OpenShift

Thomas Witte
thomas.witte@uni-ulm.de

Zusammenfassung—Cloud Computing und im speziellen Platform as a Service Dienste ermöglichen kostengünstig und energieeffizient Webanwendungen zu hosten und somit für kleine Unternehmen finanzielle Risiken durch die Anschaffung von Hardware zu minimieren. Die Arbeit beschäftigt sich mit OpenShift, einer seit 2011 entwickelten, quelloffenen PaaS Plattform und bietet einen Überblick über den Aufbau einer OpenShift Anwendung, die Verzeichnisstruktur, Deploymentmechanismen und Möglichkeiten der Skalierung. Im zweiten Teil wird die zu Grunde liegende Architektur, Redundanzaspekte und die logische Unterteilung in Nodes und Broker sowie die Kommunikation zwischen diesen Teilen näher betrachtet.

1 Einführung

Der Begriff Cloud Computing ist sehr weit gefasst und wird derzeit für fast jeden Internetdienst oder netzwerkbasierten Service genutzt. Eine deutlich bessere wenn auch oft nicht eindeutige Einordnung dieser Dienste bieten die Begriffe „Software as a Service“ (SaaS), „Platform as a Service“ (PaaS) und „Infrastructure as a Service“ (IaaS) [13]. Der Dienst wird dabei durch die Ebene im Hardware-/Software-Stack charakterisiert auf der der Kunde ihn nutzen kann. Abbildung 1 verdeutlicht diese vertikale Unterteilung und liefert Beispiele für Dienste die in die jeweilige Kategorie fallen. Diese Arbeit betrachtet Cloud Computing fast ausschließlich auf der PaaS Ebene und bietet einen Einblick in den OpenShift PaaS Dienst sowie die zu Grunde liegende Technologie.

PaaS-Dienste wie OpenShift empfehlen sich gerade für kleinere Firmen und Internetplattformen, deren Geschäftsmodell und Erfolgsaussichten noch nicht feststehen. Dank des einfachen Deployments können die Entwickler schnell eine erste Version der Seite verfügbar machen, ohne durch die Beschaffung von Hardware oder langfristige Mietverträge für Server in Rechenzentren finanzielle Risiken tragen zu müssen. Solange die Seite wenig bekannt und besucht ist werden keine Kosten dafür fällig; nimmt die Popularität schnell zu können schnell und automatisiert weitere Ressourcen verfügbar gemacht werden.

In Abschnitt 2 werden zu OpenShift ähnliche, konkurrierende Dienste aufgelistet. Es folgt ein Blick auf die Funktionen von OpenShift in Abschnitt 3 sowie auf einige OpenShift Interna in Abschnitt 4. Eine kurze Zusammenfassung liefert schließlich Abschnitt 5.

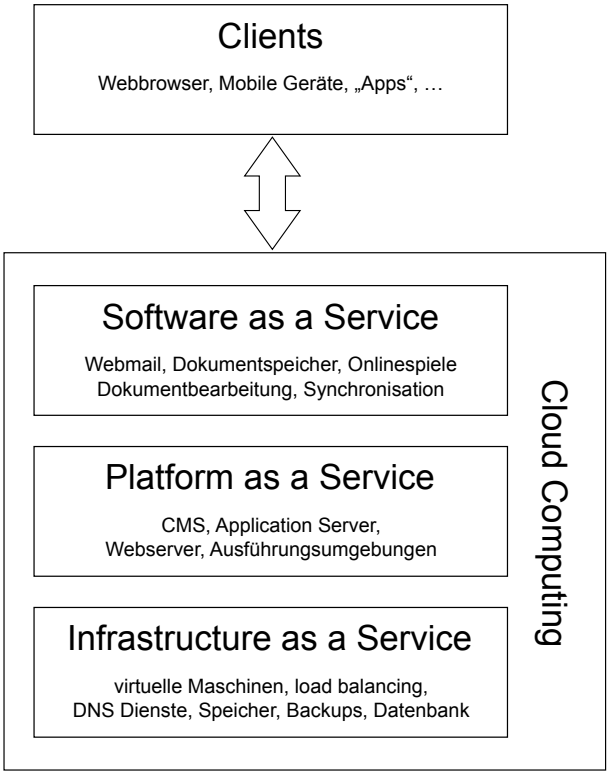

Abbildung 1: Kategorisierung verschiedener Cloud Computing Dienste.

1.1 Geschichte von OpenShift

Im November 2010 übernahm RedHat den kalifornischen Cloudspezialisten Makara. Dessen Deployment- und Managementlösungen wurden in Verbindung mit RedHat eigenen Technologien zur Plattform OpenShift vorangetrieben [18]. Nach der offiziellen Ankündigung im Mai 2011 wurden kostenlose Zugänge zur Developer Preview angeboten; seit November 2012 existiert die finale, auf OpenShift Origin beruhende Version [6].

