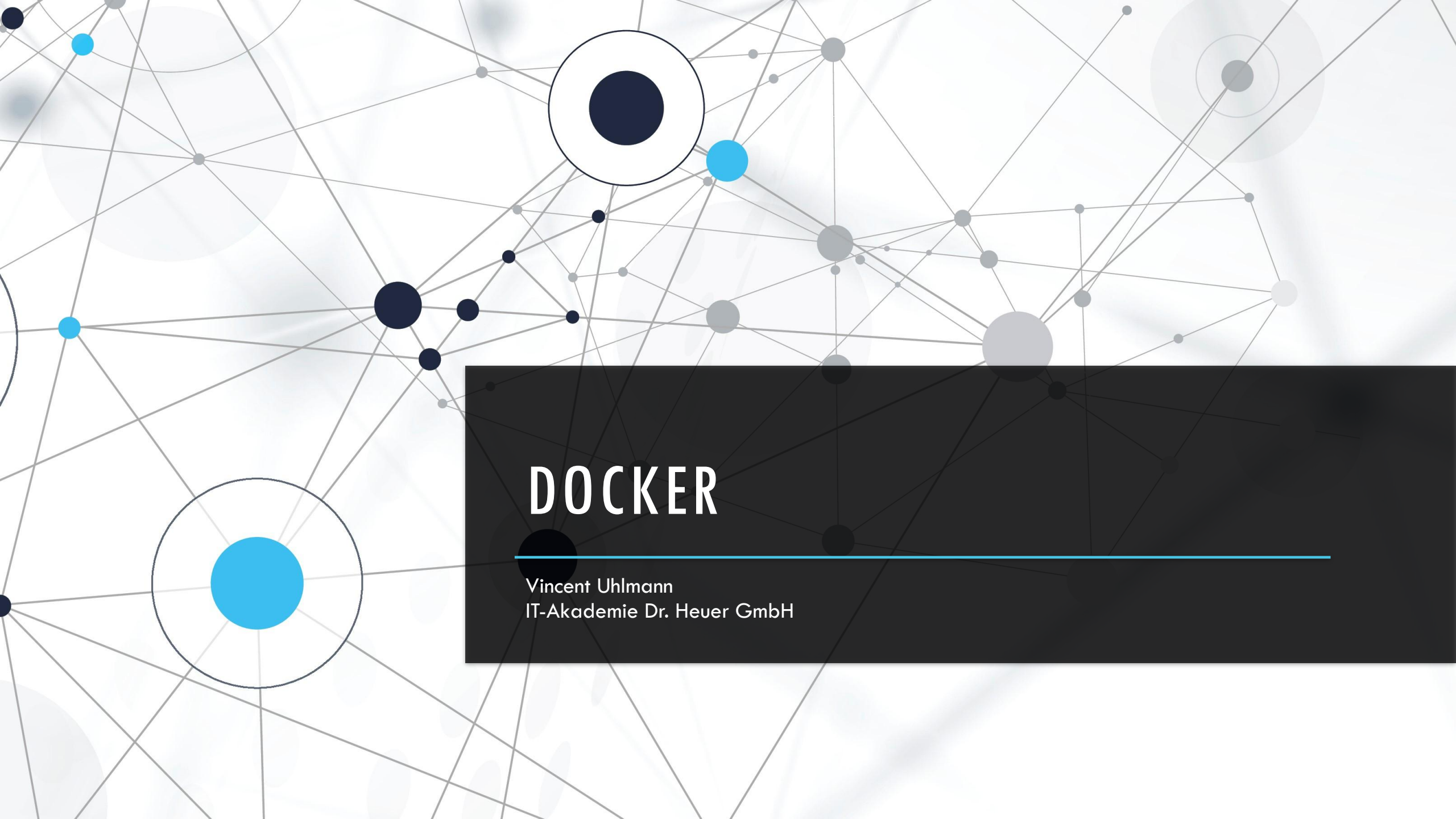


DOCKER

Vincent Uhlmann
IT-Akademie Dr. Heuer GmbH

DOCKER

- Docker ist eine Open-Source-Plattform, die Anwendungen in **Containern** isoliert
- Einheitliche Laufzeitumgebungen für Anwendungen, unabhängig von Betriebssystem und Infrastruktur
- Unterstützt Entwicklung, Testing und Deployment durch Konsistenz und Portabilität

