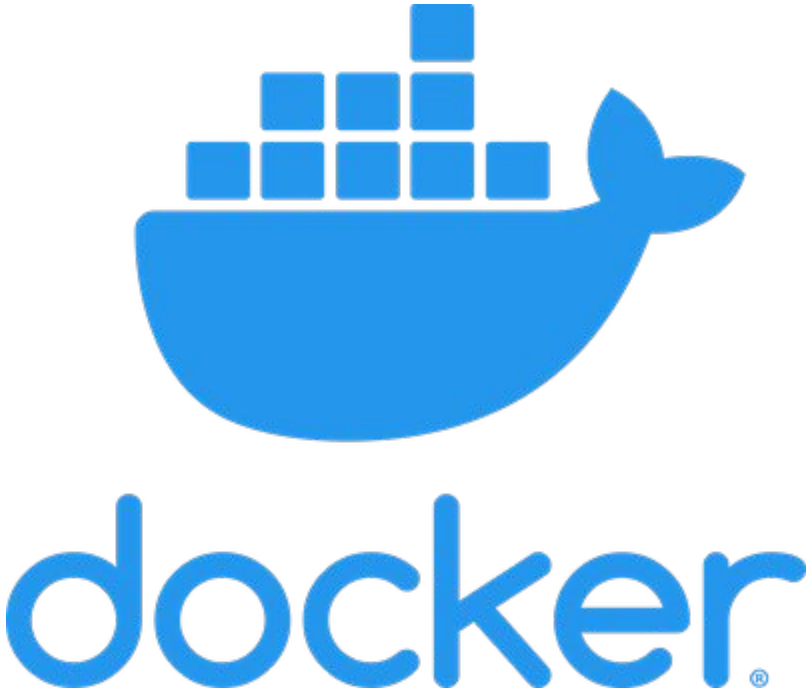
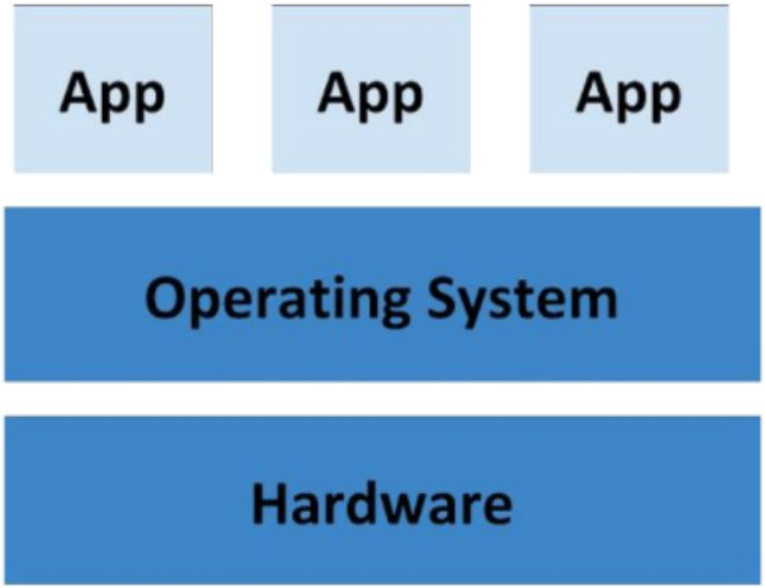

docker

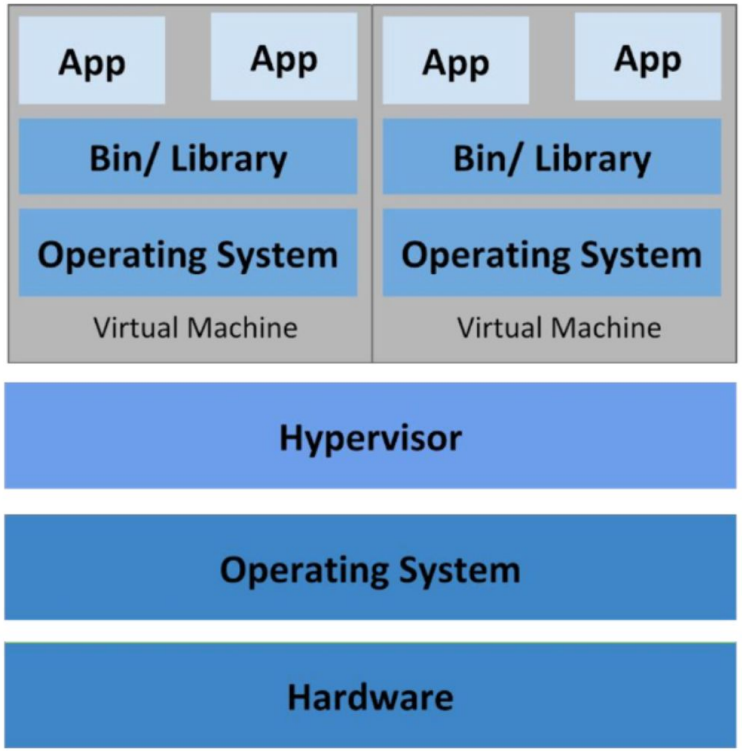
Agenda

- Introduction
- Basic of Docker
- Building container
- Running web application with Docker
- Docker compose

