Docker & Kubernetes
Teil 5

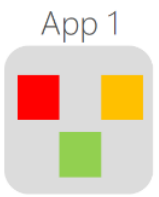
Teil 5 – … und mehr Hinweise

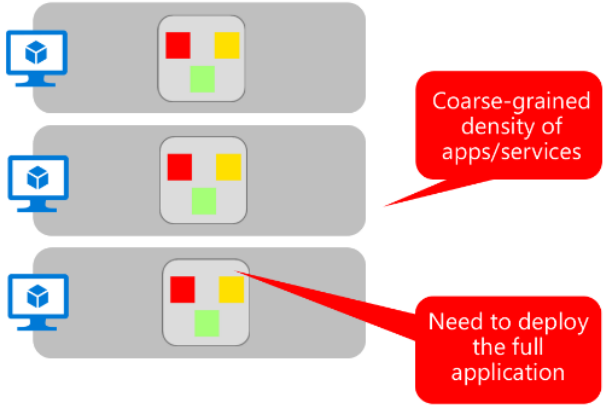
- Weiterführende Hinweise
  - Microservices Architektur
  - 12-Faktor Prinzipien
  - CNCF-Stack
  - CI/CD

Microservices Architektur

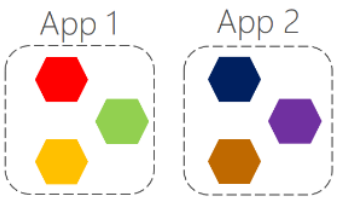
- Monolith vs. Microservices

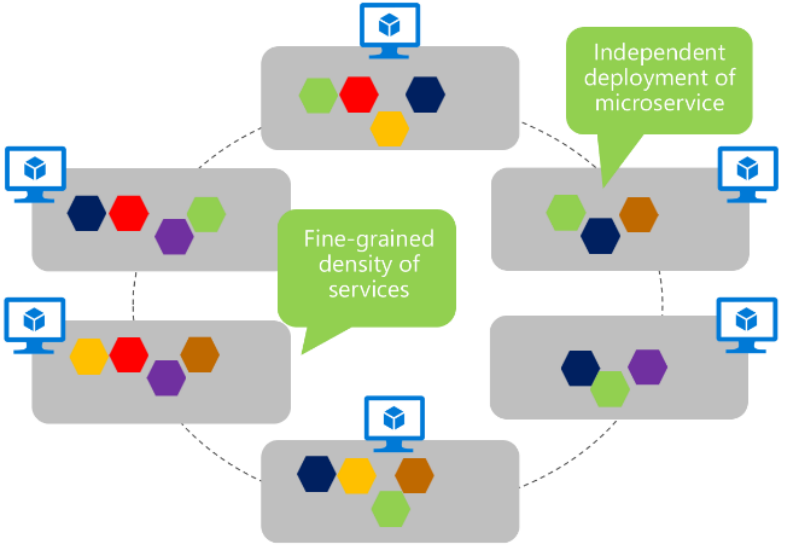
- A traditional application has most of its functionality within a few processes that are componentized with layers and libraries.
- Scales by cloning the app on multiple servers/VMs





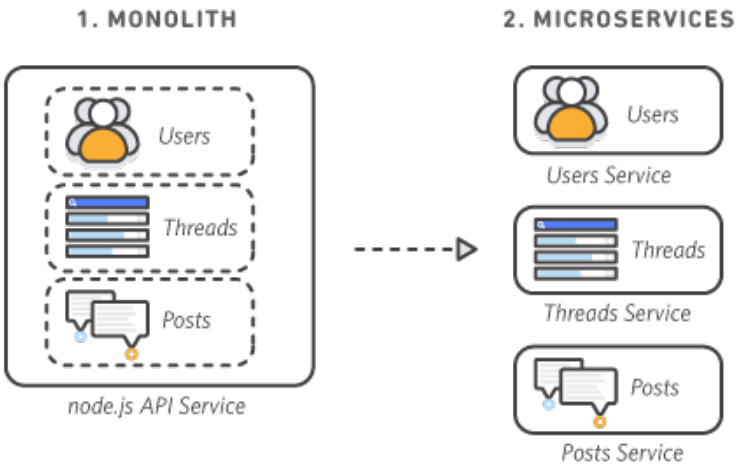
- A microservice application segregates functionality into separate smaller services.
- Scales out by **deploying each service independently** with multiple instances across servers/VMs





