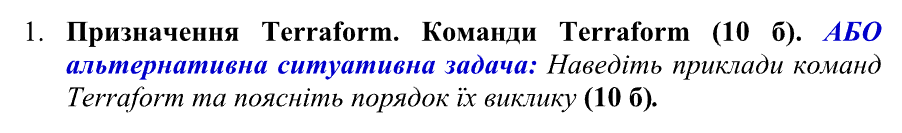
Екзаменаційна робота

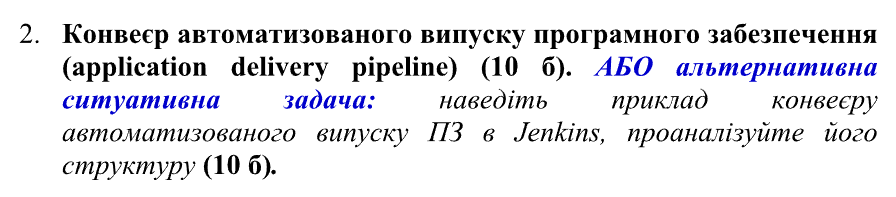
студента групи МІТ-31 Добровольського Арсенія Михайловича

з дисципліни «Технології програмування» 22.12.2023 року

Білет №1

**** До основних команд Terraform належать такі (наведено по порядку):

* terraform init – ініціалізує робочий каталог Terraform, завантажує провайдер (наприклад, AWS), модулі тощо;
* terraform validate (опціонально) – підтверджує правильність синтаксису файлу Terraform;
* terraform plan – вивчає конфігураційні файли для розуміння того, які зміни потрібно внести до наявної інфраструктури для досягнення бажаного стану;
* terraform apply – безпосередньо вносить зміни до інфраструктури;
* terraform show (опціонально) – дозволяє побачити поточний стан інфраструктури;
* terraform destroy (опціонально) – видаляє всі ресурси, які були створені Terraform.

****

pipeline {

    agent any

    environment {

        DOCKERHUB\_CREDENTIALS = credentials('arsgoo-dockerhub')

    }

    stages {

        stage('Check scm') {

            agent any

            steps {

                checkout scm

            }

        }

        stage('Build') {

            steps {

                echo "Building ...${BUILD\_NUMBER}"

                echo "Build completed"

            }

        }

        stage('Test') {

            agent {

                docker {

                    image 'alpine'

                    args '-u="root"'

                }

            }

            steps {

                sh 'apk add --update python3 py-pip'

                sh 'pip install xmlrunner'

                sh 'cp pawnshop\_tests.py .'

                sh 'python3 pawnshop\_tests.py'

            }

            post {

                always {

                    junit 'test-reports/\*.xml'

                }

                success {

                    echo "Tests successfully completed!"

                }

                failure {

                    echo "Ops! Tests failed! Try again, please"

                }

            }

        }

        stage('Image creation') {

            steps {

                sh 'cp Dockerfile .'

                sh 'docker build -t arsgoo/pawnshop\_tests:latest .'

            }

        }

        stage('Login') {

            steps {

                sh 'echo $DOCKERHUB\_CREDENTIALS\_PSW | docker login -u $DOCKERHUB\_CREDENTIALS\_USR --password-stdin'

            }

        }

        stage('Push') {

            steps {

                sh 'docker push arsgoo/pawnshop\_tests:latest'

            }

        }

    }

    post {

        always {

            sh 'docker logout'

