Practical Steps: Reproducing the System

The complete project source code is available on GitHub: https://github.com/szelese/ci-cd-gha-aws

The solution is built on GitHub Actions automation and the AWS Elastic Beanstalk service. Development utilized Python 3.11 and Django 4.2 LTS, with secure authentication provided by AWS IAM OpenID Connect (OIDC) integration. The project goal was to create a working prototype capable of executing every step automatically, from code modification to production deployment in the cloud.

Step 0: Preparations

Before starting, you must prepare the accounts and tools used for development and cloud deployment.

Accounts: Create a GitHub account for source code version control and CI/CD pipelines, and an AWS account for cloud infrastructure hosting.

Tools: I recommend installing GitBash and the ZIP command (via MSYS2).

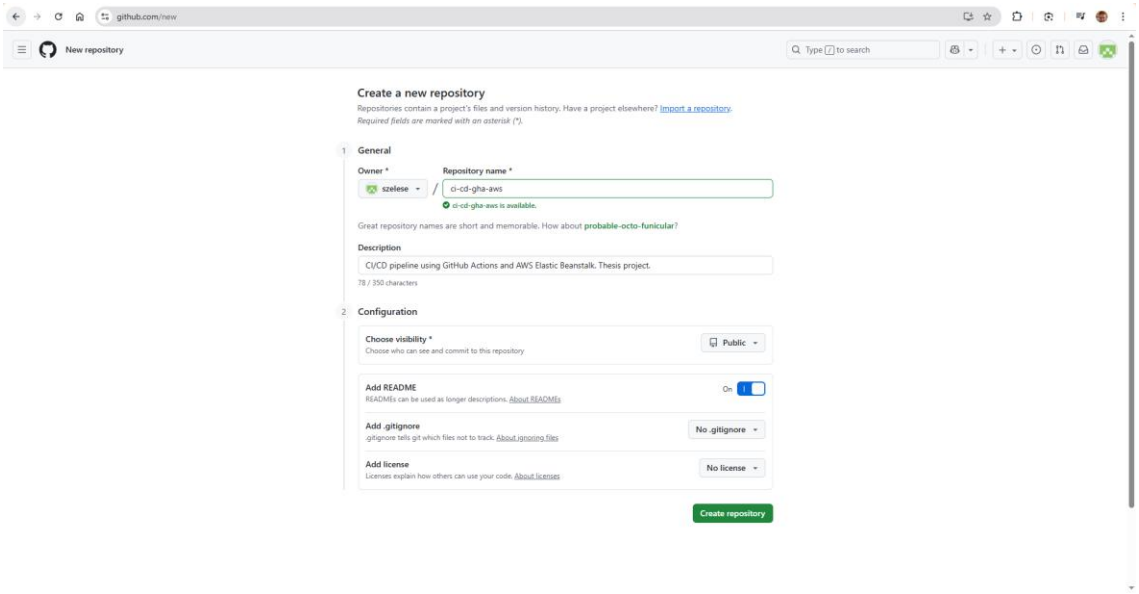
Environment (Optional):

This project was developed using Python 3.11 and managed locally with Visual Studio Code, but it is compatible with other IDEs like PyCharm or IDLE.

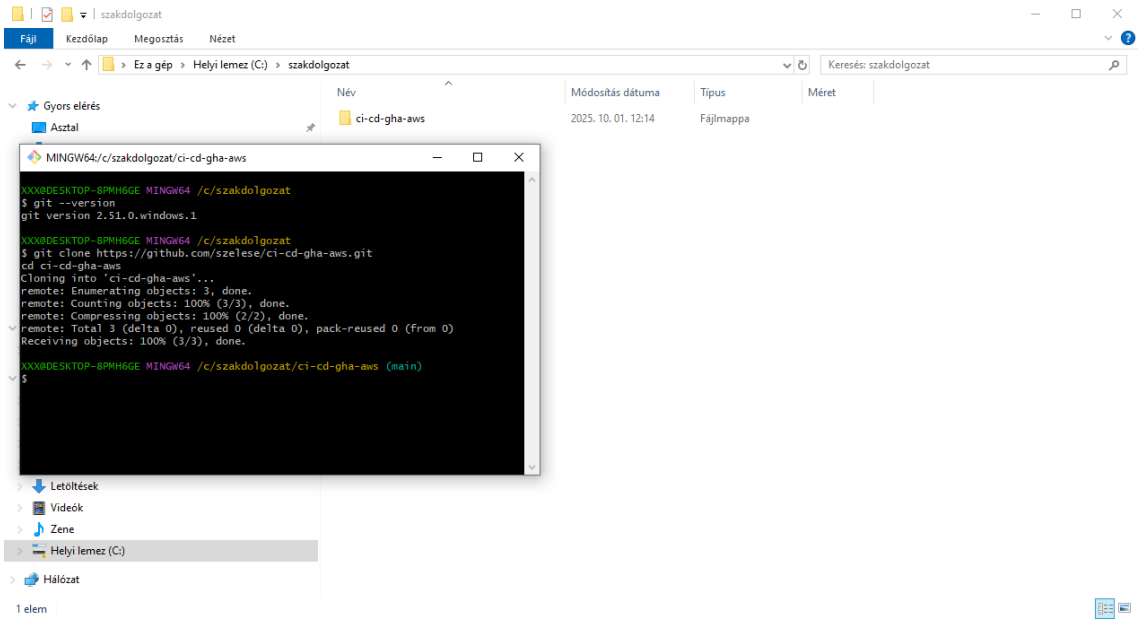
While simple text editors work, be cautious of line ending (LF/CRLF) or EOF (End Of File) errors.

Development and testing were performed on Windows 10 Enterprise, but the system can run on any operating system.

Step 1: Create New Repository



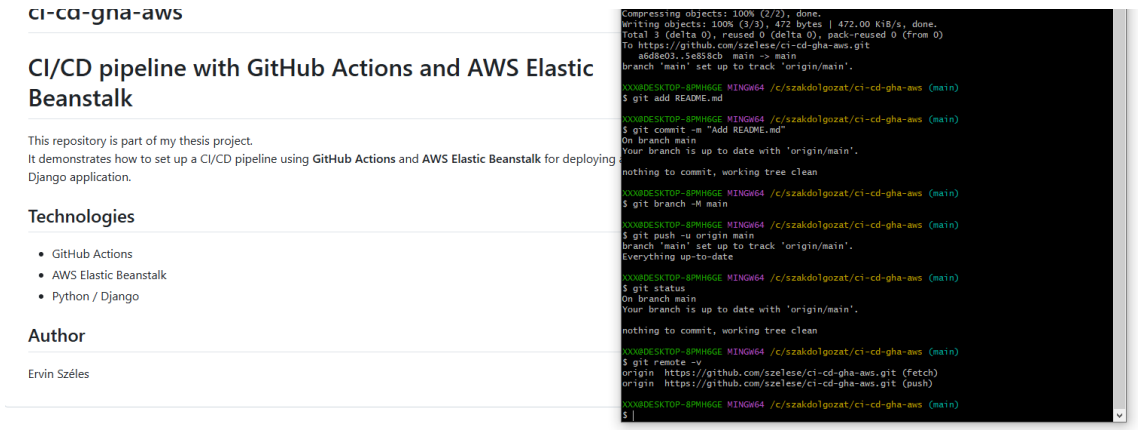
Step 2: Check GitBash and Clone Repository



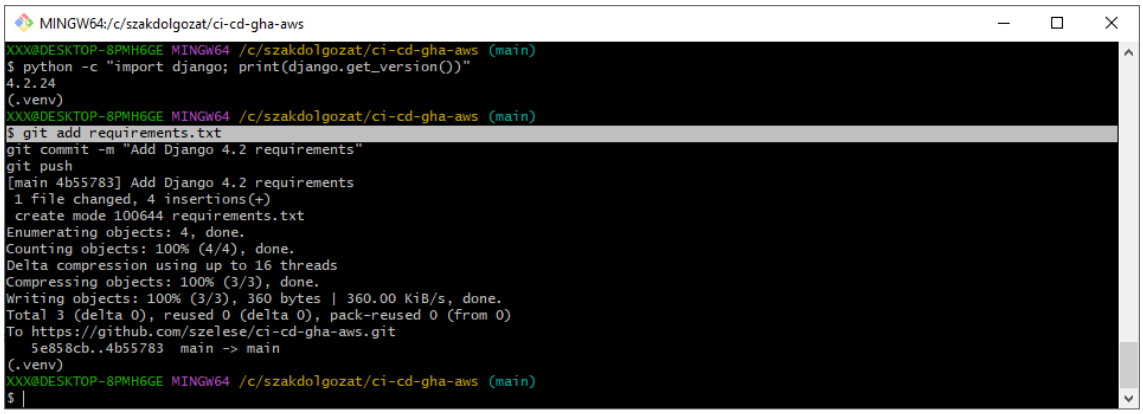
Verify your git version and clone the repo to your local machine:

```
git --version
git clone https://github.com/szelese/ci-cd-gha-aws/
```

Step 3: README.md and requirements.txt Modification, Commit, Push, and Verification

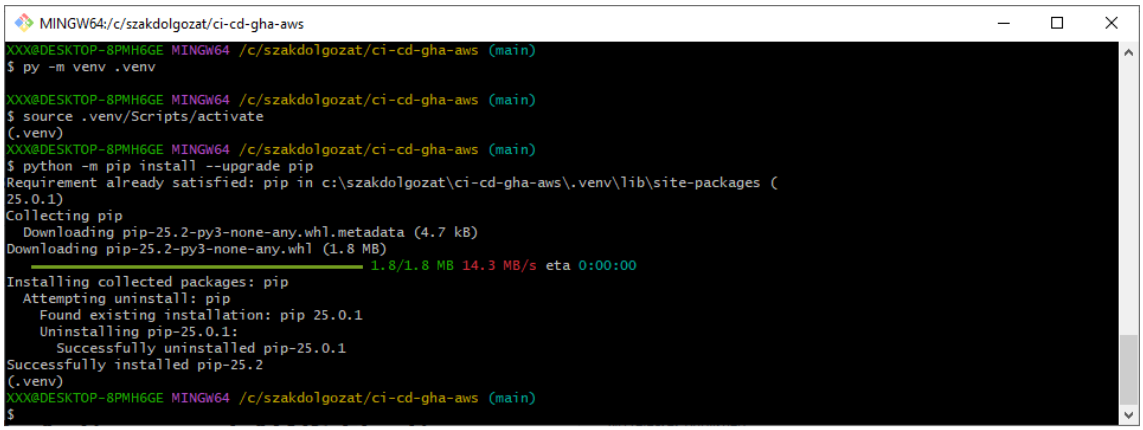



```
git add README.md
git commit -m "Add README.md"
git branch -M main
git push -u origin main
git status
git remote -v
```



