

Studienprojekt

Automatisiertes Deployment mittels Drone.io

Fakultät Informationstechnik
Studiengang Softwaretechnik und Medieninformatik

Pascal Kneisel
WS 22/23

Datum: 01.11.2022 - 28.02.2023

Erstprüfer: Prof. Dr.-Ing. Reinhard Schmidt
Zweitprüfer: Matthias Häußler

Inhaltsverzeichnis

1	Problemstellung	1
2	Herangehensweise	3
3	Docker Compose	4
4	Drone Konfiguration	5
5	Ergebnisse und Fazit	8
5.1	Ausblick	9
	Literatur	10

Abbildungsverzeichnis

1.1	Statista Umfrage: Umsatz mit Cloud Computing weltweit von 2010 bis 2021 und Prognose bis 2023 [1]	1
1.2	Statista Umfrage: Umsatz mit Software-as-a-Service (SaaS) weltweit von 2010 bis 2021 und Prognose bis 2023 [2]	2
5.1	Durchgeführtes Deployment	8
5.2	Deployment Maske Drone.io	9
5.3	Testprogramm nach dem Deploymentprozess	9

Listings

3.1	docker-compose.yml Skript zum Deployment der Software	4
4.1	.drone.yml erster Teil: Deployment des Backends	5
4.2	.drone.yml zweiter Teil: Deployment des Backends	6
4.3	.drone.yml dritter Teil: lokales Deployment mittels Docker-Compose	7

Abkürzungsverzeichnis

IaaS Infrastructure as a Service

PaaS Platform as a Service

SaaS Software as a Service

1 Problemstellung

Die Tendenz der modernen Softwareentwicklung geht in die Richtung der serviceorientierten Bereitstellung. So nehmen Themen wie Software as a Service (SaaS), Platform as a Service (PaaS) oder Infrastructure as a Service (IaaS) an Bedeutung zu. Dies wird in der Studie des Unternehmens Gartner ersichtlich welche in Abbildung 1.1 ersichtlich wird. So steigt der Umsatz im Cloud Computing Umfeld seit 2010 streng monoton und hat sich von 2010 bis 2021 beinahe verzehnfacht. [1]

