

# Летняя школа DevOps/DevSecOps

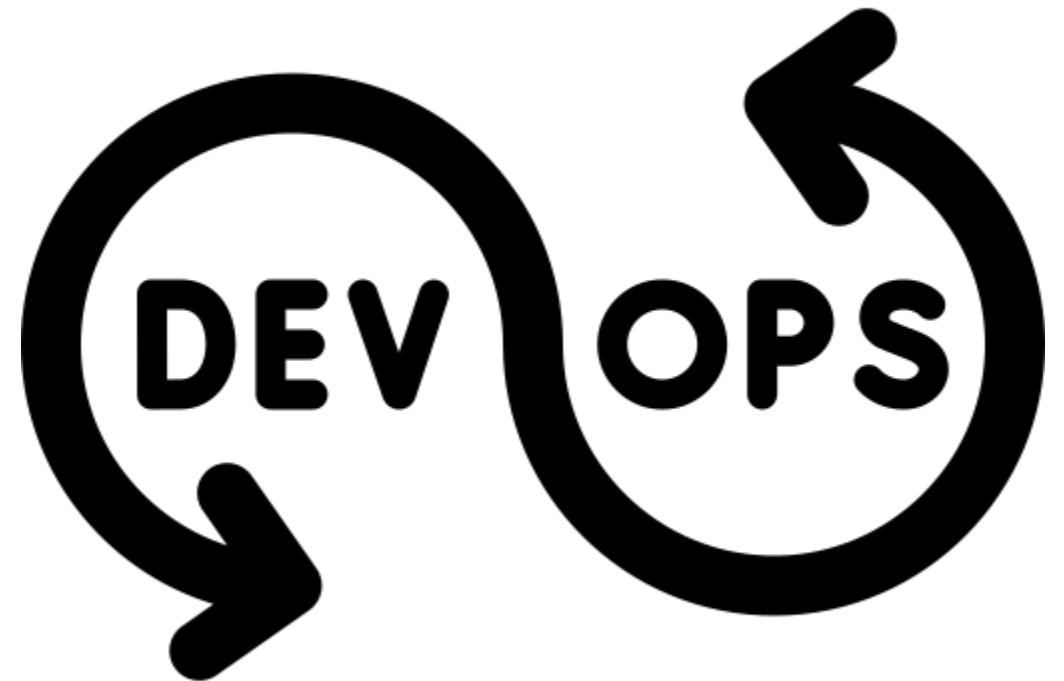
CI/CD concept

# DevOps

## Что это такое?

Это методология которая объединяет разработку приложений и операции для улучшения эффективности и надежности процесса разработки ПО.

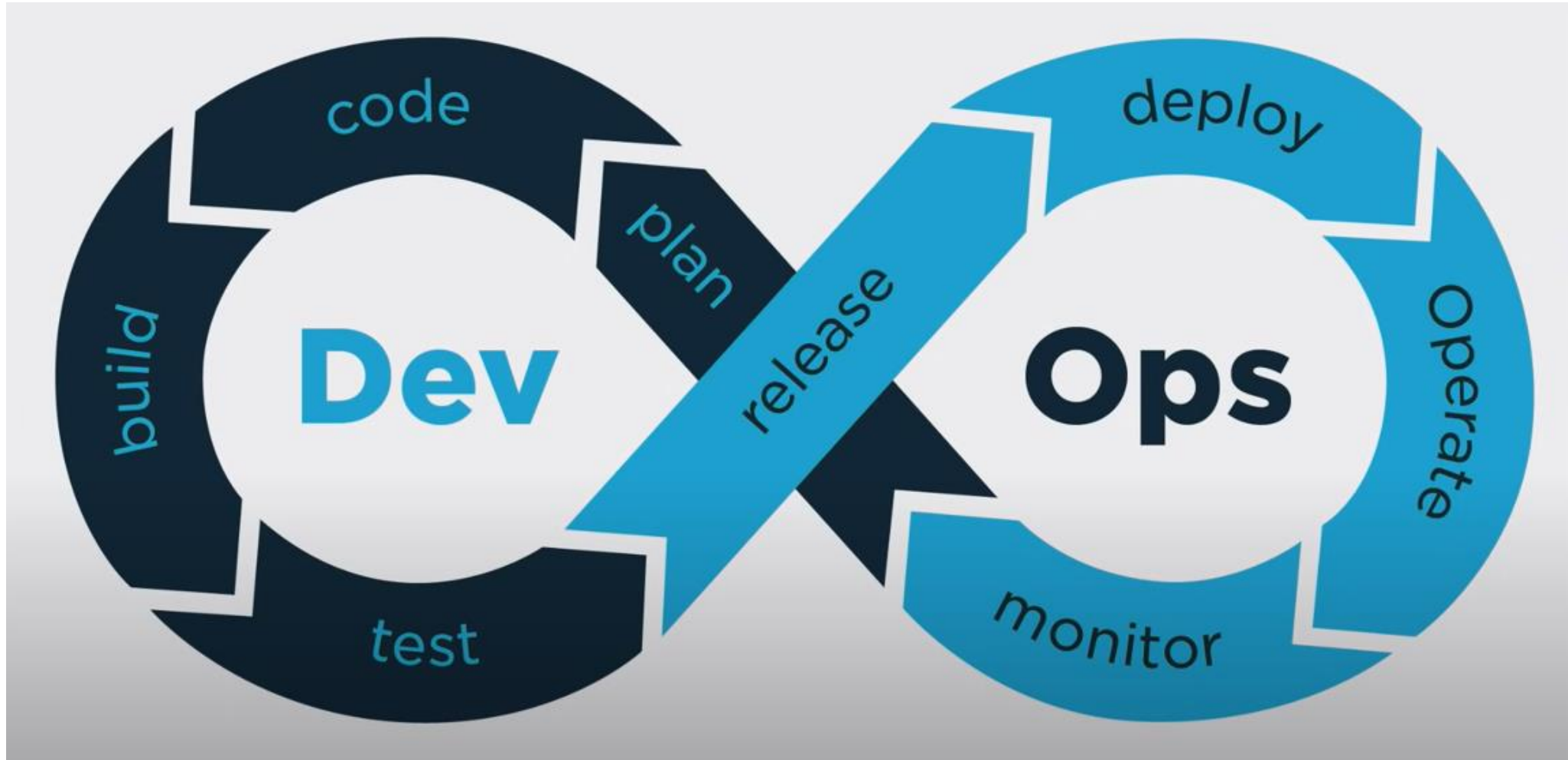
Она требует использования различных инструментов, таких как системы контроля версий, системы непрерывной интеграции и доставки (CI/CD), системы контейнеризации, утилиты IaC и другие.



# GitLab CI/CD

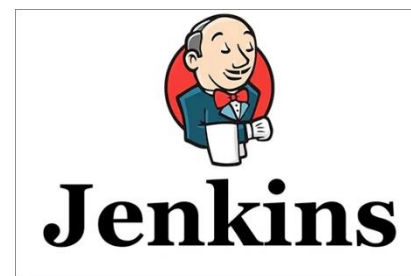
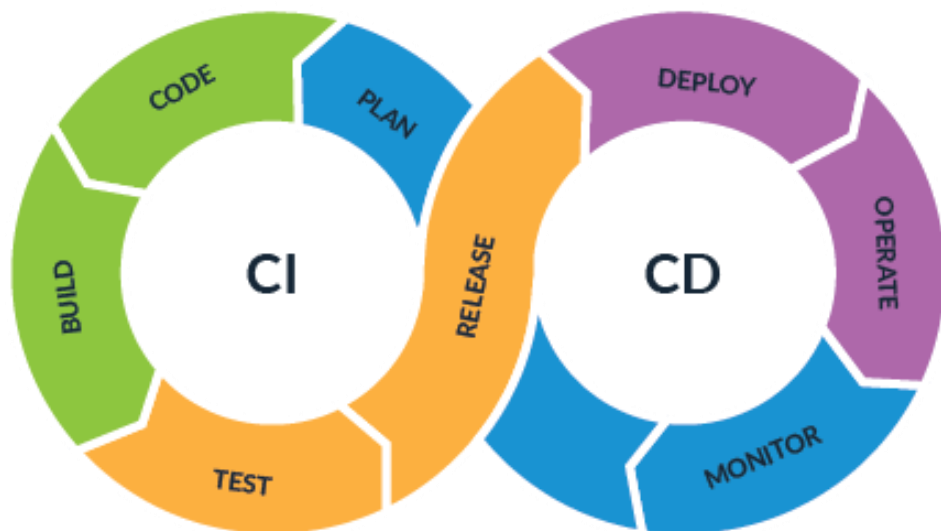
- Continuous Integration / Continuous Deployment
- GitLab CI/CD
- Настройка заданий
- Настройка pipelines
- Использование контейнеров
- Основные полезные ключевые слова
- Создание первого конвейера
  - От сборки до выгрузки
- Артефакты, окружение, кэш и т.п.

# Автоматизация рутинных операций



# GitLab CI/CD

Непрерывная интеграция (Continuous Integration, CI) и непрерывная поставка (Continuous Delivery, CD) представляют собой культуру, набор принципов и практик, которые позволяют разработчикам чаще и надежнее разворачивать изменения программного обеспечения.



