1 Deploy WordPress using AWS CloudFormation

1.1 Example Template

Für die Einarbeitung haben wir das Template „WordPress basic single instance“ verwendet. Das Template wurde direkt aus der Beispielvorlagensammlung per „Launch Stack“ deployed. Die Initialisierung erfolgt automatisch und nach einer Wartezeit ist die WordPress Instanz online.

1.2 ClouldFormation - WordPress on two Machines

Wir haben uns entschieden das im ersten Teil verwendete Template zu modifizieren. Das Template beinhaltet bereits die Installation von WordPress und MySQL, jedoch wird beides auf einer EC2-Instanz deployed. Für diese Aufgabenstellung müssen wir den MySQL Teil auf eine neue Maschine verschieben.

Die json Liste `Resources` muss zwei Sicherheitsgruppen (WebServer und MySQL), sowie zwei Maschienen vom Typ `AWS::EC2::Instance` für Web-Server und MySQL-Server enhalten. Der Web-Server Eintrag beschreibt alle notwendigen Befehle zur Installation und Initialisierung des Web-Servers, der MySQL Eintrag beschreibt analog alle notwendigen Befehle für den MySQL-Server.

Der Web-Server muss mit der internen IP-Adresse mit dem MySQL-Server verbunden werden. Dieser Schritt muss in der Konfiguration des Web-Server, genauer in der `wp-config.php` ausgeführt werden.

Dazu kann der CloudFormation-Befehl `"Fn::GetAtt": [ "MySQLServer", "PrivateIp"]` verwendet werden. Die Funktion gibt die PrivateIp der Resource „MySQLServer“ zurück. Mehr ist zur Verbindung nicht nötig.

1.3 Start

Für das Deployen auf CloudFormation verwenden wir das AWS-CLI. Das Skript `createEC2.sh` verwendet das Template `wordpress_mysql.json` und setzt alle notwendigen Properties. Die Installation benötigt ca. 5-10 Minuten.

2 Juju

2.1 Setup

Es wird folgend angenommen, dass juju mit einer validen Amazon Umgebung konfiguriert ist. Das bedeutet in erster Linie, dass Zugangsdaten entweder in der Konfigurationsdatei `/.juju/environments.yaml` angegeben, oder die Umgebugnsvariablen `AWS_ACCESS_KEY_ID` und `AWS_SECRET_ACCESS_KEY` in der genutzten Shell-Umgebung exportiert sind.

Zudem sollte ein `admin-secret` gesetzt sein, sofern die juju GUI genutzt werden soll.

2.2 Bootstrap

Zunächst muss eine Umgebung aufgesetzt werden. Dazu ist das Skript `bootstrap.sh` verfügbar, welches eine Amazon Umgebung startet, die für die weitere Ausführung vorausgesetzt wird.

2.3 Deploy WordPress

Das Skript `deploy-services.sh` setzt die Aufgabe 3.2 um. Zuerst werden Services für Wordpress und MySQL gestartet, eine Beziehung zwischen beiden hinzugefügt und anschließend Wordpress der Umgebung sichtbar gemacht (Port 80). Abschließend wird der MySQL Service durch das Hinzufügen einer Unit skaliert.

Hinweis: es kam bei der Ausführung der Skripte immer mal wieder vor, dass juju plötzlich keine Verbindung mehr zu den Amazon Servern aufbauen konnte (Fehlermeldung indiziert einen Verbindungsabbruch). Das ist kein persistentes Problem, ein erneutes Ausführen des Befehls wird erfolgreich abgeschlossen. Weshalb dies manchmal passiert, konnten wir nicht ermitteln. In aller Regel passierte es beim deployen mehrerer Services direkt hintereinander, jedoch kann auch ein `sleep` im Skript keine 100% Sicherheit schaffen.

