Kubernetes 培训课件

柯普瑞企业 IT 学院

南京柯普瑞信息技术有限公司

2018 年 11 月

一、K8S 生态环境

介绍 Docker distros

2004 年，谷歌开始在集群中使用 borg；
2007 年，cgroups 提交到 linux 内核；
2008 年，redhat 的 libvirt 发布；
2008 年，LXC 发布；
2009 年，twitter 发布 mesos 项目；
2013 年，docker 出现；
2014 年，k8s 出现
2014 年，coreOS 稳定版本发布。

介绍 Fleet, Deis, Flynn

Flynn 和 Deis 是 Docker 的两个云计算微 PaaS 技术，它们都可以作为一个 PaaS 平台，但是它们不是旧式的 PaaS 范式，将 Docker 与其他混合装机在一起，它们寻求的是一种重新定义 PaaS 途径。 Flynn 和 Deis 已经重新定义了 micro-PaaS 概念，也就是说任何人都可以在自己的硬件上付出不太多的努力就可以运行它们。

介绍 Rancher

Rancher 是一个开源的企业级容器管理平台。通过 Rancher，企业再也不必自己使用一系列的开源软件去从头搭建容器服务平台。Rancher 提供了在生产环境中使用的管理 Docker 和 Kubernetes 的全栈化容器部署与管理平台。
Rancher 由以下四个部分组成：

基础设施编排

Rancher 可以使用任何公有云或者私有云的 Linux 主机资源。Linux 主机可以是虚拟机，也可以是物理机。Rancher 仅需要主机有 CPU，内存，本地磁盘和网络资源。从 Rancher 的角度来说，一台云厂商提供的云主机和一台自己的物理机是一样的。
Rancher 为运行容器化的应用实现了一层灵活的基础设施服务。Rancher 的基础设施服务包括网络， 存储， 负载均衡， DNS 和安全模块。Rancher 的基础设施服务也是通过容器部署的，所以同样 Rancher 的基础设施服务可以运行在任何 Linux 主机上。

容器编排与调度

很多用户都会选择使用容器编排调度框架来运行容器化应用。Rancher 包含了当前全部主流的编排调度引擎，例如 Docker Swarm，Kubernetes， 和 Mesos。同一个用户可以创建 Swarm

或者 Kubernetes 集群。并且可以使用原生的 Swarm 或者 Kubernetes 工具管理应用。
除了 Swarm，Kubernetes 和 Mesos 之外，Rancher 还支持自己的 Cattle 容器编排调度引擎。Cattle 被广泛用于编排 Rancher 自己的基础设施服务以及用于 Swarm 集群，Kubernetes 集群和 Mesos 集群的配置，管理与升级。

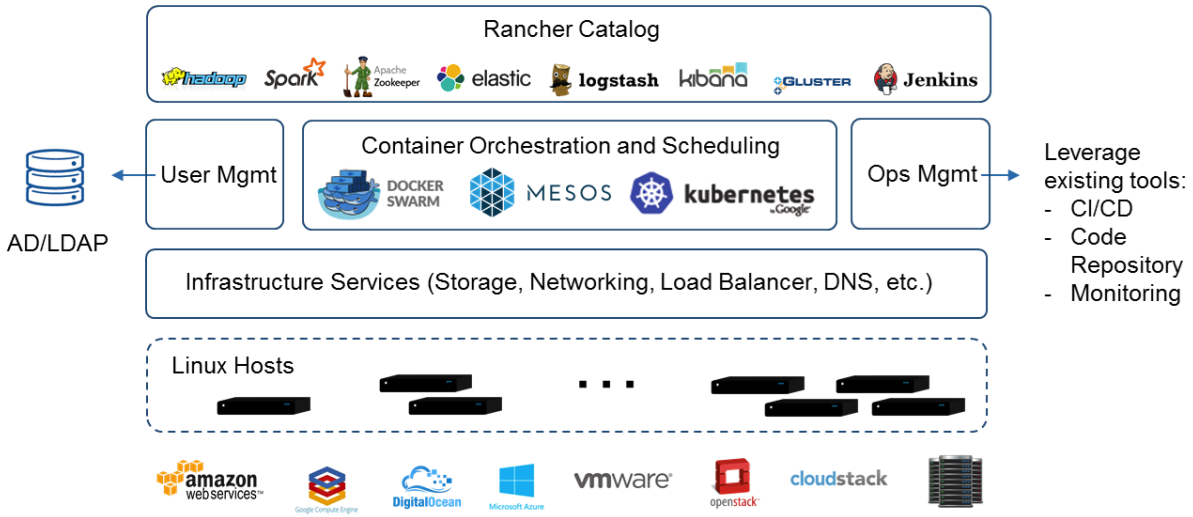
应用商店

Rancher 的用户可以在应用商店里一键部署由多个容器组成的应用。用户可以管理这个部署的应用，并且可以在这个应用有新的可用版本时进行自动化的升级。Rancher 提供了一个由 Rancher 社区维护的应用商店，其中包括了一系列的流行应用。Rancher 的用户也可以创建自己的私有应用商店。

