

NovaData LMS: инструкция по развертыванию

Область: `lms-novadata` , `SQLChecker` , `code_checker`

Документ описывает базовое развертывание трех частей проекта:

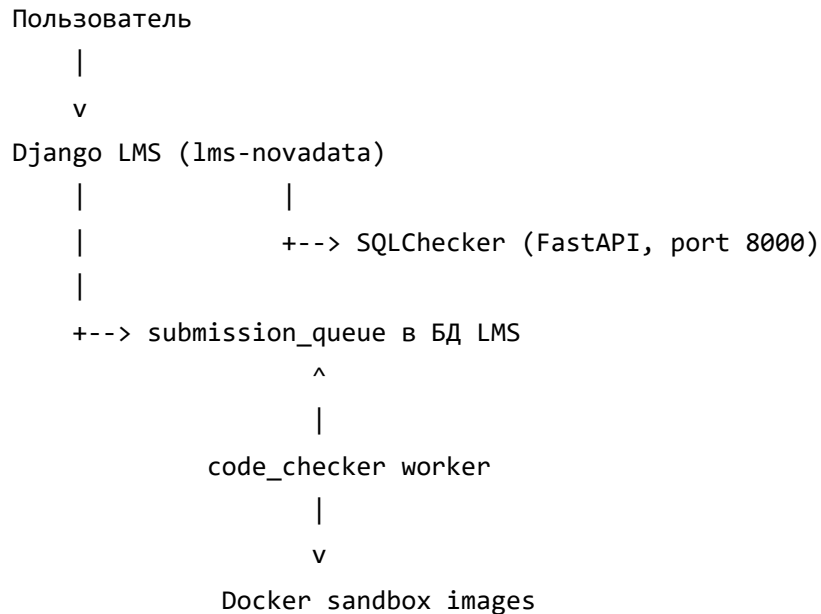
- `lms-novadata` - основное Django-приложение LMS;
- `SQLChecker` - FastAPI-сервис проверки SQL;
- `code_checker` - сервис проверки программного кода в Docker-песочницах.

Связанные документы:

- `README_DOCUMENTATION.md` - индекс комплекта документации и рекомендуемый порядок чтения.
- `PROJECT_DOCUMENTATION.md` - общая техническая документация проекта.
- `ENVIRONMENT_VARIABLES.md` - актуальная схема переменных окружения.
- `API_CONTRACTS.md` - API-контракты LMS/checkers.
- `OPERATIONS_RUNBOOK.md` - регламент эксплуатации и восстановления после сбоев.
- `PROGRAMMING_TASK_AUTHOR_GUIDE.md` - руководство по созданию programming-заданий.
- `SQL_TASK_AUTHOR_GUIDE.md` - руководство по созданию SQL-заданий.

1. Общая схема развертывания

Минимальная рабочая схема:



LMS является центральной частью системы. SQLChecker вызывается LMS напрямую для SQL-заданий. code_checker работает по pull-модели: сам забирает pending-задания из LMS, выполняет проверку и отправляет результат обратно.

2. Требования к серверу

Базовые требования:

- Linux-сервер, рекомендуемо Ubuntu 22.04/24.04;
- Python 3.12 для LMS;
- Python 3.11 для SQLChecker Docker-образа;
- Python 3.13 для code_checker Docker-образа;
- MySQL 8.x для LMS;
- Redis для Celery;
- Docker и Docker Compose для SQLChecker и code_checker;
- PostgreSQL 15 для SQLChecker;
- ClickHouse для SQLChecker, если нужны ClickHouse-задания;
- FFmpeg/FFprobe для обработки видео в LMS;
- S3-совместимое хранилище, например Selectel S3 или MinIO.

Рекомендуемый порядок установки системных пакетов для LMS:

```
sudo apt update
sudo apt install -y mysql-server mysql-client pkg-config libmysqlclient-dev \
    python3-dev python3-pip python3-venv build-essential git curl \
    redis-server ffmpeg
```

Для сервисов проверки также нужны:

```
sudo apt install -y docker.io docker-compose-plugin
sudo usermod -aG docker $USER
```

После добавления пользователя в группу `docker` нужно перелогиниться.

