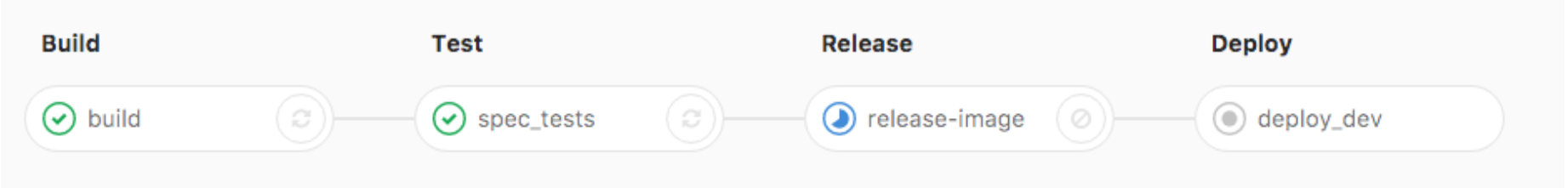
Continuous Integration

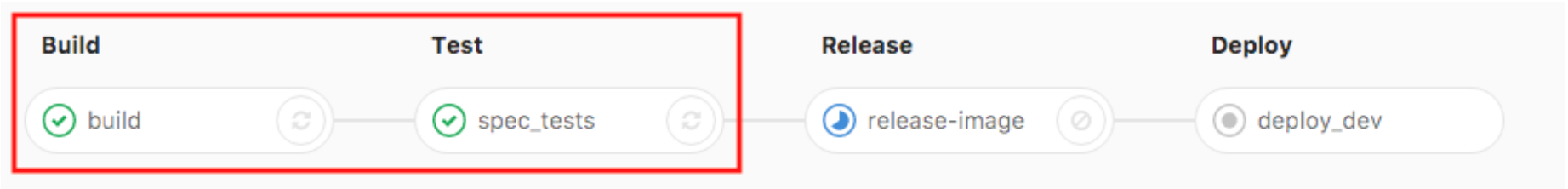
Пайплайн


Build
Test
Release
Deploy
build
spec_tests
release-image
deploy_dev

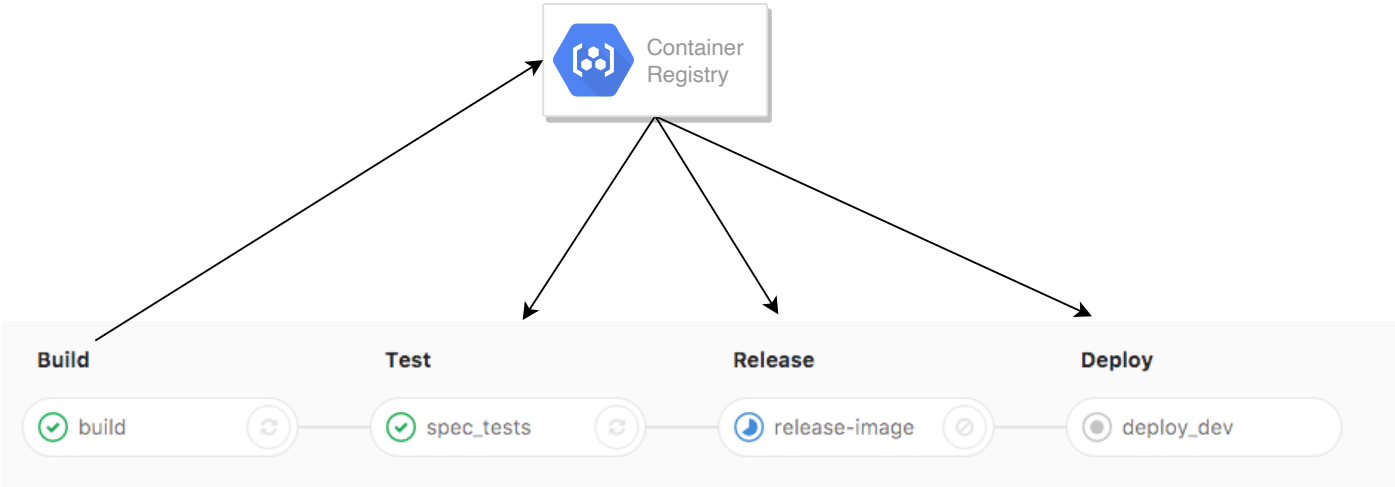
Непрерывная интеграция (Continuous Integration)

Непрерывная интеграция (Continuous Integration)

это практика разработки, в которой все участники команды часто интегрируют свою работу в общую кодовую базу. Каждая интеграция автоматически проверяется на наличие ошибок.



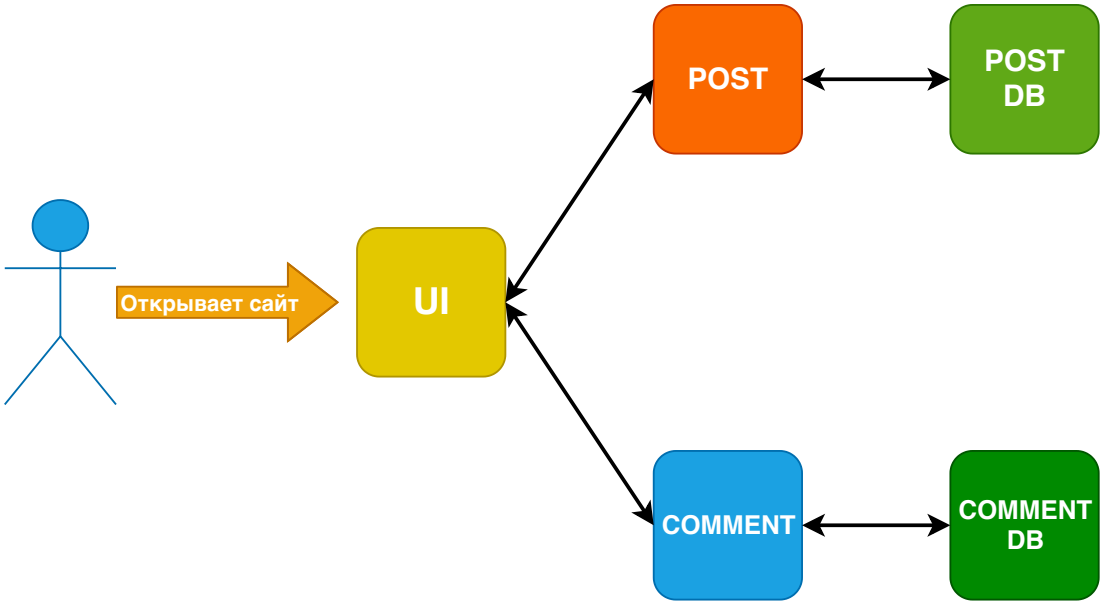
Везде один артефакт



- Разные шаги могут запуститься на разных машинах
- Один и тот же способ запуска приложения для всех стадий
- Гарантия что в проде запускается то что тестировали

Практика 1_ci

Приложение для практики



Приложение для практики

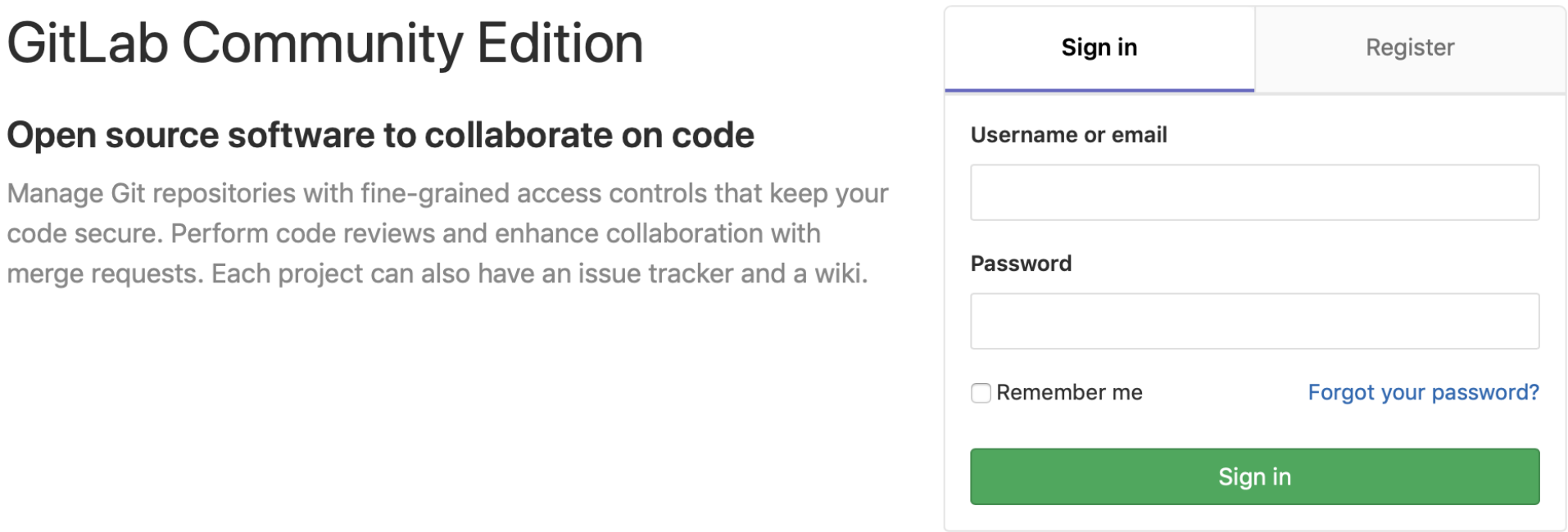
Маленькое микросервисное приложение, которое состоит из:

- `ui` – web-интерфейс приложения. Работает на порту **9292**. Взаимодействует с `post` и `comment`
- `post` – сервис, по созданию и получению постов. Посты хранятся в `mongodb`. Работает на порту **5000** Взаимодействует с `ui` и `mongodb`.
- `comment` – сервис, по созданию и получению комментариев. Комментарии хранятся в `mongodb`. Работает на порту **9292**. Взаимодействует с `ui` и `mongodb`.
- `mongodb` – база данных. Хранит посты и комментарии. Взаимодействует с `post` и `comment`

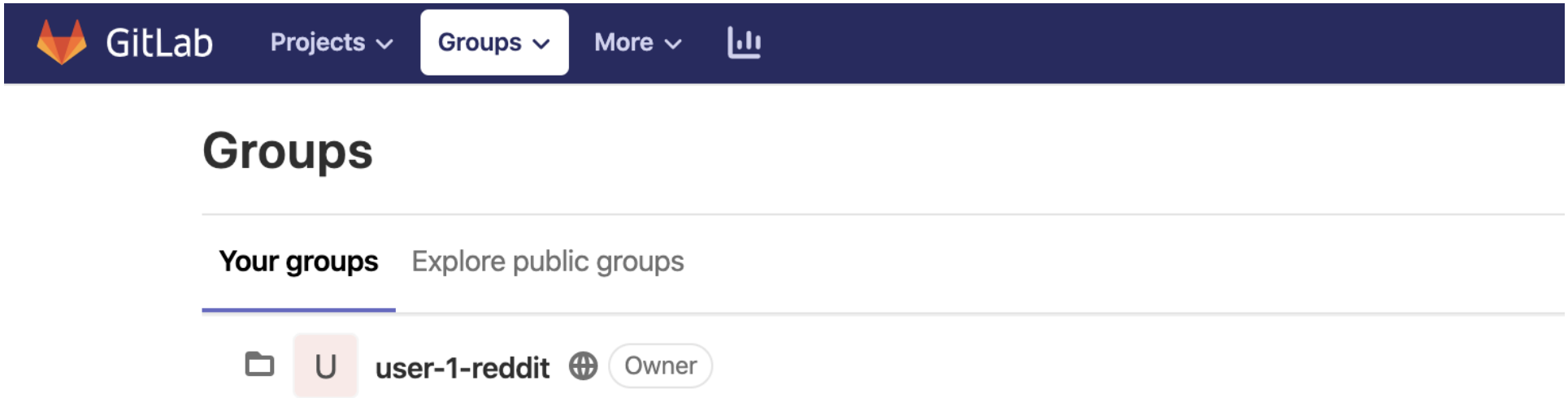
Исходники уже у вас в Web IDE в папке project.

Авторизоваться в Gitlab

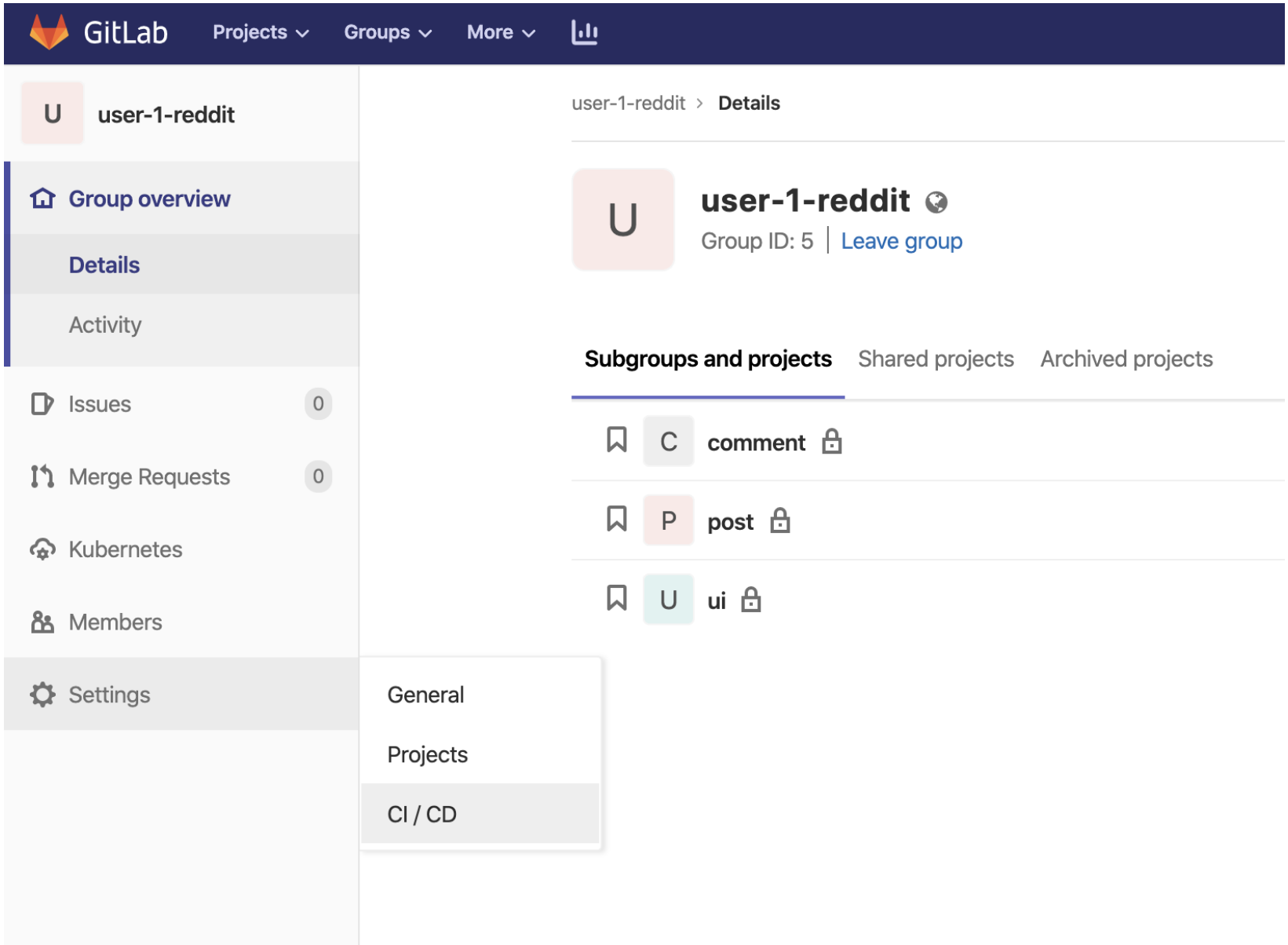
- Адрес Gitlab: http://gitlab.ci-cd.workshop.express42.io
- Логин: **user-<your number>** // пример: user-1
- Пароль: **Pa$$w0rd**



