🚀 YouTube Channel Optimizer - Complete Deployment Guide

📋 Table of Contents

🎯 Quick Start

Prerequisites

- Python 3.11+
- PostgreSQL 14+
- Redis 7+
- Docker & Docker Compose (optional)

1. Clone and Setup

```bash
```

```
# Clone repository
git clone <your-repo-url>
cd youtube-optimizer

# Create virtual environment
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt

# Setup environment variables
cp .env.example .env
# Edit .env with your configuration
```

2. Generate Secret Key

```bash
python -c "import secrets; print(secrets.token_urlsafe(32))"
# Copy output to SECRET_KEY in .env
```

3. Initialize Database

```bash
# Create database
psql -U postgres -c "CREATE DATABASE youtube_optimizer;"

# Run migrations (if you have alembic setup)
alembic upgrade head

# Or run init script
psql -U postgres -d youtube_optimizer -f scripts/init-db.sql
```

4. Run Application

```bash
```

```
# Development mode
uvicorn main:app --reload --log-level debug

# Production mode
gunicorn main:app --workers 4 --worker-class uvicorn.workers.UvicornWorker
```

5. Access API

- API Documentation: http://localhost:8000/docs
- Alternative Docs: http://localhost:8000/redoc
- Health Check: http://localhost:8000/health

📁 Project Structure

```
youtube-optimizer/
├── main.py                 # Main FastAPI application
├── config.py               # Configuration management
├── requirements.txt        # Python dependencies
├── Dockerfile              # Docker build configuration
├── docker-compose.yml      # Docker orchestration
├── .env.example            # Environment variables template
├── .env                    # Your environment variables (gitignored)
│
├── routes/                 # API route handlers
│   ├── __init__.py
│   ├── channel_routes.py   # Channel optimization endpoints
│   ├── video_routes.py     # Video management endpoints
│   ├── scheduler_routes.py # Scheduling endpoints
│   └── health_routes.py    # Health check endpoints
│
├── services/               # Business logic layer
│   ├── __init__.py
│   ├── channel.py          # Channel service
│   ├── video.py            # Video service
│   ├── optimizer.py        # Optimization engine
│   ├── scheduler.py        # Task scheduler
│   └── youtube_service.py  # YouTube API client
│
├── utils/                  # Utility modules
│   ├── __init__.py
│   ├── db.py              # Database utilities
│   ├── auth.py            # Authentication & authorization
│   ├── rate_limiter.py    # Rate limiting
│   ├── metrics.py         # Metrics collection
│   ├── exceptions.py      # Custom exceptions
│   └── cache.py           # Caching utilities
│
├── models/                # Database models (if using ORM)
│   ├── __init__.py
│   ├── channel.py
│   ├── video.py
│   └── user.py
│
├── schemas/               # Pydantic schemas
│   ├── __init__.py
│   ├── channel.py
│   ├── video.py
│   └── user.py
│
├── tests/                 # Test suite
│   ├── __init__.py
│   ├── test_channels.py
│   ├── test_videos.py
│   └── conftest.py
│
├── scripts/               # Utility scripts
│   ├── init-db.sql        # Database initialization
│   ├── seed-data.sql      # Sample data
│   └── backup-db.sh       # Backup script
│
├── monitoring/            # Monitoring configuration
│   ├── prometheus.yml     # Prometheus config
│   └── grafana/
│       ├── dashboards/
│       └── datasources/
│
└── logs/                  # Application logs
    └── .gitkeep
```

🔧 Environment Setup

Required Environment Variables

```bash
# Application
SECRET_KEY=<generate-with-command-above>
ENVIRONMENT=production
DEBUG=false

# Database
DATABASE_HOST=localhost
DATABASE_PORT=5432
DATABASE_NAME=youtube_optimizer
DATABASE_USER=postgres
DATABASE_PASSWORD=<your-db-password>

# Redis
REDIS_HOST=localhost
REDIS_PORT=6379
REDIS_PASSWORD=<optional-redis-password>

# YouTube API
YOUTUBE_API_KEY=<your-youtube-api-key>
YOUTUBE_CLIENT_ID=<your-oauth-client-id>
YOUTUBE_CLIENT_SECRET=<your-oauth-client-secret>

# Scheduler
CLOUD_SCHEDULER_SECRET=<generate-random-string>

# Sentry (Optional)
SENTRY_DSN=<your-sentry-dsn>
```

Getting YouTube API Credentials

1. Go to Google Cloud Console
2. Create a new project or select existing
3. Enable YouTube Data API v3
4. Create credentials (API key + OAuth 2.0)
5. Configure OAuth consent screen
6. Add authorized redirect URIs
7. Copy credentials to .env

💻 Local Development

Setup Development Environment

```bash
# Install development dependencies
pip install -r requirements.txt
pip install pytest pytest-asyncio black flake8 mypy

# Setup pre-commit hooks (optional)
pip install pre-commit
pre-commit install

# Run code formatting
black .
isort .

# Run linting
flake8 .
mypy .

# Run tests
pytest tests/ -v --cov=.
```

