Red Hat OpenShift

Sebastian Krieger

Hochschule Mannheim
Fakultät für Informatik
Paul-Wittsack-Straße 10
68163 Mannheim
926182@stud.hs-mannheim.de

Zusammenfassung Red Hat steht mit seiner Plattform as a Servive (PaaS) OpenShift in den Startlöchern. Es steht somit mit bereits etablierten Diensten, wie AppScale(Google), Heroku(Salesforce.com) und Windows Azure(Microsoft), im direkten Konkurenzkampf um den noch stark umkämpften Markt. Momentan befindet sich OpenShift noch in der Entwicklung und steht in dieser Version kostenlos zur Verfügung. Diese Arbeit hat sich zum Ziel gesetzt Red Hat's neue PaaS genauer zu begutachten und die Vor- und Nachteile zu erläutern.

In den nächsten Kapiteln bekommen Sie einen Überblick über Red Hat's neue PaaS OpenShift. Beginnend von der eigentlichen Plattform und ihren Resourcen bishin zu einer fertigen Applikation.

1 Grundlagen

Ziel einer PaaS ist es dem Benutzer so wenig Arbeit wie möglich aufzuzwängen. Deswegen besteht OpenShift aus Sicht des Entwicklers(High-Level) aus zwei wichtigen Komponenten, dem **Broker** und den **Cartridges**.

Abbildung 1 veranschaulicht das Zusammenspiel zwischen Benutzer, Broker und Cartridges.

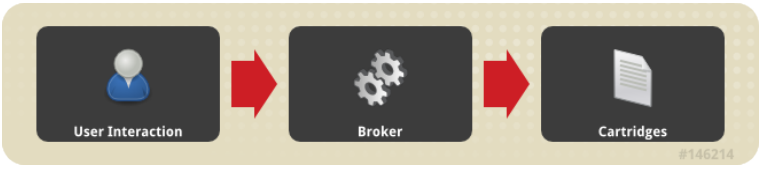

Abbildung 1. OpenShift High-level Übersicht *(Quelle: Openshift Manual)*

Zusätzlich gibt es noch einen kleinen Einblick in Gears, Nodes und Districts.

1.1 Broker

Der Broker bildet die Schnittstelle für den Benutzer um mit Openshift zu kommunizieren. Der Broker wird über eine REST-API angesprochen. Wie dies im Detail geschieht sehen wir später im Kapitel 4.

1.2 Cartridges

Cartridges sind Module, die eine spezielle Funktionalität anbieten. Eine Cartridge kann auf einem oder mehreren Gears laufen. Mit diesem Prinzip kann je nach Bedarf eine Appliktation um die nötige Funktionalität erweitert werden. Openshift gliedert die Cartridges in 5 Gruppen: (Stand: 22.05.2012)

Applikationstypen: Openshift unterstützt viele aktuelle Programmiersprachen. (JBoss/JavaEE 6, Ruby 1.8.7, Node.js 0.6, Python 2.6, PHP 5.3 oder Perl 5.10)
Openshift hält für Applikationen die nicht durch diese Programmiersprachen ausgeführt werden können, die sogenannte „Do It Yourself"-Cartridge(DIY) bereit. Damit können Benutzer versuchen nicht-unterstütze Anwendungen auf Openshift zum Laufen zu bekommen. Wie man eine solche DIY-Applikation erstellt werden wir am Anwendungsbeispiel in Kapitel 7 sehen.

Administration: Zur Verwaltung stehen Cartridges, wie PhpMyAdmin und RockMongo bereit.

Datenbanken: Mit MongoDB unterstützt Openshift sowohl NoSQL-Datenbanken, wie auch altbewährte Datenbanksysteme wie MySQL und PostgreSQL.

Entwicklungstools: Mit den Cartridges „Jenkins Server" und „Jenkins Client" können Applikationen automatisch übersetzt bzw. getestet werden.

„Andere": Zusätzlich bietet Openshift noch Cartridges für „background jobs" via Cron.

1.3 Applikationen und Sicherheitsmaßnahmen

Eines der wichtigsten Themen für einen Cloud-Anbieter ist es dem Kunden ein gewisses Maß an Sicherheit zu bieten. In OpenShift läuft jede Applikation bzw deren Gears in einem „Application Container". Abbildung 2 zeigt einen einfachen Blick hinter die Kulissen. Somit kann gewähleistet werden, dass sich Applikationen nicht in die Quere kommen.

Gears Gears bilden die Recheneinheiten der Applikation. In einem Gear läuft eine oder mehrere Cartridges. Sie bestimmen die Größe an RAM, SWAP und Festplattenspeicher. Openshift bietet 2 Arten:

Small: Mit 512MB RAM, 100MB Swap und 1GB Festplattenspeicher.

