HOCHSCHULE KONSTANZ TECHNIK, WIRTSCHAFT UND GESTALTUNG
UNIVERSITY OF APPLIED SCIENCES

CAD-Project - Mom based Information Live Flow

Paul Drautzburg, Lukas Hansen, Georg Mohr, Kim De Souza, Sebastian Thuemmel, Sascha Drobig

Vorwort Das vorliegende Dokument beschreibt die Umsetzung für das Projekt im Rahmen des MSI-Kurses *Cloud Application Development*.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Motivation

Als Anwendungsszenario haben wir uns für ein Wetterdatensystem entschieden. Hierzu benutzen wir die Wetter-API von "OpenWeatherMap". Unser System gliedert sich in die 6 verschiedenen Komponenten WetterAPI, MoM, CEP, Datenbank, Android User Client und ein User Client Webinterface. Mithilfe der WetterAPI werden die Wetterdaten für bestimmte Städte innerhalb Deutschlands über die MoM an die CEP geschickt. Die CEP verarbeitet und berechnet anhand der eingegangenen Wetterdaten bestimmte Benachrichtigungen (Alerts) für die User. Diese Alerts werden wiederrum über die MoM an die User verteilt. Zudem besteht die Möglichkeit sämtliche Wetterdaten und Alerts in einer Datenbank abzulegen. Ein Client User ist entweder per Android App oder Webinterface mit dem System bzw. der MoM verbunden. Im folgenden Schaubild ist das Zusammenspiel der einzelnen Komponenten noch einmal grafisch visualisiert:

KOMPONENTEN BILD Sebastian T

Unser WetterAPI System stellt den User Clients neben einer Darstellung der tagesbezogen Wetterdaten, eine Wettervorhersage der nächsten fünf Tage zur Verfügung. Im Unterschied zu herkömmlichen Wetterdiensten, bietet unser System zusätzlich noch Alerts an. Damit können wir den User auf plötzliche Wetterumbrüche und beispielweise vorzeitig Gefahrensituationen hinweisen und Empfehlungen aussprechen. Da der Traffic unseres Systems stark von den verbundenen Userzahlen und dem unkontrollierbaren Wetter abhängt, muss das System auf verschiedene Lastsituationen angemessen reagieren können. Dies sind optimale Voraussetzungen für eine Lösung auf Basis eines Cloud Systems.

1.1 Zielsetzung

ToDo: Verweis auf 12 Faktor APP Standard !!! Die Tabelle soll am Anfang stehen, am besten in der Zielsetzung. Die folgende Tabelle beschreibt die Kernanforderungen der 12 Faktor APP,

12 Faktor APP Anforderungen		
ID	**Anforderung**	**Beschreibung**
1.	Codebase	Eine im Versionsmanagementsystem verwaltete Codebase, viele Deployments.
2.	Abhängigkeiten	Abhängigkeiten explizit deklarieren und isolieren.
3.	Konfiguration	Die Konfiguration in Umgebungsvariablen ablegen.
4.	Unterstützende Dienste	Unterstützende Dienste als angehängte Ressourcen behandeln.
5.	Build, release, run	Build- und Run-Phase strikt trennen.
6.	Prozesse	Die App als einen oder mehrere Prozesse ausführen.
7.	Bindung an Ports	Dienste durch das Binden von Ports exportieren.
8.	Nebenläufigkeit	Mit dem Prozess-Modell skalieren.
9.	Einweggebrauch	Robuster mit schnellem Start und problemlosen Stopp.
10.	Dev-Prod-Vergleichbarkeit	Entwicklung, Staging und Produktion so ähnlich wie möglich halten.
11.	Logs	Logs als Strom von Ereignissen behandeln.
12.	Admin-Prozesse	Admin/Management-Aufgaben als einmalige Vorgänge behandeln.

Tab. 1: 12 Faktor App Anforderungen

1.2 Die 12 Faktor-APP Anforderungen

2 Einleitung

Im heutigen Zeitalter gewinnen verteilte Systeme immer mehr an Bedeutung. Um zeitgerechten Performanceanforderungen gerecht zu werden, setzen viele Entwickler und Firmen auf Cloud Architektur Lösungen, um ihre Produkte und Softwarelösung für Kunden zugänglich zu machen. Verschiedene Anbieter haben sich in den vergangenen Jahren etabliert und bieten inzwischen eine Reihe von Services an. Obwohl sich der Rahmen der Cloud Angebote stark unterscheidet, lassen sich die Vorteile des Cloud Computing im Allgemeinen herausstellen. Neben Rechen- und Speicherkapazität, lassen sich auch Dienste anmieten, wodurch eine längerfristige Kapitalbindung für benötigte Hardware vermieden werden kann. Des Weiteren werden die Hardwarekomponenten durch den Cloud Anbieter in der Regel auf dem neusten Stand der Technik gehalten, was für viele Unternehmensstrukturen im Vergleich zu eigener Hardware rentabel ist. Je nach Vertragsabkommen kann man auch auf die IT Expertise der Cloud Anbieter zurückgreifen und verringert somit die Abhängigkeit von eigenen IT-Mitarbeitern für Aufbereitung und Instandhaltung seines Systems. Zum Beispiel kann die Verantwortung für Performanceattribute wie unterbrechungsfreie Strom Versorgung (USV), Security und SLA auf den Cloud Anbieter übertragen werden. Der wertvollste Vorteil einer Cloud Architektur liegt in der Skalierbarkeit der Dienste und der zugrunde liegenden Hardware. Über konfigurierbare Einstellungen, lassen sich die Systemkomponenten ja nach Nutzungsgrad skalieren. Dies ermöglicht schnelle Reaktionen auf Wachstum oder Nutzungsspitzen. Das Angebot lässt sich in die Bereiche Infrastructure as a Service (IaaS), Platform as a Service (PaaS) und Software as a Service (SaaS) unterteilen, welche in einer Vielzahl von Variationen zur Verfügung gestellt werden. Um zukünftigen, beruflichen Aufgaben eines Cloud Entwicklers gerecht zu werden, wird in der Fachrichtung Software Engineering des Masterstudiengangs Informatik an der HTWG das Fach Cloud Application Development angeboten. Der Kurs soll helfen die komplexen Strukturen eines Cloud Systems kennenzulernen und zu beherrschen. Im Rahmen einer Projektarbeit sollen wir Studierenden den Umgang mit den Architekturelementen erlernen. Nach der Definition eines geeigneten Anwendungsszenarios sollen die Studierenden eine funktionierende Cloud Architektur einrichten und testen. Diese Dokumentation beschreibt die Umsetzung der Projektarbeit und ist Teil der Bewertung für das Studienmodul Cloud Application Development. Neben einer ersten Erfahrung sind die Studierenden in der Lage Angebote und Dienste im Bereich Cloud Computing differenziert zu betrachten und anhand entscheidender Kriterien für unterschiedliche Anwendungsfälle zu bewerten.

