Bereitstellung von Microservices in der Oracle-Cloud

Stefan Kühnlein, OPITZ CONSULTING Deutschland GmbH

Microservices sind längst nicht mehr nur ein Buzzword, sondern etablieren sich immer mehr in aktuellen IT-Vorhaben. Sie bilden inzwischen die Grundlage flexibler und robuster Software-Architekturen und viele monolithische Anwendungen werden durch eigenständige, fachliche Microservices ersetzt. Insbesondere durch den Einsatz von Container-Technologien ist die Verwaltung und Verteilung von Microservices wesentlich vereinfacht. Container-Technologien wie Docker rücken immer mehr in das Interesse von Entwicklern und IT-Architekten. Mit dem Oracle Application Container Service und dem Oracle Container Cloud Service stellt Oracle zwei Laufzeit-Umgebungen bereit, um Microservices in der Oracle-Cloud zu betreiben.

Frühere monolithische Anwendungen waren in der Regel sowohl für einen speziellen Einsatz als auch für eine spezifische Konfiguration mit fest vordefinierter Hard- und Softwarekonfiguration vorgesehen. Im Zeitalter des Cloud Computing ist die Entwicklung von Anwendungen jedoch wesentlich komplexer geworden: Diese müssen im Vergleich zu den bisherigen monolithischen Anwendungen zahlreichere und unterschiedlichere Hard- und Softwarekonfigurationen unterstützen.

Um eine Anwendung in der Cloud betreiben zu können, werden vor allem neue Anforderungen an Skalierbarkeit, Hochverfügbarkeit und Portabilität gestellt. Hier bietet die Container-Technologie eine Lösung: Mit deren Hilfe können Microservices zuverlässig auf den verschiedensten Umgebungen betrieben werden; zugleich wird die Komplexität in Bezug auf unterschiedliche Konfigurationen reduziert. Eine der bekanntesten Container-Technologien zur Virtualisierung von Anwendungen und Microservices ist Docker, das derzeit als eine der revolutionärsten technologischen Entwicklungen in diesem Bereich gilt.

Das Konzept der Container

Die Bereitstellung von Containern in der Applikationsentwicklung ist an und für sich keine neue Idee. Auf Betriebssystem-Ebene gibt es sie bereits in Form von Server-Klassen. Die Container-Technologie greift im Grunde eine zentrale Aufgabe von Betriebssystemen auf, um Zugriffe auf verschiedene Ressourcen zu ermöglichen und diese auf die laufenden Prozesse zu verteilen. Die neue Technologie führt diese Aufgabe konsequent weiter. Anstelle der gemeinsamen Nutzung von Netzwerken, Dateisystem, CPU und Speicher werden diese durch die Container-Technologie isoliert. Somit wird einem Container eine eingeschränkte Menge an CPU, Speicher und Netzwerk-Konfiguration exklusiv zur Verfügung gestellt.

Docker

Die ursprünglich zugrunde liegende Container-Technologie in Linux erstellte Google auf Basis von Userspace Interface for Linux Kernel Containers (LXC). Im ersten Schritt entwickelte Google einen Client, der auf LCX aufsetzt und mit dem erste Container und Images erstellt und verwaltet werden konnten. Diese Images waren im Wesent-

lichen nichts anderes als Dateisysteme, die sowohl die auszuführenden Prozesse als auch die Konfigurationsdatei für den Start des Prozesses beinhalteten. Bereits damals ging es vor allem darum, mit einem einzigen Befehl ein bestehendes Image zu laden und es zu starten.

Inzwischen sind aus Docker alle LXC-Abhängigkeiten entfernt und durch eine Virtualisierung auf Betriebssystem-Ebene ersetzt. Dieser Ansatz ist im Vergleich zu klassischen Virtualisierungstechniken deutlich leichtgewichtiger, da kein eigener Kernel instanziiert werden muss und somit deutlich weniger Speicher benötigt wird.

Ein Docker-Container enthält die Anwendung sowie alle benötigten Abhängigkeiten, teilt jedoch den Kernel mit anderen Containern *(siehe Abbildung 1)*.

Container werden als isolierte Prozesse im User Space des Betriebssystems ausgeführt, das auf dem Host installiert ist, und sind nicht an eine bestimmte Infrastruktur gebunden. Somit können Container auf jedem Computer, in jeder Infrastruktur-Umgebung und in der Cloud ausgeführt werden.