# Что дает?

- С точки зрения разработчика
  - Он сразу видит результат и получает быстрый feedback в случае ошибок
  - Позволяет обнаруживать и исправлять ошибки в короткое время
    - Только это работает, если он код покрыт тестами
- С точки зрения работы с чужим кодом
  - Работать с покрытым тестами кодом, который снабжен скриптами для развертывания и упакован в стандартный контейнер, значительно приятнее

# Что дает?

- С точки зрения человека, отвечающего за deploy
  - (системный администратор, release manager, operations project manager)
  - Теперь это DevOps инженер
  - Все базовые операции автоматизированы, остается только мониторить процесс и автоматически накатывать или откатывать изменения

# Что дает?

- С точки зрения product owner
  - получает возможность быстрее оценить результат, внести изменения в задание, контролировать весь процесс
- С точки зрения менеджера проекта
  - После разработки остается не просто программный код, а готовый конвейер для тестирования, сборки и развертывания

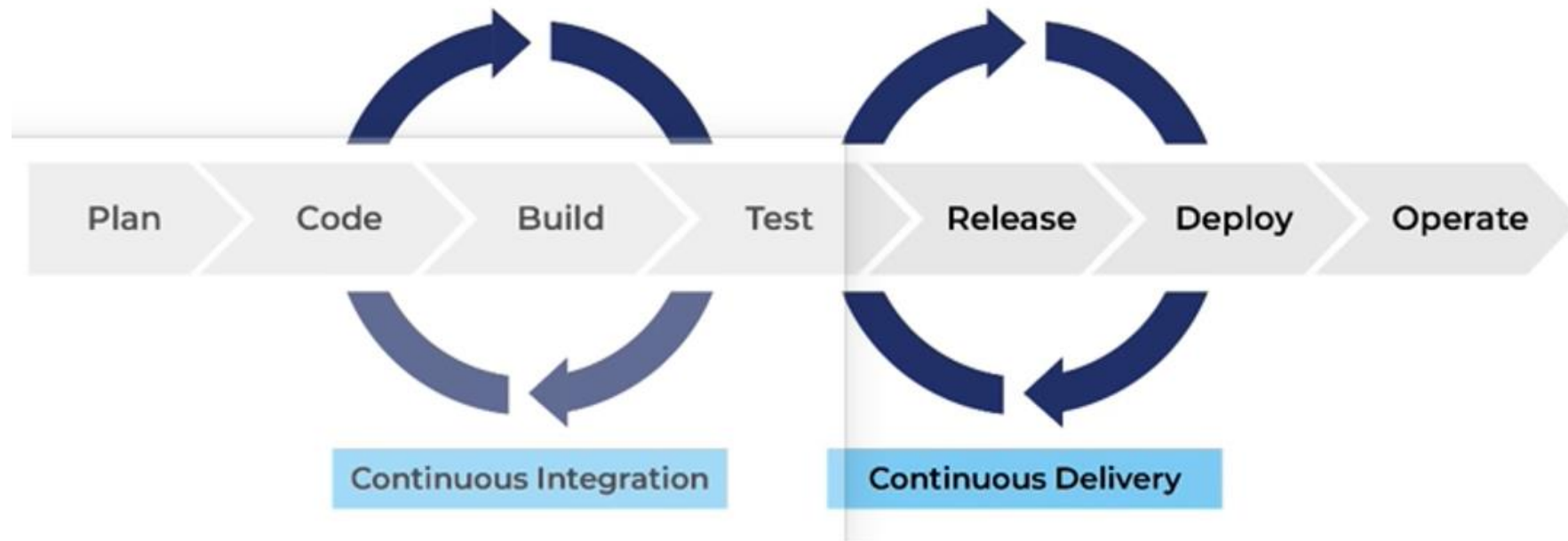


# Что дает?

- Легко и быстро деплоить новые версии приложений
- Запускать скрипты автоматизации
- Запускать тестовые стенды
- Запускать автотесты и линтеры
- Контролировать то, какой код попадает на прод
- Легко откатываться к предыдущей версии при возникновении проблем
- И многое другое...

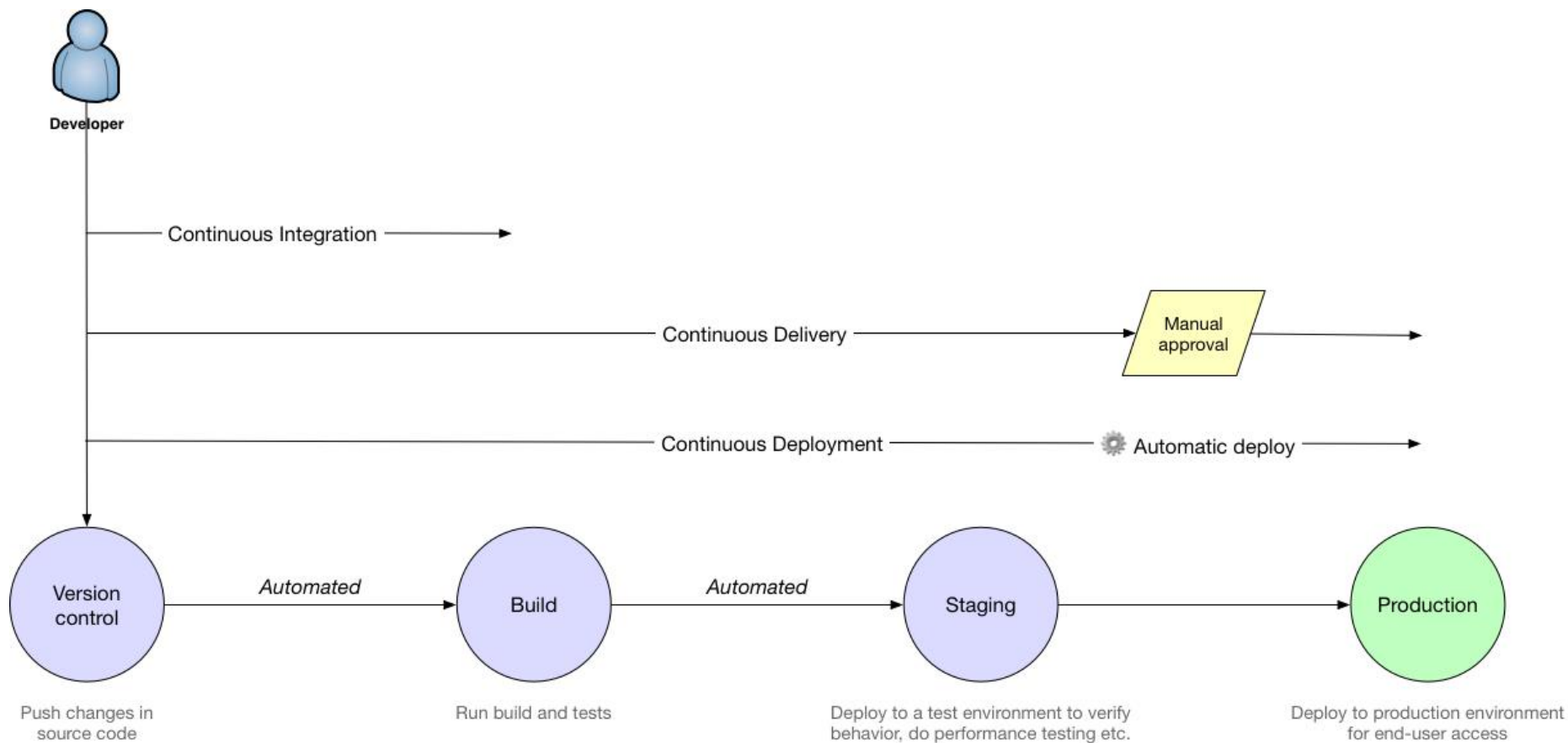
# CI/CD

- Continuous integration (CI) – непрерывная интеграция
- Continuous delivery (CD) – непрерывная доставка
- Continuous deployment (CD) – непрерывное развертывание



# CI/CD

- Continuous delivery / continuous deployment



# Continuous integration

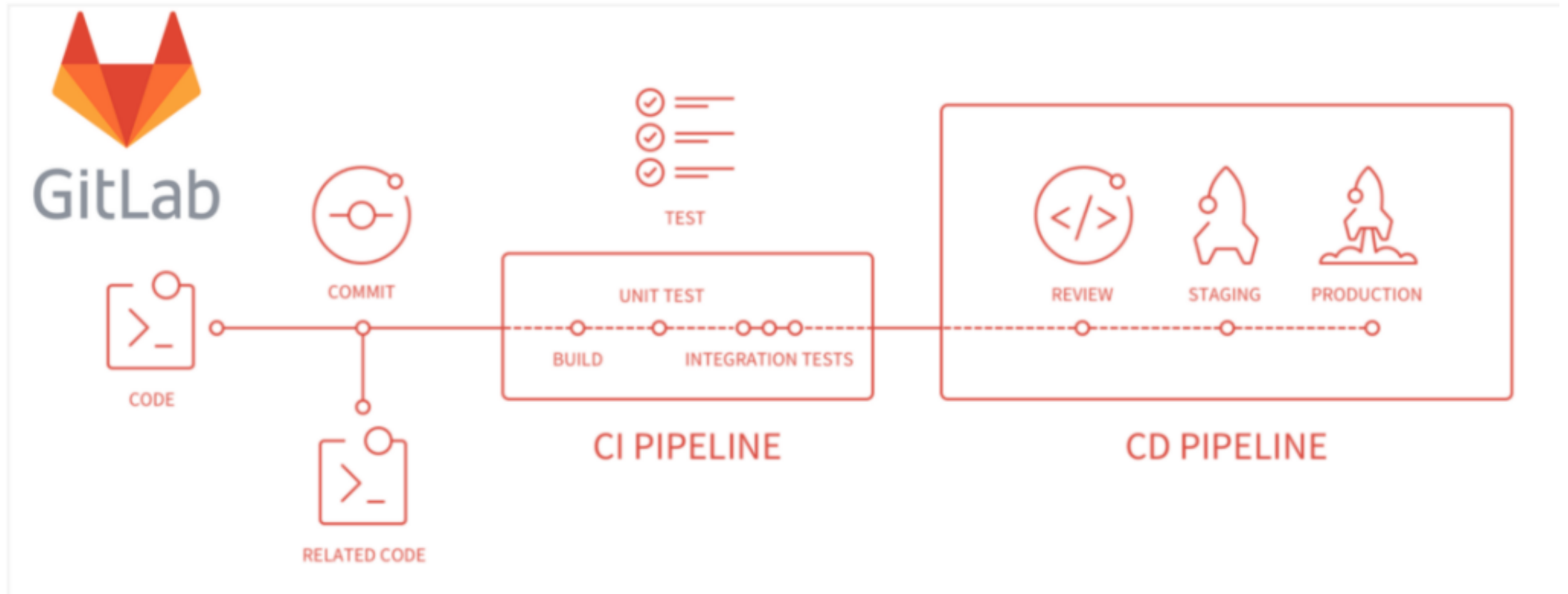
- Этапы разработки, сборки тестирования, упаковки
- Работы над кодом
  - Исправления кода, code review и т.п.
- Автотесты
  - Сюда входит этапы модульного и интеграционного тестирования
- Упаковка
  - Создание артефактов, Сборка контейнеров, заливка их в реестр
- Основная задача – это прогон автотестов и упаковка приложения (подготовка к доставке)
  - После данного этапа приложение теоретически готово к внедрению

# Continuous deployment

- Этап доставки на сервер уже готового протестированного продукта
  - К моменту deploy мы уверены, что система в консистентном состоянии и все необходимые тесты прошли
- Может включать в себя настройку инфраструктуры, доставку сборки на сервер и разворачивание на сервере
- Можно настраивать автоматически, можно вручную
- Основная задача – развернуть обновленную систему в заданном окружении

# CI/CD pipelines

- Deploy может идти в разные environments



# Staging server

- Кроме автотестов, бывает еще ручное тестирование
  - Приемочное тестирование, канареечное тестирование и т.п.
- Для этого надо где-то развернуть приложение
  - Обычно будет несколько environments: test, staging, production
- Continuous Delivery может включать в себя deploy на test, staging, production сервер
- Для всех окружений могут быть свои правила
  - например на staging делать deploy автоматом а на production вручную и т.п.

