Kubernetes实战训练营（一）

一、Docker

```
# docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

-e：设置环境变量；-e username=zhdya

–name：为容器指定一个名称；–name=zhdya

-p：指定端口映射，格式为：主机(宿主)端口:容器端口 -p 80:8080

-t：为容器重新分配一个伪输入终端，通常与 -i 同时使用；

-i：以交互模式运行容器，通常与 -t 同时使用；

-d：后台运行容器，并返回容器ID；

-v：宿主机目录:容器目录。将宿主机目录挂载到容器内。

```
# docker run -d -p 3306:3306 \
--name mysql --privileged=true \
-v /data/mysql/data:/var/lib/mysql \
-e MYSQL_ROOT_PASSWORD=123456 \
mysql
```

1.1、不常用但较为重要的命令

```
docker cp /tmp/123.txt 96f7f14e99ab:/www/

docker pause mysql / docker unpause mysql

docker inspect mysql

docker stats mysql

docker info

docker save -o ubuntu_v3.tar xxxx.xxxx.com/ubuntu:v3
docker load -i ubuntu_v3.tar

docker commit -m="描述信息" -a="作者" 容器id 目标镜像名:[TAG]
```

1.2、Docker的网络模式

1）**桥接模式（Bridge）**：**默认**的网络模式，容器通过一个虚拟的网桥接口与主机相连，并且可以通过端口映射将容器内部的端口映射到主机上。桥接模式适用于单个主机上多个容器相互通信的场景。

2）**主机模式（Host）**：容器与主机共享网络命名空间，使得容器和主机可以直接使用同一套网络设备。主机模式可以大幅度提供容器的网络性能，例如，容器化的高吞吐量数据库等。

3）**容器模式（Container）**：每个容器都拥有自己独立的网络命名空间，容器之间可以通过IP地址进行通信，但与主机网络隔离。容器模式适用于需要在多个容器之间建立私有网络的场景。

4）**无网络模式（None）**：容器没有任何网络接口，完全与外部网络隔离，只能通过主机与容器进行通信。无网络模式适用于与外部网络完全隔离、仅在主机内部进行通信的场景。

1.3、Dockerfile

```
FROM python:3.7-slim-buster
 # 设置工作目录
WORKDIR /app
 # 复制应用程序代码到工作目录中
COPY . /app
 # 运行更新并安装依赖
RUN apt-get update && \
    apt-get install -y git && \
    pip install -r requirements.txt
```

```
 # 设置环境变量
ENV FLASK_APP app.py
ENV FLASK_ENV production
 # 暴露应用程序运行的端口
EXPOSE 5000
 # 启动应用程序
CMD [ "flask", "run", "--host=0.0.0.0" ]
```

ADD和COPY的区别

- **COPY**：它只能从宿主机复制文件或目录到Docker容器中。在复制时，原始文件的元数据将保留。它可以在上下文目录内或外使用。
- **ADD**：它不仅可以从宿主机复制文件或目录到Docker容器中，还可以在复制时进行一些额外的操作。例如，它可以自动解压缩压缩文件（如.tar、.gzip、.bz2）或远程URL，并将其复制到Docker容器中。
- 但是，由于它的功能更为复杂，因此更易出错。因此，如果只需要将文件或目录从宿主机复制到Docker容器中，则应该优先使用COPY命令。

例子：

```
FROM ubuntu:18.04
 # 复制宿主机中的压缩文件到Docker容器中
ADD example.tar.gz /app
 # 设置工作目录
WORKDIR /app
 # 显示工作目录中的文件列表
RUN ls -l
 # 启动应用程序
CMD [ "npm", "start" ]
```

CMD和ENTRYPOINT的区别?

CMD 指令：

- CMD 指令用于设置容器启动后默认执行的命令。可以在 Dockerfile 中多次使用 CMD，但只有最后一个 CMD 会生效。
- CMD 指令可以包含可执行文件、shell 命令或参数。
- 如果在运行容器时提供了命令行参数，CMD 指令中的命令会被忽略，而运行时传入的命令行参数会替代 CMD 指令中的命令。

ENTRYPOINT 指令：

- ENTRYPOINT 指令也用于设置容器启动时要执行的命令，但与 CMD 不同的是，ENTRYPOINT 的命令不会被忽略，它始终会被执行。
- ENTRYPOINT 还可以与 CMD 结合使用，CMD 中的参数会作为 ENTRYPOINT 指令中命令的默认参数。
- ENTRYPOINT 指令通常用于定义容器的主要可执行程序或服务，而 CMD 则用于提供默认参数或可选的附加命令。

```
FROM ubuntu:latest

# 定义容器启动时默认执行的命令
CMD ["echo", "Hello, World!"]

# 定义容器启动时要执行的主命令，可以与 CMD 结合使用
ENTRYPOINT ["echo", "Running"]

# CMD 中的参数会作为 ENTRYPOINT 的默认参数
CMD ["default"]
```

构建并运行这个容器，将会输出 `Running default`，因为 CMD 提供的参数会作为 ENTRYPOINT 命令的默认参数。

在运行容器时提供了额外的命令行参数，如 `docker run image hello`，那么输出将会是 `Running hello`，运行时的命令行参数会替代 CMD 的默认参数。

总结：CMD 用于设置容器默认的执行命令，可以被覆盖，而 ENTRYPOINT 则始终会被执行，并且可以与 CMD 结合使用以提供默认参数。

1.4、docker-compose

