

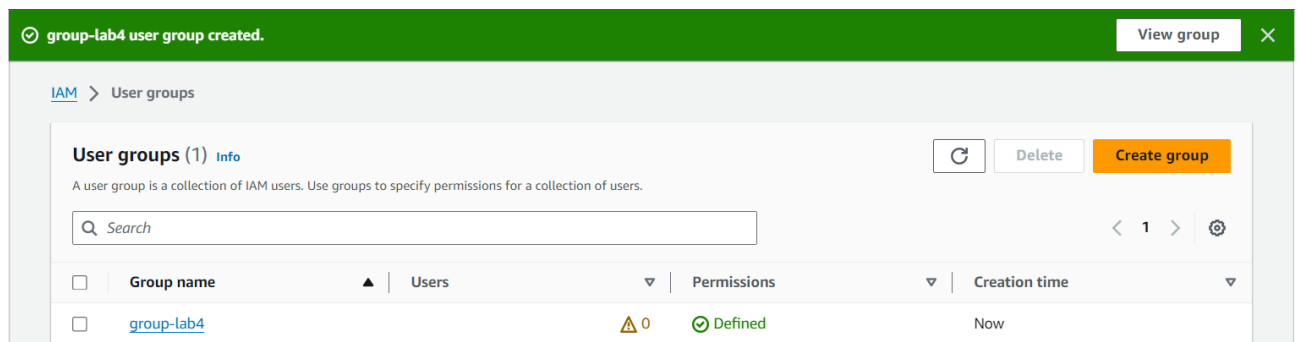
Звіт
З лабораторної роботи №4 та практичної роботи №3
Студента групи МІТ-31
Добровольського Арсенія Михайловича

Тема роботи: Інфраструктура як код (IaC)

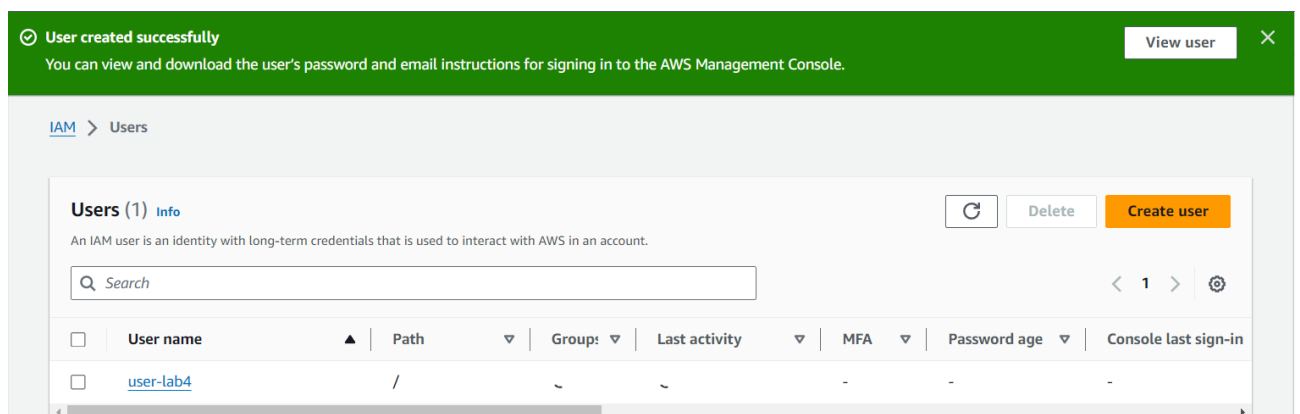
Мета роботи: розглянути поняття та принципи організації інфраструктури як коду; навчитися використовувати Terraform для розгортання інфраструктури.

Завдання 1 (створення базового робочого процесу із використанням підходу Infrastructure as Code)

Створюю новий user group:



Створюю користувача, якого додаю до новоствореної групи користувачів:



Для того, щоб можна було підключатися під цим користувачем з GitHub, створюю ключ доступу для нього:

Access keys (1)

Create access key

Use access keys to send programmatic calls to AWS from the AWS CLI, AWS Tools for PowerShell, AWS SDKs, or direct AWS API calls. You can have a maximum of two access keys (active or inactive) at a time. [Learn more](#)

AKIAWNNHIJHLEI55HSG5	<div>Actions ▼</div>
Description AK-lab4	Status ✔ Active
Last used None	Created 1 minute ago
Last used region N/A	Last used service N/A

Переходжу на GitHub, в обраному репозиторії натискаю **Settings -> Secrets and variables -> Actions -> New repository secret**. У розгорнутому вікні по черзі записую обидві частини свого ключа доступу:

Actions secrets / New secret

Name *

AWS_ACCESS_KEY_ID

Secret *

AKIAWNNHIJHLEI55HSG5

Add secret

Actions secrets / New secret

Name *

AWS_SECRET_ACCESS_KEY

Secret *

Add secret

Задля створення робочого процесу на GitHub переходжу до вкладки **Actions** і обираю варіант **Python application**. Далі у файлі з розширенням *.yaml вношу необхідні зміни і натискаю **Commit changes**:

The screenshot shows the GitHub Actions workflow editor for a file named `pawnshop-app.yml` in the `main` branch. The workflow is titled "Python app deploy on aws by gitAction v1". It is triggered on a `push` to the `main` branch or a `pull_request` to the `main` branch. The workflow has a `permissions` section with `contents: read`. It includes a `jobs` section with a `main_job` that runs on `ubuntu-latest`. The `steps` section includes a `checkout` step and a `setup-python` step.

```
1 # This workflow will install Python dependencies, run tests and lint with a single version of Python
2 # For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-python
3
4 name: Python app deploy on aws by gitAction v1
5
6 on:
7   push:
8     branches: [ "main" ]
9   pull_request:
10    branches: [ "main" ]
11
12 permissions:
13   contents: read
14
15 jobs:
16   main_job:
17
18     runs-on: ubuntu-latest
19
20     steps:
21     - uses: actions/checkout@v3
22       with:
23         fetch-depth: 0
24     - name: Set up Python 3.10
25       uses: actions/setup-python@v3
26       with:
27         python-version: "3.10"
```

On the right side, the "Marketplace" tab is active, showing a search bar and a list of featured actions. The "Commit changes..." button is highlighted with a red circle.

Одразу після створення комміту активується конвеєр, який успішно відпрацьовує усі етапи:

The screenshot shows a GitHub Actions workflow run titled "Update python-app.yml #7" with a green checkmark indicating success. The workflow is named "Python app deploy on aws by gitAction v1". The left sidebar shows the "main_job" selected under the "Jobs" section. The main area displays the job details for "main_job", which succeeded 2 minutes ago in 12s. A list of steps is shown, each with a green checkmark and a duration:

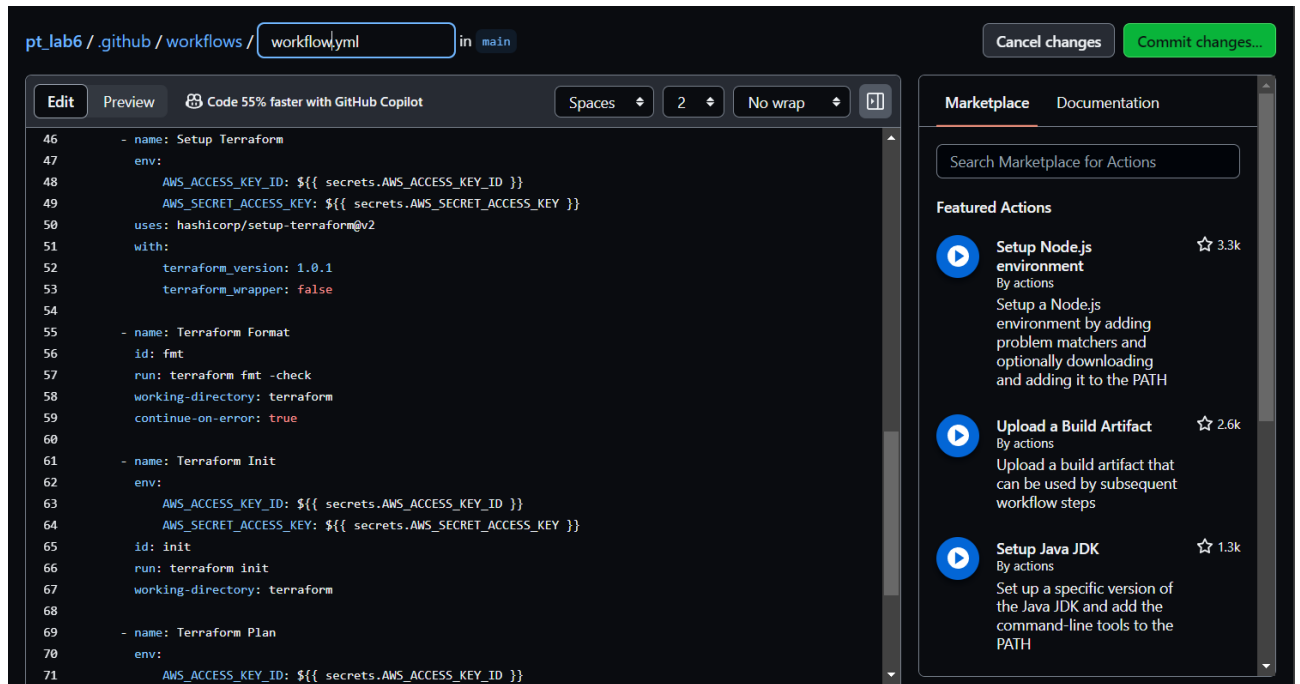
- > Set up job (0s)
- > Run actions/checkout@v3 (1s)
- > Set up Python 3.12 (0s)
- > Install dependencies (9s)
- > Lint with flake8 (0s)
- > Run tests (0s)
- > Post Set up Python 3.12 (0s)
- > Post Run actions/checkout@v3 (0s)
- > Complete job (0s)

