
DVPS
DEVELOPMENT AND OPERATIONS

Устройство Gitlab CI. Построение процесса непрерывной поставки

План

- Непрерывная поставка ПО
- Gitlab CI. Функционал и концепции
- Непрерывная поставка с Gitlab CI

Непрерывная поставка ПО

Процесс разработки и эксплуатации ПО, при котором каждое изменение может быть выкачено в боевое окружение

Непрерывная поставка ПО

- Ускоряет и упрощает выпуск продукта
- Повышает качество продукта
- Сокращает время поиска и устранения дефектов
- Снижают стресс от релизов
- Повышают прозрачность процессов
- Обеспечивают командное взаимодействие

Непрерывная поставка ПО

Меняется взаимодействие в команде

- Изменения происходят часто и быстро — меняется процесс
- Окружения стандартные, разработка лучше понимает в каких условиях ПО работает в production — общий язык с эксплуатацией

Непрерывная поставка ПО

Повышается качество продукта

- Запуск в эталонном окружении
- Автотесты и анализаторы кода
- Одни и те же подходы для всех окружений
- Доступность команде мониторинга и логов ПО

Непрерывная поставка ПО

Уменьшение рисков и стресса

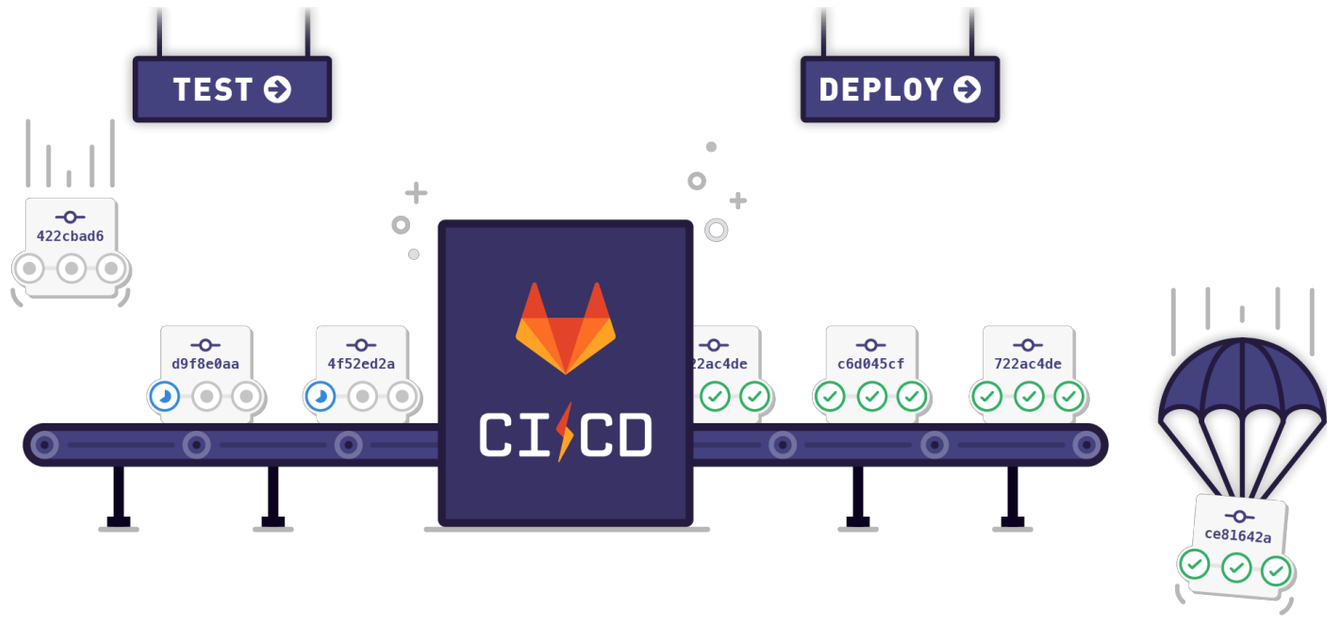
- Небольшие и частые изменения
- Меньше изменений за раз – проще откаты
- Проще отлаживать проблемы и разбирать инциденты

Из чего состоит CD и чем Gitlab может помочь

- **Version Control**
- **> Continuous Integration**
- **> Deployment Automation**
- **Monitoring**
- **Trunk-Based Development**
- **Test Automation**
- Test Data Management
- Shift Left on Security
- Loosely Coupled Architecture
- Empowered Teams
- Proactive Notification
- …

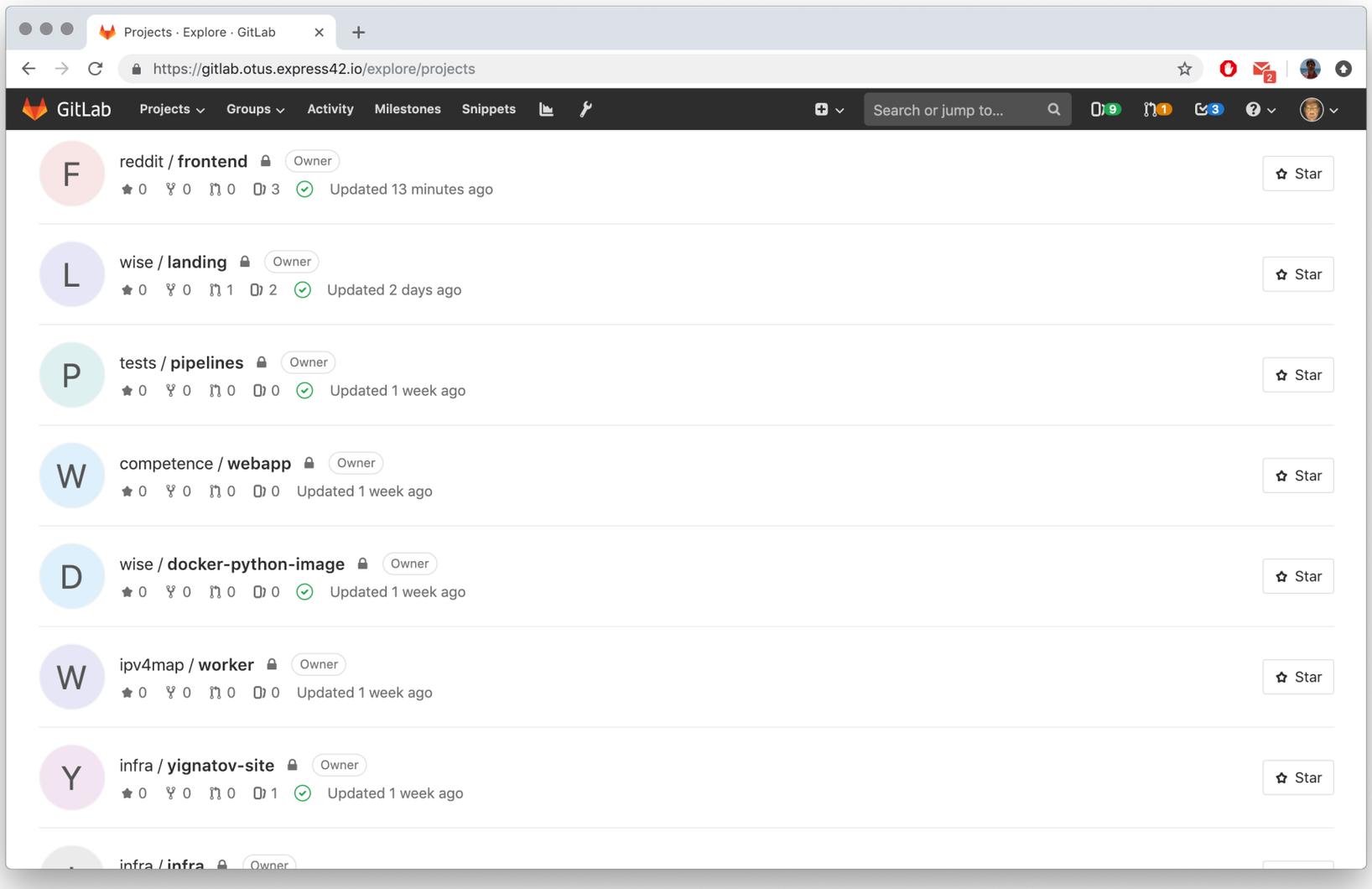
Gitlab CI

- Развивается с 2011 года
- SaaS и приватные инсталляции
- Community Edition
- Enterprise Edition
  - Multiple LDAP / AD server support
  - Remote repository mirroring
  - Support
  - HA
  - ...



