Microsoft

Module 08: Volumes

ILDC Docker Workshop

1

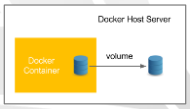
Agenda

- Docker Volumes
- Lab 08: Using Volumes
- Lab : Mount to Local directory

2

Docker Volumes

- When a container dies all the data it has created (logs, database records, etc.) dies with it (and remember containers are ephemeral).
- Volumes are external storage areas used to store data produced by a Docker container.
- Volumes can be located on the Docker host or even on remote machines



3

Docker Volumes

- By default, volumes are not deleted when the container is stopped.
- Data volumes can be shared across containers.
- Volumes could be mounted in read-only mode.
- **--volume:** Create a file or directory if it doesn't exist on the Docker host
- **--mount:** Does not automatically create it for you, but generates an error

4

Docker Volumes - Syntax

```
$ docker run [OPTIONS] -v "volume_name:/container/path" [IMAGE]
```

- Optionally you can create a volume with a default name:

```
$ docker run [OPTIONS] -v "container/path" [IMAGE]
```

5

Docker Volumes – Dockerfile

VOLUME

- Create a new volume with any data that exists at the specified location within the base image.
- Anything after the VOLUME instruction will not be able to make changes to that volume.

```
FROM microsoft/iis
RUN powershell -NoProfile –Command
Remove-Item -Recurse
C:\inetpub\wwwroot\*
WORKDIR /inetpub/wwwroot
COPY . .
VOLUME c:/inetpub/wwwroot
EXPOSE 80
```

6

Docker Volumes – Managing Volumes

- Create a volume:

```
$ docker volume create volume-name
```

```
$ docker volume create demo-volume
demo-volume
```

- List volumes:

```
$ docker volume ls
```

```
$docker volume ls
DRIVER          VOLUME NAME
local           demo-volume
local           ed702f0a2c8b6ceb56...
local           my_volume
```

7

Docker Volumes – Managing Volumes

- Inspect a volume:

```
$ docker volume inspect volume-name
```

```
$ docker volume inspect demo-volume
[
    {
        "CreatedAt": "2018-06-07T23:39:20+03:00",
        "Driver": "local",
        "Labels": {},
        "Mountpoint":
"/var/lib/docker/volumes/demo-volume/_data",
        "Name": "demo-volume",
        "Options": {},
        "Scope": "local"
    }
]
```

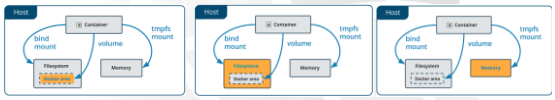
- Remove a volume:

```
$ docker volume rm volume-name
```

```
$ docker volume rm demo-volume
demo-volume
```

8

Volume vs. BindMounts vs. tmpfs



Volume BindMounts tmpfs

- Volumes are the preferred way to persisting data since they do not depend on the host filesystem structure
- BindMounts :mounts a file/directory on the host system into the container.
- tmpfs only works on Linux and cannot be shared between containers

9

Lab 08: Using Volumes

Lab

Lab-08.md - Repos (azure.com)

10

Challenge 03 – Mount to Local Directory

Lab

Challange-Volumes.md - Repos (azure.com)

11

Microsoft

Module 09: Working with Registries

ILDC Docker Workshop

1

Agenda

- Docker Registries
- Lab 10: Pushing images to Docker Hub
- Working with Azure Container Registry (ACR)

2

Create Docker-hub account

Demo

3

Docker Registries

- Used to share and manage Docker images
- Docker Hub is the default registry
- There are also private registries (Artifactory, GCP, Azure, AWS, etc)

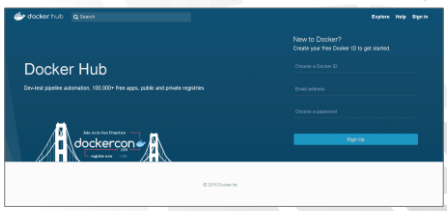
4

Pushing images to the Docker Hub

1. Create an account (free)
2. Create a new repository
3. Tag the image following the convention
4. Login to Docker Hub
5. Push the image

