Prototyp eines Multiinstanz-Managers für prozessvirtualisierte Anwendungen am Beispiel von OpenSlides

---

Bachelorarbeit von Jochen Saalfeld
Universität Osnabrück

Abstrakt

Das Herkömmliche Applikationshosting entspricht den wachsenden Anforderungen an dynamische Umgebungen immer weniger. Tools zur Prozessvirtualisierung und Steuerung der Prozesse machen es Entwicklern, Applikations- und Netzwerkarchitekten immer einfacher, ihre Applikationen dynamisch zu provisionieren.

Im Rahmen dieser Arbeit wird am Beispiel von OpenSlides `https://openslides.org`, einem freien, webbasierten Software Tool für die Verwaltung von Versammlungen, der Aufbau virtualisiert. Somit kann OpenSlides dynamisch und flexibel für den Einsatz bei kleinen und großen Veranstaltungen reagieren.

Die bereits existierende Plattform zur Verwaltung mehrerer OpenSlides-Instanzen wird evaluiert und an den aktuellen Stand der Technik angepasst, um Kunden OpenSlides als Software as a Service (SaaS) bereit zu stellen.

Abstract

Traditional application hosting is becoming less and less responsive to the growing demands of dynamic environments. Process virtualization and process control tools make it increasingly easy for developers, application and network architects to dynamically provision their applications.

This work uses OpenSlides `https://openslides.org`, a free, web-based software tool for the administration of meetings, as an example to virtualize the structure and thus react dynamically and flexibly for use at small and large events.

The existing platform for managing multiple OpenSlides instances are evaluated and updated to the current state of the art adapted to provide customers with OpenSlides as Software as a Service (SaaS).

Inhaltsverzeichnis

Prototyp eines Multiinstanz-Managers für prozessvirtualisierte Anwendungen am Beispiel von OpenSlides

Diese Arbeit wurde im Rahmen der Tätigkeit des Autors bei der Intevation GmbH[12] geschrieben. Der Autor des Dokuments ist seit über 2 Jahren bei der Intevation GmbH angestellt und hat während seines Studiums dort gearbeitet. Die Intevation GmbH schreibt seit 1999 Freie Software, weswegen im Rahmen der Bachelorarbeit ausschließlich freie und/oder quelloffene Software betrachtet wird. Unter anderem hat der Autor an Gpg4win[11] und OpenSlides[30] mitgewirkt.

OpenSlides ist eine Django[9] basierte Web-Anwendung zur Unterstützung von Versammlungen. Mit der Software lassen sich mehrere Arbeitsabläufe abbilden, die bei der Verwaltung und Durchführung von Veranstaltungen unterstützen. OpenSlides bietet dabei unter anderem die Möglichkeit, Redelisten, Tagesordnungen, Anträge und Wahlen zu begleiten. Hierbei können alle Beteiligten an einer Veranstaltung auf die Software zugreifen und damit in verschiedenen Rollen interagieren[30]. Intevation GmbH bietet auch Support und Hosting für OpenSlides bei Veranstaltungen an[29].

Beim herkömmlichen Applikationhosting von OpenSlides ist aufgefallen, dass bei Veranstaltungen mit vielen Teilnehmern entsprechend viel Kapazität beim Hosting der Applikation benötigt wird. Meist finden Veranstaltungen nur über einen beschränkten Zeitraum von wenigen Tagen statt. In diesen wenigen Tagen wird die gebotene Hardware gut genutzt; in den Wochen und Monaten der Vor- und Nachbereitung jedoch nicht. Hier wird die Möglichkeit gesehen, durch eine Prozessvirtualisierung der Software die Leistung besser zu nutzen, um so das Angebot für den Kunden attraktiver zu gestalten und den Fußabdruck der nötigen Ressourcen zu verkleinern.

Zuerst wird in Kapitel 1 das Problem beschrieben und die Motivation erläutert. Daran anschließend wird es in Kapitel 2 eine Einführung in das Thema geben. In Kapitel 3 der Stand der Technik erläutert. Anschließend wird in Kapitel 4 die Technologiewahl erörtert und die Neuentwicklung beschrieben. In Kapitel 5 werden die Umsetzungen an OpenSlides beschrieben. Das darauf folgende Kapitel 5.4 erläutert die Umsetzungen im Multiinstance-Manager. Abschließend gibt es im Kapitel 6 eine Zusammenfassung und einen Ausblick.

1 Problembeschreibung und Motivation

Schon früh ist bei der Bearbeitung von Hosting-Anfragen bei der Intevation GmbH für OpenSlides aufgefallen, dass vor allem unter Berücksichtigung des nötigen Datenschutzes, bei schützenswerten Informationen[37] der Hardwareaufwand recht hoch ist. Durch die besonderen Eigenschaften und Auflagen, die ein Rechenzentrum erfüllen muss, sind die Kosten deutlich höher. Eingangs wurde jede Instanz von OpenSlides auf einer eigenen Hardware zur Verfügung gestellt, damit Inhalte unter keinen Umständen in fremde Hände geraten können.

Die Konfiguration „1 Instanz = 1 Server“ bietet den höchsten Datenschutz, da jeder Mandant seine eigene Hardware hat und somit durch keine Fehler auf Daten anderer Mandanten bzw. Instanzen zugreifen kann. Jedoch ist diese Konfiguration auch die teuerste und ineffizienteste. Sie ist ineffizient, da bei Veranstaltungen die Leistung des Servers nur im Veranstaltungszeitraum selbst stark beansprucht wird. In der Vor- und Nachbereitungszeit wird hingegen nur ein Bruchteil der Leistung benötigt. Es wurde festgestellt, dass in dieser Phase weniger als $\frac{1}{10}$ der Nutzer in dieser Zeit OpenSlides verwenden. Diese Konfiguration ist zudem auch für den Kunden die teuerste Möglichkeit, OpenSlides einzusetzen, da er für den gesamten Einsatzzeitraum für einen Server bezahlen muss, von dem er über die meiste Zeit nur einen Bruchteil nutzt.

Die Motivation, den Kunden ein günstigeres Produkt anzubieten, muss jedoch weiterhin mit strengen Datenschutzrichtlinien einhergehen. Die Lösung für das Problem muss also in jeder Hinsicht der Konfiguration „1 Instanz = 1 Server“ entsprechen.

Der direkte Weg, um dies zu erreichen, ist die Prozessvirtualisierung (3.1) und Steuerung (Orchestrierung (3.2)) der virtualisierten Prozesse. Damit lässt sich eine Konfiguration von „m Instanzen = n Server“ erreichen.

2 Einführung

Im vorherigen Kapitel 1 wurde bereits kurz die Problematik umschrieben. In den folgenden Kapitel (2.1) wird zunächst die funktionale (2.1.1) und anschließend die technische (2.1.2) Seite von OpenSlides beschrieben.

Zunächst werden im Unterkapitel 2.1 die Funktionen von OpenSlides beschrieben. Dann werden in Kapitel 2.2 die Funktionen den OpenSlides-Multiinstance-Backends beschrieben, anschließend daran werden in Kapitel 2.3 die Probleme an diesem beschrieben. In Kapitel 2.4 wird die Lösung der Probleme besprochen.

2.1 Beschreibung von OpenSlides

OpenSlides wird von verschiedensten Verbänden und Vereinen eingesetzt, um Veranstaltungen zu begleiten und den Ablauf zu digitalisieren. In Abbildung 1 wird die Seite dargestellt, die ein Nutzer zuerst sieht.

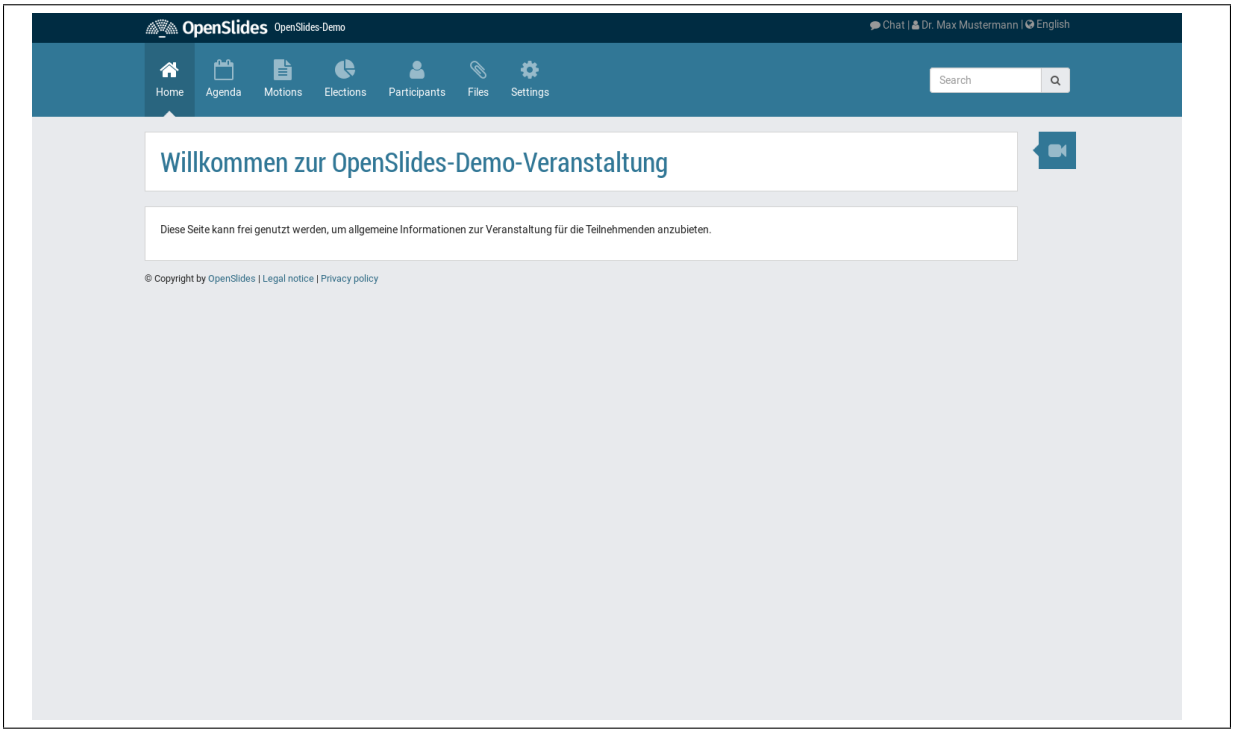

Abbildung 1: OpenSlides-Demo-Instanz „Home“-Seite [23]

Die Übersicht in Abbildung 2 ist eine Tagesordnung einer Veranstaltung. Diese kann auch zur Projektion verwendet werden und führt die Nutzer durch die Veranstaltung.

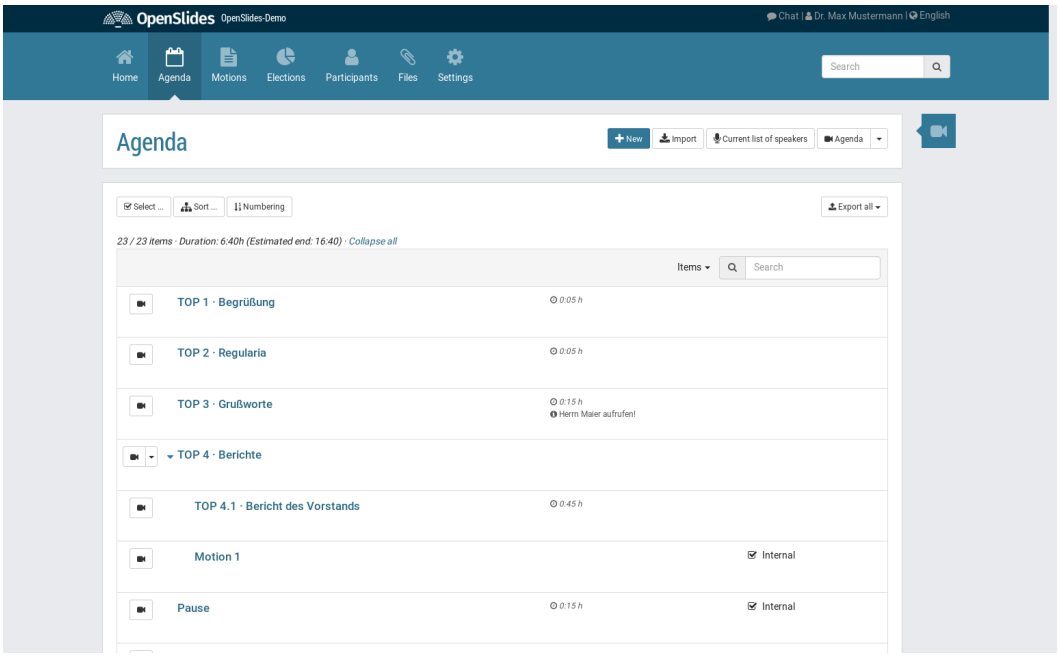

Abbildung 2: OpenSlides-Demo-Instanz „Agenda“-Seite [23]

Abbildung 3 zeigt eine beispielhafte Antragsübersicht. Diese bietet den Nutzern die Möglichkeit, eine Übersicht zu erhalten und auf einem Blick den Status der Anträge einzusehen.