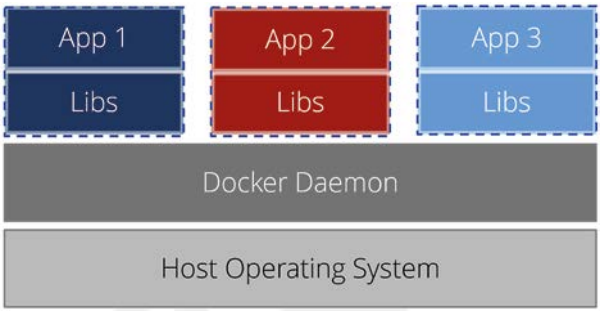

Abbildung 1: Nutzung des gemeinsamen Betriebssystems

Oracle Application Container Cloud Service

Mit dem Application Container Cloud Service (OACCS) stellt Oracle einen leichtgewichtigen PaaS-Service zur Ausführung von Microservices bereit. Dieser beinhaltet eine standardisierte Runtime-Umgebung zur Ausführung von Microservices, die mit unterschiedlichen Programmiersprachen erstellt werden können. Aktuell unterstützt der OACCS Anwendungen beziehungsweise Microservices, die mit Java SE, Node.js oder PHP implementiert werden. In naher Zukunft plant Oracle, noch weitere Programmiersprachen wie Java EE, Ruby, Phyton oder Go zu unterstützen.

Der OACCS kann sowohl als eine Non-Metered Subscription als auch als eine Metered Subscription erworben werden. Für einen ersten Test stellt Oracle eine Trial Subscription bereit, mit der der OACCS über einen Zeitraum von dreißig Tagen getestet werden kann. In allen genannten Varianten ist zusätzlich noch eine Subscription für den Oracle Developer Cloud Service *[1]* enthalten.

Die Besonderheit des OACCS besteht darin, dass ein Microservice im Rahmen des Deployments automatisch innerhalb eines Docker-Containers eingebettet und ausgeführt wird. Die Erstellung des Docker-Containers beziehungsweise Docker-Image ist im OACCS vollständig gekapselt. Somit hat der Entwickler eines Microservice keinerlei Einfluss auf die Erstellung und Konfiguration des Docker-Containers. Im Gegenzug benötigt der Entwickler aber auch keine Kenntnisse in der Erstellung von Docker-Containern. *Abbildung 2* zeigt die Architektur des OACCS.

Damit sich ein Microservice innerhalb des Docker-Containers starten lässt, muss dieser die folgenden Anforderungen erfüllen:

- *Definition des Port*
  Der Port, über den die Rest-Services des Microservice erreichbar sind, muss mit der Umgebungsvariablen „PORT" des OACCS übereinstimmen. Nach der Bereitstellung des Microservice überprüft der OACCS mithilfe eines Pings, ob der Microservice erreichbar ist.

