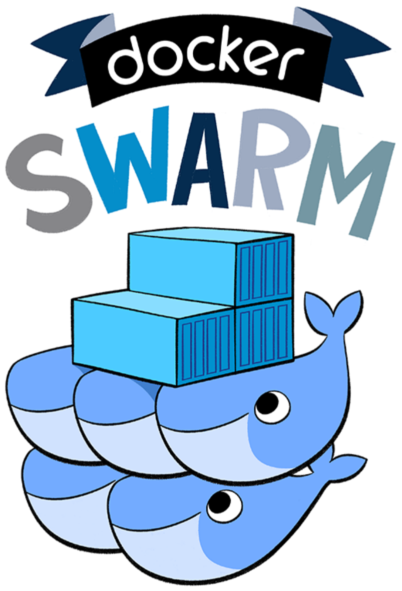
SE HABLA CODE LOSTECHIES

Docker and Swarm Mode – Part 1

Posted by Gabriel Schenker on September 5, 2016



In the following few posts I am going to demonstrate how we can use the new SwarmKit that is part of Docker 1.12 to manage a cluster of nodes (VMs) as a Docker Swarm. To not depend on any cloud provider we will be using VirtualBox on our developer machine to generate such a swarm. I will show how easy it is to get a completely working swarm in place and how to run an application consisting of a bunch of services on this swarm. The application that we're going to use is borrowed from Jérôme Petazzo from Docker. This application is a good sample for a microservices based application using various different frameworks and languages to implement individual services. It would be a nightmare to have to run this application natively on a host due to all the different technologies involved. But with Docker and a Docker Swarm it is a breeze and straight forward.

Let's start with part 1 and let's immediately dive into this adventure…

Generate a cluster

In this post I will use Virtualbox to be able to work with multiple VMs. Please make sure you have Docker and Virtualbox installed on your system. The easiest way to do so is by installing the Docker Toolbox.

Docker Toolbox by default installs a tiny VM called `default` in Virtualbox. Let's stop this VM (and any other that might be running) to have sufficient resources for our cluster. Open a terminal (e.g. Docker Quickstart) and execute the following command

```
docker-machine stop default
```

Ads

Search

Recent Posts

- Containers – Cleanup your house revisited
- Docker and Swarmkit – Part 6 – New Features of v1.13
- Docker and SwarmKit – Part 5 – going deep
- How To Bootstrap Angular with Server Side Data
- Docker and Swarmkit – Part 4

Recent Comments

- alioygur on Blue-Green Deployment in Docker Cloud
- armando sard on DDD – The aggregate
- gabrielschenker on DDD – The aggregate
- armando sard on DDD – The aggregate
- armando sard on DDD – The aggregate

Archives

- December 2016
- November 2016
- October 2016
- September 2016
- August 2016
- June 2016
- May 2016
- April 2016
- March 2016
- February 2016
- January 2016
- September 2015
- August 2015
- July 2015
- June 2015
- May 2015
- April 2015
- March 2015
- January 2015
- December 2014
- May 2014

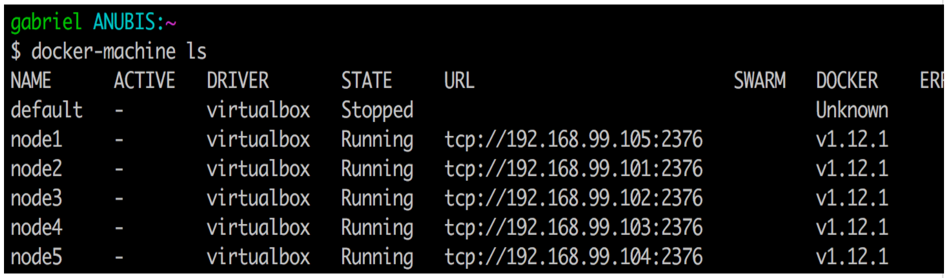
We now want to create a cluster of 5 nodes. We can do that manually using `docker-machine` or use a small script to quickly generate a bunch of nodes in VirtualBox

```
for N in 1 2 3 4 5; do docker-machine create --driver virtualbox node$N; done
```

