Arborist Deployment Guide

TypeScript Application Deployment

Rocky Linux on Proxmox Home Lab

Production Simulation & User Authentication Testing

Infrastructure & DevOps Reference

January 2026
Version 1.0

Abstract

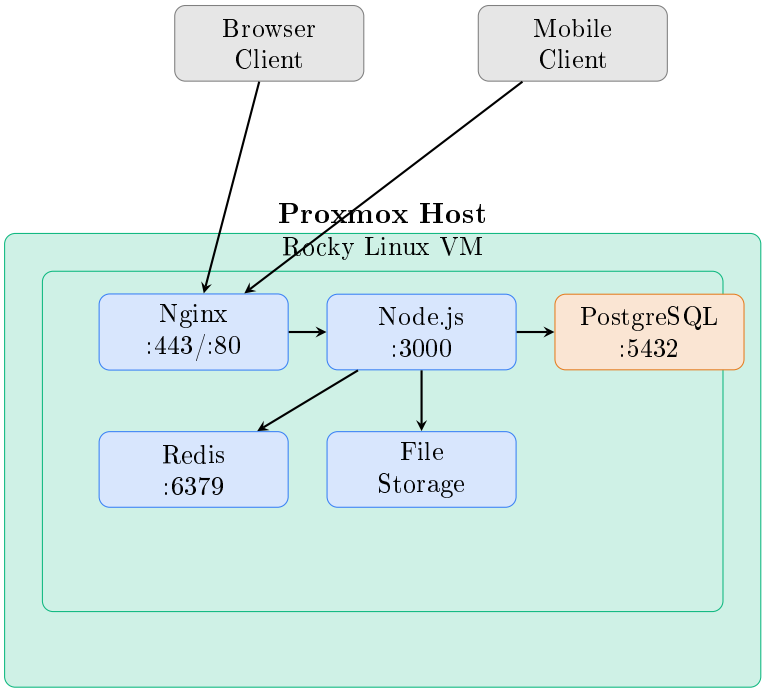
This guide provides step-by-step instructions for deploying a TypeScript web application to a Rocky Linux server running on Proxmox in a home lab environment. It covers VM provisioning, application packaging, reverse proxy configuration, SSL/TLS setup, database deployment, authentication system integration, and production simulation techniques. The goal is to create a realistic staging environment for testing user authentication and multi-user scenarios before going live.

Contents

Part I
Infrastructure Setup

1 Architecture Overview

1.1 Deployment Architecture



1.2 Component Summary

Component	Technology	Purpose
Hypervisor	Proxmox VE	VM/container hosting
Guest OS	Rocky Linux 9	Production-like RHEL environment
Reverse Proxy	Nginx	SSL termination, static files, load balancing
Runtime	Node.js 20 LTS	Application server
Process Manager	PM2	Process management, clustering, monitoring
Database	PostgreSQL 16	User data, application state
Cache/Sessions	Redis	Session storage, caching
SSL	Let's Encrypt / Self-signed	HTTPS encryption

Table 1: Technology Stack

2 Proxmox VM Setup

2.1 Create Rocky Linux VM

1. **Download Rocky Linux ISO**

   In Proxmox web UI, navigate to your storage and download the Rocky Linux 9 minimal ISO:

```
# Or download directly on Proxmox host
cd /var/lib/vz/template/iso/
wget https://download.rockylinux.org/pub/rocky/9/isos/x86_64/
    Rocky-9-latest-x86_64-minimal.iso
```

2. **Create VM in Proxmox UI**

   - Click "Create VM"
   - **General**: Name: `arborist-staging`, VM ID: auto
   - **OS**: Select Rocky Linux ISO, Type: Linux, Version: 6.x - 2.6 Kernel
   - **System**: BIOS: OVMF (UEFI), EFI Storage: local-lvm, Add TPM: unchecked
   - **Disks**: Bus: VirtIO Block, Size: 50GB, SSD emulation: checked
   - **CPU**: Cores: 4 (adjust based on host), Type: host
   - **Memory**: 8192 MB (8GB)
   - **Network**: Bridge: vmbr0, Model: VirtIO

3. **Install Rocky Linux**

   Boot VM and complete installation:

   - Select "Minimal Install" for server workload
   - Configure network with static IP (recommended) or DHCP
   - Set root password and create admin user
   - Enable automatic partitioning

