Nobel Auto Services — Deployment Requirements & Setup Guide

This document covers everything needed to deploy the full Nobel Auto Services platform on a fresh VPS from zero to running.

Table of Contents

---

1. What You Are Deploying

The platform is a **monorepo** with five services that run as Docker containers:

Container	What it does	Internal port
`postgres`	PostgreSQL 16 database — stores all data	5432
`api`	Node.js REST API (Express + Prisma)	4000
`ops`	React SPA — staff operations dashboard	80
`web`	React SPA — public website + customer portal	80
`proxy`	Nginx reverse proxy — single entry point for all traffic	80

Docker Compose orchestrates all five containers. The only port exposed to the internet is **80** (and **443** once SSL is configured). Everything else is internal.

URL layout once deployed:

URL path	Serves
`http://yourdomain.com/`	Public website (landing, services, booking)
`http://yourdomain.com/track`	Customer repair tracker / portal
`http://yourdomain.com/admin`	Staff ops dashboard (login required)
`http://yourdomain.com/api/`	REST API

---

2. Server Requirements

Minimum specs
Resource	Minimum	Recommended
CPU	1 vCPU	2 vCPU
RAM	2 GB	4 GB
Disk	20 GB SSD	40 GB SSD
OS	Ubuntu 22.04 LTS	Ubuntu 24.04 LTS
Architecture	x86_64 (amd64)	x86_64 (amd64)

> **Note on RAM:** The Docker build step (compiling TypeScript + Vite) requires ~1.5 GB RAM. If your VPS only has 1 GB, the build will fail. Use at least 2 GB or pre-build images on your local machine.

Recommended VPS providers
- **Hetzner Cloud** — CX22 (2 vCPU, 4 GB RAM) ~€5/mo — best value
- **DigitalOcean** — Basic Droplet 2 GB ~$12/mo
- **Linode/Akamai** — Nanode 2 GB ~$12/mo
- **Vultr** — Cloud Compute 2 GB ~$10/mo

---

3. Software Dependencies

On the VPS — only two things required

1. Docker Engine (v24+)
Provides the container runtime. Packages it installs automatically:
- `containerd.io` — low-level container runtime
- `docker-ce` — Docker daemon
- `docker-ce-cli` — Docker CLI
- `docker-buildx-plugin` — multi-platform builds

2. Docker Compose v2 (v2.20+)
Included as a plugin with modern Docker installs (`docker compose` not `docker-compose`).

> Everything else — Node.js 20, pnpm, nginx, PostgreSQL 16, tsx, Prisma — runs **inside Docker containers**. You do NOT install any of these on the VPS directly.

Installed automatically inside containers (for reference)

Tool	Version	Where	Purpose
Node.js	20 (alpine)	`api`, builder	JavaScript runtime
pnpm	9.12.0	builder	Package manager
Prisma CLI	5.22.0	`api`	DB migrations
Prisma Client	5.22.0	`api`	DB ORM

tsx	4.x	`api`	TypeScript runner
PostgreSQL	16 (alpine)	`postgres`	Database
Nginx	alpine	`ops`, `web`, `proxy`	Static file serving + reverse proxy

Optional (for SSL)
- **Certbot** — issues free Let's Encrypt SSL certificates
- **python3-certbot-nginx** — Certbot Nginx plugin

---

4. Port Map

Port	Protocol	Exposed to internet	Used by
22	TCP	Yes	SSH
80	TCP	Yes	Nginx proxy → all web traffic
443	TCP	Yes	HTTPS (after SSL setup)
4000	TCP	**No** — internal only	API container
5432	TCP	**No** — internal only	PostgreSQL container
3000	TCP	**No** — internal only	Ops SPA container
3001	TCP	**No** — internal only	Web SPA container

Firewall rules to set (UFW):
```bash
ufw allow 22/tcp
ufw allow 80/tcp
ufw allow 443/tcp
ufw enable
```

---

5. Environment Variables

Create a `.env` file in the project root (copy from `.env.example`). These are the variables the stack requires:

Variable	Required	Description	Example
`JWT_SECRET`	**Yes**	Signs access tokens. Min 32 random characters.	`openssl rand -hex 32`
`JWT_REFRESH_SECRET`	**Yes**	Signs refresh tokens. Must be different from JWT_SECRET.	`openssl rand -hex 32`
`DATABASE_URL`	Set by compose	Automatically set inside compose to the internal postgres container. Do not change unless using an external DB.	`postgresql://nobel:secret@postgres:5432/nobelauto`
`PORT`	No	API listen port inside container. Default: `4000`.	`4000`

| `NODE_ENV` | Set by compose | Automatically set to `production` by the compose file. | `production` |
| `CORS_ORIGINS` | No | Comma-separated extra allowed origins. Empty is fine when using the bundled nginx proxy (same-origin). | `` |

Postgres credentials (set inside docker-compose.yml)

Variable	Default	Where to change
`POSTGRES_DB`	`nobelauto`	`docker-compose.yml` → `postgres.environment`
`POSTGRES_USER`	`nobel`	`docker-compose.yml` → `postgres.environment`
`POSTGRES_PASSWORD`	`secret`	`docker-compose.yml` → `postgres.environment`

> **Important:** Change `POSTGRES_PASSWORD` from `secret` before first run. Update the matching `DATABASE_URL` in the `api` service environment inside `docker-compose.yml` to match.

---

6. Step-by-Step Setup

Step 1 — Provision the VPS

Log in as root or a sudo user via SSH:
```bash
ssh root@YOUR_SERVER_IP
```

Update system packages:
```bash
apt update && apt upgrade -y
```

Step 2 — Install Docker

```bash
# Download and run the official Docker install script
curl -fsSL https://get.docker.com | sh

# Add your non-root user to the docker group (replace "ubuntu" with your username)
usermod -aG docker ubuntu

# Apply group change without logging out
newgrp docker

# Verify installation
docker --version           # should show Docker version 24+
docker compose version     # should show Docker Compose version v2.20+
```

```

### Step 3 — Configure Firewall

```bash
ufw allow OpenSSH
ufw allow 80/tcp
ufw allow 443/tcp
ufw enable
ufw status
```

Step 4 — Clone the Repository

```bash
git clone https://github.com/aliawancb/nobel-auto.git
cd nobel-auto
```

Step 5 — Configure Environment

```bash
cp .env.example .env
```

Generate secure secrets:
```bash
echo "JWT_SECRET=$(openssl rand -hex 32)"
echo "JWT_REFRESH_SECRET=$(openssl rand -hex 32)"
```

Open `.env` and fill in the generated values:
```bash
nano .env
```

Your `.env` should look like:
```
DATABASE_URL=postgresql://nobel:YOUR_DB_PASSWORD@postgres:5432/nobelauto
JWT_SECRET=<paste generated value>
JWT_REFRESH_SECRET=<paste different generated value>
PORT=4000
NODE_ENV=production
CORS_ORIGINS=
```

Also update the Postgres password in `docker-compose.yml`:
```bash
```
```

nano docker-compose.yml
```
Find the `postgres` service and change `POSTGRES_PASSWORD: secret` to your chosen password. Also update the `DATABASE_URL` in the `api` service to match.

### Step 6 — Build and Start

```bash
docker compose up -d --build
```

This will:
1. Pull `postgres:16-alpine` and `nginx:alpine` from Docker Hub
2. Build the Node.js builder container (installs all npm packages, runs TypeScript compiler + Vite)
3. Build the `api-runner`, `ops-runner`, and `web-runner` containers
4. Start all five containers
5. On first start, the API container automatically runs `prisma migrate deploy` to create all database tables

> **First build takes 3–8 minutes** depending on server speed. Subsequent builds are faster due to Docker layer caching.

Step 7 — Seed Initial Data (first deploy only)

After the containers are up, seed the database with a default admin account and sample data:

```bash
docker compose exec api sh -c "cd /app && node -e \"
const { execSync } = require('child_process');
execSync('cd packages/db && npx tsx src/seed.ts', { stdio: 'inherit' });
\""
```

Or run it directly:
```bash
docker compose exec api sh -c "cd /app/packages/db && npx tsx src/seed.ts"
```

Default admin credentials created by the seed:
Field	Value
Email	`admin@nobelauto.com`
Password	`admin123`
Role	`admin`

> **Change the admin password immediately** after first login via Settings → Users.
```

Step 8 — Verify

```bash
# Check all containers are running
docker compose ps

# Check API health
curl http://localhost/api/health

# Check logs if something looks wrong
docker compose logs api
docker compose logs postgres
```

Expected output from `docker compose ps`:
```
NAME              STATUS          PORTS
nobel-postgres    Up (healthy)    5432/tcp
nobel-api         Up              4000/tcp
nobel-ops         Up              80/tcp
nobel-web         Up              80/tcp
nobel-proxy       Up              0.0.0.0:80->80/tcp
```

The app is now accessible at `http://YOUR_SERVER_IP`.

---

7. Verify Everything Works

Run through this checklist after deploy:

Public website
- [ ] `http://YOUR_IP/` loads the landing page
- [ ] `http://YOUR_IP/services` loads the services page
- [ ] `http://YOUR_IP/book` loads the booking form

Ops dashboard
- [ ] `http://YOUR_IP/admin` loads the login screen
- [ ] Login with `admin@nobelauto.com` / `admin123` succeeds
- [ ] Dashboard shows stats
- [ ] Work Orders page loads
- [ ] Customers page loads
- [ ] Inventory page loads (admin only)
- [ ] Settings page loads (admin only)

API

- [ ] `curl http://YOUR_IP/api/health` returns `{"status":"ok"}`
- [ ] `curl -X POST http://YOUR_IP/api/auth/login -H "Content-Type: application/json" -d '{"email":"admin@nobelauto.com","password":"admin123"}'` returns a token

Database
- [ ] `docker compose exec postgres psql -U nobel -d nobelauto -c "\dt"` lists all tables

---

8. SSL / Custom Domain (HTTPS)

Point your domain

In your domain registrar's DNS settings, create an **A record**:
```
Type: A
Name: @  (or your subdomain, e.g. "app")
Value: YOUR_SERVER_IP
TTL: 3600
```

Wait for DNS to propagate (5 minutes to 24 hours). Test with:
```bash
nslookup yourdomain.com
```

Install Certbot

```bash
apt install -y certbot python3-certbot-nginx
```

Obtain SSL Certificate

Stop the proxy container temporarily so Certbot can use port 80:
```bash
docker compose stop proxy
certbot certonly --standalone -d yourdomain.com -d www.yourdomain.com
docker compose start proxy
```

Update Nginx config for HTTPS

Edit `docker/nginx-proxy.conf` to add HTTPS and redirect HTTP:

```nginx
events { worker_connections 1024; }
```

```
http {
  upstream api { server api:4000; }
  upstream ops  { server ops:80; }
  upstream web  { server web:80; }

  # Redirect HTTP → HTTPS
  server {
    listen 80;
    server_name yourdomain.com www.yourdomain.com;
    return 301 https://$host$request_uri;
  }

  server {
    listen 443 ssl;
    server_name yourdomain.com www.yourdomain.com;

    ssl_certificate     /etc/letsencrypt/live/yourdomain.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/yourdomain.com/privkey.pem;
    ssl_protocols       TLSv1.2 TLSv1.3;

    location /api/ {
      proxy_pass http://api;
      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 https;
    }

    location /admin {
      proxy_pass http://ops;
      proxy_set_header Host $host;
    }

    location / {
      proxy_pass http://web;
      proxy_set_header Host $host;
    }
  }
}
```