The above command might take a few minutes, please be patient. Once done, double check that all nodes are up and running as expected

```
docker-machine ls
```

You should see something similar to this



```
gabriel ANUBIS:~
$ docker-machine ls
NAME      ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER    ER
default   -        virtualbox   Stopped                                       Unknown
node1     -        virtualbox   Running   tcp://192.168.99.105:2376           v1.12.1
node2     -        virtualbox   Running   tcp://192.168.99.101:2376           v1.12.1
node3     -        virtualbox   Running   tcp://192.168.99.102:2376           v1.12.1
node4     -        virtualbox   Running   tcp://192.168.99.103:2376           v1.12.1
node5     -        virtualbox   Running   tcp://192.168.99.104:2376           v1.12.1
```

`ssh` into the first node with the help of `docker-machine` using this command

```
docker-machine ssh node1
```

Once logged into node1 verify that Docker is in the latest version by issuing the command

```
docker info
```

Docker-Compose

Our nodes on Virtualbox consist of a minimal Linux installation with Docker and do not have `docker-compose` installed which we will need in our exercise. Fortunately that's no big deal, we can use a container image which has `docker-compose` installed and use this instead of having `docker-compose` directly installed on the node. Thankfully Docker has created such an image. Make sure to use the latest version of it which at the time of writing is 1.8.0. We can pull this image like

```
docker pull docker/compose:1.8.0
```

and then we can run a container with compose like this

```
COMPOSE_VERSION=1.8.0
docker run --rm -it \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -v $(pwd):/app \
    --workdir /app \
    docker/compose:$COMPOSE_VERSION --version
```

- April 2014
- February 2014
- January 2014
- December 2013
- July 2012
- June 2012
- December 2011
- September 2011
- August 2011
- January 2011
- October 2010
- September 2010
- July 2010
- June 2010
- May 2010
- January 2010
- December 2009
- November 2009
- October 2009
- September 2009
- June 2009
- May 2009
- April 2009
- March 2009
- February 2009
- January 2009

Categories

- .NET
- AngularJS
- architecture
- ASP.NET vNext
- asynchronous
- AWS
- book
- bootstrapper
- brownfield
- BUILD
- CI/CD
- ClickOnce
- commands
- Conference
- containers
- continuous deployment
- CQRS
- data binding
- DDD
- delegates
- design
- developer
- didactics
- docker
- E2E testing
- Elasticsearch
- Event sourcing
- Fluent NHibernate
- fluent Silverlight
- framework
- GetEventStore

docker-compose.sh hosted with ❤ by GitHub view raw

Note how I mount the `docker.sock` to have direct access to Docker on the host from within the container and I also mount the working directory into the container to have access to the files on the host like the `docker-compose.yml` file, etc. If I run the above command the version of docker-compose will be printed. To simplify my life I can define an alias as follows

```
alias docker-compose='docker run --rm -it -v
/var/run/docker.sock:/var/run/docker.sock -v $(pwd):/app --workdir /app
docker/compose:1.8.0'
```

and then we can use it like this

```
docker-compose --version
```

to e.g. print the version or

```
docker-compose up
```

to use the `docker-compose.yml` file in the current directory and run the application described in there.

Working with the Swarmkit

Now we're ready to create a new Docker swarm. Note that Docker (starting from version 1.12) can run in two modes, **classical** and **swarm mode**. The former is there for backwards compatibility and the latter uses the new swarm kit that is now part of Docker Engine. To initialize a new swarm use this command on node1:

```
docker swarm init --advertise-addr [ip-address]
```

