Zusammenfassung LB1

Container und Unterschiede zu Anderen

Container

Ein Container ist eine abgeschottete und isolierte Umgebung, in der eine Anwendung und ihre Abhängigkeiten ausgeführt werden können. Container verwenden Virtualisierungstechniken auf Betriebssystemebene, um Anwendungen zu isolieren und ihre Ausführung in einer konsistenten und portablen Weise zu ermöglichen.

Hier sind die technische Funktionsweise, der Sinn, die Vorteile und Nachteile von Containern:

Technische Funktionsweise:

Container nutzen Funktionen des Betriebssystems, wie z.B. cgroups und Namespaces, um Ressourcen zu isolieren und die Ausführungsumgebung abzuschotten.
Cgroups ermöglichen die Kontrolle und Begrenzung der Ressourcennutzung, wie CPU, Speicher und Netzwerkbandbreite.
Namespaces ermöglichen die Isolation von Prozessen, Dateisystemen, Netzwerkressourcen und Benutzerbereichen. Durch die Kombination dieser Techniken wird gewährleistet, dass Container in ihrer eigenen isolierten Umgebung ausgeführt werden, ohne mit anderen Containern oder dem Host-Betriebssystem zu interferieren.

Sinn von Containern:

Der Sinn von Containern liegt darin, Anwendungen und ihre Abhängigkeiten in einer konsistenten und portablen Weise zu verpacken und auszuführen.
Durch die Verwendung von Containern wird sichergestellt, dass eine Anwendung in verschiedenen Umgebungen, wie Entwicklungsrechnern, Testumgebungen oder Produktionsumgebungen, unabhängig von den spezifischen Konfigurationen und Infrastrukturen des Host-Systems, einheitlich ausgeführt wird. Container erleichtern die Bereitstellung von Anwendungen und fördern Skalierbarkeit und Portabilität.

Vorteile von Containern:

- Isolierung: Container bieten eine starke Isolation zwischen Anwendungen, wodurch Konflikte und Interferenzen vermieden werden. Jeder Container hat seine eigene isolierte Umgebung, in der er ausgeführt wird.
- Portabilität: Container sind portabel und können auf verschiedenen Systemen und Infrastrukturen ausgeführt werden, solange die Docker Engine oder ein Container-Laufzeitsystem vorhanden ist. Dies ermöglicht die einfache Bereitstellung und Skalierung von Anwendungen.
- Effizienz: Container sind leichtgewichtig und starten schnell, da sie den Kernel des Host-Betriebssystems nutzen und keine Emulation der Hardwareebene erfordern.
- Konsistenz: Da Container alle Abhängigkeiten und Konfigurationen in sich tragen, ermöglichen sie eine konsistente Ausführung von Anwendungen unabhängig von der Umgebung.

Nachteile von Containern:

- Begrenzte Isolation: Im Vergleich zu Virtual Machines bieten Container eine geringere Isolation auf der Hardwareebene. Wenn eine stärkere Isolation zwischen Anwendungen erforderlich ist, sind VMs möglicherweise die bessere Wahl.
- Begrenzte Unterstützung für bestimmte Betriebssysteme: Container basieren auf dem Host-Betriebssystem und unterstützen nicht alle Betriebssysteme gleichermaßen. Einige Betriebssysteme sind besser für die Containerisierung geeignet als andere.

Insgesamt bieten Container eine effiziente Möglichkeit, Anwendungen zu isolieren, zu verpacken und

---

Container vs VM

Container:

Ein Container ist eine isolierte Umgebung, die eine Anwendung und ihre Abhängigkeiten enthält. Container teilen sich den Kernel des

Host-Betriebssystems, sind aber in ihrer eigenen isolierten Umgebung eingeschlossen.
Sie sind leichtgewichtig und starten schnell, da sie auf dem Host-Betriebssystem direkt ausgeführt werden. Container sind portabel, da sie alle notwendigen Bibliotheken und Abhängigkeiten enthalten. Sie ermöglichen die konsistente Ausführung von Anwendungen in verschiedenen Umgebungen.

Virtual Machine (VM):

Eine Virtual Machine ist eine softwarebasierte Emulation eines physischen Computers. Sie ermöglicht es, mehrere Betriebssysteme und Anwendungen auf einer einzigen Hardwareplattform auszuführen. Eine VM emuliert die Hardware eines physischen Computers, einschließlich des Prozessors, des Speichers und der Speichergeräte.
Jede VM hat ihr eigenes Betriebssystem und führt Anwendungen innerhalb dieser virtuellen Umgebung aus. VMs sind isoliert und bieten eine hohe Flexibilität, da verschiedene Betriebssysteme gleichzeitig auf demselben physischen Server ausgeführt werden können.

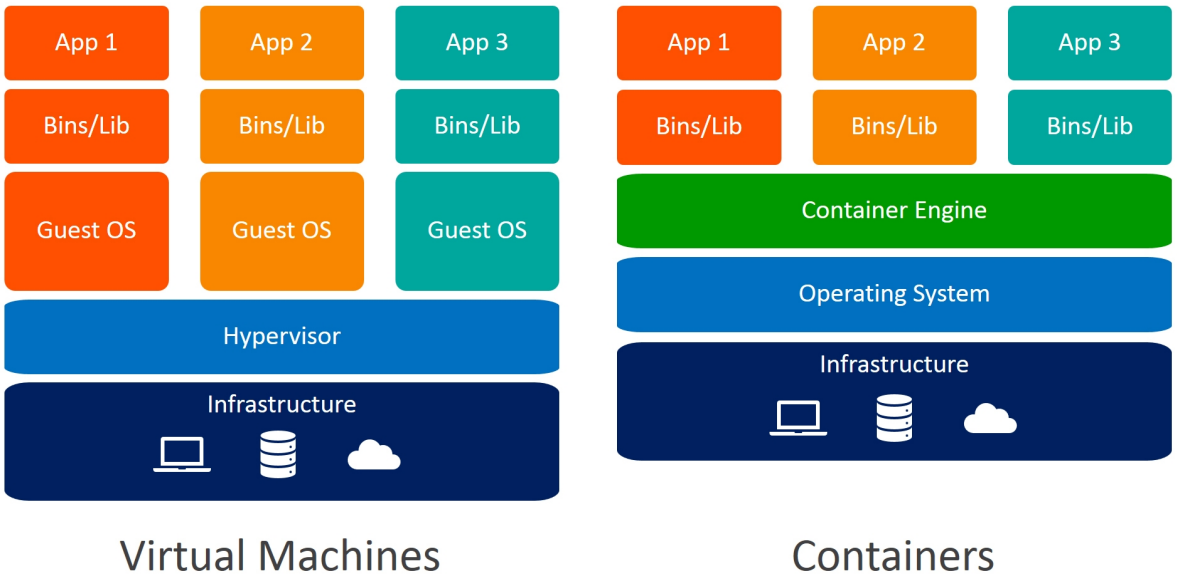
Unterschied

Der Hauptunterschied zwischen Containern und VMs liegt in der Art der Virtualisierung.

Container virtualisieren das Betriebssystem und ermöglichen eine effiziente Nutzung der Host-Ressourcen, da sie auf demselben Kernel laufen.

VMs hingegen emulieren eine vollständige Hardwareebene und ermöglichen die gleichzeitige Ausführung mehrerer Betriebssysteme, bieten jedoch eine geringere Effizienz aufgrund der erforderlichen Ressourcen für jede VM.

In Bezug auf Effizienz, Geschwindigkeit und Portabilität sind Container in der Regel leichtgewichtiger und schneller als VMs, da sie keine Emulation der Hardwareebene erfordern. VMs bieten jedoch eine stärkere Isolation zwischen den verschiedenen Betriebssystemen und sind oft besser geeignet, wenn es um die Virtualisierung von Anwendungen mit unterschiedlichen Betriebssystemanforderungen geht. Es gibt Situationen, in denen Container und VMs in Kombination eingesetzt werden, um die Vorteile beider Ansätze zu nutzen.



Container vs Container-Runtime

Ein Container ist eine abgeschottete und isolierte Ausführungsumgebung für eine Anwendung und ihre Abhängigkeiten.
Ein Container stellt eine Instanz eines Container-Images dar und enthält den ausgeführten Prozess sowie die Ressourcen, die er benötigt. Der Container isoliert die Anwendung von der Host-Umgebung und ermöglicht eine konsistente Ausführung unabhängig von der zugrunde liegenden Infrastruktur.

Eine Container-Runtime hingegen ist die Softwarekomponente, die für die Verwaltung und Ausführung von Containern verantwortlich ist. Sie bildet die Brücke zwischen dem Container-Image und dem Betriebssystem, indem sie das Image in eine ausführbare Containerinstanz umwandelt und diese innerhalb einer isolierten Umgebung ausführt.
Die Container-Runtime ermöglicht die Interaktion mit dem Betriebssystem-Kernel und verwendet Techniken wie Namespaces und cgroups, um die Isolierung und Ressourcenverwaltung der Container zu gewährleisten.

Um es zusammenzufassen: Der Container ist die isolierte Ausführungsumgebung für die Anwendung, während die Container-Runtime die Software ist, die für die Verwaltung, Ausführung und Isolierung der Container zuständig ist.

Beispiel:

Ein Beispiel für eine Container-Runtime ist Docker. Docker ermöglicht die Erstellung, Verwaltung und Ausführung von Containern auf einem Host-System. Es bietet die notwendige Infrastruktur, um Container-Images in ausführbare Container umzuwandeln und sie in einer isolierten Umgebung zu starten. Docker bietet auch Tools und Funktionen zur Verwaltung von Containern, wie das Erstellen von Images, das Verwalten von Netzwerken und Speicher, das Skalieren von Containern und das Bereitstellen von Anwendungen.