Створюю каталог **terraform** у репозиторії, в який завантажую файл **main.tf** (опис інфраструктури) з таким вмістом:

The screenshot shows a code editor with the file "main.tf" open in the "pt_lab6 / terraform" repository. The code is Terraform configuration for an AWS security group. The editor has a dark theme and a sidebar on the left showing the file structure. The code is as follows:

```
1 terraform {
2   required_version = ">=0.13.0"
3   required_providers {
4     aws = {
5       source  = "hashicorp/aws"
6       version = "~> 3.0"
7     }
8   }
9 }
10
11 # Configure the AWS provider
12 provider "aws" {
13   region = "eu-central-1"
14 }
15
16 resource "aws_security_group" "web_app" {
17   name        = "web_app"
18   description = "security group"
19   ingress {
20     from_port = 80
21     to_port   = 80
22     protocol  = "tcp"
23     cidr_blocks = ["0.0.0.0/0"]
24   }
25
26   ingress {
27     from_port = 22
28     to_port   = 22
29     protocol  = "tcp"
```

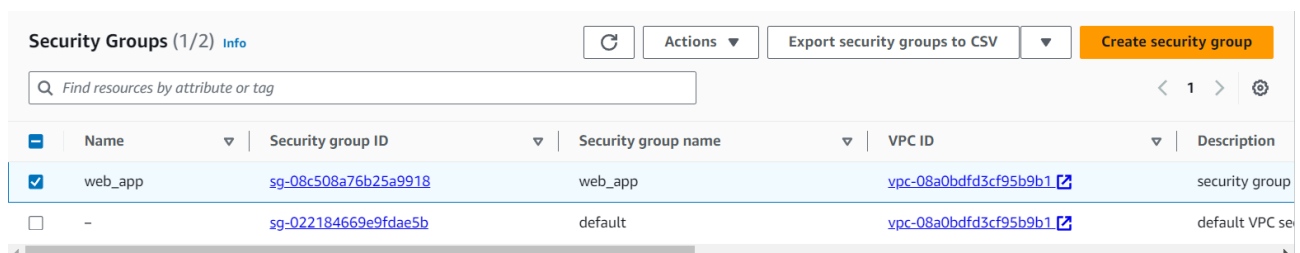
Доповнюю workflow.yml файл кодом для запуску відповідних виконавців команд terraform і роблю комміт:



Конвеєр знову запускається, проте виникає помилка:



Для її усунення слід в AWS перейти у сервіс EC2 -> Security Groups та видалити групу web_app (виділена пташкою):



Тоді конвеєр успішно спрацьовує:

Update workflow.yml #2

main_job succeeded now in 57s

- Set up job 2s
- Run actions/checkout@v3 1s
- Set up Python 3.12 0s
- Install dependencies 10s
- Lint with flake8 1s
- Run tests 0s
- Setup Terraform 0s
- Terraform Format 0s
- Terraform Init 2s
- Terraform Plan 3s
- Terraform Apply 32s
- Post Set up Python 3.12 0s
- Post Run actions/checkout@v3 0s
- Complete job 0s

Відповідна інфраструктура також розгортається у хмарі AWS:

Instances (1) Info

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DN
webapp_instance	i-00b4856dcea124da4	Running	t2.micro	2/2 checks passed	View alarms	eu-central-1a	ec2-3-77-201-1

Security Groups (2) Info

Name	Security group ID	Security group name	VPC ID	Description
-	sg-022184669e9fdae5b	default	vpc-08a0bdfd3cf95b9b1	default VPC se
web_app	sg-0179eba62be9647f4	web_app	vpc-08a0bdfd3cf95b9b1	security group

Для розміщення застосунку у хмарі скористаємося сервісом Amazon Lightsail.

Спершу створюю новий приватний репозиторій за допомогою сервісу Amazon Elastic Container Registry:

Private repositories

Repositories (1)

Repository name	URI	Created at	Tag immutability	Scan frequency	Encryption type
repo-lab4	441121786326.dkr.ecr.eu-central-1.amazonaws.com/repo-lab4	20 грудня 2023 р., 17:39:35 (UTC+02)	Disabled	Manual	AES-256

Для того, щоб Lightsail отримав доступ до репозиторію, надам йому відповідні дозволи. **Обираю репозиторій -> Actions -> Permissions.** Результат має вигляд:

Amazon ECR > Private registry > Repositories > repo-lab4 > Permissions

Permissions

Edit policy JSON Edit

▼ lightsail access permission

Effect	Service principals
Allow	-
Principal	AWS Account IDs
-	-

IAM Entities (1)

< 1 >

Name	Path	Type
user-lab4	/	user

Actions

- ecr:BatchCheckLayerAvailability
- ecr:BatchDeleteImage
- ecr:BatchGetImage
- ecr:CompleteLayerUpload