Running Services Locally

```bash
```

```
# Terminal 1: Start PostgreSQL
docker run -d --name postgres \
  -e POSTGRES_PASSWORD=postgres \
  -p 5432:5432 \
  postgres:16

# Terminal 2: Start Redis
docker run -d --name redis \
  -p 6379:6379 \
  redis:7-alpine

# Terminal 3: Start API
uvicorn main:app --reload --log-level debug

# Terminal 4: Start Celery Worker (if needed)
celery -A services.celery_app worker --loglevel=info

# Terminal 5: Start Celery Beat (if needed)
celery -A services.celery_app beat --loglevel=info
```

🐳 Docker Deployment

Basic Docker Deployment

```bash
# Build and start all services
docker-compose up -d

# View logs
docker-compose logs -f api

# Stop services
docker-compose down

# Rebuild after code changes
docker-compose up -d --build
```

With Monitoring

```bash
```

```
# Start with Prometheus & Grafana
docker-compose --profile with-monitoring up -d

# Access monitoring
# Prometheus: http://localhost:9090
# Grafana: http://localhost:3000 (admin/admin)
```

With Nginx

```bash
# Start with Nginx reverse proxy
docker-compose --profile with-nginx up -d
```

Production Build

```bash
# Build production image
docker build -t youtube-optimizer:latest --target production .

# Run production container
docker run -d \
  --name youtube-optimizer \
  -p 8000:8000 \
  --env-file .env \
  youtube-optimizer:latest
```

🌐 Production Deployment

Option 1: Cloud Run (GCP)

```bash
```

```
# Build and push to Google Container Registry
gcloud builds submit --tag gcr.io/PROJECT_ID/youtube-optimizer

# Deploy to Cloud Run
gcloud run deploy youtube-optimizer \
  --image gcr.io/PROJECT_ID/youtube-optimizer \
  --platform managed \
  --region us-central1 \
  --allow-unauthenticated \
  --set-env-vars="ENVIRONMENT=production" \
  --set-secrets="SECRET_KEY=secret-key:latest,DATABASE_PASSWORD=db-password:latest"
```

Option 2: AWS ECS

```bash
# Build and push to ECR
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin ACCOUNT_ID.dkr.ecr.us-eas

docker build -t youtube-optimizer .
docker tag youtube-optimizer:latest ACCOUNT_ID.dkr.ecr.us-east-1.amazonaws.com/youtube-optimizer:latest
docker push ACCOUNT_ID.dkr.ecr.us-east-1.amazonaws.com/youtube-optimizer:latest

# Deploy with ECS
# (Use AWS Console or Terraform)
```

Option 3: Kubernetes

```bash
# Build image
docker build -t youtube-optimizer:v1.0.0 .

# Push to registry
docker tag youtube-optimizer:v1.0.0 your-registry/youtube-optimizer:v1.0.0
docker push your-registry/youtube-optimizer:v1.0.0

# Apply Kubernetes manifests
kubectl apply -f k8s/
```

Option 4: Traditional VPS

```bash
```

```
# SSH to server
ssh user@your-server.com

# Clone repository
git clone <your-repo> /opt/youtube-optimizer
cd /opt/youtube-optimizer

# Setup environment
cp .env.example .env
nano .env  # Configure

# Install dependencies
pip install -r requirements.txt

# Setup systemd service
sudo cp scripts/youtube-optimizer.service /etc/systemd/system/
sudo systemctl enable youtube-optimizer
sudo systemctl start youtube-optimizer
```