Container Runtime

Eine Container-Laufzeitumgebung, auch als Container-Runtime bezeichnet, ist eine Softwarekomponente, die für die Ausführung und Verwaltung von Containern verantwortlich ist. Sie stellt die Infrastruktur bereit, um Container-Images in ausführbare Containerinstanzen umzuwandeln und sie innerhalb einer isolierten Umgebung auszuführen.

Vorteile einer Container-Runtime:

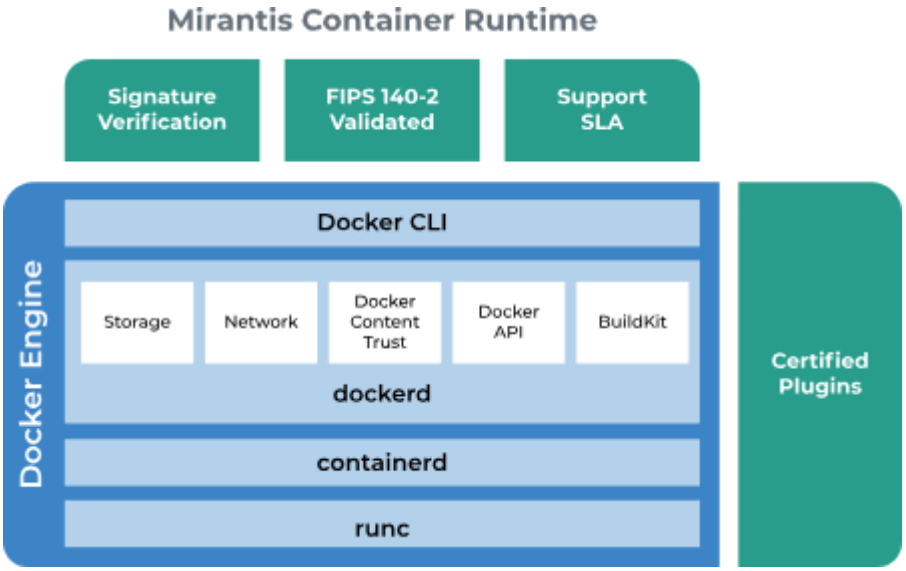
- Isolierung: Die Container-Runtime ermöglicht die strikte Isolierung von Containern, um Konflikte und Interferenzen zwischen verschiedenen Containern zu vermeiden.
- Ressourcenmanagement: Mit Hilfe von cgroups kann die Container-Runtime die Ressourcennutzung von Containern überwachen und steuern, um eine faire Verteilung der Ressourcen auf dem Host-System sicherzustellen.
- Skalierbarkeit: Eine Container-Runtime ermöglicht die einfache Skalierung von Containern, indem zusätzliche Instanzen gestartet werden, um die Arbeitslast zu bewältigen.
- Effizienz: Durch die direkte Interaktion mit dem Kernel und die Verwendung von Betriebssystemfunktionen ist die Container-Runtime in der Regel effizient und startet Container schnell.

Nachteile einer Container-Runtime:

- Abhängigkeit vom Host-Betriebssystem: Die Container-Runtime ist auf das Betriebssystem angewiesen, auf dem sie läuft. Dies bedeutet, dass sie nicht auf allen Plattformen und Betriebssystemen verfügbar oder vollständig kompatibel sein kann.
- Komplexität: Die Konfiguration und Verwaltung einer Container-Runtime erfordert oft fortgeschrittene Kenntnisse und kann komplex sein, insbesondere wenn es um erweiterte Funktionen wie Netzwerk- oder Speicherkonfiguration geht.

Verwendungszwecke einer Container-Runtime:

- Anwendungsbereitstellung: Container-Runtimes werden häufig für die Bereitstellung von Anwendungen in verschiedenen Umgebungen verwendet, sei es auf lokalen Rechnern, in der Cloud oder in Kubernetes-Clustern.
- Mikrodienste: Container-Runtimes ermöglichen die Ausführung einzelner Dienste als unabhängige Container, die in einem größeren Mikrodienstarchitektur eingesetzt werden können.
- Continuous Integration/Continuous Deployment (CI/CD): Container-Runtimes werden in CI/CD-Pipelines verwendet, um Anwendungen automatisch zu erstellen, zu testen und bereitzustellen.



Container-Images und Dockerfile

Container-Images

Ein Container-Image ist eine eigenständige, ausführbare Einheit, die alle Komponenten enthält, die zum Ausführen einer Anwendung benötigt werden, einschließlich des Betriebssystems, der Laufzeitumgebung, der Anwendungsabhängigkeiten und des Anwendungscodes.
Hier ist eine kurze Erklärung des Container-Images, seiner technischen Funktionsweise, der Vorteile, Nachteile, Verwendungszwecke und Beispiele für die Anwendung:

Technische Funktionsweise:

Ein Container-Image wird in Schichten aufgebaut.
Jede Schicht repräsentiert eine Veränderung am Dateisystem, wie das Hinzufügen, Ändern oder Entfernen von Dateien oder Anwendungen.
Die Schichten basieren auf einem Basissystem-Image und werden durch Anweisungen in einem sogenannten Dockerfile definiert. Bei der Erstellung eines Containers wird das Image verwendet, um eine isolierte Umgebung zu erzeugen, in der die Anwendung ausgeführt wird.

Vorteile von Container-Images:

- Portabilität: Container-Images sind in sich geschlossene Einheiten, die auf verschiedenen Plattformen und Systemen einheitlich ausgeführt werden können. Dadurch ist es einfach, Anwendungen zwischen verschiedenen Umgebungen zu verschieben.
- Effizienz: Container-Images sind leichtgewichtig, da sie den Host-Kernel nutzen und keine zusätzliche Emulationsebene benötigen. Dadurch sind sie schnell zu starten und verbrauchen weniger Ressourcen als traditionelle Virtual Machines.

- Skalierbarkeit: Durch die Verwendung von Container-Images können Anwendungen schnell und einfach horizontal skaliert werden, indem mehrere Instanzen der gleichen Image-Version gestartet werden.

- Konsistenz: Container-Images ermöglichen die Reproduzierbarkeit und Konsistenz von Anwendungen, da sie alle erforderlichen Abhängigkeiten und Konfigurationen enthalten.

Nachteile von Container-Images:

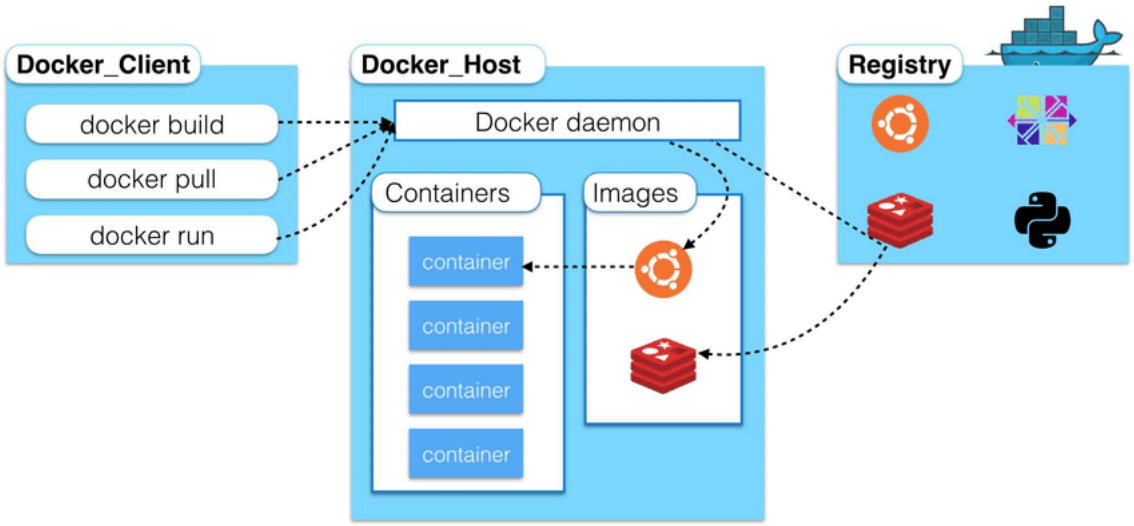
- Sicherheitsbedenken: Wenn Container-Images unsicher konfiguriert sind oder veraltete Komponenten enthalten, kann dies ein Sicherheitsrisiko darstellen.
- Größenzunahme: Container-Images können aufgrund der enthaltenen Abhängigkeiten und Schichten relativ groß sein, was zu längeren Download- und Bereitstellungszeiten führen kann.
- Abhängigkeitsmanagement: Die Verwaltung von Abhängigkeiten innerhalb von Container-Images erfordert eine sorgfältige Aktualisierung, um sicherzustellen, dass die Anwendung mit den neuesten Versionen arbeitet und Sicherheitslücken vermieden werden.

