Gruppenarbeit 1 - Cloud Fundamentals beim Provider

Alexander van Schie & Oli Dias

March 18, 2019

Contents

1 Hands-On: Hello (Cloud) World

Dieses Kapitel befasst sich mit der ersten Erfahrung mit dem gewählten Cloud Provider: Openshift online.

Die Wahl ist nach einer kurzen Recherche auf Openshift gefallen, weil die ausgedachten Kriterien (Verknüpfung mit GitHub Repository, Push auf Repo führt zu Redeploy und .NET Core und Java Spring Kompatibiltät) mit dem ersten Eindruck effizient vollbracht werden können.

1.1 Installationsanleitung

Das Ziel ist es, eine ASP.NET-Core Hello-World Applikation mittels Openshift Online zu builden und deployen. Am Schluss dieser Installationsanleitung sollte dies möglich sein.

Für das Deplyoment dieser Applikation auf der Openshift Online Plattform müssen grob folgende Schritte durchgeführt werden:

- Account und Projekt auf der Plattform erstellen
- GitHub Repository der ASP.NET-Core Applikation mit Openshift Projekt verbinden
- Applikation auf Openshift builden
- Applikation auf Openshift deployen

1.1.1 Openshift Account erstellen

Zuerst muss ein Account auf `https://manage.openshift.com/signin` erstellt werden. Danach kann zwischen den in Abbildung 1 vorgeschlagenen Plänen ausgewählt werden. Wir benutzen den Openshift Online Pro Plan (30-day trial).

Dass der Account verifiziert werden kann, muss eine Telefonnummer hinterlegt werden, auf welche darauffolgend einen Bestätigungscode zugeschickt wird.

Wurde diese eingegeben und verifiziert, erscheint eine Übersicht über das bestellte Produkt wie in Abbildung 2 angezeigt. Daraufhin kann die Subscription bestätigt werden.

Kurz nach dem Bestätigen sollte ein Bestätigungsmail eintreffen. Darauffolgend kann bereits die Web Console geöffnet werden. Es wird ein Katalog mit allen Produkten von Openshift Online dargestellt (Abbildung 3).

Das Erstellen des Openshift Accounts ist somit abgeschlossen und die Plattform ist für das Builden und Deployen von Applikationen bereit.

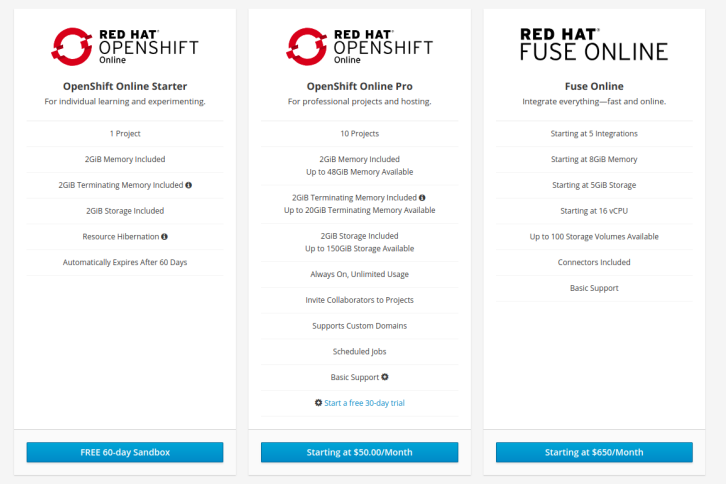


Figure 1: Gewählter Plan Openshift

1.1.2 Build und Deployment der Applikation mit dem Web-UI

In der Web-Console können wir nun auf .NET Core Projekt klicken. Daraufhin erscheint ein Wizard, dem wir Schritt für Schritt folgen können. Falls das GitHub-Repository schon während dem Wizard hinzugefügt werden soll, muss es bereits existieren und sichtbar sein.

Entsprechende Felder müssen gemäss Abbildung 4 ausgefüllt sein. Vorsicht ist bei der .NET Version geboten; wir verwenden die Version 2.2 von .NET Core.