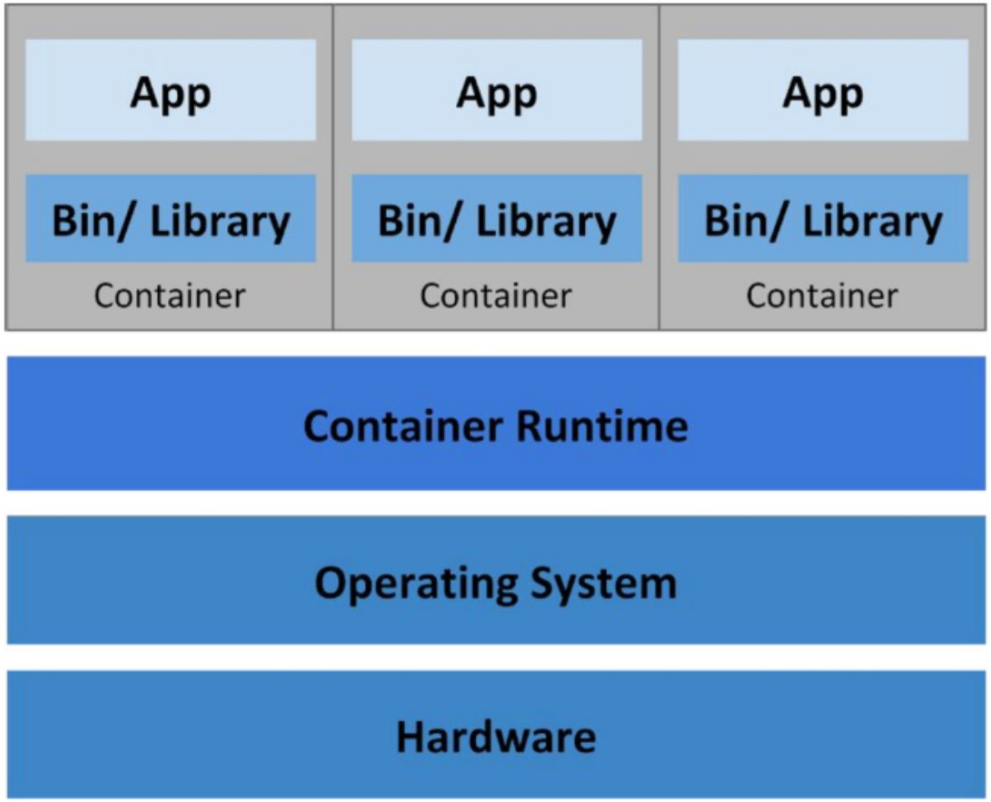
Introduction



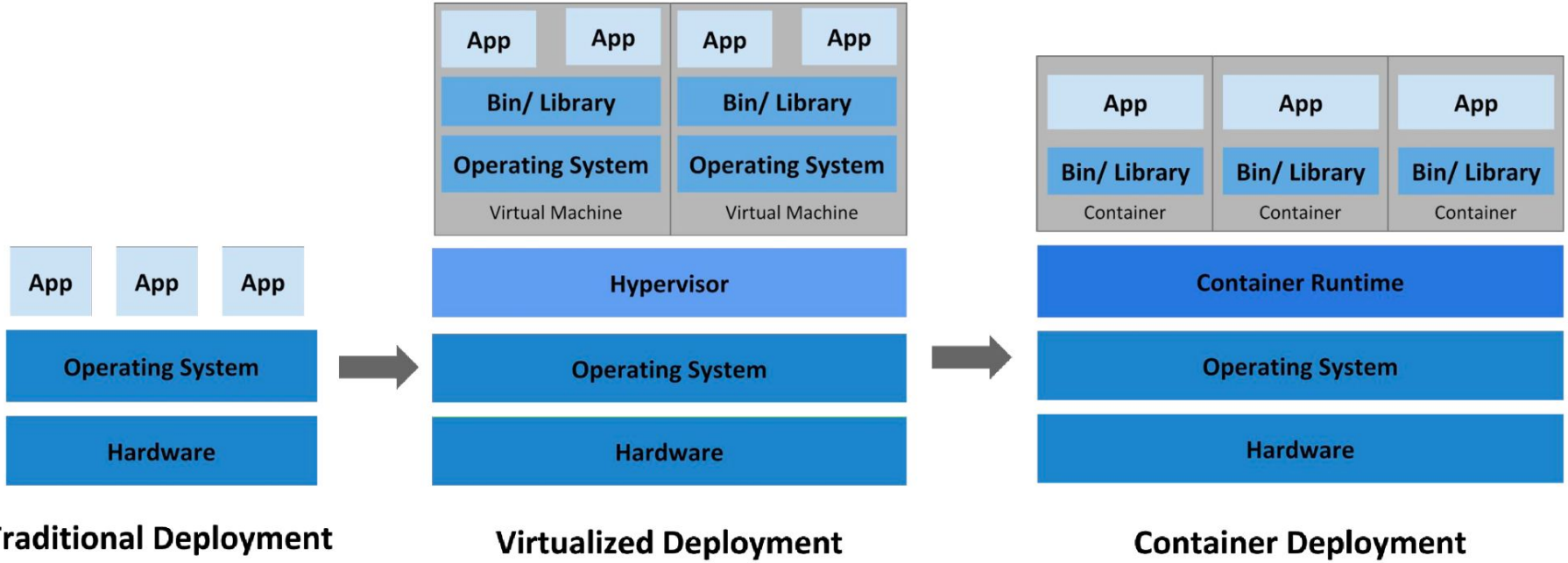
Traditional Deployment



Virtualized Deployment



Container Deployment

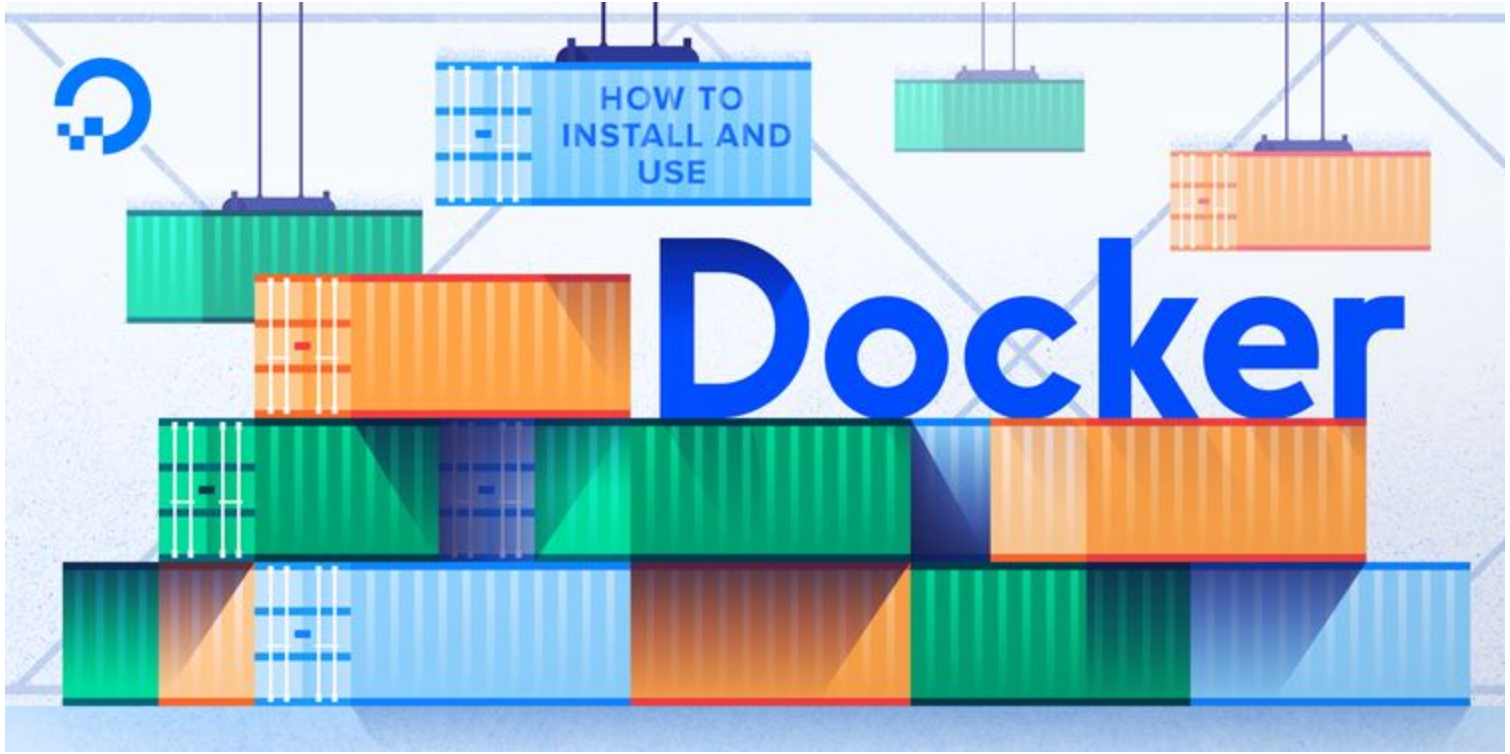


Traditional Deployment **Virtualized Deployment** **Container Deployment**

Major differences are:

Virtual Machine		docker
HIGH	Memory usage	LOW
	Performance	
	Portability	
	Boot-up time	

simplilearn


HOW TO
INSTALL AND
USE
Docker

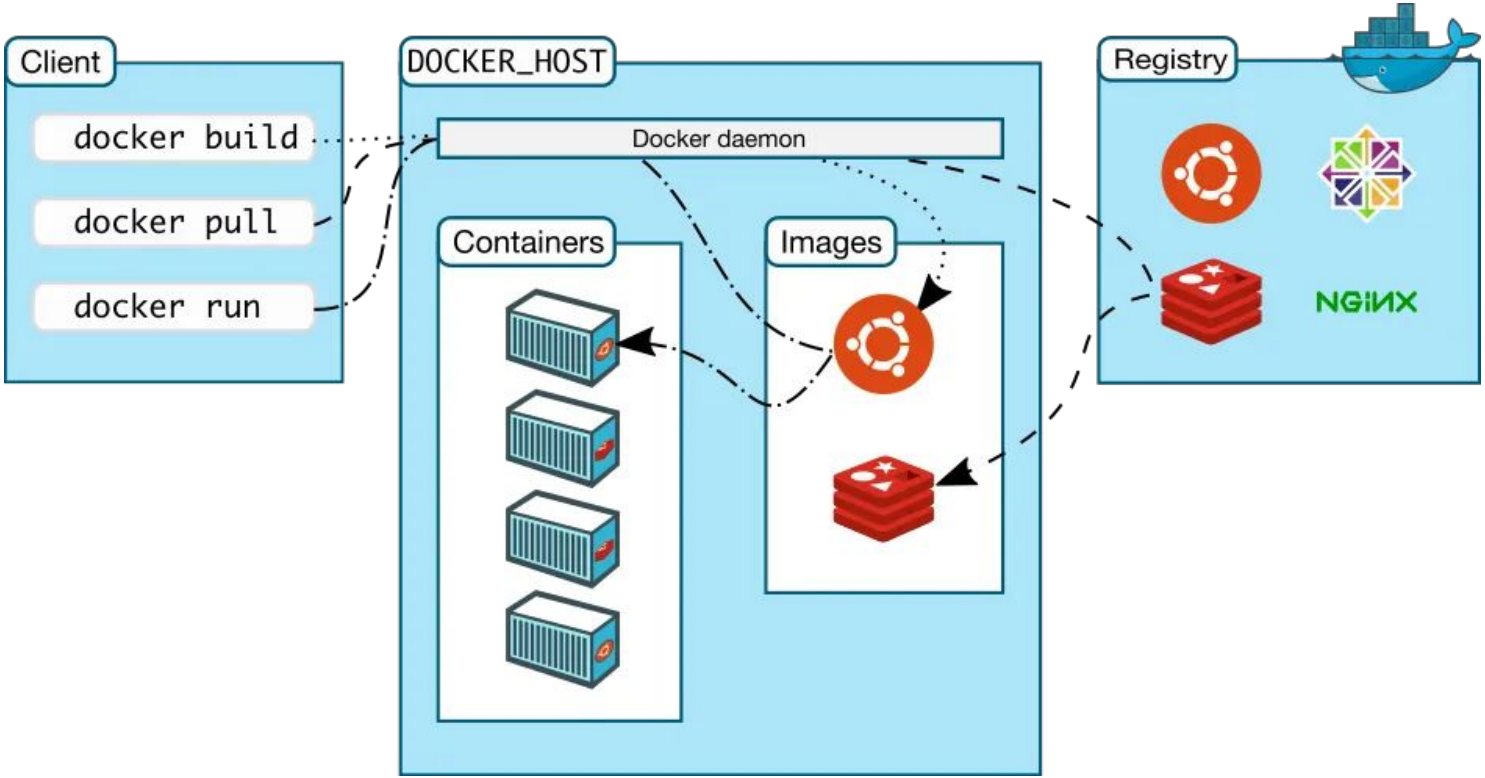
$ docker version

```
Client:
 Cloud integration: v1.0.29
 Version:           20.10.22
 API version:       1.41
 Go version:        go1.18.9
 Git commit:        3a2c30b
 Built:             Thu Dec 15 22:28:41 2022
 OS/Arch:           darwin/amd64
 Context:           orbstack
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          24.0.6
  API version:      1.43 (minimum version 1.12)
  Go version:       go1.20.7
  Git commit:       1a79695
  Built:            Mon Sep  4 12:32:17 2023
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          v1.7.6
  GitCommit:        091922f03c2762540fd057fba91260237ff86acb
 runc:
  Version:          1.1.9
  GitCommit:        82f18fe0e44a59034f3e1f45e475fa5636e539aa
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

$ docker run hello-world

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:4bd78111b6914a99dbc560e6a20eab57ff6655aea4a80c50b0c5491968cbc2e6
Status: Downloaded newer image for hello-world:latest

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.
    (amd64)
 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/
```

How Docker works ?



