ИНСТРУКЦИЯ ПО ЭКСПЛУАТАЦИИ

Cognitum AI Platform
Версия 0.4.4

ООО «ИТ-Экспертиза»

Дата сборки: 2025-12-23 Сборка: #522fc53

Оглавление

1 Термины и сокращения 4
1.1 Специфичные термины платформы 4
2 Введение 6
2.1 Назначение документа 6
2.2 Область применения 6
3 Системные требования 7
3.1 Минимальные требования 7
3.2 Рекомендуемые требования 7
3.3 Сетевые порты 8
4 Установка платформы 10
4.1 Подготовка окружения 10
4.2 Установка Docker 11
4.3 Развёртывание из дистрибутива 12
4.4 Первоначальная настройка 13
5 Конфигурирование 16
5.1 Переменные окружения 16
5.2 Настройка базы данных 17
5.3 Настройка NATS 18
5.4 Настройка безопасности 19
5.5 Настройка LDAP 21
5.6 Настройка SMTP 22
5.7 Настройка аудита 23
6 Эксплуатация 26
6.1 Запуск и остановка платформы 26
6.2 Мониторинг состояния 27
6.3 Просмотр логов 29
6.4 Резервное копирование 30
6.5 Восстановление данных 32
7 Обновление 34
7.1 Подготовка к обновлению 34
7.2 Процедура обновления 35
7.3 Откат изменений 36
8 Разработка и развёртывание агентов 38
8.1 Создание агента с SDK 38
8.2 Контейнеризация агента 39
8.3 Развёртывание агента 41
8.4 Мониторинг агента 42
9 Устранение неисправностей 44
9.1 Типичные проблемы и решения 44
9.2 Диагностика NATS 45
9.3 Диагностика базы данных 47
9.4 Работа с логами (Loki) 48

10 Приложения 50
10.1 Справочник переменных окружения 50
10.2 Структура docker-compose.yml 50
10.3 Команды управления 51

1 Термины и сокращения

В настоящем документе используются следующие термины и сокращения:

Термин	Определение
AI-агент	Автономный программный модуль, выполняющий задачи с использованием технологий искусственного интеллекта
API	Application Programming Interface — программный интерфейс приложения
Control Plane	Централизованный компонент управления платформой, координирующий работу агентов
Docker	Платформа контейнеризации для развертывания приложений
Embedding	Векторное представление текста, используемое для семантического поиска
JetStream	Подсистема персистентных сообщений в NATS
Job	Задача — единица работы, выполняемая агентом
JWT	JSON Web Token — стандарт токенов для аутентификации
LDAP	Lightweight Directory Access Protocol — протокол доступа к каталогам
LLM	Large Language Model — большая языковая модель
Loki	Система агрегации логов от Grafana
MCP	Model Context Protocol — протокол для расширения возможностей LLM
NATS	Высокопроизводительный брокер сообщений
Neo4j	Графовая база данных
Ollama	Платформа для локального запуска LLM-моделей
PostgreSQL	Реляционная СУБД с открытым исходным кодом
Qdrant	Векторная база данных для хранения эмбеддингов
RAG	Retrieval-Augmented Generation — генерация с дополнением из внешних источников
REST	Representational State Transfer — архитектурный стиль API
S3	Simple Storage Service — объектное хранилище (стандарт Amazon)
SDK	Software Development Kit — набор инструментов разработчика
SMTP	Simple Mail Transfer Protocol — протокол отправки электронной почты
SSE	Server-Sent Events — технология потоковой передачи данных
Stream	Поток сообщений в JetStream с гарантией доставки
TTL	Time To Live — время жизни объекта
UUID	Universally Unique Identifier — универсальный уникальный идентификатор

1.1 Специфичные термины платформы

Термин	Определение
Cognitum	Название платформы для создания и управления AI-агентами
Платформа	Cognitum AI Platform — программный комплекс
Агент	AI-агент, зарегистрированный в платформе
Провайдер LLM	Источник языковых моделей (OpenAI, Azure, Ollama)
Хранилище	Внешнее хранилище данных, подключенное к платформе
Очередь	JetStream-стрим для обмена сообщениями
Задача (Job)	Асинхронная единица работы с жизненным циклом
Чат-модель	Модель агента, доступная через Chat API

Термин	Определение
Дистрибутив	Пакет для развертывания платформы

2 Введение

Настоящий документ является инструкцией по эксплуатации программного продукта **Cognitum AI Platform**.

Документ содержит информацию, необходимую для установки, настройки, эксплуатации и обслуживания платформы.

2.1 Назначение документа

Настоящая инструкция предназначена для:

- Администраторов IT-инфраструктуры
- Специалистов по развёртыванию и эксплуатации
- DevOps-инженеров

Документ описывает:

1. Системные требования к оборудованию и ПО
2. Процедуру установки и первоначальной настройки
3. Конфигурирование компонентов платформы
4. Процедуры эксплуатации и обслуживания
5. Процедуры обновления
6. Процедуры разработки и развёртывания агентов
7. Диагностику и устранение неисправностей

2.2 Область применения

Инструкция применима к версии платформы **0.4.x** и описывает:

- Развёртывание в контейнерной среде Docker
- On-premise установку на серверах предприятия
- Интеграцию с корпоративной инфраструктурой

2.2.1 Поддерживаемые операционные системы

ОС	Версия	Поддержка
Ubuntu	20.04 LTS, 22.04 LTS	Полная
Debian	11, 12	Полная
CentOS	8, 9	Полная
RHEL	8, 9	Полная
Astra Linux	1.7+	Полная
macOS	12+	Разработка
Windows	10/11 + WSL2	Разработка

2.2.2 Требуемые навыки персонала

Для работы с данной инструкцией необходимы знания:

- Администрирование Linux-систем
- Работа с Docker и docker-compose
- Базовые знания сетевых протоколов
- Работа с PostgreSQL (базовый уровень)

3 Системные требования

В данном разделе описаны требования к аппаратному и программному обеспечению для развёртывания платформы Cognitum.

3.1 Минимальные требования

Минимальные требования для развёртывания платформы в режиме разработки или тестирования.

3.1.1 Аппаратное обеспечение

Компонент	Требование
CPU	4 ядра (x86_64)
RAM	8 GB
Диск	20 GB свободного места
Сеть	100 Mbit/s

3.1.2 Программное обеспечение

Компонент	Версия
Docker Engine	24.0+
Docker Compose	2.20+ (включён в Docker Desktop)

3.1.3 Операционная система

Linux: - Ubuntu 20.04 LTS или новее - Debian 11 или новее - CentOS/RHEL 8 или новее

macOS: - macOS 12 (Monterey) или новее - Docker Desktop 4.22+

Windows: - Windows 10/11 с WSL2 - Docker Desktop с WSL2 backend

3.1.4 Дополнительно

- Доступ к Docker registry (для загрузки образов)
- Свободные порты: 8080, 4222, 5432, 3100

3.2 Рекомендуемые требования

Рекомендуемые требования для production-развёртывания.

3.2.1 Аппаратное обеспечение

Компонент	Требование
CPU	8+ ядер (x86_64)
RAM	16+ GB
Диск	100+ GB SSD
Сеть	1 Gbit/s

3.2.2 Дисковая подсистема

Рекомендуется SSD для: - PostgreSQL (IOPS критичен) - NATS JetStream (персистентность) - Loki (хранение логов)

Структура дискового пространства:

Компонент	Размер	Описание
PostgreSQL	20+ GB	База данных

Компонент	Размер	Описание
NATS	10+ GB	Очереди сообщений
Loki	50+ GB	Логи
Qdrant	10+ GB	Векторы
Neo4j	10+ GB	Графы

3.2.3 Сетевые требования

- Стабильное соединение между компонентами
- Низкая задержка (<1 ms) для NATS
- Пропускная способность для логов (зависит от нагрузки)

3.2.4 Высокая доступность

Для mission-critical развёртывания:

Компонент	Рекомендация
PostgreSQL	Primary + Replica
NATS	Кластер из 3 нод
Приложение	2+ экземпляра за балансировщиком

3.2.5 Резервирование

- Регулярное резервное копирование PostgreSQL
- Репликация данных на отдельное хранилище
- Мониторинг дискового пространства

3.3 Сетевые порты

Платформа использует следующие сетевые порты.

3.3.1 Внешние порты

Порты, доступные извне Docker-сети.

Порт	Сервис	Протокол	Описание
8080	cognitum-app	HTTP	Веб-интерфейс и API
4222	NATS	TCP	Клиентские подключения
8222	NATS	HTTP	Мониторинг NATS
5432	PostgreSQL	TCP	База данных (опционально)
3100	Loki	HTTP	API логов
6333	Qdrant	HTTP	Векторная БД
7474	Neo4j	HTTP	Веб-консоль Neo4j
7687	Neo4j	Bolt	Подключения к Neo4j

3.3.2 Внутренние порты

Порты для взаимодействия внутри Docker-сети.

Порт	Сервис	Описание
8000	Backend	FastAPI
3000	Frontend	Next.js
9125	Promtail	Сбор логов

3.3.3 Настройка firewall

Для production-установки откройте только необходимые порты:

```
# Ubuntu/Debian (ufw)
sudo ufw allow 8080/tcp  # Веб-интерфейс
sudo ufw allow 4222/tcp  # NATS (если агенты внешние)

# CentOS/RHEL (firewalld)
sudo firewall-cmd --permanent --add-port=8080/tcp
sudo firewall-cmd --permanent --add-port=4222/tcp
sudo firewall-cmd --reload
```

3.3.4 Изменение портов

Порты настраиваются через переменные окружения в `.env`:

```
FRONTEND_PORT=80          # Веб-интерфейс на порту 80
NATS_PORT=14222           # NATS на альтернативном порту
POSTGRES_PORT=15432       # PostgreSQL на альтернативном порту
```

3.3.5 Проверка занятости портов

```
# Linux
sudo lsof -i :8080
sudo netstat -tlnp | grep 8080

# macOS
lsof -i :8080

# Windows
netstat -ano | findstr :8080
```

3.3.6 Рекомендации по безопасности

1. **Не открывайте PostgreSQL (5432) извне** без необходимости
2. **Используйте reverse proxy** (nginx) для HTTPS
3. **Ограничьте доступ к NATS** только для агентов
4. **Закройте порты мониторинга** (8222, 6333) извне

4 Установка платформы

В данном разделе описана процедура установки платформы Cognitum из дистрибутива.