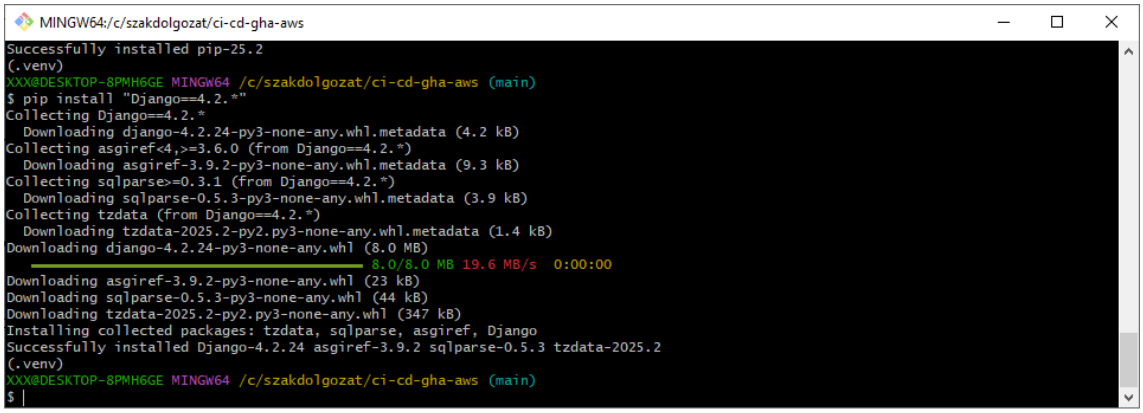

```
git add requurements.txt
git commit -m „Add Django 4.2 requirements”
```

Step 4: Virtual Environment Creation, Activation, and Pip Upgrade



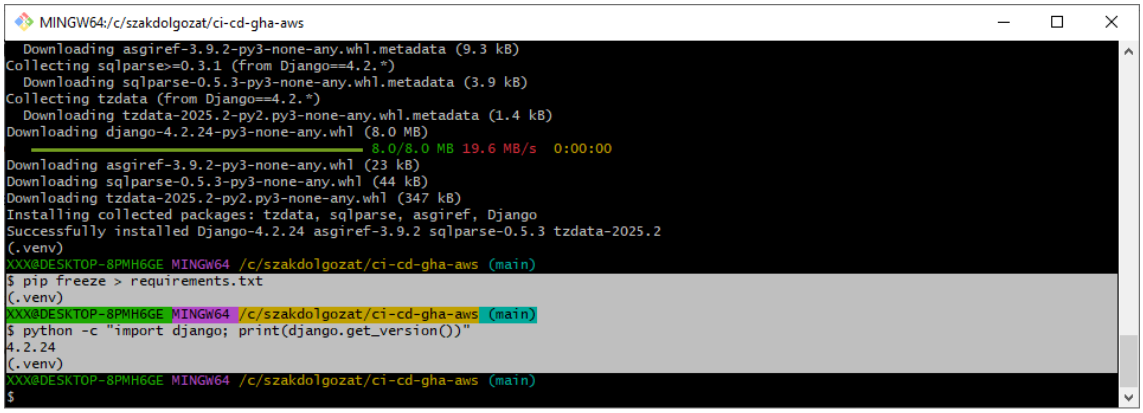

```
py -m venv .venv
source .venv/Scripts/activate
# (.venv) ha a prompt elején megjeleik, akkor azt jelenti, hogy aktív
python -m pip install --upgrade pip
```

Step 5: Django Installation (LTS 4.2)



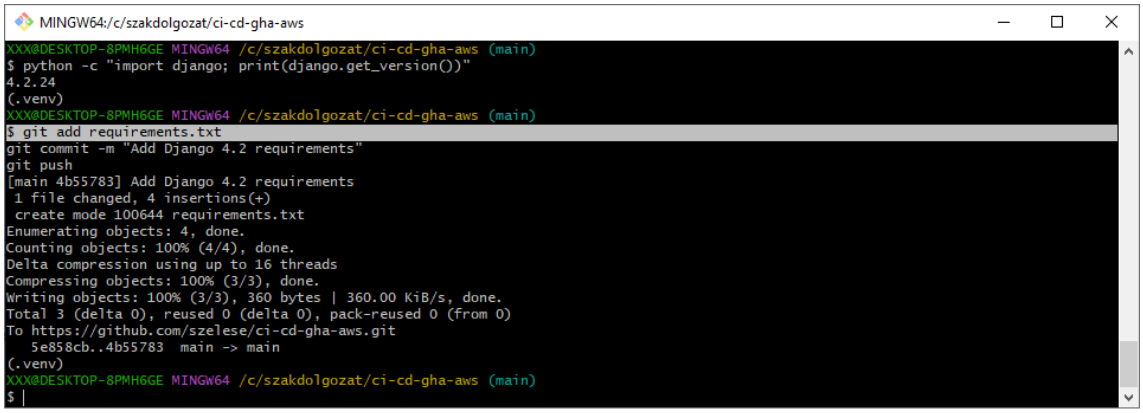

```
pip install "Django==4.2.*"
```

Step 6: Outputting the Requirements File and Quick Verification



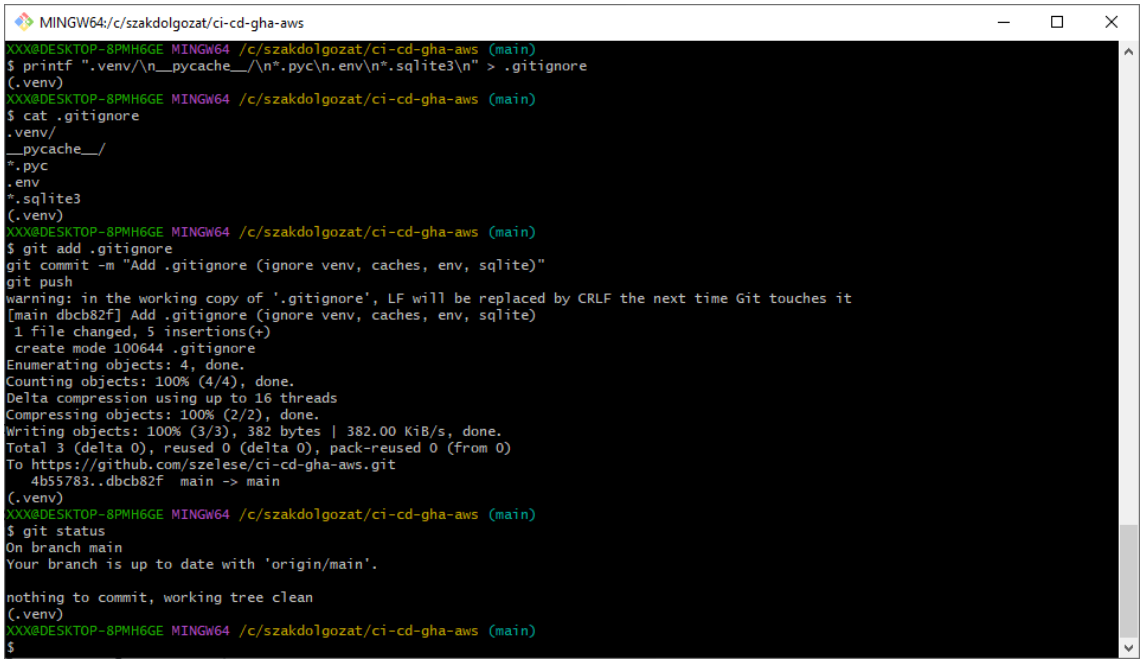

```
pip freeze > requirements.txt
python -c "import django; print(django.get_version())"
# Expected and correct verification value:: 4.2.*
```

Step 7: commit&push




```
git add requirements.txt
git commit -m "Add Django 4.2 requirements"
git push
```

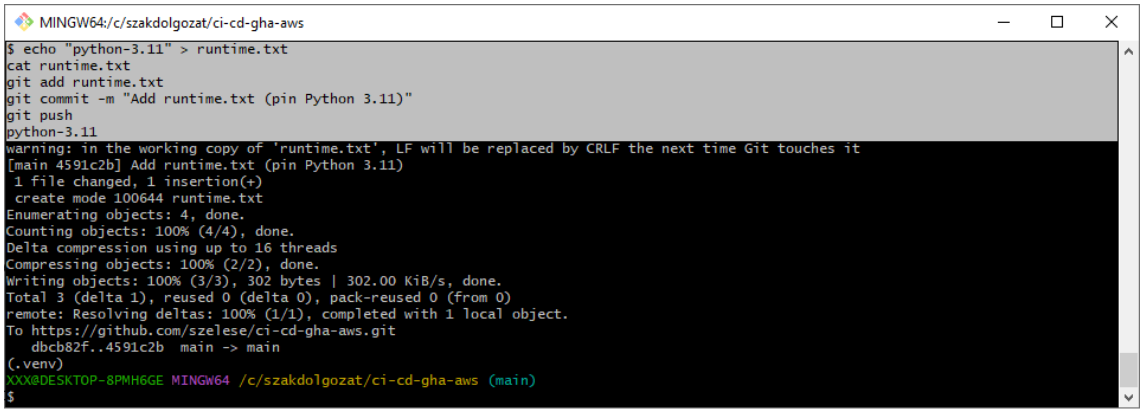
Step 8: .gitignore Creation and Verification



Technical justification: The .venv directory is machine and OS-dependent, can be extremely large, and must be rebuilt in CI or on new machines based on