2.2 Post-Installation Configuration

After Rocky Linux boots, SSH into the VM and run initial setup:

```
# Update system
sudo dnf update -y

# Install essential tools
sudo dnf install -y \
  vim \
  git \
  curl \
  wget \
  htop \
  tmux \
  unzip \
  tar \
  firewalld \
  policycoreutils-python-utils

# Enable and start firewalld
sudo systemctl enable --now firewalld

# Set hostname
sudo hostnamectl set-hostname arborist-staging

# Configure timezone
sudo timedatectl set-timezone America/New_York  # Adjust as needed

# Disable SELinux temporarily for initial setup (re-enable later)
sudo setenforce 0
```

```
sudo sed -i 's/SELINUX=enforcing/SELINUX=permissive/' /etc/selinux/
    config
```

2.3 Network Configuration

```
# Check current IP
ip addr show

# For static IP (if not set during install)
sudo nmcli con mod "enp6s18" ipv4.addresses 192.168.1.100/24
sudo nmcli con mod "enp6s18" ipv4.gateway 192.168.1.1
sudo nmcli con mod "enp6s18" ipv4.dns "1.1.1.1,8.8.8.8"
sudo nmcli con mod "enp6s18" ipv4.method manual
sudo nmcli con up "enp6s18"

# Open firewall ports
sudo firewall-cmd --permanent --add-service=http
sudo firewall-cmd --permanent --add-service=https
sudo firewall-cmd --permanent --add-service=ssh
sudo firewall-cmd --reload

# Verify
sudo firewall-cmd --list-all
```

Static IP Recommendation

For a staging server, use a static IP address to ensure consistent access. Document this IP in your `/etc/hosts` file on your development machine for easy access:

```
# On your dev machine, add to /etc/hosts:
192.168.1.100   arborist-staging arborist.local
```

3 Runtime Environment

3.1 Install Node.js 20 LTS

```
# Add NodeSource repository for Node.js 20
curl -fsSL https://rpm.nodesource.com/setup_20.x | sudo bash -

# Install Node.js
sudo dnf install -y nodejs

# Verify installation
node --version    # Should show v20.x.x
npm --version     # Should show 10.x.x

# Install pnpm globally (faster, more efficient)
sudo npm install -g pnpm

# Verify pnpm
pnpm --version
```

3.2 Install PM2 Process Manager

```
# Install PM2 globally
sudo npm install -g pm2

# Setup PM2 to start on boot
pm2 startup systemd -u $USER --hp /home/$USER
# Run the command it outputs

# Verify PM2
pm2 --version
```

3.3 Install PostgreSQL 16

```
# Add PostgreSQL repository
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/
    reporpms/EL-9-x86_64/pgdg-redhat-repo-latest.noarch.rpm

# Disable built-in PostgreSQL module
sudo dnf -qy module disable postgresql

# Install PostgreSQL 16
sudo dnf install -y postgresql16-server postgresql16

# Initialize database
sudo /usr/pgsql-16/bin/postgresql-16-setup initdb

# Start and enable PostgreSQL
sudo systemctl enable --now postgresql-16

# Verify
sudo systemctl status postgresql-16
```

3.3.1 Configure PostgreSQL

```
# Switch to postgres user
sudo -u postgres psql

-- Create application database and user
CREATE USER arborist WITH PASSWORD 'your-secure-password-here';
CREATE DATABASE arborist_db OWNER arborist;
GRANT ALL PRIVILEGES ON DATABASE arborist_db TO arborist;

-- Enable UUID extension
\c arborist_db
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

-- Exit
\q
```

```
# Configure PostgreSQL to accept local connections
sudo vim /var/lib/pgsql/16/data/pg_hba.conf
```

```
# Add/modify this line for local connections:
# TYPE  DATABASE        USER            ADDRESS                 METHOD
local   all             all                                     md5
host    all             all             127.0.0.1/32            md5
host    all             all             ::1/128                 md5
```

```
# Restart PostgreSQL
sudo systemctl restart postgresql-16

# Test connection
psql -U arborist -d arborist_db -h localhost
# Enter password when prompted
```

3.4 Install Redis

```
# Install Redis
sudo dnf install -y redis

# Start and enable Redis
sudo systemctl enable --now redis

# Verify
redis-cli ping
# Should return: PONG

# Configure Redis password (recommended)
sudo vim /etc/redis/redis.conf
# Find and set: requirepass your-redis-password-here

# Restart Redis
sudo systemctl restart redis
```

3.5 Install Nginx

```
# Install Nginx
sudo dnf install -y nginx

# Start and enable Nginx
sudo systemctl enable --now nginx

# Verify
sudo systemctl status nginx
curl http://localhost
```

