📊 После развертывания - Руководство по эксплуатации

🎉 Поздравляем с успешным развертыванием!

Ваш бот WOWsilizing теперь работает 24/7 в облаке Railway. Это руководство поможет вам:
- ✅ Проверить, что всё работает правильно
- 📊 Мониторить работу бота
- 🔄 Обновлять бота
- 🛠️ Решать проблемы
- 💰 Оптимизировать расходы

📋 Содержание

✅ Первичная проверка

День 1: Сразу после развертывания

1. Проверка статуса в Railway

1. Откройте Railway Dashboard: https://railway.app/dashboard
2. Выберите ваш проект
3. Убедитесь:
   - ✅ Статус: **“Active”** (зеленый)
   - ✅ Deployment: последний деплой успешен
   - ✅ Uptime: бот работает

2. Проверка логов

1. Перейдите в **Deployments → View Logs**
2. Должны видеть:
   `✅ INFO: Bot started successfully`

   `✅ INFO: Started polling`
   `✅ INFO: Database initialized`

3. НЕ должно быть:
   `❌ ERROR: ...`
   `❌ CRITICAL: ...`
   `❌ Failed to ...`

3. Тест базовых функций

Тест 1: Старт бота
- [ ] Отправить `/start` в боте
- [ ] Получить приветственное сообщение
- [ ] Увидеть меню команд

Тест 2: Обработка видео
- [ ] Отправить короткое видео (10-30 секунд)
- [ ] Бот принял видео
- [ ] Показал кнопки действий
- [ ] Выбрать “Извлечь аудио”
- [ ] Получить аудио файл обратно

Тест 3: Пакетная нарезка
- [ ] Отправить видео
- [ ] Выбрать “Пакетная нарезка”
- [ ] Отправить таймкоды: `00:05-00:10, 00:15-00:20`
- [ ] Получить 2 нарезанных фрагмента

Тест 4: Премиум функции (если настроены)
- [ ] Войти как @WowFUX
- [ ] Отправить видео
- [ ] Попробовать AI субтитры
- [ ] Проверить, что API ключи работают

4. Проверка метрик

1. В Railway перейдите в **Metrics**
2. Проверьте:
   - **CPU**: должно быть 5-30% в покое
   - **Memory**: 200-500 MB нормально
   - **Disk**: зависит от кеша
   - **Network**: активность при обработке

Если метрики в норме - всё отлично! ✅

Неделя 1: Регулярные проверки

Чеклист на каждый день:

Понедельник:
- [ ] Проверить статус бота (Active/Failed)
- [ ] Просмотреть логи за последние 24 часа
- [ ] Проверить Usage/Billing в Railway

Среда:
- [ ] Провести функциональный тест
- [ ] Проверить метрики (CPU, Memory, Disk)
- [ ] Очистить старые логи (если нужно)

Пятница:
- [ ] Проверить расходы на AI API
- [ ] Оценить потребление кредитов Railway
- [ ] Запланировать обновления (если есть)

Выходные:
- [ ] Провести полный стресс-тест
- [ ] Обработать несколько больших видео
- [ ] Проверить все премиум функции

📊 Ежедневный мониторинг

Автоматизация мониторинга

Настройка уведомлений в Railway

1. Откройте **Settings → Notifications**
2. Включите уведомления для:
   - ✅ Deployment failed
   - ✅ Service crashed
   - ✅ Resource limits exceeded
   - ✅ Billing alerts
3. Добавьте email или webhook

Мониторинг API расходов

OpenAI:
1. Зайдите на https://platform.openai.com/usage
2. Проверьте Usage за текущий месяц
3. Настройте Soft/Hard limits
4. Включите email уведомления

Google:
1. Зайдите на https://console.cloud.google.com/apis/dashboard
2. Проверьте Quotas
3. Настройте Alerts

ElevenLabs:
1. Зайдите на https://elevenlabs.io/
2. Проверьте Character usage
3. Отслеживайте месячный лимит

Что проверять ежедневно:

Railway Dashboard:

- ✅ **Status**: должен быть зеленый “Active”
- ✅ **CPU Usage**: среднее 5-30%, пики до 100% OK

