

Задание 4: CI/CD

Огоре Джоэль Чидике, okorejoel2017@gmail.com

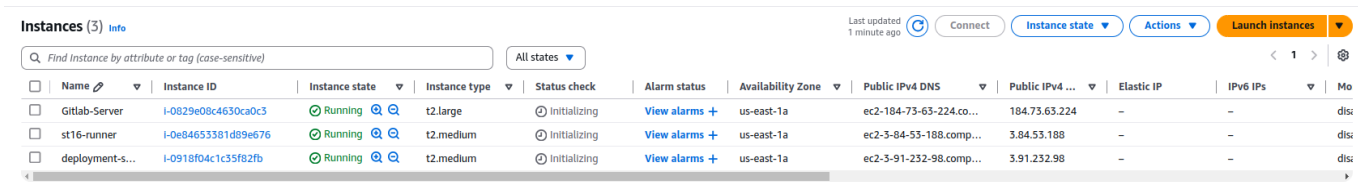
Задание 1: Развертывание

1. Разверните три виртуальные машины, которые вы будете использовать в качестве сервера Gitlab Server, Gitlab Runner и сервера развертывания. Убедитесь, что ВМ могут связаться друг с другом.

Для выполнения этой задачи я развернул на AWS три экземпляра EC2, которые будут служить в качестве сервера GitLab Server, GitLab Runner и сервера развертывания. Вместо ручного развертывания я использовал Terraform для автоматизации инициализации этих экземпляров. Конфигурационные файлы Terraform размещены в следующем репозитории:

GITHUB REPO: [LS-lab-3-CI-CD-Infrastructure - Terraform](#)

Конфигурация обеспечила развертывание всех трех экземпляров в одной сети, что позволило им взаимодействовать друг с другом. В связи с ограничением по времени и обязательным перезапуском инстансов EC2 публичный DNS каждого инстанса менялся несколько раз за время работы. Каждое изменение документировалось в явном виде для поддержания последовательности в рабочем процессе. Подробнее см. рисунок 1.



Instances (3) info										
Find Instance by attribute or tag (case-sensitive)										
All states										
<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
<input type="checkbox"/>	Gitlab-Server	i-0829e08c4630ca0c3	Running	t2.large	Initializing	View alarms +	us-east-1a	ec2-184-73-65-224.co...	184.73.63.224	-
<input type="checkbox"/>	st16-runner	i-0e84653381d89e676	Running	t2.medium	Initializing	View alarms +	us-east-1a	ec2-3-84-53-188.comp...	3.84.53.188	-
<input type="checkbox"/>	deployment-s...	i-0918f04c1c35f82fb	Running	t2.medium	Initializing	View alarms +	us-east-1a	ec2-3-91-232-98.comp...	3.91.232.98	-

Рисунок 1: Снимок экрана консоли управления AWS, на котором показаны три развернутых экземпляра EC2.

2. Настройте сервер Gitlab (VM1) и создайте файл docker-compose с приведенными ниже конфигурациями:

а. Подключите редакцию Gitlab EE или CE

Чтобы установить GitLab Server на первом экземпляре EC2 (VM1), я использовал **Ansible** для автоматизации процесса установки и настройки. Роль `geerlingguy.docker` была использована для упрощения установки Docker на экземпляр GitLab Server EC2. См. рисунок 2.

1. Установка Docker: Для установки Docker на экземпляр GitLab Server использовалась программа Ansible, обеспечивающая готовность среды к контейнеризации.
2. Образ GitLab EE Docker: Образ GitLab Enterprise Edition (EE) Docker был извлечен из реестра Docker с помощью плейбука `deploy_gitlab.yml` Ansible.
3. Конфигурация: Выполненный плейбук создал необходимые каталоги для бесперебойной работы Gitlab (`/srv/gitlab/config`, `/srv/gitlab/logs`, `/srv/gitlab/data`) и обеспечил запуск службы Docker.

Выполнение плейбука Ansible показано на рисунке 3, где показана успешная настройка сервера GitLab с помощью Docker.

```
joel@joel:~/Pictures/LS/Lab-3-CICD/ansible$ ansible-playbook -i inventory playbooks/docker_setup.yml

PLAY [Install Docker on EC2] *****

TASK [Gathering Facts] *****
[WARNING]: Platform linux on host ec2-54-88-25-187.compute-1.amazonaws.com is using the discovered Python interpreter at /usr/bin/python3.12, but future installation of another Python interpreter could change the meaning of that path. See https://docs.ansible.com/ansible-core/2.17/reference_appendices/interpreter_discovery.html for more information.
ok: [ec2-54-88-25-187.compute-1.amazonaws.com]

TASK [geerlingguy.docker : Load OS-specific vars.] *****
ok: [ec2-54-88-25-187.compute-1.amazonaws.com]

TASK [geerlingguy.docker : include tasks] *****
skipping: [ec2-54-88-25-187.compute-1.amazonaws.com]

TASK [geerlingguy.docker : include tasks] *****
included: /home/joel/.ansible/roles/geerlingguy.docker/tasks/setup-Debian.yml for ec2-54-88-25-187.compute-1.amazonaws.com

TASK [geerlingguy.docker : Ensure apt key is not present in trusted.gpg.d] *****
ok: [ec2-54-88-25-187.compute-1.amazonaws.com]

TASK [geerlingguy.docker : Ensure old apt source list is not present in /etc/apt/sources.list.d] *****
ok: [ec2-54-88-25-187.compute-1.amazonaws.com]

TASK [geerlingguy.docker : Ensure the repo referencing the previous trusted.gpg.d key is not present] *****
ok: [ec2-54-88-25-187.compute-1.amazonaws.com]

TASK [geerlingguy.docker : Ensure old versions of Docker are not installed.] *****
ok: [ec2-54-88-25-187.compute-1.amazonaws.com]

TASK [geerlingguy.docker : Ensure dependencies are installed.] *****
ok: [ec2-54-88-25-187.compute-1.amazonaws.com]

TASK [geerlingguy.docker : Ensure directory exists for /etc/apt/keyrings] *****
ok: [ec2-54-88-25-187.compute-1.amazonaws.com]

TASK [geerlingguy.docker : Add Docker apt key.] *****
changed: [ec2-54-88-25-187.compute-1.amazonaws.com]

TASK [geerlingguy.docker : Ensure curl is present (on older systems without SNI).] *****
skipping: [ec2-54-88-25-187.compute-1.amazonaws.com]

TASK [geerlingguy.docker : Add Docker apt key (alternative for older systems without SNI).] *****
skipping: [ec2-54-88-25-187.compute-1.amazonaws.com]

TASK [geerlingguy.docker : Add Docker repository.] *****
changed: [ec2-54-88-25-187.compute-1.amazonaws.com]

TASK [geerlingguy.docker : Install Docker packages.] *****
skipping: [ec2-54-88-25-187.compute-1.amazonaws.com]
```

Рисунок 2: Установка Docker на экземпляре GitLab Server с помощью роли `geerlingguy.docker`

```
joel@joel:~/Pictures/LS/Lab-3-CICD/ansible$ ansible-playbook -i inventory playbooks/deploy_gitlab.yml

PLAY [Deploy GitLab EE using Docker] *****

TASK [Gathering Facts] *****
[WARNING]: Platform linux on host ec2-54-88-25-187.compute-1.amazonaws.com is using the discovered Python interpreter at /usr/bin/python3.12, but future installation of another Python interpreter could change the meaning of that path. See https://docs.ansible.com/ansible-core/2.17/reference_appendices/interpreter_discovery.html for more information.
ok: [ec2-54-88-25-187.compute-1.amazonaws.com]

TASK [my role : Pull GitLab EE Docker image] *****
changed: [ec2-54-88-25-187.compute-1.amazonaws.com]

TASK [my role : Create GitLab data directories] *****
changed: [ec2-54-88-25-187.compute-1.amazonaws.com] => (item=src/gitlab/config)
changed: [ec2-54-88-25-187.compute-1.amazonaws.com] => (item=src/gitlab/logs)
changed: [ec2-54-88-25-187.compute-1.amazonaws.com] => (item=src/gitlab/data)

TASK [my role : Ensure Docker service is running] *****
ok: [ec2-54-88-25-187.compute-1.amazonaws.com]

PLAY RECAP *****
ec2-54-88-25-187.compute-1.amazonaws.com : ok=4 changed=2 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```

Рисунок 3: Выполнение плейбука Ansible для развертывания GitLab EE с помощью Docker.

b. Задача 2b-е: Настройка GitLab Server с помощью Docker-Compose

Сервер GitLab Server был настроен с помощью файла `docker-compose.yml` , как показано на рисунке 4. Основные настройки включали в себя:

- Именованние контейнера:
 - Запущенный контейнер был назван `st16-gitlab` , как указано.
- Сопоставление портов:
 - Порты контейнера были сопоставлены с хост-машиной следующим образом:
 - `80:80` для HTTP-доступа.
 - `443:443` для доступа по протоколу HTTPS.
 - `2424:22` для доступа по SSH.
- Конфигурация окружения:
 - Переменная окружения `external_url` была установлена на публичный DNS сервера GitLab, предоставленный AWS (`http://${GITLAB_PUBLIC_DNS}`).
 - Были применены дополнительные конфигурации GitLab, такие как отключение Prometheus, Mattermost и реестра, а также оптимизация настроек Puma и Sidekiq, которые были рекомендованы для сред с ограниченными ресурсами, как в моем случае.
- Привязка томов:
 - Необходимые каталоги были привязаны к хост-машине:
 - `/srv/gitlab/config/etc/gitlab` для файлов конфигурации.
 - `/srv/gitlab/logs/var/log/gitlab` для журналов.
 - `/srv/gitlab/data/var/opt/gitlab` для данных приложения.

Сервер GitLab Server не мог быть открыт как `st16.sne.com` , поскольку вместо него использовался публичный DNS, предоставляемый AWS. Этот DNS менялся при каждом перезапуске экземпляра, и переменная `${GITLAB_PUBLIC_DNS}` динамически обновлялась в файле `docker-compose.yml` , чтобы отразить это изменение.

```
GNU nano 7.2 docker-compose.yml *
version: '3.8'

services:
  gitlab:
    image: gitlab/gitlab-ee:latest
    container_name: st16-gitlab
    restart: always
    hostname: "${GITLAB_PUBLIC_DNS}"
    environment:
      GITLAB_OMNIBUS_CONFIG: |
        external_url "http://${GITLAB_PUBLIC_DNS}"
        gitlab_rails['gitlab_shell_ssh_port'] = 2424
        prometheus['enable'] = false
        mattermost['gitlab_enable'] = false
        registry['enable'] = false
        puma['worker_processes'] = 0
        sidekiq['max_concurrency'] = 10
    ports:
      - "80:80"
      - "443:443"
      - "2424:22"
    volumes:
      - /srv/gitlab/config:/etc/gitlab
      - /srv/gitlab/logs:/var/log/gitlab
      - /srv/gitlab/data:/var/opt/gitlab
    shm_size: '256m'
```

Рисунок 4. Файл `docker-compose.yml`, используемый для настройки сервера GitLab.

f. Запустите файл `docker-compose` и убедитесь, что конфигурации работают

Файл `docker-compose.yml` был выполнен с помощью команды `sudo docker compose up -d`, которая запустила контейнер GitLab Server в отсоединенном режиме. Результат этого процесса показан на рисунке 5.

Контейнер `st16-gitlab` был успешно запущен и стал доступен на прописанных портах:

- `80/tcp` для HTTP.
- `443/tcp` для HTTPS.
- `2424/tcp` для SSH.

Я также получил начальный пароль `root` для экземпляра GitLab с помощью команды ниже. В дальнейшем он будет использоваться для входа на сервер.

