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

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

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 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-

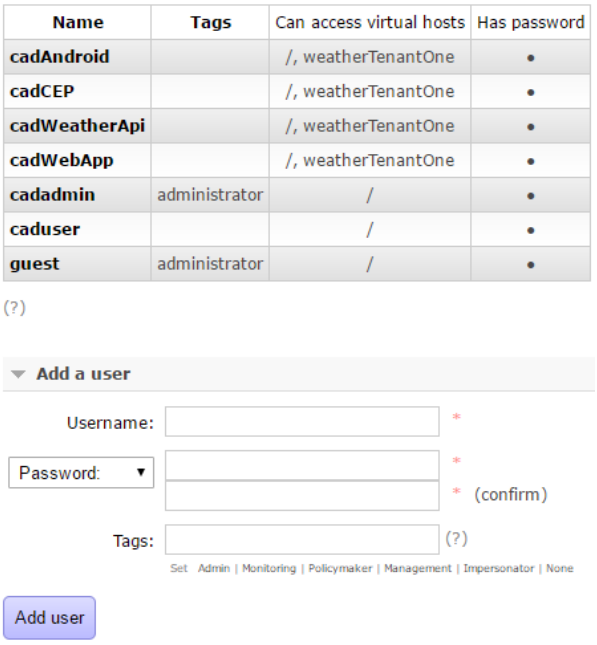

Abbildung 1: Administrationsmenü zur Benutzererstellung

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 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. Zusätzlich zum Management Plugin bietet RabbitMQ eine HTTP-Schnittstelle. Diese ermöglichen 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

Overview		Messages			Network		Message rates		+/-
Name	Users (?)	Ready	Unacked	Total	From client	To client	publish	deliver / get	
/	cadAndroid, cadCEP, cadWeatherApi, cadWebApp, cadadmin, caduser, guest	0	0	0	19kB/s	10B/s	3.6/s	0.00/s	
weatherTenantOne	cadAndroid, cadCEP, cadWeatherApi, cadWebApp	0	0	0	16B/s	23B/s	0.40/s	0.40/s	
weatherTenantTwo	No users	NaN	NaN	NaN					

Abbildung 2: Übersicht über die virtuellen Hosts

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.

4 Datenquelle (Wetter-API)

Validierung nach ”12 Faktor APP”			
ID	**Anforderung**	**Validierungs Element**	**Erfüllt**
1.	Codebase	....	Nein
2.	Abhängigkeiten	....	Nein
3.	Konfiguration	....	Nein
4.	Unterstützende Dienste	....	Nein
5.	Build, release, run	....	Nein
6.	Prozesse	....	Nein
7.	Bindung an Ports	....	Nein
8.	Nebenläufigkeit	....	Nein
9.	Einweggebrauch	....	Nein
10.	Dev-Prod-Vergleichbarkeit	....	Nein
11.	Logs	....	Nein
12.	Admin-Prozesse	....	Nein

Tab. 2: Validierung nach ”12 Faktor APP”

ToDo: Verweis auf 12 Faktor APP Standard !!!

5 Datenverarbeitung

5.1 CEP

5.2 Datenbank

6 Anwendersicht

6.1 Andriod-APP

6.2 Web-Client

7 Deployment

7.1 Allgemein

7.2 Cloudfoundry

7.3 Docker

Die in Absatz 3.1 erläuterte message-oriented Middleware RabbitMQ wird als Docker Container deployed. Durch die Verwendung von Docker Containern ist es möglich, lauffähige Software in isolierten Containern zu starten. Dies hat den Vorteil, dass die Software immer in identischen Umgebungen gestartet wird, unabhängig davon ob der Container lokal auf dem Entwicklungsrechner oder auf dem Produktivsystem läuft. Dies betrifft auch die Abhängigkeiten von notwendigen Installationen. So basiert RabbitMQ wie in Absatz 3.1 erwähnt auf der Sprache Erlang, weshalb diese auf jedem Entwicklungs- und Produktivsystem installiert werden müsste. Durch die Verwendung von Docker können notwendige Installationen bereits im Dockerfile definiert werden. So wird beispielsweise die Installation von Erlang in Abb. 3 veranschaulicht.