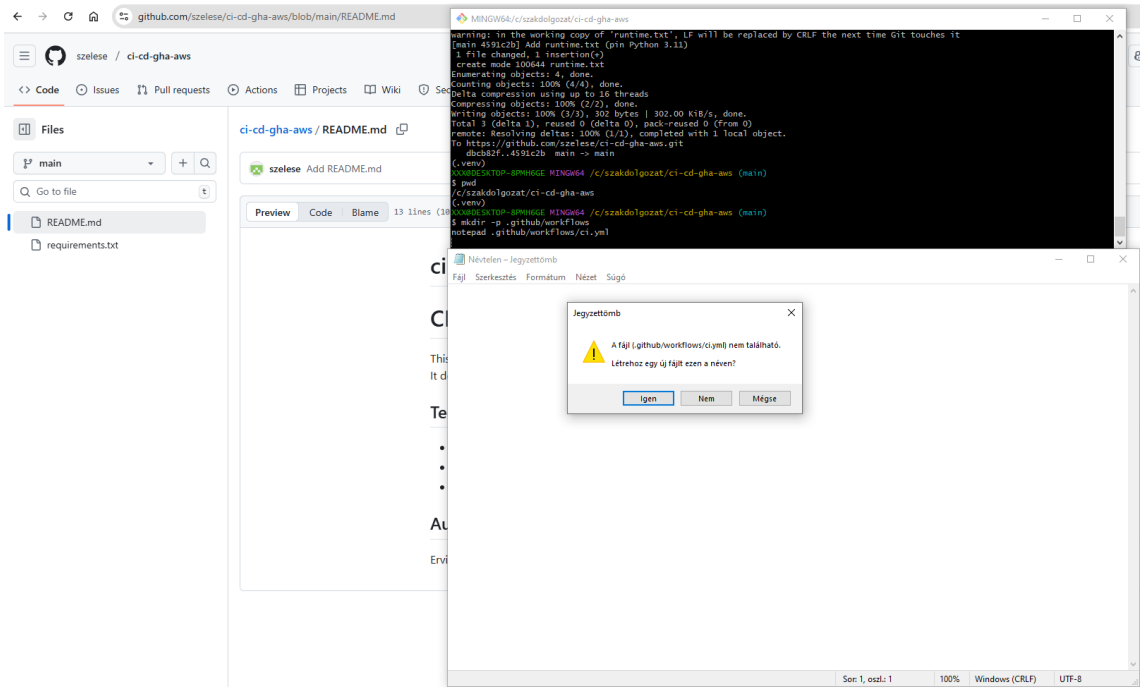
```
printf ".venv/\n__pycache__/\n*.pyc\n.env\n*.sqlite3\n" > .gitignore
cat .gitignore
git status
```

Step 9: Creating runtime.txt with Precise Content, Verification, and Commit/Push



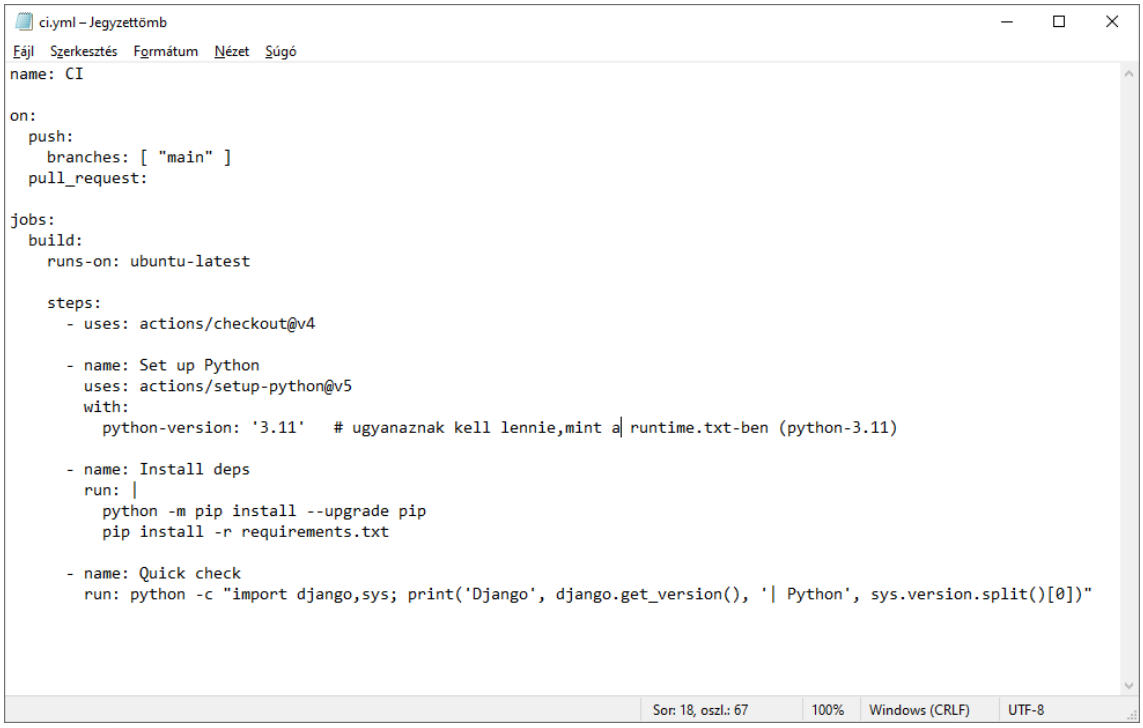

```
$ echo "python-3.11" > runtime.txt
cat runtime.txt
git add runtime.txt
git commit -m "Add runtime.txt (pin Python 3.11)"
git push
python-3.11
```

Step 10: Workflow Folder and ci.yml Creation



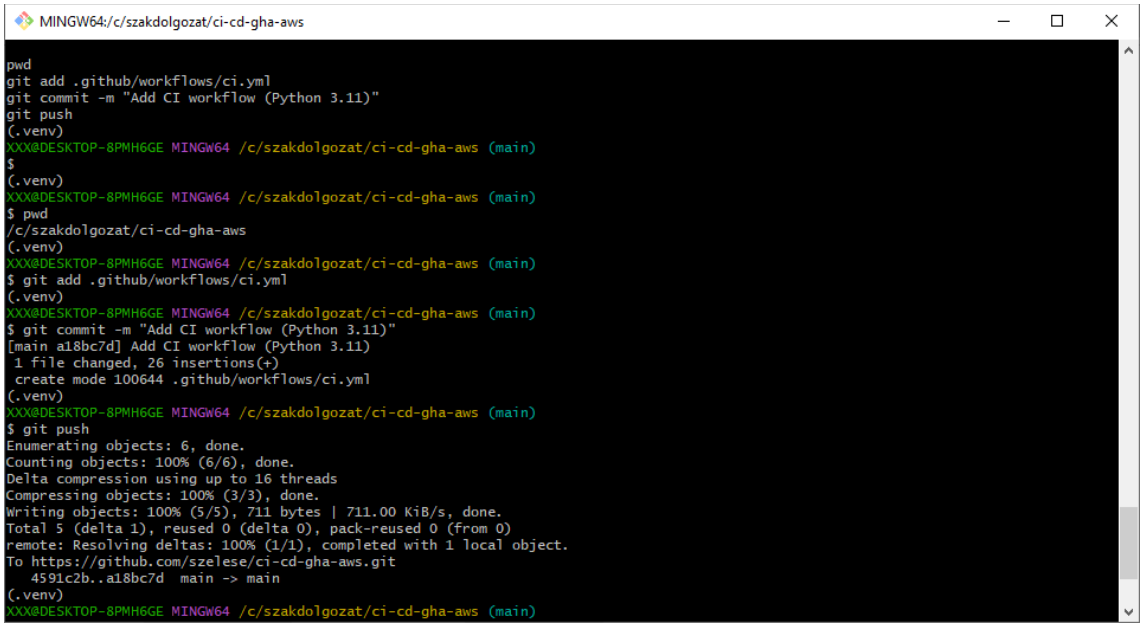

```
$ mkdir -p .github/workflows
notepad .github/workflows/ci.yml
```

Step 11: Content of the ci.yml File



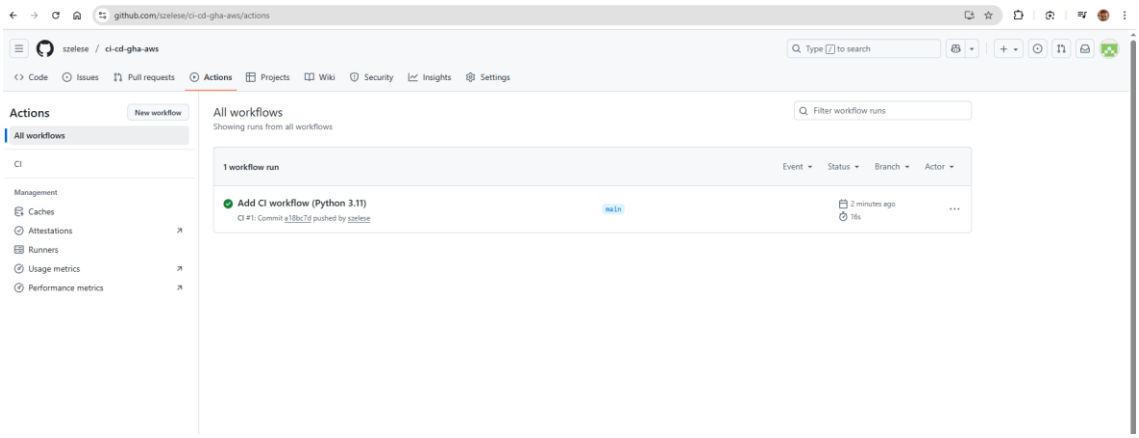
The runtime.txt file ensures that Elastic Beanstalk (EB) executes the project using the exact same Python version used during local development.

Step 12: commit&push



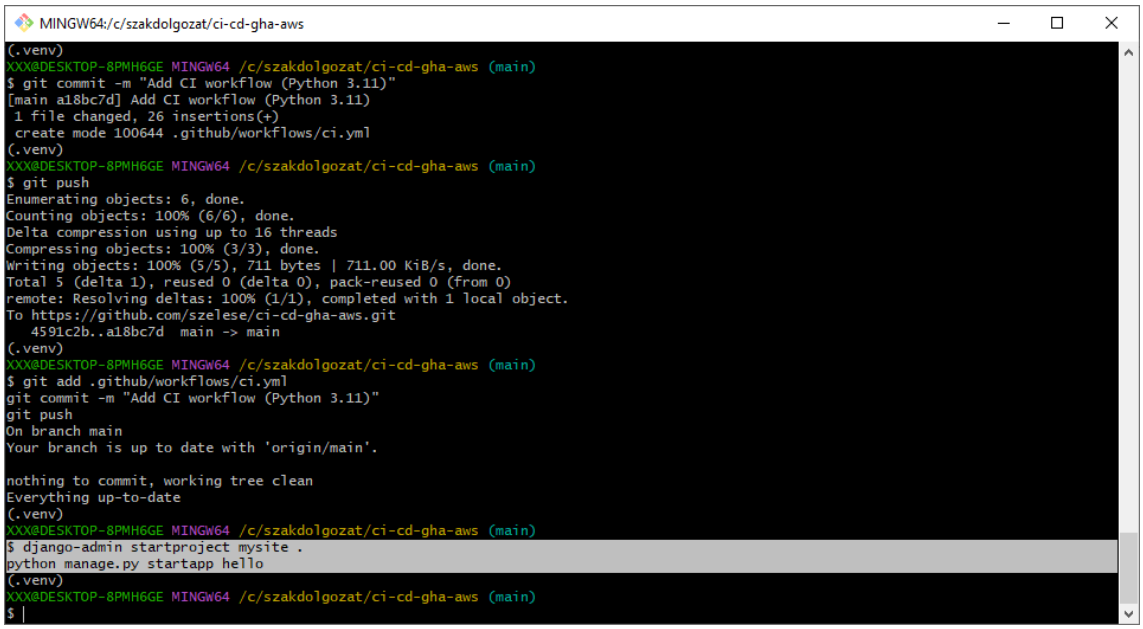

```
commit és push ci.yml -t
git add .github/workflows/ci.yml
git commit -m „Add CI workflow (Python 3.11)”
```

Step 13: GitHub Verification



The "CI" run should appear; a green checkmark indicates the process completed successfully.

