Anwendungen von Semantik MediaWiki

Seminararbeit

von

Patrick Eisele

Studiengang: Informationswirtschaft B.Sc.

Matrikelnummer: 1778090

Raphael Manke

Studiengang: Informationswirtschaft M.Sc.

Matrikelnummer: 1697607

An der KIT-Fakultät für Wirtschaftswissenschaften

Institut für Angewandte Informatik und Formale Beschreibungsverfahren

Prüfer:	Prof. Dr. York Sure-Vetter
Betreuer:	Sebastian Bader
Eingereicht am:	06.02.2018

Inhaltsverzeichnis

1 Motivation

In den letzten Jahren haben sich die Softwareentwicklung, sowie die Anforderungen an Software deutlich verändert. Anwendungen müssen heute für verteilte Systeme ausgelegt sein und nicht wie ursprünglich gedacht nur auf einem einzigen speziellen System lauffähig sein.

Damit einher gehen eine ganze Reihe neuer Probleme. Eine zeitaufwändige Herausforderung ist das Installieren und Aktuell halten aller benötigten Komponenten (Entwicklungsumgebung, Produktivsystem, Testsystem), ohne die es zu dem bekannten Problem kommt: Lokal auf meiner Maschine hat es doch funktioniert.

Speziell im Kontext eines Semantic MediaWikis ist es sehr zeitintensiv MediaWiki, Semantic MediaWiki jeweils auf den verschiedenen Servern, Entwicklungsumgebungen zu installieren, sowie aktuell zu halten. Des Weiteren ist insbesondere die erstmalige Installation für unerfahrene User eine Herausforderung. Diese Problematik soll im Rahmen dieser Seminararbeit gelöst werden.

2 Ziele dieser Arbeit

Im Rahmen dieser Seminararbeit sollen die im vorherigen Kapitel genannten Probleme beim Installieren und Aktuell halten eines Semantic MediaWikis gelöst werden.

Dazu soll es dem User, insbesondere auch unerfahrenen[1] , über ein Baukastensystem ermöglicht werden ein Semantic MediaWiki aufzusetzen. Hierzu soll eine intuitive Oberfläche gestaltet werden, mit welcher die benötigte Extension, ein Design und grundlegende Konfigurationen, wie Benutzername und Passwort ausgewählt werden können. Also eine stark vereinfachte- und zeitsparende click & run Installation.

Die Zielgruppe für unsere Lösung sehen wir durch diese Möglichkeiten breit gefächert:

- Neulinge, die bis jetzt noch nie mit einer Kommandozeile gearbeitet haben und die Inhalte für ein Wiki generieren. Ihnen wollen wir die Möglichkeit zum Testen, als auch ausprobieren, geben.
- Wenn zu Demonstrationszwecken innerhalb einer kurzen Zeit eine lauffähige Version erstellt werden muss.
- Erfahrene Nutzer, die unsere Lösung als Entwicklungsumgebung im täglichen Einsatz haben.
- Deployte Semantic MediaWikis basieren auf unserer Lösung

[1] Der User hat noch keine oder sehr wenige Erfahrung im Umgang mit der zugrundeliegenden Technologie des Semantic Mediawikis (PHP, MySQL, etc.) oder sogar nur wenig Erfahrung im Umgang mit der Kommandozeile

3 verwendete Technologie

3.1 Docker

Basis unserer Arbeit ist die Verwendung der Virtualisierungssoftware Docker[2], die es unter anderem ermöglicht, Anwendungen die aus verschiedenen Komponenten bestehen getrennt, voneinander zu betrachten. Klassische Komponenten einer Webanwendung sind hierbei ein Webserver, mit einer entsprechenden Anwendung und einer Datenbank, zum Persistieren der Inhalte der Anwendung.

Im folgende wird die Entwicklung hin zu Docker beschrieben, um dann die Vorteile gegenüber traditioneller Anwendungs-deployments aufzuzeigen. Im Anschluss daran werden die einzelnen Bestandteile von Docker näher erläutert.

3.1.1 Transformation von Anwendungslandschaften

Monolithische Software-Architektur

Zunächst einmal gibt es zu betrachten, wie es überhaupt zur Entwicklung von Docker gekommen ist. Traditionelle Anwendungen wurden in monolithischer Manier auf einzelnen Servern deployed. Hierbei mussten die Server einzeln und der Umgebung entsprechend provisioniert sowie konfiguriert werden. Meist wurden hierbei die Aufgaben in Systemadmintratoren und Entwickler strikt getrennt. Server wurden dann vom Systemadminitrator mit Betriebssystem, Anwendungsumgebung, Netzwerk, Storage, Berechtigungen und vielem mehr eingerichtet. Der Entwickler selbst lieferte dann ein Artefakt der Anwendung, das dann auf dem Server deployed wurde. Da eine Anwendung meist aus mehreren Backendkomponenten besteht, wurden auch alle benötigten Services mit auf der Maschine installiert. Am Ende hatte man eine, meist händisch zusammengestellte, Anwendung auf einer einzelnen Maschine deployed. Reichten die Ressourcen des Servers nicht mehr aus, so musste man den ganzen Aufwand erneut auf dem neuen Server durchführen.

Ähnlich problematisch war es, wollte man eine zweite Anwendung deployen. Meist wurde dann eine weitere Instanz gestartet, die dann wiederum alle Komponenten, die benötigt waren, beinhaltete. Alternativ, wenn man versuchte sie auf derselben Maschine auszurollen, so musste man stark darauf achten, die einzelnen Anwendungen voneinander zu kapseln, um so Nebeneffekte und Zugriffsrechte kontrollieren zu können.

