**15-docker**

1. **docker安装**

* 搜索仓库镜像：docker search 镜像名
* 拉取镜像：docker pull 镜像名
* 查看正在运行的容器：docker ps
* 查看所有容器：docker ps -a
* 删除容器：docker rm container\_id
* 查看镜像：docker images
* 删除镜像：docker rmi image\_id
* 启动（停止的）容器：docker start 容器ID
* 停止容器：docker stop 容器ID
* 重启容器：docker restart 容器ID
* 启动（新）容器：docker run -it ubuntu /bin/bash
* 进入容器：docker attach 容器ID或docker exec -it 容器ID /bin/bash，推荐使用后者。

更多的命令可以通过docker help命令来查看。

1.1 **代码仓库地址**

|  |
| --- |
| C++ docker安装 curl -fsSL https://get.docker.com | bash -s docker --mirror Aliyun  docker运行查看 docker ps -a  docker 镜像查看 docker images |

2. **docker 操作命令**

2.1 **docker基础操作**

2.1.1 **docker登录**

|  |
| --- |
| Bash // 官方仓库 docker login // 其他仓库 docker login [artifactory.ep.chehejia.com](http://artifactory.ep.chehejia.com) // [artifactory.ep.chehejia.com](http://artifactory.ep.chehejia.com) 公司本地系统需要修改 sudo chmod 666 /var/run/docker.sock |

2.1.2 **docker拉取镜像**

|  |
| --- |
| Bash vbs 镜像 docker pull artifactory.ep.chehejia.com/lios-docker/lios3-build-env:31 老镜像 docker pull artifactory.ep.chehejia.com/lios-docker/lios3-build-env:4 |

2.1.3 **docker启动， 带挂载目录**

|  |
| --- |
| Bash 直接run老的镜像 docker run -v 宿主机文件:容器文件 -it "镜像ID" /bin/bash |

2.1.4 **docker限制资源启动**

|  |
| --- |
| Bash docker run --privileged --ulimit rtprio=99 --cap-add=sys\_resource -v 宿主机文件:容器文件 -it "镜像ID" /bin/bash |

2.1.5 **docker镜像生成容器(run)**

run是生成容器，只在第一次生成容器使用，即可在run后可以使用，后续第二次使用只需要 start启动，stop停止容器运行

（容器名是一个标签，代替容器ID用于操作）并且启动

|  |
| --- |
| Bash docker run --name "容器名" "镜像ID" docker run -d --name "容器名" "镜像ID" 带 -d： 容器在后台运行，命令行立即返回，你需要使用 docker logs 来查看容器输出。 不带 -d：容器在前台运行，并且容器的输出直接发送到当前命令行界面，命令行会话会被阻塞直到容器停止。 docker run -it "镜像ID" bash 由于我们想通过 bash 进行交互，我们将使其在前台运行并分配一个伪终端（-it 参数）。 |

1. 如果镜像为非阻塞没有终端，run就结束退出，请使用docker run -it "镜像ID" bash，分配一个伪终端，否则后续start开始就退出、exec进不去。
2. 如果镜像为一个一直启动服务，那么不要用上一种方案(分配一个伪终) 端否则服务就被替换起不来。

使用docker run --name "容器名" "镜像ID"， 服务的输出会一直在，这个bash退不出，功能不正常。

使用docker run -d --name "容器名" "镜像ID"，输出在后台，功能正常

|  |  |
| --- | --- |
| -p:端口映射 | 宿主机端口:容器端口 |
| -v:文件挂载 | 宿主机文件:容器文件 |
| -it：-i 和 -t 的组合 | -i 意味着以交互模式运行容器，并保持 STDIN 打开，即使不附加到容器。-t 分配一个伪终端。这两个选项一起使用时允许您与运行的容器进行交互。 |
| --rm | 1. 告诉 Docker 在容器退出时自动删除该容器。这有助于保持您的系统干净，不留下停止状态的容器。 |
| --privileged | 赋予容器完全的权限。这个选项将容器内的用户赋予几乎所有在宿主机上可用的权限。这对于需要执行特权操作的容器（比如直接访问宿主机硬件的容器）是有用的。 |
| -restart always | 不管退出状态码是什么，容器都会被重新启动  除了 always，Docker 还提供了其他几种重启策略：   * no：容器不会在退出时自动重启。这是默认设置。 * on-failure[:max-retry]：仅在容器非正常退出时（退出状态非0）重启容器。可以选择性地附加最大重启尝试次数（max-retry）。 * unless-stopped：容器总是会自动重启，除非它被人工停止。如果在 Docker 重新启动时容器是停止状态，则该容器不会被自动启动。 |

2.1.6 **docker容器运行启动**

|  |
| --- |
| Bash docker start "容器ID" |

2.1.7 **docker进入容器**

|  |
| --- |
| Bash docker exec -it "容器ID" bash |

2.1.8 **docker删除**

|  |
| --- |
| Bash #镜像 docker rmi "镜像ID" #容器 docker rm "容器ID"  docker image rm <image-name>:<tag-name> docker image rm my-image:latest # 删除所有未使用镜像 docker image prune |