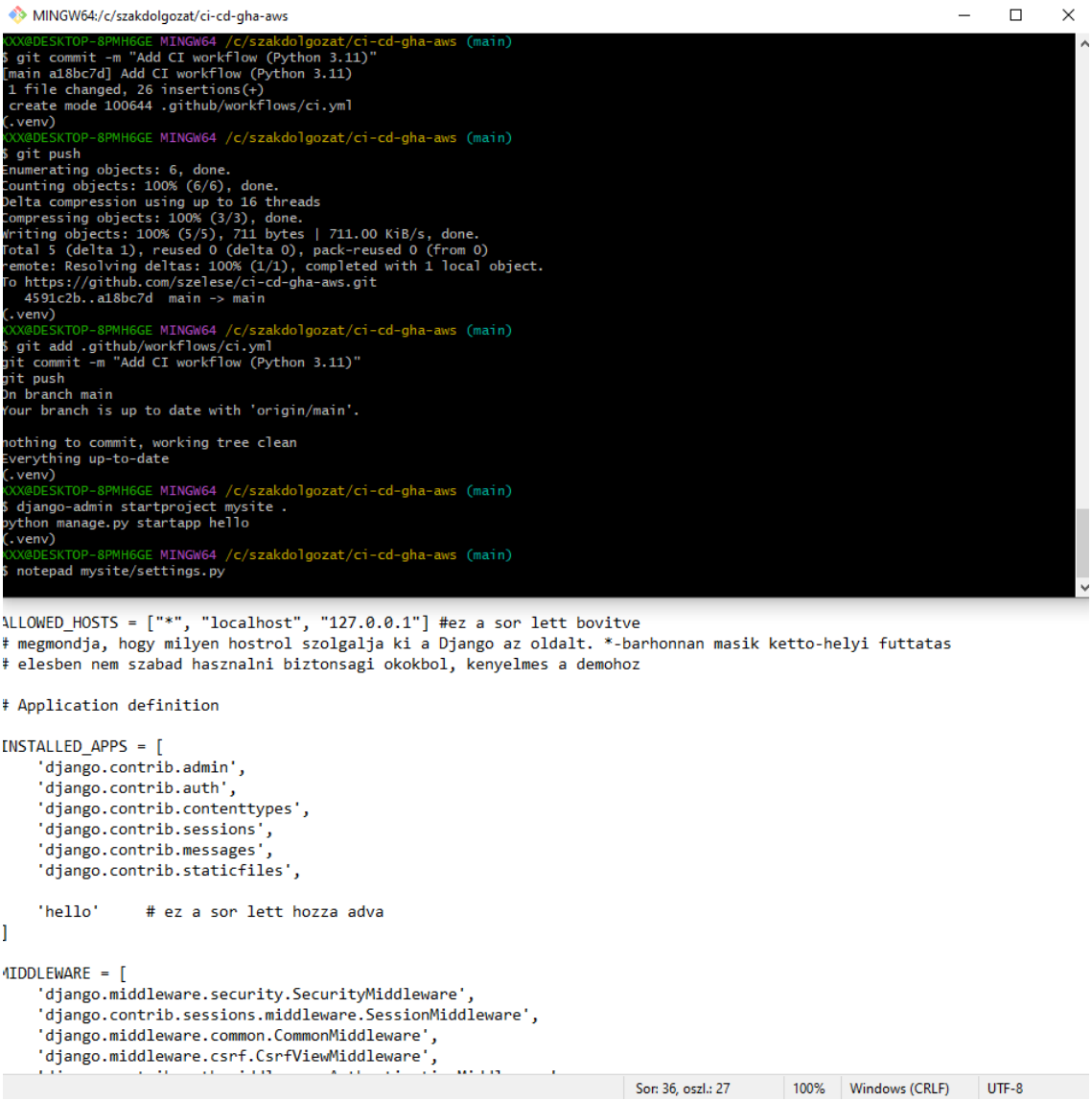
Step 14: Project and App Creation



Initialize the Django project named mysite in the current directory:

```
django-admin startproject mysite .
python manage.py startapp hello
```

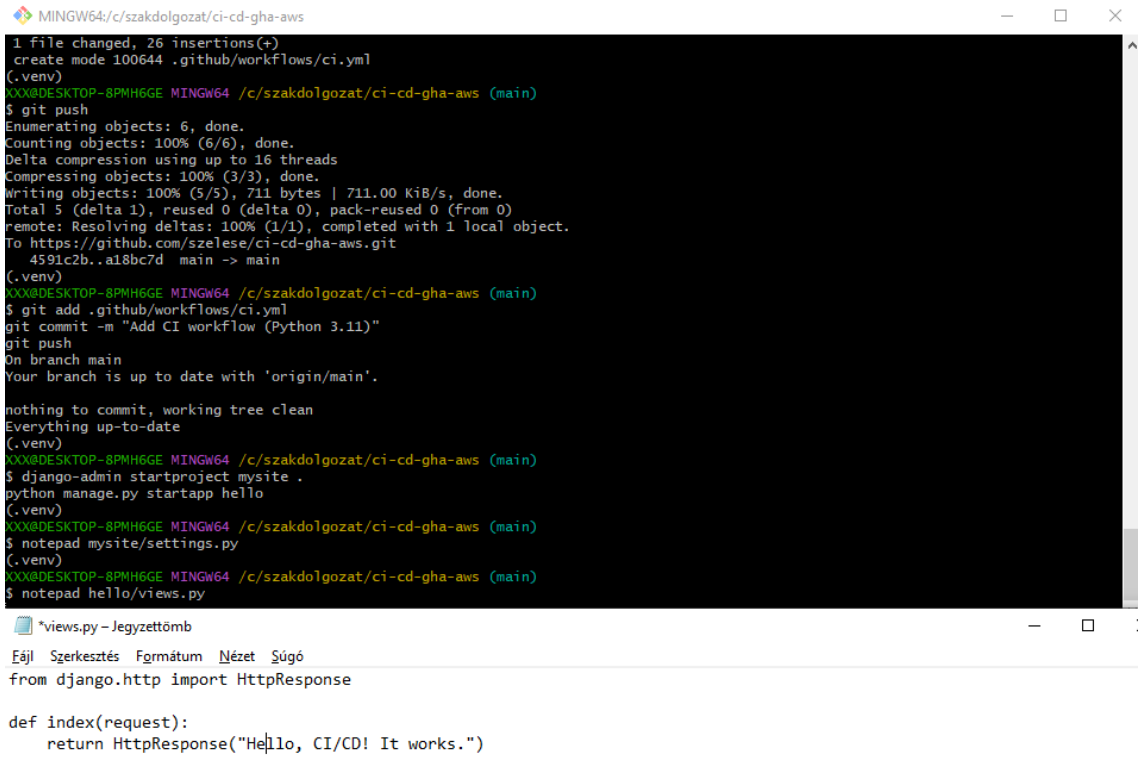
Step 15: App Registration and Demo Permissions




```
notepad mysite/settings.py
INSTALLED_APPS = [
    # ...
    'hello',
]
és
ALLOWED_HOSTS = ["*", "localhost", "127.0.0.1"]
```

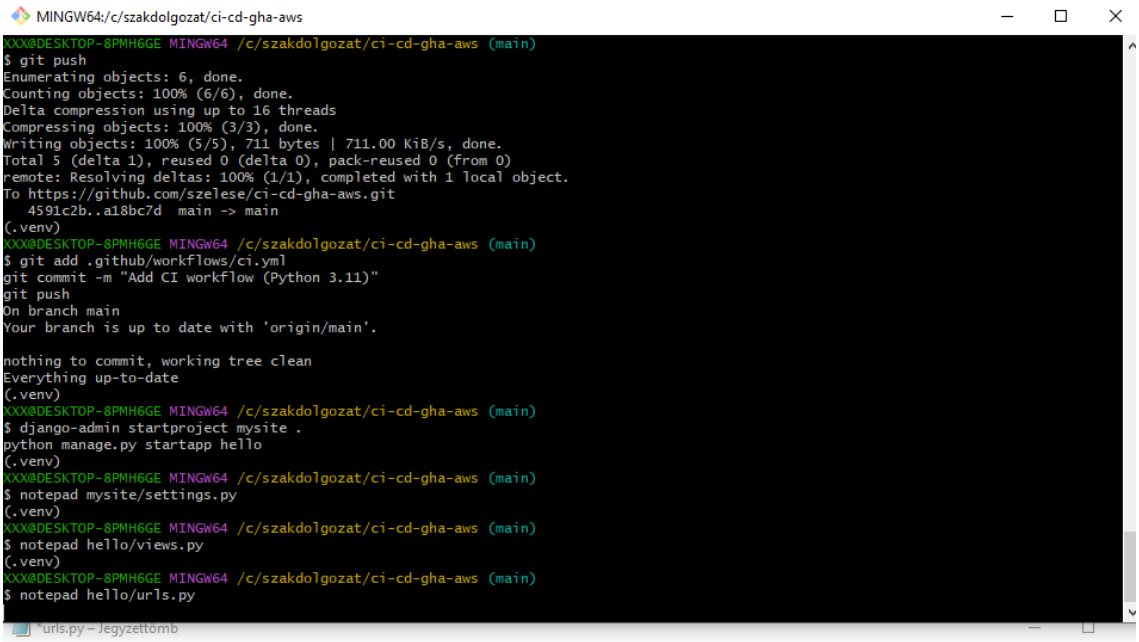
Security Warning: The ALLOWED_HOSTS = ['*'] setting is strictly for initial testing; in a production environment, specific domains must be defined for security reasons !!!