Verwendungszwecke von Container-Images:

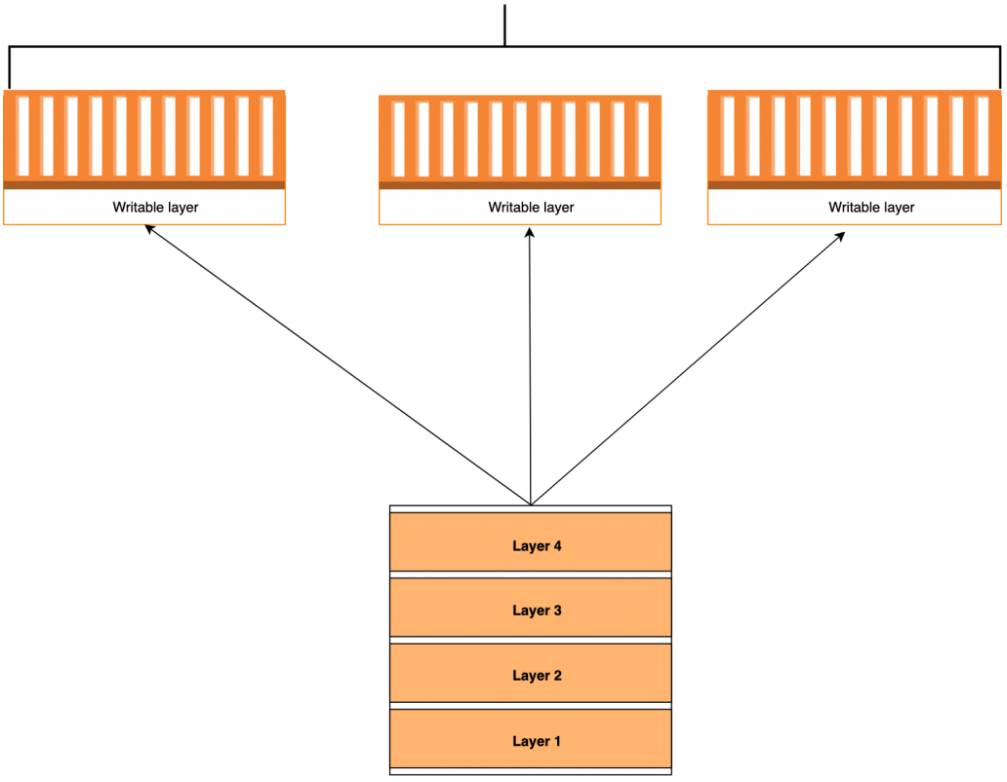
- Anwendungsbereitstellung: Container-Images ermöglichen eine einfache Bereitstellung von Anwendungen auf verschiedenen Umgebungen, wie Entwicklung, Test und Produktion.
- Mikrodienste: Container-Images werden häufig zur Bereitstellung einzelner Dienste verwendet, die zusammenarbeiten, um eine Anwendung zu bilden.
- Continuous Integration/Continuous Deployment (CI/CD): Container-Images werden in CI/CD-Pipelines verwendet, um eine automatisierte und konsistente Bereitstellung von Anwendungen zu ermöglichen.

Beispiele für Anwendungen von Container-Images:

- Docker: Docker ist eine beliebte Container-Plattform, die Docker-Images verwendet, um Container zu erstellen und auszuführen.



Docker Containers



Docker Image

Dockerfile

Ein Dockerfile ist eine Textdatei, die verwendet wird, um ein Docker-Image zu definieren und zu erstellen. Es enthält eine Reihe von Anweisungen, die beschreiben, wie das Image aufgebaut werden soll. Hier ist eine Erklärung, wie ein Dockerfile funktioniert und wie es verwendet wird:

Funktionsweise:

Ein Dockerfile besteht aus einer Sequenz von Befehlen, die Schicht für Schicht ausgeführt werden, um das gewünschte Docker-Image zu erstellen. Jeder Befehl in einem Dockerfile erzeugt eine neue Schicht im Image. Die Anweisungen können beispielsweise das Hinzufügen von Dateien, das Ausführen von Befehlen, das Festlegen von Umgebungsvariablen oder das Definieren von Netzwerkeinstellungen umfassen.

Verwendung:

Um ein Docker-Image mit einem Dockerfile zu erstellen, wird der Befehl `docker build` verwendet und das Dockerfile als Argument übergeben. Der Build-Prozess analysiert das Dockerfile Schritt für Schritt und erzeugt eine neue Schicht für jede Anweisung. Das resultierende Image kann dann mit dem Befehl `docker run` gestartet werden, um einen Container zu erstellen und auszuführen.

Beispiele:
Ein einfaches Beispiel für ein Dockerfile könnte wie folgt aussehen:

```
# Verwenden eines Basisimages
FROM ubuntu:latest

# Aktualisieren des Paketmanagers und Installieren von Abhängigkeiten
RUN apt-get update && apt-get install -y \
    python3 \
    python3-pip

# Kopieren der Anwendungsdateien in das Image
COPY . /app

# Festlegen des Arbeitsverzeichnisses
WORKDIR /app

# Installation der Python-Abhängigkeiten
```

```
RUN pip3 install -r requirements.txt

# Definieren des Ausführungsbefehls
CMD ["python3", "app.py"]
```

In diesem Beispiel wird ein Docker-Image basierend auf dem neuesten Ubuntu-Image erstellt. Es werden Python und Pip installiert, die Anwendungsdateien werden in das Image kopiert, die Abhängigkeiten werden installiert, das Arbeitsverzeichnis wird festgelegt und der Befehl zum Ausführen der Anwendung wird definiert.

Mit diesem Dockerfile können Sie dann das Image erstellen und einen Container basierend auf diesem Image ausführen.

Aufbau Container-Image

Ein Container-Image besteht aus mehreren Komponenten, die zusammen eine vollständige und isolierte Ausführungsumgebung für eine Anwendung bieten. Hier sind die Hauptbestandteile eines Container-Images:

1. Basisimage:

Das Basisimage bildet den Ausgangspunkt für das Container-Image. Es enthält das Betriebssystem und die grundlegenden Systemkomponenten, auf denen die Anwendung ausgeführt werden soll. Beispiele für Basisimages sind Ubuntu, Alpine Linux oder CentOS.

2. Laufzeitumgebung:

Die Laufzeitumgebung umfasst die notwendigen Komponenten, um den Code der Anwendung auszuführen. Dies kann eine Programmiersprache wie Python oder Node.js sein, aber auch andere Frameworks, Bibliotheken oder Laufzeitumgebungen, die für die Anwendung erforderlich sind.

3. Abhängigkeiten:

Abhängigkeiten sind externe Module, Bibliotheken oder Pakete, die von der Anwendung benötigt werden, um ordnungsgemäß zu funktionieren. Sie können spezifische Versionen von Bibliotheken oder benutzerdefinierte Pakete umfassen, die für die Anwendung erforderlich sind.

4. Anwendungscode:

Der Anwendungscode ist der eigentliche Code Ihrer Anwendung, der in den Container eingefügt wird. Dies kann ein einzelnes Skript oder eine umfangreichere Anwendungsstruktur mit mehreren Dateien und Ordnern sein.

5. Konfigurationsdateien:

Konfigurationsdateien enthalten Einstellungen und Parameter, die für den Betrieb und die Konfiguration der Anwendung erforderlich sind. Dies können Umgebungsvariablen, Netzwerkkonfigurationen, Datenbankverbindungen oder andere anwendungsspezifische Konfigurationen sein.

Zusätzlich zu diesen Hauptbestandteilen kann ein Container-Image auch andere Komponenten enthalten, wie z. B. Datenbanken, Dateien oder Skripte, die zur Einrichtung oder Initialisierung der Anwendung verwendet werden.

Es ist wichtig zu beachten, dass ein Container-Image schreibgeschützt ist und nicht verändert werden kann. Stattdessen wird für jede Instanz des Containers eine schreibbare Schicht (Container Layer) hinzugefügt, um das Speichern von Daten oder das Verfolgen von Änderungen während der Laufzeit zu ermöglichen. Dadurch bleibt das ursprüngliche Image unverändert und kann für weitere Instanzen oder Aktualisierungen verwendet werden.

Host / Hypervisor / VW

Erklärung der Begriffe

1. Host:

Der Host, auch als Host-Maschine oder Host-System bezeichnet, bezieht sich auf den physischen Computer oder das Server-System, auf dem Virtualisierungstechnologien eingesetzt werden. Der Host stellt die Hardware und Ressourcen bereit, um virtuelle Maschinen (VMs) auszuführen. Er kann sowohl ein Desktop-Computer als auch ein Server sein und enthält die CPU, den Speicher, die Festplatten und andere Komponenten, die zur Unterstützung der Virtualisierung benötigt werden.

2. VM (Virtual Machine):

Eine virtuelle Maschine ist eine vollständig isolierte und eigenständige Umgebung, die auf einem physischen Host erstellt wird. Eine VM bildet eine virtuelle Repräsentation eines Computersystems mit einem Betriebssystem (OS) und den entsprechenden Anwendungen. Sie ermöglicht es, mehrere unabhängige Betriebssysteme auf einem einzigen physischen Host auszuführen. Jede VM hat ihre eigene virtuelle Hardware, einschließlich CPU, Speicher, Netzwerkschnittstellen und Festplatten, die von der zugrunde liegenden Hardware des Hosts abstrahiert sind.

3. Hypervisor:

Ein Hypervisor, auch als Virtual Machine Monitor (VMM) bezeichnet, ist eine Software- oder Hardwarekomponente, die für die Verwaltung und Bereitstellung von virtuellen Maschinen auf dem Host-System zuständig ist. Der Hypervisor erstellt und verwaltet die virtuellen Umgebungen der VMs, indem er die physischen Ressourcen des Hosts in virtuelle Ressourcen für jede VM aufteilt. Es gibt zwei Haupttypen von Hypervisoren:

- Typ-1-Hypervisor (Bare-Metal-Hypervisor): Diese Hypervisoren werden direkt auf dem physischen Host-System installiert und haben direkten Zugriff auf die Hardware-Ressourcen. Sie bieten eine höhere Leistung und Effizienz, da sie keine zusätzliche Betriebssystemebene zwischen dem Hypervisor und der Hardware

haben. Beispiele für Typ-1-Hypervisoren sind VMware ESXi, Microsoft Hyper-V und Xen.

- Typ-2-Hypervisor (Hosted-Hypervisor): Diese Hypervisoren werden als Anwendungen innerhalb eines Betriebssystems installiert. Das Betriebssystem des Hosts verwaltet die Hardware-Ressourcen, während der Hypervisor als zusätzliche Softwareebene fungiert, um die VMs zu erstellen und zu verwalten.

Typ-2-Hypervisoren sind einfacher zu implementieren und zu verwenden, bieten jedoch in der Regel eine geringere Leistung im Vergleich zu Typ-1-Hypervisoren. Ein bekanntes Beispiel für Typ-2-Hypervisor ist Oracle VirtualBox.