```
docker-compose up -d nginx                          构建建启动nignx容器

docker-compose exec nginx bash                      登录到nginx容器中

docker-compose down                                 删除所有nginx容器,镜像

docker-compose ps                                   显示所有容器

docker-compose restart nginx                        重新启动nginx容器

docker-compose run --no-deps --rm php-fpm php -v    在php-fpm中不启动关联容器，并容器执行php -v 执行完成后删除容器

docker-compose build nginx                          构建镜像 。

docker-compose build --no-cache nginx               不带缓存的构建。

docker-compose logs  nginx                          查看nginx的日志

docker-compose logs -f nginx                        查看nginx的实时日志

docker-compose config  -q                           验证（docker-compose.yml）文件配置，当配置正确时，不输出任何内容，当文件配置错误，输出错误信息。

docker-compose events --json nginx                  以json的形式输出nginx的docker日志

docker-compose pause nginx                          暂停nignx容器

docker-compose unpause nginx                        恢复ningx容器

docker-compose rm nginx                             删除容器（删除前必须关闭容器）
```

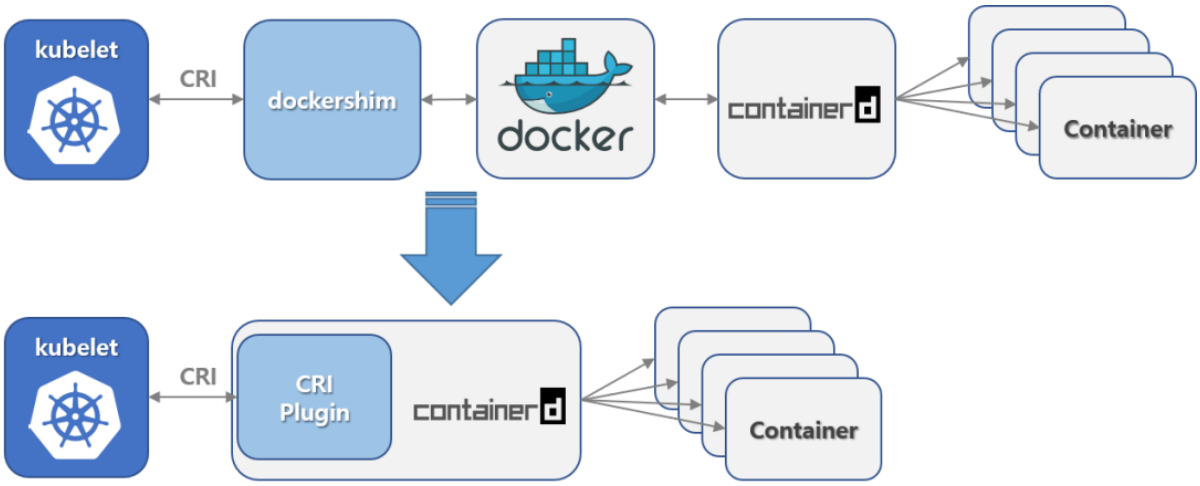
```
docker-compose stop nginx                    停止nignx容器