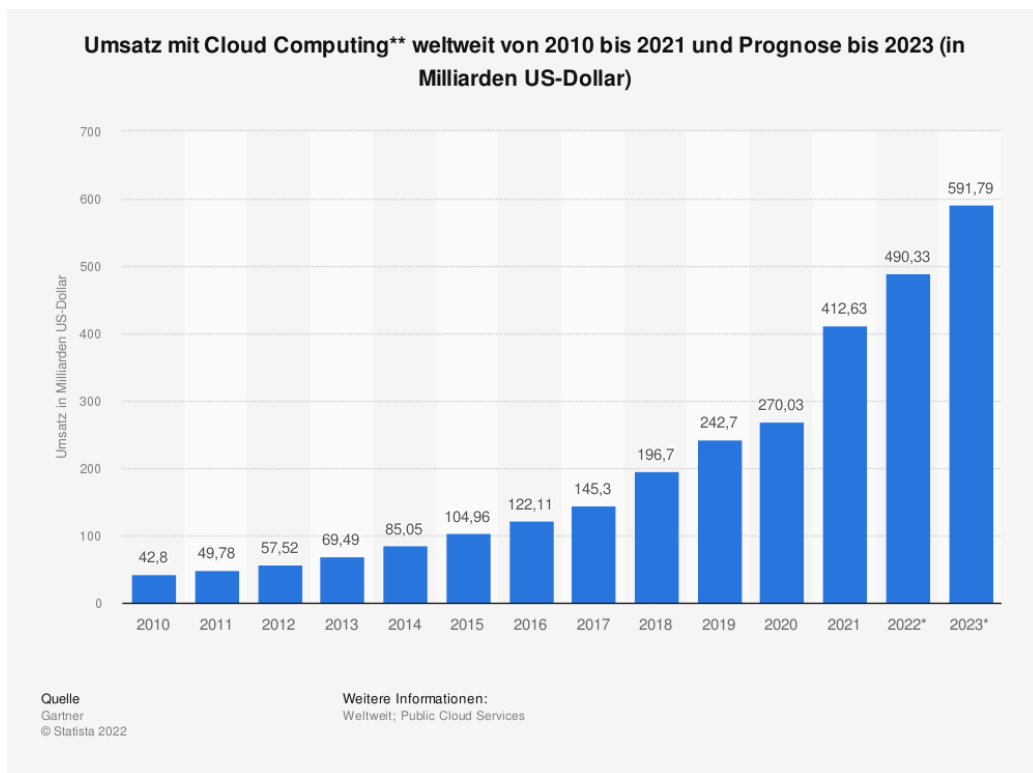


Abbildung (1.1): Statista Umfrage: Umsatz mit Cloud Computing weltweit von 2010 bis 2021 und Prognose bis 2023 [1]

Vor allem im Umfeld des SaaS steigt die Nachfrage in den letzten Jahren enorm. Dies zeigt eine Studie, welche ebenfalls durch das Unternehmen Gartner erhoben wurde. Diese ist in Abbildung 1.2 ersichtlich und unterstreicht, dass im professionellem Umfeld die serviceorientierte Dienstleistung stetig zunimmt. [2]

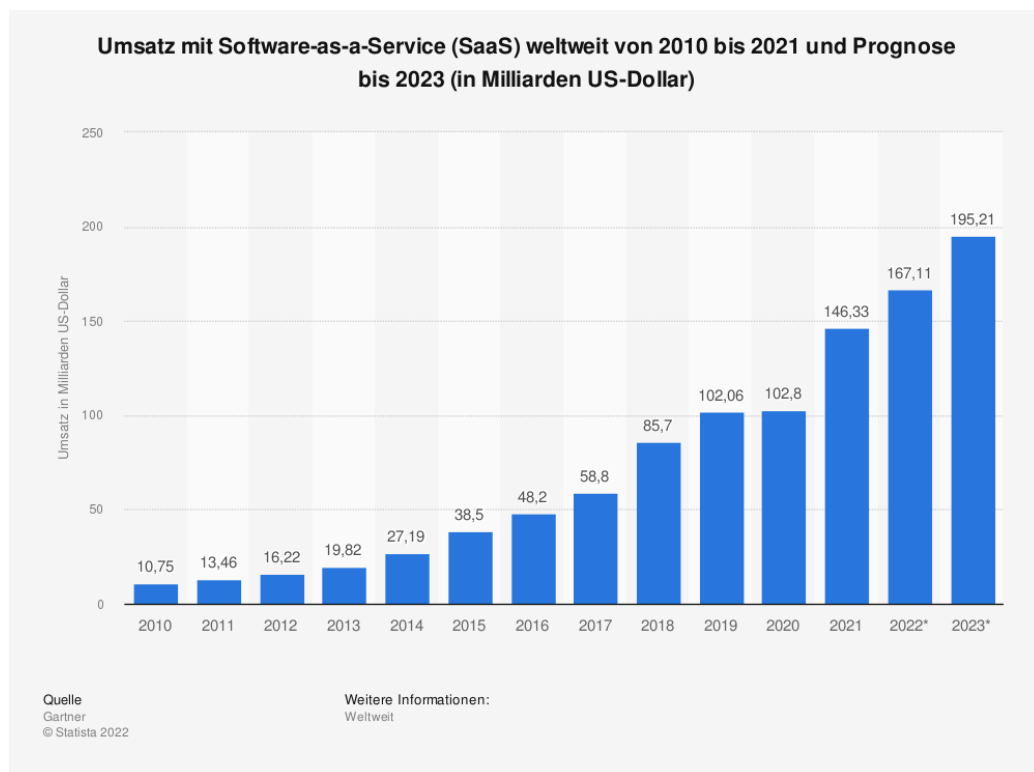


Abbildung (1.2): Statista Umfrage: Umsatz mit Software-as-a-Service (SaaS) weltweit von 2010 bis 2021 und Prognose bis 2023 [2]

Durch die steigende Nachfrage an Produkten der Kategorie SaaS, hat die Software selbst wachsende Anforderungen. Der Deploymentprozess einer Software kann hierbei jedoch schnell komplex werden, da oftmals im Bereich der Microservices weitere Dienste wie eine Datenbank oder ähnliches benötigt werden. Dies lässt sich mittels moderner Continuous Integration und Continuous Delivery Plattformen vereinfachen.

