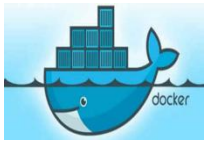
Docker 与微服务实战 2022 尚硅谷讲师:周阳



1. 基础篇(零基小白)

1.1. Docker 简介

1.1.1. 前提知识+课程定位+开场闲聊

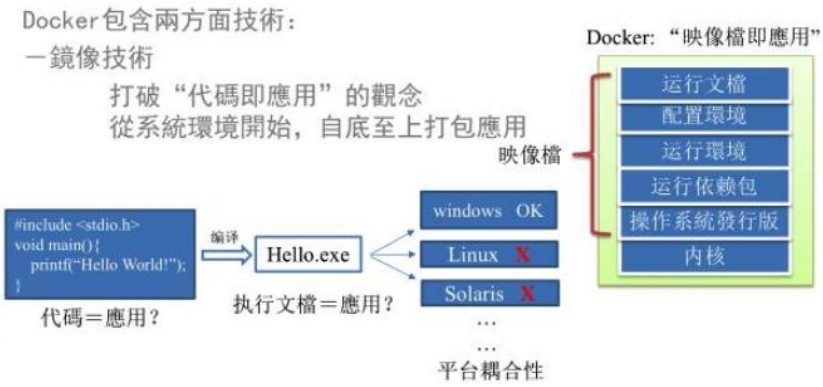
1.1.2. 是什么

- 问题：为什么会有 docker 出现

假定您在开发一个尚硅谷的谷粒商城，您使用的是一台笔记本电脑而且您的开发环境具有特定的配置。其他开发人员身处的环境配置也各有不同。您正在开发的应用依赖于您当前的配置且还要依赖于某些配置文件。此外，您的企业还拥有标准化的测试和生产环境，且具有自身的配置和一系列支持文件。您希望尽可能多在本地模拟这些环境而不产生重新创建服务器环境的开销。请问？
您要如何确保应用能够在这些环境中运行和通过质量检测？并且在部署过程中不出现令人头疼的版本、配置问题，也无需重新编写代码和进行故障修复？

答案就是使用容器。Docker 之所以发展如此迅速，也是因为它对此给出了一个标准化的解决方案-----系统平滑移植，容器虚拟化技术。

环境配置相当麻烦，换一台机器，就要重来一次，费力费时。很多人想到，能不能从根本上解决问题，软件可以带环境安装？也就是说，安装的时候，把原始环境一模一样地复制过来。开发人员利用 Docker 可以消除协作编码时“在我的机器上可正常工作”的问题。

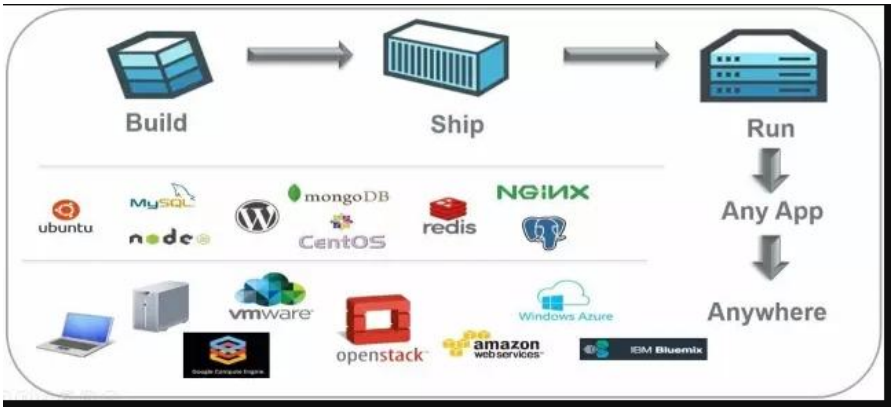


之前在服务器配置一个应用的运行环境，要安装各种软件，就拿尚硅谷电商项目的环境来说，Java/RabbitMQ/MySQL/JDBC 驱动包等。安装和配置这些东西有多麻烦就不说了，它还不能跨平台。假如我们是在 Windows 上安装的这些环境，到了 Linux 又得重新装。况且就算不跨操作系统，换另一台同样操作系统的服务器，要**移植**应用也是非常麻烦的。
传统上认为，软件编码开发/测试结束后，所产出的成果即是程序或是能够编译执行的二进制字节码等(java 为例)。而为了让这些程序可以顺利执行，开发团队也得准备完整的部署文件，让维运团队得以部署应用程式，开发需要清楚的告诉运维部署团队，用的全部配置文件+所有软件环境。不过，即便如此，仍然常常发生部署失败的状况。Docker 的出现使得 Docker 得以打破过去「程序即应用」的观念。透过镜像(images)将作业系统核心除外，运作应用程式所需要的系统环境，由下而上打包，达到应用程式跨平台间的无缝接轨运作。

- docker 理念

Docker 是基于 Go 语言实现的云开源项目。
Docker 的主要目标是“Build，Ship and Run Any App,Anywhere”，也就是通过对应用组件的封装、分发、部署、运行等生命周期的管

理，使用户的 APP（可以是一个 WEB 应用或数据库应用等等）及其运行环境能够做到“一次镜像，处处运行”。

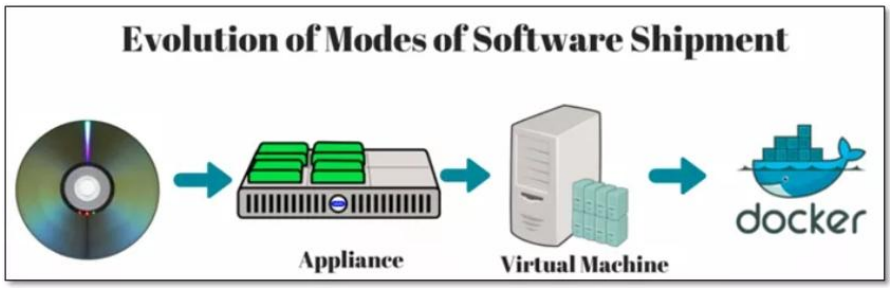


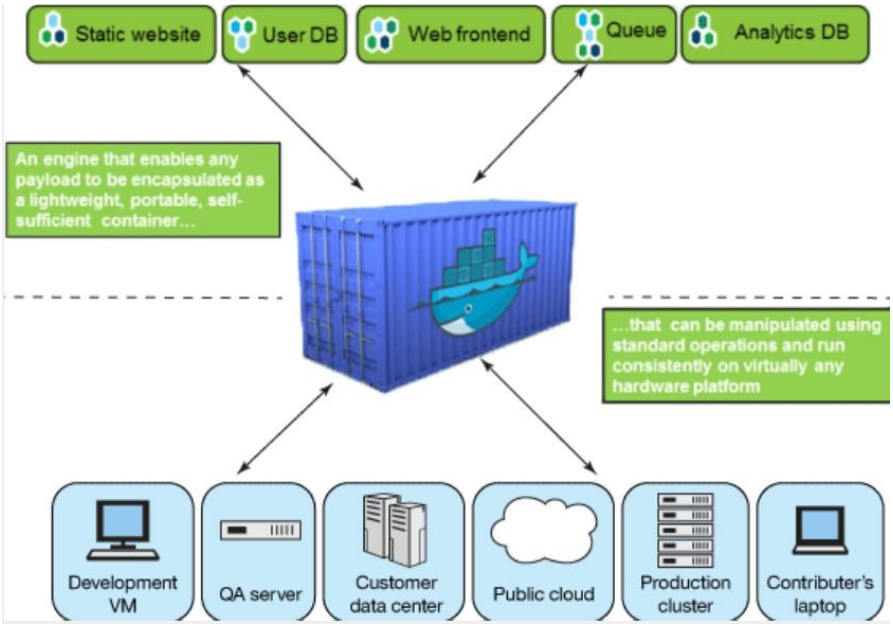
Linux 容器技术的出现就解决了这样一个问题，而 Docker 就是在它的基础上发展过来的。将应用打成镜像，通过镜像成为运行在 Docker 容器上面的实例，而 Docker 容器在任何操作系统上都是一致的，这就实现了跨平台、跨服务器。只需要一次配置好环境，换到别的机子上就可以一键部署好，大大简化了操作。

- 一句话
  - 解决了运行环境和配置问题的软件容器， 方便做持续集成并有助于整体发布的容器虚拟化技术。

1.1.3. 容器与虚拟机比较

- 容器发展简史



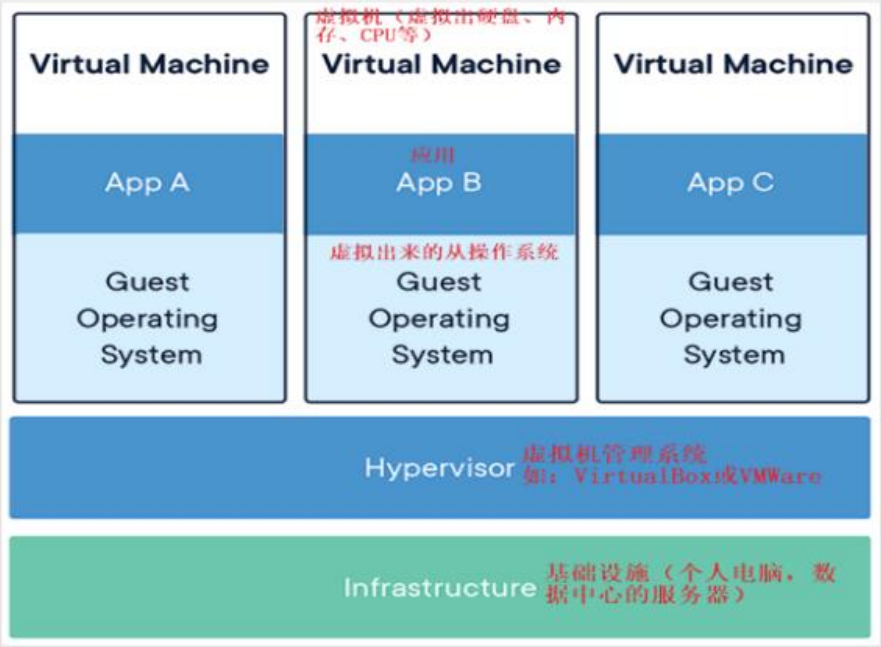


· 传统虚拟机技术

虚拟机（virtual machine）就是带环境安装的一种解决方案。
它可以在一种操作系统里面运行另一种操作系统，比如在 Windows10 系统里面运行 Linux 系统 CentOS7。应用程序对此毫无感知，因为虚拟机看上去跟真实系统一模一样，而对于底层系统来说，虚拟机就是一个普通文件，不需要了就删掉，对其他部分毫无影响。这类虚拟机完美的运行了另一套系统，能够使应用程序，操作系统和硬件三者之间的逻辑不变。

Win10	VMWare	Centos7	各种 cpu、内存网络额配置+各种软件	虚拟机实例

传统虚拟机技术基于安装在主操作系统上的虚拟机管理系统（如：VirtualBox 和 VMWare 等），创建虚拟机（虚拟出各种硬件），在虚拟机上安装从操作系统，在从操作系统中安装部署各种应用。



虚拟机的缺点：

1　资源占用多　　2　冗余步骤多　　3　启动慢

· 容器虚拟化技术

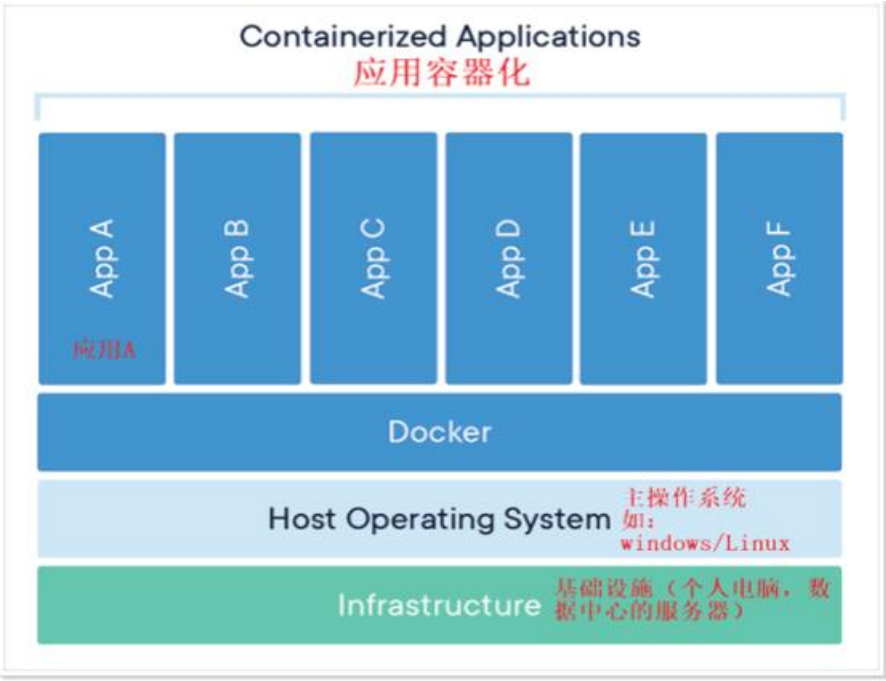
由于前面虚拟机存在某些缺点，Linux 发展出了另一种虚拟化技术：

Linux 容器(Linux Containers，缩写为 LXC)

Linux 容器是与系统其他部分隔离开的一系列进程，从另一个镜像运行，并由该镜像提供支持进程所需的全部文件。容器提供的镜像包含了应用的所有依赖项，因而在从开发到测试再到生产的整个过程中，它都具有可移植性和一致性。

Linux 容器不是模拟一个完整的操作系统而是对进程进行隔离。有了容器，就可以将软件运行所需的所有资源打包到一个隔离的容器中。容器与虚拟机不同，不需要捆绑一整套操作系统，只需要软件工作所需的库资源和设置。系

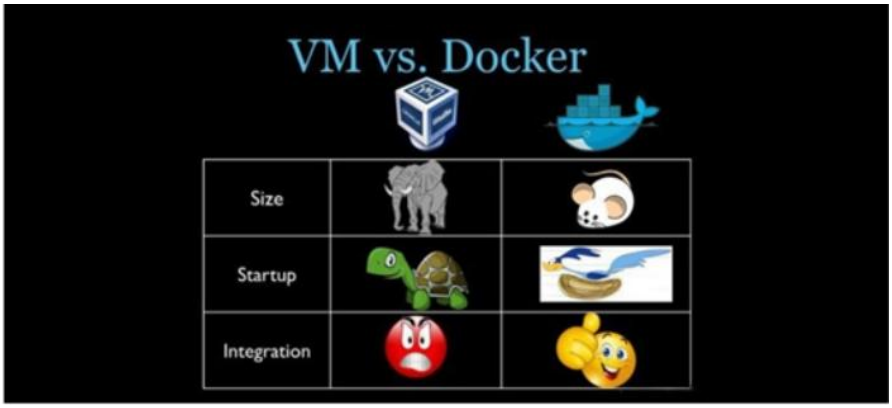
统因此而变得高效轻量并保证部署在任何环境中的软件都能始终如一地运行。



Docker 容器是在操作系统层面上实现虚拟化，直接复用本地主机的操作系统，而传统虚拟机则是在硬件层面实现虚拟化。与传统的虚拟机相比，Docker 优势体现为启动速度快、占用体积小。↵

- 对比

关系	对比 - 指向 底层原理



比较了 Docker 和传统虚拟化方式的不同之处：

*传统虚拟机技术是虚拟出一套硬件后，在其上运行一个完整操作系统，在该系统上再运行所需应用进程；
*容器内的应用进程直接运行于宿主的内核，容器内没有自己的内核且也没有进行硬件虚拟。因此容器要比传统虚拟机更为轻便。
* 每个容器之间互相隔离，每个容器有自己的文件系统，容器之间进程不会相互影响，能区分计算资源。

1.1.4. 能干嘛

- 技术职级变化
  - coder
  - programmer
  - software engineer
  - DevOps engineer
- 开发/运维（DevOps）新一代开发工程师
  - 一次构建、随处运行
  - 更快速的应用交付和部署

传统的应用开发完成后，需要提供一堆安装程序和配置说明文档，安装部署后需根据配置文档进行繁杂的配置才能正常运行。Docker 化之后只需要交付少量容器镜像文件，在正式生产环境加载镜像并运行即可，应用安装配置在镜像里已经内置好，大大节省部署配置和测试验证时间。

  - 更便捷的升级和扩缩容

随着微服务架构和 Docker 的发展，大量的应用会通过微服务方式架构，应用的开发构建将变成搭乐高积木一样，每个 Docker 容器将变成一块“积木”，应用的升级将变得非常容易。当现有的容器不足以支撑业务处理时，可通过镜像运行新的容器进行快速扩容，使应用系统的扩容从原先的天级变成分钟级甚至秒级。

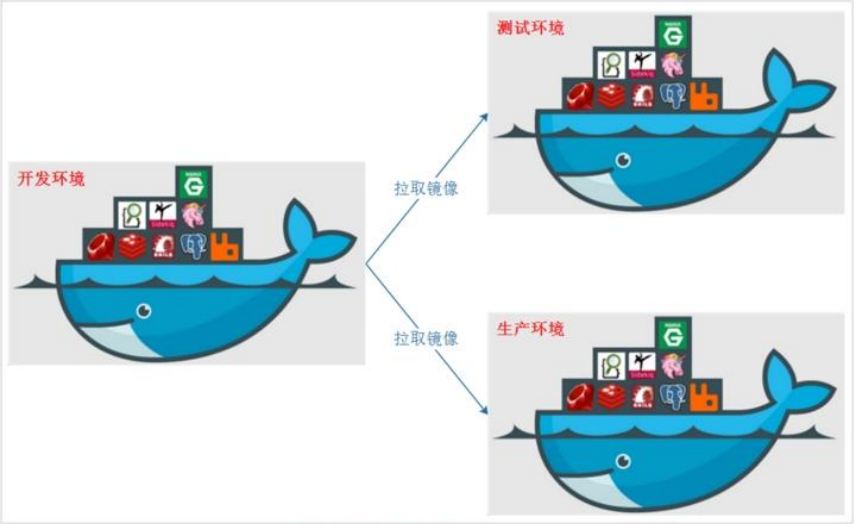
- 更简单的系统运维

应用容器化运行后，生产环境运行的应用可与开发、测试环境的应用高度一致，容器会将应用程序相关的环境和状态完全封装起来，不会因为底层基础架构和操作系统的不一致性给应用带来影响，产生新的 BUG。当出现程序异常时，也可以通过测试环境的相同容器进行快速定位和修复。

- 更高效的计算资源利用

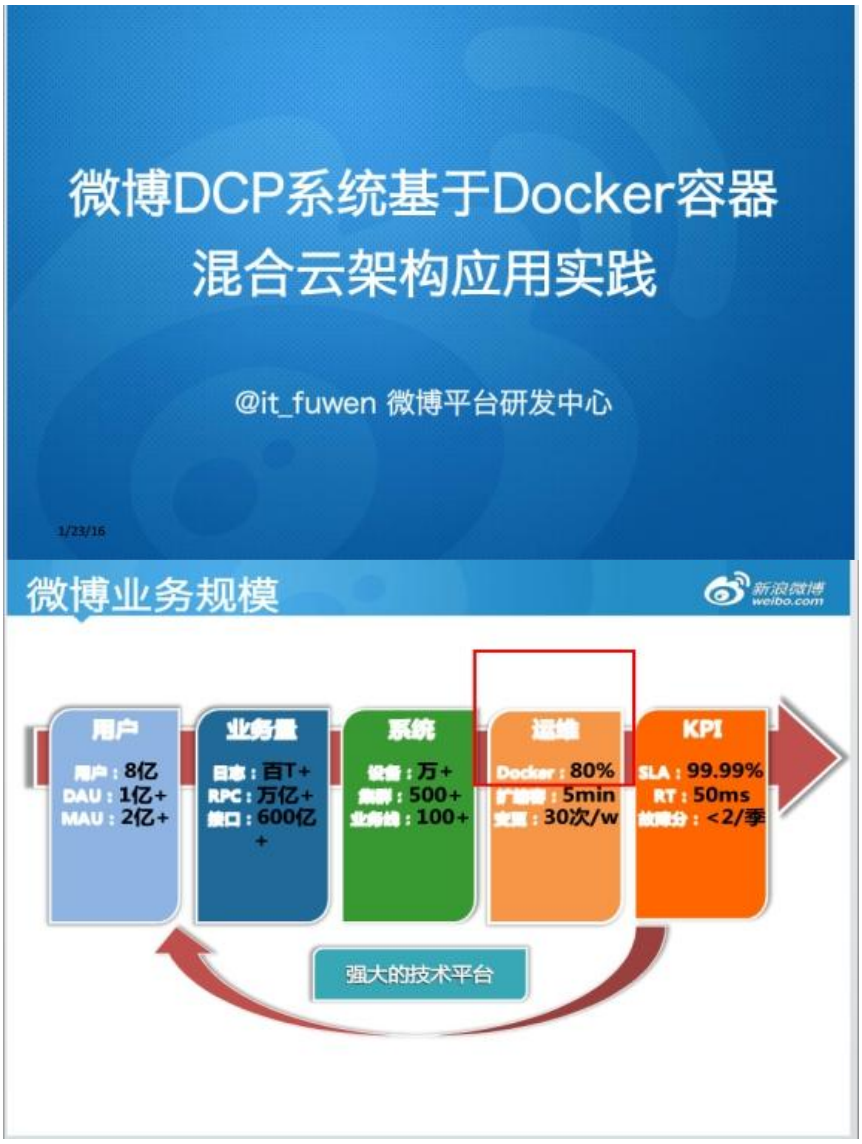
Docker 是内核级虚拟化，其不像传统的虚拟化技术一样需要额外的 Hypervisor 支持，所以在一台物理机上可以运行很多个容器实例，可大大提升物理服务器的 CPU 和内存的利用率。

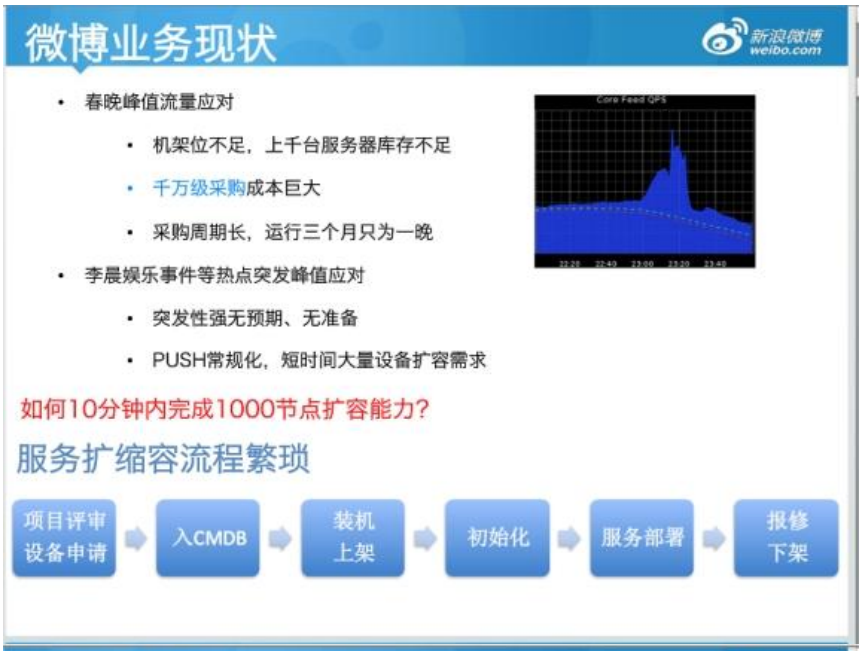
- · Docker 应用场景



Docker 借鉴了标准集装箱的概念。标准集装箱将货物运往世界各地，Docker 将这个模型运用到自己的设计中，唯一不同的是：集装箱运输货物，而 Docker 运输软件。

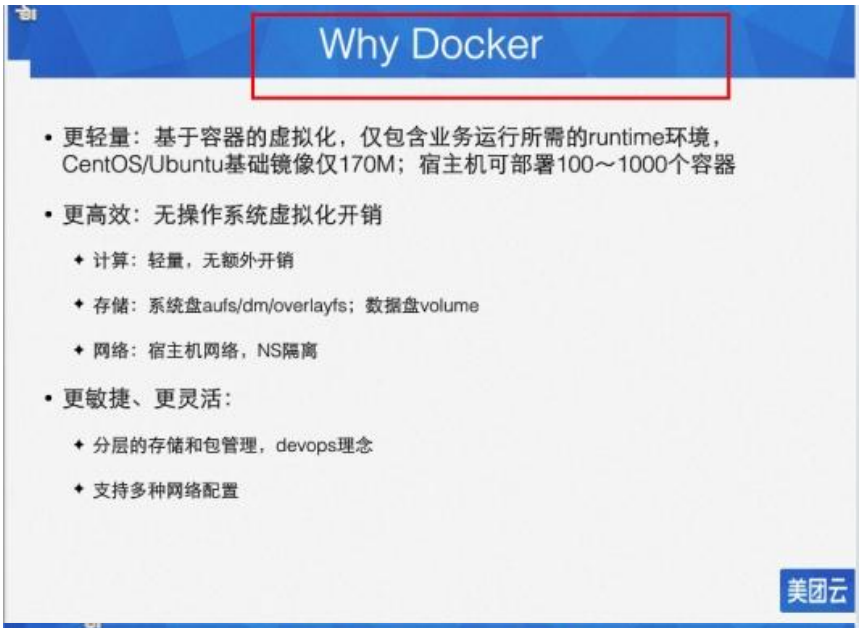
- 哪些企业在使用
  - 新浪

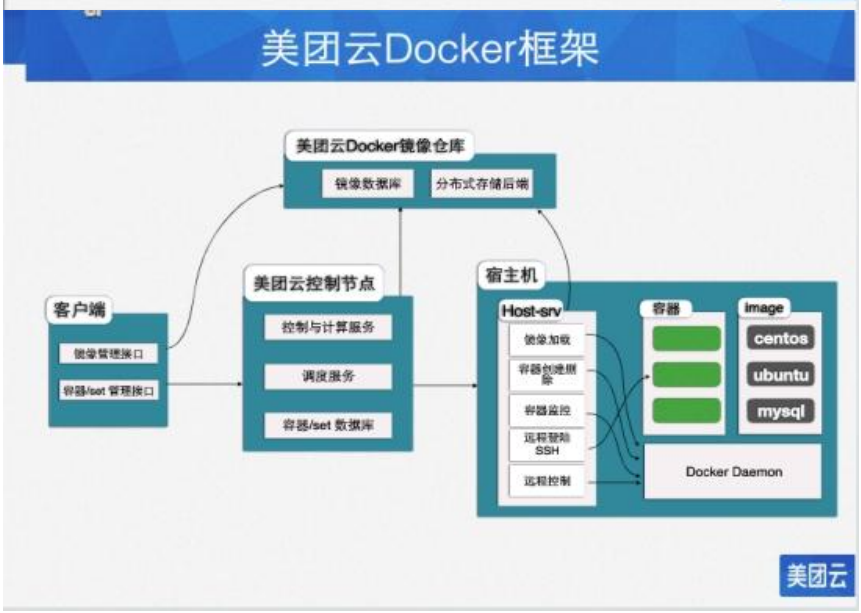

微博DCP系统基于Docker容器
混合云架构应用实践
@it_fuwen 微博平台研发中心
1/23/16
微博业务规模
新浪微博
weibo.com
用户
用户：8亿
DAU：1亿+
MAU：2亿+
业务量
日志：百T+
RPC：万亿+
接口：600亿+
系统
设备：万+
集群：500+
业务线：100+
运维
Docker：80%
扩缩容：5min
发布：30次/w
KPI
SLA：99.99%
RT：50ms
故障分：<2/季
强大的技术平台


微博业务现状
新浪微博 weibo.com
春晚峰值流量应对
机架位不足，上千台服务器库存不足
千万级采购成本巨大
采购周期长，运行三个月只为一晚
李晨娱乐事件等热点突发峰值应对
突发性强无预期、无准备
PUSH常规化，短时间大量设备扩容需求
Core Feed QPS
如何10分钟内完成1000节点扩容能力?
服务扩缩容流程繁琐
项目评审 设备申请
入CMDB
装机 上架
初始化
服务部署
报修 下架


业界趋势
新浪微博 weibo.com
混合云趋势：安全、可扩展性、成本...
AWS、阿里云等公有云平台趋于成熟
国外Zynga、Airbnb、Yelp等使用AWS进行部署
国内阿里云12306、高德、快的已部署，陌陌等部署中
12306借助阿里云解决饱受诟病的春节余票查询峰值问题
Docker、Mesos等容器新技术使大规模动态调度成为可能
京东618大促借助Docker为基础的弹性云解决峰值流量问题
amazon webservices
docker
JD.COM 京东
yelp

· 美团


Why Docker
• 更轻量：基于容器的虚拟化，仅包含业务运行所需的runtime环境，CentOS/Ubuntu基础镜像仅170M；宿主机可部署100～1000个容器
• 更高效：无操作系统虚拟化开销
✦ 计算：轻量，无额外开销
✦ 存储：系统盘aufs/dm/overlayfs；数据盘volume
✦ 网络：宿主机网络，NS隔离
• 更敏捷、更灵活：
✦ 分层的存储和包管理，devops理念
✦ 支持多种网络配置
美团云


美团云Docker框架
美团云Docker镜像仓库
镜像数据库
分布式存储后端
客户端
镜像管理接口
容器/set 管理接口
美团云控制节点
控制与计算服务
调度服务
容器/set 数据库
宿主机
Host-srv
镜像加载
容器创建删除
容器监控
远程登陆 SSH
远程控制
容器
image
centos
ubuntu
mysql
Docker Daemon
美团云

· 蘑菇街

- .....

1.1.5. 去哪下

- 官网
  - docker 官网：http://www.docker.com
- 仓库
  - Docker Hub 官网: https://hub.docker.com/

1.2. Docker 安装

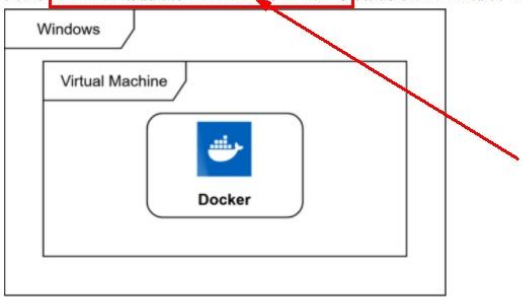
1.2.1. 前提说明

CentOS Docker 安装

Docker 并非是一个通用的容器工具，它依赖于已存在并运行的 Linux 内核环境。

Docker 实质上是在已经运行的 Linux 下制造了一个隔离的文件环境，因此它执行的效率几乎等同于所部署的 Linux 主机。

因此，Docker 必须部署在 Linux 内核的系统上。如果其他系统想部署 Docker 就必须安装一个虚拟 Linux 环境。



在 Windows 上部署 Docker 的方法都是先安装一个虚拟机，并在安装 Linux 系统的的虚拟机中运行 Docker。

前提条件

目前，CentOS 仅发行版本中的内核支持 Docker。Docker 运行在 CentOS 7 (64-bit) 上，
要求系统为 64 位、Linux 系统内核版本为 3.8 以上，这里选用 Centos7.x

查看自己的内核

uname 命令用于打印当前系统相关信息（内核版本号、硬件架构、主机名称和操作系统类型等）。

```
[root@zzyy ~]# cat /etc/redhat-release
CentOS Linux release 7.4.1708 (Core)
[root@zzyy ~]#
[root@zzyy ~]# uname -r
3.10.0-693.el7.x86_64
[root@zzyy ~]#
```

1.2.2. Docker 的基本组成

- 镜像(image)

Docker 镜像（Image）就是一个只读的模板。
镜像可以用来创建 Docker 容器，一个镜像可以创建很多容器。
它也相当于是一个 root 文件系统。比如官方镜像 centos:7 就包含了完整的一套 centos:7 最小系统的 root 文件系统。
相当于容器的“源代码”，docker 镜像文件类似于 Java 的类模板，而 docker 容器实例类似于 java 中 new 出来的实例对象。

容器与镜像的关系类似于面向对象编程中的对象与类。

Docker	面向对象
容器	对象
镜像	类

- 容器(container)

1 从面向对象角度
Docker 利用容器（Container）独立运行的一个或一组应用，应用程序或服务运行在容器里面，容器就类似于一个虚拟化的运行环境，容器是用镜像创建的运行实例。就像是 Java 中的类和实例对象一样，镜像是静态的定义，容器是镜像运行时的实体。容器为镜像提供了一个标准的和隔离的运行环境，它可以被启动、开始、停止、删除。每个容器都是相互隔离的、保证安全的平台

2 从镜像容器角度

可以把容器看做是一个简易版的 Linux 环境
（包括 root 用户权限、进程空间、用户空间和网络空间等）和运行在其中的应用程序。

- 仓库(repository)

仓库（Repository）是集中存放镜像文件的场所。

类似于
Maven 仓库，存放各种 jar 包的地方；
github 仓库，存放各种 git 项目的地方；
Docker 公司提供的官方 registry 被称为 Docker Hub，存放各种镜像模板的地方。

仓库分为公开仓库（Public）和私有仓库（Private）两种形式。
最大的公开仓库是 Docker Hub(https://hub.docker.com/)，
存放了数量庞大的镜像供用户下载。国内的公开仓库包括阿里云 、网易云等

- 小总结

需要正确的理解仓库/镜像/容器这几个概念:
Docker 本身是一个容器运行载体或称之为管理引擎。我们把应用程序和配置依赖打包好形成一个可交付的运行环境，这个打包好的运行环境就是 image 镜像文件。只有通过这个镜像文件才能生成 Docker 容器实例(类似 Java 中 new 出来一个对象)。

image 文件可以看作是容器的模板。Docker 根据 image 文件生成容器的实例。同一个 image 文件，可以生成多个同时运行的容器实例。

镜像文件

* image 文件生成的容器实例，本身也是一个文件，称为镜像文件。

容器实例

* 一个容器运行一种服务，当我们需要的时候，就可以通过 docker 客户端创建一个对应的运行实例，也就是我们的容器

仓库

* 就是放一堆镜像的地方，我们可以把镜像发布到仓库中，需要的时候再从仓库中拉下来就可以了。

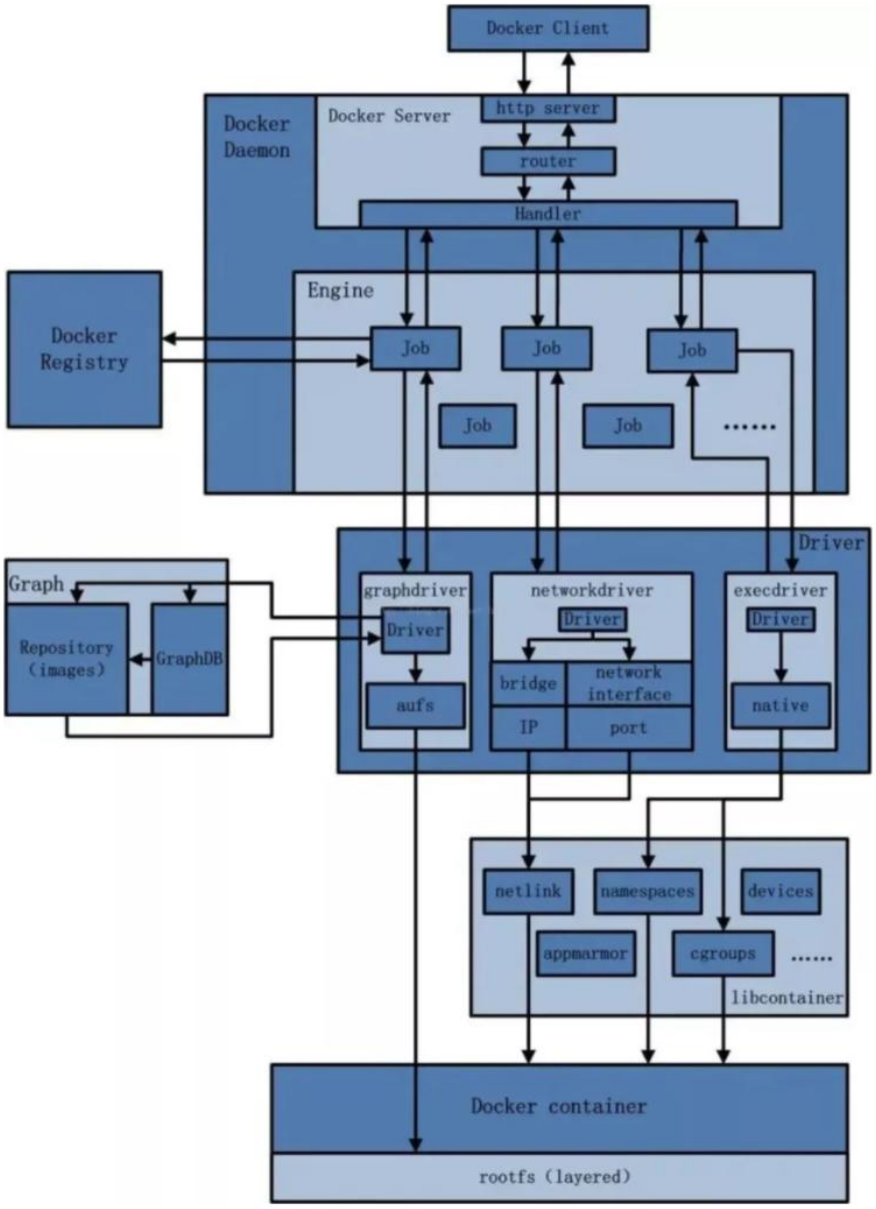
1.2.3. Docker 平台架构图解(架构版)

- 首次懵逼正常，后续深入，先有大概轮廓，混个眼熟
- 整体架构及底层通信原理简述

Docker 是一个 C/S 模式的架构，后端是一个松耦合架构，众多模块各司其职。

Docker 运行的基本流程为：

1 用户是使用 Docker Client 与 Docker Daemon 建立通信，并发送请求给后者。
2 Docker Daemon 作为 Docker 架构中的主体部分，首先提供 Docker Server 的功能使其可以接受 Docker Client 的请求。
3 Docker Engine 执行 Docker 内部的一系列工作，每一项工作都是以一个 Job 的形式的存在。
4 Job 的运行过程中，当需要容器镜像时，则从 Docker Registry 中下载镜像，并通过镜像管理驱动 Graph driver将下载镜像以Graph的形式存储。
5 当需要为 Docker 创建网络环境时，通过网络管理驱动 Network driver 创建并配置 Docker 容器网络环境。
6 当需要限制 Docker 容器运行资源或执行用户指令等操作时，则通过 Exec driver 来完成。
7 Libcontainer是一项独立的容器管理包，Network driver以及Exec driver都是通过Libcontainer来实现具体对容器进行的操作。



1.2.4. 安装步骤

- CentOS7 安装 Docker
  - https://docs.docker.com/engine/install/centos/
  - 安装步骤
  - 确定你是 CentOS7 及以上版本
  - cat /etc/redhat-release
  - 卸载旧版本

https://docs.docker.com/engine/install/centos/

The `centos-extras` repository must be enabled. This repository is enabled by default, but if you have disabled it, you need to re-enable it.

The `overlay2` storage driver is recommended.

Uninstall old versions

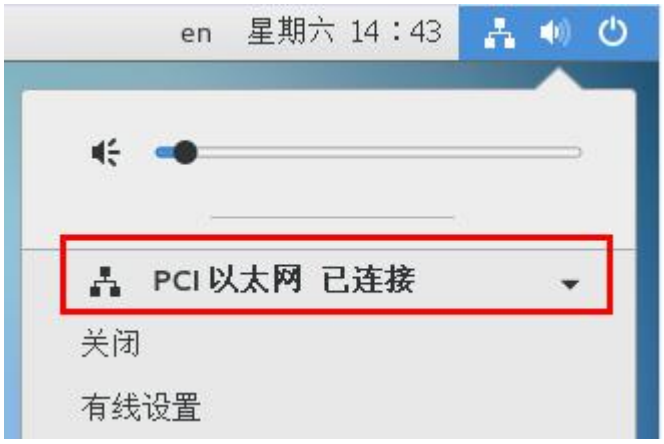
Older versions of Docker were called `docker` or `docker-engine` . If these are installed, uninstall them, along with associated dependencies.

```
$ sudo yum remove docker \
                  docker-client \
                  docker-client-latest \
                  docker-common \
                  docker-latest \
                  docker-latest-logrotate \
                  docker-logrotate \
                  docker-engine
```

It's OK if `yum` reports that none of these packages are installed.

The contents of `/var/lib/docker/` , including images, containers, volumes, and networks, are preserved. The Docker Engine package is now called `docker-ce` .

- yum 安装 gcc 相关



- CentOS7 能上外网

- yum -y install gcc
- yum -y install gcc-c++
- 安装需要的软件包
- 官网要求

You can install Docker Engine in different ways, depending on your needs:

- Most users set up Docker's repositories and install from them, for ease of installation and upgrade tasks. This is the recommended approach.
- Some users download the RPM package and install it manually and manage upgrades completely manually. This is useful in situations such as installing Docker on air-gapped systems with no access to the internet.
- In testing and development environments, some users choose to use automated convenience scripts to install Docker.

Install using the repository

Before you install Docker Engine for the first time on a new host machine, you need to set up the Docker repository. Afterward, you can install and update Docker from the repository.

Set up the repository

Install the `yum-utils` package (which provides the `yum-config-manager` utility) and set up the **stable** repository.

```
$ sudo yum install -y yum-utils

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

- 执行命令
- yum install -y yum-utils
- 设置 stable 镜像仓库
- 大坑

单图标	CustomIcon-663735520

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

2. Use the following command to set up the **stable** repository.

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

报错：

1 [Errno 14] curl#35 - TCP connection reset by peer

2 [Errno 12] curl#35 - Timeout

- 官网要求

2. Use the following command to set up the **stable** repository.

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

报错：

1 [Errno 14] curl#35 - TCP connection reset by peer

2 [Errno 12] curl#35 - Timeout

- 推荐

单图标	✔ CustomIcon--1664269521

- yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
- 我们自己

```
[root@zzyy /]# yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
已加载插件：fastestmirror, langpacks
adding repo from: http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
grabbing file http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo to /etc/yum.repos.d/docker-ce.repo
repo saved to /etc/yum.repos.d/docker-ce.repo
[root@zzyy /]#
```

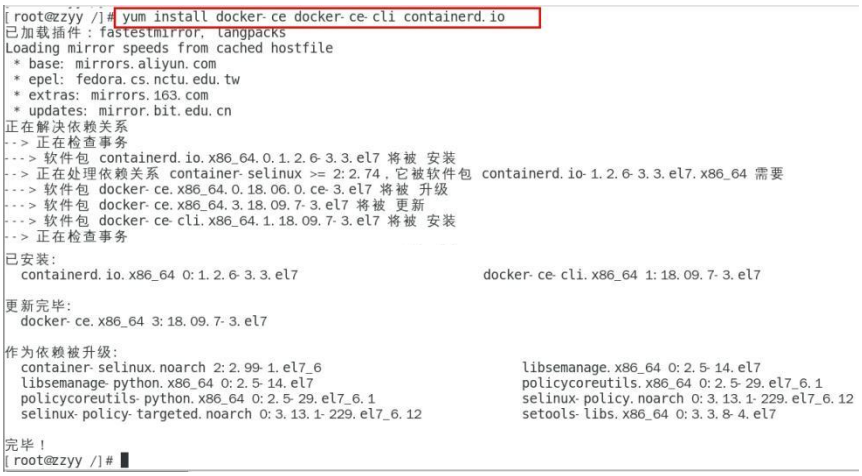
- 更新 yum 软件包索引
- yum makecache fast
- 安装 DOCKER CE
- yum -y install docker-ce docker-ce-cli containerd.io
- 官网要求

INSTALL DOCKER CE

1. Install the *latest version* of Docker CE and containerd, or go to the next step to install a specific version:

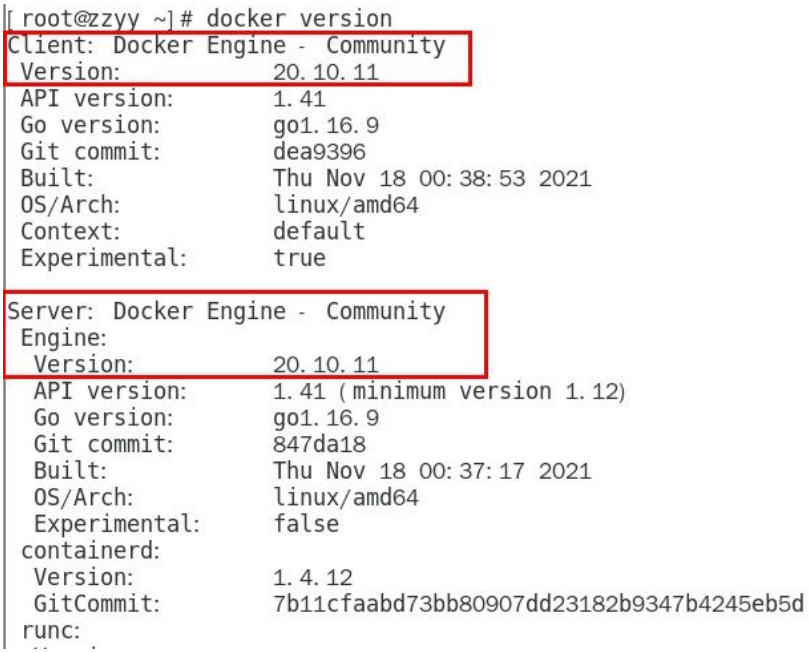
```
$ sudo yum install docker-ce docker-ce-cli containerd.io
```

- 执行结果



- 启动 docker
- systemctl start docker
- 测试
- docker version
- 本次安装时间 2021.11

本次安装时间 2021.11



- docker run hello-world

- 卸载

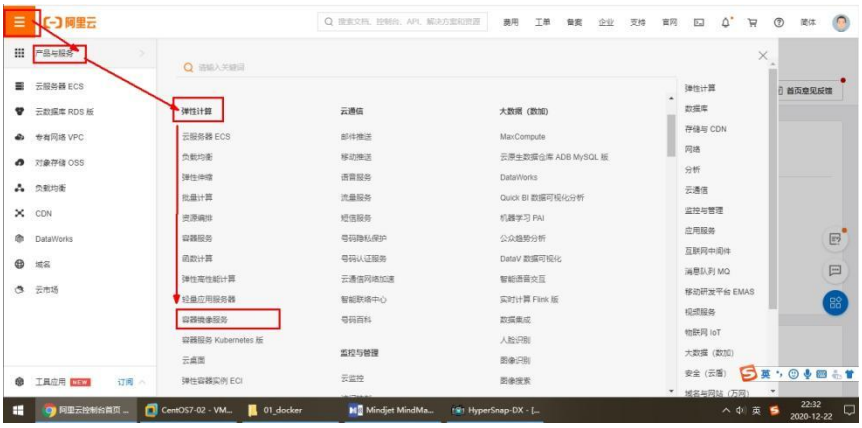
- systemctl stop docker
- yum remove docker-ce docker-ce-cli containerd.io
- rm -rf /var/lib/docker
- rm -rf /var/lib/containerd

1.2.5. 阿里云镜像加速

- 是什么
  - https://promotion.aliyun.com/ntms/act/kubernetes.html
- 注册一个属于自己的阿里云账户(可复用淘宝账号)
- 获得加速器地址连接
  - 登陆阿里云开发者平台

· 点击控制台

· 选择容器镜像服务



· 获取加速器地址

- 粘贴脚本直接执行
  - 直接粘

```
mkdir -p /etc/docker
```

```
tee /etc/docker/daemon.json <<-'EOF'
{
  "registry-mirrors":
["https://aa25jngu.mirror.aliyuncs.com"]
}
EOF
```

```
[root@zzyy ~]# mkdir -p /etc/docker
[root@zzyy ~]# tee /etc/docker/daemon.json <<-'EOF'
> {
>   "registry-mirrors": ["https://aa25jngu.mirror.aliyuncs.com"]
> }
> EOF
{
  "registry-mirrors": ["https://aa25jngu.mirror.aliyuncs.com"]
}
[root@zzyy ~]# cat /etc/docker/daemon.json
{
  "registry-mirrors": ["https://aa25jngu.mirror.aliyuncs.com"]
}
[root@zzyy ~]#
```

  - 或者分步骤都行
  - mkdir -p /etc/docker

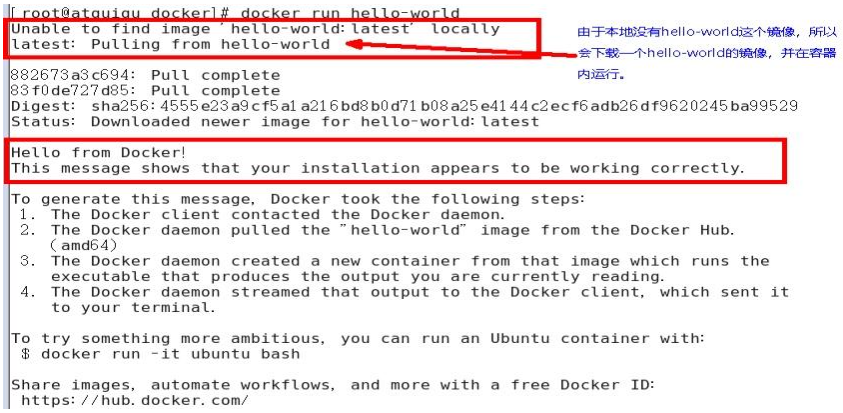
- vim /etc/docker/daemon.json

```
#阿里云
{
  "registry-mirrors": ["https://｛自己的编码｝.mirror.aliyuncs.com"]
}
```

- 重启服务器
  - systemctl daemon-reload
  - systemctl restart docker

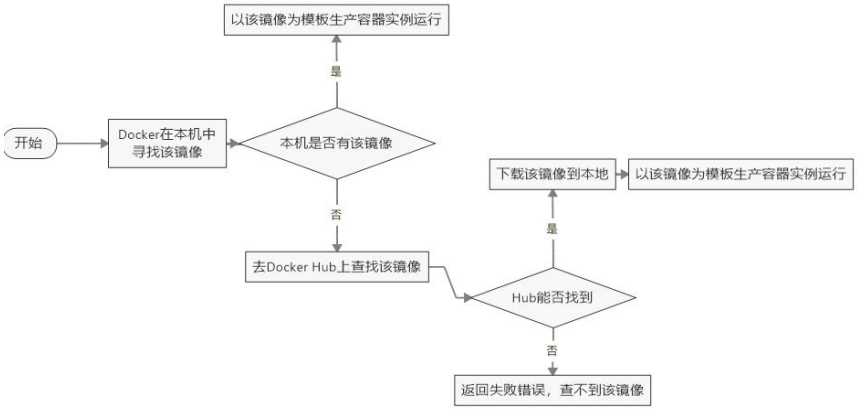
1.2.6. 永远的 HelloWorld

- 启动 Docker 后台容器(测试运行 hello-world)
  - docker run hello-world



输出这段提示以后，hello world 就会停止运行，容器自动终止。

- run 干了什么



1.2.7. 底层原理

关系	对比 - 开始 对比

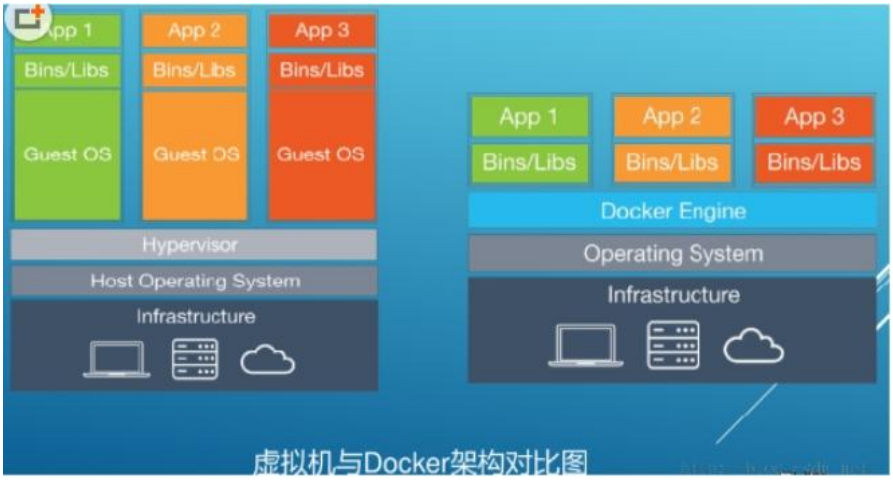
- 为什么 Docker 会比 VM 虚拟机快

(1)docker 有着比虚拟机更少的抽象层

由于 docker 不需要 Hypervisor(虚拟机)实现硬件资源虚拟化,运行在 docker 容器上的程序直接使用的都是实际物理机的硬件资源。因此在 CPU、内存利用率上 docker 将会在效率上有明显优势。

(2)docker 利用的是宿主机的内核,而不需要加载操作系统 OS 内核

当新建一个容器时,docker 不需要和虚拟机一样重新加载一个操作系统内核。进而避免引寻、加载操作系统内核返回等比较费时费资源的过程,当新建一个虚拟机时,虚拟机软件需要加载 OS,返回新建过程是分钟级别的。而 docker 由于直接利用宿主机的操作系统,则省略了返回过程,因此新建一个 docker 容器只需要几秒钟。



	Docker容器	虚拟机（VM）
操作系统	与宿主机共享OS	宿主机OS上运行虚拟机OS
存储大小	镜像小，便于存储与传输	镜像庞大（vmdk、vdi等）
运行性能	几乎无额外性能损失	操作系统额外的CPU、内存消耗
移植性	轻便、灵活，适应于Linux	笨重，与虚拟化技术耦合度高
硬件亲和性	面向软件开发者	面向硬件运维者
部署速度	快速，秒级	较慢，10s以上

1.3. Docker 常用命令

1.3.1. 帮助启动类命令

- 启动 docker： systemctl start docker
- 停止 docker： systemctl stop docker
- 重启 docker： systemctl restart docker
- 查看 docker 状态： systemctl status docker
- 开机启动： systemctl enable docker
- 查看 docker 概要信息： docker info

- 查看 docker 总体帮助文档： docker --help
- 查看 docker 命令帮助文档： docker 具体命令 --help

1.3.2. 镜像命令

- docker images
  - 列出本地主机上的镜像

各个选项说明:

REPOSITORY：表示镜像的仓库源
TAG：镜像的标签版本号
IMAGE ID：镜像 ID
CREATED：镜像创建时间
SIZE：镜像大小

同一仓库源可以有多个 TAG 版本，代表这个仓库源的不同个版本，我们使用 REPOSITORY:TAG 来定义不同的镜像。
如果你不指定一个镜像的版本标签，例如你只使用 ubuntu，docker 将默认使用 ubuntu:latest 镜像

  - OPTIONS 说明：
  - -a :列出本地所有的镜像（含历史映像层）
  - -q :只显示镜像 ID。
- docker search 某个 XXX 镜像名字
  - 网站
  - https://hub.docker.com
  - 命令

- docker search [OPTIONS] 镜像名字
- 案例

参数	说明
NAME	镜像名称
DESCRIPTION	镜像说明
STARS	点赞数量
OFFICIAL	是否是官方的
AUTOMATED	是否是自动构建的

