BabyPort

12. Dezember 2023

Image wurde mit https://imageupscaler.com/ai-image-generator/ generiert am 05.12.2023.
(Prompt: Icon für eine Software mit Port und Container und Docker)

1. ÜBERSICHT

Die Software BabyPort dient als Container Management System zur Administration und Überwachung von Docker Containern. BabyPort erreicht das mittels einer GUI, die verschiedene administrative Funktionen auf Docker-Container abbildet.

2. ZIELE

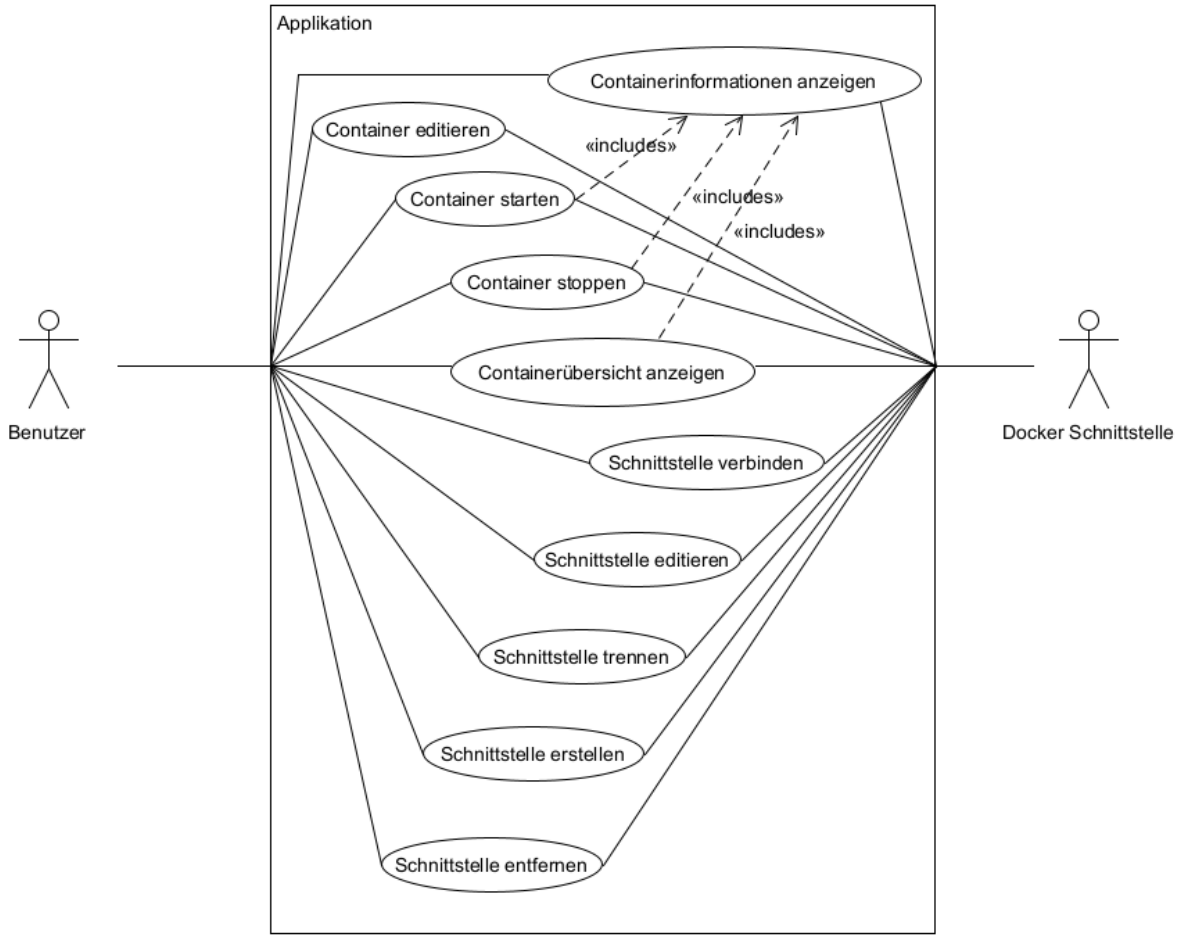
1. Containerverwaltung: Dieses System verwaltet Container mithilfe einer Systemschnittstelle. Diese Schnittstelle wird vom Hauptsystem per MQTT angesteuert. Mit Hilfe dessen können verschiedene Container zur Überwachung hinzugefügt, entfernt, gestartet, gestoppt oder bearbeitet werden.
2. Statusinformationen von Containern: Hierbei werden die Daten aufbereitet und zum Monitoring dargestellt.

