Peterchens Mondfahrt an der Cloud vorbei

Wolf G. Beckmann, TEAM

Die Cloud ist ein weites Feld und bietet viele neue Möglichkeiten, vor allem für die Anbieter. Natürlich werden wir zukünftig an der Cloud in den meisten Fällen nicht vorbeikommen, gerade deswegen lohnt es sich, genau anzuschauen, was man als Anwender wirklich braucht. Hier wird zum einen der Entscheidungsweg gegen den Einsatz der Cloud aufgezeigt und dargelegt, mit welchem System dennoch alle Ansprüche des Szenarios an Flexibilität erfüllt wurden und werden. Anschließend wird erläutert, wie die „On-Premises Cloud" funktioniert.

Cloud oder nicht Cloud ...

Der Anstoß für die folgenden Betrachtungen war ein Szenario, wie es jetzt typisch für den Entwicklungsbereich eines Unternehmens ist. Die Serverlandschaft soll modernisiert werden und es stand die Entscheidung an, wie weit Server in die Cloud verlagert werden sollten. Vom Ergebnis waren wir selbst überrascht.

Um eine gemeinsame Sicht auf „die Cloud" zu bekommen, beginnen wir mit einer groben Einteilung, welche Ebenen aus Sicht der Software-Entwicklung die Cloud bietet:

VM / Bare Metal

- Virtuelle Maschinen oder ganze Server.
- Einbindung in eigene Infrastruktur (VPN).
- Entspricht mehr oder weniger dem klassischen Hosting mit mehr Flexibilität.
- Man macht alles selbst, eignet sich daher gut, um vorhandene On-Premises-Szenarien in die Cloud zu verlagern.
- Anbieter (größtenteils) austauschbar, da es jeder im Portfolio hat.

Docker / Kubernetes / Serverless

- Hochskalierbare Systeme, Cloud-Native-Applikationen.
- Aber auch: in Docker gepackte Applikationen.
- Erfordert Um- oder Weiterdenken der aktuellen Entwicklungsstrukturen.
- Anbieter (größtenteils) austauschbar, gerade im Kubernetes-Umfeld.

Special Services

- KI-Systeme, Sprach-Dienste, Übersetzungsservices, Map-Services, alles Mögliche
- auch Datenbanken
- stark anbieterspezifisch

Das Software- und Consulting-Team hat schon früh damit begonnen, die Entwicklung nicht mehr direkt auf dem eigenen Rechner zu erledigen, sondern auf lokale virtuelle Maschinen zu verlagern, um zum einen neue Entwickler durch einfaches Kopieren einer VM in ein Projekt zu holen und zum anderen Projektumgebungen zu konservieren.

Zusätzlich wurde der Fokus für neue Projekte auf die Paketierung mit Containern wie Docker gelegt, da so die Installation und der Betrieb stark vereinfacht werden. Die gemeinsam genutzten Docker-Container wurden vom Team selbst mit einem Open-Source-Produkt von Red Hat betrieben: OpenShift.

So sind auch die ersten Tools fürs Development nicht wie üblich auf einem virtuellen Server, die von der internen Systembetreuung vergeben werden, installiert worden, sondern in Container gewandert. Einer der ersten Server, die in einem Container betrieben wurde, war beispielsweise ein Nexus-Server (hält die Maven Repositories für die Java-Entwicklung).

Nicht zuletzt hat das Team *[1]* auch mit „Special Services" zu tun (beispielsweise der Datenbank oder durch die Entwicklung von Voice- und Chatbot-Systemen mit Digital-Assistant-Services von verschiedenen Cloud-Anbietern). Allgemein wurden bei verschiedenen Cloud-Anbietern diverse Systeme mit unterschiedlichen Komplexitätsgraden aufgesetzt; natürlich auch in der Oracle-Cloud.

Grundsätzlich waren für die Modernisierung der Infrastruktur des Software- und Consulting-Teams alle Zeichen für die Cloud gesetzt. Es wurde eine Diskussionsrunde mit der Systembetreuung, der Geschäftsführung (als Geldgeber) und dem Software- und Consulting-Team gestartet. In dieser sollte auf Basis unserer Anforderungen eine konkrete Lösung definiert werden.

