

Docker基础篇之快速上手

第一章 Docker简介

是什么？

问题：为什么会有 docker 的出现

一款产品从开发到上线，从操作系统，到运行环境，再到应用配置。作为开发+运维之间的协作我们需要关心很多东西，这也是很多互联网公司都不得不面对的问题，特别是各种版本的迭代之后，不同版本环境的兼容，对运维人员都是考验

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。

环境配置如此麻烦，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-模-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吧，**Java/TomcatMySQL/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）。而为了让这程序可以顺利执行，开发团队也得准备完整的部署文件，让运维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镜像的设计，使得**Docker得以打过去「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“**一次封装，到处运行**”。



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。将应用运行在Docker容器上面，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

能干嘛

之前的虚拟机技术

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作系统，比如在**Windows系统里面运行Linux系统**。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。



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这些缺点，**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。.



比较了**Docker**和传统虚拟化方式的不同之处:

- 1、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;
- 2、而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，**而且也没有进行硬件虚拟**。因此容器要比传统虚拟机为轻便。
- 3、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。

开发/运维(DevOps)

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 inconsistency 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。

企业级

新浪



美团



蘑菇街



去哪下

1、官网

docker官网: <https://www.docker.com/>

docker中文网站: <https://www.docker-cn.com/>

2、仓库

Docker Hub官网: <https://hub.docker.com/>

第二章 Docker安装

前提说明

CentOS Docker安装

Docker支持以下的CentOS版本:

CentOS 7 (64-bit)

CentOS 6.5 (64-bit)或更高的版本

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.上，要求系统为64位、系统内核版本为3.10以上。

Docker运行在CentOS-6.5或更高的版本的CentOS上，要求系统为64位、系统内核版本为2.6.32-431或者更高版本。



Docker 的基本组成

docker架构图



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



容器(container)

Docker利用容器(Container) 独立运行的一个或一组应用。**容器是用镜像创建的运行实例。**

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。

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。

容器的定义和镜像几乎一模一样，也是一堆层的统一视角，唯一区别在于容器的最上面那一层是可读可写的。

仓库(repository)

仓库(Repository) 是集中存放镜像文件的场所。

仓库(Repository)和仓库注册服务器(Registry) 是有区别的。仓库注册服务器上往往存放着多个仓库，每个仓库中又包含了多镜像，每个镜像有不同的标签(tag)。

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 ()

需要正确的理解仓储/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，这个打好的运行环境就似乎image镜像文件。只有通过这个镜像文件才能生成Docker容器。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。

image文件生成的容器实例，本身也是一个文件，称为镜像文件。

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，也就是我们的容器至于仓储，就是放了一堆镜像的地方，我们可以把镜像发布到仓储中，需要的时候从仓储中拉下来就可以了。|

安装步骤

Centos6.8安装Docker

1、yum install -y epel-release



2、yum install -y docker-io



3、安装后的配置文件：etc/sysconfig/docker



4、启动 Docker后台服务: service docker start

5、docker version 验证

Centos7.0安装Docker

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

永远的helloworld

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

登录阿里云开发者平台

获取加速器地址

配置本机Docker运行镜像加速器

鉴于国内网络问题，后续拉取Docker镜像十分缓慢，我们可以需要配置加速器来解决，我使用的是阿里云的本人自己账号的镜像地址(需要自己注册有一个属于你自己的): ht:po/. mirror aliyuncns .com

- vim /etc/sysconfig/docker
将获得的自己账户下的阿里云加速地址配置进
other_args="--registry-mirror=https://你自己的账号加速信息.mirror.aliyuncs.com



重新启动 Docker 后台服务：service docker restart

Linux系统下配置完加速器需要检查是否生效

网易云加速

基本上同上述阿里云

启动Docker后台容器(测试运行 hello-world)

```
docker run hello world
```



run干了什么



底层原理

Docker是怎样工作的

Docker是一个Client-Server结构的系统, Docker守护进程运行在主机上, 然后通过Socket连接从客户端访问, 守护进程从客户端接受命令并管理运行在主机上的容器。**容器, 是一个运行时环境, 就是我们前面说到的集装箱。**



为什么Docker比较比vm快

1、**docker**有着比虚拟机更少的抽象层。由于docker不需要**Hypervisor**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**Guest OS**。因此,当新建一个容器时,docker不需要和虚拟机一样重新加载--个操作系统内核从而避免引导、加载操作系统内核这个比较费时费资源的过程,当新建--个虚拟机时,虚拟机软件需要加载GuestOS,这个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,则省略了这个过程,因此新建一个docker容器只需要几秒钟。



第三章 Docker常用命令

帮助命令

```
docker version
```

```
docker info
```

```
docker --help
```

自己查看官网解释, 高手都是自己练出来的, 百度上只不过是翻译了下, 加了点例子

镜像命令

docker images 列出本机上的镜像

OPTIONS 说明:

- a 列出本地所有的镜像(含中间映射层)
- q 只显示镜像ID
- digests 显示镜像的摘要信息
- no-trunc 显示完整的镜像信息

docker search 某个XXX镜像的名字

网站 www.dockerhub.com

docker search [OPTIONS] 镜像名字

OPTIONS 说明

- no-trunc 显示完整的镜像描述
- s 列出收藏数不小于指定值的镜像
- automated 只列出 automated build类型的镜像

docker pull 某个镜像的名字

下载镜像

docker pull 镜像名字[:TAG]

docker rmi 某个XXX镜像的名字ID

删除镜像

删除单个 docker rm -f 镜像ID

删除多个 docker rm -f 镜像名1:TAG 镜像名2:TAG

删除多个 docker rmi -f \${docker images -qa}

容器命令

有镜像才能创建容器，这是根本前提(下载一个Centos镜像演示)

docker pull centos

新建并启动容器

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;
- 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;
- P**:随机端口映射;
- p**:指定端口映射, 有以下四种格式
ip:hostPort:containerPort
ip::containerPort
hostPort:containerPort
containerPort



列出当前所有正在运行的容器

`dockers ps [OPTIONS]`

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。
- no-trunc** :不截断输出。

退出容器

两种退出方式

`exit` 容器停止退出

`ctrl+P+Q` 容器不停止退出

启动容器

`docker start` 容器ID或容器签名

重启容器

`docker restart` 容器ID或容器签名

停止容器

`docker stop` 容器ID或容器签名

强制停止容器

docker kill 容器ID或容器签名

删除已停止的容器

docker rm 容器ID -f

一次性删除多个容器

docker rm -f \$(docker ps -a -q)

docker ps -a -q | xargs docker rm

重要

启动守护式容器

#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, 最佳的解决方案是將你要运行的程序以前台进程的形式运行

查看容器日志

docker logs -f -t --tail 容器ID

-t 是加入时间戳

-f 跟随最新的日志打印

--tail 数字显示最后多少条

查看容器内的进程

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

docker exec -it 容器ID bashShell



重新进入docker attach 容器ID

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

exec 实在容器中打开新的终端，并且可以穷的那个新的进程

从容器内拷贝文件到主机上

docker cp 容器ID:容器内路径 目的主机路径



小总结



第四章 Docker 镜像

是什么

镜像是一种轻量级、可执行的独立软件包，用来打包软件运行环境和基于运行环境开发的软件，它包含运行某个软件所需的有内容，包括代码、运行时、库、环境变量和配置文件。

UnionFS(联合文件系统)

UnionFS (状节又件示统)

UnionFS (联合文件系统) : Union文件系统(UnionFS)是——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**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。



平时我们安装的虚拟机的Centos都是好几个G，为什么docker这里才要200m



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**。

分层的镜像



分层的镜像



为什么 Docker镜像想要采用这种分层结构

最大的一个好处就是-共享资源

比如:有多个镜像都从相同的**base**镜像构建而来，那么宿主机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。

特点

Docker镜像都是只读的，当容器启动时，一个新的可写层被加载到**镜像的顶部**，这一层通常被称为**容器层**，容器层之下都叫**镜像层**

Docker镜像Commit操作

docker commit 提交容器副本使之称为一个新的镜像

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：

1、从Hub上下载tomcat镜像到本地并成功运行



```
docker run -d -p 8080:8080 tomcat
```

-p 主机端口: docker容器端口
-P: 随机分配端口
i: 交互
t: 终端

2、故意删除上一步镜像生产tomcat容器的文档



3、也即当前的tomcat运行实例是一个没有文档内容的容器，以他为模板commit一个没有doc的tomcat新镜像 atguigu/tomcat02



4、启动我们的新镜像并和原来的对比

启动atguigu/tomcat02 没有doc

启动原来tomcat他有doc

第五章 Docker容器数据卷

是什么

先来看看**Docker**的理念:

将应用与运行的环境打包形成容器运行，运行可以伴随着容器，但是我们对数据的要求希望是持久化的容器之间希望有可能共享数据

Docker容器产生的数据，如果不通过**docker commit**生成新的镜像，使得数据做为镜像的一部分保存下来，

那么当容器删除后，数据自然也就没有了。

为了能保存数据在docker中我们使用卷。

一句话：有点类似我们Redis里面的rdb和aof文件

能干嘛

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 FileSystem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

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生效
- 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

容器的持久化

容器间继承+共享数据

数据卷

容器内添加

直接命令添加



`docker run -it -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



容器和宿主机之间数据共享



容器停止退出后，主机修改后的数据是否同步



命令(带权限)

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



DockerFile添加

根目录下新建mydocker文件夹并进入

可在Dockerfile中使用VOLUME指令来给镜像添加一个或多个数据卷



File构建



build后生成镜像



获得一个新镜像zzyy/centos

run容器



通过上述步骤，容器内的卷目录地址已经知道，对应的主机目录在哪



主机对应默认地址



备注

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

数据卷容器

是什么

命名的容器挂载数据卷，其它容器通过挂载这个(父容器)实现数据共享，挂载数据卷的容器，称之为数据卷容器。

总体介绍

以上一步新建的zzyy/centos为模板并运行容器 doc1/doc2/doc3

他们已经具有容器卷

/dataVolumeContainer1

/dataVolumeContainer2

容器间传递共享(--volumes -from)

