Secure System Development - Lab 1

The repo link to check the files used in this assignment: full report

Task 1 - GitLab Server

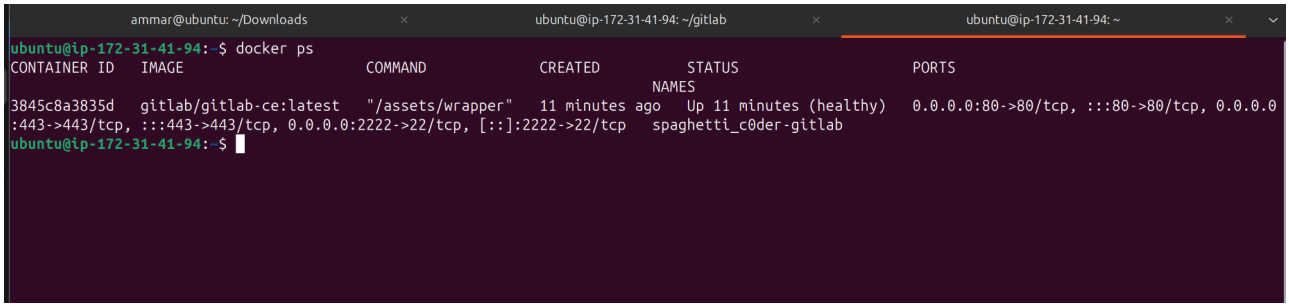
- Let's first initialize an instance on AWS and connect to it using our local machine via ssh, and then install docker and docker-compose there

```
Expanded Security Maintenance for Applications is not enabled.

63 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

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


Last login: Mon Feb 10 16:46:00 2025 from 45.67.35.120
ubuntu@ip-172-31-41-94:~$ docker --version
Docker version 27.5.1, build 9f9e405
ubuntu@ip-172-31-41-94:~$ docker-compose --version
docker-compose version 1.29.2, build unknown
ubuntu@ip-172-31-41-94:~$
```

- then let's create the docker-compose file according to the requirements for the gitlab server, and build it up taking into consideration enabling https access, disabling unneeded services (registry , mattermost , gitlab-pages , gitlab-kas) and setting up volumes to keep a backup of the data (configs, logs, app data)

```yaml
version: '3.8'

services:
  gitlab-server:
    image: 'gitlab/gitlab-ce:latest'
    container_name: spaghetti_c0der-gitlab
    restart: unless-stopped
    hostname: gitlab.test.local
    environment:
      GITLAB_OMNIBUS_CONFIG: |
        gitlab_rails['initial_root_password'] = '0123456789abC!@'
        puma['worker_processes'] = 0
        external_url 'https://gitlab.test.local'
        gitlab_rails['gitlab_shell_ssh_port'] = 2222
        gitlab_rails['registry_enabled'] = false
        mattermost['enable'] = false
        gitlab_pages['enable'] = false
        gitlab_kas['enable'] = false
        nginx['redirect_http_to_https'] = true
        nginx['ssl_certificate'] = "/etc/gitlab/ssl/crt"
        nginx['ssl_certificate_key'] = "/etc/gitlab/ssl/gitlab.test.local.key"
    ports:
      - '80:80'
      - '2222:22'
      - '443:443'
    volumes:
      - './gitlab/config:/etc/gitlab'
      - './gitlab/logs:/var/log/gitlab'
      - './gitlab/data:/var/opt/gitlab'
      - '/etc/gitlab/ssl:/etc/gitlab/ssl'
    shm_size: '256m'
```