2.1.9 **docker查看ip**

|  |
| --- |
| Bash docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' 容器名称或ID |

2.1.10 **文件复制**

|  |
| --- |
| C++ sudo docker cp 主机目录 容器ID:容器目录 |

3. **卡死处理方式**

1、已经运行容器， 不想重新生成容器设置方法

|  |
| --- |
| C++ // 限制容器大小 docker update --memory 10g --memory-swap 12g [CONTAINER ID or NAMES] |

这里将电脑交换空间设置为8G

物理机14G+8G

|  |
| --- |
| C++ #!/bin/bash  swap\_size=8  # 禁用当前的 Swap 文件 echo "禁用当前的 Swap 文件" sudo swapoff /swapfile  # 增加 Swap 文件的大小 echo "加 Swap 文件的大小: ${swap\_size}G" sudo fallocate -l ${swap\_size}G /swapfile #sudo dd if=/dev/zero of=/swapfile bs=1G count=$swap\_size  # 设置 Swap 文件的权限 echo "设置 Swap 文件的权限" sudo chmod 600 /swapfile  # 制作 Swap 空间 echo "制作 Swap 空间" sudo mkswap /swapfile  # 激活 Swap 文件 echo "激活 Swap 文件" sudo swapon /swapfile  # 确认 Swap 文件激活 echo "确认 Swap 文件激活" sudo swapon --show |

将docker 设置限制内存12G+交换空间6G：既可以编译过去，也可不卡死物理机。

4. **状态查看**

docker Stats： 你可以使用 docker stats 命令来实时监控容器的资源使用情况。这将显示包括CPU使用率、内存使用情况、网络IO等信息。

|  |
| --- |
| Bash docker stats [CONTAINER ID or NAMES] |

在内存使用列中，你会看到容器的内存使用量和限制。

docker Inspect： 用 docker inspect 命令可以查看容器的配置详情，包括内存限制。

|  |
| --- |
| Bash docker inspect -f '{{.HostConfig.Memory}}' [CONTAINER ID or NAMES] docker inspect -f '{{.HostConfig.MemorySwap}}' [CONTAINER ID or NAMES] |

这条命令会分别返回设置的内存和交换区限制的字节数。

查看Cgroup信息： Docker容器资源限制背后使用的是Cgroups（control groups）。你可以直接查看cgroup信息来确认限制：

|  |
| --- |
| Bash cat /sys/fs/cgroup/memory/docker/[CONTAINER ID]/memory.limit\_in\_bytes cat /sys/fs/cgroup/memory/docker/[CONTAINER ID]/memory.memsw.limit\_in\_bytes |

上面的命令显示了容器内存限制和内存加swap限制的字节数。确保将[CONTAINER ID]替换为你的容器ID

5. **ssh 远程开启配置**

1. 进入docker
2. 下载openssh

|  |
| --- |
| Bash apt-get update apt-get install openssh-server |

1. 设置root密码

|  |
| --- |
| Bash passwd |

1. 然后设置两遍相同的密码，之后登陆的时候要用到！ 4. 修改配置文件

|  |
| --- |
| Bash vim /etc/ssh/sshd\_config |

1. 添加这一行 PermitRootLogin yes
2. 重启ssh服务

|  |
| --- |
| Bash service ssh restart  sudo service ssh start |

6. **Hubdocker**

https://hub.docker.com/

镜像查看

|  |
| --- |
| Bash docker images |

镜像制作

|  |
| --- |
| Bash docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]] docker commit '容器ID' my\_ubuntu:22.04 |

镜像推送

|  |
| --- |
| Bash docker login [REPOSITORY] docker push my\_ubuntu:22.04  // 登录  docker login // 输入账号密码 // 打远程标签 docker tag my\_ubuntu:22.04 ljk94/ljk\_rep:22.04 // 推送特定标签版本 docker push ljk94/ljk\_rep:22.04  // 登出 docker logout |

docker镜像仓库创建，本地

|  |
| --- |
| Bash # 本地镜像仓库 docker run -d -p 5000:5000 --name registry registry:2 # push docker tag my\_image localhost:5000/my\_image docker push localhost:5000/my\_image # pulldocker pull localhost:5000/my\_image |

如何在 docker hub 分享自己的镜像?看完不踩坑\_dockerhub怎么分享链接-CSDN博客

容器自启动：supervisord

7. **镜像启动服务**

后端启动镜像时启动服务

以sql服务为准

1、官方镜像

|  |
| --- |
| Bash docker pull mysql:8.0 运行容器其中y-secret-pw为root账户密码 docker run --name some-mysql -e MYSQL\_ROOT\_PASSWORD=my-secret-pw -d mysql:tag 查看容器ip docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' 容器名称或ID 登录  mysql -h 172.17.0.5 -P 3306 -u root -p |