Die abzudeckenden Anforderungen, die in der Diskussionsrunde besprochen wurden, sind wohl typisch für Entwicklungsabteilungen:

Für Projekte sollen eigenständige Umgebungen aufgebaut werden

Meist besteht ein Projekt nicht nur aus einem Server. Es umfasst verschiedene Komponenten. In der klassischen Entwicklung wären das ein Datenbank-Server, ein Application-Server, gegebenenfalls native Dienste und eine Handvoll Shell-Scripts, die Zeit- oder Event-gesteuert ablaufen.

Für solch ein Projekt werden in der Entwicklung mindestens zwei solcher Umgebungen benötigt: Development und Test. Diese Umgebungen müssen unabhängig voneinander laufen und natürlich gibt es nicht nur ein Projekt.

Systeme für die Unterstützung der Entwicklung werden benötigt

Die Entwicklung wird immer abhängiger von zusätzlichen Systemen, wie der Versionsverwaltung (GIT), Application-Lifecycle-Management-Tools (Jira oder Polarion), Continous Integration Server (Jenkins), aber auch ein Wiki für das Wissens-Management wird benötigt.

Beispiel-Systeme sollen Kunden zur Verfügung gestellt werden

In Akquise-Phasen sagt ein Bild oft mehr als tausend Worte; in der Software-Entwicklung eine kleine Beispiel-Anwendung. Diese muss natürlich auch für den potenziellen Kunden zugreifbar sein.

Einfach und schnell sollen Test-Systeme auf- und wieder abgebaut werden können

In der Entwicklung muss man sich häufig mit neuen Technologien auseinandersetzten, da sich die Welt weiterdreht. Auch hierbei werden häufig komplexe Umgebungen gebraucht. Da soll der Aufbau der Umgebung nicht länger als der Test dauern. Deshalb wären Templates oder der Umgebungsaufbau über Skripte wünschenswert. Auch eine Beantragung einer Testumgebung, die sich über viele Diskussionsrunden erstreckt, sorgt dafür, dass die Weiterbildung auf der Strecke bleibt.

Konkret wollte der Entwicklungs-Bereich eine eigene „Spielwiese“ haben.

Tatsächlich hat sich in dieser Besprechungsrunde herauskristallisiert, dass sich ein anderes Konzept als die Cloud für das Team besser rechnet, besser passt und auch für die Infrastruktur (Systemadministration) der Firma weniger Aufwand bedeutet:

Der interne OpenShift-Server wird vergrößert und es wird bei einem klassischen Hoster (kein Cloud-Anbieter) ein weiterer OpenShift-Server aufgesetzt.

Wie kam es zu der Fokussierung auf OpenShift?

Dazu müssen mehrere Punkte beachtet werden. Der wohl wichtigste Punkt ist:

Es ist keine Entscheidung gegen die Cloud, sondern eine Entscheidung für die konsequente Nutzung moderner Cloud-Technologien.

Ressourcenschonend

Es sollte nicht mehr in klassischen Strukturen, also in Servern (Rechnern) gedacht werden, sondern in flexiblen, ressourcenschonenden Systemen.

Aufgrund des Einsatzes von Containern statt von VMs wird der benötigte Bedarf an Speicher und CPU drastisch reduziert. In der Entwicklung wird beispielsweise als GIT-Server „Gitea“ verwendet. Ein Open-Source-Server, der eine ähnliche Funktionalität wie GitHub mit Managementinterface, Pullrequests etc. zur Verfügung stellt. Wenn dieser Server in einer eigenen VM betrieben wird, verbraucht das Betriebssystem mehr Speicher und CPU als Gitea selbst. Bei anderen Systemen ist die Elastizität groß; wenn das System eingesetzt wird, braucht es kurzzeitig viel Ressourcen, aber die meiste Zeit nicht. Bei einem einfachen VMware-System muss CPU und Speicher fest zugewiesen werden (zumindest bis zum Neustart), das bedeutet, es muss auf die Lastspitzen ausgerichtet werden. Die Virtualisierung über Docker sieht vor, dass alle Systeme gemeinsam laufen und nur dahingehend abgegrenzt werden, dass sich die Systeme, vereinfacht gesagt, gegenseitig nicht sehen. Alle Container teilen sich CPU und Speicher dynamisch und somit wird auch die Last dynamisch verteilt.

Natürlich kann eingestellt werden, welche maximalen Ressourcen ein Prozess bekommt, damit ein System nicht alle anderen herunterziehen kann.