```
sudo docker exec -it st16-gitlab grep 'Password:' /etc/gitlab/initial_root_password
```

```
ubuntu@ip-172-31-86-79:~$ sudo docker compose up -d
WARN[0000] /home/ubuntu/docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Running 1/1
✔ Container st16-gitlab Started 0.3s
ubuntu@ip-172-31-86-79:~$ sudo docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
63dfae6f47aa   gitlab/gitlab-ee:latest             "/assets/wrapper"       12 minutes ago Up 36 seconds (health: starting)  0.0.0.0:80->80/tcp, :::80->80/tcp, 0.0.0.0:443->443/tcp, :::443->443/tcp, 0.0.0.0:2424->22/tcp, [::]:2424->22/tcp   st16-gitlab
ubuntu@ip-172-31-86-79:~$
ubuntu@ip-172-31-86-79:~$ sudo docker exec -it st16-gitlab grep 'Password:' /etc/gitlab/initial_root_password
Password: kxyBtT+Hun8RXIwaXLINQNh8qZU64VVeUspnMTfPN4=
ubuntu@ip-172-31-86-79:~$
```

Рисунок 5: Выполнение файла `docker-compose.yml` и получение начального корневого пароля.

g. Зайдите на сервер Gitlab и авторизуйтесь, создайте проект, назвав его `<stx>-repo`.

Доступ к серверу GitLab осуществлялся через браузер с использованием публичного DNS экземпляра EC2. Для входа использовались начальные учетные данные `root` (имя пользователя: `root`, пароль: получен ранее), как показано на рисунке 6.

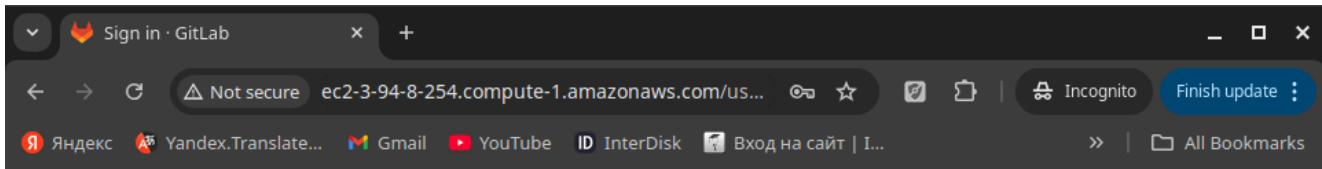


Рисунок 6: Страница входа в GitLab

Затем был создан новый проект под названием `st16-repo` с уровнем видимости Private, чтобы запретить доступ без аутентификации. Репозиторий был инициализирован путем включения файла `README.md` для немедленного клонирования. Процесс создания проекта показан на рисунке 7.

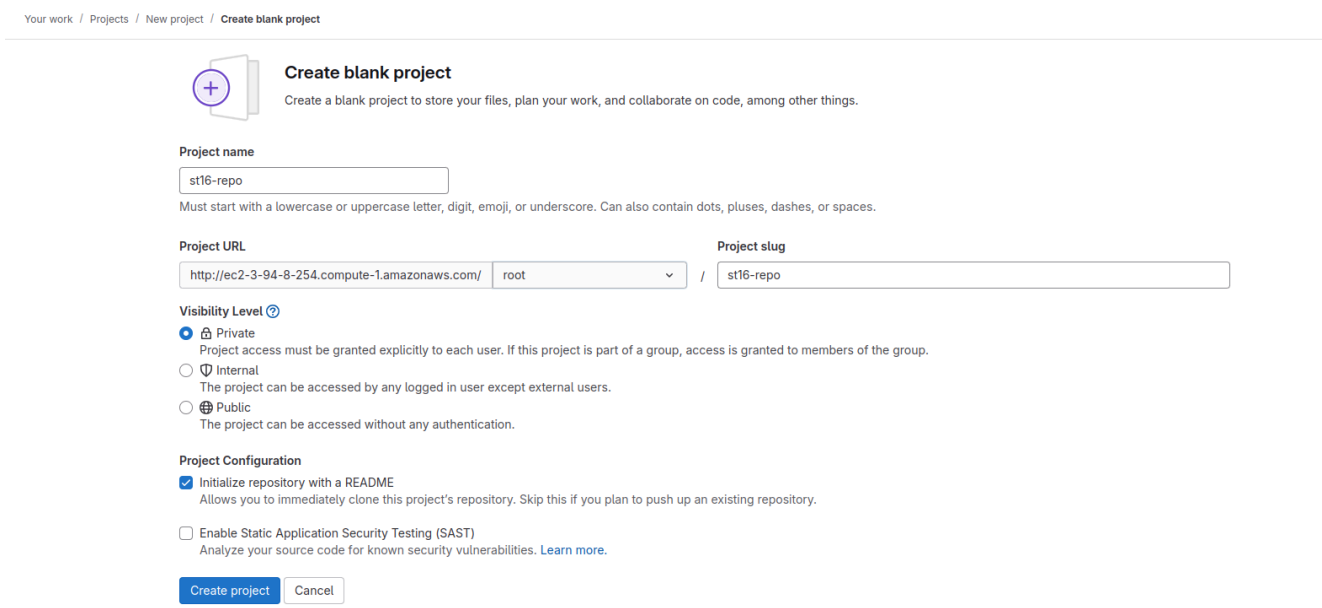


Рисунок 7: Страница создания проекта для `st16-repo`

После создания проекта на панели инструментов отобразилась информация о начальном коммите и репозитории, как показано на рисунке 8.

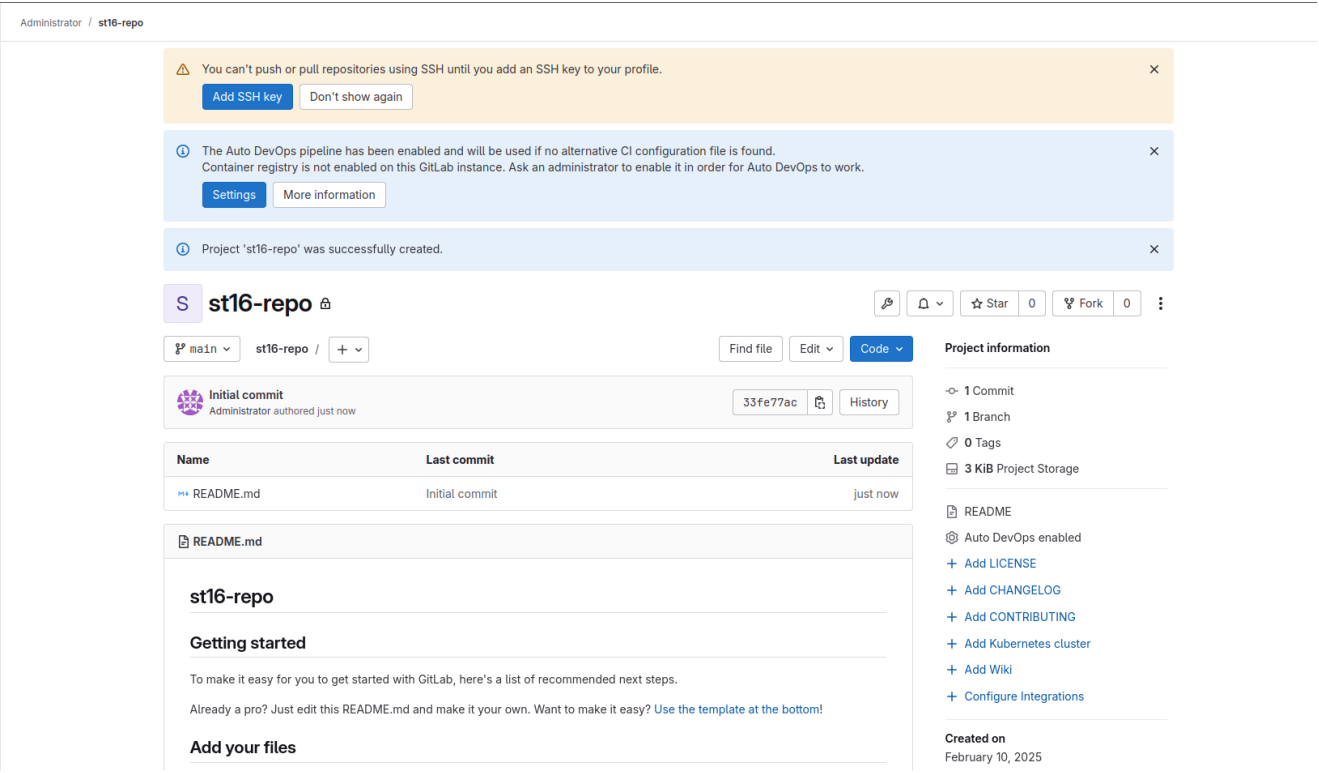


Рисунок 8: Панель успешно созданного проекта `st16-repo`

3. Настройте Gitlab Runner (VM2), в этот раз не используйте подход docker.

а. Установка и настройка общего Gitlab Runner

Для выполнения этой задачи публичные DNS и IP-адреса GitLab Server, GitLab Runner и Deployment Server были сначала получены с помощью Terraform, как показано на рисунке 9.

```
joel@joel:~/Pictures/LS/Lab-3-CICD/terraform$ terraform apply
aws_security_group.gitlab-runner-sg: Refreshing state... [id=sg-0053e899add25f0d0]
aws_security_group.gitlab-sg: Refreshing state... [id=sg-04eedc432e40f07df]
aws_instance.gitlab-runner: Refreshing state... [id=i-0a1f77d9df1bac55b]
aws_instance.gitlab-server: Refreshing state... [id=i-038001ae8bef7b518]

Changes to Outputs:
+ gitlab_runner_public_dns = "ec2-54-205-235-21.compute-1.amazonaws.com"
+ gitlab_runner_public_ip  = "54.205.235.21"

You can apply this plan to save these new output values to the Terraform state, without changing any real infrastructure.

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

Outputs:
gitlab_public_dns = "ec2-3-94-8-254.compute-1.amazonaws.com"
gitlab_public_ip  = "3.94.8.254"
gitlab_runner_public_dns = "ec2-54-205-235-21.compute-1.amazonaws.com"
gitlab_runner_public_ip  = "54.205.235.21"
joel@joel:~/Pictures/LS/Lab-3-CICD/terraform$
```

Рисунок 9: Результаты работы Terraform, показывающие публичные DNS и IP-адреса экземпляров EC2.

SSH-доступ к экземпляру GitLab Runner

(`ec2-54-205-235-21.compute-1.amazonaws.com`) был установлен с помощью предоставленного файла ключей (`labsuser.pem`), как показано на рисунке 10.

```
joel@joel:~/Downloads$ ssh -i labsuser.pem ubuntu@ec2-54-205-235-21.compute-1.amazonaws.com
The authenticity of host 'ec2-54-205-235-21.compute-1.amazonaws.com (54.205.235.21)' can't be established.
ED25519 key fingerprint is SHA256:Ei6NWzZw053JqrFn9SAhacZkrLZMK8jvBpMnnNAhMDQ.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-54-205-235-21.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.8.0-1021-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information as of Mon Feb 10 08:09:00 UTC 2025

System load:  0.06          Processes:            124
Usage of /:   3.5% of 47.39GB Users logged in:          0
Memory usage: 5%          IPv4 address for enx0: 172.31.84.136
Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
```

Рисунок 10: Доступ к экземпляру GitLab Runner с помощью SSH

Получив доступ к экземпляру GitLab Runner с помощью SSH, я выполнил сценарий установки GitLab Runner с помощью следующей команды:

```
sudo curl -L "https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh" | sudo bash
```

Этот сценарий добавил репозиторий GitLab Runner и подготовил среду к установке, как показано на рисунке 11.

```
ubuntu@ip-172-31-84-136:~$ sudo curl -L "https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh" | sudo bash
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left   Speed
100 6885 100 6885    0     0  74021    0 --:--:-- --:--:-- --:--:-- 74836
Detected operating system as Ubuntu/noble.
Checking for curl...
Detected curl...
Checking for gpg...
Detected gpg...
Running apt-get update... done.
Installing apt-transport-https... done.
Installing /etc/apt/sources.list.d/runner_gitlab-runner.list... done.
Importing packagecloud gpg key... done.
Running apt-get update... done.
```

Рисунок 11: Выполнение сценария установки GitLab Runner.

После добавления репозитория GitLab Runner для установки пакета GitLab Runner был использован инструмент Advanced Package Tool (APT). Была выполнена следующая команда:

```
sudo apt install gitlab-runner
```

Установка прошла успешно, и служба GitLab Runner была настроена, как показано на рисунке 12.


```
ubuntu@ip-172-31-84-136:~$ sudo apt install gitlab-runner
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  gitlab-runner-helper-images
Suggested packages:
  docker-engine
The following NEW packages will be installed:
  gitlab-runner gitlab-runner-helper-images
0 upgraded, 2 newly installed, 0 to remove and 92 not upgraded.
Need to get 536 MB of archives.
After this operation, 616 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 https://packages.gitlab.com/runner/gitlab-runner/ubuntu noble/main amd64 gitlab-runner-helper-images all 17.8.3-1 [510 MB]
Get:2 https://packages.gitlab.com/runner/gitlab-runner/ubuntu noble/main amd64 gitlab-runner amd64 17.8.3-1 [26.0 MB]
Fetched 536 MB in 8s (71.0 MB/s)
Selecting previously unselected package gitlab-runner-helper-images.
(Reading database ... 70614 files and directories currently installed.)
Preparing to unpack .../gitlab-runner-helper-images_17.8.3-1_all.deb ...
Unpacking gitlab-runner-helper-images (17.8.3-1) ...
Selecting previously unselected package gitlab-runner.
Preparing to unpack .../gitlab-runner_17.8.3-1_amd64.deb ...
Unpacking gitlab-runner (17.8.3-1) ...
Setting up gitlab-runner-helper-images (17.8.3-1) ...
Setting up gitlab-runner (17.8.3-1) ...
GitLab Runner: creating gitlab-runner...
Home directory skeleton not used
Runtime platform                                arch=amd64 os=linux pid=2106 revision=690ce25c version=17.8.3
gitlab-runner: the service is not installed
Runtime platform                                arch=amd64 os=linux pid=2113 revision=690ce25c version=17.8.3
gitlab-ci-multi-runner: the service is not installed
Runtime platform                                arch=amd64 os=linux pid=2132 revision=690ce25c version=17.8.3
Runtime platform                                arch=amd64 os=linux pid=2229 revision=690ce25c version=17.8.3
INFO: Docker installation not found, skipping clear-docker-cache
Scanning processes...
Scanning linux images...
```

Рисунок 12: Процесс установки GitLab Runner с помощью APT

б. Объясните, что такое исполнитель Gitlab Runner, и установите тип исполнителя на shell

Исполнитель GitLab Runner определяет среду, в которой выполняются задания. Для этой задачи тип исполнителя был установлен на shell, который запускает задания непосредственно на хост-машине без контейнеризации. Это было настроено в процессе регистрации runner, как показано на рисунке 15.

с. Установите тег Gitlab Runner в <stx>-runner. (Вы будете использовать этот тег в конвейере в следующем задании).

Бегун GitLab Runner был помечен как `st16-runner`, чтобы идентифицировать его для определенных заданий в конвейере CI/CD. Этот тег был добавлен в процессе создания бегуна в пользовательском интерфейсе GitLab, как показано на рисунках 13 и 14

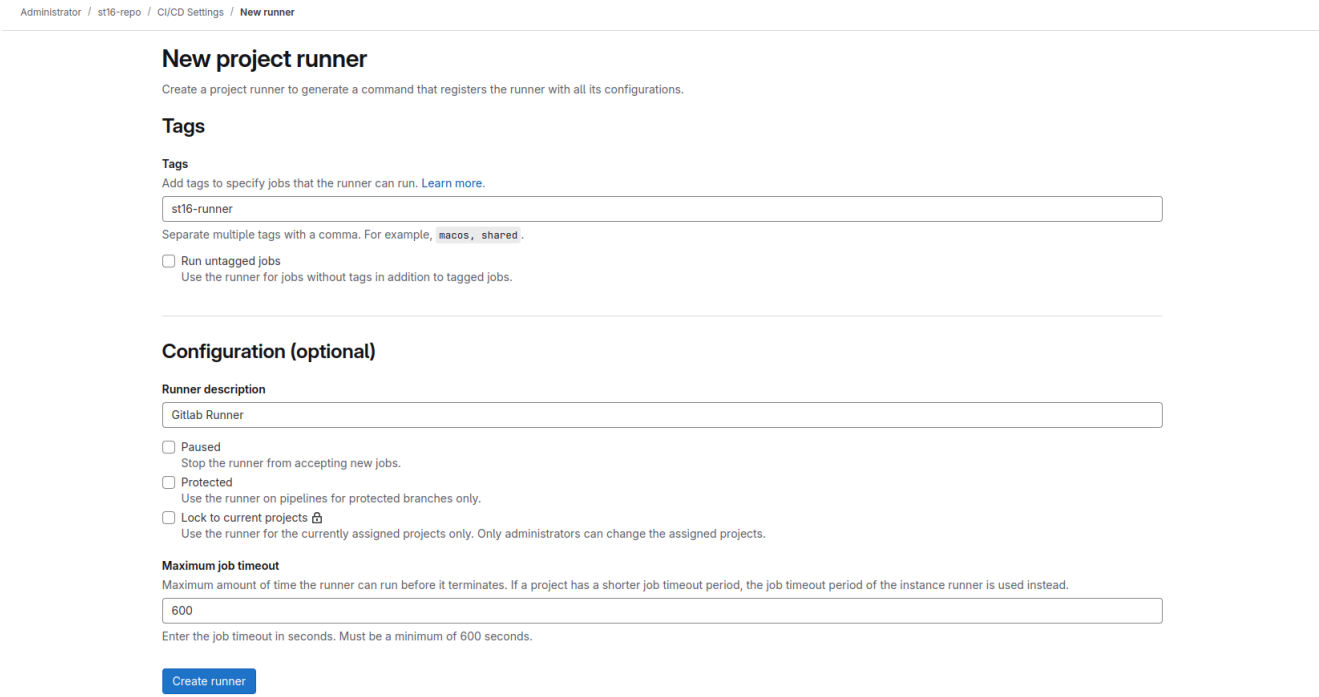


Рисунок 13: Пользовательский интерфейс GitLab, показывающий создание бегуна `st16-runner` с тегом `st16-runner`

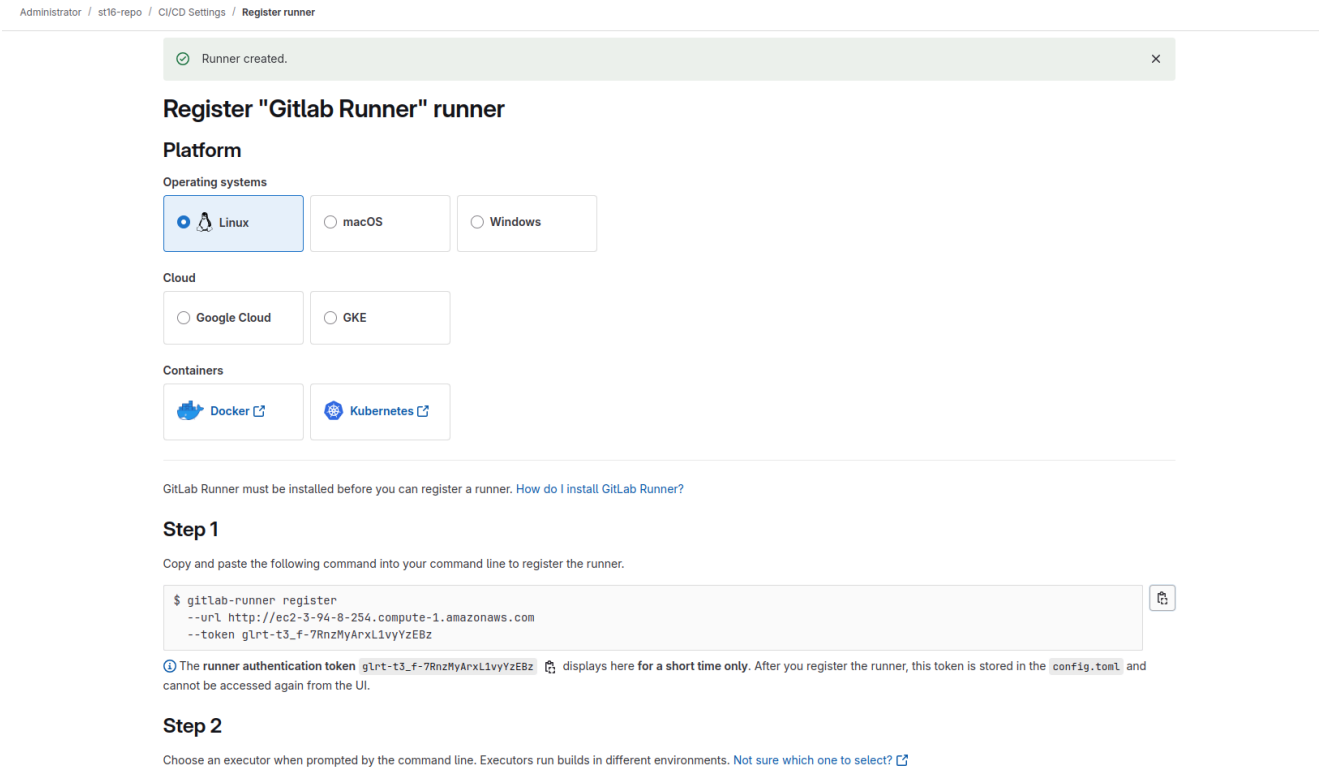


Рисунок 14: Пользовательский интерфейс GitLab показывает команду регистрации бегуна и токен.

d. Аутентифицируйте свой Gitlab runner на сервере Gitlab и проверьте.

Бегун GitLab был зарегистрирован с помощью следующей команды. Бегун получил имя `st16-runner` , а исполнителем была выбрана оболочка, как уже говорилось ранее.

```
gitlab-runner register --url http://ec2-3-94-8-254.compute-1.amazonaws.com --token glrt-t3_f-7RnzMyArxLllyYz
```

После этого бегун был запущен, и его конфигурация была проверена. Процесс регистрации и проверки показан на рисунке 15.

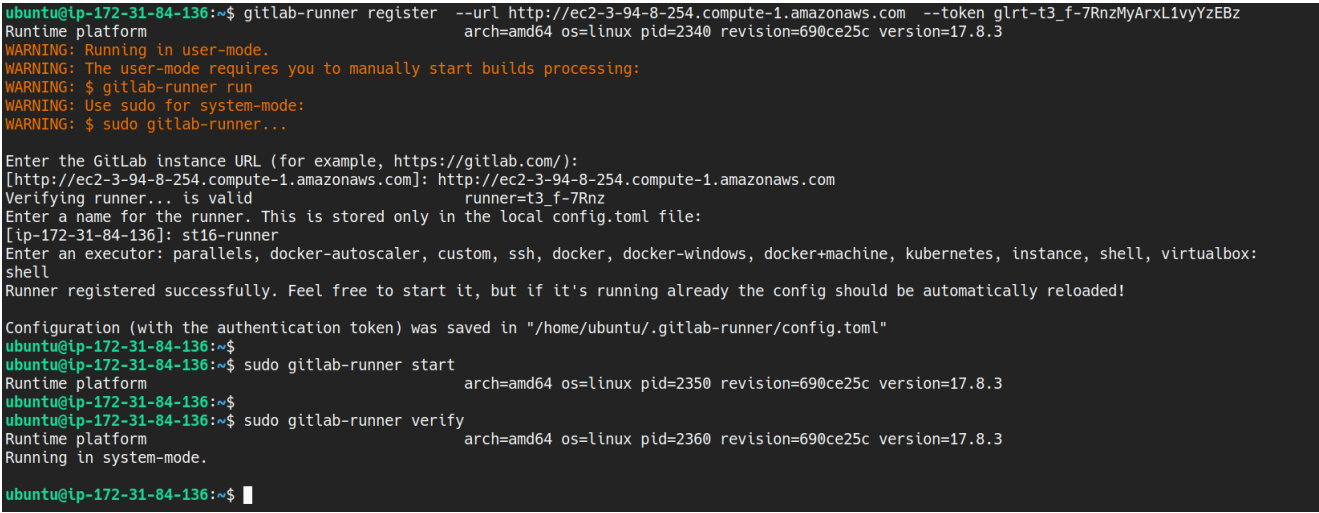


Рисунок 15: Терминал, показывающий процесс регистрации, запуска и проверки бегуна.

4. Настройте сервер развертывания (VM3).

а. Настройте аутентификацию вашего бегуна Gitlab, чтобы он мог развертываться на сервере развертывания.

Как было сказано ранее, машины были автоматически перезапущены из-за нехватки времени. Вот новые конфигурации IP и публичного DNS. Для получения более подробной информации обратитесь к рисунку 16.

```
deployment_server_public_dns = "ec2-52-23-226-234.compute-1.amazonaws.com"
gitlab_public_dns = "ec2-3-94-8-254.compute-1.amazonaws.com"
gitlab_runner_public_dns = "ec2-54-205-235-21.compute-1.amazonaws.com"
```



```
Changes to Outputs:
+ deployment_server_public_dns = (known after apply)
+ deployment_server_public_ip  = (known after apply)

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

aws_security_group.deployment-server-sg: Creating...
aws_security_group.deployment-server-sg: Creation complete after 5s [id=sg-057244551c7aa0242]
aws_instance.deployment-server: Creating...
aws_instance.deployment-server: Still creating... [10s elapsed]
aws_instance.deployment-server: Creation complete after 15s [id=i-0f2b735460ac8d7bb]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

deployment_server_public_dns = "ec2-52-23-226-234.compute-1.amazonaws.com"
deployment_server_public_ip = "52.23.226.234"
gitlab_public_dns = "ec2-3-94-8-254.compute-1.amazonaws.com"
gitlab_public_ip = "3.94.8.254"
gitlab_runner_public_dns = "ec2-54-205-235-21.compute-1.amazonaws.com"
gitlab_runner_public_ip = "54.205.235.21"
joel@joel:~/Pictures/LS/Lab-3-CICD/terraform$
```

Рисунок 16: Результаты работы Terraform, показывающие публичные DNS и IP инстансов EC2

Чтобы настроить аутентификацию Gitlab runner на Deployment Server, на экземпляре GitLab Runner была сгенерирована пара ключей RSA с помощью следующей команды. Более подробная информация приведена на рисунке 17.

