Docker Setup & Deployment Guide - Socket.IO App

For Complete Beginners - Step by Step

📚 What is Docker? (Simple Explanation)

Think of Docker like this:

```
Without Docker:
Your Computer (Windows)
  → Install Node.js
  → Install npm
  → Create .env file
  → npm install
  → npm run build
  → Works perfectly!

Then on Server (Linux):
  → Different Node version?
  → Different npm version?
  → Missing dependency?
  → Doesn't work! ❌

With Docker:
Your Computer → Package everything in a "box"
Server → Run the exact same "box"
Result → Works exactly the same! ✅
```

Docker = A container that includes:

- Node.js (exact version)
- npm (exact version)
- Your code
- All dependencies
- Configuration
- Everything needed to run

🚀 Install Docker

On Windows

1. **Download Docker Desktop**
   - Go to: https://www.docker.com/products/docker-desktop
   - Click "Download for Windows"
   - Run the installer
2. **Install:**
   - Follow the wizard (next, next, finish)
   - Restart your computer
   - Docker Desktop will launch automatically
3. **Verify Installation:**

```
docker --version
```

You should see: `Docker version 24.0.0` (or higher)

On Mac

```
# Using Homebrew (recommended)
brew install docker

# Or download Docker Desktop from:
# https://www.docker.com/products/docker-desktop
```

On Linux (Ubuntu/Debian)

```
# Update packages
sudo apt update

# Install Docker
sudo apt install docker.io -y

# Add your user to docker group
```

```
sudo usermod -aG docker $USER

# Verify
docker --version
```

📦 Docker Files for Your Project

1. Create `Dockerfile`

Create a new file named `Dockerfile` in your project root (same level as `package.json`):

```
# Use official Node.js image
FROM node:20-alpine

# Set working directory in container
WORKDIR /app

# Copy package files
COPY package*.json ./

# Install dependencies
RUN npm ci

# Copy project files
COPY . .

# Build Next.js
RUN npm run build

# Expose ports
EXPOSE 3000 3001

# Create non-root user for security
RUN addgroup -g 1001 -S nodejs
RUN adduser -S nextjs -u 1001
USER nextjs

# Start both services
CMD ["npm", "run", "dev:all"]
```

What each line does:

- `FROM node:20-alpine` - Start with a small Node.js image
- `WORKDIR /app` - Create /app folder inside container
- `COPY package*.json ./` - Copy your package files
- `RUN npm ci` - Install dependencies
- `COPY . .` - Copy your entire project
- `RUN npm run build` - Build Next.js
- `EXPOSE 3000 3001` - Open these ports
- `CMD ["npm", "run", "dev:all"]` - Run when container starts

2. Create `.dockerignore`

Create a file named `.dockerignore` in your project root:

```
node_modules
npm-debug.log
.next
.git
.gitignore
README.md
socket_events.db
.env.local
.env
dist
build
```

This tells Docker to ignore these files when building.

3. Create `docker-compose.yml`

Create a file named `docker-compose.yml` in your project root:

```yaml
version: '3.8'

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "3000:3000"  # Next.js frontend
      - "3001:3001"  # Socket.IO server
```

```
    environment:
      - NODE_ENV=production
      - SOCKET_PORT=3001
      - DATABASE_PATH=/app/socket_events.db
    volumes:
      - ./socket_events.db:/app/socket_events.db  # Persist database
    restart: unless-stopped
    container_name: socket-io-app
```

Explanation:

- `build:` - Build from Dockerfile
- `ports:` - Map container ports to host ports
- `environment:` - Set environment variables
- `volumes:` - Keep database between restarts
- `restart:` - Auto-restart if crashes
- `container_name:` - Name for the container

🏗️ Build Your Docker Image

Step 1: Open Terminal/Command Prompt

Navigate to your project folder:

```
cd path/to/socket-io-realtime-app
```

Step 2: Build the Image

```
docker build -t socket-io-app:1.0 .
```

What this does:

- `docker build` - Build a Docker image
- `-t socket-io-app:1.0` - Tag it with name:version
- `.` - Use Dockerfile in current directory

You'll see:

```
Step 1/10 : FROM node:20-alpine
Step 2/10 : WORKDIR /app
Step 3/10 : COPY package*.json ./
...
Successfully built abc123def456
Successfully tagged socket-io-app:1.0
```

⏱️ First build takes 5-10 minutes (downloads dependencies)

Step 3: Verify Image Created

```
docker images
```

You should see:

```
REPOSITORY          TAG     IMAGE ID      CREATED
socket-io-app       1.0     abc123def456  2 minutes ago
```

🎯 Run Your App Locally (Test Before Uploading)

Using Docker Compose (Recommended)

```
docker-compose up
```

Output:

```
socket-io-app-1  | > npm run dev:all
socket-io-app-1  | > concurrently "npm run dev" "npm run socket:watch"
socket-io-app-1  | [0] > next dev
socket-io-app-1  | [1] > nodemon --watch . --ext ts --exec 'ts-node --
project tsconfig.server.json' server.ts
```

✅ Now access:

- **Web App:** http://localhost:3000
- **Socket Server:** http://localhost:3001

Or Using Docker Run (Manual)

```
docker run -p 3000:3000 -p 3001:3001 \
  -e NODE_ENV=production \
  -v $(pwd)/socket_events.db:/app/socket_events.db \
  socket-io-app:1.0
```

Stop the Container

```
# Stop
docker-compose down

# Or with Ctrl+C in terminal
```

🔧 Docker Commands You'll Use

Basic Commands

```
# Build image
docker build -t socket-io-app:1.0 .

# List images
docker images

# Run container
docker run -p 3000:3000 socket-io-app:1.0

# List running containers
docker ps

# List all containers (including stopped)
docker ps -a

# Stop container
docker stop container_name

# Start container
docker start container_name

# Remove container
docker rm container_name
```

```
# Remove image
docker rmi socket-io-app:1.0

# View logs
docker logs container_name

# Follow logs (live)
docker logs -f container_name

# Execute command in running container
docker exec -it container_name bash

# Check container stats
docker stats
```

Docker Compose Commands

```
# Start services
docker-compose up

# Start in background
docker-compose up -d

# Stop services
docker-compose down

# View logs
docker-compose logs

# Follow logs
docker-compose logs -f

# Restart services
docker-compose restart

# Rebuild images
docker-compose build --no-cache
```

📤 Upload to Server

Prerequisites

You need:

- Server with Docker installed
- SSH access to server
- Domain name or server IP

Step 1: Prepare Your Project

```
# Make sure everything is committed
git add .
git commit -m "Add Docker configuration"
git push origin main
```

Step 2: SSH Into Server

```
ssh root@your-server-ip
# or
ssh username@your-server-ip
```

Step 3: Clone Your Repository

```
cd /opt
git clone https://github.com/yourusername/socket-io-realtime-app.git
cd socket-io-realtime-app
```

Step 4: Build Image on Server

```
docker build -t socket-io-app:1.0 .
```

⏱️ This takes 5-10 minutes on server

Step 5: Run Container

```
docker-compose up -d
```

Step 6: Verify It's Running

```
docker ps
```

You should see:

```
CONTAINER ID   IMAGE                 STATUS
abc123def456   socket-io-app:1.0     Up 2 minutes
```

Step 7: Check Logs

```
docker logs -f socket-io-app
```

Look for:

```
Socket.IO server running {"port":3001}
```

🌐 Configure Domain & Nginx

Create Nginx Config

SSH into server and create `/etc/nginx/sites-available/socket-io.conf`:

```
sudo nano /etc/nginx/sites-available/socket-io.conf
```

