Bharatiya Vidya Bhavan's

SARDAR PATEL INSTITUTE OF TECHNOLOGY

An Autonomous Institute Affiliated To University Of Mumbai

Munshi Nagar,Andheri (W) Mumbai 400 058

C.I.T.L. EXPERIMENT 9

Topic : Inventory Management System

Submitted By:

Akash Panicker	2021300089
Mahesh Patil	2021300095
Rohit Phalke	2021300100
Adwait Purao	2021300101

Submitted To:

Prof. Sunil Ghane

Aim:
Create and use a Docker container interactively.Create a Dockerfile, which allows you to declaratively define your containers. Run detached containers and understand port forwarding

Problem Statement:
Develop an inventory management system for a retail store that efficiently tracks and manages the inventory of products. The system should provide real-time updates on stock levels, generate alerts for low stock items, enable easy addition and removal of products, and offer insights into sales trends to optimize restocking decisions.

Theory:

- **Containerization**
  Containerization is a technology that allows you to package software into standardized units that can be easily deployed and managed. Containers are similar to virtual machines, but they are more lightweight and efficient. This makes them ideal for deploying applications in a cloud environment.

- **Docker**
  Docker is a leading platform for containerization. It provides a suite of tools that make it easy to build, run, and manage containers. Docker is a popular choice for deploying applications on a variety of platforms, including Linux, Windows, and macOS.

- **Benefits of Containerization**
  There are many benefits to using containerization, including:
    - Portability: Containers can be easily ported from one environment to another.
    - Isolation: Containers are isolated from each other, which means that they cannot interfere with each other.
    - Resource efficiency: Containers are more resource efficient than virtual machines.
    - Scalability: Containers can be easily scaled up or down to meet demand.

- **Creating and Using a Docker Container Interactively**
  Docker containers provide a lightweight and portable way to package and run applications. They are self-contained environments that include everything an application needs to run, including its code, libraries, and dependencies. This makes them ideal for deploying applications in a consistent and reliable way across different environments.

- **Creating a Dockerfile**
  A Dockerfile is a text file that contains instructions for building a Docker image. The instructions are written in a simple, declarative format that makes it easy to understand and maintain.

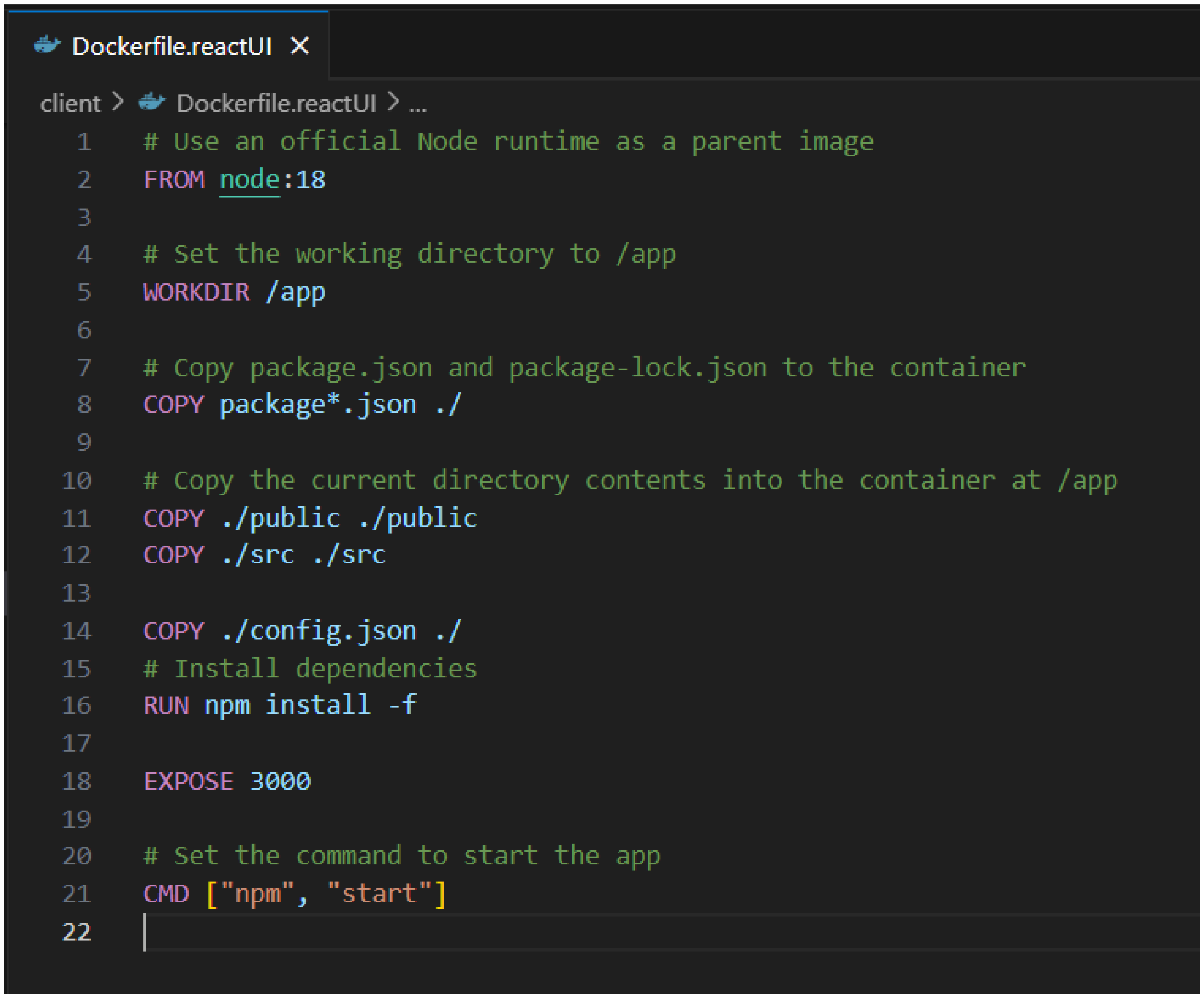
- **Building an Image from a Dockerfile**
  To build an image from a Dockerfile, you can use the following command:
  docker build -t my-image .
  This command will run a container from the my-image image in interactive mode. This means that you will be able to type commands into the container and see the results.

