# Docker

1. **docker介绍**

Docker 是一个用于开发、发布和运行应用程序的开放平台。Docker 使您能够将应用程序与基础架构分离，以便您可以快速交付软件。使用 Docker，您可以像管理应用程序一样管理基础架构。通过利用 Docker 的快速发布、测试和部署代码的方法，您可以显著减少编写代码和在生产环境中运行代码之间的延迟。

Docker 提供了在松散隔离的环境中打包和运行应用程序（称为容器）的功能。隔离和安全性允许您在给定主机上同时运行多个容器。容器是轻量级的，包含运行应用程序所需的所有内容，因此您无需依赖主机上当前安装的内容。您可以在工作时轻松共享容器，并确保与之共享的每个人都能获得以相同方式工作的相同容器。

**Docker 提供了工具和平台来管理容器的生命周期：**

* 使用容器开发应用程序及其支持组件。
* 容器将成为分发和测试应用程序的单元。
* 准备就绪后，将应用程序作为容器或编排服务部署到生产环境中。无论您的生产环境是本地数据中心、云提供商还是两者的混合体，这都是一样的。
* **可以使用docker做什么**
* **快速、一致地交付您的应用程序**

Docker 允许开发人员使用提供应用程序和服务的本地容器在标准化环境中工作，从而简化了开发生命周期。容器非常适合持续集成和持续交付 （CI/CD） 工作流。

* **响应式部署和扩展**

Docker 基于容器的平台允许高度可移植的工作负载。Docker 容器可以在开发人员的本地笔记本电脑、数据中心的物理机或虚拟机、云提供商或混合环境中运行。

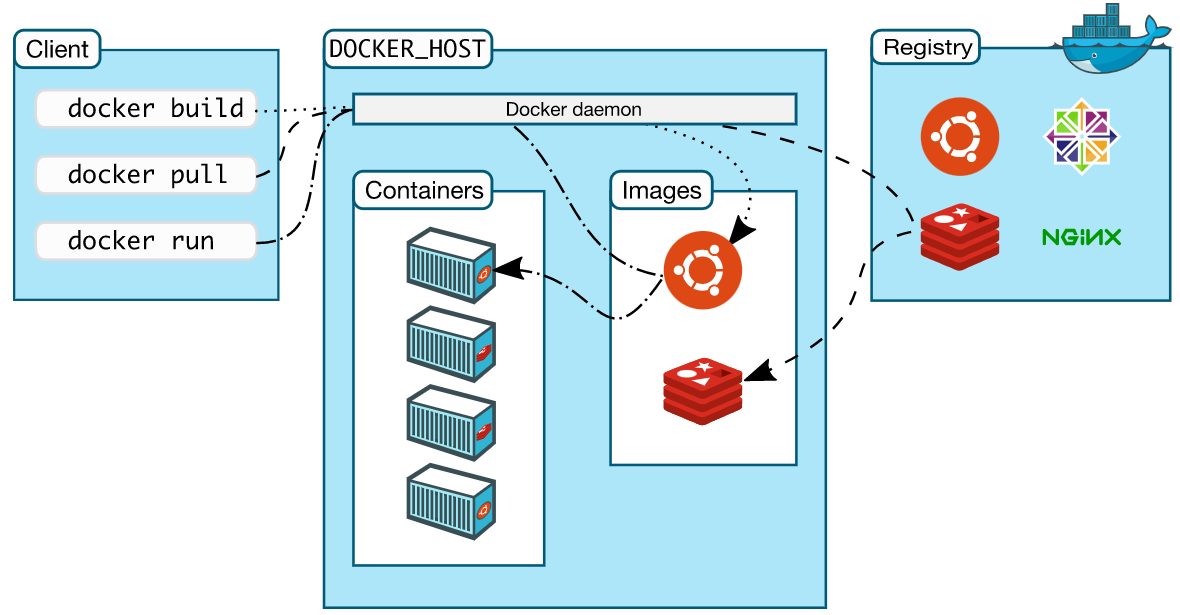
Docker 的可移植性和轻量级特性还使得能够近乎实时地动态管理工作负载、根据业务需求扩展或拆除应用程序和服务变得容易。

* **在同一硬件上运行更多工作负载**

Docker 是轻量级和快速的。它为基于虚拟机管理程序的虚拟机提供了一种可行且经济高效的替代方案，因此您可以使用更多的计算容量来实现业务目标。Docker 非常适合高密度环境以及需要用更少资源完成更多工作的中小型部署。

* **Docker体系结构**

Docker 使用客户端-服务器体系结构。Docker客户端与 Docker守护程序通信，守护程序负责构建、运行和分发 Docker 容器。Docker 客户端和守护程序可以在同一系统上运行，也可以将 Docker 客户端连接到远程 Docker 守护程序。Docker 客户端和守护程序使用 REST API、UNIX 套接字或网络接口进行通信。另一个 Docker 客户端是 Docker Compose，它允许您使用由一组容器组成的应用程序。

****

* **镜像（images）**

映像是一个只读模板，其中包含有关创建 Docker 容器的说明。通常，一个映像基于另一个映像，并具有一些额外的自定义。例如，您可以构建一个基于该映像的映像，但会安装 Apache Web 服务器和应用程序，以及使应用程序运行所需的配置详细信息。Ubuntu

您可以创建自己的映像，也可以只使用其他人创建并在注册表中发布的映像。若要生成自己的映像，请使用简单的语法创建一个Dockerfile，用于定义创建映像并运行映像所需的步骤。Dockerfile 中的每条指令都会在映像中创建一个层。更改 Dockerfile 并重新生成映像时，只会重新生成已更改的层。与其他虚拟化技术相比，这是使映像如此轻巧、小巧和快速的部分原因。

* **容器**

容器是映像的可运行实例。您可以使用 Docker API 或 CLI 创建、启动、停止、移动或删除容器。可以将容器连接到一个或多个网络，将存储附加到该网络，甚至可以根据其当前状态创建新映像。

默认情况下，容器与其他容器及其主机相对隔离。您可以控制容器的网络、存储或其他底层子系统与其他容器或主机的隔离程度。

容器由其映像以及您在创建或启动容器时提供给它的任何配置选项定义。删除容器后，未存储在持久性存储中的任何对其状态所做的任何更改都将消失。

1. **Docker与虚拟机的区别**

* **虚拟机**

虚拟机（virtual machine）就是带环境安装的一种解决方案。它可以在一种操作系统里面运行另一种操作系统，比如在 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。

**（1）资源占用多**

虚拟机会独占一部分内存和硬盘空间。它运行的时候，其他程序就不能使用这些资源了。哪怕虚拟机里面的应用程序，真正使用的内存只有 1MB，虚拟机依然需要几百 MB 的内存才能运行。

**（2）冗余步骤多**

虚拟机是完整的操作系统，一些系统级别的操作步骤，往往无法跳过，比如用户登录。

**（3）启动慢**

启动操作系统需要多久，启动虚拟机就需要多久。可能要等几分钟，应用程序才能真正运行。

* **Linux容器**

由于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。**或者说，在正常进程的外面套了一个[保护层](https://opensource.com/article/18/1/history-low-level-container-runtimes)。对于容器里面的进程来说，它接触到的各种资源都是虚拟的，从而实现与底层系统的隔离。

由于容器是进程级别的，相比虚拟机有很多优势。

**（1）启动快**

容器里面的应用，直接就是底层系统的一个进程，而不是虚拟机内部的进程。所以，启动容器相当于启动本机的一个进程，而不是启动一个操作系统，速度就快很多。

**（2）资源占用少**

容器只占用需要的资源，不占用那些没有用到的资源；虚拟机由于是完整的操作系统，不可避免要占用所有资源。另外，多个容器可以共享资源，虚拟机都是独享资源。

**（3）体积小**

容器只要包含用到的组件即可，而虚拟机是整个操作系统的打包，所以容器文件比虚拟机文件要小很多。

总之，容器有点像轻量级的虚拟机，能够提供虚拟化的环境，但是成本开销小得多。

* **Docker**

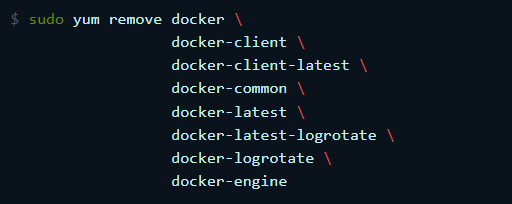
**Docker 属于 Linux 容器的一种封装，提供简单易用的容器使用接口。**它是目前最流行的 Linux 容器解决方案。

Docker 将应用程序与该程序的依赖，打包在一个文件里面。运行这个文件，就会生成一个虚拟容器。程序在这个虚拟容器里运行，就好像在真实的物理机上运行一样。有了 Docker，就不用担心环境问题。

总体来说，Docker 的接口相当简单，用户可以方便地创建和使用容器，把自己的应用放入容器。容器还可以进行版本管理、复制、分享、修改，就像管理普通的代码一样。

1. **docker卸载安装**

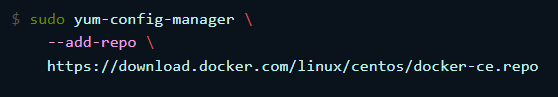
* **安装**
* **卸载旧版本**

****

* **使用存储库安装；设置存储库**

在新主机上首次安装Docker Engine之前，需要设置Docker存储库。之后，您可以从存储库安装和更新Docker。





注意：为加快yum的安装速度，我们可以使用阿里云的源。

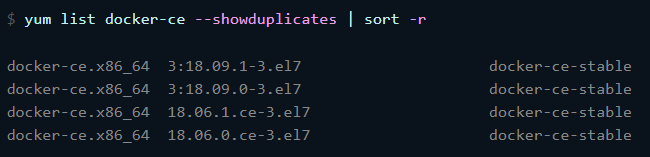
<https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo>

