Docker goes AWS

Amazon EC2 Container Service

Martin Eigenbrodt, Phillip Ghadir

Dank des Amazon EC2 Container Service finden Docker-Container nun auch bei den Amazon Web Services (AWS) Unterschlupf. In diesem Artikel stellen wir nach einem kurzen Überblick über die wichtigsten bisher verfügbaren Dienste den neuen Service vor, mit dem Docker-Container in der Cloud gehostet werden können.

Gil C / Shutterstock.com

Amazon-Cloud-Infrastruktur

AWS bieten heutzutage die Plattform für viele Online-Präsenzen. So unterschiedlich Netflix, Expedia, Foursquare, Spotify und Airbnb auch sein mögen, alle nutzen die öffentliche Amazon-Cloud-Infrastruktur.

Amazon betreibt Rechenzentren in Amerika, Australien, Deutschland, Indonesien, Irland, Japan und demnächst in China und bietet Kunden die Wahl der geografischen Region, in der die kundeneigenen Anwendungen, Services und Daten liegen dürfen.

Insbesondere mit der Eröffnung des Rechenzentrums in Frankfurt bietet Amazon nun seine Cloud-Services unter vergleichbaren Bedingungen wie andere Unternehmen mit amerikanischen Wurzeln/Mutterkonzernen an (siehe Kasten „Amazon Datacenter in Frankfurt“).

EC2 – Hardware as a Service

Vermutlich das bekannteste Cloud-Computing-Angebot des Internet-Giganten ist EC2 (Amazon Elastic Compute Cloud), mit dem sich eigene virtuelle Maschinen auf Knopfdruck erzeugen oder zerstören lassen, wobei die Nutzung stundengenau abgerechnet wird.

Die angebotenen virtuellen Maschinen sind für unterschiedliche Zwecke optimiert. Man kann in jeder der folgenden Kategorien aus verschiedenen in Bezug auf Leistungsfähigkeit und Nutzungsgebühr gestaffelten Konfigurationen wählen:

- General Purpose,
- Compute Optimized,
- GPU Instances,
- Memory Optimized,
- Storage Optimized.

Amazon Datacenter in Frankfurt

Amazons CTO Werner Vogels schreibt auf seinem Blog (siehe [Vogels15]):

- *„Customers All Over the World Are Assured that AWS Agreement Meets Rigorous EU Privacy Laws“.*

Weiter schreibt er:

- *„AWS is fully compliant with all applicable EU data protection laws and maintains robust global security standards, such as ISO 27001, SOC 1, 2, 3 and PCI DSS Level 1.“*

EC2-Instanzen starten

Im Wesentlichen instanziiert man bei EC2 sogenannte Amazon Machine Images (kurz AMIs). Solche AMIs sind im Wesentlichen Momentaufnahmen des Festplatteninhalts eines Rechners. Es gibt vorbereitete AMIs für verschiedene Betriebssysteme mit unterschiedlichen Konfigurationen. Es lassen sich aber auch eigene erstellen. Was aus einem AMI instanziiert wird, nennt man üblicherweise EC2 Instance.

Bei Bedarf kann ein AMI nicht nur einfach, sondern gleich mehrfach verteilt in mehreren sogenannten Availability Zones instanziiert werden. Verschiedene Availability Zones sind dabei so voneinander isoliert, dass ein gleichzeitiger Ausfall aufgrund der gleichen Ursache (z. B. Stromausfall oder Feuer) möglichst unwahrscheinlich ist.

Ist noch mehr Redundanz gewünscht, muss das AMI in eine andere Region – also eine andere geografische Zone – kopiert werden und kann dann dort wiederum in einer oder mehreren Availability Zones instanziiert werden.

Beim Instanziieren eines AMIs können bei Bedarf für diese Instanz eine feste IP-Adresse sowie spezifische Firewall-Regeln konfiguriert werden. All dies kann entweder über die Weboberfläche (die sogenannte AWS-Konsole), ein Web-API oder über die Kommandozeilen-Werkzeuge gesteuert werden.