OpenShift war und ist dabei immer wieder größeren Änderungen unterzogen: so wurde die ursprüngliche Dreiteilung in OpenShift Express, OpenShift Flex und OpenShift Power aufgegeben und es werden mittlerweile nur noch OpenShift Online zur privaten und kommerziellen Nutzung in der RedHat Cloud sowie OpenShift Enterprise zur Nutzung der OpenShift Technologie in eigenen Rechenzentren durch RedHat angeboten [7].

1.2 OpenShift Origin

Als OpenShift Origin wurde der Quellcode der Plattform OpenShift sowie zugehöriger Clienttools unter der Apache 2.0 Lizenz freigeben [9][3], sodass nicht nur eine in geringem Umfang kostenlose Nutzung von OpenShift auf den Servern von RedHat möglich ist, sondern auch das Aufsetzen, Nutzen und Weiterentwickeln der Plattform unabhängig von RedHat.

2 Übersicht anderer PaaS Anbieter

Neben OpenShift existieren eine vielzahl weiterer PaaS Angebote, die teilweise einen sehr ähnlichen Fokus, teilweise aber auch andere Schwerpunkte besitzen.

Zu den wichtigsten Alleinstellungsmerkmalen von OpenShift gegenüber konkurrierenden Produkten wie AWS Elastic Beanstalk von Amazon [1], Microsofts Azure Cloud [8] und Googles App Engine [5] zählen die quelloffene Plattform sowie das Ausführen beliebigen Codes, solange dieser unter Red Hat Enterprise Linux (RHEL) lauffähig ist und geltende SELinux Richtlinien nicht verletzt. Dadurch können statt aus einer begrenzten Anzahl angebotener Plattformen und Programmiersprachen zu wählen fast beliebige genutzt werden [14]. Zu beachten ist dabei jedoch, dass sich die in Abschnitt 3.4 angesprochenen Möglichkeiten zur automatischen Skalierung über mehrere Instanzen beziehungsweise Ausführungseinheiten (hier „Gears“ genannt) auf Anwendungen beschränken, die einen der mitgelieferten, bereits vorkonfigurierten Anwendungsserver („Cartridge“) nutzen.

AWS Elastic Beanstalk wurde als Erweiterung der IaaS-Dienste Amazons konzipiert und nutzt daher direkt Amazon EC2, Elastic Load Balancing, Auto Scaling und S3 Storage. Zur Nutzung des PaaS Dienstes fallen keine weiteren Kosten außer für die Nutzung der zu Grunde liegenden Infrastruktur an. Als Anwendungsserver werden nur Apache (PHP, Python), Passenger (Ruby), Tomcat (Java) und IIS (.NET) unterstützt.

Microsoft bietet mit Azure Cloud wie Amazon sowohl PaaS als auch IaaS Dienste. Der PaaS Dienst unterstützt dabei offiziell Python, Java, node.js, .NET und PHP; weitere Sprachen werden durch die Community unterstützt. Zur Speicherung persistenter Daten steht SQL Azure zur Verfügung.

Die geringste Flexibilität bietet Googles App Engine, die nur Java oder Python als Sprache ermöglicht. Auf die Konfiguration des Anwendungsservers oder der Datenbank besteht dabei kein direkter Einfluss; die Kommunikation erfolgt rein mittels Standards wie der Servlet-API oder der JPA.

OpenShift ist zusätzlich der einzige der genannten Services, dessen Software vollständig quelloffen ist, sodass kein „Vendor-Lock-in“ besteht und gehostete Anwendungen ohne Anpassungen auf eigene Server oder zu anderen Anbietern migriert werden können.

3 OpenShift aus Nutzersicht

Im folgenden wird OpenShift aus der Sicht eines Benutzers geschildert. Benutzer ist in diesem Fall der Programmierer oder Betreiber eines Internetdienstes, dessen Ziel es ist, möglichst einfach und schnell seine Anwendung im Internet verfügbar zu machen oder zu testen. PaaS-Dienste bieten hierbei vielerlei Vorteile: Es entstehen keine Kosten für die Anschaffung von Serverhardware. Auch die Betriebskosten bleiben bei geringer Bekanntheit oder Nutzung der Anwendung gering. Ändert sich die Last durch steigende Bekanntheit oder nachlassendes Interesse lässt sich die verfügbare Rechenleistung sehr schnell erhöhen oder senken. Die Entwickler werden entlastet, da der Aufwand für Konfiguration und Pflege des gesamten restlichen Soft- und Hardwarestacks entfällt. Durch Lastverteilung und höhere Auslastung der Server kann zudem Energie gespart werden.

