第6课 Docker教程

如果老师问为什么用Docker，就说这个更加轻量级，比操作系统更方便，然后简单介绍下面的Docker基本概念就行

什么是Docker?

Docker 是一款开源的应用容器引擎，基于 Go 语言开发，遵循 Apache 2.0 开源协议，它使用操作系统级虚拟化技术（如Linux的命名空间和控制组）来为应用程序提供轻量级、可移植且隔离的运行环境。开发者可以利用Docker将应用及其所有依赖打包成一个称为“容器”的标准单元，使得应用程序可以在任何安装了Docker的环境中快速部署和运行，且表现一致。

省流：它通过将应用程序及其依赖打包到一个可移植的容器中，实现了“一次构建，处处运行”的理念，极大地简化了应用的部署和交付流程。

为什么选择Docker作为轻量级服务器解决方案?

解决问题之“在我的电脑上能运行”

相较于传统的虚拟机，Docker有以下优势：

1. 轻量级：每个容器共享主机的操作系统内核，因此相比运行完整的操作系统，容器启动更快，占用资源更少。
2. 高效性：通过层叠的文件系统和高效的资源隔离机制，Docker能够实现高密度的容器部署，显著提高硬件利用率。
3. 便携性：由于容器包含了应用程序的所有依赖，它们可以在任何支持Docker的平台上轻松迁移和运行，确保了一致性和标准化。
4. 隔离性：容器之间在文件系统、网络配置以及进程空间上相互隔离，提供了安全可靠的多租户环境。

Docker的核心概念和工作原理

- **容器（Container）**：轻量级、可执行的软件包，封装软件代码及其所有依赖，保证应用在任何环境中都能一致地运行。
- **镜像（Image）**：容器的静态模版，包含创建容器所需的文件系统和应用程序。
- **层叠的文件系统**：镜像采用层叠的方式存储，每个层代表镜像的一部分。容器启动时，Docker叠加这些层并添加一个可写层。

    - Docker镜像不是一个单一的、巨大的文件，而是由一系列只读的层（或称为层叠的文件系统层）组成。每当你执行一个 `RUN` 命令来安装软件包、修改配置文件或添加文件时，Docker都会创建一个新的层，并记录下这些更改。每个新层仅包含相对于上一层的差异，因此形成了一个高效的增量式存储结构。
- **隔离性**：容器与主机和其他容器隔离，拥有自己的文件系统、网络配置和进程空间。
- **轻量级**：容器共享主机操作系统内核，运行在自己的隔离空间中，比虚拟机更轻量级。
- **可移植性**：容器包含应用程序及其所有依赖，可以在任何支持Docker的主机上运行。

容器和镜像的详细解释

- **容器**：

  容器是Docker的主要构建块，**它是轻量级、可执行的软件包**，封装了应用程序及其所有依赖，确保了应用在不同环境中都能以相同方式运行。容器具有隔离性，即拥有独立的文件系统、网络配置和进程空间；同时又具备轻量级特点，因为它不包含操作系统内核，而是与宿主机共享内核，从而节约资源。

    - **隔离性**：相互隔离，独立的文件系统、网络配置和进程空间。
    - **轻量级**：共享宿主机内核，运行在隔离的用户空间。
    - **可移植性**：在不同计算环境中无缝迁移和运行。
    - **一致性**：应用运行的一致性和环境的标准化。

- **镜像**：

  镜像是创建容器的静态模板，**它是一个只读的、分层的文件系统结构**，包含了应用程序及运行时所需的全部内容。每个镜像由一系列层组成，每层代表镜像的一部分更改，这样在创建新容器时只需加载必要的层即可，大大提高了效率。镜像可通过Dockerfile定义并被版本控制，方便用户共享、重用和维护。

    - **只读和分层的**：镜像是只读的，启动容器时添加可写层。
    - **重用和共享**：可通过Docker Hub等共享和重用。
    - **构建**：通过Dockerfile定义镜像构建步骤。
    - **版本控制和存储**：进行版本控制，每个版本有独立标签。

容器与镜像的关系：镜像是一个包含应用程序及其运行环境的只读模板，用于创建 Docker 容器；容器是镜像的一个运行实例，包含了应用程序和其依赖的运行环境。

Ubuntu操作系统与Ubuntu Docker的区别

- **Ubuntu操作系统**：

- 完整的Linux发行版，包括内核、文件系统、用户界面和应用程序。
- 直接与硬件交互，管理CPU、内存、存储和I/O设备。
- 提供图形和命令行用户界面。
- 需要独立分区安装，占用更多磁盘空间和系统资源。

- **Ubuntu Docker**：
  - 在Docker平台上运行的Ubuntu容器。
  - 依赖宿主机操作系统内核，不直接与硬件交互。
  - 通常不包含图形界面，主要通过命令行交互。
  - 只包含最小Ubuntu环境，资源占用远少于完整的Ubuntu操作系统。