3 Kommunikation der Komponenten

Die entwickelte Anwendung besteht mit einem Java-Service für die verwendete Wetter-API, der Complex Event Processing Engine und den Anwender-Clients aus drei Komponenten. Diese Komponenten müssen möglichst stark entkoppelt miteinander kommunizieren können. Durch eine starke Entkopplung wird erreicht ,dass die jeweiligen Komponenten keine Kenntnisse über vorhandene Schnittstellen oder die verwendete Programmiersprache besitzen müssen. Um dies zu realisieren, wird RabbitMQ als Message-oriented Middleware (MoM) eingesetzt.

3.1 RabbitMQ

Bei RabbitMQ handelt es sich um einen auf Erlang basierenden OpenSource Message Broker, welcher Bibliotheken für alle gängigen Programmiersprachen wie Java, JavaScript, Swift und C# anbietet. Dadurch wird die Kommunikation mit Android-, iOS- und Webapplikationen möglich. Durch die Verwendung von Queues und Topics wird die asynchrone Verteilung der Nachrichten ermöglicht. RabbitMQ verwendet als Standard das Messaging Protokoll AMQP, bietet aber Plugins für alternative Protokolle wie MQTT und STOMP. Da auch mobile Geräte zu den eingesetzten Komponenten gehören, wird das Protokoll MQTT eingesetzt, da dieses speziell für den Einsatz in Bereich der Mobilgeräten entwickelt wurde. Die Kommunikation der einzelnen Komponenten erfolgt mit MQTT über Topics. Damit der Nachrichtenaustausch stattfinden kann, müssen sich die miteinander kommunizierenden Komponenten auf ein oder mehrere gemeinsame Topics einigen. Der Aufbau eines Topics ist mit REST-Schnittstellen vergleichbar und kann aus mehreren Topic-Leveln bestehen. Zusätzlich können beim Abonnement von Topics Platzhalter wie + und # eingesetzt werden. Diese funktionieren wie reguläre Ausdrücke und ersetzen im Falle des Platzhalters + eine einzelne Topic-Ebene und beim Platzhalter # alle nachfolgenden Ebenen. Die Topics und mögliche Abonnements dieser Anwendung sind nachfolgend aufgelistet:

78467/today
: Abonnement des Wetters von Postleitzahl 78467 des heutigen Tages

+/today
: Abonnement des Wetters aller verfügbaren Postleitzahlen des heutigen Tages

78467/today/alert
: Abonnement der Wetterwarnungen für die Postleitzahl 78467

+/weekly
: Abonnement der Vorhersage der nächsten Woche aller verfügbaren Postleitzahlen

#
: Abonnement aller verfügbaren Topics

Damit Daten durch einen Client versendet oder empfangen werden kann, muss er sich beim Verbindungsaufbau authentifizieren und für den Zugriff auf das entsprechende Topic autorisiert sein. Die Authentifizierung erfolgt über eine gewöhnliche Benutzername / Passwort - Abfrage. Um den Zugriff auf MQTT-Topics zu beschränken, ermöglicht RabbitMQ die Verwendung virtueller Hosts (vHosts). Durch diese erlangen die Nutzer nur Zugriff auf ein Topic, wenn sie für den vHost des Publishers autorisiert sind. Die Erstellung neuer Nutzer und die Verwaltung der Rechte erfolgt über unter ande-

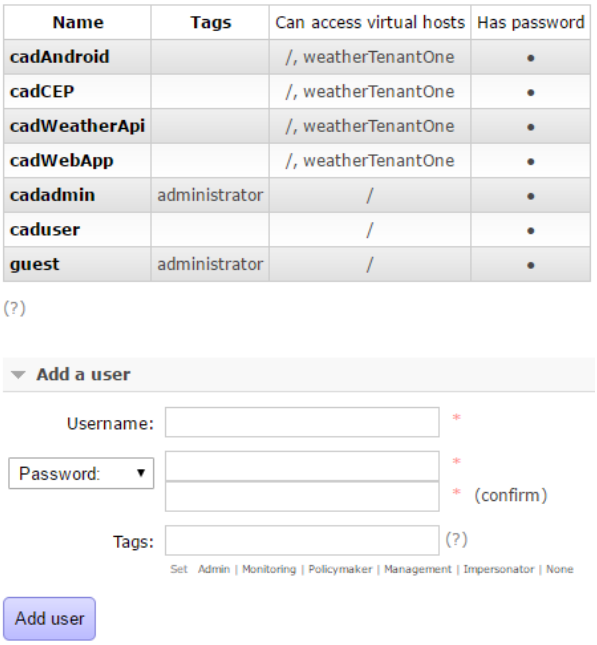

Abbildung 1: Administrationsmenü zur Benutzererstellung

rem über das Management Plugin. In Abb. 1 ist ersichtlich, dass die User *cadAndroid*, *cadCEP*, *cadWeatherApi* und *cadWebApp* Zugriff auf den gemeinsamen vHost weatherTenantOne haben. Durch dieses Verfahren kann das gleiche Topic von mehreren Nutzern mit unterschiedlichen vHosts verwendet werden, ohne das sie die Nachrichten anderer vHosts des gleichen Topics lesen können. Eine Übersicht über die einzelnen

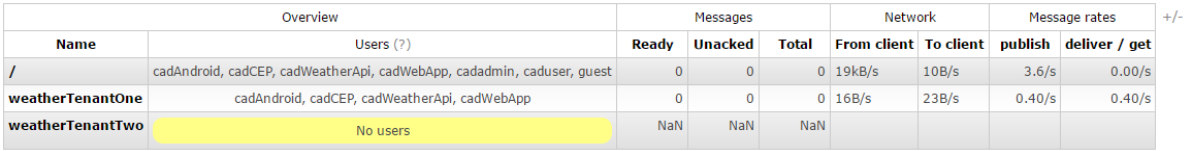

Abbildung 2: Übersicht über die virtuellen Hosts

vHosts und die berechtigten Nutzer wird in Abb. 2 dargestellt. Diese zeigt noch einmal die erwähnte Zugriffsbeschränkung auf die vier Nutzer dieses Use-Cases sowie den aktuell verursachten Datentransfer der vHosts. Auf diese Weise erfüllt die Anwendung die Anforderung der Multi-Tenancy. Die Persistierung der Nutzerdaten erfolgt in die