4.1 Подготовка окружения

4.1.1 Проверка системных требований

Перед установкой убедитесь, что система соответствует требованиям:

```
# Проверка CPU
lscpu | grep "CPU(s)"

# Проверка RAM
free -h

# Проверка дискового пространства
df -h
```

4.1.2 Настройка системы

4.1.2.1 Увеличение лимитов файловых дескрипторов

```
# /etc/security/limits.conf
* soft nofile 65535
* hard nofile 65535
```

4.1.2.2 Настройка виртуальной памяти (для Qdrant)

```
# /etc/sysctl.conf
vm.max_map_count=262144

# Применить без перезагрузки
sudo sysctl -w vm.max_map_count=262144
```

4.1.3 Создание рабочей директории

```
sudo mkdir -p /opt/cognitum
sudo chown $USER:$USER /opt/cognitum
cd /opt/cognitum
```

4.1.4 Распаковка дистрибутива

```
unzip cognitum-distro-*.zip -d .
cd cognitum
```

4.1.5 Структура дистрибутива

```
cognitum/
├── images/                    # Docker образы
│   ├── cognitum-app.tar.gz
│   ├── cognitum-echo-agent.tar.gz
│   └── load_images.sh
├── sdk/                       # SDK для разработки агентов
├── config/                    # Конфигурационные файлы
│   ├── docker-compose.yml
│   ├── env.example
│   ├── deploy.sh
│   └── ...
├── INSTALL.md
└── VERSION
```

4.1.6 Сетевая конфигурация

Убедитесь, что необходимые порты свободны:

```
for port in 8080 4222 5432 3100 6333 7474; do
    if lsof -i :$port > /dev/null 2>&1; then
        echo "Порт $port занят!"
    else
        echo "Порт $port свободен"
```

```
    fi
done
```

4.2 Установка Docker

4.2.1 Linux (Ubuntu/Debian)

```
# Удаление старых версий
sudo apt-get remove docker docker-engine docker.io containerd runc

# Установка зависимостей
sudo apt-get update
sudo apt-get install -y ca-certificates curl gnupg

# Добавление GPG ключа Docker
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
    sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

# Добавление репозитория
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] \
    https://download.docker.com/linux/ubuntu $(. /etc/os-release && echo "$VERSION_CODENAME") stable"
| \
    sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# Установка Docker
sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plug
in

# Добавление пользователя в группу docker
sudo usermod -aG docker $USER

# Применение без перезагрузки
newgrp docker
```

4.2.2 Linux (CentOS/RHEL)

```
# Удаление старых версий
sudo yum remove docker docker-client docker-client-latest \
    docker-common docker-latest docker-latest-logrotate \
    docker-logrotate docker-engine

# Установка зависимостей
sudo yum install -y yum-utils

# Добавление репозитория
sudo yum-config-manager --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo

# Установка Docker
sudo yum install -y docker-ce docker-ce-cli containerd.io \
    docker-buildx-plugin docker-compose-plugin

# Запуск Docker
sudo systemctl start docker
sudo systemctl enable docker

# Добавление пользователя в группу docker
sudo usermod -aG docker $USER
```

4.2.3 Astra Linux

```
# Установка из репозитория Astra
sudo apt-get update
sudo apt-get install -y docker.io docker-compose

# Запуск Docker
sudo systemctl start docker
sudo systemctl enable docker
```

```
# Добавление пользователя в группу docker
sudo usermod -aG docker $USER
```

4.2.4 macOS / Windows

Установите Docker Desktop:

1. Скачайте установщик с официального сайта
2. Запустите установщик
3. Следуйте инструкциям на экране
4. Перезагрузите компьютер (Windows)

4.2.5 Проверка установки

```
# Версия Docker
docker --version
# Docker version 24.x.x

# Версия Compose
docker compose version
# Docker Compose version v2.x.x

# Тестовый запуск
docker run hello-world
```

4.2.6 Конфигурация Docker

4.2.6.1 Настройка хранилища (Linux)

```
# /etc/docker/daemon.json
{
  "data-root": "/var/lib/docker",
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m",
    "max-file": "3"
  }
}

# Применить
sudo systemctl restart docker
```

4.2.6.2 Настройка ресурсов (Docker Desktop)

В настройках Docker Desktop увеличьте лимиты: - Memory: 8 GB+ - CPUs: 4+ - Disk image size: 50 GB+

4.3 Развёртывание из дистрибутива

4.3.1 Загрузка Docker образов

```
cd images

# Автоматическая загрузка всех образов
./load_images.sh

# Или вручную
gunzip -c cognitum-app.tar.gz | docker load
gunzip -c cognitum-echo-agent.tar.gz | docker load

# Проверка загруженных образов
docker images | grep cognitum
```

4.3.2 Настройка конфигурации

```
cd config
cp env.example .env
```

Отредактируйте файл `.env`:

```
nano .env
```

Обязательные изменения:

```
# Пароль PostgreSQL (замените на безопасный)
POSTGRES_PASSWORD=your_secure_password_here

# Пароль Neo4j
NEO4J_PASSWORD=your_neo4j_password_here

# Секретный ключ приложения (генерируйте случайный)
APP_SECRET_KEY=$(openssl rand -hex 32)

# Секретный ключ JWT
JWT_SECRET_KEY=$(openssl rand -hex 32)
```

4.3.3 Генерация безопасных ключей

```
# Python
python3 -c "import secrets; print(secrets.token_hex(32))"

# OpenSSL
openssl rand -hex 32
```

4.3.4 Запуск платформы

```
# Запуск в фоновом режиме
./deploy.sh up -d

# Просмотр статуса
./deploy.sh ps

# Просмотр логов
./deploy.sh logs
```

4.3.5 Проверка запуска

```
# Все контейнеры должны быть в статусе "Up" или "healthy"
docker compose ps

# Ожидаемый вывод:
# NAME                    STATUS
# cognitum-postgres       Up (healthy)
# cognitum-nats           Up (healthy)
# cognitum-loki           Up (healthy)
# cognitum-promtail       Up
# cognitum-qdrant         Up
# cognitum-neo4j          Up
# cognitum-app            Up (healthy)
# cognitum-echo-agent     Up
```

4.3.6 Проверка доступности

```
# Health check
curl http://localhost:8080/api/health

# Версия
curl http://localhost:8080/api/version
```

4.3.7 Доступ к веб-интерфейсу

Откройте в браузере: `http://localhost:8080`

Учётные данные по умолчанию: - Логин: `admin` - Пароль: `admin`

> **Важно:** Обязательно смените пароль администратора после первого входа!

4.4 Первоначальная настройка

После успешного запуска платформы выполните первоначальную настройку.

4.4.1 Смена пароля администратора

1. Войдите в систему с учётными данными по умолчанию

2. Перейдите в **Profile** (иконка пользователя)
3. Нажмите **Change Password**
4. Установите надёжный пароль

4.4.2 Добавление LLM-провайдера

Для работы агентов с LLM необходимо настроить хотя бы один провайдер:

1. Перейдите в **LLM APIs → Add Provider**
2. Выберите тип провайдера:
   - **OpenAI** — для облачных моделей OpenAI
   - **Ollama** — для локальных моделей
   - **Azure OpenAI** — для корпоративного Azure
3. Заполните параметры подключения
4. Нажмите **Save**

4.4.2.1 Пример: Добавление Ollama

Если Ollama запущен на том же хосте:

```
Type: Ollama
Name: Local Ollama
Base URL: http://host.docker.internal:11434
```

4.4.3 Проверка агента

После запуска `cognitum-echo-agent` должен автоматически зарегистрироваться:

1. Перейдите в **Agents**
2. Убедитесь, что `cognitum-echo-agent` в статусе **Online**

4.4.4 Тестирование системы задач

1. Перейдите в **Queues**
2. Убедитесь, что стримы AGENTS, JOBS, CONTROL созданы
3. Создайте тестовую задачу через API:

```
curl -X POST http://localhost:8080/api/jobs/invoke \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer <your_token>" \
  -d '{"job_type": "echo-job", "body": {"message": "Hello!"}, "timeout": 30}'
```

4.4.5 Настройка SMTP (опционально)

Для функции восстановления пароля настройте SMTP:

```
# .env
SMTP_HOST=smtp.company.com
SMTP_PORT=587
SMTP_USER=noreply@company.com
SMTP_PASSWORD=your_smtp_password
SMTP_FROM=Cognitum <noreply@company.com>
SMTP_TLS=true
```

Перезапустите платформу:

```
./deploy.sh restart
```

4.4.6 Настройка LDAP (опционально)

Для интеграции с корпоративным каталогом:

```
# .env
LDAP_ENABLED=true
```

```
LDAP_SERVER=ldaps://ldap.company.com:636
LDAP_BASE_DN=dc=company,dc=com
LDAP_USER_DN_TEMPLATE={username}@company.com
```

4.4.7 Создание API-токена

Для интеграции с внешними системами создайте API-токен:

1. Перейдите в **Admin → API Tokens**
2. Нажмите **Create Token**
3. Введите название токена
4. **Скопируйте токен —** он показывается только один раз!

5 Конфигурирование

В данном разделе описаны параметры конфигурации компонентов платформы.

Основные настройки выполняются через переменные окружения в файле `.env`.

5.1 Переменные окружения

Все настройки платформы выполняются через переменные окружения в файле `.env`.