* **开始安装**

****

此命令安装Docker，但不启动Docker。它还创建了一个docker组，但是默认情况下不会向该组添加任何用户

* **安装指定版本**
* **查看可用版本**

****

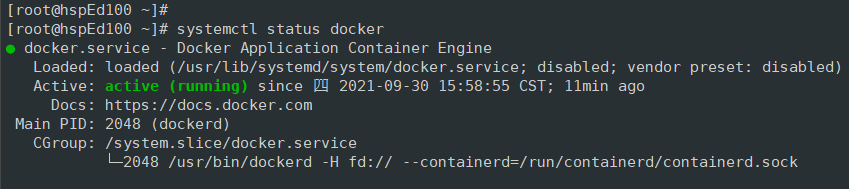
* **开始安装**

****

* **启动docker**

****

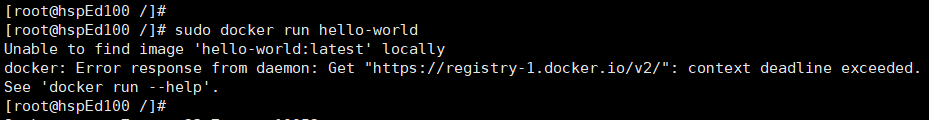
* **查看docker启动是否启动**

****

* **运行hello world映像验证Docker引擎是否已正确安装**

****

注意：此操作可能会报以下错误，可通过配置国内镜像解决；

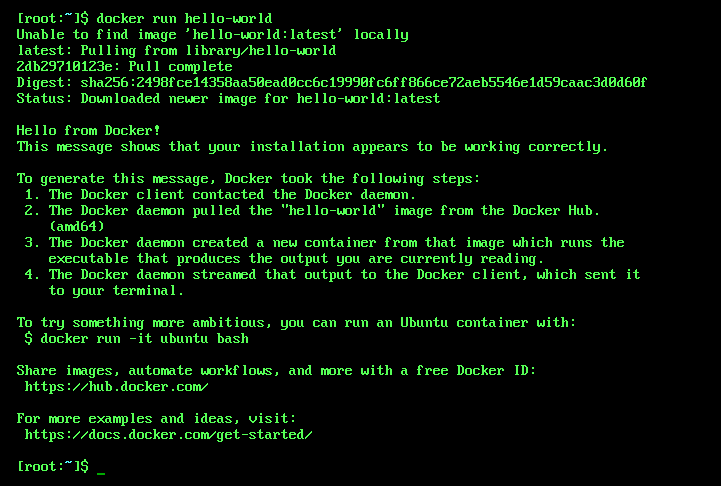


* **配置镜像加速**



* **检测镜像仓库配置是否生效**





* **卸载**

****

****

**四、快速入门**

* **什么是容器：**

简而言之，容器是计算机上的沙盒进程，与主机上的所有其他进程隔离。这种隔离利用了[内核命名空间和cgroups，](https://medium.com/@saschagrunert/demystifying-containers-part-i-kernel-space-2c53d6979504)这些特性已经在Linux中存在了很长时间。Docker 致力于使这些功能平易近人且易于使用。

* 是映像的可运行实例。您可以使用 DockerAPI 或 CLI 创建、启动、停止、移动或删除容器。
* 可以在本地机器、虚拟机上运行或部署到云中。
* 可移植（可以在任何操作系统上运行）
* 容器彼此隔离，并运行自己的软件、二进制文件和配置。
* **什么是容器镜像**

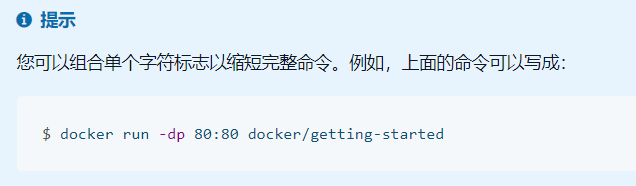
运行容器时，它使用隔离的文件系统。此自定义文件系统由**容器映像**提供。由于映像包含容器的文件系统，因此它必须包含运行应用程序所需的一切 - 所有依赖项、配置、脚本、二进制文件等。该映像还包含容器的其他配置，例如环境变量、要运行的默认命令和其他元数据。

* **运行容器案例：**

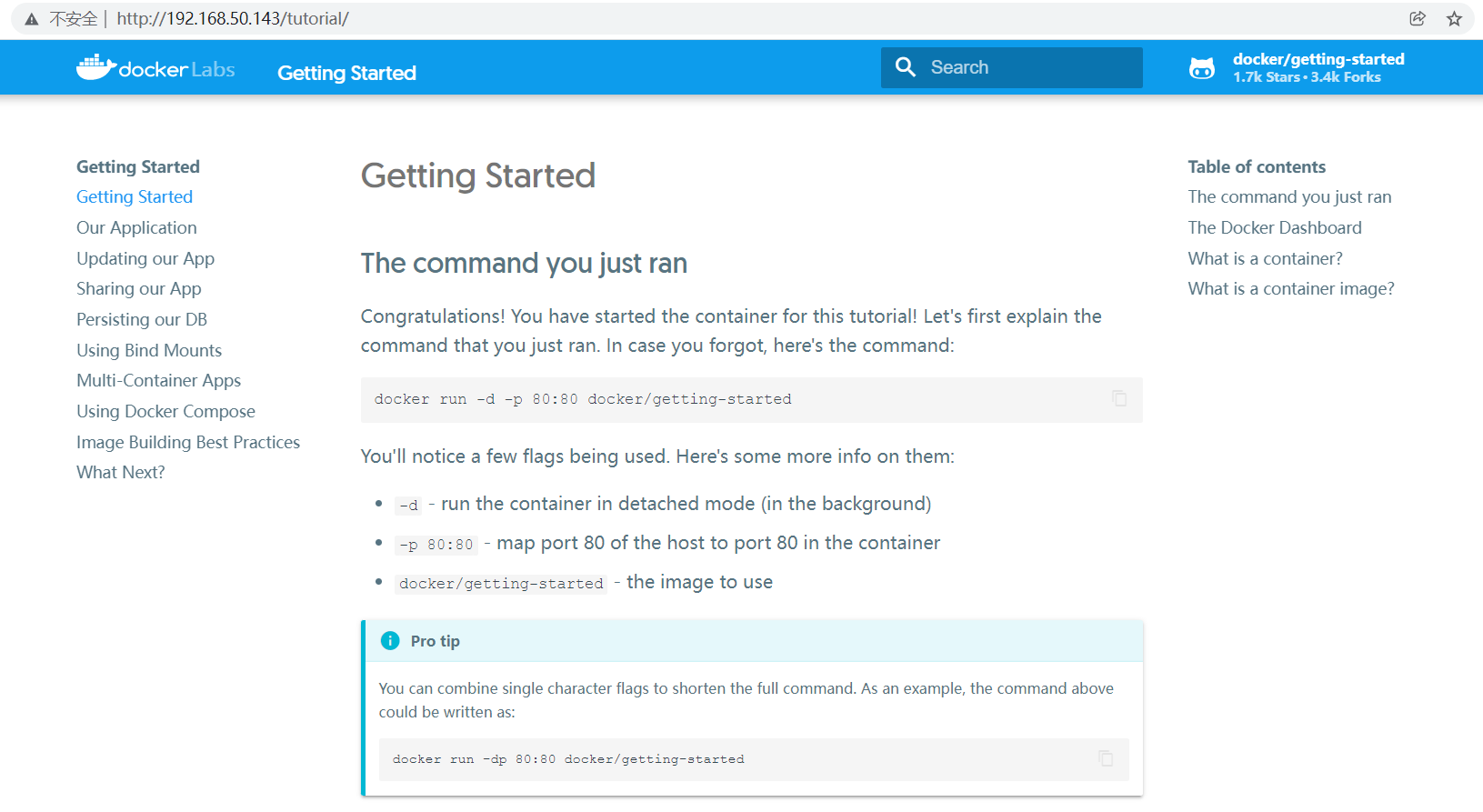
docker run -d -p 80:80 docker/getting-started



* -d- 在分离模式下运行容器（在后台）
* -p 80:80- 将主机的端口 80 映射到容器中的端口 80
* --name- 给容器起个别名
* docker/getting-started- 要使用的图像



* **访问启动的容器**

****

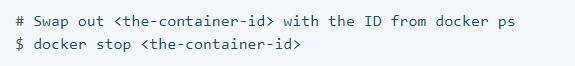
* **查看当前正在运行的容器:**

****

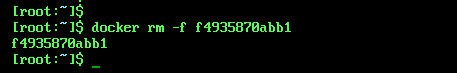
* **查看所有容器**

docker ps -a

* **停止容器运行**

****

* **删除停止的容器**

****

* **启动已终止的容器**

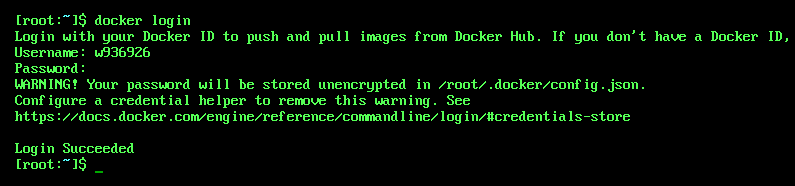
docker start 容器ID

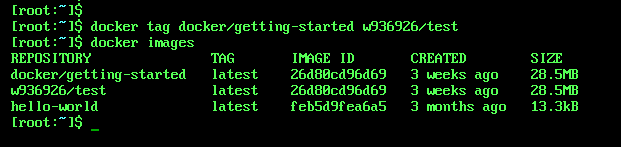
* **重启已终止的容器**

docket restart 容器ID

* **共享应用程序**
* [注册](https://www.docker.com/pricing?utm_source=docker&utm_medium=webreferral&utm_campaign=docs_driven_upgrade)或 登录到[Docker Hub](https://hub.docker.com/" \t "_blank)。
* 单击**创建存储库**按钮。
* 对于存储库名称，请使用 。确保可见性为 。getting-startedPublic
* 点击**创建**按钮！







* **保留数据库**

如果您没有注意到，每次启动容器时，我们的待办事项列表都会被擦除干净。

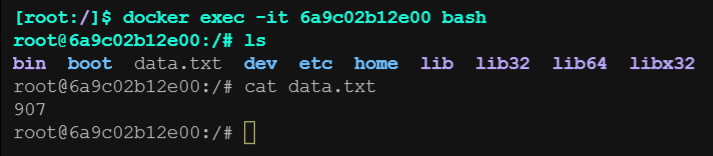
* 实践：

****

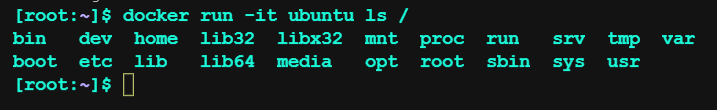
启动一个 bash shell 并调用两个命令。第一部分选取一个随机数并将其写入 。第二个命令是简单地监视文件以保持容器运行。&&/data.txt

* **进入容器**

docker exec -it ubuntu bash



* 启动另一个容器（相同的映像），我们将看到我们没有相同的文件



* **停止并删除第一个容器**

docker rm -f 容器ID

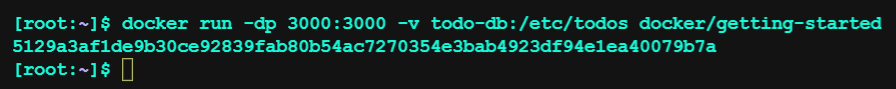
* **容器卷-解决以上问题**

[卷](https://docs.docker.com/storage/volumes/)提供了将容器的特定文件系统路径连接回主机的功能。如果装载了容器中的目录，则在主机上也会看到该目录中的更改。如果我们在容器重新启动时挂载相同的目录，我们将看到相同的文件。