Rabbitmq-eigene Datenbank. Leider erfolgt die Persistierung in Verbindung mit dem Hostnamen der aktuellen Instanz. Dies führt dazu, dass bei der Änderung des Containers die Task Definition aktualisiert werden muss und dadurch automatisch die Instanz neu gestartet wird, wodurch sich der Hostname ändert und die Datenbank leer gestartet wird. Dieses Problem wird bereits sehr häufig thematisiert. Leider konnte keiner der Lösungsvorschläge das Problem beheben. Da dieser Fall nicht häufig auftritt und der Rabbitmq-Container kein Teil des Continuous Delivery Prozesses ist, wurde durch das Team eine Lösung entwickelt, um die dadurch entstandenen Probleme zu minimieren. Die Umsetzung wird in Absatz 4.2.2 erläutert. Zusätzlich zum Management Plugin bietet RabbitMQ eine HTTP-Schnittstelle, diese ermöglicht es dem Administrator zum einen über eine Kommandozeile in Verbindung mit Kommandozeilenprogrammen wie cURL (Client for URLs) die angebotenen Schnittstellen aufzurufen und dadurch unter anderem Nutzer anzulegen oder die Verbindungsraten der vHosts auszugeben und auszuwerten (vgl. https://pulse.mozilla.org/api/). Aufgrund der vorhandenen Schnittstellen bietet sich dem Entwickler die Möglichkeit, die Administration über eine eigene Applikation durchzuführen.

3.2 12 Faktor App

In diesem Absatz wird dargestellt, ob und wie die Anforderungen umgesetzt wurden.

Validierung nach "12 Faktor APP"			
ID	**Anforderung**	**Validierungs Element**	**Erfüllt**
1.	Codebase	Andere Komponenten sind nur für das Testszenario da. Deployment verschiederner Versionen über Repo möglich	Ja
2.	Abhängigkeiten	Keine Abhängigkeiten zu anderen Komponenten	Nein
3.	Konfiguration	Konfigurationsdatei wird beim Start des Containers aufgerufen und schränkt den Gast-Zugang auf localhost ein. Credentials werden über Umgebungsvariablen im Amazon Container Service (ACS) gepflegt.	Ja
4.	Unterstützende Dienste	Keine unterstützenden Dienste vorhanden	Nein
5.	Build, release, run	Könnte durch Jenkins verwaltet werden. Da die MOM keine regelmäßigen Update-Zyklen durchläuft, wurde darauf verzichtet.	Möglich, nicht aktiv
6.	Prozesse	Der Start des RabbitMQ wird durch das init-Skript im Dockerfile gesteuert. Dieses wird durch dem CMD-Befehl automatisch ausgerufen, so dass der Anwender nur den Task im ACS starten muss (refacs).	Ja
7.	Bindung an Ports	Notwendige Ports werden über die EXPOSE-Befehle im Dockerfile deklariert. Das Mapping dieser Ports wird in den Container-Einstellungen von ACS definiert.	Ja
8.	Nebenläufigkeit	Die Skalierung kann über ACS erfolgen. ACS ist bereits hoch skalierbar und bietet dem Verwalter des Containers die Konfiguration der Skalierung an. So können Minimum- und Maximum-Tasks sowie eine gewünschte Anzahl an Tasks definiert werden. Die Anzahl der laufenden Tasks bestimmt die Anzahl der laufenden Instanzen. Im verwendeten Nutzungsplan von ACS ist eine Instanz im Mittel über einen Monat verteilt kostenlos verfügbar.	Ja
9.	Einweggebrauch	Der Container kann schnell stoppt und gestartet werden. Durch Probleme mit der Rabbitmq-eigenen Datenbank besteht dann allerdings die Gefahr, im laufenden Betrieb hinzugefügte Nutzer neu hinzufügen zu müssen, da diese bei Rabbitmq auf den Namen des Hosts gespeichert werden und sich dieser beim Neustart des Containers ändert. Ein Workaround wurde entwickelt, dieser aktualiert aber lediglich das Init-Skript	Teilweise

4 Wetter-API(Datenquelle)

In diesem Kapitel wird auf die im Rahmen dieses Projektes Wetter-API vom Anbieter "OpenWeatherMap" eingegangen und anschließend eigens dafür implementierte Services zu dieser beschrieben. Anschließend werden die für den Services implementierten Klassen, Methoden und Tests beschrieben. Abschließend wird diese Komponente gegen die zu Beginn definierten Anforderungen validiert.

4.1 Die API

Der Anbieter "OpenWeatherMap" bietet kostenlos die Möglichkeit Wetterdaten via Http-Request abzufragen. Hierzu ist lediglich ein Nutzerzugang erforderlich, welchen sich jeder anlegen kann. Es können verschiedene Vorhersagen abgefragt werden, für diese Arbeit beschränkt es sich jedoch auf die tägliche Vorhersage und eine 5-tages Vorhersage. Solch ein angesprochener Http-Request (hier für eine 5tägige Vorhersage) setzt sich wie folgt zusammen,

```
http://api.openweathermap.org/data/2.5/forecast?zip=78467,de&
APPID=41c464d95d33fabc24d44a5086ea9848
```

Der Parameter "ZIP" wird zum setzten der Postleitzahl für die gewünschte Stadt genutzt, zu diesem muss noch das Länderkürzel hinzugefügt werden. Die "APPID" wird von "OpenweatherMap" für jeden Nutzeraccount spezifisch vergeben und dient als Authentifizierung. Der Parameter"forecast" dient zur Unterscheidung zwischen einer aktuellen Vorhersage und einer 5-tages Vorhersage, bei Erster würde der Request wie folgt aussehen,

```
http://api.openweathermap.org/data/2.5/weather?zip=78467,de&
APPID=41c464d95d33fabc24d44a5086ea9848
```

hier muss lediglich der Parameter "forecast" durch "weather" ersetzt werden.

"OpenWeatherMap" unterscheidet das Angebot zwischen kostenfrei und kostenpflichtig. Innerhalb der kostenpflichtigen Varianten gibt es Staffelungen, das gesamte Angebot wird aus Abb. 3 genauer ersichtlich.

Für die in dieser Arbeit beschriebenen Lösung wurde auf die kostenfreie Variante gesetzt. Daher musste bei der Implementierung auf einige Einschränkungen geachtet werden, zu diesen gehören, TODO erkläre die Einschränkungen

- "Calls per minute (no more than)",
- "Availibilty",
- "Weather API data update".