docker-compose start nginx                   启动nignx容器
```

二、Containerd

2.1、kubelet？ docker? containerd?

- docker 作为 k8s 容器运行时，调用关系为：`kubelet --> dockershim （在 kubelet 进程中） --> dockerd --> containerd`
- containerd 作为 k8s 容器运行时，调用关系为：`kubelet --> cri plugin（在 containerd 进程中） --> containerd`



混淆注意点：

`ctr` 是 containerd 的一个客户端工具。

`crictl` 是 CRI 兼容的容器运行时命令行接口，可以使用它来检查和调试 k8s 节点上的容器运行时和应用程序。

2.2、常用命令

命令	docker	ctr（containerd）	crictl（kubernetes）
显示本地镜像列表	docker images	ctr i ls	crictl images
下载镜像	docker pull	ctr i pull	crictl pull
上传镜像	docker push	ctr i push	无
删除本地镜像	docker rmi	ctr i rm	crictl rmi
查看镜像详情	docker inspect	无	crictl inspecti
重命名镜像	docker tag	ctr i tag	无
导出镜像	docker export	ctr i export	无
导入镜像	docker import	ctr i import	无
构建镜像	docker build	无	无

命令	docker	ctr（containerd）	crictl（kubernetes）
显示容器列表	docker ps	ctr c ls	crictl ps
创建容器	docker create	ctr c create	crictl create
启动容器	docker start	ctr t start	crictl start
停止容器	docker stop	无	crictl stop
删除容器	docker rm	ctr c rm	crictl rm
查看容器详情	docker inspect	ctr c info	crictl inspect
attach	docker attach	ctr t attach	crictl attach
exec	docker exec	ctr t exec	crictl exec
logs	docker logs	无	crictl logs
stats	docker stats	ctr t metrics	crictl stats
显示pod列表	无	无	crictl pods
查看pod详情	无	无	crictl inspectp
运行pod	无	无	crictl runp
停止pod	无	无	crictl stopp
删除pod	无	无	crictl rmp

三、Kubernetes

3.1、为什么使用Kubernetes?

1）自动化容器编排：Kubernetes 提供了自动化的容器编排功能，可以根据定义的规则和策略自动部署、调度和管理容器。它可以根据资源需求、负载均衡和健康检查等条件自动调整容器的数量和位置。

2）弹性扩展：Kubernetes 允许根据工作负载的变化自动扩展或收缩容器。通过水平扩展，可以根据需要增加或减少容器的副本数量，以满足不同的流量和负载需求。

3）服务发现和负载均衡：Kubernetes 提供了内建的 DNS 服务和负载均衡功能，可以自动将网络请求路由到正确的容器。通过标签和选择器，可以定义服务的访问方式，并实现容器之间的通信和协作。

4）自我修复：Kubernetes 具有自我修复能力，可以监测容器的状态并在发生故障时自动进行恢复。如果容器崩溃或无响应，Kubernetes 将重新启动容器，并确保系统的可用性和稳定性。

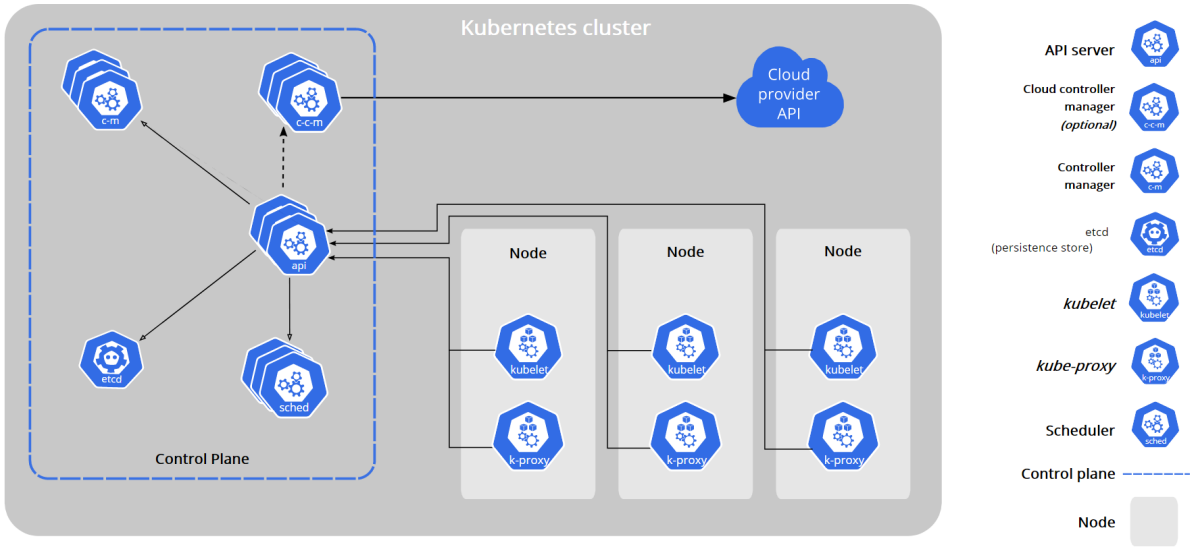
5）水平伸缩：Kubernetes 可以根据需求进行水平扩展，动态调整副本数以匹配流量的增长或减少。这样可以提高应用程序的性能和容量，并且可以根据需求自动缩减资源消耗。

6）配置和存储管理：Kubernetes 提供了灵活的配置管理和存储管理功能。可以通过 ConfigMaps 和 Secrets 管理应用程序的配置项和敏感信息，而 Persistent Volumes 和存储类（StorageClasses）允许对持久化数据进行统一管理和分配。

7）多云和混合云支持：Kubernetes 支持多云和混合云环境，可以在不同的云提供商或私有数据中心中部署和管理应用程序。这使得应用程序具备更高的灵活性和可移植性，可以根据需求选择最适合的部署环境。

3.2、Kubernetes集群架构及组件

一个Kubernetes集群至少有一个主控制平面节点（Control Plane）和一台或者多台工作节点（Node）组成，控制面板和工作节点实例可以是物理设备或云中的实例。Kubernetes 架构如下：