Ephemere EC2-Instanzen

Ephemer bedeutet flüchtig: EC2-Instanzen können zwar temporäre Dateien im lokalen Dateisystem speichern, haben allerdings kein persistentes Dateisystem, das den Neustart einer Instanz übersteht.

Sollen Daten so gespeichert werden, dass sie für mehrere Instanzen (gleichzeitig oder zeitlich versetzt) zur Verfügung stehen, können dafür andere Amazon-Dienste wie zum Beispiel Elastic Block Storage (EBS), Simple Storage Service (S3) oder die Datenbank-Services Relational Database Service (RDS), Simple DB, Dynamo DB sowie der Data-Ware-House-Service Redshift angebunden werden.

Elastisch Skalieren

Nicht nur das nutzungsbezogene Bezahlmodell, sondern auch die Leichtigkeit, mit der EC2-Instanzen hoch- und wieder heruntergefahren werden können, erlauben das situative Anpassen der konfigurierten Instanzen an die aktuelle Last.

Auch das kann Amazon automatisch übernehmen: Dafür definiert man eine sogenannte Autoscaling Group in der – abhängig von Skalierungsregeln, Metriken und Schwellwerten – automatisch Instanzen aus dem gleichen AMI gestartet oder deprovisioniert werden. Dabei kann man auch minimale und maximale Pool-Größe und Wachstumsraten definieren.

Docker-Images bei AWS

Docker-Images können auf EC2-Instanzen einfach verwendet werden, sofern auf dem gewählten AMI Docker läuft. Die Neuerungen von Docker können dabei auch recht nützlich sein (siehe Kasten „Neues in Docker“). Allerdings weiß dann die Amazon-Cloud-Infrastruktur nichts von den laufenden Docker-Containern, die dann selbst verwaltet werden müssen.

Neues in Docker

Seit der Vorstellung von Docker in dieser Kolumne [Ghadir14] ist viel passiert. Zu dem damals vorgestellten Kommandozeilen-Client, mit dem Images installiert, verwaltet und instanziiert werden konnten, sind nun zwei neue Werkzeuge hinzugekommen, die die Nutzung von Docker für die Konfiguration komplexer Systeme vereinfachen, die aus mehreren Teilsystemen bestehen:

- *Docker Compose* (ehemals unter dem Namen fig bekannt) ist eine separat installierbare Erweiterung von Docker, die es erlaubt, eine „Applikation” in einer zusammenhängenden Konfiguration zu definieren, die aus mehreren miteinander verlinkten Docker-Containern bestehen kann. Dabei lassen sich für jeden Bestandteil unterschiedliche Parameter für die Skalierung konfigurieren. Eine solche Applikation kann mit Docker Compose als Ganzes gestartet oder gestoppt werden.
- *Docker Swarm* (Stand März in der Beta) erlaubt das Clustern von Docker-Engines zu einer virtuellen. Swarm steht als Docker-Image zur Verfügung, mit dem ein zentraler Master aufgesetzt wird, über den der Cluster konfiguriert wird und dann nach außen wie ein gewöhnlicher Docker-Daemon aussieht. Das ermöglicht beispielsweise das Starten einer Applikation, die von Docker Compose auf mehrere Maschinen im Cluster verteilt ausgeführt werden kann.

Abhilfe schafft hier der neue Amazon EC2 Container Service (ECS), der Docker-Images und -Container verwalten kann. ECS kann Docker-Images sowohl von dem öffentlichen Docker-Repository (dem Docker Hub) als auch von eigenen abgesicherten Repositories beziehen.

Leider haben Docker und AWS ihre eigene Historie mit jeweils eigenen Begriffen. Im Kontext von Amazon ECS gibt es ein paar Begriffe, die leicht missverstanden werden könnten, da sie auch im Docker- oder im EC2-Kontext belegt sind.

Fangen wir mit den ECS-spezifischen Begriffen und Konzepten an und schlagen von dort aus die Brücke in die Docker-Welt.

ECS-Container-Management

ECS erlaubt das Verwalten von Docker-Containern und das Starten und Stoppen von verteilten Applikationen auf Clustern von EC2-Instanzen. In ECS-Terminologie spricht man von einem Task und meint damit eine Applikation, die potenziell aus mehreren Docker-Containern bestehen kann.

