Multi container acrchitecture:

To connect b/w one container and another we use this multi container architecture . We can make it happen using 2 ways

1. manual
2. docker compose

Docker Compose is a tool that allows you to define and manage multi-container Docker applications.

[eg . if ur application has 3 containers , how do u start those 3 containers , how do u manage those 3 containers , ya u have to start them together or stop them together , how do u work with them together ? and thats where docker compose comes into picture]

[why an application will have multiple container ? , thats nothing but a micro service application]

Micro service application :

Suppose if ur application might be composed of multiple services , eg ur application has frontend , backend

frontend = UI = php === This will be running in container1

backend = Db = mysql === This will be running in cntainr2

this is the best practice because rather than deploying all together in a single vm u can create seprate containers and deploy them that leads to less breakage of application.

These both containers should be communicating with one another or linked with one another which is done by docker compose . So docker

compose is used to launch microservice application.

This can be achieved by using yml file , in the yml file i define container1 and container2

Lets do the same in adhoc commands first then will go to compose

---> open aws terminal where docker is installed.

[delete all the images and containers]

---> docker system prune --all

deletes all the images unused (not attached to container) and not running containers

---> docker rm -f container-id

deletes running container

---> docker images

---> docker rmi img-id

---> docker images

---> docker ps -a

---> docker run -itd --name web -p 8001:80 devopstrainer/deploy:webapp

---> docker images

---> docker ps

---> open browser --- public-ip-of-instance:8001

[u can see a sample frontend , here if u add ur data and submit it will fail because db is not connected]

---> docker exec -it web /bin/bash

---> apt install nano

---> cd /var/www/html

---> ls

[here u can find the index.php]

---> nano index.php

[here u can find all details regarding the db it should br connected with , i.e servername =db , uid = root , password = example, dbname = docker , in this db we will store name and phnum]

[here we found whats happening in webapp container , now lets connect this container with db]

-[go to doker hub look for mysql , read overview to understand how to connect it to instance , environment variables etc ... give the values for environment variables according to how the values u have given in webapp container i.e

servername =db ,

uid = root ,

password = example,

dbname = docker]

---> docker run -itd --name db -e MYSQL_ROOT_PASSWORD=example -e MYSQL_DATABASE=docker mysql:5.6

---> docker images

---> docker ps

[container is created lets check wether the forntend is connected]

---> open browser --- public-ip-of-instance:8001 --- enter details --- submit

[error , means not yet connected]

---> docker exec -it db /bin/bash

---> mysql -u root -p

(enter to db)

Enter password : example

use docker;

show tables;

Create table emp(name VARCHAR(10), phone VARCHAR(11));

show tables;

[show display tables]

select * from emp;

[should display columns]

---> try connecting frontend (still error since we havent made the connection yet)

---> q

---> exit

---> ctrl-c -- exit

[u will reach instance terminal now]

---> docker ps

---> docker inspect db

[copy ip of db]

---> docker exec -it web /bin/bash

---> cd /var/www/html

---> vim index.php

[servername = "db" change it to

servername = "ip-of-container"]

---> ctrl P ctrl Q

---> cd ~

---> docker exec -it db /bin/bash

---> mysql -u root -p

enter the password:

use docker;

select * from emp;

[go to browser where web page is displayed

public-ip-of-instance:8001

enter the name and phne and submit]

select * from emp;

[now ur container is connected to db , ur front end is talking to backend , but here we did lot of steps manually for this simple project but what if u have complex project and complex process , doing manually will be hindrance]

----------Here the picture of docker compose comes , we can automate the process of starting multiple containers by docker compose here the dependencies will be taken care by compose , communication will be taken care by compose , we do not need to worry about setting up ip address , attaching volumes can also be done by docker compose .

---> docker rm -f db web

---> docker rmi mysql:5.6

---> docker system prune --all

---> docker-compose

[command not found , soo install docker compose using following below commands]

---> sudo apt update

---> sudo apt install curl

---> sudo curl -L "<https://github.com/docker/compose/releases/download/v2.15.1/doc

ker-compose-$(uname -s)-$(uname -m)>" -o /usr/local/bin/docker-compose

---> sudo chmod +x /usr/local/bin/docker-compose

---> docker-compose --version

---> create a yaml file where we define our container

---> mkdir /compose

---> touch docker-compose.yml

---> nano docker-compose.yml

```
version: '3.8'
 services:
   web:
     image: sreejitha1996/web_container:web1
     container_name: web_container
     ports:
       - "8080:80"
     environment:
        - DB_HOST=mysql
       - DB_PORT=3306
       - DB_NAME=app_db
       - DB_USER=app_user
```

```
      - DB_PASSWORD=app_password
    depends_on:
      - mysql
  mysql:
    image: sreejitha1996/mysql:8
    container_name: mysql_container
    environment:-
      - MYSQL_ROOT_PASSWORD=example
      - MYSQL_DATABASE=app_db
      - MYSQL_USER=app_user
      - MYSQL_PASSWORD=app_password
    volumes:
      -   mysql_data:/var/lib/mysql
volumes:
    mysql_data:   {}
```

---> docker-compose -f compose.yml config

[if it returns u docker file , then no error]

---> docker-compose up -d

[create link b/w container and attaches the container]

---> docker images

[docker-compose would have downloaded the image]

---> docker ps

[created and run both the container]

---> docker volume ls

[volume wld hve created under name compose_db_data]

---> docker inspect compose_db_data

[displays the path in mount section where the volume is stored, i.e]

