Отчёт

Настройка и интеграция сервисов **Kanboard/Gitbucket/Kraken**

Работу выполнила:

Студентка группы ИВТ-32БО

К.А. Ципилева

24.06.2021

1. Общая структура конфигурации Docker Compose

```
version: '3.6'

services:
  gitbucket:
    build: ./gitbucket
    image: gitbucket
    ports:
      - 3000:8080
    volumes:
      - ./gitbucket/data/:/gitbucket


  kanboard:
    image: kanboard/kanboard:latest
    ports:
      - 8000:80
      - 443:443
    volumes:
      - ./kanboard/data:/var/www/app/data
      - ./kanboard/plugins:/var/www/app/plugins


  server:
    restart: always
    image: eu.gcr.io/kraken-261806/kkserver:0.567
    environment:
      - KRAKEN_REDIS_ADDR
      - KRAKEN_DB_URL
      - KRAKEN_CLICKHOUSE_PORT
      - KRAKEN_CLICKHOUSE_ADDR
      - KRAKEN_CLICKHOUSE_URL
      - KRAKEN_SERVER_PORT
      - KRAKEN_SERVER_ADDR
      - KRAKEN_PLANNER_URL
      - KRAKEN_MINIO_ADDR
      - MINIO_ACCESS_KEY
      - MINIO_SECRET_KEY
    ports:
      - $KRAKEN_SERVER_PORT:$KRAKEN_SERVER_PORT
    networks:
      - db_net
      - web_net
      - lab_net
```

```
    - lab_net
  depends_on:
    - postgres
    - controller
    - celery
    - clickhouse
    - clickhouse-proxy


controller:
  restart: always
  image: eu.gcr.io/kraken-261806/kkcontroller:0.567
  environment:
    - KRAKEN_REDIS_ADDR
    - KRAKEN_DB_URL
    - KRAKEN_CLICKHOUSE_PORT
    - KRAKEN_CLICKHOUSE_ADDR
    - KRAKEN_CLICKHOUSE_URL
    - KRAKEN_SERVER_PORT
    - KRAKEN_SERVER_ADDR
    - KRAKEN_PLANNER_URL
  networks:
    - db_net
  depends_on:
    - celery
    - postgres
    - clickhouse-proxy


celery:
  restart: always
  image: eu.gcr.io/kraken-261806/kkcelery:0.567
  environment:
    - KRAKEN_REDIS_ADDR
    - KRAKEN_DB_URL
    - KRAKEN_CLICKHOUSE_PORT
    - KRAKEN_CLICKHOUSE_ADDR
    - KRAKEN_SERVER_PORT
    - KRAKEN_SERVER_ADDR
    - KRAKEN_PLANNER_URL
    - KRAKEN_MINIO_ADDR
    - MINIO_ACCESS_KEY
    - MINIO_SECRET_KEY
  networks:
    - db_net
  depends_on:
    - postgres
    - redis


agent:
  restart: always
  image: eu.gcr.io/kraken-261806/kkagent:0.567
  environment:
    - KRAKEN_CLICKHOUSE_ADDR
    - KRAKEN_SERVER_ADDR
    - KRAKEN_AGENT_BUILTIN=1
```

```
    - KRAKEN_MINIO_ADDR
  networks:
    - lab_net
  depends_on:
    - server
    - minio
    - clickhouse-proxy
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock


ui:
  image: eu.gcr.io/kraken-261806/kkui:0.567
  environment:
    - KRAKEN_SERVER_ADDR
  ports:
    - $KRAKEN_UI_PUBLIC_PORT:80
  networks:
    - web_net
  depends_on:
    - server


postgres:
  image: postgres:11
  environment:
    - POSTGRES_USER
    - POSTGRES_PASSWORD
    - POSTGRES_DB
  volumes:
    - db-data:/var/lib/postgresql/data
  networks:
    - db_net


redis:
  image: redis:alpine
  networks:
    - db_net


# clickhouse & co.
clickhouse:
  image: eu.gcr.io/kraken-261806/clickhouse-server:20.11.4.13.0.567
  #--ulimit nofile=262144:262144
  volumes:
    - clickhouse:/var/lib/clickhouse
  ports:
    - "8123:8123"
    - "9000:9000"
  networks:
    - db_net


clickhouse-proxy:
  image: eu.gcr.io/kraken-261806/kkchproxy:0.567
  environment:
```

```
  clickhouse-proxy:
    image: eu.gcr.io/kraken-261806/kkchproxy:0.567
    environment:
      - KRAKEN_CLICKHOUSE_URL
    ports:
      - $KRAKEN_CLICKHOUSE_PORT:$KRAKEN_CLICKHOUSE_PORT/udp
    networks:
      - db_net
      - lab_net
    depends_on:
      - clickhouse


  minio:
    image: minio/minio:RELEASE.2020-12-18T03-27-42Z
    environment:
      - MINIO_ACCESS_KEY
      - MINIO_SECRET_KEY
    command: server --address :$KRAKEN_MINIO_PORT /data
    volumes:
      - minio:/data
    ports:
      - $KRAKEN_MINIO_PORT:$KRAKEN_MINIO_PORT
    networks:
      - lab_net
      - db_net


volumes:
  db-data:
  clickhouse:
  minio:

networks:
  db_net:
    driver: bridge
  web_net:
    driver: bridge
  lab_net:
    driver: bridge
```

У сервисов Kanboard и GitBucket есть свои Docerfile’. Kraken описан в общей конфигурации. У Kraken’а есть свои сервисы: server, controller, celery, agent, ui, postgres, redis, clickhouse, clickhouse-proxy, minio, которые нужны для работы данного инструмента. Информация об этих сервисах берется с это ресурса: **eu.gcr.io./kraken-261806** . Все сервисы взяты с официальных сатов.

Приложения работают на портах:

- Kanboard – 8000
- GitBucket – 3000
- Kraken – 8080

2. Описание Dockerfile

2.1. GitBucket

Данный файл брался с официального git репозитория проекта GitBucket

Образ базируется на openjdk:8-jre

```
FROM openjdk:8-jre
```

Добавление приложения gitbucket.war

```
ADD https://github.com/gitbucket/gitbucket/releases/download/4.35.3/gitbucket.war /opt/gitbucket.war
```

Создание символической ссылки на директорию /gitbucket

```
RUN ln -s /gitbucket /root/.gitbucket
```

Создание хранилища с базами данных и конфигурационными файлами приложения

```
VOLUME /gitbucket
```

Открытие портов

```
# Port for web page and Port for SSH access to git repository (Optional)
EXPOSE 8080 29418
```

Задание первоначальной команды

```
CMD ["sh", "-c", "java -jar /opt/gitbucket.war"]
```

2.2. Kanboard

Данный файл брался с официального git репозитория проекта Kanboard.

Аргумент, который указывает на требующуюся архитектору процессора

```
ARG BASE_IMAGE_ARCH="amd64"
```

Образ базируется на amd64/alpine:3.12

```
FROM ${BASE_IMAGE_ARCH}/alpine:3.12
```

Указание хранилищ для плагинов, баз данных, секретные ключи

```
VOLUME /var/www/app/data
VOLUME /var/www/app/plugins
VOLUME /etc/nginx/ssl
```

Открытие портов

```
EXPOSE 80 443
```

Установка зависимостей

```
RUN apk --no-cache --update add \
    openssl unzip nginx bash ca-certificates s6 curl ssmtp mailx php7 php7-phar php7-curl \
    php7-fpm php7-json php7-zlib php7-xml php7-dom php7-ctype php7-opcache php7-zip php7-iconv \
    php7-pdo php7-pdo_mysql php7-pdo_sqlite php7-pdo_pgsql php7-mbstring php7-session php7-bcmath \
    php7-gd php7-mcrypt php7-openssl php7-sockets php7-posix php7-ldap php7-simplexml && \
    rm -rf /var/www/localhost && \
    rm -f /etc/php7/php-fpm.d/www.conf
```

Добавление всего git-репозитория в контейнер, а также отдельно папки docker/

```
ADD . /var/www/app
ADD docker/ /
```

Удаление конфигурационного файла и папки docker /var/www/app/docker

```
RUN rm /var/www/app/config.php
RUN rm -rf /var/www/app/docker && echo $VERSION > /version.txt
```