Step 16: Writing a Minimal View




```
notepad hello/views.py # views.py megnyitása után:
from django.http import HttpResponse
def index(request):
    return HttpResponse("Hello, CI/CD! It works.")
```

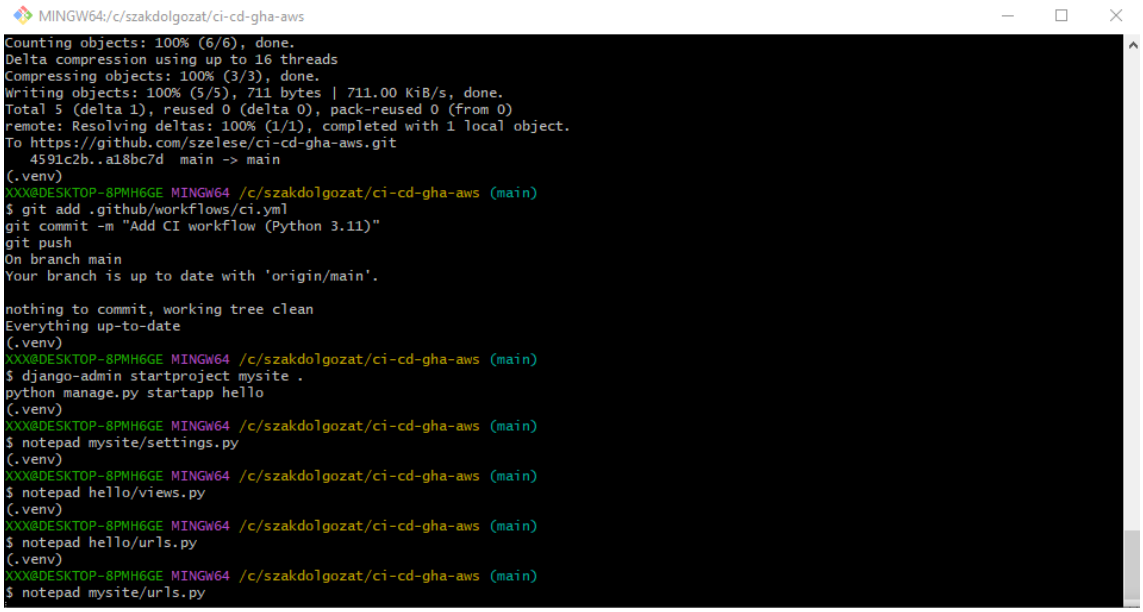
Step 17: Connecting the URL to the Hello Application

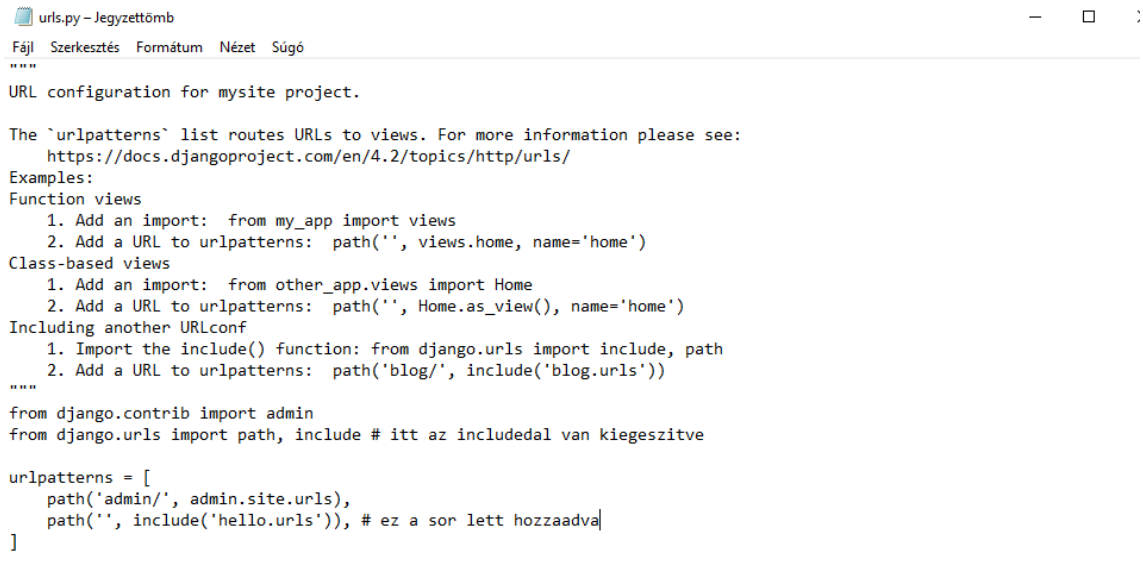


Create and open the app-level URL file: notepad hello/urls.py.

```
from django.urls import path
from . import views
urlpatterns = [    path("", views.index, name="index"),]
```

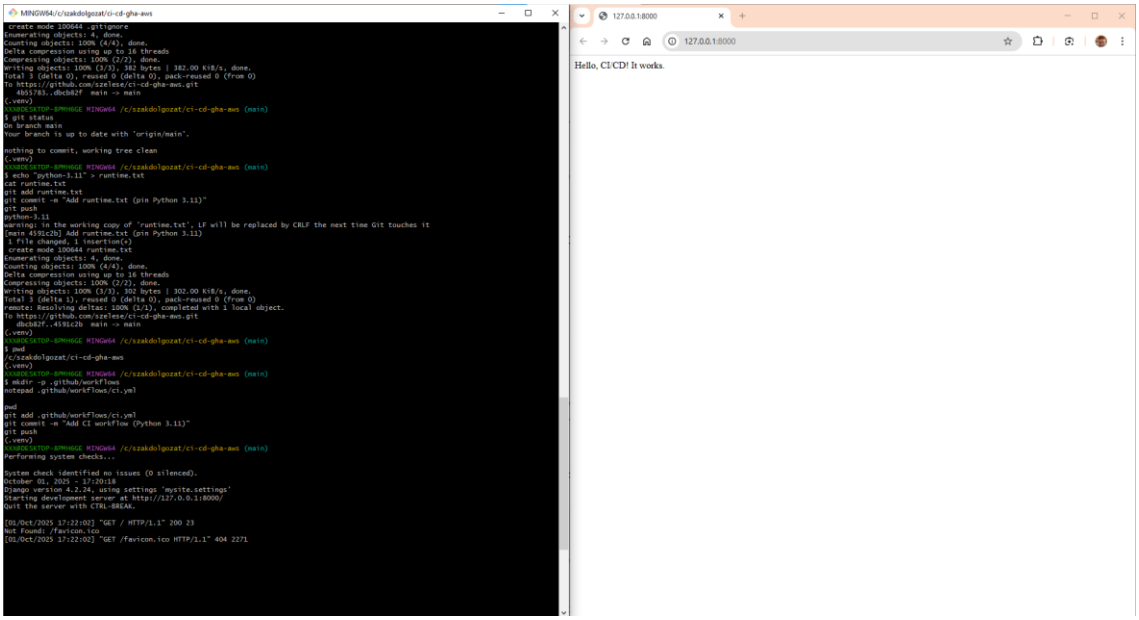
Step 18: Connecting App URLs to the Main Project





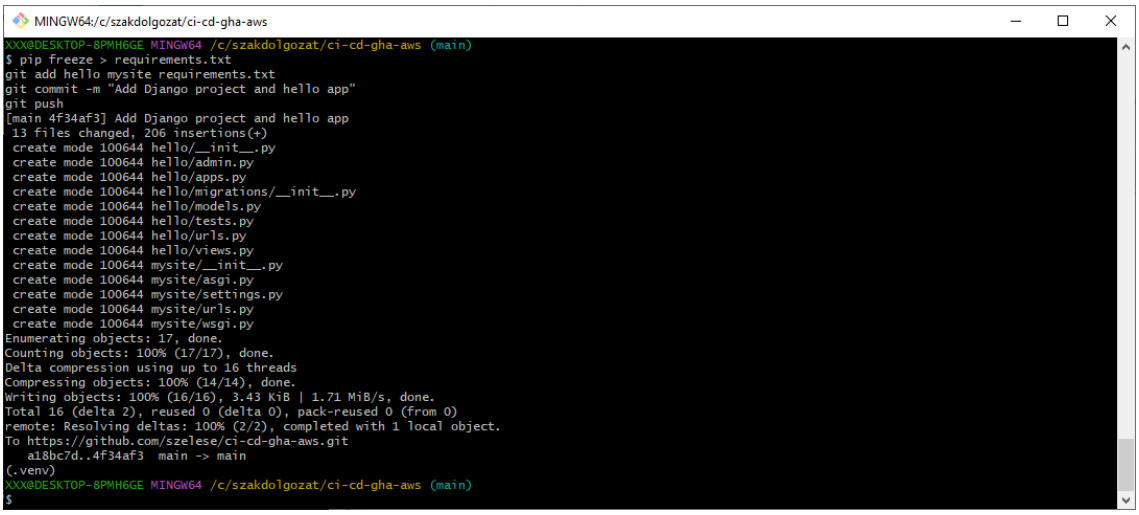

```
notepad mysite/urls.py # megnyitas és modositas:
from django.contrib import admin
from django.urls import path, include
urlpatterns = [    path('admin/', admin.site.urls),
    path('', include('hello.urls')),]
```