```
[root@zzyy ~]# docker search redis
NAME                              DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
redis                             Redis is an open source key-value store that…   10018     [OK]
sameersbn/redis                                                                   83                   [OK]
grokzen/redis-cluster             Redis cluster 3.0, 3.2, 4.0, 5.0, 6.0, 6.2      79
rediscommander/redis-commander    Alpine image for redis-commander - Redis man…   66                   [OK]
redislabs/redisearch              Redis With the RedisSearch module pre-loaded…   39
redislabs/redisinsight            RedisInsight - The GUI for Redis                35
kubeguide/redis-master            redis-master with "Hello World!"                33
redislabs/redis                   Clustered in-memory database engine compatib…   31
oliver006/redis_exporter          Prometheus Exporter for Redis Metrics, Supp…    30
redislabs/rejson                  RedisJSON - Enhanced JSON data type processi…   27
arm32v7/redis                     Redis is an open source key-value store that…   25
redislabs/redisgraph              A graph database module for Redis               16                   [OK]
arm64v8/redis                     Redis is an open source key-value store that…   15
```

- OPTIONS 说明：
- --limit : 只列出 N 个镜像，默认 25 个
- docker search --limit 5 redis

- docker pull 某个 XXX 镜像名字
  - 下载镜像
  - docker pull 镜像名字[:TAG]
  - docker pull 镜像名字
  - 没有 TAG 就是最新版
  - 等价于
  - docker pull 镜像名字:latest
  - docker pull ubuntu

```
[root@zzyy ~]# docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
c549ccf8d472: Pull complete
Digest: sha256:aba80b77e27148d99c034a987e7da3a287ed455390352663418c0f2ed40417fe
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
[root@zzyy ~]# docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
ubuntu        latest    9873176a8ff5   2 days ago     72.7MB
hello-world   latest    d1165f221234   3 months ago   13.3kB
[root@zzyy ~]#
```

- docker system df 查看镜像/容器/数据卷所占的空间

```
[root@zzyy zzyyuse]# docker system df

TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images          7         5         1.434GB   715.9MB (49%)
Containers      29        0         63.52MB   63.52MB (100%)
Local Volumes   23        17        1.291GB   878.6MB (68%)
Build Cache     0         0         0B        0B
```

- docker rmi 某个 XXX 镜像名字 ID
  - 删除镜像
  - 删除单个
  - docker rmi -f 镜像 ID
  - 删除多个
  - docker rmi -f 镜像名 1:TAG 镜像名 2:TAG
  - 删除全部
  - docker rmi -f $(docker images -qa)
- 面试题：谈谈 docker 虚悬镜像是什么?
  - 是什么
  - 仓库名、标签都是<none>的镜像，俗称虚悬镜像 dangling image
  - 长什么样

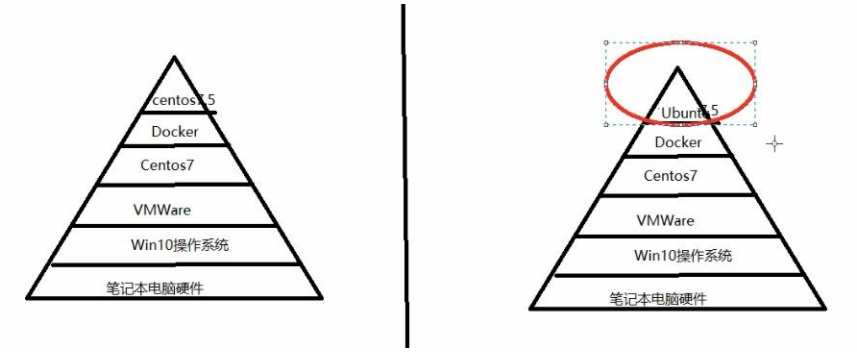
```
[root@zzyy zzyyuse]# docker images
REPOSITORY                    TAG       IMAGE ID       CREATED          SIZE
<none>                        <none>    da0fd00b7d91   9 seconds ago    72.8MB
atguigu/ubuntu                2.1       630e2281c13b   14 minutes ago   72.8MB
```

  - 后续 Dockerfile 章节再介绍

- 思考
  - 结合我们 Git 的学习心得，大家猜猜是否会有 docker commit /docker push？？

1.3.3. 容器命令

- **有镜像才能创建容器， 这是根本前提(下载一个 CentOS 或者 ubuntu 镜像演示)**
  - 说明



  - docker pull centos
  - docker pull ubuntu

```
[root@zzyy ~]# docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
c549ccf8d472: Pull complete
Digest: sha256:aba80b77e27148d99c034a987e7da3a287ed455390352663418c0f2ed40417fe
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
[root@zzyy ~]# docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
ubuntu        latest    9873176a8ff5   2 days ago     72.7MB
hello-world   latest    d1165f221234   3 months ago   13.3kB
[root@zzyy ~]#
```

  - 本次演示用 ubuntu 演示
- 新建+启动容器
  - docker run [OPTIONS] **IMAGE** [COMMAND] [ARG...]
  - OPTIONS 说明

OPTIONS 说明（常用）：有些是一个减号，有些是两个减号

--name="容器新名字"　　为容器指定一个名称；
-d: 后台运行容器并返回容器 ID，也即启动守护式容器(后台运行)；

-i：以交互模式运行容器，通常与 -t 同时使用；
-t：为容器重新分配一个伪输入终端，通常与 -i 同时使用；
也即启动交互式容器(前台有伪终端，等待交互)；

-P: 随机端口映射，大写 P
-p: 指定端口映射，小写 p

参数	说明
-p hostPort:containerPort	端口映射 -p 8080:80
-p ip:hostPort:containerPort	配置监听地址 -p 10.0.0.100:8080:80
-p ip::containerPort	随机分配端口 -p 10.0.0.100::80
-p hostPort:containerPort:udp	指定协议 -p 8080:80:tcp
-p 81:80 –p 443:443	指定多个

- 启动交互式容器(前台命令行)

关系	前后对比 - 开始 启动守护式容器(后台服务器)

```
[root@atguigu 桌面]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
tomcat              latest              8e956bfcc8a4        3 days ago          467.1 MB
nginx               latest              2be5c57f9f29        6 days ago          109 MB
centos              latest              88ec626ba223        7 days ago          199.7 MB
hello-world         latest              83f0de727d85        8 weeks ago         1.848 kB
[root@atguigu 桌面]# docker run -it centos /bin/bash
[root@b5bc0a5edac0 /]# ps -ef
UID         PID   PPID  C STIME TTY          TIME CMD
root          1      0  2 04:20 ?        00:00:00 /bin/bash
root         14      1  0 04:20 ?        00:00:00 ps -ef
[root@b5bc0a5edac0 /]#
```

#使用镜像 centos:latest 以交互模式启动一个容器,在容器内执行/bin/bash 命令。
docker run -it centos /bin/bash

参数说明：
-i: 交互式操作。
-t: 终端。
centos : centos 镜像。
/bin/bash：放在镜像名后的是命令，这里我

们希望有个交互式 Shell，因此用的是 /bin/bash。
要退出终端，直接输入 exit:

- 列出当前所有正在运行的容器
  - docker ps [OPTIONS]
  - OPTIONS 说明

OPTIONS 说明（常用）：

-a :列出当前所有正在运行的容器+历史上运行过的
-l :显示最近创建的容器。
-n：显示最近 n 个创建的容器。
-q :静默模式，只显示容器编号。

- 退出容器

关系	指向 进入正在运行的容器并以命令行交互

  - 两种退出方式
  - exit
  - run 进去容器，exit 退出，容器停止
  - ctrl+p+q
  - run 进去容器，ctrl+p+q 退出，容器不停止
- 启动已停止运行的容器
  - docker start 容器 ID 或者容器名
- 重启容器
  - docker restart 容器 ID 或者容器名
- 停止容器
  - docker stop 容器 ID 或者容器名

- 强制停止容器
  - docker kill 容器 ID 或容器名
- 删除已停止的容器
  - docker rm 容器 ID
  - 一次性删除多个容器实例
  - docker rm -f $(docker ps -a -q)
  - docker ps -a -q | xargs docker rm
- **重要**
  - **有镜像才能创建容器，这是根本前提(下载一个 Redis6.0.8 镜像演示)**
  - 启动守护式容器(后台服务器)

关系	前后对比 - 指向 启动交互式容器(前台命令行)

  - 在大部分的场景下，我们希望 docker 的服务是在后台运行的， 我们可以过 -d 指定容器的后台运行模式。
  - docker run -d 容器名

#使用镜像 centos:latest 以后台模式启动一个容器
docker run -d centos

问题：然后 docker ps -a 进行查看, 会发现容器已经退出
很重要的要说明的一点: Docker 容器后台运行, 就必须有一个前台进程.
容器运行的命令如果不是那些一直挂起的命令（比如运行 top，tail），就是会自动退出的。

这个是 docker 的机制问题,比如你的 web 容器,我们以 nginx 为例，正常情况下,

我们配置启动服务只需要启动响应的 service 即可。例如 service nginx start
但是,这样做,nginx 为后台进程模式运行,就导致 docker 前台没有运行的应用,
这样的容器后台启动后,会立即自杀因为他觉得他没事可做了.
所以，最佳的解决方案是,将你要运行的程序以前台进程的形式运行，
常见就是命令行模式，表示我还有交互操作，别中断，O(∩_∩)O 哈哈~

- redis 前后台启动演示 case
- 前台交互式启动
- docker run -it redis:6.0.8
- 后台守护式启动
- docker run -d redis:6.0.8
- 查看容器日志
- docker logs 容器 ID
- 查看容器内运行的进程
- docker top 容器 ID
- 查看容器内部细节
- docker inspect 容器 ID
- 进入正在运行的容器并以命令行交互

关系	开始 退出容器

- docker exec -it 容器 ID bashShell

```
[ root@atguigu 桌面]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
a2b057419f1f        centos              "/bin/sh -c 'while t…"   3 minutes ago       Up 3 minutes                            compassionate_leakey
[ root@atguigu 桌面]# docker exec -it a2b057419f1f /bin/bash
[ root@a2b057419f1f /]# ps -ef
UID         PID   PPID  C STIME TTY          TIME CMD
root          1      0  0 04:58 ?        00:00:00 /bin/sh -c while true;do echo hello zzyy;sleep 2;done
root        125      0  4 05:02 ?        00:00:00 /bin/bash
root        137      1  0 05:02 ?        00:00:00 sleep 2
root        138    125  0 05:02 ?        00:00:00 ps -ef
```

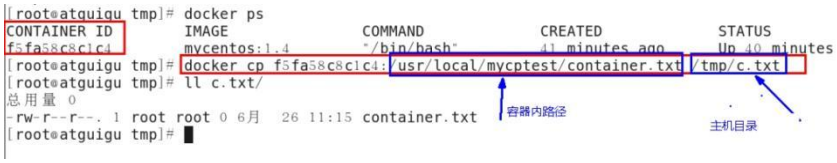
```
[root@zzyy ~]# docker exec --help

Usage:  docker exec [OPTIONS] CONTAINER COMMAND [ARG...]

Run a command in a running container

Options:
  -d, --detach               Detached mode: run command in the background
      --detach-keys string   Override the key sequence for detaching a container
  -e, --env list             Set environment variables
      --env-file list        Read in a file of environment variables
  -i, --interactive          Keep STDIN open even if not attached
      --privileged           Give extended privileges to the command
  -t, --tty                  Allocate a pseudo-TTY
  -u, --user string          Username or UID (format: <name|uid>[:<group|gid>])
  -w, --workdir string       Working directory inside the container
```

- 重新进入 docker attach 容器 ID
- 案例演示，用 centos 或者 unbuntu 都可以
- 上述两个区别
- attach 直接进入容器启动命令的终端，不会启动新的进程 用 exit 退出，会导致容器的停止。

- exec 是在容器中打开新的终端，并且可以启动新的进程 用 exit 退出，不会导致容器的停止。

- 推荐大家使用 docker exec 命令，因为退出容器终端，不会导致容器的停止。
- 用之前的 redis 容器实例进入试试
- 进入 redis 服务
- docker exec -it 容器 ID /bin/bash
- docker exec -it 容器 ID redis-cli
- 一般用-d 后台启动的程序，再用 exec 进入对应容器实例

- 从容器内拷贝文件到主机上
- 容器→主机
- docker cp 容器 ID:容器内路径 目的主机路径



公式：docker cp 容器 ID:容器内路径 目的主机路径

- 导入和导出容器
- export 导出容器的内容留作为一个 tar 归档文件[对应 import 命令]
- import 从 tar 包中的内容创建一个新的文件系统再导入为镜像[对应 export]
- 案例
- docker export 容器 ID > 文件名.tar

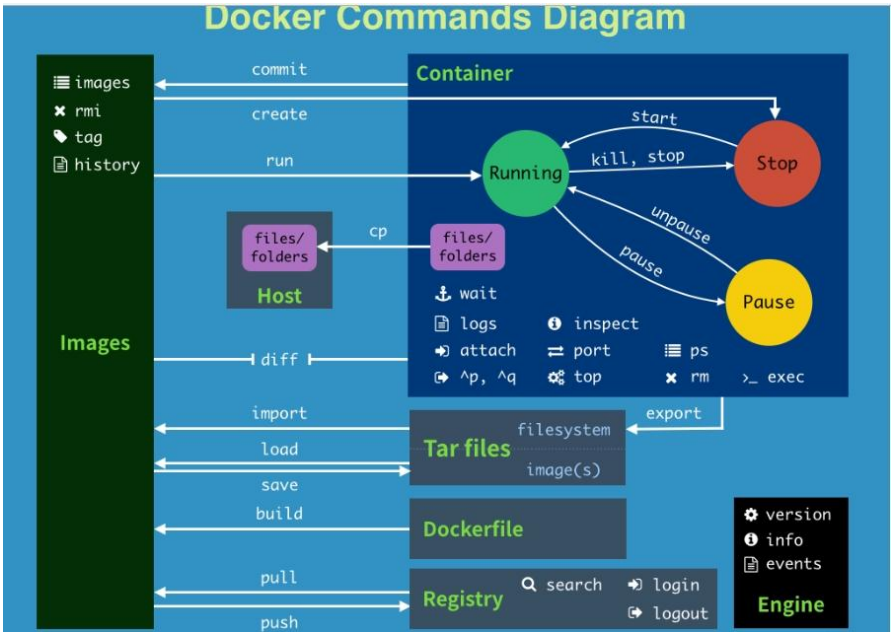
- cat 文件名.tar | docker import - 镜像用户/镜像名:镜像版本号

```
[root@zzyy zzyyuse]# docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED         STATUS         PORTS     NAMES
61279fc3cc6a   ubuntu    "/bin/bash"   4 minutes ago   Up 4 minutes             fervent_hawking
[root@zzyy zzyyuse]# docker stop 61279fc3cc6a
61279fc3cc6a
[root@zzyy zzyyuse]# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[root@zzyy zzyyuse]# docker images
REPOSITORY                          TAG       IMAGE ID       CREATED          SIZE
zzyy_docker                         1.6       5a3ccaebcb4a   19 hours ago     678MB
ubuntu                              latest    ba6acccedd29   11 days ago      72.8MB
192.168.111.162:5000/zzyyubuntu     1.2       d6ca70a4f932   11 days ago      105MB
mysql                               5.7       8a8a506ccfdc   2 weeks ago      448MB
registry                            latest    b2cb11db9d3d   8 weeks ago      26.2MB
redis                               6.0.8     16ecd2772934   12 months ago    104MB
java                                8         d23bdf5b1b1b   4 years ago      643MB
[root@zzyy zzyyuse]# docker rmi -f ubuntu
Untagged: ubuntu:latest
Untagged: ubuntu@sha256:626ffe58f6e7566e00254b638eb7e0f3b11d4da9675088f4781a50ae288f3322
Deleted: sha256:ba6acccedd2923aee4c2acc6a23780b14ed4b8a5fa4e14e252a23b846df9b6c1
[root@zzyy zzyyuse]# cat abcd.tar.gz | docker import - atguigu/ubuntu:2.1
sha256:630e2281c13b91c9ae3ba98f99da7969b0c2698e5a3d89de74ceabada39a861b
[root@zzyy zzyyuse]#
```

1.3.4. 小总结

- 常用命令

图片正下方还有命令



attach Attach to a running container # 当前 shell 下 attach 连接指定运行镜像
build Build an image from a Dockerfile # 通过 Dockerfile 定制镜像
commit Create a new image from a container changes # 提交当前容器为新的镜像
cp Copy files/folders from the containers filesystem to the host path #从容器中拷贝指定文件或者目录到宿主机中

create Create a new container # 创建一个新的容器，同 run，但不启动容器
diff Inspect changes on a container's filesystem # 查看 docker 容器变化
events Get real time events from the server # 从 docker 服务获取容器实时事件
exec Run a command in an existing container # 在已存在的容器上运行命令
export Stream the contents of a container as a tar archive # 导出容器的内容流作为一个 tar 归档文件[对应 import]
history Show the history of an image # 展示一个镜像形成历史
images List images # 列出系统当前镜像
import Create a new filesystem image from the contents of a tarball # 从 tar 包中的内容创建一个新的文件系统映像[对应 export]
info Display system-wide information # 显示系统相关信息
inspect Return low-level information on a container # 查看容器详细信息
kill Kill a running container # kill 指定 docker 容器
load Load an image from a tar archive # 从一个 tar 包中加载一个镜像[对应 save]
login Register or Login to the docker registry server # 注册或者登陆一个 docker 源服务器
logout Log out from a Docker registry server # 从当前 Docker registry 退出
logs Fetch the logs of a container # 输出当前容器日志信息
port Lookup the public-facing port which is NAT-ed to PRIVATE_PORT # 查看映射端口对应的容器内部源端口

pause Pause all processes within a container # 暂停容器
ps List containers # 列出容器列表
pull Pull an image or a repository from the docker registry server # 从 docker 镜像源服务器拉取指定镜像或者库镜像
push Push an image or a repository to the docker registry server # 推送指定镜像或者库镜像至 docker 源服务器
restart Restart a running container # 重启运行的容器
rm Remove one or more containers # 移除一个或者多个容器
rmi Remove one or more images # 移除一个或多个镜像[无容器使用该镜像才可删除，否则需删除相关容器才可继续或 -f 强制删除]
run Run a command in a new container # 创建一个新的容器并运行一个命令
save Save an image to a tar archive # 保存一个镜像为一个 tar 包[对应 load]
search Search for an image on the Docker Hub # 在 docker hub 中搜索镜像
start Start a stopped containers # 启动容器
stop Stop a running containers # 停止容器
tag Tag an image into a repository # 给源中镜像打标签
top Lookup the running processes of a container # 查看容器中运行的进程信息
unpause Unpause a paused container # 取消暂停容器
version Show the docker version information # 查看 docker 版本号

wait　　Block until a container stops, then print its exit code　# 截取容器停止时的退出状态值

1.4. Docker 镜像

1.4.1. 是什么

- 是什么

镜像

是一种轻量级、可执行的独立软件包，它包含运行某个软件所需的所有内容，我们把应用程序和配置依赖打包好形成一个可交付的运行环境(包括代码、运行时需要的库、环境变量和配置文件等)，这个打包好的运行环境就是 image 镜像文件。

只有通过这个镜像文件才能生成 Docker 容器实例(类似 Java 中 new 出来一个对象)。

- 分层的镜像

以我们的 pull 为例，在下载的过程中我们可以看到 docker 的镜像好像是在一层一层的在下载

```
[root@atguigu docker]# docker pull tomcat
latest: Pulling from tomcat

841c6d57cd5e: Pulling fs layer
4f5904b5ae9a: Pulling fs layer
ce84c5fb1edc: Pulling fs layer
08e49d2ba62c: Pulling fs layer
6d59a7a1e69a: Pull complete
8069d49c4094: Pull complete
37874ba173f4: Pull complete
b8d64e7efa17: Pull complete
217bbfb8e851: Pull complete
9d56e04171d7: Pull complete
d092f10455ea: Pull complete
6db0adc8ce3c: Pull complete
c96fcb778526: Downloading [==============================>                    ] 72.44 MB/122.1 MB
b1f139ba4dd2: Download complete
8834a7ffc401: Download complete
995ada5f35d2: Download complete
ec6ce6f03e11: Download complete
4bac12a809fc: Download complete
0b043134e0bc: Download complete
0e7992fbc673: Download complete
0b7a981495f3: Download complete
2630c6ab33a1: Download complete
845f8c1e3527: Download complete
21b55be4aa8a: Download complete
ac5033d93a00: Download complete
```

- UnionFS（联合文件系统）

UnionFS（联合文件系统）：Union 文件系统（UnionFS）是一种分层、轻量级并且高性能的文件系统，它支持对文件系统的修改作为一次提交来一层层的叠加，同时可以将不同目录挂载到同一个虚拟文件系统下(unite several directories into a single virtual filesystem)。Union 文件系统是 Docker 镜像的基础。镜像可以通过分层来进行继承，基于基础镜像（没有父镜像），可以制作各种具体的应用镜像。

特性：一次同时加载多个文件系统，但从外面看起来，只能看到一个文件系统，联合加载会把各层文件系统叠加起来，这样最终的文件系统会包含所有底层的文件和目录

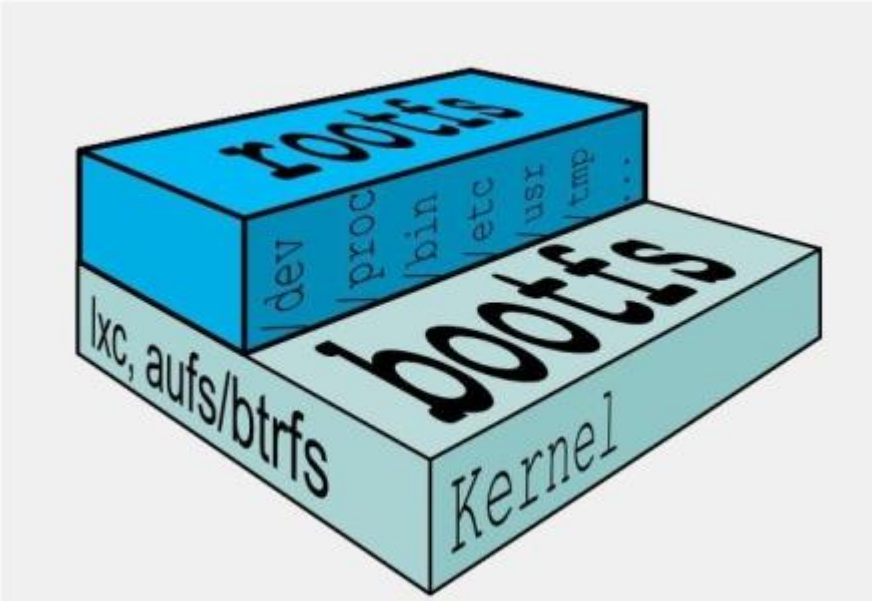
- Docker 镜像加载原理

Docker 镜像加载原理：

docker 的镜像实际上由一层一层的文件系统组成，这种层级的文件系统 UnionFS。bootfs(boot file system)主要包含 bootloader 和 kernel, bootloader 主要是引导加载 kernel, Linux 刚启动时会加载 bootfs 文件系统，在 Docker 镜像的最底层是引导文件系统 bootfs。这一层与我们典型的 Linux/Unix 系统是一样的，包含 boot 加载器和内核。当 boot 加载完成之后整个内核就都在内存中了，此时内存的使用权已由 bootfs 转交给内核，此时系统也会卸载 bootfs。

rootfs (root file system)，在 bootfs 之上。包含的就是典型 Linux 系统中的 /dev, /proc, /bin, /etc 等标准目录和文件。rootfs 就是各种不同的操作系统发行版，比如 Ubuntu，Centos 等等。

。



平时我们安装进虚拟机的 CentOS 都是好几个 G，为什么 docker 这里才 200M？？

```
[root@atguigu docker]# docker images centos
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
centos              latest              88ec626ba223        2 weeks ago         199.7 MB
[root@atguigu docker]#
```

对于一个精简的 OS，rootfs 可以很小，只需要包括最基本的命令、工具和程序库就可以了，因为底层直接用 Host 的 kernel，自己只需要提供 rootfs 就行了。由此可见对于不同的 linux 发行版, bootfs 基本是一致的, rootfs 会有差别, 因此不同的发行版可以公用 bootfs。

- 为什么 Docker 镜像要采用这种分层结构呢

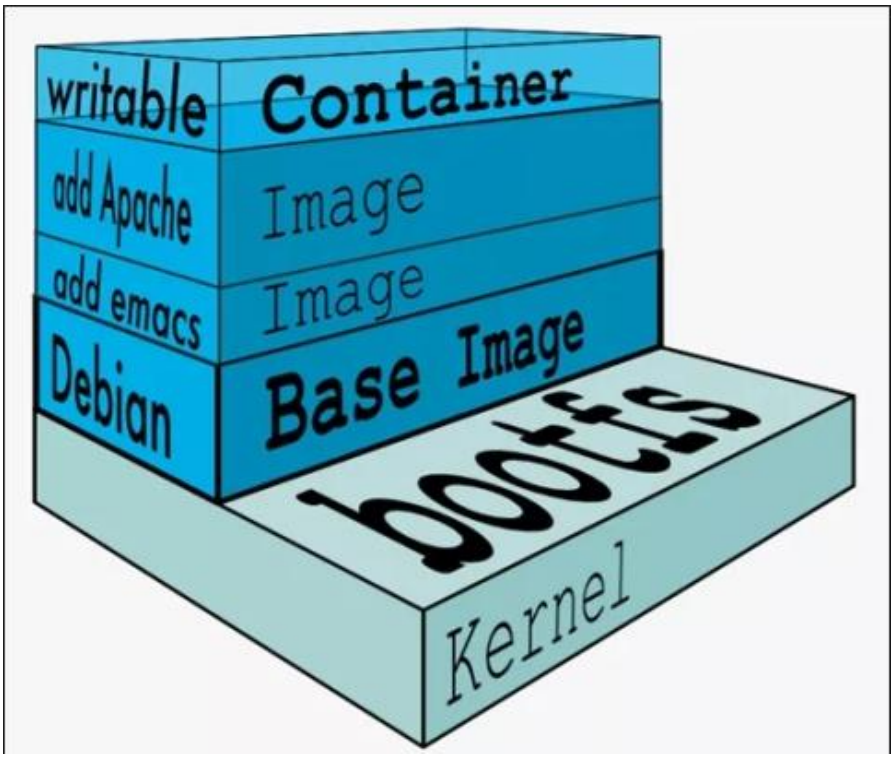
镜像分层最大的一个好处就是共享资源，方便复制迁移，就是为了复用。

比如说有多个镜像都从相同的 base 镜像构建而来，那么 Docker Host 只需在磁盘上保存一份 base 镜像；
同时内存中也只需加载一份 base 镜像，就可以为所有容器服务了。而且镜像的每一层都可以被共享。

1.4.2. 重点理解

- Docker 镜像层都是只读的，容器层是可写的 当容器启动时，一个新的可写层被加载到镜像的顶部。 这一层通常被称作“容器层”，“容器层”之下的都叫“镜像层”。

当容器启动时，一个新的可写层被加载到镜像的顶部。这一层通常被称作“容器层”，“容器层”之下的都叫“镜像层”。
所有对容器的改动 - 无论添加、删除、还是修改文件都只会发生在容器层中。只有容器层是可写的，容器层下面的所有镜像层都是只读的。



1.4.3. Docker 镜像 commit 操作案例

- docker commit 提交容器副本使之成为一个新的镜像
- docker commit -m="提交的描述信息" -a="作者" 容器 ID 要创建的目标镜像名:[标签名]
- 案例演示 ubuntu 安装 vim
  - 从 Hub 上下载 ubuntu 镜像到本地并成功运行
  - 原始的默认 Ubuntu 镜像是不带着 vim 命令的

```
[root@zzyy docker]# docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
ubuntu        latest    7e0aa2d69a15   6 weeks ago     72.7MB
hello-world   latest    d1165f221234   3 months ago    13.3kB
redis         6.0.8     16ecd2772934   7 months ago    104MB
[root@zzyy docker]# docker run -it ubuntu /bin/bash
root@685c94a50fb1:/# vim a.txt
bash: vim: command not found
root@685c94a50fb1:/#
```

  - 外网连通的情况下，安装 vim

```
# 先更新我们的包管理工具
apt-get update
# 然后安装我们需要的vim
apt-get install vim
```

docker 容器内执行上述两条命令：
apt-get update
apt-get -y install vim

```
[root@zzyy mytest]# docker run -it ubuntu
root@5fa45c3697b7:/# vi a.txt
bash: vi: command not found
root@5fa45c3697b7:/# apt-get update
Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:2 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:3 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [728 kB]
Get:4 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:5 http://archive.ubuntu.com/ubuntu focal-backports InRelease [101 kB]
Get:6 http://archive.ubuntu.com/ubuntu focal/restricted amd64 Packages [33.4 kB]
Get:7 http://archive.ubuntu.com/ubuntu focal/multiverse amd64 Packages [177 kB]
Get:8 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 Packages [27.6 kB]
Get:9 http://archive.ubuntu.com/ubuntu focal/universe amd64 Packages [11.3 MB]
Get:10 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [876 kB]
Get:11 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [323 kB]
37% [9 Packages 1882 kB/11.3 MB 17%]
```

```
Fetched 18.0 MB in 1min 34s (191 kB/s)
Reading package lists... Done
root@5fa45c3697b7:/# apt-get -y install vim
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  alsa-topology-conf alsa-ucm-conf file libasound2 libasound2-data libcanberra0 libexpat1 libgpm2 liblt
  libmagic1 libmpdec2 liboggo libpython3.8 libpython3.8-minimal libpython3.8-stdlib libreadline8 libsql
  libtdb1 libvorbis0a libvorbisfile3 mime-support readline-common sound-theme-freedesktop vim-common vi
  xz-utils
Suggested packages:
```

- 安装完成后，commit 我们自己的新镜像

```
Fetched 18.0 MB in 1min 34s (191 kB/s)
Reading package lists... Done
root@5fa45c3697b7:/# apt-get -y install vim
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  alsa-topology-conf alsa-ucm-conf file libasound2 libasound2-data libcanberra0 libexpat1 libgpm2 liblt
  libmagic1 libmpdec2 liboggo libpython3.8 libpython3.8-minimal libpython3.8-stdlib libreadline8 libsql
  libtdb1 libvorbis0a libvorbisfile3 mime-support readline-common sound-theme-freedesktop vim-common vi
  xz-utils
Suggested packages:
[root@zzyy mytest]# docker commit -m="add vim cmd" -a="zzyy" 5fa45c3697b7 atguigu/myubuntu:1.1
sha256:e7dd7fdf3fba278ea75e4e147c45258cb76cf291674efc3d0baef56d255a64b9
[root@zzyy mytest]# docker images
REPOSITORY                                          TAG       IMAGE ID       CREATED          SIZE
atguigu/myubuntu                                    1.1       e7dd7fdf3fba   4 seconds ago    170MB
registry.cn-hangzhou.aliyuncs.com/atguiguwh/myubuntu 1.1      cea1bb40441c   3 hours ago      170MB
ubuntu                                              latest    7e0aa2d69a15   6 weeks ago      72.7MB
hello-world                                         latest    d1165f221234   3 months ago     13.3kB
redis                                               6.0.8     16ecd2772934   7 months ago     104MB
[root@zzyy mytest]# docker run -it e7dd7fdf3fba
root@1e548eefd5f8:/# vim a.txt
root@1e548eefd5f8:/#
```

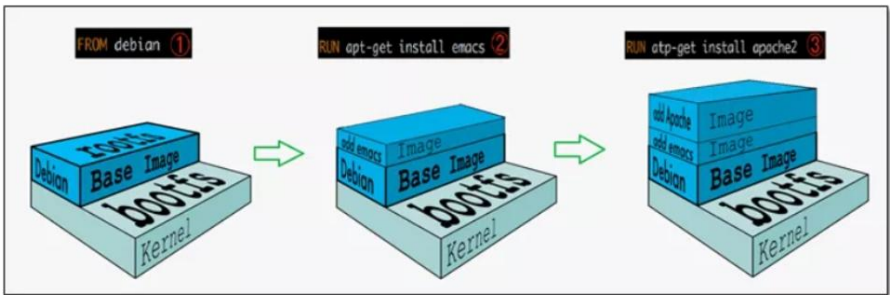
- 启动我们的新镜像并和原来的对比

```
[root@zzyy mytest]# docker commit -m="add vim cmd" -a="zzyy" 5fa45c3697b7 atguigu/myubuntu:1.1
sha256:e7dd7fdf3fba278ea75e4e147c45258cb76cf291674efc3d0baef56d255a64b9
[root@zzyy mytest]# docker images
REPOSITORY                                          TAG       IMAGE ID       CREATED          SIZE
atguigu/myubuntu                                    1.1       e7dd7fdf3fba   4 seconds ago    170MB
registry.cn-hangzhou.aliyuncs.com/atguiguwh/myubuntu 1.1      cea1bb40441c   3 hours ago      170MB
ubuntu                                              latest    7e0aa2d69a15   6 weeks ago      72.7MB
hello-world                                         latest    d1165f221234   3 months ago     13.3kB
redis                                               6.0.8     16ecd2772934   7 months ago     104MB
[root@zzyy mytest]# docker run -it e7dd7fdf3fba
root@1e548eefd5f8:/# vim a.txt
root@1e548eefd5f8:/#
```

1 官网是默认下载的 Ubuntu 没有 vim 命令
2 我们自己 commit 构建的镜像，新增加了 vim 功能，可以成功使用。

- 小总结

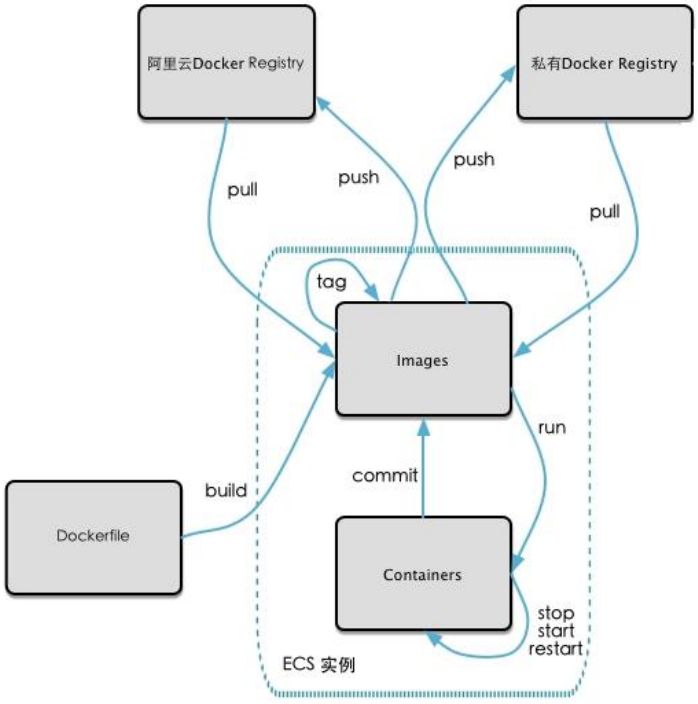
Docker 中的镜像分层，支持通过扩展现有镜像，创建新的镜像。类似 Java 继承于一个 Base 基础类，自己再按需扩展。
新镜像是从 base 镜像一层一层叠加生成的。每安装一个软件，就在现有镜像的基础上增加一层



1.5. 本地镜像发布到阿里云

1.5.1. 本地镜像发布到阿里云流程

阿里云ECS Docker生态如下图所示：



1.5.2. 镜像的生成方法

- 上一讲已经介绍过
  - 基于当前容器创建一个新的镜像，新功能增强 docker commit [OPTIONS] 容器 ID [REPOSITORY[:TAG]]

OPTIONS 说明：

-a :提交的镜像作者；

-m :提交时的说明文字；

本次案例 centos+ubuntu 两个，当堂讲解一个，家庭作业一个，请大家务必动手，亲自实操。

```
[root@atguigu mydockerfile]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
c167ee237b87        mycentos:1.3        "/bin/sh -c /bin/bas…"   About a minute ago   Up About a minute   80/tcp              
[root@atguigu mydockerfile]# docker commit -a zzyy -m "new mycentos 1.4 from 1.3" c167ee237b87 mycentos:1.4
ea5d769ae5106fdd44dd8769066245c2bd955871bb6344775b7f5c13a0638903
[root@atguigu mydockerfile]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
mycentos            1.4                 ea5d769ae510        6 seconds ago       353.4 MB
mycentos            1.3                 cd535f1b1fbd        31 minutes ago      353.4 MB
redis               3.2                 d7feea0c3d63        4 days ago          99.7 MB
tomcat              latest              8e956bfcc8a4        12 days ago         467.1 MB
ubuntu              latest              7feff7652c69        2 weeks ago         81.15 MB
nginx               latest              2be5c57f9f29        2 weeks ago         109 MB
centos              latest              88ec626ba223        2 weeks ago         199.7 MB
mysql               5.6                 338cffc3e06a        6 weeks ago         256.1 MB
hello-world         latest              83f0de727d85        10 weeks ago        1.848 kB
[root@atguigu mydockerfile]# docker run -it mycentos:1.4
[root@57e73277f969 local]# pwd
/usr/local
[root@57e73277f969 local]#
```

```
[root@zzyy ~]# docker images
REPOSITORY          TAG         IMAGE ID       CREATED          SIZE
atguigu/myubuntu    1.1         cea1bb40441c   4 minutes ago    170MB
ubuntu              latest      7e0aa2d69a15   6 weeks ago      72.7MB
hello-world         latest      d1165f221234   3 months ago     13.3kB
redis               6.0.8       16ecd2772934   7 months ago     104MB
[root@zzyy ~]# docker run -it ubuntu /bin/bash
root@2903fd8e9795:/# vim a.txt
bash: vim: command not found
root@2903fd8e9795:/# exit
exit
[root@zzyy ~]# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[root@zzyy ~]# docker run -it atguigu/myubuntu:1.1 /bin/bash
root@0be6531cb088:/# vim a.txt
root@0be6531cb088:/#
```

- 后面的 DockerFile 章节，第 2 种方法

1.5.3. 将本地镜像推送到阿里云

- 本地镜像素材原型

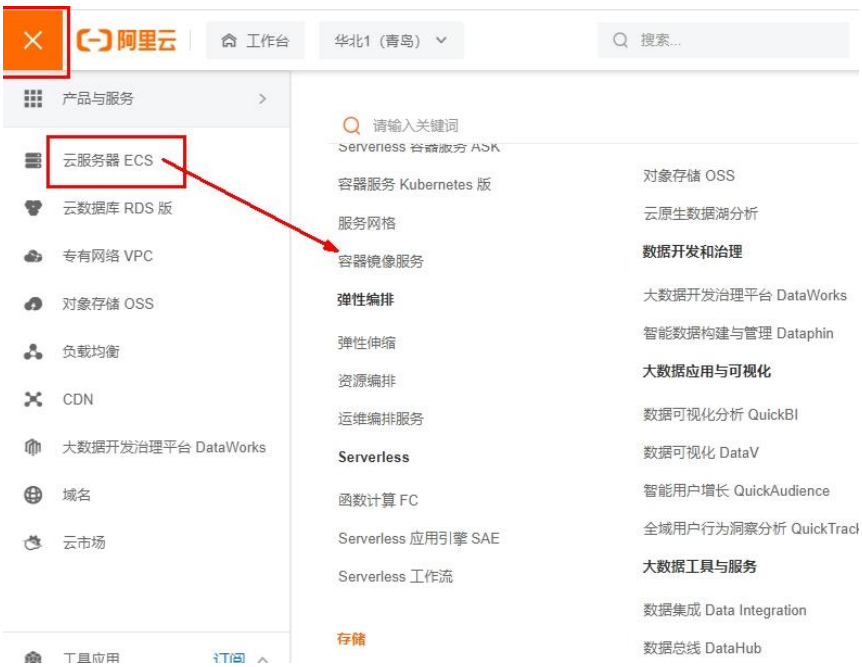
```
[root@atguigu mydockerfile]# docker images mycentos
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
mycentos            1.4                 7dce21d22a80        About a minute ago   353.4 MB
```

```
[root@zzyy ~]# docker images
REPOSITORY          TAG         IMAGE ID       CREATED          SIZE
atguigu/myubuntu    1.1         cea1bb40441c   8 minutes ago    170MB
```

- 阿里云开发者平台
  - https://promotion.aliyun.com/ntms/act/kubernetes.html

- 创建仓库镜像

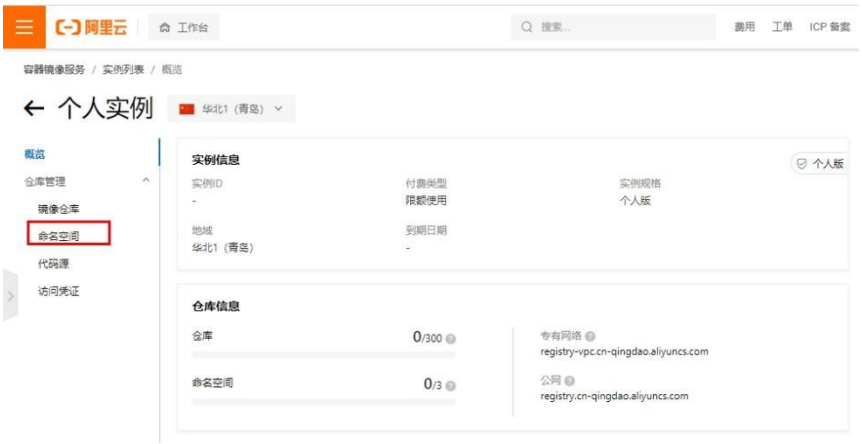
- 选择控制台，进入容器镜像服务


阿里云
工作台
华北1（青岛）
搜索...
产品与服务
云服务器 ECS
云数据库 RDS 版
专有网络 VPC
对象存储 OSS
负载均衡
CDN
大数据开发治理平台 DataWorks
域名
云市场
请输入关键词
容器服务 Kubernetes 版
服务网格
容器镜像服务
弹性编排
弹性伸缩
资源编排
运维编排服务
Serverless
函数计算 FC
Serverless 应用引擎 SAE
Serverless 工作流
存储
对象存储 OSS
云原生数据湖分析
数据开发和治理
大数据开发治理平台 DataWorks
智能数据构建与管理 Dataphin
大数据应用与可视化
数据可视化分析 QuickBI
数据可视化 DataV
智能用户增长 QuickAudience
大数据工具与服务
数据集成 Data Integration
数据总线 DataHub

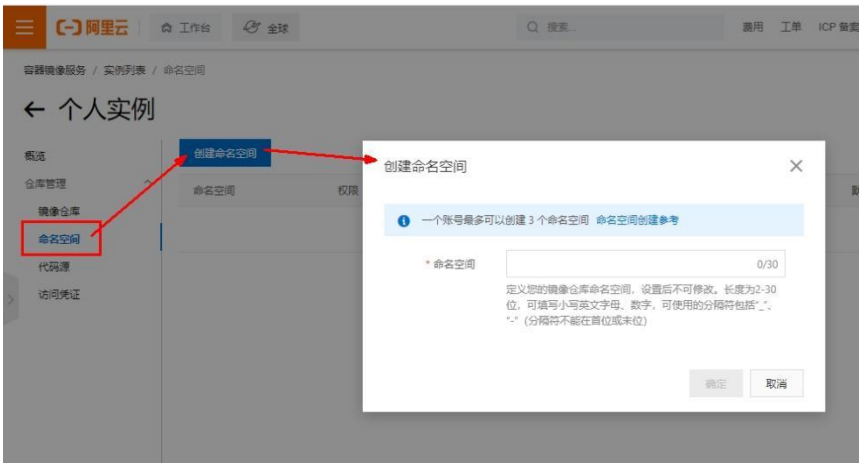
· 选择个人实例


阿里云
工作台
华北1（青岛）
费用
工单
ICP 备案
容器镜像服务
实例列表
镜像中心
镜像工具
容器镜像服务 / 实例列表
实例列表
创建企业版实例
全部展示
个人实例
个人版
个人容器镜像托管服务，限额使用
运行状态
运行中
实例规格
个人版
操作
导入企业版
企业版
企业级安全托管和高效分发服务，限时优惠购买
什么是企业版实例?
如何无缝从自建迁移企业版实例?

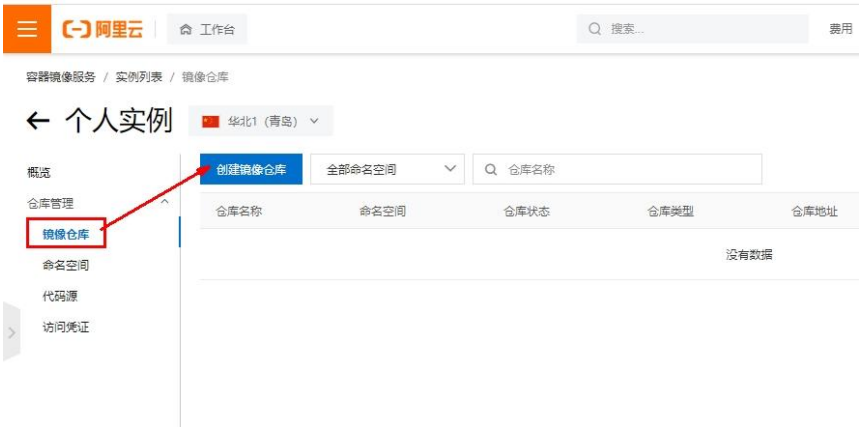
· 命名空间


阿里云
工作台
费用
工单
ICP 备案
容器镜像服务 / 实例列表 / 概览
个人实例
华北1（青岛）
概览
仓库管理
镜像仓库
命名空间
代码源
访问凭证
实例信息
个人版
实例ID
付费类型
限额使用
实例规格
个人版
地域
华北1（青岛）
到期日期
仓库信息
仓库
0/300
命名空间
0/3
专有网络
registry-vpc.cn-qingdao.aliyuncs.com
公网
registry.cn-qingdao.aliyuncs.com

· 继续

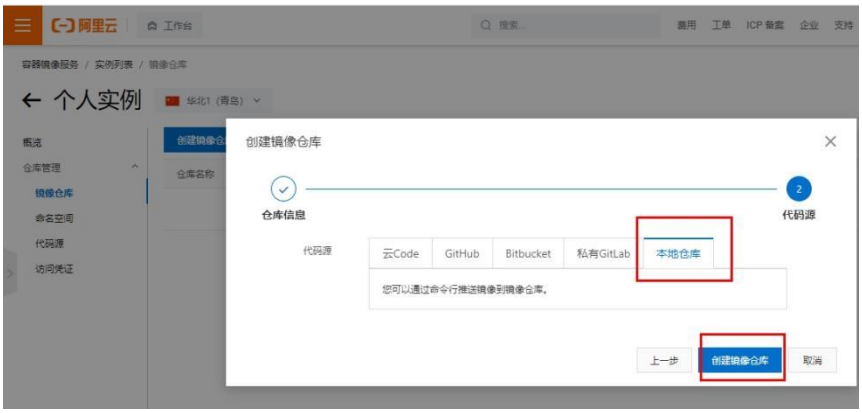

阿里云
工作台
全球
容器镜像服务 / 实例列表 / 命名空间
← 个人实例
概览
仓库管理
镜像仓库
命名空间
代码源
访问凭证
创建命名空间
命名空间
权限
创建命名空间
一个账号最多可以创建 3 个命名空间 命名空间创建参考
* 命名空间
0/30
定义您的镜像仓库命名空间，设置后不可修改。长度为2-30位，可填写小写英文字母、数字，可使用的分隔符包括"_"、"-"（分隔符不能在首位或末位）
确定
取消

· 仓库名称


阿里云
工作台
搜索...
费用
容器镜像服务 / 实例列表 / 镜像仓库
← 个人实例
华北1（青岛）
概览
仓库管理
镜像仓库
命名空间
代码源
访问凭证
创建镜像仓库
全部命名空间
仓库名称
仓库名称
命名空间
仓库状态
仓库类型
仓库地址
没有数据

· 继续


创建镜像仓库
仓库信息
代码源
地域
华北1（青岛）
* 命名空间
atguigubj
仓库名称
myubuntu
8/64
长度为2-64个字符，可使用小写英文字母、数字，可使用分隔符"_"、"-"、"."（分隔符不能在首位或末位）
仓库类型
公开
私有
* 摘要
添加了vim命令的加强版ubuntu
18/100
下一步
取消



· 进入管理界面获得脚本

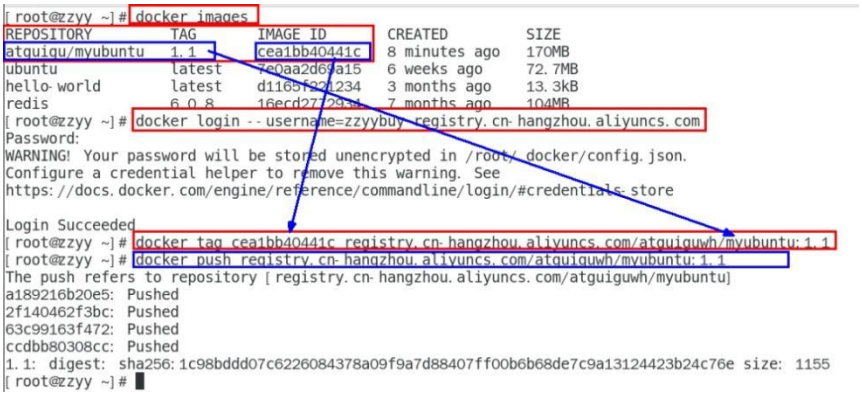
· 将镜像推送到阿里云

· 将镜像推送到阿里云 registry

· 管理界面脚本

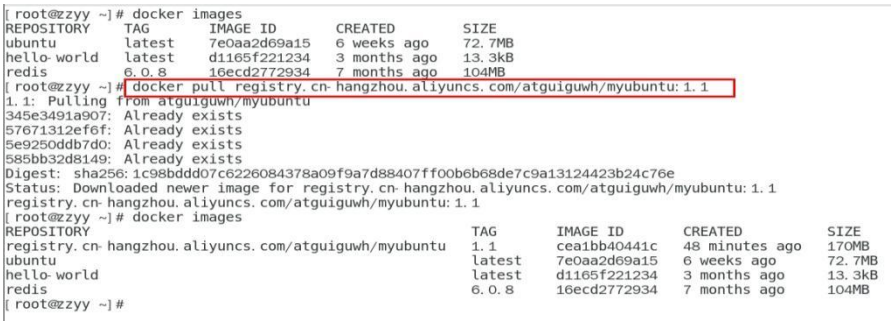
· 脚本实例

docker login --username=zzyybuy registry.cn-hangzhou.aliyuncs.com
docker tag cea1bb40441c registry.cn-hangzhou.aliyuncs.com/atguiguwh/myubuntu:1.1
docker push registry.cn-hangzhou.aliyuncs.com/atguiguwh/myubuntu:1.1
上面命令是阳哥自己本地的，你自己酌情处理，不要粘贴我的。



1.5.4. 将阿里云上的镜像下载到本地

- 下载到本地

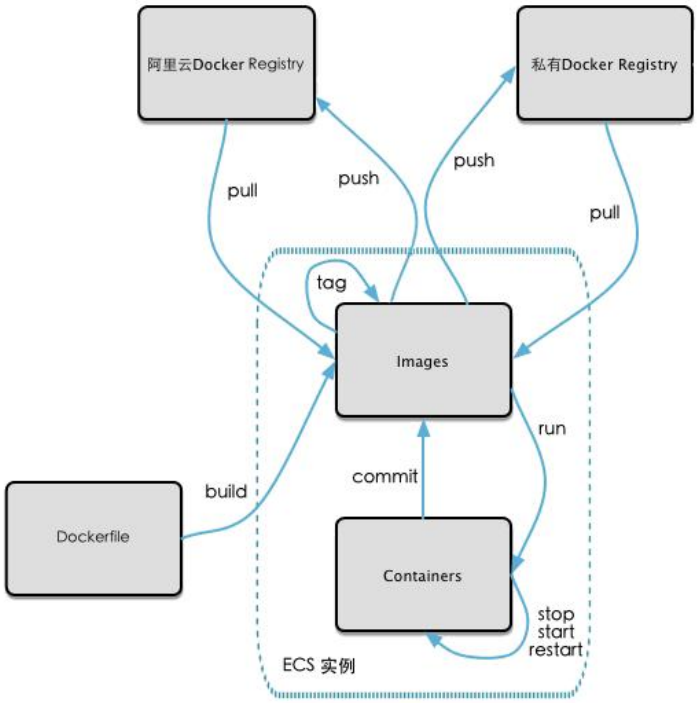


