

Gitea: хостинг и управление репозиториями с возможностями Gitea

Actions

Цель работы

Исследование и рассмотрение процесса установки и настройки платформы хостинга и управления репозиториями Gitea, а также предоставление конкретных примеров непрерывной интеграции (CI) и непрерывной доставки (CD).

Краткие теоретические сведения

Gitea - это система управления репозиториями Git, предоставляющая пользовательский интерфейс для работы с Git-репозиториями. Этот проект был создан в качестве альтернативы другим популярным системам, таким как GitLab и GitHub. Gitea предлагает легковесный и простой в установке вариант для тех, кто предпочитает самостоятельное управление своими Git-репозиториями.

Вот несколько ключевых особенностей Gitea:

Легковесность: Gitea разрабатывается с акцентом на минимальные системные требования и быструю установку. Это делает его отличным выбором для тех, кто хочет запустить свой сервер управления репозиториями с минимальными усилиями.

Открытый исходный код: Gitea является проектом с открытым исходным кодом, что позволяет пользователям свободно изучать, модифицировать и распространять программное обеспечение в соответствии с лицензией MIT.

Поддержка протокола Git: Gitea полностью совместим с протоколом Git, что означает, что он может работать с любыми совместимыми клиентами и инструментами Git.

Web-интерфейс: Gitea предоставляет пользовательский интерфейс через веб-браузер, что упрощает создание и управление репозиториями без необходимости использования командной строки Git.

Встроенные инструменты для совместной работы: Gitea включает в себя различные инструменты для управления задачами, отслеживания ошибок и ведения форумов, что делает его полнофункциональной платформой для совместной работы.

Безопасность и аутентификация: Gitea поддерживает аутентификацию через различные источники, такие как LDAP, OAuth, и включает в себя механизмы безопасности для защиты от угроз.

Добавление функциональности, такой как обсуждения в коде, рецензии, и возможность интеграции с инструментами для планирования проектов, может улучшить совместную работу в команде.

Ход работы

1 Установка

Импортируйте виртуальную машину из файла «Gitea.ova». Это можно сделать через внутреннее меню программы VirtualBox (Файл – Импорт конфигураций). После импортирования виртуальной машины установите настройки сетевого адаптера в режим «Сетевой мост».

После запуска виртуальной машины выполните вход от имени учетной записи «student» с помощью пароля «123». Далее откройте терминал через меню приложений или с помощью нажатия клавиш «Ctrl»+«Alt»+«T».

В данном окне выполните команду «ip a», чтобы узнать IP-адрес в вашей локальной сети.

Для установки Gitea необходимо открыть браузер и перейти по ссылке «<http://127.0.0.1:3000>», если есть возможность сделать это на виртуальной машине. Или узнать IP-адрес с помощью команды «ip a» и подключиться из основной системы (сеть у виртуальной машины должна иметь параметр «Сетевой мост») (рисунок 1.1). В открывшемся окне будут представлены графические настройки для первоначальной конфигурации сервера (рисунок 1.2).

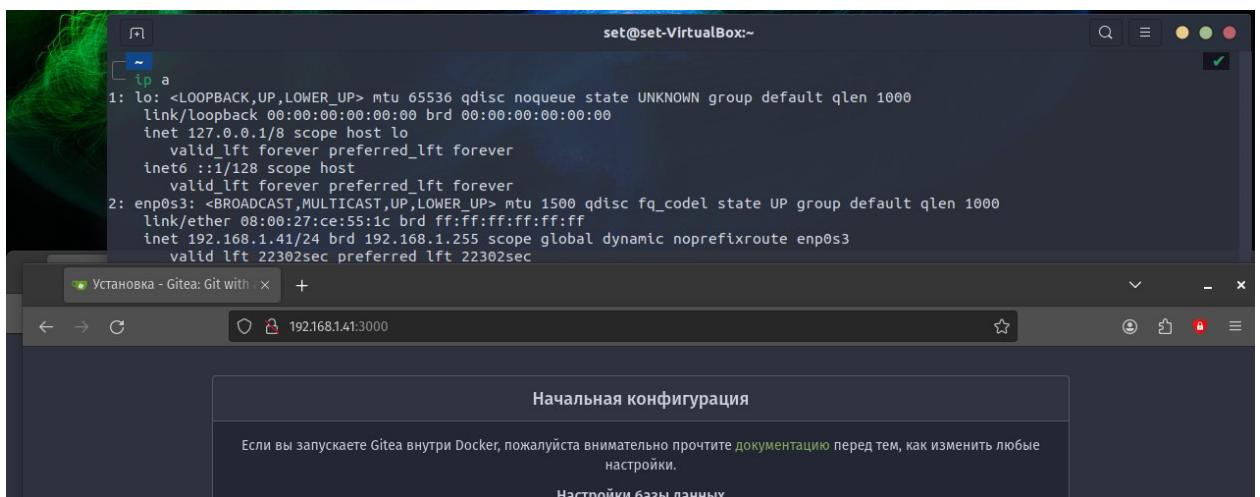


Рисунок 1.1 – Открытие начальной конфигурации

Если вы запускаете Gitea внутри Docker, пожалуйста внимательно прочтите [документацию](#) перед тем, как изменить любые настройки.

Настройки базы данных

Gitea требует MySQL, PostgreSQL, MSSQL, SQLite3 или TiDB (через протокол MySQL).

Тип базы данных*	MySQL
Хост*	127.0.0.1:3306
Имя пользователя*	gitea
Пароль*
Имя базы данных*	giteadb

Основные настройки

Рисунок 1.2 – Начальная конфигурация

В данном окне необходимо указать имя базы данных, которая будет использоваться (giteadb), имя пользователя (gitea) и пароль (Gitea123!), которые будут использоваться gitea. В данных настройках есть возможность настроить доменное имя, взаимодействие с почтовым сервером, если они есть. Далее необходимо в конце страницы открыть вкладку «Настройки учетной записи администратора» и создать учетную запись для администратора в формате «fio-1111», где первые три буквы – ФИО, а цифры – номер группы (рисунок 1.3).

▶ Сервер и настройки внешних служб
▼ Настройки учётной записи администратора
Создание учётной записи администратора необязательно. Первый зарегистрированный пользователь автоматически становится администратором.
Логин администратора fio-1111
Адрес эл. почты fio-1111@example.com
Пароль
Подтвердить пароль
These configuration options will be written into: /etc/gitea/app.ini
Установить Gitea

Рисунок 1.3 – Создание учетной записи администратора

После проверки правильности введенных данных необходимо нажать на кнопку «Установить Gitea». После чего откроется окно, в котором надо будет подождать пока произойдет автоматическое завершение установки gitea (рисунок 1.4).

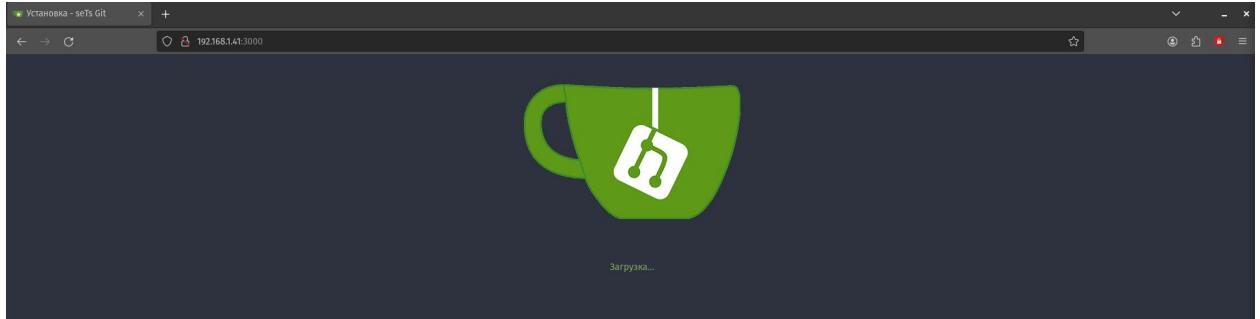


Рисунок 1.4 – Установка gitea

Если установка затянется более чем 35 минут, перезагрузите виртуальную машину и выполните переход по адресу «<http://localhost:3000>» повторно.