先启动一个父容器doc1



启动后在 dataVolumeContainer1中新增内容

doc2/doc3 继承doc1

--volumes -from



doc2/doc3 分别在dataVolumeContainer2各自新增内容

回到doc1可以看到02/03各自添加的都能共享了



删除doc1 doc2修改后doc3是否可以访问



删除doc02后doc3是否访问



在进一步



新建doc04继承doc03 然后删除doc03



结论：容器之间配置信息的传递，数据卷的生命周期一直持续到没有容器使用它为止

第六章 DockerFile解析

是什么

Dockerfile是用来构建Docker镜像的构建文件，由一系列命令和参数构成的脚本

构建三步骤

编写Dockerfile文件

docker build

docker run

文件什么样？？？

熟悉的Centos为例

http://hub.docker.com/_/centos



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的运行态。

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。

DockerFile体系结构(保留字指令)



小总结



案例

Base 镜像(scratch)

Docker Hub中 99%的镜像都是通过在base镜像中安装和配置需要的软件构建出来的



自定义镜像mycentos

1、编写

Hub默认Centos镜像是什么情况



准备Dockerfile文件

myCentOS内容Dockerfile



```
FROM centos
MAINTAINER ZZYY<zzyy167@126.com>

ENV MYPATH /usr/local
WORKDIR $MYPATH

RUN yum -y install vim
RUN yum -y install net-tools

EXPOSE 80

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

2、构建

docker build -t 新镜像名字:TAG .



3、运行

docker run -it 新镜像名字:TAG



4、列出镜像的变更历史

docker history 镜像名

CMD/ENTRYPOINT 镜像案例

都是指定一个容器启动时要运行的命令

CMD

Dockerfile中可以有多个CMD指令，但只有最后一个生效，CMD会被dockerrun之后的参数替换

Case

tomcat的讲解演示 docker run -it -p 8080:8080 tomcat ls -l

ENTRYPOINT

docker run 之后的参数会被当做参数传递给 ENTRYPOINT 之后形成新的命令组合

Case



制作CMD版可以查询IP信息的容器



curl的命令解释

curl命令可以用来执行下载、发送各种HTTP请求，指定HTTP头部等操作。

如果系统没有curl可以使用yum install curl安装，也可以下载安装。

curl是将下载文件输出到stdout

使用命令: curl <http://www.baidu.com>

执行后，www.baidu.com的html就会显示在屏幕上了

这是最简单的使用方法。用这个命令获得了<http://curl.haxx.se>指向的页面，同样，如果这里的URL指向的是一个文件或者一幅图都可以直接下载到本地。如果下载的是HTML文档，那么缺省的将只显示文件头部，即HTML文档的header。要全部显示，请加参数-i

WHY

我们可以看到可执行文件找不到的报错，**executable file not found**。

之前我们说过，**跟在镜像名后面的是command,运行时会替换CMD的默认值。**

因此这里的-i替换了原来的CMD，而不是添加在原来的curl -s <http://ip.cn>后面。而-i 根本不是命令，所以自然找不到。

那么如果我们希望加入-i这参数，我们就必须重新完整的输入这个命令：

\$ docker run myip curl -s <http://ip.cn> -i



自定义镜像Tomcat

1、mkdir -p /zzyy/mydockerfile/tomcat9

2、在上述目录下 touch c.txt

3、将jdk和tomcat安装的压缩包拷贝进上一步目录

4、在zzyyuse/mydockerfile/tomcat9目录下新建Dockerfile文件

```
FROM centos
MAINTAINER zzyy<zzyybs@ 126.com>
#把宿主机当前上下文的c .txt拷贝到容器/usr/local/路径下
COPY c.txt /usr/local/cincontainer.txt
#把java与tomcat添加到容器中
ADD jdk-8u171-linux x64.tar.gz sr/local/
ADD apache-tomcat-9.0.8.tar.gz /usr/local/
#安装vim编辑器
RUN yum -y install vim
#设置工作访问时候的WORKDIR路径,登录落脚点
ENV MYPATH /usr/local
WORKDIR $MYPATH
#配置java与tomcat环境变量
ENV JAVA_HOME /usr/local/jdk1.8.0_171
ENV CLASSPATH $JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
ENV CATALINA_HOME /usr/local/apache-tomcat-9.0.8
ENV CATALINA_BASE /usr/local/apache-tomcat-9.0.8
ENV PATH $PATH:$JAVA_HOME/bin:$CATALINA_HOME/lib:$CATALINA_HOME/bin
#容器运行时监听的端口
EXPOSE 8080
#启动时运行tomcat
# ENTRYPOINT ["/usr/local/apache-tomcat-9.0.8/bin/startup.sh" ]
# CMD ["/usr/local/apache-tomcat-9.0.8/bin/catalina.sh","run"]
CMD /usr/local/apache-tomcat-9.0.8/bin/startup.sh && tail -F /usr/local/apache-tomcat-9.0.8/logs/catalina.out
```

目录内容



5、构建



构建完成



6、run