```
docker pull registry.cn-hangzhou.aliyuncs.com/atguiguwh/myubuntu:1.1
```

1.6. 本地镜像发布到私有库

1.6.1. 本地镜像发布到私有库流程

阿里云ECS Docker生态如下图所示：



1.6.2. 是什么

1 官方 Docker Hub 地址：https://hub.docker.com/，中国大陆访问太慢了且准备被阿里云取代的趋势，不太主流。

2 Dockerhub、阿里云这样的公共镜像仓库可能不太方便，涉及机密的公司不可能提供镜像给公网，所以需要创建一个本地私人仓库供给团队使用，基于公司内部项目构建镜像。

Docker Registry 是官方提供的工具，可以用于构建私有镜像仓库

- Docker Registry

1.6.3. 将本地镜像推送到私有库

- 下载镜像 Docker Registry

docker pull registry

```
[root@zzyy ~]# docker pull registry
Using default tag: latest
latest: Pulling from library/registry
Digest: sha256:265d4a5ed8bf0df27d1107edb00b70e658ee9aa5acb3f37336c5a17db634481e
Status: Image is up to date for registry:latest
docker.io/library/registry:latest
[root@zzyy ~]#
```

```
[root@zzyy ~]# docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
mysql         5.7       8a8a506ccfdc   2 days ago       448MB
ubuntu        latest    597ce1600cf4   2 weeks ago      72.8MB
hello-world   latest    feb5d9fea6a5   3 weeks ago      13.3kB
registry      latest    b2cb11db9d3d   6 weeks ago      26.2MB
redis         6.0.8     16ecd2772934   11 months ago    104MB
[root@zzyy ~]#
```

- 运行私有库 Registry，相当于本地有个私有 Docker hub

docker run -d -p 5000:5000 -v /zzyyuse/myregistry/:/tmp/registry --privileged=true registry

默认情况，仓库被创建在容器的/var/lib/registry 目录下，建议自行用容器卷映射，方便于宿主机联调

```
[root@zzyy ~]#
[root@zzyy ~]# docker run -d -p 5000:5000 -v /zzyyuse/myregistry/:/tmp/registry --privileged=true registry
5c0ed56024f66484a7a18e1c9ebb9cf7063c1fc9ce11b2a34cfee6d380d77a19
[root@zzyy ~]#
[root@zzyy ~]# docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED         STATUS         PORTS
                                                                                                  NAMES
5c0ed56024f6   registry   "/entrypoint.sh /etc…"   6 seconds ago   Up 5 seconds   0.0.0.0:5000->5000/tcp, :::5000->5000/tcp   inspiring_bardeen
[root@zzyy ~]#
```

- 案例演示创建一个新镜像，ubuntu 安装 ifconfig 命令
  - 从 Hub 上下载 ubuntu 镜像到本地并成功运行
  - 原始的 Ubuntu 镜像是不带着 ifconfig 命令的

```
[root@zzyy ~]# docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
mysql         5.7       8a8a506ccfdc   2 days ago      448MB
ubuntu        latest    597ce1600cf4   2 weeks ago     72.8MB
hello-world   latest    feb5d9fea6a5   3 weeks ago     13.3kB
registry      latest    b2cb11db9d3d   6 weeks ago     26.2MB
redis         6.0.8     16ecd2772934   11 months ago   104MB
[root@zzyy ~]# docker run -it ubuntu /bin/bash
root@a69d7c825c4f:/# ifconfig
bash: ifconfig: command not found
root@a69d7c825c4f:/#
```

  - 外网连通的情况下，安装 ifconfig 命令并测试通过

docker 容器内执行上述两条命令：

apt-get update

apt-get install net-tools

```
root@a69d7c825c4f:/# apt-get update
Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [
Get:2 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:3 http://security.ubuntu.com/ubuntu focal-security/multiverse
Get:4 http://security.ubuntu.com/ubuntu focal-security/main amd64
Get:5 http://archive.ubuntu.com/ubuntu focal-updates InRelease [1
Get:6 http://archive.ubuntu.com/ubuntu focal-backports InRelease [
Get:7 http://archive.ubuntu.com/ubuntu focal/restricted amd64 Pac
Get:8 http://archive.ubuntu.com/ubuntu focal/universe amd64 Packa
Get:9 http://security.ubuntu.com/ubuntu focal-security/restricted
Get:10 http://security.ubuntu.com/ubuntu focal-security/universe
Get:11 http://archive.ubuntu.com/ubuntu focal/main amd64 Packages
Get:12 http://archive.ubuntu.com/ubuntu focal/multiverse amd64 Pa
Get:13 http://archive.ubuntu.com/ubuntu focal-updates/multiverse
Get:14 http://archive.ubuntu.com/ubuntu focal-updates/main amd64
Get:15 http://archive.ubuntu.com/ubuntu focal-updates/universe am
Get:16 http://archive.ubuntu.com/ubuntu focal-updates/restricted
Get:17 http://archive.ubuntu.com/ubuntu focal-backports/main amd6
Get:18 http://archive.ubuntu.com/ubuntu focal-backports/universe
Fetched 19.3 MB in 1min 31s (212 kB/s)
Reading package lists... Done
root@a69d7c825c4f:/# apt-get install net-tools
Reading package lists... Done
```

```
root@a69d7c825c4f:/#
root@a69d7c825c4f:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.3  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:03  txqueuelen 0  (Ethernet)
        RX packets 8862  bytes 19999599 (19.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6781  bytes 372481 (372.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@a69d7c825c4f:/#
```

- 安装完成后，commit 我们自己的新镜像

公式：

docker commit -m="提交的描述信息" -a="作者" 容器 ID 要创建的目标镜像名:[标签名]

命令：在容器外执行，记得

docker commit -m="ifconfig cmd add" -a="zzyy" a69d7c825c4f zzyyubuntu:1.2

```
[root@zzyy ~]# docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS
      NAMES
a69d7c825c4f   ubuntu     "/bin/bash"              7 minutes ago    Up 7 minutes
      boring_lehmann
5c0ed56024f6   registry   "/entrypoint.sh /etc…"   14 minutes ago   Up 14 minutes   0.0.0.0:5000->5000/tcp,
tcp   inspiring_bardeen
[root@zzyy ~]#
[root@zzyy ~]# docker commit -m="ifconfig cmd add" -a="zzyy" a69d7c825c4f zzyyubuntu:1.2
sha256:d6ca70a4f9321e207585925507 5e0b2121b9a2ed1f08e8aac285f9515b03e2b4
[root@zzyy ~]#
[root@zzyy ~]# docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
zzyyubuntu    1.2       d6ca70a4f932   6 seconds ago    105MB
mysql         5.7       8a8a506ccfdc   2 days ago       448MB
ubuntu        latest    597ce1600cf4   2 weeks ago      72.8MB
hello-world   latest    feb5d9fea6a5   3 weeks ago      13.3kB
registry      latest    b2cb11db9d3d   6 weeks ago      26.2MB
redis         6.0.8     16ecd2772934   11 months ago    104MB
[root@zzyy ~]#
```

- 启动我们的新镜像并和原来的对比

1 官网是默认下载的 Ubuntu 没有 ifconfig 命令

2 我们自己 commit 构建的新镜像，新增加了 ifconfig 功能，可以成功使用。

```
[root@zzyy ~]# docker images
REPOSITORY    TAG       IMAGE ID       CREATED              SIZE
zzyyubuntu    1.2       d6ca70a4f932   About a minute ago   105MB
mysql         5.7       8a8a506ccfdc   2 days ago           448MB
ubuntu        latest    597ce1600cf4   2 weeks ago          72.8MB
hello-world   latest    feb5d9fea6a5   3 weeks ago          13.3kB
registry      latest    b2cb11db9d3d   6 weeks ago          26.2MB
redis         6.0.8     16ecd2772934   11 months ago        104MB
[root@zzyy ~]# docker run -it zzyyubuntu:1.2 /bin/bash
root@10c21a92ecc4:/# ifconfig
eth0:  flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.3  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:03  txqueuelen 0  (Ethernet)
        RX packets 5  bytes 418 (418.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo:  flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
```

- curl 验证私服库上有什么镜像

curl -XGET http://192.168.111.162:5000/v2/_catalog

可以看到，目前私服库没有任何镜像上传过。。。。。。

- 将新镜像 zzyyubuntu:1.2 修改符合私服规范的 Tag

按照公式： docker tag 镜像:Tag Host:Port/Repository:Tag

自己 host 主机 IP 地址，填写同学你们自己的，不要粘贴错误，O(∩_∩)O

使用命令 docker tag 将 zzyyubuntu:1.2 这个镜像修改为 192.168.111.162:5000/zzyyubuntu:1.2

docker tag zzyyubuntu:1.2 192.168.111.162:5000/zzyyubuntu:1.2

```
[root@zzyy ~]# docker tag  zzyyubuntu:1.2  192.168.111.162:5000/zzyyubuntu:1.2
[root@zzyy ~]# docker images
REPOSITORY                                TAG       IMAGE ID       CREATED          SIZE
192.168.111.162:5000/zzyyubuntu           1.2       d6ca70a4f932   9 minutes ago    105MB
zzyyubuntu                                1.2       d6ca70a4f932   9 minutes ago    105MB
mysql                                     5.7       8a8a506ccfdc   2 days ago       448MB
ubuntu                                    latest    597ce1600cf4   2 weeks ago      72.8MB
hello-world                               latest    feb5d9fea6a5   3 weeks ago      13.3kB
registry                                  latest    b2cb11db9d3d   6 weeks ago      26.2MB
redis                                     6.0.8     16ecd2772934   11 months ago    104MB
[root@zzyy ~]#
```

- 修改配置文件使之支持 http

```
[root@zzyy ~]# cat  /etc/docker/daemon.json
{
  "registry-mirrors": ["https://aa25jngu.mirror.aliyuncs.com"],
  "insecure-registries": ["192.168.111.162:5000"]
}
[root@zzyy ~]#
```

别无脑照着复制，registry-mirrors 配置的是国内阿里提供的镜像加速地址，不用加速的话访问官网的会很慢。

2 个配置中间有个逗号 ',' 别漏了，这个配置是 json 格式的。

2 个配置中间有个逗号 ',' 别漏了，这个配置是 json 格式的。

2 个配置中间有个逗号 ',' 别漏了，这个配置是 json 格式的。

vim 命令新增如下红色内容：vim /etc/docker/daemon.json

```
{
  "registry-mirrors":
["https://aa25jngu.mirror.aliyuncs.com"],
  "insecure-registries":
["192.168.111.162:5000"]
}
```

上述理由：docker 默认不允许 http 方式推送镜像，通过配置选项来取消这个限制。====> 修改完后如果不生效，建议重启 docker

- push 推送到私服库

docker push 192.168.111.162:5000/zzyyubuntu:1.2

```
[root@zzyy ~]# cat  /etc/docker/daemon.json
{
  "registry-mirrors": ["https://aa25jngu.mirror.aliyuncs.com"],
  "insecure-registries": ["192.168.111.162:5000"]
}
[root@zzyy ~]# docker push 192.168.111.162:5000/zzyyubuntu:1.2
The push refers to repository [192.168.111.162:5000/zzyyubuntu]
6f54f0889e14: Pushed
da55b45d310b: Pushed
1.2: digest: sha256:f48b33273b125219b674def76a385ebe020e3a3f6eabfed86c74e89fdf029f0a size: 741
[root@zzyy ~]# docker images
REPOSITORY                              TAG       IMAGE ID       CREATED          SIZE
zzyyubuntu                              1.2       d6ca70a4f932   18 minutes ago   105MB
192.168.111.162:5000/zzyyubuntu         1.2       d6ca70a4f932   18 minutes ago   105MB
mysql                                   5.7       8a8a506ccfdc   2 days ago       448MB
ubuntu                                  latest    597ce1600cf4   2 weeks ago      72.8MB
```

- curl 验证私服库上有什么镜像 2

curl -XGET http://192.168.111.162:5000/v2/_catalog

```
[root@zzyy ~]# docker images
REPOSITORY                              TAG       IMAGE ID       CREATED          SIZE
zzyyubuntu                              1.2       d6ca70a4f932   18 minutes ago   105MB
192.168.111.162:5000/zzyyubuntu         1.2       d6ca70a4f932   18 minutes ago   105MB
mysql                                   5.7       8a8a506ccfdc   2 days ago       448MB
ubuntu                                  latest    597ce1600cf4   2 weeks ago      72.8MB
hello-world                             latest    feb5d9fea6a5   3 weeks ago      13.3kB
registry                                latest    b2cb11db9d3d   6 weeks ago      26.2MB
redis                                   6.0.8     16ecd2772934   11 months ago    104MB
[root@zzyy ~]# curl -XGET http://192.168.111.162:5000/v2/_catalog
{"repositories":["zzyyubuntu"]}
[root@zzyy ~]#
```

- pull 到本地并运行

docker pull 192.168.111.162:5000/zzyyubuntu:1.2

```
[root@zzyy ~]# docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
mysql         5.7       8a8a506ccfdc   2 days ago      448MB
ubuntu        latest    597ce1600cf4   2 weeks ago     72.8MB
hello-world   latest    feb5d9fea6a5   3 weeks ago     13.3kB
registry      latest    b2cb11db9d3d   6 weeks ago     26.2MB
redis         6.0.8     16ecd2772934   11 months ago   104MB
[root@zzyy ~]#
[root@zzyy ~]# docker pull 192.168.111.162:5000/zzyyubuntu:1.2
1.2: Pulling from zzyyubuntu
f3ef4ff62e0d: Already exists
959fcc8bc6c0: Already exists
Digest: sha256:f48b33273b125219b674def76a385ebe020e3a3f6eabfed86c74e89fdf029f0a
Status: Downloaded newer image for 192.168.111.162:5000/zzyyubuntu:1.2
192.168.111.162:5000/zzyyubuntu:1.2
[root@zzyy ~]# docker images
REPOSITORY                           TAG       IMAGE ID       CREATED          SIZE
192.168.111.162:5000/zzyyubuntu      1.2       d6ca70a4f932   27 minutes ago   105MB
mysql                                5.7       8a8a506ccfdc   2 days ago       448MB
ubuntu                               latest    597ce1600cf4   2 weeks ago      72.8MB
```

docker run -it 镜像 ID /bin/bash

```
[root@zzyy ~]# docker images
REPOSITORY                           TAG       IMAGE ID       CREATED          SIZE
192.168.111.162:5000/zzyyubuntu      1.2       d6ca70a4f932   27 minutes ago   105MB
mysql                                5.7       8a8a506ccfdc   2 days ago       448MB
ubuntu                               latest    597ce1600cf4   2 weeks ago      72.8MB
hello-world                          latest    feb5d9fea6a5   3 weeks ago      13.3kB
registry                             latest    b2cb11db9d3d   6 weeks ago      26.2MB
redis                                6.0.8     16ecd2772934   11 months ago    104MB
[root@zzyy ~]# docker run -it d6ca70a4f932 /bin/bash
root@794207b92907:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.3  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:03  txqueuelen 0  (Ethernet)
```

1.7. Docker 容器数据卷

1.7.1. 坑：容器卷记得加入

单图标	CustomIcon-663735520

- --privileged=true
- why

Docker 挂载主机目录访问如果出现 cannot open directory .: Permission denied

解决办法：在挂载目录后多加一个--privileged=true 参数即可

如果是 CentOS7 安全模块会比之前系统版本加强，不安全的会先禁止，所以目录挂载的情况被默认为不安全的行为，
在 SELinux 里面挂载目录被禁止掉了额，如果要开启，我们一般使用--privileged=true 命令，扩大容器的权限解决挂载目录没有权限的问题，也即
使用该参数，container 内的 root 拥有真正的 root 权限，否则，container 内的 root 只是外部的一个普通用户权限。

1.7.2. 回顾下上一讲的知识点，参数 V

还记得蓝色框框中的内容吗？

1.7.3. 是什么

卷就是目录或文件，存在于一个或多个容器中，由 docker 挂载到容器，但不属于联合文件系统，因此能够绕过 Union File System 提供一些用于持续存储或共享数据的特性：
卷的设计目的就是数据的持久化，完全独立于容器的生存周期，因此 Docker 不会在容器删除时删除其挂载的数据卷

- 一句话：有点类似我们 Redis 里面的 rdb 和 aof 文件
- 将 docker 容器内的数据保存进宿主机的磁盘中

- 运行一个带有容器卷存储功能的容器实例
  - docker run -it --privileged=true -v /宿主机绝对路径目录:/容器内目录 镜像名

1.7.4. 能干嘛

* 将运用与运行的环境打包镜像，run 后形成容器实例运行 ，但是我们对数据的要求希望是持久化的

Docker 容器产生的数据，如果不备份，那么当容器实例删除后，容器内的数据自然也就没有了。
为了能保存数据在 docker 中我们使用卷。

特点：
1：数据卷可在容器之间共享或重用数据
2：卷中的更改可以直接实时生效，爽
3：数据卷中的更改不会包含在镜像的更新中
4：数据卷的生命周期一直持续到没有容器使用它为止

1.7.5. 数据卷案例

- 宿主 vs 容器之间映射添加容器卷
  - 直接命令添加
  - 命令

公式：docker run -it -v /宿主机目录:/容器内目录 ubuntu /bin/bash

```
docker run -it --name myu3 --privileged=true -v /tmp/myHostData:/tmp/myDockerData ubuntu /bin/bash
```

- docker run -it --privileged=true -v /宿主机绝对路径目录:/容器内目录 镜像名

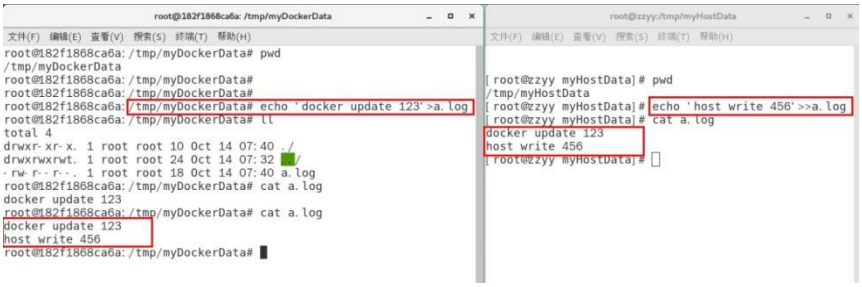
- 查看数据卷是否挂载成功

docker inspect 容器 ID

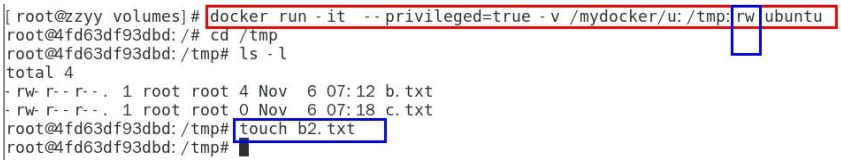
```
"Mounts": [
    {
        "Type": "bind",
        "Source": "/tmp/myHostData",
        "Destination": "/tmp/myDockerData",
        "Mode": "",
        "RW": true,
        "Propagation": "rprivate"
    }
],
```

- 容器和宿主机之间数据共享

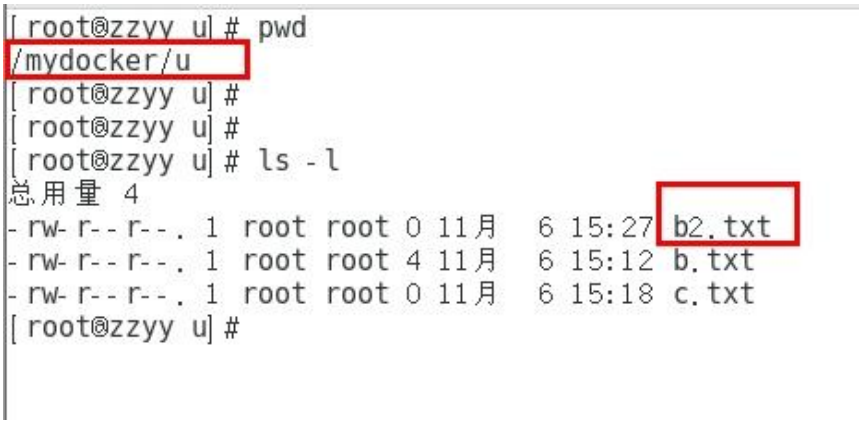
1 docker 修改，主机同步获得
2 主机修改，docker 同步获得
3 docker 容器 stop，主机修改，docker 容器重启看数据是否同步。



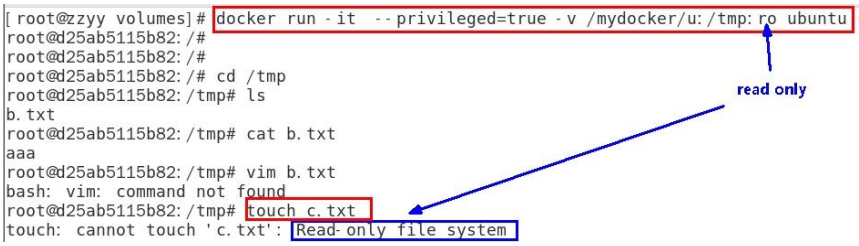
- 读写规则映射添加说明
  - 读写(默认)



rw = read + write



  - docker run -it --privileged=true -v /宿主机绝对路径目录:/容器内目录:rw 镜像名
  - 默认同上案例，默认就是 rw
  - 只读
  - 容器实例内部被限制，只能读取不能写



/容器目录:ro 镜像名　　就能完成功能，此时容器自己只能读取不能写

ro = read only

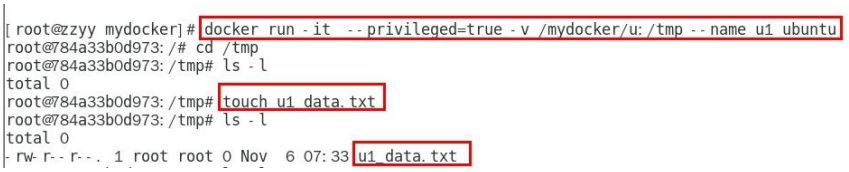
此时如果宿主机写入内容，可以同步给容器内，容器可以读取到。

- docker run -it --privileged=true -v /宿主机绝对路径目录:/容器内目录:ro 镜像名

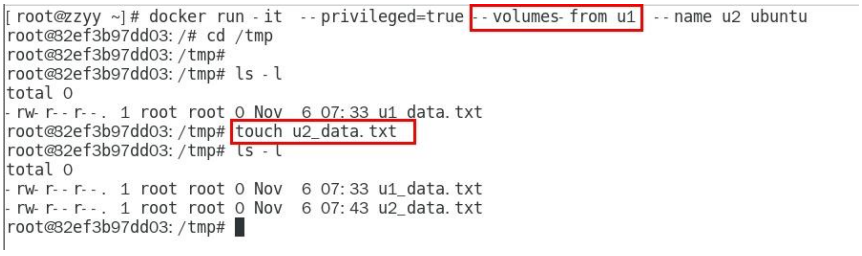
- 卷的继承和共享

  - 容器 1 完成和宿主机的映射

docker run -it --privileged=true -v /mydocker/u:/tmp --name u1 ubuntu



  - 容器 2 继承容器 1 的卷规则



- docker run -it --privileged=true --volumes-from 父类 --name u2 ubuntu

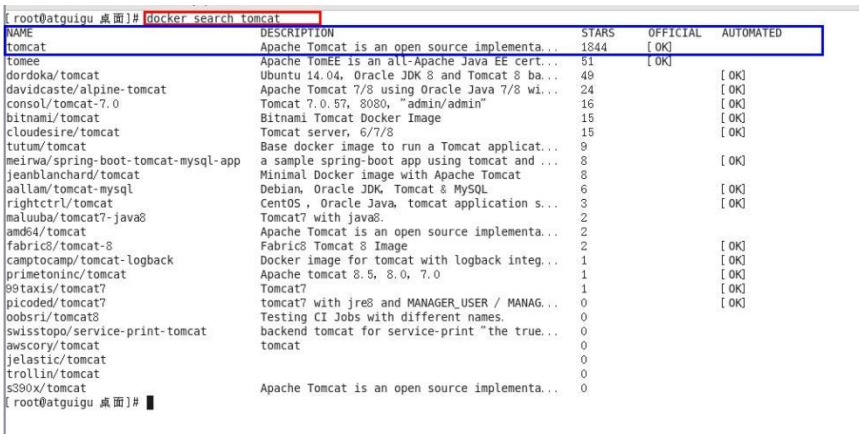
1.8. Docker 常规安装简介

1.8.1. 总体步骤

- 搜索镜像
- 拉取镜像
- 查看镜像
- 启动镜像
  - 服务端口映射
- 停止容器
- 移除容器

1.8.2. 安装 tomcat

- docker hub 上面查找 tomcat 镜像
  - docker search tomcat



```
[root@atguigu 桌面]# docker search tomcat
NAME                                DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
tomcat                              Apache Tomcat is an open source implementa...   1844      [OK]
tomee                               Apache TomEE is an all-Apache Java EE cert...   51        [OK]
dordoka/tomcat                      Ubuntu 14.04, Oracle JDK 8 and Tomcat 8 ba...   49                   [OK]
davidcaste/alpine-tomcat            Apache Tomcat 7/8 using Oracle Java 7/8 wi...   24                   [OK]
consol/tomcat-7.0                   Tomcat 7.0.57, 8080, "admin/admin"              16                   [OK]
bitnami/tomcat                      Bitnami Tomcat Docker Image                     15                   [OK]
cloudesire/tomcat                   Tomcat server, 6/7/8                            15                   [OK]
tutum/tomcat                        Base docker image to run a Tomcat applicat...   9
meirwa/spring-boot-tomcat-mysql-app a sample spring-boot app using tomcat and ...   8                    [OK]
jeanblanchard/tomcat                Minimal Docker image with Apache Tomcat         8
aallam/tomcat-mysql                 Debian, Oracle JDK, Tomcat & MySQL              6                    [OK]
rightctrl/tomcat                    CentOS , Oracle Java, tomcat application s...   3                    [OK]
maluuba/tomcat7-java8               Tomcat7 with java8.                             2
amd64/tomcat                        Apache Tomcat is an open source implementa...   2
fabric8/tomcat-8                    Fabric8 Tomcat 8 Image                          2                    [OK]
camptocamp/tomcat-logback           Docker image for tomcat with logback integ...   1                    [OK]
primetoninc/tomcat                  Apache tomcat 8.5, 8.0, 7.0                     1                    [OK]
99taxis/tomcat7                     Tomcat7                                         1                    [OK]
picoded/tomcat7                     tomcat7 with jre8 and MANAGER_USER / MANAG...   0                    [OK]
oobsri/tomcat8                      Testing CI Jobs with different names.           0
swisstopo/service-print-tomcat      backend tomcat for service-print "the true...   0
awscory/tomcat                      tomcat                                          0
jelastic/tomcat                                                                     0
trollin/tomcat                                                                      0
s390x/tomcat                        Apache Tomcat is an open source implementa...   0
[root@atguigu 桌面]#
```

- 从 docker hub 上拉取 tomcat 镜像到本地
  - docker pull tomcat

```
[root@atguigu docker]# docker pull tomcat
latest: Pulling from tomcat

841c6d57cd5e: Pulling fs layer
4f5904b5ae9a: Pulling fs layer
ce84c5fb1edc: Pulling fs layer
08e49d2ba62c: Pulling fs layer
6d59a7a1e69a: Pull complete
8069d49c4094: Pull complete
37874ba173f4: Pull complete
b8d64e7efa17: Pull complete
217bbfb8e851: Pull complete
9d56e04171d7: Pull complete
d092f10455ea: Pull complete
6db0adc8ce3c: Pull complete
c96fcb778526: Downloading [==============================>                    ] 72.44 MB/122.1 MB
b1f139ba4dd2: Download complete
8834a7ffc401: Download complete
995ada5f35d2: Download complete
ec6ce6f03e11: Download complete
4bac12a809fc: Download complete
0b043134e0bc: Download complete
0e7992fbc673: Download complete
0b7a981495f3: Download complete
2630c6ab33a1: Download complete
845f8c1e3527: Download complete
21b55be4aa8a: Download complete
ac5033d93a00: Download complete
0871e859937d: Pull complete
de532f3545fe: Pull complete
5d829bb22671: Pull complete
cb315192e016: Pull complete
Digest: sha256:c3d5ae0f207fcef755289fb98a8cd7a3272def86983c3d0de1b6502818ace3ee
Status: Downloaded newer image for tomcat:latest
[root@atguigu docker]#
```

- docker images 查看是否有拉取到的 tomcat

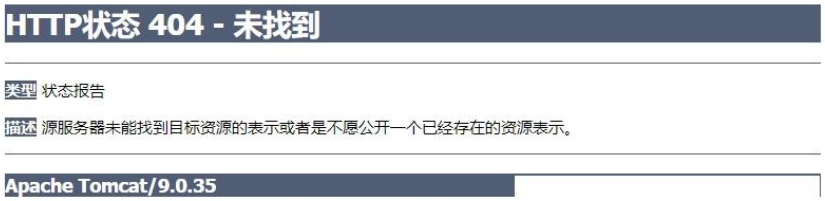
```
[root@zzyy u]# docker images tomcat
REPOSITORY   TAG       IMAGE ID       CREATED      SIZE
tomcat       latest    904a98253fbf   7 days ago   680MB
[root@zzyy u]#
```

- 使用 tomcat 镜像创建容器实例(也叫运行镜像)
  - docker run -it -p 8080:8080 tomcat
  - -p 小写，主机端口:docker 容器端口
  - -P 大写，随机分配端口

  - i:交互
  - t:终端
  - d:后台
- 访问猫首页

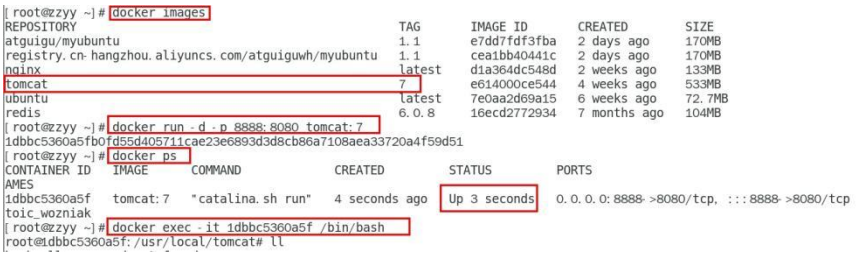
- 问题

最后访问报404的情况：



- 解决
- 可能没有映射端口或者没有关闭防火墙

- 把 webapps.dist 目录换成 webapps
- 先成功启动 tomcat



- 查看 webapps 文件夹查看为空

```
root@1dbbc5360a5f:/usr/local/tomcat# ls -l
total 188
-rw-r--r--. 1 root root 17849 Apr 22 18:45 BUILDING.txt
-rw-r--r--. 1 root root  5586 Apr 22 18:45 CONTRIBUTING.md
-rw-r--r--. 1 root root 56846 Apr 22 18:45 LICENSE
-rw-r--r--. 1 root root  1281 Apr 22 18:45 NOTICE
-rw-r--r--. 1 root root  3257 Apr 22 18:45 README.md
-rw-r--r--. 1 root root  9362 Apr 22 18:45 RELEASE-NOTES
-rw-r--r--. 1 root root 17251 Apr 22 18:45 RUNNING.txt
drwxr-xr-x. 2 root root  4096 May 13 05:36 bin
drwxr-xr-x. 1 root root  4096 Jun 11 02:15 conf
drwxr-xr-x. 2 root root  4096 May 13 05:36 lib
drwxrwxrwx. 1 root root  4096 Jun 11 02:15 logs
drwxr-xr-x. 2 root root  4096 May 13 05:36 native-jni-lib
drwxrwxrwx. 2 root root  4096 May 13 05:36 temp
drwxr-xr-x. 2 root root  4096 May 13 05:36 webapps
drwxr-xr-x. 7 root root  4096 Apr 22 18:45 webapps.dist
drwxrwxrwx. 2 root root  4096 Apr 22 18:43 work
root@1dbbc5360a5f:/usr/local/tomcat# rm -r webapps
root@1dbbc5360a5f:/usr/local/tomcat# mv webapps.dist webapps
root@1dbbc5360a5f:/usr/local/tomcat#
```

- 免修改版说明
  - docker pull billygoo/tomcat8-jdk8
  - docker run -d -p 8080:8080 --name mytomcat8 billygoo/tomcat8-jdk8

```
[root@zzyy ~]# docker run -d -p 8080:8080 --name mytomcat8 billygoo/tomcat8-jdk8
Unable to find image 'billygoo/tomcat8-jdk8:latest' locally
latest: Pulling from billygoo/tomcat8-jdk8
55cbf04beb70: Pull complete
1607093a898c: Pull complete
9a8ea045c926: Pull complete
1290813abd9d: Pull complete
8a6b982ad6d7: Pull complete
abb029e68402: Pull complete
8cd067dc06dc: Pull complete
1b9ce2097b98: Pull complete
d6db5874b692: Pull complete
25b4aa3d52c5: Pull complete
d26b86f009c9: Pull complete
e54998e5e699: Pull complete
4a1e415a3c2e: Pull complete
Digest: sha256:4e21f52d29e3a0baafc18979da2f9725449b54652db69d4cdaef9ba807097e11
Status: Downloaded newer image for billygoo/tomcat8-jdk8:latest
a41c6ed4d18f6c4f6296e16a4a88ba3307e938f0b21bc6c0dfa2e736a0ab2d61
[root@zzyy ~]#
```

1.8.3. 安装 mysql

- docker hub 上面查找 mysql 镜像

```
[root@atguigu 桌面]# docker search mysql
NAME                                                DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
mysql                                               MySQL is a widely used, open-source relati...   6373      [OK]
mariadb                                             MariaDB is a community-developed fork of M...   2003      [OK]
mysql/mysql-server                                  Optimized MySQL Server Docker images. Crea...   459                  [OK]
percona                                             Percona Server is a fork of the MySQL rela...   343       [OK]
zabbix/zabbix-server-mysql                          Zabbix Server with MySQL database support       102                  [OK]
hypriot/rpi-mysql                                   RPi-compatible Docker Image with Mysql          87
centurylink/mysql                                   Image containing mysql. Optimized to be li...   60                   [OK]
zabbix/zabbix-web-nginx-mysql                       Zabbix frontend based on Nginx web-server ...   54                   [OK]
1and1internet/ubuntu-16-nginx-php-phpmyadmin-mysql-5   ubuntu-16-nginx-php-phpmyadmin-mysql-5       36                   [OK]
tutum/mysql                                         Base docker image to run a MySQL database ...   32
centos/mysql-57-centos7                             MySQL 5.7 SQL database server                   28
mysql/mysql-cluster                                 Experimental MySQL Cluster Docker images. ...   27
schickling/mysql-backup-s3                          Backup MySQL to S3 (supports periodic back...   19                   [OK]
bitnami/mysql                                       Bitnami MySQL Docker Image                      15                   [OK]
linuxserver/mysql                                   A Mysql container, brought to you by Linux...   14
zabbix/zabbix-proxy-mysql                           Zabbix proxy with MySQL database support        13                   [OK]
```

- 从 docker hub 上(阿里云加速器)拉取 mysql 镜像到本地标签为 5.7

```
[root@zzyy conf]# docker pull mysql:5.7
5.7: Pulling from library/mysql
b380bbd43752: Already exists
f23cbf2ecc5d: Already exists
30cfc6c29c0a: Already exists
b38609286cbe: Already exists
8211d9e66cd6: Already exists
2313f9eeca4a: Already exists
7eb487d00da0: Already exists
bb9cc5c700e7: Already exists
88676eb32344: Already exists
8fea0b38a348: Already exists
3dc585bfc693: Already exists
Digest: sha256:b8814059bbd9c80b78fe4b2b0b70cd70fe3772b3c5d8ee1edfa46791db3224f9
Status: Downloaded newer image for mysql:5.7
docker.io/library/mysql:5.7
[root@zzyy conf]#
```

- 使用 mysql5.7 镜像创建容器(也叫运行镜像)
  - 命令出处，哪里来的?

  - 简单版
  - 使用 mysql 镜像

docker run -p 3306:3306 -e MYSQL_ROOT_PASSWORD=123456 -d mysql:5.7
docker ps
docker exec -it 容器 ID /bin/bash
mysql -uroot -p

```
[root@zzyy /]# docker run -p 3306:3306 -e MYSQL_ROOT_PASSWORD=123456 -d mysql:5.7
b82f14a1750c933cfe4f7d81e542eca49fc680d92a58846dc9bc2be6e1a37763
[root@zzyy /]#
[root@zzyy /]# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS
                NAMES
b82f14a1750c   mysql:5.7   "docker-entrypoint.s…"   34 seconds ago   Up 34 seconds   0.0.0.0:3306->3306/tcp, :::3306->3306
/tcp, 33060/tcp   upbeat_engelbart
[root@zzyy /]# docker exec -it b82f14a1750c /bin/bash
root@b82f14a1750c:/#
root@b82f14a1750c:/# mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.35 MySQL Community Server (GPL)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
```

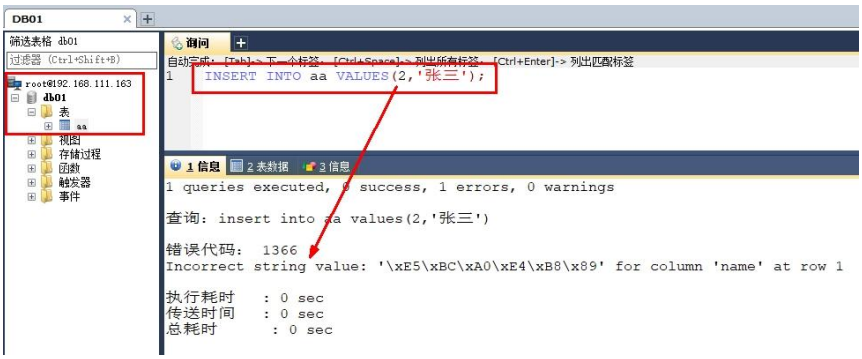
  - 建库建表插入数据

```
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
mysql> create database db01;
Query OK, 1 row affected (0.00 sec)

mysql> use db01;
Database changed
```

```
mysql> create table aa(id int,name varchar(20));
Query OK, 0 rows affected (0.02 sec)

mysql> insert into aa values(1,'z3');
Query OK, 1 row affected (0.03 sec)

mysql> select * from aa;
+------+------+
| id   | name |
+------+------+
|    1 | z3   |
+------+------+
1 row in set (0.00 sec)

mysql>
```

- 外部 Win10 也来连接运行在 dokcer 上的 mysql 容器实例服务

- 问题
- 插入中文数据试试



- 为什么报错?
- docker 上默认字符集编码隐患

docker 里面的 mysql 容器实例查看，内容如下：

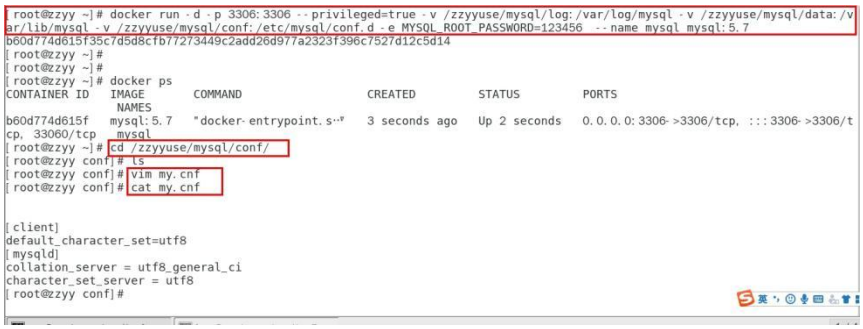
SHOW VARIABLES LIKE 'character%'

```
mysql> SHOW VARIABLES LIKE 'character%';
+--------------------------+----------------------------+
| Variable_name            | Value                      |
+--------------------------+----------------------------+
| character_set_client     | latin1                     |
| character_set_connection | latin1                     |
| character_set_database   | latin1                     |
| character_set_filesystem | binary                     |
| character_set_results    | latin1                     |
| character_set_server     | latin1                     |
| character_set_system     | utf8                       |
| character_sets_dir       | /usr/share/mysql/charsets/ |
+--------------------------+----------------------------+
8 rows in set (0.00 sec)

mysql>
```

- 删除容器后，里面的 mysql 数据如何办
- 容器实例一删除，你还有什么？ 删容器到跑路。。。。。？
- 实战版
- 新建 mysql 容器实例

```
docker run -d -p 3306:3306 --privileged=true -v
/zzyyuse/mysql/log:/var/log/mysql -v
/zzyyuse/mysql/data:/var/lib/mysql -v
/zzyyuse/mysql/conf:/etc/mysql/conf.d -e
```

```
MYSQL_ROOT_PASSWORD=123456  --name
mysql mysql:5.7
```



- 新建 my.cnf
- 通过容器卷同步给 mysql 容器实例

```
[client]
default_character_set=utf8
[mysqld]
collation_server = utf8_general_ci
character_set_server = utf8
```

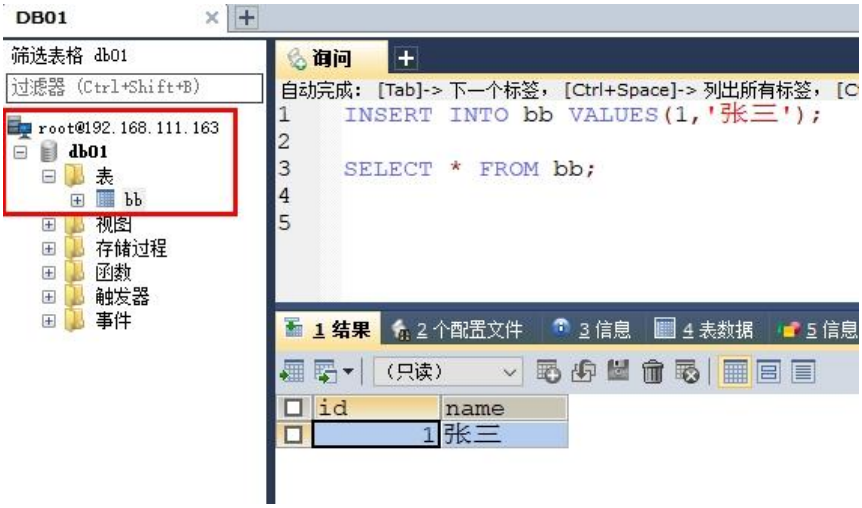
```
[root@zzyy ~]# cd /zzyyuse/mysql/conf/
[root@zzyy conf]# ls
[root@zzyy conf]# vim my.cnf
[root@zzyy conf]# cat my.cnf


[client]
default_character_set=utf8
[mysqld]
collation_server = utf8_general_ci
character_set_server = utf8
[root@zzyy conf]#
```

- 重新启动 mysql 容器实例再重新进入并查看字符编码

```
[root@zzyy conf]# docker restart mysql
mysql
[root@zzyy conf]#
[root@zzyy conf]#
[root@zzyy conf]#
[root@zzyy conf]#
[root@zzyy conf]#
[root@zzyy conf]#
[root@zzyy conf]# docker exec -it mysql bash
root@b60d774d615f:/# mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.35 MySQL Community Server (GPL)

mysql> SHOW VARIABLES LIKE 'character%';
+--------------------------+----------------------------+
| Variable_name            | Value                      |
+--------------------------+----------------------------+
| character_set_client     | utf8                       |
| character_set_connection | utf8                       |
| character_set_database   | utf8                       |
| character_set_filesystem | binary                     |
| character_set_results    | utf8                       |
| character_set_server     | utf8                       |
| character_set_system     | utf8                       |
| character_sets_dir       | /usr/share/mysql/charsets/ |
+--------------------------+----------------------------+
8 rows in set (0.00 sec)
```

- 再新建库新建表再插入中文测试

```
mysql> create database db01;
Query OK, 1 row affected (0.00 sec)

mysql> use db01;
Database changed

mysql> create table bb(id int,name varchar(20));
Query OK, 0 rows affected (0.02 sec)
```



· 结论

之前的 DB 无效

修改字符集操作+重启 mysql 容器实例

之后的 DB 有效，需要新建

结论：docker 安装完 MySQL 并 run 出容器后，建议请先修改完字符集编码后再新建 mysql 库-表-插数据

```
+--------------------------+----------------------------+
| Variable_name            | Value                      |
+--------------------------+----------------------------+
| character_set_client     | utf8                       |
| character_set_connection | utf8                       |
| character_set_database   | utf8                       |
| character_set_filesystem | binary                     |
| character_set_results    | utf8                       |
| character_set_server     | utf8                       |
| character_set_system     | utf8                       |
| character_sets_dir       | /usr/share/mysql/charsets/ |
+--------------------------+----------------------------+
8 rows in set (0.00 sec)

mysql> create database db02;
Query OK, 1 row affected (0.00 sec)

mysql> use db02;
Database changed
mysql> create table t1(id int ,name varchar(20));
Query OK, 0 rows affected (0.01 sec)
```

- 假如将当前容器实例删除，再重新来一次，之前建的 db01 实例还有吗?
  trytry

1.8.4. 安装 redis

- 从 docker hub 上(阿里云加速器)拉取 redis 镜像到本地标签为 6.0.8

```
[root@zzyy ~]# docker pull redis:6.0.8
Trying to pull repository docker.io/library/redis ...
6.0.8: Pulling from docker.io/library/redis
Digest: sha256:21db12e5ab3cc343e9376d655e8eabbdbe5516801373e95a8a9e66010c5b8819
Status: Image is up to date for docker.io/redis:6.0.8
[root@zzyy ~]#
[root@zzyy ~]#
[root@zzyy ~]# docker images
REPOSITORY                    TAG                 IMAGE ID            CREATED             SIZE
docker.io/rabbitmq            3.8.9-management    92c743b6c055        4 months ago        198 MB
docker.io/redislabs/rebloom   latest              24a7020676a0        5 months ago        146 MB
docker.io/redis               6.0.8               16ecd2772934        6 months ago        104 MB
docker.io/hello-world         latest              bf756fb1ae65        16 months ago       13.3 kB
[root@zzyy ~]#
```

- 入门命令

```
[root@zzyy docker]# docker run -d -p 6379:6379 redis:6.0.8
f216c38ea035fcefb5d54245956f9f29750ae2edd423450a0c59d12b448724f9
[root@zzyy docker]#
[root@zzyy docker]# docker ps
CONTAINER ID   IMAGE         COMMAND                  CREATED         STATUS         PORTS
      NAMES
f216c38ea035   redis:6.0.8   "docker-entrypoint.s…"   6 seconds ago   Up 4 seconds   0.0.0.0:6379->6379/tcp, :::6379->6379
/tcp   agitated_pascal
[root@zzyy docker]#
[root@zzyy docker]# docker exec -it f216c38ea035 /bin/bash
root@f216c38ea035:/data# redis-cli
127.0.0.1:6379> set k1 v1
OK
127.0.0.1:6379> get k1
"v1"
127.0.0.1:6379> ping
PONG
127.0.0.1:6379>
```

- **命令提醒：容器卷记得加入**--privileged=true

Docker 挂载主机目录 Docker 访问出现 cannot open directory .: Permission denied
解决办法：在挂载目录后多加一个--privileged=true 参数即可

- 在 CentOS 宿主机下新建目录/app/redis

```
[root@zzyy ~]# mkdir -p /app/redis/
[root@zzyy ~]# cd /app
[root@zzyy app]# cd redis
[root@zzyy redis]# pwd
/app/redis
[root@zzyy redis]#
```

1 建目录

mkdir -p /app/redis

- mkdir -p /app/redis

- 将一个 redis.conf 文件模板拷贝进/app/redis 目录下

```
[root@zzyy /]# mkdir -p /app/redis
[root@zzyy /]#
[root@zzyy /]# cp /myredis/redis.conf /app/redis/
[root@zzyy /]#
[root@zzyy /]# cd /app/redis/
[root@zzyy redis]#
[root@zzyy redis]# ls -l
总用量 84
-rw-r--r--. 1 root root 84672 5月  16 16:21 redis.conf
[root@zzyy redis]#
```

2 拷贝配置文件

将准备好的 redis.conf 文件放进/app/redis 目录下

- /app/redis 目录下修改 redis.conf 文件

3 /app/redis 目录下修改 redis.conf 文件

3.1 开启 redis 验证　可选

requirepass 123

3.2 允许 redis 外地连接　必须

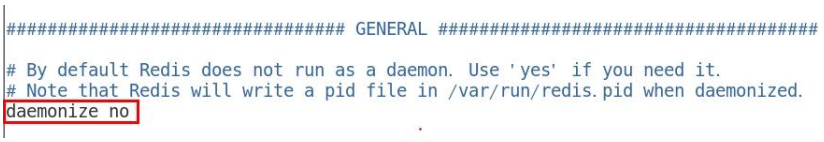
注释掉 # bind 127.0.0.1

```
#
# IF YOU ARE SURE YOU WANT YOUR INSTANCE TO LISTEN TO ALL THE INTERFACES
# JUST COMMENT THE FOLLOWING LINE.
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# bind 127.0.0.1
```

允许redis外地连接

3.3 daemonize no

将 daemonize yes 注释起来或者 daemonize no 设置，因为该配置和 docker run 中-d 参数冲突，会导致容器一直启动失败



3.4 开启 redis 数据持久化 appendonly yes 可选

- 默认出厂的原始 redis.conf

```
# Redis configuration file example.
#
# Note that in order to read the configuration file, Redis must be
# started with the file path as first argument:
#
# ./redis-server /path/to/redis.conf

# Note on units: when memory size is needed, it is possible to specify
# it in the usual form of 1k 5GB 4M and so forth:
#
# 1k => 1000 bytes
# 1kb => 1024 bytes
# 1m => 1000000 bytes
# 1mb => 1024*1024 bytes
# 1g => 1000000000 bytes
# 1gb => 1024*1024*1024 bytes
#
# units are case insensitive so 1GB 1Gb 1gB are all the same.