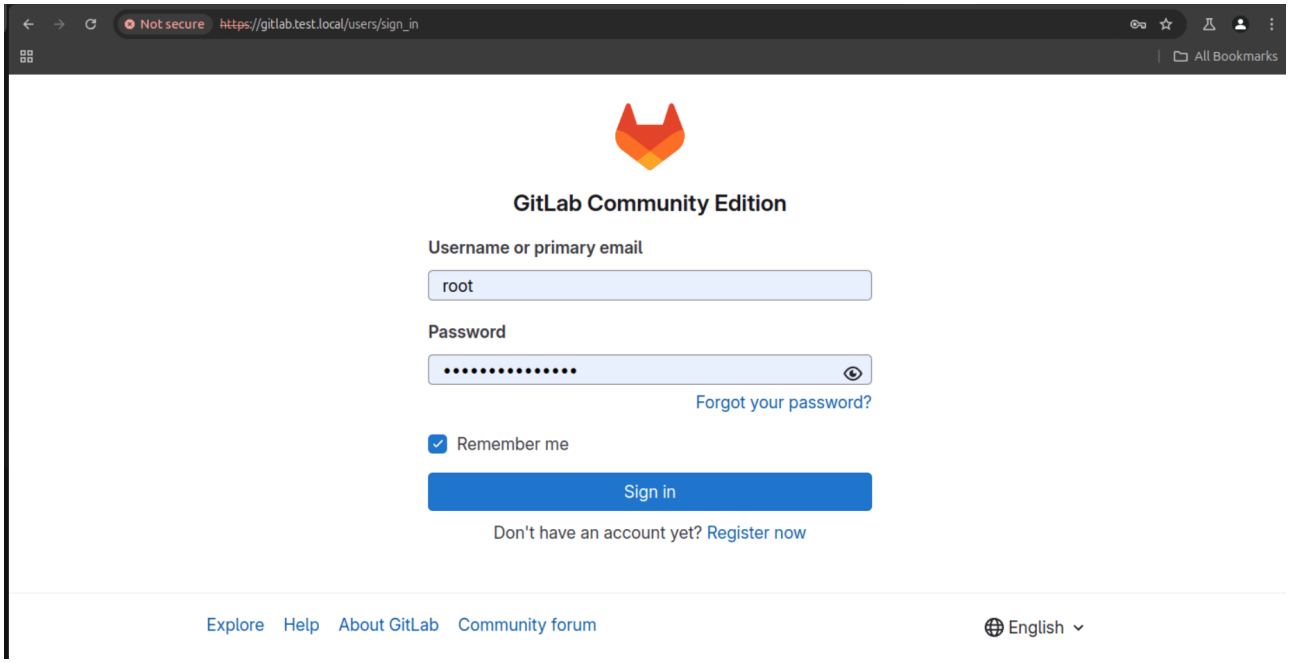
- after building the docker-compose file we can see



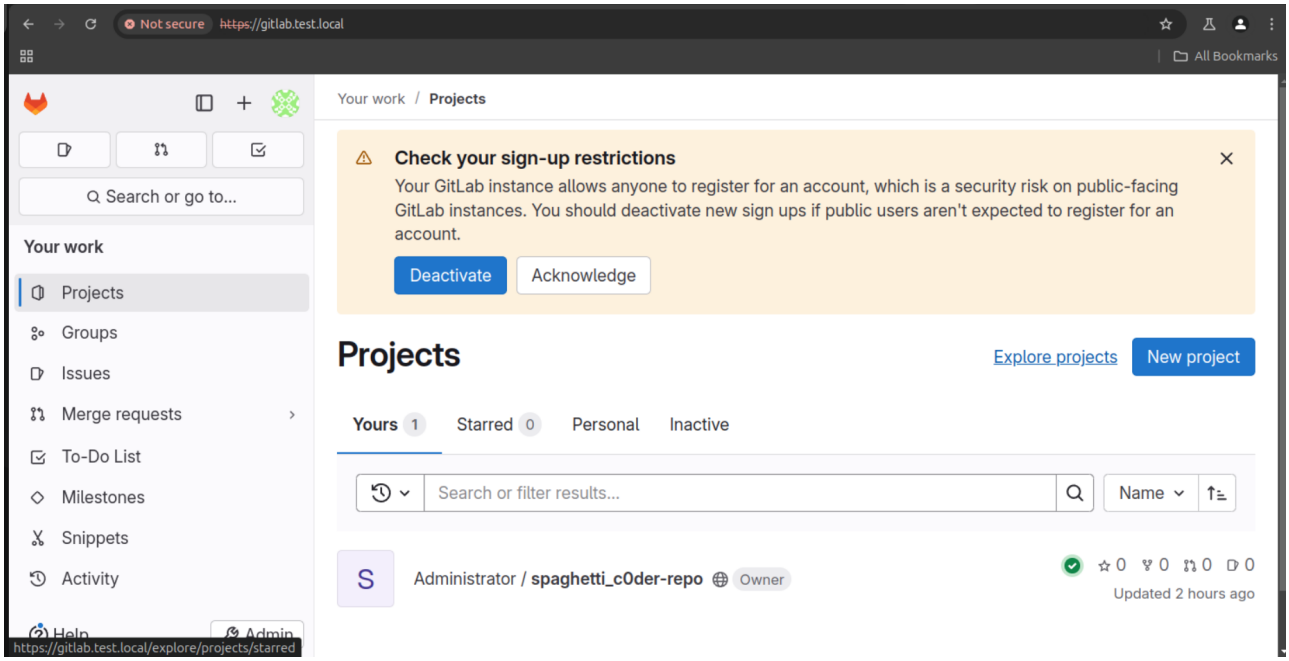
- let's now setup the certificate using `mkcert`

```
ubuntu@ip-172-31-41-94:~$ chmod +x mkcert-v*-linux-amd64
ubuntu@ip-172-31-41-94:~$ sudo cp mkcert-v*-linux-amd64 /usr/local/bin/mkcert
ubuntu@ip-172-31-41-94:~$ mkcert -install
Created a new local CA 💥
The local CA is now installed in the system trust store! ⚡

ubuntu@ip-172-31-41-94:~$ cd /etc
ubuntu@ip-172-31-41-94:/etc$ cd gitlab/
ubuntu@ip-172-31-41-94:/etc/gitlab$ ls
ssl
ubuntu@ip-172-31-41-94:/etc/gitlab$ cd ssl
ubuntu@ip-172-31-41-94:/etc/gitlab/ssl$ ls
ubuntu@ip-172-31-41-94:/etc/gitlab/ssl$ ls -al
total 8
drwxr-xr-x 2 root root 4096 Feb 10 17:34 .
drwxr-xr-x 3 root root 4096 Feb 10 17:34 ..
ubuntu@ip-172-31-41-94:/etc/gitlab/ssl$ mkcert -key-file gitlab.local.test.key -cert-file cert gitlab.local.test
ERROR: failed to save certificate: open cert: permission denied
ubuntu@ip-172-31-41-94:/etc/gitlab/ssl$ sudo mkcert -key-file gitlab.local.test.key -cert-file cert gitlab.local.test
Created a new local CA 💥
Note: the local CA is not installed in the system trust store.
Run "mkcert -install" for certificates to be trusted automatically ⚠️

Created a new certificate valid for the following names 📜
 - "gitlab.local.test"

The certificate is at "cert" and the key at "gitlab.local.test.key" ✅

It will expire on 10 May 2027 🗓
```