```
docker run -d -p 9080:8080 -name myt9
-v /zyuse/mydockerfile/tomcat9/test:/usr/local/apache-tomcat9.0.8/webapps/test
-v /zzyyuse/mydockerfile/tomcat9/tomcat9/logs:/usr/local/apache-tomcat-9.0.8/logs zzyytomcat9
```



备注

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

7、验证



8、综合前述容器卷测试的web服务test发布



web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
id="webApp_ID" version="2.5">
<display-name>test</display-name>

</web-app>
```

a.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here </title>
</head>
<body>
welcome-
<%= "i am in docker tomcat self" %>
<br>
<br>
<% System.out.println("=====docker tomcat self");%>
</body>
</html>
```

测试

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here </title>
</head>
<body>
welcome-
<%= "i am in docker tomcat self" %>
<br>
<br>
<% System.out.println("=====docker tomcat self");%>
```

```
</body>
</html>
```

小总结



第七章 Docker常用安装

总体步骤

搜索镜像

拉取镜像

查看镜像

启动镜像

停止容器

移除容器

安装Mysql

docker hub 上查找mysql镜像



从 docker hub(阿里云加速器)拉取mysql镜像到本地标签为5.6



使用mysql5.6镜像创建容器(也叫运行镜像)

使用mysql镜像

```
docker run -p 12345:3306 --name mysql
-v /ggcc/mysql/conf:/etc/mysql/conf.d
-v /ggcc/mysql/logs:/logs
-v /ggcc/mysql/data:/var/lib/mysql
-e MYSQL_ROOT_PASSWORD=123456 -d mysql:5.6
```

命令说明:

-p 12345:3306:将主机的12345端口映射到docker容器的3306端口。
-name mysql:运行服务名字
-v /ggcc/mysql/conf:/etc/mysql/conf.d :将主机/zzyyuse/mysql录下的conf/my.cnf挂载到容器的/etc/mysql/conf.d
-v /ggcc/mysqllogs/logs: 将主机/zzyyuse/mysql目 录下的logs 目录挂载到容器的/logs。
-v /ggcc/mysqldata:/var/lib/mysql :将主机lzzyyuse/mysql目录下的data目录挂载到容器的/var/lib/mysql .
-e MYSQL_ROOT_PASSWORD=123456: 初始化root用户的密码。.
-d mysql:5.6:后台程序运行mysql5.6

```
-----  
docker exec -it Mysql运行成功后的容器ID /bin/bash  
-----
```

数据备份小测试

```
docker exec mysql服务容器ID sh -c 'exec mysqldump --all-databases -uroot -  
p"123456"' >/ggcc/all-database.sql
```



数据备份测试



安装Redis

从docker hu上(阿里云加速器)拉取redis镜像到本地标签为: 3.2



使用redis3.2镜像创建容器(也叫运行镜像)

使用镜像

```
docker run -p 6379:6379 -v /ggcc/myredis/data:/data -v  
/ggcc/myredis/conf/redis.conf:/usr/local/etc/redis/redis.conf -d redis redis-  
server /usr/local/etc/redis/redis.conf --appendonly yes
```

在主机/ggcc/myredis/conf/redis.conf目录上新建redis.conf文件

vim /ggcc/myredis/conf/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 yes

# 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

##### TLS/SSL #####

# By default, TLS/SSL is disabled. To enable it, the "tls-port" configuration
# directive can be used to define TLS-listening ports. To enable TLS on the
# default port, use:
#
# port 0
# tls-port 6379

```

```
# Configure a X.509 certificate and private key to use for authenticating the
# server to connected clients, masters or cluster peers. These files should be
# PEM formatted.
#
# tls-cert-file redis.crt
# tls-key-file redis.key

# Configure a DH parameters file to enable Diffie-Hellman (DH) key exchange:
#
# tls-dh-params-file redis.dh

# Configure a CA certificate(s) bundle or directory to authenticate TLS/SSL
# clients and peers. Redis requires an explicit configuration of at least one
# of these, and will not implicitly use the system wide configuration.
#
# tls-ca-cert-file ca.crt
# tls-ca-cert-dir /etc/ssl/certs

# By default, clients (including replica servers) on a TLS port are required
# to authenticate using valid client side certificates.
#
# If "no" is specified, client certificates are not required and not accepted.
# If "optional" is specified, client certificates are accepted and must be
# valid if provided, but are not required.
#
# tls-auth-clients no
# tls-auth-clients optional

# By default, a Redis replica does not attempt to establish a TLS connection
# with its master.
#
# Use the following directive to enable TLS on replication links.
#
# tls-replication yes

# By default, the Redis Cluster bus uses a plain TCP connection. To enable
# TLS for the bus protocol, use the following directive:
#
# tls-cluster yes

# Explicitly specify TLS versions to support. Allowed values are case
insensitive
# and include "TLSv1", "TLSv1.1", "TLSv1.2", "TLSv1.3" (OpenSSL >= 1.1.1) or
# any combination. To enable only TLSv1.2 and TLSv1.3, use:
#
# tls-protocols "TLSv1.2 TLSv1.3"