3.2.1、Kubernetes 控制平面（Contorl Plane）

Kubernetes控制平面也称为主节点（Master Node），其管理集群中的工作节点（Worker Node）和Pod，在生产环境中，Master节点可以运行在多台节点实例上形成主备，提供Kubernetes集群的容错和高可用性。我们可以通过CLI或者UI页面中向Master节点输入参数控制Kubernetes集群。

Master节点是Kubernetes集群的管理中心，包含很多组件，这些组件管理Kubernetes集群各个方面，例如集群组件通信、工作负载调度和集群状态持久化。这些组件可以在集群内任意节点运行，但是为了方便会在一台实例上运行Master所有组件，并且不会在此实例上运行用户容器。

Kubernetes Master主节点包含组件如下：

- **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。例如：通过REST/kubectl 操作Kubernetes集群调用的就是Kube-apiserver。

- **etcd：**

etcd是一个一致的、高度可用的键值存储库，是kubernetes提供默认的存储系统，用于存储Kubernetes集群的状态和配置数据。

- **kube-scheduler：**

scheduler负责监视新创建、未指定运行节点的Pods并选择节点来让Pod在上面运行。如果没有合适的节点，则将Pod处于挂起的状态直到出现一个健康的Node节点。

- **kube-controller-manager：**

controller-manager 负责运行Kubernetes中的Controller控制器，这些Controller控制器包括：

> - 节点控制器（Node Controller）：负责在节点出现故障时进行通知和响应。
> - 任务控制器（Job Controller）：监测代表一次性任务的 Job 对象，然后创建 Pods 来运行这些任务直至完成。
> - 端点分片控制器（EndpointSlice controller）：填充端点分片（EndpointSlice）对象（以提供 Service 和 Pod 之间的链接）。

> - 服务账号控制器（ServiceAccount controller）：为新的命名空间创建默认的服务账号（ServiceAccount）。

- **cloud-controller-manager**

云控制器管理器（Cloud Controller Manager）嵌入了特定于云平台的控制逻辑，允许你将你的集群连接到云提供商的 API 之上， 并将与该云平台交互的组件同与你的集群交互的组件分离开来。cloud-controller-manager 仅运行特定于云平台的控制器。 因此如果你在自己的环境中运行 Kubernetes，或者在本地计算机中运行学习环境， 所部署的集群不需要有云控制器管理器。

3.2.2、Kubernetes Node节点

Kubernetes Node节点又称为工作节点（Worker Node），一个Kubernetes集群至少需要一个工作节点，但通常很多，工作节点也包含很多组件，用于运行以及维护Pod及service等信息, 管理volume(CVI)和网络(CNI)。在Kubernetes集群中可以动态的添加和删除节点来扩展和缩减集群。

工作节点Node上的组件如下：

- **Kubelet:**

Kubelet会在集群中每个Worker节点上运行，负责维护容器（Containers）的生命周期(创建pod，销毁pod)，确保Pod处于运行状态且健康。同时也负责Volume(CVI)和网络(CNI)的管理。Kubelet不会去管理**非** 来自于Kubernetes创建的容器。

- **kube-proxy:**

Kube-proxy是集群中每个Worker节点上运行的网络代理，管理IP转换和路由，确保每个Pod获得唯一的IP地址，维护网络规则，这些网络规则会允许从集群内部或外部的网络会话与Pod进行网络通信。

- **container Runtime:**

容器运行时(Container Runtime)负责运行容器的软件，为了运行容器每个Worker节点都有一个Container Runtime引擎，负责镜像管理以及Pod和容器的启动停止。

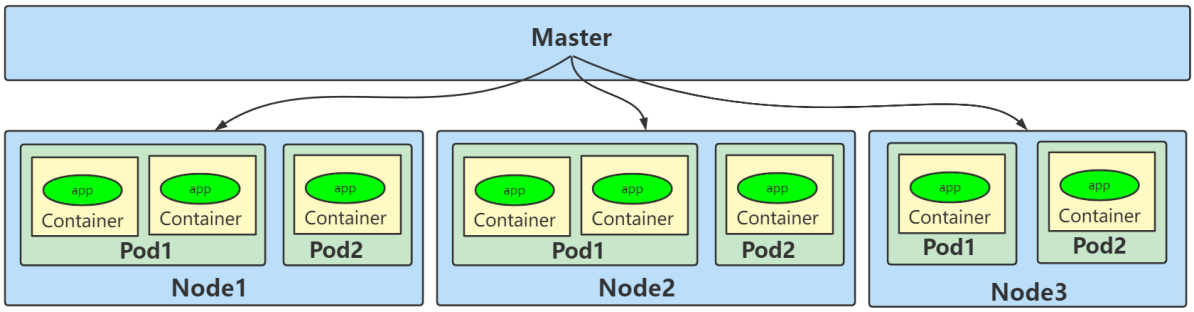
3.3、Kubernetes 核心概念

Kubernetes中有非常多的核心概念，下面主要介绍Kubernetes集群中常见的一些概念。

3.3.1、Pod

Pod 是可以在 Kubernetes 中创建和管理的、最小的可部署的计算单元，是Kubernetes调度的基本单位，Pod设计的理念是每个Pod都有一个唯一的IP。Pod就像豌豆荚一样，其中包含着一组（一个或多个）容器，这些容器共享存储、网络、文件系统以及怎样运行这些容器的声明。

Node & Pod & Container & 应用程序关系如下图所示：



3.3.2、Label