5.1.1 Приоритет конфигурации

1. Переменные окружения системы (высший приоритет)
2. Файл `.env`
3. Значения по умолчанию

5.1.2 Формат файла .env

```
# Комментарии начинаются с #
VARIABLE_NAME=value

# Значения с пробелами в кавычках
DESCRIPTION="My Platform Instance"

# Пустые значения
OPTIONAL_SETTING=
```

5.1.3 Применение изменений

После изменения `.env` перезапустите платформу:

```
./deploy.sh down
./deploy.sh up -d
```

Или для отдельных сервисов:

```
docker compose restart app
```

5.1.4 Проверка конфигурации

```
# Показать текущую конфигурацию
docker compose config

# Проверить переменные в контейнере
docker exec cognitum-app env | grep APP_
```

5.1.5 Секреты

Для production рекомендуется использовать менеджеры секретов:

- Docker Secrets
- HashiCorp Vault
- AWS Secrets Manager

Пример с Docker Secrets:

```
# docker-compose.yml
services:
  app:
    secrets:
      - postgres_password
    environment:
      POSTGRES_PASSWORD_FILE: /run/secrets/postgres_password

secrets:
  postgres_password:
    file: ./secrets/postgres_password.txt
```

5.2 Настройка базы данных

5.2.1 PostgreSQL

PostgreSQL — основная база данных платформы.

5.2.1.1 Переменные окружения

```
POSTGRES_USER=aiplatform       # Пользователь
POSTGRES_PASSWORD=password     # Пароль
POSTGRES_DB=aiplatform         # База данных
POSTGRES_PORT=5432             # Порт
```

5.2.1.2 Подключение к PostgreSQL

```
# Через docker exec
docker exec -it cognitum-postgres psql -U aiplatform -d aiplatform

# Внешнее подключение
psql -h localhost -p 5432 -U aiplatform -d aiplatform
```

5.2.1.3 Резервное копирование

```
# Создание дампа
docker exec cognitum-postgres pg_dump -U aiplatform aiplatform > backup.sql

# Восстановление
cat backup.sql | docker exec -i cognitum-postgres psql -U aiplatform aiplatform
```

5.2.2 Qdrant

Векторная база данных для эмбеддингов.

5.2.2.1 Переменные окружения

```
QDRANT_PORT=6333               # HTTP порт
```

5.2.2.2 Веб-интерфейс

Доступен по адресу: `http://localhost:6333/dashboard`

5.2.3 Neo4j

Графовая база данных.

5.2.3.1 Переменные окружения

```
NEO4J_PASSWORD=changeme        # Пароль пользователя neo4j
NEO4J_HTTP_PORT=7474           # HTTP порт
NEO4J_BOLT_PORT=7687           # Bolt порт
```

5.2.3.2 Веб-интерфейс

Доступен по адресу: `http://localhost:7474`

Подключение: - URL: `bolt://localhost:7687` - User: `neo4j` - Password: значение `NEO4J_PASSWORD`

5.2.4 Производительность PostgreSQL

Для production рекомендуется настроить параметры в `postgresql.conf`:

```
# Создайте файл config/postgresql.conf
shared_buffers = 256MB
effective_cache_size = 768MB
maintenance_work_mem = 64MB
checkpoint_completion_target = 0.9
wal_buffers = 16MB
default_statistics_target = 100
random_page_cost = 1.1
effective_io_concurrency = 200
```

```
work_mem = 4MB
min_wal_size = 1GB
max_wal_size = 4GB
max_worker_processes = 4
max_parallel_workers_per_gather = 2
max_parallel_workers = 4
```

Подключите через docker-compose:

```
volumes:
  - ./config/postgresql.conf:/etc/postgresql/postgresql.conf
command: postgres -c config_file=/etc/postgresql/postgresql.conf
```

5.3 Настройка NATS

NATS JetStream — брокер сообщений платформы.

5.3.1 Переменные окружения

```
NATS_PORT=4222                  # Порт клиентских подключений
NATS_MONITORING_PORT=8222       # Порт мониторинга
```

5.3.2 Конфигурационный файл

Платформа использует `nats-config.conf`:

```
# nats-config.conf

# Порты
port: 4222
http_port: 8222

# JetStream
jetstream {
    store_dir: /data/jetstream
    max_memory_store: 1G
    max_file_store: 10G
}

# Логирование
debug: false
trace: false
logtime: true
```

5.3.3 Мониторинг NATS

Веб-интерфейс мониторинга: `http://localhost:8222`

5.3.3.1 Полезные endpoints

```
# Общая информация
curl http://localhost:8222/varz

# JetStream информация
curl http://localhost:8222/jsz

# Подключения
curl http://localhost:8222/connz

# Подписки
curl http://localhost:8222/subsz
```

5.3.4 JetStream стримы

Платформа автоматически создаёт стримы при запуске:

Стрим	Subjects	Retention
AGENTS	`agent.>`	LIMITS (7 дней)

Стрим	Subjects	Retention
JOBS	`job.>`	WORK_QUEUE (24 часа)
CONTROL	`control.>`	WORK_QUEUE (1 час)

5.3.5 Управление стримами

```
# Подключение к NATS CLI (в контейнере)
docker exec -it cognitum-nats nats

# Список стримов
nats stream list

# Информация о стриме
nats stream info JOBS

# Очистка стрима
nats stream purge JOBS
```

5.3.6 Производительность

Для высоких нагрузок настройте:

```
# nats-config.conf

jetstream {
    max_memory_store: 4G
    max_file_store: 100G
}

# Увеличение буферов
max_payload: 8MB
max_pending: 64MB
write_deadline: 10s
```

5.3.7 Кластеризация (production)

Для высокой доступности используйте кластер из 3 нод:

```
# nats-config.conf (нода 1)
server_name: nats-1
cluster {
    name: cognitum-cluster
    port: 6222
    routes: [
        nats://nats-2:6222
        nats://nats-3:6222
    ]
}
```

5.4 Настройка безопасности

5.4.1 Секретные ключи

5.4.1.1 APP_SECRET_KEY

Используется для шифрования внутренних данных:

```
# Генерация
openssl rand -hex 32
# или
python3 -c "import secrets; print(secrets.token_hex(32))"

# .env
APP_SECRET_KEY=a1b2c3d4e5f6...
```

5.4.1.2 JWT_SECRET_KEY

Используется для подписи JWT-токенов:

```
JWT_SECRET_KEY=9z8y7x6w5v4u...
JWT_EXPIRATION_HOURS=24
```

Важно: Используйте разные ключи для APP_SECRET_KEY и JWT_SECRET_KEY!

5.4.2 Требования к паролям

Рекомендуемые требования:

- Минимум 12 символов
- Буквы верхнего и нижнего регистра
- Цифры
- Специальные символы

5.4.3 HTTPS (TLS)

Для production **обязательно** используйте HTTPS.

5.4.3.1 Nginx как reverse proxy

```
# /etc/nginx/sites-available/cognitum
server {
    listen 80;
    server_name cognitum.company.com;
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl http2;
    server_name cognitum.company.com;

    ssl_certificate /etc/ssl/certs/cognitum.crt;
    ssl_certificate_key /etc/ssl/private/cognitum.key;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256;
    ssl_prefer_server_ciphers off;

    location / {
        proxy_pass http://localhost:8080;
        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;
    }
}
```

5.4.4 Ограничение доступа

5.4.4.1 По IP-адресам

```
location / {
    allow 10.0.0.0/8;
    allow 192.168.0.0/16;
    deny all;

    proxy_pass http://localhost:8080;
}
```

5.4.4.2 Basic Auth (дополнительно)

```
location / {
    auth_basic "Restricted Access";
    auth_basic_user_file /etc/nginx/.htpasswd;

    proxy_pass http://localhost:8080;
}
```

5.4.5 Firewall

Ограничьте доступ к внутренним портам:

```
# Разрешить только веб-интерфейс
sudo ufw allow 443/tcp
sudo ufw allow 80/tcp

# Закрыть внутренние порты
sudo ufw deny 5432/tcp
sudo ufw deny 4222/tcp
```

5.4.6 Аудит

Включите аудит для отслеживания действий:

```
AUDIT_ENABLED=true
AUDIT_RETENTION_DAYS=365
```

5.5 Настройка LDAP

5.5.1 Включение LDAP

```
# .env
LDAP_ENABLED=true
```

5.5.2 Параметры подключения

5.5.2.1 Active Directory

```
LDAP_SERVER=ldap://ad.company.com:389
LDAP_BASE_DN=dc=company,dc=com
LDAP_USER_DN_TEMPLATE={username}@company.com
LDAP_SERVICE_ACCOUNT=cognitum-svc@company.com
LDAP_SERVICE_ACCOUNT_PASSWORD=****
```

5.5.2.2 OpenLDAP

```
LDAP_SERVER=ldap://ldap.company.local:389
LDAP_BASE_DN=dc=company,dc=local
LDAP_USER_DN_TEMPLATE=uid={username},ou=users,dc=company,dc=local
LDAP_SERVICE_ACCOUNT=cn=admin,dc=company,dc=local
LDAP_SERVICE_ACCOUNT_PASSWORD=****
```

5.5.3 LDAPS (SSL/TLS)

Для защищённого подключения:

```
LDAP_SERVER=ldaps://ldap.company.com:636
```

Если используется самоподписанный сертификат, добавьте его в trust store контейнера.

5.5.4 Проверка подключения

```
# Из контейнера
docker exec cognitum-app python -c "
import ldap
conn = ldap.initialize('ldap://ldap.company.local:389')
conn.simple_bind_s('cn=admin,dc=company,dc=local', 'password')
print('LDAP connection successful')
"

# Через ldapsearch
```

```
ldapsearch -x -H ldap://ldap.company.local:389 \
    -D "cn=admin,dc=company,dc=local" \
    -w password \
    -b "dc=company,dc=local" \
    "(uid=testuser)"
```

5.5.5 Сопоставление атрибутов

Платформа использует следующие атрибуты LDAP:

Атрибут платформы	LDAP атрибут
username	uid (OpenLDAP) / sAMAccountName (AD)
email	mail
display_name	cn / displayName

5.5.6 Troubleshooting

5.5.6.1 Ошибка "Invalid credentials"

- Проверьте формат `LDAP_USER_DN_TEMPLATE`
- Убедитесь, что пароль правильный
- Проверьте, не заблокирована ли учётная запись

5.5.6.2 Ошибка "Can't contact LDAP server"

- Проверьте сетевую доступность сервера
- Убедитесь, что порт открыт
- Проверьте DNS-разрешение имени сервера

5.5.6.3 Пользователь не найден

- Проверьте `LDAP_BASE_DN`
- Убедитесь, что пользователь существует в указанном OU
- Проверьте права сервисной учётной записи

5.6 Настройка SMTP

SMTP используется для: - Восстановления пароля - Уведомлений (опционально)

5.6.1 Параметры

```
SMTP_HOST=smtp.company.com
SMTP_PORT=587
SMTP_USER=noreply@company.com
SMTP_PASSWORD=****
SMTP_FROM=Cognitum Platform <noreply@company.com>
SMTP_TLS=true
```

5.6.2 Примеры конфигурации

5.6.2.1 Gmail

```
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USER=your-email@gmail.com
SMTP_PASSWORD=your-app-password    # App Password, не основной пароль!
SMTP_FROM=your-email@gmail.com
SMTP_TLS=true
```

Для Gmail необходимо создать App Password в настройках безопасности.