# Configure allowed ciphers. See the ciphers(1ssl) manpage for more information
# about the syntax of this string.
#
# Note: this configuration applies only to <= TLSv1.2.
#
# tls-ciphers DEFAULT:!MEDIUM

# Configure allowed TLSv1.3 ciphersuites. See the ciphers(1ssl) manpage for
more
# information about the syntax of this string, and specifically for TLSv1.3
# ciphersuites.
```



```

#
# tls-ciphersuites TLS_CHACHA20_POLY1305_SHA256

# When choosing a cipher, use the server's preference instead of the client
# preference. By default, the server follows the client's preference.
#
# tls-prefer-server-ciphers yes

# By default, TLS session caching is enabled to allow faster and less expensive
# reconnections by clients that support it. Use the following directive to
# disable
# caching.
#
# tls-session-caching no

# Change the default number of TLS sessions cached. A zero value sets the cache
# to unlimited size. The default size is 20480.
#
# tls-session-cache-size 5000

# Change the default timeout of cached TLS sessions. The default timeout is 300
# seconds.
#
# tls-session-cache-timeout 60

##### 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

# Remove RDB files used by replication in instances without persistence
# enabled. By default this option is disabled, however there are environments
# where for regulations or other security concerns, RDB files persisted on
# disk by masters in order to feed replicas, or stored on disk by replicas
# in order to load them for the initial synchronization, should be deleted
# ASAP. Note that this option ONLY WORKS in instances that have both AOF
# and RDB persistence disabled, otherwise is completely ignored.
#
# An alternative (and sometimes better) way to obtain the same effect is
# to use diskless replication on both master and replicas instances. However
# in the case of replicas, diskless is not always an option.
rdb-del-sync-files no

# 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>
#
# However this is not enough if you are using Redis ACLs (for Redis version
# 6 or greater), and the default user is not capable of running the PSYNC
# command and/or other commands needed for replication. In this case it's
# better to configure a special user to use with replication, and specify the
# masteruser configuration as such:
#
# masteruser <username>
#
# When masteruser is specified, the replica will authenticate against its
# master using the new AUTH form: AUTH <username> <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.
#
# 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

# -----
# WARNING: RDB diskless load is experimental. Since in this setup the replica
# does not immediately store an RDB on disk, it may cause data loss during
# failovers. RDB diskless load + Redis modules not handling I/O reads may also
# cause Redis to abort in case of I/O errors during the initial synchronization

```

```

# stage with the master. Use only if your do what you are doing.
# -----
#
# Replica can load the RDB it reads from the replication link directly from the
# socket, or store the RDB to a file and read that file after it was completely
# received from the master.
#
# In many cases the disk is slower than the network, and storing and loading
# the RDB file may increase replication time (and even increase the master's
# Copy on Write memory and slave buffers).
# However, parsing the RDB file directly from the socket may mean that we have
# to flush the contents of the current database before the full rdb was
# received. For this reason we have the following options:
#
# "disabled" - Don't use diskless load (store the rdb file to the disk first)
# "on-empty-db" - Use diskless load only when it is completely safe.
# "swapdb" - Keep a copy of the current db contents in RAM while parsing
#             the data directly from the socket. note that this requires
#             sufficient memory, if you don't have it, you risk an OOM kill.
repl-diskless-load disabled

# 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

##### KEYS TRACKING #####

# Redis implements server assisted support for client side caching of values.
# This is implemented using an invalidation table that remembers, using
# 16 millions of slots, what clients may have certain subsets of keys. In turn
# this is used in order to send invalidation messages to clients. Please
# to understand more about the feature check this page:
#
#   https://redis.io/topics/client-side-caching
#
# When tracking is enabled for a client, all the read only queries are assumed
# to be cached: this will force Redis to store information in the invalidation
# table. When keys are modified, such information is flushed away, and
# invalidation messages are sent to the clients. However if the workload is
# heavily dominated by reads, Redis could use more and more memory in order
# to track the keys fetched by many clients.
#
# For this reason it is possible to configure a maximum fill value for the
# invalidation table. By default it is set to 1M of keys, and once this limit
# is reached, Redis will start to evict keys in the invalidation table
# even if they were not modified, just to reclaim memory: this will in turn
# force the clients to invalidate the cached values. Basically the table
# maximum size is a trade off between the memory you want to spend server
# side to track information about who cached what, and the ability of clients
# to retain cached objects in memory.
#
# If you set the value to 0, it means there are no limits, and Redis will
# retain as many keys as needed in the invalidation table.

```



```

# In the "stats" INFO section, you can find information about the number of
# keys in the invalidation table at every given moment.
#
# Note: when key tracking is used in broadcasting mode, no memory is used
# in the server side so this setting is useless.
#
# tracking-table-max-keys 1000000

##### SECURITY #####

# Warning: since Redis is pretty fast an outside user can try up to
# 1 million passwords per second against a modern box. This means that you
# should use very strong passwords, otherwise they will be very easy to break.
# Note that because the password is really a shared secret between the client
# and the server, and should not be memorized by any human, the password
# can be easily a long string from /dev/urandom or whatever, so by using a
# long and unguessable password no brute force attack will be possible.