Додам ще два секрети у GitHub репозиторії: `REPOSITORY_NAME` (назва репозиторію) та `REPOSITORY_URI` (посилання на репозиторій):

Actions secrets / New secret

Name *

REPOSITORY_NAME

Secret *

repo-lab4

Actions secrets / New secret

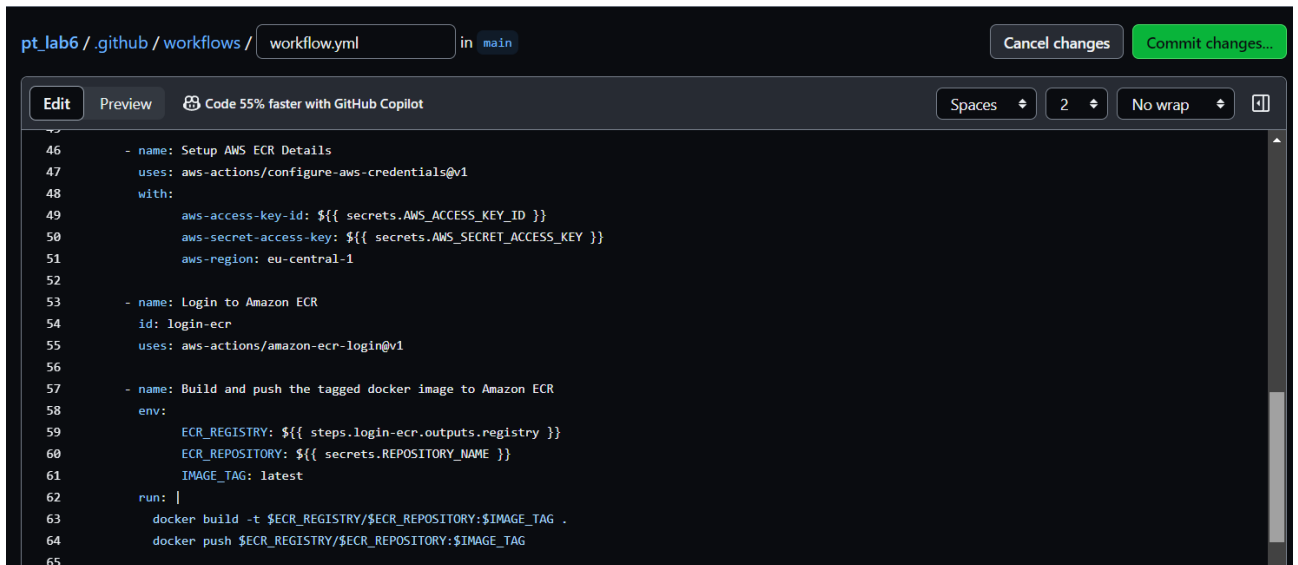
Name *

REPOSITORY_URI

Secret *

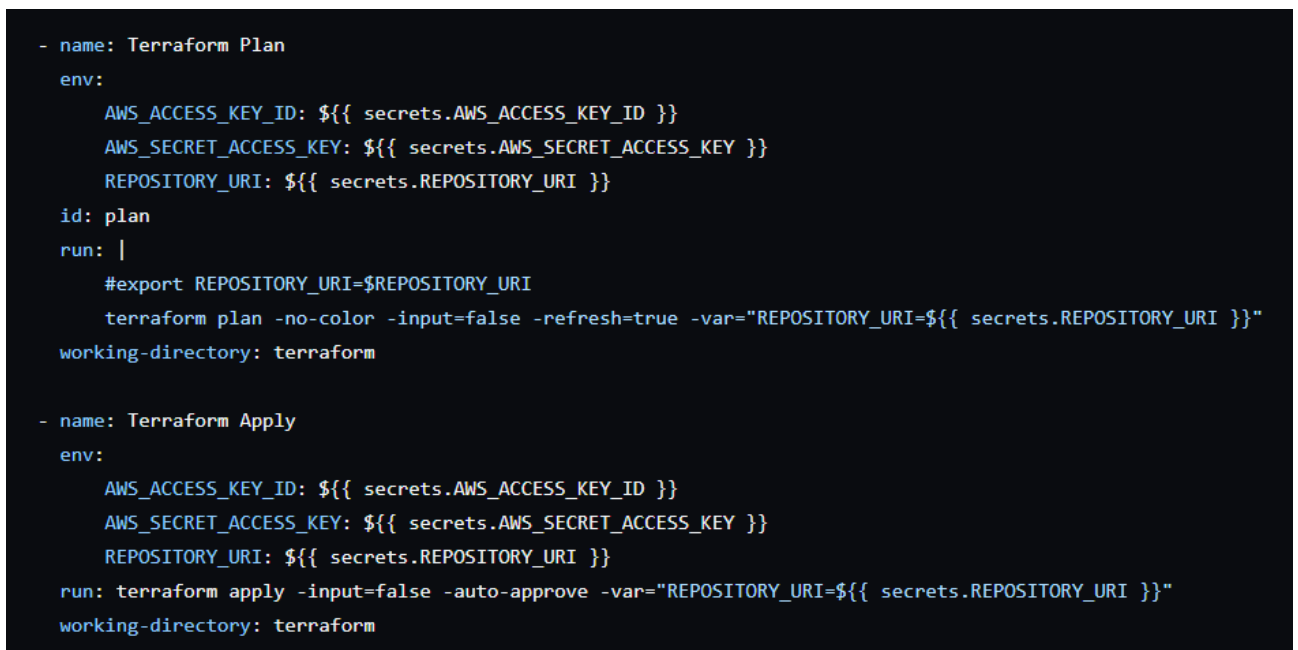
441121786326.dkr.ecr.eu-central-1.amazonaws.com/repo-lab4