Basic of Docker

Image vs Container vs Registry

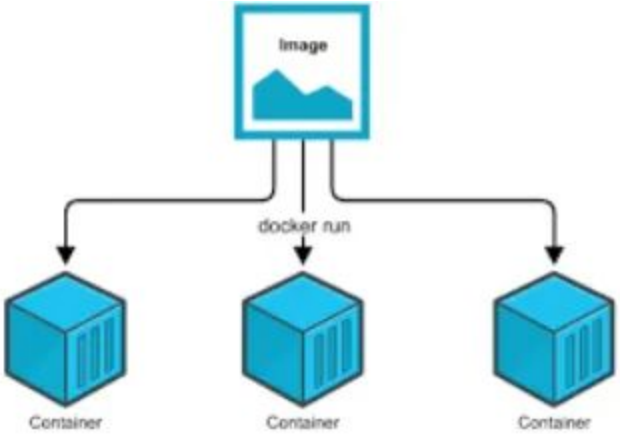
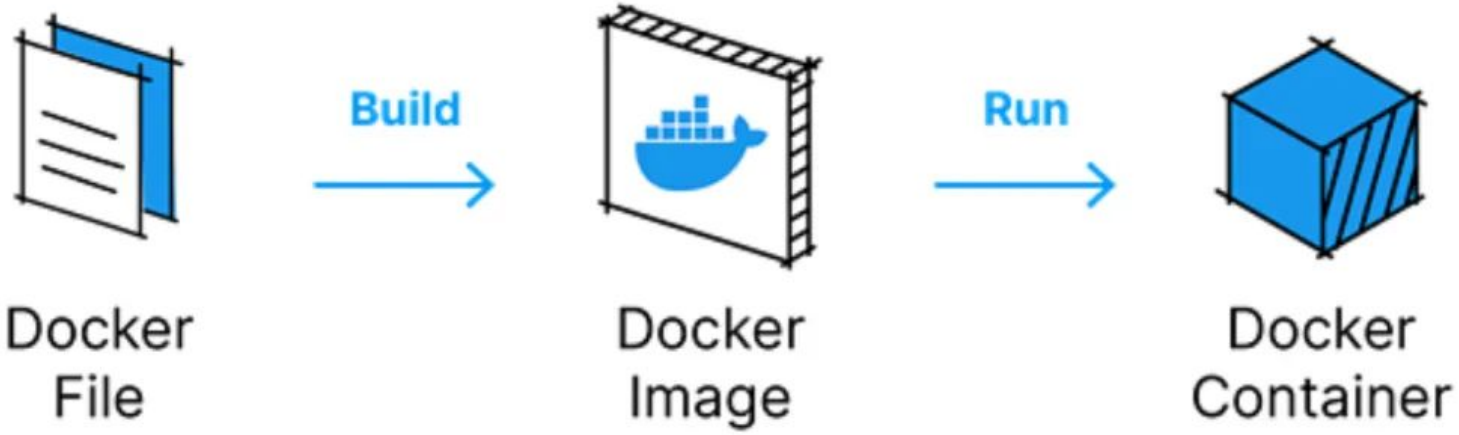
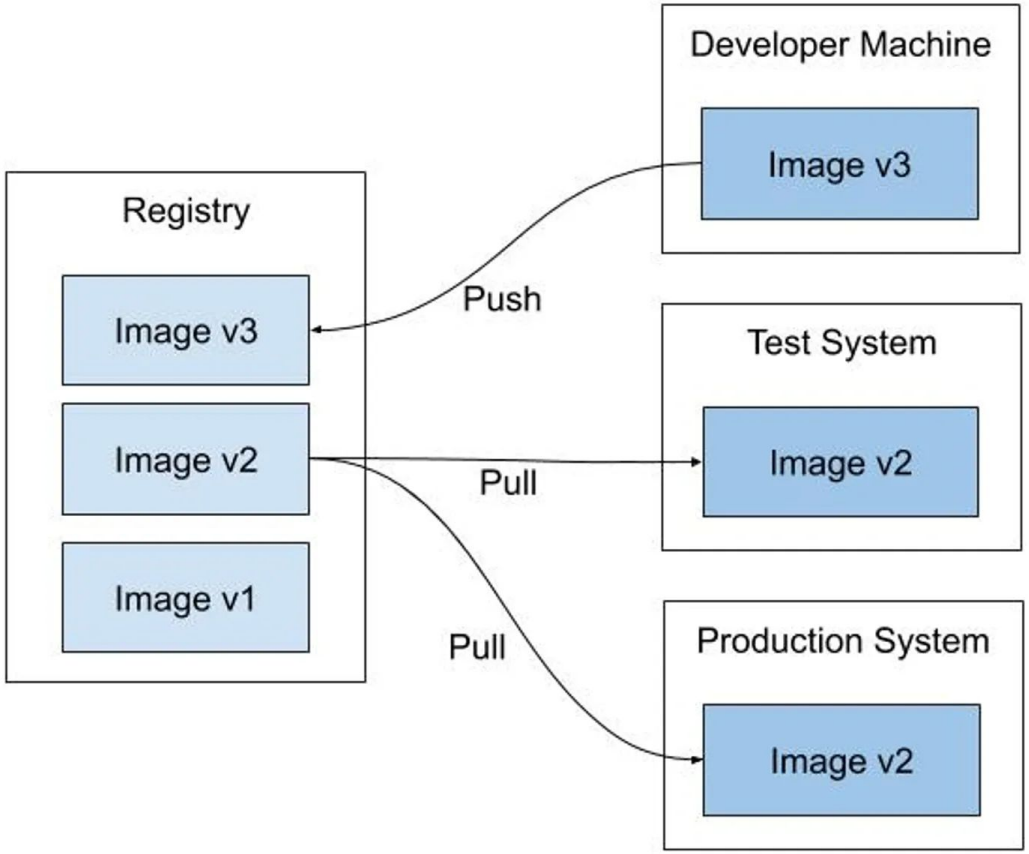


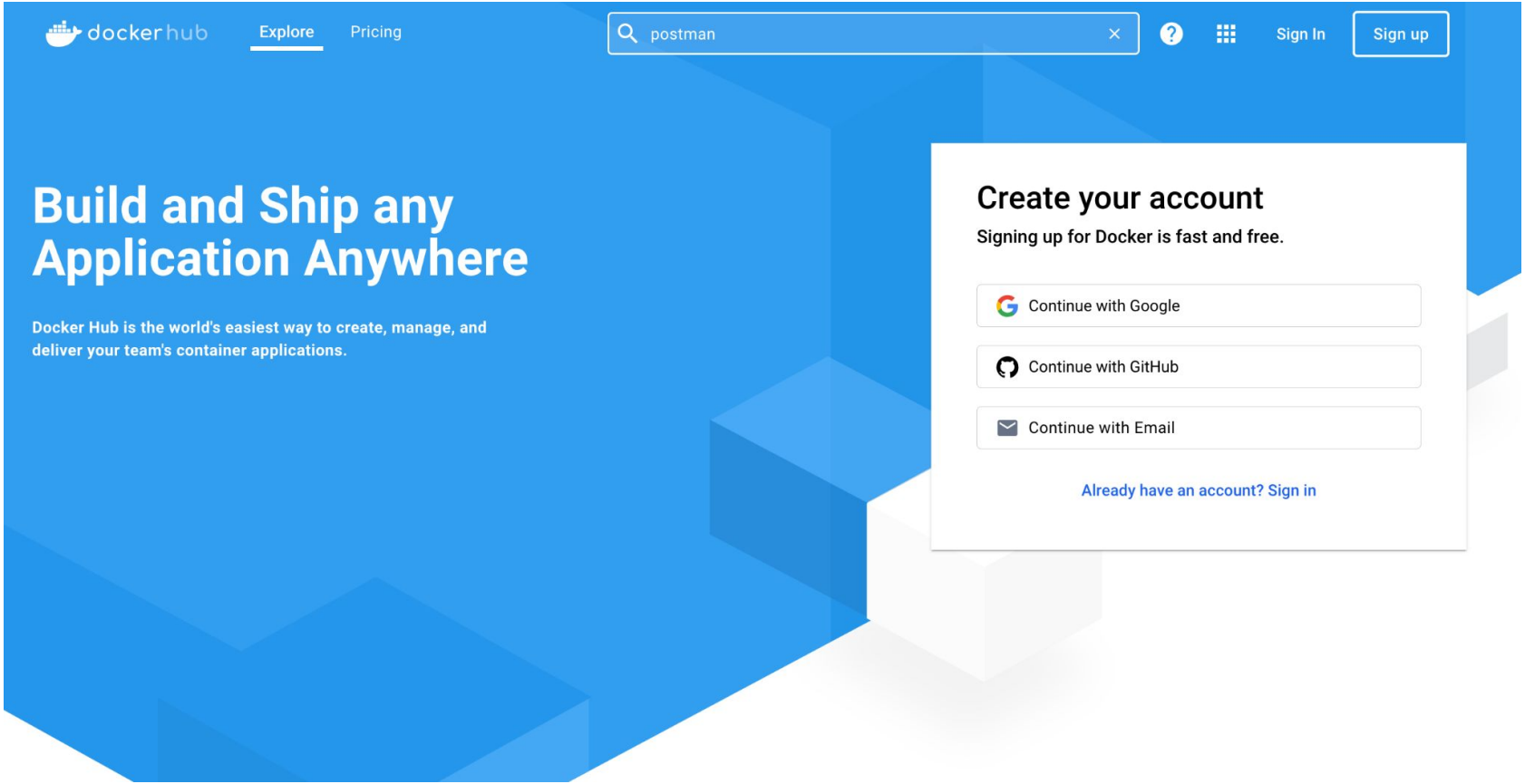
Image
docker run
Container
Container
Container