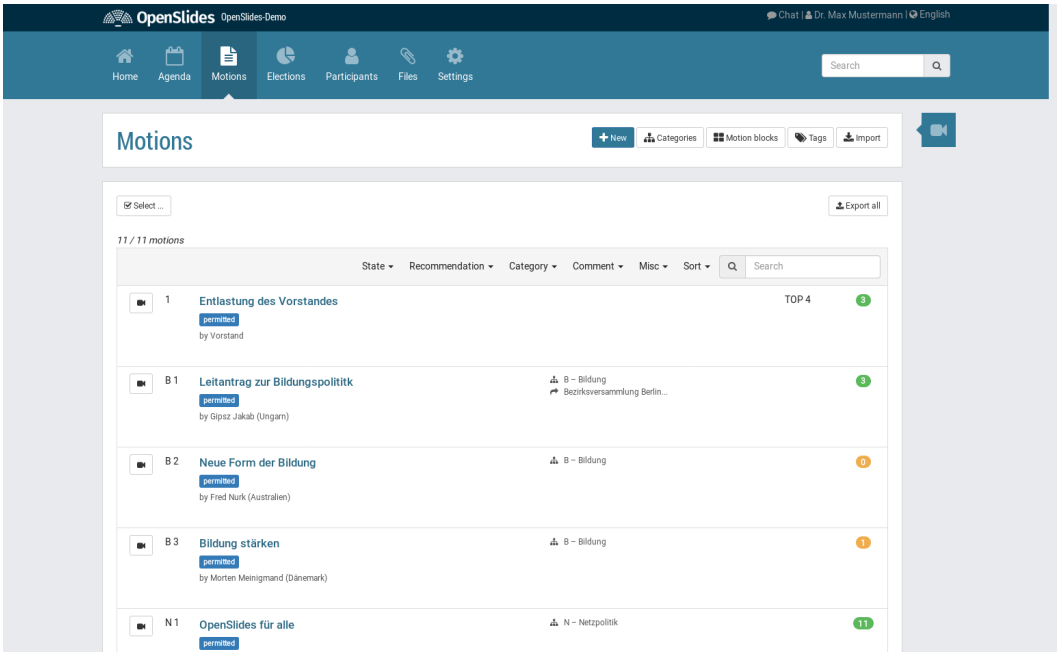

Abbildung 3: OpenSlides-Demo-Instanz „Anträge“-Seite [23]

2.1.1 Funktionale Beschreibung

OpenSlides [30] ist eine Software zur Begleitung von Veranstaltungen, wie Mitgliederversammlungen. Sie unterstützt dabei die Organisation und Durchführung in folgenden Punkten:

- Teilnehmerverwaltung
- Antragsverwaltung und -beratung (vgl. Abbildung 3)
- Tagesordnung (vgl. Abbildung 2)
- Anwesenheitskontrolle
- Durchführung und Auswertung von Wahlen
- Projektion
- Redelisten

Bei der Struktur und Mitglieder-Abbildung haben Organisationen die Möglichkeit, ihre Mitglieder in Gruppen zuzuordnen, sodass sie den Organisationsstrukturen in der Organisation selbst entspricht. Es können auch zusätzliche Gruppenzuordnungen gemacht werden, um die Adminstratoren der OpenSlides Instanz oder einzelner Funktionen festzulegen.

In der Antragsverwaltung oder Antragsberatung können verschiedene Arbeitsabläufe genutzt werden, um den Abläufen der Organisationen gerecht zu werden. Aufgrund der verschiedenen Arbeitsflüsse und Einstellungen kann hier eingestellt werden, welche Mitgliedergruppen auf welche Funktionen Zugriff haben.

Die Tagesordnung kann, wie bei Versammlungen üblich, nach Tagesordnungspunkten sortiert werden. Einzelne Tagesordnungspunkte können auch Unterpunkte enthalten. Sie sind für alle Mitglieder einsehbar.

Durch die Anwesenheitskontrolle kann man markieren, welche Mitglieder tatsächlich einer Versammlung beiwohnen. Dies kann z.B. bei der Auswertung und Begleitung von Wahlen genutzt werden, um zu prüfen, ob alle Anwesenden oder mehr als die anwesenden Personen gewählt haben. Die Auswertung und Durchführung von Wahlen ist derzeit, ohne Plugins (wie z.B. dem OpenSlides Votecollector Plugin [28]), nur analog möglich. Hierbei werden die Ergebnisse dann in OpenSlides eingetragen und können anschließend projiziert werden.

Dank der Projektion können Tagesordnungpunkte, Redelisten, Anträge und Wahlen über einen oder mehrere Beamer projiziert werden. Somit kann an einer Stelle die Versammlung geführt und geleitet werden.

2.1.2 Technische Beschreibung

Die OpenSlides-Architektur [25] kann in verschiedene Pakete herunter gebrochen werden:

- Kern
- Web-Darstellung
- Datenbank
- Zwischenspeicher
- Anfragenverarbeitung
- Webserver
- Mail-Versender

Der Kern baut OpenSlides und richtet die einzelnen Komponenten ein. Er wird in der Regel nur einmal gestartet, kann aber auch zum Upgrade auf neuere Versionen erneut gestartet werden und übernimmt dabei die Migration.

OpenSlides basiert auf Django [9] im Backend und auf AngularJS [1] im Frontend. Die im Hintergrund von der Anfragenverarbeitung (sog. `worker` [6]) generierten und ausgewerteten Inhalte werden zur Darstellung (`daphne` [6]) zum Webserver (`nginx` [17]) weitergeleitet.

Damit der `worker` die Anfragen auswerten kann, liest er sie aus dem Zwischenspeicher (`redis` [35]) und der Datenbank (`postgres` [32]) aus. Um Nutzern Informationen, wie Passwörter oder Rundschreiben aus OpenSlides zukommen zu lassen, ist auch ein Mail-Versender (`postfix` [31]) Teil des Gesamtsystems.

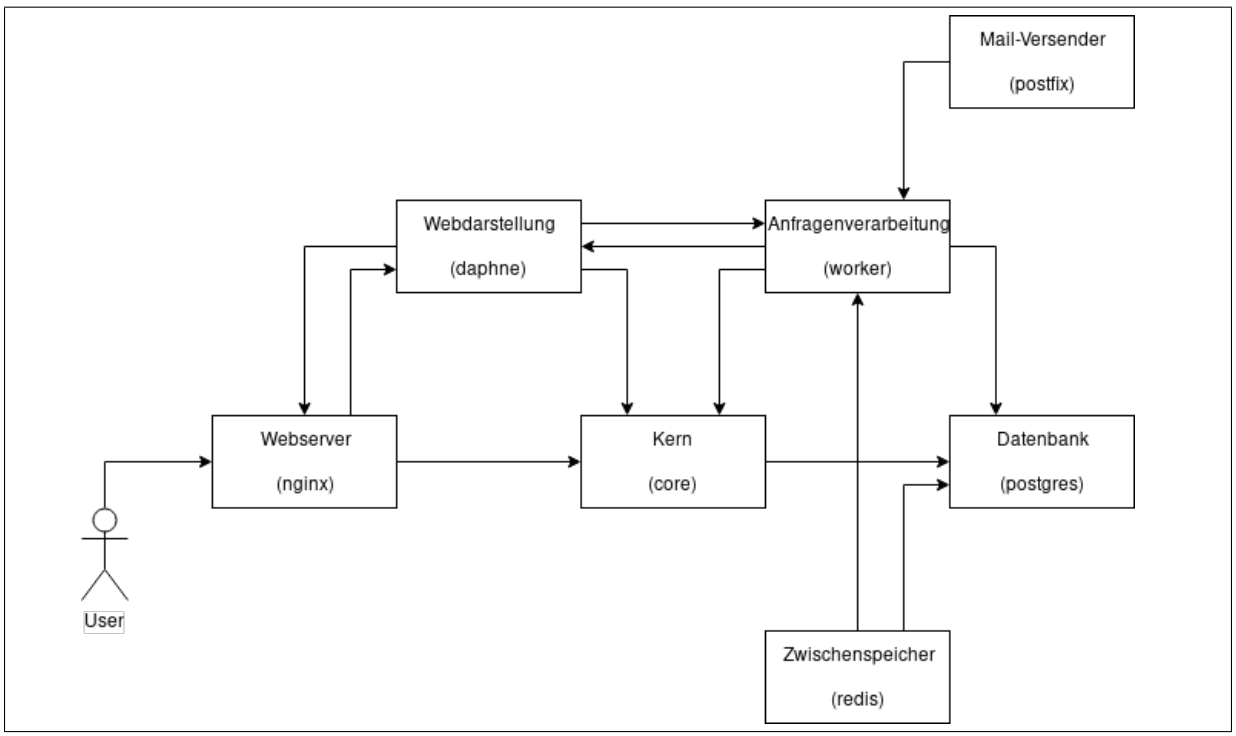

Abbildung 4: OpenSlides - Technischer Aufbau

In Abbildung 4 kann man sehen, wie die Instanz aufgebaut ist, und welche Komponenten miteinander kommunizieren bzw. wie der Fluss der Daten unter den Komponenten ist.

2.2 Funktionsbeschreibung OpenSlides-Multiinstance-Backend

Das OpenSlides-Multiinstance-Backend [26] vereint alle Teile in der technischen Beschreibung 2.1.2, bis auf die Datenbank auf einem `rkt` [5] basierenden Container. Die Container werden via `ansible` [2] gestartet, gesteuert und konfiguriert. Die Verwaltung kann über einen Einzelnutzer in einer Weboberfläche gesteuert werden.

Die Datenbanken werden für alle Instanzen zentral auf dem Server verwaltet, wobei sie sich die Software-Instanz der Datenbank-Applikation teilen. Jede Instanz erhält dabei eine eigene Datenbank mit eigenem User in dem Cluster der Datenenbank-Software. Die Software an sich ist jedoch nicht pro Instanz getrennt.

2.3 Problematik an OpenSlides-Multiinstance-Backend

Eines der größten Probleme des in Kapitel 2.2 beschriebenen Multiinstance-Backends ist die geteilte Datenbank. In Kapitel 2.1.2 wurde beschrieben, dass unter anderem Personendaten und personenbezogene Daten in der Datenbank gespeichert werden. Dadurch, dass sich alle Instanzen die Datenbank auf dem Host-Betriebssystem teilen, kann es potentiell zu einem unerlaubten Zugriff kommen. Die geteilte Datenhaltung macht es auch nicht möglich, dass wir mehrere Mandaten in dem Multiinstanz-System halten.

Des Weiteren sind die Container, in denen die Instanzen laufen, nicht an ein Überwachungssystem angeschlossen. Somit können Probleme und Fehler, die zur Laufzeit auftreten, nicht zentral ausgewertet werden.