Die technische Funktionsweise des Hypervisors besteht darin, die Hardware-Ressourcen des Hosts zu virtualisieren und sie den VMs zuzuweisen. Der Hypervisor übernimmt die Aufgabe der Ressourcenverwaltung, Planung, Isolierung und Kontrolle der VMs, um sicherzustellen, dass sie unabhängig voneinander und ohne Beeinträchtigung arbeiten können.
Er ermöglicht den sicheren Betrieb mehrerer Betriebssysteme und Anwendungen auf einem einzigen physischen Host.

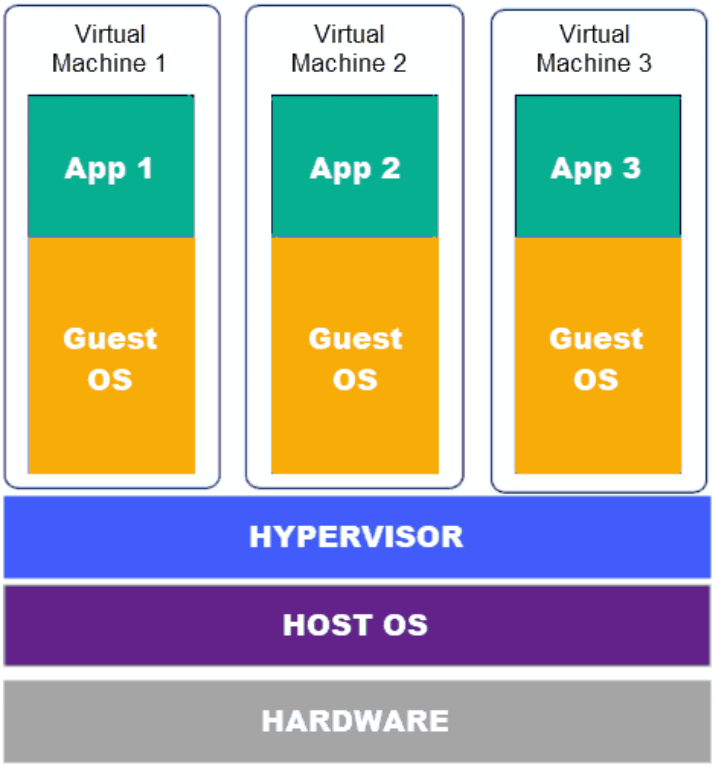


Image-Repository

Erklärung

Ein Image-Repository ist ein Speicherort, in dem Container-Images gespeichert, verwaltet und geteilt werden können. Es dient als zentraler Ort für die Speicherung und Verteilung von Container-Images, sodass sie von anderen Entwicklern, Teams oder Systemen verwendet werden können.

Unterschied zwischen privat und öffentlichem Image-Repository:

- Öffentliches Image-Repository: Ein öffentliches Image-Repository ist für jedermann zugänglich und ermöglicht es Benutzern, Container-Images frei zu durchsuchen, herunterzuladen und zu verwenden.

Beliebte öffentliche Image-Repositorys sind Docker Hub, das von der Docker-Community betrieben wird, oder der Container Registry Service von Cloud-Anbietern wie Google Cloud Platform oder Amazon Web Services.

Öffentliche Repositories bieten eine große Auswahl an vorgefertigten Images, die von der Community oder von Entwicklern und Organisationen bereitgestellt werden. Sie sind nützlich, um beliebte Anwendungen oder Frameworks schnell zu finden und zu verwenden.

- Privates Image-Repository: Ein privates Image-Repository ist auf einen bestimmten Benutzer oder eine bestimmte Organisation beschränkt und erfordert Zugriffsberechtigungen, um auf die Images zugreifen zu können.

Es ermöglicht die private Speicherung und Verwaltung von Container-Images, die nur intern oder von bestimmten Teammitgliedern verwendet werden sollen. Private Repositories bieten eine erhöhte Kontrolle über die Zugriffsrechte und ermöglichen die Verwaltung vertraulicher oder proprietärer Images.

Organisationen können ihre eigenen internen Repositories einrichten, um ihre spezifischen Anwendungs Images zu verwalten.

Funktionsweise:

Ein Image-Repository fungiert als zentraler Speicherort für Container-Images. Entwickler können Images hochladen, verwalten und teilen, während andere Benutzer auf diese Images zugreifen und sie für die Erstellung und Ausführung von

Containern verwenden können. Die Funktionsweise eines Image-Repositorys kann je nach Plattform variieren, aber im Allgemeinen umfasst sie folgende Schritte:

1. Hochladen:

Entwickler erstellen Container-Images entweder manuell oder automatisch über einen Build-Prozess. Diese Images werden dann in das Repository hochgeladen, entweder über eine Befehlszeilenschnittstelle oder eine grafische Benutzeroberfläche.

2. Speicherung und Verwaltung:

Das Repository speichert die hochgeladenen Images und verwaltet sie in einer organisierten Struktur. Es ermöglicht die Versionierung von Images, das Hinzufügen von Metadaten und Tags zur Identifizierung und Organisation der Images.

3. Freigabe und Zugriff:

Abhängig von den Zugriffsberechtigungen kann das Repository den Zugriff auf die Images für bestimmte Benutzer, Teams oder Systeme einschränken oder freigeben. Andere Benutzer können dann die Images suchen, anzeigen und herunterladen.

Vorteile von Image-Repositorys:
- Zentrale Speicherung: Ein Repository bietet einen zentralen Speicherort für Container-Images, was die Verwaltung und gemeinsame Nutzung erleichtert.

- Wiederverwendbarkeit: Entwickler können auf vorgefertigte Images zugreifen und sie in ihren eigenen Projekten wiederverwenden, um Zeit und Aufwand zu

---

Cloud Native

Erklärung

Cloud Native bezieht sich auf eine moderne Ansatzweise bei der Entwicklung und Bereitstellung von Anwendungen, die speziell für die Cloud-Umgebung konzipiert

ist. Hier sind Erklärungen zur Funktionsweise, den Vorteilen, Nachteilen und Anwendungsbereichen von Cloud Native:

Funktionsweise:

Cloud Native-Anwendungen werden mit bestimmten Prinzipien und Technologien entwickelt, die es ermöglichen, in Cloud-Umgebungen effizient zu arbeiten. Die wichtigsten Merkmale und Funktionen von Cloud Native sind:

1. Containerisierung: Cloud Native-Anwendungen werden in Containern bereitgestellt. Container bieten eine leichte, portable und konsistente Umgebung für den Anwendungscode, die Abhängigkeiten und die Konfiguration. Container ermöglichen eine schnelle Bereitstellung, Skalierung und Isolierung von Anwendungen.

2. Microservices: Cloud Native-Anwendungen werden als eine Sammlung von kleinen, unabhängigen Diensten entwickelt, die jeweils eine spezifische Funktionalität erfüllen. Diese Microservices kommunizieren miteinander über standardisierte APIs. Die Verwendung von Microservices ermöglicht eine bessere Skalierbarkeit, Wartbarkeit und Flexibilität.

3. Automatisierung und Orchestrierung: Cloud Native setzt auf Automatisierung und Orchestrierung, um eine effiziente Bereitstellung und Verwaltung von Anwendungen zu ermöglichen. Tools wie Kubernetes werden verwendet, um das Management von Containern, Skalierung, Lastenausgleich und das Failover von Anwendungen zu automatisieren.

4. Skalierbarkeit und Elastizität: Cloud Native-Anwendungen können einfach und dynamisch in der Größe angepasst werden, um auf schwankende Lasten zu reagieren. Sie können schnell hoch- oder herunterskaliert werden, um eine effiziente Ressourcennutzung und eine gute Benutzererfahrung zu gewährleisten.

Vorteile von Cloud Native:

- Skalierbarkeit: Cloud Native-Anwendungen können problemlos in der Größe angepasst werden, um mit wechselnden Workload-Anforderungen umzugehen.

- Flexibilität: Cloud Native ermöglicht eine schnellere Bereitstellung neuer Funktionen und Updates, was die Markteinführungszeit verkürzt.

- Effiziente Ressourcennutzung: Durch die Containerisierung und Automatisierung können Ressourcen effizienter genutzt werden, was Kosten reduziert.

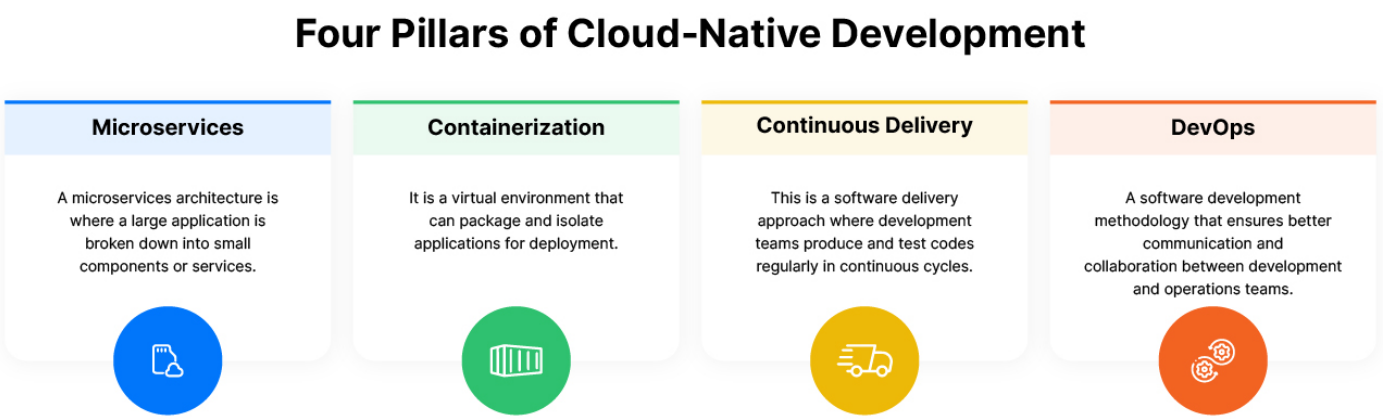
- Ausfallsicherheit: Die Orchestrierung und Verteilung von Microservices in Cloud Native-Anwendungen erhöhen die Ausfallsicherheit und ermöglichen das Failover von Diensten.