Die Skripte sind dadurch mehr als eine Sammlung an Befehlen zu betrachten und nicht unbedingt zur unbeaufsichtigten Ausführung geeignet (da ggf. ein Abbruch stattfindet). Ein Ausführen aller Befehle im Skript manuell „per Hand“ im Terminal funktioniert immer, gänzlich frei von Verbindungsabbrüchen.

2.3.1 Genutzte Instanzen

Per default nutzt juju auf Amazon m1/m3 Instanzen.

Es ist möglich mittels z.B. `-constraints` (siehe `bootstrap.sh`) die Nutzung von t2 mikro Instanzen zu erzwingen, was in juju 1.23 jedoch aufgrund eines kleinen Fehlers nicht möglich ist.

Dieser Fehler wurde von unserer Gruppe behoben (https://github.com/juju/juju/commit/143e4fea), allerdings ist der nächste stable Release, der diesen Patch enthält (1.24), zum jetzigen Zeitpunkt noch nicht erschienen.

Selbst kompilieren ist zwar möglich, erfordert jedoch auch ein kompilieren der Tools (juju agent), welche juju auf die erstellten EC2 Instanzen läd. Die erstellten Binaries müssen beim Bootstrap entweder als custom `agent-stream` (juju Konfiguration), oder mittels `-upload-tools` angegeben werden.
`build-juju-agent.sh` baut den Agenten (auf einer EC2 Instanz), um die benötigten Binaries zu bekommen.

Es dürfte einfacher sein auf den 1.24 Release zu warten.

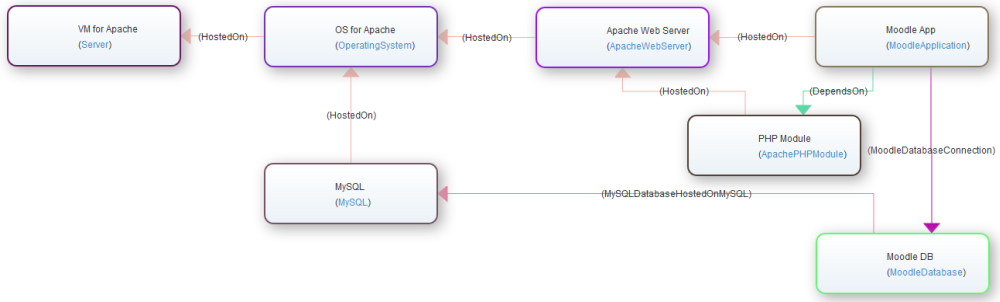

Abbildung 1: Moodle in Winery

3 Moodle mit OpenTOSCA

3.1 Winery mit Docker

Um Winery mit Docker zu deployen muss Docker zunächst installiert werden. Mittels Docker kann dann der Docker-Container von Winery deployed werden. Im Github Repository befindet sich ein Skript (`deploy_docker.sh`) dafür. Der Einfachheit beim Aufruf halber haben wir den Port `8080` des Containers an den Port `80` der Hostmaschine gebunden. Winery kann also unter `<HOST-IP>/winery)` aufgerufen werden.

3.2 Moodle in Winery

Abbildung 1 zeigt die Topologie vom Moodle-CSAR, dargestellt von Winery.

3.3 Moodle mittels OpenTOSCA

Wie bereits auf der Mailing-Liste dargestellt, ist es derzeit nicht möglich, Moodle mittels OpenTOSCA zu instanziieren.

3.3.1 Deployment von OpenTOSCA

OpenTOSCA wurde von uns auf zwei verschiedenen Wegen deployed, zum einen mittels Amazon CloudFormation und zum anderen durch das in der Anleitung angegebene Shell-Skript (sowohl stable als auch unstable). Bei allen Varianten steht OpenTOSCA nach kurzer Zeit unter `<SERVER-IP>:8080` zur Verfügung.