Label是附着到object上（例如Pod）的键值对。可以在创建object的时候指定，也可以在object创建后随时指定。Labels的值对系统本身并没有什么含义，只是对用户才有意义。

一个Label是一个key=value的键值对，其中key与value由用户自己指定。Label 可以附加到各种资源对象上，例如Node、Pod、Service、RC等，一个资源对象可以定义任意数量的Label，同一个Label也可以被添加到任意数量的资源对象上去，Label通常在资源对象定义时确定，也可以在对象创建后动态添加或者删除。

我们可以通过指定的资源对象捆绑一个或多个不同的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。例如：部署不同版本的应用到不同的环境中；或者监控和分析应用（日志记录、监控、告警）等。

一些常用abel示例如下所示:

- 版本标签："release" : "stable" , "release" : "canary"...
- 环境标签："environment" : "dev" , "environment" : "production"
- 架构标签："tier" : "frontend" , "tier" : "backend" , "tier" : "middleware"
- 分区标签："partition" : "customerA" , "partition" : "customerB"...
- 质量管控标签："track" : "daily" , "track" : "weekly"
- 应用标签："apptype" : "core" , "apptype" : "normal"...

Label相当于我们熟悉的“标签”，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，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。

3.3.3、NameSpace

Namespace 命名空间是对一组资源和对象的抽象集合，比如可以用来将系统内部的对象划分为不同的项目组或者用户组。常见的pod、service、replicaSet和deployment等都是属于某一个namespace的(默认是default)，而node, persistentVolumes等则不属于任何namespace。

当删除一个命名空间时会自动删除所有属于该namespace的资源，default和kube-system命名空间不可删除。

3.3.4、Controller控制器

在 Kubernetes 中，Contorller用于管理和运行Pod的对象，控制器通过监控集群的公共状态，并致力于将当前状态转变为期望的状态。一个Controller控制器至少追踪一种类型的 Kubernetes 资源。这些对象有一个代表期望状态的spec字段。该资源的控制器负责确保其当前状态接近期望状态。

不同类型的控制器实现的控制方式不一样，以下介绍常见的几种类型的控制器。

3.3.4.1、deployments控制器

deployments控制器用来部署无状态应用。基于容器部署的应用一般分为两种，无状态应用和有状态应用。

- 无状态应用：认为Pod都一样，没有顺序要求，随意进行扩展和伸缩。例如：nginx，请求本身包含了响应端为响应这一请求所需的全部信息。每一个请求都像首次执行一样，不会依赖之前的数据进行响应，不需要持久化数据，无状态应用的多个实例之间互不依赖，可以无序的部署、删除或伸缩。
- 有状态应用：每个pod都是独立运行，有唯一的网络表示符，持久化存储，有序。例如：mysql主从，主机名称固定，而且其扩容以及升级等操作也是按顺序进行。有状态应用前后请求有关联与依赖，需要持久化数据，有状态应运用的多个实例之间有依赖，不能相互替换。

在Kubernetes中，一般情况下我们不需要手动创建Pod实例，而是采用更高一层的抽象或定义来管理Pod，针对无状态类型的应用，Kubernetes使用Deloyment的Controller对象与之对应，其典型的应用场景包括：

> - 定义Deployment来创建Pod和ReplicaSet
> - 滚动升级和回滚应用
> - 扩容和缩容
> - 暂停和继续Deployment

3.3.4.2、ReplicaSet控制器

通过改变Pod副本数量实现Pod的扩容和缩容，一般Deployment里包含并使用了ReplicaSet。对于ReplicaSet而言，它希望pod保持预期数目、持久运行下去，除非用户明确删除，否则这些对象一直存在，它们针对的是耐久性任务，如web服务等。

3.3.4.3、statefulSet控制器

Deployments和ReplicaSets是为无状态服务设计的，StatefulSet则是为了有状态服务而设计，其应用场景包括：

- 唯一性: 每个Pod会被分配一个唯一序号。

- 顺序性: Pod启动,更新,销毁是按顺序进行。
- 稳定的网络标识: Pod主机名,DNS地址不会随着Pod被重新调度而发生变化。
- 稳定的持久化存储: Pod被重新调度后,仍然能挂载原有的PV,从而保证了数据的完整性和一致性。