Paste this:

```
upstream socket_server {
    server 127.0.0.1:3001;
}

upstream next_app {
    server 127.0.0.1:3000;
}

server {
    listen 80;
    server_name apg-socket.com www.apg-socket.com;
```

```
    # Redirect HTTP to HTTPS
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl http2;
    server_name apg-socket.com www.apg-socket.com;

    ssl_certificate /etc/letsencrypt/live/apg-socket.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/apg-socket.com/privkey.pem;

    # Socket.IO endpoint
    location / {
        proxy_pass http://socket_server;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # Next.js app (admin panel)
    location /admin {
        proxy_pass http://next_app;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

Save: `Ctrl+X` then `Y` then `Enter`

Enable Nginx Config

```
sudo ln -s /etc/nginx/sites-available/socket-io.conf \
  /etc/nginx/sites-enabled/socket-io.conf

sudo nginx -t  # Test config

sudo systemctl restart nginx
```

Setup SSL Certificate (Free)

```
# Install Certbot
sudo apt install certbot python3-certbot-nginx -y

# Get certificate
sudo certbot certonly --nginx -d apg-socket.com -d www.apg-socket.com

# Auto-renew
sudo systemctl enable certbot.timer
sudo systemctl start certbot.timer
```

📊 Monitor Your Container

Check Status

```
# Is it running?
docker ps

# How much CPU/memory?
docker stats

# What are the logs?
docker logs -f socket-io-app
```

Common Issues & Solutions

1. Container Keeps Restarting

```
# Check logs
docker logs socket-io-app

# Might be database locked
docker exec socket-io-app npm run db:reset
```

2. Port Already in Use

```
# See what's using port 3000
lsof -i :3000

# Or change ports in docker-compose.yml
ports:
  - "3000:3000"  # Change first number
```

3. Database Not Persisting

Make sure `docker-compose.yml` has volumes:

```
volumes:
  - ./socket_events.db:/app/socket_events.db
```

4. Out of Disk Space

```
# Clean up unused images
docker image prune -a

# Clean up unused containers
docker container prune

# See disk usage
docker system df
```

🔄 Update Your App (After Making Changes)

When You Update Your Code

```
# On your computer - commit and push
git add .
git commit -m "Update feature X"
git push origin main

# On server - pull and rebuild
ssh root@your-server-ip
cd /opt/socket-io-realtime-app
git pull origin main
```

```
docker-compose down
docker build -t socket-io-app:1.0 .
docker-compose up -d
```

Faster Update (Without Rebuilding)

If you only changed TypeScript/config (not package.json):

```
docker-compose down
docker-compose up -d
```

📈 Production Best Practices

1. Use Environment File

Create `.env.production` on server:

```
sudo nano /opt/socket-io-realtime-app/.env.production
```

```
NODE_ENV=production
SOCKET_PORT=3001
DATABASE_PATH=/app/socket_events.db
CORS_ORIGINS=https://apg-socket.com,https://yourdomain.com
```

Update `docker-compose.yml` :

```
env_file:
  - .env.production
```

2. Auto-Restart on Reboot

```
docker-compose up -d

# Verify it's set to restart
docker inspect socket-io-app | grep -i "restart"
```

Should show:

```
"RestartPolicy": {
    "Name": "unless-stopped",
```

3. Regular Backups

```
# Backup database daily
sudo crontab -e

# Add this line:
0 2 * * * cp /opt/socket-io-realtime-app/socket_events.db /backups/socket
```

4. Monitor Logs

```
# View recent errors
docker logs --tail 100 socket-io-app | grep -i error

# Or use a monitoring tool
sudo apt install htop
htop  # See CPU/memory
```

5. Security

```
# Don't use root user
sudo useradd -m -s /bin/bash deploy
sudo usermod -aG docker deploy

# Use SSH keys instead of passwords
ssh-keygen -t ed25519
ssh-copy-id -i ~/.ssh/id_ed25519.pub deploy@server-ip

# Update regularly
sudo apt update
sudo apt upgrade
```

🎯 Step-by-Step Deployment Checklist

Before First Deploy

- [ ] Install Docker on your computer
- [ ] Create Dockerfile
- [ ] Create .dockerignore
- [ ] Create docker-compose.yml
- [ ] Test locally with `docker-compose up`
- [ ] Commit to git

First Deploy to Server

- [ ] Server has Docker installed
- [ ] SSH access working
- [ ] Domain registered and pointing to server
- [ ] Clone repository on server
- [ ] Build image: `docker build -t socket-io-app:1.0 .`
- [ ] Run container: `docker-compose up -d`
- [ ] Test: Visit http://your-domain.com

Post-Deploy

- [ ] Setup Nginx with domain
- [ ] Get SSL certificate with Certbot
- [ ] Test HTTPS connection
- [ ] Setup backups
- [ ] Monitor logs
- [ ] Document process

🚨 Troubleshooting

Problem: "docker: command not found"

```
# Reinstall Docker
# https://docs.docker.com/engine/install/

# Or try
sudo apt install docker.io
```

Problem: "Cannot connect to Docker daemon"

```
# Start Docker service
sudo service docker start

# Or
sudo systemctl start docker
```

Problem: "Permission denied while trying to connect"

```
# Add yourself to docker group
sudo usermod -aG docker $USER

# Apply new group
newgrp docker

# Or use sudo
sudo docker ps
```

Problem: "Build fails with 'npm ERR!'"

```
# Clear npm cache in Dockerfile
# Add this before RUN npm ci:

RUN npm cache clean --force
```

Problem: "Container runs but app not accessible"

```
# Check if listening on ports
docker exec socket-io-app lsof -i :3000

# Check logs
docker logs -f socket-io-app

# Verify Nginx
sudo nginx -t
sudo systemctl reload nginx
```

📚 Docker Quick Reference

File Structure After Setup

```
socket-io-realtime-app/
├── Dockerfile              ← Create this
├── .dockerignore           ← Create this
├── docker-compose.yml      ← Create this
├── .env.local
├── .env.production         ← Create on server
├── server.ts
├── app/
├── lib/
├── package.json
└── ...
```

Essential Docker Compose Commands

```bash
# Start all services in foreground
docker-compose up

# Start in background
docker-compose up -d

# Stop all services
docker-compose down

# View logs of all services
docker-compose logs

# Rebuild images (use when Dockerfile changes)
docker-compose build --no-cache

# Restart services
docker-compose restart

# See what's running
docker-compose ps
```

Useful Linux Commands on Server

```
# Check if Docker is running
sudo systemctl status docker

# Restart Docker
sudo systemctl restart docker

# See system resources
free -h           # Memory
df -h             # Disk space
top               # Live monitoring

# Check open ports
sudo netstat -tlnp | grep LISTEN

# Check file permissions
ls -la /path/to/file
```

✅ Verification Checklist

After deployment, verify everything works:

```
# 1. Container is running
docker ps

# 2. Can access web app
curl http://localhost:3000

# 3. Can access socket server
curl http://localhost:3001

# 4. Database exists
ls -la socket_events.db

# 5. Logs show no errors
docker logs socket-io-app

# 6. HTTPS works (if SSL configured)
curl https://apg-socket.com
```

📖 Next Steps

1. **Create the 3 Docker files** (Dockerfile, .dockerignore, docker-compose.yml)
2. **Test locally** with `docker-compose up`
3. **Push to GitHub** (commit docker files)
4. **Deploy to server** following the upload steps
5. **Configure Nginx** for your domain
6. **Setup SSL** with Certbot
7. **Monitor** the logs

You now have a **containerized production-ready app**! 🎉

🔗 Useful Resources

- **Docker Docs:** https://docs.docker.com
- **Docker Compose:** https://docs.docker.com/compose
- **Nginx Config:** https://nginx.org/en/docs
- **Certbot SSL:** https://certbot.eff.org
- **Node.js Docker:** https://hub.docker.com/_/node

Questions? Check the troubleshooting section or search "[Your Issue] Docker" online!