Medium: Mit 1GB RAM, 100MB Swap und 1GB Festplattenspeicher.

Jedem Benutzer stehen drei kostenlose „small“ Gears zur Verfügung. Diese können zum Beispiel auf drei einzelne kleine oder auf eine etwas mächtigere Applikation aufgeteilt werden.

Nodes Mehrere Gears laufen auf physikalischen bzw virtuellen Maschinen. Diese Maschinen bezeichnet Openshift als Nodes. Gears werden in der Regel auf mehreren Nodes angelegt und je nach Bedarf zugeschalten.

Districts Der District sorgt dafür das kein Node durch stark genutzte Gears überladen wird. Er definiert dafür ein Set aus Nodes, in denen die Gears einfach und schnell ausbalanciert werden können.

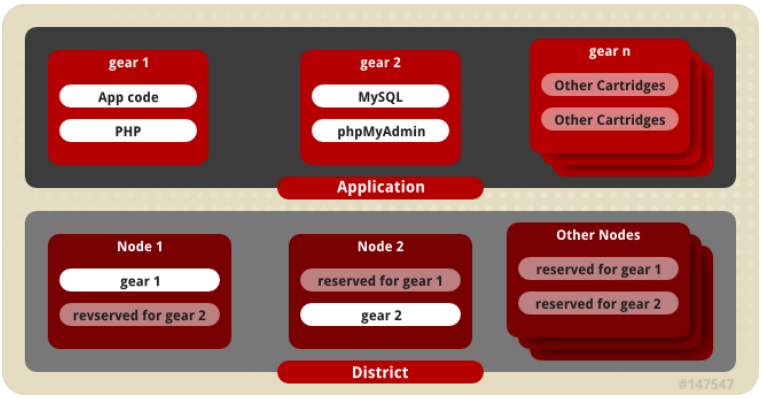

Abbildung 2. „Application Container“ *(Quelle: Openshift Manual)*

1.4 Namespace

Jeder Benutzer der sich bei Openshift registriert, muss bei der Registrierung einen eindeutigen Namespace angeben. Jeder Benutzer besitzt nur einen Namespace unter dem die Applikationen laufen. Die Kombination aus Applikationsnamen und Namespace ergeben dann eine eindeutige Url der Applikation.

```
Applikationsname: Beispiel
Namespace:        Seminar
Domainname:       seminar.rhcloud.com
Url:              http://beispiel-seminar.rhcloud.com
```

2 Die Openshift-Console

Openshift bietet zur Verwaltung der Applikationen ein Webinterface an.[1] Damit können von überall(Voraussetzung ist ein internetfähiger Browser) Applikationen erstellt bzw. verwaltet werden. Der Umfang des Webinterfaces lässt noch etwas zu wünschen übrig. (Stand 22.05.2012) Wer kein Interesse an tiefgehendere Informationen über die Applikation hat, kann das Webinterface getrost benutzen. Wer aber zum Beispiel den Status der Applikationen abfragen möchte oder einstellen möchte wieviele Gears der Applikation zu gewiesen werden sollen, der kommt momentan nicht ohne die „Openshift Client Tools“ aus.

3 Openshift Client Tools

Die Openshift Client Tools(OCT) bieten mit Abstand die meisten Möglichkeiten zur Verwaltung von Applikationen in Openshift. Sie bedienen sich der REST-API von Openshift und stehen auf Github zur freien Verfügung.[2] Die Anwendung dieser Tools sehen wir später im Anwendungsbeispiel in Kapitel 7.

4 Openshifts REST-API

Die komplette Kommunikation von Openshift und Benutzer läuft über den Broker und der REST-API. Dadurch lässt sich mit den Operationen GET, POST, PUT und DELETE Openshift komplett verwalten. Dies kann theoretisch auf verschiedene Arten geschehen. An dieser Stelle nur ein kurzes Beispiel wie der Zugriff mit dem Programm „curl“ aussehen würde.[3]

```
curl https://openshift.redhat.com/broker/userinfo \
-d "password=XXXXXXX" \
-d 'json_data={ "rhlogin":"sebastian-krieger@online.de"}'
```

Eine ausführliche Beschreibung der API ist im offiziellen API-Guide von Openshift zu finden.

5 Openshift Origin

Red Hat typisch wurde vor kurzem auch Openshift unter dem Projekt „Origin“ veröffentlicht. Das Origin Projekt beinhaltet alle opensource Komponenten die von Openshifts PaaS benutzt werden. Der Quellcode der einzelnen Komponenten liegt auf Github zum ausführlichen Studium bereit. Wer Openshift für zuhause haben möchte kann sich die Live-CD herunterladen und mittels VMware und Virtualbox seine eigene PaaS erstellen.

6 Red Hats Zukunftspläne für Openshift

Nachdem Red Hat bereits die freie gehostete Variante für Entickler auf den Weg gebracht hat, soll im Laufe dieses Jahres eine erste Preisliste für die kostenpflichtige Variante kommen. Wie diese Variante im wesentlichen Aussehen wird ist noch nicht sicher. Sicher ist aber, dass die freie Variante mit drei freien Gears bestehen bleibt.

7 Anwendungsbeispiele

7.1 Installation der Client Tools

Um die Client Tools installieren zu können müssen folgende Voraussetzungen erfüllt sein:

- Git
- Ruby (ab 1.8.7; Empfohlen: 1.9.2)
- Rubygems (ab Ruby 1.9.2 automatisch dabei)

Sind diese Voraussetzungen gegeben, können wir mit folgendem Befehl die Client Tools installieren.

```
> gem install rhc
```

7.2 Erstellen einer neuen Domain

Mit folgendem Aufruf wird eine neue Domain und im Home-Verzeichnis eine Konfigurationsdatei ($HOME/.openshift/express.conf) mit den Logindaten erstellt.

```
> rhc domain create -n example -l user@example.com
```

7.3 Erstellen einer neuen Anwendung

Nachdem die Domain erstellt wurde, können wir unsere erste Anwendung erstellen. In diesem Beispiel erstellen wir eine neue Anwendung mit Ruby 1.8.7 Unterstützung.

```
> rhc app create -a rubyapp -t ruby-1.8
```

Dieser Aufruf erstellt im aktuellen Verzeichnis unter ./rubyapp ein neues Git Repository mit der Verzeichnisstruktur von Abbildung 3.

Die Anwendung ist nun unter der `http://rubyapp-example.rhcloud.com` aufrufbar.

```
.
|-- config.ru
|-- .git
|-- .openshift
|   |-- action_hooks
|   |   |-- build
|   |   |-- deploy
|   |   |-- post_deploy
|   |   `-- pre_build
|   |-- cron
|   `-- markers
|-- public
|-- README
`-- tmp
```

Abbildung 3. Verzweichnisstruktur einer neuen Ruby-Anwendung in Openshift

7.4 Erstellen einer nicht unterstützten Anwendung mit dem Do-It-Yourself Applikationstyp

Wie wir gesehen haben ist es ein Kinderspiel eine unterstützte Anwendnung zu erstellen. Openshift bietet aber auch die Möglichkeit eine nicht offiziell unterstützte Anwendung in Openshift laufen zu lassen. Hierfür gibt es den Do-It-Yourself Applikationstyp(DIY). In diesem Beispiel werden wir eine Rails-Applikation in der Version 3.2 erstellen. Rails 3.2 setzt aber Ruby 1.9.3 voraus und das wird nicht offiziell unterstützt. Der erste Schritt ist zunächst eine neue Applikation zu erstellen.

```
> rhc app create -a diy -t diy-0.1
```

Wie im vorhergehenden Beispiel (Abbildung 3) wird der Ordner `./.openshift/action_hooks` erstellt. Die darin enthaltenen Skripte spielen in diesem Beispiel eine Hauptrolle.

- deploy
- post_deploy
- pre_build
- build
- start
- stop

Diese werden in der obigen Reihenfolge bei jedem Deploy (`git push`) ausgeführt. Nun brauchen wir erstmal die richtige Version für Ruby (1.9.3). Dafür installieren wir Rbenv auf dem entsprechenden Gear.

Doch wie können wir auf unser Gear zugreifen? Openshift bietet die Möglichkeit sich per ssh auf dem Gear anzumelden. Jede Applikation besitzt eine UUID. Diese kann man über den Befehl

```
> rhc app show -a diy
```

abrufen. Mit dieser UUID können wir uns mit unserem Gear verbinden.

```
> ssh <UUID>@diy-example.rhcloud.com
```

Sobald wir mit dem Gear verbunden sind, installieren wir zunächst Rbenv[5]:

```
> curl -L https://raw.github.com/Seppone/\
openshift-rbenv-installer/master/bin/rbenv-installer | bash
```

Jetzt kann Rbenv mit folgendem Befehl geladen werden:

```
>. $OPENSHIFT_DATA_DIR/.rbenv/plugins/\
openshift-rbenv-installer/bin/rbenv-bootstrap-openshift
```

Nachdem Rbenv geladen wurde, kann auch Ruby durch den Befehl

```
> rbenv install 1.9.3-p194
```

installiert werden. Dieser Vorgang kann mehrere Minuten dauern.

