Docker及其应用

Docker作为当今云计算和应用部署领域的领先技术，正改变着我们构建、分发及运行软件的方式。

什么是Docker?

Docker是一种开源容器化平台，用于开发、交付和运行应用。它提供了将软件编译成一个便携、自给自足的容器，从而简化部署和扩展过程。

为什么选择Docker

解决问题："在我的电脑上能运行"

它提供了 **一致的运行环境**。这是通过容器化应用程序来实现的，容器将代码、配置文件、依赖库等应用运行所需的一切都打包在一起，使得在任何 Docker 支持的平台上运行时，环境都是一致

Docker的核心概念和工作原理

容器

打包应用及其依赖的软件单元，保障跨环境运行一致性。

镜像

静态的层叠文件系统，是创建容器的模板。

文件系统

通过层叠的方式存储，提高存储的效率和速度。

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。

镜像特点

分层结构

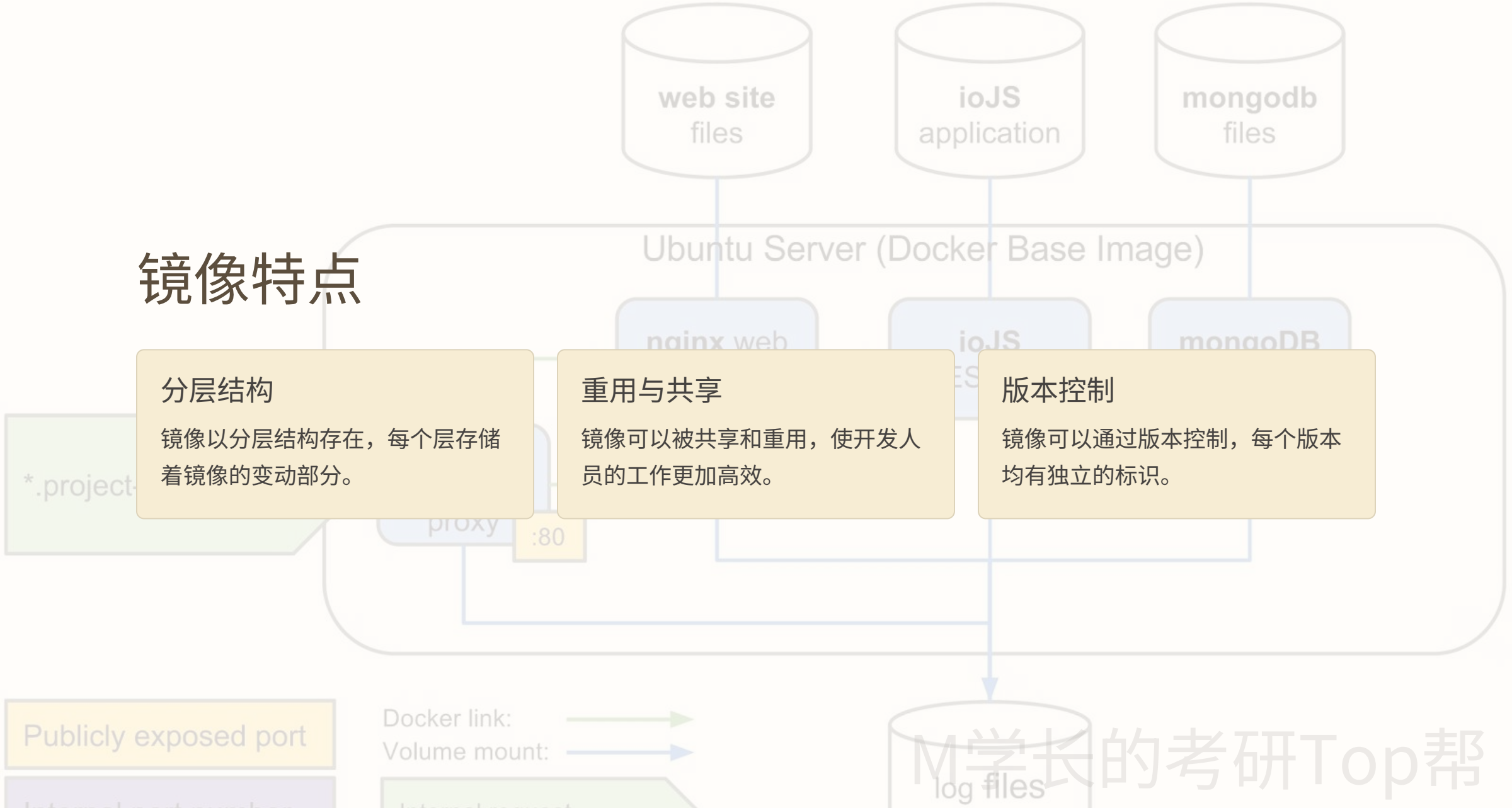
镜像以分层结构存在，每个层存储着镜像的变动部分。

重用与共享

镜像可以被共享和重用，使开发人员的工作更加高效。

版本控制

镜像可以通过版本控制，每个版本均有独立的标识。

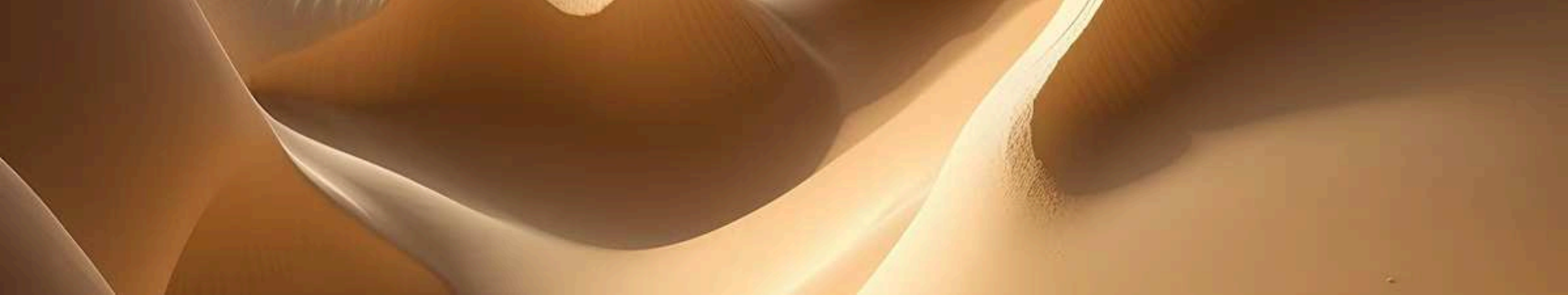


容器与镜像的异同

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。

容器	镜像
运行时实体，隔离性和可移植性	静态定义，分层存储
可执行软件包	构建容器的模板