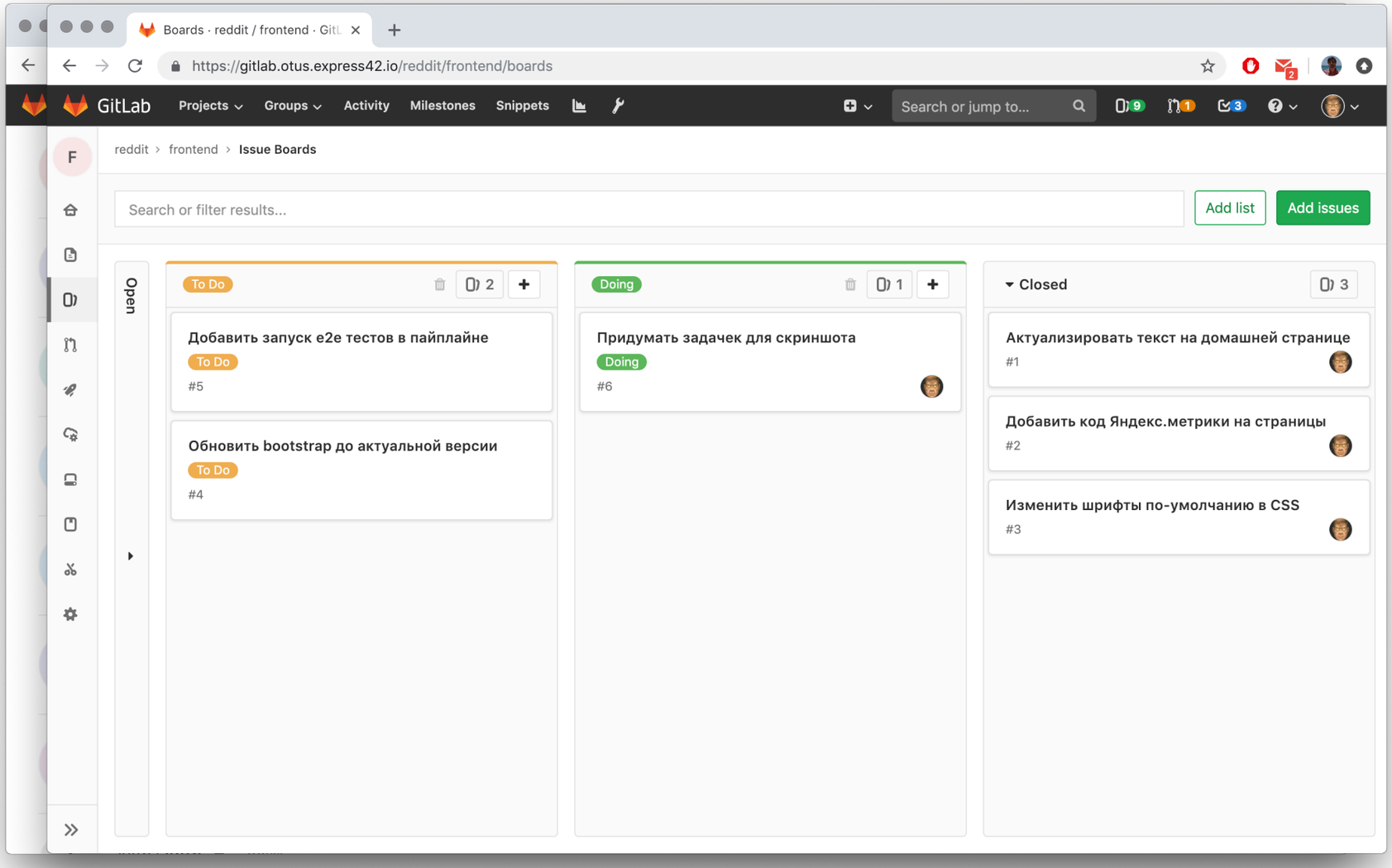
Функционал Gitlab CI

Проекты и группы проектов



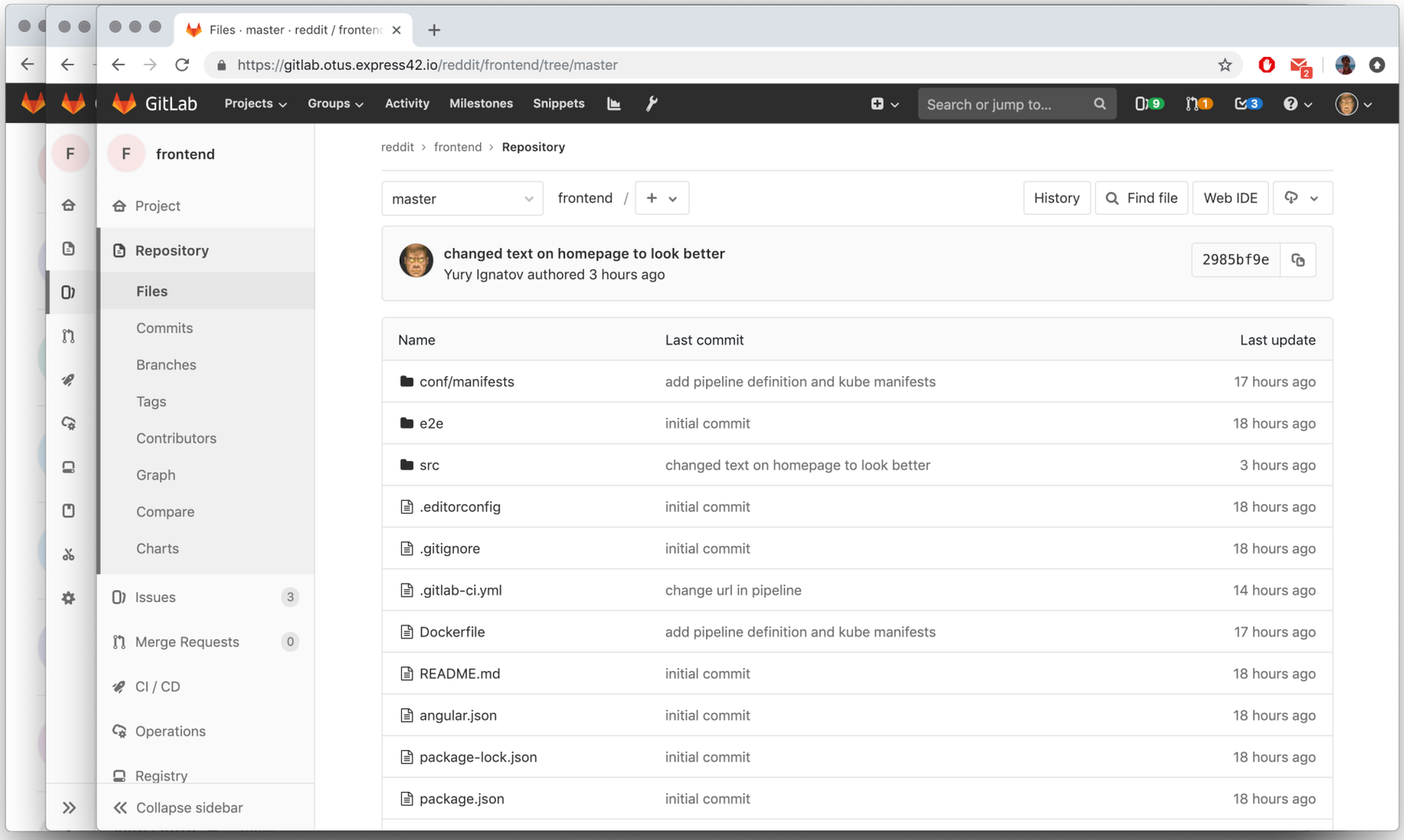
Функционал Gitlab CI

Таск-треккер и доски



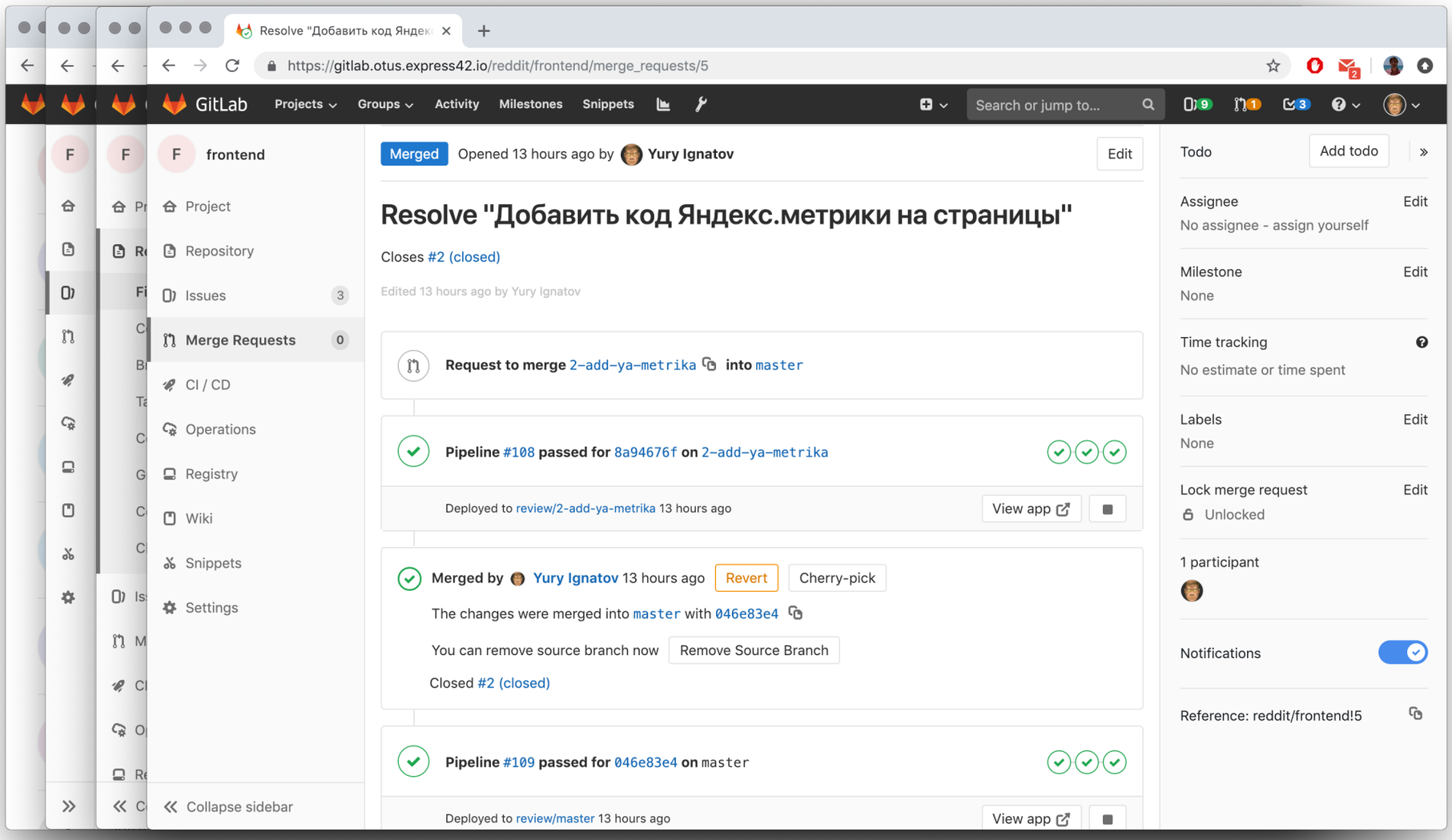
Функционал Gitlab CI

Репозитории



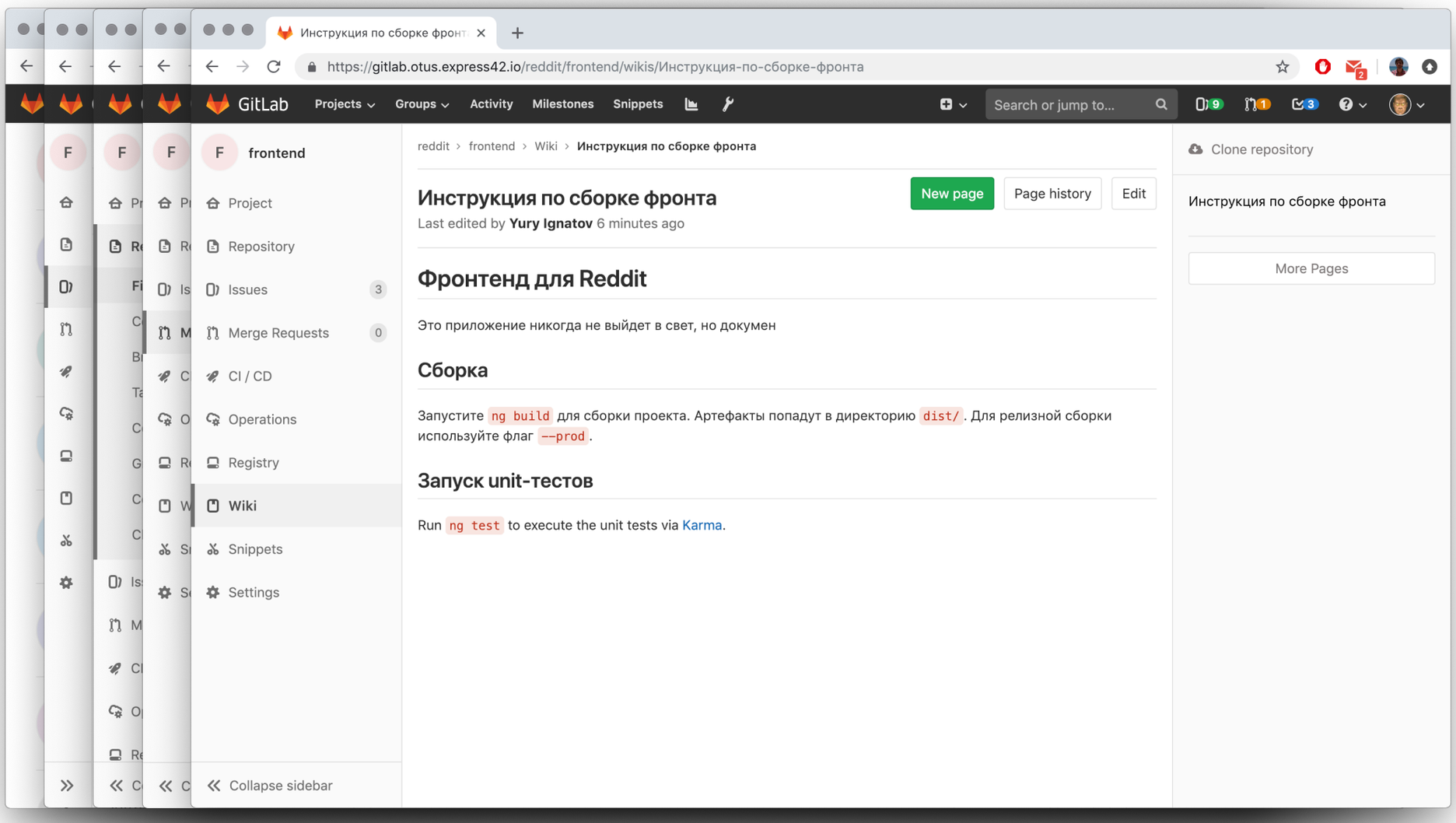
Функционал Gitlab CI

Merge requests



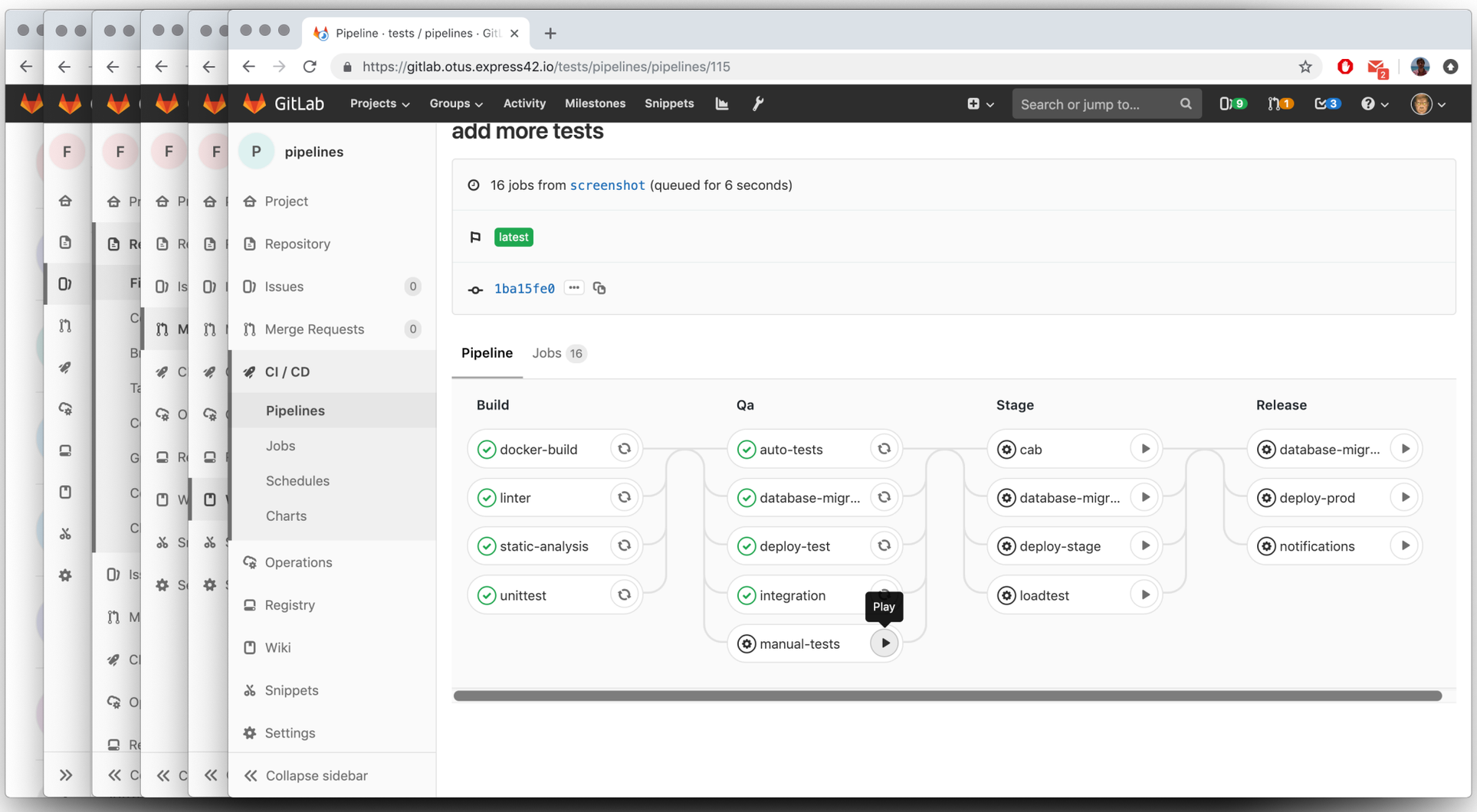
Функционал Gitlab CI

wiki



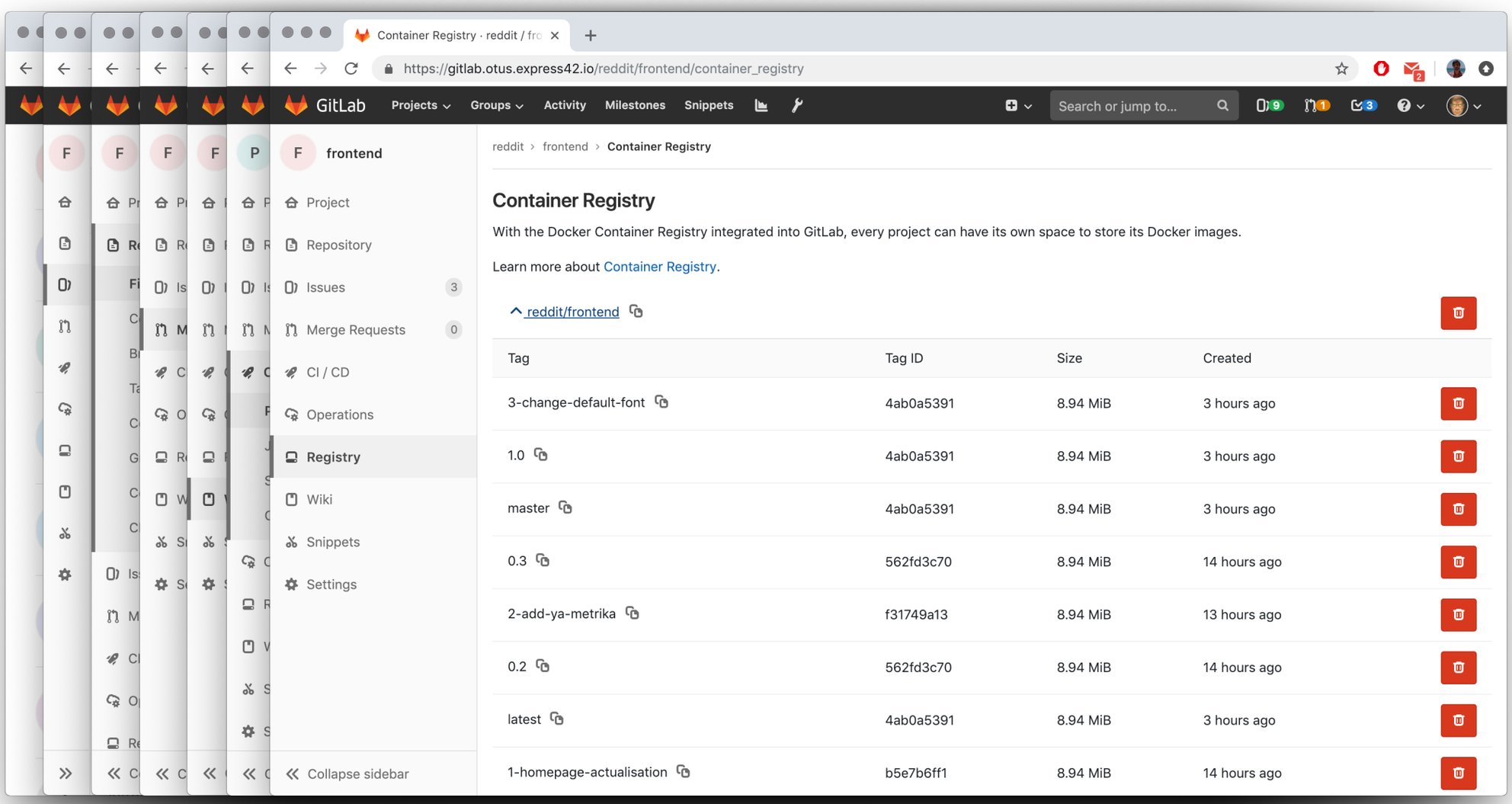
Функционал Gitlab CI

CI/CD Pipelines



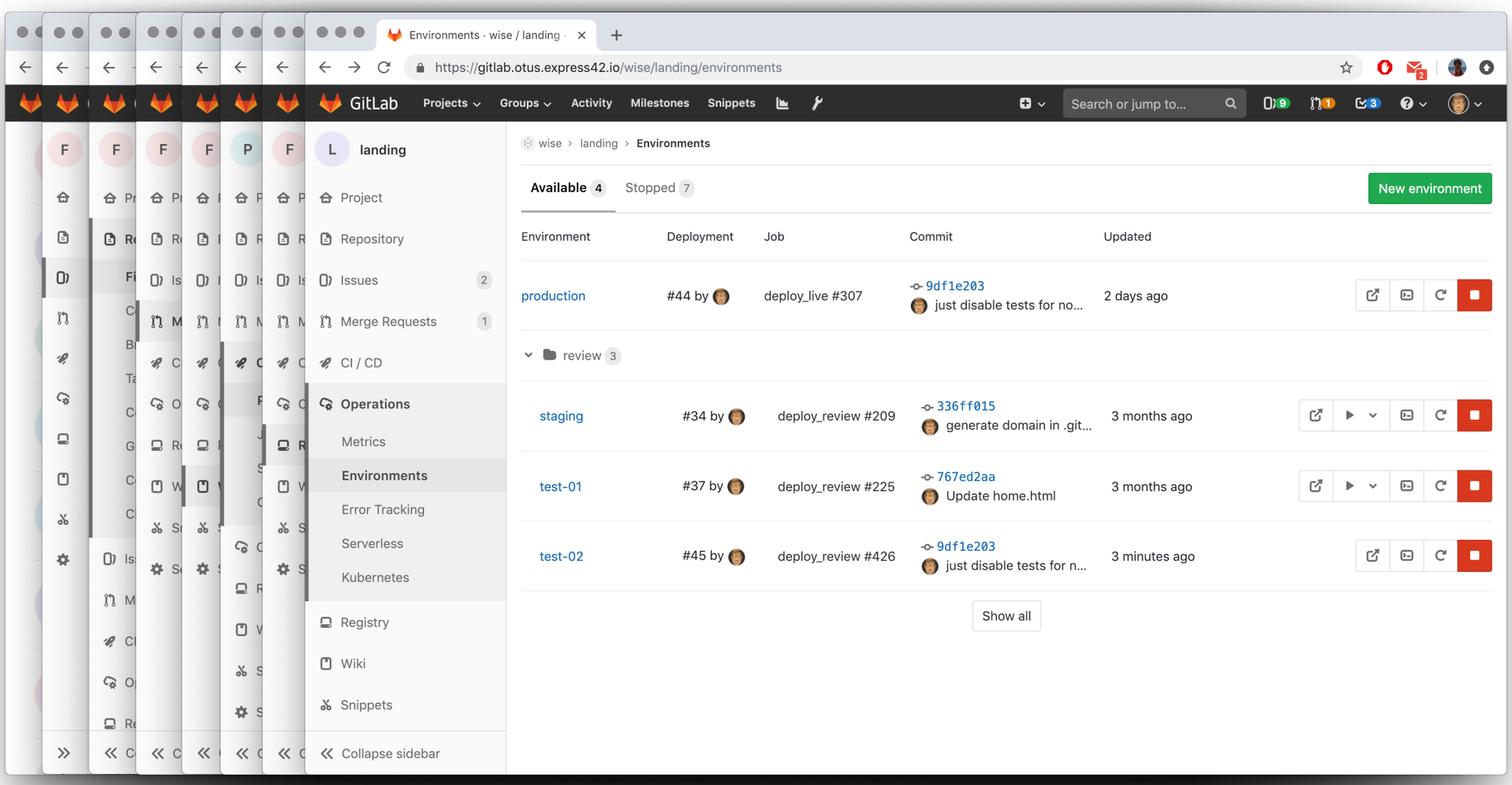
Функционал Gitlab CI

Хранилище артефактов



