目录

[1. Docker简介 2](#_Toc9871784)

[2. Docker的架构 2](#_Toc9871785)

[3. Centos7安装docker 2](#_Toc9871786)

[3.1 卸载老版本的docker 2](#_Toc9871787)

[3.2 安装仓库 2](#_Toc9871788)

[3.2.1 执行以下命令，安装docker所需的包 2](#_Toc9871789)

[3.2.2 执行以下命令，安装stable仓库 2](#_Toc9871790)

[3.2.3 执行以下命令，启用edge及test仓库(可选) 3](#_Toc9871791)

[4. 安装docker ce 3](#_Toc9871792)

[4.1 执行以下命令，更新yum的包索引 3](#_Toc9871793)

[4.2 执行以下命令，安装最新版本的docker ce 3](#_Toc9871794)

[4.3 启动docker 4](#_Toc9871795)

[4.4 验证安装是否正确 4](#_Toc9871796)

[4.5 升级docker 4](#_Toc9871797)

[5. 配置镜像加速器 4](#_Toc9871798)

[6．Docker常用命令 5](#_Toc9871799)

[6.1 docker镜像常用命令 5](#_Toc9871800)

[6.1.1 搜索镜像 5](#_Toc9871801)

[6.1.2 下载镜像 5](#_Toc9871802)

[6.1.3 列出镜像 5](#_Toc9871803)

[6.1.4 删除本地镜像 6](#_Toc9871804)

[6.1.5 保存镜像 6](#_Toc9871805)

[6.1.6 加载镜像 6](#_Toc9871806)

[6.1.7 构建镜像 7](#_Toc9871807)

[6.2 docker容器常用命令 7](#_Toc9871808)

[6.2.1 新建并启动容器 7](#_Toc9871809)

[6.2.2 列出容器 8](#_Toc9871810)

[6.2.3 停止容器 8](#_Toc9871811)

[6.2.4 强制停止容器 8](#_Toc9871812)

[6.2.5 启动已停止的容器 8](#_Toc9871813)

[6.2.6 重启容器 8](#_Toc9871814)

[6.2.7 进入容器 8](#_Toc9871815)

[6.2.8 删除容器 9](#_Toc9871816)

[6.2.9 导出容器 9](#_Toc9871817)

[6.2.10 导入容器 9](#_Toc9871818)

[7. 将微服务运行在docker上 9](#_Toc9871819)

[7.1 使用Dockerfile构建docker镜像 9](#_Toc9871820)

[7.1.1 将项目打成可执行jar(以config-client为例) 9](#_Toc9871821)

[7.1.2 将项目上传到linux 10](#_Toc9871822)

[7.1.3 构建镜像 10](#_Toc9871823)

[7.1.4 启动镜像进行测试 11](#_Toc9871824)

[7.1.5 将config-server运行在docker上 11](#_Toc9871825)

# Docker简介

Docker是一个开源的容器引擎，它有助于更快的交互应用。Docker可将应用程序和基础设施层隔离，并且能将基础设施当做程序一样进行管理。使用docker，可更快的打包、测试及部署应用程序，并可缩短从编写到部署运行代码的周期

# Docker的架构

# Centos7安装docker

## 卸载老版本的docker

sudo yum remove docker

若需删除docker存储的文件，例如镜像、容器等信息，需执行如下命令

sudo rm -rf /var/lib/docker

sudo rm -rf /var/lib/docker-engine

sudo rm -rf /var/lib/dockershim

## 安装仓库

### 3.2.1 执行以下命令，安装docker所需的包

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



其中，yum-utils提供了yum-config-manager工具；device-mapper-persistent-data及lvm2则是devicemapper存储驱动所需的包

