kubernetes

Kubernetes是谷歌严格保密十几年的秘密武器——Borg的一个开源版本，是Docker分布式系统解决方案。

Borg

Borg是谷歌内部使用的大规模集群管理系统，基于容器技术，目的是实现资源管理的自动化，以及跨多个数据中心的资源利用率的最大化；

容器编排引擎三足鼎立：

- Mesos
- Docker Swarm
- Kubernetes

早在 2015 年 5 月，Kubernetes 在 Google 上的的搜索热度就已经超过了 Mesos 和 Docker Swarm，从那儿之后更是一路飙升，将对手甩开了十几条街,容器编排引擎领域的三足鼎立时代结束。

目前，AWS、Azure、Google、阿里云、腾讯云等主流公有云提供的是基于 Kubernetes 的容器服务；Rancher、CoreOS、IBM、Mirantis、Oracle、Red Hat、VMWare 等无数厂商也在大力研发和推广基于 Kubernetes 的容器 CaaS 或 PaaS 产品。可以说，Kubernetes 是当前容器行业最炙手可热的明星。

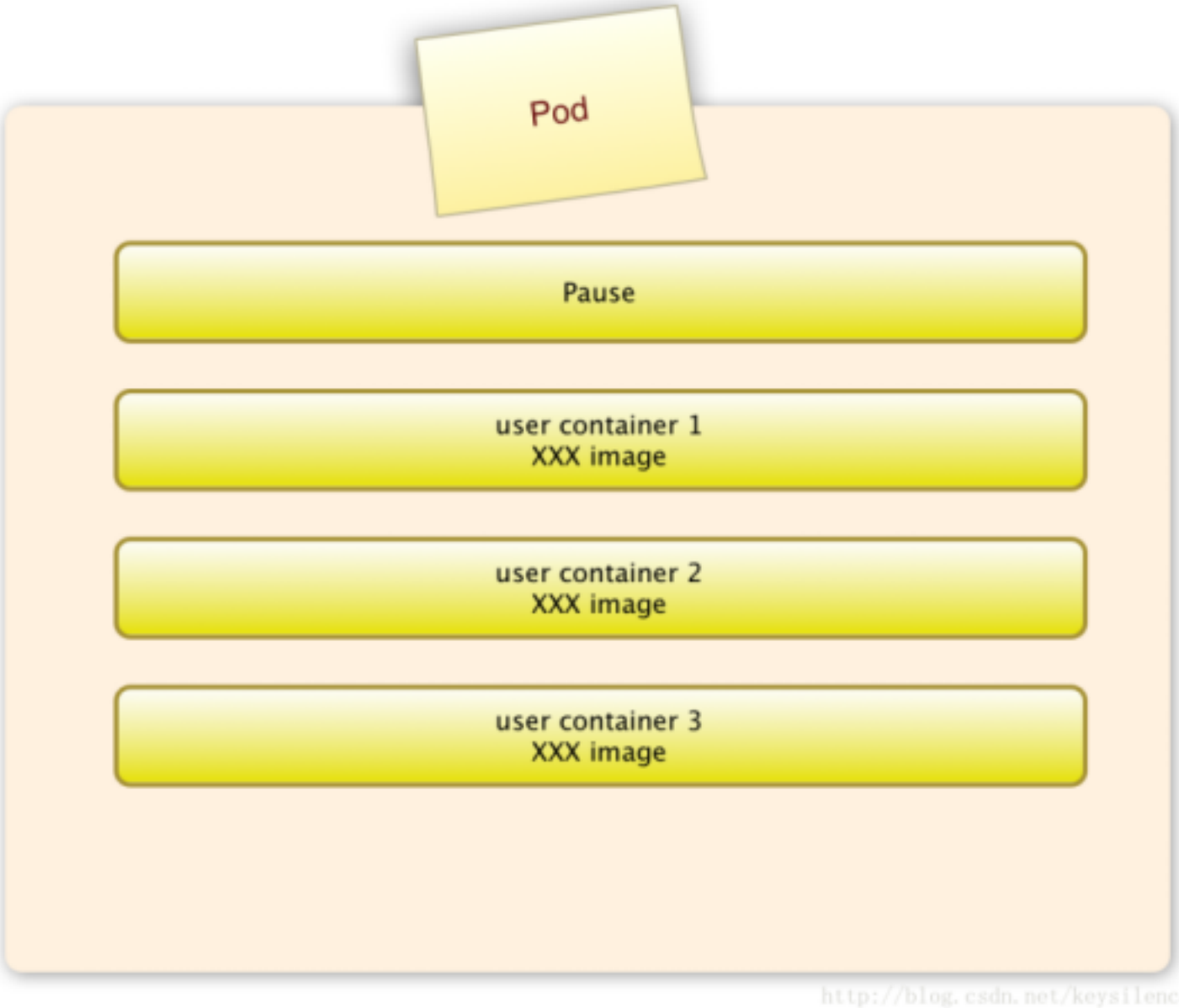
Google 的数据中心里运行着超过 20 亿个容器，而且 Google 十年前就开始使用容器技术。最初，Google 开发了一个叫 Borg 的系统（现在命名为 Omega欧米伽）来调度如此庞大数量的容器和工作负载。在积累了这么多年的经验后，Google 决定重写这个容器管理系统，并将其贡献到开源社区，让全世界都能受益。这个项目就是 Kubernetes。简单的讲，Kubernetes 是 Google Omega 的开源版本。

