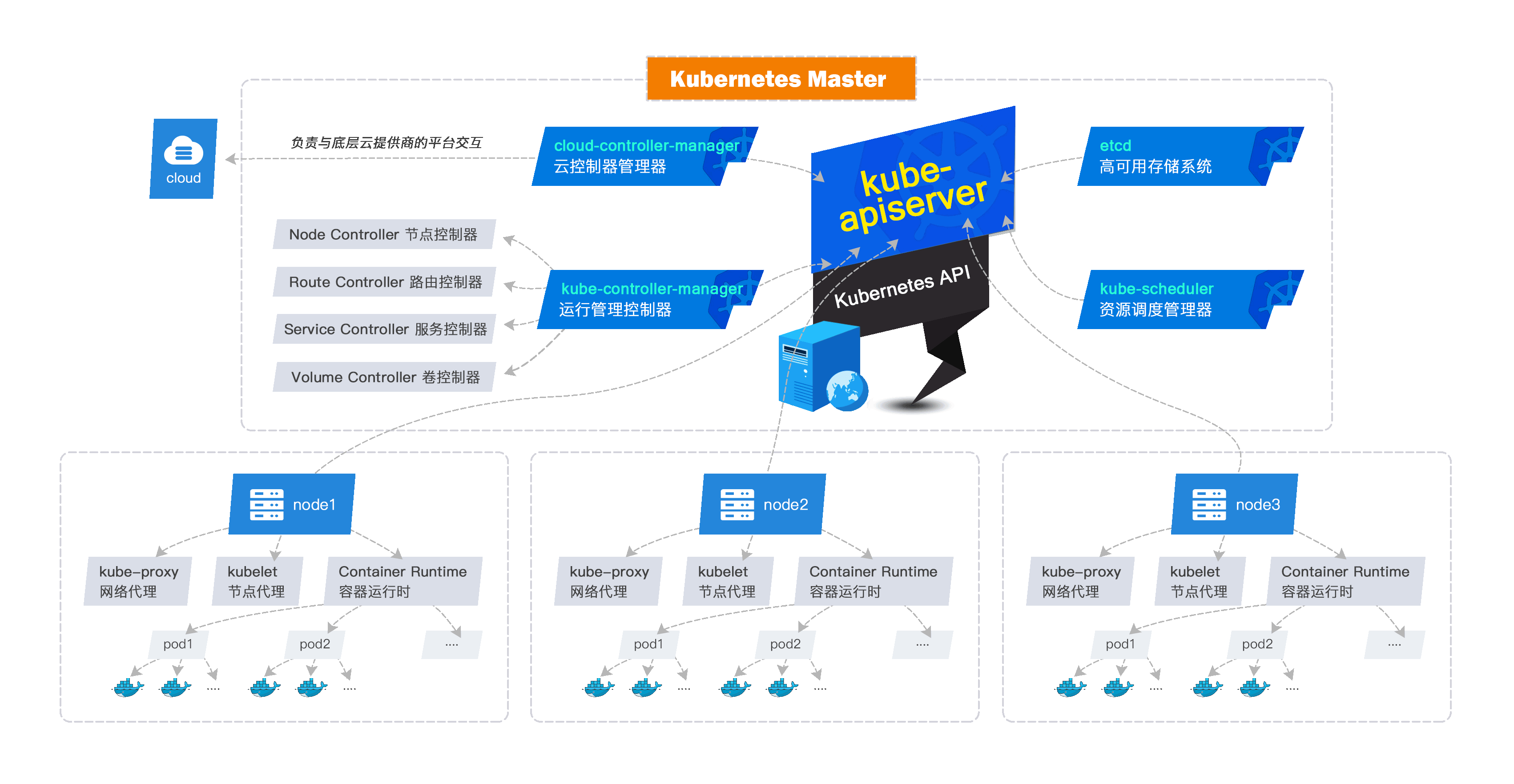
Kubernetes

# Kubernetes主体架构



## 主要核心组件

### Master组件

Master为集群控制管理节点，负责整个集群的管理和控制。Master的组件如下所示：

1）kube-apiserver

kube-apiserver用于暴露Kubernetes API，提供了资源操作的唯一入口。任何的资源请求/调用操作都是通过kube-apiserver提供的接口进行。

2）etcd

etcd是Kubernetes的高可用的一致性键值存储系统，也是其提供的默认的存储系统，用于存储所有的集群数据，使用时需要为etcd数据提供备份计划。

3）kube-scheduler

kube-scheduler 监视新创建没有分配到Node的Pod，为Pod选择一个Node以供其运行。

4）kube-controller-manager

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。逻辑上，每个控制器是一个单独的进程，但为了降低复杂性，它们都被编译成单个二进制文件，并在单个进程中运行。

这些控制器包括：

* 节点（Node）控制器：负责在节点出现故障时警示和响应。
* 副本（Replication）控制器：负责为系统中的每个副本控制器对象维护正确的pod数量。
* 端点（Endpoints）控制器：填充Endpoints对象（即连接Services＆Pods）。
* Service Account和Token控制器：为新的Namespace创建默认帐户访问API 访问Token。

5）cloud-controller-manager

云控制器管理器负责与底层云提供商的平台交互。云控制器管理器是Kubernetes版本1.6中引入的。

云控制器管理器仅运行云提供商特定的（controller loops）控制器循环。可以通过将--cloud-provider flag设置为external启动kube-controller-manager ，来禁用控制器循环。

cloud-controller-manager 具体功能：

* 节点（Node）控制器：检查云端节点，以确保节点在停止响应之后在云中是否删除。
* 路由（Route）控制器：用于在底层云基础架构中设置路由。
* 服务（Service）控制器：用于创建，更新和删除云提供商的负载均衡器。
* 卷（Volume）控制器：用于创建，附加和装载卷，以及与云提供商交互以协调卷。

### 节点(Node)组件

Node是k8s集群中的工作负载节点，用于被Master分配工作负载（容器）。Node的组件有：

1）kubelet

kubelet是节点代理，它会监视已分配给节点的pod，确保容器在pod中运行。

2）kube-proxy

kube-proxy为节点的网络代理，通过在主机上维护网络规则并执行连接转发来实现Kubernetes的服务抽象。

kube-proxy负责请求转发。kube-proxy允许通过一组后端功能进行TCP和UDP流转发或循环TCP和UDP转发。

3）Container Runtime

容器运行时是负责运行容器的软件。

Kubernetes支持多个容器运行时：Docker， containerd，cri-o， rktlet以及Kubernetes CRI（容器运行时接口）的任何实现。目前最佳组合为Kubernetes+Docker。

