ulm university universität uulm

Cross-organizational distributed systems and Clouds

Exercise 6: Container Orchestration 1

Christopher B. Hauser

Institute of Information Resource Management

2018-06-14

Exercise 6: Container Orchestration 1

Welcome to exercise 6. This time we will have the following lessons:

1. Container Orchestration with Docker Swarm
2. Container Orchestration with Rancher

Lessons learned

- How to install and use Docker Swarm
- How to install and use Rancher
- What are core features of Container Orchestrators
- How to deploy and use a private Docker Registry

Lesson 1: Container Orchestration with Docker Swarm

In this lesson, we will install and use Docker Swarm, as the built in Docker orchestrator across multiple hosts.

Research: How Docker Swarm works

Make yourself familiar with Docker Swarm. The two links should give you enough information to work with Swarm in the following:

https://docs.docker.com/engine/swarm/how-swarm-mode-works/nodes/

https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/

Questions: Docker Swarm

- In the Swarm terminology, what are services, tasks, and containers?
- Where in our Cloud Stack do you place Docker Swarm?

Cloud Stack	Example	Deployment Tool
Application Component	Mediawiki	?
Containers	Docker	?
Virtual Resource	Instance m1.small	?
Cloud Platform	OpenStack	-

Task: Setup Docker Swarm

Deploy three virtual machines with docker installed on bwcloud. We can use the terraform bundle for this purpose.

Log in to the three VMs via SSH. Validate that Docker is running (e.g. via `docker ps -a`). Docker Swarm is inactive, validate output of `docker info` and search for line `Swarm: inactive`.

Lets take dockernode1 as swarm manager. Log in to dockernode1, look up the private IP address of this VM (`$PRIV_IP_DOCKERNODE1`), then run `docker swarm init --advertise-addr $PRIV_IP_DOCKERNODE1`. The output of this command returns:

```
Swarm initialized: current node (....) is now a manager.
```

```
To add a worker to this swarm, run the following command:

  docker swarm join \
  --token SWMTKN-1..... \
  192.168.5.4:2377
```

Copy/Paste the `docker swarm join` command to dockernode2 and dockernode3. Validate your Docker Swarm cluster on dockernode1 with `docker node ls` - you should see three hosts.

Task: Start a test service

For testing purposes we will now start three instances of the Ghost blog software on our Swarm cluster. Therefore create the service with 3 replicas:

```
docker service create --replicas 3 --name ghost-blog\
  --publish 80:2368 ghost
```

Validate that the service is booting with `docker service ls`, where the counter in column "REPLICAS" will go from 0/3 to 3/3 eventually. You can see that the three replicas are distributed among the three nodes: `docker service ps ghost-blog`. On each virtual machine run `docker ps`, you will have one container of the ghost service on each node. Take a browser, and go to any of your three floating IPs on port 80 to access the ghost instances.

Swarm allows you to scale horizontally up and down (`docker service scale $SERVICE_NAME=$NUMBER_OF_REPLICAS`). You can easily scale down the ghost service to only one replica with `docker service scale ghost-blog=1`. Validate the number of replicas with `docker service ps ghost-blog`. Note: if your container runs on e.g. dockernode1 you can access the ghost blog via http://$FLOATING_IP_DOCKERNODE1/. But Swarm handles published ports across all swarm nodes. You can also access the ghost blog via any other floating ip.

Updating the software is also handled by Swarm, where sequentially containers are removed and recreated to achieve zero downtime of the service. In case of errors, Swarm can rollback to the previous version.

Remove the test service with `docker service rm ghost` before you continue.

Task: Mediawiki with Docker Swarm

Docker Swarm and Docker Compose work together (software is experimental at present).

So far we built the images from Dockerfiles on the node where we started containers from those images. Since we're having a distributed Swarm cluster, we need a central store for our built images: a (private) docker registry.