Für jeden Docker-Container können Anforderungen in Bezug auf CPU, RAM oder Verfügbarkeit definiert werden, sodass ECS beim Starten eines Tasks selbstständig eine passende EC2-Instanz aus dem Cluster auswählen kann.

Stand heute kann so ein Task nur als Ganzes auf einer EC2-Instanz ausgeführt werden, das heißt, ECS kann zwar eine Menge von Tasks automatisch im Cluster verteilen, allerdings müssen alle Docker-Container, aus denen ein Task besteht, auf demselben Docker-Host und damit auf derselben EC2-Instanz ausgeführt werden.

Tasks in ECS

Tasks bestehen aus einer oder mehreren Beschreibungen von Docker-Containern – den sogenannten Container-Definitions. Für jeden Container legt man dabei das zugrunde liegende Docker-Image und darüber hinaus eine ganze Reihe Konfigurationsoptionen fest. Die meisten entsprechen dabei Parametern, die von docker run bekannt sind.

So können zum Beispiel CPU- und RAM-Bedarf definiert, aber auch Container verlinkt und die exponierten Ports gesetzt werden. Der im Docker-Container ausgeführte Prozess kann über `entrypoint` oder `command` verändert werden, und es lassen sich Umgebungsvariablen definieren.

Task-Definitionen werden in JSON geschrieben. Das folgende Beispiel aus der ECS-Dokumentation zeigt einige der Möglichkeiten:

```
{
  "family": "hello_world",
  "containerDefinitions": [
    {
      "name": "wordpress",
      "links": [
        "mysql"
      ],
      "image": "wordpress",
      "essential": true,
      "portMappings": [
        {
          "containerPort": 80,
          "hostPort": 80
        }
      ],
      "memory": 500,
      "cpu": 10
    },
    {
      "environment": [
        {
          "name": "MYSQL_ROOT_PASSWORD",
          "value": "password"
        }
      ],
      "name": "mysql",
      "image": "mysql",
      "cpu": 10,
      "memory": 500,
      "essential": true
    }
  ]
}
```

Das Attribut `family` definiert einen Namen für die Task-Definition. Unter `containerDefinitions` finden wir ein Array mit den Container-Definitionen, die den Task ausmachen. In diesem Fall handelt es sich um einen Container mit einer Datenbank („mysql“) und einen Container mit einer „wordpress“-Installation. Das Attribut `links` im Wordpress-Container referenziert den mysql-Container mit seinem Namen.

Wenn der Task „hello_world“ ausgeführt wird, delegiert ECS für jede zugehörige Container-Definition an Docker, das einen entsprechenden Docker-Container erzeugt und die Container (in diesem Fall) so linkt, dass in der Wordpress-Instanz geeignete Einträge in `/etc/hosts` und Umgebungsvariablen auf die Datenbank-Instanz (im Container mysql) verweisen.

Essenzielle Container

Nicht aus dem Docker-Kontext bekannt ist das Boolesche Attribut `essential`. Damit wird ein Container als unverzichtbar für den gesamten Task deklariert. Sobald ein als unverzichtbar deklarierter Container terminiert, wird der gesamte Task gestoppt. Jede Task-Definition muss mindestens einen Container als unverzichtbar deklarieren.

Container, die nicht als unverzichtbar deklariert sind, können zum Beispiel für applikationsspezifische Initialisierungen genutzt werden.

Volumes

Innerhalb einer Container-Definition kann man spezifische Volumes auf Pfade innerhalb des Containers einbinden (hier als Beispiel „MyVolume“):

```
"mountPoints" : [
  {
    "sourceVolume" : "MyVolume"
    "containerPath" : "/home/user/myvolume" }
]
```

Mit Namen referenzierte Volumes müssen in der Task-Definition zuvor erstellt werden. Dazu dient das Attribut `volumes`, welches parallel zu `family` oder `containerDefinitions` steht:

```
"volumes" : [
  {
    "name": "MyVolume",
    "host": {
      "sourcePath": "/path/on/docker/host"
    }
  }
]
```

Ein Volume bietet die Möglichkeit, einen Pfad auf dem Docker-Host mit einem logischen Namen zu verknüpfen. Solche Volumes sind persistent. Die Daten werden auf dem Host gespeichert und überleben eine Container-Instanz. Wird das Attribut „host“ leer gelassen, weist Docker selbst einen Pfad zu (per default unterhalb von `/var/lib/docker/vfs/dir/`) und löscht ihn irgendwann, nachdem kein Container mehr auf ihn referenziert.