- 固定链接方式：${pod name}.${service name}.${namespace}.svc.cluster.local

注意从写法上来看statefulSet与deployment几乎一致，就是类型不一样。

3.3.4.4、DaemonSet控制器

DaemonSet保证在每个Node上都运行一个相同Pod实例，常用来部署一些集群的日志、监控或者其他系统管理应用。DaemonSet使用注意以下几点：

- 当节点加入到Kubernetes集群中，pod会被（DaemonSet）调度到该节点上运行。
- 当节点从Kubernetes集群中被移除，被DaemonSet调度的pod会被移除。
- 如果删除一个Daemonset，所有跟这个DaemonSet相关的pods都会被删除。
- 如果一个DaemonSet的Pod被杀死、停止、或者崩溃，那么DaemonSet将会重新创建一个新的副本在这台计算节点上。
- DaemonSet一般应用于日志收集、监控采集、分布式存储守护进程等。

3.3.4.5、Job控制器

ReplicaSet针对的是耐久性任务，对于非耐久性任务，比如压缩文件，任务完成后，pod需要结束运行，不需要pod继续保持在系统中，这个时候就要用到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。

3.3.4.6、Cronjob控制器

Cronjob类似于Linux系统的crontab，在指定的时间周期运行相关的任务。

3.3.5、Service

使用kubernetes集群运行工作负载时，由于Pod经常处于用后即焚状态，Pod经常被重新生成，因此Pod对应的IP地址也会经常变化，导致无法直接访问Pod提供的服务，Kubernetes中使用了Service来解决这一问题，即在Pod前面使用Service对Pod进行代理，无论Pod怎样变化 ，只要有Label，就可以让Service能够联系上Pod，把PodIP地址添加到Service对应的端点列表（Endpoints）实现对Pod IP跟踪，进而实现通过Service访问Pod目的。

Service有以下几个注意点：

- 通过service为pod客户端提供访问pod方法，即可客户端访问pod入口
- 通过标签动态感知 pod IP地址变化等
- 防止pod失联
- 定义访问pod访问策略
- 通过label-selector相关联
- 通过Service实现Pod的负载均衡

Service 有如下四种类型：

- ClusterIP：默认，分配一个集群内部可以访问的虚拟IP。
- NodePort：在每个Node上分配一个端口作为外部访问入口。nodePort端口范围为:30000-32767
- LoadBalancer：工作在特定的Cloud Provider上，例如：Google Cloud，AWS，OpenStack。

- ExternalName：表示把集群外部的服务引入到集群内部中来，即实现了集群内部pod和集群外部的服务进行通信，适用于外部服务使用域名的方式，缺点是不能指定端口。

3.3.6、Volume 存储卷

默认情况下容器的数据是非持久化的，容器消亡以后数据也会跟着丢失。Docker容器提供了Volume机制以便将数据持久化存储。Kubernetes提供了更强大的Volume机制和插件，解决了容器数据持久化以及容器间共享数据的问题。

Kubernetes存储卷的生命周期与Pod绑定，容器挂掉后Kubelet再次重启容器时，Volume的数据依然还在，Pod删除时，Volume才会清理。数据是否丢失取决于具体的Volume类型，比如emptyDir的数据会丢失，而PV的数据则不会丢。

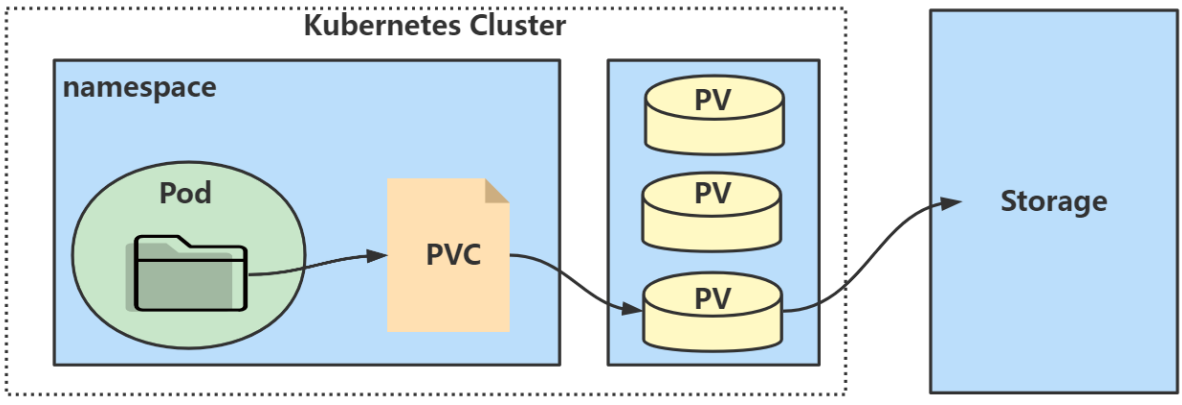
目前Kubernetes主要支持以下Volume类型：

- emptyDir：Pod存在，emptyDir就会存在，容器挂掉不会引起emptyDir目录下的数据丢失，但是pod被删除或者迁移，emptyDir也会被删除。
- hostPath：hostPath允许挂载Node上的文件系统到Pod里面去。
- NFS（Network File System）：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。
- glusterfs：同NFS一样是一种网络文件系统，Kubernetes可以将glusterfs挂载到Pod中，并进行永久保存。
- cephfs：一种分布式网络文件系统，可以挂载到Pod中，并进行永久保存。
- subpath：Pod的多个容器使用同一个Volume时，会经常用到。
- secret：密钥管理，可以将敏感信息进行加密之后保存并挂载到Pod中。
- persistentVolumeClaim：用于将持久化存储（PersistentVolume）挂载到Pod中。

除了以上几种Volume类型，Kubernetes还支持很多类型的Volume，详细可以参考：https://kubernetes.io/docs/concepts/storage/

3.3.7、PersistentVolume(PV) 持久化存储卷

