Docker Image & Containers

Author Name: David Darigan
Author Student ID: C00263218

1. Starting & Stopping

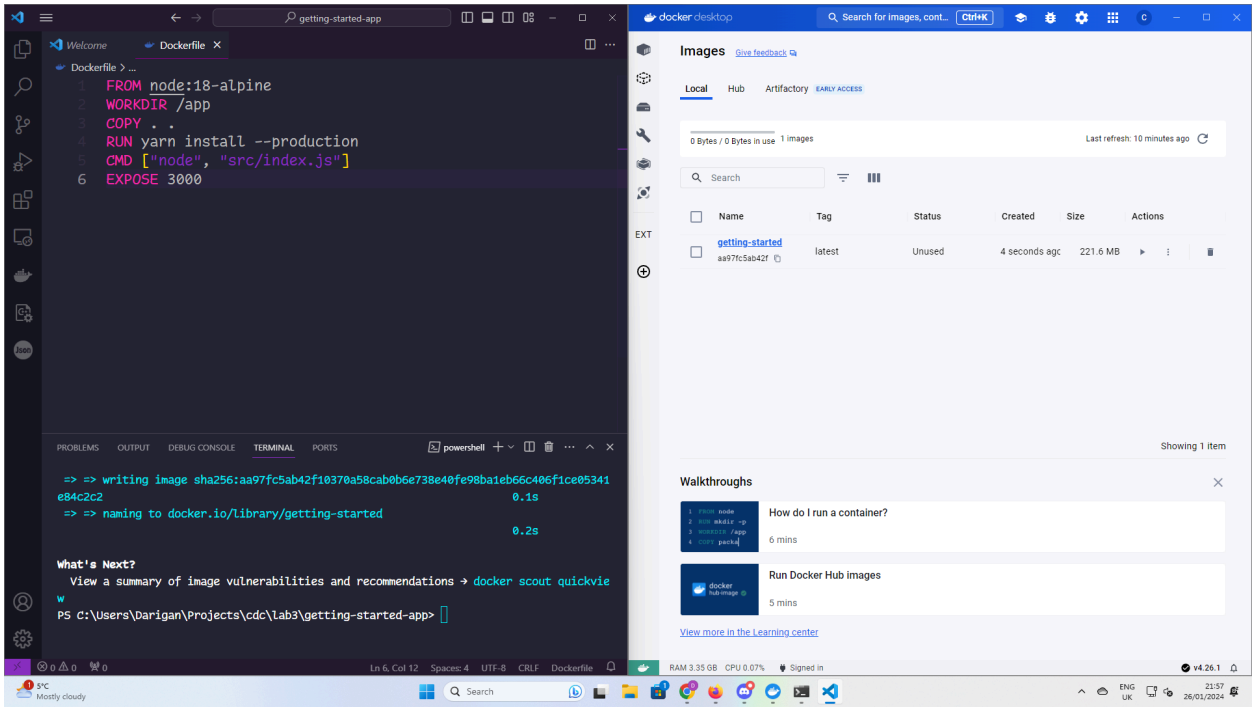
Example Docker File

```
# Source Image (usually an Operating System)
FROM node:18-alpine

# Directory where files are stored inside our container
WORKDIR /app

# Command to copy files from our local folder on..
# ..local machine to local folder in the container
COPY . .

# Running the build command of the yarn build tool
RUN yarn install --production

# Executing a terminal command to run a node server
CMD ["node", "src/index.js"]

# Exposing our image port so outside images can attach to it
EXPOSE 3000
```

