Gut gerüstet

SCA, OSGi und die Service Cloud

Nicole Wengatz, Oliver Arafat

Cloud Computing ist derzeit in aller Munde. Die Kernidee ist, dass sich Anwendungen und Daten nicht mehr auf dem lokalen Rechner oder im Firmenrechenzentrum befinden, sondern auf der virtualisierten IT-Infrastruktur eines Dritt-Anbieters, wie beispielsweise Amazon oder Microsoft. Abgerechnet wird nach Gebrauch, ein Ziel ist die Senkung von IT-Kosten. Der vorliegende Artikel erklärt, wie SCA und OSGi eingesetzt werden können, um von den Vorteilen von Cloud Computing zu profitieren.

Einführung

Die Service Component Architecture (SCA) und die OSGi-Technologie haben den Java-Enterprise-Bereich erobert. Gängige Applikationsserver wie der GlassFish basieren auf OSGi und SCA wird der neue Standard zur Entwicklung von Applikationen auf Basis von serviceorientierten Architekturen werden.

Nach einem kurzen Überblick über SCA und OSGi werden die Herausforderungen beim Einsatz dieser Technologien in Kombination mit einer Cloud-Plattform vorgestellt. Anschließend werden mögliche Lösungen diskutiert. Auf einige Worte zum aktuellen Status folgt das Fazit.

Kurzer Überblick

Die Service Component Architecture (SCA) ist eine Menge von Spezifikationen und beschreibt ein Modell zum Erstellen von Applikationen, die eine serviceorientierte Architektur verwenden. Die OSGi-Technologie stellt eine serviceorientierte, komponentenbasierte Ablaufumgebung zur Verfügung. In dem Artikel [Weng07] wurde beschrieben, wie gut sich SCA und OSGi technologisch ergänzen und welche Vorteile der Einsatz von OSGi als Ablaufumgebung für einen SCA-Container bietet.

Abbildung 1 zeigt eine typische serviceorientierte Umgebung mit Geschäftslogik, die in Java und C++ erstellt ist. Die linke Seite zeigt einen SCA-Container mit Java-Implementierungstyp, der die Java-Komponenten hosted. In OSGi werden Services in sogenannten Bundles zur Verfügung gestellt. Ein Bundle ist eine Java-Archiv-Datei (jar) mit einer zusätzlichen Manifest-Datei, welche die Konfiguration des Bundles beschreibt. Der SCA-Container selbst wird als Menge von Bundles implementiert. Komponente A wird über JMS und SOAP als Service angeboten.

Auf der rechten Seite von Abbildung 1 wird ein SCA-Container mit C++-Implementierungstyp dargestellt, der die C++-Komponenten hosted. Als Ablaufumgebung bietet sich hier PocoCapsule [PocoCapsule], ein Inversion of Control (IoC)-Container, an. Komponente C wird über SOAP als Service angeboten.

Sowohl Komponente A als auch Komponente C verwenden eine Datenzugriffskomponente für den Zugriff auf eine Datenbank. Die SCA-Container verwenden eine verteilte Service-Registry, um geeignete Services zu finden.

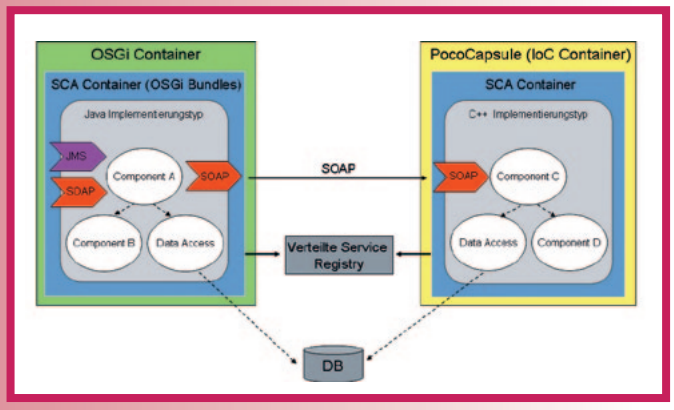

Abb. 1: Heterogene serviceorientierte Landschaft mit SCA

Herausforderungen beim Einsatz mit einer Cloud-Plattform

Services werden den Benutzern oft über Service-Marktplätze angeboten. Um IT-Kosten zu sparen, kann der Provider als Basis für den Marktplatz eine Cloud-Plattform einsetzen. Daraus ergeben sich die folgenden Herausforderungen, die in Abbildung 2 dargestellt sind:

❶ Für den Service-Nutzer ist es unerheblich, ob der Marktplatz technisch auf einer Cloud-Plattform oder einer klassischen Service-Ablaufumgebung basiert.

❷ Sollte der Marktplatz-Provider von einer Cloud-Plattform zu einer anderen wechseln, zum Beispiel aufgrund besserer Konditionen, so sollte dies keine Änderungen im Quellcode der angebotenen Services A und C erfordern.

❸ Die von der Cloud-Plattform angebotenen Services, wie zum Beispiel Amazon S3 (Simple Storage Service) oder Amazon SimpleDB [AWS], sollten ohne Gefahr des Vendor-Lock-Ins, also ohne Gefahr, dass eine Wechsel des Herstellers zu hohen Kosten führt, genutzt werden können.

Lösungen

Typischerweise setzt sich eine Cloud aus drei aufeinander aufbauenden logischen Schichten zusammen:

- Unten angesiedelt ist die Infrastructure as a Service (*IaaS*)-Schicht. Darunter versteht man das Anbieten von Rechen-

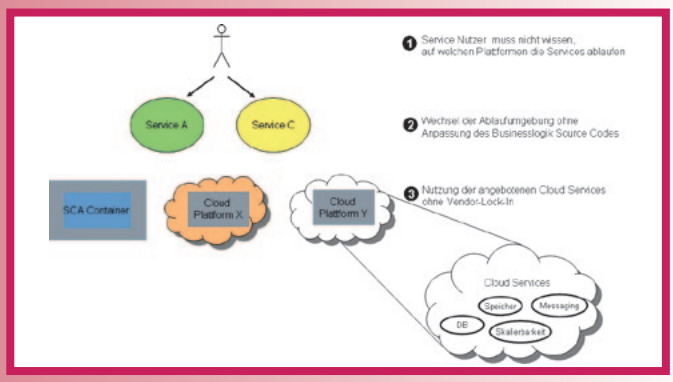

Abb. 2: Herausforderungen

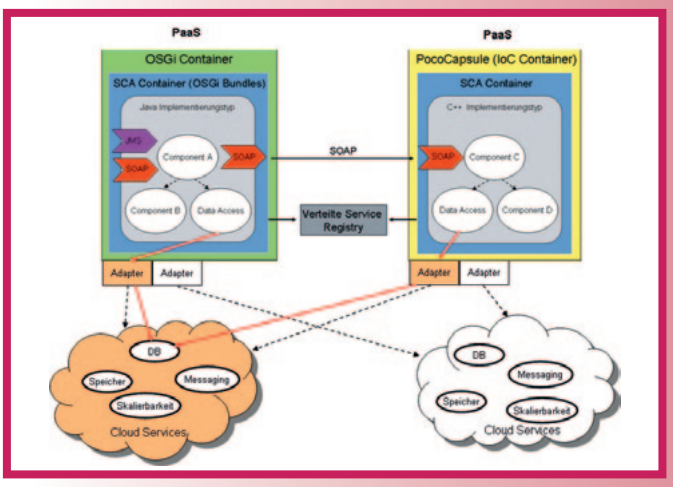

Abb. 3: SCA-Container als PaaS

und Speicherressourcen als On-Demand-Service. Diese Schicht beinhaltet entsprechende Schnittstellen für die Konfiguration dieser Ressourcen und das Deployment von Images für den Betrieb der virtuellen Maschinen.