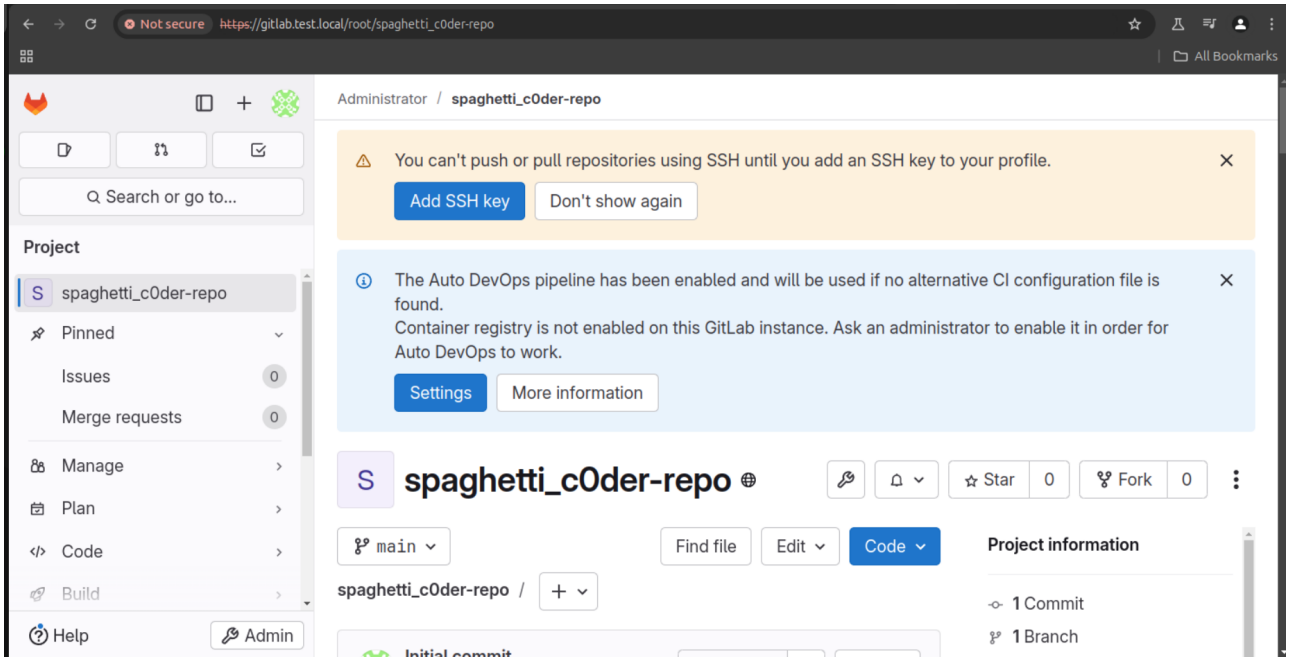
- then let's head to the browser and check the server:

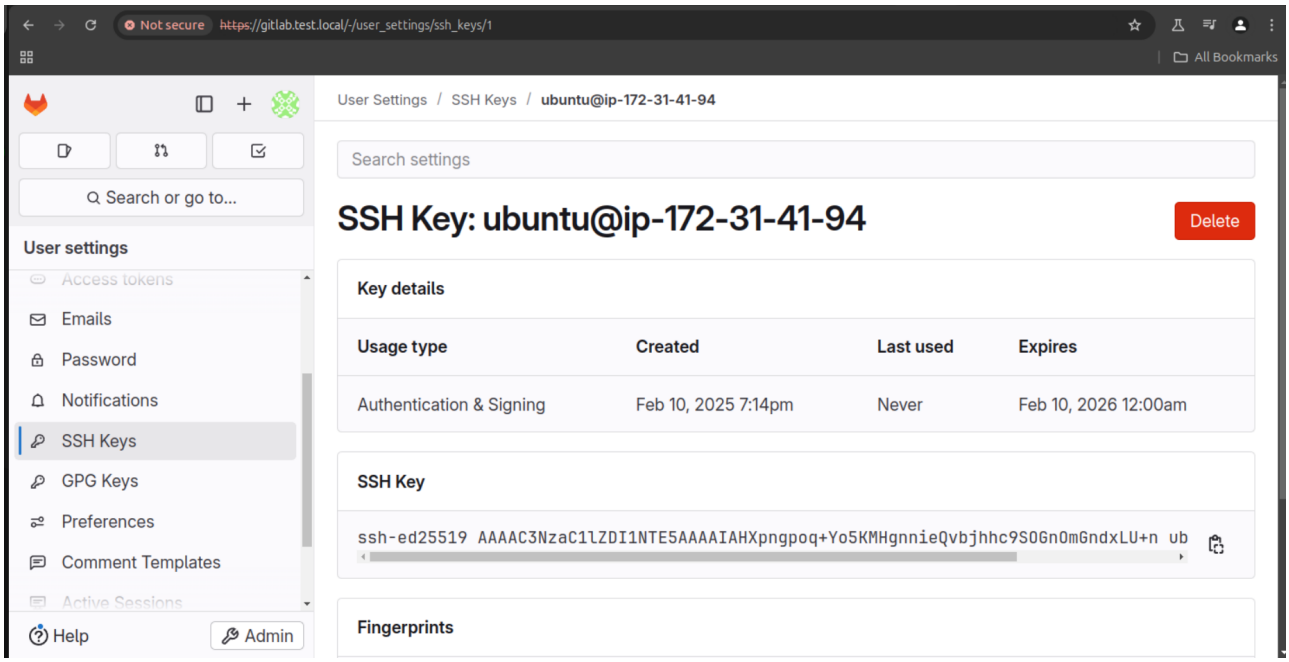


- let's log in:



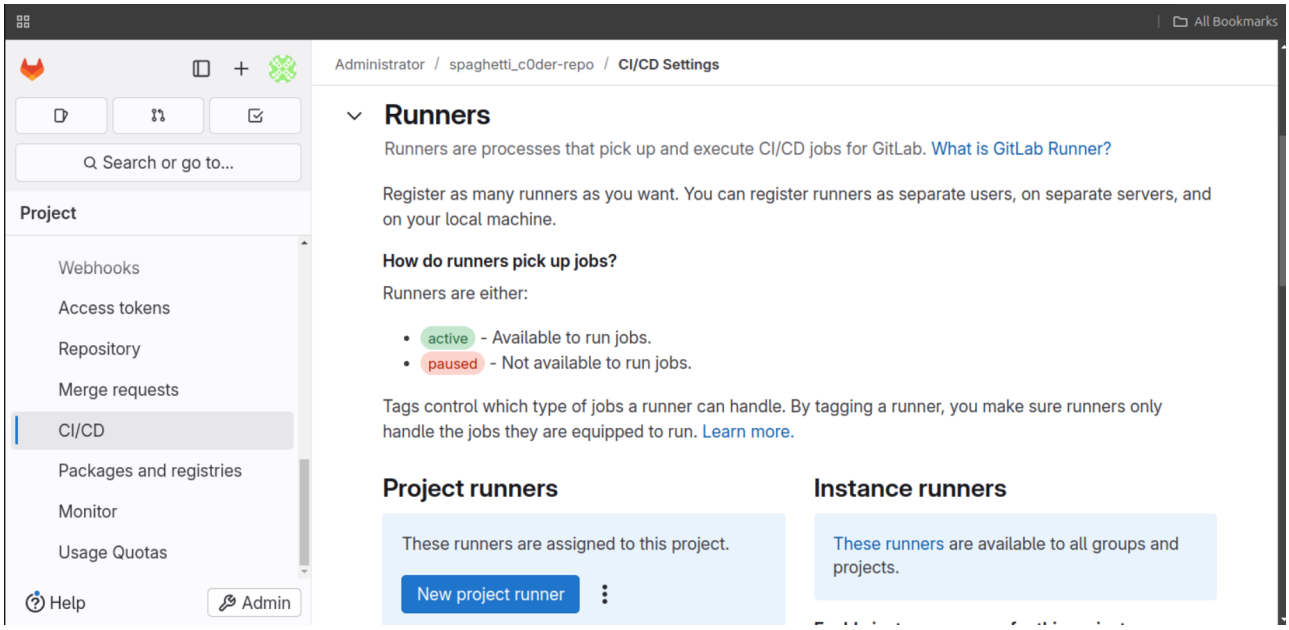
- then let's generate an ssh key to the server and copy it to the gitlab platform