Build a Docker Image

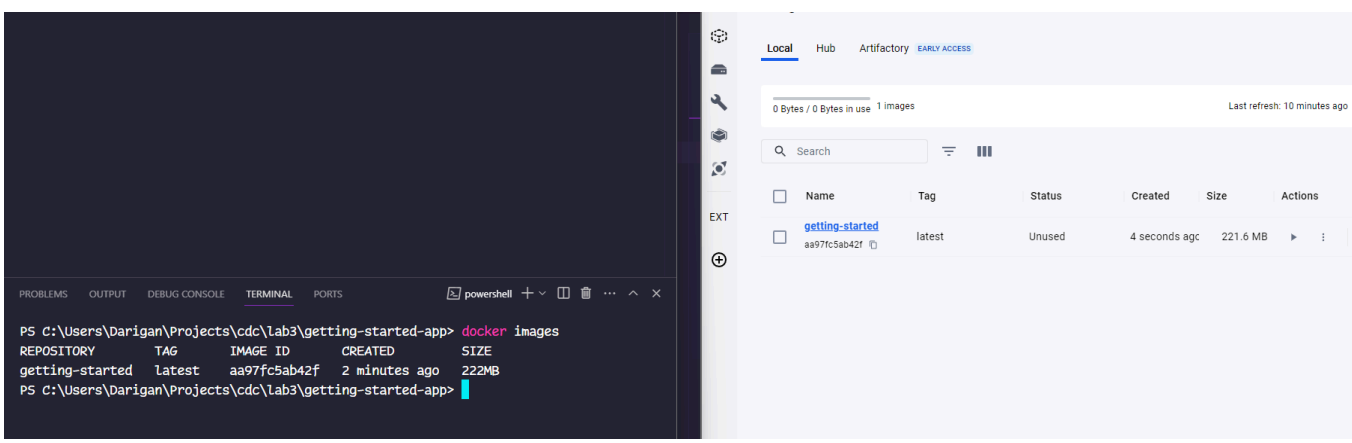
```
docker build -t <image_name> .
```

(Run this in the same directory as the Dockerfile)



View Docker Images

```
docker images
```



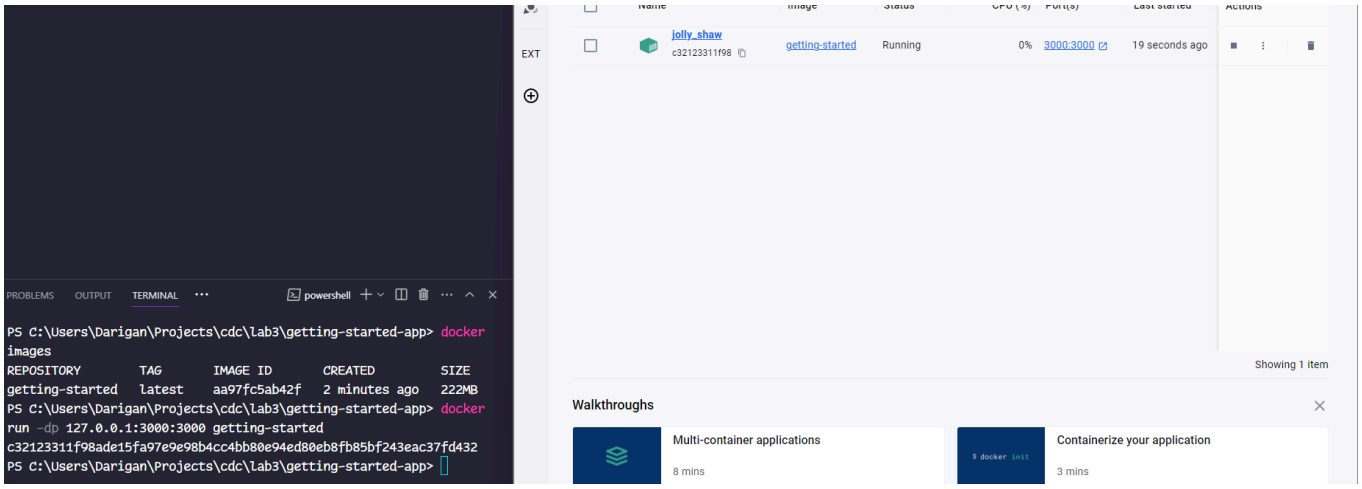
Run a Container

```
docker run -dp 127.0.0.1:3000:3000 getting-started
```

Notes:

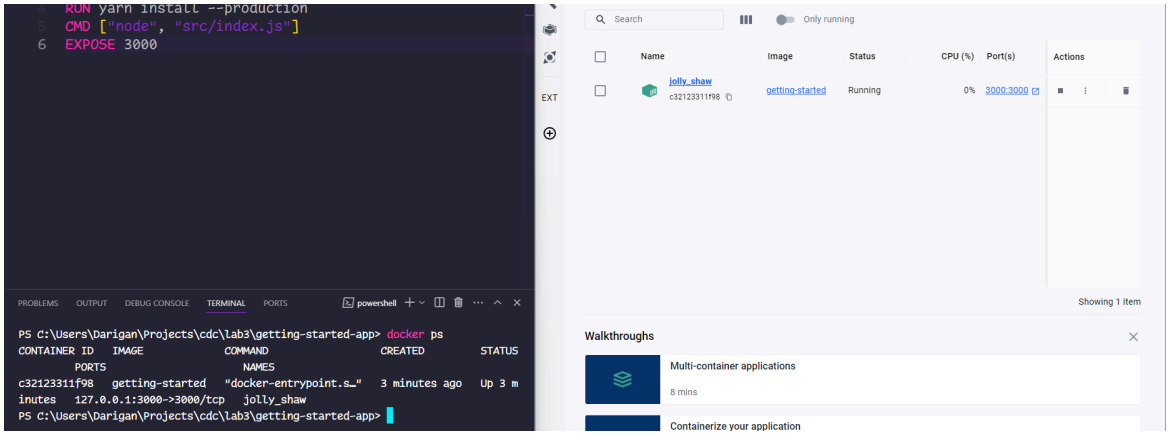
-d stands for “detach” which runs the container in the background

-p stands for “publish” which allows us to map our localhost 3000 port with the exposed 3000 port of the Dockerfile (therefore anything running on 3000 in the Dockerfile will run on port 3000 in our local host)



View Running Containers:

```
docker ps
```



Stop a Running Container

```
docker stop <the-container-id>
```

Remove a Container

```
Docker rm <the-container-id>
```

```
STATUS          PORTS                       NAMES
c32123311f98   aa97fc5ab42f   "docker-entrypoint.s…"   15 minutes ago   Up 23 seconds   127.0.0.1:3000->3000/tcp   jolly_shaw
PS C:\Users\Darigan\Projects\cdc\lab3\getting-started-app> docker stop c32123311f98
c32123311f98
PS C:\Users\Darigan\Projects\cdc\lab3\getting-started-app> docker rm c32123311f98
c32123311f98
PS C:\Users\Darigan\Projects\cdc\lab3\getting-started-app>
```

2. Persistence & Volumes

Containers do not share data with each other by default. If you start two containers from the same image, that is 2 separate instances of that image with their own process environment. Docker introduces the ideas of “Volumes” which are data storage containers on the host system for the containers to take advantage of.

Create a Volume

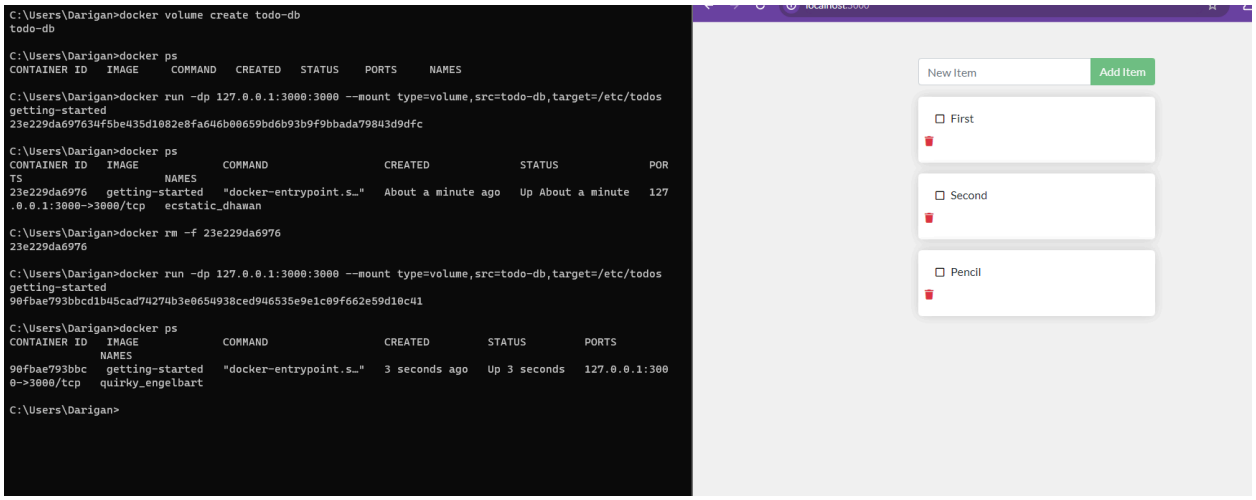
```
docker volume create <volume_name>
```