# Redis ACL users are defined in the following format:
#
# user <username> ... acl rules ...
#
# For example:
#
# user worker +@list +@connection ~jobs:* on >ffa9203c493aa99
#
# The special username "default" is used for new connections. If this user
# has the "nopass" rule, then new connections will be immediately authenticated
# as the "default" user without the need of any password provided via the
# AUTH command. Otherwise if the "default" user is not flagged with "nopass"
# the connections will start in not authenticated state, and will require
# AUTH (or the HELLO command AUTH option) in order to be authenticated and
# start to work.
#
# The ACL rules that describe what an user can do are the following:
#
# on          Enable the user: it is possible to authenticate as this user.
# off         Disable the user: it's no longer possible to authenticate
#             with this user, however the already authenticated connections
#             will still work.
# +<command>  Allow the execution of that command
# -<command>  Disallow the execution of that command
# +@<category> Allow the execution of all the commands in such category
#             with valid categories are like @admin, @set, @sortedset, ...
#             and so forth, see the full list in the server.c file where
#             the Redis command table is described and defined.
#             The special category @all means all the commands, but currently
#             present in the server, and that will be loaded in the future
#             via modules.
# +<command>|subcommand Allow a specific subcommand of an otherwise
#             disabled command. Note that this form is not
#             allowed as negative like -DEBUG|SEGFALT, but
#             only additive starting with "+".
# allcommands Alias for +@all. Note that it implies the ability to execute
#             all the future commands loaded via the modules system.
# nocommands  Alias for -@all.
# ~<pattern>  Add a pattern of keys that can be mentioned as part of
#             commands. For instance ~* allows all the keys. The pattern

```

```

# is a glob-style pattern like the one of KEYS.
# It is possible to specify multiple patterns.
# allkeys Alias for ~*
# resetkeys Flush the list of allowed keys patterns.
# ><password> Add this password to the list of valid password for the user.
# For example >mypass will add "mypass" to the list.
# This directive clears the "nopass" flag (see later).
# <<password> Remove this password from the list of valid passwords.
# nopass All the set passwords of the user are removed, and the user
# is flagged as requiring no password: it means that every
# password will work against this user. If this directive is
# used for the default user, every new connection will be
# immediately authenticated with the default user without
# any explicit AUTH command required. Note that the "resetpass"
# directive will clear this condition.
# resetpass Flush the list of allowed passwords. Moreover removes the
# "nopass" status. After "resetpass" the user has no associated
# passwords and there is no way to authenticate without adding
# some password (or setting it as "nopass" later).
# reset Performs the following actions: resetpass, resetkeys, off,
# -@all. The user returns to the same state it has immediately
# after its creation.
#
# ACL rules can be specified in any order: for instance you can start with
# passwords, then flags, or key patterns. However note that the additive
# and subtractive rules will CHANGE MEANING depending on the ordering.
# For instance see the following example:
#
# user alice on +@all -DEBUG ~* >somepassword
#
# This will allow "alice" to use all the commands with the exception of the
# DEBUG command, since +@all added all the commands to the set of the commands
# alice can use, and later DEBUG was removed. However if we invert the order
# of two ACL rules the result will be different:
#
# user alice on -DEBUG +@all ~* >somepassword
#
# Now DEBUG was removed when alice had yet no commands in the set of allowed
# commands, later all the commands are added, so the user will be able to
# execute everything.
#
# Basically ACL rules are processed left-to-right.
#
# For more information about ACL configuration please refer to
# the Redis web site at https://redis.io/topics/acl

# ACL LOG
#
# The ACL Log tracks failed commands and authentication events associated
# with ACLs. The ACL Log is useful to troubleshoot failed commands blocked
# by ACLs. The ACL Log is stored in memory. You can reclaim memory with
# ACL LOG RESET. Define the maximum entry length of the ACL Log below.
aclog-max-len 128

# Using an external ACL file
#
# Instead of configuring users here in this file, it is possible to use
# a stand-alone file just listing users. The two methods cannot be mixed:

```

```

# if you configure users here and at the same time you activate the external
# ACL file, the server will refuse to start.
#
# The format of the external ACL user file is exactly the same as the
# format that is used inside redis.conf to describe users.
#
# aclfile /etc/redis/users.acl

# IMPORTANT NOTE: starting with Redis 6 "requirepass" is just a compatibility
# layer on top of the new ACL system. The option effect will be just setting
# the password for the default user. Clients will still authenticate using
# AUTH <password> as usually, or more explicitly with AUTH default <password>
# if they follow the new protocol: both will work.
#
# requirepass foobared