In einem kurzen Beispiel wird die CI/CD Plattform Drone.io [3] dazu verwendet, um Software automatisiert zu bauen, auf DockerHub zu veröffentlichen und auf einer lokalen Docker-Infrastruktur zu deployen.

2 Herangehensweise

Der Deployment-Prozess von Software kann sehr unterschiedlich sein. Einerseits ist es möglich, die Software manuell bereit zu stellen. Andererseits gibt es Automatisierungswerkzeuge wie Ansible um die Software aus dem Repository zu klonen, zu bauen und schließlich auf dem Produktivsystem zu installieren.

Eine seit einigen Jahren etablierte Form von Deployment-Prozessen ist Continuous Delivery. Hierbei wird die CI/CD Plattform bei jedem Push in das Repository getriggert und arbeitet je nach Konfiguration die einzelnen Schritte des Prozesses ab. Diese Konfiguration kann flexibel je nach Bedarf gestaltet werden. So ist es beispielsweise möglich, die Software bei jedem Push in das Repository automatisiert zu testen. Auch denkbar ist es, bei jedem Merge in den master-Branch, die Software automatisiert zu bauen und auf einem Repository wie DockerHub hochzuladen oder ein Release in dem GitHub Repository zu erstellen.

In diesem Studienprojekt kommt Drone.io mit Verbindung zu GitHub zum Einsatz. Drone.io ist eine CI/CD Plattform, welche im eigenen Heimnetz gehostet werden kann und mittels OAuth2 die Verbindung zum Repository-Provider wie GitHub oder auch Gitea herstellt. Drone.io wird durch einen Trigger angestoßen, dieser ist in der Regel ein Push in einen Branch. Zur Ausführung der einzelnen Schritte können Container aus einem Container-Repository wie DockerHub verwendet werden.

Die zu testende Software ist eine Spring-Boot Applikation mit einem Next.js Frontend. Next.js ist ein Wrapper um das React-Framework und nimmt dem Entwickler die Arbeit rund um das Bereitstellen des Frontends ab. Die Beispielapplikation hat keine bedeutende Funktion, es soll nur die Kommunikation zwischen Front- und Backend beweisen. Die Daten des Backends werden in einer MariaDB-Datenbank gespeichert, welche nicht Teil des Docker-Compose Skripts ist, sondern manuell bereitgestellt wird.

Das Deployment der Software erfolgt mittels Docker-Compose. Hierfür wird das offizielle docker Image auf DockerHub verwendet, welches die aktuellste Docker-Compose Version inkludiert. Da im Drone.io Runner bereits der Socket des Docker-Hosts eingebunden ist, wird hierüber die Software mittels Docker-Compose deployed.

Die Kommunikation der beiden Microservices, namentlich das Front- und das Backend erfolgt über HTTP-Anfragen. Damit das Frontend auf das Backend zugreifen kann, muss während der Kompilierung der Pfad zum Backend angegeben werden. Dies wird durch die `.env`-Datei verwirklicht, welche sich im Stammverzeichnis des Frontends befindet. Durch das lokale Deployment ist die URL des Backends `http://192.168.0.125:8080`.