Переходим в группу своих приложений

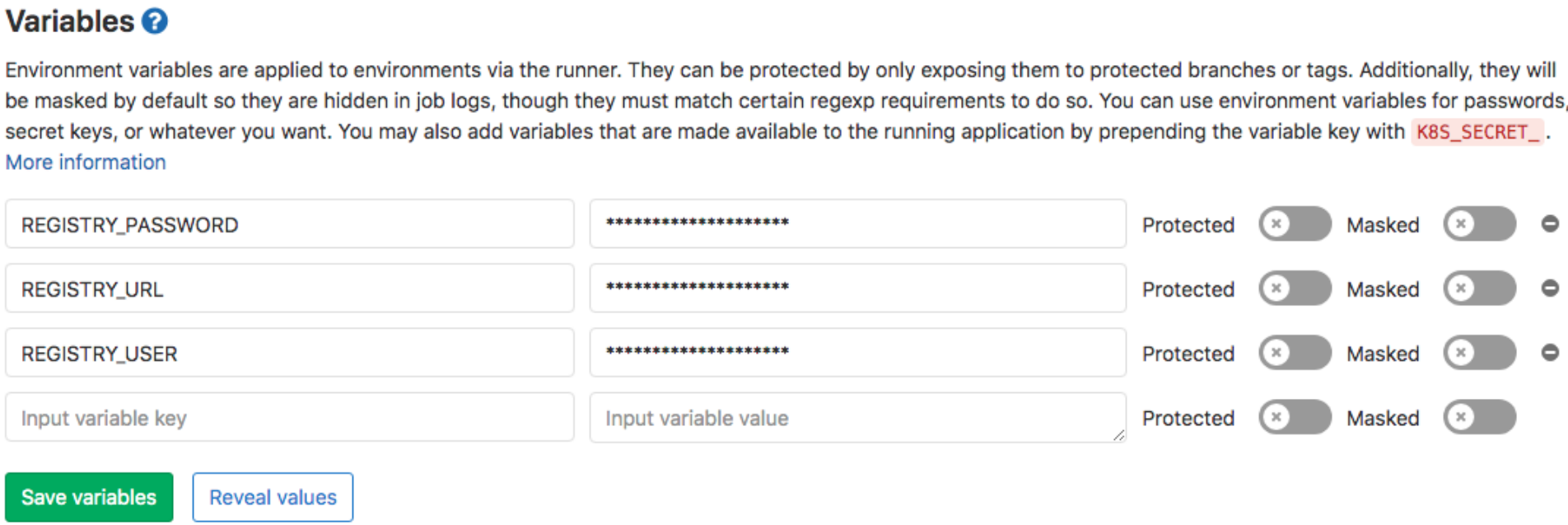


Переходим в настройки CI/CD

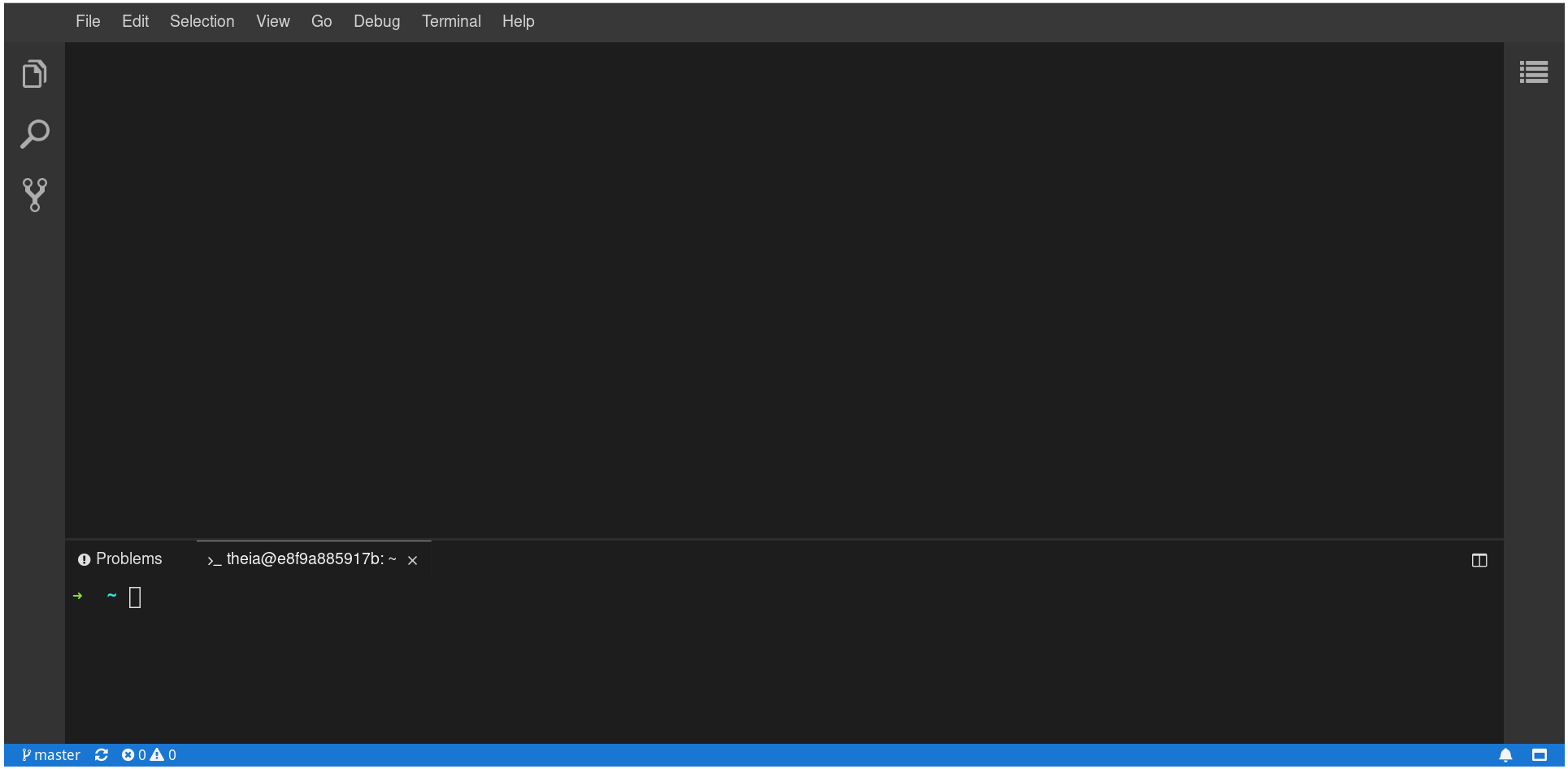


Задаем общие ENV-переменные

Задаем настройки авторизации в docker registry. Ваши значения можно найти в файле `~/project/registry.creds`