企业级权限管理

Rancher 支持灵活的插件式的用户认证。支持 Active Directory，LDAP， Github 等 认证方式。Rancher 支持在环境级别的基于角色的访问控制 (RBAC)，可以通过角色来配置某个用户或者用户组对开发环境或者生产环境的访问权限。

下图展示了 Rancher 的主要组件和功能：



介绍 OpenShift

OpenShift 是红帽公司的一个开源容器应用平台，以 docker 作为容器引擎，以 k8s 模型编排、调度容器。在两者的基础上，红帽公司提供了一套更加完善的容器应用管理平台。可以部署在物理机，虚拟机，公有云，私有云等各种环境下。

功能（有很多基于 k8s 的概念）：

s2i：基于 git 或 svn 的代码快速生成应用容器镜像；

deployment：快熟部署容器镜像，配置过程可高度定制，保证容器运行持续可靠。

service：提供访问容器集群统一端口，实现负载均衡。

route：定义内外网络连通，简单连通外网，保障内网集群安全。

pv：持久存储卷，支持多种存储类型，解决容器持久化存储问题。

webconsole：提供 web 端管理页面，方便管理。

优点：

支持快速部署，实现敏捷开发。

提供动态伸缩功能，将过程简化至只需更改一个值。

管理资源，为容器分配合适的资源，提高资源利用率。

有对应的平台自动化运维工具，大大减少运维负担。

在大规模集群时提供方便高效的管理方法。

有完善的结构，部署以后能快速地测试应用。

丰富的接口，提供给各种插件与二次开发使用

上手难度：是基于 docker 和 k8s 的开源项目，有丰富的社区技术支持。还有关于 openshift 中文参考书。

总结来说，openshift 提供了一套完善的 docker 应用平台解决方案，如果公司需要部署并管理 docker 容器集群，我相信 openshift 是一款非常高效方便的方案。

二、配置管理

介绍 Puppet 和 Ansible

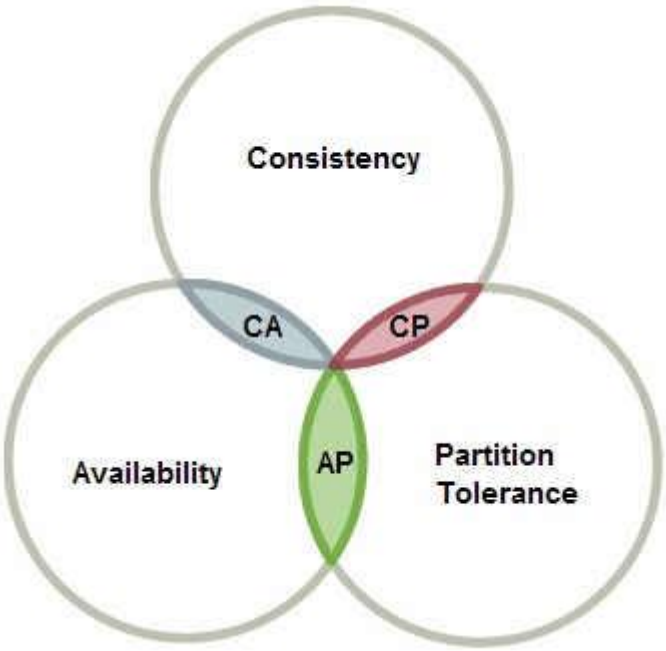
ansible 是新出现的自动化运维工具，基于 Python 开发，集合了众多运维工具（puppet、cfengine、chef、func、fabric）的优点，实现了批量系统配置、批量程序部署、批量运行命令等功能。
ansible 是基于模块工作的，本身没有批量部署的能力。真正具有批量部署的是 ansible 所运行的模块，ansible 只是提供一种框架。主要包括：

(1)、连接插件 connection plugins：负责和被监控端实现通信；
(2)、host inventory：指定操作的主机，是一个配置文件里面定义监控的主机；
(3)、各种模块核心模块、command 模块、自定义模块；
(4)、借助于插件完成记录日志邮件等功能；
(5)、playbook：剧本执行多个任务时，非必需可以让节点一次性运行多个任务。

三、服务发现

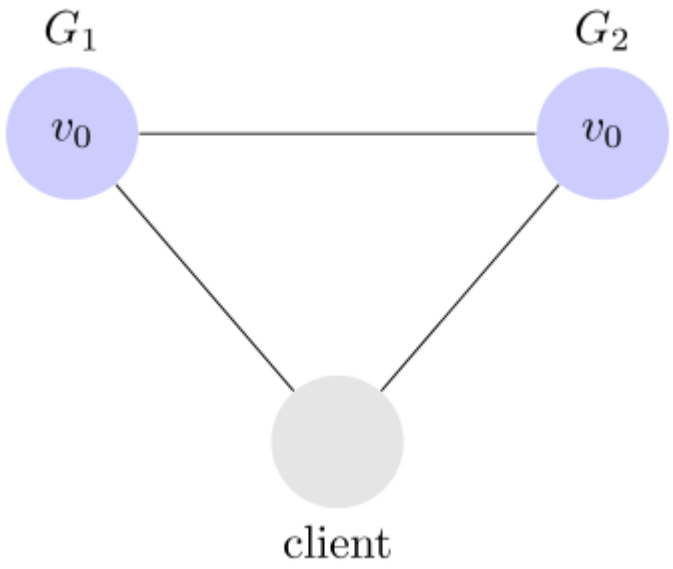
分布式系统（distributed system）正变得越来越重要，大型网站几乎都是分布式的。
分布式系统的最大难点，就是各个节点的状态如何同步。CAP 定理是这方面的基本定理，也是理解分布式系统的起点。

