Docker & AnsibleForms

1 CONTENT

2 DOCKER

2.1 WHAT IS DOCKER

Docker is an application to run virtual machines from a command line.

Such a virtual machine is called a docker container.

The difference with VMware is that a container loses all the changes that were made to moment it gets updated or redeployed. That means we need a mechanism to make the data persistent. Using mounts from the mother-OS or external storage (NFS mounts) we can make changes and data persistent. The cool part about this is that data and code is 100% separated, making updates and backups extremely easy.

A docker container uses a docker image and is essentially an operating system, running on top of the mother OS. However with docker, the footprint is a lot smaller.

You will see that the words image and container are easily mixed. The image is however the source, the container is a containerized version of the image. But in terms of talking, we will probably mix them and say, “start an image”, “stop an image” while it’s essentially “stop a container from an image”.

2.2 DOCKER COMMANDS

Docker itself is a command (`docker`) and has a couple of subcommands

- `run` : run a docker image
- `start` : start a docker image
- `stop`: stop a docker container
- `image` : interact with docker images
- `ps` : show docker containers (running and stopped)
- `logs` : show docker logs
- `rm` : remove a container
- `inspect` : show docker container details
- `pull` : download a docker image from docker hub (=dockers repo)

2.2.1 docker run

This is the most basic action : it will download (if needed) the image, start a container, run a command, and stop the container after the command is finished.

```
[root@mother-OS]# docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Another very small docker image to play with is busybox. It's a **4MB**!! linux operating system with a bunch of utilities.

```
[root@mother-OS]# docker run busybox echo "hello from busybox"
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
809d8e20e203: Pull complete
```

```
Digest: sha256:2376a0c12759aa1214ba83e771ff252c7b1663216b192fbe5e0fb364e952f85c
Status: Downloaded newer image for busybox:latest
hello from busybox
```

Can you imagine this ? it downloaded the latest version of busybox, started the operating system, asked to run the echo command and stopped the operating system. Probably in 1ms !!

Important to realize is that docker images are NOT getting cleaned up, they are stopped but remain on the mother-OS.

2.2.2 docker ps

```
[root@mother-OS]# docker ps
CONTAINER ID   IMAGE         COMMAND                  CREATED          STATUS          PORTS           NAMES
```

Now the ps subcommand only shows running containers, however -a shows us all containers that are present

```
[root@mother-OS]# docker ps -a
CONTAINER ID   IMAGE         COMMAND                  CREATED          STATUS                      PORTS   NAMES
21e09db3d18c   busybox       "echo 'hello from bu…"   10 minutes ago   Exited (0) 10 minutes ago           
strange_maxwell
a9da69b129bd   hello-world   "/hello"                 12 minutes ago   Exited (0) 12 minutes ago           
exciting_fermat
```

you see that the hello-world and busybox containers are still there. Also notice that if a container is not given a specific name (option -name), docker gives it funny names like strange_maxwell and exciting_fermat (in red). It's easy to type and to spot.
This allows us to remove the container with a name. Note you can also use the container id (in blue)

2.2.3 docker rm

We can now remove the docker containers

```
[root@mother-OS]# docker rm 21e09db3d18c
21e09db3d18c
[root@mother-OS]# docker rm exciting_fermat
exciting_fermat
[root@mother-OS]# docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS                      PORTS
NAMES
```

the containers are removed (once using id and once using name)

2.2.4 docker image ls

`docker image` command interacts with the images. Above we removed the containers, but the images are still there.