where `[ip-address]` is the public IP address of the node (e.g. 192.168.99.101). The above command will tell us in the output which command to use to join other worker nodes to the swarm. In my case this looks like this

```
$ docker swarm init --advertise-addr 192.168.99.105
Swarm initialized: current node (ddec2zwjaes2kpkmnshkmpstq) is now a manager.

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

    docker swarm join \
    --token SWMTKN-1-3jv31q64ofkl3j20d8lxeq2eejyhcde7h1o25p6c0i1b1a2e3x-az1nzpiuxaa1
    192.168.99.105:2377


To add a manager to this swarm, run 'docker swarm join-token manager' and follow the
```

init-swarm.sh hosted with ❤ by GitHub view raw

- Git
- hiring
- How To
- installation
- interactive extensions
- introduction
- JAVA
- job
- jQuery
- lambda expressions
- learning
- legacy code
- Lucene
- methodology
- Micro services
- misc
- Mock
- mocking framework
- MongoDB
- Mono
- Mono Cecil
- MSIL
- multi threading
- MVVM pattern
- NHibernate
- no-database
- node
- NSIS
- patterns
- personal
- plugins
- practices
- presentation
- PTOM
- queries
- Read model
- refactoring
- reflection
- reporting
- REST
- review
- Setup
- Silverlight
- SOLID
- Styles and Templates
- TDD
- tutorial
- Ubuntu
- Uncategorized
- upgrade
- validation
- VM
- Windows
- WIX
- Workflow
- workshop

Meta

- Log in

- Entries RSS
- Comments RSS
- WordPress.org

Don't worry if you forget this command. At any time we can retrieve it again using

```
docker swarm join-token worker
```

to get the join command for a worker or

```
docker swarm join-token manager
```

to get the equivalent for a node that should join as a manager.

Open another terminal window and `ssh` into `node2`

```
docker-machine ssh node2
```

and run the join command needed for a worker (in my case this is)

```
docker swarm join \
    --token SWMTKN-1-3jv31q64ofkl3j20d8lxeq2eejyhcde7h1o25p6c0i1b1a2e3x-az1nzpiuxaa1r
    192.168.99.105:2377
```

join-swarm-as-worker.sh hosted with ❤ by GitHub view raw

In your case you will of course have another swarm token and probably a different IP address.

Now, we can do the very same for nodes 3 to 5 but that's a bit tedious, especially if we don't have 5 but 10 or more nodes. Let's automate this

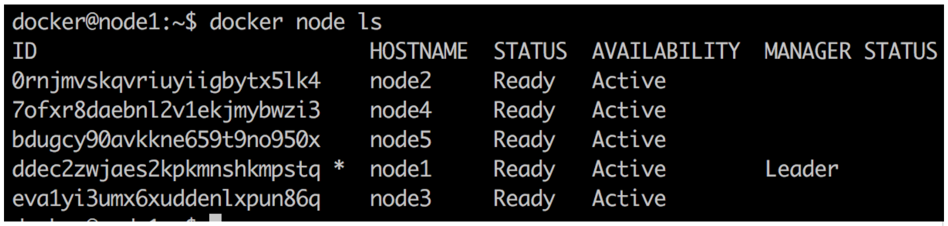
```
for NODE in node3 node4 node5; do
  docker-machine ssh $NODE docker swarm join \
    --token SWMTKN-1-3jv31q64ofkl3j20d8lxeq2eejyhcde7h1o25p6c0i1b1a2e3x-az1nzpiuxaa1r
    192.168.99.105:2377
done
```

gistfile1.txt hosted with ❤ by GitHub view raw

OK, so now we have a 5 node swarm as we can easily test by running the command

```
docker node ls
```