```
ssh-keygen -t rsa -b 4096 -C 'st16-runner'
```

```
ubuntu@ip-172-31-84-136:~$ ssh-keygen -t rsa -b 4096 -C "st16-runner"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ubuntu/.ssh/id_rsa):
/home/ubuntu/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ubuntu/.ssh/id_rsa
Your public key has been saved in /home/ubuntu/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:UC0/aohC4q3mQFPs6Tsy06rX/kw0BW01eRtc+aqWqeI st16-runner
The key's randomart image is:
+---[RSA 4096]-----+
|  .  +0.0..  |
| 0 0.+= 0   |
| 0 +..+. = . |
| 0 +  +0+ 0 .|
| . 0 .0.50 0 |
| .  ....0 .  |
| ..0.... +   |
| +=+00  =    |
| 0==0Eo+0    |
+-----[SHA256]-----+
ubuntu@ip-172-31-84-136:~$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQADNGMSqK05KcCw9b2225Cr4F2fTfj+dyebfcJUL/Gs97nvbZV0g+aJpa2Rv0gLEncy8Knahh6gWA72VtdFrZ99AX+Mrq9Mg2moa/FHlU4fJti6t4H26pCN0lrTPQGo+Rtz3JD
FoHqm9ISpLG/P5g94a4DKTlw3pcRLaLURXncWpG4i+afXKFJ92H9smk4mTNXlodFDoHm9KHlJ/CDRVXc5z96eZ1B1B4CNBVq48Er6pYfZryththxqlprizhn+5aCz5H/TQm+LIxmayK8rYTWf8/ucWwrtIqYfv8kjpZzPqfHosx
ocnTK8DL7v8BTpt2t8nmrUteVzQ4nXgntB7RIm0l+QpCQ0F+ZdGggnpzm0L3tKluHmxQmeEC21KwklxwmMAwveHpnFaozh4ELXMJ6wEc5Yw41pqCms1mRIBkXQRRGdQSHatrP1W1fdC8jR0l4bVnXL/2Fz/P+ZzH9zYklee5ruz
emy25vVnPW4H5wLYxyxeZ0pTo8wPMRddXhVv834YnzWNFI7d1EngxtQmh380lu9hkSIyIFHzFWRryUVl6Mg2t36uVY0+8zI7osHelwLcTdqG83bkeemdkw98DdDmeNkwBHxX1CJl8WRW4CqxENYIuIJFzSwTZ+uhZFx71L/Vv5
pd15sEa3eIfao86dzoxf7Yb/X30sm8dy8nxJQ== st16-runner
ubuntu@ip-172-31-84-136:~$
```

Рисунок 17: Генерация пары ключей RSA для GitLab Runner

Открытый ключ был добавлен в файл `~/.ssh/authorized_keys` на сервере развертывания, чтобы включить SSH-аутентификацию, как показано на рисунке 18.

```
ubuntu@ip-172-31-89-127:~$ echo "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQADNGMSqK05KcCw9b2225Cr4F2fTfj+dyebfcJUL/Gs97nvbZV0g+aJpa2Rv0gLEncy8Knahh6gWA72VtdFrZ99AX+Mrq9Mg2moa/FHlU4fJti6t4H26pCN0lrTPQGo+Rtz3JD
FoHqm9ISpLG/P5g94a4DKTlw3pcRLaLURXncWpG4i+afXKFJ92H9smk4mTNXlodFDoHm9KHlJ/CDRVXc5z96eZ1B1B4CNBVq48Er6pYfZryththxqlprizhn+5aCz5H/TQm+LIxmayK8rYTWf8/ucWwrtIqYfv8kjpZzPqfHosx
ocnTK8DL7v8BTpt2t8nmrUteVzQ4nXgntB7RIm0l+QpCQ0F+ZdGggnpzm0L3tKluHmxQmeEC21KwklxwmMAwveHpnFaozh4ELXMJ6wEc5Yw41pqCms1mRIBkXQRRGdQSHatrP1W1fdC8jR0l4bVnXL/2Fz/P+ZzH9zYklee5ruz
emy25vVnPW4H5wLYxyxeZ0pTo8wPMRddXhVv834YnzWNFI7d1EngxtQmh380lu9hkSIyIFHzFWRryUVl6Mg2t36uVY0+8zI7osHelwLcTdqG83bkeemdkw98DdDmeNkwBHxX1CJl8WRW4CqxENYIuIJFzSwTZ+uhZFx71L/Vv5
pd15sEa3eIfao86dzoxf7Yb/X30sm8dy8nxJQ== st16-runner" > ~/.ssh/authorized_keys
ubuntu@ip-172-31-89-127:~$ sudo chmod 600 ~/.ssh/authorized_keys
ubuntu@ip-172-31-89-127:~$
```

Рисунок 18: Добавление сгенерированного GitLab Runner открытого ключа в файл `authorized_keys` на сервере развертывания

Далее, SSH-доступ с GitLab Runner на сервер развертывания был успешно протестирован, что подтвердило работоспособность настроек аутентификации, как показано на рисунке 19.

```
ubuntu@ip-172-31-84-136:~$ ssh ubuntu@ec2-52-23-226-234.compute-1.amazonaws.com
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.8.0-1021-aws x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:       https://ubuntu.com/pro

System information as of Mon Feb 10 09:03:08 UTC 2025

System load:  0.0          Processes:      116
Usage of /:   3.6% of 47.39GB Users logged in: 1
Memory usage: 5%          IPv4 address for enX0: 172.31.89.127
Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

Last login: Mon Feb 10 08:51:26 2025 from 5.189.254.17
ubuntu@ip-172-31-89-127:~$
```

Рисунок 19: Успешный SSH-доступ к серверу развертывания из GitLab Runner

Задача 2: Создание CI/CD конвейера

1. Клонировать проект, который вы создали в шаге 1.2.g.

Созданный ранее проект GitLab (`st16-repo`) был клонирован в экземпляр GitLab Runner с помощью следующей команды:

```
git clone http://ec2-3-94-8-254.compute-1.amazonaws.com/root/st16-repo
```

Репозиторий был успешно клонирован, и файл `README.md` был проверен в клонированном каталоге, как показано на рисунке 20.

```
ubuntu@ip-172-31-84-136:~$ git clone http://ec2-3-94-8-254.compute-1.amazonaws.com/root/st16-repo
Cloning into 'st16-repo'...
Username for 'http://ec2-3-94-8-254.compute-1.amazonaws.com': root
Password for 'http://root@ec2-3-94-8-254.compute-1.amazonaws.com':
warning: redirecting to http://ec2-3-94-8-254.compute-1.amazonaws.com/root/st16-repo.git/
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
ubuntu@ip-172-31-84-136:~$ ls
st16-repo
ubuntu@ip-172-31-84-136:~$ cd st16-repo/
ubuntu@ip-172-31-84-136:~/st16-repo$ ls
README.md
ubuntu@ip-172-31-84-136:~/st16-repo$ sudo nano app.py
ubuntu@ip-172-31-84-136:~/st16-repo$ sudo nano requirements.txt
ubuntu@ip-172-31-84-136:~/st16-repo$ sudo cat requirements.txt
Flask==2.2.2
ubuntu@ip-172-31-84-136:~/st16-repo$
```

Рисунок 20: Успешное клонирование репозитория `st16-repo`

2. Напишите простое веб-приложение на любом языке программирования. (Например, случайный текст или сложение двух чисел).

Простое веб-приложение было написано на языке Python с использованием фреймворка Flask. Приложение предоставляет два маршрута:

- 1. Главный маршрут (`/`):
 - Отображает приветственное сообщение: `"Welcome to Joe!'s Root App!"`.
- 2. Корневой маршрут (`/root`):
 - Принимает параметр запроса `num` и возвращает квадратный корень из заданного числа.
 - Обрабатывает неверный ввод, возвращая сообщение об ошибке: `"Недопустимый ввод. Укажите единственное число"`.

Приложение настроено на запуск на `0.0.0.0` на порту `5000`. Код показан на рисунке 21.

```

app.py > add_numbers
1  from flask import Flask, request, jsonify
2  import math
3
4  app = Flask(__name__)
5
6  Tabnine | Edit | Test | Explain | Document
7  @app.route('/')
8  def home():
9      return "Welcome to Joel's Root App!"
10
11  Tabnine | Edit | Test | Explain | Document
12  @app.route('/root', methods=['GET'])
13  def add_numbers():
14      try:
15          num = int(request.args.get('num', 0))
16          return jsonify({"result": math.sqrt(int(num))})
17      except ValueError:
18          return jsonify({"error": "Invalid input. Provide a single number."}), 400
19
20  if __name__ == '__main__':
21      app.run(host='0.0.0.0', port=5000)
22

```

Рисунок 21: Файл `app.py`, содержащий код веб-приложения Flask.

3. Создание CI/CD-конвейера (.gitlab-ci.yml)

а. Этапы CI конвейера должны:

- i. Собрать приложение
- ii. Запустить тест (чтобы проверить, что приложение работает нормально)
- iii. Собрать образ docker (Примечание: вам нужен Dockerfile).
- iv. Отправить на ваш аккаунт docker hub.

б. CD-этапы конвейера должны:

- i. Извлечь образ докера и развернуть его на сервере развертывания.

Конвейер CI/CD был определен в файле `.gitlab-ci.yml`, который включает в себя этапы **CI (Continuous Integration)** и **CD (Continuous Deployment)**. Ниже приведено описание этапов конвейера и их функциональных возможностей:

Все этапы CI/CD были помечены для запуска на бегуне с меткой `st16-runner`

Этапы CI:

1. Build Stage:

- Проверяет версию Python для обеспечения совместимости.

2. Этап тестирования:

- Создает виртуальное окружение Python и устанавливает зависимости из файла `requirements.txt`.
- Выполняет модульные тесты с помощью файла `test_app.py`, которые включают:
 - Тестирование маршрута `home (/)` на успешный ответ и корректное содержимое.
 - Тестирование маршрута `/root` с правильными и неправильными входными данными для обеспечения правильной функциональности и обработки ошибок.

3. Этап сборки Docker:

- Сборка образа Docker с использованием предоставленного `Dockerfile`.
- В `Dockerfile` указываются:
 - Базовый образ `python:3.12-alpine`.
 - Копирование файлов приложения в контейнер.
 - Установка зависимостей и открытие порта `5000`.

4. Стадия Docker Push:

- Вход в Docker Hub с использованием учетных данных, хранящихся в переменных GitLab CI/CD (`$DOCKER_USERNAME` и `$DOCKER_PASSWORD`).
- Помещает собранный Docker-образ (`joellots/app:v1`) на Docker Hub.
- Запускается только на `основной` ветке.

Этапы работы с CD

1. Этап развертывания:

- Подключается к серверу развертывания по SSH, используя его публичный DNS (`$DEPLOYMENT_SERVER_DNS`).
- Извлекает образ Docker из Docker Hub и запускает его в отсоединенном режиме, сопоставляя порт `80` на хосте с портом `5000` в контейнере.
- Запускается только на `основной` ветке.

Полный проект, включая файл `.gitlab-ci.yml` (в репозитории он называется `.gitlab-ci-first.yml`), `Dockerfile` и код приложения, доступен в репозитории:



[LS-lab-3-CI-CD-Infrastructure - st16-repo.](#)

Поддерживающие файлы

1. Тестовый файл (`test_app.py`): Содержит модульные тесты для приложения Flask, обеспечивающие корректную работу маршрута `home` и маршрута `/root`. Для получения более подробной информации обратитесь к рисунку 22.

```
test_app.py > ...
1  import unittest
2  from app import app
3
4  class TestAdditionApp(unittest.TestCase):
5
6      def setUp(self):
7          self.app = app.test_client()
8
9      def test_home(self):
10         response = self.app.get('/')
11         self.assertEqual(response.status_code, 200)
12         self.assertIn(b'Welcome to Joel\'s Root App!', response.data)
13
14     def test_root_number(self):
15         response = self.app.get('/root?num=64')
16         self.assertEqual(response.status_code, 200)
17         self.assertIn(b'"result":8.0', response.data)
18
19     def test_invalid_input(self):
20         response = self.app.get('/root?num=abc')
21         self.assertEqual(response.status_code, 400)
22         self.assertIn(b'Invalid input', response.data)
23
24 if __name__ == '__main__':
25     unittest.main()
26
```

Рисунок 22: Файл `test_app.py`, содержащий модульные тесты для приложения Flask.

2. `Dockerfile`: Определяет шаги по контейнеризации приложения Flask. Более подробная информация приведена на рисунке 23.

