Docker tutorial

- **Rajiv**

CS695

Spring 2022-23

Structure

1. Introduction to Docker
2. Steps to run a container in Docker: from image to container
    a. Setting up the environment: basic installations/ configurations.
    b. Commands to create a docker image.
    c. Commands to create and execute a docker container: Dockerfile way, Docker-compose way
3. Demo: deploying an echo server in Docker container
4. Task: Deploying a C echo server in Docker.
5. What next? Introduction to container orchestration with K8s.

Some theory first…

Introduction to Docker

- An **isolated** execution environment to develop, ship and run production-scale applications.
- **Faster** deployment: Developers can reuse existing Docker images and just focus on their application.
- **Portability:** solves the problem “works in their machine, but not in mine”.

How does Docker work?

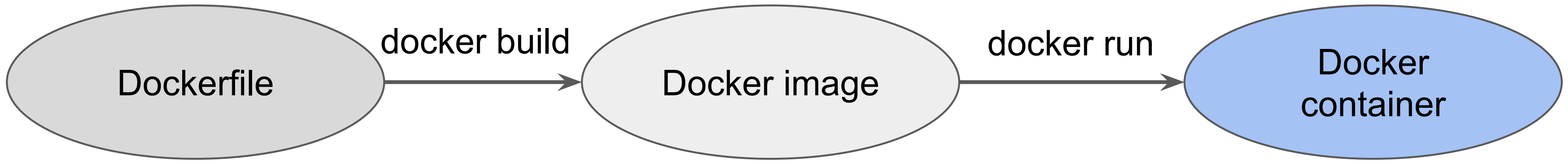
- Docker uses the following **namespaces**:
  - The pid namespace: Process isolation.
  - The net namespace: Managing network interfaces.
  - The ipc namespace: Managing access to IPC resources.
  - The mnt namespace: Managing filesystem mount points.
  - The uts namespace: Isolating kernel and version identifiers.

- Uses **cgroups** to limit resources per namespace.

Hands-on

Docker lifecycle - specification to container - Method 1

- Used to launch a single container.



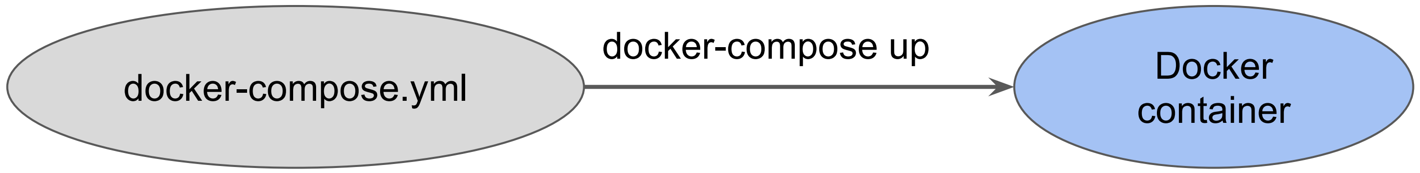
A text file containing information about the Docker image: the base image, additional installations, commands to execute inside container at startup

Read-only template consisting of layers generated from Dockerfile.
The container will be created using these layers.

Running instance of the Docker image which provides an isolated execution environment for the application.

Docker lifecycle - specification to container - Method 2

- Used to launch multiple containers with a single command.



A text file containing information about how to run multiple containers. Contains instructions to create the Docker image, container related specifications such as port number, resources limit, volume mounting, etc.

Running instance of the Docker image which provides an isolated execution environment for the application.

Examples of Dockerfile and docker-compose.yml

Dockerfile

```
FROM node:18-alpine
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
EXPOSE 3000
```

docker-compose.yml

```
services:
  worker:
    image:
dockersamples/examplevot
ingapp_worker
    networks:
      - frontend
      - backend
    deploy:
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
        reservations:
          cpus: '0.25'
          memory: 20M
      ...
```

Demo

THE DOCKER CONTAINER
MY HOST MACHINE
imgflip.com
ProgrammerHumor.io

Link to follow along

Source:

https://github.com/Rajiv2605/docker-tutorial

Echo server

Echo server uses **netcat** library to run the server.

Netcat:

- A network utility that performs data reads/writes data across the network.
- Runs on top of TCP/ IP.

Demo:

- Running the application outside Docker.
- How to dockerize the echo server?
  - Method 1: using Dockerfile
  - Method 2: using Docker-compose

Dockerizing echo server - Method 1

1. Install docker.
2. Prepare Dockerfile.
3. Prepare Docker image for the container.
4. Create and execute in the Docker container.
5. Test echo server operation.

Dockerizing echo server - Method 2

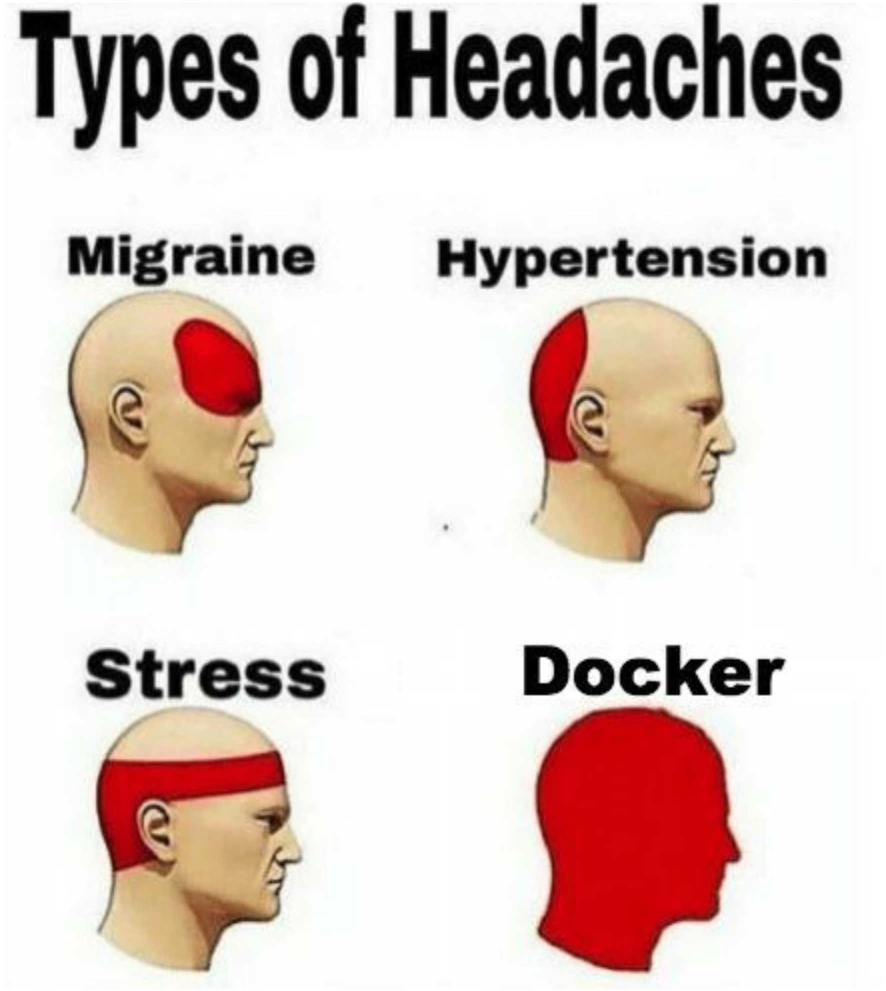
1. Install docker-compose.
2. Prepare docker-compose.yml file.
3. Create and execute in the Docker container.
4. Test echo server operation.

Link to follow along

Source:
https://github.com/Rajiv2605/docker-tutorial

Follow link for local setup of dockerized echo server as shown in demo

Task



Dockering a echo server implemented in C

1. Use provided C implementation of echo server in docker-tutorial/task and test it in a non dockerized setup.
2. Dockerize it and test its working.

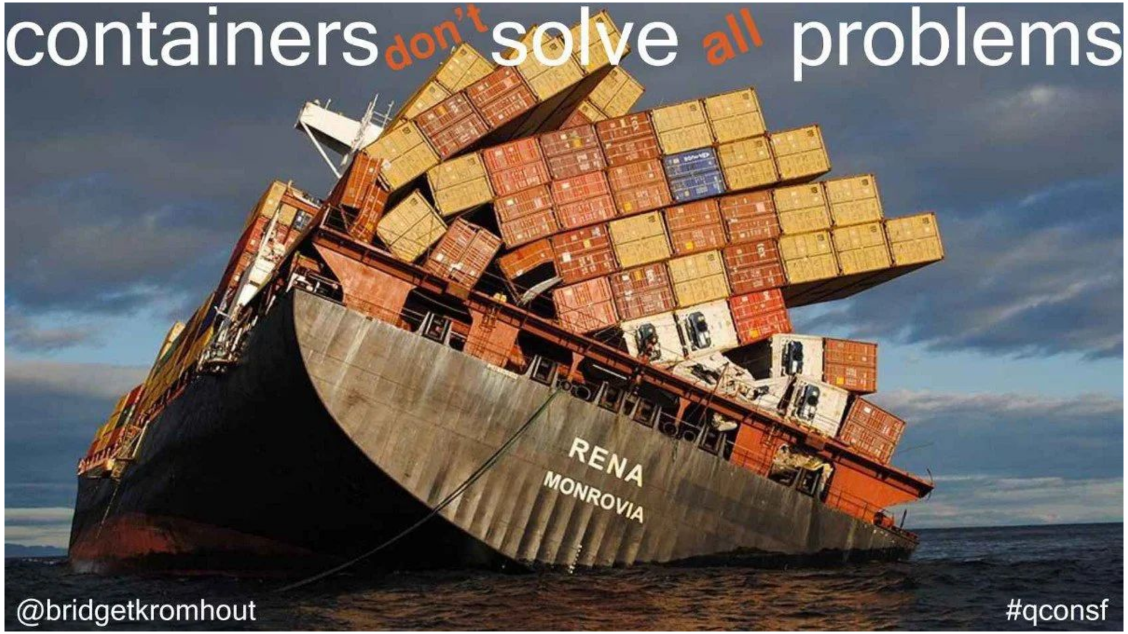
Dockering a echo server implemented in C

1. Use provided C implementation of echo server in docker-tutorial/task and test it in a non dockerized setup.
2. Dockerize it and test its working.

Hint:

3. Install gcc.
4. Copy C file to docker image.
5. Setup command sequence to compile the C program and execute it inside the container.

What next?


containers don't solve all problems
RENA
MONROVIA
@bridgetkromhout
#qconsf

Is container management trivial?

In a commercial cloud, the arrival rate of requests is highly varying (few requests in an hour to 100s of requests/s).

Manual container provisioning faces the following issues:

- Over-provisioning containers to satisfy sudden spikes in requests
  => resource wastage (low utilization)
- Under-provisioning
  => requests will have to wait till the container is created
- Unexpected failures affect the availability of the deployed application.

Difficult to manually manage containers ⇒ need a **container manager tool**.

Kubernetes - container management plane

- Open-source system for automating deployment, scaling, and management of containerized applications
- Autoscales containers based on the request arrival rates, avoiding over-/ under-provisioning
- Works across multiple servers (k8s cluster), can move services to another server once a particular server fails

Thank you