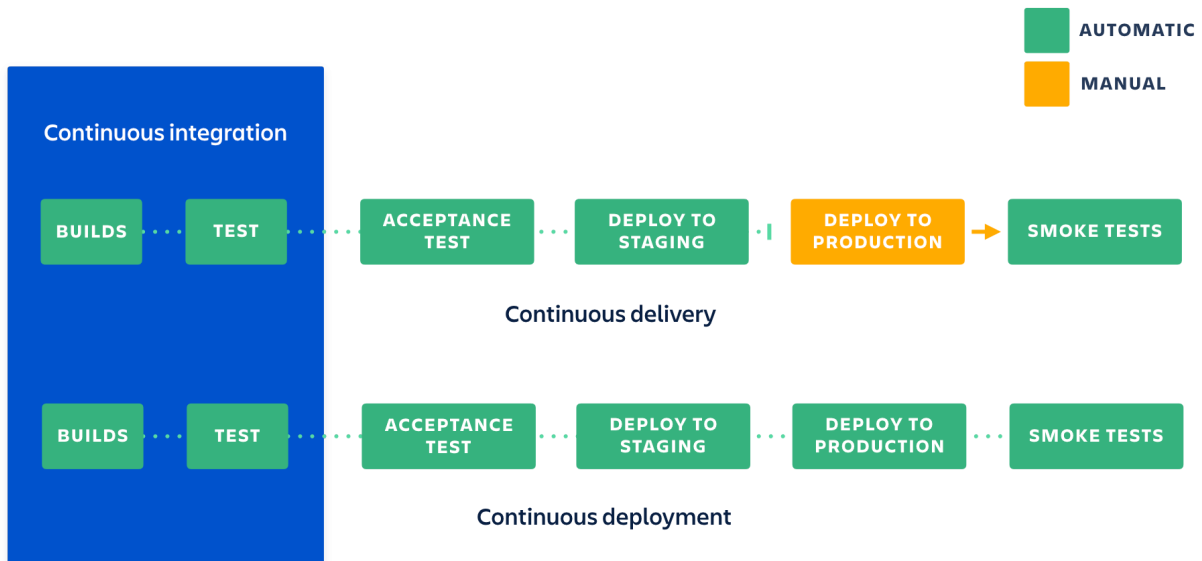


CI/CD

Continuous Delivery и Continuous Development





На этом уроке

1. Более подробно рассмотрим Continuous Delivery и Continuous Deployment.
2. Узнаем, какие существуют виды deploy.
3. Узнаем, что такое environments и для чего они нужны.
4. Узнаем, что такое deployments и для чего они нужны.
5. Узнаем, как откатывать код на предыдущую версию.
6. Выясним, как сделать deployment'ы максимально безопасными.

Оглавление

[Виды deploy](#)

[Recreate](#)

[Плюсы](#)

[Минусы](#)

[Rolling / Ramped](#)

[Плюсы](#)

[Минусы](#)

[Blue/green deployment или Red/black deployment](#)

[Плюсы](#)

[Минусы](#)

[Canary deployment](#)

[Плюсы](#)

[Минусы](#)

[Environments](#)

[Deployments](#)

[Manual deployments](#)

[Rollback](#)

[Безопасность deployment'ов](#)

[Разграничение прав в environments](#)

[Одновременное выполнение одной deployment job](#)

[Устаревшие deployment job'ы](#)

[Deploy Freeze](#)

[Практическое задание](#)

[Глоссарий](#)

[Дополнительные материалы](#)

[Используемые источники](#)

На этом занятии мы более подробно рассмотрим основные инструменты Continuous Delivery и Continuous Deployment.

Виды deploy

Есть несколько способов разворачивать программное обеспечение в боевой среде. Каждый из них имеет свои достоинства и недостатки.

Recreate

Recreate — это повторное создание, самый простой вид деплоя. Основной принцип: сначала тушится текущая версия программного обеспечения, потом выкладывается и поднимается новая.

Плюсы

Простота реализации.

Минусы

Время простоя. В промежутке, когда текущая версия уже потушена, но новая ещё не поднята, сервис не работает.

Из-за того, что этот подход требует полной остановки сервиса, он практически не используется в боевой (production) среде.