- **Detached Containers**
  A detached container is a container that is not running in interactive mode. This means that you will not be able to type commands into the container after it is started. To run a container in detached mode, you can use the -d flag:
  docker run -d my-image

- **Port Forwarding**
  Port forwarding is a way to map a port on the host machine to a port on a container. This allows you to access the container's application from the host machine. To forward a port, you can use the -p flag:
  docker run -d -p 8000:8000 my-image
  This command will run a container from the my-image image in detached mode and map port 8000 on the host machine to port 8000 on the container. This means that you can access the container's application at http://localhost:8000.

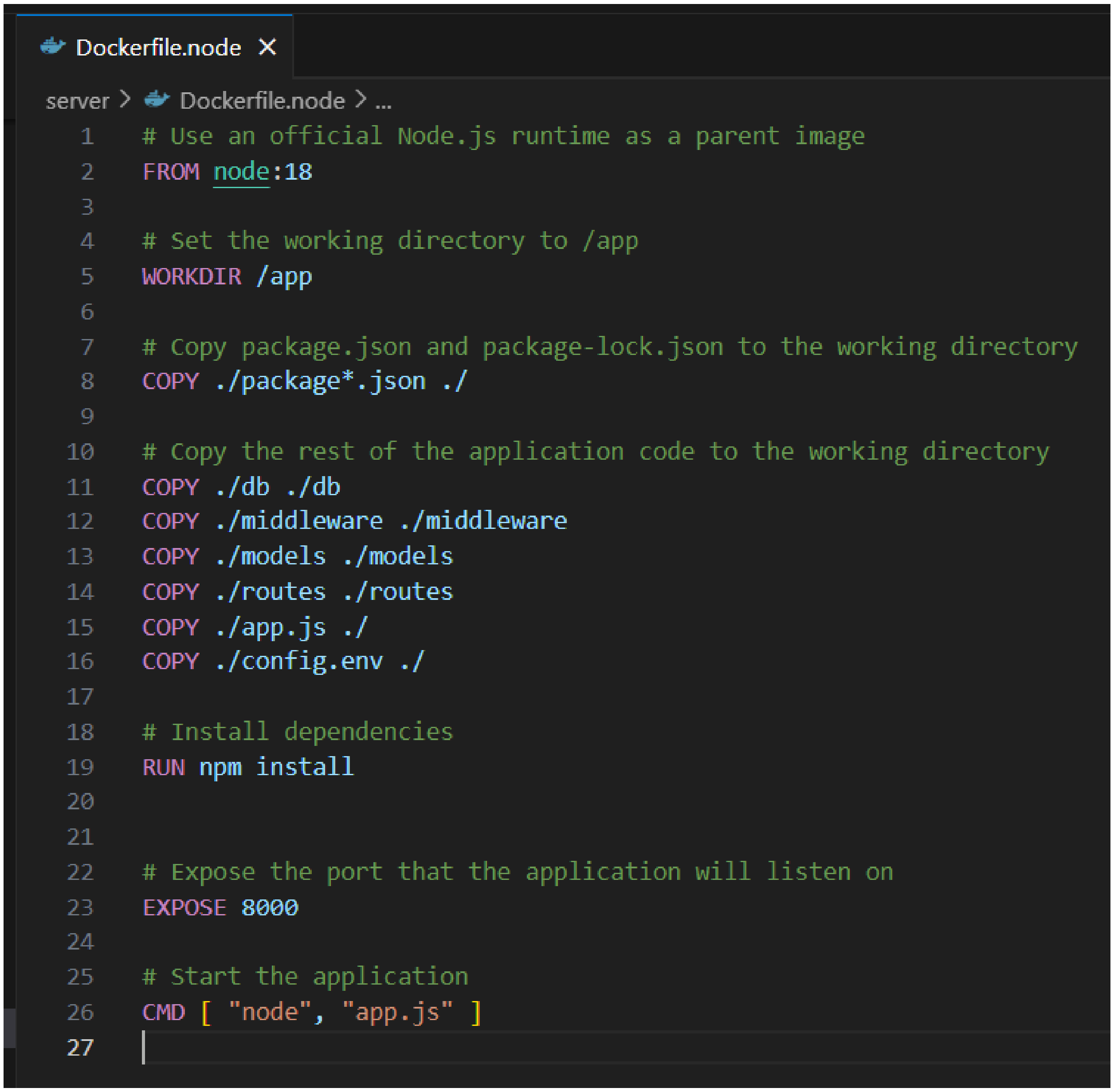
Screenshots:

Docker file for ReactJS frontend



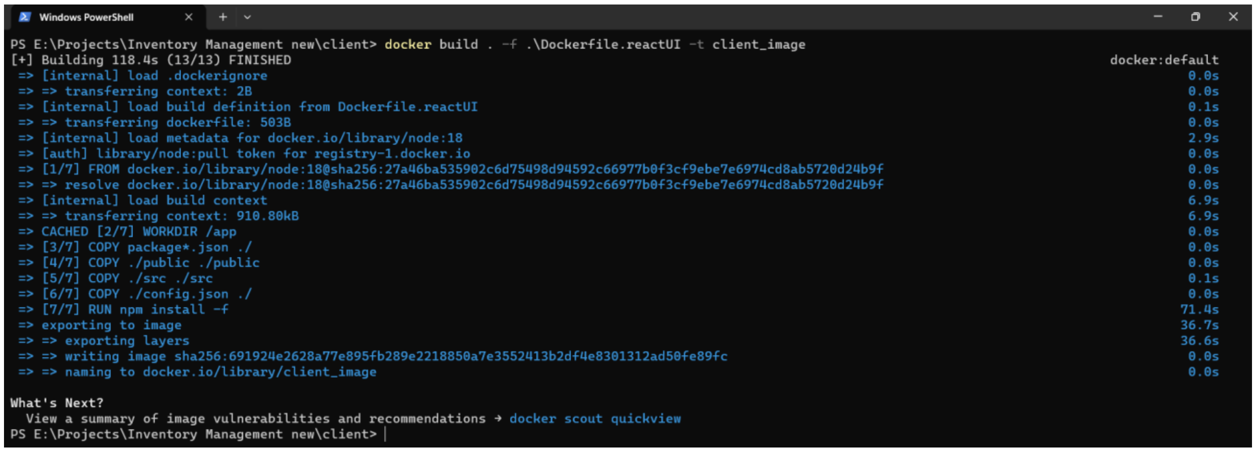
```dockerfile
# Use an official Node runtime as a parent image
FROM node:18

# Set the working directory to /app
WORKDIR /app

# Copy package.json and package-lock.json to the container
COPY package*.json ./

# Copy the current directory contents into the container at /app
COPY ./public ./public
COPY ./src ./src

COPY ./config.json ./
# Install dependencies
RUN npm install -f

EXPOSE 3000

# Set the command to start the app
CMD ["npm", "start"]
```

