Устройство Gitlab CI. Построение процесса непрерывной поставки

Домашнее задание

Проект microservices и проверка ДЗ

Создайте новую ветку в вашем microservices репозитории для выполнения данного ДЗ. Т.к. это первое задание, посвященное работе с Gitlab CI, то ветку назовите **gitlab-ci-1**.

Проверка данного ДЗ будет производиться через Pull Request ветки с ДЗ.

После того, как один из преподавателей сделает approve пул реквеста, ветку с ДЗ можно смерджить.

Цель задания

- Подготовить инсталляцию Gitlab CI
- Подготовить репозиторий с кодом приложения
- Описать для приложения этапы пайплайна
- Определить окружения

Инсталляция Gitlab CI

Для выполнения этого ДЗ нам потребуется развернуть свой Gitlab CI с помощью Docker.

CI-сервис является одним из ключевых инфраструктурных сервисов в процессе выпуска ПО и к его доступности, бесперебойной работе и безопасности должны предъявляться повышенные требования.

Мы же в академических целях упростим процедуру установки и настройки.

Новая и мощная виртуальная машина

Gitlab CI состоит из множества компонент и выполняет ресурсозатратную работу, например, компиляция приложений.

Нам потребуется создать в Google Cloud новую виртуальную машину со следующими параметрами

- 1 CPU
- 3.75GB RAM
- 50-100 GB HDD
- Ubuntu 16.04

В официальной документации описаны рекомендуемые характеристики сервера https://docs.gitlab.com/ce/install/requirements.html

Создаем виртуальную машину

Для создания сервера вы можете использовать любой из удобных вам способов

- Веб-интерфейс облака Google
- Terraform
- Утилиту gcloud
- Docker-machine

Также нужно разрешить подключение по HTTP/HTTPS

Пример

Name
gitlab-ci

Zone
europe-west3-a

Machine type
1 vCPU 3.75 GB memory Customize
Upgrade your account to create instances with up to 64 cores

Boot disk
New 100 GB standard persistent disk
Image
Ubuntu 16.04 LTS Change

Identity and API access
Service account
Compute Engine default service account

Access scopes
- Allow default access
- Allow full access to all Cloud APIs
- Set access for each API

Firewall
Add tags and firewall rules to allow specific network traffic from the Internet
- [x] Allow HTTP traffic
- [x] Allow HTTPS traffic

Omnibus

Для запуска Gitlab CI мы будем использовать omnibus-установку, у этого подхода есть как свои плюсы, так и минусы.

Основной плюс для нас в том, что мы можем быстро запустить сервис и сконцентрироваться на процессе непрерывной поставки.

Минусом такого типа установки является то, что такую инсталляцию тяжелее эксплуатировать и дорабатывать, но долговременная эксплуатация этого сервиса не входит в наши цели.

Более подробно об этом опять же в документации

https://docs.gitlab.com/omnibus/README.html

https://docs.gitlab.com/omnibus/docker/README.html

Ставим Docker

На нашем новом сервере для Gitlab CI должен каким-то образом появится Docker.

Вы уже знаете, как установить Docker на сервер

- Руками :)
- С помощью docker-machine
- С помощью Ansible

```
# если руками, то из под root на новом сервере выполнить
# curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
# add-apt-repository "deb https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
# apt-get update
# apt-get install docker-ce docker-compose
```

Подготавливаем окружение

Помимо всего прочего нам нужно на новом сервере создать необходимые директории и подготовить docker-compose.yml

```
# mkdir -p /srv/gitlab/config /srv/gitlab/data /srv/gitlab/logs
# cd /srv/gitlab/
# touch docker-compose.yml
```

docker-compose.yml

Обратите внимание на **external_url ‘http://<YOUR-VM-IP>’**

Нужно заменить **<YOUR-VM-IP>** внешним IP адресом, который Google присвоил вашему серверу.

```
web:
  image: 'gitlab/gitlab-ce:latest'
  restart: always
  hostname: 'gitlab.example.com'
  environment:
    GITLAB_OMNIBUS_CONFIG: |
      external_url 'http://<YOUR-VM-IP>'
  ports:
    - '80:80'
    - '443:443'
    - '2222:22'
  volumes:
    - '/srv/gitlab/config:/etc/gitlab'
    - '/srv/gitlab/logs:/var/log/gitlab'
    - '/srv/gitlab/data:/var/opt/gitlab'
```

ссылка на gist

Запускаем Gitlab CI

Запускаем и ждем, пока скачаются образы и Gitlab CI запустится и произведет первичную настройку

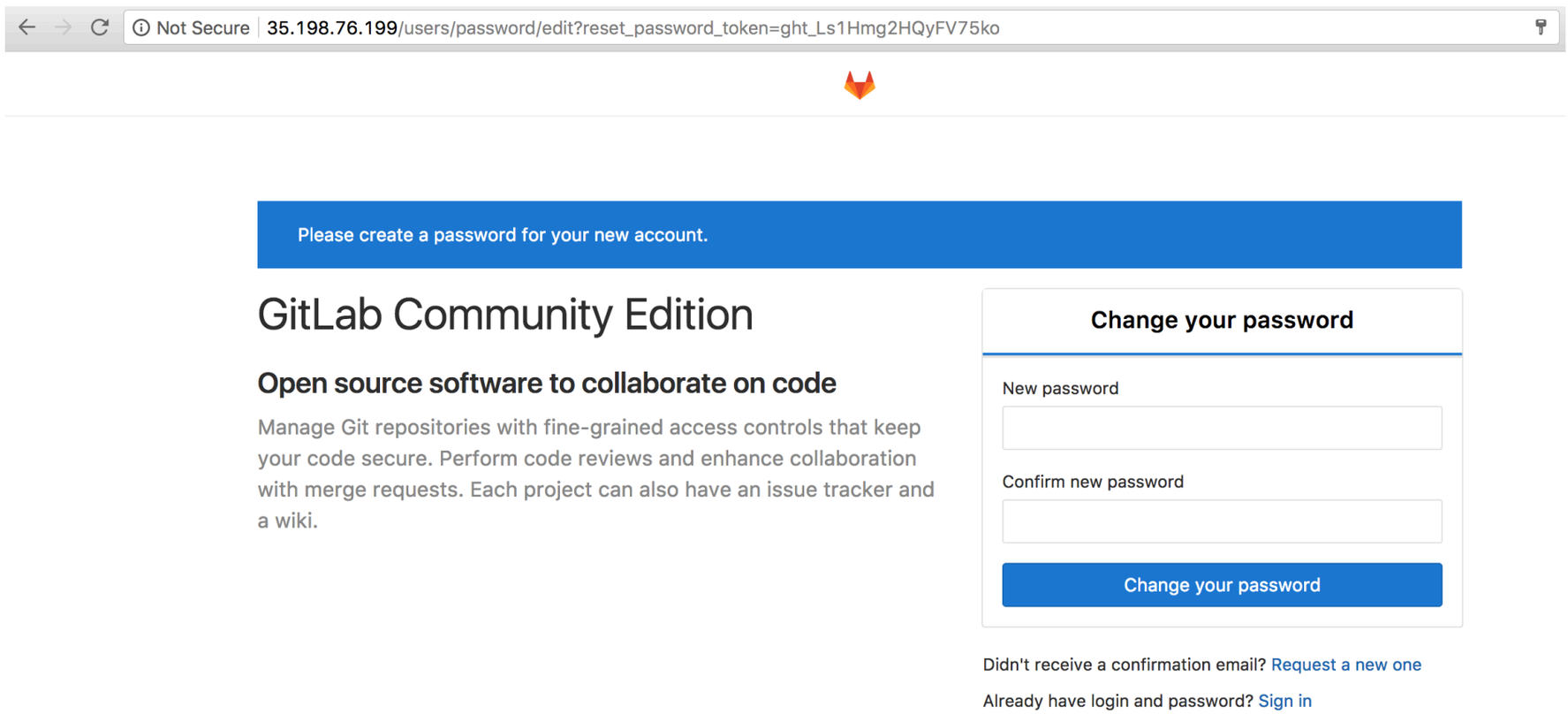
```
# В той же директории, где docker-compose.yml ( /srv/gitlab )

# docker-compose up -d
```

Для первого запуска Gitlab CI необходимо подождать несколько минут, пока он стартует можно почитать, откуда мы взяли содержимое файла docker-compose.yml

https://docs.gitlab.com/omnibus/docker/README.html#install-gitlab-using-docker-compose