Wenn man Daten auch über die Lebenszeit der EC2-Instanz (im ECS-Sprachgebrauch Container Instance genannt) hinaus persistieren möchte, kann man beispielsweise über Elastic Block Storage (EBS) ein Volume in eine Instanz mounten, die dann wiederum als Volume im ECS gemountet wird.

Neben dem expliziten Mounten von Volumes kann man auch pauschal alle Volumes eines anderen Containers einbinden:

```
"volumesFrom" : [
  {
    "sourceContainer": "Name eines anderen Containers"
  }
]
```

Hierbei werden auch die Volumes importiert, die bei der Erstellung des Docker-Images im Dockerfile mit „VOLUME“ definiert wurden.

Registrieren von Task-Definitionen

Liegt eine vollständige Task-Definiton in einer lokalen JSON-Datei vor, kann sie mit dem AWS-Kommandozeilen-Client hochgeladen werden:

```
aws ecs register-task-definition --cli-input-json file://./task.json
```

Lädt man mehrfach Definitionen mit identischem `family`-Attribut hoch, erhält man jeweils eine neue Version der Task-Definition.

```
aws ecs list-task-definitions
```

listet die dem Dienst bekannten Tasks und ihre Revisionen auf.

Cluster

Docker-Container werden auf einem Server mit Docker-Daemon gestartet. Das ist auch bei Amazon ECS nicht anders. Neu ist, dass mehrere solcher Server zu einer logischen Einheit – dem Cluster – verbunden werden können.

Etwas verwirrend ist, dass Amazon die Server mit Docker-Daemon als Container Instance bezeichnet. Es werden also Docker-Container auf Container Instances ausgeführt.

Wenn kein Cluster explizit angelegt wurde, existiert implizit ein Default-Cluster, zu dem neue Container-Instanzen beim Start automatisch hinzugefügt werden.

Einen neuen Docker-Host erstellt man durch das Starten der von Amazon bereitgestellten AMIs. Diese enthalten ein Amazon Linux inklusive einer Docker-Installation und eines ECS-Agenten, der als Schnittstelle zwischen dem ECS-API und dem Docker-Daemon dient. Alternativ kann auch eine Distribution eigener Wahl verwendet werden (z. B. CoreOS); dann muss der ECS-Agent selbst installiert werden.

Damit der ECS-Agent auf der neuen Instanz richtig funktioniert, braucht er die Berechtigung, auf das Amazon ECS API und – um Loadbalancer zu konfigurieren – auf das EC2 API zuzugreifen. Solche Berechtigungen werden typischerweise durch ein Instance Profile verliehen. Soll ein Elastic Loadbalancer verwendet werden oder will man sich per SSH auf den Docker-Hosts einloggen, muss darüber hinaus eine Security Group konfiguriert werden, die diesen Netzwerkverkehr erlaubt.

Glücklicherweise liefert die AWS-Konsole einen Assistenten, mit dem der erste Cluster durch einfaches Durchklicken der Dialoge erstellt werden kann.

Starten eines Tasks

Das Verwalten von Tasks in einem Cluster ist Aufgabe eines Schedulers. Stand März 2015 gibt es in ECS zwei Scheduler:

Service Scheduler und Default Scheduler. Der Service Scheduler wird verwendet, um immer eine bestimmte Zahl von Instanzen eines Tasks aufrecht zu erhalten.

Fallen eine oder mehrere Instanzen aus, startet der Scheduler automatisch neue. Das gilt sowohl bei Beenden des Containers als auch bei Ausfall der Container Instance. Dabei kann auch automatisch ein Loadbalancer mit den Informationen über die jeweils aktuellen Instanzen konfiguriert werden.