### 插件

插件是实现集群功能的pod和服务，他们扩展了Kubernetes的功能。主要的插件有：

* DNS

Kubernetes的集群DNS扩展插件用于支持k8s集群系统中各服务之间的发现和调用。

* Web UI (Dashboard)

Dashboard（仪表盘）是Kubernetes集群的基于Web的通用UI，它允许用户管理群集，以及管理集群中运行的应用程序。

* Container Resource Monitoring（容器资源监测）

Container Resource Monitoring工具提供了UI监测容器、Pods、服务以及整个集群中的数据，用于检查Kubernetes集群中，应用程序的性能。

* Cluster-level Logging（集群级日志记录）

Cluster-level Logging提供了容器日志存储，并且提供了搜索/浏览界面。

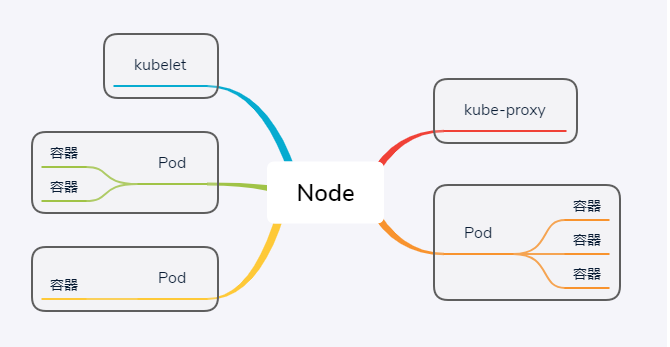
## 基本概念

初步了解了Kubernetes的主题架构和核心组件，我们还需要进一步了解一些Kubernetes的基本概念。

### 容器组(Pod)

* Pod是k8s集群中运行部署应用或服务的最小单元，一个Pod由一个或多个容器组成。在一个Pod中，容器共享网络和存储，并且在一个Node上运行。
* Kubernetes为每个Pod都分配了唯一的IP地址，称之为Pod IP，一个Pod里的多个容器共享Pod IP地址。Kubernetes要求底层网络支持集群内任意两个Pod之间的TCP/IP直接通信，这通常采用虚拟二层网络技术来实现，例如Flannel、Open vSwitch等。因此，在Kubernetes里，一个Pod里的容器与另外主机上的Pod容器能够直接通信。
* Pod有两种类型：普通的Pod和静态Pod（Static Pod），静态Pod不存放在etcd存储里，而是存放在某个具体的Node上的一个具体文件中，并且只在此Node上启动运行。普通的Pod一旦被创建，就会被存储到etcd中，随后会被Kubernetes Master调度到某个具体的Node上并进行绑定（Binding），该Node上的kubelet进程会将其实例化成一组相关的容器并启动起来。当Pod里的某个容器停止时，Kubernetes会自动检测到这个问题并且重新启动这个Pod（重启Pod里的所有容器）；如果Pod所在的Node宕机，则会将这个Node上的所有Pod重新调度到其他节点上运行。

Pod、容器与Node的关系如下图：



apiVersion: v1

kind: Pod

metadata:

name: myapp-pod

labels:

app: myapp

spec:

containers:

- name: myapp-container

image: busybox

command: ['sh', '-c', 'echo Hello Kubernetes! && sleep 3600']

### 服务（Service）

在Kubernetes中，Pod会经历“生老病死”而无法复活，也就是说，分配给Pod的IP会随着Pod的销毁而消失，这就导致一个问题——如果有一组Pod组成一个集群来提供服务，某些Pod提供后端服务API，某些Pod提供前端界面UI，那么该如何保证前端能够稳定地访问这些后端服务呢？这就是Service的由来。

Service在Kubernetes中是一个抽象的概念，它定义了一组逻辑上的Pod和一个访问它们的策略（通常称之为微服务）。这一组Pod能够被Service访问到，通常是通过Label选择器确定的。

例如，一个图片处理的后端程序，它运行了3个副本，这些副本是可互换的——前端程序不需要关心它们调用了哪个后端副本。虽然组成这一组的后端程序的Pod实际上可能会发生变化，但是前端无需知道也没必要知道，也不需要跟踪后端的状态。Service的抽象解耦了这种关联。

apiVersion: v1

kind: Service

metadata:

name: my-service

spec:

selector:

app: MyApp

ports:

- protocol: TCP

port: 80

targetPort: 9376

### 卷(Volume)

和Docker不同，Kubernetes的Volume定义在Pod上，被一个Pod里的多个容器挂载到具体的文件目录下，当容器终止或者重启时，Volume中的数据也不会丢失。也就是说，在Kubernetes中，Volume是Pod中能够被多个容器访问的共享目录。

目前，Kubernetes支持以下类型的卷：

* awsElasticBlockStore

awsElasticBlockStore可以挂载AWS上的EBS盘到容器，需要Kubernetes运行在AWS的EC2上。

* azureDisk

Azure是微软提供的公有云服务，如果使用Azure上面的虚拟机来作为Kubernetes集群使用时，那么可以通过AzureDisk这种类型的卷插件来挂载Azure提供的数据磁盘。

* azureFile

AzureFileVolume用于将Microsoft Azure文件卷（SMB 2.1和3.0）挂载到Pod中。

* cephfs

cephfs Volume可以将已经存在的CephFS Volume挂载到pod中，与emptyDir特点不同，pod被删除的时，cephfs仅被被卸载，内容保留。cephfs能够允许我们提前对数据进行处理，而且这些数据可以在Pod之间“切换”。

* cinder

cinder用于将OpenStack Cinder Volume安装到Pod中。

* configMap

configMap提供了一种将配置数据注入Pod的方法。存储在ConfigMap对象中的数据可以在configMap类型的卷中引用，然后由在Pod中运行的容器化应用程序使用。

* csi

Container Storage Interface（CSI）为Kubernetes定义了一个标准接口，以将任意存储系统暴露给其容器工作负载。在Kubernetes集群上部署CSI兼容卷驱动程序后，用户可以使用csi卷类型来附加，装载等CSI驱动程序公开的卷。