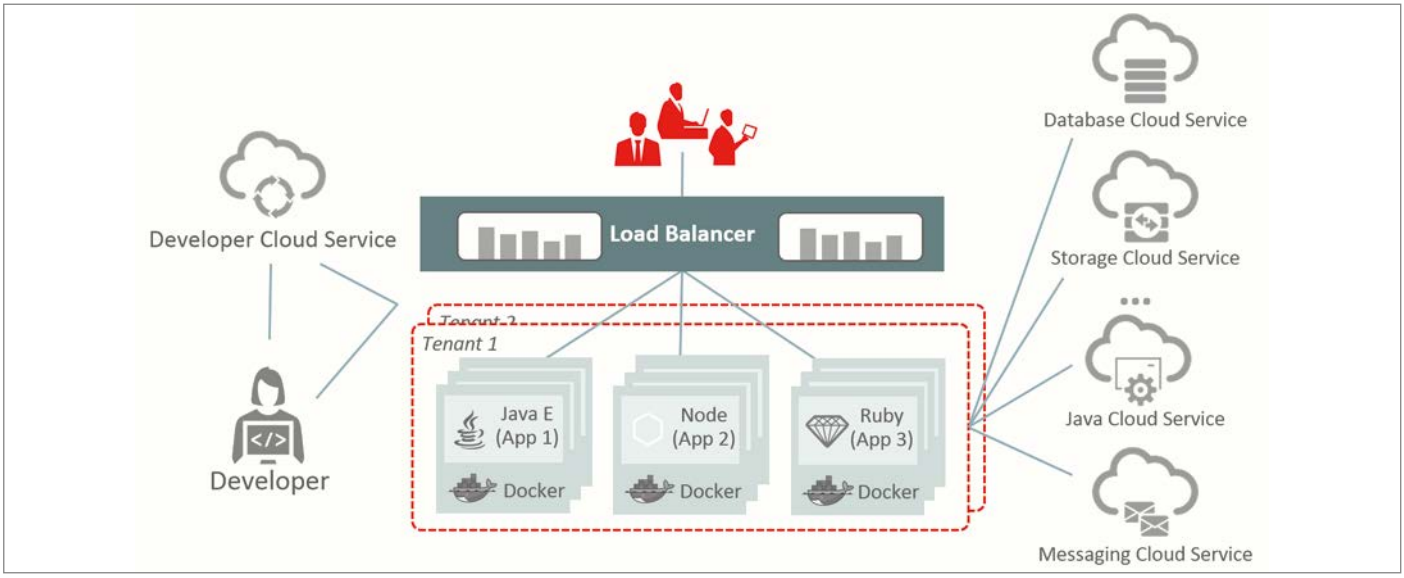

Abbildung 2: Die Architektur des OACCS

```
static{
   protocol = "http://";
   host = Optional.ofNullable(System.getenv("HOSTNAME"));
   port = Optional.ofNullable(System.getenv("PORT"));
   path = "myapp";
   BASE_URI = protocol + host.orElse("localhost") + ":" + port.
orElse("8080") + "/" + path + "/";
}
```

Listing 1: Auswertung der Umgebungsvariablen

```
public static HttpServer startServer {
   final ResourceConfig rc = new ResourceConfig().packages
       ("com.example.rest");
   return GrizzlyHttpServerFactory.createHttpServer
(URI.create(BASE_URI), rc);
}
```

Listing 2: Start und Initialisierung des HTTP-Servers

```
{
   „runtime“:{
       "majorVersion": "8"
   },
   "command": "java -jar myRestService.jar",
   "release": {
       "build": "Build_2017/03_002",
       "commit": "Doag Red Stack",
       "version": "Version_2017/02"
   },
   "notes": "Example REST app"
}
```

Listing 3: „manifest.json“ mit den Metadaten des Microservice

- *Dynamische Konfiguration*
  Der Microservice wird im OACCS innerhalb eines Docker-Containers gestartet; somit wird der Hostname dynamisch erzeugt. Damit der Microservice im OACCS gestartet werden kann, muss dieser vor dem Start die Umgebungsvariablen „HOSTNAME“ und „PORT“ auswerten und beim Start den Microservice entsprechend initialisieren.

- *Standalone*
  Damit ein Microservice innerhalb des Docker-Containers ausgeführt werden kann, müssen alle benötigten Bibliotheken bereitgestellt sein. Im Falle eines Microservice, der mit Java SE erstellt wurde, werden sowohl der Microservice als auch die benötigten Bibliotheken in einem sogenannten „uber JAR“ zur Verfügung gestellt. Alternativ können die Libraries auch separat zum Einsatz kommen und über die Option „-classpath“ beim Start des Microservice referenziert werden.

Listing 1 zeigt die Ermittlung der Umgebungsvariablen „HOSTNAME“ und „PORT“ im Static Initializer der Hauptklasse zur Laufzeit, *Listing 2* den Start und die Initialisierung eines HTTP-Servers unter Verwendung des Frameworks Grizzly mit den aus *Listing 1* ermittelten Umgebungsvariablen.

Für das Deployment eines Microservice im OACCS ist neben der „.jar“-Datei, die den Microservice mit allen notwendigen Libraries enthält, zusätzlich noch eine Manifest-Datei erforderlich. Diese enthält alle für die Ausführung benötigten Metadaten wie Laufzeitumgebung, Start-Kommando, Versions-Informationen und zusätzliche Notizen. *Listing 3* zeigt den Aufbau der Manifest-Datei.

Für das Deployment des Microservice im OACCS müssen sowohl die „.jar“-Datei als auch die „manifest.json“ in einer zip-Datei zusammengefasst sein. Das Deployment des Service erfolgt über die Konsole des OACCS oder über den Developer Cloud Service *[1]*.