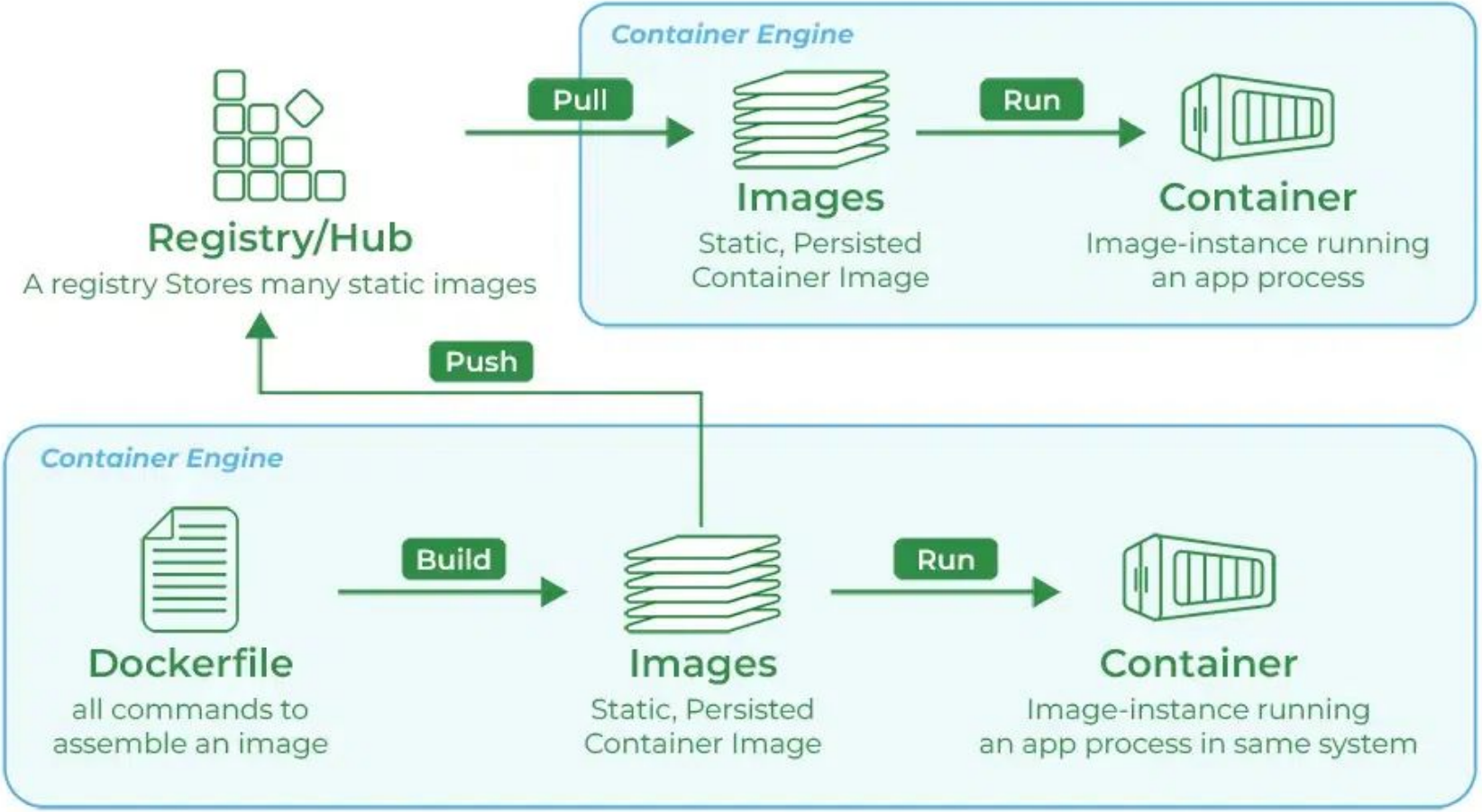

Build
Run
Docker File
Docker Image
Docker Container


UNDAI
EVERGREEN


Developer Machine
Image v3
Registry
Image v3
Image v2
Image v1
Push
Test System
Image v2
Pull
Pull
Production System
Image v2

https://hub.docker.com/




Container Engine
Pull
Run
Registry/Hub
A registry Stores many static images
Images
Static, Persisted
Container Image
Container
Image-instance running
an app process
Push
Container Engine
Build
Run
Dockerfile
all commands to
assemble an image
Images
Static, Persisted
Container Image
Container
Image-instance running
an app process in same system

Docker command

image command

- docker images หรือ docker image ls -a
- docker push <image_name> หรือ docker image push <image_name>
- docker pull <image_name> หรือ docker image pull <image_name>
- docker rmi <image_name> หรือ docker image rm <image_name>
- docker rmi $(docker images -a -q) หรือ docker image rm $(docker image ls -a -q)
- docker image

container command

- docker ps -a หรือ docker container ls -a
- docker container logs <container_id>
- docker container logs --tail <num_line> <container_id>
- docker container logs --tail <num_line> --follow <container_id>
- docker stop <container_id> หรือ docker container stop <container_id>
- docker stop $(docker ps -a -q) หรือ docker container stop $(docker ps -a -q)
- docker rm <container_id> หรือ docker container rm <container_id>
- docker container prune
- docker container

Workshop

Try Docker command with nginx

Let’s do it!!

- docker container run nginx
- docker container run jpetazzo/clock
- docker container run -d nginx
- docker container run -p 3000:80 -d nginx
- docker container run -p 3000:80 --name my_nginx -d nginx
- docker container exec -it my_nginx bash
    - cd /usr/share/nginx/html
    - apt update
    - apt install nano
- docker container run -p 3000:80 --name my_nginx -v $(pwd):/usr/share/nginx/html -d nginx
    - try to exec again and change index.html in the container
    - Back to index.html in host, what’s happen

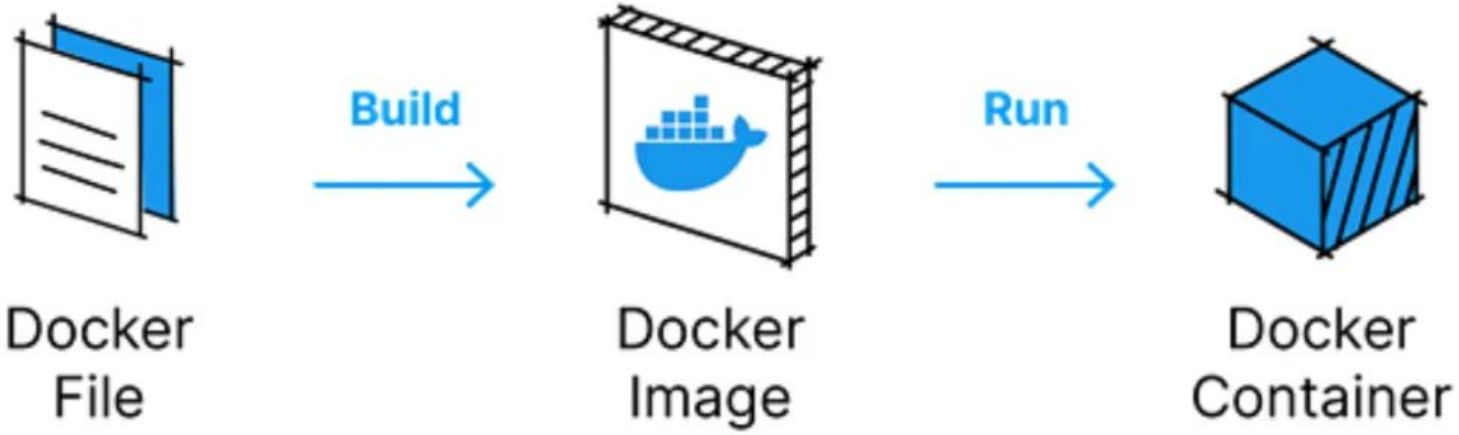
Workshop

Try create the own image & push to registry

Let’s do it!!

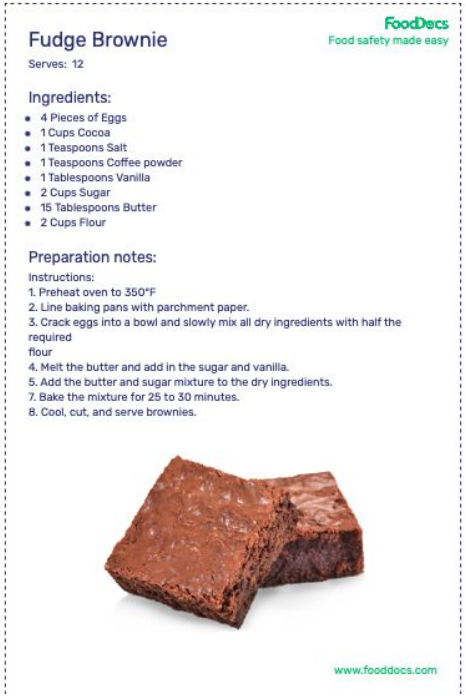
- docker container run -p 3000:80 --name my_nginx -d nginx
- docker container exec -it my_nginx bash
  - cd /usr/share/nginx/html
  - apt update
  - apt install nano
- docker container commit <container_id> <new_image_name>
    - docker container commit my_nginx my_nginx_update
- docker tag <image_name>:<tag_number> <username>/<repo_name>
    - docker tag my_nginx_update:0.0.1 srank123/nginx
- docker push <username>/<repo_name>
    - docker push srank123/nginx
- docker container run -p 3000:80 -d srank123/nginx

Dockerfile


Build
Run
Docker File
Docker Image
Docker Container

Dockerfile

- list instructions for the Docker daemon to follow when building a container image



FoodDocs
Food safety made easy

Fudge Brownie

Serves: 12

Ingredients:

- 4 Pieces of Eggs
- 1 Cups Cocoa
- 1 Teaspoons Salt
- 1 Teaspoons Coffee powder
- 1 Tablespoons Vanilla
- 2 Cups Sugar
- 15 Tablespoons Butter
- 2 Cups Flour

Preparation notes:

Instructions:
1. Preheat oven to 350°F
2. Line baking pans with parchment paper.
3. Crack eggs into a bowl and slowly mix all dry ingredients with half the required
flour
4. Melt the butter and add in the sugar and vanilla.
5. Add the butter and sugar mixture to the dry ingredients.
7. Bake the mixture for 25 to 30 minutes.
8. Cool, cut, and serve brownies.

www.fooddocs.com

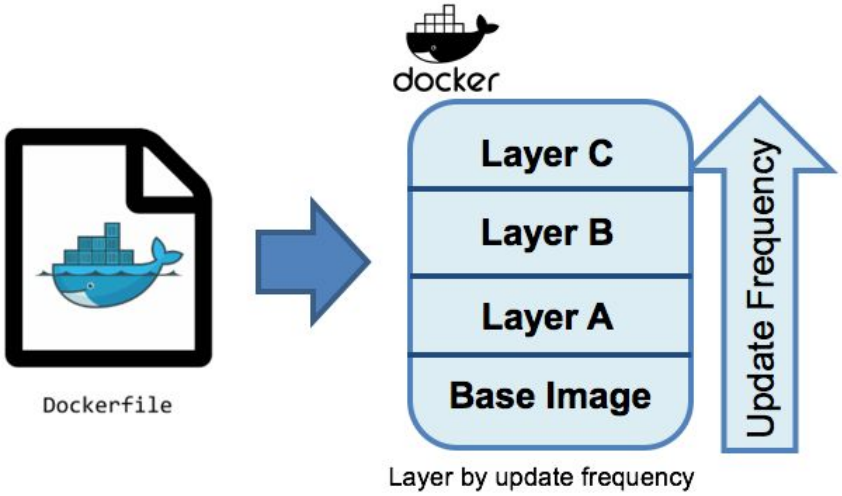
Dockerfile

- list instructions for the Docker daemon to follow when building a container image

```
FROM node:18.7.0

WORKDIR /code

COPY package.json package.json
COPY package-lock.json package-lock.json

RUN npm install

COPY . .

CMD [ "node", "server.js" ]
```

Dockerfile

```
FROM node:18.7.0

WORKDIR /code

COPY package.json package.json
COPY package-lock.json package-lock.json

RUN npm install

COPY . .

CMD [ "node", "server.js" ]
```



Dockerfile

- Instructions are executed in order
- Each instruction creates a new layer in images
- Instructions are cached
- FROM instruction must be the first
- Comment with #
- One CMD, One ENTRYPOINT

Dockerfile - Instructions

- FROM
- WORKDIR
- COPY
- RUN
- CMD

```
FROM node:18.7.0

WORKDIR /code

COPY package.json package.json
COPY package-lock.json package-lock.json

RUN npm install

COPY . .

CMD [ "node", "server.js" ]
```

https://docs.docker.com/engine/reference/builder/#overview

Dockerfile - cache

```
1   FROM node:18
2
3   WORKDIR /app
4   COPY . .
5   RUN npm install
6
7   EXPOSE 3000
8   CMD npm run dev
```

```
1   FROM node:18
2
3   WORKDIR /app
4   COPY package*.json ./
5   RUN npm install
6   COPY . .
7   EXPOSE 3000
8   CMD npm run dev
```

Dockerfile - muti stage

Workshop

Build own nginx image with Dockerfile

Dockerfile workshop

- Create index.html include “Hello docker” in file
- Create Dockerfile in your directory

Dockerfile

```
FROM nginx:1.25.3

COPY index.html /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

- Run command `docker build -t my_nginx_image .`
- Run command `docker container run -p 3000:80 --name my_nginx -d my_nginx_image`
- Run command `docker container run -p 3000:80 --name my_nginx -v $(pwd):/usr/share/nginx/html -d my_nginx_image`

Workshop

Build own json-server image with Dockerfile

Dockerfile workshop

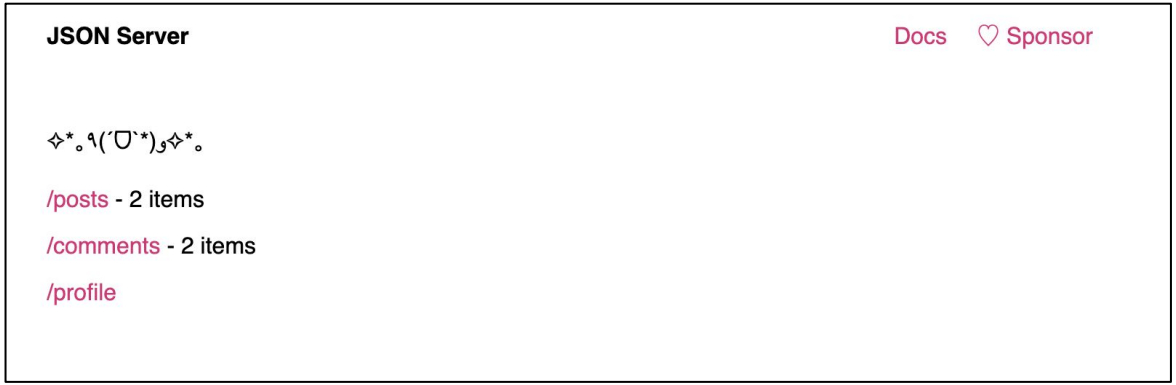
- Create directory json-server
- Create db.json in your directory (db.json ⇒ https://www.npmjs.com/package/json-server)
- Create Dockerfile in your directory

```
FROM node:alpine

WORKDIR /app
RUN npm install -g json-server

COPY . .
EXPOSE 3000
ENTRYPOINT ["json-server", "db.json"]
```

- Run command `docker build -t my_json_image .`
- Run command `docker container run -p 3001:3000 --name my_json -d my_json_image`

Dockerfile workshop

- localhost:3001



It’s work on db.json only, cannot work with new json file.

Let’s modify Dockerfile !!

Dockerfile workshop

- Create new file => db2.json
- Modify Dockerfile

```
FROM node:alpine

WORKDIR /app
RUN npm install -g json-server

COPY . .
EXPOSE 3000
ENTRYPOINT ["json-server"]
CMD ["db.json"]
```

- Run command `docker build -t my_json_image .`
- Run command `docker container run -p 3001:3000 --name my_json -d my_json_image`
- Run command `docker container run -p 3001:3000 --name my_json -d my_json_image db.json`
- Run command `docker container run -p 3001:3000 --name my_json -d my_json_image db2.json`

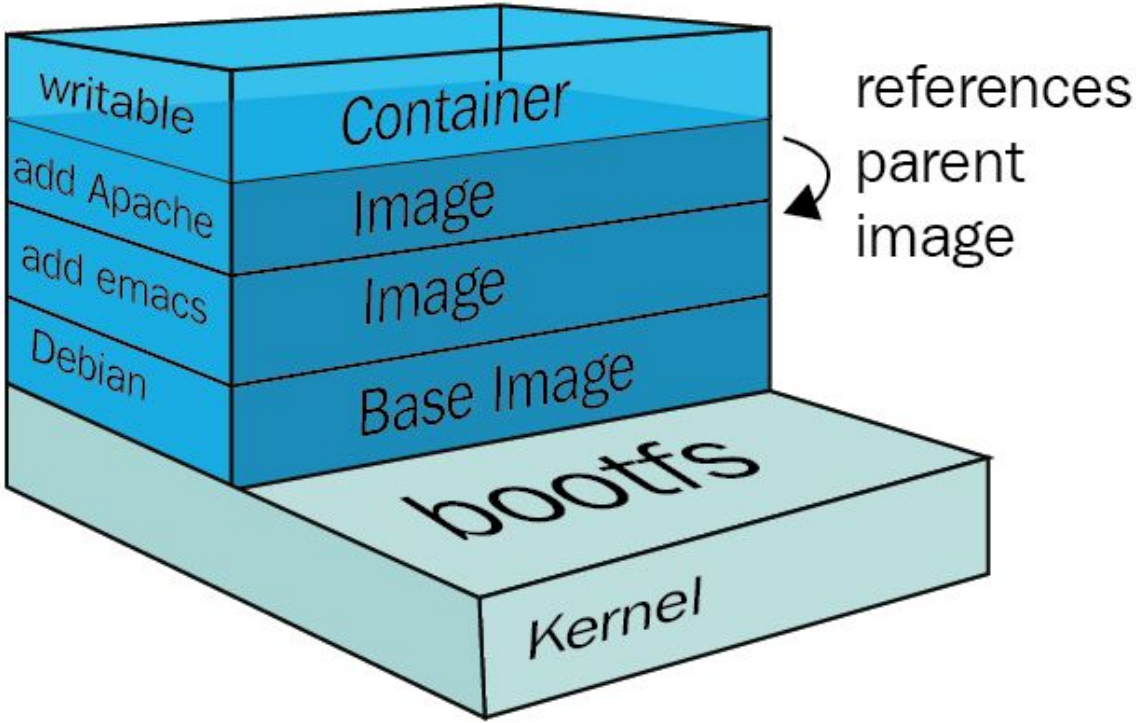
Workshop

Build own mountebank image with Dockerfile

TRY BY YOURSELF !!

- ??
- ??
-

Image layer


writable
add Apache
add emacs
Debian
Container
Image
Image
Base Image
bootfs
Kernel
references
parent
image

Docker history

- Run command: docker history <image_name>
    - docker history my_mb_image