Um es konkret aufzuzeigen, das externe System besitzt 12 Kerne, 64 GB RAM und 1,5 TB Plattenplatz. Es laufen auf dem System mehrere Oracle-XE-Datenbanken, Payara und JBoss Server, ein Eclipse-Che-System, ein Node.js-Server, ein Wiki-System, ein Gitea-Server, ein Jenkins-Server, Postgres-Datenbanken und noch einiges mehr.

Dennoch ist der Server noch lange nicht ausgelastet, wie in *Abbildung 1* zu sehen ist.

Es ist kein Produktiv-System, deshalb werden alle Systeme nur kurzfristig verwendet und alle Systeme reagieren schnell.

Vereinfachung von Installation und Betrieb

Tatsächlich hat sich gezeigt, dass die meisten Systeme, die wir benötigen, als Docker-Images angeboten werden oder sich leicht selbst in Containern paketieren lassen.

Durch den Einsatz von OpenShift sind die Administration und das Monitoring sehr einfach, es lassen sich auch ohne großen Aufwand ganze Infrastrukturen aufbauen und für wiederkehrende Situationen können Templates angelegt werden.

Ein weiteres Feature ist die Betriebssicherheit. OpenShift kann die Container überwachen und sobald ein Container nicht mehr reagiert, wird er neu gestartet. Im einfachsten Fall gibt man eine Test-URL an, die nur irgendeine Webseite zurückliefern muss.

```
top - 15:35:17 up 110 days,  7:11,  1 user,  load average: 2.41, 2.03, 2.08
Tasks: 1110 total,   1 running, 1066 sleeping,   0 stopped,  43 zombie
%Cpu(s): 11.8 us,  8.0 sy,  1.2 ni, 78.0 id,  0.9 wa,  0.0 hi,  0.1 si,  0.0 st
KiB Mem : 65773264 total,  6407396 free, 33043040 used, 26322828 buff/cache
```

Abbildung 1: TOP eines OpenShift-Test-Servers (Quelle: Wolf G. Beckmann)

Kosten

Aufgrund anderer Anforderungen außerhalb des Teams war die Weiterführung des Betriebs einer internen Virtualisierungslösung auf Basis von VMware fest vorgegeben. Deshalb kam es zu der Zweiteilung in eine interne und eine externe OpenShift-Installation. Dienste, die ausschließlich intern verwendet werden, bleiben intern. Ansonsten werden sie auf dem gehosteten OpenShift-Server installiert.

Da es sich bei dem externen Server also primär um ein Testsystem handelt, konnten wir günstige Systeme suchen. Fündig wurden wir bei einem Server-Hoster, der Systeme, die für Kundenwünsche zusammengestellt wurden, nach ihrer Laufzeit günstig anbietet. So wurde das oben beschriebene System (12CPUs, 64 GB RAM 1,5 TB Platte) dort für 50€ pro Monat angeboten. In diesem Preissegment konnten die Cloud-Anbieter nicht mithalten.

Selbst der Einwurf, Äpfel mit Birnen zu vergleichen, „Gebraucht gegen Neu", hilft nicht, da das System für die Anforderungen passend ist. Bei einem Produktivsystem wäre der Einsatz eines Servers gegebenenfalls auch von einem Cloud-Anbieter in Betracht gezogen worden.

OpenShift – die „On-Premises Cloud"

OpenShift ist ein Produkt von Red Hat und wird sowohl kostenfrei als OKD als auch kostenpflichtig angeboten. Das Cloud-Angebot von Red Hat basiert auf OpenShift.

OpenShift ist ein Aufsatz auf Kubernetes. Kubernetes selbst ist ein System, um (Docker)- Container zu managen. Dazu werden unter anderem folgende Features angeboten:

- Virtuelle Umgebungen
  Es werden mehrere Container in einem privaten Netzwerk zusammengefasst. Dabei können die Container sich über Namen gegenseitig adressieren.
- (Auto-) Skalierung
  Jeder Container kann mehrfach gestartet werden. Intern findet ein Loadbalancing statt. Die Skalierung kann auch aufgrund von Ressourcennutzung automatisch über Kubernetes erfolgen.
- Ausfallsicherheit
  Kubernetes kann testen (z.B. über einen URL-Aufruf), ob der Container funktioniert. Wenn nicht, wird er automatisch neu gestartet.
- Clusterfähigkeit
  Die Container können über mehrere Nodes (Cluster-Knoten) verteilt werden.