Da auf das kostenfreie Modell gebaut wird, muss vor allem auf die erstgenannte Einschränkung (Abb. 3) geachtet werden. Laut dieser sind lediglich 60 Requests mit den oben gezeigten URLs erlaubt. Im Rahmen der gesamten Anwendung wurde entscheiden, dass diese Applikation für alle Hauptstädte der 16 Bundesländer und Konstanz zur Verfügung stehen soll. Somit errechnet sich der Anteil an Requests pro Minute wie folgt,

$$RequestsPerMin. = (16 * 2) + (2 * 2) \tag{1}$$

$$RequestsPerMin. = 36 \tag{2}$$

somit kann ohne auf Probleme zu stoßen, ein Request-Intervall von 60 Sekunden, gewählt werden.

Im Bezug auf den "Response-Type" der API, kann entschieden werden, ob als "Response-Type" das Json- oder XML-Format gewählt werden soll. Aufgrund der guten Möglichkeiten von Java mit Json-Dokumente zu arbeiten, wurde das Json-Format, als "Response-Type", gewählt. Um bessere Vorstellungen von solch einem Response im Json-Format zu bekommen wird dieser untenstehend am Beispiel einer täglichen Vorhersage gezeigt.


```
{"coord":{"lon":9.16,"lat":47.67},"weather":[{"id":800,"main":"
    Clear","description":"clear sky","icon":"01d"}],"base":"
    stations","main":{"temp":297.6,"pressure":1021,"humidity
    ":41,"temp_min":296.15,"temp_max":298.15},"visibility
    ":10000,"wind":{"speed":5.7,"deg":30},"clouds":{"all":0},"dt
    ":1497797400,"sys":{"type":1,"id":4915,"message":0.0038,"
    country":"DE","sunrise":1497756293,"sunset":1497813872},"id
    ":0,"name":"Konstanz","cod":200}
```


Des weiteren wird am nächsten Beispiel der Response-Type für die 5-tägige Vorhersage aufgezeigt, jedoch wird hier aufgrund des hohen Umfangs nur ein kleiner Teil (für 3 Stunden) gezeigt. Im Normalfall besteht ein vollständiger Response aus im 3 stunden Takt folgenden Informationen für 5 Tage.


```
{"cod":"200","message":0.0035,"cnt":40,"list":[{"dt
    ":1497808800,"main":{"temp":295.56,"temp_min":294.226,"
    temp_max":295.56,"pressure":957.98,"sea_level":1034.02,"
    grnd_level":957.98,"humidity":53,"temp_kf":1.34},"weather
    ":[{"id":800,"main":"Clear","description":"clear sky","icon
    ":"01d"}],"clouds":{"all":0},"wind":{"speed":2.46,"deg
    ":51.5021},"sys":{"pod":"d"},"dt_txt":"2017-06-18 18:00:00"}
```

	Free	Startup	Developer	Professional	Enterprise
Price Price is fixed, no other hidden costs.	**Free**	**40 USD** / month	**180 USD** / month	**470 USD** / month	**2,000 USD** / month
Subscribe	Get API key and Start	Subscribe	Subscribe	Subscribe	Subscribe
Calls per minute (no more than)	60	600	3,000	30,000	200,000
Current weather API	✔	✔	✔	✔	✔
5 days/3 hour forecast API	✔	✔	✔	✔	✔
16 days/daily forecast API	-	-	✔	✔	✔
Weather maps API	✔	✔	✔	✔	✔
Bulk download	-	-	-	✔	✔
UV index (beta)	✔	✔	✔	✔	✔
Air pollution (beta)	✔	✔	✔	✔	✔
Weather alerts (beta)	✔	✔	✔	✔	✔
Service					
Availability	95.0%	95.0%	99.5%	99.5%	99.9%
SLA	-	-	-	-	✔
Weather API data update	< 2 hours	< 2 hours	< 1 hour	< 10 min	< 10 min
Weather maps data update	< 3 hours	< 3 hours	< 3 hours	< 3 hours	< 3 hours
API lifetime support	Current version	Current version	Current version	**All versions**	**All versions**
SSL	-	-	✔	✔	✔
License for maps, APIs, and other products	CC BY-SA 4.0	CC BY-SA 4.0	CC BY-SA 4.0	CC BY-SA 4.0	CC BY-SA 4.0 or custom
License for data and database	ODbL	ODbL	ODbL	ODbL	ODbL or custom
Tech support	Helpdesk	Helpdesk	Helpdesk	Direct	Direct 24x7

Abbildung 3: OpenWeatherMap Konditionen

Um mit diesen Json-Response-Types besser und freier arbeiten zu können wurde ein Service implementiert, welche die Daten vorher Filtert, somit werden nur wichtige Informationen genutzt. Die Implementierung dieses Services wird im folgenden Abschnitt beschrieben. Der Service greift die Grundlagen und Problemstellungen der "OpenWeatherMap"-API auf, welche in diesem Abschnitt beschrieben wurden, von den Request-Intervallen bis hin zu den zwei verschiedenen Arten von Vorhersagen.

4.2 Die Komponenten und Klassen

TODO: Komponenten Diagramm**

Im oben stehenden Diagramm werden die einzelnen Teilkomponenten der in diesem Kapitel beschrieben Komponente gezeigt.

Die zugrundeliegenden Klassen der einzelnen Komponenten werden im folgenden kurz beschrieben. Anschließend wird in den nächsten Abschnitten auf die wichtigsten Klassen und die darin enthaltenen Methoden noch genauer eingegangen.

MessagingController: Diese Klasse nimmt die Aufrufe der Wetterformulare der Startseite entgegen und leitet sie an den MessagingService weiter

UserController: Der User-Controller enthält die REST-Schnittstellen für den Login, die Registrierung neuer Nutzer und die Vergabe von Rechten.

User: Die Userklasse beinhaltet die Attribute zur Persistierung und Verwaltung der Nutzerdaten.

VHost: Die Klasse VHost enthält den Namen des virtuellen Hosts, den Namen des Nutzers dessen Rechte geändert werden sowie die Ausprägung der Rechte, unterteilt in Lese-, Schreib- und Konfigurationsrechte.

WeatherData: Die Klasse dient als Grundlage für die Datenstruktur der täglichen Daten des Json-Reponse, welche von der Klasse WeatherAPIService an die MOM gesendet werden.

WeatherDataWeekly: Die Klasse dient als Grundlage für die Datenstruktur der wöchentlichen Daten des Json-Reponse, welche von der Klasse WeatherAPIService an die MOM gesendet werden.

AuthenticationService: Diese Klasse dient dem Aufruf der notwendigen Methoden im UserRepository.

