New Page

Table of Contents

- Prerequisites
- Server Requirements
- Pre-Deployment Checklist
- Deployment Methods
- Production Configuration
- Database Setup
- Environment Variables
- Running in Production
- Reverse Proxy Setup
- SSL/TLS Configuration
- Monitoring
- Backup Strategy
- Troubleshooting
- Rollback Procedures
- Performance Tuning

Prerequisites

Before deploying to production, ensure you have:

- ☐ Server with minimum specifications (see Server Requirements)
- ☐ Domain name configured
- ☐ SSL/TLS certificate
- ☐ PostgreSQL database server
- ☐ MongoDB server (if using Mitra MongoDB features)

- [ ] Firebase project (if using Firebase features)
- [ ] Backup solution
- [ ] Monitoring tools

Server Requirements

Minimum Specifications

Single Server Setup:

- **CPU**: 2 cores (4 cores recommended)
- **RAM**: 4GB (8GB recommended)
- **Storage**: 50GB SSD
- **OS**: Ubuntu 20.04 LTS or higher, CentOS 8+, or similar Linux distribution
- **Network**: Static IP address, ports 80/443 open

Multi-Server Setup (Recommended for Production):

- **Application Server**: 2-4 cores, 4-8GB RAM
- **Database Server**: 4-8 cores, 16-32GB RAM
- **Load Balancer**: 2 cores, 4GB RAM (if using multiple app servers)

Software Requirements

- **Node.js**: v18.x or v20.x LTS
- **npm**: v9.x or higher
- **PostgreSQL**: v12.x or higher
- **MongoDB**: v4.4 or higher (optional)
- **Nginx** or **Apache**: For reverse proxy
- **PM2** or **systemd**: For process management
- **Git**: For code deployment

Pre-Deployment Checklist

Code Preparation

- [ ] All tests passing
- [ ] Code reviewed and approved
- [ ] No console.log statements in production code
- [ ] All dependencies updated and audited (`npm audit`)
- [ ] Environment variables documented
- [ ] Database migrations prepared
- [ ] API documentation updated

Security Checklist

- [ ] JWT_SECRET is strong and unique
- [ ] Database passwords are strong
- [ ] CORS configured for production domains only
- [ ] Helmet.js security headers configured
- [ ] Rate limiting implemented (if required)
- [ ] Input validation enabled on all endpoints
- [ ] File upload size limits set
- [ ] SQL injection prevention verified
- [ ] XSS prevention verified

Infrastructure Checklist

- [ ] Database backups configured
- [ ] Log rotation configured
- [ ] Monitoring set up
- [ ] Alerting configured
- [ ] SSL certificate obtained and installed
- [ ] Firewall rules configured
- [ ] DNS records configured

Deployment Methods

Method 1: Manual Deployment

1. Server Setup

```
# Update system
sudo apt update && sudo apt upgrade -y

# Install Node.js (v20.x)
curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash -
sudo apt install -y nodejs

# Install PM2 globally
sudo npm install -g pm2

# Install PostgreSQL
sudo apt install -y postgresql postgresql-contrib

# Install Nginx
sudo apt install -y nginx

# Install Git
sudo apt install -y git
```

2. Create Application User

```
# Create user for running the application
sudo useradd -m -s /bin/bash ondelivery
sudo su - ondelivery
```

3. Clone Repository

```
# Clone the application
git clone <repository-url> /home/ondelivery/on-internal-api
cd /home/ondelivery/on-internal-api

# Checkout production branch
git checkout main  # or production branch
```

4. Install Dependencies

```
# Install production dependencies only
NODE_ENV=production npm ci
```

5. Configure Environment

```
# Create .env file
nano .env

# Add production environment variables (see Environment Variables section)
```

6. Set Permissions

```
# Ensure proper permissions
chmod 755 /home/ondelivery/on-internal-api
chmod 600 /home/ondelivery/on-internal-api/.env

# Create resources directories
mkdir -p resources/static/{employee,ticketing,cms,branding-approval,onapps}
mkdir -p resources/temp
```