* downwardAPI

downwardAPI可以将Pod和Container字段公开给正在运行的Container。

* emptyDir

使用emptyDir时，Pod分配给节点时就会首先创建卷，并且只要Pod在该节点上运行，这个卷就会一直存在。当Pod被删除时，emptyDir中的数据也不复存在。

* fc (fibre channel)

光纤通道区域存储网络，需要购买支持FC的磁盘阵列设备、控制器、光纤、光接口以及与设置相匹配的软件。

* flocker

flocker是一个开源的容器集群数据卷管理器，它提供由各种存储后端支持的数据卷的管理和编排。

* gcePersistentDisk

gcePersistentDisk可以挂载GCE（Google的云计算引擎）上的永久磁盘到容器，需要Kubernetes运行在GCE的VM中。与emptyDir不同，Pod删除时，gcePersistentDisk被删除，但Persistent Disk的内容任然存在。这就意味着gcePersistentDisk能够允许我们提前对数据进行处理，而且这些数据可以在Pod之间“切换”。

* glusterfs

glusterfs，允许将Glusterfs（一个开源网络文件系统）Volume安装到pod中。不同于emptyDir，Pod被删除时，Volume只是被卸载，内容被保留。

* hostPath

hostPath允许挂载Node上的文件系统到Pod里面去。如果Pod需要使用Node上的文件，可以使用hostPath。

* iscsi

iscsi允许将iscsi磁盘挂载到pod中，Pod被删除时，Volume只是被卸载，内容被保留。

* local

Local 是Kubernetes集群中每个节点的本地存储（如磁盘，分区或目录），在Kubernetes1.7中kubelet可以支持对kube-reserved和system-reserved指定本地存储资源。

通过上面的这个新特性可以看出来，Local Storage同HostPath的区别在于对Pod的调度上，使用Local Storage可以由Kubernetes自动的对Pod进行调度，而是用HostPath只能人工手动调度Pod，因为Kubernetes已经知道了每个节点上kube-reserved和system-reserved设置的本地存储限制。

但是，本地卷仍受基础节点可用性的限制，并不适用于所有应用程序。如果节点变得不健康，则本地卷也将变得不可访问，并且使用它的Pod将无法运行。使用本地卷的应用程序必须能够容忍这种降低的可用性以及潜在的数据丢失，具体取决于底层磁盘的持久性特征。

* nfs

NFS是Network File System的缩写，即网络文件系统。Kubernetes中通过简单地配置就可以挂载NFS到Pod中，而NFS中的数据是可以永久保存的，同时NFS支持同时写操作。Pod被删除时，Volume被卸载，内容被保留。这就意味着NFS能够允许我们提前对数据进行处理，而且这些数据可以在Pod之间相互传递。

使用NFS数据卷适用于多读多写的持久化存储，适用于大数据分析、媒体处理、内容管理等场景。

* persistentVolumeClaim

persistentVolumeClaim用来挂载持久化磁盘。PersistentVolumes是用户在不知道特定云环境的细节的情况下，实现持久化存储（如GCE PersistentDisk或iSCSI卷）的一种方式。

* projected

Projected volume将多个Volume源映射到同一个目录，目前，可以支持以下类型的卷源：secret、downwardAPI、configMap、serviceAccountToken。

所有源都需要与Pod在同一名称空间中。

* portworxVolume

portworxVolume是一个运行在Kubernetes中的弹性块存储层，可以通过Kubernetes动态创建，也可以在Kubernetes Pod中预先配置和引用。它能够聚合多个服务器之间的容量，可以将服务器变成一个聚合的高可用的计算和存储节点。

* quobyte

Quobyte是Quobyte公司推出的分布式文件存储系统，它实现了Container Storage Interface（CSI，容器存储接口）。

* rbd

RBD允许Rados Block Device格式的磁盘挂载到Pod中，同样的，当pod被删除的时候，rbd也仅仅是被卸载，内容保留。

RBD的一个特点是它可以同时由多个消费者以只读方式安装，但是不允许同时写入。这意味着我们可以使用数据集预填充卷，然后根据需要从多个Pod中并行使用。

* scaleIO

ScaleIO是一种基于软件的存储平台（虚拟SAN），可以使用现有硬件来创建可扩展的共享块网络存储的集群。ScaleIO卷插件允许部署的pod访问现有的ScaleIO卷。

* secret

secret volume用于将敏感信息（如密码）传递给pod。我们可以将secrets存储在Kubernetes API中，使用的时候以文件的形式挂载到pod中，而无需直接连接Kubernetes。secret volume由tmpfs（RAM支持的文件系统）支持，因此它们永远不会写入非易失性存储。

* storageos

StorageOS是一家英国的初创公司，给无状态容器提供简单的自动块存储、状态来运行数据库和其他需要企业级存储功能。StorageOS在Kubernetes环境中作为Container运行，从而可以从Kubernetes集群中的任何节点访问本地或附加存储。可以复制数据以防止节点故障。精简配置和内容压缩可以提高利用率并降低成本。

StorageOS的核心是为容器提供块存储，可通过文件系统访问。StorageOS Container需要64位Linux，并且没有其他依赖项。提供免费的开发人员许可。

* vsphereVolume

vsphereVolume用于将vSphere VMDK Volume挂载到Pod中。卸载卷后，内容将被保留。它同时支持VMFS和VSAN数据存储。

### 标签（Labels）和标签选择器

Labels其实就是附加到对象（例如Pod）上的键值对。给某个资源对象定义一个Label，就相当于给它打了一个标签，随后就可以通过Label Selector（标签选择器）来查询和筛选拥有某些Label的资源对象，Kubernetes通过这种方式实现了类似SQL的简单又通用的对象查询机制。

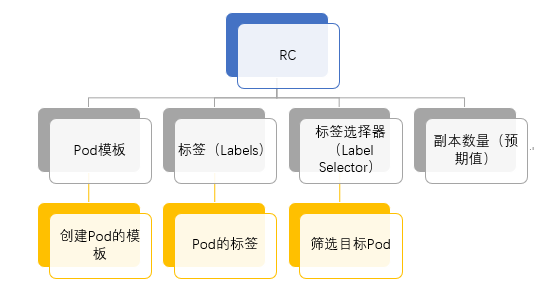
总的来说，使用Label可以给对象创建多组标签，Label和Label Selector共同构成了Kubernetes系统中最核心的应用模型，使得被管理对象能够被精细地分组管理，同时实现了整个集群的高可用性。