################################## INCLUDES
###################################

# Include one or more other config files here.  This is useful if you
# have a standard template that goes to all Redis servers but also need
# to customize a few per-server settings.  Include files can include
# other files, so use this wisely.
#
# Notice option "include" won't be rewritten by command "CONFIG
REWRITE"
# from admin or Redis Sentinel. Since Redis always uses the last
processed
# line as value of a configuration directive, you'd better put includes
# at the beginning of this file to avoid overwriting config change at
runtime.
```

```
#
# If instead you are interested in using includes to override configuration
# options, it is better to use include as the last line.
#
# include /path/to/local.conf
# include /path/to/other.conf

################################## MODULES
#####################################

# Load modules at startup. If the server is not able to load modules
# it will abort. It is possible to use multiple loadmodule directives.
#
# loadmodule /path/to/my_module.so
# loadmodule /path/to/other_module.so

################################## NETWORK
#####################################

# By default, if no "bind" configuration directive is specified, Redis
listens
# for connections from all the network interfaces available on the server.
# It is possible to listen to just one or multiple selected interfaces using
# the "bind" configuration directive, followed by one or more IP
addresses.
#
# Examples:
#
# bind 192.168.1.100 10.0.0.1
# bind 127.0.0.1 ::1
#
# ~~~ WARNING ~~~ If the computer running Redis is directly exposed
to the
# internet, binding to all the interfaces is dangerous and will expose the
# instance to everybody on the internet. So by default we uncomment
the
# following bind directive, that will force Redis to listen only into
# the IPv4 loopback interface address (this means Redis will be able to
# accept connections only from clients running into the same computer it
# is running).
#
# IF YOU ARE SURE YOU WANT YOUR INSTANCE TO LISTEN TO
ALL THE INTERFACES
# JUST COMMENT THE FOLLOWING LINE.
#
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~
#bind 127.0.0.1

# Protected mode is a layer of security protection, in order to avoid that
# Redis instances left open on the internet are accessed and exploited.
#
# When protected mode is on and if:
#
# 1) The server is not binding explicitly to a set of addresses using the
#    "bind" directive.
```

```
# 2) No password is configured.
#
# The server only accepts connections from clients connecting from the
# IPv4 and IPv6 loopback addresses 127.0.0.1 and ::1, and from Unix
domain
# sockets.
#
# By default protected mode is enabled. You should disable it only if
# you are sure you want clients from other hosts to connect to Redis
# even if no authentication is configured, nor a specific set of interfaces
# are explicitly listed using the "bind" directive.
protected-mode no

# Accept connections on the specified port, default is 6379 (IANA
#815344).
# If port 0 is specified Redis will not listen on a TCP socket.
port 6379

# TCP listen() backlog.
#
# In high requests-per-second environments you need an high backlog
in order
# to avoid slow clients connections issues. Note that the Linux kernel
# will silently truncate it to the value of /proc/sys/net/core/somaxconn so
# make sure to raise both the value of somaxconn and
tcp_max_syn_backlog
# in order to get the desired effect.
tcp-backlog 511

# Unix socket.
#
# Specify the path for the Unix socket that will be used to listen for
# incoming connections. There is no default, so Redis will not listen
# on a unix socket when not specified.
#
# unixsocket /tmp/redis.sock
# unixsocketperm 700

# Close the connection after a client is idle for N seconds (0 to disable)
timeout 0

# TCP keepalive.
#
# If non-zero, use SO_KEEPALIVE to send TCP ACKs to clients in
absence
# of communication. This is useful for two reasons:
#
# 1) Detect dead peers.
# 2) Take the connection alive from the point of view of network
#    equipment in the middle.
#
# On Linux, the specified value (in seconds) is the period used to send
ACKs.
# Note that to close the connection the double of the time is needed.
# On other kernels the period depends on the kernel configuration.
#
```

```
# A reasonable value for this option is 300 seconds, which is the new
# Redis default starting with Redis 3.2.1.
tcp-keepalive 300

################################## GENERAL
#####################################

# By default Redis does not run as a daemon. Use 'yes' if you need it.
# Note that Redis will write a pid file in /var/run/redis.pid when
daemonized.
daemonize no

# If you run Redis from upstart or systemd, Redis can interact with your
# supervision tree. Options:
#   supervised no      - no supervision interaction
#   supervised upstart - signal upstart by putting Redis into SIGSTOP
mode
#   supervised systemd - signal systemd by writing READY=1 to
$NOTIFY_SOCKET
#   supervised auto    - detect upstart or systemd method based on
#                        UPSTART_JOB or NOTIFY_SOCKET environment
variables
# Note: these supervision methods only signal "process is ready."
#       They do not enable continuous liveness pings back to your
supervisor.
supervised no

# If a pid file is specified, Redis writes it where specified at startup
# and removes it at exit.
#
# When the server runs non daemonized, no pid file is created if none is
# specified in the configuration. When the server is daemonized, the pid
file
# is used even if not specified, defaulting to "/var/run/redis.pid".
#
# Creating a pid file is best effort: if Redis is not able to create it
# nothing bad happens, the server will start and run normally.
pidfile /var/run/redis_6379.pid

# Specify the server verbosity level.
# This can be one of:
# debug (a lot of information, useful for development/testing)
# verbose (many rarely useful info, but not a mess like the debug level)
# notice (moderately verbose, what you want in production probably)
# warning (only very important / critical messages are logged)
loglevel notice

# Specify the log file name. Also the empty string can be used to force
# Redis to log on the standard output. Note that if you use standard
# output for logging but daemonize, logs will be sent to /dev/null
logfile ""

# To enable logging to the system logger, just set 'syslog-enabled' to
yes,
# and optionally update the other syslog parameters to suit your needs.
# syslog-enabled no
```

```
# Specify the syslog identity.
# syslog-ident redis

# Specify the syslog facility. Must be USER or between LOCAL0-
LOCAL7.
# syslog-facility local0

# Set the number of databases. The default database is DB 0, you can
select
# a different one on a per-connection basis using SELECT <dbid>
where
# dbid is a number between 0 and 'databases'-1
databases 16

# By default Redis shows an ASCII art logo only when started to log to
the
# standard output and if the standard output is a TTY. Basically this
means
# that normally a logo is displayed only in interactive sessions.
#
# However it is possible to force the pre-4.0 behavior and always show
a
# ASCII art logo in startup logs by setting the following option to yes.
always-show-logo yes

################################
SNAPSHOTTING  ################################
#
# Save the DB on disk:
#
#   save <seconds> <changes>
#
#   Will save the DB if both the given number of seconds and the given
#   number of write operations against the DB occurred.
#
#   In the example below the behaviour will be to save:
#   after 900 sec (15 min) if at least 1 key changed
#   after 300 sec (5 min) if at least 10 keys changed
#   after 60 sec if at least 10000 keys changed
#
#   Note: you can disable saving completely by commenting out all
"save" lines.
#
#   It is also possible to remove all the previously configured save
#   points by adding a save directive with a single empty string argument
#   like in the following example:
#
#   save ""

save 900 1
save 300 10
save 60 10000

# By default Redis will stop accepting writes if RDB snapshots are
enabled
```

```
# (at least one save point) and the latest background save failed.
# This will make the user aware (in a hard way) that data is not
persisting
# on disk properly, otherwise chances are that no one will notice and
some
# disaster will happen.
#
# If the background saving process will start working again Redis will
# automatically allow writes again.
#
# However if you have setup your proper monitoring of the Redis server
# and persistence, you may want to disable this feature so that Redis
will
# continue to work as usual even if there are problems with disk,
# permissions, and so forth.
stop-writes-on-bgsave-error yes

# Compress string objects using LZF when dump .rdb databases?
# For default that's set to 'yes' as it's almost always a win.
# If you want to save some CPU in the saving child set it to 'no' but
# the dataset will likely be bigger if you have compressible values or
keys.
rdbcompression yes

# Since version 5 of RDB a CRC64 checksum is placed at the end of
the file.
# This makes the format more resistant to corruption but there is a
performance
# hit to pay (around 10%) when saving and loading RDB files, so you
can disable it
# for maximum performances.
#
# RDB files created with checksum disabled have a checksum of zero
that will
# tell the loading code to skip the check.
rdbchecksum yes

# The filename where to dump the DB
dbfilename dump.rdb

# The working directory.
#
# The DB will be written inside this directory, with the filename specified
# above using the 'dbfilename' configuration directive.
#
# The Append Only File will also be created inside this directory.
#
# Note that you must specify a directory here, not a file name.
dir ./

################################# REPLICATION
#################################

# Master-Replica replication. Use replicaof to make a Redis instance a
copy of
```

```
# another Redis server. A few things to understand ASAP about Redis
replication.
#
#   +------------------+      +---------------+
#   |      Master      | ---> |    Replica    |
#   | (receive writes) |      |  (exact copy) |
#   +------------------+      +---------------+
#
# 1) Redis replication is asynchronous, but you can configure a master
to
#    stop accepting writes if it appears to be not connected with at least
#    a given number of replicas.
# 2) Redis replicas are able to perform a partial resynchronization with
the
#    master if the replication link is lost for a relatively small amount of
#    time. You may want to configure the replication backlog size (see
the next
#    sections of this file) with a sensible value depending on your needs.
# 3) Replication is automatic and does not need user intervention. After
a
#    network partition replicas automatically try to reconnect to masters
#    and resynchronize with them.
#
# replicaof <masterip> <masterport>

# If the master is password protected (using the "requirepass"
configuration
# directive below) it is possible to tell the replica to authenticate before
# starting the replication synchronization process, otherwise the master
will
# refuse the replica request.
#
# masterauth <master-password>

# When a replica loses its connection with the master, or when the
replication
# is still in progress, the replica can act in two different ways:
#
# 1) if replica-serve-stale-data is set to 'yes' (the default) the replica will
#    still reply to client requests, possibly with out of date data, or the
#    data set may just be empty if this is the first synchronization.
#
# 2) if replica-serve-stale-data is set to 'no' the replica will reply with
#    an error "SYNC with master in progress" to all the kind of commands
#    but to INFO, replicaOF, AUTH, PING, SHUTDOWN, REPLCONF,
ROLE, CONFIG,
#    SUBSCRIBE, UNSUBSCRIBE, PSUBSCRIBE, PUNSUBSCRIBE,
PUBLISH, PUBSUB,
#    COMMAND, POST, HOST: and LATENCY.
#
replica-serve-stale-data yes

# You can configure a replica instance to accept writes or not. Writing
against
# a replica instance may be useful to store some ephemeral data
(because data
```

```
# written on a replica will be easily deleted after resync with the master)
but
# may also cause problems if clients are writing to it because of a
# misconfiguration.
#
# Since Redis 2.6 by default replicas are read-only.
#
# Note: read only replicas are not designed to be exposed to untrusted
clients
# on the internet. It's just a protection layer against misuse of the
instance.
# Still a read only replica exports by default all the administrative
commands
# such as CONFIG, DEBUG, and so forth. To a limited extent you can
improve
# security of read only replicas using 'rename-command' to shadow all
the
# administrative / dangerous commands.
replica-read-only yes

# Replication SYNC strategy: disk or socket.
#
# -------------------------------------------------------
# WARNING: DISKLESS REPLICATION IS EXPERIMENTAL
CURRENTLY
# -------------------------------------------------------
#
# New replicas and reconnecting replicas that are not able to continue
the replication
# process just receiving differences, need to do what is called a "full
# synchronization". An RDB file is transmitted from the master to the
replicas.
# The transmission can happen in two different ways:
#
# 1) Disk-backed: The Redis master creates a new process that writes
the RDB
#                 file on disk. Later the file is transferred by the parent
#                 process to the replicas incrementally.
# 2) Diskless: The Redis master creates a new process that directly
writes the
#              RDB file to replica sockets, without touching the disk at all.
#
# With disk-backed replication, while the RDB file is generated, more
replicas
# can be queued and served with the RDB file as soon as the current
child producing
# the RDB file finishes its work. With diskless replication instead once
# the transfer starts, new replicas arriving will be queued and a new
transfer
# will start when the current one terminates.
#
# When diskless replication is used, the master waits a configurable
amount of
# time (in seconds) before starting the transfer in the hope that multiple
replicas
# will arrive and the transfer can be parallelized.
```

```
#
# With slow disks and fast (large bandwidth) networks, diskless
replication
# works better.
repl-diskless-sync no

# When diskless replication is enabled, it is possible to configure the
delay
# the server waits in order to spawn the child that transfers the RDB via
socket
# to the replicas.
#
# This is important since once the transfer starts, it is not possible to
serve
# new replicas arriving, that will be queued for the next RDB transfer, so
the server
# waits a delay in order to let more replicas arrive.
#
# The delay is specified in seconds, and by default is 5 seconds. To
disable
# it entirely just set it to 0 seconds and the transfer will start ASAP.
repl-diskless-sync-delay 5

# Replicas send PINGs to server in a predefined interval. It's possible to
change
# this interval with the repl_ping_replica_period option. The default
value is 10
# seconds.
#
# repl-ping-replica-period 10

# The following option sets the replication timeout for:
#
# 1) Bulk transfer I/O during SYNC, from the point of view of replica.
# 2) Master timeout from the point of view of replicas (data, pings).
# 3) Replica timeout from the point of view of masters (REPLCONF ACK
pings).
#
# It is important to make sure that this value is greater than the value
# specified for repl-ping-replica-period otherwise a timeout will be
detected
# every time there is low traffic between the master and the replica.
#
# repl-timeout 60

# Disable TCP_NODELAY on the replica socket after SYNC?
#
# If you select "yes" Redis will use a smaller number of TCP packets
and
# less bandwidth to send data to replicas. But this can add a delay for
# the data to appear on the replica side, up to 40 milliseconds with
# Linux kernels using a default configuration.
#
# If you select "no" the delay for data to appear on the replica side will
# be reduced but more bandwidth will be used for replication.
#
```

```
# By default we optimize for low latency, but in very high traffic
conditions
# or when the master and replicas are many hops away, turning this to
"yes" may
# be a good idea.
repl-disable-tcp-nodelay no

# Set the replication backlog size. The backlog is a buffer that
accumulates
# replica data when replicas are disconnected for some time, so that
when a replica
# wants to reconnect again, often a full resync is not needed, but a
partial
# resync is enough, just passing the portion of data the replica missed
while
# disconnected.
#
# The bigger the replication backlog, the longer the time the replica can
be
# disconnected and later be able to perform a partial resynchronization.
#
# The backlog is only allocated once there is at least a replica
connected.
#
# repl-backlog-size 1mb

# After a master has no longer connected replicas for some time, the
backlog
# will be freed. The following option configures the amount of seconds
that
# need to elapse, starting from the time the last replica disconnected, for
# the backlog buffer to be freed.
#
# Note that replicas never free the backlog for timeout, since they may
be
# promoted to masters later, and should be able to correctly "partially
# resynchronize" with the replicas: hence they should always
accumulate backlog.
#
# A value of 0 means to never release the backlog.
#
# repl-backlog-ttl 3600

# The replica priority is an integer number published by Redis in the
INFO output.
# It is used by Redis Sentinel in order to select a replica to promote into
a
# master if the master is no longer working correctly.
#
# A replica with a low priority number is considered better for promotion,
so
# for instance if there are three replicas with priority 10, 100, 25 Sentinel
will
# pick the one with priority 10, that is the lowest.
#
```

```
# However a special priority of 0 marks the replica as not able to
perform the
# role of master, so a replica with priority of 0 will never be selected by
# Redis Sentinel for promotion.
#
# By default the priority is 100.
replica-priority 100

# It is possible for a master to stop accepting writes if there are less than
# N replicas connected, having a lag less or equal than M seconds.
#
# The N replicas need to be in "online" state.
#
# The lag in seconds, that must be <= the specified value, is calculated
from
# the last ping received from the replica, that is usually sent every
second.
#
# This option does not GUARANTEE that N replicas will accept the write,
but
# will limit the window of exposure for lost writes in case not enough
replicas
# are available, to the specified number of seconds.
#
# For example to require at least 3 replicas with a lag <= 10 seconds
use:
#
# min-replicas-to-write 3
# min-replicas-max-lag 10
#
# Setting one or the other to 0 disables the feature.
#
# By default min-replicas-to-write is set to 0 (feature disabled) and
# min-replicas-max-lag is set to 10.

# A Redis master is able to list the address and port of the attached
# replicas in different ways. For example the "INFO replication" section
# offers this information, which is used, among other tools, by
# Redis Sentinel in order to discover replica instances.
# Another place where this info is available is in the output of the
# "ROLE" command of a master.
#
# The listed IP and address normally reported by a replica is obtained
# in the following way:
#
#   IP: The address is auto detected by checking the peer address
#   of the socket used by the replica to connect with the master.
#
#   Port: The port is communicated by the replica during the replication
#   handshake, and is normally the port that the replica is using to
#   listen for connections.
#
# However when port forwarding or Network Address Translation (NAT)
is
# used, the replica may be actually reachable via different IP and port
# pairs. The following two options can be used by a replica in order to
```

```
# report to its master a specific set of IP and port, so that both INFO
# and ROLE will report those values.
#
# There is no need to use both the options if you need to override just
# the port or the IP address.
#
# replica-announce-ip 5.5.5.5
# replica-announce-port 1234

################################## SECURITY
###################################

# Require clients to issue AUTH <PASSWORD> before processing any
other
# commands.  This might be useful in environments in which you do not
trust
# others with access to the host running redis-server.
#
# This should stay commented out for backward compatibility and
because most
# people do not need auth (e.g. they run their own servers).
#
# Warning: since Redis is pretty fast an outside user can try up to
# 150k passwords per second against a good box. This means that you
should
# use a very strong password otherwise it will be very easy to break.
#
# requirepass foobared

# Command renaming.
#
# It is possible to change the name of dangerous commands in a shared
# environment. For instance the CONFIG command may be renamed
into something
# hard to guess so that it will still be available for internal-use tools
# but not available for general clients.
#
# Example:
#
# rename-command CONFIG
b840fc02d524045429941cc15f59e41cb7be6c52
#
# It is also possible to completely kill a command by renaming it into
# an empty string:
#
# rename-command CONFIG ""
#
# Please note that changing the name of commands that are logged into
the
# AOF file or transmitted to replicas may cause problems.

################################### CLIENTS
####################################

# Set the max number of connected clients at the same time. By default
# this limit is set to 10000 clients, however if the Redis server is not
```

```
# able to configure the process file limit to allow for the specified limit
# the max number of allowed clients is set to the current file limit
# minus 32 (as Redis reserves a few file descriptors for internal uses).
#
# Once the limit is reached Redis will close all the new connections
sending
# an error 'max number of clients reached'.
#
# maxclients 10000

############################## MEMORY MANAGEMENT
################################

# Set a memory usage limit to the specified amount of bytes.
# When the memory limit is reached Redis will try to remove keys
# according to the eviction policy selected (see maxmemory-policy).
#
# If Redis can't remove keys according to the policy, or if the policy is
# set to 'noeviction', Redis will start to reply with errors to commands
# that would use more memory, like SET, LPUSH, and so on, and will
continue
# to reply to read-only commands like GET.
#
# This option is usually useful when using Redis as an LRU or LFU
cache, or to
# set a hard memory limit for an instance (using the 'noeviction' policy).
#
# WARNING: If you have replicas attached to an instance with
maxmemory on,
# the size of the output buffers needed to feed the replicas are
subtracted
# from the used memory count, so that network problems / resyncs will
# not trigger a loop where keys are evicted, and in turn the output
# buffer of replicas is full with DELs of keys evicted triggering the
deletion
# of more keys, and so forth until the database is completely emptied.
#
# In short... if you have replicas attached it is suggested that you set a
lower
# limit for maxmemory so that there is some free RAM on the system for
replica
# output buffers (but this is not needed if the policy is 'noeviction').
#
# maxmemory <bytes>

# MAXMEMORY POLICY: how Redis will select what to remove when
maxmemory
# is reached. You can select among five behaviors:
#
# volatile-lru -> Evict using approximated LRU among the keys with an
expire set.
# allkeys-lru -> Evict any key using approximated LRU.
# volatile-lfu -> Evict using approximated LFU among the keys with an
expire set.
# allkeys-lfu -> Evict any key using approximated LFU.
```

```
# volatile-random -> Remove a random key among the ones with an
expire set.
# allkeys-random -> Remove a random key, any key.
# volatile-ttl -> Remove the key with the nearest expire time (minor TTL)
# noeviction -> Don't evict anything, just return an error on write
operations.
#
# LRU means Least Recently Used
# LFU means Least Frequently Used
#
# Both LRU, LFU and volatile-ttl are implemented using approximated
# randomized algorithms.
#
# Note: with any of the above policies, Redis will return an error on write
#       operations, when there are no suitable keys for eviction.
#
#       At the date of writing these commands are: set setnx setex
append
#       incr decr rpush lpush rpushx lpushx linsert lset rpoplpush sadd
#       sinter sinterstore sunion sunionstore sdiff sdiffstore zadd zincrby
#       zunionstore zinterstore hset hsetnx hmset hincrby incrby decrby
#       getset mset msetnx exec sort
#
# The default is:
#
# maxmemory-policy noeviction

# LRU, LFU and minimal TTL algorithms are not precise algorithms but
approximated
# algorithms (in order to save memory), so you can tune it for speed or
# accuracy. For default Redis will check five keys and pick the one that
was
# used less recently, you can change the sample size using the
following
# configuration directive.
#
# The default of 5 produces good enough results. 10 Approximates very
closely
# true LRU but costs more CPU. 3 is faster but not very accurate.
#
# maxmemory-samples 5

# Starting from Redis 5, by default a replica will ignore its maxmemory
setting
# (unless it is promoted to master after a failover or manually). It means
# that the eviction of keys will be just handled by the master, sending
the
# DEL commands to the replica as keys evict in the master side.
#
# This behavior ensures that masters and replicas stay consistent, and
is usually
# what you want, however if your replica is writable, or you want the
replica to have
# a different memory setting, and you are sure all the writes performed
to the
```

```
# replica are idempotent, then you may change this default (but be sure
to understand
# what you are doing).
#
# Note that since the replica by default does not evict, it may end using
more
# memory than the one set via maxmemory (there are certain buffers
that may
# be larger on the replica, or data structures may sometimes take more
memory and so
# forth). So make sure you monitor your replicas and make sure they
have enough
# memory to never hit a real out-of-memory condition before the master
hits
# the configured maxmemory setting.
#
# replica-ignore-maxmemory yes

############################## LAZY FREEING
####################################

# Redis has two primitives to delete keys. One is called DEL and is a
blocking
# deletion of the object. It means that the server stops processing new
commands
# in order to reclaim all the memory associated with an object in a
synchronous
# way. If the key deleted is associated with a small object, the time
needed
# in order to execute the DEL command is very small and comparable to
most other
# O(1) or O(log_N) commands in Redis. However if the key is
associated with an
# aggregated value containing millions of elements, the server can block
for
# a long time (even seconds) in order to complete the operation.
#
# For the above reasons Redis also offers non blocking deletion
primitives
# such as UNLINK (non blocking DEL) and the ASYNC option of
FLUSHALL and
# FLUSHDB commands, in order to reclaim memory in background.
Those commands
# are executed in constant time. Another thread will incrementally free
the
# object in the background as fast as possible.
#
# DEL, UNLINK and ASYNC option of FLUSHALL and FLUSHDB are
user-controlled.
# It's up to the design of the application to understand when it is a good
# idea to use one or the other. However the Redis server sometimes
has to
# delete keys or flush the whole database as a side effect of other
operations.
# Specifically Redis deletes objects independently of a user call in the
# following scenarios:
```

```
#
# 1) On eviction, because of the maxmemory and maxmemory policy
configurations,
#    in order to make room for new data, without going over the specified
#    memory limit.
# 2) Because of expire: when a key with an associated time to live (see
the
#    EXPIRE command) must be deleted from memory.
# 3) Because of a side effect of a command that stores data on a key
that may
#    already exist. For example the RENAME command may delete the
old key
#    content when it is replaced with another one. Similarly
SUNIONSTORE
#    or SORT with STORE option may delete existing keys. The SET
command
#    itself removes any old content of the specified key in order to
replace
#    it with the specified string.
# 4) During replication, when a replica performs a full resynchronization
with
#    its master, the content of the whole database is removed in order to
#    load the RDB file just transferred.
#
# In all the above cases the default is to delete objects in a blocking way,
# like if DEL was called. However you can configure each case
specifically
# in order to instead release memory in a non-blocking way like if
UNLINK
# was called, using the following configuration directives:

lazyfree-lazy-eviction no
lazyfree-lazy-expire no
lazyfree-lazy-server-del no
replica-lazy-flush no

############################## APPEND ONLY MODE
###############################

# By default Redis asynchronously dumps the dataset on disk. This
mode is
# good enough in many applications, but an issue with the Redis
process or
# a power outage may result into a few minutes of writes lost
(depending on
# the configured save points).
#
# The Append Only File is an alternative persistence mode that provides
# much better durability. For instance using the default data fsync policy
# (see later in the config file) Redis can lose just one second of writes in
a
# dramatic event like a server power outage, or a single write if
something
# wrong with the Redis process itself happens, but the operating system
is
# still running correctly.
```

```
#
# AOF and RDB persistence can be enabled at the same time without
problems.
# If the AOF is enabled on startup Redis will load the AOF, that is the
file
# with the better durability guarantees.
#
# Please check http://redis.io/topics/persistence for more information.

appendonly no

# The name of the append only file (default: "appendonly.aof")

appendfilename "appendonly.aof"

# The fsync() call tells the Operating System to actually write data on
disk
# instead of waiting for more data in the output buffer. Some OS will
really flush
# data on disk, some other OS will just try to do it ASAP.
#
# Redis supports three different modes:
#
# no: don't fsync, just let the OS flush the data when it wants. Faster.
# always: fsync after every write to the append only log. Slow, Safest.
# everysec: fsync only one time every second. Compromise.
#
# The default is "everysec", as that's usually the right compromise
between
# speed and data safety. It's up to you to understand if you can relax
this to
# "no" that will let the operating system flush the output buffer when
# it wants, for better performances (but if you can live with the idea of
# some data loss consider the default persistence mode that's
snapshotting),
# or on the contrary, use "always" that's very slow but a bit safer than
# everysec.
#
# More details please check the following article:
# http://antirez.com/post/redis-persistence-demystified.html
#
# If unsure, use "everysec".

# appendfsync always
appendfsync everysec
# appendfsync no

# When the AOF fsync policy is set to always or everysec, and a
background
# saving process (a background save or AOF log background rewriting)
is
# performing a lot of I/O against the disk, in some Linux configurations
# Redis may block too long on the fsync() call. Note that there is no fix
for
# this currently, as even performing fsync in a different thread will block
# our synchronous write(2) call.
```

```
#
# In order to mitigate this problem it's possible to use the following
option
# that will prevent fsync() from being called in the main process while a
# BGSAVE or BGREWRITEAOF is in progress.
#
# This means that while another child is saving, the durability of Redis is
# the same as "appendfsync none". In practical terms, this means that it
is
# possible to lose up to 30 seconds of log in the worst scenario (with the
# default Linux settings).
#
# If you have latency problems turn this to "yes". Otherwise leave it as
# "no" that is the safest pick from the point of view of durability.

no-appendfsync-on-rewrite no

# Automatic rewrite of the append only file.
# Redis is able to automatically rewrite the log file implicitly calling
# BGREWRITEAOF when the AOF log size grows by the specified
percentage.
#
# This is how it works: Redis remembers the size of the AOF file after
the
# latest rewrite (if no rewrite has happened since the restart, the size of
# the AOF at startup is used).
#
# This base size is compared to the current size. If the current size is
# bigger than the specified percentage, the rewrite is triggered. Also
# you need to specify a minimal size for the AOF file to be rewritten, this
# is useful to avoid rewriting the AOF file even if the percentage
increase
# is reached but it is still pretty small.
#
# Specify a percentage of zero in order to disable the automatic AOF
# rewrite feature.

auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb

# An AOF file may be found to be truncated at the end during the Redis
# startup process, when the AOF data gets loaded back into memory.
# This may happen when the system where Redis is running
# crashes, especially when an ext4 filesystem is mounted without the
# data=ordered option (however this can't happen when Redis itself
# crashes or aborts but the operating system still works correctly).
#
# Redis can either exit with an error when this happens, or load as much
# data as possible (the default now) and start if the AOF file is found
# to be truncated at the end. The following option controls this behavior.
#
# If aof-load-truncated is set to yes, a truncated AOF file is loaded and
# the Redis server starts emitting a log to inform the user of the event.
# Otherwise if the option is set to no, the server aborts with an error
# and refuses to start. When the option is set to no, the user requires
# to fix the AOF file using the "redis-check-aof" utility before to restart
```

```
# the server.
#
# Note that if the AOF file will be found to be corrupted in the middle
# the server will still exit with an error. This option only applies when
# Redis will try to read more data from the AOF file but not enough
bytes
# will be found.
aof-load-truncated yes

# When rewriting the AOF file, Redis is able to use an RDB preamble in
the
# AOF file for faster rewrites and recoveries. When this option is turned
# on the rewritten AOF file is composed of two different stanzas:
#
#   [RDB file][AOF tail]
#
# When loading Redis recognizes that the AOF file starts with the
"REDIS"
# string and loads the prefixed RDB file, and continues loading the AOF
# tail.
aof-use-rdb-preamble yes

################################ LUA
SCRIPTING  ###############################

# Max execution time of a Lua script in milliseconds.
#
# If the maximum execution time is reached Redis will log that a script is
# still in execution after the maximum allowed time and will start to
# reply to queries with an error.
#
# When a long running script exceeds the maximum execution time only
the
# SCRIPT KILL and SHUTDOWN NOSAVE commands are available.
The first can be
# used to stop a script that did not yet called write commands. The
second
# is the only way to shut down the server in the case a write command
was
# already issued by the script but the user doesn't want to wait for the
natural
# termination of the script.
#
# Set it to 0 or a negative value for unlimited execution without warnings.
lua-time-limit 5000

################################ REDIS
CLUSTER  ###############################

# Normal Redis instances can't be part of a Redis Cluster; only nodes
that are
# started as cluster nodes can. In order to start a Redis instance as a
# cluster node enable the cluster support uncommenting the following:
#
# cluster-enabled yes
```

```
# Every cluster node has a cluster configuration file. This file is not
# intended to be edited by hand. It is created and updated by Redis
nodes.
# Every Redis Cluster node requires a different cluster configuration file.
# Make sure that instances running in the same system do not have
# overlapping cluster configuration file names.
#
# cluster-config-file nodes-6379.conf

# Cluster node timeout is the amount of milliseconds a node must be
unreachable
# for it to be considered in failure state.
# Most other internal time limits are multiple of the node timeout.
#
# cluster-node-timeout 15000

# A replica of a failing master will avoid to start a failover if its data
# looks too old.
#
# There is no simple way for a replica to actually have an exact measure
of
# its "data age", so the following two checks are performed:
#
# 1) If there are multiple replicas able to failover, they exchange
messages
#    in order to try to give an advantage to the replica with the best
#    replication offset (more data from the master processed).
#    Replicas will try to get their rank by offset, and apply to the start
#    of the failover a delay proportional to their rank.
#
# 2) Every single replica computes the time of the last interaction with
#    its master. This can be the last ping or command received (if the
master
#    is still in the "connected" state), or the time that elapsed since the
#    disconnection with the master (if the replication link is currently
down).
#    If the last interaction is too old, the replica will not try to failover
#    at all.
#
# The point "2" can be tuned by user. Specifically a replica will not
perform
# the failover if, since the last interaction with the master, the time
# elapsed is greater than:
#
#   (node-timeout * replica-validity-factor) + repl-ping-replica-period
#
# So for example if node-timeout is 30 seconds, and the replica-validity-
factor
# is 10, and assuming a default repl-ping-replica-period of 10 seconds,
the
# replica will not try to failover if it was not able to talk with the master
# for longer than 310 seconds.
#
# A large replica-validity-factor may allow replicas with too old data to
failover
```

```
# a master, while a too small value may prevent the cluster from being
able to
# elect a replica at all.
#
# For maximum availability, it is possible to set the replica-validity-factor
# to a value of 0, which means, that replicas will always try to failover
the
# master regardless of the last time they interacted with the master.
# (However they'll always try to apply a delay proportional to their
# offset rank).
#
# Zero is the only value able to guarantee that when all the partitions
heal
# the cluster will always be able to continue.
#
# cluster-replica-validity-factor 10

# Cluster replicas are able to migrate to orphaned masters, that are
masters
# that are left without working replicas. This improves the cluster ability
# to resist to failures as otherwise an orphaned master can't be failed
over
# in case of failure if it has no working replicas.
#
# Replicas migrate to orphaned masters only if there are still at least a
# given number of other working replicas for their old master. This
number
# is the "migration barrier". A migration barrier of 1 means that a replica
# will migrate only if there is at least 1 other working replica for its
master
# and so forth. It usually reflects the number of replicas you want for
every
# master in your cluster.
#
# Default is 1 (replicas migrate only if their masters remain with at least
# one replica). To disable migration just set it to a very large value.
# A value of 0 can be set but is useful only for debugging and
dangerous
# in production.
#
# cluster-migration-barrier 1

# By default Redis Cluster nodes stop accepting queries if they detect
there
# is at least an hash slot uncovered (no available node is serving it).
# This way if the cluster is partially down (for example a range of hash
slots
# are no longer covered) all the cluster becomes, eventually,
unavailable.
# It automatically returns available as soon as all the slots are covered
again.
#
# However sometimes you want the subset of the cluster which is
working,
# to continue to accept queries for the part of the key space that is still
# covered. In order to do so, just set the cluster-require-full-coverage
```

```
# option to no.
#
# cluster-require-full-coverage yes

# This option, when set to yes, prevents replicas from trying to failover
its
# master during master failures. However the master can still perform a
# manual failover, if forced to do so.
#
# This is useful in different scenarios, especially in the case of multiple
# data center operations, where we want one side to never be promoted
if not
# in the case of a total DC failure.
#
# cluster-replica-no-failover no

# In order to setup your cluster make sure to read the documentation
# available at http://redis.io web site.

########################## CLUSTER DOCKER/NAT
support  ########################

# In certain deployments, Redis Cluster nodes address discovery fails,
because
# addresses are NAT-ted or because ports are forwarded (the typical
case is
# Docker and other containers).
#
# In order to make Redis Cluster working in such environments, a static
# configuration where each node knows its public address is needed.
The
# following two options are used for this scope, and are:
#
# * cluster-announce-ip
# * cluster-announce-port
# * cluster-announce-bus-port
#
# Each instruct the node about its address, client port, and cluster
message
# bus port. The information is then published in the header of the bus
packets
# so that other nodes will be able to correctly map the address of the
node
# publishing the information.
#
# If the above options are not used, the normal Redis Cluster auto-
detection
# will be used instead.
#
# Note that when remapped, the bus port may not be at the fixed offset
of
# clients port + 10000, so you can specify any port and bus-port
depending
# on how they get remapped. If the bus-port is not set, a fixed offset of
# 10000 will be used as usually.
#
```

```
# Example:
#
# cluster-announce-ip 10.1.1.5
# cluster-announce-port 6379
# cluster-announce-bus-port 6380

################################## SLOW LOG
###################################

# The Redis Slow Log is a system to log queries that exceeded a
specified
# execution time. The execution time does not include the I/O
operations
# like talking with the client, sending the reply and so forth,
# but just the time needed to actually execute the command (this is the
only
# stage of command execution where the thread is blocked and can not
serve
# other requests in the meantime).
#
# You can configure the slow log with two parameters: one tells Redis
# what is the execution time, in microseconds, to exceed in order for the
# command to get logged, and the other parameter is the length of the
# slow log. When a new command is logged the oldest one is removed
from the
# queue of logged commands.

# The following time is expressed in microseconds, so 1000000 is
equivalent
# to one second. Note that a negative number disables the slow log,
while
# a value of zero forces the logging of every command.
slowlog-log-slower-than 10000

# There is no limit to this length. Just be aware that it will consume
memory.
# You can reclaim memory used by the slow log with SLOWLOG
RESET.
slowlog-max-len 128

################################ LATENCY MONITOR
##############################

# The Redis latency monitoring subsystem samples different operations
# at runtime in order to collect data related to possible sources of
# latency of a Redis instance.
#
# Via the LATENCY command this information is available to the user
that can
# print graphs and obtain reports.
#
# The system only logs operations that were performed in a time equal
or
# greater than the amount of milliseconds specified via the
# latency-monitor-threshold configuration directive. When its value is set
# to zero, the latency monitor is turned off.
```

```
#
# By default latency monitoring is disabled since it is mostly not needed
# if you don't have latency issues, and collecting data has a
performance
# impact, that while very small, can be measured under big load.
Latency
# monitoring can easily be enabled at runtime using the command
# "CONFIG SET latency-monitor-threshold <milliseconds>" if needed.
latency-monitor-threshold 0

############################## EVENT NOTIFICATION
##############################

# Redis can notify Pub/Sub clients about events happening in the key
space.
# This feature is documented at http://redis.io/topics/notifications
#
# For instance if keyspace events notification is enabled, and a client
# performs a DEL operation on key "foo" stored in the Database 0, two
# messages will be published via Pub/Sub:
#
# PUBLISH __keyspace@0__:foo del
# PUBLISH __keyevent@0__:del foo
#
# It is possible to select the events that Redis will notify among a set
# of classes. Every class is identified by a single character:
#
#  K     Keyspace events, published with __keyspace@<db>__ prefix.
#  E     Keyevent events, published with __keyevent@<db>__ prefix.
#  g     Generic commands (non-type specific) like DEL, EXPIRE,
RENAME, ...
#  $     String commands
#  l     List commands
#  s     Set commands
#  h     Hash commands
#  z     Sorted set commands
#  x     Expired events (events generated every time a key expires)
#  e     Evicted events (events generated when a key is evicted for
maxmemory)
#  A     Alias for g$lshzxe, so that the "AKE" string means all the events.
#
#  The "notify-keyspace-events" takes as argument a string that is
composed
#  of zero or multiple characters. The empty string means that
notifications
#  are disabled.
#
#  Example: to enable list and generic events, from the point of view of
the
#           event name, use:
#
#  notify-keyspace-events Elg
#
#  Example 2: to get the stream of the expired keys subscribing to
channel
#             name __keyevent@0__:expired use:
```

```
#
  notify-keyspace-events Ex
#
#  By default all notifications are disabled because most users don't
need
#  this feature and the feature has some overhead. Note that if you don't
#  specify at least one of K or E, no events will be delivered.
#notify-keyspace-events ""

############################### ADVANCED CONFIG
###############################

# Hashes are encoded using a memory efficient data structure when
they have a
# small number of entries, and the biggest entry does not exceed a
given
# threshold. These thresholds can be configured using the following
directives.
hash-max-ziplist-entries 512
hash-max-ziplist-value 64

# Lists are also encoded in a special way to save a lot of space.
# The number of entries allowed per internal list node can be specified
# as a fixed maximum size or a maximum number of elements.
# For a fixed maximum size, use -5 through -1, meaning:
# -5: max size: 64 Kb  <-- not recommended for normal workloads
# -4: max size: 32 Kb  <-- not recommended
# -3: max size: 16 Kb  <-- probably not recommended
# -2: max size: 8 Kb   <-- good
# -1: max size: 4 Kb   <-- good
# Positive numbers mean store up to _exactly_ that number of elements
# per list node.
# The highest performing option is usually -2 (8 Kb size) or -1 (4 Kb
size),
# but if your use case is unique, adjust the settings as necessary.
list-max-ziplist-size -2

# Lists may also be compressed.
# Compress depth is the number of quicklist ziplist nodes from *each*
side of
# the list to *exclude* from compression.  The head and tail of the list
# are always uncompressed for fast push/pop operations.  Settings are:
# 0: disable all list compression
# 1: depth 1 means "don't start compressing until after 1 node into the
list,
#    going from either the head or tail"
#    So: [head]->node->node->...->node->[tail]
#    [head], [tail] will always be uncompressed; inner nodes will
compress.
# 2: [head]->[next]->node->node->...->node->[prev]->[tail]
#    2 here means: don't compress head or head->next or tail->prev or
tail,
#    but compress all nodes between them.
# 3: [head]->[next]->[next]->node->node->...->node->[prev]->[prev]->[tail]
# etc.
list-compress-depth 0
```

```
# Sets have a special encoding in just one case: when a set is
composed
# of just strings that happen to be integers in radix 10 in the range
# of 64 bit signed integers.
# The following configuration setting sets the limit in the size of the
# set in order to use this special memory saving encoding.
set-max-intset-entries 512

# Similarly to hashes and lists, sorted sets are also specially encoded in
# order to save a lot of space. This encoding is only used when the
length and
# elements of a sorted set are below the following limits:
zset-max-ziplist-entries 128
zset-max-ziplist-value 64

# HyperLogLog sparse representation bytes limit. The limit includes the
# 16 bytes header. When an HyperLogLog using the sparse
representation crosses
# this limit, it is converted into the dense representation.
#
# A value greater than 16000 is totally useless, since at that point the
# dense representation is more memory efficient.
#
# The suggested value is ~ 3000 in order to have the benefits of
# the space efficient encoding without slowing down too much PFADD,
# which is O(N) with the sparse encoding. The value can be raised to
# ~ 10000 when CPU is not a concern, but space is, and the data set is
# composed of many HyperLogLogs with cardinality in the 0 - 15000
range.
hll-sparse-max-bytes 3000

# Streams macro node max size / items. The stream data structure is a
radix
# tree of big nodes that encode multiple items inside. Using this
configuration
# it is possible to configure how big a single node can be in bytes, and
the
# maximum number of items it may contain before switching to a new
node when
# appending new stream entries. If any of the following settings are set
to
# zero, the limit is ignored, so for instance it is possible to set just a
# max entires limit by setting max-bytes to 0 and max-entries to the
desired
# value.
stream-node-max-bytes 4096
stream-node-max-entries 100

# Active rehashing uses 1 millisecond every 100 milliseconds of CPU
time in
# order to help rehashing the main Redis hash table (the one mapping
top-level
# keys to values). The hash table implementation Redis uses (see dict.c)
# performs a lazy rehashing: the more operation you run into a hash
table
```

```
# that is rehashing, the more rehashing "steps" are performed, so if the
# server is idle the rehashing is never complete and some more memory
is used
# by the hash table.
#
# The default is to use this millisecond 10 times every second in order to
# actively rehash the main dictionaries, freeing memory when possible.
#
# If unsure:
# use "activerehashing no" if you have hard latency requirements and it
is
# not a good thing in your environment that Redis can reply from time to
time
# to queries with 2 milliseconds delay.
#
# use "activerehashing yes" if you don't have such hard requirements
but
# want to free memory asap when possible.
activerehashing yes

# The client output buffer limits can be used to force disconnection of
clients
# that are not reading data from the server fast enough for some reason
(a
# common reason is that a Pub/Sub client can't consume messages as
fast as the
# publisher can produce them).
#
# The limit can be set differently for the three different classes of clients:
#
# normal -> normal clients including MONITOR clients
# replica  -> replica clients
# pubsub -> clients subscribed to at least one pubsub channel or pattern
#
# The syntax of every client-output-buffer-limit directive is the following:
#
# client-output-buffer-limit <class> <hard limit> <soft limit> <soft
seconds>
#
# A client is immediately disconnected once the hard limit is reached, or
if
# the soft limit is reached and remains reached for the specified number
of
# seconds (continuously).
# So for instance if the hard limit is 32 megabytes and the soft limit is
# 16 megabytes / 10 seconds, the client will get disconnected
immediately
# if the size of the output buffers reach 32 megabytes, but will also get
# disconnected if the client reaches 16 megabytes and continuously
overcomes
# the limit for 10 seconds.
#
# By default normal clients are not limited because they don't receive
data
# without asking (in a push way), but just after a request, so only
```

```
# asynchronous clients may create a scenario where data is requested
faster
# than it can read.
#
# Instead there is a default limit for pubsub and replica clients, since
# subscribers and replicas receive data in a push fashion.
#
# Both the hard or the soft limit can be disabled by setting them to zero.
client-output-buffer-limit normal 0 0 0
client-output-buffer-limit replica 256mb 64mb 60
client-output-buffer-limit pubsub 32mb 8mb 60

# Client query buffers accumulate new commands. They are limited to a
fixed
# amount by default in order to avoid that a protocol desynchronization
(for
# instance due to a bug in the client) will lead to unbound memory
usage in
# the query buffer. However you can configure it here if you have very
special
# needs, such us huge multi/exec requests or alike.
#
# client-query-buffer-limit 1gb

# In the Redis protocol, bulk requests, that are, elements representing
single
# strings, are normally limited ot 512 mb. However you can change this
limit
# here.
#
# proto-max-bulk-len 512mb

# Redis calls an internal function to perform many background tasks,
like
# closing connections of clients in timeout, purging expired keys that are
# never requested, and so forth.
#
# Not all tasks are performed with the same frequency, but Redis
checks for
# tasks to perform according to the specified "hz" value.
#
# By default "hz" is set to 10. Raising the value will use more CPU when
# Redis is idle, but at the same time will make Redis more responsive
when
# there are many keys expiring at the same time, and timeouts may be
# handled with more precision.
#
# The range is between 1 and 500, however a value over 100 is usually
not
# a good idea. Most users should use the default of 10 and raise this up
to
# 100 only in environments where very low latency is required.
hz 10

# Normally it is useful to have an HZ value which is proportional to the
# number of clients connected. This is useful in order, for instance, to
```

```
# avoid too many clients are processed for each background task
invocation
# in order to avoid latency spikes.
#
# Since the default HZ value by default is conservatively set to 10, Redis
# offers, and enables by default, the ability to use an adaptive HZ value
# which will temporary raise when there are many connected clients.
#
# When dynamic HZ is enabled, the actual configured HZ will be used
as
# as a baseline, but multiples of the configured HZ value will be actually
# used as needed once more clients are connected. In this way an idle
# instance will use very little CPU time while a busy instance will be
# more responsive.
dynamic-hz yes

# When a child rewrites the AOF file, if the following option is enabled
# the file will be fsync-ed every 32 MB of data generated. This is useful
# in order to commit the file to the disk more incrementally and avoid
# big latency spikes.
aof-rewrite-incremental-fsync yes

# When redis saves RDB file, if the following option is enabled
# the file will be fsync-ed every 32 MB of data generated. This is useful
# in order to commit the file to the disk more incrementally and avoid
# big latency spikes.
rdb-save-incremental-fsync yes

# Redis LFU eviction (see maxmemory setting) can be tuned. However
it is a good
# idea to start with the default settings and only change them after
investigating
# how to improve the performances and how the keys LFU change over
time, which
# is possible to inspect via the OBJECT FREQ command.
#
# There are two tunable parameters in the Redis LFU implementation:
the
# counter logarithm factor and the counter decay time. It is important to
# understand what the two parameters mean before changing them.
#
# The LFU counter is just 8 bits per key, it's maximum value is 255, so
Redis
# uses a probabilistic increment with logarithmic behavior. Given the
value
# of the old counter, when a key is accessed, the counter is incremented
in
# this way:
#
# 1. A random number R between 0 and 1 is extracted.
# 2. A probability P is calculated as 1/(old_value*lfu_log_factor+1).
# 3. The counter is incremented only if R < P.
#
# The default lfu-log-factor is 10. This is a table of how the frequency
# counter changes with a different number of accesses with different
# logarithmic factors:
```

```
#
# +--------+------------+------------+------------+------------+------------+
# | factor | 100 hits   | 1000 hits  | 100K hits  | 1M hits    | 10M hits   |
# +--------+------------+------------+------------+------------+------------+
# | 0      | 104        | 255        | 255        | 255        | 255        |
# +--------+------------+------------+------------+------------+------------+
# | 1      | 18         | 49         | 255        | 255        | 255        |
# +--------+------------+------------+------------+------------+------------+
# | 10     | 10         | 18         | 142        | 255        | 255        |
# +--------+------------+------------+------------+------------+------------+
# | 100    | 8          | 11         | 49         | 143        | 255        |
# +--------+------------+------------+------------+------------+------------+
#
# NOTE: The above table was obtained by running the following
commands:
#
#   redis-benchmark -n 1000000 incr foo
#   redis-cli object freq foo
#
# NOTE 2: The counter initial value is 5 in order to give new objects a
chance
# to accumulate hits.
#
# The counter decay time is the time, in minutes, that must elapse in
order
# for the key counter to be divided by two (or decremented if it has a
value
# less <= 10).
#
# The default value for the lfu-decay-time is 1. A Special value of 0
means to
# decay the counter every time it happens to be scanned.
#
# lfu-log-factor 10
# lfu-decay-time 1

############################## ACTIVE DEFRAGMENTATION
#######################
#
# WARNING THIS FEATURE IS EXPERIMENTAL. However it was
stress tested
# even in production and manually tested by multiple engineers for
some
# time.
#
# What is active defragmentation?
# -------------------------------
#
# Active (online) defragmentation allows a Redis server to compact the
# spaces left between small allocations and deallocations of data in
memory,
# thus allowing to reclaim back memory.
#
# Fragmentation is a natural process that happens with every allocator
(but
```

```
# less so with Jemalloc, fortunately) and certain workloads. Normally a
server
# restart is needed in order to lower the fragmentation, or at least to
flush
# away all the data and create it again. However thanks to this feature
# implemented by Oran Agra for Redis 4.0 this process can happen at
runtime
# in an "hot" way, while the server is running.
#
# Basically when the fragmentation is over a certain level (see the
# configuration options below) Redis will start to create new copies of
the
# values in contiguous memory regions by exploiting certain specific
Jemalloc
# features (in order to understand if an allocation is causing
fragmentation
# and to allocate it in a better place), and at the same time, will release
the
# old copies of the data. This process, repeated incrementally for all the
keys
# will cause the fragmentation to drop back to normal values.
#
# Important things to understand:
#
# 1. This feature is disabled by default, and only works if you compiled
Redis
#    to use the copy of Jemalloc we ship with the source code of Redis.
#    This is the default with Linux builds.
#
# 2. You never need to enable this feature if you don't have
fragmentation
#    issues.
#
# 3. Once you experience fragmentation, you can enable this feature
when
#    needed with the command "CONFIG SET activedefrag yes".
#
# The configuration parameters are able to fine tune the behavior of the
# defragmentation process. If you are not sure about what they mean it
is
# a good idea to leave the defaults untouched.

