Docker概述

预计阅读时间： 10分钟

Docker是一个用于开发，交付和运行应用程序的开放平台。Docker使您能够将应用程序与基础架构分开，从而可以快速交付软件。借助Docker，您可以以与管理应用程序相同的方式来管理基础架构。通过利用Docker的快速交付，测试和部署代码的方法，您可以大大减少编写代码和在生产环境中运行代码之间的延迟。

Docker平台

Docker提供了在松散隔离的环境（称为容器）中打包和运行应用程序的功能。隔离和安全性使您可以在给定主机上同时运行多个容器。容器是轻量级的，因为它们不需要管理程序的额外负担，而是直接在主机的内核中运行。这意味着与使用虚拟机相比，您可以在给定的硬件组合上运行更多的容器。您甚至可以在实际上是虚拟机的主机中运行Docker容器！

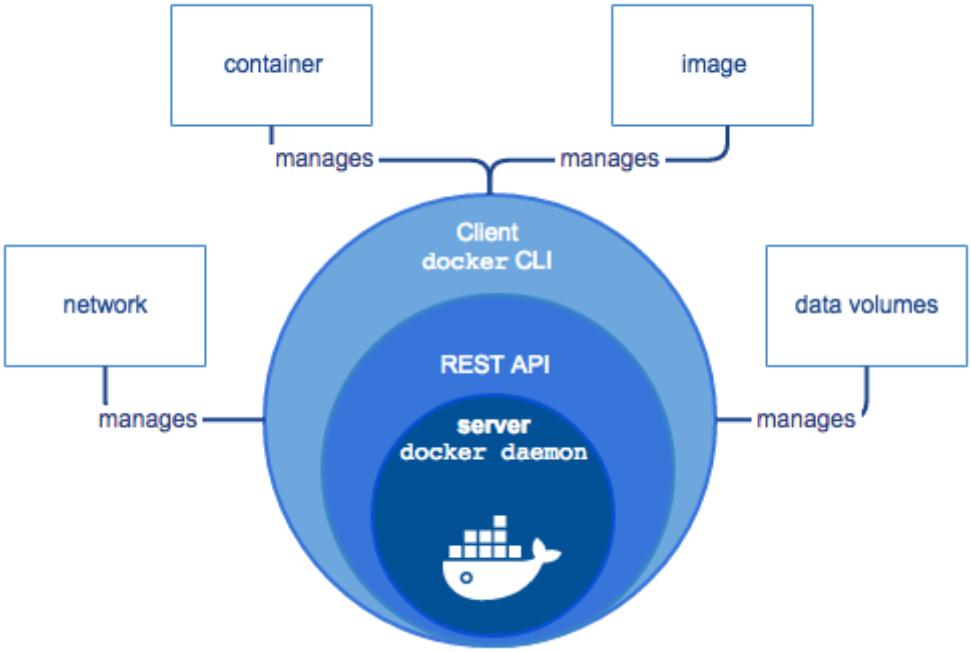
Docker提供了工具和平台来管理容器的生命周期：

- 使用容器开发应用程序及其支持组件。
- 容器成为分发和测试应用程序的单元。
- 准备就绪后，可以将应用程序作为容器或协调服务部署到生产环境中。无论您的生产环境是本地数据中心，云提供商还是两者的混合，其工作原理都相同。

Docker引擎

*Docker Engine*是具有以下主要组件的客户端-服务器应用程序：

- 服务器是一种长期运行的程序，称为守护程序进程（ `dockerd` 命令）。
- REST API，它指定程序可以用来与守护程序进行通信并指示其操作的接口。
- 命令行界面（CLI）客户端（ `docker` 命令）。



CLI使用Docker REST API通过脚本或直接CLI命令控制或与Docker守护程序交互。许多其他Docker应用程序都使用基础API和CLI。

守护程序创建和管理Docker *对象*，例如镜像 容器，网络和卷。

> **注意**：Docker已获得开源Apache 2.0许可证的许可。

有关更多详细信息，请参阅下面的Docker体系结构 (/engine/docker-overview/#docker-architecture)。

我可以将Docker用于什么？

快速，一致地交付您的应用程序

Docker通过允许开发人员使用提供您的应用程序和服务的本地容器在标准化环境中工作，从而简化了开发生命周期。容器非常适合持续集成和持续交付（CI / CD）工作流程。

请考虑以下示例方案：

- 您的开发人员在本地编写代码，并使用Docker容器与同事共享他们的工作。
- 他们使用Docker将其应用程序推送到测试环境中，并执行自动和手动测试。
- 当开发人员发现错误时，他们可以在开发环境中对其进行修复，然后将其重新部署到测试环境中以进行测试和验证。
- 测试完成后，将修补程序推送给生产环境就像将更新的映像推送到生产环境一样简单。

