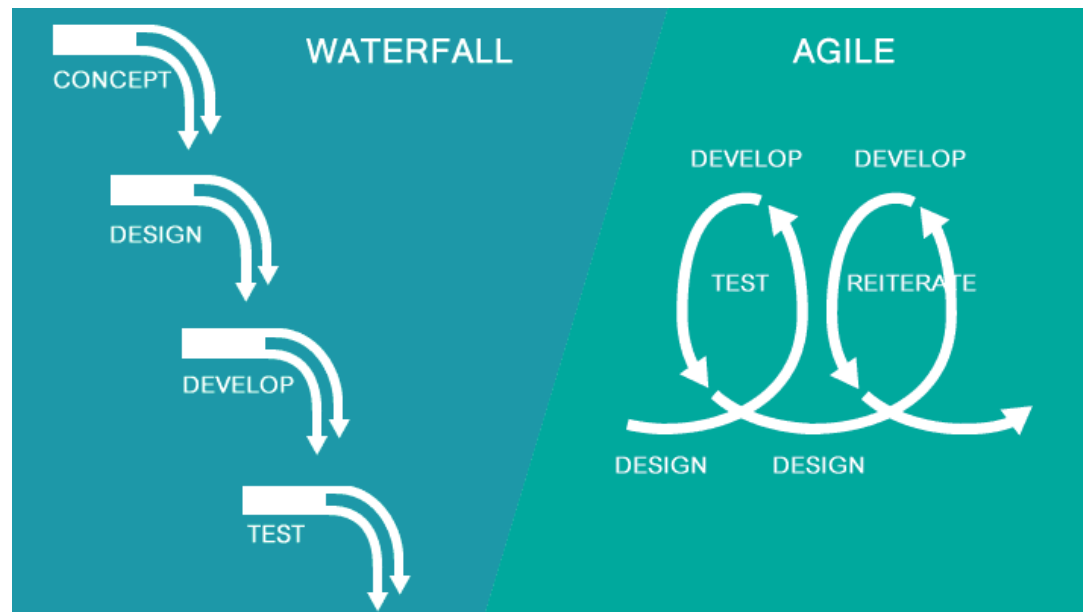


CI/CD для ML

Гущин Александр

DMIA Production ML  весна 2021

Мы уже выяснили, что есть разные подходы к разработке продукта. Подходы с частыми итерациями предполагают регулярное обновление кода, работающего в production.



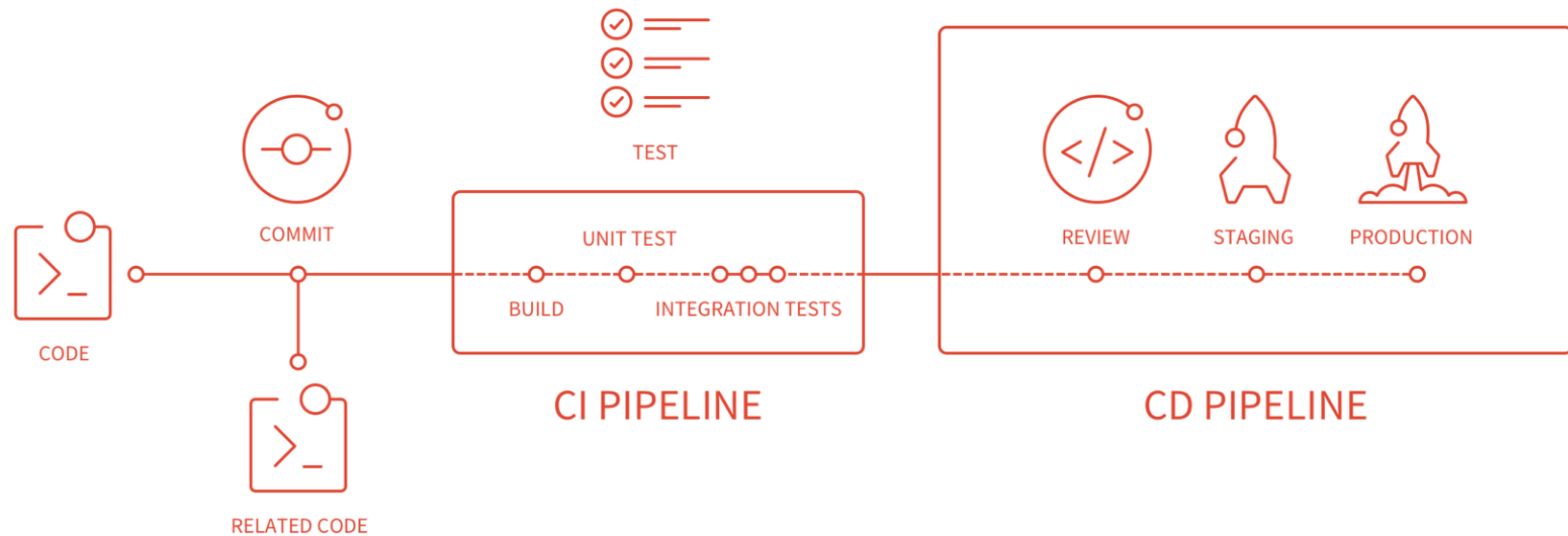
<https://blog.planview.com/waterfall-or-agile/>