### 执行以下命令，安装stable仓库

sudo yum-config-manager \ --add-repo \

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

### 执行以下命令，启用edge及test仓库(可选)

sudo yum-config-manager --enable docker-ce-edge #启用edge仓库

sudo yum-config-manager --enable docker-ce-test #启用test仓库

若再次禁用，可加上—disable标签

sudo yum-config-manager --disable docker-ce-edge #禁用edge仓库

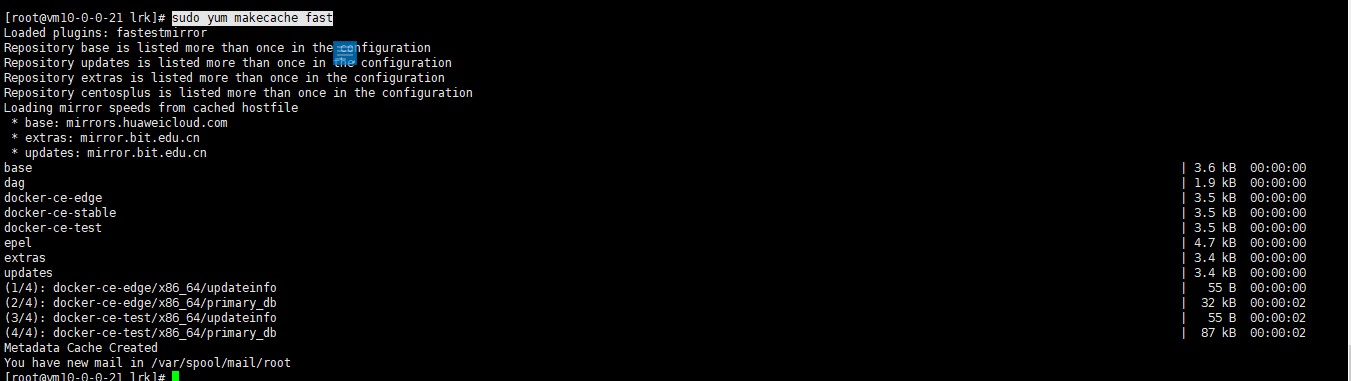
sudo yum-config-manager --disable docker-ce-test #禁用test仓库

