.conf <<EOF
net.ipv4.ip_forward = 1
vm.swappiness = 0
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF

# sysctl -p /etc/sysctl.d/k8s.conf
# modprobe br_netfilter
# lsmod |grep br_netfilter
```

7, **所有节点**设置kube-proxy开启ipvs的前置条件

由于ipvs已经加入到了内核的主干，所以为kube-proxy开启ipvs的前提需要加载以下的内核模块

```
# cat > /etc/sysconfig/modules/ipvs.modules <<EOF
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
# sh /etc/sysconfig/modules/ipvs.modules
# lsmod |egrep 'ip_vs|nf_conntrack'
```

安装软件

1, **所有节点**安装docker, **一定要注意docker版本，docker最新版本kubernetes不一定支持**,下面就是使用最新19.03.01跑docker在集群始化时报的错.所以在这里我们使用18.09的版本

```
        [WARNING SystemVerification]: this Docker version is not on the list of validated versions: 19.03.1. Latest val
dated version: 18.09
```

```
# yum list docker-ce.x86_64 --showduplicates | sort -r

# yum install docker-ce-18.09.8-3.el7 docker-ce-cli-
18.09.8-3.el7 --setopt=obsoletes=0 -y

# docker -v
Docker version 18.09.8, build 0dd43dd87f


# systemctl start docker
# systemctl enable docker
```

2, 所有节点配置加速器和将cgroupdrivier改为systemd,并重启docker服务

```
# vim /etc/docker/daemon.json
{
   "registry-mirrors":
["https://42h8kzrh.mirror.aliyuncs.com"],
   "exec-opts": ["native.cgroupdriver=systemd"]
}

# systemctl restart docker
```

3, **所有节点**安装kubelet,kubeadm,kubectl.并`enable kubelet`服务(**注意:不要start启动**)

```
# yum install kubelet-1.15.1-0 kubeadm-1.15.1-0 kubectl-
1.15.1-0 -y
# systemctl enable kubelet
```

> Kubelet负责与其他节点集群通信，并进行本节点Pod和容器的管理。

> 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集群管理工具。

kubeadm初始化

在master节点上操作(**其它节点不操作**)

注意: 初始化的过程中需要下载1G大小左右的镜像,所以可以提前将我在笔记里准备好的镜像分别在master和node节点上使用docker load导入

```
[root@master ~]# kubeadm init --kubernetes-version=1.15.1
--apiserver-advertise-address=192.168.122.11 --image-
repository registry.aliyuncs.com/google_containers --
service-cidr=10.2.0.0/16 --pod-network-cidr=10.3.0.0/16

[init] Using Kubernetes version: v1.15.1
[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 kuberne
       tes.default.svc
kubernetes.default.svc.cluster.local] and IPs [10.2.0.1
192.168.122.11]
[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/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names
[master localhost] and IPs [192.168.122.11 127.0.0.1 ::1]
[certs] Generating "etcd/healthcheck-client" certificate
and key
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names
[master localhost] and IPs [192.168.122.11 127.0.0.1 ::1]
[certs] Generating "apiserver-etcd-client" certificate and
key
[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
25.503078 seconds
[upload-config] Storing the configuration used in
ConfigMap "kubeadm-config" in the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config-1.15" in
namespace kube-system with the configurationfor the
kubelets in the cluster
[upload-certs] Skipping phase. Please see --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: cyoesa.wyuw7x3Oj0hqu7sr
[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 192.168.122.11:6443 --token
cyoesa.wyuw7x3Oj0hqu7sr \
    --discovery-token-ca-cert-hash
sha256:883260472d0cab8c301b99aefcbedf156209a4bf4df1d98466e
6bb34c1                    dcfb37
```

验证镜像

```
[root@master /]# docker images
REPOSITORY
       TAG                 IMAGE ID            CREATED
        SIZE
registry.aliyuncs.com/google_containers/kube-scheduler
       v1.15.1             b0b3c4c404da        2 weeks ago
        81.1MB
registry.aliyuncs.com/google_containers/kube-proxy
       v1.15.1             89a062da739d        2 weeks ago
        82.4MB
registry.aliyuncs.com/google_containers/kube-apiserver
       v1.15.1             68c3eb07bfc3        2 weeks ago
        207MB
registry.aliyuncs.com/google_containers/kube-controller-
manager   v1.15.1             d75082f1d121        2 weeks
ago         159MB
registry.aliyuncs.com/google_containers/coredns
        1.3.1               eb516548c180        6 months
ago        40.3MB
registry.aliyuncs.com/google_containers/etcd
       3.3.10              2c4adeb21b4f        8 months
ago        258MB
registry.aliyuncs.com/google_containers/pause
        3.1                 da86e6ba6ca1        19 months
ago       742kB
```

初始化可能出现的问题

警告:

```
[WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the
Docker cgroup driver. The recommended driver is "systemd".
Please follow the guide at
https://kubernetes.io/docs/setup/cri/
```

解决: cgroup 驱动建议为systemd

报错:

```
[ERROR Swap]: running with swap on is not supported.
Please disable swap
[preflight] If you know what you are doing, you can make a
check non-fatal with `--ignore-preflight-errors=...`
```

解决: kubernetes1.8开始需要关闭swap

启动集群

在master节点上操作(**其它节点不操作**)

执行`export KUBECONFIG=/etc/kubernetes/admin.conf`就可以启动集群(加到/etc/profile里实现开机自动启动)