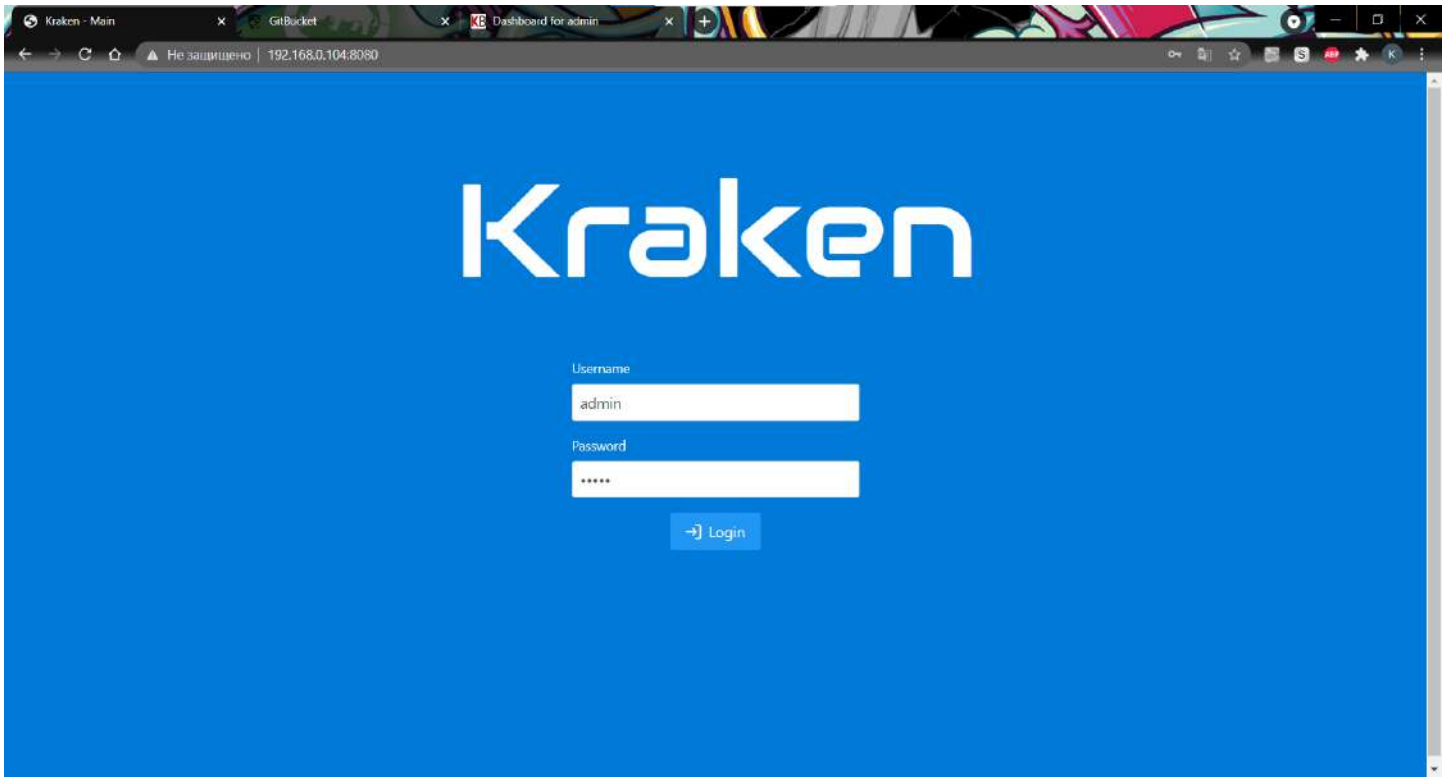
Указание входной точки

```
ENTRYPOINT ["/usr/local/bin/entrypoint.sh"]
```

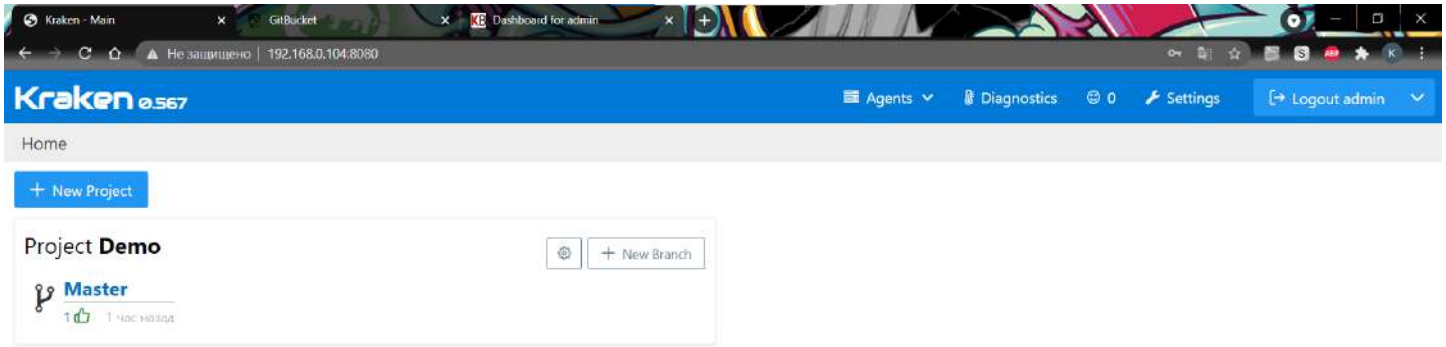
3. Пример настройки проекта

3.1. Kraken

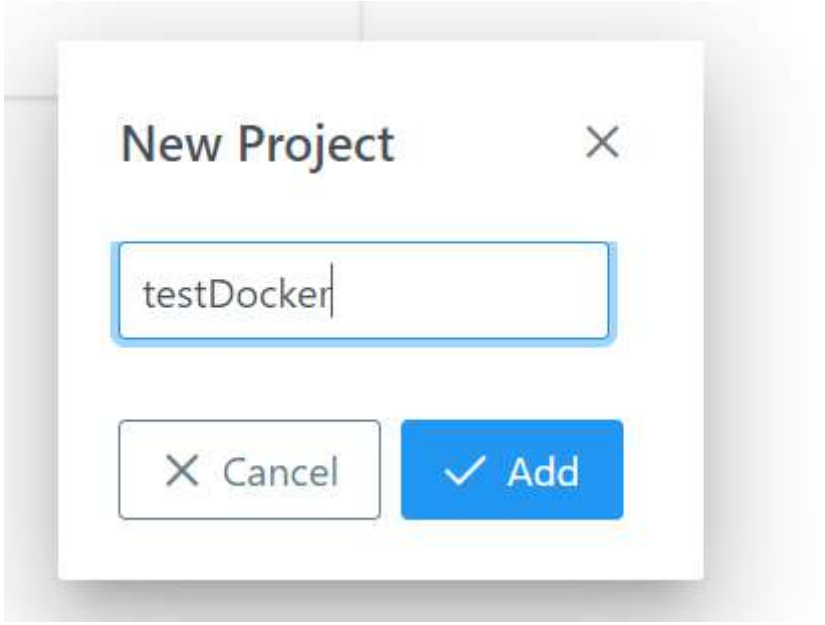
При переходе по адресу 192.168.0.104:8080 нас встречает начальная страница сервиса Kraken.



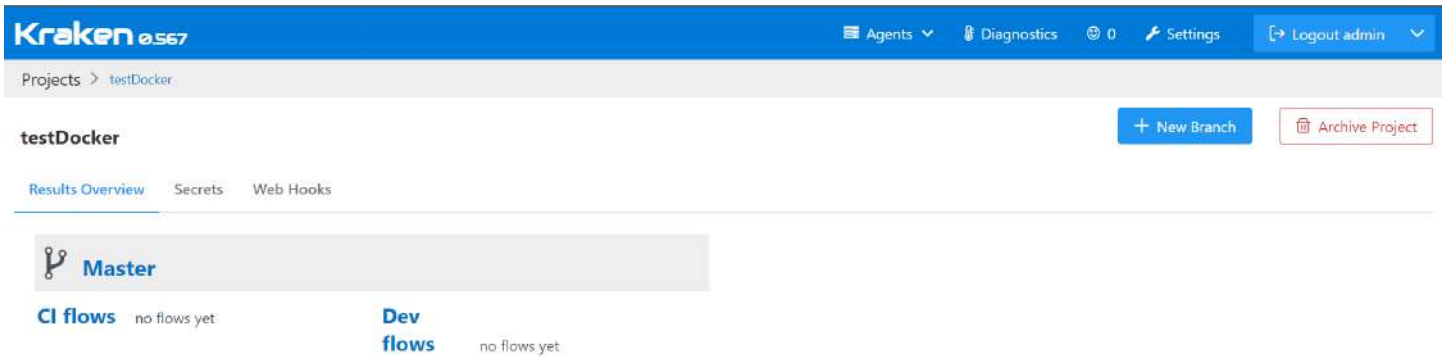
Нужно зайти под администратором (admin/admin).



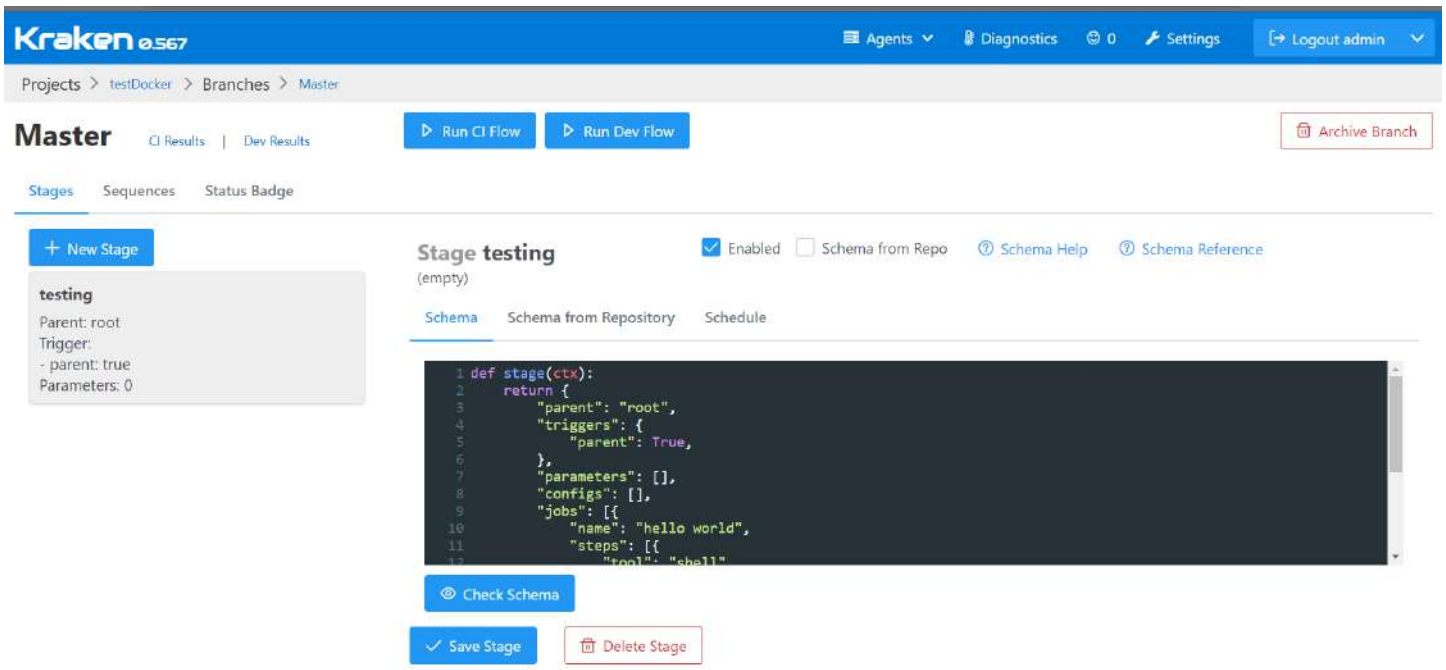
Создаем новый проект, нажимая кнопку New Project. Вводим название проекта testDocker и нажимаем Add. У нас создался новый проект.