- Anwendungsfälle
- Microservices-Architekturen
- Skalierbares Deployment in der Cloud
- Testing und Continuous Integration/Continuous Deployment (CI/CD)

DOCKER IMAGE

- Ein Docker Image ist eine **schreibgeschützte** Vorlage, die alle notwendigen Dateien und Konfigurationen enthält, um einen Container zu starten

- Bestandteile
- Basis-Image (z. B. ein Betriebssystem wie Ubuntu oder Alpine)
- Installierte Software und Abhängigkeiten
- Anwendungscode und Konfigurationen

UNTERSCHIED ZU EINER VM

- Docker Image
- Enthält nur die Anwendung und ihre Abhängigkeiten
- Nutzt den Kernel des Hostsystems

- VM
- Vollständiges Betriebssystem
- Simulierter Kernel

WAS IST EIN CONTAINER

- Ein Container ist eine laufende Instanz eines Images
- Isolierte Umgebung für Anwendungen
- Leichtgewichtig und schnell startbar im Vergleich zu VMs
- Kann gestoppt, pausiert und wieder gestartet werden
- Container teilen den Kernel des Hostsystems, nutzen aber separate Ressourcen wie Netzwerk, Speicher und CPU

VORTEILE

- Container teilen den Kernel, wodurch weniger Overhead entsteht
- Container laufen auf jedem System mit Docker. Keine Anpassungen notwendig
- Container starten in Sekunden, VMs in Minuten
- Container können einfacher skaliert und verwaltet werden
- Anwendungen bleiben unabhängig und beeinflussen sich nicht gegenseitig

WAS SIND DOCKER VOLUMES

- Volumes dienen dazu, Daten außerhalb des Containers zu speichern und wiederzuverwenden
- Container sind kurzlebig; Daten im Container gehen bei Neustart oder Löschung verloren
- Volumes sichern Daten unabhängig vom Lebenszyklus eines Containers

- Persistente Speicherung für Datenbanken
- Gemeinsame Nutzung von Konfigurationsdateien oder Logs zwischen Containern

MONOLITHISCHE ARCHITEKTUR

- Eine monolithische Architektur ist eine Softwarestruktur, bei der alle Komponenten einer Anwendung in einer einzigen, großen Codebasis integriert sind

- Eine einzige ausführbare Einheit (z. B. ein .jar- oder .exe-File)
- Gemeinsame Ressourcen und Speicherbereiche
- Eng gekoppelte Module und Funktionen

- Beispiel Onlineshop
- Enthält Funktionen für Benutzerverwaltung, Produktkatalog, Warenkorb und Zahlung in einem einzigen System

MONOLITHISCHE ARCHITEKTUR

- Probleme
- Schwer skalierbar: Alle Funktionen müssen zusammen skaliert werden
- Änderungen an einem Modul können das gesamte System beeinträchtigen
- Langsame Entwicklungszyklen durch Abhängigkeiten

MICROSERVICE ARCHITEKTUR

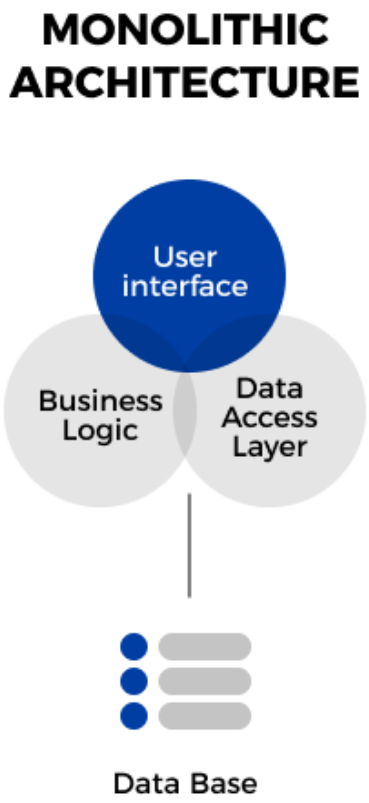
- Microservices sind eine Architektur, bei der Anwendungen in unabhängige, kleinere Services aufgeteilt werden, die über APIs miteinander kommunizieren