核心概念

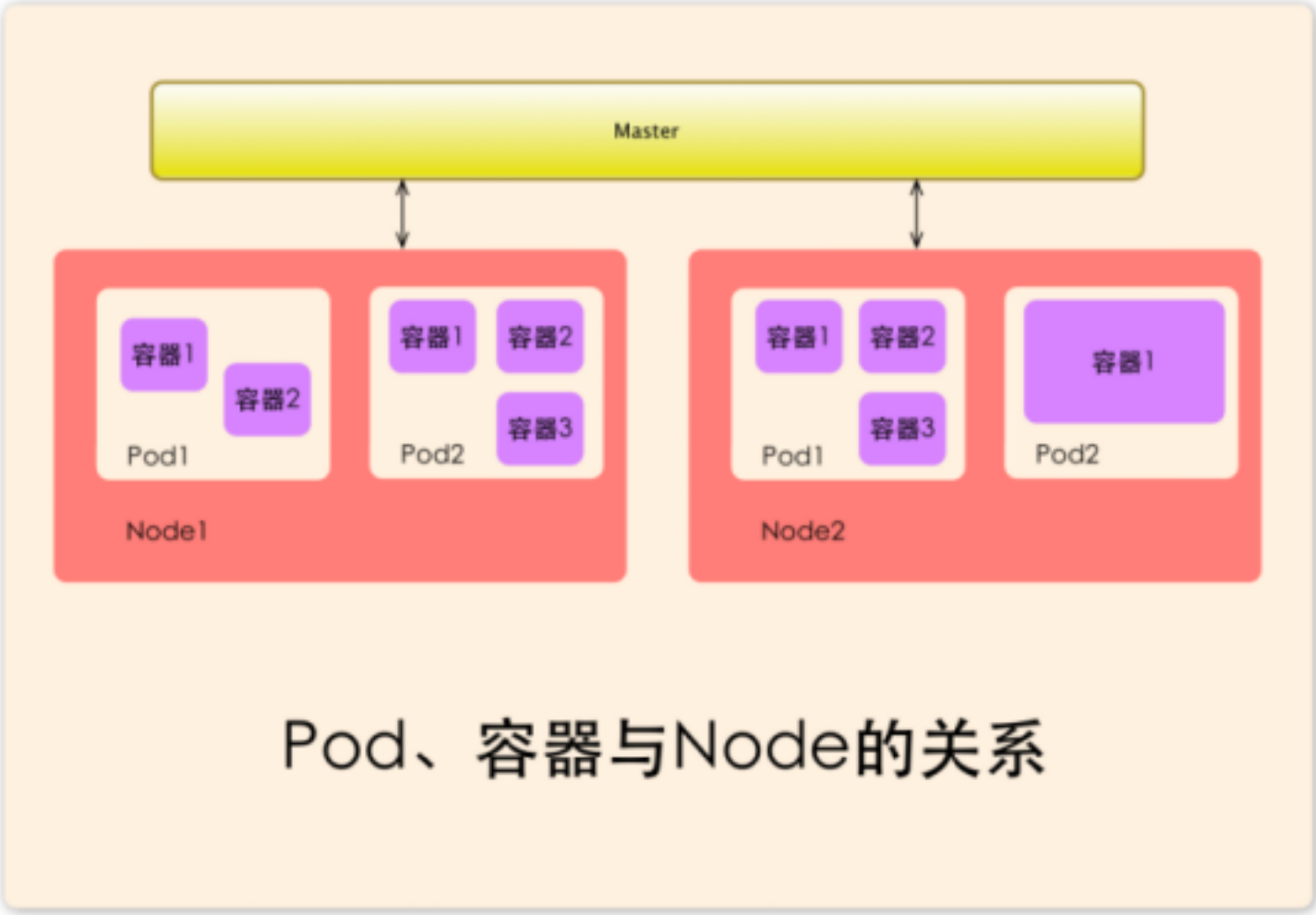
Kubernetes的核心概念

1.Pod

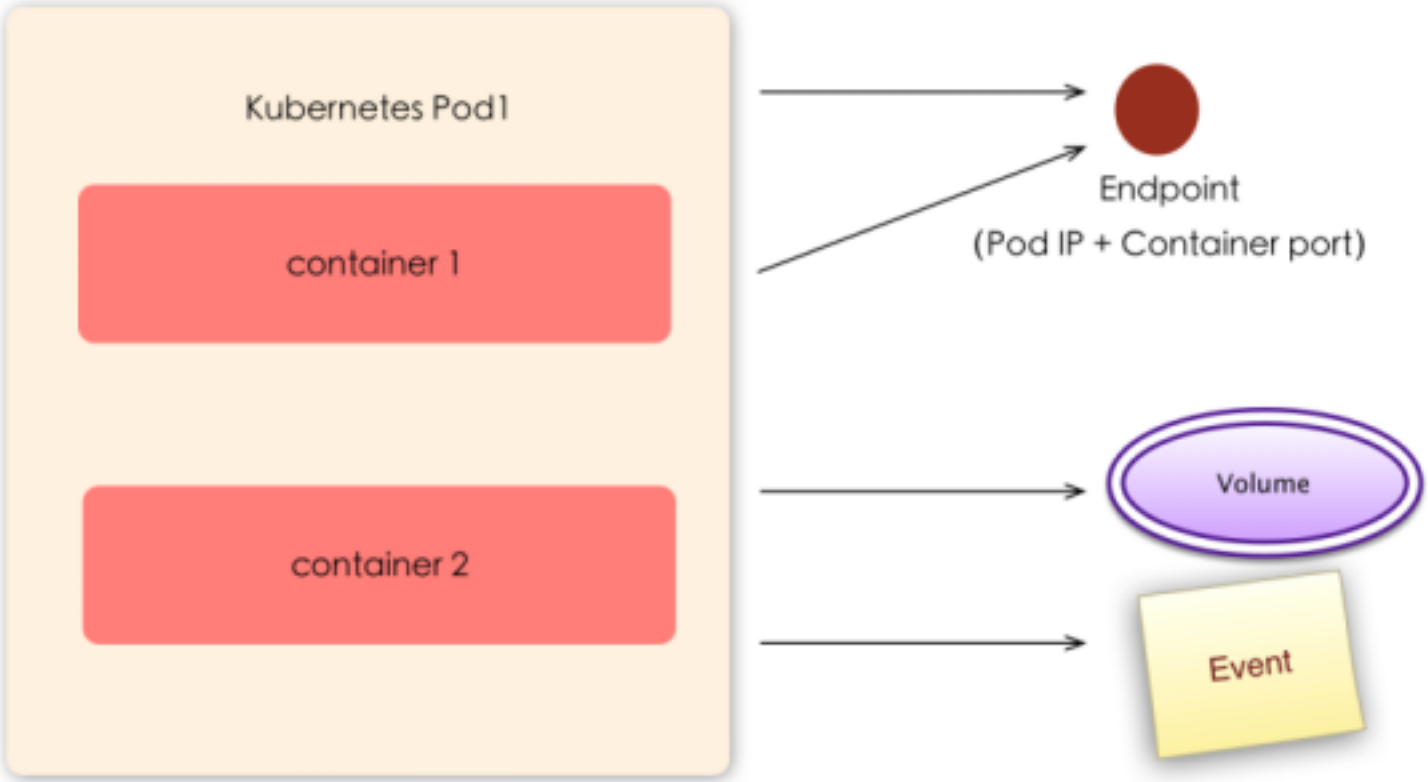
Pod直译是豆荚，可以把容器想像成豆荚里的豆子，把一个或多个关系紧密的豆子包在一起就是豆荚（一个Pod）。在k8s中我们不会直接操作容器，而是把容器包装成Pod再进行管理，运行于Node节点上，若干相关容器的组合。Pod内包含的容器运行在同一宿主机上，使用相同的网络命名空间、IP地址和端口，能够通过localhost进行通信。Pod是Kubernetes进行创建、调度和管理的最小单位，它提供了比容器更高层次的抽象，使得部署和管 理更加灵活。一个Pod可以包含一个容器或者多个相关容器。