Part II
Application Packaging

4 Build Process

4.1 Project Structure for Deployment

Ensure your project has the following structure:

```
arborist/
          package.json
          pnpm-lock.yaml
          tsconfig.json
          vite.config.ts          # Frontend build config
          src/
                client/             # Frontend React/TypeScript
                      index.tsx
                      ...
                server/             # Backend API (if applicable)
                    index.ts
                    ...
          public/                 # Static assets
          prisma/                 # Database schema (if using Prisma)
                schema.prisma
          .env.example            # Environment template
          Dockerfile              # Container build (optional)
          ecosystem.config.js     # PM2 configuration
```

4.2 Create Build Scripts

Update your `package.json` with production build scripts:

```
{
  "name": "arborist",
  "version": "1.0.0",
  "scripts": {
    "dev": "vite",
    "build": "tsc && vite build",
    "build:server": "tsc -p tsconfig.server.json",
    "build:all": "pnpm build && pnpm build:server",
    "start": "node dist/server/index.js",
    "preview": "vite preview",
    "lint": "eslint src --ext ts,tsx",
    "typecheck": "tsc --noEmit",
    "db:migrate": "prisma migrate deploy",
    "db:generate": "prisma generate",
    "prepare": "pnpm db:generate"
  },
  "engines": {
    "node": ">=20.0.0"
  }
}
```

4.3 Create PM2 Ecosystem File

```
// ecosystem.config.js
module.exports = {
  apps: [
```

```
    {
      name: 'arborist-api',
      script: './dist/server/index.js',
      instances: 'max',           // Use all CPU cores
      exec_mode: 'cluster',       // Enable cluster mode
      env: {
        NODE_ENV: 'development',
        PORT: 3000,
      },
      env_production: {
        NODE_ENV: 'production',
        PORT: 3000,
      },
      // Logging
      log_file: '/var/log/arborist/combined.log',
      out_file: '/var/log/arborist/out.log',
      error_file: '/var/log/arborist/error.log',
      log_date_format: 'YYYY-MM-DD HH:mm:ss Z',
      // Process management
      max_memory_restart: '1G',
      exp_backoff_restart_delay: 100,
      // Watch for changes (disable in production)
      watch: false,
      ignore_watch: ['node_modules', 'logs', '.git'],
    },
  ],
};
```

4.4 Create Environment Template

```
# .env.example - Copy to .env and fill in values

# Application
NODE_ENV=production
PORT=3000
APP_URL=https://arborist.local

# Database
DATABASE_URL=postgresql://arborist:password@localhost:5432/
    arborist_db

# Redis
REDIS_URL=redis://:password@localhost:6379

# Authentication (Lucia)
AUTH_SECRET=generate-a-32-char-secret-here

# Session
SESSION_SECRET=another-32-char-secret-here
SESSION_MAX_AGE=604800  # 7 days in seconds

# OAuth (optional)
GITHUB_CLIENT_ID=
GITHUB_CLIENT_SECRET=
GOOGLE_CLIENT_ID=
GOOGLE_CLIENT_SECRET=
```

```
# File Storage
STORAGE_PATH=/var/lib/arborist/storage
MAX_UPLOAD_SIZE=52428800  # 50MB

# Sync Service (if applicable)
SYNC_ENABLED=true
SYNC_ENDPOINT=wss://arborist.local/sync
```

4.5 Build Application Locally

```
# On your development machine

# Install dependencies
pnpm install

# Run type checking
pnpm typecheck

# Run linter
pnpm lint

# Build frontend and backend
pnpm build:all

# Verify build output
ls -la dist/
# Should contain:
#   - client/     (built frontend assets)
#   - server/     (compiled backend)
```

5 Deployment Package

5.1 Create Deployment Archive

```
# Create deployment directory
mkdir -p deploy

# Copy necessary files
cp -r dist/ deploy/
cp package.json deploy/
cp pnpm-lock.yaml deploy/
cp ecosystem.config.js deploy/
cp .env.example deploy/
cp -r prisma/ deploy/  # If using Prisma

# Create archive
tar -czvf arborist-$(date +%Y%m%d-%H%M%S).tar.gz -C deploy .

# Verify archive
tar -tzvf arborist-*.tar.gz | head -20
```