Ein weiteres Problem war die schlechte Auslastung der Systeme. In Zeiten mit schwacher Nachfrage liefen die Server mit sehr geringer Auslastung und verbrauchten dabei unnötigen Strom und verursachten Kosten. In Zeiten mit starker Nachfrage konnten die Server nicht skalieren, da eine weitere Instanz der Anwendung erheblichen Aufwand mit sich zog.

[2] `https://www.docker.com/what-docker`

Demnach musste man die Systeme stets für den Worst-Case mit maximaler Auslastung dimensionieren. Plante man mit dem average-case oder gar weniger, so musste man mit Ausfällen bei hoher Nachfrage rechnen.

Virtualisierte Maschinen

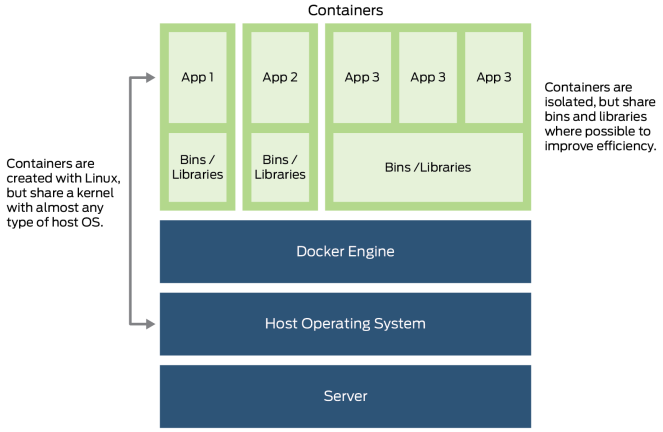
Eine große Weiterentwicklung stellte die Möglichkeit von virtuellen Maschinen (VM) dar. Hierbei wird auf einer Maschine, wie beispielsweise einem Server eine Virtualisierungssoftware installiert, die die Ressourcen wie CPU, Arbeitsspeicher und Festplatten auf verschiedene Virtuelle Maschinen verteilt. Somit können die Ressourcen eines Servers auf mehrere kleine (virtuelle) Maschinen verteilt werden. Jede virtuelle Maschine wiederum verhält sich wie eine vollständiges System. Einer virtuellen Maschine wird eine feste Menge an Ressourcen zur Verfügung gestellt, die exklusiv von ihr genutzt werden können. Eine VM muss dann ähnlich wie eine „echte“ Maschine ein Betriebssystem, Anwendungsumgebung und alle für die Anwendung benötigten Komponenten enthalten.

Eine konfigurierte virtuelle Maschine wird dann als „Image“ bezeichnet. Ein Image kann dann direkt auf einem Server mit entsprechender Virtualisierungssoftware deployed werden und verhält sich dabei immer gleich. Auf einem Server können wiederum mehrere Images gestartet werden, die jeweils voneinander komplett gekapselt sind. Kommunikation zwischen den einzelnen VM funktioniert dann lediglich über das Netzwerk. Die Ressourcen des Host System können dann auf die verschiedenen virtuellen Maschinen verteilt werden.

Insbesondere ermöglichen virtuelle Maschinen eine schnelle und einfache Skalierung von Anwendungen, da neue Instanzen eines Images einfach auf neuen Servern gestartet werden können, ohne das neue System individuell anpassen zu müssen.

Virtualisierung mit Containern

Abbildung 1: Aufbau eines System mit mehreren Containern



Quelle:
`https://www.juniper.net/assets/img/misc/diagram-what-is-docker-container.png`

Nachdem durch die Virtualisierung von Maschinen eine erste Abstraktionsebene bewältigt

wurde und der Anwender (beziehungsweise Entwickler) sich keine Gedanken mehr über die Ressourcen selbst machen muss, ist die nächste Ebene auch das Betriebssystem an sich zu teilen, unter mehreren Anwendungen um somit eine noch bessere Auslastung der System zu erreichen. Bei Containern wird das System in mehrere Schichten aufgeteilt. Als unterster Ebene wird ein grundlegendes Basissystem angegeben. Grundsätzlich eine bestimmte Linux Distribution wie zum Beispiel Debian. Dieses Grundsystem kann dann mithilfe einer Virtualisierungssoftware auf die tatsächlichen Ressourcen des Systems, dynamisch, zugreifen. Ein wichtiger Punkt hierbei ist, dass das Basissystem ebenfalls über mehrere Container geteilt werden kann.

Das Basissystem kann dann über mehrere Layer schichtweise erweitert werden. Ein Basissystem mit mehreren Layern kann dann als Image gespeichert werden und wiederrum als Basissystem für weitere Anwendungen verwendet werden. Ein Beispielhafter Aufbau wäre beispielsweise ein Debian Basissystem mit einem Layer der einen HTTP Server beinhaltet. Diese zwei Ebenen zusammen in einem Image können dann wiederrum verwendet werden um eine Webanwendung zu deployen. Um so interesannter wird es wenn mehrere Anwendungen die selben Basisimages verwenden, denn in diesem Fall wird dieses Image nur einmal auf dem Host System deployed und unter den einzelnen Webanwendungen gesteilt. Wird nun dieses Basissystem erneuert, so sind die Änderungen automatisch für alle darüberliegenden Schichten verwendbar. Eine Kapselung findet dann für jede Anwendung seperat statt und für jede Kommunikation nach außen, beispielsweise welche Ports geöffnet werden, muss explizit definiert werden. Auch Container untereinander sehen sich nicht gegenseitig, sofern nicht definiert. Genau diese Starke Kapselung ist eine Stärke von Containerbassiertem deployment. Es kann sehr prezise definiert werden welche Services miteinander Kommunizieren können und welche nicht.