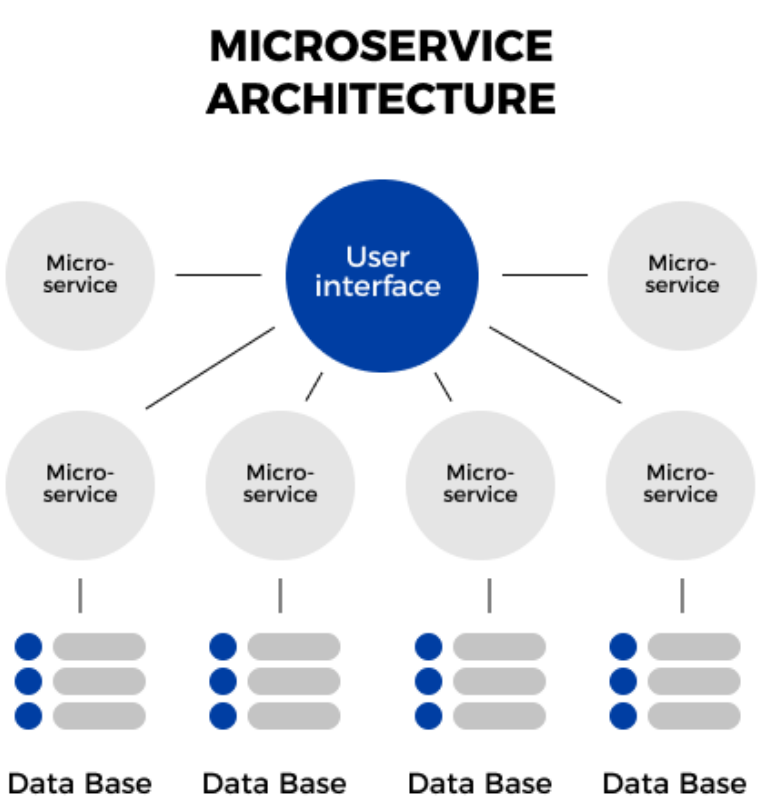
- Lose gekoppelte, unabhängige Module
- Jeder Service kann eigenständig entwickelt, bereitgestellt und skaliert werden
- Services sind oft zuständig für eine spezifische Domäne oder Funktion

MICROSERVICE ARCHITEKTUR

- Beispiel Onlineshop
- Benutzer-Service: Verwaltung von Konten und Authentifizierung
- Produkt-Service: Verwaltung des Produktkatalogs
- Bestell-Service: Abwicklung von Warenkorb und Bestellung
- Zahlungs-Service: Integration von Zahlungsmethoden wie PayPal oder Kreditkarte

MICROSERVICE ARCHITEKTUR





DOCKER UND MICROSERVICES

- Jeder Microservice wird in einem eigenen Container ausgeführt
- Unterschiedliche Services können verschiedene Programmiersprachen, Frameworks oder Abhängigkeiten nutzen
- Docker sorgt für eine einheitliche Ausführung unabhängig von der Host-Umgebung
- Container enthalten alles, was für die Ausführung des Microservices benötigt wird
- Images können einmal erstellt und auf verschiedenen Servern bereitgestellt werden
- Services können unabhängig voneinander skaliert werden, indem zusätzliche Container gestartet werden
- Beispiel: Bei hoher Last kann der Bestell-Service separat skaliert werden, ohne den Produkt-Service zu beeinträchtigen