### 复制控制器（Replication Controller, RC）

RC的作用是保障Pod的副本数量在任意时刻都符合某个预期值，不多也不少。如果多了，就杀死几个，如果少了，就创建几个。

RC有点类似于进程管理程序，但是它不是监视单个节点上的各个进程，而是监视多个节点上的多个pod，确保Pod的数量符合预期值。

RC的定义由以下内容组成：



当我们定义了一个RC并提交到Kubernetes集群中后，Master节点上的Controller Manager组件就得到通知，定期巡检系统中当前存活的目标Pod，并确保目标Pod实例的数量刚好等于此RC的期望值。如果有过多的Pod副本在运行，系统就会停掉多余的Pod；如果运行的Pod副本少于期望值，即如果某个Pod挂掉，系统就会自动创建新的Pod以保证数量等于期望值。

通过RC，Kubernetes实现了用户应用集群的高可用性，并且大大减少了运维人员在传统IT环境中需要完成的许多手工运维工作（如主机监控脚本、应用监控脚本、故障恢复脚本等）。

**常见使用场景：**

* 重新规划

比如重新设置Pod数量。

* 缩放
* 滚动更新

RC支持滚动更新，也就是允许我们在更新服务时，逐个的替换Pod。也就是说，滚动更可以保障应用的可用性，确保任何时间都有可用的Pod来提供服务。

* 多个发布版本追踪

除了在程序更新过程中同时可以运行多个版本的程序外，也可以在更新完成之后的一段时间内或者持续的同时运行多个版本（新旧）。需通过标签选择器来完成。

### 副本集控制器（Replica Set, RS）

ReplicaSet是下一代复制控制器。ReplicaSet和 Replication Controller之间的目前的唯一区别是选择器的支持——Replica Set支持基于集合的Label selector（Set-based selector），而RC只支持基于等式的Label selector（equality-based selector），所以Replica Set的功能更强大。

Replica Set很少单独使用，它主要被Deployment（部署）这个更高层的资源对象所使用，从而形成一整套Pod创建、删除、更新的编排机制。

### 部署控制器（Deployment）

Deployment（部署控制器）为Pod和Replica Set提供声明式更新。

我们只需要在Deployment中描述想要的目标状态是什么，Deployment controller就会帮我们将Pod和Replica Set的实际状态改变到目标状态。我们可以定义一个全新的Deployment，也可以创建一个新的替换旧的Deployment。

Deployment相对于RC的最大区别是我们可以随时知道当前Pod“部署”的进度。一个Pod的创建、调度、绑定节点及在目标Node上启动对应的容器这一完整过程需要一定的时间，所以我们期待系统启动N个Pod副本的目标状态，实际上是一个连续变化的“部署过程”导致的最终状态。

Deployment的典型使用场景有以下几个：

* 创建一个Deployment对象来生成对应的Replica Set并完成Pod副本的创建过程。
* 检查Deployment的状态来看部署动作是否完成（Pod副本的数量是否达到预期的值）。
* 更新Deployment以创建新的Pod（比如镜像升级）。
* 如果当前Deployment不稳定，则回滚到一个早先的Deployment版本。
* 暂停Deployment以便于一次性修改多个Pod Template Spec的配置项，之后再恢复Deployment，进行新的发布。
* 扩展Deployment以应对高负载。
* 查看Deployment的状态，以此作为发布是否成功的指标。
* 清理不再需要的旧版本ReplicaSet。

### StatefulSet

StatefulSet用于管理有状态应用程序，比如Mysql集群、MongoDB集群ZooKeeper集群等。

StatefulSet主要特性如下：

* StatefulSet里的每个Pod都有稳定、唯一的网络标识，可以用来发现集群内的其他成员。
* 稳定的持久化存储，即Pod重新调度后还是能访问到相同的持久化数据，基于PersistentVolume来实现，删除Pod时默认不会删除与StatefulSet相关的存储卷（为了保证数据的安全）。
* 有序部署，有序扩展，即Pod是有顺序的，在部署或者扩展的时候要依据定义的顺序依次依次进行（即从0到N-1，在下一个Pod运行之前所有之前的Pod必须都是Running和Ready状态），基于init containers来实现。
* 有序收缩，有序删除。

### 后台服务支撑(DaemonSet)

DaemonSet保证在每个Node上都运行一个容器副本，常用来部署一些集群的日志、监控或者其他系统管理应用。典型的应用包括：

* 日志收集守护程序，比如fluentd，logstash等。
* 系统监控，比如Prometheus Node Exporter，collectd，New Relic agent，Ganglia gmond等。
* 集群存储后台进程，比如glusterd，ceph等。
* 系统程序，比如kube-proxy, kube-dns, glusterd, ceph等。

