📦 Docker Custom Images + Node Server Dockerization Cheatsheet 🚀

> **Learn to build, optimize, and run custom Docker images for Node.js applications like a pro.**

📁 1. Folder Structure for a Dockerized Node App

```
my-app/
├── Dockerfile
├── .dockerignore
├── package.json
├── package-lock.json
├── index.js
```

🛠️ 2. Create a Custom Docker Image for Node.js App

☑️ Sample Optimized `Dockerfile`

```
# 🏗️ Stage 1: Build
FROM node:20-alpine AS builder

# Set working directory
WORKDIR /app

# Install dependencies
COPY package*.json ./
RUN npm ci

# Copy source code
COPY . .

# 🧹 Prune dev dependencies
RUN npm prune --production

# 🚀 Stage 2: Run
FROM node:20-alpine
WORKDIR /app

# Copy from builder stage
COPY --from=builder /app .

# Set environment and expose port
ENV NODE_ENV=production
ENV PORT=8000
EXPOSE 8000
```

```
# Run the app
CMD ["npm", "start"]
```

🚫 3. `.dockerignore` File (Very Important!)

```
node_modules
.dockerignore
Dockerfile
npm-debug.log
.git
.env
```

> Prevents unnecessary files from being copied into your image, reducing build time and size.

⚙️ 4. Build & Run Docker Image for Node App

🧱 Build the Image

```
# 🏷️ Build with a custom tag
docker build -t my-node-app .
```

🚀 Run the Container

```
# 🏷️ Run interactively
docker run -p 8000:8000 my-node-app
```

```
# 🏷️ Run in background
docker run -d -p 8000:8000 my-node-app
```

📂 Run with Mount (Dev Mode - Hot Reload)

```
docker run -v ${PWD}:/app -p 8000:8000 my-node-app
```

> Mounts your current directory inside the container, ideal for development.

🔍 5. Inspect, Debug, and Clean Containers

☑ List Containers & Images

```
docker ps           # Running containers
docker ps -a        # All containers
docker images       # Local images
```

🛑 Stop & Remove

```
docker stop <id>            # Stop container
docker rm <id>              # Remove container
docker rmi my-node-app      # Remove image
```

🪚 Exec Into a Container

```
docker exec -it <id> /bin/sh     # Alpine or BusyBox
docker exec -it <id> /bin/bash   # Ubuntu-based
```

📜 Logs & Details

```
docker logs <id>             # Show logs
docker inspect <id>          # Low-level config info
docker history my-node-app   # See image layers
```

🧹 6. Prune & Clean Docker Environment

```
docker container prune      # Remove stopped containers
docker image prune          # Remove dangling images
docker volume prune         # Clean volumes
docker system prune -a      # Remove all unused data
```

🛡 7. Best Practices for Custom Docker Images

Practice	Benefit
Use `alpine` base	Reduces image size drastically 📉

Practice	Benefit
Multi-stage builds	Keep final image clean & minimal
Set `.dockerignore`	Avoid bloat, faster builds
Cache dependencies	Speeds up builds
Pin base image versions	Prevents future breaking changes
Avoid root user	Increases container security
Clean caches in `RUN` steps	Reduces leftover junk

---

8. Example Node.js Server (`index.js`)

```
// index.js
const express = require('express');
const app = express();

const PORT = process.env.PORT || 8000;

app.get('/', (req, res) => {
  res.send('Hello from Dockerized Node Server 🚀');
});

app.listen(PORT, () => {
  console.log(`Server running on http://localhost:${PORT}`);
});
```

---

9. Image Size Analyzer Tools

```
docker history my-node-app      # View layers and size
docker image inspect my-node-app

# Use Dive CLI tool
dive my-node-app

# Use DockerSlim to shrink it
docker-slim build my-node-app
```

---

10. Docker Compose (Bonus)

```
# docker-compose.yml
version: '3.8'
services:
```

```
  web:
    build: .
    ports:
      - "8000:8000"
    volumes:
      - .:/app
    environment:
      - NODE_ENV=production
```

```
# Run with:
docker-compose up --build
```

✅ Summary of Key Docker Commands for Node Server

Task	Command
🧱 Build Image	`docker build -t my-node-app .`
🚀 Run Container	`docker run -p 8000:8000 my-node-app`
🔍 View Running	`docker ps`
🛑 Stop Container	`docker stop <id>`
🗑️ Prune Containers	`docker container prune`
🧹 Clean All	`docker system prune -a`

🎁 Bonus Tip

💡 To auto-rebuild and restart on file changes during dev:

```
npm install -D nodemon
```

Update `package.json`:

```
"scripts": {
  "start": "node index.js",
  "dev": "nodemon index.js"
}
```

Then in Dockerfile:

```
RUN npm install --only=production
# OR use dev mode in compose
```
