

Getting Started with Docker

Containerize an application:

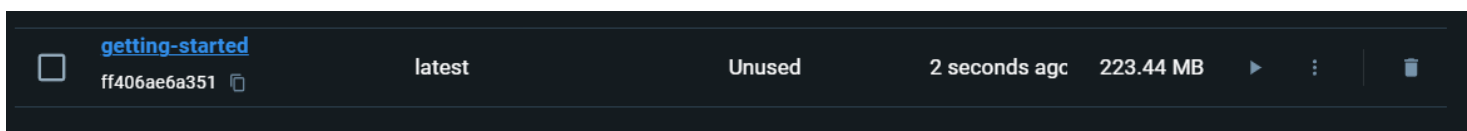
- Using 'git clone <https://github.com/docker/welcome-to-docker>' to pull the getting started repository into WSL directory.

```
conor@Dev-System:~$ git clone https://github.com/docker/getting-started-app.git
Cloning into 'getting-started-app'...
remote: Enumerating objects: 68, done.
remote: Counting objects: 100% (31/31), done.
remote: Compressing objects: 100% (22/22), done.
remote: Total 68 (delta 10), reused 9 (delta 9), pack-reused 37
Receiving objects: 100% (68/68), 1.75 MiB | 3.71 MiB/s, done.
Resolving deltas: 100% (11/11), done.
```

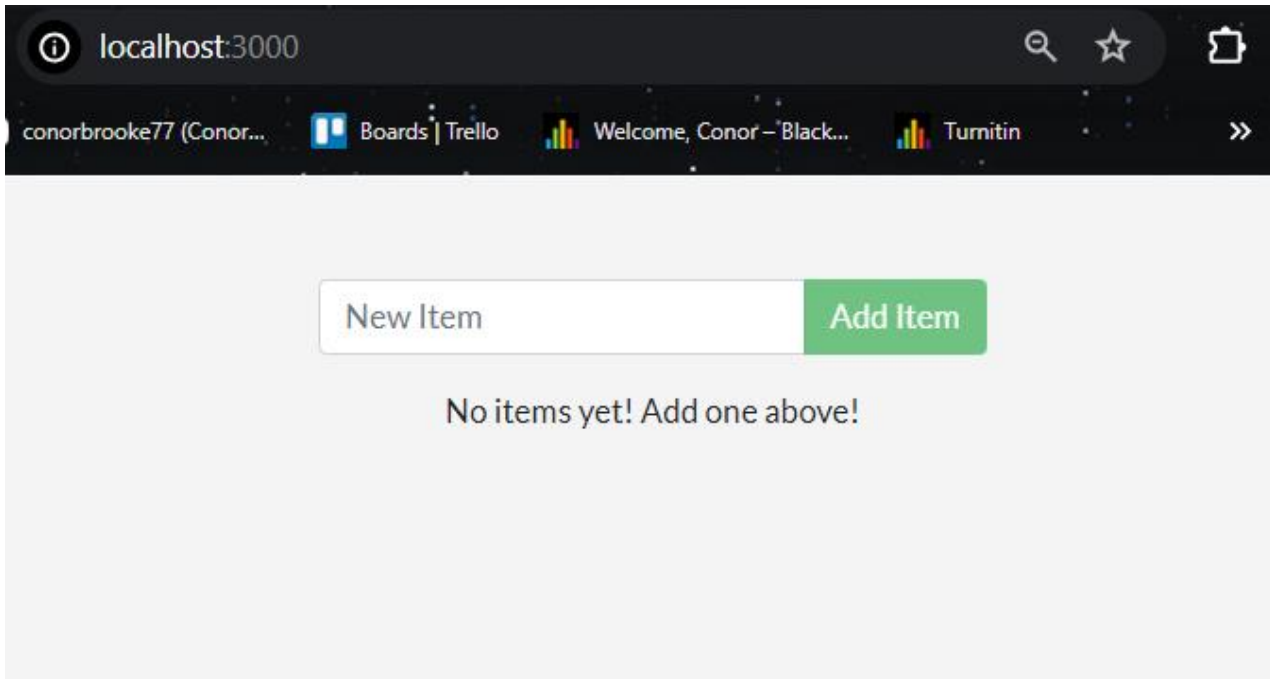
- Creating my Dockerfile with the given properties

```
1  # syntax=docker/dockerfile:1
2
3  FROM node:18-alpine
4  WORKDIR /app
5  COPY . .
6  RUN yarn install --production
7  CMD ["node", "src/index.js"]
8  EXPOSE 3000
```