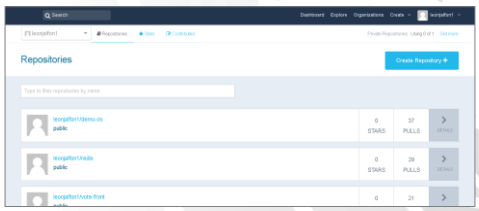
5

Push to Docker Hub – Create an account



6

Push to Docker Hub – Create a repository



7

Push to Docker Hub – Tag the Image

- To create a new tag for the image use:

```
$ docker tag <image-id> <username>/<repo-name>:<tag>
```

- The name of the image must follow the format:

  <username>/<repository>:<tag>

8

Push to Docker Hub – Login

- Login using the Docker Hub credentials

```
$ docker login -u <username> -p <password>
```

9

Push to Docker Hub – Push the Image

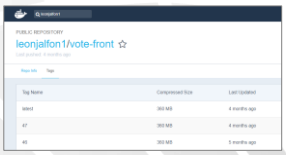
```
$ docker push <username>/<repo-name>:<tag>
```

- Then the image will be available in the docker hub
- You can see all the version pushed to the repository in the repository page

10

Pulling images from the Docker Hub

1. Login (for private repos)
2. Pull the image




```
$ docker pull <username>/<repo-name>:<tag>
```

11

What can we do with all this information

- by now you know:
  - To write your app (da)
  - To "dockerize" it (docker build)
  - To run the container , control it and stop it
  - To enhance it with volumes and network
  - To control which app to run according to the label system.
  - Where does it come handy in a developer /product workflow ?

12

CI / CD

- Continuous Integration (CI):
  - a practice where you merge the work of all your developers several times a day. Each integration is verified by your automated tests and built (when it needs to be, for compiled languages mostly).
  - Why is it hard to perform CI ?
    - several branches / merge conflicts
    - Dependencies
    - Versions
    - Different tools / languages / platforms and libraries
    - Unless done effectively can slow down the dev cycle and hurt the release timeline.
- Continuous Delivery (CD) :
  - A practice where the team produce the software (release) at any time.
  - Aims for building /testing and releasing the software with greater speed and frequency

13

Using Docker with Azure Container Registry (ACR)

Demo

14

Lab 10: Working with Registry (Docker Hub/ACR)

Lab

Lab-10.md - Repos (azure.com)

15

Microsoft

Module 10: Docker-Compose

ILDC Docker Workshop

1

Agenda

- Basic Docker Overview
- Docker compose Introduction
- Container Orchestration Introduction
- Architecture Considerations

2

Building Containers

- FROM
- MANTAINER
- RUN
- CMD
- EXPOSE
- ENTRYPOINT
- ENV
- COPY
- ADD
- WORKDIR
- VOLUME

```
FROM ubuntu:14.04
RUN \
  apt-get update && \
  apt-get -y install apache2

VOLUME /myvol
ADD index.html /var/www/html/index.html

EXPOSE 80

CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

3

Managing Containers

- $ docker run
- $ docker ps
- $ docker images
- $ docker rm
- $ docker rmi
- $ docker attach
- $ docker exec
- $ docker save/load
- $ docker commit
- $ docker stop
- $docker kill
- $ docker start
- $ docker restart
- $ docker info
- $ docker top
- $ docker history
- $docker inspect
- $ docker logs

4

Docker Registries

```
$ docker login -u <username> -p <password>
```

```
$ docker push <username>/<repo-name>:<tag>
```

```
$ docker pull <username>/<repo-name>:<tag>
```

5

Containers Need Management

6

What is docker compose?

- Compose is a tool for defining and running complex applications with Docker.
- Define a multi-container application in a single file.
- Spin your application up in a single command.
- Docker-compose uses a YML file called "docker-compose.yml"

7

docker-compose.yml

```
version: '3'
services:
  db:
    image: postgres:9.4
    labels:
      io.skopos.singleton: '1'
      io.skopos.visual.position: 900,380
  nginx:
    depends_on:
    - vote
    deploy:
      replicas: 1
    image: datagridsys/sample-lb:1.0
    labels:
      io.skopos.dt.vote: _
      io.skopos.lb.name: vote-in
      io.skopos.lb.position: 20,80
      io.skopos.singleton: '1'
      io.skopos.visual.position: 300,80
    ports:
    - 8888:80
```