Oracle Container Cloud Service

Mit dem Oracle Container Cloud Service (OCCS) stellt Oracle einen weiteren Service in der Cloud bereit, mit dem Microservices innerhalb eines Docker-Containers in der Cloud bereitgestellt und ausgeführt werden können. Während im OACCS die Infrastruktur komplett weggekapselt ist, stellt der OCCS eine vollständig transparente Infrastruktur zur Ausführung und Verwaltung von Docker-Containern bereit. Zusätzlich zur Infrastruktur beinhaltet der OCCS umfangreiche Werkzeuge, um containerbasierte Microservices in der Oracle Cloud zur Verfügung zu stellen, zu orchestrieren und zu verwalten. Analog zum OACCS kann der OCCS sowohl als eine Non-Metered Subscription als auch als eine Metered Subscription erworben werden. Auch dieser Dienst lässt sich mit der Trial Subscription testen.

Instanzen des Container Cloud Service

Für die Bereitstellung von Docker-Containern im OCCS sind zuerst die notwendigen Instanzen zu erzeugen. Eine Instanz des OCCS besteht immer aus einem Manager- und mindestens einem Worker-Knoten. Im OCCS können bis zu 250 Worker-Knoten erzeugt werden. Zusätzlich zur Anzahl der Worker-Knoten muss die Größe des Worker-Knotens feststehen. Ein Worker-Knoten besteht mindestens aus einer OCPU mit 7,5 GB RAM – die maximale Größe beträgt 16 OCPUs mit 120 GB RAM. Auf die Konfiguration des Manager-Knotens kann im Rahmen der Erstellung der OCCS-Instanz kein Einfluss genommen werden.

Sowohl der Manager- als auch die Worker-Knoten werden im OCCS als Oracle Compute Virtual Machines (VMs) abgebildet. Auf die Konfiguration der VMs kann aktuell noch kein Einfluss genommen werden. Mit einem der nächsten Releases soll es eine Möglichkeit geben, zum Beispiel die Kernel-Parameter der VMs zu modifizieren. Dies ist im Falle des Deployments