# Летняя школа DevOps/DevSecOps

GitLab runners & job execution



# GitLab architecture

- Рассмотрим, как устроен gitlab с точки зрения выполнения заданий
- Мы помним, что gitlab это комплексное приложение с множеством функций, сейчас интересует именно job execution
- Вне зависимости от типа установки (облачная или self-managed), у gitlab будут следующие элементы:
  - Сервер gitlab
  - Специальные хосты для выполнения заданий – раннеры (runners)

# GitLab server

- Главный компонент
- На нем хранится вся конфигурация
  - В том числе, конфигурация pipeline
- Он управляет запуском заданий – ставит их на выполнение
- На сервере хранятся результаты выполнения заданий
- Однако сам он задания не выполняет, а передает их специальному узлу – раннеру
  - Если точнее, раннер сам забирает готовые задачи по мере их появления

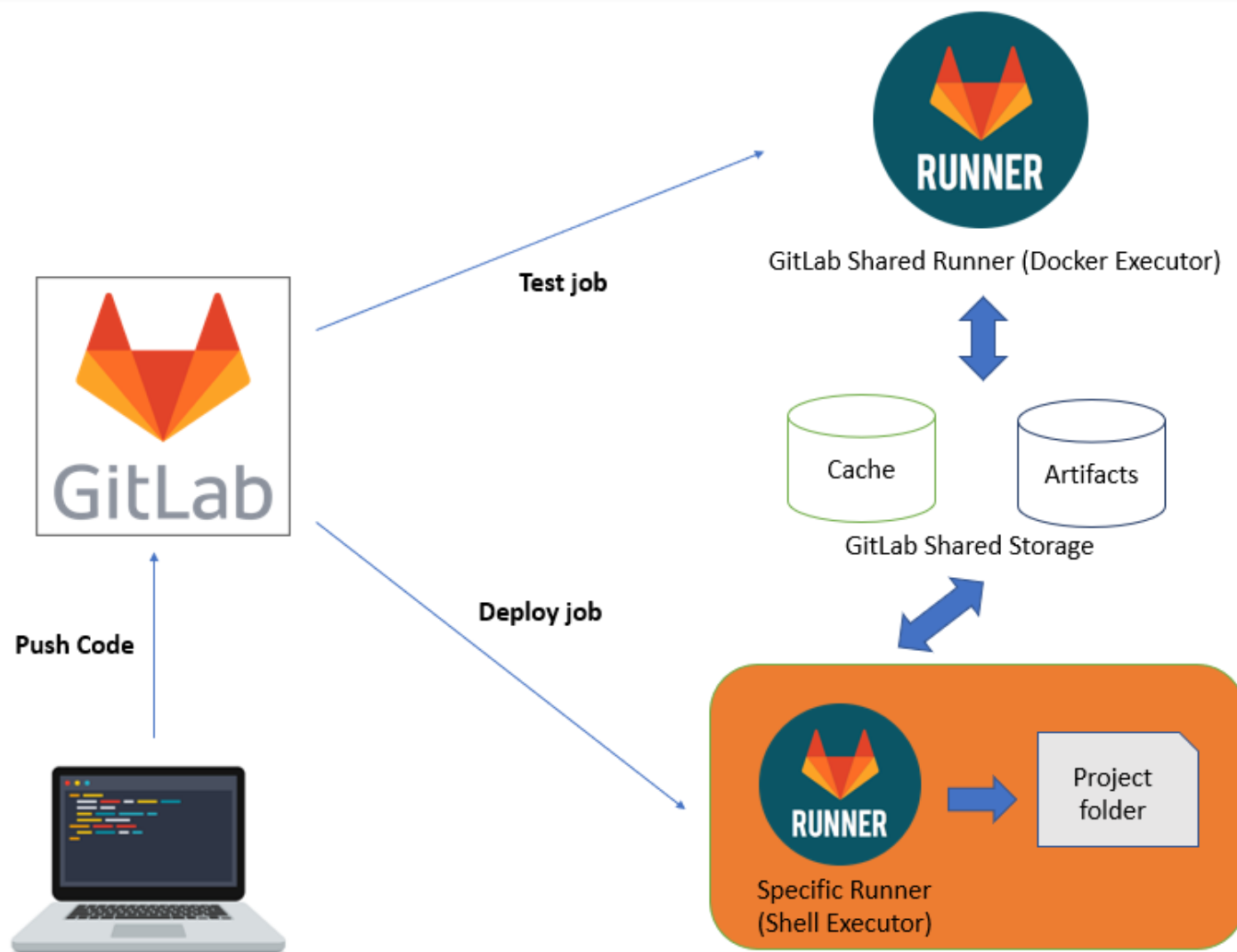
# GitLab runner

- Специальное приложение, которое работает совместно с GitLab CI для выполнения заданий
- Запускается и берет задание в работу в нужный момент
- Можно настроить необходимое количество раннеров в настройках GitLab
- Могут быть разные типы раннеров под разные задачи
- Это достаточно простое приложение, которое может быть установлено на разные ОС
  - Linux, Windows, Mac

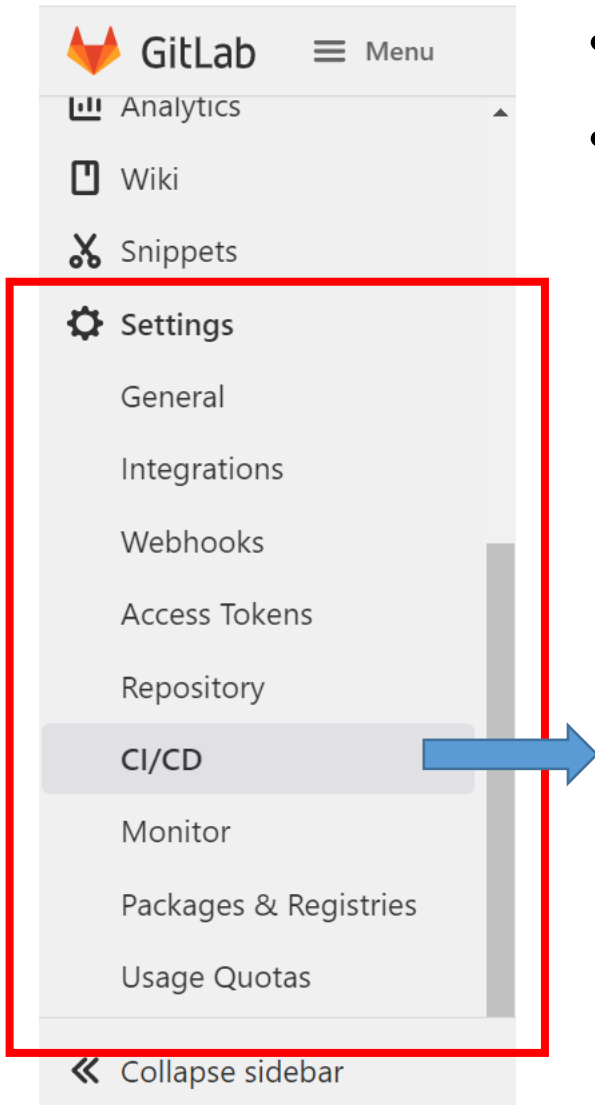
# GitLab runner

- Есть несколько видов раннеров с точки зрения доступности
- Shared runner
  - Доступен из любого проекта данного сервера gitlab
  - Доступен всем пользователям данного gitlab на конкурентной основе
  - Настраивается для экземпляра gitlab в целом
- Specific runner
  - Доступен для данного проекта только
- Group runner
  - Доступен для всех проектов конкретной группы

# GitLab runner



# GitLab runner



- Как узнать, какие раннеры доступны для проекта?
- Раздел Settings => CI/CD => Runners

## Runners

Runners are processes that pick up and execute CI/CD jobs for GitLab. [How do I configure runners?](#)

Register as many runners as you want. You can register runners as separate users, on separate servers, and on your local machine. Runners are either:

- **active** - Available to run jobs.
- **paused** - Not available to run jobs.

## Specific runners

These runners are specific to this project.