Nachteile von Cloud Native:

- Komplexität: Der Einsatz von Cloud Native erfordert ein Verständnis für Containerisierung, Orchestrierung und andere Technologien, was zu einer steileren Lernkurve führen kann.

- Infrastrukturabhängigkeit: Cloud Native-Anwendungen sind stark von der zugrunde liegenden Cloud-Infrastruktur abhängig. Eine fehlerhafte oder instabile Infrastruktur kann sich auf die Anwendungsleistung auswirken.

- Erhöhter Entwicklungs- und Wartungsaufwand: Der Umstieg auf Cloud Native erfordert Anpassungen an best



Was ist Docker

Docker Container

“Docker Container sind eine Art light-Version von virtuellen Maschinen.

Während virtuelle Maschinen die Hardware eines Rechners, also den Prozessor, die Festplatte, usw. virtualisieren (Hypervisor), virtualisieren Docker Container das Betriebssystem. Dies wird auch Container Virtualisierung genannt. Aber was bedeutet das?

Container Virtualisierung bedeutet, dass basierend auf einem auf der Maschine laufenden Host-Betriebssystems (z.B. Ubuntu) weitere Linux Distributionen wie CentOS, Debian oder auch Ubuntu parallel betrieben werden können. Der Trick: "Es wird der Kern des Host-Betriebssystems (der Kernel) mit den Containern geteilt.”

Die Theorie bleibt hier also gleich wie im oberen Abschnitt. Jedoch gibt es einen grundlegenden Unterschied. Während Container die grundsätzliche und allgemeine Anwendung sind,ist ein Docker-Container etwas, was auf einem Image basiert.

Erklärung

Ein Container ist eine standardisierte, isolierte Umgebung, die eine Anwendung und ihre Abhängigkeiten umfasst. Container ermöglichen die Kapselung einer Anwendung mit allen erforderlichen Komponenten wie dem Betriebssystem, Bibliotheken und Konfigurationsdateien.
Container sind portabel und können auf verschiedenen Betriebssystemen und Infrastrukturen ausgeführt werden, solange sie von einer Container-Runtime unterstützt werden.

Docker ist eine der bekanntesten Implementierungen von Container-Technologien. Docker bietet eine Plattform, die das Erstellen, Bereitstellen und Verwalten von Containern vereinfacht.
Docker verwendet eine Kombination aus Linux-Container-Technologien wie cgroups und namespaces, um eine effiziente und isolierte Ausführungsumgebung für Container bereitzustellen. Docker bietet auch Tools und APIs, die die Arbeit mit Containern erleichtern.

Der Begriff "Docker-Container" wird oft verwendet, um sich auf Container zu beziehen, die mit Docker-Technologien erstellt und verwaltet werden. Ein

Docker-Container basiert auf einem Docker-Image, das die erforderlichen Komponenten und Einstellungen für den Container enthält.
Docker ermöglicht das Erstellen, Bereitstellen und Verwalten von Containern über Befehle und Konfigurationsdateien, die in Dockerfiles definiert sind.

Es ist wichtig zu beachten, dass Container-Technologien nicht auf Docker beschränkt sind. Es gibt auch andere Container-Runtimes und -Plattformen wie Kubernetes, die Container-Orchestrierung und -Management ermöglichen. Docker hat jedoch aufgrund seiner Benutzerfreundlichkeit und weiten Verbreitung die Container-Technologien populär gemacht und ist oft die erste Wahl für Entwickler und Unternehmen, die mit Containern arbeiten möchten.

Docker im Grundsatz

Quelle: Unterrichtsmatrial

Vorteile eines Docker Containers?

Der wesentliche Vorteil von Containern ist es, dass diese sowohl deutlich weniger Speicher benötigen, als auch deutlich schneller laufen wie VMs. Man muss für das Hochfahren eines neuen Containers nicht ein komplettes VM-Image mit mehreren GB laden, sondern kann mit abgespeckten Linux Distributionen wie Alpine Linux bereits mit wenigen MB starten.

Container können sowohl lokal auf einem Laptop mit der Anwendung Docker gemanaged werden, als auch in der Cloud betrieben werden.

Doch müssen Sie nicht von 0 weg starten. Es gibt ein globales Verzeichnis namens Docker Hub (Repository) wo bereits fertige Container-Images mit den unterschiedlichsten Applikationen wie Webservern, Datenbanken, usw. bereits verfügbar sind und mit einem Klick bei Ihnen gestartet werden können.

Docker Container stellen somit eine schnelle und kostengünstige Variante dar, die unterschiedlichsten Web- und Offline-Anwendungen in Bereichen wie der Logistik zu betreiben. Diese können innerhalb von Sekunden gestartet, gestoppt und auch wieder gelöscht werden.

Was sind Images?

Der Begriff „Image“ ist Ihnen im Zusammenhang mit Virtualisierung möglicherweise von virtuellen Maschinen (VMs) bekannt. Für gewöhnlich handelt es sich bei einem VM-Image um eine Kopie eines Betriebssystems. Ggf. enthält ein

VM-Image weitere installierte Komponenten wie Datenbank und Webserver. Der Begriff entstammt einer Zeit, in der Software auf optischen Datenträgern wie CD-ROMs und DVDs verteilt wurde. Wollte man eine lokale Kopie des Datenträgers anlegen, erstellte man mit einer speziellen Software ein Abbild, auf Englisch „Image“.

Bei der Container-Virtualisierung handelt es sich um die konsequente Weiterentwicklung der VM-Virtualisierung. Anstatt einen virtuellen Computer (Maschine) mit eigenem Betriebssystem zu virtualisieren, umfasst ein Docker-Image in der Regel lediglich eine Anwendung. Dabei kann es sich um eine einzelne Binärdatei handeln oder um einen Verbund mehrerer Software-Komponenten.

Um die Anwendung auszuführen, wird aus dem Image zunächst ein Container erzeugt. Alle auf einem Docker-Host laufenden Container greifen auf denselben Betriebssystem-Kernel zurück. Dadurch sind Docker-Container und Docker-Images in der Regel deutlich leichtgewichtiger als vergleichbare virtuelle Maschinen und deren Images.

Die Konzepte Docker-Container und Docker-Image sind eng miteinander verknüpft. So kann nicht nur ein Docker-Container aus einem Docker-Image erzeugt werden, sondern auch aus einem laufenden Container ein neues Image.

Weitere Begriffe aus der container Technologie

docker deamon

Der Docker Deamon ist ein dauerhafter Hintergrundprozess, der Docker-Images, Container, Netzwerke und Speichervolumen verwaltet. Er „schaut“ ständig auf Docker-API-Anforderungen und verarbeitet sie.

docker client

Der Docker-Client ermöglicht den Benutzern die Interaktion mit Docker. Er kann sich auf demselben Host wie der Daemon befinden oder eine Verbindung zu einem Daemon auf einem entfernten Host herstellen und mit mehr als einem Daemon kommunizieren. Der Docker-Client bietet eine Befehlszeilenschnittstelle (CLI = Command Line Interpreter), über die Sie einem Docker-Daemon Befehle zum Erstellen, Ausführen und Anhalten von Anwendungen erteilen können.

Der Hauptzweck des Docker-Clients besteht darin, ein Mittel zur Verfügung zu stellen, mit dem Images aus einer Registry gezogen und auf einem Docker-Host ausgeführt werden können. Befehle werden unter docker command line interface erklärt.

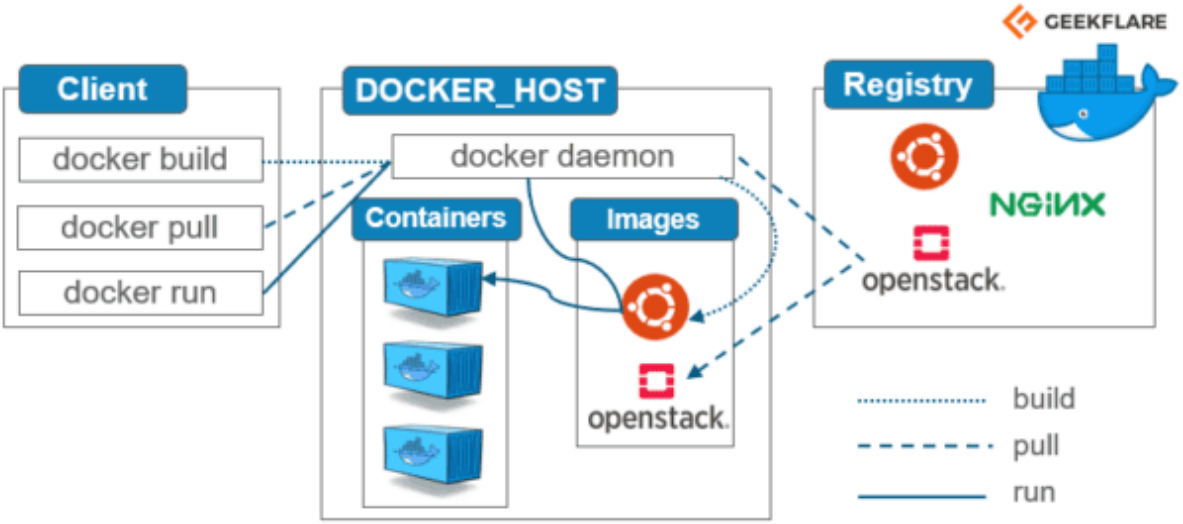
Docker Registry

In Docker Registries werden Images versioniert abgelegt und verteilt.