Method 2: Docker Deployment

Docker Setup

Dockerfile:

```
FROM node:20-alpine

# Create app directory
WORKDIR /usr/src/app

# Install dependencies
COPY package*.json ./
RUN npm ci --only=production

# Copy app source
COPY . .

# Create directories
```

```
RUN mkdir -p resources/static resources/temp

# Expose port
EXPOSE 3601

# Start application
CMD ["node", "server.js"]
```

docker-compose.yml:

```
version: '3.8'

services:
  app:
    build: .
    ports:
      - "3601:3601"
    environment:
      - NODE_ENV=production
    env_file:
      - .env
    volumes:
      - ./resources:/usr/src/app/resources
    depends_on:
      - postgres
    restart: unless-stopped

  postgres:
    image: postgres:14
    environment:
      POSTGRES_USER: ${PG_SUNSHINE_USER}
      POSTGRES_PASSWORD: ${PG_SUNSHINE_PASSWORD}
      POSTGRES_DB: ${PG_SUNSHINE_DB}
    volumes:
      - postgres_data:/var/lib/postgresql/data
    restart: unless-stopped

volumes:
  postgres_data:
```

Deploy with Docker:

```
# Build and start
docker-compose up -d

# View logs
docker-compose logs -f app

# Stop
docker-compose down
```

Method 3: CI/CD Pipeline

GitHub Actions Example:

```
# .github/workflows/deploy.yml
name: Deploy to Production

on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - name: Deploy to Server
        uses: appleboy/ssh-action@master
        with:
          host: ${{ secrets.SERVER_HOST }}
          username: ${{ secrets.SERVER_USER }}
          key: ${{ secrets.SSH_PRIVATE_KEY }}
          script: |
            cd /home/ondelivery/on-internal-api
            git pull origin main
            npm ci --only=production
```

```
pm2 restart on-internal-api
```

Production Configuration

PM2 Configuration

ecosystem.config.js:

```
module.exports = {
  apps: [{
    name: 'on-internal-api',
    script: './server.js',
    instances: 'max',  // Use all available CPU cores
    exec_mode: 'cluster',
    env: {
      NODE_ENV: 'production',
      PORT: 3601
    },
    error_file: './logs/err.log',
    out_file: './logs/out.log',
    log_date_format: 'YYYY-MM-DD HH:mm:ss Z',
    merge_logs: true,
    max_memory_restart: '1G',
    autorestart: true,
    watch: false
  }]
};
```

Start with PM2:

```
# Start application
pm2 start ecosystem.config.js

# Save PM2 process list
pm2 save

# Setup PM2 to start on boot
```

```
pm2 startup
# Follow the instructions provided

# Monitor application
pm2 monit

# View logs
pm2 logs on-internal-api
```

Systemd Service (Alternative to PM2)

Create service file:

```
sudo nano /etc/systemd/system/on-internal-api.service
```

Service configuration:

```
[Unit]
Description=ON Internal API
After=network.target postgresql.service

[Service]
Type=simple
User=ondelivery
WorkingDirectory=/home/ondelivery/on-internal-api
ExecStart=/usr/bin/node /home/ondelivery/on-internal-api/server.js
Restart=on-failure
RestartSec=10
StandardOutput=syslog
StandardError=syslog
SyslogIdentifier=on-internal-api
Environment=NODE_ENV=production

[Install]
WantedBy=multi-user.target
```

Enable and start service:

```
sudo systemctl daemon-reload
sudo systemctl enable on-internal-api
sudo systemctl start on-internal-api
sudo systemctl status on-internal-api
```

Database Setup

PostgreSQL Configuration

1. Create Databases

```
-- Connect as postgres user
sudo -u postgres psql

-- Create databases
CREATE DATABASE sunshine_db;
CREATE DATABASE fleet_db;
CREATE DATABASE cms_db;
CREATE DATABASE mitra_db;

-- Create user
CREATE USER ondelivery WITH PASSWORD 'strong_password_here';

-- Grant privileges
GRANT ALL PRIVILEGES ON DATABASE sunshine_db TO ondelivery;
GRANT ALL PRIVILEGES ON DATABASE fleet_db TO ondelivery;
GRANT ALL PRIVILEGES ON DATABASE cms_db TO ondelivery;
GRANT ALL PRIVILEGES ON DATABASE mitra_db TO ondelivery;

\q
```

2. Run Migrations

```
cd /home/ondelivery/on-internal-api

# If using migrations
npm run migrate
```

```
# Or sync models (development approach)
# Uncomment sync commands in server.js temporarily
node server.js
# Then ctrl+C and comment them back out
```