Use a Volume with a Container

```
docker run -dp 127.0.0.1:3000:3000 --mount type=volume,src=todo-db,target=/etc/todos getting-started
```

Notes:

--mount type=volume specifies a volume to mount
src=todo-db means that the volume to use is the todo-tb
target=/etc/todos means that any files created in at that path in the container will be captured by the volume



(items were created in the first Container and retrieved by the second)

Inspect a Volume

```
docker volume inspect <volume-name>
```

```
C:\Users\Darigan>docker volume inspect todo-db
[
    {
        "CreatedAt": "2024-01-26T22:21:41Z",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/todo-db/_data",
        "Name": "todo-db",
        "Options": null,
        "Scope": "local"
    }
]
```

Volume & Bind Mounts

- Volume Mount is used to store data persistently
- Bind Mounts can be used to access data from host system

If you create data in your app and want to store it somewhere, you store it in a volume. It doesn't really matter where it is provided you can save the data.

Bind Mounts are inverse, where they want to acquire new data from outside their isolated environment. You will need to enable docker file sharing with the directories in questions

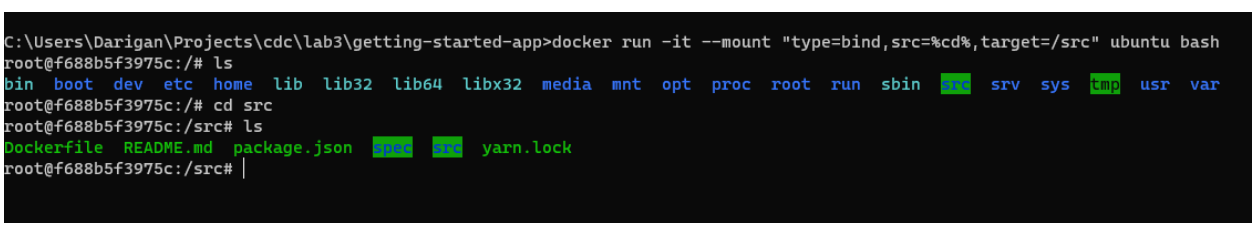
Run Shared File Session

```
docker run -it --mount "type=bind,src=%cd%,target=/src" ubuntu bash
```

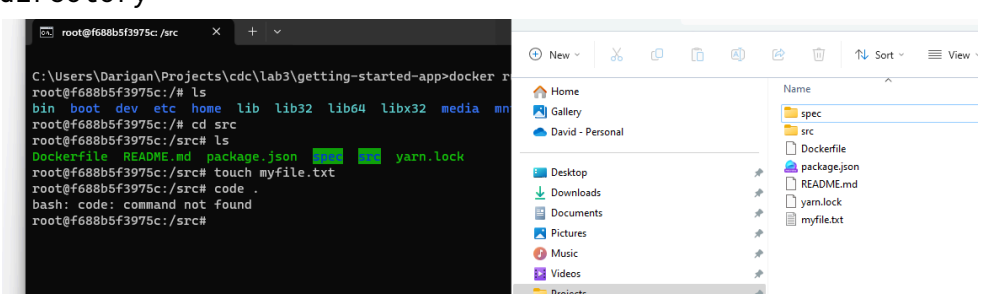
Notes:

src="%cd"% is setting the source folder (%cd% will give us the current directory)

target=/src is setting the target folder where we will access the data from the container side