* **创建卷**

****

* **启动待办事项应用容器，但添加标志以指定卷装载。我们将使用命名卷并将其装载到 ，这将捕获在路径上创建的所有文件。-v/etc/todos**

****

1. **Docker相关命令**

* **docker info**

查看docker相关信息

* **docker version**

查看当前docker版本

* **docker –help/docker**

查看相关命令

1. **镜像相关命令**

* **docker images**

查看本地所有镜像

* **docker images 镜像名**

查看某个镜像

* **docker images -q**

查看所有镜像ID

* **docker images -a**

查看所有镜像类似docker images

* **docker pull mysql:版本号**

拉取相关镜像

* **docker search 镜像名**

查看相关镜像

* **docker image rm 镜像名/ID**

删除镜像

* **docker image rm -f 镜像名/ID**

强制删除镜像

* **docker rmi 镜像名/ID**

删除镜像（简化）

* **docker rmi -f $(docker images -q)**

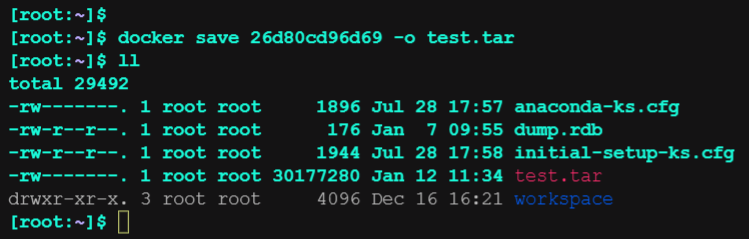
强制删除本地所有镜像

* **docker load -I tar包**

把tar包导入到本地镜像

* **docker save 镜像名/ID -o test.tar**

备份镜像



1. **容器相关命令**

* **docker run 镜像名/ID**

运行一个镜像生成一个容器实例

* **相关参数**
* **-p（**把宿主机端口映射到容器端口，后面可以跟多个**）**
* **-d（**以后台的方式启动**）**
* **--name（**给容器起一个别名**）**
* **docker ps**

查看当前正在运行的容器

* **相关参数**
* **-a （**查看所有容器**）**
* **-q （**返回正在运行的容器ID**）**

****

* **-qa（**返回所有容器ID**）**
* **docker start 容器名/ID**

启动一个已经关闭的容器

* **docker restart 容器名/ID**

重启容器

* **docker stop 容器名/ID**

停止容器

* **docker rm 容器名/ID**

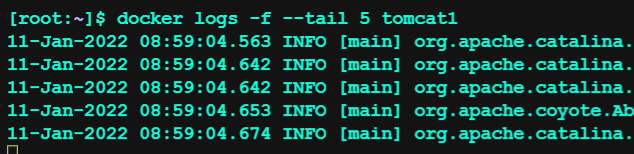
删除一个停止的容器

* **相关参数**
* **-f（**强制删除一个容器，先停止容器然后删除**）**

* **docker rm -f $(docker ps -qa)（**删除所有的容器**）**
* **docker logs 容器名/ID**

查看容器日志

* **相关参数**
* **-f （**实时查看容器日志**）**
* **-t （**打印时间戳**）**
* **--tail 5（**显示日志文件的最后5行**）**

****

* **docker top 容器名/ID**

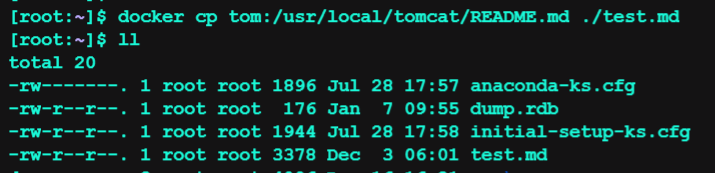
查看容器内的进程

* **docker exec -it 容器名/ID <相关命令>**

进入容器

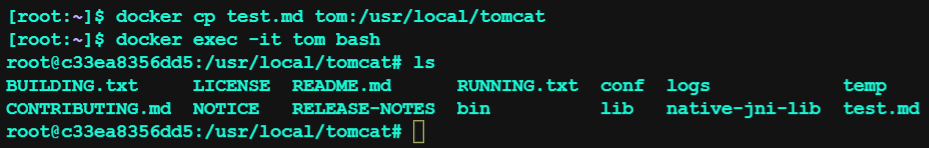
* **docker cp 容器名/ID：容器中的文件路径 复制到宿主机的路径地址**

复制容器中的文件到宿主机



* **docker cp 文件/目录名 容器名/ID：复制到容器的路径地址**

把宿主机文件复制到容器中



* **docker inspect 容器名/ID**

查看容器内部的信息

* **将容器打包成功新的镜像**

方便共享，不需要每次都要部署

* **docker commit 容器名/ID image-name：tag**

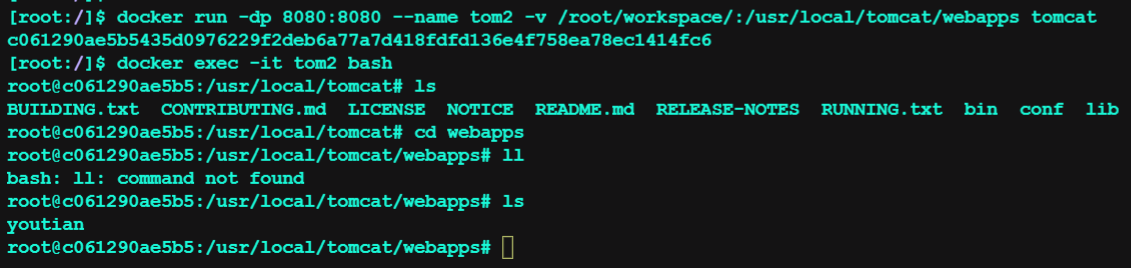
**相关参数：**

* **-m（**描述**）**
* **-a（**作者**）**

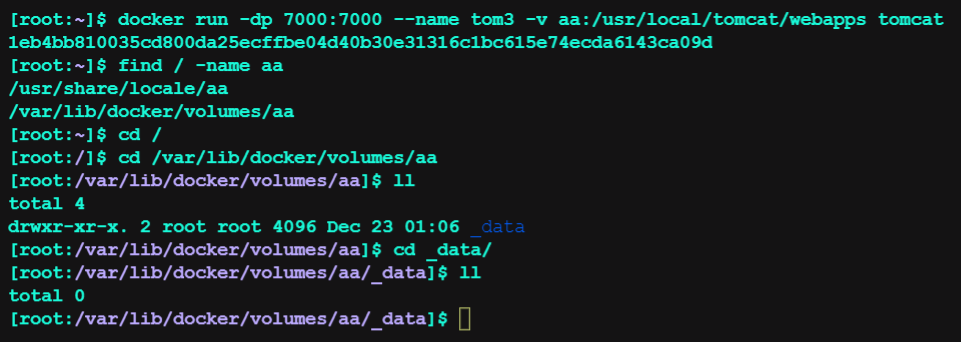
1. **数据卷进阶（Volume）**

实现宿主机与容器之间的文件共享

* **特点：**
* 容器之间的数据共享和重用
* 对数据卷的修改会立刻影响到对应的容器
* 对数据卷的更新修改，不会影响镜像
* 数据卷会一直存在，即使容器被删除
* **相关参数**
* **-v 宿主机目录地址（绝对路径）：容器内目录地址**

****

* **高级用法例子**

****

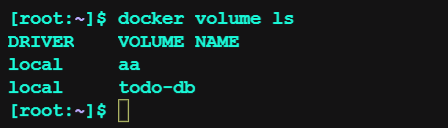
说明：aa代表一个数据卷的别名，当数据卷别名不存在时，docker会自动创建这个数据卷的同时自动映射宿主机中某个目录。

同时在启动容器时，会将aa对应的容器目录中全部内容复制到该目录中

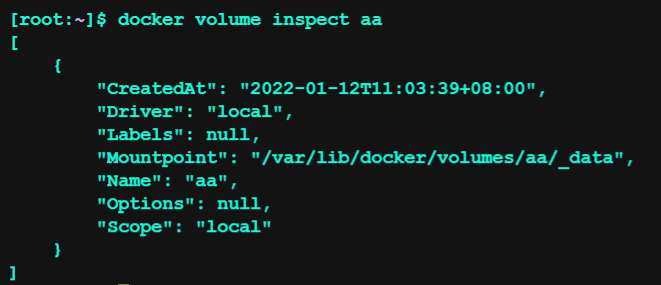
* **设置宿主机数据卷对应的容器为只读操作，只有宿主机修改数据卷才会影响之间的互通（:ro）**

****

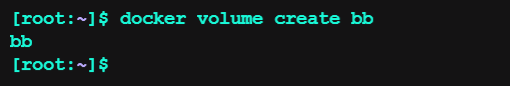
* **相关命令**
* **查看数据卷**

****

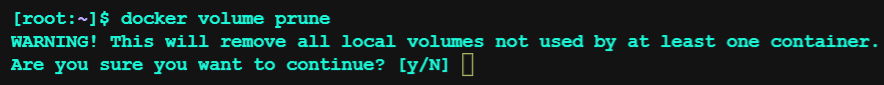
* **查看某个数据卷信息**

****

* **创建数据卷**

****

* **删除没有使用的数据卷**

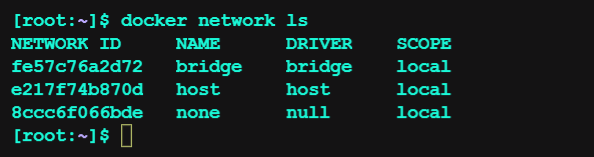
****

* **删除指定数据卷**

****

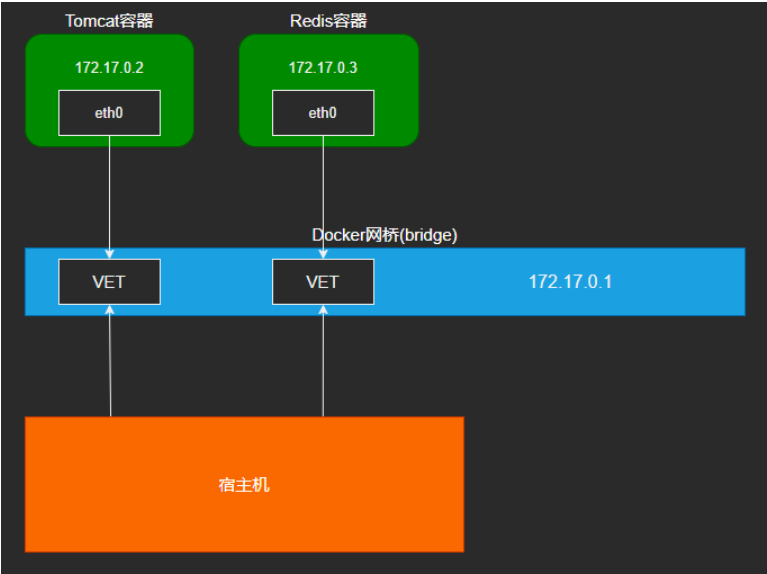
1. **Docker网络通信机制及网桥的使用**

当Docker启动时，会自动在主机上创建一个docker虚拟网桥，实际上是 Linux 的一个 **bridge**，可以理解为一个软件交换机，它会在挂载到它的网口之间进行转发。同时，Docker 随机分配一个本地未占用的私有网段中的一个地址给 docker 接口。比如典型的 172.17.0.1，掩码为 255.255.0.0。此后启动的容器内的网口也会自动分配一个同一网段（172.17.0.0/16）的地址

