

Ярославский государственный университет им. П. Г. Демидова

Факультет информатики и вычислительной техники

Отчёт

Настройка и интеграция сервисов **Gitbucket/Kanboard/Jenkins.**

Работу выполнил:

Студент группы ИВТ-41 БО

И.С. Гусев

« ____ » _____ 2020 г.

Jenkins:

При переходе по адресу 192.168.1.49:8080 нас встречает начальная страница сервиса **Jenkins** рис (1). (Указан адрес виртуальной машины, с которой запускаются контейнеры)

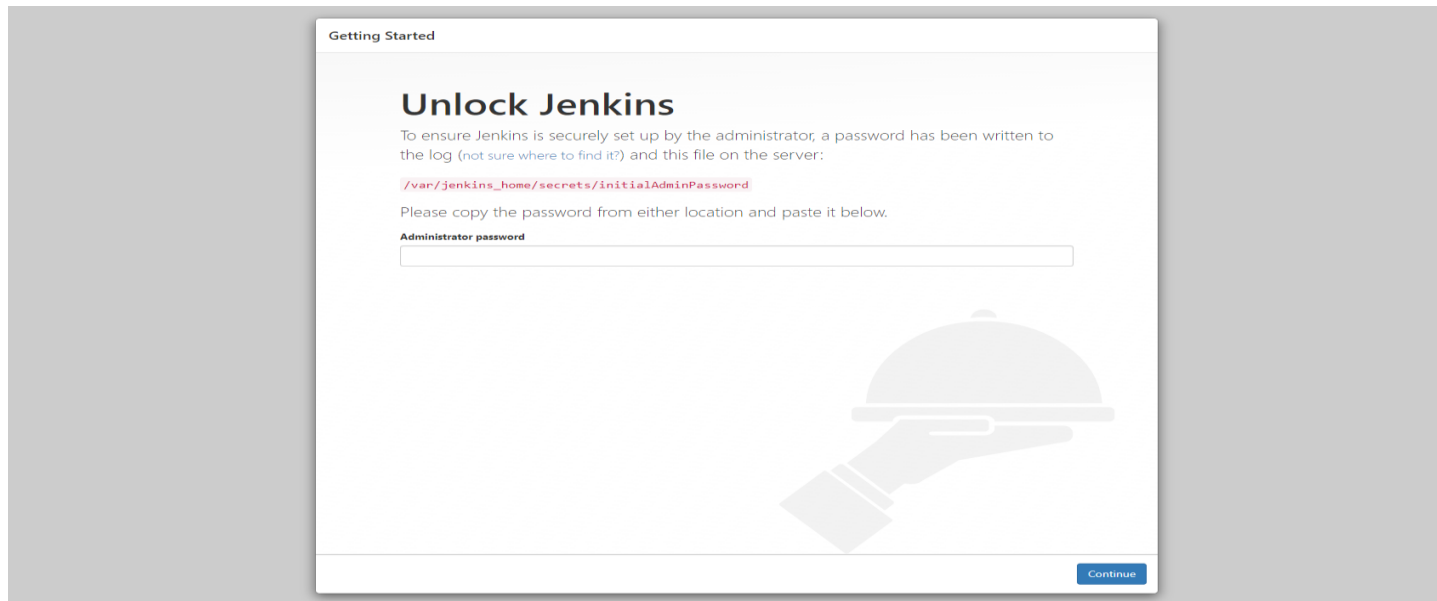


Рис.1

На данном этапе требуется ввести пароль администратора из папки secrets. На хосте эта папка доступна в хранилище, которое связано с нужной директорией в докер-контейнере. Этот файл можно найти по следующему пути /tmp/initialAdminPassword, ввести команду `sudo cat /tmp/initialAdminPassword`, скопировать и вставить вывод.

Далее нужно выбрать установить предложенные плагины, либо выбрать их самому. Для корректной работы сервиса, а также соединения с gitbucket, я выбрал установку всех предложенных плагинов(рис.2).

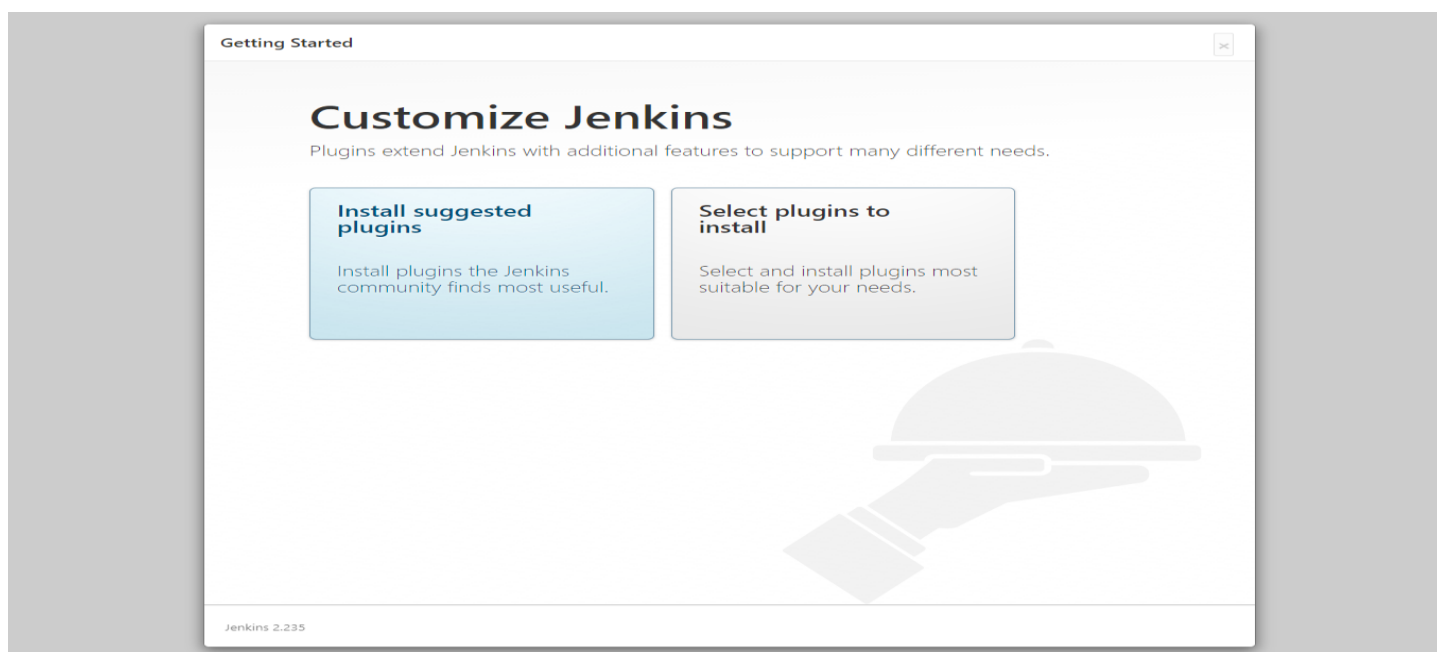


Рис.2

После завершения установки плагинов, нас встречает окно регистрации первого пользователя. Нужно ввести логин, пароль, Ф.И.О., а также адрес электронной почты. После регистрации, уже будучи авторизованными, откроется главное окно сервиса(рис.3).

