we're discussing the difference between virtual machines and containers. But before diving into that, let's rewind and talk about the pre-virtual machine and container era.

Traditionally, businesses operated with applications running on servers, where each server hosted just one application - be it a website, database, or email service.

This was due to the limitation of operating systems like Linux and Windows, which couldn't securely run multiple applications on a single server. Consequently, companies had to buy separate servers for each new application, resulting in underutilization of server capacity and wastage of resources.

To address this issue, engineers developed virtual machines (VMs), which allowed multiple applications to run on a single server by simulating hardware and software. For example, if a company had three servers each running one application, these applications could be consolidated onto a single server by creating three virtual machines. Each virtual machine ran its own operating system and applications, enabling multiple applications to coexist on one physical server.

Virtual machines solved the problem of resource wastage but came with drawbacks such as high disk space consumption, RAM, and CPU usage, slow startup times, and additional licensing costs for operating systems.

Enter containers - lightweight, portable, and efficient alternatives to virtual machines. Unlike VMs, containers encapsulate only the application and its dependencies, allowing them to run on any computing environment without additional configuration. Docker, a leading containerization software, packages applications with all necessary files and configurations into a single container image, making them easy to distribute and deploy.

Containers share the underlying operating system kernel, resulting in smaller file sizes, faster startup times, and lower resource consumption compared to VMs. However, containers must match the host operating system, and all containers on a server will go down if the host operating system crashes.

In conclusion, while both virtual machines and containers offer benefits, containers, especially Docker containers, are favored for their size, portability, and speed. With their ability to efficiently package and deploy applications, containers are shaping the future of software development and deployment.

Hey, are you tired of remembering passwords and managing login credentials? Introducing RoboForm, the number one password manager that securely stores and autofills your passwords across all your devices. With RoboForm, you only need to remember one master password, and it takes care of the rest. Say goodbye to password hassles and try RoboForm today !  
  
  
  
**Dockerfile**

FROM node:module

# create react folder and cd into it within docker container

WORKDIR /app

# copy from currentDir to docker workDir

COPY . .

# installing dependencies

RUN npm install

# build the code

RUN npm run build

# expose the app port when running the container

CMD ["npm","run","start"]

# dockerfile ends

docker run -p 5000:4173 react:v0

# when you run this command, Docker will create a new container based on the react:v0 image and map port 5000 on the host to port 4173 inside the container, allowing you to access the application running inside the container via port 5000 on your host machine.

# in case if you want to name it

docker run -p 5000:4173 --name react-client react:v0

**If you want to make it more efficient**

FROM node:14-alpine

WORKDIR /app

COPY package\*.json .

RUN npm install

COPY . .

RUN npm run build

CMD [“npm” , “run” , “preview”]

Cmd [ “echo”, “Hey user”]   
  
#prints

Entrypoint -> Appends arguments

Writing docker file smartly to make if efficient

# keep the node module in the docker ignore file

FROM node:latest