OpenShift erweitert Kubernetes noch um weitere Features, beispielsweise:

- Einfaches Management und Monitoring-Web-Interface
  Durch OpenShift können im Browser einfach Projekte angelegt und administriert werden. Dazu gleich mehr.

- Benutzerverwaltung
- Templates
  Wir haben beispielsweise Templates für eine einfache Oracle-XE-Datenbank oder eine Oracle-XE-DB mit Apex (inkl. https-Zugang).
- Rolling Deployments
  Bei einem Update eines Containers wird erst der neue Container hochgefahren. Erst wenn er voll funktionsfähig ist, werden die Zugriffe auf den neuen Container umgeleitet und der alte Container heruntergefahren. Das vermindert die Ausfallzeit bei Updates.

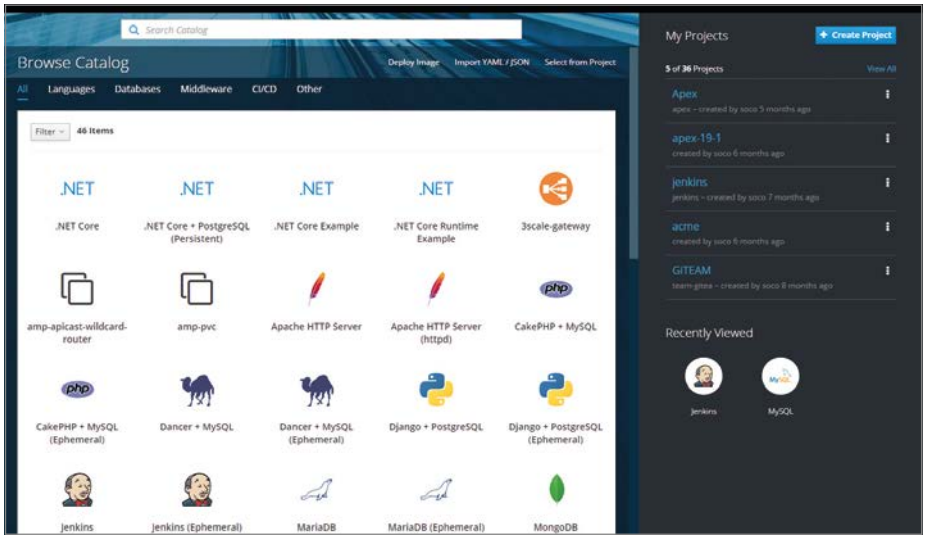

Abbildung 2: Einstiegsseite von OpenShift (Quelle: Wolf G. Beckmann)

Es gibt noch eine Menge weiterer Features, die im Rahmen der Entwicklung sehr spannend sind, die hier allerdings aufgrund von Platzmangel nicht weiter behandelt werden.

Die Installation eines einfachen Ein-Server-Systems war kein Problem. Neueinsteiger finden dazu auch YouTube-Videos, die von Red Hat selbst angeboten werden.

Nach der Installation von OpenShift (OKD) steht dem Benutzer eine Einstiegsseite, wie sie in *Abbildung 2* zu sehen ist, zur Verfügung.

Die Templates werden links angezeigt, über die Leiste rechts kann direkt auf die bereits vorhandenen Projekte zugegriffen werden. Die Darstellung des Status eines Containers ist in *Abbildung 3* zu sehen (hier eine Jenkins-Anwendung).

Um beispielsweise so einen Jenkins-Server in OpenShift aufzusetzen, muss lediglich im Startfenster das Jenkins-Template angeklickt werden, im Browser wird ein Projektname eingegeben sowie einige (wenige) Grundeinstellungen vorgenommen und der Jenkins-Server steht in weniger als fünf Minuten zur Verfügung.

Erstellen einer individuellen Projektumgebung

In OpenShift stellen Projekte Umgebungen dar. In einem Projekt existieren mehrere Container, die sich gegenseitig über Aliasse ansprechen können. Von außen besteht aber kein direkter Zugriff. Openshift bietet Router, die erlauben, dass ein Server von außen über eine konfigurierbare URL erreichbar ist. In der Umgebung aus *Abbildung 4* gibt es beispielsweise keinen direkten Zugriff auf den MySQL-Container.