Docker file for NodeJS backend



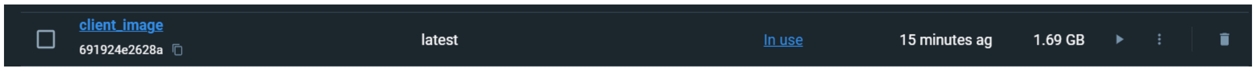
```dockerfile
# Use an official Node.js runtime as a parent image
FROM node:18

# Set the working directory to /app
WORKDIR /app

# Copy package.json and package-lock.json to the working directory
COPY ./package*.json ./

# Copy the rest of the application code to the working directory
COPY ./db ./db
COPY ./middleware ./middleware
COPY ./models ./models
COPY ./routes ./routes
COPY ./app.js ./
COPY ./config.env ./

# Install dependencies
RUN npm install


# Expose the port that the application will listen on
EXPOSE 8000

# Start the application
CMD [ "node", "app.js" ]

```

Building image of frontend



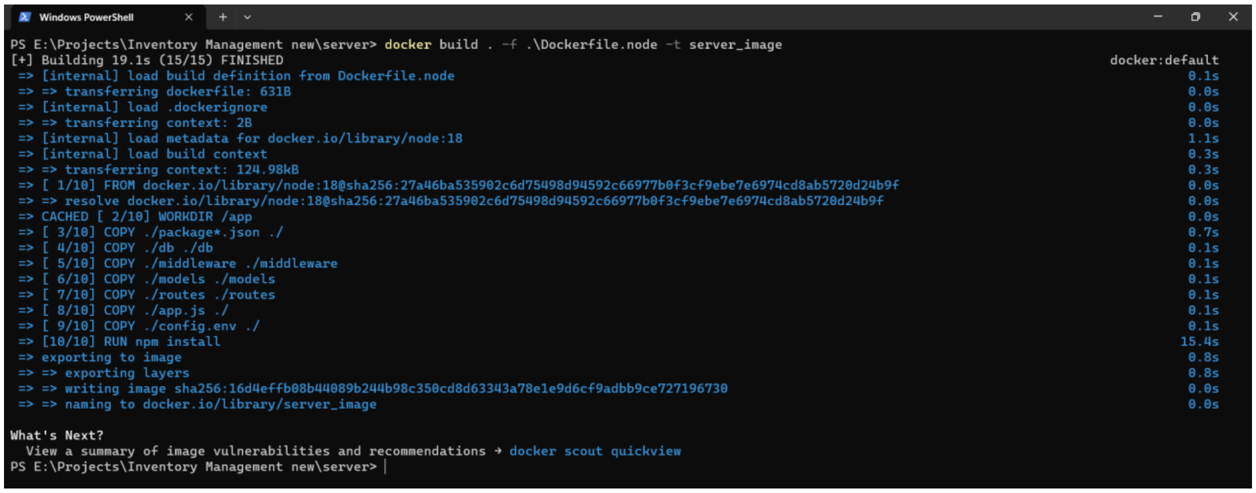
```
PS E:\Projects\Inventory Management new\client> docker build . -f .\Dockerfile.reactUI -t client_image
[+] Building 118.4s (13/13) FINISHED                                                                                  docker:default
 => [internal] load .dockerignore                                                                                               0.0s
 => => transferring context: 2B                                                                                                 0.0s
 => [internal] load build definition from Dockerfile.reactUI                                                                    0.1s
 => => transferring dockerfile: 503B                                                                                            0.0s
 => [internal] load metadata for docker.io/library/node:18                                                                      2.9s
 => [auth] library/node:pull token for registry-1.docker.io                                                                     0.0s
 => [1/7] FROM docker.io/library/node:18@sha256:27a46ba535902c6d75498d94592c66977b0f3cf9ebe7e6974cd8ab5720d24b9f                0.0s
 => => resolve docker.io/library/node:18@sha256:27a46ba535902c6d75498d94592c66977b0f3cf9ebe7e6974cd8ab5720d24b9f                0.0s
 => [internal] load build context                                                                                               6.9s
 => => transferring context: 910.80kB                                                                                           6.9s
 => CACHED [2/7] WORKDIR /app                                                                                                   0.0s
 => [3/7] COPY package*.json ./                                                                                                 0.0s
 => [4/7] COPY ./public ./public                                                                                                0.0s
 => [5/7] COPY ./src ./src                                                                                                      0.1s
 => [6/7] COPY ./config.json ./                                                                                                 0.0s
 => [7/7] RUN npm install -f                                                                                                   71.4s
 => exporting to image                                                                                                         36.7s
 => => exporting layers                                                                                                        36.6s
 => => writing image sha256:691924e2628a77e895fb289e2218850a7e3552413b2df4e8301312ad50fe89fc                                    0.0s
 => => naming to docker.io/library/client_image                                                                                 0.0s

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
PS E:\Projects\Inventory Management new\client> |
```