- Building the docker image using the Dockerfile



- Ran the container using the 'docker run' command and opened my web browser to <http://localhost:3000>



- Used the following 'docker ps' command in the terminal to display my new container.

```
conor@Dev-System:~/getting-started-app$ docker ps
```

CONTAINER ID	IMAGE	NAMES	COMMAND	CREATED	STATUS	PORTS
bf25036fa8c9	getting-started		"docker-entrypoint.s..."	4 minutes ago	Up 4 minutes	127
.0.0.1:3000->3000/tcp		pensive_solomon				

Summary

In this section, I learned the basics about creating a Dockerfile to build an image. After building an image, I started the new container and saw the running app.

Update the application:

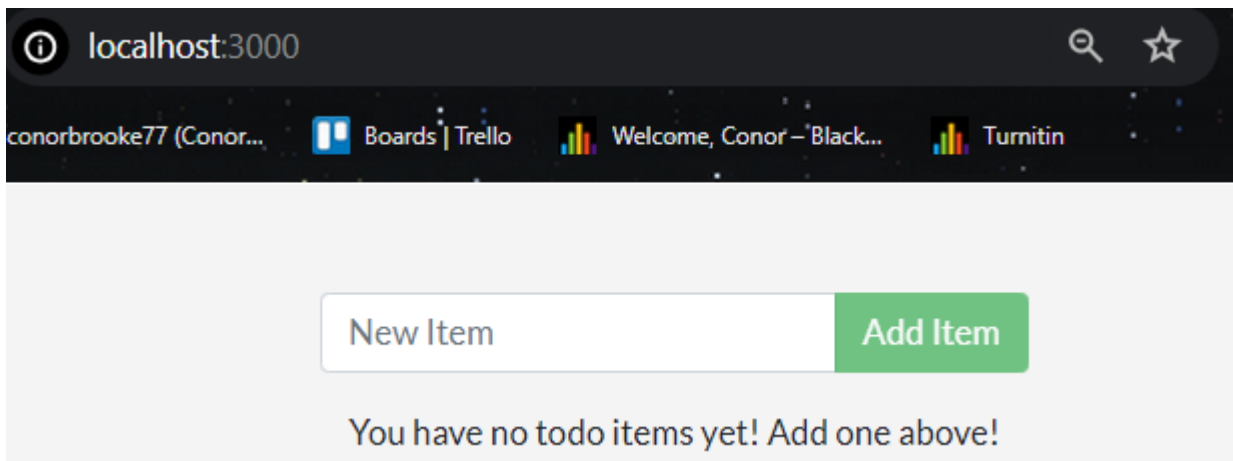
- Updated the source code '`<p className="text-center">You have no Todo items yet! Add one above!</p>`'
- Built an updated version of the image.
- Tried to start a new container using the updated code, but got the error below:

```
conor@Dev-System:~/getting-started-app$ docker run -dp 127.0.0.1:3000:3000 getting-started
ec52c04fb3b38c74580db5a992e5afd263e2d60e0a7905e60e8186423e8aea82
docker: Error response from daemon: driver failed programming external connectivity on endpo
int flamboyant_benz (771256678d0c799ed8c41ee7bd1d915b5aa38ee3dd19a76e42ad6137013da028): Bind
for 127.0.0.1:3000 failed: port is already allocated.
```

Solution to error, stop and remove the old container:

```
conor@Dev-System:~/getting-started-app$ docker stop pensive_solomon
pensive_solomon
conor@Dev-System:~/getting-started-app$ docker rm pensive_solomon
pensive_solomon
conor@Dev-System:~/getting-started-app$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES
conor@Dev-System:~/getting-started-app$
```

- Started the updated app container with the updated code:



Summary

In this section, I learned how to update and rebuild a container, as well as how to stop and remove a container.