备注：edge/test仓库其实也包含在了docker,repo文件中，但默认是禁用的

# 安装docker ce

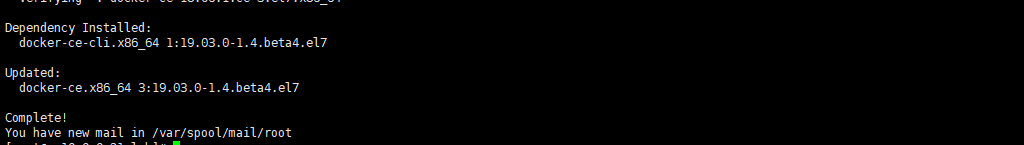
## 4.1 执行以下命令，更新yum的包索引

sudo yum makecache fast



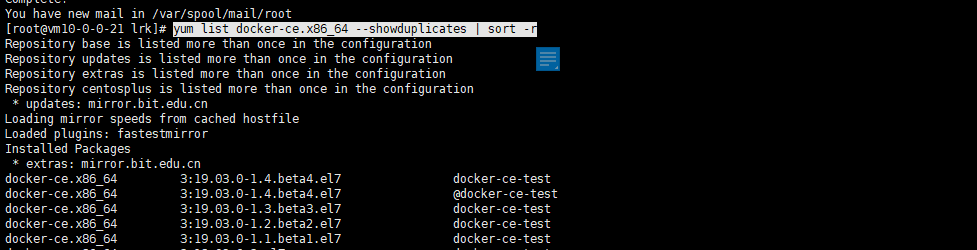
## 4.2 执行以下命令，安装最新版本的docker ce

sudo yum install docker-ce



查看docker当前可用版本

yum list docker-ce.x86\_64 --showduplicates | sort -r



安装指定版本的docker

sudo yum install docker-ce-<version>

## 4.3 启动docker

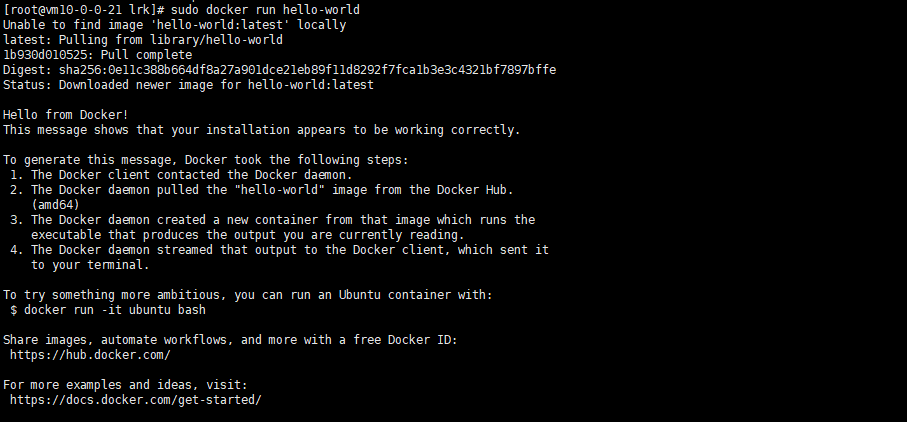
sudo systemctl start docker



## 4.4 验证安装是否正确

sudo docker run hello-world

然后docker将下载测试镜像，并使用该镜像启动一个容器。如果能够看到如下的输出，说明安装成功



## 4.5 升级docker

执行以下命令

sudo yum makecache fast

然后按照安装docker的步骤，即可完成升级

# 配置镜像加速器

国内访问docker hub的速度很不稳定，有时甚至出现连接不上的情况。

常用的docker加速器有阿里云加速器、DaoCloud加速器等。

以aliyun为例：

注册aliyun账号，

# 6．Docker常用命令

## 6.1 docker镜像常用命令

### 6.1.1 搜索镜像

可使用docker search命令搜索存放在docker hub中的镜像

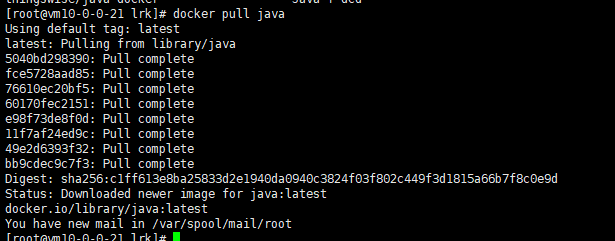
docker search java



### 6.1.2 下载镜像

使用docker pull 命令即可从docker registry上下载镜像

docker pull java 如果下载慢，就配置加速器，参考5

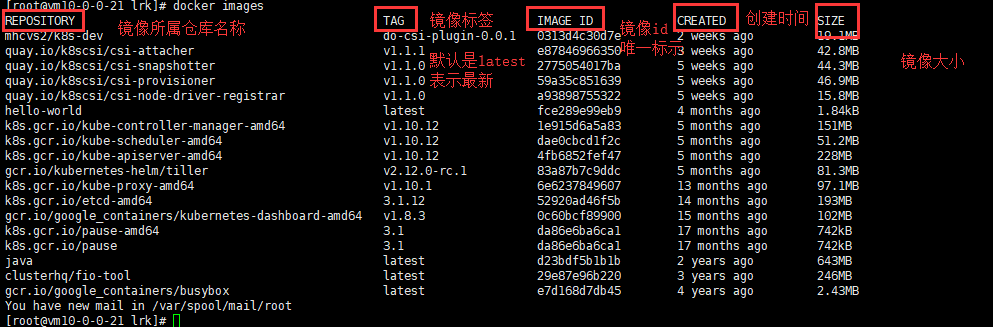


指定想要下载的镜像标签及docker registry地址

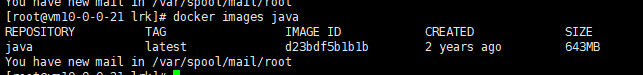
docker pull reg.itmuch.com/java:7

### 6.1.3 列出镜像

docker images

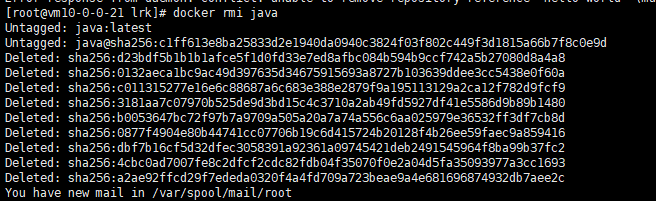


docker images java



### 6.1.4 删除本地镜像

docker rmi hello-world

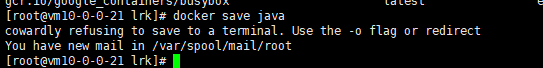


删除所有镜像

docker rmi -f $(docker images)

### 6.1.5 保存镜像

docker sava java



### 6.1.6 加载镜像

docker load

### 6.1.7 构建镜像

docker build

## 6.2 docker容器常用命令

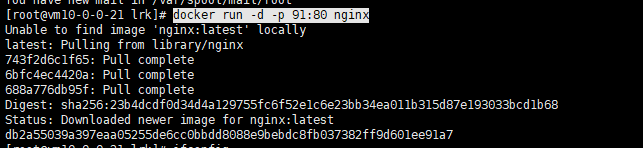
### 6.2.1 新建并启动容器

docker run使用docker可以新建并启动一个容器，使用docker run启动一个容器时，会先检查本地是否存在指定镜像。如果本地不存在该名称的镜像，docker会自动从GitHub上下载镜像并启动一个docker容器