Rolling / Ramped

Rolling — также встречается название Ramped — постепенно накатываемый деплоймент. Основной принцип — постепенное накатывание новой версии ПО вместо старой на один за другим инстансы.

Плюсы

Нет простоя.

Минусы

Изменение общих ресурсов. Например, для работы новой версии ПО нужно перестроить структуру базы данных или переименовать таблицу. За одну итерацию этого не сделать, иначе получим ошибки при работе старого кода с новой базой данных.

В этих случаях изменения в БД разбиваются на несколько итераций. Например, нужно переименовать колонку в БД. В первой итерации мы просто добавляем ещё одну колонку с нужным нам названием. Таким образом, для старого кода ничего не поменяется, он просто не будет использовать эту колонку, а хосты с новым кодом уже будут использовать новую колонку. Второй итерацией удаляем старую колонку.

Blue/green deployment или Red/black deployment

Blue/green deploy — также встречается название Red/black deploy — сине-зелёное развертывание. Основной принцип: одновременное развёртывание старой (зелёной) и новой (синей) версий ПО. Новая версия доступна для QA, и пользователи работают со старой версией. После того как новая (синяя) версия успешно протестирована, вся боевая нагрузка (пользователи) переключается на новую версию.

Плюсы

1. Возможность протестировать продукт непосредственно в боевой среде.
2. Возможность быстрого переключения со старой версии на новую, и наоборот.

Минусы

Необходимо в два раза больше ресурсов, чтобы иметь возможность одновременно развернуть и старую, и новую версию кода.

Canary deployment

Canary deployment — канареечное развёртывание. Принцип похож на Blue/green deployment, но здесь используется поэтапный переход на новую версию. Создаём два практически одинаковых сервера: один обслуживает почти всех пользователей, а другой, с новыми функциями, обслуживает лишь небольшую подгруппу пользователей. Результаты их работы сравниваются. Если всё проходит без ошибок, новая версия постепенно выкатывается на все хосты.

Плюсы

Постепенное накатывание новой версии. Это значит, что в случае проблем пострадает небольшое количество пользователей.

Минусы

Более медленный переход на новую версию, в отличие от Blue/green.

Environments

Как вы помните из предыдущих занятий, прежде чем попасть в production, код должен пройти через несколько шагов (stage). Например, это могут быть:

- разработка и сборка кода (build);
- тестирование кода (test);
- деплой кода в тестовую или стейджинг (staging) среду.

Это помогает отловить ошибки на более ранних этапах. Кроме тестирования (test) и сборки (build), которые мы разобрали ранее, GitLab CI/CD также предоставляет возможность деплоить (deploy) код в созданную инфраструктуру с возможностью отследить каждый deploy.

Environment — это окружение. Environment'ы помогают контролировать деплой программного обеспечения. В environments описывается, куда код должен быть задеплоен, например в staging или в production.

В [списке предустановленных переменных](#) уже есть несколько переменных, касающихся окружения (environment). Рассмотрим некоторые из них.

`CI_ENVIRONMENT_NAME` — имя окружения для текущей job. Существует, только если задан параметр `environment:name`.

`CI_ENVIRONMENT_SLUG` — упрощённая версия имени окружения, которая подходит для использования в DNS, URLs, Kubernetes и т. п. Например, в этой версии прямой слеш заменяется на дефис. Существует, только если задан параметр `environment:name`.

```
$ echo $CI_ENVIRONMENT_NAME
test/master
$ echo $CI_ENVIRONMENT_SLUG
test-master-cjwnyy
```

`CI_ENVIRONMENT_URL` — URL окружения для текущей job. Существует, только если задан параметр `environment:url`.

Чтобы задать окружение (environment), необходимо указать параметр `name` — имя окружения. Например, `name: staging`. Если его не указать, ошибок не будет, job успешно отработает, но в Operations → Environments будет пусто.

Пример создания environment:

```
deploy on staging:
  stage: deploy
  environment:
    name: staging
  script:
    - echo "${CI_COMMIT_REF_NAME} was deployed on $CI_ENVIRONMENT_NAME"
  only:
    - master
```





Environment имеет имя, которое было задано в `gitlab_ci.yml` в `environment:name`.