После установки откроется окно, в котором необходимо выбрать репозиторий, так как на данный момент ни один не создан, то необходимо нажать на «+» справа от слова «Репозитории» для создания нового (Рисунок 1.5).

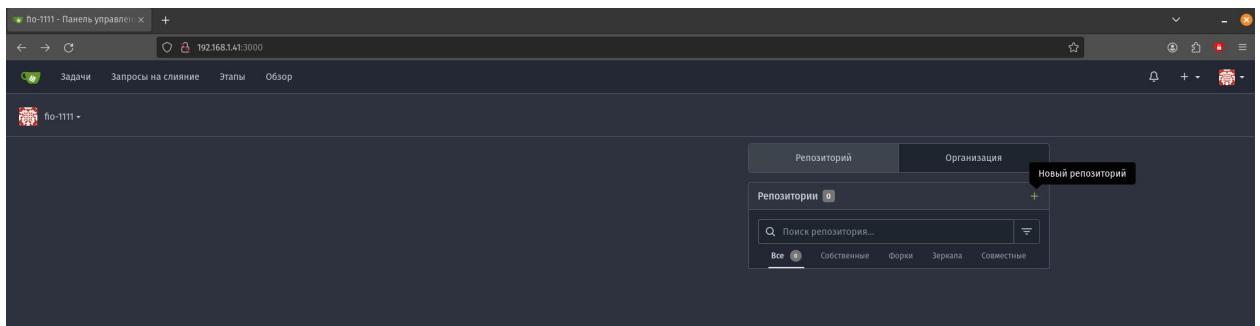


Рисунок 1.5 – Выбор репозитория

В открывшемся окне необходимо заполнить информацию для создаваемого репозитория (рисунок 1.6). В первой части вводится информация, описывающая создаваемый репозиторий для дальнейшей идентификации. Во второй половине можно выбрать файл «`.gitignore`», который позволяет указать шаблоны файлов и папок, которые Git должен игнорировать при отслеживании изменений, лицензию, которая определяет

права и обязанности тех, кто будет использовать ваш код, а также ограничения по его использованию, ветку по умолчанию и модель доверия подписи.

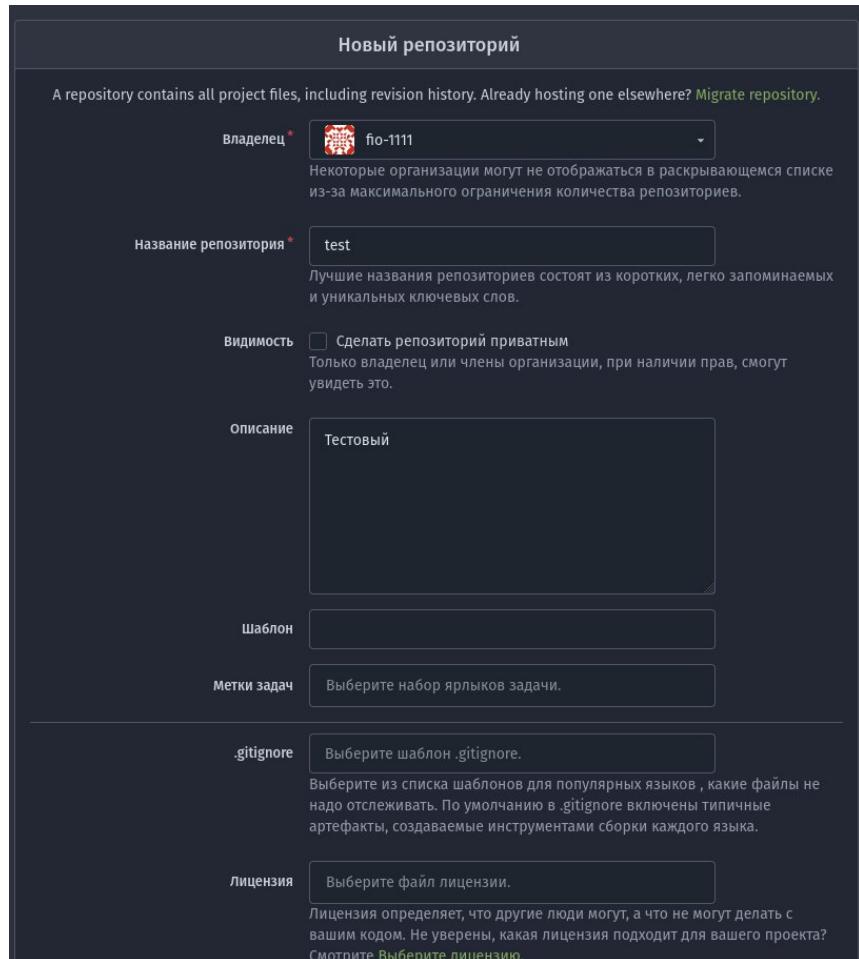


Рисунок 1.6 – Создание репозитория

В gitea есть возможность из созданного репозитория сделать шаблон для дальнейших проектов.

После нажатия кнопки «Создать репозиторий» открывается краткое руководство по созданию репозитория из командной строки, как это было уже рассмотрено в работе с git. Создадим новый репозиторий на локальной машине и добавим файл README.md с описанием репозитория (рисунок 1.7).

```
set@set-VirtualBox:~/test
mkdir test
cd test
~/test
nano README.md
~/test
took 27s
git init
подсказка: Using 'master' as the name for the initial branch. This default branch name
подсказка: is subject to change. To configure the initial branch name to use in all
подсказка: of your new repositories, which will suppress this warning, call:
подсказка:
подсказка:     git config --global init.defaultBranch <name>
подсказка:
подсказка: Names commonly chosen instead of 'master' are 'main', 'trunk' and
подсказка: 'development'. The just-created branch can be renamed via this command:
подсказка:
подсказка:     git branch -m <name>
Инициализирован пустой репозиторий Git в /home/set/test/.git/
~/test on master ?1
git add .
~/test on master +1
git commit -m "initial"
[master (корневой коммит) 3135fc8] initial
1 file changed, 2 insertions(+)
create mode 100644 README.md
~/test on master
git remote add origin http://192.168.1.41:3000/fio-1111/test.git
~/test on master
git push -u origin main
```

Рисунок 1.7 – Добавление файла в созданный репозиторий

После обновления страницы Gitea отобразится добавленный файл (рисунок 1.8).

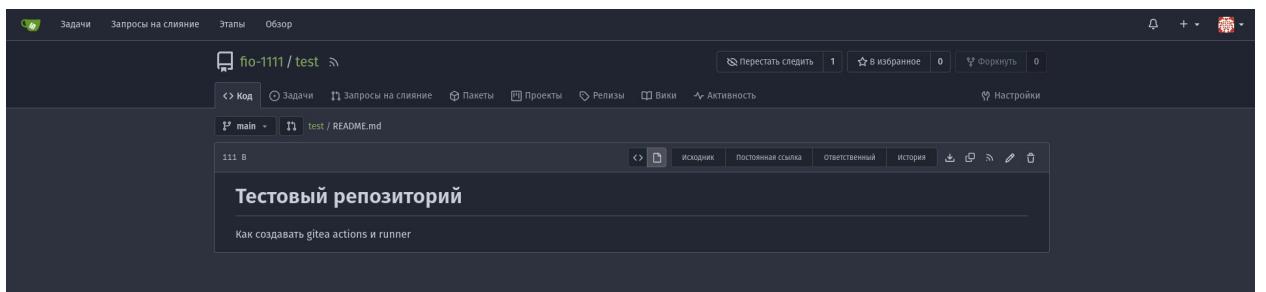


Рисунок 1.8 – Изменение страницы репозитория после добавления файла

2 Автоматизация разработки: CI/CD в Gitea

CI/CD (Continuous Integration/Continuous Deployment) - это практика разработки программного обеспечения, направленная на автоматизацию процессов интеграции кода, его тестирования и развертывания в среде продакшн. Этот подход позволяет командам разработки быстро и надежно внедрять новые функции и изменения в продукт. Давайте рассмотрим основные этапы CI/CD и что происходит на каждом из них.