# Enabled active defragmentation
# activedefrag yes

# Minimum amount of fragmentation waste to start active defrag
# active-defrag-ignore-bytes 100mb

# Minimum percentage of fragmentation to start active defrag
# active-defrag-threshold-lower 10

# Maximum percentage of fragmentation at which we use maximum
effort
# active-defrag-threshold-upper 100

# Minimal effort for defrag in CPU percentage
```

```
# active-defrag-cycle-min 5

# Maximal effort for defrag in CPU percentage
# active-defrag-cycle-max 75

# Maximum number of set/hash/zset/list fields that will be processed
from
# the main dictionary scan
# active-defrag-max-scan-fields 1000
```

- 使用 redis6.0.8 镜像创建容器(也叫运行镜像)

```
docker run  -p 6379:6379 --name myr3 --
privileged=true -v
/app/redis/redis.conf:/etc/redis/redis.conf
-v /app/redis/data:/data -d redis:6.0.8
redis-server /etc/redis/redis.conf
```

```
[root@zzyy redis]# docker run  -p 6379:6379 --name myr3 --privileged=true -v /app/redis/redis.conf:/etc/redis/redis.conf -
v /app/redis/data:/data -d redis:6.0.8 redis-server /etc/redis/redis.conf
bff87d0ed459ff0a8f34e7e141aef476121425c77f084611af606936b5f8652c
[root@zzyy redis]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
                    NAMES
bff87d0ed459        redis:6.0.8         "docker-entrypoint..."   2 seconds ago       Up 1 second         0.0.0.0:6379->637
9/tcp               myr3
[root@zzyy redis]#
```

- 测试 redis-cli 连接上来

```
[root@zzyy redis]# docker run  -p 6379:6379 --name myr3 --privileged=true -v /app/redis/redis.conf:/etc/redis/redis.conf -
v /app/redis/data:/data -d redis:6.0.8 redis-server /etc/redis/redis.conf
d673b7fe3bc8877d9ca36bbaa67e0f2091455389ba8ef00e94b6c51f1afece1a
[root@zzyy redis]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
                    NAMES
d673b7fe3bc8        redis:6.0.8         "docker-entrypoint..."   2 seconds ago       Up 1 second         0.0.0.0:6379->637
9/tcp               myr3
[root@zzyy redis]# docker exec -it myr3 /bin/bash
root@d673b7fe3bc8:/data# redis-cli
127.0.0.1:6379> set k1 v1
OK
127.0.0.1:6379> get k1
"v1"
127.0.0.1:6379>
```

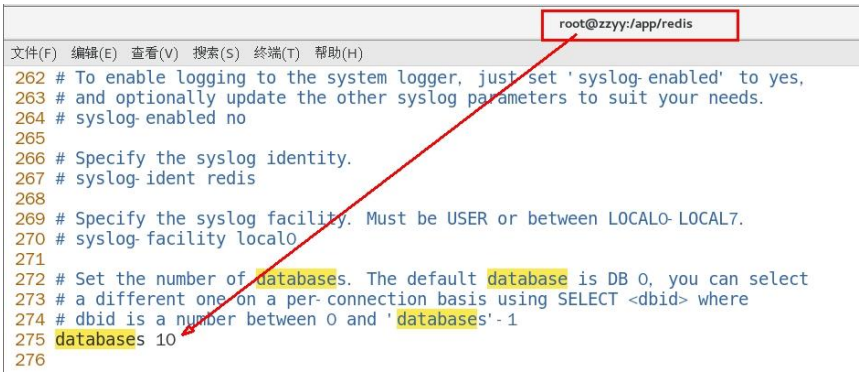
docker exec -it 运行着 Rediis 服务的容器 ID redis-cli

- 请证明 docker 启动使用了我们自己指定的配置文件
  - 修改前

```
[root@zzyy redis]# docker exec -it myr3 /bin/bash
root@d673b7fe3bc8:/data# redis-cli
127.0.0.1:6379> set k1 v1
OK
127.0.0.1:6379> get k1
"v1"
127.0.0.1:6379> select 15
OK
127.0.0.1:6379[15]> exit
root@d673b7fe3bc8:/data# exit
exit
[root@zzyy redis]#
```

我们用的配置文件，数据库默认是 16 个

- 修改后



宿主机的修改会同步给 docker 容器里面的配置。

- 记得重启服务

- 测试 redis-cli 连接上来第 2 次

```
[root@zzyy redis]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
                    NAMES
d673b7fe3bc8        redis:6.0.8         "docker-entrypoint..."   5 minutes ago       Up 5 minutes        0.0.0.0:6379->637
9/tcp               myr3
[root@zzyy redis]# docker restart myr3
myr3
[root@zzyy redis]# docker exec -it myr3 /bin/bash
root@d673b7fe3bc8:/data# redis-cli
127.0.0.1:6379> select 11
(error) ERR DB index is out of range
127.0.0.1:6379> select 9
OK
127.0.0.1:6379[9]> select 10
(error) ERR DB index is out of range
127.0.0.1:6379[9]> select 15
(error) ERR DB index is out of range
127.0.0.1:6379[9]>
```

1.8.5. 安装 Nginx

- 见高级篇 Portainer

2. 高级篇(大厂进阶)

2.1. Docker 复杂安装详说

2.1.1. 安装 mysql 主从复制

- 主从复制原理
  - 默认你懂
- 主从搭建步骤
  - 新建主服务器容器实例 3307

```
docker run -p 3307:3306 --name mysql-master \
-v /mydata/mysql-master/log:/var/log/mysql \
-v /mydata/mysql-master/data:/var/lib/mysql \
-v /mydata/mysql-master/conf:/etc/mysql \
-e MYSQL_ROOT_PASSWORD=root  \
-d mysql:5.7
```

  - 进入/mydata/mysql-master/conf 目录下新建 my.cnf
  - vim my.cnf

```
[mysqld]
## 设置 server_id，同一局域网中需要唯一
server_id=101
## 指定不需要同步的数据库名称
binlog-ignore-db=mysql
## 开启二进制日志功能
log-bin=mall-mysql-bin
## 设置二进制日志使用内存大小（事务）
```

```
binlog_cache_size=1M
## 设置使用的二进制日志格式
（mixed,statement,row）
binlog_format=mixed
## 二进制日志过期清理时间。默认值为 0，表示不
自动清理。
expire_logs_days=7
## 跳过主从复制中遇到的所有错误或指定类型的错
误，避免 slave 端复制中断。
## 如：1062 错误是指一些主键重复，1032 错误是
因为主从数据库数据不一致
slave_skip_errors=1062
```

- 修改完配置后重启 master 实例
- docker restart mysql-master
- 进入 mysql-master 容器
- docker exec -it mysql-master /bin/bash
- mysql -uroot -proot
- master 容器实例内创建数据同步用户
- CREATE USER 'slave'@'%' IDENTIFIED BY '123456';
- GRANT REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'slave'@'%';
- 新建从服务器容器实例 3308

```
docker run -p 3308:3306 --name mysql-slave \
-v /mydata/mysql-slave/log:/var/log/mysql \
-v /mydata/mysql-slave/data:/var/lib/mysql \
-v /mydata/mysql-slave/conf:/etc/mysql \
-e MYSQL_ROOT_PASSWORD=root  \
-d mysql:5.7
```

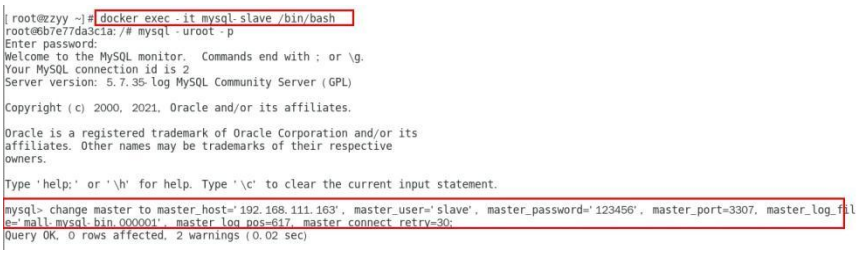
- 进入/mydata/mysql-slave/conf 目录下新建 my.cnf
- vim my.cnf

```
[mysqld]
## 设置 server_id，同一局域网中需要唯一
server_id=102
## 指定不需要同步的数据库名称
binlog-ignore-db=mysql
## 开启二进制日志功能，以备 Slave 作为其它数据库实例的 Master 时使用
log-bin=mall-mysql-slave1-bin
## 设置二进制日志使用内存大小（事务）
binlog_cache_size=1M
## 设置使用的二进制日志格式（mixed,statement,row）
binlog_format=mixed
## 二进制日志过期清理时间。默认值为 0，表示不自动清理。
expire_logs_days=7
## 跳过主从复制中遇到的所有错误或指定类型的错误，避免 slave 端复制中断。
## 如：1062 错误是指一些主键重复，1032 错误是因为主从数据库数据不一致
slave_skip_errors=1062
## relay_log 配置中继日志
relay_log=mall-mysql-relay-bin
## log_slave_updates 表示 slave 将复制事件写进自己的二进制日志
log_slave_updates=1
## slave 设置为只读（具有 super 权限的用户除外）
read_only=1
```

- 修改完配置后重启 slave 实例
- docker restart mysql-slave
- 在主数据库中查看主从同步状态
- show master status;
- 进入 mysql-slave 容器

- docker exec -it mysql-slave /bin/bash
- mysql -uroot -proot
- 在从数据库中配置主从复制

```
change master to master_host='宿主机 ip',
master_user='slave', master_password='123456',
master_port=3307, master_log_file='mall-mysql-
bin.000001', master_log_pos=617,
master_connect_retry=30;
```



- 主从复制命令参数说明

master_host：主数据库的 IP 地址；
master_port：主数据库的运行端口；
master_user：在主数据库创建的用于同步数据的用户账号；
master_password：在主数据库创建的用于同步数据的用户密码；
master_log_file：指定从数据库要复制数据的日志文件，通过查看主数据的状态，获取 File 参数；
master_log_pos：指定从数据库从哪个位置开始复制数据，通过查看主数据的状态，获取 Position 参数；
master_connect_retry：连接失败重试的时间间隔，单位为秒。

```
mysql> change master to master_host='192.168.111.163', master_user='slave', master_password='123456', master_port=3307, master_log_file='mall-mysql-bin.000001', master_log_pos=617, master_connect_retry=30;
Query OK, 0 rows affected, 2 warnings (0.02 sec)

mysql> show slave status \G;
*************************** 1. row ***************************
               Slave_IO_State:
                  Master_Host: 192.168.111.163
                  Master_User: slave
                  Master_Port: 3307
                Connect_Retry: 30
              Master_Log_File: mall-mysql-bin.000001
          Read_Master_Log_Pos: 617
               Relay_Log_File: mall-mysql-relay-bin.000001
                Relay_Log_Pos: 4
        Relay_Master_Log_File: mall-mysql-bin.000001
             Slave_IO_Running: No
            Slave_SQL_Running: No
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
```

没开始

- 在从数据库中查看主从同步状态
- show slave status \G;


```
mysql> change master to master_host='192.168.111.163', master_user='slave', master_password='123456', master_port=3307, master_log_file='mall-mysql-bin.000001', master_log_pos=617, master_connect_retry=30;
Query OK, 0 rows affected, 2 warnings (0.02 sec)

mysql> show slave status \G;
*************************** 1. row ***************************
               Slave_IO_State:
                  Master_Host: 192.168.111.163
                  Master_User: slave
                  Master_Port: 3307
                Connect_Retry: 30
              Master_Log_File: mall-mysql-bin.000001
          Read_Master_Log_Pos: 617
               Relay_Log_File: mall-mysql-relay-bin.000001
                Relay_Log_Pos: 4
        Relay_Master_Log_File: mall-mysql-bin.000001
             Slave_IO_Running: No
            Slave_SQL_Running: No
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
```

没开始


```
mysql> start slave;
Query OK, 0 rows affected (0.01 sec)
```

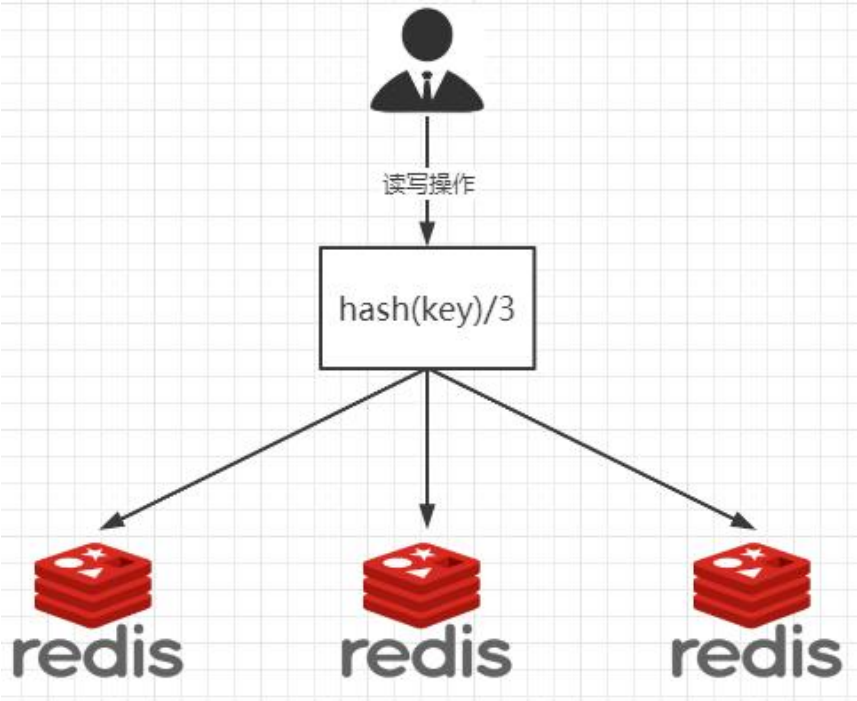
- 在从数据库中开启主从同步
- 查看从数据库状态发现已经同步

```
mysql> start slave;
Query OK, 0 rows affected (0.01 sec)

mysql> show slave status \G;
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 192.168.111.163
                  Master_User: slave
                  Master_Port: 3307
                Connect_Retry: 30
              Master_Log_File: mall-mysql-bin.000001
          Read_Master_Log_Pos: 617
               Relay_Log_File: mall-mysql-relay-bin.000002
                Relay_Log_Pos: 325
        Relay_Master_Log_File: mall-mysql-bin.000001
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
```

- 主从复制测试
- 主机新建库-使用库-新建表-插入数据，ok
- 从机使用库-查看记录，ok

2.1.2. 安装 redis 集群(大厂面试题第 4 季-分布式存储案例真题)

- cluster(集群)模式-docker 版 哈希槽分区进行亿级数据存储
  - 面试题
  - 1~2 亿条数据需要缓存，请问如何设计这个存储案例
  - 回答
  - 单机单台 100%不可能，肯定是分布式存储，用 redis 如何落地?
  - 上述问题阿里 P6~P7 工程案例和场景设计类必考题目， 一般业界有 3 种解决方案
  - 哈希取余分区



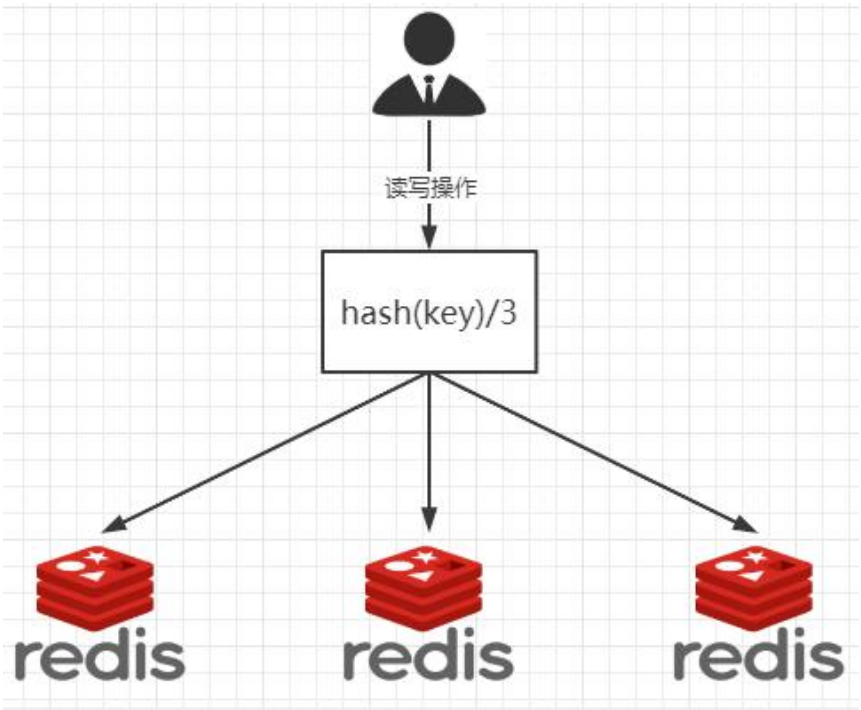
2 亿条记录就是 2 亿个 k,v，我们单机不行必须要分布式多机，假设有 3 台机器构成一个集群，用户每次读写操作都是根据公式：
hash(key) % N 个机器台数，计算出哈希值，用来决定数据映射到哪一个节点上。

优点：
简单粗暴，直接有效，只需要预估好数据规划好节点，例如 3 台、8 台、10 台，就能保证一段时间的数据支撑。使用 Hash 算法让固定的一部分请求落到同一台服务器上，这样每台服务器固定处理一部分请求（并维护这些请求的信息），起到负载均衡+分而治之的作用。

缺点：
原来规划好的节点，进行扩容或者缩容就比较麻烦了额，不管扩缩，每次数据变动导致节点有变动，映射关系需要重新进行计算，在服务器个数固定不变时没有问题，如果需要弹性扩容或故障停机的情况下，原来的取模公式就会发生变化：
Hash(key)/3 会变成 Hash(key) /?。此时地址经过取

余运算的结果将发生很大变化，根据公式获取的服务器也会变得不可控。
某个 redis 机器宕机了，由于台数数量变化，会导致 hash 取余全部数据重新洗牌。

- 缺点那？？？



缺点：

原来规划好的节点，进行扩容或者缩容就比较麻烦了额，不管扩缩，每次数据变动导致节点有变动，映射关系需要重新进行计算，在服务器个数固定不变时没有问题，如果需要弹性扩容或故障停机的情况下，原来的取模公式就会发生变化：Hash(key)/3 会变成 Hash(key) /?。此时地址经过取余运算的结果将发生很大变化，根据公式获取的服务器也会变得不可控。
某个 redis 机器宕机了，由于台数数量变化，会导

致 hash 取余全部数据重新洗牌。

- 一致性哈希算法分区
- 是什么

一致性 Hash 算法背景

一致性哈希算法在 1997 年由麻省理工学院中提出的，设计目标是为了解决
分布式缓存数据变动和映射问题，某个机器宕机了，分母数量改变了，自然取余数不 OK 了。

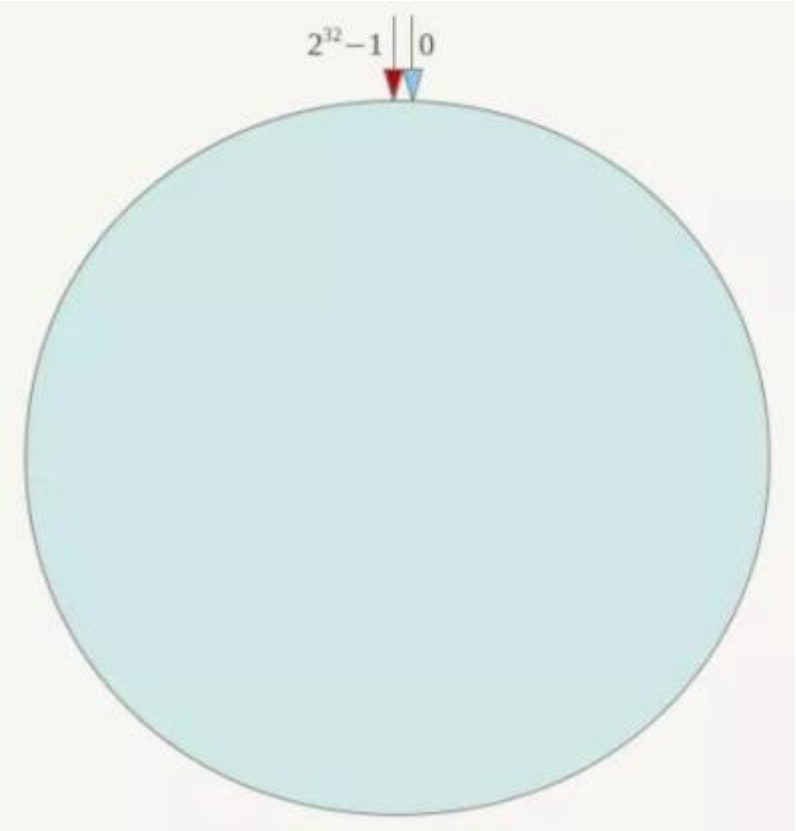
- 能干嘛
- 提出一致性 Hash 解决方案。 目的是当服务器个数发生变动时， 尽量减少影响客户端到服务器的映射关系
- **3 大步骤**
- 算法构建一致性哈希环

一致性哈希环

一致性哈希算法必然有个 hash 函数并按照算法产生 hash 值，这个算法的所有可能哈希值会构成一个全量集，这个集合可以成为一个 hash 空间[0,2^32-1]，这个是一个线性空间，但是在算法中，我们通过适当的逻辑控制将它首尾相连(0 = 2^32),这样让它逻辑上形成了一个环形空间。

它也是按照使用取模的方法，前面笔记介绍的节点取模法是对节点（服务器）的数量进行取模。而一致性 Hash 算法是对 2^32 取模，简单来说，一致性 Hash 算法将整个哈希值空间组织成一个虚拟的圆环，如假设某哈希函数 H 的值空间为 0-2^32-1（即哈希值是一个 32 位无符号整形），整个哈希环如下图：整个空间按顺时针方向组织，圆环的正上方的

点代表 0，0 点右侧的第一个点代表 1，以此类推，2、3、4、……直到 2^32-1，也就是说 0 点左侧的第一个点代表 2^32-1， 0 和 2^32-1 在零点中方向重合，我们把这个由 2^32 个点组成的圆环称为 Hash 环。

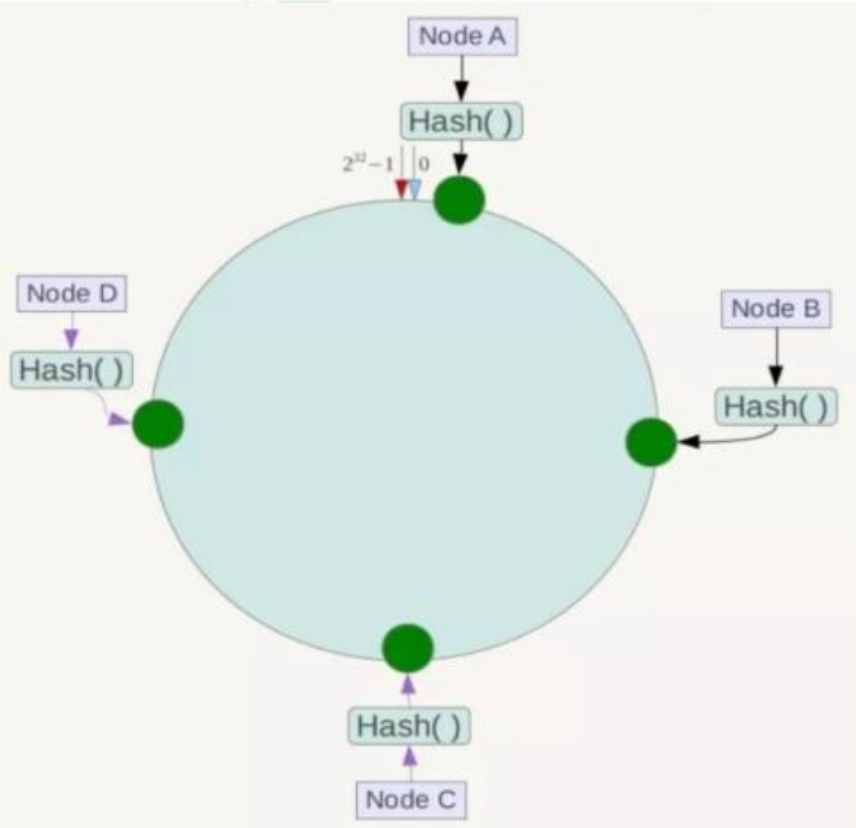


- 服务器 IP 节点映射

节点映射

将集群中各个 IP 节点映射到环上的某一个位置。

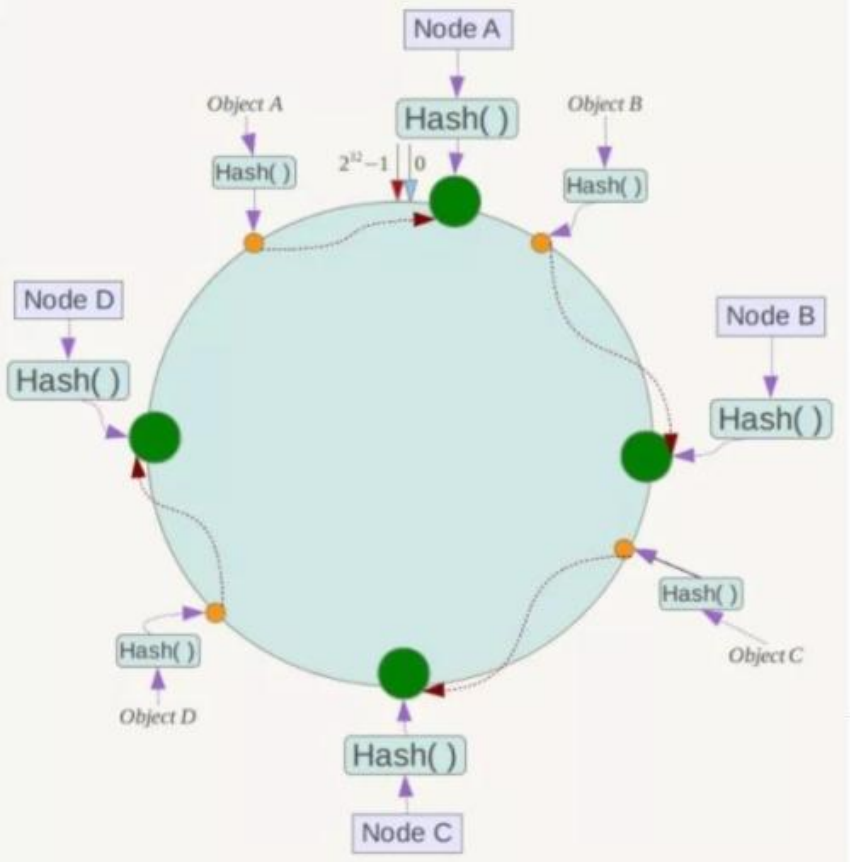
将各个服务器使用 Hash 进行一个哈希，具体可以选择服务器的 IP 或主机名作为关键字进行哈希，这样每台机器就能确定其在哈希环上的位置。假如 4 个节点 NodeA、B、C、D，经过 IP 地址的哈希函数计算(hash(ip))，使用 IP 地址哈希后在环空间的位置如下：



- key 落到服务器的落键规则

当我们需要存储一个 kv 键值对时，首先计算 key 的 hash 值，hash(key)，将这个 key 使用相同的函数 Hash 计算出哈希值并确定此数据在环上的位置，**从此位置沿环顺时针“行走”**，第一台遇到的服务器就是其应该定位到的服务器，并将该键值对存储在该节点上。

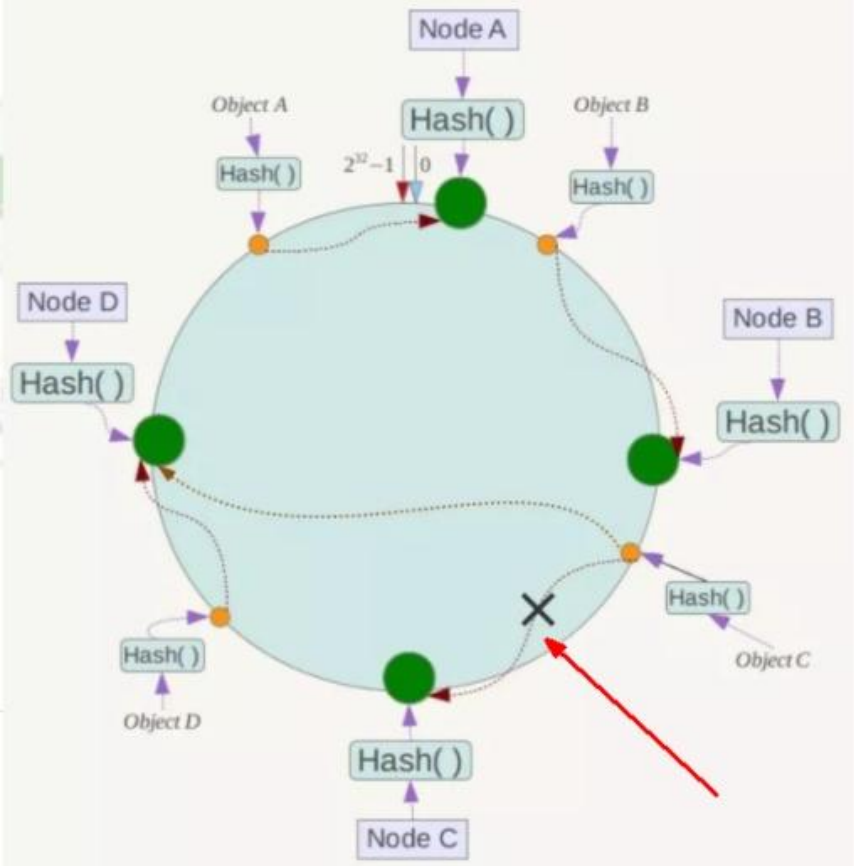
如我们有 Object A、Object B、Object C、Object D 四个数据对象，经过哈希计算后，在环空间上的位置如下：根据一致性 Hash 算法，数据 A 会被定为到 Node A 上，B 被定为到 Node B 上，C 被定为到 Node C 上，D 被定为到 Node D 上。



- 优点
- 一致性哈希算法的容错性

容错性

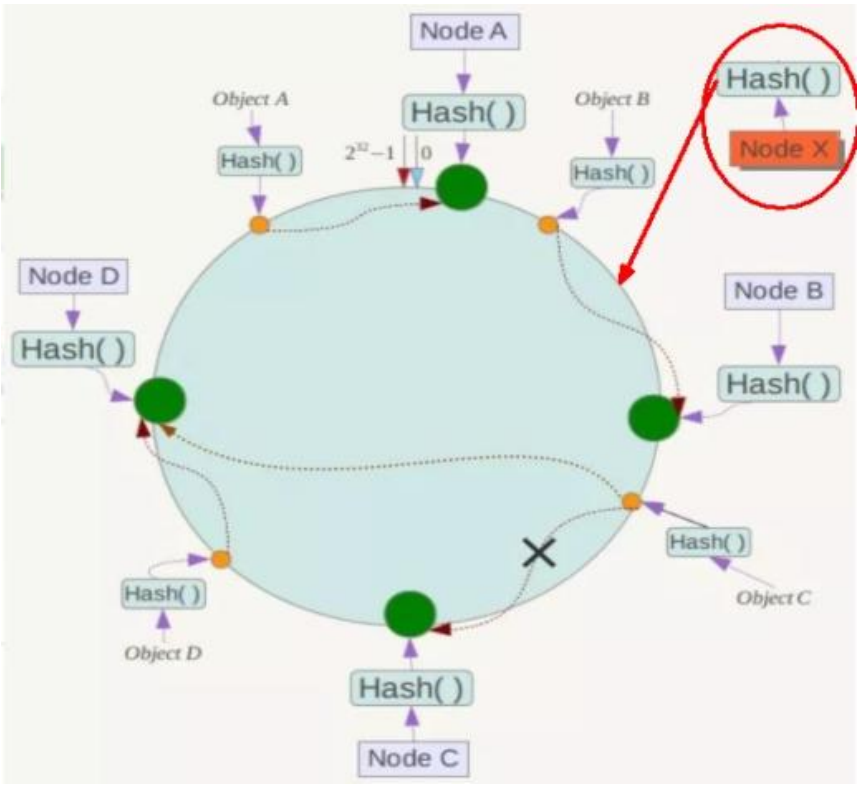
假设 Node C 宕机，可以看到此时对象 A、B、D 不会受到影响，只有 C 对象被重定位到 Node D。一般的，在一致性 Hash 算法中，如果一台服务器不可用，则受影响的数据仅仅是此服务器到其环空间中前一台服务器（即沿着逆时针方向行走遇到的第一台服务器）之间数据，其它不会受到影响。简单说，就是 C 挂了，受到影响的只是 B、C 之间的数据，并且这些数据会转移到 D 进行存储。



· 一致性哈希算法的**扩展性**

扩展性

数据量增加了，需要增加一台节点 NodeX，X 的位置在 A 和 B 之间，那收到影响的也就是 A 到 X 之间的数据，重新把 A 到 X 的数据录入到 X 上即可，不会导致 hash 取余全部数据重新洗牌。

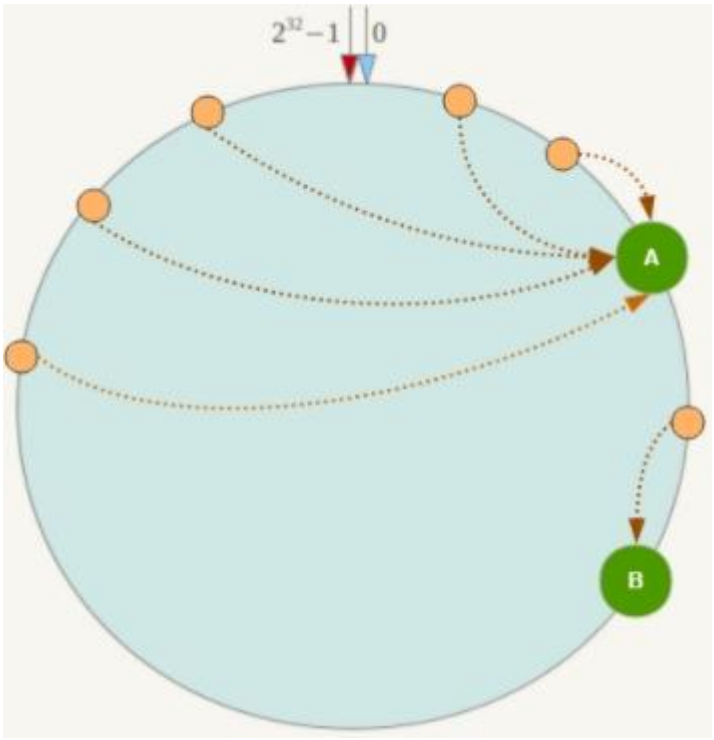


- 缺点
- 一致性哈希算法的数据倾斜问题

Hash 环的数据倾斜问题

一致性 Hash 算法在服务节点太少时，容易因为节点分布不均匀而造成数据倾斜（被缓存的对象大部分集中缓存在某一台服务器上）问题，

例如系统中只有两台服务器：



- 小总结

为了在节点数目发生改变时尽可能少的迁移数据

将所有的存储节点排列在收尾相接的 Hash 环上，每个 key 在计算 Hash 后会顺时针找到临近的存储节点存放。
而当有节点加入或退出时仅影响该节点在 Hash 环上顺时针相邻的后续节点。

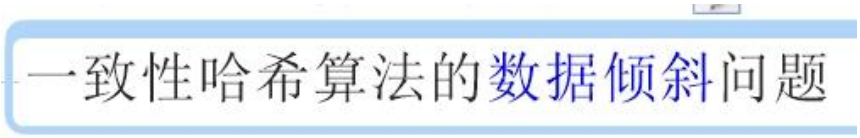
优点
加入和删除节点只影响哈希环中顺时针方向的相邻的节点，对其他节点无影响。

缺点

数据的分布和节点的位置有关，因为这些节点不是均匀的分布在哈希环上的，所以数据在进行存储时达不到均匀分布的效果。

- 哈希槽分区
- 是什么

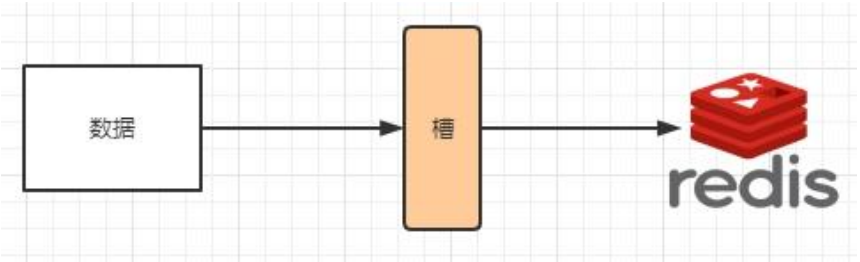
1 为什么出现



哈希槽实质就是一个数组，数组[0,2^14 -1]形成 hash slot 空间。

2 能干什么

解决均匀分配的问题，在数据和节点之间又加入了一层，把这层称为哈希槽（slot），用于管理数据和节点之间的关系，现在就相当于节点上放的是槽，槽里放的是数据。



槽解决的是粒度问题，相当于把粒度变大了，这样便于数据移动。
哈希解决的是映射问题，使用 key 的哈希值来计算所在的槽，便于数据分配。

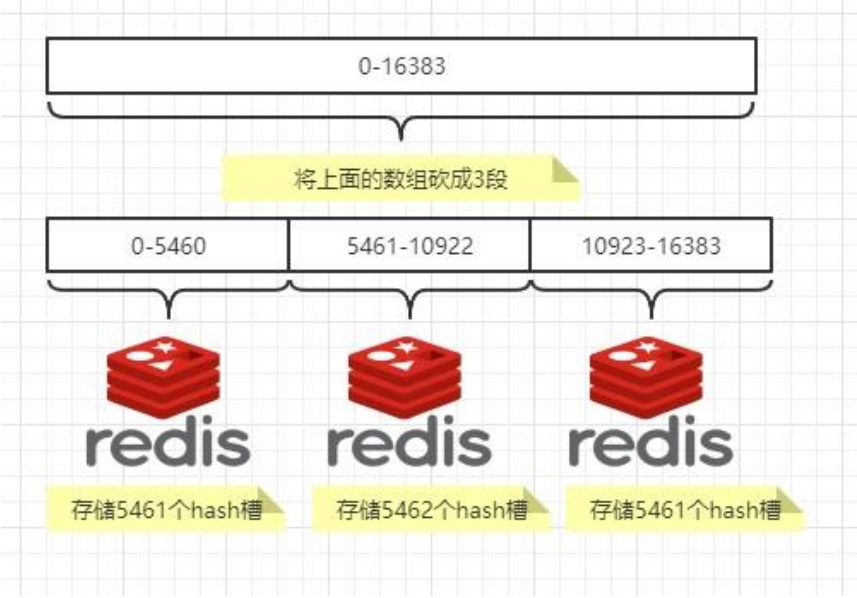
3 多少个 hash 槽

一个集群只能有 16384 个槽，编号 0-16383（0-2^14-1）。这些槽会分配给集群中的所有主节点，

分配策略没有要求。可以指定哪些编号的槽分配给哪个主节点。集群会记录节点和槽的对应关系。解决了节点和槽的关系后，接下来就需要对 key 求哈希值，然后对 16384 取余，余数是几 key 就落入对应的槽里。slot = CRC16(key) % 16384。以槽为单位移动数据，因为槽的数目是固定的，处理起来比较容易，这样数据移动问题就解决了。

- 哈希槽计算

Redis 集群中内置了 16384 个哈希槽，redis 会根据节点数量大致均等的将哈希槽映射到不同的节点。当需要在 Redis 集群中放置一个 key-value 时，redis 先对 key 使用 crc16 算法算出一个结果，然后把结果对 16384 求余数，这样每个 key 都会对应一个编号在 0-16383 之间的哈希槽，也就是映射到某个节点上。如下代码，key 之 A 、B 在 Node2， key 之 C 落在 Node3 上




```
@Test
public void test3()
{
    //import io.lettuce.core.cluster.SlotHash;
    System.out.println(SlotHash.getSlot( key: "A")); //6373
    System.out.println(SlotHash.getSlot( key: "B")); //10374
    System.out.println(SlotHash.getSlot( key: "C")); //14503
    System.out.println(SlotHash.getSlot( key: "hello")); //866
}
```

- 3 主 3 从 redis 集群扩缩容配置案例架构说明
- 见自己的 processon 笔记
- 开打步骤
- 3 主 3 从 redis 集群配置
- 关闭防火墙+启动 docker 后台服务

```
Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?
```

- systemctl start docker
- 新建 6 个 docker 容器 redis 实例

```
docker run -d --name redis-node-1 --net host --
privileged=true -v /data/redis/share/redis-node-1:/data
redis:6.0.8 --cluster-enabled yes --appendonly yes --port
6381
```

```
docker run -d --name redis-node-2 --net host --
privileged=true -v /data/redis/share/redis-node-2:/data
redis:6.0.8 --cluster-enabled yes --appendonly yes --port
6382

docker run -d --name redis-node-3 --net host --
privileged=true -v /data/redis/share/redis-node-3:/data
redis:6.0.8 --cluster-enabled yes --appendonly yes --port
6383

docker run -d --name redis-node-4 --net host --
privileged=true -v /data/redis/share/redis-node-4:/data
redis:6.0.8 --cluster-enabled yes --appendonly yes --port
6384

docker run -d --name redis-node-5 --net host --
privileged=true -v /data/redis/share/redis-node-5:/data
redis:6.0.8 --cluster-enabled yes --appendonly yes --port
6385

docker run -d --name redis-node-6 --net host --
privileged=true -v /data/redis/share/redis-node-6:/data
redis:6.0.8 --cluster-enabled yes --appendonly yes --port
6386
```

如果运行成功，效果如下：

```
[root@zzyy ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
   NAMES
98db1dc67057        redis:6.0.8         "docker-entrypoint..."   17 seconds ago      Created
   redis-node-6
bef9ed5e8986        redis:6.0.8         "docker-entrypoint..."   17 seconds ago      Created
   redis-node-5
cd9c83ffa4ef        redis:6.0.8         "docker-entrypoint..."   17 seconds ago      Created
   redis-node-4
0949158978d3        redis:6.0.8         "docker-entrypoint..."   17 seconds ago      Created
   redis-node-3
70f4ca34ec8d        redis:6.0.8         "docker-entrypoint..."   17 seconds ago      Created
   redis-node-2
6d4bcccd048c        redis:6.0.8         "docker-entrypoint..."   17 seconds ago      Created
   redis-node-1
```

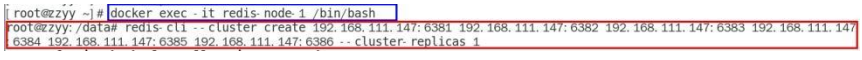
- 命令分步解释
- docker run
- 创建并运行 docker 容器实例
- --name redis-node-6
- 容器名字
- --net host
- 使用宿主机的 IP 和端口，默认
- --privileged=true

- 获取宿主机 root 用户权限
- -v /data/redis/share/redis-node-6:/data
- 容器卷，宿主机地址:docker 内部地址
- redis:6.0.8
- redis 镜像和版本号
- --cluster-enabled yes
- 开启 redis 集群
- --appendonly yes
- 开启持久化
- --port 6386
- redis 端口号
- 进入容器 redis-node-1 并为 6 台机器构建集群关系
- 进入容器
- docker exec -it redis-node-1 /bin/bash
- 构建主从关系

//注意，进入 docker 容器后才能执行一下命令，且注意自己的真实 IP 地址

```
redis-cli --cluster create 192.168.111.147:6381
192.168.111.147:6382 192.168.111.147:6383
192.168.111.147:6384 192.168.111.147:6385
192.168.111.147:6386 --cluster-replicas 1
```

--cluster-replicas 1 表示为每个 master 创建一个 slave 节点



```
>>> Performing hash slots allocation on 6 nodes...
Master[0] -> Slots 0 - 5460
Master[1] -> Slots 5461 - 10922
Master[2] -> Slots 10923 - 16383
Adding replica 192.168.111.147:6385 to 192.168.111.147:6381
Adding replica 192.168.111.147:6386 to 192.168.111.147:6382
Adding replica 192.168.111.147:6384 to 192.168.111.147:6383
>>> Trying to optimize slaves allocation for anti-affinity
[WARNING] Some slaves are in the same host as their master
M: 6753e3bb260fdb7b949a0388e1a30152ace37eb5 192.168.111.147:6381
   slots:[0-5460] (5461 slots) master
M: 6278214da93683debcf7e93ea08a5b445c800214 192.168.111.147:6382
   slots:[5461-10922] (5462 slots) master
M: fb72b1036f992145cf332ea3a8aeb4fa6a588889 192.168.111.147:6383
   slots:[10923-16383] (5461 slots) master
S: 4783e547973e8a0179080a45682f50e878985884 192.168.111.147:6384
   replicates fb72b1036f992145cf332ea3a8aeb4fa6a588889
S: 617e598eabccc21e9e03224e1cc17d090a2b942f 192.168.111.147:6385
   replicates 6753e3bb260fdb7b949a0388e1a30152ace37eb5
S: 841d887ac94df90de3ca0694da9ca8e8db9a28f2 192.168.111.147:6386
   replicates 6278214da93683debcf7e93ea08a5b445c800214
Can I set the above configuration? (type 'yes' to accept): yes
>>> Nodes configuration updated
>>> Assign a different config epoch to each node
>>> Sending CLUSTER MEET messages to join the cluster
Waiting for the cluster to join
..
>>> Performing Cluster Check (using node 192.168.111.147:6381)
M: 6753e3bb260fdb7b949a0388e1a30152ace37eb5 192.168.111.147:6381
   slots:[0-5460] (5461 slots) master
   1 additional replica(s)
S: 841d887ac94df90de3ca0694da9ca8e8db9a28f2 192.168.111.147:6386
   slots: (0 slots) slave
   replicates 6278214da93683debcf7e93ea08a5b445c800214
M: fb72b1036f992145cf332ea3a8aeb4fa6a588889 192.168.111.147:6383
   slots:[10923-16383] (5461 slots) master
   1 additional replica(s)
S: 4783e547973e8a0179080a45682f50e878985884 192.168.111.147:6384
   slots: (0 slots) slave
   replicates fb72b1036f992145cf332ea3a8aeb4fa6a588889
M: 6278214da93683debcf7e93ea08a5b445c800214 192.168.111.147:6382
   slots:[5461-10922] (5462 slots) master
   1 additional replica(s)
S: 617e598eabccc21e9e03224e1cc17d090a2b942f 192.168.111.147:6385
   slots: (0 slots) slave
   replicates 6753e3bb260fdb7b949a0388e1a30152ace37eb5
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
root@zzyy:/data#
```

- 一切 OK 的话，3 主 3 从搞定
- 链接进入 6381 作为切入点，查看集群状态
- 链接进入 6381 作为切入点，查看节点状态

```
root@zzyy:/data# redis-cli -p 6381
127.0.0.1:6381> keys *
(empty array)
127.0.0.1:6381> cluster info
cluster_state:ok
cluster_slots_assigned:16384
cluster_slots_ok:16384
cluster_slots_pfail:0
cluster_slots_fail:0
cluster_known_nodes:6
cluster_size:3
cluster_current_epoch:6
cluster_my_epoch:1
cluster_stats_messages_ping_sent:1102
cluster_stats_messages_pong_sent:1088
cluster_stats_messages_sent:2190
cluster_stats_messages_ping_received:1083
cluster_stats_messages_pong_received:1102
cluster_stats_messages_meet_received:5
cluster_stats_messages_received:2190
127.0.0.1:6381>
```

```
127.0.0.1:6381> cluster nodes
841d887ac94df90de3ca0694da9ca8e8db9a28f2 192.168.111.147:6386@16386 slave 6278214da93683debcf7e93ea08a5b445c800214 0 1608970106195 2 connected
fb72b1036f992145cf332ea3a8aeb4fa6a588889 192.168.111.147:6383@16383 master - 0 1608970107203 3 connected 10923-16383
4783e547973e8a0179080a45682f50e878985884 192.168.111.147:6384@16384 slave fb72b1036f992145cf332ea3a8aeb4fa6a588889 0 1608970104000 3 connected
6278214da93683debcf7e93ea08a5b445c800214 192.168.111.147:6382@16382 master - 0 1608970106000 2 connected 5461-10922
617e598eabccc21e9e03224e1cc17d090a2b942f 192.168.111.147:6385@16385 slave 6753e3bb260fdb7b949a0388e1a30152ace37eb5 0 1608970106000 1 connected
6753e3bb260fdb7b949a0388e1a30152ace37eb5 192.168.111.147:6381@16381 myself,master - 0 1608970105000 1 connected 0-5460
127.0.0.1:6381>
```

- cluster info
- cluster nodes
- 主从容错切换迁移案例
- 数据读写存储
- 启动 6 机构成的集群并通过 exec 进入
- 对 6381 新增两个 key
- 防止路由失效加参数-c 并新增两个 key

```
root@zzyy:/data# redis-cli -p 6381
127.0.0.1:6381> set k1 v1
(error) MOVED 12706 192.168.111.147:6383
```

加入参数-c，优化路由

```
127.0.0.1:6381> quit
root@zzyy:/data# redis-cli -p 6381 -c
127.0.0.1:6381> set k1 v-cluster1
-> Redirected to slot [12706] located at 192.168.111.147:6383
OK
192.168.111.147:6383> set k2 v-cluster2
-> Redirected to slot [449] located at 192.168.111.147:6381
OK
192.168.111.147:6381>
```

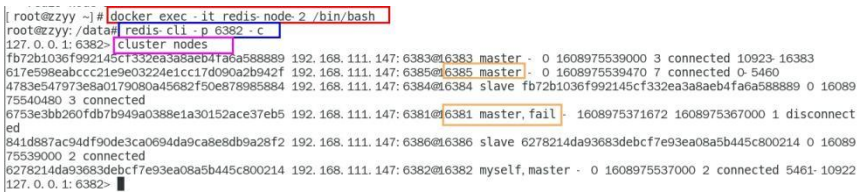
- 查看集群信息

redis-cli --cluster check 192.168.111.147:6381

```
root@zzyy:/data# redis-cli -p 6381 -c
127.0.0.1:6381> get k1
-> Redirected to slot [12706] located at 192.168.111.147:6383
"v1"
192.168.111.147:6383> get k2
-> Redirected to slot [449] located at 192.168.111.147:6381
"v2"
192.168.111.147:6381> get k3
(nil)
192.168.111.147:6381> quit
root@zzyy:/data# redis-cli -p 6382 -c
127.0.0.1:6382> get k1
-> Redirected to slot [12706] located at 192.168.111.147:6383
"v1"
192.168.111.147:6383> get k2
-> Redirected to slot [449] located at 192.168.111.147:6381
"v2"
192.168.111.147:6381> get k3
(nil)
192.168.111.147:6381>

root@zzyy:/data# redis-cli --cluster check 192.168.111.147:6381
192.168.111.147:6381 (6753e3bb...) -> 1 keys | 5461 slots | 1 slaves.
192.168.111.147:6382 (6278214d...) -> 0 keys | 5462 slots | 1 slaves.
192.168.111.147:6383 (fb72b103...) -> 1 keys | 5461 slots | 1 slaves.
[OK] 2 keys in 3 masters.
0.00 keys per slot on average.
>>> Performing Cluster Check (using node 192.168.111.147:6381)
M: 6753e3bb260fdb7b949a0388e1a30152ace37eb5 192.168.111.147:6381
   slots:[0-5460] (5461 slots) master
   1 additional replica(s)
S: 841d887ac94df90de3ca0694da9ca8e8db9a28f2 192.168.111.147:6386
   slots: (0 slots) slave
   replicates 6278214da93683debcf7e93ea08a5b445c800214
M: 6278214da93683debcf7e93ea08a5b445c800214 192.168.111.147:6382
   slots:[5461-10922] (5462 slots) master
   1 additional replica(s)
M: fb72b1036f992145cf332ea3a8aeb4fa6a588889 192.168.111.147:6383
   slots:[10923-16383] (5461 slots) master
   1 additional replica(s)
S: 4783e547973e8a0179080a45682f50e878985884 192.168.111.147:6384
   slots: (0 slots) slave
```

- 容错切换迁移

- 主 6381 和从机切换，先停止主机 6381
- 6381 主机停了，对应的真实从机上位
- 6381 作为 1 号主机分配的从机以实际情况为准，具体是几号机器就是几号
- 再次查看集群信息

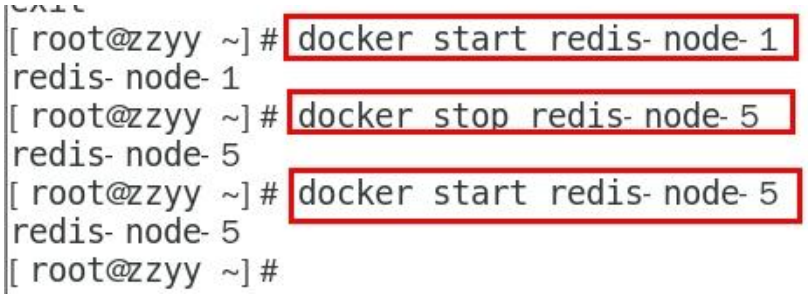


6381 宕机了，6385 上位成为了新的 master。

备注：本次脑图笔记 6381 为主下面挂从 6385。

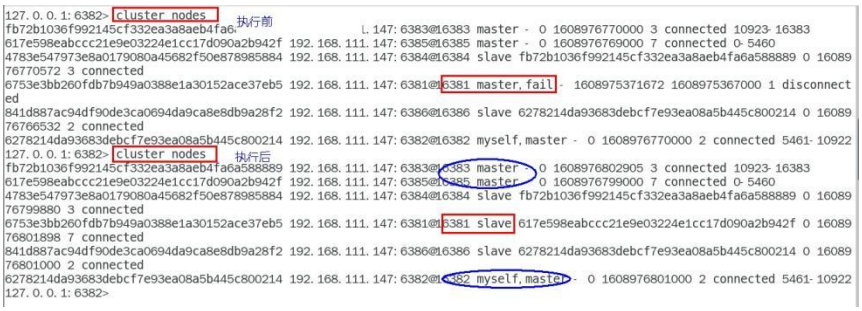
每次案例下面挂的从机以实际情况为准，具体是几号机器就是几号

- 先还原之前的 3 主 3 从



中间需要等待一会儿，docker 集群重新响应。

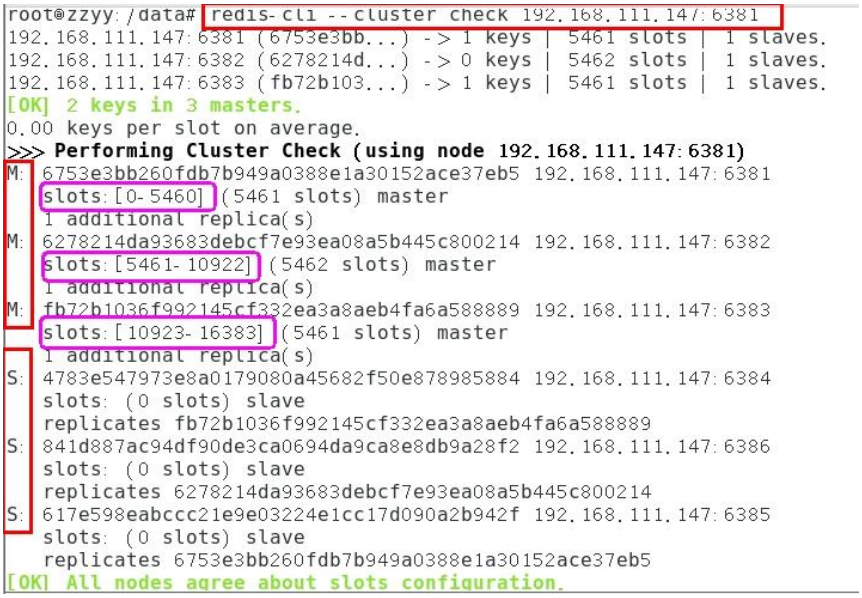
- 先启 6381



- docker start redis-node-1
- 再停 6385

- docker stop redis-node-5
- 再启 6385

- docker start redis-node-5
- 主从机器分配情况以实际情况为准
- 查看集群状态
- redis-cli --cluster check 自己 IP:6381



- 主从扩容案例
- 新建 6387、6388 两个节点+新建后启动+查看是否 8 节点

docker run -d --name redis-node-7 --net host --privileged=true -v /data/redis/share/redis-node-7:/data redis:6.0.8 --cluster-enabled yes --appendonly yes --port 6387
docker run -d --name redis-node-8 --net host --privileged=true -v /data/redis/share/redis-node-8:/data redis:6.0.8 --cluster-enabled yes --appendonly yes --port 6388
docker ps

- 进入 6387 容器实例内部
- docker exec -it redis-node-7 /bin/bash
- 将新增的 6387 节点(空槽号)作为 master 节点加入原集群

将新增的 6387 作为 master 节点加入集群 redis-cli --cluster add-node 自己实际 IP 地址:6387

自己实际 IP 地址:6381
6387 就是将要作为 master 新增节点
6381 就是原来集群节点里面的领路人，相当于
6387 拜拜 6381 的码头从而找到组织加入集群