Die Standard-Registry ist der **Docker Hub**, auf dem tausende öffentlich verfügbarer Images zur Verfügung stehen, aber auch "offizielle" Images.

Viele Organisationen und Firmen nutzen eigene Registries, um kommerzielle oder "private" Images zu hosten, aber auch um den Overhead zu vermeiden, der mit dem Herunterladen von Images über das Internet einhergeht. Die Nutzung eigener Registries (private Registries) kann auch aus Sicherheitsüberlegungen geschehen. Die Details entnehmen Sie der entsprechenden Theorie.

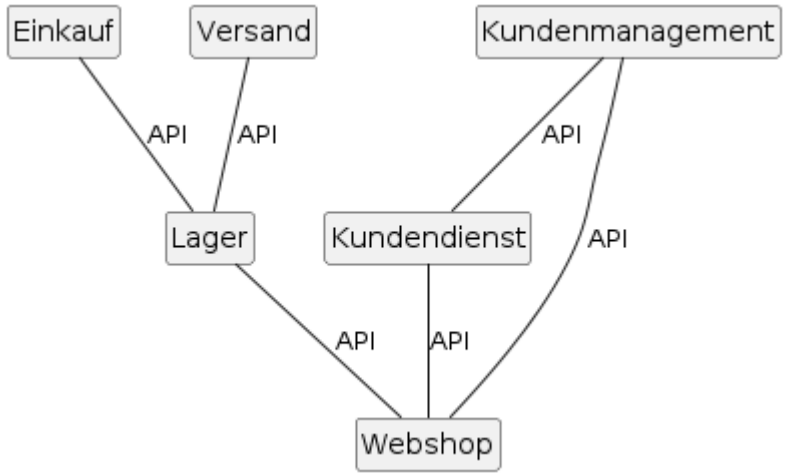
Weiterführende Informationen zu container registries erhalten Sie im Verlaufe des Moduls.



Full Stack Journey mit Docker Compose

Microservice Architekturmuster

Microservices sind ein Ansatz in der Softwareentwicklung, bei dem Software aus kleinen unabhängigen Services besteht, die über sorgfältig definierte Schnittstellen (APIs) kommunizieren.



Eigenschaften von Microservices

- *Eigenständig*: Jeder Service in einer Microservice Architektur kann entwickelt, bereitgestellt betrieben und skaliert werden, ohne die anderen Services zu beeinträchtigen.
- *Spezialisierung*: Jeder Service konzentriert sich auf die Lösung eines bestimmten Problems. Entsprechend kann ein dediziertes Team von Fachspezialisten einen Service kontinuierlich weiterentwickeln.

REST Schnittstellen (API)

REST steht für "Representational State Transfer". Diese Art von Schnittstelle zwischen Services bietet sich besonders bei Microservice Architekturen an. Sie ermöglicht es einzelne Services während dem Betrieb auszutauschen.

Folgende Kriterien zeichnen RESTful APIs aus:

- Eine aus Clients, Servern und Resourcen bestehende Client/Server-Architektur, die Anfragen über HTTP verwaltet.

- Zustandlose Client/Server-Kommunikation, d.h. zwischen GET-Anforderungen werden keine Client-Informationen gespeichert, und die einzelnen Anforderungen sind separat und nicht verbunden.

Quellen: RedHat

Hot Reload

Während der Entwicklung ist es sehr mühsam, wenn man bei jeder Änderung den Container neu erstellen und starten muss. Deshalb haben die meisten Frameworks eine Möglichkeit zum Hot Reload eingeführt.

Hot Reload ist ein Entwicklungsverfahren, bei dem Code Änderungen in Echtzeit angewendet werden, während eine Anwendung läuft, ohne dass die Anwendung neu gestartet werden muss. Es ermöglicht den Entwicklern, den Code dynamisch zu ändern und die Auswirkungen sofort zu sehen, ohne den gesamten Entwicklungszyklus zu durchlaufen.

Die Funktionsweise von Hot Reload variiert je nach Entwicklungsplattform und Framework, aber im Allgemeinen umfasst es folgende Schritte:

1. Überwachung des Quellcodes:

Das Hot Reload-System überwacht die Quellcodedateien der Anwendung auf Änderungen. Es kann entweder kontinuierlich überwachen oder auf Anforderung des Entwicklers aktiviert werden.

2. Erkennung von Änderungen:

Wenn eine Änderung in einer Quellcodedatei festgestellt wird, wird sie vom Hot Reload-System erkannt.

3. Aktualisierung des Codes:

Das Hot Reload-System lädt den geänderten Code in den Speicher der laufenden Anwendung und ersetzt den entsprechenden Teil des Codes. Dies geschieht in der Regel auf Modulebene oder auf granulärer Ebene, sodass nur die geänderten Teile

des Codes neu geladen werden, während der Rest der Anwendung unverändert bleibt.

4. Anwendungsaktualisierung:

Nach dem Aktualisieren des Codes führt die Anwendung die aktualisierten Codeabschnitte aus, wodurch die Änderungen sofort wirksam werden. Der Zustand der Anwendung kann beibehalten werden, um eine nahtlose Fortsetzung des Entwicklungsprozesses zu ermöglichen.

Hot Reload wird in der Regel in der Softwareentwicklung für eine schnellere Iteration und eine verbesserte Entwicklererfahrung eingesetzt. Einige der Anwendungsfälle von Hot Reload sind:

1. Schnelle Entwicklung und Test:
Hot Reload ermöglicht es Entwicklern, Codeänderungen sofort zu überprüfen, während sie an einer Anwendung arbeiten. Dadurch können Fehler schneller erkannt und behoben werden, wodurch die Entwicklungszeit verkürzt wird.

2. UI-Entwicklung: Bei der Entwicklung von Benutzeroberflächen können Entwickler mit Hot Reload Änderungen in Echtzeit sehen, ohne die Anwendung neu laden zu müssen. Dies erleichtert die Anpassung des Layouts, des Designs und der Interaktionen.

3. Prototyping: Hot Reload ist auch nützlich beim Prototyping von Anwendungen, da es den Entwicklern ermöglicht, Ideen schnell umzusetzen und Änderungen vorzunehmen, um das gewünschte Ergebnis zu erreichen.

Vorteile von Hot Reload:

- Schnellere Iteration: Durch die sofortige Anwendung von Codeänderungen ermöglicht Hot Reload eine schnellere Iteration im Entwicklungsprozess.
- Verbesserte Produktivität: Entwickler können Änderungen testen und Probleme schneller beheben, ohne den Arbeitsfluss zu unterbrechen oder Zeit mit dem erneuten Starten der Anwendung zu verschwenden.
- Bessere Entwicklererfahrung: Hot Reload bietet eine reibungslose und nahtlose Entwicklungserfahrung, da Entwickler den Code in Echtzeit ändern und die Auswirkungen sofort sehen können.

Nachteile von Hot Reload:

- Einschränkungen bei bestimmten Änderungen: Bei komplexeren Änderungen, die tiefgreifende Struktur- oder Framework-Änderungen erfordern, kann Hot Reload möglicherweise nicht angewendet werden und ein vollständiger Neustart der Anwendung ist erforderlich.
- Performance-Einbußen: Das Hot Reload-Verfahren erfordert zusätzliche Ressourcen und Overhead, was zu leichten Performance-Einbußen führen kann.

Insgesamt ist Hot Reload ein leistungsstarkes Werkzeug für Entwickler, um ihre Produktivität zu steigern und den Entwicklungsprozess zu verbessern, indem sie Codeänderungen in Echtzeit anwenden können.

Daten Persistierung und Volumen

Sinn und Erklärung Datenpersistierung

In der Welt der Containerisierung werden Container normalerweise als ephemere Einheiten betrachtet, was bedeutet, dass sie dazu gedacht sind, transient und leichtgewichtig zu sein. Dies bedeutet, dass Container standardmäßig keinen dauerhaften Speicher haben und ihre Daten beim Neustart oder Beenden des Containers verloren gehen können.

In einigen Fällen ist es jedoch erforderlich, Daten über den Lebenszyklus eines Containers hinaus zu speichern und persistent zu machen. Hier kommen Volumen ins Spiel. Volumen sind Mechanismen, mit denen Daten in einem Container persistiert und zwischen verschiedenen Containern geteilt werden können.

Die Verwendung von Volumen bietet mehrere Vorteile:

1. Persistenz: Volumen ermöglichen es, Daten dauerhaft zu speichern, selbst wenn der Container beendet oder neu gestartet wird. Dadurch bleiben die Daten intakt und können von anderen Containern oder Services weiterhin genutzt werden.

2. Datenfreigabe: Volumen können zwischen verschiedenen Containern oder sogar über verschiedene Hosts hinweg gemeinsam genutzt werden. Dies ermöglicht eine effiziente Datenfreigabe und Zusammenarbeit zwischen Anwendungen oder Services.

3. Datensicherheit: Durch die Verwendung von Volumen können wichtige Daten außerhalb des Containers gespeichert werden. Dies bietet eine zusätzliche Sicherheitsebene, da die Daten nicht verloren gehen, selbst wenn der Container fehlschlägt oder beschädigt wird.

Welche Daten persistiert und in Volumen abgelegt werden sollten, hängt von der spezifischen Anwendung oder dem spezifischen Use Case ab. Hier sind einige Beispiele:

- Konfigurationsdateien: Volumen können verwendet werden, um Konfigurationsdateien bereitzustellen, die von der Anwendung gelesen werden. Dadurch können Konfigurationseinstellungen außerhalb des Containers geändert und verwaltet werden, ohne dass der Container selbst geändert werden muss.