```

Dockerfile > FROM
1 FROM python:3.12-alpine
2
3 WORKDIR /app
4
5 COPY . .
6
7 RUN pip install -r requirements.txt
8
9 EXPOSE 5000
10
11 CMD ["python", "app.py"]
12
13

```

Рисунок 23: `Dockerfile`, используемый для создания образа Docker.

4. Убедитесь, что развертывание прошло успешно, зайдя в веб-приложение через браузер на стороне сервера развертывания.

В последний раз машины были автоматически перезагружены из-за нехватки времени. Вот новые конфигурации IP и публичного DNS. Более подробную информацию см. на рисунке 24.



`deployment_server_public_dns = "ec2-34-238-252-26.compute-1.amazonaws.com"`

`gitlab_public_dns = "ec2-3-95-26-207.compute-1.amazonaws.com"`

`gitlab_runner_public_dns = "ec2-52-91-103-252.compute-1.amazonaws.com"`

```

Changes to Outputs:
- deployment_server_public_dns = "ec2-44-204-36-72.compute-1.amazonaws.com" -> "ec2-34-238-252-26.compute-1.amazonaws.com"
- deployment_server_public_ip = "44.204.36.72" -> "34.238.252.26"
- gitlab_public_dns = "ec2-54-84-140-215.compute-1.amazonaws.com" -> "ec2-3-95-26-207.compute-1.amazonaws.com"
- gitlab_public_ip = "54.84.140.215" -> "3.95.26.207"
- gitlab_runner_public_dns = "ec2-44-202-46-251.compute-1.amazonaws.com" -> "ec2-52-91-103-252.compute-1.amazonaws.com"
- gitlab_runner_public_ip = "44.202.46.251" -> "52.91.103.252"

You can apply this plan to save these new output values to the Terraform state, without changing any real infrastructure.

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

Outputs:
deployment_server_public_dns = "ec2-34-238-252-26.compute-1.amazonaws.com"
deployment_server_public_ip = "34.238.252.26"
gitlab_public_dns = "ec2-3-95-26-207.compute-1.amazonaws.com"
gitlab_public_ip = "3.95.26.207"
gitlab_runner_public_dns = "ec2-52-91-103-252.compute-1.amazonaws.com"
gitlab_runner_public_ip = "52.91.103.252"
joel@joel:~/Pictures/L5/Lab-3-CICD/terraform$

```

Рисунок 24: Результаты работы Terraform, показывающие публичные DNS и IP экземпляров EC2

Для проверки развертывания первым шагом было размещение всех необходимых файлов (`app.py`, `test_app.py`, `Dockerfile`, `.gitlab-ci.yml` и `requirements.txt`) в репозитории `st16-repo` на сервере GitLab. Это потребовало установки восходящей ветки и решения проблем с хранилищем учетных данных, как показано на рисунках 25 и 26.


```
• (venv) joel@joel:~/Downloads/st16-repo$ git remote add origin http://ec2-3-95-26-207.compute-1.amazonaws.com/root/st16-repo
• (venv) joel@joel:~/Downloads/st16-repo$ git push
fatal: The current branch main has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin main

To have this happen automatically for branches without a tracking
upstream, see 'push.autoSetupRemote' in 'git help config'.
• (venv) joel@joel:~/Downloads/st16-repo$ git push -u origin main
fatal: No credential store has been selected.

Set the GCM_CREDENTIAL_STORE environment variable or the credential.credentialStore Git configuration setting to one of the following options:

secretservice : freedesktop.org Secret Service (requires graphical interface)
gpg           : GNU 'pass' compatible credential storage (requires GPG and 'pass')
cache        : Git's in-memory credential cache
plaintext    : store credentials in plain-text files (UNSECURE)
none         : disable internal credential storage

See https://aka.ms/gcm/credstores for more information.

fatal: No credential store has been selected.

Set the GCM_CREDENTIAL_STORE environment variable or the credential.credentialStore Git configuration setting to one of the following options:

secretservice : freedesktop.org Secret Service (requires graphical interface)
gpg           : GNU 'pass' compatible credential storage (requires GPG and 'pass')
cache        : Git's in-memory credential cache
plaintext    : store credentials in plain-text files (UNSECURE)
none         : disable internal credential storage

See https://aka.ms/gcm/credstores for more information.

warning: redirecting to http://ec2-3-95-26-207.compute-1.amazonaws.com/root/st16-repo.git/
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 368 bytes | 368.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
To http://ec2-3-95-26-207.compute-1.amazonaws.com/root/st16-repo
    fd56c44..85a2ad8  main -> main
branch 'main' set up to track 'origin/main'.
• (venv) joel@joel:~/Downloads/st16-repo$
```

Рисунок 25: Размещение файлов в репозитории `st16-repo`

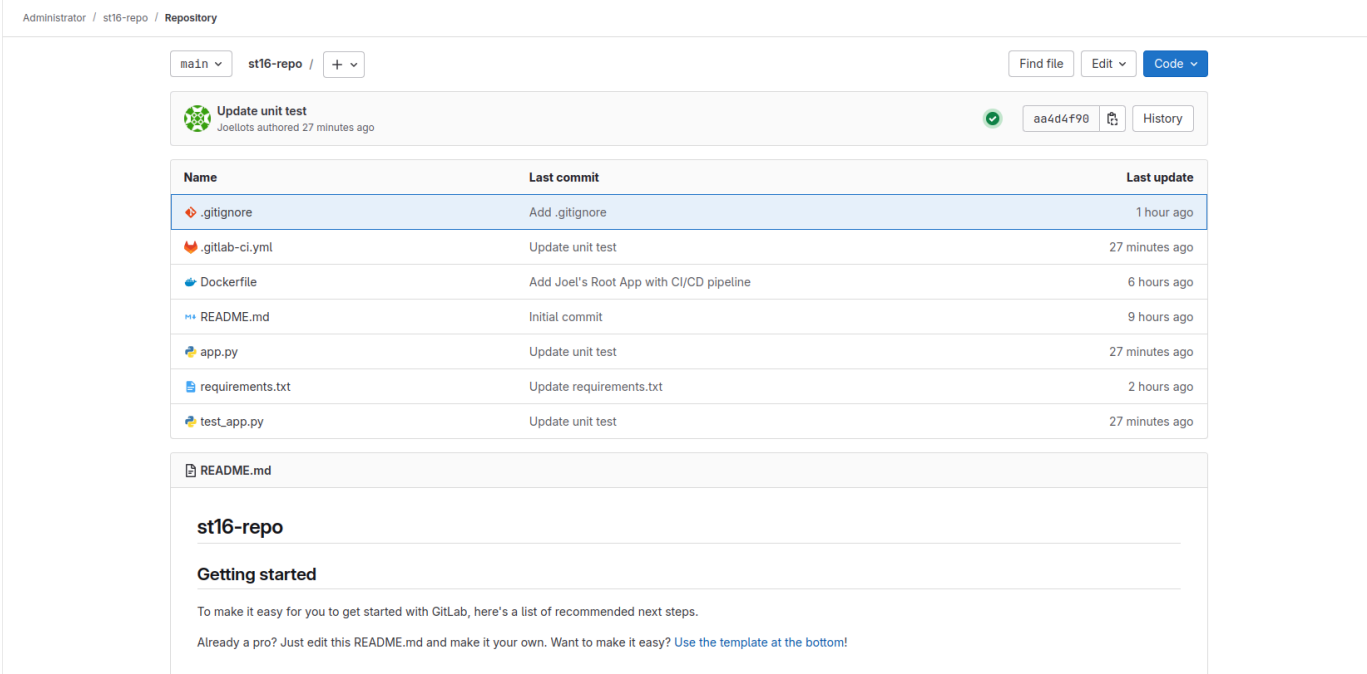


Рисунок 26: Репозиторий GitLab, показывающий обновленные файлы и историю коммитов.

После того как файлы были размещены, следующим шагом была настройка необходимых переменных CI/CD в настройках проекта GitLab. Эти переменные включают `DEPLOYMENT_SERVER_DNS`, `DOCKER_USERNAME` и `DOCKER_PASSWORD`, которые необходимы конвейеру для аутентификации в Docker Hub и развертывания на Deployment Server. Эта настройка показана на рисунке 27.

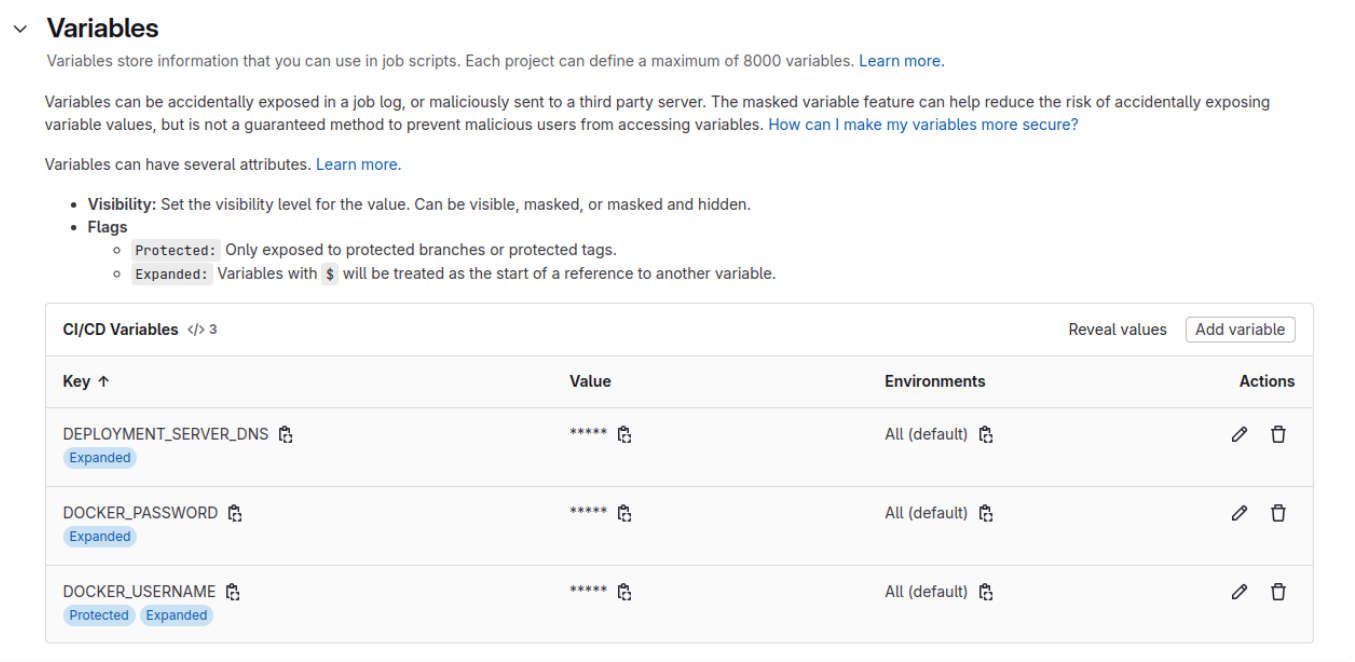


Рисунок 27: Конфигурация переменных GitLab CI/CD

Когда переменные были установлены, ранее созданный `st16-runner` был назначен проекту `st16-repo`. Это гарантировало, что бегун, который был специально помечен для этого проекта, будет обрабатывать задания CI/CD. Назначение бегуна показано на рисунке 28.

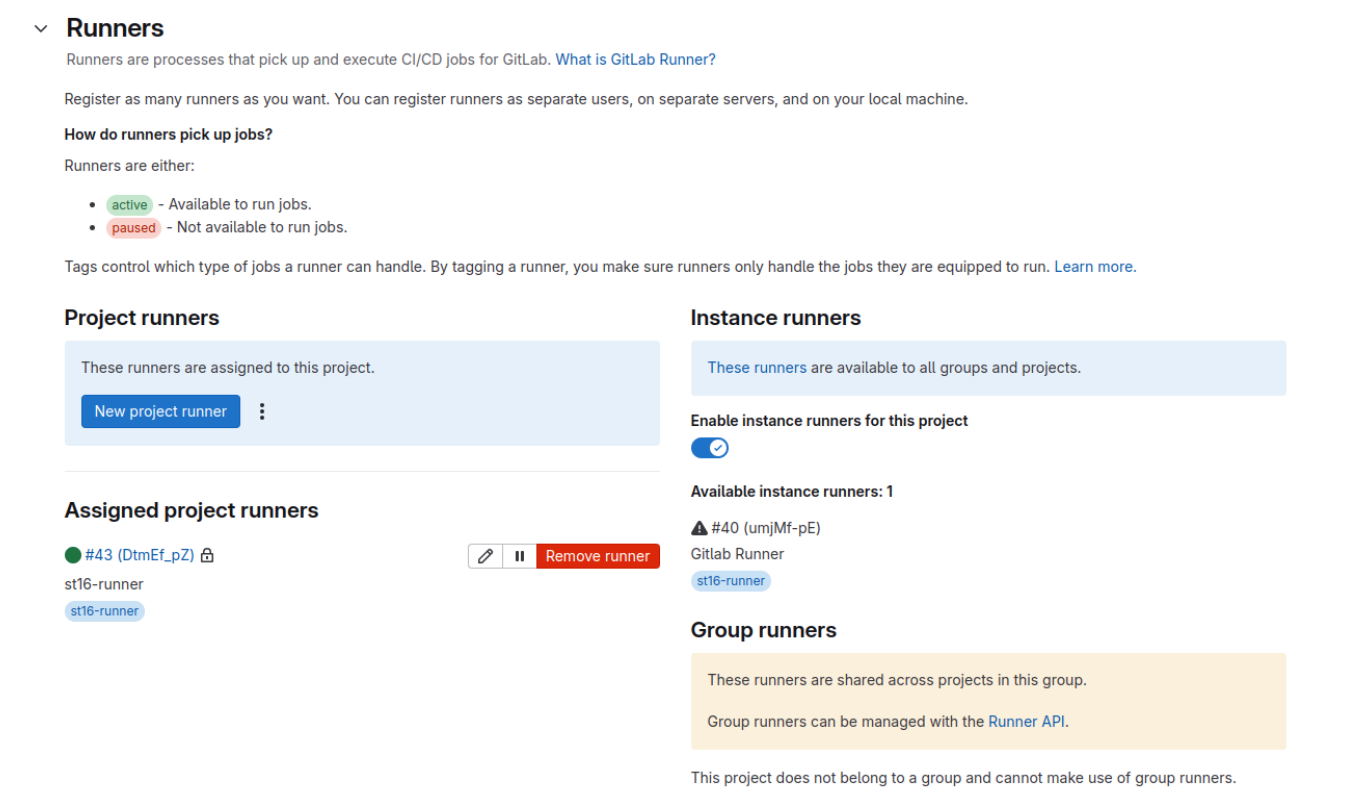


Рисунок 28: Страница GitLab Runners, на которой показан назначенный для проекта `st16-runner`.

После этих настроек запуск файлов привел в действие конвейер CI/CD. Конвейер `успешно` выполнил все этапы - `сборку`, `тестирование`, `docker_build`, `docker_push` и `развертывание`. Каждый этап выполнял свою задачу: от сборки и тестирования приложения до сборки и отправки Docker-образа и, наконец, его развертывания на сервере развертывания. Успешное выполнение конвейера показано на рисунке 29.

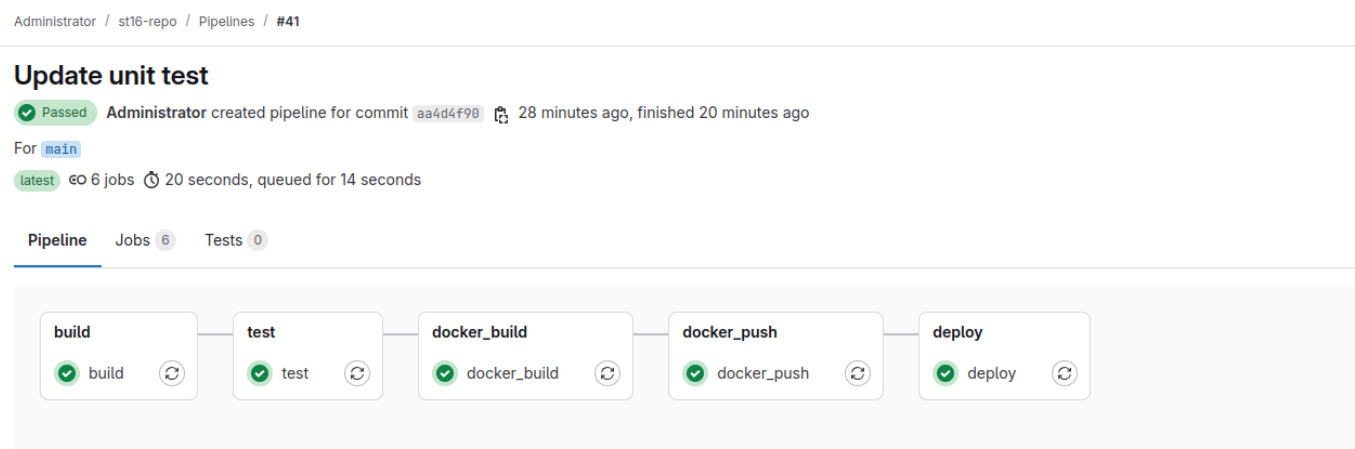


Рисунок 29: Конвейер GitLab, показывающий успешное прохождение всех этапов

Для проверки разворачивания была использована команда `curl` для взаимодействия с развернутым веб-приложением. Были выполнены следующие команды:

- Доступ к главному маршруту