Da die ganze Instanz immer nur in einem Container läuft, kann sie auch nicht dynamisch auf größere Anfragen reagieren. Wenn die einzelnen Komponenten ausgelastet sind, können sie nur durch manuellen Eingriff skaliert werden. Diese Konfiguration eignet sich nicht für einen dynamischen Aufbau.

2.4 Zielsetzung

Die eingesetzten Technologien im Multiinstance-Backend (2.2) sollen auch weiterhin zum Einsatz kommen. Somit soll weiterhin Prozessvirtualisierung eingesetzt werden, um eine „n Instanzen = 1 Server“ zu erreichen. Darauf aufbauend sollen Orchestrierung und Steuerung genutzt werden, um die virtualisierten Prozesse zu verwalten und eine „n Instanzen = m Server“ Konfiguration zu erreichen. Dabei sollte vor allem die geteilte Datenbank eliminiert werden, damit das System mandantenfähig wird. Dies bedeutet, dass mehrere unterschiedliche Kunden über ein System verwaltet werden und, dass sie sich die Hardware teilen können. Des Weiteren sollen die Log-Einträge der einzelnen Instanzen zentral geführt werden, und es soll eine einfache Möglichkeit geben, die Instanzen dynamisch auf Nutzeranfragen skalieren können.

Das Ziel soll sein, zu prüfen, wie das jetzige Multiinstance-Backend ersetzt werden kann. Dabei sollen die Kosten für den Kunden durch die Nutzung der gleichen Hardware und die Teilung des Systems so günstig wie möglich gehalten werden.

3 Stand der Technik

Um einen Überblick zu gewinnen, welche Technologien verwendet werden können, werden alle marktüblichen Technologien betrachtet.

In Kapitel 3.1 wird zunächst die Prozessvirtualisierung erörtert. Im darauf folgenden Kapitel 3.2 wird die Orchestrierung dieser diskutiert.

3.1 Prozessvirtualisierung

Die Virtualisierung, die Nutzung von Software und Prozessen, als auch die Ausgliederung von Abhängigkeitsmanagement in Container, ist nicht erst ein Thema der letzten Jahre. Bereits 1979, als Unix in der Version 7 entwickelt wurde, konnte `chroot` eingeführt werden. Es bietet sich als Visualisierung nicht für OpenSlides an, da chroot keine Root Privilegien Isolation unterstützt.[40]

Zwischen 1972 und heute sind viele neue Tools hinzugekommen, die in irgendeiner Art und Weise die Virtualisierung von Prozessen unterstützen [40]. Es sollen jedoch nur jene betrachtet werden, welche die Spezifikationen der Open Container Initiative (OCI)[22] unterstützen. Die OCI hat sich im Jahre 2015 aus verschiedenen führenden Technologieanbietern im Bereich der Container basierten Virtualisierung gegründet, um einen offenen Standard zu entwickeln. Dieser wird von den Technologieführern[18] entwickelt und implementiert. Im Rahmen der Betrachtung der Technolgien zur Prozessvirtualisierung werden nur Tools betrachtet, welche diesem Standard folgen. Diese Entscheidung wurde getroffen, da abzusehen ist, dass dieser Standard noch lange existiert und das auch über verschiedene Tools hinweg. Eine Zukunft ist also gesichert.

Durch den Standard, der durch die OCI gepflegt wird, unterscheiden sich die Produkte nur noch in einigen Punkten. So kann lediglich die Beschreibungssprache zum Bereitstellen eines Containers als Image unterschiedlich sein. Einmal erstellt, sind die Images durch jedes Tool nutzbar. Dann können die Images wiederum unterschiedlich durch die Werkzeuge behandelt werden. Den „Rohling“ für einen Container nennt man Image.

Zur Zeit gibt es drei große Lösungen, welche den Richtlinien der OCI folgen. Zum einen `rkt` (rock-it gesprochen), welches von den Entwicklern des Betriebssystems “CoreOS„ im Jahre 2014 veröffentlicht wurde[5]. Des Weiteren gibt es seit Kurzem `railcar` von “Oracle„, welches eine auf der Programmiersprache `rust` basierende Implementation des OCI Standards ist[34]. Weiter gibt es `docker` von der „Docker Inc.“. Sie sind Initiator der OCI, mit der Veröffentlichung im Jahre 2013 seit längster Zeit auf dem Markt und mit über 13 Milliarden Downloads (2017) Marktführer in dem Segment[38].

3.2 Orchestrierung

Da die Anwendung skalieren soll, wird ein Werkzeug benötigt, mit dem die Anzahl der Container für die einzelnen Komponenten der Applikation angepasst werden können. Dabei wurde die Applikation in 4 Images unterteilet. Zunächst wird ein `nginx`[17] Container als „Proxy“ benötigt, um die Anwendung aus dem Web über die Standard-Ports erreichbar zu machen. Dahinter liegt `daphne`[6]. `daphne` ist ein HTTP-basierter WebSocket-Protokoll-Server, der über die Channel-Technolgie aus Django die `worker`-Endpunkte anspricht, die wiederum jeweils ein Container sind. Schlussendlich gibt es noch `redis`[35] als Container, der für den Server-seitigen Cache bei den `worker`n sorgt und `postgres`[32] im Container als Datenbank.

`redis` mit dem Cache und `postgres` mit den Daten ebenso wie der Proxy und Load Balancer `nginx` sind schlecht zu skalieren. Diese Container dürfen also nur einmal pro Instanz der Applikation gestartet werden. In der Dokumentation zu Openslides[25] findet man das Beispiel, dass pro `daphne`-Instanz vier `worker` gestartet werden. Diese beiden Container können also im

Verhältnis 1 : 4 skaliert werden. Das Trennen der Instanzen in mehrere virtueller Netze und das Skalieren basiert auf Metriken (wie z.B. CPU-Auslastung oder Anzahl aktiver Nutzer) oder auf „Knopfdruck“, nennt man Orchesteriren.

Die in Kapitel 3.1 genannten Tools bringen teilweise ihre eigenen Werkzeuge zur Orchestrierung mit, so bringt `docker` die Engine `docker swarm`[10] mit. Dieses arbeitet nativ mit `docker` und der Docker-Engine zusammen. Zusätzlich gibt es `kubernetes`[15], welches ursprünglich von Google entwickelt wurde, mittlerweile aber in Cloud Native Computing Foundation übernommen wurde. `kubernetes` vereinfacht das Verteilen von `containern` und bringt, wie `ansible`[2] oder `puppet`[33], ähnliche Eigenschaften mit. Alle drei bieten Möglichkeiten zur automatischen Softwarebereitstellung, dem Konfigurations-Management und dem Applikations-Deployment; lassen sich aber auch für die Orchestrierung von Containern (teilweise mit Plugins) einsetzen. `kubernetes` und `docker swarm` sind von den genannten Tools die, welche eine gute Integration mit `docker` bieten.

4 Technologiewahl & Neuentwicklung

Durch die Spezifikationen des OCI sind viele der Technolgien untereinander kompatibel, weswegen die Wahl für ein Tool zur Prozessvirtualisierung größtenteils unabhängig von der Wahl für ein Tool zur Orchestrierung stattfinden kann. Lediglich das Orchestrierungstool Docker Swarm ist von dem Prozessvirtualisierungstool Docker abhängig.

Zunächst wird in Kapitel 4.1 die Prozessvirtualisierung besprochen. Anschließend wird in Kapitel 4.2 die Orchestrierung dieser erörtert. Abschließend wird in Kapitel 4.3 auf den Server und die Umgebung eingegangen.

4.1 Prozessvirtualisierung

Wie in Kapitel 3.1 erwähnt, unterscheiden sich die Werkzeuge `rkt`, `railcar` und `docker` nur in der Beschreibung der Images und in der Ausführung dieser. Wenn das Image dann bereit gestellt ist, sind sie untereinander kompatibel, können aber als breitgestellte Container wiederum unterschiedlich behandelt werden. Da `docker` der de-facto Industriestandard ist, werden die anderen Technolgien damit verglichen.

Bei einem Vergleich zwischen `docker` und `rkt` fallen nur wenige Unterschiede auf. Einer der bestechenden Unterschiede ist jedoch, dass Container, die mit `rkt` gestartet werden, mit Root-Rechten gestartet werden müssen. Dies ist bei neueren Versionen von `docker` nicht mehr nötig. Das minimiert das Risiko, dass Container mit Sicherheitslücken Schäden im Host-System anrichten können. Des Weiteren ist die Auswahl und Unterstützung an 3rd-Party Images unter `docker` deutlich größer, als bei `rkt`.[41]

Weiterhin ist die Struktur der Virtualisierung zwischen `rkt` und `docker` unterschiedlich (vgl. Abbildung 5). Wo bei `rkt` ein Prozess direkt gestartet wird, wird er bei `docker` über eine

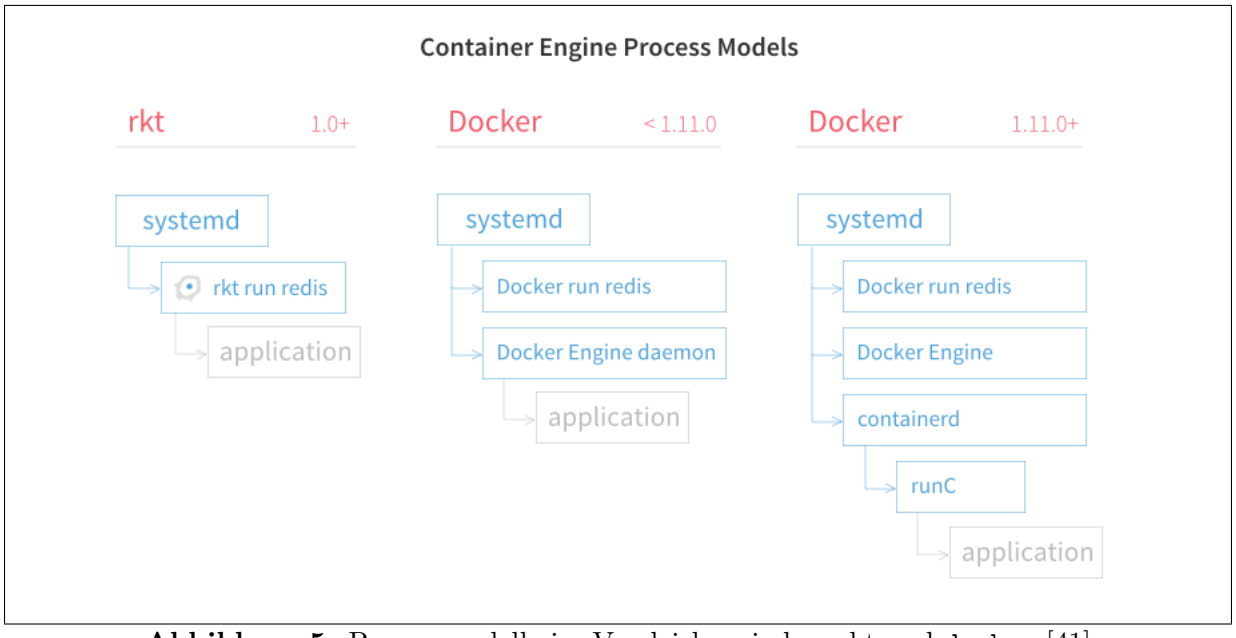

Abbildung 5: Prozessmodelle im Vergleich zwischen rkt und `docker` [41]

Engine verwaltet. Die direkte Verwaltung hat den Vorteil, dass es eine Abstraktionsschicht weniger gibt. Die Engine bietet dabei den Vorteil, dass man Containern die Möglichkeit gibt, sich selbst oder sogar andere Container in der Engine zu verwalten, ohne dass man dabei Zugriff auf das System geben muss. Die Abstraktionsebene kann also auch als Sicherheitslayer gesehen werden.

Ein Vergleich zwischen `docker` und `railcar` fällt schwer, da es keine wirklichen Unterschiede dazwischen gibt. Nach Aussage eines Entwicklers von `railcar` sind die Laufzeitkomponenten zu 98% identisch.[39] Des Weiteren ist `railcar` nur ein drop-in für die Laufzeitkomponente von OCI basierenden Applikationen.