Aus diesem Grund gehen die folgenden Abschnitte auf mögliche Plattform/Sprachkombinationen, nötige Schritte zur Vorbereitung der eigenen Anwendung für OpenShift sowie deren Deployment ein.

3.1 Mögliche Plattformen/Cartridges

Eine Anwendung in OpenShift beinhaltet neben dem Code des Nutzers ein oder mehrere vorkonfigurierte Softwarepakete, sogenannte Cartridges. Hierbei stehen einige, hier näher vorgestellte, Cartridges als Teil von OpenShift Origin immer zur Verfügung; speziellere Plattformen oder Software können entweder in Form von Community-Cartridges, oder selbst durch „Do It Yourself“-Cartridges bereitgestellt werden [12].

Die offiziell bereitgestellten Cartridges lassen sich in zwei Kategorien einteilen:

Web-Cartridges: Hierunter fallen Ausführungsumgebungen wie PHP, Python, ruby, node.js oder Perl, sowie Server wie Tomcat, JBoss AS oder Zend. Alternativ kann auch eine leere Anwendung mittels der DIY-Cartridge („Do-It-Yourself“) erstellt werden.

Addon-Cartridges: Dies sind Datenbanken wie MySQL, MongoDB oder PostgreSQL sowie weitere Komponenten wie ein Cron-Dienst, Monitoring-Dienste oder Managementprogramme wie phpMyAdmin.

Jede Anwendung muss dabei genau eine Web-Cartridge nutzen. Diese muss beim Erstellen der Anwendung angegeben weden und lässt sich nicht ändern. Addon-Cartridges können in beliebiger Kombination sofort oder zur Laufzeit hinzugefügt oder entfernt werden [17, S. 28].

3.2 Vorbereitung der eigenen Anwendung

Die Anpassung einer eigenen Anwendung zur Nutzung von OpenShift erfolgt in wenigen Schritten.

Zuerst müssen Dateipfade im Code so angepasst werden, dass Dateien im persistenten Speicher gespeichert werden. Jedem Gear steht dabei ein

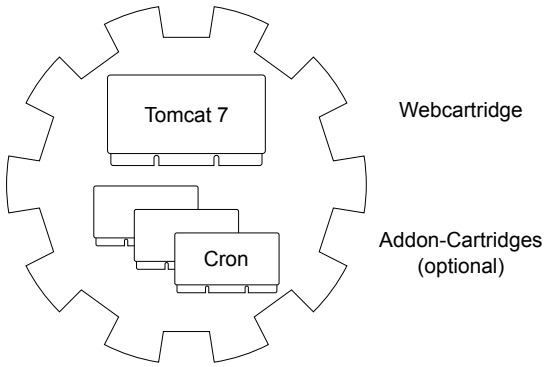

Abbildung 2: Aufbau einer nicht skalierenden, auf Tomcat basierenden Anwendung.

Gigabyte persistenter Speicher zur Verfügung [17, S. 14], der entsprechende Verzeichnispfad wird als Umgebungsvariable `OPENSHIFT_DATA_DIR` bereitgestellt. Werden Dateien außerhalb dieses Verzeichnisses gespeichert, gehen diese spätestens bei einer Migration auf einen anderen Node oder dem zwischenzeitlichen Deaktivieren der Anwendung verloren. Eine grobe Übersicht über die Verzeichnisstruktur einer OpenShift-Instanz wird in Abbildung 3 gezeigt.

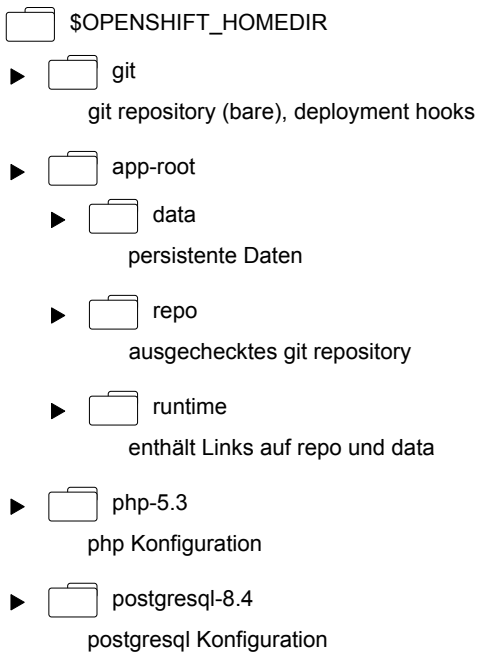

Abbildung 3: Beispielhafte Verzeichnisstruktur eines OpenShift Gears mit installierten PHP und PostgreSQL Cartridges [10].