5.6.2.2 Yandex

```
SMTP_HOST=smtp.yandex.ru
SMTP_PORT=587
SMTP_USER=your-email@yandex.ru
```

```
SMTP_PASSWORD=****
SMTP_FROM=your-email@yandex.ru
SMTP_TLS=true
```

5.6.2.3 Microsoft 365

```
SMTP_HOST=smtp.office365.com
SMTP_PORT=587
SMTP_USER=noreply@company.com
SMTP_PASSWORD=****
SMTP_FROM=noreply@company.com
SMTP_TLS=true
```

5.6.2.4 Локальный SMTP (без аутентификации)

```
SMTP_HOST=mail.company.local
SMTP_PORT=25
SMTP_USER=
SMTP_PASSWORD=
SMTP_FROM=cognitum@company.local
SMTP_TLS=false
```

5.6.3 Проверка отправки

```
# Из контейнера
docker exec cognitum-app python -c "
import smtplib
from email.mime.text import MIMEText

msg = MIMEText('Test message')
msg['Subject'] = 'Test'
msg['From'] = 'noreply@company.com'
msg['To'] = 'admin@company.com'

with smtplib.SMTP('smtp.company.com', 587) as server:
    server.starttls()
    server.login('noreply@company.com', 'password')
    server.send_message(msg)
    print('Email sent successfully')
"
```

5.6.4 Troubleshooting

5.6.4.1 Ошибка подключения

- Проверьте сетевую доступность SMTP-сервера
- Убедитесь, что порт не заблокирован firewall
- Проверьте правильность хоста и порта

5.6.4.2 Ошибка аутентификации

- Проверьте логин и пароль
- Для Gmail используйте App Password
- Проверьте, разрешён ли доступ приложений

5.6.4.3 Письма не доходят

- Проверьте папку “Спам”
- Настройте SPF/DKIM/DMARC для домена
- Проверьте логи SMTP-сервера

5.7 Настройка аудита

Система аудита фиксирует все значимые действия пользователей.

5.7.1 Включение аудита

```
AUDIT_ENABLED=true
```

5.7.2 Параметры

```
AUDIT_FILE_PATH=/var/log/cognitum/audit.log
AUDIT_ROTATION_SIZE=100MB
AUDIT_ROTATION_COUNT=10
AUDIT_RETENTION_DAYS=365
```

5.7.3 Описание параметров

Параметр	Описание
`AUDIT_FILE_PATH`	Путь к файлу аудита
`AUDIT_ROTATION_SIZE`	Размер файла для ротации
`AUDIT_ROTATION_COUNT`	Количество файлов для хранения
`AUDIT_RETENTION_DAYS`	Срок хранения записей

5.7.4 Регистрируемые события

Категория	События
Аутентификация	Вход, выход, неудачные попытки
Пользователи	Создание, изменение, удаление
Токены	Создание, отзыв
Агенты	Изменение конфигурации
LLM	Добавление/удаление провайдеров
Хранилища	Создание, изменение

5.7.5 Формат записей

```
{
  "timestamp": "2024-01-15T10:30:00Z",
  "event_type": "user.login",
  "user_id": "admin",
  "ip_address": "192.168.1.100",
  "user_agent": "Mozilla/5.0...",
  "details": {
    "method": "local"
  },
  "status": "success"
}
```

5.7.6 Просмотр аудит-лога

5.7.6.1 В веб-интерфейсе

1. Перейдите в **Admin → Audit Log**
2. Используйте фильтры для поиска

5.7.6.2 Из файла

```
# Последние записи
docker exec cognitum-app tail -100 /var/log/cognitum/audit.log

# Поиск по пользователю
docker exec cognitum-app grep '"user_id":"admin"' /var/log/cognitum/audit.log

# С форматированием
docker exec cognitum-app cat /var/log/cognitum/audit.log | jq .
```

5.7.7 Экспорт аудита

```
# Экспорт за период
docker exec cognitum-app cat /var/log/cognitum/audit.log | \
    jq 'select(.timestamp >= "2024-01-01" and .timestamp <= "2024-01-31")' > audit_jan.json
```

5.7.8 Хранение аудита

Для compliance рекомендуется:

1. Экспортировать аудит во внешнее хранилище
2. Использовать SIEM-системы
3. Настроить длительный срок хранения

```
# Пример: копирование на внешний сервер
docker cp cognitum-app:/var/log/cognitum/audit.log ./
scp audit.log backup-server:/archive/cognitum/
```

6 Эксплуатация

В данном разделе описаны процедуры повседневной эксплуатации платформы.

6.1 Запуск и остановка платформы

6.1.1 Запуск

```
cd /opt/cognitum/config

# Запуск всех сервисов
./deploy.sh up -d

# Или через docker compose напрямую
docker compose up -d
```

6.1.2 Остановка

```
# Остановка всех сервисов (данные сохраняются)
./deploy.sh down

# Или
docker compose down
```

6.1.3 Перезапуск

```
# Перезапуск всех сервисов
./deploy.sh restart

# Перезапуск конкретного сервиса
docker compose restart app
docker compose restart nats
```

6.1.4 Статус сервисов

```
# Статус всех контейнеров
./deploy.sh ps

# Подробный статус
docker compose ps -a
```

6.1.5 Просмотр логов

```
# Логи всех сервисов
./deploy.sh logs

# Логи конкретного сервиса
docker compose logs app
docker compose logs -f app     # В реальном времени

# Последние N строк
docker compose logs --tail=100 app
```

6.1.6 Порядок запуска

При запуске сервисы стартуют в порядке зависимостей:

1. PostgreSQL
2. NATS
3. Loki, Qdrant, Neo4j
4. cognitum-app
5. Агенты

6.1.7 Порядок остановки

При остановке:

1. Сначала останавливаются агенты
2. Затем приложение

3. Вспомогательные сервисы
4. Базы данных

6.1.8 Graceful shutdown

Платформа поддерживает корректное завершение:

- Агенты завершают текущие задачи
- Соединения закрываются корректно
- Данные сохраняются

Таймаут по умолчанию: 10 секунд.

```
# Увеличить таймаут остановки
docker compose stop -t 30
```

6.1.9 Автозапуск

Для автоматического запуска при перезагрузке сервера:

```
# docker-compose.yml
services:
  app:
    restart: unless-stopped
```

Или через systemd:

```
# /etc/systemd/system/cognitum.service
[Unit]
Description=Cognitum AI Platform
After=docker.service
Requires=docker.service

[Service]
Type=oneshot
RemainAfterExit=yes
WorkingDirectory=/opt/cognitum/config
ExecStart=/usr/bin/docker compose up -d
ExecStop=/usr/bin/docker compose down

[Install]
WantedBy=multi-user.target

sudo systemctl enable cognitum
```

6.2 Мониторинг состояния

6.2.1 Health Check API

```
# Проверка здоровья приложения
curl http://localhost:8080/api/health

# Ответ
{
  "status": "healthy",
  "components": {
    "database": "healthy",
    "nats": "healthy"
  }
}
```

6.2.2 Мониторинг контейнеров

```
# Статус контейнеров
docker compose ps

# Использование ресурсов
docker stats
```

```
# Использование диска
docker system df
```

6.2.3 Мониторинг NATS

```
# Общая информация
curl http://localhost:8222/varz | jq .

# JetStream информация
curl http://localhost:8222/jsz | jq .

# Активные подключения
curl http://localhost:8222/connz | jq .

# Стримы
curl http://localhost:8222/jsz?streams=true | jq .
```

6.2.4 Мониторинг PostgreSQL

```
# Подключение
docker exec -it cognitum-postgres psql -U aiplatform -d aiplatform

# Активные соединения
SELECT * FROM pg_stat_activity WHERE datname = 'aiplatform';

# Размер базы
SELECT pg_size_pretty(pg_database_size('aiplatform'));

# Размер таблиц
SELECT tablename, pg_size_pretty(pg_total_relation_size(tablename::regclass))
FROM pg_tables WHERE schemaname = 'public' ORDER BY pg_total_relation_size(tablename::regclass) DESC;
```

6.2.5 Мониторинг дискового пространства

```
# Docker volumes
docker system df -v

# Размер data директорий
du -sh /var/lib/docker/volumes/cognitum_*
```

6.2.6 Оповещения

Рекомендуется настроить оповещения при:

- Диск заполнен > 80%
- Контейнер перезапустился
- Health check failed
- Очередь сообщений переполнена

6.2.7 Интеграция с Prometheus/Grafana

NATS предоставляет метрики в формате Prometheus:

```
curl http://localhost:8222/metrics
```

Пример dashboard для Grafana доступен в официальном репозитории NATS.

