DOCKER RESOURCES

1. What is Docker? - India (4)
2. আমার ডকার শেখা/ বাংলায় ডকার
3. Docker in 100 Seconds (Quick introduction) (1)
4. Orientation and setup (Official documentation || If anyone feels more comfortable with video guides, I recommend watching the video walkthrough available here.) (6)
5. ডকার ও লিনাক্স কন্টেইনার — ভার্চুয়ালাইজেশনের নতুন রূপ | by Cyan Tarek | প্রোগ্রামিং পাতা (Container & Docker intro) (2) (Bangla)
6. ডকার এ পাইথন ওয়েব এপ্লিকেশন চালানো এবং ডকার এর মূল ধারণা (Docker with examples. Good read. There are some references too.) (3) (Bangla)
7. Learn Docker in 7 Easy Steps - Full Beginner's Tutorial (Recommended) (5)
8. Docker Tutorial for Beginners - A Full DevOps Course on How to Run Applications … (freeCodeCamp :D)
9. https://laracasts.com/series/guest-spotlight/episodes/2?fbclid=IwAR0FUHQmey-dpzyJM0TF5imzrtGy1CWA2XGzntUN7LOFs-2H6dShxSXBzbI
10. Building Docker Images with Dockerfiles - (Must read while building first own image)

DOCKER RESOURCES

Demo: https://hub.docker.com/r/hijibijee/guess_square_root
How to run: docker run -it hijibijee/guess_square_root:1
Source code: https://github.com/hijibijee/Docker-practice

What is Docker?

Docker is a tool that containerizes different components of a project that can run on the same underlying OS. So for a new developer using a different environment does not need to worry about setting up all the dependencies required, simply a docker run will do the work and it is guaranteed to run in the same way everywhere.

Why use Docker?

- Package and containerize applications and ship them, run them anywhere anytime as many times as the user wants.
- Docker containers are lightweight and faster to boot up than Virtual Machines.
- Images of most common OS, database, other services and tools are already available on docker hub.
- Everything can be containerized. e.g. OS, browser, apps, simple programs, database…
- No headache of installing an item, simply run it using docker.

Which projects should be dockerized?

- Large projects that consist of separate teams for each component / feature. Dockerizing the work of an individual team will ease the process of accessing / using that part for other teams.
- Projects that need to be maintained for a long period.
- Teams that consist of developers with a variety of tastes in OS. One might be comfortable using windows, others might love ubuntu. .

Which projects should not be dockerized?

- Small projects. Docker might add additional complexities for the developers.
- If we need to speed things up. Docker does not take any special care to speed things up. Running too many containers may use a lot of CPU resources. (While practicing docker I was trying to run my code in IntelliJ IDEA with different versions of Java using docker. What I didn't notice is, though my code failed to run due to version mismatch, an image was created for every version in my laptop. All of a sudden my poor laptop started slowing down and C drive was flooding.)
- Projects that deal with valuable data. Once a container is destroyed, all of its data within it gets destroyed. We need to map a local repository that will store these data even after that container is destroyed.
- Ref: https://www.freecodecamp.org/news/7-cases-when-not-to-use-docker/

DOCKER RESOURCES

Self Notes:

Container: Container is a tool that provides environment and dependencies in an isolated way. Setting up a container is a low level complex work. Docker provides a high level tool and functionalities to make this easy. Docker uses Alex C container. A container only lives as long as a process in it is running. If you run a container of ubuntu it will exit immediately.

Image: Image is a template / package used to create containers. Containers are running instances of an image.

Basic commands:

- Check running containers: docker ps
- Check available images: docker images
- Check all containers: docker ps -a
- Stop a running container: docker stop container_name (or container_id)
- Remove a container: docker rm container_name (or container_id)
- Remove a image: docker rmi image_name (all of its container must be removed)
- Run a container: docker run image_name
- Pull a image into host: docker pull image_name
- Execute a command: docker exec container_name command _to_exec
- Run using tag: docker run image_name:tag
- Run and listen to std in: docker run -it image_name
- Port mapping: docker run -p 80:5000 image_name (port 80 mapped to port 5000)
- Volume mapping: docker run -v out_directory:container_directory imageName
- Get additional details: docker inspect container_name (or id)
- Logs: docker logs container_name (or id)
- Set environment variable: docker run -e var_name = value image_name
- To run in detached mode -d

How to build own image:

- Note down all the things to run our application if we had to do it manually
- Create a Dockerfile (without any extension)
- Write the Dockerfile in “Command Argument” manner
- All Dockerfile must start with a FROM command. It indicates the base OS or other image available in docker hub. (In my case it was FROM jdk:15)
- RUN command runs a particular command to install all the dependencies.
- COPY copies source code on to the docker image
- ENTRYPOINT specifies the command in order to run the specified container.

DOCKER RESOURCES

- After creating the Dockerfile, run “docker build -f Dockerfile -t image_name:tag .” in the command prompt to build the image. Image_name must consist of all lowercase letters.
- Run “docker push image_name:tag” to push to docker hub.

Docker builds images in layered architecture. So if it fails to build in any particular step it will rebuild again from that step, no need to rebuild those that are already built.

CMD vs ENTRYPOINT: If we use additional commands in docker run, CMD gets overwritten whereas in case of ENTRYPOINT it gets appended.

Networking in Docker:
Docker automatically creates three network:

- Bridge:
  - Default network (docker run image_name)
  - Private internal network created by docker on the host.
  - Each container gets an internal IP address and can access each other using this.
  - To access from the outside world Map port of the container to a port of the docker host.

- Host:
  - docker run image_name --network = host
  - Associate container to the host network
  - No need to map port

- None:
  - docker run image_name --network = none
  - totally isolated.

DOCKER RESOURCES

Task:

- Run MySQL docker image in a container.
- Create a table “User” [“name”, “id”, “mobile number”]
- Make an app in Node.js / Java that will request get post to update the database.

I will complete this task using Node.js to learn Node.js. For now, the most friendly site to learn these for me is W3schools.

1. **Running a container of mysql image:**
   - **docker run --name** container_name **-p 8080:3306 -e MYSQL_ROOT_PASSWORD=**pass **-it mysql**
     (Here we are setting the name of our container using --name. 3306 is the default for mysql inside the container. We are mapping it to port 8080 of localhost using -p)

2. **Preparing database:** Open a separate command prompt (as we are in attached mode)
   - **docker exec -it** container_name **bash**
     (We are now inside the container)

   - **mysql -u root -p**pass
     (Notice no space between -p and password)
   - **create database** database_name;
   - **use** database_name;
   - create & prepare tables

3. Write necessary codes to connect database server using Node.js

4. **Fixing Error (ER_NOT_SUPPORTED_AUTH_MODE Client does not support authentication protocol requested by server error):**
   Go to the cmd prompt where we were running mysql commands. Then run following commands:
   - **ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY '**password**';**
   - **flush privileges;**

   Try to connect now. In my case it was still showing the error. After removing the @’localhost’ part it was resolved.

   Reference: [MySQL 8.0 - Client does not support authentication protocol requested by server; consider upgrading MySQL client]

MySQL database is now connected to Node.js. :D

Demo app

Source code: https://github.com/hijibijee/MySQL-Node-js-Docker-practice-

Description: This web app maintains a MySQL database in a docker container. It has a table ‘user’ with two fields ‘id’ (primary key) and ‘name’. In the frontend, it has two textboxes to receive inputs (id, field), three buttons to insert, delete and update the database, and a text field to show a response from the backend. The backend handles these operations and provides appropriate responses. Invalid input, blank input, and other issues are dealt with in the backend.