

# Training

## GitLab CI/CD



09.09.2020

<b>1. Theory</b>	<b>2</b>
1.1. User groups	2
1.1.1. Create a new group	2
1.1.2. Add users to a group	3
1.2. Projects	4
1.3. CI/CD	5
1.3.1. Getting started: Repository/Git	6
1.3.2. Getting started: Runners	7
<b>2. Practice</b>	<b>9</b>
2.1. Synchronize UZGRANT2020 project code in <a href="https://git.uzgrant2020.sungis.lv/">https://git.uzgrant2020.sungis.lv/</a> and <a href="http://gitlab.dshk.uz/">http://gitlab.dshk.uz/</a>	9
2.2. Set up runner	9
2.3. Add .gitlab-ci.yml configuration	10

# 1. Theory

## 1.1. User groups

With GitLab Groups, you can:

- Assemble related projects together.
- Grant members access to several projects at once.

Find your groups by clicking Groups > Your Groups in the top navigation.  
<http://gitlab.dshk.uz:3223/admin/groups>

The Groups page displays:

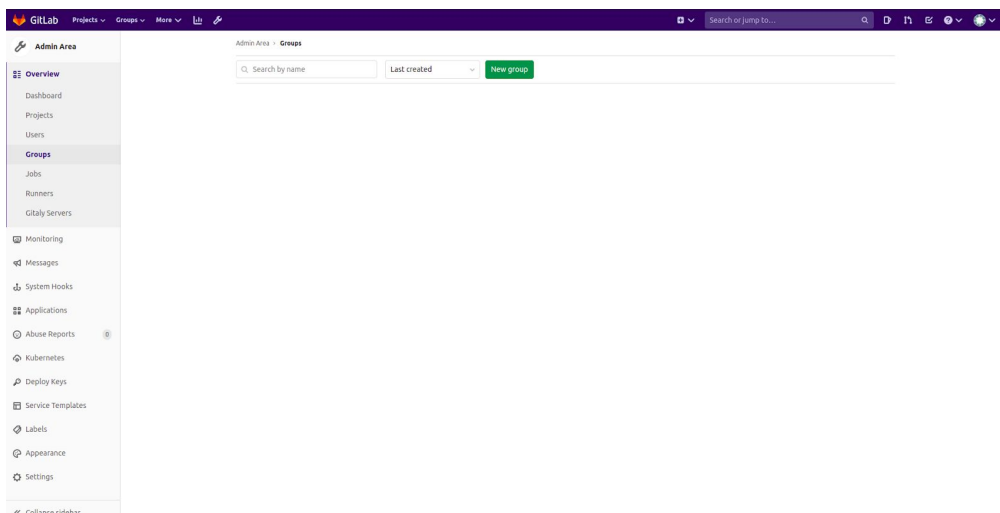
- All groups you are a member of, when Your groups is selected.
- A list of public groups, when Explore public groups is selected.

Each group on the Groups page is listed with:

- How many subgroups it has.
- How many projects it contains.
- How many members the group has, not including members inherited from parent group(s).
- The group's visibility.
- A link to the group's settings, if you have sufficient permissions.
- A link to leave the group, if you are a member.

### 1.1.1. Create a new group

To create a new Group, navigate to <http://gitlab.dshk.uz:3223/admin/groups>



Add the following information:

The screenshot shows the 'New group' form in the GitLab Admin Area. The form includes the following fields and options:

- Group name:** A text input field with the value 'My Awesome Group'.
- Group URL:** A text input field with the value 'http://gitlab.dnn.ac3223/'.
- Group description (optional):** A text area field.
- Group avatar:** A section with a 'Choose file...' button and a note: 'The maximum file size allowed is 200KB.'
- Visibility level:** A dropdown menu with 'Private' selected. Below it, three radio buttons are shown: 'Private' (selected), 'Internal', and 'Public'. Descriptions for each are provided.
- Allow users to request access (if visibility is public or internal):** A checkbox that is checked.
- Large File Storage:** A section with a checkbox 'Allow projects within this group to use Git LFS' which is checked.
- Allowed to create projects:** A dropdown menu with 'Developers + Maintainers' selected.
- Allowed to create subgroups:** A dropdown menu with 'Maintainers' selected.
- Two-factor authentication:** A checkbox 'Require all users in this group to set up Two-factor authentication' which is checked. Below it is a text input field with the value '48' and a note: 'Amount of time (in hours) that users are allowed to skip forced configuration of two-factor authentication'.