```
[root@mother-OS]# docker image ls
REPOSITORY                          TAG          IMAGE ID       CREATED         SIZE
busybox                             latest       5242710cbd55   2 weeks ago     4.26MB
ansibleguy/AnsibleForms             beta         658ca0436736   4 weeks ago     1.33GB
ansibleguy/AnsibleForms             <none>       119d2219309d   4 weeks ago     1.33GB
ansibleguy/AnsibleForms             latest       13fd9e094f06   5 weeks ago     1.33GB
mysql                               5.7          8aea3fb7309a   3 months ago    455MB
mcr.microsoft.com/mssql/server      2019-latest  4885f6112d74   10 months ago   1.47GB
hello-world                         latest       feb5d9fea6a5   21 months ago   13.3kB
karelverhelst/AnsibleForms          latest       4c6a8d75a0b3   23 months ago   177MB
mariadb                             latest       fd17f5776802   23 months ago   409MB
busybox                             latest       c7c37e472d31   3 years ago     1.22MB
netapp/trident                      20.04.0      b5c3b90f1b51   3 years ago     141MB
```

2.2.5 docker image rm

To remove an image, use `rm`

```
[root@mother-OS]# docker image rm 5242710cbd55
```

```
Untagged: busybox:latest
Untagged: busybox@sha256:2376a0c12759aa1214ba83e771ff252c7b1663216b192fbe5e0fb364e952f85c
Deleted: sha256:5242710cbd55829f6c44b34ff249913bb7cee748889e7e6925285a29f126aa78
Deleted: sha256:feb4513d4fb7052bcff38021fc9ef82fd409f4e016f3dff5c20ff5645cde4c02
```

2.2.6 ALIASES

docker also uses aliases to make life easier, below 2 examples

docker images => docker image ls
docker rmi => docker image rm

```
[root@mother-OS]# docker rmi feb5d9fea6a5
Untagged: hello-world:latest
Untagged: hello-world@sha256:4e83453afed1b4fa1a3500525091dbfca6ce1e66903fd4c01ff015dbcb1ba33e
Deleted: sha256:feb5d9fea6a5e9606aa995e879d862b825965ba48de054caab5ef356dc6b3412
Deleted: sha256:e07ee1baac5fae6a26f30cabfe54a36d3402f96afda318fe0a96cec4ca393359
```

```
[root@mother-OS]# docker images
REPOSITORY                          TAG           IMAGE ID       CREATED         SIZE
ansibleguy/AnsibleForms             beta          658ca0436736   4 weeks ago     1.33GB
ansibleguy/AnsibleForms             <none>        119d2219309d   4 weeks ago     1.33GB
ansibleguy/AnsibleForms             latest        13fd9e094f06   5 weeks ago     1.33GB
mysql                               5.7           8aea3fb7309a   3 months ago    455MB
mcr.microsoft.com/mssql/server      2019-latest   4885f6112d74   10 months ago   1.47GB
karelverhelst/AnsibleForms          latest        4c6a8d75a0b3   23 months ago   177MB
mariadb                             latest        fd17f5776802   23 months ago   409MB
busybox                             latest        c7c37e472d31   3 years ago     1.22MB
netapp/trident                      20.04.0       b5c3b90f1b51   3 years ago     141MB
```

2.2.7 docker exec

Once a docker container is running a longer process (i.e. running a service application) you can still ask it to execute tasks, using the `exec` subcommand

```
[root@mother-OS]# docker exec AnsibleForms-docker_mysqldb_1 ls
bin
boot
dev
docker-entrypoint-initdb.d
entrypoint.sh
```

2.2.8 docker exec -it

More interesting is to interact with the container directly, using `-it` (interactive terminal)

```
[root@mother-OS]# docker exec -it AnsibleForms-docker_mysqldb_1 /bin/bash
bash-4.2# mysql -u root -p
mysql: [Warning] World-writable config file '/etc/mysql/my.cnf' is ignored.
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 18
Server version: 5.7.41 MySQL Community Server (GPL)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases
    -> ;
+--------------------+
| Database           |
+--------------------+
| information_schema |
```

```
| AnsibleForms       |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.01 sec)

mysql>
```

NOTE : we execute the bash-shell. `/bin/bash`
NOTE 2 : For containers based on the alpine linux base (like Ansible Forms), the shell is `/bin/ash`

```
[root@mother-OS]# docker exec -it AnsibleForms-docker_app_1 /bin/ash
/app #
/app # cd dist/persistent
/app/dist/persistent # cd playbooks
/app/dist/persistent/playbooks # ansible-playbook dummy.yaml
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not
match 'all'