Set up a specific Runner for a project

1. [Install GitLab Runner and ensure it's running.](#)
2. Register the runner with this URL:

`https://git.miem.hse.ru/`

## Shared runners

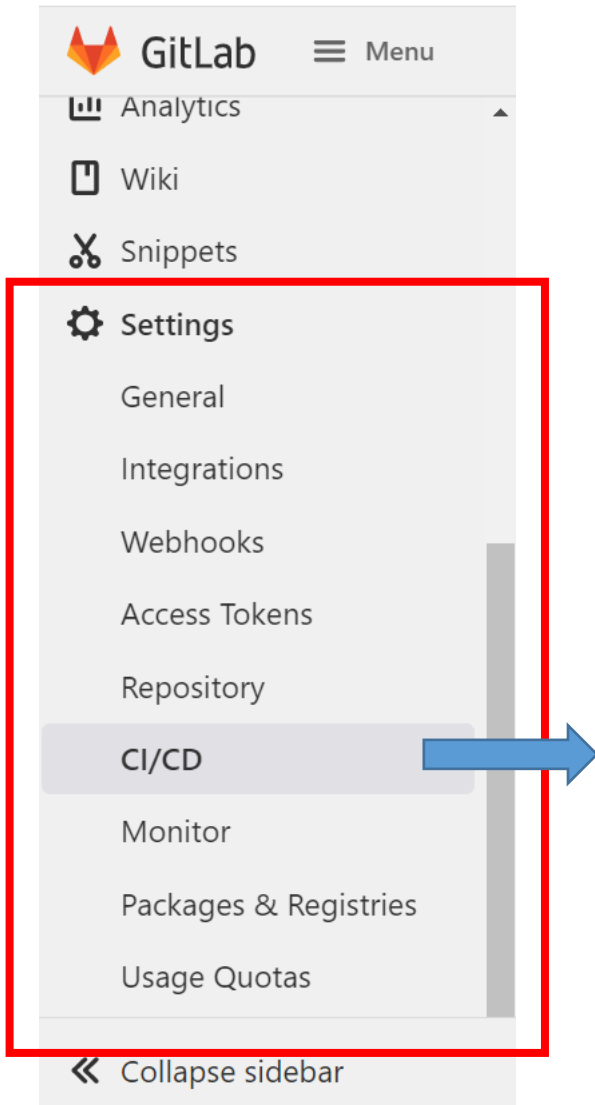
These runners are shared across this GitLab instance.

The same shared runner executes code from multiple projects, unless you configure autoscaling with [MaxBuilds](#) set to 1 (which it is on GitLab.com).

Enable shared runners for this project



# Shared runners



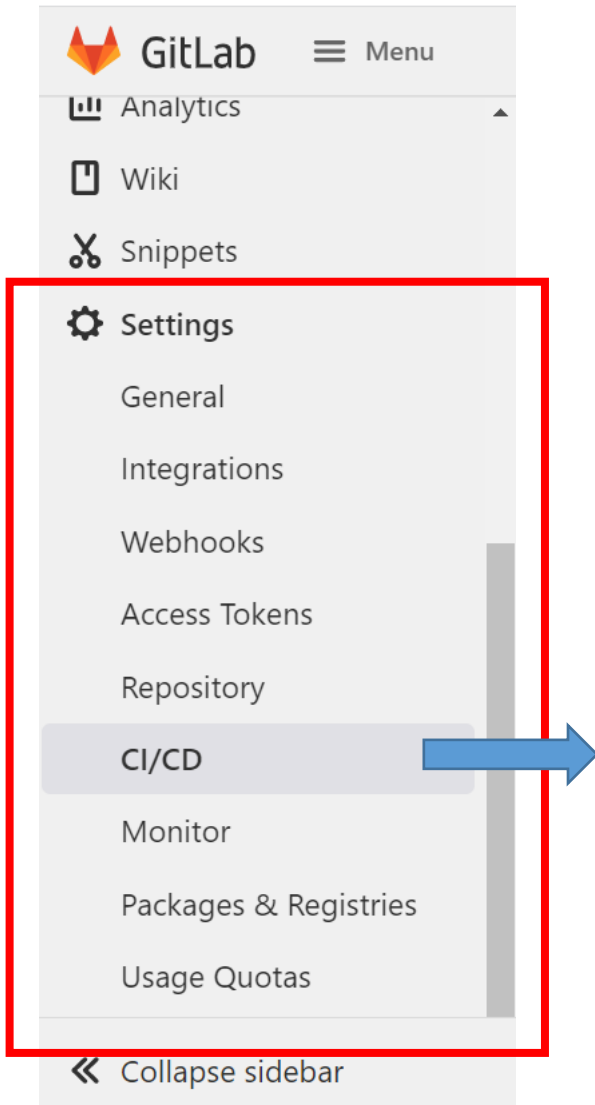
Раздел Settings => CI/CD => Runners

## Available shared runners: 4

- #73 (ZAM4rVjE)  
shared-runner-2-2  
docker
- #55 (9ZzsxiHx) 🔒  
shared cluster  
docker
- #60 (1qKTCpzJ) 🔒  
shared-cluster2  
video-lab-miem
- #72 (6ZBc681U)  
shared-runner-1-2  
docker

- Их настраивает администратор gitlab, но можно использовать
- Доступные раннеры видны в секции available shared runners

# Specific runners



Раздел Settings => CI/CD => Runners

## Available specific runners

<div><div>—</div><div>#94 (KPPnACsW)</div><div>🔒</div></div> <div>vbox-ubuntu-docker</div> <div><div>docker-linux</div><div>my</div></div>	<div><div>✎</div><div>⏸</div><div>Remove runner</div></div>
<div><div>●</div><div>#93 (q6qR52MA)</div><div>🔒</div></div> <div>my-windows-runner-2</div> <div><div>my</div><div>shell-runner</div></div>	<div><div>✎</div><div>⏸</div><div>Remove runner</div></div>
<div><div>●</div><div>#91 (C8a9KJPf)</div><div>🔒</div></div> <div>my-win-runner</div> <div><div>docker-run</div><div>my</div></div>	<div><div>✎</div><div>⏸</div><div>Remove runner</div></div>


- Можно настроить для конкретного проекта
- Достаточно установить приложение-агент на доступный хост
- Мы позже посмотрим, как установить и настроить раннер для своего проекта



# GitLab runner

- Runner это отдельные машины, их нужно настраивать и добавлять в настройках
- Смотрим, какие раннеры нам сейчас доступны
  - Надо зайти в раздел Settings -> CI/CD
  - Выбираем нужный (по тегу)
- В настройках задания надо указать тег нужного раннера

```
7  ∨ job:
8      stage: build
9      ∨ tags:
10         - docker
11  ∨ script:
12      # provide a shell script as argument for this keyword.
13      - echo "Hello World"
14
```



## Available shared runners: 4

● #55 (9ZzsxiHx) 🔒

shared cluster

docker

docker\_cache\_3

docker\_miem

● #12 (7dY1o7Pa) 🔒

shared runner 1

docker

docker\_cache\_1

docker\_itsoft

● #60 (1qKTCpzJ) 🔒

shared-cluster2

video-lab-miem

# Практика — установка своего runner

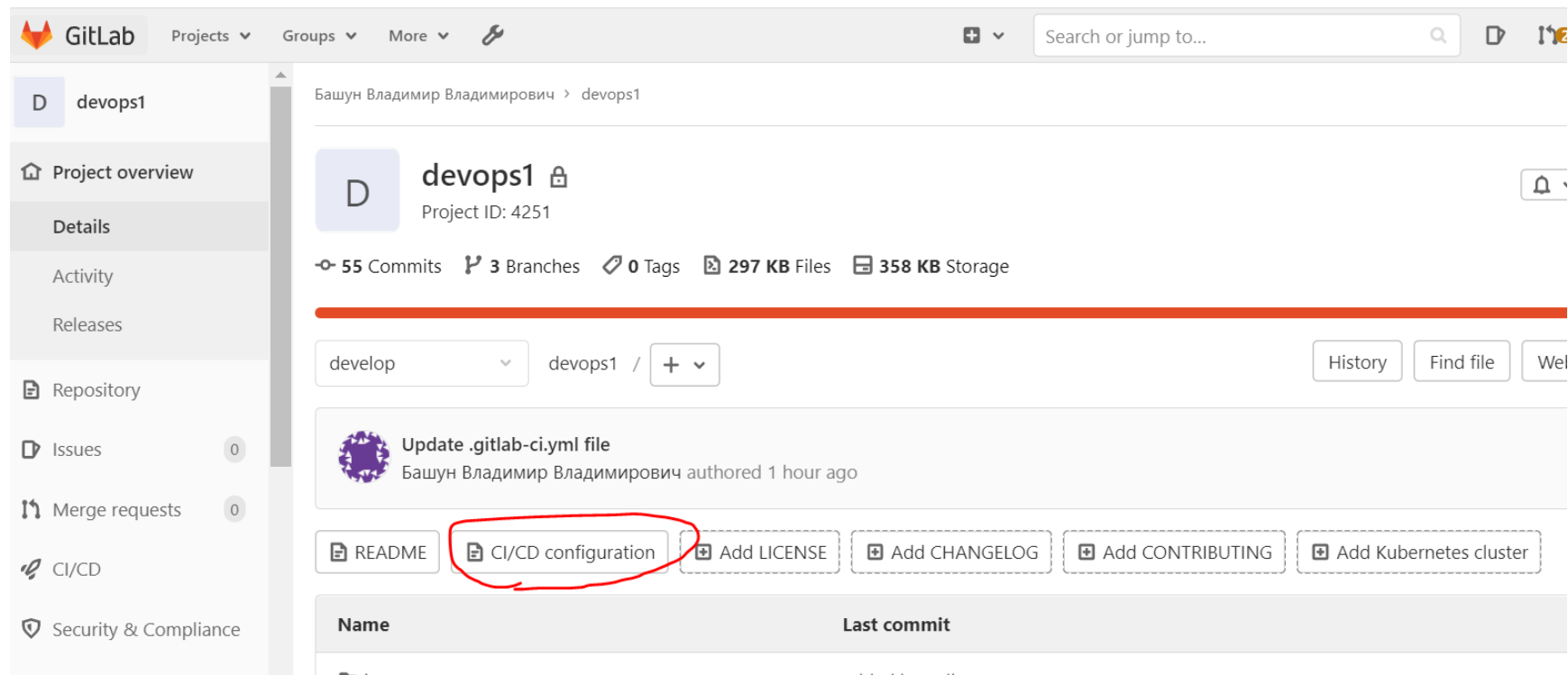
- Установим и настроим свой раннер, и зарегистрируем его в проекте

# Летняя школа DevOps/DevSecOps

GitLab CI/CD basics

# Настройка CI/CD в GitLab

- Как именно настроить CI/CD процессы в GitLab
  - Нужно создать конфигурационный yaml файл
  - По умолчанию это будет .gitlab-ci.yml