3 Docker Compose

Zum Deployment wird, wie in Kapitel 2 erläutert, Docker-Compose verwendet. Hierzu wurde eine `docker-compose.yml`-Datei erstellt. Diese ist in Listing 3.1 dargestellt. Die sicherheitsrelevanten Daten wie die Zugangsdaten der MariaDB-Instanz werden durch Umgebungsvariablen eingefügt. Außerdem werden die Container in das Netzwerk `studienprojekt` eingebunden, worüber der Zugriff auf die MariaDB-Instanz erfolgt.

```
version: '2'

services:
  backend:
    image: selfmaderulez/studienprojekt-hse-pakn
    environment:
      MARIADB_USERNAME: ${DRONE_SECRET_DB_USER}
      MARIADB_PASSWORD: ${DRONE_SECRET_DB_PASS}
      MARIADB_HOST: ${DRONE_SECRET_DB_HOST}
      MARIADB_PORT: 3306
      MARIADB_DATABASE: studienprojekt
    ports:
      - "8080:8080"
    networks:
      - network1

  frontend:
    image: selfmaderulez/studienprojekt-hse-pakn-frontend
    ports:
      - "3001:3000"
    networks:
      - network1

networks:
  network1:
    name: studienprojekt
    external: true
```

Listing (3.1): `docker-compose.yml` Skript zum Deployment der Software

4 Drone Konfiguration

Die Drone.io Konfiguration besteht aus drei Schritten:

1. Frontend bauen und auf DockerHub deployen
2. Backend bauen und auf DockerHub deployen
3. Falls Deployment angestoßen wird: lokales Deployment über Docker-Compose

In dem Listing 4.1 ist die Pipeline dargestellt, welche das Frontend baut und auf DockerHub deployed. Hierfür wird das Drone.io Docker-Plugin verwendet. Dieses baut anhand des Dockerfiles, welches im Frontend-Verzeichnis vorhanden ist, das Frontend und deployed es auf DockerHub mit den Zugangsdaten, welche von Drone.io während der Ausführung eingefügt werden. Hierdurch müssen keine sensiblen Daten wie Passwörter oder ähnliches in dem Repository gehalten werden.

```
---
kind: pipeline
type: docker
name: frontend

steps:
- name: deploy frontend to DockerHub
  image: plugins/docker
  settings:
    context: frontend
    dockerfile: frontend/Dockerfile
    username:
      from_secret: docker_user
    password:
      from_secret: docker_pass
    repo: selfmaderulez/studienprojekt-hse-pakn-frontend
    tags: [latest, 1.0.0]
```

Listing (4.1): .drone.yml erster Teil: Deployment des Backends

In dem Listing 4.2 wird der zweite Teil der Konfiguration dargestellt. Dieser baut das Backend über das maven Docker-Image und veröffentlicht es schließlich auf DockerHub. Hier wird ersichtlich, dass auch zwischen den Schritten das benötigte Image gewechselt werden kann. So wird im ersten Schritt das maven-Image verwendet und im zweiten Schritt wiederum das Drone.io Docker-Plugin Image wie in der Pipeline zuvor.

```
---
kind: pipeline
type: docker
name: backend

steps:
- name: build backend
  image: maven
  commands:
    - mvn clean package
- name: deploy backend to DockerHub
  image: plugins/docker
  settings:
    username:
      from_secret: docker_user
    password:
      from_secret: docker_pass
  repo: selfmaderulez/studienprojekt-hse-pakn
  tags: [latest, 1.0.0]
```

Listing (4.2): .drone.yml zweiter Teil: Deployment des Backends

Im dritten Teil der Drone-Konfiguration, welcher in Listing 4.3 dargestellt wird, findet das Deployment statt. Hierfür muss der Docker-Socket in diesem Beispiel in die Deployment-Pipeline übernommen werden. Da das mounten eine kritische Aktion ist, muss das Repository als Trusted-Repository angelegt werden. Dies ist eine einfache Einstellung in den Einstellungen des Repositorys. Der Deployment-Prozess soll nur gestartet werden, wenn das Event promote mit dem Ziel production angestoßen wurde. Wie im Schritt zuvor werden die sensiblen Daten von Drone automatisiert geladen. Schlussendlich wird das docker-compose Skript über den Befehl `docker compose up -d` ausgeführt und somit die Software deployed.

```
---
kind: pipeline
type: docker
name: deployment

steps:
- name: deploy
  image: docker
  environment:
    DRONE_SECRET_DB_USER:
      from_secret: db_user
    DRONE_SECRET_DB_PASS:
      from_secret: db_pass
    DRONE_SECRET_DB_HOST:
      from_secret: db_host
  volumes:
  - name: docker-socket
    path: /var/run/docker.sock

  when:
    event:
      - promote
    target:
      - production

  commands:
    - docker compose up -d

depends_on:
- frontend
- backend

volumes:
- name: docker-socket
  host:
    path: /var/run/docker.sock
```