```
IMAGE          CREATED        CREATED BY                                      SIZE      COMMENT
e3eea268bc7b   9 hours ago    CMD ["start"]                                   0B        buildkit.dockerfile.v0
<missing>      9 hours ago    ENTRYPOINT ["mb"]                               0B        buildkit.dockerfile.v0
<missing>      9 hours ago    EXPOSE map[2525/tcp:{}]                         0B        buildkit.dockerfile.v0
<missing>      9 hours ago    COPY . . # buildkit                             7.14kB    buildkit.dockerfile.v0
<missing>      34 hours ago   RUN /bin/sh -c npm install -g mountebank # b…   31.5MB    buildkit.dockerfile.v0
<missing>      34 hours ago   WORKDIR /imposters                              0B        buildkit.dockerfile.v0
<missing>      5 days ago     /bin/sh -c #(nop)  CMD ["node"]                 0B
<missing>      5 days ago     /bin/sh -c #(nop)  ENTRYPOINT ["docker-entry…   0B
<missing>      5 days ago     /bin/sh -c #(nop) COPY file:4d192565a7220e13…   388B
<missing>      5 days ago     /bin/sh -c apk add --no-cache --virtual .bui…   7.82MB
<missing>      5 days ago     /bin/sh -c #(nop)  ENV YARN_VERSION=1.22.19     0B
<missing>      5 days ago     /bin/sh -c addgroup -g 1000 node     && addu…   126MB
<missing>      5 days ago     /bin/sh -c #(nop)  ENV NODE_VERSION=21.6.1      0B
<missing>      9 days ago     /bin/sh -c #(nop)  CMD ["/bin/sh"]              0B
<missing>      9 days ago     /bin/sh -c #(nop) ADD file:37a76ec18f9887751…   7.38MB
```

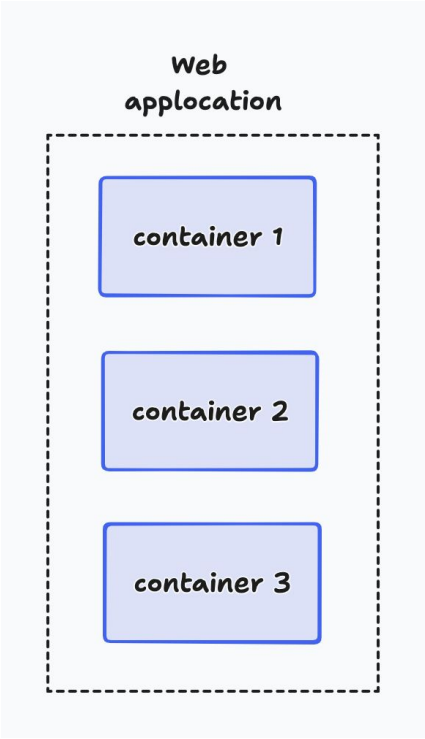
node:alpine

Dockerfile best practice

- Do not use latest image
- Use official images
- Order matters for caching
- Use instruction RUN together in same line
- Avoid use instruction copy all file (COPY .)
    - Able to use .dockerignore
- Use rootless user

Docker Compose

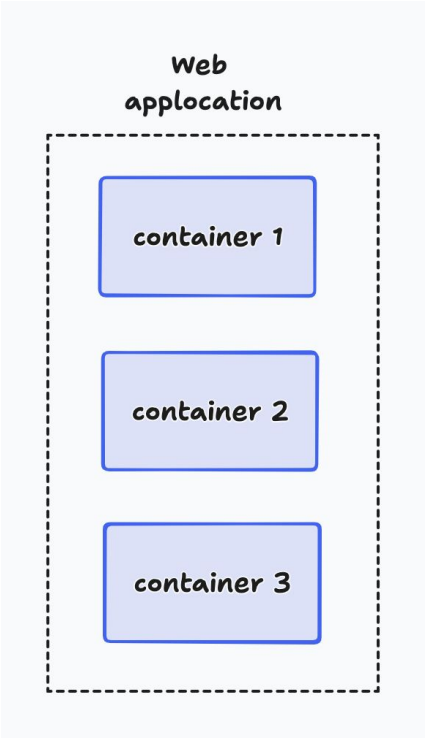
Mutiple containers



container 1

```
$ docker pull image
$ docker build -t image1 .
$ docker run -p 8080:8080 –name name image1
```

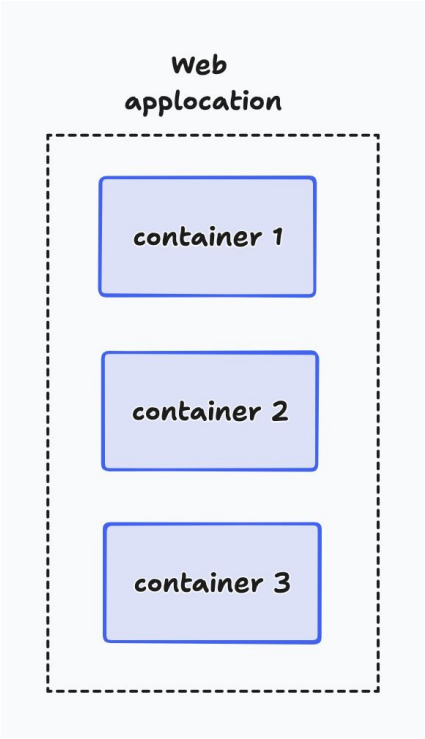
Mutiple containers



container 2

```
$ docker pull image
$ docker build -t image2 .
$ docker run -p 5000:5000 –name name -v $pwd:dir/sub image2
```

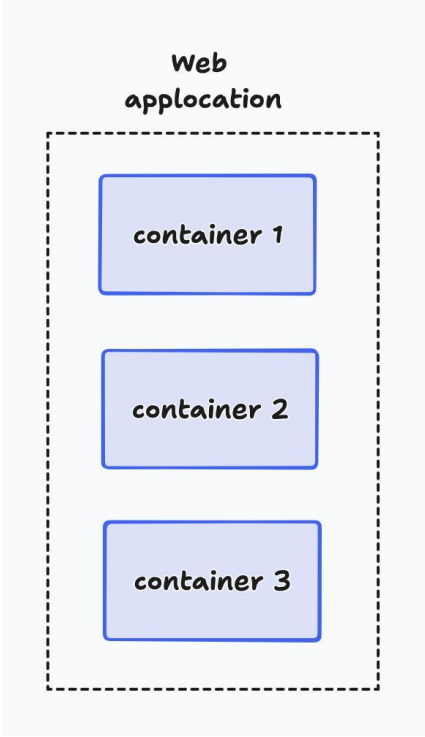
Mutiple containers



container 3

```
$ docker pull image
$ docker build -t image3 .
$ docker run -p 2525:2525 –name name -v $pwd:dir/sub image3
```

Mutiple containers

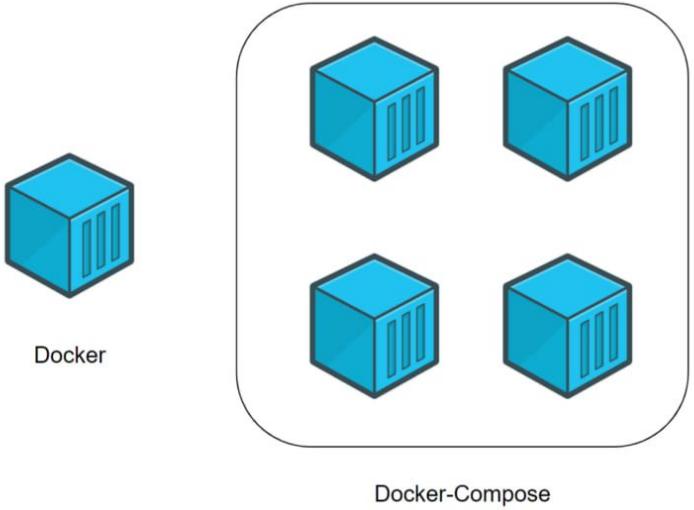


all

```
$ docker pull …
$ docker pull …
$ docker pull …

$ docker build …
$ docker build …
$ docker build …

$ docker run …
$ docker run …
$ docker run …
```

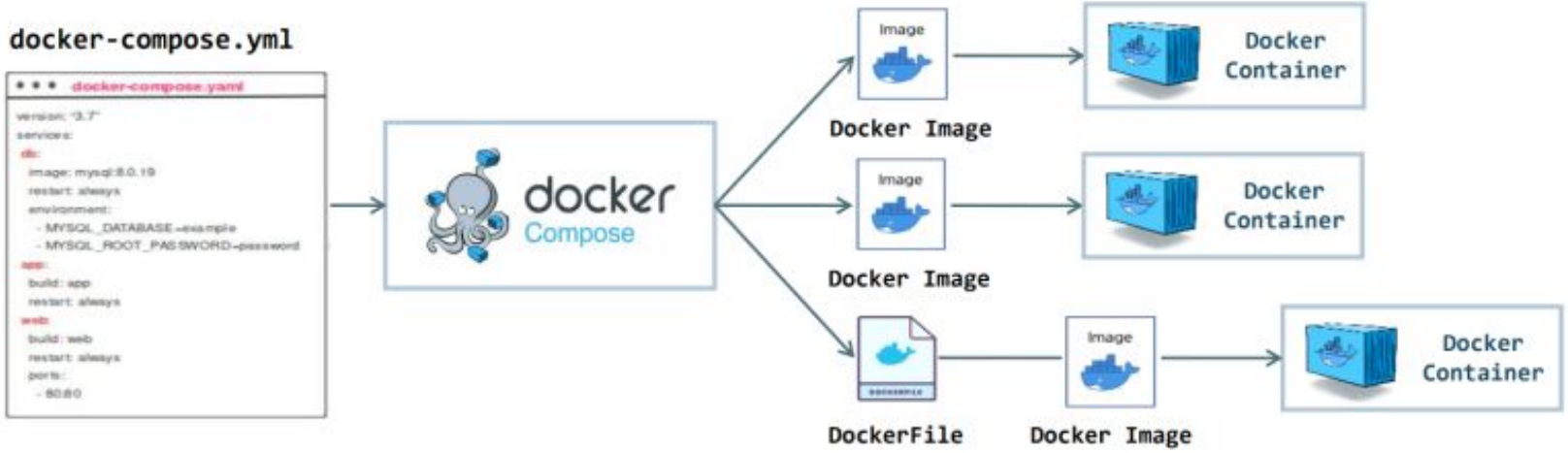
Docker compose

- Application have multiple components
- You can easily run multiplay-container application with Docker Compose



Docker compose

Running app in one command-line



Docker compose

docker-compose.yml

docker-compose.yml