Проверяем

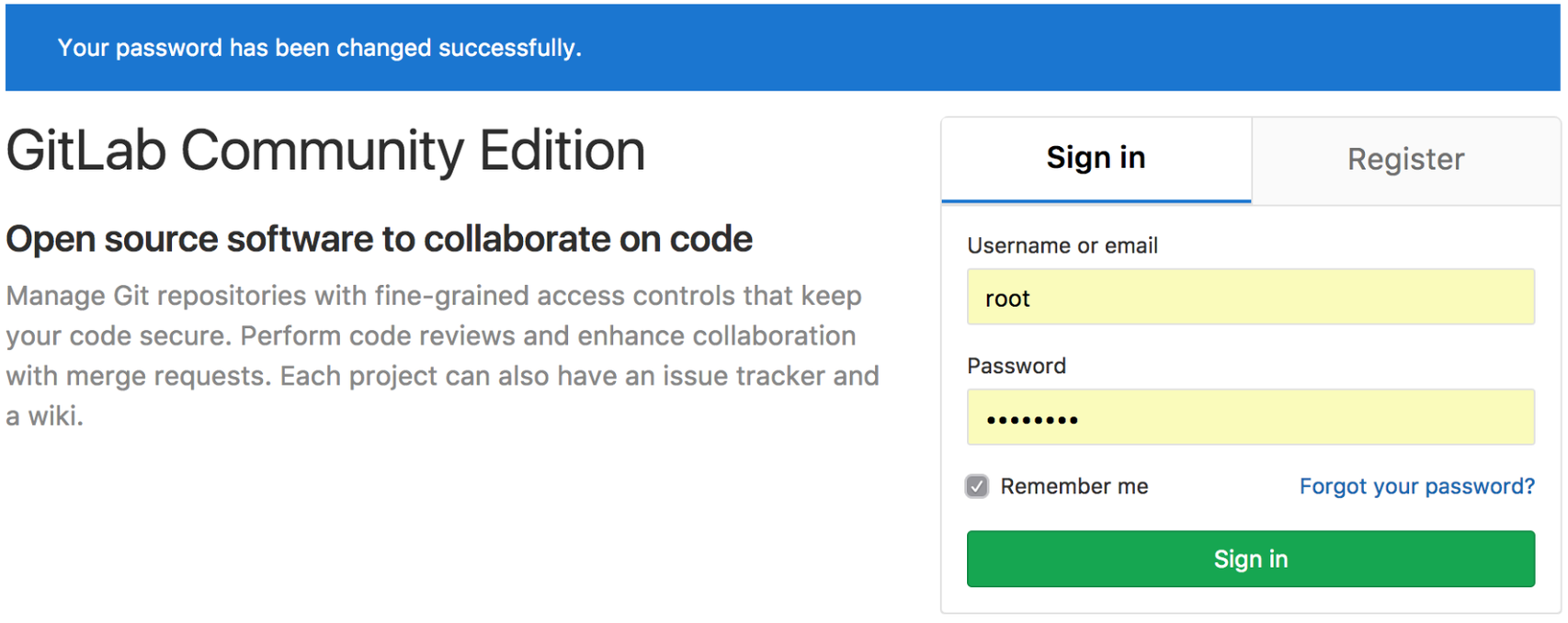
Если все прошло успешно, то вы сможете в браузере перейти на http://<your-vm-ip> и увидеть там:



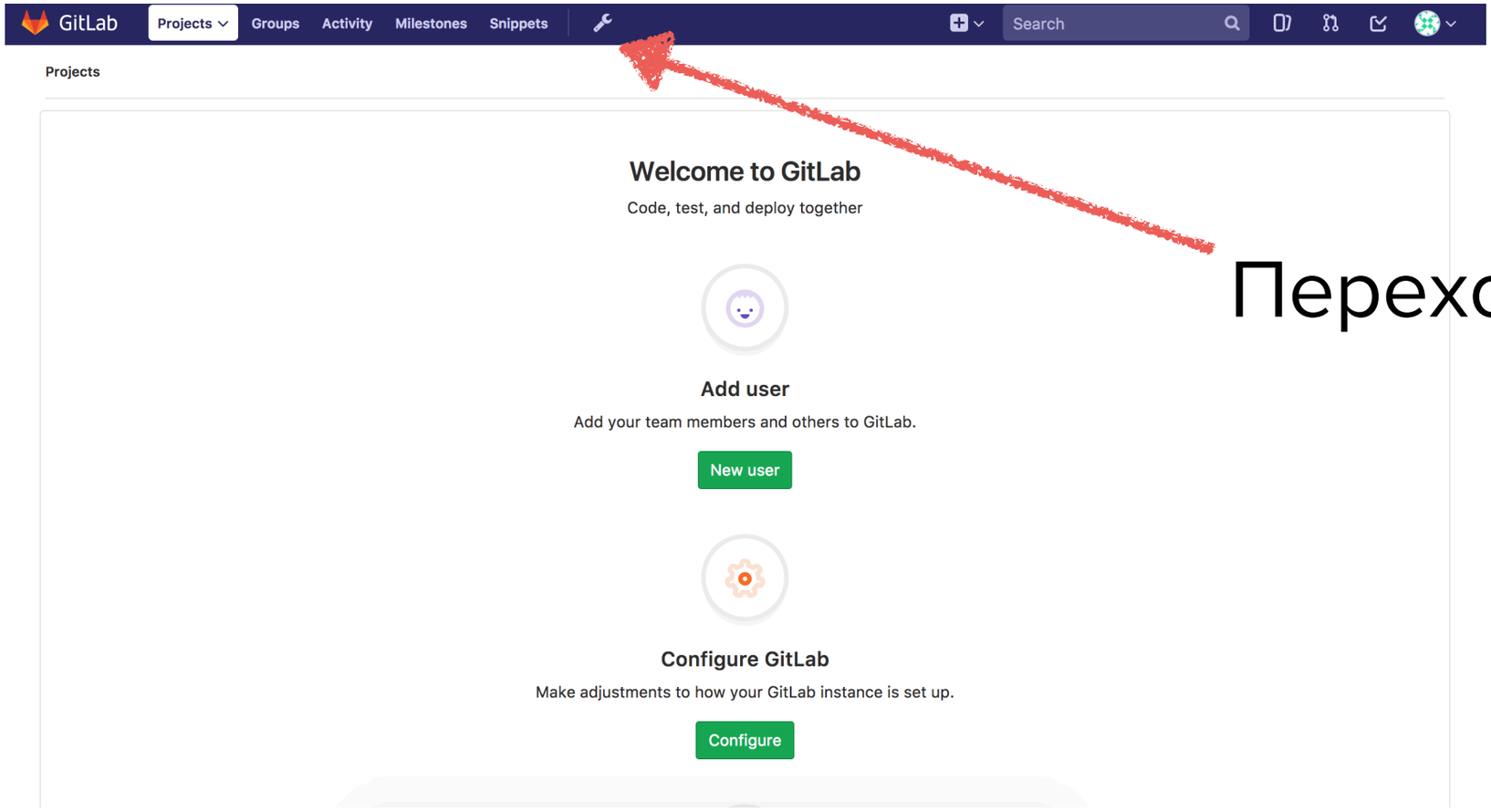
Указываем пароль для root-аккаунта

На этой же странице Gitlab CI предлагает вам указать пароль для административного аккаунта (root)

После ввода нового пароля вы сможете авторизоваться.