Abbildung 1 zeigt einen solchen schematischen Aufbau. App 1 und 2 haben jeweils einen eigenen Anwendungskontext. App 3 läuft repliziert mit der selben Basis.

3.1.2 Vorteile von Containern

Container haben im Vergleich zu virtuellen Maschinen den Vorteil, dass sie viel leichtgewichtiger sind, da sie nur nur den Anwendungskontext betrachten und darunterliegende Voraussetzungen in Images auslagert, die unter mehreren Container geteilt werden kann. Virtuelle Maschinen hatte das Problem das jede VM alles bis hin zum Betriebssystem beinhaltete und dadurch sehr großwurden. Gegenüber monolithischen Systemen hat man den Vorteil, dass das deployment auf jedem System gleich funktioniert und keine Unterschiede zwischen lokalem und entferntem Ausführen auf dem Server vorliegt. Des weiteren ist ein Container deutlich schneller gestartet als eine komplette Maschine, egal ob virtuell oder physisch. Dies ist insbesondere vorteilhaft, wenn die Anzahl an redundanten Instanzen

mit der Last zusammen skalieren muss.

Ein weiterer Vorteil von Containern ist, dass die zur Verfügung stehenden Ressourcen, unter den einzelnen Containern, dynamisch, geteilt werden (sofern nicht anderweitig konfiguriert). Dadurch kann eine bessere Gesamtauslastung einer Maschine erreicht werden. Trotz gemeinsamer Ressourcennutzung bleibt dennoch eine klare Abgrenzung der einzelnen Container zueinander.

3.1.3 Komponenten von Docker

In diesem Abschnitt werden die grundlegenden Komponenten und Konzepte von Docker dargestellt und kurz in einen allgemeinen Kontext gebracht.

Dockerfile

Zunächst beginnt die Definition mit einem Dockerfile. Dieses gilt als Grundgerüst für ein Image.

Listing 1: Dockerfile Beispiel

```
FROM ubuntu
RUN apt-get update && apt-get install nginx
COPY ./config:/var/etc/config
EXPOSE 80 443
CMD [nginx]
```

Ein Beispielhafter Aufbau eines Dockerfiles ist in Listing 1 dargestellt.

Es beginnt im Normalfall mit einer „FROM“ Klausel die angibt auf welches Basisimage aufgebaut wird. Im Beispiel ist dies Ubuntu. Daraufhin wird eine Befehl, vergleichbar mit einer Kommandozeileneingabe ausgeführt. Dieser Befehl wird mit der Syntax
RUN <Befehl>
angegeben. Über den Befehl COPY werden Dateien vom lokalen Filesystem in das neue Image kopiert. EXPOSE wiederum gibt an welche HTTP-Ports von außerhalb erreichbar sein können. Der CMD Befehl gibt den Einstiegspunkt nach dem Containerstart an. Dieser Befehl wird beim start ausgeführt. Wichtig ist hierbei, dass an dieser Stelle ein Prozess aufgerufen wird, der im Vordergrund offen bleibt, da sonst der Container als Fehlerhaft erkannt wird und beendet wird.

Die hier genannten Befehle sind nicht abschließend. Eine vollständige Dokumetation lässt sich in der Dockerfile Referenz[3] nachschlagen.

Docker build

[3] https://docs.docker.com/engine/reference/builder/

Nachdem ein Dockerfile erstellt wurde kann daraus nun ein tatsächliches Docker-Image erstellt werden. Hierzu startet man den Docker build Prozess mit beispielsweise dem Kommando

```
docker build .
```

Hierbei wird dann zunächst das unter der FROM Klausel spezifizierte Image heruntergeladen und gestartet. Daraufhin werden die im Dockerfile spezifizierten Aktionen ausgeführt und das Ergebnis als neues Docker-Image gespeichert. Jeder Befehl des Dockerfiles erzeugt dabei einen neuen Layer der auf das Baisisimage „oben draufgepackt“ wird. Zur besseren Handhabung kann es an dieser Stelle sinnvoll sein den Images sinnvolle Namen zu geben und diese Images wiederverwenden zu können.

Docker run

Nachdem nun ein Image gebaut wurde kann man eine Instanz davon starten. Hierbei spricht man dann vom starten eines Containers. Mit dem Befehl

```
docker run <name des Images>
```

startet man den Container und es wird der Befehl der CMD Klausel des Dockerfiles aufgerufen. Das Ergenis ist eine Intanz des Images auch Container genannt.

Docker compose

Da eine moderne Anwendung nicht nur aus einem einzelnen Container, sondern meist aus mehreren besteht müsste man nun jeden container einzeln mit docker run starten und ihn mit den bereits gestarteten Containern verknüpfen. Um dieses Problem zu lösen gibt es Docker compose.

Hierbei handelt es sich um eine Spezifikation wie man mehrere Container gemeinsam starten und Konfigurieren kann. Die vollständigen Möglichkeiten können in der zugehörigen Dokumentation[4] nachgelesen werden.