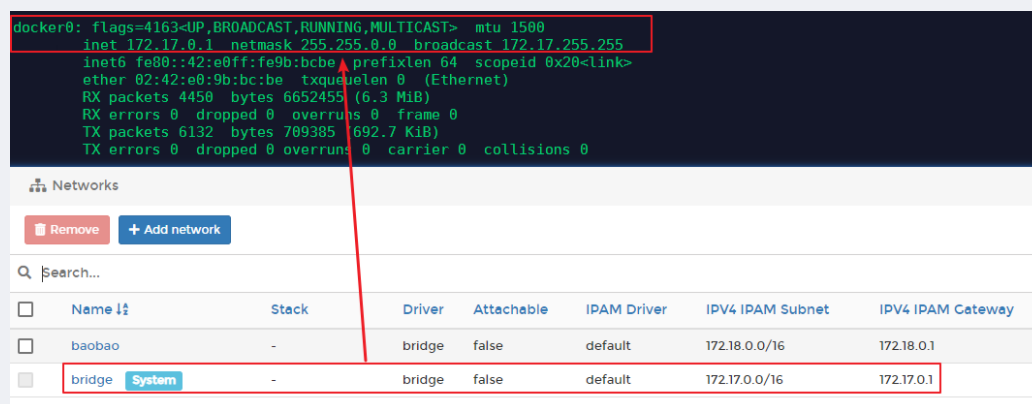




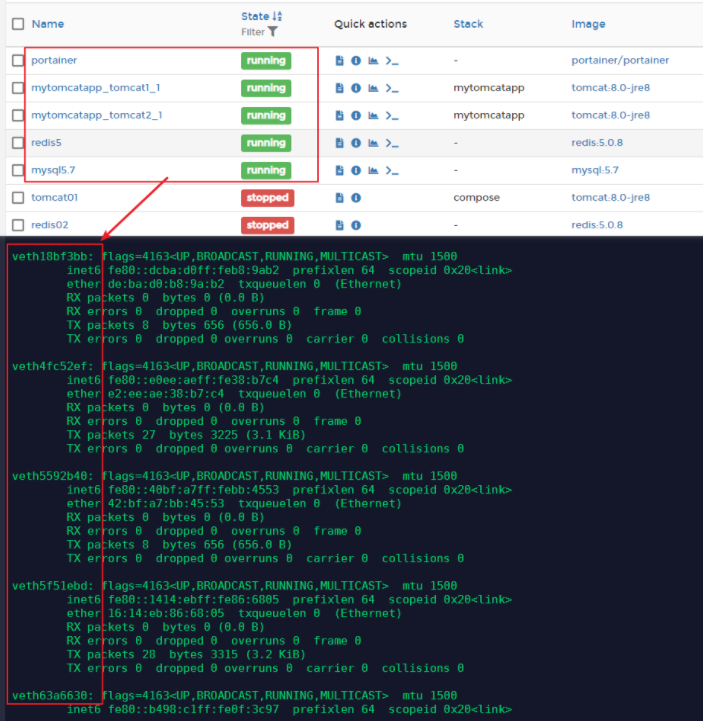
当创建一个 Docker 容器的时候，同时会创建了一对 veth pair 接口（当数据包发送到一个接口时，另外一个接口也可以收到相同的数据包）。这对接口一端在容器内，即 eth0；另一端在宿主机本地并被挂载到 docker 网桥，名称以 veth 开头（例如 vethAQI2QT）。通过这种方式，宿主机可以跟容器通信，容器之间也可以通过docker网桥相互通信。Docker就创建了在宿主机和所有容器之间一个虚拟共享网络



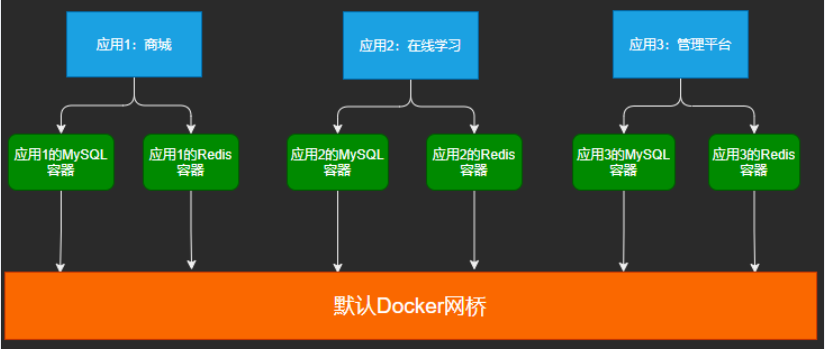
安装docker时创建的默认docker网桥，用docker network ls查看其名称是bridge，如果在宿主机上用ifconfig查看，实际上是对应了docker0虚拟网卡



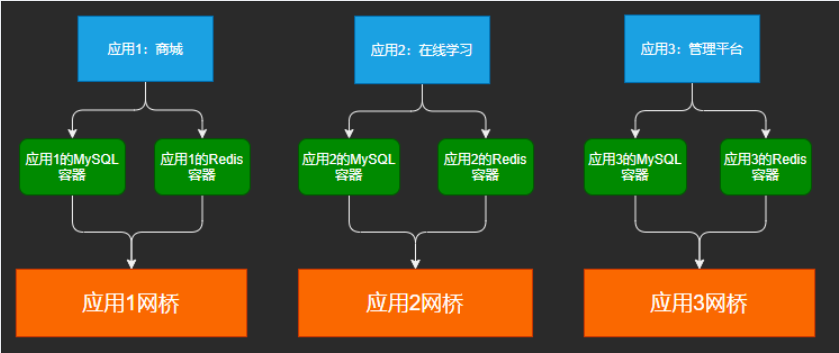
我们启动的所有容器，如果不做特殊的网络指定，默认都是连接到docker网桥上。**每启动一个容器，都会在宿主机上创建一个veth开头的虚拟网卡，用来进行容器与宿主机、容器与容器间的通信**。容器通过DHCP获取一个与docker0同网段的IP地址



注意：我们启动的所有容器，默认都是连接到docker网桥上，但是这不是一个好的实践。假设我们有多个应用，每个应用启动了自己独有的mysql、redis、tomcat等容器，那么默认情况下所有容器都连接到同一个网桥，此时应用1的tomcat发送给应用1自己的mysql的数据，应用2和应用3的所有容器也都能收到，这增加了不必要的网络流量压力：



**所以最佳的方式是：以应用为单位划分网络，同个应用对应的容器连接到1个网络，不同应用之间的容器做到网络隔离**

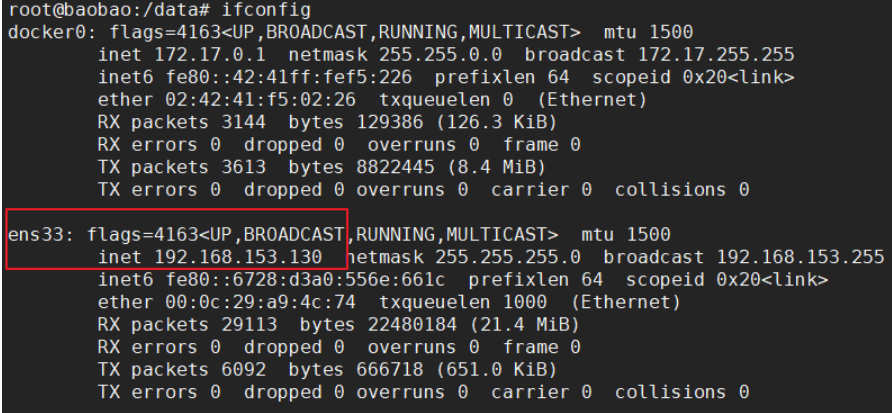
****

docker默认的网络除了bridge外，还有host和none

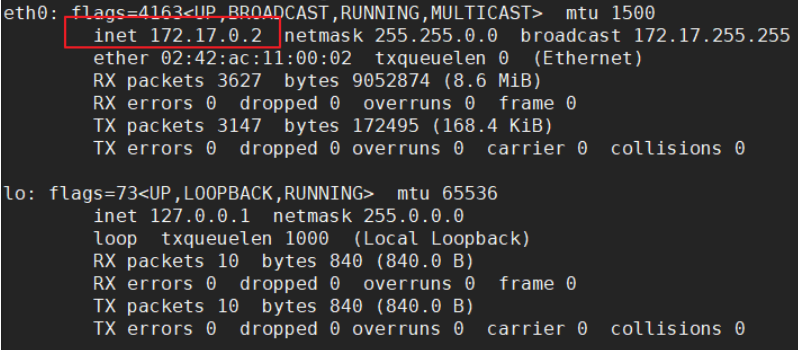
* host：如果指定容器网络为host，那么表示容器与宿主机共享网络，即容器的ip与宿主机相同。这样做的好处是访问宿主机就相当于访问容器，提升网络性能，不需要宿主机转发给容器。坏处是容器与宿主机共享端口，不能冲突。我们可以在启动容器时通过--network host指定为host网络



当指定网络为host之后，进入容器可以发现容器拥有与宿主机相同的ip配置

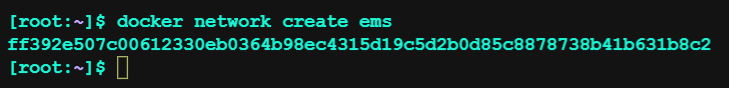


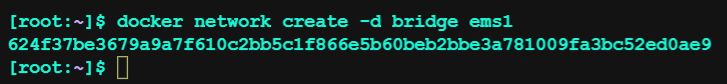
而默认bridge模式的容器只有自己的ip



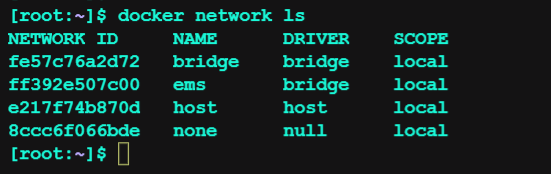
host模式可以用于所有应用和中间件部署在同一台机器的情况下，应用在容器中连接中间件只需指定ip为localhost即可，因为此时localhost就是代表了宿主机

* none：如果指定容器的网络为none，那么将无法通过网络访问容器，容器只有一个本地回环地址。如果禁止容器联网，为了安全可以采用这种方式
* **网桥的使用**
* **自定义网桥**

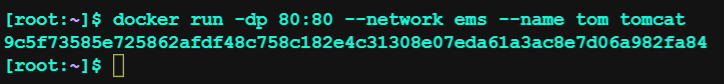
****

****

* **查看网桥**

****

* **运行容器指定所在网桥**

****

****

注意：一旦启动容器时，指定网桥之后，可以任何这个网桥关联的容器中使用容器别名进行与其它容器通信





注意：采用容器名称互相通信的方式，必须在创建容器时指定自定义网络才会生效。如果容器连接到默认的docker0网络是无法用容器名通信的；如果没有给容器指定自定义网络，那么会连接到默认的docker0网段，此时可以用ip进行容器互联，但是这个ip是通过DHCP动态分配的，所以这种情况下更推荐直接用宿主机ip+容器映射端口进行容器间通信