- 每个Pod都有一个特殊的被称为"根容器"的Pause容器，还包含一个或多个紧密相关的用户业务容器；
- 一个Pod里的容器与另外主机上的Pod容器能够直接通信；
- 如果Pod所在的Node宕机，会将这个Node上的所有Pod重新调度到其他节点上；
- 普通Pod及静态Pod，前者存放在etcd(存储主管)中，后者存放在具体Node上的一个具体文件中，并且只能在此Node上启动运行；
- Docker Volume对应Kubernetes中的Pod Volume；
- 每个Pod可以设置限额的计算机资源有CPU和Memory； docker也行
- Requests，资源的最小申请量；
- Limits，资源最大允许使用的量；



Pod、容器与Node的关系



Pod及周边对象

Event

是一个事件记录(类似于日志)，记录了事件最早产生的时间、最后重复时间、重复次数、发起者、类型，以及导致此事件的原因等信息。Event通常关联到具体资源对象上，是排查故障的重要参考信息；

Pod IP

Pod的IP地址，是Docker Engine（docker引擎）根据docker0网桥的IP地址段进行分配的，通常是一个虚拟的二层网络，Node上的Pod能够彼此通信，需要通过Pod IP所在的虚拟二层网络进行通信（内部通信），而真实的TCP流量（对外通信流量）则是通过Node IP所在的物理网卡流出的；

Cluster IP

Service的IP地址。特性如下：

- 仅仅作用于Kubernetes Servcie这个对象，并由Kubernetes管理和分配IP地址；
- 无法被Ping，因为没有一个"实体网络对象"来响应；
- 只能结合Service Port组成一个具体的通信端口；
- Node IP、Pod IP与Cluster IP之间的通信，采用的是Kubernetes自己设计的一种编程方式的特殊的路由规则，与IP路由有很大的不同；

Node IP

Node节点的IP地址，是Kubernetes集群中每个节点的物理网卡的IP地址，是真是存在的物理网络，所有属于这个网络的服务器（各个node）之间都能通过这个网络直接通信；

2.Replication Controller（复制控制器）

Replication Controller用来管理Pod的副本，保证集群中存在指定数量的Pod副本。集群中副本的数量大于指定数量，则会停止指定数量之外的多余容器数量，反之，则会启动少于指定数量个数的容器，保证数量不变。Replication Controller是实现弹性伸缩、动态扩容和滚动升级的核心。

部署和升级Pod，声明某种Pod的副本数量在任意时刻都符合某个预期值；

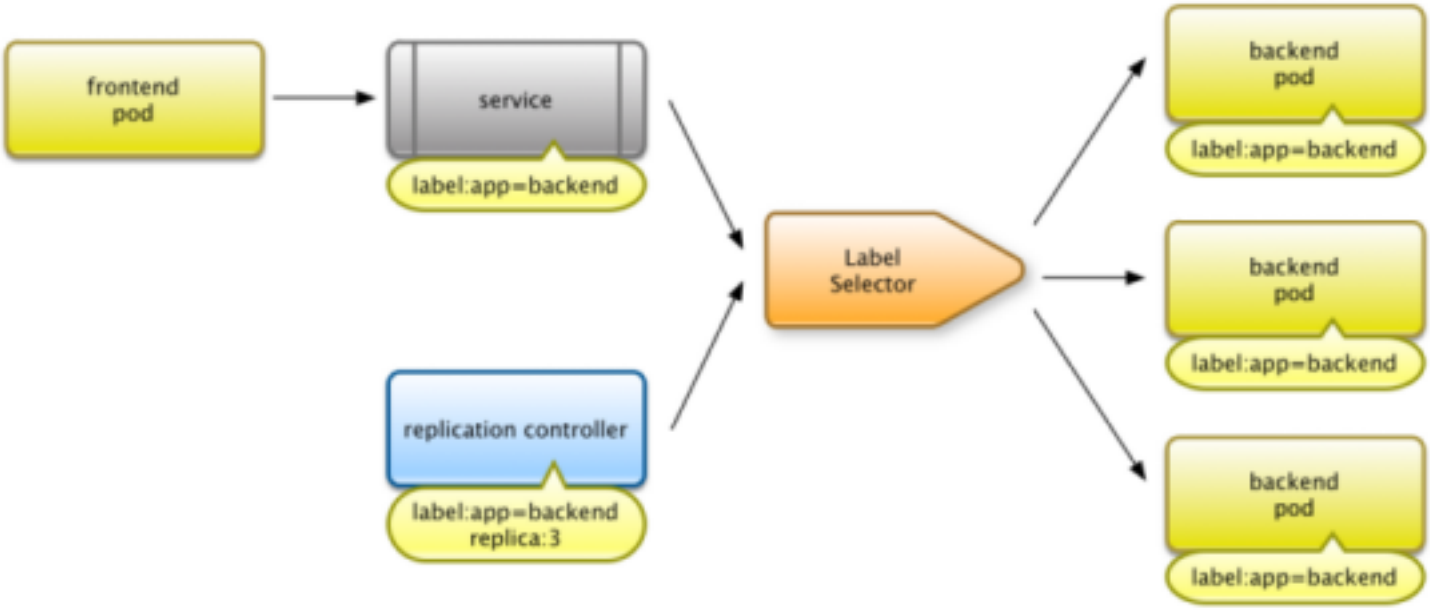
- Pod期待的副本数；
- 用于筛选目标Pod的Label Selector（标签选择器）；
- 当Pod副本数量小于预期数量的时候，用于创建新Pod的Pod模板（template）；