# Command renaming (DEPRECATED).
#
# -----
# WARNING: avoid using this option if possible. Instead use ACLs to remove
# commands from the default user, and put them only in some admin user you
# create for administrative purposes.
# -----
#
# 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'.
#
# IMPORTANT: When Redis Cluster is used, the max number of connections is also
# shared with the cluster bus: every node in the cluster will use two
# connections, one incoming and another outgoing. It is important to size the
# limit accordingly in case of very large clusters.
#
# 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 one from the following behaviors:
#
# volatile-lru -> Evict using approximated LRU, only keys with an expire set.
# allkeys-lru -> Evict any key using approximated LRU.
# volatile-lfu -> Evict using approximated LFU, only keys with an expire set.
# allkeys-lfu -> Evict any key using approximated LFU.
# volatile-random -> Remove a random key having 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 hincrby hincrby 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

# Redis reclaims expired keys in two ways: upon access when those keys are
# found to be expired, and also in background, in what is called the
# "active expire key". The key space is slowly and interactively scanned
# looking for expired keys to reclaim, so that it is possible to free memory
# of keys that are expired and will never be accessed again in a short time.
#
# The default effort of the expire cycle will try to avoid having more than
# ten percent of expired keys still in memory, and will try to avoid consuming
# more than 25% of total memory and to add latency to the system. However
# it is possible to increase the expire "effort" that is normally set to
# "1", to a greater value, up to the value "10". At its maximum value the
# system will use more CPU, longer cycles (and technically may introduce
# more latency), and will tolerate less already expired keys still present
# in the system. It's a tradeoff between memory, CPU and latency.
#
# active-expire-effort 1

##### 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

# It is also possible, for the case when to replace the user code DEL calls
# with UNLINK calls is not easy, to modify the default behavior of the DEL
# command to act exactly like UNLINK, using the following configuration
# directive:

lazyfree-lazy-user-del no

##### THREADED I/O #####

# Redis is mostly single threaded, however there are certain threaded
# operations such as UNLINK, slow I/O accesses and other things that are
# performed on side threads.
#
# Now it is also possible to handle Redis clients socket reads and writes
# in different I/O threads. Since especially writing is so slow, normally

```

```

# Redis users use pipelining in order to speedup the Redis performances per
# core, and spawn multiple instances in order to scale more. Using I/O
# threads it is possible to easily speedup two times Redis without resorting
# to pipelining nor sharding of the instance.
#
# By default threading is disabled, we suggest enabling it only in machines
# that have at least 4 or more cores, leaving at least one spare core.
# Using more than 8 threads is unlikely to help much. We also recommend using
# threaded I/O only if you actually have performance problems, with Redis
# instances being able to use a quite big percentage of CPU time, otherwise
# there is no point in using this feature.
#
# So for instance if you have a four cores boxes, try to use 2 or 3 I/O
# threads, if you have a 8 cores, try to use 6 threads. In order to
# enable I/O threads use the following configuration directive:
#
# io-threads 4
#
# Setting io-threads to 1 will just use the main thread as usually.
# When I/O threads are enabled, we only use threads for writes, that is
# to thread the write(2) syscall and transfer the client buffers to the
# socket. However it is also possible to enable threading of reads and
# protocol parsing using the following configuration directive, by setting
# it to yes:
#
# io-threads-do-reads no
#
# Usually threading reads doesn't help much.
#
# NOTE 1: This configuration directive cannot be changed at runtime via
# CONFIG SET. As this feature currently does not work when SSL is
# enabled.
#
# NOTE 2: If you want to test the Redis speedup using redis-benchmark, make
# sure you also run the benchmark itself in threaded mode, using the
# --threads option to match the number of Redis threads, otherwise you'll not
# be able to notice the improvements.

##### KERNEL OOM CONTROL #####

# On Linux, it is possible to hint the kernel OOM killer on what processes
# should be killed first when out of memory.
#
# Enabling this feature makes Redis actively control the oom_score_adj value
# for all its processes, depending on their role. The default scores will
# attempt to have background child processes killed before all others, and
# replicas killed before masters.

oom-score-adj no

# When oom-score-adj is used, this directive controls the specific values used
# for master, replica and background child processes. Values range -1000 to
# 1000 (higher means more likely to be killed).
#
# Unprivileged processes (not root, and without CAP_SYS_RESOURCE capabilities)
# can freely increase their value, but not decrease it below its initial
# settings.
#

```

```
# Values are used relative to the initial value of oom_score_adj when the server
# starts. Because typically the initial value is 0, they will often match the
# absolute values.
```

```
oom-score-adj-values 0 200 800
```

```
##### 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

# This option, when set to yes, allows nodes to serve read traffic while the
# the cluster is in a down state, as long as it believes it owns the slots.
#
# This is useful for two cases. The first case is for when an application
# doesn't require consistency of data during node failures or network
# partitions.
# One example of this is a cache, where as long as the node has the data it
# should be able to serve it.
#
# The second use case is for configurations that don't meet the recommended
# three shards but want to enable cluster mode and scale later. A
# master outage in a 1 or 2 shard configuration causes a read/write outage to
# the
# entire cluster without this option set, with it set there is only a write
# outage.
# Without a quorum of masters, slot ownership will not change automatically.
#
# cluster-allow-reads-when-down 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)
# t      Stream commands
# m      Key-miss events (Note: It is not included in the 'A' class)
# A      Alias for g$lshxet, so that the "AKE" string means all the events
#        (Except key-miss events which are excluded from 'A' due to their
#        unique nature).
#
# 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 ""