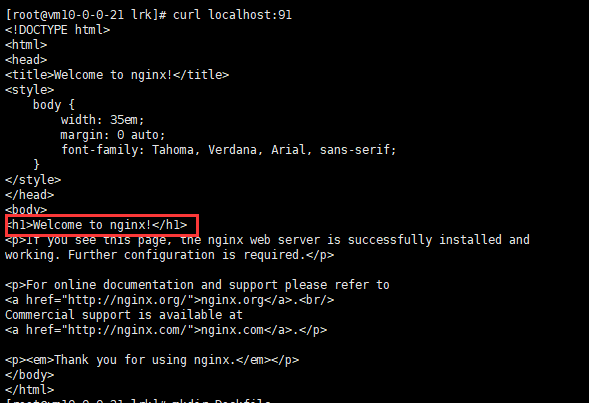
1. docker run java /bin/echo 'Hello World'



1. docker run -d -p 91:80 nginx



访问<http://localhost:91>，可以看到nginx欢迎页面



### 6.2.2 列出容器

docker ps



### 6.2.3 停止容器

docker stop 容器id



### 6.2.4 强制停止容器

docker kill 容器id



### 6.2.5 启动已停止的容器

docker start 容器id



### 6.2.6 重启容器

docker restart



### 6.2.7 进入容器

docker attach 容器id

在很多场景下，使用docker attach命令并不方便，当多个窗口同时attach到同一个容器时，所有窗口都会同步显示。同理，如果某个窗口发生阻塞，其它窗口也无法执行操作

使用nsenter进入容器

nsenter工具包含在util-linux2.23或更高版本中，为了连接到容器，需要找到容器中的第一个进程PID，可通过以下命令获取

docker inspect --format "{{State.Pid}}" 容器id

获得PID后，就可以使用nsenter命令进入容器了

nsenter –target PID –mount –uts –ipc –net –pid

### 6.2.8 删除容器

删除指定容器

docker rm 容器id

删除所有容器

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

### 6.2.9 导出容器

docker export

### 6.2.10 导入容器

docker import

# 将微服务运行在docker上

## 7.1 使用Dockerfile构建docker镜像

Dockerfile是一个文本文件，其中包含了若干条指令，指令描述了构建镜像的细节，并支持以#开头的注释行

### 将项目打成可执行jar(以config-client为例)

进入到config-client目录下，执行mvn clean build

|  |
| --- |
|  |

执行成功后，在target目录下可以看到jar

|  |
| --- |
|  |

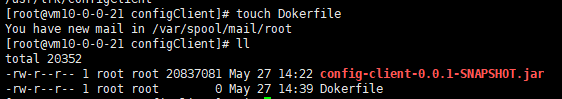
|  |
| --- |
|  |

### 7.1.2 将项目上传到linux

### 7.1.3 构建镜像

在jar包所在路径，构建名为Dokerfile的文件

touch Dokerfile(或者直接使用vim Dokerfile命令)



并添加如下内容

|  |
| --- |
| # 基于哪个镜像  FROM java:8  # 将本地文件挂载到当前容器  VOLUME /tmp  # 复制文件到容器，也可以直接写成ADD config-client-0.0.1-SNAPSHOT.jar /app.jar  ADD config-client-0.0.1-SNAPSHOT.jar app.jar  RUN bash -c 'touch /app.jar'  # 声明需要暴露的端口  EXPOSE 8761  # 配置容器启动后执行的命令  ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"] |

使用docker build命令构建镜像

docker build -t config-repo/config-client-0.0.1 .

格式：docker build -t 仓库名称/镜像名称(:标签) Dockerfile的相对位置

-t 选项指定了镜像的标签，执行该命令，将输出如下结果

|  |
| --- |
|  |

### 7.1.4 启动镜像进行测试

docker run -d -p 8761:8761 config-repo/config-client-0.0.1

访问http://ip:8761/ datasourcePassword

可使用相同的方式，将其他微服务也构建成Docker镜像

### 7.1.5 将config-server运行在docker上

（1）打包并上传

|  |
| --- |
|  |
|  |

（2）构建镜像

|  |
| --- |
|  |
| # 基于哪个镜像  FROM java:8  # 将本地文件挂载到当前容器  VOLUME /tmp  # 复制文件到容器，也可以直接写成ADD config-server-0.0.1-SNAPSHOT.jar /app.jar  ADD config-server-0.0.1-SNAPSHOT.jar app.jar  RUN bash -c 'touch /app.jar'  # 声明需要暴露的端口  EXPOSE 8762  # 配置容器启动后执行的命令  ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"] |
|  |