Для этого придуманы практики под названиями:

- CI - Continious Integration
- CD - Continious Delivery (or Deployment)

Цель этих практик - сократить разрыв между разработкой приложений и управлением их работой с помощью **автоматизации** сборки, тестирования и деплоя. CI/CD является основой DevOps практик.

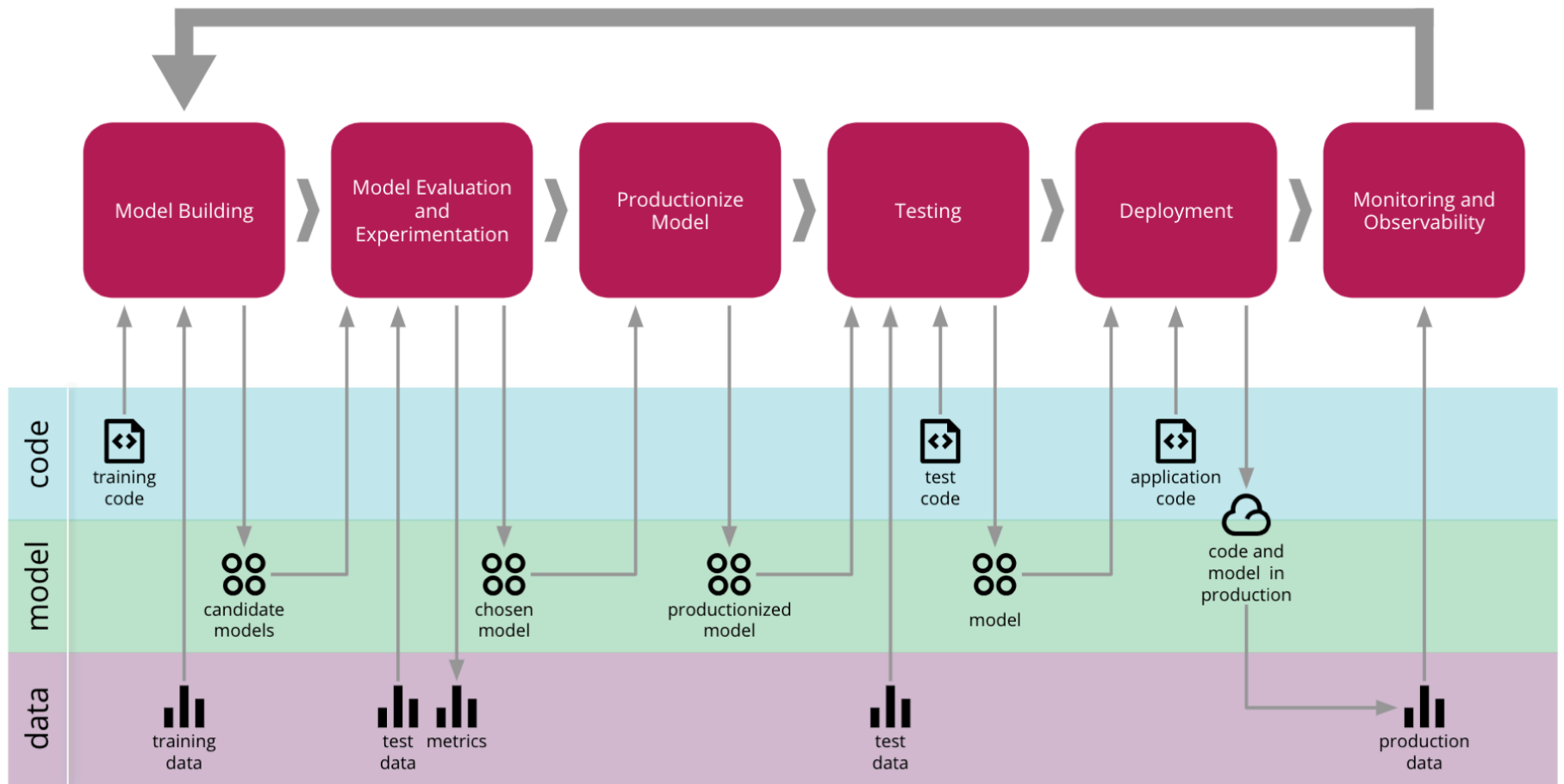
Представим, что мы разрабатываем обычное приложение, без ML. Тогда наш CI/CD может выглядеть так:



Иногда отдельно выделяют Continuous Testing, Continuous Monitoring.

<https://forge.etsi.org/rep/help/ci/README.md>

Вместе с ML мы имеем такие этапы нашего пайплайна

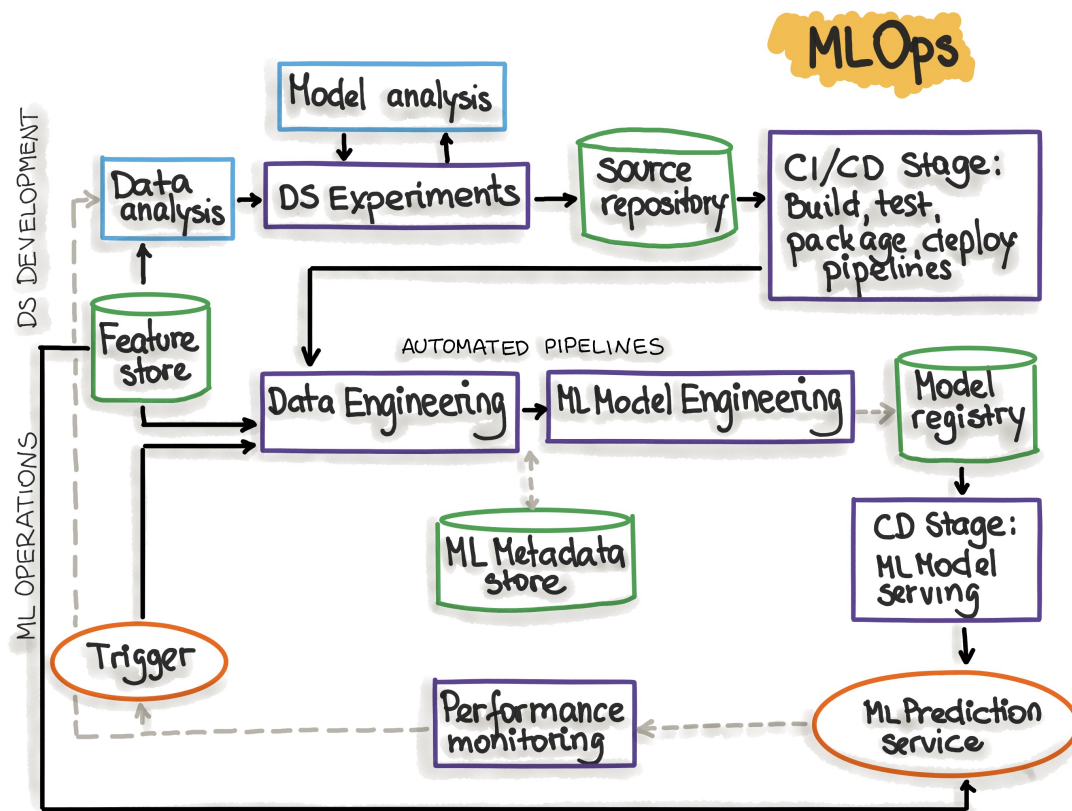


<https://martinfowler.com/articles/cd4ml.html>

ML and other software systems are similar in continuous integration of source control, unit testing, integration testing, and continuous delivery of the software module or the package. However, in ML, there are a few notable differences:

- CI is no longer only about testing and validating code and components, but also testing and validating data, data schemas, and models.
- CD is no longer about a single software package or a service, but a system (an ML training pipeline) that should automatically deploy another service (model prediction service).
- CT is a new property, unique to ML systems, that's concerned with automatically retraining and serving the models.