Persist the DB:

- To see a containers “scratch space” in action, I started two containers and created a unique file in each.
- In a new ubuntu container, I created a file named random.txt using the command ‘docker run -d ubuntu bash -c "shuf -i 1-10000 -n 1 -o /random.txt && tail -f /dev/null"’

```
conor@Dev-System:~/getting-started-app$ docker run -d ubuntu bash -c "shuf -i 1-10000 -n 1 -o /random.txt && tail -f /dev/null"
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
57c139bbda7e: Pull complete
Digest: sha256:e9569c25505f33ff72e88b2990887c9dcf230f23259da296eb814fc2b41af999
Status: Downloaded newer image for ubuntu:latest
799b622d7165d16ce4a78952406dfe41317327b354a38858a5d5db1b97176ccf
```

- To test if the bash command worked, I used the command ‘ docker exec kind_northcutt cat /random.txt’ and got the random number ‘785.’

```
conor@Dev-System:~/getting-started-app$ docker exec kind_northcutt cat /random.txt
785
conor@Dev-System:~/getting-started-app$
```

- I then started another ubuntu container (with the same image), to see if the ‘random.txt’ file exists in the new container.

```
conor@Dev-System:~/getting-started-app$ docker run -it ubuntu ls /
bin    dev    home  lib32  libx32  mnt    proc  run    srv    tmp    var
boot   etc    lib   lib64  media   opt    root  sbin   sys    usr
```

- This shows that changes made to a container are isolated from that container and when a container is removed those changes are lost.
- Container ‘Volumes’ can provide the ability to connect specific filesystem paths of the container back to the host machine.
- This offers the ability to mount the same directory across container restarts and see the same files.

Persist the Todo data:

- By default, the Todo app stores its data in an SQLite database in the container's filesystem.
- By creating a volume and attaching it to the directory where I stored the data, I can persist the data for the To-do app.

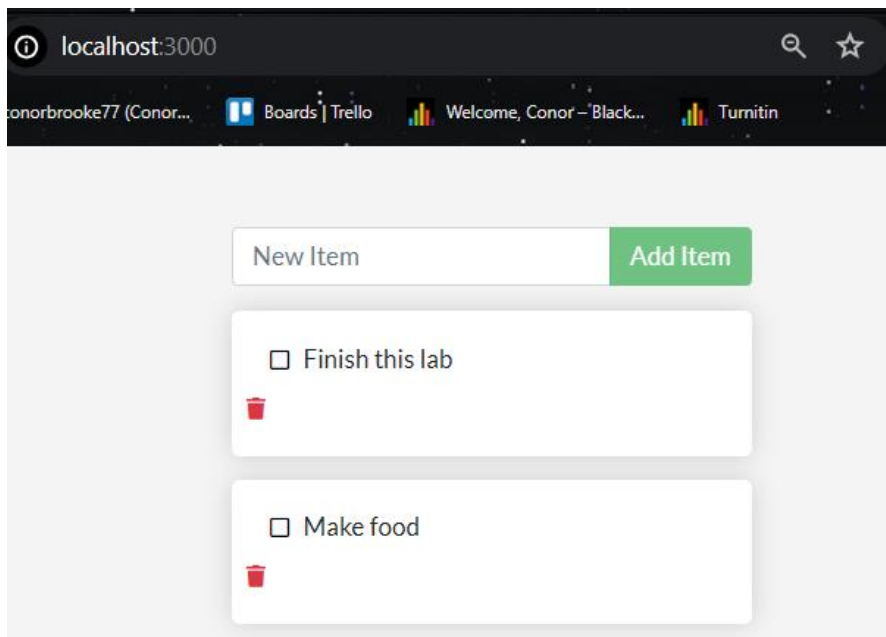
Creating a volume to persist the data and removing current container:

```
conor@Dev-System:~/getting-started-app$ docker volume create todo-db
todo-db
conor@Dev-System:~/getting-started-app$ ls
Dockerfile  README.md  package.json  spec  src  text  yarn.lock
conor@Dev-System:~/getting-started-app$ docker stop gracious_austin
gracious_austin
conor@Dev-System:~/getting-started-app$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
conor@Dev-System:~/getting-started-app$ docker rm -f gracious_austin
gracious_austin
```

