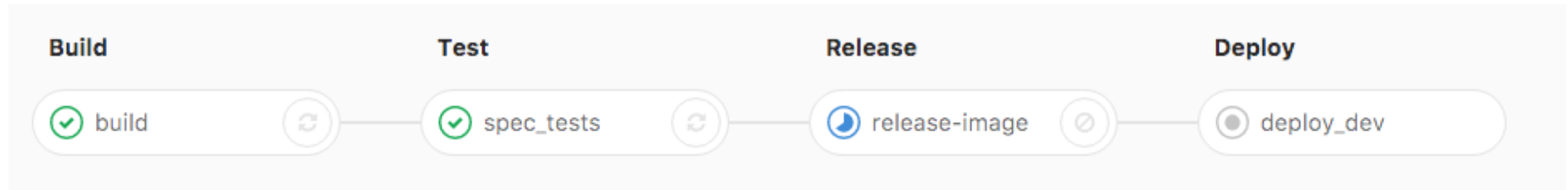


Continuous Integration

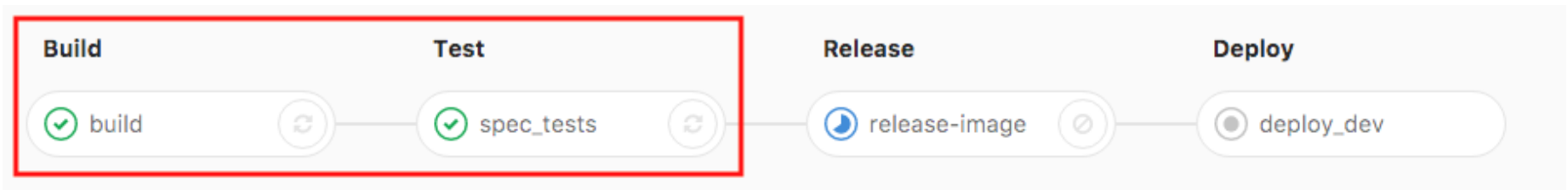
Пайплайн



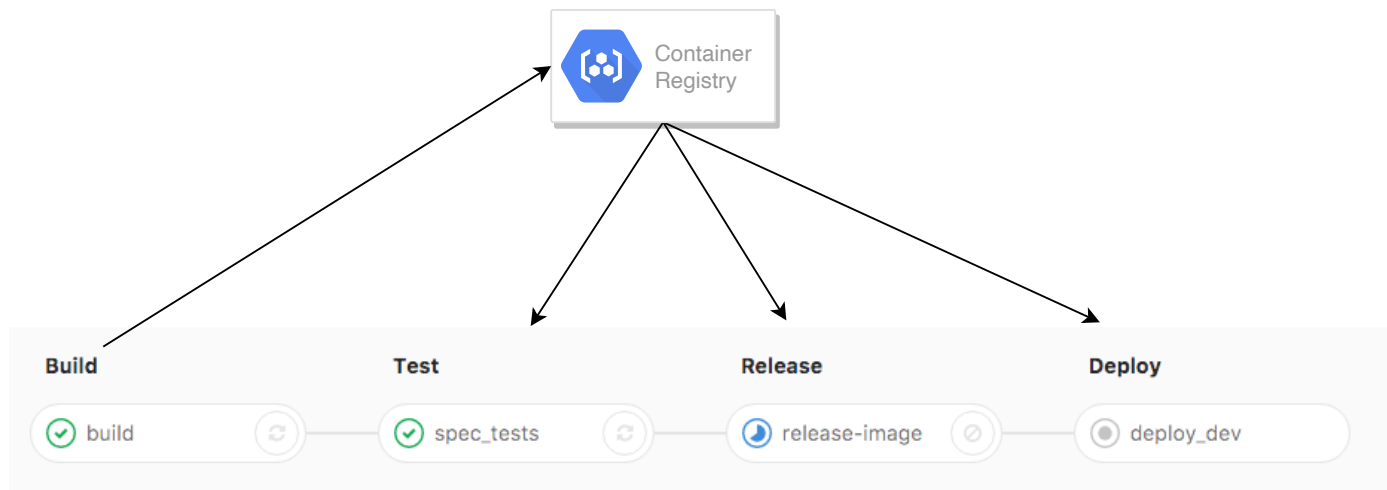
Непрерывная интеграция (Continuous Integration)

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.