Вместе с ML наш CI/CD должен быть похож на



<https://ml-ops.org/content/mlops-principles>

<https://cloud.google.com/solutions/machine-learning/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning>

Возможный CI/CD для Шляпы

Стадии выполняются одна за другой.

Continuous X	Stage	Triggered by
CI for pipeline	test pipeline	pipeline change
CI for pipeline	build pipeline	
CT	run pipeline	schedule run
CT	update model	
CI for api	test api	api change
CI for api	build api	
CD for api	deploy api	

Continuous Integration

На этом этапе мы обычно:

- запускаем тесты на ML Pipeline
- готовим ML Pipeline для деплоимента, например, пакуем в контейнер
- запускаем тесты на ML API
- готовим ML API для деплоимента

Continious Delivery (Deployment)

На этом этапе мы обычно:

- разворачиваем / настраиваем инфраструктуру для деплоев
- деплоим ML Pipeline
- деплоим ML API

Continuous Training

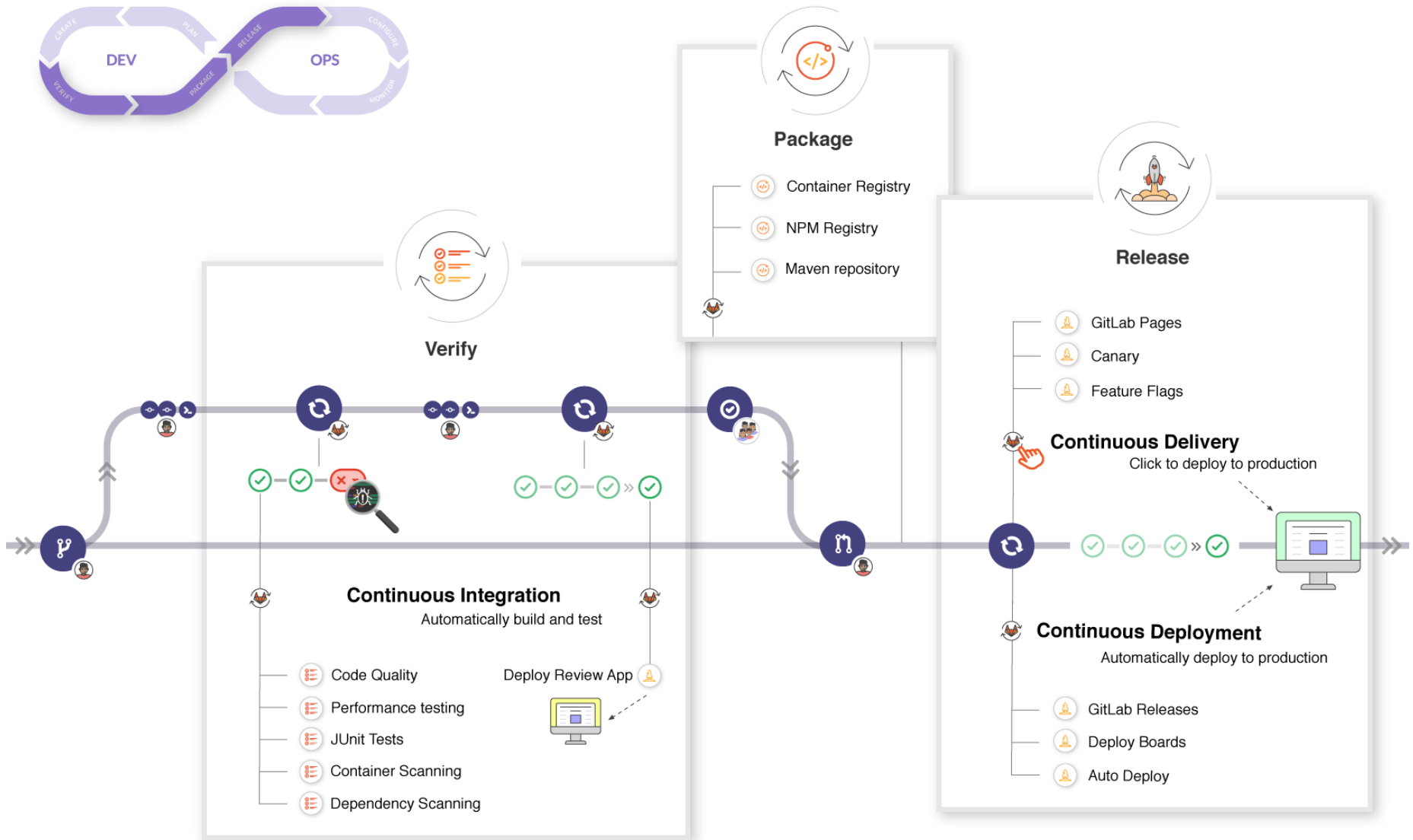
На этом этапе мы запускаем ML Pipeline:

- подтягиваем свежие данные, валидируем их и подготавливаем для обучения
- переобучаем модель, ищем оптимальные гиперпараметры
- сохраняем метрики и артефакты
- принимаем решение об обновлении модели в проде и, при необходимости, триггерим выполнение CI/CD для api

Инструменты для CI/CD

- Их огромное количество. Одни из наиболее популярных: Jenkins, Circle CI, Gitlab CI, Teamcity, Travis CI...
- Не так важно, какой инструмент выбрать для CI/CD, главное - использовать его.
- Хорошая идея - реализовать "пустой" пайплайн, а потом заполнить его настоящим кодом.

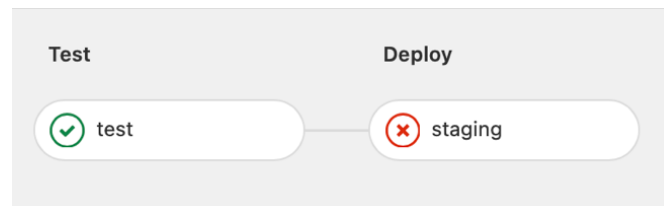
Gitlab CI/CD



Файл `.gitlab-ci.yml` в корне репозитория

Что происходит

Создается и выполняется граф



Каждая джоба выполняется в контейнере на сервере с Gitlab runner

Gitlab Runner --> Container --> Job

Есть разные способы организации пайплайнов:

https://docs.gitlab.com/ee/ci/pipelines/pipeline_architectures.html

Gitlab Runners

The screenshot shows the GitLab web interface for the 'Runners' settings. The left sidebar contains navigation links: CI / CD, Operations, Packages & Registries, Analytics, Wiki, Snippets, Members, Settings (highlighted), General, Integrations, Webhooks, Repository, CI / CD (highlighted), Operations, and Pages. At the bottom of the sidebar is a 'Collapse sidebar' button.

The main content area is titled 'Runners' and includes a 'Collapse' button. It explains that runners are processes that execute CI/CD jobs and provides a link to 'How do I configure runners?'. It states that runners can be registered as separate users, on separate servers, or on a local machine, and lists two states: 'active' (available to run jobs) and 'paused' (not available to run jobs).

There are two main sections: 'Specific runners' and 'Shared runners'.

Specific runners

These runners are specific to this project.

Set up a specific runner automatically

Register a runner on a Kubernetes cluster. [Learn more.](#)

1. Click the button below.
2. Select an existing Kubernetes cluster or create a new one.
3. From the Kubernetes cluster details view, applications list, install GitLab Runner.

[Install GitLab Runner on Kubernetes](#)

Set up a specific runner manually

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

Shared runners

These runners are shared across this GitLab instance.

[Shared Runners on GitLab.com](#) run in **autoscale mode** and are powered by Google Cloud Platform. Autoscaling means reduced wait times to spin up builds, and isolated VMs for each project, thus maximizing security.