操作系统与Docker的区别

- 资源管理：操作系统直接管理硬件资源，而Docker依赖宿主操作系统进行硬件资源管理。
- 运行层级：操作系统运行在硬件层面，提供系统级的服务；Docker作为应用层软件，运行在操作系统之上，提供应用隔离和部署服务。
- 隔离性与共享性：操作系统为每个应用提供全面的运行环境，应用间相对独立；Docker容器间共享宿主OS的内核，但在用户空间实现隔离。
- 部署与移植：操作系统部署在物理或虚拟机器上，通常与硬件紧密相关；Docker容器可以轻松迁移和部署，无需考虑底层硬件和宿主OS的差异。

Dockerfile 语法教程

1. Dockerfile 基础结构与命令概述

Dockerfile 是一个文本文件，用于定义如何构建Docker镜像。它包含了一系列指令，每条指令在新的一层中执行。

以下是一些基本的Dockerfile指令示例：

```
1  # 使用特定版本的Ubuntu镜像
```

```
# FROM ubuntu:latest
FROM ubuntu:latest

# 替换为清华大学的 Ubuntu 镜像源
RUN sed -i 's/http:\/\/ports.ubuntu.com/http:\/\/mirrors.tuna.tsinghua.edu.cn/g' /etc/apt/sources.list

# 更新软件包并安装所需的库
RUN apt-get update && \
    apt-get install -y --no-install-recommends build-essential python3 python3-pip libsqlite3-dev curl && \
    rm -rf /var/lib/apt/lists/*

# 复制代码到容器中
COPY . /usr/src/myapp

# 设置工作目录
WORKDIR /usr/src/myapp

# 暴露端口
EXPOSE 80 8080 8081

```

Dockerfile 中的一些常用指令说明：

Dockerfile 是一个文本文件，包含了一系列指令，用于构建 Docker 镜像。每条指令对应一个镜像层。

- **FROM**：指定基础镜像，是所有 Dockerfile 必须的指令。

```
FROM ubuntu:20.04
```

- **RUN**：在镜像内执行命令，常用于安装软件或依赖。

```
RUN apt-get update && apt-get install -y python3
```

- **COPY**：将文件或目录从上下文目录复制到镜像中。

```
COPY . /app
```

- **WORKDIR**：设置工作目录。

```
WORKDIR /app
```

- **CMD**：指定容器启动时要运行的命令。

```
CMD ["python3", "app.py"]
```

- **EXPOSE**：声明容器监听的端口。

```
EXPOSE 8080
```

- **ENV**：设置环境变量。

```
ENV DEBUG=true
```


Docker 常见命令教程：

- **docker build**: 使用Dockerfile构建镜像。

```
docker build -t my-image-name .
```

其中 `.` 表示当前目录作为构建上下文， `-t` 用来指定标签名。

- **docker run**: 创建并启动一个新的容器。

```
docker run -d --name container-name -p host-port:container-port my-image-name
```

`-d` 表示后台运行， `--name` 为容器命名， `-p` 进行端口映射。

- **docker start/stop/restart**: 控制容器的生命周期。

```
docker start container-name
docker stop container-name
docker restart container-name
```

- **docker ps**: 列出正在运行的容器。

```
docker ps
```

若要查看所有容器（包括未运行的），可使用 `docker ps -a` 。

- **docker exec**: 在运行中的容器内执行命令。

```
docker exec -it container-name bash
```

- **docker logs**: 查看容器的日志输出。

```
docker logs container-name
```

- **docker rm**: 删除容器。

```
docker rm container-name
```

- **docker rmi**: 删除镜像。

```
docker rmi my-image-name
```

以上仅为部分基础命令及Dockerfile示例，更多高级用法请参考官方文档。

虚拟机和Docker

虚拟机（Virtual Machine, VM）和Docker是两种不同的计算资源隔离和软件部署技术，它们的主要区别在于实现机制、资源利用率和灵活性等方面：

1. 实现机制：
   - 虚拟机：在宿主机上运行一个称为Hypervisor（或虚拟机监控器）的软件，该软件模拟完整的硬件环境，如CPU、内存、硬盘等。**每个虚拟机内都有自己的操作系统，这些操作系统与宿主机操作系统完全独立。**
   - Docker：基于容器技术，**它不是模拟整个硬件层，而是直接利用宿主机的操作系统内核**，并通过Namespace进行进程、网络、文件系统等资源隔离，以及通过cgroups进行资源限制。这意味着Docker容器共享宿主机的操作系统内核，但拥有各自独立的应用程序及其依赖环境。
2. 资源占用与效率：
   - 虚拟机：由于需要为每个VM运行一个完整操作系统及配套的服务，所以启动速度较慢，资源消耗相对较大，包括额外的内存开销（用于操作系统的内核空间）、磁盘空间（存储每个VM的镜像）和CPU开销。
   - Docker：因为容器不包含操作系统内核，因此**启动速度快得多，资源占用小**，更接近原生性能。多个容器共享同一内核，可以显著提高服务器的资源利用率。
3. 便携性：
   - 虚拟机：虽然VM镜像是可移植的，但在不同架构间迁移可能需要重新编译或适配，且迁移成本相对较高。