- ✅ **Memory**: 200-500 MB, не должно расти постоянно
- ✅ **Disk**: проверить, что кеш не раздувается
- ✅ **Network**: активность соответствует использованию
- ✅ **Cost**: отслеживать расход кредитов

Логи Railway:

- ✅ Нет ERROR или CRITICAL сообщений
- ✅ Бот успешно обрабатывает запросы
- ✅ Нет повторяющихся предупреждений
- ✅ API запросы успешны

Telegram Bot:

- ✅ Отвечает на команды быстро (< 2 секунды)
- ✅ Обрабатывает файлы корректно
- ✅ Нет жалоб от пользователей

🔄 Обновление бота

Метод 1: Автоматическое обновление (через GitHub)

Если вы развернули через GitHub:

Шаг 1: Внесите изменения локально

```
cd /home/ubuntu/wowsilizing_bot

# Внесите изменения в код
nano bot.py

# Или добавьте новый файл
touch new_feature.py
```

Шаг 2: Закоммитьте и запушьте

```
git add .
git commit -m "Добавлена новая функция / Исправлена ошибка"
git push origin main
```

Шаг 3: Railway автоматически обновится

- Railway обнаружит изменения в GitHub
- Автоматически запустит новую сборку
- Через 3-5 минут новая версия будет развернута
- Старая версия автоматически остановится

Шаг 4: Проверка после обновления

1. Откройте Railway Logs
2. Убедитесь, что сборка успешна
3. Проверьте, что бот запустился
4. Протестируйте новую функциональность

Метод 2: Ручное обновление (через Railway UI)

Шаг 1: Откройте Railway Dashboard

1. Перейдите в ваш проект
2. Нажмите на сервис

Шаг 2: Redeploy

1. Нажмите **“•••”** (три точки) → **“Redeploy”**
2. Или загрузите новый ZIP архив
3. Дождитесь пересборки

Шаг 3: Проверка

- Проверьте логи
- Протестируйте бота

Откат к предыдущей версии

Если что-то пошло не так:

Быстрый откат:

1. Откройте **Deployments**
2. Найдите предыдущее успешное развертывание
3. Нажмите **“•••”** → **“Rollback”**
4. Подтвердите
5. Через 1-2 минуты старая версия будет восстановлена

Откат через GitHub:

```
# Посмотреть историю коммитов
git log --oneline

# Откатиться на конкретный коммит
git revert <commit-hash>

# Или полный сброс (осторожно!)
git reset --hard <commit-hash>
git push origin main --force
```

Best Practices для обновлений

✅ ДЕЛАЙТЕ:

1. **Тестируйте локально** перед деплоем
2. **Коммитьте часто** с понятными сообщениями
3. **Делайте бэкапы** перед большими изменениями
4. **Используйте ветки** для новых функций
5. **Пишите changelog** для отслеживания изменений

❌ НЕ ДЕЛАЙТЕ:

1. Не пушьте неработающий код
2. Не обновляйте в часы пик
3. Не меняйте критические функции без тестирования
4. Не забывайте обновлять requirements.txt
5. Не игнорируйте предупреждения в логах

🔑 Управление API ключами

Добавление новых API ключей

Если вы изначально развернули без AI функций:

Шаг 1: Получите API ключ

- OpenAI: https://platform.openai.com/api-keys
- Google: https://makersuite.google.com/app/apikey
- ElevenLabs: https://elevenlabs.io/

Шаг 2: Добавьте в Railway

1. Откройте ваш проект в Railway
2. Перейдите в **Variables**
3. Нажмите **”+ New Variable”**
4. Введите:
   - Variable name: `OPENAI_API_KEY` (или другой)
   - Value: ваш ключ
5. Нажмите **“Add”**

Шаг 3: Бот перезапустится

- Railway автоматически перезапустит бота
- Через 30-60 секунд новый ключ будет активен

Шаг 4: Проверка

1. Откройте Telegram бота
2. Войдите как премиум пользователь (@WowFUX)
3. Попробуйте AI функцию
4. Убедитесь, что работает

Ротация API ключей

Рекомендуется менять ключи каждые 3-6 месяцев для безопасности.