Listing 2: Beispiel Docker-Compose Datei

```
version: '3'

services:
  db:
    image: mariadb
  web-app:
    image: mediawiki
    volumes:
      - ./LocalSettings.php:/var/html/LocalSettings.php
```

[4] https://docs.docker.com/compose/compose-file/

```
ports:
  - "8000:8000"
depends_on:
  - db
```

In Listing 2 werden zwei Container gestartet, zum einen eine Datenbank und zum andern eine Webanwendung. Zu beachten ist, dass bei der Webanwendung die Datei LocalSettings.php durch den volumes Befehl, zur Laufzeit ersetzt wird. Der Container greift dadurch auf die Daten des Hostsystems und nicht mehr auf die lokale im Container vorhandene Datei zu. Desweiteren wird ein Portmapping durchgeführt wodurch die App für den Anwender unter http://localhost:8000 erreichbar wird. Zuletzt wird noch die Startreihenfolge festgelegt, sodass die Webapp erst gestartet wird wenn die Datenbank deployed wurde. Der Befehl:

```
docker-compose up
```

startet das deployment der docker-compose Datei.

Docker Hub

Als letztes ist noch die Komponente Docker Hub[5] zu erwähnen. Hierbei handelt es sich um eine zentrale Registry für Docker Images. In dieser Registry können Nutzer oder auch Unternehmen fertige Images bereitstellen die dann einfach über den Docker run oder Docker-Compose gestartet werden können. Darüber hinaus kann ein solches Image als Basis für eigene Images verwendet werden.

3.2 Angular

Zur Umsetzung der graphischen Oberfläche wurde das Webapplikationsframework[6] Angular[7] in der Version fünf eingesetzt. Das ist eine Open-Source-Software, welche von einer großen Community und angeführt von Google entwickelt wird. Eine Vielzahl an bekannten Webseiten[8], wie Tesla, Oskars, Microsoft (Support) und natürlich Google eigene, verwenden Angular in unterschiedlichen Versionen.

Angular wurde hier verwendet, da das Framework durch die Verwendung von HTML5

[5] `https://hub.docker.com`

[6] „Ein Framework ist eine semi-vollständige Applikation. Es stellt für Applikationen eine wiederverwendbare, gemeinsame Struktur zur Verfügung. Die Entwickler bauen das Framework in ihre eigene Applikation ein, und erweitern es derart, dass es ihren spezifischen Anforderungen entspricht. Frameworks unterscheiden sich von Toolkits dahingehend, dass sie eine kohärente Struktur zur Verfügung stellen, anstatt einer einfachen Menge von Hilfsklassen.“ (Ralph E. Johnson, Brian Foote: Designing Reusable Classes, "Journal of Object-Oriented Programming"(1988))

[7] `https://angular.io/`

[8] `https://news.digicomp.ch/de/2017/01/16/angular-ein-uberblick/`

in Verbindung mit TypeScript erlaubt, eine Codebasis für jede Plattform zu verwenden. TypeScript ist eine Obermenge zu ECMAScript-6 und Rückwärtskompatibel zu ECMAScript-5 und wird damit von nahezu allen modernen Webbrowsern unterstützt. Damit setzt Angular durch die Lauffähigkeit in jedem modernen Webbrowser das ursprüngliche Java-Paradigma „Write Once, Run Anywhere“[9] um.

Schließlich ermöglicht die Kombination aus HTML5 und TypeScript dem Entwickler schon bekannte Architekturkonzepte aus einer Desktop-Applikation auf den Client zu übertragen und komplexe, sowie mächtige Webanwendungen zu entwickeln. Hierfür existieren eine große Anzahl an vordefinierten Packages, beispielsweiße Material Design Components[10] , sodass eine modern gestaltete Anwendung entwickelt werden kann, in der der User schnell und einfach zurechtfindet.

[9] JavaSoft ships Java 1.0", Sun Microsystems (23.01.1996)
[10] `https://material.angular.io/`

4 Vorgehensweise

In diesem Kapitel wird die Vorgehensweise erläutert. Zu Beginn wird die Herangehensweise, wie man ein Semantic MediaWiki in einem Docker Container aufsetzen kann, ermittelt. Des weiteren wird beschrieben wie Erweiterungen allgemein installiert werden können und die Installation des Semantic MediaWikis erklärt.

4.1 Vorbereitung MediaWiki

Ein Semantic MediaWiki ist eine Erweiterung des Content-Managment-System MediaWiki. Demnach muss zunächst ein MediaWiki auf Docker-Basis erstellt werden. Gemäß dem allgemeinen „Don't repeat yourself" – Prinzips gibt es bereits eine Vielzahl an Docker Images, die direkt verwendet werden können. Eine weit verbreitete Anwendung gibt es auch als offizielles Image, die meist direkt von der Apache Community verwaltet und gepflegt werden.

Ein offizielles Image gibt es auch von MediaWiki, sodass stets eine aktuelle MediaWiki-Version, mit wenig Aufwand, verwendet werden kann. Aufbauend auf dieses Image muss dann die Erweiterung des Semantic MediaWikis installiert werden.

Installation von Erweiterungen allgemein

Allgemein gibt es zwei Möglichkeiten eine Erweiterung für ein MediaWiki zu installieren.

