Aufgabe 2.2

Das Skript `docker-wordpress.sh` (siehe Listing 1) sorgt für ein automatisiertes Aufsetzen einer WordPress Instanz mittels Docker auf Amazon Linux.

Dazu wird Docker falls notwendig zunächst installiert und anschließend als Service gestartet. Docker lädt nun ein fertiges Image für WordPress inklusive LAMP-Stack. Alternativ könnte hier auch ein entsprechendes Repository mit Dockerfile geladen werden, um das Image lokale via `docker build` zu bauen. Anschließend prüft das Skript ob bereits ein WordPress Container läuft und startet das vorher geladene (oder gebaute) Image mit offenem Port 80. Damit WordPress von außen erreichbar ist, muss zusätzlich noch mittels einer Security-Rule Port 80 für den HTTP-Traffic freigegeben werden. WordPress kann nun im Webbrowser aufgerufen und eingerichtet werden.

Listing 1: Automatisiertes Aufsetzen von Docker

```
1  #!/usr/bin/env sh
2
3  R="\e[1;31m"
4  G="\e[32m"
5  B="\e[34m"
6  RES="\e[0m"
7
8  set -e
9  trap "{ echo -e '${R}Error${RES}'; }" ERR
10
11 # check if docker is installed. If not, install docker
12 if [[ -z $(command -v docker &> /dev/null) ]]; then
13     sudo yum install -y docker
14 fi
15
16 # start docker -> ignored if docker already running
17 echo -e "${B}==> Starting docker${RES}"
18 sudo service docker start
19
20 echo -e "${B}==> Pulling images${RES}"
21 sudo docker pull tutum/wordpress
22
23 # run docker image
24 if [[ -z $(sudo docker ps | grep "tutum/wordpress:latest") ]]; then
25     echo -e "${B}==> Starting wordpress container${RES}"
26     sudo docker run -d --name wordpress -p 80:80 tutum/wordpress
27 else
28     echo -e "${B}Wordpress container already running${RES}"
29 fi
30
31 echo -e "${G}Done${RES}"
```

Diskussion

Bei der Virtualisierung mittels Hypervisor wird eine komplette Maschine inklusive Hardware virtualisiert. Auf dieser virtuellen Maschine muss zunächst ein Betriebssystem installiert werden. Die Virtualisierung erlaubt dabei auch, dass in der VM ein anderes Betriebssystem als auf dem Host-System installiert wird. Da es sich um komplett getrennte Maschinen handelt, wirkt sich die Kompromittierung einer VM nicht auf die anderen aus. Allerdings entsteht durch die Virtualisierung eines kompletten Systems ein relativ großer Overhead, sowohl beim Ressourcenverbrauch bei der Ausführung als auch bei der Imagegröße.

Bei der Virtualisierung mittels Container wird dagegen nur die Laufzeitumgebung der Anwendung innerhalb des Betriebssystems virtualisiert. Dadurch sind die Container im

Vergleich zu VMs klein und benötigen weniger Speicher, da nicht mehrere Betriebssysteme parallel laufen. Die Initialisierung von Containern geht sehr schnell, während bei VMs zunächst ein Betriebssystem gestartet werden muss. Da keine ganze Maschine virtualisiert wird, kann bei der Container-Virtualisierung außerdem die Zuteilung der Systemressourcen dynamisch zur Laufzeit angepasst werden. Das geteilte Betriebssystem der Container stellt gleichzeitig jedoch auch den größten Schwachpunkt dieser Virtualisierungsart dar, da sich jegliche Betriebssystemänderung (z.B. Kernel-Update, Service-Packs, ...) direkt auf alle Container auswirkt.

