cscc01 (summer 2021)
introduction to software engineering

tutorial 6

docker

tutorial outline

01 containers

02 docker

03 docker-compose

info

- **A1 Due this week!! June 18th 11:59 PM EST**

- **Sprint 2 Timeline**
  - **Start:** June 19th 12:00 AM EST
  - **Due:** July 2nd 11:59 PM EST
  - **Demo:** Week of July 5th

let’s begin with a story...

i want to create a chat system similar to Messenger/Discord/Slack/IRC, with client-server architecture

when one client sends a message, it goes to a server which processes it and then sends it to the correct recipient clients

i have my client and server software. users download the client and run it. the server will need to be run somewhere too.

this is the problem of deployment of services.

outline

needs: hardware, software dependencies, software

general responsibilities: add, remove, update

who's responsible?
me? my business? another business?

containers

containers are a complete runtime environment, a standard unit of software, with an application and all the required software to run it.

the runtime is abstracted from the environment required to run it

similar to virtualization, containers function to divide system resources between users

why containers?

over time, practices and supporting technology has evolved to meet the changing needs of service deployment.

from my home basement, to my business's server rooms, to another business's server rooms

containers are a part of this evolution!

dev environment

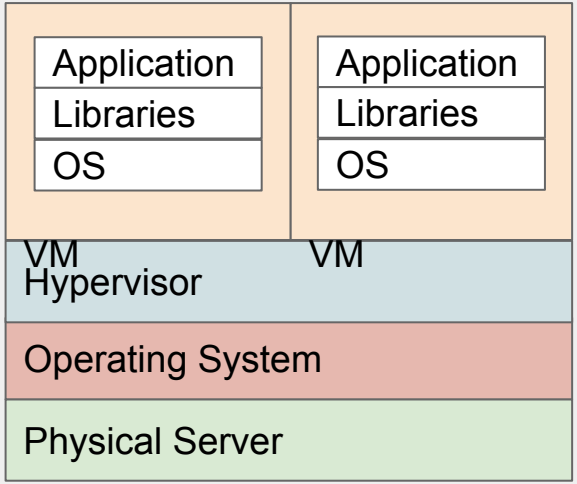
in addition, containers provide a standardized interface for dev environments.

this helps to

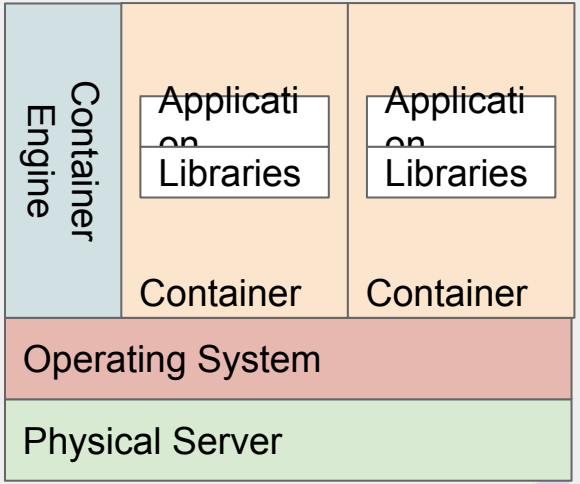
- reduce environment setup time
- allow developers to spin up/spin down services
- get rid of all claims of “bUt iT WOrKs oN My ComPUTer”

vms vs. containers

Virtual Machines



Containers



docker

docker is a Platform as a Service (PaAs) enabling users to deliver software in isolated packages called containers.

Containers run/execute **images**, which are files describing how to build and run a docker container.

docker builds images automatically by reading the instructions in a **dockerfile**

usage: docker build, docker run
other helpful commands: docker ps

dockerfiles

dockerfiles have the general format of:

INSTRUCTION arguments

- all dockerfiles should start with “FROM image:release-ver” to load the container OS image

```
1  FROM node:14-alpine
2
3  WORKDIR /client
4
5  COPY package*.json ./
6
7  RUN npm install
8
9  COPY . .
10 EXPOSE 3000
11 CMD [ "npm", "start" ]
```

React dockerfile

```
1  FROM maven:3.6.3-jdk-8
2
3  WORKDIR /root/.m2/repository
4  COPY . ./
5  RUN mvn verify clean --fail-never
6  RUN mvn compile
7  ENTRYPOINT [ "mvn","exec:java" ]
8  EXPOSE 8000
```

Java API dockerfile

demo

let’s see containers in action!

goal:

run a simple node.js service in a container, accessible externally

instructions:

```
cd docker
npm run d_build
npm run d_start
Visit http://localhost:3000/
```

docker-compose

docker-compose is a tool for defining and running multiple docker containers. It uses the docker-compose.yml file to configure the building and running of specified containers.
usage: docker-compose build, docker-compose up, docker-compose down
why?
we usually want to run multiple services when we create full stack applications

- frontend (React, Vue, Angular)
- backend (Node.js, Java, Go)
- databases (PostgreSQL, MySQL, MongoDB, CockroachDB, MariaDB)
- caching (Redis, Memcached)
- load balancing (Apache/Nginx)
- messaging (rabbitmq, kafka)

why? (cont'd)

- just like with development, we want to use a consistent interface for services.
- if I have Postgres 13 installed, and you have Postgres 9, odds are I will have more features than you, and if you try it locally, it will fail.

- compose makes this easy, we just spin up new services as needed
- that is only one part of the puzzle though, if we are deploying services for development we need to worry about:
  - networking
  - storage
  - orchestration

docker-compose.yml

Format:

version: “3.8” □ define docker-compose file version at the beginning, version needs to get quoted by double quotation.

services: □ list the service below with indentation

service_name: □ define a service

build: ./dir □ set the service build directory

dockerfile: Dockerfile-alternate □ if you have alternative dockerfile name, use this format.

args: □ arguments below with indentation

environment: □ environment variables below with indentation

volumes: □ mount real system directory to container, having the - <src>: <det> below with indentation

ports: □ port forwarding to the real system, having - “forward_port:container_port” below with indentation

docker-compose.yml

```yaml
version: "3.8"
services:
  frontend:
    stdin_open: true
    build:
      context: ./frontend
      dockerfile: Dockerfile
    ports:
      - 3000:3000
    volumes:
      - /frontend/node_modules
      - ./frontend:/frontend
    environment:
      - PORT=80
  locationmicroservice:
    build:
        context: ./LocationMicroservice
        dockerfile: Dockerfile
    depends_on:
      - neo4j
    ports:
      - "8000:8000"
  usermicroservice:
    build:
        context: ./UserMicroservice
        dockerfile: Dockerfile
    depends_on:
      - postgres
    ports:
      - "8001:8000"
  tripinfomicroservice:
    build:
        context: ./TripinfoMicroservice
        dockerfile: Dockerfile
    depends_on:
      - mongodb
    ports:
      - "8002:8000"
```

```yaml
mongodb:
  image: mongo
  environment:
    MONGO_INITDB_ROOT_USERNAME: root
    MONGO_INITDB_ROOT_PASSWORD: 123456
    MONGO_INITDB_DATABASE: trip
  ports:
    - "27017:27017"
postgres:
  image: postgres:latest
  environment:
    POSTGRES_PASSWORD: 123456
    POSTGRES_USER: root
  ports:
    - "5432:5432"
  volumes:
    - ./docker_postgres_init.sql:/docker-entrypoint-initdb.d/docker_postgres_init.sql
neo4j:
  image: neo4j:latest
  environment:
    - NEO4J_AUTH=neo4j/123456
  ports:
    - '7474:7474'
    - '7473:7473'
    - '7687:7687'
```

demo

let's see docker-compose in action!

goal:

run a containerized full stack application using:

- A Node.js backend
- A database instance of Postgres
- A React frontend

instructions:

```
cd compose
cd backend && npm i
cd frontend && npm i
cd ../ && docker-compose up --build
Visit http://localhost:3000/ (frontend may take some time)
```