一、了解 docker

1、 docker 是什么？

Docker 是一种开源的容器化平台，用于轻松地打包、发布和运行应用程序。它允许开发人员将应用程序及其所有依赖项（包括库、环境变量、配置文件等）打包到一个称为容器的单独单元中。这个容器可以在几乎任何地方运行，而不必担心环境差异或依赖冲突。Docker 利用操作系统级别的虚拟化技术，如 Linux 容器（LXC），以实现这种轻量级的隔离。通过使用 Docker，开发人员可以更快速、可靠地构建、测试和部署应用程序，从而简化了应用程序开发和交付的过程。

2、 容器与虚拟机的区别

2.1、 虚拟机：

2.1.1、 虚拟机是什么？

虚拟机是一种软件模拟的计算机系统，可以在物理计算机上运行多个独立的操作系统实例。虚拟机使得一台物理计算机可以同时运行多个操作系统和应用程序，每个虚拟机都像是一台独立的计算机，具有自己的 CPU、内存、存储和网络接口。虚拟机技术通过在物理硬件上创建一个或多个虚拟的计算环境，从而实现了对硬件资源的高效利用和隔离。常见的虚拟机软件包括 VMware、VirtualBox 和 Hyper-V 等。

2.1.2、 虚拟机与传统物理机优势：

- 虚拟机的优势：

资源利用率高：虚拟机允许在一台物理服务器上同时运行多个虚拟机，有效利用了硬件资源，提高了资源利用率。

灵活性和可移植性：虚拟机可以在不同的物理服务器之间迁移，也可以在不同的虚拟化平台上运行，提供了更大的灵活性和可移植性。

资源隔离：虚拟机之间具有良好的隔离性，一个虚拟机的故障不会影响其他虚拟机，提高了系统的稳定性和可靠性。

快速部署：通过虚拟机模板，可以快速部署新的虚拟机实例，加快了应用程序的部署速度。

灵活的配置和管理：虚拟机可以根据需要调整配置，如 CPU、内存、存储等，同时也提供了丰富的管理工具和接口，便于管理和监控。

- 传统物理机的优势：

性能：传统物理机直接访问硬件资源，因此通常具有更高的性能，特别是对于需要大量计算资源或对性能要求较高的应用程序。

直接硬件访问：传统物理机直接与物理硬件进行交互，可以获得更低的延迟和更高的吞吐量，适用于一些对性能要求较高的场景。

简单：相对于虚拟化环境，传统物理机的部署和管理通常更简单直接，不需要额外的虚拟化软件和管理工具。

适用于特殊硬件需求：对于一些需要直接访问特定硬件设备或资源的应用场景，传统物理机可能更适合。

2.2、 容器：

2.2.1、 容器是什么？

容器是一种虚拟化技术，用于打包和运行应用程序及其所有依赖项，包括库、运行时、系统工具和系统库等，使其可以在任何环境中以相同的方式运行。与传统的虚拟机相比，容器更加轻量级，因为它们共享主机操作系统的内核，并且没有额外的操作系统镜像。这使得容器的启动速度更快、资源消耗更少，同时也提供了更高的可移植性和扩展性。

2.2.2、 容器与虚拟机优势：

- 容器的优势：

轻量级：容器共享主机操作系统的内核，因此相比虚拟机更轻量级，启动更快，占用更少的资源。

高可移植性：容器可以在任何支持容器引擎的环境中运行，保持一致的运行行为，提高了可移植性。

快速启动：由于容器不需要启动完整的操作系统，其启动时间比虚拟机更快，有助于快速部署和弹性伸缩。

资源利用率高：容器可以在同一台主机上运行多个实例，提高了系统资源的利用率和密度。

一致的运行环境：容器包含应用程序及其所有依赖项，确保在不同环境中具有一致的运行环境，减少了因环境差异引起的问题。

- 虚拟机的优势：

更强的隔离性：虚拟机提供了硬件级别的隔离，每个虚拟机都有独立的操作系统和内核，适用于多租户环境和需要更高隔离性的场景。

更广泛的兼容性：虚拟机可以运行不同类型的操作系统，包括不同版本的 Linux、Windows 等，适用于需要同时运行多种操作系统的场景。

更灵活的配置：虚拟机可以根据需求调整配置，如 CPU、内存、存储等，提供更灵活的配置选项。

适用于特殊硬件需求：对于需要直接访问特定硬件设备或资源的应用场景，虚拟机可能更适合。

更高的安全性：由于虚拟机提供了更强的隔离性，因此在一些对安全性要求较高的场景中，虚拟机可能更合适。

总的来说，容器适用于轻量级、可移植和快速部署的场景，而虚拟机适用于需要更强隔离性、更广泛兼容性和更灵活配置的场景。在实际应用中，根据具体的需求和系统要求选择合适的虚拟化技术。

2.2.3、 容器与虚拟机的隔离性：

- 容器的隔离性：

进程级隔离：容器运行在共享主机操作系统内核的环境中，因此容器之间的隔离是基于进程的。每个容器都有自己的文件系统和网络命名空间，但它们共享主机操作系统的内核。这使得容器之间的隔离性相对较弱，一些特殊情况下可能存在横向攻击的潜在风险。

Cgroups 和命名空间：容器使用 Linux 内核的 cgroups（控制组）和命名空间来实现资源隔离，包括 CPU、内存、磁盘和网络。这些技术允许容器拥有自己的资源配额和视图，但仍然共享主机内核。

共享内核：由于容器共享主机内核，可能存在一些内核漏洞或安全性风险，因此在一些对安全性要求极高的场景中，需要采取额外的安全措施。

- 虚拟机的隔离性：

硬件级隔离：虚拟机提供了硬件级别的隔离，每个虚拟机都运行在独立的虚拟硬件环境中，包括虚拟 CPU、虚拟内存、虚拟网络接口等。这种隔离性更强大，减少了虚拟机之间的直接影响。

独立操作系统：每个虚拟机都有自己独立的操作系统和内核，这使得虚拟机之间的隔离性更高。即使主机操作系统受到攻击，虚拟机之间的影响也会被限制。

虚拟交换机：虚拟机通常连接到虚拟交换机，这提供了更高级别的网络隔离，可以防止虚拟机之间直接通信。

总的来说，虚拟机提供了更强大的硬件级隔离，每个虚拟机都拥有独立的操作系统和内核。而容器在进程级别上提供了隔离，共享主机内核，这在轻量级、快速启动和高密度部署等方面具有优势，但在一些对隔离性要求极高的安全场景中可能需要采取额外的安全措施。选择容器或虚拟机时，应根据具体需求权衡它们的优缺点。