Dies ist ein Ausschnitt aus dem Dockerfile des offiziellen Rabbitmq-Baseimages, einer Abbildung des Containers. Der Abschnitt stellt die Installation von Erlang auf einem Linux-System unter Verwendung des Paketmanagers *Advanced Packaging Tool* (APT) dar.

```
# install Erlang
RUN apt-get update \
        && apt-get install -y --no-install-recommends \
                erlang-asn1 \
                erlang-base-hipe \
                erlang-crypto \
                erlang-eldap \
                erlang-inets \
                erlang-mnesia \
                erlang-nox \
                erlang-os-mon \
                erlang-public-key \
                erlang-ssl \
                erlang-xmerl \
        && rm -rf /var/lib/apt/lists/*
```

Abbildung 3: Installation von Erlang im Dockerfile Quelle: https://github.com/docker-library/rabbitmq/blob/a6cb36022a5c1a17df78cfafe45a73d941ad4eb8/3.6/-debian/Dockerfile

7.3.1 Dockerfile

Das Dockerfile für diesen Use-Case besteht neben dem erwähnten Baseimage aus Aktivierungen des in Absatz 3.1 vorgestellten Management Plugins, des MQTT Plugins sowie des MQTT-Websocket Plugins. Durch die Verwendung des Websocket Plugins ist die Kommunikation mit RabbitMQ auch über Webseiten möglich. Ein weiterer Bestandteil des Dockerfiles ist die rabbitmq.config. Diese ist notwendig um den Zugriff auf die MOM einzuschränken, da andernfalls eine Default-Konfiguration verwendet werden würde und durch diese der Nutzer *guest* vollen Zugriff hätte. Um dennoch Administrations-Zugriff zu erlangen, wird über das Shell-Skript *init.sh* ein Nutzer mit Administratorrechten angelegt. Die Zugangsdaten dieses Nutzers sind als Umgebungsvariablen hinterlegt. Die Ausführung des Skripts wird über den CMD-Befehl des Dockerfiles gesteuert. Zusätzlich ist es notwendig, Docker über den Befehl EXPOSE zu informieren, welche Ports der Container abhört. Im Anschluss an die Erstellung des Dockerfiles kann durch den Befehl *docker build* ein Image erzeugt werden. Dieses Image wird auf den Amazon Container Service deployed.

7.3.2 Amazon Container Service

Dabei handelt es sich um einen hoch skalierbaren Container Managementservice welcher eine unkomplizierte Handhabung von Docker Containern in Amazon EC2 Instanzen anbietet. Die Erstellung eines Clusters ist in wenigen Schritten möglich. Zunächst wird ein Repository zur Speicherung des erstellten Images angelegt. Im Anschluss daran kann nach der erforderlichen Installation des Amazon Web Service Command Line Interfaces (AWS CLI) der Zugriff auf das Repository erfolgen. Nachdem das Image gepusht wurde,

muss die Task Definition erstellt werden. Dabei handelt es sich um eine Anleitung für den Start des Containers. Ein Bestandteil der Task Definition ist das Port Mapping. Dabei wird dem Service mitgeteilt, wie die im Dockerfile deklarierten Ports ausserhalb des Containers erreichbar sein sollen. Zusätzlich werden in den erweiterten Optionen die Umgebungsvariablen und somit die Zugangsdaten für den in der *init.sh* erstellten Administator hinterlegt. Im nächsten Schritt erfolgt die Konfiguration des Services, in welcher ein Load Balancer erstellt und ausgewählt werden kann. Im abschließenden Schritt erfolgt die Konfiguration des eigentlichen Clusters. Hierbei erfolgt die Auswahl der Art und der Anzahl des gewünschten Instanz Typs. Durch diesen wird festgelegt, welche Resourcen für eine einzelne Instanz verfügbar sein werden. Darüber hinaus kann in diesem Abschnitt ein Schlüsselpaar für den SSH-Zugriff erstellt und ausgewählt werden. Erfolgt dies nicht, ist kein Zugriff über die EC2 Konsole auf den Container möglich.

7.4 REDIS

7.5 AWS

8 Kostenmodell