- Datenbanken: Wenn eine Anwendung eine Datenbank verwendet, sollten die Datenbankdateien in einem Volumen gespeichert werden, um die Daten persistenz und Skalierbarkeit zu gewährleisten. Dadurch können Datenbankinstanzen einfach gestartet, gestoppt und auf verschiedene Hosts verschoben werden, ohne Daten zu verlieren.

- Log-Dateien: Log-Dateien können in Volumen gespeichert werden, um eine langfristige Aufbewahrung und Analysierbarkeit zu ermöglichen. Dies ist besonders wichtig, um das Verhalten der Anwendung im Laufe der Zeit zu überwachen und Fehler zu diagnostizieren.

- Benutzeruploads: Wenn Benutzer in einer Anwendung Dateien hochladen, sollten diese Dateien in einem Volumen gespeichert werden, um sicherzustellen, dass sie auch nach dem Neustart oder der Skalierung der Anwendung verfügbar bleiben.

Der Einsatz von Volumen bietet die Flexibilität und Zuverlässigkeit, um wichtige Daten in Container-basierten Umgebungen zu speichern und zu teilen, und ist entscheidend für die erfolgreiche Nutzung von Containern in produktiven Anwendungen.

Docker Volumes

In Docker gibt es drei Hauptarten von Volumes, die verwendet werden können, um Daten zwischen einem Hostsystem und einem Container auszutauschen:

1. Bind Mounts (Bind-Montage): Mit Bind Mounts wird ein bestimmtes Verzeichnis oder eine Datei auf dem Hostsystem direkt mit einem Verzeichnis im Container verbunden. Das bedeutet, dass Änderungen sowohl auf dem Host als auch im

Container sichtbar sind. Bind Mounts werden verwendet, wenn Sie Daten zwischen dem Host und dem Container in Echtzeit synchronisieren möchten. Sie eignen sich gut für die lokale Entwicklung, das Testen oder das Bereitstellen von Anwendungen, bei denen der Fokus auf der gemeinsamen Nutzung von Daten zwischen dem Host und dem Container liegt.

2. Volumes: Volumes sind eigenständige Verzeichnisse innerhalb des Docker-Ökosystems und sind unabhängig vom Dateisystem des Hostsystems. Docker erstellt ein spezielles Verzeichnis auf dem Hostsystem, das den Containerdaten entspricht. Volumes bieten eine höhere Isolierung und Portabilität. Sie sind auch robust gegenüber dem Löschen von Containern und können von mehreren Containern gleichzeitig verwendet werden. Volumes werden in Produktionsumgebungen häufig verwendet, um persistente Daten zu speichern, wie beispielsweise Datenbankdateien, Konfigurationen oder Protokolldateien.

3. tmpfs-Mounts: tmpfs ist ein virtuelles Dateisystem im Arbeitsspeicher, das in Docker als tmpfs-Mount genutzt werden kann. Ein tmpfs-Mount ist ein temporärer Speicherort im Container, der nur im Arbeitsspeicher existiert und nicht auf die Festplatte geschrieben wird. tmpfs-Mounts sind nützlich, wenn Sie temporäre Daten speichern müssen, die während der Laufzeit des Containers benötigt werden, aber nicht über den Containerlebenszyklus hinaus persistiert werden müssen. Beispiele für die Verwendung von tmpfs-Mounts sind das Speichern von temporären Zwischenergebnissen, Caches oder temporären Log-Dateien.

Die Wahl des richtigen Volume-Typs hängt von den spezifischen Anforderungen Ihrer Anwendung ab:

- Bind Mounts eignen sich gut für die lokale Entwicklung, bei der der Fokus auf der Echtzeitsynchronisierung von Daten zwischen dem Host und dem Container liegt.

- Volumes sind am besten geeignet, wenn Sie persistente Daten speichern müssen, die über den Containerlebenszyklus hinaus bestehen bleiben sollen und von mehreren Containern gemeinsam genutzt werden können.

- tmpfs-Mounts sind sinnvoll, wenn temporäre Daten benötigt werden, die nur während der Container-Laufzeit existieren sollen und keinen permanenten Speicher erfordern.

Es ist wichtig zu beachten, dass diese Volume-Typen auch kombiniert werden können, um den spezifischen Anforderungen einer Anwendung gerecht zu werden. Docker bietet eine flexible und leistungsstarke Infrastruktur für den Umgang mit Daten innerhalb von Containern und ermöglicht es Ihnen, das passende Volume für Ihre Anforderungen auszuwählen.

In Docker gibt es drei Hauptarten von Volumes, die verwendet werden können, um Daten zwischen einem Hostsystem und einem Container auszutauschen:

1. Bind Mounts (Bind-Montage): Mit Bind Mounts wird ein bestimmtes Verzeichnis oder eine Datei auf dem Hostsystem direkt mit einem Verzeichnis im Container verbunden. Das bedeutet, dass Änderungen sowohl auf dem Host als auch im Container sichtbar sind. Bind Mounts werden verwendet, wenn Sie Daten zwischen dem Host und dem Container in Echtzeit synchronisieren möchten. Sie eignen sich gut für die lokale Entwicklung, das Testen oder das Bereitstellen von Anwendungen, bei denen der Fokus auf der gemeinsamen Nutzung von Daten zwischen dem Host und dem Container liegt.

2. Volumes: Volumes sind eigenständige Verzeichnisse innerhalb des Docker-Ökosystems und sind unabhängig vom Dateisystem des Hostsystems. Docker erstellt ein spezielles Verzeichnis auf dem Hostsystem, das den Containerdaten entspricht. Volumes bieten eine höhere Isolierung und Portabilität. Sie sind auch robust gegenüber dem Löschen von Containern und können von mehreren Containern gleichzeitig verwendet werden. Volumes werden in Produktionsumgebungen häufig verwendet, um persistente Daten zu speichern, wie beispielsweise Datenbankdateien, Konfigurationen oder Protokolldateien.

3. tmpfs-Mounts: tmpfs ist ein virtuelles Dateisystem im Arbeitsspeicher, das in Docker als tmpfs-Mount genutzt werden kann. Ein tmpfs-Mount ist ein temporärer Speicherort im Container, der nur im Arbeitsspeicher existiert und nicht auf die Festplatte geschrieben wird. tmpfs-Mounts sind nützlich, wenn Sie temporäre Daten speichern müssen, die während der Laufzeit des Containers benötigt werden, aber nicht über den Containerlebenszyklus hinaus persistiert werden müssen. Beispiele für die Verwendung von tmpfs-Mounts sind das Speichern von temporären Zwischenergebnissen, Caches oder temporären Log-Dateien.

Die Wahl des richtigen Volume-Typs hängt von den spezifischen Anforderungen Ihrer Anwendung ab:

- Bind Mounts eignen sich gut für die lokale Entwicklung, bei der der Fokus auf der Echtzeitsynchronisierung von Daten zwischen dem Host und dem Container liegt.

- Volumes sind am besten geeignet, wenn Sie persistente Daten speichern müssen, die über den Containerlebenszyklus hinaus bestehen bleiben sollen und von mehreren Containern gemeinsam genutzt werden können.

- tmpfs-Mounts sind sinnvoll, wenn temporäre Daten benötigt werden, die nur während der Container-Laufzeit existieren sollen und keinen permanenten Speicher erfordern.

Es ist wichtig zu beachten, dass diese Volume-Typen auch kombiniert werden können, um den spezifischen Anforderungen einer Anwendung gerecht zu werden. Docker bietet eine flexible und leistungsstarke Infrastruktur für den Umgang mit Daten innerhalb von Containern und ermöglicht es Ihnen, das passende Volume für Ihre Anforderungen auszuwählen.

Mounts in Linux

In Linux bezieht sich "mount" auf den Vorgang, ein Dateisystem an einem bestimmten Ort im Dateisystem-Hierarchiebaum einzuhängen. Ein Mount-Punkt ist ein Verzeichnis in der Verzeichnisstruktur, das als Startpunkt für das eingehängte Dateisystem fungiert. Mounts ermöglichen den Zugriff auf externe Dateisysteme oder Geräte und erweitern so die Funktionalität des Betriebssystems.

Hier sind einige Verwendungszwecke und Vorteile von Mounts in Linux:

1. Zugriff auf externe Speichermedien: Durch das Einhängen von externen Speichergeräten wie USB-Sticks, Festplatten oder Netzwerkfreigaben können Sie auf die darin enthaltenen Dateien und Verzeichnisse zugreifen.

2. Netzwerkdateisysteme: Mounts ermöglichen den Zugriff auf Dateien, die auf entfernten Servern über ein Netzwerkdateisystem (z.B. NFS oder CIFS) freigegeben sind. Dadurch können Sie entfernte Dateien wie lokale Dateien behandeln und darauf zugreifen.

3. Einbindung virtueller Dateisysteme: Linux bietet verschiedene virtuelle Dateisysteme wie "/proc", "/sys" und "/tmpfs". Durch das Einhängen dieser virtuellen Dateisysteme können Sie auf systemrelevante Informationen zugreifen oder temporären Speicher nutzen.

4. Chroot-Umgebungen: Durch das Einhängen von Verzeichnissen in einer Chroot-Umgebung können Sie den Zugriff auf bestimmte Teile des Dateisystems einschränken und eine isolierte Umgebung für Prozesse oder Dienste erstellen.

5. Einrichtung von Dateisystem-Hierarchien: Durch das Einhängen von Dateisystemen an bestimmten Punkten im Dateisystem-Hierarchiebaum können Sie eine ordnungsgemäße Organisation und Strukturierung von Dateien und Verzeichnissen sicherstellen.

6. Flexibilität und Skalierbarkeit: Mounts ermöglichen es Ihnen, Speicherplatz und Ressourcen flexibel zu nutzen und verschiedene Dateisysteme nahtlos in das System einzubinden. Dadurch können Sie die Speicherkapazität erweitern oder neue Funktionen hinzufügen, ohne das Betriebssystem neu zu installieren oder neu zu starten.