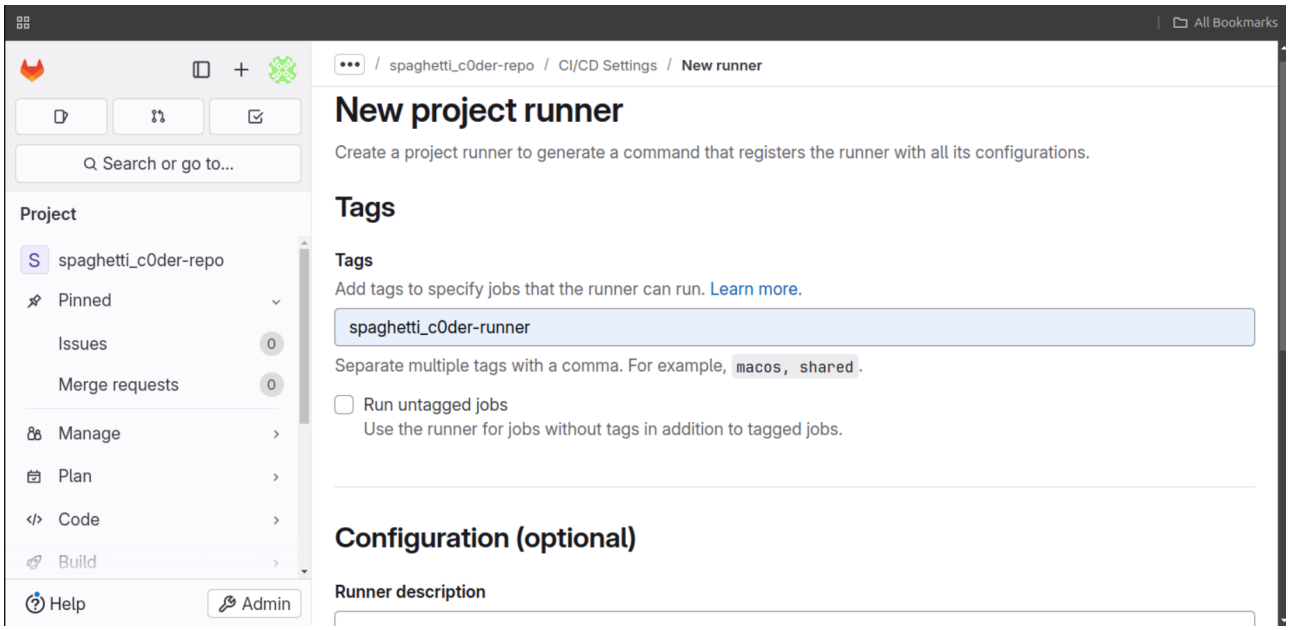


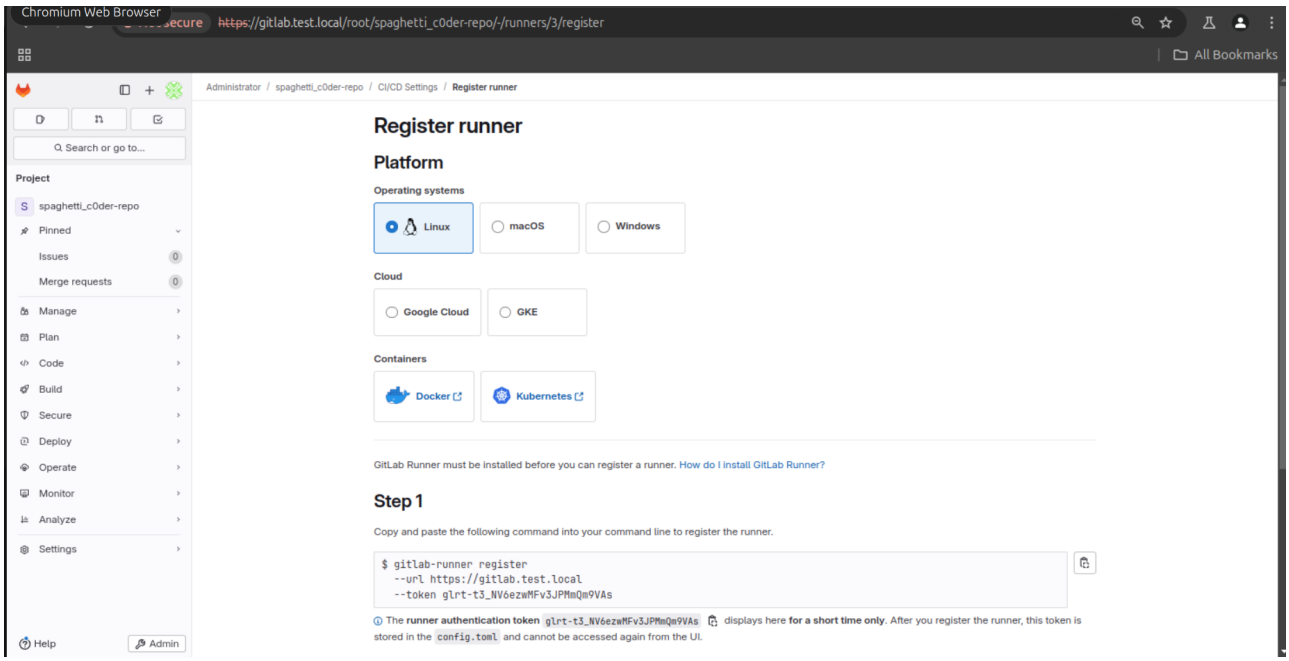
- 

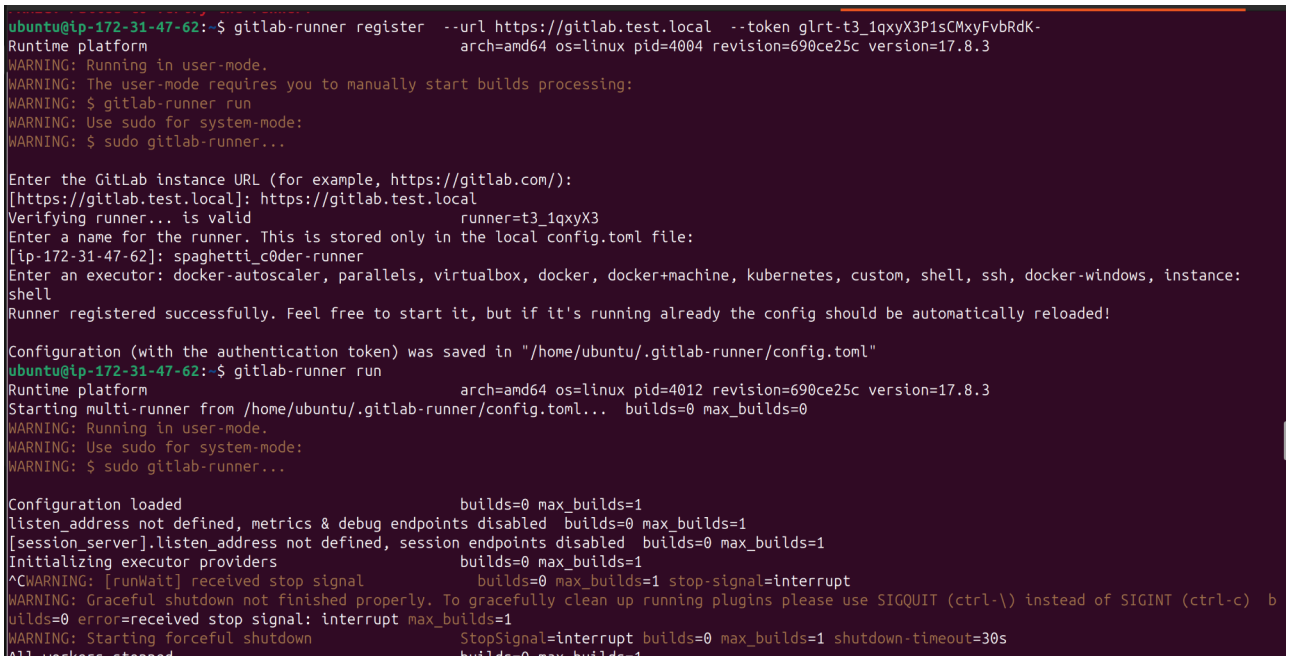
Task 2 - GitLab Runner

- let's install gitlab-runner on the 2nd vm instance
- 