Mount the certificates into the proxy container by updating `docker-compose.yml`:

```yaml
 proxy:
  image: nginx:alpine
  ports:
    - '80:80'
    - '443:443'
```

```
    volumes:
      - ./docker/nginx-proxy.conf:/etc/nginx/nginx.conf:ro
      - /etc/letsencrypt:/etc/letsencrypt:ro
    depends_on:
      - api
      - ops
      - web
    restart: unless-stopped
```

Restart the proxy:
```bash
docker compose up -d proxy
```

Auto-renew certificates

```bash
# Test renewal
certbot renew --dry-run

# Certbot installs a systemd timer automatically — verify it's active
systemctl status certbot.timer
```

Add a post-renew hook to restart the proxy after renewal:
```bash
cat > /etc/letsencrypt/renewal-hooks/post/restart-proxy.sh << 'EOF'
#!/bin/bash
cd /root/nobel-auto
docker compose restart proxy
EOF
chmod +x /etc/letsencrypt/renewal-hooks/post/restart-proxy.sh
```

---

9. Ongoing Operations

Update to latest code

```bash
cd ~/nobel-auto
git pull origin main
docker compose up -d --build
```

Compose will only rebuild containers whose source changed. Database data is preserved in the `pgdata` Docker volume.

View logs

```bash
docker compose logs -f          # all services
docker compose logs -f api      # API only
docker compose logs -f postgres # database only
```

Database backup

```bash
# Create a backup
docker compose exec postgres pg_dump -U nobel nobelauto > backup_$(date
+%Y%m%d).sql

# Restore from backup
docker compose exec -T postgres psql -U nobel nobelauto < backup_20260101.sql
```

Restart a single service

```bash
docker compose restart api
```

Stop everything

```bash
docker compose down          # stops containers, preserves volumes (data safe)
docker compose down -v       # stops containers AND deletes all data (destructive!)
```

Check disk usage

```bash
docker system df             # shows space used by images, containers, volumes
docker system prune          # remove unused images/containers (safe, keeps volumes)
```

---

10. Troubleshooting

Containers won't start

```bash
docker compose logs          # read the error output
docker compose ps            # check which containers exited
```

API says "database connection failed"

- Check postgres is healthy: `docker compose ps` — the postgres container should say `Up (healthy)`
- Check the `DATABASE_URL` in `.env` matches the credentials in `docker-compose.yml`
- Check postgres logs: `docker compose logs postgres`

"JWT_SECRET is required" error on startup

- You haven't created `.env` or it's missing the variable
- Run: `cp .env.example .env` and fill in the secrets

Build runs out of memory

The TypeScript + Vite build needs ~1.5 GB RAM. If your VPS has 1 GB:
```bash
# Add 2 GB swap space
fallocate -l 2G /swapfile
chmod 600 /swapfile
mkswap /swapfile
swapon /swapfile
echo '/swapfile none swap sw 0 0' >> /etc/fstab
```
Then retry the build.

Port 80 already in use

Another process (e.g. Apache or a system Nginx) is using port 80:
```bash
ss -tlnp | grep :80          # find what's using port 80
systemctl stop nginx         # if it's system nginx
systemctl disable nginx      # prevent it from starting again
```

Ops dashboard loads but shows blank/errors

Check browser console for network errors. Most likely the `/api/` path is not proxying correctly — check `docker compose logs proxy`.

Can't reach the server at all

Check firewall:
```bash
```

```
ufw status                  # confirm ports 80 and 443 are allowed
docker compose ps           # confirm proxy container is Up
```

Quick Reference

```bash
# First deploy
git clone https://github.com/aliawancb/nobel-auto.git && cd nobel-auto
cp .env.example .env && nano .env
docker compose up -d --build

# Update
git pull origin main && docker compose up -d --build

# Logs
docker compose logs -f api

# Backup database
docker compose exec postgres pg_dump -U nobel nobelauto > backup.sql

# Full restart
docker compose down && docker compose up -d
```

Default admin login: `admin@nobelauto.com` / `admin123` — change immediately after first deploy.