3. PostgreSQL Performance Tuning

Edit `/etc/postgresql/14/main/postgresql.conf`:

```
# Memory settings (for 16GB RAM server)
shared_buffers = 4GB
effective_cache_size = 12GB
maintenance_work_mem = 1GB
work_mem = 32MB

# Connection settings
max_connections = 200

# Query optimization
random_page_cost = 1.1  # For SSD
effective_io_concurrency = 200

# Write-ahead log
wal_buffers = 16MB
min_wal_size = 1GB
max_wal_size = 4GB

# Checkpoints
checkpoint_completion_target = 0.9
```

Restart PostgreSQL:

```
sudo systemctl restart postgresql
```

MongoDB Configuration (If Used)

```
# Install MongoDB
wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key add -
```

```
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4 multiverse" |
sudo tee /etc/apt/sources.list.d/mongodb-org-4.4.list
sudo apt update
sudo apt install -y mongodb-org

# Start MongoDB
sudo systemctl start mongod
sudo systemctl enable mongod

# Secure MongoDB
mongo
use admin
db.createUser({
  user: "ondelivery",
  pwd: "strong_password",
  roles: ["readWriteAnyDatabase"]
})
exit

# Enable authentication in /etc/mongod.conf
sudo nano /etc/mongod.conf
```

Add:

```
security:
  authorization: enabled
```

Restart:

```
sudo systemctl restart mongod
```

Environment Variables

Production .env Template

```
# Server Configuration
NODE_ENV=production
```

```
SERVER_PORT=3601
BASE_DOMAIN=https://api.yourdomain.com
DIR=/home/ondelivery/resources

# PostgreSQL - Sunshine Database
PG_SUNSHINE_HOST=localhost
PG_SUNSHINE_PORT=5432
PG_SUNSHINE_USER=ondelivery
PG_SUNSHINE_PASSWORD=your_strong_password
PG_SUNSHINE_DB=sunshine_db
PG_SUNSHINE_DIALECT=postgres
PG_SUNSHINE_MAX_POOL=20
PG_SUNSHINE_MIN_POOL=5
PG_SUNSHINE_ACQUIRE_POOL=30000
PG_SUNSHINE_IDLE_POOL=10000

# PostgreSQL - Fleet Database
PG_FLEET_HOST=localhost
PG_FLEET_PORT=5432
PG_FLEET_USER=ondelivery
PG_FLEET_PASSWORD=your_strong_password
PG_FLEET_DB=fleet_db
PG_FLEET_dialect=postgres
PG_FLEET_MAX_POOL=10
PG_FLEET_MIN_POOL=2
PG_FLEET_ACQUIRE_POOL=30000
PG_FLEET_IDLE_POOL=10000

# PostgreSQL - CMS Database
PG_CMS_HOST=localhost
PG_CMS_PORT=5432
PG_CMS_USER=ondelivery
PG_CMS_PASSWORD=your_strong_password
PG_CMS_DB=cms_db
PG_CMS_dialect=postgres
PG_CMS_MAX_POOL=10
PG_CMS_MIN_POOL=2
PG_CMS_ACQUIRE_POOL=30000
PG_CMS_IDLE_POOL=10000
```

```
# PostgreSQL - Mitra Database
PG_MITRA_HOST=localhost
PG_MITRA_PORT=5432
PG_MITRA_USER=ondelivery
PG_MITRA_PASSWORD=your_strong_password
PG_MITRA_DB=mitra_db
PG_MITRA_DIALECT=postgres
PG_MITRA_MAX_POOL=10

# MongoDB (if used)
MONGO_MITRA=mongodb://ondelivery:your_strong_password@localhost:27017/mitra_db

# JWT Configuration
JWT_SECRET=your_very_long_random_secret_key_min_32_characters
JWT_EXPIRATION=86400

# Firebase Configuration (if used)
FIREBASE_SERVICE_ACCOUNT_PATH=/home/ondelivery/firebase-service-account.json
FIREBASE_DATABASE_URL=https://your-project.firebaseio.com
```

Running in Production

Start Application

With PM2:

```
cd /home/ondelivery/on-internal-api
pm2 start ecosystem.config.js
pm2 save
```