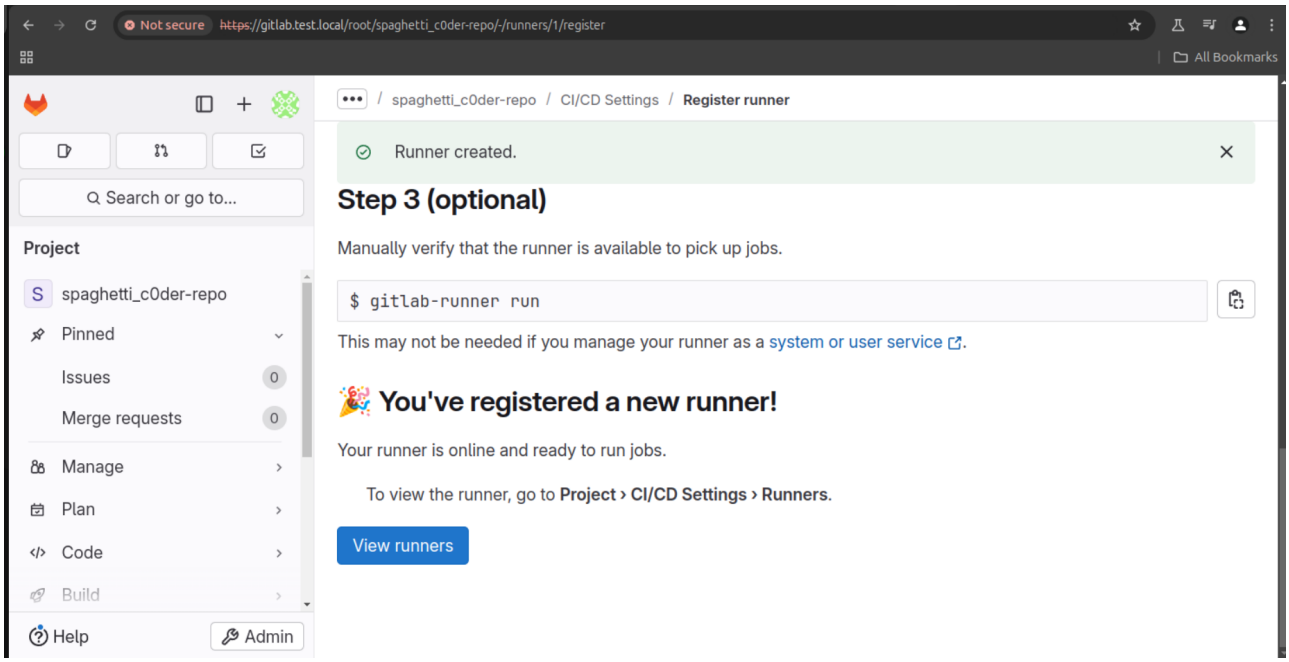
- then let's setup the certificate on the runner side by copying the certificate that we have created on the server side before, and then let's add the runner to our previous project
- 