Настройки

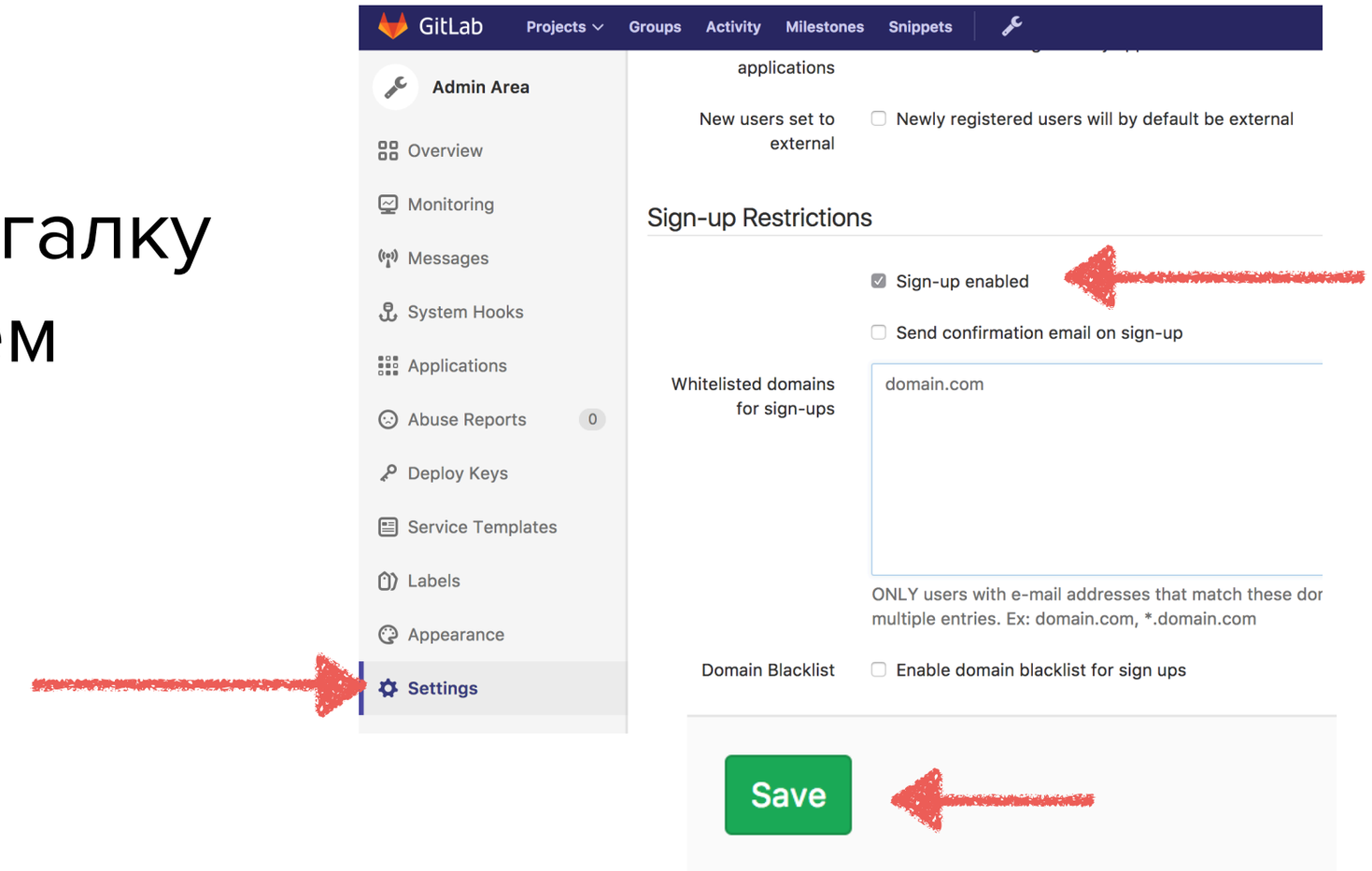


Переходим в админку

Настройки

Выключаем регистрацию новых пользователей

1. Settings
2. Снимаем галку
3. Сохраняем



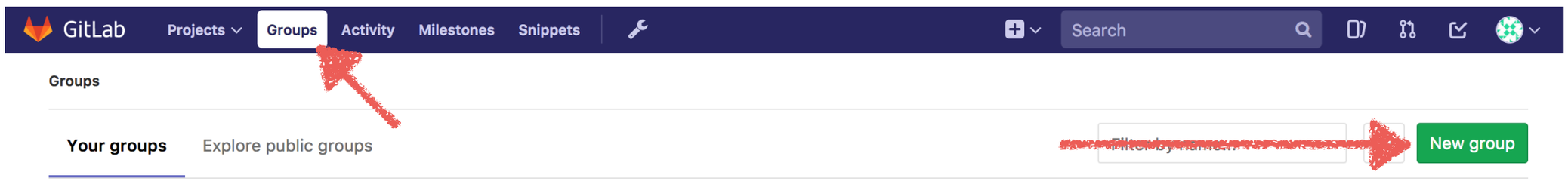
Опишем первый проект

Из лекции:

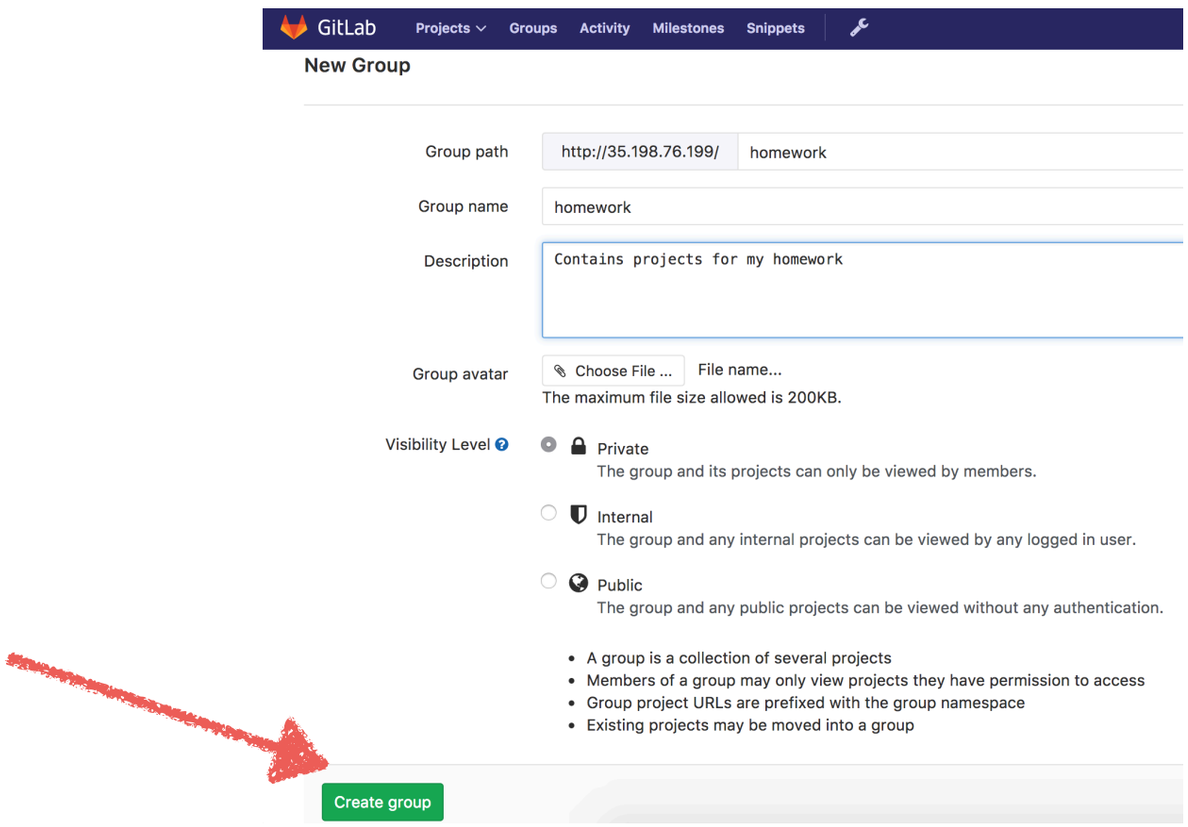
- Каждый проект в Gitlab CI принадлежит к группе проектов
- В проекте может быть определен CI/CD пайплайн
- Задачи (jobs) входящие в пайплайн должны исполняться на runners