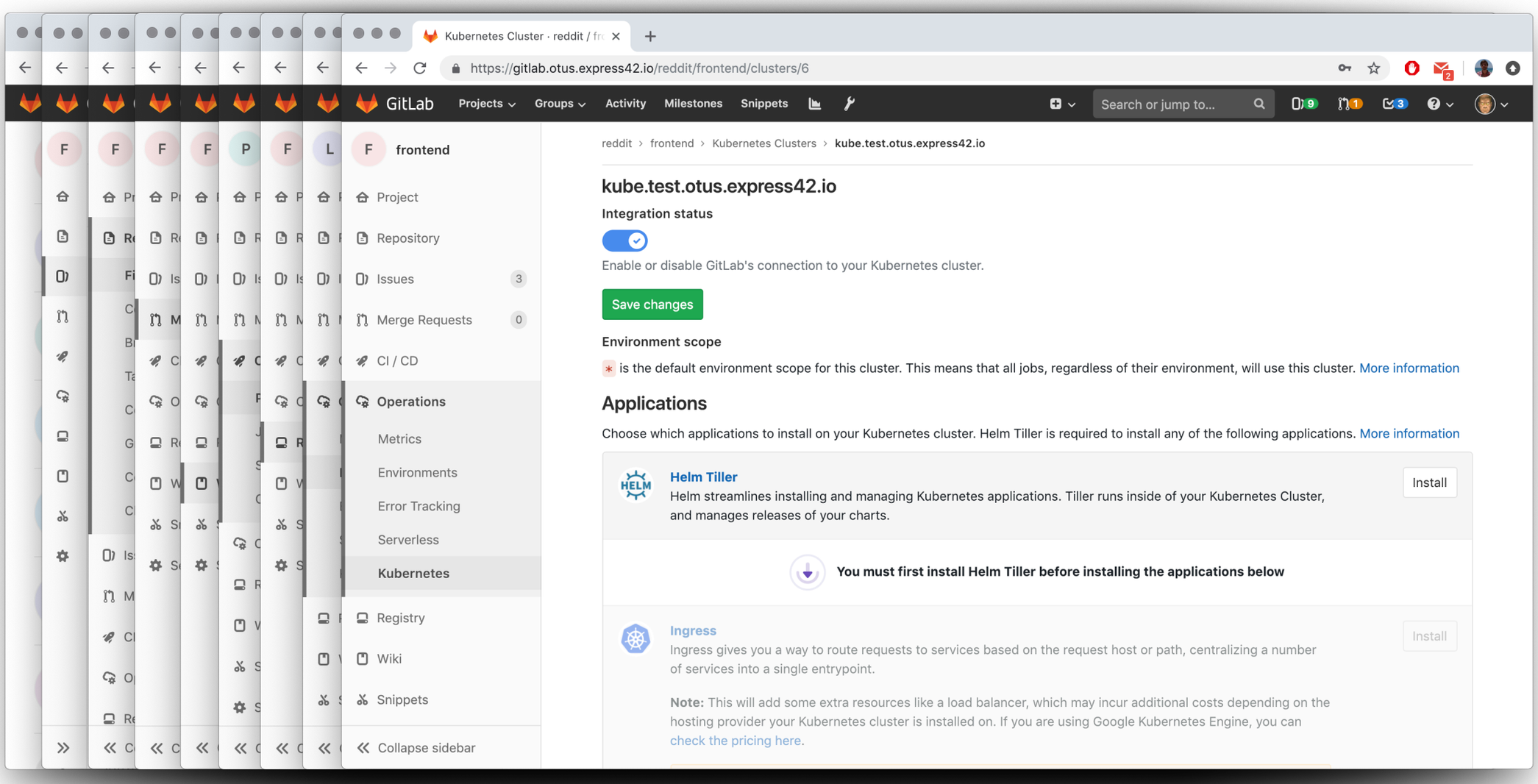
Функционал Gitlab CI

Статус окружений



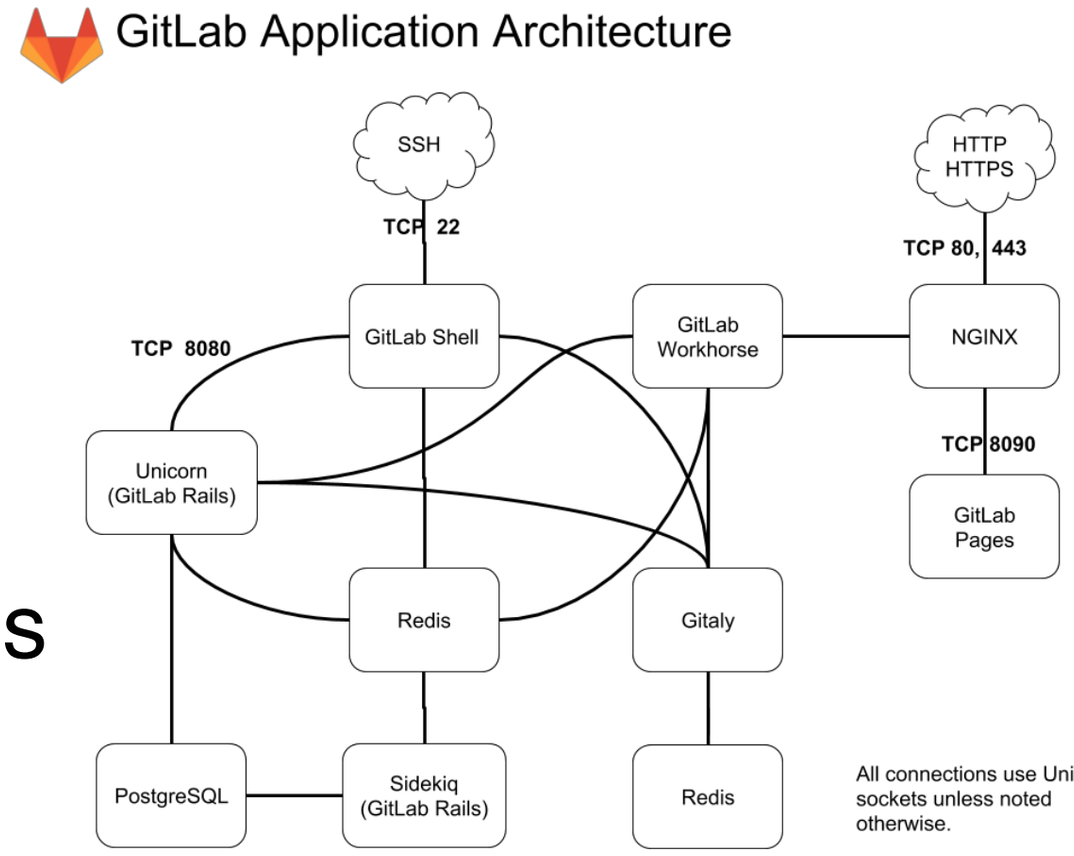
Функционал Gitlab CI

Интеграция с kubernetes, prometheus, sentry и многое другое



Устройство Gitlab CI

- Ruby on Rails, Go
- Postgres, Redis
- Sidekiq, Nginx, Prometheus

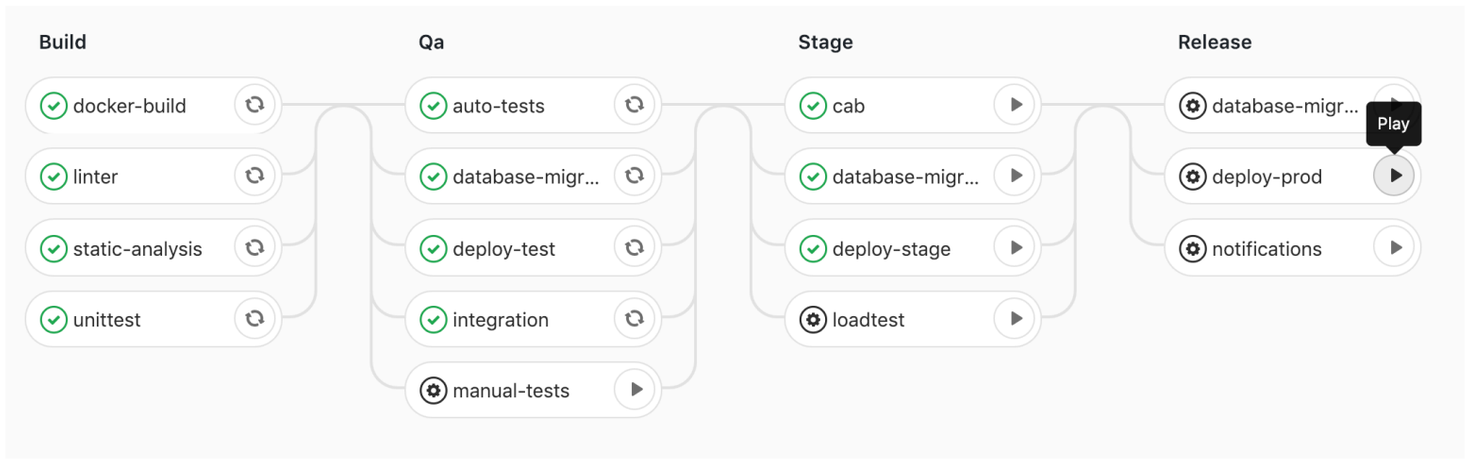


https://docs.gitlab.com/ce/development/architecture.html

https://about.gitlab.com/handbook/engineering/infrastructure/production-architecture/

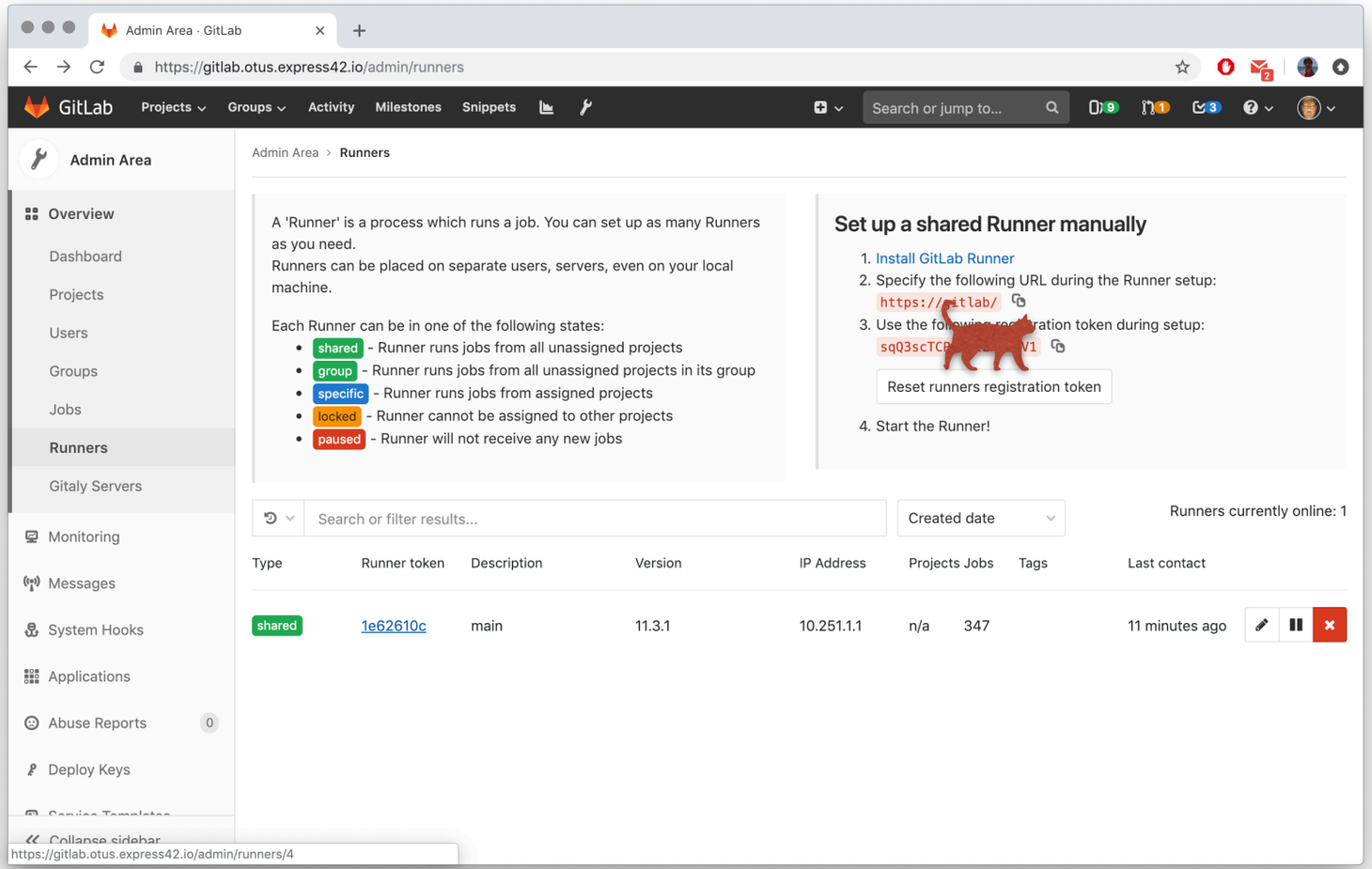
CI/CD Pipelines