### 一次性任务(Job)

Job负责批量处理短暂的一次性任务 (short lived one-off tasks)，即仅执行一次的任务，它保证批处理任务的一个或多个Pod成功结束。

Kubernetes支持以下几种Job：

非并行任务。

具有固定完成计数要求的并行任务。

带有工作队列的并行任务。

# 使用Minikube部署本地Kubernetes集群

## 什么是kubernets集群

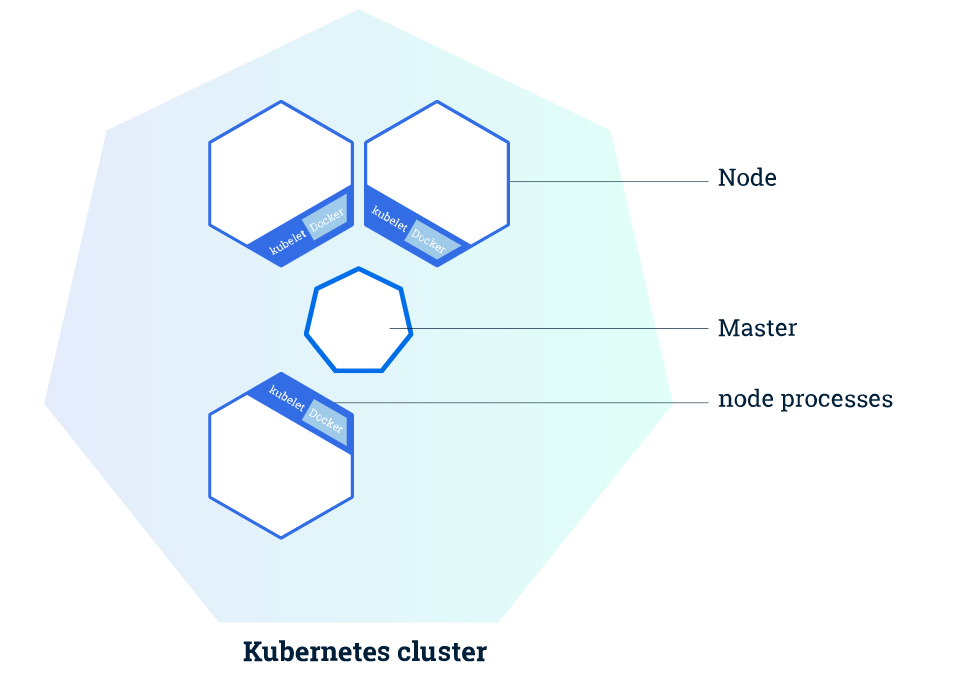
Kubernetes 用于协调高度可用的计算机集群，这些计算机被连接作为单个工作单元。 Kubernetes允许用户将容器化的应用程序部署到集群，而不必专门将其绑定到单个计算机。为了利用这种新的部署模型，应用程序需要被容器化。容器化应用程序比过去的部署模型更灵活和可用——而不是将应用程序直接安装到特定机器上，作为深入集成到主机中的软件包。Kubernetes 在一个集群上以更有效的方式自动分发和调度容器应用程序。

Kubernetes 集群由两种类型的资源组成:

Master：即集群的调度节点，负责管理集群，例如调度应用程序、维护应用程序的所需状态、扩展应用程序和滚动更新。

Nodes：即应用程序实际运行的工作节点，可以是物理机或者虚拟机。每个工作节点都有一个 Kubelet（节点代理），它是管理节点并与Kubernetes Master节点进行通信的代理。节点上还应支持容器操作，例如 Docker或rkt。一个 Kubernetes 工作集群至少有三个节点。

当我们在Kubernetes上部署应用程序时， Master会启动应用程序容器，并调度容器在集群的Nodes上运行，而Nodes使用Master公开的Kubernetes API与Master进行通信。最终用户还可以直接使用 Kubernetes 的API与集群交互



## 使用Minikube创建本地Kubernetes实验环境

在大部分情况下，我们需要在本地玩转Kubernetes，以便于Kubernetes应

用程序的开发和调测。搭建完整的Kubernetes集群毕竟太重，那么使Minikube

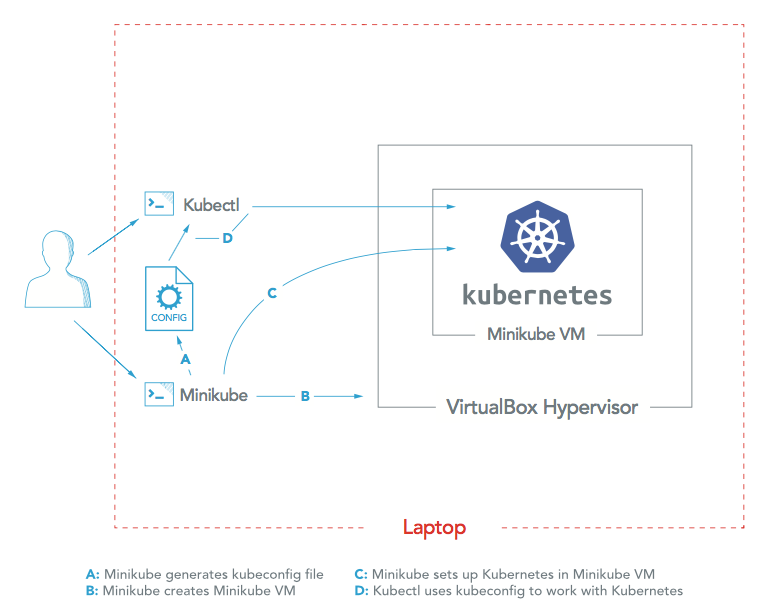
则是不二选择。

## 什么是Minikube

Minikube 是一个轻量级的Kubernetes实现，会在本机创建一台虚拟机，并部署一个只包含一个节点的简单集群。 Minikube适用于Linux, Mac OS和Windows系统。Minikube CLI提供了集群的基本引导操作，包括启动、停止、状态和删除。

Minikube的目标是成为本地Kubernetes应用程序开发的最佳工具，并支持所有适合的Kubernetes功能

官方GitHub地址：[*https://github.com/kubernetes/minikube*](https://github.com/kubernetes/minikube)



Minikube支持以下Kubernetes功能：

* DNS
* NodePorts（可使用“minikube service”命令来管理）
* ConfigMaps和Secrets
* 仪表板（Dashboards，minikube dashboard）
* 容器运行时：Docker，rkt，CRI-O和containerd
* Enabling CNI（容器网络接口）
* Ingress
* LoadBalancer（负载均衡，可以使用“minikube tunnel”命令来启用）
* Multi-cluster（多集群，可以使用“minikube start -p <name>”命令来启用）
* Persistent Volumes
* RBAC
* 通过命令配置apiserver和kubelet

## Windows 10安装

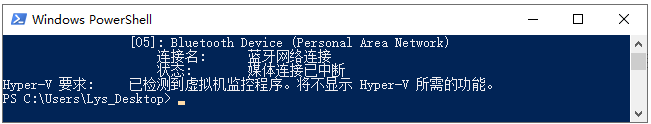
1. 安装要求

Windows必须支持虚拟化，可以执行“systeminfo”命令来确认。如果支持虚拟化，则【Hyper-V要求】一栏如下图所示：



C:\Users\Lys\_Desktop\Documents\Tencent Files\512982554\Image\C2C\M%YWM2VE`R)$F2KF`{~M{LU.jpg