```
[root@terry dockerapp]# docker-compose build
redis uses an image, skipping
Building dockerapp
Step 1/6 : FROM python:3.5
 ---> b0d7fc8a7ace
Step 2/6 : RUN pip install Flask==0.11.1 redis==2.10.5
 ---> Using cache
 ---> b463f1061f34
Step 3/6 : RUN useradd -ms /bin/bash asfye
 ---> Running in 80c6015b840a
 ---> 38bd504db65b
Removing intermediate container 80c6015b840a
Step 4/6 : USER asfye
 ---> Running in 4e03cf70f3b2
 ---> edc9013d502a
Removing intermediate container 4e03cf70f3b2
Step 5/6 : WORKDIR /root/dockerapp/app
 ---> cc080b462c85
Removing intermediate container 9f6441f3906c
Step 6/6 : CMD python app.py
 ---> Running in 8ab5f65de135
 ---> 1df04b7f00d1
Removing intermediate container 8ab5f65de135
Successfully built 1df04b7f00d1
Successfully tagged dockerapp_dockerapp:latest
[root@terry dockerapp]#
```

8

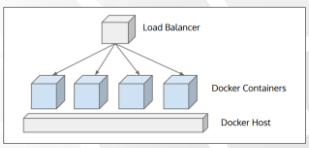
Lab 11: Docker Compose

Lab

9

Problems with Standalone Docker

- Running a server cluster on a set of Docker containers, on a single Docker host is vulnerable to single point of failure!



10

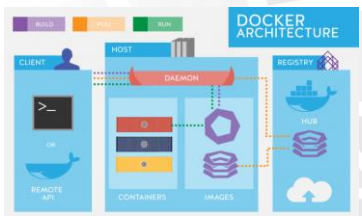
Docker Orchestrators

- Scalability
- Service Discovery
- Networking
- Volume Management
- Monitoring and Logging
- High Availability
- Load Balancing
- Health Checks
- Rolling Upgrades

kubernetes

11

Docker on the Cloud



12

Architecture Considerations

- In 2011 , Adam Wiggins , the founder of Heroku, published an article called "the twelve factor app" https://12factor.net/
- The "12-factor-app" describes 12 practices to follow when designing and architecting Software-As-A-Service (SaaS) environment.
- Those guidelines are ideal for the docker

13

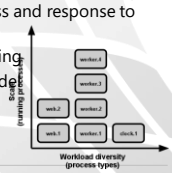
12-Factor-App

- Codebase – a single codebase tracked in revision control (each and every docker image should be build from a single source code repository)
- Dependencies – Explicitly declare and isolate dependencies
- Config – Store config in Evn. Variables not in files checked into the code base (e.g –e App_ENV=production). Keeping the configuration out of the source code helps deploy the same container to multiple environments
- Backing Services- treat backing services as attached resource

14

12-Factor-App

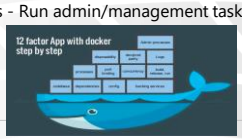
- Build, release, run –Strictly separate build and run stages.
- Processes – Execute the app as one or more stateless processes It is preferred to write an application that do not need to keep state longer than the time it takes to process and response to single request.
- Port Binding – Export services via port binding
- Concurrency - Scale out via the process model
- Disposability -Maximize robustness with fast startup and graceful shutdown



15

12-Factor-App

- Dev/Prod Parity – Keep Development, Staging and Production as similar as possible
- Logs – Treat Logs as event Streams . Services should not concern themselves with routing or storing logs. Events should streamed to STDOUT.
- Admin Processes - Run admin/management tasks as one-off processes.



16

Where To Go Next ?

- Experiment with docker with small scale on your laptop
- Gain a stronger understanding on how the pieces fit together
- Move to the cloud (AWS / Azure / Google all have free tiers)
- Learn how Kubernetes and Orchestration tool can assist you in more complex scenarios such as Service Discovery, Deployment sets, Load balancing etc.
- Spread the word.....

Happy Development !

17