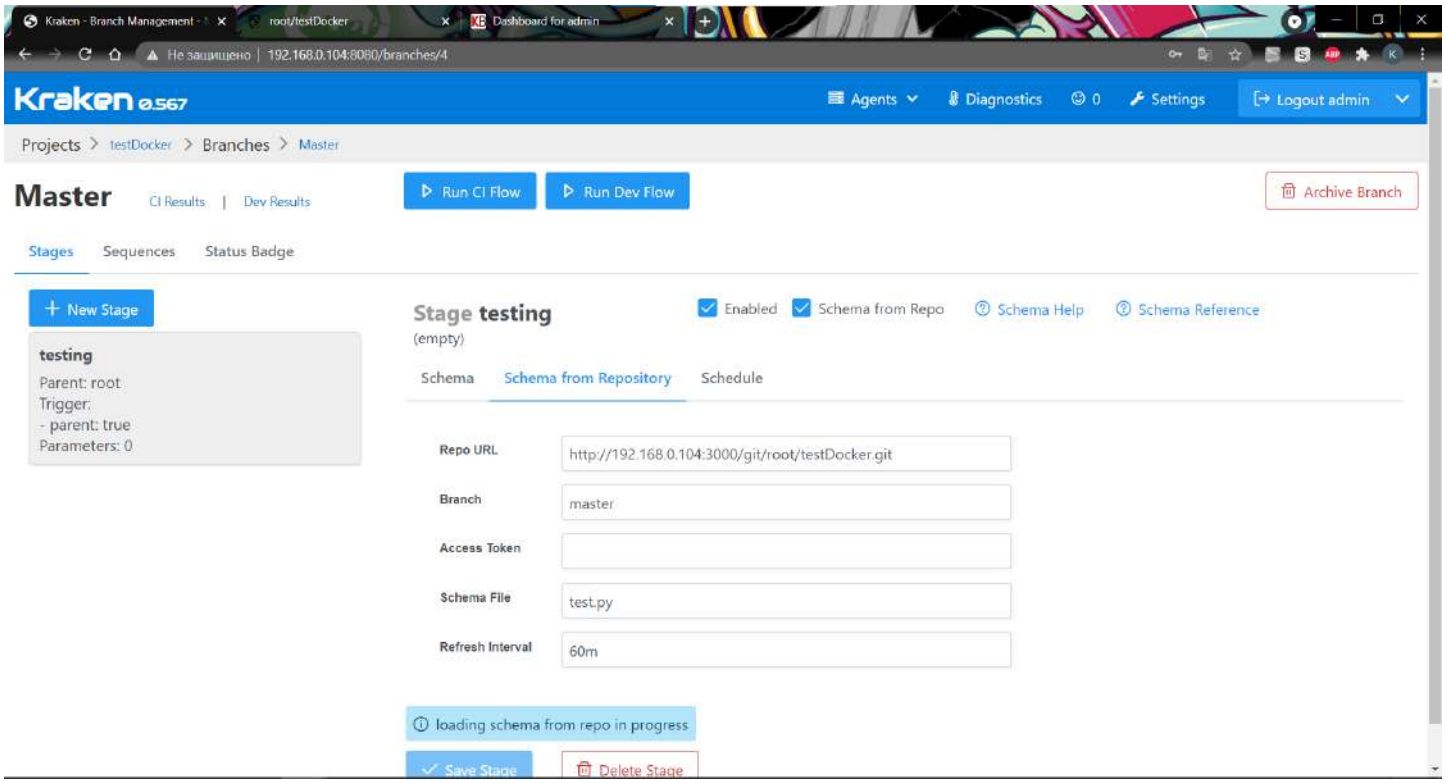
Добавляем новую ветку с помощью New Branch. Называем ее Master.



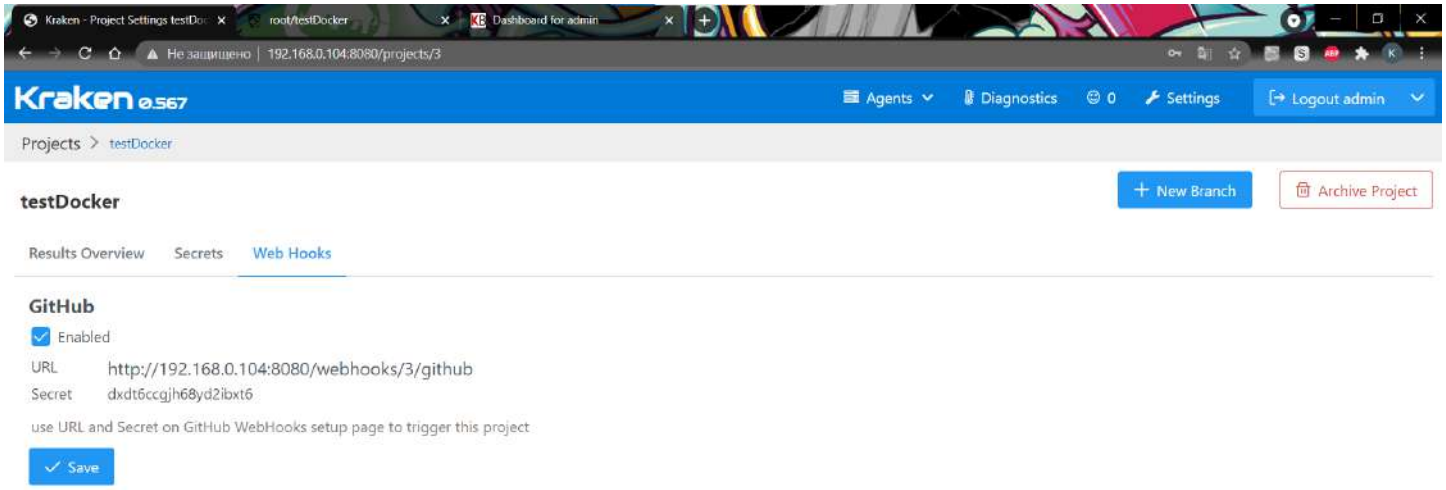
Щелкаем по созданной ветке и создаем там New Stage, которую называем testing. Дальше нам нужно внедрить свою схему, чтобы потом можно было проверять ее на своих данных в GitBucket’е.



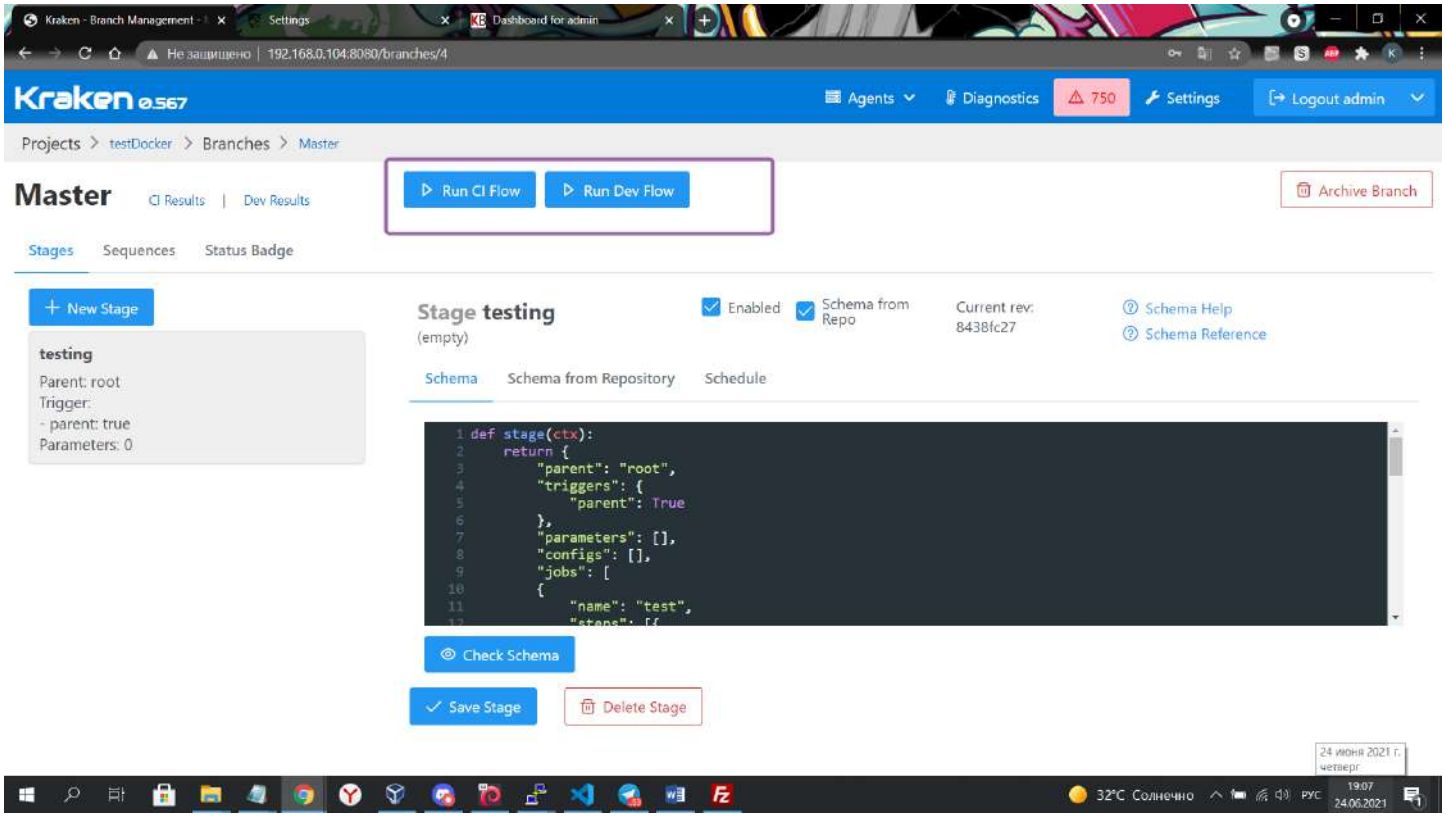
Добавляем свою схему из репозитория на GitBucket’е. И сохраняем. Теперь на вкладке Schema появилась схема из нашего репозитория.

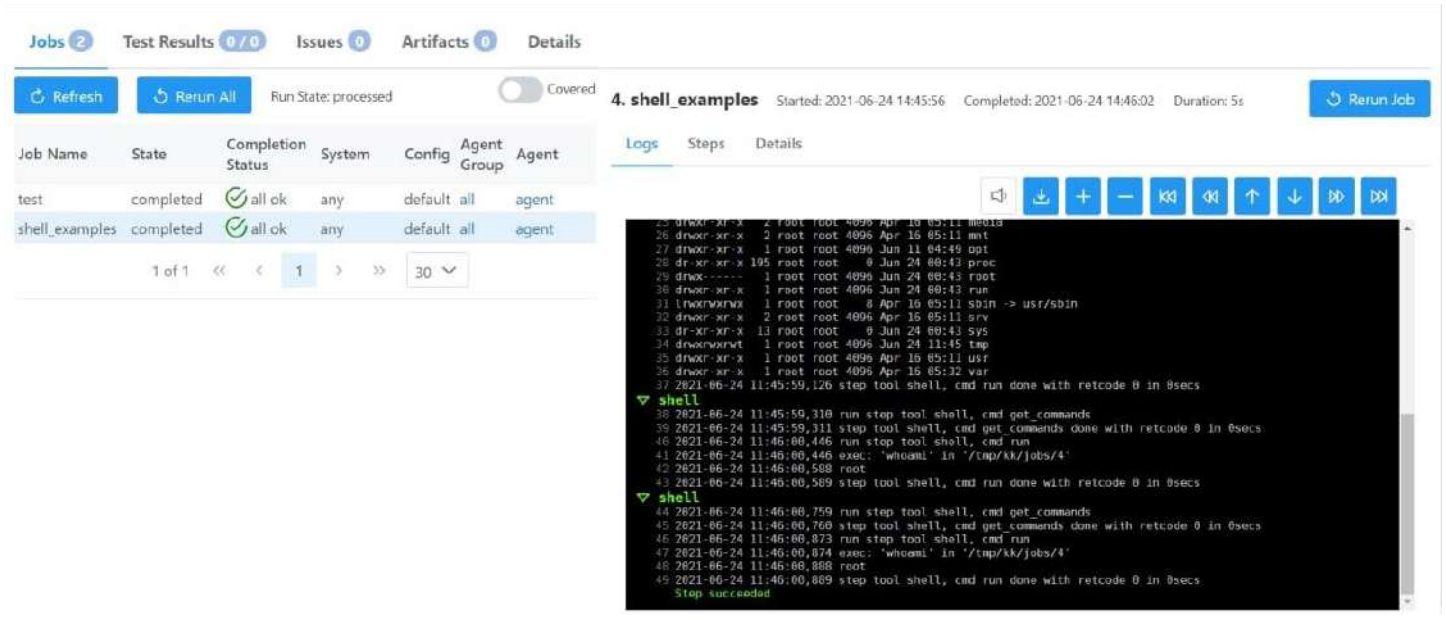


Возвращаемся к нашему проекту и заходим на вкладку Web Hooks. Там ставим флажок на Enable GitHub. Эти данные будут использоваться для интеграции GitBucket’а.



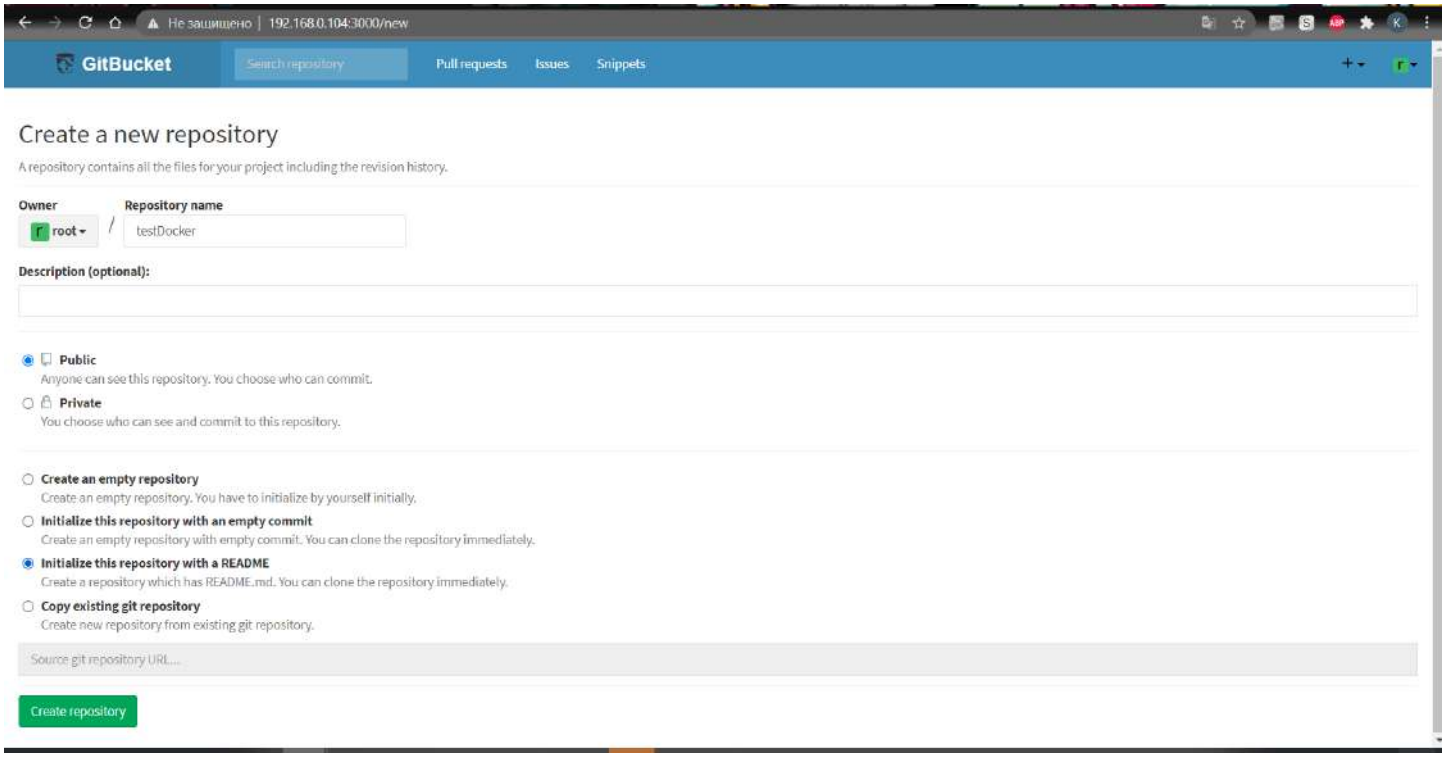
Когда успешно ввели webhooks, можно проверить систему. Выбираем нужную кнопку и запускаем тест, ждем когда он закончиться и смотрим результаты.


Kraken - Branch Management
Settings
Dashboard for admin
Не защищено | 192.168.0.104:8080/branches/4
Kraken 0.5.67
Agents
Diagnostics
750
Settings
Logout admin
Projects > testDocker > Branches > Master
Master
CI Results | Dev Results
Run CI Flow
Run Dev Flow
Archive Branch
Stages
Sequences
Status Badge
New Stage
testing
Parent: root
Trigger:
- parent: true
Parameters: 0
Stage testing
(empty)
Enabled
Schema from Repo
Current rev: 8438fc27
Schema Help
Schema Reference
Schema
Schema from Repository
Schedule
def stage(ctx):
return {
"parent": "root",
"triggers": {
"parent": True
},
"parameters": [],
"configs": [],
"jobs": [
{
"name": "test",
Check Schema
Save Stage
Delete Stage
32°C Солнечно
РУС
19:07
24.06.2021

