**Dockerizing a Node.js App with Multistage Build**

**In this blog, I’ll walk you through how I cloned a Node.js application, dockerized it using best practices, and implemented a multistage build to optimize image size and performance. You'll also find the exact commands used and some key DevOps learnings from the process**

#### **Setting up a cloud server is a foundational step for deploying and running backend applications like Node.js in a scalable and publicly accessible environment.**

**🔹 1. Provisioning a Virtual Server To host a Node.js application on the cloud, you first need to provision a virtual server using a cloud provider such as AWS, Google Cloud, or Azure. In AWS, this is done via EC2 (Elastic Compute Cloud).**

**You select a base operating system image (e.g., Ubuntu).**

**Define instance specifications (CPU, RAM, disk).**

**Configure a key pair for secure SSH access.**

**🔹 2. Configuring Network Access (Security Groups) Once the instance is launched, network access rules must be configured using what AWS calls Security Groups.**

**Security groups define inbound and outbound rules:**

**SSH (Port 22) is opened for secure terminal access.**

**HTTP (Port 80) is opened to allow web traffic.**

## **what is node .js**

**"Node.js is an open-source, cross-platform JavaScript runtime environment that allows developers to execute JavaScript code outside of a web browser."**

## **Document the process in a step-by-step blog**

For Ubuntu/Linux

```bash

sudo apt update

sudo apt install -y nodejs npm

``

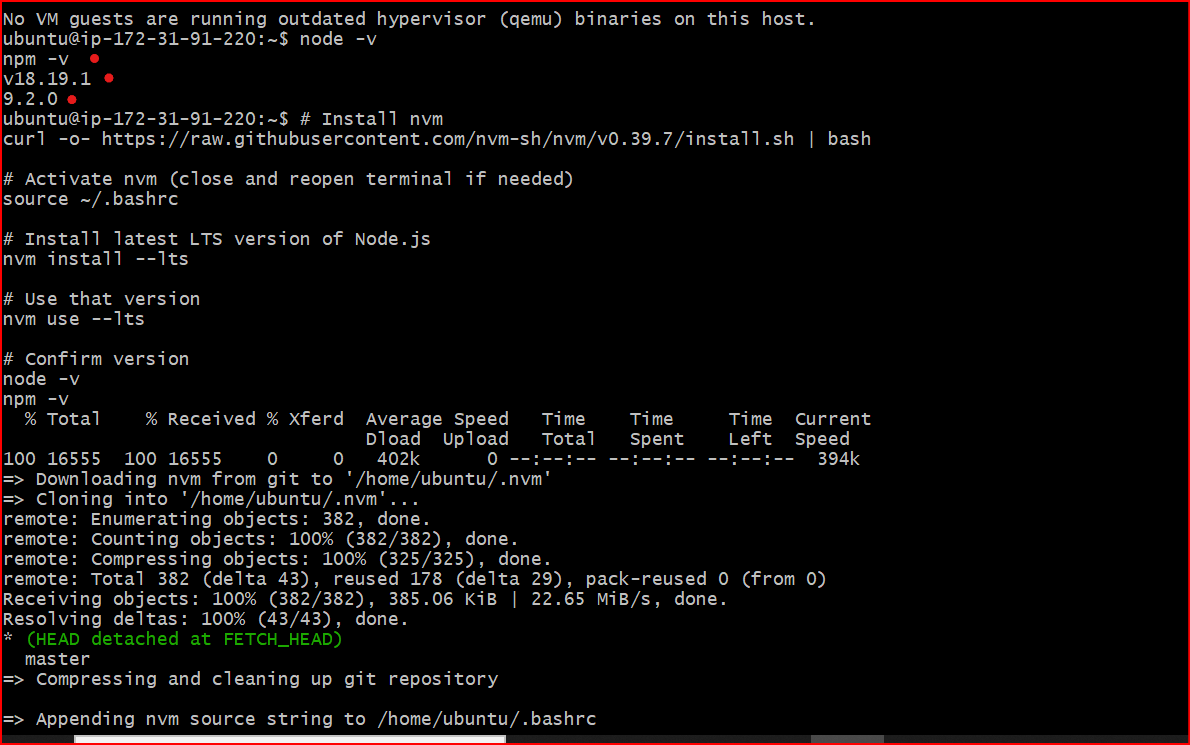
### **to To verify npm and node version:**

