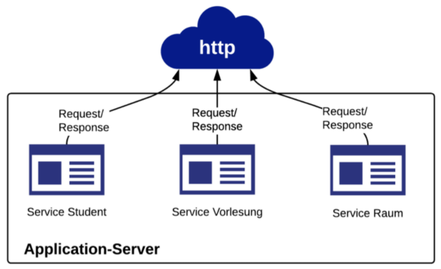
Docker & Microservices Cheatsheet

Sebastian Aurin, Marcel Otega Oßwald

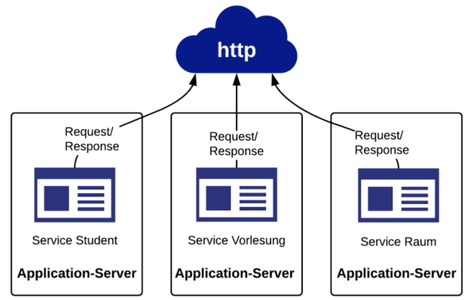
Hochschule Mannheim, Fakultät für Informatik

Microservices ohne Docker

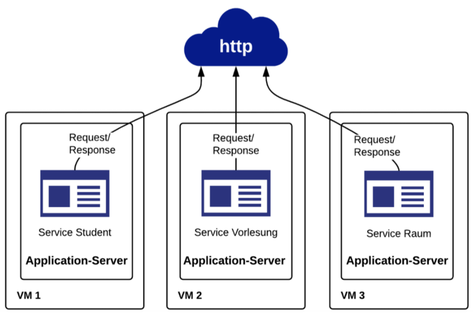
- Deployment und Services beeinflussen sich gegenseitig im Laufzeitverhalten
- Ein Service kann alle Services zum Ausfall bringen
- Einzelne Services können nicht skaliert werden
- Mögliche Konflikte in den Abhängigkeiten



- Hoher Aufwand für System-Konfiguration, Deployment und Monitoring
- Sehr ressourcenhungrig, jeder Service eigene Ablaufumgebung und Abhängigkeiten
- Beeinflussen sich gegenseitig im Laufzeitverhalten



- Sehr ressourcenhungrig, jede VM benötigt eigenes Betriebssystem



Docker Compose

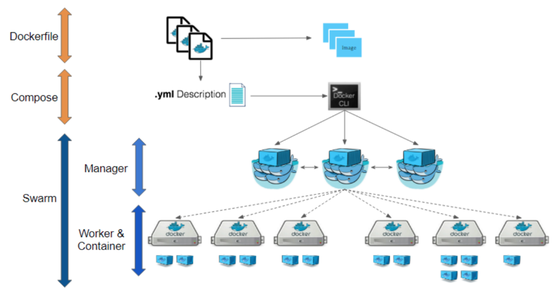
Mit einer docker-compose-Datei können mehrere Container automatisiert gestartet werden. Diese können dabei auch automatisch verlinkt werden, es können Volumes gemountet oder Ports freigegeben werden und vieles mehr. Auf diese Weise, kann man mit einem einzigen Befehl, **docker-compose up -d**, alle seine Container starten.

```
version: '3'
services:
  beispiel-container
    image:java:8
    ports:
      - 8080:8080
    volumes:
      - /var/lib/docker/data:/var/lib/appdata
    links:
      - mysql-container
    deploy:
      replicas: 5
      restart_policy:
        condition: on-failure
    command: ['java', '-jar', 'app.jar']
```

- **image** gibt das Baseimage an, das für den Container verwendet werden
- **ports** mappt einen Port vom Host auf den Container
- **volumes** mountet ein Volume, um Daten zu persistieren
- **links** verlinkt den angegebenen Container mit dem aktuellen Container
- **replicas** gibt an wieviele Kopien gestartet werden sollen, wenn es als Stack auf einem Manager gestartet wird
- **restart_policy** gibt an, ob bzw. wann der Container neu gestartet werden soll command definiert den Startbefehl

Docker Swarm - Grundkonzept

Als erste Ausgangslage hat man ein oder mehrere Dockerfiles erstellt. Da jedes Dockerfile einzeln gebaut und gestartet werden muss, ist der nächste Schritt eine docker-compose.yml - Datei zu erstellen, die mithilfe der Dockerfiles alle Images automatisch erzeugt und die Container startet, inkl. Verlinken, Portmapping und allem was sonst beim Starten von Containern zu beachten ist. Außerdem werden an dieser Stelle auch deploy-Befehle in der Compose-Datei berücksichtigt. Mithilfe von docker stack kann im nächsten Schritt die Compose-Datei auf einem Swarm Manager ausgerollt werden. Ab diesem Punkt kann man den Swarm um beliebige Worker erweitern, auf denen dann auch die Container ausgeführt werden.



Docker Stack- & Swarm-Befehle

- **$ docker stack deploy -c docker-compose.yml stackname1** – startet die angegebene Compose-Datei auf einem (Swarm) Manager
- **$ docker stack ls** – zeigt alle Stacks an
- **$ docker stack ps stackname1** – zeigt alle Container auf dem Stack an
- **$ docker stack rm stackname1** – löscht einen Stack und alle seine Container
- **$ docker swarm init** – initialisiert einen swarm
- **$ docker swarm leave** – force entfernt einen Swarm

Vorteile die Docker & Swarm mitbringt

- einheitliche Ablaufumgebung
  - weniger Dokumentation notwendig
  - 'Works for me' Syndrom fällt weg
    - Konfigurations- und Kompatibilitätsprobleme sind auf ein Minimum reduziert
- Docker kann dutzende Services auf einen Dev-System starten
  - produktive Cluster-Umgebung mit Containern am lokalen Entwicklerrechner nachstellen
- einfache Wartung
  - Entwickler liefern lauffähiges Image
  - Container können einfach zur Laufzeit ausgetauscht werden
- Einheitliche Entwicklungsumgebung mit Docker möglich
  - weniger Aufwand für die Einrichtung der Entwicklungsumgebung
  - spezifische Images für jedes Team möglich
- Testen der Anwendung
  - virtuelle Testserver in denen die Anwendungen automatisiert in Sekunden gestartet werden
  - Bei Bedarf kann für jeden Test ein eigener Container gestartet werden, somit ist es möglich das die Tests sich nicht gegenseitig beeinflussen (isoliert testen)
- blitzschnell Services starten
  - zusätzliche Instanzen
  - neue Services
- Änderungen der Systemlast
  - kann umgehend darauf reagiert werden
  - Container können bedarfsgerecht zu- oder abgeschaltet werden (Auto-Scaling)
- bei Softwareprobleme im laufenden Betrieb
  - einfach neuen Service starten
  - nicht langwierig versuchen die Instanz zu reparieren
  - Fehlerlogs können in Ruhe geprüft werden
- einfache zentrale Überwachung der Container ist direkt mit docker stats möglich
  - CPU-Auslastung
  - Speichernutzung/verfügbarer Speicher
  - Netzwerkdatenverkehr

Sebastian Aurin

Marcel Ortega Oßwald