1. 启动镜像

|  |
| --- |
|  |

1. 测试

# 管理docker镜像

## 使用docker hub管理镜像

### 8.1.2 注册与登录(<https://hub.docker.com> )

账号：liurunkai 密码：lrk291988731

|  |
| --- |
|  |

也可以使用命令docker login进行登录

|  |
| --- |
|  |

### 8.1.2 创建仓库

|  |  |
| --- | --- |
|  |  |

点击create进行创建

|  |
| --- |
|  |

### 8.1.3 推送镜像

|  |
| --- |
|  |

docker push docker push liurunkai/config-client:0.0.1

|  |
| --- |
|  |

查看已推送的镜像

|  |
| --- |
|  |
|  |

## 使用私有仓库管理镜像

### 搭建私有仓库

docker run -d -p 5000:5000 --restart=always --name registry2 registry:2

|  |
| --- |
|  |

## 将镜像推送到私有仓库

# 使用maven插件构建docker镜像

## 添加maven的docker插件(以docker-server为例)

|  |
| --- |
|  |

## 执行以下命令，构建Docker镜像

mvn clean package docker:build

会报错：connected refused localhost:2375

解决方式：（1）传到服务器上去执行

（2）在本地安装docker for windows

右键docker小图标，点击settings

|  |
| --- |
|  |

重新执行上述命令，会看到如下输出，说明成功

|  |
| --- |
|  |

查看镜像：docker images

|  |
| --- |
|  |

## 启动镜像

docker run -d -p 8080:8080 liurunkai/config-server:0.0.1

|  |
| --- |
|  |

## 访问测试

<http://localhost:8080/application-test.yml>

|  |
| --- |
|  |

可以成功访问，说明启动成功

# 插件读取Dockerfile进行构建

## 以config-client为例，main/docker新建Dockerfile

|  |
| --- |
| **FROM** java:8 **VOLUME /**tmp **ADD** config-client-0.0.1**-**SNAPSHOT.jar app.jar **RUN** bash **-**c **'touch /app.jar' EXPOSE** 9000 **ENTRYPOINT** [**"java"**,**"-Djava.security.egd=file:dev/./urandom"**,**"-jar"**,**"/app.jar"**] |
|  |

## 修改pom.xml

|  |
| --- |
| <**plugin**>  <**groupId**>com.spotify</**groupId**>  <**artifactId**>docker-maven-plugin</**artifactId**>  <**version**>0.4.12</**version**>  <**configuration**>  <**imageName**>liurunkai/config-client:0.0.1</**imageName**>  <**dockerDirectory**>${project.basedir}/src/main/docker</**dockerDirectory**>  <**resources**>  <**resource**>  <**targetPath**>/</**targetPath**>  <**directory**>${project.build.directory}</**directory**>  <**include**>${project.build.finalName}.jar</**include**>  </**resource**>  </**resources**>  </**configuration**> </**plugin**> |

这个pom.xml没有指定baseImage和entryPoint，而是使用dockerDirectory指定Dockerfile所在的路径，这样就可以使用Dockerfile构建Docker镜像了

## 构建Docker镜像

方式一：跟之前一样，执行如下命令

docker clean package docker:build -X test

|  |
| --- |
|  |

方式二：将插件绑定在某个phase执行

修改pom.xml: 添加如下配置

|  |
| --- |
| <**executions**>  <**execution**>  <**id**>build-image</**id**>  <**phase**>package</**phase**>  <**goals**>  <**goal**>build</**goal**>  </**goals**>  </**execution**> </**executions**> |
|  |

只需执行mvn package，就会自动执行mvn package docker:build

|  |
| --- |
|  |

## 推送镜像(使用maven插件)

### 修改maven的全局配置文件settings.xml

|  |
| --- |
| <server>  <id>docker-hub</id>  <username>liurunkai</username>  <password>lrk291988731</password>  <configuration>  <email>17600272806@163.com</email>  </configuration>  </server> |

### 修改pom.xml

|  |
| --- |
| <**serverId**>docker-hub</**serverId**> |
|  |

### 推送镜像

mvn clean package docker:build -DpushImage

其中pushImage表示推送镜像

## 10.5 启动并测试

docker run -d -p 8081:8081 liurunkai/config-client:0.0.1

|  |
| --- |
|  |