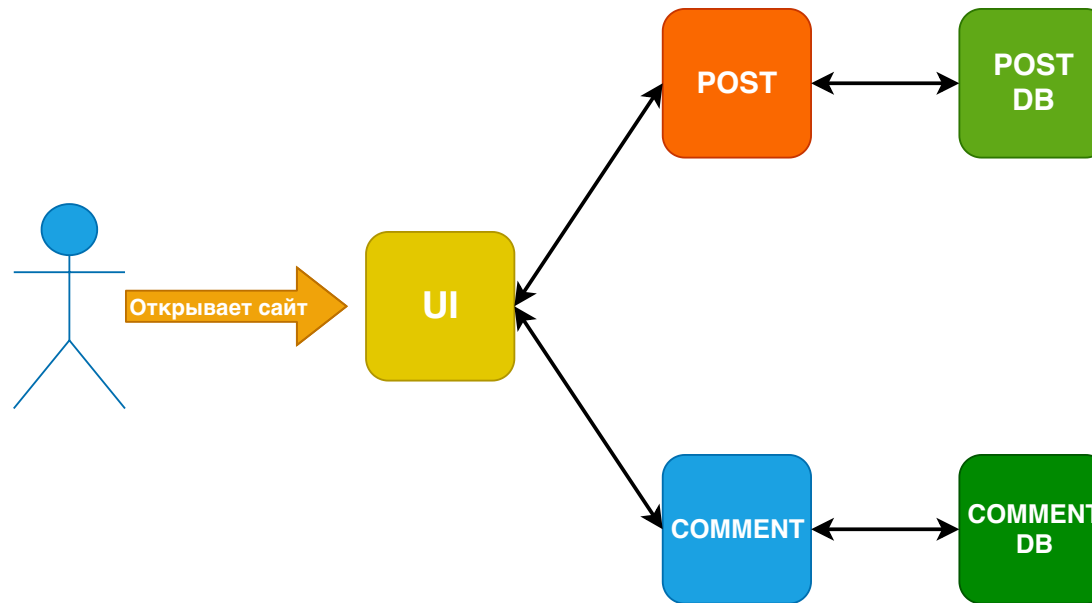
Везде один артефакт



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

Практика 1_сі

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**

GitLab Community Edition

Open source software to collaborate on code

Manage Git repositories with fine-grained access controls that keep your code secure. Perform code reviews and enhance collaboration with merge requests. Each project can also have an issue tracker and a wiki.

Sign in	Register
----------------	----------

Username or email

Password

Remember me [Forgot your password?](#)

Sign in

Переходим в группу своих приложений



Groups

Your groups [Explore public groups](#)

  user-1-reddit  Owner

Переходим в настройки CI/CD

The screenshot shows the GitLab web interface. At the top is a dark blue navigation bar with the GitLab logo and menu items: "Projects", "Groups", and "More". Below this is a sidebar on the left with a navigation menu for the "user-1-reddit" group. The menu items are: "user-1-reddit", "Group overview", "Details" (highlighted with a blue bar), "Activity", "Issues" (0), "Merge Requests" (0), "Kubernetes", "Members", "Settings", and a dropdown menu for "Settings". The dropdown menu is open, showing three options: "General", "Projects", and "CI / CD" (highlighted). The main content area on the right shows the "Details" page for the "user-1-reddit" group. It includes a group profile card with the name "user-1-reddit", a globe icon, and "Group ID: 5 | Leave group". Below this are tabs for "Subgroups and projects", "Shared projects", and "Archived projects". Under the "Subgroups and projects" tab, there is a list of three subgroups: "comment", "post", and "ui", each with a bookmark icon and a lock icon.

Задаем общие ENV-переменные

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

Variables

Environment variables are applied to environments via the runner. They can be protected by only exposing them to protected branches or tags. Additionally, they will be masked by default so they are hidden in job logs, though they must match certain regexp requirements to do so. You can use environment variables for passwords, secret keys, or whatever you want. You may also add variables that are made available to the running application by prepending the variable key with `K8S_SECRET_`.