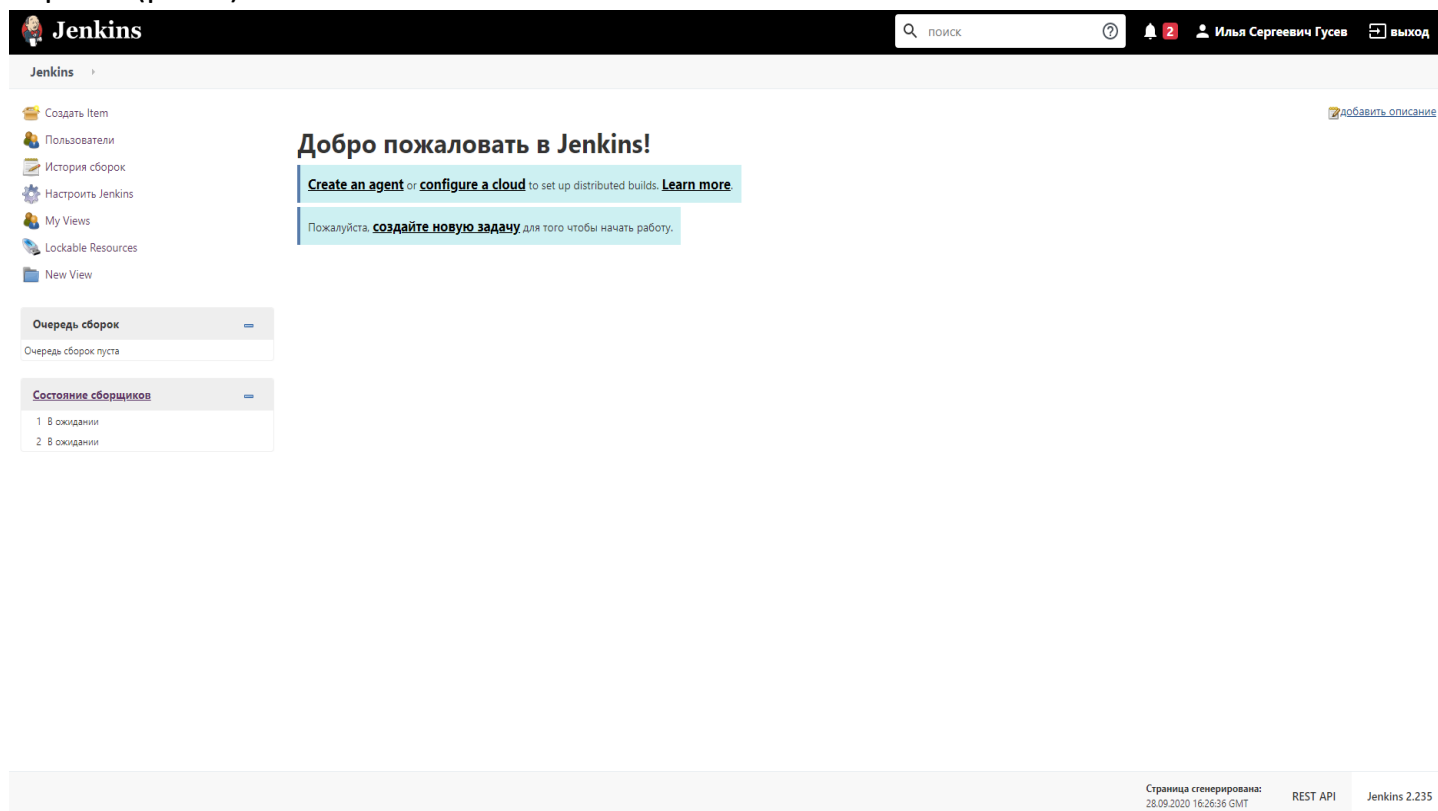


Рис.3

Далее нажимаем на кнопку “создать новую задачу”, вводим имя задачи и выбираем тип “Pipeline”.

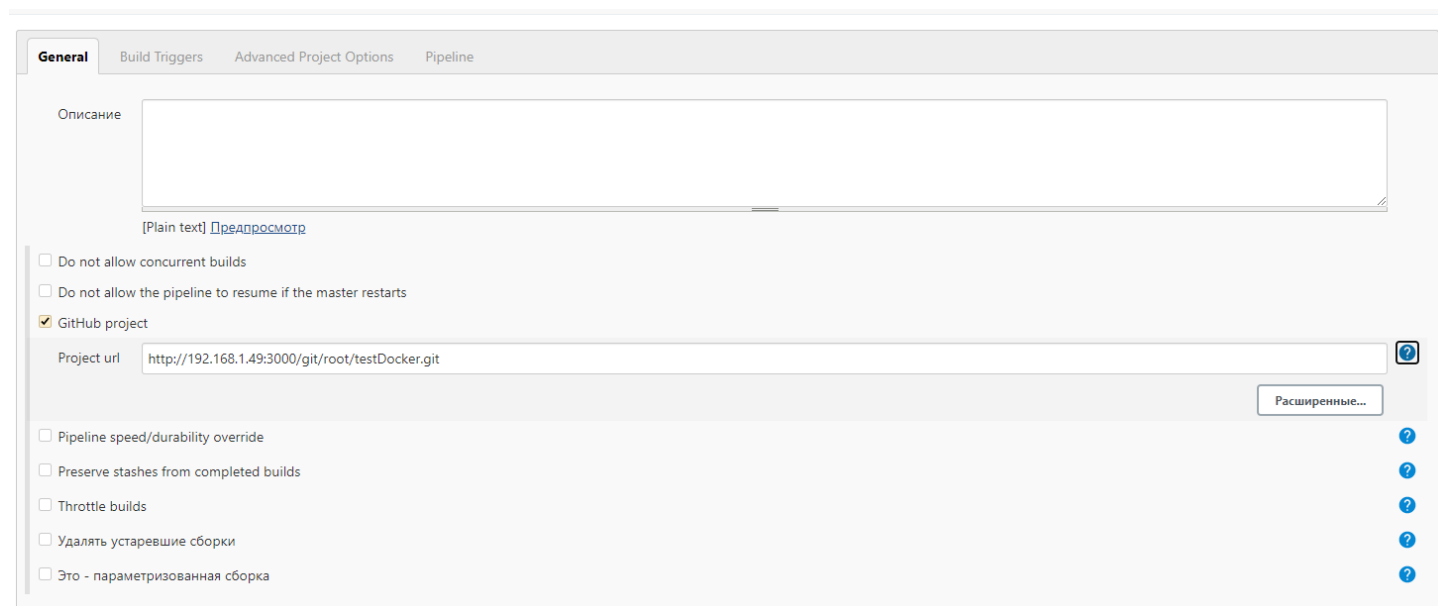


Рис.4

В данном разделе настроек(рис.4) можно задать описание пайплайна, а также нужно выбрать в меню Github project, и ввести url-адрес.

В данном случае это 192.168.1.49:3000/git/root/testDocker.git

В следующем разделе настроек(рис.5) нашей задачи нужно выбрать Build Trigger. Это то действие, по которому Jenkins будет собирать заданный ей проект. Выбираем пункт “Опрашивать SCM об изменениях” и в поле Расписание ввести (* * * * *), это означает, что Jenkins будет каждую минуту проверять случилось ли изменение в нашем git-репозитории.