Sobald das Projekt in Openshift erstellt wurde, startet der Build. Womöglich schlägt der Build aufgrund fehlender `.s2i`-Konfiguration (Source-2-Image) fehl. Um diesen Fehler zu beheben, muss Openshift gesagt werden, wo das Startup-Projekt liegt. Dazu muss ein Ordner und File mit dem Namen `.s2i/environment` erstellt werden. Dieses beinhaltet folgendes:

```
1 DOTNET_STARTUP_PROJECT=HelloWorld-netcore/HelloWorld
    -netcore.csproj
```

Wichtig ist weiter zu beachten, dass die .NET Versionen (.NET sowie NuGet-Pakete) mit denjenigen von Openshift kompatibel sind.

War der Build erfolgreich, muss noch das Deployment konfiguriert werden. Dazu kommt ein weiteres File mit dem Namen `run` ins `.s2i` Verzeichnis. Darin muss die Applikation noch gestartet werden. Dies funktioniert so:

```
1 exec dotnet run
```

Ist auch dieser Schritt vollbracht und der Build erfolgreich, kann im Control Panel des Projektes zu Deployments → Routes navigiert werden. Dort erscheint eine

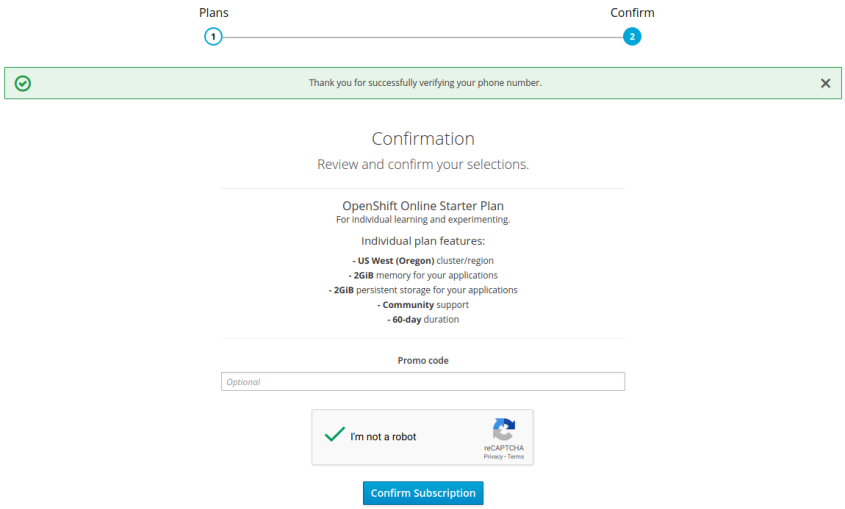


Figure 2: Übersicht des abgeschlossenen Plans

Tabelle, wo der Hostname bereits angegeben ist und womit nun die ASP.NET-Core Applikation vom Internet her erreichbar ist.

1.1.3 Installation der Openshift CLI

Neben dem Web-UI besteht ähnlich wie mit GAE die Möglichkeit, ein Konsolentool für das Konfigurieren der Cloud Projekte zu verwenden. Die Installation des Tools stellte auf Ubuntu 18.04 LTS ein Problem dar, wie im Folgenden beschrieben wird.

Beim Ausführen der Konsolenapplikation `oc`, auf die in der Doku [1] verwiesen wird, kam der Fehler in Abbildung 6 auf.

Gemäss einem Issue auf GitHub[2] handelt es sich um einen fehlerhaften Symlink, der in dieser spezifischen Version enthalten war. Es wird des Weiteren auf eine aktueller Version verwiesen, mit welcher das Problem nicht mehr existiert. Openshift garantiert nur für RHEL[3] Support, weshalb auf anderen Linux Distributionen oder Betriebssystemen mit solchen Fehlern gerechnet werden muss.