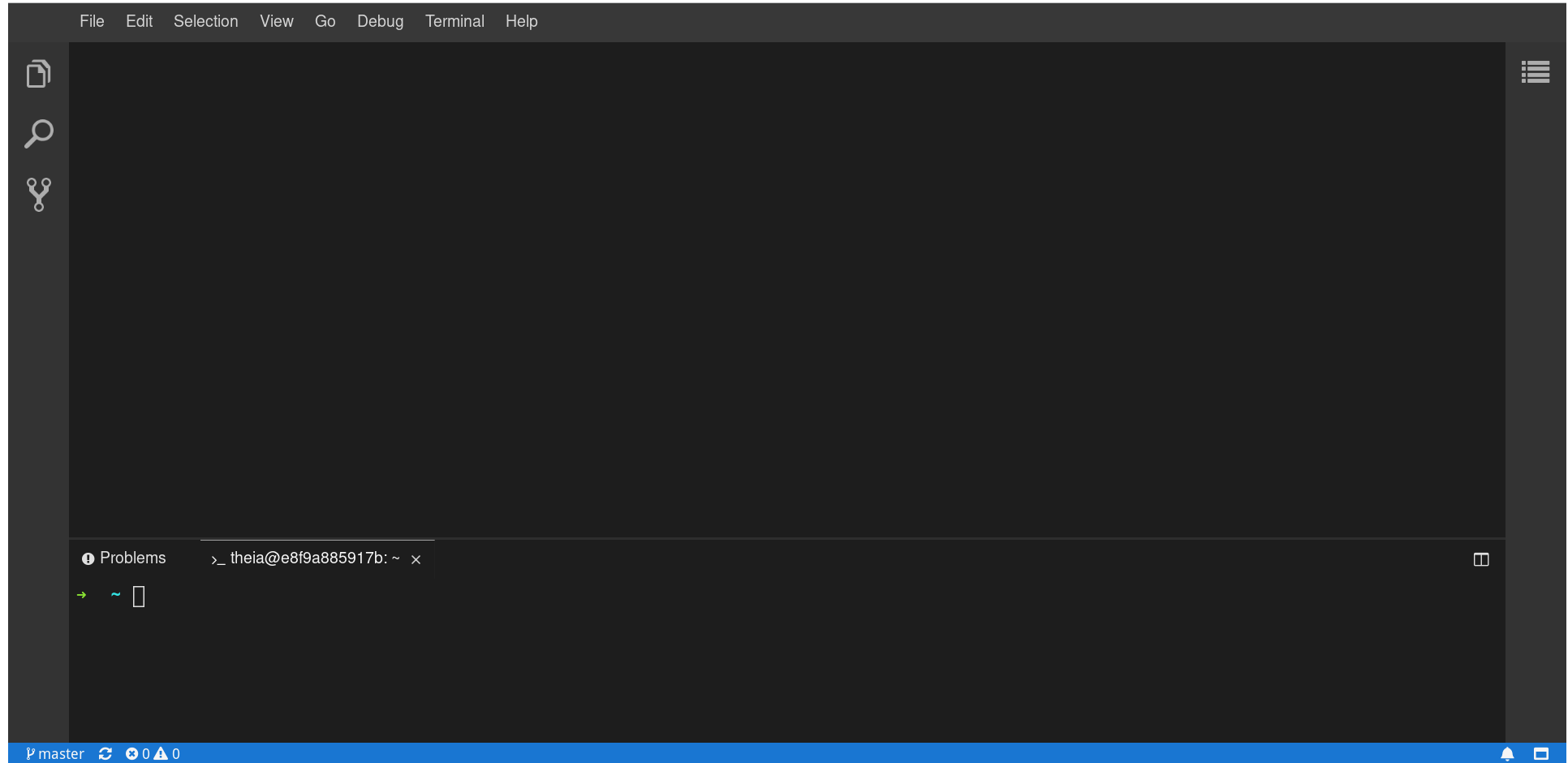
[More information](#)

REGISTRY_PASSWORD	*****	Protected <input checked="" type="checkbox"/>	Masked <input checked="" type="checkbox"/>	⊖
REGISTRY_URL	*****	Protected <input checked="" type="checkbox"/>	Masked <input checked="" type="checkbox"/>	⊖
REGISTRY_USER	*****	Protected <input checked="" type="checkbox"/>	Masked <input checked="" type="checkbox"/>	⊖
Input variable key	Input variable value	Protected <input checked="" type="checkbox"/>	Masked <input checked="" type="checkbox"/>	

Save variables

Reveal values

Theia IDE



Theia IDE

Установлено:

- zsh
- git, kubectl, helm
- YAML linter

Обычно:

- bash
- Автодополняющих плагинов нет

Настраиваем git

- Переходим в директорию project

```
1 cd project/
```

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

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

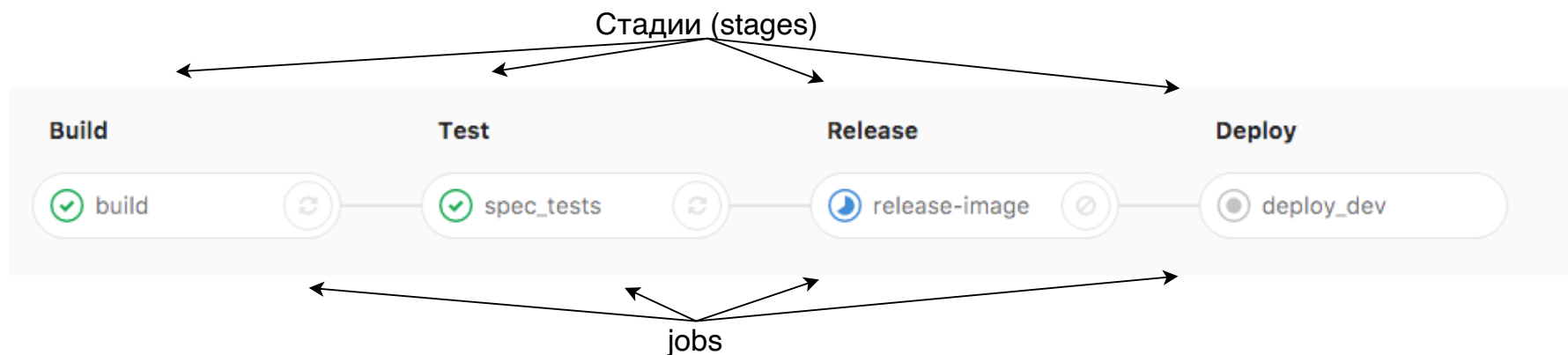
Клонируем код из репозитория

Склонируем репозитории:

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

Пример .gitlab-ci.yml

```
1  before_script:  # Команды, которые должны выполняться перед всем остальным
2    - ...
3
4  stages:         # Порядок стадий в pipeline
5    - build
6    - test
7
8  build:         # job
9    stage: build # в какой стадии job выполняется
10   script:
11     - ...      # Список команд, которые нужно выполнить
12
```



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:

```
1  stages:
2    - build
3
4  build:
5    stage: build
6    except:
7      - tags
8    script:
9      - docker build -t gcr.io/cd-k8s-236617/user-<your number>-ui:$CI_COMMIT_SHORT_SHA ./
10     - docker push gcr.io/cd-k8s-236617/user-<your number>-ui:$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:

```
1  stages:
2    - build
3    - test
4
5  ...
6
7  test:
8    stage: test
9    except:
10     - tags
11   script:
12     - 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 запускаться процесс сборки:

The screenshot shows the GitHub interface for a repository named 'ui'. At the top left, there is a profile icon 'U' and the repository name 'ui' with 'Project ID: 2'. To the right are buttons for 'Star' (0), 'Fork' (0), and 'Clone'. Below this, there are statistics: 'Add license', '4 Commits', '1 Branch', '0 Tags', and '287 KB Files'. A progress bar is visible, with a dark red section on the left and an orange section on the right. Below the progress bar, there are navigation options: 'master' (selected), 'ui / +', 'History', 'Find file', 'Web IDE', and a dropdown menu. The main content area shows a commit titled 'add build and test' by 'user01' from '1 minute ago'. A red box highlights a blue circular icon with a white play symbol, which is the build status icon for this commit. To the right of the icon is the commit hash 'c225620f' and a copy icon.