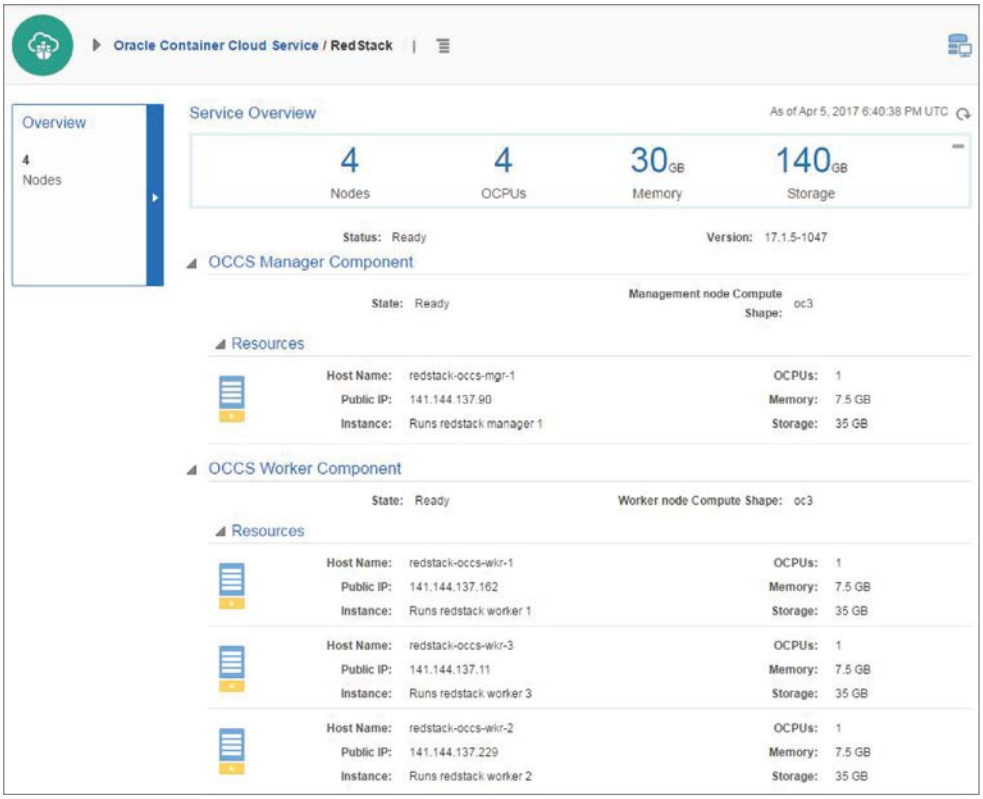

Abbildung 3: Übersicht über die Manager- und Worker-Knoten

einer ElasticSearch-Datenbank in OCCS auch notwendig, da für ihren Betrieb die Standardeinstellung für „mmap" zu niedrig ist *[2]*. *Abbildung 3* zeigt die OCCS-Instanz mit einem Manager-Knoten und drei Worker-Knoten einer Service-Definition.

Oracle Container Service Dashboard

Nach erfolgreicher Provisionierung der OCCS-Instanz kann nun die Service-Konsole beziehungsweise das Dashboard über das Hamburger-Menü gestartet werden. Für den Start der Service-Konsole ist eine Autorisierung mit Benutzername und Passwort des Administrators erforderlich. Beides wurde bei der Initialisierung des Service vergeben. *Abbildung 4* zeigt das Dashboard des OCCS.

Hosts

Wie bereits erwähnt, wird im Rahmen der Definition einer neuen Service-Instanz die Anzahl der Hosts beziehungsweise Work Nodes angegeben. Jeder Host repräsentiert eine eigenständige Oracle Compute Virtual Machine, auf der im Rahmen des Deployments entweder einzelne Container (Services) oder eine Orchestration von Containern (Stacks) bereitgestellt werden.

Zusätzlich wird bei der Initialisierung eines neuen Service zu den Worker-Knoten noch ein Manager-Knoten hinzugefügt. Dieser übernimmt beim Deployment die Verteilung der Services beziehungsweise des Stacks auf die entsprechenden Worker-Nodes und überwacht die Ausführung der Container. Dazu ist auf den Worker-Knoten ein entsprechender Agent installiert. Sollte die Kommunikation zwischen Worker- und Manager-Knoten aufgrund einer Störung für einen Zeitraum von mehr als einer Minute nicht möglich sein, wird der Worker-Knoten automatisch auf „inaktiv" gesetzt.

Resource Pools

In vielen Fällen ist es notwendig, Hosts in logische Einheiten zu gruppieren. Hierzu stellt der OCCS sogenannte „Resource Pools" bereit, um Hosts in isolierten Gruppen zusammenzufassen. Sie vereinfachen erheblich die Verwaltung der Hosts sowie das Deployment der Services und Stacks auf die entsprechenden Hosts. Initial werden bei der Erstellung der Service-Instanz die folgenden drei Resource Pools angelegt:

- Default
- Development
- Production

Die Namen der Pools sind lediglich Vorschläge und können bei Bedarf beliebig umbenannt, erzeugt oder gelöscht werden. *Abbildung 5* zeigt die Aufteilung der Hosts auf die entsprechenden Resource Pools.