[1] `https://docs.openshift.com/online/getting_started/basic_walkthrough.html`

[2] `https://github.com/openshift/origin/issues/21061`

[3] Red Hat Enterprise Linux

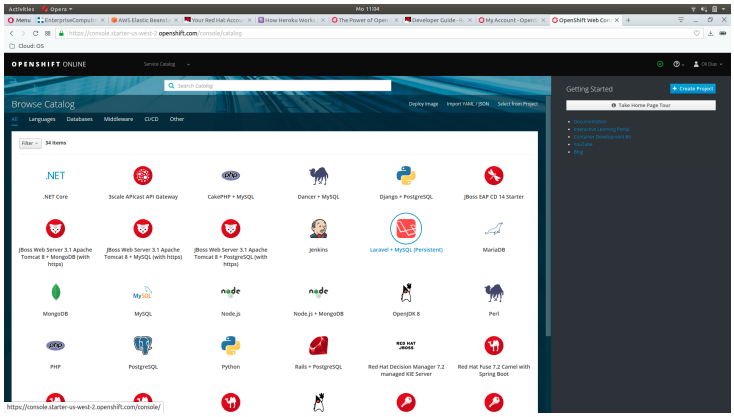


Figure 3: Katalog von Openshift

1.1.4 Build und Deployment der Applikation mit dem CLI

Nachdem die Installation des CLI vollbracht ist, können die gleichen Befehle des Web-UIs in der Konsole ausgeführt werden.

Erstellen eines Projektes

```
oc new-project helloWorld --display-name='ASP.NET Core Hello World'
```

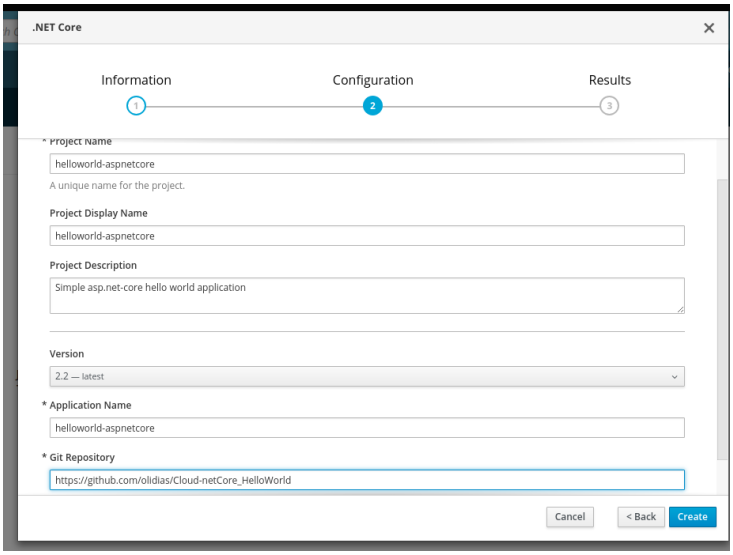


Figure 4: Konfiguration des neuen Projektes

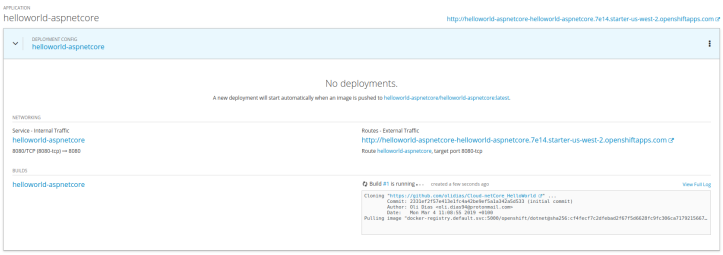


Figure 5: Builden der .NET core Applikation

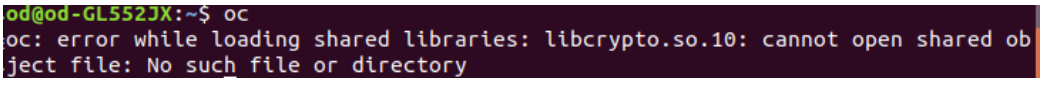


Figure 6: Fehler beim Ausführen des OC

2 Analyse: OSSM-Definition

Damit sich jemand als Cloud Computing Provider ausgeben kann, sollten folgende Chrakteristiken erfüllt sein:

- On-demand
- Self-service
- Scalable
- Measurable

In den folgenden Kapiteln erläutern wir, wie Openshift diese umsetzt.

