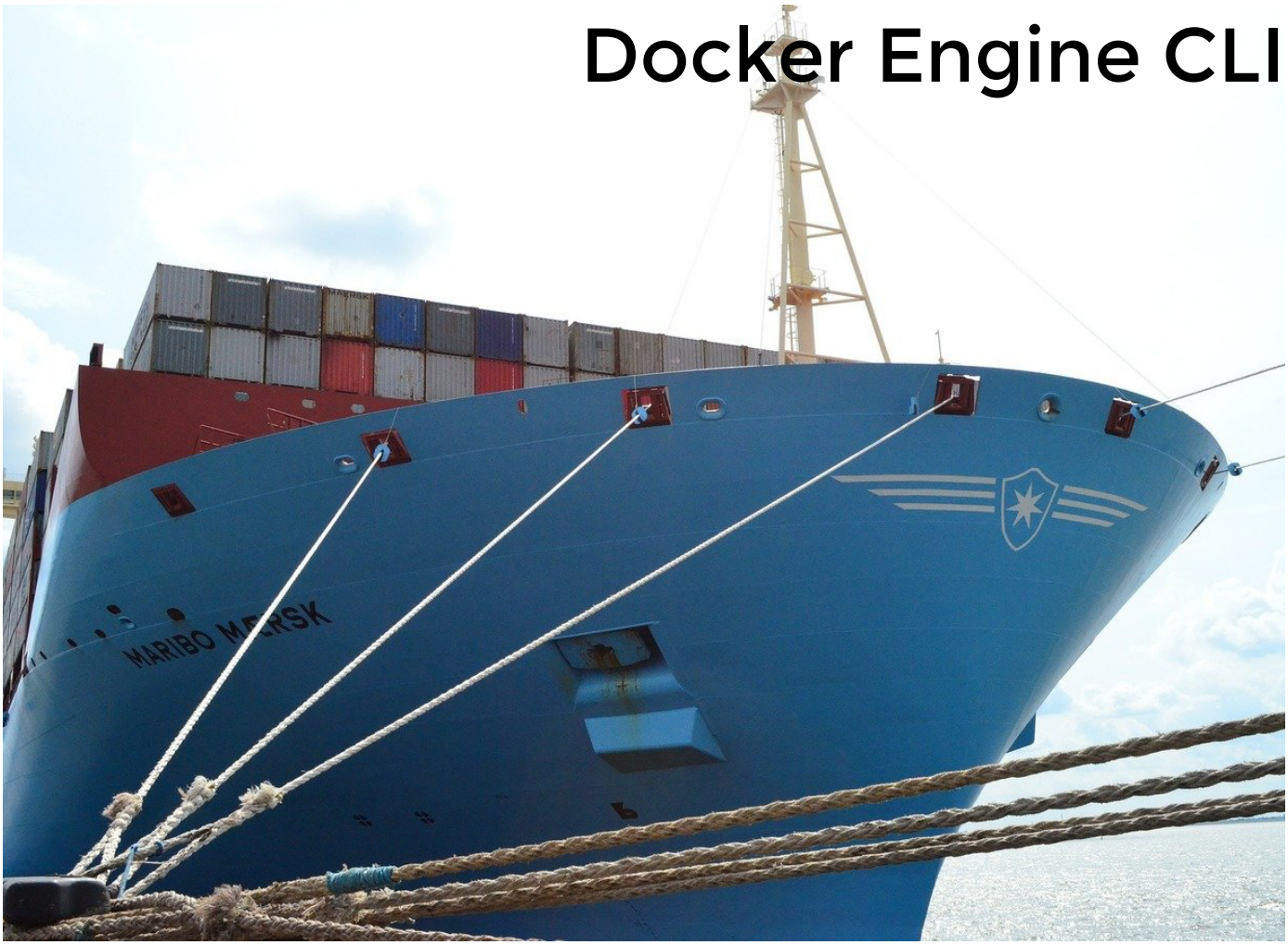
Agenda

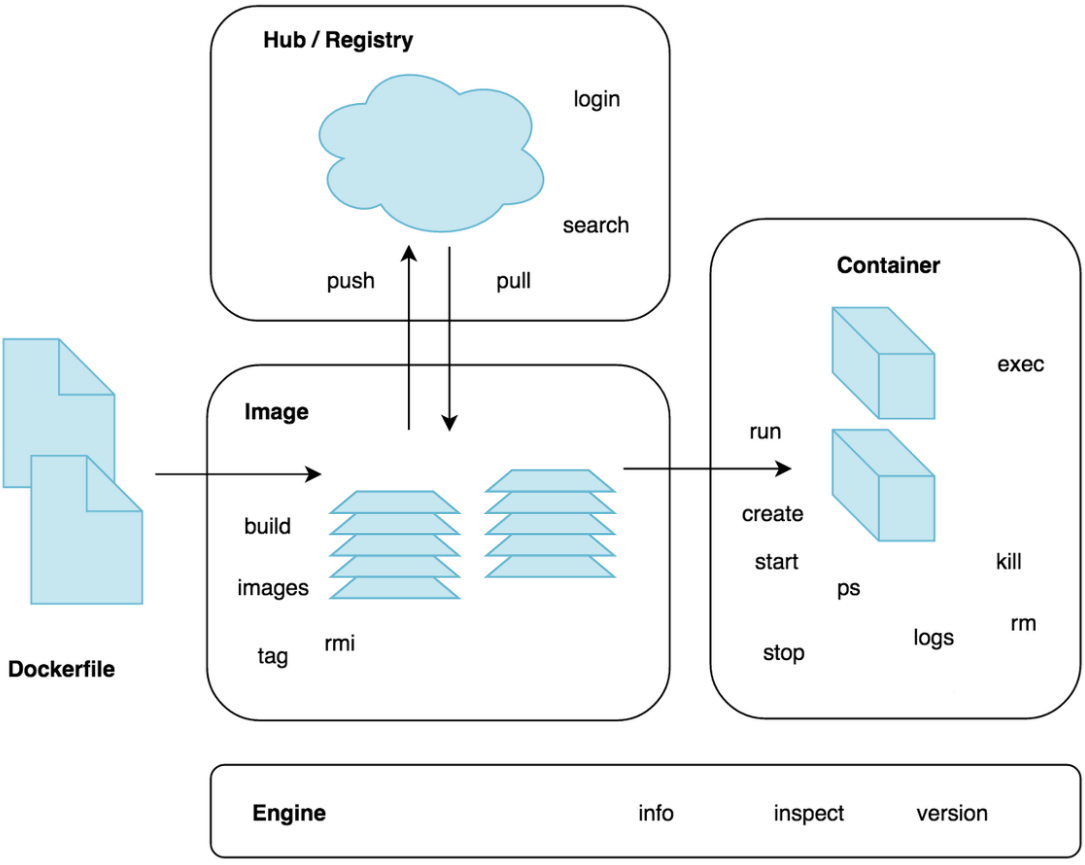
Docker Engine CLI (Recap of Day 1)
Docker Plattform (Recap of Day 1)