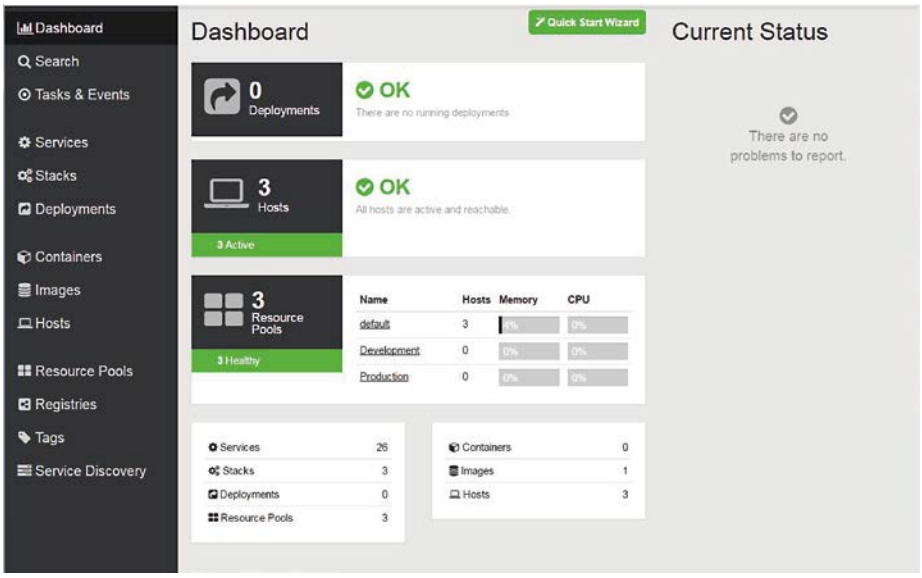

Abbildung 4: Dashboard des Container Cloud Service

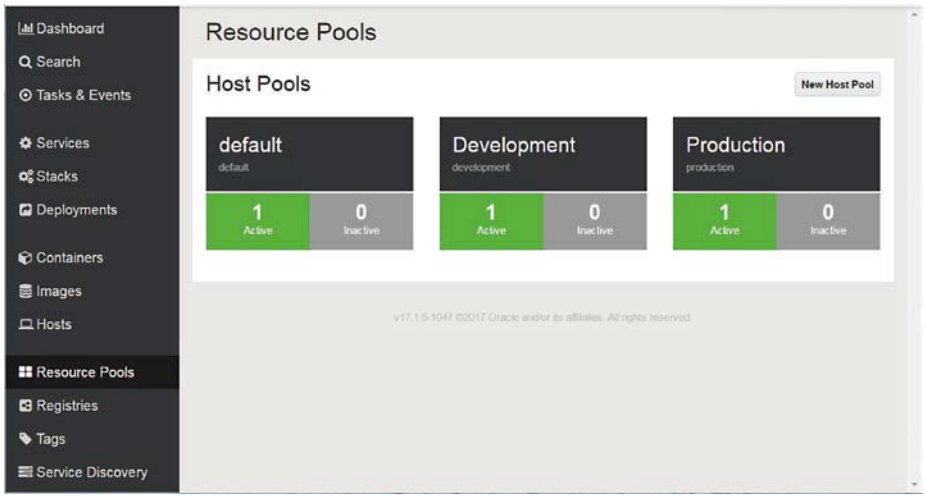

Abbildung 5: Aufteilung der Hosts in Resource Pools

Die Zuordnung der Hosts zu den Resource Pools lässt sich jederzeit ändern. Dabei ist jedoch zu beachten, dass das Verschieben eines Hosts in einen anderen Resource Pool alle laufenden Services beendet. Stehen im ursprünglichen Resource Pool noch freie Kapazitäten zur Verfügung, werden die beendeten Services automatisch auf einem neuen Host innerhalb des ursprünglichen Resource Pool gestartet.

Services

Im OCCS umfasst ein Service alle notwendigen Konfigurationen für die Bereitstellung eines Docker-Image auf einem Host im entsprechenden Resource Pool. Services stellen im OCCS noch nicht den lauffähigen Container dar, sondern sie entsprechen vielmehr einer High-Level-Konfiguration oder einem Template, in dem alle Konfigurationsparameter für die spätere Ausführung gespeichert sind.

Nach der Initialisierung des OCCS stehen eine Reihe vorkonfigurierter Services bereit, die entweder out of the box verwendet werden können oder gemäß spezifischen Anforderungen angepasst werden. Die Liste der Services kann durch die Angabe eines Image und der notwendigen Konfiguration beliebig erweitert werden. *Abbildung 6* zeigt die Konfiguration von NGINX im Service Editor. Für das Anlegen beziehungsweise das Erweitern einer Konfiguration der Services stehen dem Benutzer die folgenden Möglichkeiten zur Verfügung:

- Auswahl von Optionen aus einer Liste
- Einfügen oder Kopieren eines Docker-Befehls
- Einfügen oder Kopieren einer Definition in YAML

Neben dem Erstellen und Anlegen von Services können in diesem Bereich auch nicht mehr benötigte Services gelöscht werden. Allerdings lässt sich ein Service nur dann löschen, wenn dazu kein entsprechender Container läuft.