The screenshot shows the 'Build Triggers' section of the Jenkins configuration interface. It includes several checkboxes for different build triggers, with 'Poll SCM for changes' selected. A cron schedule field is set to '* * * * *'. A warning message is displayed below the schedule field, advising that the current schedule triggers a build every minute. Below the warning, there is a button labeled 'Расширенные...' (Advanced...). The section is titled 'Build Triggers' and is part of a larger configuration page that also includes 'Advanced Project Options' and 'Pipeline' sections.

Build Triggers

- ☐ Build after other projects are built
- ☐ Запускать периодически
- ☐ GitHub hook trigger for GITScm polling
- ☒ Опрашивать SCM об изменениях

Расписание: * * * * *

⚠ Do you really mean "every minute" when you say "* * * * *"? Perhaps you meant "H * * * *" to poll once per hour

Последний запланированный запуск: Monday, September 28, 2020 6:32:29 PM GMT; следующий запланированный запуск: Monday, September 28, 2020 6:32:29 PM GMT.

☐ Ignore post-commit hooks

☐ Приостановить сборки

☐ Пауза перед сборкой

☐ Trigger builds remotely (e.g., from scripts)

Advanced Project Options

Расширенные...

Pipeline

Рис.5

В последнем разделе настроек(рис.6) в поле “Definition” указываем “Pipeline script from SCM”, это означает что мы хотим использовать пайплайн, который находится в SCM-сервисе.

В качестве SCM-сервиса выбираем Git, после чего нужно ввести url git-репозитория в котором лежит нужный скрипт, изначальное имя которого Jenkinsfile (можно поменять).

Осталось нажать кнопку сохранить.

Pipeline

Definition Pipeline script from SCM

SCM Git

Repositories

Repository URL

Credentials [Add](#)

[Расширенные...](#)

[Add Repository](#)

Branches to build

Branch Specifier (blank for 'any')

[Add Branch](#)

Просмотрщик репозитория (Автоматически)

Additional Behaviours [Добавить](#)

Script Path

Lightweight checkout ☒

[Pipeline Syntax](#)

[Сохранить](#) [Применить](#)

Рис.6

Далее будет показано то, как выглядит успешная сборка.

Jenkins

Jenkins > testdocker >

[Back to Dashboard](#)
[Status](#)
[Changes](#)
[Собрать сейчас](#)
[Удалить Pipeline](#)
[Настройки](#)
[Full Stage View](#)
[GitHub](#)
[Rename](#)
[Pipeline Syntax](#)
[Лог опроса Git](#)

Pipeline testdocker

[Recent Changes](#)

Stage View

Average stage times:
(Average full run time: ~18s)

| | Declarative: Checkout SCM | Build | Test | Deploy |
|---------|---------------------------|-------|-------|--------|
| Average | 1s | 2s | 391ms | 443ms |
| #1 | 1s | 2s | 391ms | 443ms |

#1 Sep 28 22:19 No Changes

Постоянные ссылки

- [Последняя сборка \(#1\). 7 минут 3 секунды назад](#)
- [Последняя стабильная сборка \(#1\). 7 минут 3 секунды назад](#)
- [Последняя успешная сборка \(#1\). 7 минут 3 секунды назад](#)
- [Last completed build \(#1\). 7 минут 3 секунды назад](#)

История сборок

[тренд](#)

#1 28.09.2020 19:19

[Atom feed для всех](#) [Atom feed для неудачных](#)

Рис.7

Также прилагается log данной сборки.

Back to Project

Status

Changes

Console Output

View as plain text

Edit Build Information

Delete build #1

for onpoca

Git Build Data

No Tags

Restart from Stage

Replay

Pipeline Steps

Workspaces

Timestamps

System clock time

Use browser timezone

Elapsed time

None

Вывод на консоль

started by an SCM change

obtained Jenkinsfile from git <http://192.168.1.49:3000/git/root/testDocker.git>

Running in Durability level: MAX_SURVIVABILITY

[Pipeline] Start of Pipeline

[Pipeline] node

Running on Jenkins in /var/jenkins_home/workspace/testdocker

[Pipeline] {

[Pipeline] stage

[Pipeline] { (Declarative: Checkout SCM)

[Pipeline] checkout

The recommended git tool is: NONE

No credentials specified

Cloning the remote Git repository

Cloning repository <http://192.168.1.49:3000/git/root/testDocker.git>

> git init /var/jenkins_home/workspace/testdocker # timeout=10

Fetching upstream changes from <http://192.168.1.49:3000/git/root/testDocker.git>

> git --version # timeout=10

> git --version # 'git version 2.20.4'

> git fetch --tags --force --progress -- <http://192.168.1.49:3000/git/root/testDocker.git> +refs/heads/*:refs/remotes/origin/* # timeout=10

> git config remote.origin.url <http://192.168.1.49:3000/git/root/testDocker.git> # timeout=10

> git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10

Avoid second fetch

> git rev-parse refs/remotes/origin/master^{commit} # timeout=10

Checking out Revision 4071f8d9fee5316cfca39d2219c8f22de460d8d7 (refs/remotes/origin/master)

> git config core.sparsecheckout # timeout=10

> git checkout -f 4071f8d9fee5316cfca39d2219c8f22de460d8d7 # timeout=10

Commit message: "first commit"

First time build. Skipping changelog.

[Pipeline] }

[Pipeline] // stage

[Pipeline] withEnv

[Pipeline] {

[Pipeline] timestamps

[Pipeline] {

[Pipeline] stage

[Pipeline] { (Build)

[Pipeline] echo

22:19:22 Building..

[Pipeline] sh

22:19:23 + ruby rubyapp.rb

22:19:24 Hello world , I am created from a Docker container - Jenkins

[Pipeline] }

[Pipeline] // stage

[Pipeline] stage

[Pipeline] { (Test)

[Pipeline] echo

22:19:24 Testing..

[Pipeline] }

[Pipeline] // stage

[Pipeline] stage

[Pipeline] { (Deploy)

[Pipeline] echo

22:19:25 Deploying....

[Pipeline] }

[Pipeline] // stage

[Pipeline] }

[Pipeline] // timestamps

Finished: SUCCESS

Рис.8

В данном выводе можно увидеть, что все заданные в пайплайне этапы успешно пройдены.

На этапе Building происходила сборка простейшего ruby-приложения, с выводом строки “Hello World...”, это означает , что приложение работает.

Далее , чтобы проверить сборку на отрицательный результат я внес изменения в гит-репозитории, заменив строку puts на pts(ошибка синтаксиса), после чего автоматически запустилась сборка, со следующим логом.