3.Service

Service定义了Pod的逻辑集合和访问该集合的策略，是真实服务的抽象。Service提供了一个统一的服务访问入口以及服务代理和发现机制，用户不需要了解后台Pod是如何运行。

一个service定义了访问pod的方式，就像单个固定的IP地址和与其相对应的DNS名之间的关系。

Service其实就是我们经常提起的微服务架构中的一个“微服务”，通过分析、识别并建模系统中的所有服务为微服务——Kubernetes Service，最终我们的系统由多个提供不同业务能力而又彼此独立的微服务单元所组成，服务之间通过TCP/IP进行通信，从而形成了我们强大而又灵活的弹性网络，拥有了强大的分布式能力、弹性扩展能力、容错能力；



Pod、RC与Service的关系

如图示，每个Pod都提供了一个独立的Endpoint（Pod IP+ContainerPort）以被客户端访问，多个Pod副本组成了一个集群来提供服务，一般的做法是部署一个负载均衡器来访问它们，为这组Pod开启一个对外的服务端口如8000，并且将这些Pod的Endpoint列表加入8000端口的转发列表中，客户端可以通过负载均衡器的对外IP地址+服务端口来访问此服务。运行在Node上的kube-proxy其实就是一个智能的软件负载均衡器，它负责把对Service的请求

转发到后端的某个Pod实例上，并且在内部实现服务的负载均衡与会话保持机制。Service不是共用一个负载均衡器的IP地址，而是每个Servcie分配一个全局唯一的虚拟IP地址，这个虚拟IP被称为Cluster IP。

4.Label

Kubernetes中的任意API对象都是通过Label进行标识，Label的实质是一系列的K/V键值对。Label是Replication Controller和Service运行的基础，二者通过Label来进行关联Node上运行的Pod。

一个label是一个被附加到资源上的键/值对，譬如附加到一个Pod上，为它传递一个用户自定的并且可识别的属性.Label还可以被应用来组织和选择子网中的资源

selector是一个通过匹配labels来定义资源之间关系得表达式，例如为一个负载均衡的service指定所目标Pod

Label可以附加到各种资源对象上，一个资源对象可以定义任意数量的Label。给某个资源定义一个Label，相当于给他打一个标签，随后可以通过Label Selector（标签选择器）查询和筛选拥有某些Label的资源对象。我们可以通过给指定的资源对象捆绑一个或多个Label来实现多维度的资源分组管理功能，以便于灵活、方便的进行资源分配、调度、配置、部署等管理工作；

5.Node

Node是Kubernetes集群架构中运行Pod的服务节点（亦叫agent或minion）。Node是Kubernetes集群操作的单元，用来承载被分配Pod的运行，是Pod运行的宿主机。

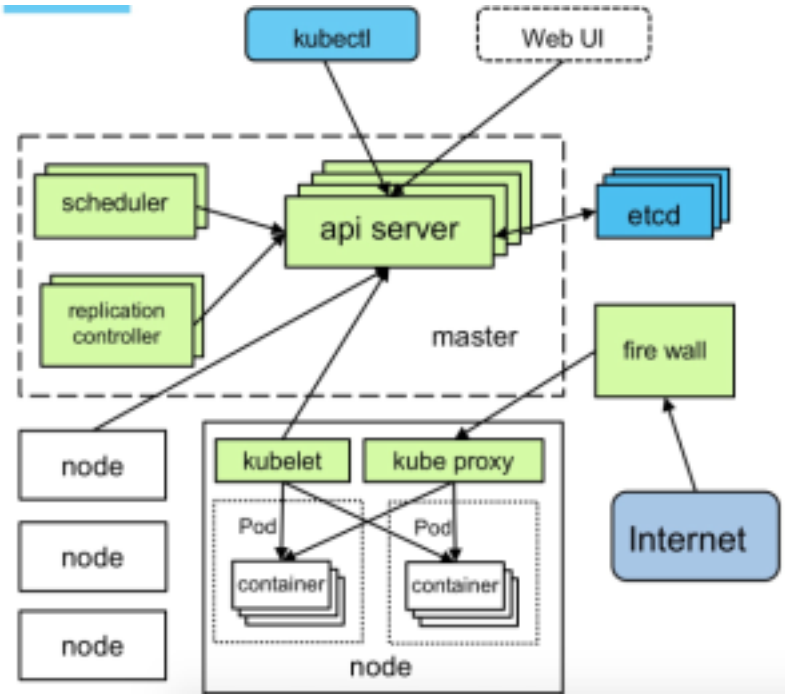
6.Endpoint（IP+Port）

标识服务进程的访问点；

注：Node、Pod、Replication Controller和Service等都可以看作是一种"资源对象"，几乎所有的资源对象都可以通过Kubernetes提供的kubectl工具执行增、删、改、查等操作并将其保存在ectd中持久化存储。