Multicontainer Application
Docker Compose
Docker Swarm

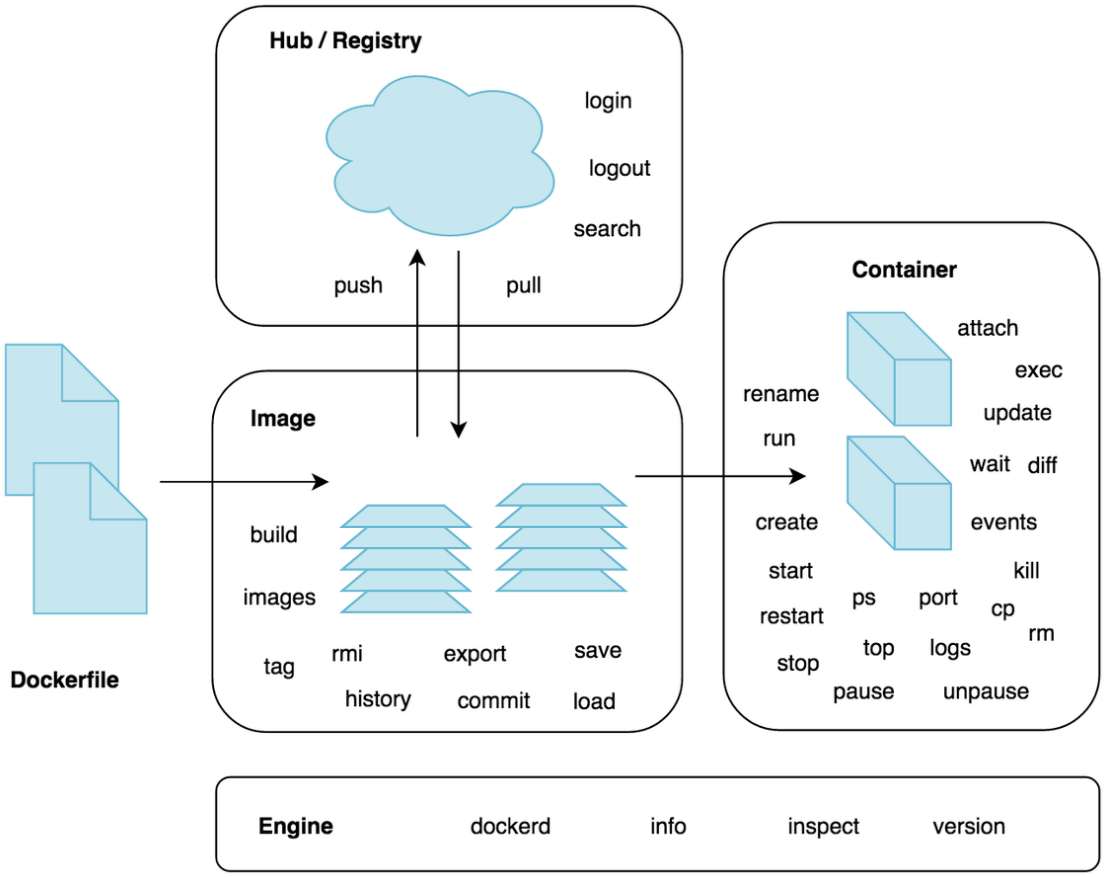
Docker Engine CLI



Commands



All Commands



Solutions

Docker Engine (free/commercial)
Docker Compose

Docker Machine
Docker Swarm

Docker Hub (service, public/private)
vs. Docker Registry (free, CLI only)
vs. Docker Trusted Registry (enterprise)
Docker Store (beta, commercial)

Docker Cloud (service, depends on providers)
vs. Docker Datacenter with DTR & UCP (enterprise)

Running a multicontainer app with Docker - 1

Start a mysql container:

1. Use version 5.17
2. Give the container a name
3. Inject env vars:
   MYSQL_ROOT_PASSWORD=somewordpress
   MYSQL_DATABASE=wordpress
   MYSQL_USER=wordpress
   MYSQL_PASSWORD=wordpress

Running a multi-container app with Docker - 2

Start a wordpress container:

1. Use version latest
2. Give the container a name
3. Link the mysql container
4. Inject env vars:
   WORDPRESS_DB_HOST=<link-name>:3306
   WORDPRESS_DB_USER=wordpress
   WORDPRESS_DB_PASSWORD=wordpress

Docker Compose

Overview

"Compose is great for development, testing, and staging environments, as well as CI workflows"

- fig was once acquired by Docker in 2014
- Define and run multi-container apps

Docker Compose

Use Cases

Development environments:

- running web apps in an isolated environment is crucial
- the Compose file allows to document service dependencies
- multi-page “developer getting started guides” can be avoided

Automated testing environments

- create & destroy isolated testing environments easily

Single host deployments

- deploy to a remote Docker Engine or Swarm cluster
- production-readiness of Docker Compose is coming

Docker Compose

Features

Multiple isolated environments on a single host

A project name can used to isolate environments, e.g.:

Dev host: multiple copies of single env (= feature branch names)

CI server: do not allow builds to interfer (= unique build number)

Shared host: prevent different projects which may use same service names from interfering with each other

Docker Compose

Features

Preserve volume data when containers are created

When *docker-compose up* runs it finds any containers from previous runs and copies the volumes from the old container to the new container

Only recreate containers that are changed

When a services restarts and nothing has changed Compose re-uses existing containers because it caches the configuration that was used to create a container

Docker Compose

Features

Variables and moving a compostion between environments

Variables in the Compose file can be used to customize your composition for different environments or users.