`TIPP:` Sollte die Verbindung bei der Installation von Ruby via Rbenv wegen einem Timeout abbrechen, dann sollte man dem `ssh`-Befehl den Parameter `-oServerAliveInterval=10` hinzufügen.

Als letzten Schritt auf dem Gear sollte noch Bundler[6] installiert werden. Dies geschieht wie üblich durch:

```
gem install bundler
```

Bundler dient zum Auflösen und Installieren der Abhängigkeiten der Applikation. Mit der entsprechenden Ruby-Version und Bundler ist der Gear bereit um eine Rails 3.2 Applikation auszuführen.

Nun müssen nur noch die „Action Hooks“ der Applikation (siehe 7.4) angepasst werden, damit Openshift weiß was zu tun ist.

Wichtig sind in diesem Fall die Skripte „start“, „stop“ und „deploy“:

Listing 1.1. start

```
#!/bin/bash
# The logic to start up your application should be put in this
# script. The application will work only if it binds to
# $OPENSHIFT_INTERNAL_IP:8080


. $OPENSHIFT_DATA_DIR/.rbenv/plugins/openshift-rbenv-installer/bin/rbenv-bootstrap-openshift

cd $OPENSHIFT_REPO_DIR

echo "Precompiling assets..."
rake assets:precompile

echo "Starting unicorn..."
unicorn_rails -D -o $OPENHIFT_INTERNAL_IP -p $OPENSHIFT_INTERNAL_PORT -E production
```

Listing 1.2. stop

```
#!/bin/bash
# The logic to stop your application should be put in this script.

. $OPENSHIFT_DATA_DIR/.rbenv/plugins/openshift-rbenv-installer/bin/rbenv-bootstrap-openshift

kill `cat $OPENSHIFT_REPO_DIR/tmp/pids/unicorn.pid`
```

Listing 1.3. deploy

```
#!/bin/bash
# This deploy hook gets executed after dependencies are resolved and the
# build hook has been run but before the application has been started back
# up again.  This script gets executed directly, so it could be python, php,
# ruby, etc.

. $OPENSHIFT_DATA_DIR/.rbenv/plugins/openshift-rbenv-installer/bin/rbenv-bootstrap-openshift

cd $OPENSHIFT_REPO_DIR

echo "Running Bundler..."
bundle >/dev/null

echo "Migrate Database..."
rake db:migrate RAILS_ENV=production >/dev/null
```

Um eine Verbindung zur Datenbank herzustellen stehen im Gear folgende Umgebungsvariablen zur Verfügung.

- OPENSHIFT_DB_HOST
- OPENSHIFT_DB_PORT
- OPENSHIFT_DB_USERNAME
- OPENSHIFT_DB_PASSWORD
- OPENSHIFT_DB_SOCKET

Sind diese Schritte vollbracht, kann man mit

```
> git add .
> git commit -m 'rails 3.2 on openshift'
> git push
```

die Applikation Openshift übergeben werden.

8 Fazit

Openshift macht in der aktuellen *Developer Preview* einen leicht unreifen Eindruck. Doch wenn man die rapide Entwicklung und die lebendige Community beachtet, dann könnte Openshift in den nächsten Monaten zu einem ernsten Mitstreiter auf dem PaaS-Markt werden. Gerade der „Do-It-Yourself“ Applikationstyp bietet unzählige Möglichkeiten. Besonders gespannt kann man auf die Möglichkeit sein, einen eigenen PaaS mittels Openshift Origin zu erstellen. Doch dieses Projekt befindet sich noch in Arbeit.[7]

Wer einen genauren Blick in die Plattform werfen möchte, der kann sich direkt den Code unter Github anschauen. Oder lädt sich die Live-CD[8] herunter und probiert in einer virtuellen Umgebung Openshift selbst aus.

Literatur

1. Openshift Console
   `https://openshift.redhat.com/app/console/applications`

2. Das Github-Repository der OpenShift Client Tools
   `https://github.com/openshift/os-client-tools`
3. curl and libcurl
   `http://curl.haxx.se/`
4. Offizieller API-Guide von Openshift
   `http://docs.redhat.com/docs/en-US/OpenShift/2.0/html/API_Guide/index.html`
5. Rbenv auf Github.com
   `https://github.com/sstephenson/rbenv`
6. Hompage von Bundler
   `http://gembundler.com/`
7. Build your own PaaS
   `https://openshift.redhat.com/community/wiki/build-your-own`
8. Openshift Origin Live-CD
   `https://openshift.redhat.com/app/opensource/download`
9. Heroku
   `http://www.heroku.com/`