2、自己镜像

Dockerfile

|  |
| --- |
| Bash FROM ubuntu:22.04  # 安装 MySQL RUN apt-get update && DEBIAN\_FRONTEND=noninteractive apt-get install -y mysql-server && \  rm -rf /var/lib/apt/lists/\* && \  systemctl enable mysql  # 其他设置...  # 使用自定义入口点脚本启动 MySQL 服务 COPY entrypoint.sh /entrypoint.sh RUN chmod +x /entrypoint.sh ENTRYPOINT ["/entrypoint.sh"] |

entrypoint.sh

|  |
| --- |
| Bash #!/bin/bash  # 启动 MySQL 服务 service mysql start  # 保持容器运行 exec "$@" |

创建新镜像

|  |
| --- |
| Bash docker build -t my-custom-mysql . |

运行新容器

|  |
| --- |
| Bash docker run -d --name my-mysql-container -p 3306:3306 my-custom-mysql |

后续操作 设置远程账户和绑定

查看账户和设置远程账户密码，和容器内sql配置ip绑定

8. **Dockerfile**

在Dockerfile文件目录，运行下面命令 ，就会生成brpc\_image镜像

|  |
| --- |
| Bash docker build -t brpc\_image . |

|  |
| --- |
| Dockerfile # 使用基础镜像 FROM debian:latest  # 准备环境 RUN apt-get update && apt-get install -y --no-install-recommends \  curl \  apt-utils \  openssl \  ca-certificates  # 安装依赖 RUN apt-get update && apt-get install -y --no-install-recommends \  git \  g++ \  make \  libssl-dev \  libgflags-dev \  libprotobuf-dev \  libprotoc-dev \  protobuf-compiler \  libleveldb-dev \  libsnappy-dev \  net-tools && \  apt-get clean -y  # 安装 OpenSSH RUN apt-get update && apt-get install -y --no-install-recommends \  openssh-server && \  apt-get clean -y  # 配置 OpenSSH RUN mkdir /var/run/sshd  # 设置 SSH 的启动脚本 CMD ["/usr/sbin/sshd", "-D"] |

**关键词**

|  |  |
| --- | --- |
| FROM | 基础镜像，必须是可以下载下来的，定制的镜像都是基于 FROM 的镜像 |
| MAINTAINER | 指定镜像的作者信息 |
| RUN | 指定在当前镜像构建过程中要运行的命令  包含两种模式  （1）Shell  RUN (shell模式，这个是最常用的，需要记住)  RUN echo hello  （2）exec模式  RUN “executable”，“param1”，“param2”  RUN [“/bin/bash”,”-c”,”echo hello”]  等价于/bin/bash -c echo hello |
| EXPOSE | 仅仅只是声明端口。  作用：  1、帮助镜像使用者理解这个镜像服务的守护端口，以方便配置映射。  2、在运行时使用随机端口映射时，也就是 docker run -P 时，会自动随机映射 EXPOSE 的端口。  3、可以是一个或者多个端口，也可以指定多个EXPOSE  格式：EXPOSE <端口1> [<端口2>…] |
| CMD | 类似于 RUN 指令，用于运行程序，但二者运行的时间点不同:  1、CMD 在docker run 时运行。  2、RUN 是在 docker build构建镜像时运行的 |
| ENTERYPOINT | 类似于 CMD 指令，但其不会被 docker run 的命令行参数指定的指令所覆盖，而且这些命令行参数会被当作参数送给 ENTRYPOINT 指令指定的程序。  但是, 如果运行 docker run 时使用了 --entrypoint 选项，将覆盖 entrypoint指令指定的程序。  优点：在执行 docker run 的时候可以指定 ENTRYPOINT 运行所需的参数。  注意：如果 Dockerfile 中如果存在多个 ENTRYPOINT 指令，仅最后一个生效。 |
| COPY |  |
| ADD | ADD包含类似tar的解压功能 如果单纯复制文件，dockerfile推荐使用COPY |
| VOLUME | 定义匿名数据卷。在启动容器时忘记挂载数据卷，会自动挂载到匿名卷。 作用： 1、避免重要的数据，因容器重启而丢失，这是非常致命的。 2、避免容器不断变大。 |
| WORKDIR | 指定工作目录。用 WORKDIR 指定的工作目录，会在构建镜像的每一层中都存在。（WORKDIR 指定的工作目录，必须是提前创建好的）。 |
| ENV | 设置环境变量 |
| USER | 用于指定执行后续命令的用户和用户组，这边只是切换后续命令执行的用户（用户和用户组必须提前已经存在） |
| ONBUILD | 用于延迟构建命令的执行 |

https://blog.csdn.net/weixin\_43883625/article/details/128845219?ops\_request\_misc=%257B%2522request%255Fid%2522%253A%2522171645423116800211585128%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request\_id=171645423116800211585128&biz\_id=0&utm\_medium=distribute.pc\_search\_result.none-task-blog-2~all~sobaiduend~default-1-128845219-null-null.142^v100^pc\_search\_result\_base1&utm\_term=Dockerfile%20%E6%96%87%E4%BB%B6%E4%BD%BF%E7%94%A8&spm=1018.2226.3001.4187