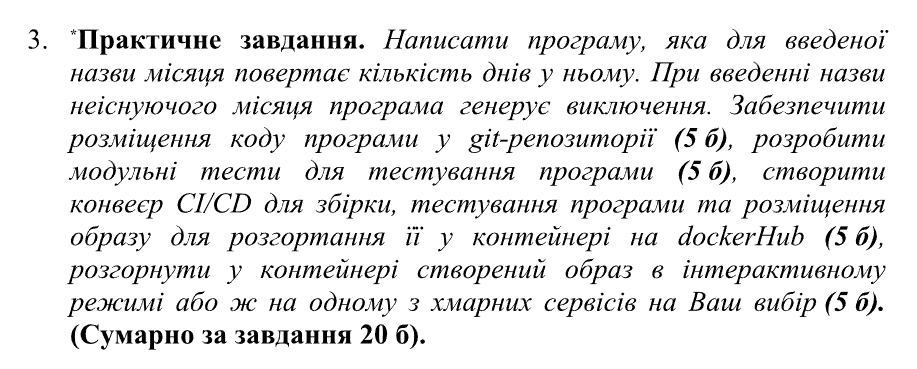
        }

    }

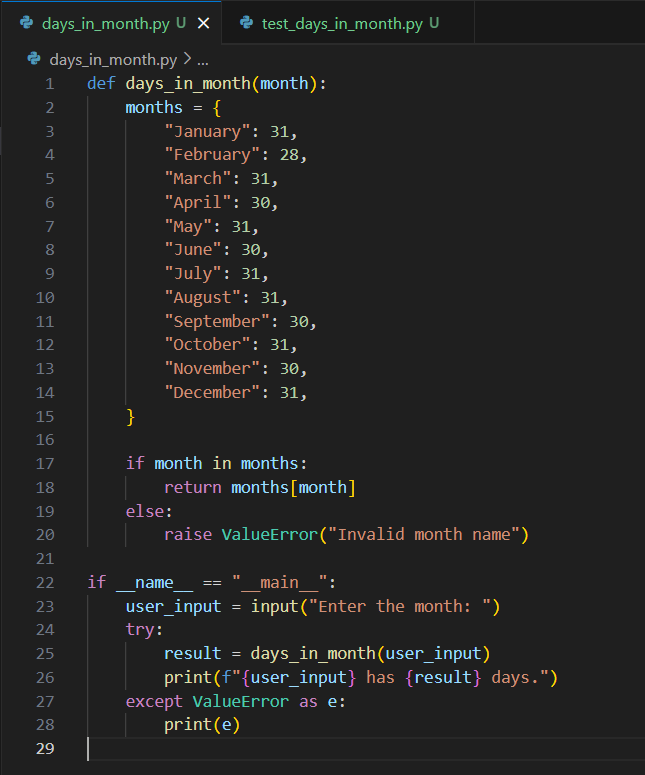
}

**Аналіз структури конвеєру:**

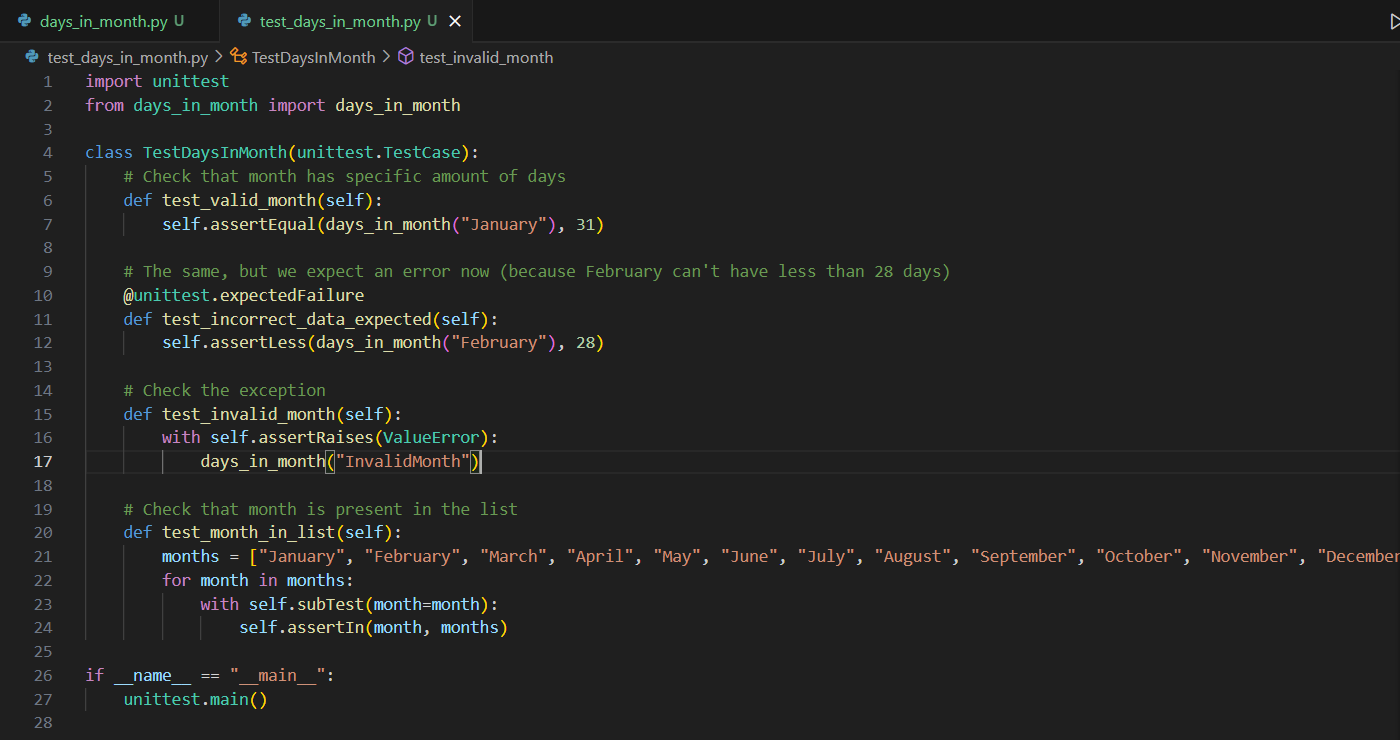
* **Pipeline** – оголошує конвеєр і визначає, де він повинен виконуватися (agent any вказує на використання будь-якого доступного агента).
* **Environment** – визначає змінні середовища, такі як шляхи до інструментів, URL репозиторію, назва