响应式部署和扩展

Docker基于容器的平台允许高度可移植的工作负载。Docker容器可以在开发人员的本地笔记本电脑上，数据中心中的物理或虚拟机上，云提供商上或混合环境中运行。

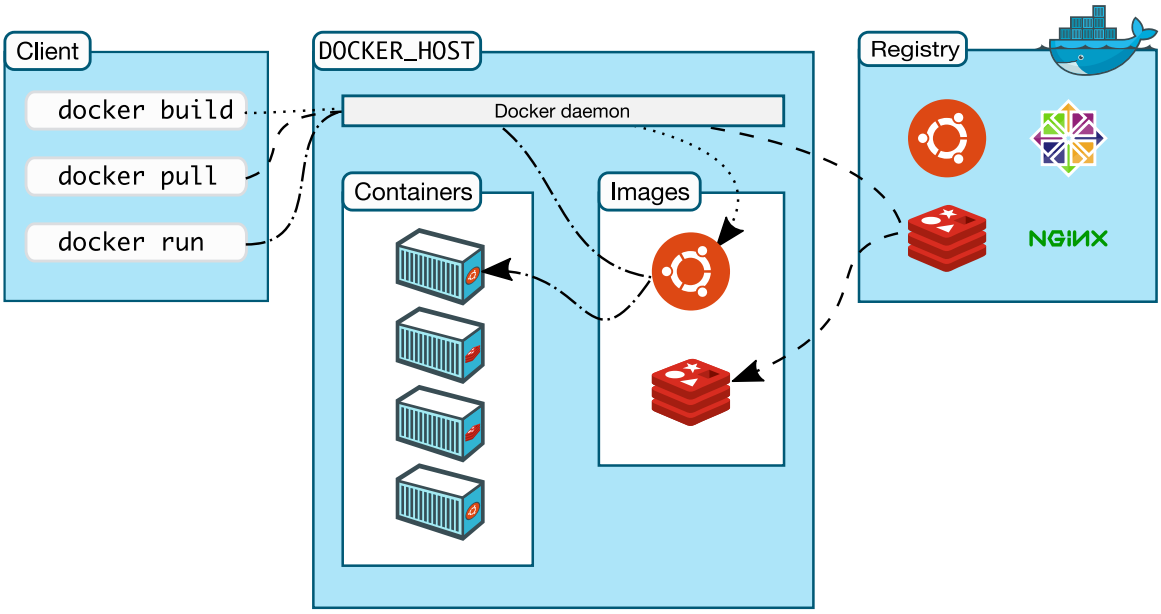
Docker的可移植性和轻量级的特性还使您可以轻松地动态管理工作负载，并根据业务需求指示实时扩展或关闭应用程序和服务。

在同一硬件上运行更多工作负载

Docker轻巧快速。它为基于虚拟机管理程序的虚拟机提供了可行且经济高效的替代方案，因此您可以利用更多的计算能力来实现业务目标。Docker非常适合高密度环境和中小型部署，而您需要用更少的资源做更多的事情。

Docker架构

Docker使用客户端-服务器架构。Docker *客户端*与Docker *守护进程*进行对话，该*守护进程*完成了构建，运行和分发Docker容器的繁重工作。Docker客户端和守护程序*可以*在同一系统上运行，或者您可以将Docker客户端连接到远程Docker守护程序。Docker客户端和守护程序在UNIX套接字或网络接口上使用REST API进行通信。



Docker守护程序

Docker守护程序（ `dockerd` ）侦听Docker API请求并管理Docker对象，例如镜像 容器，网络和卷。守护程序还可以与其他守护程序通信以管理Docker服务。

Docker客户端

Docker客户端（ `docker` ）是许多Docker用户与Docker交互的主要方式。当您使用诸如之类的命令时 `docker run` ，客户端会将这些命令发送到 `dockerd` ，以执行这些命令。该 `docker` 命令使用Docker API。Docker客户端可以与多个守护程序通信。

Docker注册表

Docker *注册表*存储Docker映像。Docker Hub是任何人都可以使用的公共注册表，并且Docker配置为默认在Docker Hub上查找映像。您甚至可以运行自己的私人注册表。如果使用Docker数据中心（DDC），则其中包括Docker可信注册表（DTR）。

使用 `docker pull` 或 `docker run` 命令时，所需的图像将从配置的注册表中提取。使用该 `docker push` 命令时，会将映像推送到配置的注册表。

Docker对象

使用Docker时，您正在创建和使用映像，容器，网络，卷，插件和其他对象。本节是其中一些对象的简要概述。

镜像

一个镜像是用于创建一个码头工人容器指令的只读模板。通常，一个映像*基于*另一个映像，并进行一些其他自定义。例如，您可以基于该 `ubuntu` 映像构建映像，但安装Apache Web服务器和您的应用程序，以及运行该应用程序所需的配置详细信息。

您可以创建自己的镜像，也可以仅使用其他人创建并在注册表中发布的镜像。要构建自己的映像，您可以 使用简单的语法创建一个*Dockerfile*，以定义创建映像并运行它所需的步骤。Dockerfile中的每条指令都会在映像中创建一个层。更改Dockerfile并重建映像时，仅重建那些已更改的层。与其他虚拟化技术相比，这是使映像如此轻巧，小型和快速的部分原因。

容器

容器是镜像的可运行实例。您可以使用Docker API或CLI创建，启动，停止，移动或删除容器。您可以将容器连接到一个或多个网络，将存储附加到该网络，甚至根据其当前状态创建新映像。

默认情况下，容器与其他容器及其主机之间的隔离程度相对较高。您可以控制容器的网络，存储或其他基础子系统与其他容器或与主机的隔离程度。

容器由其映像以及在创建或启动时为其提供的任何配置选项定义。删除容器后，未存储在持久性存储中的状态更改将消失。

示例 `docker run` 命令

以下命令运行一个 `ubuntu` 容器，以交互方式附加到本地命令行会话，然后运行 `/bin/bash` 。

```
$ docker run -i -t ubuntu /bin/bash
```

当您运行此命令时，会发生以下情况（假设您使用的是默认注册表配置）：

1. 如果您在 `ubuntu` 本地没有该映像，则Docker会将其从已配置的注册表中拉出，就像您已 `docker pull ubuntu` 手动运行一样。
2. Docker会创建一个新容器，就像您已 `docker container create` 手动运行命令一样。
3. Docker将一个读写文件系统分配给容器，作为其最后一层。这允许运行中的容器在其本地文件系统中创建或修改文件和目录。
4. Docker创建了一个网络接口以将容器连接到默认网络，因为您未指定任何网络选项。这包括为容器分配IP地址。默认情况下，容器可以使用主机的网络连接连接到外部网络。
5. Docker启动容器并执行 `/bin/bash` 。因为容器是交互式运行的，并且已附加到您的终端（由于 `-i` 和 `-t` 标志），所以您可以在输出记录到终端时使用键盘提供输入。

6. 当您键入 `exit` 以终止 `/bin/bash` 命令时，容器将停止但不会被删除。您可以重新启动或删除它。

服务

服务允许你扩展在多个Docker守护进程的容器，这是所有工作一起作为一个*群*有多个*管理人员*和*工人*。一个集群的每个成员都是一个Docker守护程序，并且所有守护程序都使用Docker API进行通信。服务允许您定义所需的状态，例如在任何给定时间必须可用的服务副本数。默认情况下，该服务在所有工作节点之间实现负载平衡。对于消费者而言，Docker服务似乎是一个单独的应用程序。Docker Engine在Docker 1.12及更高版本中支持集群模式。

底层技术

Docker用Go (https://golang.org/)编写，并利用Linux内核的多个功能来交付其功能。

命名空间

Docker使用一种称为 `namespaces` 提供*容器*的隔离工作区的技术。运行容器时，Docker会为该容器创建一组 *名称空间*。

这些名称空间提供了一层隔离。容器的每个方面都在单独的名称空间中运行，并且其访问仅限于该名称空间。

Docker Engine在Linux上使用以下名称空间：

- `pid` **命名空间**：进程隔离（PID：进程ID）。
- `net` **命名空间**：管理网络接口（NET：网络）。
- `ipc` **命名空间**：管理访问IPC资源（IPC：进程间通信）。
- `mnt` **命名空间**：管理文件系统挂载点（MNT：摩）。
- `uts` **命名空间**：隔离内核和版本标识符。（UTS：Unix时间共享系统）。

对照组

Linux上的Docker引擎还依赖于另一种称为*控制组*（ `cgroups` ）的技术。cgroup将应用程序限制为一组特定的资源。控制组允许Docker Engine将可用的硬件资源共享给容器，并有选择地实施限制和约束。例如，您可以限制特定容器可用的内存。

联合文件系统

联合文件系统或UnionFS是通过创建分层进行操作的文件系统，使其非常轻便且快速。Docker Engine使用UnionFS为容器提供构建模块。Docker Engine可以使用多个UnionFS变体，包括AUFS，btrfs，vfs和DeviceMapper。

容器格式

Docker Engine将名称空间，控制组和UnionFS组合到一个称为容器格式的包装器中。默认容器格式为 `libcontainer` 。将来，Docker可以通过与BSD Jails或Solaris Zones等技术集成来支持其他容器格式。