PLAY [This is a hello-world example]
*******************************************************************************************************
***********

TASK [Gathering Facts]
*******************************************************************************************************
***********
ok: [localhost]
```

```
TASK [Output 'Welcome'.]
*********************************************************************************************************
***********
ok: [localhost] => {
    "msg": "Hi there and welcome to ansible"
}

PLAY RECAP
*********************************************************************************************************
***********
localhost                  : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

/app/dist/persistent/playbooks #
```

As you can see, the AnsibleForms container is fully equipped with ansible
you can run a playbook even in 1 command using the container

```
[root@mother-OS]# docker exec AnsibleForms-docker_app_1 ansible-playbook ./dist/persistent/playbooks/dummy.yaml
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: provided hosts list is empty, only localhost is available. Note that
the implicit localhost does not match 'all'

PLAY [This is a hello-world example] ******************************************

TASK [Gathering Facts] ********************************************************
ok: [localhost]

TASK [Output 'Welcome'.] ******************************************************
ok: [localhost] => {
    "msg": "Hi there and welcome to ansible"
}
```

```
PLAY RECAP *********************************************************************
localhost                  : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[root@worker-node2 playbooks]#
```

3 APPLICATION DOCKER IMAGES AND ANSIBLEFORMS

For now, we only played with containers that execute a single task and then stop. (`docker run`)

But real-life docker images have an entry-point that is a running service. As long as that service runs, the docker container stays up.

In the example of AnsibleForms, the docker image will start the web application (node js express), and to successfully do that, the application needs a lot of information.

For starters, it needs to know where the database is and how to connect to it, do we run http or https, where are the certificates, ... We do that by passing **environment-variables**.

```
docker run ansibleguy/AnsibleForms:latest -e VAR1=value1 -e VAR2=value2, ....
```

In the documentation you can see all possible environment variables, and I encourage you to have a look at them, as it might answer some questions you might already have.

https://ansibleforms.com/customization/

You can also assign volume mappings (`-v`) and port mappings (`-p`). This way you can connect parts of the mother-OS to the docker container. You can mount directories and files from the mother-OS into the container and export ports to the mother-OS. This way a docker container can run a web application that is accessible from outside the mother-OS. Note that the port internally in the docker image can be different that the port exposed on the mother-OS.

So AnsibleForms could be started this way, using a long docker run command passing all the environment variables, folder-mounts and port-mapping. But why stop here. Since AnsibleForms also needs a database, why not run the database as a container too.

But the database would again need environment variables like the root password. By now, you start realizing that spinning up containers running actual real-life applications, become long commands to enter. So why not use a configuration file, that nicely summarize all these port mappings, volume mounts and environment variables.

That's why they invented **docker-compose**.

4 DOCKER COMPOSE

the docker command interacts with 1 specific container. Real-life docker applications require a lot more parameters to start. Also, an application can consist out of more than 1 container and perhaps these containers have dependencies and internal interactions.

Docker compose allows us to describe a multi-container configuration.

Let's have a look at the docker-compose file for AnsibleForms.

You can download the docker-compose project from GitHub. Follow the instructions here : https://github.com/ansibleguy76/ansibleforms-docker

4.1 DOCKER-COMPOSE.YML

```yaml
version: '3.0'

services:
  # MySql Server
  mysqldb:
    image: mysql:5.7
    restart: unless-stopped
    # load extra environment variables from file
    env_file: ./.env
    # Set manual environment variables
    environment:
      - MYSQL_ROOT_PASSWORD=$MYSQLDB_PASSWORD
      - MYSQL_DATABASE=AnsibleForms
```