Listing (4.3): .drone.yml dritter Teil: lokales Deployment mittels Docker-Compose

5 Ergebnisse und Fazit

Der Deployment-Prozess ist spielend einfach, wenn dieser einmal konfiguriert ist. So ist ein durchgeführtes Deployment mittels Drone.io in Abbildung 5.1 ersichtlich. Im Dropdown-Menü am oberen rechten Bildrand ist außerdem der Promote-Button ersichtlich, welcher das Deployment anstößt. Dieser ruft die Deployment-Maske auf, auf der weitere parameter eingefügt werden könnten. Diese wird in Abbildung 5.2 gezeigt.

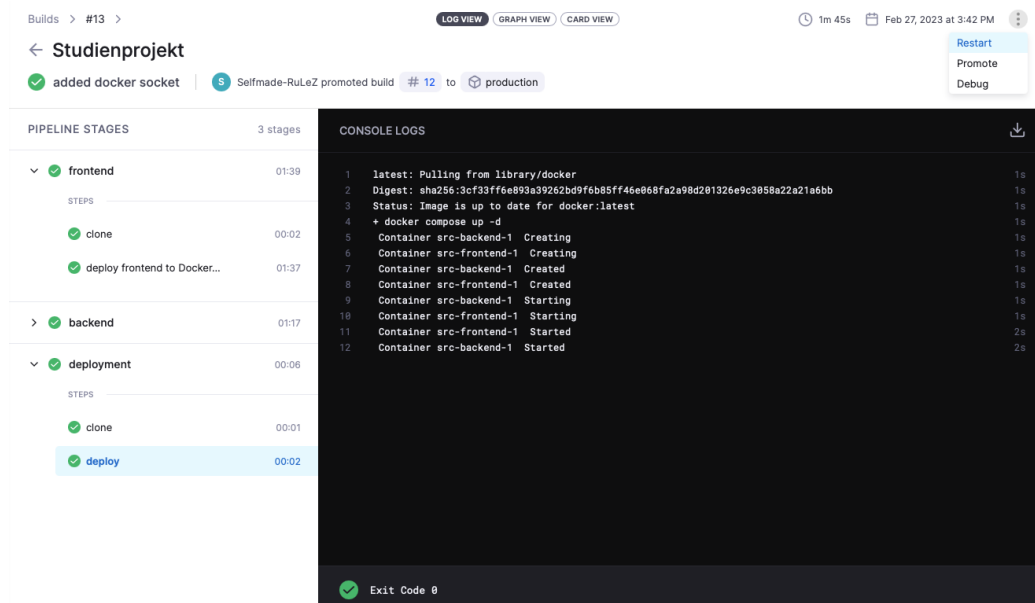


Abbildung (5.1): Durchgeführtes Deployment

Das Testprogramm, welches eine simple CRUD-API bietet, ist nach dem Deployment-Prozess auf dem lokalen Docker-Host deployed. Dieses ist in Abbildung 5.3 ersichtlich und zeigt, dass dieses Studienprojekt erfolgreich verlaufen ist.

CI/CD Plattformen können den Berufsalltag des Softwareentwicklers deutlich erleichtern. Vor allem bei dem derzeitigen Hype der serviceorientierten Softwareentwicklung ist dieses Werkzeug ein Segen für jedes Unternehmen.