Frontend image created



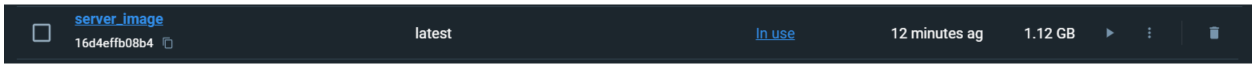
client_image 691924e2628a	latest	In use	15 minutes ag	1.69 GB	

Building image of backend



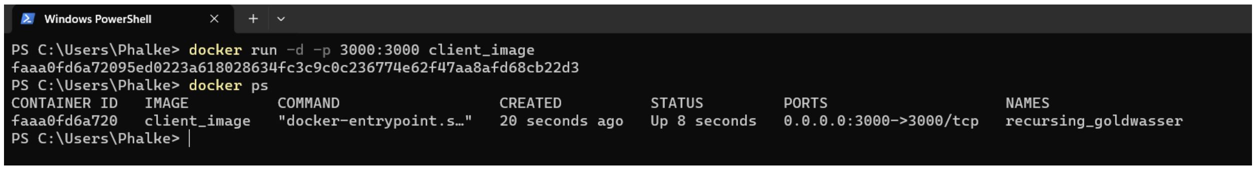
```
PS E:\Projects\Inventory Management new\server> docker build . -f .\Dockerfile.node -t server_image
[+] Building 19.1s (15/15) FINISHED                                                                                   docker:default
 => [internal] load build definition from Dockerfile.node                                                                       0.1s
 => => transferring dockerfile: 631B                                                                                            0.0s
 => [internal] load .dockerignore                                                                                               0.0s
 => => transferring context: 2B                                                                                                 0.0s
 => [internal] load metadata for docker.io/library/node:18                                                                      1.1s
 => [internal] load build context                                                                                               0.3s
 => => transferring context: 124.98kB                                                                                           0.3s
 => [ 1/10] FROM docker.io/library/node:18@sha256:27a46ba535902c6d75498d94592c66977b0f3cf9ebe7e6974cd8ab5720d24b9f              0.0s
 => => resolve docker.io/library/node:18@sha256:27a46ba535902c6d75498d94592c66977b0f3cf9ebe7e6974cd8ab5720d24b9f                0.0s
 => CACHED [ 2/10] WORKDIR /app                                                                                                 0.0s
 => [ 3/10] COPY ./package*.json ./                                                                                             0.7s
 => [ 4/10] COPY ./db ./db                                                                                                      0.1s
 => [ 5/10] COPY ./middleware ./middleware                                                                                      0.1s
 => [ 6/10] COPY ./models ./models                                                                                              0.1s
 => [ 7/10] COPY ./routes ./routes                                                                                              0.1s
 => [ 8/10] COPY ./app.js ./                                                                                                    0.1s
 => [ 9/10] COPY ./config.env ./                                                                                                0.1s
 => [10/10] RUN npm install                                                                                                    15.4s
 => exporting to image                                                                                                          0.8s
 => => exporting layers                                                                                                         0.8s
 => => writing image sha256:16d4effb08b44089b244b98c350cd8d63343a78e1e9d6cf9adbb9ce727196730                                    0.0s
 => => naming to docker.io/library/server_image                                                                                 0.0s

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
PS E:\Projects\Inventory Management new\server> |
```