1. Continuous Integration (CI) этапы:

- Кодирование (Coding) - Разработчики пишут код для новых функций или вносят изменения в существующий код.
- Версионирование кода (Version Control) - Использование системы контроля версий (например, Git) для отслеживания изменений в коде.
- Интеграция кода (Code Integration) - Регулярное слияние кода из разных веток в основную ветку (например, master).
- Автоматическая сборка (Automated Build) - Автоматизированный процесс компиляции и сборки приложения из исходного кода.
- Запуск юнит-тестов (Running Unit Tests) - Выполнение автоматических юнит-тестов для проверки базовой функциональности.
- Статический анализ кода (Static Code Analysis) - Использование инструментов для выявления потенциальных проблем в коде (например, линтеры).

2. Continuous Deployment (CD) этапы:

- Автоматическое тестирование (Automated Testing) - Запуск различных видов тестов, включая интеграционные, системные и другие.
- Статический анализ кода (Static Code Analysis) - Дополнительная проверка кода на наличие потенциальных проблем.
- Создание артефакта (Artifact Creation) - Формирование готового к развертыванию артефакта, например, дистрибутива или контейнера.

- Автоматическое развертывание (Automated Deployment) - Автоматизированное развертывание артефакта в тестовой среде для дополнительного тестирования.
- Тестирование производительности (Performance Testing) - Проведение тестирования производительности для оценки работы приложения под нагрузкой.
- Ручное тестирование (Manual Testing) - Проведение ручных тестов, если необходимо.
- Развертывание в продакшн (Deployment to Production) - Автоматизированное развертывание в продакшн, если все тесты прошли успешно.
- Мониторинг (Monitoring) - Отслеживание работы приложения в реальном времени с использованием инструментов мониторинга.

Gitea Actions - это инструмент для автоматизации процессов в вашем репозитории на Gitea. Они позволяют создавать и выполнять различные рабочие процессы (workflows) в ответ на события в вашем репозитории или внешних событий, таких как отправка кода, создание запроса на слияние, выпуск релиза и другие.

Раннеры (runners) - это виртуальные машины или физические устройства, на которых выполняются ваши рабочие процессы. Когда Gitea Actions запускает рабочий процесс, он использует выбранный раннер для выполнения задач этого процесса. Раннеры могут быть развернуты на различных платформах, таких как Linux, Windows, macOS, и они могут обеспечивать различные окружения для вашего кода.

Вот для чего нужны раннеры:

Автоматизация рабочих процессов: Gitea Actions позволяет автоматизировать различные задачи, такие как сборка, тестирование, развертывание и многие другие. Это улучшает эффективность разработки и помогает предотвращать ошибки.

Непрерывная интеграция (CI): Gitea Actions часто используется для создания системы непрерывной интеграции, автоматической проверки кода при его отправке и предотвращения проблем интеграции.

Непрерывная доставка (CD): Gitea Actions также может использоваться для настройки процессов непрерывной доставки, которые автоматически разворачивают приложение после успешной сборки и тестирования.

Работа с различными окружениями: Раннеры позволяют вам определять, на каких платформах и в каких окружениях выполняются ваши рабочие процессы. Например, вы можете проверять, как ваш код работает на Linux, Windows и macOS.

2.1 Непрерывная Интеграция (CI) в Gitea

В данной работе будет происходить разворачивание раннера на той же системе, где был развернут gitea. В реальных проектах рекомендуется делать это на отдельной машине для снижения нагрузки на сам gitea.

Далее необходимо открыть страницу gitea веб-интерфейса и по нажатию на иконку пользователя выполнить переход на страницу панели управления (рисунок 2.1.1).

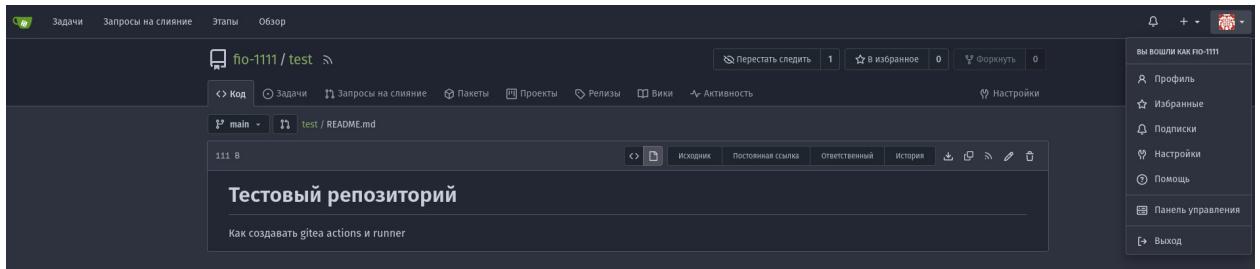


Рисунок 2.1.1 – Открытие панели управления

В открывшемся окне необходимо выполнить переход в раздел «Действия» - «Раннеры» и нажать на кнопку «Создать новый раннер». Из выпадающего окна необходимо скопировать регистрационный токен (рисунок 2.1.2).

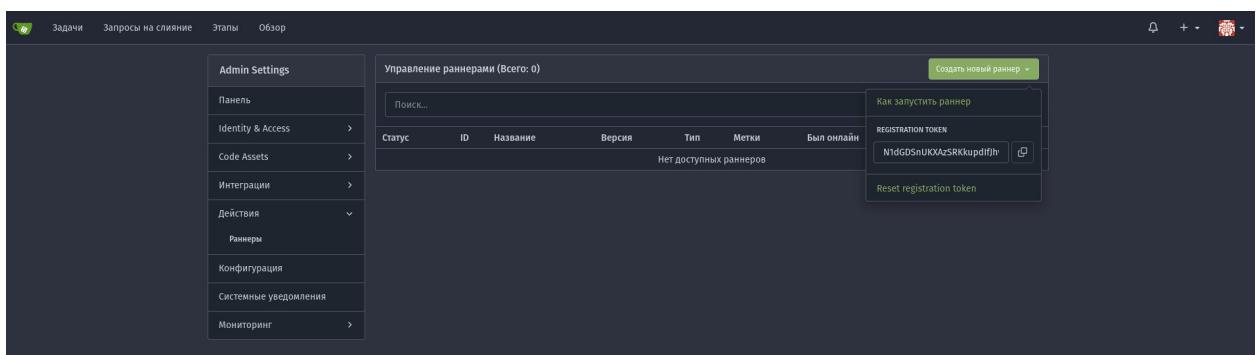
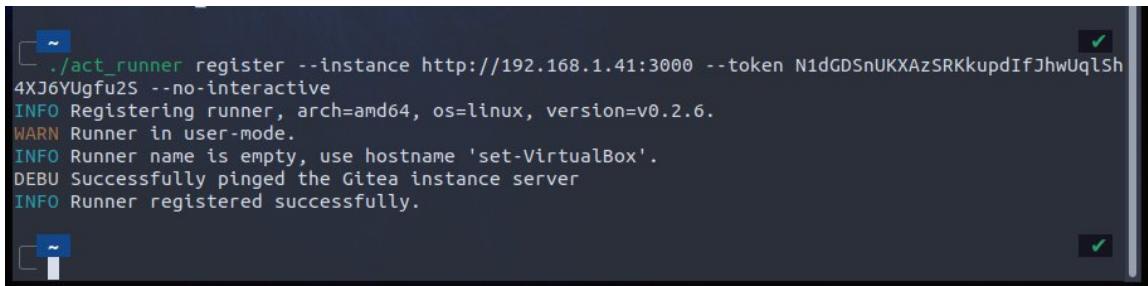


