****

Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

Лабораторна робота №1

**Тенденції розвитку інформаційних систем та технологій**

*CI/CD. Хмарні обчислення.*

|  |  |  |
| --- | --- | --- |
| Виконав  студент групи ІТ-41ф |  | Перевірив: |
|  |  |  |
| Новиков Д. М. |  |  |
|  |  |  |
|  |  | ас. Цимбал С. І. |

Київ 2024

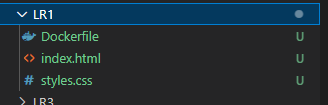
*Мета роботи:* створити CI/CD, який буде доставляти в хмару оновлені зміни до застосунку при коміті коду в GitHub репозиторій.

*Хід роботи:*

1. Обрати хмарну платформу і створити аккаунт на ній (наприклад, AWS, GCP).

Обрано AWS як хмарну платформу, і аккаунт вже існує.

2. Створити простий веб-застосунок html + css, створити для нього Dockerfile на базі nginx (можна використати матеріали з минулої лабораторної роботи):



index.html:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>My Web App</title>

    <link rel="stylesheet" href="styles.css">

</head>

<body>

    <h1>I'm not a fan of front-end development, so I apologize for the simplicity of the app :(</h1>

    <p>This is a simple HTML and CSS application deployed using Docker on AWS.</p>

</body>

</html>

styles.css:

body {

    font-family: Arial, sans-serif;

    text-align: center;

    background-color: #f5f5f5;

    margin: 0;

    padding: 20px;

}

h1 {

    color: #2c3e50;

}

p {

    color: #34495e;

}

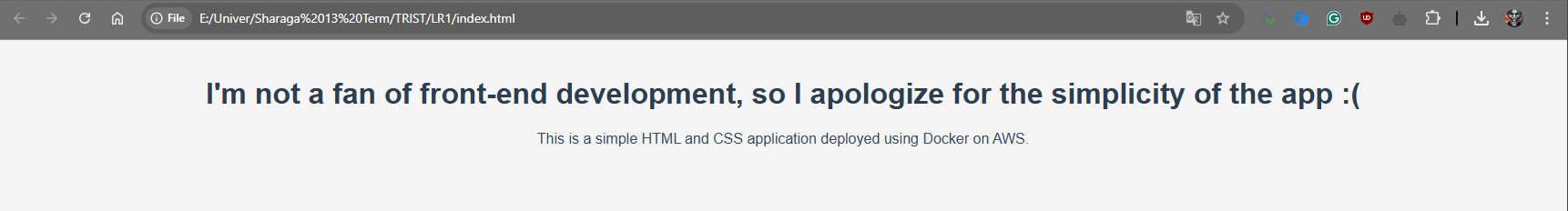
Dockerfile:

FROM nginx:alpine

COPY . /usr/share/nginx/html

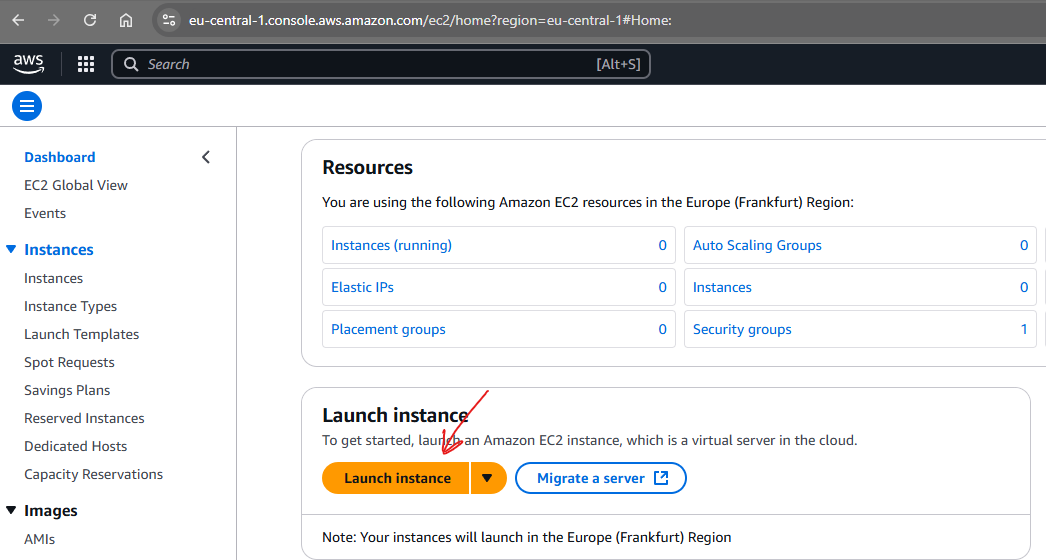
EXPOSE 80

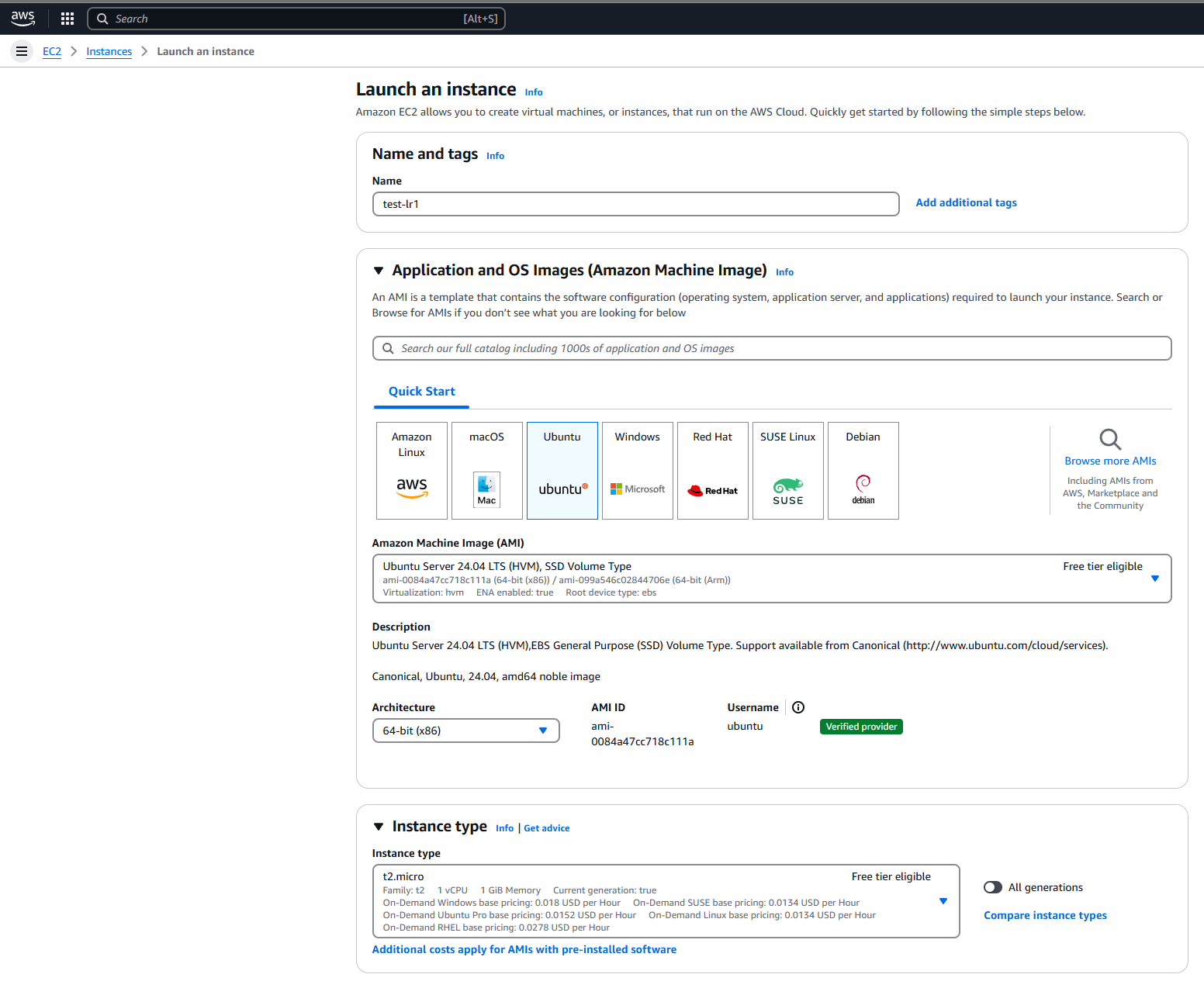
Застосунок має наступний вигляд:



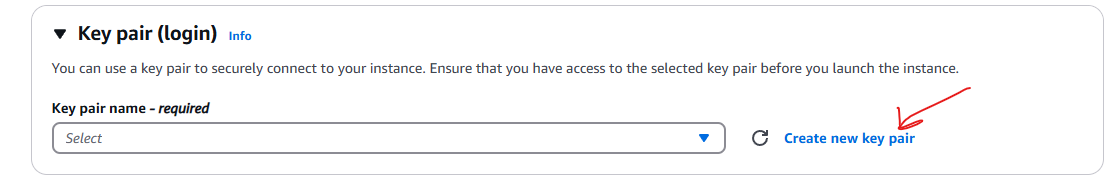
3. Створити VM instance та security group (застосунок має бути доступний на 80 порті, HTTP), встановити на нього docker.

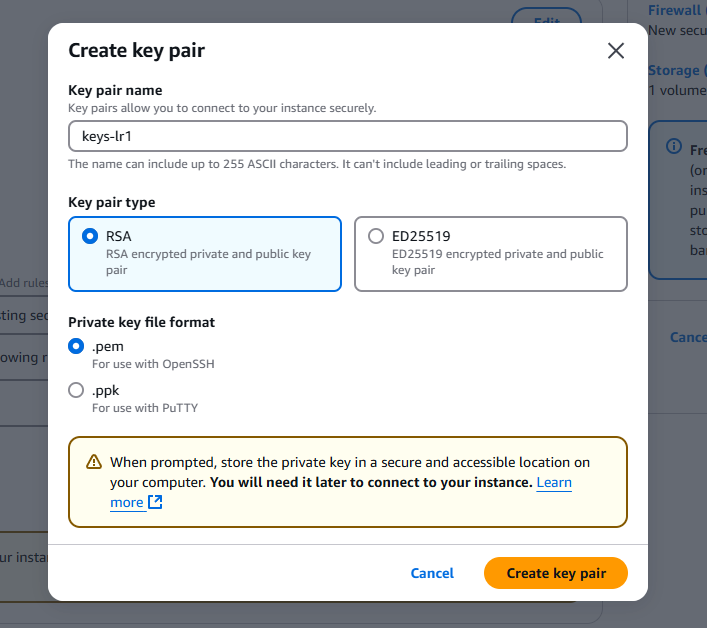
1. Cтворимо інстанс EC2:



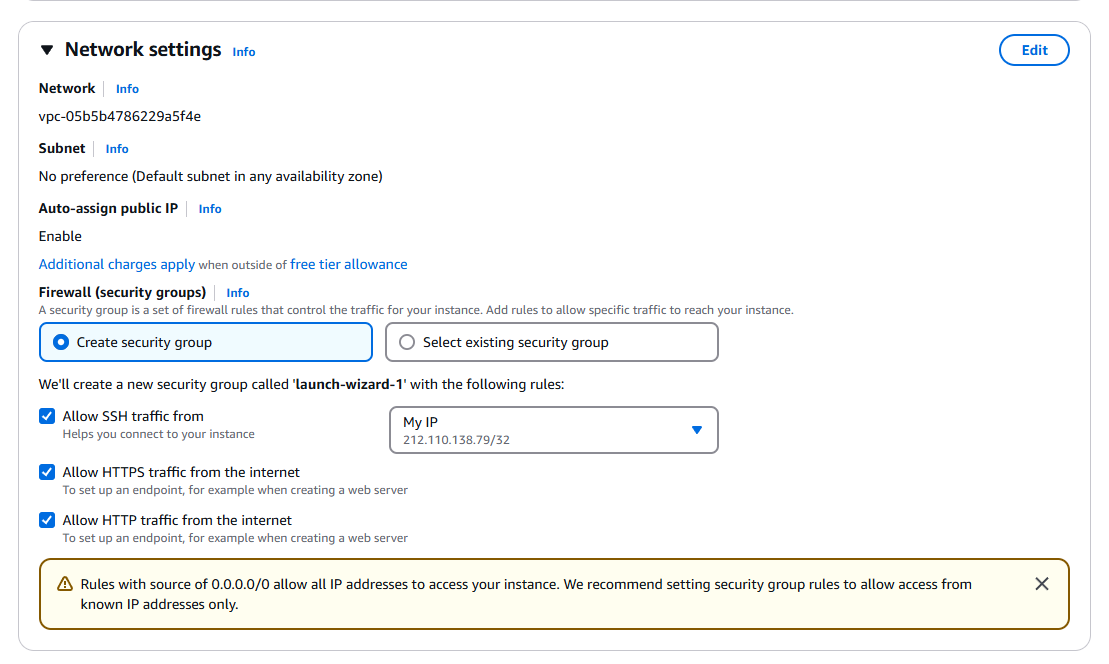


1. Згенеруємо пару ключів для підключення, завантажимо приватний ключ на локальний пристрій:

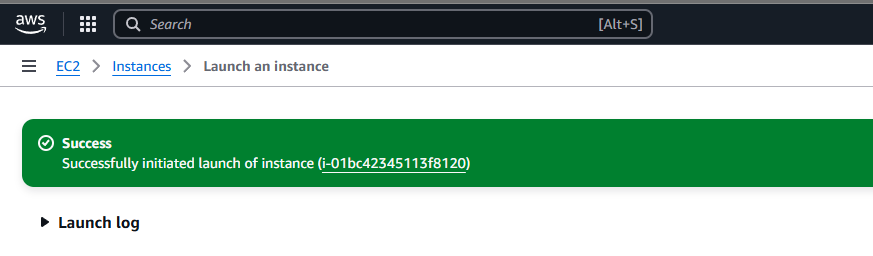




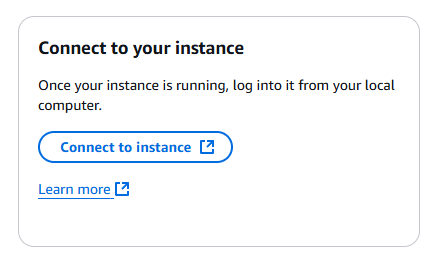
1. Створимо групу безпеки з наступними налаштуваннями:
   1. Дозволимо HTTP-трафік, щоб забезпечити доступ до веб-застосунку;
   2. Дозволимо SSH-підключення для управління інстансом.