Microservices Architektur

- Eigenständigkeit
  - Alle Services sind komplett *separiert* und *isoliert* … es gibt keine Abhängigkeiten
  - Kommunikation über klar definierte Schnittstellen
- Spezialisierung
  - Jeder Service führt klar definierte Funktionen aus
  - Konzentration auf Lösung *eines* Problems



Microservices Architektur

- Weniger Abhängigkeiten in den Modulen
- Kleine Module
- Kapselung der Funktionalität
- Skalierbar auf Funktionsebene
- Evolvierbarkeit ist entscheidend … Unterstützung von kleinen, agilen Teams
- Einfache Korrektur und Bereitstellung von Fehler und Erweiterungen
- Unterbrechungsfreier Betrieb
- Unabhängigkeit von Technologien/Umgebungen … technologische Flexibilität
- Robuster Betrieb

Microservices Architektur

- 12 Factors … Industriestandard
- Message-Queues
  - NATS
  - Redis
  - RabbitMQ
  - ZeroMQ … NNG
- REST-API‘s
  - Swagger / OpenAPI
  - Grpc
- CI/CD
  - Qualitätssicherung einbinden (statische und dynamische Checks)
  - Verknüpfung mit Versionskontrollsystem
  - Jenkins, Drone, …

12 Factors – ausgewählte Faktoren

- https://12factor.net/de/
- 1 Codebase
  - Eine Codebase im Versionsverwaltungssystem
  - Unterschiedliche Deployments aus der Codebase
  - Gleicher Code wird über Bibliotheken und eine Abhängigkeitsverwaltung zwischen Apps & Services geteilt

12 Factors – ausgewählte Faktoren

- 2 Abhängigkeiten
  - Für eine App/einen Service werden alle Abhängigkeiten vollständig definiert
  - Keine Abhängigkeit von „systemweiten Paketen“
  - Jede App/jeder Service ist eine geschlossene, vollständige Einheit

12 Factors – ausgewählte Faktoren

- 3 Konfiguration
  - Strikte Trennung von Code und Konfiguration
  - Konfiguration immer über Umgebungsvariablen

    => einfaches und flexibles Deployment
- 6 Prozesse
  - Zustandslos
  - Share nothing … jeder Service hat seine eigenen Daten (die nie geteilt werden)

12 Factors – ausgewählte Faktoren

- 9 Einweggebrauch
  - Prozesse/Services können weggeworfen werden
  - Schnelles Starten und schnelles Beenden

    => schnelles Skalieren und Deployment von Änderungen
- 10 Dev-Prod-Vergleichbarkeit
  - Geringer „Abstand“ zwischen Entwicklung, Staging und Produktion in Funktionalität, Zeit und Infrastruktur

12 Factors – ausgewählte Faktoren

- 11 Logging
  - Prozesse/Services kümmern sich nicht um Logs
  - Ströme von Ereignissen werden von Laufzeitumgebung gesammelt und zusammengefasst
  - Andere Systeme werten die Zusammenfassungen aus und signalisieren Probleme

CNCF-Stack

- https://www.cncf.io/
- Cloud Native Computing Foundation
- Sammlung von OS-Projekten/Tools, die stabil und produktiv verwendbar für das Cloud Computing sind
  - Graduated … stabil und produktiv ausgereift
  - Incubating … stabil
  - Sandbox … noch in der frühen Entwicklung/Nutzung
- CNCF-Landscape
  - https://landscape.cncf.io/card-mode?project=sandbox
  - Suche nach passenden Tools/Projekten

CI/CD

- Continuous Integration/Continuous Delivery
  - Weiterentwicklungen und Änderungen sollen schneller in die Codebasis eingefügt werden
  - Neue Versionen sollen häufig und schnell ausgeliefert werden
- Agile Entwicklung notwendig
- CI/CD-Pipeline: Verknüpfung und Automatisierung möglichst aller Schritte
  - Entwicklung (Versionsverwaltung, lokale Tests)
  - Bau (jede Codeänderung löst Bau einer neuen Version aus)
  - Testautomatisierung (statische Codeanalyse, Unittests, Integrationstests, Akzeptanztests, UI-Tests, Sicherheitstests, Lasttests, Lizenzchecks, ...)
  - Auslieferung (packaging, deployment)

CI/CD

- Vorteile
  - Schnelle und verlässliche Entwicklung
  - Frühe Fehlererkennung im Entwicklungsprozess
  - Mehr Transparenz
  - Weniger Probleme mit Kunden
  - Geringere Kosten
- Tools (Auswahl)
  - Jenkins
  - CircleCI
  - Gitlab
  - Drone
  - Travis

… Bildquellen

- RedHat
- IDG
- Microsoft
- Docker
- Earthly
- Aquasec
- dev.to