1、分布式系统的三个指标



1998 年，加州大学的计算机科学家 Eric Brewer 提出，分布式系统有三个指标。
Consistency
Availability
Partition tolerance
它们的第一个字母分别是 C、A、P。
Eric Brewer 说，这三个指标不可能同时做到。这个结论就叫做 CAP 定理。

2、Partition tolerance
先看 Partition tolerance，中文叫做"分区容错"。
大多数分布式系统都分布在多个子网络。每个子网络就叫做一个区（partition）。分区容错的意思是，区间通信可能失败。比如，一台服务器放在中国，另一台服务器放在美国，这就是两个区，它们之间可能无法通信。

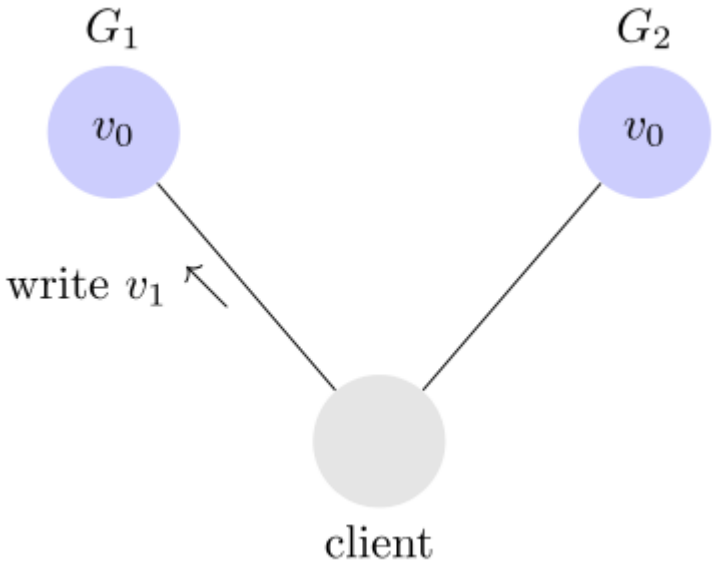


上图中，G1 和 G2 是两台跨区的服务器。G1 向 G2 发送一条消息，G2 可能无法收到。系统设计的时候，必须考虑到这种情况。

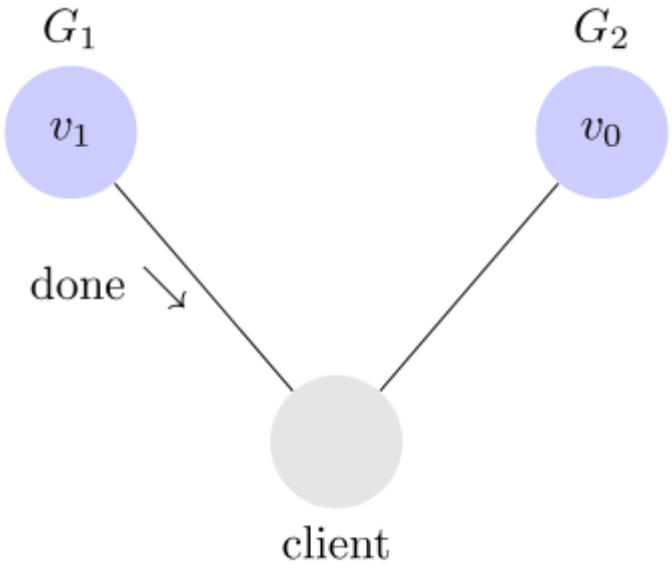
一般来说，分区容错无法避免，因此可以认为 CAP 的 P 总是成立。CAP 定理告诉我们，剩下的 C 和 A 无法同时做到。