With systemd:

```
sudo systemctl start on-internal-api
sudo systemctl status on-internal-api
```

Verify Application

```
# Check if application is running
curl http://localhost:3601/
# Should return: {"message":"sunshine"}

# Check logs
pm2 logs on-internal-api
# OR
sudo journalctl -u on-internal-api -f
```

Reverse Proxy Setup

Nginx Configuration

Create Nginx config:

```
sudo nano /etc/nginx/sites-available/on-internal-api
```

Configuration:

```
upstream on_internal_api {
    # If using PM2 cluster mode
    server 127.0.0.1:3601;

    # Add more if running multiple instances manually
    # server 127.0.0.1:3602;
    # server 127.0.0.1:3603;
}

server {
    listen 80;
    server_name api.yourdomain.com;

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

server {
```

```
listen 443 ssl http2;
server_name api.yourdomain.com;

# SSL Configuration
ssl_certificate /etc/letsencrypt/live/api.yourdomain.com/fullchain.pem;
ssl_certificate_key /etc/letsencrypt/live/api.yourdomain.com/privkey.pem;
ssl_protocols TLSv1.2 TLSv1.3;
ssl_ciphers HIGH:!aNULL:!MD5;
ssl_prefer_server_ciphers on;

# Security Headers
add_header X-Frame-Options "SAMEORIGIN" always;
add_header X-Content-Type-Options "nosniff" always;
add_header X-XSS-Protection "1; mode=block" always;

# Client upload size
client_max_body_size 10M;

# Timeouts
proxy_connect_timeout 60s;
proxy_send_timeout 60s;
proxy_read_timeout 60s;

# Proxy settings
location / {
    proxy_pass http://on_internal_api;
    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;
    proxy_cache_bypass $http_upgrade;
}

# WebSocket support
location /ws {
    proxy_pass http://on_internal_api;
    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_read_timeout 86400;
    }
}
```

Enable site:

```
sudo ln -s /etc/nginx/sites-available/on-internal-api /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl reload nginx
```

SSL/TLS Configuration

Using Let's Encrypt (Recommended)

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

# Obtain certificate
sudo certbot --nginx -d api.yourdomain.com

# Test auto-renewal
sudo certbot renew --dry-run
```

Manual Certificate Installation

If using a purchased certificate:

```
# Copy certificate files
sudo cp fullchain.pem /etc/ssl/certs/api.yourdomain.com.crt
sudo cp privkey.pem /etc/ssl/private/api.yourdomain.com.key
sudo chmod 644 /etc/ssl/certs/api.yourdomain.com.crt
sudo chmod 600 /etc/ssl/private/api.yourdomain.com.key
```

```
# Update Nginx config with certificate paths
sudo nano /etc/nginx/sites-available/on-internal-api
```

Monitoring

Application Monitoring with PM2

```
# Install PM2 Plus for monitoring (optional)
pm2 install pm2-logrotate

# Configure log rotation
pm2 set pm2-logrotate:max_size 10M
pm2 set pm2-logrotate:retain 7
pm2 set pm2-logrotate:compress true
```

System Monitoring

Install monitoring tools:

```
sudo apt install -y htop iotop nethogs
```

Monitor resources:

```
# CPU and memory
htop

# Disk I/O
iotop

# Network usage
nethogs
```

Log Management

Create log directory:

```
mkdir -p /home/ondelivery/on-internal-api/logs
```

Configure log rotation:

```
sudo nano /etc/logrotate.d/on-internal-api
```

```
/home/ondelivery/on-internal-api/logs/*.log {
    daily
    rotate 14
    compress
    delaycompress
    notifempty
    create 0640 ondelivery ondelivery
    sharedscripts
    postrotate
        pm2 reloadLogs
    endscript
}
```

External Monitoring (Recommended)

Consider using:

- **New Relic** - Application performance monitoring
- **Datadog** - Infrastructure and application monitoring
- **Sentry** - Error tracking
- **UptimeRobot** - Uptime monitoring

Backup Strategy

Database Backups

Automated PostgreSQL backup script:

```
#!/bin/bash
# /home/ondelivery/scripts/backup-databases.sh
```

```
BACKUP_DIR="/home/ondelivery/backups/databases"
DATE=$(date +%Y%m%d_%H%M%S)
RETENTION_DAYS=30