```
确认kubelet服务启动了
[root@master ~]# systemctl status kubelet.service


[root@master ~]# vim /etc/profile
export KUBECONFIG=/etc/kubernetes/admin.conf

[root@master ~]# source /etc/profile
```

查看集群状态

```
[root@master ~]# kubectl get cs                 # cs为
componentstatus
NAME                 STATUS    MESSAGE             ERROR
scheduler            Healthy   ok
controller-manager   Healthy   ok
etcd-0               Healthy   {"health":"true"}

[root@master ~]# kubectl get node
NAME              STATUS     ROLES    AGE   VERSION
master            NotReady   master   6m    v1.15.1
```

创建flannel网络

参考: https://github.com/coreos/flannel

在master节点上操作(**其它节点不操作**)

1,下载kube-flannel.yml

```
[root@master ~]# mkdir /root/k8s
[root@master ~]# cd /root/k8s/
[root@master k8s]# curl -O
https://raw.githubusercontent.com/coreos/flannel/master/Do
cumentation/kube-flannel.yml
```

2, 应用kube-flannel.yml创建pod(**这一步非常慢,因为要下载镜像,可以使用共享的镜像先导入**)

```
[root@master k8s]# kubectl apply -f kube-flannel.yml
podsecuritypolicy.extensions/psp.flannel.unprivileged
created
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel
created
serviceaccount/flannel created
configmap/kube-flannel-cfg created
daemonset.extensions/kube-flannel-ds-amd64 created
daemonset.extensions/kube-flannel-ds-arm64 created
daemonset.extensions/kube-flannel-ds-arm created
daemonset.extensions/kube-flannel-ds-ppc64le created
daemonset.extensions/kube-flannel-ds-s390x created
```

3, 要确认所有的pod为running状态

```
[root@master k8s]# kubectl get pods -n kube-system
NAME                                 READY   STATUS
 RESTARTS   AGE
coredns-bccdc95cf-d576d              1/1     Running   0
   64m
coredns-bccdc95cf-xc8l4              1/1     Running   0
   64m
etcd-master                          1/1     Running   0
   63m
kube-apiserver-master                1/1     Running   0
   63m
kube-controller-manager-master       1/1     Running   0
   64m
kube-flannel-ds-amd64-5vp8k          1/1     Running   0
   2m15s
kube-proxy-22x22                     1/1     Running   0
   64m
kube-scheduler-master                1/1     Running   0
   63m
```

```
[root@master k8s]# kubectl get pod --all-namespaces -o wide
NAMESPACE     NAME                             READY   STATUS    RESTARTS   AGE     IP              NODE     NOMINATED NODE   READINESS GATES
kube-system   coredns-bccdc95cf-d576d          1/1     Running   0          66m     10.3.0.2        master   <none>           <none>
kube-system   coredns-bccdc95cf-xc8l4          1/1     Running   0          66m     10.3.0.3        master   <none>           <none>
kube-system   etcd-master                      1/1     Running   0          65m     192.168.122.11  master   <none>           <none>
kube-system   kube-apiserver-master            1/1     Running   0          65m     192.168.122.11  master   <none>           <none>
kube-system   kube-controller-manager-master   1/1     Running   0          66m     192.168.122.11  master   <none>           <none>
kube-system   kube-flannel-ds-amd64-5vp8k      1/1     Running   0          4m12s   192.168.122.11  master   <none>           <none>
kube-system   kube-proxy-22x22                 1/1     Running   0          66m     192.168.122.11  master   <none>           <none>
kube-system   kube-scheduler-master            1/1     Running   0          65m     192.168.122.11  master   <none>           <none>
```

如果初始化遇到问题，尝试使用下面的命令清理,再重新初始化

```
[root@master ~]# kubeadm reset
[root@master ~]# ifconfig cni0 down
[root@master ~]# ip link delete cni0
[root@master ~]# ifconfig flannel.1 down
[root@master ~]# ip link delete flannel.1
[root@master ~]# rm -rf /var/lib/cni/
```

验证master节点OK

```
[root@master k8s]# kubectl get nodes
NAME     STATUS   ROLES    AGE   VERSION
master   Ready    master   66m   v1.15.1
```

加入其它节点

1, node1上join集群

```
[root@node1 ~]# kubeadm join 192.168.122.11:6443 --token
cyoesa.wyuw7x30j0hqu7sr --discovery-token-ca-cert-hash
sha256:883260472d0cab8c301b99aefcbedf156209a4bf4df1d98466e
6bb34c1dcfb37

[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-1.15" 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.
```

2, node2上join集群

```
[root@node2 ~]# kubeadm join 192.168.122.11:6443 --token
cyoesa.wyuw7x30j0hqu7sr --discovery-token-ca-cert-hash
sha256:883260472d0cab8c301b99aefcbedf156209a4bf4df1d98466e
6bb34c1dcfb37

[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-1.15" 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.
```

确认集群OK

在master上验证集群OK

```
[root@master ~]# kubectl get nodes
NAME     STATUS   ROLES    AGE     VERSION
master   Ready    master   88m     v1.15.1
node1    Ready    <none>   3m42s   v1.15.1
node2    Ready    <none>   101s    v1.15.1
```

移除节点的做法(假设移除node2)

1, 在master节点上执行

```
[root@master ~]# kubectl drain node2 --delete-local-data --force --ignore-daemonsets
[root@master ~]# kubectl delete node node2
```

2, 在node2节点上执行

```
[root@node2 ~]# kubeadm reset
[root@node2 ~]# ifconfig cni0 down
[root@node2 ~]# ip link delete cni0
[root@node2 ~]# ifconfig flannel.1 down
[root@node2 ~]# ip link delete flannel.1
[root@node2 ~]# rm -rf /var/lib/cni/
```

3,在node1上执行

```
[root@node1 ~]# kubectl delete node node2
```