3、Consistency

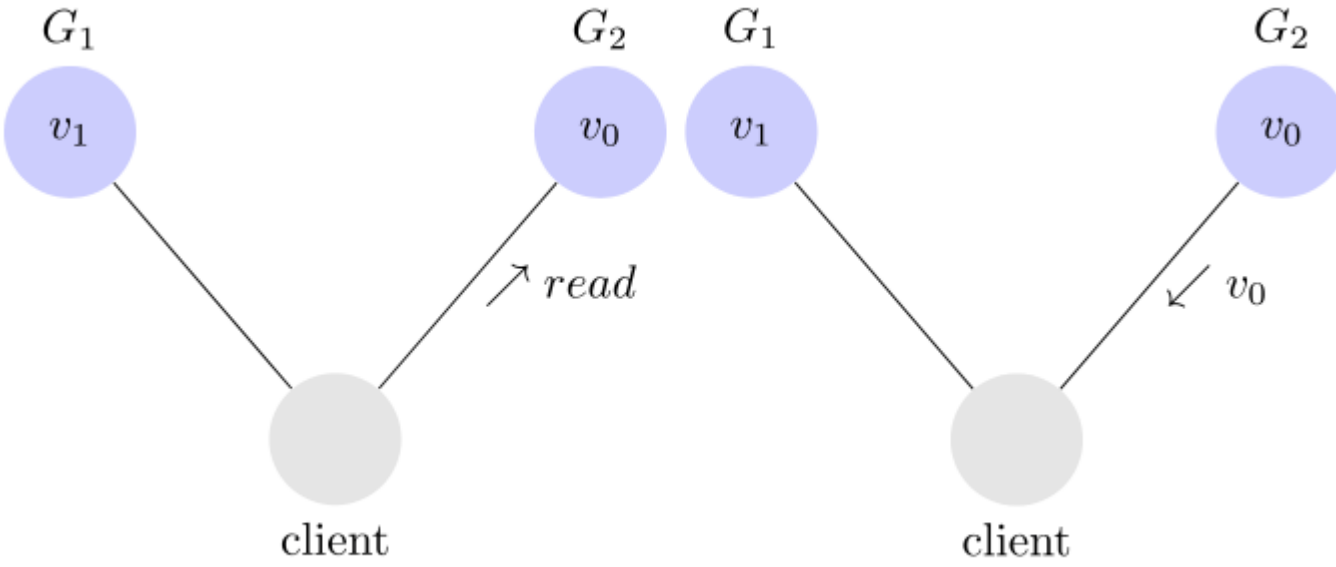
Consistency 中文叫做"一致性"。意思是，写操作之后的读操作，必须返回该值。举例来说，某条记录是 v0，用户向 G1 发起一个写操作，将其改为 v1。



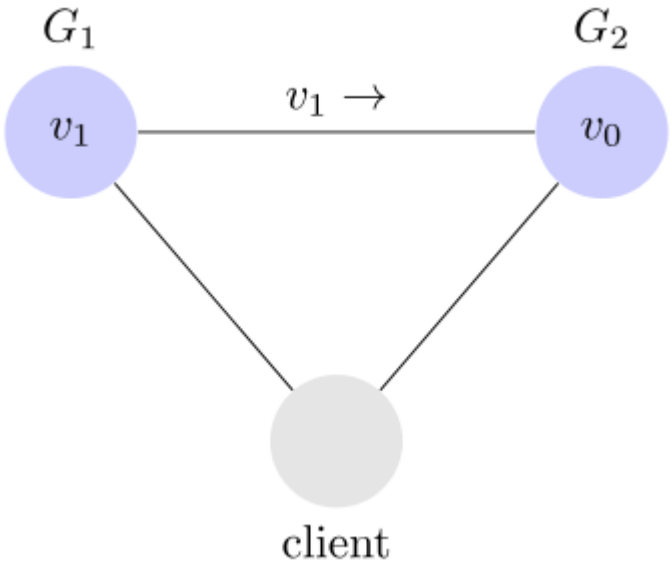
接下来，用户的读操作就会得到 v1。这就叫一致性。



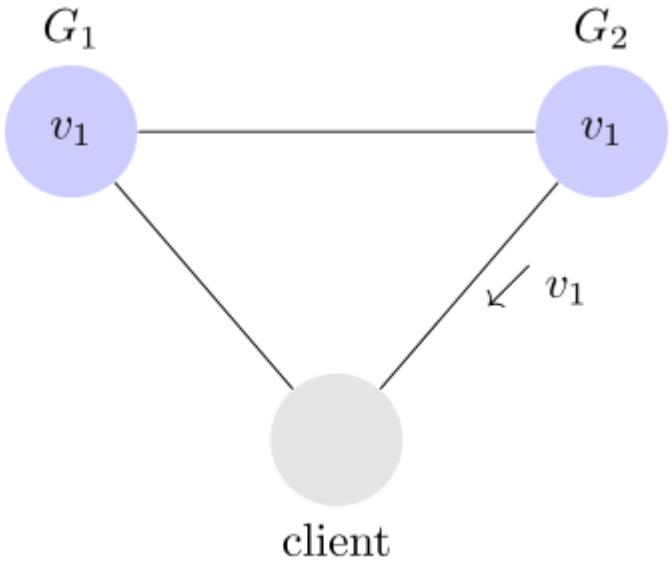
问题是，用户有可能向 G2 发起读操作，由于 G2 的值没有发生变化，因此返回的是 v0。G1 和 G2 读操作的结果不一致，这就不满足一致性了。