- Starting a new container but adding the ‘—mount’ option to specify a volume mount.

```
conor@Dev-System:~/getting-started-app$ docker run -dp 127.0.0.1:3000:3000 --mount type=volume,src=todo-db,target=/etc/todos getting-started
5169f7b661bfec4a1988a97da6eeee24080027cd8071a1ee2185994b77332032
conor@Dev-System:~/getting-started-app$
```

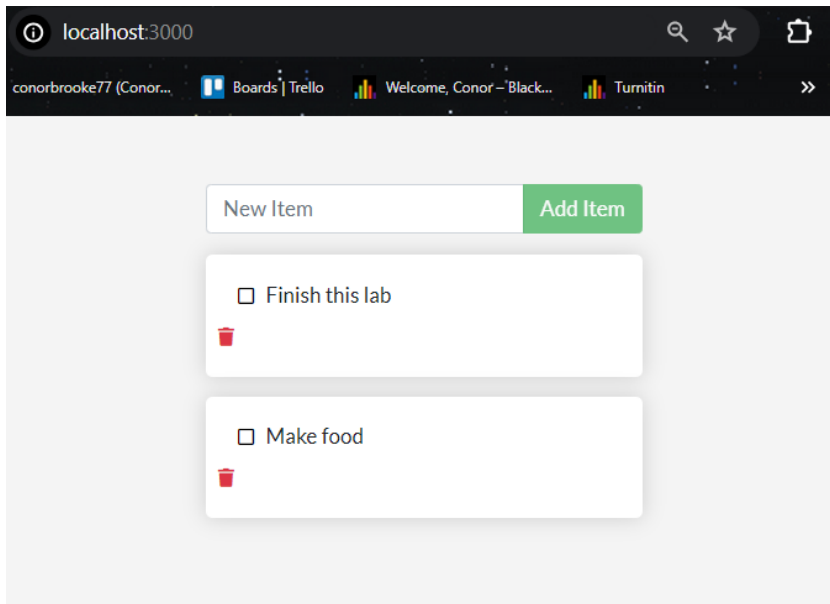
- Adding a few sample tasks to test the mounted volume.



- Removed the container and started it again to see if the tasks were saved.

```
conor@Dev-System:~/getting-started-app$ docker stop blissful_saha
blissful_saha
conor@Dev-System:~/getting-started-app$ docker rm -f blissful_saha
blissful_saha
conor@Dev-System:~/getting-started-app$ docker run -dp 127.0.0.1:3000:3000 --mount type=volume,src=todo-db,target=/etc/todos getting-started
8fcd0cf763e83bbc4282aa3e7b0a9fee9d24bcd4f425fce732d44386aeadbf3ae9
conor@Dev-System:~/getting-started-app$
```

- The volume saved the tasks:



Where docker is storing this volume:

```
conor@Dev-System:~/getting-started-app$ docker volume inspect todo-db
[
  {
    "CreatedAt": "2024-02-04T18:45:01Z",
    "Driver": "local",
    "Labels": null,
    "Mountpoint": "/var/lib/docker/volumes/todo-db/_data",
    "Name": "todo-db",
    "Options": null,
    "Scope": "local"
  }
]
```

Summary

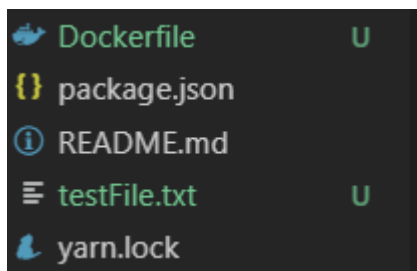
I learned how to use Docker volumes to persist data across container restarts, ensuring that data created in one container can be accessed in another, despite the isolation of container file systems. This is important for applications that need to save states or data beyond the lifecycle of a single container, such as databases or file uploads. The guide showed me how to create and attach volumes to containers, making my applications more robust and data resilient.