如果已经装了Hyper-V，则提示如下：



1. 启用Hyper-V（推荐）

可以通过【程序和功能】=》【打开或关闭 Windows 功能】=》勾选【Hyper-V】来启用Hyper-V，也可以通过管理员执行以下Powershell脚本：

Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V -All

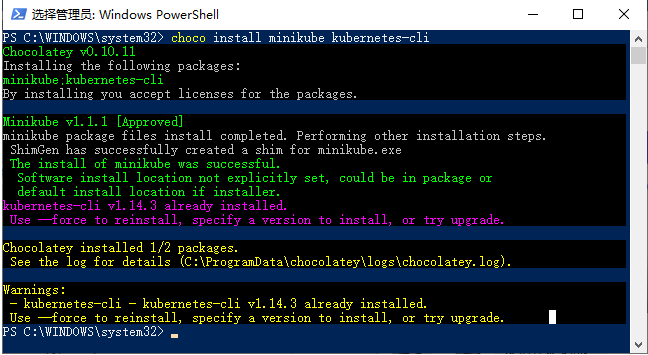
如果不支持启用Hyper-V，大家可以安装“VirtualBox”，而且目前“VirtualBox”是官方默认的虚拟机管理程序。

1. 安装Minikube和kubectl

* 使用Chocolatey安装Minikube（推荐）

Chocolatey我们前面已经进行了讲解，这里面我们可以使用Chocolatey以管理员身份一键安装Minikube：

choco install minikube kubernetes-cli



PS C:\WINDOWS\system32> choco install minikube kubernetes-cli

Chocolatey v0.10.11

Installing the following packages:

minikube;kubernetes-cli

By installing you accept licenses for the packages.

Minikube v1.1.1 [Approved]

minikube package files install completed. Performing other installation steps.

ShimGen has successfully created a shim for minikube.exe

The install of minikube was successful.

Software install location not explicitly set, could be in package or

default install location if installer.

kubernetes-cli v1.14.3 already installed.

Use --force to reinstall, specify a version to install, or try upgrade.

Chocolatey installed 1/2 packages.

See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

Warnings:

- kubernetes-cli - kubernetes-cli v1.14.3 already installed.

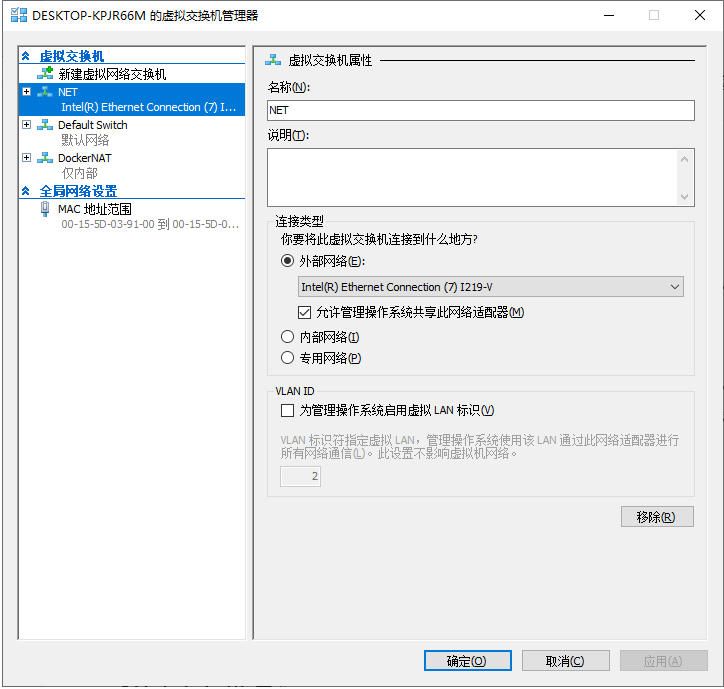
Use --force to reinstall, specify a version to install, or try upgrade.

* 通过下载安装包安装

下载地址：*https://github.com/kubernetes/minikube/releases/*

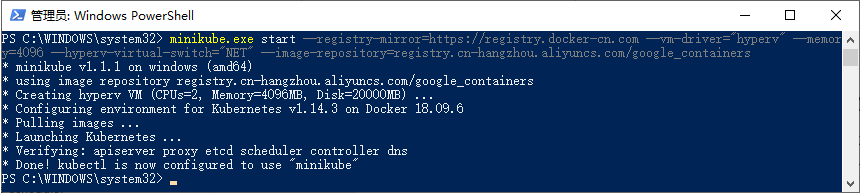
需下载【minikube-windows-amd64.exe】，下载完成后，需要重命名为“minikube.exe”进行使用。

1. 启动Minikube
2. Minikube在Windows上支持使用VirtualBox和Hyper-V，这里我们使用Hyper-V进行实践。
3. 我们需要执行“minikube start”命令来启动Minikube。这个过程中会下载Minikube ISO镜像，如果Minikube ISO镜像下载失败，可复制链接手工下载或者配置容器代理再试。如果是手工下载，下载后，请将ISO文件放置C:\Users\<用户名>\.minikube\cache\iso目录，然后再次执行“start”命令。具体命令如下所示：
4. 需使用管理员执行以下Powershell脚本：
5. minikube.exe start --registry-mirror=https://registry.docker-cn.com --vm-driver="hyperv" --memory=4096
6. 其中，--registry-mirror参数用于设置镜像服务地址，这里设置为国内镜像服务地址。--vm-driver参数设置了虚拟机类型，这里我们使用Hyper-V，默认是VirtualBox。--memory参数设置了虚拟机内存大小。执行此脚本后，会使用默认的Hyper-V的虚拟交换机，我们也可以使用参数--hyperv-virtual-switch进行指定指定的虚拟网络交换机。虚拟网络交换机设置如下图所示：



在中国，由于网络和防火墙的原因，通常会无法拉取k8s相关镜像或者下载速度过于缓慢，因此，我们可以通过参数--image-repository来设置Minikube使用阿里云镜像。如下列命令：

minikube.exe start --registry-mirror=https://registry.docker-cn.com --vm-driver="hyperv" --memory=4096 --hyperv-virtual-switch="NET" --image-repository=registry.cn-hangzhou.aliyuncs.com/google\_containers



PS C:\WINDOWS\system32> minikube.exe start --registry-mirror=https://registry.docker-cn.com --vm-driver="hyperv" --memory=4096 --hyperv-virtual-switch="NET" --image-repository=registry.cn-hangzhou.aliyuncs.com/google\_containers

\* minikube v1.1.1 on windows (amd64)

\* using image repository registry.cn-hangzhou.aliyuncs.com/google\_containers

\* Creating hyperv VM (CPUs=2, Memory=4096MB, Disk=20000MB) ...

\* Configuring environment for Kubernetes v1.14.3 on Docker 18.09.6

\* Pulling images ...

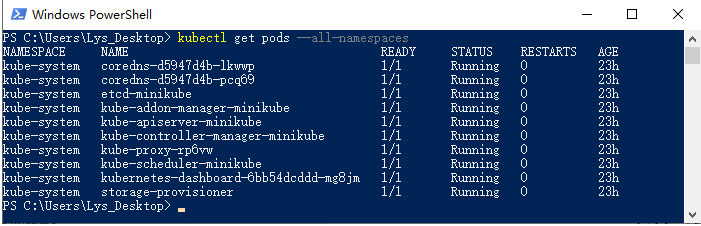
\* Launching Kubernetes ...

\* Verifying: apiserver proxy etcd scheduler controller dns

\* Done! kubectl is now configured to use "minikube"

成功之后，我们就可以使用kubectl来操作集群了，比如查看当前所有pod的状态：

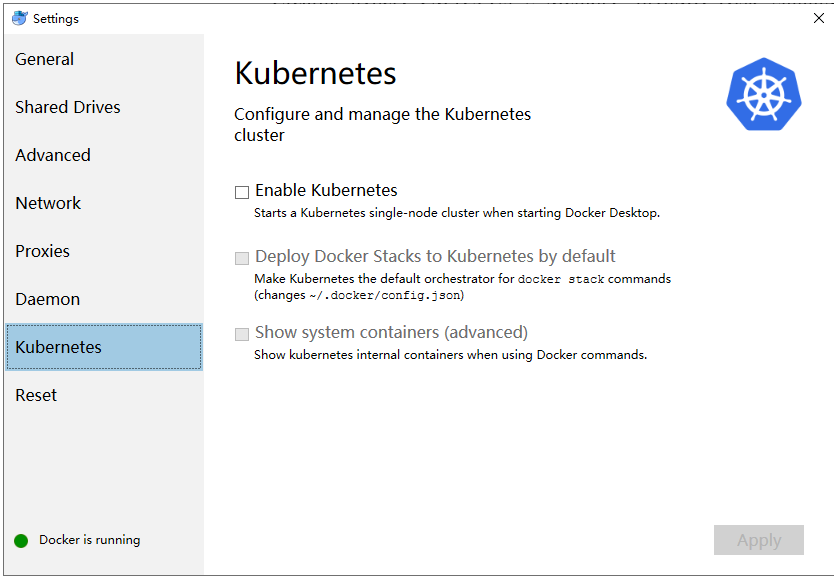
kubectl get pods --all-namespaces



刚才我们使用Minikube创建了默认的集群，我们还可以使用Minikube创建新的集群，比如：minikube start -p mycluster值得注意的是，Minikube搭配Hyper-V使用需要禁用动态内存（Docker for Windows初始化时指定禁用了相关虚拟机使用动态内存），执行Powershell脚本如下所示：

Set-VMMemory -VMName 'minikube' -DynamicMemoryEnabled $false。

在Windows 10下，我们还可以使用docker-desktop来启用k8s，不过由于网络的原因，并不是很推荐：



打开Minikube可视化面板

成功启动Minikube之后，我们就可以通过以下命令来打开Minikube可视化

面板:

minikube dashboard

启用面板

PS C:\WINDOWS\system32> minikube dashboard

\* Enabling dashboard ...

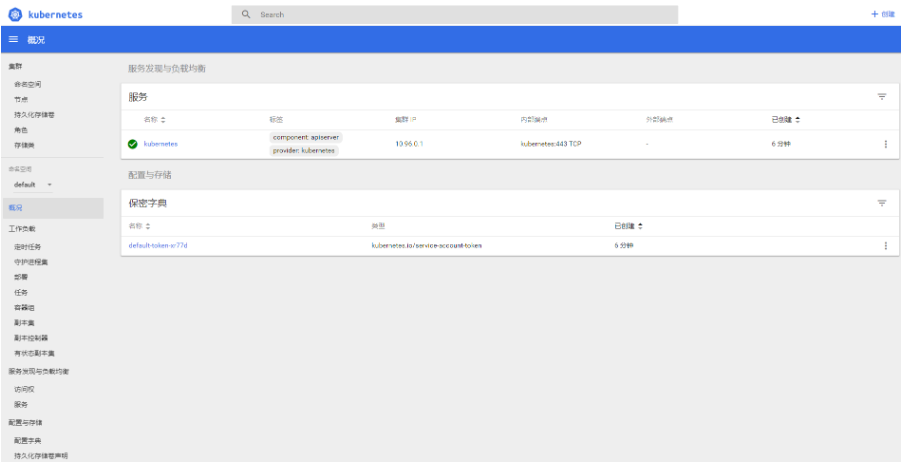
\* Verifying dashboard health ...

\* Launching proxy ...

\* Verifying proxy health ...

\* Opening http://127.0.0.1:3173/api/v1/namespaces/kube-system/services/http:kubernetes-dashboard:/proxy/ in your default browser...





常见错误

启动Minikube时提示错误如下：X Unable to start VM: start: exit status 1

处理方案：执行“minikube delete”或者手工清理时虚拟机目录存在残留内容，需手动删除目录“C:\Users\{your username} \.minikube\machines”后再次尝试。

## Linux安装

1. 安装kubectl

由于Google网络不太稳定，我们使用阿里云镜像进行安装。

* CentOS

echo '#k8s

[kubernetes]

name=Kubernetes

baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86\_64

enabled=1

gpgcheck=0

'>/etc/yum.repos.d/kubernetes.repo

#kubeadm和相关工具包

yum -y install kubelet kubeadm kubectl kubernetes-cni

* Debian / Ubuntu

apt-get update && apt-get install -y apt-transport-https

curl -s https://mirrors.aliyun.com/kubernetes/apt/doc/apt-key.gpg | apt-key add -

echo 'deb https://mirrors.aliyun.com/kubernetes/apt/ kubernetes-xenial main' >/etc/apt/sources.list.d/kubernetes.list

apt-get update

apt-get install -y kubelet kubeadm kubectl

1. 安装Minikube

这里我们直接下载安装：

curl -Lo minikube [https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 \](https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64%20\)

&& chmod +x minikube

由于网络问题，我们可以选择使用阿里云的执行程序：

curl -Lo minikube https://kubernetes.oss-cn-hangzhou.aliyuncs.com/minikube/releases/latest/minikube-linux-amd64 \