Рисунок 2.1.2 – Регистрационный токен

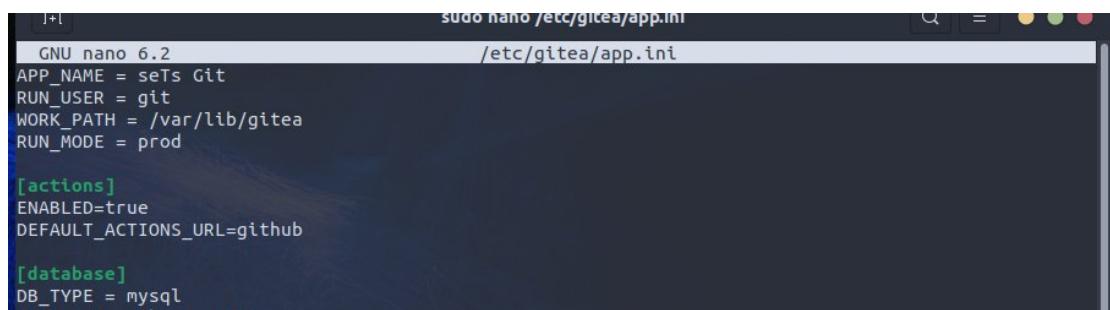
Затем в терминале сервера необходимо выполнить команду «cd» и «./act_runner register --instance http://(IP-адрес):3000 --token (Скопированный регистрационный токен) --no-interactive», что позволит зарегистрировать новый раннер (рисунок 2.1.3).



```
./act_runner register --instance http://192.168.1.41:3000 --token N1dGDSnUKXAzSRKkupdIfJhwUqlSh4XJ6YUgf2S --no-interactive
INFO Registering runner, arch=amd64, os=linux, version=v0.2.6.
WARN Runner in user-mode.
INFO Runner name is empty, use hostname 'set-VirtualBox'.
DEBU Successfully pinged the Gitea instance server
INFO Runner registered successfully.
```

Рисунок 2.1.3 – Регистрация раннера

Затем необходимо в конфигурационном файле gitea, который располагается по пути «`sudo nano /etc/gitea/app.ini`» добавить строчку (“`[actions] ENABLED = true DEFAULT_ACTIONS_URL = github`”), которая обозначает, что сервер использует раннера для активации данной функции (рисунок 2.1.4).



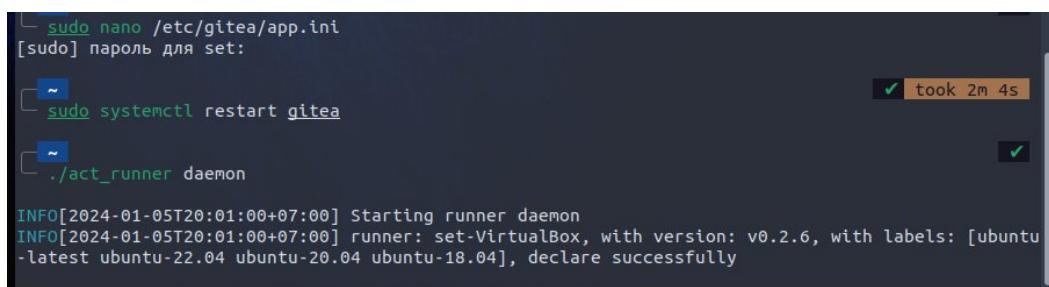
```
GNU nano 6.2
/etc/gitea/app.ini
APP_NAME = set's Git
RUN_USER = git
WORK_PATH = /var/lib/gitea
RUN_MODE = prod

[actions]
ENABLED=true
DEFAULT_ACTIONS_URL=github

[database]
DB_TYPE = mysql
```

Рисунок 2.1.4 – Включение функции использования Gitea Actions

После внесения изменений необходимо перезапустить сервис gitea с помощью команды «`sudo systemctl restart gitea`». Далее можно запустить зарегистрированный раннер. Для этого в терминале необходимо выполнить команду «`./act_runner daemon`» (рисунок 2.1.5).



```
sudo nano /etc/gitea/app.ini
[sudo] пароль для set:

sudo systemctl restart gitea
[  took 2m 4s]

./act_runner daemon

INFO[2024-01-05T20:01:00+07:00] Starting runner daemon
INFO[2024-01-05T20:01:00+07:00] runner: set-VirtualBox, with version: v0.2.6, with labels: [ubuntu-latest ubuntu-22.04 ubuntu-20.04 ubuntu-18.04], declare successfully
```

Рисунок 2.1.5 – Запуск раннера

В данном терминале будут отображаться логи выполняемых действий раннером. Для проверки создадим несколько каталогов и файлов в созданном ранее репозитории, как это представлено на рисунке 2.1.6.

```

./act_runner daemon
set@set-VirtualBox:~/test
└─~ cd test
  └─~/test on main
    └─mkdir .gitea
      └─~/test on main
        └─mkdir .gitea/workflows
          └─~/test on main
            └─touch .gitea/workflows/demo.yaml
          └─~/test on main ?1
            └─mkdir code_test
              └─~/test on main ?1
                └─cd code_test
                  └─~/test/code_test on main ?1
                    └─nano test_hello_world.py
                  └─~/test/code_test on main ?2
                    └─cd
                    └─~ cd test
                      └─~/test on main ?2
                        └─ls
                          code_test README.md
                      └─~/test on main ?2
                        └─nano hello_world.py

```

Рисунок 2.1.6 – Добавление файлов и каталогов

В файл `test_hello_world.py` добавьте следующее содержимое:

```

# code_test/test_hello_world.py

import sys
from io import StringIO
from pathlib import Path
from termcolor import colored

# Добавляем каталог с проектом в sys.path
sys.path.append(str(Path(__file__).resolve().parents[1]))

from hello_world import print_hello_world

def test_print_hello_world(capsys):
    # Захватываем вывод на стандартный поток
    captured_stdout = StringIO()
    sys.stdout = captured_stdout

    # Вызываем функцию, которую мы тестируем
    print_hello_world()

    # Возвращаем stdout на стандартный поток
    sys.stdout = sys.__stdout__

    # Получаем захваченный вывод
    captured_output = captured_stdout.getvalue()

```

```
# Ожидаемая строка вывода
expected_output = "Hello World!\n"

# Проверяем, что вывод соответствует ожидаемой строке
assert captured_output == expected_output
```

В файл hello_world.py добавьте следующее содержимое:

```
# Содержимое файла hello_world.py
```

```
def print_hello_world():
    print("Hello World!")

if __name__ == "__main__":
    print_hello_world()
```

В файл demo.yaml добавьте следующее содержимое:

```
name: Run Test

on:
  push:

jobs:
  test:
    runs-on: ubuntu-latest
    container:
      image: catthehacker/ubuntu:act-22.04

    steps:
      - name: Echo Test Message
        run: echo "Тест был запущен в результате выполнения push"

      - name: Checkout repository
        uses: actions/checkout@v3

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.9'
          architecture: 'x64'

      - name: Install dependencies
        run:
          pip install termcolor
          pip install pytest

      - name: Run tests
        run:
          pytest code_test/test_hello_world.py
```

В данном файле происходит конфигурация Gitea Actions:

name: Run Test: Это название вашего workflow (рабочего процесса), именно так его увидят другие разработчики в вашем репозитории.

on: и push: Это событие, при котором запускается workflow. В данном случае, workflow будет запускаться при каждом push'e в репозиторий.

jobs: Определение рабочих задач.

test: Имя задачи. Задача определена для выполнения тестов.

runs-on: ubuntu-latest: Задача будет выполняться на последней версии Ubuntu.

container: Использование контейнера для выполнения задачи.

image: catthehacker/ubuntu:act-22.04: Указание образа Docker, который будет использоваться для создания контейнера. В данном случае, это образ Ubuntu с предустановленным актуальным ПО.

steps: Шаги, которые будут выполнены в рамках данной задачи.

name: Echo Test Message: Вывод сообщения в консоль. В данном случае, просто информационное сообщение, описывающее, что тест был запущен в результате push.

name: Checkout repository: Использование действия (action) для проверки репозитория. Это позволяет вашему workflow получить доступ к коду в вашем репозитории.

name: Set up Python: Настройка Python внутри контейнера.

uses: actions/setup-python@v4: Использование действия для настройки Python. В данном случае, используется версия 4 этого действия.

with: python-version: '3.9', architecture: 'x64': Указание версии Python и архитектуры.

name: Install dependencies: Установка зависимостей проекта.

run: |: Запуск команд внутри контейнера.

pip install termcolor: Установка пакета termcolor с использованием pip.

pip install pytest: Установка пакета pytest с использованием pip.

name: Run tests: Запуск тестов.

run: |: Запуск команд внутри контейнера.

pytest code_test/test_hello_world.py: Запуск тестов с использованием pytest, предполагая, что есть файл тестов test_hello_world.py в папке code_test вашего репозитория.

Кроме описанных ранее действий необходимо включить выполнение Gitea Actions в самом репозитории. Для этого в настройках репозитория необходимо поставить галочку у пункта «Действия Включить действия репозитория» и нажать кнопку «Обновить настройки» (рисунок 2.1.7).