Jenkins > testdocker > #2

Back to Project

Status

Changes

Console Output

View as plain text

Edit Build Information

Delete build '#2'

for onpoca

Git Build Data

No Tags

Restart from Stage

Replay

Pipeline Steps

Workspaces

Previous Build

Timestamps

View as plain text

System clock time

Use browser timezone

Elapsed time

None

Вывод на консоль

Started by an SCM change

Obtained Jenkinsfile from git <http://192.168.1.49:3000/git/root/testDocker.git>

Running in Durability level: MAX_SURVIVABILITY

[Pipeline] Start of Pipeline

[Pipeline] node

Running on Jenkins in /var/jenkins_home/workspace/testdocker

[Pipeline] {

[Pipeline] stage

[Pipeline] { (Declarative: Checkout SCM)

[Pipeline] checkout (hide)

The recommended git tool is: NONE

No credentials specified

> git rev-parse --is-inside-work-tree # timeout=10

Fetching changes from the remote Git repository

> git fetch --tags --force --progress -- <http://192.168.1.49:3000/git/root/testDocker.git> # timeout=10

Fetching upstream changes from <http://192.168.1.49:3000/git/root/testDocker.git>

> git --version # timeout=10

> git --version # 'git version 2.20.4'

> git fetch --tags --force --progress -- <http://192.168.1.49:3000/git/root/testDocker.git> +refs/heads/*:refs/remotes/origin/* # timeout=10

> git rev-parse refs/remotes/origin/master^{commit} # timeout=10

Checking out Revision de8e90836c0c4dc25c3165a3af7c6982f935883 (refs/remotes/origin/master)

> git config core.sparsecheckout # timeout=10

> git checkout -f de8e90836c0c4dc25c3165a3af7c6982f935883 # timeout=10

Commit message: "create bug for testing jenkins"

> git rev-list --no-walk 4071f8d9f0e5316cfca39d2219c8f22de460d8d7 # timeout=10

[Pipeline] }

[Pipeline] // stage

[Pipeline] withEnv

[Pipeline] {

[Pipeline] timestamps

[Pipeline] {

[Pipeline] stage

[Pipeline] { (Build)

[Pipeline] echo

22:34:10 Building..

[Pipeline] sh

22:34:11 + ruby rubyapp.rb

22:34:11 rubyapp.rb:1:in `main': undefined method `puts' for main:Object (NoMethodError)

22:34:11 Did you mean? puts

[Pipeline] }

[Pipeline] // stage

[Pipeline] stage

[Pipeline] { (Test)

Stage "Test" skipped due to earlier failure(s)

[Pipeline] }

[Pipeline] // stage

[Pipeline] stage

[Pipeline] { (Deploy)

Stage "Deploy" skipped due to earlier failure(s)

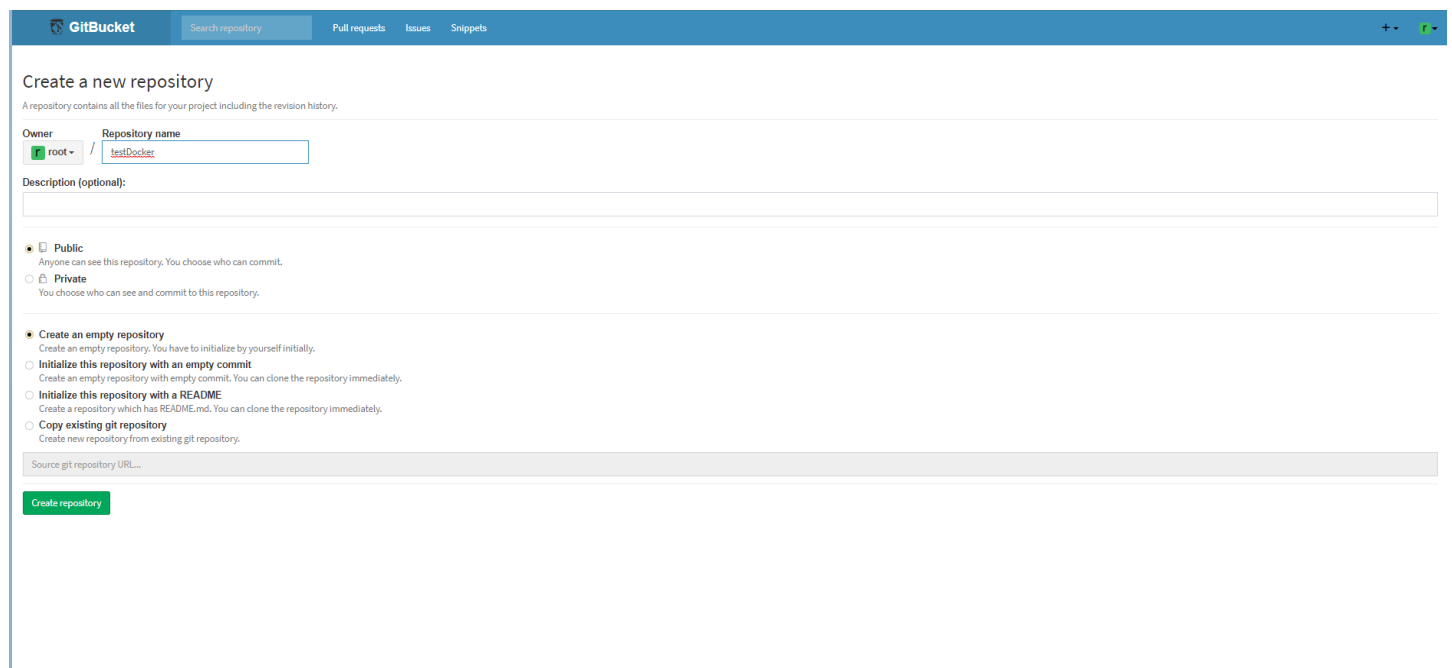
[Pipeline] }

Рис.9

На данном скриншоте видно, что на этапе Building произошла ошибка, после чего все остальные этапы были пропущены (так как произошла ошибка в этапе, запускаемом раньше них).

Gitbucket:

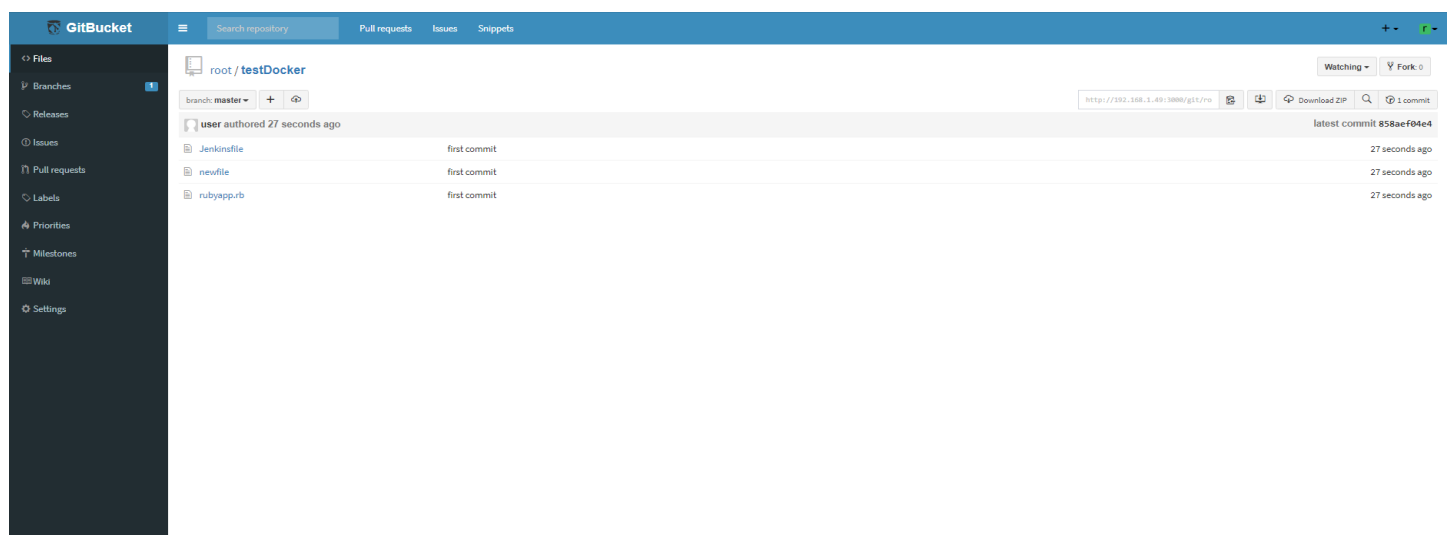
При переходе по адресу 192.168.1.49:3000 нас сразу встречает главная страница сервиса **Gitbucket**, нажимаем на кнопку “sign in” и заходим. При входе под рутом можно создавать пользователей, но для наших целей достаточно создать репозиторий под пользователем root. Для этого в верхнем правом углу нажимаем ‘+’ и выбираем “New repository”, выбираем имя и нажимаем кнопку “Create repository”, после чего будет создан репозиторий с нужным именем.