```bash

node -v

npm -v

```



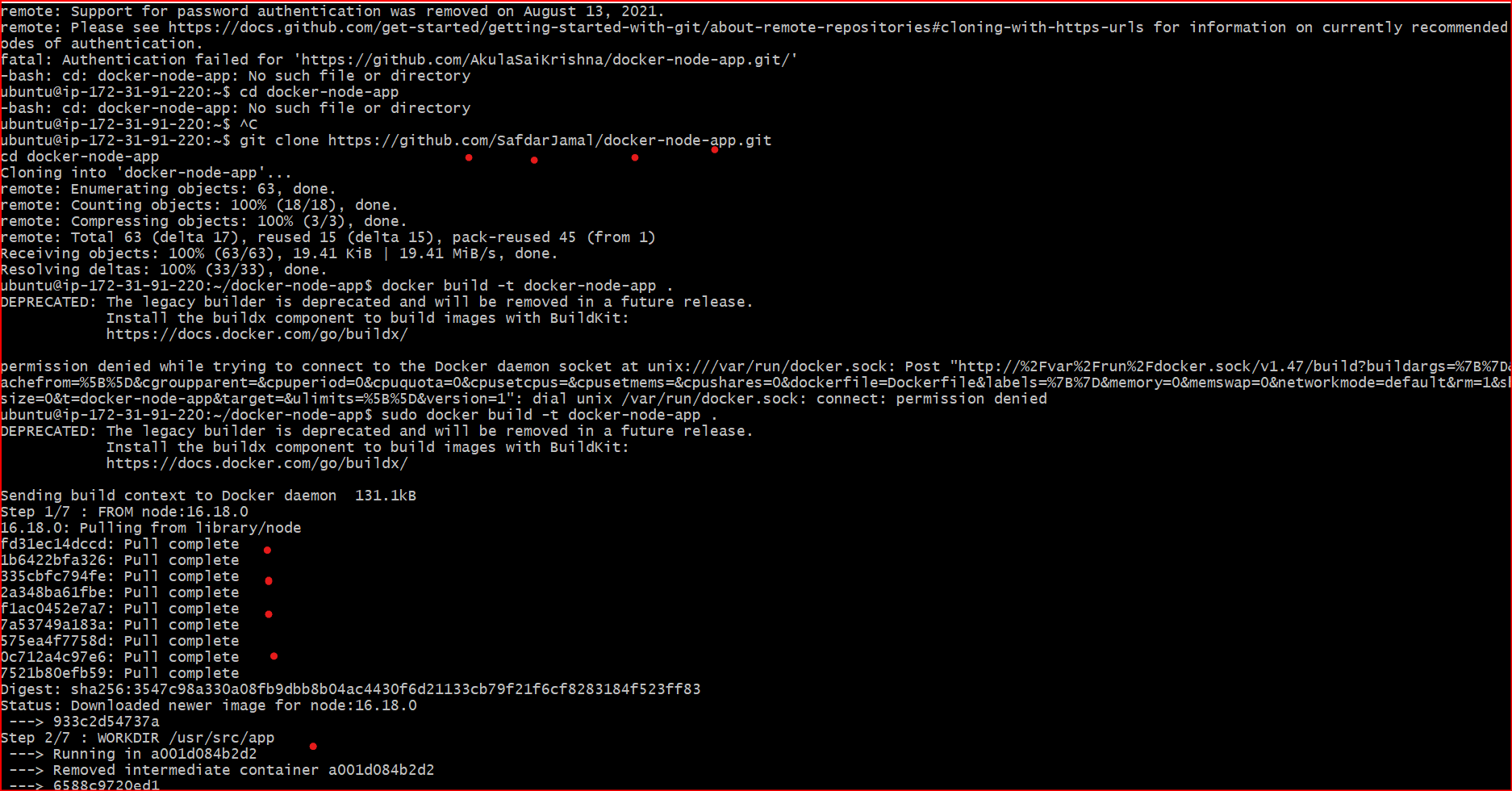
## **cloning a GitHub repository**