5.2 Alternative: Git-based Deployment

For ongoing development, use Git for deployments:

```
# On your development machine, ensure code is pushed
git add .
git commit -m "Prepare for deployment"
git push origin main

# On the server, clone the repository
cd /var/www
sudo git clone https://github.com/yourusername/arborist.git
sudo chown -R $USER:$USER arborist/
```

Part III
Server Deployment

6 Deploy Application

6.1 Create Application Directory

```
# On the Rocky Linux server

# Create application directories
sudo mkdir -p /var/www/arborist
sudo mkdir -p /var/log/arborist
sudo mkdir -p /var/lib/arborist/storage

# Set ownership
sudo chown -R $USER:$USER /var/www/arborist
sudo chown -R $USER:$USER /var/log/arborist
sudo chown -R $USER:$USER /var/lib/arborist
```

6.2 Transfer Application Files

```
# From your development machine

# Option 1: SCP the archive
scp arborist-*.tar.gz user@arborist-staging:/var/www/arborist/

# Option 2: Rsync (better for updates)
rsync -avz --progress \
  --exclude='node_modules' \
  --exclude='.git' \
  --exclude='.env' \
  ./deploy/ user@arborist-staging:/var/www/arborist/
```

6.3 Install Dependencies on Server

```
# On the server
```

```
cd /var/www/arborist

# If using archive, extract it
tar -xzvf arborist-*.tar.gz

# Install production dependencies only
pnpm install --prod --frozen-lockfile

# Generate Prisma client (if using Prisma)
pnpm db:generate
```

6.4 Configure Environment

```
# Create production environment file
cp .env.example .env
vim .env

# Generate secrets
openssl rand -base64 32  # Use output for AUTH_SECRET
openssl rand -base64 32  # Use output for SESSION_SECRET

# Set secure permissions
chmod 600 .env
```

6.5 Run Database Migrations

```
# Apply database migrations
pnpm db:migrate

# Verify tables were created
psql -U arborist -d arborist_db -h localhost -c "\dt"
```

6.6 Start Application with PM2

```
# Start the application
pm2 start ecosystem.config.js --env production

# Verify it's running
pm2 status
pm2 logs arborist-api --lines 50

# Save PM2 process list
pm2 save

# Test the application
curl http://localhost:3000/health
# Should return: {"status":"ok"}
```

7 Nginx Configuration

7.1 Create Nginx Site Configuration

```
# /etc/nginx/conf.d/arborist.conf

# Upstream for Node.js app
upstream arborist_backend {
    server 127.0.0.1:3000;
    keepalive 64;
}

# HTTP - Redirect to HTTPS
server {
    listen 80;
    listen [::]:80;
    server_name arborist.local arborist-staging;

    # Allow Let's Encrypt challenges
    location /.well-known/acme-challenge/ {
        root /var/www/certbot;
    }

    # Redirect all other traffic to HTTPS
    location / {
        return 301 https://$host$request_uri;
    }
}

# HTTPS
server {
    listen 443 ssl http2;
    listen [::]:443 ssl http2;
    server_name arborist.local arborist-staging;

    # SSL Configuration (update paths after cert generation)
    ssl_certificate /etc/nginx/ssl/arborist.crt;
    ssl_certificate_key /etc/nginx/ssl/arborist.key;

    # Modern SSL settings
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-
        SHA256;
    ssl_prefer_server_ciphers off;
    ssl_session_timeout 1d;
    ssl_session_cache shared:SSL:10m;
    ssl_session_tickets off;

    # 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;
    add_header Referrer-Policy "strict-origin-when-cross-origin"
        always;

    # Logging
    access_log /var/log/nginx/arborist_access.log;
    error_log /var/log/nginx/arborist_error.log;

```

```
    # Max upload size
    client_max_body_size 50M;

    # Gzip compression
    gzip on;
    gzip_vary on;
    gzip_proxied any;
    gzip_comp_level 6;
    gzip_types text/plain text/css text/xml application/json
       application/javascript application/rss+xml application/atom+
       xml image/svg+xml;

    # Root for static files
    root /var/www/arborist/dist/client;
    index index.html;

    # Static assets with cache
    location /assets/ {
        expires 1y;
        add_header Cache-Control "public, immutable";
    }

    # API proxy
    location /api/ {
        proxy_pass http://arborist_backend;
        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;
        proxy_read_timeout 86400s;
        proxy_send_timeout 86400s;
    }

    # WebSocket for sync
    location /sync {
        proxy_pass http://arborist_backend;
        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 86400s;
    }

    # Auth endpoints
    location /auth/ {
        proxy_pass http://arborist_backend;
        proxy_http_version 1.1;
        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;
    }