- Docker：Docker镜像具有很强的跨平台兼容性，可以在支持Docker的任何环境中快速部署。只要宿主机的操作系统内核版本一致，Docker容器就能在不同主机之间轻松迁移。

4. 应用场景：
   - 虚拟机：适合于需要多种操作系统环境或者对隔离度要求极高的场景，例如不同版本的操作系统测试、多租户环境中的客户隔离等。
   - Docker：特别适用于服务化架构和持续集成/持续部署（CI/CD）流程中，尤其当应用是由一组相互依赖的服务组成时，Docker容器可以简化开发、测试和部署过程，确保环境一致性。

Docker的其他优势（了解即可）

1. 容器编排：
   - Docker Compose：简化了多容器应用程序的定义、配置和运行流程，通过单一YAML文件统一管理依赖和服务。
   - Kubernetes：独立于Docker但广泛配合使用的容器编排系统，用于大规模集群中容器的部署、管理和自动化运维，包括服务发现、负载均衡、自动伸缩等。
2. 安全增强：
   - seccomp：限制容器内进程可调用的系统调用，强化安全性。
   - AppArmor和SELinux策略：为容器实施细粒度的安全规则，控制进程权限。
3. 存储驱动：
   - 支持多种存储驱动，如AUFS、Overlay2等，确保不同环境下的灵活性与性能需求得到满足。
4. 网络模式：
   - 提供多样化的网络模式，涵盖bridge（默认桥接网络）、host、none以及自定义网络，便于实现容器间通信、端口映射及DNS解析等网络功能。
5. 镜像仓库：
   - Docker Hub 是官方公共镜像仓库，同时支持搭建私有仓库如Harbor，以便安全地存储和分发镜像。
6. 构建优化：
   - Docker BuildKit 提升了镜像构建速度，并引入了多阶段构建等功能，有效减小镜像体积。
7. 集群管理：
   - Docker Swarm 将多个Docker守护进程整合成一个分布式集群引擎，简化跨多主机的容器集群部署与管理。
8. 数据管理：
   - Docker Secrets 和 Configs 在1.13版后引入，用于安全存储和分发敏感数据和配置信息给容器。

9. 插件系统：
    - Docker 支持扩展插件机制，覆盖网络、存储、日志处理等领域，以适应各种特定用户需求。

面试常考 Docker 问题及答案

问题1：Docker 与传统的虚拟机相比，有哪些优势？

回答：

- **资源占用更少**：Docker 容器共享宿主机内核，启动速度快，资源占用少。
- **环境一致性**：容器内的环境是标准化的，避免了环境配置差异导致的问题。
- **更快的部署速度**：容器的创建和销毁都非常迅速，适合敏捷开发和部署。

问题2：什么是 Dockerfile？它的作用是什么？

回答：Dockerfile 是一个文本文件，包含了一系列指令，用于定义如何构建 Docker 镜像。通过编写 Dockerfile，可以自动化地构建镜像，确保构建过程的可重复性和一致性。

问题3：如何在 Docker 容器之间共享数据？

回答：可以使用 **数据卷（Volume）** 或 **挂载主机目录** 的方式来共享数据。数据卷是由 Docker 管理的，可以在多个容器之间共享；挂载主机目录则可以将宿主机的目录映射到容器内，实现数据共享。

问题4：解释一下 Docker 的分层镜像技术。

回答：Docker 镜像采用了分层存储（UnionFS）技术，每个镜像由多层只读层组成，每一层都基于前一层进行修改。这样可以复用公共层，节省存储空间，加快镜像的构建和分发。

问题 5: Docker 是什么？为什么要使用 Docker?

回答: Docker 是一个开源的容器化平台，它可以让开发者打包应用程序及其依赖项到一个可移植的容器中，确保应用程序可以在任何环境下稳定运行。使用 Docker 的主要原因包括：

- **隔离性**：每个容器都有独立的运行环境，不会干扰主机系统或其他容器。
- **可移植性**：容器可以在不同的操作系统和云平台上无缝运行。
- **效率**：相比虚拟机，容器更加轻量，启动和运行效率更高。
- **简化部署**：通过 Docker，可以更轻松地部署和管理应用程序