They're free to use for public open source projects and limited to 2000 CI minutes per month per group for private projects. Read about all [GitLab.com plans](#).

Enable shared runners for this project

☒

Available shared runners: 15

● 6QgxEPvR

windows-shared-runners-manager-2 #1506021

[shared-windows-1000](#) [shared-windows-1000](#) [shared-windows-1000](#)

Артефакты

Передаем между джобами

```
1 image: python:latest
2
3 pages:
4   script:
5     - pip install sphinx sphinx-rtd-theme
6     - cd doc ; make html
7     - mv build/html/ ../public/
8   artifacts:
9     paths:
10    - public
11  only:
12    - master
```

[https://gitlab.com/gitlab-org/gitlab/-
/blob/master/lib/gitlab/ci/templates/Python.gitlab-ci.yml](https://gitlab.com/gitlab-org/gitlab/-/blob/master/lib/gitlab/ci/templates/Python.gitlab-ci.yml)

Кэш

```
1 # Change pip's cache directory to be inside the project directory since we can
2 # only cache local items.
3 variables:
4   PIP_CACHE_DIR: "$CI_PROJECT_DIR/.cache/pip"
5
6 # Pip's cache doesn't store the python packages
7 # https://pip.pypa.io/en/stable/reference/pip_install/#caching
8 #
9 # If you want to also cache the installed packages, you have to install
10 # them in a virtualenv and cache it as well.
11 cache:
12   paths:
13     - .cache/pip
14     - venv/
```

<https://docs.gitlab.com/ee/ci/caching/#caching-python-dependencies>

before_script

```
1 before_script:  
2   - python -V # Print out python version for debugging  
3   - pip install virtualenv  
4   - virtualenv venv  
5   - source venv/bin/activate
```

https://docs.gitlab.com/ee/ci/yaml/#before_script

Переменные окружения

The screenshot shows the GitLab interface for a project. The left sidebar contains navigation links: Project overview, Repository, Issues (0), Merge Requests (0), CI / CD, Security & Compliance, Operations, Packages & Registries, Analytics, Wiki, Snippets, and Members. The 'Settings' link is highlighted, with sub-links for General and Integrations. The main content area is titled 'Variables' and includes a 'Collapse' button. It explains that environment variables are applied via the Runner and can be used for passwords, secret keys, etc. It lists two types of variables: Protected (exposed to protected branches or tags) and Masked (hidden in job logs). A note states that environment variables are configured by the administrator to be protected by default. Below this is a table of variables:

Type	Key	Value	Protected	Masked	Environments	
Variable	HEROKU_STAGING_API_KEY	*****	✓	✗	All (default)	
Variable	HEROKU_STAGING_APP	*****	✓	✗	All (default)	
File	KAGGLE_JSON	*****	✓	✗	All (default)	
Variable	KAGGLE_KEY	*****	✓	✗	All (default)	
Variable	KAGGLE_USERNAME	*****	✓	✗	All (default)	

Below the table are buttons for 'Reveal values' and 'Add Variable'. The bottom section, 'Group variables (inherited)', explains that these variables are configured in the parent group settings and will be active in the current project in addition to the project variables.

Запуск пайплайна с помощью webhook

Production service to predict password complexity

- Project overview
- Repository
- Issues 0
- Merge Requests 0
- CI / CD
- Security & Compliance
- Operations
- Packages & Registries
- Analytics
- Wiki
- Snippets
- Members
- Settings**
 - General
 - Integrations

Pipeline triggers

Trigger a pipeline for a branch or tag by generating a trigger token and using it with an API call. The token impersonates a user's project access and permissions. [Learn more.](#)

Manage your project's triggers

Description

Trigger description

Add trigger

Token	Description	Owner	Last used
d04d992a19679ef2d486f4daadb2c0	trigger_by_a...		Never

In the following examples, you can see the exact API call you need to make in order to rebuild a specific `ref` (branch or tag) with a trigger token.

All you need to do is replace the `TOKEN` and `REF_NAME` with the trigger token and the branch or tag name respectively.

Use cURL

Copy one of the tokens above, set your branch or tag name, and that reference will be rebuilt.

```
curl -X POST \  
  -F token=TOKEN \  
  -F ref=REF_NAME \  
  https://gitlab.com/api/v4/projects/23343565/trigger/pipeline
```

Use .gitlab-ci.yml

In the `.gitlab-ci.yml` of another project, include the following snippet. The project will be rebuilt at the end of the

Merge requests

The screenshot shows a GitLab Merge Request (MR) interface. The top navigation bar is green with the GitLab logo and links to Projects, Groups, and More. A search bar is on the right. The left sidebar is light gray with a list of project items: Example of CI CD (selected), Project overview, Repository, Issues (0), Merge Requests (1), CI / CD, Security & Compliance, Operations, Packages & Registries, Analytics, Wiki, Snippets, Members, and Settings. The main content area has a green header with the project path 'Production ML course > Example of CI CD > Merge Requests > 11'. Below this, the MR is titled 'Update api version after pipeline run' and is 'Open' status, created 16 hours ago by Alexander Guschin (Owner). Action buttons include 'Edit', 'Mark as draft', and a dropdown. Tabs for Overview (0), Commits (7), Pipelines (6), and Changes (3) are shown. The MR details section includes: a request to merge 'feature-add-git-push' into 'master' with buttons for 'Open in Web IDE', 'Check out branch', and a dropdown; a green checkmark indicating 'Pipeline #247357068 passed for c97bb21e on feature-add-git-push'; an 'Approve' button with the note 'Approval is optional'; a green checkmark and a 'Merge' button with checkboxes for 'Delete source branch' and 'Squash commits'; a summary stating '1 commit and 1 merge commit will be added to master' with a link to 'Modify commit messages'; and a note 'You can merge this merge request manually using the command line'. Below this are thumbs up/down and a smiley face reaction bar. The activity feed shows: Alexander Guschin added 1 commit 13 hours ago (c99340fc - Update .gitlab-ci.yml) with a 'Compare with previous version' link; and Alexander Guschin changed the title from 'Update .gitlab-ci.yml' to 'Update api version after pipeline run' 13 hours ago. The right sidebar is light gray with a 'To Do' section, an 'Add a to do' button, and a list of assignees (Alexander Guschin), reviewers (0), milestones (None), time tracking (No estimate), labels (None), lock status (Unlocked), participants (2), and notifications (checked). A reference to the project and source branch is at the bottom.

GitLab Projects Groups More

Search or jump to...

Example of CI CD

Project overview

Repository

Issues 0

Merge Requests 1

CI / CD

Security & Compliance

Operations

Packages & Registries

Analytics

Wiki

Snippets

Members

Settings

Production ML course > Example of CI CD > Merge Requests > 11

Open Opened 16 hours ago by Alexander Guschin Owner

Edit Mark as draft

Update api version after pipeline run

Overview 0 Commits 7 Pipelines 6 Changes 3

Request to merge feature-add-git-push into master

Open in Web IDE Check out branch

Pipeline #247357068 passed for c97bb21e on feature-add-git-push

Approve Approval is optional

Merge Delete source branch Squash commits

1 commit and 1 merge commit will be added to master. Modify commit messages

You can merge this merge request manually using the command line

0 0

Oldest first Show all activity

Alexander Guschin @aguschin added 1 commit 13 hours ago

- c99340fc - Update .gitlab-ci.yml

Compare with previous version

Alexander Guschin @aguschin changed title from Update .gitlab-ci.yml to Update api version after pipeline run 13 hours ago

To Do Add a to do

Assignee Alexander Guschin @aguschin Edit

0 Reviewers Edit

Milestone None Edit

Time tracking No estimate or time spent

Labels None Edit


Lock merge request Unlocked Edit

2 participants

Notifications






Reference: production-ml/exa... Source branch: feature-add-...

Запуск по расписанию


 GitLab


Projects ▾ Groups ▾ More ▾


Search or jump to...


  5 ▾  4 ▾  2 ▾ 

E Example of CI CD

 Project overview

 Repository

 Issues 0

 Merge Requests 1


CI / CD


Pipelines


Editor


Jobs


Schedules


 Security & Compliance


 Operations


 Packages & Registries

 Analytics


 Wiki

 Snippets

 Members

 Collapse sidebar

Production ML course > Example of CI CD > Schedules







Scheduling Pipelines

The pipelines schedule runs pipelines in the future, repeatedly, for specific branches or tags. Those scheduled pipelines will inherit limited project access based on their associated user.