```

```
    # SPA fallback - serve index.html for client routes
    location / {
        try_files $uri $uri/ /index.html;
    }

    # Health check
    location /health {
        proxy_pass http://arborist_backend;
        proxy_http_version 1.1;
    }
}
```

7.2 Generate Self-Signed SSL Certificate

For local/staging testing, create a self-signed certificate:

```
# Create SSL directory
sudo mkdir -p /etc/nginx/ssl

# Generate self-signed certificate
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
  -keyout /etc/nginx/ssl/arborist.key \
  -out /etc/nginx/ssl/arborist.crt \
  -subj "/C=US/ST=State/L=City/O=Arborist/CN=arborist.local" \
  -addext "subjectAltName=DNS:arborist.local,DNS:arborist-staging,IP
      :192.168.1.100"

# Set permissions
sudo chmod 600 /etc/nginx/ssl/arborist.key
sudo chmod 644 /etc/nginx/ssl/arborist.crt
```

Browser Warning

Self-signed certificates will show a browser warning. For local testing, you can:

- Accept the warning and proceed
- Add the certificate to your system's trust store
- Use mkcert for locally-trusted certificates

7.3 Test and Apply Nginx Configuration

```
# Test configuration syntax
sudo nginx -t

# If test passes, reload Nginx
sudo systemctl reload nginx

# Verify Nginx is serving
curl -k https://localhost/health
# -k flag ignores SSL certificate warnings
```

8 SELinux Configuration

For production, re-enable SELinux with proper policies:

```
# Allow Nginx to connect to upstream
sudo setsebool -P httpd_can_network_connect 1

# Allow Nginx to serve files from /var/www
sudo semanage fcontext -a -t httpd_sys_content_t "/var/www/arborist(/.*)?"
sudo restorecon -Rv /var/www/arborist

# Allow Nginx to read SSL certs
sudo semanage fcontext -a -t cert_t "/etc/nginx/ssl(/.*)?"
sudo restorecon -Rv /etc/nginx/ssl

# Re-enable SELinux
sudo setenforce 1
sudo sed -i 's/SELINUX=permissive/SELINUX=enforcing/' /etc/selinux/config

# Verify
getenforce  # Should return: Enforcing
```

Part IV
Authentication Setup

9 User Authentication with Lucia

9.1 Database Schema for Auth

If using Prisma, add auth tables to your schema:

```
// prisma/schema.prisma

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model User {
  id             String    @id @default(uuid())
  email          String    @unique
  emailVerified  DateTime?
  passwordHash   String?
  name           String?
  avatarUrl      String?

  // Subscription
  subscription   String    @default("free")
```

```
22
23    // Timestamps
24    createdAt      DateTime  @default(now())
25    updatedAt      DateTime  @updatedAt
26
27    // Relations
28    sessions       Session[]
29    oauthAccounts  OAuthAccount[]
30    trunks         Trunk[]
31  }
32
33  model Session {
34    id        String   @id @default(uuid())
35    userId    String
36    expiresAt DateTime
37
38    user      User     @relation(fields: [userId], references: [id],
          onDelete: Cascade)
39
40    @@index([userId])
41  }
42
43  model OAuthAccount {
44    id             String  @id @default(uuid())
45    userId         String
46    provider       String
47    providerUserId String
48
49    user           User    @relation(fields: [userId], references: [id
          ], onDelete: Cascade)
50
51    @@unique([provider, providerUserId])
52    @@index([userId])
53  }
54
55  model Trunk {
56    id        String   @id @default(uuid())
57    name      String
58    userId    String
59    settings  Json     @default("{}")
60
61    createdAt DateTime @default(now())
62    updatedAt DateTime @updatedAt
63
64    user      User     @relation(fields: [userId], references: [id],
          onDelete: Cascade)
65    trees     Tree[]
66
67    @@index([userId])
68  }
69
70  model Tree {
71    id            String   @id @default(uuid())
72    name          String
73    trunkId       String
74    path          String?
75    isCloudSync   Boolean  @default(false)
76
```

```
  createdAt      DateTime  @default(now())
  updatedAt      DateTime  @updatedAt

  trunk          Trunk     @relation(fields: [trunkId], references: [id
     ], onDelete: Cascade)

  @@index([trunkId])
}
```

9.2 Run Migration

```
# Create and apply migration
pnpm prisma migrate dev --name init_auth

# On production server
pnpm db:migrate
```