Um darzustellen, wie einfach mit OpenShift-Projektumgebungen aufgebaut werden können und dass sich OpenShift auch wie „Cloud" anfühlt, wird im Folgenden der Aufbau einer einfachen Projektumgebung Schritt für Schritt erläutert. Das Beispielprojekt besteht aus folgenden Bausteinen:

- Datenbank-Server (MySQL)
- Management-Interface für die Datenbank (MyPHP-Admin)
- PHP-Server mit der Anwendung

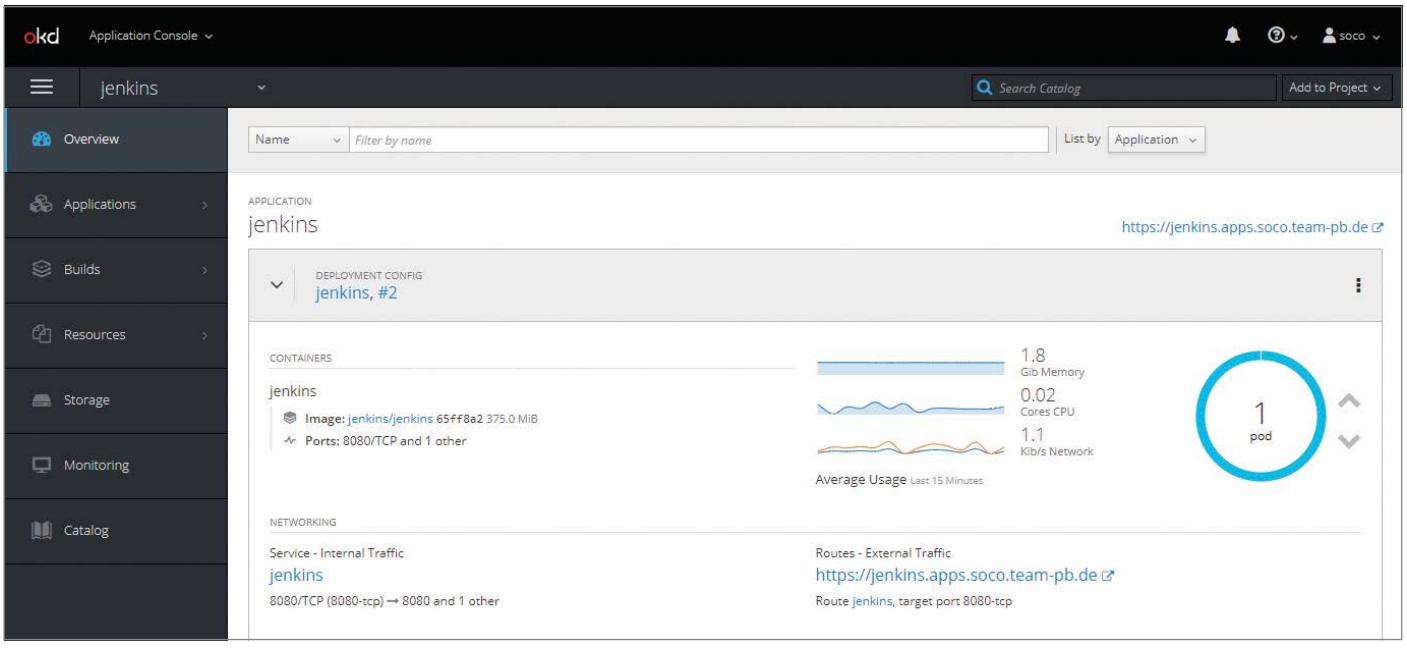

Abbildung 3: Status eines Jenkins-Servers (Quelle: Wolf G. Beckmann)

1. Erstellung eines Projektes

Auf der Startseite wird über den Button „+ Create Projekt“ der Wizard für die Projekterstellung aufgerufen. Hier braucht lediglich der Name des Projektes eingegeben werden. Danach wechselt man in das neu erstellte Projekt.

2. Erstellen der Datenbank

Über den Button oder das rechte „Add to Project“-Menü mit „Browse Catalog“ das Template „MySQL“ auswählen. Die wichtigsten Eingaben sind Connection Username, Password und das Root Password, natürlich können auch Datenbank-Name etc. angepasst werden. Auch hier sind keine weiteren Angaben notwendig.