# Настройка CI/CD в GitLab

- Как и все с концепции IaC, все должно быть в форме скриптов / кода
- Используется стандартный формат файла – YAML
- Настройки действий по автоматизации процессов лежат рядом с самим проектом, в том же репозитории
  - Хотя можно настроить и по другому
- Это элемент авто-документации
  - Сразу видно, как нужно собирать/тестировать/разворачивать продукт

# Создание заданий (jobs)

- Необходимо определить следующие сущности
  - Задания (jobs) – определяют, что именно надо делать
  - Этапы (stages) – определяют, когда именно надо выполнять работы
  - Конвейеры (pipelines) – собирают все вместе, определяют поэтапное выполнение заданий

# Настройка pipelines

- Это компонент верхнего уровня
- Pipelines могут быть достаточно сложными
  - объединять несколько проектов,
  - завязаны на merge request в отличие от комитов
  - родитель-потомок и т.п.
- Мы рассмотрим простые pipeline (базовые)
- Pipeline формируется из отдельных заданий (jobs)
  - Т.е. надо сначала сформировать задание и добавить его в конвейер

# Создаем задание

- Это самый базовый элемент, из которого строятся конвейеры работ при непрерывной интеграции или доставке
- Минимально у задания есть название и сценарий (script)

```
7   job:  
8   |   script:  
9   |       # provide a shell script as argument for this keyword.  
10  |       - echo "Hello World"
```

- В script можно писать bash команды
  - Можно набор команд, можно запустить bash скрипт, лежащий в репозитории
  - Дальше увидим, какие вообще команды допустимы



# Создаем задание

- Задаются прямо в gitlab-ci файле
  - Создадим работы run\_unit\_test, run\_lint\_test, build\_image, push\_image

```
run_tests:  
  script:  
    - echo "Running tests..."  
  
build_image:  
  script:  
    - echo "Building the docker image..."  
    - echo "Tagging the docker image"  
  
push_image:  
  script:  
    - echo "Logging into docker registry..."  
    - echo "Pushing docker image to registry"
```

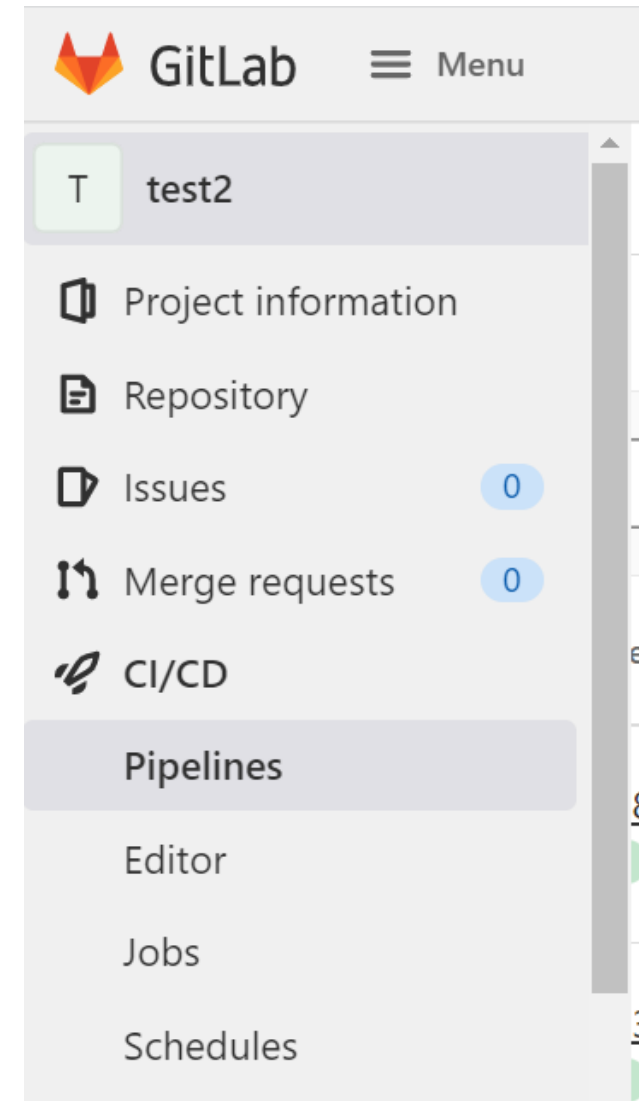
# Создаем задание

- Задаются прямо в gitlab-ci файле
  - Создадим работы run\_unit\_test, run\_lint\_test, build\_image, push\_image
- Кроме раздела script, можно так же добавить before\_script, after\_script

```
1  run_tests:
2    before_script:
3      - echo "Preparing test data..."
4    script:
5      - echo "Running tests..."
6    after_script:
7      - echo "Cleaning up temporary files..."
8
9  build_image:
10    script:
11      - echo "Building the docker image..."
12      - echo "Tagging the docker image"
```

# Запуск задания

- Создадим несколько заданий, и автоматически создастся конвейер
- Результат выполнения можно увидеть в разделе pipelines



# Запуск задания

- Результат выполнения можно увидеть в разделе pipelines
- Видим, что работа не выполняется, т.к. нет подходящего раннера (runner)





🕒 3 jobs for **master**



🚩 **latest** **stuck**

🔑 **b467fe6d** 🗄️

🔗 No related merge requests found.

Pipeline Needs Jobs **3** Tests **0**

Build	Test
 <b>build-job</b> 	 <b>lint-test-job</b> 

 **pending** **Job build-job** created 1 minute ago by  **Башун Владимир Владимирович**

⚠️ This job is stuck because the project doesn't have any runners online assigned to it.  
Go to project [CI settings](#)

# Запуск задания

- Результат выполнения можно увидеть в разделе pipelines
- Видим, что работа не выполняется, т.к. нет подходящего раннера (runner)
- Позже разберемся что это, а пока добавим еще раздел tags: - docker

```
✓ deploy-job:      # This job runs in the deploy s  
| stage: deploy    # It only runs when *both* jobs  
| tags:              
|   - docker         
✓ script:            
|   - echo "Deploying application..."  
|   - echo "Application successfully deployed."
```

# Запуск задания

- Возвращаемся в раздел pipelines, видим, что работы запущены

devops > test2 > Pipelines

All14FinishedBranchesTags

Clear runner cachesCI lintRun pipeline

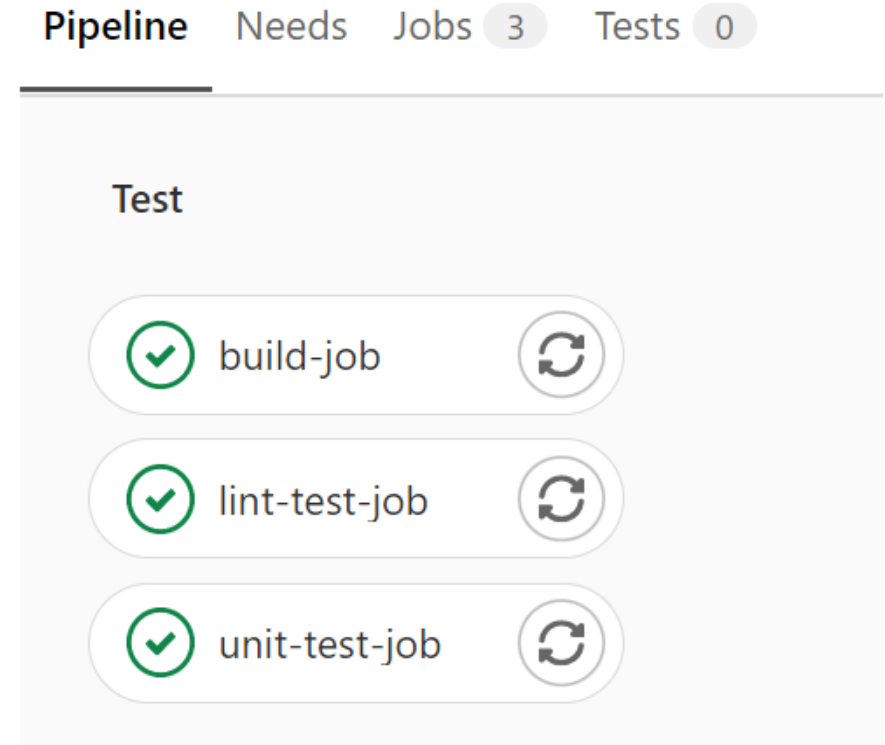
Filter pipelines

Show Pipeline ID

Status	Pipeline ID	Triggerer	Commit	Stages	Duration
<div>running</div>	<div>#71572</div> <div>latest</div>	<div></div>	<div>master 7c747622</div> <div>Update .gitlab-ci.yml ...</div>	<div></div>	<div>In progress</div>

# Запуск задания

- Возвращаемся в раздел pipelines, видим, что работы запущены
- Можно зайти внутрь конвейера и посмотреть сами работы








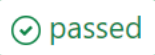




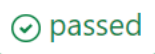






# Запуск задания

- Так же их видно в разделе Jobs

devops > test2 > Jobs

All 31 Pending 0 Running 0 Finished 29

Status	Name	Job	Pipeline	Stage	Duration	Coverage
	lint-test-job	#153533  master  7c747622 docker-linux	#71572 by 	test	 00:00:16  4 minutes ago	
	unit-test-job	#153532  master  7c747622 docker-run	#71572 by 	test	 00:00:24  4 minutes ago	
	build-job	#153531  master  7c747622 shell-runner	#71572 by 	test	 00:00:05  3 minutes ago	



# Запуск задания

- Логи выполнения самого задания

```
8  Preparing environment 00:01
9  Running on runner-kppnacs-w-project-7914-concurrent-0 via student-VirtualBox...
11 Getting source from Git repository 00:03
12 Fetching changes with git depth set to 50...
13 Reinitialized existing Git repository in /builds/devops/test2/.git/
14 Checking out 7c747622 as master...
15 Skipping Git submodules setup
17 Executing "step_script" stage of the job script 00:05
18 Using docker image sha256:9c6f0724472873bb50a2ae67a9e7adcb57673a183cea8b06eb778dca85918
    1b5 for alpine:3.16 with digest alpine@sha256:bc41182d7ef5ffc53a40b044e725193bc10142a124
    3f395ee852a8d9730fc2ad ...
19 $ echo "Linting code... This will take about 10 seconds."
20 Linting code... This will take about 10 seconds.
21 $ sleep 4
22 $ echo "No lint issues found."
23 No lint issues found.
25 Cleaning up project directory and file based variables 00:01
27 Job succeeded
```

**Duration:** 16 seconds

**Timeout:** 1h (from project) ?

**Runner:** #94 (KPPnACsW) vbox-ubuntu-docker

**Tags:** docker-linux

Commit [7c747622](#) ?