Weitere Umgebungsvariablen enthalten Pfade zu Log-Verzeichnissen, Datenbankzugangsdaten, etc.; eine Übersicht der wichtigsten Variablen liefert Tabelle 1.

Alle Verzeichnispfade außer `OPENSHIFT_TMP_DIR` sind relative Pfade bezüglich `OPENSHIFT_HOMEDIR`.

Möglicherweise notwendige Konfigurationsänderungen der Cartridges beziehungsweise der Import eines

Umgebungsvariable	Beschreibung
OPENSHIFT_HOMEDIR	Basisverzeichnis der OpenShift Anwendung
OPENSHIFT_DATA_DIR	Verzeichnis in dem persistente Daten gespeichert werden können
OPENSHIFT_TMP_DIR	Verzeichnis für temporäre Dateien
OPENSHIFT_REPO_DIR	Verzeichnis, in das die Anwendung deployed wurde
OPENSHIFT_X_LOG_DIR	Logverzeichnis der Cartridge X
OPENSHIFT_X_DB_LOG_DIR	Logverzeichnis der Datenbank X
OPENSHIFT_APP_DNS	DNS-Adresse der Anwendung
OPENSHIFT_APP_NAME	Name der Anwendung
OPENSHIFT_INTERNAL_IP	IP-Adresse der Anwendung
OPENSHIFT_INTERNAL_PORT	Port auf dem die Anwendung lauscht
OPENSHIFT_X_DB_HOST	Host auf dem Datenbank X läuft
OPENSHIFT_X_DB_PORT	Port für Verbindungen zu Datenbank X
OPENSHIFT_X_DB_USERNAME	Benutzername für den Datenbankzugriff
OPENSHIFT_X_DB_PASSWORD	Passwort um auf die Datenbank zuzugreifen

Tabelle 1: Übersicht wichtiger durch OpenShift gesetzter Umgebungsvariablen [17, S. 59].

Datenbankschemas kann entweder manuell durch ssh-Zugriff auf die Anwendungsinstanz geschehen oder automatisiert durch entsprechende Deployment-Hooks, die zum Beispiel bei jeder Aktualisierung der Anwendung ausgeführt werden. Diese Hooks werden im nächsten Abschnitt näher erläutert.

3.3 Deployment

Das Deployment, also das Kopieren und Starten der Anwendung auf dem Server, erfolgt unter OpenShift mittels git [4]. Dies vereinfacht die Integration in den Workflow der Entwicklung, ermöglicht eine Versionierung und Versionsverwaltung der ausgelieferten Anwendung und erspart die Einarbeitung in spezielle Deploymentwerkzeuge.

Beim Erstellen einer Anwendung wird automatisch ein git-Repository gleichen Namens angelegt und das git-Verzeichnis innerhalb des Gears in dem die Anwendung läuft als remote hinzugefügt. Die deployte Anwendung wird aus diesem git-Verzeichnis immer dann gepullt wenn ein push in das remote-Repository erfolgt. Abbildung 4 zeigt diesen Ablauf schematisch.

Da das Beenden und Neustarten einer Anwendung beim Deployment längere Zeit dauern kann und somit störende Ausfallzeiten dieser bedeutet, unterstützt OpenShift für die meisten Anwendungsserver auch hot Deployment. Ist die Markerdatei `.openshift/markers/hot_deploy` vorhanden, erfolgt das Deployment ohne Neustart der Anwendung [17, S.40f].

Große Anwendungen können teilweise aufgrund begrenzter Ressourcen oder zu starker Beeinträchtigung

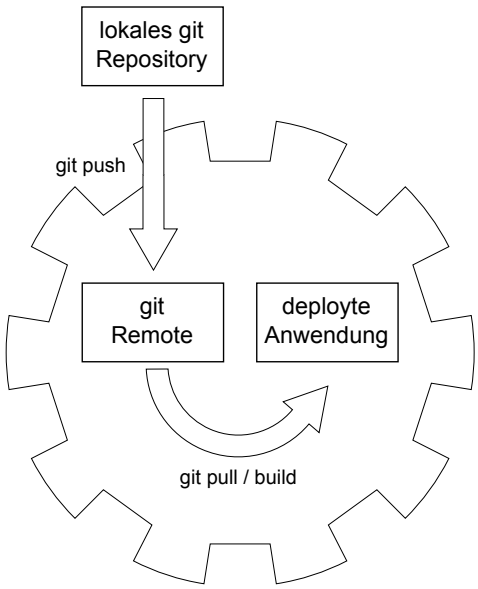

Abbildung 4: Ablauf des Deployments einer OpenShift Anwendung.

der laufenden Instanz nicht innerhalb des Anwendungsgears gebaut werden. OpenShift ermöglicht aus diesem Grund die Nutzung eines Jenkins-Buildservices, der in einem eigenen Gear läuft, bei jedem `git push` die Anwendung neu baut, und das Deployment startet, sollte der Bau erfolgreich verlaufen sein. Nach 15 Minuten Inaktivität wird der Buildservice wieder gestoppt und das Gear wieder freigegeben [17, S.45].

