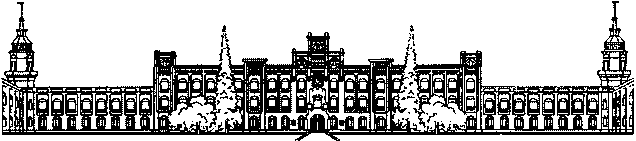
****

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 №2

**Тенденції розвитку інформаційних систем та технологій**

*Інфраструктура як код. Terraform.*

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

Київ 2024

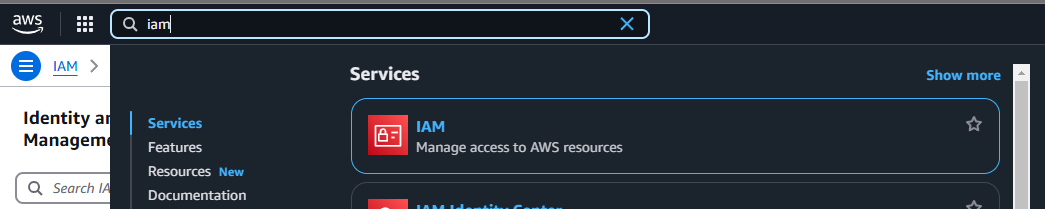
*Мета роботи:* перенести створення інфраструктури з минулої лабораторної роботи в код.

*Хід роботи:*

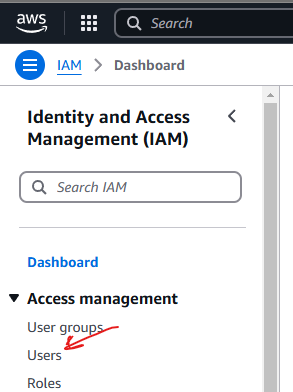
1. Створити ключ-файл, за допомогою якого будете підключатися до інфраструктури (можна використати з минулої лабораторної роботи):

Використаємо ключ з попередньої ЛР.

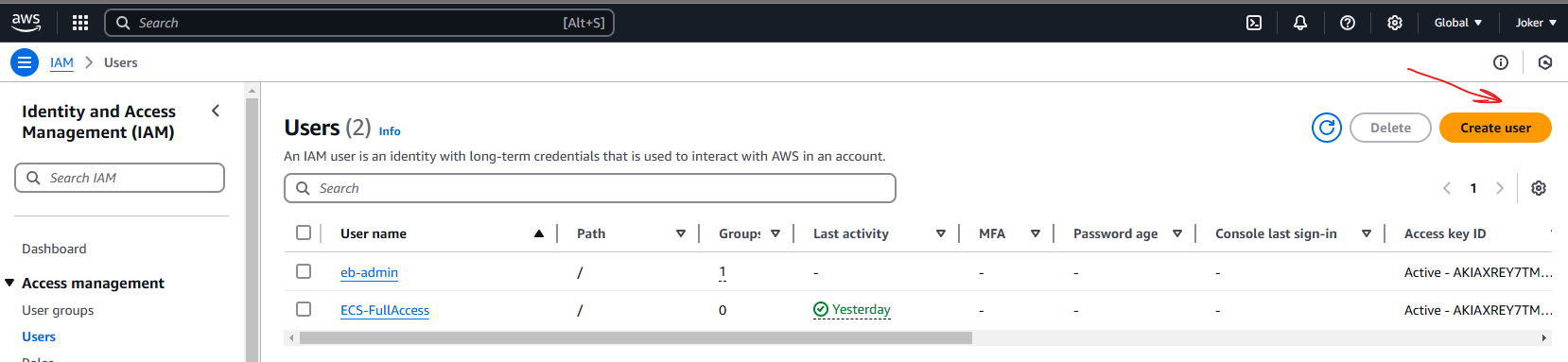
1. Створити IAM ключ, за допомогою якого буде доступ до створення інстансів, та зберегти ID та KEY згідно з інструкцією, з яким ви ознайомились під час опрацювання матеріалу:
   1. Перейдемо до IAM у AWS Management Console:

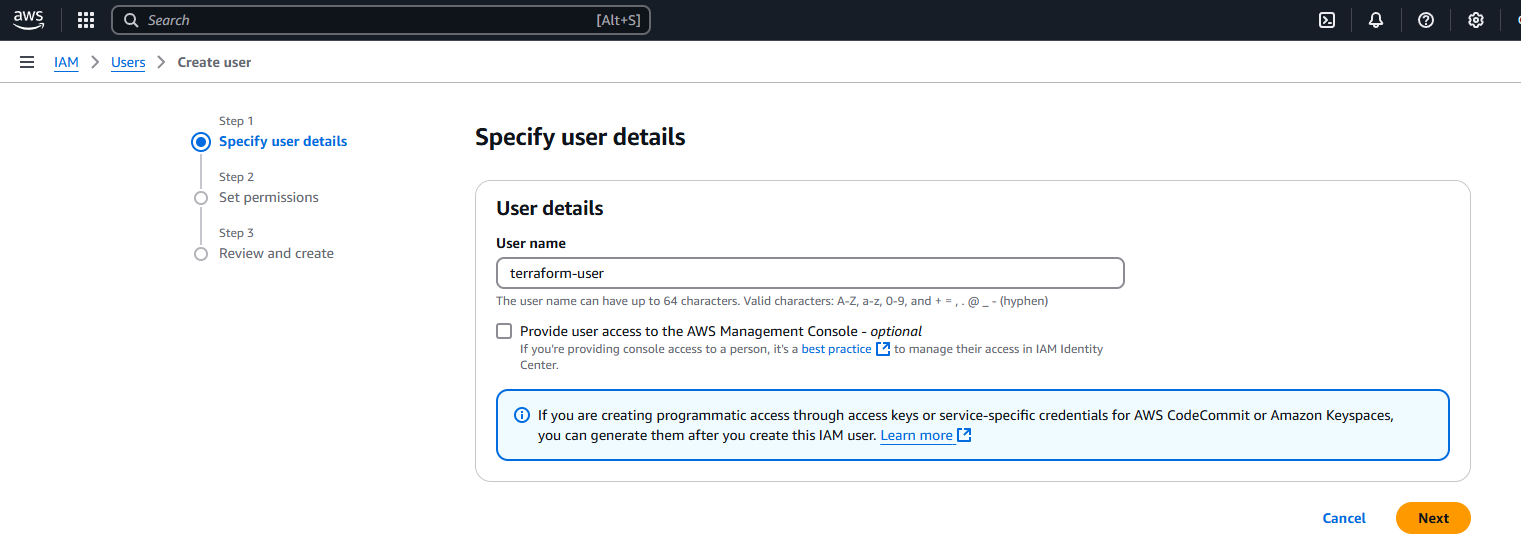


* 1. Перейдемо до розділу Users (Користувачі):

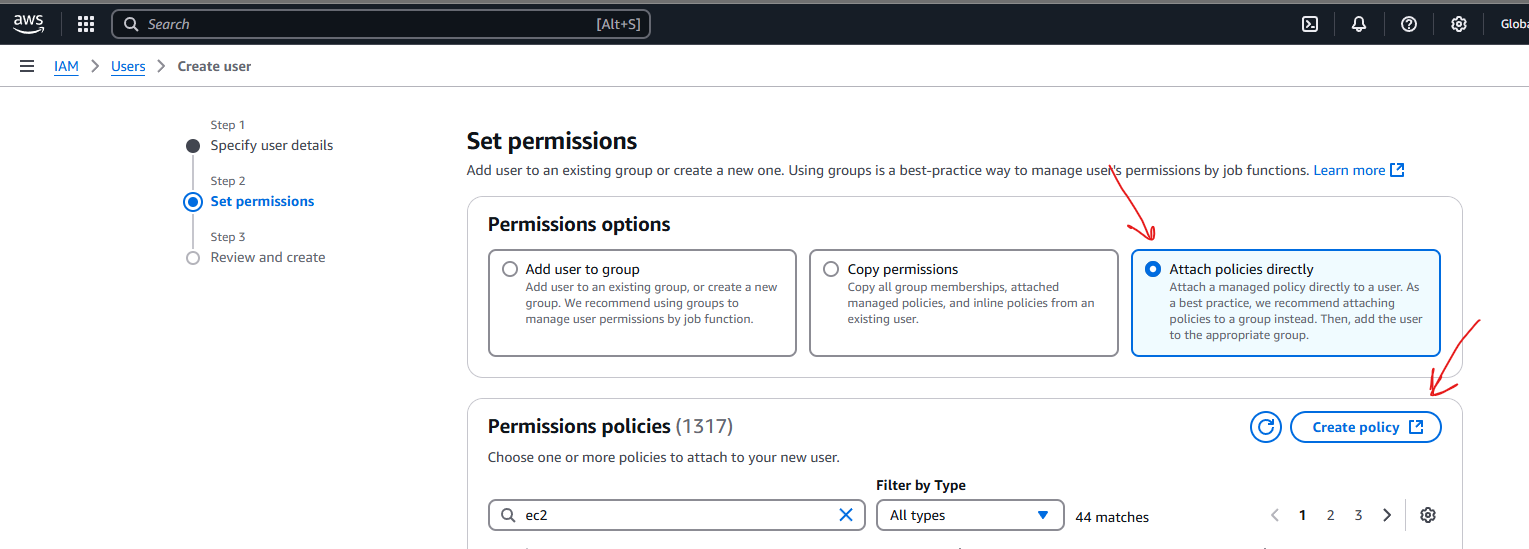


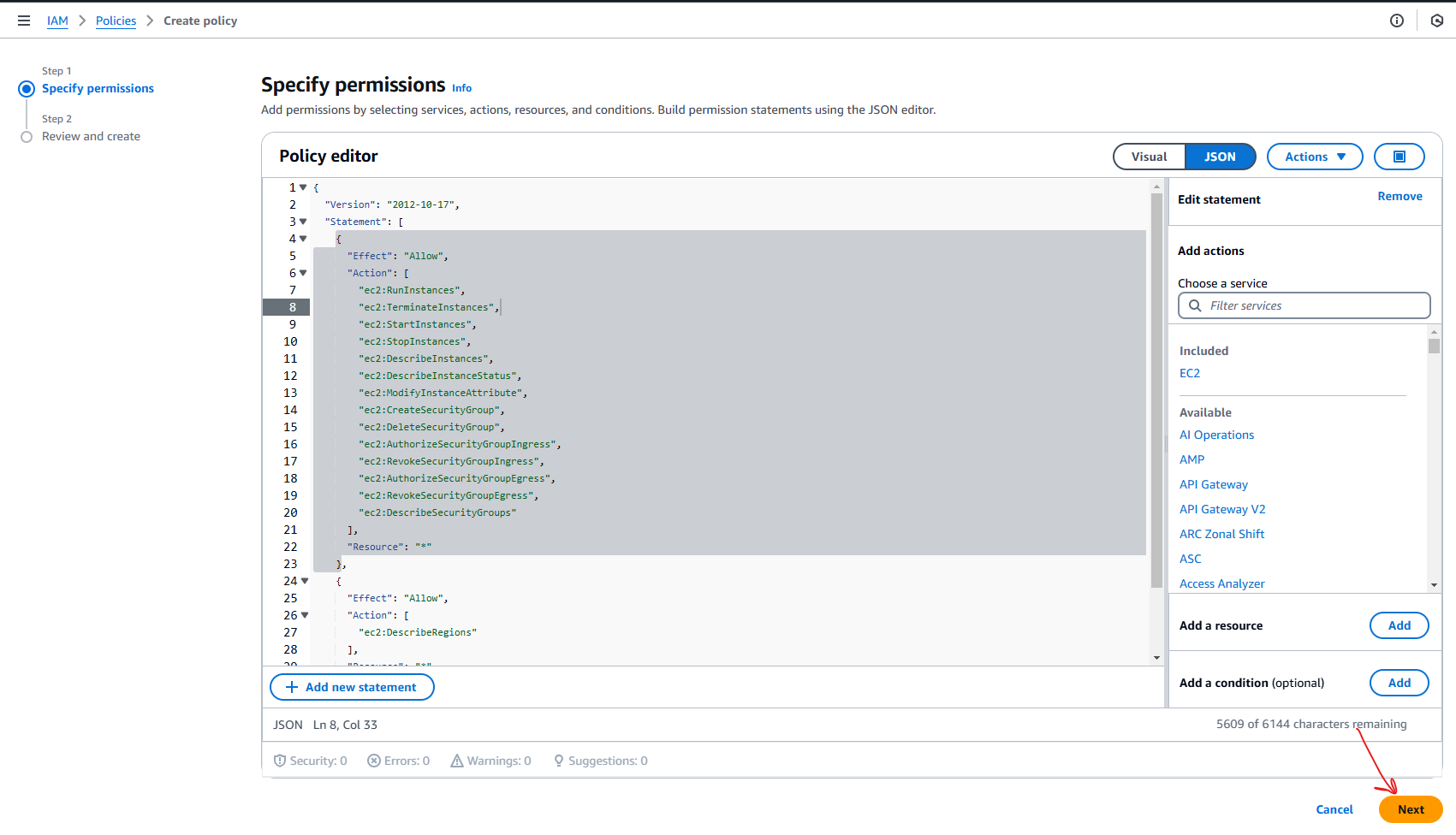
* 1. Створимо нового користувача, наприклад, terraform-user:

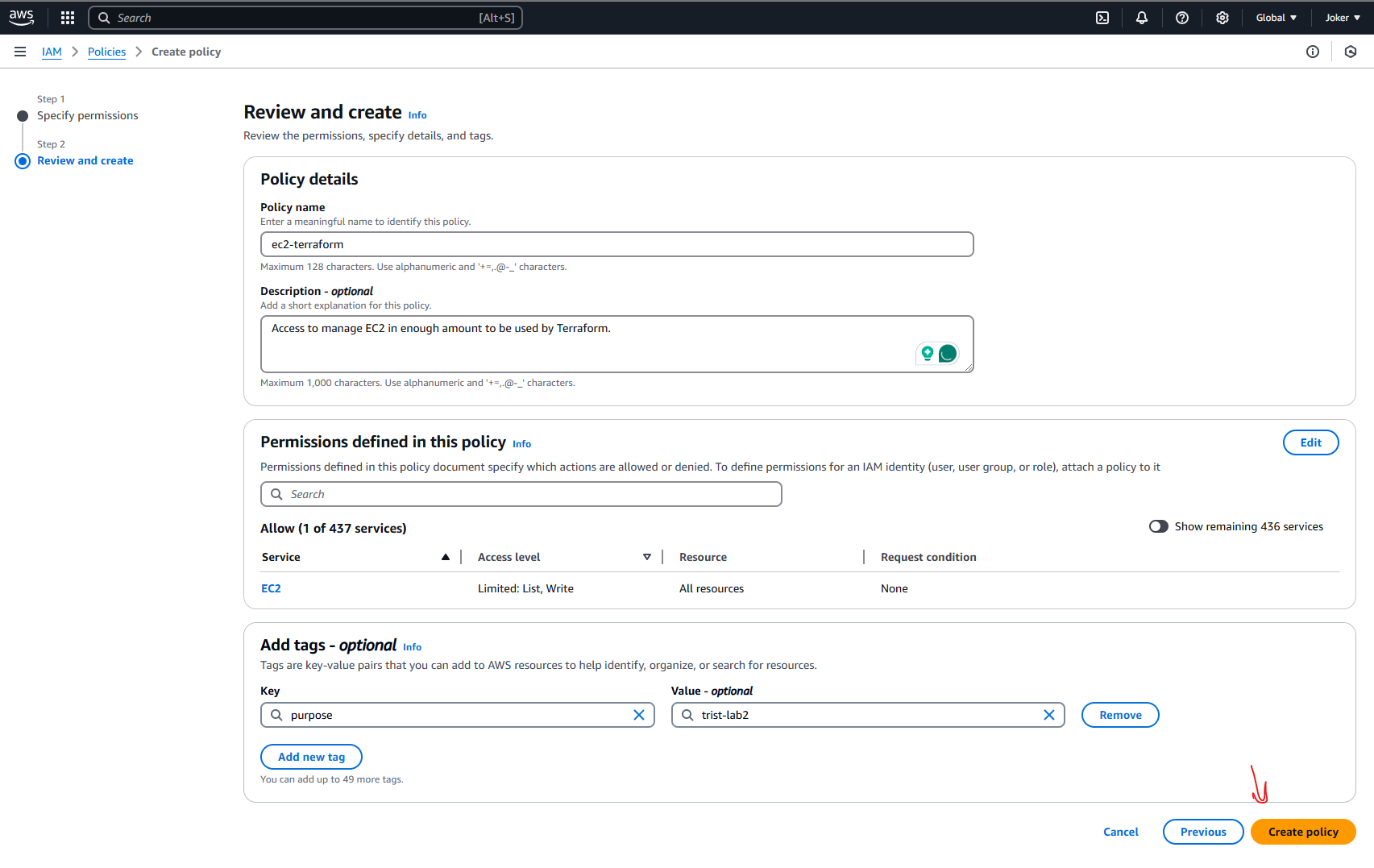




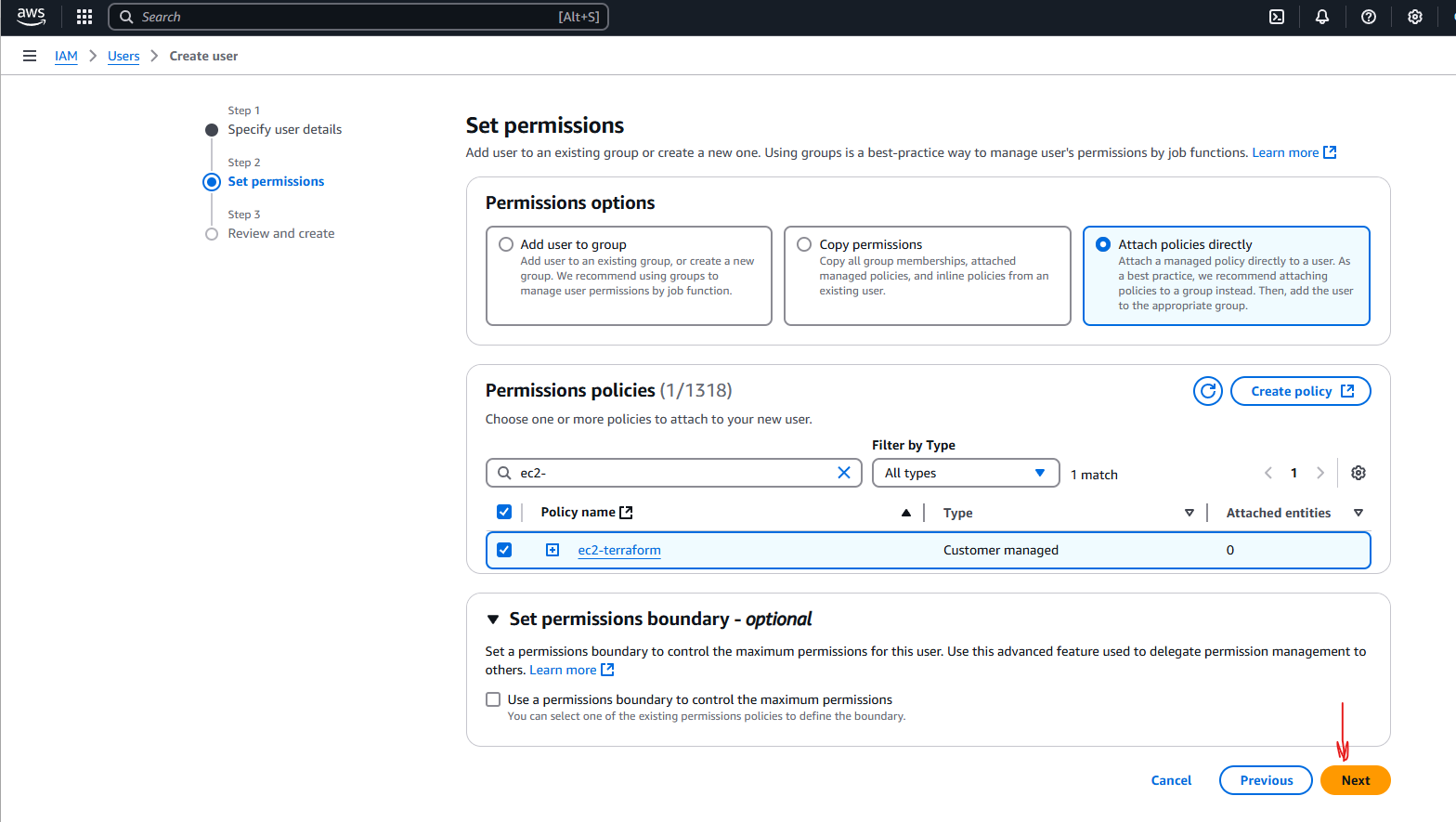
* 1. Щоб уникнути надання повного доступу (навіть для лабораторної роботи, оскільки це суперечить best practices), створимо нову policy із мінімально необхідними правами:

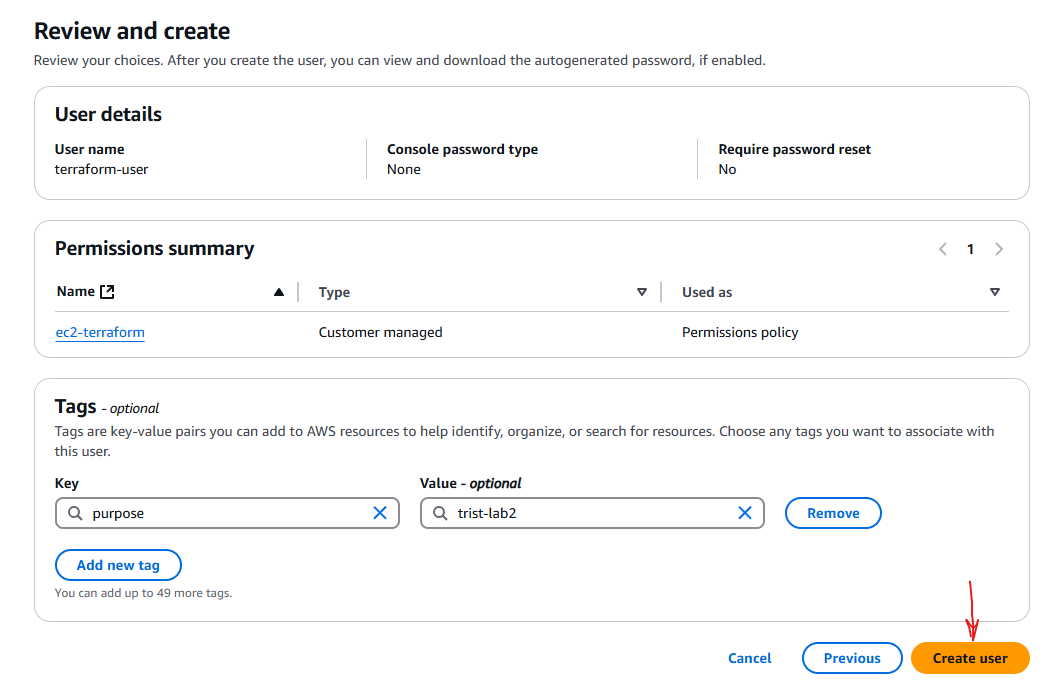




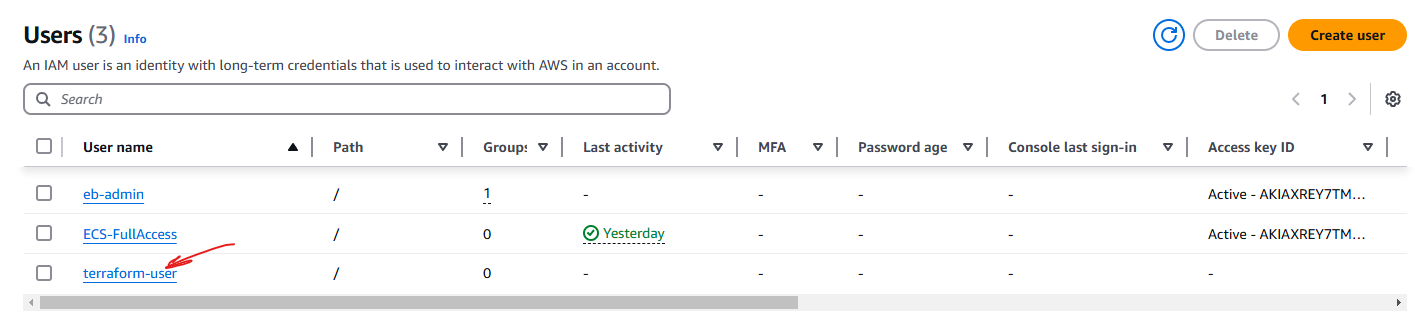


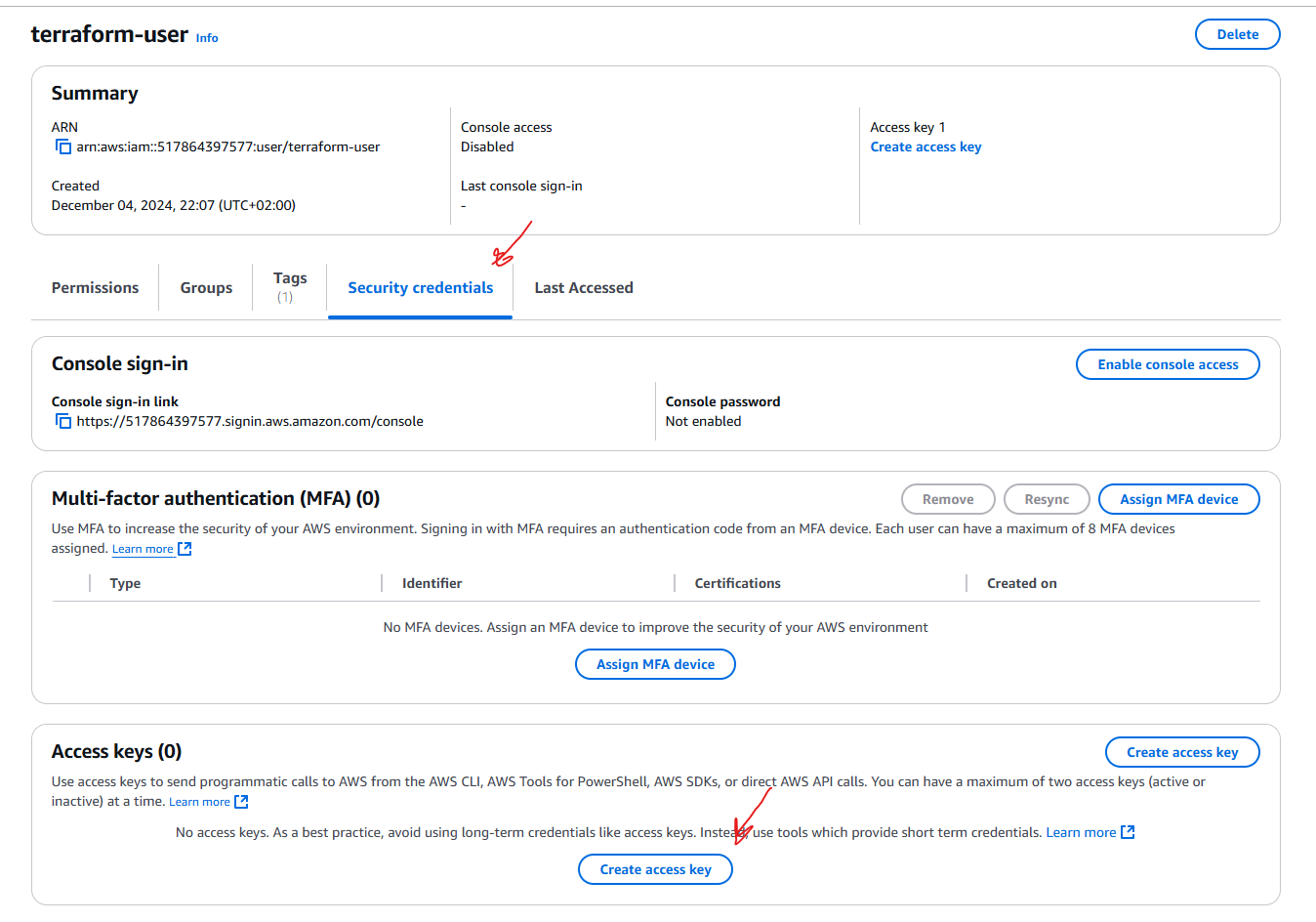
* 1. Повернемося до процесу створення користувача IAM, оберемо новостворену policy, перейдемо далі та завершимо створення користувача:

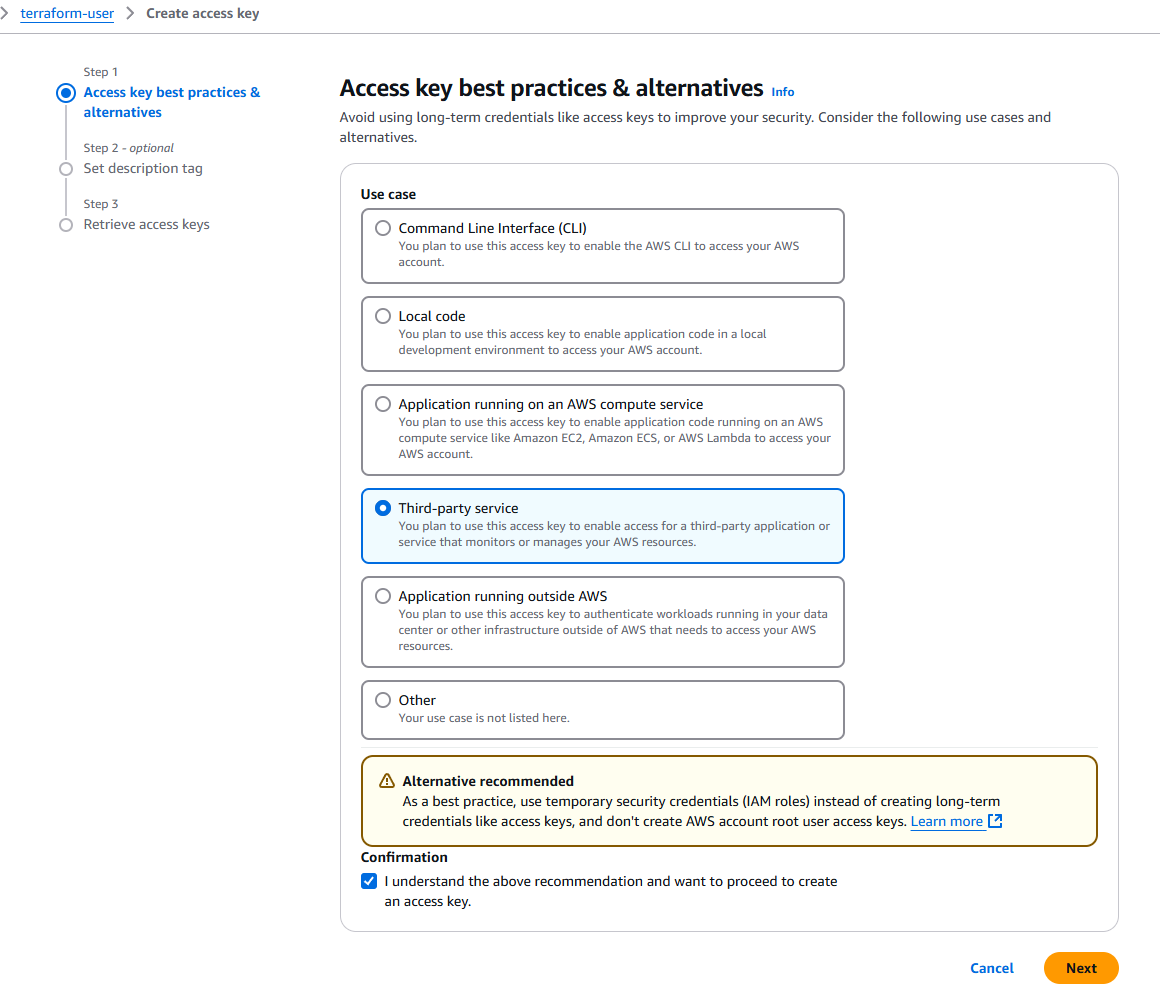


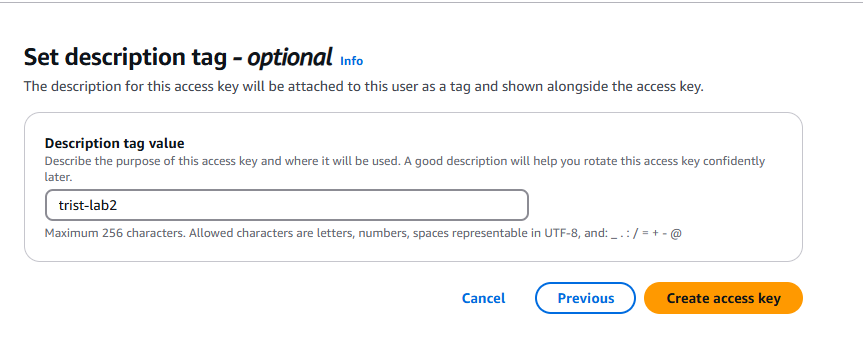


* 1. Створимо Access Key для нового користувача:

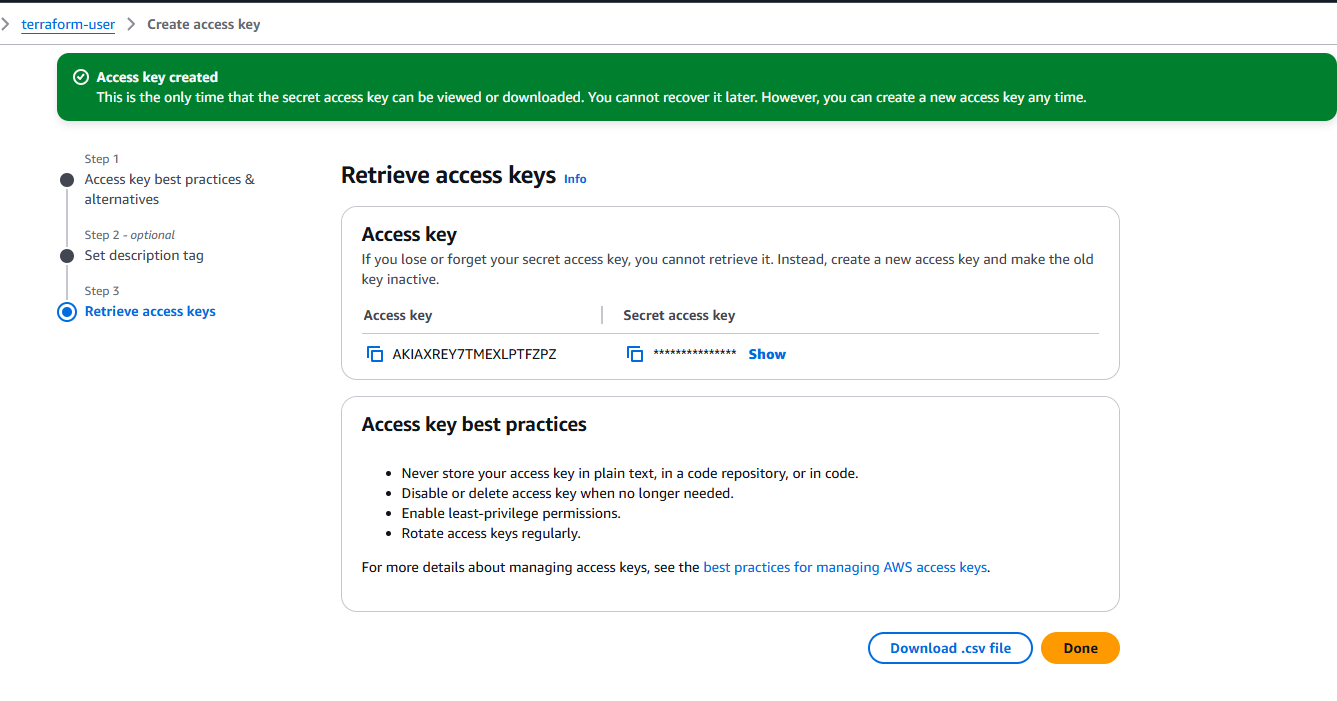


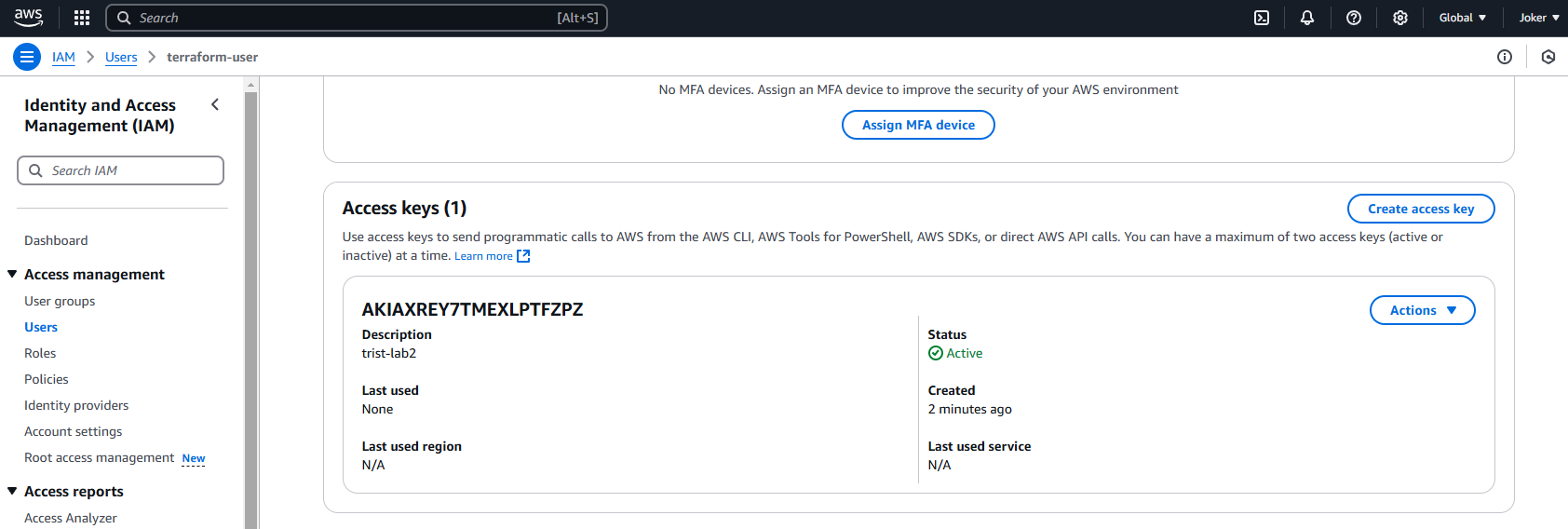




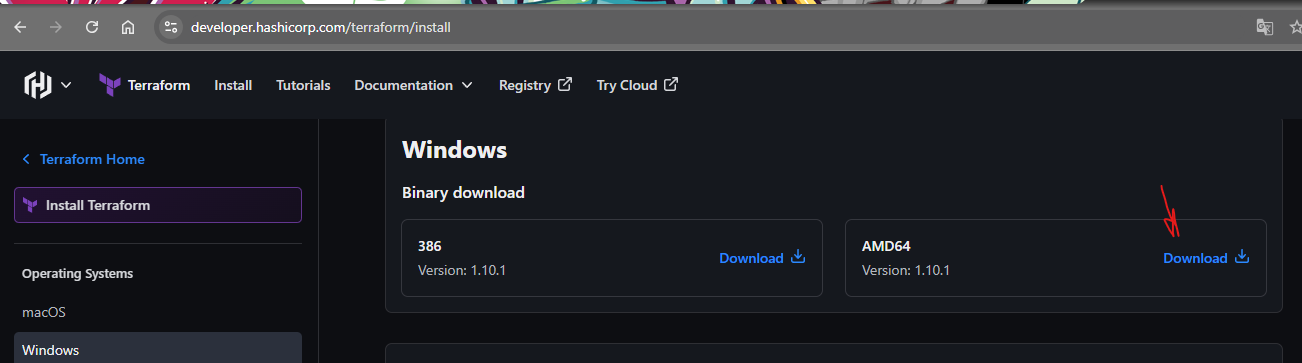


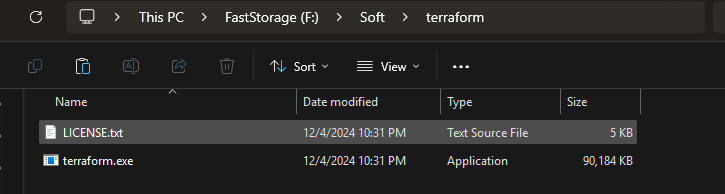
(ВАЖЛИВО: на цьому кроці обов’язково збережемо отримані значення Access Key ID та Secret Access Key, оскільки їх неможливо переглянути повторно!)



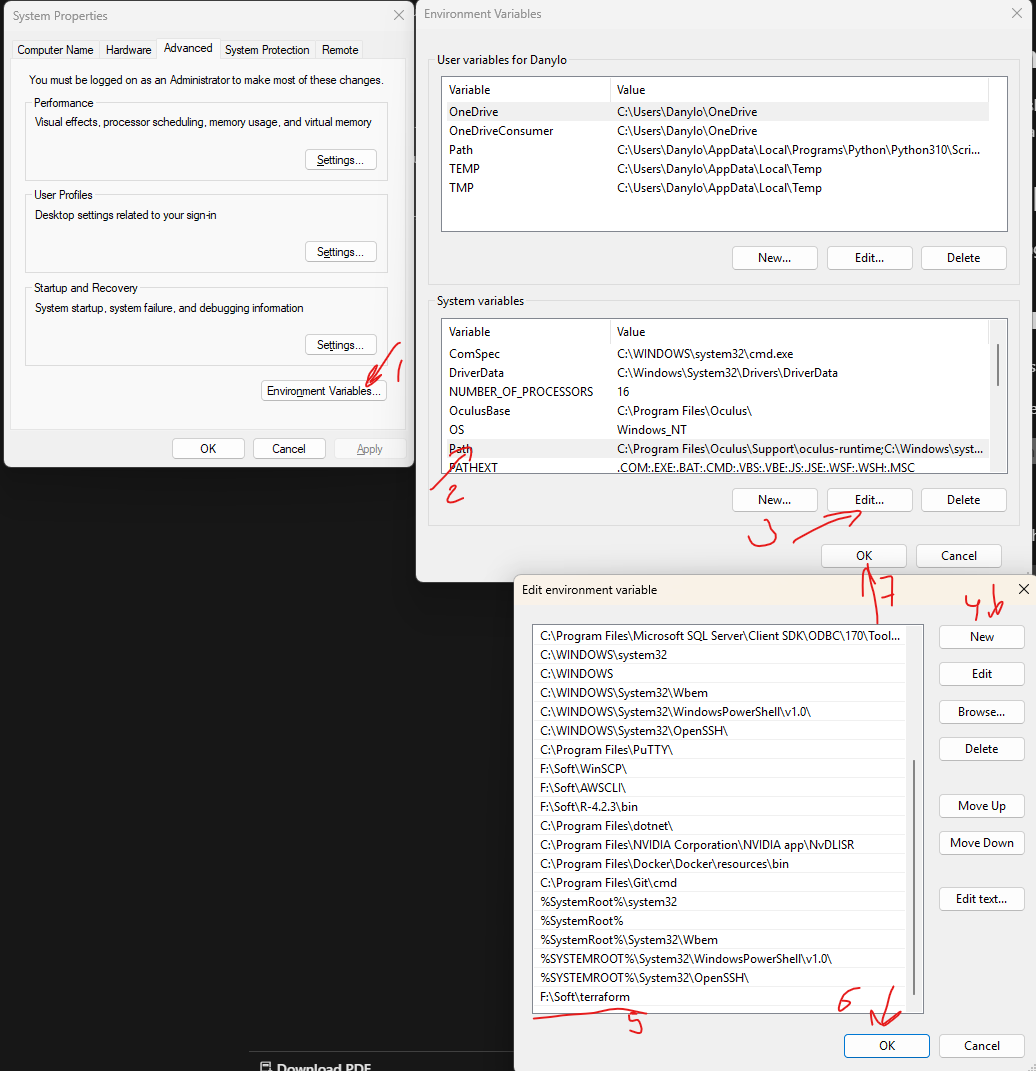


1. Створити main.tf файл, в якому буде описано всю інфраструктуру застосунку (створення інстансу та Security group з минулої л/р). Використовуйте репозиторій з минулої л/р для розміщення в ньому необхідних файлів.
   1. Для виконання дій із Terraform потрібно виконати попередні налаштування:
      1. Встановити Terraform CLI (версія 1.2.0+);
      2. Встановити AWS CLI;
      3. Мати AWS обліковий запис та облікові дані, що дозволяють створювати ресурси (виконано на попередньому кроці).
   2. Завантажимо та встановимо Terraform із офіційного сайту <https://developer.hashicorp.com/terraform/install>:

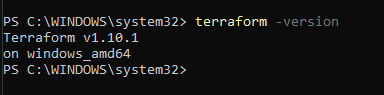




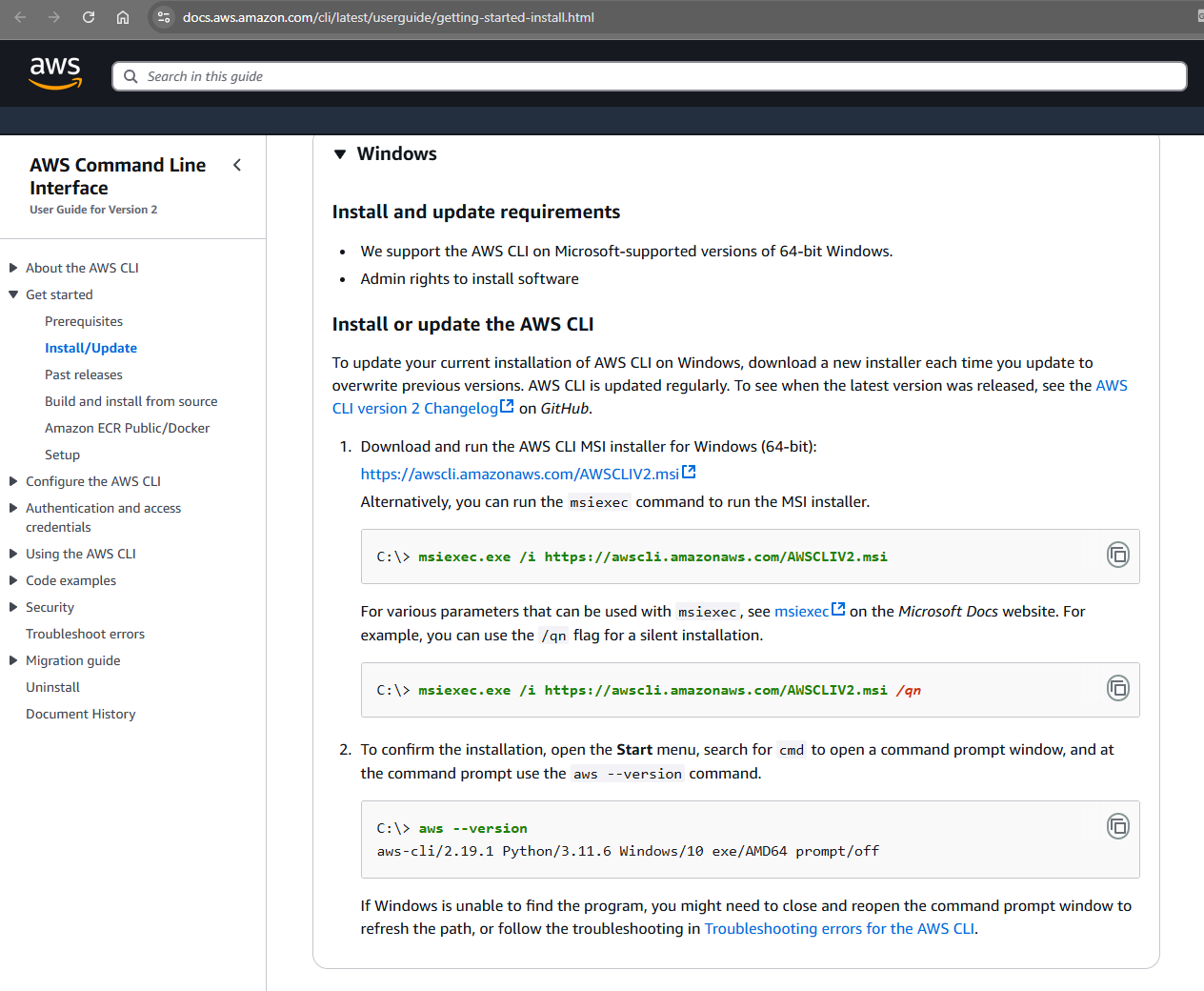
* 1. Оновимо змінну середовища PATH, додавши шлях до каталогу з виконуваним файлом Terraform:

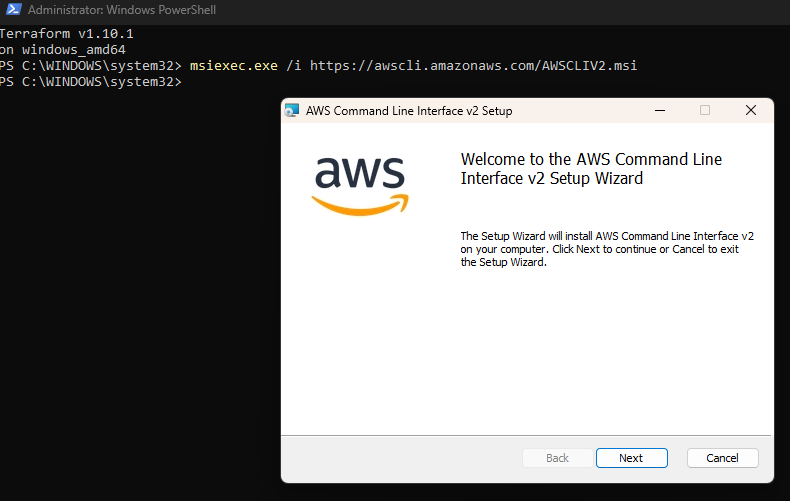


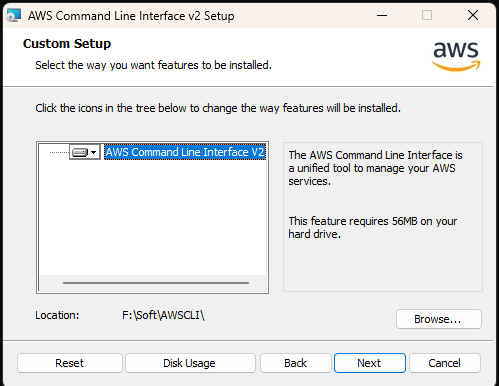
* 1. Перевіримо встановлення Terraform за допомогою PowerShell, виконавши команду «terraform -version» (запустіть новий інстанс PowerShell після змінення змінних середовища):

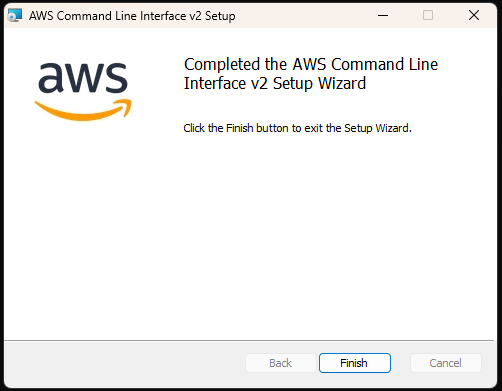


* 1. Перед виконанням Terraform-команд потрібно налаштувати доступ до AWS. Для того щоб взаємодіяти з AWS з локального пристрою встановимо AWS CLI з офіційного сайту <https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>:

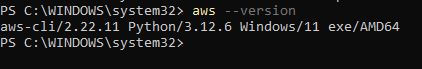




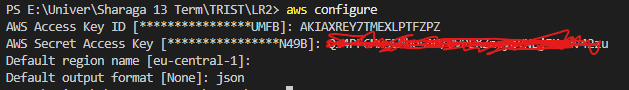




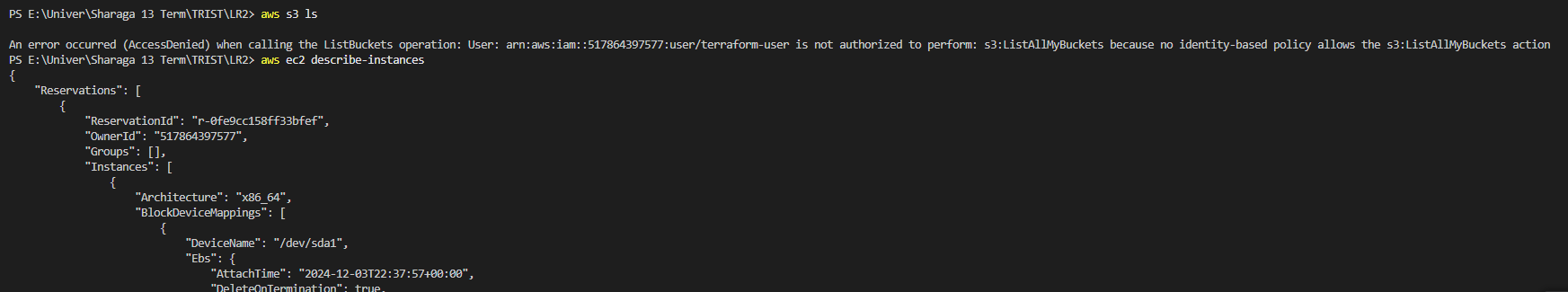
* 1. Перевіримо встановлення AWS CLI за допомогою PowerShell, виконавши команду «aws --version»:



* 1. Налаштуємо доступ до AWS, виконавши команду «aws configure», використовуючи access token з 2го кроку:



* 1. Перевіримо права доступу користувача, щоб переконатися, що доступ до EC2 надано, але доступу до інших сервісів, наприклад, S3, немає:



* 1. Створимо файл main.tf і опишемо інфраструктуру за допомогою конфігурації Terraform:

# Встановлення провайдера AWS.

provider "aws" {

region = "eu-central-1"

}

# Створення Security Group.

resource "aws\_security\_group" "trist\_lab2\_sg" {

name = "trist-lab2-sg"

description = "Allow SSH, HTTPS, and HTTP traffic"

# Вхідний трафік (ingress rules).

ingress {

description = "Allow SSH traffic from your IP"

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = ["212.110.138.79/32"] # Мій IP адрес (якщо вже брати аналогію з минулою ЛР).

}

ingress {

description = "Allow HTTPS traffic from anywhere"

from\_port = 443

to\_port = 443

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"] # HTTPS з будь-якого місця.

}

ingress {

description = "Allow HTTP traffic from anywhere"

from\_port = 80

to\_port = 80

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"] # HTTP з будь-якого місця.

}

# Вихідний трафік (egress rules).

egress {

description = "Allow all outbound traffic"

from\_port = 0

to\_port = 0

protocol = "-1" # Всі протоколи.

cidr\_blocks = ["0.0.0.0/0"] # Вихідний трафік до будь-якого місця.

}

tags = {

Name = "trist-lab2"

}

}

# Створення EC2 інстансу.

resource "aws\_instance" "trist\_lab2\_web" {

ami = "ami-0084a47cc718c111a" # ubuntu-noble-24.04-amd64-server-20240927.

instance\_type = "t2.micro"

key\_name = "keys-lr1"

security\_groups = [aws\_security\_group.trist\_lab2\_sg.name]

# Встановлення Docker, запуск застосунку з ЛР1 + watchtower.

user\_data = <<EOF

#!/bin/bash

sudo apt update

sudo apt install -y docker.io

sudo usermod -aG docker $USER

newgrp docker

docker run -d -p 80:80 --name rist-lr1 joker759/trist-lr1:latest

docker ps -a

docker run -d --name watchtower -v /var/run/docker.sock:/var/run/docker.sock -e WATCHTOWER\_CLEANUP=true -e WATCHTOWER\_POLL\_INTERVAL=120 containrrr/watchtower

docker ps -a

docker logs watchtower

EOF

}

tags = {

Name = "trist-lab2"

}

# Вихідні дані: Public IP

output "instance\_ip" {

description = "Public IP of EC2 instance"

value = aws\_instance.trist\_lab2\_web.public\_ip

}

# Вихідні дані: Application URL

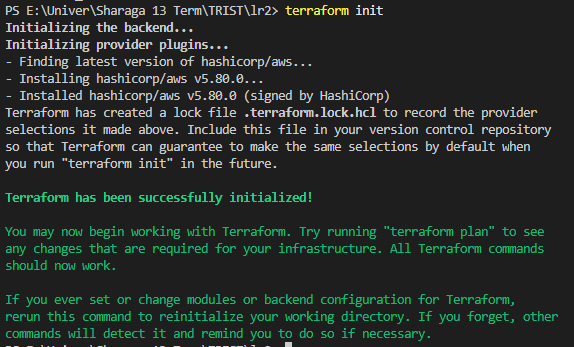
output "nginx\_url" {

description = "Address of the deployed application"

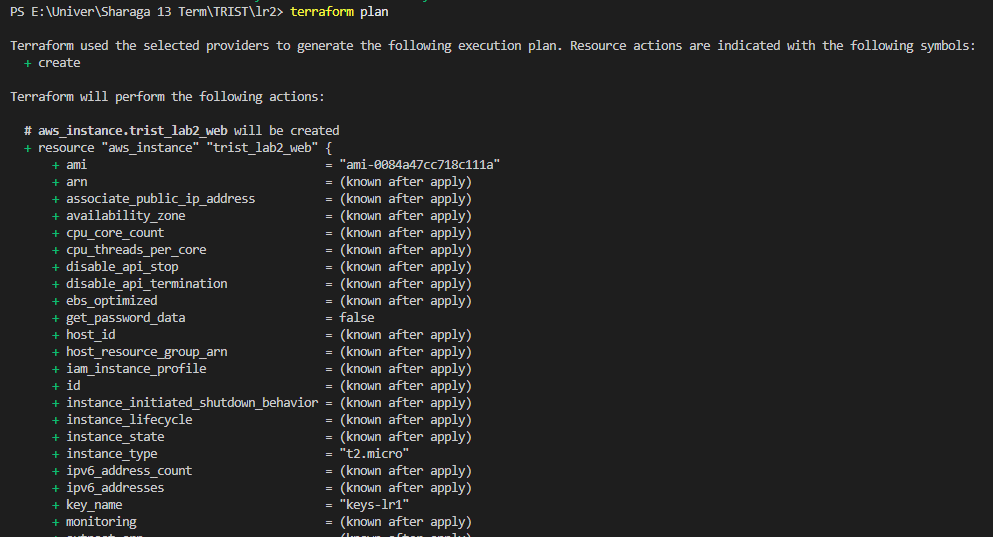
value = "http://${aws\_instance.trist\_lab2\_web.public\_ip}"

}

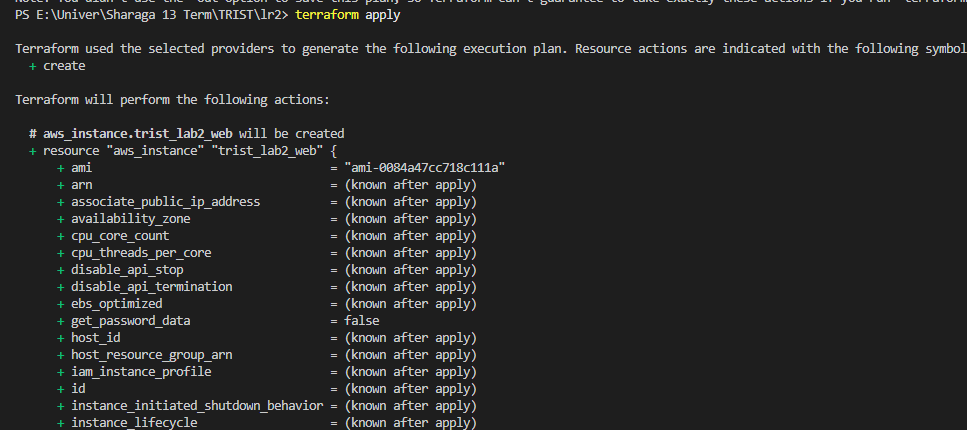
* 1. Виконаємо ініціалізацію Terraform плагінів за допомогою команди «terraform init»:



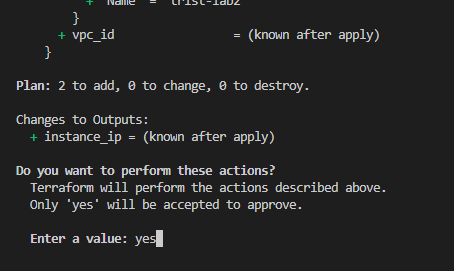
* 1. Переглянемо план розгортання за допомогою команди «terraform plan»:

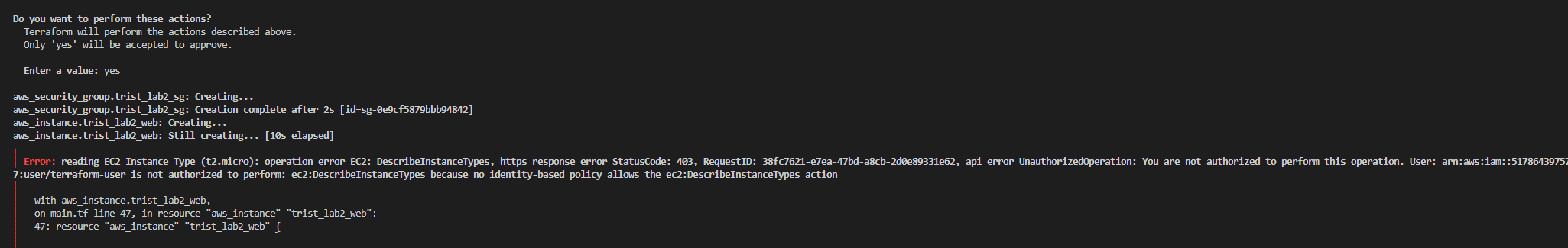


* 1. Застосуємо конфігурацію за допомогою команди «terraform apply»:

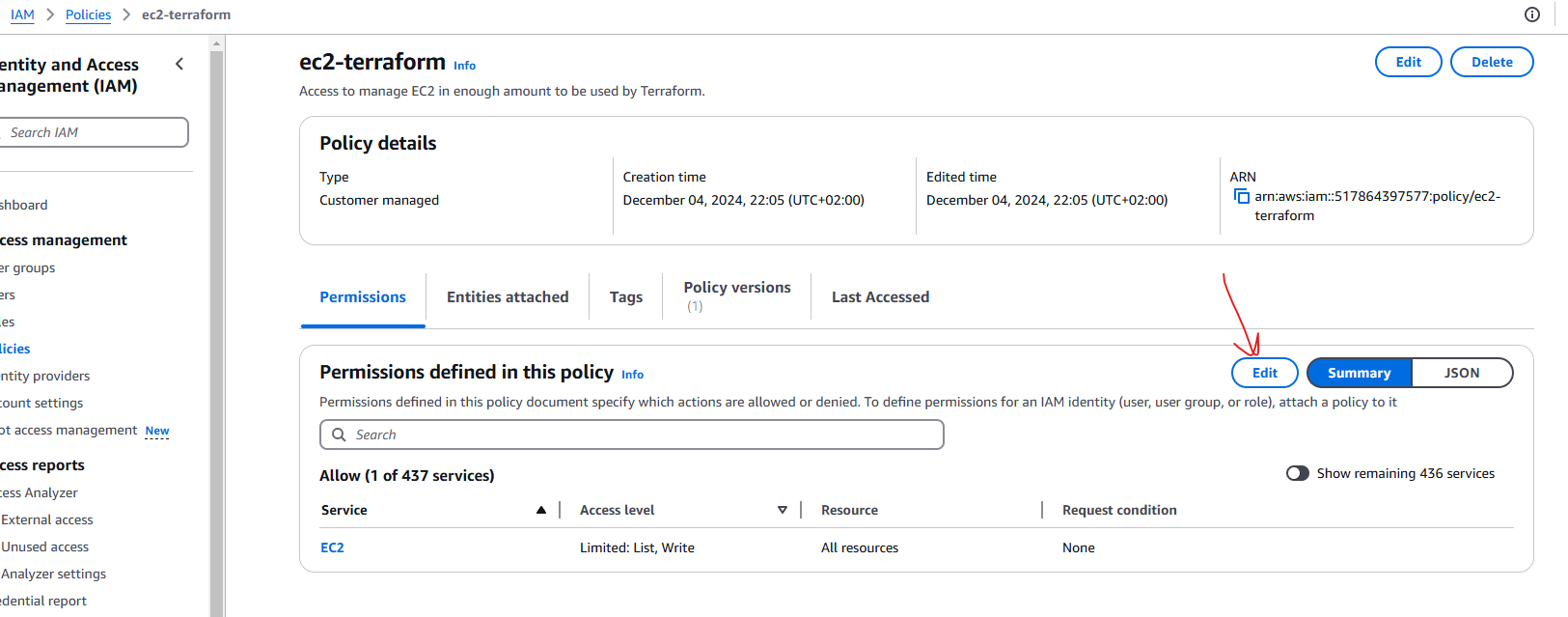


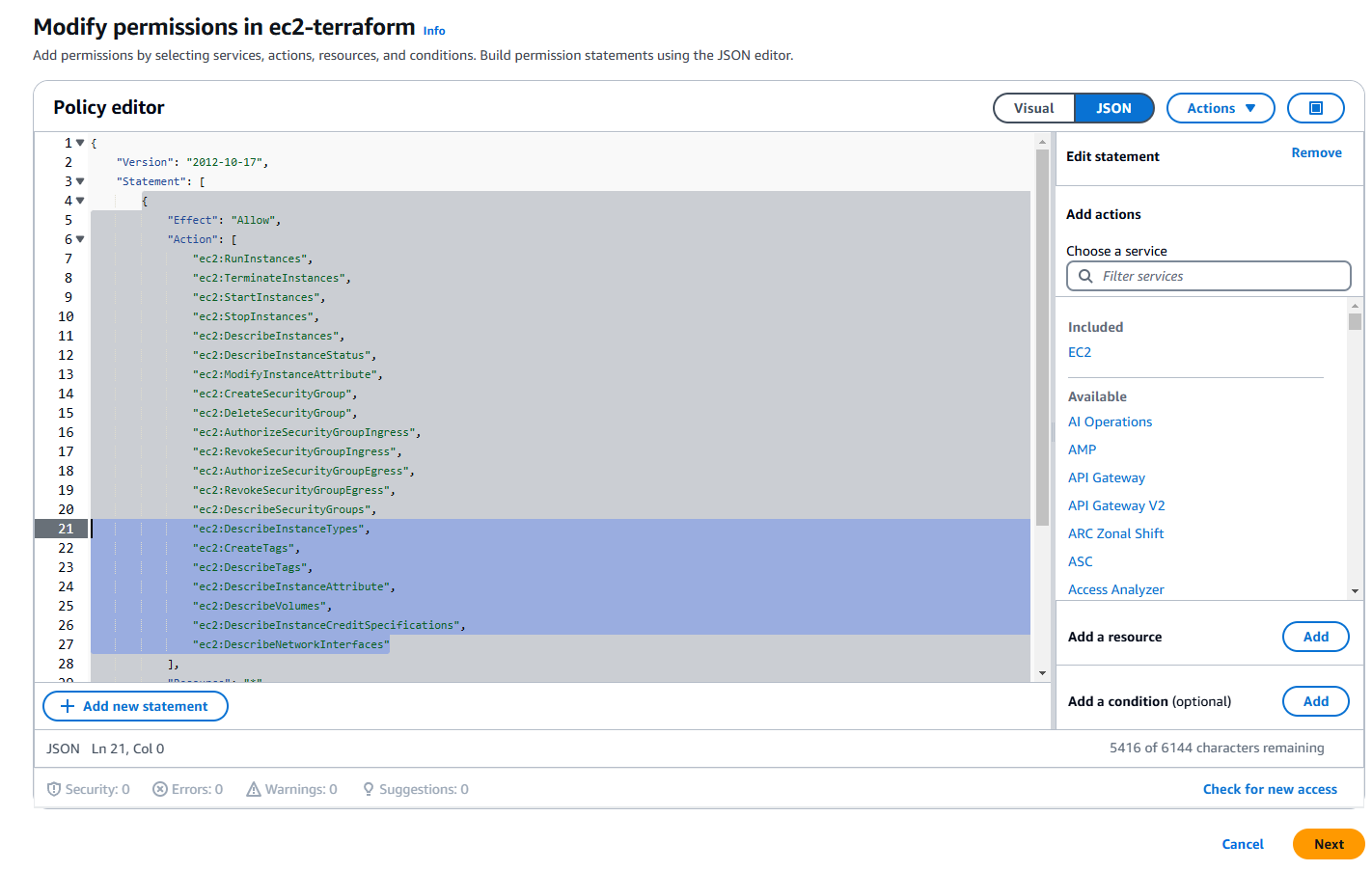
…

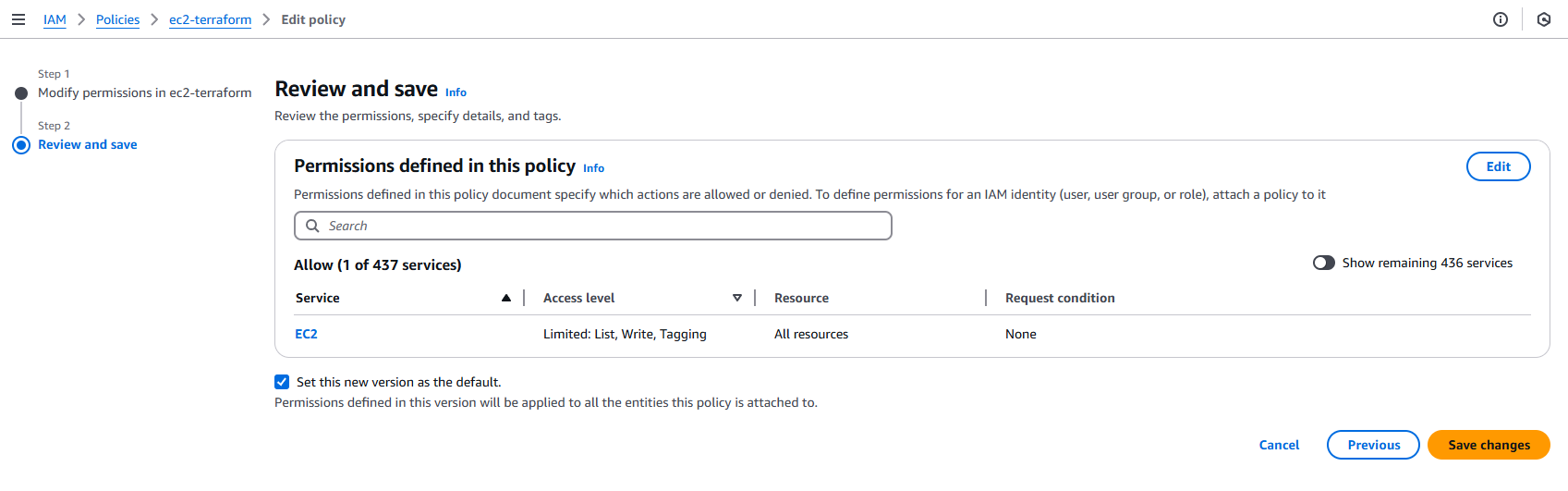


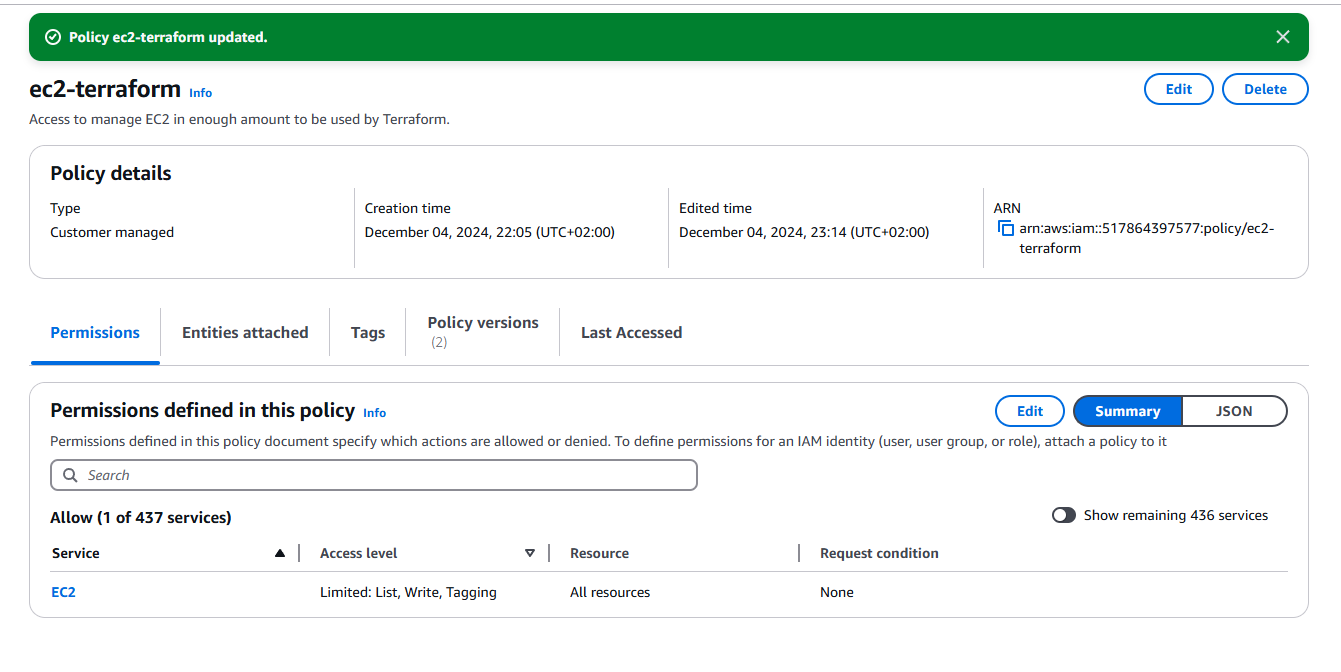


* 1. Модифікуємо створену раніше політику ec2-terraform-policy, додавши необхідні права:
     1. ec2:DescribeInstanceTypes;
     2. ec2:CreateTags;
     3. ec2:DescribeTags;
     4. ec2:DescribeInstanceAttribute;
     5. ec2:DescribeVolumes;
     6. ec2:DescribeInstanceCreditSpecifications;
     7. ec2:DescribeNetworkInterfaces.

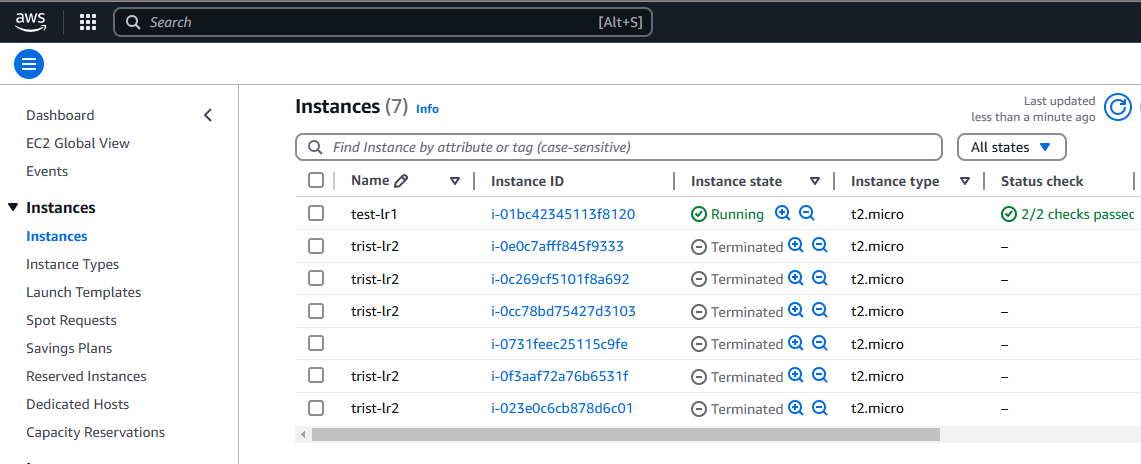


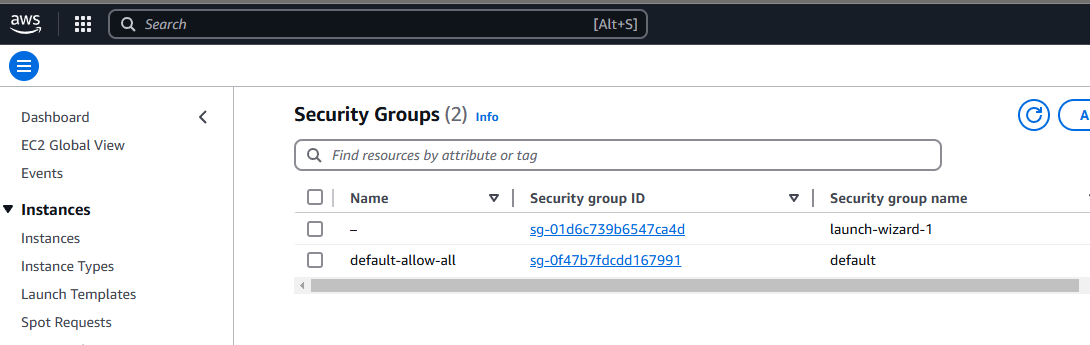


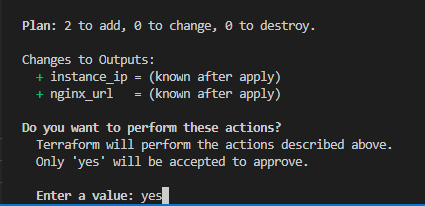


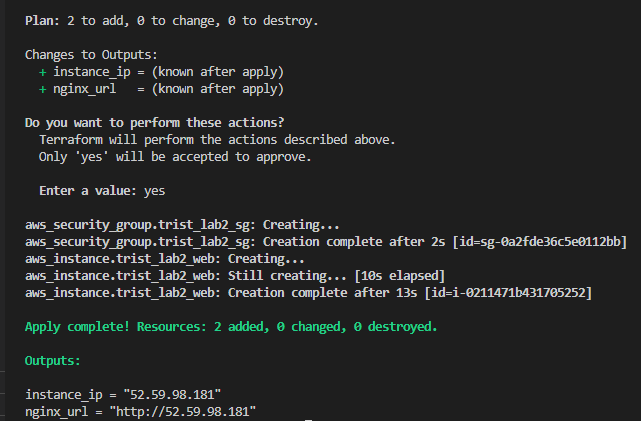


* 1. Повторно застосуємо конфігурацію за допомогою команди «terraform apply»:

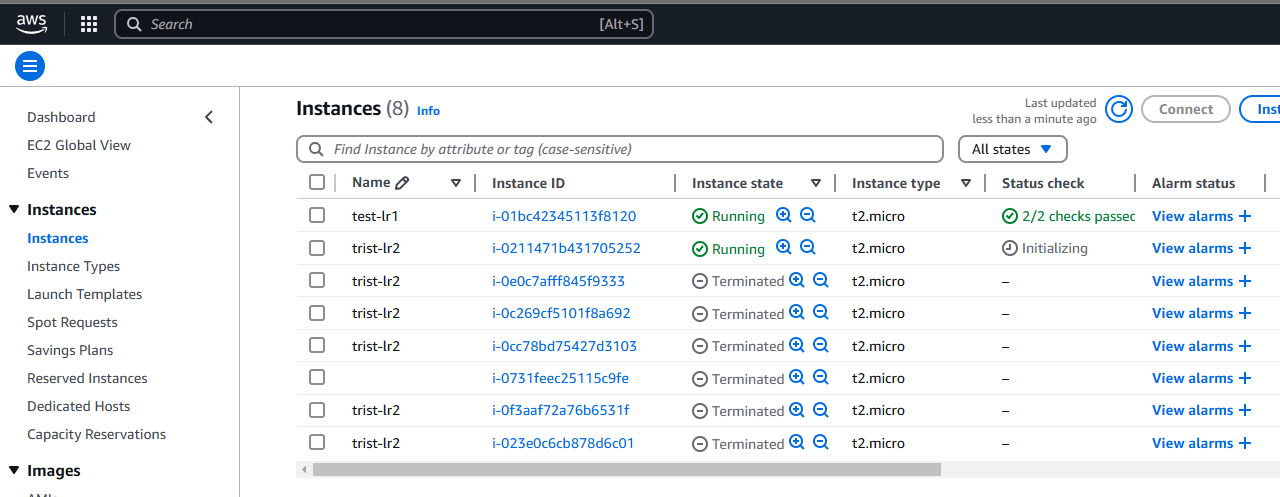


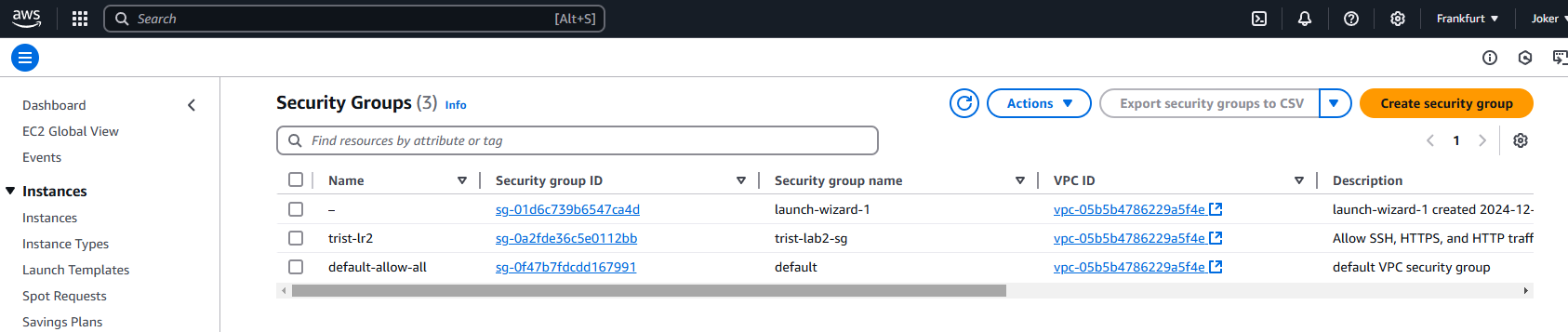


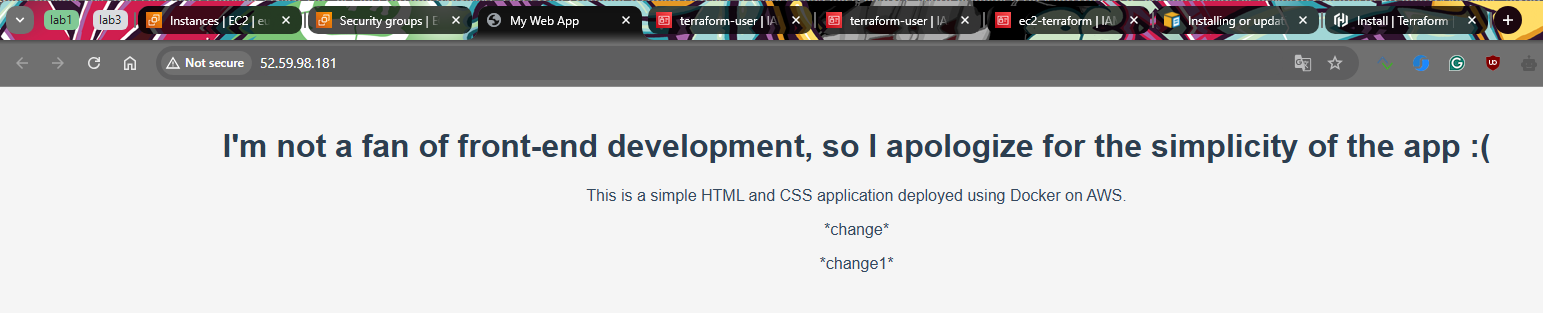




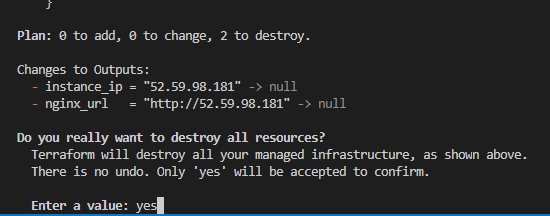
* 1. Перевіримо результат розгортання, переконавшись, що ресурси створено коректно:

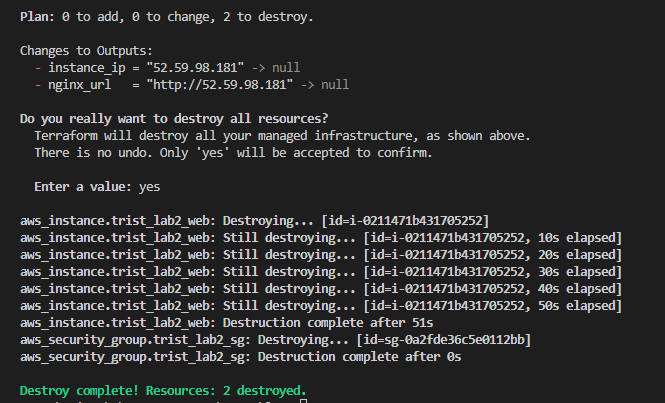




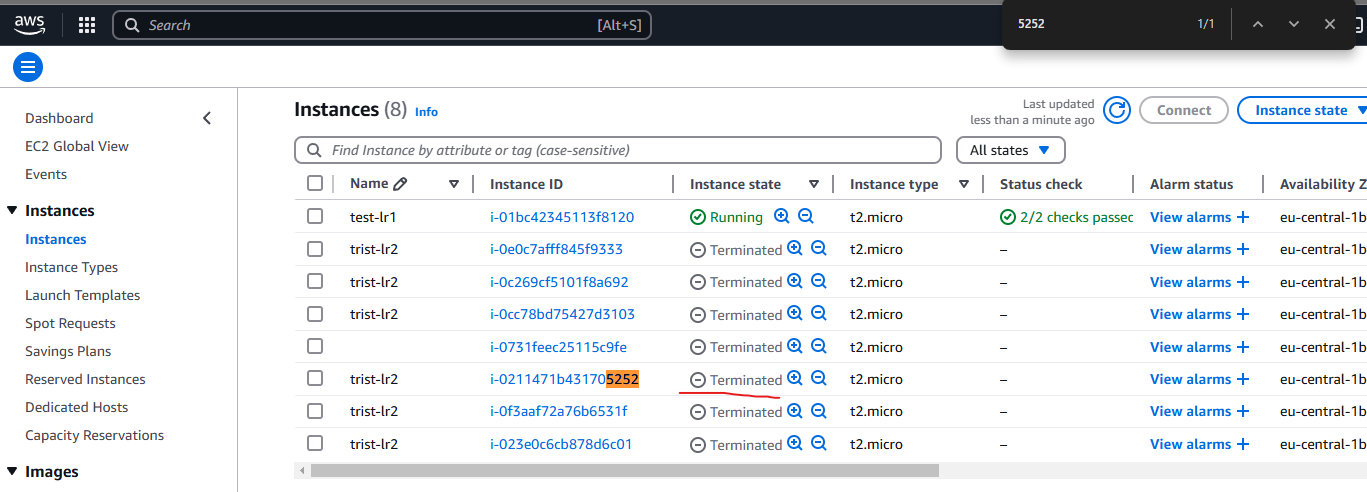


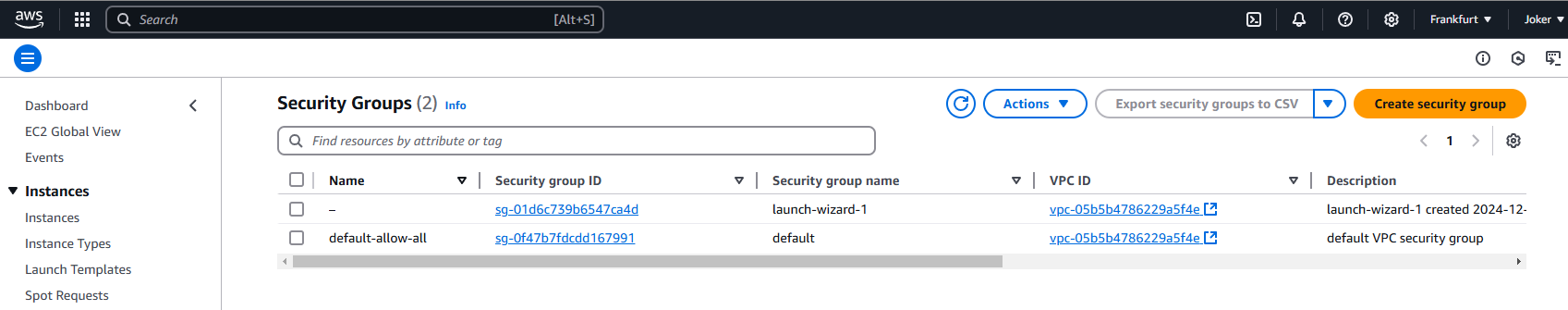
* 1. Видалимо створені ресурси за допомогою команди «terraform destroy»:

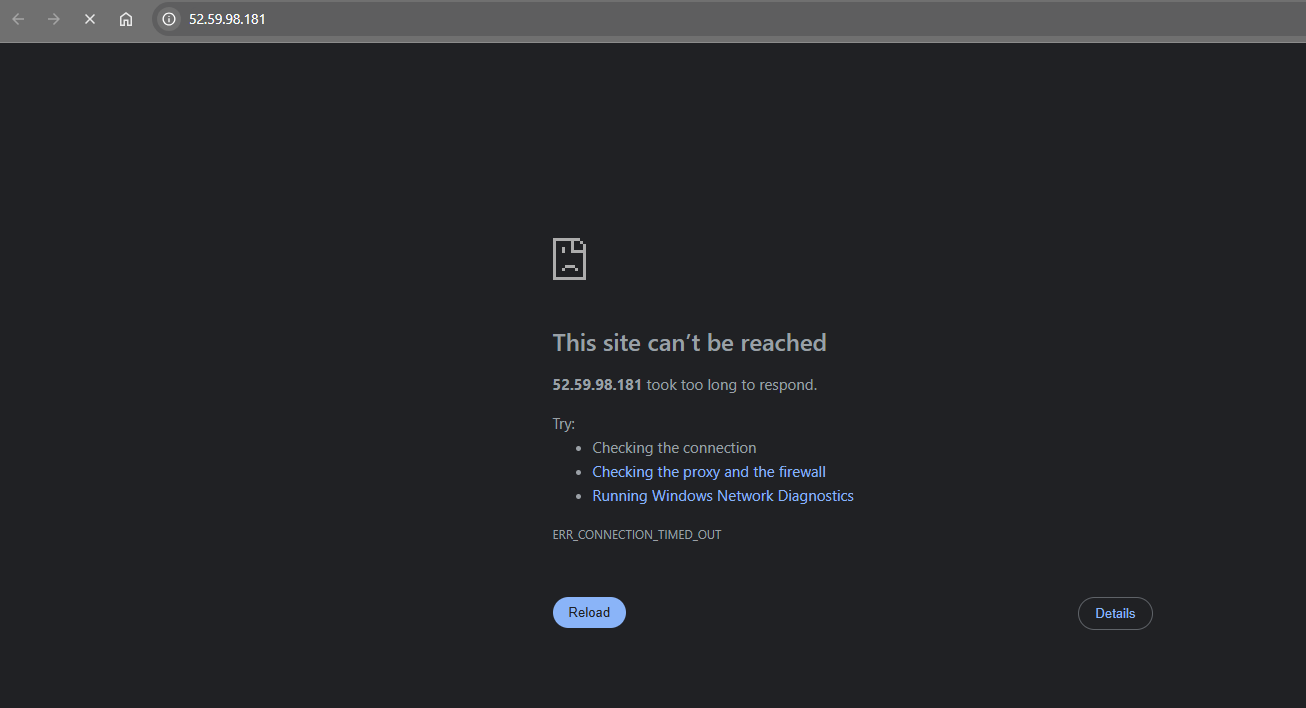




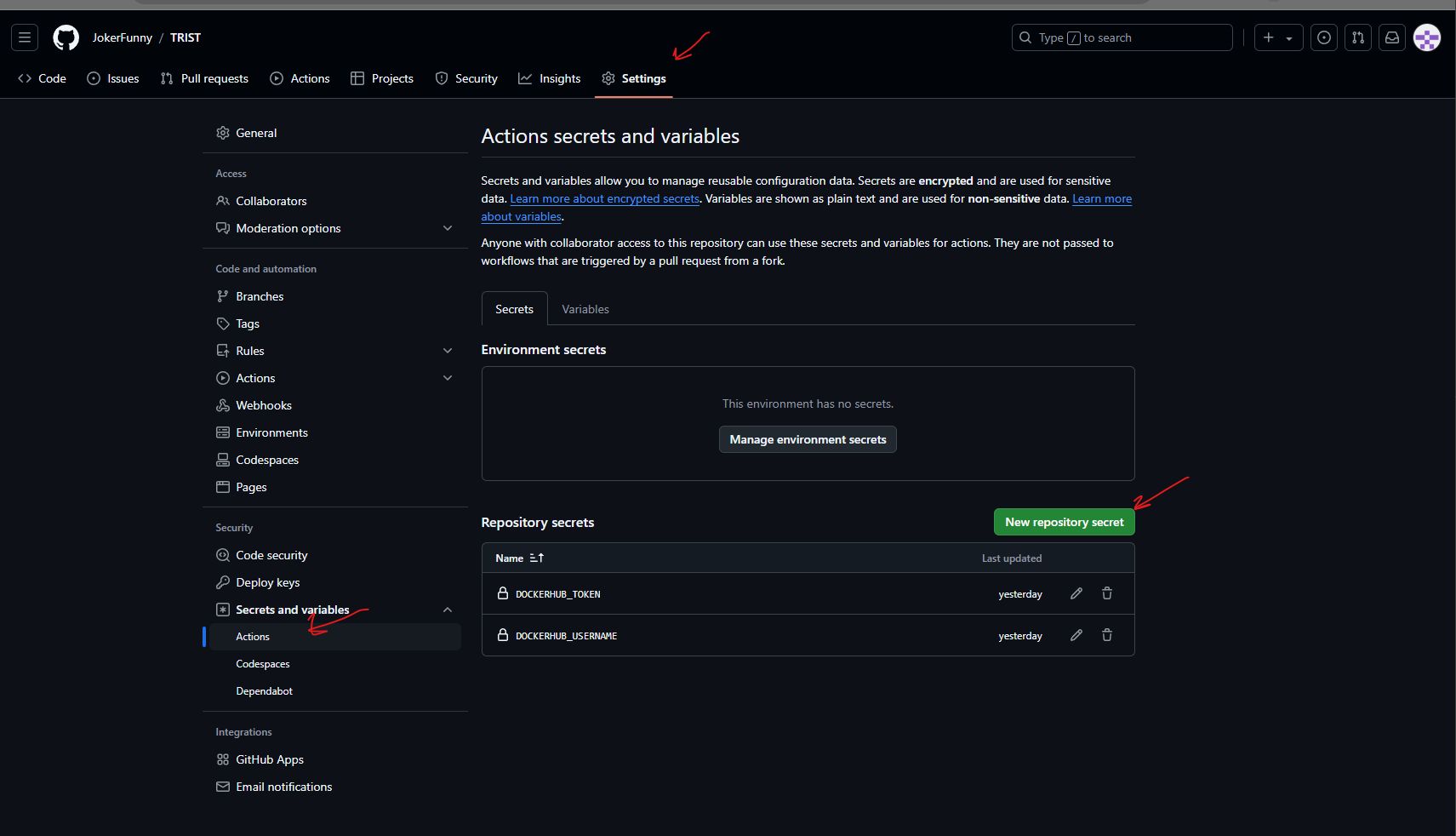
* 1. Перевіримо результат знищення, щоб переконатися, що всі ресурси успішно видалено:



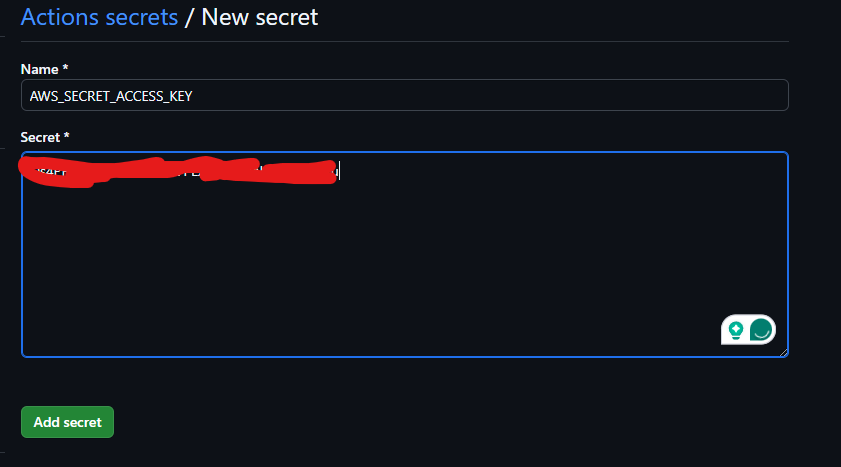


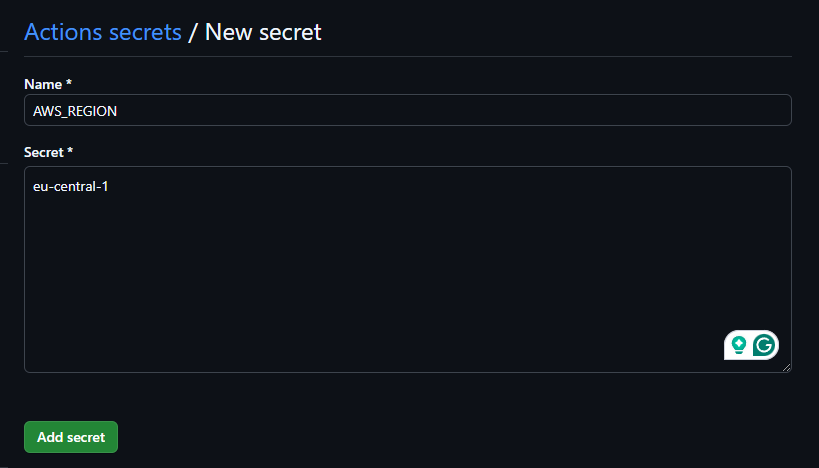


1. (Додатково) Створити GitHub Actions pipeline, який буде виконувати terraform init, plan, apply замість ручного виконання. Передбачити коректне виконання джобів у різних станах інфраструктури:
2. Додамо токен із другого кроку в GitHub Secrets для подальшого використання у GitHub Actions:
   * 1. AWS\_ACCESS\_KEY\_ID;
     2. AWS\_SECRET\_ACCESS\_KEY;
     3. AWS\_REGION.