6.2.8 Скрипт мониторинга

```
#!/bin/bash
# monitor.sh

echo "=== Cognitum Health Check ==="
echo ""

# API Health
echo "API Health:"
curl -s http://localhost:8080/api/health | jq .
echo ""
```

```
# Container Status
echo "Containers:"
docker compose ps --format "table {{.Name}}\t{{.Status}}"
echo ""

# Resource Usage
echo "Resources:"
docker stats --no-stream --format "table {{.Name}}\t{{.CPUPerc}}\t{{.MemUsage}}"
echo ""

# Disk Usage
echo "Disk:"
docker system df
```

6.3 Просмотр логов

6.3.1 Логи Docker контейнеров

```
# Все логи
docker compose logs

# Конкретный сервис
docker compose logs app
docker compose logs nats
docker compose logs postgres

# В реальном времени
docker compose logs -f app

# Последние N строк
docker compose logs --tail=100 app

# С временными метками
docker compose logs -t app

# За определённый период (Docker 20.10+)
docker compose logs --since="2024-01-15T10:00:00" app
```

6.3.2 Логи агентов через Loki

Агенты отправляют логи в платформу, которые доступны:

1. **В веб-интерфейсе:** Раздел **Logs**
2. **Через Loki API:**

```
# Запрос логов
curl -G -s "http://localhost:3100/loki/api/v1/query_range" \
  --data-urlencode 'query={job="cognitum"}' \
  --data-urlencode 'start=1705312800' \
  --data-urlencode 'end=1705399200' | jq .
```

6.3.3 Уровни логирования

Уровень	Описание
DEBUG	Отладочная информация
INFO	Информационные сообщения
WARNING	Предупреждения
ERROR	Ошибки
CRITICAL	Критические ошибки

6.3.4 Фильтрация логов

```
# Только ошибки
docker compose logs app 2>&1 | grep -i error

# По агенту
docker compose logs app 2>&1 | grep "agent_name=echo"
```

```
# По job_id
docker compose logs app 2>&1 | grep "job_id=550e8400"
```

6.3.5 Логи аудита

```
# Просмотр аудит-лога
docker exec cognitum-app cat /var/log/cognitum/audit.log | jq .

# Фильтрация по событию
docker exec cognitum-app cat /var/log/cognitum/audit.log | \
  jq 'select(.event_type=="user.login")'

# Фильтрация по пользователю
docker exec cognitum-app cat /var/log/cognitum/audit.log | \
  jq 'select(.user_id=="admin")'
```

6.3.6 Ротация логов Docker

Настройте в `/etc/docker/daemon.json`:

```
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m",
    "max-file": "3"
  }
}
```

6.3.7 Экспорт логов

```
# Экспорт в файл
docker compose logs --no-color app > app_logs.txt

# С компрессией
docker compose logs --no-color app | gzip > app_logs.gz
```

6.3.8 Централизованное логирование

Для production рекомендуется отправлять логи во внешнюю систему:

- ELK Stack (Elasticsearch, Logstash, Kibana)
- Grafana Loki
- Splunk
- Graylog

6.4 Резервное копирование

6.4.1 Компоненты для резервного копирования

Компонент	Приоритет	Описание
PostgreSQL	Критичный	Основная БД
Конфигурация (.env)	Критичный	Настройки
Docker volumes	Важный	Данные сервисов

6.4.2 Резервное копирование PostgreSQL

6.4.2.1 Создание дампа

```
# Полный дамп
docker exec cognitum-postgres pg_dump -U aiplatform aiplatform > backup_$(date +%Y%m%d).sql

# Сжатый дамп
docker exec cognitum-postgres pg_dump -U aiplatform -Fc aiplatform > backup_$(date +%Y%m%d).dump

# Только данные (без схемы)
docker exec cognitum-postgres pg_dump -U aiplatform --data-only aiplatform > data_$(date +%Y%m%d).sql
```

6.4.2.2 Восстановление

```
# Из SQL дампа
cat backup.sql | docker exec -i cognitum-postgres psql -U aiplatform aiplatform

# Из сжатого дампа
docker exec -i cognitum-postgres pg_restore -U aiplatform -d aiplatform < backup.dump
```

6.4.3 Резервное копирование конфигурации

```
# Копирование .env
cp /opt/cognitum/config/.env /backup/cognitum/env_$(date +%Y%m%d)

# Копирование всей конфигурации
tar -czf /backup/cognitum/config_$(date +%Y%m%d).tar.gz /opt/cognitum/config/
```

6.4.4 Резервное копирование Docker volumes

```
# Список volumes
docker volume ls | grep cognitum

# Backup volume
docker run --rm -v cognitum_postgres_data:/data -v $(pwd):/backup \
    alpine tar czf /backup/postgres_data.tar.gz -C /data .

# Backup всех volumes
for vol in $(docker volume ls -q | grep cognitum); do
    docker run --rm -v $vol:/data -v $(pwd):/backup \
        alpine tar czf /backup/${vol}.tar.gz -C /data .
done
```

6.4.5 Автоматизация резервного копирования

```
#!/bin/bash
# backup.sh

BACKUP_DIR=/backup/cognitum
DATE=$(date +%Y%m%d_%H%M%S)

mkdir -p $BACKUP_DIR/$DATE

# PostgreSQL
docker exec cognitum-postgres pg_dump -U aiplatform -Fc aiplatform \
    > $BACKUP_DIR/$DATE/postgres.dump

# Конфигурация
cp /opt/cognitum/config/.env $BACKUP_DIR/$DATE/

# Volumes
for vol in postgres_data nats_data; do
    docker run --rm -v cognitum_$vol:/data -v $BACKUP_DIR/$DATE:/backup \
        alpine tar czf /backup/${vol}.tar.gz -C /data .
done

# Очистка старых backup (оставить последние 7)
find $BACKUP_DIR -maxdepth 1 -type d -mtime +7 -exec rm -rf {} \;

echo "Backup completed: $BACKUP_DIR/$DATE"
```

6.4.6 Расписание резервного копирования

```
# crontab -e
# Ежедневно в 2:00
0 2 * * * /opt/cognitum/scripts/backup.sh >> /var/log/cognitum-backup.log 2>&1
```

6.4.7 Хранение резервных копий

Рекомендации:

- Храните копии на отдельном сервере
- Используйте шифрование для конфиденциальных данных
- Проверяйте целостность копий

- Храните минимум 7 ежедневных + 4 еженедельных копии

6.5 Восстановление данных

6.5.1 Восстановление PostgreSQL

6.5.1.1 Из SQL дампа

```
# Остановить приложение
docker compose stop app

# Пересоздать базу
docker exec -it cognitum-postgres psql -U aiplatform -c "DROP DATABASE IF EXISTS aiplatform;"
docker exec -it cognitum-postgres psql -U aiplatform -c "CREATE DATABASE aiplatform;"

# Восстановить данные
cat backup.sql | docker exec -i cognitum-postgres psql -U aiplatform aiplatform

# Запустить приложение
docker compose start app
```

6.5.1.2 Из сжатого дампа

```
docker exec -i cognitum-postgres pg_restore -U aiplatform -d aiplatform --clean < backup.dump
```

6.5.2 Восстановление Docker volumes

```
# Остановить сервисы
docker compose down

# Удалить старый volume
docker volume rm cognitum_postgres_data

# Создать новый volume
docker volume create cognitum_postgres_data

# Восстановить данные
docker run --rm -v cognitum_postgres_data:/data -v $(pwd):/backup \
    alpine tar xzf /backup/postgres_data.tar.gz -C /data

# Запустить сервисы
docker compose up -d
```

6.5.3 Восстановление конфигурации

```
# Восстановить .env
cp /backup/cognitum/20240115/env /opt/cognitum/config/.env

# Перезапустить
docker compose down
docker compose up -d
```

6.5.4 Полное восстановление

```
#!/bin/bash
# restore.sh

BACKUP_DIR=$1  # Путь к директории с backup

if [ -z "$BACKUP_DIR" ]; then
    echo "Usage: restore.sh /path/to/backup"
    exit 1
fi

echo "Stopping services..."
cd /opt/cognitum/config
docker compose down

echo "Restoring configuration..."
cp $BACKUP_DIR/.env /opt/cognitum/config/.env

echo "Restoring PostgreSQL volume..."
docker volume rm cognitum_postgres_data
```

```
docker volume create cognitum_postgres_data
docker run --rm -v cognitum_postgres_data:/data -v $BACKUP_DIR:/backup \
    alpine tar xzf /backup/postgres_data.tar.gz -C /data

echo "Starting services..."
docker compose up -d

echo "Waiting for database..."
sleep 10

echo "Restoring database..."
docker exec -i cognitum-postgres pg_restore -U aiplatform -d aiplatform \
    --clean < $BACKUP_DIR/postgres.dump

echo "Restoration completed!"
```

6.5.5 Проверка восстановления

После восстановления проверьте:

1. Доступность веб-интерфейса
2. Авторизация работает
3. Агенты подключаются
4. Задачи выполняются

```
# Health check
curl http://localhost:8080/api/health

# Проверка данных
docker exec -it cognitum-postgres psql -U aiplatform -c "SELECT COUNT(*) FROM users;"
```

6.5.6 Аварийное восстановление

При полной потере сервера:

1. Установите Docker на новом сервере
2. Скопируйте дистрибутив и backup
3. Выполните полное восстановление
4. Проверьте работоспособность
5. Обновите DNS/IP при необходимости

7 Обновление

В данном разделе описаны процедуры обновления платформы Cognitum.