- Darauf aufbauend setzt *PaaS* (Platform as a Service) auf. Unter PaaS versteht man den Ansatz, eine integrierte Laufzeitumgebung als Dienst zur Verfügung zu stellen, für den der Nutzer auf Anforderung zahlen muss.
- Wiederum darauf basierend befindet sich die Software as a Service (*SaaS*)-Schicht. Auf dieser Schicht laufen sowohl die Services des Cloud-Anbieters (wie z. B. ein Billing-Service) als auch die Services des Applikationsanbieters.

Daraus ergeben sich vier unterschiedliche Anbieter-Rollen, die jedoch nicht zwingend von verschiedenen Personen eingenommen werden müssen:

- Der Cloud-Anbieter stellt die technische Infrastruktur, wie Rechenkapazität und Speicher, durch Service-Schnittstellen bereit.
- Der Cloud-Plattform-Anbieter wiederum bietet darauf aufbauend eine integrierte Ablaufumgebung für Services bzw. Applikationen an.
- Cloud-Services, die von in der Cloud laufenden Applikationen genutzt werden können, werden von einem Cloud-Service-Provider angeboten.
- Am Ende der Kette steht der Applikations-Service-Anbieter. Diese Services werden folgerichtig von Applikations-Service-Konsumenten genutzt.

Die zentrale Lösungsidee ist, dass die beiden SCA-Container als PaaS zur Verfügung gestellt werden. Die Services A und C können damit unverändert in der Cloud verwendet werden. Für den Service-Nutzer macht es keinen Unterschied, wo die Services laufen, da diese nach wie vor über SOAP beziehungsweise über JMS aufrufbar sind und über die verteilte Service-Registry gefunden werden.

Die von der Cloud angebotenen Services werden über Adapter genutzt. Die Datenzugriffskomponente greift zum Beispiel nicht mehr direkt auf eine Datenbank zu, sondern verwendet den Adapter, um auf einen angebotenen Cloud-Datenbank-Service, wie zum Beispiel Amazon SimpleDB, zuzugreifen.

Durch das Adapterkonzept wird erreicht, dass der PaaS mit verschiedenen Cloud-Plattformen zusammenarbeitet, sodass Vendor-Lock-In vermieden wird. Die Geschäftslogik in den Komponenten A und C muss nicht angepasst werden, da die Datenzugriffslogik in der Datenzugriffskomponente gekapselt ist und über das SCA-Prinzip der Dependency Injection (DI) den Komponenten zur Verfügung gestellt wird.

Durch diesen Lösungsansatz wird erreicht, dass die notwendige Variabilität für den Zugriff auf verschiedene Cloud-Plattformen nicht in die Geschäftslogik implementiert werden muss, sondern durch die als PaaS angebotenen Ablaufumgebungen realisiert wird. Derzeit müssen die PaaS-Provider für jede unterstützte Cloud-Plattform einen Adapter zur Verfügung stellen, da die Cloud-Plattform-APIs noch nicht standardisiert sind.

Aktueller Status

OSGi- und SCA-Container lassen sich in einer Cloud-Plattform starten. Als Beispiel sei das Starten eines OSGi-Equinox-Containers in einer Amazon EC2 (Elastic Compute Cloud)-Instanz genannt. Amazon Web Services [AWS] ist eine Sammlung verschiedener Webservices, die auf dem Webportal vom Amazon im Internet angeboten werden. Amazon EC2 ist unter dem Dach der Amazon Web Services angesiedelt, der Benutzer kann Rechenleistung mieten. Um eine eigene Anwendung innerhalb der Amazon-Cloud ablaufen lassen zu können, muss ein Image der Anwendung erzeugt werden. Das Image enthält sowohl den OSGi-Container als auch die Applikation.

OSGi- und SCA-Container als PaaS, die mit verschiedenen Cloud-Plattformen interagieren, gibt es heute noch nicht. Die Idee wird aber bereits in verschiedenen Projekten verfolgt, unter anderem bei der Service Cloud [OSGiCloud], im g-Eclipse-Projekt [gEclipse] und im NEXOF RA-Projekt [Nexof].

