ОПИСАНИЕ ЖИЗНЕННОГО ЦИКЛА

Cognitum AI Platform
Версия 0.4.4

ООО «ИТ-Экспертиза»

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

Оглавление

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

2.1 Назначение документа

Настоящий документ описывает жизненный цикл платформы Cognitum AI Platform и разрабатываемых на её основе AI-агентов.

Документ предназначен для:

- Архитекторов и DevOps-инженеров, проектирующих процессы CI/CD
- Разработчиков агентов, внедряющих решения в production
- Руководителей проектов, планирующих развёртывание платформы

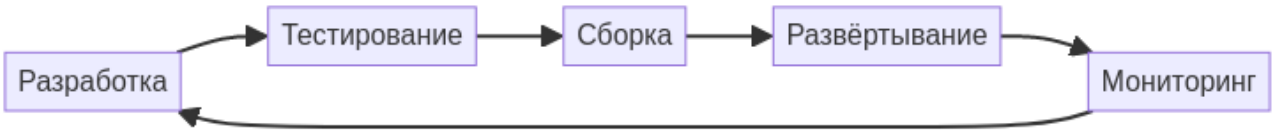
2.2 Область применения

Документ охватывает следующие этапы жизненного цикла:

1. **Разработка агентов** — создание, тестирование, упаковка
2. **Развёртывание** — контейнеризация, оркестрация, обновление
3. **Эксплуатация** — мониторинг, масштабирование, обслуживание
4. **Развитие** — новые возможности, интеграции, оптимизация

2.3 Модель жизненного цикла

Cognitum AI Platform использует итеративную модель разработки:



2.3.1 Ключевые принципы

1. **Контейнеризация** — все компоненты поставляются как Docker-образы
2. **Декларативная конфигурация** — настройки через YAML и переменные окружения
3. **Автоматизация** — CI/CD пайплайны для сборки и развёртывания
4. **Observability** — централизованные логи и метрики

2.4 Связь с другими документами

- **Руководство пользователя** — функциональные возможности платформы
- **Инструкция по эксплуатации** — процедуры установки и обслуживания

3 Разработка агентов

3.1 Обзор процесса разработки

Разработка AI-агента включает следующие этапы:

1. Проектирование функциональности
2. Реализация с использованием SDK
3. Локальное тестирование
4. Интеграционное тестирование
5. Подготовка к развёртыванию

3.2 SDK для разработки агентов

3.2.1 Установка SDK

```
# Из дистрибутива
pip install ./sdk/

# Или из репозитория (при наличии доступа)
pip install git+https://gitlab.company.com/cognitum/sdk.git
```

3.2.2 Структура агента

```
from cognitum_agent import Agent, AgentConfig

# Конфигурация агента
config = AgentConfig(
    name="my-agent",
    description="Описание агента",
    job_schemas=[
        {"job_type": "task-type", "description": "Описание задачи"}
    ],
    settings_schema=[
        {"name": "model", "type": "llm", "llm_type": "llm"}
    ]
)

# Создание агента
agent = Agent(config)

# Обработчик задач
@agent.job_handler("task-type")
async def handle_task(job):
    result = await process(job.body)
    return {"output": result}
```

3.2.3 Типы обработчиков

Тип	Декоратор	Назначение
Job Handler	`@agent.job_handler()`	Обработка асинхронных задач
Chat Handler	`@agent.chat_handler()`	Обработка chat-сообщений

3.3 Локальное тестирование

3.3.1 Подготовка окружения

```
# Создание виртуального окружения
python -m venv venv
source venv/bin/activate

# Установка зависимостей
pip install -r requirements.txt

# Переменные окружения
export NATS_URL=nats://localhost:4222
```

3.3.2 Запуск тестов

```
# Unit-тесты
pytest tests/

# Тест обработчика без платформы
python -c "
from main import agent

# Эмуляция задачи
result = agent.handlers['task-type']({'input': 'test'})
print(result)
"
```

3.4 Интеграционное тестирование

3.4.1 С локальной платформой

1. Запустите платформу:

```
cd /opt/cognitum/config
./deploy.sh up -d
```

2. Запустите агента:

```
python main.py
```

3. Отправьте тестовую задачу:

```
curl -X POST http://localhost:8080/api/jobs/invoke \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer <token>" \
    -d '{"job_type": "task-type", "body": {"input": "test"}, "timeout": 30}'
```

3.4.2 Автоматические тесты

```
# tests/test_integration.py
import pytest
import asyncio
from cognitum_agent import Agent, AgentConfig

@pytest.mark.asyncio
async def test_job_handler():
    agent = Agent(AgentConfig(name="test-agent"))

    @agent.job_handler("test")
    async def handler(job):
        return {"result": "ok"}

    # Эмуляция вызова
    result = await handler(MockJob({"input": "test"}))
    assert result["result"] == "ok"
```