7.1 Подготовка к обновлению

7.1.1 Перед обновлением

1. **Ознакомьтесь с Release Notes** — изучите изменения в новой версии
2. **Проверьте совместимость** — убедитесь, что агенты совместимы
3. **Создайте резервную копию** — обязательно!
4. **Запланируйте окно обслуживания** — предупредите пользователей

7.1.2 Создание резервной копии

```
# Полный backup перед обновлением
BACKUP_DIR=/backup/cognitum/pre_upgrade_$(date +%Y%m%d_%H%M%S)
mkdir -p $BACKUP_DIR

# PostgreSQL
docker exec cognitum-postgres pg_dump -U aiplatform -Fc aiplatform \
    > $BACKUP_DIR/postgres.dump

# Конфигурация
cp /opt/cognitum/config/.env $BACKUP_DIR/
cp /opt/cognitum/config/docker-compose.yml $BACKUP_DIR/

# Volumes
for vol in postgres_data nats_data loki_data; do
    docker run --rm -v cognitum_$vol:/data -v $BACKUP_DIR:/backup \
        alpine tar czf /backup/${vol}.tar.gz -C /data .
done

echo "Backup created: $BACKUP_DIR"
```

7.1.3 Проверка текущей версии

```
# Версия платформы
curl http://localhost:8080/api/version

# Версии образов
docker images | grep cognitum
```

7.1.4 Скачивание нового дистрибутива

```
# Распаковка в отдельную директорию
unzip cognitum-distro-X.Y.Z.zip -d /opt/cognitum-new
```

7.1.5 Сравнение конфигураций

```
# Сравнение env.example
diff /opt/cognitum/config/env.example /opt/cognitum-new/config/env.example

# Сравнение docker-compose.yml
diff /opt/cognitum/config/docker-compose.yml /opt/cognitum-new/config/docker-compose.yml
```

7.1.6 Тестирование на staging (рекомендуется)

Перед обновлением production:

1. Разверните новую версию на тестовом окружении
2. Восстановите копию данных production
3. Проверьте работоспособность
4. Протестируйте агентов

7.2 Процедура обновления

7.2.1 Стандартное обновление

```
# 1. Остановить текущую версию
cd /opt/cognitum/config
docker compose down

# 2. Загрузить новые образы
cd /opt/cognitum-new/images
./load_images.sh

# 3. Обновить конфигурацию
cd /opt/cognitum/config

# Скопировать новые файлы (сохранив .env)
cp /opt/cognitum-new/config/docker-compose.yml .
cp /opt/cognitum-new/config/deploy.sh .

# Добавить новые переменные из env.example в .env
# (вручную сравнить и добавить)

# 4. Запустить новую версию
docker compose up -d

# 5. Проверить статус
docker compose ps
curl http://localhost:8080/api/health
```

7.2.2 Обновление с миграцией БД

Если новая версия содержит миграции базы данных:

```
# Миграции применяются автоматически при запуске
docker compose up -d

# Проверить логи миграций
docker compose logs app | grep -i migration
```

7.2.3 Обновление агентов

После обновления платформы обновите SDK в агентах:

```
# В директории агента
pip install --upgrade /opt/cognitum/sdk/

# Пересоберите образ агента
docker build -t my-agent:new .

# Обновите образ в docker-compose агента
# IMAGE: my-agent:new

# Перезапустите агента
docker compose up -d --force-recreate
```

7.2.4 Обновление с минимальным простоем

Для минимизации простоя:

```
# 1. Загрузить образы заранее
cd /opt/cognitum-new/images
./load_images.sh

# 2. Быстрое переключение
cd /opt/cognitum/config
docker compose down
cp /opt/cognitum-new/config/docker-compose.yml .
docker compose up -d
```

```
# Простой: ~30 секунд
```

7.2.5 Проверка после обновления

```
# Версия
curl http://localhost:8080/api/version

# Health check
curl http://localhost:8080/api/health

# Агенты подключились
curl http://localhost:8080/api/agents -H "Authorization: Bearer <token>"

# Тестовая задача
curl -X POST http://localhost:8080/api/jobs/invoke \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer <token>" \
    -d '{"job_type": "echo-job", "body": {"test": true}, "timeout": 30}'
```

7.3 Откат изменений

Если после обновления возникли проблемы, выполните откат.

7.3.1 Откат к предыдущей версии

```
# 1. Остановить новую версию
cd /opt/cognitum/config
docker compose down

# 2. Восстановить предыдущую конфигурацию
cp /backup/cognitum/pre_upgrade_*/docker-compose.yml .
cp /backup/cognitum/pre_upgrade_*/.env .

# 3. Загрузить предыдущие образы (если удалены)
docker load < /backup/cognitum/pre_upgrade_*/cognitum-app.tar.gz

# 4. Запустить предыдущую версию
docker compose up -d
```

7.3.2 Откат базы данных

Если новая версия изменила схему БД:

```
# 1. Остановить приложение
docker compose stop app

# 2. Восстановить базу из backup
docker exec -it cognitum-postgres psql -U aiplatform -c "DROP DATABASE IF EXISTS aiplatform;"
docker exec -it cognitum-postgres psql -U aiplatform -c "CREATE DATABASE aiplatform;"
docker exec -i cognitum-postgres pg_restore -U aiplatform -d aiplatform \
    < /backup/cognitum/pre_upgrade_*/postgres.dump

# 3. Запустить с предыдущей версией приложения
docker compose up -d
```

7.3.3 Полный откат

```
#!/bin/bash
# rollback.sh

BACKUP_DIR=/backup/cognitum/pre_upgrade_*  # Указать конкретный backup

echo "=== Starting rollback ==="

# Остановить текущую версию
cd /opt/cognitum/config
docker compose down

# Восстановить конфигурацию
cp $BACKUP_DIR/.env .
```

```
cp $BACKUP_DIR/docker-compose.yml .

# Восстановить volume PostgreSQL
docker volume rm cognitum_postgres_data
docker volume create cognitum_postgres_data
docker run --rm -v cognitum_postgres_data:/data -v $BACKUP_DIR:/backup \
    alpine tar xzf /backup/postgres_data.tar.gz -C /data

# Запустить предыдущую версию
docker compose up -d

# Проверить
sleep 10
curl http://localhost:8080/api/health

echo "=== Rollback completed ==="
```

7.3.4 Частичный откат

Если проблема в конкретном компоненте:

```
# Откат только приложения
docker compose pull app  # предыдущий тег
docker compose up -d --force-recreate app

# Откат только агента
docker compose stop echo-agent
docker compose pull echo-agent
docker compose up -d echo-agent
```

7.3.5 Документирование отката

После отката:

1. Зафиксируйте причину отката
2. Соберите логи проблемной версии
3. Сообщите разработчикам
4. Запланируйте повторное обновление после исправления

8 Разработка и развёртывание агентов

В данном разделе описаны процедуры разработки, контейнеризации и развёртывания AI-агентов.

8.1 Создание агента с SDK

8.1.1 Структура проекта агента

```
my-agent/
├── main.py                 # Точка входа
├── requirements.txt        # Зависимости Python
├── Dockerfile              # Образ контейнера
├── docker-compose.yml      # Конфигурация развёртывания
└── .env                    # Переменные окружения (не коммитить!)
```

8.1.2 Минимальный агент

```python
# main.py
import asyncio
from cognitum_agent import Agent, AgentConfig

agent = Agent(AgentConfig(
    name="my-agent",
    description="Мой первый агент",
    job_schemas=[
        {
            "job_type": "my-task",
            "description": "Обработка задачи"
        }
    ]
))

@agent.job_handler("my-task")
async def handle_task(job):
    await agent.publish_log("info", f"Processing job: {job.job_id}")

    # Ваша логика обработки
    result = job.body.get("input", "") + " processed"

    return {"output": result}

async def main():
    async with agent:
        await agent.publish_log("info", "Agent started")
        # Агент работает до остановки
        await asyncio.Event().wait()

if __name__ == "__main__":
    asyncio.run(main())
```

8.1.3 Зависимости

```
# requirements.txt
# SDK устанавливается из локальной копии
# httpx>=0.24.0  # если нужны HTTP-запросы
# qdrant-client>=1.0.0  # если работаете с Qdrant
```

8.1.4 Агент с LLM

```python
@agent.job_handler("analyze")
async def handle_analyze(job):
    text = job.body["text"]

    # Вызов LLM через платформу
    response = await agent.llm_chat(
        messages=[
            {"role": "system", "content": "Analyze the following text."},
            {"role": "user", "content": text}
        ],
        model="gpt-4o-mini"
    )
```

```
    analysis = response["choices"][0]["message"]["content"]
    return {"analysis": analysis}
```

8.1.5 Агент с настройками

```
agent = Agent(AgentConfig(
    name="configurable-agent",
    settings_schema=[
        {"name": "model", "type": "llm", "llm_type": "llm"},
        {"name": "temperature", "type": "number", "default": 0.7},
        {"name": "storage", "type": "storage", "storage_type": "qdrant"}
    ]
))

@agent.job_handler("process")
async def handle_process(job):
    # Получение настроек
    model = agent.get_setting("model")
    temperature = agent.get_setting("temperature", 0.7)

    response = await agent.llm_chat(
        messages=[...],
        model=model,
        temperature=temperature
    )
    return {"result": response}
```

8.1.6 Локальное тестирование

```
# Установка SDK
pip install -e /opt/cognitum/sdk/

# Установка зависимостей
pip install -r requirements.txt

# Запуск
NATS_URL=nats://localhost:4222 python main.py
```

8.2 Контейнеризация агента

8.2.1 Dockerfile

```
# Dockerfile
FROM python:3.11-slim

# Установка системных зависимостей (если нужны)
RUN apt-get update && apt-get install -y --no-install-recommends \
    && rm -rf /var/lib/apt/lists/*

WORKDIR /app

# Копирование SDK
COPY sdk/ /sdk/
RUN pip install --no-cache-dir /sdk/

# Копирование зависимостей
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Копирование кода агента
COPY . .

# Пользователь без root-прав
RUN useradd -m agent && chown -R agent:agent /app
USER agent

# Запуск
CMD ["python", "main.py"]
```

8.2.2 Оптимизированный Dockerfile

```
# Многоэтапная сборка
FROM python:3.11-slim as builder

WORKDIR /build
COPY sdk/ /sdk/
COPY requirements.txt .
RUN pip wheel --no-cache-dir --wheel-dir /wheels /sdk/
RUN pip wheel --no-cache-dir --wheel-dir /wheels -r requirements.txt

# Финальный образ
FROM python:3.11-slim

WORKDIR /app

COPY --from=builder /wheels /wheels
RUN pip install --no-cache-dir /wheels/*

COPY . .

RUN useradd -m agent && chown -R agent:agent /app
USER agent

CMD ["python", "main.py"]
```

8.2.3 Сборка образа

```
# Копирование SDK в директорию агента
cp -r /opt/cognitum/sdk ./sdk

# Сборка
docker build -t my-agent:latest .

# Проверка
docker images | grep my-agent
```

8.2.4 docker-compose.yml

```
# docker-compose.yml
version: '3.8'

services:
  my-agent:
    image: my-agent:latest
    build: .
    environment:
      - NATS_URL=nats://nats:4222
    networks:
      - cognitum-network
    restart: unless-stopped
    depends_on:
      - nats
    logging:
      driver: json-file
      options:
        max-size: "10m"
        max-file: "3"

networks:
  cognitum-network:
    external: true
```