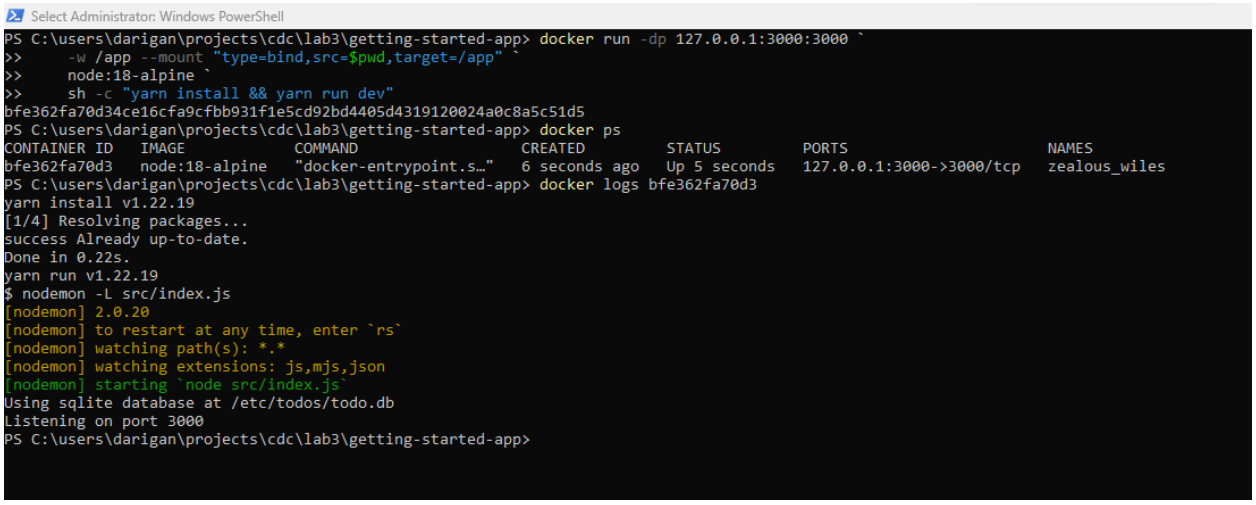
You can create files on the container and they will appear in the host directory



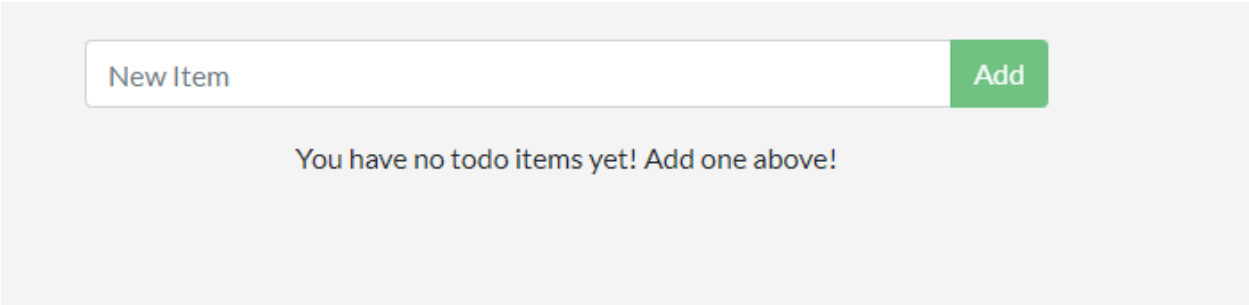
Run Container with Bounded Workspace

```
docker run -dp 127.0.0.1:3000:3000 `-w /app
--mount "type=bind,src=$pwd,target=/app" `
node:18-alpine `
sh -c "yarn install && yarn run dev"
```

```
-w stands for workspace (in this case “/app”
-- mount is the same as previous
Node:18-alpine is our target os
The last command executes the build and run step of the container
```



If you edit the src code now, and reload, you should see the changes take effect.



3. Multi-Container Apps

- Containers should do a single thing well
- Multiple Containers handle multiple services
    - Website Frontend
    - Website Backend
    - Persistence Layers (MySQL)

Container Networking

Containers can talk to each other if they are running on the same network.

