

# GitLab



Алексей  
Метляков



**Алексей Метляков**

DevOps Engineer

OpenWay



Алексей Метляков

---

# План занятия

1. [GitLab](#)
2. [Own Issues](#)
3. [Own Wiki](#)
4. [Own Repository](#)
5. [Own CI\CD](#)
6. [Own Packages](#)
7. [Итоги](#)
8. [Домашнее задание](#)



# Что такое GitLab?

**GitLab** - ППО для **полного контроля** над производственным циклом ПО. Активно использует **Ruby** и **Redis**, есть возможность использовать **SaaS** (Software as a Service), так и установить на **своей** инфраструктуре.

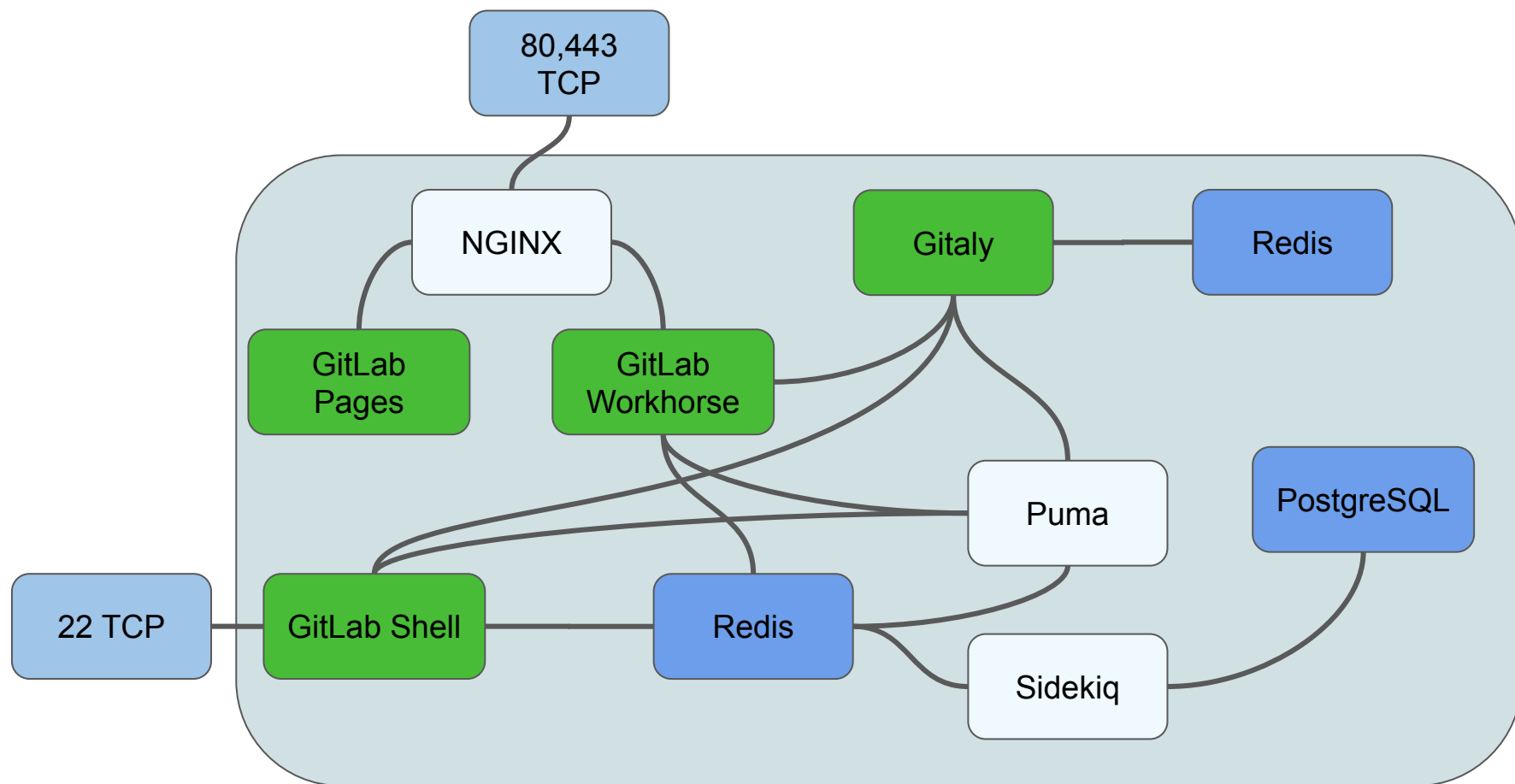
- Существует в следующих исполнениях:
  - **Community Edition (CE)** - бесплатная версия с функционалом, не требующим проприетарного кода
  - **Enterprise Edition (EE)** - полноценная версия GitLab  
Полное описание различий CE и EE на [сайте](#)
- Использует **YAML** для описание пайплайнов

# Что такое GitLab?

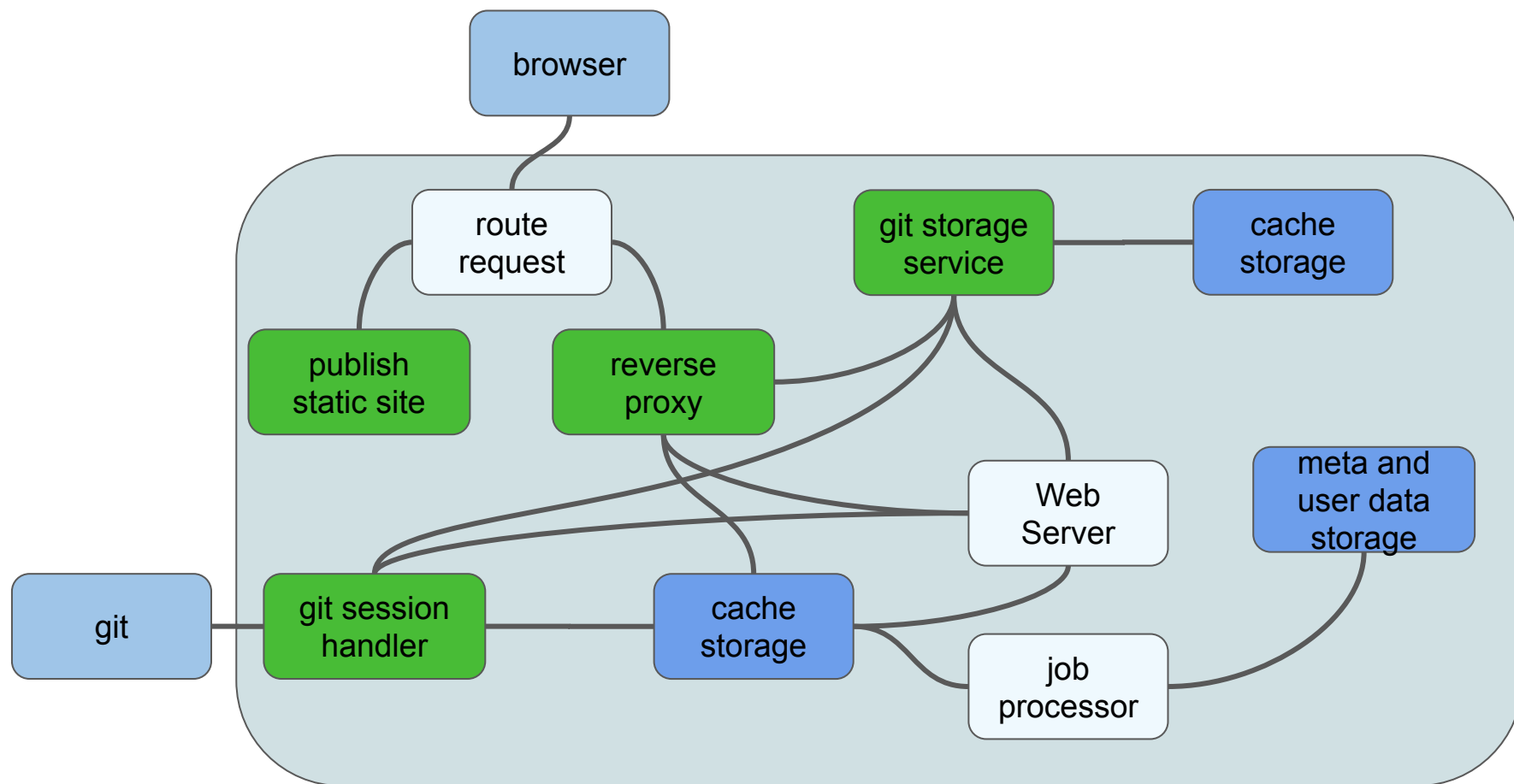
Некоторые основные возможности системы:

- Хранение **репозитория** (git)
- **Отслеживание** задач (issue tracking)
- Хранение **страниц** описания продукта (wiki, static site)
- Хранение **артефактов** (repository, registry)
- Создание **CI\CD** конвейера