---> cd /var/lib/docker/volumes/compose_db_data/_data

---> ls

[u find few files folders here that is present in container

now lets enter into container and check]

---> docker exec -it container-id /bin/bash

---> cd /var/lib/mysql

---> ls

[here u see same files and folders]

---

now lets see the o/p . To which port did we forword the application ?

to 8080

---> open browser --- public-ip-of-instance:8080

---> enter values --- submit [error coz no table]

---> aws terminal --- mysql -u root -p (example)

---> enter username and pwd of mysql db

---> use app_db;

---> clear

---> SHOW TABLES;

-->CREATE DATABASE IF NOT EXISTS app_db;

USE app_db;

CREATE TABLE IF NOT EXISTS users (

id INT AUTO_INCREMENT PRIMARY KEY,

username VARCHAR(255) NOT NULL UNIQUE,

password VARCHAR(255) NOT NULL

);

---> SHOW TABLES

---> INSERT INTO users (username, password) VALUES ('admin', '12345');

---> select * from users;

---> open browser --- public-ip-of-instance:8080

---> exit

[lets delete the container and see how volume works]

---> docker rm -f compose_db_1

---

yaml - aint markup language

It is data serialization language where data is represented in form of maps and list

--- map --- represented in form of ---key:value

--- lists --- represented in form of --- - (hypen)

indentation matters

About compose:

Compose will read the yml file and find the section given like service , volume in above example . It will read the file line by line and execute in sequential order . But If any depends-on parameter comes while reading microservice 1 then it will create the micro-service given under depends-on section.

eg . mysql service in above example will be created before web service . But to create mysql service we need volumes , thats how we scripted our docker-compose . So what compose will do ?

As step 1 it will noyt create web or mysql but it will create volume named mysql_data .Once volume is created db container will be created then web will be created.

If one microservice is dependent on more than one microservice we will

mention name of all those dependencies under depends-on key . It will execute as per line of execution

Version:

latest version of docker-compose

Services and Volumes are the 2 sections i have created

Services :

These define my containers

Volumes:

These are docker volumes created for data persistence . Name of volume is mysql_data. I can have multiple volumes under volume tab

Services:

I scripted in yaml that i have 2 microservices ,

1. web

2. mysql

1. web:

image : pulls image from dockerhub if image is not found locally.

container_name : allots name to container

port : run your application in mentioned port

environment : this describes about the service this micro-service is going to attach

db_host : enter name to the service this micro-service is going to get connected to .
eg mysql in above yaml file

db_port , db_name, db_user, db_password:

enter details of service u gonna attach to this service

depends-on: this service depends on which service

2. mysql:

all the sections similar as above except

volumes: here u will attach the path where ur data will be stored to volumes variable created using volumes section.

----------Practise 2:

```
version:'3.8'
services:
    wordpress:
        depends_on:
            - database
        image: wordpress:latest
        ports:
```

```
            - 80
        environment:
            WORDPRESS_DB_HOST: database:3306
            WORDPRESS_DB_NAME: wp_db
            WORDPRESS_DB_USER: wordpress
            WORDPRESS_DB_PASSWORD: wordpress123
    database:
        image: mysql:5.7
        environment:
            MYSQL_ROOT_PASSWORD: root123
            MYSQL_DATABASE: wp_db
            MYSQL_USER: wordpress
            MYSQL_PASSWORD: wordpress123
        volumes:
            - db-vol: /var/lib/mysql
volumes:
    db-vol: {}
networks:
  wp-network:
    driver: bridge
```

---> docker-compose up -d

[executes the services and create containers accordingly]

---> docker ps

---> docker volume ls

---> docker exec -it mysql-container bash

---> mysql -u root -p root123

---> SHOW DATABASES;

[here database called wp_db is created as a part of yaml]

---> use wp_db;

---> SHOW TABLES;

[no table because front-end havent pushed any data yet]

---> ctrl p ctrl q

---> docker ps

---> copy port of wordpress container

---> public-ip-of-instance:copied-port

---> enter details and press install wrdpress

---> use the username and password and login

---> docker exec -it mysql-container bash

---> SHOW TABLES;

---> select * from wp_users;

---

here both ur front-end micro-service and backend micro-service are connected to each other . this happened with the help of docker network .

---> docker network ls

here u can c the default docker network created , u can also create custom docker n/w ,

---

Drawbacks of docker-compose:

When 2 containers are running somebody accidently deleted a container , for example from above scenario lets delete wordpress container

---> docker rm -f container-id

---> docker ps

now i cannot access my application , my front-end is gone , is there any fault-tolerance ? No because once a container deleted then its deleted , my container doesnt comes online alternatively . u have to manualy run the compose again.

---> docker-compose up -d

---> docker ps

So here we have no fault tolerance

1. Fault tolerance means:

    Automatically recovering from failure .

2. Single point of failure:(SPOF)

What happens if i stop this instance , the containers will die . i.e if node gets corrupted or node went offline the containers will be eliminated.

So the solution for this is i need a container Orchestrator to provide me high availability and fault tolerance for my containerized applictions over multiple nodes. ANd thats where docker swarm and kuberernetes comes to picture.

Docker swarm and kuberernetes are container orchestration tools that provides u haigh availability and fault tolerance .

If container is failed then alternative container will be created if node is failed then container will be moved to other nodes . So here we need cluster of nodes . And to overcome all the drawbacks we listed above we need orchestration tool like docker swarm or kubernetes

---