Using Bind Mounts:

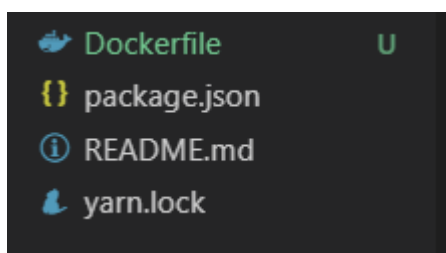
- Another type of mount is a bind mount, this mount allows me to share a directory from my filesystem into the container.
- This can be used to mount source code into the container and see changes I make to my code immediately inside the container.

Created new container using the bind mount and added a new text file named 'testFile.txt':

```
conor@Dev-System:~/getting-started-app$ docker run -it --mount type=bind,src="$(pwd)",target=/src ubuntu bash
root@4a55346bfa15:/# ls
bin    dev    home   lib32  libx32 mnt    proc   run    src    sys    usr
boot   etc    lib    lib64  media  opt    root   sbin   srv    tmp    var
root@4a55346bfa15:/# cd src
root@4a55346bfa15:/src# ls
Dockerfile  README.md  package.json  spec  src  text  yarn.lock
root@4a55346bfa15:/src# touch testFile.txt
root@4a55346bfa15:/src#
```



- I then deleted the 'testFile.txt' from my files in VSCode outside the container.
- In the container, I list the contents of the app directory and observe that the file is now gone.



```
root@4a55346bfa15:/src# ls
Dockerfile  README.md  package.json  spec  src  yarn.lock
root@4a55346bfa15:/src#
```

Running the app in a development container:

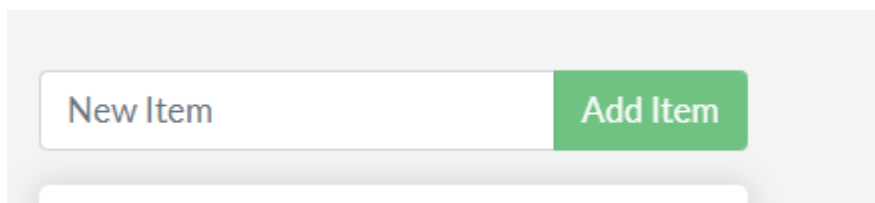
- The advantage of using bind mounts for local development setups is that the machine doesn't need to have all the build tools and environments installed. Using a 'docker run' command, Docker pulls all dependencies and tools needed.

- Creating the development container with 'yarn' installed.

```
conor@Dev-System:~/getting-started-app$ 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"
Unable to find image 'node:18-alpine' locally
18-alpine: Pulling from library/node
4abcf2066143: Already exists
eb6c7c29ba4d: Already exists
3d4a65156edf: Already exists
5bdb6c27eb32: Already exists
Digest: sha256:0085670310d2879621f96a4216c893f92e2ded827e9e6ef8437672e1bd72f437
Status: Downloaded newer image for node:18-alpine
837c97bbe0791756a5959f9ce1f40fc6e45baf4459a94b785ee7a1a8918b39dd
```

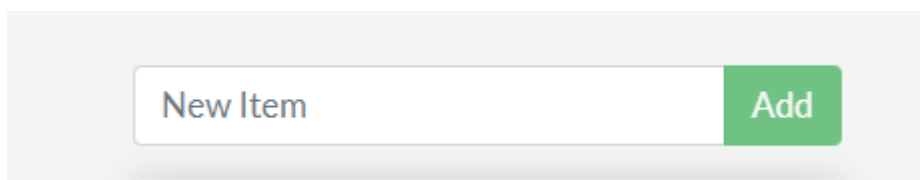
Updating app with Development Container:

- Changing 'Add Item' to 'Add'



```
disabled={!newItem.length}
className={submitting ? 'disabled' : ''}
>
{submitting ? 'Adding...' : 'Add'}
</Button>
</InputGroup.Append>
InputGroup>
```

- After refreshing the web browser, I see the change reflected almost immediately because of the bind mount.