如果指定了自定义网络，那么推荐用容器名称进行通信

* **将容器连接另一个网络**

****

执行完成后该容器与网络2就连通了，可以与网络2下的所有容器通信。其原理很简单，**就是docker会给该容器再分配一个网络2网段的ip，相当于容器有2个ip，一个是网络1网段的，另一个是网络2网段的**

* **断开容器与网桥的连接**

****

* **删除网桥**

****

1. **Dockerfile**

用来构建自定义的镜像

* **相关指令：**
* **FROM <**当前镜像是基于哪个镜像，第一个指令必须是FROM**>**
* **RUN<**构建镜像时需要运行的指令**>**
* **EXPOSE<**当前容器对外暴露出的端口号**>**
* **WORKDIR<**指定在创建容器后，终端默认登录进来的工作目录，一个落脚点**>**
* **ENV<**用来在构建镜像过程中设置环境变量**>**
* **ADD<**将宿主机目录下的文件拷贝进镜像且ADD命令会自动处理URL和解压的tar包**>**
* **COPY<**类似ADD，拷贝文件和目录到镜像中，将从构建上下文目录中（原路径）的文件/目录复制到新的一层的镜像内的（目标路径）位置**>**
* **VOLUME<**容器数据卷，用于数据保存和持久化工作**>**
* **CMD<**指定一个容器启动时运行的命令，dockerfile中可以有多个CMD指令，但只有最后一个生效，CMD会被docker run之后的参数替换**>**
* **ENTRYPOINT<**指定一个容器启动时要运行的命令，和CMD类似，都是在指定容器启动程序及其参数**>**

注意：CMD和ENTRYPOINT往往结合使用，ENTRYPOINT写固定部分，CMD动态传参

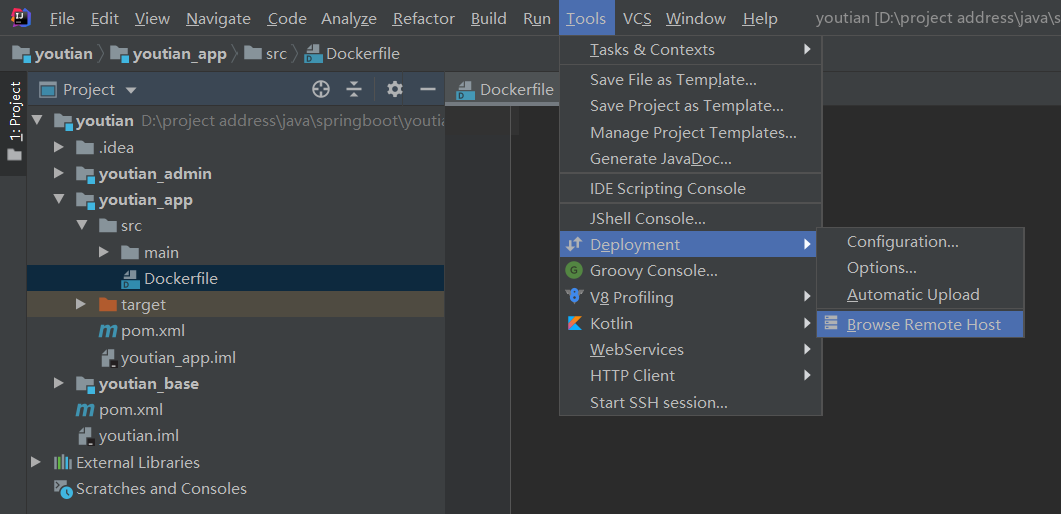
* **相关语法：**

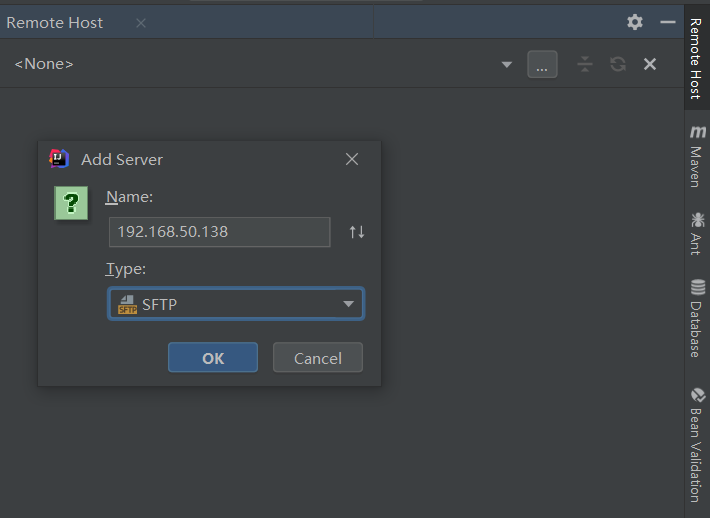
****

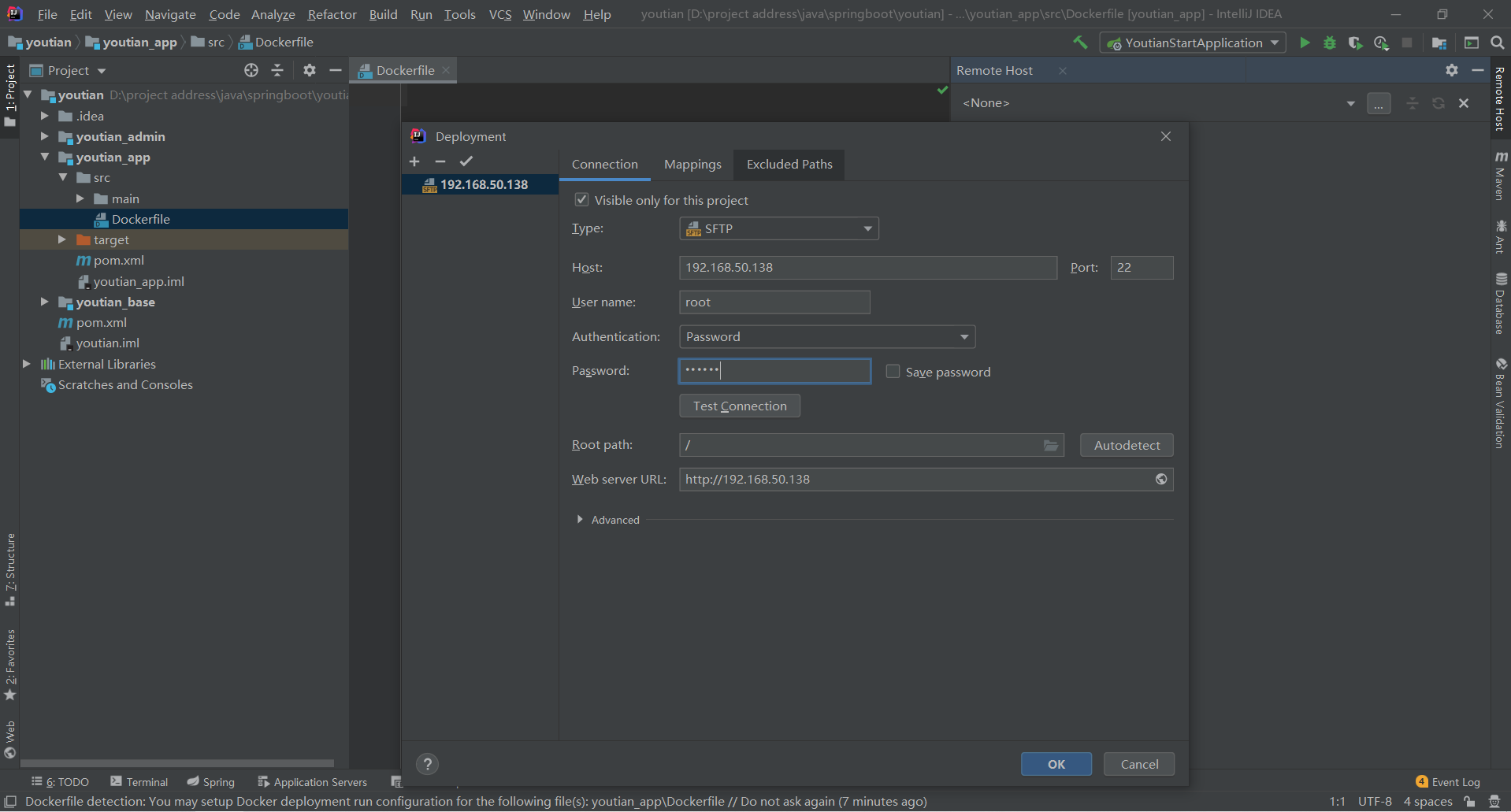
* **进行构建**

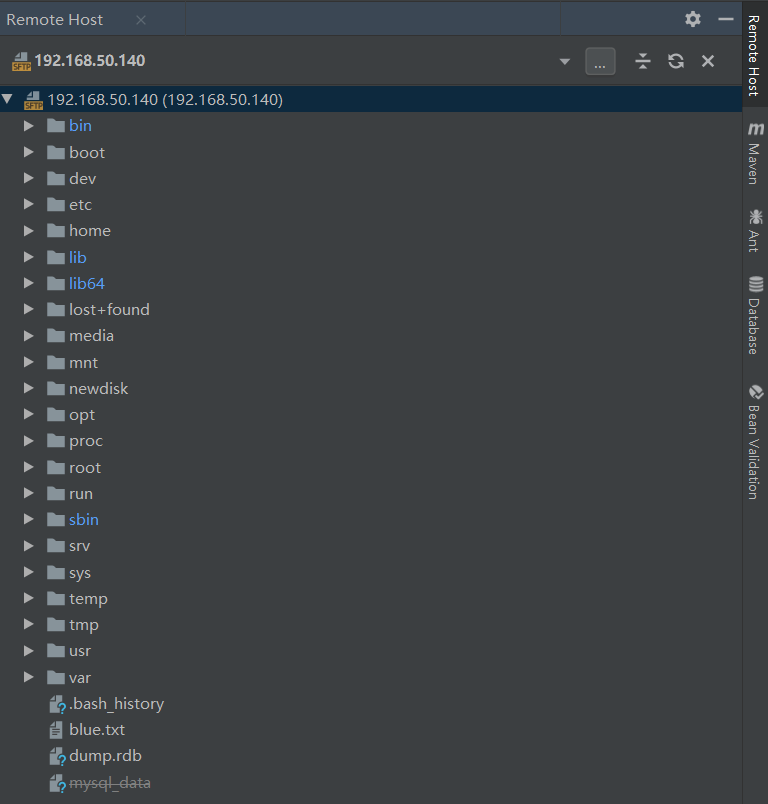
****

1. **Idea中安装Docker插件，远程操作服务器**

****

****

****

****

**十一、docker-compose**

[Docker Compose](https://docs.docker.com/compose/)是一个工具，旨在帮助定义和共享多容器应用程序。使用Compose，我们可以创建一个YAML文件来定义服务，并且使用单个命令，可以启动所有内容或将其全部拆解。