2.1 On-demand & Self-service

Auf der Startseite findet man eine Katalog aller möglichen Projekttypen. Nach einem Klick auf den gewünschten Projekttyp erscheint ein Dialog, in welchem die spezifische Konfiguration vorgenommen werden kann. Gleichzeitig wird geprüft, ob die definierte Konfiguration plausibel ist. Ist dies der Fall, kann das Projekt erstellt werden. Nach wenigen Sekunden ist das Projekt unter der Rubrik "My Projects" ersichtlich.

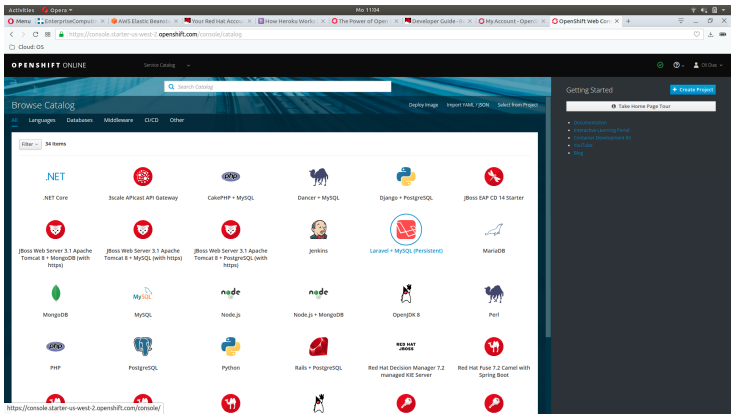
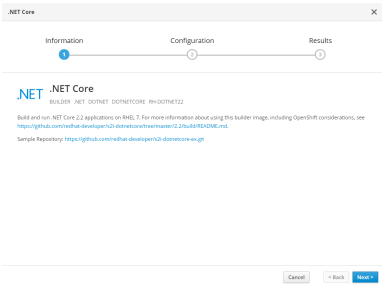
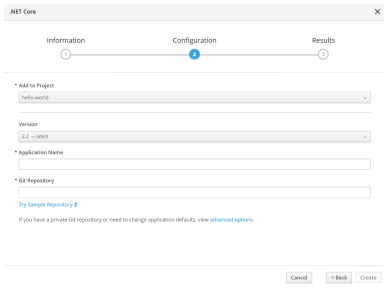


Figure 7: Auswahlkatalog aller möglichen Projekttypen

Die Projektübersicht bietet nebst einigen generellen Informationen die Möglichkeit zum Build und Deployment.



(a) Setup Dialog Schritt 1



(b) Setup Dialog Schritt 2

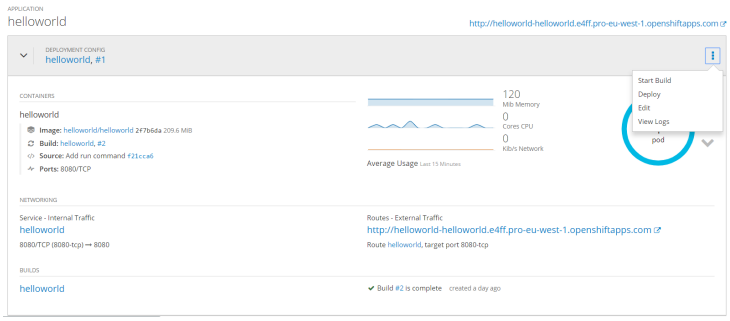


Figure 8: Ansicht der Projektübersicht

2.2 Scalable

Die Nutzung von Openshift erfordert, dass man sich für ein Abonnement entscheidet. Nebst den kostenlosen Einführungsangeboten bedarf die langfristige Nutzung das Abonnement "OpenShift Online Pro". Im Standard bekommt man hierfür folgende Ressourcen:

- 10 Projects
- 2 GB Memory
- 2 GB Terminating Memory
- 2 GB Storage

Um mehr als 10 Projekte zu verwalten bedarf es einem neuen Abonnement. Falls mehr Arbeitsspeicher oder Speicher nötig ist, kann das aktuelle Abonnement angepasst werden, was natürlich einen Einfluss auf den Preis hat. Trotzdem gibt es folgende Begrenzungen:

- Arbeitsspeicher: maximal 48 GB
- Speicher: maximal 150 GB

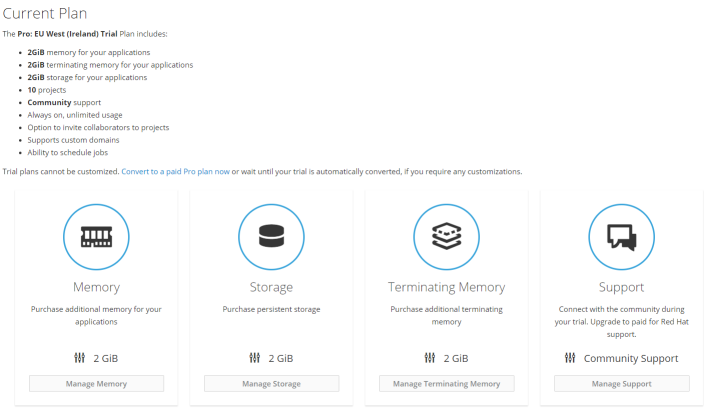


Figure 9: Verwaltung der Ressourcen eines Abonnements

2.3 Measurable

Die aktuelle Nutzung der Ressourcen kann lediglich auf Projektstufe eingesehen werden. Diese Übersicht ist ziemlich einfach gehalten, ledicglich genutzer Arbeitsspeicher und Speicher werden im Verhältnis zum Maximum angezeigt.

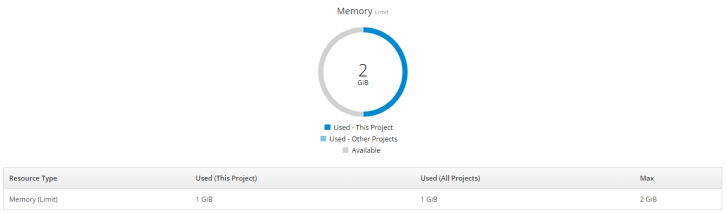


Figure 10: Übersicht der genutzten Ressourcen

3 Konzept: Cloud Computing Patterns

Openshift bietet Entwicklern mit dem Konzept PaaS eine Plattform an, auf welcher eine App relative einfach deployed werden kann. Dies bringt den Vorteil, dass der Entwickler sich nicht mit der Komplexität der Building-/Deploying Infrastruktur auseinandersetzen muss. Die Applikation läuft anschliessend auf sogenannten "Pods", welche vergleichbar mit Docker-Container sind. Einerseits kann die Anzahl Pod's pro Projekt manuell festeglegt und geändert werden. Zudem gibt es die Mgölichkeit für Pod Autoscaling. Sobald ein Pod bis zu einem gewissen Grad ausgelastet wird, kommt ein zusätzlicher Pod in Aktion, sofern die maximale Anzahl definierter Pod's nicht erreicht wurde.

