Docker介绍

1）Docker是什么

> Docker是一个用于开发、交付和运行应用的开放平台。它允许使用者从基础设备中分离应用程序，以此达到快速交付软件。通过Docker快速交付、测试、和部署代码的方法，使用Docker能够显著减少开发代码与在生产环境上运行的延迟。

2）Docker平台

> Docker提供了在一个宽松隔离的环境（容器Container）里面打包与运行一个应用程序的能力。这种隔离与安全机制允许使用者在一个主机上同时允许多个容器。容器是轻量级的，且包含运行应用需要的所有东西，因此使用者不需要考虑当前主机上安装了什么。当你工作时可以轻松分享你的容器，并确保其他人也是用相同的方式使用这些容器。

- Docker提供以下工具和平台用于管理容器的生命周期
- 1、通过容器开发应用程序及其支持组件
- 2、容器成为分发和测试应用程序的单元
- 3、当你准备好部署你的应用程序到生产环境时，作为容器或者编排好的服务。无论你的生产环境是本地数据中心、云服务提供者、或者两者混合，它都能以相同的方式工作。

3）通过Docker能做什么

1）快速、一致的应用交付

> 通过允许开发者使用一个提供应用程序和服务的本地容器在标准环境中工作，Docker简化了开发的生命周期。容器非常适合于持续性的集成（CI）和持续性的交付（CD）的工作流。如以下情形：
>
> - 开发者在本地编写代码，并通过docker容器分享代码给同事
> - 然后使用docker推送应用进入测试环境，进行自动和手动测试
> - 当开发者发现bug时，他们可以在开发环境中解决bug并重新部署至测试环境进行测试和验证
> - 当测试完成后，向客户提供解决方法就像上传镜像到生产环境一样简单

2）响应式部署和扩展

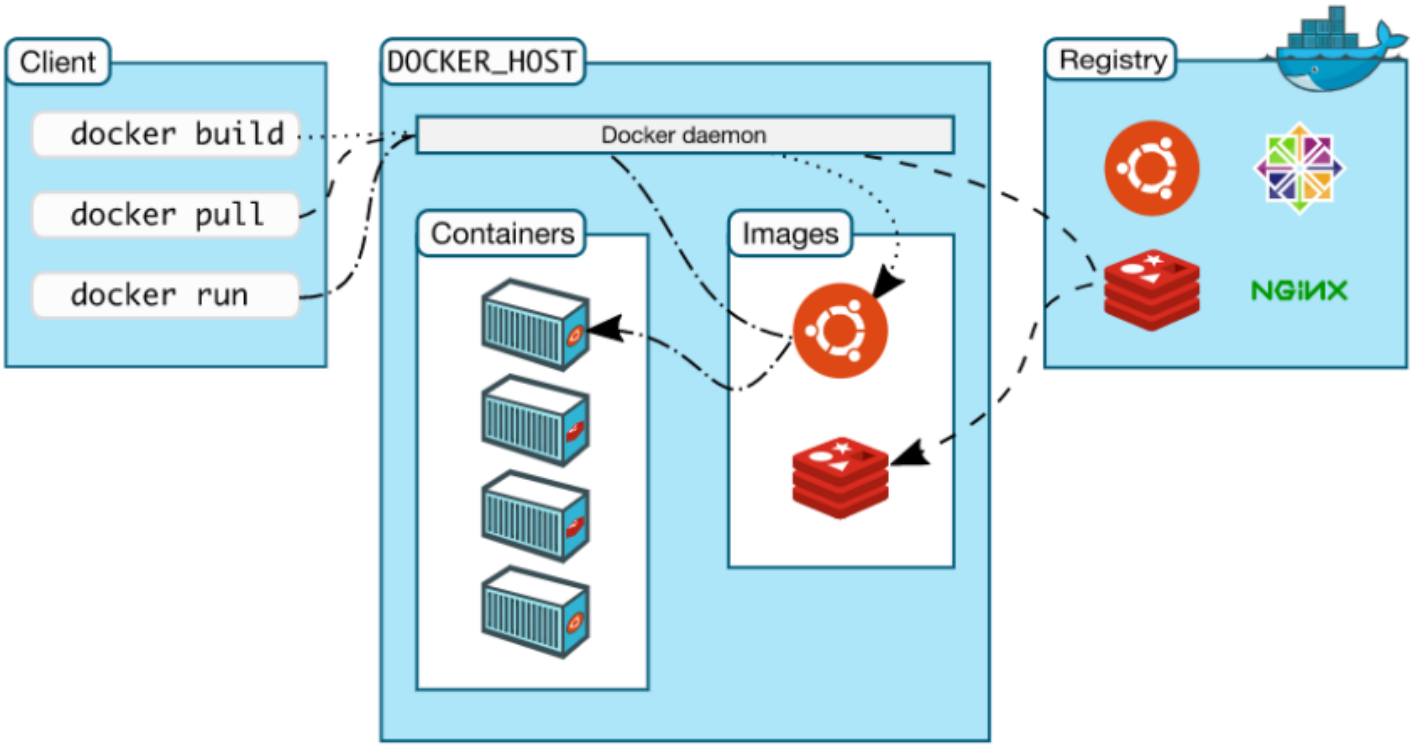
> Docker的基于容器的平台允许高可移植性的工作负载，它可以运行在本地电脑、物理或虚拟机等环境。Docker的可移植性和轻量级特性使得动态管理工作负载变成容易，根据业务的需求做到几