使用 Compose的最大优点是，您可以在文件中定义应用程序堆栈，将其保留在项目存储库的根目录下（现在是受版本控制的），并轻松地让其他人能够为您的项目做出贡献。有人只需要克隆你的存储库并启动撰写应用。

* **安装**

1. **运行以下命令下载 Docker Compose 的当前稳定版本：**

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

1. **将可执行权限应用于二进制文件**

chmod +x /usr/local/bin/docker-compose

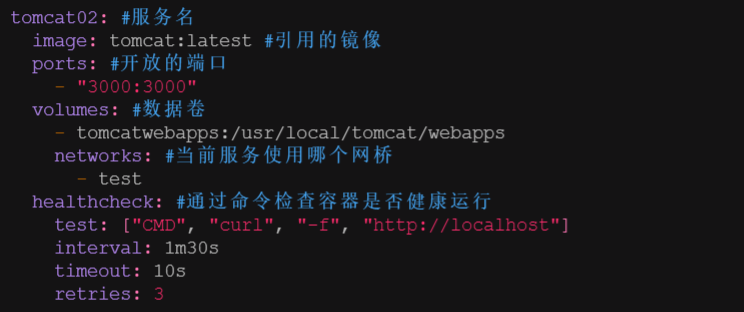
1. **测试安装是否成功**

docker-compose --version

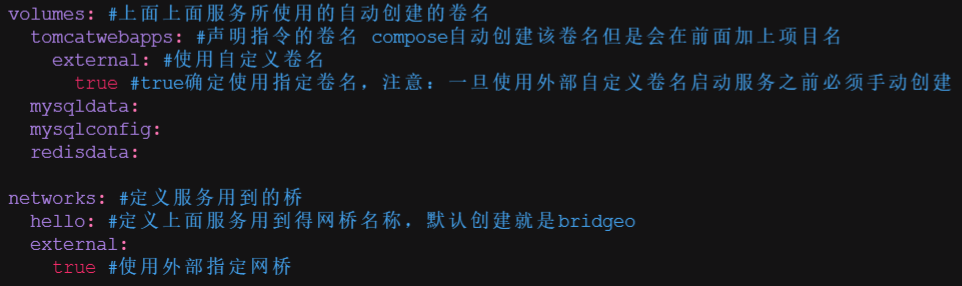
* **模板指令使用案例：**

1. **编写内容**

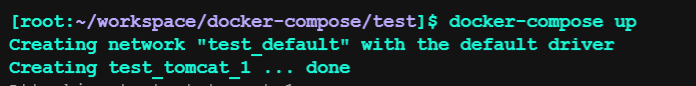
****

****

****

****

1. **运行**



注意：运行此命令必须要保证当前目录下存在docker-compose.yml文件

* **常用指令**

1. **up**

**格式：docker-compose up 【option】**

* **-d：后台启动**

1. **down**

**停止up命令所启动的容器,并移除网络**

1. **exec**

**进入相关容器，类似docker中的exec**

**格式：docker-compose exec 服务名称 bash**

1. **ps**

**列出当前项目所启动的容器（docker-compose.yml所指定的服务）**

1. **restart**

**重启项目中的某个服务，如果不指定，默认重启所有服务（服务名称/ID）**

1. **rm**

**删除项目中的某个服务，如果不指定，默认删除所有**

**格式：docker-compose rm -fv 【服务ID】**

注意：谨慎操作，-v删除服务的数据卷

* **-f：强制删除**
* **-v：删除数据卷**

1. **start**

**启动相关服务，如果不指定，默认启动所有**

1. **stop**

**停止相关服务，如果不指定，默认停止所有**

1. **top**

**查看docker-compose内云的进程**

1. **pause**

**暂停相关服务**

1. **unpause**

**恢复处于暂停状态中的服务**

1. **logs**

**查看相关服务日志，如果不指定，默认查看所有**

**十二、可视化工具（Portainer）使用**

* **安装**

****

* **创建数据卷（默认也会创建）**

****

* **启动**



**以上启动方式也可以通过compose进行管理启动**

**十三、Docker安装相关服务**

* **安装mysql**

1. **拉取镜像**

****

1. **启动mysql**

****

1. **使用数据卷把数据持久化到宿主机**

****

1. **以指定的配置文件进行启动**

****

* **安装tomcat**
* **安装redis**

**三、虚拟机方式项目部署**

* **部署方式一：**
* **配置JAVA环境变量（/etc/profile）**

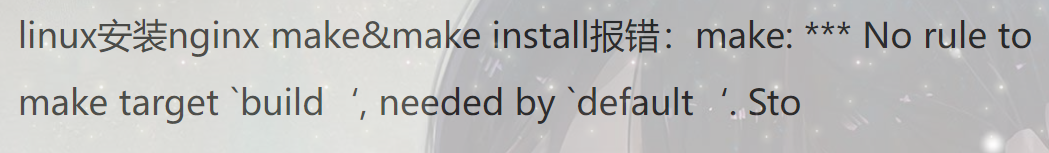
export JAVA\_HOME=/usr/local/java/jdk1.8.0\_291

export CLASS\_PATH=${JAVA\_HOME}/lib:${JAVA\_HOME}/jre/lib

export PATH=${JAVA\_HOME}/bin:$PATH

* **配置nginx**
* **上传nginx压缩文件到服务器并解压**
* **执行**./configure
* **执行make**

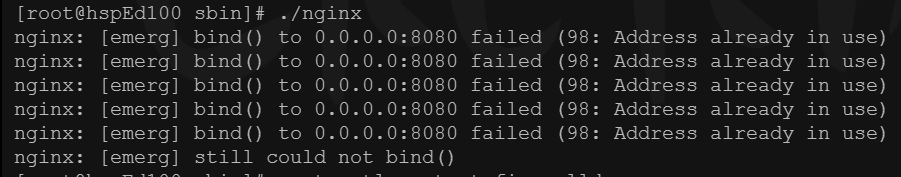
**注意：执行此命令可能会报以下错误**

****

**解决：**

* yum -y install pcre-devel
* yum install -y gcc pcre pcre-devel openssl openssl-devel gd gd-devel
* **执行make install**
* **执行whereis nginx查看nginx所在目录**
* **进入conf目录进行配置**
* **进行sbin目录执行nginx文件进行启动**

**注意：可能会报以下错误**

****

* **重启防火墙**

**systemctl restart firewalld**

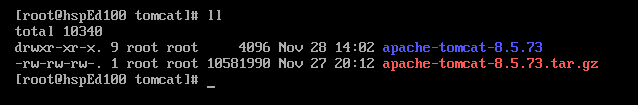
* **查看端口被占用情况：**

**lsof -i:8080**

* **杀死进程**

**kill pid**

* **配置tomcat**
* **解压文件**

****

* **配置环境变量**

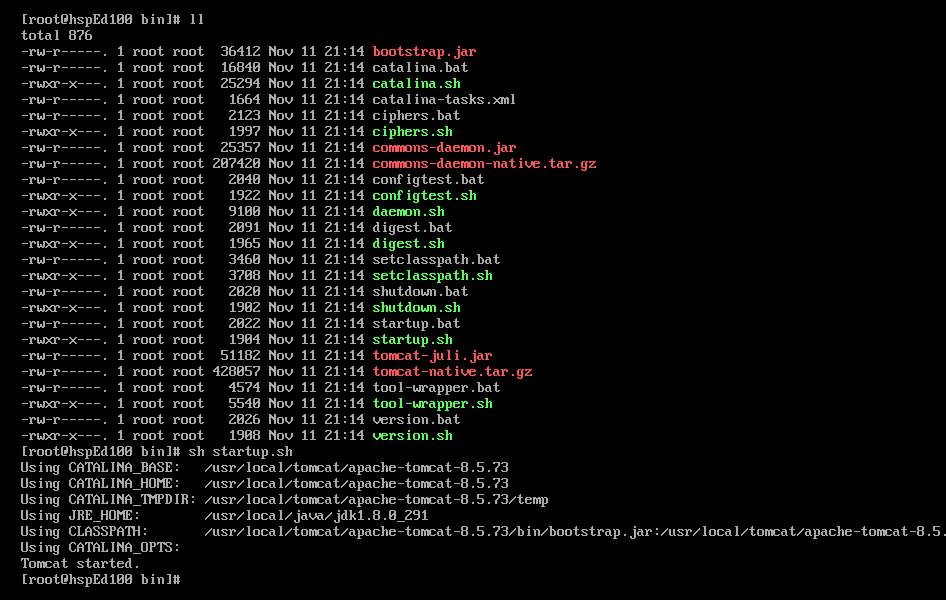
export CATALINA\_HOME=/usr/local/tomcat/apache-tomcat-8.5.73

export PATH=${JAVA\_HOME}/bin:${CATALINA\_HOME}/bin:$PATH

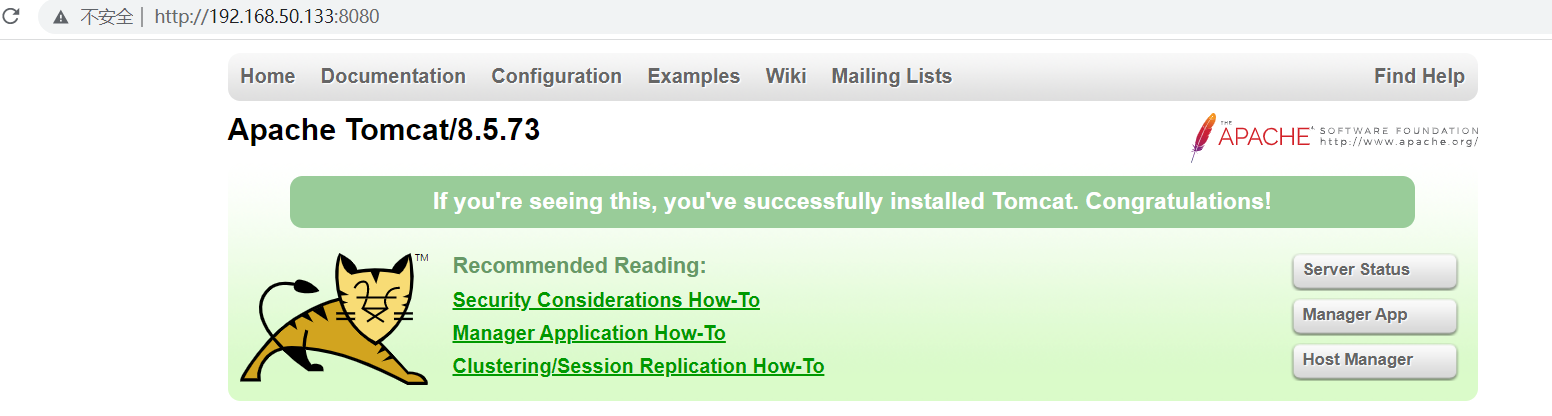
* **刷新环境配置**

****

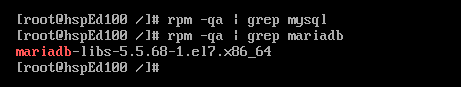
* **启动**

****

* **访问**

****

* **配置mysql**
* **检查linux系统是否已安装mysql或mariadb**

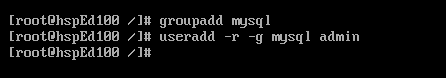
****

* **删除**

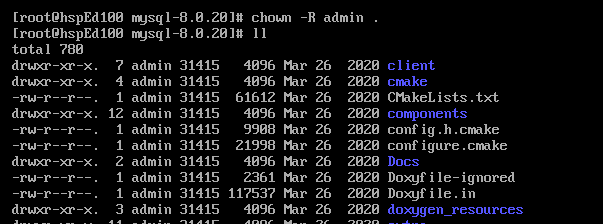
****

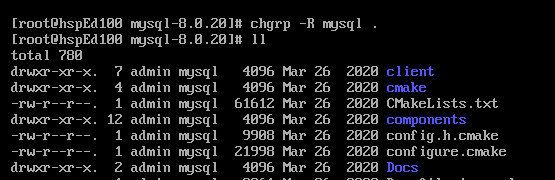
* **解压mysql文件**
* **创建mysql用户组和用户，可以不用创建，这里为了安全起见**