The screenshot shows the 'Create a new repository' page in GitBucket. The page has a blue header with the GitBucket logo and navigation links: 'Search repository', 'Pull requests', 'Issues', and 'Snippets'. The main content area is titled 'Create a new repository' and includes a subtext: 'A repository contains all the files for your project including the revision history.' Below this, there are two input fields: 'Owner' (set to 'root') and 'Repository name' (set to 'testDocker'). There is also a 'Description (optional):' field. Below these fields, there are three radio button options: 'Public' (selected), 'Private', and 'Create an empty repository'. The 'Public' option is selected, and the 'Create an empty repository' option is also selected. Below these options, there are three radio button options: 'Initialize this repository with an empty commit', 'Initialize this repository with a README', and 'Copy existing git repository'. The 'Initialize this repository with an empty commit' option is selected. Below these options, there is a 'Source git repository URL' field. At the bottom of the form, there is a green 'Create repository' button.

Рис.10

После создания репозитория будет показана инструкция, с помощью которой можно будет связать гит-репозиторий с папкой на хосте (“git remote add ...”). После ввода необходимых команд появится следующая страница(рис.11).



The screenshot shows the repository page for 'root / testDocker' in GitBucket. The page has a blue header with the GitBucket logo and navigation links: 'Search repository', 'Pull requests', 'Issues', and 'Snippets'. On the left side, there is a sidebar with a list of navigation links: 'Files', 'Branches', 'Releases', 'Issues', 'Pull requests', 'Labels', 'Priorities', 'Milestones', 'Wiki', and 'Settings'. The main content area shows the repository details. At the top, there is a 'branch: master' dropdown and a '+ -' button. Below this, there is a 'user authored 27 seconds ago' section. The repository contains three files: 'Jenkinsfile', 'newfile', and 'rubyapp.rb'. Each file has a 'first commit' status and a timestamp of '27 seconds ago'. At the bottom of the repository details, there is a 'latest commit 858aeF04e4' section.

Рис.11

Для соединения с Jenkins использовался адрес “192.168.1.49:3000/git/root/testDocker.git”, который можно скопировать из поля в верхней правой части страницы.

Далее для связи с kanboard требуется зайти в настройки репозитория(кнопка “Settings”) и выбрать раздел “Service Hooks” и нажать кнопку “Add webhook”(рис.12).

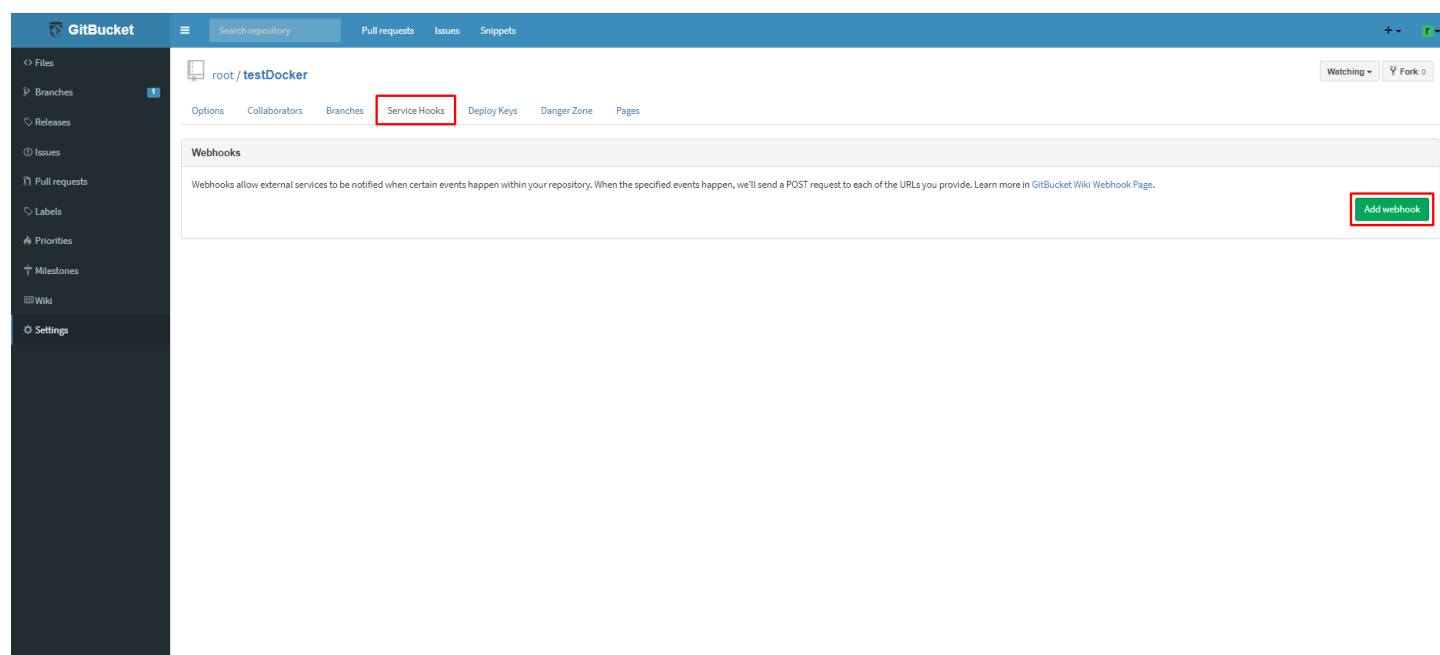


Рис.12

В поле “Payload URL” нужно ввести адрес вебхука из настроек Kanboard.

В поле “Content type” нужно выбрать формат json.

Поле “Security Token” вводить ничего не нужно.

В разделе триггеров нужно выбрать все значения, как показано на рисунке 13.