A blue information box at the bottom contains the following text:

- 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

The 'Create group' button is at the bottom right of the form.

1. The Group name will automatically populate the URL. Optionally, you can change it. This is the name that displays in group views. The name can contain only:
  - Alphanumeric characters
  - Underscores
  - Dashes and dots
  - Spaces
2. The Group URL is the namespace under which your projects will be hosted. The URL can contain only:
  - Alphanumeric characters
  - Underscores
  - Dashes and dots (it cannot start with dashes or end in a dot)
3. Optionally, you can add a brief description to tell others what this group is about.
4. Optionally, choose an avatar for your group.
5. Choose the [visibility level](#).

For more details on creating groups, watch the video [GitLab Namespaces \(users, groups and subgroups\)](#).

### 1.1.2. Add users to a group

A benefit of putting multiple projects in one group is that you can give a user access to all projects in the group with one action.

Add members to a group by navigating to the group's dashboard


Add user(s) to the group:

Read more about project permissions [here](#)

Add users to group

uzgrant2020-developers group members 1

Manage access

 **Aigars Znotins** @aigars It's you Owner

Given access just now

Select the [permission level](#), and add the new member. You can also set the expiring date for that user; this is the date on which they will no longer have access to your group.

## 1.2. Projects

In GitLab, you can create projects for hosting your codebase, use it as an issue tracker, collaborate on code, and continuously build, test, and deploy your app with built-in GitLab CI/CD.

To create a new project navigate to <http://gitlab.dshk.uz:3223/admin/projects> and press “New project”.

**New project**

A project is where you house your files (repository), plan your work (issues), and publish your documentation (wiki), among other things.

All features are enabled for blank projects, from templates, or when importing, but you can disable them afterward in the project settings.

Information about additional Pages templates and how to install them can be found in our [Pages getting started guide](#).

**Tip:** You can also create a project from the command line. [Show command](#)

**Blank project** | Create from template | Import project

**Project name**  
uzgrant2020

**Project URL**  
http://gitlab.dshk.uz:3223/

**Project slug**  
uzgrant2020-developers

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. If this project is part of a group, access will be granted to members of the group.

☐ Internal

☐ Public

☐ **Initialize repository with a README**  
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

[Create project](#) [Cancel](#)

- Blank projects create empty repository with no files except readme.
- Project templates can pre-populate a new project with the necessary files to get you started quickly.
- Import project allows importing existing project.

Project requires such a information:

- Name - short name of project;
- Url - specify how url will look. It's combinations of group/user and project slug;
- Slug - usually stays slugged project name, but can be changed;
- Notes - project descriptions or other notes;
- Visibility level - tells if a project is private, or can be accessible by other GitLab users.

### 1.3. CI/CD

GitLab CI/CD is a tool built into GitLab for software development through the continuous methodologies:

- Continuous Integration (CI)
- Continuous Delivery (CD)
- Continuous Deployment (CD)

Continuous Integration works by pushing small code chunks to your application's code base hosted in a Git repository, and, to every push, run a pipeline of scripts to build, test, and validate the code changes before merging them into the main branch.

Continuous Delivery and Deployment consist of a step further CI, deploying your application to production at every push to the default branch of the repository.

These methodologies allow you to catch bugs and errors early in the development cycle, ensuring that all the code deployed to production complies with the code standards you established for your app.

For a complete overview of these methodologies and GitLab CI/CD, read the [Introduction to CI/CD with GitLab](#).

### 1.3.1. Getting started: Repository/Git

To use GitLab CI/CD, all you need is an application codebase hosted in a Git repository, and for your build, test, and deployment scripts to be specified in a file called `.gitlab-ci.yml`, located in the root path of your repository.