Aufgabe 2.3

Vorbereitung

Da bei der Verwendung von *Chef Solo* kein Chef-Server genutzt wird, muss Chef zunächst lokal installiert werden und wird von dort genutzt um die VMs in der Cloud einzurichten und zu konfigurieren. Zusätzlich werden lokal einige Tools benötigt, insbesondere `knife-ec2` um EC2-Instanzen zu erzeugen, `librarian-chef` um lokal die benötigten Rezepte herunterzuladen und zu installieren und `knife-solo` um Chef und die Rezepte auf den Servern zu installieren. Um das Einrichten der EC2-Instanzen zu vereinfachen, kann eine Konfigurationsdatei erzeugt werden, in der die Standardparameter zur Kommunikation mit den Amazon-Servern und dem Einrichten der Instanzen gespeichert sind.

```
1  knife[:region] = "<AmazonRegion>"
2  knife[:availability_zone] = "<AvailabilityZone>"
3  knife[:aws_access_key_id] = "<AWS_ACCESS_KEY_ID>"
4  knife[:aws_secret_access_key] = "<AWS_SECRET_ACCESS_KEY>"
5  knife[:image] = "ami-4b471c7b"
6  knife[:flavor] = "t1.micro"
7  knife[:chef_mode] = "solo"
8  knife[:ssh_user] = "ubuntu"
9  knife[:identity_file] = "<PathToSecretKeyForSSHAuthentification>"
10 knife[:aws_ssh_key_id] = "<KeyNameInAWSDashboard>"
```

Nachdem alle Tools installiert sind, muss lokal zunächst eine Küche eingerichtet werden:

```
knife solo init .chef/knife-solo
```

In dieser Küche müssen nun die benötigten Kochbücher im `Cheffile` spezifiziert und mittels `librarian-chef install` heruntergeladen und installiert werden.

Aufgabe 2.3

Um eine WordPress Instanz auf einer EC2-Instanz zu installieren, muss eine EC2-Instanz mit Security-Regeln für SSH, HTTP und HTTPS erzeugt werden und der Server dann für die Nutzung mit Chef vorbereitet werden, also insbesondere Chef installiert werden. Die benötigte `<Server-IP>` für diesen Befehl wird bei der Server-Instanziierung angezeigt.

```
knife ec2 server create --groups=sshhttphttps
knife solo prepare ubuntu@<Server-IP>
```

Um `knife` mitzuteilen, welche Kochbücher auf dem Server installiert werden sollen, muss dies in der lokalen Konfiguration für diesen Server festgelegt werden. In dieser Datei können auch die Attribute, die im Kochbuch beschrieben sind, festgelegt werden. Bei dieser Aufgabe haben wir uns für fast alle Attribute auf die Standardwerte beschränkt und fordern beispielhaft nur, dass die aktuellste WordPress-Version installiert wird.

```
{
    "run_list": [
        "recipe[wordpress]"
    ],
    "wordpress": {
        "version": "latest"
    }
}
```

Nun kann die Instanz gekocht werden, also WordPress inklusive aller Abhängigkeiten auf dem Server installiert werden:

```
knife solo cook ubuntu@<Server-IP>
```

Die WordPress-Installation ist nun unter der `<Server-IP>` erreichbar.

Diskussion

Chef ist ein Werkzeug zum Konfigurationsmanagement. Mit Chef kann ein komplexes Setup eingerichtet werden, von der Installation von Anwendungen über das Erzeugen und Einrichten von Nutzern und Gruppen bis zur Konfiguration eines Systems und der Anwendungen. Außerdem ermöglicht Chef es, die Konfiguration laufender Systeme anzupassen. Des Weiteren stehen viele Kochbücher bereits zur Verfügung, die ein schneller installieren von Anwendungen auf Servern erlauben. Diese sind Betriebssystem unabhängig verfasst, so dass mit demselben Kochbuch z.B. WordPress sowohl auf Windows als auch auf Linux eingerichtet werden kann. Allerdings haben die Kochbücher von Chef externe Abhängigkeiten (z.B. Softwarequellen im Internet), es ist also möglich, dass Kochbücher nach einiger Zeit nicht mehr funktionieren, da die Abhängigkeiten nicht mehr zur Verfügung stehen. Außerdem initialisieren die Kochbücher nicht das System oder bringen es in einen bestimmten Zustand, sondern werden auf dem aktuellen Stand des Systems ausgeführt. Dadurch entsteht die Gefahr von versteckten Abhängigkeiten, dass z.B. ein Tool auf der einen Maschine bereits installiert ist, nicht jedoch auf einer anderen, dies beim Erstellen des Kochbuchs jedoch nicht beachtet wurde.