Update .gitlab-ci.yml file

✓ Pipeline #71572 for master ?

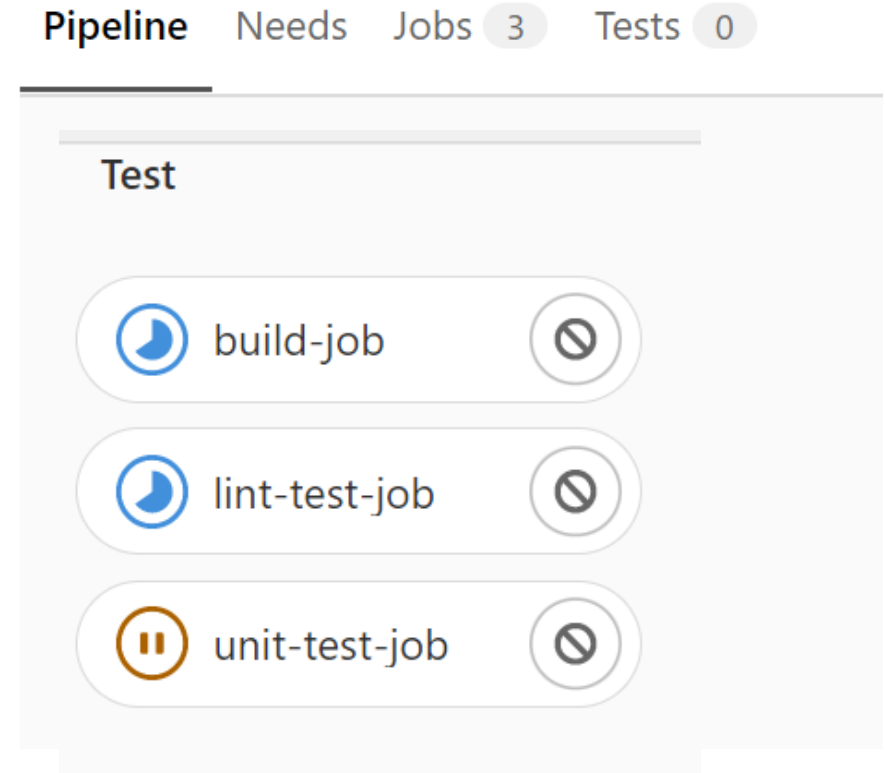
test

✓ build-job

→ ✓ lint-test-job

# Запуск задания

- Можно заметить, что работы запустились одновременно
- Это явно не то, что мы хотели, т.к. нельзя протестировать до сборки и т.п.
- Хочется выстроить последовательную цепочку
  - Сборка => Тестирование => Упаковка



# Настройка этапов

- Чтобы настроить порядок выполнения работ, нужно задать этапы
- Вспоминаем основные стадии
  - Сборка, тестирование, упаковка, заливка на staging сервер, заливка на production сервер
- Можно определять их столько, сколько нужно по необходимости
  - В файле `.gitlab-ci.yml` это будут stages

`.gitlab-ci.yml` 1.41 KB

```
1 # This file is a template, and might need editing
2 # This file is a template demonstrating the `scri
3 # Learn more about this keyword here: https://doc
4
5 # After committing this template, visit CI/CD > J
6 stages:
7   - build
8   - testing
9   - staging
10  - production
11
```

# Настройка этапов

- Определяем некоторые обязательные параметры
  - К какому этапу относится задание

```
7  ✓ job:
8    stage: build
9    ✓ script:
10         # provide a shell script as argument for this keyword.
11         - echo "Hello World"
12
```

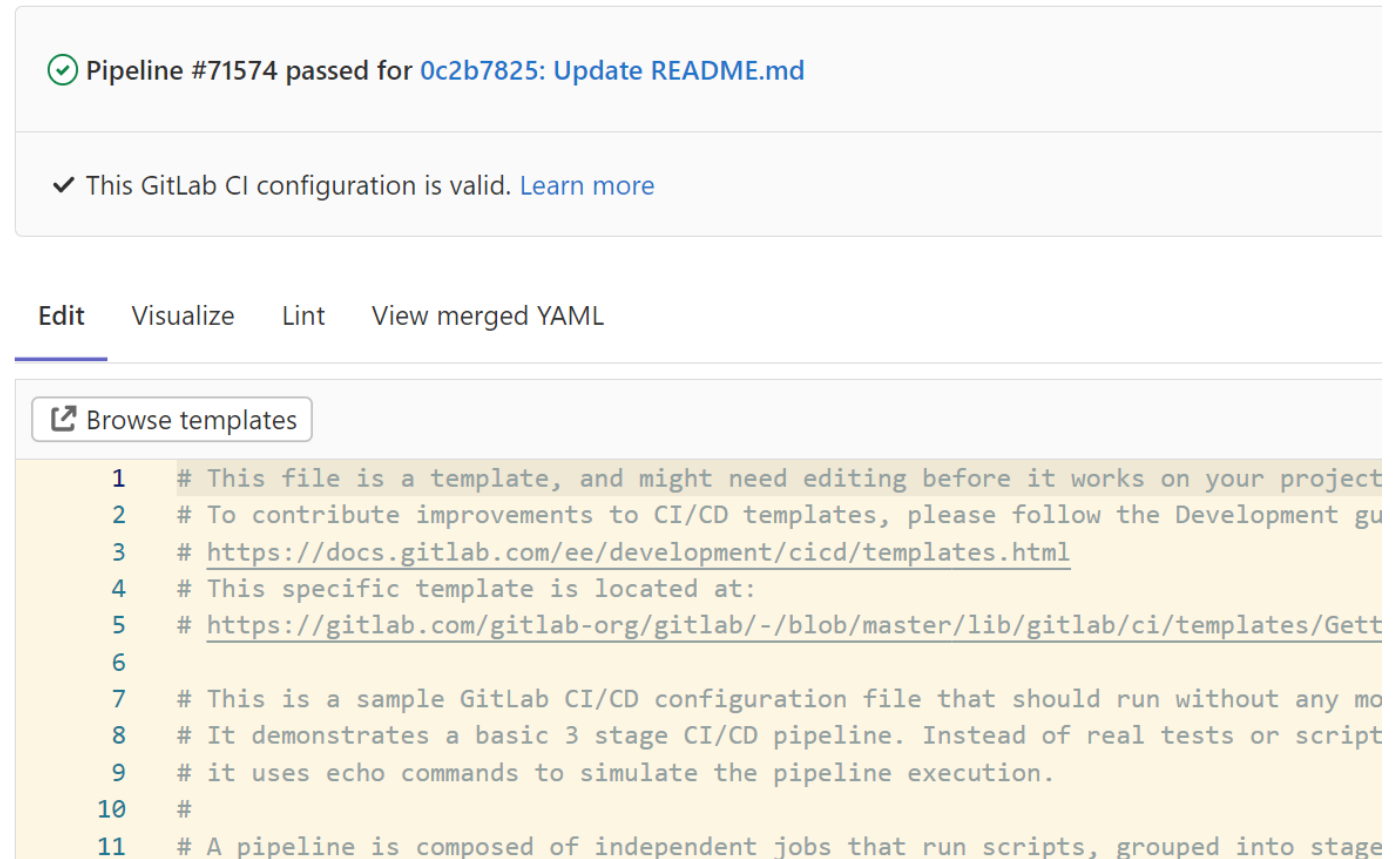
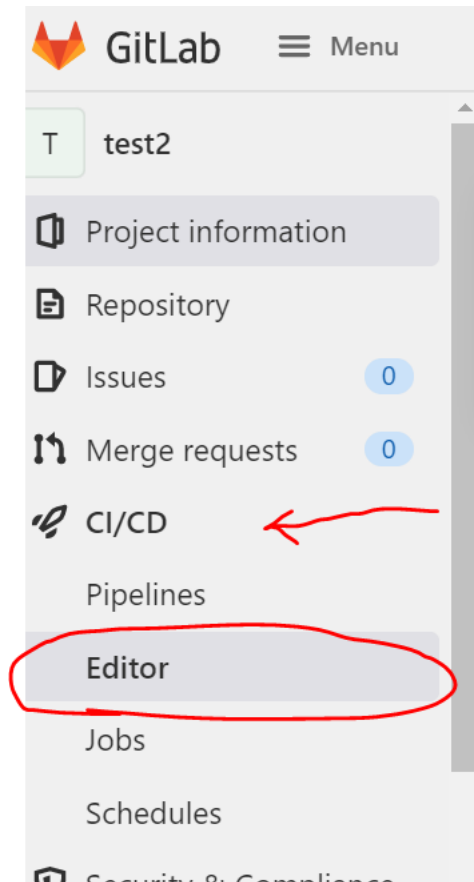
- Задания будут выполняться по этапам

# Настройка этапов

- Позволяют настроить последовательность запуска заданий
  - Т.е. например сначала произойдет сборка, потом тестирование, потом сборка образа, а потом уже отправка образа в репозиторий
- Позволяют контролировать ход выполнения
  - Если на одном этапе задание не выполнится, конвейер остановится (остальные выполняться не будут)
- Позволяет группировать задания одного этапа
  - Разные виды тестирования


# Pipeline editor


- Отдельный интерфейс для настройки pipeline через веб-приложение




# Pipeline editor

- Кроме собственно редактирования ci файла дает дополнительные возможности

 Pipeline #71574 passed for [0c2b7825: Update README.md](#)

 This GitLab CI configuration is valid. [Learn more](#)

Edit Visualize Lint View merged YAML

 Browse templates

```
1 # This file is a template, and might need editing before it works on your project
2 # To contribute improvements to CI/CD templates, please follow the Development gu
3 # https://docs.gitlab.com/ee/development/cicd/templates.html
4 # This specific template is located at:
5 # https://gitlab.com/gitlab-org/gitlab/-/blob/master/lib/gitlab/ci/templates/Gett
6
7 # This is a sample GitLab CI/CD configuration file that should run without any mo
8 # It demonstrates a basic 3 stage CI/CD pipeline. Instead of real tests or script
9 # it uses echo commands to simulate the pipeline execution.
10 #
11 # A pipeline is composed of independent jobs that run scripts, grouped into stage
```

# Pipeline editor

- Кроме собственно редактирования ci файла дает дополнительные возможности

Быстрый доступ и информация о последнем запуске конвейера

The screenshot displays the GitLab Pipeline Editor interface. At the top, a green checkmark icon indicates that 'Pipeline #71574 passed for 0c2b7825: Update README.md'. Below this, a message states 'This GitLab CI configuration is valid. Learn more'. The interface includes tabs for 'Edit', 'Visualize', 'Lint', and 'View merged YAML'. A 'Browse templates' button is visible. The main area shows a sample CI/CD configuration file with the following content:

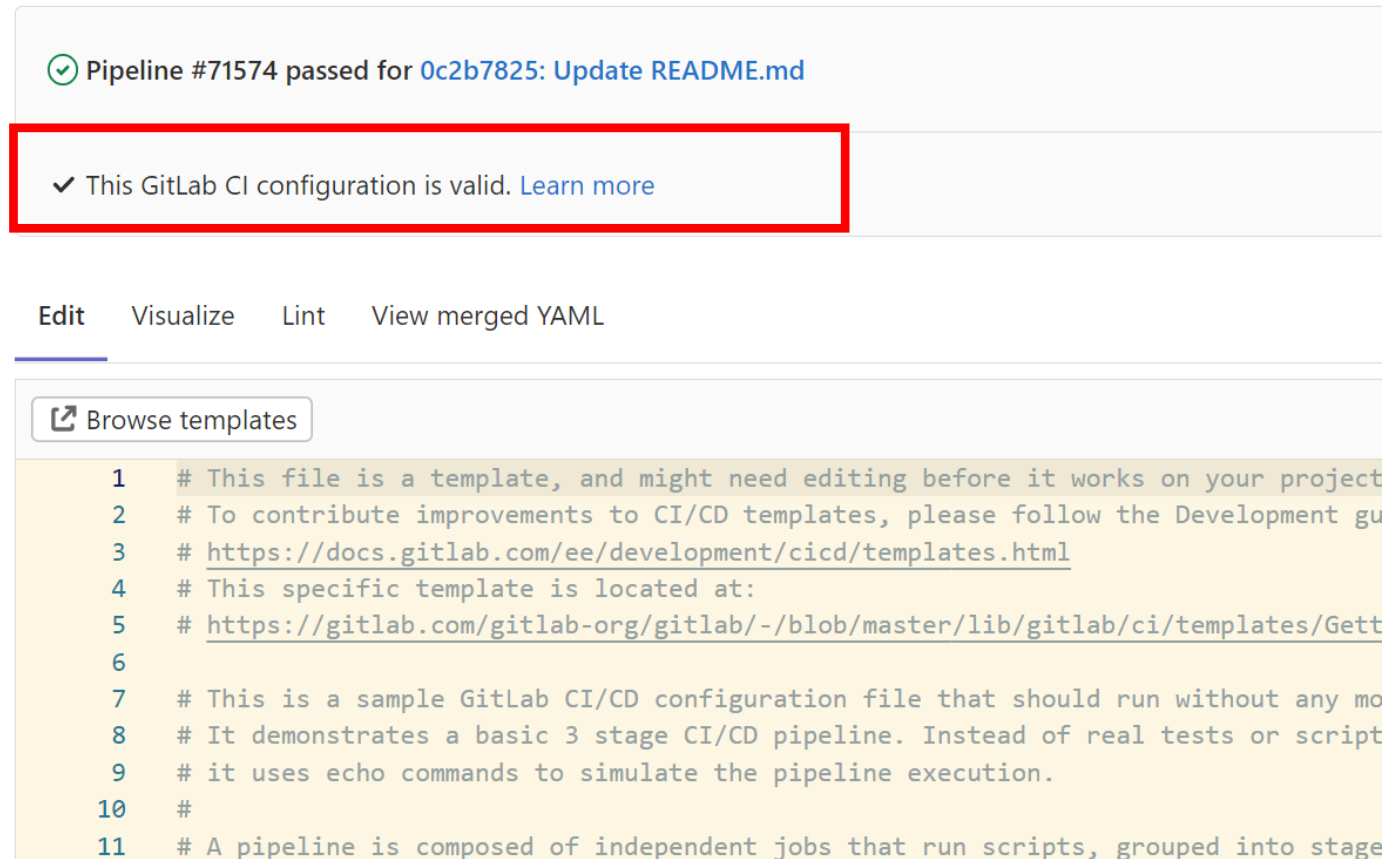
```
1 # This file is a template, and might need editing before it works on your project
2 # To contribute improvements to CI/CD templates, please follow the Development gu
3 # https://docs.gitlab.com/ee/development/cicd/templates.html
4 # This specific template is located at:
5 # https://gitlab.com/gitlab-org/gitlab/-/blob/master/lib/gitlab/ci/templates/Gett
6
7 # This is a sample GitLab CI/CD configuration file that should run without any mo
8 # It demonstrates a basic 3 stage CI/CD pipeline. Instead of real tests or script
9 # it uses echo commands to simulate the pipeline execution.
10 #
11 # A pipeline is composed of independent jobs that run scripts, grouped into stage
```



# Pipeline editor

- Кроме собственно редактирования ci файла дает дополнительные возможности

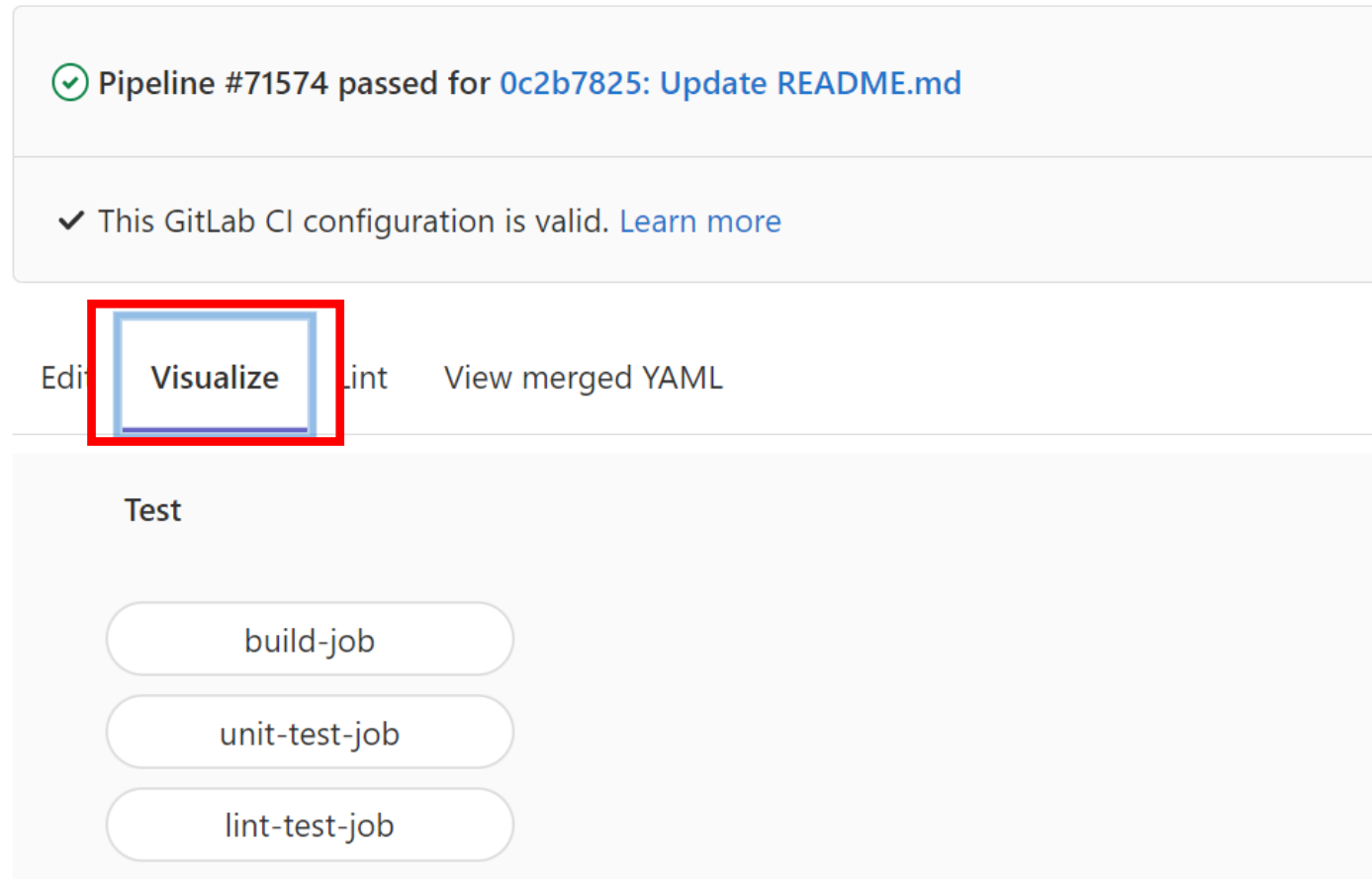
Быстрая проверка  
конфигурации и  
синтаксиса ci yaml  
файла