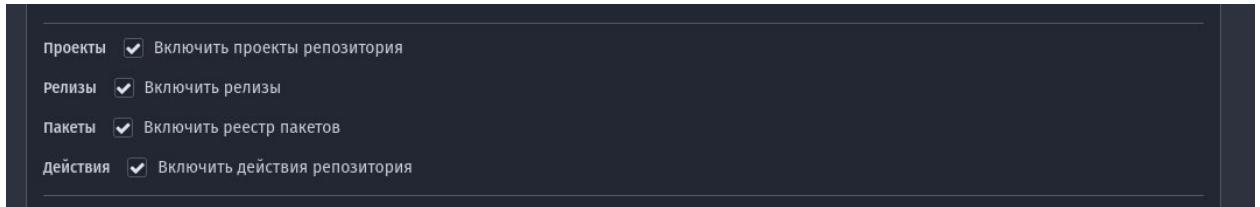


Рисунок 2.1.7 – Включение действий в репозитории

После этого в репозитории появится вкладка «Действия». На этом подготовительные действия были завершены и можно загружать созданные ранее файлы на удаленный репозиторий на сервер (рисунок 2.1.8).

A screenshot of a terminal window titled 'set@set-VirtualBox:~/test'. The command 'git push -u origin main' is being run. The output shows the user being prompted for a password for the remote 'http://fio-1111@192.168.1.41:3000'. The process then continues with object counting, compression, and finally pushing the changes to the remote repository. The message at the end indicates that the 'main' branch is tracking the 'main' branch from 'origin'.

Рисунок 2.1.8 – Загрузка файлов

После этого на странице репозитория во вкладке «Действия» отобразится результат выполнения созданного теста (рисунок 2.1.9).

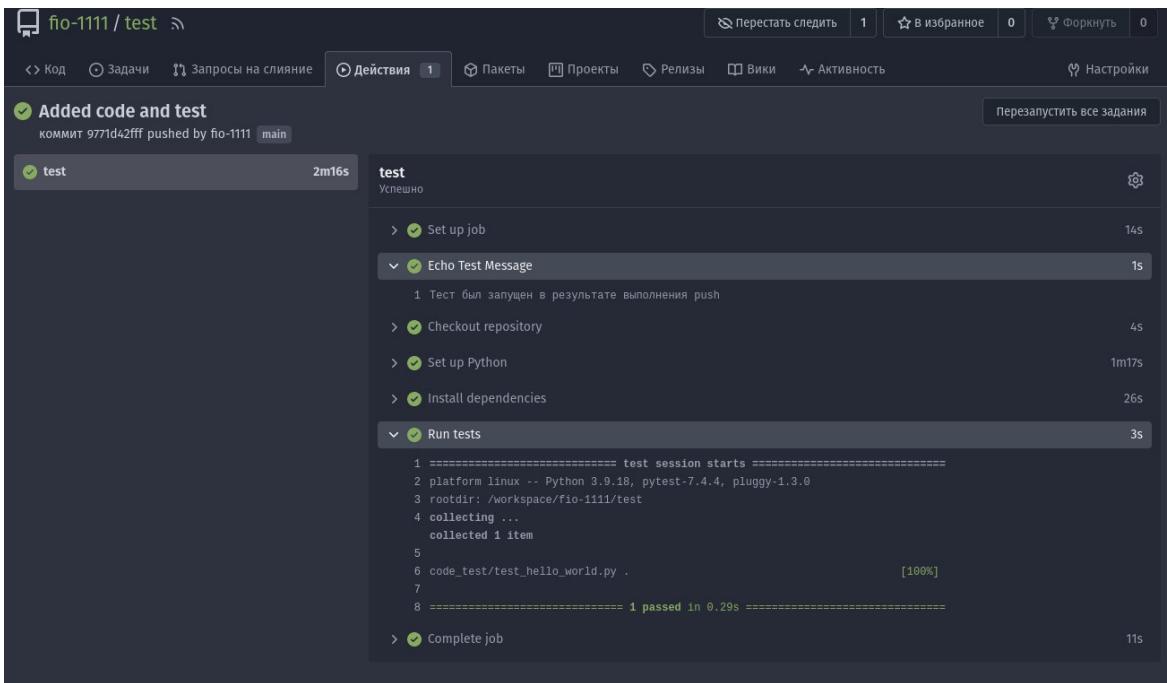


Рисунок 2.1.9 – Результат выполнения тестов

Теперь измените выводимый текст в коде файла «hello_world.py». Отобразите в отчете результат прохождения теста.

3.2 Непрерывная Доставка (CD) в Gitea

Кроме автоматического тестирования кода есть возможность автоматического развертывания после успешной сборки и тестирования.

Для реализации данной возможности создайте новый репозиторий, в котором будет создан новый проект сайта. Для данного репозитория создайте каталог на машине, в котором необходимо инициализировать новый репозиторий, как это представлено на рисунке 3.2.1.

```
~ mkdtr site
~ cd site
~/site touch README.md
~/site git init
подсказка: Using 'master' as the name for the initial branch. This default branch name
подсказка: is subject to change. To configure the initial branch name to use in all
подсказка: of your new repositories, which will suppress this warning, call:
подсказка:
подсказка:     git config --global init.defaultBranch <name>
подсказка:
подсказка: Names commonly chosen instead of 'master' are 'main', 'trunk' and
подсказка: 'development'. The just-created branch can be renamed via this command:
подсказка:
подсказка:     git branch -m <name>
Инициализирован пустой репозиторий Git в /home/set/site/.git/

~/site on master ?1
git checkout -b main
Переключились на новую ветку «main»

~/site on main ?1
git add .

~/site on main +1
git commit -m "first commit"
[main (корневой коммит) 5156039] first commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 README.md

~/site on main
git remote add origin http://192.168.1.41:3000/fio-1111/site.git

~/site on main
git push -u origin main
Username for 'http://192.168.1.41:3000': fio-1111
Password for 'http://fio-1111@192.168.1.41:3000':
Перечисление объектов: 3, готово.
Подсчет объектов: 100% (3/3), готово.
Запись объектов: 100% (3/3), 206 байтов | 206.00 Киб/с, готово.
```

Рисунок 3.2.1 – Инициализация нового репозитория

Далее в данном каталоге создайте четыре основных файла: index.php (в данном файле описывается как именно будет выглядеть пользовательский интерфейс, приложение А), calculate.php (в данном файле будут производиться вычисления на стороне сервера на основе данных, полученных с стороны пользователя, приложение Б), script.js (в данном файле описывается функция, которая вызывается при нажатии на кнопку для отправки данных на сервер,

приложение В), styles.css (в данном файле описывается оформление стилей пользовательского интерфейса, приложение Г)(рисунок 3.2.2).

```
~/site on main
nano index.php
✓ took 10s

~/site on main ?1
nano calculate.php
✓ took 7s

~/site on main ?2
nano script.js
✓ took 26s

~/site on main ?3
nano styles.css
✓ took 10s
```

Рисунок 3.2.2 – Создание основных файлов для проекта

Затем необходимо создать каталог сайта для веб-сервера. Для этого в директории /var/www/html создайте каталог с именем репозитория, скопируйте созданные ранее файлы в данный каталог и выдайте права для доступа к ним с помощью команд (рисунок 3.2.3):

```
sudo mkdir site
sudo chown -R $USER:$USER /var/www/html/site
sudo chmod -R 755 /var/www
cp /home/(имя пользователя)/site/index.php /var/www/html/site/
cp /home/(имя пользователя)/site/calculate.php /var/www/html/site/
cp /home/(имя пользователя)/site/script.js /var/www/html/site/
cp /home/(имя пользователя)/site/styles.css /var/www/html/site/
sudo chown -R $USER:$USER /var/www/html/site
sudo chmod -R 755 /var/www
```

```

    Ⓛ Ø /var/www/html
        sudo mkdir site

    Ⓛ Ø /var/www/html
        sudo chown -R $USER:$USER /var/www/html/site

    Ⓛ Ø /var/www/html
        sudo chmod -R 755 /var/www

    Ⓛ Ø /var/www/html
        cp /home/set/site/index.php /var/www/html/site/

    Ⓛ Ø /var/www/html
        cp /home/set/site/calculate.php /var/www/html/site/

    Ⓛ Ø /var/www/html
        cp /home/set/site/script.js /var/www/html/site/

    Ⓛ Ø /var/www/html
        cp /home/set/site/styles.css /var/www/html/site/

    Ⓛ Ø /var/www/html
        cd site

    Ⓛ /var/www/html/site
        ls
        calculate.php index.php script.js styles.css

    Ⓛ /var/www/html/site
        cd ..

    Ⓛ Ø /var/www/html
        sudo chown -R $USER:$USER /var/www/html/site

    Ⓛ Ø /var/www/html
        sudo chmod -R 755 /var/www

    Ⓛ Ø /var/www/html
        cd site