Um bei Chef ein weiteres System aufzusetzen, muss wieder das gesamte Kochbuch ausgeführt werden. Bei Docker dagegen kann von einem existierenden Image einfach eine weitere Instanz gestartet werden oder basierend auf diesem Image weitere Anwendungen installiert werden. Docker ermöglicht es mit der Snapshot-Funktion außerdem, verschiedene Versionen des Containers zu versionieren.

Da Chef und Docker jedoch eigentlich unterschiedliche Anwendungsgebiete haben, können sie auch gemeinsam genutzt werden. So kann Chef dafür genutzt werden, Container zu bauen, zu starten und zu verwalten.

Aufgabe 2.4

VMs

Sofern der Vorbereitungsschritt von Aufgabe 2.3 noch nicht ausgeführt wurde, muss dieser zunächst ausgeführt werden.

Das Skript `run.sh` (siehe Listing 2) setzt zwei unterschiedliche EC2-Instanzen auf und installiert mittels Chef auf der einen MySQL und auf der anderen das zugehörige WordPress.

Dazu wird zunächst der Datenbankserver eingerichtet. Dieser wird mit den Secruity-Regeln für SSH- und MySQL-Zugriff instanziiert und Chef installiert. Für den Datenbankserver haben wir ein neues Kochbuch erzeugt, das das Einrichten des Wordpress-Nutzers und Datenbank ermöglicht. Dann wird der Server mit der zugehörigen Konfigurationsdatei von Chef eingerichtet. In dieser Konfigurationsdatei muss das MySQL-Root-Passwort und die Zugriffsdaten des MySQL-WordPress-Nutzers festgelegt werden. Um den externen Zugriff auf den Server zu ermöglichen, muss dies in der Datenbank explizit erlaubt werden, dies wird durch ein Skript ermöglicht.

Danach wird der WordPress-Server eingerichtet. Auch dieser wird zunächst mit den Seurity-Regeln für SSH-, HTTP- und HTTPS-Zugriff instanziiert und Chef installiert. In der Konfigurationsdatei für den WordPress-Server muss der Datenbank-Server und der MySQL-Nutzer eingetragen werden. Der Benutzername und das Passwort für den Datenbankbenutzer muss dabei dem auf dem Datenbankserver eingerichteten Benutzer entsprechen. Danach wird der Server mit der Konfigurationsdatei von Chef eingerichtet.

Die WordPress-Installation ist nun unter der Adresse der WordPress-Instanz erreichbar.

Listing 2: Automatisiertes Aufsetzen zweier VMs für WordPress und MySQL

```
1  #!/usr/bin/env bash
2
3  # MySQL-Server
4  echo "Creating MySQL-Server:"
5  knife ec2 server create --groups=sshmysql | tee createsql.tmp
6  sqlIP=$(grep "Public DNS Name:" createsql.tmp | tail -1)
7  sqlIP=${sqlIP#*:}
8  sqlIP=\${sqlIP#*:}
9  rm createsql.tmp
10 knife solo prepare ubuntu@$(echo $sqlIP)
11 cp nodes/mysql.json nodes/$(echo $sqlIP).json
12 knife solo cook ubuntu@$(echo $sqlIP)
13 scp -i ~/.ssh/cloudfapra.pem helpers/my.cnf ubuntu@$(echo $sqlIP):.my.cnf
14 scp -i ~/.ssh/cloudfapra.pem helpers/flush.sql ubuntu@$(echo $sqlIP):.
15 ssh -i ~/.ssh/cloudfapra.pem ubuntu@$(echo $sqlIP) 'mysql<flush.sql;rm flush.sql .my.cnf'
16 echo "MySQL-Server ready and running."
17
18 # WordPress-Server
19 echo "Creating WordPress-Server:"
20 knife ec2 server create --groups=sshhttphttps | tee createwp.tmp
21 wpIP=$(grep "Public DNS Name:" createwp.tmp | tail -1)
22 wpIP=${wpIP#*:}
23 wpIP=${wpIP#*:}
24 rm createwp.tmp
25 knife solo prepare ubuntu@$(echo $wpIP)
26 cp nodes/wpnosql.json nodes/$(echo $wpIP).json
27 sed -i "s/MYSQLSERVER/$(echo $sqlIP)/g" nodes/$(echo $wpIP).json
28 knife solo cook ubuntu@$(echo $wpIP)
29 echo "WordPress-Server ready and running."
30 echo "All done. Your WordPress-Instance: $wpIP"
```

Aufgabe 2.4 - Container

Das Skript `run-all.sh` (siehe Listing 3) sorgt für ein automatisiertes Aufsetzen einer WordPress Instanz mittels zweier Docker Container (auf Amazon Linux), wobei der eine die Webapp WordPress selbst enthält, während im anderen Container die Datenbank liegt.