In this file, you can define the scripts you want to run, define include and cache dependencies, choose commands you want to run in sequence and those you want to run in parallel, define where you want to deploy your app, and specify whether you will want to run the scripts automatically or trigger any of them manually. Once you're familiar with GitLab CI/CD you can add more advanced steps into the configuration file.

To add scripts to that file, you'll need to organize them in a sequence that suits your application and are in accordance with the tests you wish to perform. To visualize the process, imagine that all the scripts you add to the configuration file are the same as the commands you run on a terminal on your computer.

Once you've added your `.gitlab-ci.yml` configuration file to your repository, GitLab will detect it and run your scripts with the tool called `GitLab Runner`, which works similarly to your terminal.

The scripts are grouped into jobs, and together they compose a pipeline. A minimalist example of `.gitlab-ci.yml` file could contain:

```
before_script:
  - apt-get install npm -y

stages:
  - test_environments

uzgrant2020-test:
  stage: test_environments
  type: deploy
  tags:
    - uzgrant2020-test
  only:
    - alpha
  script:
    - echo Deploy fronetnd
    - cd frontend
    - npm install
    #- npm run lint
    - ng build
    - rm -r /var/www/html/frontend/*
    - mv dist/frontend/* /var/www/html/frontend/
    - rm -r dist
```

```

- echo Deploy backend
- cd ..
- cp backend/deploy.sh /home/gitlab-runner/deploy.sh
- chmod +x /home/gitlab-runner/deploy.sh
- /home/gitlab-runner/deploy.sh

```

The `before_script` attribute would install the dependencies for your app before running anything, and a job called `uzgrant2020-test` would build `uzgrant2020` project frontend with help of `npm`. It will run only for the `alpha` branch (it's specified in `only` tag) and execute all in `script` tag specified commands.

GitLab CI/CD not only executes the jobs you've set but also shows you what's happening during execution, as you would see in your terminal (available under <https://gitlab.dshk.uz:3223/{project}/-jobs> by clicking on specific job):

```

1 Running with gitlab-ci-multi-runner 9.5.1 (96b34cc)
2 on UZGRANT2020 test server (bqoRmErf)
3 Using Shell executor...
4 Running on uzgrant2020-test...
5 Fetching changes...
6 Removing frontend/node_modules/
7 HEAD is now at a27a5a6 #8 Improvements in frontend deployment
8 From http://git.uzgrant2020.sungis.lv/uzgrant2020-developers/uzgrant2020
9 a27a5a6..95903fd alpha -> origin/alpha
10 Checking out 95903fd as alpha...
11 Skipping git submodule setup
12 $ echo Deploy frontend
13 Deploy frontend
14 $ cd frontend
15 $ npm install
16 > deasync@0.1.20 install /home/gitlab-runner/builds/bqoRmErf/0/uzgrant2020-developers/uzgrant2020/frontend/node_modules/deasync
17 > node ./build.js
18 'linux-x64-node-12' exists; testing
19 Binary is fine; exiting
20 > husky@3.0 install /home/gitlab-runner/builds/bqoRmErf/0/uzgrant2020-developers/uzgrant2020/frontend/node_modules/husky
21 > node husky install
22 husky > Setting up git hooks
23 CI detected, skipping Git hooks installation.
24 husky > Done
25 > core-js@2.6.11 postinstall /home/gitlab-runner/builds/bqoRmErf/0/uzgrant2020-developers/uzgrant2020/frontend/node_modules/babel-runtime/node_modules/core-js
26 > node -e "try(require('./postinstall'))catch(e){}"
27 > core-js@3.6.4 postinstall /home/gitlab-runner/builds/bqoRmErf/0/uzgrant2020-developers/uzgrant2020/frontend/node_modules/core-js
28 > node -e "try(require('./postinstall'))catch(e){}"
29 > core-js@2.6.11 postinstall /home/gitlab-runner/builds/bqoRmErf/0/uzgrant2020-developers/uzgrant2020/frontend/node_modules/parcel-bundler/node_modules/core-js
30 > node -e "try(require('./postinstall'))catch(e){}"
31 > @angular/cli@8.1.1 postinstall /home/gitlab-runner/builds/bqoRmErf/0/uzgrant2020-developers/uzgrant2020/frontend/node_modules/@angular/cli
32 > node ./bin/postinstall/script.js
33 > husky@3.0 postinstall /home/gitlab-runner/builds/bqoRmErf/0/uzgrant2020-developers/uzgrant2020/frontend/node_modules/husky
34 > opencollective-postinstall || exit 0
35 > parcel-bundler@1.12.4 postinstall /home/gitlab-runner/builds/bqoRmErf/0/uzgrant2020-developers/uzgrant2020/frontend/node_modules/parcel-bundler
36 > node --no-warnings --env-cwd /home/gitlab-runner/builds/bqoRmErf/0/uzgrant2020-developers/uzgrant2020/frontend/node_modules/parcel-bundler