&& chmod +x minikube

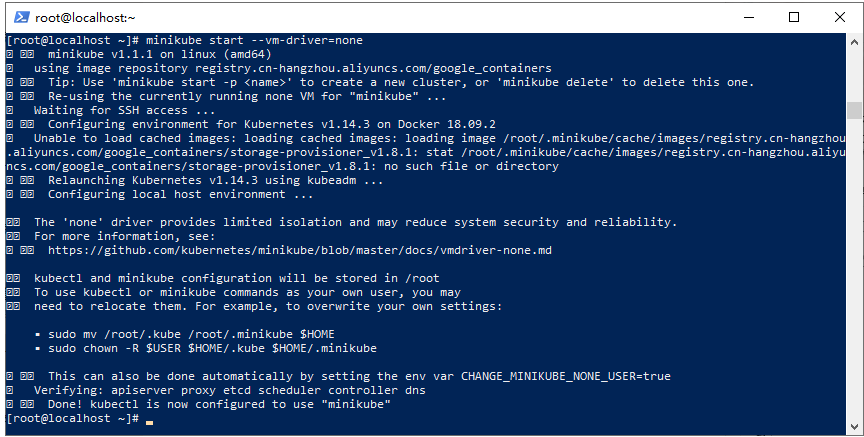
然后将可执行文件添加到/usr/local/bin目录下

sudo install minikube /usr/local/bin

1. 启动Minikube

minikube start --vm-driver=none

如果存在网络问题，请使用--image-repository=registry.cn-hangzhou.aliyuncs.com/google\_containers指定镜像仓库地址。



安装过程中如出现问题，可以执行以下命令之后再重新尝试：

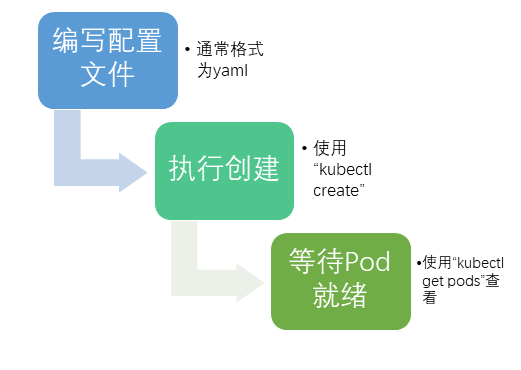
minikube delete

rm ~/.minikube

# 使用kubectl管理k8s集群

# 使用kubectl部署应用

## Kubectl部署流程



## 部署一个简单的Demo网站

通过创建Kubernetes Deployment对象来运行应用程序。那么我们需要编写一个YAML文件来定义Deployment对象。

1. 编写Deployment对象的配置文件

在开始之前，我们需要对Deployment对象的配置有初步的了解。官方介绍文档如下所示:

<https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.15/#deployment-v1-apps>

根据官方标准，我们定义了一个简单的Deployment配置：

apiVersion: apps/v1 #API对象版本，可通过“kubectl api-versions”命令查看

kind: Deployment #资源类型，区分大小写，可通过“kubectl api-resources”命令查看，这里使用Deployment对象

metadata: #标准的元数据

name: demo-deployment #当前Deployment对象名称，同一个命名空间下必须唯一

spec: #部署规范（目标），Deployment控制器会根据此模板调整当前Pod到最终的期望状态

replicas: 5 # Pod数量，这里指运行5个Pod

selector: #选择器，其定义了Deployment控制器如何找到要管理的Pod

matchLabels: #匹配标签

app: demo #待匹配的标签键值对

template: # Pod模板定义

metadata: #标准的元数据

labels: #Pod标签

app: demo #定义Pod标签，由键值对组成

spec: #Pod规范

containers: #容器列表，Pod中至少有一个容器

- name: demo #容器名称

image: microsoft/dotnet-samples:aspnetapp #镜像地址

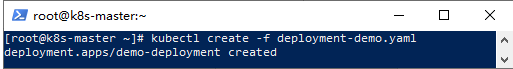
ports: #端口列表

- containerPort: 80 #设置容器端口

如上面定义所示，我们定义了一个简单的部署示例，它将使用创建一个ReplicaSet对象以利用复制控制器创建5个Pod来运行“dotnet-samples”。

1. 使用“kubectl create”执行资源创建

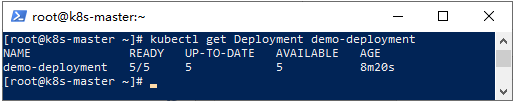
YAML文件准备好了，接下来我们来执行创建：kubectl create -f deployment-demo.yaml



结合配置，这里有几个重点说明一下：

* 如上面配置所示，部署名称为“demo-deployment”。
* 此部署对象将创建5个复制的Pod，由replicas字段决定。如上图所示该部署创建了5个Pod。
* selector字段定义了Deployment控制器如何找到要管理的Pod，所以标签的键值对一定不能出错。
* template字段则定义了Pod模板，其子字段labels定义了Pod 的标签，spec字段则定义了容器。

执行创建部署之后，我们可以通过命令“kubectl get Deployment demo-deployment”来检查部署对象是否已经创建，部署是否已经完成：

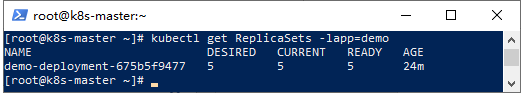


如上图所示，上述字段分别代表：

* READY代表是否已就绪，左侧数字表示当前已运行的副本数，右侧表示所需的副本数。
* UP-TO-DATE表示已更新已实现预期状态的副本数。
* AVAILABLE则表示用户可以使用的应用程序副本数。
* AGE表示应用已运行的时间。

通用的，我们可以运行以下命令来查看副本集（ReplicaSet）对象：

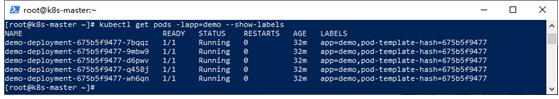
kubectl get ReplicaSets -lapp=demo



通过上图可知，我们创建Deployment对象的过程实际上就是生成对应的副本集对象（Replica Set）并完成Pod副本的创建过程。

值得注意的是，副本集的名称格式为[部署名称]-[随机字符串]。随机字符串是随机生成的，并使用pod-template-hash作为种子。如何查看pod-template-hash呢？使用如下命令即可：

kubectl get pods -lapp=demo --show-labels



如上图所示，5个Pod已经部署完成。