- Пайплайн определяется на репозиторий файлом .gitlab-ci.yml
- Состоит из задач (jobs)
- Задачи поделены на этапы (stages)
- Задачи выполняются на Gitlab Runners
- Запускается по коммиту, по кнопке, по расписанию, по триггеру



Gitlab Runner

- Процесс, выполняющий Jobs
- Состояния runner (shared, group, specific)
- Executors
- Теги



Job

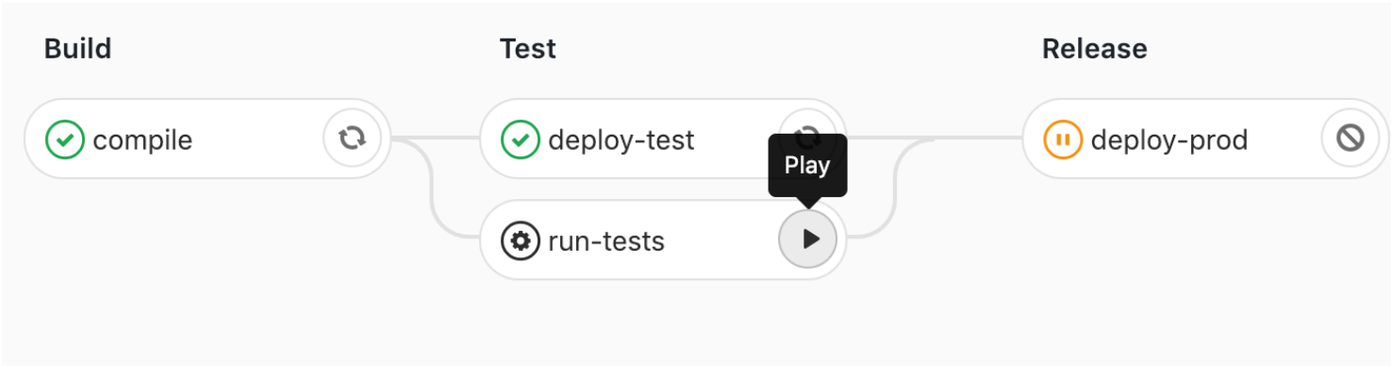
- Определяется кодом
- Относится к конкретному этапу (stage) пайплайна
- Каждый Job выполняется независимо
- Могут использовать переменные
- Директивы only/except, when, allow_failure, retry