动态和可写	静态和只读



Docker如何解决问题

“我的电脑能运行”

1. 容器打包应用和依赖

Docker 容器包含应用程序的所有依赖，包括：

- **代码**：应用程序的源代码或编译后的二进制文件。
- **依赖库**：应用程序需要的各种库文件、包管理器等。例如，Python 的容器可以包含 Python 解释器和所有所需的依赖库。
- **配置文件**：运行应用程序的相关配置文件，确保配置在不同环境下保持一致。
- **操作系统层面的依赖**：如系统库、工具等，容器可以基于特定的基础镜像（如 Ubuntu、Alpine）构建，确保运行时所需的环境一致。

通过这种打包方式，Docker 容器**封装了应用的整个环境**，使得应用在不同系统上运行时不会因为环境差异导致问题

2. 隔离的运行环境

Docker 容器提供了**进程、文件系统、网络、资源等的隔离性**，这意味着容器内的应用程序与主机系统和其他容器完全隔离。

- **文件系统隔离**：每个容器有自己独立的文件系统，应用程序只能看到容器内的文件和依赖，外部系统的变更不会影响容器内的应用。
- **进程隔离**：容器中的应用程序以独立的进程运行，与主机和其他容器的进程完全分离，不会相互影响。

这种隔离确保了容器内的环境始终与创建时相同，**避免了外部环境的干扰**，从而保证了容器的运行一致性。

3. 跨平台兼容性

Docker 容器基于**操作系统级虚拟化**，通过 Docker 引擎运行，不依赖主机操作系统的具体配置。只要主机安装了 Docker，无论是 Linux、Windows 还是 macOS，Docker 容器都可以在这些平台上运行。