```
    ports:
      # Mount host port to docker internal port
      - $MYSQLDB_LOCAL_PORT:$MYSQLDB_DOCKER_PORT
    volumes:
      # Map database location (to maintain persistency)
      - ./data/mysql/db:/var/lib/mysql
      # Map my.cnf file (to maintain persistency)
      - ./data/mysql/my.cnf:/etc/mysql/my.cnf
      # Map init sql scripts
      - ./data/mysql/init:/docker-entrypoint-initdb.d
  # AnsibleForms application
  app:
    # Only start after MySql
    depends_on:
      - mysqldb
    image: ansibleguy/ansibleforms:latest
    restart: unless-stopped
    ports:
      # Mount host port to docker internal port
      - $WEBAPP_LOCAL_PORT:$WEBAPP_DOCKER_PORT
    # Load extra environment variables from file
    env_file:
      - ./.env
    # Set environment variables
    environment:
      - DB_HOST=mysqldb
      - DB_USER=$MYSQLDB_USER
      - DB_PASSWORD=$MYSQLDB_PASSWORD
      - DB_PORT=$MYSQLDB_DOCKER_PORT
      - PORT=$WEBAPP_DOCKER_PORT
    # allow interactive shell
```

```
    stdin_open: true
    # allow terminal
    tty: true
    volumes:
      # Mount application folder to host folder (to maintain persistency)
      - ./data:/app/dist/persistent
      # Mount images folder to host folder (to have custom images)
      - ./data/images:/app/dist/views/assets/images
      # Mount logo (to have custom logo)
      #- ./data/images/mylogo.svg:/app/dist/views/assets/img/logo_ansible_forms_full_white.svg
      # Mount background image
      #- ./data/bg.jpg:/app/dist/views/assets/img/bg.jpg
      # Map custom functions for js expressions and jq
      - ./data/functions/custom.js:/app/dist/src/functions/custom.js
      - ./data/functions/jq.custom.definitions.js:/app/dist/src/functions/jq.custom.definitions.js
      # Map custom sshkey to local node .ssh location
      - ./data/ssh:$HOME_DIR/.ssh
      - ./data/git/.gitconfig:$HOME_DIR/.gitconfig
```

Let's deep dive into this yaml file. You can see it contains 2 docker containers (services)

- **mysqldb** : MySQL 5.7
- **app** : AnsibleForms

Each service has a few basic sub sections :

- **image** : the docker image to run
- **env_file & environment** : the variables to load and set
- **ports** : port mappings (outside-port : inside-port)
- **volumes** : directory and file mapping (outside-path : inside-path)

The docker-compose project that you download from my GitHub repo is just a starter project. Feel free to adjust and set more environment variables.

NOTE : the docker-compose.yml file refers to an exteran .env file. The .env file contains the customer environment variables, include the database password. With docker compose that's about the most secure you can get to define a password. By loading it from the .env file, you can secure the .env file and by loading it from file, the content is not revealed in logs.

NOTE 2 : if you want to encrypt passwords you need to move away from docker-compose and move to docker swarm or Kubernetes, which have the option to create secrets (`docker secret create`)

4.2 DOCKER-COMPOSE UP

From within the directory where your docker compose file is (`docker-compose.yml`) , you can start with the command `docker-compose up`, however, you will see that this is in "interactive" mode, this is nice for debugging, but is not what you want.

4.3 DOCKER-COMPOSE UP -D

Run `docker-compose up -d` to start in **detached** mode.

```
[root@mother-os]# docker-compose up -d
Starting ansibleforms-docker_mysqldb_1 ... done
Starting ansibleforms-docker_app_1     ... done
[root@mother-os]#
```

4.4 DOCKER-COMPOSE DOWN

Run `docker-compose down` to stop

```
[root@mother-os]# docker-compose down
Stopping ansibleforms-docker_app_1     ... done
Stopping ansibleforms-docker_mysqldb_1 ... done
Removing ansibleforms-docker_app_1     ... done
Removing ansibleforms-docker_mysqldb_1 ... done
Removing network ansibleforms-docker_default
```

5 UPGRADING ANSIBLEFORMS

Upgrading AnsibleForms is as simple as typing these 3 commands.

```
[root@mother-os]# docker pull ansibleguy/ansibleforms:latest
[root@mother-os]# docker-compose down
[root@mother-os]# docker-compose up -d
```

6 CLEANING UP

Note that when you have upgraded ansibleforms, the older version is still on your system. Use the command your learned earlier to cleanup older versions (`docker image rm` or `docker rmi`)