3. STATISTIKEN BZGL. AUFWENDUNGEN

Person	Hauptbeitrag	Arbeitszeit (in Stunden (gerundet))
Felix Schiele	Plattform (GitLab, CI/CD, Sonar) Agent (Software-Entwicklung) Dokumentation	55
Nils Heinzelmann	Client-Server (Software-Entwicklung) Agent (Software-Entwicklung) UML-Modellierung	86
Marius Wörfel	Client-Server (Software-Entwicklung) Agent (Software-Entwicklung) Dokumentation	43
Sarah Ficht	Client-Server (UI-Entwicklung)	23

4. OVERALL USE CASE DIAGRAMM

Der Fokus dieser Implementierungsphase lag auf der Container Interaktion. Damit sind alle CRUD Operationen, welche in dem Use Case Diagramm abgebildet sind, gemeint. Dabei lag ein großer Teil der Implementierung auf Seiten der UI. Zudem wurde an der Schnittstelle gearbeitet. Für die korrekte Umsetzung aller Use Cases ist jedoch eine Verbindung zwischen der Hauptapplikation und dem Agent erforderlich. Diese wurde zu großen Teilen fertiggestellt.



5. ARCHITEKTURSTILE UND ARCHITEKTURENTSCHEIDUNGEN

Die einzelnen Software-Teile des Server-Clients wurden in Komponenten unterteilt und diese Komponenten dann in einzelnen Packages zusammengefasst. Die Haupt-Architektur des Softwareentwurfs umfasst die *MVC* (Model-View-Controller) Architektur. Die Wahl MVC zu verwenden hat den Hauptgrund, da wir durch die Trennung von Model, View und Controller eine künstliche Abstraktionsebene schaffen, in der wir die MQTT-Clients/Kommunikation einfach integrieren können, ohne die Software Architektur negativ zu beeinflussen.

Ferner wurde der Agent als Sidecar implementiert. Da wir nicht nur eine Hauptapplikation besitzen, benötigen wir eine weitere Komponente, den Agent, womit ein Zugriff betriebssystemunabhängig auf andere Geräte möglich wird. Daher haben wir das Sidecar Pattern gewählt, da dieses in der Industrie (Kubernetes Sidecar Container / Jenkins Build Prozessor Agents) für genau diesem Zweck Anwendung findet.

Ferner werden verschiedene Design-Patterns (wie z.B. *Builder*, *Factory*, *Singleton)* verwendet, um Code den zu entkoppeln, eine klare Struktur zu implementieren und die Wartbarkeit zu gewährleisten.

Die Haupt Kommunikation zwischen den Komponenten besteht aus einem *Observerpattern* welches eigenen Events erstellen, empfangen und versenden kann.

Aus dieser Modellierung ergibt sich, dass wir insbesondere die folgenden SOLID-Prinzipien einhalten.

- **Interface-Segregation-Prinzip:** Alle Funktionalitäten, die dazu dienen, zwischen Komponenten zu kommunizieren, wurden in viele kleine Interfaces oder andere Abstraktion-Konstrukte unterteilt, um so viele Abhängigkeiten wie möglich zu vermeiden.
- **Single-Responsibility-Prinzip:** Beim Entwurf wurde besonders darauf geachtet, dass eine Klasse nur einen Abhängig zu einer anderen besitzt, um eine Clean-Code-Architektur zu erzielen.

Außerdem haben wir uns vorgenommen, weitere Prinzipien wie **KISS** und **DRY** zu verwenden, um eine bessere Lesbarkeit sowie Erweiterbarkeit in der Zukunft zu gewährleisten.

6. SOFTWARE TOOLS UND PLATTFORM

Software-Tools

- Java (Sprache)
- Java Swing (UI-Framework)
- JUnit 5 (Unit Tests)
- Docker (Engine und Docker CLI)
- MQTT (Kommunikation Standard)
- SonarLint (Linter für Entwicklungsumgebung)
- Mockito (Mock-Testing Framework)
- Maven (Dependency Management + Building)
- Jetbrains IntelliJ Ultimate (Entwicklungsumgebung)
- Eclipse (Entwicklungsumgebung)
- Figma (UI-Mockup)
- Visual Paradigm (UML-Diagramm Editor)
- UMLet (UML-Diagramm Editor)

Plattform

- Ubuntu Server (MQTT-Broker und Agent)
- Desktop Environment (Linux, MacOS, Windows)
- SonarQube (statische Code-Analyse)
    - CheckStyle von Google (für Java)
    - Quality Gate Settings entsprechen Abgabebedingungen
- Jenkins (CI/CD)
- GitLab (Source-Code-Control)
- Jira (Projektmanagement + Zeiterfassung)
- Medium (Blog Posting Plattform)
- GitHub (Plattform für Dokumente Veröffentlichungen)

7. Highlights der Demo

- Erste Operationen in der UI verfügbar
- User-Input wird von Modellklassen gespeichert
- Erste UI Steuerung
- UI-Zustandsspeicherung (Look and Feel)
- Event-Processing mithilfe des Observer-Patterns
- Jenkins Pipeline
- Statische Code Analyse mit SonarQube