```bash

git clone https://github.com/SafdarJamal/docker-node-app.git

cd docker-node-app

```



## **what is Docker**

Docker is a software platform that allows you to build, test, and deploy applications quickly. Docker packages software into standardized units called containers that have everything the software needs to run including libraries, system tools, code, and runtime.

## **what is Image in Docker**

**Image is a read-only template that contains the application code, runtime, libraries, and dependencies needed to run a container. Think of it as a blueprint or snapshot of a container.**

## **now Here’s a complete guide to install Docker on Ubuntu**

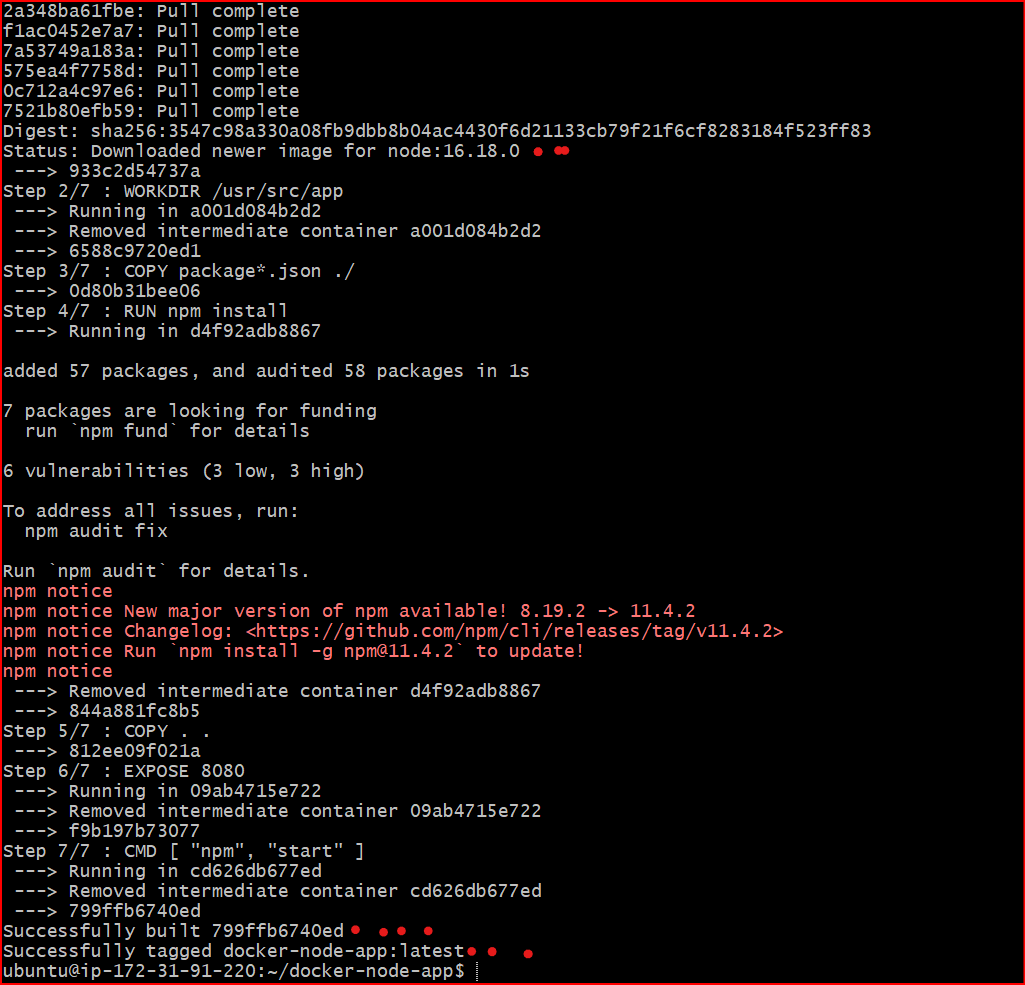
sudo apt update

sudo apt -y install docker

sudo apt install docker.io

sudo systemctl start docker

sudo systemctl enable docker



# What is a Dockerfile?

**A Dockerfile is a text file that contains a set of instructions used to build a Docker image.**

**Think of it like a recipe that tells Docker:**

* **What base system to use (e.g., Ubuntu, Node.js)**
* **What dependencies to install**
* **What files to copy**
* **What command to run to start your application**

FROM node:16.18.0

# Create app directory

WORKDIR /usr/src/app

# A wildcard is used to ensure both package.json AND package-lock.json are copied

COPY package\*.json ./

# Install app dependencies

RUN npm install

# Bundle app source

COPY . .

EXPOSE 8080

CMD [ "npm", "start" ]

## **Now that you have some source code and a Dockerfile, it’s time to build your first image:**

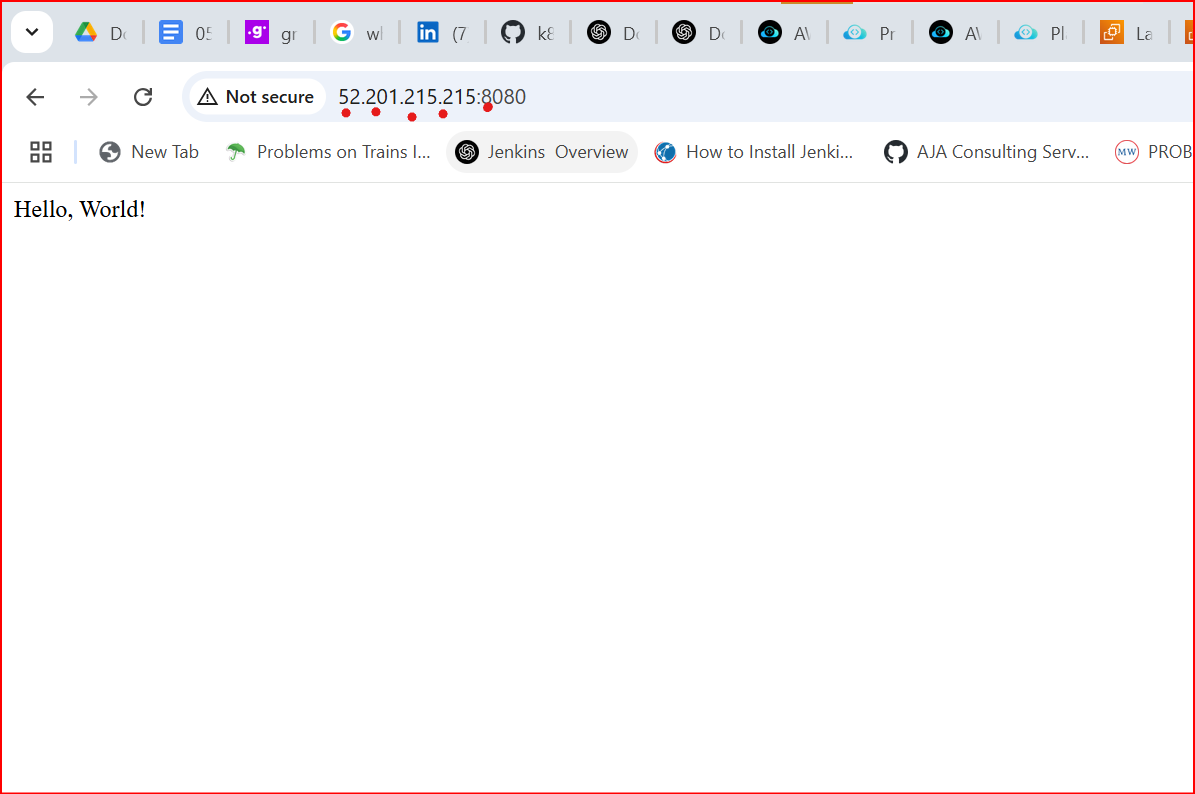
docker build -t docker-node-app .

## **Start a container based on your new image:**

sudo docker run -d -p 8080:8080 docker-node-app

**If you're hosting your app on a cloud server (like AWS EC2), and it listens on port 8080, you need to update the security group settings to allow traffic.**

**Allow Port 8080 in Security Group (for Cloud Hosting) If your app runs on a remote server (like AWS EC2) and listens on port 8080, you must allow incoming traffic to that port.**

****

Steps (Example: AWS EC2)

Go to the AWS EC2 Dashboard

Select your Instance