Step 19: Database Initialization and Local Run Test



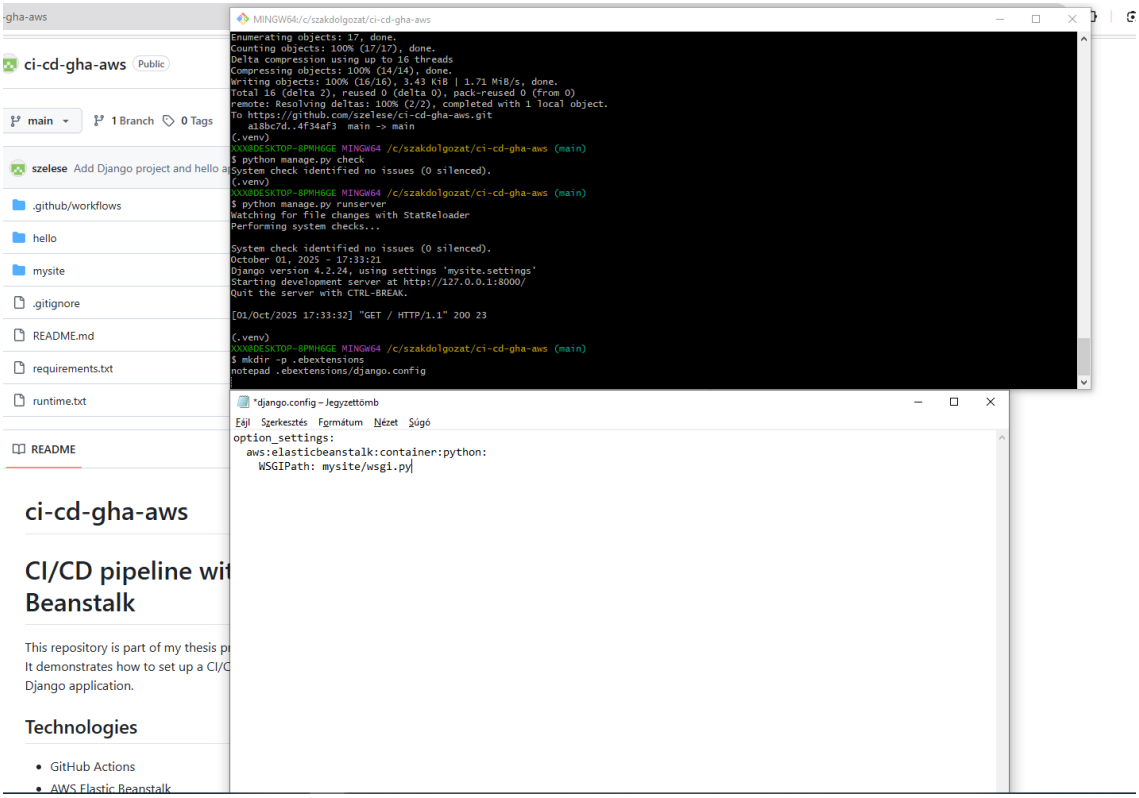

```
python manage.py migrate # db inicializálása
python manage.py runserver
# utána a böngészőben tesztelés : http://127.0.0.1:8000/
```

Step 20: Updating Requirements and Pushing to GitHub



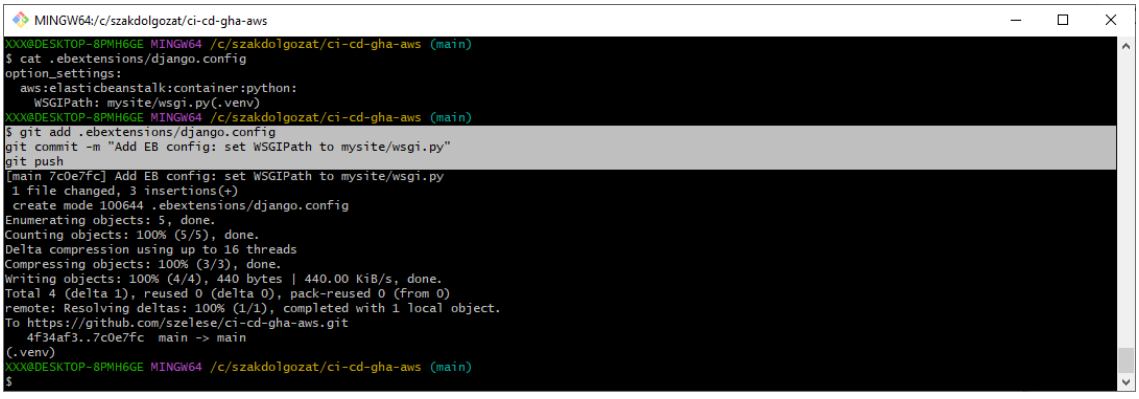

```
pip freeze > requirements.txt
git add hello mysite requirements.txt
git commit -m "Add Django project and hello app"
git push
```

Step 21: Configuration for Elastic Beanstalk (EB)




```
mkdir -p .ebextensions
notepad .ebextensions/django.config
django.config file tartalma: # spaces!!!
option_settings:
  aws:elasticbeanstalk:container:python:
    WSGIPath: mysite/wsgi.py
```

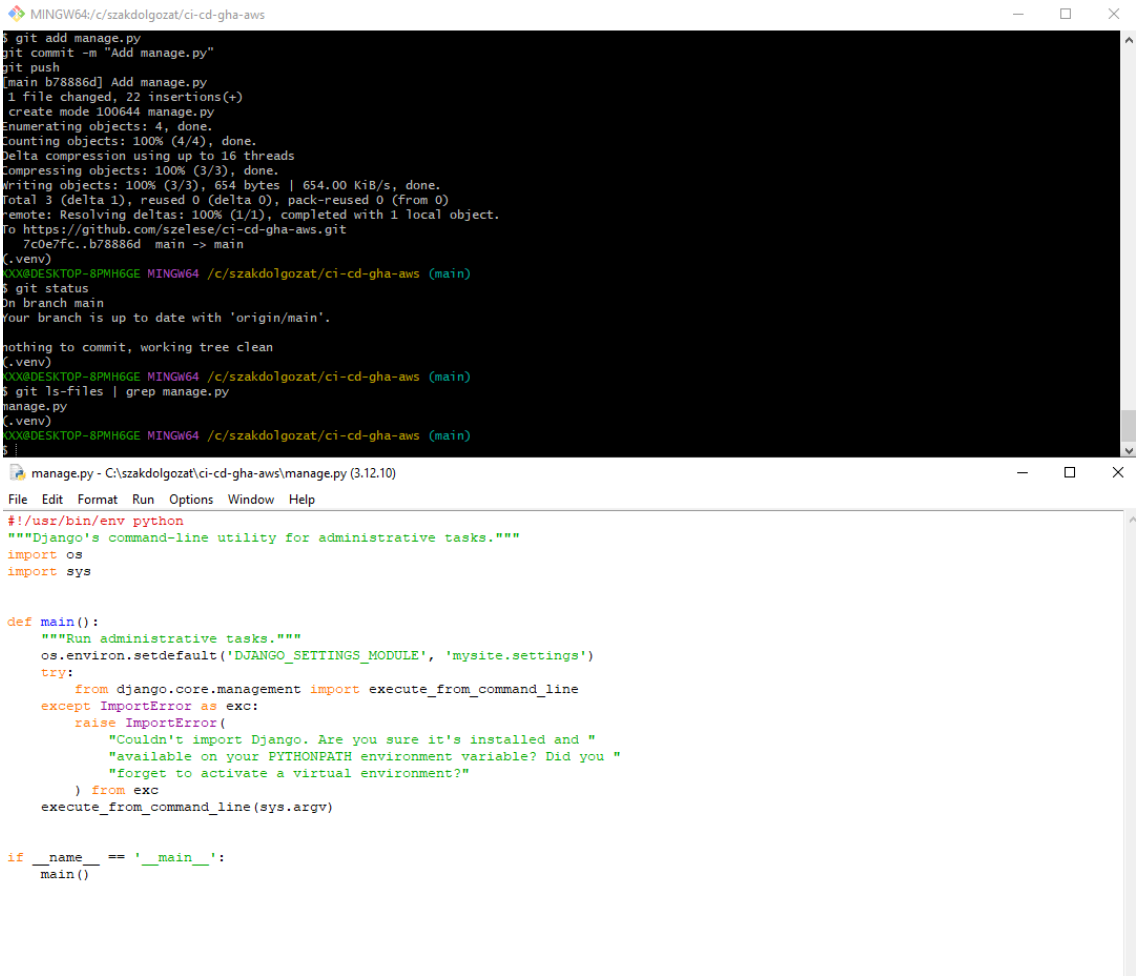
Step 22: Verification, Commit & Push



commit&push után még egy ellenőrzés a githubon, hogy létrejött a file ezzel a tartalommal

```
cat .ebextensions/django.config
git add .ebextensions/django.config
git commit -m "Add EB config: set WSGIPath to mysite/wsgi.py"
git push
```

Step 23: Setting up manage.py



Technical Note: While manage.py is not strictly mandatory for running on EB (as it uses WSGI), it is essential for local development, migrations, collectstatic, and future automation.

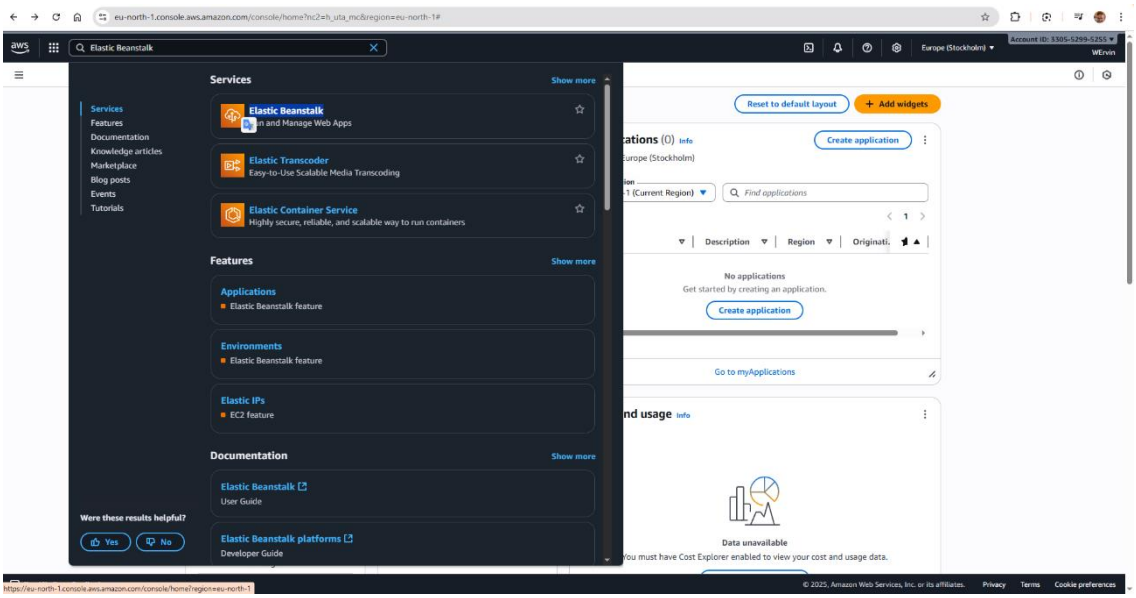
```
notepad manage.py
git add manage.py
git commit -m "Add manage.py"
git push
```