Die Wahl zum Tool für die Prozessvirtualisierung fällt somit auf `docker`. Dieses Tool ist bereits seit einigen Jahren im Einsatz und hierzu gibt es bereits viele Lösungen und Dokumentationen durch die Community. Darüber hinaus ist `docker` bekannter, weswegen die Unterstützung bei Infrastructure as a Service (IaaS) und Platform as a Service (PaaS) Anbietern einfacher ist.

4.2 Orchestrierung

Anders wie die Werkzeuge zur Prozessvirtualisierung, gibt es für die in Kapitel 3.2 genannten Tools (`docker swarm`, `ansible`, `puppet` und `kubernetes`) keinen Standard. Schon beim Lesen der Webseiten der Produkte fällt auf, dass lediglich `kubernetes` direkt mit der Orchestrierung von Container-Engines wirbt.

In der aktuellen Lösung [26] werden `ansibile`-Skripte eingesetzt, um die Container zu verwalten. Diese Lösung basiert auf einem einzelnen Container und müsste für die Orchestrierung stark erweitert werden. Die Mittel zur Container-Orchestrierung müssten hierbei selbst geschrieben werden, wie auch bei `puppet`.[2][33]

`docker swarm` bietet keine direkte Möglichkeit, Orchestrierung zu betreiben. Anders wie `ansible` und `puppet`, ist `docker swarm` direkt in `docker` integriert und kommuniziert somit nativ mit der Container-Engine. Auch hier müssten die Werkzeuge zur Orchestrierung selbst geschrieben werden.[10]

Unterschiedlich zu allen anderen genannten Werkzeugen bringt `kubernetes` sowohl alle Tools zur Verwaltung von Containern mit, als auch alles, um die Container zu orchestrieren. Es arbeitet eng mit der Container-Engine zusammen und es kann einfach von einem `docker swarm` dorthin migriert werden, sodass auch eine lokale Entwicklung leicht fällt.[15]

Die Vorauswahl zum Tool für die Orchestrierung fällt somit auf `docker swarm` oder `kubernetes`. Auch in der Community und Wirtschaft gibt es für diese Lösung eine breite Unterstützung und `kubernetes` ist der de-facto Standard für das Orchesteriren von Applikationen. Hier bieten schon viele PaaS-Anbieter ein „`kubernetes`-as-a-Service“ an. Es können sich allerdings auch andere Tools ergeben. Je nach Dienstleister oder vom Kunden gewünschten Verfahren können auch Technologien, wie z.B. CloudFoundry [4] zum Einsatz kommen.

4.3 Server und Umgebung

Die in Kapitel 4.1 beschriebenen Prozessvirtualisierungen eigenen sich für alle Linux basierten Systeme. Es werden nur freie Linuxdistributionen in Betracht gezogen, da die Firma, bei der diese Software entwickelt wird, ausschließlich freie Software entwickelt und auch nur freie Software verwendet. Es könnten auch BSD-Distributionen genutzt werden, jedoch fehlt die nötige Expertise, um sie in einem Event sicher einzusetzen. Andere Betriebssysteme, wie Windows oder Macintosh, können nicht genutzt werden, da sie nicht frei sind.

In Kapitel 4.2 wurde noch keine Entscheidung für die Software zur Orchestrierung gefällt. Die Wahl für das Werkzeug zur Orchestrierung wurde von der Bewertung für die Server und Umgebung abhängig gemacht. Zunächst wurden einige Anbieter für PaaS und SaaS angefragt. Diese Anbieter konnten meist alle mit `kubernetes` arbeiten, mindestens aber `docker-swarm` ausführen, oder haben es direkt angeboten. Bei der Besprechung mit den Anbietern ist aufgefallen, dass OpenSlides in kleineren Rahmen skaliert, als es für große Systeme, wie `kubernetes` nötig ist. Des Weiteren waren alle Anbieter so teuer, dass der Kostenfaktor in etwa um den Faktor 10 skalierte.

Die Wahl für die Server und Umgebung ist somit auf einen normalen Root-Server gefallen, mit der Nutzung von `docker-swarm` zur Orchestrierung. Mit diesem Aufbau kann die virtuelle Architektur zunächst aufgebaut und getestet werden. Der Aufbau bringt den Vorteil mit sich, dass die Migration von `docker-swarm` zu `kubernetes` gut beschrieben ist, und ein etablierter Pfad existiert. Es wird nicht direkt `kubernetes` verwendet, da der Mehraufwand, welcher durch die Komplexität von `kubernetes` entsteht, zu groß ist.

5 Umsetzung - OpenSlides

Die Umsetzung erfolgt in mehreren Schritten. Zunächst wird OpenSlides in die einzelnen Container (5.1) aufgeteilt; diese werden dann mittels `docker-compose` (5.2) die Infrastruktur konfigurieren und diese wird mit Hilfe von `docker-swarm` (5.3) gesteuert.

In Kapitel 5.1 werden zunächst die erstellten Docker-Container erläutert, welche dann in Kapitel 5.2 zum Einsatz kommen. Das darauf folgende Kapitel 5.3 beschreibt den Einsatz in `docker-swarm`.

5.1 Docker-Container

Die einzelnen Container wurden nach dem in Kapitel 2.1.2 beschriebenen Paket entwickelt. Für einige Teile konnten dabei schon vorhandene Pakete aus der Community genutzt werden; für andere mussten diese wiederum selbst entwickelt oder angepasst werden. Dabei wird in folgende Schritte unterteilt:

- `core` (Kern von OpenSlides)
- `web` (`Daphne` zur Web-Darstellung)
- `postgres` (Datenbank)
- `redis` (Zwischenspeicher)
- `worker` (Anfragenverarbeitung)
- `nginx` (Webserver und Proxy für Daphne)
- `letsencrypt` (SSL-Zertifikatsverwaltung)
- `pg-slave` (Datenbank Backupsystem)
- `filesync` (Backupsystem für statische Dateien)
- `postfix` (Mail Versender)

Diese Übersicht wurde an Abbildung 4 angelehnt. Die Kommunikation der einzelnen Komponenten entspricht der Abbildung. Das `core` Image ist jedoch kein aktiver Teil des Aufbaus, sondern erstellt lediglich statische Daten, auf die einzelne Komponenten zugreifen, und übernimmt die Erst-Konfiguration.

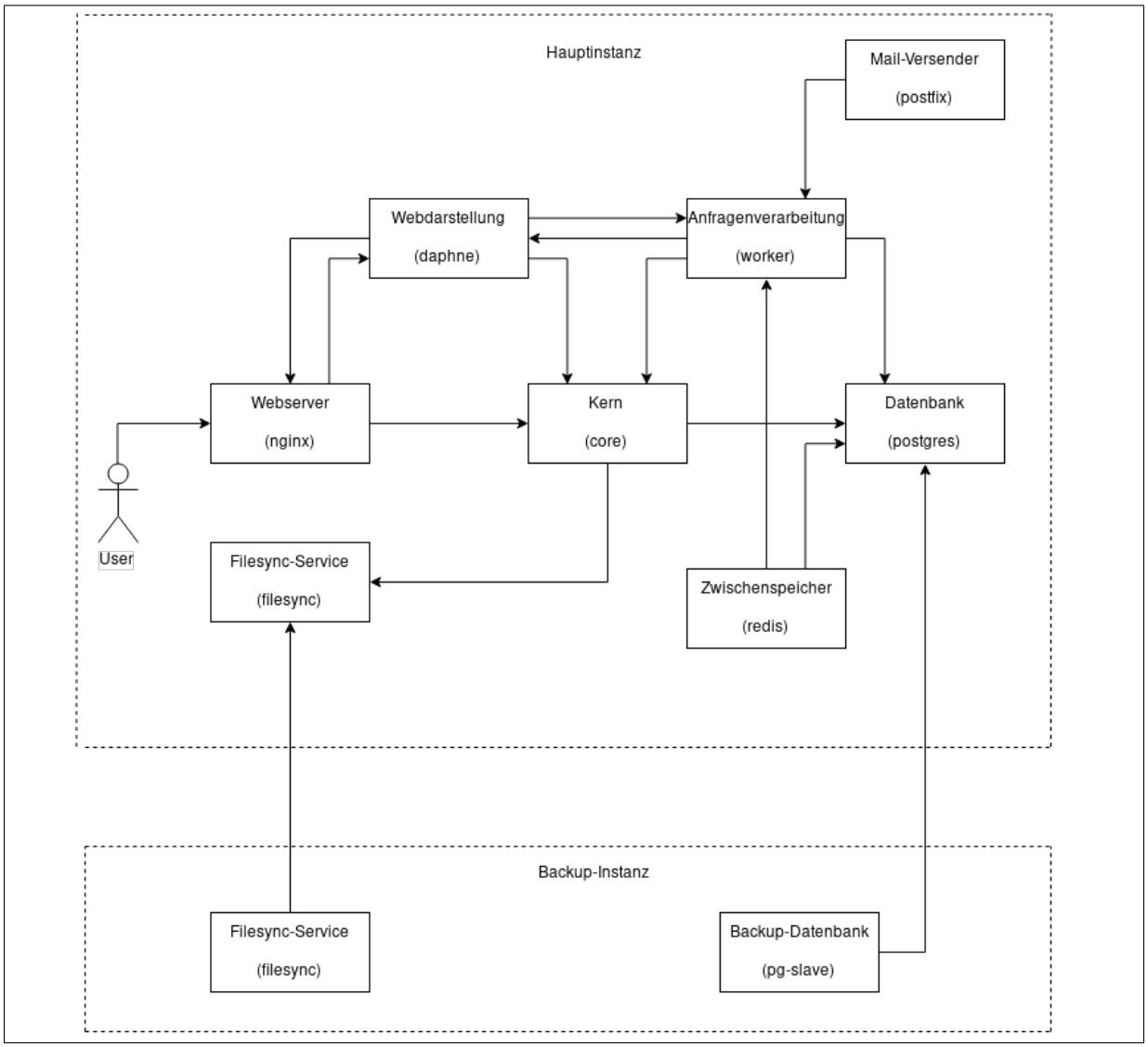

Abbildung 6: OpenSlides - Docker Container Aufbau

Abbildung 6 beschreibt den Aufbau, den eine Instanz annehmen kann. Der `core` ist dabei kein aktiver Service, sondern stellt nur passiv die Daten für andere Services bereit. Ein User greift über seinen Browser auf den Webserver zu. Dieser leitet die Anfrage weiter zur Webdarstellung. Hier werden dynamisch hinterlegte Daten angefragt, die keiner weiteren Berechnung bedürfen. Statische Inhalte werden direkt vom Webserver ausgeliefert. Anfragen, die weiterer Berechnung bedürfen, werden an die Anfragenverarbeitung weitergeleitet, welche die Daten aus der Datenbank oder dem Zwischenspeicher lädt. Wenn die Anfrage verarbeitet wurde, wird sie wieder zurück über die Webdarstellung an den Webserver und schließlich in den Browser des Users geleitet.

Es kann eine Backup-Instanz eingerichtet werden, sodass über den Filesync-Service die statischen Daten getauscht werden können und ein Streaming mit einer Backup-Datenbank stattfindet. Wenn die erste Hauptinstanz ausfällt, kann der Filesync-Service und die Backup-Datenbank abgeschaltet werden, und alle anderen Services können gestartet werden, sodass man nach kurzer Zeit wieder ein intaktes System hat.

Der `core` konnte zu einem Teil aus dem offiziellen OpenSlides GitHub Reposity genommen werden [25]. Um den Aufgaben zu entsprechen, musste er angepasst werden, sodass die Konfiguration dem Aufbau entspricht. Er richtet beim Start alle Komponenten ein und bei einem Upgrade führt er die entsprechenden Anpassungen in allen Konfigurationen durch und verteilt die neuen Daten.

Aus dem Dockerfile des OpenSlides GitHub Repositories wurde der Aufruf für den `Daphne` [25] entnommen. Dieses Image erbt alle Eigenschaften des `core` Images, führt aber keine Anpassungen durch, sondern startet nur einen Service zur Web-Darstellung. Auf die gleiche Weise funktioniert auch das Image für den `worker`. Hier wird ein Prozess zur Anfragenverarbeitung gestartet.