3.3.2 Moodle mit OpenTOSCA

Beim Deployment von Moodle treten laut Log (Error Logs im GitHub-Repository, `sh-deploy_2015-05_erro` (OpenTOSCA mittels Shell-Skript) und `cf-deploy_2015-05_error.log` (OpenTOSCA mittels CloudFormation)) Fehler auf, obwohl im Webinterface angezeigt wird, dass das

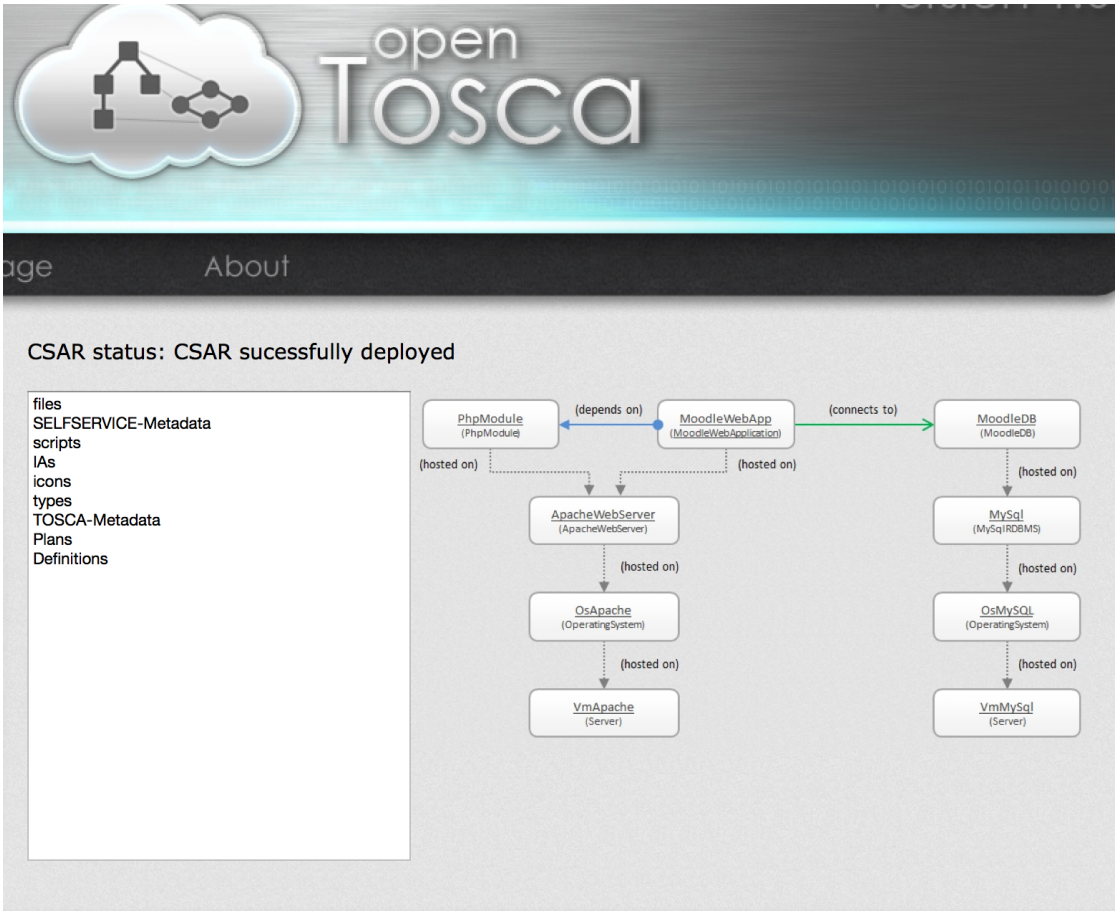

Abbildung 2: Moodle Deployment

Deployment fehlerfrei war (siehe Abbildung 2). Wenn Moodle jedoch im Folgenden mittels der Vinothek instantiiert werden soll, so schlägt dies fehl, das Webinterface bleibt bei Ïnstantiating Application... Please wait"hängen, im Log selbst tauchen keine Fehler oder weiterführende Informationen auf. Möglicherweise hängt dies nach der Bug-Beschreibung mit diesem Bug auf Github zusammen: https://github.com/CloudCycle2/YAML_Transformer/issues/143. Auch die vorgeschlagene Methode Moodle mittels SoapUI zu instantiieren schlägt fehl, da keine Verbindung aufgebaut werden kann.