```
[root@zzyy ~]# docker exec -it redis-node-7 /bin/bash
root@zzyy:/data# redis-cli --cluster add-node 192.168.111.147:6387 192.168.111.147:6381
>>> Adding node 192.168.111.147:6387 to cluster 192.168.111.147:6381
>>> Performing Cluster Check (using node 192.168.111.147:6381)
M: 6753e3bb260fdb7b949a0388e1a30152ace37eb5 192.168.111.147:6381
   slots:[0-5460] (5461 slots) master
   1 additional replica(s)
M: 6278214da93683debcf7e93ea08a5b445c800214 192.168.111.147:6382
   slots:[5461-10922] (5462 slots) master
   1 additional replica(s)
M: fb72b1036f992145cf332ea3a8aeb4fa6a588889 192.168.111.147:6383
   slots:[10923-16383] (5461 slots) master
   1 additional replica(s)
S: 4783e547973e8a0179080a45682f50e878985884 192.168.111.147:6384
   slots: (0 slots) slave
   replicates fb72b1036f992145cf332ea3a8aeb4fa6a588889
S: 841d887ac94df90de3ca0694da9ca8e8db9a28f2 192.168.111.147:6386
   slots: (0 slots) slave
   replicates 6278214da93683debcf7e93ea08a5b445c800214
S: 617e598eabccc21e9e03224e1cc17d090a2b942f 192.168.111.147:6385
   slots: (0 slots) slave
   replicates 6753e3bb260fdb7b949a0388e1a30152ace37eb5
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
>>> Send CLUSTER MEET to node 192.168.111.147:6387 to make it join the cluster.
[OK] New node added correctly.
root@zzyy:/data#
```

- 检查集群情况第 1 次

redis-cli --cluster check 真实 ip 地址:6381

```
[OK] New node added correctly.
root@zzyy:/data#
root@zzyy:/data# redis-cli --cluster check 192.168.111.147:6381
192.168.111.147:6381 (6753e3bb...) -> 1 keys | 5461 slots | 1 slaves.
192.168.111.147:6382 (6278214d...) -> 0 keys | 5462 slots | 1 slaves.
192.168.111.147:6383 (fb72b103...) -> 1 keys | 5461 slots | 1 slaves.
192.168.111.147:6387 (e4781f64...) -> 0 keys | 0 slots | 0 slaves.
[OK] 2 keys in 4 masters.
0.00 keys per slot on average.
>>> Performing Cluster Check (using node 192.168.111.147:6381)
M: 6753e3bb260fdb7b949a0388e1a30152ace37eb5 192.168.111.147:6381
   slots:[0-5460] (5461 slots) master
   1 additional replica(s)
M: 6278214da93683debcf7e93ea08a5b445c800214 192.168.111.147:6382
   slots:[5461-10922] (5462 slots) master
   1 additional replica(s)
M: fb72b1036f992145cf332ea3a8aeb4fa6a588889 192.168.111.147:6383
   slots:[10923-16383] (5461 slots) master
   1 additional replica(s)
S: 4783e547973e8a0179080a45682f50e878985884 192.168.111.147:6384
   slots: (0 slots) slave
   replicates fb72b1036f992145cf332ea3a8aeb4fa6a588889
S: 841d887ac94df90de3ca0694da9ca8e8db9a28f2 192.168.111.147:6386
   slots: (0 slots) slave
   replicates 6278214da93683debcf7e93ea08a5b445c800214
M: e4781f644d4a4e4d4b4d107157b9ba8144631451 192.168.111.147:6387
```

暂时没有槽号

```
   slots: (0 slots) master
S: 617e598eabccc21e9e03224e1cc17d090a2b942f 192.168.111.147:6385
   slots: (0 slots) slave
   replicates 6753e3bb260fdb7b949a0388e1a30152ace37eb5
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
root@zzyy:/data#
```

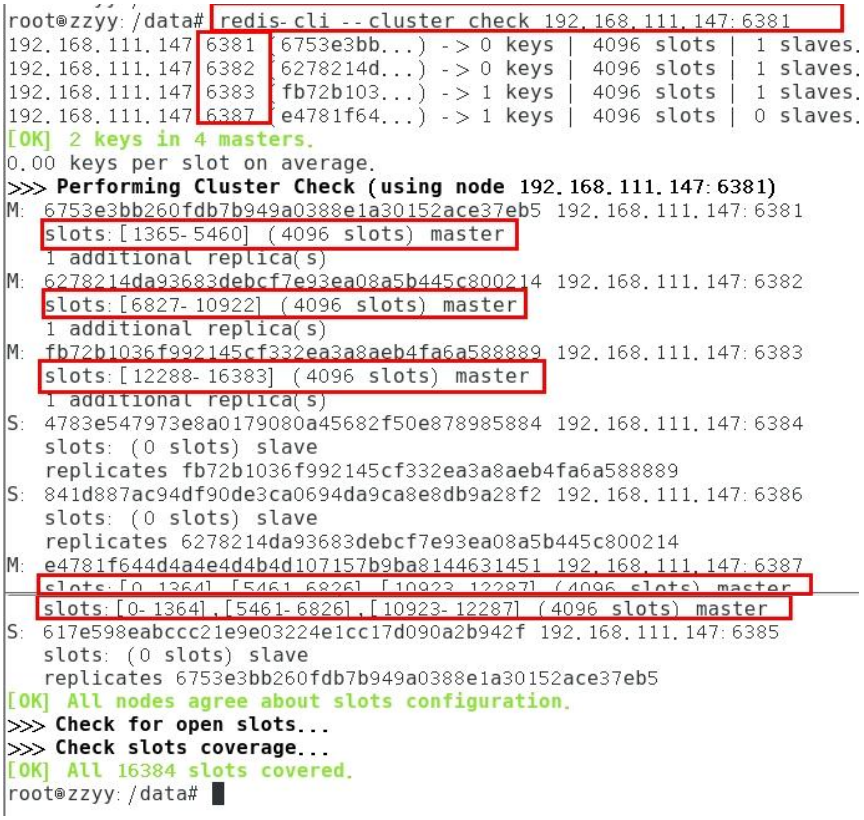
- 重新分派槽号

重新分派槽号
命令:redis-cli --cluster **reshard** IP 地址:端口号
redis-cli --cluster reshard 192.168.111.147:6381

```
root@zzyy:/data# redis-cli --cluster reshard 192.168.111.147:6381
>>> Performing Cluster Check (using node 192.168.111.147:6381)
M: 6753e3bb260fdb7b949a0388e1a30152ace37eb5 192.168.111.147:6381
   slots:[0-5460] (5461 slots) master
   1 additional replica(s)
M: 6278214da93683debcf7e93ea08a5b445c800214 192.168.111.147:6382
   slots:[5461-10922] (5462 slots) master
   1 additional replica(s)
M: fb72b1036f992145cf332ea3a8aeb4fa6a588889 192.168.111.147:6383
   slots:[10923-16383] (5461 slots) master
   1 additional replica(s)
S: 4783e547973e8a0179080a45682f50e878985884 192.168.111.147:6384
   slots: (0 slots) slave
   replicates fb72b1036f992145cf332ea3a8aeb4fa6a588889
S: 841d887ac94df90de3ca0694da9ca8e8db9a28f2 192.168.111.147:6386
   slots: (0 slots) slave
   replicates 6278214da93683debcf7e93ea08a5b445c800214
M: e4781f644d4a4e4d4b4d107157b9ba8144631451 192.168.111.147:6387
   slots: (0 slots) master
S: 617e598eabccc21e9e03224e1cc17d090a2b942f 192.168.111.147:6385
   slots: (0 slots) slave
   replicates 6753e3bb260fdb7b949a0388e1a30152ace37eb5
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
M: e4781f644d4a4e4d4b4d107157b9ba8144631451 192.168.111.147:6387
   slots: (0 slots) master
S: 617e598eabccc21e9e03224e1cc17d090a2b942f 192.168.111.147:6385
   slots: (0 slots) slave
   replicates 6753e3bb260fdb7b949a0388e1a30152ace37eb5
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered
How many slots do you want to move (from 1 to 16384)? 4096
What is the receiving node ID? e4781f644d4a4e4d4b4d107157b9ba8144631451
Please enter all the source node IDs.
  Type 'all' to use all the nodes as source nodes for the hash slots.
  Type 'done' once you entered all the source nodes IDs.
Source node #1: all

Ready to move 4096 slots.                    16384/master台数
  Source nodes:
    M: 6753e3bb260fdb7b949a0388e1a30152ace37eb5 192.168.111.147:6381
       slots:[0-5460] (5461 slots) master
       1 additional replica(s)
    M: 6278214da93683debcf7e93ea08a5b445c800214 192.168.111.147:6382
       slots:[5461-10922] (5462 slots) master
       1 additional replica(s)
    M: fb72b1036f992145cf332ea3a8aeb4fa6a588889 192.168.111.147:6383
       slots:[10923-16383] (5461 slots) master
```

- 检查集群情况第 2 次

redis-cli --cluster check 真实 ip 地址:6381



- 槽号分派说明

为什么 6387 是 3 个新的区间，以前的还是连续？重新分配成本太高，所以前 3 家各自匀出来一部分，从 6381/6382/6383 三个旧节点分别匀出 1364 个坑位给新节点 6387

```
>>> Performing Cluster Check (using node 192.168.111.147:6381)
M: 6753e3bb260fdb7b949a0388e1a30152ace37eb5 192.168.111.147:6381
   slots:[1365-5460] (4096 slots) master
   1 additional replica(s)
M: 6278214da93683debcf7e93ea08a5b445c800214 192.168.111.147:6382
   slots:[6827-10922] (4096 slots) master
   1 additional replica(s)
M: fb72b1036f992145cf332ea3a8aeb4fa6a588889 192.168.111.147:6383
   slots:[12288-16383] (4096 slots) master
   1 additional replica(s)
S: 4783e547973e8a0179080a45682f50e878985884 192.168.111.147:6384
   slots: (0 slots) slave
   replicates fb72b1036f992145cf332ea3a8aeb4fa6a588889
S: 841d887ac94df90de3ca0694da9ca8e8db9a28f2 192.168.111.147:6386
   slots: (0 slots) slave
   replicates 6278214da93683debcf7e93ea08a5b445c800214
M: e4781f644d4a4e4d4b4d107157b9ba8144631451 192.168.111.147:6387
   slots:[0-1364],[5461-6826],[10923-12287] (4096 slots) master
S: 617e598eabccc21e9e03224e1cc17d090a2b942f 192.168.111.147:6385
   slots: (0 slots) slave
   replicates 6753e3bb260fdb7b949a0388e1a30152ace37eb5
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
```

· 为主节点 6387 分配从节点 6388

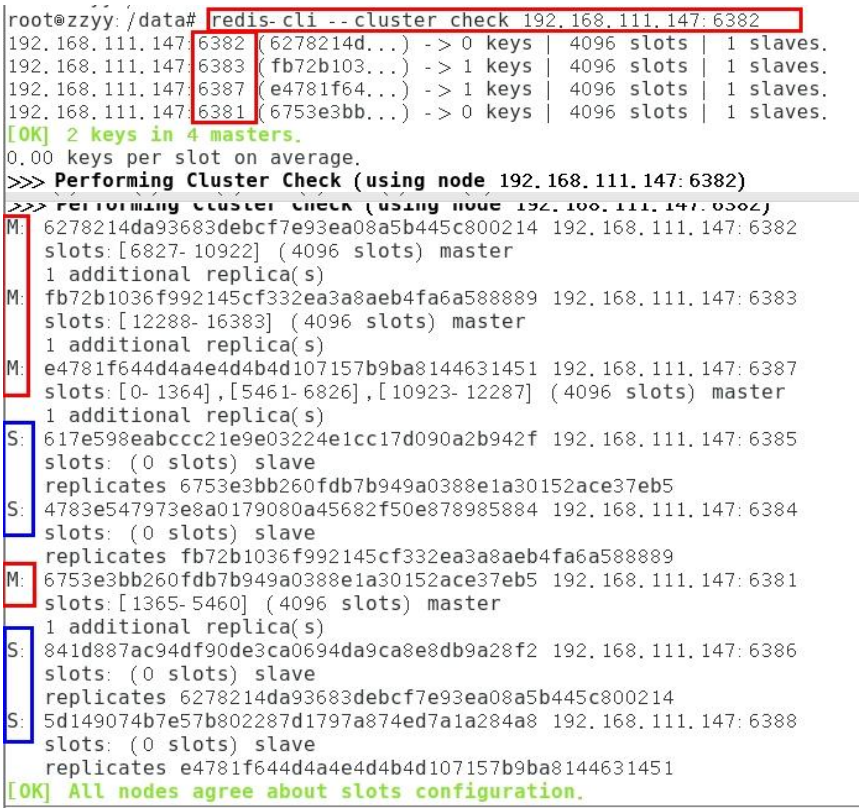
命令：redis-cli --cluster add-node ip:新 slave 端口 ip:新 master 端口 --cluster-slave --cluster-master-id 新主机节点 ID

redis-cli --cluster add-node 192.168.111.147:6388 192.168.111.147:6387 --cluster-slave --cluster-master-id e4781f644d4a4e4d4b4d107157b9ba8144631451------这个是 6387 的编号，按照自己实际情况

```
root@zzyy:/data# redis-cli --cluster add-node 192.168.111.147:6388 192.168.111.147:6387 --cluster-slave --cluster-master-id e4781f644d4a4e4d4b4d107157b9ba8144631451
>>> Adding node 192.168.111.147:6388 to cluster 192.168.111.147:6387
>>> Performing Cluster Check (using node 192.168.111.147:6387)
M: e4781f644d4a4e4d4b4d107157b9ba8144631451 192.168.111.147:6387
   slots:[0-1364],[5461-6826],[10923-12287] (4096 slots) master
S: 617e598eabccc21e9e03224e1cc17d090a2b942f 192.168.111.147:6385
   slots: (0 slots) slave
   replicates 6753e3bb260fdb7b949a0388e1a30152ace37eb5
M: 6278214da93683debcf7e93ea08a5b445c800214 192.168.111.147:6382
   slots:[6827-10922] (4096 slots) master
   1 additional replica(s)
S: 4783e547973e8a0179080a45682f50e878985884 192.168.111.147:6384
   slots: (0 slots) slave
   replicates fb72b1036f992145cf332ea3a8aeb4fa6a588889
M: 6753e3bb260fdb7b949a0388e1a30152ace37eb5 192.168.111.147:6381
   slots:[1365-5460] (4096 slots) master
   1 additional replica(s)
S: 841d887ac94df90de3ca0694da9ca8e8db9a28f2 192.168.111.147:6386
   slots: (0 slots) slave
   replicates 6278214da93683debcf7e93ea08a5b445c800214
M: fb72b1036f992145cf332ea3a8aeb4fa6a588889 192.168.111.147:6383
   slots:[12288-16383] (4096 slots) master
   1 additional replica(s)
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
>>> Send CLUSTER MEET to node 192.168.111.147:6388 to make it join the cluster.
Waiting for the cluster to join

>>> Configure node as replica of 192.168.111.147:6387.
[OK] New node added correctly.
root@zzyy:/data#
```

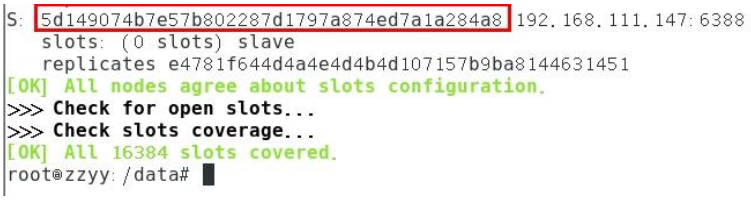
- 检查集群情况第 3 次

```
redis-cli --cluster check 192.168.111.147:6382
```



- 主从缩容案例
- 目的：6387 和 6388 下线
- 检查集群情况 1 获得 6388 的节点 ID

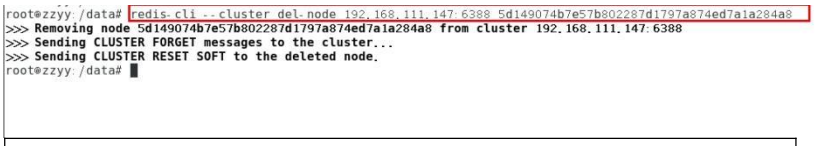
```
redis-cli --cluster check 192.168.111.147:6382
```



- 将 6388 删除 从集群中将 4 号从节点 6388 删除

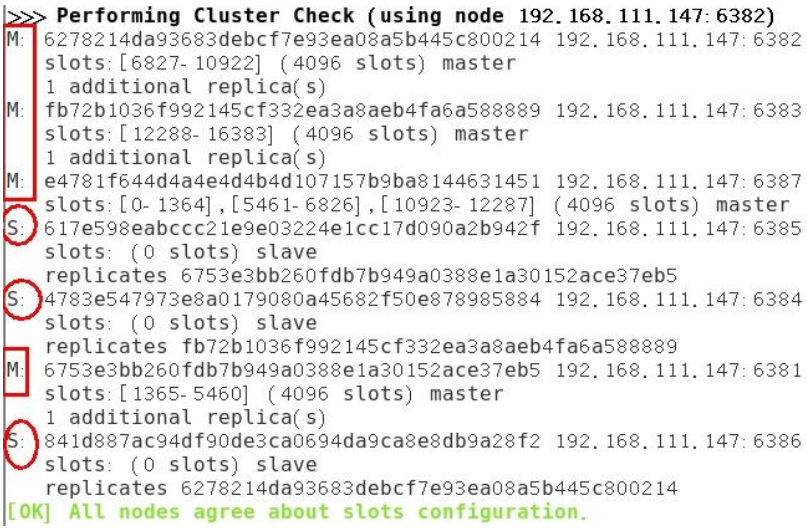
命令：redis-cli --cluster del-node ip:从机端口 从机 6388 节点 ID

redis-cli --cluster del-node 192.168.111.147:6388 5d149074b7e57b802287d1797a874ed7a1a284a8



redis-cli --cluster check 192.168.111.147:6382

检查一下发现，6388 被删除了，只剩下 7 台机器了。



- 将 6387 的槽号清空，重新分配，本例将清出来的槽号都给 6381

redis-cli --cluster reshard 192.168.111.147:6381

- 检查集群情况第二次

redis-cli --cluster check 192.168.111.147:6381

4096 个槽位都指给 6381，它变成了 8192 个槽位，相当于全部都给 6381 了，不然要输入 3 次，一锅端

```
[root@zzyy ~]# redis-cli --cluster check 192.168.111.147:6381
192.168.111.147:6381 (6753e3bb...) -> 1 keys | 8192 slots | 1 slaves.
192.168.111.147:6382 (6278214d...) -> 0 keys | 4096 slots | 1 slaves.
192.168.111.147:6383 (fb72b103...) -> 1 keys | 4096 slots | 1 slaves.
192.168.111.147:6387 (e4781f64...) -> 0 keys | 0 slots | 0 slaves.
[OK] 2 keys in 4 masters.
0.00 keys per slot on average.
>>> Performing Cluster Check (using node 192.168.111.147:6381)
M: 6753e3bb260fdb7b949a0388e1a30152ace37eb5 192.168.111.147:6381
   slots:[0-6826],[10923-12287] (8192 slots) master
   1 additional replica(s)
M: 6278214da93683debcf7e93ea08a5b445c800214 192.168.111.147:6382
   slots:[6827-10922] (4096 slots) master
   1 additional replica(s)
M: fb72b1036f992145cf332ea3a8aeb4fa6a588889 192.168.111.147:6383
   slots:[12288-16383] (4096 slots) master
   1 additional replica(s)
S: 4783e547973e8a0179080a45682f50e878985884 192.168.111.147:6384
   slots: (0 slots) slave
   replicates fb72b1036f992145cf332ea3a8aeb4fa6a588889
S: 841d887ac94df90de3ca0694da9ca8e8db9a28f2 192.168.111.147:6386
   slots: (0 slots) slave
   replicates 6278214da93683debcf7e93ea08a5b445c800214
M: e4781f644d4a4e4d4b4d107157b9ba8144631451 192.168.111.147:6387
   slots: (0 slots) master
S: 617e598eabccc21e9e03224e1cc17d090a2b942f 192.168.111.147:6385
```

- 将 6387 删除

命令：redis-cli --cluster del-node ip:端口 6387 节点 ID

redis-cli --cluster del-node 192.168.111.147:6387 e4781f644d4a4e4d4b4d107157b9ba8144631451

```
[root@zzyy ~]# redis-cli --cluster del-node 192.168.111.147:6387 e4781f644d4a4e4d4b4d107157b9ba8144631451
>>> Removing node e4781f644d4a4e4d4b4d107157b9ba8144631451 from cluster 192.168.111.147:6387
>>> Sending CLUSTER FORGET messages to the cluster...
>>> Sending CLUSTER RESET SOFT to the deleted node.
[root@zzyy ~]#
```

- 检查集群情况第三次

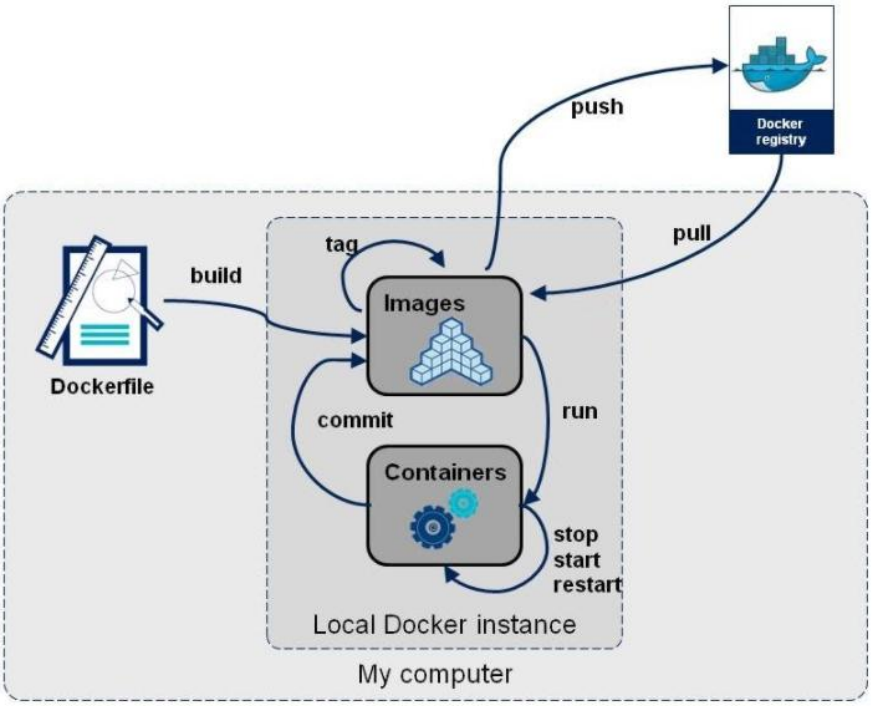
redis-cli --cluster check 192.168.111.147:6381

```
root@zzyy:/data# redis-cli --cluster check 192.168.111.147:6381
192.168.111.147:6381 (6753e3bb...) -> 1 keys | 8192 slots | 1 slaves.
192.168.111.147:6382 (6278214d...) -> 0 keys | 4096 slots | 1 slaves.
192.168.111.147:6383 (fb72b103...) -> 1 keys | 4096 slots | 1 slaves.
[OK] 2 keys in 3 masters.
0.00 keys per slot on average.
>>> Performing Cluster Check (using node 192.168.111.147:6381)
M: 6753e3bb260fdb7b949a0388e1a30152ace37eb5 192.168.111.147:6381
   slots:[0-6826],[10923-12287] (8192 slots) master
   1 additional replica(s)
M: 6278214da93683debcf7e93ea08a5b445c800214 192.168.111.147:6382
   slots:[6827-10922] (4096 slots) master
   1 additional replica(s)
M: fb72b1036f992145cf332ea3a8aeb4fa6a588889 192.168.111.147:6383
   slots:[12288-16383] (4096 slots) master
   1 additional replica(s)
S: 4783e547973e8a0179080a45682f50e878985884 192.168.111.147:6384
   slots: (0 slots) slave
   replicates fb72b1036f992145cf332ea3a8aeb4fa6a588889
S: 841d887ac94df90de3ca0694da9ca8e8db9a28f2 192.168.111.147:6386
   slots: (0 slots) slave
   replicates 6278214da93683debcf7e93ea08a5b445c800214
S: 617e598eabccc21e9e03224e1cc17d090a2b942f 192.168.111.147:6385
   slots: (0 slots) slave
   replicates 6753e3bb260fdb7b949a0388e1a30152ace37eb5
```

2.2. DockerFile 解析

2.2.1. 是什么

- Dockerfile 是用来构建 Docker 镜像的文本文件，是由一条条构建镜像所需的指令和参数构成的脚本。
- 概述



- 官网
  - https://docs.docker.com/engine/reference/builder/
- 构建三步骤
  - 编写 Dockerfile 文件
  - docker build 命令构建镜像
  - docker run 依镜像运行容器实例

2.2.2. DockerFile 构建过程解析

- Dockerfile 内容基础知识
  - 1：每条保留字指令都必须为大写字母且后面要跟随至少一个参数
  - 2：指令按照从上到下，顺序执行
  - 3：#表示注释
  - 4：每条指令都会创建一个新的镜像层并对镜像进行提交
- Docker 执行 Dockerfile 的大致流程
  - （1）docker 从基础镜像运行一个容器

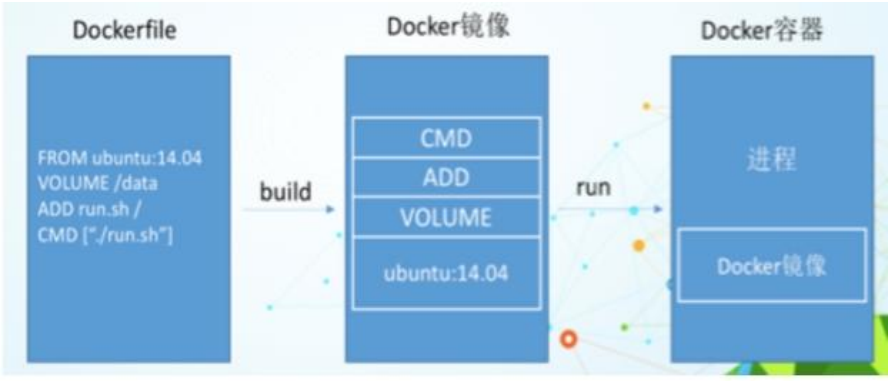
- （2）执行一条指令并对容器作出修改
- （3）执行类似 docker commit 的操作提交一个新的镜像层
- （4）docker 再基于刚提交的镜像运行一个新容器
- （5）执行 dockerfile 中的下一条指令直到所有指令都执行完成

- **小总结**

从应用软件的角度来看，Dockerfile、Docker 镜像与 Docker 容器分别代表软件的三个不同阶段，

* Dockerfile 是软件的原材料
* Docker 镜像是软件的交付品
* Docker 容器则可以认为是软件镜像的运行态，也即依照镜像运行的容器实例

Dockerfile 面向开发，Docker 镜像成为交付标准，Docker 容器则涉及部署与运维，三者缺一不可，合力充当 Docker 体系的基石。



1 Dockerfile，需要定义一个 Dockerfile，Dockerfile 定义了进程需要的一切东西。Dockerfile 涉及的内容包括执行代码或者是文件、环境变量、依赖包、运行时环境、动态链接库、操作系统的发行版、服务进程和内核进程(当应用进程需要和系统服务和内核进程打交

道，这时需要考虑如何设计 namespace 的权限控制)等等;

2 Docker 镜像，在用 Dockerfile 定义一个文件之后，docker build 时会产生一个 Docker 镜像，当运行 Docker 镜像时会真正开始提供服务;

3 Docker 容器，容器是直接提供服务的。

2.2.3. DockerFile 常用保留字指令

- 参考 tomcat8 的 dockerfile 入门
  - https://github.com/docker-library/tomcat
- FROM
  - 基础镜像，当前新镜像是基于哪个镜像的，指定一个已经存在的镜像作为模板，第一条必须是 from
- MAINTAINER
  - 镜像维护者的姓名和邮箱地址
- RUN
  - 容器构建时需要运行的命令
  - 两种格式
  - shell 格式

```
RUN <命令行命令>
# <命令行命令> 等同于，在终端操作的 shell 命令。
```

RUN yum -y install vim

- exec 格式

```
RUN ["可执行文件", "参数1", "参数2"]
# 例如：
# RUN ["./test.php", "dev", "offline"] 等价于 RUN ./test.php dev offline
```

- RUN 是在 docker build 时运行

- EXPOSE
  - 当前容器对外暴露出的端口
- WORKDIR
  - 指定在创建容器后，终端默认登陆的进来工作目录，一个落脚点
- USER
  - 指定该镜像以什么样的用户去执行，如果都不指定，默认是 root
- ENV
  - 用来在构建镜像过程中设置环境变量

ENV MY_PATH /usr/mytest
这个环境变量可以在后续的任何 RUN 指令中使用，这就如同在命令前面指定了环境变量前缀一样；
也可以在其它指令中直接使用这些环境变量，

比如：WORKDIR $MY_PATH

- ADD
  - 将宿主机目录下的文件拷贝进镜像且会自动处理 URL 和解压 tar 压缩包
- COPY
  - 类似 ADD，拷贝文件和目录到镜像中。 将从构建上下文目录中 <源路径> 的文件/目录复制到新的一层的镜像内的 <目标路径> 位置
  - COPY src dest
  - COPY ["src", "dest"]
  - <src 源路径>：源文件或者源目录

- <dest 目标路径>：容器内的指定路径，该路径不用事先建好，路径不存在的话，会自动创建。

- VOLUME
  - 容器数据卷，用于数据保存和持久化工作
- CMD
  - 指定容器启动后的要干的事情

CMD 容器启动命令

`CMD` 指令的格式和 `RUN` 相似，也是两种格式：

- `shell` 格式：`CMD <命令>`
- `exec` 格式：`CMD ["可执行文件", "参数1", "参数2"...]`
- 参数列表格式：`CMD ["参数1", "参数2"...]`。在指定了 `ENTRYPOINT` 指令后，用 `CMD` 指定具体的参数。

  - 注意
  - Dockerfile 中可以有多个 CMD 指令，但只有最后一个生效，CMD 会被 docker run 之后的参数替换
  - 参考官网 Tomcat 的 dockerfile 演示讲解

```
157   EXPOSE 8080
158   CMD ["catalina.sh", "run"]
```

  - 官网最后一行命令

```
[root@zzyy ~]# docker run -it -p 8080:8080 57800e5b1cbf /bin/bash
[root@eaedbd079d89 local]#
```

  - 我们演示自己的覆盖操作
  - 它和前面 RUN 命令的区别
  - CMD 是在 docker run 时运行。
  - RUN 是在 docker build 时运行。
- ENTRYPOINT
  - 也是用来指定一个容器启动时要运行的命令

- 类似于 CMD 指令，但是 ENTRYPOINT 不会被 docker run 后面的命令覆盖， 而且这些命令行参数会被当作参数送给 ENTRYPOINT 指令指定的程序

- 命令格式和案例说明

命令格式：

```
ENTRYPOINT ["<executeable>","<param1>","<param2>",...]
```

ENTRYPOINT 可以和 CMD 一起用，一般是变参才会使用 CMD ，这里的 CMD 等于是在给 ENTRYPOINT 传参。
当指定了 ENTRYPOINT 后，CMD 的含义就发生了变化，不再是直接运行其命令而是将 CMD 的内容作为参数传递给 ENTRYPOINT 指令，他两个组合会变成 `<ENTRYPOINT> "<CMD>"`

案例如下：假设已通过 Dockerfile 构建了 nginx:test 镜像：

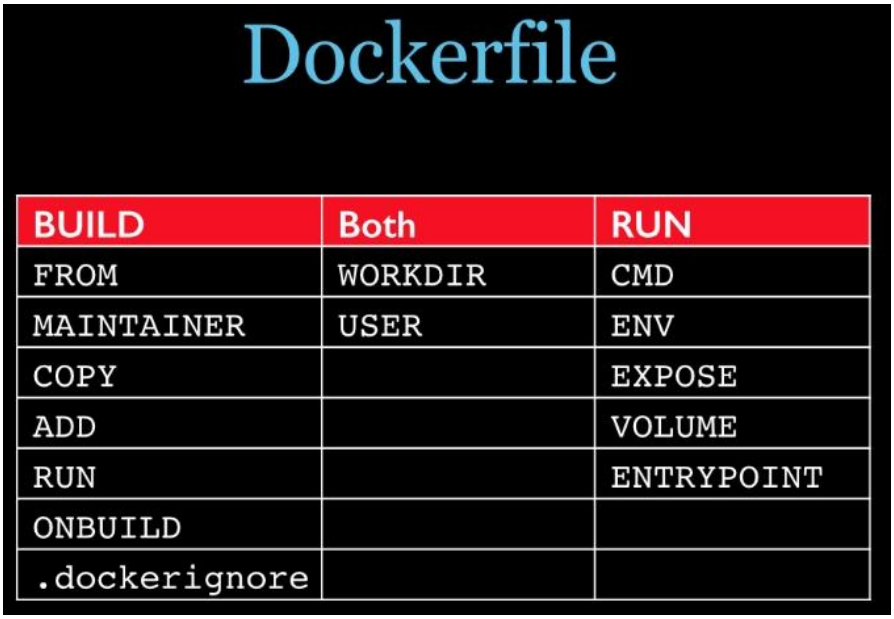
```
FROM nginx

ENTRYPOINT ["nginx", "-c"] # 定参
CMD ["/etc/nginx/nginx.conf"] # 变参
```

是否传参	按照 dockerfile 编写执行	传参运行
Docker 命令	docker run nginx:test	docker run nginx:test -c /etc/nginx/new.conf
衍生出的实际命令	nginx -c /etc/nginx/nginx.conf	nginx -c /etc/nginx/new.conf

- 优点

- 在执行 docker run 的时候可以指定 ENTRYPOINT 运行所需的参数。
- 注意
- 如果 Dockerfile 中如果存在多个 ENTRYPOINT 指令，仅最后一个生效。

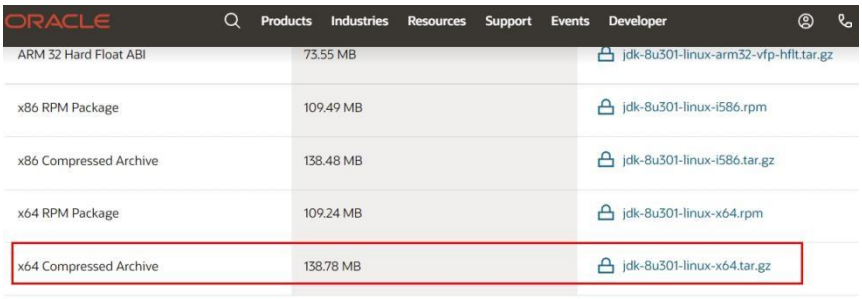
- **小总结**



2.2.4. 案例

- 自定义镜像 mycentosjava8
  - 要求
  - Centos7 镜像具备 vim+ifconfig+jdk8
  - JDK 的下载镜像地址
  - 官网

下载地址：
https://www.oracle.com/java/technologies/downloads/#java8



- https://mirrors.yangxingzhen.com/jdk/
- 编写
- 准备编写 Dockerfile 文件

```
[root@zzyy myfile]# pwd
/myfile
[root@zzyy myfile]# ll
总用量 186428
-rw-r--r--. 1 root root       585 8月   3 21:21 Dockerfile
-rwxr--r--. 1 root root 190890122 8月   3 21:20 jdk-8u171-linux-x64.tar.gz
[root@zzyy myfile]#
```

```
FROM centos
MAINTAINER zzyy<zzyybs@126.com>

ENV MYPATH /usr/local
WORKDIR $MYPATH

#安装 vim 编辑器
RUN yum -y install vim
#安装 ifconfig 命令查看网络 IP
RUN yum -y install net-tools
#安装 java8 及 lib 库
RUN yum -y install glibc.i686
RUN mkdir /usr/local/java
#ADD 是相对路径 jar,把 jdk-8u171-linux-x64.tar.gz 添加到容器中,安装包必须要和 Dockerfile 文件在同一位置
ADD jdk-8u171-linux-x64.tar.gz /usr/local/java/
#配置 java 环境变量
ENV JAVA_HOME /usr/local/java/jdk1.8.0_171
```

```
ENV JRE_HOME $JAVA_HOME/jre
ENV CLASSPATH
$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
:$JRE_HOME/lib:$CLASSPATH
ENV PATH $JAVA_HOME/bin:$PATH

EXPOSE 80

CMD echo $MYPATH
CMD echo "success----------ok"
CMD /bin/bash
```

- 大写字母 D
- 构建
- docker build -t 新镜像名字:TAG .

docker build -t centosjava8:1.5 .

```
[root@zzyy myfile]# vim Dockerfile
[root@zzyy myfile]# docker build -t centosjava8:1.5 .
Sending build context to Docker daemon  190.9MB
Step 1/17 : FROM centos
 ---> 300e315adb2f
Step 2/17 : MAINTAINER zzyy<zzyy167@126.com>
 ---> Using cache
 ---> e2dc53bd70a6
Step 3/17 : ENV MYPATH /usr/local
 ---> Using cache
 ---> 70737eff6f42
Step 4/17 : WORKDIR $MYPATH
 ---> Using cache
 ---> 0040cbbf46aa
Step 5/17 : RUN yum -y install vim
 ---> Using cache
 ---> d6a587ed2d5c
Step 6/17 : RUN yum -y install net-tools
 ---> Using cache
 ---> a1e0f85501b3
Step 7/17 : RUN yum -y install glibc.i686
```

```
Step 14/17 : EXPOSE 80
 ---> Running in 8d0551403838
Removing intermediate container 8d0551403838
 ---> dc0ce41071d7
Step 15/17 : CMD echo $MYPATH
 ---> Running in 88fbf82e3df3
Removing intermediate container 88fbf82e3df3
 ---> 72230cddcdb1
Step 16/17 : CMD echo "success--------------ok"
 ---> Running in c857ee9c31e4
Removing intermediate container c857ee9c31e4
 ---> 65333f0d2571
Step 17/17 : CMD /bin/bash
 ---> Running in 6f269a7d46ea
Removing intermediate container 6f269a7d46ea
 ---> fbec75b1a533
Successfully built fbec75b1a533
Successfully tagged centosjava8:1.5
[root@zzyy myfile]#
```

- 注意，上面 TAG 后面有个空格，有个点
- 运行
- docker run -it 新镜像名字:TAG

docker run -it centosjava8:1.5 /bin/bash

```
[root@zzyy myfile]# docker images
REPOSITORY                                                TAG       IMAGE ID
centosjava8                                               1.5       fbec75b1a533
mycentos                                                  1.1       a9ebb842dcc8
atguigu/myubuntu                                          1.1       3177bcdd22af
registry.cn-qingdao.aliyuncs.com/atguigush/myubuntu1.1    1.1       3177bcdd22af
ubuntu                                                    latest    c29284518f49
mysql                                                     5.6       e1aa75e199d7
nginx                                                     latest    d1a364dc548d
tomcat                                                    7         e614000ce544
centos                                                    latest    300e315adb2f
redis                                                     6.0.8     16ecd2772934
[root@zzyy myfile]# docker run -it centosjava8:1.5 /bin/bash
[root@1e020d23a70d local]# pwd
/usr/local
[root@1e020d23a70d local]# java -version
java version "1.8.0_171"
Java(TM) SE Runtime Environment (build 1.8.0_171-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.171-b11, mixed mode)
[root@1e020d23a70d local]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.2  netmask 255.255.0.0  broadcast 172.17.255.255
```

- 再体会下 UnionFS（联合文件系统）

UnionFS（联合文件系统）：Union 文件系统（UnionFS）是一种分层、轻量级并且高性能的文件系统，它支持对文件系统的修改作为一

次提交来一层层的叠加，同时可以将不同目录挂载到同一个虚拟文件系统下(unite several directories into a single virtual filesystem)。Union 文件系统是 Docker 镜像的基础。镜像可以通过分层来进行继承，基于基础镜像（没有父镜像），可以制作各种具体的应用镜像。

特性：一次同时加载多个文件系统，但从外面看起来，只能看到一个文件系统，联合加载会把各层文件系统叠加起来，这样最终的文件系统会包含所有底层的文件和目录

- 虚悬镜像
  - 是什么
  - 仓库名、标签都是<none>的镜像，俗称 dangling image
  - Dockerfile 写一个

1 vim Dockerfile

```
from ubuntu
CMD echo 'action is success'
```

2 docker build .

```
[root@zzyy zzyyuse]# docker images
REPOSITORY                    TAG       IMAGE ID       CREATED          SIZE
<none>                        <none>    da0fd00b7d91   9 seconds ago    72.8MB
atguigu/ubuntu                2.1       630e2281c13b   14 minutes ago   72.8MB
```

- 查看
- docker image ls -f dangling=true
- 命令结果

```
[root@zzyy zzyyuse]# docker image ls -f dangling=true
REPOSITORY   TAG       IMAGE ID       CREATED              SIZE
<none>       <none>    da0fd00b7d91   About a minute ago   72.8MB
[root@zzyy zzyyuse]#
```

- 删除

docker image prune

虚悬镜像已经失去存在价值，可以删除

```
[root@zzyy zzyyuse]# docker image ls -f dangling=true
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
<none>       <none>    92f13bd94a26   2 minutes ago    72.8MB
[root@zzyy zzyyuse]#
[root@zzyy zzyyuse]# docker image prune
WARNING! This will remove all dangling images.
Are you sure you want to continue? [y/N] y
Deleted Images:
deleted: sha256:92f13bd94a2651865d1bc15a68222a48d7ba2529ec90e59aaf161c6fffc2397c

Total reclaimed space: 0B
[root@zzyy zzyyuse]# docker image ls -f dangling=true
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
[root@zzyy zzyyuse]#
```

- 家庭作业-自定义镜像 myubuntu
  - 编写
  - 准备编写 DockerFile 文件

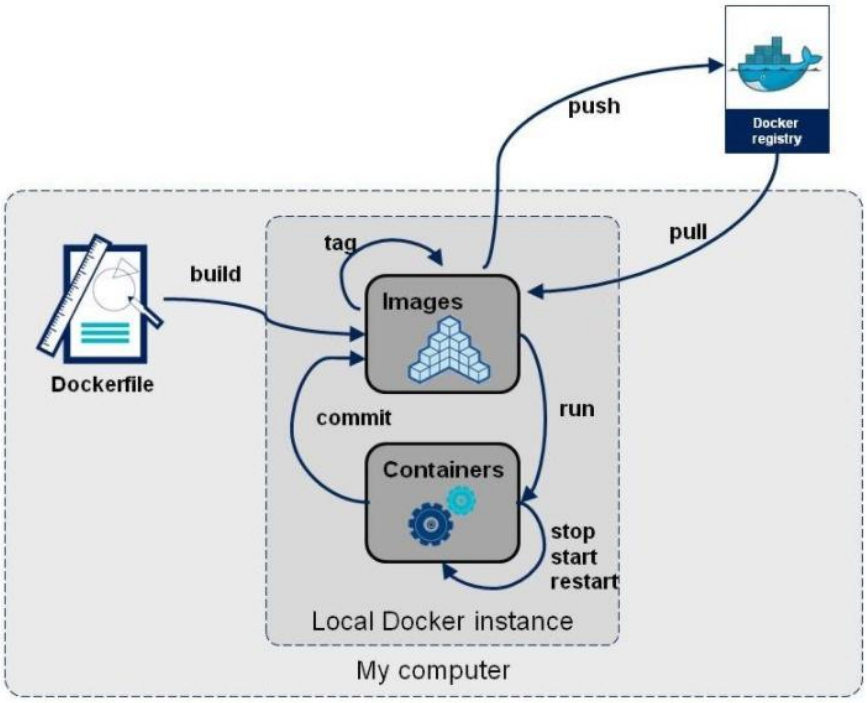
```
[root@atguigu mydockerfile]# pwd
/zzyyuse/mydockerfile
[root@atguigu mydockerfile]# vim Dockerfile
```

```
FROM ubuntu
MAINTAINER zzyy<zzyybs@126.com>

ENV MYPATH /usr/local
```

```
WORKDIR $MYPATH

RUN apt-get update
RUN apt-get install net-tools
#RUN apt-get install -y iproute2
#RUN apt-get install -y inetutils-ping

EXPOSE 80

CMD echo $MYPATH
CMD echo "install inconfig cmd into ubuntu
success--------------ok"
CMD /bin/bash
```

- 构建
- docker build -t 新镜像名字:TAG .
- 运行
- docker run -it 新镜像名字:TAG

2.2.5. 小总结



2.3. Docker 微服务实战

2.3.1. 通过 IDEA 新建一个普通微服务模块

- 建 Module
  - docker_boot
- 改 POM

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
         xsi:schemaLocation="http://maven.apache.org/
POM/4.0.0 https://maven.apache.org/xsd/maven-
4.0.0.xsd">
```

```
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-
parent</artifactId>
        <version>2.5.6</version>
        <relativePath/>
    </parent>

    <groupId>com.atguigu.docker</groupId>
    <artifactId>docker_boot</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <properties>
        <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.so
urce>
        <maven.compiler.target>1.8</maven.compiler.tar
get>
        <junit.version>4.12</junit.version>
        <log4j.version>1.2.17</log4j.version>
        <lombok.version>1.16.18</lombok.version>
        <mysql.version>5.1.47</mysql.version>
        <druid.version>1.1.16</druid.version>
        <mapper.version>4.1.5</mapper.version>
        <mybatis.spring.boot.version>1.3.0</mybatis.spri
```

```
ng.boot.version>
    </properties>

    <dependencies>
        <!--SpringBoot 通用依赖模块-->
        <dependency>
            <groupId>org.springframework.boot</groupId
>
            <artifactId>spring-boot-starter-
web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId
>
            <artifactId>spring-boot-starter-
actuator</artifactId>
        </dependency>
        <!--test-->
        <dependency>
            <groupId>org.springframework.boot</groupId
>
            <artifactId>spring-boot-starter-
test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</grou
pId>
                <artifactId>spring-boot-maven-
plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</grou
pId>
                <artifactId>maven-resources-
plugin</artifactId>
                <version>3.1.0</version>
            </plugin>
        </plugins>
    </build>

</project>
```

- 写 YML

```
server.port=6001
```

- 主启动

```
package com.atguigu.docker;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootAp
plication;

@SpringBootApplication
public class DockerBootApplication
{
    public static void main(String[] args)
    {
        SpringApplication.run(DockerBootApplication.cla
ss, args);
    }

}
```

- 业务类

```
package com.atguigu.docker.controller;

import
org.springframework.beans.factory.annotation.Value;
import
org.springframework.web.bind.annotation.RequestMap
ping;
```

```
import
org.springframework.web.bind.annotation.RequestMet
hod;
import
org.springframework.web.bind.annotation.RestControll
er;

import java.util.UUID;

/**
 * @auther zzyy
 * @create 2021-10-25 17:43
 */
@RestController
public class OrderController
{
    @Value("${server.port}")
    private String port;

    @RequestMapping("/order/docker")
    public String helloDocker()
    {
        return "hello docker"+"\t"+port+"\t"+
UUID.randomUUID().toString();
    }

    @RequestMapping(value ="/order/index",method =
```

```
RequestMethod.GET)
    public String index()
    {
        return "服务端口号:
"+"\t"+port+"\t"+UUID.randomUUID().toString();
    }
}
```

2.3.2. 通过 dockerfile 发布微服务部署到 docker 容器

- IDEA 工具里面搞定微服务 jar 包

docker_boot-0.0.1-SNAPSHOT.jar

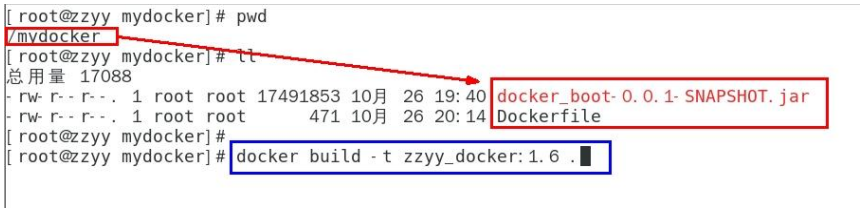
- 编写 Dockerfile

  - Dockerfile 内容

```
# 基础镜像使用 java
FROM java:8
# 作者
MAINTAINER zzyy
# VOLUME 指定临时文件目录为/tmp，在主机/var/lib/docker 目录下创建了一个临时文件并链接到容器的/tmp
VOLUME /tmp
```

```
# 将 jar 包添加到容器中并更名为 zzyy_docker.jar
ADD docker_boot-0.0.1-SNAPSHOT.jar zzyy_docker.jar
# 运行 jar 包
RUN bash -c 'touch /zzyy_docker.jar'
ENTRYPOINT ["java","-jar","/zzyy_docker.jar"]
#暴露 6001 端口作为微服务
EXPOSE 6001
```

- 将微服务 jar 包和 Dockerfile 文件上传到同一个目录下/mydocker




```
docker build -t zzyy_docker:1.6 .
```

- 构建镜像

```
[root@zzyy mydocker]# docker build -t zzyy_docker:1.6 .
Sending build context to Docker daemon  17.49MB
Step 1/7 : FROM java:8
 ---> d23bdf5b1b1b
Step 2/7 : MAINTAINER zzyy
 ---> Using cache
 ---> 9a0a5985e77b
Step 3/7 : VOLUME /tmp
 ---> Using cache
 ---> 1a9d0ebe4b34
Step 4/7 : ADD docker_boot-0.0.1-SNAPSHOT.jar zzyy_docker.jar
 ---> Using cache
 ---> d73f7279534b
Step 5/7 : RUN bash -c 'touch /zzyy_docker.jar'
 ---> Using cache
 ---> 3854efaabc4c
Step 6/7 : ENTRYPOINT ["java","-jar","/zzyy_docker.jar"]
 ---> Using cache
 ---> 300298ad4b59
Step 7/7 : EXPOSE 6001
 ---> Using cache
 ---> 5a3ccaebcb4a
```

- docker build -t zzyy_docker:1.6 .
- 打包成镜像文件

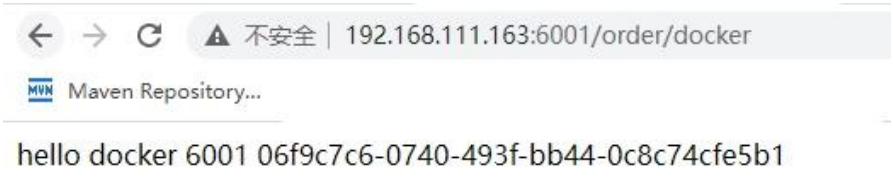

```
[root@zzyy mydocker]# docker build -t zzyy_docker:1.6 .
Sending build context to Docker daemon  17.49MB
Step 1/7 : FROM java:8
 ---> d23bdf5b1b1b
Step 2/7 : MAINTAINER zzyy
 ---> Using cache
 ---> 9a0a5985e77b
Step 3/7 : VOLUME /tmp
 ---> Using cache
 ---> 1a9d0ebe4b34
Step 4/7 : ADD docker_boot-0.0.1-SNAPSHOT.jar zzyy_docker.jar
 ---> Using cache
 ---> d73f7279534b
Step 5/7 : RUN bash -c 'touch /zzyy_docker.jar'
 ---> Using cache
 ---> 3854efaabc4c
Step 6/7 : ENTRYPOINT ["java","-jar","/zzyy_docker.jar"]
 ---> Using cache
 ---> 300298ad4b59
Step 7/7 : EXPOSE 6001
 ---> Using cache
 ---> 5a3ccaebcb4a
```


- 运行容器

```
docker run -d -p 6001:6001 zzyy_docker:1.6
```

```
[root@zzyy mydocker]# docker images
REPOSITORY                          TAG       IMAGE ID       CREATED          SIZE
zzyy_docker                         1.6       5a3ccaebcb4a   11 minutes ago   678MB
192.168.111.162:5000/zzyyubuntu     1.2       d6ca70a4f932   11 days ago      105MB
mysql                               5.7       8a8a506ccfdc   13 days ago      448MB
ubuntu                              latest    597ce1600cf4   3 weeks ago      72.8MB
hello-world                         latest    feb5d9fea6a5   4 weeks ago      13.3kB
registry                            latest    b2cb11db9d3d   7 weeks ago      26.2MB
redis                               6.0.8     16ecd2772934   12 months ago    104MB
java                                8         d23bdf5b1b1b   4 years ago      643MB
[root@zzyy mydocker]#
[root@zzyy mydocker]# docker run -d -p 6001:6001 zzyy_docker:1.6
97c4077b4653ed146a2c369e55f23fe0a0bf1b4243b7258f5be10ea148ae2b9e
[root@zzyy mydocker]#
[root@zzyy mydocker]# curl 127.0.0.1:6001/order/docker
hello docker    6001  curl 127.0.0.1:6001/order/index
服务端口号:     6001    f551410a-0342-433a-addf-b614b636640f[root@zzyy mydocker]#
[root@zzyy mydocker]#
[root@zzyy mydocker]#
```