```

Рисунок 3.2.3 – Создание каталога для веб-сервера

Далее необходимо создать конфигурационный файл для виртуального хоста веб-сервера Apache2. Для этого создайте файл с помощью команды «`sudo nano /etc/apache2/sites-available/site.conf`» и добавьте следующее содержимое:

```

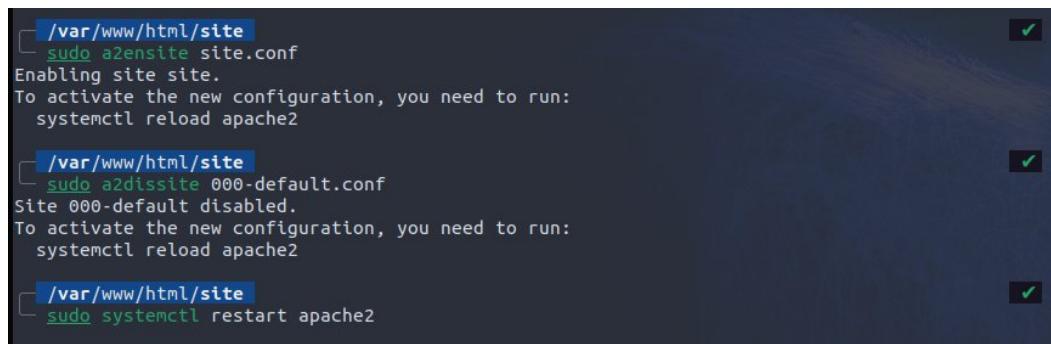
<VirtualHost *:80>
    ServerName Ваш IP
    DocumentRoot /var/www/html/site

    <Directory /var/www/html/site>
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>

```

Далее необходимо заменить страницу, которая отображается по умолчанию, на страницу, которая была создана ранее (рисунок 3.2.4).



```
/var/www/html/site
  sudo a2ensite site.conf
Enabling site site.
To activate the new configuration, you need to run:
  systemctl reload apache2

/var/www/html/site
  sudo a2dissite 000-default.conf
Site 000-default disabled.
To activate the new configuration, you need to run:
  systemctl reload apache2

/var/www/html/site
  sudo systemctl restart apache2
```

Рисунок 3.2.4 – Замена отображаемой страницы

```
sudo a2ensite site.conf
sudo a2dissite 000-default.conf
sudo systemctl restart apache2
```

В репозитории на gitea включите использование действий репозитория, как это представлено на рисунке 3.2.5.

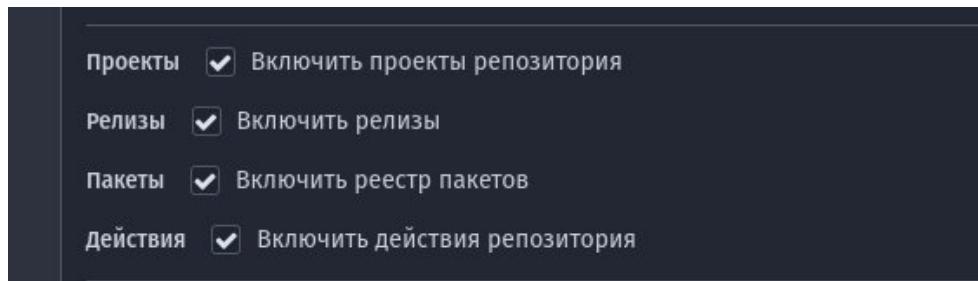


Рисунок 3.2.5 – Включение действий

Далее создайте новую пару SSH-ключей с использованием алгоритма RSA с помощью команды «ssh-keygen -t rsa -b 4096» и добавьте содержимое открытого ключа в файл authorized_keys с помощью команды «cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys», выведите содержимое файла .ssh/id_rsa с помощью команды «cat .ssh/id_rsa» и скопируйте результат выполнения команды, включая строки «-----BEGIN OPENSSH PRIVATE KEY-----» и «-----END OPENSSH PRIVATE KEY-----».

После откроите веб-интерфейс и выполните переход в настройки репозитория, в которых в меню необходимо выбрать пункт «Действия» и в подменю выбрать «Секреты». В данном меню будут добавлены переменные и значения, которые используются в коде, но должны быть сохранены в тайне, например, пароли, токены доступа и другие секреты, необходимые для выполнения определенных операций.

В данном меню нажмите на кнопку «Добавить секрет». В открывшемся меню введите название секрета «SSH_PRIVATE_KEY» и в поле «значение» вставьте скопированные ранее данные, как это представлено на рисунке 3.2.6.

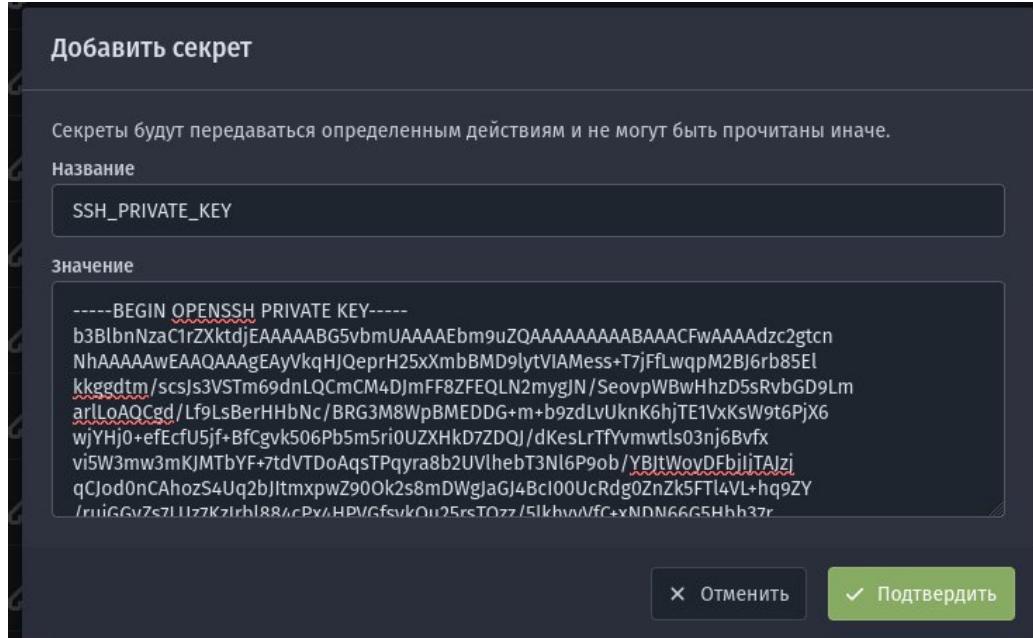


Рисунок 3.2.6 – Добавление секрета

Если секрет был успешно добавлен, то он отобразится в списке и вместо значения будут отображаться звездочки.

После надо добавить еще несколько секретов, которые будут использоваться в данной работе:

APACHE_DIR – директория, в которой располагается каталог сайта для веб-сервера

FILE1 – путь до файла index.php в репозитории, например, /home/(название учетной записи)/(каталог репозитория)/index.php

FILE2 - путь до файла calculate.php в репозитории, например, /home/(название учетной записи)/(каталог репозитория)/calculate.php

FILE3 - путь до файла script.js в репозитории, например, /home/(название учетной записи)/(каталог репозитория)/script.js

FILE4 - путь до файла styles.css в репозитории, например, /home/(название учетной записи)/(каталог репозитория)/styles.css

MAIN_BRANCH – название ветки, main

WORK_DIR – путь до каталога репозитория на сервере
SSH_HOST – IP-адрес сервера
SSH_USER – Имя учетной записи, от имени которой будут выполняться действия. В данном случае используется та же учетная запись, что и для обычного пользователя, но рекомендуется создавать отдельного пользователя для таких задач.