9.3 Test User Registration

Create a test script or use curl to test registration:

```
# Test user registration endpoint
curl -X POST https://arborist.local/auth/register \
  -H "Content-Type: application/json" \
  -d '{
    "email": "test@example.com",
    "password": "SecurePassword123!",
    "name": "Test User"
  }' \
  -k -v

# Test login
curl -X POST https://arborist.local/auth/login \
  -H "Content-Type: application/json" \
  -d '{
    "email": "test@example.com",
    "password": "SecurePassword123!"
  }' \
  -k -c cookies.txt -v

# Test authenticated endpoint
curl https://arborist.local/api/user/me \
  -k -b cookies.txt
```

Part V
Production Simulation

10 Simulating Production Environment

10.1 Environment Parity Checklist

Aspect	Status	Notes
HTTPS/TLS		SSL certificate configured
Reverse Proxy		Nginx handling traffic
Process Manager		PM2 with clustering
Database		PostgreSQL with proper user/perms
Session Store		Redis for sessions
Environment Variables		Secrets in .env, not in code
Logging		Centralized log files
Firewall		Only necessary ports open
SELinux		Enforcing with proper policies
Backups		Database backup script
Monitoring		PM2 monitoring or external

Table 2: Production Parity Checklist

10.2 Load Testing

Install and use `k6` for load testing:

```
# Install k6 on your dev machine
# For Rocky Linux:
sudo dnf install https://dl.k6.io/rpm/repo.rpm
sudo dnf install k6

# Or download binary
wget https://github.com/grafana/k6/releases/download/v0.47.0/k6-v0
    .47.0-linux-amd64.tar.gz
tar -xzf k6-v0.47.0-linux-amd64.tar.gz
sudo mv k6-v0.47.0-linux-amd64/k6 /usr/local/bin/
```

Create a load test script:

```
// load-test.js
import http from 'k6/http';
import { check, sleep } from 'k6';

export const options = {
  stages: [
    { duration: '30s', target: 10 },    // Ramp up to 10 users
    { duration: '1m', target: 10 },     // Stay at 10 users
    { duration: '30s', target: 50 },    // Ramp up to 50 users
    { duration: '1m', target: 50 },     // Stay at 50 users
    { duration: '30s', target: 0 },     // Ramp down
  ],
  thresholds: {
    http_req_duration: ['p(95)<500'], // 95% of requests under 500ms
    http_req_failed: ['rate<0.01'],   // Less than 1% failure rate
  },
};

const BASE_URL = 'https://arborist.local';

export default function () {
  // Test homepage
  const homeRes = http.get(`${BASE_URL}/`, {
    insecureSkipTLSVerify: true,
```

```
  });
  check(homeRes, {
    'homepage status is 200': (r) => r.status === 200,
  });

  // Test API health
  const healthRes = http.get(`${BASE_URL}/health`, {
    insecureSkipTLSVerify: true,
  });
  check(healthRes, {
    'health check ok': (r) => r.status === 200,
  });

  sleep(1);
}
```

```
# Run load test
k6 run load-test.js

# View results
# k6 will output metrics including:
# - Request rate
# - Response times (p50, p90, p95, p99)
# - Error rate
```

10.3 Database Backup Script

```
#!/bin/bash
# /usr/local/bin/backup-arborist-db.sh

# Configuration
DB_NAME="arborist_db"
DB_USER="arborist"
BACKUP_DIR="/var/backups/arborist"
RETENTION_DAYS=7

# Create backup directory
mkdir -p "$BACKUP_DIR"

# Generate backup filename with timestamp
BACKUP_FILE="$BACKUP_DIR/${DB_NAME}_$(date +%Y%m%d_%H%M%S).sql.gz"

# Create backup
PGPASSWORD="your-db-password" pg_dump -U "$DB_USER" -h localhost "
    $DB_NAME" | gzip > "$BACKUP_FILE"

# Check if backup was successful
if [ $? -eq 0 ]; then
    echo "Backup successful: $BACKUP_FILE"

    # Remove old backups
    find "$BACKUP_DIR" -name "*.sql.gz" -mtime +$RETENTION_DAYS -
        delete
    echo "Old backups cleaned up"
else
```

```
    echo "Backup failed!"
    exit 1
fi
```

```
# Make executable and schedule
sudo chmod +x /usr/local/bin/backup-arborist-db.sh

# Add to crontab for daily backups at 2 AM
sudo crontab -e
# Add line:
0 2 * * * /usr/local/bin/backup-arborist-db.sh >> /var/log/arborist/backup.log 2>&1
```