3. Структура каталогов на сервере

Рекомендуемая структура:

```
/var/www/lms          # lms-novadata
/opt/nd_lms/SQLChecker # SQLChecker
/opt/nd_lms/code_checker # code_checker
```

В разработке проект может лежать в одном общем каталоге:

```
nd_lms/
  lms-novadata/
  SQLChecker/
  code_checker/
```

4. Развертывание LMS

4.1. Подготовка кода и окружения

```
sudo mkdir -p /var/www/lms
sudo chown -R $USER:$USER /var/www/lms

cd /var/www/lms
python3 -m venv venv
source venv/bin/activate
pip install --upgrade pip
pip install -r requirements.txt
```

Если проект копируется из общей папки, в `/var/www/lms` должен попасть содержимый каталог `lms-novadata`, то есть там должен находиться `manage.py`.

4.2. MySQL

Создать базу и пользователя:

```
sudo mysql

CREATE DATABASE lms CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
CREATE USER 'lms_user'@'localhost' IDENTIFIED BY 'CHANGE_ME';
GRANT ALL PRIVILEGES ON lms.* TO 'lms_user'@'localhost';
FLUSH PRIVILEGES;
EXIT;
```

Если Django подключается к MySQL не с `localhost`, нужно отдельно создать пользователя для нужного `host`.

4.3. Файл окружения LMS

Файл окружения ожидается в:

```
/var/www/lms/lms_project/.env
```

Минимальный пример:

```
DB_USER=lms_user
DB_PASSWORD=CHANGE_ME
DB_HOST=localhost
DB_PORT=3306

S3_ACCESS_KEY=CHANGE_ME
S3_SECRET_KEY=CHANGE_ME
S3_BUCKET_NAME=lms
S3_ENDPOINT=s3.ru-7.storage.selcloud.ru

VALIDATOR_PROGRAMMING_TOKEN=CHANGE_ME
VALIDATOR_SQL_TOKEN=CHANGE_ME

JITSI_SERVER_URL=
JITSI_APP_ID=lms-app
JITSI_APP_SECRET=
JITSI_JWT_EXPIRE_SECONDS=7200

JUPYTERHUB_URL=
```

Текущая особенность settings.py : AWS_S3_ENDPOINT_URL собирается как https:// + S3_ENDPOINT .
Поэтому в S3_ENDPOINT нужно указывать host без https:// .

4.4. Миграции, статика, суперпользователь

```
cd /var/www/lms
source venv/bin/activate

python manage.py migrate
python manage.py collectstatic --noinput
python manage.py createsuperuser
```

4.5. Gunicorn

В проекте есть шаблон:

```
lms-novadata/deploy/gunicorn.service
```

Текущий шаблон использует:

```
User=vitaliy
Group=vitaliy
WorkingDirectory=/var/www/lms
ExecStart=/var/www/lms/venv/bin/gunicorn --workers 3 --bind 0.0.0.0:8000 lms_project.wsgi:applic
```

Перед установкой нужно заменить User и Group на пользователя сервера.

Установка:

```
sudo cp deploy/gunicorn.service /etc/systemd/system/gunicorn.service
sudo systemctl daemon-reload
sudo systemctl enable gunicorn
sudo systemctl start gunicorn
sudo systemctl status gunicorn --no-pager -l
```

4.6. Celery worker и beat

В проекте есть шаблоны:

```
lms-novadata/deploy/celery.service
lms-novadata/deploy/celery-beat.service
```

Celery worker запускается с очередями:

```
celery,progress_updates,video_processing
```

Установка:

```
sudo cp deploy/celery.service /etc/systemd/system/celery.service
sudo cp deploy/celery-beat.service /etc/systemd/system/celery-beat.service
sudo systemctl daemon-reload
sudo systemctl enable celery celery-beat
sudo systemctl start redis-server celery celery-beat
sudo systemctl status celery --no-pager -l
sudo systemctl status celery-beat --no-pager -l
```

Текущая особенность settings.py : Redis задан как redis://localhost:6379/0 . Если Redis находится на другом хосте или порту, настройки нужно вынести в env или изменить settings.py .

5. Развертывание SQLChecker

5.1. Назначение

SQLChecker проверяет SQL-задания. Он предоставляет FastAPI API на порту `8000` и использует PostgreSQL/ClickHouse для выполнения проверок.

5.2. Файл окружения

SQLChecker/docker-compose.yml ожидает `.env` в корне SQLChecker.