```yaml
version: "3.5"
services:
  nginx01:
    image: nginx:1.25.3
    container_name: my_nginx
    ports:
      - 3000:80
    volumes:
      - .:/usr/share/nginx/html
```

Docker compose

- Run `docker-compose up -d` or `docker-compose up`
- Without docker compose: `docker container run -p 3000:80 --name my_nginx -v $(pwd):/usr/share/nginx/html -d my_nginx_image`
- Define informations (port, name, volumes, etc.) in docker-compose.yml
- Easy to use, shorter command

Docker compose - command line

- docker compose up
- docker compose down
- docker compose build
- docker compose start
- docker compose stop

https://docs.docker.com/compose/reference/

Docker compose - multiple containers

- Modify docker-compose.yml
    - add service nginx02
- Run `docker compose up -d`
- Run `docker compose up nginx01 -d`
- Run `docker compose up nginx02 -d`
- Run `docker compose down nginx01`
- Run `docker compose down nginx02`

Docker compose - pull own hub

docker-compose.yml

```yaml
version: "3.5"
services:
  nginx01:
    image: srank123/nginx
    container_name: my_nginx
    ports:
      - 3000:80
```

Docker compose - own image

```yaml
json_server:
  image: my_json_images:1.0.0
  container_name: my_json_server
  build: # when Dockerfile directory is not place same docker-compose.yml directory
    context: ./json-server
    dockerfile: ./Dockerfile
  ports:
    - 3002:3000
```

Docker compose - own image

```yaml
json_server:
  image: my_json_images:1.0.0
  container_name: my_json_server
  build: # when Dockerfile directory is not place same docker-compose.yml directory
    context: ./json-server
    dockerfile: ./Dockerfile
  ports:
    - 3002:3000
```

Your image name

Depend on directory name

Depend on Dockerfile name

docker compose up json-server --build ⇒ Build new image

Docker compose - mountebank

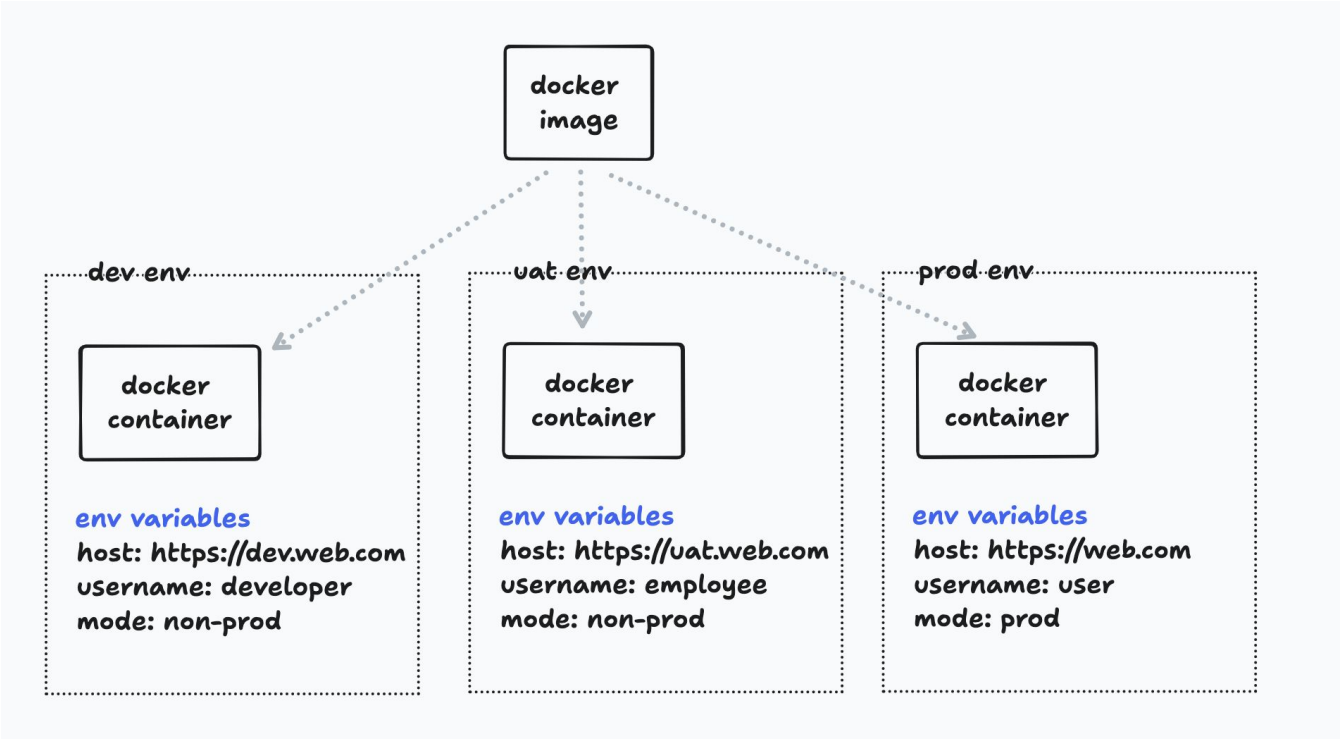
- ???
- ???
- ???

Docker compose - depend on service

```yaml
nginx01:
  image: nginx:1.25.3
  container_name: my_nginx
  ports:
    - 3000:80
  depends_on:
    - json_server
```

- Run docker compose up, then see logs what's happened ?
- Remove depends_on and up again, what's difference ?

Docker compose - environment variables



Docker compose - environment variables