Kubernetes架构和组件

架构:



主从分布式架构，Master/Node

- 服务分组，小集群，多集群
- 服务分组，大集群，单集群

组件:

Kubernetes Master：

集群控制节点，负责整个集群的管理和控制，基本上Kubernetes所有的控制命令都是发给它，它来负责具体的执行过程，我们后面所有执行的命令基本都是在Master节点上运行的；

包含如下组件:

1.Kubernetes API Server

作为Kubernetes系统的入口，其封装了核心对象的增删改查操作，以RESTful API接口方式提供给外部客户和内部组件调用。维护的REST对象持久化到Etcd中存储。

2.Kubernetes Scheduler

为新建立的Pod进行节点(node)选择(即分配机器)，负责集群的资源调度。组件抽离，可以方便替换成其他调度器。

3.Kubernetes Controller

负责执行各种控制器，目前已经提供了很多控制器来保证Kubernetes的正常运行。

- Replication Controller

管理维护Replication Controller，关联Replication Controller和Pod，保证Replication Controller定义的副本数量与实际运行Pod数量一致。

- Node Controller

管理维护Node，定期检查Node的健康状态，标识出(失效|未失效)的Node节点。

- Namespace Controller

管理维护Namespace，定期清理无效的Namespace，包括Namesapce下的API对象，比如Pod、Service等。

- Service Controller

管理维护Service，提供负载以及服务代理。

- EndPoints Controller

管理维护Endpoints，关联Service和Pod，创建Endpoints为Service的后端，当Pod发生变化时，实时更新Endpoints。

- Service Account Controller

管理维护Service Account，为每个Namespace创建默认的Service Account，同时为Service Account创建Service Account Secret。

- Persistent Volume Controller

管理维护Persistent Volume和Persistent Volume Claim，为新的Persistent Volume Claim分配Persistent Volume进行绑定，为释放的Persistent Volume执行清理回收。

- Daemon Set Controller

管理维护Daemon Set，负责创建Daemon Pod，保证指定的Node上正常的运行Daemon Pod。

- Deployment Controller

管理维护Deployment，关联Deployment和Replication Controller，保证运行指定数量的Pod。当Deployment更新时，控制实现Replication Controller和 Pod的更新。

- Job Controller

管理维护Job，为Jod创建一次性任务Pod，保证完成Job指定完成的任务数目

- Pod Autoscaler Controller

实现Pod的自动伸缩，定时获取监控数据，进行策略匹配，当满足条件时执行Pod的伸缩动作。

Kubernetes Node：

除了Master，Kubernetes集群中的其他机器被称为Node节点，Node节点才是Kubernetes集群中的工作负载节点，每个Node都会被Master分配一些工作负载（Docker容器），当某个Node宕机，其上的工作负载会被Master自动转移到其他节点上去；

包含如下组件:

1.Kubelet

负责管控容器，Kubelet会从Kubernetes API Server接收Pod的创建请求，启动和停止容器，监控容器运行状态并汇报给Kubernetes API Server。

2.Kubernetes Proxy pod之间通信

负责为Pod创建代理服务，Kubernetes Proxy会从Kubernetes API Server获取所有的Service信息，并根据Service的信息创建代理服务，实现Service到Pod的请求路由和转发，从而实现Kubernetes层级的虚拟转发网络。

3.Docker Engine（docker），Docker引擎，负责本机的容器创建和管理工作；

常见面试问题

kubernetes（通常简称为K8S），是一个用于管理在容器中运行的应用的容器编排工具。

Kubernetes不仅有你所需要的用来支持复杂容器应用的所有东西，它还是市面上最方便开发和运维的框架。

Kubernetes的工作原理是通过将容器分组来把一个应用程序拆分成多个逻辑单元，以方便管理和发现。它对由小且独立的服务组成的微服务应用特别有用。

尽管Kubernetes运行在Linux上，他其实是平台无关的，可以在裸机、虚拟机、云实例或OpenStack上运行。

Cluster
Cluster 是计算、存储和网络资源的集合，Kubernetes 利用这些资源运行各种基于容器的应用。
Master
Master 是 Cluster 的大脑，它的主要职责是调度，即决定将应用放在哪里运行。Master 运行 Linux 操作系统，可以是物理机或者虚拟机。为了实现高可用，可以运行多个 Master。
Node
Node 的职责是运行容器应用。Node 由 Master 管理，Node 负责监控并汇报容器的状态，并根据 Master 的要求管理容器的生命周期。Node 运行在 Linux 操作系统，可以是物理机或者是虚拟机。
在前面交互式教程中我们创建的 Cluster 只有一个主机 host01，
它既是 Master 也是 Node。
Pod
Pod 是 Kubernetes 的最小工作单元。每个 Pod 包含一个或多个容器。Pod 中的容器会作为一个整体被 Master 调度到一个 Node 上运行。
Kubernetes 引入 Pod 主要基于下面两个目的：
可管理性。
有些容器天生就是需要紧密联系，一起工作。Pod 提供了比容器更高层次的抽象，将它们封装到一个部署单元中。Kubernetes 以 Pod 为最小单位进行调度、扩展、共享资源、管理生命周期。
通信和资源共享。
Pod 中的所有容器使用同一个网络 namespace，即相同的 IP 地址和 Port 空间。它们可以直接用 localhost 通信。同样的，这些容器可以共享存储，当 Kubernetes 挂载 volume 到 Pod，本质上是将 volume 挂载到 Pod 中的每一个容器。
Pods 有两种使用方式：
运行单一容器。
one-container-per-Pod 是 Kubernetes 最常见的模型，这种情况下，只是将单个容器简单封装成 Pod。即便是只有一个容器，Kubernetes 管理的也是 Pod 而不是直接管理容器。
运行多个容器。
但问题在于：哪些容器应该放到一个 Pod 中？
答案是：这些容器联系必须 非常紧密，而且需要 直接共享资源。
Controller
Kubernetes 通常不会直接创建 Pod，而是通过 Controller 来管理 Pod 的。Controller 中定义了 Pod 的部署特性，比如有几个副本，在什么样的 Node 上运行等。为了满足不同的业务场景，Kubernetes 提供了多种 Controller，包括 Deployment、ReplicaSet、DaemonSet、StatefuleSet、Job 等，我们逐一讨论。
Deployment 是最常用的 Controller，比如前面在线教程中就是通过创建 Deployment 来部署应用的。Deployment 可以管理 Pod 的多个副本，并确保 Pod 按照期望的状态运行。
ReplicaSet 实现了 Pod 的多副本管理。使用 Deployment 时会自动创建 ReplicaSet，也就是说 Deployment 是通过 ReplicaSet 来管理 Pod 的多个副本，我们通常不需要直接使用 ReplicaSet。
DaemonSet 用于每个 Node 最多只运行一个 Pod 副本的场景。正如其名称所揭示的，DaemonSet 通常用于运行 daemon。
StatefuleSet 能够保证 Pod 的每个副本在整个生命周期中名称是不变的。而其他 Controller 不提供这个功能，当某个 Pod 发生故障需要删除并重新启动时，Pod 的名称会发生变化。同时 StatefuleSet 会保证副本按照固定的顺序启动、更新或者删除。
Job 用于运行结束就删除的应用。而其他 Controller 中的 Pod 通常是长期持续运行。
Service

