Blog-to-Podcast Deployment Manual

Table of Contents

Development Deployment

Quick Start

```
# Clone repository
git clone <repository-url>
cd blog-to-podcast

# Setup environment
python -m venv venv
venv\Scripts\activate  # Windows
source venv/bin/activate  # macOS/Linux

# Install dependencies
pip install -r requirements.txt

# Configure
echo "GOOGLE_API_KEY=your_key_here" > .env
echo "DEBUG=True" >> .env

# Initialize database
python manage.py migrate

# Run development server
python manage.py runserver
```

Development Environment Variables

```
DEBUG=True
SECRET_KEY=dev-secret-key-change-in-production
GOOGLE_API_KEY=your_google_api_key
ALLOWED_HOSTS=localhost,127.0.0.1
DATABASE_URL=sqlite:///db.sqlite3
```

Staging Deployment

Prerequisites

- Ubuntu 20.04+ or similar Linux distribution
- Python 3.11+
- Nginx
- PostgreSQL 13+
- 4GB RAM minimum

Step 1: System Setup

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

# Install dependencies
sudo apt install python3.11 python3.11-venv python3-pip nginx postgresql postgresql-contrib -y

# Install system libraries
sudo apt install libpq-dev python3-dev -y
```

Step 2: Database Setup

```
# Create database and user
sudo -u postgres psql
CREATE DATABASE blog_to_podcast_staging;
CREATE USER podcast_user WITH PASSWORD 'secure_password';
GRANT ALL PRIVILEGES ON DATABASE blog_to_podcast_staging TO podcast_user;
\q
```

Step 3: Application Setup

```
# Create app directory
sudo mkdir -p /var/www/blog-to-podcast
sudo chown $USER:$USER /var/www/blog-to-podcast
cd /var/www/blog-to-podcast

# Clone and setup
git clone <repository-url> .
python3.11 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
pip install gunicorn psycopg2-binary
```

Step 4: Configuration

```
# Create .env file
cat > .env << EOF
DEBUG=False
SECRET_KEY=$(python -c 'from django.core.management.utils import get_random_secret_key; print(ge
GOOGLE_API_KEY=your_google_api_key
ALLOWED_HOSTS=staging.yourdomain.com
DATABASE_URL=postgresql://podcast_user:secure_password@localhost/blog_to_podcast_staging
MEDIA_ROOT=/var/www/blog-to-podcast/media
STATIC_ROOT=/var/www/blog-to-podcast/static
EOF

# Run migrations
python manage.py migrate
python manage.py collectstatic --noinput
```

Step 5: Gunicorn Setup

```
# Create systemd service
sudo nano /etc/systemd/system/blog-to-podcast.service
```

```
[Unit]
Description=Blog-to-Podcast Gunicorn Service
After=network.target

[Service]
User=www-data
Group=www-data
WorkingDirectory=/var/www/blog-to-podcast
Environment="PATH=/var/www/blog-to-podcast/venv/bin"
```

```
ExecStart=/var/www/blog-to-podcast/venv/bin/gunicorn \
   --workers 3 \
   --bind unix:/var/www/blog-to-podcast/gunicorn.sock \
   blog_to_podcast.wsgi:application

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

```
# Start service
sudo systemctl start blog-to-podcast
sudo systemctl enable blog-to-podcast
```

Step 6: Nginx Configuration

```
sudo nano /etc/nginx/sites-available/blog-to-podcast
```

```
server {
   listen 80;
   server_name staging.yourdomain.com;

   client_max_body_size 100M;

   location /static/ {
       alias /var/www/blog-to-podcast/static/;
   }

   location /media/ {
       alias /var/www/blog-to-podcast/media/;
   }

   location / {
       proxy_pass http://unix:/var/www/blog-to-podcast/gunicorn.sock;
       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;
   }
}
```

```
# Enable site
sudo ln -s /etc/nginx/sites-available/blog-to-podcast /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl restart nginx
```

Production Deployment

Additional Requirements

- SSL Certificate (Let's Encrypt)
- Redis for caching
- Celery for background tasks
- Monitoring (Sentry, New Relic)

SSL Setup

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

# Get certificate
sudo certbot --nginx -d yourdomain.com -d www.yourdomain.com
```

Redis & Celery Setup

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

# Update requirements.txt
echo "celery[redis]" >> requirements.txt
pip install -r requirements.txt
```

Celery Service

```
sudo nano /etc/systemd/system/celery.service
```

```
[Unit]
Description=Celery Service
After=network.target

[Service]
Type=forking
User=www-data
```

```
Group=www-data
WorkingDirectory=/var/www/blog-to-podcast
Environment="PATH=/var/www/blog-to-podcast/venv/bin"
ExecStart=/var/www/blog-to-podcast/venv/bin/celery -A blog_to_podcast worker --loglevel=info

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

Production Environment Variables

```
DEBUG=False
SECRET_KEY=<strong-random-key>
GOOGLE_API_KEY=<your-key>
ALLOWED_HOSTS=yourdomain.com,www.yourdomain.com
DATABASE_URL=postgresql://user:pass@localhost/dbname
REDIS_URL=redis://localhost:6379/0
CELERY_BROKER_URL=redis://localhost:6379/0
SENTRY_DSN=<your-sentry-dsn>
```

Configuration

Django Settings (Production)

```
# settings.py additions
import sentry_sdk
from sentry_sdk.integrations.django import DjangoIntegration

if not DEBUG:
   sentry_sdk.init(
       dsn=os.getenv('SENTRY_DSN'),
       integrations=[DjangoIntegration()],
       traces_sample_rate=0.1,
   )

# Security settings
SECURE_SSL_REDIRECT = True
SESSION_COOKIE_SECURE = True
CSRF_COOKIE_SECURE = True
SECURE_BROWSER_XSS_FILTER = True
SECURE_CONTENT_TYPE_NOSNIFF = True
X_FRAME_OPTIONS = 'DENY'
```

Rate Limiting

```
# Install django-ratelimit
pip install django-ratelimit

# In views.py
from django_ratelimit.decorators import ratelimit

@ratelimit(key='ip', rate='10/h', method='POST')
def convert_view(request):
   # ... existing code
```

Monitoring & Maintenance

Log Monitoring

```
# Application logs
sudo journalctl -u blog-to-podcast -f

# Nginx logs
sudo tail -f /var/log/nginx/access.log
sudo tail -f /var/log/nginx/error.log
```

Health Checks

```
# Create health check endpoint
# In urls.py
path('health/', health_check_view)

# Monitor with
curl http://yourdomain.com/health/
```

Performance Monitoring

- **Sentry**: Error tracking and performance
- **New Relic**: Application performance monitoring
- **Prometheus + Grafana**: Metrics and dashboards

Database Maintenance

```
# Vacuum database (weekly)
sudo -u postgres psql blog_to_podcast_prod -c "VACUUM ANALYZE;"

# Check database size
sudo -u postgres psql -c "SELECT pg_size_pretty(pg_database_size('blog_to_podcast_prod'));"
```

Backup & Recovery

Database Backup

```
# Create backup script
cat > /usr/local/bin/backup-db.sh << 'EOF'
#!/bin/bash
BACKUP_DIR="/var/backups/blog-to-podcast"
DATE=$(date +%Y%m%d_%H%M%S)
mkdir -p $BACKUP_DIR
sudo -u postgres pg_dump blog_to_podcast_prod | gzip > $BACKUP_DIR/db_$DATE.sql.gz
# Keep only last 7 days
find $BACKUP_DIR -name "db_*.sql.gz" -mtime +7 -delete
EOF

chmod +x /usr/local/bin/backup-db.sh

# Add to crontab (daily at 2 AM)
(crontab -l 2>/dev/null; echo "0 2 * * * /usr/local/bin/backup-db.sh") | crontab -
```

Media Files Backup

```
# Backup media files
rsync -avz /var/www/blog-to-podcast/media/ /var/backups/blog-to-podcast/media/
```

Recovery

```
# Restore database
gunzip -c /var/backups/blog-to-podcast/db_20251201_020000.sql.gz | sudo -u postgres psql blog_t

# Restore media
rsync -avz /var/backups/blog-to-podcast/media/ /var/www/blog-to-podcast/media/
```

Scaling

Horizontal Scaling

- Use load balancer (Nginx, HAProxy)
- Multiple Gunicorn instances
- Shared media storage (S3, NFS)
- Centralized database (RDS, managed PostgreSQL)

Vertical Scaling

- Increase worker count: `--workers $(nproc --all)`
- Optimize database queries
- Add database indexes
- Implement caching (Redis)

Troubleshooting

Common Issues

502 Bad Gateway

```
# Check Gunicorn status
sudo systemctl status blog-to-podcast
# Check socket permissions
ls -l /var/www/blog-to-podcast/gunicorn.sock
```

Static files not loading

```
# Recollect static files
python manage.py collectstatic --noinput --clear
```

Database connection errors

```
# Check PostgreSQL status
sudo systemctl status postgresql
# Verify credentials in .env
```

Version: 1.0

Last Updated: December 2025