

# GitLab CI/CD

---

## Подготовка

### Настройка DNS

- Добавить домен `gitlab.local` в `/etc/hosts` (`C:\Windows\System32\drivers\etc\hosts`) для локальной и виртуальной машин:

```
127.0.0.1 gitlab.local
```

- Изменить `hostname` сервиса `gitlab` в `docker-compose.yml` на `gitlab.local`
- Заменить `localhost` на новый домен `gitlab.local` в переменной `GITLAB_OMNIBUS_CONFIG` для сервиса `gitlab` в `docker-compose.yml`:

```
GITLAB_OMNIBUS_CONFIG: |
  external_url 'http://gitlab.local:8929'
  gitlab_rails['gitlab_shell_ssh_port'] = 2224
```

### Включение Container Registry

- Добавить в переменную `GITLAB_OMNIBUS_CONFIG` для сервиса `gitlab` в `docker-compose.yml`:

```
registry_external_url 'http://gitlab.local:5005'
gitlab_rails['registry_enabled'] = true
gitlab_rails['registry_host'] = 'gitlab.local'
gitlab_rails['registry_port'] = 5005
```

- Открыть порт 5005 у контейнера GitLab
- Перезапустить GitLab (`docker compose up -d gitlab`)
- Убедиться, что Container Registry включен (`http://gitlab.local:8929/admin`)

### Настройка GitLab Runner

- Отредактировать конфиг runner-a `runner/config.toml`
  - Увеличить кол-во параллельно выполняемых задач до двух (`concurrent`)
  - Изменить `url` в секции `[[runners]]` на `http://gitlab.local:8929`
  - Добавить строчку `extra_hosts = ["gitlab.local:host-gateway"]` в секцию `[runners.docker]`.
  - Добавить `volume` на сокет докера (`"/var/run/docker.sock:/var/run/docker.sock"`) в список `volumes` в секции `[runners.docker]`

```
concurrent = 2
check_interval = 0
shutdown_timeout = 0

[session_server]
  session_timeout = 1800

[[runners]]
  name = "default"
  url = "http://gitlab.local:8929"
  id = 1
  token = "#####"
  token_obtained_at = 2023-10-15T17:17:10Z
  token_expires_at = 0001-01-01T00:00:00Z
  executor = "docker"
[runners.cache]
  MaxUploadedArchiveSize = 0
[runners.docker]
  tls_verify = false
  image = "docker:stable"
  privileged = false
  disable_entrypoint_overwrite = false
  oom_kill_disable = false
  disable_cache = false
  extra_hosts = ["gitlab.local:host-gateway"]
  volumes = ["/var/run/docker.sock:/var/run/docker.sock", "/cache"]
  shm_size = 0
```

- Перезапустить runner (`docker compose restart runner`)

## Настройка GitLab

- Включить импорт проектов по ссылке ([http://gitlab.local:8929/admin/application\\_settings/general](http://gitlab.local:8929/admin/application_settings/general) -> Import and export settings -> Import sources -> Repository by URL)
- Разрешить запускать untagged jobs для подключенного runner-а (<http://gitlab.local:8929/admin/runners/1>)

## Подготовка репозитория

- Ознакомиться с проектом [devops-todo](#) на GitHub
- Импортировать проект [devops-todo](#) ([http://gitlab.local:8929/projects/new#import\\_project](http://gitlab.local:8929/projects/new#import_project))

## Подготовка локального репозитория

- Сгенерировать ssh ключ `ssh-keygen -t ed25519 -C "your_email@example.com"`
- Скопировать свой публичный ключ `cat ~/.ssh/id_ed25519.pub`
- Добавить публичный ключ в GitLab (<http://gitlab.local:8929/-/profile/keys>)
- Склонировать репозиторий (`git clone ssh://git@gitlab.local:2224/dmitry.sinelnikov/devops-todo.git`)
- Создать новую ветку (`git checkout -b task3`)