```yaml
postgres:
  image: postgres:13.7
  environment:
    POSTGRES_USER: postgres
    POSTGRES_PASSWORD: postgres
  ports:
    - "5433:5432"
```

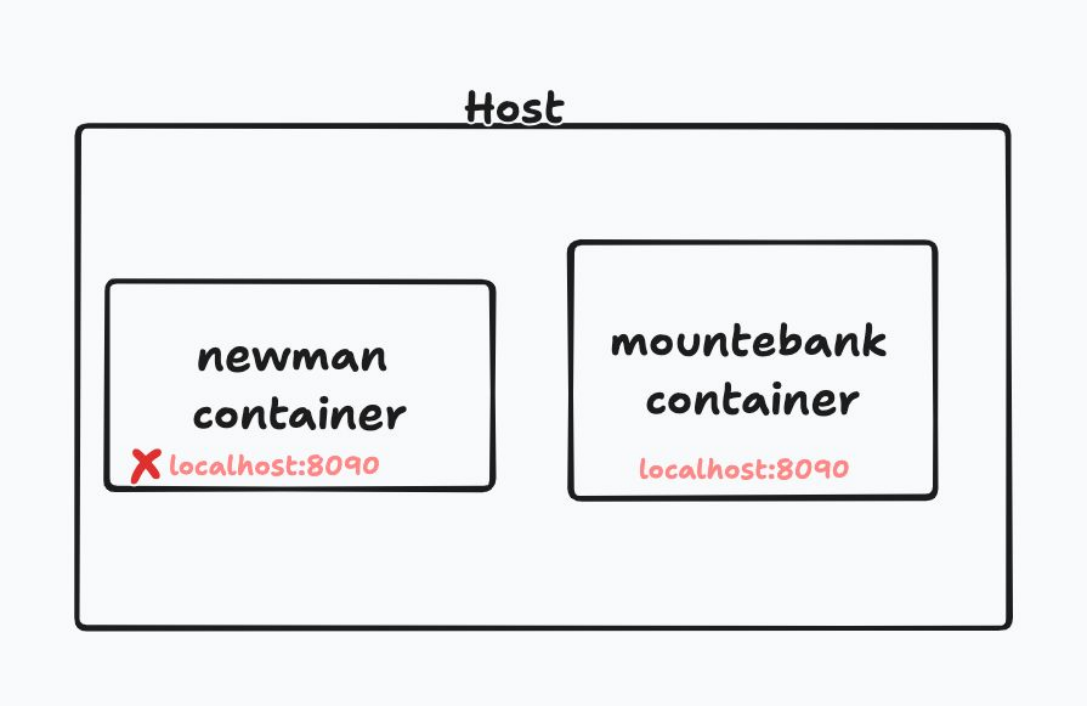
Workshop

Run newman with docker compose

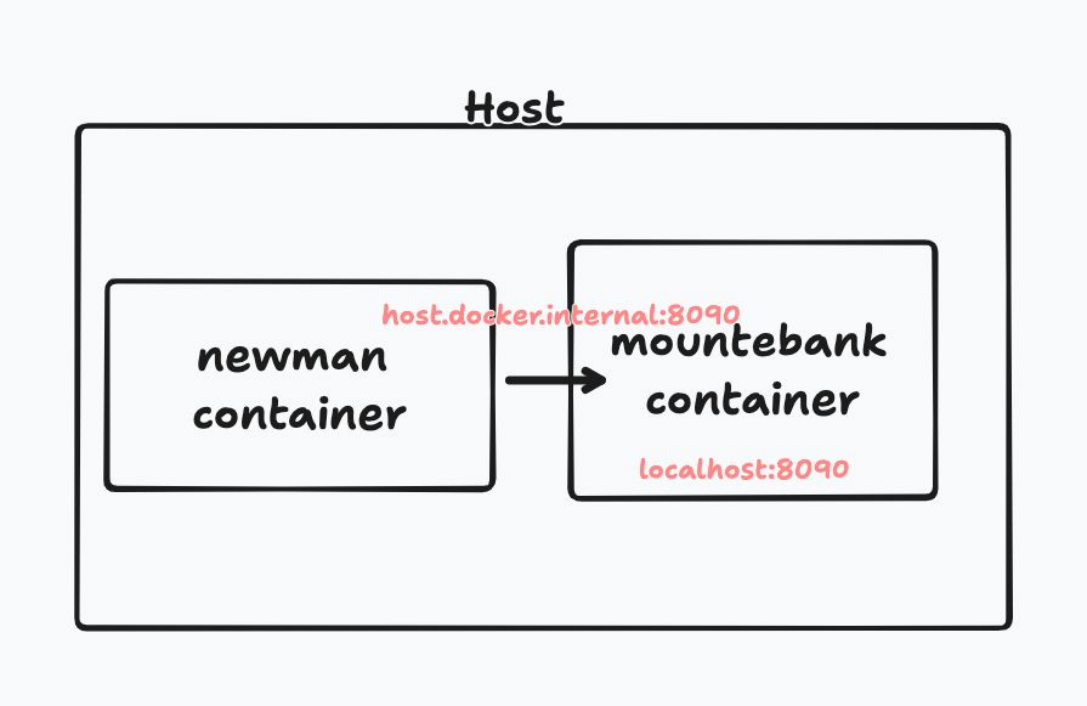
Docker compose - newman

- Create service newman in docker-compose.yml
- Running newman container after mountebank started
- Able to call mountebank service
- Generate reports

Docker compose - newman



Docker compose - newman



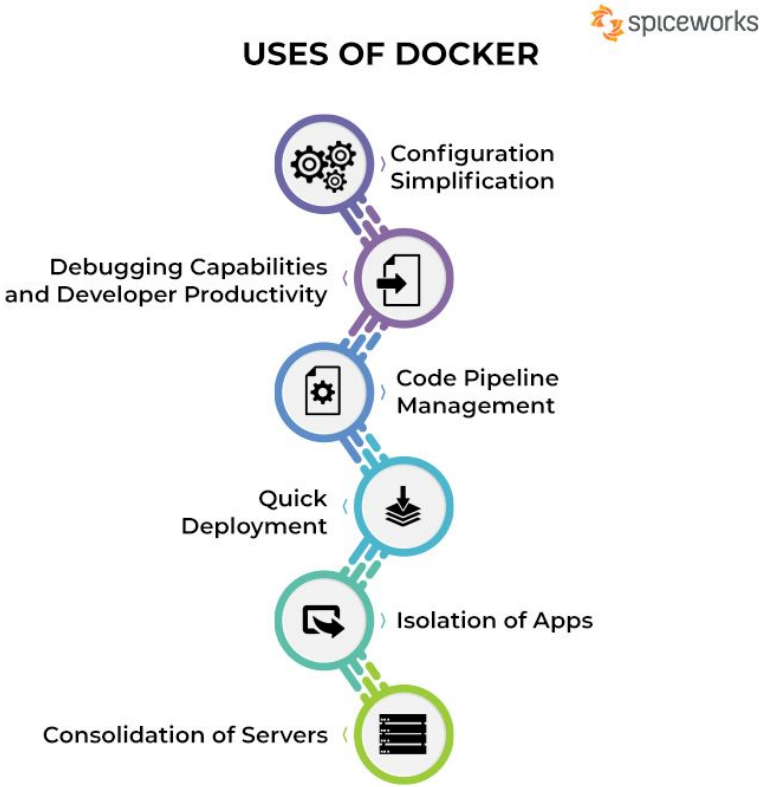
Docker compose - more example

- https://github.com/docker/awesome-compose/tree/master

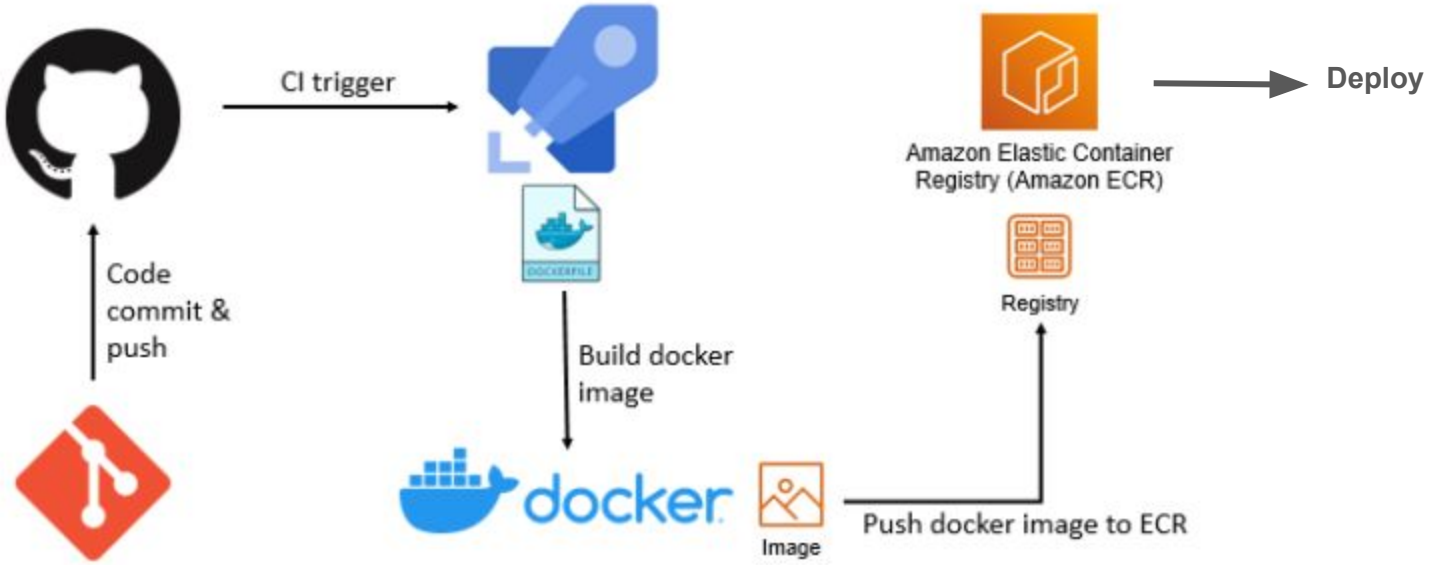
Nextjs

- Install Nextjs
- Create Dockerfile
- Create docker-compose.yml
- Deploy on VM

Docker usecases

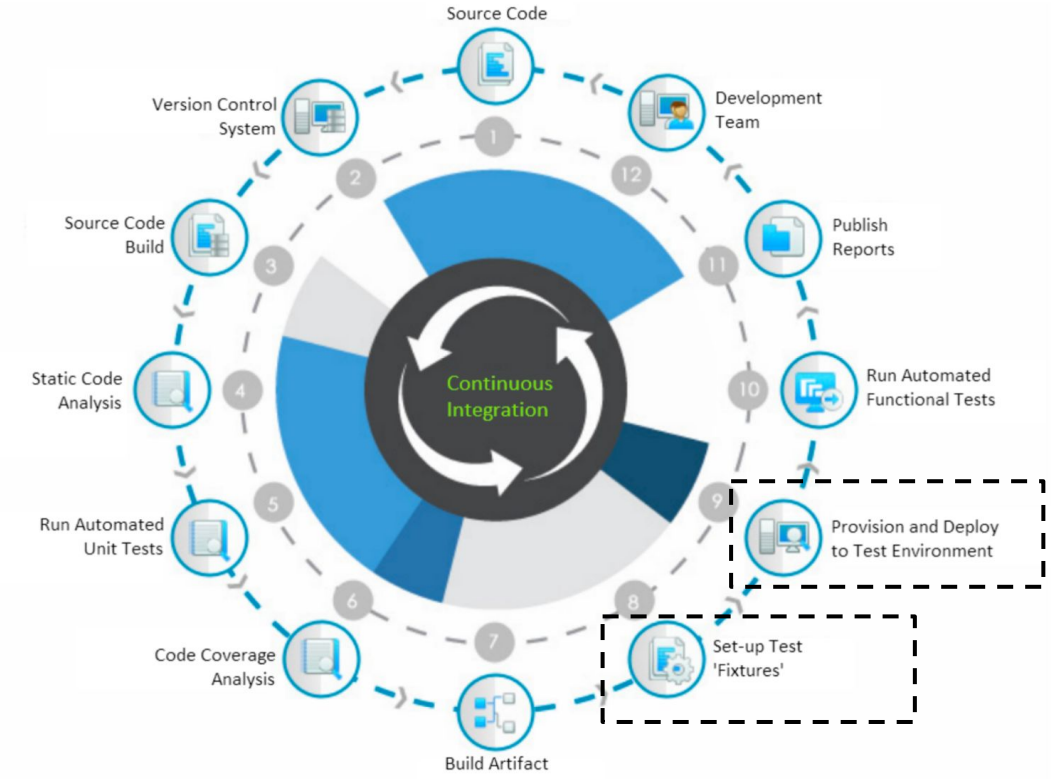


Apply Docker with pipeline



Skundunotes: Push Docker images to Amazon ECR using Azure Pipelines

Apply Docker with pipeline



docker swarm