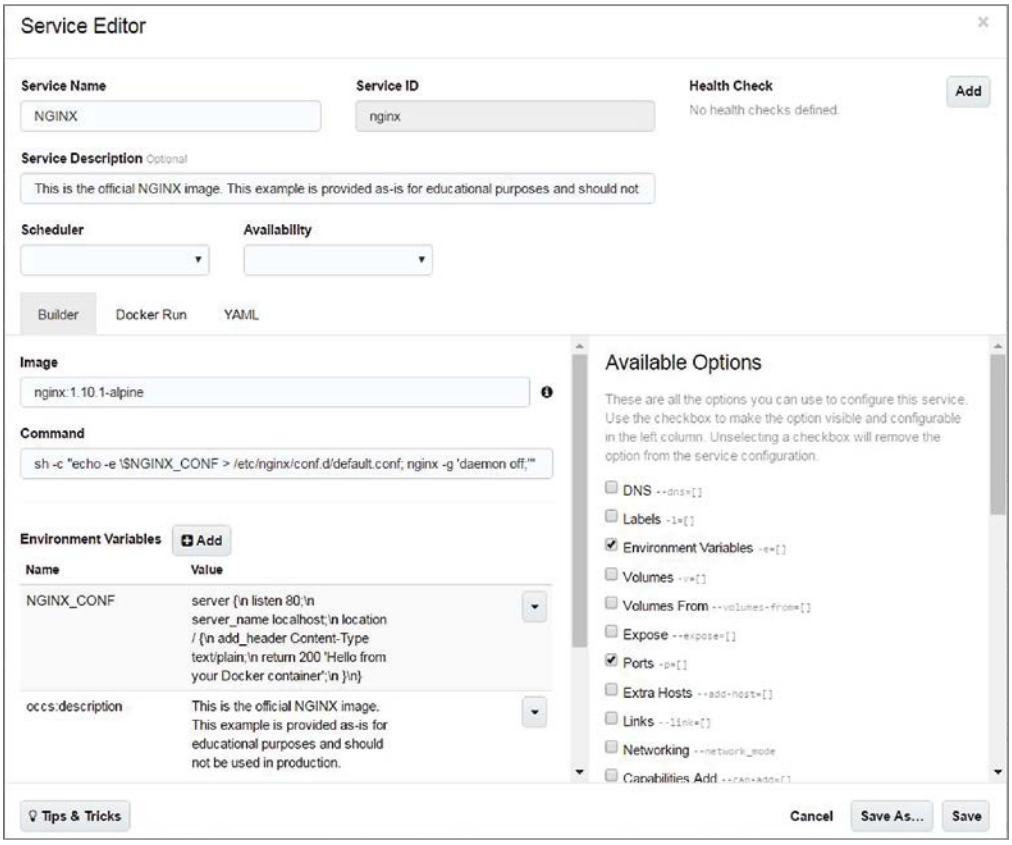

Abbildung 6: Konfiguration von NGINX im Service Editor

Images

Richtet man im OCCS ein Docker-Image zum ersten Mal ein, wird das zugehörige Image aus der entsprechenden Registry geladen und der Liste der geladenen Images hinzugefügt. Im Bereich „Images" des Dashboards des OCCS erfolgt die Verwaltung aller geladenen Images. Docker Images können hier direkt gestartet werden. Allerdings müssen die für die Ausführung notwendigen Parameter in der Konsole erfasst werden. Jedoch wird hierbei kein Service angelegt, sodass die bereits erfasste Konfiguration für die Ausführung des Image nicht überschrieben wird.

Deployments

Im Bereich „Deployment" erfolgt die Bereitstellung, Skalierung und Verwaltung von Services oder Stacks gemäß den definierten Orchestrierungsregeln. Ein Deployment wird im OCCS jedes Mal automatisch beim Start eines Service beziehungsweise eines Stacks erstellt.

Im Rahmen des Deployments eines Service oder Stacks greift die Orchestrierungsfunktionalität, die im OCCS enthalten ist. So werden bei der Bereitstellung ein neues Deployment-Objekt erzeugt und, falls notwendig, die benötigten Images aus der Registry geladen. Im nächsten Schritt erfolgen die Erzeugung der Container sowie bei Bedarf die Einrichtung der Kommunikation der Services untereinander. Anhand der Orchestrierungsregeln erfolgt die Verteilung der benötigten Instanzen eines Containers auf die definierten Hosts in den entsprechenden Resource Pools. *Abbildung 7* zeigt das Deployment eines NGINX-Image im Resource Pool „default".