**useradd -r -s /sbin/nologin 用户名（禁止登录操作系统只用于操作mysql）**

****

* **进入mysql解压目录，修改默认（root）所属组和用户**

****

****

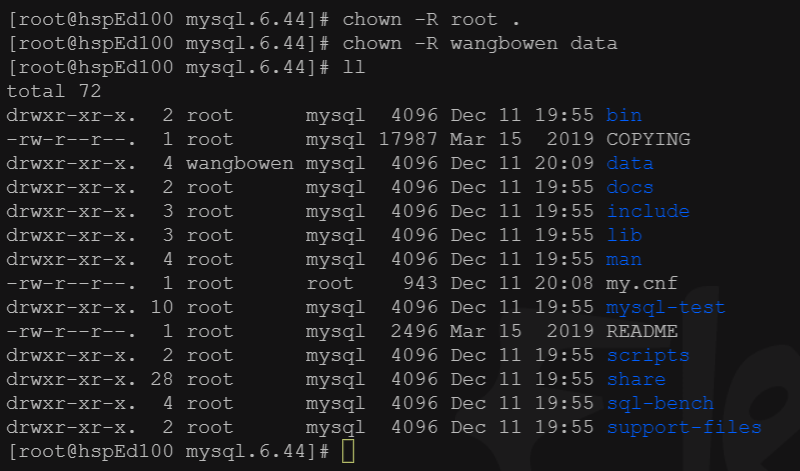
* **执行mysql\_install\_db脚本，对mysql中的data目录进行初始化并创建一些系统表格。注意mysql服务进程mysqlId运行是会访问data目录，所有必须有启动mysqlId的用户（就前面设置的用户）执行这个脚本，或者用root权限执行，可以加上参数 –user=mysql；**

**注意：下载 MySQL 免安装(zip)版，从 5.7 版本以后，不再含有 data 目录，没有 data 目录也就是没有默认数据库，需要用户手动创建。**

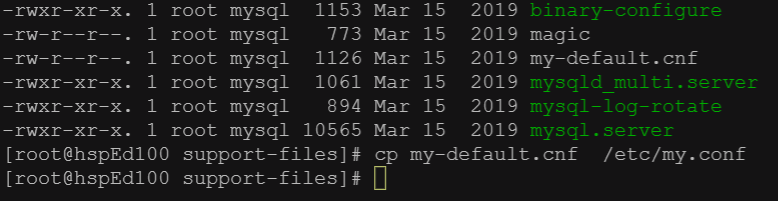
**mysqld --initialize-insecure --user=mysql**或**scripts/mysql\_install\_db –user=admin**

将自动创建默认目录于 home 位置，之后可以通过 my.ini 设置到任意位置

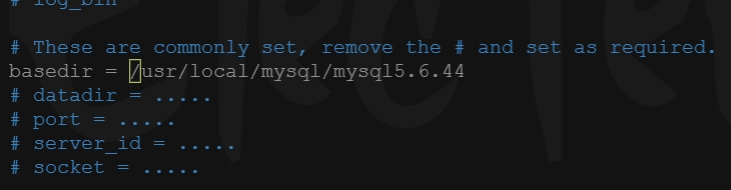
* **将mysql文件除data之外所属用户重新改回root**



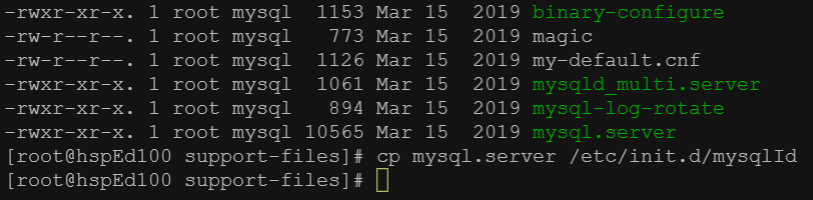
* **将配置文件复制到etc目录下**

****

* **修改etc下的my.conf配置文件basedir位mysql安装路径**

****

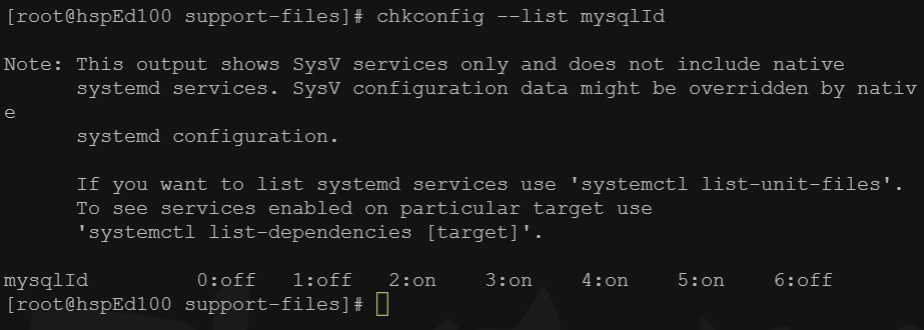
* **将mysql设置为开机既启动；将mysql.server文件复制到etc的init.d目录下，并命名为mysqld**

****

* **通过chkconfig命令将myqlId服务添加到自启动服务项中**

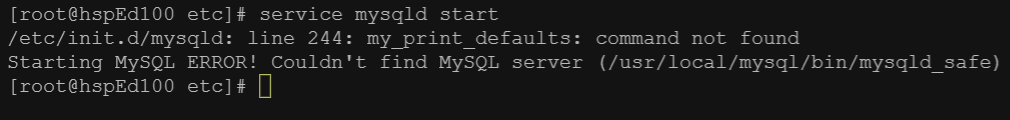
****

* **查看是否添加成功（**chkconfig --list | grep -i mysql**）**

****

**删除（**chkconfig --del mysql**）**

* **启动服务可能会遇到的问题（**因为系统默认会查找/usr/bin下的命令，由于mysql没有在这个目录下，所以出现not found。因此需要做一个软连接到/usr/bin目录下 **）**

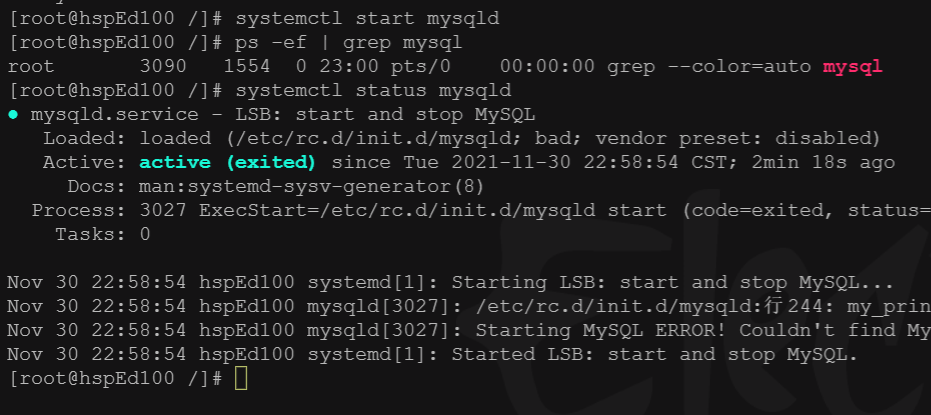
****

* 如果想在linux的任何位置登录mysql的话，就需要创建软链接。 因为我的mysql是安装在/usr/local/mysql/bin/mysql 这个里面的mysql执行文件。但是系统会默认寻找/usr/bin下的执行文件。

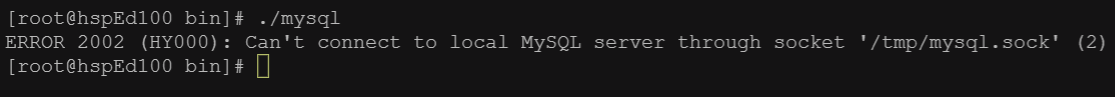
执行命令： ln -s  /usr/local/mysql/bin/mysql  /usr/bin