Der erste Weg ist „der alte". Der alte Weg, weil es der Vorgehensweise der alten MediaWiki Versionen vor 1.25 entspricht. Hierbei werden Extensions in das gleichnamige Verzeichnis kopiert und in der LocalSettings.php entsprechende Ladebefehle hinzugefügt. Ein Problem dieser Art und Weise Erweiterungen zu installieren ist, dass dabei auch immer alle benötigten Abhängigkeit mit installiert werden und zwar für jede Erweiterung. Im Zweifelsfall werden dadurch mehrere gleiche Abhängigkeiten installiert. Darüber hinaus ist ein automatisches Updaten mit wiederholten Installationen der Extension verbunden. Auch Versions-Kompatibilitäten werden nicht berücksichtigt. Dennoch wird diese Möglichkeit vermehrt in den Installationsanleitung der Extension, als erste Möglichkeit zur Installation, gennant. Diese Möglichkeit Extensions zu installieren ist auch weiterhin in neueren Versionen möglich, um ältere Extensions weiterhin zu unterstützen.

Seit Version 1.21 gibt es eine neue, präferierte, Möglichkeit Extensions zu installieren. Mithilfe des in der PHP–Welt, bekannten Werkzeugs „Composer"[11] lassen sich mehrere Extensions in einer composer.json Datei zusammenfassen. Jede Extension besitzt wiederum eine eigene composer.json, in der sie unter anderem ihre Abhängigkeiten, Metadaten und Kompatibilität mit entsprechender MediaWiki-Version angibt. Prinzipiell können über

[11] `https://getcomposer.org/doc/00-intro.md`

den Composer auch Skins installiert werden, die offizielle Liste an Composer-unterstützten Skins[12] ist jedoch sehr kurz.

Installation von Semantic MediaWiki Erweiterung

Nachdem das Basisimage für ein MediaWiki ausfindig gemacht wurde, kann jetzt die die Semantic MediaWiki Erweiterung installiert werden. Hierzu wird das Aufbauend auf das MediaWiki Image ein neues Dockerfile erzeugt. Da das Basisimage den Composer nicht installiert, wird dieser zunächst heruntergeladen und installiert. Da die Semantic MediaWiki Extension in einem offiziellen MediaWiki Repository verwaltet wird, kann sie direkt über den Composer installiert werden. Der entsprechende Befehl lautet:

```
php composer.phar require mediawiki/semantic-media-wiki \
"~2.5" --update-no-dev
```

Damit werden alle notwendigen Abhängigkeiten mit installiert. Des weiteren wird ein Skript mit in das Image kopiert, dass aufgerufen wird wenn der Container gestartet wird. Dieses Skript sorgt im wesentlichen dafür, dass beim Start des Images beziehungsweise des daraus resultierenden Containers, die Datenbanken geupdatet und weitere Extensions installiert werden. Das dadurch entstandene Image wird dann als Basisimage für das weiter Vorgehen verwendet.

4.2 Konfiguration des individuellen Semantic MediaWiki

Dieser Abschnitt befasst sich mit der Thematik wie eine individuelle Instanz des Semantic MediaWikis eingerichtet werden kann. Der zweite Abschnitt erklärt die technische Seite bezüglich der Umsetzung der Konfiguration.