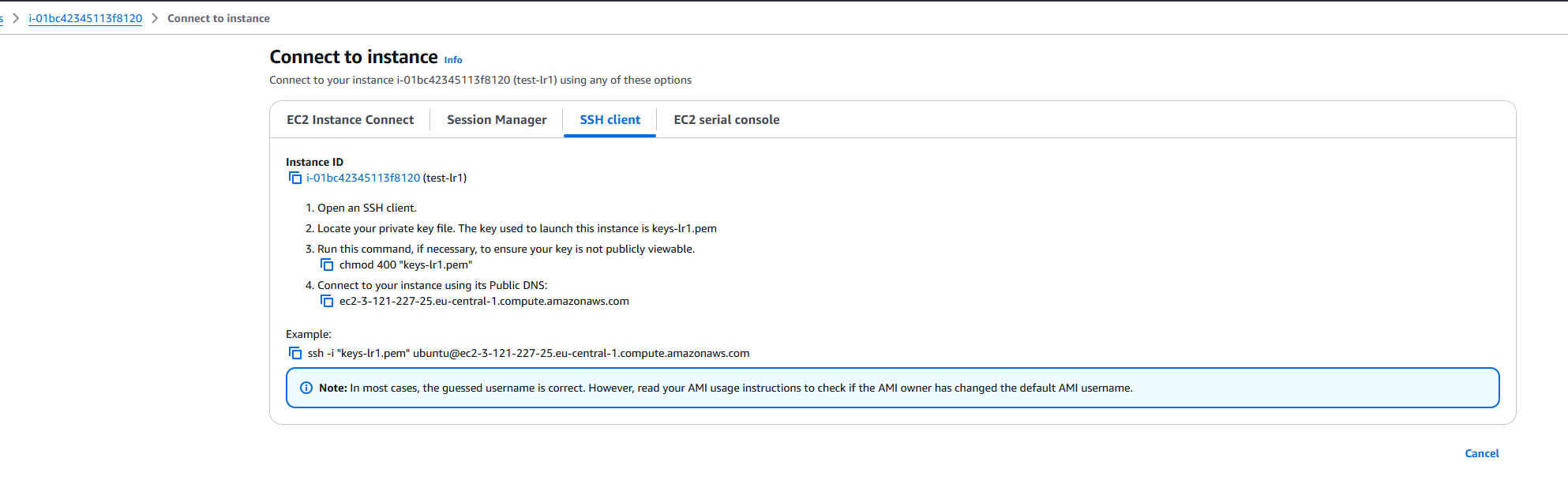


1. Інші налаштування залишаємо за замовчуванням і натискаємо Launch Instance:

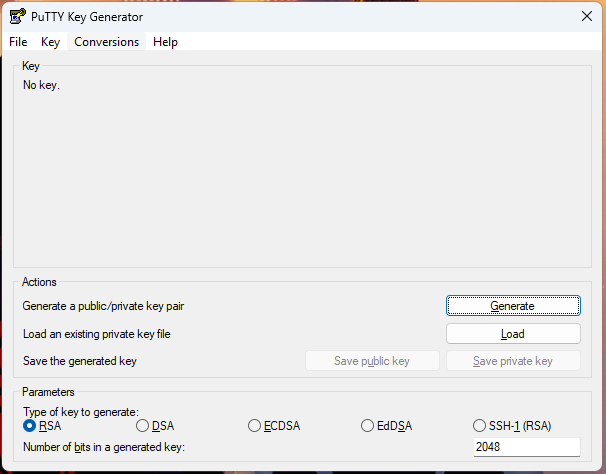


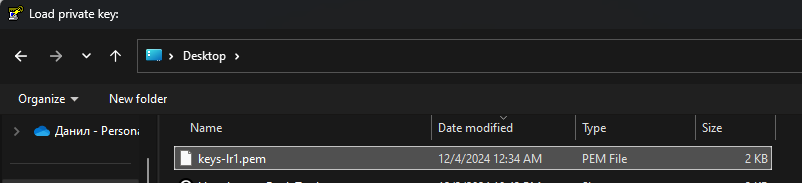
1. Після створення перевіримо інформацію для підключення:

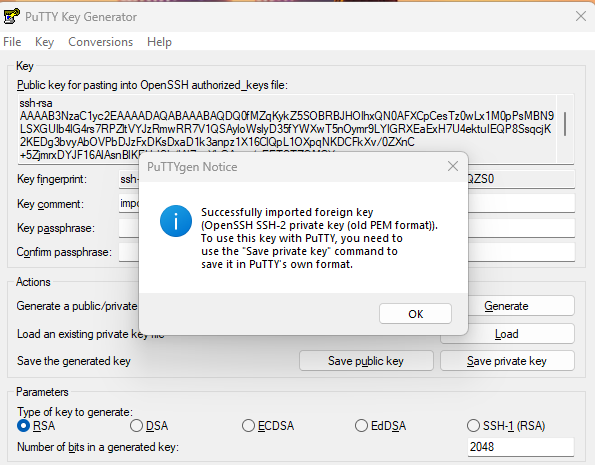


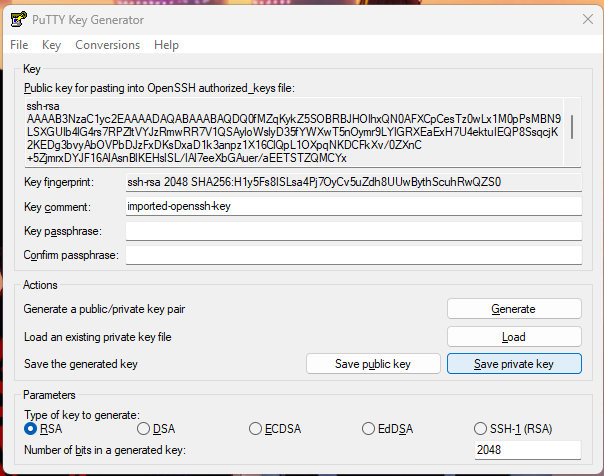


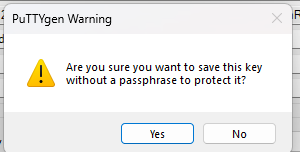
1. Підключимося через SSH до створеного EC2-інстансу, використовуючи PuTTY. Оскільки PuTTY вимагає ключ у форматі .ppk, для його генерації скористаємося утилітою PuTTY Key Generator:
   1. Відкриваємо PuTTY Key Generator;
   2. Завантажуємо приватний ключ у форматі .pem;
   3. Генеруємо та зберігаємо ключ у форматі .ppk.



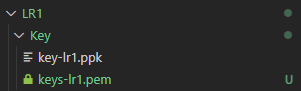




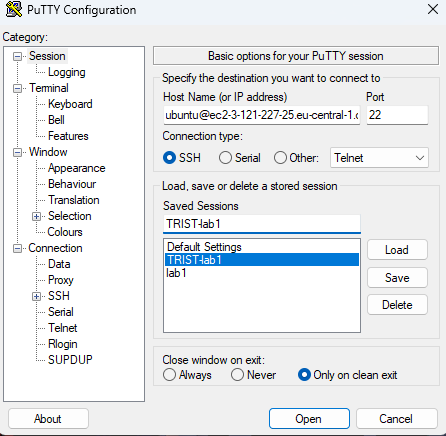


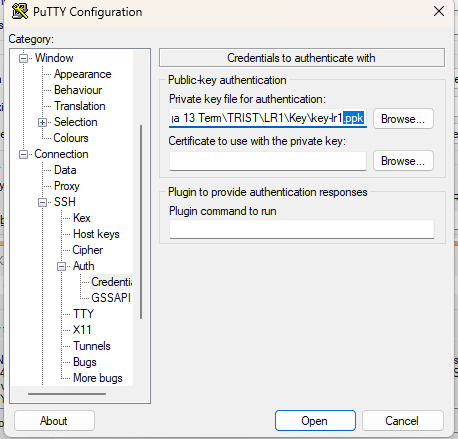


1. У результаті маємо пару ключів:
   1. .pem для використання у CLI або інших інструментах;
   2. .ppk для використання у PuTTY.

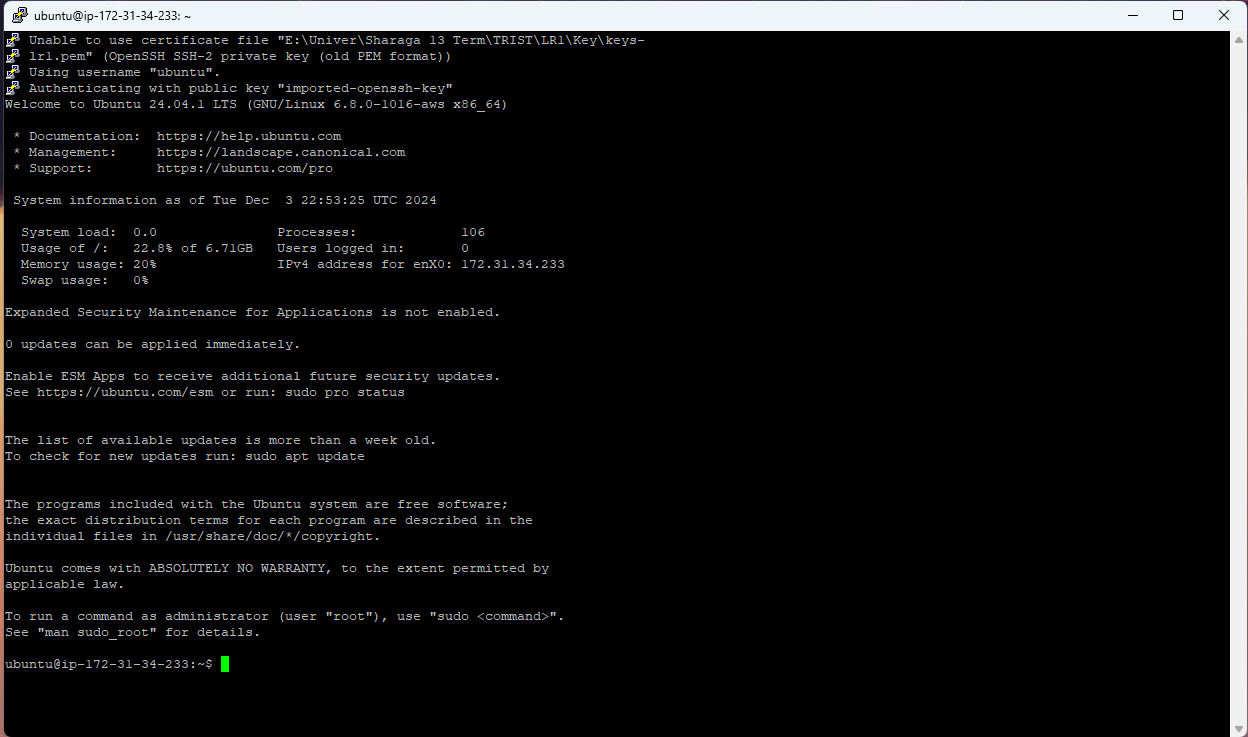


1. Налаштуємо підключення в PuTTY:
   1. У полі Host Name (or IP address) вкажемо публічну IP-адресу інстансу, отриману з розділу "Інформація про підключення";
   2. У розділі SSH > Auth додамо згенерований .ppk ключ:





1. Виконаємо підключення до EC2-інстансу через SSH:



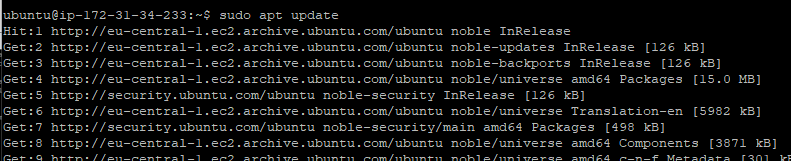
1. Встановимо Docker на EC2-інстанс через SSH для запуску контейнерів із додатком і Watchtower:

sudo apt update

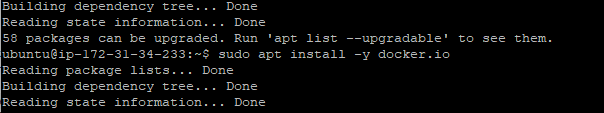
sudo apt install -y docker.io

sudo usermod -aG docker $USER

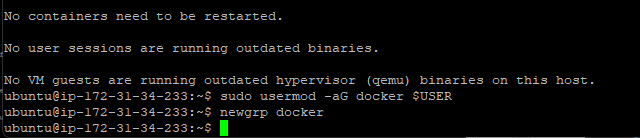
newgrp docker



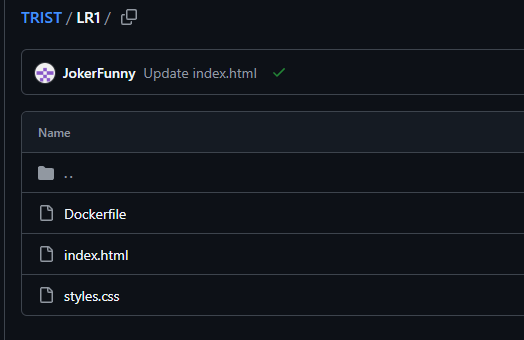
…



…

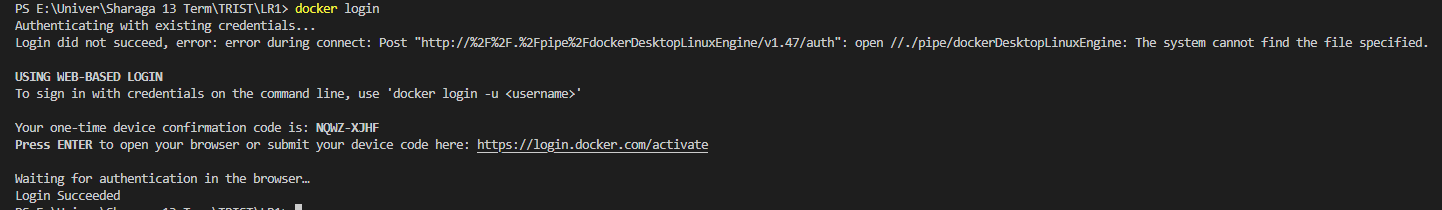


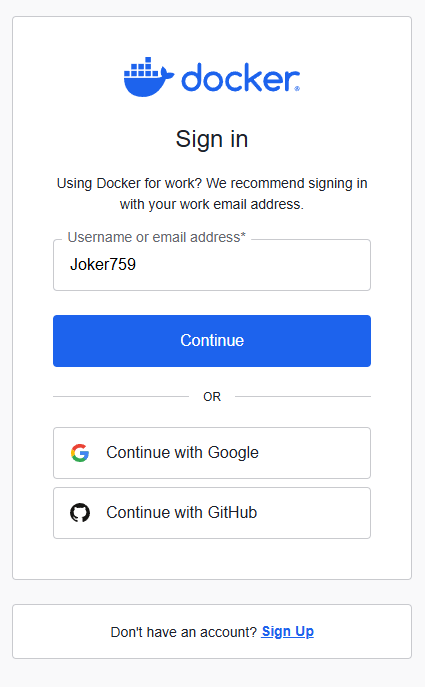
4. Створити репозиторій з кодом (Dockerfile, код вашого застосунка):