# Контейнеризация проекта

## Написание Dockerfile для приложения

- Добавить файл .dockerignore в корень проекта:

```
# See https://help.github.com/articles/ignoring-files/ for more about
ignoring files.

# dependencies
/node_modules
/.pnp
.pnp.js

# testing
/coverage

# next.js
/.next/
/out/

# production
/build

# misc
.DS_Store
*.pem

# debug
npm-debug.log*
yarn-debug.log*
yarn-error.log*

# local env files
.env*.local

# vercel
.vercel

# typescript
*.tsbuildinfo
next-env.d.ts

.env
Dockerfile
docker-compose.yml
```

- Создать Dockerfile в корне проекта

## Рекомендации по написанию Dockerfile

### Сборка приложения

- Использовать базовый образ [https://hub.docker.com/\\_/node](https://hub.docker.com/_/node) (нужная версия указана в описании проекта)
- Выполнить команду `npm install` из корня проекта
- Выполнить команду `npm run build` из корня проекта
- Скорировать папку `prisma` из корня проекта в папку `.next/standalone`

### Запуск приложения

- Все необходимые файлы для запуска проекта находятся в директории `.next/standalone`, которая получается в результате сборки проекта
- Установить зависимости: `openssl, zlib, libgcc` (для alpine: `apk add openssl zlib libgcc`)
- Установить prisma: `npm install -g prisma`
- В контейнере необходимо указать переменную среды `NODE_ENV=production`
- Для запуска миграций выполнить команду `npm run migrate deploy` из папки `.next/standalone`
- Для запуска приложения выполнить `node server.js` из папки `.next/standalone`
- Приложение слушает запросы на порту 3000

### Написание тестового docker-compose для приложения

- Создать `docker-compose.local.yml` в корне проекта

### Рекомендации по написанию docker compose

- Приложение использует базу данных (БД) PostgreSQL (версия указана в описании проекта)
- Пример docker compose сервиса для PostgreSQL

```
postgres:
  image: postgres:15.4-alpine
  environment:
    POSTGRES_USER: todo
    POSTGRES_PASSWORD: TODOPASSWORD
    POSTGRES_DB: todo
  volumes:
    - postgres:/var/lib/postgresql/data
  ports:
    - "5432:5432"

volumes:
  postgres:
```

- Для указания адреса БД используйте переменную среды `DATABASE_URL` (`postgresql://<user>:<password>@<address>:5432/<database>?schema=public`).  
Пример: `DATABASE_URL=postgresql://todo:TODOPASSWORD@postgres:5432/todo?schema=public`

## Проверка работы приложения

- Собрать проект `docker compose -f docker-compose.local.yml build`
- Запустить СУБД `docker compose -f docker-compose.local.yml up -d postgres`
- Запустить миграции `docker compose -f docker-compose.local.yml run --rm todo npm run migrate deploy`
- Запустить приложение `docker compose -f docker-compose.local.yml up -d`
- Открыть `http://localhost:3000` в браузере и убедиться, что приложение работает
- Выключить проект `docker compose down -v`

## Workflow

```
%%{init: { 'theme': 'base' } }%%
gitGraph
    commit
    commit
    branch feature
    checkout feature
    commit
    commit
    checkout main
    merge feature
    commit
    commit
```

```
stateDiagram-v2
    direction LR
    Checkout --> Code
    Code --> MR
    MR --> Test
    Test --> Review
    Review --> Merge
    Merge --> Release
```

## DevOps Pipeline

```
flowchart LR
    subgraph TestStage [Test Stage]
        Lint
        Test
    end
    subgraph PublishStage [Publish Stage]
        Publish
    end
    subgraph DeployStage [Deploy Stage]
        Deploy
    end
```

```

end

MR([Merge Request])
Merge(Review\n&\nMerge)
Registry[(Container\nRegistry)]
Server((Production\nServer))

MR --> Lint --> Merge
MR --> Test --> Merge
Merge --> Publish
Publish -.-> Registry
Publish --> Deploy
Registry -.-> Deploy
Deploy -.-> Server