Минимальный пример:

```
POSTGRES_USER=user
POSTGRES_PASSWORD=password
POSTGRES_DB=mydb
```

```
DB_HOST=postgres-db
DB_PORT=5432
DB_USER=user
DB_PASSWORD=password
DB_NAME=mydb
```

```
CH_HOST=clickhouse-db
CH_PORT=8123
CH_USER=default
CH_PASSWORD=
```

```
MAX_PARALLEL_CHECKS=10
QUERY_TIMEOUT_MS=30000
```

Важно: `POSTGRES_*` используются контейнером PostgreSQL, а `DB_*` используются приложением SQLChecker.

5.3. Запуск

```
cd /opt/nd_lms/SQLChecker
docker compose up -d --build
```

Будут подняты:

- sql-checker-api на порту 8000 ;
- postgres-db на порту 5432 ;
- clickhouse-db на портах 8123 и 9000 .

Проверка:

```
docker compose ps
curl http://localhost:8000/health
curl http://localhost:8000/
```

Логи:

```
docker compose logs -f sql-checker-api
docker compose logs -f postgres-db
docker compose logs -f clickhouse-db
```

5.4. Связь LMS с SQLChecker

В текущем lms-novadata/lms_project/settings.py SQL_CHECKER_URL задан жестко:

```
SQL_CHECKER_URL = 'http://10.10.10.18:8000/'
```

При развертывании нужно убедиться, что:

- SQLChecker реально доступен по этому адресу;
- firewall разрешает подключение от LMS к порту 8000 ;
- если адрес другой, значение нужно изменить в settings.py или вынести в переменную окружения.

Рекомендуемая доработка: заменить жестко заданный URL на `os.getenv('SQL_CHECKER_URL', 'http://localhost:8000/')` .

6. Развертывание code_checker

6.1. Назначение

code_checker проверяет programming-задания. Он состоит из:

- api - FastAPI API на порту 5001 ;

- worker - процесс, который забирает задания из LMS;
- sandbox Docker images - образы для выполнения кода студентов.

6.2. Файл окружения

code_checker/docker-compose.yml ожидает .env в корне code_checker .

Пример:

```
LMS_BASE_URL=http://LMS_HOST:8000/  
LMS_API_TOKEN=CHANGE_ME
```

LMS_API_TOKEN должен совпадать с токеном в LMS для programming-валидатора:

```
VALIDATOR_PROGRAMMING_TOKEN=CHANGE_ME
```

Токен передается в LMS через HTTP-заголовок:

```
X-API-Key: <token>
```

6.3. Сборка sandbox images

Перед запуском worker нужно собрать образы песочниц:

```
cd /opt/nd_lms/code_checker  
docker compose --profile build_only build
```

Собираются образы:

- code-checker-python-base:3.14 ;
- code-checker-python-pyspark:3.11 ;
- code-checker-python-sklearn:3.11 ;
- code-checker-python-tensorflow:3.11 ;
- code-checker-python-pytorch:3.11 ;
- code-checker-java-base:21 ;
- code-checker-java-spark:11 ;
- code-checker-scala-spark:2.12 ;
- code-checker-cpp-base:17 ;
- code-checker-csharp-base:10.0 ;

- code-checker-go-base:1.25 ;
- code-checker-haskell-base:9.12 ;
- code-checker-r-base:4.5 ;
- code-checker-rust-base:1.93 .

6.4. Запуск code_checker

```
cd /opt/nd_lms/code_checker
docker compose up -d --build api worker
```

Проверка:

```
docker compose ps
curl http://localhost:5001/docs
docker compose logs -f worker
```

Worker должен писать в логах, что он подключился к LMS_BASE_URL и опрашивает:

```
/api/validator/submissions/?status=pending
```

6.5. Важные особенности Docker

code_checker монтирует Docker socket:

```
volumes:
  - /var/run/docker.sock:/var/run/docker.sock
```

Это нужно, чтобы worker мог запускать sandbox-контейнеры. Одновременно это повышает уровень доступа контейнера к Docker-хосту, поэтому сервис нужно запускать только на доверенном сервере.

Песочницы по умолчанию запускаются с ограничениями:

- mem_limit ;
- pids_limit ;
- security_opt: no-new-privileges ;
- cap_drop: ALL ;
- network_disabled: True для большинства профилей.

Исключение: некоторые Spark-профили могут требовать сеть.

