

Введение в DevOps: Gitlab



Дмитрий
Томин



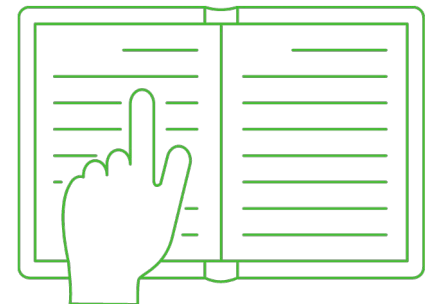
Дмитрий Томин
Ведущий сетевой инженер

Предисловие

На этом занятии мы поговорим о:

- Gitlab,
- Gitlab CI.

По итогу занятия вы узнаете, какие инструменты предоставляет Gitlab для разработчиков и DevOps-инженеров.



План занятия

1. [Gitlab](#)
2. [Gitlab CI](#)
3. [Dind \(Docker in docker\)](#)
4. [Настройка пайплайна](#)
5. [Sonarqube](#)
6. [Итоги](#)
7. [Домашнее задание](#)



Gitlab: обзор



Gitlab — веб-инструмент жизненного цикла DevOps с открытым исходным кодом, представляющий систему управления репозиториями кода для Git с собственной вики-системой отслеживания ошибок, CI/CD пайплайном и другими функциями.

GitLab имеет вариант установки на свои сервера или использование в облаке. Он делится на бесплатную CE версию и EE энтерпрайз.

Gitlab: обзор

Функционал Gitlab не заканчивается просто хранением репозиториев. В число инструментов можно включить:

- CI/CD
- Wiki
- Issues
- Snippets
- Pages
- Группы, права доступа
- ...



Gitlab CI

Gitlab CI

Настройки CI/CD хранятся вместе с репозиторием в файле **.gitlab-ci.yml**.

После коммита в репозиторий этот файл читается, и запускается пайплайн.

Полный синтаксис описан в [документации](#).

Запуск процессов может выполняться на специальных агентах — **раннерах**, которые могут быть расположены в облаке Gitlab или на своих серверах.

Gitlab CI: установка раннера

Раннер может быть расположен на любом сервере, имеющем доступ в интернет (если используется собственный сервер gitlab, то необходимо обеспечить доступ к нему).

В настройках проекта или группы необходимо получить токен [для регистрации раннера](#).

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/`

And this registration token:

`DSC_Zaskdjkxasdasd`

Gitlab CI: установка раннера

После получения токена необходимо установить раннер в любом виде на сервер: <https://docs.gitlab.com/runner/install/>

Перед запуском раннер необходимо зарегистрировать:

```
docker run -rm -v /srv/gitlab-runner/config:/etc/gitlab-runner -v  
/var/run/docker.sock:/var/run/docker.sock gitlab/gitlab-runner:latest register
```

После регистрации будет сгенерирован конфиг [/etc/gitlab-runner/config.toml](#) и раннер можно запускать:

```
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
```

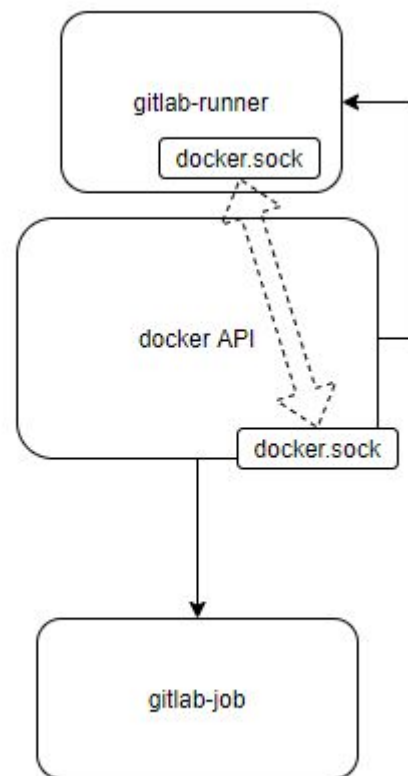
Gitlab CI: настройка раннера

Раннер может быть запущен в нескольких режимах. Самые популярные: **shell** и **docker**.

Запуск в режиме **shell** позволит вам выполнять любые команды просто из shell.

В примере выше мы запустили раннер внутри контейнера, предусмотрительно добавив в этот контейнер сокет docker.

Теперь мы можем запустить раннер в режиме **docker**.



Gitlab CI: пример написания .gitlab-ci.yml

Stages описывает этапы пайплайна и порядок их запуска.

Эти этапы могут не входить в итоговый пайплайн.

При описании этапа мы указываем docker image, в котором будем запускать команды из раздела script.

```
stages:  
  - build  
  - test  
  
test:  
  stage: test  
  image: golang:latest  
  script:  
    - echo "Starting tests"
```



Dind (Docker in docker)

Dind

Для доступа к docker изнутри пайплайна можно применять несколько подходов:

- запуск docker демона внутри контейнера:
 - + не надо дополнительно настраивать раннеры,
 - отсутствие docker кэша, относительно увеличенное время запуска, так как необходимо будет этот демон запустить;
- доступ к docker демону хостовой машины:
 - требуется дополнительная настройка раннеров,
 - + отсутствуют минусы первого подхода.

Настройка раннеров

Для настройки раннера и проброса докер сокета необходимо добавить в конфигурацию раннера строку для подключения в контейнеры пайплайна дополнительного volume:

```
[runners.docker]  
  volumes = ["/var/run/docker.sock:/var/run/docker.sock"]
```

Также если необходимо дополнительно настроить различные параметры, их полный список можно найти [здесь](#).



Настройка пайплайна

Настройка пайплайна

Для начала определимся с этапами, в нашем примере сделаем тестирование, сборку.

Клонируем репозиторий

<https://github.com/killmeplz/netology-test> и перемещаем его на гитлаб:

```
git remote rename origin old-origin  
git remote add origin  
https://gitlab.com/killmeplz685/netology-test.git  
git push -u origin --all  
git push -u origin --tags
```

Далее начинаем настраивать **.gitlab-ci.yml**.

Настройка пайплайна

Создадим CI согласно тем этапам, которые оговорили.

```
stages:  
  - test  
  - build  
  
test:  
  stage: test  
  image: golang:latest  
  script:  
    - go test .  
  
build:  
  stage: build  
  image: docker:latest  
  script:  
    - docker build .
```

Настройка пайплайна

Далее сделаем сборку только в мастер ветке, чтобы не собирать каждый коммит и экономить место.

```
stages:
  - test
  - build

test:
  stage: test
  image: golang:latest
  script:
    - go test .

build:
  stage: build
  only:
    - master
  image: docker:latest
  script:
    - docker build .
```

Настройка пайплайна

Однако разработчики хотят иметь возможность собирать иногда промежуточные версии для их тестирования вручную.

Давайте добавим ручную сборку контейнера для всех веток, кроме мастера.

```
stages:
  - test
  - build

test:
  stage: test
  image: golang:latest
  script:
    - go test .

build_manual:
  stage: build
  except:
    - master
  image: docker:latest
  script:
    - docker build .

build:
  stage: build
  only:
    - master
  image: docker:latest
  script:
    - docker build .
```



Sonarqube



Sonarqube

SonarQube — платформа с открытым исходным кодом для непрерывного анализа и измерения качества программного кода. Поддерживает анализ кода и поиск ошибок согласно правилам стандартов программирования.

SonarScanner — компонент, который непосредственно занимается сканированием кода, может быть запущен в качестве контейнера.

Давайте добавим его в наш пайплайн.

Sonarqube

Добавим проверку sonar-scanner на этап test. Это позволит ему запускаться параллельно с тестами, и не тратить дополнительное время на ожидание.

```
sonarqube-check:  
  stage: test  
  image:  
    name:  
    sonarsource/sonar-scanner-cli  
  entrypoint: [""]  
  variables:  
  script:  
    - sonar-scanner
```

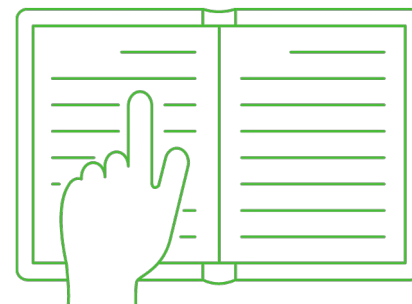



Итоги

Итоги

Сегодня мы:

- рассмотрели основы Gitlab CI;
- научились запускать раннеры и составлять pipeline.





Домашнее задание

Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера .
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

**Задавайте вопросы и
пишите отзыв о лекции!**

Дмитрий Томин