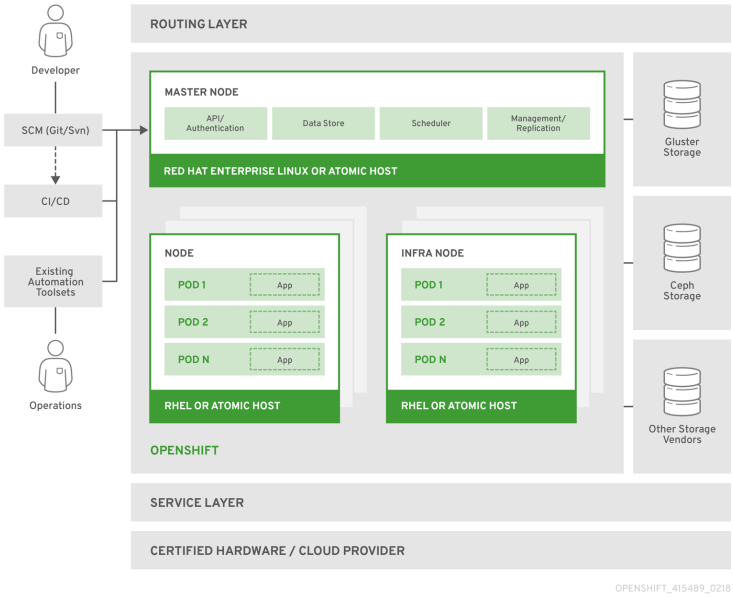


Figure 11: Architekturübersicht der Openshift-Plattform

4 Hands-On: Self Information

Für das Deployment der gegebenen Self-Information Spring Applikation sind ähnliche Schritte wie schon bei der ASP.NET Core Applikation notwendig. Da es sich um eine Java Applikation handelt, muss auf jedenfall eine JVM auf dem Zielrechner existieren. Der Applikationserver Wildfly bietet eine gute Umgebung inklusive JVM (Oracle), auf welcher unsere Applikation ausgeführt wird.

4.1 Vorgehen

Analog zu Kapitel 1.1.2 muss zuerst ein Projekt erstellt werden. Wie erwähnt wählen wir hier aber ein Wildfly-Projekt. Nachdem die nötigen Informationen ausgefüllt wurden, muss das Projekt mit dem GitHub Repository verknüpft werden. Optional kann ein Webhook konfiguriert werden, mit welchem der CI/CD-Prozess nach einem Push auf das Repository automatisch angestossen wird.

Nach dem Erstellen und Verknüpfen des Projektes wird der initale Build und das Deployment automatisch getriggered. Ohne weitere Konfiguration für Openshift wird nun ein Wildfly Applikationsserver aufgezogen und gestartet. Wir müssen jetzt Openshift so konfigurieren, dass das Jar-File vom Build der Spring Applikation ausgeführt und geroutet wird. Dazu reicht ein `run` File im Ordner `.s2i/bin` mit folgendem Inhalt:

```
1 exec java -jar /wildfly/standalone/deployments/self-
    information-0.0.1-SNAPSHOT.jar
```

Mit diesem Befehl wird nach dem Aufziehen des Applikationsserver die Spring Applikation gestartet. Anstelle des absoluten Pfades wäre es denkbar, den Pfad des Jar-Files in eine Openshift-Umgebungsvariable zu speichern.

Nach diesen Schritten sollte die Spring Applikation erfolgreich deployt sein. Im Web-UI kann die URL unter 'Application Console' → 'Applications' → 'Routes' → 'Hostname' gefunden werden.

4.2 Zusatzinformationen

In der Standardkonfiguration des Pro-Abonnements von Openshift Cloud sind bis zu 2 GiB RAM sowie 10 GiB Storage enthalten. Die Self-Information Applikation verbraucht davon ca. 260MiB RAM verteilt auf 2 Pods. Gemäss der laufenden Applikation ist die IP 10.129.29.151.