Docker Registry

Die Speicherung der Docker-Images erfolgt in einer sogenannten „Docker Registry". Diese verwaltet die verschiedenen Versionen eines Docker-Image. Um einen Service in einem Resource Pool einrichten zu können, muss der OCCS das gewünschte Image aus einer Service Registry herunterladen und starten. Für das Laden von Images ist die öffentliche Docker-Hub-Registry bereits vorkonfiguriert. Somit kann der OCCS ohne zusätzlichen Konfigurationsaufwand auf öffentliche Images zugreifen. Für den Zugriff auf Images aus einer anderen öffentlichen oder privaten Docker Registry kann im OCCS die Definition weiterer Registries erfolgen. Im OCCS ist aktuell leider keine Docker Service Registry enthalten, sodass eigene Images nur über die Docker Hub Registry eingerichtet werden können. Alternativ muss eine eigene Service Registry aufgebaut und im OCCS registriert werden.

Fazit

Mit dem Oracle Application Container Cloud Service und dem Oracle Container Cloud Service stellt Oracle zwei unterschiedliche Services für den Betrieb von Microservices in der Cloud zur Verfügung. Der OACCS eignet sich insbesondere für die Bereitstellung von leichtgewichtigen Microservices in einem Docker Container. Wird hingegen eine vollständige Integration zu weiteren Services oder Standard-Containern benötigt, so ist an dieser Stelle der OCCS zu empfehlen.

Verweise

[1] S. Kühnlein: Application Lifecyle Management mit dem Oracle Developer Cloud Service, Red Stack Magazin, Ausgabe 02/2017

[2] *https://www.elastic.co/guide/en/elasticsearch/reference/current/vm-max-map-count.html*

Stefan Kühnlein
stefan.kuehnlein@opitz-consulting.com

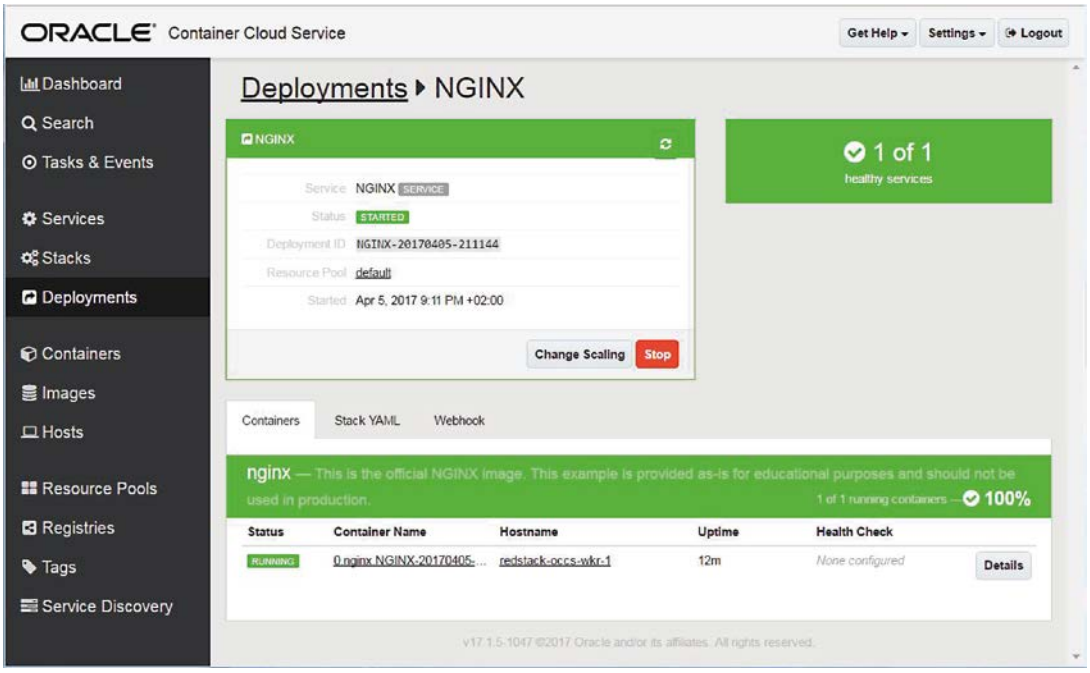

Abbildung 7: Deployment des NGINX-Image