4 Контейнеризация

4.1 Принципы контейнеризации

Все агенты поставляются как Docker-образы для обеспечения:

- Воспроизводимости окружения
- Изоляции зависимостей
- Простоты развёртывания
- Масштабирования

4.2 Создание Docker-образа

4.2.1 Базовый Dockerfile

```
FROM python:3.11-slim

WORKDIR /app

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

# Зависимости приложения
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Код агента
COPY . .

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

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

4.2.2 Оптимизированный Dockerfile

```
# Этап сборки
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

# Установка собранных пакетов
COPY --from=builder /wheels /wheels
RUN pip install --no-cache-dir /wheels/* && rm -rf /wheels

COPY . .

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

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

4.2.3 Best Practices

1. **Минимальный базовый образ —** используйте `-slim` варианты

2. **Многоэтапная сборка** — отделяйте build-time зависимости
3. **Кэширование слоёв** — сначала COPY requirements.txt
4. **Непривилегированный пользователь** — не запускайте от root
5. **Health check** — добавляйте проверку здоровья

4.3 Реестр образов

4.3.1 Локальный реестр

```
# Запуск локального registry
docker run -d -p 5000:5000 --name registry registry:2

# Тегирование образа
docker tag my-agent:latest localhost:5000/my-agent:latest

# Push
docker push localhost:5000/my-agent:latest
```

4.3.2 Корпоративный реестр

```
# Авторизация
docker login nexus.company.com

# Тегирование
docker tag my-agent:latest nexus.company.com/cognitum/my-agent:1.0.0

# Push
docker push nexus.company.com/cognitum/my-agent:1.0.0
```

4.4 Версионирование образов

4.4.1 Стратегия тегирования

Тег	Назначение
`latest`	Последняя стабильная версия
`1.0.0`	Конкретная версия (SemVer)
`1.0.0-rc1`	Release candidate
`dev`	Версия для разработки
`sha-abc123`	Конкретный commit

4.4.2 Пример workflow

```
# При релизе
docker build -t my-agent:1.0.0 .
docker tag my-agent:1.0.0 my-agent:latest
docker push my-agent:1.0.0
docker push my-agent:latest

# При разработке
docker build -t my-agent:dev .
docker push my-agent:dev
```

4.5 Безопасность образов

4.5.1 Сканирование уязвимостей

```
# Trivy
trivy image my-agent:latest

# Docker Scout
docker scout cves my-agent:latest
```

4.5.2 Подпись образов

```
# Cosign
cosign sign --key cosign.key my-agent:latest
cosign verify --key cosign.pub my-agent:latest
```

5 CI/CD пайплайны

5.1 Обзор CI/CD

Непрерывная интеграция и доставка обеспечивают:

- Автоматическую сборку при изменениях кода
- Запуск тестов
- Сборку Docker-образов
- Развёртывание на целевые окружения

5.2 GitLab CI/CD

5.2.1 Базовый pipeline

```yaml
# .gitlab-ci.yml
stages:
  - test
  - build
  - deploy

variables:
  DOCKER_IMAGE: $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA

test:
  stage: test
  image: python:3.11
  script:
    - pip install -r requirements.txt
    - pip install pytest
    - pytest tests/

build:
  stage: build
  image: docker:24
  services:
    - docker:dind
  script:
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
    - docker build -t $DOCKER_IMAGE .
    - docker push $DOCKER_IMAGE
    - docker tag $DOCKER_IMAGE $CI_REGISTRY_IMAGE:latest
    - docker push $CI_REGISTRY_IMAGE:latest
  only:
    - main

deploy:
  stage: deploy
  script:
    - ssh deploy@server "cd /opt/cognitum/agents/my-agent && docker compose pull && docker compose up
-d"
  only:
    - main
  when: manual
```

5.2.2 Pipeline с окружениями

```yaml
stages:
  - test
  - build
  - deploy-staging
  - deploy-production

deploy-staging:
  stage: deploy-staging
  environment:
    name: staging
    url: https://staging.cognitum.company.com
```

```
  script:
    - ./deploy.sh staging
  only:
    - main

deploy-production:
  stage: deploy-production
  environment:
    name: production
    url: https://cognitum.company.com
  script:
    - ./deploy.sh production
  only:
    - tags
  when: manual
```

5.3 GitHub Actions

```
# .github/workflows/ci.yml
name: CI/CD

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v4
        with:
          python-version: '3.11'
      - run: pip install -r requirements.txt
      - run: pytest tests/

  build:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}
      - uses: docker/build-push-action@v5
        with:
          push: true
          tags: ghcr.io/${{ github.repository }}:${{ github.sha }}
```