UserRepository: Über das UserRepository erfolgt der Zugriff auf die hinterlegte Datenbank.

WeatherApiService: Die Klasse vom Typ MessagingService regelt die HTTP-Requests an die "OpenWeatherAPI" und ließt die HTTP-Responses aus, diese Klasse wird im nächsten Abschnitt noch genauer eingeführt, da sie essentiell ist.

MomService: Über den MomService erfolgt der Zugriff auf die HTTP-Api von RabbitMQ.

SecurityService: Durch den SecurityService wird der Name des eingeloggten Nutzers ermittelt.

UserDetailsService: Der UserDetailService ist eine Klasse von Spring Security und lädt einen User und seine Rechte.

UserValidator: Mit Hilfe des UserValidators erfolgt die Validierung des Registrierungsformulars auf ungültige Passwörter und bereits vorhandene Nutzernamen.

VHostValidator: Durch den VHostValidator werden die Eingaben im Formular zur Vergabe von Rechten validiert.

WeatherFormValidator: Der WeatherFormValidator überprüft, ob alle notwendigen Eingaben zur Erstellung eigener Wetterdaten vorhanden sind.

WeatherApiApplication: Diese Klasse ist die Startklasse der Spring-Boot Applikation und lädt die properties-Dateien welche die Nachrichten der verschiedenen Validierungsklassen enthalten.

WebSecurityConfig: In dieser Klasse wird definiert, auf welche URLs dieser Anwendung ohne gültigen Login zugegriffen werden kann.

4.2.1 Wetter-API-Service

Für den Zugriff und die Abfrage der "OpenWeatherMap“ -API wurde ein Service mit der Klasse "WeatherDataService" implementiert, welcher die Abfrage regelt und ein Json mit der gewünschten Struktur zurückliefert. Dieses Json wird dann mit Hilfe von "Mqtt" an die MOM (vlg. Kap.3.1) gepublished.

Die gesamte Komponente verfügt über einen Web-Client. Dieser Web-Client stellt die Möglichkeit bereit Testdaten an die MOM zu senden oder aber die Abfrage der Wetterdaten an die "OpenWeatherMap"-API zu starten. Da für die gesamte Applikation die Livedaten von grundlegender Wichtigkeit sind, muss dieser Service zu jeder Zeit laufen. Es ist im Realbetrieb nicht vorgesehen, dass dieser gestoppt wird.

Nachfolgend werden die wichtigsten Methoden der Klasse "WeatherDataService" beschrieben.

"init()" füllt eine HashMap mit allen gewünschten Postleitzahlen auf, nach welchen die "OpenWeatherMap"-API abgefragt werden soll.

"public void publishLiveWeatherData()" ruft im Intervall von 60 Sekunden die Methoden "handlePLZtoday(plz, countryCode)“ und „handlePLZweekly(plz, countryCode)“ mit allen Postleitzahlen und Countrycodes auf, welche in der Init() -Methode eingelesen wurden.

"public void handlePLZtoday(String plz, String countryCode)" stellt einen HTTP-Request und fragt die tagesaktuellen Wetterdaten im Json-Format ab.

"public void dailyToWeatherData" liest den von der Methode "handlePLZtoday" gestellten HTTP-Request aus und zieht die gewünschten Daten anhand der Struktur der Klasse "WeatherData" aus dem Response-Json der "OpenWeatherMap"-API. Anschließend wird die gewünschte Json-Struktur zusammengebaut.

"public void handlePLZweekly(String plz, String countryCode)" stellt einen HTTP-Request und fragt die 5-tägigen Wetterdaten im Json-Format ab.

"public static ArrayList WeatherDataWeekly weeklyToWeatherDataWeekly (JsonElement root, liest den von der Methode "handlePLZweekly" gestellten HTTP-Request aus und zieht die gewünschten Daten anhand der Struktur der Klasse "WeatherDataWeekly" aus dem Response-Json der "OpenWeatherMap"-API. Anschließend wird die gewünschte Json-Struktur zusammengebaut.

"private void reconnectToMoM()" stellt sicher, dass bei einem Verbindungsverlust zur MOM wieder eine Verbindung hergestellt wird.

"public void publishFakeWeatherData(WeatherData weatherData)" sendet die Daten, via Web-Client eingepflegt wurden, an die MOM.

Im nachfolgenden Abschnitt wird die Implementierung zur Lösung des in Absatz 3.1 erläuterten Problems beschrieben. Dafür wurde ein Registrierungs- und Rechtevergabeprozess implementiert. Auf Grundlage der dabei erstellten Datensätze wird zum Abschluss eines Registrierungsprozesses oder der Vergabe von Rechten das init.sh Shellskript erstellt. Dieses kann dem Dockerimage hinzugefügt werden und legt beim Start eines Containers automatisch die Nutzer in Rabbitmq an. Damit der Zugriff auf die Registrierungs- und Rechteseite auf den Administrator beschränkt ist, wird dies in der Klasse WebSecurityConfig so definiert. In der selben Klasse werden auch die REST-Schnittstellen definiert die ohne Authentifizierung erreichbar sind. Ruft der Nutzer eine Schnittstelle auf, für die eine Authentifizierung oder eine andere Rolle notwendig ist, erscheint ein Login-Fenster. Die REST-Schnittstellen für den Login und die Rechtevergabe der virtuellen Hosts erfolgt über den UserController. Im UserController erfolgt die Persistierung und Validierung der Daten mit der in Kapitel 5 erläuterten Datenbank durch den Authentifizierungsservice. Gleichzeitig werden die Daten dieser Nutzer über die HTTP-API von Rabbitmq in selbigem gespeichert. Dies erfolgt durch den MomService.

4.2.2 Authentifizierungsservice

Der Authentifizierungsservice wird über den UserController aufgerufen. Der Service selbst wird für den Login-Prozess, die Registrierung neuer Nutzer und die Rechtevergabe für die virtuellen Hosts aufgerufen. Der Login-Prozess wird dabei vollständig von Spring Security übernommen. Wird ein neuer Nutzer registriert, erfolgt zuerst die Validierung

der eingebenen Daten durch den UserValidator. Dabei wird geprüft, ob der Nutzer bereits vorhanden ist und das Passwort mit dem Bestätigungspasswort übereinstimmt. Bei einer fehlerhaften Eingabe oder einem bereits vorhandenen Nutzer wird die entsprechende Ausgabe auf der Weboberfläche ausgegeben. War die Validierung erfolgreich, wird die Methode createUser(userForm) aufgerufen. Diese ruft die Methode insertSystemUser(Username, encodedPassword, Description) auf. Die Verschlüsselung des Passworts erfolgt über den BCryptPasswordEncoder von Spring Security. Im UserReposity werden die notwendigen SQL Statements aufgerufen, um den Nutzer in der Datenbank zu sichern. Die Vergabe von Rechten für virtuelle Hosts erfolgt über die Methode addPermission(vHostForm) des Authentifizierungsservices. Die vHostForm wird vorab durch den VHostValidator geprüft, ob der zugewiesene Nutzer bereits existiert. Anschließend erfolgt der Aufruf der addPermission-Methode und die Speicherung der Rechte des Nutzers für den virtuellen Host in der Datenbank über das UserRepository.