- 容器基于 Docker 镜像，镜像定义了应用所需的所有组件和环境，确保了跨平台运行时的环境一致性。
- 应用打包为容器后，Docker 引擎提供的抽象层消除了底层硬件和操作系统的差异，因此可以在任意 Docker 支持的平台上运行，环境都是一致的

4. 基础镜像确保一致性

Docker 提供了**基础镜像**（如 `ubuntu`、`node`、`python` 等），这些基础镜像包含了某个特定操作系统或运行环境的基础组件。开发者可以在这些基础镜像之上构建自己的应用程序镜像，确保应用的操作系统和依赖库版本是确定且一致的。

- 例如，基于同样的 `python:3.8` 镜像构建的容器，无论在哪台主机上运行，都会提供相同的 Python 版本和相关依赖，确保 Python 应用在不同平台上的行为是一致的。

5. Docker 镜像的不可变性

Docker 镜像是**不可变的**，每个容器都是从同一个镜像生成的。这意味着开发者在开发环境中创建的镜像在生产环境中运行时不会发生变化。镜像的不可变性确保了：

- 不同环境下运行时不会产生**“环境漂移”**（环境差异）的问题。
- 开发、测试、生产环境的镜像完全一致，从而避免了由于环境差异导致的故障。

选择Docker的优势

1 轻量级

Docker容器共享操作系统内核，启动速度比虚拟机快，资源利用率高。

2 高效性

Docker的层叠文件系统高效存储，容器化部署提高硬件利用。

3 便携性

容器封装依赖，确保在任何Docker环境中一致运行。

4 隔离性

容器间相互隔离，为不同租户提供安全的运行环境。

文件系统

Docker 文件系统是 Docker 容器化技术中用于管理容器内部文件和目录的机制。Docker 容器的文件系统是基于**联合文件系统（Union File System）实现的，允许多个层叠加在一起，从而实现文件共享、资源隔离和高效管理。Docker 文件系统具有分层结构**和写时复制（Copy-on-Write, CoW）特性，确保了容器的轻量级和高效性。