Beim Default Scheduler werden eine oder mehrere Instanzen eines Tasks einmalig erstellt. Diese Variante eignet sich zum Ausprobieren oder für einmalige oder periodische Batch-Aufgaben, die nicht permanent laufen müssen.

Darüber hinaus stellt das ECS API auch eine Möglichkeit zur Verfügung, um genau zu bestimmen, auf welcher Container Instance ein Task gestartet werden soll. Diese Schnittstelle ist für die Implementierung eigener Scheduler gedacht, beziehungsweise zum Einbinden existierender Software wie Apache Mesos [Mesos].

Um den interessanteren Service Scheduler zu verwenden, beschreibt man zunächst den Service in einem JSON-Dokument. In diesem wird der zu verwendende Cluster referenziert (ein leerer String steht für den Default), ebenso wie die auszuführende Task-Definition und die Anzahl der gewünschten Task-Instanzen.

Eine einfache Service-Definition sieht so aus:

```
{
  "cluster": "",
  "serviceName": "exampleService",
  "taskDefinition": "hello_world",
  "loadBalancers": [
    {
      "loadBalancerName": "NameEinesExistierendenLB",
      "containerName": "wordpress",
      "containerPort": 80
    }
  ],
  "role": "ecsServiceRole",
  "desiredCount": 4
}
```

Im `loadBalancers`-Abschnitt wird dabei ein existierender Loadbalancer referenziert. Bei diesem werden die „wordpress“-Container-Instanzen aus dem Task registriert. Die mit `role` referenzierte Rolle muss der Container-Instanz erlauben, den Loadbalancer zu konfigurieren.

Liegt diese Beschreibung als service.json im aktuellen Verzeichnis, kann mit

```
aws ecs create-service --cli-input-json file://./service.json
```

der Service erzeugt werden und ECS startet vier Instanzen des Tasks. Services können auch über die Weboberfläche erzeugt werden.

Fazit

Amazon ECS bringt Docker-Container auf einfache Weise in die Amazon-Cloud. Durch Verwendung des Service Schedulers und eines auf mehrere Availability Zones verteilten Clusters in Verbindung mit einem Loadbalancer kann man leicht ausfallsichere Anwendungen betreiben. Mit Docker-Containern lassen sich EC2-Instanzen besser auslasten, ohne dabei die Isolation verschiedener Dienste aufgeben zu müssen.

Links

[AWS] Amazon Web Services, `http://aws.amazon.com/`
[Docker] `https://docker.com`
[ECS] Amazon EC2 Container Service, `http://aws.amazon.com/ecs`
[Ghadir14] Ph. Ghadir, Micro-Services in Java realisieren – Teil 2: Web-Apps in Docker-Umgebungen, in: JavaSPEKTRUM 05/2014, s. a. `http://www.sigs-datacom.de/fileadmin/user_upload/zeitschriften/js/2014/05/ghadir_JS_05_14_ld5M.pdf`
[Mesos] `https://mesos.apache.org/`
[Swarm] `http://blog.docker.com/2015/02/scaling-docker-with-swarm/`
[Vogels15] W. Vogels, European Union Data Protection Authorities Approve Amazon Web Services' Data Processing Agreement, 31.3.2015, `http://www.allthingsdistributed.com/2015/03/aws-and-eu-data-protection.html`

Martin Eigenbrodt ist Senior Consultant bei innoQ. Er verfügt über mehrjährige Erfahrung in der Entwicklung von Software auf der JVM. Seine Schwerpunkte sind Design und Implementierung von RESTful Webservices, Continuous Delivery und DevOps mit und ohne Cloud-Technologien.
E-Mail: martin.eigenbrodt@innoq.com

Phillip Ghadir baut am liebsten tragfähige, langlebige Softwaresysteme. Er ist Mitglied der Geschäftsleitung bei innoQ und hat sich früh auf Architekturen für verteilte, unternehmenskritische Systeme spezialisiert. Darüber hinaus ist er Mitbegründer und aktives Mitglied des iSAQB, des International Software Architecture Qualification Board.
E-Mail: phillip.ghadir@innoq.com