We could use Docker Hub, or deploy our own registry. On dockernode1, run the following commands to run your own docker registry:

```
# install certbot to get certificate
sudo apt-get install -y software-properties-common
sudo add-apt-repository -y ppa:certbot/certbot
sudo apt-get update
sudo apt-get install -y certbot

# lookup your public hostname
floatingIP=$(curl  -s  http://169.254.169.254/latest/meta-data/public-
ipv4)
publicHostname=$(nslookup $floatingIP | grep ”in-addr.arpa” |\
   cut -d'=' -f2 | tr -d ' ''')

# get certificate
sudo certbot certonly --standalone -d $publicHostname --agree-tos \
  --register-unsafely-without-email --preferred-challenges http
sudo mkdir /opt/certs
folder=$(ls /etc/letsencrypt/live/)
sudo cp /etc/letsencrypt/live/$folder/fullchain.pem /opt/certs/
sudo cp /etc/letsencrypt/live/$folder/privkey.pem /opt/certs/

# start registry as container
docker run -d -p 5000:5000 --restart=always --name registry \
  -v /opt/certs:/certs \
  -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/fullchain.pem \
  -e REGISTRY_HTTP_TLS_KEY=/certs/privkey.pem \
  registry:2
```

Next, copy the Dockerfiles from exercise 5 to dockernode1. Change the `docker-compose.yml` file to the following content (replace `bwcloud-fipXYZ` to fit your `$publicHostname`):

```
version: '3'
services:
    web:
        build: Mediawiki
        image: bwcloud-fipXYZ.rz.uni-ulm.de:5000/mediawiki
        ports:
          - 80:80
        deploy:
          replicas: 2
          mode: replicated
    database:
        build: Database
        image: bwcloud-fipXYZ.rz.uni-ulm.de:5000/database
        deploy:
          replicas: 1
          mode: replicated
```

Then build and push the images to your registry (still on dockernode1):

```
docker-compose build
docker push bwcloud-fipXYZ.rz.uni-ulm.de:5000/database
docker push bwcloud-fipXYZ.rz.uni-ulm.de:5000/mediawiki
```

Validate, that the registry works. Go to dockernode2 and download the image via `docker pull bwcloud-fipXYZ.rz.uni-ulm.de:5000/database`

Now we can use Docker Swarm to deploy our Docker Compose file. On dockernode1, run:

```
docker stack deploy --compose-file docker-compose.yaml mediawiki
```

Validate with `docker stack ps mediawiki` that you have three containers running, and check if you can access the mediawiki application via web browser.

We have now deployed the mediawiki in Docker containers across several (manually created) Docker hosts.

Lesson 2: Container Orchestration with Rancher

Docker Swarm offers clustering of hosts to one large container cluster, and orchestrates services by placing containers on available hosts. Rancher is one level above: it offers a web gui to

Research: Rancher

What is Rancher? Become familiar with the tool before we start.

http://rancher.com/

http://docs.rancher.com/rancher/v1.6/en/

Question: Rancher

Where in our Cloud Stack do you place Rancher?

Cloud Stack	Example	Deployment Tool
Application Component	Mediawiki	?
Containers	Docker	?
Virtual Resource	Instance m1.small	?
Cloud Platform	OpenStack	-

Task: Install and start Rancher

In bwcloud create a new keypair via `Access&Security > Key Pairs > Create Key Pair` with name rancher. Next, use the terraform scripts in `terraform-rancher` to start a new virtual machine with docker and a docker registry. When the VM is up, place your Keypair rancher.pem file inside the vm at /opt/keypair/.