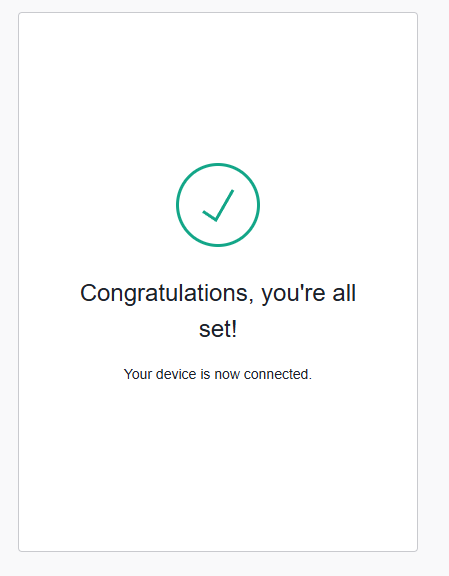


5. Запустити два контейнери – створений образ та watchtower:

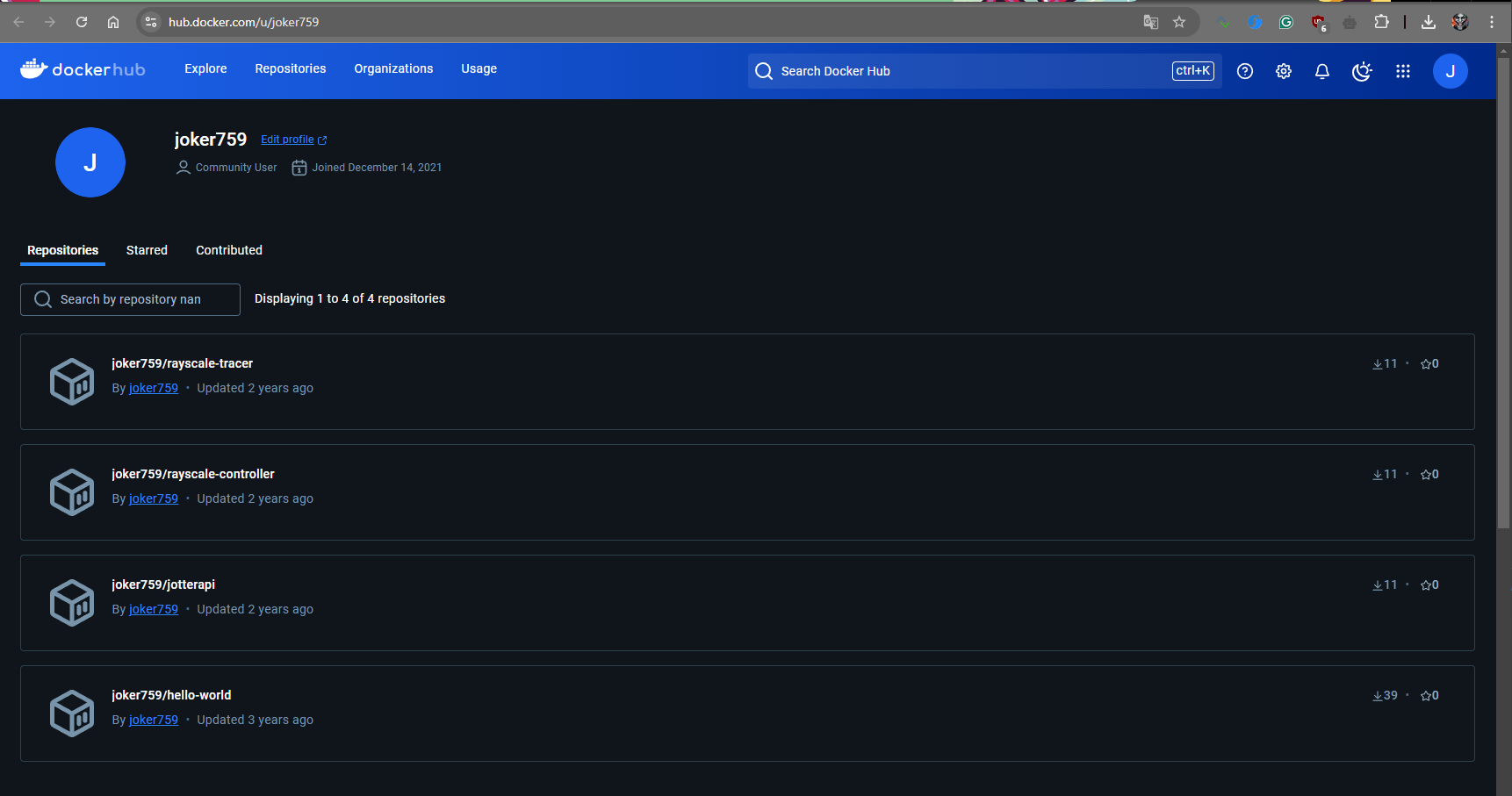
1. Для запуску контейнерів спершу необхідно створити образ у публічному Docker-реєстрі (оскільки використання приватного не вимагається). Використаємо DockerHub. Увійдемо в обліковий запис за допомогою команди «docker login»:



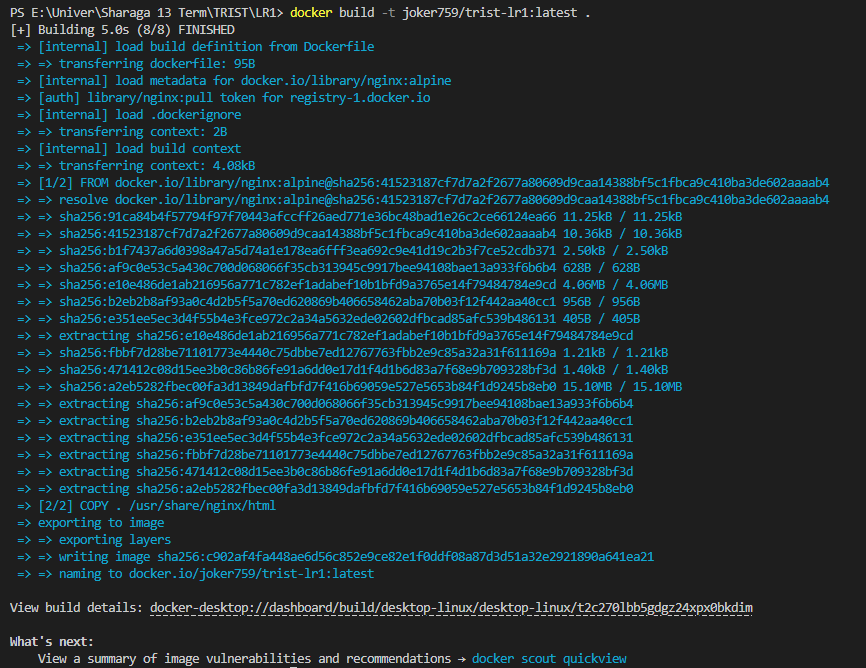




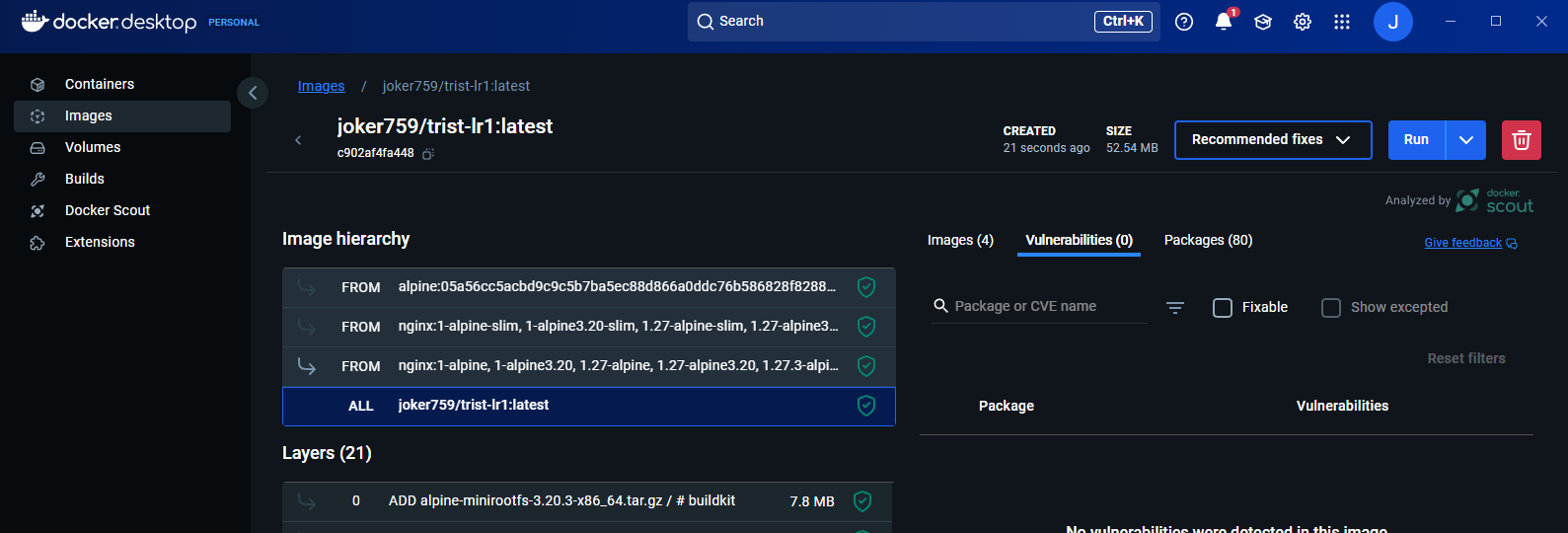
1. Перейдемо на DockerHub (https://hub.docker.com/u/joker759) та перевіримо існуючі репозиторії:



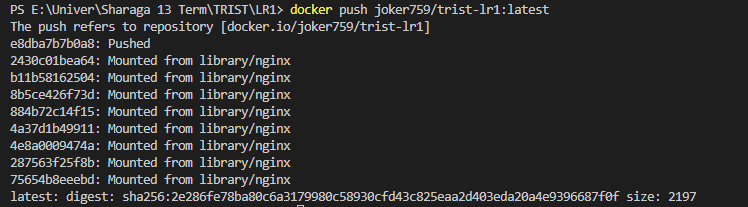
1. Збілдимо Dockerfile для створення Docker-образу, використовуючи команду «docker build -t joker759/trist-lr1:latest .»:



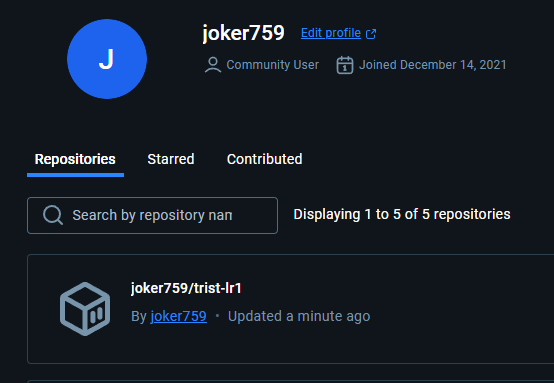
1. Перевіримо через Docker Desktop або команду docker images, що образ було створено успішно:

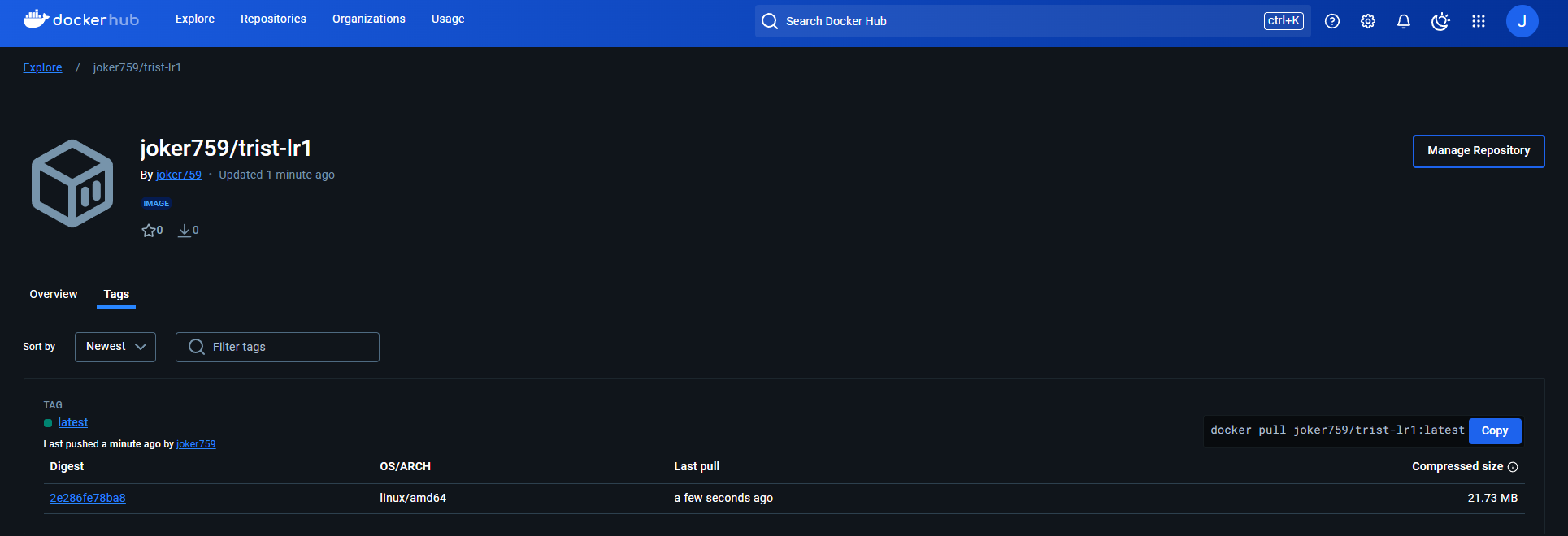


1. Завантажимо створений образ у DockerHub, щоб використати його для розгортання на EC2 за допомогою команди «docker push joker759/trist-lr1:latest»:

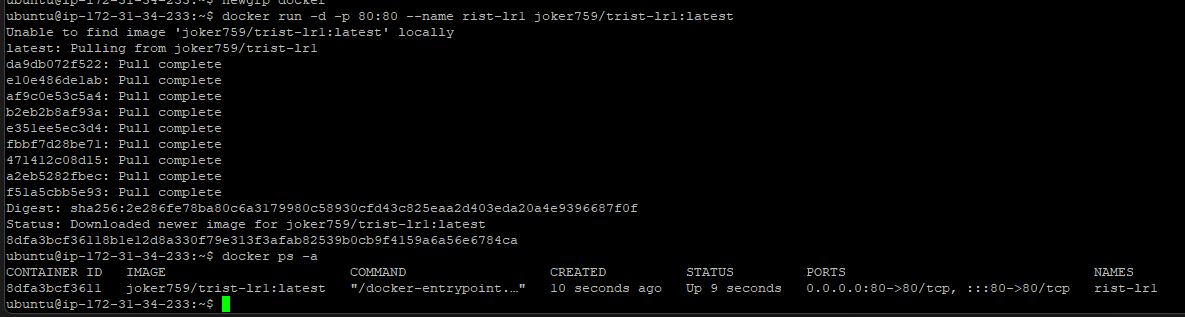


1. Перевіримо на DockerHub, що репозиторій із назвою joker759/trist-lr1 створено і він містить один образ із тегом latest:

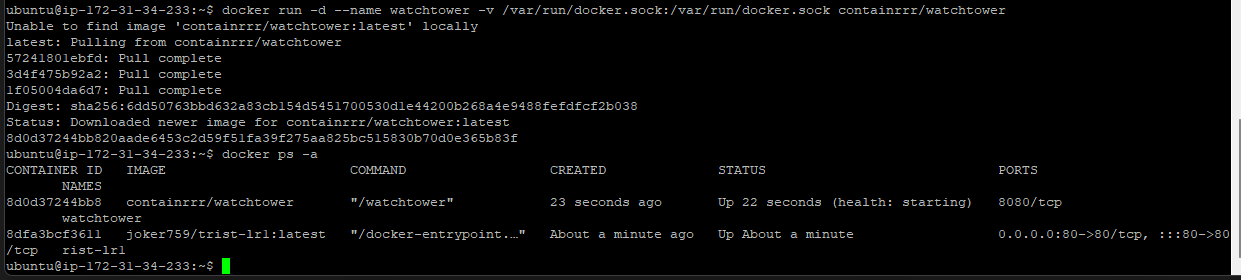




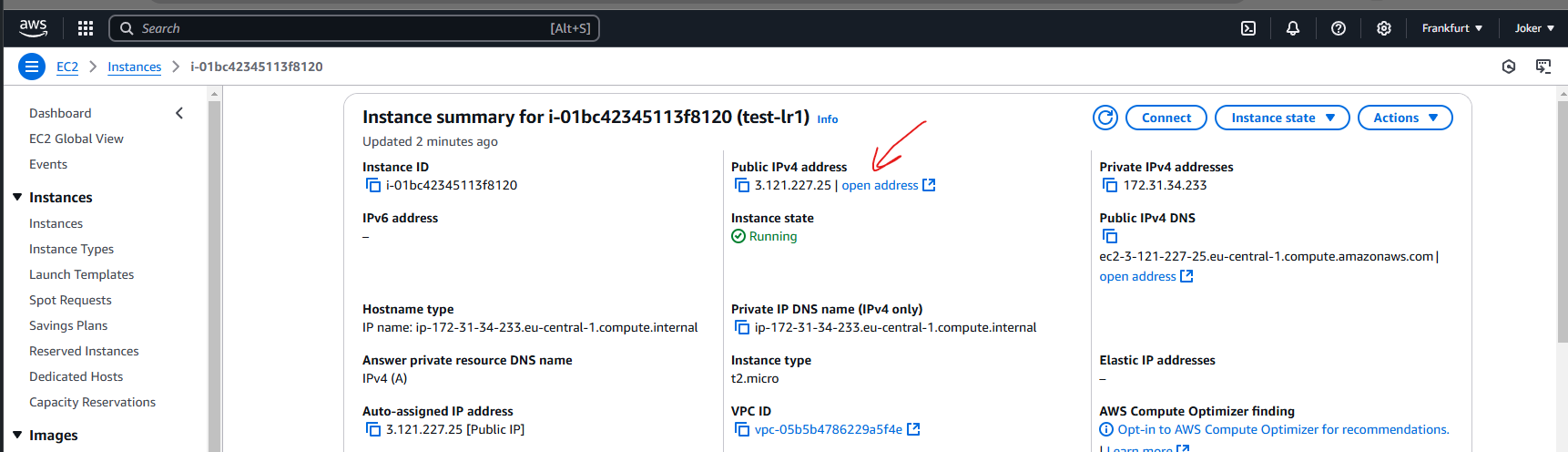
1. Запустимо застосунок як Docker-контейнер на EC2 за допомогою команди «docker run -d -p 80:80 --name rist-lr1 joker759/trist-lr1:latest»:

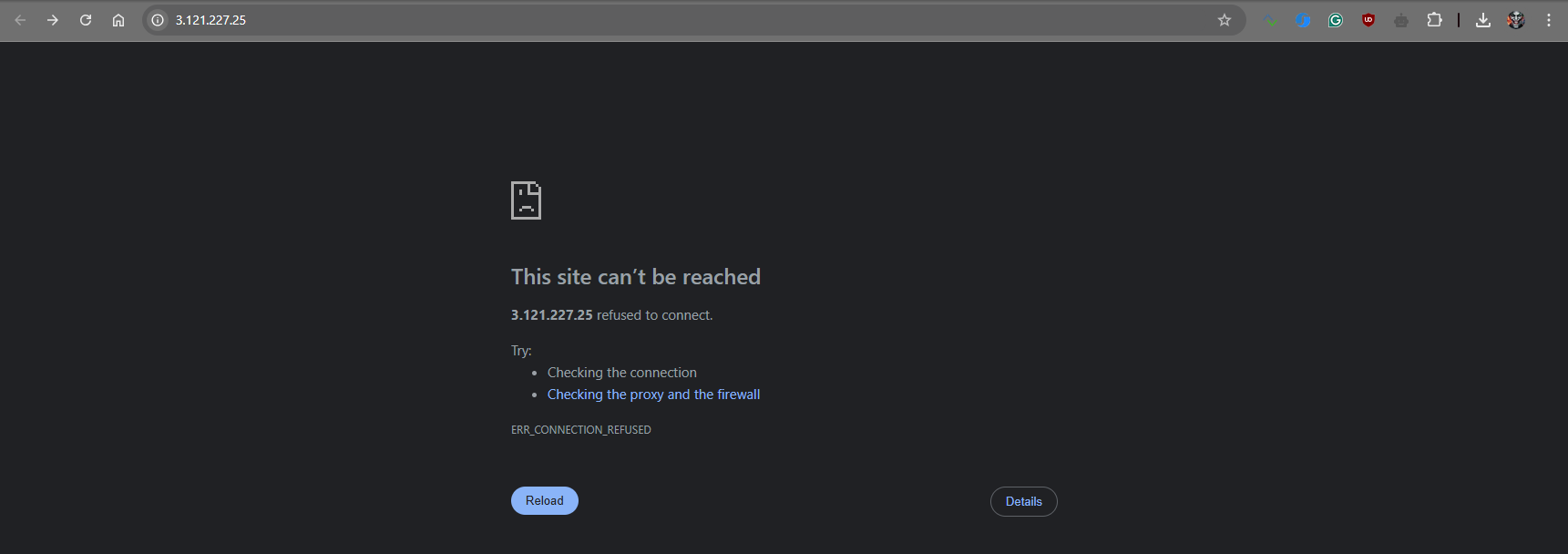


1. Запустимо контейнер із Watchtower для автоматичного оновлення контейнерів за допомогою команди «docker run -d --name watchtower -v /var/run/docker.sock:/var/run/docker.sock containrrr/watchtower»:

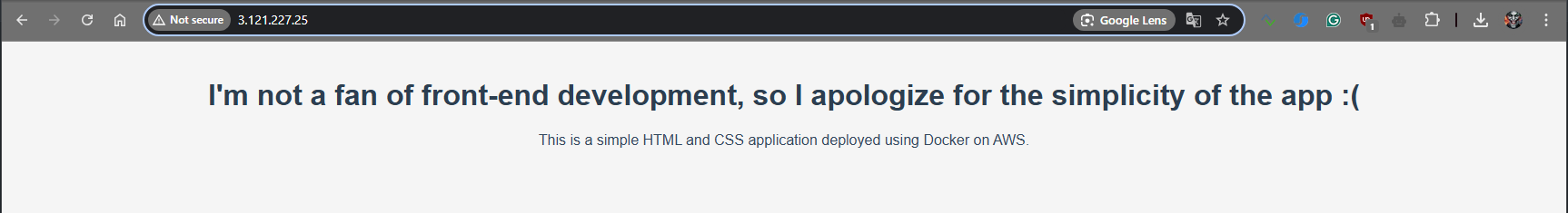


1. Перевіримо, що застосунок працює коректно. На сторінці AWS у вкладці створеного EC2-інстансу знайдемо поле «Public IPv4 address» та перейдемо за відповідним посиланням:



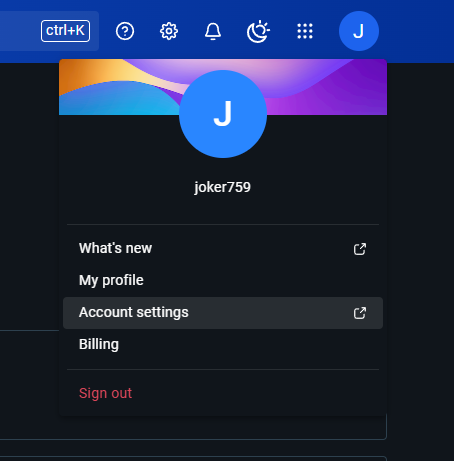


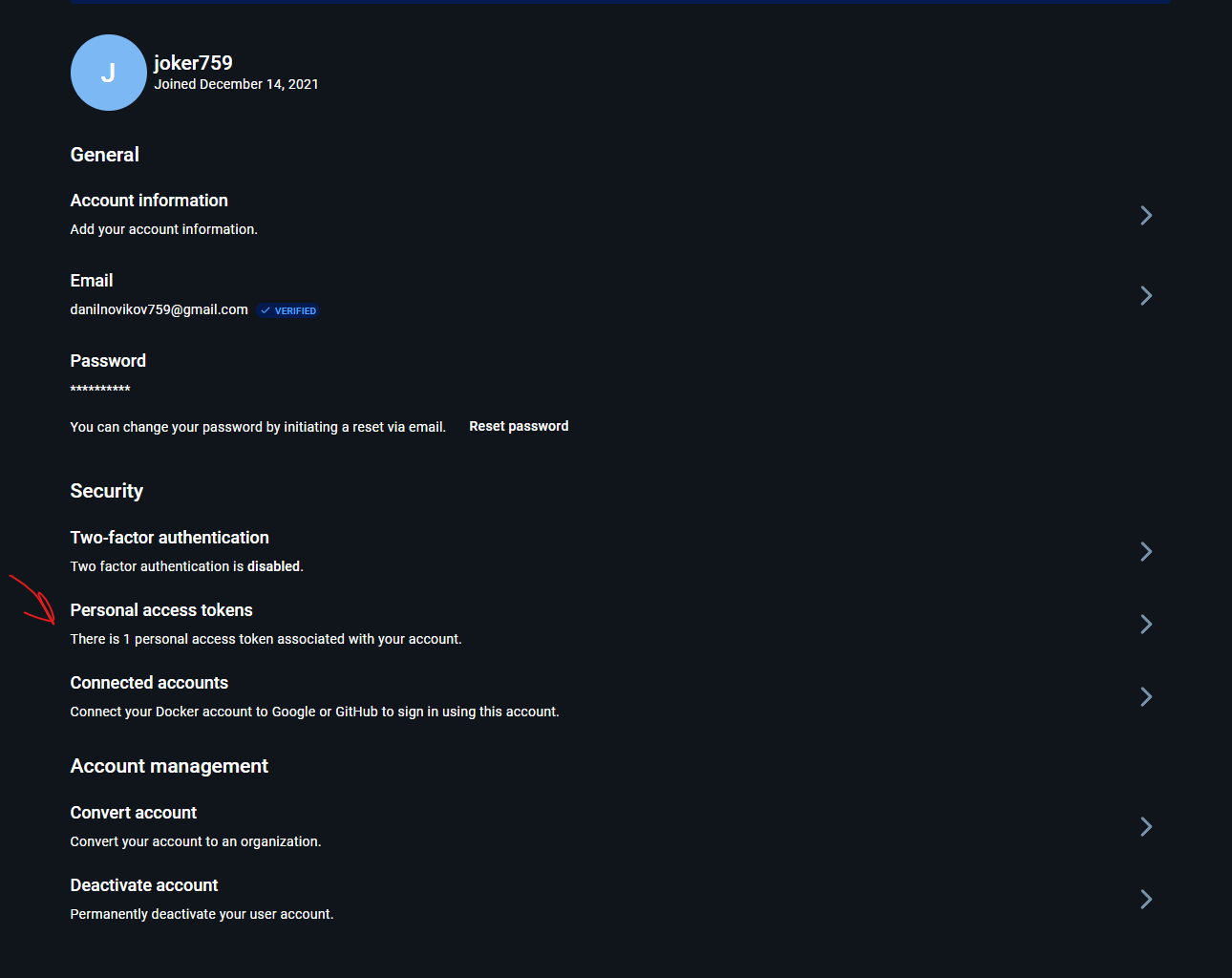
1. Якщо адреса за замовчуванням перенаправляє на HTTPS, виправимо її на HTTP. Застосунок успішно працює:

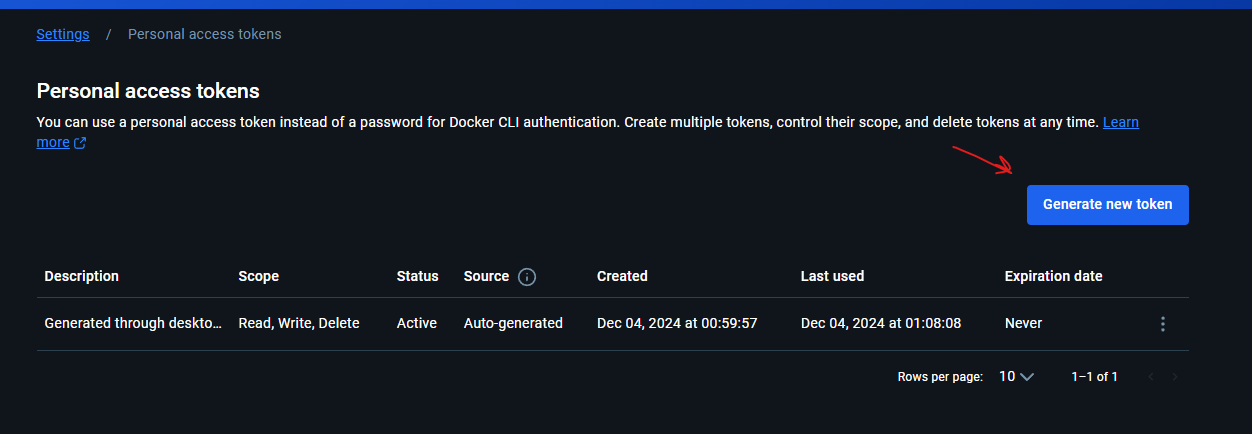


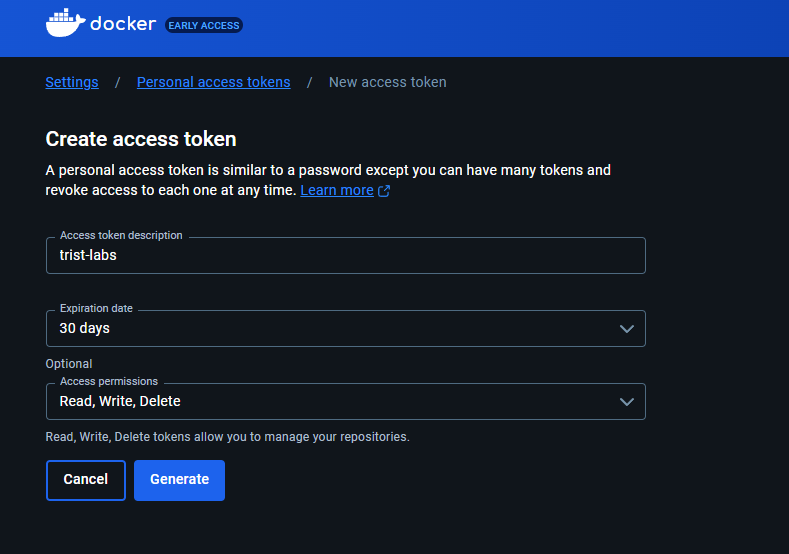
6. Написати github actions pipeline, який збиратиме docker образ і пушитиме його в docker hub при коміті в гілку main.

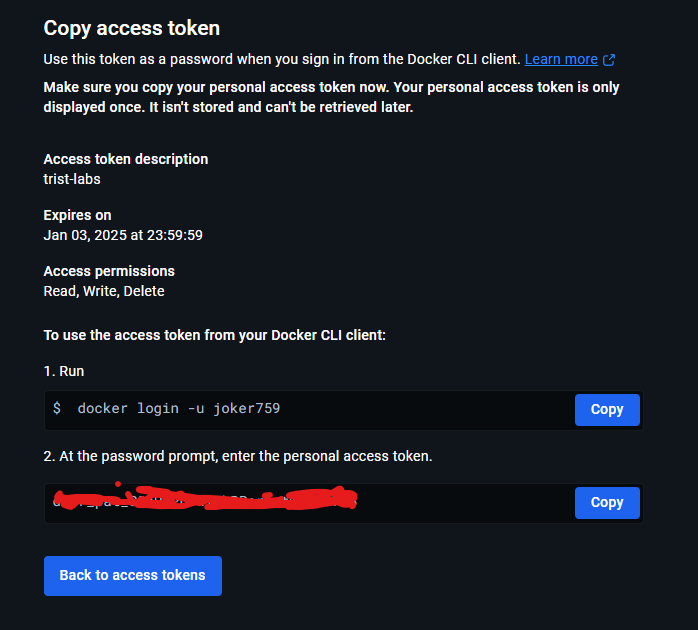
1. Для взаємодією з Dockerhub’ом, нам необхідно отримати токен. Для цього перейдемо в налаштування аккаунта -> Security -> Personal access tokens -> створимо новий Access token (і збережемо для подальшого використання):