Доповнюю workflow.yml кодом для підключення до AWS, формування Docker-образу та його вивантаження в репозиторій Amazon ECR:



```
46 - name: Setup AWS ECR Details
47   uses: aws-actions/configure-aws-credentials@v1
48   with:
49     aws-access-key-id: ${ secrets.AWS_ACCESS_KEY_ID }
50     aws-secret-access-key: ${ secrets.AWS_SECRET_ACCESS_KEY }
51     aws-region: eu-central-1
52
53 - name: Login to Amazon ECR
54   id: login-ecr
55   uses: aws-actions/amazon-ecr-login@v1
56
57 - name: Build and push the tagged docker image to Amazon ECR
58   env:
59     ECR_REGISTRY: ${ steps.login-ecr.outputs.registry }
60     ECR_REPOSITORY: ${ secrets.REPOSITORY_NAME }
61     IMAGE_TAG: latest
62   run: |
63     docker build -t $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG .
64     docker push $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG
65
```

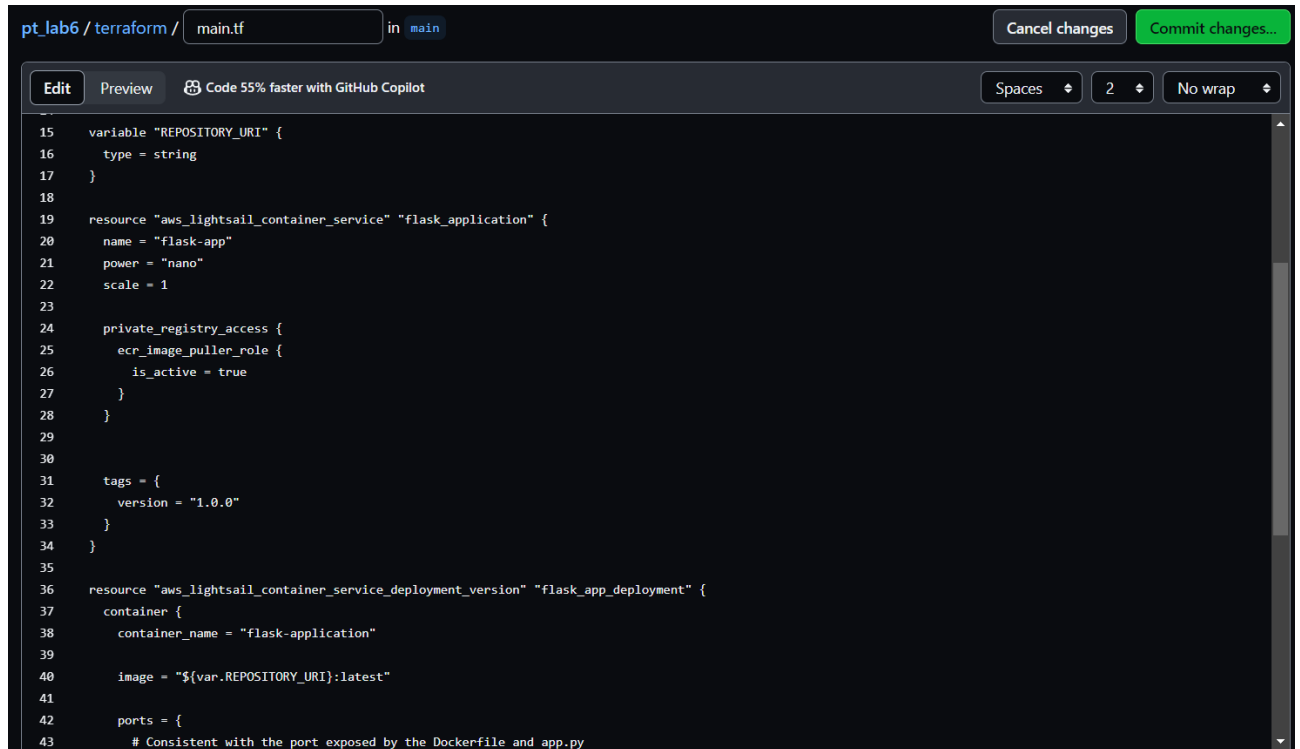
Задаю адресу образу через змінну середовища REPOSITORY_URI для того, щоб Terraform міг в подальшому розгорнути Lightsail:



```
- name: Terraform Plan
  env:
    AWS_ACCESS_KEY_ID: ${ secrets.AWS_ACCESS_KEY_ID }
    AWS_SECRET_ACCESS_KEY: ${ secrets.AWS_SECRET_ACCESS_KEY }
    REPOSITORY_URI: ${ secrets.REPOSITORY_URI }
  id: plan
  run: |
    #export REPOSITORY_URI=$REPOSITORY_URI
    terraform plan -no-color -input=false -refresh=true -var="REPOSITORY_URI=${ secrets.REPOSITORY_URI }"
  working-directory: terraform

- name: Terraform Apply
  env:
    AWS_ACCESS_KEY_ID: ${ secrets.AWS_ACCESS_KEY_ID }
    AWS_SECRET_ACCESS_KEY: ${ secrets.AWS_SECRET_ACCESS_KEY }
    REPOSITORY_URI: ${ secrets.REPOSITORY_URI }
  run: terraform apply -input=false -auto-approve -var="REPOSITORY_URI=${ secrets.REPOSITORY_URI }"
  working-directory: terraform
```