kubernetes存储卷的分类太丰富了，每种类型都要写相应的接口与参数才行，这就让维护与管理难度加大，PersistentVolume(PV)是集群之中的一块网络存储，跟 Node 一样，也是集群的资源。PV是配置好的一段存储(可以是任意类型的存储卷)，将网络存储共享出来,配置定义成PV。PersistentVolume (PV)和PersistentVolumeClaim (PVC)提供了方便的持久化卷， PV提供网络存储资源，而PVC请求存储资源并将其挂载到Pod中，通过PVC用户不需要关心具体的volume实现细节,只需要关心使用需求。



3.3.8、ConfigMap

ConfigMap用于保存配置数据的键值对，可以用来保存单个属性，也可以用来保存配置文件，实现对容器中应用的配置管理，可以把ConfigMap看作是一个挂载到pod中的存储卷。ConfigMap跟secret很类似，但它可以更方便地处理不包含敏感信息的明文字符串。

3.3.9、Secret

Sercert-密钥解决了密码、token、密钥等敏感数据的配置问题，而不需要把这些敏感数据暴露到镜像或者Pod Spec中。

3.3.10、ServiceAccount

Service account是为了方便Pod里面的进程调用Kubernetes API或其他外部服务而设计的。Service Account为服务提供了一种方便的认证机制，但它不关心授权的问题。可以配合RBAC(Role Based Access Control)来为Service Account鉴权，通过定义Role、RoleBinding、ClusterRole、ClusterRoleBinding来对sa进行授权。

3.3.11、RBAC

1）Role（角色）：Role 定义了一组操作权限的集合，可以授予指定命名空间内的用户或用户组。Role 只能用于授予命名空间内资源的权限，如 Pod、Service、Deployment 等。

2）RoleBinding（角色绑定）：RoleBinding 将 Role 与用户或用户组之间进行绑定，指定了哪些用户或用户组具有特定的权限。一个 RoleBinding 可以将多个用户或用户组与一个 Role 相关联。

3）ClusterRole（集群角色）：ClusterRole 类似于 Role，但作用范围更广泛，可以授予集群范围内资源的权限，如节点、命名空间、PersistentVolume 等。ClusterRole 不限于单个命名空间。

4）ClusterRoleBinding（集群角色绑定）：ClusterRoleBinding 将 ClusterRole 与用户或用户组之间进行绑定，指定了哪些用户或用户组具有特定的集群级别权限。一个 ClusterRoleBinding 可以将多个用户或用户组与一个 ClusterRole 相关联。

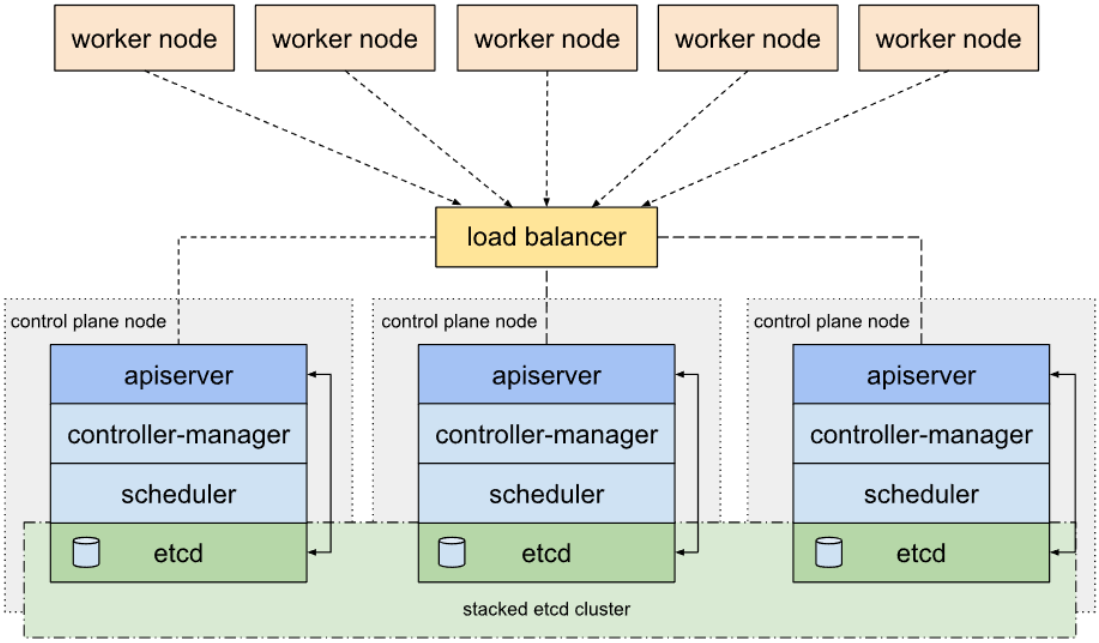
绑定关系：

- **Role 和 RoleBinding**：用于在特定命名空间内实现细粒度的授权和访问控制。通过创建自定义的 Role，可以指定特定用户或用户组对命名空间内的资源进行访问、创建、更新和删除等操作。RoleBinding 则将用户或用户组与特定的 Role 关联起来。
- **ClusterRole 和 ClusterRoleBinding**：用于在集群级别实现更宽泛的授权和访问控制。通过创建自定义的 ClusterRole，可以授予用户或用户组对整个集群范围内的资源进行操作的权限。ClusterRoleBinding 将用户或用户组与特定的 ClusterRole 关联起来。
- **ClusterRole 和 RoleBinding**：将集群级别的权限授予特定命名空间内的用户或用户组。通过在 RoleBinding 中指定 ClusterRole，并将其与命名空间内的用户或用户组进行关联，这些用户或用户组将同时具备命名空间内资源的访问权限和集群范围的权限。

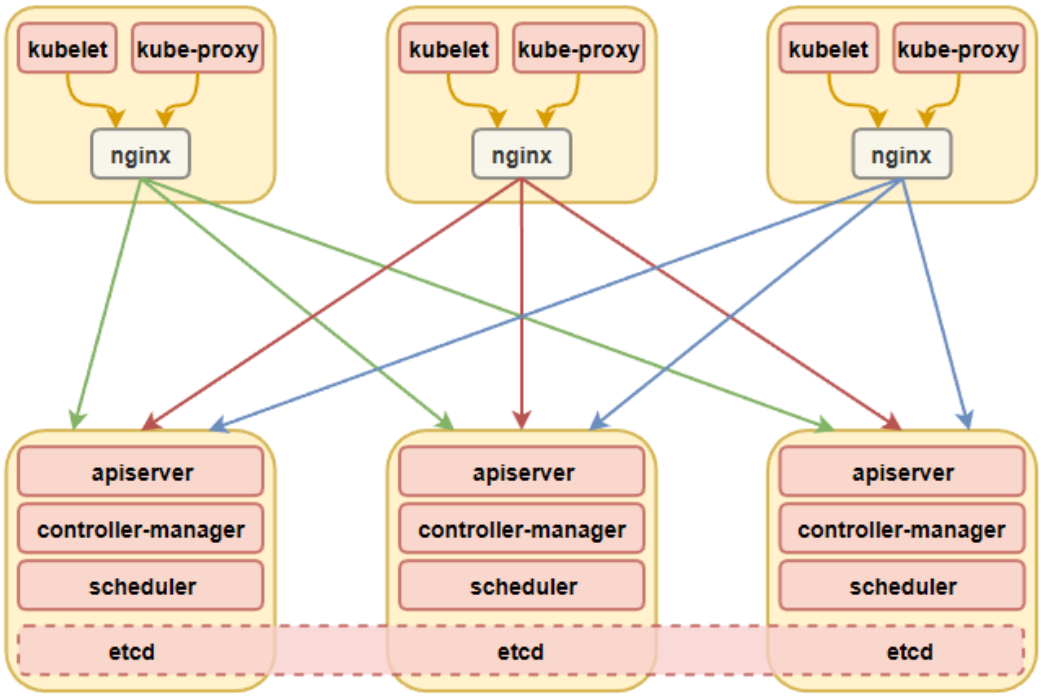
四、Kubernetes集群高可用

4.1、Apiserver高可用

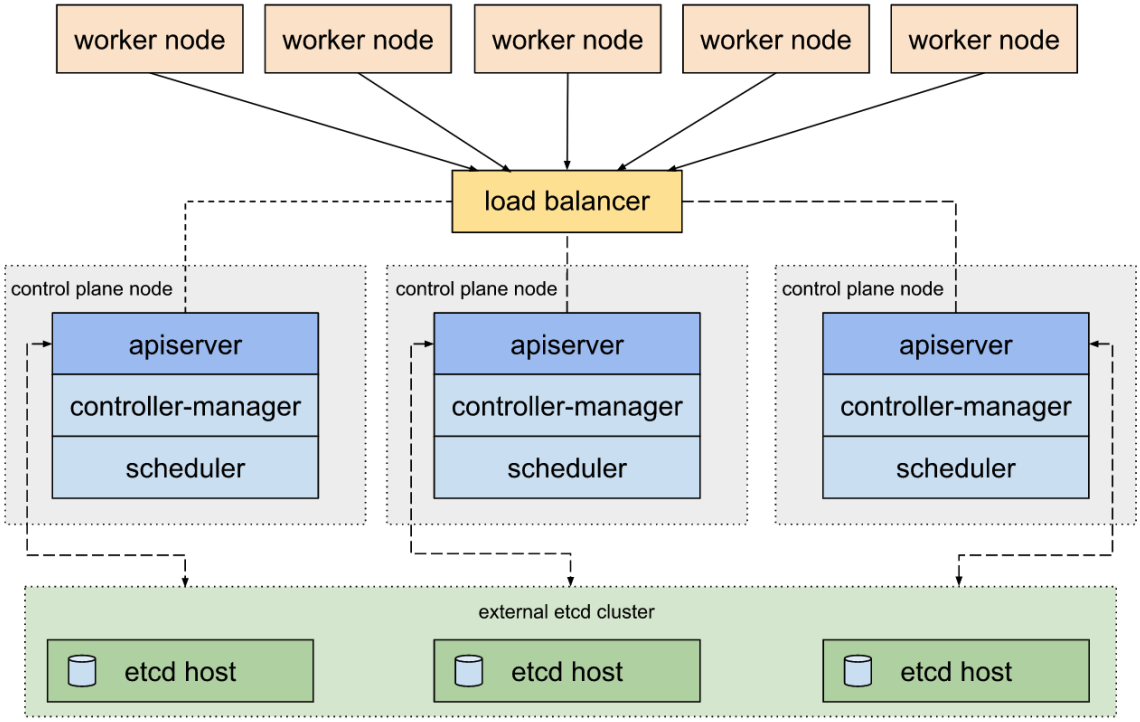
KEEPALIVED+LVS / 公有云LB



4.2、工作节点（worker）高可用



4.3、ETCD外部高可用



五、常用Kubernetes基础命令

1. kubectl apply -f <filename.yaml>：使用 YAML 或 JSON 文件创建或更新 Kubernetes 对象

2. kubectl get <resource>：获取 Kubernetes 资源信息，例如获取所有的 Pod 信息：kubectl get pods

3. kubectl describe <resource> <resource_name>：查看 Kubernetes 资源的详细信息，例如：kubectl describe pod nginx

4. kubectl delete <resource> <resource_name>：删除 Kubernetes 资源，例如：kubectl delete pod nginx