# Pipeline editor

- Кроме собственно редактирования ci файла дает дополнительные возможности

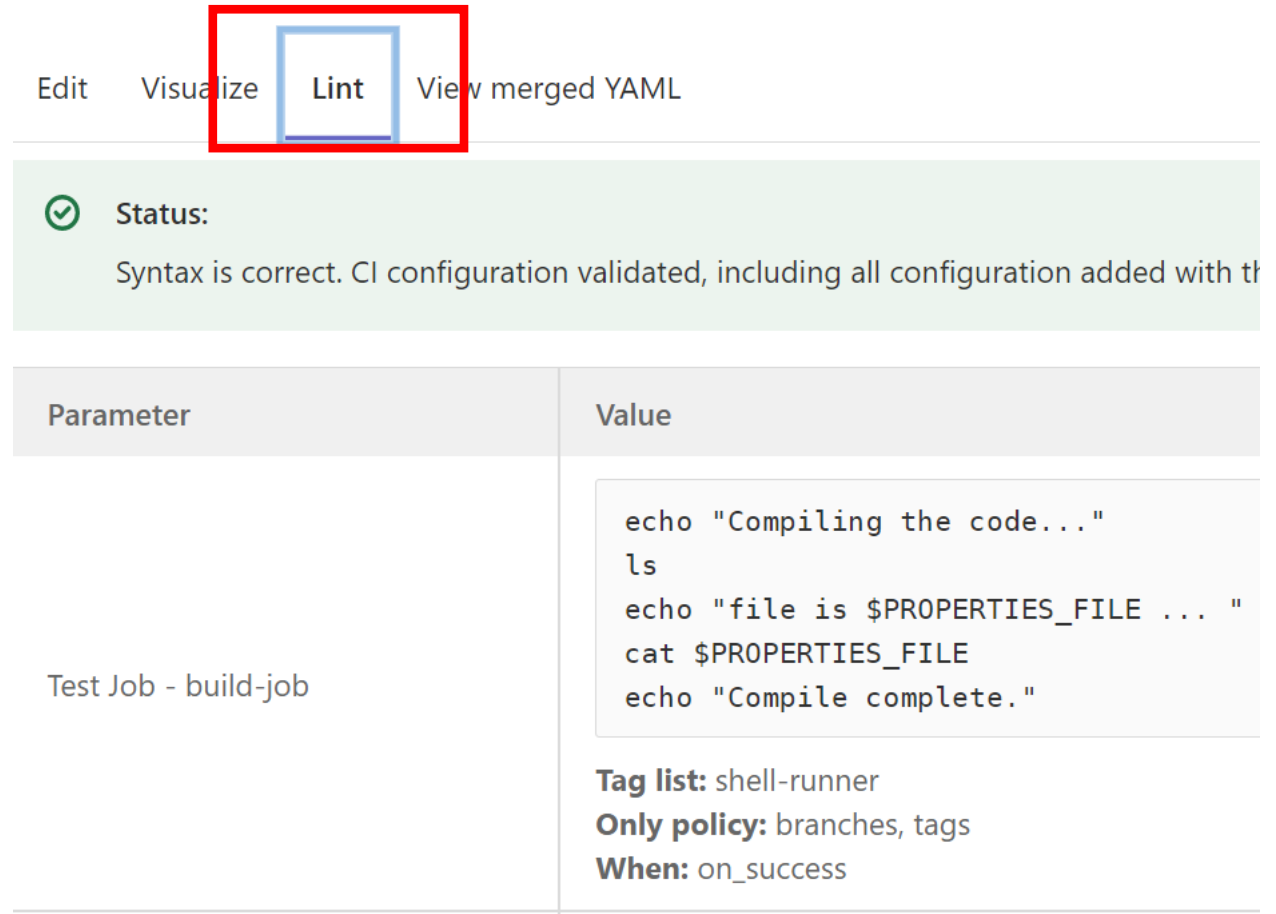
Визуализация  
конвейера



# Pipeline editor

- Кроме собственно редактирования ci файла дает дополнительные возможности

Альтернативный вариант просмотра с дополнительной информацией



The screenshot shows the 'Pipeline editor' interface. At the top, there are four tabs: 'Edit', 'Visualize', 'Lint', and 'View merged YAML'. The 'Lint' tab is selected and highlighted with a blue border, and it is also enclosed in a red rectangle. Below the tabs, there is a green status bar with a checkmark icon and the text 'Status: Syntax is correct. CI configuration validated, including all configuration added with th'. Below this, there is a table with two columns: 'Parameter' and 'Value'. The table has one row with the parameter 'Test Job - build-job' and a value containing a shell script snippet and metadata.

Parameter	Value
Test Job - build-job	<pre>echo "Compiling the code..." ls echo "file is \$PROPERTIES_FILE ... " cat \$PROPERTIES_FILE echo "Compile complete."</pre> <p><b>Tag list:</b> shell-runner <b>Only policy:</b> branches, tags <b>When:</b> on_success</p>

# Настройка этапов

- Создадим стадии build, test, package, deploy
- Распределим задания по стадиям
  - На каждой стадии может быть несколько заданий
  - Build
    - build-job
  - Test
    - Unit-test-job
    - Lint-test-job
  - Package
    - build-image
    - push-image

# Pipeline editor

- Получится вариант, разбитый на этапы выполнения

✓ Pipeline #71574 passed for 0c2b7825: Update README.md

✓ This GitLab CI configuration is valid. [Learn more](#)








Edit **Visualize** Lint View merged YAML

Build	Test	Deploy
build-job	unit-test-job	deploy-job
	lint-test-job	

# Pipeline editor

- Сохраним изменения и посмотрим, как изменится запуск
  - Видно, что добавились стадии

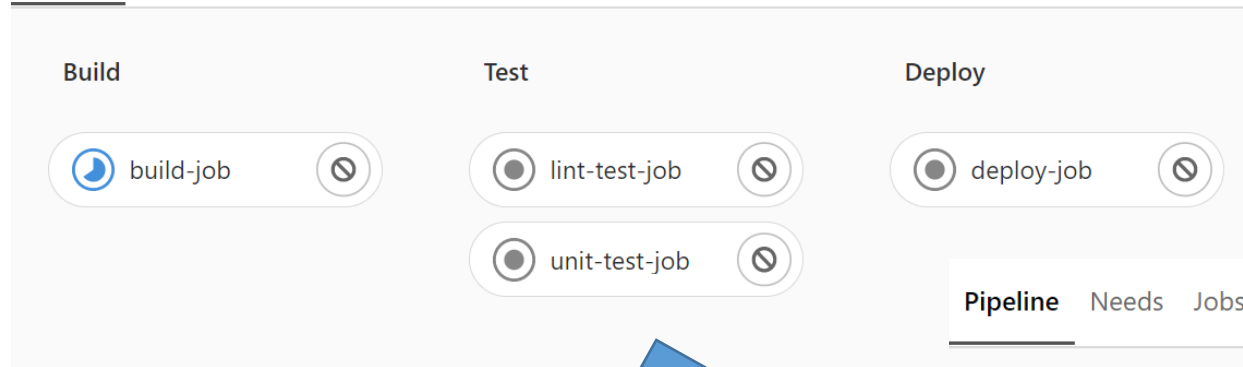
devops > test2 > Pipelines

All 18 Finished Branches Tags				Clear runner caches	CI lint	Run pipeline
Filter pipelines				Q	Show Pipeline ID v	
Status	Pipeline ID	Triggerer	Commit	Stages	Duration	
 pending	<a href="#">#71597</a> latest		 master  <a href="#">b3f60c97</a>  Update .gitlab-ci.yml ...		 In progress	

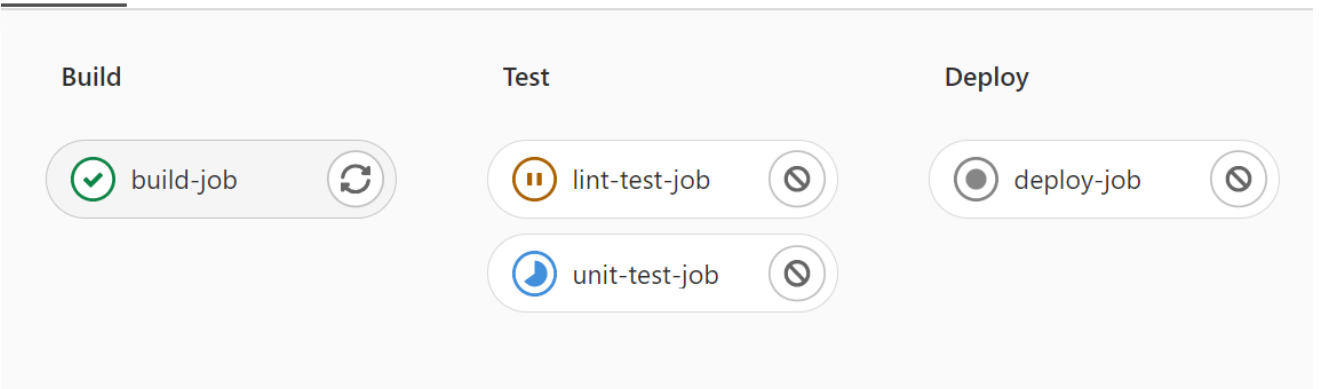
# Pipeline editor

- Если открыть информацию о pipeline, видно, что работы запускаются по этапам

Pipeline Needs Jobs 4 Tests 0



Pipeline Needs Jobs 4 Tests 0





# ЗАЩИТА ОТ ДУРАКА ЧЕЛОВЕЧЕСКОГО ФАКТОРА

## С помощью инструментов Gitlab

- Деплоить только из master ветки
- Управлять ролями участников проекта
- Запретить пуш в мастер ветку вне MR
- Запретить мердж MR без прохождения пайплайна и апрувов
- Использовать защищенные ветки
- Запретить пушить секреты
- Защитить секретные переменные





# ХОРОШИЕ ПРАКТИКИ

## Построения релизного процесса

1. Меньше дублирования – 1 репозиторий с CI на все проекты
2. Ручной запуск «опасных» джоб
3. Экономия ресурсов ранеров
4. Не использовать `tag latest!`
5. Отделять `develop`-образы от релизных