Um eine genaue Kontrolle über den Start, eventuelle Migration von Daten, das Anlegen der Datenbankschemata oder Aufräumarbeiten beim Beenden der Anwendung zu erhalten stehen mehrere Hooks zum Ausführen von Skripten bereit. Diese Skripte sind nicht auf bestimmte Sprachen festgelegt; es können neben compilierten Programmen auch fast alle gängigen Skriptsprachen wie bash, perl, lua, ruby und python verwendet werden. Eine Auflistung möglicher Hooks liefert Tabelle 2.

Name des Skripts	Beschreibung
pre_build	Die alte Version der Anwendung wurde gestoppt, die neue Version wurde gepusht aber noch nicht gebaut.
build	Die Anwendung wurde gebaut, aber noch nicht deployed.
deploy	Die Anwendung wurde deployed aber noch nicht gestartet.
post_deploy	Ausgeführt direkt nachdem die neue Version der Anwendung gestartet wurde.

Tabelle 2: Übersicht nutzbarer Hooks in OpenShift.

Zusätzlich bietet OpenShift auch einen direkten ssh-Zugriff auf die Anwendung an, sodass das testen kleinerer Konfigurationsänderungen, der Zugriff auf Logs und eventuelle Debuggingarbeiten schnell und ohne Änderungen an den Deploymentskripten erfolgen kann.

3.4 Skalieren von Anwendungen/Limitierungen

OpenShift führt Anwendungen in sogenannten Gears aus. Ein kleines Gear verfügt über 512 MB Arbeitsspeicher, sowie über 1 GB persistenten Speicher. Anwendungen können entweder in nur einem Gear laufen oder über mehrere Gears skaliert werden. Ob eine Anwendung skaliert muss beim Erstellen angegeben werden und lässt sich danach nicht mehr ändern. Eine skalierende Anwendung erhält dabei automatisch HAProxy als Cartridge zum Monitoring und load-Balancing. HAProxy belegt dabei immer ein eigenes Gear, sodass eine skalierende Anwendung immer mindestens zwei Gears belegt. Zusätzlich spielt die gewählte Plattform eine wichtige Rolle, ob sich eine Anwendung skalieren lässt. Grundsätzlich lassen sich Anwendungen die persistent Daten im Dateisystem speichern oder einen globalen Zustand besitzen nicht skalieren [17, S.28].

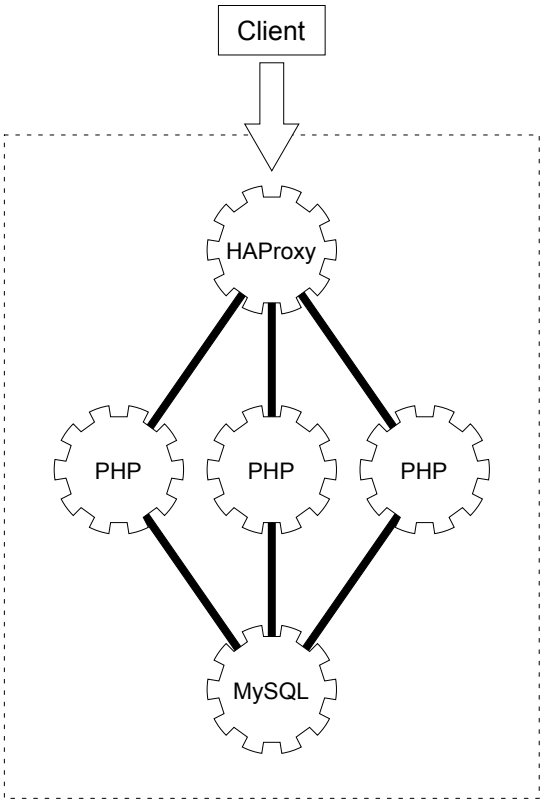

Abbildung 5: Aufbau einer skalierenden OpenShift Anwendung mit fünf Gears.