Ubuntu操作系统与Ubuntu Docker的区别

Ubuntu操作系统

完整的系统，管理硬件资源，占用较多空间。

Ubuntu Docker

提供最小化Ubuntu环境，利用主机内核，资源占用少。

操作系统与Docker的区别

1

资源管理

操作系统直接管理，Docker依赖主机OS管理。

2

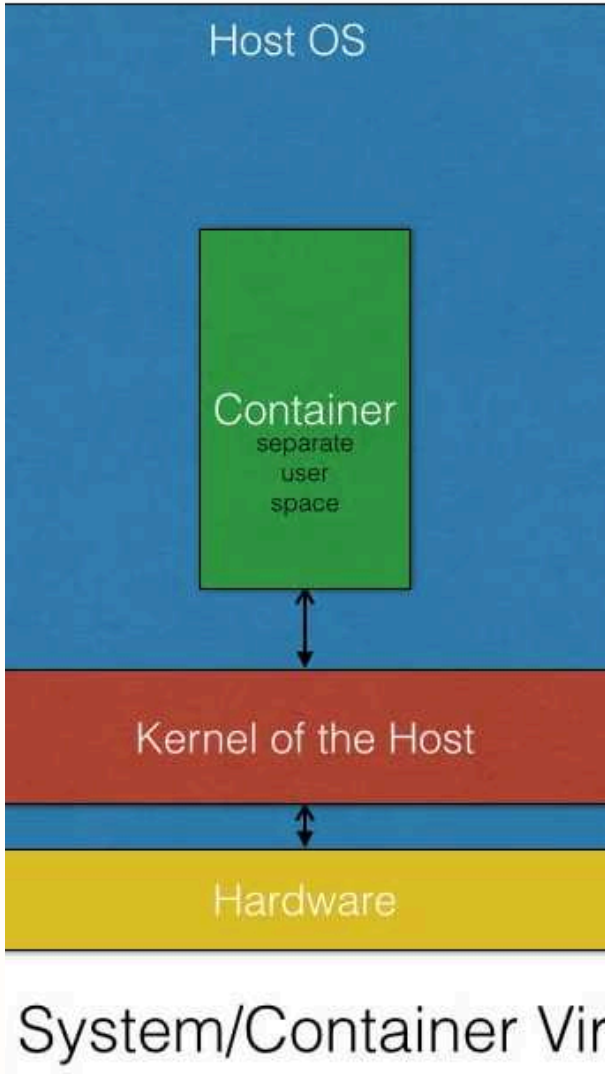
运行层级

操作系统在硬件上，Docker在操作系统之上。

3

隔离与共享

操作系统为应用单独运行，Docker多个容器间共享内核。



System/Container Vir

虚拟机

虚拟机（Virtual Machine, VM） 是一种基于软件的技术，允许在一台物理计算机上运行多个相互独立的操作系统，每个操作系统运行在一个独立的虚拟环境中。虚拟机技术通过创建**虚拟的硬件资源**（如 CPU、内存、存储、网络等），使得多个操作系统可以共享同一套物理硬件资源，但彼此互相隔离。

虚拟机的核心概念

- **虚拟化**： 虚拟化是虚拟机的基础技术，它通过一层虚拟化层（通常由**虚拟机管理程序**或**Hypervisor**实现）来抽象底层硬件资源，并将这些资源分配给运行在虚拟机中的操作系统。这使得每个虚拟机能够像真实的物理计算机一样运行独立的操作系统和应用程序。
- **虚拟机管理程序（Hypervisor）**： 虚拟机管理程序是虚拟化技术的核心组件，它负责创建、运行和管理虚拟机。根据运行位置的不同，虚拟机管理程序分为两类：
  - **Type 1（裸金属 Hypervisor）**：直接运行在物理硬件上，不依赖于主机操作系统，如 VMware ESXi、Microsoft Hyper-V、Xen。
  - **Type 2（托管式 Hypervisor）**：运行在主机操作系统之上，再在其上创建虚拟机，如 VMware Workstation、Oracle VirtualBox。

虚拟机的核心概念

- **虚拟硬件**： 每个虚拟机都有自己独立的虚拟硬件设备，包括虚拟的 CPU、内存、硬盘、网络接口等。这些虚拟设备由 Hypervisor 提供，虚拟机内的操作系统和应用程序将这些设备视为真实的硬件设备。
- **独立的操作系统环境**： 每个虚拟机可以运行完全不同的操作系统，无论是 Linux、Windows、macOS，还是其他操作系统，彼此之间相互独立，互不干扰。虚拟机内的操作系统被称为“**客操作系统**”或“**虚拟操作系统**”。

虚拟机的工作原理

1 抽象硬件资源

虚拟机管理程序将物理服务器的硬件资源抽象为虚拟资源，并将这些虚拟资源分配给虚拟机。

2 独立虚拟硬件

每个虚拟机都有自己独立的虚拟硬件设备，包括虚拟的 CPU、内存、硬盘、网络接口等。

3 完全隔离

虚拟机之间是完全隔离的，运行在一个虚拟机中的进程和应用不会影响其他虚拟机。

4 隔离资源

每个虚拟机的文件系统、内存和进程表对其他虚拟机都是不可见的。

虚拟机的优点

- **资源利用率提升**：通过在一台物理服务器上运行多个虚拟机，虚拟化技术可以有效提升硬件资源的利用率。多个虚拟机可以共享同一套硬件资源，而不会导致闲置资源浪费。
- **隔离性和安全性**：虚拟机之间彼此隔离，运行在一个虚拟机中的程序无法影响其他虚拟机，即使某个虚拟机崩溃，也不会影响其他虚拟机的正常运行。
- **跨平台兼容性**：虚拟机允许在同一台物理主机上运行不同的操作系统，支持跨平台开发和测试。例如，你可以在 Windows 主机上运行 Linux 虚拟机，测试不同平台下的应用。
- **灵活性和可移植性**：虚拟机可以轻松地在不同的物理服务器之间迁移，支持弹性扩展和资源调度。虚拟机的快照功能还可以轻松进行备份和恢复，方便管理和维护。
- **开发和测试环境**：开发者可以快速创建和销毁虚拟机，用于开发、测试和调试。不同的开发环境可以在不同的虚拟机中配置，而不会干扰主机操作系统。

虚拟机的缺点

1. **资源开销较大：** 虚拟机需要虚拟化整个操作系统，因此每个虚拟机会消耗较多的 CPU、内存和存储资源。与轻量级的容器技术相比，虚拟机的资源利用率较低。
2. **性能损失：** 由于虚拟机是通过虚拟化技术模拟硬件，虚拟机中的应用程序在性能上会比直接运行在物理机上的应用稍有损失，尤其是在 I/O 密集型或高性能计算任务中。

虚拟机和Docker的比较

Docker 相比虚拟机的主要优势可以总结为以下几条：

1. **轻量级**：Docker 容器共享主机操作系统内核，不需要运行完整的操作系统，资源占用更少。
2. **启动速度快**：容器启动只需几秒甚至毫秒，而虚拟机需要启动完整的操作系统，通常较慢。
3. **一致的运行环境**：容器打包了应用程序及其依赖，确保开发、测试和生产环境一致，避免环境差异问题。
4. **更高的资源利用率**：容器可以在同样的硬件上运行更多实例，资源利用效率更高。
5. **跨平台部署**：Docker 容器可以在不同操作系统上无缝运行，而无需关心底层环境。

虚拟机和Docker的比较

虽然 Docker 提供了轻量级、快速启动、资源利用率高的优势，但相对于虚拟机，Docker 容器的不足之处主要包括以下几点：

1. **隔离性和安全性较弱**，无法像虚拟机那样提供完全的操作系统隔离。
2. **只能运行与主机操作系统内核兼容的操作系统**，无法像虚拟机一样运行不同架构的操作系统。
3. **数据持久化管理复杂**，不如虚拟机的持久化机制直观和简单。
4. **不支持完全的硬件虚拟化**，在一些特殊硬件需求场景下（如 GPU 虚拟化），可能不如虚拟机灵活。
5. **容器网络和资源管理相对复杂和不足**，尤其在大规模部署和复杂网络场景下不如虚拟机成熟。

Dockerfile 语法教程

基础结构

每条指令创建新的层，有序地构建镜像。

Dockerfile常用指令

1

FROM

指定构建镜像的基础。

2

RUN

在新层中执行命令，构建文件系统。

3

CMD

定义容器启动后的默认命令。

	Important Options	
	-t Define image name and tag	Create a new imag
G...]	**-e** Set environment variables **-i** Interactive (allow to attach local stdin, stdout, and stderr) **-p** Define port mapping **-t** Set up a terminal in the container **-v** Define a volume	**docker crea** Create a new **Ubu** **FOO**in the contai directory o
	–a Attach **stdout**and **stderr**(show stdout and stderr of container on host) **-i** Attach **stdin**(read stdin for container from host)	Start a previously c have the same
		Attach local stdc
		Stop a running co
G...]	**-d** Don't attach to the container **-e** Set environment variables **-i** Interactive (allow to attach local stdin, stdout, and stderr) **-p** Define port mapping **-t** Set up a terminal in the container **-v** Define a volume	**docker run** Same example as **run**
	-a List all containers, not just running ones **-q** List only IDs of containers	
	-a Also list intermediate images **-q** List only IDs of images	
	-f Force removal **-v** Also remove volumes	
	-f Force removal	

常见指令

- FROM: 必须是Dockerfile中的第一条非注释指令，用于指定构建新镜像的基础镜像。
- RUN: 执行任意shell命令或Linux命令，生成新的镜像层。
- COPY: 从构建上下文目录复制文件/目录到容器内部指定路径。
- ADD: 类似于COPY，但还支持tar归档和自动解压功能。
- ENV: 设置环境变量，可以在后续命令中使用这些变量。
- WORKDIR: 设定工作目录，所有接下来的RUN、CMD、ENTRYPOINT等命令将在该目录下执行。
- CMD: 容器启动后默认执行的命令。可以被docker run命令后的参数覆盖。
- ENTRYPOINT: 同样定义容器启动时执行的命令，但不能被docker run命令后的参数直接覆盖，除非通过--entrypoint选项重写。
- EXPOSE: 指定容器监听的端口，但并不映射到宿主机，需要在运行容器时使用 -p 参数进行端口映射。
- VOLUME: 指定持久化存储区域，用于数据持久化。
- USER: 设置运行容器的用户ID或者用户名。
- HEALTHCHECK: 配置对容器健康状况的检查方式。