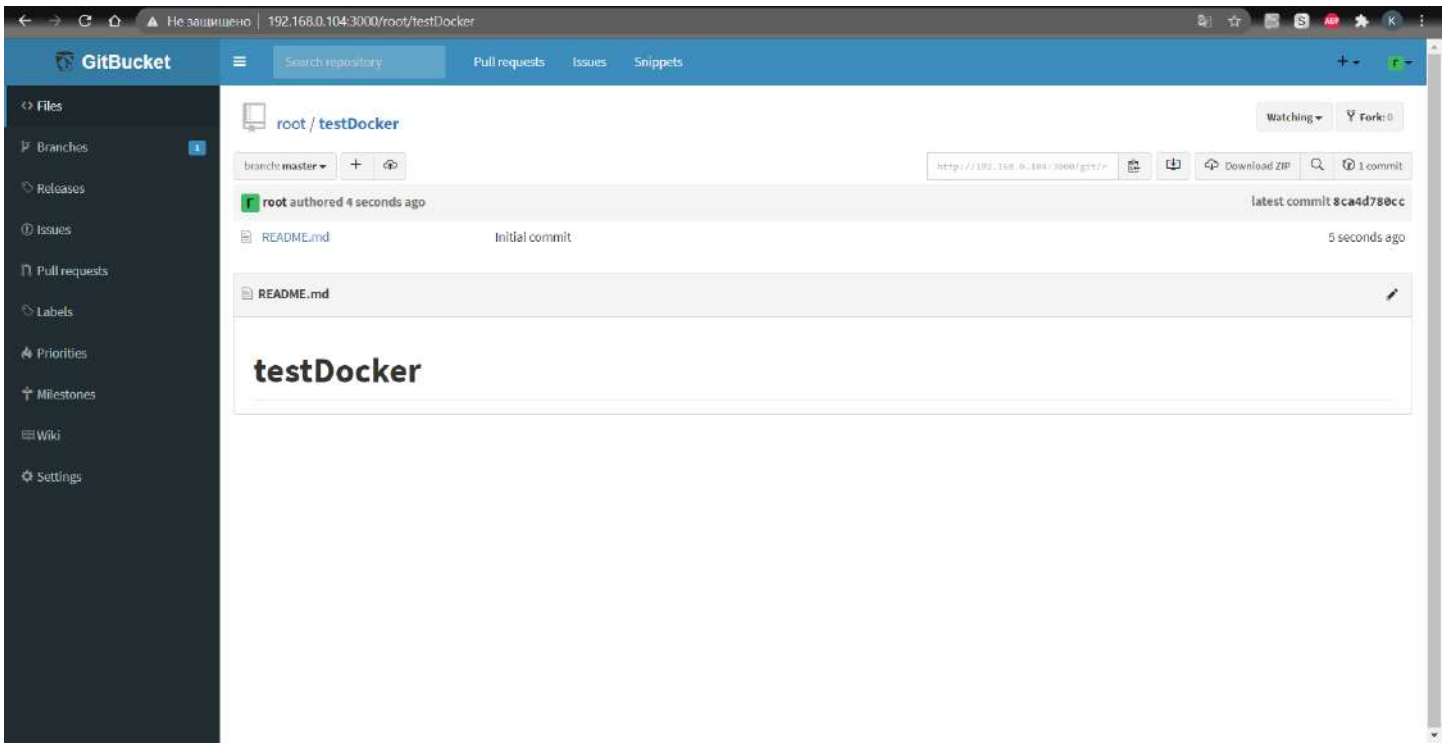

Jobs 2
Test Results 0 / 0
Issues 0
Artifacts 0
Details
Refresh
Rerun All
Run State: processed
Covered
Job Name
State
Completion Status
System
Config
Agent Group
Agent
test
completed
all ok
any
default
all
agent
shell_examples
completed
all ok
any
default
all
agent
1 of 1
30
4. shell_examples
Started: 2021-06-24 14:45:56
Completed: 2021-06-24 14:46:02
Duration: 5s
Rerun Job
Logs
Steps
Details
2021-06-24 11:45:59,126 step tool shell, cmd run done with retcode 0 in 0secs
shell
2021-06-24 11:45:59,310 run step tool shell, cmd get_commands
2021-06-24 11:45:59,311 step tool shell, cmd get_commands done with retcode 0 in 0secs
2021-06-24 11:46:00,446 run step tool shell, cmd run
2021-06-24 11:46:00,446 exec: 'whoami' in '/tmp/kk/jobs/4'
2021-06-24 11:46:00,588 root
2021-06-24 11:46:00,589 step tool shell, cmd run done with retcode 0 in 0secs
shell
2021-06-24 11:46:00,759 run step tool shell, cmd get_commands
2021-06-24 11:46:00,760 step tool shell, cmd get_commands done with retcode 0 in 0secs
2021-06-24 11:46:00,873 run step tool shell, cmd run
2021-06-24 11:46:00,874 exec: 'whoami' in '/tmp/kk/jobs/4'
2021-06-24 11:46:00,888 root
2021-06-24 11:46:00,889 step tool shell, cmd run done with retcode 0 in 0secs
Stop succeeded

3.2. GitBucket

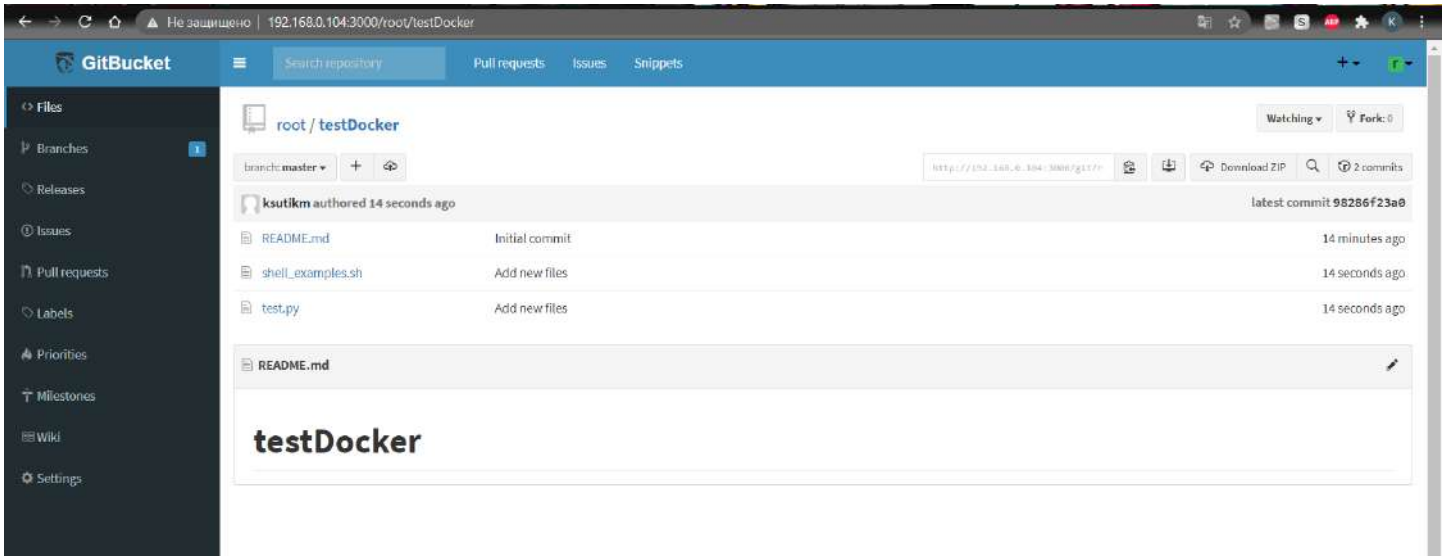
При переходе по адресу 192.168.0.104:3000 нас встречает главная страница сервиса **GitBucket**. Авторизуемся под root'ом (root/root). При входе под root'ом можно создать пользователей, но для наших целей будет достаточно создать репозиторий под пользователем root. Для этого нажимаем "+" в правом верхнем углу и выбираем New repository. Выбираем имя testDocker и нажимаем кнопку Create repository.



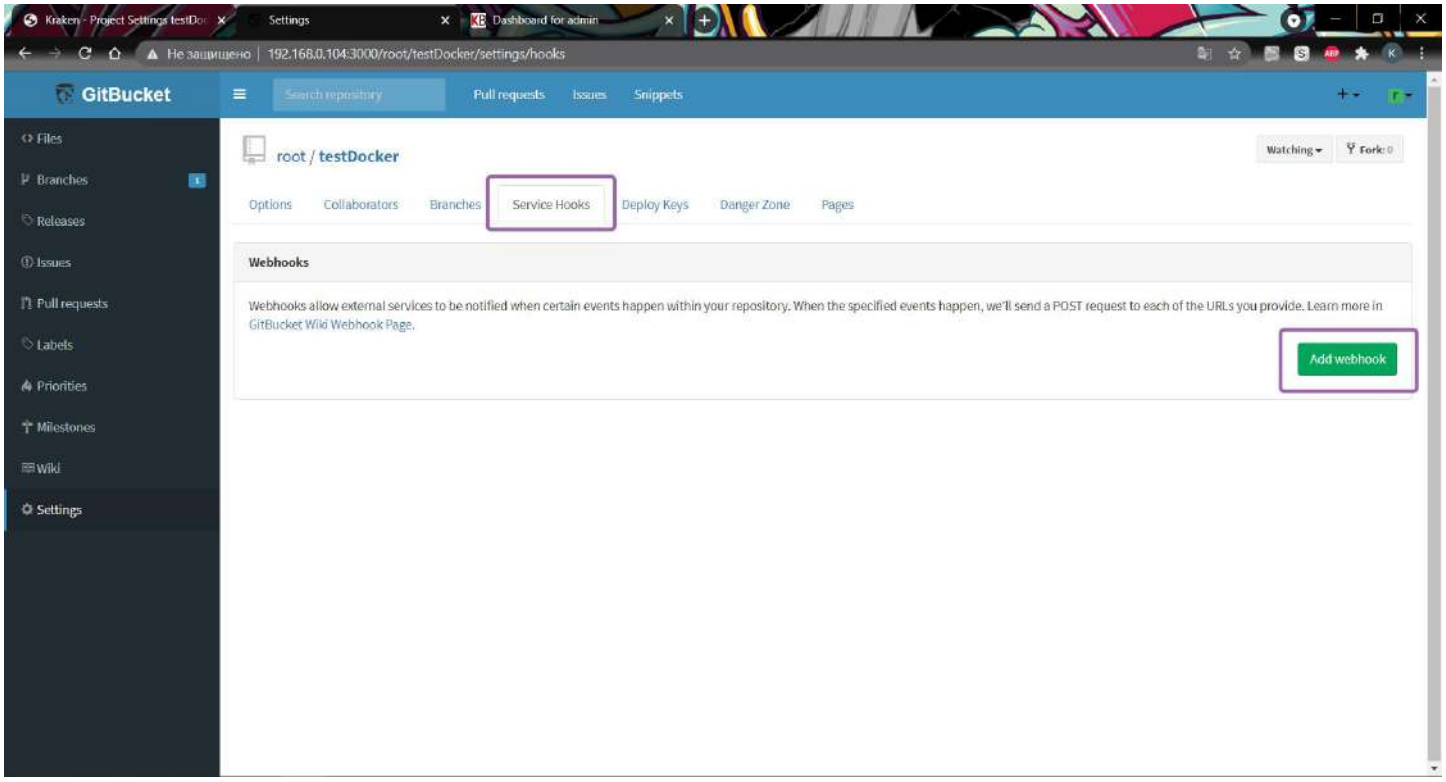
После создания репозитория будет показана инструкция, с помощью которой можно будет создать git репозиторий с папкой на хосте ("git remode add …").



Добавляем нужные файлы в репозиторий. Файл **test.py** – схема для Kraken'а. shell_examples.sh – скрипт, на котором будем проверять работоспособность Kraken'а.



Для соединения с Kraken используем webhooks. Заходим в настройки репозитория (Settings) и выбираем раздел Service Hooks и нажимаем кнопку Add webhook.

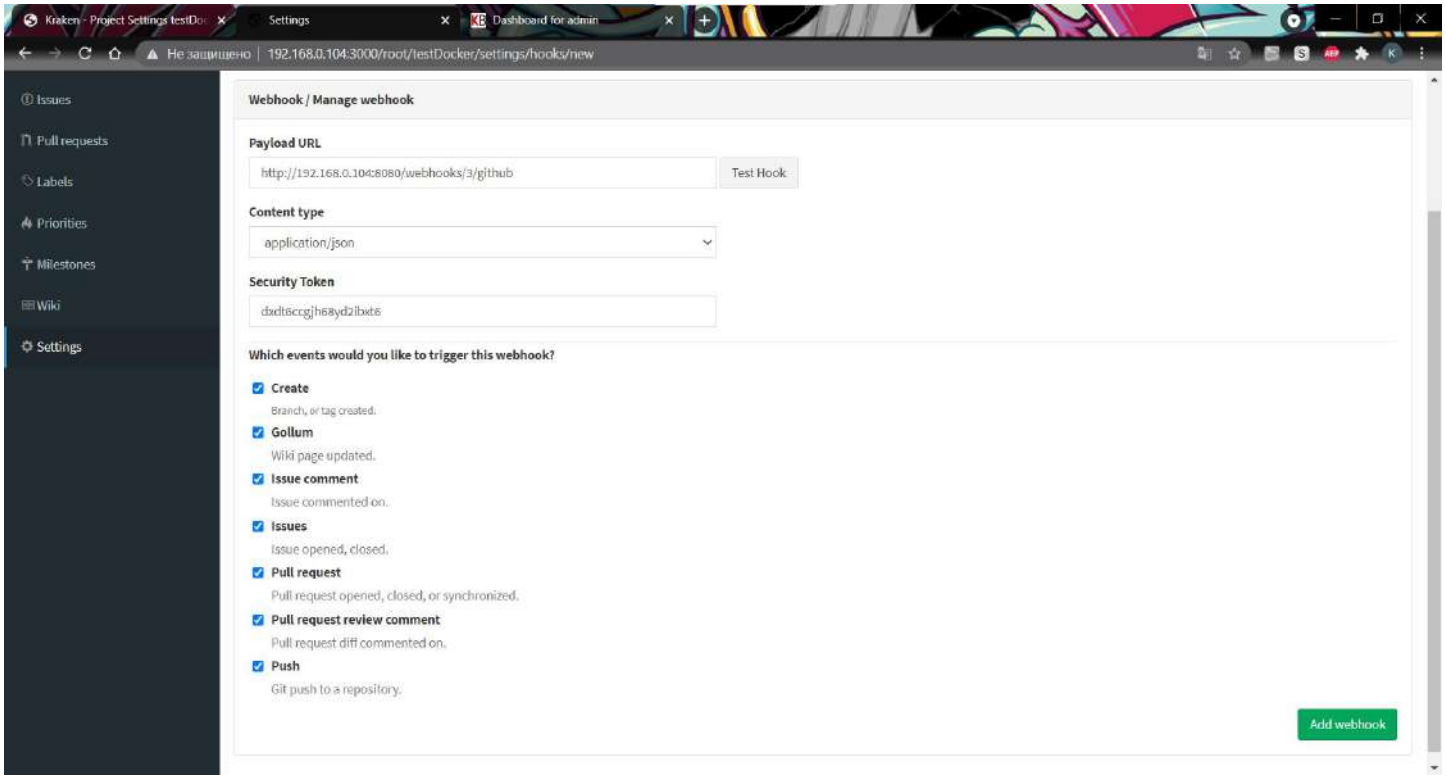


В поле Payload URL нужно вести адрес вебхука Kraken.

В поле Content type выбираем формат json.

В поле Security Token вводим secret из Kraken’a.

Выбираем все флажки ниже.



После этого Kraken и GitBucket будут связаны.

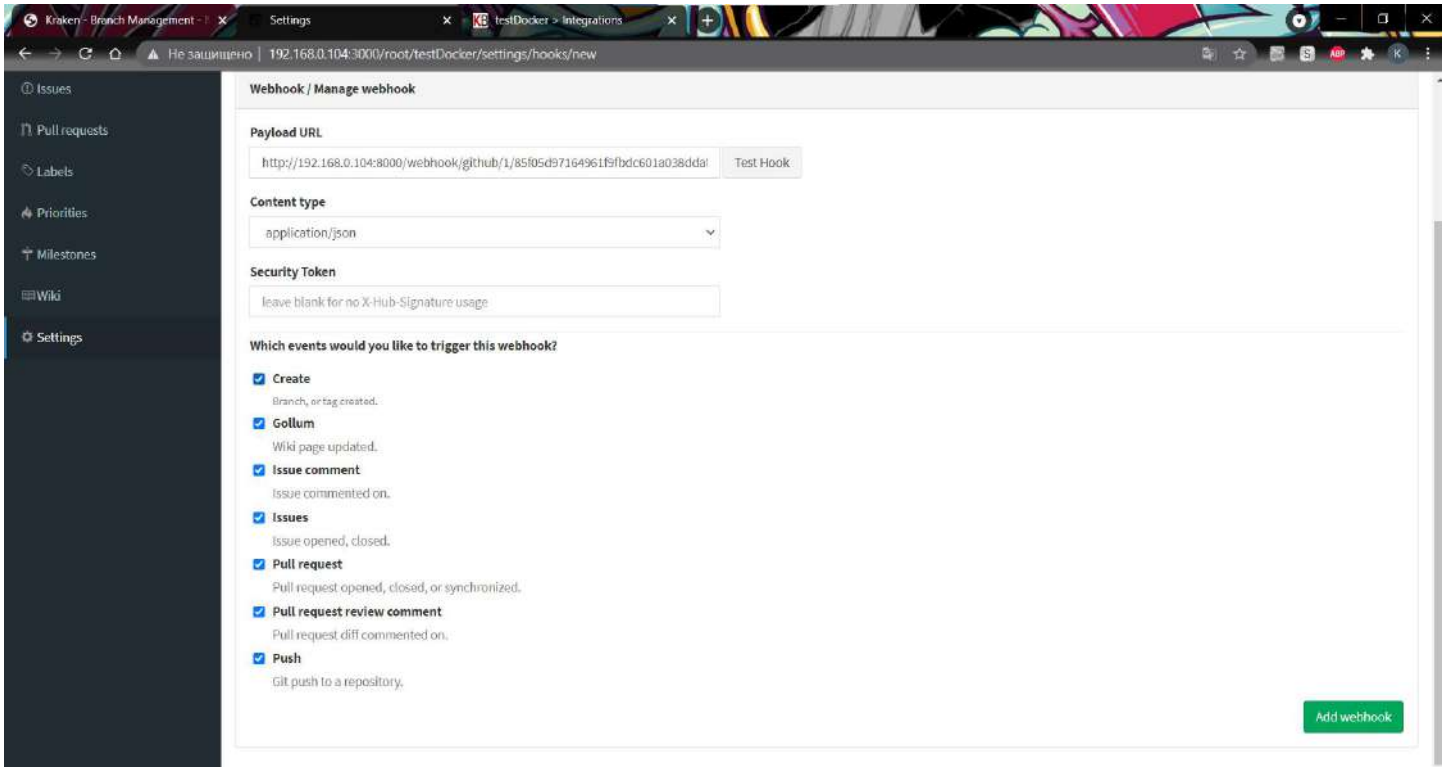
Добавляем еще один вубхук для Kanboard.

В поле Payload URL нужно вести адрес вебхука из настроек Kanboard.

В поле Content type выбираем формат json.

В поле Security Token вводить ничего не нужно.

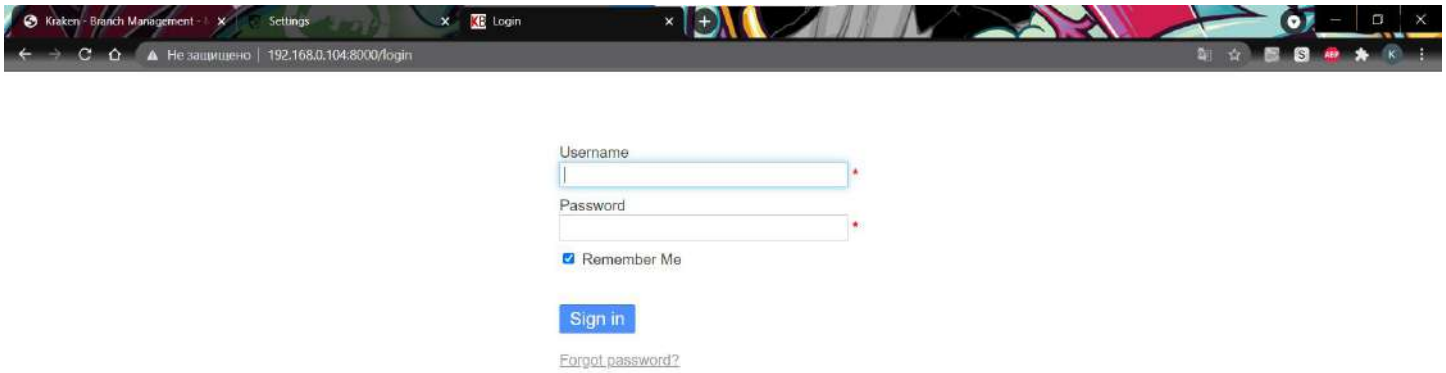
Выбираем все флажки ниже.

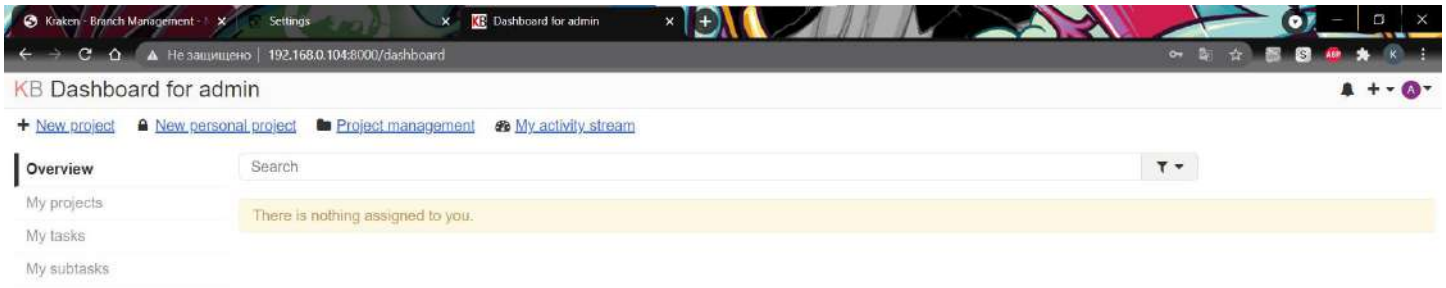


После этого Kanboard и GitBucket будут связаны.

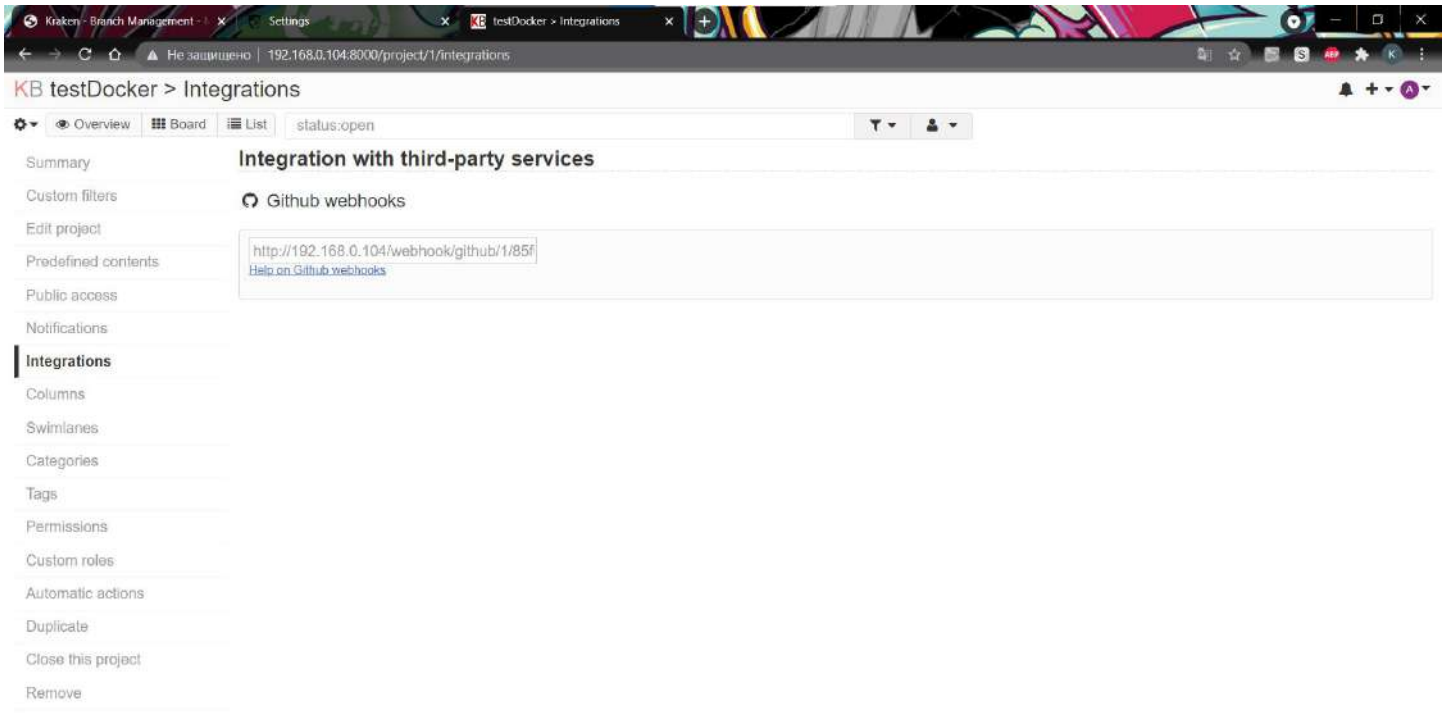
3.3. Kanboard

Перейдя по адресу 192.168.0.104:8000, появится страница с логином **Kanboard.** Входим под админом (admin/admin).





В верхнем левом углу нажимаем на New project, после чего вводим имя проекта и сохраняем. Находим на главной странице созданного проекта кнопку Integrations. Там указана ссылка на вебхук для сервиса GitBucket.

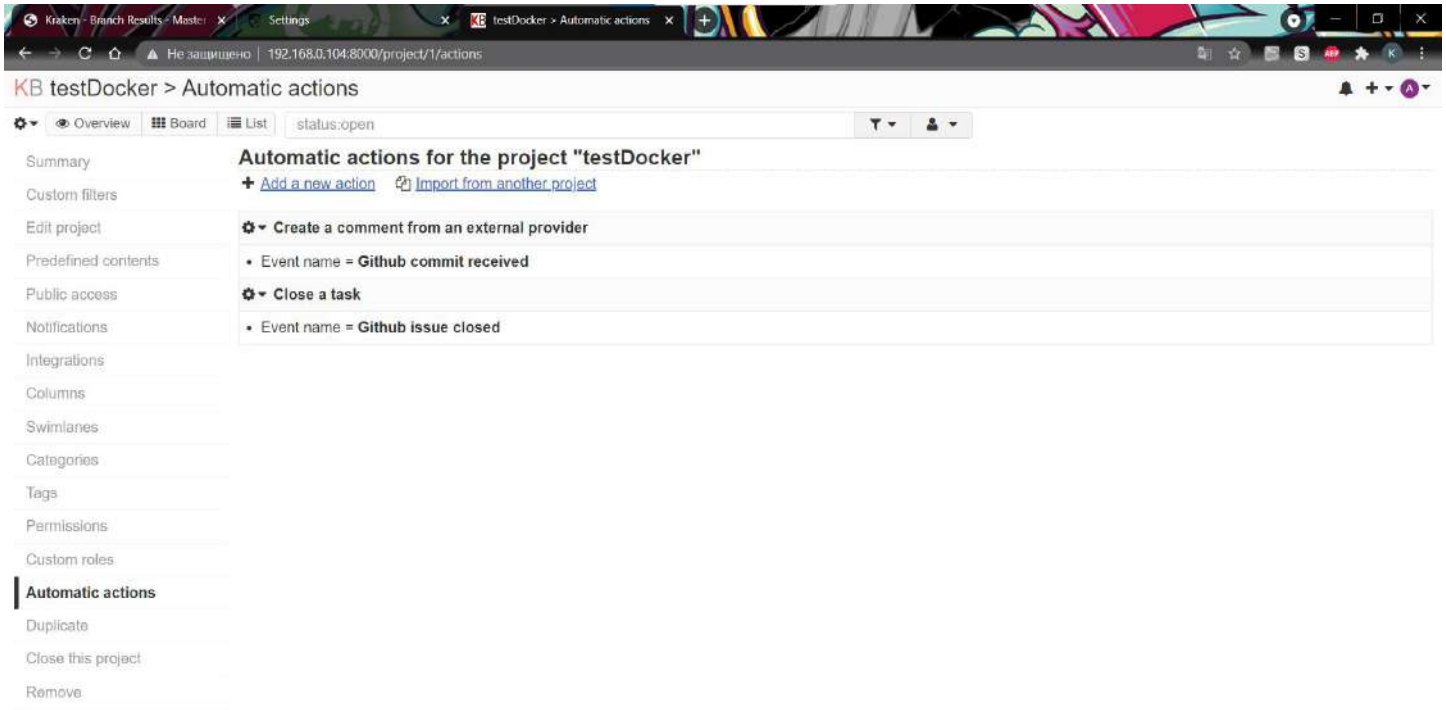


Далее для интергации нажимаем кнопку Automatic actions, и там добавим несколько пар действий (Kanboard-GitBucket).