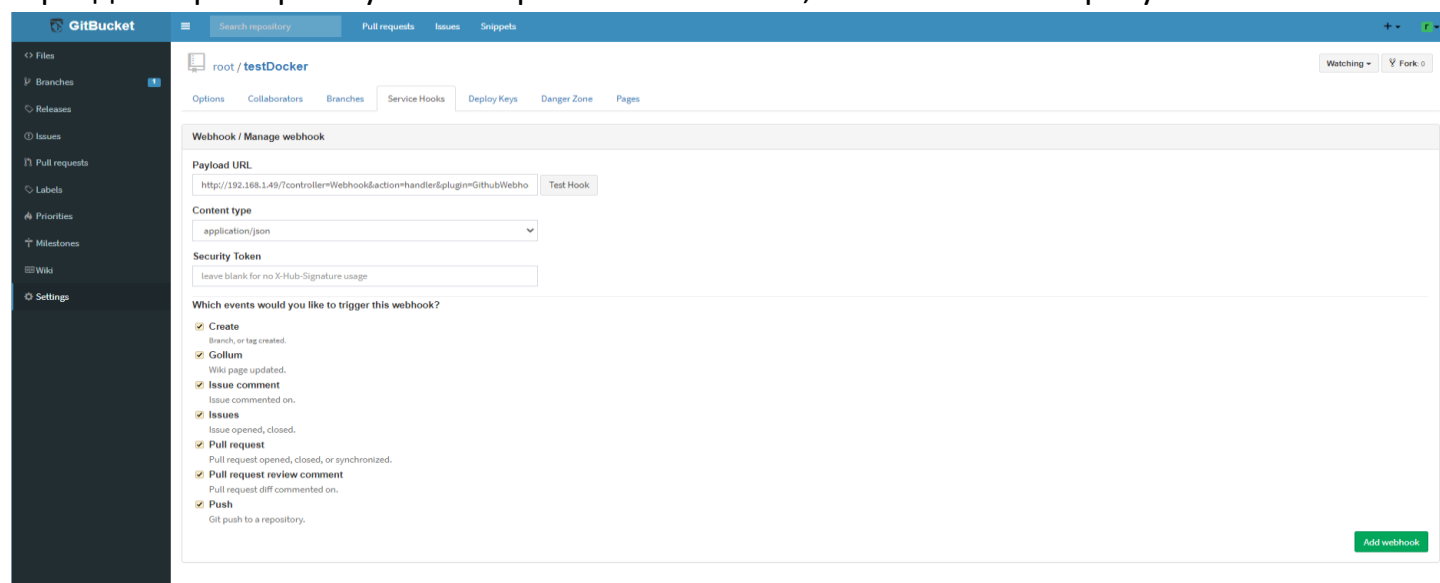


Рис.13

После этого kanboard и gitbucket будут связаны, оставшиеся настройки интеграции нужно будет проводить в сервисе kanboard.

Kanboard:

Перейдя по адресу 192.168.1.49 появится страница с полями ввода логина и пароля.

Заходим с логином:admin и паролем:admin, после чего появляется следующая страница(рис.14).

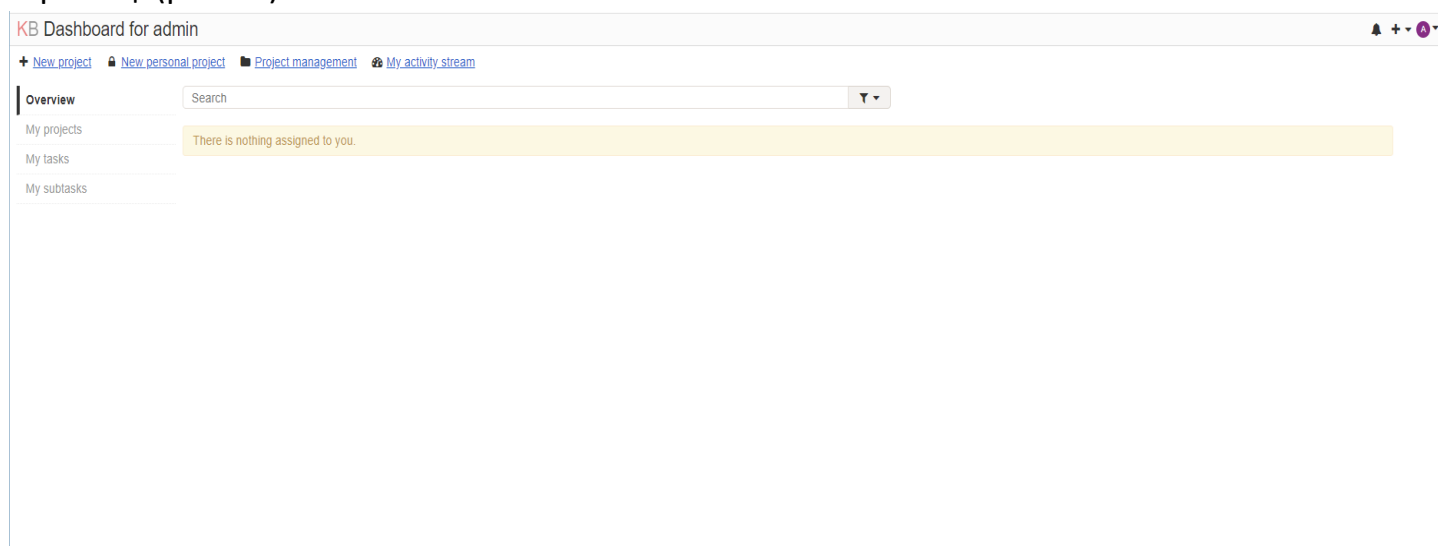


Рис.14

Здесь в верхнем левом углу следует нажать на кнопку “New project”, после чего ввести имя проекта и нажать кнопку “save”. После этого появится главная страница созданного проекта, на которой нужно нажать на кнопку “Integrations”. На этой странице будет доступна ссылка вебхука, которую нужно указать в сервисе Gitbucket.

Далее для настройки интеграции нажимаем на кнопку “Automatic actions”, и там добавим несколько пар действий (Kanboard-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 bug #3”)

Create a task from external provider – Github issue opened (задача в Kanboard открывается при создание issue в Gitbucket.

Аналогичным образом можно создать еще несколько пар действий, но для демонстрации интеграции сервисов достаточно этих двух пар(рис.15).