Learn more in the [pipeline schedules documentation](#).

All 1 Active 0 Inactive 1

New schedule

Description	Target	Last Pipeline	Next Run	Owner	
daily run	✓ master	✓ #246918723	Inactive	 Alexander Guschin	  

23

Local debug. Способ 1:

```
1 # Запускаем bash в контейнере локально
2 # и воспроизводим шаги из секции "script"
3 docker run -ti python:latest bash
4 # if you want to save changes in container,
5 # find your container id
6 docker ps --all
7 # and create new image
8 docker commit 21cc648e3f7b python:mychanges
9 # now you can run it with all changes already applied
10 docker run -ti python:mychanges
```


Local debug. Способ 2:

```
1 # 1. Устанавливаем gitlab-runner локально. Например, для MacOS:  
2 sudo curl --output /usr/local/bin/gitlab-runner "https://gitlab-runner-downloads.s3.amazonaws.com/v12.10.0/linux-amd64/gitlab-runner"  
3 sudo chmod +x /usr/local/bin/gitlab-runner  
4 # 2. Заходим в наш репозиторий  
5 cd passwords-complexity  
6 # 3. Запускаем джобу  
7 gitlab-runner exec docker my-job-name
```

Сам дебаг

```
1 # 1. Перед ошибкой добавляем в .gitlab-ci.yml "sleep 1000"  
2 vim .gitlab-ci.yml  
3 # 2. Запускаем джобу  
4 gitlab-runner exec docker my-job-name  
5 # 3. В соседнем терминале ищем нужный docker container  
6 docker ps  
7 # 4. Заходим в нужный контейнер и выполняем дебаг  
8 docker exec -ti $CONTAINER_ID bash
```

<https://www.lullabot.com/articles/debugging-jobs-gitlab-ci>

Local debug. Способ 2:

Нужно иметь в виду:

1. Перед запуском изменения нужно коммитить (пушить не обязательно)
2. Выполнение идет локально, поэтому `variables` из репозитория недоступны
3. Можно зарегистрировать gitlab-runner и запускать на нем CI/CD с gitlab.com
4. Раннеры отличаются друг от друга

<https://docs.gitlab.com/runner/install/>

Саммари

1. Continuous Integration включает в себя этапы тестирования и сборки приложения.
2. Continuous Delivery (Deployment) включает в себя этапы настройки окружений и деплоя в них.
3. Цель CI/CD - сократить разрыв между разработкой приложений и управлением их работой с помощью автоматизации процессов.
4. Не так важно, какой инструмент выбрать для CI/CD, главное - использовать его.
5. Для одного и того же проекта можно собирать разные пайплайны CI/CD – выбирайте наиболее простой вариант.
6. Используйте локальный дебаг

<https://sean-bradley.medium.com/auto-devops-with-gitlab-ci-and-docker-compose-f931233f080f>

A walkthrough of how to deploy your app from GitLab using Docker Compose

<https://medium.com/@vitalypanukhin/docker-compose-and-gitlab-b209d09210f6>

Simple continuous deployment with docker compose, docker machine and Gitlab CI

<https://medium.com/@Empanado/simple-continuous-deployment-with-docker-compose-docker-machine-and-gitlab-ci-9047765322e1>

A tool for local debug of Github CI <https://github.com/nektos/act>

Семинар

1. Смотрим на все этапы на графе выполнения пайплайна - проговариваем какие этапы есть в принципе. Смотрим на CI/CD файл в репозитории, обсуждаем первые пару джобов про тестирование и билд докер-образа
2. Запускаем первые джобы локально, показываем, как делать локальный дебаг
3. Обсуждаем следующие джобы про train model и update-if-qualified. Обсуждаем реализацию обновления модели с пушом в репозиторий.
4. Обсуждаем запуск следующей части пайплайна и джобы для передеплоя.
5. Обсуждаем варианты, как можно было бы сделать пайплайн иначе.
6. Обсуждаем типичные ошибки, которые были при пайплайне, мотивируем использовать локальный дебаг.
7. Обсуждаем что потребуется изменить в пайплайне для шляпы.