Anwendungen lassen sich in zwei Richtungen skalieren: es können sowohl einzelne Cartridges in seperaten Gears laufen, als auch neue Anwendungsinstanzen erzeugt werden, sodass mehrere Gears Anfragen entgegen nehmen. Für eine optimale Verteilung der Anfragen wird dabei automatisch ein Load-Balancer vorgeschaltet. Falls ein Kunde über freie Gears verfügt, die der Anwendung bereitgestellt wurden und die Last einen Grenzwert von 90% überschreitet wird

automatisch eine neue Anwendungsinstanz erstellt und gestartet. Fällt die Last einer Instanz über längere Zeit wieder unter 50%, werden die Gears wieder freigegeben [15]. In Zukunft sollen sich diese Grenzwerte auch konfigurieren lassen um Kosten zu sparen oder die Leistung zu erhöhen. Derzeit lässt sich die automatische Skalierung nur beeinflussen, indem man die maximale und minimale Anzahl der zu nutzenden Gears beschränkt.

Eine weitere Limitierung besteht darin, dass OpenShift nur Verbindungen an Port 80 (HTTP), 443 (HTTPS) und 22 (SSH) akzeptiert. Zusätzlich werden nur HTTP/HTTPS durch den vorgeschalteten HTTP-Proxy weitergeleitet. Eine Nutzung anderer Protokolle ist deshalb nur durch Tunneling möglich.

4 Technische Grundlagen

Um die Skalierbarkeit und Verfügbarkeit der via OpenShift angebotenen Anwendungen zu gewährleisten sind bei der Implementierung von OpenShift selbst einige Probleme zu lösen: Die Plattform selbst muss leicht skalier- und erweiterbar sein und sich verteilt auch in weit entfernten Rechenzentren betreiben lassen um nicht selbst zum Flaschenhals für die Anwendungsperformance zu werden.

Da bei einem stark verteilten System die Ausfallwahrscheinlichkeit einzelner Knoten groß ist, darf der Betrieb der Plattform auch nicht durch den Ausfall einzelner Systeme beeinträchtigt werden; beziehungsweise müssen diese sich im Betrieb neu starten lassen und während dieser Zeit die Arbeit durch redundante Systeme übernommen werden.

4.1 Anforderungen an Hosts

OpenShift ist zur Nutzung mit RHEL ausgelegt, kann aber auch mit freien Klonen wie Scientific Linux oder CentOS sowie Fedora verwendet werden. Daher kann OpenShift Origin seinerseits populäre IaaS Dienste wie Amazon EC2 oder Openstack als Ausführungsumgebung nutzen. Des Weiteren nutzt OpenShift eine Vielzahl weiterer freier Projekte wie Bind, Apache ActiveMQ und MongoDB [11, 3.1].

4.2 Allgemeine Architektur und Redundanz

OpenShift besteht aus zwei verschiedenen Hosttypen: Nodes und Broker. Nodes dienen der Ausführung mehrerer Gears, Broker übernehmen die Steuerung und Management der Anwendungen. Zur internen Kommunikation zwischen Nodes und Broker wird das auf Apache ActiveMQ aufsetzende MCollective verwendet. Bei der Implementierung sämtlicher Komponenten stand Redundanz im Vordergrund [11, 2.1], das heißt Broker können redundant verwendet werden, da sie zustandslos sind und auch MCollective kann mehrere ActiveMQ Instanzen verwenden. Da die Broker MongoDB als Datenbank nutzen, kann auch diese mittels Replica-Sets redundant verwendet werden. Abbildung 6 verdeutlicht den Aufbau einer OpenShift Installation mit nur einem Broker und mehreren Nodes.

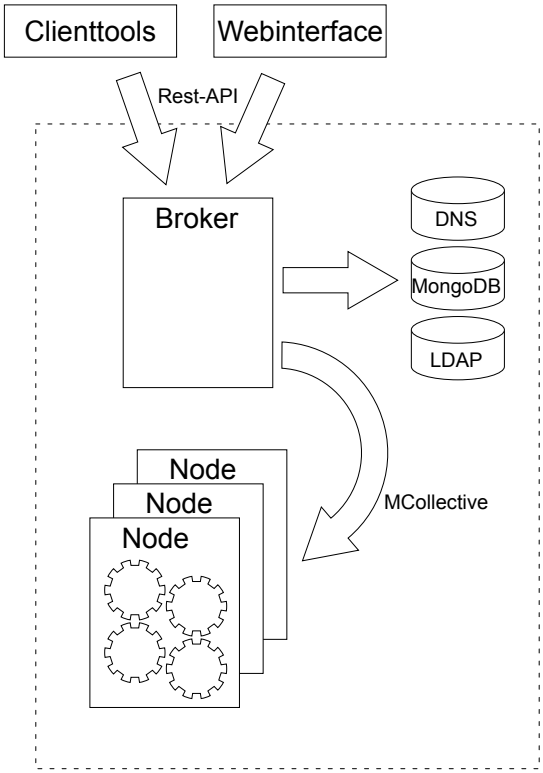

Abbildung 6: Schematischer Aufbau einer OpenShift Installation.