Eine weitere Beochbachtung ist, dass der Export von CSAR-Archiven aus der Winery fehlschlägt, zumindest sind diese erheblich kleiner als das zuvor importierte CSAR und OpenTOSCA bezeichnet das Archiv als beschädigt.

4 WordPress mit OpenTOSCA

Es existieren drei mögliche Ansätze um die CSAR entsprechend zu modifizieren:

1. Winery: vorhandene Moodle CSAR modifizieren
2. Winery: neue CSAR erstellen
3. Moddle CSAR Inhalte manuell modifizieren

Im ersten Falle lädt man die Moodle CSAR in die grafische Oberfläche von Winery. Die Nodes die dabei geändert werden müssen sind „Moodle App“ und „Moodle DB“, respektive deren Typen „MoodleApplication“ und „MoodleDatabase“, sowie die zugehörige Verknüpfung „MoodleDatabaseConnection“. Zugehörig muss der Orchestrierungsplan abgeändert (Eclipse mit BPEL Plugin). Des weiteren müssen diverse Metadaten (Icon, Bilder, Beschreibung) der Instanzen abgeändert werden, welche der Benutzer später in der Vinothek sieht. Die im Hintergrund werkelnden Shell-Skripte lassen sich über die Artefakte unter „Other Elements“ definieren und ändern. Diese kümmern sich um die Installation von Apache, PHP, MySQL und Moodle (jetzt: Wordpress), sowie um die Konfiguration dieser Programme.

Der zweite Fall beginnt mit dem Erstellen eines neuen Service-Templates und erfordert ein selbstständiges Erstellen sämtlicher Nodes(Types), Relationship(Types), der Topologie, des Orchestrationplan und der ArtefaktTemplates. Natürlich müssen auch die entsprechenden Installations- und Konfigurationsskripte erstellt werden.

Der dritte Fall ist ein minimalinvasiver Ansatz: ändern des CSAR Archivs auf Dateiebene. Das ist möglich, da sich Wordpress und Moodle von der Topologie und den Voraussetzungen (MySQL, PHP, ...) gleichen und der Name einer Relation für deren Funktionalität keinen Unterschied macht. Bei diesem Ansatz ändern wir nur die tatsächliche Funktionalität in Form der Skripte ab, die sich in der CSAR Datei finden.

Die anzupassenden Skripte sind:

- `scripts/MoodleApplication/install.sh`
- `scripts/MoodleApplication/configure.sh`
- `scripts/MoodleDatabase/configure.sh`
- `scripts/MoodleDatabaseConnection/configureDatabaseEndpoint.sh`

Änderungen an den Konfigurationsskripten die Applikation und Datenbank sind analog zu den eingebetteten Skripten im CloudFormation Template `wordpress_mysql.json` aus Task 1. Um das Installationsskript anzupassen, gibt es drei Wege. Entweder man packt das Installationsarchiv mit in das CSAR-Archiv und entpackt es im Installationsskript in das WebRoot-Verzeichnis, oder man lädt es im Skript herunter und entpackt es oder man installiert WordPress in diesem Skript mittels der Paketverwaltung.

Zusätzlich sollten die Meta-Daten angepasst werden, damit das CSAR sich korrekterweise als WordPress-CSAR zu erkennen gibt. Die anzupassenden Metadaten sind:

- `SELFSERVICE-Metadata/data.xml`
- `SELFSERVICE-Metadata/icon.jpg`
- `SELFSERVICE-Metadata/image.jpg`

Hinweis: Die Neuerstellung des Orchestrierungsplans ist immer bei der Abänderung der Skripte und der Namen der Node/Replationship(Types) nötig, da sich diese in diesem Plan wiederfinden.