Bei der *Service Cloud* wird eine auf OSGi und Java basierende Plattform zwischen die Applikationen und die Plattformen der Cloud-Anbieter gelegt. Die Extraschicht kümmert sich um den Zugriff auf die verschiedenen Cloud-Plattformen und entlastet damit den Applikationsentwickler.

g-Eclipse ist ein auf Eclipse und OSGi basierendes Open-Source-Projekt, welches Anwendern und Entwicklern Zugriff auf Grid- und Cloud-Computing-Ressourcen auf einheitliche Weise ermöglicht. Angeboten wird zum Beispiel ein Adapter für den Zugriff auf Amazon EC2 und S3.

NEXOF RA und Reservoir sind strategische Forschungsprojekte im Rahmen von *NESSI* (Networked European Software & Services Initiative). Ziel von Reservoir ist eine serviceorientierte Infrastruktur, die mit verschiedenen Cloud-Plattformen zusammenarbeitet. Insbesondere hat sie die Standardisierung von IaaS zum Ziel. NEXOF RA definiert die Referenzarchitektur für das NESSI-Framework, basierend auf offenen Standards wie SCA und OSGi.

Im Rahmen von NESSI und NEXOF RA wird aktuell auch diskutiert, wie von der Cloud profitiert werden kann, ohne die

Geschäftslogik anpassen zu müssen. Abbildung 4 zeigt die Lösungsidee. Services laufen in dem NEXOF-Container, welcher zukünftig als PaaS zur Verfügung gestellt wird. Der Zugriff auf die Cloud-Services erfolgt über Dependency Injection (DI), sodass Portabilität erreicht wird. Entwickelt wird also gegen standardisierte Interfaces. Durch DI wird hierbei die Entkopplung von der dahinterliegenden Implementierung erreicht.

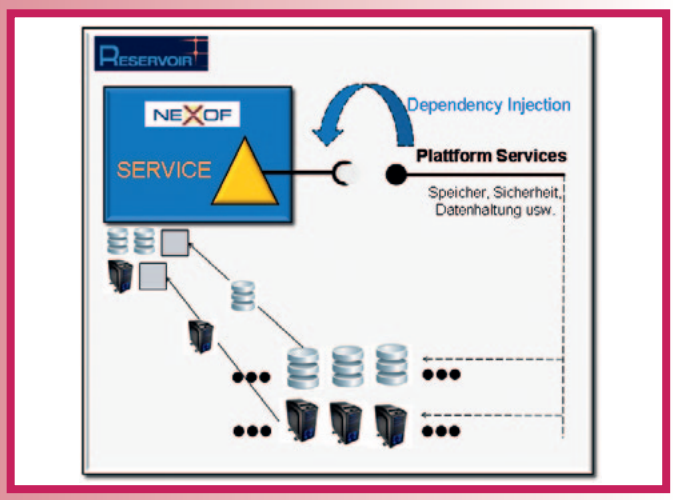

Abb. 4: NEXOF-Container als PaaS

Eigener Protoyp

Da die oben beschriebenen Ansätze zwar alle sehr vielversprechend sind, sich aber noch in einer frühen Entwicklungsphase befinden, haben wir einen Prototyp erstellt. Abbildung 5 zeigt das Szenario.

Der Equinox-OSGi-Container wird in einer Amazon-EC2-Instanz gestartet. Zugriff auf die Amazon SimpleDB erfolgt über SimpleJPA [SimpleJPA], eine Java Persistence API (JPA)-Implementierung für die Amazon-Cloud-Datenbank. Spring wird verwendet, um SimpleJPA als EntityManagerFactory bei der OSGi-Registry zu registrieren. Der Service A bekommt die Referenz über Dependency Injection.

Für einen Ausbau Richtung PaaS würde neben dem Zugriff auf die Datenbank auch der Zugriff auf andere Cloud-Services, wie zum Beispiel Speicher oder Sicherheit, zur Verfügung gestellt. Weiter muss der PaaS-Provider den Zugriff für alle unterstützten Cloud-Plattformen anbieten. Dies ist heute noch sehr arbeitsintensiv, da die angebotenen APIs nicht standardisiert sind. Zu diesem Zweck wurde das Open Cloud Manifesto [OpenCloud] gegründet. Bereits über 300 Firmen und Gruppen haben sich hier zusammengeschlossen, um dafür Sorge zu tragen, dass die Cloud offen bleibt und Vendor-Lock-In möglichst vermieden wird.