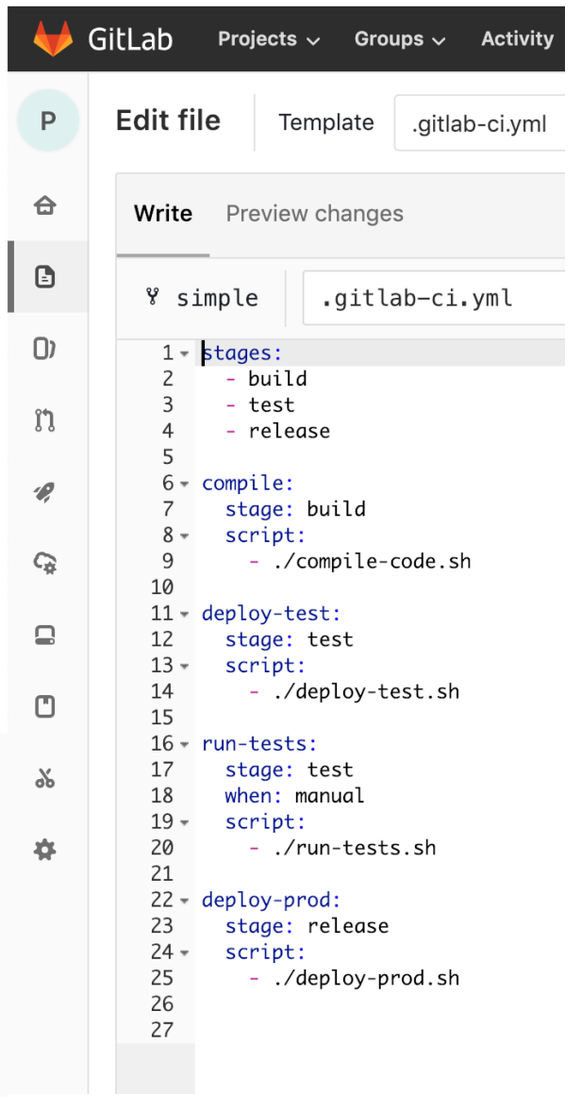
Описание пайплайна .gitlab-ci.yml

Pipeline состоит из нескольких этапов

Jobs в одном этапе выполняются параллельно

Stages выполняются последовательно

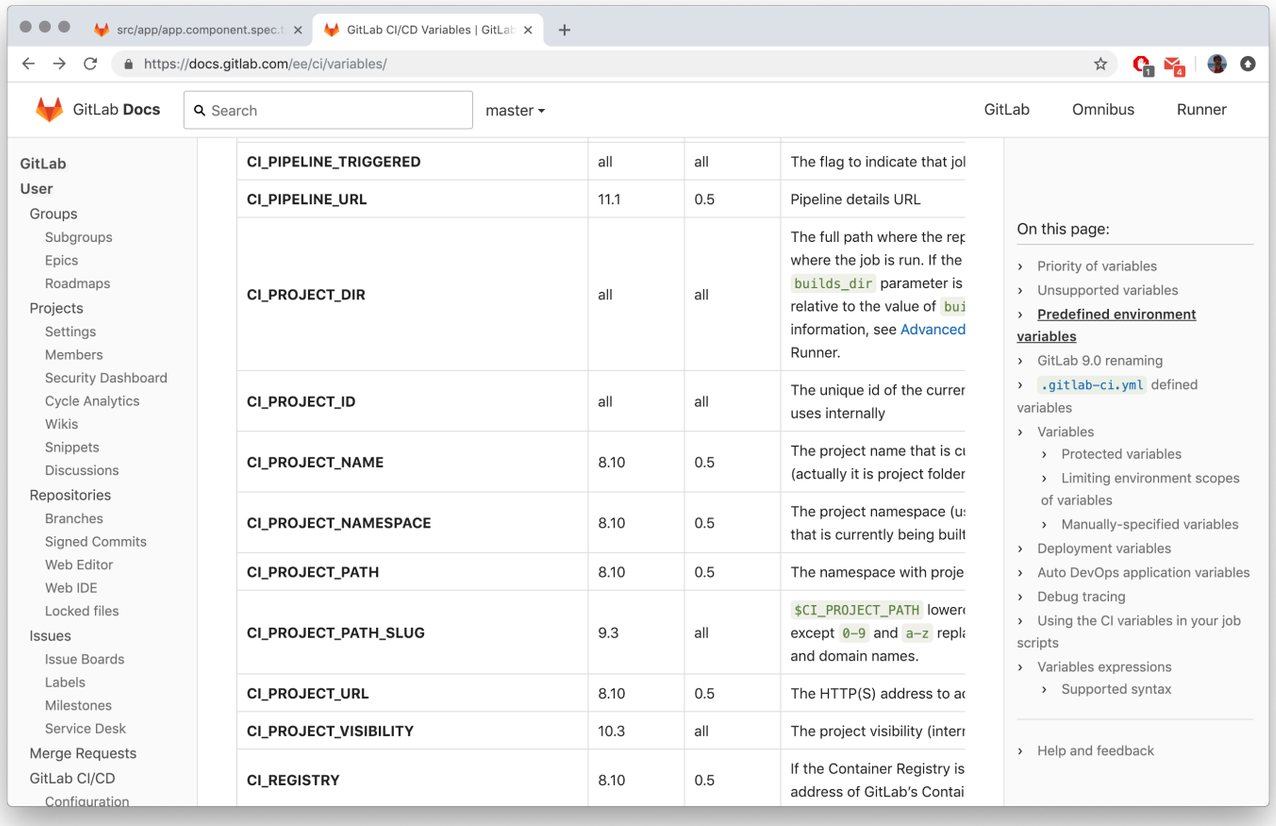




Variables

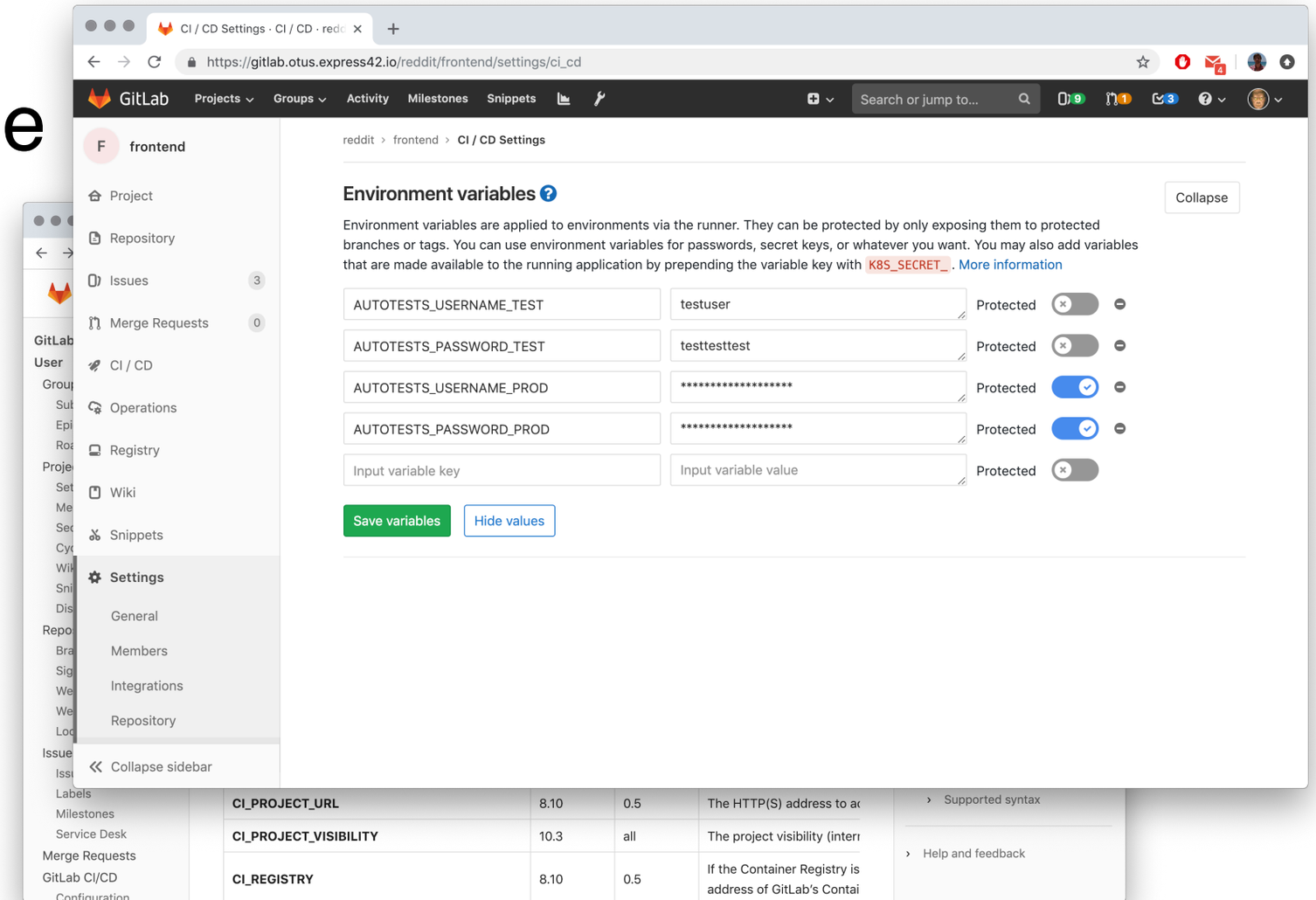
- Встроенные
- Пользовательские
- Секретные

https://docs.gitlab.com/ce/ci/variables/



Variables

- Встроенные
- Пользовательские
- Секретные



Variables

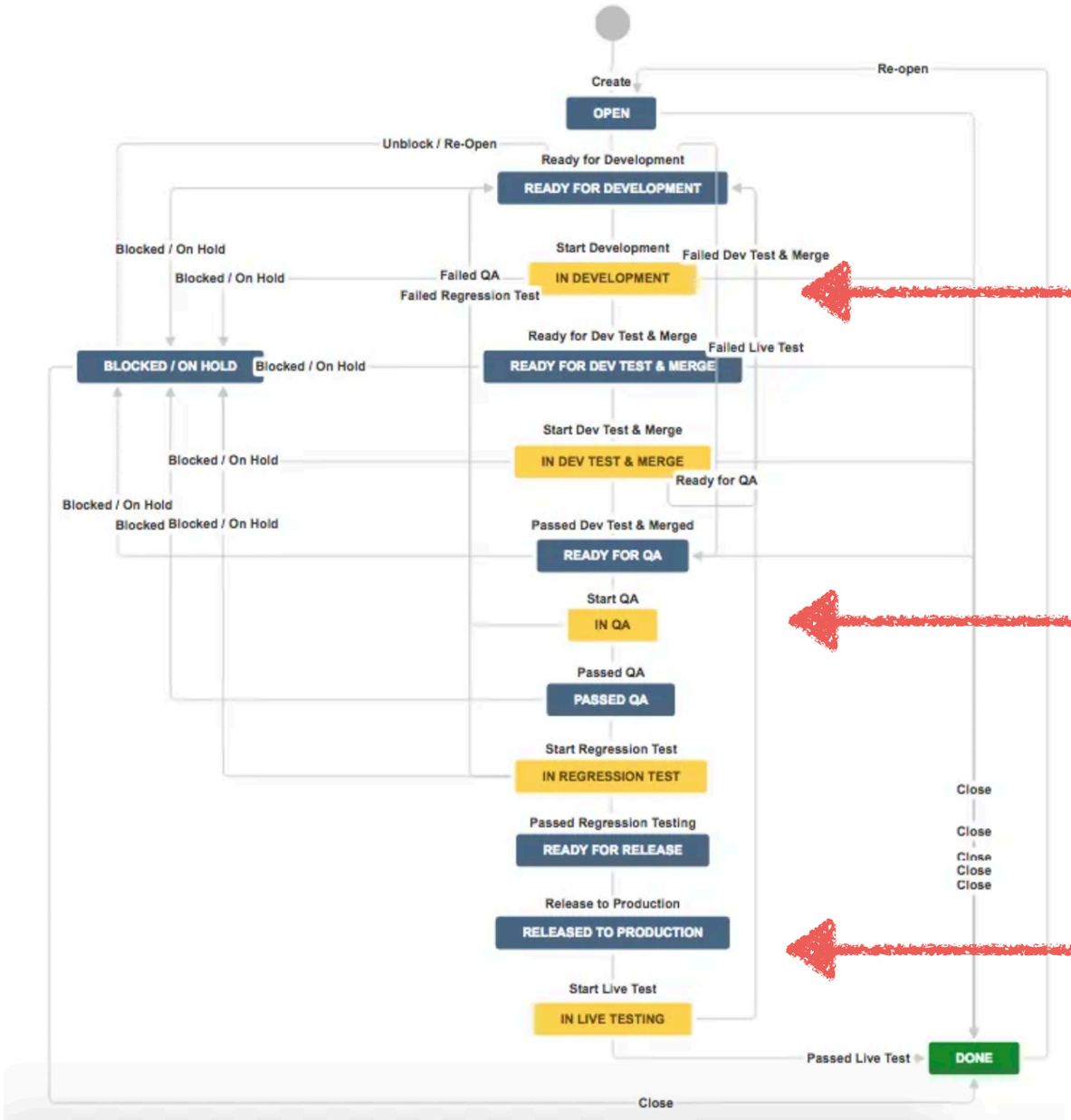
```
variables:
  REGISTRY_URL: registry.otus.express42.io
  CONTAINER_IMAGE: $REGISTRY_URL/$CI_PROJECT_NAMESPACE/$CI_PROJECT_NAME

before_script:
  - docker info
  - docker login -u gitlab-ci-token -p $CI_JOB_TOKEN $REGISTRY_URL

build:
  stage: build
  script:
    - docker build --pull -t $CONTAINER_IMAGE .
    - docker tag $CONTAINER_IMAGE:latest $CONTAINER_IMAGE:${CI_COMMIT_SHA}
    - docker push $CONTAINER_IMAGE:${CI_COMMIT_SHA}

test:
  stage: test
  script:
    - docker pull $CONTAINER_IMAGE:${CI_COMMIT_SHA}
    - docker run $CONTAINER_IMAGE:${CI_COMMIT_SHA} nginx -t
    - docker run $CONTAINER_IMAGE:${CI_COMMIT_SHA} ./tests.sh
```

Непрерывная поставка ПО

Процесс разработки и эксплуатации ПО, при котором каждое изменение может быть выкачено в боевое окружение

Какой путь проходит изменение?