Create a task from external provider – Github issue opened (задача в Kanboard открывается при создании issue в GitBucket).

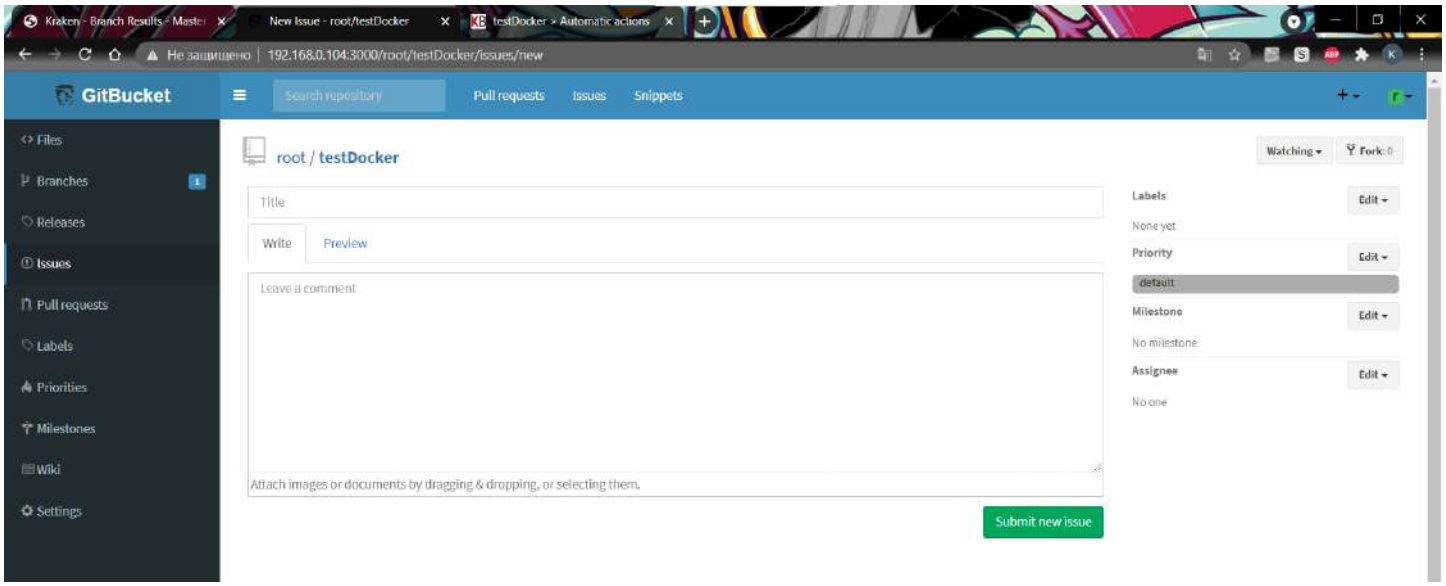
Close a task – Github commit received (задача в Kanboard закрывается при коммите с указанием номера задачи, которую нужно закрыть). Пример: git commit –m “fix code #2”.

Аналогично можно создать еще несколько действий, но для демонстрации интеграции сервисов достаточно этих двух пар.

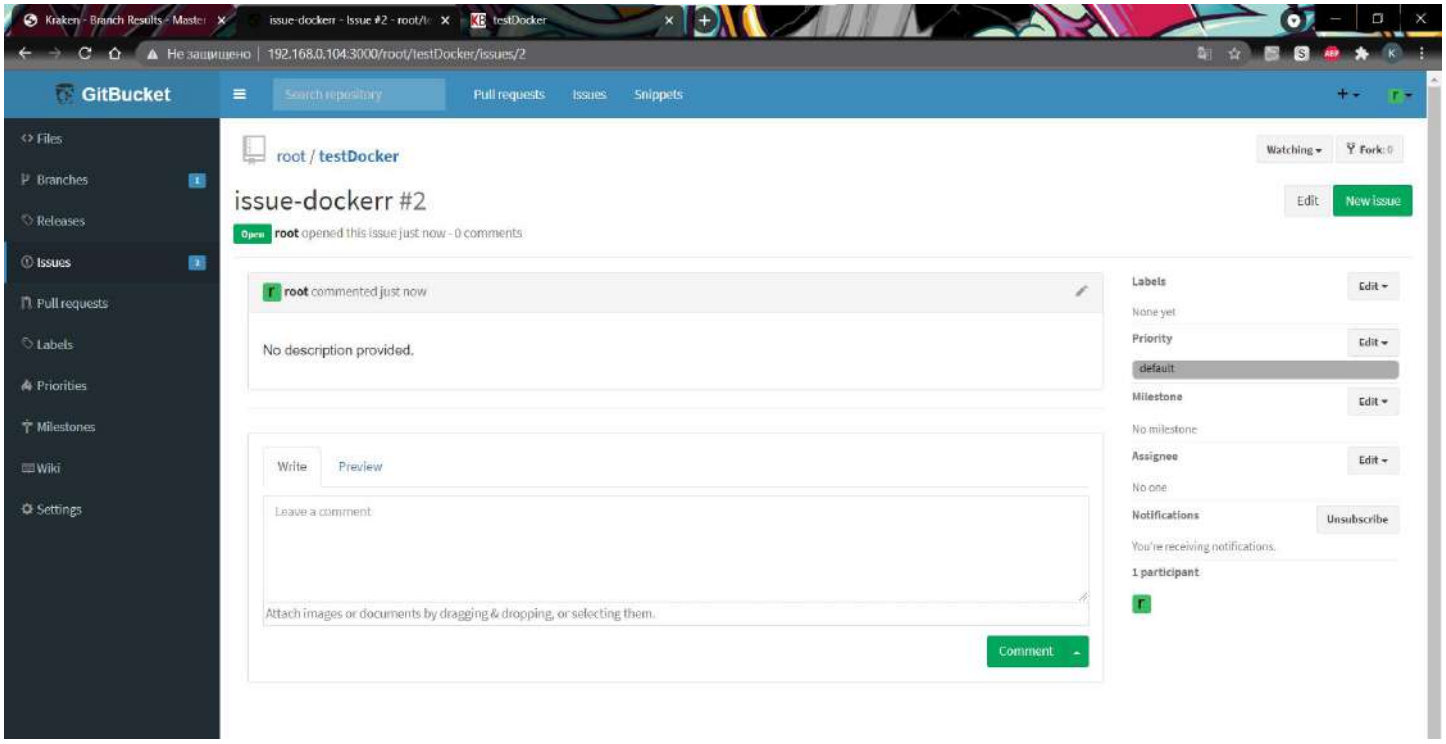


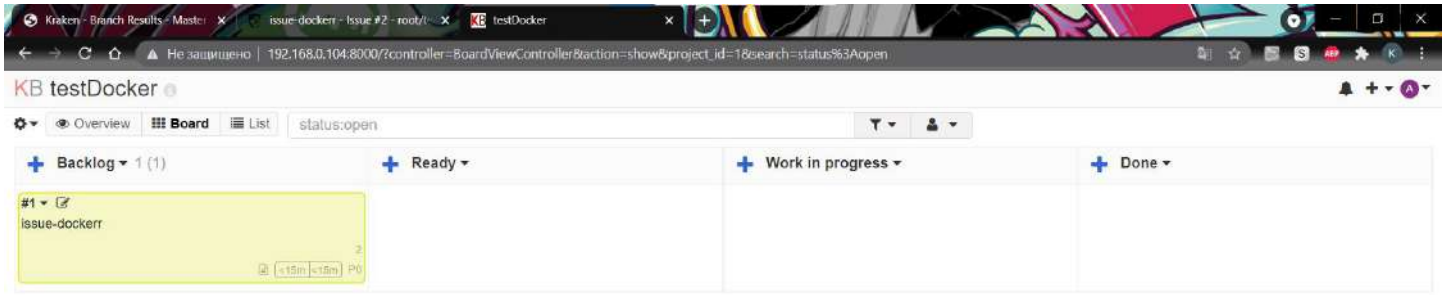
Демонстрация работы двух сервисов (Kanboard и GitBucket):

В сервисе GitBucket заходим на вкладку Issues и нажимаем на кнопку New issue.



Заполняем поля, для теста хватит всего лишь названия. Затем нажимаем на Submit new issue, после чего заходим на страницу проекта Kanboard и видим задачу с нашим заголовком (issue-dockerr).





Делаем изменения в наших файлах, которые лежат на GitBucket.

```
user@playground:~/unix_exam/testDocker$ git add .
user@playground:~/unix_exam/testDocker$ git commit -m "fix code #1"
[master 447495d] fix code #1
 1 file changed, 1 deletion(-)
user@playground:~/unix_exam/testDocker$ git push
Username for 'http://192.168.0.104:3000': root
Password for 'http://root@192.168.0.104:3000':
Перечисление объектов: 5, готово.
Подсчет объектов: 100% (5/5), готово.
При сжатии изменений используется до 4 потоков
Сжатие объектов: 100% (3/3), готово.
Запись объектов: 100% (3/3), 342 bytes | 342.00 KiB/s, готово.
Всего 3 (изменения 1), повторно использовано 0 (изменения 0)
remote: Resolving deltas: 100% (1/1)
remote: Updating references: 100% (1/1)
To http://192.168.0.104:3000/git/root/testDocker.git
   8438fc2..447495d  master -> master
user@playground:~/unix_exam/testDocker$
```

В сервисе Kanboard задача после коммита была закрыта.