Start a Docker Network

```
docker network create <network_name>
```

Starting a MySQL Container on the same network (Windows)

```
  docker run -d ^
  --network <network_name> --network-alias mysql ^
-v todo-mysql-data:/var/lib/mysql ^
-e MYSQL_ROOT_PASSWORD=secret ^
-e MYSQL_DATABASE=todos ^
mysql:8.0
```

```
C:\Users\Darigan\Projects\test>docker network create todo-app
063df1988db17ed99845070f14db189771838e4fe7be6eb2b17b214ef4f9ca44

C:\Users\Darigan\Projects\test>docker run -d ^
More?     --network todo-app --network-alias mysql ^
More?     -v todo-mysql-data:/var/lib/mysql ^
More?     -e MYSQL_ROOT_PASSWORD=secret ^
More?     -e MYSQL_DATABASE=todos ^
More?     mysql:8.0
Unable to find image 'mysql:8.0' locally
8.0: Pulling from library/mysql
558b7d69a2e5: Pull complete
c7e714ea5470: Pull complete
508ada0981f6: Pull complete
1ae11d1b9d69: Pull complete
3c9be74ddc84: Pull complete
d4a59c718252: Pull complete
ac42afee6c9c: Pull complete
dcf1f586b2e2: Pull complete
210e58b61fa3: Pull complete
d9719ad703de: Pull complete
1ef92b20c8ec: Pull complete
Digest: sha256:3f75dccd64fffa40a06a4a9256206280a5ddc3e26dea3f1ab0df35b2cc12f472
Status: Downloaded newer image for mysql:8.0
20c6fd9ce77744f2eed8de58fd0d7295ef9ab096d55bcb2c2abadeb93bfeb393
```

Confirm MySQL DB Container is running

```
docker exec -it <mysql-container-id> mysql -u root -p
```

```
C:\Users\Darigan\Projects\test>docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED         STATUS         PORTS                   NAMES
20c6fd9ce777   mysql:8.0   "docker-entrypoint.s…"   5 minutes ago   Up 5 minutes   3306/tcp, 33060/tcp     wonderful_swanson

C:\Users\Darigan\Projects\test>docker exec -it 20c6fd9ce777 mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.36 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| todos              |
+--------------------+
5 rows in set (0.00 sec)

mysql>
```

Exit MySQL DB

```
mysql> exit
Bye
```

Find MySQL Container IP using netshoot image

- Each container has its own IP Address
- Netshoot is an image with a number of networking related tools

Running the netshoot image on the todo-app network

```
docker run -it --network todo-app nicolaka/netshoot
```

```
C:\Users\Darigan\Projects\test>docker run -it --network todo-app nicolaka/netshoot
                    dP             dP                           dP
                    88             88                           88
88d888b. .d8888b. d8888P .d8888b. 88d888b. .d8888b. .d8888b. d8888P
88'  `88 88ooood8   88   Y8ooooo. 88'  `88 88'  `88 88'  `88   88
88    88 88.  ...   88         88 88    88 88.  .88 88.  .88   88
dP    dP `88888P'   dP   `88888P' dP    dP `88888P' `88888P'   dP

Welcome to Netshoot! (github.com/nicolaka/netshoot)
Version: 0.11

4c3cf95728f3 ~
```

Dig MySQL

```
4c3cf95728f3 ~ dig mysql

; <<>> DiG 9.18.13 <<>> mysql
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 28647
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;mysql.                         IN      A

;; ANSWER SECTION:
mysql.                  600     IN      A       172.18.0.2
```

172.18.0.2 is the IP of our MySQL Container.

Run App with MySQL

Prerequisites Environment Variables:

- MYSQL_HOST
  - The hostname for the running MySQL server
- MYSQL_USER
  - The username to use for the connection
- MYSQL_PASSWORD
  - The password to use for the connection
- MYSQL_DB
  - The database to use once connected

Note: Typical to use env vars for this information but is discouraged by docker itself, instead it is suggested to use secret support provided by your chosen container orchestration.