# Create backup directory
mkdir -p $BACKUP_DIR

# Backup each database
pg_dump -h localhost -U ondelivery sunshine_db > $BACKUP_DIR/sunshine_db_$DATE.sql
pg_dump -h localhost -U ondelivery fleet_db > $BACKUP_DIR/fleet_db_$DATE.sql
pg_dump -h localhost -U ondelivery cms_db > $BACKUP_DIR/cms_db_$DATE.sql
pg_dump -h localhost -U ondelivery mitra_db > $BACKUP_DIR/mitra_db_$DATE.sql

# Compress backups
gzip $BACKUP_DIR/*_$DATE.sql

# Delete old backups
find $BACKUP_DIR -name "*.sql.gz" -mtime +$RETENTION_DAYS -delete

echo "Database backup completed: $DATE"
```

Make executable and schedule:

```
chmod +x /home/ondelivery/scripts/backup-databases.sh

# Add to crontab (daily at 2 AM)
crontab -e
# Add: 0 2 * * * /home/ondelivery/scripts/backup-databases.sh
```

File Backups

```
#!/bin/bash
# Backup resources directory
tar -czf /home/ondelivery/backups/resources_$(date +%Y%m%d).tar.gz \
    /home/ondelivery/on-internal-api/resources

# Keep last 7 days
find /home/ondelivery/backups -name "resources_*.tar.gz" -mtime +7 -delete
```

Troubleshooting

Application Won't Start

```
# Check logs
pm2 logs on-internal-api --lines 100

# Check if port is available
sudo lsof -i :3601

# Check environment variables
pm2 env 0

# Test database connections
psql -h localhost -U ondelivery -d sunshine_db
```

High Memory Usage

```
# Check memory usage
pm2 list
free -h

# Restart application
pm2 restart on-internal-api

# If issue persists, reduce max_memory_restart in ecosystem.config.js
```

Database Connection Errors

```
# Check PostgreSQL status
sudo systemctl status postgresql

# Check connections
sudo -u postgres psql -c "SELECT count(*) FROM pg_stat_activity;"
```

```
# Check connection limits
sudo -u postgres psql -c "SHOW max_connections;"

# Increase if needed in postgresql.conf
```

SSL Certificate Issues

```
# Test certificate
sudo certbot certificates

# Renew if needed
sudo certbot renew

# Check Nginx config
sudo nginx -t
```

Rollback Procedures

Quick Rollback

```
cd /home/ondelivery/on-internal-api

# Checkout previous version
git log --oneline  # Find commit hash
git checkout <previous-commit-hash>

# Install dependencies
npm ci --only=production

# Restart application
pm2 restart on-internal-api
```

Database Rollback

```
# Restore from backup
gunzip < /home/ondelivery/backups/databases/sunshine_db_20240115_020000.sql.gz | \
    psql -h localhost -U ondelivery sunshine_db
```

Performance Tuning

Node.js Optimization

```
// ecosystem.config.js
module.exports = {
  apps: [{
    name: 'on-internal-api',
    script: './server.js',
    instances: 'max',
    exec_mode: 'cluster',
    node_args: [
      '--max-old-space-size=2048',  // Max heap size
      '--optimize-for-size',
      '--gc-interval=100'
    ]
  }]
};
```

Database Query Optimization

- Add indexes on frequently queried columns
- Use connection pooling effectively
- Implement caching for read-heavy operations
- Use database query monitoring

Caching Strategy (Redis - Optional)

```
# Install Redis
sudo apt install -y redis-server
```

```
# Configure Redis
sudo nano /etc/redis/redis.conf
# Set maxmemory and maxmemory-policy

# Restart Redis
sudo systemctl restart redis
```

Post-Deployment Verification

After deployment, verify:

- ☐ Application is running and accessible
- ☐ All API endpoints working
- ☐ Database connections established
- ☐ File uploads working
- ☐ WebSocket connections working
- ☐ SSL certificate valid
- ☐ Logs are being written
- ☐ Scheduled tasks running
- ☐ Backups configured and tested
- ☐ Monitoring alerts configured

Document Version: 1.0.0
Last Updated: 2026-02-04

For deployment support, contact the DevOps team.

Revision #1
Created 5 February 2026 03:39:04 by ondeliveloper
Updated 5 February 2026 03:39:37 by ondeliveloper