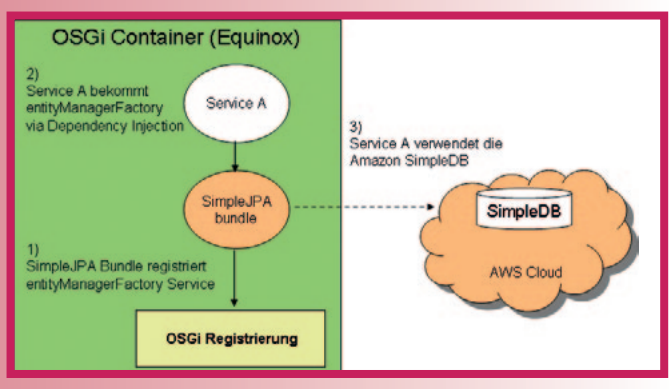

Abb. 5: Prototyp für Zugriff auf die AWS-Cloud

Fazit

OSGi und SCA etablieren sich immer mehr als Technologien der Wahl für serviceorientierte Systeme. Im Zusammenhang mit Cloud nimmt die Bedeutung noch einmal zu.

Viele Cloud-Computing-Provider werben mit günstigen Startangeboten. Die Erfahrung zeigt aber, dass beim Umstieg von einer Cloud-Plattform auf eine andere enormer Anpassungsaufwand zu leisten ist. Grund dafür ist die fehlende Standardisierung für den Zugriff auf die Cloud-Services.

Die Kombination von OSGi und SCA als PaaS bietet hier klare Vorteile. Zum einen wird die Geschäftslogik sauber von der Infrastruktur getrennt und ermöglicht so Portabilität. Zum anderen sind Cloud-Umgebungen üblicherweise sehr dynamisch, sodass Grundprinzipien von OSGi wie das Auflösen und Aktualisieren von Abhängigkeiten zwischen Komponenten sowie Remote Management voll zum Tragen kommen.

OSGi und SCA sind gut gerüstet für die Zukunft, die Erfolgsgeschichte geht weiter.

Links

[AWS] Amazon Web Services, `http://aws.amazon.com/`
[gEclipse] g-Eclipse Project, `http://www.eclipse.org/geclipse/`
[Nexof] NESSI Open Service Framework, Reference Architecture, `http://www.nexof-ra.eu/`
[OpenCloud] Open Cloud Manifesto, `http://www.opencloudmanifesto.org/`
[OSGiCloud] CloudPlatform, Cloud Services Ltd., `http://www.service-cloud.com/node/46`
[PocoCapsule] PocoCapsule/C++ IoC and DSM container, `http://pocomatic.com/docs/whitepapers/pococapsule-cpp`
[SimpleJPA] Java Persistence API for Amazon SimpleDB, `http://code.google.com/p/simplejpa/`
[Weng07] N. Wengatz, SCA, OSGi und Spring, in: JavaSPEKTRUM, 04/2007, `http://www.sigs.de/publications/js/2007/04/wengatz_JS_04_07.pdf`

Nicole Wengatz arbeitet als Senior Softwarearchitektin und Program-Managerin in der Corporate Technology Abteilung der Siemens AG. Sie verfügt über langjährige Java- und OSGi-Erfahrung und vertritt Siemens in der SCA/OSOA Collaboration.
E-Mail: nicole.wengatz@siemens.com.

Oliver Arafat arbeitet als Softwarearchitekt im Bereich Corporate Technology (CT) der Siemens AG. CT berät die Siemensbereiche beim Einsatz innovativer Technologien und Methoden. Oliver Arafat beschäftigt sich schwerpunktmäßig mit serviceorientierten und modularen Architekturen. Er vertritt Siemens im Architecture Board von NEXOF RA.
E-Mail: oliver.arafat@siemens.com.