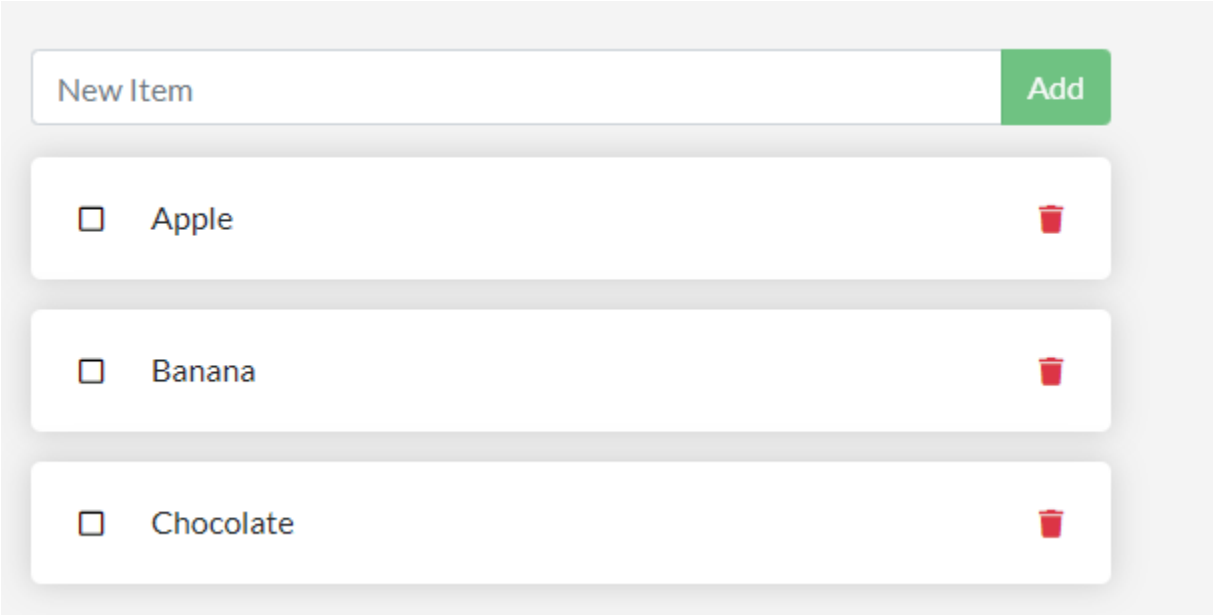
Run Container (from dir with the Dockerfile for the app) while providing env variable values for MySQL

```
PS C:\Users\Darigan\projects\cdc\lab3\getting-started-app> docker run -dp 127.0.0.1:3000:3000 `
>>   -w /app -v "$(pwd):/app" `
>>   --network todo-app `
>>   -e MYSQL_HOST=mysql `
>>   -e MYSQL_USER=root `
>>   -e MYSQL_PASSWORD=secret `
>>   -e MYSQL_DB=todos `
>>   node:18-alpine `
>>   sh -c "yarn install && yarn run dev"
a202e2ddd91d9a9275b8cbea6a72fffe05fcca4a50f5c73c51c65b857a22e43e
PS C:\Users\Darigan\projects\cdc\lab3\getting-started-app>
```

Checking that the Container has connected to MySQL

```
PS C:\Users\Darigan\projects\cdc\lab3\getting-started-app> docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED             STATUS              PORTS                     NAMES
a202e2ddd91d   node:18-alpine     "docker-entrypoint.s…"   About a minute ago  Up About a minute   127.0.0.1:3000->3000/tcp  quirky_banzai
4c3cf95728f3   nicolaka/netshoot  "zsh"                    13 minutes ago      Up 13 minutes                                 cranky_dewdney
20c6fd9ce777   mysql:8.0          "docker-entrypoint.s…"   26 minutes ago      Up 26 minutes       3306/tcp, 33060/tcp       wonderful_swanson
PS C:\Users\Darigan\projects\cdc\lab3\getting-started-app> docker logs -f a202e2ddd91d
yarn install v1.22.19
[1/4] Resolving packages...
success Already up-to-date.
Done in 0.34s.
yarn run v1.22.19
$ nodemon -L src/index.js
[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node src/index.js`
Waiting for mysql:3306.
Connected!
Connected to mysql db at host mysql
Listening on port 3000
```