5.4 Развёртывание

5.4.1 Скрипт развёртывания

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

ENV=$1

if [ "$ENV" == "staging" ]; then
    SERVER=staging.company.com
elif [ "$ENV" == "production" ]; then
    SERVER=prod.company.com
else
    echo "Usage: deploy.sh <staging|production>"
    exit 1
fi
```

```
ssh deploy@$SERVER << 'EOF'
cd /opt/cognitum/agents/my-agent
docker compose pull
docker compose up -d --force-recreate
docker system prune -f
EOF
```

5.4.2 Rollback

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

PREVIOUS_TAG=$1

ssh deploy@server << EOF
cd /opt/cognitum/agents/my-agent
docker compose down
docker pull my-agent:$PREVIOUS_TAG
docker tag my-agent:$PREVIOUS_TAG my-agent:latest
docker compose up -d
EOF
```

5.5 Мониторинг pipeline

- Настройте уведомления в Slack/Teams
- Используйте badges в README
- Храните артефакты (логи, coverage reports)
- Анализируйте время выполнения pipeline

6 Оркестрация

6.1 Docker Compose

6.1.1 Базовая конфигурация

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

services:
  my-agent:
    image: my-agent:latest
    environment:
      - NATS_URL=nats://nats:4222
    networks:
      - cognitum-network
    restart: unless-stopped
    deploy:
      resources:
        limits:
          cpus: '1'
          memory: 1G

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

6.1.2 Масштабирование

```
# Запуск нескольких экземпляров
docker compose up -d --scale my-agent=3
```

6.1.3 Health checks

```yaml
services:
  my-agent:
    healthcheck:
      test: ["CMD", "python", "-c", "print('ok')"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 10s
```

6.2 Kubernetes (перспектива)

6.2.1 Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-agent
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-agent
  template:
    metadata:
      labels:
        app: my-agent
    spec:
      containers:
      - name: my-agent
        image: registry.company.com/cognitum/my-agent:1.0.0
        env:
        - name: NATS_URL
          value: nats://nats:4222
        resources:
          limits:
            cpu: "1"
            memory: 1Gi
```

```
          requests:
            cpu: "500m"
            memory: 512Mi
        livenessProbe:
          exec:
            command: ["python", "-c", "print('ok')"]
          initialDelaySeconds: 10
          periodSeconds: 30
```

6.2.2 Horizontal Pod Autoscaler

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: my-agent-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-agent
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
```

6.3 Стратегии развёртывания

6.3.1 Rolling Update

```
spec:
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
```

6.3.2 Blue-Green Deployment

1. Разверните новую версию параллельно
2. Переключите трафик
3. Удалите старую версию

6.3.3 Canary Deployment

1. Разверните новую версию на часть реплик
2. Мониторьте метрики
3. Постепенно увеличивайте процент

6.4 Управление секретами

6.4.1 Docker Secrets

```
services:
  app:
    secrets:
      - api_key

secrets:
  api_key:
    file: ./secrets/api_key.txt
```

6.4.2 Kubernetes Secrets

```
apiVersion: v1
kind: Secret
metadata:
```

```
  name: my-agent-secrets
type: Opaque
data:
  api-key: <base64-encoded-value>
```

7 Мониторинг и обслуживание

7.1 Уровни мониторинга

7.1.1 Инфраструктурный уровень

- Использование CPU, RAM, диска
- Сетевой трафик
- Состояние контейнеров

7.1.2 Прикладной уровень

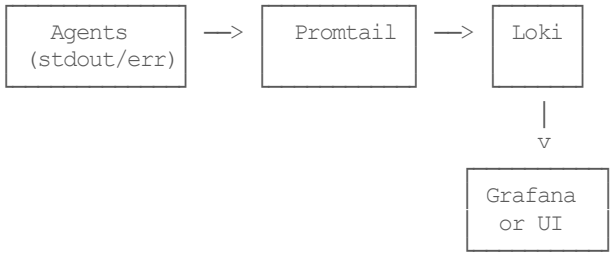
- Статус агентов (online/offline)
- Количество обработанных задач
- Время обработки задач
- Количество ошибок

7.1.3 Бизнес-уровень

- Количество LLM-вызовов
- Стоимость API-запросов
- SLA метрики

7.2 Централизованное логирование

7.2.1 Архитектура



7.2.2 Структурированные логи

```
# В агенте
await agent.publish_log("info", "Task processed", {
    "job_id": job.job_id,
    "duration_ms": duration,
    "tokens_used": tokens
})
```