6.6. Известное несоответствие конфигурации

В `code_checker/app/config.py` в `DEFAULTS` указано:

```
"java_base": "17"
```

При этом профиль и Docker Compose используют:

```
java_base_21  
code-checker-java-base:21
```

Перед промышленным запуском нужно проверить, какой Java base считается актуальным, и синхронизировать default-версию с доступным sandbox image.

7. Проверка интеграции после запуска

7.1. Проверить LMS

```
curl http://LMS_HOST:8000/  
sudo systemctl status gunicorn --no-pager -l  
sudo systemctl status celery --no-pager -l  
sudo systemctl status celery-beat --no-pager -l
```

7.2. Проверить SQLChecker

```
curl http://SQL_CHECKER_HOST:8000/health  
curl http://SQL_CHECKER_HOST:8000/
```

7.3. Проверить code_checker

```
curl http://CODE_CHECKER_HOST:5001/docs  
docker compose logs -f worker
```

7.4. Проверить очередь programming-заданий

На стороне LMS:

1. Создать programming-шаг.
2. Отправить решение студентом.
3. Проверить, что в `submission_queue` появилась запись со статусом `pending`.
4. Проверить, что `worker` забрал задачу.
5. Проверить, что результат появился в `submission_history`.

7.5. Проверить SQL-задание

На стороне LMS:

1. Создать SQL-шаг.
2. Указать эталонный запрос и тестовые данные.
3. Выполнить студенческий запрос.
4. Отправить решение на проверку.
5. Убедиться, что LMS получает ответ от `SQLChecker` и обновляет прогресс.

8. Базовые команды обслуживания

8.1. LMS

```
sudo systemctl restart gunicorn
sudo systemctl restart celery
sudo systemctl restart celery-beat
```

```
sudo journalctl -u gunicorn -f
sudo journalctl -u celery -f
sudo journalctl -u celery-beat -f
```

8.2. SQLChecker

```
cd /opt/nd_lms/SQLChecker
docker compose ps
docker compose logs -f sql-checker-api
docker compose restart sql-checker-api
```

8.3. code_checker

```
cd /opt/nd_lms/code_checker
docker compose ps
docker compose logs -f worker
docker compose restart worker
```

9. Типовые проблемы при развертывании

9.1. code_checker не забирает задания

Проверить:

- LMS_BASE_URL заканчивается на / ;
- LMS_API_TOKEN совпадает с VALIDATOR_PROGRAMMING_TOKEN ;
- LMS доступна из контейнера worker;
- endpoint /api/validator/submissions/?status=pending отвечает;
- В submission_queue есть задания с task_type='programming' .

9.2. SQL-задания не проверяются

Проверить:

- доступность SQLChecker по SQL_CHECKER_URL ;
- порт 8000 ;
- health endpoint SQLChecker;
- соединения SQLChecker с PostgreSQL и ClickHouse;
- корректность DB_* и CH_* переменных.

9.3. Celery не обрабатывает задачи LMS

Проверить:

- Redis запущен;
- Celery worker запущен;
- worker слушает очереди celery , progress_updates , video_processing ;
- в settings.py Redis host совпадает с фактическим расположением Redis.

9.4. Sandbox-контейнеры не запускаются

Проверить:

- Docker daemon запущен;
- `/var/run/docker.sock` проброшен в контейнер `code_checker`;
- `sandbox images` собраны;
- имя `image` в `app/config.py` совпадает с именем собранного Docker image;
- пользователю/контейнеру хватает прав на Docker socket.

10. Что нужно уточнить перед production

Перед production-развертыванием нужно отдельно зафиксировать:

- домены и IP всех сервисов;
- схему reverse proxy/Nginx;
- TLS-сертификаты;
- backup MySQL;
- backup S3/MinIO;
- стратегию обновления Docker-образов;
- ротацию логов;
- monitoring и alerting;
- секреты и порядок их ротации;
- политику доступа к Docker socket на сервере `code_checker`.

11. Рекомендованные доработки конфигурации

Чтобы deployment был устойчивее, рекомендуется:

- вынести `SQL_CHECKER_URL` в переменную окружения;
- вынести Redis URL в переменную окружения;
- добавить `.env.example` для SQLChecker без секретов;
- убрать реальные токены из `example`-файлов;
- синхронизировать `java_base default` в `code_checker/app/config.py` ;
- добавить healthcheck для `code_checker-api` ;
- добавить отдельный systemd unit или compose restart-policy для production worker;
- документировать реальные Nginx location/proxy правила после их утверждения.