8.2.5 Переменные окружения

```
# .env для локальной разработки
NATS_URL=nats://localhost:4222
LOG_LEVEL=DEBUG
```

8.2.6 Health check

```
# Добавить в Dockerfile
HEALTHCHECK --interval=30s --timeout=10s --start-period=5s --retries=3 \
```

```
    CMD python -c "import socket; s=socket.socket(); s.connect(('localhost', 8080)); s.close()" || exi
t 1
```

Или в docker-compose:

```
healthcheck:
  test: ["CMD", "python", "-c", "print('healthy')"]
  interval: 30s
  timeout: 10s
  retries: 3
```

8.3 Развёртывание агента

8.3.1 Развёртывание вместе с платформой

Добавьте агента в основной docker-compose:

```
# docker-compose.yml платформы
services:
  # ... существующие сервисы ...

  my-agent:
    image: my-agent:latest
    environment:
      - NATS_URL=nats://nats:4222
    networks:
      - cognitum-network
    restart: unless-stopped
    depends_on:
      nats:
        condition: service_healthy
```

8.3.2 Развёртывание отдельно

```
# Создать директорию агента
mkdir -p /opt/cognitum/agents/my-agent
cd /opt/cognitum/agents/my-agent

# Скопировать файлы агента
cp -r /path/to/my-agent/* .

# Запустить
docker compose up -d
```

8.3.3 Подключение к существующей сети

```
# Убедиться, что сеть существует
docker network ls | grep cognitum

# Если сеть не существует, создать
docker network create cognitum-network
```

8.3.4 Обновление агента

```
# Пересборка образа
docker compose build --no-cache

# Перезапуск с новым образом
docker compose up -d --force-recreate
```

8.3.5 Масштабирование

Для обработки большой нагрузки:

```
services:
  my-agent:
    image: my-agent:latest
    deploy:
      replicas: 3
```

Или через docker compose:

```
docker compose up -d --scale my-agent=3
```

Убедитесь, что агент поддерживает параллельную работу.

8.3.6 Развёртывание на другом сервере

```
# 1. Экспорт образа
docker save my-agent:latest | gzip > my-agent.tar.gz

# 2. Копирование на сервер
scp my-agent.tar.gz server:/opt/cognitum/agents/

# 3. На целевом сервере
gunzip -c my-agent.tar.gz | docker load

# 4. Настройка и запуск
cd /opt/cognitum/agents/my-agent
# Отредактировать NATS_URL в .env или docker-compose.yml
docker compose up -d
```

8.3.7 Проверка развёртывания

```
# Статус контейнера
docker compose ps

# Логи
docker compose logs -f

# Проверка регистрации в платформе
curl http://localhost:8080/api/agents -H "Authorization: Bearer <token>"
```

8.4 Мониторинг агента

8.4.1 Статус в веб-интерфейсе

1. Перейдите в **Agents**
2. Найдите агента в списке
3. Проверьте статус:
   - **Online** — агент активен
   - **Offline** — агент недоступен

8.4.2 Логи агента

8.4.2.1 В веб-интерфейсе

1. Перейдите в **Logs**
2. Выберите агента в фильтре
3. Просмотрите логи

8.4.2.2 Через Docker

```
# Логи контейнера
docker compose logs -f my-agent

# Последние 100 строк
docker compose logs --tail=100 my-agent
```

8.4.3 Метрики агента

Откройте карточку агента в веб-интерфейсе для просмотра:

- Количество обработанных задач
- Количество LLM-вызовов
- Время работы (uptime)

8.4.4 Проверка подключения к NATS

```
# В контейнере агента
docker exec my-agent python -c "
import asyncio
import nats

async def check():
    nc = await nats.connect('nats://nats:4222')
    print('Connected to NATS')
    await nc.close()

asyncio.run(check())
"
```

8.4.5 Healthcheck агента

```
# Если настроен healthcheck
docker inspect --format='{{.State.Health.Status}}' my-agent-container
```

8.4.6 Диагностика проблем

8.4.6.1 Агент не регистрируется

1. Проверьте подключение к NATS
2. Проверьте логи агента
3. Убедитесь, что имя агента уникально

```
docker compose logs my-agent | grep -i error
```

8.4.6.2 Задачи не обрабатываются

1. Проверьте, что агент online
2. Проверьте, что job_type зарегистрирован
3. Проверьте логи на ошибки

```
# Проверка регистрации типов задач
curl http://localhost:8080/api/agents/<agent_name> \
    -H "Authorization: Bearer <token>" | jq .job_types
```

8.4.6.3 Высокое потребление ресурсов

```
# Статистика контейнера
docker stats my-agent

# Если память утекает, перезапустите
docker compose restart my-agent
```

8.4.7 Алерты

Настройте оповещения при:

- Агент перешёл в offline
- Высокий процент ошибок задач
- Контейнер перезапустился

9 Устранение неисправностей

В данном разделе описаны типичные проблемы и способы их решения.

9.1 Типичные проблемы и решения

9.1.1 Платформа не запускается

Симптомы: Контейнеры не стартуют или падают.

Решение:

```
# Проверить логи
docker compose logs

# Проверить статус
docker compose ps -a

# Типичные причины:
# 1. Порты заняты
lsof -i :8080
lsof -i :5432

# 2. Недостаточно ресурсов
docker system df
free -h

# 3. Ошибки в конфигурации
docker compose config
```

9.1.2 Веб-интерфейс недоступен

Симптомы: Браузер показывает ошибку подключения.

Решение:

```
# Проверить, запущен ли контейнер app
docker compose ps app

# Проверить порт
curl http://localhost:8080/api/health

# Проверить firewall
sudo ufw status
```

9.1.3 Ошибка аутентификации

Симптомы: “Invalid credentials” при входе.

Решение:

1. Проверьте правильность логина/пароля
2. Сбросьте пароль admin через БД:

```
docker exec -it cognitum-postgres psql -U aiplatform -c "
UPDATE users SET password_hash = '\$2b\$12\$...' WHERE username = 'admin';
"
```

3. Если используется LDAP, проверьте настройки:

```
docker compose logs app | grep -i ldap
```

9.1.4 Агенты не подключаются

Симптомы: Агенты в статусе Offline.

Решение:

```
# Проверить NATS
docker compose logs nats

# Проверить подключение агента
docker compose logs echo-agent | grep -i nats

# Проверить сеть
docker network inspect cognitum-network
```

9.1.5 Задачи зависают

Симптомы: Задачи остаются в статусе “running” или “queued”.

Решение:

```
# Проверить агента
docker compose logs my-agent

# Проверить очередь
curl http://localhost:8222/jsz?streams=true

# Перезапустить агента
docker compose restart my-agent
```

9.1.6 Ошибки LLM

Симптомы: Ошибки при обращении к моделям.

Решение:

1. Проверьте настройки провайдера в UI
2. Проверьте API-ключи
3. Проверьте доступность API:

```
# OpenAI
curl https://api.openai.com/v1/models \
    -H "Authorization: Bearer $OPENAI_API_KEY"

# Ollama
curl http://ollama:11434/api/tags
```

9.1.7 Диск заполнен

Симптомы: Ошибки записи, контейнеры падают.

Решение:

```
# Проверить использование диска
df -h
docker system df

# Очистка Docker
docker system prune -a

# Очистка логов
truncate -s 0 /var/lib/docker/containers/*/*-json.log

# Очистка старых образов
docker image prune -a
```

9.2 Диагностика NATS

9.2.1 Проверка состояния NATS

```
# Статус контейнера
docker compose ps nats

# Логи
docker compose logs nats
```

```
# Информация о сервере
curl http://localhost:8222/varz | jq .
```

9.2.2 Проверка JetStream

```
# JetStream информация
curl http://localhost:8222/jsz | jq .

# Стримы
curl http://localhost:8222/jsz?streams=true | jq .

# Consumers
curl http://localhost:8222/jsz?consumers=true | jq .
```

9.2.3 Проверка подключений

```
# Активные подключения
curl http://localhost:8222/connz | jq .

# Подробная информация
curl "http://localhost:8222/connz?subs=true" | jq .
```

9.2.4 Проверка подписок

```
# Все подписки
curl http://localhost:8222/subsz | jq .
```

9.2.5 Типичные проблемы NATS

9.2.5.1 Стримы не созданы

```
# Проверить стримы
curl http://localhost:8222/jsz?streams=true | jq '.streams[].name'

# Если пусто, перезапустить app (стримы создаются при старте)
docker compose restart app
```

9.2.5.2 Consumer отстал

```
# Информация о consumer
curl "http://localhost:8222/jsz?consumers=true" | jq '.streams[].consumers[]'

# Если num_pending большой, consumer не успевает обрабатывать
```

9.2.5.3 Слишком много сообщений в очереди

```
# Очистка стрима (ОСТОРОЖНО!)
docker exec cognitum-nats nats stream purge JOBS --force
```

9.2.6 NATS CLI

```
# Войти в контейнер
docker exec -it cognitum-nats sh

# Список стримов
nats stream list

# Информация о стриме
nats stream info JOBS

# Публикация тестового сообщения
nats pub test.subject "Hello"
```

9.2.7 Мониторинг NATS

Веб-интерфейс мониторинга: `http://localhost:8222`

Полезные endpoints: - `/varz` — общая информация - `/connz` — подключения - `/subsz` — подписки - `/jsz` — JetStream

9.3 Диагностика базы данных

9.3.1 PostgreSQL

9.3.1.1 Проверка состояния

```
# Статус контейнера
docker compose ps postgres

# Логи
docker compose logs postgres

# Подключение
docker exec -it cognitum-postgres psql -U aiplatform -d aiplatform
```

9.3.1.2 Проверка соединений

```
-- Активные соединения
SELECT * FROM pg_stat_activity WHERE datname = 'aiplatform';

-- Количество соединений
SELECT count(*) FROM pg_stat_activity WHERE datname = 'aiplatform';

-- Максимум соединений
SHOW max_connections;
```