Шаг 1: Создайте новый ключ

1. Зайдите в сервис (OpenAI/Google/ElevenLabs)
2. Создайте новый API ключ
3. Скопируйте его

Шаг 2: Обновите в Railway

1. Откройте **Variables**
2. Найдите старый ключ
3. Нажмите на него → **“Edit”**
4. Вставьте новый ключ
5. Сохраните

Шаг 3: Отзовите старый ключ

1. Вернитесь в сервис
2. Отзовите/удалите старый ключ
3. Готово!

Важно: Делайте это быстро, чтобы бот не потерял доступ.

Мониторинг использования API

OpenAI Usage Dashboard:

```
URL: https://platform.openai.com/usage

Что смотреть:
- Daily API calls
- Cost breakdown (Whisper, GPT-4, TTS)
- Rate limits
- Errors
```

Google Cloud Console:

```
URL: https://console.cloud.google.com/apis/dashboard

Что смотреть:
- API requests per day
- Quota usage
- Errors and latency
```

ElevenLabs Dashboard:

```
URL: https://elevenlabs.io/

Что смотреть:
- Characters used
- Remaining quota
- Billing cycle
```

💰 Оптимизация расходов

Railway

Текущие расходы:

- Бесплатно: $5 кредитов/месяц

- Обычно этого хватает для 500+ часов работы легкого бота

Как уменьшить расходы Railway:

1. Оптимизируйте использование ресурсов:

```
# В config.py уменьшите лимиты
MAX_FILE_SIZE_MB = 20  # было 50
MAX_QUEUE_SIZE = 5  # было 10
CACHE_MAX_SIZE_GB = 2  # было 5
```

2. Включите агрессивную очистку:

```
# Удаляйте временные файлы сразу после обработки
import os
import shutil

# После обработки
os.remove(temp_file)
shutil.rmtree(temp_dir)
```

3. Уменьшите качество обработки:

```
# В config.py
DEFAULT_CRF = 28  # было 23 (выше = меньше качество = быстрее)
DEFAULT_PRESET = "fast"  # было "medium"
```

4. Ограничьте одновременные обработки:

```
# Не обрабатывайте много видео одновременно
MAX_CONCURRENT_TASKS = 2
```

OpenAI

Текущие расходы (примерно):

- Whisper: $0.006/минуту
- GPT-4: $0.03/1K токенов
- TTS: $0.015/1K символов

Как уменьшить расходы OpenAI:

1. Используйте более дешевые модели:

```
# Вместо GPT-4
model = "gpt-3.5-turbo"  # в 10 раз дешевле

# Вместо tts-1-hd
model = "tts-1"  # в 2 раза дешевле
```

2. Кешируйте результаты:

```
# Не транскрибируйте одно и то же видео дважды
if video_hash in cache:
    return cache[video_hash]
```

3. Сократите промпты:

```
# Короткие промпты = меньше токенов
prompt = "Переведи на русский: ..."  # вместо длинного объяснения
```

4. Установите лимиты:

```
OpenAI Dashboard → Billing → Usage limits

Soft limit: $10
Hard limit: $20
```

Google/ElevenLabs

Google бесплатный:

- 60 запросов/минуту бесплатно
- 1M символов TTS бесплатно в месяц

ElevenLabs:

- Free: 10,000 символов/месяц
- Если превысили - используйте OpenAI или Google TTS

Общая стратегия экономии:

1. Приоритеты:
- Бесплатные функции для всех пользователей
- AI функции только для премиум (@WowFUX)
- Ограничения на размер файлов

2. Кеширование:
- Кешировать результаты AI обработки
- Не обрабатывать одинаковые запросы дважды

3. Очистка:
- Автоматически удалять старые файлы (> 7 дней)
- Ограничивать размер кеша
- Очищать логи регулярно

4. Мониторинг:
- Ежедневно проверять расходы
- Установить alerts на превышение бюджета
- Анализировать, какие функции дорогие

💾 Резервное копирование

Что нужно бэкапить:

1. База данных

```
# Скачать DB из Railway
railway run "cat data/bot.db" > backup_$(date +%Y%m%d).db

# Или через SSH (если настроен)
scp railway:/app/data/bot.db ./backups/
```

Частота: Ежедневно (если активно используется)

2. Конфигурация (переменные окружения)

```
# Сохранить список переменных
railway variables > env_backup_$(date +%Y%m%d).txt
```