Deployment 可以部署多个副本，每个 Pod 都有自己的 IP，外界如何访问这些副本呢？
通过 Pod 的 IP 吗？
要知道 Pod 很可能会被频繁地销毁和重启，它们的 IP 会发生变化，用 IP 来访问不太现实。
答案是 Service。
Kubernetes Service 定义了外界访问一组特定 Pod 的方式。Service 有自己的 IP 和端口，Service 为 Pod 提供了负载均衡。
Kubernetes 运行容器（Pod）与访问容器（Pod）这两项任务分别由 Controller 和 Service 执行。
Namespace
如果有多个用户或项目组使用同一个 Kubernetes Cluster，如何将他们创建的 Controller、Pod 等资源分开呢？
答案就是 Namespace。
Namespace 可以将一个物理的 Cluster 逻辑上划分成多个虚拟 Cluster，每个 Cluster 就是一个 Namespace。不同 Namespace 里的资源是完全隔离的。
Kubernetes 默认创建了两个 Namespace。
default -- 创建资源时如果不指定，将被放到这个 Namespace 中。
kube-system -- Kubernetes 自己创建的系统资源将放到这个 Namespace 中。
API Server（kube-apiserver）
API Server 提供 HTTP/HTTPS RESTful API，即 Kubernetes API。API Server 是 Kubernetes Cluster 的前端接口，各种客户端工具（CLI 或 UI）以及 Kubernetes 其他组件可以通过它管理 Cluster 的各种资源。
Scheduler（kube-scheduler）
Scheduler 负责决定将 Pod 放在哪个 Node 上运行。Scheduler 在调度时会充分考虑 Cluster 的拓扑结构，当前各个节点的负载，以及应用对高可用、性能、数据亲和性的需求。
Controller Manager（kube-controller-manager）
Controller Manager 负责管理 Cluster 各种资源，保证资源处于预期的状态。Controller Manager 由多种 controller 组成，包括 replication controller、endpoints controller、namespace controller、serviceaccounts controller 等。
不同的 controller 管理不同的资源。例如 replication controller 管理 Deployment、StatefulSet、DaemonSet 的生命周期，namespace controller 管理 Namespace 资源。
etcd
etcd 负责保存 Kubernetes Cluster 的配置信息和各种资源的状态信息。当数据发生变化时，etcd 会快速地通知 Kubernetes 相关组件。
Pod 网络
Pod 要能够相互通信，Kubernetes Cluster 必须部署 Pod 网络，flannel 是其中一个可选方案。
kubelet
kubelet 是 Node 的 agent，当 Scheduler 确定在某个 Node 上运行 Pod 后，会将 Pod 的具体配置信息（image、volume 等）发送给该节点的 kubelet，kubelet 根据这些信息创建和运行容器，并向 Master 报告运行状态。
kube-proxy
service 在逻辑上代表了后端的多个 Pod，外界通过 service 访问 Pod。service 接收到的请求是如何转发到 Pod 的呢？这就是 kube-proxy 要完成的工作。
每个 Node 都会运行 kube-proxy 服务，它负责将访问 service 的 TCP/UPD 数据流转发到后端的容器。如果有多个副本，kube-proxy 会实现负载均衡。