Создадим группу

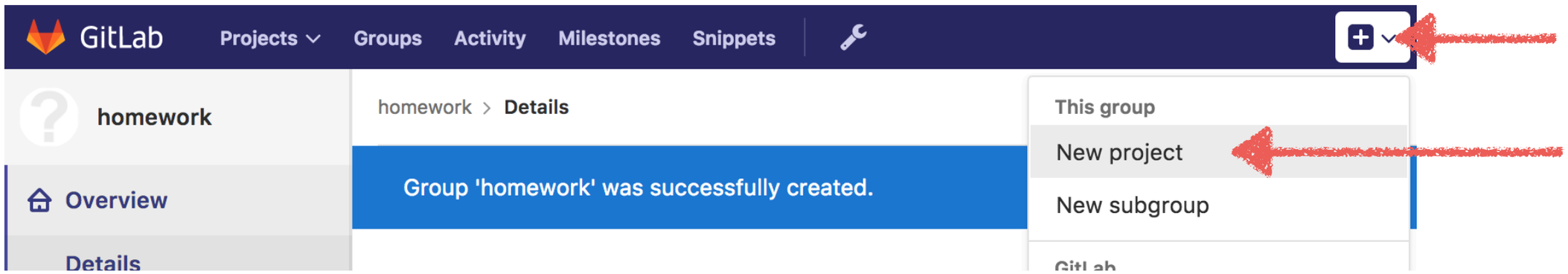
- Переходим в соответствующий раздел

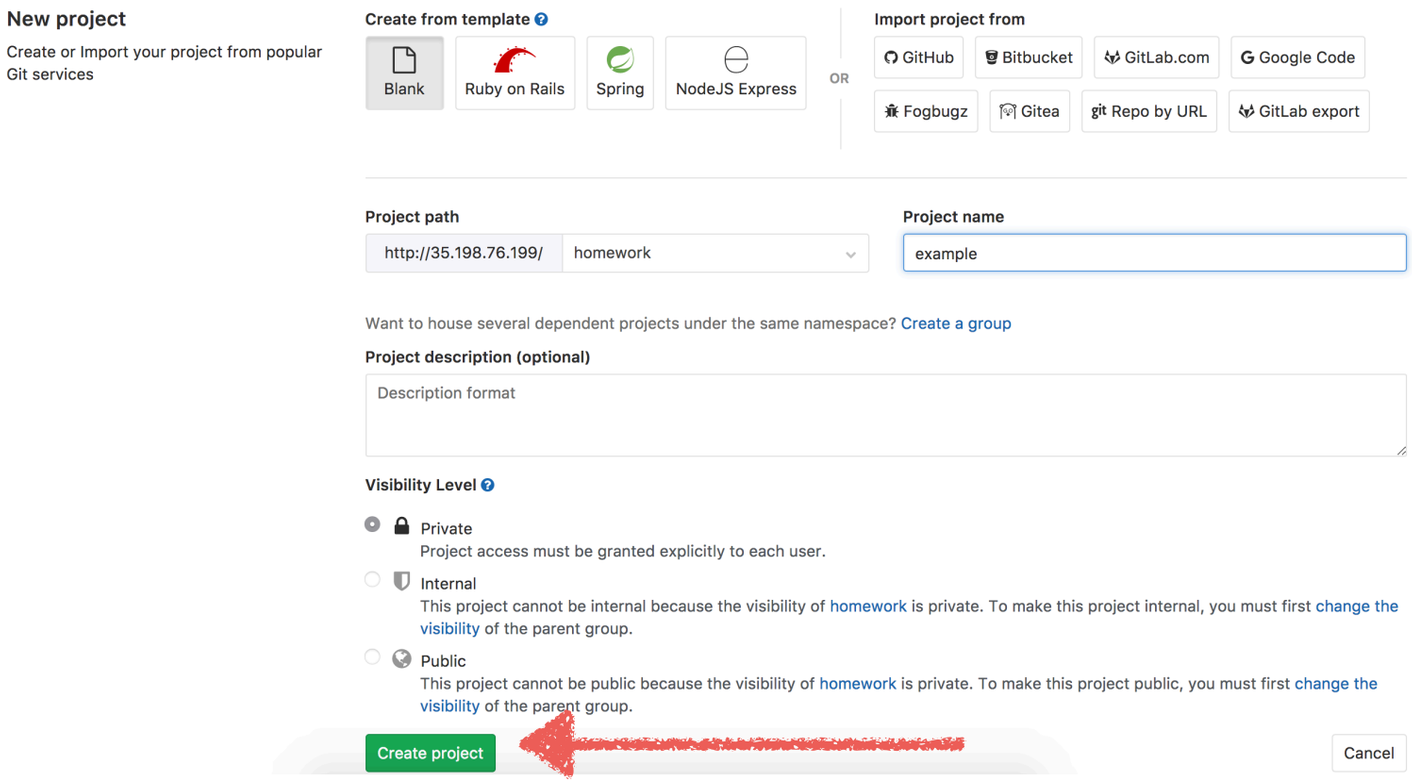


- Заполняем описание



Создадим проект





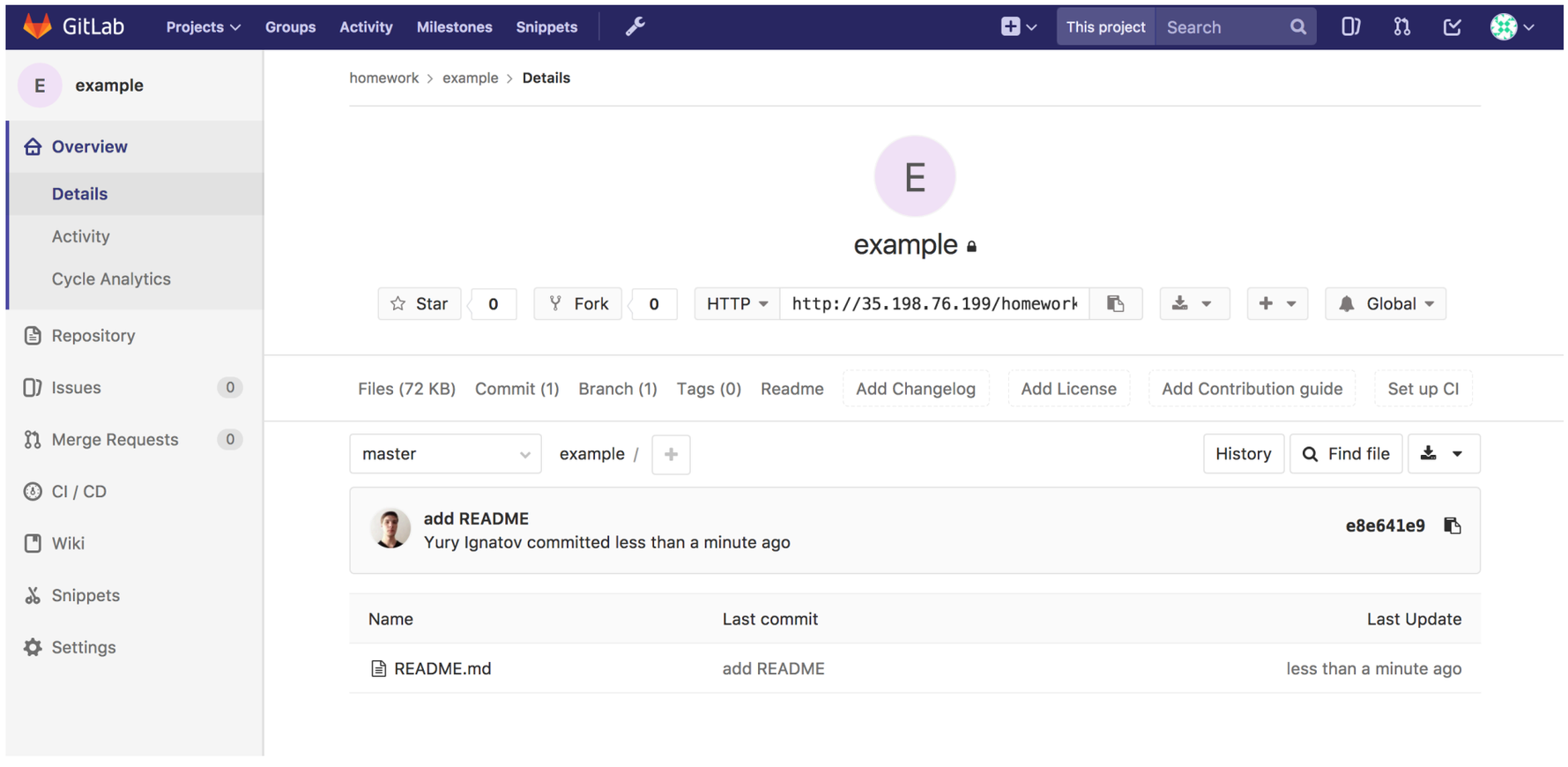
Добавляем remote в <username>_microservices

```
> git checkout -b gitlab-ci-1

> git remote add gitlab http://<your-vm-ip>/homework/example.git

> git push gitlab gitlab-ci-1
```

CI/CD Pipeline

Теперь мы можем переходить к определению CI/CD Pipeline для проекта.

Чтобы сделать это нам нужно добавить в репозиторий файл **.gitlab-ci.yml**



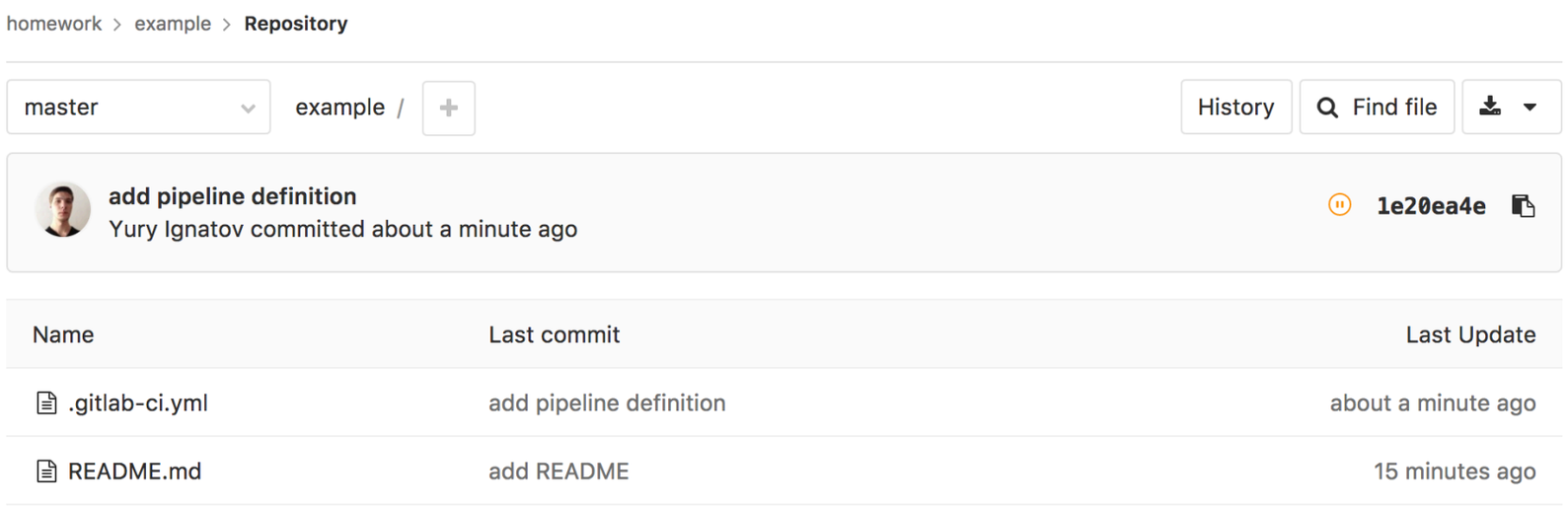
CI/CD Pipeline

Содержимое файла .gitlab-ci.yml (ссылка на gist)