Частота: После каждого изменения

3. Код (GitHub)

```
# Код автоматически бэкапится в GitHub
# Но можно создать релизы для важных версий
git tag -a v1.0.0 -m "Стабильная версия 1.0"
git push origin v1.0.0
```

Частота: При каждом значительном обновлении

Автоматический бэкап

Настройка cron job (если нужно):

```
# На вашем локальном сервере
crontab -e

# Добавить:
0 2 * * * cd /home/ubuntu/wowsilizing_bot && ./backup.sh
```

Скрипт backup.sh:

```
#!/bin/bash
DATE=$(date +%Y%m%d)
BACKUP_DIR="/home/ubuntu/backups"

# Скачать DB
railway run "cat data/bot.db" > $BACKUP_DIR/bot_$DATE.db

# Сохранить переменные
railway variables > $BACKUP_DIR/env_$DATE.txt

# Удалить старые бэкапы (> 30 дней)
find $BACKUP_DIR -name "*.db" -mtime +30 -delete

echo "Backup completed: $DATE"
```

Восстановление из бэкапа

Восстановление базы данных:

```
# 1. Скачать последний бэкап
cd /home/ubuntu/backups
ls -lt | head

# 2. Загрузить в Railway
railway run "cat > data/bot.db" < bot_20241125.db

# 3. Перезапустить бота
railway restart
```

Восстановление переменных:

```
# Просмотреть сохраненные переменные
cat env_20241125.txt

# Вручную добавить через Railway UI
# Или использовать Railway CLI
```

📈 Масштабирование

Когда масштабировать?

Признаки, что нужно масштабирование:
- ⚠️ CPU постоянно > 80%
- ⚠️ Memory постоянно > 80%
- ⚠️ Задержки в ответах бота
- ⚠️ Очереди обработки растут
- ⚠️ Пользователи жалуются на медленную работу

Вертикальное масштабирование (больше ресурсов)

В Railway:

1. Откройте **Settings** → **Resources**
2. Увеличьте:
   - **Memory**: с 512MB до 1GB или 2GB
   - **CPU**: автоматически масштабируется

Стоимость увеличится пропорционально.

Горизонтальное масштабирование (больше инстансов)

Для Telegram бота:

НЕ рекомендуется! Telegram не поддерживает несколько инстансов одного бота одновременно.

Альтернатива:
- Оптимизировать код
- Использовать очереди задач
- Увеличить вертикальные ресурсы

Оптимизация производительности

1. Асинхронная обработка:

```
# Используйте async/await для I/O операций
import asyncio

async def process_video(video):
    # Обработка в фоне
    await asyncio.sleep(0)  # Yield control
```

2. Очереди задач:

```
from asyncio import Queue

queue = Queue(maxsize=10)

# Обрабатывать задачи последовательно
while True:
    task = await queue.get()
    await process_task(task)
```

3. Кеширование:

```
from functools import lru_cache

@lru_cache(maxsize=100)
def expensive_operation(param):
    # Результат будет кеширован
    return result
```

🔒 Безопасность

Регулярные проверки безопасности

Ежемесячно:

1. Ротация ключей:
- [ ] Проверить дату последней ротации API ключей
- [ ] Если > 6 месяцев - ротировать

2. Проверка доступа:
- [ ] Кто имеет доступ к Railway проекту
- [ ] Удалить неактивных пользователей
- [ ] Проверить 2FA включен

3. Аудит логов:
- [ ] Просмотреть логи на подозрительную активность
- [ ] Проверить необычные паттерны использования
- [ ] Убедиться, что нет утечек данных

4. Обновления зависимостей:
- [ ] Проверить обновления для Python пакетов
- [ ] Обновить критические security патчи
- [ ] Протестировать после обновления

Защита от злоупотреблений

1. Rate Limiting:

```
# В bot.py добавить ограничения
from aiogram.utils import rate_limit

@rate_limit(limit=5, key="video_processing")
async def process_video_handler(message):
    # Не более 5 запросов в минуту
    pass
```

2. Черный список:

```
# В database.py
class Database:
    async def is_banned(self, user_id):
        # Проверка banned пользователей
        pass
```