Under "Security", click the attached Security Group

Click Edit Inbound Rules

Add a new rule:

Type: Custom TCP

Port Range: 8080

Source: 0.0.0.0/0 (for public access) or restrict to your IP

Click Save Rules

**Visit your application in a browser at**[**http://publicip:8080**](http://publicip:8080/)**. You should see your hello world application up and running**

## **Benefits of Docker Multistage Builds:**

**Docker multistage builds let you use multiple FROM statements in one Dockerfile, each representing a different stage. You can copy only the necessary output (e.g., the final binary or build artifacts) from one stage to another, significantly improving the final image.**

## **Key Benefits:**

### **Smaller Image Size**

**Only the essential files are copied to the final image.**

**Dev/build-time dependencies (like compilers, test tools) are excluded.**

**Example: No node modules/Dev Dependencies or build tools in the final production image.**

### **Improved Security**

**Attack surface is reduced by excluding unnecessary tools or libraries.**

**No secrets or credentials left behind during the build.**

### **Faster Deployment**

**Smaller images mean faster upload/download times.**

**Less bandwidth and storage used.**

### **Cleaner Separation of Concerns**

**One stage handles the build, another runs the app.**

**Easier to maintain and update specific parts of the process.**

### **Reusable Build Logic**

**Build steps can be reused across multiple stages or containers.**

## **Example: Node.js App with Multistage dockerfile**

# Stage 1: Build

FROM node:18-alpine as builder

WORKDIR /app

COPY package\*.json ./

RUN npm install

COPY . .

RUN npm run build

# Stage 2: Production

FROM node:18-alpine

WORKDIR /app

COPY --from=builder /app ./

EXPOSE 3000

CMD ["node", "index.js"]

# Dockerfile Best Practices

#### **Minimize Image Layers**

**Combine RUN commands using && to reduce layers.**

**dockerfile RUN apt-get update && apt-get install -y curl 2. #### Use .dockerignore Just like .gitignore, prevents unnecessary files from being copied.**

**Example .dockerignore:**

**node modules**

.git

\*.log

.env