> 乎实时的扩展和卸载应用和服务。

3）在相同硬件上允许更多工作负载

> Docker是轻量级且快的。它为基于管理程序的虚拟机提供一种可行的、经济有效的替代方案，因此可以使用计算能力取实现业务目标。Docker非常适合高密度和需要更少的资源做更多的中小级部署。

4）Docker架构



> Docker使用客户-服务器架构。客户端与Docker守护进程进行交流，后者负责构建、允许和分发Docker容器。客户端可以与本地或者远程的守护进程进行连接，通过UNIX套接字和网络接口进行REST API交流。另一种Docker客户端是Docker Compose，它允许工作一个由多个容器组成的应用程序。

- Docker Daemon（Docker守护进程）

> 监听Docker API请求与管理Docker 对象如镜像、容器、网络和容量等。可以与其他守护进程进行交流从而管理Docker 服务

- Docker Client（Docker 客户端）

> Docker使用者与Docker交互的主要方式，通过使用命令来使用Docker API。客户端可以与多个守护进程进行交流

- Docker Registries（Docker 注册）

> 用于存储Docker镜像。Docker Hub是一个公共的注册中心，所有人都可以使用，默认Docker被配置在Docker Hub上寻找镜像。当然，也可以使用自己的私人注册中心。当使用Docker pull或push

> 命令时，就会从配置的注册中心上拉取或推送镜像

- images镜像

> 一个镜像是一个只读模板，包括创建Docker 容器的说明。通常，镜像是基于另一个镜像并加上客制化与特定配置的。要构建自己的镜像需要创建一个Docker文件，每个命令都会在镜像中生成一层，当重建镜像时只有修改了的层才需要重建，这也是它轻量级且快的原因

- Container容器

> 容器是镜像的一个可执行化实例，可通过Docker API或CLI（命令行接口）来创建、启动、停止、删除容器。一个容器可以和多个网络连接，并附加存储空间，甚至基于当前状态新建一个镜像。
>
> 通常容器之间是相对隔离的，可以控制它们之间、网络、存储空间、子系统等等的隔离程度。

Docker安装与使用

1）安装（CentOS上）

```
# 安装yum-utils
yum install -y yum-utils
# 通过yum-config-manager安装
yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
# 安装Docker-Engine(默认最新稳定版)
yum install -y docker-ce docker-ce-cli containerd.io
# 安装特定版本的Docker-Engine
yum list docker-ce --showduplicates | sort -r
yum install -y docker-ce-20.10.7 docker-ce-cli-20.10.7 containerd.io
```

2）开启Docker与验证

```
# 开启Docker
systemctl start docker
# 验证Docker-Engine是否安装成功，允许hello-world镜像（官方验证镜像）
docker run hello-world
```

3）添加Docker用户组

> 由于Docker的守护进程绑定在Unix套接字，而Linux默认只有root可以使用，因此其他用户如果不希望使用命令时加上sudo，可以建立一个Unix组名为docker，把用户加进去（需注意这种方式会给予其他用户相当于root的权利）。

```
# 添加用户组
groupadd docker
# 添加用户到附加用户组
usermod -aG docker $USER
# 之后重新登陆使生效，虚拟机需重启；或使用下面命令
newgrp docker
# 其他用户进行验证
docker run hello-world
# 若在添加用户前已使用过sudo执行Docker命令，那么用户操作可能没权限
WARNING: Error loading config file: /home/user/.docker/config.json -
stat /home/user/.docker/config.json: permission denied
# 为了处理这种情况，可以删除.docker文件夹(但会使一些用户设置被删除)
# 或者使用命令修改所有权以及权限
chown "$USER":"$USER" /home/"$USER"/.docker -R
chmod g+rwx "$HOME/.docker" -R
```

> 也可以选择在无root权限下运行Docker

4）Docker的系统设置

- 系统启动时启动Docker

```
systemctl enable docker.service
systemctl enable containerd.service
```