Docker常用命令介绍

常见命令

- 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

常见命令

- 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

Docker的其他优势

1 **容器编排**

通过Docker Compose和Kubernetes简化多容器管理。

2 **安全增强**

采用seccomp、AppArmor和SELinux策略提高安全性。

3 **灵活的存储与网络**

支持各种存储驱动，多样化的网络模式。

Docker镜像和集群管理

1 镜像仓库

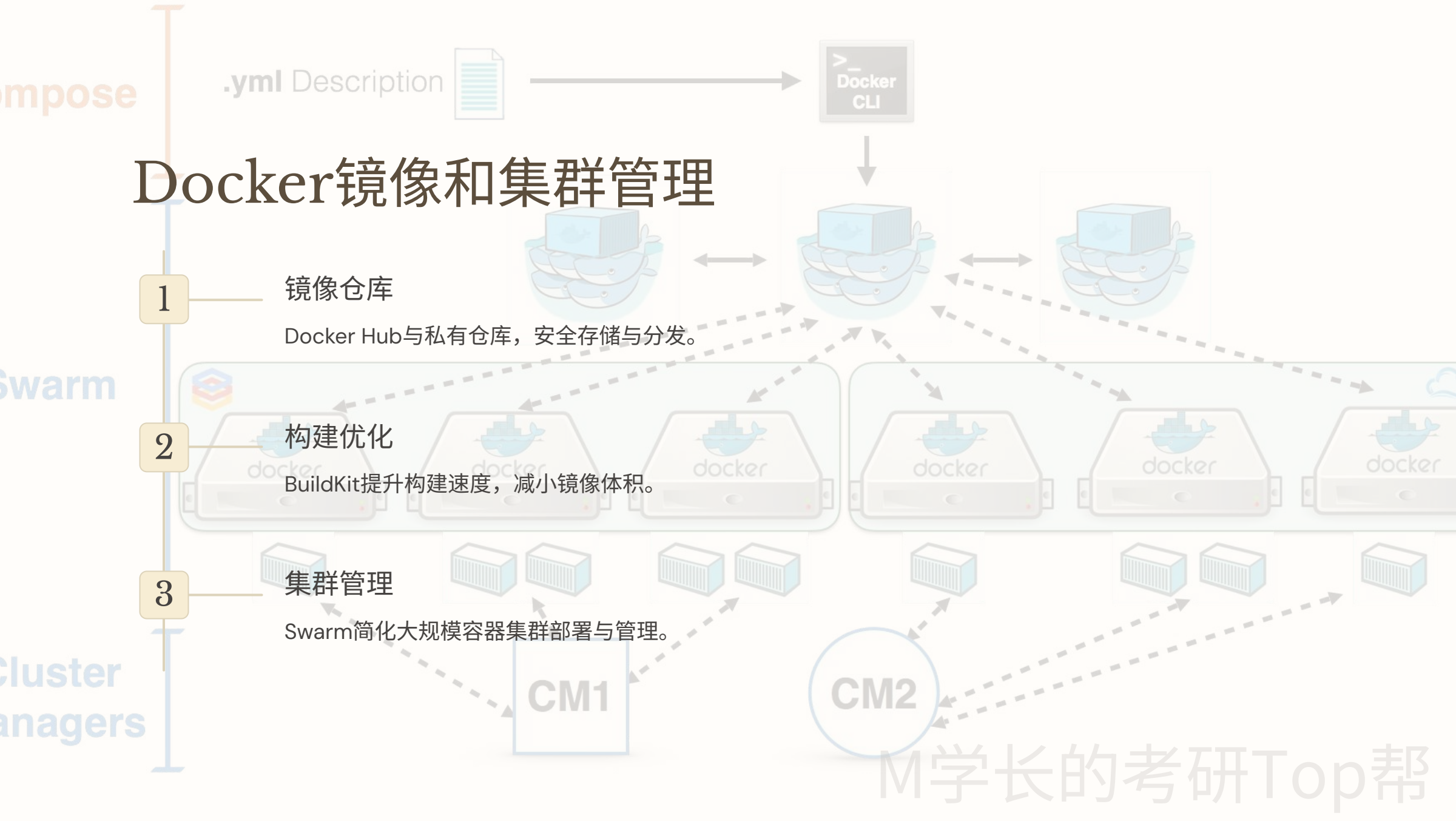
Docker Hub与私有仓库，安全存储与分发。

2 构建优化

BuildKit提升构建速度，减小镜像体积。

3 集群管理

Swarm简化大规模容器集群部署与管理。



镜像仓库

Docker 镜像是用来打包应用和依赖的文件，类似于可执行程序。镜像仓库是用来存储和分发这些镜像的地方。常见的镜像仓库有：

- **Docker Hub**：官方的公共镜像仓库，可以免费使用和分发公开镜像，也支持私有镜像（付费）。

- **私有仓库**：在某些场景下，企业需要更高的安全性或控制权，可以搭建自己的私有镜像仓库来存储和分发镜像。可以使用工具如 Docker Registry 搭建。