После клика на кружок должна быть подобная картина





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

parent [84b54bb1](#) master

🔗 🔄

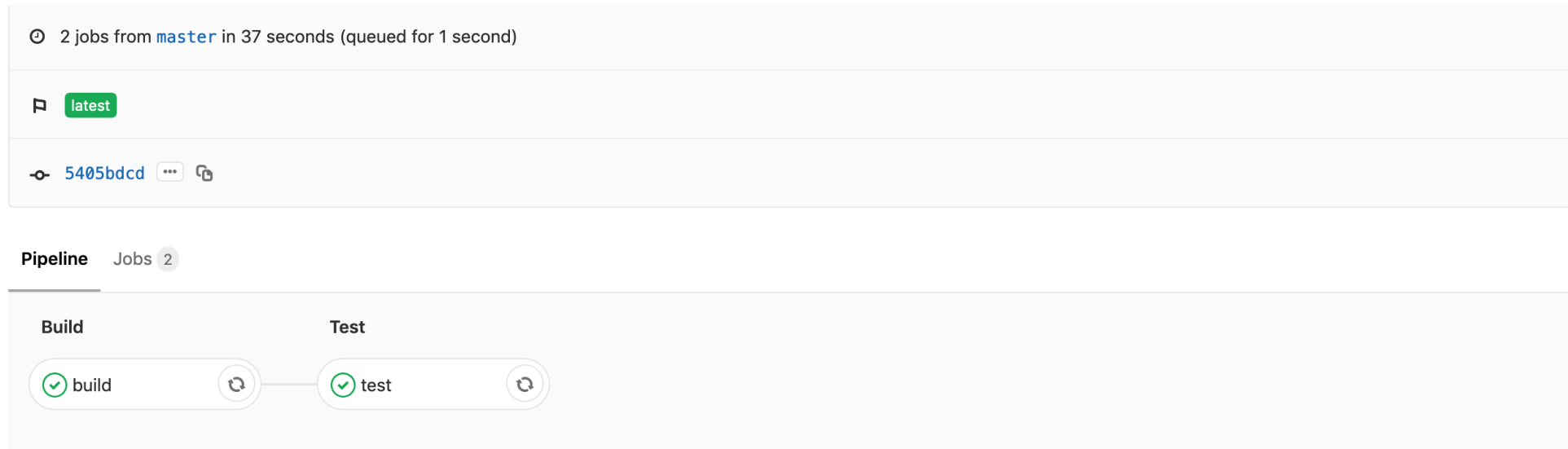
🔄 Pipeline #30 running with stages 🔄 ⏸

Changes **1** Pipelines **1**

Status	Pipeline	Commit	Stages	
	#30 by  latest	be3296ca  rack test	🔄 ⏸	

Кликнув на job можно увидеть pipeline

Все шаги должны быть зелеными.



The screenshot displays a GitLab pipeline interface. At the top, it shows '2 jobs from master in 37 seconds (queued for 1 second)'. Below this, there is a 'latest' tag and a commit hash '5405bdcd'. The main part of the image shows a pipeline diagram with two stages: 'Build' and 'Test'. The 'Build' stage contains a job named 'build' with a green checkmark and a refresh icon. The 'Test' stage contains a job named 'test' with a green checkmark and a refresh icon. A line connects the 'build' job to the 'test' job, indicating a sequential flow.

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.

```
1  stages:
2    ...
3    - release
4
5    ...
6  release:
7    stage: release
8    only:
9      - tags          # Условие запуска Job – только если есть метка Git
10   script:
11     - docker pull gcr.io/cd-k8s-236617/user-<your number>-ui:$CI_COMMIT_SHORT_SHA
12     - docker tag gcr.io/cd-k8s-236617/user-<your number>-ui:$CI_COMMIT_SHORT_SHA
13     gcr.io/cd-k8s-236617/user-<your number>-ui:$CI_COMMIT_TAG
14     - docker push gcr.io/cd-k8s-236617/user-<your number>-ui:$CI_COMMIT_TAG
```

Release

- Сохраняем изменения:

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

- Пушим в GitLab:

```
1 git push origin master
```

- Выставляем Git tag:

```
1 git tag v0.0.1
```

- Пушим метки в GitLab:

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

Самостоятельное задание

- Повторите настройку pipeline для репозиторийев `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.