3. Erstellen einer Management-Oberfläche für die Datenbank

Es soll MyPHP-Admin eingesetzt werden, aber OpenShift bietet dafür keinTemplate, also wird ein Image aus dem Docker Hub „phpmyadmin/phpmyadmin“ verwendet. Es wird direkt von den Autoren bereitgestellt. Laut der Dokumentation im Docker Hub (*https://hub.docker.com/r/phpmyadmin/phpmyadmin*) wird die Datenbank-Verbindung über folgende Variablen gesetzt: PMA_HOST=mysql (war vorbelegt bei der Anlage der Datenbank), MYSQL_ROOT_PASSWORD=rootpassword (wurde angegeben bei der Anlage der Datenbank).

Da ansonsten der Standard passend ist, werden keine weiteren Parameter gesetzt. Um das Image dem Projekt hinzuzufügen, wählt man im OpenShift-Projekt über das „Add to Project“-Menü oben rechts „Deploy Image“ und gibt das Image an, wie in *Abbildung 5* zu sehen ist.

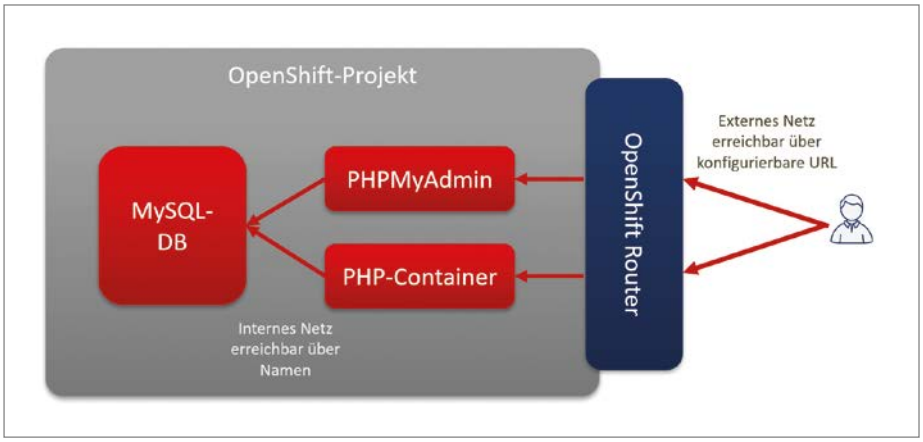

Abbildung 4: Beispielumgebung in OpenShift (Quelle: Wolf G. Beckmann)

Nach dem Anklicken der Lupe können weiter unten in dem Wizard die Variablen eingegeben werden, wie es in *Abbildung 6* zu sehen ist. Über „Deploy“ wird das Image deployt.

Nach kurzer Zeit ist das Deployment fertig. Um von außen auf diese Admin-Oberfläche zugreifen zu können, muss eine Route erstellt werden. Auch hier sind nicht mehr als drei Klicks notwendig, über „Create Route“ wird die Route erstellt. Die primäre Angabe ist die gewünschte URL, unter der der Service erreichbar sein soll. Die URL ist typischerweise eine Subdomain des Servers.

Anschließend kann im Browser nun PHP MyAdmin aufrufen werden. Wenn die Management-Oberfläche nicht mehr benötigt wird, kann die Anzahl der PODs (Container) auf 0 gesetzt werden. Dann existiert lediglich noch die Konfiguration, die Software selbst läuft aber nicht mehr.

Bis zu diesem Punkt dauert es ca. fünf Minuten.

4. Installieren der PHP-Anwendung

OpenShift bietet hier für eine Vielzahl von Programmiersprachen (Java, JavaScript (node.js), Python, PHP, Django...) vorgefertigte Templates, die den Source einer Anwendung in einer Laufzeitumgebung als Container paketieren. OpenShift nennt die Technologie S2i (Source to Image). Als Beispielanwendung fungiert eine einfache CRUD-Demo in PHP, sie ist zu finden unter *https://github.com/wgbeckmann/php-mysql-crud*. Die Anwendung liest über Umgebungsvariablen des Containers (DBSERVER, DBUSER, DBPASSWORD, DBNAME) die Information zur Datenbank-Verbindung aus.