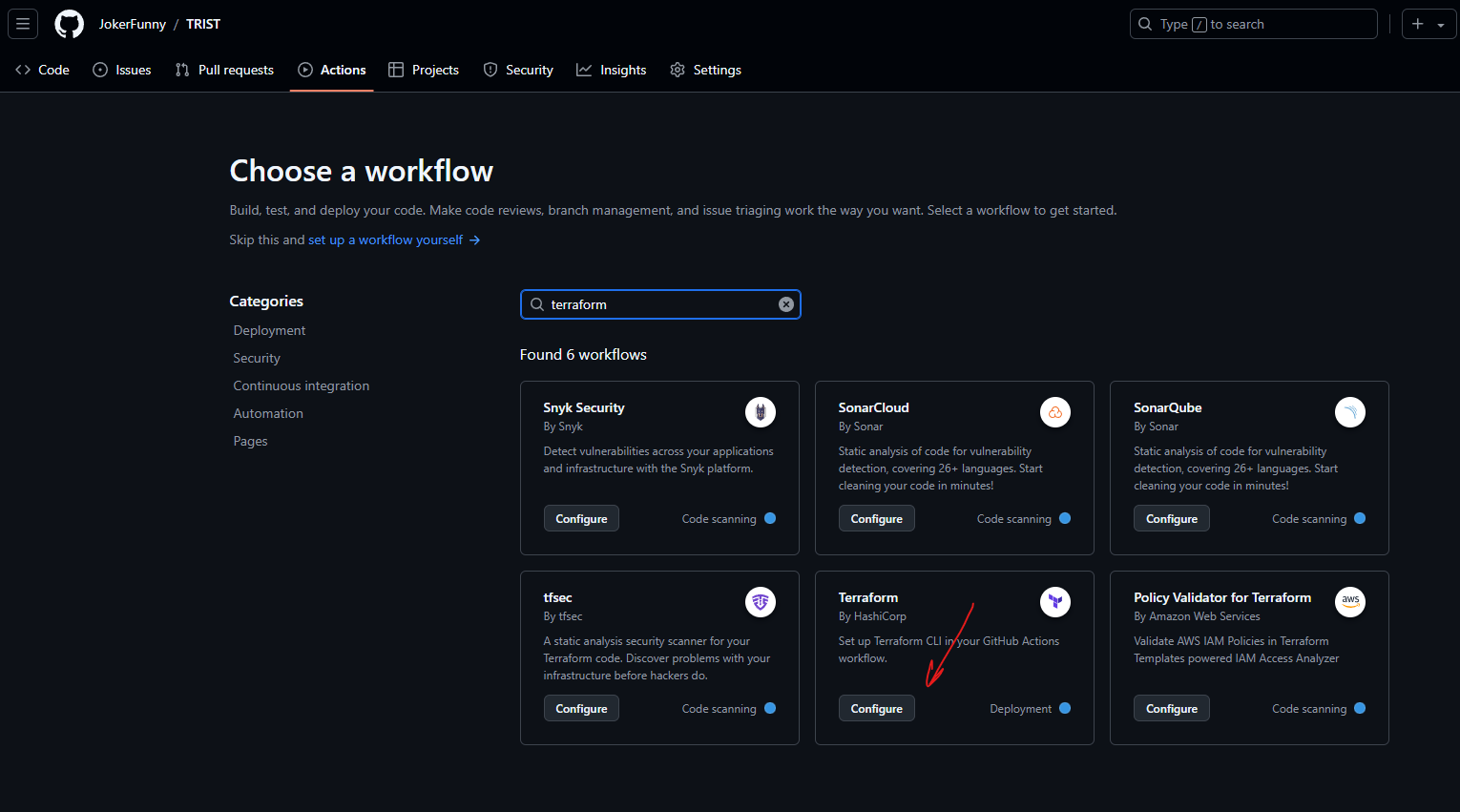


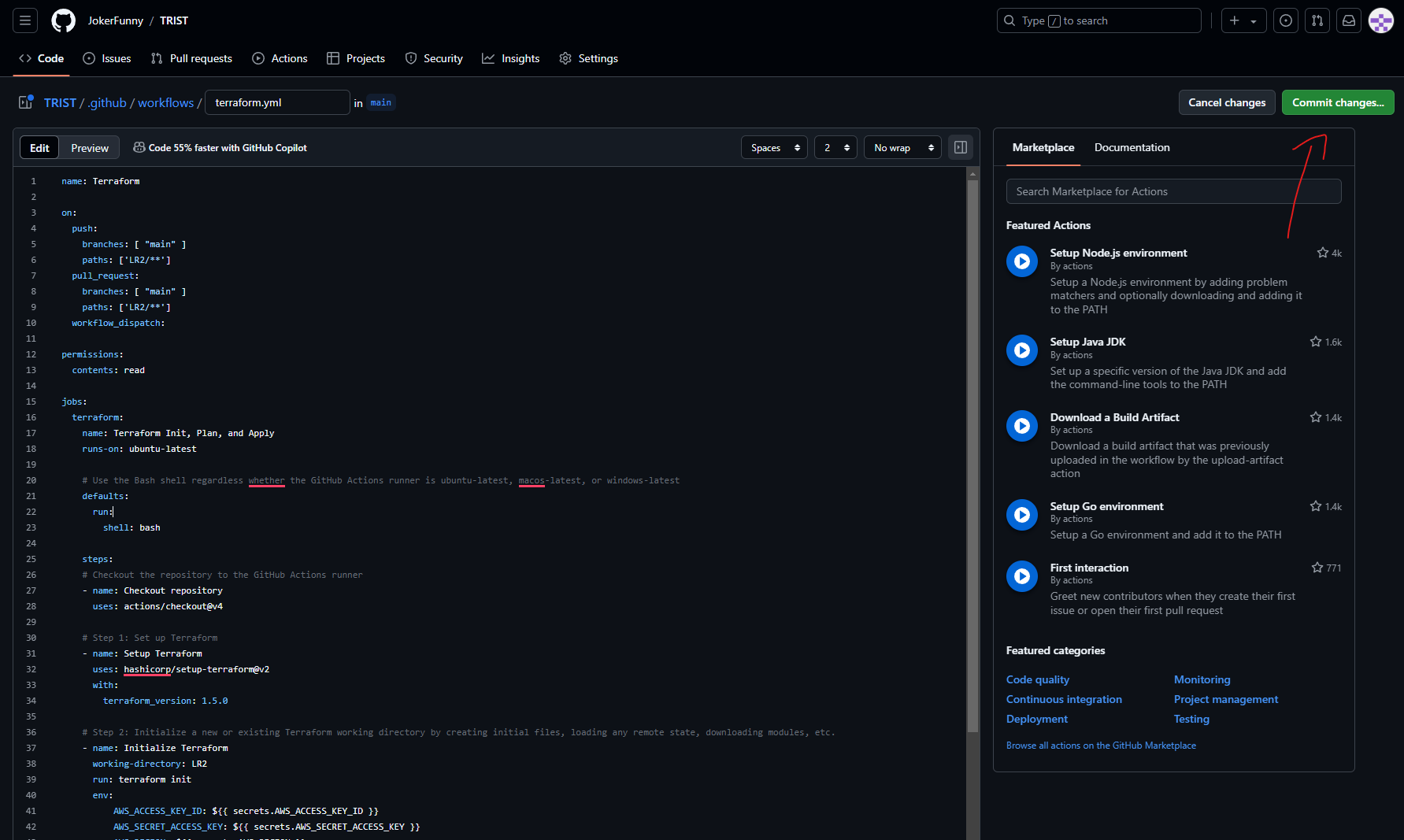






1. Створимо GitHub Actions для автоматизації процесу деплою за допомогою Terraform. Workflow має включати такі етапи:
   * 1. Ініціалізація Terraform (terraform init);
     2. Перегляд плану (terraform plan);
     3. Застосування конфігурації (terraform apply);
     4. Вивід інформації про застосунок (адресу).





terraform.yml:

name: Terraform

on:

push:

branches: [ "main" ]

paths: ['LR2/\*\*']

pull\_request:

branches: [ "main" ]

paths: ['LR2/\*\*']

workflow\_dispatch:

permissions:

contents: read

jobs:

terraform:

name: Terraform Init, Plan, and Apply

runs-on: ubuntu-latest

# Use the Bash shell regardless whether the GitHub Actions runner is ubuntu-latest, macos-latest, or windows-latest

defaults:

run:

shell: bash

steps:

# Checkout the repository to the GitHub Actions runner

- name: Checkout repository

uses: actions/checkout@v4

# Step 1: Set up Terraform

- name: Setup Terraform

uses: hashicorp/setup-terraform@v2

with:

terraform\_version: 1.5.0

# Step 2: Initialize a new or existing Terraform working directory by creating initial files, loading any remote state, downloading modules, etc.

- name: Initialize Terraform

working-directory: LR2

run: terraform init

env:

AWS\_ACCESS\_KEY\_ID: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

AWS\_SECRET\_ACCESS\_KEY: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

AWS\_REGION: ${{ secrets.AWS\_REGION }}

# Step 3: Generates an execution plan for Terraform

- name: Plan Terraform

working-directory: LR2

run: terraform plan

# Step 4: Apply Terraform

- name: Apply Terraform

working-directory: LR2

run: terraform apply -auto-approve

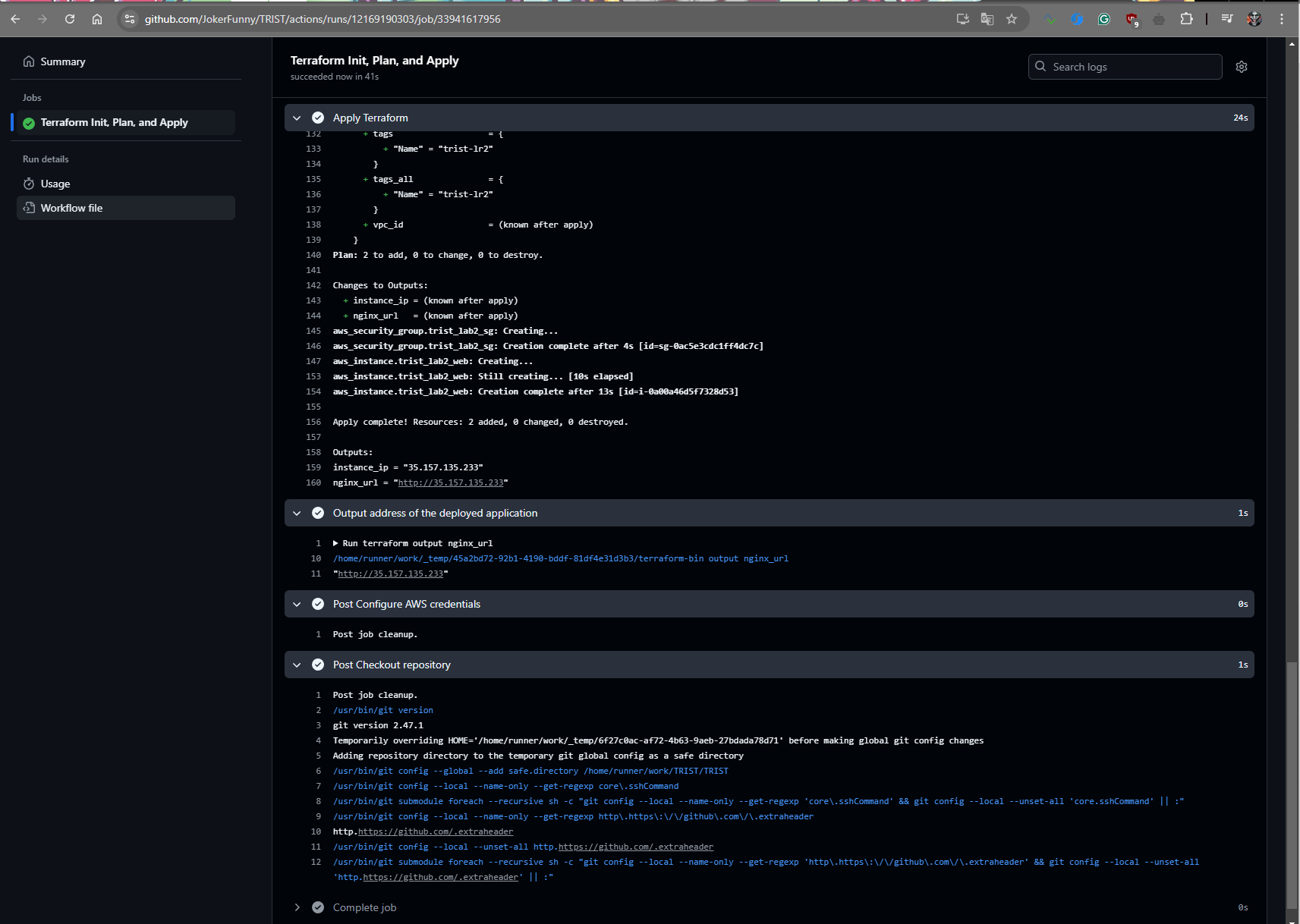
# Step 5: Output Instance IP

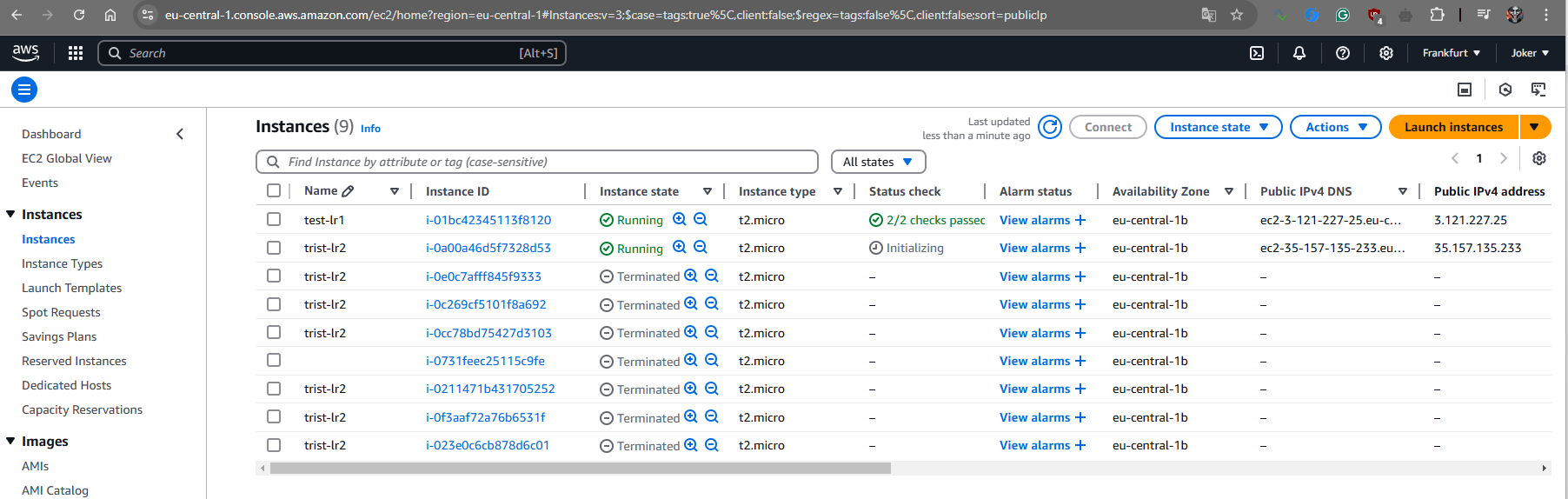
- name: Output address of the deployed application