Now start a Rancher service (it takes several seconds until rancher is accessible):

```
docker run -d --restart=unless-stopped -p 8080:8080 --name rancher \
  -v /opt/keypair:/opt/keypair \
  rancher/server:stable
```

Next, we will add hosts (virtual machines with a Rancher agent) to Rancher, while Rancher will create the virtual resources. Open the web dashboard of your Rancher server http://PUBLIC_IP:8080/ and run the following instructions:

- Admin > Machine Drivers
- Activate OpenStack by pressing play button
- Infrastructure > Hosts
- Press “Add Host” Button
- Select “Other” and in the form insert the following values:

Field	Value
Name	rancher-hosts
Quantity	3
Driver	openstack
authUrl	https://bwcloud.ruf.uni-freiburg.de:5000/v2.0
availabilityZone	nova
flavorName	m1.small
floatingipPool	routed
imageName:	Ubuntu Server 16.04 RAW
keypairName:	rancher
netName:	private-net
password:	YOUR_BWCLOUD_PW
privateKeyFile:	/opt/keypair/rancher.pem
region:	Ulm
secGroups:	default,rancher
sshUser:	ubuntu
tenantName:	Projekt_ehx27@uni-ulm.de

Field	Value
username:	ehx27@uni-ulm.de

- Press the Create button and wait for the virtual machines to be spawned…

Starting the rancher hosts will take some time. You can follow the process: the virtual machines will be created, Docker will be installed, Rancher agents will be installed, and several predefined containers like healthchecker, or scheduler will be started. When the hosts are up, you can browse through the utilisation metrics and the deployed containers in the web dashboard. When selecting single containers, you get get statistics about this container only, its even possible to get execution shells into containers from the dashboard.

Finally install the Rancher CLI tool inside the rancher VM:

```
wget -O rancher.tar.gz \
     https://github.com/rancher/cli/releases/download/v0.6.1/rancher-
linux-amd64-v0.6.1.tar.gz
tar xfzv rancher.tar.gz
sudo mv rancher-v0.6.1/rancher /usr/bin/
sudo chmod +x /usr/bin/rancher
```

Besides the web dashboard you can now connect to rancher via CLI.

To authenticate in the CLI, you need an API Key. In the web dashboard, to to `API > Keys` and press the `Add Account API Key` Button. Enter any name and press create. You will get an Access Key and a Secret Key. Copy both to a place where you will find them later.

To use the CLI, run `rancher config` and provide `URL []: http://134.60.47.XYZ:8080/v1`, the Access Key and Secret Key from before. Validate that the login works with `rancher ps`.

Task: Start Mediawiki via Rancher

Copy and unzip the dockerfiles.zip from exercise 4 to the rancher vm. Inside the extracted dockerfiles folder, create a docker-compose.yml with the following content:

```
version: '2'
services:
    web:
        build: Mediawiki
        image: bwcloud-fipXYZ.rz.uni-ulm.de:5000/mediawiki
        ports:
          - 80:80
    database:
        build: Database
        image: bwcloud-fipXYZ.rz.uni-ulm.de:5000/database
```

Then build and push the images to your registry:

```
docker-compose build
docker push bwcloud-fipXYZ.rz.uni-ulm.de:5000/database
docker push bwcloud-fipXYZ.rz.uni-ulm.de:5000/mediawiki
```

Now we are ready to import the docker-compose file as a `stack` into Rancher, either by web dashboard or by cli.

In the web dashboard, go to “Stacks”, and press the “Add Stack” button. Add as name “mediawiki” and place as content of docker-compose.yml:

```
version: '2'
services:
    web:
        image: bwcloud-fipXYZ.rz.uni-ulm.de:5000/mediawiki
        ports:
          - 80:80
    database:
        image: bwcloud-fipXYZ.rz.uni-ulm.de:5000/database
```

Mediawiki will be started. Try to access it. You will recognize, that it is only accessible via the floating ip of the host where the web container is running.

To allow any of the rancher hosts, we have to add a load balancer: In the mediawiki stack, click “add service” and select “load balancer”. Scale: Always run one instance of this container on every host Name: wiki-loadblancer Request Port 80, Target: web Port 80.

When selecting the web service, you can configure the Health Check, so rancher will restart failed containers automatically e.g. when the web server is not responding (Docker will only react on stopped containers). When you edit a service, you can scale it up and down. The loadbalancer will be configured automatically.