Путь изменения: воркфлоу

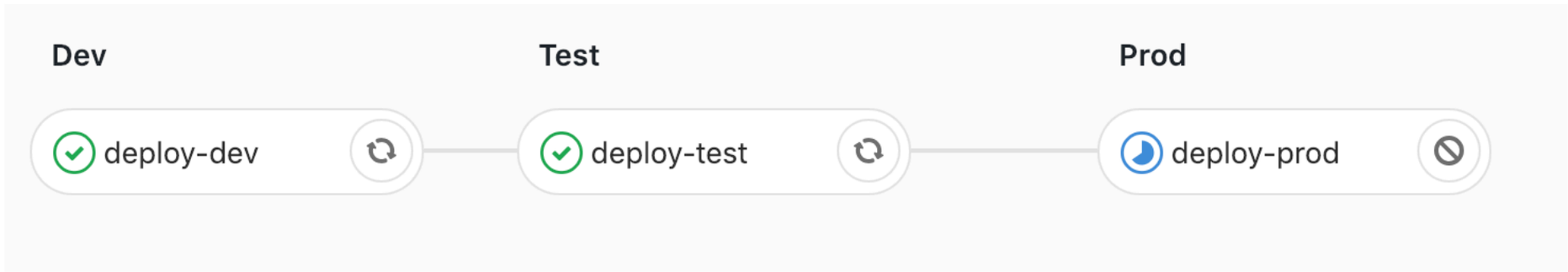


Окружение разработки

Окружение тестирования

Продакшн

Путь изменения: деплой

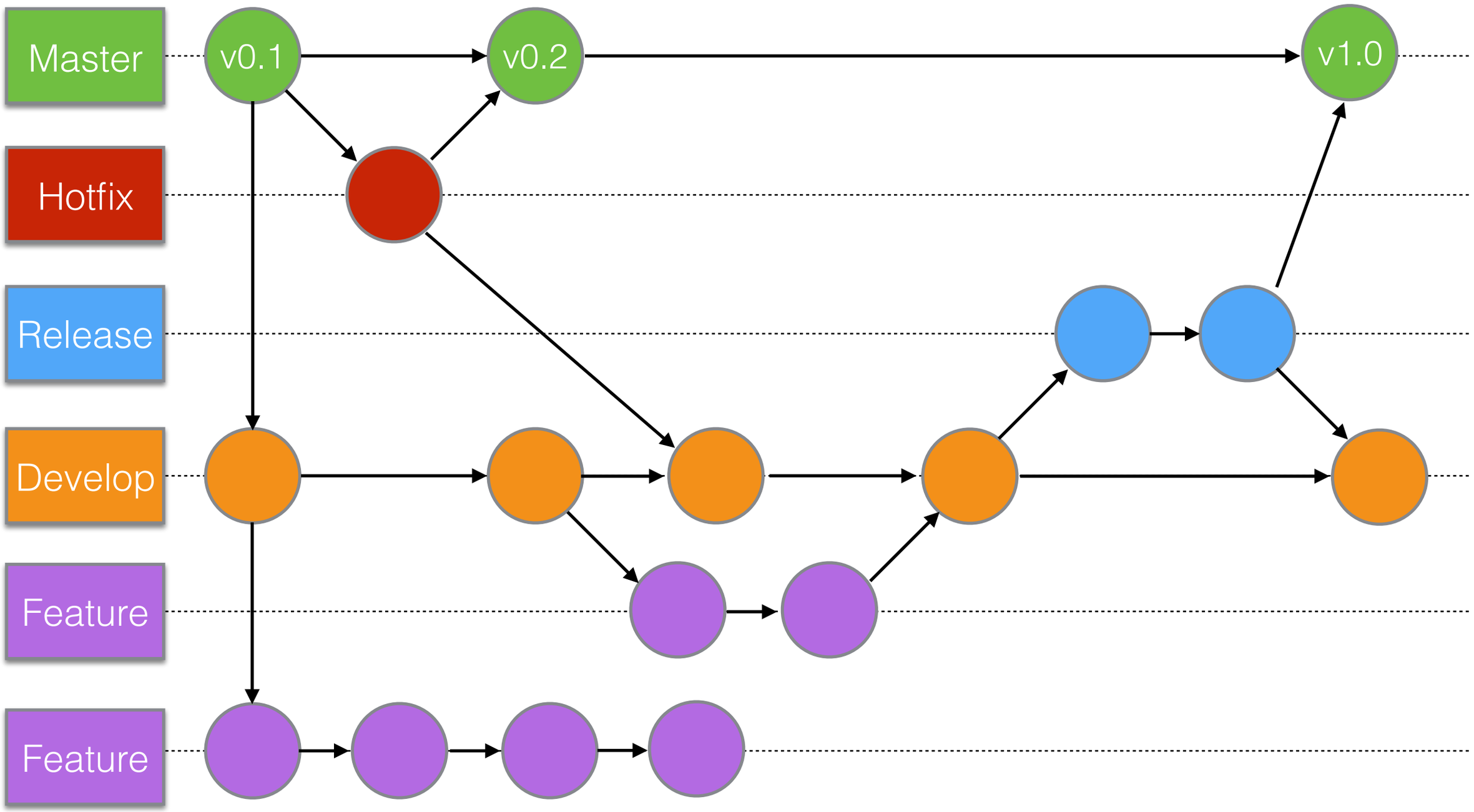


Изменение

landing/migrations/0003_auto_20181007_2026.py 0 → 100644 View file @ f41d8db3

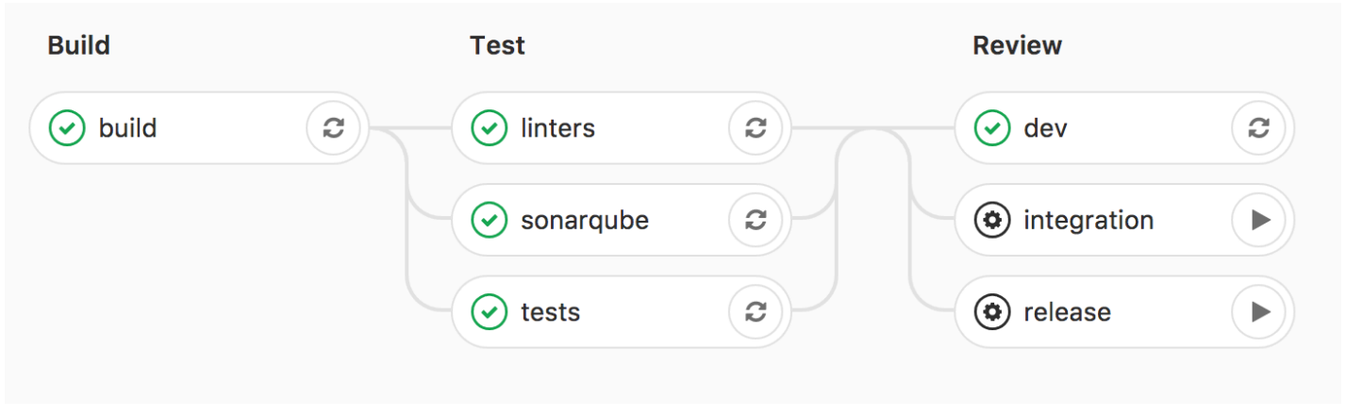
```
+ class Migration(migrations.Migration):
+
+     dependencies = [
+         ('landing', '0002_auto_20181007_2013'),
+     ]
+
+     operations = [
+         migrations.CreateModel(
+             name='Concept',
+             fields=[
+                 ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),
+                 ('symbol', models.CharField(max_length=64)),
+                 ('display_name', models.CharField(max_length=64)),
+                 ('created_date', models.DateTimeField(default=django.utils.timezone.now)),
```

Путь изменения: в git flow

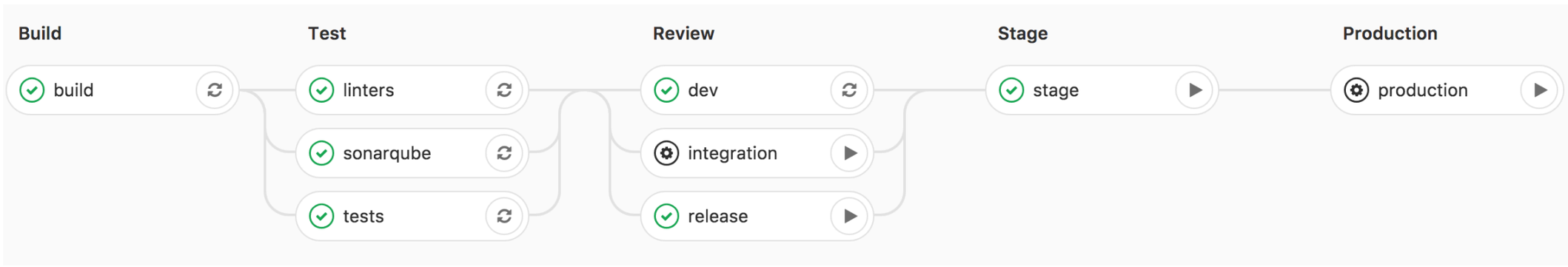


Не каждое изменение нужно выкатывать на все окружения

```
git commit -a -m 'Add more comments for code from make it more readable'
git push
```

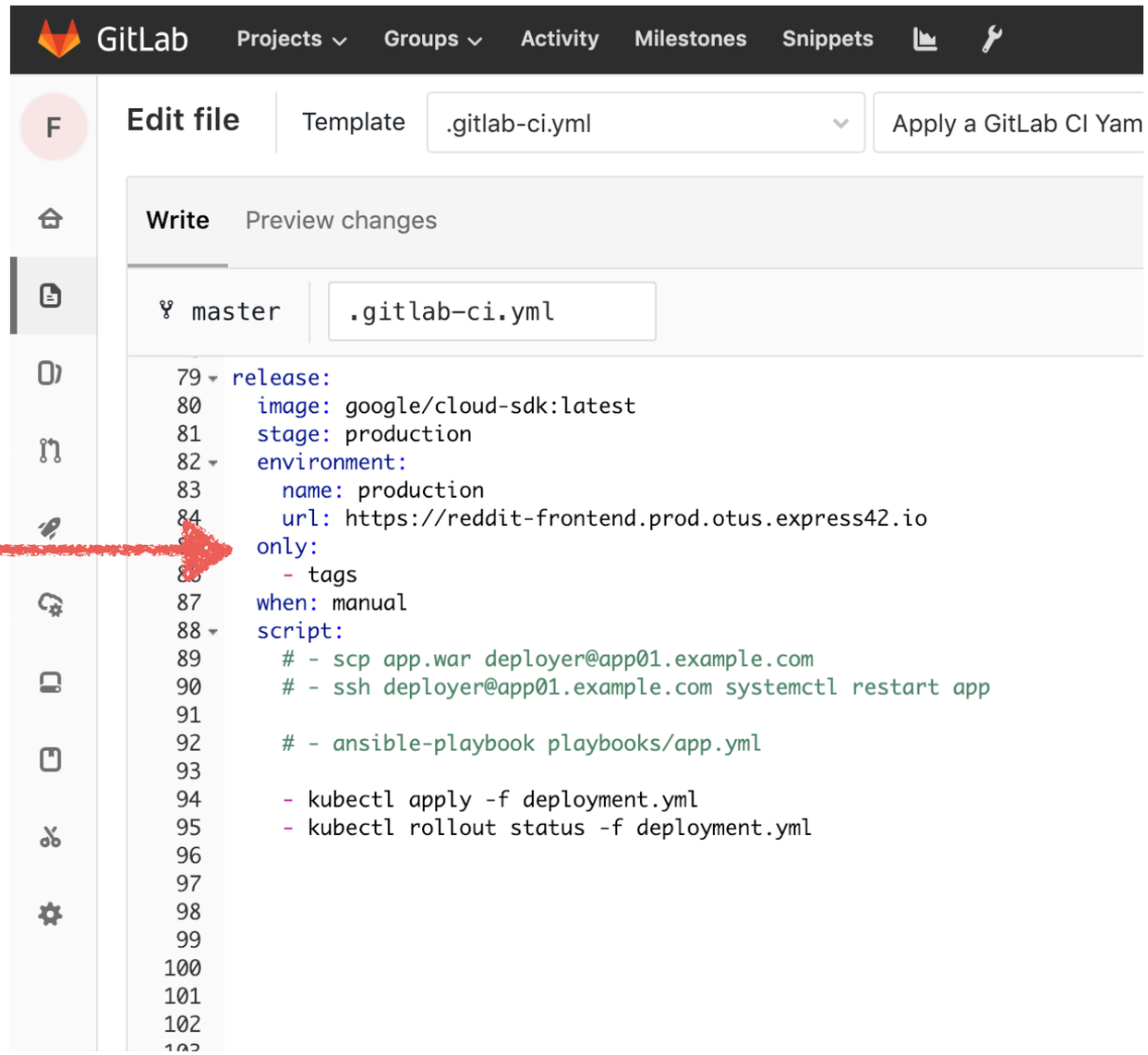



```
git commit -a -m ‘New login page design’
git tag 2.1.0
git push --tags
```



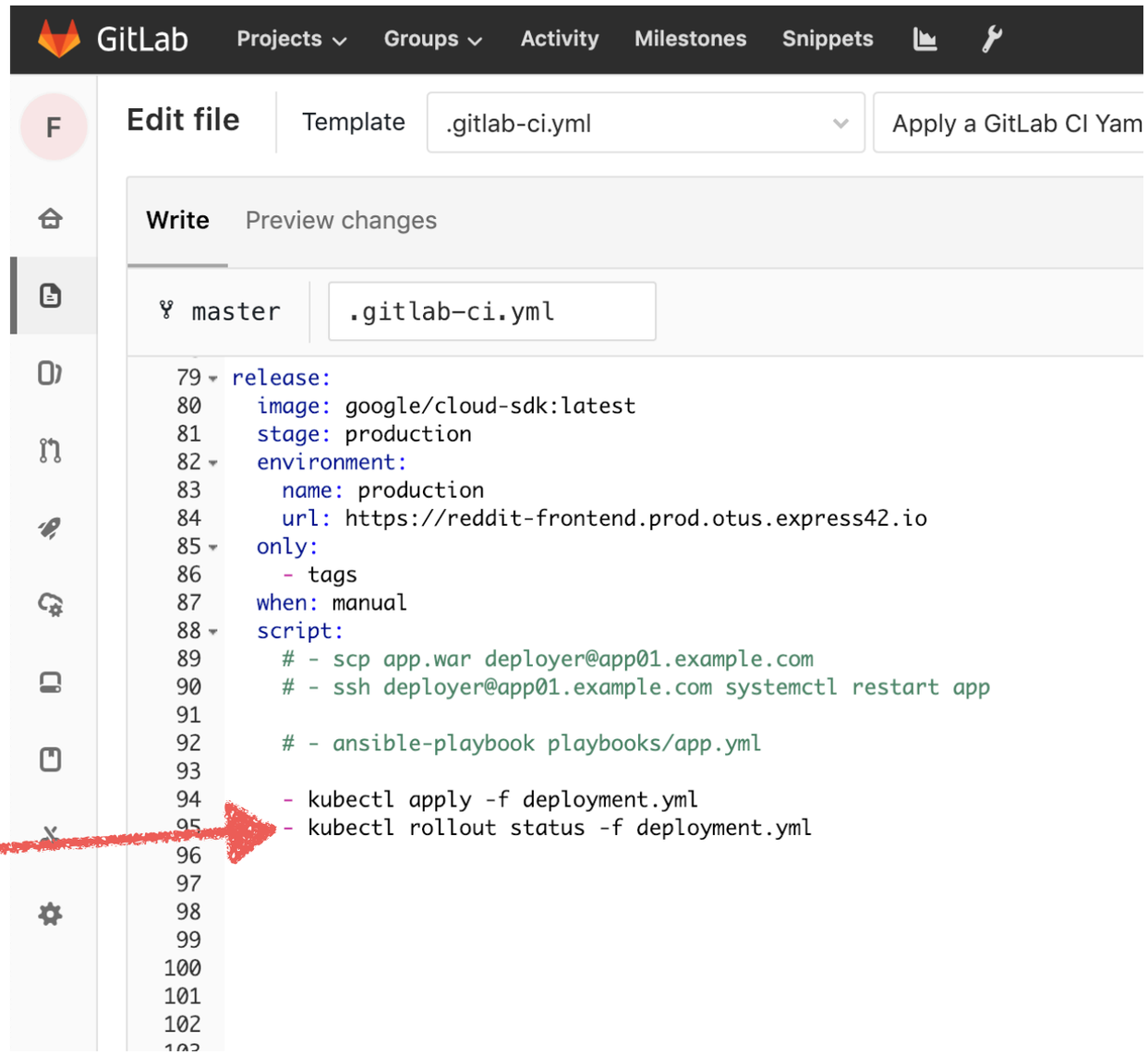
Не каждое изменение нужно выкатывать на все окружения