Das Image für die `postgres`-Datenbank konnte aus der Community entnommen werden. Zunächst wurde das offizielle `postgres`-Image verwendet [20]. Dieses bietet leider nicht die Möglichkeit, den Datenstrom zu spiegeln, damit ein Backupsystem implementiert werden kann, weswegen die Version von „sameersbn“ verwendet wird [36]. Für das `redis`-Image wurde das offizielle Image verwendet [21].

Für den Webserver und die Zertifikatsverwaltung wurde eine Entwicklung aus der Community von „jwilder“ und „JrCs“ verwendet. Diese macht es möglich, dass gestartete Container zur Webdarstellung automatisch als Route aufgenommen werden [14]. Das Image zur Verwaltung der Let's Encrypt basierten SSL-Zertifikate[16] arbeitet mit dem Image für `nginx` zusammen[13].

Um auch Mails zu versenden, wurde das `postfix`-Image von “catatnight„ aus der Community verwendet[3] und leicht angepasst, sodass es auf einem aktuellen Stand, ist und die Logging-Outputs an zentraler Stelle zusammen gefasst werden können.

Das `filesync`-Image wurde erstellt, damit Kunden Zugriff auf ihre statischen Daten für Backups haben und, um diese automatisch von einer zweiten Instanz spiegeln zu können. Dieses Image ist eine Eigenentwicklung.

Um auch ein Backup der Datenbank automatisch anlegen zu können, wurde vom gleichen Image, wie auch bei `postgres` das `pg-slave`-Image erstellt, welches die Daten aus einem anderen `postgres`-Image empfangen kann.

5.1.1 `core`, `web`, `worker`-Image

Für das `core`-Image konnte das Docker Image aus dem offiziellen OpenSlides [25] Repository zum Vorbild genommen werden. Es musste jedoch überarbeitet werden.

```
1  from python:3.5
2
3  ARG BRANCH
4  ARG REPOSITORY_URL
5  ARG COMMIT_HASH
6
7  RUN mkdir /app
8
9  RUN apt-get update && \
10   apt-get upgrade -y &&\
11   apt-get install -y libpq-dev supervisor curl vim
12 RUN wget https://nodejs.org/dist/v6.11.3/node-v6.11.3-linux-x64.tar.xz -P /tmp
13 RUN cd /tmp && tar xfvJ node-v6.11.3-linux-x64.tar.xz
14 RUN ln -sf /tmp/node-v6.11.3-linux-x64/bin/node /usr/bin/node
15 RUN useradd -m openslides
16 RUN chown -R openslides /app
17 WORKDIR /app
18
19 USER openslides
20 RUN git clone -b $BRANCH $REPOSITORY_URL .
21 RUN git reset --hard $COMMIT_HASH
22 RUN rm -rf .git
```

Listing 1: Dockerfile für den Bau des `core` - Teil 1 [24]

Die Argumente in Listing 1 in Zeile 3-5 sind in Zeile 20 und 21 dafür verantwortlich, dass eine spezifizierte Version gebaut werden kann. Unter `BRANCH` kann ein Branch eines Git-`REPOSITORY` angegeben werden. Aus diesem Branch kann dann der `COMMIT_HASH` gebaut werden. Allerdings können nicht beliebige Versionen gebaut werden. Lediglich Versionen, die den gleichen Bauablauf und die gleichen Abhängigkeiten (vgl. Zeile 9-16 in Listing 1 und Zeile 23-33 in Listing 2) haben.

```
23 USER root
24 RUN pip install -r requirements_big_mode.txt
25 RUN rm -rf /var/lib/apt/lists/*
26
27 USER openslides
28 RUN curl -o- -L https://yarnpkg.com/install.sh | bash
29 RUN $HOME/.yarn/bin/yarn --non-interactive
30
31 RUN node_modules/.bin/gulp --production
32 RUN rm -fr bower_components
33 RUN rm -fr node_modules
```

Listing 2: Dockerfile für den Bau des `core` - Teil 2 [24]

```
34 USER openslides
35 RUN python manage.py createsettings
36 RUN sed -i 's/use_redis = False/use_redis= True/' personal_data/var/settings.py
37 RUN sed -i 's#redis://127.0.0.1:6379/0#redis://redis:6379/0#' personal_data/var/
       settings.py
38 RUN sed -i 's/EMAIL_HOST =/#EMAIL_HOST =/' personal_data/var/settings.py
39 RUN sed -i 's/EMAIL_PORT =/#EMAIL_PORT =/' personal_data/var/settings.py
40 RUN sed -i 's/EMAIL_HOST_USER =/#EMAIL_HOST_USER =/' personal_data/var/settings.py
41 RUN sed -i 's/EMAIL_HOST_PASSWORD =/#EMAIL_HOST_PASSWORD =/' personal_data/var/
       settings.py
42 RUN echo "\
43 CHANNEL_LAYERS['default']['CONFIG']['hosts'] = [('redis', 6379)]\
44 " >> personal_data/var/settings.py
45
46 RUN echo "\
47 SESSION_ENGINE = 'redis_sessions.session'\
48 " >> personal_data/var/settings.py
49 RUN echo "\
50 SESSION_REDIS_HOST = 'redis'\
51 " >> personal_data/var/settings.py
52 RUN echo "\
53 SESSION_REDIS_PORT = 6379\
54 " >> personal_data/var/settings.py
55 RUN echo "\
56 SESSION_REDIS_DB = 0\
57 " >> personal_data/var/settings.py
58
59 RUN echo "DATABASES = {\
60   'default': {\
61     'ENGINE': 'django.db.backends.postgresql',\
62     'NAME': 'openslides',\
63     'USER': 'openslides',\
64     'PASSWORD': 'openslides',\
65     'HOST': 'postgres',\
66     'PORT': '5432',\
67   }\
68 }\
69 \
70 " >> personal_data/var/settings.py
71 RUN echo "\
72 EMAIL_HOST = 'postfix'\
73 " >> personal_data/var/settings.py
74 RUN echo "\
75 EMAIL_PORT = 25\
76 " >> personal_data/var/settings.py
77 RUN echo "\
78 EMAIL_HOST_USER = 'openslides'\
79 " >> personal_data/var/settings.py
80 RUN echo "\
81 EMAIL_HOST_PASSWORD = 'openslides'\
82 " >> personal_data/var/settings.py
```

Listing 3: Dockerfile für den Bau des `core` - Teil 3 [24]

In Listing 3 werden die Einstellungen in die Einstellungs-Datei geschrieben, sodass sie in dem geteilten Aufbau funktionieren. Die Parameter werden in Kapitel 5.2 erläutert. In Zeile 35 wird die Einstellungsdatei erstellt. In den beiden darauf folgenden Zeilen wird der `redis` konfiguriert. Zwischen Zeile 38 und 41 wird der Mailversand eingestellt und in Zeile 43 wird der `redis` dem `worker` bekannt gemacht. In den Zeilen 46-57 werden die Einstellungen mit dem `redis` eingestellt, ab Zeile 59 wird die Verbindung zum `postgres` beendet. Die Einstellungen zwischen Zeile 36 und 41 werden in der ursprünglichen Einstellung ersetzt. Die darauf folgenden Einstellungen werden an das Dokument angehangen, somit werden die Variablen überschreiben.

Diese Image-Beschreibungen wurden als `Dockerfile` gespeichert und können dann mit dem Befehl in Listing 4

```
docker build --tag tagname--build-arg argument=value
```

Listing 4: Befehl zum erzeugen eines Images

gebaut werden. Sobald das Image gebaut ist, liegt es in einer lokalen Image-Registry vor, und von dem Image können dann Container erzeugt werden. Die Container können über den Befehl in 5 gestartet werden.

```
docker run -p interner-port:externer-port tagname
```

Listing 5: Befehl zum starten eines Containers

Der `tagname` referenziert dabei auf das vorher erstellte Image.

Nimmt man das `Dockerfile` aus Listing 1 bis 3 von Zeile 1-35, so kann man eine einfache, nicht für produktive Einsätze geeignete, OpenSlides Version bereitstellen, welche die eingebaute Datenbank und Caching-Features nutzt. Dies wurde auch im ersten Schritt gemacht. Die Optionen nach Zeile 35 in Listing 3 sind mit den weiteren Images und deren Nutzung im Gesamtumfeld hinzu gekommen.

In Listing 3 kann man in Zeile 37 oder 43 sehen, dass als Hostname für den `redis` (analog Zeile 65 für `postgres` oder Zeile 72 für `postfix`) lediglich ein Service-Name angegeben wurde. Dies ist im Docker-Umfeld möglich, da die Docker Engine selbst einen DNS betreibt, der die Namen für die Services auflöst.

```
1 from openslides
2
3 CMD DJANGO_SETTINGS_MODULE=settings \
4   PYTHONPATH=personal_data/var/ \
5   daphne openslides.asgi:channel_layer -p 8000 -b 0.0.0.0
```

Listing 6: Dockerfile für den Bau des `web` - [24]

Die erste Zeile im Listing 6 zeigt die Vererbung von einem Image namens `openslides` an. Der in Kapitel 5.2 erklärte Aufbau, baut das in Listing 1 bis 3 und nennt es `openslides`. Somit kann der `daphne` aus Listing 6 und der `worker` aus 7 auf der gleichen Datengrundlage und mit dem gleichen Quellcode gestartet werden. Deswegen wird auch das Listing 1 bis 3 auch als `core` bezeichnet. Im Umkehrschluss bedeutet es auch, wenn ein neuer `core` eingesetzt werden soll, falls z.B. die Version verändert werden soll, müssen neben dem `core` Image auch das `web` und `worker`-Image neu gebaut werden.

```
1 from openslides
2
3 CMD sleep 5 && python manage.py runworker
4   daphne openslides.asgi:channel_layer -p 8000 -b 0.0.0.0
```

Listing 7: Dockerfile für den Bau des `worker` - [24]

5.1.2 `filesync`-Image

```
#!/bin/bash
shopt -s extglob

sleep 25

while true
do
  if [ "$REMOTE_MODE" == "SLAVE" ]; then
    echo "Start backing up static files from https://$REMOTE_HOST"
    wget --quiet --http-password=$REMOTE_PASS --http-user=$REMOTE_USER -nH https
    ://$REMOTE_HOST/static-backup/backup.zip
    echo "Done Downloading - Start copying"
    rm -rf static/*
    unzip backup.zip -d tmp
    mkdir -p static
    mkdir -p static/var
    mv tmp/static/var/* static/var
    rm -rf backup.zip
    rm -rf tmp
    chown openslides -R static
    echo "Backup Done - Sleeping for 5m"
  fi;
    if [ "$REMOTE_MODE" == "MASTER" ]; then
    echo "Start backing up static files"
    zip -r backup.zip static/var/*
    mkdir -p static/backup
    mv backup.zip static/backup
    echo "Backup Done - Sleeping for 5m"
  fi;
  sleep 600
done
```

Listing 8: run.sh für das `filesync`-Image - [24]

In Listing 8 ist ein einfaches Skript, welches die statischen Dateien einer OpenSlides Instanz alle 5 Minuten in einen Ordner namens `static` verschiebt, oder eben jene Dateien von einer anderen Instanz aus in den `static` Ordner verschiebt. Dieses Script wird in dem Image, welches aus Listing 9 gebaut wird, ausgeführt.

```
FROM debian:stretch

RUN mkdir /app

RUN apt-get update && \
apt-get upgrade -y &&\
apt-get install -y zip wget

COPY run.sh /app/run.sh
RUN useradd -m openslides
WORKDIR /app

CMD ["/app/run.sh"]
```

Listing 9: Dockerfile für den Bau des `filesync` - [24]