3. Лимиты на размер файлов:

```
# В config.py
MAX_FILE_SIZE_MB = 50
MAX_VIDEO_DURATION = 3600  # 1 час
```

🛠️ Troubleshooting

Частые проблемы и решения

Проблема 1: Бот периодически падает

Симптомы:

- Статус меняется на “Crashed”
- Railway автоматически перезапускает
- В логах: “Out of memory” или “Killed”

Решение:

```
1. Увеличить Memory в Railway Settings
2. Оптимизировать обработку (меньше CRF, быстрее preset)
3. Ограничить размер входных файлов
4. Удалять временные файлы агрессивнее
```

Проблема 2: Медленная обработка

Симптомы:

- Обработка видео занимает > 5 минут
- Пользователи жалуются на ожидание

Решение:

```python
# Оптимизировать FFmpeg настройки
DEFAULT_PRESET = "ultrafast"  # было "medium"
DEFAULT_CRF = 28  # было 23

# Использовать параллельную обработку
import multiprocessing
pool = multiprocessing.Pool(processes=2)
```

Проблема 3: Превышение API лимитов

Симптомы:

- “Rate limit exceeded” в логах
- AI функции не работают

Решение:

```python
# Добавить retry с экспоненциальной задержкой
import time

for attempt in range(3):
    try:
        result = openai_api_call()
        break
    except RateLimitError:
        time.sleep(2 ** attempt)  # 1s, 2s, 4s
```

Проблема 4: База данных повреждена

Симптомы:
- “Database is locked”
- “Database malformed”

Решение:

```
# 1. Восстановить из бэкапа
railway run "cat > data/bot.db" < backup_latest.db

# 2. Или пересоздать
railway run "rm data/bot.db && python -c 'from database import Database; import
asyncio; asyncio.run(Database().init_db())'"
```

📚 Best Practices

Для стабильной работы:

1. Мониторинг

- ✅ Проверять логи ежедневно
- ✅ Настроить alerts в Railway
- ✅ Следить за расходами API
- ✅ Отслеживать метрики (CPU, Memory)

2. Обновления

- ✅ Тестировать перед деплоем
- ✅ Использовать ветки git для разработки
- ✅ Делать бэкапы перед большими изменениями
- ✅ Обновлять зависимости регулярно

3. Безопасность

- ✅ Ротировать ключи каждые 3-6 месяцев
- ✅ Никогда не коммитить .env
- ✅ Использовать 2FA на всех сервисах
- ✅ Ограничивать доступ к Railway проекту

4. Оптимизация

- ✅ Кешировать результаты
- ✅ Удалять временные файлы
- ✅ Использовать rate limiting
- ✅ Мониторить размер базы данных

5. Документация

- ✅ Документировать изменения (changelog)
- ✅ Обновлять README при добавлении функций
- ✅ Комментировать сложный код
- ✅ Вести журнал инцидентов

📞 Поддержка и ресурсы

Официальная документация:

- 🌐 **Railway Docs:** https://docs.railway.app/
- 🤖 **Aiogram Docs:** https://docs.aiogram.dev/
- 🎬 **FFmpeg Docs:** https://ffmpeg.org/documentation.html
- 🤖 **OpenAI API Docs:** https://platform.openai.com/docs/

Сообщества:

- 💬 **Railway Discord:** https://discord.gg/railway
- 💬 **Aiogram Chat:** https://t.me/aiogram

Ваша документация:

- 📖 **README:** README.md (README.md)
- 🚀 **Развертывание:** DEPLOYMENT.md (DEPLOYMENT.md)
- ⚡ **Быстрый старт:** QUICK_START.md (QUICK_START.md)
- 🔑 **Переменные:** ENV_VARIABLES.md (ENV_VARIABLES.md)
- ✅ **Чеклист:** CHECKLIST.md (CHECKLIST.md)

🎯 Заключение

Поздравляем! Теперь вы знаете всё для успешной эксплуатации бота:

✅ Как мониторить работу
✅ Как обновлять и откатывать
✅ Как управлять API ключами
✅ Как оптимизировать расходы
✅ Как масштабировать
✅ Как обеспечивать безопасность
✅ Как решать проблемы

Ваш бот готов к долгосрочной стабильной работе! 🚀

Последнее обновление: Ноябрь 2024
Версия: 1.0