Например, только изменения, помеченные тегом



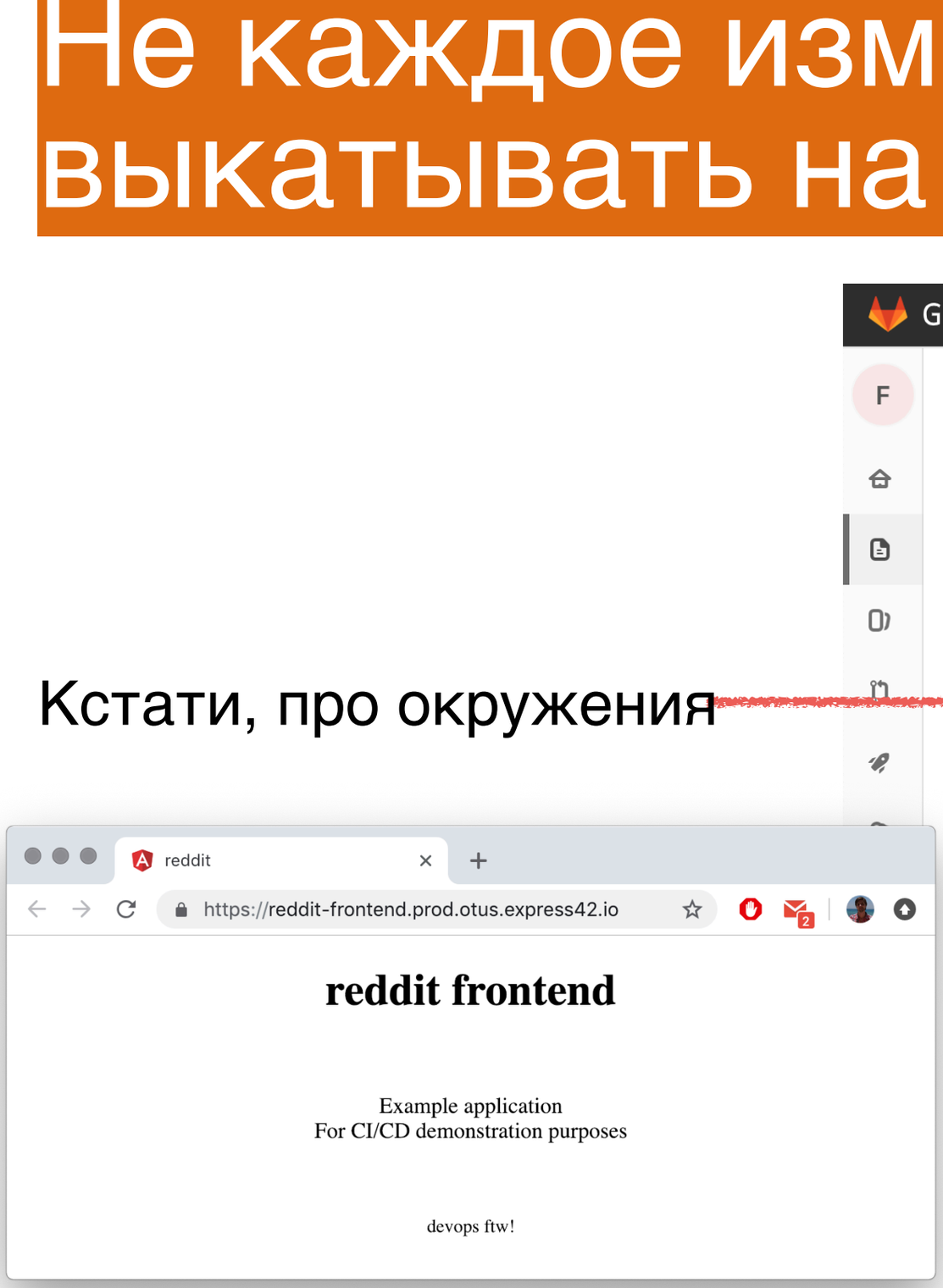
Не каждое изменение нужно выкатывать на все окружения

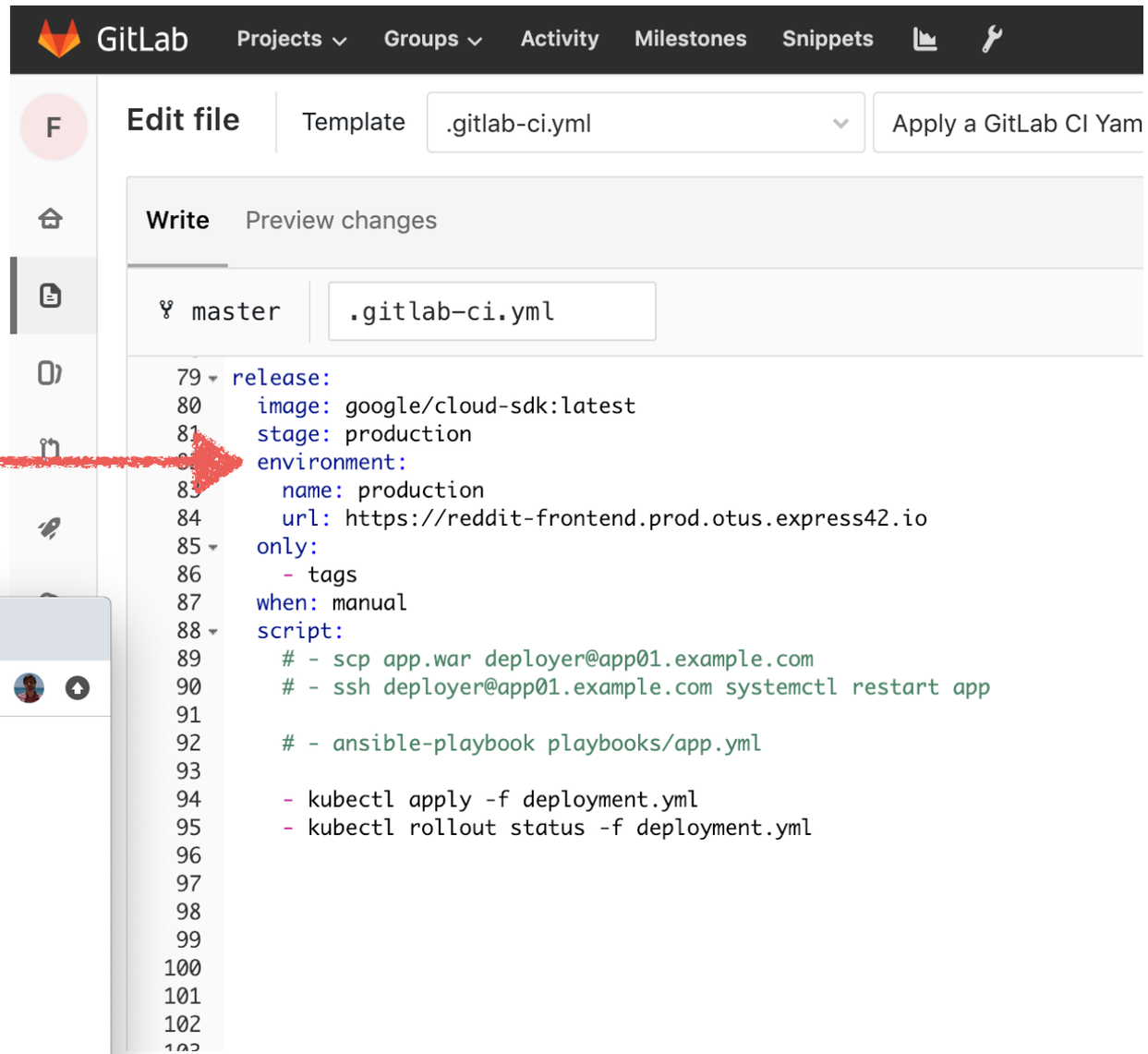
Для частых релизов нужно обновление без простоев (Zero-downtime deploy)



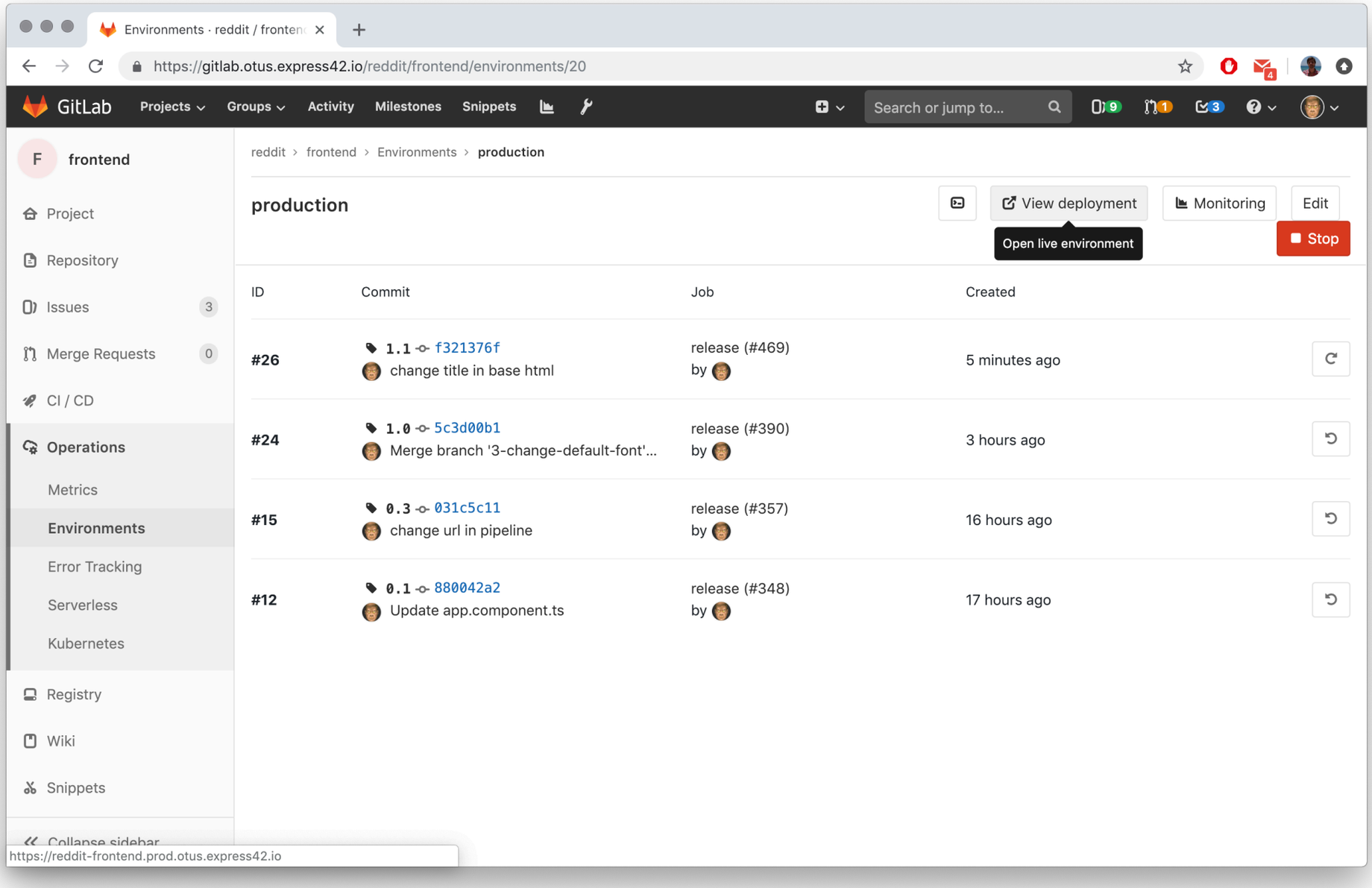
Не каждое изменение нужно выкатывать на все окружения