🗄️ Database Setup

Create Tables

```
sql
```

```sql
-- Users table
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT NOW(),
    last_login TIMESTAMP
);

-- YouTube channels table
CREATE TABLE youtube_channels (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
    channel_id VARCHAR(255) UNIQUE NOT NULL,
    channel_name VARCHAR(255),
    description TEXT,
    keywords TEXT,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Channel optimizations table
CREATE TABLE channel_optimizations (
    id SERIAL PRIMARY KEY,
    channel_id INTEGER REFERENCES youtube_channels(id) ON DELETE CASCADE,
    original_description TEXT,
    optimized_description TEXT,
    original_keywords TEXT,
    optimized_keywords TEXT,
    optimization_notes TEXT,
    status VARCHAR(50) DEFAULT 'pending',
    progress INTEGER DEFAULT 0,
    is_applied BOOLEAN DEFAULT FALSE,
    applied_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- YouTube videos table
CREATE TABLE youtube_videos (
    id SERIAL PRIMARY KEY,
    channel_id INTEGER REFERENCES youtube_channels(id) ON DELETE CASCADE,
    video_id VARCHAR(255) UNIQUE NOT NULL,
    title VARCHAR(255),
    description TEXT,
    view_count INTEGER DEFAULT 0,
    like_count INTEGER DEFAULT 0,
    comment_count INTEGER DEFAULT 0,
    published_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Video optimizations table
CREATE TABLE video_optimizations (
    id SERIAL PRIMARY KEY,
    video_id INTEGER REFERENCES youtube_videos(id) ON DELETE CASCADE,
    original_title VARCHAR(255),
    optimized_title VARCHAR(255),
    original_description TEXT,
    optimized_description TEXT,
    original_tags TEXT,
    optimized_tags TEXT,
    status VARCHAR(50) DEFAULT 'pending',
    is_applied BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT NOW()
);

-- Scheduler tables
CREATE TABLE channel_optimization_schedules (
    id SERIAL PRIMARY KEY,
    channel_id INTEGER REFERENCES youtube_channels(id) ON DELETE CASCADE,
    is_active BOOLEAN DEFAULT TRUE,
    auto_apply BOOLEAN DEFAULT FALSE,
    last_run TIMESTAMP,
    next_run TIMESTAMP,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE scheduler_run_history (
    id SERIAL PRIMARY KEY,
    schedule_id INTEGER REFERENCES channel_optimization_schedules(id),
    start_time TIMESTAMP,
    end_time TIMESTAMP,
    status VARCHAR(50),
    optimization_id INTEGER,
    applied BOOLEAN,
    error_message TEXT
);

-- Permissions & roles
CREATE TABLE roles (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) UNIQUE NOT NULL
);

CREATE TABLE permissions (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) UNIQUE NOT NULL
);

CREATE TABLE user_roles (
    user_id INTEGER REFERENCES users(id),
    role_id INTEGER REFERENCES roles(id),
    PRIMARY KEY (user_id, role_id)
);

CREATE TABLE role_permissions (
    role_id INTEGER REFERENCES roles(id),
    permission_id INTEGER REFERENCES permissions(id),
    PRIMARY KEY (role_id, permission_id)
);

-- Indexes for performance
CREATE INDEX idx_channels_user_id ON youtube_channels(user_id);
CREATE INDEX idx_videos_channel_id ON youtube_videos(channel_id);
CREATE INDEX idx_optimizations_channel_id ON channel_optimizations(channel_id);
CREATE INDEX idx_optimizations_status ON channel_optimizations(status);
CREATE INDEX idx_video_optimizations_video_id ON video_optimizations(video_id);
```

📊 Monitoring & Observability

Prometheus Metrics

The application exposes metrics at `/metrics`:

```bash
```

```
# Query metrics
curl http://localhost:8000/metrics

# Example metrics:
# - http_requests_total
# - http_request_duration_seconds
# - optimization_events_total
# - app_info
```

Grafana Dashboards

1. Access Grafana: http://localhost:3000
2. Login: admin/admin
3. Import dashboard JSON from `monitoring/grafana/dashboards/`

Logging

Logs are structured JSON (production) or colored text (development):

```bash
# View logs
docker-compose logs -f api

# View specific service
docker-compose logs -f celery_worker

# Tail logs file
tail -f logs/app.log
```

Health Checks

```bash
```

```
# Basic health
curl http://localhost:8000/health

# Response:
{
  "status": "healthy",
  "timestamp": 1234567890,
  "version": "1.0.0",
  "checks": {
    "database": "healthy",
    "redis": "healthy"
  }
}
```

🔍 Troubleshooting

Common Issues

1. Database connection fails

```bash
# Check PostgreSQL is running
docker ps | grep postgres

# Test connection
psql -h localhost -U postgres -d youtube_optimizer

# Check environment variables
echo $DATABASE_PASSWORD
```

2. Redis connection fails

```bash
```

```
# Check Redis is running
docker ps | grep redis

# Test connection
redis-cli ping

# With password
redis-cli -a your-password ping
```

3. Module import errors

```
bash

# Ensure virtual environment is activated
source venv/bin/activate

# Reinstall dependencies
pip install -r requirements.txt --force-reinstall
```

4. Port already in use

```
bash

# Find process using port 8000
lsof -i :8000

# Kill process
kill -9 <PID>
```

5. Permission denied errors

```
bash

# Fix file permissions
chmod +x scripts/*.sh

# Fix log directory
sudo chown -R $USER:$USER logs/
```

Debug Mode

```
bash
```

```
# Enable debug logging
export LOG_LEVEL=DEBUG
export DEBUG=true

# Run with verbose output
uvicorn main:app --reload --log-level debug
```

Database Migrations

```bash
# Generate migration
alembic revision --autogenerate -m "description"

# Apply migrations
alembic upgrade head

# Rollback
alembic downgrade -1
```

📚 Additional Resources

- FastAPI Documentation
- YouTube Data API
- PostgreSQL Docs
- Redis Documentation
- Docker Documentation

🆘 Support

For issues and questions:

1. Check the Troubleshooting section
2. Review application logs
3. Check `/health` endpoint
4. Create an issue on GitHub

Happy Deploying! 🚀