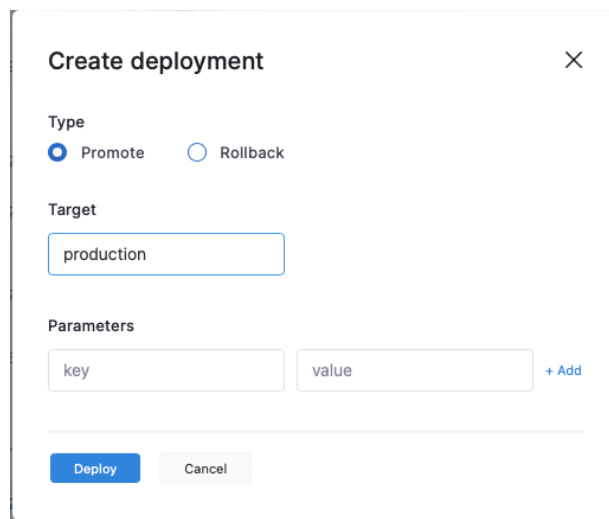


Abbildung (5.2): Deployment Maske Drone.io

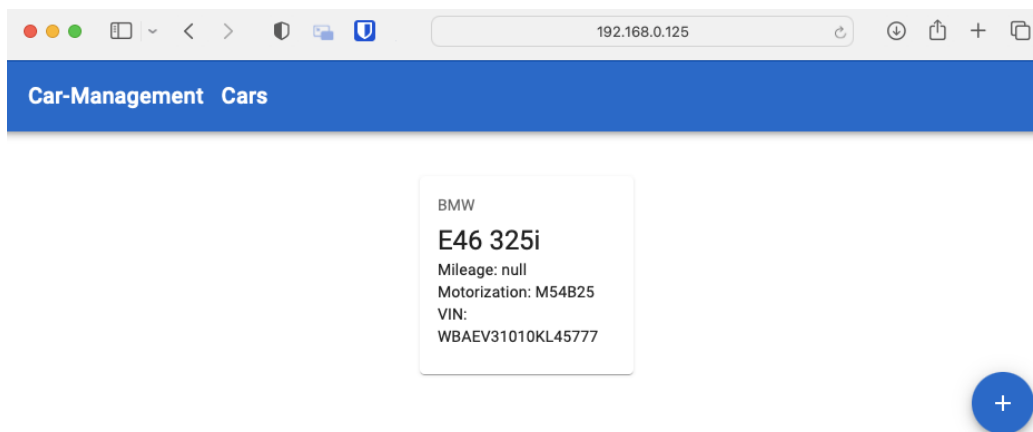


Abbildung (5.3): Testprogramm nach dem Deploymentprozess

5.1 Ausblick

Anstelle des bisherigen Deployments mittels der lokalen Dockerumgebung wäre es denkbar, dies auf einem externen Kubernetescluster zu verwirklichen. Zusätzlich zum hier demonstrierten Ablauf ist es möglich, verschiedene Umgebungen basierend auf den Branches zu erstellen. So wäre es denkbar, für eine Staging-Umgebung einen separaten Branch zu erstellen und den main-Branch automatisch auf dem Produktivsystem zu deployen.

Literatur

- [1] Gartner. *Umsatz mit Cloud Computing** weltweit von 2010 bis 2021 und Prognose bis 2023*. Okt. 2022. URL: <https://de.statista.com/statistik/daten/studie/195760/umfrage/umsatz-mit-cloud-computing-weltweit/> (besucht am 20.02.2023).
- [2] Gartner. *Umsatz mit Software-as-a-Service (SaaS) weltweit von 2010 bis 2021 und Prognose bis 2023*. Okt. 2022. URL: <https://de.statista.com/statistik/daten/studie/194117/umfrage/umsatz-mit-software-as-a-service-weltweit-seit-2010/> (besucht am 20.02.2023).
- [3] Harness Inc. *Drone.io*. Feb. 2023. URL: <https://www.drone.io> (besucht am 20.02.2023).