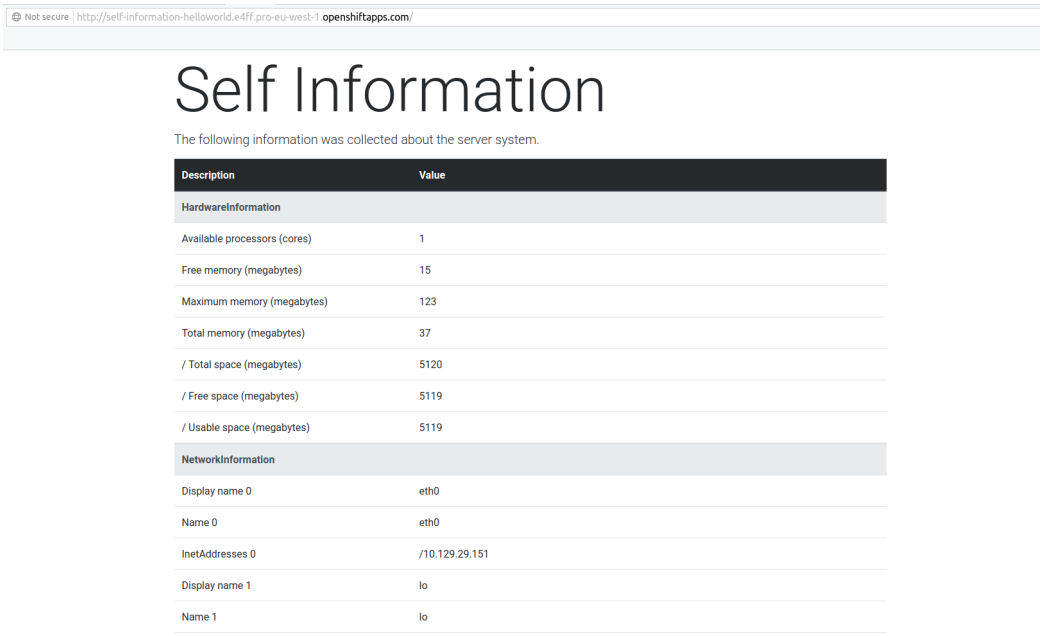


Figure 12: Self-Information Screenshot

5 Analyse: Preisrecherche

Openshift bietet dem Endnutzer nur einige wenige Möglichkeiten am, um ihre Dienste nutzen zu können. Für die geschilderte Ausgangslage bedarf es einer Subskription für den Plan Openshift Online Pro mit zusätzlichen Ressourcen. Openshift bietet in ihrem Standard des Pro Plans lediglich 2 GB RAM und 2 GB Speicher zur Verfügung. Der Erwerb der zusätzlichen Ressourcen führt zu einem monatlichen Kostenpunkt von 118 Dollar. Nebst RAM und Speicher werden terminierter Speicher (welcher nur temporär z.B. für das Builden gebraucht wird) und Support als Ressourcen erworben werden.

Der Rechnungsbetrag kann lediglich mit Kreditkarte beglichen werden, welche beim Kauf hinterlegt werden muss. Um sich lediglich ein Bild der Plattform machen zu können bietet Openshift kostenlos einen 30-tägigen Testzugang an.

Vergleichsweise lässt sich bei Amazon ein Rechner mit ähnlichen Konfigurationen für jährlich 990 Dollar mieten, was ziemlich nahe dem monatlichen Preis von Openshift kommt. Die Schwierigkeit des Vergleichs besteht darin, dass sich die Angebot vieler Provider unterscheiden. Bei Openshift hat man die Freiheit, die Ressourcen sehr genau selbst definieren zu können, wogleich bei Amazon vordefinierte Pakete von Rechner geschnürt wurden.

6 Analyse: Preisvergleich eigenes Hosting, IaaS und PaaS

Mit der geschilderten Ausgangslage wurden folgende Annahmen getroffen:

- Beschaffungskosten eigener Server: 1'000
- Server Administrationskosten pro Jahr: 8'000 (10% Stelle mit Ansatz 80'000 Jahresgehalt)

Daraus ergibt sich folgende Kostenaufstellung:

Kostenart	Eigener Server	Google	Amazon E2C	Openshift
Beschaffung	1'000	3'800	3'600	13'200
Betrieb	16'000	4'000	4'000	4'000
Total	17'000	7'800	7'600	17'200

Table 1: Vergleich zwischen lokalen Servern und Cloud Servern

Im Vergleich schneiden die beiden grossen Cloud Provider Amazon und Google am besten ab. Mit etwas Interpretation lässt sich jedoch erkennen, dass die totalen Kosten stark abhängig von der Anzahl Servern ist. Das heisst, je mehr Server für eine Applikation gebraucht werden, umso mehr lohnt sich die Investition in eine eigene Infrastruktur.

Ein weiterer Aspekt stellt die Vertraulichkeit der Applikationsdaten dar. So erwartet der normale Kunde einer Bank, dass seine Kreditkarteninformationen nicht bei einem Server eines externen Anbieters taummeln.

Generell macht es Sinn, Cloud Provider nach den benötigten Funktionalitäten einzuschränken und erst in einem zweiten Schritt preistechnisch zu optimieren. Eine Herausforderung stellt hierbei der Vergleich der einzelnen Provider dar. Oftmals haben unterschiedliche Provider verschieden Angebote mit unterschiedlichen Konfigurationen, was einen Vergleich sinnlos macht.