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

build_job:
  stage: build
  script:
    - echo 'Building'

test_unit_job:
  stage: test
  script:
    - echo 'Testing 1'

test_integration_job:
  stage: test
  script:
    - echo 'Testing 2'

deploy_job:
  stage: deploy
  script:
    - echo 'Deploy'
```

После чего сохраняем файл

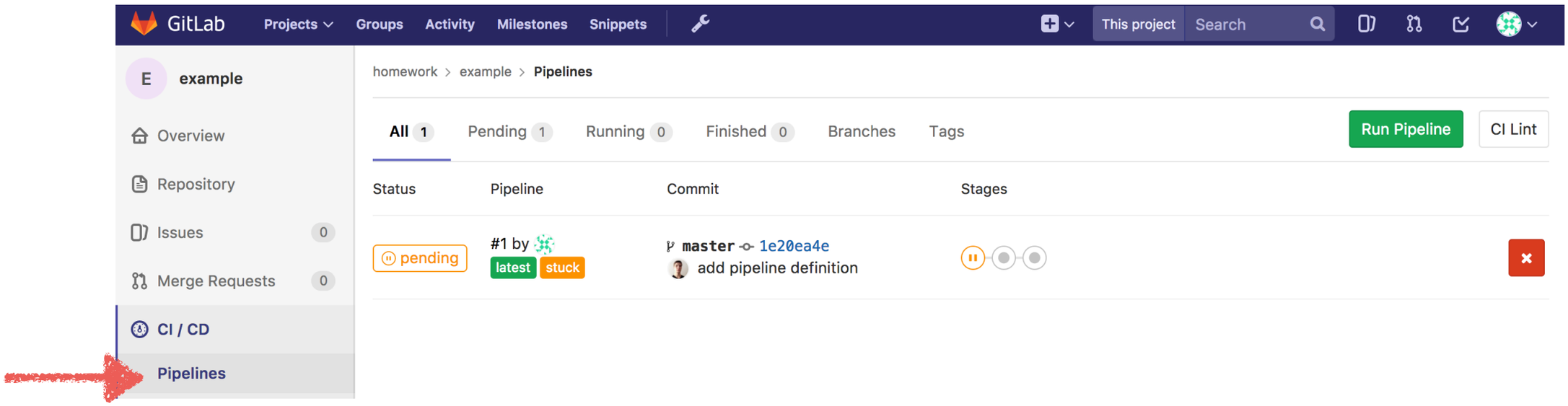
```
› git add .gitlab-ci.yml
› git commit -m 'add pipeline definition'
› git push gitlab gitlab-ci-1
```



CI/CD Pipeline

Теперь если перейти в раздел CI/CD мы увидим, что пайплайн готов к запуску

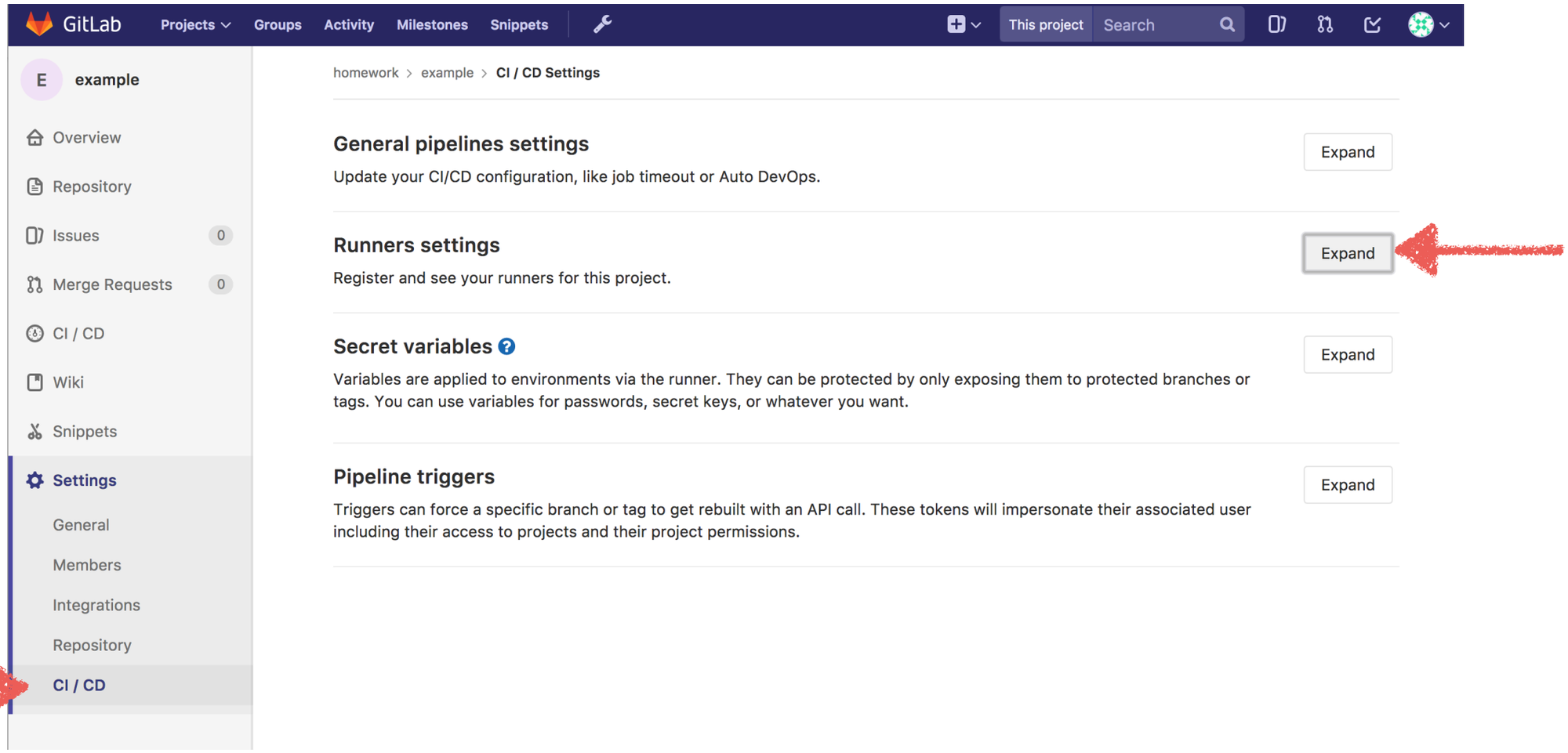
Но находится в статусе pending / stuck так как у нас нет runner



Запустим Runner и зарегистрируем его в интерактивном режиме

Runner

Перед тем, как запускать и регистрировать runner нужно получить токен



Runner

Нужно скопировать, токен пригодится нам при регистрации

Specific Runners

How to setup a specific Runner for a new project

1. Install a Runner compatible with GitLab CI (checkout the GitLab Runner section for information on how to install it).
2. Specify the following URL during the Runner setup: `http://35.198.76.199/`
3. Use the following registration token during setup: `9Sdmydz9ZX8eYA-rA1-V`
4. Start the Runner!

Shared Runners

GitLab Shared Runners execute code of different projects on the same Runner unless you configure GitLab Runner Autoscale with MaxBuilds 1 (which it is on GitLab.com).

Disable shared Runners for this project

This GitLab server does not provide any shared Runners yet. Please use the specific Runners or ask your administrator to create one.

Runner

На сервере, где работает Gitlab CI выполните команду:

```
docker run -d --name gitlab-runner --restart always \
  -v /srv/gitlab-runner/config:/etc/gitlab-runner \
  -v /var/run/docker.sock:/var/run/docker.sock \
  gitlab/gitlab-runner:latest
```

После запуска Runner нужно зарегистрировать, это можно сделать командой:

```
root@gitlab-ci:~# docker exec -it gitlab-runner gitlab-runner register --run-untagged --locked=false
Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.com/):
http://<YOUR-VM-IP>/
Please enter the gitlab-ci token for this runner:
<TOKEN>
Please enter the gitlab-ci description for this runner:
[38689f5588fe]: my-runner
Please enter the gitlab-ci tags for this runner (comma separated):
linux,xenial,ubuntu,docker
Please enter the executor:
docker
Please enter the default Docker image (e.g. ruby:2.1):
alpine:latest
Runner registered successfully.
```

Runner

Если все получилось, то в настройках вы увидите новый runner

Specific Runners

How to setup a specific Runner for a new project

1. Install a Runner compatible with GitLab CI (checkout the GitLab Runner section for information on how to install it).
2. Specify the following URL during the Runner setup: `http://35.198.76.199/`
3. Use the following registration token during setup: `9Sdmydz9ZX8eYA-rA1-V`
4. Start the Runner!

Shared Runners

GitLab Shared Runners execute code of different projects on the same Runner unless you configure GitLab Runner Autoscale with MaxBuilds 1 (which it is on GitLab.com).

Disable shared Runners for this project

This GitLab server does not provide any shared Runners yet. Please use the specific Runners or ask your administrator to create one.

Runners activated for this project

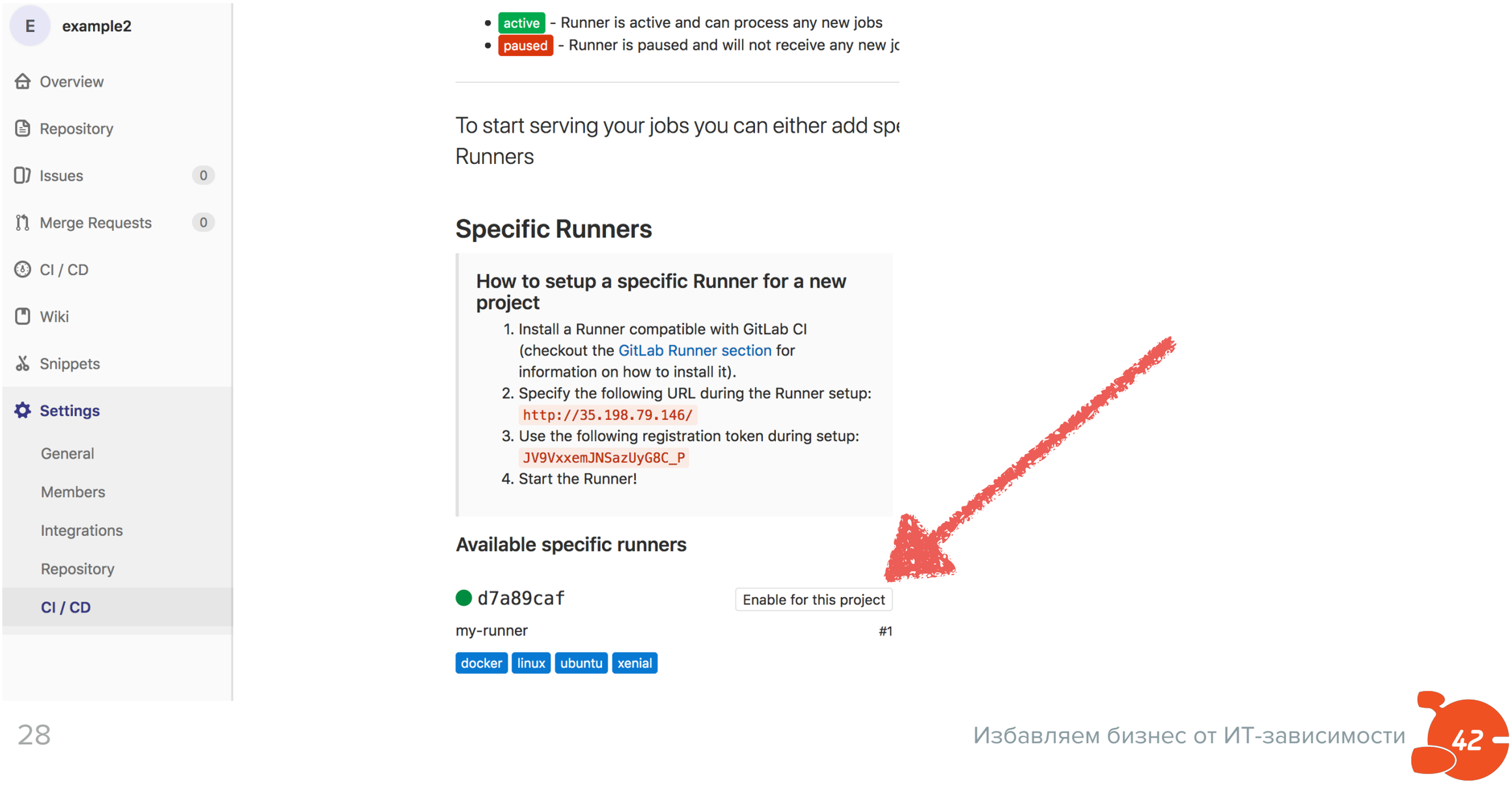
621614a8 Remove Runner

my-runner #1

docker linux ubuntu xenial

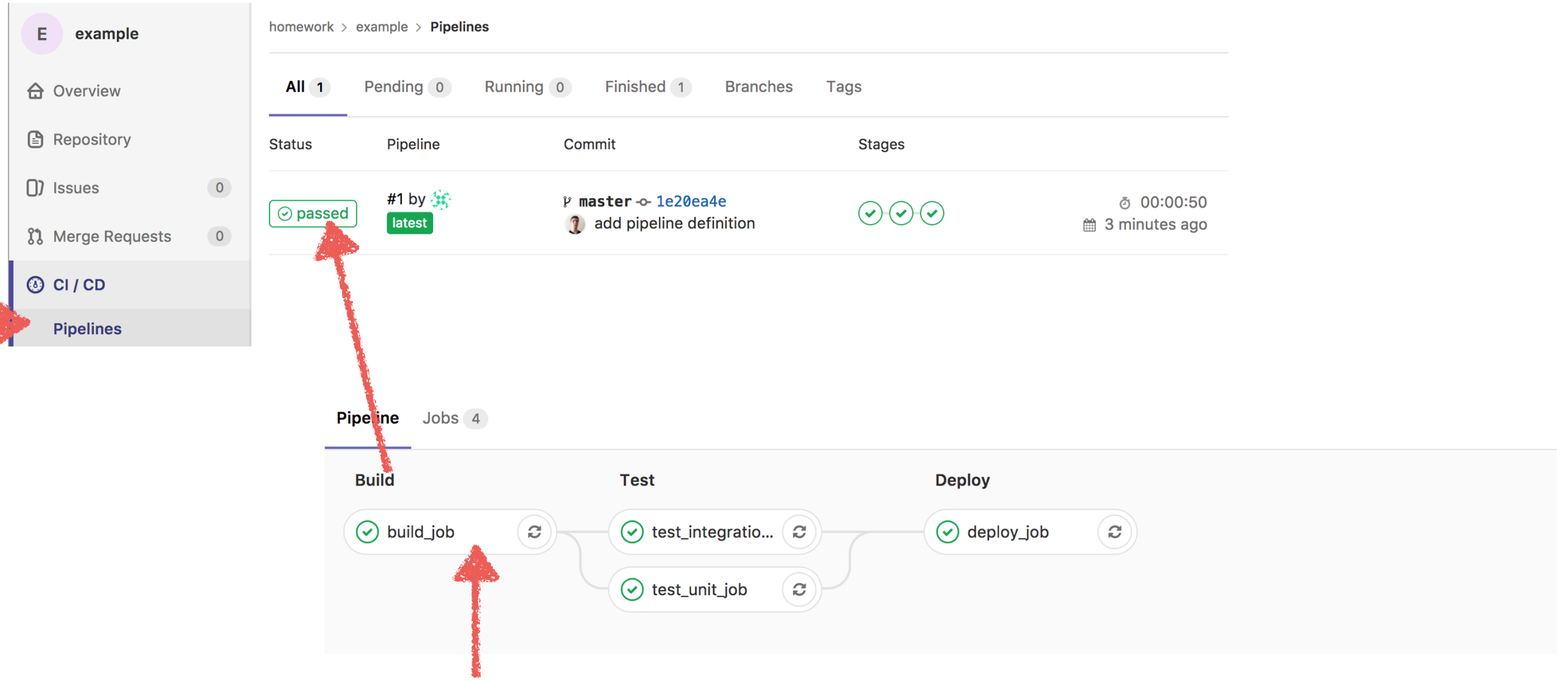
Проверьте статус runner

Исправить это можно включив runner для проекта



CI/CD Pipeline

После добавления Runner пайплайн должен был запуститься



Сюда тоже можно тыкнуть :)

Добавим тестирование приложения reddit в pipeline

Добавим исходный код reddit в репозиторий

> git clone https://github.com/express42/reddit.git && rm -rf ./reddit/.git
> git add reddit/
> git commit -m “Add reddit app”
> git push **gitlab gitlab-ci-1**

Тестируем reddit

Изменим описание пайплайна в .gitlab-ci.yml

```
image: ruby:2.4.2

stages:
...

variables:
  DATABASE_URL: 'mongodb://mongo/user_posts'

before_script:
  - cd reddit
  - bundle install

...
test_unit_job:
  stage: test
  services:
    - mongo:latest
  script:
    - ruby simpletest.rb
...
...
```

Приложение reddit

В описании pipeline мы добавили вызов теста в файле simpletest.rb, нужно создать его **в папке reddit**

ссылка на gist

```
require_relative './app'
require 'test/unit'
require 'rack/test'

set :environment, :test

class MyAppTest < Test::Unit::TestCase
  include Rack::Test::Methods

  def app
    Sinatra::Application
  end

  def test_get_request
    get '/'
    assert last_response.ok?
  end
end
```


Приложение reddit

Последним шагом нам нужно добавить библиотеку для тестирования в **reddit/Gemfile** приложения.

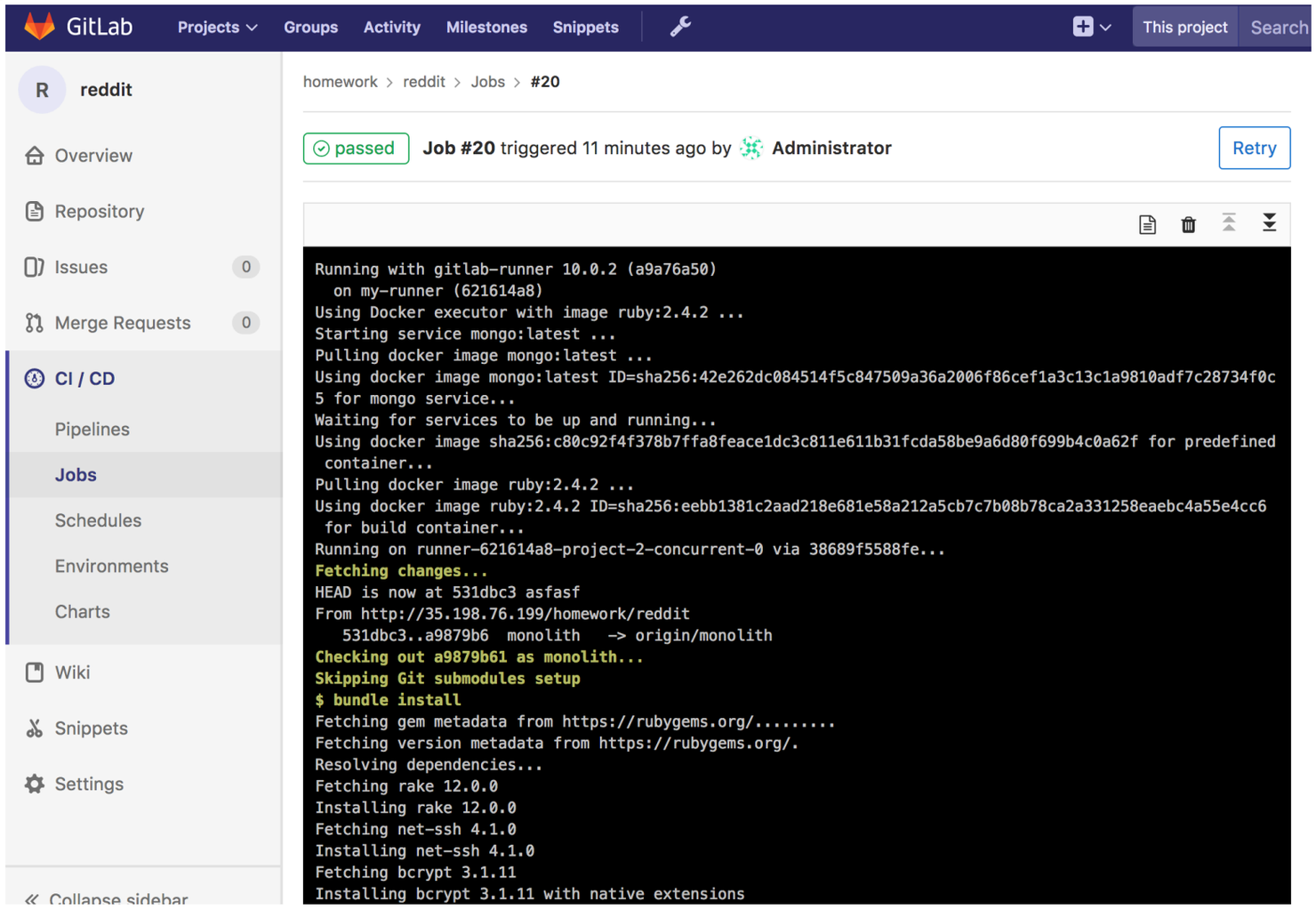
Добавим `gem 'rack-test'`

Gemfile

```
...  ...  @@ -6,6 +6,7 @@ gem 'bson_ext'
 6    6   gem 'bcrypt'
 7    7   gem 'puma'
 8    8   gem 'mongo'
      9 + gem 'rack-test'
 9   10
10   11   group :development do
11   12       gem 'capistrano',         require: false
...  ...
```

Приложение reddit

Теперь на каждое изменение в коде приложения будет запущен тест



Опишем окружения

Вернемся к академическому пайплайну, который описывает шаги сборки, тестирования и деплоймента.

В прошлом занятии мы создали job с названием deploy_job, но не разбирались что и куда будет задеплоено.

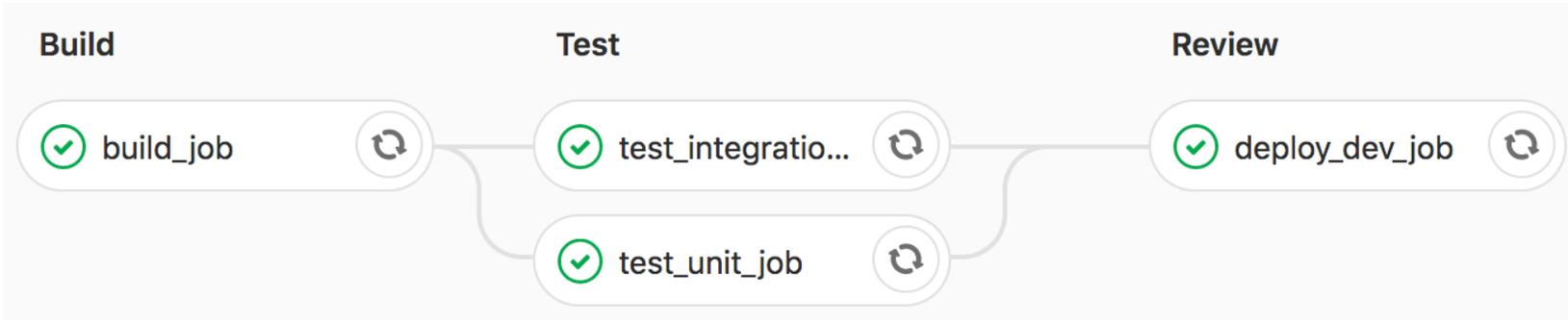
Dev-окружение

Изменим пайплайн таким образом, чтобы job deploy стал определением окружения dev, на которое условно будет выкатываться каждое изменение в коде проекта.

После изменения файла .gitlab-ci.yml не забывайте зафиксировать изменение в git и отправить изменения на сервер. (git commit и git push **gitlab gitlab-ci-1**)

Dev-окружение

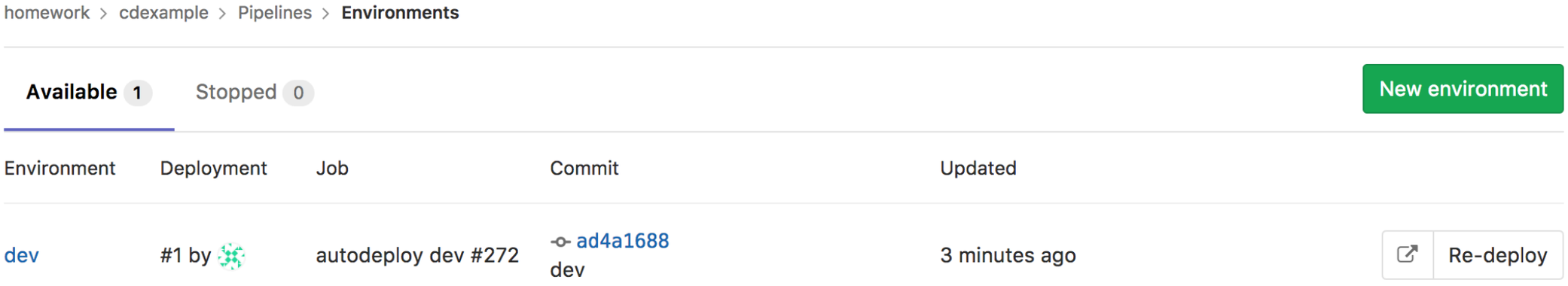
```
stages:
  - build
  - test
  - review

...
build_job:
...

test_unit_job:
...

test_integration_job:
...

deploy_dev_job:
  stage: review
  script:
    - echo 'Deploy'
  environment:
    name: dev
    url: http://dev.example.com
```



1. Переименуем deploy stage в review.
2. deploy_job заменим на deploy_dev_job
3. Добавим environment

Dev-окружение

Если после успешного завешения пайплайна с определением окружения перейти в CI/CD > Environments, то там появится определение первого окружения.



Staging и Production

Если на dev мы можем выкатывать последнюю версию кода, то к production окружению это может быть неприменимо, если, конечно, вы не стремитесь к continuous deployment.

Определим два новых этапа: stage и production, первый будет содержать job имитирующий выкатку на staging окружение, второй на production окружение.

Определим эти job таким образом, чтобы они запускались с кнопки

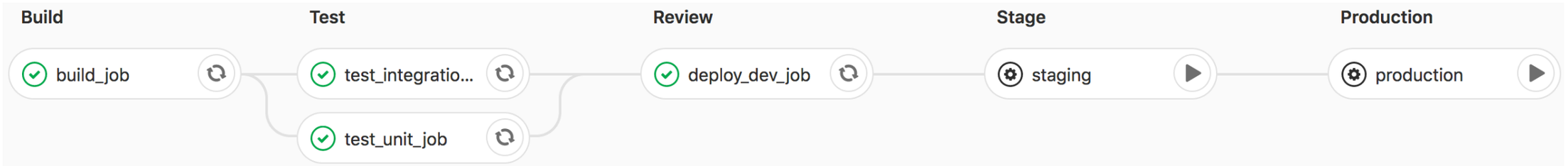
Staging и Production

```
stages:
  - build
  - test
  - review
  - stage
  - production

...

deploy_dev_job:
  stage: review
  script:
    - echo 'Deploy'
  environment:
    name: dev
    url: http://dev.example.com

staging:
  stage: stage
  when: manual
  script:
    - echo 'Deploy'
  environment:
    name: stage
    url: https://beta.example.com

production:
  stage: production
  when: manual
  script:
    - echo 'Deploy'
  environment:
    name: production
    url: https://example.com
```

when: manual – говорит о том, что job должен быть запущен человеком из UI.

Staging и Production

Теперь пайплайн должен выглядеть следующим образом



На странице окружений должны появиться оружения staging и production

Environment	Deployment	Job	Commit	Updated	
dev	#2 by	autodeploy dev #276	d880f67d asfasfas	6 minutes ago	Re-deploy
production			No deployments yet		
stage			No deployments yet		

Условия и ограничения

Обычно, на production окружение выводится приложение с явно зафиксированной версией (например, 2.4.10).

Добавим в описание pipeline директиву, которая не позволит нам выкатить на staging и production код, не помеченный с помощью тэга в git.

Условия и ограничения

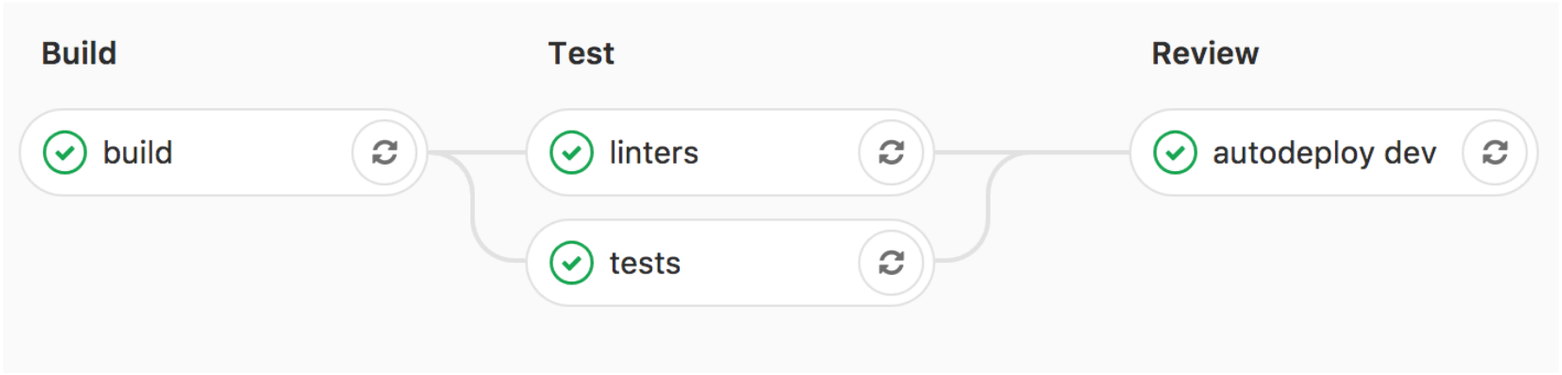
```
stages:
  - build
  - test
  - review
  - stage
  - production

… … …

stage:
  stage: stage
  when: manual
  only:
    - /^\d+\.\d+\.\d+/
 script:
    - echo 'Deploy'
  environment:
    name: stage
    url: https://beta.example.com

production:
  stage: production
  when: manual
  only:
    - /^\d+\.\d+\.\d+/
 script:
    - echo 'Deploy'
  environment:
    name: production
    url: https://example.com
```

Директива **only** описывает список условий, которые должны быть истинны, чтобы job мог запуститься.

Регулярное выражение слева означает, что должен стоять semver тэг в git, например, **2.4.10**

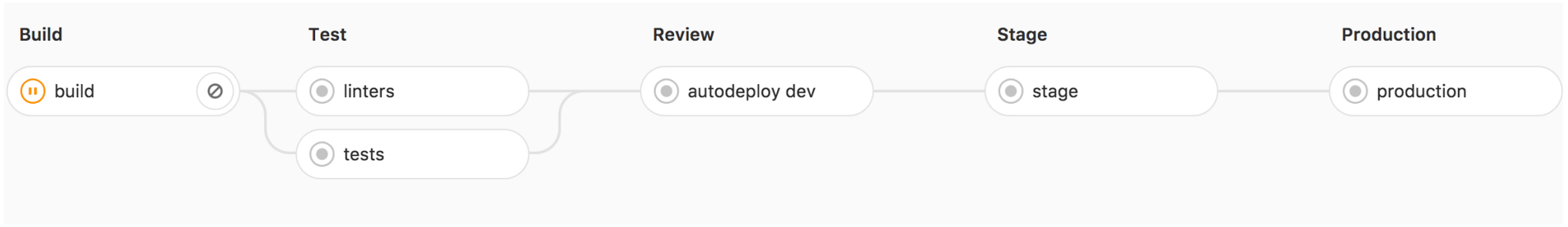
Условия и ограничения

Изменение без указания тэга запустят пайплайн без job staging и production



Изменение, помеченное тэгом в git запустит полный пайплайн

```
git commit -a -m '#4 add logout button to profile page'
git tag 2.4.10
git push gitlab gitlab-ci-1 --tags
```



Динамические окружения

Gitlab CI позволяет определить динамические окружения, это мощная функциональность позволяет вам иметь выделенный стенд для, например, каждой feature-ветки в git.

Определяются динамические окружения с помощью переменных, доступных в **.gitlab-ci.yml**

Динамические окружения

```
stages:
  - build
  - test
  - review
  - stage
  - production

. . .

branch review:
  stage: review
  script: echo "Deploy to $CI_ENVIRONMENT_SLUG"
  environment:
    name: branch/$CI_COMMIT_REF_NAME
    url: http://$CI_ENVIRONMENT_SLUG.example.com
  only:
    - branches
  except:
    - master

staging:
  stage: stage
  when: manual
… … …
```

Этот job определяет динамическое окружение для каждой ветки в репозитории, кроме ветки master

Динамические окружения

Теперь, на каждую ветку в git отличную от master Gitlab CI будет определять новое окружение.

Если создать ветки new-feature и bugfix, то на странице окружений будет следующее:

Environment	Deployment	Job
dev	#3 by	autodeploy dev #420
review 2		
review/bugfix		
review/new-fe...		

Задание с *

В шаг build добавить сборку контейнера с приложением reddit

Деплойте контейнер с reddit на созданный для ветки сервер.

Задачи со *

- Продумайте автоматизацию развертывания и регистрации Gitlab CI Runner. В больших организациях количество Runners может превышать 50 и более, сетапить их руками становится проблематично.
  Реализацию функционала добавьте в репозиторий в папку **gitlab-ci**;

- Настройте интеграцию вашего Pipeline с тестовым Slack-чатом, который вы использовали ранее. Для этого перейдите в **Project Settings > Integrations > Slack notifications**. Нужно установить **active**, выбрать события и заполнить поля с URL вашего Slack webhook.
  Добавьте ссылку на канал в слаке, в котором можно проверить работу оповещений, в файл README.md;

Удаляем VM

- Данная инсталляция Gitlab нам больше не нужна, VM можно удалить

- Создайте папку **gitlab-ci** в корне репозтория. Добавьте в нее docker-compose.yml для развертывания Gitlab.

- Отправьте свой код в репозиторий на github в ветку **gitlab-ci-1.**

```
> git push origin gitlab-ci-1
```

Проверка ДЗ

- Результаты вашей работы находятся в ветке **gitlab-ci-1** вашего microservices репозитория;

- В README внесите описание того, что сделано;

- Создайте Pull Request к ветке master (**описание PR нужно заполнять**);

- В ревьюверы можно никого не добавлять;

- Добавьте "Labels" **GitlabCI** и **gitlab-ci-1** к вашему Pull Request (если готовых лейблов нет, создайте);

- После того, как один из преподавателей сделает approve пул реквеста, ветку с ДЗ можно смерджить и закрыть PR.