Кстати, про окружения





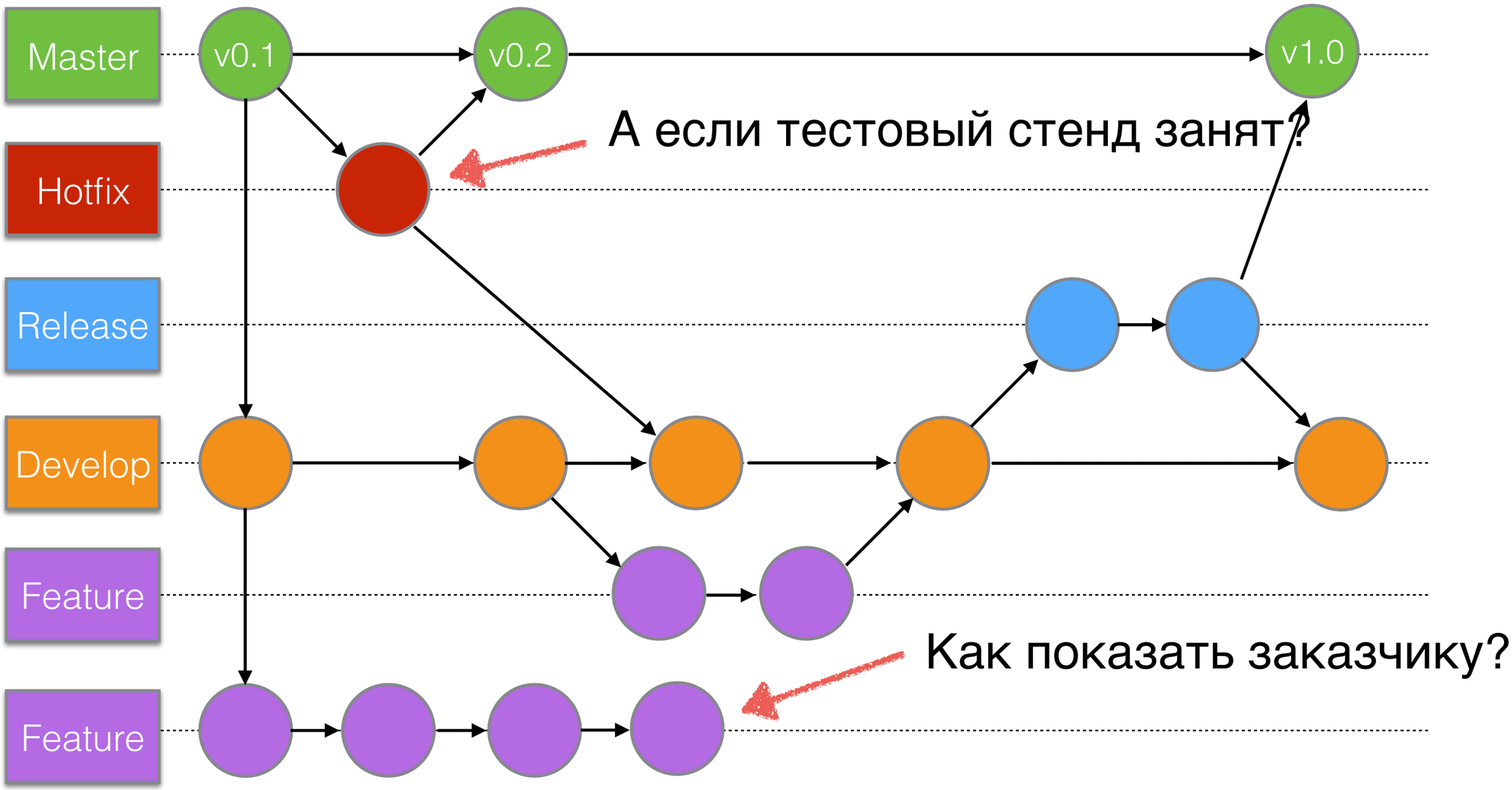
Окружения в Gitlab CI



Динамические окружения

- Изолированные окружения под задачу / фичу
- Создаются и удаляются “по кнопке”
- Должны быть короткоживущими, как и фича-ветки
- Детали реализации

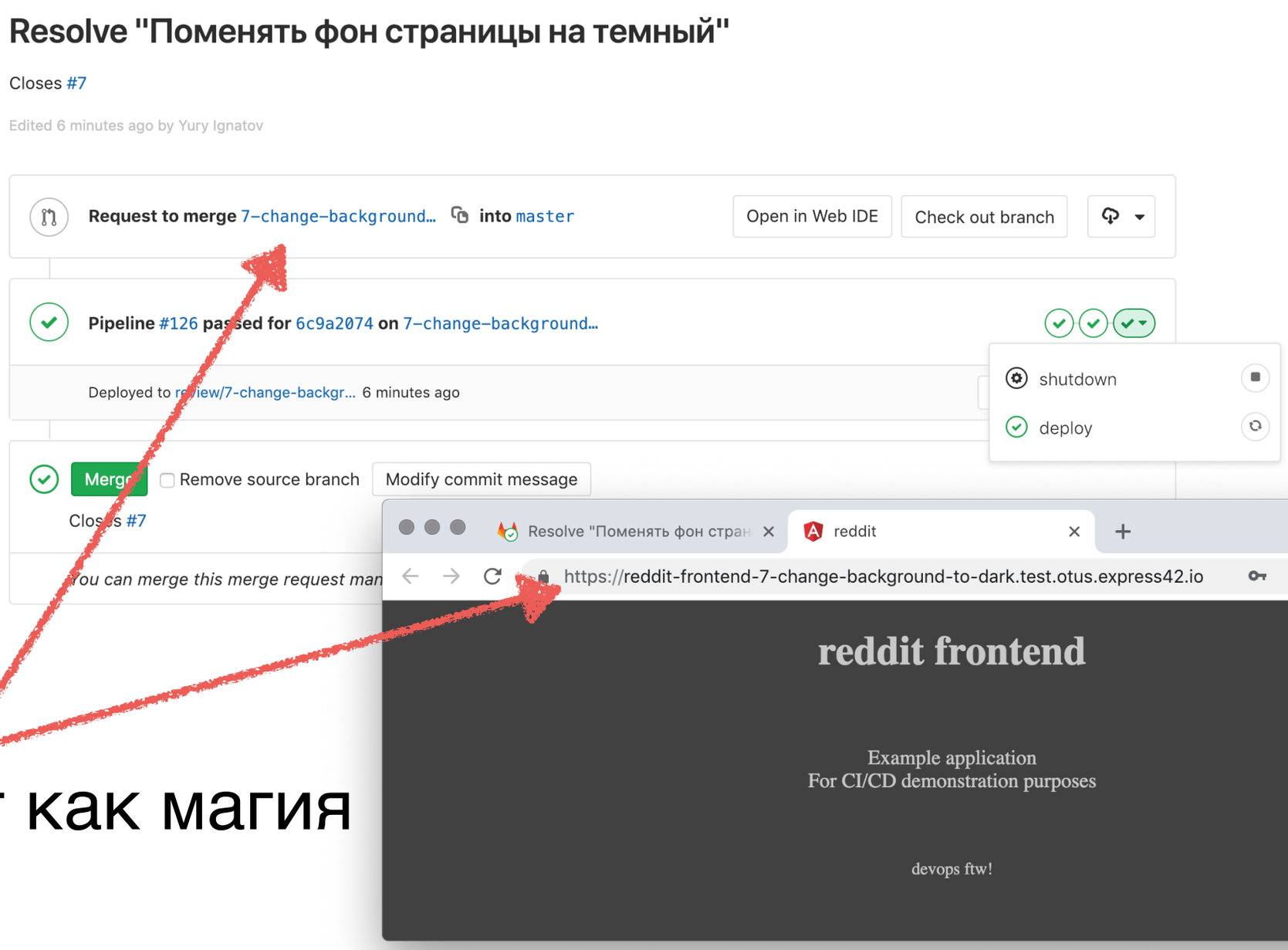
Путь изменения: в git flow



Динамические окружения

```
deploy-feature:
  stage: review
  environment:
    name: review/$CI_BUILD_REF_NAME
    url: https://reddit-frontend-$CI_BUILD_REF_SLUG.test.otus.express42.io
    on_stop: shutdown
  script:
    - ./create-environment-if-not-exists.sh
    - ./deploy-app.sh

shutdown:
  stage: review
  variables:
    GIT_STRATEGY: none
  when: manual
  environment:
    name: review/$CI_BUILD_REF_NAME
    action: stop
  script:
    - ./destroy-environment.sh
```

Динамические окружения

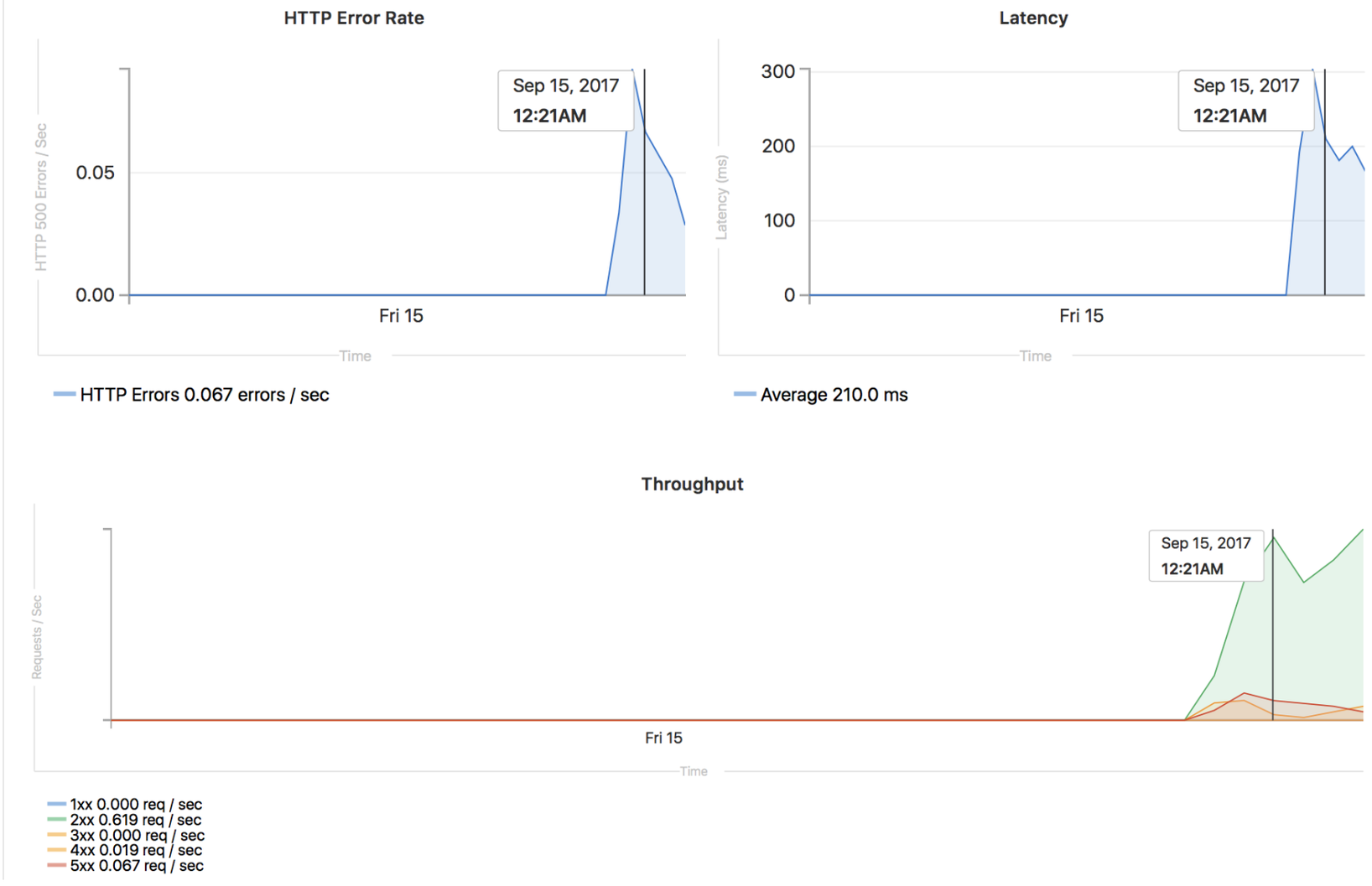


Выглядит как магия

Мониторинг окружений

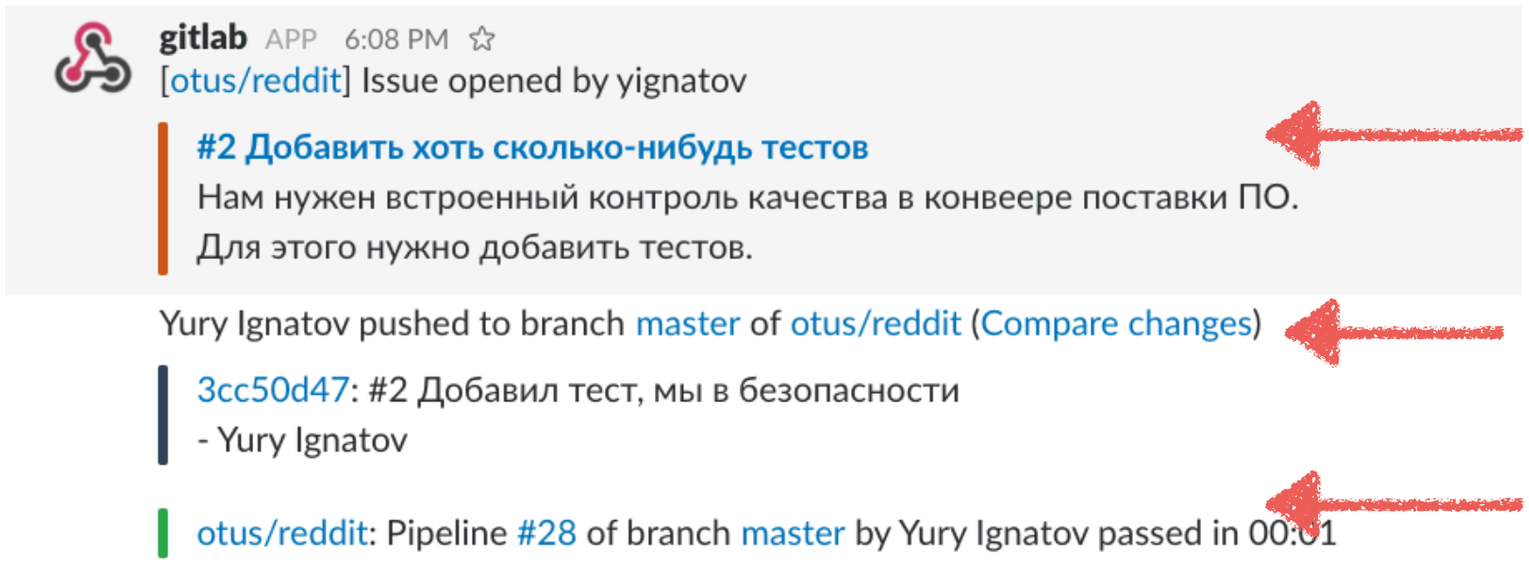
Environment: production

Response metrics (NGINX Ingress)



Интеграции

- Хорошие интеграции повышают прозрачность
- Много всего из коробки
- Легко интегрируется по HTTP API
- Подходит для ChatOps



создали задачу

выполнили работу

получили фидбек

Спасибо!

- Ждем ваших вопросов
- Дело