```

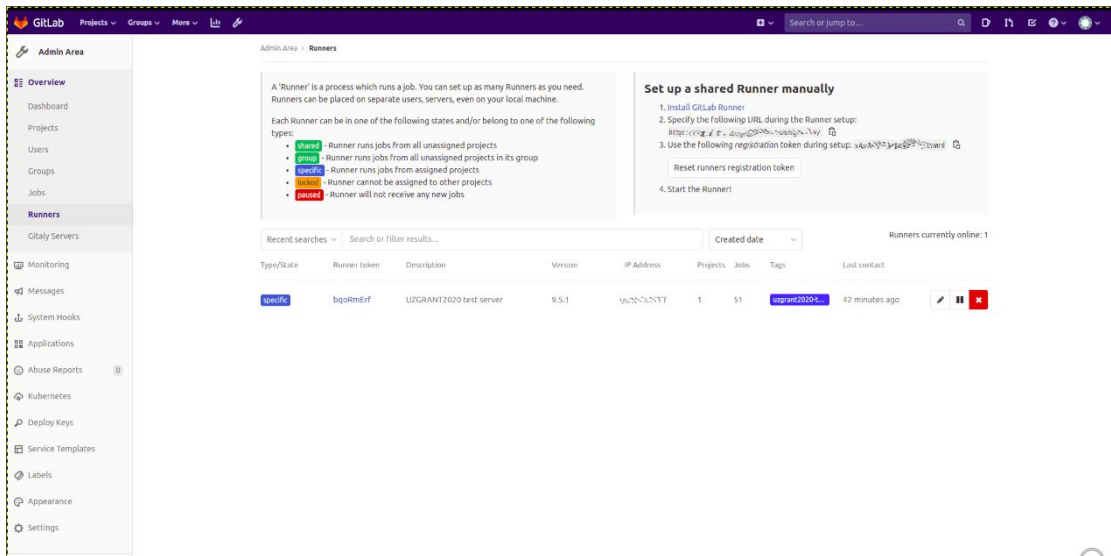
### 1.3.2. Getting started: Runners

Runners run the code defined in `.gitlab-ci.yml`. A runner is a lightweight, highly-scalable agent that picks up a CI job through the coordinator API of GitLab CI/CD, runs the job, and sends the result back to the GitLab instance.

Runners are created by an administrator and are visible in the GitLab UI. Runners can be specific to certain projects or available to all projects.

Runner can be managed under <http://gitlab.dshk.uz:3223/admin/runners>





Runner run code on remote machine with help of [gitlab-ci-multi-runner](#) package. Package allows registering plenty of runners which can be run from different git repositories and projects.