Open the App & add a few items



Login to the MySQL Database (replace id with your container ID, password is secret)

```
C:\Users\Darigan>docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED         STATUS          PORTS                     NAMES
a202e2ddd91d   node:18-alpine     "docker-entrypoint.s…"   3 minutes ago   Up 3 minutes    127.0.0.1:3000->3000/tcp  quirky_banzai
4c3cf95728f3   nicolaka/netshoot  "zsh"                    15 minutes ago  Up 15 minutes                             cranky_dewdney
20c6fd9ce777   mysql:8.0          "docker-entrypoint.s…"   28 minutes ago  Up 28 minutes   3306/tcp, 33060/tcp       wonderful_swanson

C:\Users\Darigan>docker exec -it 20c6fd9ce777 mysql -p todos
Enter password:
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
```

Checking that the items were added

```
mysql> select * from todo_items;
+--------------------------------------+-----------+-----------+
| id                                   | name      | completed |
+--------------------------------------+-----------+-----------+
| dcda9af3-2ac7-46d8-bb82-970b243dd069 | Apple     |         0 |
| 7fa714a1-a561-454b-a042-2bf35f47e8ce | Banana    |         0 |
| 67de9e7e-d6b7-47a2-9c11-2b2238473999 | Chocolate |         0 |
+--------------------------------------+-----------+-----------+
3 rows in set (0.00 sec)

mysql>
```

Docker Compose

Docker Compose is essentially a file that contains all of your commands entered manually as stored in a compose.yaml file. This file can be stored at the root of your project (beside the original Dockerfile)

Adding app Service

```yaml
# Services are all the different containers we want to run
services:

  # Our initial App Service
  app:
    image: node:18-alpine
    command: sh -c "yarn install && yarn run dev"
    ports:
      - 127.0.0.1:3000:3000

    # Working Dir relative and our volumes for persistence
    working_dir: /app
    volumes:
      - ./:/app

    # Defining Enviroment Variables
    environment:
      MYSQL_HOST: mysql
      MYSQL_USER: root
      MYSQL_PASSWORD: secret
      MYSQL_DB: todos
```

Adding MySQL Service

```yaml
  # Defining our MySQL Service
  mysql:
    image: mysql:8.0

    # Defining our MySQL Mapping (this happened automatically
    # ..but requires us to be explicit here)
    volumes:
      - todo-mysql-data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: secret
      MYSQL_DATABASE: todos

# Defining all of the volumes we're using, you can see this referenced in the MySQL Services
volumes:
  todo-mysql-data:
```

Entire compose.yaml file

```yaml
services:
  app:
    image: node:18-alpine
    command: sh -c "yarn install && yarn run dev"
    ports:
      - 127.0.0.1:3000:3000
    working_dir: /app
    volumes:
      - ./:/app

    environment:
      MYSQL_HOST: mysql
      MYSQL_USER: root
      MYSQL_PASSWORD: secret
      MYSQL_DB: todos

  mysql:
    image: mysql:8.0
    volumes:
      - todo-mysql-data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: secret
      MYSQL_DATABASE: todos

volumes:
  todo-mysql-data:
```

Running Containers via Compose

docker compose up -d

```
C:\Users\Darigan\Projects\cdc\lab3\getting-started-app>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

C:\Users\Darigan\Projects\cdc\lab3\getting-started-app>docker compose up -d
[+] Running 4/4
 ✓ Network getting-started-app_default           Created
 ✓ Volume "getting-started-app_todo-mysql-data"  Created
 ✓ Container getting-started-app-mysql-1         Started
 ✓ Container getting-started-app-app-1           Started

C:\Users\Darigan\Projects\cdc\lab3\getting-started-app>
```

Note: You can use ‘docker compose logs -f’ to check the stream of all the containers together. This allows you see which triggered first, in case there any situations like trying to connect to a database that isn’t initialized yet.

Shut Composed Services Down

docker compose down

```
PS C:\Users\Darigan\projects\cdc\lab3\getting-started-app> docker compose down
[+] Running 3/3
 ✓ Container getting-started-app-app-1    Removed
 ✓ Container getting-started-app-mysql-1  Removed
 ✓ Network getting-started-app_default    Removed
PS C:\Users\Darigan\projects\cdc\lab3\getting-started-app>
```

Use “--volumes” flag if you want to remove the volumes as well