on `node1`. We should see something similar to this

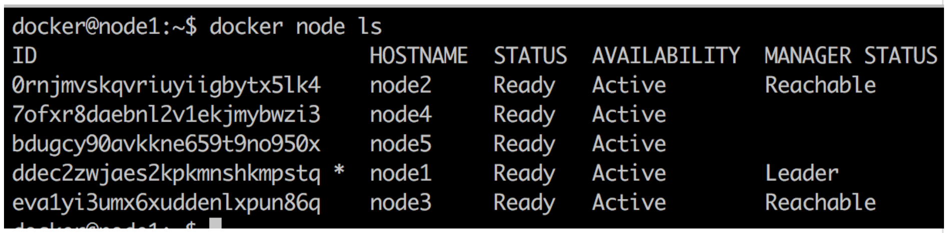


```
docker@node1:~$ docker node ls
ID                           HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
0rnjmvskqvriuyiigbytx5lk4    node2     Ready   Active
7ofxr8daebnl2v1ekjmybwzi3    node4     Ready   Active
bdugcy90avkkne659t9no950x    node5     Ready   Active
ddec2zwjaes2kpkmnshkmpstq *  node1     Ready   Active        Leader
eva1yi3umx6xuddenlxpun86q    node3     Ready   Active
```

We can see that we have one master node (node1) that is also the leader in a quorum of master nodes. The other 4 nodes are all worker nodes. Worker nodes can be promoted to master nodes and vice versa. In a production environment we should have at least 3 master nodes for high availability since master nodes store all the information about the swarm and its state. Let's promote `node2` and `node3` to **master** status using this command

```
docker node promote node2 node3
```

and then double check the new status with `docker node ls` where we should see this

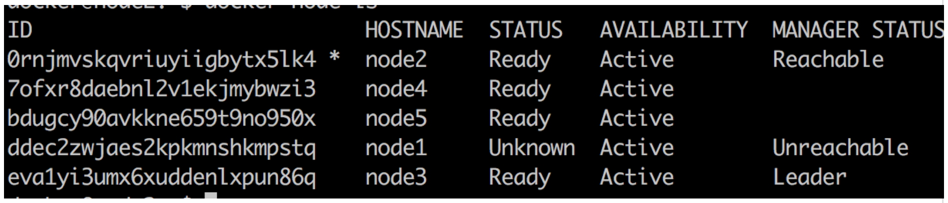


```
docker@node1:~$ docker node ls
ID                           HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
0rnjmvskqvriuyiigbytx5lk4    node2     Ready   Active        Reachable
7ofxr8daebnl2v1ekjmybwzi3    node4     Ready   Active
bdugcy90avkkne659t9no950x    node5     Ready   Active
ddec2zwjaes2kpkmnshkmpstq *  node1     Ready   Active        Leader
eva1yi3umx6xuddenlxpun86q    node3     Ready   Active        Reachable
```

Evidently `node2` and `node3` are now also master nodes and are "reachable". `node1` still remains the leader, but **if** for some reason it goes away then `node2` or `node3` will take the leader position. Let's try that and use `docker-machine` to stop `node1`

```
docker-machine stop node1
```

by doing this we are of course kicked out of our ssh session on node1. Let's ssh into node2, one of the other leaders and again use `docker node ls` to check the new status. You might need to give it some time to reach the final state



```
ID                           HOSTNAME  STATUS   AVAILABILITY  MANAGER STATUS
0rnjmvskqvriuyiigbytx5lk4 *  node2     Ready    Active        Reachable
7ofxr8daebnl2v1ekjmybwzi3    node4     Ready    Active
bdugcy90avkkne659t9no950x    node5     Ready    Active
ddec2zwjaes2kpkmnshkmpstq    node1     Unknown  Active        Unreachable
eva1yi3umx6xuddenlxpun86q    node3     Ready    Active        Leader
```

And we see that now the former leader node1 is unreachable while node3 became the new leader. Let's start node1 again and after it has stabilized it will be a master again but not the leader. Node3 will remain leader.

Private Registry

When we are building Docker images we have to store them somewhere. Usually we can use Docker Hub to store public images for free or if we have a private account we can also store private images there. But if that is not OK and we want to have our own private registry for images then we can use the Docker registry in our environment. Docker registry is OSS and is run as a container. By default images are stored in the container and thus will be lost if the container is removed or crashes. But it is very easy to configure the container to use a durable storage for the images like AWS S3 or so. To run the OSS version of Docker registry as a service listening on port 5000 use this command.

```
docker service create --name registry --publish 5000:5000 registry:2
```

By publishing the port on the swarm we make sure the registry can be reached from each node. This is a “trick” so each node can communicate with the registry via `localhost:5000` and we don’t have to use TLS.