More detailed about GitLab CI/CD in <https://docs.gitlab.com/ee/ci/README.html>

## 2. Practice

### 2.1. Synchronize UZGRANT2020 project code in <https://git.uzgrant2020.sungis.lv/> and <http://gitlab.dshk.uz/>

1. Create a developer group and project (see 1.1 and 1.2 sections). Don't forget to add sungis\_synhronizer user to the group.
2. Create alpha and beta branches. It can be done by cloning repository and with commands:

```
az@dell:~/Desktop/Projekti/UZGRANT2020/UZ$ git clone
http://gitlab.dshk.uz:3223/uzgrant2020-developers/uzgrant2020.git
Cloning into 'uzgrant2020'...
Username for 'http://gitlab.dshk.uz:3223': aigars.znotins@sungis.lv
Password for 'http://aigars.znotins@sungis.lv@gitlab.dshk.uz:3223':
warning: You appear to have cloned an empty repository.
az@dell:~/Desktop/Projekti/UZGRANT2020/UZ$ cd uzgrant2020/
az@dell:~/Desktop/Projekti/UZGRANT2020/UZ/uzgrant2020$ git branch alpha
az@dell:~/Desktop/Projekti/UZGRANT2020/UZ/uzgrant2020$ git branch beta
az@dell:~/Desktop/Projekti/UZGRANT2020/UZ/uzgrant2020$ git push origin
alpha
az@dell:~/Desktop/Projekti/UZGRANT2020/UZ/uzgrant2020$ git push origin
beta
```

3. Wait till SunGIS team runs script for code synchronization

### 2.2. Set up runner

First on remote machine install gitlab-ci-multi-runner package

```
sudo apt-get install gitlab-ci-multi-runner
```

If package not available add repository and try to install one more time

```
curl -L
https://packages.gitlab.com/install/repositories/runner/gitlab-ci-multi-r
unner/script.deb.sh | sudo bash
cat > /etc/apt/preferences.d/pin-gitlab-runner.pref <<EOF
Explanation: Prefer GitLab provided packages over the Debian native ones
Package: gitlab-ci-multi-runner
Pin: origin packages.gitlab.com
Pin-Priority: 1001
EOF

sudo apt-get install gitlab-ci-multi-runner
```

After installing, register runner. It can be done by command

```
sudo gitlab-ci-multi-runner register
```

In the registration process there will be plenty of questions:

1. *coordinator URL* - It's your gitlab ci page <http://gitlab.dshk.uz:3223/ci>
2. *token* - copy it from <http://gitlab.dshk.uz:3223/admin/runners>

### Set up a shared Runner manually

1. Install GitLab Runner
2. Specify the following URL during the Runner setup:  
`http://gitlab.dshk.uz:3223/`
3. Use the following registration token during setup: `s57m8ychwuqSHzjPL4KN`
4. Start the Runner!

Reset runners registration token

3. *description* - shortly describe runner in free text
4. *tag* - name of tag, what will be/is specified in `.gitlab-ci.yml`
5. *untagged builds* - answer true
6. *lock runner to current project* - answer false
7. *executor* - answer shell

When it's done, on the gitlab server open <http://gitlab.dshk.uz:3223/admin/runners> and choose just a registered runner. Open it by clicking the Runner token link. It will open a runner administration form. Enable projects, for what run ci

### Restrict projects for this Runner

Assigned projects	
uzgrant2020-developers / uzgrant2020	Disable
Project	
<input type="text"/>	Search
project-steering / project-steering	Enable
uzgrant2020-developers / cad	Enable

Runner setup is finished. After the next push, the pipeline should run on a remote machine. If `.gitlab-ci.yml` script requires, a remote machine should be pre configured. See project readme for more notes.

## 2.3. Add `.gitlab-ci.yml` configuration

1. Add `.gitlab-ci.yml` file and copy following content:

```
stages:
  - test_environments

uzgrant2020-accept-test:
  stage: test_environments
  type: deploy
  tags:
    - uzgrant2020-accept-test
  only:
    - beta
  script:
    - echo Deploy fronetnd
    - cd frontend
    - npm install
    #- npm run lint
    - ng build
    - rm -r /var/www/html/frontend/*
    - mv dist/frontend/* /var/www/html/frontend/
    - rm -r dist
    - echo Deploy backend
    - cd ..
    - cp backend/deploy.sh /home/gitlab-runner/deploy.sh
    - chmod +x /home/gitlab-runner/deploy.sh
    - /home/gitlab-runner/deploy.sh
```

2. Push changes, and see if pipeline runs correctly