- 访问测试



2.4. Docker 网络

2.4.1. 是什么

- docker 不启动，默认网络情况

```
[root@zzyy ~]# ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.111.163  netmask 255.255.255.0  broadcast 192.168.111.255
        inet6 fe80::c381:7898:3629:7bfd  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:85:cd:aa  txqueuelen 1000  (Ethernet)
        RX packets 395  bytes 47191 (46.0 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 290  bytes 27417 (26.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1  (Local Loopback)
        RX packets 148  bytes 15660 (15.2 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 148  bytes 15660 (15.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

virbr0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 192.168.122.1  netmask 255.255.255.0  broadcast 192.168.122.255
        ether 52:54:00:54:cd:0d  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

- ens33
- lo
- virbr0

在 CentOS7 的安装过程中如果有选择相关虚拟化的的服务安装系统后，启动网卡时会发现有一个以网桥连接的私网地址的 virbr0 网卡(virbr0 网卡：它还有一个固定的默认 IP 地址 192.168.122.1)，是做虚拟机网桥的使用的，其作用是为连接其上的虚机网卡提供 NAT 访问外网的功能。

我们之前学习 Linux 安装，勾选安装系统的时候附带了 libvirt 服务才会生成的一个东西，如果不需要可以直接将 libvirtd 服务卸载，
yum remove libvirt-libs.x86_64

- docker 启动后，网络情况

会产生一个名为 docker0 的虚拟网桥

```
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:4a:99:db:b6  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.111.166  netmask 255.255.255.0  broadcast 192.168.111.255
        inet6 fe80::30ef:bb67:e282:278d  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:85:cd:aa  txqueuelen 1000  (Ethernet)
        RX packets 2033  bytes 165047 (161.1 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 484  bytes 58636 (57.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1  (Local Loopback)
        RX packets 316  bytes 35316 (34.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 316  bytes 35316 (34.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

- 查看 docker 网络模式命令

默认创建 3 大网络模式

```
[root@zzyy tmp]# 当我们安装 Docker 后，默认会自动创建三个网络
bash: 当我们安装：未找到命令...
[root@zzyy tmp]#
[root@zzyy tmp]# docker network ls
NETWORK ID     NAME              DRIVER    SCOPE
a03815281634   bridge            bridge    local

ae13c04d7626   host              host      local

3bc4d8335e63   none              null      local

[root@zzyy tmp]#
```

2.4.2. 常用基本命令

- All 命令

```
[root@zzyy ~]# docker network --help

Usage:  docker network COMMAND

Manage networks

Commands:
  connect     Connect a container to a network
  create      Create a network
  disconnect  Disconnect a container from a network
  inspect     Display detailed information on one or more networks
  ls          List networks
  prune       Remove all unused networks
  rm          Remove one or more networks

Run 'docker network COMMAND --help' for more information on a command.
[root@zzyy ~]#
```

- 查看网络
  - docker network ls
- 查看网络源数据
  - docker network inspect XXX 网络名字
- 删除网络
  - docker network rm XXX 网络名字
- 案例

```
[root@zzyy ~]# docker network create aa_network
ca9a0e268bfd2bf7b94649a7005df6d603b0b9db0fd8d28e97e392df2f048bfd
[root@zzyy ~]# docker network ls
NETWORK ID          NAME                DRIVER    SCOPE
ca9a0e268bfd        aa_network          bridge    local
5abc9a503429        bridge              bridge    local
9db449f3a9b2        cig_default         bridge    local
ae13c04d7626        host                host      local
e3e51b1cfbbb        mydocker_default    bridge    local
3bc4d8335e63        none                null      local
db9e8053d380        zzyy_network        bridge    local
[root@zzyy ~]# docker network rm aa_network
aa_network
[root@zzyy ~]# docker network ls
NETWORK ID          NAME                DRIVER    SCOPE
5abc9a503429        bridge              bridge    local
9db449f3a9b2        cig_default         bridge    local
ae13c04d7626        host                host      local
e3e51b1cfbbb        mydocker_default    bridge    local
3bc4d8335e63        none                null      local
db9e8053d380        zzyy_network        bridge    local
[root@zzyy ~]#
```

2.4.3. 能干嘛

- 容器间的互联和通信以及端口映射
- 容器 IP 变动时候可以通过服务名直接网络通信而不受到影响

2.4.4. 网络模式

- 总体介绍

网络模式	简介
bridge	为每一个容器分配、设置 IP 等，并将容器连接到一个 `docker0` 虚拟网桥，默认为该模式。
host	容器将不会虚拟出自己的网卡，配置自己的 IP 等，而是使用宿主机的 IP 和端口。
none	容器有独立的 Network namespace，但并没有对其进行任何网络设置，如分配 veth pair 和网桥连接，IP 等。
container	新创建的容器不会创建自己的网卡和配置自己的 IP，而是和一个指定的容器共享 IP、端口范围等。

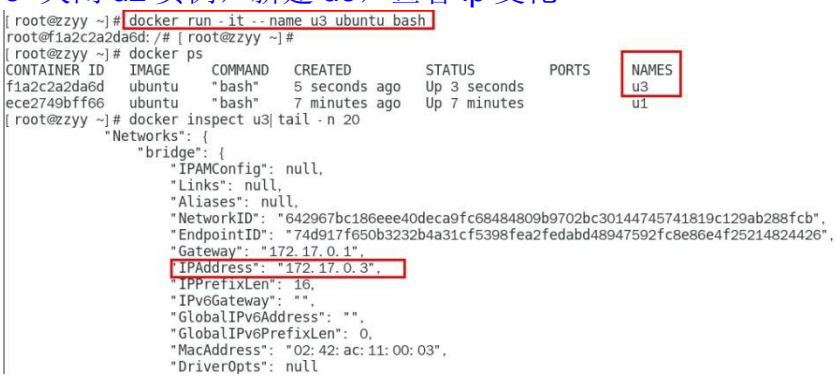
  - bridge 模式：使用--network bridge 指定，默认使用 docker0
  - host 模式：使用--network host 指定
  - none 模式：使用--network none 指定
  - container 模式：使用--network container:NAME 或者容器 ID 指定
- 容器实例内默认网络 IP 生产规则
  - 说明

1 先启动两个 ubuntu 容器实例

```
[root@zzyy ~]# docker run -it --name u1 ubuntu bash
root@ece2749bff66:/# [root@zzyy ~]#
[root@zzyy ~]#
[root@zzyy ~]# docker run -it --name u2 ubuntu bash
root@1229e931c85b:/# [root@zzyy ~]#
[root@zzyy ~]#
[root@zzyy ~]# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED          STATUS          PORTS     NAMES
1229e931c85b   ubuntu    "bash"    5 seconds ago    Up 3 seconds              u2
ece2749bff66   ubuntu    "bash"    16 seconds ago   Up 15 seconds             u1
[root@zzyy ~]#
```

2 docker inspect 容器 ID or 容器名字

3 关闭 u2 实例，新建 u3，查看 ip 变化



- 结论
- docker 容器内部的 ip 是有可能会发生改变的

- **案例说明**
  - bridge
  - 是什么

Docker 服务默认会创建一个 docker0 网桥（其上有一个 docker0 内部接口），该桥接网络的名称为 docker0，它在内核层连通了其他的物理或虚拟网卡，这就将所有容器和本地主机都放到同一个物理网络。Docker 默认指定了 docker0 接口 的 IP 地址和子网掩码，让主机和容器之间可以通过网桥相互通信。

查看 bridge 网络的详细信息，并通过 grep 获取名称项
docker network inspect bridge | grep name

```
[root@zzyy tmp]# docker network inspect bridge | grep name
            "com.docker.network.bridge.name": "docker0",
[root@zzyy tmp]#
```

ifconfig

```
[root@zzyy tmp]# ifconfig | grep docker
docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
[root@zzyy tmp]#
```

- 案例
- 说明

1 Docker 使用 Linux 桥接，在宿主机虚拟一个 Docker 容器网桥(docker0)，Docker 启动一个容器时会根据 Docker 网桥的网段分配给容器一个 IP 地址，称为 Container-IP，同时 Docker 网桥是每个容器的默认网关。因为在同一宿主机内的容器都接入同一个网桥，这样容器之间就能够通过容器的 Container-IP 直接通信。

2 docker run 的时候，没有指定 network 的话默认使用的网桥模式就是 bridge，使用的就是 docker0。在宿主机 ifconfig,就可以看到 docker0 和自己 create 的 network(后面讲)eth0，eth1，eth2……代表网卡一，网卡二，网卡三……，lo 代表 127.0.0.1，即 localhost，inet addr 用来表示网卡的 IP 地址

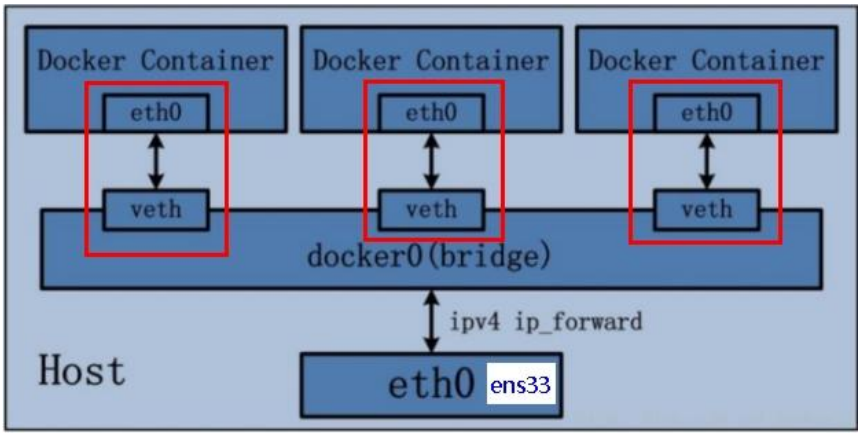
3 网桥 docker0 创建一对对等虚拟设备接口一个叫 veth，另一个叫 eth0，成对匹配。

3.1 整个宿主机的网桥模式都是 docker0，类似一个交换机有一堆接口，每个接口叫 veth，在本地主机和容器内分别创建一个虚拟接口，并让他们彼此联通（这样一对接口叫 veth pair）；

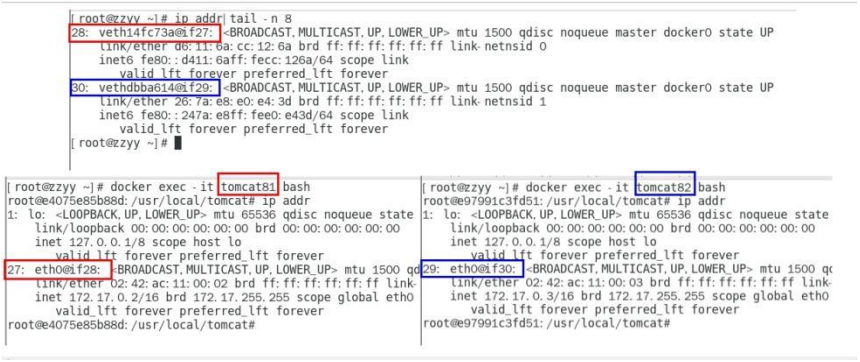
3.2 每个容器实例内部也有一块网卡，每个接口叫 eth0；

3.3 docker0 上面的每个 veth 匹配某个容器实例内部的 eth0，两两配对，一一匹配。

通过上述，将宿主机上的所有容器都连接到这个内部网络上，两个容器在同一个网络下,会从这个网关下各自拿到分配的 ip，此时两个容器的网络是互通的。



- 代码
- docker run -d -p 8081:8080 --name tomcat81 billygoo/tomcat8-jdk8
- docker run -d -p 8082:8080 --name tomcat82 billygoo/tomcat8-jdk8
- 两两匹配验证



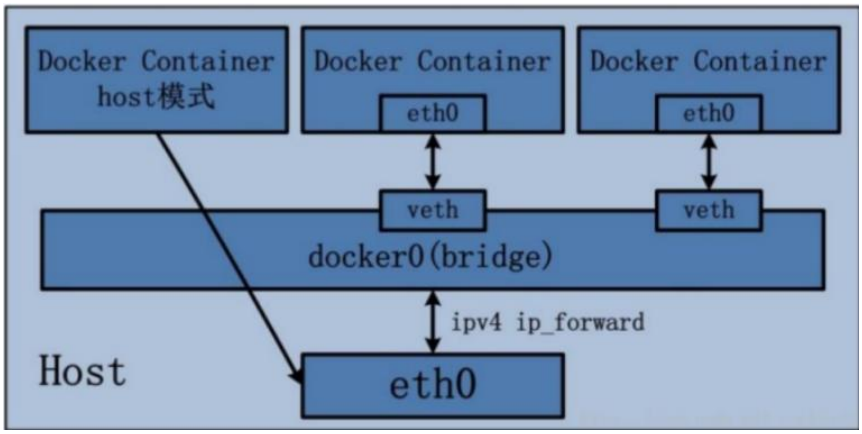
- host
- 是什么

直接使用宿主机的 IP 地址与外界进行通信，不再需要额外进行 NAT 转换。

- 案例

- 说明

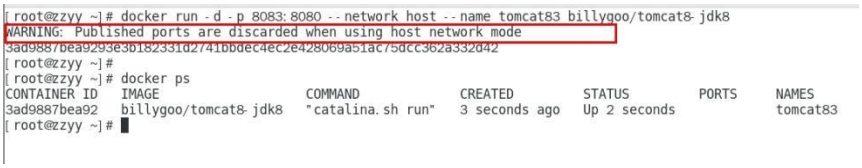
容器将不会获得一个独立的 Network Namespace，而是和宿主机共用一个 Network Namespace。容器将不会虚拟出自己的网卡而是使用宿主机的 IP 和端口。



- 代码

- 警告

- docker run -d -p 8083:8080 --network host --name tomcat83 billygoo/tomcat8-jdk8



问题：

docke 启动时总是遇见标题中的警告

原因：

docker 启动时指定--network=host 或-net=host，如果还指定了-p 映射端口，那这个时候就会有此警告，

并且通过-p 设置的参数将不会起到任何作用，端口号会以主机端口号为主，重复时则递增。

解决:

解决的办法就是使用 docker 的其他网络模式，例如--network=bridge，这样就可以解决问题，或者直接无视。。。。O(∩_∩)O 哈哈~

- 正确
- docker run -d --network host --name tomcat83 billygoo/tomcat8-jdk8
- 无之前的配对显示了，看容器实例内部

- 没有设置-p 的端口映射了，如何访问启动的 tomcat83？？

http://宿主机 IP:8080/

在 CentOS 里面用默认的火狐浏览器访问容器内的 tomcat83 看到访问成功，因为此时容器的 IP 借用主机的，
所以容器共享宿主机网络 IP，这样的好处是外部主机与容器可以直接通信。

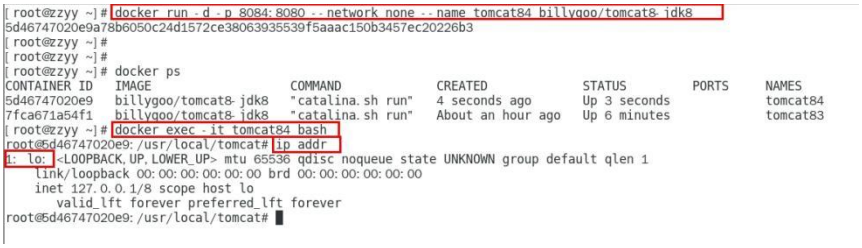
- none
- 是什么

在 none 模式下，并不为 Docker 容器进行任何网络配置。
也就是说，这个 Docker 容器没有网卡、IP、路由等信息，只有一个 lo
需要我们自己为 Docker 容器添加网卡、配置 IP 等。

- 禁用网络功能，只有 lo 标识(就是 127.0.0.1 表示本地回环)
- 案例

```
docker run -d -p 8084:8080 --network none --name tomcat84 billygoo/tomcat8-jdk8
```

进入容器内部查看



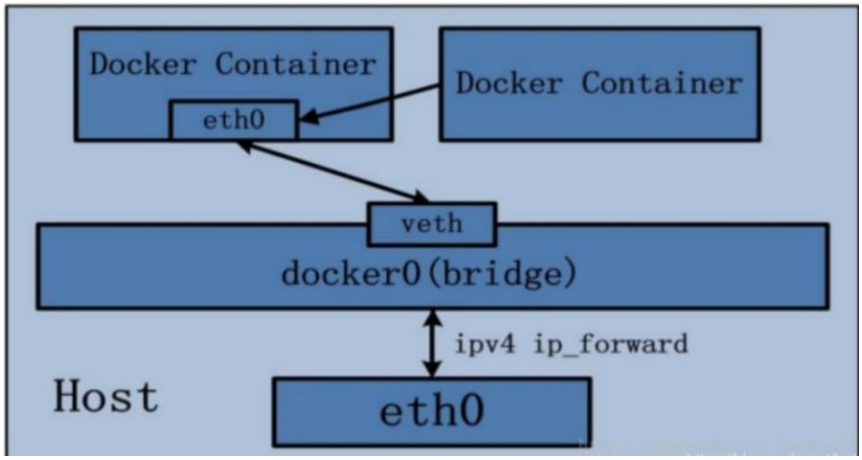
在容器外部查看

```
[root@zzyy ~]# docker inspect tomcat84|tail -n 20
            "Networks": {
                "none": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID": "3bc4d8335e637398d3a73bdebbb943f68d6
                    "EndpointID": "1ecbf8b68e6c47e9d3949f125e4f801a4c
                    "Gateway": "",
                    "IPAddress": "",
                    "IPPrefixLen": 0,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "",
                    "DriverOpts": null
                }
            }
        }
    }
]
[root@zzyy ~]#
```

- docker run -d -p 8084:8080 --network none --name tomcat84 billygoo/tomcat8-jdk8
- container
- 是什么

container网络模式

新建的容器和已经存在的一个容器共享一个网络 ip 配置而不是和宿主机共享。新创建的容器不会创建自己的网卡，配置自己的 IP，而是和一个指定的容器共享 IP、端口范围等。同样，两个容器除了网络方面，其他的如文件系统、进程列表等还是隔离的。



- 案例

单图标 CustomIcon-663735520

- docker run -d -p 8085:8080 --name tomcat85 billygoo/tomcat8-jdk8
- docker run -d -p 8086:8080 --network container:tomcat85 --name tomcat86 billygoo/tomcat8-jdk8
- 运行结果

```
[root@zzyy ~]# docker run -d -p 8086:8080 --network container:tomcat85 --name tomcat86 billygoo/tomcat8-jdk8
docker: Error response from daemon: conflicting options: port publishing and the container type network mode.
See 'docker run --help'.
[root@zzyy ~]#
```

相当于 tomcat86 和 tomcat85 公用同一个 ip 同一个端口，导致端口冲突

本案例用 tomcat 演示不合适。。。演示坑。。。。。。o(╥﹏╥)o

换一个镜像给大家演示，

- 案例 2

单图标 CustomIcon--1664269521

- Alpine 操作系统是一个面向安全的轻型 Linux 发行版

Alpine Linux 是一款独立的、非商业的通用 Linux 发行版，专为追求安全性、简单性和资源效率的用户而设计。 可能很多人没听说过这个 Linux 发行版本，但是经常用 Docker 的

朋友可能都用过，因为他小，简单，安全而著称，所以作为基础镜像是非常好的一个选择，可谓是麻雀虽小但五脏俱全，镜像非常小巧，不到 6M 的大小，所以特别适合容器打包。

- docker run -it --name alpine1 alpine /bin/sh
- docker run -it --network container:alpine1 --name alpine2 alpine /bin/sh
- 运行结果，验证共用搭桥

```
[root@zzyy ~]# docker run -it --name alpine1  alpine /bin/sh
/ # ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
15: eth0@if16: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
/ #
```

```
[root@zzyy ~]# docker run -it --network container:alpine1 --name alpine2  alpine /bin/sh
/ # ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
15: eth0@if16: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
/ #
```

- 假如此时关闭 alpine1，再看看 alpine2

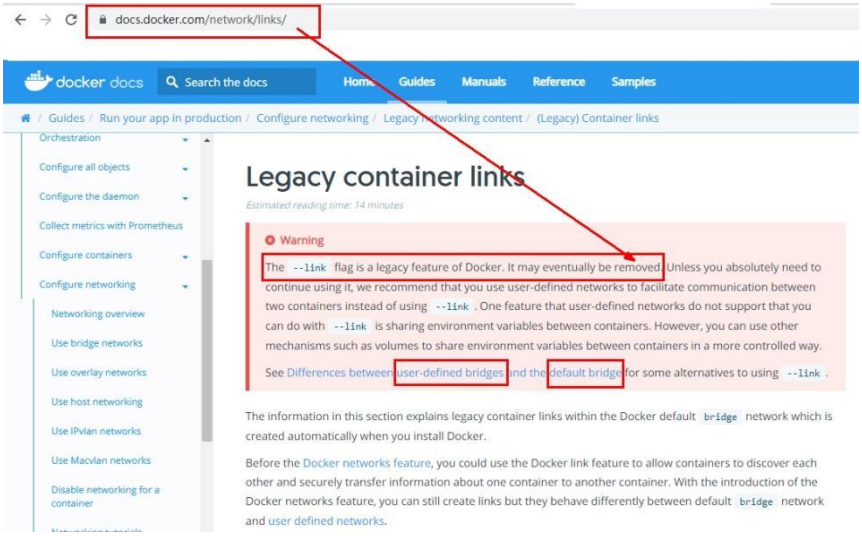
```
[root@zzyy ~]# docker ps
CONTAINER ID   IMAGE     COMMAND     CREATED         STATUS         PORTS     NAMES
c7a184e29941   alpine    "/bin/sh"   5 minutes ago   Up 5 minutes             alpine2
[root@zzyy ~]#
```

15: eth0@if16: 消失了。。。。。。关闭 alpine1，再看看 alpine2

```
[root@zzyy ~]# docker run -it --network container:alpine1 --name alpine2  alpine /bin/sh
/ # ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
15: eth0@if16: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
/ #
/ # ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
/ #
```

- 自定义网络

- 过时的 link



- 是什么
- 案例
- before
- 案例
- docker run -d -p 8081:8080 --name tomcat81 billygoo/tomcat8-jdk8
- docker run -d -p 8082:8080 --name tomcat82 billygoo/tomcat8-jdk8
- 上述成功启动并用 docker exec 进入各自容器实例内部
- 问题
- 按照 IP 地址 ping 是 OK 的

```
root@de7ac8b2d203:/usr/local/tomcat# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
27: eth0@if28: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:04 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.4/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
root@de7ac8b2d203:/usr/local/tomcat# ping 172.17.0.5
PING 172.17.0.5 (172.17.0.5) 56(84) bytes of data.
64 bytes from 172.17.0.5: icmp_seq=1 ttl=64 time=0.056 ms
64 bytes from 172.17.0.5: icmp_seq=2 ttl=64 time=0.048 ms
root@8ceacd7f8f97:/usr/local/tomcat# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
29: eth0@if30: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:05 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.5/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
root@8ceacd7f8f97:/usr/local/tomcat# ping 172.17.0.4
PING 172.17.0.4 (172.17.0.4) 56(84) bytes of data.
64 bytes from 172.17.0.4: icmp_seq=1 ttl=64 time=0.059 ms
64 bytes from 172.17.0.4: icmp_seq=2 ttl=64 time=0.061 ms
```

- 按照服务名 ping 结果???

```
root@de7ac8b2d203: /usr/local/tomcat# ping tomcat82
ping: tomcat82: Name or service not known
root@de7ac8b2d203: /usr/local/tomcat#
```

```
root@8ceacd7f8f97: /usr/local/tomcat# ping tomcat81
ping: tomcat81: Name or service not known
root@8ceacd7f8f97: /usr/local/tomcat#
```

- after
- 案例
- 自定义桥接网络,自定义网络默认使用的是桥接网络 bridge
- 新建自定义网络

```
[root@zzyy ~]# docker network ls
NETWORK ID     NAME              DRIVER    SCOPE
319c16b92bd7   bridge            bridge    local

ae13c04d7626   host              host      local

3bc4d8335e63   none              null      local
[root@zzyy ~]# docker network create zzyy_network
c8a058b763df8044e7fd6c7a6740115bd5e61ff6656a2c747a11229d8eca558b
[root@zzyy ~]# docker network ls
NETWORK ID     NAME              DRIVER    SCOPE
319c16b92bd7   bridge            bridge    local

ae13c04d7626   host              host      local

3bc4d8335e63   none              null      local
c8a058b763df   zzyy_network      bridge    local
[root@zzyy ~]#
```

- 新建容器加入上一步新建的自定义网络
- docker run -d -p 8081:8080 --network zzyy_network --name tomcat81 billygoo/tomcat8-jdk8
- docker run -d -p 8082:8080 --network zzyy_network --name tomcat82 billygoo/tomcat8-jdk8
- 互相 ping 测试

```
[root@zzyy ~]# docker exec -it tomcat81 bash
root@f6a5b55bbfb1:/usr/local/tomcat# ping tomcat82
PING tomcat82 (172.20.0.3) 56(84) bytes of data.
64 bytes from tomcat82.zzyy_network (172.20.0.3): icmp_seq=1 ttl=64 time=0.099 ms
64 bytes from tomcat82.zzyy_network (172.20.0.3): icmp_seq=2 ttl=64 time=0.198 ms
64 bytes from tomcat82.zzyy_network (172.20.0.3): icmp_seq=3 ttl=64 time=0.111 ms
^C
--- tomcat82 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.099/0.136/0.198/0.044 ms
root@f6a5b55bbfb1:/usr/local/tomcat#

[root@zzyy ~]# docker exec -it tomcat82 bash
root@d4359ff6df38:/usr/local/tomcat# ping tomcat81
PING tomcat81 (172.20.0.2) 56(84) bytes of data.
64 bytes from tomcat81.zzyy_network (172.20.0.2): icmp_seq=1 ttl=64 time=0.038 ms
64 bytes from tomcat81.zzyy_network (172.20.0.2): icmp_seq=2 ttl=64 time=0.116 ms
^C
--- tomcat81 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.038/0.077/0.116/0.039 ms
root@d4359ff6df38:/usr/local/tomcat#
```

- 问题结论

- 自定义网络本身就维护好了主机名和 ip 的对应关系（ip 和域名都能通）

- 自定义网络本身就维护好了主机名和 ip 的对应关系（ip 和域名都能通）

- 自定义网络本身就维护好了主机名和 ip 的对应关系（ip 和域名都能通）

2.4.5. Docker 平台架构图解

- 整体说明

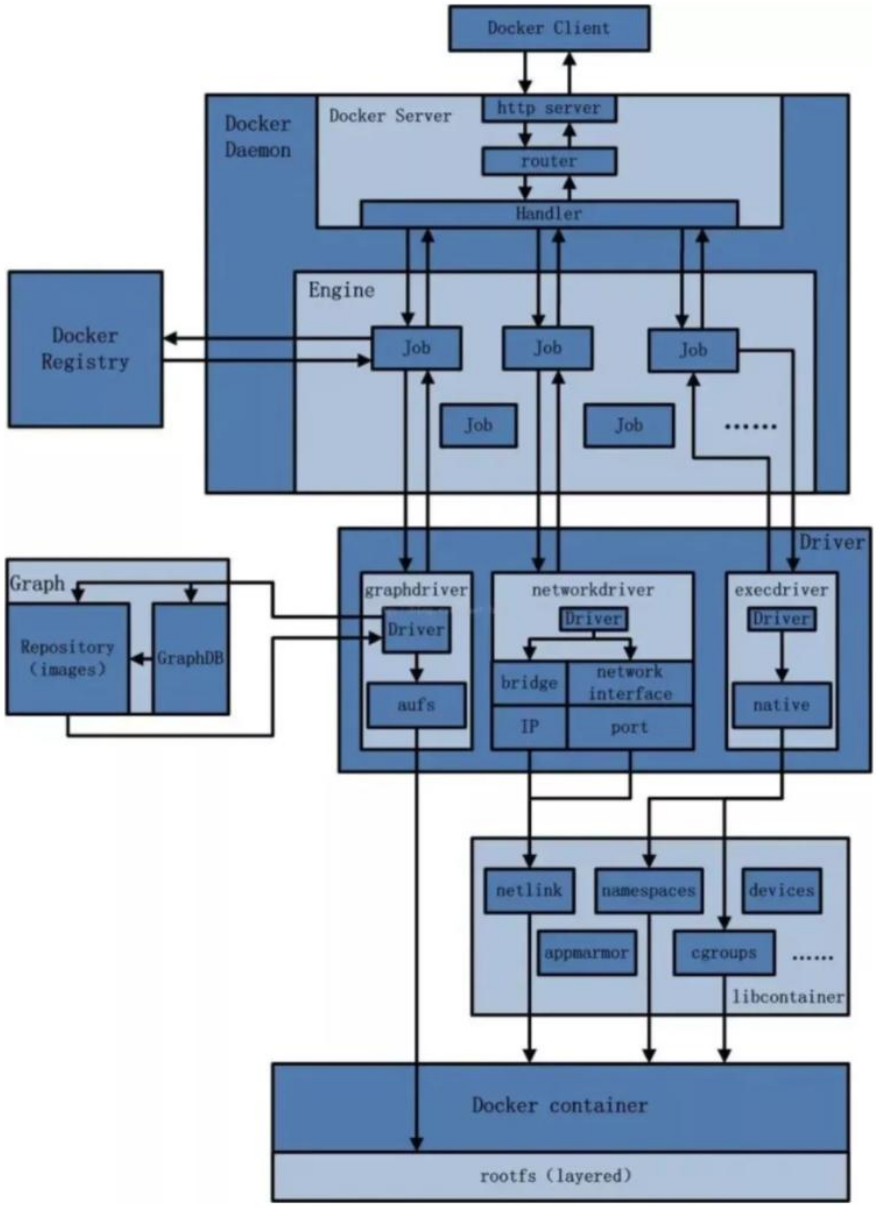
从其架构和运行流程来看，Docker 是一个 C/S 模式的架构，后端是一个松耦合架构，众多模块各司其职。

Docker 运行的基本流程为：

1 用户是使用 Docker Client 与 Docker Daemon 建立通信，并发送请求给后者。
2 Docker Daemon 作为 Docker 架构中的主体部分，首先提供 Docker Server 的功能使其可以接受 Docker Client 的请求。
3 Docker Engine 执行 Docker 内部的一系列工作，每一项工作都是以一个 Job 的形式的存在。
4 Job 的运行过程中，当需要容器镜像时，则从 Docker Registry 中下载镜像，并通过镜像管理驱动 Graph driver 将下载镜像以 Graph 的形式存储。
5 当需要为 Docker 创建网络环境时，通过网络管理驱动 Network driver 创建并配置 Docker 容器网络环境。
6 当需要限制 Docker 容器运行资源或执行用户指令等操作时，则通过 Execdriver 来完成。

7 Libcontainer 是一项独立的容器管理包，Network driver 以及 Exec driver 都是通过 Libcontainer 来实现具体对容器进行的操作。

· 整体架构



2.5. Docker-compose 容器编排

2.5.1. 是什么

Compose 是 Docker 公司推出的一个工具软件，可以管理多个 Docker 容器组成一个应用。你需要定义一个 YAML 格式的配置文件 docker-compose.yml，写好多个容器之间的调用关系。然后，只要一个命令，就能同时启动/关闭这些容器

- Docker-Compose 是 Docker 官方的开源项目， 负责实现对 Docker 容器集群的快速编排。

2.5.2. 能干嘛

docker 建议我们每一个容器中只运行一个服务,因为 docker 容器本身占用资源极少,所以最好是将每个服务单独的分割开来但是这样我们又面临了一个问题？

如果我需要同时部署好多个服务,难道要每个服务单独写 Dockerfile 然后在构建镜像,构建容器,这样累都累死了,所以 docker 官方给我们提供了 docker-compose 多服务部署的工具

例如要实现一个 Web 微服务项目，除了 Web 服务容器本身，往往还需要再加上后端的数据库 mysql 服务容器，redis 服务器，注册中心 eureka，甚至还包括负载均衡容器等等。。。。。。

Compose 允许用户通过一个单独的 docker-compose.yml 模板文件（YAML 格式）来定义一组相关联的应用容器为一个项目（project）。

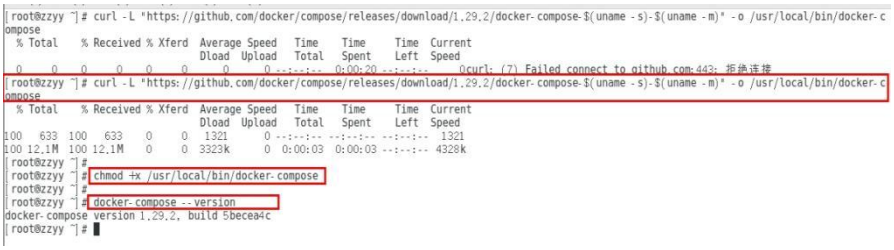
可以很容易地用一个配置文件定义一个多容器的应用，然后使用一条指令安装这个应用的所有依赖，完

成构建。Docker-Compose 解决了容器与容器之间如何管理编排的问题。

2.5.3. 去哪下

- 官网
  - https://docs.docker.com/compose/compose-file/compose-file-v3/
- 官网下载
  - https://docs.docker.com/compose/install/
- 安装步骤

curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
chmod +x /usr/local/bin/docker-compose
docker-compose --version



- 卸载步骤

2.5.4. Compose 核心概念

- 一文件

- docker-compose.yml

  - 两要素

    - 服务（service）

    - 一个个应用容器实例，比如订单微服务、库存微服务、mysql 容器、nginx 容器或者 redis 容器

    - 工程（project）

    - 由一组关联的应用容器组成的一个完整业务单元，在 docker-compose.yml 文件中定义。

2.5.5. Compose 使用的三个步骤

- 编写 Dockerfile 定义各个微服务应用并构建出对应的镜像文件

- 使用 docker-compose.yml 定义一个完整业务单元，安排好整体应用中的各个容器服务。

- 最后，执行 docker-compose up 命令 来启动并运行整个应用程序，完成一键部署上线

2.5.6. Compose 常用命令

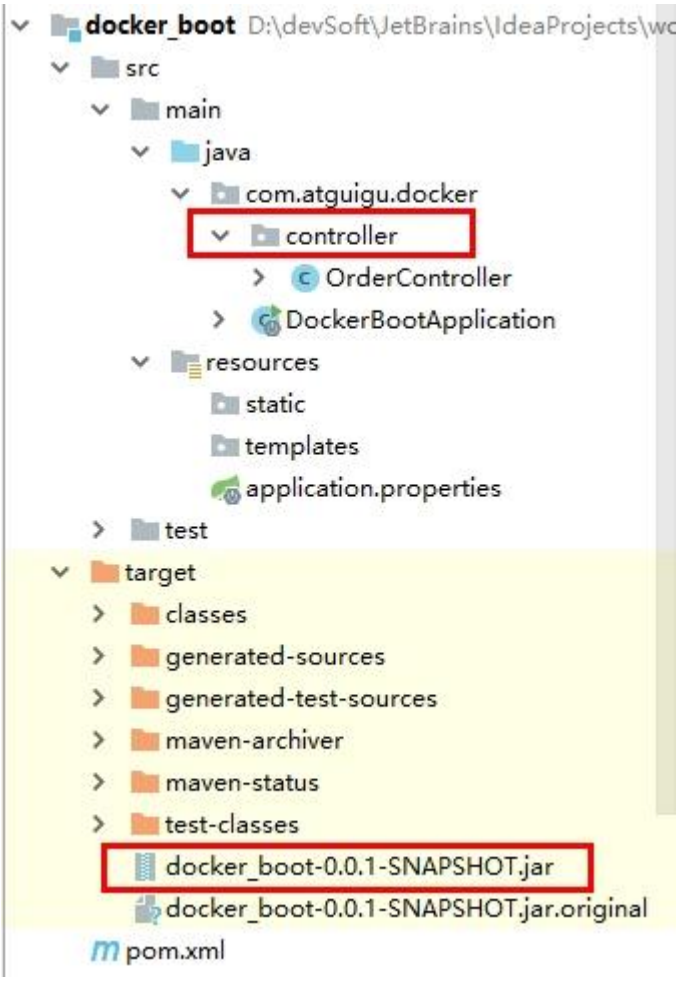
Compose 常用命令

```
docker-compose -h                    # 查看帮助
docker-compose up                    # 启动所有 docker-compose 服务
docker-compose up -d                 # 启动所有 docker-compose 服务并后台运行
docker-compose down                  # 停止并删除容器、网络、卷、镜像。
docker-compose exec  yml 里面的服务 id          # 进入容器实例内部  docker-compose exec docker-compose.yml 文件中写的服务 id /bin/bash
docker-compose ps                    # 展示当前 docker-compose 编排过的运行的所有容器
docker-compose top                   # 展示当前 docker-compose 编排过的容器进程
```

docker-compose logs yml 里面的服务 id　# 查看容器输出日志
docker-compose config　# 检查配置
docker-compose config -q # 检查配置，有问题才有输出
docker-compose restart　# 重启服务
docker-compose start　# 启动服务
docker-compose stop　# 停止服务

2.5.7. Compose 编排微服务

- 改造升级微服务工程 docker_boot
  - 以前的基础版



- SQL 建表建库

```
CREATE TABLE `t_user` (
  `id` int(10) unsigned NOT NULL
AUTO_INCREMENT,
  `username` varchar(50) NOT NULL DEFAULT ''
COMMENT '用户名',
  `password` varchar(50) NOT NULL DEFAULT ''
COMMENT '密码',
  `sex` tinyint(4) NOT NULL DEFAULT '0'
COMMENT '性别 0=女 1=男 ',
  `deleted` tinyint(4) unsigned NOT NULL
DEFAULT '0' COMMENT '删除标志，默认 0 不删
除，1 删除',
  `update_time` timestamp NOT NULL DEFAULT
CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP COMMENT '更新时间',
  `create_time` timestamp NOT NULL DEFAULT
CURRENT_TIMESTAMP COMMENT '创建时间',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1
DEFAULT CHARSET=utf8 COMMENT='用户表'
```

- 一键生成说明

- 改 POM

```
<?xml version="1.0" encoding="UTF-8"?>
<project
xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
        xsi:schemaLocation="http://maven.apache.or
g/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
```

```
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-
parent</artifactId>
        <version>2.5.6</version>
        <!--<version>2.3.10.RELEASE</version>-->
        <relativePath/> <!-- lookup parent from
repository -->
    </parent>

    <groupId>com.atguigu.docker</groupId>
    <artifactId>docker_boot</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <properties>
        <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.
source>
        <maven.compiler.target>1.8</maven.compiler.t
arget>
        <junit.version>4.12</junit.version>
        <log4j.version>1.2.17</log4j.version>
        <lombok.version>1.16.18</lombok.version>
        <mysql.version>5.1.47</mysql.version>
        <druid.version>1.1.16</druid.version>
```

```
        <mapper.version>4.1.5</mapper.version>
        <mybatis.spring.boot.version>1.3.0</mybatis.spring.boot.version>
    </properties>

    <dependencies>
        <!--guava Google 开源的 Guava 中自带的布隆过滤器-->
        <dependency>
            <groupId>com.google.guava</groupId>
            <artifactId>guava</artifactId>
            <version>23.0</version>
        </dependency>
        <!-- redisson -->
        <dependency>
            <groupId>org.redisson</groupId>
            <artifactId>redisson</artifactId>
            <version>3.13.4</version>
        </dependency>
        <!--SpringBoot 通用依赖模块-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
```

```
            <groupId>org.springframework.boot</group
Id>
            <artifactId>spring-boot-starter-
actuator</artifactId>
        </dependency>
        <!--swagger2-->
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-
swagger2</artifactId>
            <version>2.9.2</version>
        </dependency>
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger-
ui</artifactId>
            <version>2.9.2</version>
        </dependency>
        <!--SpringBoot 与 Redis 整合依赖-->
        <dependency>
            <groupId>org.springframework.boot</group
Id>
            <artifactId>spring-boot-starter-data-
redis</artifactId>
        </dependency>
        <!--springCache-->
        <dependency>
```

```
            <groupId>org.springframework.boot</group
Id>
            <artifactId>spring-boot-starter-
cache</artifactId>
        </dependency>
        <!--springCache 连接池依赖包-->
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-pool2</artifactId>
        </dependency>
        <!-- jedis -->
        <dependency>
            <groupId>redis.clients</groupId>
            <artifactId>jedis</artifactId>
            <version>3.1.0</version>
        </dependency>
        <!--Mysql 数据库驱动-->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-
java</artifactId>
            <version>5.1.47</version>
        </dependency>
        <!--SpringBoot 集成 druid 连接池-->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid-spring-boot-
```

```
starter</artifactId>
            <version>1.1.10</version>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>${druid.version}</version>
        </dependency>
        <!--mybatis 和 springboot 整合-->
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId
>
            <artifactId>mybatis-spring-boot-
starter</artifactId>
            <version>${mybatis.spring.boot.version}</ve
rsion>
        </dependency>
        <!-- 添加 springboot 对 amqp 的支持 -->
        <dependency>
            <groupId>org.springframework.boot</group
Id>
            <artifactId>spring-boot-starter-
amqp</artifactId>
        </dependency>
        <dependency>
            <groupId>commons-codec</groupId>
            <artifactId>commons-codec</artifactId>
```

```
            <version>1.10</version>
        </dependency>
        <!--通用基础配置
junit/devtools/test/log4j/lombok/hutool-->
        <!--hutool-->
        <dependency>
            <groupId>cn.hutool</groupId>
            <artifactId>hutool-all</artifactId>
            <version>5.2.3</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>${junit.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</group
Id>
            <artifactId>spring-boot-
devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</group
Id>
            <artifactId>spring-boot-starter-
```

```
test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>${log4j.version}</version>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>${lombok.version}</version>
            <optional>true</optional>
        </dependency>
        <!--persistence-->
        <dependency>
            <groupId>javax.persistence</groupId>
            <artifactId>persistence-api</artifactId>
            <version>1.0.2</version>
        </dependency>
        <!--通用 Mapper-->
        <dependency>
            <groupId>tk.mybatis</groupId>
            <artifactId>mapper</artifactId>
            <version>${mapper.version}</version>
        </dependency>
    </dependencies>
```

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</gr
oupId>
                <artifactId>spring-boot-maven-
plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</gr
oupId>
                <artifactId>maven-resources-
plugin</artifactId>
                <version>3.1.0</version>
            </plugin>
        </plugins>
    </build>

</project>
```

- 写 YML

```
server.port=6001
# ========================alibaba.druid 相关配
置=====================
spring.datasource.type=com.alibaba.druid.pool.Druid
```

```
DataSource
spring.datasource.driver-class-
name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://192.168.111.169:3
306/db2021?useUnicode=true&characterEncoding=
utf-8&useSSL=false
spring.datasource.username=root
spring.datasource.password=123456
spring.datasource.druid.test-while-idle=false
# =========================redis 相关配置
=====================
spring.redis.database=0
spring.redis.host=192.168.111.169
spring.redis.port=6379
spring.redis.password=
spring.redis.lettuce.pool.max-active=8
spring.redis.lettuce.pool.max-wait=-1ms
spring.redis.lettuce.pool.max-idle=8
spring.redis.lettuce.pool.min-idle=0
# ========================mybatis 相关配置
====================
mybatis.mapper-locations=classpath:mapper/*.xml
mybatis.type-aliases-
package=com.atguigu.docker.entities
#
=========================swagger==========
```

```
==========
spring.swagger2.enabled=true
```

- 主启动

```
package com.atguigu.docker;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBoot
Application;
import tk.mybatis.spring.annotation.MapperScan;

@SpringBootApplication
@MapperScan("com.atguigu.docker.mapper")
//import tk.mybatis.spring.annotation.MapperScan;
public class DockerBootApplication
{
    public static void main(String[] args)
    {
        SpringApplication.run(DockerBootApplication.cl
ass, args);
    }

}
```

- 业务类

- config 配置类
- RedisConfig

```
package com.atguigu.docker.config;

import lombok.extern.slf4j.Slf4j;
import org.springframework.context.annotation.Bean;
import
org.springframework.context.annotation.Configuratio
n;
import
org.springframework.data.redis.connection.lettuce.L
ettuceConnectionFactory;
import
org.springframework.data.redis.core.RedisTemplate;
import
org.springframework.data.redis.serializer.GenericJac
kson2JsonRedisSerializer;
import
org.springframework.data.redis.serializer.StringRedis
Serializer;

import java.io.Serializable;

/**
 * @auther zzyy
 * @create 2021-10-27 17:19
```

```
 */

@Configuration
@Slf4j
public class RedisConfig
{
    /**
     * @param lettuceConnectionFactory
     * @return
     *
     * redis 序列化的工具配置类，下面这个请一定开启配置
     * 127.0.0.1:6379> keys *
     * 1) "ord:102" 序列化过
     * 2) "\xac\xed\x00\x05t\x00\aord:102" 野生，没有序列化过
     */
    @Bean
    public RedisTemplate<String,Serializable> redisTemplate(LettuceConnectionFactory lettuceConnectionFactory)
    {
        RedisTemplate<String,Serializable> redisTemplate = new RedisTemplate<>();

        redisTemplate.setConnectionFactory(lettuceConnectionFactory);
```

```
        //设置 key 序列化方式 string
        redisTemplate.setKeySerializer(new
StringRedisSerializer());
        //设置 value 的序列化方式 json
        redisTemplate.setValueSerializer(new
GenericJackson2JsonRedisSerializer());

        redisTemplate.setHashKeySerializer(new
StringRedisSerializer());
        redisTemplate.setHashValueSerializer(new
GenericJackson2JsonRedisSerializer());

        redisTemplate.afterPropertiesSet();

        return redisTemplate;
    }

}
```

- SwaggerConfig

```
package com.atguigu.docker.config;

import
org.springframework.beans.factory.annotation.Value;
```

```
import org.springframework.context.annotation.Bean;
import
org.springframework.context.annotation.Configuratio
n;
import
springfox.documentation.builders.ApiInfoBuilder;
import
springfox.documentation.builders.PathSelectors;
import
springfox.documentation.builders.RequestHandlerSel
ectors;
import springfox.documentation.service.ApiInfo;
import
springfox.documentation.spi.DocumentationType;
import
springfox.documentation.spring.web.plugins.Docket;
import
springfox.documentation.swagger2.annotations.Enab
leSwagger2;

import java.text.SimpleDateFormat;
import java.util.Date;

/**
 * @auther zzyy
 * @create 2021-05-01 16:18
 */
```

```
@Configuration
@EnableSwagger2
public class SwaggerConfig
{
    @Value("${spring.swagger2.enabled}")
    private Boolean enabled;

    @Bean
    public Docket createRestApi() {
        return new
Docket(DocumentationType.SWAGGER_2)
                .apiInfo(apiInfo())
                .enable(enabled)
                .select()
                .apis(RequestHandlerSelectors.basePack
age("com.atguigu.docker")) //你自己的 package
                .paths(PathSelectors.any())
                .build();
    }

    public ApiInfo apiInfo() {
        return new ApiInfoBuilder()
                .title("尚硅谷 Java 大厂技术"+"\t"+new
SimpleDateFormat("yyyy-MM-dd").format(new
Date()))
                .description("docker-compose")
                .version("1.0")
```

```
                .termsOfServiceUrl("https://www.atguigu.
com/")
                .build();
    }
}
```

- 新建 entity
- User

```
package com.atguigu.docker.entities;

import javax.persistence.Column;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
import java.util.Date;

@Table(name = "t_user")
public class User
{
    @Id
    @GeneratedValue(generator = "JDBC")
    private Integer id;

    /**
```

```
     * 用户名
     */
    private String username;

    /**
     * 密码
     */
    private String password;

    /**
     * 性别 0=女 1=男
     */
    private Byte sex;

    /**
     * 删除标志，默认 0 不删除，1 删除
     */
    private Byte deleted;

    /**
     * 更新时间
     */
    @Column(name = "update_time")
    private Date updateTime;

    /**
     * 创建时间
```

```
     */
    @Column(name = "create_time")
    private Date createTime;

    /**
     * @return id
     */
    public Integer getId() {
        return id;
    }

    /**
     * @param id
     */
    public void setId(Integer id) {
        this.id = id;
    }

    /**
     * 获取用户名
     *
     * @return username - 用户名
     */
    public String getUsername() {
        return username;
    }
```

```java
    /**
     * 设置用户名
     *
     * @param username 用户名
     */
    public void setUsername(String username) {
        this.username = username;
    }

    /**
     * 获取密码
     *
     * @return password - 密码
     */
    public String getPassword() {
        return password;
    }

    /**
     * 设置密码
     *
     * @param password 密码
     */
    public void setPassword(String password) {
        this.password = password;
    }
```

```
    /**
     * 获取性别 0=女 1=男
     *
     * @return sex - 性别 0=女 1=男
     */
    public Byte getSex() {
        return sex;
    }

    /**
     * 设置性别 0=女 1=男
     *
     * @param sex 性别 0=女 1=男
     */
    public void setSex(Byte sex) {
        this.sex = sex;
    }

    /**
     * 获取删除标志，默认 0 不删除，1 删除
     *
     * @return deleted - 删除标志，默认 0 不删除，1
删除
     */
    public Byte getDeleted() {
        return deleted;
    }
```

```
    /**
     * 设置删除标志，默认 0 不删除，1 删除
     *
     * @param deleted 删除标志，默认 0 不删除，1 删除
     */
    public void setDeleted(Byte deleted) {
        this.deleted = deleted;
    }

    /**
     * 获取更新时间
     *
     * @return update_time – 更新时间
     */
    public Date getUpdateTime() {
        return updateTime;
    }

    /**
     * 设置更新时间
     *
     * @param updateTime 更新时间
     */
    public void setUpdateTime(Date updateTime) {
        this.updateTime = updateTime;
```

```
    }

    /**
     * 获取创建时间
     *
     * @return create_time - 创建时间
     */
    public Date getCreateTime() {
        return createTime;
    }

    /**
     * 设置创建时间
     *
     * @param createTime 创建时间
     */
    public void setCreateTime(Date createTime) {
        this.createTime = createTime;
    }
}
```

- UserDTO

```
package com.atguigu.docker.entities;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
```

```
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.io.Serializable;
import java.util.Date;

@NoArgsConstructor
@AllArgsConstructor
@Data
@ApiModel(value = "用户信息")
public class UserDTO implements Serializable
{
    @ApiModelProperty(value = "用户 ID")
    private Integer id;

    @ApiModelProperty(value = "用户名")
    private String username;

    @ApiModelProperty(value = "密码")
    private String password;

    @ApiModelProperty(value = "性别 0=女 1=男 ")
    private Byte sex;

    @ApiModelProperty(value = "删除标志，默认 0 不
删除，1 删除")
```

```
    private Byte deleted;

    @ApiModelProperty(value = "更新时间")
    private Date updateTime;

    @ApiModelProperty(value = "创建时间")
    private Date createTime;

    /**
     * @return id
     */
    public Integer getId() {
        return id;
    }

    /**
     * @param id
     */
    public void setId(Integer id) {
        this.id = id;
    }

    /**
     * 获取用户名
     *
     * @return username - 用户名
     */
```

```
public String getUsername() {
    return username;
}

/**
 * 设置用户名
 *
 * @param username 用户名
 */
public void setUsername(String username) {
    this.username = username;
}

/**
 * 获取密码
 *
 * @return password - 密码
 */
public String getPassword() {
    return password;
}

/**
 * 设置密码
 *
 * @param password 密码
 */
```

```
    public void setPassword(String password) {
        this.password = password;
    }

    /**
     * 获取性别 0=女 1=男
     *
     * @return sex - 性别 0=女 1=男
     */
    public Byte getSex() {
        return sex;
    }

    /**
     * 设置性别 0=女 1=男
     *
     * @param sex 性别 0=女 1=男
     */
    public void setSex(Byte sex) {
        this.sex = sex;
    }

    /**
     * 获取删除标志，默认 0 不删除，1 删除
     *
     * @return deleted - 删除标志，默认 0 不删除，1
删除
```

```
     */
    public Byte getDeleted() {
        return deleted;
    }

    /**
     * 设置删除标志，默认 0 不删除，1 删除
     *
     * @param deleted 删除标志，默认 0 不删除，1 删除
     */
    public void setDeleted(Byte deleted) {
        this.deleted = deleted;
    }

    /**
     * 获取更新时间
     *
     * @return update_time – 更新时间
     */
    public Date getUpdateTime() {
        return updateTime;
    }

    /**
     * 设置更新时间
     *
```

```
     * @param updateTime 更新时间
     */
    public void setUpdateTime(Date updateTime) {
        this.updateTime = updateTime;
    }

    /**
     * 获取创建时间
     *
     * @return create_time - 创建时间
     */
    public Date getCreateTime() {
        return createTime;
    }

    /**
     * 设置创建时间
     *
     * @param createTime 创建时间
     */
    public void setCreateTime(Date createTime) {
        this.createTime = createTime;
    }

    @Override
    public String toString() {
        return "User{" +
```

```
                "id=" + id +
                ", username='" + username + '\'' +
                ", password='" + password + '\'' +
                ", sex=" + sex +
                '}';
    }
}
```

- 新建 mapper
- 新建接口 UserMapper

```
package com.atguigu.docker.mapper;

import com.atguigu.docker.entities.User;
import tk.mybatis.mapper.common.Mapper;

public interface UserMapper extends Mapper<User>
{
}
```

- src\main\resources 路径下新建 mapper 文件夹并新增 UserMapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD
Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-
3-mapper.dtd">
<mapper
```

```
namespace="com.atguigu.docker.mapper.UserMapp
er">
  <resultMap id="BaseResultMap"
type="com.atguigu.docker.entities.User">
    <!--
      WARNING - @mbg.generated
    -->
    <id column="id" jdbcType="INTEGER"
property="id" />
    <result column="username"
jdbcType="VARCHAR" property="username" />
    <result column="password"
jdbcType="VARCHAR" property="password" />
    <result column="sex" jdbcType="TINYINT"
property="sex" />
    <result column="deleted" jdbcType="TINYINT"
property="deleted" />
    <result column="update_time"
jdbcType="TIMESTAMP" property="updateTime" />
    <result column="create_time"
jdbcType="TIMESTAMP" property="createTime" />
  </resultMap>
</mapper>
```

- 新建 service

```
package com.atguigu.docker.service;
```

```
import com.atguigu.docker.entities.User;
import com.atguigu.docker.mapper.UserMapper;
import lombok.extern.slf4j.Slf4j;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import
org.springframework.beans.factory.annotation.Autow
ired;
import
org.springframework.data.redis.core.RedisTemplate;
import
org.springframework.data.redis.core.ValueOperation
s;
import org.springframework.stereotype.Service;
import
org.springframework.web.bind.annotation.PathVariab
le;

import javax.annotation.Resource;
import java.util.concurrent.TimeUnit;

/**
 * @auther zzyy
 * @create 2021-05-01 14:58
 */
@Service
@Slf4j
```

```
public class UserService {

    public static final String CACHE_KEY_USER =
"user:";

    @Resource
    private UserMapper userMapper;
    @Resource
    private RedisTemplate redisTemplate;

    /**
     * addUser
     * @param user
     */
    public void addUser(User user)
    {
        //1 先插入 mysql 并成功
        int i = userMapper.insertSelective(user);

        if(i > 0)
        {
            //2 需要再次查询一下 mysql 将数据捞回来并
ok
            user =
userMapper.selectByPrimaryKey(user.getId());
            //3 将捞出来的 user 存进 redis，完成新增功
能的数据一致性。
```

```
            String key =
CACHE_KEY_USER+user.getId();
            redisTemplate.opsForValue().set(key,user);
        }
    }

    /**
     * findUserById
     * @param id
     * @return
     */
    public User findUserById(Integer id)
    {
        User user = null;
        String key = CACHE_KEY_USER+id;

        //1 先从 redis 里面查询，如果有直接返回结果，
如果没有再去查询 mysql
        user = (User)
redisTemplate.opsForValue().get(key);

        if(user == null)
        {
            //2 redis 里面无，继续查询 mysql
            user = userMapper.selectByPrimaryKey(id);
            if(user == null)
            {
```

```
                //3.1 redis+mysql 都无数据
                //你具体细化，防止多次穿透，我们规定，
记录下导致穿透的这个 key 回写 redis
                return user;
            }else{
                //3.2 mysql 有，需要将数据写回 redis，保
证下一次的缓存命中率
                redisTemplate.opsForValue().set(key,user)
;
            }
        }
        return user;
    }
}
```

- 新建 controller

```
package com.atguigu.docker.controller;

import cn.hutool.core.util.IdUtil;
import cn.hutool.core.util.ReferenceUtil;
import com.atguigu.docker.entities.User;
import com.atguigu.docker.entities.UserDTO;
import com.atguigu.docker.service.UserService;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
```

```
import io.swagger.models.auth.In;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.BeanUtils;
import
org.springframework.beans.factory.annotation.Autow
ired;
import org.springframework.web.bind.annotation.*;

import javax.annotation.Resource;
import java.util.Random;

/**
 * @auther zzyy
 * @create 2021-05-01 15:02
 */
@Api(description = "用户 User 接口")
@RestController
@Slf4j
public class UserController
{
    @Resource
    private UserService userService;

    @ApiOperation("数据库新增 3 条记录")
    @RequestMapping(value = "/user/add",method =
RequestMethod.POST)
    public void addUser()
```

```
    {
        for (int i = 1; i <=3; i++) {
            User user = new User();

            user.setUsername("zzyy"+i);
            user.setPassword(IdUtil.simpleUUID().substri
ng(0,6));
            user.setSex((byte) new Random().nextInt(2));

            userService.addUser(user);
        }
    }

    @ApiOperation("删除 1 条记录")
    @RequestMapping(value =
"/user/delete/{id}",method = RequestMethod.POST)
    public void deleteUser(@PathVariable Integer id)
    {
        userService.deleteUser(id);
    }

    @ApiOperation("修改 1 条记录")
    @RequestMapping(value =
"/user/update",method = RequestMethod.POST)
    public void updateUser(@RequestBody UserDTO
userDTO)
    {
```

```
        User user = new User();
        BeanUtils.copyProperties(userDTO,user);
        userService.updateUser(user);
    }

    @ApiOperation("查询 1 条记录")
    @RequestMapping(value =
"/user/find/{id}",method = RequestMethod.GET)
    public User findUserById(@PathVariable Integer id)
    {
        return userService.findUserById2(id);
    }
}
```

- mvn package 命令将微服务形成新的 jar 包 并上传到 Linux 服务器 /mydocker 目录下
- 编写 Dockerfile

```
# 基础镜像使用 java
FROM java:8
# 作者
MAINTAINER zzyy
# VOLUME 指定临时文件目录为/tmp，在主机/var/lib/docker 目录下创建了一个临时文件并链接到容器的/tmp
VOLUME /tmp
# 将 jar 包添加到容器中并更名为 zzyy_docker.jar
```

```
ADD docker_boot-0.0.1-SNAPSHOT.jar
zzyy_docker.jar
# 运行 jar 包
RUN bash -c 'touch /zzyy_docker.jar'
ENTRYPOINT ["java","-
jar","/zzyy_docker.jar"]
#暴露 6001 端口作为微服务
EXPOSE 6001
```

- 构建镜像
- docker build -t zzyy_docker:1.6 .