Once the service is running we can now (on any node of our cluster) execute the following command to get a list of all images

```
curl localhost:5000/v2/_catalog
```

Let’s test the registry. We can try to push e.g. the official alpine which we first want to pull from Docker hub

```
docker pull alpine
```

We use the alpine image since it is so small. Now to be able to push it to our private registry we first need to tag the image

```
docker tag alpine localhost:5000/alpine
```

and then push it

```
docker push localhost:5000/alpine
```

If we now query the registry we get this

```
docker@node1:~$ curl localhost:5000/v2/_catalog
{"repositories":["alpine"]}
```

query-registry hosted with ❤ by GitHub view raw

Building and pushing services

Let’s first clone the sample application

```
git clone https://github.com/jpetazzo/orchestration-workshop
```

cd into the source directory of the repository

```
cd orchestration-workshop/dockercoins
```

and then build and push all the services using this script

```
REGISTRY=localhost:5000
TAG=v0.1
for SERVICE in rng hasher worker webui; do
  docker build -t $SERVICE $SERVICE
  docker tag $SERVICE $REGISTRY/$SERVICE:$TAG
  docker push $REGISTRY/$SERVICE:$TAG
done;
```

build-and-push.sh hosted with ♥ by GitHub view raw

If we query the registry again we should now find all the services just built in the catalog too.

Summary

In part 1 I have demonstrated how we can easily create a new Docker swarm on our development machine using VirtualBox and the Docker SwarmKit. We have learned how we can add nodes to the swarm and promote or demote them from worker to to master status and vice versa. We have also experienced what happens if the leader of the master nodes disappears. Another master node takes its role and the swarm continues to work just normally. Finally we have installed a private Docker registry in our cluster which we'll be using to store our Docker images that we build from the sample application.

In part 2 we will use the new `docker service` keyword to create, scale, and manipulate individual services. Stay tuned.

About Gabriel Schenker
Gabriel N. Schenker started his career as a physicist. Following his passion and interest in stars and the universe he chose to write his Ph.D. thesis in astrophysics. Soon after this he dedicated all his time to his second passion, writing and architecting software. Gabriel has since been working for over 25 years as a consultant, software architect, trainer, and mentor mainly on the .NET platform. He is currently working as senior software architect at Alien Vault in Austin, Texas. Gabriel is passionate about software development and tries to make the life of developers easier by providing guidelines and frameworks to reduce friction in the software development process. Gabriel is married and father of four children and during his spare time likes hiking in the mountains, cooking and reading.
View all posts by Gabriel Schenker →

This entry was posted in containers, docker, Elasticsearch, How To and tagged containers, Docker, logging, swarm. Bookmark the permalink. Follow any comments here with the RSS feed for this post.

← Pimp my report with Docker Docker and Swarm Mode – Part 2 →

15 Comments LosTechies 1 Login

Recommend 1 Share Sort by Best

Join the discussion…

Konstantin • 5 months ago

Hi Gabriel,

Once again, thanks for this blog post series. There is a bit I'm struggling to get working - Private Regsitry. I'm using cloud-based servers instead of virtualbox for hosting my docker swarm nodes, but otherwise my setup is very similar to yours. I can deploy services to it and it works, for the most part, except for any communication with the published ports. So, when I deploy a private registry:

```
# docker service create --name registry --publish 5000:5000 registry:2
9qae3uigwimoc0khsrj7svpms
```

and then try to curl localhost:5000, the command just hangs indefinitely. I have a similar problem with the "whoami" example in part 2. Is there something I could be missing in my configuration?

Update: if I deploy the registry normally then it functions as expected

```
# docker run -d -p 5000:5000 --restart=always --name registry registry:2
# curl localhost:5000/v2/_catalog
```

see more

Reply • Share ›

gabrielschenker Mod → Konstantin • 5 months ago

Have you tried to use the loopback address 127.0.0.1 directly instead of localhost? See my note on this in part 4 of this series...