Somit wird die Anwendung über das PHP-Template installiert. Die Angaben zur Installation sind das GIT-Repository und in

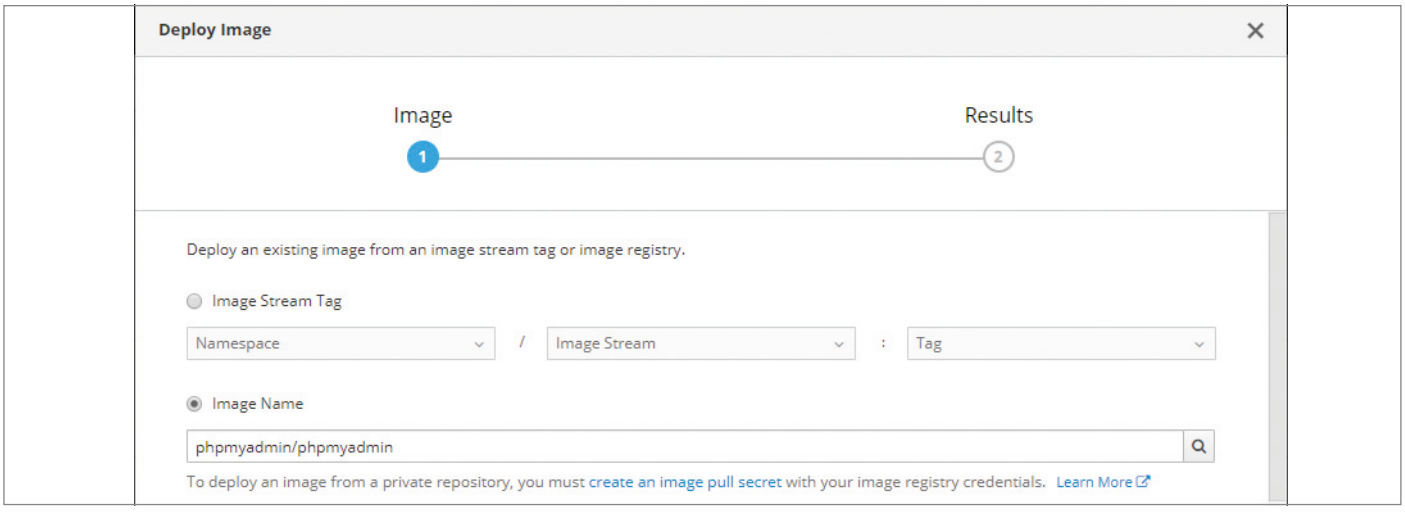

Abbildung 5: Erstellen eines Containers über Image aus Docker Hub (Quelle: Wolf G. Beckmann)

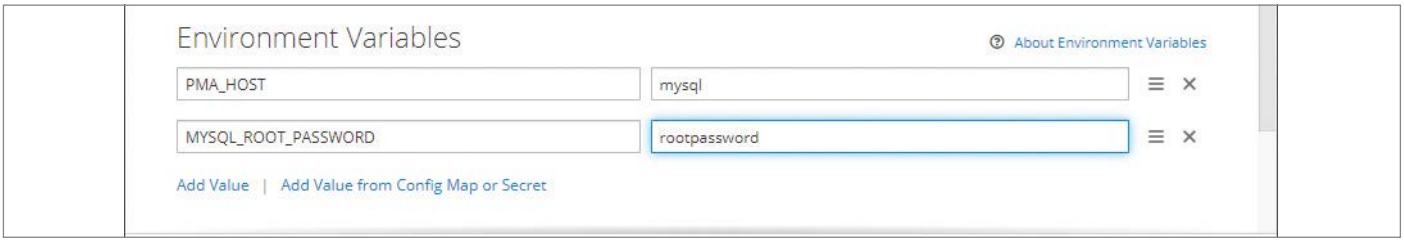

Abbildung 6: Angabe von Umgebungsvariablen für einen Container (Quelle: Wolf G. Beckmann)