- 不用 **Compose**
  - 单独的 mysql 容器实例
  - 新建 mysql 容器实例

```
docker run -p 3306:3306 --name mysql57 --
privileged=true -v
/zzyyuse/mysql/conf:/etc/mysql/conf.d -v
/zzyyuse/mysql/logs:/logs -v
/zzyyuse/mysql/data:/var/lib/mysql -e
MYSQL_ROOT_PASSWORD=123456 -d
mysql:5.7
```

  - 进入 mysql 容器实例并新建库 db2021+新建表 t_user

```
docker exec -it mysql57 /bin/bash
mysql -uroot -p
create database db2021;
use db2021;
CREATE TABLE `t_user` (
```

```
  `id` INT(10) UNSIGNED NOT NULL AUTO_INCREMENT,
  `username` VARCHAR(50) NOT NULL DEFAULT ''
COMMENT '用户名',
  `password` VARCHAR(50) NOT NULL DEFAULT ''
COMMENT '密码',
  `sex` TINYINT(4) NOT NULL DEFAULT '0' COMMENT '
性别 0=女 1=男 ',
  `deleted` TINYINT(4) UNSIGNED NOT NULL DEFAULT
'0' COMMENT '删除标志，默认 0 不删除，1 删除',
  `update_time` TIMESTAMP NOT NULL DEFAULT
CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP COMMENT '更新时间',
  `create_time` TIMESTAMP NOT NULL DEFAULT
CURRENT_TIMESTAMP COMMENT '创建时间',
  PRIMARY KEY (`id`)
) ENGINE=INNODB AUTO_INCREMENT=1 DEFAULT
CHARSET=utf8mb4 COMMENT='用户表';
```

- 单独的 redis 容器实例

```
docker run  -p 6379:6379 --name redis608 --
privileged=true -v
/app/redis/redis.conf:/etc/redis/redis.conf -v
/app/redis/data:/data -d redis:6.0.8 redis-server
/etc/redis/redis.conf
```

- 微服务工程

```
docker run -d -p 6001:6001 zzyy_docker:1.6
```

- 上面三个容器实例依次顺序启动成功

```
[root@zzyy ~]# docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED          STATUS          PORTS
ES
44dfe1f9b356   zzyy_docker:1.6    "java -jar /zzyy_doc…"   41 minutes ago   Up 41 minutes   0.0.0.0:6001->6001/tcp, :::6001->6001/tcp
ny_goldberg
db606755c941   redis:6.0.8        "docker-entrypoint.s…"   44 minutes ago   Up 43 minutes   0.0.0.0:6379->6379/tcp, :::6379->6379/tcp
is608
da637607675d   mysql:5.7          "docker-entrypoint.s…"   48 minutes ago   Up 48 minutes   0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp
ql57
[root@zzyy ~]#
```

- swagger 测试
  - http://localhost:你的微服务端口/swagger-ui.html#/
- 上面成功了，有哪些问题?

- 先后顺序要求固定，先 mysql+redis 才能微服务访问成功
- 多个 run 命令.....
- 容器间的启停或宕机，有可能导致 IP 地址对应的容器实例变化，映射出错， 要么生产 IP 写死(可以但是不推荐)，要么通过服务调用

- 使用 **Compose**
  - 服务编排，一套带走，安排
  - 编写 docker-compose.yml 文件

```
version: "3"

services:
  microService:
    image: zzyy_docker:1.6
    container_name: ms01
    ports:
      - "6001:6001"
    volumes:
      - /app/microService:/data
    networks:
      - atguigu_net
    depends_on:
      - redis
      - mysql

  redis:
    image: redis:6.0.8
    ports:
      - "6379:6379"
    volumes:
      - /app/redis/redis.conf:/etc/redis/redis.conf
      - /app/redis/data:/data
    networks:
      - atguigu_net
    command: redis-server /etc/redis/redis.conf

  mysql:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: '123456'
      MYSQL_ALLOW_EMPTY_PASSWORD: 'no'
      MYSQL_DATABASE: 'db2021'
      MYSQL_USER: 'zzyy'
      MYSQL_PASSWORD: 'zzyy123'
```

```
    ports:
      - "3306:3306"
    volumes:
      - /app/mysql/db:/var/lib/mysql
      - /app/mysql/conf/my.cnf:/etc/my.cnf
      - /app/mysql/init:/docker-entrypoint-initdb.d
    networks:
      - atguigu_net
    command: --default-authentication-
plugin=mysql_native_password #解决外部无法访问

networks:
  atguigu_net:
```

- 第二次修改微服务工程 docker_boot
- 写 YML

```
server.port=6001


# ========================alibaba.druid 相关配
置=====================
spring.datasource.type=com.alibaba.druid.pool.Druid
DataSource
spring.datasource.driver-class-
name=com.mysql.jdbc.Driver
#spring.datasource.url=jdbc:mysql://192.168.111.169:
3306/db2021?useUnicode=true&characterEncoding
=utf-8&useSSL=false
spring.datasource.url=jdbc:mysql://mysql:3306/db2
021?useUnicode=true&characterEncoding=utf-
8&useSSL=false
spring.datasource.username=root
spring.datasource.password=123456
```

```
spring.datasource.druid.test-while-idle=false

# =========================redis 相关配置
=====================
spring.redis.database=0
#spring.redis.host=192.168.111.169
spring.redis.host=redis
spring.redis.port=6379
spring.redis.password=
spring.redis.lettuce.pool.max-active=8
spring.redis.lettuce.pool.max-wait=-1ms
spring.redis.lettuce.pool.max-idle=8
spring.redis.lettuce.pool.min-idle=0

# =========================mybatis 相关配置
====================
mybatis.mapper-locations=classpath:mapper/*.xml
mybatis.type-aliases-
package=com.atguigu.docker.entities

#
=========================swagger==========
===========
spring.swagger2.enabled=true
```

- 通过服务名访问，IP 无关
- mvn package 命令将微服务形成新的 jar 包 并上传到 Linux 服务器 /mydocker 目录下

- 编写 Dockerfile

```
# 基础镜像使用 java
FROM java:8
# 作者
MAINTAINER zzyy
# VOLUME 指定临时文件目录为/tmp，在主机/var/lib/docker 目录下创建了一个临时文件并链接到容器的/tmp
VOLUME /tmp
# 将 jar 包添加到容器中并更名为 zzyy_docker.jar
ADD docker_boot-0.0.1-SNAPSHOT.jar zzyy_docker.jar
# 运行 jar 包
RUN bash -c 'touch /zzyy_docker.jar'
ENTRYPOINT ["java","-jar","/zzyy_docker.jar"]
#暴露 6001 端口作为微服务
EXPOSE 6001
```

- 构建镜像
- docker build -t zzyy_docker:1.6 .
- 执行 docker-compose up 或者 执行 docker-compose up -d

```
[root@zzyy mydocker]# docker-compose up
Starting mydocker_mysql_1 ... done
Starting mydocker_redis_1 ... done
Starting mydocker_webapp_1 ... done
Attaching to mydocker_mysql_1, mydocker_redis_1, mydocker_webapp_1
```

```
[root@zzyy ~]# docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED        STATUS         PORTS
0b97500596a7   zzyy_docker:1.6    "java -jar /zzyy_doc…"   5 days ago     Up 3 minutes   0.0.0.0:6001->6001/tcp, :::60
_webapp_1
dc38664ca38e   mysql:5.7          "docker-entrypoint.s…"   5 days ago     Up 3 minutes   0.0.0.0:3306->3306/tcp, :::33
_mysql_1
7308364913bf   redis:6.0.8        "docker-entrypoint.s…"   5 days ago     Up 3 minutes   0.0.0.0:6379->6379/tcp, :::63
_redis_1
[root@zzyy ~]#
```

- 进入 mysql 容器实例并新建库 db2021+新建表 t_user

docker exec -it 容器实例 id /bin/bash
mysql -uroot -p
create database db2021;
use db2021;
CREATE TABLE `t_user` (`id` INT(10) UNSIGNED NOT NULL AUTO_INCREMENT, `username` VARCHAR(50) NOT NULL DEFAULT '' COMMENT '用户名', `password` VARCHAR(50) NOT NULL DEFAULT '' COMMENT '密码', `sex` TINYINT(4) NOT NULL DEFAULT '0' COMMENT '性别 0=女 1=男 ', `deleted` TINYINT(4) UNSIGNED NOT NULL DEFAULT '0' COMMENT '删除标志，默认 0 不删除，1 删除', `update_time` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT '更新时间', `create_time` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间', PRIMARY KEY (`id`)) ENGINE=INNODB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8mb4 COMMENT='用户表';

- 测试通过
- Compose 常用命令

Compose 常用命令

docker-compose -h # 查看帮助

docker-compose up # 启动所有 docker-compose 服务

docker-compose up -d # 启动所有 docker-compose 服务并后台运行

docker-compose down # 停止并删除容器、网络、卷、镜像。

docker-compose exec yml 里面的服务 id # 进入容器实例内部 docker-compose exec docker-compose.yml 文件中写的服务 id /bin/bash
docker-compose ps # 展示当前 docker-compose 编排过的运行的所有容器
docker-compose top # 展示当前 docker-compose 编排过的容器进程

docker-compose logs yml 里面的服务 id # 查看容器输出日志
dokcer-compose config # 检查配置
dokcer-compose config -q # 检查配置，有问题才有输出
docker-compose restart # 重启服务
docker-compose start # 启动服务
docker-compose stop # 停止服务

- 关停

```
[root@zzyy mydocker]# pwd
/mydocker
[root@zzyy mydocker]#
[root@zzyy mydocker]# ll
总用量 56864
drwxr-xr-x. 1 root    root       44 10月 31 16:03 cig
drwxr-xr-x. 1 root    root       12 10月 28 17:44 conf
drwxr-xr-x. 1 polkitd root      384 11月  3 14:48 db
-rw-r--r--. 1 root    root 58220044 10月 28 18:52 docker_boot-0.0.1-SNAPSHOT.jar
-rw-r--r--. 1 root    root      864 10月 28 18:04 docker-compose.yml
-rw-r--r--. 1 root    root      471 10月 26 20:14 Dockerfile
drwxr-xr-x. 1 root    root        0 10月 28 17:44 init
[root@zzyy mydocker]#
[root@zzyy mydocker]# docker-compose stop
Stopping mydocker_webapp_1 ... done
Stopping mydocker_mysql_1  ... done
Stopping mydocker_redis_1  ... done
[root@zzyy mydocker]#
```

2.6. Docker 轻量级可视化工具 Portainer

2.6.1. 是什么

Portainer 是一款轻量级的应用，它提供了图形化界面，用于方便地管理 Docker 环境，包括单机环境和集群环境。

2.6.2. 安装

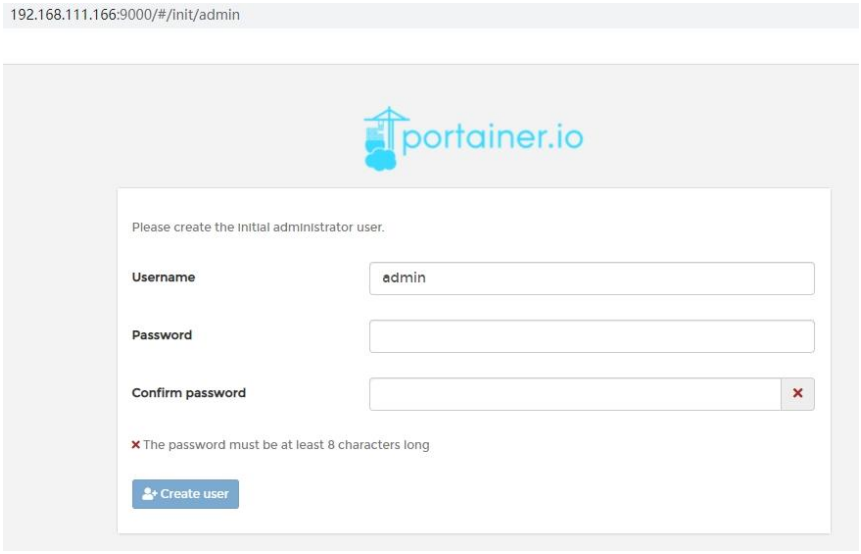
- 官网
  - https://www.portainer.io/
  - https://docs.portainer.io/v/ce-2.9/start/install/server/docker/linux
- 步骤
  - docker 命令安装

```
docker run -d -p 8000:8000 -p 9000:9000 --name portainer --restart=always -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer
```

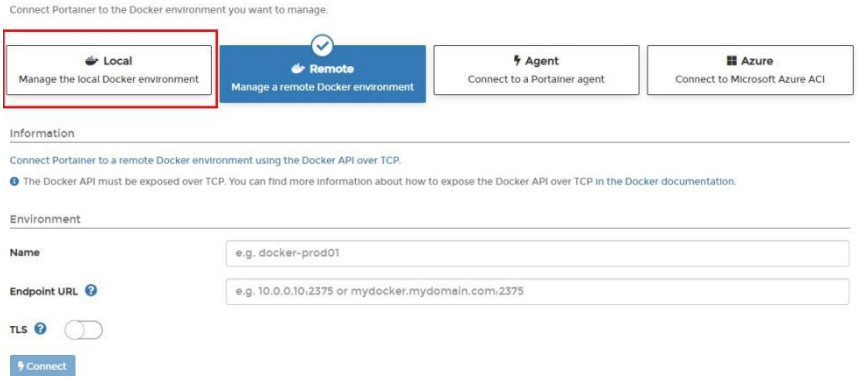
```
[root@zzyy mydocker]# docker run -d -p 8000:8000 -p 9000:9000 --name portainer \
>     --restart=always \
>     -v /var/run/docker.sock:/var/run/docker.sock \
>     -v portainer_data:/data \
>     portainer/portainer-ce:latest
Unable to find image 'portainer/portainer-ce:latest' locally
latest: Pulling from portainer/portainer-ce
7721cab3d696: Pull complete
0645e7e2a110: Pull complete
8f36f6c7596b: Pull complete
Digest: sha256:af387baba14e0342e40d274c0c894fd333d3cca0d6737a8e1e0d6d9523c87a8a
Status: Downloaded newer image for portainer/portainer-ce:latest
ef9093ad230882392964707c0b54b70a10c8032ff1163390716db26feffdd3e0
[root@zzyy mydocker]# docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED
                                                                       NAMES
ef9093ad2308   portainer/portainer-ce:latest  "/portainer"             31 seconds ago
 :::8000->8000/tcp, 0.0.0.0:9000->9000/tcp, :::9000->9000/tcp, 9443/tcp   portainer
3921e00780c7   zzyy_docker:1.6                "java -jar /zzyy_doc…"   About an hour ago
 :::6001->6001/tcp                                                     ms01
22fb3f788c8d   mysql:5.7                      "docker-entrypoint.s…"   About an hour ago
 :::3306->3306/tcp, 33060/tcp                                          mydocker_mysql_1
```

- 第一次登录需创建 admin，访问地址：xxx.xxx.xxx.xxx:9000

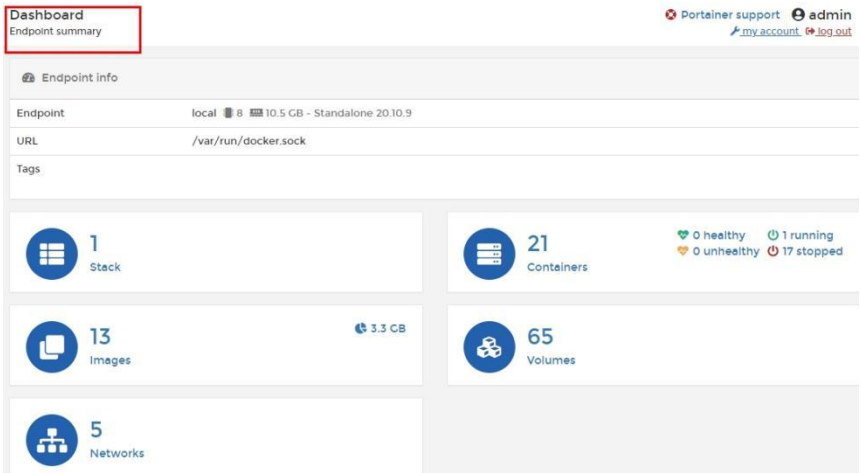
用户名，直接用默认 admin
密码记得 8 位，随便你写



· 设置 admin 用户和密码后首次登陆



· 选择 local 选项卡后本地 docker 详细信息展示



· 上一步的图形展示，能想得起对应命令吗?

```
[root@zzyy tmp]# docker system df
TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images          13        10        2.679GB   747.7MB (27%)
Containers      32        4         31.61kB   24.74kB (78%)
Local Volumes   65        11        1.96GB    1.959GB (99%)
Build Cache     0         0         0B        0B
[root@zzyy tmp]#
```

2.6.3. 登陆并演示介绍常用操作 case

2.7. Docker 容器监控之 CAdvisor+InfluxDB+Granfana

2.7.1. 原生命令

- 操作

```
[root@zzyy mydocker]# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS
              NAMES
7760c8b45236   mysql:5.7      "docker-entrypoint.s…"   17 seconds ago   Up 16 seconds
06/tcp, 33060/tcp   vigilant_shirley
c18cdcb894eb   redis:6.0.8    "docker-entrypoint.s…"   4 minutes ago    Up 4 minutes
79/tcp              confident_mcnulty
[root@zzyy mydocker]#
[root@zzyy mydocker]# docker stats
```

docker stats 命令的结果

```
CONTAINER ID   NAME             CPU %     MEM USAGE / LIMIT     MEM %     NET I/O       BLOCK I/O       PIDS
7760c8b45236   vigilant_shirley 0.13%     269.6MiB / 9.75GiB    2.70%     648B / 0B     60.4MB / 400MB  27
c18cdcb894eb   confident_mcnulty 0.12%    10.32MiB / 9.75GiB    0.10%     648B / 0B     27.5MB / 0B     6
```

- 问题

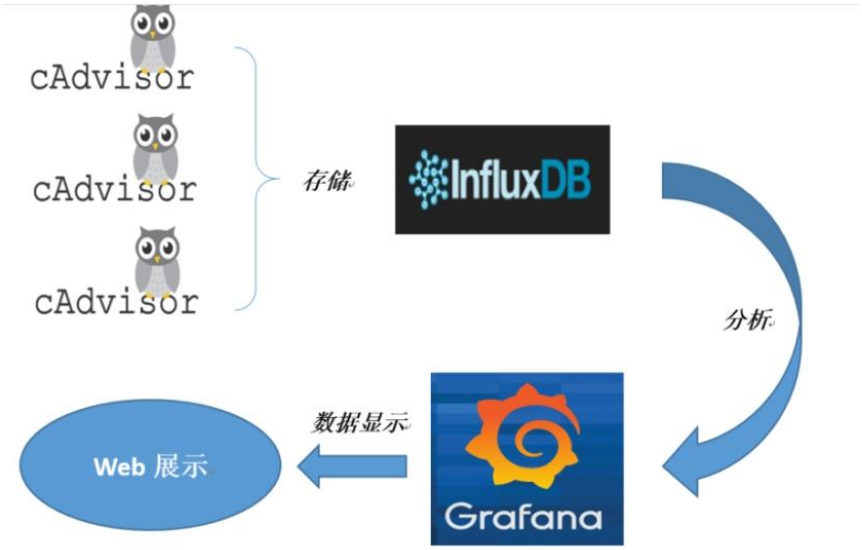
通过 docker stats 命令可以很方便的看到当前宿主机上所有容器的 CPU,内存以及网络流量等数据，一般小公司够用了。。。。

但是，

docker stats 统计结果只能是当前宿主机的全部容器，数据资料是实时的，没有地方存储、没有健康指标过线预警等功能

2.7.2. 是什么

- 容器监控 3 剑客
  - 一句话



  - CAdvisor 监控收集+InfluxDB 存储数据+Granfana 展示图表
  - CAdvisor

CAdvisor

CAdvisor是一个容器资源监控工具,包括容器的内存,CPU,网络IO,磁盘IO等监控,同时提供了一个WEB页面用于查看容器的实时运行状态。CAdvisor默认存储2分钟的数据,而且只是针对单物理机。不过，CAdvisor提供了很多数据集成接口,支持InfluxDB,Redis,Kafka,Elasticsearch等集成,可以加上对应配置将监控数据发往这些数据库存储起来。

CAdvisor功能主要有两点:

- 展示Host和容器两个层次的监控数据。
- 展示历史变化数据。

  - InfluxDB

InfluxDB

InfluxDB是用Go语言编写的一个开源分布式时序、事件和指标数据库,无需外部依赖。

CAdvisor默认只在本机保存最近2分钟的数据，为了持久化存储数据和统一收集展示监控数据，需要将数据存储到InfluxDB中。InfluxDB是一个时序数据库,专门用于存储时序相关数据，很适合存储CAdvisor的数据。而且，CAdvisor本身已经提供了InfluxDB的集成方法，丰启动容器时指定配置即可。

InfluxDB主要功能:

- 基于时间序列,支持与时间有关的相关函数(如最大、最小、求和等);
- 可度量性:你可以实时对大量数据进行计算;
- 基于事件:它支持任意的事件数据;

· Granfana

Granfana

Grafana是一个开源的数据监控分析可视化平台,支持多种数据源配置(支持的数据源包括InfluxDB,MySQL,Elasticsearch,OpenTSDB,Graphite等)和丰富的插件及模板功能,支持图表权限控制和报警。

Grafan主要特性:

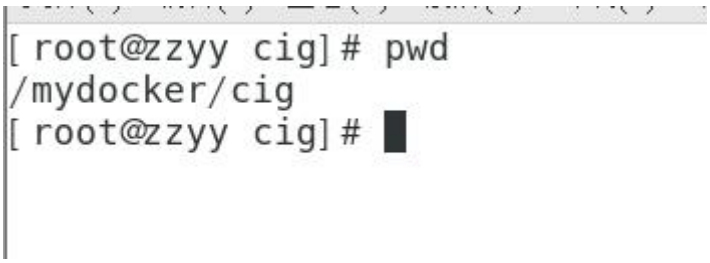
- 灵活丰富的图形化选项
- 可以混合多种风格
- 支持白天和夜间模式
- 多个数据源

· 总结

- cAdvisor – Collects, aggregates, processes, and exports information about running containers
- InfluxDB – Time Series Database stores all the metrics
- Grafana – Metrics Dashboard

2.7.3. compose 容器编排，一套带走

· 新建目录



- 新建 3 件套组合的 docker-compose.yml

```
version: '3.1'

volumes:
  grafana_data: {}

services:
 influxdb:
  image: tutum/influxdb:0.9
  restart: always
  environment:
    - PRE_CREATE_DB=cadvisor
  ports:
    - "8083:8083"
    - "8086:8086"
  volumes:
    - ./data/influxdb:/data

 cadvisor:
  image: google/cadvisor
  links:
    - influxdb:influxsrv
  command: -storage_driver=influxdb -
storage_driver_db=cadvisor -
storage_driver_host=influxsrv:8086
  restart: always
  ports:
    - "8080:8080"
  volumes:
    - /:/rootfs:ro
    - /var/run:/var/run:rw
```

```
    - /sys:/sys:ro
    - /var/lib/docker/:/var/lib/docker:ro

  grafana:
   user: "104"
   image: grafana/grafana
   user: "104"
   restart: always
   links:
    - influxdb:influxsrv
   ports:
    - "3000:3000"
   volumes:
    - grafana_data:/var/lib/grafana
   environment:
    - HTTP_USER=admin
    - HTTP_PASS=admin
    - INFLUXDB_HOST=influxsrv
    - INFLUXDB_PORT=8086
    - INFLUXDB_NAME=cadvisor
    - INFLUXDB_USER=root
    - INFLUXDB_PASS=root
```

- 启动 docker-compose 文件
  - docker-compose up

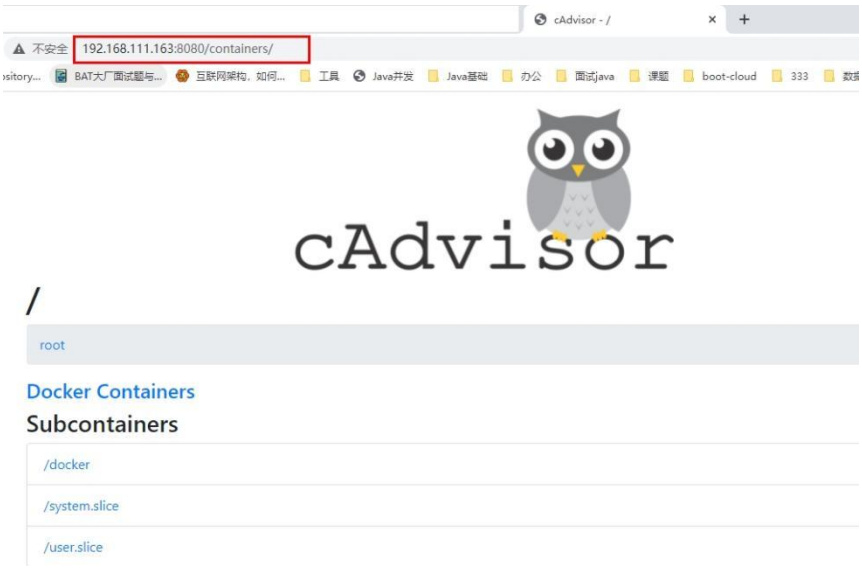
```
[root@zyy cig]# docker-compose up
Creating network "cig_default" with the default driver
Pulling influxdb (tutum/influxdb:0.9)...
0.9: Pulling from tutum/influxdb
Image docker.io/tutum/influxdb:0.9 uses outdated schema1 manifest format. Please upgrade to a schema2
ility. More information at https://docs.docker.com/registry/spec/deprecated-schema-v1/
a3ed95caeb02: Pull complete
23efb549476f: Pull complete
aa2f8df21433: Pull complete
ef072d3c9b41: Pull complete
c9f371853f28: Pull complete
a248b0871c3c: Pull complete
749db6d368d0: Pull complete
09a7be9c234a: Pull complete
ce08d78abe37: Pull complete
45c4e95cdf8a: Pull complete
cc4fd199ba04: Pull complete
Digest: sha256:2852933ccb4bde186031f819e15bcf8ba58e83c0fdf31e5f6bee5bb23a1c1f63
Status: Downloaded newer image for tutum/influxdb:0.9
Pulling cadvisor (google/cadvisor:)...
latest: Pulling from google/cadvisor
ff3a5c916c92: Pull complete
```

```
Status: Downloaded newer image for google/cadvisor:latest
Pulling grafana (grafana/grafana:)...
latest: Pulling from grafana/grafana
a0d0a0d46f8b: Pull complete
ac27e49d9e31: Pull complete
a40f2a35c016: Pull complete
0ae21741e86c: Pull complete
b55eac8c5e31: Pull complete
f73d9d8d4dd9: Pull complete
4f4fb700ef54: Pull complete
ccd7ded5cb2a: Pull complete
a88d683c5ad2: Pull complete
Digest: sha256:00568d89c4f8a2cfa0d56f0fcd875b23ec8000b743a62f442e1ee91fce9a6e24
Status: Downloaded newer image for grafana/grafana:latest
Creating cig_influxdb_1 ... done
Creating cig_cadvisor_1 ... done
Creating cig_grafana_1  ... done
```

- 查看三个服务容器是否启动

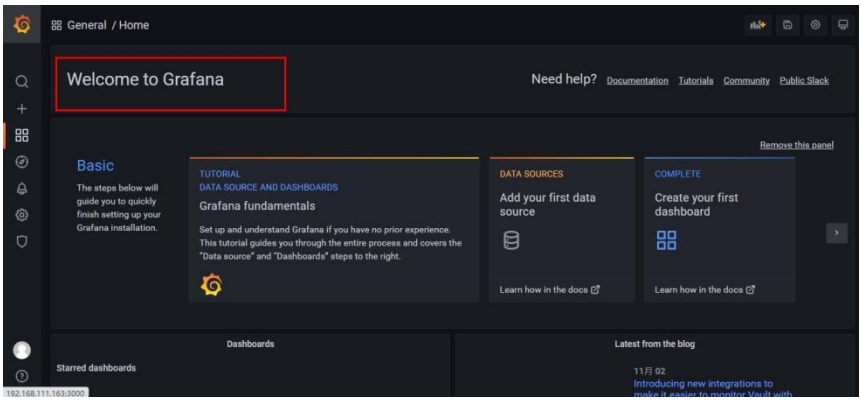
```
[root@zzyy cig]# docker ps
CONTAINER ID   IMAGE                 COMMAND                  CREATED
de9199c5cc64   grafana/grafana       "/run.sh"                11 minutes ago
000->3000/tcp
fc168274cb21   google/cadvisor       "/usr/bin/cadvisor -…"   11 minutes ago
080->8080/tcp
8a9ad1d15822   tutum/influxdb:0.9    "/run.sh"                11 minutes ago
083->8083/tcp, 8090/tcp, 0.0.0.0:8086->8086/tcp, :::8086->8086/tcp, 8099/tcp
[root@zzyy cig]#
```

- 测试
  - 浏览 cAdvisor 收集服务，http://ip:8080/



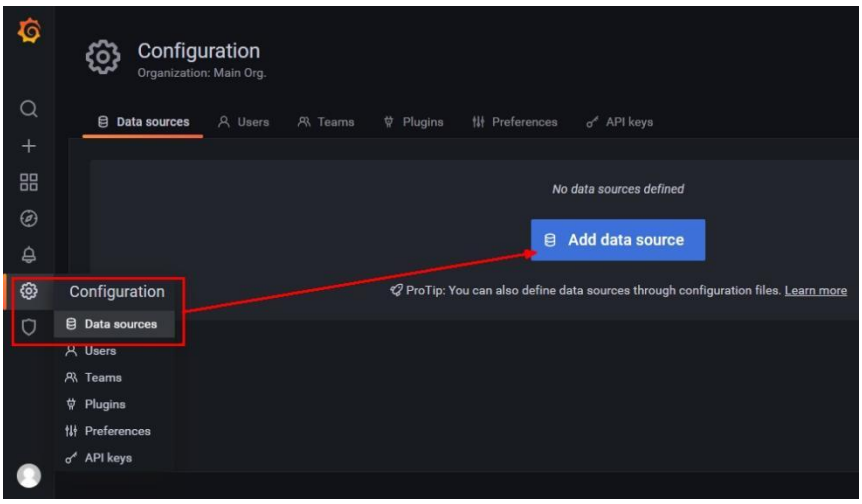
  - 第一次访问慢，请稍等
  - cadvisor 也有基础的图形展现功能，这里主要用它来作数据采集
  - 浏览 influxdb 存储服务，http://ip:8083/
  - 浏览 grafana 展现服务，http://ip:3000

- ip+3000 端口的方式访问,默认帐户密码（admin/admin）

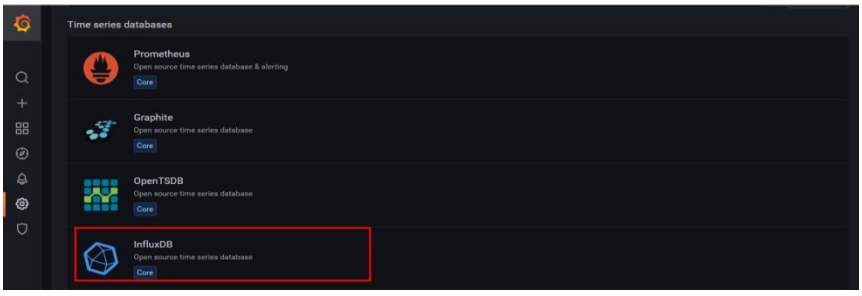


- 配置步骤

- 配置数据源

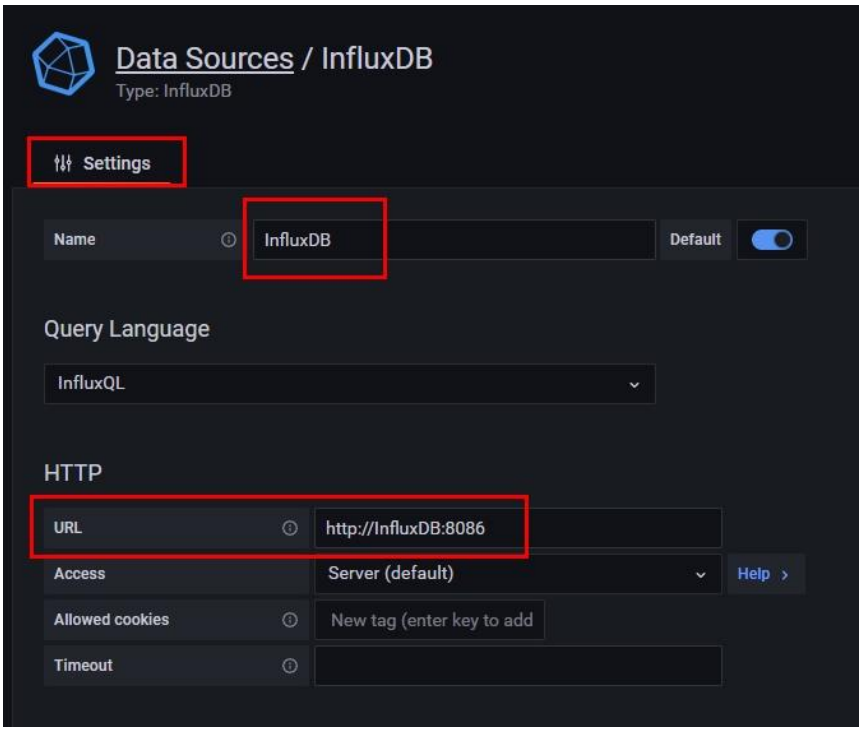


- 选择 infuxdb 数据源

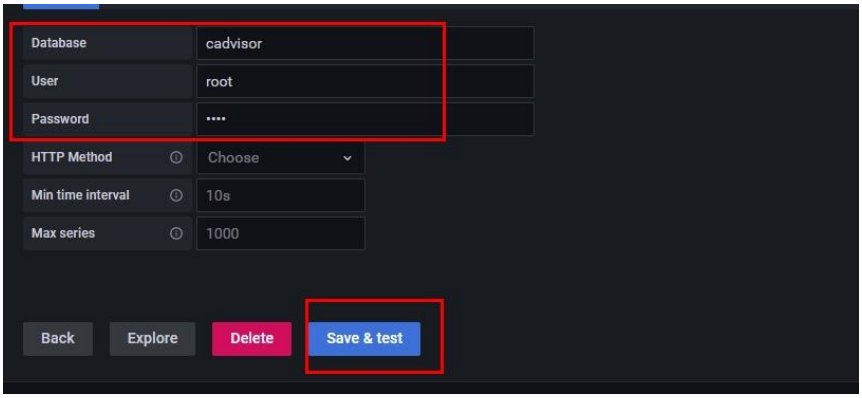


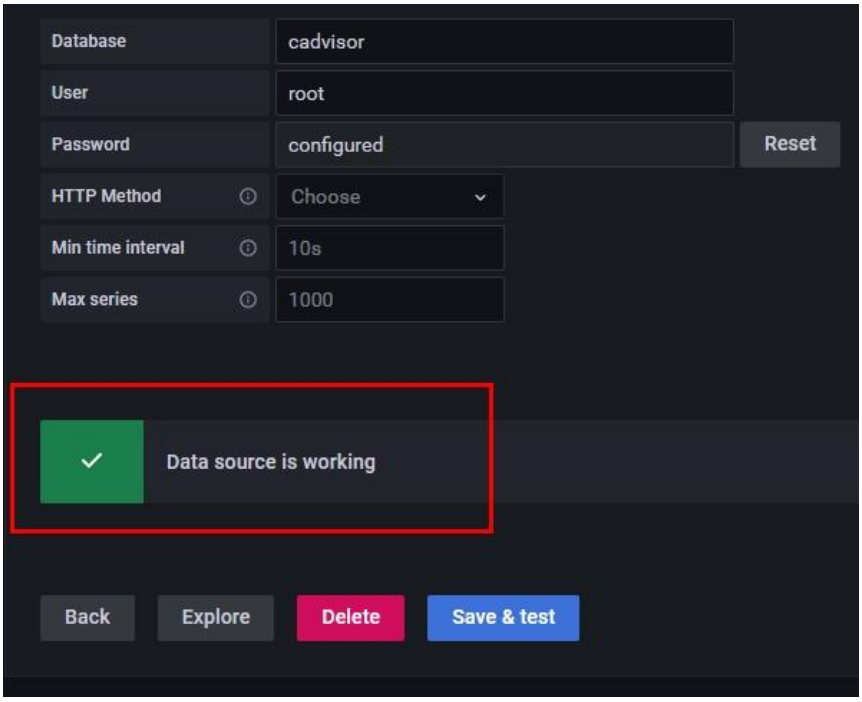
- 配置细节

- 1

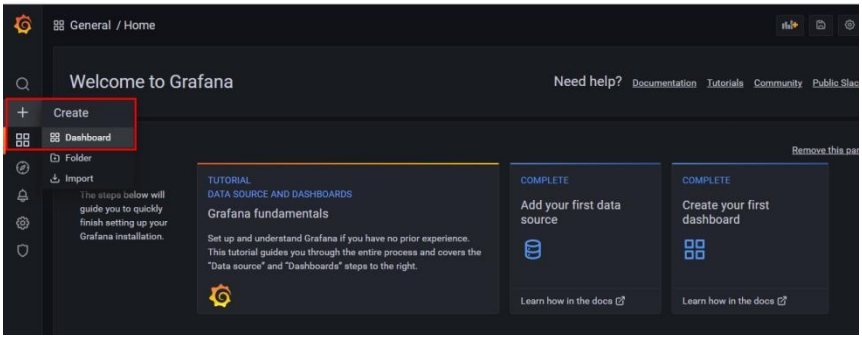

Data Sources / InfluxDB
Type: InfluxDB
Settings
Name
InfluxDB
Default
Query Language
InfluxQL
HTTP
URL
http://InfluxDB:8086
Access
Server (default)
Help
Allowed cookies
New tag (enter key to add
Timeout

· 2

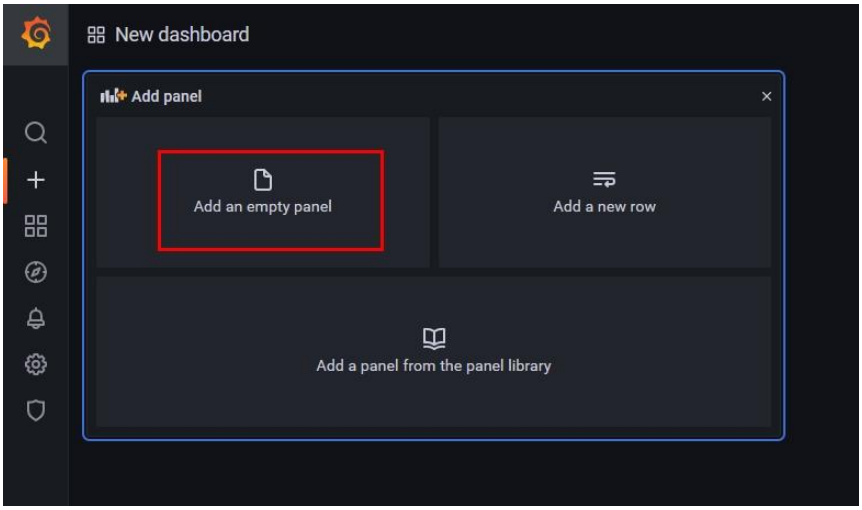

Database
cadvisor
User
root
Password
HTTP Method
Choose
Min time interval
10s
Max series
1000
Back
Explore
Delete
Save & test



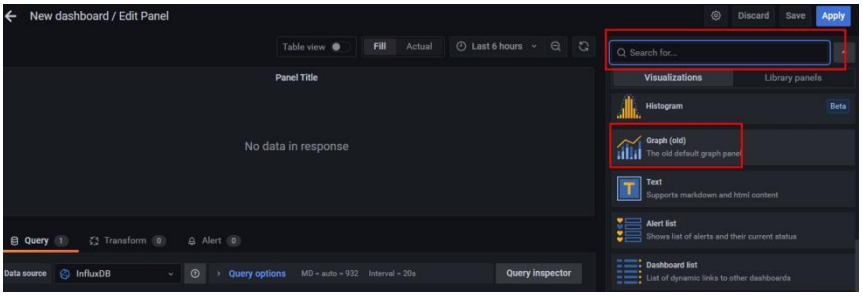
- 配置面板 panel
- 1



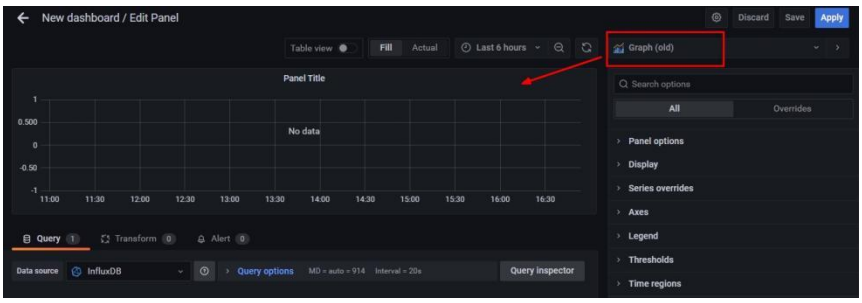
- 2


New dashboard
Add panel
Add an empty panel
Add a new row
Add a panel from the panel library

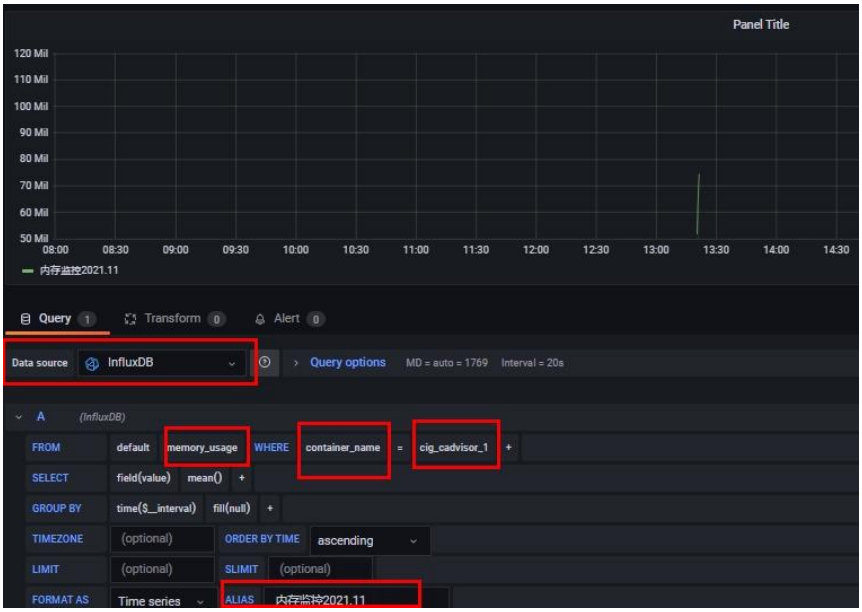
· 3


New dashboard / Edit Panel
Discard
Save
Apply
Table view
Fill
Actual
Last 6 hours
Search for...
Visualizations
Library panels
Panel Title
No data in response
Histogram
Beta
Graph (old)
The old default graph panel
Text
Supports markdown and html content
Alert list
Shows list of alerts and their current status
Dashboard list
List of dynamic links to other dashboards
Query
Transform
Alert
Data source
InfluxDB
Query options
Query inspector

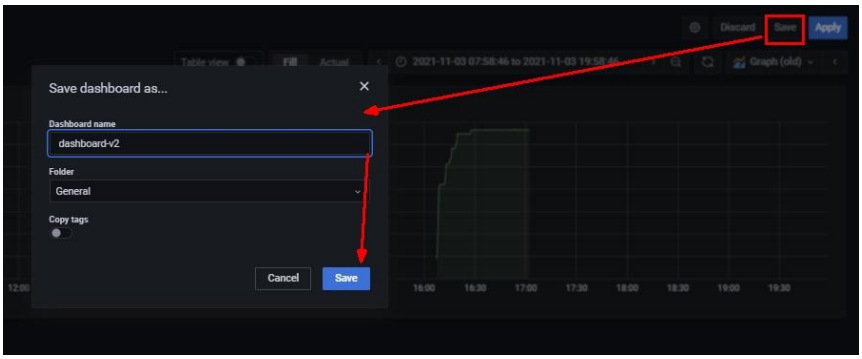
· 4


New dashboard / Edit Panel
Discard
Save
Apply
Table view
Fill
Actual
Last 6 hours
Graph (old)
Panel Title
No data
Search options
All
Overrides
Panel options
Display
Series overrides
Axes
Legend
Thresholds
Time regions
Query
Transform
Alert
Data source
InfluxDB
Query options
Query inspector

· 5



· 6



· 到这里 cAdvisor+InfluxDB+Grafana 容器监控系统就部署完成了

2.8. 终章の总结

2.8.1. 知识回顾简单串讲和总结

```
[root@zzyy mydocker]# docker images
REPOSITORY                                                    TAG       IMAGE ID       CREATED          SIZE
zzyy_docker                                                   1.6       3628162e85ae   21 hours ago     760MB
centosjava8                                                   1.5       a3857c8e1a16   2 days ago       756MB
192.168.111.167:5000/zzyyubuntu                               1.2       62e7fed04e2c   5 days ago       105MB
registry.cn-qingdao.aliyuncs.com/atguigubj/myubuntu           1.3       84a940bd3cb1   5 days ago       173MB
alpine                                                        latest    c059bfaa849c   6 days ago       5.58MB
tomcat                                                        latest    904a98253fbf   12 days ago      680MB
grafana/grafana                                               latest    ddfae340d068   12 days ago      253MB
nginx                                                         latest    ea335eea17ab   13 days ago      141MB
mysql                                                         5.7       8b43c6af2ad0   13 days ago      448MB
registry                                                      latest    b8604a3fe854   2 weeks ago      26.2MB
ubuntu                                                        latest    ba6acccedd29   6 weeks ago      72.8MB
hello-world                                                   latest    feb5d9fea6a5   2 months ago     13.3kB
centos                                                        latest    5d0da3dc9764   2 months ago     231MB
portainer/portainer                                           latest    580c0e4e98b0   8 months ago     79.1MB
redis                                                         6.0.8     16ecd2772934   13 months ago    104MB
google/cadvisor                                               latest    eb1210707573   3 years ago      69.6MB
billygoo/tomcat8-jdk8                                         latest    30ef4019761d   3 years ago      523MB
java                                                          8         d23bdf5b1b1b   4 years ago      643MB
tutum/influxdb                                                0.9       7aa2a38f2ef6   5 years ago      275MB
[root@zzyy mydocker]#
```

2.8.2. 进阶篇：雷丰阳老师的 K8S

加油