Zusammenfassend bieten Mounts in Linux die Möglichkeit, externe Speichermedien, Netzwerkdateisysteme und virtuelle Dateisysteme in das Dateisystem des Betriebssystems einzubinden. Dies erweitert die Funktionalität des Systems, ermöglicht den Zugriff auf entfernte Ressourcen und schafft Flexibilität bei der Verwaltung von Dateien und Verzeichnissen.

Sinnvoller einsatz

Volumes können bei der Entwicklung von Containern auf verschiedene Weisen richtig und sinnvoll eingesetzt werden. Hier sind einige bewährte Praktiken:

1. Persistenz von Daten während der Entwicklung: Wenn Sie an einer Anwendung arbeiten und Änderungen an Dateien im Container vornehmen, können Sie ein Volume verwenden, um diese Änderungen persistenz zu machen. Auf diese Weise gehen Ihre Änderungen nicht verloren, wenn der Container neu gestartet wird. Dies ist besonders nützlich, wenn Sie Anwendungen lokal entwickeln und regelmäßig Codeänderungen vornehmen.

2. Trennung von Quellcode und Abhängigkeiten: Es ist oft empfehlenswert, den Quellcode einer Anwendung und ihre Abhängigkeiten getrennt voneinander zu halten. Sie können ein Volume verwenden, um den Quellcode des Containers einzubinden, während die Abhängigkeiten über das Basisimage oder Paketmanager installiert werden. Dadurch können Sie den Quellcode aktualisieren, ohne den gesamten Container neu bauen zu müssen.

3. Externe Konfigurationen bereitstellen: Manchmal ist es erforderlich, Konfigurationsdateien oder Geheimnisse in den Container einzubinden. Anstatt diese direkt im Container selbst zu speichern, können Sie ein Volume verwenden, um diese Dateien bereitzustellen. Auf diese Weise können Sie die Konfigurationen leichter ändern, ohne den Container neu erstellen oder aktualisieren zu müssen.

4. Datenfreigabe zwischen Containern: Wenn Sie mehrere Container ausführen und Daten zwischen ihnen teilen müssen, können Sie Volumes verwenden, um einen gemeinsamen Speicherort bereitzustellen. Dadurch können Container auf

dieselben Daten zugreifen, was die Kommunikation und Zusammenarbeit zwischen den Containern erleichtert.

5. Testdaten verwalten: Beim Testen von Anwendungen können Sie Volumes verwenden, um Testdaten bereitzustellen oder Ergebnisse zu speichern. Dadurch können Sie Testdaten leicht aktualisieren oder austauschen und haben eine bessere Kontrolle über die Testumgebung.

Beachten Sie jedoch, dass es auch einige Überlegungen gibt:

- Sicherheit: Stellen Sie sicher, dass sensible Daten, wie Zugangsdaten oder Geheimnisse, nicht in Volumes gespeichert werden, die von anderen Containern oder Services gelesen werden können.

- Versionierung: Behalten Sie die Versionierung von Volumes im Auge, um sicherzustellen, dass die richtige Version des Volumes mit dem richtigen Container verwendet wird.

- Reinigung: Löschen Sie nicht mehr benötigte Volumes, um Speicherplatz zu sparen und potenzielle Sicherheitsrisiken zu minimieren.

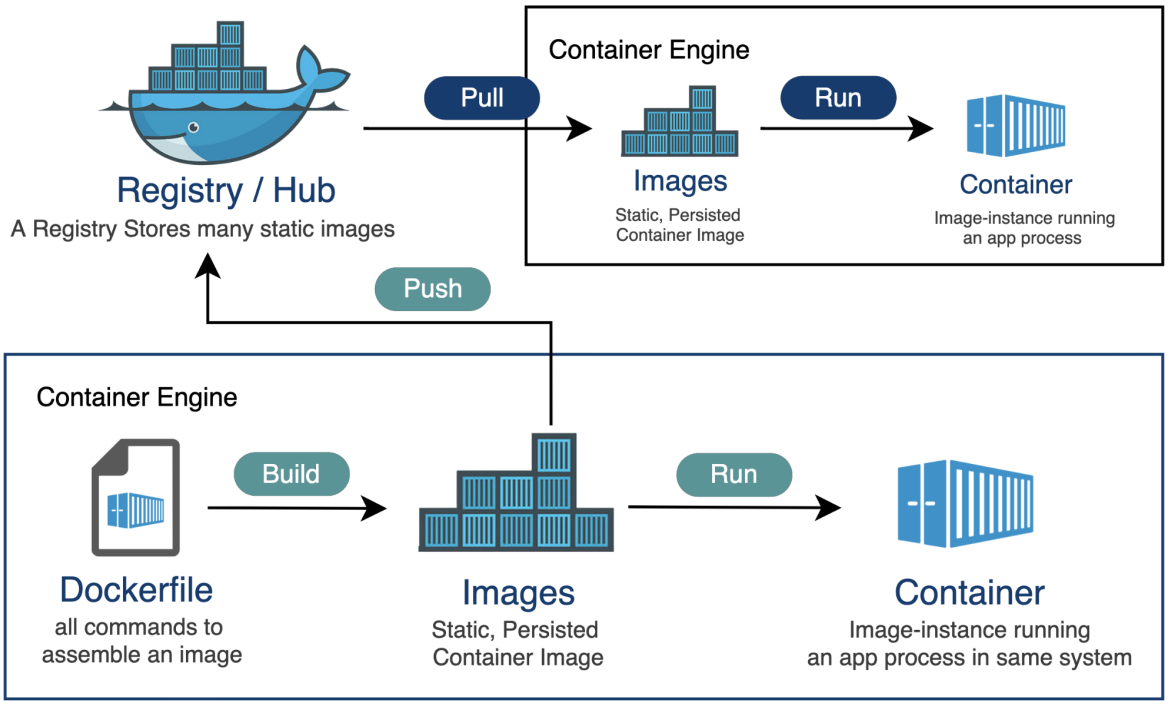
Die Verwendung von Volumes erfordert ein Verständnis der spezifischen Anforderungen Ihrer Anwendung und des Entwicklungsprozesses. Durch sorgfältige Planung und Berücksichtigung der oben genannten Punkte können Sie Volumes effektiv in Ihre Containerentwicklung einbinden und von ihren Vorteilen profitieren.

Tag 5

Ein Container-Registry ist ein zentraler Speicherplatz für Docker-Images oder andere Container-Images. Docker-Images oder Container-Images sind Vorlagen, die genutzt werden, um Container zu erstellen. Container-Images enthalten alle notwendigen Abhängigkeiten und Konfigurationsinformationen, um eine Anwendung oder einen Dienst innerhalb eines Containers zu betreiben.
Ein Container-Registry ermöglicht es Benutzern, Docker-Images oder andere Container-Images zu verwalten und zu speichern.
Benutzer können diese Images einfach herunterladen (pull) und auf ihren Systemen ausführen, um Container zu erstellen und Anwendungen innerhalb dieser Container auszuführen.

Images lassen sich auch hochladen (push), um für andere Benutzer diese für den Download zur Verfügung zu stellen. Container-Registrys können öffentlich (public) sein, was bedeutet, dass jeder auf sie zugreifen und Images herunterladen kann. Es gibt auch private Registry (private), was bedeutet, dass sie nur für autorisierte Benutzer oder Organisationen zugänglich sind.
Die Verwendung eines Container-Registrys hat viele Vorteile, darunter die Möglichkeit, Images zentral zu speichern und zu verwalten, die Wiederverwendbarkeit von Images zu fördern, die Reproduzierbarkeit von Builds (Herstellung von Versions-Zuständen) zu gewährleisten und die Verteilung von Images über verschiedene Systeme und Umgebungen zu erleichtern.

Container-Registrys sind ein wichtiger Bestandteil von DevOps und der modernen Anwendungsentwicklung und ermöglichen es Unternehmen, schnellere und effizientere Software-Entwicklungs- und Bereitstellungsprozesse zu implementieren.



Versionierung von Images in container registries

Die Versionierung von Images in einem Container-Registry wird oft mit sogenannten Tags durchgeführt. Tags sind einfach zu verwaltende Kennzeichnungen, die einem Container-Image zugewiesen werden, um es innerhalb des Container-Registrys eindeutig zu identifizieren und zu versionieren.

Ein Tag ist im Grunde eine Zeichenfolge, die dem Namen des Images angehängt wird und es kennzeichnet. Ein Tag kann aus einer Version, einem Datum, einem

Namen oder einer anderen Kennzeichnung bestehen, die der Entwickler oder Administrator wählt.
Container-Registrys ermöglichen es Ihnen, verschiedene Tags auf dasselbe Image anzuwenden, was es Benutzern ermöglicht, verschiedene Versionen des Images mit unterschiedlichen Tags abzurufen.

Ein Beispiel für die Verwendung von Tags in einem Container-Registry wäre die Kennzeichnung eines Images mit der Versionsnummer "1.0.0" und dem Tag "latest". Wenn Sie später ein Update an der Anwendung durchführen und ein neues Image erstellen, können Sie es mit derselben Versionsnummer "1.0.0" versehen, aber mit einem neuen Tag, z.B. "myapp:1.0.0-update1". Dadurch können Benutzer und Entwickler zwischen verschiedenen Versionen des Images wählen, je nachdem, welche Tags ihnen zur Verfügung stehen.
Ein weiterer nützlicher Aspekt der Verwendung von Tags in einem Container-Registry ist, dass sie dazu beitragen können, die Kompatibilität zwischen verschiedenen Anwendungen und Diensten sicherzustellen. Wenn ein Image mit einem bestimmten Tag ausgeführt wird, kann man sicher sein, dass es mit anderen Anwendungen und Diensten, die ebenfalls mit diesem Tag arbeiten, kompatibel ist.