##### GOPHER SERVER #####

# Redis contains an implementation of the Gopher protocol, as specified in
# the RFC 1436 (https://www.ietf.org/rfc/rfc1436.txt).
#
# The Gopher protocol was very popular in the late '90s. It is an alternative
# to the web, and the implementation both server and client side is so simple
# that the Redis server has just 100 lines of code in order to implement this
# support.
#
# What do you do with Gopher nowadays? Well Gopher never *really* died, and
# lately there is a movement in order for the Gopher more hierarchical content
# composed of just plain text documents to be resurrected. Some want a simpler
# internet, others believe that the mainstream internet became too much
# controlled, and it's cool to create an alternative space for people that
# want a bit of fresh air.
#
# Anyway for the 10th birthday of the Redis, we gave it the Gopher protocol
# as a gift.
#
# --- HOW IT WORKS? ---
#
# The Redis Gopher support uses the inline protocol of Redis, and specifically
# two kind of inline requests that were anyway illegal: an empty request
# or any request that starts with "/" (there are no Redis commands starting
# with such a slash). Normal RESP2/RESP3 requests are completely out of the
# path of the Gopher protocol implementation and are served as usually as well.
#
# If you open a connection to Redis when Gopher is enabled and send it
# a string like "/foo", if there is a key named "/foo" it is served via the
# Gopher protocol.
#
# In order to create a real Gopher "hole" (the name of a Gopher site in Gopher
# talking), you likely need a script like the following:
#
# https://github.com/antirez/gopher2redis
#
# --- SECURITY WARNING ---
#
# If you plan to put Redis on the internet in a publicly accessible address
# to server Gopher pages MAKE SURE TO SET A PASSWORD to the instance.
# Once a password is set:
#
# 1. The Gopher server (when enabled, not by default) will still serve
#    content via Gopher.
# 2. However other commands cannot be called before the client will
#    authenticate.
#
# So use the 'requirepass' option to protect your instance.
#

```

```

# To enable Gopher support uncomment the following line and set
# the option from no (the default) to yes.
#
# gopher-enabled no

##### 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 entries 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 "activeresharding 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 "activeresharding yes" if you don't have such hard requirements but
# want to free memory asap when possible.
activeresharding 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 as 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 to 512 mb. However you can change this limit
# here, but must be 1mb or greater
#
# 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 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 #####
#
# 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 no

# 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, to be used when the lower
# threshold is reached
# active-defrag-cycle-min 1

# Maximal effort for defrag in CPU percentage, to be used when the upper
# threshold is reached
# active-defrag-cycle-max 25

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

# Jemalloc background thread for purging will be enabled by default
jemalloc-bg-thread yes

# It is possible to pin different threads and processes of Redis to specific
# CPUs in your system, in order to maximize the performances of the server.
# This is useful both in order to pin different Redis threads in different
# CPUs, but also in order to make sure that multiple Redis instances running
# in the same host will be pinned to different CPUs.
#
# Normally you can do this using the "taskset" command, however it is also
# possible to this via Redis configuration directly, both in Linux and FreeBSD.
#
# You can pin the server/IO threads, bio threads, aof rewrite child process, and
# the bgsave child process. The syntax to specify the cpu list is the same as
# the taskset command:
#
# Set redis server/io threads to cpu affinity 0,2,4,6:
# server_cpulist 0-7:2
#
# Set bio threads to cpu affinity 1,3:
# bio_cpulist 1,3
#
# Set aof rewrite child process to cpu affinity 8,9,10,11:
# aof_rewrite_cpulist 8-11
#
# Set bgsave child process to cpu affinity 1,10,11
# bgsave_cpulist 1,10-11
```

测试 redis-cli连接上来

```
docker exec -it 运行着redis服务容器的ID redis-cli
```



测试持久化文件生成



第 8 章 将镜像推送到阿里云

本地镜像发布到阿里云流程



镜像生成方法

- 1、前面的Dockerfile
- 2、从容器中创建一个新的镜像

```
docker commit [OPTIONS] 容器ID [REPOSITORY[:TAG]]
```



将本地镜像推送到阿里云

1、本地镜像素材原型



- 2、阿里云开发者平台

<https://promotion.aliyun.com/ntms/act/kubernetes.html>



3、创建镜像仓库



命名空间

仓库名称

4、将镜像推送到registry

```
$ sudo docker login --username=white3e registry.cn-shenzhen.aliyuncs.com
$ sudo docker tag [ImageId] registry.cn-shenzhen.aliyuncs.com/ggccqq/mycentos:[镜像版本号]
$ sudo docker push registry.cn-shenzhen.aliyuncs.com/ggccqq/mycentos:[镜像版本号]
其中[ImageId][镜像版本]自己填写
```



5、公有云可以查询得到



6、查看详情



将阿里云上的镜像下载到本地



至此Docker基础篇完结

2020-10-4