# Архитектура GitLab



# Архитектура GitLab



Полное описание архитектуры есть [на официальном сайте](#)





# Компоненты Gitlab

# Own Issues

**Issue Tracking** - отдельная компонента GitLab, позволяющая создавать issue или incident для конкретного проекта.

Позволяет:

- Выставлять **label**
- Организовать **service desk**
- Группировать через **milestone**
- Связывать **issue** между собой



## Own Wiki

**Wiki** - компонент, позволяющий оформить странички с произвольным содержанием и привязать их к проекту, аналог **Confluence** от Atlassian

# Own Repository

**Gitaly - gRPC (Remote Procedure Call)** система, которая обрабатывает вызовы (TCP / Socket) процедур обработки информации и транслирует их на файловую систему напрямую (local / NFS) или через gRPC (TCP / Socket)

Также **GitLab** имеет собственную встроенную IDE для работы со всеми файлами репозитория

---

# Own CI\CD

**GitLab CI\CD** - компонента, позволяющая строить **CI**, **CDL** и **CDP** процессы над конкретным репозиторием.

Некоторые особенности системы:

- Пайплайн пишется на **YAML**-синтаксисе
- Пайплайн хранится в репозитории в файле **.gitlab-ci.yml**
- Есть возможность использовать **AutoDevOps**
- Есть отдельные возможности миграции с **Jenkins** и **CircleCI**
- Позволяет использовать **ChatOps**

# Own CI\CD

**Pipeline** описывается в файле **.gitlab-ci.yml** и имеет следующую структуру:

```
stages:
  - build
  - test
  - deploy
image: alpine
our_builder:
  stage: build
  script:
    - echo "Some build everything"
our_tester:
  stage: test
  script:
    - echo "We can test everything"
our_deployer:
  stage: deploy
  script:
    - echo "We can deploy sometimes"
    - echo "And some more action too"
```

# Own CI\CD

При этом, можно использовать триггеры разных видов, для разных конвейерных лент:

```
stages:
  - triggers

triggers_a:
  stage: triggers
  trigger:
    include: a/.gitlab-ci.yml
  rules:
    - changes:
      - a/*
triggers_b:
  stage: triggers
  trigger:
    include: b/.gitlab-ci.yml
  rules:
    - changes:
      - b/*
```

# Own CI\CD

Структура файлов с конвейерами ниже:

a/.gitlab-ci.yml

```
stages:
  - build
  - test
  - deploy
image: alpine
our_builder_a:
  stage: build
  script:
    - echo "Some build everything"
our_tester_a:
  stage: test
  needs: [our_builder_a]
  script:
    - echo "We can test everything"
our_deployer_a:
  stage: deploy
  needs: [our_tester_a]
  script:
    - echo "We can deploy sometimes"
    - echo "And some more action too"
```

b/.gitlab-ci.yml

```
stages:
  - build
  - test
  - deploy
image: alpine
our_builder_b:
  stage: build
  script:
    - echo "Some build everything else"
our_tester_b:
  stage: test
  needs: [our_builder_b]
  script:
    - echo "We can test everything else"
our_deployer_a:
  stage: deploy
  needs: [our_tester_b]
  script:
    - echo "We deploy sometimes else"
    - echo "And some else action too"
```



# Own CI\CD

Основные ключевые слова, необходимые при проектировании конвейеров:

after_script	allow_failure	artifacts	before_script	coverage
dependencies	environment	except	extends	image
include	inherit	interruptible	needs	only
pages	parallel	release	resource_group	retry
rules	script	secrets	services	stage
tags	timeout	trigger	variables	when

Полный перечень ключевых слов и их параметров доступен в [официальной документации](#)

---

# Own Packages

**Packages & Registries** - компонента хранения артефактов сборки и образов. Подсистема **packages** поддерживает хранение:

- Composer
- Conan
- Go
- Maven
- npm
- Nuget
- PyPI
- Generic Packages

Подсистема **registries** хранит набор **docker-образов**



# Итоги

---

# Итоги

- **GitLab** - полноценный комплексный продукт управления вашим проектом
- Позволяет **хранить** исходный код
- **Контролировать** жизненный цикл
- **Строить** конвейер и **управлять** им
- **Создавать** странички с документацией
- **Сохранять** и **использовать** артефакты сборки
- **Изменять** файлы в репозитории, **создавать** новые, при помощи встроенной **IDE**
- И **всё** это внутри **единого** web-интерфейса

---

# Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

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

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

**Алексей Метляков**