Step 24: Creating the deploy.zip Bundle

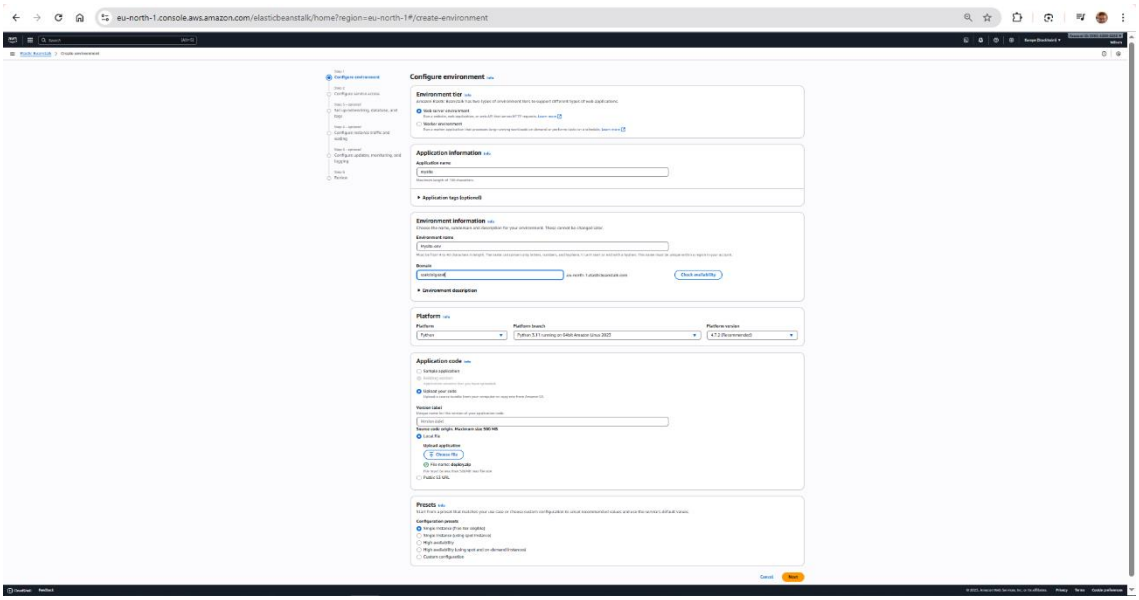

```
zip -r deploy.zip . -x ".git/*" ".venv/*" "__pycache__/*" "*.pyc"
```

Note: The zip tool from MSYS2 was used to handle this via command line, but it can also be done via File Explorer as long as .git and .venv are excluded.

25. lépés: AWS search / Elastic Beanstalk/Create Application



Step 26: AWS Environment Configuration



Enviromment tier: Web Server Enviroment # default

Application information: mysite

enviroment information:

enviroment name: Mysite.env

Domain: szakdolgozat # or any unique, valid string

Platform: Python

platform branch: Python 3.11 # matching the runtime.txt specification

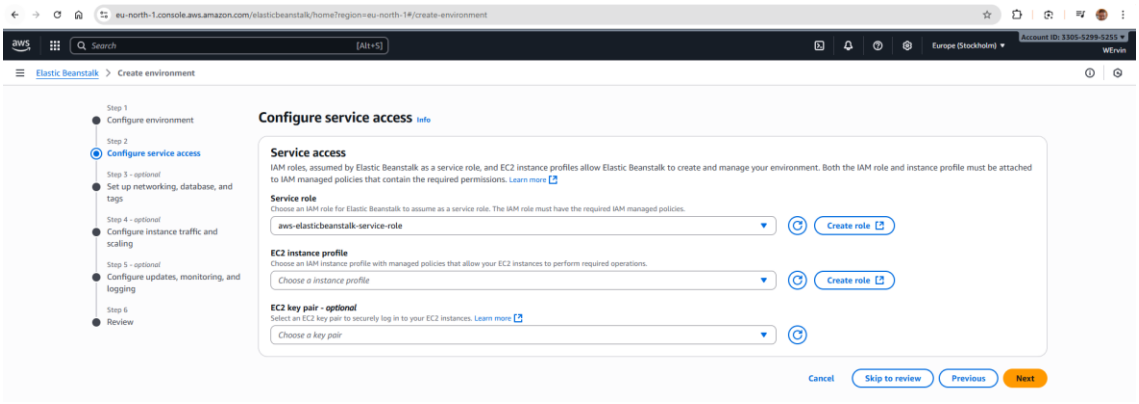
Version label: mysite-v1

Application code: Upload your code ->local file-> deploy.zip # the bundle generated in Step 24

Presets:

Single instance (free tier eligible) # which is sufficient for demonstration purposes

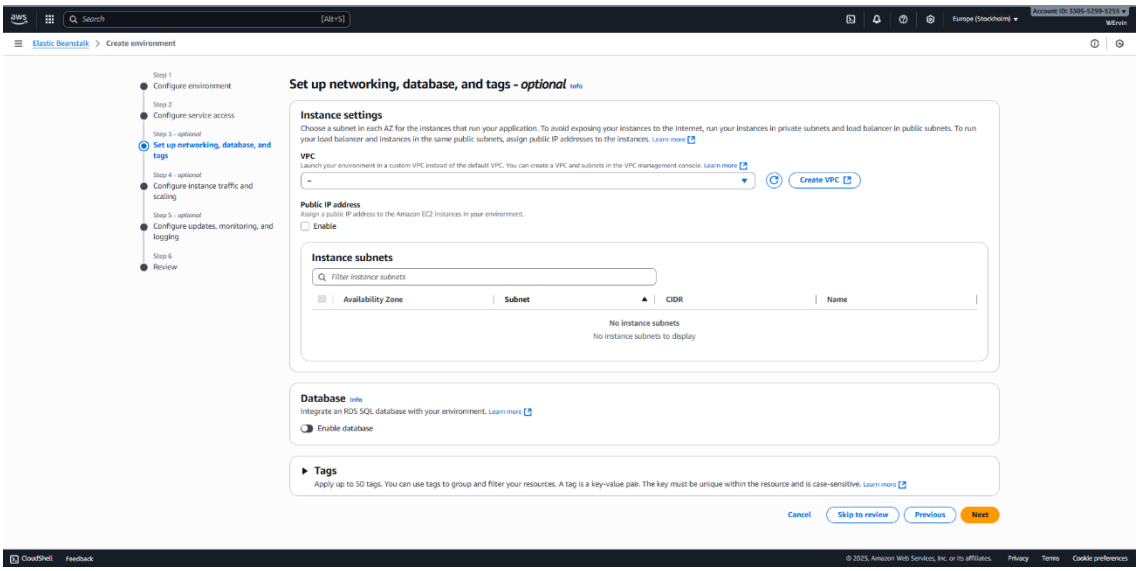
Step 27: Step 2 - Configure Service Access



Service role: For the service role, approve the automatically populated roles and select the specified IAM roles. EC2 key pair: The EC2 key pair (optional) field can be left empty.

Technical Note: SSH settings only need to be configured if you intend to control the server directly from a console.

Step 28: Step 3 - Networking, Database, and Tags (Optional)



VPC: Leave at the default - value; do not create a new VPC or select a subnet.

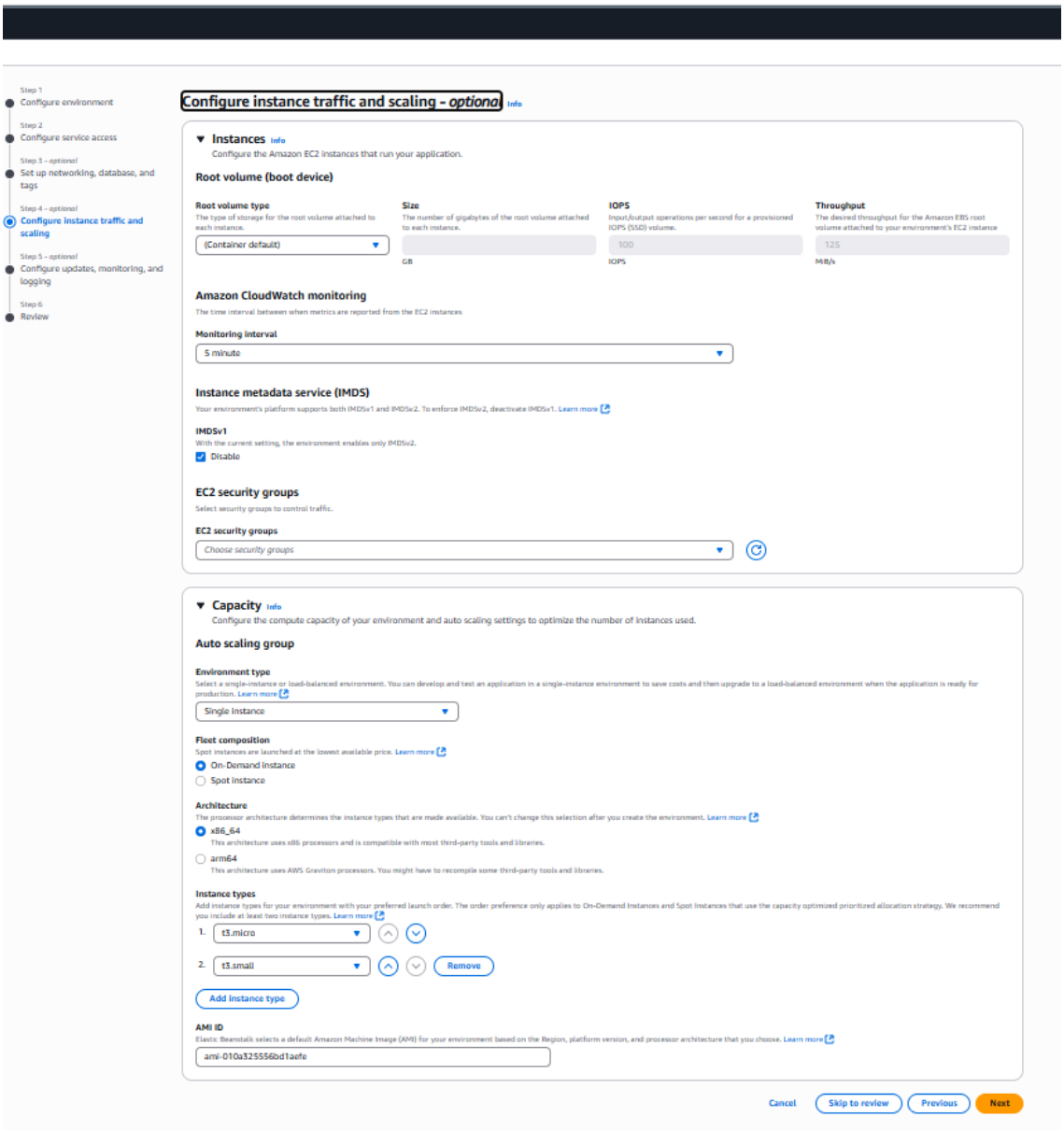
Public IP address: This can remain disabled; the Single instance preset makes the instance public by default.