Reply • Share ›

Konstantin → gabrielschenker • 5 months ago

if anyone else is having similar issues, you can follow this: https://github.com/docker/d...

Reply • Share ›

gabrielschenker Mod → Konstantin • 5 months ago

Thanks for the reference

1 • Reply • Share ›

Konstantin → gabrielschenker • 5 months ago

thanks, but I think my problem is somewhat deeper: none of my services can reach any other services on exposed ports. e.g. if I deploy 2 services of nginx:

```
docker network create main --driver overlay
docker service create --network main --name foo1 nginx
docker service create --network main --name foo2 nginx
```

```
docker exec -it <foo1 container="" id=""> /bin/bash
# apt-get update && apt-get install curl -y
# ping foo2
PING foo2 (10.0.0.4): 56 data bytes
92 bytes from 8dee74043ee4 (10.0.0.3): Destination Host
Unreachable
# curl foo2
curl: (7) Failed to connect to foo2 port 80: No route to host
```

though this of course could also be 2 unrelated issues...

Reply • Share ›

Pranab Sharma ➦ Konstantin • 5 months ago

I am also having same problem, when I try to curl localhost:5000 while deploying registry as service, the command hangs.

• Reply • Share ›

gabrielschenker Mod ➦ Pranab Sharma • 5 months ago

see my reply above

• Reply • Share ›

Pranab Sharma ➦ gabrielschenker • 5 months ago

Thanks Gabriel, it worked using loopback address 127.0.0.1

• Reply • Share ›

Dominic Sondermann • 6 months ago

Hey, nice tutorial. You showed how to create a private registry and to tag and push an image to it. But what is with the pull from this private registry? Do you have to go to all nodes and pull it manually or is it possible to start a service and the nodes, where the task should run, pull the image from this private registry?

• Reply • Share ›

gabrielschenker Mod ➦ Dominic Sondermann • 6 months ago

when you create and run a service Docker will distribute as many containers as you required on the swarm and will have the respective node pull the image from the registry

• Reply • Share ›

Dominic Sondermann ➦ gabrielschenker • 6 months ago

My problem is, that on the nodes where I didn't pull the image manually from the private registry I get the following error (at docker service ps myservice):
ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR
xxxx _ myservice.1 regserver.com:5000/myservice worker2 Shutdown Rejected 12 seconds ago "No such image: regserver.com:..."

The call i did was:
docker service create --name myservice --network my-network --with-registry-auth -p 8761:8761 regserver.com:5000/myservice

Any ideas why the nodes can't automatically pull the image from my private registry?

• Reply • Share ›

Mursil Sayed • 7 months ago

Before the orchestration features introduced in Docker Engine 1.12, I remember Docker swarm documentation recommending using a separate service discovery app(etcd, consul, zookeeper) for production deployment. What Does Docker swarm mode recommend? Will the approach that you shared work in production env?

• Reply • Share ›

gabrielschenker Mod ➦ Mursil Sayed • 7 months ago

There is no more extra distributed key-value store like etcd or zookeeper needed anymore to keep track of the topology and state of the cluster. This data is directly maintained by the raft protocol of the master nodes

• Reply • Share ›

Ashish • 7 months ago

Nice tutorial for all docker machine/compose/swarm. As swarm is used to create

group of docker host. Question is
-After creation of multiple number of manager/worker in swarm group...what kind of application can be hosted by such group?
-Is docker swarm a distributed computing .i.e. performance of the application/service will be increase with number of workers?

Home
About Us
E-Books
Events
In Print
Topic of the Month
BY FEEDBURNER

Friends of Pablo
WatchMeCode
TekPub
St. Edward's Professional Education Center
Pragmatic Bookshelf
ReSharper - Develop with Pleasure!
NHProf

Pablo's Extended Family
CodeBetter
Devlicious
Dimecasts
ElegantCode