Theia IDE


File
Edit
Selection
View
Go
Debug
Terminal
Help
Problems
theia@e8f9a885917b: ~
master
0
0

Theia IDE

Установлено:

- zsh
- git, kubectl, helm
- YAML linter

Обычно:

- bash
- Автодополняющих плагинов нет

Настраиваем git

- Переходим в директорию project

```
cd project/
```

- Указываем email и имя (они будут видны в истории изменений)

```
git config --global user.email "user-<your number>@example.com"
git config --global user.name "user-<your number>"
```

Клонируем код из репозитория

Склонируем репозитории:

```
git clone git@gitlab.ci-cd.workshop.express42.io:user-<your number>-reddit/ui.git
git clone git@gitlab.ci-cd.workshop.express42.io:user-<your number>-reddit/post.git
git clone git@gitlab.ci-cd.workshop.express42.io:user-<your number>-reddit/comment.git
```

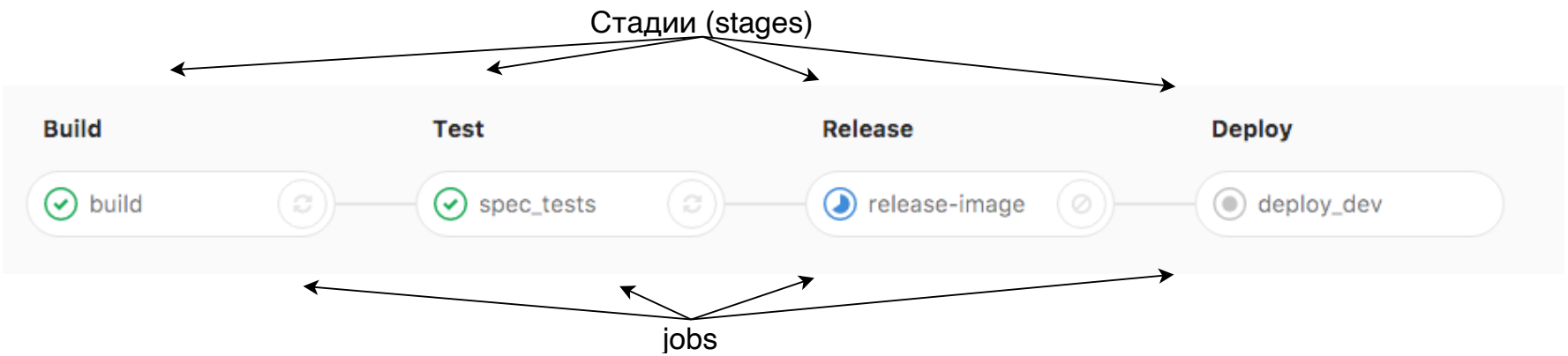
Пример .gitlab-ci.yml

```yaml
before_script:    # Команды, которые должны выполниться перед всем остальным
  - …

stages:           # Порядок стадий в pipeline
  - build
  - test

build:            # job
  stage: build    # в какой стадии job выполняется
  script:
    - …           # Список команд, которые нужно выполнить

```



gitlab-ci.yml для ui. Шаг 1

- Создайте файл `project/ui/.gitlab-ci.yml`
- Опишите в нем `before_script`, с командой:

```
1   before_script:
2     - echo $REGISTRY_PASSWORD | docker login -u $REGISTRY_USER --password-stdin $REGISTRY_URL
```

> `docker login` нужен для того чтобы делать push и pull образов из хранилища артефактов.

gitlab-ci.yml для ui. Шаг 2

Добавьте стадию `build` с командами:

```
stages:
  - build

build:
  stage: build
  except:
    - tags
  script:
    - docker build -t gcr.io/cd-k8s-236617/user-<your number>-ui:$CI_COMMIT_SHORT_SHA ./
    - docker push gcr.io/cd-k8s-236617/user-<your number>-ui:$CI_COMMIT_SHORT_SHA
```

- `$CI_COMMIT_SHORT_SHA` - переменная с хешом коммита. Обычно используется как тег для временных сборок.
- `docker push` вызывается для того чтобы отправить образ в хранилище и использовать его на других стадиях.

gitlab-ci.yml для ui. Шаг 3

Добавьте стадию `test` с командой:

```
stages:
  - build
  - test

...

test:
  stage: test
  except:
    - tags
  script:
    - docker run --rm -t gcr.io/cd-k8s-236617/user-<your number>-ui:$CI_COMMIT_SHORT_SHA
sh ./very_hard_test.sh
```

gitlab-ci.yml для ui. Шаг 4

- Сохраняем изменения локально:

```
1    git add .gitlab-ci.yml
2    git commit -m 'Add build and test'
```