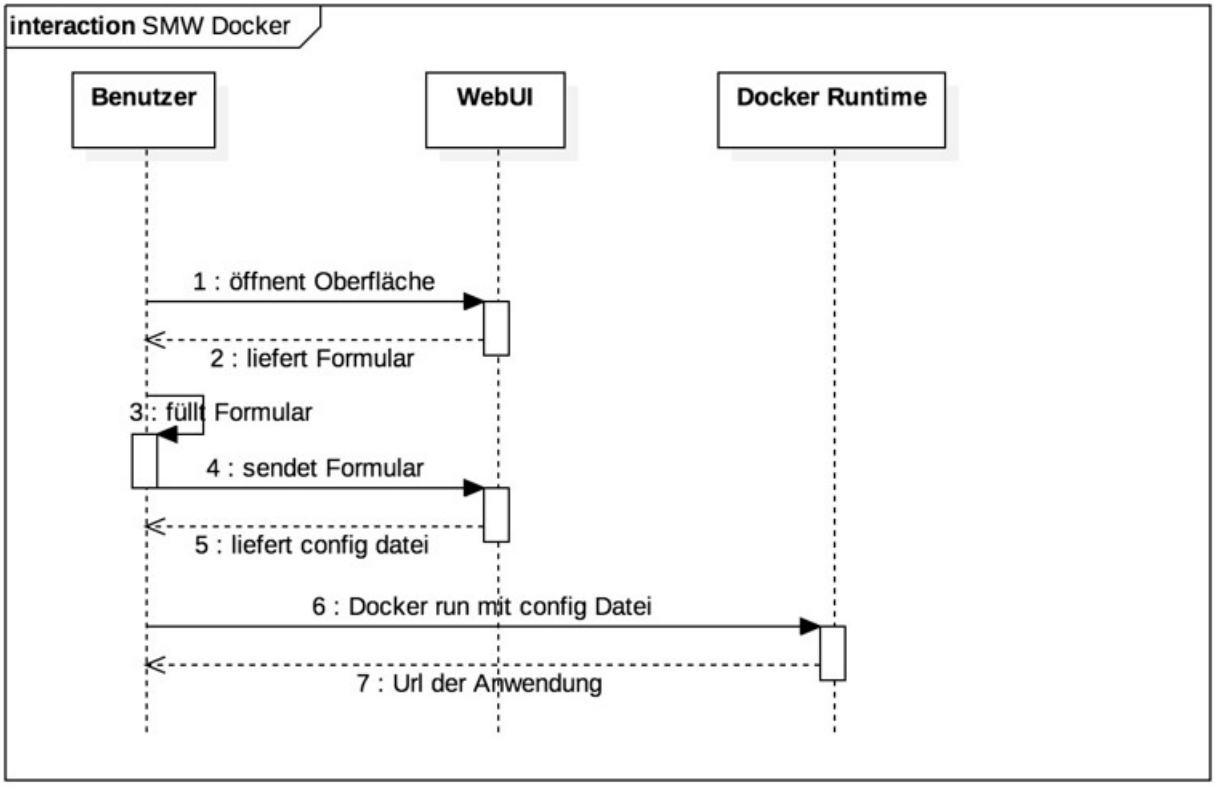
4.2.1 Konfiguration über Weboberfläche

Die entwickelte Weboberfläche erlaubt die Auswahl an Extensionsion, einem Skin und grundlegenden benutzerspezifischen Einstellungen (wie Benutzername oder Passwort). Daraus werden dann verschiedene Konfigurationsfateien erzeugt, die für individuelle Anpassung eines Semantic MediaWikis benötigt werden.

Für die Gestaltung der Weboberfläche wurden verschiedene vordefinierte Material Desgin Komponenten[13] verwendet und die Richtlinien zur Gestaltung zumindest teilsweiße umgesetzt. Die grundlegende Gestaltung, als auch der Aufbau der Oberfläche erfolgt dabei durch einen Stepper[14]. Die Idee war mittels des Steppers einen asistenzähnlichen Aufbau

[12] https://www.mediawiki.org/wiki/Category:Skins_supporting_Composer
[13] https://material.angular.io/
[14] https://material.angular.io/components/stepper/overview

Abbildung 2: schematischer Lösungsansatz



zu gestalten, welcher in einzelne (für den Benutzer) logische Schritte unterteilt ist. Jeder Schritt folgt dabei einer Ebene des MediaWikis und die ersten drei, in denen verschiedene Arten an Erweiterungen und Skins ausgewählt werden können, sind hierbei identisch- und dadurch einfach verständlich aufgebaut.

Auf der ersten Seite befindet sich der Nutzer auf der untersten Ebene. Deshalb findet er hier als erstes Vorschläge für Erweiterungen[15] vor, die direkt mit dem Installer gebündelt werden sollten, da sie sehr weit verbreitet sind. Im weiteren Verlauf sind Erweiterungen, die in den Core integriert werden sollten, aufgelistet. Für diese Entscheidung muss der Nutzer abwägen, wie hoch die Wahrscheinlichkeit, dass Funktionaltäten aus dem Core entfernt werden könnten und es somit Abhängigkeitsproblemen einstünden. Die Auswahl ist jeweils über eine Selection-List umgesetzt.

Die nächsten beiden folgenden Schritte des Steppers sind sehr vergleichbar aufgebaut, mit dem Unterschied, dass hier nur Semantic MediaWiki Erweiterungen und Skins zur Auswahl stehen.

Im vierten Schritt muss der Benutzer nun userspezifische Angaben als letzten Schritt in der Anpassung des Wikis vornehmen. Hierbei wird in der Oberfläche zwischen notwendigen

[15] `https://www.mediawiki.org/wiki/Suggestions_for_extensions_to_be_integrated`

Eingaben (die zur Konfiguration zwingend benötigt werden) und optionalen Eingaben (die sonst automatisch generiert werden) unterschieden.

Die technische Umsetzung, mit der sichergestellt wird, ob der Nutzer in jedem Schritt wirklich die benötigten Informationen richtig eingegeben hat, erfolgt mit „Reactive Forms“[16], die hier im spzeziellen als je als FormGroup umgesetzt werden. Diese sind an den Stepper gebunden und aktivieren die Buttons für weitere Konfigurationen erst, wenn die vorherigen zufriedenstellend eingegeben wurden.

Schließlich findet der Nutzer ganz unten auf der Seite ein Texteingabefeld vor. Mit diesem kann er weitere, hier nicht aufgelistete Extensions dem Installationsprozess hinzufügen. Wobei hier allerdings beachtet werden muss, dass nicht sichergestellt werden kann, dass die eingegebenen Extensions fuktionieren (genaueres ist hierzu unter Kapitel 5.1 zu finden).

Der schematische Aufbau der Konfiguration des individuellen Semantic MediaWikis ist im Sequenzdiagramm in Abbildung zwei dargestellt. Darin ist zu erkennen, dass der Benutzer die WebUI öffnet[17]. Die WebUI liefert darauf ein Formular, das der Nutzer ausfüllt (siehe oben genaueres). Zuletzt sendet der Nutzer das ausgefüllte Formular über den Button „Start MediaWiki“, woraus die Konfigurationsdateien generiert werden, die Docker für die weitere Konfiguration benötigt.