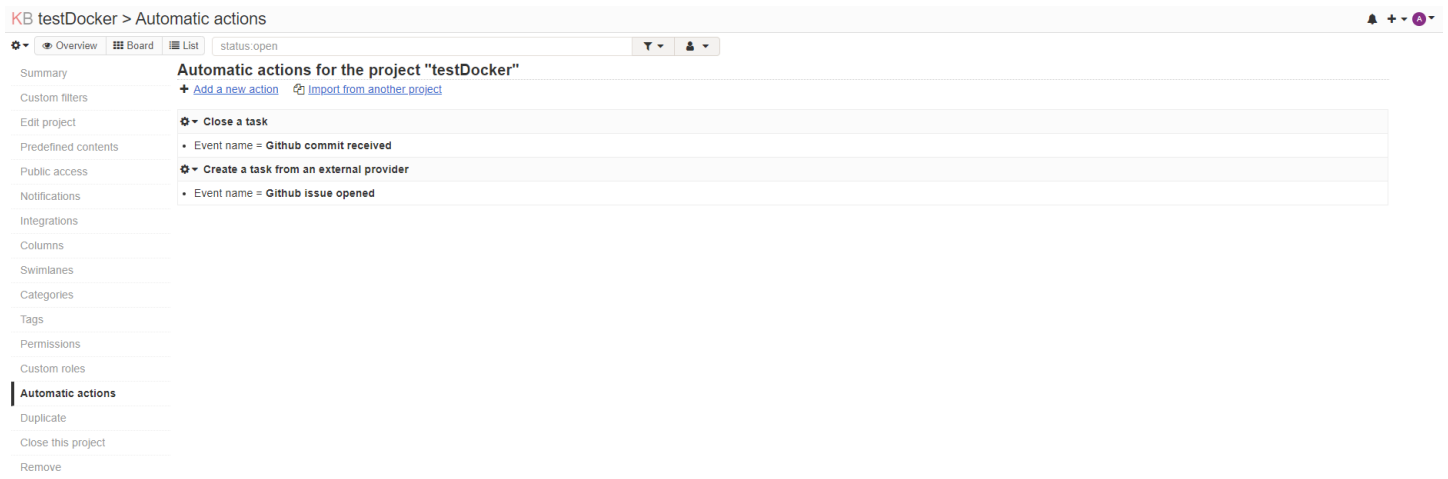


Рис.15

Демонстрация связи сервисов:

В сервисе Gitbucket переходим на вкладку “Issues” и нажимаем кнопку “New issue”, после чего появляется следующая страница(рис.16)

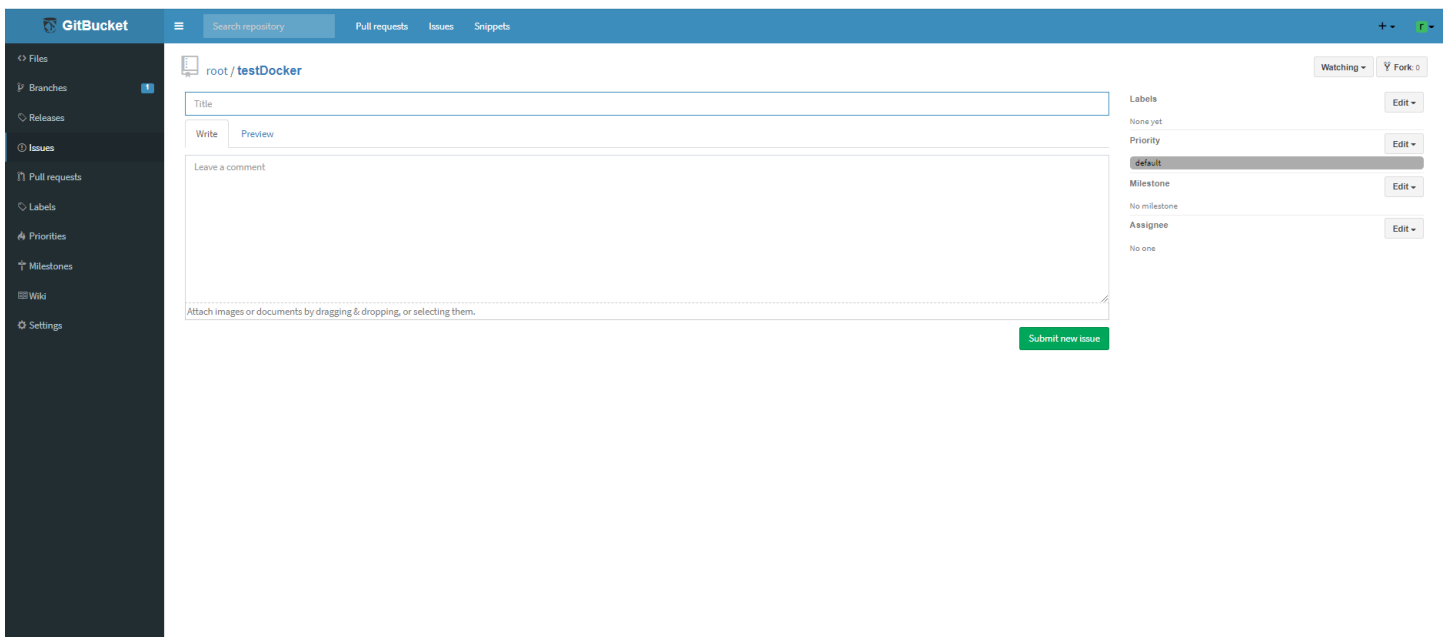


Рис.16

Заполняем поля на свое усмотрение, для теста хватит лишь одного название, допустим “docker-issue”, после чего нажимаем “Submit new issue”. После этого заходим на страницу проекта kanboard и видим задачу с аналогичным заголовком(рис.17)

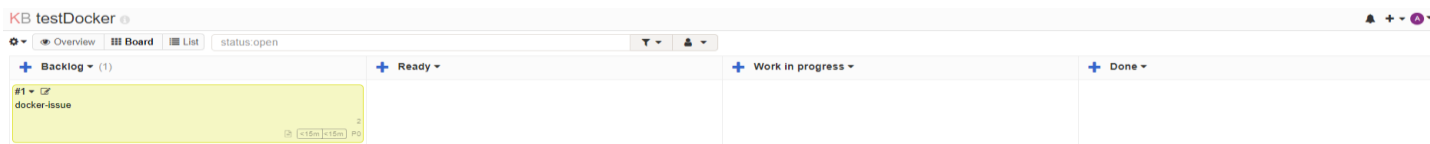


Рис.17

Далее нужно исправить строку в файле `rubyapp.rb`, чтобы он заработал. В коммите укажем, что нужно закрыть задачу #1, после чего должна закрыться задача в Kanboard, а также запустится автоматическая сборка в Jenkins.

```
user@user:~/forgitbucket$ git add rubyapp.rb
user@user:~/forgitbucket$ git commit -m "commit for test integration #1"
[master 0a8427e] commit for test integration #1
  Committer: user <user@user.user>
  Ваше имя или электронная почта настроены автоматически на основании вашего
  имени пользователя и имени машины. Пожалуйста, проверьте, что они
  определены правильно.
  Вы можете отключить это уведомление установив их напрямую. Запустите следующую
  команду и следуйте инструкциям вашего текстового редактора, для
  редактирования вашего файла конфигурации:

  git config --global --edit

После этого, изменить авторство этой коммита можно будет с помощью команды:

  git commit --amend --reset-author

1 file changed, 1 insertion(+), 1 deletion(-)
user@user:~/forgitbucket$ git push
Username for 'http://192.168.1.49:3000': root
Password for 'http://root@192.168.1.49:3000':
Перечисление объектов: 5, готово.
Подсчет объектов: 100% (5/5), готово.
Сжатие объектов: 100% (3/3), готово.
Запись объектов: 100% (3/3), 330 bytes | 330.00 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://192.168.1.49:3000/git/root/testDocker.git
   858aef0..0a8427e  master -> master
user@user:~/forgitbucket$
```

Рис.18

В сервисе Kanboard задача после коммита была закрыта.