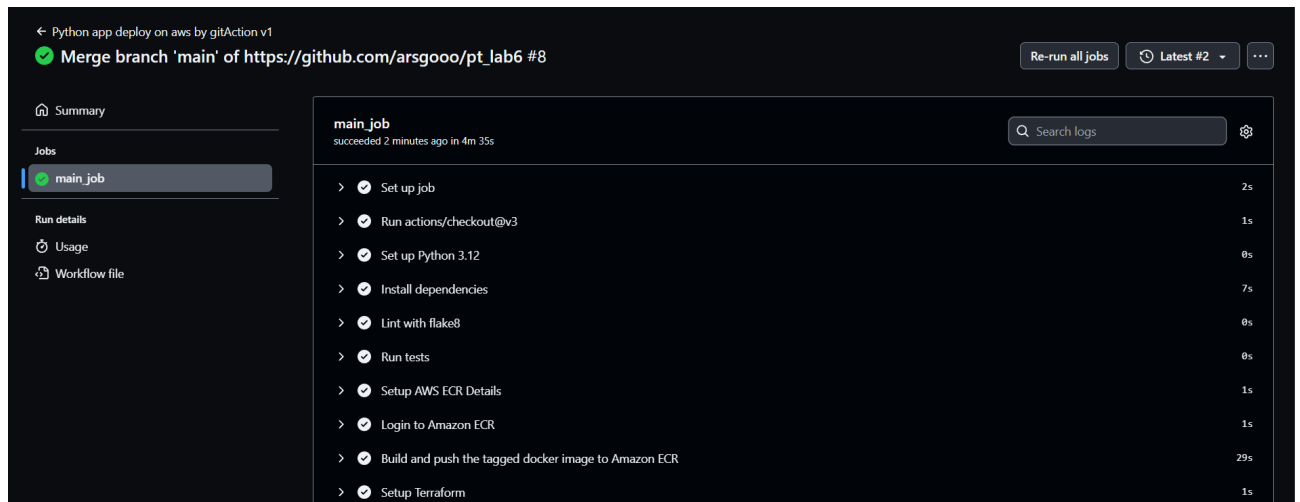

Також редагую вміст файлу main.tf:



The screenshot shows a code editor interface for a Terraform file named `main.tf`. The editor has tabs for 'Edit' and 'Preview', and a status bar indicating 'Code 55% faster with GitHub Copilot'. The code defines a variable `REPOSITORY_URI` and two resources: `aws_lightsail_container_service.flask_application` and `aws_lightsail_container_service_deployment_version.flask_app_deployment`. The first resource configures the container service with a nano instance, private registry access, and a tag. The second resource configures the deployment version with the container name and the repository URI.

```
15 variable "REPOSITORY_URI" {
16   type = string
17 }
18
19 resource "aws_lightsail_container_service" "flask_application" {
20   name = "flask-app"
21   power = "nano"
22   scale = 1
23
24   private_registry_access {
25     ecr_image_puller_role {
26       is_active = true
27     }
28   }
29
30
31   tags = {
32     version = "1.0.0"
33   }
34 }
35
36 resource "aws_lightsail_container_service_deployment_version" "flask_app_deployment" {
37   container {
38     container_name = "flask-application"
39
40     image = "${var.REPOSITORY_URI}:latest"
41
42     ports = {
43       # Consistent with the port exposed by the Dockerfile and app.py
```

Конвеєр вже традиційно успішно відпрацьовує:



The screenshot shows the summary of a GitHub Actions workflow run. The workflow is named 'Python app deploy on aws by gitAction v1' and is triggered by a merge to the 'main' branch. The run is successful and took 4 minutes and 35 seconds. The summary lists the steps of the workflow, including setting up the job, checking out the code, setting up Python, installing dependencies, linting, running tests, setting up AWS ECR, logging in to ECR, building and pushing the Docker image, and setting up Terraform.