4.2.3 MOM-Service

Neben den Speicherung der Nutzer und Berechtigungen in der Datenbank müssen diese ebenfalls im Rabbitmq gespeichert werden. Dies erfolgt im Mom-Service. Der Service enthält dazu die Methoden addUser(loggedInUser, userToSave), setPermission(loggedInUser, userToSave, vHost) und createVHost(loggedInUser, vHost). Diese Methoden greifen über definierte Schnittstellen der HTTP-API auf Rabbitmq zu. Die Variable loggedInUser wird benötigt, um mit dessen Zugangsdaten über einen curl-Request die notwendigen Schnittstellen aufzurufen. Dies ist nur möglich, wenn der eingeloggte Nutzer Administrator-Rechte auf der Rabbitmq-Instanz hat. Zusätzlich enthält der Mom-Service die Methode writeScript(). Durch diese Methode werden die aktuellen Nutzer und ihre Rechte aus der Datenbank geladen und als rabbitmqctl-Aufruf im init-Skript für das Containerimage gespeichert. Das Skript kann in ein Repository für das Containerimage gespeichert werden. Beim der Ausführung des Dockerfiles wird das Skript als Startpunkt für den Container definiert. Wird der Container gestartet, erfolgt die Ausführung des Skripts und die Nutzer werden gemeinsam mit den aus der Datenbank ermittelten Berechtigungen beim Start des Containers angelegt. Auf das Deployment des Dockercontainers wird in Absatz **??** noch genauer eingegangen.

4.3 12 Faktor App

Validierung nach "12 Faktor APP"			
ID	**Anforderung**	**Validierungs Element**	**Erfüllt**
1.	Codebase	Andere Komponenten sind nur für das Testszenario da. Deployment verschiedener Versionen über Repo möglich	Ja
2.	Abhängigkeiten	Zugriff auf die Datenbank und die "OpenWeatherMap"-API sind in eigenen Klassen isoliert	Ja
3.	Konfiguration	Credentials werden über Umgebungsvariablen im Jenkings-Server verwaltet	Ja
4.	Unterstützende Dienste	Credentials der "OpenWeatherMap"-API werden über die Umgebungsvariablen im Jenkins-Server verwaltet	Ja
5.	Build, release, run	Wird über einen Jenkins-Server verwaltet	Ja
6.	Prozesse	Wird via Web-Gui gestartet	Ja
7.	Bindung an Ports	Die Ports werden von Pivotal verwaltet	Ja
8.	Nebenläufigkeit	Es könnte zu jeder Zeit eine neue Instanz auf Pivotal gestartet werden, jedoch bedarf es bei dieser Komponente keiner Skalierung	Ja
9.	Einweggebrauch	Pivotal generiert aus der hochgeladenen .jar oder .war einen Container, welcher jeder Zeit über die GUI in Pivotal gestoppt werden kann.	Ja
10.	Dev-Prod-Vergleichbarkeit	Containerisierung durch Pivotal	Ja
11.	Logs	Pivotal gibt die Logs standardmäßig über Stdout aus	Ja
12.	Admin-Prozesse	Nicht vorhanden	Nein

Tab. 3: Validierung nach "12 Faktor APP"

5 Datenverarbeitung

Eine gehostete Datenbank eröffnet unserem WetterAPI-System viele weitere Möglichkeiten. Neben System User Informationen der Messaging-Oriented Middleware RabbitMQ , können die empfangenen Wetterdaten kontinuierlich abgespeichert werden. Die hinterlegten Daten können nun für eine Reihe von Statistiken und Auswertungen genutzt werden und bieten außerdem den Vorteil den künftigen Ressourceneinsatz und die Skalierung effizienter zu gestalten. In diesem Kapitel wird der Aufbau einer passenden Datenbank beschrieben. Da das Testen eines skalierbaren Datenbankservices in der Regel sehr teuer ist, werden einige Funktionalitäten nur in der Theorie beschrieben. Dennoch lässt sich der Nutzen eines Datenbankkonzepts klar herausstellen. Zur Speicherung der Datensätze wird ein relationales Datenbankschema auf MYSQL Basis verwendet. Gehostet wird über den Amazon Web Service Dienst RDS. Für eine einfache Implementierung und Verwendung der Datenbank in den Systemmodulen CEP und MQTT (RabbitMQ) wird eine JDBC Schnittstelle zur Verfügung gestellt.

5.1 Datenbankschema

System User und VHost ER Modell Das Datenbankschema lässt sich in 2 verschiedene Bereiche unterteilen. Neben einer Speicherstruktur für die eingehenden Wetterdaten, lassen sich auch User Accountinformationen der RabbitMQ abspeichern. In Abbildung ... ist das Schema der Tabellenkonstellationen für die Speicherung der SystemUser zu sehen: Die Tabelle SystemUser beinhaltet die LogIn Credentials und die Tabelle VHost lässt die Sicherung der RabbitMQ Informationen zu. Die n:m Beziehung zwischen System Usern und VHost wird in der Tabelle Assigned abgebildet. Hier lassen sich auch die jeweiligen Berechtigungen read, write und configure des Users auf der RabbitMQ Instanz hinterlegen.

Wetter Daten ER Modell Damit die eingehenden Daten der WetterAPI korrekt abgelegt werden können, sind Stammdaten in der Datenbank gepflegt. Neben den zur Verfügung stehenden Städten in der Tabelle City, werden auch die Standardwetterdaten (Tabelle DeaultWeather) aus der verwendeten WetterAPI hinterlegt. Jedes eingehende Wetterdatenobjekt ist einer Stadt zugeordnet und referenziert ein DefaultWeather Eintrag. Die User unserer Systemlösung können ebenfalls in der Datenbank gespeichert werden. Die Tabelle Subscribe beinhaltet die n:m Referenzen von Usern, welche bestimmte Städte beziehungsweise deren Wetterdaten abonniert haben. Jeder Eintrag in der Tabelle Subscribe besitzt zudem das Datum der Aktivierung des Abonnements. Abbildung ... visualisiert das ER Modell der Wetterdatenspeicherung: Die Central Processing Unit, kurz CEP, errechnet anhand der eingehenden Wetterdaten verschiedene Benachrichtigungen (Alerts), welche über die MoM als Topic an die User verteilt werden. In der Datenbank können diese Alerts in Abhängigkeit der betroffenen Stadt gespeichert werden.

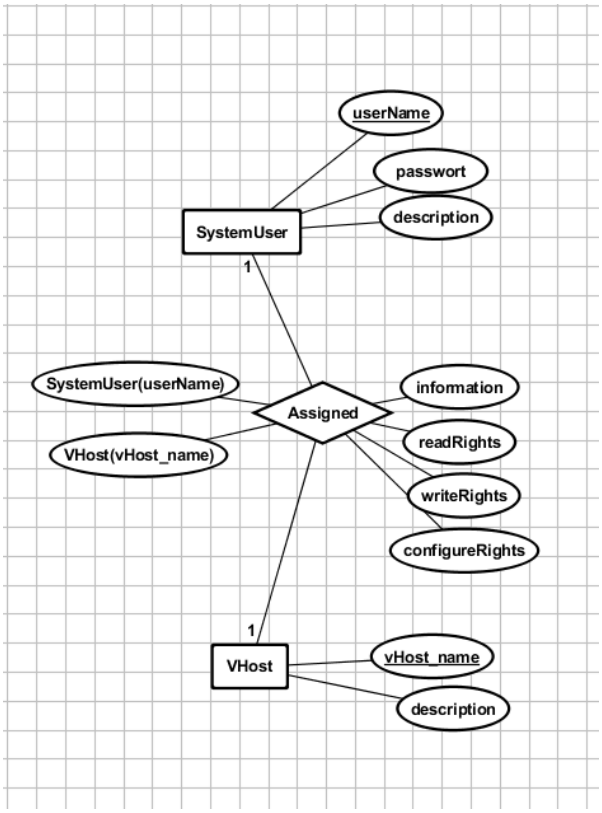

Abbildung 4: Datenbank ER Modellierung - Rabbit MQ System Data

Abbildung 5: Datenbank ER Modellierung - Wetter Daten Modellierung

5.2 Datenbank – JDBC Schnittstelle

Der Zugriff und die Verwaltung der Datensätze nach dem, im vorherigen Kapitel beschrieben, Datenschema, ist in der Java Klasse CadWeatherSystemDatabaseAPI realisiert. Diese Klasse lässt sich in alle Teilmodule unseres Gesamtsystems implementieren und bietet eine Schnittstellefunktionalität zur Verwendung der Datenbankinstanzen. In den Java Methoden, werden die MYSQL Kommunikationssatetments codiert als String über ein connection Objekt auf der Datenbank ausgeführt. Grundsätzlich stehen INSERT Operationen für alle Tabellen, sowie diverse SELECT Abfragemöglichkeiten zur Verfügung. Ziel dieser Struktur ist eine simple, vereinheitlichte Methodik zur Benutzung der Datenbankinstanzen. Der Konstruktor der Klasse gibt dem Entwickler die Möglichkeit ein MYSQL Datenbankobjekt in die Schnittstelle einzubinden.

In der nachfolgenden Tabelle sind alle derzeit programierten Methoden aufgelistet. Die INSERT Methoden liefern das Feedback der Datenbankoperation in einem String zurück. Die Rückgabe ResultSet der select-Methoden beinhaltet die gefunden Datentupel der Abfrage.

GENERAL DB

- checkDatabaseConnection()
- java.sql.Connection getConnection()
- setConnection(java.sql.Connection databaseConnection)

CEP DB - INSERT

- String insertDefaultWeather(int WeatherID, String main, String description, String $\text{icon}_r eferenz) String insertCity(int ZipCode, String cityName, Double logitude, Double latitude, i$
- String insertUser(String $\text{E}_M ail$, $String userName$, $String passwort$, $int mainLocation) String insertSub$
- String insertAlert(int ZipCode, Timestamp timestamp, String title, String code, String message)
- String insertWeather(Timestamp TimeStamp, int ZipCode, int WeatherID, double $\text{main}_t emp$, $double ma$

CEP DB - SELECT

- ResultSet selectUserByMail(String $\text{user}_{EM} ail) ResultSet selectCityByZipCode(int city_Z ipCode)$
- ResultSet selectDefaultWeatherByID(int $\text{defaultWeather}_I D) ResultSet selectAllWeatherByCityZipC$
- ResultSet selectWeatherByCityAndDay(int $\text{city}_Z ipCode, Timestamp weatherDay) ResultSet selectWe$
- ResultSet selectSubscribeByUser(String $\text{user}_{EM} ail) ResultSet selectSubscribeByCity(int ZipCode)$
- ResultSet selectUserPwByEMail(String $\text{E}_M ail$)

Rabbit MQ DB - Insert

- String insertSystemUser(String userName, String password, String additionalDescription)
- String insertVHost(String vHostName, String additionalDescription)

- String insertAssigned(String systemUser$_{u}serName, StringvHost_{n}ame, StringadditionalInformation

Rabbit MQ DB - SELECT

- ResultSet selectVHostAll()
- ResultSet selectSystemUserAll()
- ResultSet selectSystemUserByUserName(String userName)
- ResultSet ResultSet selectAssignedAll()

5.3 Auswertungen Nutzen der Databankspeicherung

Die Datensätze können vielseitig genutzt werden. Zum einen hält die CEP Komponente unseres Systems die empfangen Wetterdaten der MoM bzw. der WetterAPI nur temporär. Gleiches gilt für erzeugte Benachrichtigungen in Form von Alerts. Durch eine Sicherung in der Datenbank stehen die Wetterdaten unabhängig vom Status der CEP persistent zur Verfügung. Des Weiteren können verschiedene Auswertungen der Datensätze weitere Erkenntnisse bringen.

Wetter API User Durch die Protokollierung der Alerts lassen sich Statistiken über die Ereignisse und deren Verteilung aufstellen. Daraus könnten wir für die User beispielsweise Vorabwarnungen zukommen lassen. Außerdem könnte man den Usern auf lange Sicht gesehen Wettervergleiche bzw. Wetterentwicklungen bereitstellen.