```
curl http://ec2-34-238-252-26.compute-1.amazonaws.com/
```

В ответ было получено следующее: `Welcome to Joel's Root App!`, подтверждающий, что приложение запущено.

- Доступ к корневому маршруту

```
curl http://ec2-34-238-252-26.compute-1.amazonaws.com/root?num=318096
```

Ответ был: `{"result":564.0}`, подтверждая, что маршрут `/root` функционирует правильно.

Эти результаты, показанные на рисунке 30, подтвердили, что приложение успешно развернуто и работает на сервере разворачивания.

```
joel@joel:~/Downloads$ curl http://ec2-34-238-252-26.compute-1.amazonaws.com/
Welcome to Joel's Root App! joel@joel:~/Downloads$
joel@joel:~/Downloads$
joel@joel:~/Downloads$ curl http://ec2-34-238-252-26.compute-1.amazonaws.com/root?num=318096
{"result":564.0}
joel@joel:~/Downloads$
```

Рисунок 30: Терминал, показывающий команды `curl` и ответы, подтверждающие разворачивание.

Задача 3: Отполировать CI/CD



Github Repo Ссылка на файл `.gitlab-ci.yml` для Polished CI/CD: <https://github.com/Joellots/LS-lab-3-CI-CD-Infrastructure/blob/main/st16-repo/.gitlab-ci-polished.yml>.

5. Обновите этапы CD, чтобы иметь возможность разворачивать веб-приложение с помощью Ansible.

Этап CD (Continuous Deployment) был обновлен для использования Ansible для разворачивания веб-приложения. Это изменение было сделано для дальнейшей автоматизации процесса разворачивания и обеспечения согласованности в разных средах.

deploy.yml (Ansible Playbook)

Файл `deploy.yml` определяет задачи для разворачивания приложения, созданного в Docker:

1. Извлечь образ Docker:

- Плейбук извлекает образ Docker `{{ image_name }}:{{ image_tag }}` из Docker Hub с помощью модуля `docker_image`.

2. Запустить контейнер Docker:

- Модуль `docker_container` используется для запуска контейнера со следующими конфигурациями:
 - Port Mapping: Сопоставляет порт `80` на хосте с портом `5000` в контейнере.
 - Политика перезапуска: Обеспечивает автоматический перезапуск контейнера в случае его остановки.

Этот плейбук выполняется на этапе `разворачивания` конвейера CI/CD, как показано в файле `.gitlab-ci.yml`.



GITHUB REPO: <https://github.com/Joellots/LS-lab-3-CI-CD-Infrastructure/blob/main/st16-repo/deploy.yml>

6. Обновите конвейер для поддержки нескольких ветвей (например, master и develop), и задания должны запускаться на основе конкретной целевой ветви.

Конвейер был обновлен для поддержки нескольких ветвей (main и develop) и запуска заданий на основе целевой ветви. Это было достигнуто с помощью ключевого слова rules в файле .gitlab-ci.yml.

Ключевые изменения:

1. Выполнение с учетом ветки:

- Задания на этапах docker_build, docker_push и deploy выполняются только в том случае, если коммит находится в ветке main или develop.

```
rules:
- if: '$CI_COMMIT_BRANCH == "main" || $CI_COMMIT_BRANCH == "develop"'
```

2. Поддержка нескольких ветвей: Это гарантирует, что только стабильные ветки запускают полный конвейер, в то время как функциональные ветки могут использоваться для разработки и тестирования без развертывания в производство.

7. Обновление ключевых слов, таких как cache, artifact, needs и dependencies, чтобы иметь больше контроля над выполнением конвейера.

В конвейер были добавлены такие ключевые слова, как cache, artifacts, needs и dependencies, чтобы оптимизировать выполнение и улучшить контроль над зависимостями заданий.

Ключевые обновления:

1. Кэш:

- Ключевое слово cache используется для хранения зависимостей (например, venv/ и .cache/pip/) между заданиями, что сокращает время сборки.

```
cache:
paths:
- venv/
- .cache/pip/
```

2. Артефакты:

- На этапе сборки создаются артефакты (например, каталог venv/), которые передаются на последующие этапы.

```
artifacts:
paths:
- venv/
```

3. Потребности: Ключевое слово needs используется для определения зависимостей между заданиями, позволяя запускать задания не по порядку, если их зависимости выполнены.



- Этап тестирования зависит от этапа сборки.
- Этап docker_push зависит от этапа docker_build.
- Этап развертывания зависит от этапа docker_push.

4. Зависимости: Благодаря явному указанию зависимостей конвейер гарантирует, что задания будут выполняться только после завершения их предварительных условий, что повышает эффективность.

Поток выполнения

- Этап сборки:
 - Настраивает окружение Python и устанавливает зависимости.
 - Кэширует каталог `venv/` для повторного использования на последующих этапах.
- Этап тестирования:
 - Запускает модульные тесты, используя кэшированную директорию `venv/`.
 - Зависит от этапа `сборки`.
- Этап сборки Docker:
 - Собирает образ Docker, если фиксация происходит на `main` или `develop`.
- Docker Push Stage:
 - Выталкивает образ Docker в Docker Hub, если коммит находится на `main` или `develop`.
 - Зависит от стадии `docker_build`.
- Этап развертывания:
 - Использует Ansible для развертывания докеризованного приложения на сервере развертывания.
 - Зависит от стадии `docker_push`.

После редактирования файла `gitlab-ci.yml` я решил выложить все необходимые файлы в репозиторий `st16-repo` на сервере GitLab, как показано на рисунках 31 и 32.

```

joel@joel:~/Downloads/st16-repo$ git add .
joel@joel:~/Downloads/st16-repo$ git commit -m "Polish CICD"
[main f4c896e] Polish CICD
1 file changed, 1 insertion(+)
joel@joel:~/Downloads/st16-repo$ git push -u origin main
fatal: No credential store has been selected.

Set the GCM_CREDENTIAL_STORE environment variable or the credential.credentialStore Git configuration setting to one of the following options:

secretservice : freedesktop.org Secret Service (requires graphical interface)
gpg            : GNU 'pass' compatible credential storage (requires GPG and 'pass')
cache         : Git's in-memory credential cache
plaintext     : store credentials in plain-text files (UNSECURE)
none          : disable internal credential storage

See https://aka.ms/gcm/credstores for more information.

fatal: No credential store has been selected.

Set the GCM_CREDENTIAL_STORE environment variable or the credential.credentialStore Git configuration setting to one of the following options:

secretservice : freedesktop.org Secret Service (requires graphical interface)
gpg            : GNU 'pass' compatible credential storage (requires GPG and 'pass')
cache         : Git's in-memory credential cache
plaintext     : store credentials in plain-text files (UNSECURE)
none          : disable internal credential storage

See https://aka.ms/gcm/credstores for more information.

Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 305 bytes | 305.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
To http://ec2-18-208-134-162.compute-1.amazonaws.com/root/st16-repo.git
 897d9e2..f4c896e  main -> main
branch 'main' set up to track 'origin/main'.
joel@joel:~/Downloads/st16-repo$

```

Рисунок 31: Размещение файлов в репозитории `st16-repo`

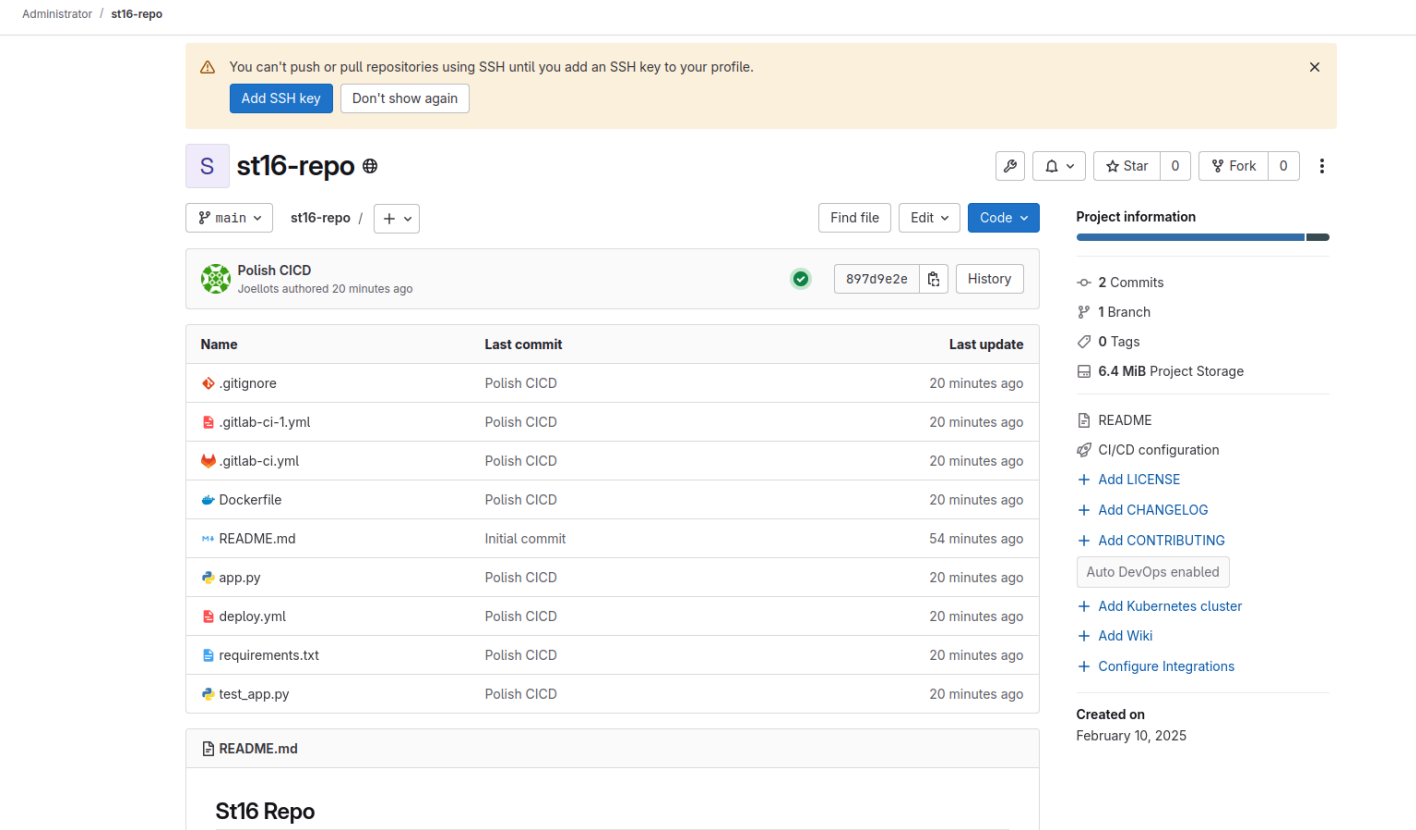


Рисунок 32: Репозиторий GitLab, показывающий обновленные файлы и историю коммитов.

После этого размещение файлов запустило конвейер CI/CD. Конвейер **успешно** выполнил все **этапы** - сборку, **тестирование**, **docker_build**, **docker_push** и **развертывание**. Успешное выполнение конвейера показано на рисунке 33.

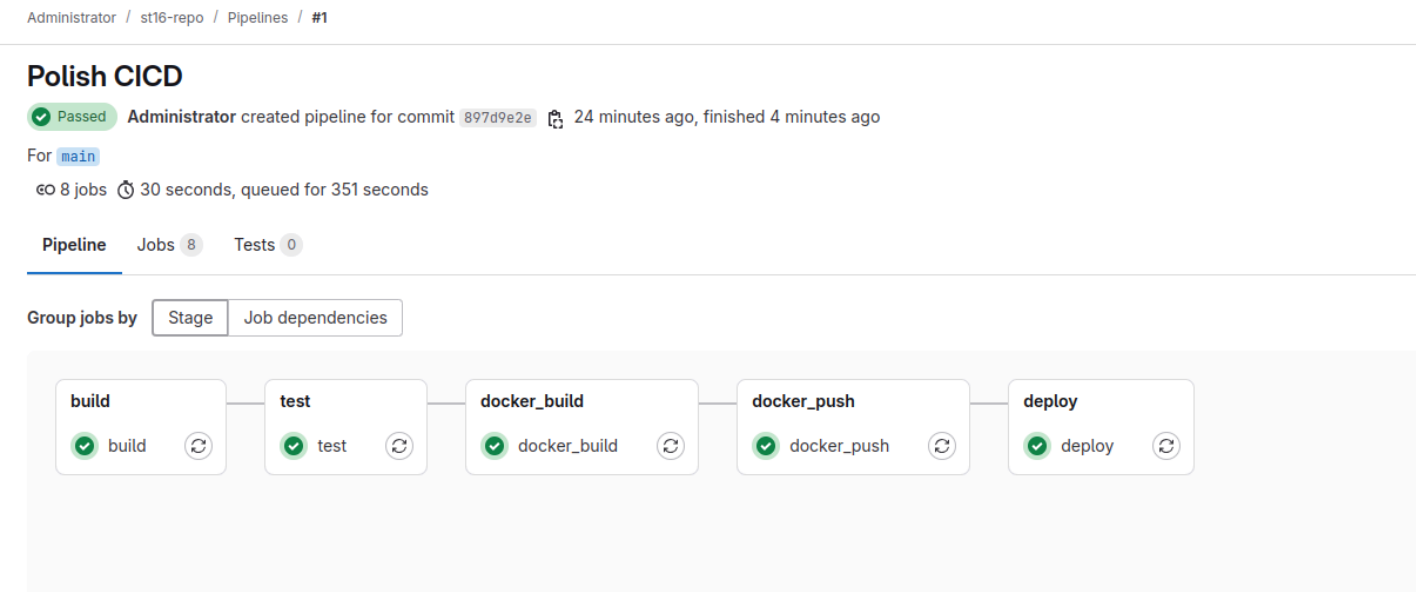


Рисунок 33: Отполированный конвейер GitLab, показывающий успешное прохождение всех этапов

Для проверки развертывания была использована команда **curl** для взаимодействия с развернутым веб-приложением. Были выполнены следующие команды:

- Доступ к главному маршруту

```
curl http://ec2-3-84-92-5.compute-1.amazonaws.com/
```

Ответ был следующим: **Welcome to Joel's Root App!**, подтверждающий, что приложение запущено.

- Доступ к корневому маршруту

```
curl http://ec2-3-84-92-5.compute-1.amazonaws.com/root?num=81
```

Ответ был: **{ "result":9.0 }**, подтверждая, что маршрут **/root** работает правильно.

Эти результаты, показанные на рисунке 34, подтвердили, что приложение было успешно развернуто с помощью полированного конвейера CI/CD и работает на сервере развертывания.


```
joel@joel:~/Downloads$ curl http://ec2-3-84-92-5.compute-1.amazonaws.com/
Welcome to Joel's Root App!joel@joel:~/Downloads$
joel@joel:~/Downloads$
joel@joel:~/Downloads$ curl http://ec2-3-84-92-5.compute-1.amazonaws.com/root?num=81
{"result":9.0}
joel@joel:~/Downloads$
```

Рисунок 34: Терминал, показывающий команды `curl` и ответы, подтверждающие развертывание.

Задача 4: Сценарий резервного копирования и аварийного восстановления

8. Запуск онлайн/горячего резервного копирования сервера Gitlab

Для обеспечения безопасности данных было инициировано онлайн/горячее резервное копирование сервера GitLab. Этот процесс резервного копирования гарантирует, что экземпляр GitLab может быть восстановлен в случае потери данных или сбоя сервера.

Команда `sudo docker ps -a` была использована для получения списка всех контейнеров Docker и идентификации контейнера GitLab (`st16-gitlab`).

Затем я выполнил приведенную ниже команду, чтобы создать резервную копию сервера GitLab:

```
sudo docker exec -it st16-gitlab gitlab-backup create
```

Эта команда запускает встроенную утилиту резервного копирования GitLab, которая создает резервную копию базы данных, репозиториев и конфигураций. Более подробная информация приведена на рисунке 35.

```
ansible: bash x Downloads: bash x Gitlab x Gitlab Runner x Deployment Server x
ubuntu@ip-172-31-86-106:~$ sudo docker ps -a
CONTAINER ID   IMAGE      NAMES      COMMAND      CREATED      STATUS      PORTS
39860aa457083   gitlab/gitlab-ee:latest   "/assets/wrapper"   2 hours ago   Up 2 hours (healthy)   0.0.0.0:80->80/tcp, :::80->80/tcp, 0.0.0.0:443->443/tcp, :::443->443/tcp, 0.0.0.0:2424->22/tcp, [::]:2424->22/tcp
ubuntu@ip-172-31-86-106:~$ sudo docker exec -it st16-gitlab gitlab-backup create
2025-02-10 23:33:57 UTC -- Dumping database ...
2025-02-10 23:33:57 UTC -- Dumping PostgreSQL database gitlabhq_production ...
2025-02-10 23:34:03 UTC -- [DONE]
2025-02-10 23:34:03 UTC -- Dumping database ... done
2025-02-10 23:34:03 UTC -- Dumping repositories ...
{"command":"create","gl_project_path":"root/st16-repo","level":"info","msg":"started create","pid":3564,"relative_path":"@hashed/6b/86/6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b.wiki.git","storage_name":"default","time":"2025-02-10T23:34:03.616Z"}
{"command":"create","gl_project_path":"root/st16-repo.wiki","level":"info","msg":"started create","pid":3564,"relative_path":"@hashed/6b/86/6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b.wiki.git","storage_name":"default","time":"2025-02-10T23:34:03.617Z"}
{"command":"create","gl_project_path":"root/st16-repo.wiki","level":"info","msg":"completed create","pid":3564,"relative_path":"@hashed/6b/86/6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b.wiki.git","storage_name":"default","time":"2025-02-10T23:34:03.646Z"}
{"command":"create","gl_project_path":"root/st16-repo","level":"info","msg":"completed create","pid":3564,"relative_path":"@hashed/6b/86/6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b.git","storage_name":"default","time":"2025-02-10T23:34:03.659Z"}
2025-02-10 23:34:03 UTC -- Dumping repositories ... done
2025-02-10 23:34:03 UTC -- Dumping uploads ...
2025-02-10 23:34:03 UTC -- Dumping uploads ... done
2025-02-10 23:34:03 UTC -- Dumping builds ...
2025-02-10 23:34:03 UTC -- Dumping builds ... done
2025-02-10 23:34:03 UTC -- Dumping artifacts ...
2025-02-10 23:34:04 UTC -- Dumping artifacts ... done
2025-02-10 23:34:04 UTC -- Dumping pages ...
2025-02-10 23:34:04 UTC -- Dumping pages ... done
2025-02-10 23:34:04 UTC -- Dumping lfs objects ...
2025-02-10 23:34:04 UTC -- Dumping lfs objects ... done
2025-02-10 23:34:04 UTC -- Dumping terraform states ...
2025-02-10 23:34:04 UTC -- Dumping terraform states ... done
2025-02-10 23:34:04 UTC -- Dumping container registry images ... [DISABLED]
2025-02-10 23:34:04 UTC -- Dumping packages ...
2025-02-10 23:34:04 UTC -- Dumping packages ... done
2025-02-10 23:34:04 UTC -- Dumping ci secure files ...
2025-02-10 23:34:04 UTC -- Dumping ci secure files ... done
2025-02-10 23:34:04 UTC -- Dumping external diffs ...
2025-02-10 23:34:04 UTC -- Dumping external diffs ... done
2025-02-10 23:34:04 UTC -- Creating backup archive: 1739230437_2025_02_10_17.8.1-ee_gitlab_backup.tar ...
2025-02-10 23:34:04 UTC -- Creating backup archive: 1739230437_2025_02_10_17.8.1-ee_gitlab_backup.tar ... done
2025-02-10 23:34:04 UTC -- Uploading backup archive to remote storage ... [SKIPPED]
2025-02-10 23:34:04 UTC -- Deleting old backups ... [SKIPPED]
2025-02-10 23:34:04 UTC -- Deleting tar staging files ...
2025-02-10 23:34:04 UTC -- Cleaning up /var/opt/gitlab/backups/backup_information.yml
2025-02-10 23:34:04 UTC -- Cleaning up /var/opt/gitlab/backups/db
```

Рисунок 35: Процесс резервного копирования сервера GitLab

На рисунке 36 ниже показан результат выполнения команды, выделено имя файла резервной копии:

```
1739230437_2025_02_10_17.8.1-ee
```

```
2025-02-10 23:34:04 UTC -- Cleaning up /var/opt/gitlab/backups/repositories
2025-02-10 23:34:04 UTC -- Cleaning up /var/opt/gitlab/backups/uploads.tar.gz
2025-02-10 23:34:04 UTC -- Cleaning up /var/opt/gitlab/backups/builds.tar.gz
2025-02-10 23:34:04 UTC -- Cleaning up /var/opt/gitlab/backups/artifacts.tar.gz
2025-02-10 23:34:04 UTC -- Cleaning up /var/opt/gitlab/backups/pages.tar.gz
2025-02-10 23:34:04 UTC -- Cleaning up /var/opt/gitlab/backups/lfs.tar.gz
2025-02-10 23:34:04 UTC -- Cleaning up /var/opt/gitlab/backups/terraform_state.tar.gz
2025-02-10 23:34:04 UTC -- Cleaning up /var/opt/gitlab/backups/packages.tar.gz
2025-02-10 23:34:04 UTC -- Cleaning up /var/opt/gitlab/backups/ci_secure_files.tar.gz
2025-02-10 23:34:04 UTC -- Cleaning up /var/opt/gitlab/backups/external_diffs.tar.gz
2025-02-10 23:34:04 UTC -- Deleting tar staging files ... done
2025-02-10 23:34:04 UTC -- Deleting backups/tmp ...
2025-02-10 23:34:04 UTC -- Deleting backups/tmp ... done
2025-02-10 23:34:04 UTC -- Warning: Your gitlab.rb and gitlab-secrets.json files contain sensitive data
and are not included in this backup. You will need these files to restore a backup.
Please back them up manually.
2025-02-10 23:34:04 UTC -- Backup 1739230437_2025_02_10_17.8.1-ee is done.
2025-02-10 23:34:04 UTC -- Deleting backup and restore PID file at [/opt/gitlab/embedded/service/gitlab-rails/tmp/backup_restore.pid] ... done
ubuntu@ip-172-31-86-106:~$
```

Рисунок 36: Результат выполнения резервного копирования GitLab с выделением имени созданного файла резервной копии

9. Уничтожение данных Gitlab и восстановление Gitlab из резервной копии

Чтобы смоделировать сценарий аварийного восстановления, данные GitLab были намеренно уничтожены, а сервер восстановлен из ранее созданной резервной копии.

Команда `sudo docker exec -it st16-gitlab rm -rf /var/opt/gitlab/*` была использована для удаления всех данных в каталоге `/var/opt/gitlab/`, эффективно уничтожив данные экземпляра GitLab, как показано на рисунке 33.

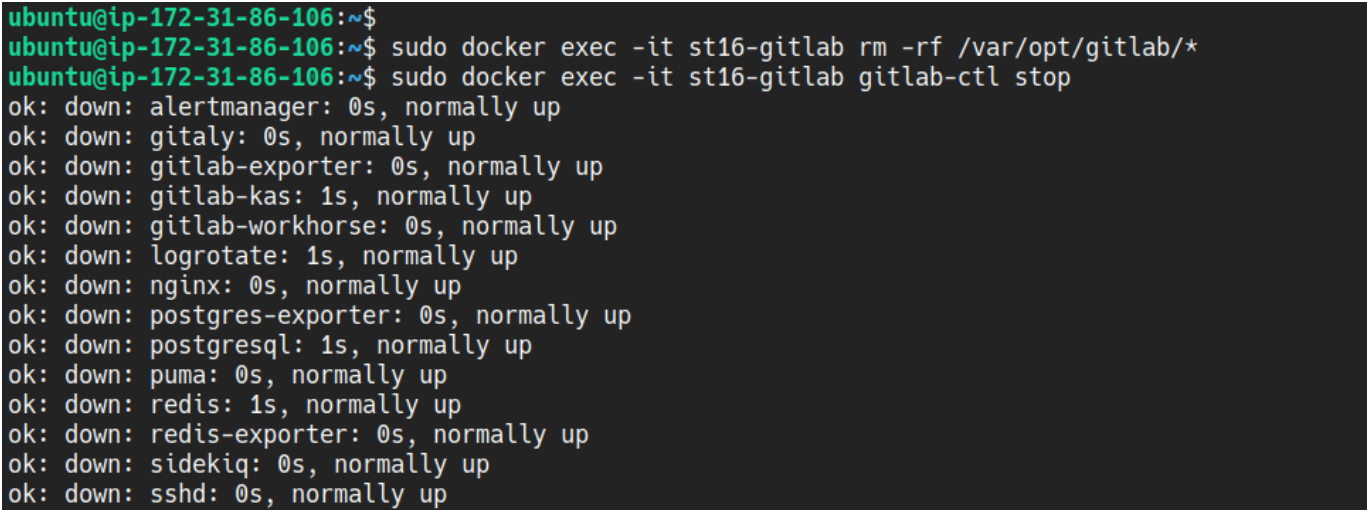


Рисунок 37: Уничтожение данных GitLab

10. Подтверждение наличия всех репозиториев и файлов Для выполнения этой задачи была выполнена команда `sudo docker exec -it st16-gitlab gitlab-backup restore BACKUP=1739230437_2025_02_10_17.8.1-ee` для восстановления сервера GitLab из указанной резервной копии. Смотрите рисунок 34.

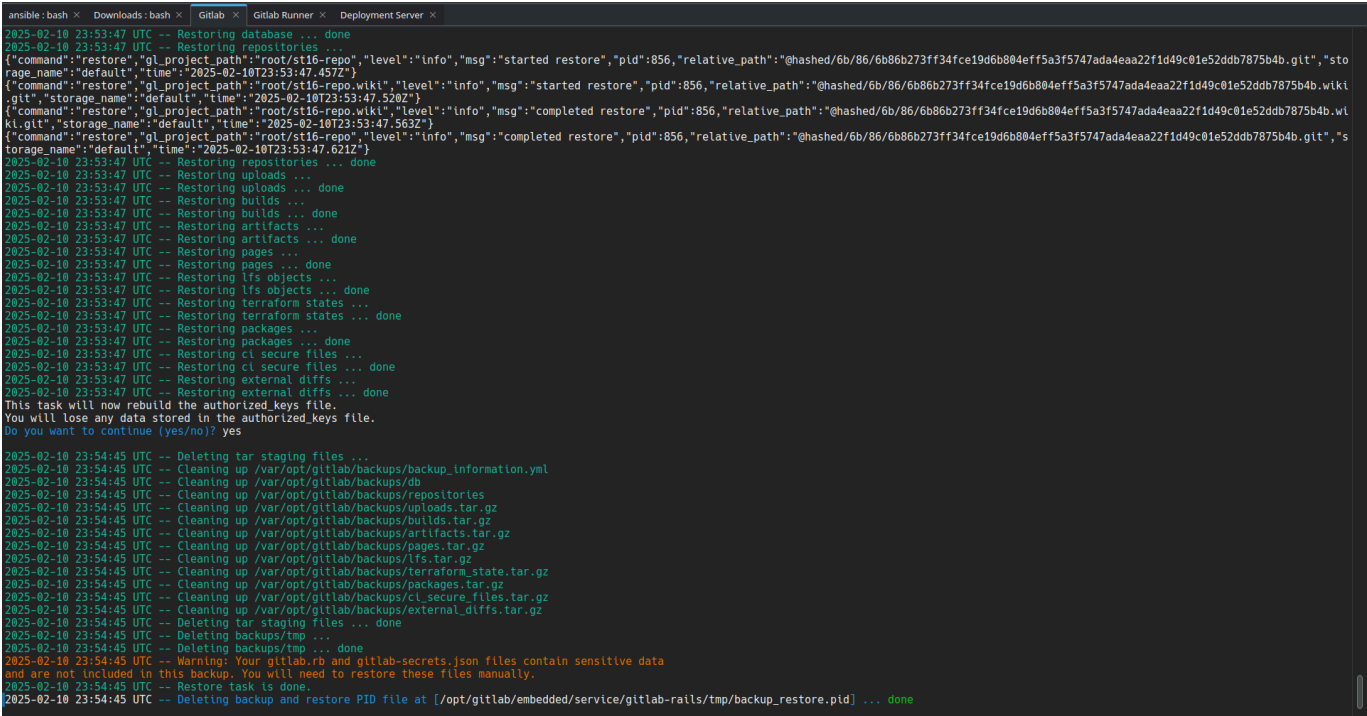


Рисунок 37: Восстановление сервера GitLab из указанного файла резервной копии

В пользовательский интерфейс GitLab зашли, чтобы убедиться, что все репозитории и файлы, включая `st16-repo`, были успешно восстановлены. Было подтверждено, что все репозитории и файлы присутствуют и не повреждены, как показано на рисунке 38. Это подтвердило эффективность процесса резервного копирования и восстановления.

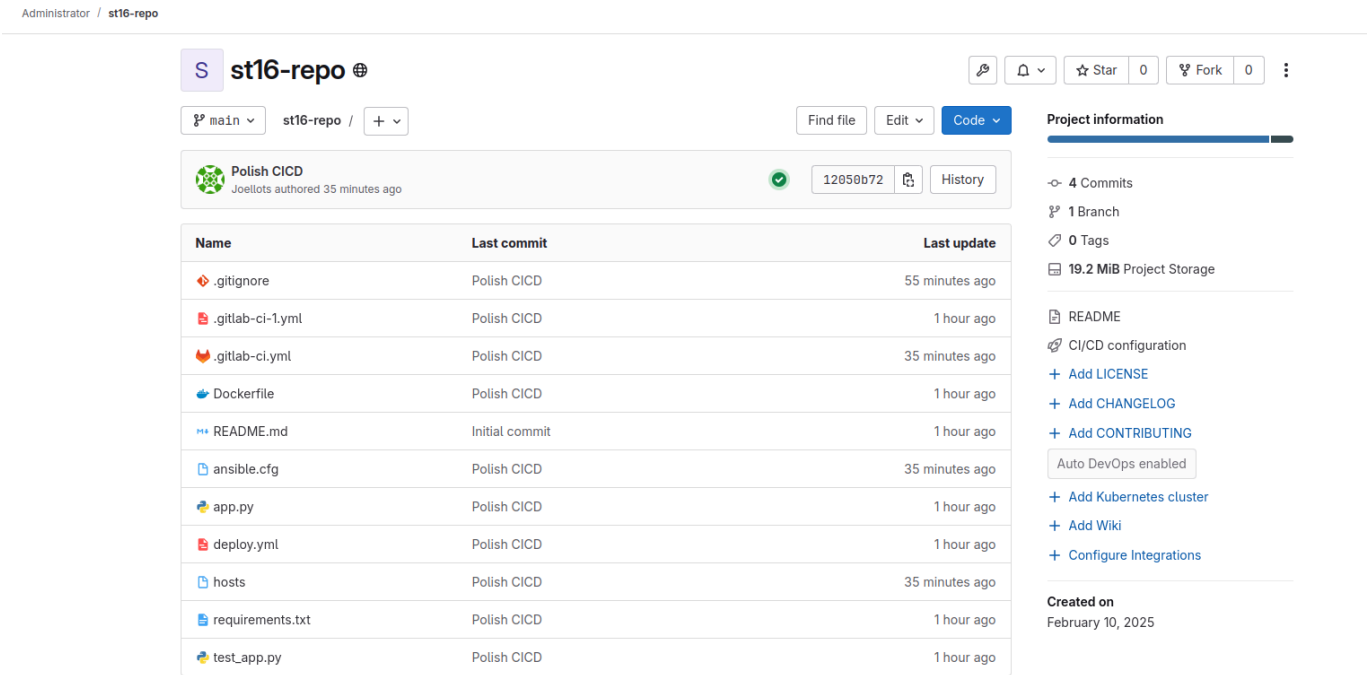


Рисунок 38: Пользовательский интерфейс GitLab, показывающий восстановленный репозиторий `st16-repo` и его файлы.

ССЫЛКИ

1. [Конфигурация CI/CD-конвейера GitLab](#)
2. [Резервное копирование и восстановление GitLab](#)
3. [Terraform AWS Provider](#)
4. [Модули Docker в Ansible](#)