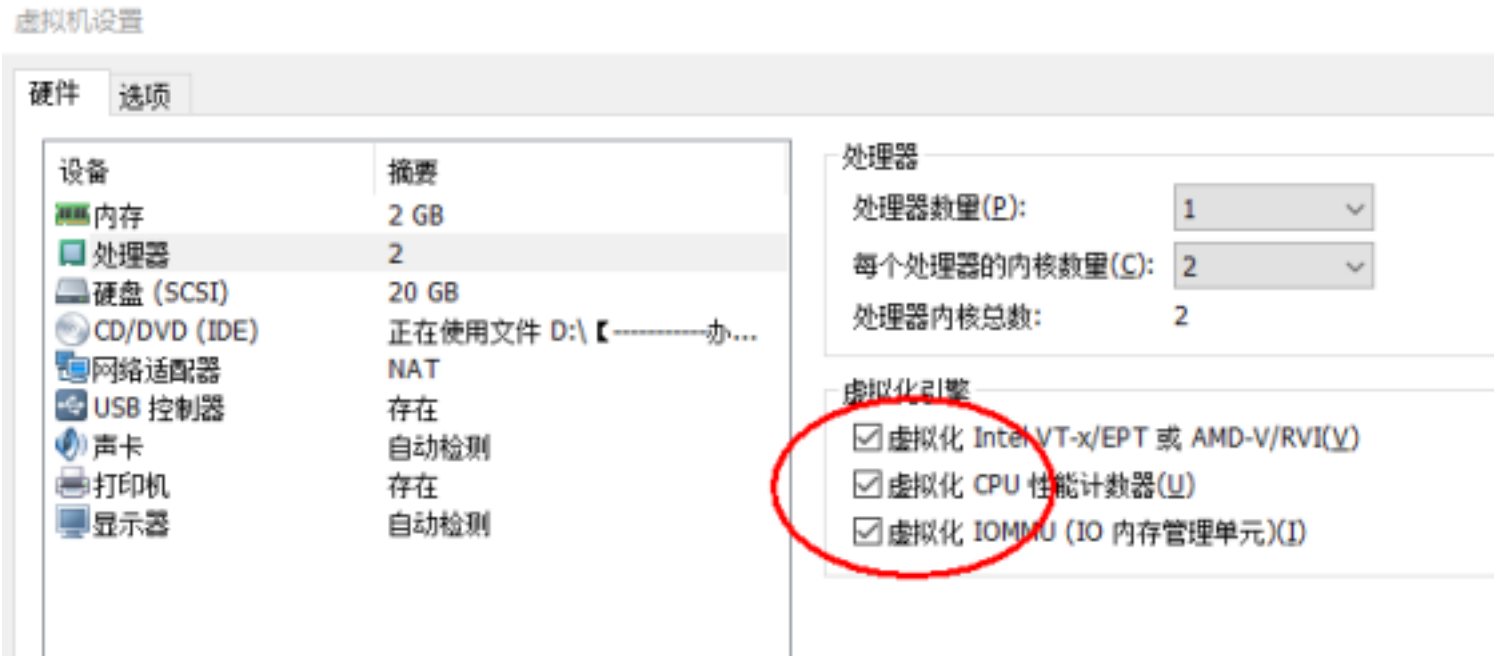
kubernetes部署

生产环境都是二进制安装（预编译安装）
yum无法自定义安装
编译安装可以自定义安装
预编译介于两者之间

准备三台服务器：

1、一台master
配置： IP：192.168.2.11 ，2H2G
2、两台node
配置： IP：192.168.2.12，192.168.2.13 ， 1H1G

三台虚拟机需要打开虚拟化：



moster控制节点

==========master节点操作==========
1、修改各主机名并配置解析与双击互信：
修改主机名：
hostnamectl set-hostname master
hostnamectl set-hostname node1
hostnamectl set-hostname node2

配置主机解析
vim /etc/hosts
192.168.2.11 master
192.168.2.12 node1
192.168.2.13 node2
生成公钥并传输
ssh-keygen
ssh-copy-id node1
ssh-copy-id node2
传输主机解析文件
for i in node{1..2}
do
scp /etc/hosts $i:/etc/hosts
done

2、更新YUM源（网络源）
rm -rf /etc/yum.repos.d/*
mv /root/yum.repos.d/* /etc/yum.repos.d/ （本地镜像需要联网才能使用）
yum clean all && yum makecache

3、安装docker（网络源）
yum install docker-ce -y　(# cd docker-ce && yum -y install *.rpm （本地源）)

4、查看kubernetes相关安装包
yum list all | grep "^kub"

5、下载kubeadm以及相关插件服务
yum install kubeadm kubectl kubelet -y

6、添加阿里云加速器和科技大加速器（阿里云的要自己的账号注册替换我下面的）
# mkdir /etc/docker && cd /etc/docker/ && vim daemon.json
{
  "registry-mirrors": [
    "https://registry.docker-cn.com","https://0gacarv3.mirror.aliyuncs.com" //3台都做一下
  ]
}

蓝底色部分可换成自己的加速器地址

7、启动docker与
systemctl daemon-reload   //上一步的原因，我们在启动docker之前现重新加载守护进程

systemctl start docker
systemctl enable docker.service

systemctl start kubelet.service
systemctl enable kubelet.service

8、下载（加载）镜像

docker load --input kebert-master1.14.1.tar

9、查看镜像

docker images

10、设置忽略swap//防止其他设备干扰
swapoff 或者
vim /etc/sysconfig/kubelet

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

11、安装k8s

kubeadm init --kubernetes-version="v1.14.1" --pod-network-cidr="10.244.0.0/16" --ignore-preflight-errors=Swap //忽视报错

#初始化输出的最后一行内容复制到文本中后面安装node节点，保存连接秘钥

```
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 192.168.2.11:6443 --token ogoiff.ijbxfkel50gdickh \
    --discovery-token-ca-cert-hash sha256:d69d18e1d9d461c61154f627f6113af154fc354c252e4b6e354cebecd8640b8e
[root@master ~]#
```

kubeadm join 192.168.2.37:6443 --token 5rikld.gc1kg1e7pjva6drs \
    --discovery-token-ca-cert-hash
sha256:00556debaadc73ae4ac034cfc13d9c70c44ef38e2864d48f6d014bdabf096b27

vim k8s.txt

```
kubeadm join 192.168.2.11:6443 --token ogoiff.ijbxfkel50gdickh \
    --discovery-token-ca-cert-hash sha256:d69d18e1d9d461c61154f627f6113af154fc354c252e4b6e354cebecd8640b8e
~
~
~
~
~
```

mkdir /$USER.kube (# mkdri /root/.kube)

cp /etc/kubernetes/admin.conf .kube/config

12、安装flannel插件(node上不用配置)+

kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/
Documentation/kube-flannel.yml   //github上的公有仓库

```
[root@master ~]#  kubectl  apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/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
[root@master ~]#
```