Cloud System Analyse Mithilfe des gespeicherten Datums der Aktivierung von User Abonnements zu den jeweiligen Städten, lässt sich auch eine Prognose für unser System erstellen. Je nach Modell der Verteilung der gehosteten Komponenten unseres Systems können wir aus den Datensätzen abschätzen, wie sich die Last auf unserem System entwickelt. Diese Prognosen können für eine Kostenabschätzung des Cloudsytsem wertvoll sein, um Ressourcen effizient einzusetzen und damit verbundene Kostenpunkte besser vorherzusagen. Selbstverständlich ist der größte Vorteil einer Cloudlösung das dynamische Reagieren auf verschiedene Lastverhalten. Dennoch ist es im Rahmen von Businessmodellen notwendig zu wissen, welche Kosten an welcher Stelle entstehen beziehungsweise geplant werden können.

5.4 AWS RDS Datenbankverfügbarkeit

In Bezug auf die Performance spielt bei der Datenbank nur die Anzahl der CEP Calls eine Rolle. Dies führt je nach Szenario und Wetterveränderungen unterschiedlich viele

I/O Operationen aus. Dennoch würde unser WetterAPI System selbst bei extrem hohen Userzahlen keine Datenlast liefern, welche die Datenbank an ihre Grenze bringen könnte. Das Bottleneck unseres Systems sind die Instanzen der CEP. Im Rahmen unseres Projektes ist die Skalierung des Datenbank Services nicht eingerichtet, da zusätzliche oder skalierte Datenbankinstanzen mit hohen Kosten verbunden sind, da die benötigten Konfigurationen nicht im Umfang des AWS Trialkontingents enthalten sind. Ganz wichtig ist die Tatsache, dass das Starten einer weiteren Instanz das kostenlose Kontigent an Ressourcen aufhebt. Aus eigener Erfahrung wissen wir, wie sich ein solches Szenario zu einer gewissen Kostenfalle entwickeln kann. An dieser Stelle sollen dennoch die Möglichkeiten der Datenbankverfügbarkeit in Bezug auf den AWS RDS Dienst beschrieben werden.

Vertikale Skalierung Die Datenbank empfängt viele write-Operationen, weshalb eine vertikale Skalierung in unserem Fall der beste Ansatz wäre. Amazon bietet in seinem Service RDS (Relationale Datenbank Services) automatische Skalierungsoptionen an. So lassen sich verschiedene Monitoring Parameter (z.B. CPU Auslastung) einstellen, welche beim Erreichen eines Grenzwertes automatisch weitere Kapazitäten zur Verfügung stellt. Der Datenbankinstanz können je nach genutzer Datenbankengine bis zu 32 vCPU's mit 244GB RAM uns bis zu 64TB Datenspeicher zugeschaltet werden. Ein minimaler Nachteil ist eine kurze Downtime Phase in der die neuen Ressourcen eingebunden werden. Es gehen aber keine gespeicherten Daten verloren, sodass man in die Write Operationen der CEP im internen Speicher der CEP halten und bei erneuter Datenbankverfügbarkeit an den RDS Dienst übergeben könnte.

Horizontale Skalierung Eine horizontale Skalierung eignet sich vor allem bei Applikationen mit einem hohen Anteil an read-Operationen. Hierfür kann man beispielsweise den Amazon Container Service verwenden. Auch hier kann man Amazon konfigurieren, bei einem gewissen Lastverhalten, über einen Dockercontainer eine weitere Instanz aufzuziehen. Eine weitere Möglichkeit für eine horizontale Skalierung, sind Replica Objekte. Ein Replica Objekt bildet den Aufbau und den Inhalt einer Datenbankinstanz ab und steht für weitere Instanzdeployments zur Verfügung.

Ausfallsicherheit Für die Ausfallsicherheit bietet der AWS RDS Dienst die Option Multi-AvailabilityZone. Wenn die Option aktiviert wird, erstellt Amazon eine synchronisierte Datenbankinstanz in einer anderen Region bzw. in einem anderen Rechenzentrum. Beim Ausfall der aktiven Datenbankinstanz wird innerhalb einer Minute die synchrone Datenbankinstanz zugeschaltet und das DNS angepasst, damit sich aus Applikationssicht nichts verändert.

5.5 12 Faktor App

In diesem Absatz wird dargestellt, ob und wie die Anforderungen umgesetzt wurden.

Validierung nach "12 Faktor APP"			
ID	**Anforderung**	**Validierungs Element**	**Erfüllt**
1.	Codebase	Andere Komponenten sind nur für das Testszenario da. Deployment verschiederner Versionen über Repo möglich	Ja
2.	Abhängigkeiten	Keine Abhängigkeiten zu anderen Komponenten	Nein
3.	Konfiguration	Konfigurationsdatei wird beim Start des Containers aufgerufen und schränkt den Gast-Zugang auf localhost ein. Credentials werden über Umgebungsvariablen im Amazon Container Service (ACS) gepflegt.	Ja
4.	Unterstützende Dienste	Keine unterstützenden Dienste vorhanden	Nein
5.	Build, release, run	Könnte durch Jenkins verwaltet werden. Da die MOM keine regelmäßigen Update-Zyklen durchläuft, wurde darauf verzichtet.	Möglich, nicht aktiv
6.	Prozesse	Der Start des RabbitMQ wird durch das init-Skript im Dockerfile gesteuert. Dieses wird durch dem CMD-Befehl automatisch ausgerufen, so dass der Anwender nur den Task im ACS starten muss (refacs).	Ja
7.	Bindung an Ports	Notwendige Ports werden über die EXPOSE-Befehle im Dockerfile deklariert. Das Mapping dieser Ports wird in den Container-Einstellungen von ACS definiert.	Ja
8.	Nebenläufigkeit	Die Skalierung kann über ACS erfolgen. ACS ist bereits hoch skalierbar und bietet dem Verwalter des Containers die Konfiguration der Skalierung an. So können Minimum- und Maximum-Tasks sowie eine gewünschte Anzahl an Tasks definiert werden. Die Anzahl der laufenden Tasks bestimmt die Anzahl der laufenden Instanzen. Im verwendeten Nutzungsplan von ACS ist eine Instanz im Mittel über einen Monat verteilt kostenlos verfügbar.	Ja
9.	Einweggebrauch	Der Container kann schnell stoppt und gestartet werden. Durch Probleme mit der Rabbitmq-eigenen Datenbank besteht dann allerdings die Gefahr, im laufenden Betrieb hinzugefügte Nutzer neu hinzufügen zu müssen, da diese bei Rabbitmq auf den Namen des Hosts gespeichert werden und sich dieser beim Neustart des Containers ändert. Ein Workaround wurde entwickelt, dieser aktualiert aber lediglich das Init-Skript	Teilweise