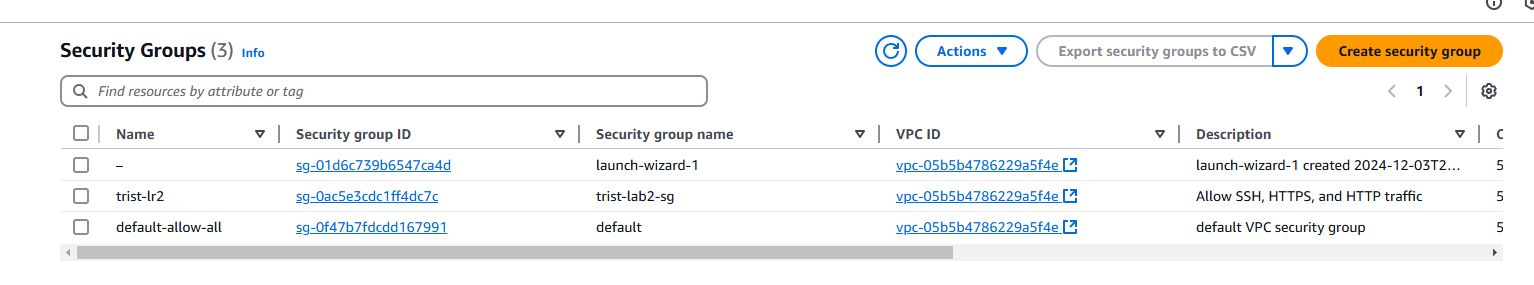
working-directory: LR2

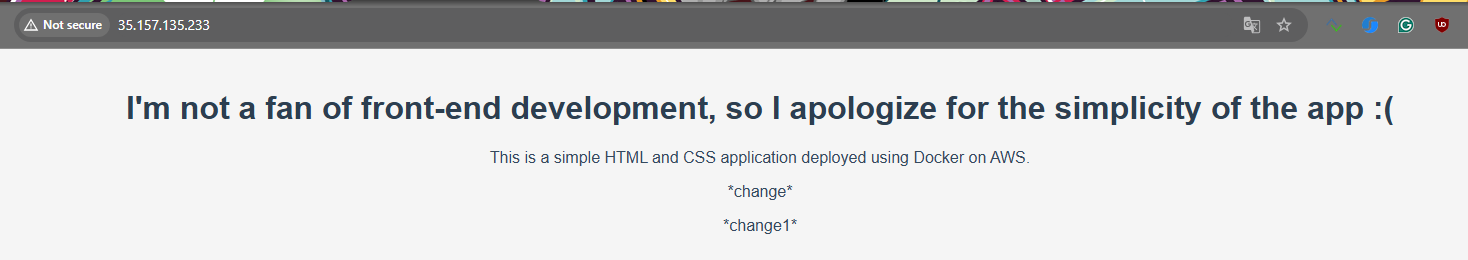
run: terraform output nginx\_url

1. Перевіримо виконання пайплайну, переконавшись, що ресурси створені успішно:

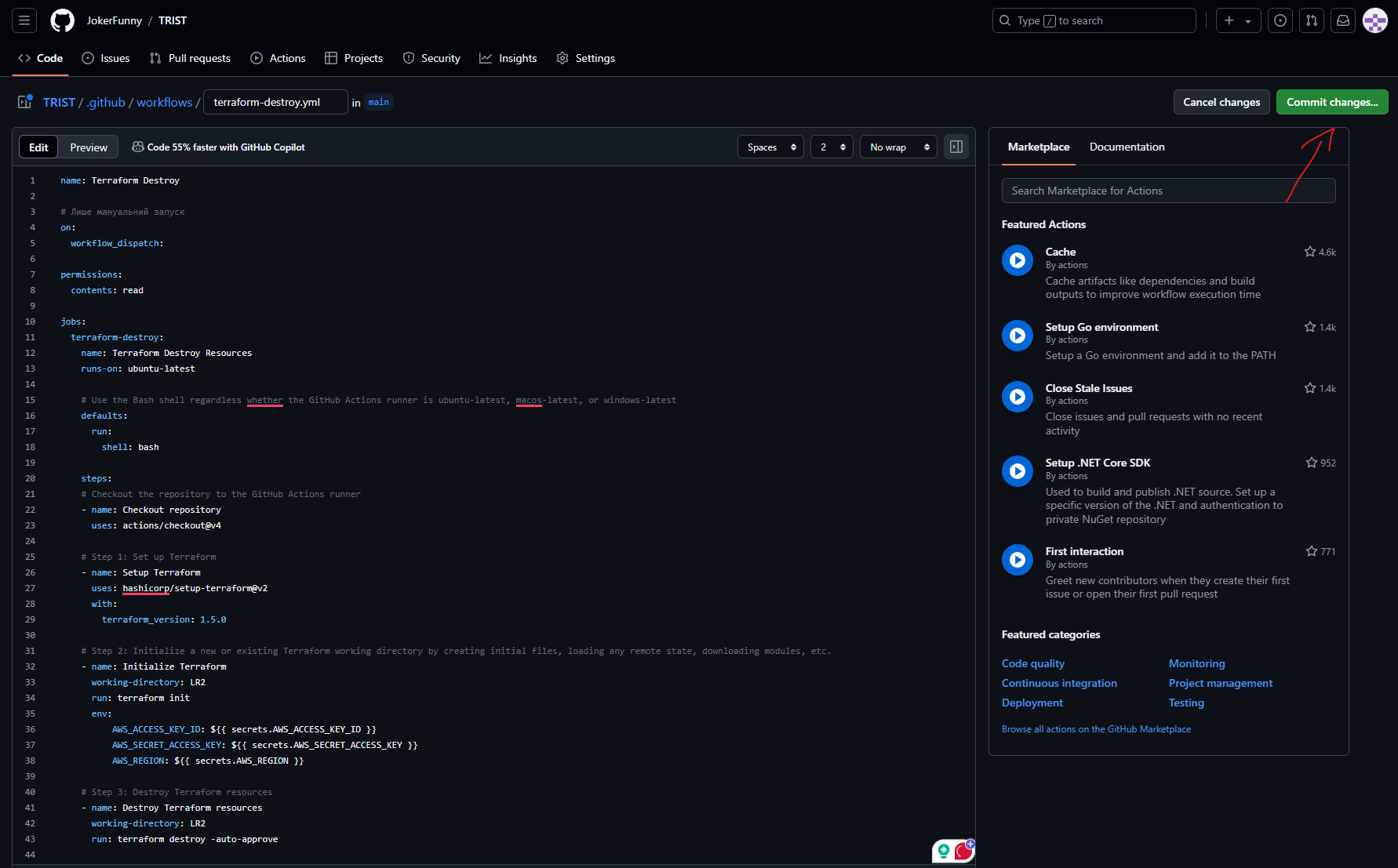








1. Створимо окремий GitHub Actions workflow для автоматизації процесу знищення інфраструктури за допомогою Terraform. Workflow налаштовується на manual trigger (ручний запуск). Має включати наступні етапи:
   * 1. Ініціалізацію Terraform (terraform init);
     2. Знищення інфраструктури (terraform destroy).



terraform-destroy.yml:

name: Terraform Destroy

# Лише мануальний запуск

on:

workflow\_dispatch:

permissions:

contents: read

jobs:

terraform-destroy:

name: Terraform Destroy Resources

runs-on: ubuntu-latest

# Use the Bash shell regardless whether the GitHub Actions runner is ubuntu-latest, macos-latest, or windows-latest

defaults:

run:

shell: bash

steps:

# Checkout the repository to the GitHub Actions runner

- name: Checkout repository

uses: actions/checkout@v4

# Step 1: Set up Terraform

- name: Setup Terraform

uses: hashicorp/setup-terraform@v2

with:

terraform\_version: 1.5.0

# Step 2: Initialize a new or existing Terraform working directory by creating initial files, loading any remote state, downloading modules, etc.

- name: Initialize Terraform

working-directory: LR2

run: terraform init

env:

AWS\_ACCESS\_KEY\_ID: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

AWS\_SECRET\_ACCESS\_KEY: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

AWS\_REGION: ${{ secrets.AWS\_REGION }}

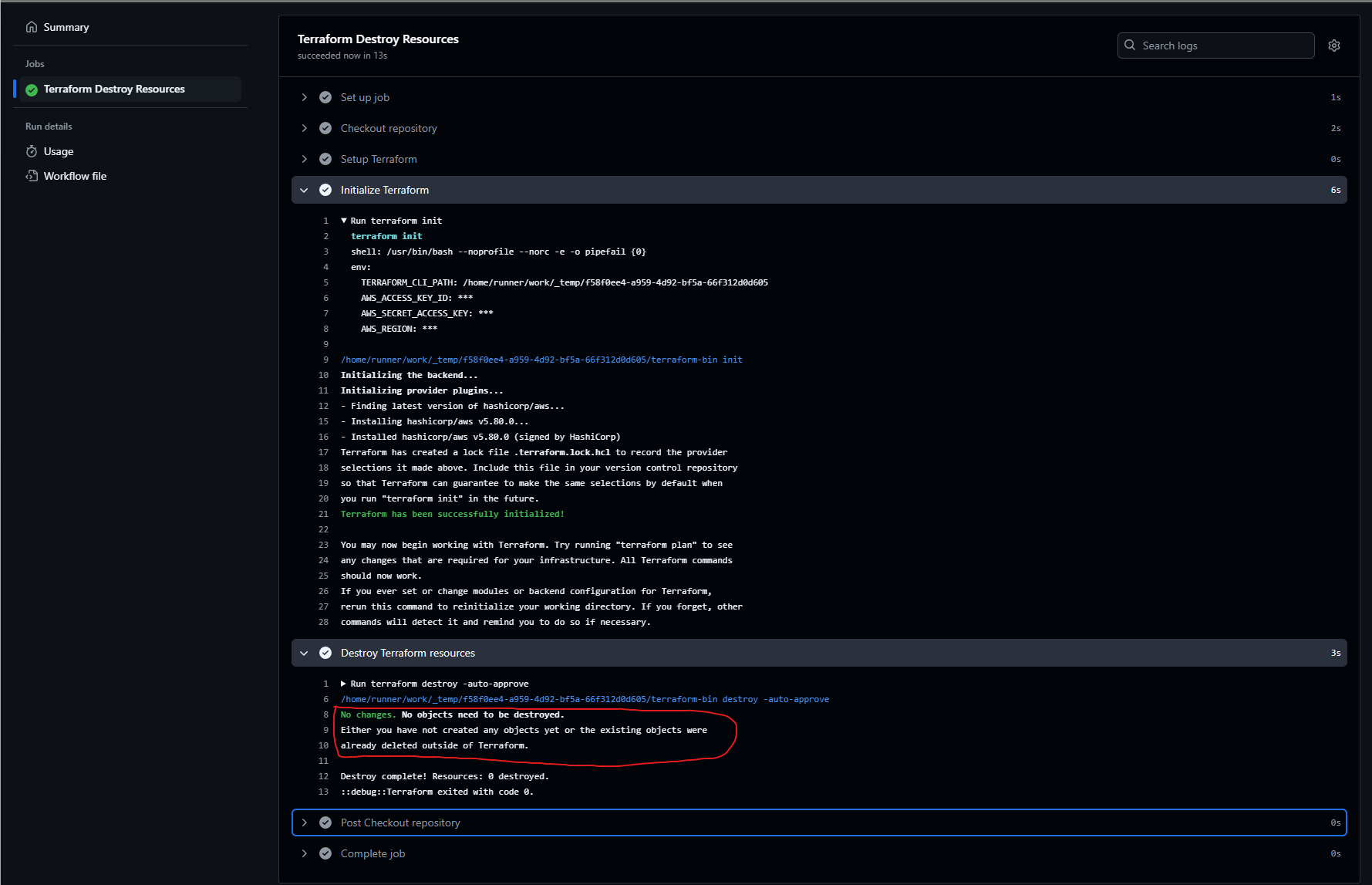
# Step 3: Destroy Terraform resources

- name: Destroy Terraform resources

working-directory: LR2

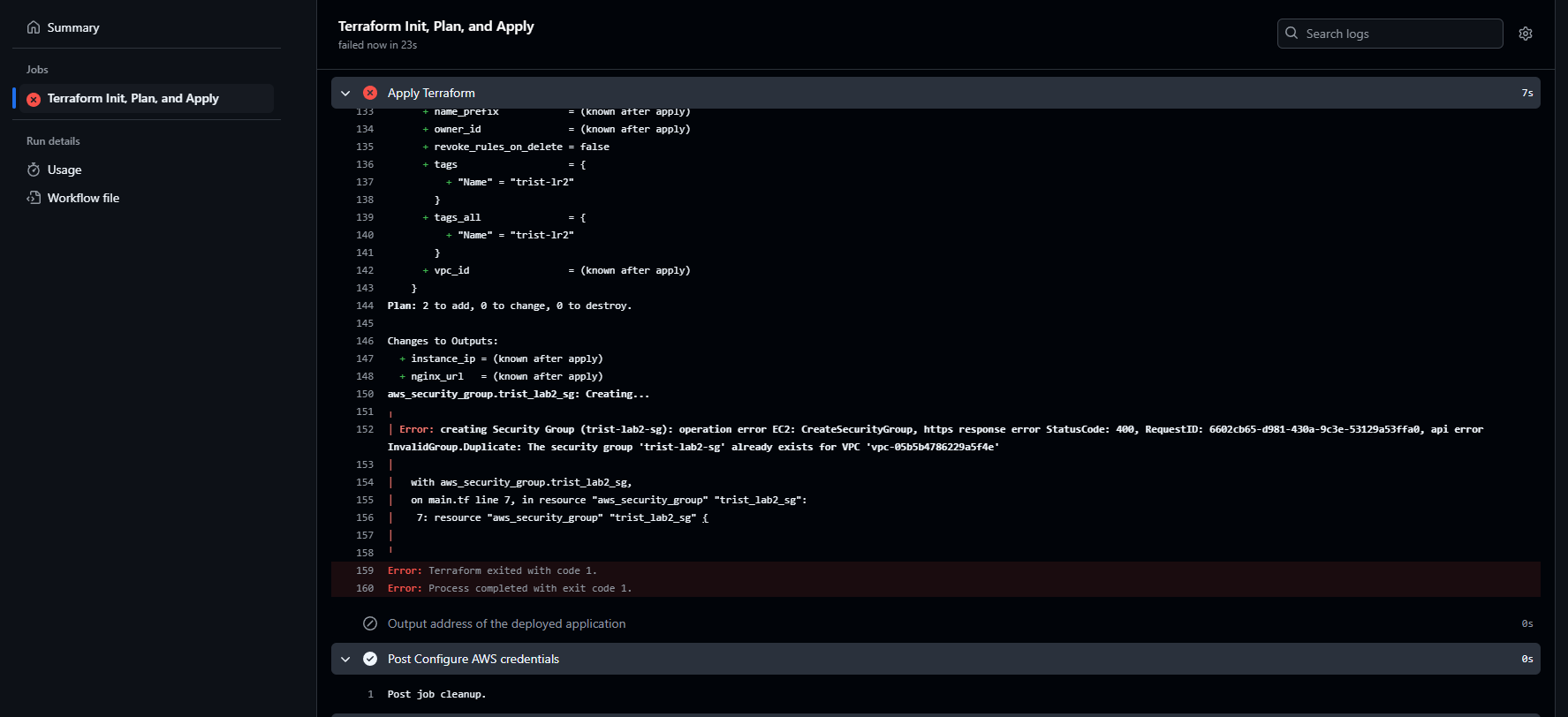
run: terraform destroy -auto-approve

1. Перевіримо виконання пайплайну знищення інфраструктури:



Нажаль, при виконанні Workflow для знищення інфраструктури (наприклад, terraform-destroy.yml), ми отримаємо повідомлення про те, що немає інфраструктури для знищення. Це відбувається через те, що Workflow не бачить інфраструктури, яка була створена раніше в іншому Workflow (наприклад, terraform.yml).

Так само, повторне виконання Workflow terraform.yml для створення інфраструктури призведе до помилки – Terraform спробує повторно створити вже існуючі ресурси:



Це відбувається через те, що Terraform state (файл terraform.tfstate) був створений під час виконання першого пайплайну, але не збережений після його завершення. Це унеможливлює коректну взаємодію Terraform із вже створеними ресурсами (Terraform залежить від state-файлу для відстеження існуючої інфраструктури). Без цього файлу Terraform вважає, що інфраструктури немає, і не може виконати terraform destroy/повторно застосувати terraform apply (який без наявних змін в існуючій інфраструктурі має нічого не робити).

Для коректного виконання всіх workflow, які взаємодіють із інфраструктурою, необхідно зберігати та синхронізувати файл terraform.tfstate. Це може бути виконано багатьма варіантами, наприклад використовуючи інфраструктуру AWS:

* Використовувати S3 bucket для зберігання Terraform state;
* Використовувати DynamoDB для блокування стану, щоб уникнути конфліктів при одночасному доступі;
* Додати налаштування backend у Terraform;
* Включити синхронізацію state-файлу у всіх Terraform workflow.

*Висновки:* в результаті виконання цієї лабораторної роботи було ознайомлено з базовими концепціями автоматизації розгортання інфраструктури за допомогою Terraform і GitHub Actions.

На основі отриманих знань було реалізовано практичну частину, яка полягала у підготовці інфраструктури з минулої лабораторної роботи засобами Terraform та створенні пайплайнів для автоматизації процесів створення та знищення інфраструктури в AWS EC2.

Було розглянуто важливість використання віддаленого Terraform state для синхронізації між Workflow та забезпечення узгодженості інфраструктури.

Вихідний код та конфігурації можна знайти за наступним посиланням на GitHub:

* [main.tf](https://github.com/JokerFunny/TRIST/tree/main/LR2);
* [Github actions](https://github.com/JokerFunny/TRIST/tree/main/.github/workflows).