Containers

The Problem:

- Software development rate has skyrocketed but this has introduced discrepancies in terms of environment configuration, application isolation, and avoiding resource access conflicts.

Solution #1: Virtual Machines

- A Virtual Machine runs an instance of some Operating System and the host machine hardware is virtualized by a bare-metal hypervisor (a software responsible for the allocation of resources to virtual machines).

Solution #1: Virtual Machines (cont.)

The reason VMs are not always required to run our applications is that:

- Creation of a VM takes time.
- Boot time of a VM is high.
- Most OS sizes are in GBs.
- Dependencies are needed to be installed from scratch.
- Inability to share a VM instance and recreate the exact environment.
- Hardware resource wastage.

Solution #2: Containers

- **Containers** were introduced as a lightweight alternative.
- They made it much easy to develop and deploy software applications.
- These containers are created via software that runs on the host machine serving certain OS.

Here, we do not virtualize hardware but we virtualize the Operating System

Solution #2: Containers (cont.)

- Containers do not run a complete OS, they operate on kernels.
- A kernel is a core program in any OS that has complete control over all processes.
- The size of these kernels is far less as compared to that of an entire OS. For instance, Ubuntu 18.04 LTS is 2.00 GB whereas Ubuntu docker image is 27.25 MB only.
- Containers allowed developers to recreate an environment free of all dependency conflicts.
- Containers run on-demand, they boot quickly and only use resources as and when needed.

Docker components

- **Docker image:** Docker images are a blueprint of docker containers. It contains information about which software the container will run and how. Docker images are built using Dockerfile.
- **Docker container:** If the docker image is a blueprint then the docker container is an instance of these blueprints. Docker container takes the specifications from docker image and executes them to create an environment ready to run your application.
- **Dockerfile:** A Dockerfile is a simple human-readable text file that instructs docker how to build a docker image. Each line in this file is a command in the form `[INSTRUCTION] [ARGUMENT]`.

Examples

- `docker run hello-world`
- `docker run -it ubuntu bash`
- `docker ps`
- `docker ps -a`
- `docker images`

Containerizing an application

1. Go to the `app` directory in `dockerised-applications`.
2. Create a Dockerfile (no extension!)

```
FROM node:12-alpine
# Adding build tools to make yarn install work on Apple silicon / arm64 machines
RUN apk add --no-cache python3 g++ make
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
```

Containerizing an application (cont.)

- Build the image:

  ```
  docker build -t getting-started .
  ```

- Start a container with the image:

  ```
  docker run -dp 3000:3000 todo-app
  ```

  - d is for detached mode.
  - p maps the host port with the container port.

- `docker logs -f <container-id>`

Updating an application

- Make some small change in `src/static/app.js` file.
- Rebuild the image.
- Before starting a new container, you need to stop the existing:
  - `docker stop <container-id>`
  - `docker rm <container-id>`

Persisting data

- You can create, update and delete files inside of the container, but these are lost if the container is restarted.
- There are two main ways to preserve data: named volumes and bind mounts.

Named volumes

- Docker maintains the physical location on host and gets data as required.
  - `docker volume create todo-db`
  - `docker rm -f <container-id>`
  - `docker run -dp 3000:3000 -v todo-db:/etc/todos todo-app`
- You can see this location with `docker volume inspect todo-db`.

Bind mounts

- Bind mounts allow to control the mounting point.
- Besides creating persisting volumes, they can be used to pass extra data.
- Common use case: create a bind mount that watches for changes in the development environment and copies them intro the container.

Bind mounts (cont.)

- Launch a container with a bind mount

```
docker run -dp 3000:3000
    -w /app -v "$(pwd):/app"
    node:12-alpine
    sh -c "yarn install && yarn run dev"
```

- Make a change on `src/static/js/app.js`
- Changes should reflect almost immediately on the browser.

Multi-container apps

- In general, a container should **do one thing**.
- The app supports SQLite, let's add a MySQL database instead (on its own container).
- We can use a `docker-compose.yml` file for defining all the containers we need.
- This has the advantage that all services will be visible in the same network.

```
services:
  app:
    image: node:12-alpine
    command: sh -c "yarn install && yarn run dev"
    ports:
      - 3000:3000
    working_dir: /app
    volumes:
      - ./:/app
    environment:
      MYSQL_HOST: mysql
      MYSQL_USER: root
      MYSQL_PASSWORD: secret
      MYSQL_DB: todos

  mysql:
    image: mysql:5.7
    volumes:
      - todo-mysql-data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: secret
      MYSQL_DATABASE: todos

volumes:
  todo-mysql-data:
```

===

Running the app

Best practices

- Security scanning with `docker scan <container>`
  - 10 free per month
- `.dockerignore` file.

Best practices (cont.)

- Image layering: Once a layer changes, all layers below have to be recreated as well
  - `docker image history todo-app`
  - To use cached images in our example, move the `COPY . .` command below `yarn install`

Best practices (cont.)

- Multi-stage builds: use a second `FROM` command to reduce image size.

```
FROM maven AS build
WORKDIR /app
COPY . .
RUN mvn package

FROM tomcat
COPY --from=build /app/target/file.war /usr/local/tomcat/webapps
```

- Also useful for React apps.

Multi-stage (cont.)

- You can use a previous stage as intermediate build stage.

```
FROM alpine:latest AS builder
RUN apk --no-cache add build-base

FROM builder AS build1
COPY source1.cpp source.cpp
RUN g++ -o /binary source.cpp

FROM builder AS build2
COPY source2.cpp source.cpp
RUN g++ -o /binary source.cpp
```

Resources

- Deploying your own registry
- Docker Getting Started