Docker-образу тощо (у наведеному прикладі це облікові дані для доступу до DockerHub).
* **Stages** – оголошує окремі етапи конвеєру, які повинні бути виконані.
* **Steps** – описує дії, які потрібно виконати на кожному етапі (stage).
* **Post** – визначає дії, які виконуються після завершення конвеєру, залежно від стану конвеєру (завжди, успіх, провал тощо).



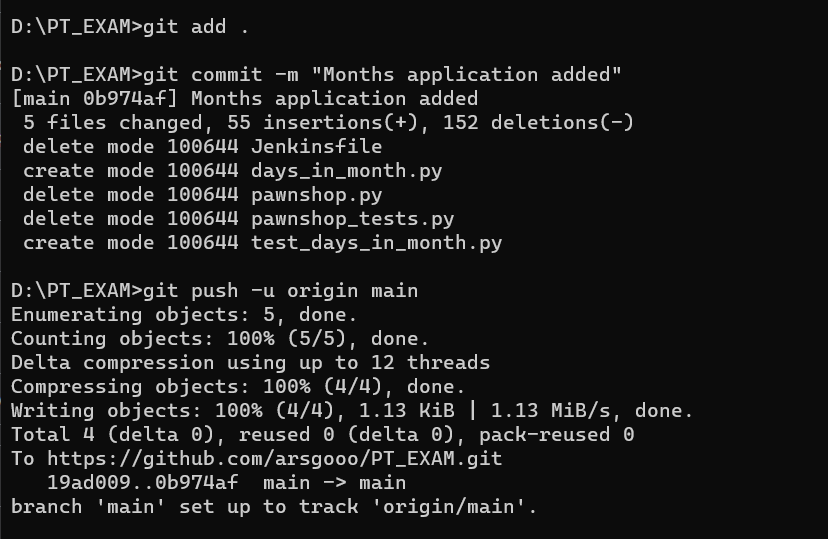
Основна програма:

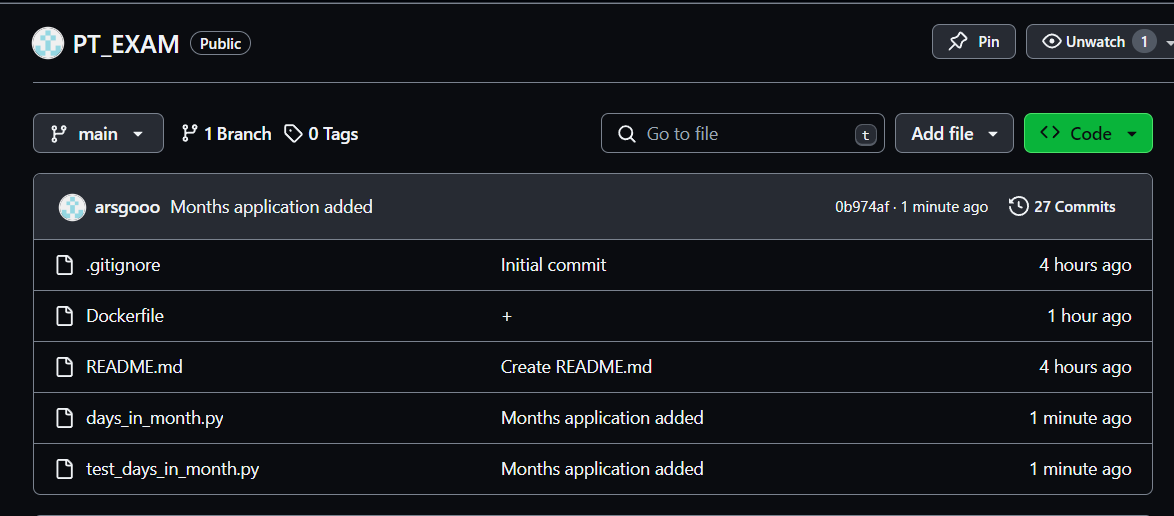


Файл з тестами:

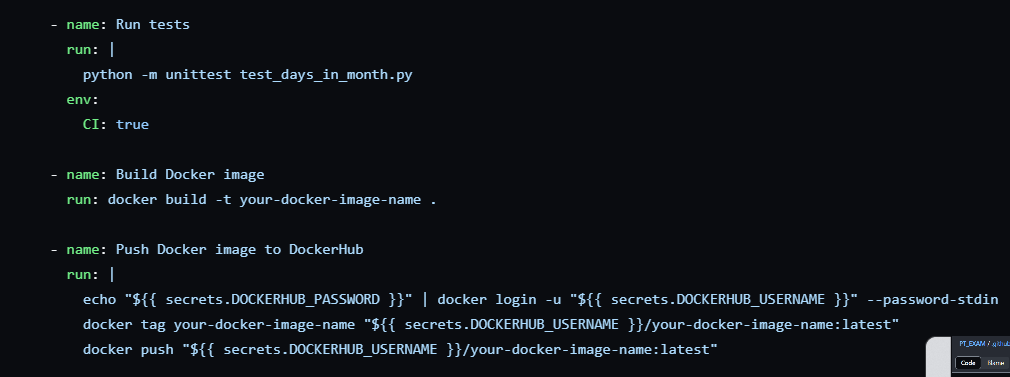


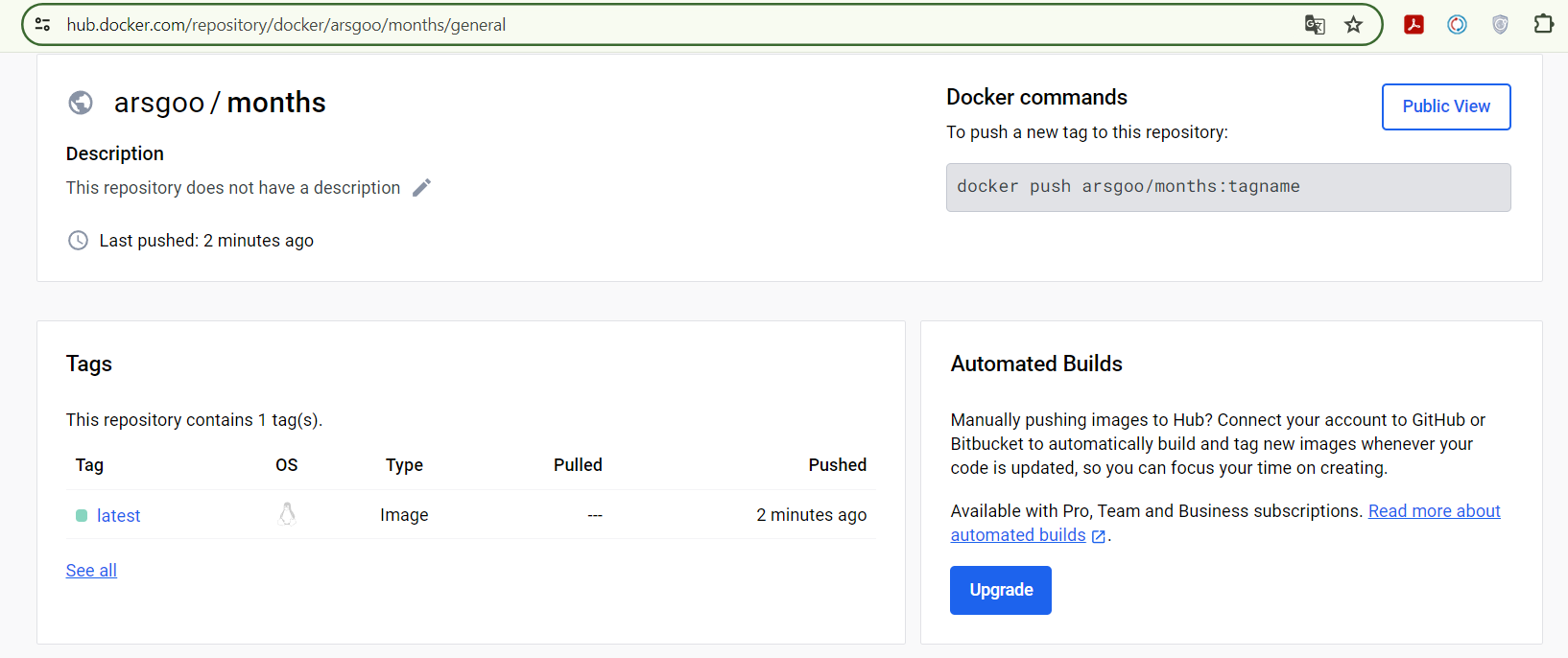
Розміщення програми у git-репозиторії:

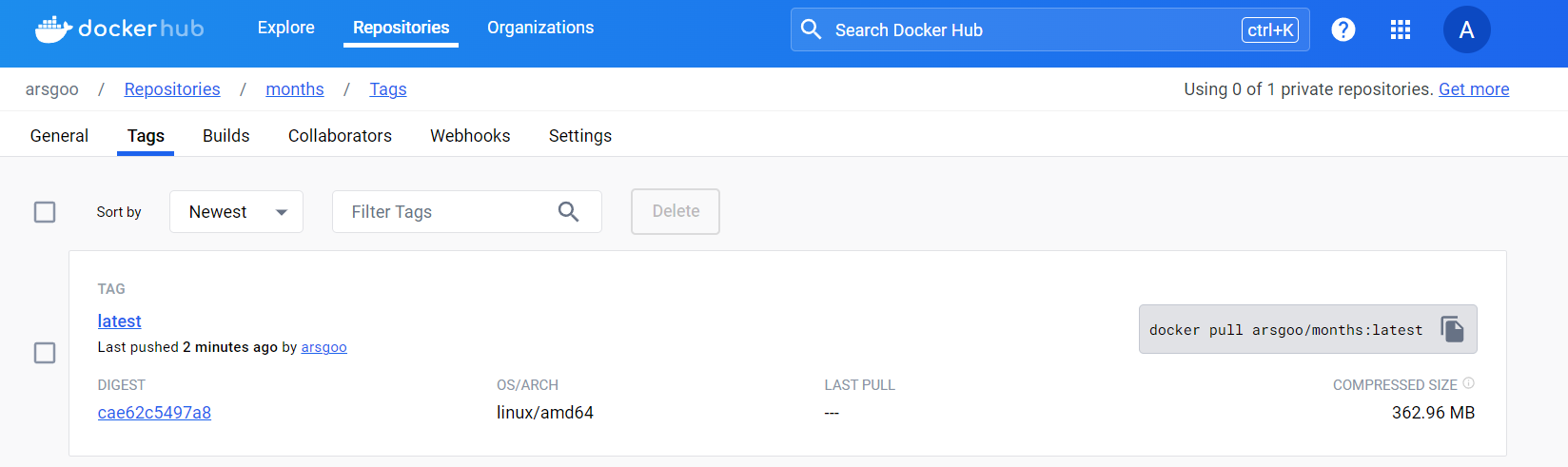




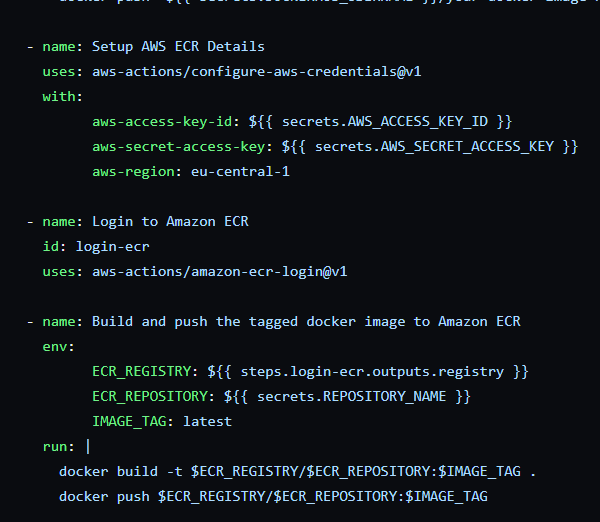
Побудова образу і вивантаження його на DockerHub (yml-файл):

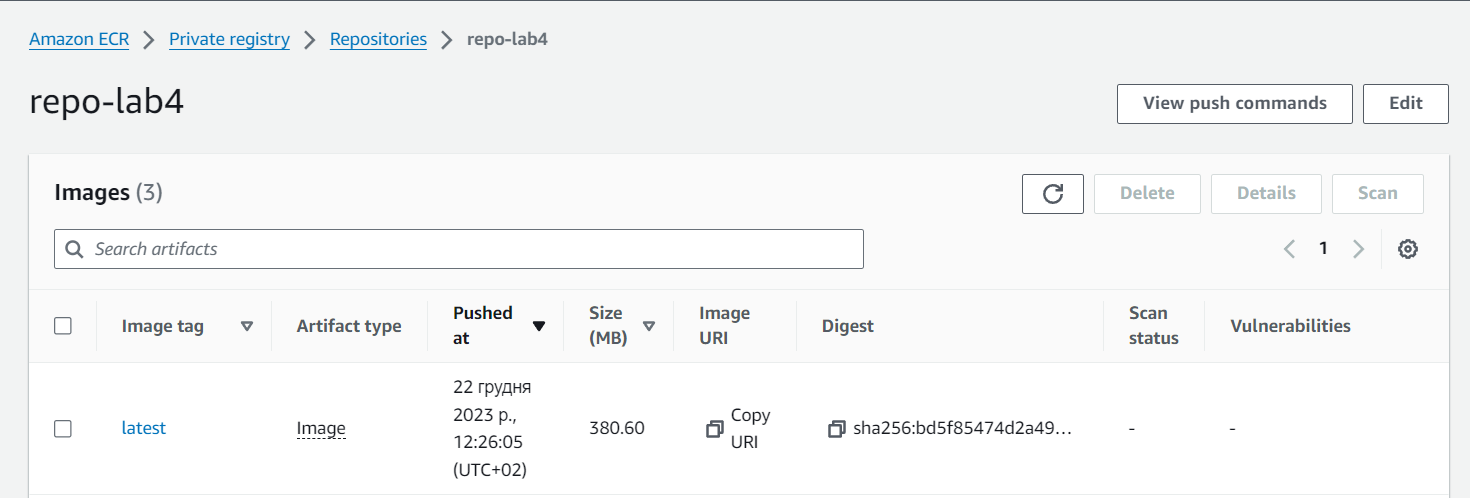






Розгортання образу в хмарі:





**(Скриншот не з лабораторної, просто не встиг створити новий репо)**

При запуску конвеєра на етапі Terraform Apply виникає помилка:

