

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 ?

Zone ?


europa-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

- Allow HTTP traffic
- 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 появиться 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 )
```

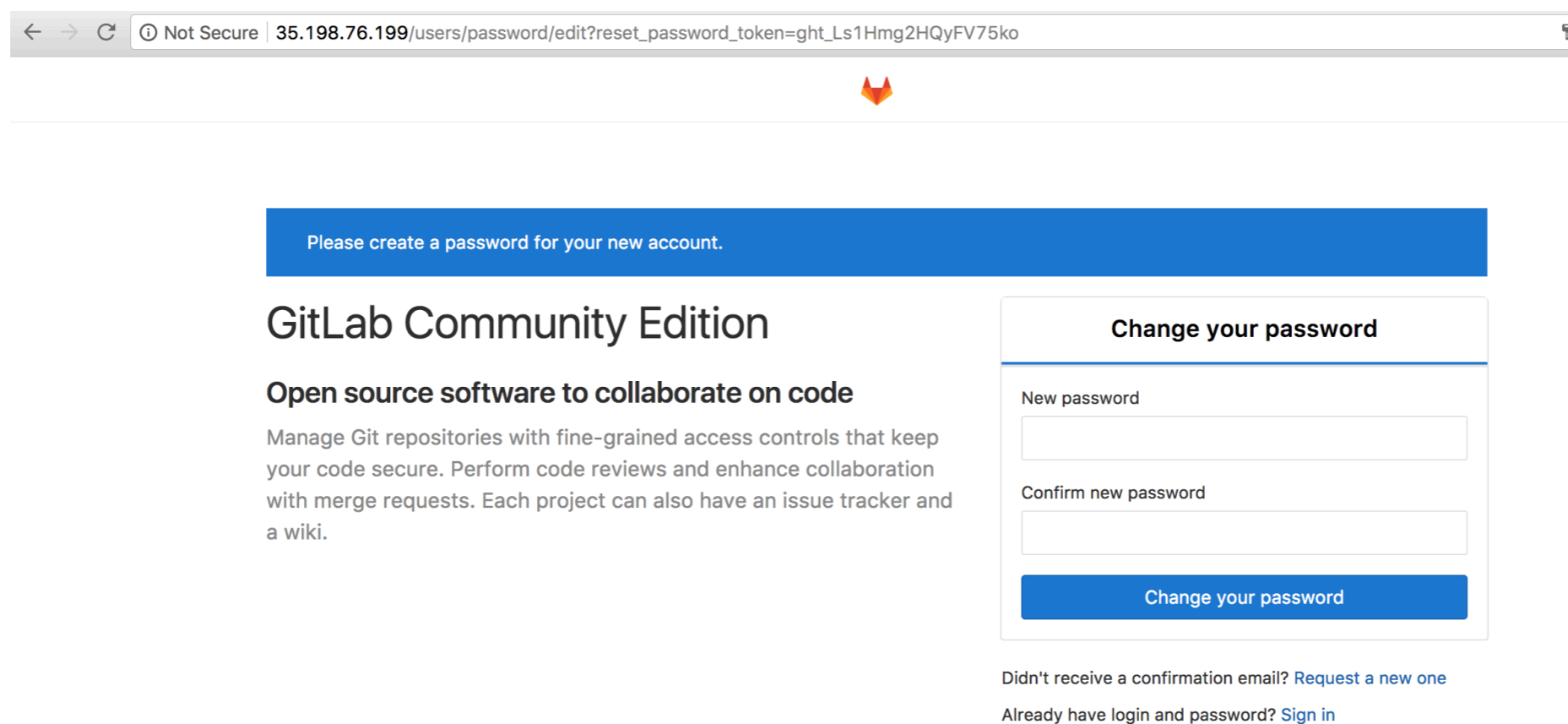
```
# docker-compose up -d
```

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:



← → ↻ ⓘ Not Secure | 35.198.76.199/users/password/edit?reset_password_token=ght_Ls1Hmg2HQyFV75ko

Please create a password for your new account.

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.

Change your password

New password

Confirm new password

[Change your password](#)

Didn't receive a confirmation email? [Request a new one](#)

Already have login and password? [Sign in](#)

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.

Your password has been changed successfully.

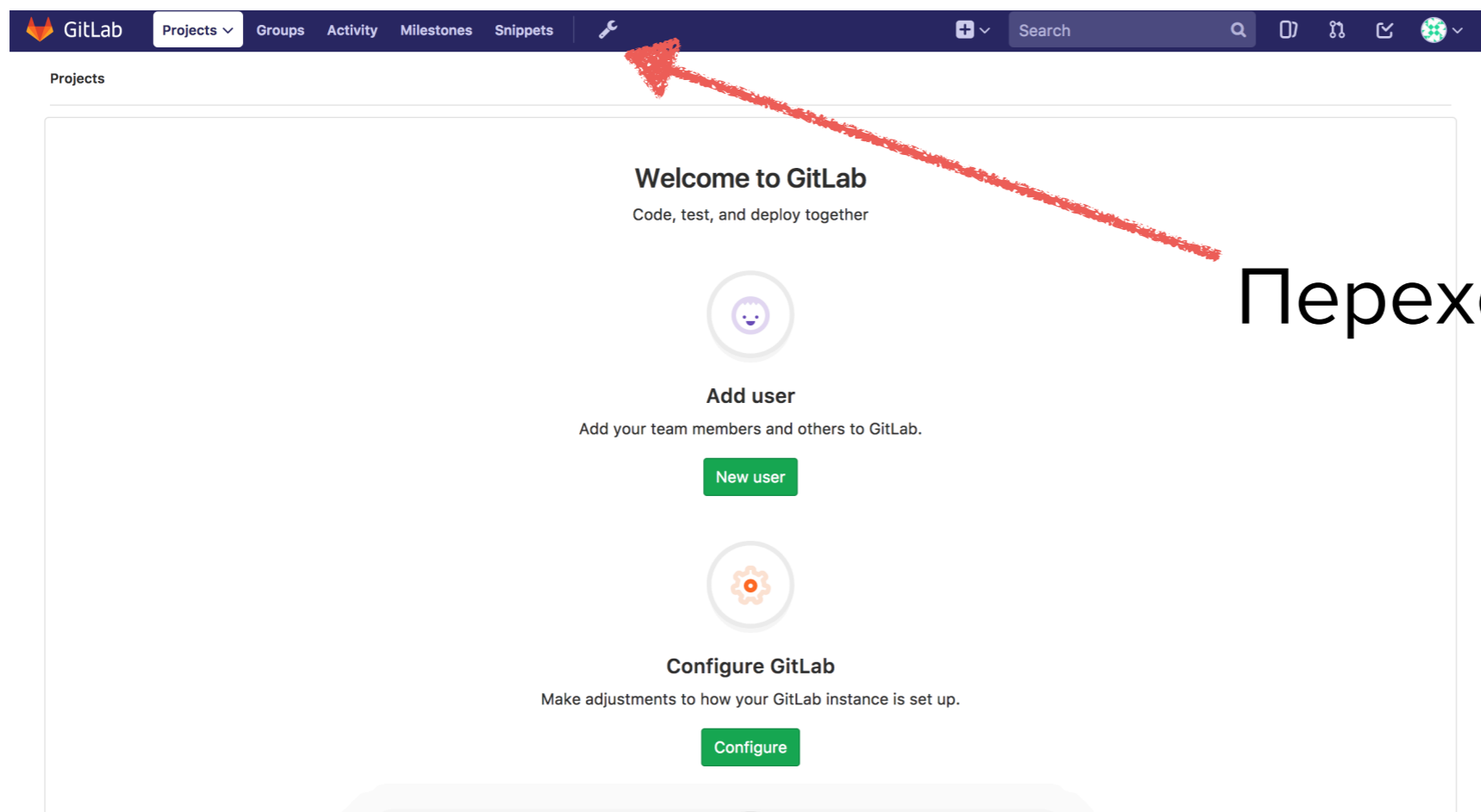
GitLab Community Edition

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	
<input type="text" value="root"/>	
Password	
<input type="password" value="....."/>	
<input checked="" type="checkbox"/> Remember me	Forgot your password?
<input type="button" value="Sign in"/>	

Настройки



Переходим в админку

Настройки

Выключаем регистрацию новых пользователей

1. Settings
2. Снимаем галку
3. Сохраняем

The screenshot shows the GitLab Admin Area interface. The left sidebar contains the 'Admin Area' menu with 'Settings' highlighted. The main content area is titled 'applications' and shows 'Sign-up Restrictions'. The 'Sign-up enabled' checkbox is checked, and a red arrow points to it. Below it, there is a text input field for 'Whitelisted domains for sign-ups' containing 'domain.com'. At the bottom, there is a green 'Save' button with a red arrow pointing to it.

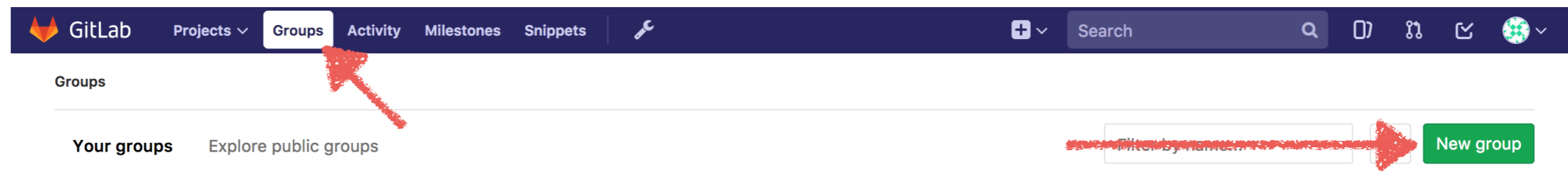
Опишем первый проект

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

New Group

Group path

Group name

Description

Group avatar
The maximum file size allowed is 200KB.

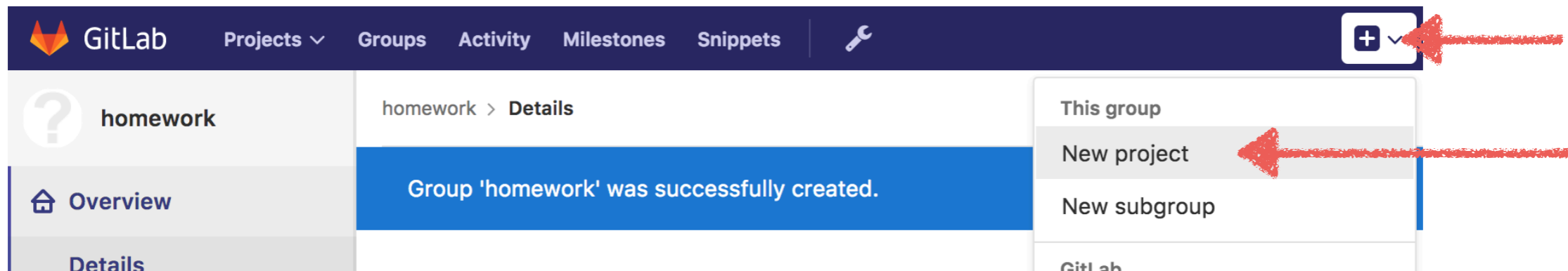
Visibility Level Private
The group and its projects can only be viewed by members.

Internal
The group and any internal projects can be viewed by any logged in user.

Public
The group and any public projects can be viewed without any authentication.

- A group is a collection of several projects
- Members of a group may only view projects they have permission to access
- Group project URLs are prefixed with the group namespace
- Existing projects may be moved into a group

Создадим проект

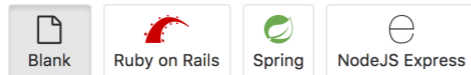


The screenshot shows the GitLab navigation bar with a dark blue background. On the right side, there is a '+' icon in a white square, which is highlighted by a red arrow. A dropdown menu is open, showing options: 'This group', 'New project', and 'New subgroup'. A second red arrow points to the 'New project' option. The main content area shows a notification: 'Group 'homework' was successfully created.'

New project

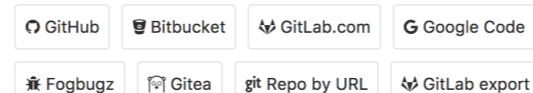
Create or Import your project from popular Git services

Create from template



OR

Import project from



Project path

http://35.198.76.199/ homework

Project name

example

Want to house several dependent projects under the same namespace? [Create a group](#)

Project description (optional)

Description format

Visibility Level

- Private
Project access must be granted explicitly to each user.
- Internal
This project cannot be internal because the visibility of **homework** is private. To make this project internal, you must first [change the visibility](#) of the parent group.
- Public
This project cannot be public because the visibility of **homework** is private. To make this project public, you must first [change the visibility](#) of the parent group.

Create project

Cancel

Избавляем бизнес от ИТ-зависимости



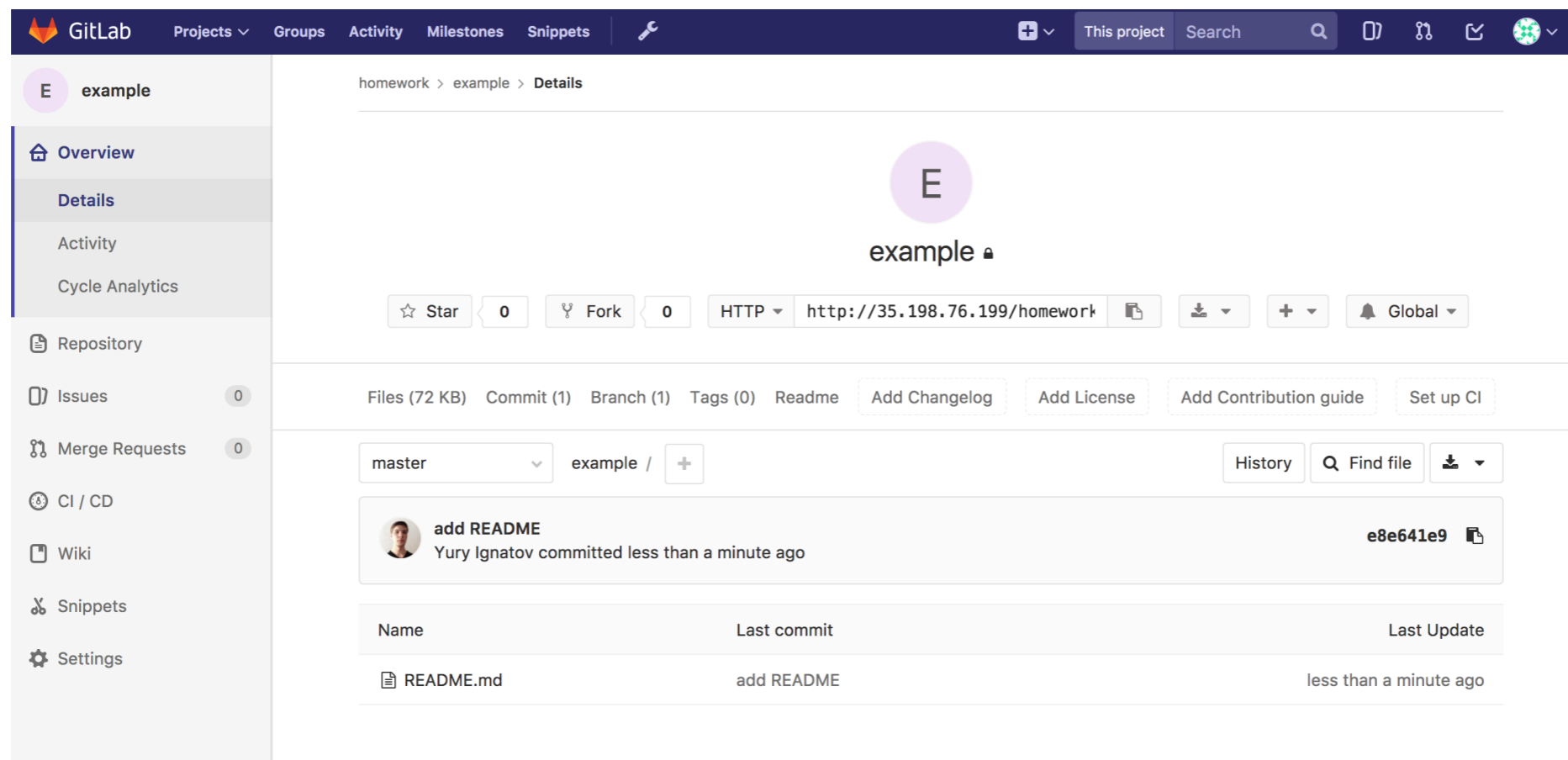
Добавляем remote в `<username>_microservices`

- > `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**



The screenshot shows the GitLab web interface for a project named 'example'. The left sidebar contains navigation options: Overview, Details (selected), Activity, Cycle Analytics, Repository, Issues (0), Merge Requests (0), CI / CD, Wiki, Snippets, and Settings. The main content area displays the project details, including the repository name 'example', star and fork counts (both 0), and the repository URL 'http://35.198.76.199/homework'. Below this, there are buttons for 'Add Changelog', 'Add License', 'Add Contribution guide', and 'Set up CI'. The current branch is 'master', and the commit history shows a recent commit 'add README' by Yury Ignatov, committed less than a minute ago, with the commit hash 'e8e641e9'. A table below the commit history lists the files in the repository:

Name	Last commit	Last Update
README.md	add README	less than a minute ago

CI/CD Pipeline

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

```
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
```

homework > example > Repository

| master                                                                                                                                                            | example / +             | History            | Find file | Download |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------|--------------------|-----------|----------|
|  <b>add pipeline definition</b><br>Yury Ignatov committed about a minute ago |                         | 1e20ea4e           |           |          |
| Name                                                                                                                                                              | Last commit             | Last Update        |           |          |
|  .gitlab-ci.yml                                                              | add pipeline definition | about a minute ago |           |          |
|  README.md                                                                   | add README              | 15 minutes ago     |           |          |

# CI/CD Pipeline

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

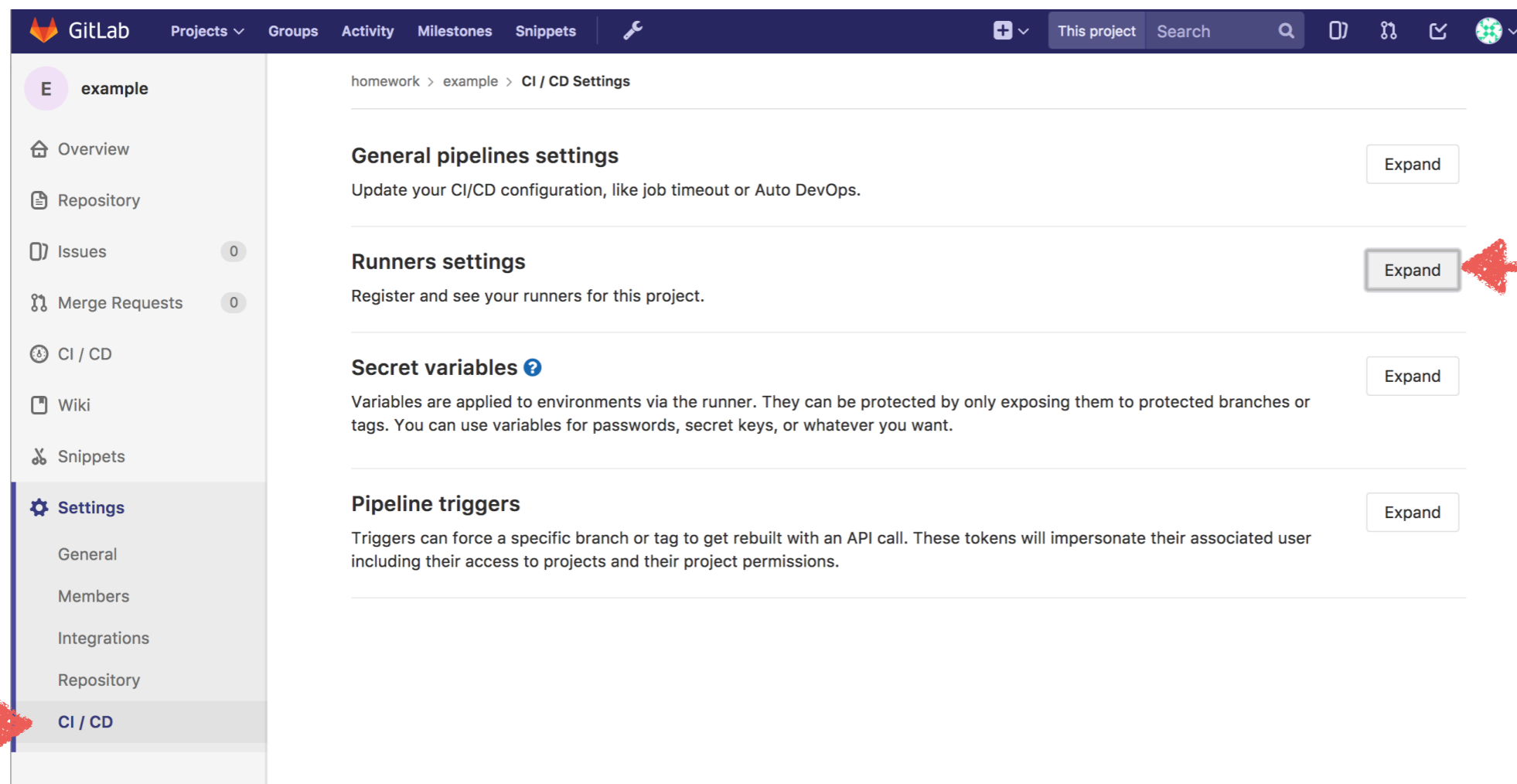
Но находится в статусе pending / stuck так как у нас нет runner

The screenshot shows the GitLab interface for a project named 'example'. The left sidebar contains navigation items: Overview, Repository, Issues (0), Merge Requests (0), CI / CD, and Pipelines. A red arrow points to the 'Pipelines' item. The main content area displays the 'Pipelines' page for the 'example' project. At the top, there are filters for 'All' (1), 'Pending' (1), 'Running' (0), and 'Finished' (0). There are also buttons for 'Run Pipeline' and 'CI Lint'. Below the filters is a table with columns: Status, Pipeline, Commit, and Stages. The table contains one entry: a pipeline with status 'pending', ID '#1 by latest', commit 'master 1e20ea4e', and a 'stuck' status. The pipeline is currently pending.

Запустим Runner и зарегистрируем его в интерактивном режиме

# Runner

Перед тем, как запускать и регистрировать runner  
нужно получить токен



The screenshot shows the GitLab interface for a project named 'example'. The left sidebar contains navigation options: Overview, Repository, Issues (0), Merge Requests (0), CI / CD, Wiki, Snippets, and Settings. The 'Settings' section is expanded, showing sub-options: General, Members, Integrations, Repository, and CI / CD. A red arrow points to the 'CI / CD' option in the sidebar. The main content area displays the 'CI / CD Settings' page with the following sections:


- General pipelines settings** (Expand button)
- Runners settings** (Expand button) - A red arrow points to this button.
- Secret variables** (Expand button)
- Pipeline triggers** (Expand button)

# 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!

## Runners activated for this project

● 621614a8 

my-runner

docker linux ubuntu xerial

Remove Runner

#1



## 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.

# Проверьте статус runner

Исправить это можно включив runner для проекта

E example2

- Overview
- Repository
- Issues 0
- Merge Requests 0
- CI / CD
- Wiki
- Snippets
- Settings**
  - General
  - Members
  - Integrations
  - Repository
  - CI / CD**

- active** - Runner is active and can process any new jobs
- paused** - Runner is paused and will not receive any new jobs

To start serving your jobs you can either add specific Runners

## 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://35.198.79.146/`
3. Use the following registration token during setup:  
`JV9VxxemJNSazUyG8C_P`
4. Start the Runner!

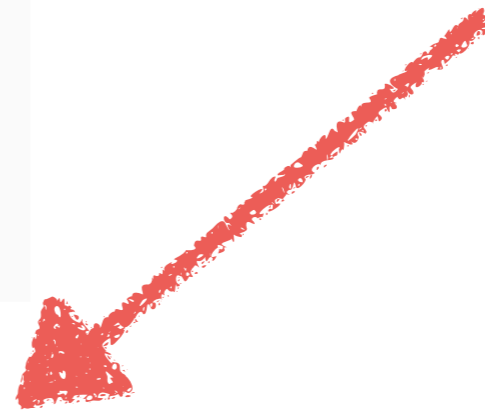
### Available specific runners

**d7a89caf**

Enable for this project

my-runner #1

docker linux ubuntu xenial



# CI/CD Pipeline

После добавления Runner пайплайн должен был запуститься

The screenshot displays the GitHub Actions interface for a repository named 'example'. The left sidebar shows navigation options: Overview, Repository, Issues (0), Merge Requests (0), CI / CD, and Pipelines. The main content area shows the 'Pipelines' view for the 'example' repository. A table lists pipeline runs, with the most recent one (#1 by 'latest') in a 'passed' status. This run is associated with the commit 'master -> 1e20ea4e' by user 'add pipeline definition' and consists of three stages, each marked with a green checkmark. The run completed 3 minutes ago with a duration of 00:00:50. Below the table, the 'Pipeline Jobs' view is shown, detailing the stages: 'Build' (containing 'build\_job'), 'Test' (containing 'test\_integratio...' and 'test\_unit\_job'), and 'Deploy' (containing 'deploy\_job'). All jobs are marked as successful with green checkmarks. Red arrows point from the 'passed' status in the table to the 'Build' stage in the pipeline diagram, and from the 'CI / CD' menu item to the 'Pipelines' section.

# 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

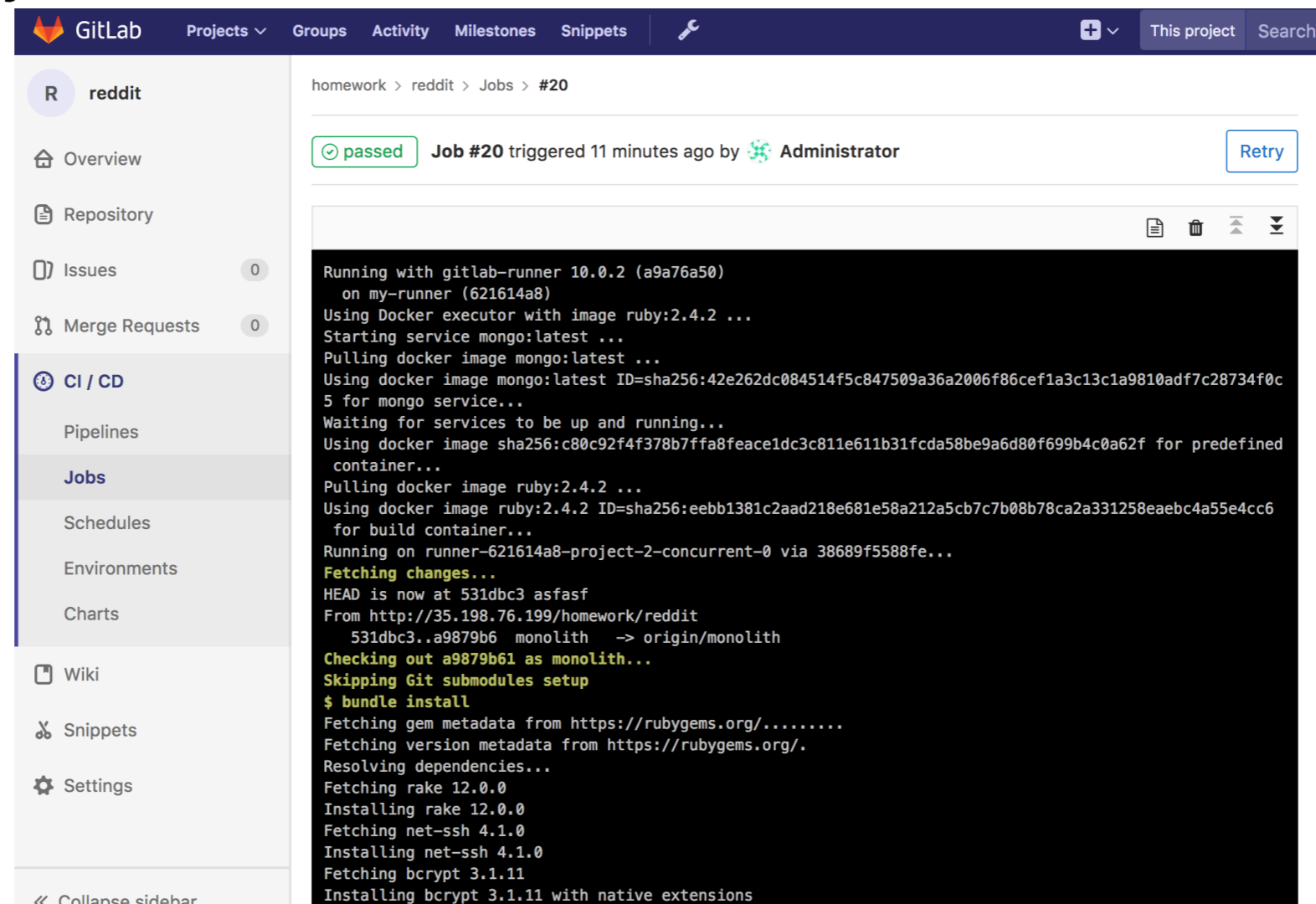
Последним шагом нам нужно добавить библиотеку для тестирования в **reddit/Gemfile** приложения.

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 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



```
GitLab Projects Groups Activity Milestones Snippets This project Search
R reddit
Overview
Repository
Issues 0
Merge Requests 0
CI / CD
 Pipelines
 Jobs
 Schedules
 Environments
 Charts
Wiki
Snippets
Settings
Collapse sidebar

homework > reddit > Jobs > #20
passed Job #20 triggered 11 minutes ago by Administrator Retry

Running with gitlab-runner 10.0.2 (a9a76a50)
on my-runner (621614a8)
Using Docker executor with image ruby:2.4.2 ...
Starting service mongo:latest ...
Pulling docker image mongo:latest ...
Using docker image mongo:latest ID=sha256:42e262dc084514f5c847509a36a2006f86cef1a3c13c1a9810adf7c28734f0c5 for mongo service...
Waiting for services to be up and running...
Using docker image sha256:c80c92f4f378b7ffa8feace1dc3c811e611b31fcda58be9a6d80f699b4c0a62f for predefined container...
Pulling docker image ruby:2.4.2 ...
Using docker image ruby:2.4.2 ID=sha256:eebb1381c2aad218e681e58a212a5cb7c7b08b78ca2a331258eaebc4a55e4cc6 for build container...
Running on runner-621614a8-project-2-concurrent-0 via 38689f5588fe...
Fetching changes...
HEAD is now at 531dbc3 asfasf
From http://35.198.76.199/homework/reddit
531dbc3..a9879b6 monolith -> origin/monolith
Checking out a9879b61 as monolith...
Skipping Git submodules setup
$ bundle install
Fetching gem metadata from https://rubygems.org/.....
Fetching version metadata from https://rubygems.org/.
Resolving dependencies...
Fetching rake 12.0.0
Installing rake 12.0.0
Fetching net-ssh 4.1.0
Installing net-ssh 4.1.0
Fetching bcrypt 3.1.11
Installing bcrypt 3.1.11 with native extensions
```

# Опишем окружения

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

В прошлом занятии мы создали job с названием `deploy_job`, но не разобрались 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`)

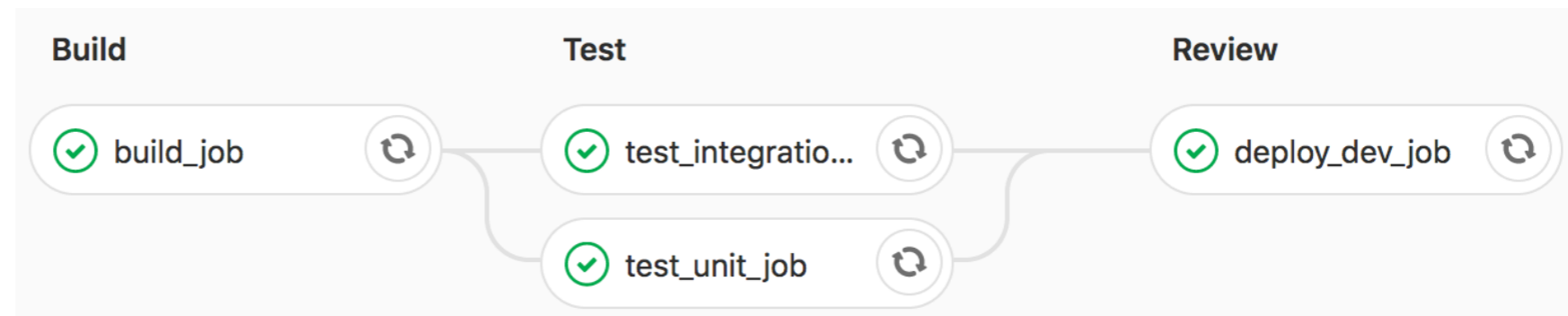
# Dev-окружение

```
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 завершения пайплайна с определением окружения перейти в CI/CD > Environments, то там появится определение первого окружения.

homework > cdexample > Pipelines > Environments

Available 1 Stopped 0

New environment

| Environment | Deployment                                                                                | Job                 | Commit                          | Updated       |                                                                                                 |
|-------------|-------------------------------------------------------------------------------------------|---------------------|---------------------------------|---------------|-------------------------------------------------------------------------------------------------|
| dev         | #1 by  | autodeploy dev #272 | <a href="#">ad4a1688</a><br>dev | 3 minutes ago |  Re-deploy |

# 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

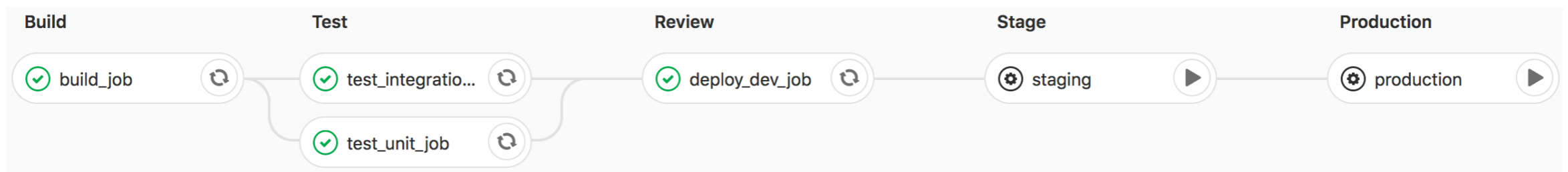
staging:
 stage: stage
 when: manual
 script:
 - echo 'Deploy'
 environment:
 name: stage
 url: https://beta.example.com

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 окружения staging и production

| Environment                | Deployment | Job                 | Commit                               | Updated       |           |
|----------------------------|------------|---------------------|--------------------------------------|---------------|-----------|
| <a href="#">dev</a>        | #2 by      | autodeploy dev #276 | <a href="#">d880f67d</a><br>asfasfas | 6 minutes ago | Re-deploy |
| <a href="#">production</a> |            |                     | No deployments yet                   |               |           |
| <a href="#">stage</a>      |            |                     | 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

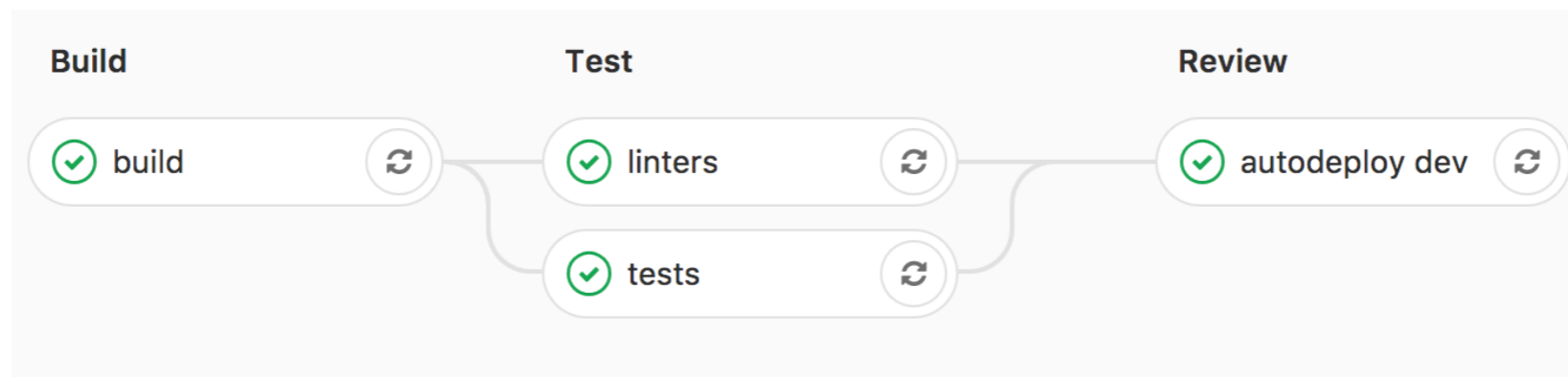
production:
 stage: production
 when: manual
 only:
 - /^\d+\.\d+\.\d+/
 script:
 - echo 'Deploy'
 environment:
 name: production
 url: https://example.com
```

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**

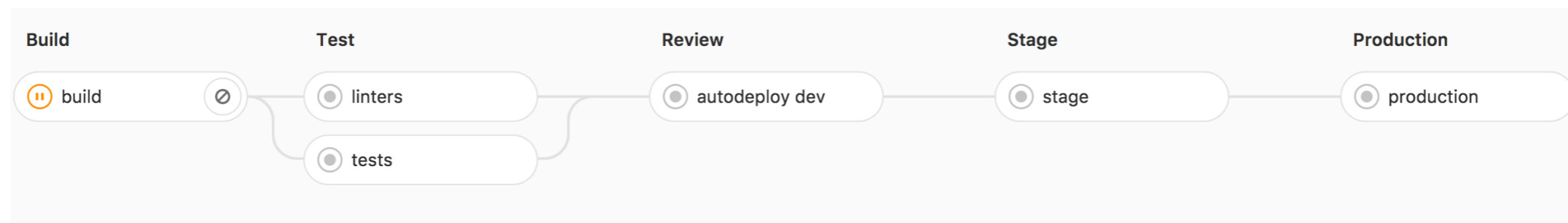
# Условия и ограничения

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                 |
|----------------------------------|---------------------------------------------------------------------------------------------|---------------------|
| <a href="#">dev</a>              | #3 by  | autodeploy dev #420 |
| ▼ 📁 review 2                     |                                                                                             |                     |
| <a href="#">review/bugfix</a>    |                                                                                             |                     |
| <a href="#">review/new-fe...</a> |                                                                                             |                     |

# 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 репозитория. 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 ревьюеры 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.