无论使用 Docker Hub 还是私有仓库，都可以安全地存储和共享镜像，使团队成员更方便地分发和部署应用。

构建优化

构建 Docker 镜像的效率和质量非常重要，好的构建优化可以提升速度并减小镜像体积。

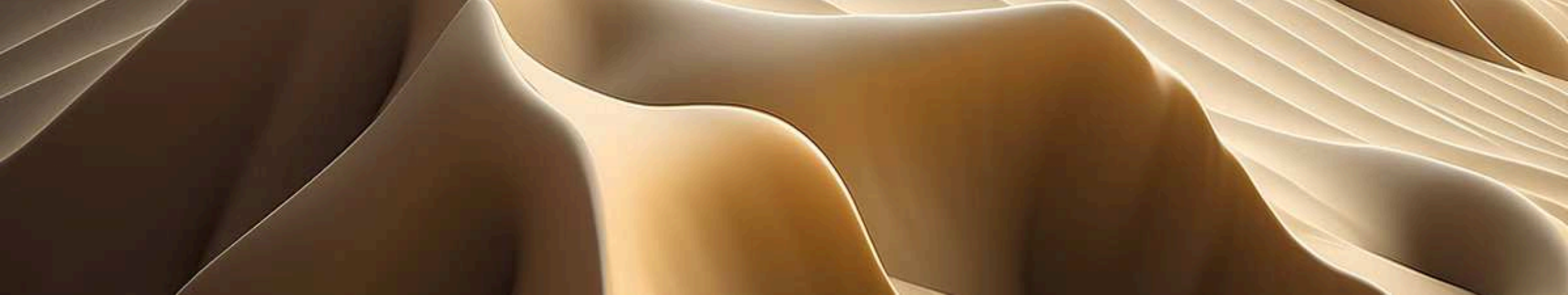
• **BuildKit**：是 Docker 提供的新构建工具，可以显著加快镜像的构建速度。它支持并行构建和缓存重用，减少重复步骤的执行。

• **减小镜像体积**：可以通过多阶段构建（multi-stage builds）、使用轻量级基础镜像、减少镜像层数等方式来减小镜像大小，提高传输和部署效率。

集群管理

集群管理指的是管理大量容器的部署、扩展和编排工作。

• **Swarm**：是 Docker 自带的集群管理工具，能够简化大规模容器集群的部署和管理。Swarm 提供了自动化部署、容器负载均衡、容器扩缩容、故障恢复等功能，并且与 Docker 的命令行工具无缝集成，便于快速上手。使用 Swarm，你可以轻松将多个 Docker 容器编排到一起，并实现集群的高可用性。



关于Docker面试中常见的问题

问题：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：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。

谢谢大家