Summary

I learned the application of sharing a directory from the host's filesystem into a container. This allows for real-time development changes, as modifications made to the source code on my machine are immediately visible within the container. I set up a development environment using bind mounts, enabling live updates without needing to rebuild the container for every change.

Multi container apps:

- Utilizing multi-container apps allows for independent scaling and versioning of APIs/front ends and databases, enhancing flexibility and maintainability.
- Separate containers support the use of managed services in production and simplify container management by avoiding the complexity of running multiple processes in a single container.

Container Networking:

- Since containers run in isolation by default, both containers must be on the same network to be able to communicate with each other.
- Below I created a network in Docker:

```
conor@Dev-System:~/getting-started-app$ docker network create todo-app  
ec41a2544478189c50f328513a2919d6858d7c4cb1ebb26bc0492462c498f50c
```

- Starting MySQL container and attaching it to the network:

```
conor@Dev-System:~/getting-started-app$ docker run -d \  
--network todo-app --network-alias mysql \  
-v todo-mysql-data:/var/lib/mysql \  
-e MYSQL_ROOT_PASSWORD=secret \  
-e MYSQL_DATABASE=todos \  
mysql:8.0
```

- Connecting to the 'MySQL' database.

```
conor@Dev-System:~/getting-started-app$ docker exec -it heuristic_blackburn 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
```

- Checking out the current databases:

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

- Using 'dig' command to find the IP address for the hostname 'MySQL'.

```

88d888b. .d8888b. 88      88      88
88'  `88 880000d8 88  Y800000. 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

f6a3a6697188 ~ dig mysql

; <<>> DiG 9.18.13 <<>> mysql
;; global options: +cmd
;; Got answer:
;; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 53681
;; 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

```

- The 'A' record for 'MySQL' resolves to the IP address '172.18.0.2'
- My app now just simply needs to connect to 'MySQL', and it'll talk to the database.

Starting a dev-ready container:


```


conor@Dev-System:~/getting-started-app$ docker run -dp 127.0.0.1:3000:3000 \
-w /app -v "$(pwd):/app" \
--network todo-app \
-e MYSQL_HOST=mysql \
-e MYSQL_USER=root \
-e MYSQL_PASSWORD=secret \
-e MYSQL_DB=todos \
node:18-alpine \
sh -c "yarn install && yarn run dev"
bea19ddf8a1dca4f3cb3e9e9bb38373ea3133c1aa2daec2b914078ddde772e70
conor@Dev-System:~/getting-started-app$ docker ps


```


CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PO
RTS	NAMES				
bea19ddf8a1d	node:18-alpine	"docker-entrypoint.s..."	4 seconds ago	Up 2 seconds	12
7.0.0.1:3000->3000/tcp	lucid_pare				
92b5509ac0a3	mysql:8.0	"docker-entrypoint.s..."	15 minutes ago	Up 15 minutes	33
06/tcp, 33060/tcp	heuristic_blackburn				

- Now each item I add to my to-do list will be written to the database.

☐ Get Food


☐ Finish Lab


☐ Code Something


☐ Go GYM


```
mysql> select * from todo_items;
```

id	name	completed
413b175a-1194-4bae-a366-74e372454416	Get Food	0
580ad049-c539-43fe-b137-2d7580bd7162	Finish Lab	0
5a9bc07d-1892-4b4f-a9a2-af941490f282	Code Something	0
8b321e08-b2ce-4fa7-a424-e89640ab29f7	Go GYM	0

4 rows in set (0.00 sec)

Summary

I learned how to set up a multi-container application that includes a MySQL database. This involved creating a Docker network to allow containers to communicate with each other and using environment variables to configure the MySQL container. I also learned how to use DNS for service discovery within the Docker network, enabling my application container to connect to the MySQL database using its network alias.