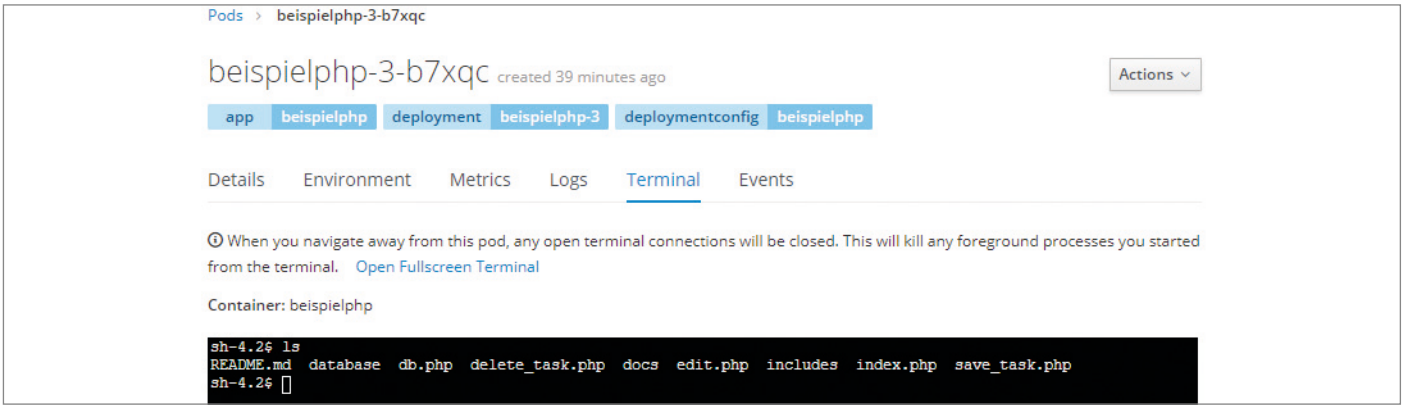

Abbildung 7: Über den Browser kann direkt eine Shell im Container ausgeführt werden. (Quelle: Wolf G. Beckmann)

der erweiterten Konfiguration werden die vier Umgebungsvariablen gesetzt. Danach erstellt OpenShift selbstständig einen passenden Container. In dem Template ist schon das Erstellen der Route vorhanden.

Nachdem über die Management-Oberfläche (PHP MyAdmin) die notwendigen Tabellen in der MySQL-Datenbank erzeugt wurden, kann die Anwendung aufgerufen werden. Alles zusammen dauert weniger als zehn Minuten.

Einbinden beliebiger Applikationen als Container

Wie bei der Installation von PHP MyAdmin gezeigt wurde, können nahezu beliebige Container in OpenShift aus dem Docker Hub installiert werden. Natürlich können auch selbsterstellte Container verwendet werden. Dazu müssen sie lediglich im Repository der OpenShift-Installation deployt werden. So haben wir beispielsweise Container für eine XE-Datenbank mit und ohne Apex erstellt. Zum einfachen Deployment können wiederum OpenShift-Templates erstellt werden.

Für einfache Tests rechnet sich der Aufwand für die Erstellung von Containern nicht immer. Deshalb haben wir einen Container mit einer Oracle-Linux-Installation als Template erstellt. Da OpenShift, wie in *Abbildung 7* zu sehen, den Zugriff auf die Shell innerhalb des Containers über die Administrationsoberfläche erlaubt, kann dann einfach in dem Container beliebige Software installiert werden.

Fazit

Über den Einsatz von OpenShift hat das Entwicklerteam seine eigene Cloud mit den Vorteilen aus allen Welten bekommen:

- Konsequenter Einsatz von Cloud-Technologien
- Kein administrativer Aufwand (Kostenkalkulation und Beantragung), um Testprojekte aufzusetzen
- Festigung und Erweiterung des DevOps-Wissens im Team
- Einfache Bedienung und Betrieb über eine Web-Oberfläche
- Schnelles Aufsetzen von Umgebungen
- Sehr kostengünstig

Ausblick

Kubernetes ist die typische Betriebsumgebung für Cloud-Native-Applikationen. Somit ergibt es Sinn, bei neuen Applikationen direkt Cloud-Native-Architekturen anzuwenden, selbst wenn sie (noch) nicht in der Cloud betrieben werden.

Über den Autor

Wolf Beckmann ist Teamleiter des Software- und Consulting-Teams bei der TEAM GmbH und Consultant primär in Migrationsprojekten aus der PL/SQL-Welt in die Java-Welt.

Anmerkung

[1] Der Begriff Team ist in unserem Hause nicht einfach, da einmal von TEAM (unserer Firma) und auch vom Team (dem Software- und Consulting-Team) gesprochen wird. Der Leser möge auf die Schreibweise mit Großbuchstaben achten.

Wolf G. Beckmann
wb@team-pb.de