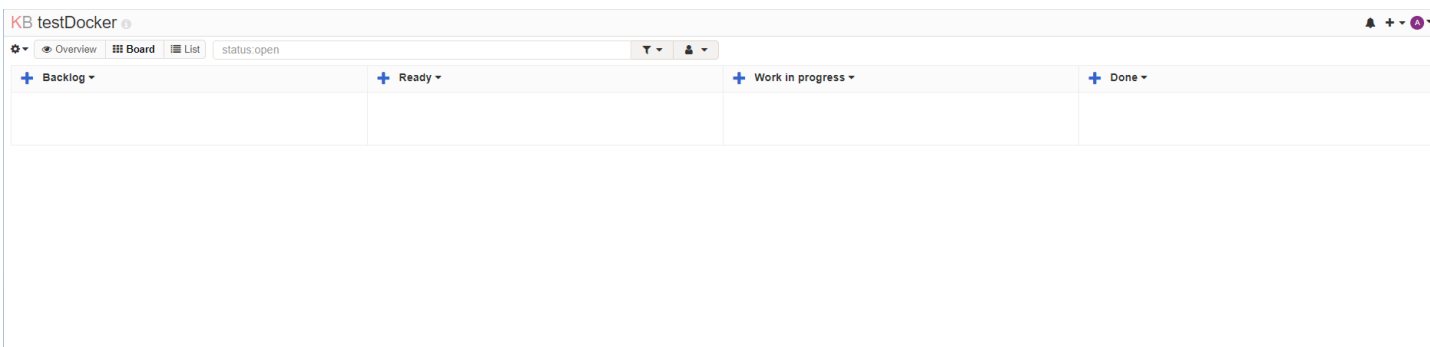


Рис.19

В сервисе Jenkins было отслежено изменение на Gitbucket(рис.20), после чего была запущена автоматическая сборка, которая завершилась со следующим логом(рис.21).

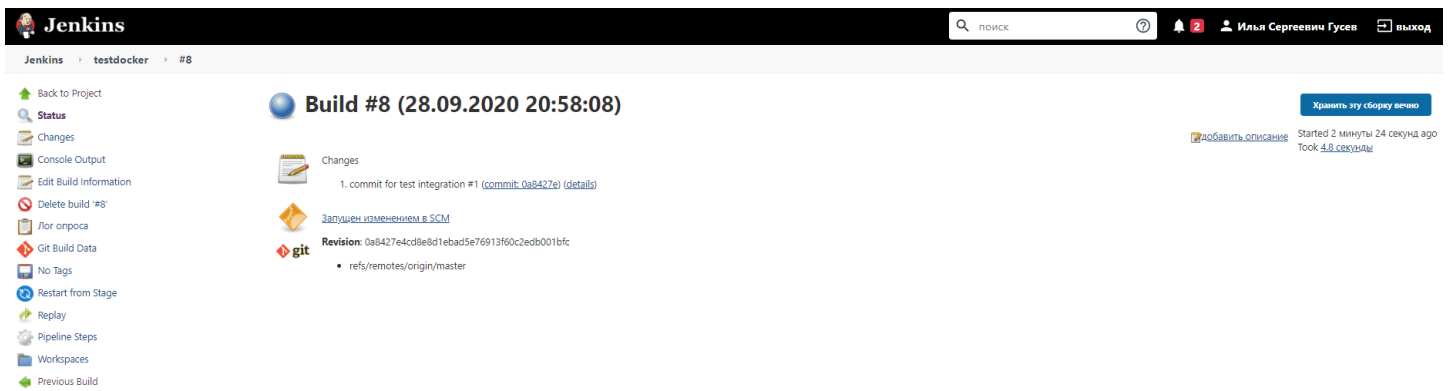


Рис.20

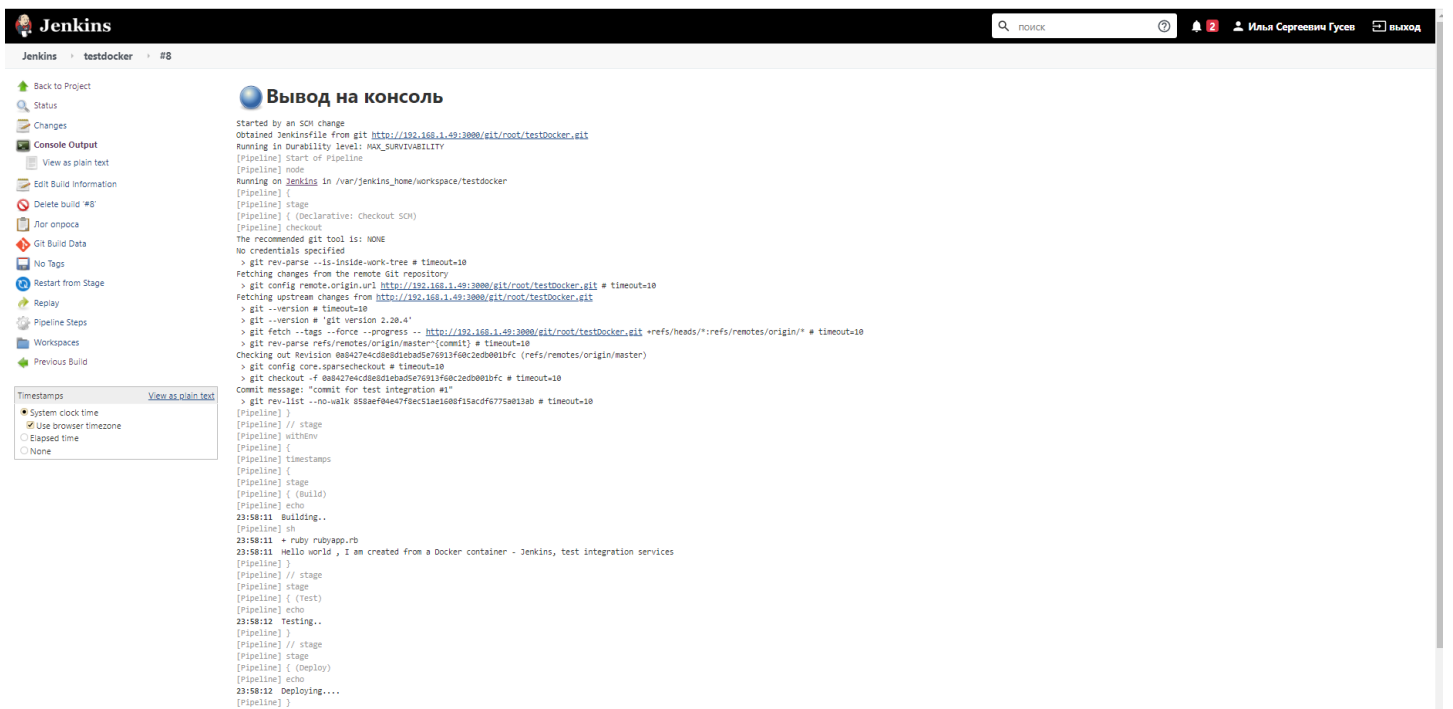


Рис.21

На рисунке 21 видно, что приложение ruby запустилось, и сборка прошла успешно.