- 日志驱动

> Docker默认使用json-file的日志驱动去存储主机下所有容器产生的日志，随着时间，文件尺寸扩大，有潜在的耗尽磁盘资源的可能。
>
> 为了解决这个问题，可以配置json-file日志驱动使用日志切分log rotate，如local日志驱动默认切分日志，或者使用一种日志驱动将日志发生给远程日志聚合器。

- 配置Docker监听tcp端口

> 需注意这样会使远程机器可以直接访问到Docker守护进程

```
# 修改Docker的service文件
systemctl edit docker.service
# 加入以下句子
[Service]
ExecStart=
ExecStart=/usr/bin/dockerd -H fd:// -H tcp://127.0.0.1:2375
# 重载systemctl配置，重启Docker
systemctl daemon-reload
systemctl restart docker.service
# 验证
netstat -lntp | grep dockerd
```

```
# 新建daemon.json（另一种方式，同时使用会启动Docker失败）
vim /etc/docker/daemon.json
# 加入语句，之后重启docker
{
        "hosts": ["unix:///var/run/docker.sock", "tcp://127.0.0.1:2375"]
}
```

5）Docker的使用

1. 项目下新建Dockerfile文件

```
FROM node:12-alpine
# 使用国内镜像
RUN sed -i 's/dl-cdn.alpinelinux.org/mirrors.ustc.edu.cn/g' /etc/apk/repositories
run mkdir -p /usr/src/app
WORKDIR /usr/src/app
COPY . /usr/src/app
RUN npm install
ENTRYPOINT node index.js
```

2. 根据Dockerfile创建镜像文件

```
docker build -t myapp/myweb:v1.0.0 .
```

3. 启动镜像

```
docker run -d -p 80:8080 myapp/myweb:v1.0.0
# -d 表示运行在分离模式，即后台
# -p 80:80 表示映射 主机端口：容器端口
# 最后的表示使用的镜像名
```

4. 修改文件后重新生成镜像（此时Dockerfile中未改变的操作只会使用缓存，因此生成速度极快）

```
docker build -t myapp/myweb:v1.0.0 .
```

5. 此时旧的容器还在运行，需要进行移除出docker

```
docker ps
docker stop <container-id>
docker rm <container-id>
```

6. 通过docker hub分享镜像

```
# 登录自己的Docker Hub账号
docker login -u onlyzuri
# 标准上传命令，其中onlyzuri为用户名，myapp为仓库名
docker push onlyzuri/myapp:tagname
# 为了使用这种名字，需要修改本地镜像的名字
docker tag myapp/myweb:v1.0.0 onlyzuri/myapp:v1.0.0
# 推送，冒号后面为版本号，未定义的话，hub会使用默认的tag，latest
# 推送可使用国内镜像
docker push onlyzuri/myapp:v1.0.0
# 拉取
docker run -dp 80:3000 onlyzuri/myapp:v1.0.0
```

6）数据持久化

> 当容器允许时，它使用镜像中的多层作为自己的文件系统。每一个容器都有自己的暂存空间去增加、修改、删除文件，不为其他容器所看见，因此即使同一镜像，第一个容器中做的改变，第二个容器是看不见的。
>
> 而使用Volumes卷，容器可以将其特定文件系统连接回主机。容器中挂载的某个目录，其改变也会同步显示在主机上。若将目录同样挂载在其他容器上，则可以看到其中的改变。

- 使用命名卷实现，named volumes

```
# 创建命名卷
docker volume create todo-db
# 假设todo list的数据文件放在/etc/todos/todo.db上，挂载此目录
# 此后通过该volume启动的容器都能看到一样的文件变化
docker run -dp 80:3000 -v todo-db:/etc/todos onlyzuri/myapp:v1.0.0
# 查看volume实际存储位置
docker volumn inspect todo-db
```

7）绑定挂载

> 若每次进行应用程序的代码修改都要重新组建镜像，重新运行后才能看到修改效果，是一件很麻烦的事。而通过绑定挂载则可以很快看到容器内的源代码改变，且很快看出修改的效果，等待测试完成后再进行镜像的重建。

```
# 如基于node的文件有自带的nodemon用于检测和重启，目前挂载/home/myapp
docker run -dp 80:3000 -w /home/myapp -v "$(pwd):/home/myapp" node:12-alpine sh -c "yarn instal
# 查看docker日志
docker logs -f <container_id>
# 测试完成后重建镜像
docker build -t onlyzuri/myapp:v1.0.0 .
```