5. kubectl logs <resource_name>：查看 Pod 中容器的日志信息，例如：kubectl logs nginx -c 指定容器

6. kubectl exec -it <resource_name> bash：在 Pod 中的容器中执行命令，例如：kubectl exec -it nginx bash

7. kubectl port-forward <resource_name> <local_port>:<remote_port>：将 Pod 中的端口映射到本地端口，例如：kubectl port-forward nginx 8080:80

8. kubectl create secret <secret_type> <secret_name> --from-literal=<key>=<value>：创建密钥，例如：kubectl create secret generic db-credentials --from-literal=username=myuser --from-literal=password=mypassword

9. kubectl scale --replicas=<number_of_replicas> <resource> <resource_name>：扩容或缩容 Kubernetes 资源，例如：kubectl scale --replicas=3 deployment/nginx

10. kubectl rollout status <resource_name>：查看资源的滚动升级状态，例如：kubectl rollout status deployment/nginx

六、高阶些的Kubernetes命令

1. kubectl config get-contexts：查看当前 kubectl 的上下文信息

2. kubectl config use-context <context-name>：切换 kubectl 的上下文

3. kubectl create -f <filename.yaml>：通过 YAML 或 JSON 文件创建 Kubernetes 对象

4. kubectl apply -f <filename.yaml>：更新 Kubernetes 对象

5. kubectl edit <resource> <resource_name>：通过编辑器修改 Kubernetes 资源

6. kubectl api-resources：列出所有可用的 Kubernetes 资源类型

7. kubectl get <resource> <resource_name> -o <output_format>：以指定格式获取 Kubernetes 资源信息，例如：kubectl get pods nginx -o json

8. kubectl logs <resource_name> -c <container_name>：查看 Pod 中指定容器的日志信息

9. kubectl exec -it <resource_name> -c <container_name> -- <command>：在 Pod 中指定容器中执行命令，例如：kubectl exec -it nginx -c nginx-container -- bash

10. kubectl rollout history <resource> <resource_name>：查看资源的滚动升级历史，例如：kubectl rollout history deployment/nginx

11. kubectl rollout undo <resource> <resource_name>：撤销资源的滚动升级，例如：kubectl rollout undo deployment/nginx

12. kubectl label <resource> <resource_name> <key>=<value>：为资源添加标签，例如：kubectl label pods nginx app=web

13. kubectl delete <resource> <resource_name> --grace-period=<seconds>：删除 Kubernetes 资源，并设置指定的优雅期，例如：kubectl delete pod nginx --grace-period=30

14. kubectl get events：查看 Kubernetes 集群中发生的事件信息

15. kubectl taint <resource> <resource_name> <key>=<value>:<effect>：为节点添加污点，例如：kubectl taint node node1 key=value:NoSchedule

七、YAML

不用记录其写法，孰能生巧！多操作练习

```
##手动创建一个nginx的deployment
$ kubectl create deploy nginx --image=nginx --replicas=3 --dry-run -oyaml
```