Dazu wird Docker zunächst installiert, falls notwendig, anschließend als Service gestartet. Docker läd nun ein fertiges Image für WordPress ohne LAMP Stack sowie ein Image für eine Datenbank, hier MySQL. Alternativ könnten hier auch entsprechende Repositories mit Dockerfile(s) geladen werden, um die Images lokale via `docker build` zu bauen.

Das Skript startet nun zunächst den Container für die Datenbank und anschließend den für Wordpress, wobei ein Link zum Datenbank-Container hergestellt wird. Ein Docker Link sorgt für eine sicherer Verbindung zwischen Containern, die Informationen über Umgebungsvariablen austausschen können. In unserem Fall werden Informationen wie DB_HOST oder DB_PORT kommuniziert.

Listing 3: Automatisiertes Aufsetzen zweier Container für WordPress und MySQL

```
#!/usr/bin/env sh

R="\e[1;31m"
G="\e[32m"
B="\e[34m"
RES="\e[0m"

set -e
trap "{ echo -e '${R}Error${RES}'; }" ERR

# check if docker is installed. If not, install docker
if [[ -z $(command -v docker &> /dev/null) ]]; then
    sudo yum install -y docker
fi

# start docker -> ignored if docker already running
echo -e "${B}==> Starting docker${RES}"
sudo service docker start

echo -e "${B}==> Pulling images${RES}"
sudo docker pull tutum/mysql:5.5
sudo docker pull tutum/wordpress-stackable

# run docker images, link them, exchange information with environment variables
# mysql
if [[ -z $(sudo docker ps | grep "tutum/mysql:5.5") ]]; then
    echo -e "${B}==> Starting database container${RES}"
    sudo docker run -d -e MYSQL_PASS="cloud1fapra" --name db -p 3306:3306 tutum/mysql:5.5
else
    echo -e "${B}Database container already running${RES}"
fi
# wordpress
if [[ -z $(sudo docker ps | grep "tutum/wordpress-stackable") ]]; then
    echo -e "${B}==> Starting wordpress container${RES}"
    sudo docker run -d --link db:db -e DB_PASS="cloud1fapra" --name wordpress -p 80:80 tutum/wordpress-stackable
else
    echo -e "${B}Wordpress container already running${RES}"
fi

echo -e "${G}Done${RES}"
```

Aufgabe 2.5

Verwendung eines ECS Cluster. Kern für die Erstellung der Wordpress Instanz ist ein „Service“ mit einem Task. Der Task beinhaltet die Erstellung zweier Docker Container

(Worpress und MySQL). Die Konfiguration des Sercive der Container inklusive der Verknüpfen des Wordpress Containers mit dem MySQL Datenbank Container erfolgt über eine JSON Konfigurationsdatei (siehe Listing 4). Das Element „containerDefinitions“ beschreibt alle erforderlichen Container.

Listing 4: Konfigurationsdatei für Docker Container

```
{
    "family": "Clould_FaPra_SS15_G2_A2-5",
    "containerDefinitions":[
    {
        "name": "wordpress",
        "links":[
            "mysql"
        ],
        "image":"wordpress",
        "essential":true,

        "memory":300,
        "cpu":10,
        "portMappings":[
        {
            "containerPort":80,
            "hostPort":80
        }
        ]
    },
    {
        "name":"mysql",
        "image":"mysql",
        "essential":true,
        "memory":400,
        "cpu":10,
        "environment":[
        {
            "name":"MYSQL_ROOT_PASSWORD",
            "value":"Clould_FaPra_SS15_G2_A2-5"
        },
        {
            "name":"MYSQL_DATABASE",
            "value":"wordpress"
        }
        ]
    }
    ]
}
```