4.2.2 Umsetzung der Konfiguration zur Laufzeit

Die Weboberfläche gibt verschiedene Konfigurationsdateien, sowie eine docker-compose.yaml Datei, abgespeichert in einer Zip-Datei, aus. Diese Dateien sind notwendig um die allgemeine Semantic MediaWiki Version, individuell anzupassen. Die Konfiguration beinhaltet eine Datei extensions.json. Hierin sind alle zu installierenden Extensions und Skins angegeben. In der .env Datei sind alle notwendigen Zugangs- und Benutzerdaten für Datenbank und MediaWiki enthalten. Alle benötigten Container werden mit folgendem Befehl gestartet.

```
docker-compose up
```

Initiale Installation

An dieser Stelle kommt nun das Skript das beim Erstellen des Semantic MediaWiki Images kopiert wurde, zum Einsatz. Das check-db.sh Skript wird hierbei während des startens des Containers aufgerufen. Zunächst wird die Erreichbarkeit der Datenbank mittels Ping-Befehl überprüft. Ist der Ping nicht erfolgreich wird so lange probiert bis er positiv ist. Als nächster Schritt wird überprüft ob bereits eine LocalSettings.php vorhanden ist. Dies kann beispielsweise der Fall sein, wenn der Benutzer bereits eine MediaWiki Installation

[16] `https://angular.io/guide/reactive-forms`
[17] localhost:4200

hat und diese nun mit Docker starten möchte.

Sollte keine LocalSettings.php vorhanden sein, so wird der Installationsprozess gestartet. Dieser ist der gleiche wie die Konfigurationsmaske des MediaWikis beim ersten Start, nur mit dem Unterschied, dass in diesem Fall die Daten aus der .env verwendet werden. Der Benutzer muss daher die Konfiguration nicht händisch durchführen. Im Anschluss wurde eine LocalSettings.php erzeugt, die Datenbank initialisiert und eine Admin Benutzer angelegt.

Der nächste Schritt umfasst die Installation der weiteren Extensions und gegebenenfalls eines alternativen Skins. Hierbei wird die die extensions.json Datei mittels Shell-Skript gelesen. Die Datei besteht dabei aus zwei Bereichen. Zum einen Extensions und zum andern Skins. Jeder Eintrag besteht dabei wiederum aus drei Elementen. Als erstes dem Namen, dann der URL zum dazugehörigen Git Repository und zuletzt einer URL zur Hilfeseite der Erweiterung/Skins. Der Namen ist notwendig um die Extension später in der LocalSettings.php zu „registrieren“. Die Git-URL ist notwendig um die benötigen Dateien herunterladen zu können. Die Hilfeurl hat keine direkte Funktion, soll dem Benutzer allerdings die Möglichkeit geben sich über die installierte Extension zu informieren. Oftmals sind weitere Parameter in der LocalSettings.php nötig oder möglich um die Extension weiter anpassen zu können.

Die Installation beginnt damit, dass die Extension aus dem Git-Repository ausgecheckt wird und in das Extensionsverzeichnis kopiert. Daraufhin wird ein zur Version des MediaWikis passender Eintrag in die LocalSetings.php angelegt. Damit ist die Extension installiert. Selbes wird auch für Skins durchgeführt.

Nach der Installation wird das vom MediaWiki mitgelieferte Update-Skript aufgerufen, dass für eventuelle Änderungen oder Aktualisierungen an der Datenbank verantwortlich ist. Nachdem an dieser Stelle das MediaWiki im gewünschten Zustand ist, wird als letztes der Webserver gestartet und das MediaWiki ist unter dem konfigurierten Port unter http://localhost:[port] erreichbar.

Installation mit vorhandener Datenbank

Sollte bereits ein Backup eines vorhandenen MediaWikis vorhanden sein, so kann dieses vor dem Initialen Start in der Datenbank automatisch eingespielt werden. Hierzu muss lediglich die .sql Datei in das Verzeichnis „docker-entrypoint-initdb.d“ kopiert werden. Sollte auch bereits eine LocalSettings.php vorhanden sein, so kann diese auch verwendet werden, muss jedoch an die angebundenen Datenbank angepasst werden. Wichtig ist zudem, dass alle verwendeten Extension installierbar sind. Im Falle einer bereits vorhandenen LocalSettings.php werden dann keine initiale Installation durchgeführt sondern lediglich die Datenbank auf die aktuell verwendete MediaWiki Version aktualisiert.

5 Evaluation

Nachdem gezeigt wurde, wie es möglich ist ein dockerbasiertes MediaWiki inklusive Semantik MediaWiki Erweiterung einzurichten, werden in diesem Abschnitt die dabei aufgetretenen Probleme diskutiert. Danach werden die praktischen Anwendungen des dockerbasierten Semantic MediaWikis dargelegt, bevor mögliche Erweiterungen dieser Arbeit aufgezeigt werden.

5.1 Probleme

Installation von Erweiterungen

Zunächst einmal gibt es Probleme bei der Installation von Erweiterungen. Da es keinen einheitlichen Marktplatz für Erweiterungen gibt, ist das Finden von passenden Erweiterungen keine triviale Aufgabe. Zwar bietet die MediaWiki Webseite eine Übersicht an Extensions, jedoch ist diese Auflistung wenig übersichtlich.

Des weiteren gibt es noch keine einheitliche Installation. Viele Erweiterungen unterstützen die Installation mithilfe des Composeres nicht oder deren Installationshinweise geben keinen Hinweis, dass sie den Composer unterstützen. Dadurch bleibt nur der Blick in den Quellcode der Extension um zu erkennen ob eine Unterstützung gegeben ist. Um dieses Problem anzugehen, wurde in dieser Arbeit der Composer nicht zur Erweiterungsinstallation verwendet, sondern auf die alte Variante über kopieren in das Extensionsverzeichnis und registrieren in der LocalSettings.php vertraut.

Dieses Problem zeigt sich auch (wie schon oben angesprochen) bei der Auswahl der unterstützten Extensions, welche dem User in der UI angeboten werden. Deshalb konnte dem User nur eine rudimentäre Auswahl an Extensions direkt zur Auswahl angeboten werden, welche händisch ausgewählt wurden.