Backend image created



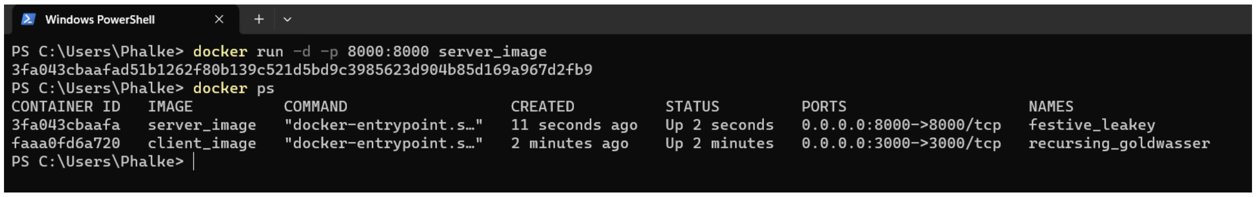
server_image 16d4effb08b4	latest	In use	12 minutes ag	1.12 GB	

Creating container and Running frontend image on port 3000 in detached mode



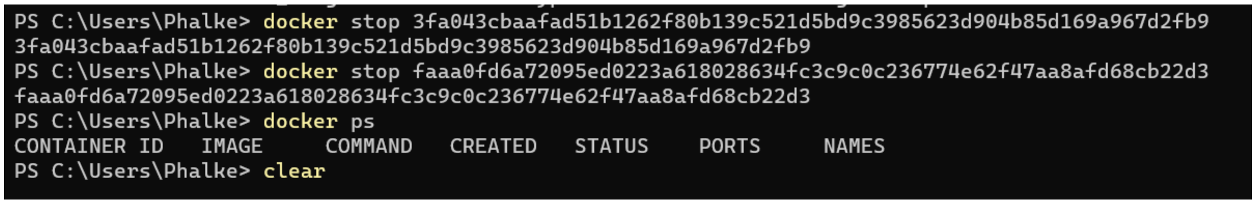
```
PS C:\Users\Phalke> docker run -d -p 3000:3000 client_image
faaa0fd6a72095ed0223a618028634fc3c9c0c236774e62f47aa8afd68cb22d3
PS C:\Users\Phalke> docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS         PORTS                    NAMES
faaa0fd6a720   client_image   "docker-entrypoint.s…"   20 seconds ago   Up 8 seconds   0.0.0.0:3000->3000/tcp   recursing_goldwasser
PS C:\Users\Phalke> |
```

Creating container and Running backend image on port 8000 in detached mode



```
PS C:\Users\Phalke> docker run -d -p 8000:8000 server_image
3fa043cbaafad51b1262f80b139c521d5bd9c3985623d904b85d169a967d2fb9
PS C:\Users\Phalke> docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS         PORTS                    NAMES
3fa043cbaafa   server_image   "docker-entrypoint.s…"   11 seconds ago   Up 2 seconds   0.0.0.0:8000->8000/tcp   festive_leakey
faaa0fd6a720   client_image   "docker-entrypoint.s…"   2 minutes ago    Up 2 minutes   0.0.0.0:3000->3000/tcp   recursing_goldwasser
PS C:\Users\Phalke> |
```

Images stopped



```
PS C:\Users\Phalke> docker stop 3fa043cbaafad51b1262f80b139c521d5bd9c3985623d904b85d169a967d2fb9
3fa043cbaafad51b1262f80b139c521d5bd9c3985623d904b85d169a967d2fb9
PS C:\Users\Phalke> docker stop faaa0fd6a72095ed0223a618028634fc3c9c0c236774e62f47aa8afd68cb22d3
faaa0fd6a72095ed0223a618028634fc3c9c0c236774e62f47aa8afd68cb22d3
PS C:\Users\Phalke> docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
PS C:\Users\Phalke> clear
```

Conclusion:

Docker containers have become a popular choice for developing, deploying, and managing applications due to their lightweight, portable, and efficient nature. They provide a self-contained environment that encapsulates an application and its dependencies, ensuring consistency and predictability across different deployment environments. Docker's portability and isolation capabilities make it well-suited for cloud environments, enabling rapid application delivery and flexible resource management. In summary, Docker containers have transformed the way we build and run applications, offering a powerful and versatile solution for modern application development and deployment.