Inheritance is support by using the extends field or by creating multiple Compose files.

Docker Compose

Features

Variables and moving a compostion between environments

Variables in the Compose file can be used to customize your composition for different environments or users.

Inheritance is support by using the extends field or by creating multiple Compose files.

Docker Compose

3-Step-Process

1. Define your app environment via *Dockerfile (if needed)*

2. Define services in *docker-compose.yml* file

3. Run *docker-compose up*

Compose File

Version 3

```
version: '3'

services:
   <service-1>:
     image: <image>
     restart: <retsart policy>
     environment:
       <List of env variable>

   <service-1>:
     depends_on:
       - <service-1>
     image: <image>
     ports:
       - <ports to be exposed>
```

Writing an own compose file

1. Finalise the given docker-compose.yml file
2. Run `docker-compose up`
    1. Open localhost:8000
3. Are the containers deleted after finishing the process?
4. Run the environment in background
    1. Tear down the services
    2. Are the containers now deleted?
5. Start only one service in background

Persisting running containers - 1

1. Start the application and sign in
2. Is your account stored after restart?
3. Mount a volume into the db container to persist data
    1. Use a local folder
    2. The folder to be mounted is `/var/lib/mysql`
4. Start the application and sign in
5. After restart the data should not get lost

Persisting running containers - 2

1. Define a volume in the docker-compose.yml
2. Replace the local folder with the volume name
3. Start the application
    1. There should be no data loss
4. List all volumes and inspect the newly created
5. Tear down the application
6. Delete the volume
7. Start the application
    1. Is your data still available?

docker-compose CLI

Reference

- build
- bundle
- config
- create
- down
- events
- exec
- help
- images
- kill
- logs
- pause
- port
- ps
- pull
- push
- restart
- rm
- run
- scale
- start
- stop
- top
- unpause
- up

Maintain your environment

1. Pause and unpause your application
2. Pull all needed images
3. Scale the db service to 3 instance
    1. Step into db service
    2. In which container did you step?
4. List all running docker-compose containers
5. Tear down environment and delete all images and volumes in one step
6. Stop and remove wordpress service
7. Create and start wordpress service

Debug your environment

1. Have a look at all logs
    1. Are you able to only display the logs of one service
    2. Also follow the logs
2. Step into Wordpress service
    1. Kill all apache processes `kill $(pgrep -f apache)`
    2. Is the container restarted?
3. Retrieve the public port of the wordpress service for a port binding
4. Restart the wordpress service

Running from local development

1. Write docker-compose file
2. Remove all flask-app images
3. Start the docker-compose environment
    1. What do you recognize?
    2. Can you also build the image with docker-compose?
    3. How is this image named?
4. Can you change the image name and tag?
5. Push the newly created image
    1. This is only possible if you have an account and if you are logged in

Docker Swarm

Sandbox

www.play-with-docker.com

Creating a Docker Swarm

1. Go to the playground
2. Add several nodes
3. Init a swarm on the first node
4. Add all other nodes to the cluster

Deploy to docker swarm

1. Init a docker swarm locally
2. Can you deploy the current docker-compose config?
    1. `docker stack deploy --compose-file=docker-compose.yml <stack-name>`
    2. Try to access the website
3. Extend the compose file
    1. Add deploy section with replicas
    2. Update the stack
    3. Test the resilience

Further commands

1. Scale the service via the command line
2. Watch the logs of a service
3. Try to apply your knowledge :)

Summary Professional Training Day 2