5.1.3 `postfix`-Image

Das Image für den Mailversand aus Listing 10 erbt, wie in Kapitel 5.1 erläutert, von „catatnight“s Entwicklung[3]. Allerdings wird es zunächst in Zeile 3 bis 5 geupdated und dann werden die Logging-Ausgaben in den allgemeinen `/dev/stdout` und `/dev/stderr` umgeleitet. Aus diesen Buffern setzt sich der allgemeine Docker-Log zusammen.

```
1 from catatnight/postfix
2
3 RUN apt-get update && \
4   apt-get upgrade -y && \
5   rm -rf /var/lib/apt/lists/*
6
7 RUN ln -sf /dev/stdout /var/log/mail.log && \
8   ln -sf /dev/stderr /var/log/mail.err
```

Listing 10: Dockerfile für den Bau des `postgres` - [24]

5.1.4 `nginx`-Image

Das `nginx`-Image ist vollständig, wie in Kapitel 5.1 beschrieben, aus der Community. Es wurden einige Einstellungen daran geändert, um den Anforderungen zu entsprechen. In Listing 11 wird die `custom_proxy_settings.conf` geändert, um die maximale Upload-Dateigröße, die übertragen werden kann, von 10 Mb auf 100 Mb zu erhöhen.

```
1 client_max_body_size 100M;
```

Listing 11: `custom_proxy_settings.conf` für das `nginx`-Image - [24]

```
1  location ~* ^/(?!ws|wss|webclient|core/servertime|core/version|users/whoami|users/
       login|users/logout|users/setpassword|motions/docxtemplate|agenda/docxtemplate|
       projector|real-projector|static|media|rest|nginx).*$ {
2    rewrite ^.*$ /static/templates/index.html;
3  }
4  location ~* ^/projector.*$ {
5    rewrite ^.*$ /static/templates/projector-container.html;
6  }
7  location ~* ^/real-projector.*$ {
8    rewrite ^.*$ /static/templates/projector.html;
9  }
10 location ~* ^/webclient.*$ {
11   rewrite ^/webclient/(site|projector).*$ /static/js/webclient-$1.js;
12 }
13 location /static {
14   alias /app/static/var/collected-static;
15 }
16
17 location /nginx_status {
18   stub_status;
19   access_log off;
20 }
21
22 location /static-backup {
23   alias /app/static/backup;
24   autoindex on;
25   auth_basic "Restricted Content";
26   auth_basic_user_file /etc/nginx/.htpasswd;
27 }
```

Listing 12: `default_location` für das `nginx`-Image - [24]

In Listing 12 wird die `default_location` umgeschrieben. OpenSlides erzeugt einige statische Dateien, die in bestimmten Ordnern abgelegt werden. Diese Einstellungen sorgen dafür, dass sie bei entsprechenden Anfragen gefunden werden. Die Anfragen für die dynamischen Inhalte werden weiter an den `daphne` gegeben. Dadurch, dass der `daphne` keine großen Daten ausliefern muss, wird das System entlastet. Des Weiteren wird zwischen Zeile 17 und 20 der `nginx_status` aktiviert, sodass man zur Laufzeit erkennen kann, wie viele Personen auf die Instanz zugreifen. Die darauf folgenden Zeilen ermöglichen es, dem `filesync`-Image Backup-Daten herunterzuladen, oder bereitzustellen. Weiter sieht man in Zeile 25 und 26, dass dieser Bereich durch die Datei aus Listing 13 passwortgeschützt ist.

```
1  openslides:$apr1$ilfQt2m.$bJ0fT2aVP971CNWcHKQ1w.
```

Listing 13: `htpasswd` für das `nginx`-Image - [24]

```
1  user  nginx;
2  worker_processes  auto;
3
4  error_log  /var/log/nginx/error.log warn;
5  pid        /var/run/nginx.pid;
6
7
8  events {
9    worker_connections  16384;
10 }
11
12
13 http {
14   include       /etc/nginx/mime.types;
15   default_type  application/octet-stream;
16
17   log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
18   '$status $body_bytes_sent "$http_referer" '
19   '"$http_user_agent" "$http_x_forwarded_for"';
20
21   access_log  /var/log/nginx/access.log  main;
22
23   sendfile        on;
24
25   keepalive_timeout  65;
26
27   include /etc/nginx/conf.d/*.conf;
28 }
29 daemon off;
```

Listing 14: `nginx.conf` für das `nginx`-Image - [24]

Es musste eine eigene `nginx.conf` hinterlegt werden, da die ursprüngliche Konfiguration lediglich 100 Verbindungen gleichzeitig haben konnte. Diese wurden in Zeile 9 des Listings 14 auf 16384 angehoben.

Anders wie im `postfix`-Image musste hier nicht die allgemeine Log-Ausgabe auf die Linux-Standard-Buffer umgelenkt werden, da dieses Image die Einstellungen schon vorgenommen hat.

Die verbleibenden Images: `letsencrypt`, `pg-slave`, `postfix` und `redis` konnten ohne weitere Anpassungen auf Dateiebene verwendet werden.

5.1.5 Entwicklungsverlauf

Zunächst wurde ein einzelnes Image gebaut, welches nur den `core`, `web` und `worker` enthielt. Da OpenSlides mit einer minimal-Datenbank paketiert ist, konnte hiermit zunächst ein erster Prototyp gebaut und eine grundsätzliche Richtung für den Bauprozess gelegt werden.

Anschließend wurde es in 2 Images aufgeteilt. Zunächst wurden `web` und `core` in einem Image und `worker` in einem anderen betrieben. Dazu wurde eine Verbindung zwischen den Images aufgebaut, sodass der `worker` die Daten aus dem `web` empfangen konnte und umgekehrt.

Im nächsten Schritt wurde zunächst das offizielle `postgres`-Image [20] verwendet, um das `postgres`-Image bereit zu stellen und das `redis`-Image wurde auch aufgesetzt. Sie wurden zunächst händisch in die jeweiligen Einstellungen eingetragen.

Um mehrere `web`-Prozesse starten zu können, wurde im letzten Schritt dieser Phase das `nginx`-Image hinzugebracht. Hier wurde auch zunächst das offizielle `nginx`-Image verwendet [19]. In diesem wurde für jeden `web` Container, der gestartet wurde, die Konfiguration entsprechend angepasst.

Das `filesync` und das `pg-slave`-Image sind erst mit der Entwicklung der `docker-compose`-Konfiguration hinzugekommen.

5.2 docker-compose

Die in Kapitel 5.1 beschriebenen Images können mit Hilfe von `docker-compose` in eine Architektur gebracht werden. Dabei können Verbindungen unter den Containern geknüpft werden. Sie können sich Dateigrundlagen teilen und in Netzwerke aufgeteilt werden. Für OpenSlides ergibt sich dabei die in Abbildung 7 aufgezeigte Architektur. Zunächst werden die logischen Festplatten

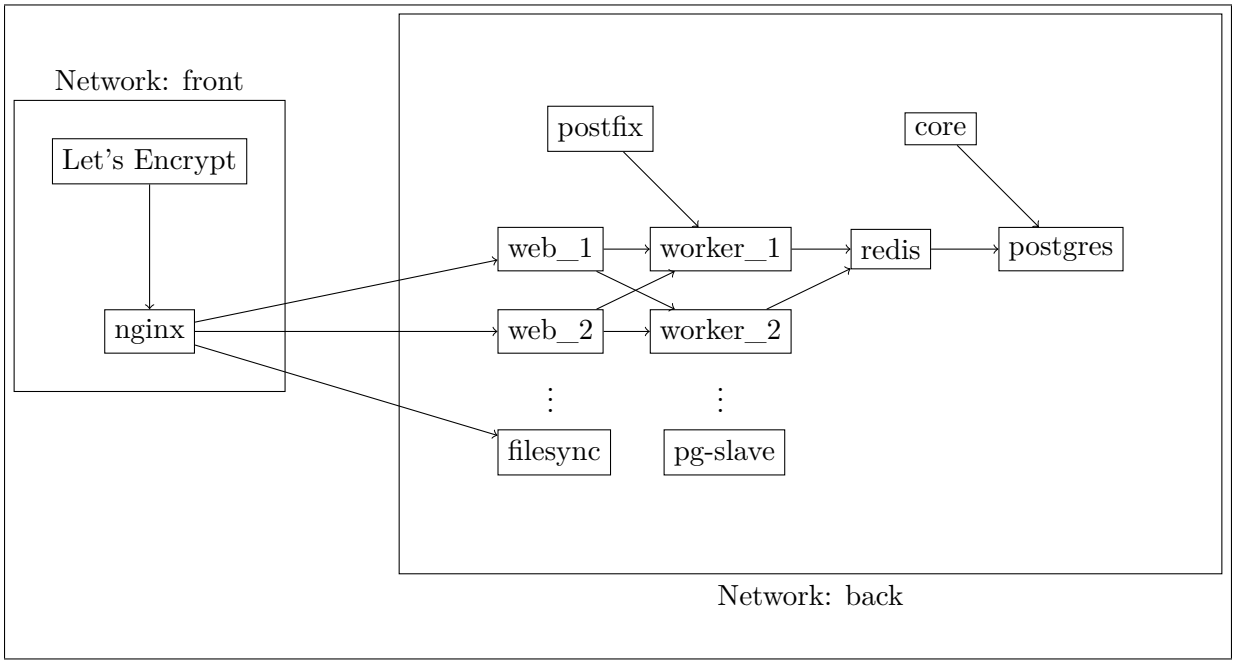

Abbildung 7: Visualisierung des Instanzaufbaus mit `docker-compose` [24]

(`voluemes`) und die Netzwerke in Listing 15 aufgeteilt. Dabei werden die logischen Festplatten einerseits dafür genutzt, Dateien dauerhaft zu speichern, andererseits, damit mehrere Container die gleiche Datengrundlage haben können.

```
152 volumes:
153   dbdata:
154   staticfiles:
155   certs:
156   redisdata:
157   nginx_vhost:
158   nginx_html:
159   nginx_conf:
160   nginx_dhparam:
161 networks:
162   front:
163   back:
```

Listing 15: `docker-compose.yml` Teil 11 - [24]

```
3  core:
4    build:
5      context: ./core
6    args:
7      # Change according to your details
8      REPOSITORY_URL: https://github.com/OpenSlides/OpenSlides.git
9      BRANCH: master
10     COMMIT_HASH: 03b17837ed2c88692f1b99ec5b9b477f86fdddb6
11   image: openslides
12   command: bash -c "rm -rf /app/personal_data/var/static && rm -rf /app/
       personal_data/var/collected-static && sleep 15 && python manage.py migrate &&
       python manage.py collectstatic --noinput"
13   depends_on:
14     - postgres
15   volumes:
16     - "staticfiles:/app/personal_data"
17   networks:
18     - back
```

Listing 16: `docker-compose.yml` Teil 1 - [24]

In Listing 16 wird der `core` Service zusammen gestellt. Hier werden die in 1 erwähnten Argumente und der Image-Name gesetzt. Im `command`-Block werden zunächst die vorhandenen Dateien entfernt und dann erneut erstellt. Bevor die Datenbank migriert werden kann, muss noch einige Zeit gewartet werden, um sicher zu stellen, dass der `postgres`-Service bereits gestartet ist. Die statischen Dateien werden dann im `staticfiles`-Volume gespeichert. Nachdem der Befehl abgeschlossen ist, schaltet sich der Service wieder ab.

```
19 web:
20   build: ./web
21   image: os-web
22   restart: always
23   links:
24     - redis
25   depends_on:
26     - core
27     - nginx
28   expose:
29     - 8000
30   environment:
31     # Change according to your details
32     VIRTUAL_HOST: 'localhost'
33     LETSENCRYPT_HOST: 'localhost'
34     LETSENCRYPT_EMAIL: 'localhost'
35     LETSENCRYPT_TEST: 'true'
36   volumes:
37     - "staticfiles:/app/personal_data"
38   networks:
39     - back
```

Listing 17: `docker-compose.yml` Teil 2 - [24]

Das Listing 17 baut den `web`-Service auf. Dieser wird mit dem `redis`-Service verbunden. Die Angaben im `environment`-Block machen dem `nginx`- und `letsencrpyt`-Service klar, dass dieser Service ansprechbar sein soll und unter welcher URL dieser erreicht werden muss. Auch dieser Service hat Zugriff auf die `staticfiles`.

```
redis:
  image: redis:alpine
  restart: always
  volumes:
    - "redisdata:/data"
  networks:
    - back
```