PASS – пароль для получения привилегий суперпользователя.

После в репозитории добавьте директорию `.gitea/workflows`, в которой необходимо добавить файл «`deploy.yml`», в котором необходимо добавить следующее содержимое:

```
name: Deploy to Production
```

```
on:
```

```
push:
```

```
jobs:
```

```
deploy:
```

```
  runs-on: ubuntu-latest
```

```
  container:
```

```
    image: catthehacker/ubuntu:act-22.04
```

```
  steps:
```

```
    - name: Echo Test Message
```

```
      run: echo "Действие запущено в результате push"
```

```
    - name: Checkout repository
```

```
      uses: actions/checkout@v3
```

```
    - name: Setup PHP
```

```
      uses: shivammathur/setup-php@v2
```

```
      with:
```

```
        php-version: '8.1'
```

```
    - name: Install Composer dependencies
```

```
      run: composer install
```

```
    - name: Run PHPUnit tests
```

```
      run: vendor/bin/phpunit tests
```

```
    - name: install ssh keys
```

```
      run: |
```

```
        install -m 600 -D /dev/null ~/.ssh/id_rsa
```

```
        echo "${{ secrets.SSH_PRIVATE_KEY }}" > ~/.ssh/id_rsa
```

```
        ssh-keyscan -H ${{ secrets.SSH_HOST }} > ~/.ssh/known_hosts
```

```
    - name: connect and pull
```

```
    run: ssh ${{ secrets.SSH_USER }}@${{ secrets.SSH_HOST }} "cd  
${{ secrets.WORK_DIR }} && git checkout ${{ secrets.MAIN_BRANCH }} && git pull &&  
cp -f ${{ secrets.FILE1 }} ${{ secrets.APACHE_DIR }} && cp -f ${{ secrets.FILE2 }}  
${{ secrets.APACHE_DIR }} && cp -f ${{ secrets.FILE3 }} ${{ secrets.APACHE_DIR }} &&  
cp -f ${{ secrets.FILE4 }} ${{ secrets.APACHE_DIR }} && echo "${{ secrets.PASS }}" | sudo  
-S systemctl restart apache2 && exit"  
    - name: cleanup  
      run: rm -rf ~/.ssh
```

name: Deploy to Production: Устанавливается название этого Gitea Actions workflow.

on: push: Задает событие, при котором должен запускаться этот workflow - в данном случае, при каждом push'e в репозиторий.

jobs: Определяет список задач (jobs), которые должны выполняться в рамках workflow.

deploy: Определяет задачу с именем "deploy".

runs-on: ubuntu-latest: Указывает, что эта задача должна выполняться на виртуальной машине с операционной системой Ubuntu последней версии.

container: image: catthehacker/ubuntu:act-22.04: Задает контейнер для выполнения задачи, в данном случае, используется образ Ubuntu 22.04.

steps: Определяет последовательность шагов, которые должны выполняться в рамках задачи.

Echo Test Message: Выводит тестовое сообщение в лог.

Checkout repository: Загружает содержимое репозитория в виртуальную машину для последующих шагов.

Setup PHP: Устанавливает PHP версии 8.1.

Install Composer dependencies: Запускает установку зависимостей проекта с использованием Composer.

Run PHPUnit tests: Запускает тесты с использованием PHPUnit.

install ssh keys: Устанавливает ключи SSH, необходимые для подключения к удаленному серверу.

connect and pull: Подключается к удаленному серверу по SSH и выполняет несколько команд:

Переходит в рабочий каталог на сервере.

Выполняет команды Git: переключается на указанную ветку и делает pull (обновление локального репозитория).

Копирует несколько файлов из репозитория в указанный каталог Apache на сервере.

Перезапускает службу Apache2.

Завершает SSH-сеанс.

cleanup: Удаляет временные файлы, связанные с SSH, после завершения задачи.

Затем в каталоге репозитория необходимо создать каталог «tests», в котором будет добавлен тест для проверки работоспособности перед автоматическим развертыванием приложения. В созданном каталоге добавьте файл «CalculatorTest.php», в котором добавьте следующее содержимое:

```
<?php

use PHPUnit\Framework\TestCase;
require_once __DIR__ . '/../calculate.php';

class CalculatorTest extends TestCase
{
    public function testAddNumbers()
    {
        $this->assertEquals(5, addNumbers(2, 3));
        $this->assertEquals(0, addNumbers(0, 0));
        $this->assertEquals(-5, addNumbers(-2, -3));
        $this->assertEquals(7.5, addNumbers(3.5, 4));
    }

    public function testHandlePostRequest()
    {
        $_POST['num1'] = '2';
        $_POST['num2'] = '3';
        ob_start();
        handlePostRequest();
        $output = json_decode(ob_get_clean(), true);

        $this->assertEquals(['result' => 5], $output);
    }

    public function testHandlePostRequestWithInvalidInput()
    {
        $_POST['num1'] = 'invalid';
        $_POST['num2'] = '3';
```

```

ob_start();
handlePostRequest();
$output = json_decode(ob_get_clean(), true);

$this->assertEquals(['error' => 'Пожалуйста, введите числа.'], $output);
$this->assertEquals(400, http_response_code());
}

public function testHandleUnsupportedMethod()
{
    ob_start();
    handleUnsupportedMethod();
    $output = json_decode(ob_get_clean(), true);

    $this->assertEquals(['error' => 'Метод не поддерживается'], $output);
    $this->assertEquals(405, http_response_code());
}
}

```

Для установки зависимостей для тестов добавьте файл composer.json в корне репозитория, в котором укажите необходимые компоненты:

```
{
    "name": "your/vendor-name",
    "description": "Your project description",
    "type": "project",
    "license": "MIT",
    "require": {
        "php": ">=7.4",
        "phpunit/phpunit": "^9.0"
    },
    "autoload": {
        "psr-4": {
            "YourNamespace\\": "src/"
        }
    },
    "require-dev": {
        "phpunit/phpunit": "^9.0"
    },
    "scripts": {
        "test": "phpunit"
    }
}
```

После выполнения всех выше описанных действий произведите push. В веб-интерфейсе на вкладке «Действия» должен отобразиться процесс и после результат выполнения автоматического тестирования и автоматического развертывания на сервере (результат 3.2.7).

The screenshot shows a GitHub Actions deployment log for a workflow named 'deploy'. The status is 'Успешно' (Successful). The log details the following steps:

- > ✓ Set up job 59s
- > ✓ Echo Test Message 1s
- > ✓ Checkout repository 17s
- > ✓ Setup PHP 4m52s
- > ✓ Install Composer dependencies 32m10s
- > ✓ Run PHPUnit tests 13s
- > ✓ **install ssh keys** 9s

```
1 # ***:22 SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.6
2 # ***:22 SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.6
3 # ***:22 SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.6
4 # ***:22 SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.6
5 # ***:22 SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.6
```

- > ✓ **connect and pull** 40s

```
1 Уже на «***»
2 Эта ветка соответствует «origin/**».
3 Из http://***:3000/fio-1111/site
4 56daf9f..b7fe16d ***      -> origin/**
5 Обновление 56daf9f..b7fe16d
6 Fast-forward
7 .gitea/workflows/demo.yml | 8 ++++++-
8 1 file changed, 4 insertions(+), 4 deletions(-)
```

- > ✓ cleanup 1s
- > ✓ Complete job 1m2s

Рисунок 3.2.7 – Результат автоматического тестирования и развертывания

Обратите внимание, что секреты, которые были указаны в коде автоматически заменены на звездочки, что означает, что было подставлено то, значение, которое было указано в соответствующем меню репозитория.

Далее в браузере откройте страницу «<http://IP-адрес сервера:80>», на которой должна отобразиться созданная ранее страница, проверьте работоспособность (рисунок 3.2.8).