#### **Use Small Base Images**

**Prefer alpine or minimal base images for smaller size and lower attack surface.**

**dockerfile**

FROM node:18-alpine

#### **Pin Dependency Versions**

**Avoid unexpected updates and breakage.**

**dockerfile**

FROM node:18.17.0-alpine

#### **Avoid Installing Unnecessary Packages**

**Don't install tools or dependencies not required in production.**

#### **Leverage Multistage Builds**

**As shown above, use one stage for building and a second for runtime.**

#### **Set a Working Directory**

**Always use WORKDIR instead of manually managing paths.**

**dockerfile**

WORKDIR /app

#### **Use CMD Instead of RUN for Starting App**

**CMD defines the default command for container execution.**

**dockerfile**

CMD ["node", "index.js"]

#### **Don't Run as Root (if possible)**

**Create a user and switch to it for better security.**

**dockerfile**

RUN adduser -D appuser

USER appuser

#### **Clean Up Temporary Files**

**Especially in build stages, clean up cache or build files to reduce size.**

**docker-node-app$ history**

1  sudo apt update

    2  sudo apt install -y nodejs npm

    3  node -v

    4  npm -v

    5  # Install nvm

    6  curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash

    7  # Activate nvm (close and reopen terminal if needed)

    8  source ~/.bashrc

    9  # Install latest LTS version of Node.js

   10  nvm install --lts

   11  # Use that version

   12  nvm use --lts

   13  # Confirm version

   14  node -v

   15  npm -v

   16  sudo apt update

   17  sudo apt install -y apt-transport-https ca-certificates curl gnupg lsb-release

   18  curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

   19  sudo apt install docker.io

   20  docker --version

   21  sudo systemctl start docker

   22  sudo systemctl enable docker

   23  ~sudo systemctl enable docker

   26  git clone https://github.com/SafdarJamal/docker-node-app.git

   27  cd docker-node-app

   28  docker build -t docker-node-app .

   29  sudo docker build -t docker-node-app .

   30  docker build -t docker-node-app .

   31  sudo docker build -t docker-node-app .

   32  docker run -d -p 8080:8080 docker-node-app

   33  sudo docker run -d -p 8080:8080 docker-node-app