```

## Continuous Integration (CI)

```

flowchart LR
    MR([Merge Request]) --> Lint & Test
    Lint & Test -->|Fail| Fail[Block MR]

```

- Добавить `.gitlab-ci.yml` в корень проекта
- Для тестирования дальнейших шагов необходимо делать создать новый Merge Request своей ветки в ветку main и пушить изменения в git (`git push`)
- Добавить стадию stages в `.gitlab-ci.yml`:

```

stages:
  - test

```

## Проверка качества кода (linting)

- Описать задачу `lint` для стадии `test`
- Указать базовый образ `node:18.17.1-alpine`
- Прописать команды для выполнения:
  - `npm install` - установка зависимостей
  - `npm run lint` - запуск проверки качества кода
- Пример:

```

lint:
  stage: test
  image: node:18.17.1-alpine
  before_script:
    - npm install
  script:
    - npm run lint

```

## Тестирование

- Описать задачу `test` для стадии `test`
- Указать базовый образ `node:18.17.1-alpine`
- Для запуска тестов необходима БД PostgreSQL. Поднимать дополнительные зависимости возможно через указание опции `services` в задаче. Пример:

```
services:  
- name: postgres:15.4-alpine  
  alias: postgres  
  variables:  
    POSTGRES_DB: todo-test  
    POSTGRES_USER: todo  
    POSTGRES_PASSWORD: todo
```

- Указать переменную среды `DATABASE_URL`, указывающую сервис postgres:

```
variables:  
  DATABASE_URL: postgresql://todo:todo@postgres:5432/todo-test?  
  schema=public
```

- Прописать установку зависимостей:
  - `apk add openssl zlib libgcc`
  - `npm install`
  - `npm install -g prisma`
- Прописать команду для выполнения тестов: `npm run test`
- Опционально, для сбора тестового покрытия использовать параметр `coverage`:

```
coverage: /All files[^\]*\| [^\]*\s+([\d\.]+)/
```

## Continuous Delivery (CD)

```
flowchart LR  
  Push([Push to main]) --> Publish --> Deploy  
  Publish -.-> Registry[(Container\nRegistry)] -.-> Deploy -.->  
  Server[(Production\nServer)]
```

- Смержить Merge Request, созданный на предыдущих шагах
- Сменить ветку на `main` (`git checkout main`)
- Обновить локальную ветку `main` (`git pull`)
- Для тестирования дальнейших изменений можно пушить в ветку `main` напрямую

## Подготовка

### Установка PostgreSQL

- Создать сеть для приложения (`docker network create todo`)
- Написать `docker-compose.yml` и добавить на сервер. Пример:

```
version: "3"

services:
  postgres:
    image: postgres:15.4-alpine
    container_name: postgres
    networks:
      - todo
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: MYSECRETPASSWORD
    volumes:
      - postgres:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    restart: unless-stopped

volumes:
  postgres:

networks:
  todo:
    external: true
```

### Настройка PostgreSQL

- Запустить PostgreSQL
- Подключиться к PostgreSQL `docker compose exec postgres psql -U postgres`
- Создать базу данных (БД) для приложения `CREATE DATABASE todo;`
- Создать пользователя для приложения `CREATE USER todo WITH ENCRYPTED PASSWORD 'TODOSECRETPASSWORD';`
- **Записать имя пользователя и пароль. Они понадобятся позднее**
- Сделать пользователя владельцем БД `ALTER DATABASE todo OWNER TO todo;`
- Выдать пользователю все права на БД `GRANT ALL PRIVILEGES ON DATABASE todo TO todo;`
- Отключиться от PostgreSQL

### Добавление ssh доступа для GitLab CI

- Сгенерировать ssh ключи в папку `./sshkeys`



```
mkdir ./sshkeys
ssh-keygen -t ed25519 -f ./sshkeys/id_ed25519 -C "gitlab-ci"
```

- Добавить публичный ключ `./sshkeys/id_ed25519.pub` в файл `~/.ssh/authorized_keys` на сервере (если файл на сервере не существует, то его нужно создать)

## Настройка переменных GitLab

- На странице проекта в GitLab открыть Settings -> CI/CD -> Variables
- Добавить следующие переменные:
  - `DB_NAME` - имя базы данных (`todo`)
  - `DB_PASSWORD` - пароль от базы данных (указывался при настройке PostgreSQL). Обязательно указать флаг "Mask variable". Иначе пароль будет виден в логах.
  - `DB_USER` - имя пользователя базы данных (`todo`)
  - `SERVER_ADDR` - адрес сервера, куда будет деплоиться приложение (`localhost`)
  - `SERVER_USER` - имя пользователя на сервере из под которого будет происходить запуск приложения (указать свое имя пользователя)
  - `SSH_KNOWN_HOSTS` - ключи сервера. Для получения ключей использовать команду `ssh-keyscan localhost`. В качестве типа указать File
  - `SSH_PRIVATE_KEY` - сгенерированный приватный ключ от сервера (`./sshkeys/id_ed25519`). В качестве типа указать File

## Публикация образа приложения (publish)

- Добавить стадию сборки `publish`:

```
stages:
  - test
  - publish
```

- Описать задачу `publish` для стадии `publish`
- В качестве базового образа использовать `docker:stable`
- В начале выполнения задачи подключиться к docker registry: `docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY`
- В время выполнения задачи:
  - Собрать docker образ `docker build -t $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_NAME .`
  - Опубликовать docker образ `docker push $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_NAME`
- После выполнения задачи отключиться от docker registry `docker logout $CI_REGISTRY`
- Чтобы задача исполнялась только для коммитов в ветке `main`, а не для всех коммитов, необходимо добавить специальное правило в задачу:

```
rules:
  - if: $CI_COMMIT_BRANCH == main
```

## Деплой приложения

- Добавить стадию сборки `deploy`:

```
stages:
  - test
  - publish
  - deploy
```

- Описать задачу `deploy` для стадии `deploy`
- В качестве базового образа использовать `docker:stable`
- В variables прописать `DATABASE_URL`  
(`postgresql://$DB_USER:$DB_PASSWORD@postgres:5432/$DB_NAME?schema=public`)
- В начале выполнения задачи:
  - Настроить `ssh` для подключения к серверу

```
apk add openssh
eval $(ssh-agent -s)
chmod 400 "$SSH_PRIVATE_KEY"
ssh-add "$SSH_PRIVATE_KEY"
mkdir -p ~/.ssh
chmod 700 ~/.ssh
cp "$SSH_KNOWN_HOSTS" ~/.ssh/known_hosts
chmod 644 ~/.ssh/known_hosts
```

- Настроить `docker` для работы через `ssh`

```
docker context create production --docker
"host=ssh://$SERVER_USER@$SERVER_ADDR"
docker context use production
```

- Подключиться к `docker registry`:

```
docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD
$CI_REGISTRY
```

- Во время выполнения задачи:
  - Скачать образ из `docker registry`:

```
docker pull $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_NAME
```

- Запустить миграции

```
docker run --rm --network=todo -e DATABASE_URL="$DATABASE_URL"  
$CI_REGISTRY_IMAGE:$CI_COMMIT_REF_NAME npm run migrate deploy
```

- Остановить и удалить старый контейнер, если запущен: `docker rm -f todo || true`
- Запустить новый контейнер:

```
docker run -d --name=todo --network=todo --restart=unless-stopped  
-e DATABASE_URL="$DATABASE_URL" -p 80:3000  
$CI_REGISTRY_IMAGE:$CI_COMMIT_REF_NAME
```

- После выполнения задачи:
  - Отключиться от docker registry: `docker logout $CI_REGISTRY`
  - Удалить docker context: `docker context rm -f $CONTEXT`