Schließlich sind auf der Webseite des Mediawiki auch noch eine Vielzahl, teils schon seit Jahren, nicht mehr unterstützte und mit der aktuellen Version des Mediawikis auch nicht mehr kompatible Erweiterungen zu finden. Deshalb werden in dieser Seminararbeit nur Erweiterungen des offiziellen Mediawiki Github-Repository[18] unterstützt.

Migration alter MediaWikis

Beim Versuch eine komplexere MediaWiki Installation mithilfe von Docker zu migrieren sind neue Probleme aufgetreten, die im folgenden kurz erläutert werden.

Zunächst einmal wurde die Datenbank eingespielt und die alte LocalSettings.php mit den neuen Zugangsdaten der Datenbank versehen. Extensions wurden mithilfe des Composers

[18] `https://github.com/wikimedia/mediawiki-extensions`

installiert. Das Problem bei der Extensionsinstallation war, dass nicht alle Extensions in öffentlichen Repositories liegen und somit nicht alle installiert werden konnten. Des weiteren klappte der erste Start nicht, da die LocalSettings.php Konfigurationen beinhaltete, sodass nicht ohne weiteres ein funktionfähiges MediaWiki zu erstellen war, beziehungsweise das Update Skript damit nicht funktionierte. Lösungsansätze waren, eine minimale LocalSettings.php zu bauen und mit dieser, die dann lediglich die Information zur Datenbank und die Standartwerte beinhaltete, ein Datenbank Update zu starten. Mit dieser Anpassung war es zumindest möglich die Inhalte der Datenbank wieder in einer Standard Mediawiki Instanz anzuzeigen.

Als Erkenntnis daraus lässt sich ableiten:

- Eine Migration ist möglich, aber mit Anpassungen verbunden
- Eine Liste mit installierten Extensions ist unumgänglich. Diese Liste muss enthalten welche Extensions über den Composer installierbar sind und welche „manuell“ installiert werden müssen, sodass eine entsprechende extensions.json und composer.json gebildet werden kann.
- manuelle Änderungen an einer frischen MediaWiki Version müssen bekannt sein, sodass diese Änderungen für jeden Docker Container nachgeholt werden können. Sinnvollerweise keine Änderungen am Grundsystem, sondern lediglich über Extensions.
- Der Docker build Agent oder der Container selber müssen Zugang zu allen verwendeten Repositories besitzen.

5.2 praktische Anwendung

Diese Arbeit hat gezeigt, dass es für den Anwender schnell und einfach möglich ist, ein Semantic MediaWiki aufzusetzen. Hierbei ist der Anwender in der Lage eine ganz persönliche Zusammenstellung an installierten Erweiterungen und Skins zu testen.

Das Ergebnis ist eine lauffähige Semantik MediaWiki Instanz die ideal zum Testen und Ausprobieren ist. Nichts desto trotz ist es auch möglich vorhandene MediaWiki Instanzen in Docker nachzubauen, sofern dabei auf Standardkomponenten zurückgegriffen wurde. Die hier vorgestellte Methode ermöglicht es darüber hinaus leicht die Wechselwirkung von verschiedenen Erweiterungen zu testen oder eine Testumgebung zur Entwicklung eigener Extensions aufzusetzen.

5.3 weiterführende Arbeit

Nachdem in dieser Arbeit die lokale Ausführung von Semantic MediaWikis in Docker Containern betrachtet wurde, ist in weiterführenden Arbeiten eine Weiterentwicklung hin zu Produktivanwendungen möglich. Hierbei liefert Docker bereits einige Handwerkszeuge mit, die das Ausrollen von Container und Clustern ermöglichen. Die von Docker selbst entwickelte Variante ist hierbei Docker Swarm[19], aber auch die in der Industrie weiterverbreitete Clusterverwaltungen, mit Mesosphere DCOS[20] und/oder Kubernetes[21], können hierfür eingesetzt werden.

Ein anderer interessanter und auch wichtiger Schritt wäre die einheitliche Verwaltung von Erweiterungen (sowie Skins). Hierbei wäre ein visuell ansprechender und intuitiv aufgebauter Marktplatz sinnvoll. Dieser sollte zumindest zu Beginn dem Nutzer erlauben, die Erweiterungen nach Installationsart zu filtern und dem Nutzer aufzeigen, welche jede Erweiterung nutzt.

Darüber hinaus könnte als Basis für den Marktplatz die hier vorgestellte UI verwendet werden. Denn dann würde es dem Nutzer nicht nur ermöglicht werden die Erweiterungen einfach zu finden, sondern auch direkt die benötigten Konfigurationsdateien für die ausgewählten Erweiterungen (sowie userspezifschen Eingaben) automatisiert zu generieren. Hieraus ergäbe sich eine große Vereinfachung, sowie Zeitersparnis.

Mittelfristig ist über die Konvertierung von von „alten“ Extension zu neuen Composerkompatiblen Erweiterungen, nachzudenken. Natürlich wäre es auch sinnvoll in Zukunft einheitlich auf die neue Installationsmethode mittels des Composers zu setzten.

[19] `https://docs.docker.com/engine/swarm/`
[20] `https://dcos.io`
[21] `https://kubernetes.io`

6 Schriftliche Erklärung

Ich, Patrick Eisele versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, 06.02.2018

Ich, Raphael Manke versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, 06.02.2018