其中/usr/local/mysql 为：mysql的安装路径

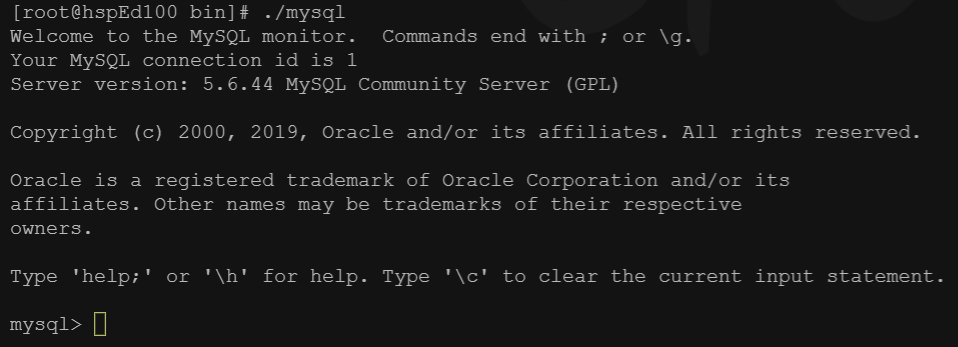
* **启动**mysql



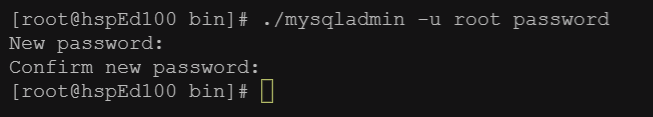
进入mysql遇到遇到问题



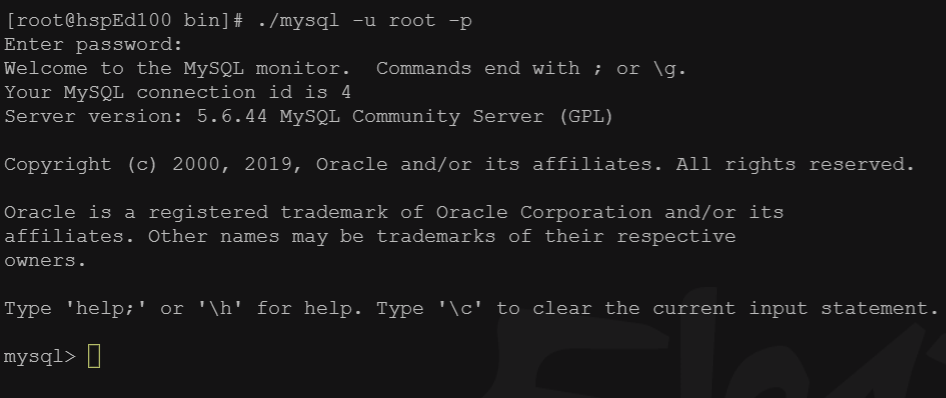
* 输入命令解决：./mysqld\_safe --user=mysql& #



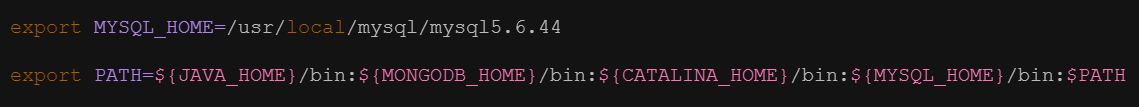
* 设置mysql密码



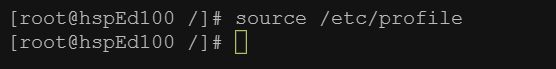
* 输入密码登录



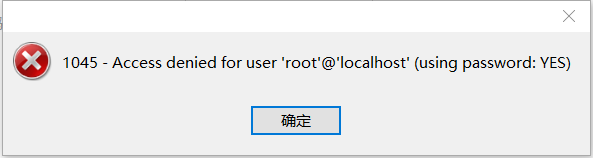
* 配置mysql环境变量

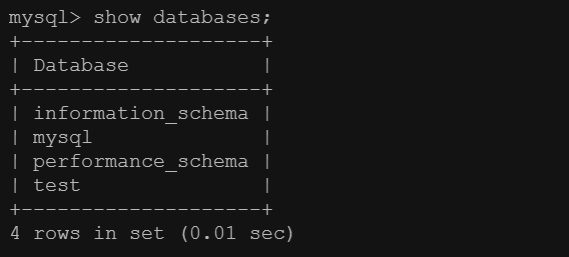


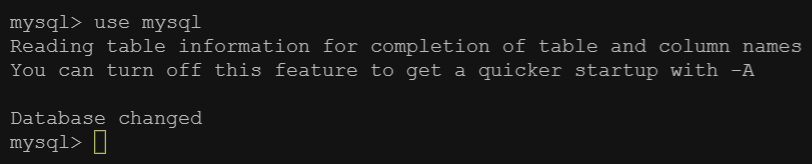
* 刷新配置文件

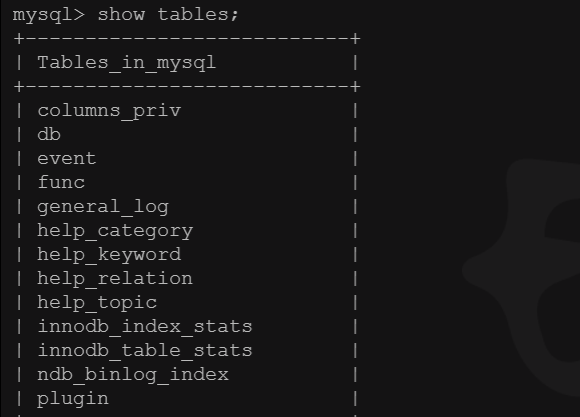
****

* **Netcat连接本地linux的mysql异常并·解决**

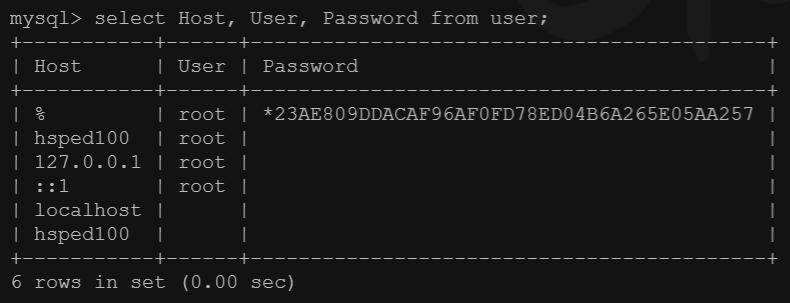
****

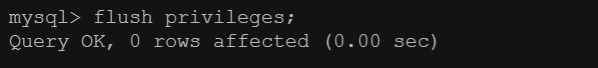
****

****

****

**update user set Host='%' where User='root' and Host='localhost';**

****

****

* **配置redis**

**RPM方式安装Mysql**

* **查看防火墙是否关闭**

systemctl status firewalld

* **关闭防火墙**

systemctl stop firewalld

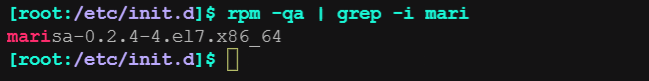
* **关闭linux安全服务**

sesatatus或getenforce

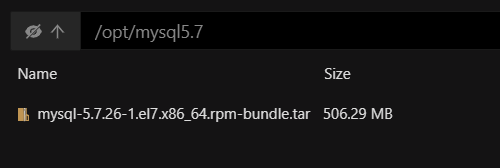
* **查看是否已安装mysql或mariadb**

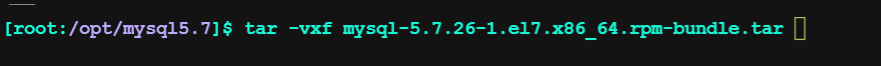
rpm -qa | grep mysql/mariadb

* **卸载mysql或mariadb**

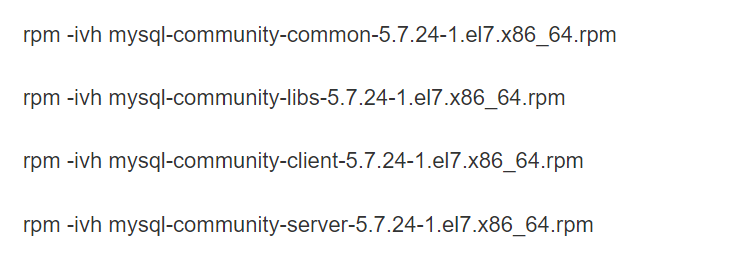
****

* **上传rpm包并解压**

****

****

* **依次执行**

****

* **启动mysql**

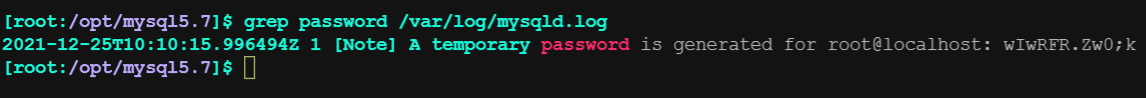
systemctl start mysqld

* **查看mysql启动状态**

systemctl status mysqld

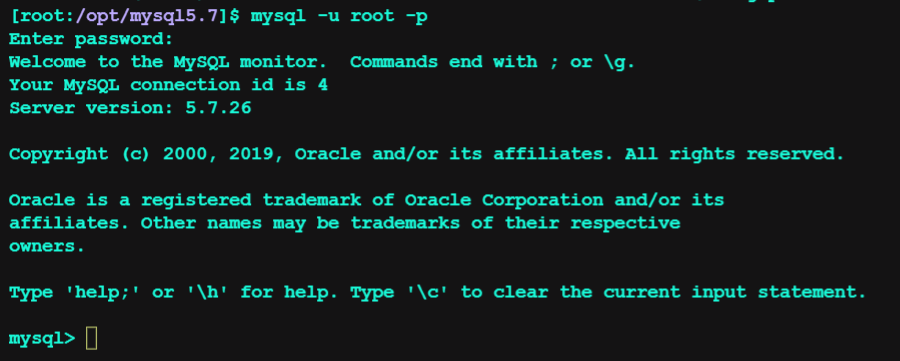
* **查看初始化root密码**

grep password /var/log/mysqld.log



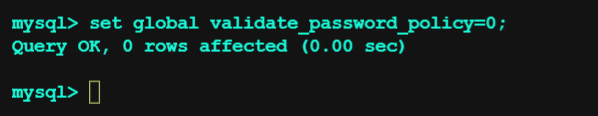
* **登录mysql**

mysql -u root password

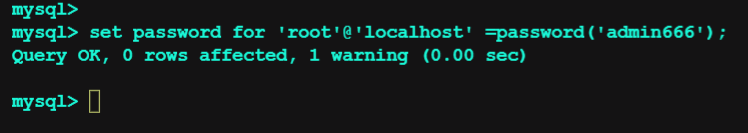
****

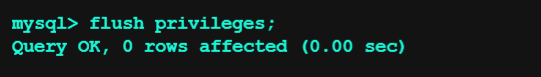
* **设置提升密码设置策略（默认为1）**

set global validate\_password\_policy=0;

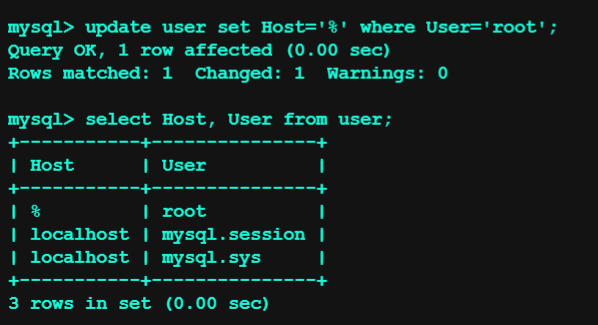
****

* **设置root密码**

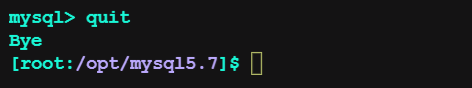
****

****

* **开启远程连接，允许远程连接数据库**

****

* **退出**

****

**Mysql卸载**

* **rpm -qa | grep -i mysql：**查看是否安装mysql
* **chkconfig --list | grep -i mysql：**查询mysql服务
* **chkconfig --del mysqld：**删除mysql服务
* **find / -name mysql：**查询mysql文件
* **rm -rf mysql：**删除mysql相关文件
* **rm -rf my.conf：**删除mysql配置文件
* **groupdel mysql：**删除用户组
* **userdel -r admin：**删除用户
* **rm -rf /etc/init.d/mysqld：**删除mysql开机自启动配置

**安装docker**

**配置镜像加速**

**防火墙配置**

**安全组配置**

**Docker开启mysql**

**Docker开启redis**

**编写DockerFile**