2.3、 容器与虚拟机区别：

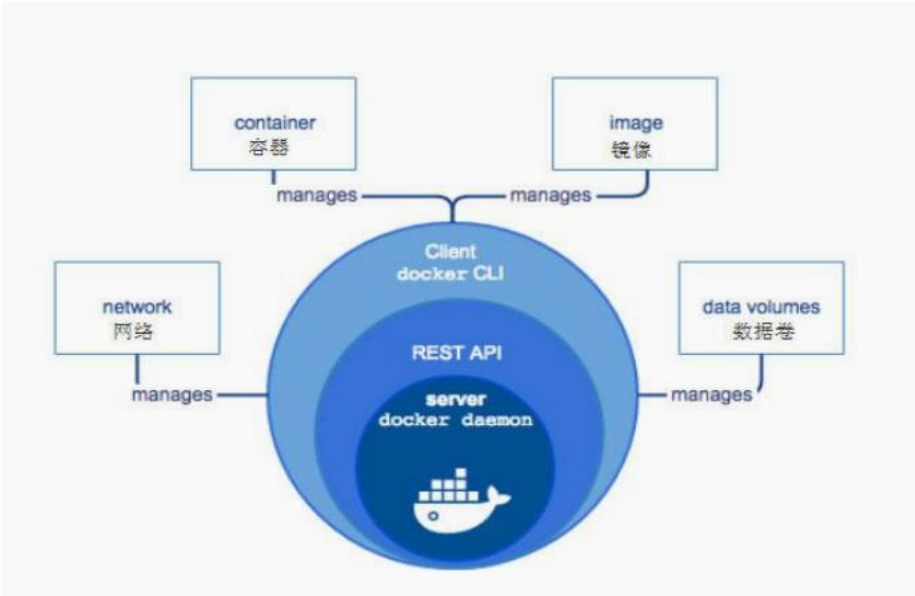
3、 docker 引擎

3.1、 docker 引擎组件

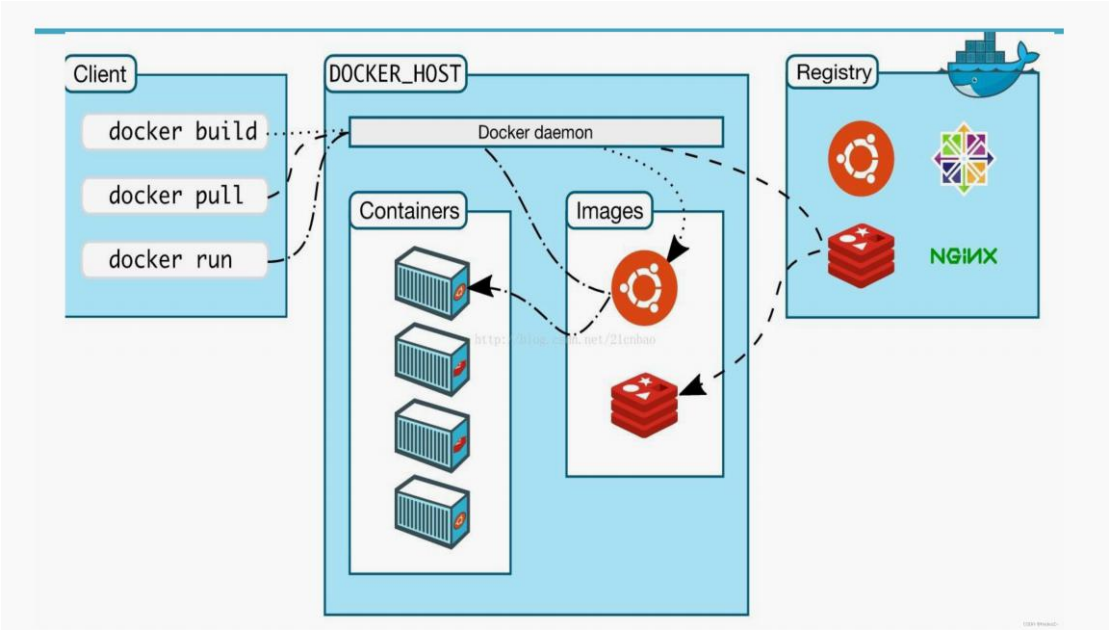
3.1.1、 client

管理 docker 平台对象（容器、网络、镜像、数据卷）

3.1.2、 Server



4、 docker 架构



5、 所用的底层技术

- Namespace 系统调用参数隔离内容
- UTS CLONE_NEWUTS 主机名与域名
- IPC CLONE_NEWIPC 信号量、消息队列和共享内存
- PID CLONE_NEWPID 进程编号
- Network CLONE_NEWNET 网络设备、网络栈、端口等等
- Mount CLONE_NEWNS 挂载点（文件系统）
- User CLONE_NEWUSER 用户和用户组

6、 docker 优势

- 跨平台性
- 轻量级和高效性
- 快速部署
- 标准化
- 可移植性
- 可伸缩性
- 环境隔离

二、docker 安装

1、 了解 docker 版本

- Docker Engine 社区版（Docker CE）
- Docker Engine 企业版（Docker EE）

2、 docker-ce 所支持的平台

2.1、 windows 操作系统

- windows server

2.2、 Linux 操作系统

- centos
- debian
- ubuntu

3、 docker 的安装方式

3.1、 在 CentOS 上安装 Docker：

- 安装所需的软件包：

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

- 设置 Docker 稳定版仓库：

yum-config-manager--add-repo https://download.docker.com/linux/centos/docker-ce.repo

- 安装最新版本的 Docker CE：

yum install docker-ce

- 启动 Docker 服务：

systemctl start docker

- 验证 Docker 是否正确安装：

docker --version

三、Docker 命令行使用

1、 快速入门

1.1、 查看 docker 运行状态

```
[root@localhost ~]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: dis>
   Active: active (running) since Sun 2024-03-10 19:57:38 CST; 5min ago
     Docs: https://docs.docker.com
 Main PID: 1195 (dockerd)
    Tasks: 9
   Memory: 111.8M
   CGroup: /system.slice/docker.service
           └─1195 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.so>

Mar 10 19:57:33 localhost.localdomain systemd[1]: Starting Docker Application Containe>
Mar 10 19:57:36 localhost.localdomain dockerd[1195]: time="2024-03-10T19:57:36.6137975>
```

1.2、 启动 docker 的 nginx 服务器

```
[root@localhost ~]# docker run -d --name nginx -p 80:80 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
e1caac4eb9d2: Pull complete
88f6f236f401: Pull complete
c3ea3344e711: Pull complete
cc1bb4345a3a: Pull complete
da8fa4352481: Pull complete
c7f80e9cdab2: Pull complete
18a869624cb6: Pull complete
Digest: sha256:c26ae7472d624ba1fafd296e73cecc4f93f853088e6a9c13c0d52f6ca5865107
Status: Downloaded newer image for nginx:latest
e10d51e4048dfbca5a60d0e4c486b2477e786be157a0b456ab84e67d68dcdbb6
[root@localhost ~]#
```

1.3、 查看 nginx 运行的日志信息

```
[root@localhost ~]# docker logs -f nginx
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
```

1.4、 查看 docker 中的当前进程

```
[root@localhost ~]# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                                   NAMES
e10d51e4048d   nginx     "/docker-entrypoint.…"   8 minutes ago   Up 8 minutes   0.0.0.0:80->80/tcp, :::80->80/tcp   nginx
[root@localhost ~]#
```

1.5、 查看端口号来检查 nginx 服务是否启动

```
[root@localhost ~]# ss -lntup
Netid State  Recv-Q Send-Q Local Address:Port   Peer Address:Port Process
udp   UNCONN 0      0          127.0.0.1:323         0.0.0.0:*     users:(("chronyd",pid=890,fd=5))
udp   UNCONN 0      0              [::1]:323            [::]:*     users:(("chronyd",pid=890,fd=6))
tcp   LISTEN 0      2048         0.0.0.0:80          0.0.0.0:*     users:(("docker-proxy",pid=1932,fd=4))
tcp   LISTEN 0      128          0.0.0.0:22          0.0.0.0:*     users:(("sshd",pid=921,fd=3))
tcp   LISTEN 0      2048           [::]:80             [::]:*     users:(("docker-proxy",pid=1936,fd=4))
tcp   LISTEN 0      128            [::]:22             [::]:*     users:(("sshd",pid=921,fd=4))
[root@localhost ~]#
```

1.6、 nginx 运行结果