7.3 Метрики агентов

7.3.1 Встроенные метрики

Метрика	Описание
`jobs_processed`	Количество обработанных задач
`jobs_failed`	Количество ошибок
`llm_calls`	Количество LLM-вызовов
`uptime_seconds`	Время работы агента

7.3.2 Просмотр метрик

1. Веб-интерфейс → Agents → Карточка агента
2. API: `GET /api/agents/{name}/metrics`

7.4 Алертинг

7.4.1 Критические алерты

- Агент перешёл в offline
- Контейнер перезапустился > 3 раз
- Очередь задач переполнена
- Ошибки > 10% от общего числа задач

7.4.2 Предупреждения

- Высокая нагрузка CPU/RAM
- Большое время обработки задач
- Приближение к лимитам API

7.5 Обслуживание

7.5.1 Регулярные процедуры

Периодичность	Процедура
Ежедневно	Проверка health status
Еженедельно	Анализ логов на ошибки
Ежемесячно	Ротация логов, очистка
Квартально	Обновление зависимостей

7.5.2 Ротация логов

```yaml
# docker-compose.yml
logging:
  driver: json-file
  options:
    max-size: "100m"
    max-file: "3"
```

7.5.3 Очистка ресурсов

```
# Удаление неиспользуемых образов
docker image prune -a

# Очистка volumes
docker volume prune

# Очистка всего
docker system prune -a
```

7.6 Масштабирование

7.6.1 Вертикальное масштабирование

Увеличение ресурсов контейнера:

```yaml
deploy:
  resources:
    limits:
      cpus: '4'
      memory: 8G
```

7.6.2 Горизонтальное масштабирование

Увеличение количества экземпляров:

```
docker compose up -d --scale my-agent=5
```

7.6.3 Критерии масштабирования

- CPU > 70% на протяжении 5 минут
- Очередь задач > 100 pending
- Среднее время обработки > целевого

8 Перспективы развития

8.1 Текущий статус платформы

Cognitum AI Platform версии 0.4.x предоставляет:

- Управление AI-агентами
- Систему задач (Jobs)
- Интеграцию с LLM-провайдерами
- Централизованное логирование
- Базовую работу с хранилищами

8.2 Планируемые возможности

8.2.1 RAG (Retrieval-Augmented Generation)

Система дополнения контекста LLM из базы знаний:

- Автоматическое разбиение документов на chunks
- Векторизация и хранение в Qdrant
- Семантический поиск релевантного контекста
- Интеграция с SDK агентов

Применение: - Корпоративные чат-боты - Поиск по документации - Q&A системы

8.2.2 Fine-Tuning

Дообучение моделей на специфических данных:

- Управление датасетами
- Запуск процессов fine-tuning
- Версионирование моделей
- Сравнение качества

Применение: - Специализированные агенты - Доменная адаптация - Улучшение качества ответов

8.2.3 Маркетплейс агентов

Каталог готовых агентов:

- Публикация и поиск агентов
- Версионирование
- Документация и примеры
- Лицензирование

Применение: - Повторное использование решений - Экономия времени разработки - Стандартизация подходов

8.2.4 Chatbot Integration

Интеграция агентов с мессенджерами:

- Telegram
- Slack
- Microsoft Teams
- Веб-виджет

Применение: - Пользовательские интерфейсы - Автоматизация поддержки - Внутренние инструменты

8.3 Улучшения SDK

8.3.1 Планируемые возможности

- Middleware для обработчиков
- Dependency Injection
- Встроенное кэширование
- Structured outputs

8.3.2 Пример будущего API

```python
from cognitum_agent import Agent, cache, depends

@agent.job_handler("analyze")
@cache(ttl=3600)
async def analyze(job, llm=depends(LLMClient)):
    response = await llm.chat(messages=[...])
    return response.structured(AnalysisResult)
```

8.4 Интеграции

8.4.1 Внешние системы

- **1С** — интеграция через HTTP-сервисы
- **SAP** — коннекторы к RFC
- **Bitrix24** — webhook и REST API
- **Jira/Confluence** — автоматизация задач

8.4.2 Источники данных

- **SQL базы** — PostgreSQL, MySQL, Oracle
- **NoSQL** — MongoDB, Elasticsearch
- **Файловые хранилища** — S3, MinIO

8.5 Roadmap

Версия	Основные изменения
0.5.0	RAG базовая реализация
0.6.0	Fine-Tuning управление
0.7.0	Chatbot интеграции
1.0.0	Стабильный релиз

8.6 Участие в развитии

Для предложений по развитию платформы:

1. Создайте Issue в GitLab
2. Опишите use case
3. Предложите реализацию
4. Участвуйте в обсуждении