8）多容器APP的运行

> 出于多种原因，如将来可能分离前后端、数据库等，以及分离容器可以使本地版本和更新版本隔离，不希望发布数据库引擎等等原因，一般会让单个容器只做一件事。而为了让多容器之间相互交流协作，则需要将他们放于一个网络内。

```
# 创建docker网络
docker network create todo-app
# 执行一个MySQL容器，并把它添加进这个网络
docker run -d --network todo-app --network-alias mysql -v todo-mysql-data:/var/lib/mysql -e MYS
# 使用exec命令在已启动的容器里面运行一条命令，i表示interact，t表示terminal
# 登录后可以看到已通过容器启动了MySQL，并且有一个todos的数据库
docker exec -it <mysql-container-id> mysql -u root -p
# 使用nicolaka/netshoot容器可以进行故障排除和调试网络，是一个检测容器的工具
docker run -it --network todo-app nicolaka/netshoot
# 在nicolaka/netshoot容器内使用dig查看网络内成员的ip地址
dig mysql
# 通过配置连接MySQL，启动todo镜像(注意通过参数建立连接的方式不适用于生产环境)
docker run -dp 80:3000 -w /home/myapp -v "$(pwd):/home/myapp" --network todo-app -e MYSQL_HOST=
```

9）Docker compose

> docker compose是一个被开发于帮助定义与分享多容器app的一个工具，通过创建一个YAML文件去定义服务，通过一个简单命令，将它启动或停止。
>
> docker compose的一个较大的好处是可以将程序的栈定义在一个文件中，并将它放在项目的根目录下，方便其他人克隆与执行这个compose app。

- 下载docker compose

```
# 下载
curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-
# 赋予执行权限
chmod +x /usr/local/bin/docker-compose
# 快链接
ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
# 测试效果
docker-compose version
```

- 在应用的根目录下创建docker-compose.yml文件

```
# compose架构版本
version: "3.8"
services:
# services里面的容器均视为同一个网络，可直接访问
    app:
        image: node:12-alpine
        command: sh -c "yarn install && yarn run dev"
        ports:
            - 80:3000
        working_dir: /home/myapp
        # 挂载目录到容器中
        volumes:
            - ./:/app
        environment:
            MYSQL_HOST: mysql
            MYSQL_USER: root
            MYSQL_PASSWORD: secret
            MYSQL_DB: todos
    mysql:
        image: mysql:5.7
        # compose不能自动创建volumes，需要显示表示
        volumes:
            - todo-mysql-data:/var/lib/mysql
        environment:
            MYSQL_ROOT_PASSWORD: secret
            MYSQL_DATABASE: todos

volumes:
    todo-mysql-data:
```

- 启动compose app

```
# -d表示后台启动
docker-compose up -d
# 查看日志，-f表示following，后面可接具体service名
docker-compose logs -f [service name]
```

- 关闭compose app

```
# 一般来说关闭compose并不会删除命名卷named volumes，除非加上 --volumes
docker-compose down
```

10）镜像建立安全扫描

```
# 使用docke scan扫描镜像，会使用内置的Snyk扫描漏洞
docker scan onlyzuri/myapp:v1.0.0
```

11）查看镜像建立的每个过程

```
docker image history onlyzuri/myapp:v1.0.0
```

- 镜像中的某一层发生了改变，其他下楼层全部需要重建，无法使用缓存

```
# 修改Dockerfile中的顺序，使得经常改动重建的层次放在下面
FROM node:12-alpine
WORKDIR /home/myapp
# 基于node的应用依赖被定义在package.json文件中，一般它不改变，则需要的依赖基本不变
COPY package.json yarn.lock ./
RUN sed -i 's/dl-cdn.alpinelinux.org/mirrors.ustc.edu.cn/g' /etc/apk/repositories
RUN apk add --no-cache python g++ make
RUN yarn install --production
copy . /app
CMD ["node", "src/index.js"]
# 创建.dockerignore文件，把node_modules写在里面，则第二次copy会忽略它
# 里面基本上是依赖，如果每次重建都重新下载，将会很费时间
# 此时修改src里面的内容，会发现上面几层全部使用的缓存，只有从copy . /app开始重建
```

12）多阶段构建Multi-stage builds

> 使用多阶段构建可以分离建立时依赖和运行时依赖，减小镜像体积
>
> 例如基于Java的应用程序需要JDK来运行代码，或者需要Maven和tomcat帮助建立app，但这些在应用程序的最终镜像中是不需要的。

```
# 例如基于Maven和tomcat的Java应用程序，如果不命名则从0开始算FROM
FORM maven AS build
WORKDIR /app
COPY . .
RUN mvn package

# 这里的--from也可以写成--from=0
FROM tomcat
COPY --from=build /app/target/file.war /usr/local/tomcat/webapps
# 最终构建镜像时可能只需要最上面的FROM
docker build --target build -t onlyzuri/myapp:v1.0.0
```