4.3 Nodes

Die mittels OpenShift gehosteten Anwendungen laufen auf Nodes. Diese werden zur einfachen Verwaltung der Ressourcen in logische Gears aufgeteilt. Ein einzelner Node kann dabei eine Vielzahl Gears ausführen um eine möglichst hohe Auslastung zu erreichen. Zur fairen Nutzung gemeinsamer Ressourcen zwischen Gears nutzt OpenShift Linux Kernel cgroups, mit denen sich für Prozessgruppen CPU-Zeit, Arbeitsspeicher inklusive Dateisystemcaches und Netzwerkbandbreite limitieren lassen. Disk Quotas stellen zusätzlich sicher, dass ein Gear nicht zu viel Speicher belegt. Mittels SELinux werden zusätzlich enge Richtlinien gesetzt um Gears vollständig voneinander abzuschotten [11, 3.5]. Zusammen sorgen all diese Maßnahmen dafür, dass auch ohne Ressourcenintensive Maßnahmen wie eine vollständige Virtualisierung mehrere Gears unabhängig auf einem Node laufen können.

4.4 Broker

Der zweite Hosttyp einer OpenShift installation sind Broker. Diese verwalten die Nodes, erzeugen Anwendungsinstanzen, allokieren weitere Gears bei zusätzlichem Ressourcenbedarf und geben diese wieder frei,

falls kein weiterer Bedarf besteht oder die Anwendung gestoppt wird.

Des Weiteren kümmert sich der Broker um den kompletten Zustand der Anwendungen: DNS-Einträge werden durch Bind verwaltet, Benutzerauthentifizierung erfolgt durch LDAP oder Kerberos, der restliche Zustand in einer MongoDB gehalten.

Die Hauptarbeit übernimmt dabei eine auf dem Broker laufende Rails-Anwendung, die über eine Rest-API angesprochen werden kann und andererseits via MCollective mit den Nodes kommunizieren kann.

Bei skalierbaren Anwendungen kann die jeweils laufende HAProxy Cartridge über den Broker mittels der Rest-API die Allokation weiterer Gears veranlassen falls dem Nutzer weitere Gears zur Verfügung stehen oder umgekehrt bei geringer Last dies an den Broker melden um eine Reduzierung der parallelen Instanzen zu erreichen.

4.5 Rest API

Der Broker selbst bietet keine direkte Nutzerschnittstelle, sondern nur eine Rest-API über die er sich ansprechen lässt [16]. Diese wird beispielsweise durch die Weboberfläche und die Clienttools implementiert. Die Weboberfläche ist dabei eine einfache Webanwendung, die beispielsweise selbst wieder mittels OpenShift gehostet werden kann. Die Clienttools sind eine Sammlung in Ruby geschriebener Kommandozeilenprogramme, die die API menschenlesbar verfügbar machen.

Die OpenShift API hält sich an die „Hypermedia as the engine of application state“ (HATEOAS) Prinzip [2]; das heißt nur ein einziger Zugangspunkt der API muss dem Client bekannt sein, weitere Zugangspunkte, Methoden und Parameter verschickt der Server daraufhin als Antwort an den Client.

Beispielhaft könnte das Erstellen einer neuen Anwendung in folgenden Schritten ablaufen: Der Client sendet mittels Basic Authentication seinen Nutzernamen und Passwort an den API-Zugangspunkt https://openshift.redhat.com/broker/rest/api; die Antwort des Servers enthält unter anderem den Status, die unterstützten API-Versionen und weitere Zugangspunkte zur API. Der Antwort kann der Link zum Erstellen einer neuen Nutzer-Domain entnommen werden; mittels Post wird der Name an https://openshift.redhat.com/broker/rest/domains gesendet. Nun kann wieder mittels Post eine neue Anwendung erstellt werden: https://openshift.redhat.com/broker/rest/domains/[Domain_ID]/applications, wobei unter anderem der Anwendungsname und die Basis-Cartridge angegeben werden müssen. Im letzten Schritt kann mittels Post noch ein Startevent an https://openshift.redhat.com/broker/rest/domains/[Domain_ID]/applications/[App_Name]/events gesendet werden um die Anwendung zu starten.