13、#查看集群中的pod 看是不是running 状态

kubectl get pods -n kube-system

初始状态如下：（需让子弹飞一会）

```
[root@master ~]# kubectl get pods -n kube-system
NAME                              READY   STATUS     RESTARTS   AGE
coredns-fb8b8dccf-66bjf           0/1     Pending    0          5m51s
coredns-fb8b8dccf-cvm8c           0/1     Pending    0          5m51s
etcd-master                       1/1     Running    0          4m53s
kube-apiserver-master             1/1     Running    0          4m49s
kube-controller-manager-master    1/1     Running    0          5m10s
kube-flannel-ds-amd64-trwz6       0/1     Init:0/1   0          10s
kube-proxy-7jnxp                  1/1     Running    0          5m52s
kube-scheduler-master             1/1     Running    0          5m11s
```

如下表示启动成功：

```
[root@master ~]# kubectl get pods -n kube-system
NAME                              READY   STATUS    RESTARTS   AGE
coredns-fb8b8dccf-66bjf           1/1     Running   0          7m18s
coredns-fb8b8dccf-cvm8c           1/1     Running   0          7m18s
etcd-master                       1/1     Running   0          6m20s
kube-apiserver-master             1/1     Running   0          6m16s
kube-controller-manager-master    1/1     Running   0          6m37s
kube-flannel-ds-amd64-trwz6       1/1     Running   0          97s
kube-proxy-7jnxp                  1/1     Running   0          7m19s
kube-scheduler-master             1/1     Running   0          6m38s
[root@master ~]#
```

手动配置**YUM**源**--docker-ce**

```
[docker-ce-stable]
name=Docker CE Stable - $basearch
baseurl=https://mirrors.aliyun.com/docker-ce/linux/centos/7/$basearch/stable
enabled=1
gpgcheck=1
gpgkey=https://mirrors.aliyun.com/docker-ce/linux/centos/gpg

[docker-ce-stable-debuginfo]
name=Docker CE Stable - Debuginfo $basearch
baseurl=https://mirrors.aliyun.com/docker-ce/linux/centos/7/debug-$basearch/stable
enabled=0
gpgcheck=1
gpgkey=https://mirrors.aliyun.com/docker-ce/linux/centos/gpg

[docker-ce-stable-source]
name=Docker CE Stable - Sources
baseurl=https://mirrors.aliyun.com/docker-ce/linux/centos/7/source/stable
enabled=0
gpgcheck=1
gpgkey=https://mirrors.aliyun.com/docker-ce/linux/centos/gpg

[docker-ce-edge]
name=Docker CE Edge - $basearch
baseurl=https://mirrors.aliyun.com/docker-ce/linux/centos/7/$basearch/edge
enabled=0
gpgcheck=1
gpgkey=https://mirrors.aliyun.com/docker-ce/linux/centos/gpg

[docker-ce-edge-debuginfo]
name=Docker CE Edge - Debuginfo $basearch
baseurl=https://mirrors.aliyun.com/docker-ce/linux/centos/7/debug-$basearch/edge
enabled=0
gpgcheck=1
gpgkey=https://mirrors.aliyun.com/docker-ce/linux/centos/gpg
```

手动配置**YUM**源**--kubernetes**

vim /etc/yum.repos.d/kube.repo

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

node节点

首先关闭防火墙和selinux

1、安装docker的YUM源

2、安装kubernetes的YUM源(#手动添加阿里云的kubernetes源)

3、安装docker

yum install docker-ce -y　(# cd docker-ce && yum -y install *.rpm　)

4、安装kubeadm kubectl kubelet

yum install kubeadm kubectl kubelet -y

5、设置忽略swap设备

vim /etc/sysconfig/kubelet

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

6、添加阿里云加速器和科技大加速器（阿里云的要自己的账号注册替换我下面的）

mkdir /etc/docker

vim /etc/docker/daemon.json

```
{
  "registry-mirrors": [
    "https://registry.docker-cn.com","https://5jvldcev.mirror.aliyuncs.com"
  ]
}
```

7、启动docker与kuberlet并开机启动

systemctl daemon-reload && systemctl start docker

systemctl enable docker.service

systemctl start kubelet.service

systemctl enable kubelet.service
12、安装flannel插件

kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml   //github上的公有仓库 //未安装成功也可以加入集群

8、注入docker镜像并加入k8s集群

docker load --input k8s-kube-1.14.1.tar

 # kubeadm join 192.168.2.11:6443 --token ogoiff.ijbxfkel50gdickh \
    --discovery-token-ca-cert-hash sha256:d69d18e1d9d461c61154f627f6113af154fc354c252e4b6e354cebecd8640b8e \
    --ignore-preflight-errors=Swap /没添加会报swap错误

错误

// 错误1

```
[root@master ~]# kubeadm init --kubernetes-version="v1.14.1" --pod-network-cidr="10.244.0.0/16" --ignore-preflight-errors
=Swap
[init] Using Kubernetes version: v1.14.1
[preflight] Running pre-flight checks
        [WARNING Service-Docker]: docker service is not enabled, please run 'systemctl enable docker.service'
        [WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the Docker cgroup driver. The recommended driver is "syste
md". Please follow the guide at https://kubernetes.io/docs/setup/cri/
        [WARNING Swap]: running with swap on is not supported. Please disable swap
        [WARNING Service-Kubelet]: kubelet service is not enabled, please run 'systemctl enable kubelet.service'
error execution phase preflight: [preflight] Some fatal errors occurred:
        [ERROR FileContent--proc-sys-net-bridge-bridge-nf-call-iptables]: /proc/sys/net/bridge/bridge-nf-call-iptables co
ntents are not set to 1
[preflight] If you know what you are doing, you can make a check non-fatal with `--ignore-preflight-errors=...`
[root@master ~]# echo 1 > /proc/sys/net/bridge/bridge-nf-call-iptables
```

解决：

echo 1 > /proc/sys/net/bridge/bridge-nf-call-iptables