Listing 18: `docker-compose.yml` Teil 3 - [24]

Der `redis`-Service in Listing 18 hat keine Möglichkeit, Daten dauerhaft zu speichern, da er lediglich ein Zwischenspeicher ist.

```
worker:
  build: ./worker
  image: os-worker
  links:
    - redis
  depends_on:
    - core
  volumes:
    - "staticfiles:/app/personal_data"
  networks:
    - back
```

Listing 19: `docker-compose.yml` Teil 4 - [24]

Das Listing 19 zum `worker`-Service basiert ebenfalls auf der Datengrundlage der `staticfiles`. Er kommuniziert, anders als der `web`-Service, nur mit dem `redis`-Service.

```
nginx:
  image: jwilder/nginx-proxy
  restart: always
  volumes:
    - "/var/run/docker.sock:/tmp/docker.sock:ro"
    - "certs:/etc/nginx/certs:ro"
    - "nginx_vhost:/etc/nginx/vhost.d"
    - "nginx_html:/usr/share/nginx/html"
    - "nginx_conf:/etc/nginx/conf.d"
    - "nginx_dhparam:/etc/nginx/dhparam"
    - "staticfiles:/app/static:ro"
    - "./nginx/default_location:/etc/nginx/vhost.d/default_location"
    - "./nginx/htpasswd:/etc/nginx/.htpasswd"
    - "./nginx/custom_proxy_settings.conf:/etc/nginx/conf.d/custom_proxy_settings.
      conf"
    - "./nginx/nginx.conf:/etc/nginx/nginx.conf"
  ports:
  - "80:80"
  - "443:443"
  environment:
    - ENABLE_IPV6=true
  labels:
    com.github.jrcs.letsencrypt_nginx_proxy_companion.nginx_proxy: "true"
  networks:
    - front
    - back
```

Listing 20: `docker-compose.yml` Teil 5 - [24]

Der `nginx`-Service in Listing 20 benötigt mehrere volumes, die er sich zum Teil mit dem `letsencrypt`-Service (Listing 21) teilt. Der `nginx`-Service braucht, um Daten über eine `https` verschlüsselt bereit zu stellen, die Daten, die der `letsencrpt`-Service bereit stellt. Das `label`

dient dem letsencrpyt-Service dazu, den nginx-Service zu identifizieren. Allerdings kann auch ein eigenes Zertifikat in den Ordner gelegt werden, falls ein automatisches Erstellen über letsencrypt nicht gewünscht ist. Damit der nginx-Service automatisch neue Instanzen des web-Service finden kann, braucht dieser einen Zugriff auf den docker Unix-Socket.

```
83 letsencrypt:
84   image: jrcs/letsencrypt-nginx-proxy-companion
85   restart: always
86   volumes:
87     - "/var/run/docker.sock:/var/run/docker.sock:ro"
88     - "certs:/etc/nginx/certs:rw"
89     - "nginx_vhost:/etc/nginx/vhost.d"
90     - "nginx_html:/usr/share/nginx/html"
91     - "nginx_conf:/etc/nginx/conf.d"
92   depends_on:
93     - nginx
94   environment:
95     NGINX_PROXY_CONTAINER: nginx
96   networks:
97     - back
```

Listing 21: docker-compose.yml Teil 6 - [24]

```
98 postgres:
99   image: sameersbn/postgresql:9.6-2
100  restart: always
101  volumes:
102    - "dbdata:/var/lib/postgresql"
103  environment:
104    - DB_USER=openslides
105    - DB_PASS=openslides
106    - DB_NAME=openslides
107    - REPLICATION_USER=repluser
108    - REPLICATION_PASS=repluserpass
109  ports:
110    - "5432:5432"
111  networks:
112    - front
113    - back
```

Listing 22: docker-compose.yml Teil 7 - [24]

Die postgres und pg-slave Listings (21 und 22) speichern beide permanent in einem Ordner. Der postgres-Service ist dabei, eine normale Datenbank, und der pg-slave-Service kann von einer Datenbank, wie der im postgres-Service, die Daten streamen. Dadurch, dass beide Services in dem gleichen volume speichern, können sie untereinander ausgetauscht werden. Sie dürfen aber nie zur gleichen Zeit laufen.

```
114 pg-slave:
115   image: sameersbn/postgresql:9.6-2
116   restart: always
117   volumes:
118     - "dbdata:/var/lib/postgresql"
119   environment:
120     - REPLICATION_MODE=slave
121     - REPLICATION_SSLMODE=prefer
122     - REPLICATION_HOST=127.0.0.1
123     - REPLICATION_PORT=5432
124     - REPLICATION_USER=repluser
125     - REPLICATION_PASS=repluserpass
126   networks:
127     - back
```

Listing 23: docker-compose.yml Teil 8 - [24]

```
128 filesync:
129   build: ./filesync
130   image: os-filesync
131   restart: always
132   volumes:
133     - "staticfiles:/app/static"
134   environment:
135     - REMOTE_MODE=MASTER
136     - REMOTE_HOST=localhost
137     - REMOTE_USER=openslides
138     - REMOTE_PASS=openslides
139   networks:
140     - back
```

Listing 24: `docker-compose.yml` Teil 9 - [24]

In Listing 24 wird der `filesync`-Service definiert. Hier wird im `environment`-Block festgelegt, ob er die Rolle des `MASTER` oder `SLAVE` (vgl. Listing 8) übernimmt. Hier werden auch die Nutzerdaten aus dem Listing 13 eingetragen. Er bekommt Zugriff auf die `staticfiles`, sodass der `nginx`-Service diese ausliefern kann.

```
141 postfix:
142   build: ./postfix
143   image: postfix
144   restart: always
145   environment:
146     - maildomain=localhost
147     - smtp_user=openslides:openslides
148   volumes:
149     - "certs:/etc/postfix/certs:ro"
150   networks:
151     - back
```

Listing 25: `docker-compose.yml` Teil 10 - [24]

Der `postfix`-Service in Listing 25 hat lediglich Zugriff auf die Zertifikate, damit dieser verbindungsverschlüsselt versenden kann. Die Variablen im `environment` werden benötigt, um den Service in OpenSlides korrekt konfigurieren zu können.

5.2.1 Entwicklungsverlauf

Zunächst wurde der `core` Service aufgesetzt, wie im vorherigen Kapitel beschrieben, in der kleineren Version im Listing 1 bis 3 von Zeile 1-35.

Danach wurde der `postgres` hinzugefügt und ein Teil der Einstellungsänderung aus Listing 3 hinzugefügt. Nachdem die Verbindung zwischen den beiden erfolgreich hergestellt werden konnte, wurde der `redis` angefügt. Danach konnte der `web` und `worker` aus dem `core` heraus gelöst und als eigener Service angesetzt werden.

Anschließend wurde der `nginx` und `letsencrpyt` mit dem `postgres` Service hinzugefügt, damit eine verschlüsselte Verbindung aufgebaut werden konnte.

Schlussendlich wurden die `pg-slave` und `filesnyc`-Images aufgebaut und angefügt.

5.3 docker-swarm

Mit der in Kapitel 5.2 erstellen Konfiguration für `docker compose` kann nun ein `stack` für `docker-swarm` gebaut werden. Ein Stack besteht aus einer Konfigurationsdatei von `docker compose`. Ein Stack kann in einem `swarm` deployed werden. Der `swarm` kann aus mehreren Servern bestehen, die einzelnen Images erzeugen und innerhalb des Schwarms verteilen. Last kann somit gleichmäßig verteilt werden. Es können auch mehrere `stacks` bzw. OpenSlides Instanzen in einem `swarm` gestartet werden.

In der Arbeit wurde eine Schwarm-Umgebung, bestehend aus mehr als einem Server im Schwarm vernachlässigt. Um mehrere Server in der Schwarm-Umgebung zu betreiben, müssen alle Server des Schwarms die Images kennen, dazu muss ein Registry-Server in der Umgebung registriert werden. Auf dieser Registry können die Images veröffentlicht werden und die Teilnehmer des Schwarms laden die Images von diesem Server herunter. Damit die Registry in der Umgebung funktionieren kann, muss sie mit TLS-Zertifikaten ausgestattet werden. Ohne diese können die Images nicht von den anderen Servern im Schwarm herunter geladen werden.

Es ergibt sich eine Architektur, die in Abbildung 8 zu sehen ist. Dadurch, dass `docker-swarm`

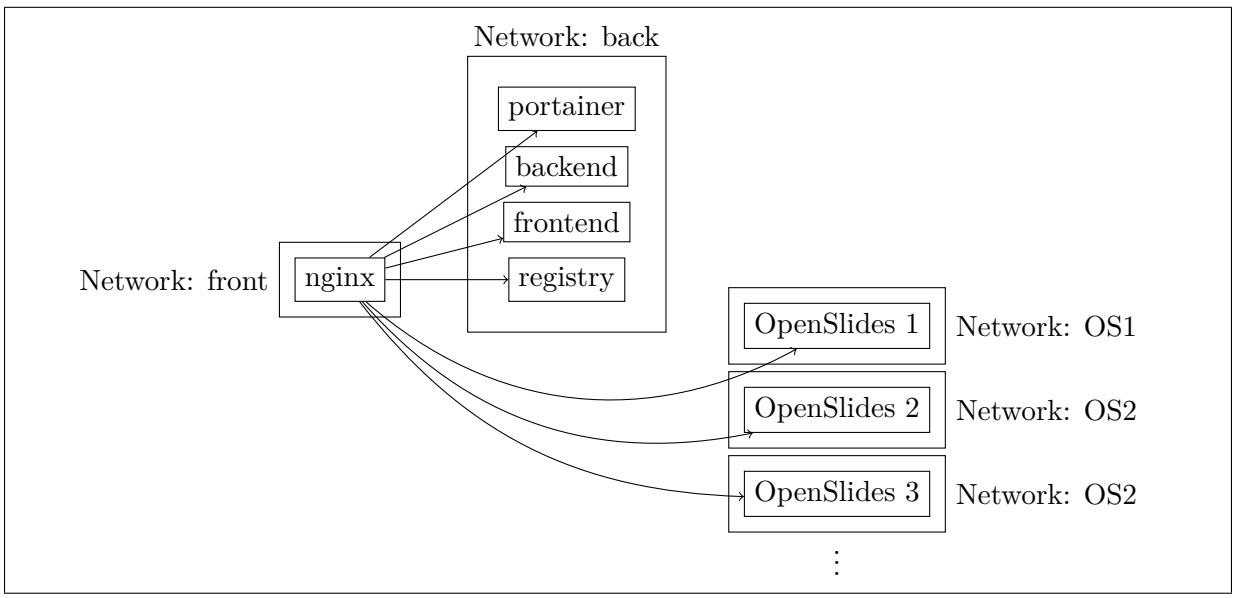

Abbildung 8: Visualisierung des Instanzaufbaus mit `docker-compose` [27]

auch auf mehreren Servern verteilt laufen kann, wird ein anderer `nginx` für das routing benötigt. Dieser hat auch Zugriff auf den Unix Sockel von Docker, er schaut jedoch danach, ob neue Stacks von OpenSlides gestartet werden und routet diese nach den Parametern aus der `docker-compose` Konfigurationsdatei.

Der `portainer` ist eine fertige Software, die bei der Überwachung und Übersicht über die Instanzen, auf der technischen Seite hilft. Hier werden die Log-Details der einzelnen Stacks zusammen gefasst, und man gewinnt eine Übersicht über den Status der einzelnen Instanzen.

Schlussendlich sind `backend` und `frontend` die Manager, welche die Konsolen-Operationen für Nutzer zur Verfügung stellen.

Da in den einzelnen OpenSlides Instanzen ein `nginx`-Image verwendet wird, welches den lokalen Docker-Socket nutzt und neue `web` Instanzen als Route aufnimmt, muss bei den Instanzen darauf geachtet werden, dass sie jeweils nur auf einem Server gestartet werden.