4.6 Kommunikation

Neben der Rest-API zur Kommunikation mit den Clienttools benötigt ein Broker eine Möglichkeit um mit den Nodes zu Kommunizieren. OpenShift nutzt Marionette Collective (MCollective) um einerseits Broadcasts an bestimmten Filterregeln entsprechenden, oder alle Nodes versenden zu können und andererseits einen RPC-Mechanismus zu implementieren um Anwendungen zu starten, stoppen, skalieren oder auf andere Nodes zu verlagern.

4.7 DNS

Zur Vergabe dynamischer Hostnames für die einzelnen Anwendungen vergibt OpenShift Subdomains nach dem Namensschema [App_Name]-[Suffix]. Suffix ist dabei ein vom Nutzer gewählter Namensraum, den alle Anwendungen eines Nutzers gemein haben. Zusätzlich kann vom Benutzer auch ein alias vergeben werden, falls die Anwendung unter anderem Domainnamen laufen soll.

Derzeit kann als DNS-Server einer OpenShift-Installation nur Bind verwendet werden; da das für DNS updates zuständige Plugin aber austauschbar ist, können theoretisch auch alternative DNS-Server verwendet werden, wenn ein entsprechendes Plugin dafür geschrieben wird [11, 3.4.2].

5 Zusammenfassung

OpenShift ist ein noch recht junges Projekt, das noch aktiv weiterentwickelt wird und sich ständig verändert. Dennoch bietet die Plattform schon heute eine große Zahl möglicher Anwendungsszenarien durch die Vielzahl unterstützter Anwendungsserver und Sprachen. Die Möglichkeit OpenShift nicht als durch Red Hat angebotenen Dienst zu nutzen, sondern auch selbst zu hosten gepaart mit dem streng auf Skalierbarkeit und Redundanz ausgelegten Design machen den Dienst sowohl für kleine Startups oder mittelständische Firmen mit wenigen eigenen Servern, als auch für Betreiber ganzer Rechenzentren interessant. In wie weit sich diese Offenheit gegen viel größere Wettbewerber mit starkem Vendor Lock-in durchsetzen kann lässt sich derzeit jedoch noch nicht absehen.

Literatur

[1] Amazon Deutschland. AWS Elastic Beanstalk (Beta). [Online]. Available: http://aws.amazon.com/de/elasticbeanstalk/ [Accessed: 2013-05-26]

[2] T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” Ph.D. dissertation, University of California, Irvine, 2000.

[3] A. Foundation. (2004) Apache License, Version 2.0. [Online]. Available: http://www.apache.org/licenses/LICENSE-2.0.html [Accessed: 2013-05-26]

[4] git. Git. [Online]. Available: http://git-scm.com/ [Accessed: 2013-05-26]

[5] Google Developers. Google App Engine. [Online]. Available: https://developers.google.com/appengine/ [Accessed: 2013-05-26]

[6] J. Guerrero. Announcing OpenShift Enterprise – an On-Premise Platform as a Service from Red Hat. [Online]. Available: https://www.openshift.com/blogs/announcing-openshift-enterprise-paas-from-redhat [Accessed: 2013-05-26]
[7] M. Huber, "Cloud Küche – Red Hat Open Shift: Plattform für Webentwickler und Hoster," *Linux Magazin*, 10 2011.
[8] Microsoft. Cloud Services on Windows Azure. [Online]. Available: http://www.windowsazure.com/en-us/home/scenarios/cloud-services/ [Accessed: 2013-05-26]
[9] openshift. OpenShift Origin – GitHub. [Online]. Available: https://github.com/openshift [Accessed: 2013-05-26]
[10] OpenShift Wiki. Architecture Overview. [Online]. Available: https://www.openshift.com/wiki/architecture-overview [Accessed: 2013-05-26]
[11] ——. Build your own PaaS on RHEL 6. [Online]. Available: https://www.openshift.com/wiki/build-your-own [Accessed: 2013-05-26]
[12] ——. Introduction to Cartridge Building. [Online]. Available: https://www.openshift.com/wiki/introduction-to-cartridge-building [Accessed: 2013-05-26]
[13] T. G. Peter Mell, *The NIST Definition of Cloud Computing*, Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899-8930, 2011.
[14] Red Hat. OpenShift Plattform as a Service. [Online]. Available: https://www.openshift.com/paas [Accessed: 2013-05-26]
[15] Red Hat Inc. Scale your applications on the web. [Online]. Available: https://www.openshift.com/developers/scaling [Accessed: 2013-05-26]
[16] ——, *OpenShift – REST API Guide*, 1801 Varsity Drive Raleigh, NC 27 606-207 2 USA, 2012.
[17] ——, *OpenShift All Versions User Guide*, 1801 Varsity Drive Raleigh, NC 27 606-207 2 USA, 2012.
[18] E. D. Schabell, *OpenShift Primer – Get your Code into the Cloud*. Developer.Press, 2012.