10.4 Monitoring with PM2

```
# View real-time monitoring
pm2 monit

# View detailed process info
pm2 show arborist-api

# View logs
pm2 logs arborist-api --lines 100

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

11 CI/CD Simulation

11.1 Deployment Script

Create an automated deployment script:

```
#!/bin/bash
# deploy.sh - Run on server or via SSH

set -e  # Exit on error

APP_DIR="/var/www/arborist"
BACKUP_DIR="/var/backups/arborist/releases"
TIMESTAMP=$(date +%Y%m%d_%H%M%S)

echo "=== Arborist Deployment Started ==="
echo "Timestamp: $TIMESTAMP"

# 1. Create backup of current release
echo "Creating backup..."
mkdir -p "$BACKUP_DIR"
if [ -d "$APP_DIR/dist" ]; then
```

```
    tar -czf "$BACKUP_DIR/release_$TIMESTAMP.tar.gz" -C "$APP_DIR"
        dist/
fi

# 2. Pull latest code (if using Git)
echo "Pulling latest code..."
cd "$APP_DIR"
git fetch origin
git reset --hard origin/main

# 3. Install dependencies
echo "Installing dependencies..."
pnpm install --frozen-lockfile --prod

# 4. Generate Prisma client
echo "Generating Prisma client..."
pnpm db:generate

# 5. Run database migrations
echo "Running database migrations..."
pnpm db:migrate

# 6. Build application
echo "Building application..."
pnpm build:all

# 7. Restart application
echo "Restarting application..."
pm2 reload ecosystem.config.js --env production

# 8. Wait and verify
echo "Waiting for application to start..."
sleep 5

# 9. Health check
echo "Running health check..."
HTTP_STATUS=$(curl -s -o /dev/null -w "%{http_code}" -k https://
    localhost/health)
if [ "$HTTP_STATUS" -eq 200 ]; then
    echo "Health check passed!"
else
    echo "Health check failed! Rolling back..."
    # Rollback logic here
    exit 1
fi

# 10. Cleanup old backups (keep last 5)
echo "Cleaning up old backups..."
ls -t "$BACKUP_DIR"/release_*.tar.gz | tail -n +6 | xargs -r rm

echo "=== Deployment Complete ==="
```

```
# Make executable
chmod +x deploy.sh

# Run deployment
./deploy.sh
```

11.2 Rollback Script

```
#!/bin/bash
# rollback.sh - Restore previous release

set -e

APP_DIR="/var/www/arborist"
BACKUP_DIR="/var/backups/arborist/releases"

# Get most recent backup
LATEST_BACKUP=$(ls -t "$BACKUP_DIR"/release_*.tar.gz | head -1)

if [ -z "$LATEST_BACKUP" ]; then
    echo "No backup found to rollback to!"
    exit 1
fi

echo "Rolling back to: $LATEST_BACKUP"

# Extract backup
rm -rf "$APP_DIR/dist"
tar -xzf "$LATEST_BACKUP" -C "$APP_DIR"

# Restart application
pm2 reload ecosystem.config.js --env production

echo "Rollback complete!"
```

12 Multi-User Testing

12.1 Create Test Users

```
# Create multiple test users via API or database

# Via API (if registration endpoint exists)
for i in {1..5}; do
  curl -X POST https://arborist.local/auth/register \
    -H "Content-Type: application/json" \
    -d "{
      \"email\": \"testuser$i@example.com\",
      \"password\": \"TestPassword$i!\",
      \"name\": \"Test User $i\"
    }" -k
done

# Or directly in database
sudo -u postgres psql arborist_db << EOF
INSERT INTO "User" (id, email, name, "passwordHash", subscription)
VALUES
  (gen_random_uuid(), 'admin@example.com', 'Admin User', '
     hashed_password', 'pro'),
  (gen_random_uuid(), 'user1@example.com', 'User One', '
     hashed_password', 'free'),
```

```
  (gen_random_uuid(), 'user2@example.com', 'User Two', '
    hashed_password', 'free');
EOF
```

12.2 Concurrent User Testing

```
// concurrent-test.js - k6 script for multi-user testing
import http from 'k6/http';
import { check, sleep } from 'k6';
import { SharedArray } from 'k6/data';

const users = new SharedArray('users', function () {
  return [
    { email: 'testuser1@example.com', password: 'TestPassword1!' },
    { email: 'testuser2@example.com', password: 'TestPassword2!' },
    { email: 'testuser3@example.com', password: 'TestPassword3!' },
    { email: 'testuser4@example.com', password: 'TestPassword4!' },
    { email: 'testuser5@example.com', password: 'TestPassword5!' },
  ];
});

export const options = {
  vus: 5,
  duration: '2m',
};

export default function () {
  const user = users[__VU - 1]; // Each VU gets a different user

  // Login
  const loginRes = http.post(
    'https://arborist.local/auth/login',
    JSON.stringify(user),
    {
      headers: { 'Content-Type': 'application/json' },
      insecureSkipTLSVerify: true,
    }
  );

  check(loginRes, {
    'login successful': (r) => r.status === 200,
  });

  // Simulate user activity
  const jar = http.cookieJar();

  // Get user's trunks
  const trunksRes = http.get('https://arborist.local/api/trunks', {
    insecureSkipTLSVerify: true,
  });

  check(trunksRes, {
    'trunks retrieved': (r) => r.status === 200,
  });

  sleep(Math.random() * 3 + 1); // 1-4 second delay
```

```
}
```

Part VI
Troubleshooting

13 Common Issues

13.1 Application Won't Start

```
# Check PM2 logs
pm2 logs arborist-api --err --lines 100

# Common issues:
# 1. Port already in use
sudo lsof -i :3000
# Kill the process or change port

# 2. Database connection failed
psql -U arborist -d arborist_db -h localhost
# Verify credentials in .env

# 3. Missing environment variables
node -e "console.log(require('dotenv').config())"
```

13.2 Nginx 502 Bad Gateway

```
# Check if Node.js app is running
pm2 status
curl http://localhost:3000/health

# Check Nginx error logs
sudo tail -f /var/log/nginx/arborist_error.log

# Check SELinux denials
sudo ausearch -m avc -ts recent
# If SELinux is blocking:
sudo setsebool -P httpd_can_network_connect 1
```

13.3 SSL Certificate Issues

```
# Verify certificate
openssl x509 -in /etc/nginx/ssl/arborist.crt -text -noout

# Test SSL connection
openssl s_client -connect arborist.local:443 -servername arborist.
    local

# Check Nginx SSL configuration
sudo nginx -t
```

13.4 Database Connection Issues

```
# Check PostgreSQL is running
sudo systemctl status postgresql-16

# Check pg_hba.conf allows connections
sudo cat /var/lib/pgsql/16/data/pg_hba.conf

# Test connection
psql -U arborist -d arborist_db -h localhost -W

# Check PostgreSQL logs
sudo tail -f /var/lib/pgsql/16/data/log/postgresql-*.log
```

14 Quick Reference

14.1 Service Management

```
# Application
pm2 start/stop/restart arborist-api
pm2 logs arborist-api
pm2 monit

# Nginx
sudo systemctl start/stop/restart/reload nginx
sudo nginx -t

# PostgreSQL
sudo systemctl start/stop/restart postgresql-16

# Redis
sudo systemctl start/stop/restart redis

# Firewall
sudo firewall-cmd --list-all
sudo firewall-cmd --reload
```

14.2 Log Locations

Service	Log Location
Application	`/var/log/arborist/*.log`
PM2	`~/.pm2/logs/`
Nginx Access	`/var/log/nginx/arborist_access.log`
Nginx Error	`/var/log/nginx/arborist_error.log`
PostgreSQL	`/var/lib/pgsql/16/data/log/`
Redis	`/var/log/redis/redis.log`
System	`journalctl -xe`

Table 3: Log File Locations

14.3 Useful Commands

```
# Disk usage
df -h
du -sh /var/www/arborist/*

# Memory usage
free -h
pm2 monit

# Network connections
ss -tlnp
netstat -tlnp

# Process status
htop
ps aux | grep node

# SELinux status
getenforce
sestatus

# Firewall status
sudo firewall-cmd --state
sudo firewall-cmd --list-all
```

15 Summary

Deployment Complete Checklist

- Rocky Linux VM created on Proxmox
- Node.js 20 LTS installed
- PostgreSQL 16 configured with app database
- Redis installed for sessions
- Application built and deployed to `/var/www/arborist`
- PM2 managing Node.js process with clustering
- Nginx configured as reverse proxy with SSL
- Firewall configured (ports 22, 80, 443)
- SELinux policies applied
- Database migrations applied
- Test users created
- Health check passing
- Backup script scheduled
- Deployment script tested

Your staging environment is ready.

Test thoroughly. Break things. Fix them. Repeat.