Job указывает имя job, в которой был создан environment.

Commit указывает, какая ветка и какой коммит сейчас находятся в staging'e.

Job `deploy on staging` запускается только в `master` ветке. Это значит, что все изменения в ветках не будут задеплоены на staging.

Когда merge request будет вмержен в мастер, все job'ы в пайплайне запустятся. В том числе `deploy on staging` задеплоит код в staging, и в Operations → Environments появится environment staging.

Environment	Deployment	Job	Commit	Updated	Auto stop in
staging	#22 by 	deploy on staging #7...	 master → aa36630f add staging	59 minutes a...	 

Если в pipeline в разных job'ах сделать одинаковые имена environment'ам, например у двух разных job проставить `environment:name:prod`, то при выполнении pipeline в продакшн задеплоится результат выполнения обеих job. Это произойдёт последовательно, в зависимости от того, в каком порядке шли job'ы в самом pipeline.

```
deploy on production:
  stage: deploy
  environment:
    name: prod
  script:
    - echo "${CI_COMMIT_REF_NAME} was deployed on production"
  only:
    - master

pages:
  stage: pages
  environment:
    name: prod
  script:
    - echo 'dream house'
  artifacts:
    name: "${CI_JOB_NAME}"
    paths:
      - public
    exclude:
      - public/style.css
  only:
    - master
```

 success	#20		Y master → dd00ffcf Update .gitlab-ci.yml	pages (#723...	22 hours ago	22 hours ago	
 success	#19		Y master → dd00ffcf Update .gitlab-ci.yml	deploy on pro...	22 hours ago	22 hours ago	

Как видно на скриншоте, у нас в обоих случаях один коммит, то есть одно изменение, — dd00ffcf, а environment обновился два раза. Первый раз в job'е deploy on production, и второй раз в job'е pages.

В нашем примере эти job'ы не пересекаются, но в целом такое поведение может приводить к путанице и неочевидному поведению, например, если в обеих job'ах изменяются общие данные. Поэтому стоит избегать подобного поведения. В идеале в pipeline должно быть по одной job для каждого environment.

Deployments

Deployment — задеплоенное изменение кода, релиз.

Deployment'ы создаются, когда job деплоит код в environment. Таким образом, каждый environment может иметь один и более deployment'ов.

GitLab хранит в себе всю историю deployment'ов для каждого environment'a. Посмотреть историю можно, нажав на интересующий environment в Operations → Environments.

staging							Monitoring	Edit	Stop
Status	ID	Triggerer	Commit	Job	Created	Deployed			
success	#25		Y master ↔ b0693686 Update .gitlab-ci.yml	deploy on sta...	4 minutes ago	2 minutes ago			
success	#22		Y master ↔ aa36630f add staging	deploy on sta...	22 hours ago	22 hours ago			

Manual deployments

В предыдущем примере мы рассмотрели ситуацию, когда у нас появился второй environment prod.

Если мы посмотрим pipeline и environment'ы, то увидим, что код **одновременно** раскатился и в staging, и в prod.

Pipeline DAG Jobs 5 Tests 0

Build

Test

Deploy

Pages

✓ docker build th...

✓ test docker

✓ deploy on prod...

✓ deploy on staging

✓ pages:deploy

✓ pages

Environment	Deployment	Job	Commit	Updated	Auto stop in
prod	#29 by	deploy on production...	Y master ↔ ad0012ab add deploy on production	1 minute ago	
staging	#28 by	deploy on staging #7...	Y master ↔ ad0012ab add deploy on production	1 minute ago	

В такой ситуации смысл staging'a, как среды, в которой можно ещё раз проверить и протестировать код, теряется. Одновременно с тем, как код попадает на staging, этот же код попадает в prod без каких-либо промежуточных тестов и проверок.