Um neue Instanzen bzw `stacks` in das Routing mit aufzunehmen, muss der `nginx` des Manageres zum deployen jedes neuen Stacks angepasst werden. Diese Funktionalität wird vom `backend` übernommen.

5.4 Umsetzung - Multiinstance-Manager

Im `backend` des Multiinstance-Managers werden letztendlich nur die Konsolenbefehle unter unterschiedlichen URLs mit verschiedenen Parametern bereitgestellt. Diese können durch das Frontend angesteuert werden. Das `frontend` verarbeitet und visualisiert die Antworten.

In Kapitel 5.4.1 wird die Funktion des `backend` erörtert. Das darauf folgende Kapitel 5.4.2 beschreibt das `frontend`.

5.4.1 `backend`

Zunächst werden die einzelnen Befehle und Schritte, die nötig sind, um eine neue Instanz zu starten und einzurichten, händisch durchgeführt. Diese wurden in einer `flask`-basierten `python`-Applikation in technisch zusammengehörigen Routen zusammengefasst.

Da das `backend` nur durch die Images im `back`-Network erreichbar ist und nur durch das `frontend` ansteuerbar ist, werden keine zusätzlichen Absicherungen für Aufrufe benötigt. Die Routen im `backend` sind nur aufrufbar, wenn das `frontend` übernommen wird, oder eine Angreiferin sich Zugriff auf den Server verschafft.

5.4.2 `frontend`

Das `frontend` bietet Zugriff auf die im `backend` erstellten Funktionen, um die Instanzen zu steuern oder koordinieren. Es basiert auf einer `django`-Applikation, um die Nutzer und die Zugehörigkeit dieser zu den einzelnen Instanzen, so wie deren Berechtigungen zu verwalten.

Zugriff auf das `frontend` ist nur für Personen nötig, welche die Instanzen technisch betreuen müssen. Hier wird darauf geachtet, dass die einzelnen Instanzen auch nur von den Nutzern verwaltet werden können, die die entsprechenden Rechte haben.

6 Zusammenfassung und Ausblick

In Kapitel 6.1 werden die Ergebnisse dieser Arbeit zusammen gefasst und in Kapitel 6.2 wird ein Ausblick auf kommende Entwicklungen gegeben.

6.1 Zusammenfassung der Ergebnisse

In dieser Arbeit wurde die Virtualisierung von OpenSlides in Ausblick auf einen Multiinstanz-Kontext auf der Basis von Docker entwickelt. Des Weiteren wurde ein Proof of Concept für den Multiinstanz-Betrieb erstellt.

Die Entwicklungen dieser Arbeit wurden auf GitHub unter `http://github.com/OpenSlides/openslides-docker` und `https://github.com/jsaalfeld/openslides-manager` bereit gestellt. Weitere kleinere Arbeiten wurden an OpenSlides direkt durchgeführt (`http://github.com/OpenSlides/OpenSlides`).

Zunächst wurde eine Einführung in das Thema gegeben und die Versammlungssoftware OpenSlides wurde erklärt. Anschließend wurden verschiedene Tools zur Prozessvirtualisierung und Orchestrierung erörtert und es wurde erklärt, welche Tools für die Umsetzungen verwendet werden. Darauf hin wurde die Umsetzung beschrieben.

Die bisherige Virtualisierung von OpenSlides war bislang nur in einem nicht-skalierbaren Modus möglich und basierte auf `rkt`. Diese wurde durch `docker` abgelöst und die Software-Architektur wurde auf neuere Versionen angepasst und für einen produktiven Einsatz in einem großen Umfeld umgesetzt.

Dadurch, dass die einzelnen Komponenten getrennt von einander laufen, und die Software eine einfache Aktualisierung unterstützt, ist der administrative Aufwand sehr gering, und die Kosten für den eigenen Server sind auch günstig. Somit konnte das allgemeine Ziel der Kostenreduktion erreicht werden. In Zukunft könnte hier noch besser geplant werden, wenn Kunden nicht nur Mietzeiträume angeben, sondern auch den eigentlich Durchführungszeitraum der Veranstaltung. So könnte genau geplant werden, wann und ob große Veranstaltungen zusammen fallen, sodass zusätzliche Hardware kurzfristig dazu oder abgeschaltet werden kann.

Die Grundstruktur des `docker-compose` Aufbaus, welches in Kapitel 5.2 beschrieben wurde, war bereits erfolgreich beim DGB-Bundeskongress 2018 in Berlin im Einsatz[7]. Bei vorherigen Veransaltungen vom DGB, wie dem Bundesjugendkonferenz[8] des DGB, wurde der alte Multiinstance-Server eingesetzt. Durch die Dynamik der Neuentwicklungen konnte einfacher auf plötzlich steigende Anfragen reagiert werden, und der Einsatz verlief reibungslos.

Die Entwicklungen des Multiinstance-Backend hat in einem großen Teil das bisherige Backend abgelöst. Diese Entwicklungen und bisherigen Erkenntnisse werden den anderen OpenSlides Entwicklern vorgestellt, um das weitere Vorgehen zu besprechen.

6.2 Ausblick

Die Lösung soll wartungsärmer werden und sowohl bei größeren Kunden zum Einsatz kommen als auch eine Zentralinstanz zum Anbieten von Einzelinstanzen aufgesetzt werden. Somit können Kunden günstige Einzelinstanzen angeboten werden. Hier soll auch ein automatisches Bezahlsystem implementiert werden, sodass die Instanzen so lange aktiv sind, wie für sie bezahlt wird.

Weitere Ideen für Weiterentwicklungen umfassen:

- Anbindung von Identitäsverwaltungen
- Single-Sign-On über mehrere Instanzen
- Automatisches, georedundantes Backup
- Zentrale Übersicht und Fehlersammelstelle
- Softwaregestützte Upgrademechanismen

Akronyme

IaaS Infrastructure as a Service

PaaS Platform as a Service

SaaS Software as a Service

OCI Open Container Initiative

OSMIB OpenSlides Multiinstace Backend

PPU Pay per use

Literatur

[1] *AngularJS Homepage*

[2] *Ansible Homepage*. `https://ansible.com/`. – Abgerufen: 24-08-2018

[3] *catatnight postfix docker Image GitHub Website*. `https://github.com/catatnight/docker-postfix`. – Abgerufen: 26-08-2018

[4] *Cloud Foundry Website*. `https://www.cloudfoundry.org/`. – Abgerufen: 24-08-2018

[5] *CoreOS rkt Homepage*. `https://coreos.com/rkt/`. – Abgerufen: 24-08-2018

[6] *Daphne GitHub Repository*. `https://github.com/django/daphne/`. – Abgerufen: 24-08-2018

[7] *DGB Bundeskongress 21 - Website*. `http://www.dgb.de/themen/++co++7eedf93e-6327-11e8-b4b8-52540088cada`. – Abgerufen: 30-08-2018

[8] *DGB Jugenkongress - Website*. `http://jugend.dgb.de/dgb_jugend/ueber-uns/wer-wir-sind/bundesjugendkonferenz/bjk-2017`. – Abgerufen: 30-08-2018

[9] *Django Homepage*. `https://www.djangoproject.com/`. – Abgerufen: 24-08-2018

[10] *Docker Swarm Homepage*. `https://docs.docker.com/engine/swarm/`. – Abgerufen: 24-08-2018

[11] *Gpg4win Homepage*. `https://www.gpg4win.de/`. – Abgerufen: 24-08-2018

[12] *Intevation GmbH Homepage*. `http://intevation.de/`. – Abgerufen: 24-08-2018

[13] *JrCs letsencrypt-nginx-proxy-companion Image GitHub Website*. `https://github.com/JrCs/docker-letsencrypt-nginx-proxy-companion`. – Abgerufen: 26-08-2018

[14] *jwilder nginx-proxy Image GitHub Website*. `https://github.com/jwilder/nginx-proxy/`. – Abgerufen: 26-08-2018

[15] *Kubernetes Homepage*. `https://kubernetes.io/`. – Abgerufen: 24-08-2018

[16] *Let's Encrypt Website*. `https://letsencrypt.org/`. – Abgerufen: 26-08-2018

[17] *ngix Website*. `https://nginx.org/`. – Abgerufen: 24-08-2018

[18] *OCI Member Page*. `https://www.opencontainers.org/about/member`. – Abgerufen: 24-08-2018

[19] *Official nginx Docker Image GitHub Website*. `https://github.com/nginxinc/docker-nginx`. – Abgerufen: 30-08-2018

[20] *Official Postgres Docker Image Github Website*. `https://github.com/docker-library/docs/tree/master/postgres/`. – Abgerufen: 26-08-2018

[21] *Official Redis Docker Image GitHub Website*. `https://github.com/docker-library/redis`. – Abgerufen: 26-08-2018

[22] *Open Container Initiative*. `https://www.opencontainers.org/`. – Abgerufen: 24-08-2018

[23] *OpenSlides Demo Instance*. `https://demo.openslides.org/`. – Abgerufen: 30-08-2018

[24] *OpenSlides-Docker GitHub Website*. `https://github.com/OpenSlides/openslides-docker`. – Abgerufen: 26-08-2018

[25] *OpenSlides Github Page*. https://github.com/OpenSlides/OpenSlides. – Abgerufen: 24-08-2018

[26] *OpenSlides Multiinstance Backend GitHub Page*. https://github.com/OpenSlides/openslides-multiinstance-backend. – Abgerufen: 24-08-2018

[27] *OpenSlides Multiinstance Manager GitHub Website*. https://github.com/jsaalfeld/openslides-manager. – Abgerufen: 26-08-2018

[28] *OpenSlides Votecollector Plugin*. https://github.com/OpenSlides/openslides-votecollector. – Abgerufen: 25-08-2018

[29] *OpenSlides.com Homepage*. https://openslides.com/. – Abgerufen: 24-08-2018

[30] *OpenSlides.org Homepage*. https://openslides.org/. – Abgerufen: 24-08-2018

[31] *Postfix Homepage*. http://www.postfix.org/. – Abgerufen: 30-08-2018

[32] *Postgres Website*. https://www.postgresql.org/. – Abgerufen: 24-08-2018

[33] *Puppet Homepage*. https://puppet.com/. – Abgerufen: 24-08-2018

[34] *RailCar GitHub Homepage*. https://github.com/oracle/railcar. – Abgerufen: 24-08-2018

[35] *Redis Website*. https://redis.io/. – Abgerufen: 24-08-2018

[36] *samersbn Postgres Docker Image Github Website*. https://github.com/sameersbn/docker-postgresql. – Abgerufen: 26-08-2018

[37] *Was sind personenbezogene Daten?* https://www.datenschutz.org/personenbezogene-daten. – Abgerufen: 24-08-2018

[38] Bernheim, Laura: Docker's Tools of Mass Innovation: Explosive Growth From Open-Source Containers to Commercial Platform for Modernizing and Managing Apps. (2017). http://www.hostingadvice.com/blog/dockers-explosive-growth-from-open-source-containers-to-commercial-platform/. – Abgerufen: 24-08-2018

[39] Handy, Alex: Oracle Releases an OCI-Based Container Runtime. (2017). https://thenewstack.io/oracle-opens-oci-container-runtime/. – Abgerufen: 24-08-2018

[40] Osnat, Rani: A Brief History of Containers: From 1970s chroot to Docker 2016. (2016). https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016. – Abgerufen: 24-08-2018

[41] Upguard: Docker vs CoreOS Rkt. (2017). https://www.upguard.com/articles/docker-vs-coreos. – Abgerufen: 24-08-2018

Erklärung zur selbstständigen Abfassung der Bachelorarbeit

Ich versichere, dass ich die eingereichte Bachelorarbeit selbstständig und ohne unerlaubte Hilfe verfasst habe. Anderer, als der von mir angegebenen Hilfsmittel und Schriften, habe ich mich nicht bedient. Alle wörtlich oder sinngemäß der Schriften anderer entnommenen Stellen habe ich kenntlich gemacht.

. . . . . .

Ort, Datum, Unterschrift