Docker

Alexander Zerpa

Section 1

Docker

Docker

Docker is a platform for containerizing applications. Containers run on the kernel, isolated from other processes and tend to have better performance than virtualisation.

Containers vs Virtual Machines

Containers

- Run in runtime
- Alongside OS
- Not OS configuration
- Usually one app at a time

Virtual Machines

- Run on hypervisor
- Hardware emulation
- Require OS configuration
- Many apps at once

Architecture

Client

Tool for interacting with the docker system.

Daemon

Principal process that listens for the API and manages images, containers, networks and volumes.

Registry

Where images are stored, Docker Hub is the primary repository.

Section 2

Objects

Subsection 1

Images

Images

Blue-print for constructing the container.

Making Images of Containers

docker commit <container> <image>

Listing Images

docker images

Renaming Images

docker tag <image> <repo>:<tag>

Removing Images

docker rmi <repo>:<tag>

Saving Images

docker save -o <arch>.tar.gz <images>

Loading Images

docker load -i <arch>.tar.gz

NOTE: When <tag> is not specified latest is used.

Building Images

When building, docker caches each step. Every line is run independently.

Dockerfile Instructions		
Directive	Description	
FROM	Base image	
COPY	Copy from build context	
RUN	Execute command	
CMD	Cmd for container to run	
ENTRYPOINT	Start of command	
ENV	Set environment variable	
EXPOSE	Maps ports	
VOLUME	Defines volumes	
WORKDIR	Set working directory	

Dockerfile Example

FORM img

COPY src trg

Building

docker build -t <repo>:<tag> <path>

Multi-stage builds

Use multiple FROM statements for different stages of the build process, copying form pass stages only what you want in the final image.

Remote Images

If docker cant find a Image locally it will try to pull it from the official repo.

Sharing Images

- Create an account on the official docker repository.
- Select "Create Repository +".
- Fill at least the name field on the form.
- Connect the docker client.
- Build image with namespace and repository name.
- Upload image.

Downloading Images

docker pull <repo>:<tag>

Connect docker client

docker login

Upload Images

docker push <namespace>/<repo>:<tag>

By default <namespace> is the same as username.

Subsection 2

Containers

Containers

Runnable instance of an image. A container can be referenced by id or name.

Namespaces

Different views of system.

Namespace	Description
USERNS	User list
MOUNT	Access to file system
NET	Network communication
IPC	Interprocess communication
TIME	Change time (not supported)
PID	Process ID management
CGROUP	Create control groups
UTC	Create host/domain names

Control groups

Restrict resources a container can use.

Command names

New
docker container run
docker container start
docker container stop
docker container rm
docker container inspect
docker container exec

Managing Containers

Creating Containers

creates an image
docker container create <image>

creates an image with set name

docker container create <image> --name <name>

Starting a Container

docker start <container>

Stopping a Container

docker stop <container>

Listing

shows running containers
docker container ls

shows all containers

docker container ls -a

Kill Container

Similar to docker stop but send SIGKILL instead of SIGTERM.

docker kill <container>

Removing a Container

docker rm <container>

Pausing Containers

docker pause <container>

Running Images

Run Options

Option	Description
-t	Allocate a pseudo TTY
-i	For interacting with console
-d	Run container in background
-e	Sets environment variables
-V	Bind mounts a volume
-p	Links container and host ports
-rm	Removes container on exit
-name	Set container name
-net	Specify network to connect to
-mount	Attach filesystem mount

Run Command

docker run <repo>:<tag>
run = create + start + attach.

Resource Constraints

Memory

docker run --memory <bytes> <image>

CPU

```
# relative to other containers
docker run --cpu-shares <num>
# limit CFS quota
docker run --cpu-quota
```

Interacting with a Container

Executing Commands

```
# run command on container
docker exec <container> <cmd>
# specifi a working directory
docker exec -w <path> <container> <cmd>
```

Attaching to Container

You can exit a container without stopping it with P Q .

docker attach <container>

Logging

docker log <container>

Legacy Linking

- Connects all ports
- Only one way
- Same with Secret environment variables
- Depends on startup order

docker run --link <container> <image

Port Information

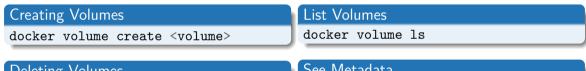
docker port <container>

Subsection 3

Volumes

Volumes

Persistent data for containers.



Deleting Volumes

docker volume rm <volume>

See Metadata

docker volume inspect <volume>

Backups

Backup

```
docker run --rm -v /tmp:/backip \
    --volumes-from <container-name> \
    busybox tar -cvf /backup/backup.tar <path-to-data>
```

Restore

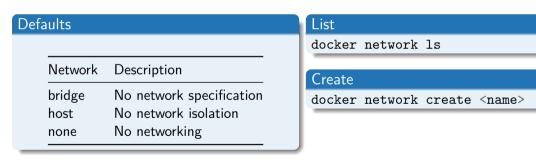
```
docker run --rm -v /tmp:/backup \
    --volumes-form <container-name> \
    busybox tar -xvf /backup/backup.tar <path-to-data>
```

Subsection 4

Networks

Networks

Virtual connections between containers and external devices.



Connections

Connecting Containers

```
# conects container to network
docker network connect <network> <container>
# disconects container form network
docker network disconnect <network> <container>
```

Listing Connections

```
# list container on network
docker network inspect <network> -f "{{json .Containers }}"
# list networks a container is attached to
docker inspect <container> -f "{{json .NetworkSettings.Networks }}"
```

Section 3

Docker Compose

Docker Compose

Docker configuration as code.

Designed for:

- Local development
- Staging server
- Continuous integration testing environment

For production environments use clustering tools like kubernetes.

V1 vs V2

V2 is integrated into docker cli platform and let's you use shared flags on the root docker command.

Service container names

V1 uses _ as word separator and V2 uses -.

--compatibility or COMPOSE COMPATIBILITY to set V2 word separator as _.

Unsupported Command-line flags and subcommands

- docker-compose scale. Use docker compose up --scale.
- docker-compose rm --all.

Commands

Starting

```
# build create and start containers
docker compose up
# for spesific steps
docker compose build
docker compose create
docker compose start
# start only service and dependencies
docker compose up <service>
```

Stopping

```
# stop and delete services
docker compose down
# same as down
docker compose stop
docker compose rm
```

Restarting

same as stop then start
docker compose restart

Subsection 1

Structure

Services

- Configuration to be applied to each service container.
- Can be build or use and existing image.

Configuring Images

Configurations and arguments depend on the image. Read image documentation to know what to use.

```
Build Args
```

Environment Variable

No value passes the host variable

env file

Volumes

```
# deletes named volumes
docker compose down --volumes
```

Syntax

short syntax

<src>:<target>:<mode>

Long syntax

type: volume
source: <src>
target: <target>

read_only: (true|false)

nameless

serives:

<service>:
 volumes:

umes:

- <src>:<target1>:<mode>

- <target2>

If no src> docker makes volume
automatically.

<mode> can be rw (defaul) or ro.

named

volumes:

<volume>:

Can use <volume> instead of path in <scr>.

Ports

NOTE: Port protocol can be declared with port/protocol.

Start Options

Startup Order

Starts and stops on dependency order.

services:

```
<service>:
```

depends_on:

- <other-service>

Starting service by name also starts its dependencies.

docker compose up <service>

Service Profiles

If not profile specified, it is included in default and starts with every other service profile.

services:

```
<service>:
```

```
profiles:
```

- cprofile>

```
## run only defualt profile services
docker compose up
## run only prifile services
docker compose --profile cmd>
```

Multiple Compose File

- Distinct desired behaviors that do no coincide
- Different environments

docker compose reads from docker-compose.ymal and docker-compose.override.yaml, merging its contents with preference to override.

docker compose -f docker-compose.yaml -f docker-compose.<override>.yaml <cmd>

distinct overrides

Replace override in file name.

docker-compose.<name>.ymal

NOTE: first field of -f doesn't need to be docker-compose.yaml.

Environment Variables

Use \${VAR} to replace within the docker file.

Default

- \${VAR:-default}: VAR if set and not-empty, otherwise default.
- \${VAR-default}: VAR if set, otherwise default.

Required

- \${VAR:?error}:VAR if set andnot-empty, otherwiseexit with error.
- \${VAR?error}: VAR if set, otherwise exit with error.

Alternative

- \${VAR:+replacement}: replacement if VAR is set and not-empty, otherwise empty.
- \${VAR+replacement}: replacement if VAR is set, otherwise empty.

Variable defaults

docker compose automatically use declaration in the shell, variables form .env file or in:

docker compose --env-file <path>

Section 4

WordPress with MariaDB

Subsection 1

Using Docker

Database Container

Creating Volume

docker volume create wordpress-db

Creating Container

```
docker run -d --name wordpress-db \
    --mount source=wordpress-db,target=/var/lib/mysql \
    -e MYSQL_ROOT_PASSWORD=secret \
    -e MYSQL_DATABASE=wordpress \
    -e MYSQL_USER=manager \
    -e MYSQL_PASSWORD=secret \
    mariadb:10
```

Wordpress Container

Working Space

For editing files and modifying behaviour.

mkdir -p Sites/wordpress/target && cd Sites/wordpress

Running docker

```
docker run -d --name wordpress \
    --link wordpress-db:mysql \
    --mount type=bind,source="$(pwd)"/target,target=/var/www/html \
    -e WORDPRESS_DB_USER=manager \
    -e WORDPRESS_DB_PASSWORD=secret \
    -p 8080:80 \
    wordpress:6
```

Subsection 2

Using Docker Compose

Using Docker Compose

```
Compose File
services:
                                           volumes:
    db:
                                               - ./target:/var/www/html
                                           environment:
         image: mariadb:10
                                               - WORDPRESS_DB_USER=manager
    volumes:
         - data:/var/lib/mysql
                                               - WORDPRESS_DB_PASSWORD=secret
    environment:
                                               - WORDPRESS DB HOST=db
         - MYSQL ROOT PASSWORD=secret
                                               - WORDPRESS DB NAME=wordpress
         - MYSQL DATABASE=wordpress
                                           ports:
         - MYSQL USER=manager
                                               - 8080:80
         - MYSQL PASSWORD=secret
    web:
                                       volumes:
         image: wordpress:6
                                           data:
    depends_on:
                                      Start Service
         - db
                                       docker compose up -d
```