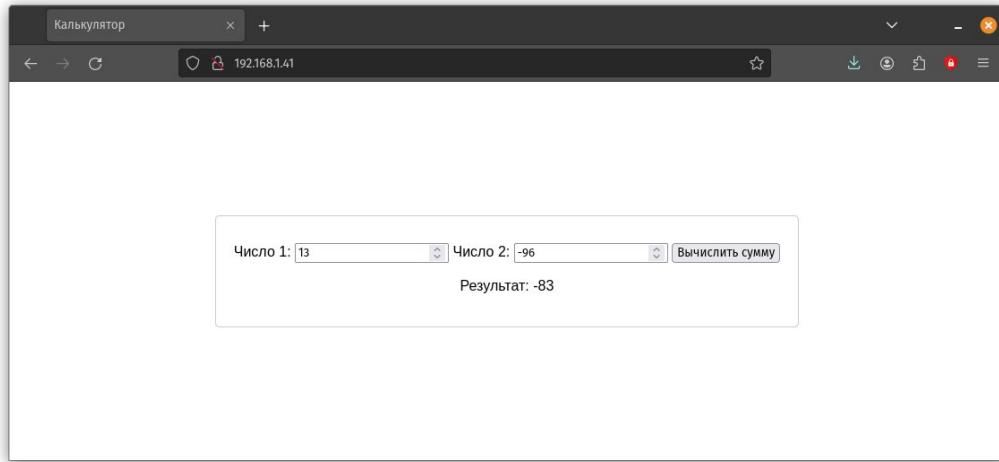


Рисунок 3.2.8 – Созданная страница приложения

Затем через веб-интерфейс gitea внесите изменения в код страницы, как это представлено на рисунке 3.2.9.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Калькулятор</title>
7      <link rel="stylesheet" href="styles.css">
8  </head>
9  <body>
10     <p>Это новые <strong>НОВЫЕ</strong> изменения.</p>
11     <div class="calculator">
12         <form id="calcForm">
13             <label for="num1">Число 1:</label>
14             <input type="number" id="num1" placeholder="Введите число">
15
16             <label for="num2">Число 2:</label>
17             <input type="number" id="num2" placeholder="Введите число">
18
19             <button type="button" onclick="calculateSum()">Вычислить сумму</button>
20         </form>
21
22         <p id="result">Результат: </p>
23     </div>
24
25     <script src="script.js"></script>
26 </body>
27 </html>
```

Рисунок 3.2.9 – Внесение изменений в страницу

После сохранения изменений начнется автоматическое развертывание приложения на сервере. После завершения операции обновите страницу по адресу «<http://IP-адрес сервера:80>» (Рисунок 3.2.10).

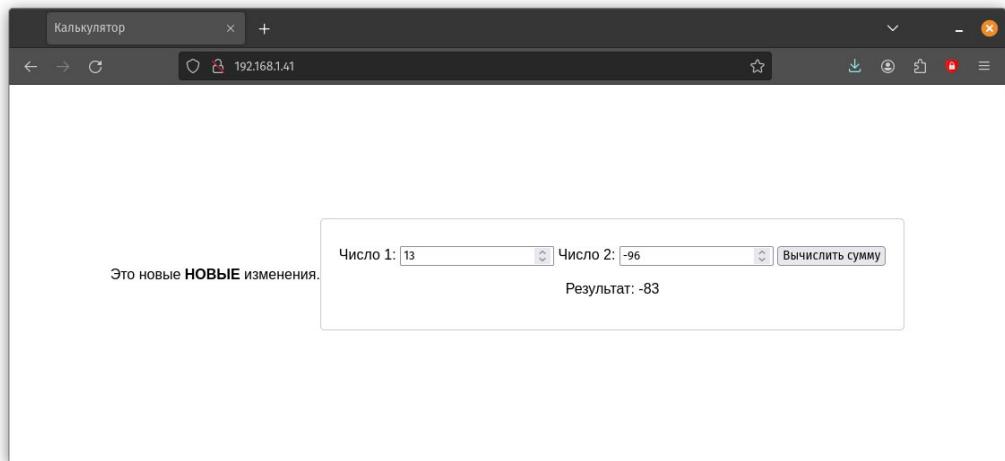


Рисунок 3.2.10 – Результат изменения страницы

Таким образом, можно организовать автоматическое развертывание приложений при наступлении определенного события, например push, pull request, tags, изменения в зависимостях, результаты тестирования, запросы пользователя.

4 Порядок выполнения работы

1. Изучить теоретические сведения.
2. Установить и настроить Gitea на виртуальной машине.
3. Реализовать пример непрерывной интеграции, заменив строку из примера на строки формата «ИмяНомерСтуденческогоБилета». Изменить автоматические тесты в соответствии с данной строкой. В отчете указать пример успешного и неуспешного прохождения теста.
4. Реализовать пример непрерывной доставки, изменив функцию в соответствии с таблицей 4.1. Изменить автоматические тесты в соответствии с данной функцией. В отчете указать пример успешного и неуспешного прохождения теста.
5. Составить отчет о проделанной работе, защитить у преподавателя.

Таблица 4.1 - Индивидуальные задания

Номер варианта	Математическая функция
1	Деление
2	Возведение в степень
3	Логарифм по основанию
4	Остаток от деления
5	Максимум из двух чисел
6	Минимум из двух чисел
7	Гипотенуза прямоугольного треугольника (числа длины сторон)
8	Модуль разности
9	Сделать случайное число в диапазоне от первого числа до второго
10	Сложение с квадратом второго числа
11	Сложение с квадратом первого числа

Продолжение таблицы 4.1

Номер варианта	Математическая функция
12	Разность с корнем из первого числа
13	Разность с корнем из второго числа
14	Сложение с корнем из первого числа
15	Среднее арифметическое

Приложение А

(Обязательное)

Код пользовательского интерфейса

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Калькулятор</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="calculator">
        <form id="calcForm">
            <label for="num1">Число 1:</label>
            <input type="number" id="num1" placeholder="Введите число">

            <label for="num2">Число 2:</label>
            <input type="number" id="num2" placeholder="Введите число">

            <button type="button" onclick="calculateSum()">Вычислить сумму</button>
        </form>

        <p id="result">Результат: </p>
    </div>

    <script src="script.js"></script>
</body>
</html>
```

Приложение Б
(Обязательное)

Код вычисления арифметических функций

```
<?php

function respondWithJson($data, $statusCode = 200) {
    http_response_code($statusCode);
    header('Content-Type: application/json');
    echo json_encode($data);
    exit();
}

function addNumbers($num1, $num2) {
    return $num1 + $num2;
}

function handlePostRequest() {
    try {
        $num1 = floatval($_POST['num1']);
        $num2 = floatval($_POST['num2']);
        $result = addNumbers($num1, $num2);
        respondWithJson(['result' => $result]);
    } catch (Exception $e) {
        respondWithJson(['error' => 'Пожалуйста, введите числа.'], 400);
    }
}

function handleUnsupportedMethod() {
    respondWithJson(['error' => 'Метод не поддерживается'], 405);
}

// Основная логика

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    handlePostRequest();
} else {
    handleUnsupportedMethod();
}
```

Приложение В

(Обязательное)

Код динамического поведения

```
function calculateSum() {
    const num1 = parseFloat(document.getElementById('num1').value);
    const num2 = parseFloat(document.getElementById('num2').value);

    if (!isNaN(num1) && !isNaN(num2)) {
        const formData = new FormData();
        formData.append('num1', num1);
        formData.append('num2', num2);

        fetch('calculate.php', {
            method: 'POST',
            body: formData,
        })
        .then(response => response.json())
        .then(data => {
            document.getElementById('result').textContent = 'Результат: ' + data.result;
        })
        .catch(error => {
            console.error('Ошибка:', error);
            alert('Произошла ошибка при отправке запроса на сервер.');
        });
    } else {
        alert('Пожалуйста, введите числа.');
    }
}
```

Приложение Г
(Обязательное)

Код стилей пользовательского интерфейса

```
body {  
    font-family: Arial, sans-serif;  
    display: flex;  
    justify-content: center;  
    align-items: center;  
    height: 100vh;  
    margin: 0;  
}  
  
.calculator {  
    text-align: center;  
    padding: 20px;  
    border: 1px solid #ccc;  
    border-radius: 5px;  
}  
  
button {  
    margin-top: 10px;  
    cursor: pointer;  
}
```