- 

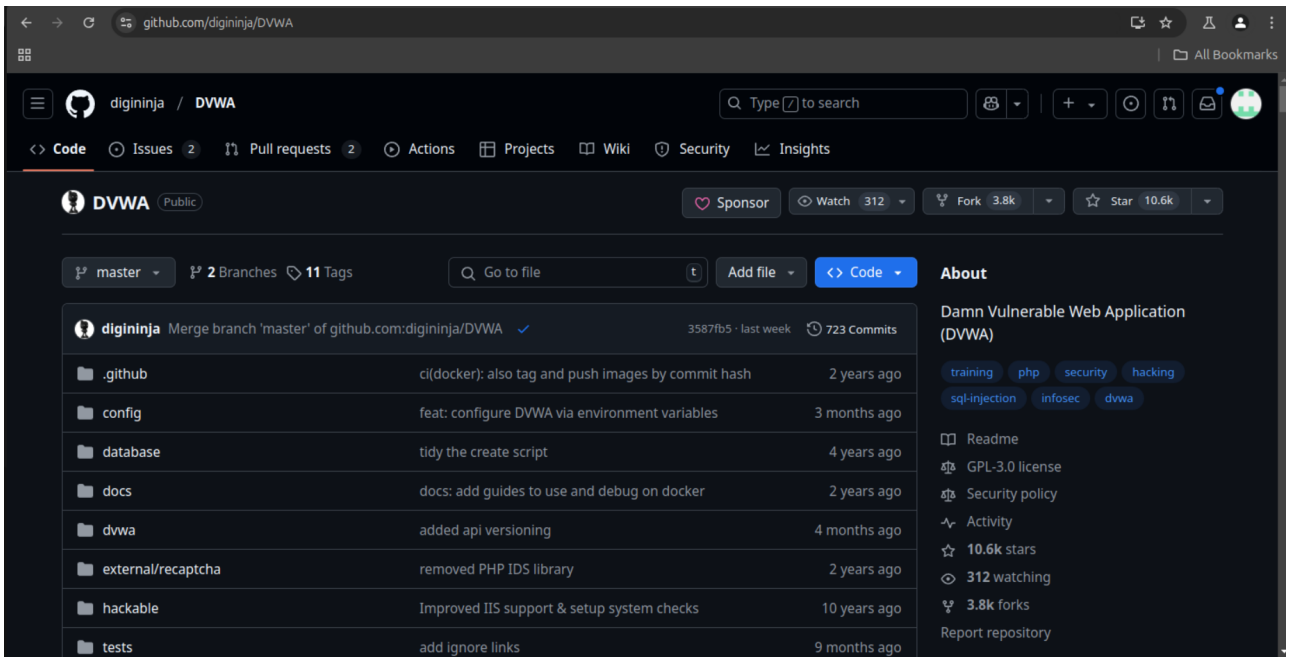
- 

- 

- the runner has been added successfully

- 

Task 3 - GitLab SAST

- selecting the following repo as a target
- 

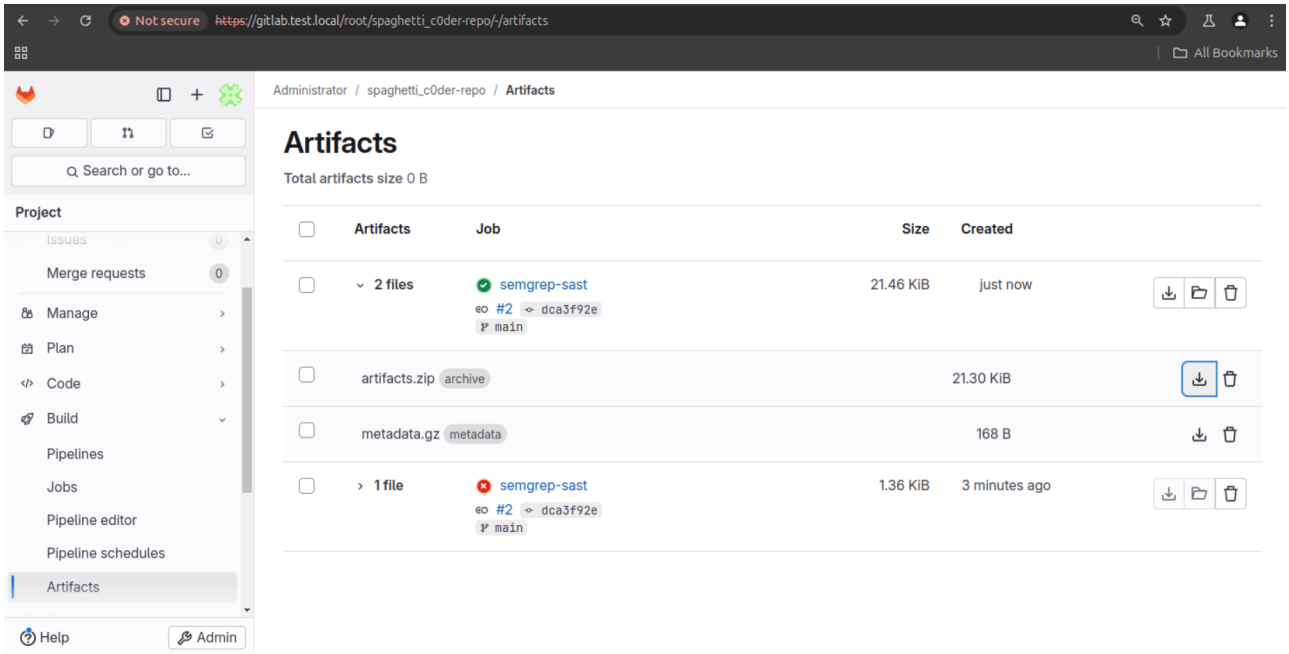
- let's clone the project, delete `.git` directory and then set the remote origin to our gitlab repo and push the project there

```
 create mode 100644 vulnerabilities/xss_r/source/high.php
 create mode 100644 vulnerabilities/xss_r/source/impossible.php
 create mode 100644 vulnerabilities/xss_r/source/low.php
 create mode 100644 vulnerabilities/xss_r/source/medium.php
 create mode 100644 vulnerabilities/xss_s/help/help.php
 create mode 100644 vulnerabilities/xss_s/index.php
 create mode 100644 vulnerabilities/xss_s/source/high.php
 create mode 100644 vulnerabilities/xss_s/source/impossible.php
 create mode 100644 vulnerabilities/xss_s/source/low.php
 create mode 100644 vulnerabilities/xss_s/source/medium.php
ubuntu@ip-172-31-41-94:~/dvwa/spaghetti_c0der-repo$ git push -uf origin main
Enumerating objects: 317, done.
Counting objects: 100% (317/317), done.
Delta compression using up to 2 threads
Compressing objects: 100% (286/286), done.
Writing objects: 100% (315/315), 905.85 KiB | 10.91 MiB/s, done.
Total 315 (delta 35), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (35/35), done.
To ssh://gitlab.test.local:2222/root/spaghetti_c0der-repo.git
   44a08bc..7b26c44  main -> main
branch 'main' set up to track 'origin/main'.
```

- now let's install `semgrep` on the server and create our ci pipeline and push it to our repo

```
 GNU nano 7.2                                    .gitlab-ci.yml
stages:
  - test

semgrep-sast:
  stage: test
  script:
    - semgrep --config=auto --json > semgrep-report.json
  artifacts:
    paths:
      - semgrep-report.json
  rules:
    - if: $CI_COMMIT_REF_NAME == "main"
```

- the pipeline job has finished successfully, so let's check the artifacts



- by analyzing the json file, one of the dangerous vulnerabilits is `Command Injection (PHP)` with a **high** impact. this is because of using `shell_exec('ping ' . $target)` which allows for command injection if `$target` is user-controlled and not properly sanitized. this could allow an attacker to execute arbitrary commands on the server. and to avoid this we may:
  - avoid using functions like `shell_exec`, `exec`, or `system` with user-controlled input
  - sanitize and validate all inputs rigorously

- use safer alternatives like parameterized commands or libraries that handle shell execution securely