Use Docker Compose:

- Creating the compose file to define my application stack in a file.

```
conor@Dev-System:~/getting-started-app$ touch compose.yaml
conor@Dev-System:~/getting-started-app$ ls
Dockerfile  README.md  compose.yaml  node_modules  package.json  spec  src  yarn.lock
```

Defining the app service:

```
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
```

Defining the 'MySQL' service:

```
mysql:
  image: mysql:8.0
  volumes:
    - todo-mysql-data:/var/lib/mysql
```

Defining the Volume and added the environment variables:

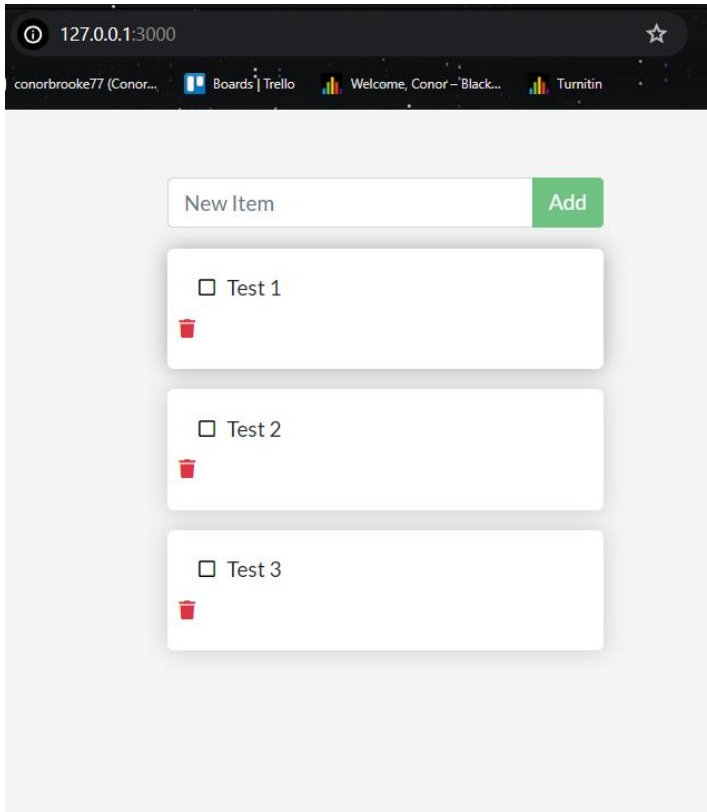
```
services:
  app:
    image: node:18-alpine
    command: sh -c "yarn install && yarn run dev"
    ports:
      - 127.0.0.1:3000:3000
    working_dir: /app
    volumes:
      - .:/app
    environment:
      MYSQL_HOST: mysql
      MYSQL_USER: root
      MYSQL_PASSWORD: secret
      MYSQL_DB: todos

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

volumes:
  todo-mysql-data:
```

Run the application stack:

```
conor@Dev-System:~/getting-started-app$ docker compose up -d
[+] Running 4/4
✓Network getting-started-app_default      Created           0.0s
✓Volume "getting-started-app_todo-mysql-data" Created           0.0s
✓Container getting-started-app-mysql-1    Started           0.1s
✓Container getting-started-app-app-1      Started           0.1s
```



Tearing the containers down:

```
conor@Dev-System:~/getting-started-app$ docker compose down
[+] Running 3/3
✓Container getting-started-app-app-1      Rem...
✓Container getting-started-app-mysql-1    R...
✓Network getting-started-app_default      Rem...
conor@Dev-System:~/getting-started-app$
```

Summary

This section details the process of defining and starting all services from a configuration file with docker-compose up, managing the services as a single unit while keeping their data in volumes. This approach simplifies the process of running complex applications with multiple services, demonstrating the use of Compose to build, deploy, and manage an application with ease.