Python app deploy on aws by gitAction v1
Merge branch 'main' of https://github.com/arsgo00/pt_lab6 #8
Re-run all jobs Latest #2

Summary
Jobs
main_job
Run details
Usage
Workflow file

main_job
succeeded 2 minutes ago in 4m 35s
Search logs

- Set up job 2s
- Run actions/checkout@v3 1s
- Set up Python 3.12 0s
- Install dependencies 7s
- Lint with flake8 0s
- Run tests 0s
- Setup AWS ECR Details 1s
- Login to Amazon ECR 1s
- Build and push the tagged docker image to Amazon ECR 29s
- Setup Terraform 1s


Перейшовши у сервіс Lightsail, можна побачити новостворений контейнер:

Good afternoon!

Filter by name, location, tag, or type

Sort by **Date** ▾

Create container service

**flask-app**
Nano (512 MB RAM, 0.25 vCPUs) ×1 node

Running

Frankfurt

version ↗ 1.0.0

Для перевірки працездатності застосунку заходжу в цей контейнер та переходжу за посиланням public domain:



flask-app

Container service
Nano (512 MB RAM, 0.25 vCPUs) ×1 node
Frankfurt

Disable

Status: **Running**

Public domain: flask-app.66u5g5r32dl52.eu-central-1.cs.amazonlightsail.com

[How do I use my domain with my container service?](#)

Private domain: flask-app.service.local

Getting started ×

Deployments

Capacity

Images

Custom domains

Metrics

Current deployment

Your deployment is the set of containers currently running on your container service.

[Learn more about deployments](#)

[Modify your deployment](#)

CONTAINERS

Deployed: December 21, 2023 - 12:27 PM

Public endpoint

[flask-application](#)

Image: 441121786326.dkr.ecr.eu-central-1.amazonaws.com/repo-lab4:latest

Open log

← → ↺ flask-app.66u5g5r32dl52.eu-central-1.cs.amazonlightsail.com

Available Goods

- Radio: 200
- Headphones: 300
- Watch: 499.55555
- Ring: 749.47755

Add Good

Name: Price:

Remove Good

Name:

Goods Amount

Total number of goods: 4

Total Cost

Total cost of goods: \$1749.0331

Завдання 2 (дослідження довільного сервісу AWS та його розгортання у хмарі з використанням Terraform та GitHub Actions)

Для реалізації цього завдання скористаюся технологією OpenID.

OpenID – це відкритий стандарт аутентифікації, який дозволяє користувачам автентифікуватися на різних веб-сайтах та службах, використовуючи одні й ті самі облікові дані. Це стандарт для одноразової аутентифікації (Single Sign-On, SSO), який спрощує процес реєстрації та входу на веб-ресурси.

Основні складові OpenID включають:

1. **OpenID Provider (OP):** Це система, яка виконує аутентифікацію користувачів і надає їм унікальні ідентифікатори (OpenID Identifier). Сервер OpenID Provider може бути розгорнутий на окремому веб-сайті або вбудований в інші послуги.
2. **OpenID Relying Party (RP):** Це веб-сайти або служби, які використовують OpenID для аутентифікації своїх користувачів. Relying Party отримує унікальний ідентифікатор від користувача через процес OpenID.
3. **OpenID Identifier:** Унікальний ідентифікатор, який видається користувачеві OpenID Provider. Він може бути представлений URL або ідентифікатором.

Спершу створюю провайдера ідентифікаційних даних для GitHub. Для цього переходжу в IAM -> Access management -> Identity providers -> Add provider. У вікні, що розгорнулося, вказую такі дані:

The screenshot shows the AWS IAM console interface for adding a new identity provider. The breadcrumb navigation at the top reads: IAM > Identity providers > Create Identity Provider. The main heading is 'Add an Identity provider' with an 'Info' link. Below this is a 'Configure provider' section. Under 'Provider type', 'OpenID Connect' is selected with a radio button. The 'Provider URL' field contains 'https://token.actions.githubusercontent.com' and has a 'Get thumbprint' button next to it. The 'Audience' field contains 'sts.amazonaws.com'. Each field has a maximum character limit of 255.

IAM > Identity providers > Create Identity Provider

Add an Identity provider [Info](#)

Configure provider

Provider type [Info](#)

☐ SAML
Establish trust between your AWS account and a SAML 2.0 compatible Identity Provider such as Shibboleth or Active Directory Federation Services.

☒ OpenID Connect
Establish trust between your AWS account and Identity Provider services, such as Google or Salesforce.

Provider URL
Specify the secure OpenID Connect URL for authentication requests.

[Get thumbprint](#)

Maximum 255 characters. URL must begin with "https"

Audience [Info](#)
Specify the client ID issued by the Identity provider for your app.

Maximum 255 characters. Use alphanumeric or '._-/' characters.

Також створюю нову роль (вкладка Roles):

Select trusted entity [Info](#)

Trusted entity type

☐ AWS service
Allow AWS services like EC2, Lambda, or others to perform actions in this account.

☐ AWS account
Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.

☒ Web identity
Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.

☐ SAML 2.0 federation
Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

☐ Custom trust policy
Create a custom trust policy to enable others to perform actions in this account.

Web identity

Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.

Identity provider

token.actions.githubusercontent.com

↕

↻

Create new ↗

Audience

sts.amazonaws.com

▼

Подальші кроки аналогічні до тих, що були виконані у першому завданні (створення репозиторію, надання дозволів, створення файлу terraform тощо). Проте суттєвою відмінністю є вміст файлу workflow.yml, який цього разу виглядає так (наведено лише відмінні фрагменти):

```
- name: Configure AWS Credentials
  uses: aws-actions/configure-aws-credentials@v1
  with:
    role-to-assume: arn:aws:iam::441121786326:role/pr3-role
    aws-region: eu-central-1

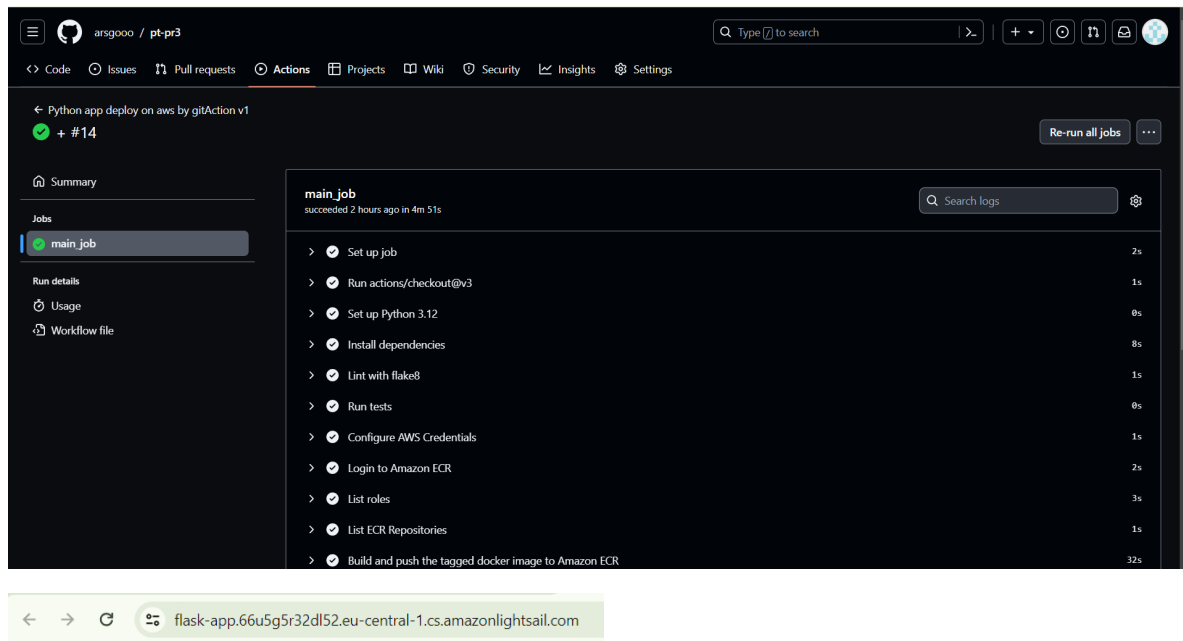
- name: Login to Amazon ECR
  id: login-ecr
  uses: aws-actions/amazon-ecr-login@v1
  with:
    mask-password: true

- name: List roles
  run: |
    aws sts get-caller-identity

- name: List ECR Repositories
  run: |
    aws ecr describe-repositories
```

Параметр **role-to-assume** вказує ARN (Amazon Resource Name) роль, яку GitHub Actions має «приміряти на себе». В цьому випадку це роль **pr3-role**. Параметр **mask-password: true** захищає конфіденційні дані пароля, приховуючи їх у виведеній інформації (інакше при запуску конвеєра буде виведено попередження про те, що пароль неприхований). **List roles** виводить інформацію про поточну роль, а **List ECR Repositories** виводить інформацію про репозиторії контейнерів, доступні в Amazon ECR.

Роблю останній комміт і перевіряю працездатність конвеєра:



Available Goods

- Radio: 200
- Headphones: 300
- Watch: 499.55555
- Ring: 749.47755

Add Good

Name: Price:

Remove Good

Name:

Goods Amount

Total number of goods: 4

Total Cost

Total cost of goods: \$1749.0331

Висновок: за час виконання лабораторної роботи було ретельно опрацьовано поняття інфраструктури як коду та розгорнуто її за допомогою Terraform та GitHub Actions. В першому завданні це було реалізовано шляхом залучення секретів. В другому ж використовувався стандарт OpenID, який дозволяє створити єдиний обліковий запис для автентифікації на багатьох непов'язаних між собою інтернет-ресурсах.