9.3.1.3 Проверка размера БД

```
-- Размер базы
SELECT pg_size_pretty(pg_database_size('aiplatform'));

-- Размер таблиц
SELECT
    tablename,
    pg_size_pretty(pg_total_relation_size(tablename::regclass)) as size
FROM pg_tables
WHERE schemaname = 'public'
ORDER BY pg_total_relation_size(tablename::regclass) DESC;
```

9.3.1.4 Проверка блокировок

```
-- Активные блокировки
SELECT * FROM pg_locks WHERE granted = false;

-- Заблокированные запросы
SELECT
    pid,
    now() - pg_stat_activity.query_start AS duration,
    query,
    state
FROM pg_stat_activity
WHERE (now() - pg_stat_activity.query_start) > interval '5 minutes';
```

9.3.1.5 Очистка

```
-- Vacuum (освобождение места)
VACUUM VERBOSE;

-- Analyze (обновление статистики)
ANALYZE;

-- Полная очистка (блокирует таблицу!)
VACUUM FULL;
```

9.3.2 Qdrant

9.3.2.1 Проверка состояния

```
# Статус
curl http://localhost:6333/

# Коллекции
curl http://localhost:6333/collections | jq .
```

9.3.3 Neo4j

9.3.3.1 Проверка состояния

```
# Веб-интерфейс
# http://localhost:7474

# Через cypher-shell
docker exec -it cognitum-neo4j cypher-shell -u neo4j -p $NEO4J_PASSWORD

# Статус БД
CALL dbms.listDatabases();
```

9.3.4 Типичные проблемы БД

9.3.4.1 Соединение отклонено

```
# Проверить, запущен ли контейнер
docker compose ps postgres

# Проверить логи
docker compose logs postgres

# Проверить порт
netstat -tlnp | grep 5432
```

9.3.4.2 Слишком много соединений

```
-- Проверить лимит
SHOW max_connections;

-- Закрыть idle соединения
SELECT pg_terminate_backend(pid)
FROM pg_stat_activity
WHERE state = 'idle' AND query_start < now() - interval '1 hour';
```

9.3.4.3 Медленные запросы

```
-- Включить логирование медленных запросов
ALTER SYSTEM SET log_min_duration_statement = 1000;
SELECT pg_reload_conf();
```

9.4 Работа с логами (Loki)

9.4.1 Проверка состояния Loki

```
# Статус контейнера
docker compose ps loki

# Логи
docker compose logs loki

# Ready check
curl http://localhost:3100/ready
```

9.4.2 Запросы к Loki API

```
# Метки (labels)
curl http://localhost:3100/loki/api/v1/labels | jq .

# Значения метки
curl http://localhost:3100/loki/api/v1/label/job/values | jq .

# Запрос логов
curl -G http://localhost:3100/loki/api/v1/query_range \
    --data-urlencode 'query={job="cognitum"}' \
    --data-urlencode 'limit=100' | jq .
```

9.4.3 LogQL запросы

```
# Все логи контейнера
{container_name="cognitum-app"}

# Логи с ошибками
```

```
{container_name="cognitum-app"} |= "error"

# Логи агента
{container_name=~"cognitum-.*-agent"}

# Логи за последний час
{job="cognitum"} [1h]

# Фильтрация по JSON полю
{job="cognitum"} | json | level="error"
```

9.4.4 Promtail

Promtail собирает логи Docker и отправляет в Loki.

```
# Статус
docker compose ps promtail

# Логи
docker compose logs promtail

# Конфигурация
cat promtail-config.yml
```

9.4.5 Типичные проблемы

9.4.5.1 Логи не поступают

```
# Проверить Promtail
docker compose logs promtail | grep -i error

# Проверить доступ к Docker socket
docker exec cognitum-promtail ls -la /var/run/docker.sock

# Проверить конфигурацию
docker exec cognitum-promtail cat /etc/promtail/config.yml
```

9.4.5.2 Loki не отвечает

```
# Перезапустить
docker compose restart loki

# Проверить диск
df -h
```

9.4.5.3 Слишком много логов

```
# Настройка retention в loki-config.yml
compactor:
  retention_enabled: true
  retention_delete_delay: 2h
  retention_delete_worker_count: 150

limits_config:
  retention_period: 168h  # 7 дней
```

9.4.6 Очистка логов

```
# Loki хранит данные в volumes
# Для очистки удалите volume (ОСТОРОЖНО!)

docker compose down
docker volume rm cognitum_loki_data
docker compose up -d
```

9.4.7 Экспорт логов

```
# Через API
curl -G http://localhost:3100/loki/api/v1/query_range \
    --data-urlencode 'query={job="cognitum"}' \
    --data-urlencode 'start=1705312800000000000' \
    --data-urlencode 'end=1705399200000000000' \
    --data-urlencode 'limit=10000' > logs_export.json
```

10 Приложения

Справочная информация для администраторов.

10.1 Справочник переменных окружения

Полный список переменных окружения см. в разделе “Справочник переменных окружения” Руководства пользователя.

10.1.1 Краткая справка

Категория	Переменные
База данных	POSTGRES_USER, POSTGRES_PASSWORD, POSTGRES_DB, POSTGRES_PORT
NATS	NATS_PORT, NATS_MONITORING_PORT
Приложение	APP_ENV, APP_SECRET_KEY, FRONTEND_PORT
JWT	JWT_SECRET_KEY, JWT_EXPIRATION_HOURS
LDAP	LDAP_ENABLED, LDAP_SERVER, LDAP_BASE_DN, etc.
SMTP	SMTP_HOST, SMTP_PORT, SMTP_USER, SMTP_PASSWORD, etc.
Аудит	AUDIT_ENABLED, AUDIT_FILE_PATH, AUDIT_RETENTION_DAYS
Дополнительные БД	QDRANT_PORT, NEO4J_*, LOKI_PORT

10.1.2 Обязательные для изменения в production

```
POSTGRES_PASSWORD=<secure_password>
NEO4J_PASSWORD=<secure_password>
APP_SECRET_KEY=<random_32_hex>
JWT_SECRET_KEY=<random_32_hex>
```

10.2 Структура docker-compose.yml

10.2.1 Сервисы

```yaml
services:
  # Основное приложение
  app:
    image: ${APP_IMAGE:-cognitum-app:latest}
    ports:
      - "${FRONTEND_PORT:-8080}:8080"
    depends_on:
      postgres:
        condition: service_healthy
      nats:
        condition: service_healthy
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8000/api/health"]
      interval: 30s
      timeout: 10s
      retries: 3

  # База данных
  postgres:
    image: postgres:15
    volumes:
      - postgres_data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${POSTGRES_USER}"]

  # Брокер сообщений
  nats:
    image: nats:2.10-alpine
    command: ["--config", "/etc/nats/nats-config.conf"]
    volumes:
      - nats_data:/data
    healthcheck:
```

```
      test: ["CMD", "nats-server", "--signal", "ldm"]

  # Логирование
  loki:
    image: grafana/loki:2.9.0
    volumes:
      - loki_data:/loki

  promtail:
    image: grafana/promtail:2.9.0
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro

  # Векторная БД
  qdrant:
    image: qdrant/qdrant:latest
    volumes:
      - qdrant_data:/qdrant/storage

  # Графовая БД
  neo4j:
    image: neo4j:5
    volumes:
      - neo4j_data:/data

  # Пример агента
  echo-agent:
    image: ${AGENT_IMAGE:-cognitum-echo-agent:latest}
    depends_on:
      - nats
```

10.2.2 Volumes

```
volumes:
  postgres_data:
  nats_data:
  loki_data:
  qdrant_data:
  neo4j_data:
  audit_logs:
```

10.2.3 Сети

```
networks:
  default:
    name: cognitum-network
```

10.2.4 Переопределение конфигурации

Создайте `docker-compose.override.yml` для локальных изменений:

```
# docker-compose.override.yml
services:
  app:
    environment:
      - DEBUG=true
    ports:
      - "8081:8080"
```

10.3 Команды управления

10.3.1 deploy.sh

```
# Запуск
./deploy.sh up -d

# Остановка
./deploy.sh down

# Перезапуск
./deploy.sh restart
```

```
# Статус
./deploy.sh ps

# Логи
./deploy.sh logs

# Сборка образов
./deploy.sh build

# Обновление образов
./deploy.sh pull
```

10.3.2 docker compose

```
# Запуск
docker compose up -d

# Остановка
docker compose down

# Остановка с удалением volumes
docker compose down -v

# Перезапуск сервиса
docker compose restart app

# Пересоздание контейнера
docker compose up -d --force-recreate app

# Логи в реальном времени
docker compose logs -f app

# Статус
docker compose ps

# Масштабирование
docker compose up -d --scale echo-agent=3
```

10.3.3 docker

```
# Выполнить команду в контейнере
docker exec -it cognitum-app bash
docker exec -it cognitum-postgres psql -U aiplatform

# Копирование файлов
docker cp cognitum-app:/app/config.json ./
docker cp ./file.txt cognitum-app:/app/

# Статистика ресурсов
docker stats

# Логи конкретного контейнера
docker logs -f cognitum-app

# Информация о контейнере
docker inspect cognitum-app
```

10.3.4 PostgreSQL

```
# Подключение
docker exec -it cognitum-postgres psql -U aiplatform -d aiplatform

# Выполнить SQL
docker exec cognitum-postgres psql -U aiplatform -c "SELECT 1;"

# Дамп
docker exec cognitum-postgres pg_dump -U aiplatform aiplatform > backup.sql

# Восстановление
cat backup.sql | docker exec -i cognitum-postgres psql -U aiplatform
```

10.3.5 NATS

```
# Подключение к NATS CLI
docker exec -it cognitum-nats nats

# Информация о стримах
curl http://localhost:8222/jsz | jq .

# Публикация сообщения
docker exec cognitum-nats nats pub test.subject "Hello"
```

10.3.6 Системные

```
# Очистка Docker
docker system prune -a

# Размер Docker
docker system df

# Список volumes
docker volume ls

# Удаление volume
docker volume rm cognitum_postgres_data
```