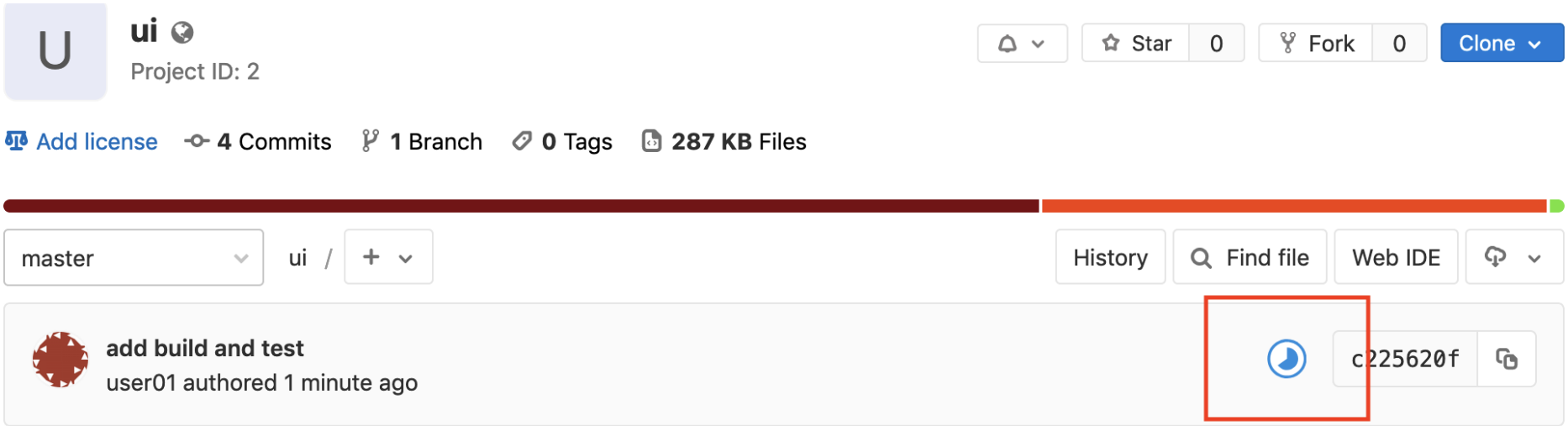
- Пушим (выгружаем) изменения в репозиторий Gitlab

```
1   git push origin master
```

> Не забудьте перейти в терминале в директорию `project/ui/`

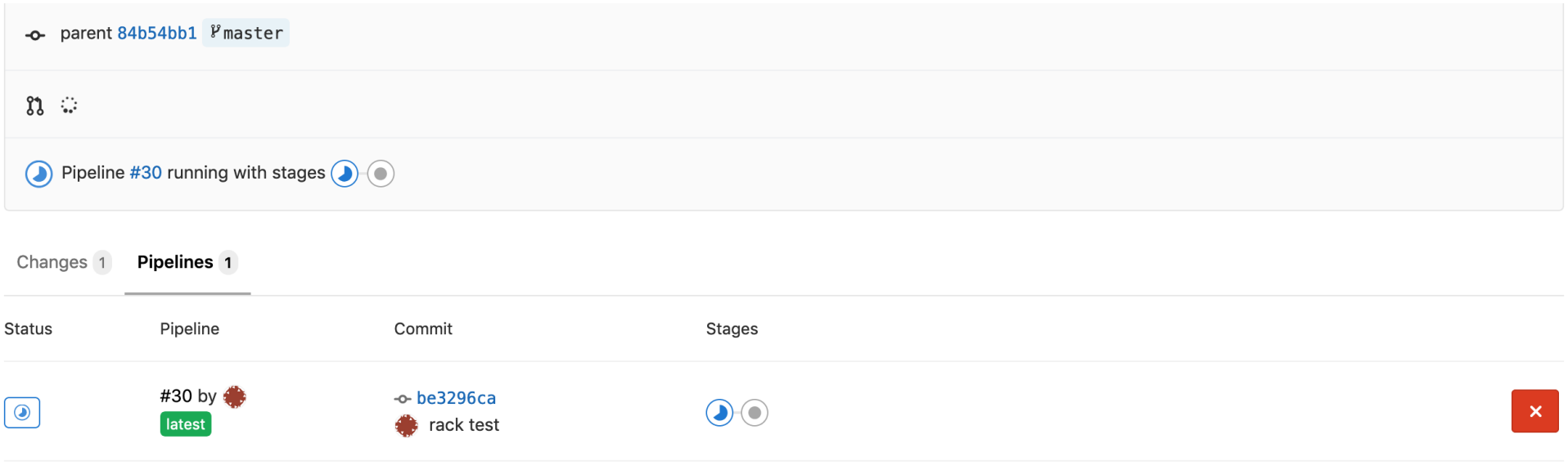
Проверим

В ui-репозитории должен запуститься процесс сборки:



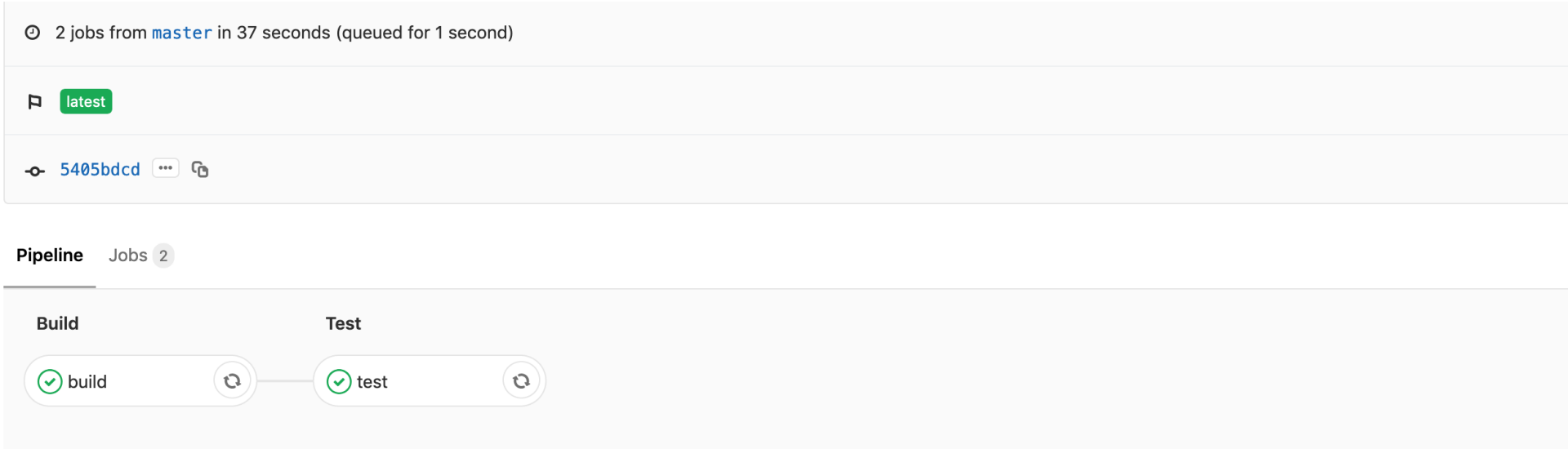
После клика на кружок должна быть подобная картина

Тут можно посмотреть историю запусков (jobs) и какой из них работает сейчас.



Кликнув на job можно увидеть pipeline

Все шаги должны быть зелеными.



> **P.S.** GitLab может не показывать статус в реальном времени, обновляйте страницу.

Дополняем gitlab-ci.yml для UI

Мы протестировали наш артефакт сборки и убедились, что он работает. Пришла пора получить “релизный” образ, который будет использоваться для развертывания на наше production окружение.

Для сигнала о том, что мы готовы сделать релиз будем использовать команду `git tag`.

Release

Добавим стадию `release`. В ней мы получим образ, который прошел тестирование. Явно укажем, что данная стадия должна выполняться только тогда, когда выставлен Git tag.

```
stages:
  ...
  - release

...
release:
  stage: release
  only:
    - tags        # Условие запуска Job - только если есть метка Git
  script:
    - docker pull gcr.io/cd-k8s-236617/user-<your number>-ui:$CI_COMMIT_SHORT_SHA
    - docker tag gcr.io/cd-k8s-236617/user-<your number>-ui:$CI_COMMIT_SHORT_SHA
gcr.io/cd-k8s-236617/user-<your number>-ui:$CI_COMMIT_TAG
    - docker push gcr.io/cd-k8s-236617/user-<your number>-ui:$CI_COMMIT_TAG
```

Release

- Сохраняем изменения:

```
git commit -am 'Add release stage'
```

- Пушим в GitLab:

```
git push origin master
```

- Выставляем Git tag:

```
git tag v0.0.1
```

- Пушим метки в GitLab:

```
git push origin master --tags
```

Самостоятельное задание

- Повторите настройку pipeline для репозиториев `post` и `comment`
- Убедитесь, что сборка выполняется успешно

> ! Не забудьте подставить название нужного сервиса в конфигурацию пайплайна

Что мы сделали

- Склонировали локально репозитории
- Настроили CI-систему
- Получили артефакты сборки (образы Docker), которые будем использовать в дальнейшем

Дополнительные рекомендации

- Достигнуть ежедневного мержа в master-ветку, иначе это не CI
- Использовать один артефакт на всех стадиях. Что собрали, то тестируем. Что тестировали, то идет в прод.