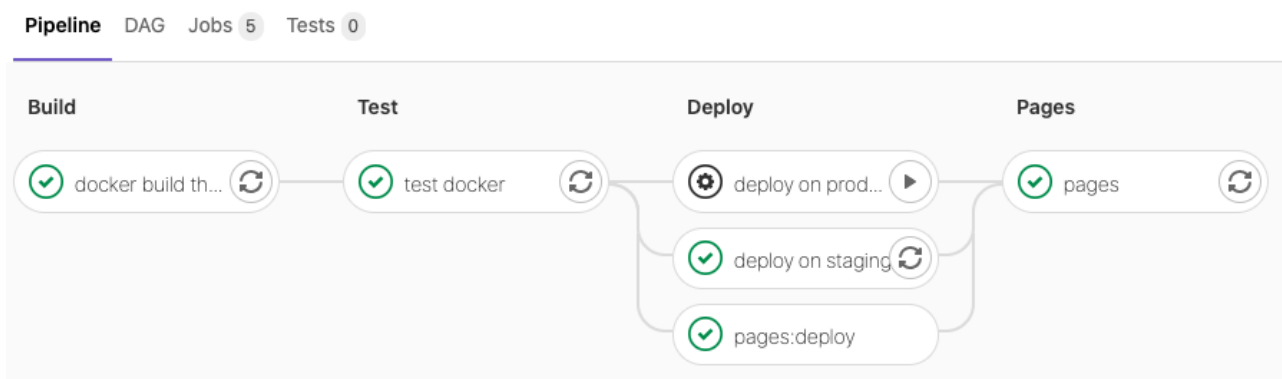
Чтобы такого не происходило и у нас была возможность ручного контроля того, что попадает в prod среду, нужно добавить параметр `when: manual`.

`when: manual` указывает, что требуется дополнительное ручное действие.

Как это выглядит на практике: мы добавляем `when: manual` в environment prod.

```
deploy on production:
  stage: deploy
  environment:
    name: prod
  when: manual
  script:
    - echo "${CI_COMMIT_REF_NAME} was deployed on $CI_ENVIRONMENT_NAME"
  only:
    - master
```

И видим в pipeline, что job deploy on production не стала выполняться автоматически:



И в environment обновился только staging:

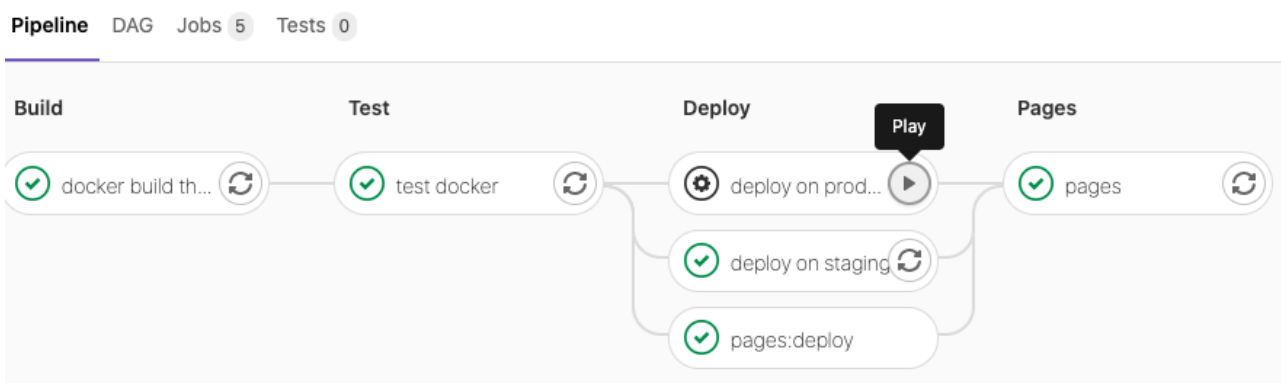
Environment	Deployment	Job	Commit	Updated	Auto stop in
staging	#31 by	deploy on staging #7...	master -> 3a2eac67 add when: manual to deploy ...	1 minute ago	
prod	#29 by	deploy on production...	master -> ad0012ab add deploy on production	26 minutes a...	

Запустить job `deploy to production`, которая была описана с параметром `when: manual`, то есть выложить код в `environment prod`, можно пятью способами:

1. В списке пайплайнов (CI/CD → Pipelines) у интересующего пайплайна выбрать `deploy on production`, потому что так называется наша job с ручным запуском:

Status	Pipeline	Triggerer	Commit	Stages	
passed	#187016388 latest		master → 3a2eac67 add when: manual to deploy ...	✓✓✓✓	00:03:42 8 minutes ago Run manual or delayed jobs ▶ ⌵ ⬇
passed	#187011556		master → ad0012ab add deploy on production	✓✓✓✓	42 minutes ago ▶ ⌵ ⬇
passed	#186996337		master → b0693686 Update .gitlab-ci.yml	✓✓✓✓	00:03:08 2 hours ago ▶ ⌵ ⬇

2. В конкретном пайплайне нажать на Play в нужной job:



3. В `environments` выбрать `deploy on production`:

Environment	Deployment	Job	Commit	Updated	Auto stop in
staging	#31 by	deploy on staging #7...	master → 3a2eac67 add when: manual to deploy ...	8 minutes ago	
prod	#29 by	deploy on production...	master → ad0012ab add deploy on production	33 minutes a...	

Deploy to...
▶ ⌵ ⬇
deploy on production
▶ ⌵ ⬇

4. В `deployment's` выбрать `deploy on production` у интересующего `deployment'a`. Список `deployment'ов` можно увидеть, зайдя в конкретный `environment`, например в `staging`:

staging Monitoring Edit Stop

Status	ID	Triggerer	Commit	Job	Created	Deployed	
success	#31		Y master -> 3a2eac67 add when: manual to deploy on ...	deploy on sta...	15 minutes ago	13 minutes ago	Deploy to... ▶ ↺
success	#28		Y master -> ad0012ab add deploy on production	deploy on sta...	40 minutes a...	38 minutes a...	↺
success	#25		Y master -> b0693686 Update .gitlab-ci.yml	deploy on sta...	2 hours ago	2 hours ago	↺
success	#22		Y master -> aa36630f add staging	deploy on sta...	1 day ago	1 day ago	↺

5. В списке job (CI/CD → Jobs) нажать Play у интересующей job:

passed	#725407027 Y master -> 3a2eac67	#187016388 by	deploy	deploy on production	00:00:42 11 minutes ago	▶
---------------------	---------------------------------	---------------	--------	----------------------	----------------------------	----------------

Во всех случаях будет одинаковый результат.

Rollback

Чтобы откатиться на предыдущую версию кода или на другой коммит, достаточно в Operations → Environments перейти в историю интересующего environment'a и выбрать, на какую версию откатиться.

prod Monitoring Edit Stop

Status	ID	Triggerer	Commit	Job	Created	Deployed	
success	#32		Y master -> 3a2eac67 add when: manual to deploy on ...	deploy on pro...	11 hours ago	11 hours ago	↺
success	#29		Y master -> ad0012ab add deploy on production	deploy on pro...	12 hours ago	11 hours ago	Rollback environment ↺

Запустится pipeline, который накатит выбранный коммит:

prod

Monitoring

Edit

Stop

Status	ID	Triggerer	Commit	Job	Created	Deployed	
success	#33		Y master → ad0012ab add deploy on production	deploy on pro...	11 minutes ago	10 minutes ago	↺
success	#32		Y master → 3a2eac67 add when: manual to deploy on ...	deploy on pro...	11 hours ago	11 hours ago	↺
success	#29		Y master → ad0012ab add deploy on production	deploy on pro...	12 hours ago	12 hours ago	↺

Безопасность deployment'ов

Job'ы деплоя наиболее уязвимы по сравнению с остальными job'ами пайплайна, так как любая ошибка может попасть в prod и отразиться на пользователях.

По этой причине в GitLab предусмотрено несколько фичей, которые могут обеспечить стабильность и безопасность deployment'ов.

Разграничение прав в environments

По дефолту изменять environment может любой человек с правами Developer и выше. Это не очень хорошо по отношению к environment'у prod, так как любой разработчик может разложить свой код на prod, минуя staging.

Чтобы защитить environment, — в нашем случае это будет prod — мы можем настроить доступ к нему. Для этого в Settings → CI/CD нужно перейти в Protected environments, выбрать, какое окружение мы хотим защитить, и проставить, кто может деплоить в это окружение.

Например, что только Maintainer'ы могут деплоить в prod:

Protect an environment

Environment

prod

Allowed to deploy

Maintainers

Protect

Одновременное выполнение одной deployment job

В GitLab job'ы в пайплайнах запускаются параллельно. Поэтому возможна ситуация, когда две разные job'ы пайплайнов деплоятся в одно и то же окружение в одно и то же время. Это не совсем ожидаемое поведение: deployment'ы должны запускаться последовательно, друг за другом.

В GitLab можно настроить, чтобы только одна deployment job выполнялась в один момент времени, используя параметр `resource_group`.

Пример проблемного pipeline без использования `resource_group`:

1. Deploy job в Pipeline A запускается.
2. Deploy job в Pipeline B запускается. Одновременно два deployment'а запущены, что может привести к неожиданному результату.
3. Deploy job в Pipeline A завершает работу.
4. Deploy job в Pipeline B завершает работу.

Что происходит после добавления `resource_group`:

1. Deploy job в Pipeline A запускается.
2. Deploy job в Pipeline B пытается запуститься, но ждёт, пока job в Pipeline A завершится.
3. Deploy job в Pipeline A завершает работу.
4. Deploy job в Pipeline B запускается.
5. Deploy job в Pipeline B завершает работу.

Устаревшие deployment job'ы

Поскольку job'ы пайплайнов запускаются одновременно, может возникнуть такая ситуация, при которой job deployment'а из более свежего пайплайна запускается **до** job deployment'а из более старого пайплайна.

Таким образом в prod может получиться следующая ситуация:

1. Запускается Pipeline A.
2. Запускается Pipeline B.
3. Job деплоя в Pipeline B завершается раньше, и на prod деплоится самый свежий код.
4. Job деплоя в Pipeline A завершается позже, и в prod уезжает более старая версия, перезаписывая новый deployment из Pipeline B.

Чтобы такого не происходило, необходимо в Settings → CI/CD → General pipeline поставить галочку в Skip outdated deployment jobs. Тогда будет происходить следующее:

1. Запускается Pipeline A.
2. Запускается Pipeline B.

3. Job деплоя в Pipeline B завершается раньше, и на prod деплоится самый свежий код.
4. Job деплоя в Pipeline A автоматически отменяется, соответственно prod **не будет перезаписан старой версией**.

Deploy Freeze

Можно запрещать деплои на какое-то время, установив Settings → CI/CD → Deploy Freeze.

Add deploy freeze



Define a custom deploy freeze pattern with [cron syntax](#)

Freeze start

Freeze end

Cron time zone



Cancel

Add deploy freeze

Также необходимо в саму job добавить `rules: if: $CI_DEPLOY_FREEZE == null`.

```
deploy on prod:
  stage: deploy
  environmen:
    name: prod
  script:
    - echo "${CI_COMMIT_REF_NAME} was deployed on $CI_ENVIRONMENT_NAME"
  rules:
    - if: $CI_DEPLOY_FREEZE == null
  resource_group: prod
```

Внимание! У этого метода есть несколько нюансов:

- 1) С rules нельзя использовать only и when. То есть мы не можем оставить `only:master` и `when:manual`, чтобы job запускалась только в master ветке и в ручном режиме.
- 2) Если не указать rules в gtilab_ci.yml, но создать Deploy freeze, то ничего не произойдёт. Job будет запускаться, как если бы Deploy Freeze вообще не был бы установлен.

Практическое задание

Будет прикреплено к лекции отдельно.

Глоссарий

Environment — окружение. Помогает контролировать деплой программного обеспечения. В environments описывается, куда код должен быть задеплоен, например в staging или в production.

Deployment — задеплоенное изменение кода, релиз. В GitLab'е environment'ы состоят из deployment'ов.

Дополнительные материалы

1. [Официальная документация по GitLab CI.](#)
2. [CI/CD pipelines](#)

Используемые источники

1. Бетси Бейер, Крис Джоунс, Дженнифер Петофф, Нейл Ричард Мёрфи «Site Reliability Engineering. Надёжность и безотказность, как в Google», 2018.
2. [Официальная документация по GitLab CI.](#)
3. [Стратегии деплоя в Kubernetes: rolling, recreate, blue/green, canary, dark \(A/B-тестирование\).](#)
4. [Six Strategies for Application Deployment.](#)
5. [Environments and deployments.](#)
6. [GitLab CI/CD environment variables.](#)
7. [Protected Environments.](#)
8. [Deployment safety.](#)