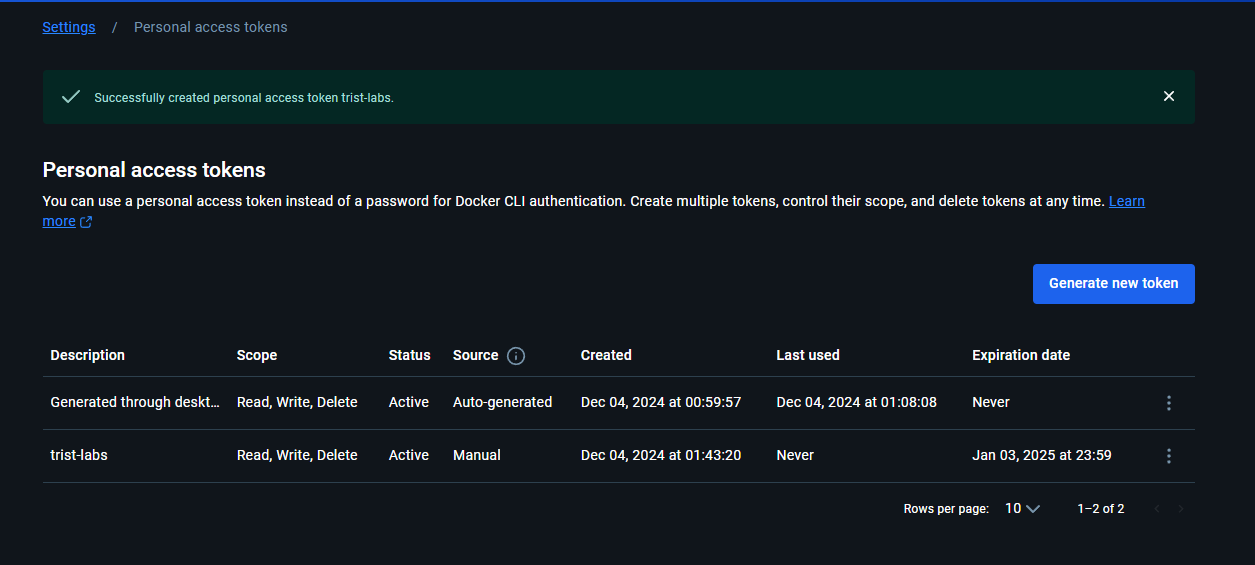




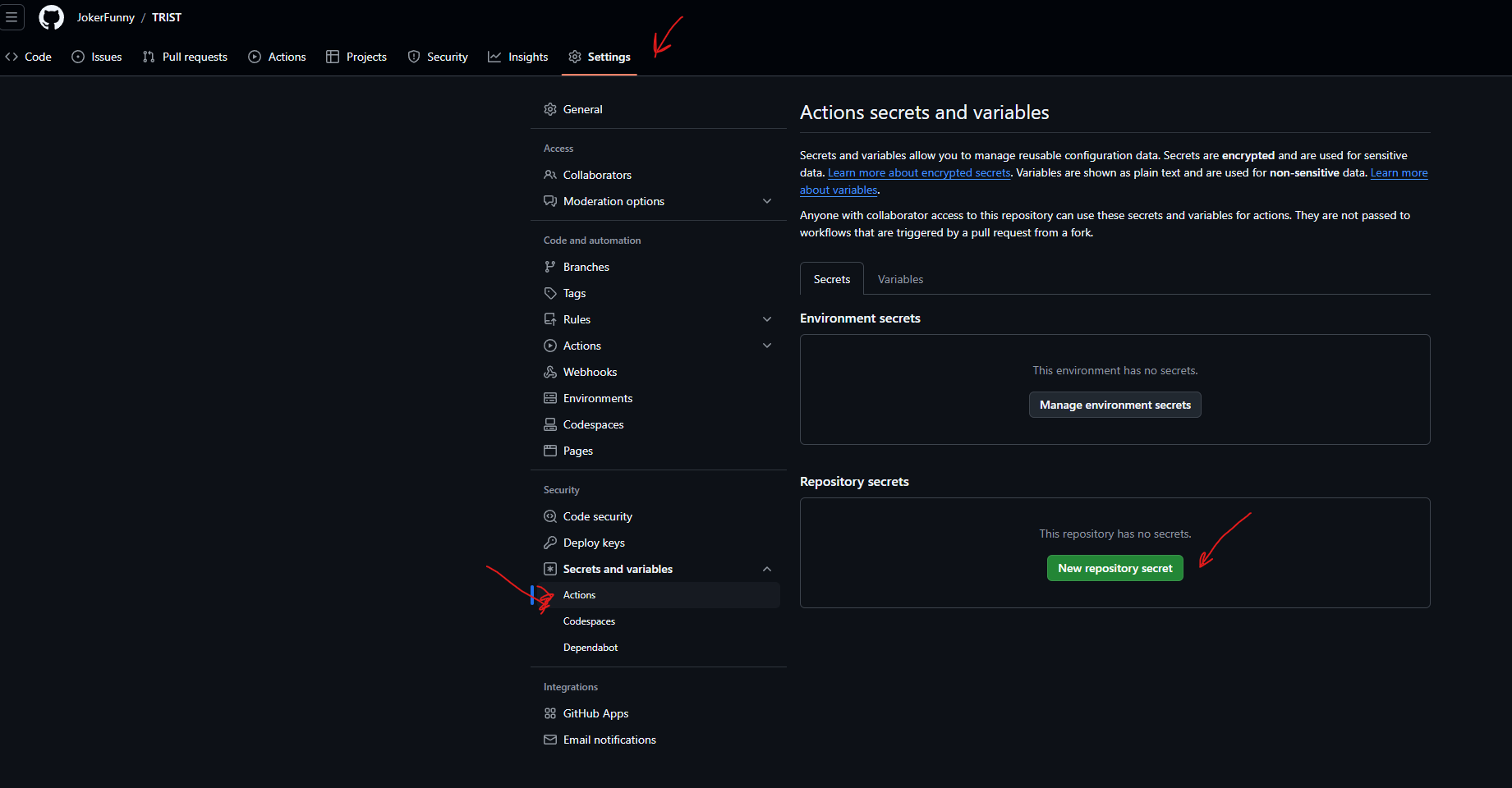


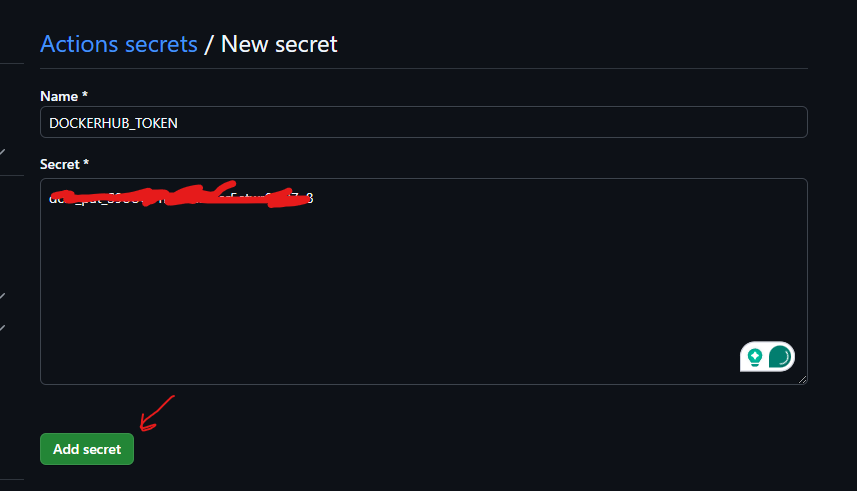


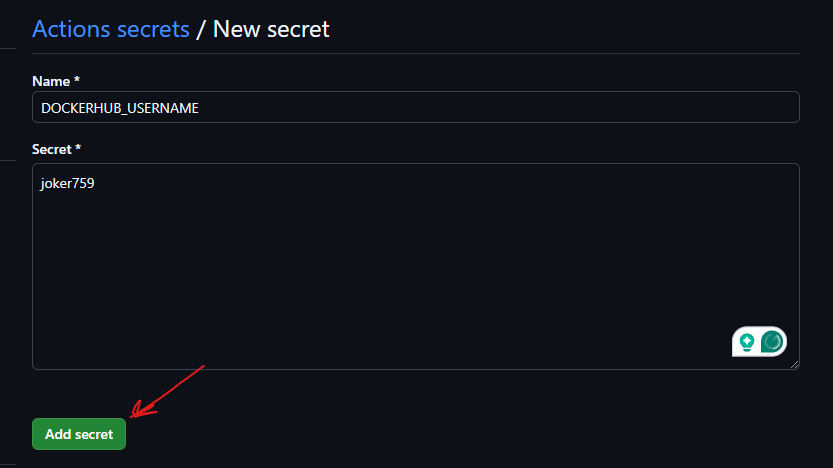


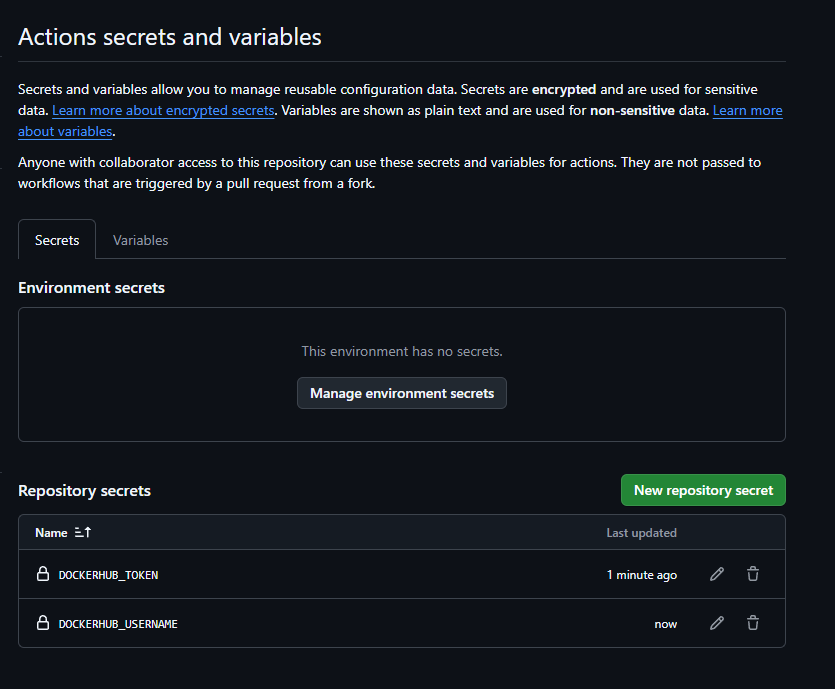


1. Додамо DockerHub username та згенерований токен у GitHub Secrets для подальшого використання під час виконання Github actions:
   * 1. DOCKERHUB\_USERNAME;
     2. DOCKERHUB\_TOKEN.

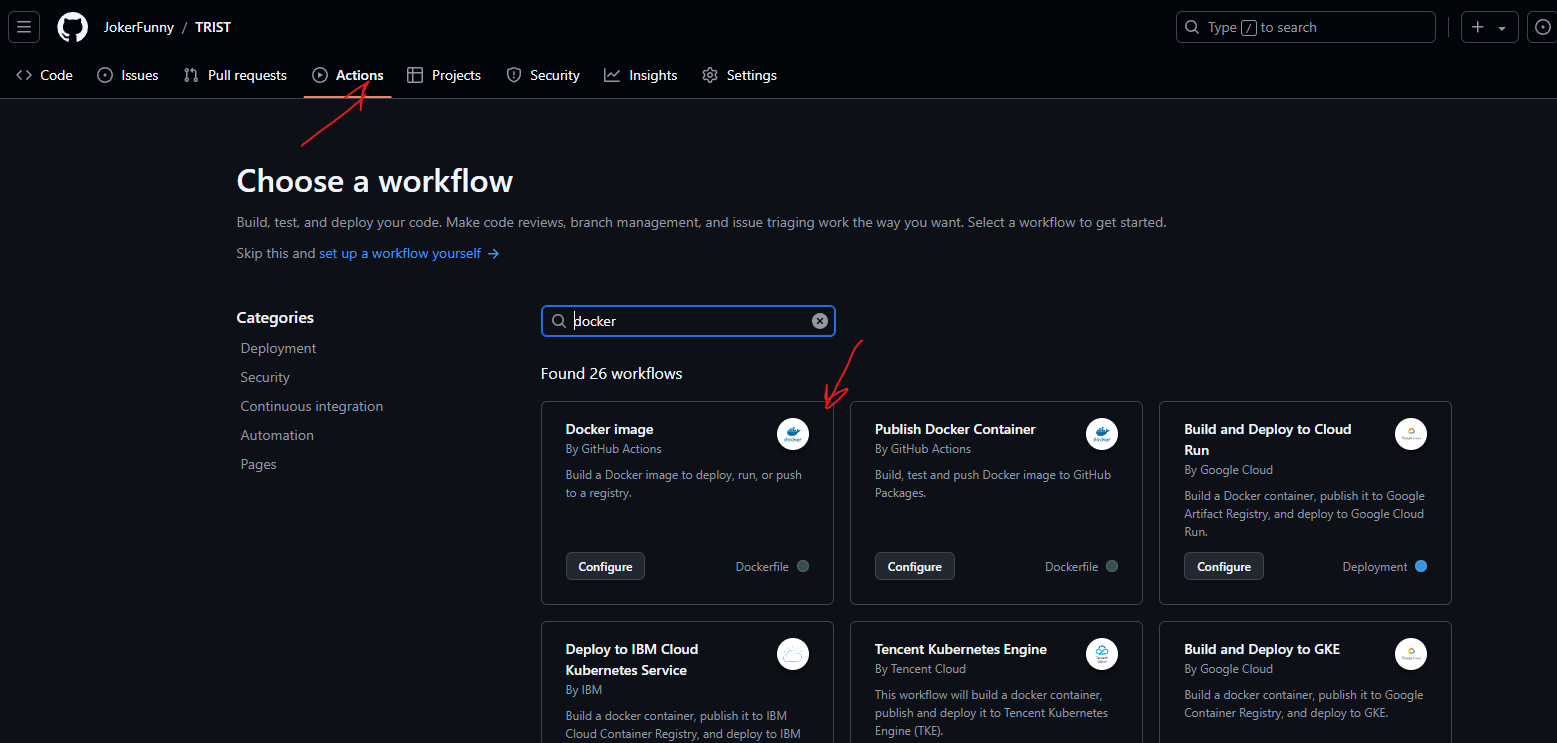




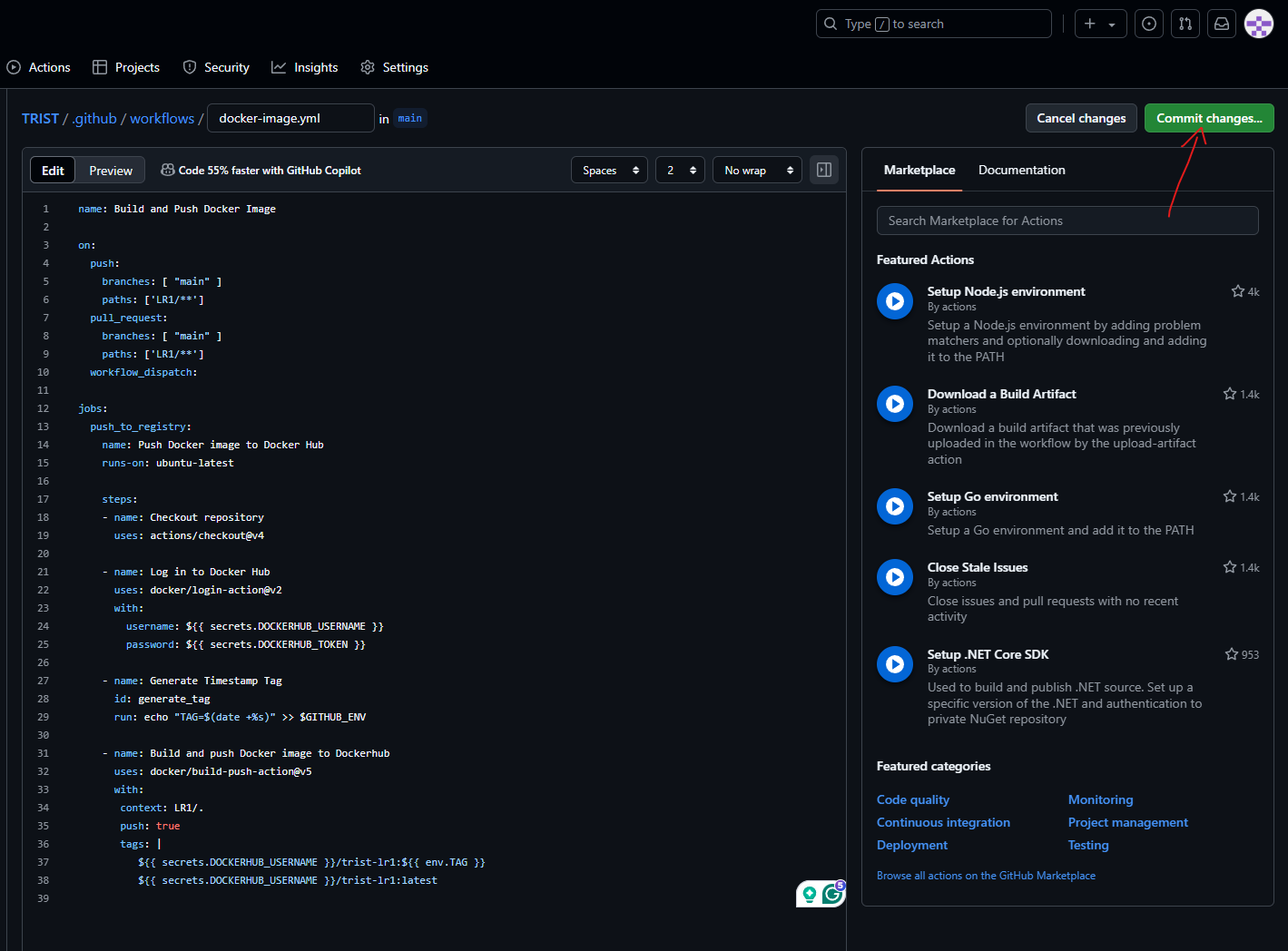




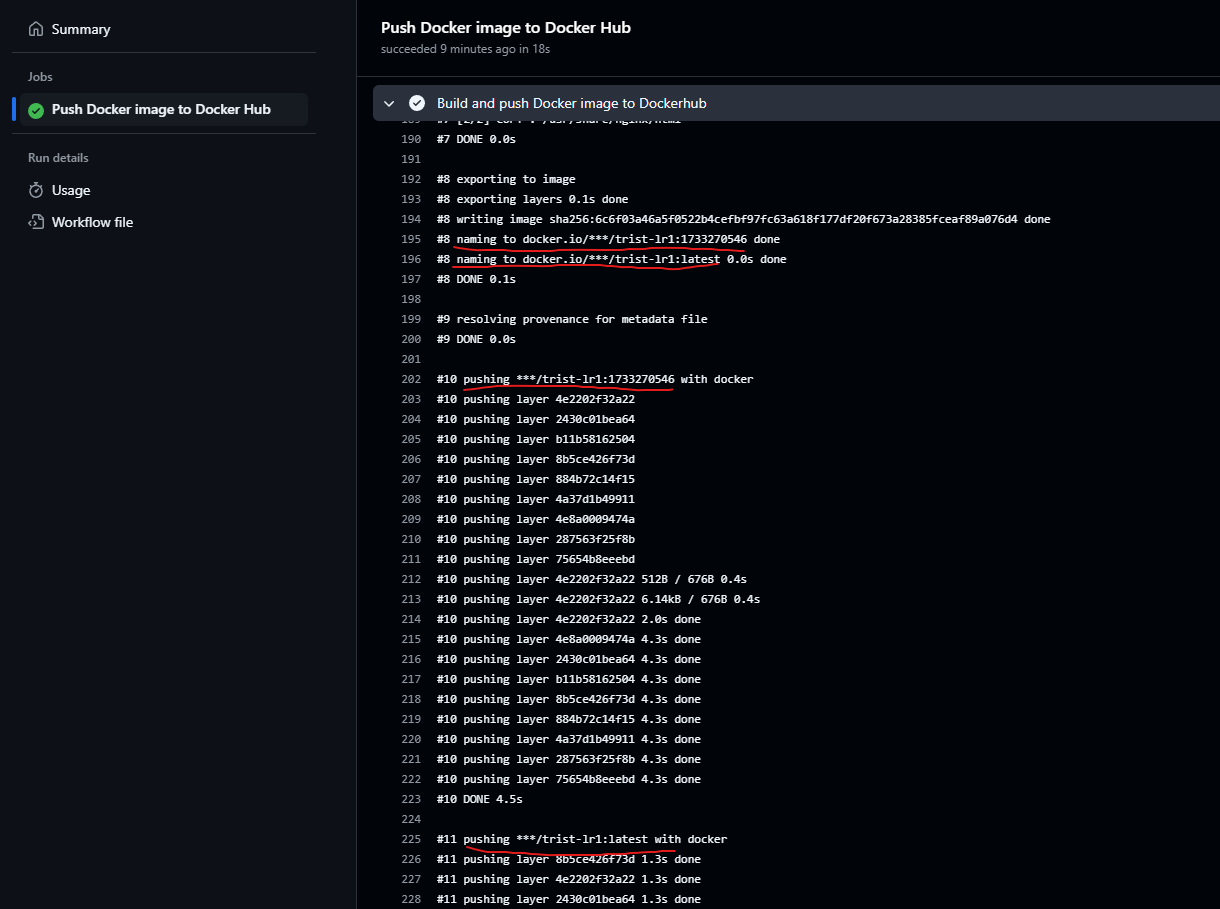
1. Створимо GitHub Actions для автоматизації процесу:



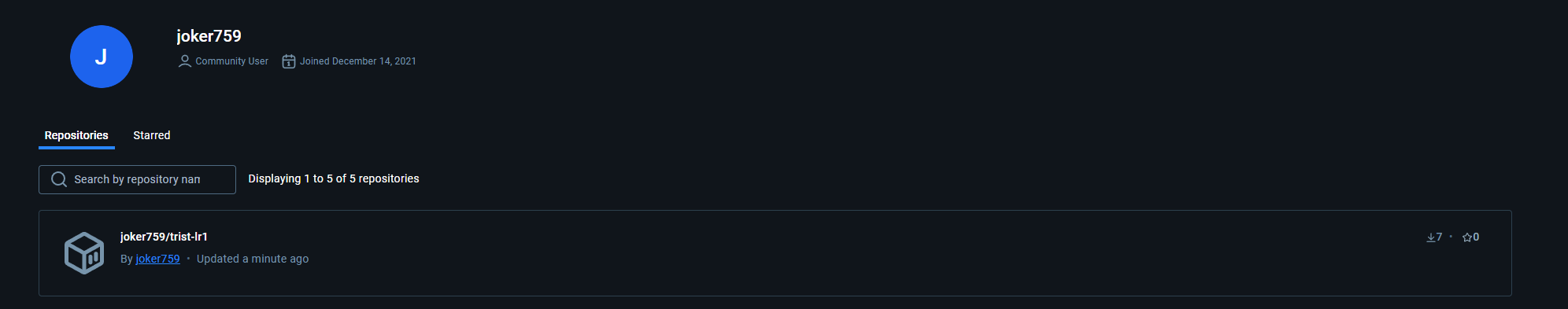
1. Додамо такі кроки у workflow:
   * 1. Перевірка репозиторію: використовується дія actions/checkout, яка завантажує код з GitHub-репозиторію, в якому знаходиться додаток та гарантує, що вся структура проекту (включно з папкою LR1/, де зберігаються Dockerfile, index.html та styles.css) буде доступна для наступних етапів;
     2. Аутентифікація в DockerHub: використовується дія docker/login-action, яка виконує авторизацію в обліковому записі DockerHub, використовуючи секрети DOCKERHUB\_USERNAME та DOCKERHUB\_TOKEN (додані раніше). Це забезпечує можливість публікувати образи;
     3. Генерація мітки часу для тегу Docker-образу: виконує команду echo, щоб створити динамічний тег для Docker-образу на основі поточного часу (у форматі Unix timestamp);
     4. Збірка та пуш Docker-образу: використовується дія docker/build-push-action, яка виконує наступне:
        1. Збірка Docker-образу: використовуючи папку LR1/ як контекст, у якій розташовані Dockerfile і веб-додаток (HTML + CSS), на основі Dockerfile створює образ, що містить веб-додаток, розміщений у контейнері NGINX;
        2. Тегування Docker-образу: додає два теги до одного й того ж образу - тег з міткою часу (наприклад, trist-lr1:1698961321) та тег latest. В результаті маємо 2 образи, що будемо публікувати в DockerHub;
        3. Публікація Docker-образу: пушить зібраний образ у DockerHub з обома тегами.

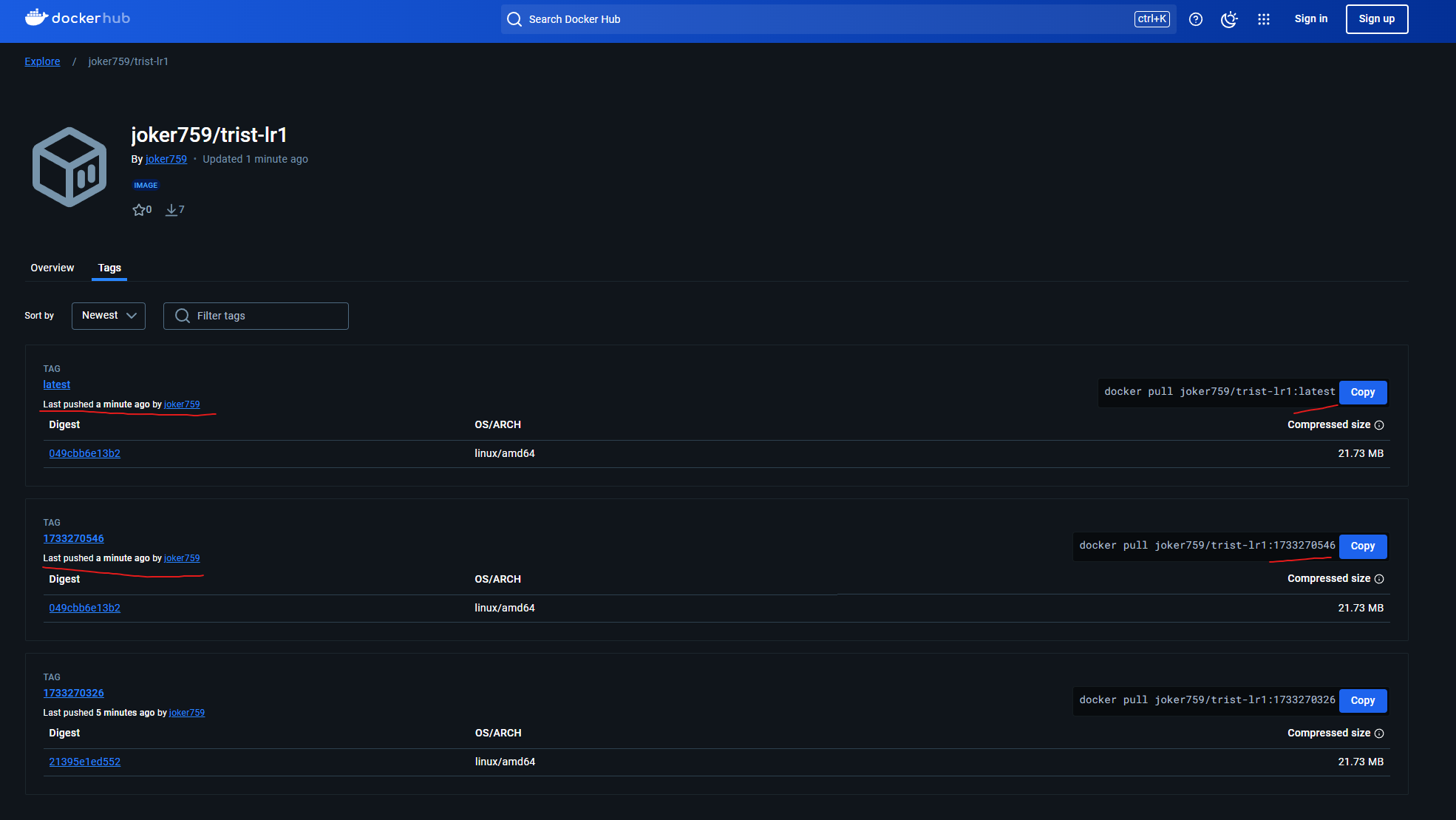


1. Після виконання workflow переглянемо результат у розділі GitHub Actions:
   * 1. Переконаємося, що процес завершився успішно;
     2. У DockerHub було створено 2 образи (із тегами latest та з міткою часу).



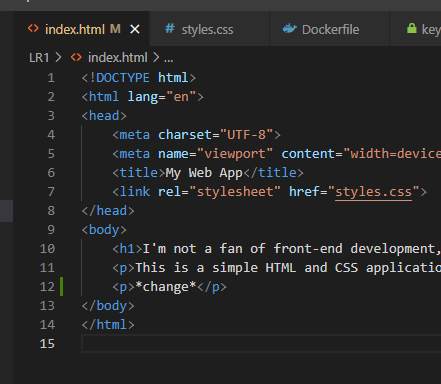
1. Перевіримо DockerHub, щоб переконатися, що нові образи додано з відповідними тегами:



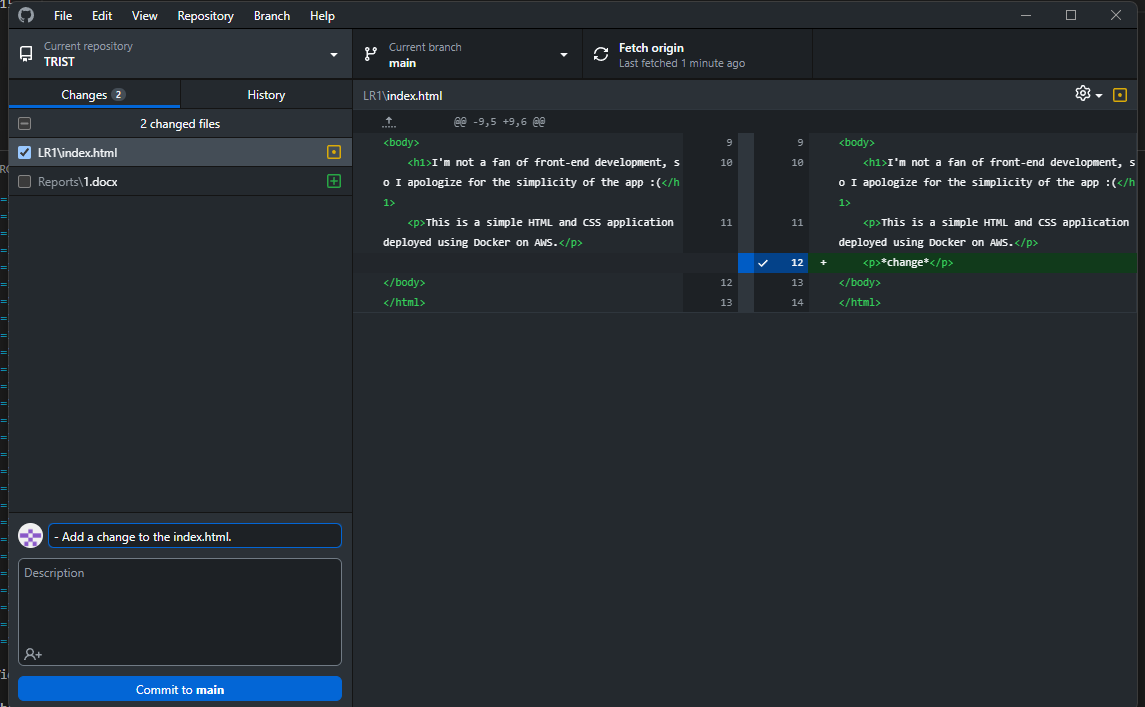


7. Валідація результату:

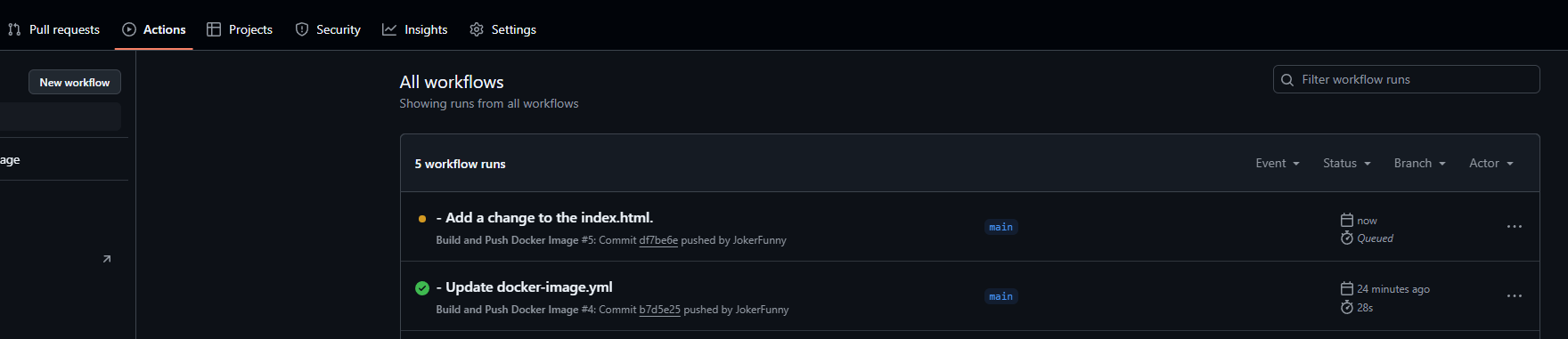
1. Зробимо зміну в застосунку та перевіримо роботу GitHub Actions, оновлення образів у DockerHub, а також відпрацювання Watchtower для автоматичного оновлення контейнера на EC2. Наприклад, додамо текст **change** до файлу index.html:

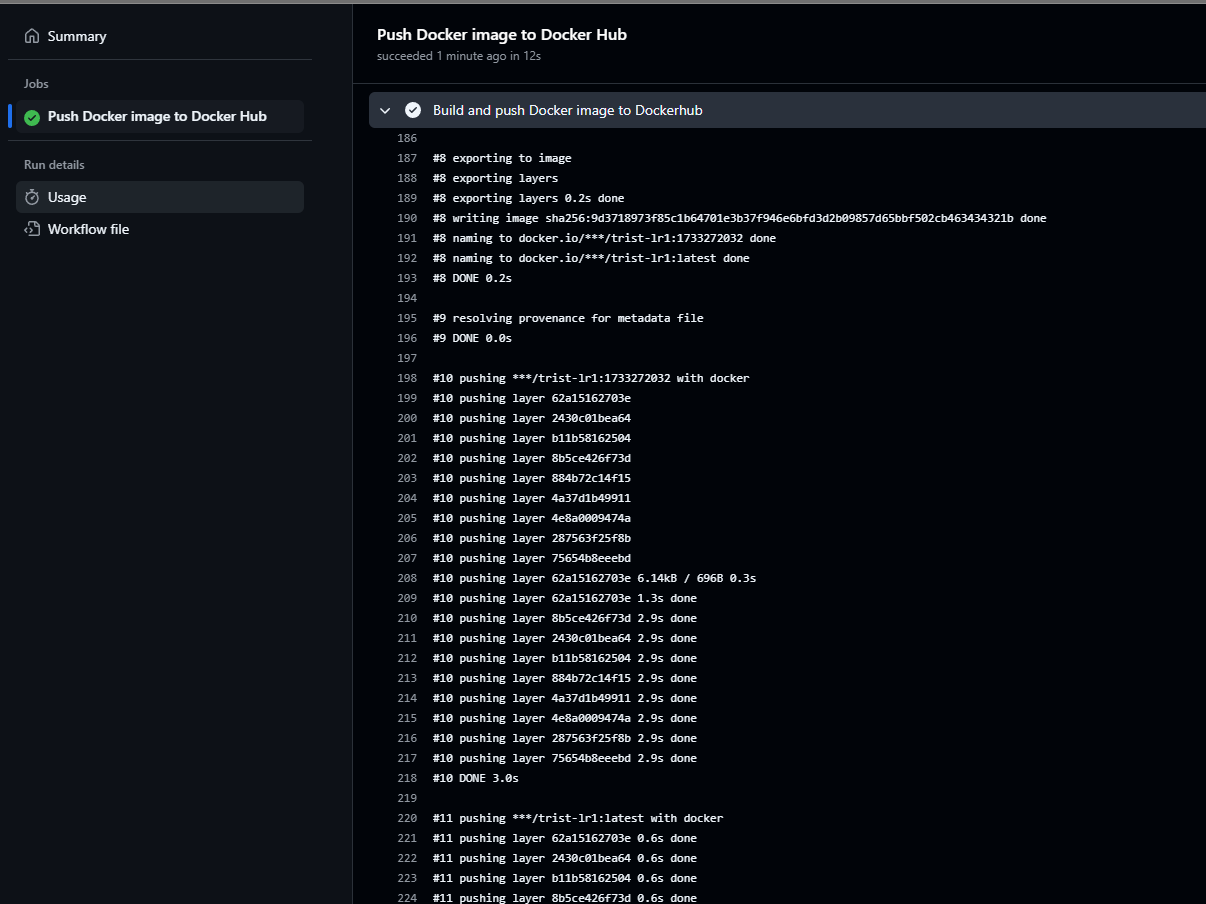


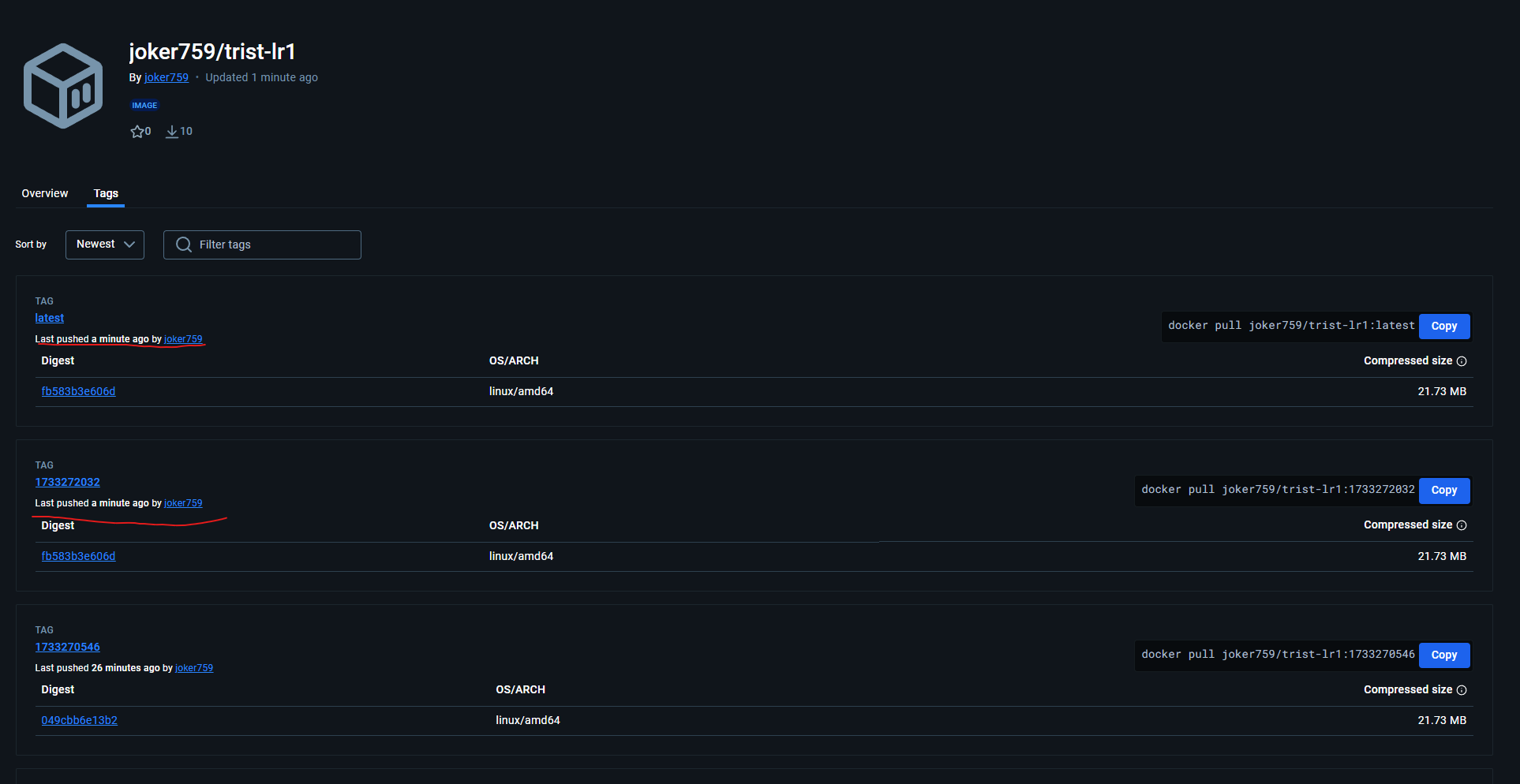
1. Закомітимо оновлену версію файлу до репозиторію:



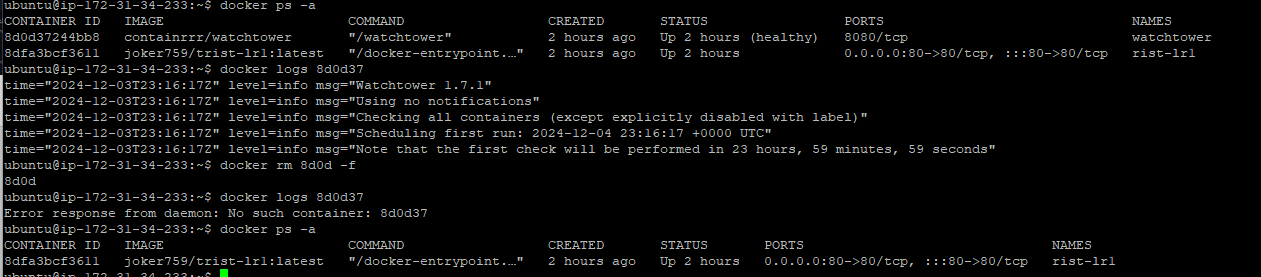
1. У нас автоматично запуститься Github actions, який створить нові образи на основі зміненого файлу та запушить їх в DockerHub:



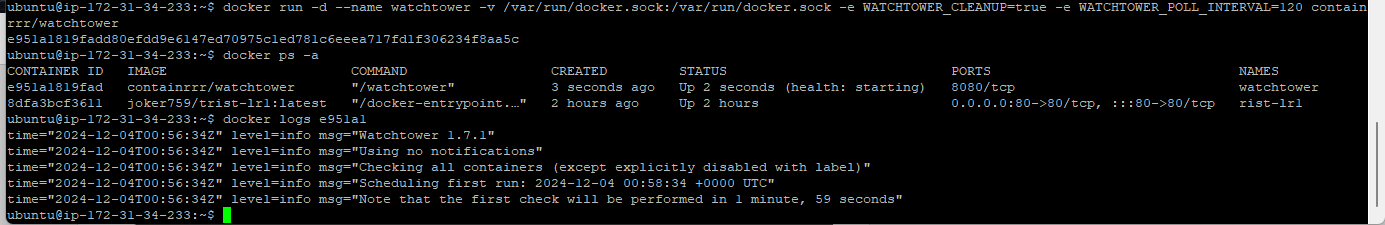




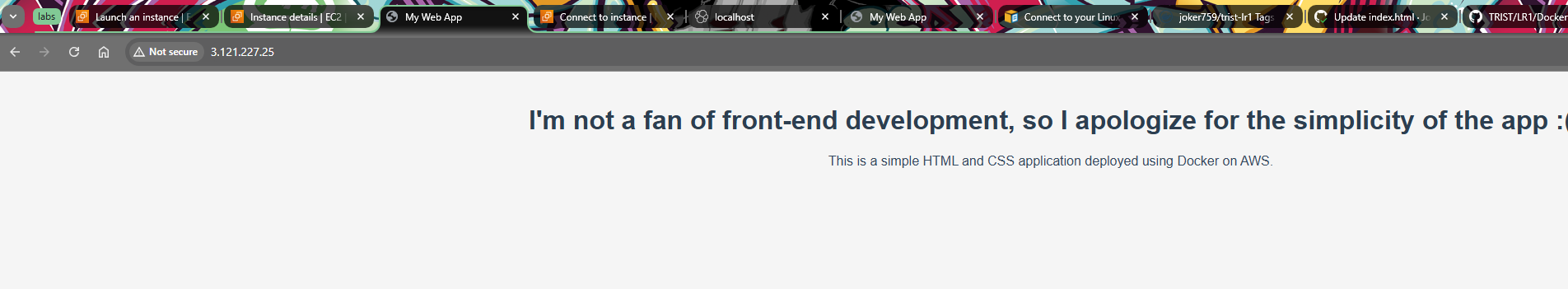
1. Підключимося до EC2 через SSH і перевіримо статус контейнерів за допомогою команди «docker ps -a»:



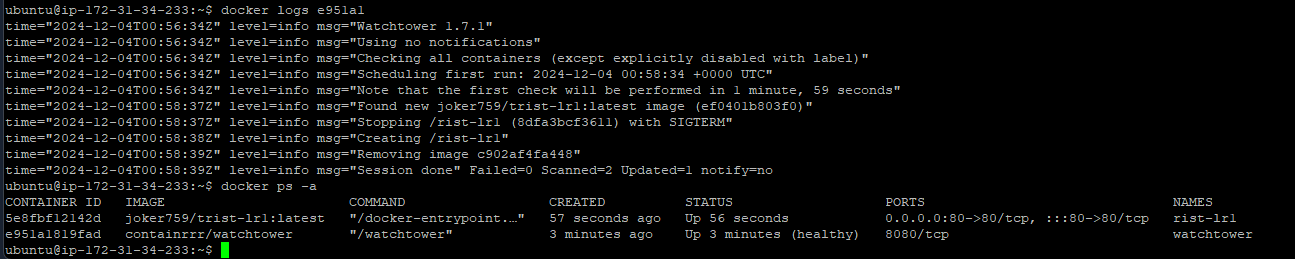
1. Оскільки Watchtower було запущено з дефолтними налаштуваннями, він перевіряє оновлення контейнерів раз на добу (86400 секунд). Щоб пришвидшити процес, перезапустимо контейнер Watchtower із вказанням змінного інтервалу перевірки (вказавши WATCHTOWER\_POLL\_INTERVAL=120, щоб він перевіряв апдейт образа інших запущених контейнерів 1 раз на 120 секунд):



1. Перевіримо, що на EC2 поки що використовується стара версія додатку (до оновлення):



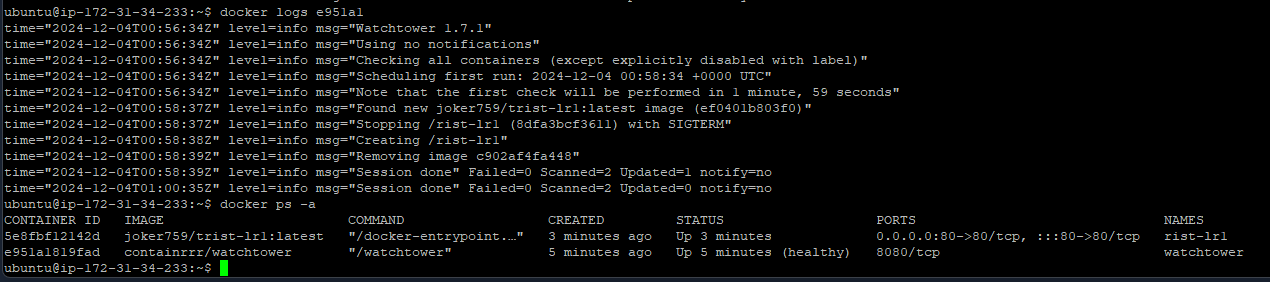
1. Почекаємо декілька хвилин (приблизно 2 хвилини на перевірку Watchtower + час на перезапуск контейнера з новим образом). Після цього перевіримо:
   * 1. Логи контейнеру Watchtower за допомогою команди «docker logs <container\_id>»:



* + 1. Роботу застосунку через браузер або інший клієнт за публічною IP-адресою EC2 (маємо побачити оновлену версію):



1. Перевіримо логи після оновлення контейнера (коли немає нових змін):



*Висновки:* в результаті виконання цієї лабораторної роботи було ознайомлено з базовими концепціями автоматизації розгортання додатків і їх оновлення за допомогою GitHub Actions, Docker, DockerHub та Watchtower.

На основі отриманих знань було реалізовано практичну частину, яка полягала у створенні та автоматизації процесу збірки, публікації й оновлення Docker-образів.

Було успішно налаштовано оновлення застосунку в середовищі AWS EC2 з використанням Watchtower, що забезпечує автоматичне оновлення контейнерів на основі змінених Docker-образів.

Вихідний код застосунку можна знайти за наступним посиланням на [GitHub](https://github.com/JokerFunny/TRIST/tree/main/LR1).

Github actions pipeline доступний за посиланням \*[посилання](https://github.com/JokerFunny/TRIST/actions)\*.