为了让 G2 也能变为 v1，就要在 G1 写操作的时候，让 G1 向 G2 发送一条消息，要求 G2 也改成 v1。



这样的话，用户向 G2 发起读操作，也能得到 v1。



4、Availability

Availability 中文叫做"可用性"，意思是只要收到用户的请求，服务器就必须给出回应。

用户可以选择向 G1 或 G2 发起读操作。不管是哪台服务器，只要收到请求，就必须告诉用户，到底是 v0 还是 v1，否则就不满足可用性。

5、Consistency 和 Availability 的矛盾

一致性和可用性，为什么不可能同时成立？答案很简单，因为可能通信失败（即出现分区容错）。

如果保证 G2 的一致性，那么 G1 必须在写操作时，锁定 G2 的读操作和写操作。只有数据同步后，才能重新开放读写。锁定期间，G2 不能读写，没有可用性不。

如果保证 G2 的可用性，那么势必不能锁定 G2，所以一致性不成立。

综上所述，G2 无法同时做到一致性和可用性。系统设计时只能选择一个目标。如果追求一

致性，那么无法保证所有节点的可用性；如果追求所有节点的可用性，那就没法做到一致性。

四、部署 Kubernetes 集群

介绍

docker 是一个容器引擎。docker 前面的祖先可以认为是 lxc。也是一个容器应用。

容器是一个进程，与其他进程完全隔离 cpu、内存、网络、权限、文件系统等。可以看作是一个独立的操作系统。实际上不是，只是一个进程。所以启动特别快，关闭特别快。并发量特别大。

Docker 的应用场景

Web 应用的自动化打包和发布。

自动化测试和持续集成、发布。

在服务型环境中部署和调整数据库或其他的后台应用。

从头编译或者扩展现有的 OpenShift 或 Cloud Foundry 平台来搭建自己的 PaaS 环境。

Docker 的优点

1、简化程序：

Docker 让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的 Linux 机器上，便可以实现虚拟化。Docker 改变了虚拟化的方式，使开发者可以直接将自己的成果放入 Docker 中进行管理。方便快捷已经是 Docker 的最大优势，过去需要用数天乃至数周的 任务，在 Docker 容器的处理下，只需要数秒就能完成。

2、避免选择恐惧症：

如果你有选择恐惧症，还是资深患者。Docker 帮你 打包你的纠结！比如 Docker 镜像；Docker 镜像中包含了运行环境和配置，所以 Docker 可以简化部署多种应用实例工作。比如 Web 应用、后台应用、数据库应用、大数据应用比如 Hadoop 集群、消息队列等等都可以打包成一个镜像部署。

3、节省开支：

一方面，云计算时代到来，使开发者不必为了追求效果而配置高额的硬件，Docker 改变了高性能必然高价格的思维定势。Docker 与云的结合，让云空间得到更充分的利用。不仅解决了硬件管理的问题，也改变了虚拟化的方式。

安装

检查内核版本

docker 要求 centos 的内核版本高于 3.10，因此必须查看内核版本。命令是 uname -r。

更新 yum 源

yum update

卸载已有的 docker

yum remove docker docker-engine docker-common \ docker-client docker-client-latest docker-latest docker-latest-logrotate \ docker-logrotate docker-selinux docker-engine-selinux

安装依赖包

yum install -y yum-utils device-mapper-persistent-data lvm2

设置 yum 源

yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

修改 yum 源内容为国内镜像

编辑文件 etc/yum.repos.d/docker-ce.repo 内容如下：

```
[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

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

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

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

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

保存成功后， 更新缓存 sudo yum makecache

查看仓库中的所有 docker 版本

```
yum list docker-ce --showduplicates | sort -r
```

安装 docker

```
yum install docker-ce
```

启动并加入开机启动

```
systemctl start docker
systemctl enable docker
```

验证安装是否成功(有 client 和 service 两部分表示 docker 安装启动都成功了)

```
docker version
```

```
[root@v1 ~]# docker version
Client:
 Version:           18.09.0
 API version:       1.39
 Go version:        go1.10.4
 Git commit:        4d60db4
 Built:             Wed Nov  7 00:48:22 2018
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          18.09.0
  API version:      1.39 (minimum version 1.12)
  Go version:       go1.10.4
  Git commit:       4d60db4
  Built:            Wed Nov  7 00:19:08 2018
  OS/Arch:          linux/amd64
  Experimental:     false
[root@v1 ~]#
```

运行 helloworld

docker run hello-world

```
[root@v1 ~]# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
d1725b59e92d: Pull complete
Digest: sha256:0add3ace90ecb4adbf7777e9aacf18357296e799f81cabc9fde470971e499788
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

配置国内镜像

编辑文件/etc/docker/daemon.json 内容如下：

{

"registry-mirrors": [

"http://aad0405c.m.daocloud.io"

],

"insecure-registries": []

}

这里的镜像地址来自于 DaoCloud。保存后，执行 systemctl restart docker 重启 docker 服务。

查看镜像的网址 https://hub.daocloud.io/

docker 常见操作

docker pull centos

会从远程拉一个镜像下来。这个过程特别慢。

docker images

可以查看本地的所有的镜像。

docker run centos /bin/sh "hello world"

会从原创/本地拉一个镜像，然后根据镜像创建一个容器，并运行。运行的时候，执行/bin/sh "hello world"命令。运行完毕后，关闭容器。再次执行该命令时，速度就很快。因为节省了 pull 时间。

docker run -i -t centos
命令是运行一个 centos 容器，参数-t 表示创建一个终端，参数-i 表示允许输入。

docker run -d centos /bin/sh -c "while true; do echo hello world; sleep 1; done"
命令在后台运行一个容器，容器内每秒钟输出一次 hello world。
该命令会产生一个容器 id

docker logs 容器 id
查看容器中的输出信息。

```
[root@localhost ~]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
9723a90a1628        centos              "/bin/sh -c 'while t欽   18 seconds ago      Up 17 seconds
95e93ab3efb9        centos              "/bin/sh"                9 minutes ago       Up 9 minutes
c56923b1b9e6        centos              "/bin/bash"              9 minutes ago       Up 9 minutes
[root@localhost ~]# clear
[root@localhost ~]# docker logs 9723a90a1628bb25bb590bb39af83bcae001672ac1f6562379a7e458b3d375cb
hello
hello
hello
hello
hello
hello
```

docker stop 容器 id
命令表示停止一个容器。

docker ps
查看正在运行的容器。

docker ps -a
查看所有的容器。

docker ps -l
查看最后一次运行的容器。

docker ps --help
查看 ps 的帮助。

docker stats
查看容器的资源占用情况。

docker run --name xxx
运行容器的时候，指定名称 xxx。

docker rm 容器 id
命令删除一个容器。

docker rmi -f 镜像 id
命令删除一个镜像。

运行 nginx 镜像

容器中的 nginx 存放页面的目录是/usr/share/nginx/html，我们可以指定外部的目录/usr/share/www，作为 nginx 的 web 页面所在的目录。

docker run -d -P -v /us/share/www:/usr/share/nginx/html nginx
这条命令把外部目录指定为容器中的 web 目录。

运行 web 镜像
docker pull training/webapp # 载入镜像
docker run -d -P training/webapp python app.py
参数说明：-d:让容器在后台运行。 -P:将容器内部使用的网络端口映射到我们使用的主机上。

```
[root@localhost yum.repos.d]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                     N
5b7e0b71efc0        training/webapp     "python app.py"     12 seconds ago      Up 11 seconds       0.0.0.0:32768->5000/tcp   r
[root@localhost yum.repos.d]#
```

查看后发现端口映射，指的是容器开启了 5000 端口，映射到本机是 32768 端口。

在本机运行 curl http://localhost:32768

```
[root@localhost yum.repos.d]# curl  http://localhost:32768
Hello world![root@localhost yum.repos.d]#
```

可以看到输出。

查看端口 docker port 5b7e0b71efc0
查看日志 docker logs -f 5b7e0b71efc0

命令：docker run -d -p 127.0.0.1:5001:5000 training/webapp python app.py 这里使用小写 p，不是大写 p。小写 p 指定端口映射，大写 p 自动端口映射。

运行 mysql 镜像
docker run －name mysql -P -e MYSQL_ROOT_PASSWORD=admin -d mysql
启动一个 mysql 容器，指定了数据库 root 用户的密码是 admin。

docker exec -it 6227910d6f6b bash
进入该容器。

运行 tomcat 镜像
docker run -d -P tomcat

五、Kubernetes 资源管理

安装前准备

所有节点，安装 centos7 操作系统。执行命令 cat /etc/redhat-release

所有节点，时间同步，执行命令 ntpdate -u cn.pool.ntp.org

在从节点安装 redhat-ca.crt，执行命令 yum -y install *rhsm*

minikube 安装

先下载安装 kubectl

```
KUBECTL_VERSION=1.19.1
curl -L
https://code.aliyun.com/khs1994-docker/kubectl-cn-mirror/raw/${KUBECTL_VERSION}
/kubectl-`uname -s`-`uname -m` > kubectl-`uname -s`-`uname -m`
chmod +x kubectl-`uname -s`-`uname -m`
./kubectl-`uname -s`-`uname -m` version
sudo mv kubectl-`uname -s`-`uname -m` /usr/local/bin/kubectl
```

再安装 minikube

```
curl -Lo minikube
http://kubernetes.oss-cn-hangzhou.aliyuncs.com/minikube/releases/v0.30.0/miniku
be-linux-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/
```

启动 minukube

```
minikube start --registry-mirror=https://docker.mirrors.ustc.edu.cn/
```

```
minikube config set WantKubectlDownloadMsg false
==========================================
Starting local Kubernetes v1.10.0 cluster...
Starting VM...
Getting VM IP address...
Moving files into cluster...
Downloading kubeadm v1.10.0
Downloading kubelet v1.10.0
Finished Downloading kubelet v1.10.0
Finished Downloading kubeadm v1.10.0
Setting up certs...
Connecting to cluster...
Setting up kubeconfig...
Starting cluster components...
Kubectl is now configured to use the cluster.
===================
```

关闭 minikube stop
查看日志 minikube logs
服务器重启后，集群会自动启动。

minikube 常见操作

常见命令 http://docs.kubernetes.org.cn/
查找镜像地址 https://dashboard.daocloud.io/packages/explore

run

docker pull nginx

kubectl run hello-minikube --image=nginx --port=80

```
[root@localhost ~]# kubectl run hello-minikube --image=nginx --port=80
deployment "hello-minikube" created
[root@localhost ~]# kubectl get deployment
NAME             DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
hello-minikube   1         1         1            1           5m
[root@localhost ~]# kubectl get deployment -o wide
NAME             DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE       CONTAINERS       IMAGES    SELECTOR
hello-minikube   1         1         1            1           5m        hello-minikube   nginx     run=hello-minikube
[root@localhost ~]#
```

```
[root@localhost ~]# kubectl get replicaset
NAME                       DESIRED   CURRENT   READY     AGE
hello-minikube-7cd97d7574   1         1         1         7m
[root@localhost ~]# kubectl get rs
NAME                       DESIRED   CURRENT   READY     AGE
hello-minikube-7cd97d7574   1         1         1         7m
[root@localhost ~]#
```

```
[root@localhost ~]# docker ps |grep hello
6d04ead4c05e        nginx                          "nginx -g 'daemon of鈥   7 minutes ago       Up 7 minutes
llo-minikube-7cd97d7574-7n54g_default_2bcf4a16-e8c3-11e8-a135-00505639a232_0
06af745c8e83        k8s.gcr.io/pause-amd64:3.1     "/pause"                 8 minutes ago       Up 7 minutes
e-7cd97d7574-7n54g_default_2bcf4a16-e8c3-11e8-a135-00505639a232_0
[root@localhost ~]#
```

```
[root@localhost ~]# kubectl describe pod
Name:           hello-minikube-7cd97d7574-7n54g
Namespace:      default
Node:           minikube/192.168.180.8
Start Time:     Thu, 15 Nov 2018 05:42:38 -0500
Labels:         pod-template-hash=3785383130
                run=hello-minikube
Annotations:    <none>
Status:         Running
IP:             172.17.0.4
Controlled By:  ReplicaSet/hello-minikube-7cd97d7574
Containers:
  hello-minikube:
    Container ID:   docker://6d04ead4c05e39588aa6d28b58f3dea09fcef3553f8dfbaca7b858c1d9152f64
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:ef9fbb3d773d4201c5dc781c75d8270326e113aff5317f8f6659e8a997011165
    Port:           80/TCP
    State:          Running
      Started:      Thu, 15 Nov 2018 05:42:46 -0500
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-sqwbt (ro)
Conditions:
  Type           Status
  Initialized    True
  Ready          True
  PodScheduled   True
Volumes:
  default-token-sqwbt:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-sqwbt
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason                 Age   From               Message
  ----    ------                 ----  ----               -------
  Normal  Scheduled              12m   default-scheduler  Successfully assigned hello-minikube-7cd97d7574-7n54g to minikube
  Normal  SuccessfulMountVolume  12m   kubelet, minikube  MountVolume.SetUp succeeded for volume "default-token-sqwbt"
  Normal  Pulling                12m   kubelet, minikube  pulling image "nginx"
  Normal  Pulled                 12m   kubelet, minikube  Successfully pulled image "nginx"
  Normal  Created                12m   kubelet, minikube  Created container
  Normal  Started                12m   kubelet, minikube  Started container
[root@localhost ~]#
```

```
[root@localhost ~]# kubectl get pod   hello-minikube-7cd97d7574-7n54g  -o yaml>aa
[root@localhost ~]#
```

expose

```
[root@localhost ~]# kubectl expose deployment  hello-minikube  --type=NodePort
service "hello-minikube" exposed
[root@localhost ~]# kubectl get services
NAME             TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
hello-minikube   NodePort    10.110.20.132   <none>        80:30459/TCP   10s
kubernetes       ClusterIP   10.96.0.1       <none>        443/TCP        4h
[root@localhost ~]#
```

将资源暴露为新的服务，供外部访问。

```
[root@localhost ~]# kubectl get services hello-minikube  -o yaml >bb
[root@localhost ~]#
```

kubectl get services

kubectl delete service xxxx

六、Kubernetes 服务

1、概述

为了适应快速的业务需求，微服务架构已经逐渐成为主流，微服务架构的应用需要有非常好的服务编排支持，k8s 中的核心要素 Service 便提供了一套简化的服务代理和发现机制，天然适应微服务架构，任何应用都可以非常轻易地运行在 k8s 中而无须对架构进行改动；k8s 分配给 Service 一个固定 IP，这是一个虚拟 IP(也称为 ClusterIP)，并不是一个真实存在的 IP，而是由 k8s 虚拟出来的。虚拟 IP 的范围通过 k8s API Server 的启动参数 --service-cluster-ip-range=19.254.0.0/16 配置;

虚拟 IP 属于 k8s 内部的虚拟网络，外部是寻址不到的。在 k8s 系统中，实际上是由 k8s Proxy 组件负责实现虚拟 IP 路由和转发的，所以 k8s Node 中都必须运行了 k8s Proxy，从而在容器覆盖网络之上又实现了 k8s 层级的虚拟转发网络。

服务代理：

在逻辑层面上，Service 被认为是真实应用的抽象，每一个 Service 关联着一系列的 Pod。在物理层面上，Service 有事真实应用的代理服务器，对外表现为一个单一访问入口，通过 k8s Proxy 转发请求到 Service 关联的 Pod。

Service 同样是根据 Label Selector 来刷选 Pod 进行关联的，实际上 k8s 在 Service 和 Pod 之间通过 Endpoint 衔接，Endpoints 同 Service 关联的 Pod；相对应，可以认为是 Service 的服务代理后端，k8s 会根据 Service 关联到 Pod 的 PodIP 信息组合成一个 Endpoints。

#kubectl get service my-nginx2 #kubectl get pod --selector app=nginx3 k8s 创建 Service 的同时，会自动创建跟 Service 同名的 Endpoints：4 #kubectl get endpoints my-nginx -o yaml5 #kubectl describe service my-nginx

Service 不仅可以代理 Pod，还可以代理任意其他后端，比如运行在 k8s 外部的服务。加速现在要使用一个 Service 代理外部 MySQL 服务，不用设置 Service 的 Label Selector。

Service 的定义文件: mysql-service.yaml:

apiVersion: v12 kind: Service3 metadata:4 name: mysql5 spec:6 ports:7 - port: 33068 targetPort: 33069 protocol: TCP

同时定义跟 Service 同名的 Endpoints，Endpoints 中设置了 MySQL 的 IP：192.168.3.180；

Endpoints 的定义文件 mysql-endpoints.yaml:

apiVersion: v1 2 kind: Endpoints 3 metadata: 4 name: mysql 5 subsets: 6 - addresses: 7 - ip: 192.168.39.175 8 ports: 9 - port: 330610 protocol: TCP

#kubectl create -f mysql-service.yaml -f mysql-endpoints.yaml

微服务化应用的每一个组件都以 Service 进行抽象，组件与组件之间只需要访问 Service 即可以互相通信，而无须感知组件的集群变化。

这就是服务发现；

#kubectl exec my-pod -- nslookup my-service.my-ns --namespace=default

#kubectl exec my-pod -- nslookup my-service --namespace=my-ns

2、Service 发布

k8s 提供了 NodePort Service、 LoadBalancer Service 和 Ingress 可以发布 Service；

NodePort Service

NodePort Service 是类型为 NodePort 的 Service，k8s 除了会分配给 NodePort Service 一个内部的虚拟 IP，另外会在每一个 Node 上暴露端口 NodePort，外部网络可以通过 [NodeIP]:[NodePort]访问到 Service。

LoadBalancer Service (需要底层云平台支持创建负载均衡器,比如 GCE)

LoadBalancer Service 是类型为 LoadBalancer 的 Service，它是建立在 NodePort Service 集群基础上的，k8s 会分配给 LoadBalancer；Service 一个内部的虚拟 IP，并且暴露 NodePort。除此之外，k8s 请求底层云平台创建一个负载均衡器，将每个 Node 作为后端，负载均衡器将转发请求到[NodeIP]:[NodePort]。

apiVersion: v1 2 kind: Service 3 metadata: 4 name: my-nginx 5 spec: 6 selector: 7 app: nginx 8 ports: 9 - name: http10 port: 8011 targetPort: 8012 protocol: TCP13 type: LoadBalancer

负载均衡器由底层云平台创建提供，会包含一个 LoadBalancerIP， 可以认为是 LoadBalancer Service 的外部 IP，查询 LoadBalancer Service：

#kubectl get svc my-nginx

Ingress

k8s 提供了一种 HTTP 方式的路由转发机制，称为 Ingress。Ingress 的实现需要两个组件支持， Ingress Controller 和 HTTP 代理服务器。HTTP 代理服务器将会转发外部的 HTTP 请求到 Service，而 Ingress Controller 则需要监控 k8s API，实时更新 HTTP 代理服务器的转发规则；

apiVersion: extensions/v1beta1 2 kind: Ingress 3 metadata: 4 name: my-ingress 5 spec: 6 rules: 7 - host: my.example.com 8 http: 9 paths:10 - path: /app11 backend:12 serviceName: my-app13 servicePort: 80

Ingress 定义中的.spec.rules 设置了转发规则，其中配置了一条规则，当 HTTP 请求的 host 为 my.example.com 且 path 为/app 时，转发到 Service my-app 的 80 端口；

#kubectl create -f my-ingress.yaml; kubectl get ingress my-ingress

NAME	RULE	BACKEND	ADDRESS
my-ingress	-		
	my.example.com		
	/app	my-app:80	

当 Ingress 创建成功后，需要 Ingress Controller 根据 Ingress 的配置，设置 HTTP 代理服务器的转发策略，外部通过 HTTP 代理服务就

可以访问到 Service；