Adjusting access: If the instance is not accessible externally later, it can be enabled via Environment → Configuration → Instances → Public IP: Enable.

Database: Do not enable (RDS is not used in this phase).

Tags: Optional; leave at default settings.

Step 29: Step 4 - Configure Instance Traffic and Scaling (Optional)



Root volume: Container default

Monitoring interval: 5 minutes. IMDSv1: Disable (ensures only IMDSv2 is permitted)

EC2 security groups: leave as the default EB groups.

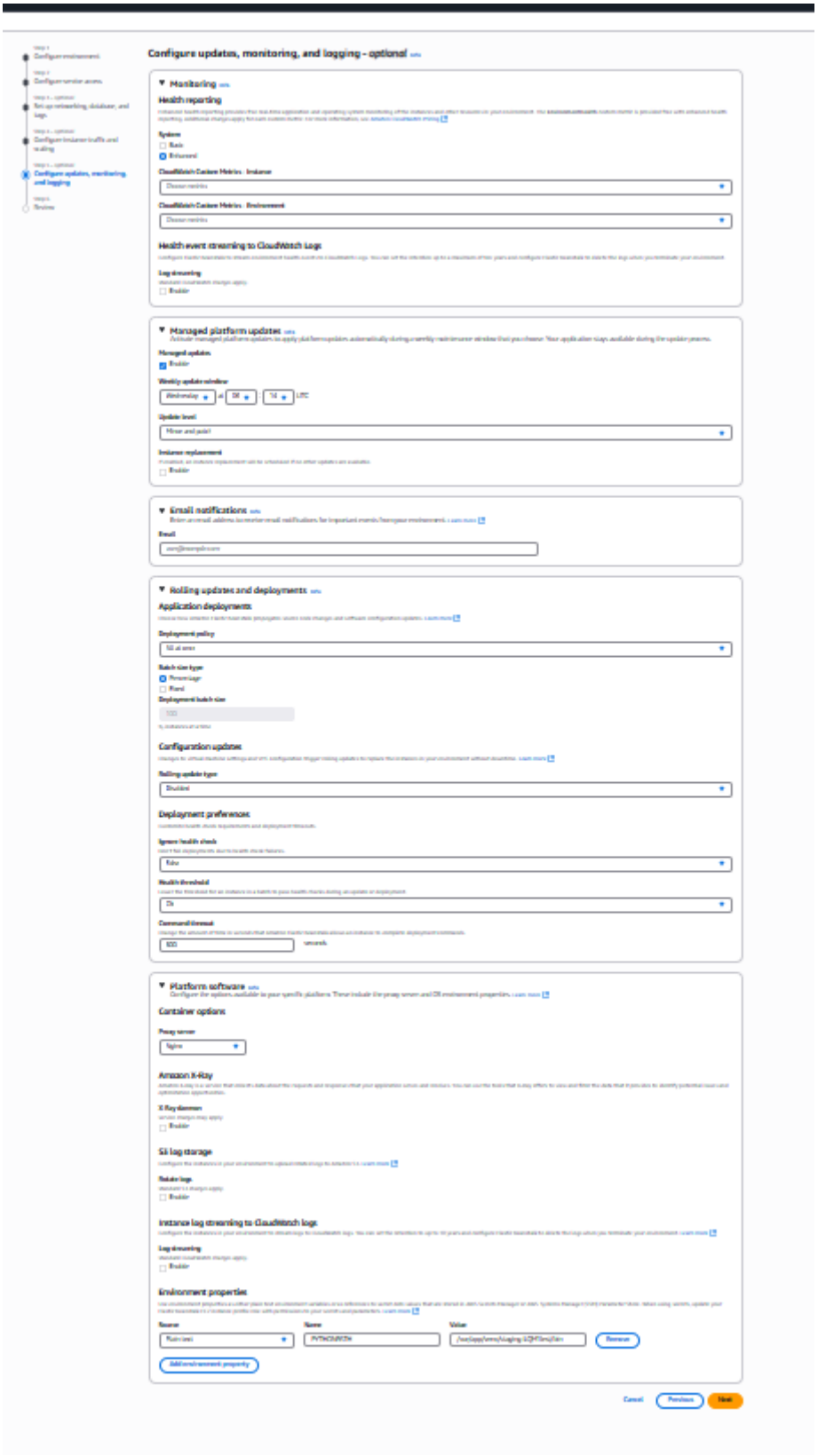
Capacity: Environment type: Single instance

Fleet composition: On-Demand Architecture: x86_64

Instance types: 1) t3.micro 2) (opció) t3.small

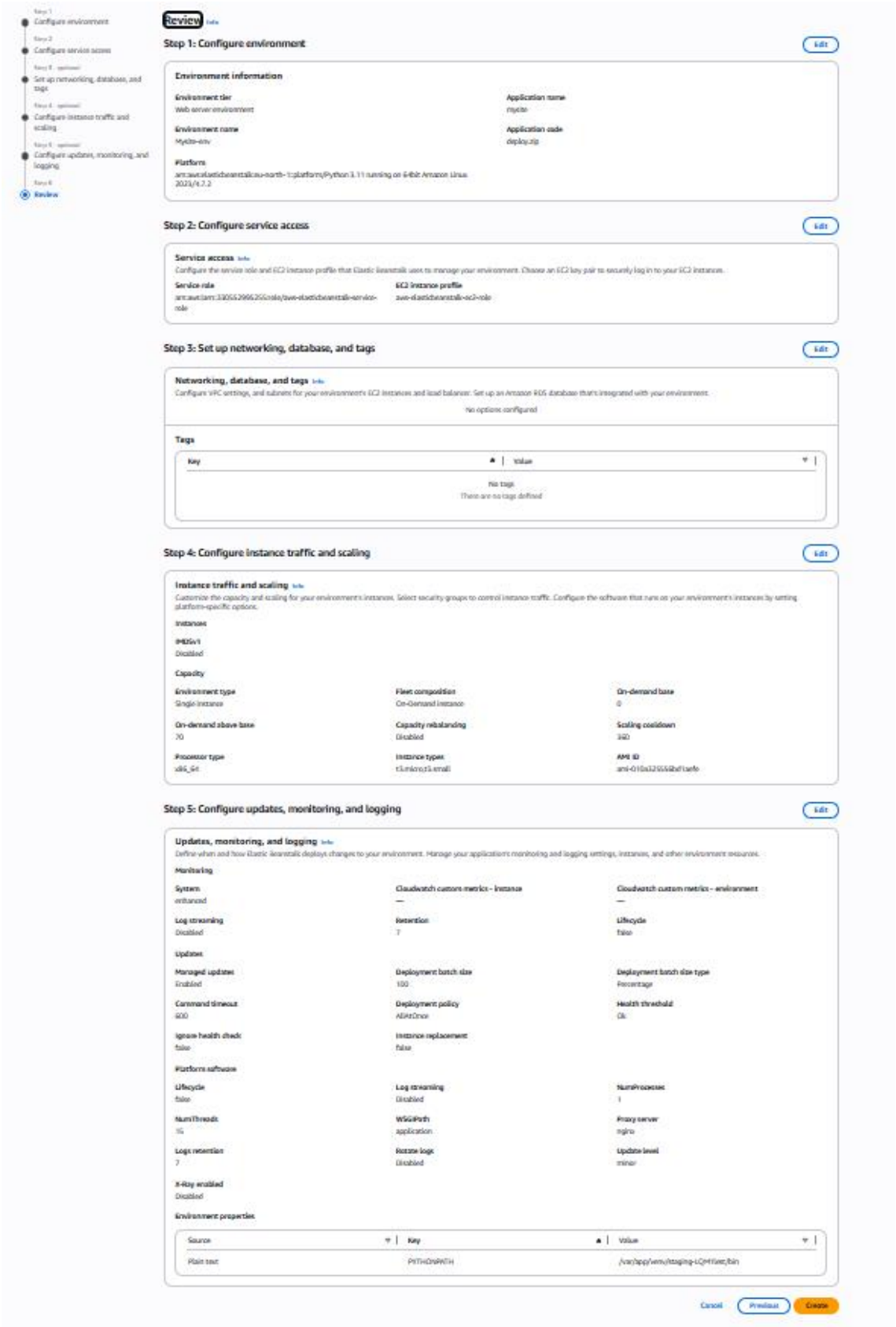
AMI ID: Leave at the default value.

Step 30: Step 5 - Updates, Monitoring, and Logging (Optional)

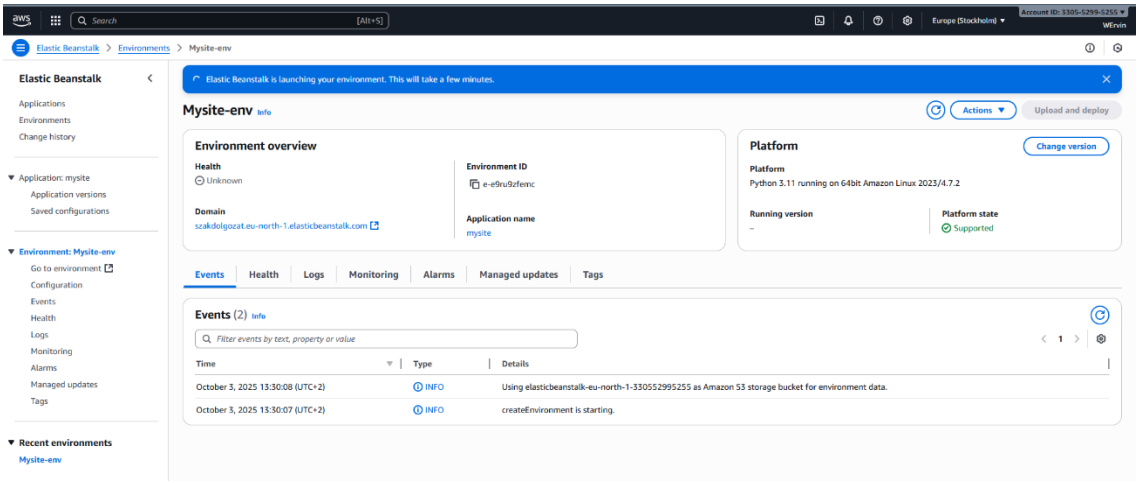


Standard settings: The default values are suitable for this project

Step 31: Step 6 - Review and Create

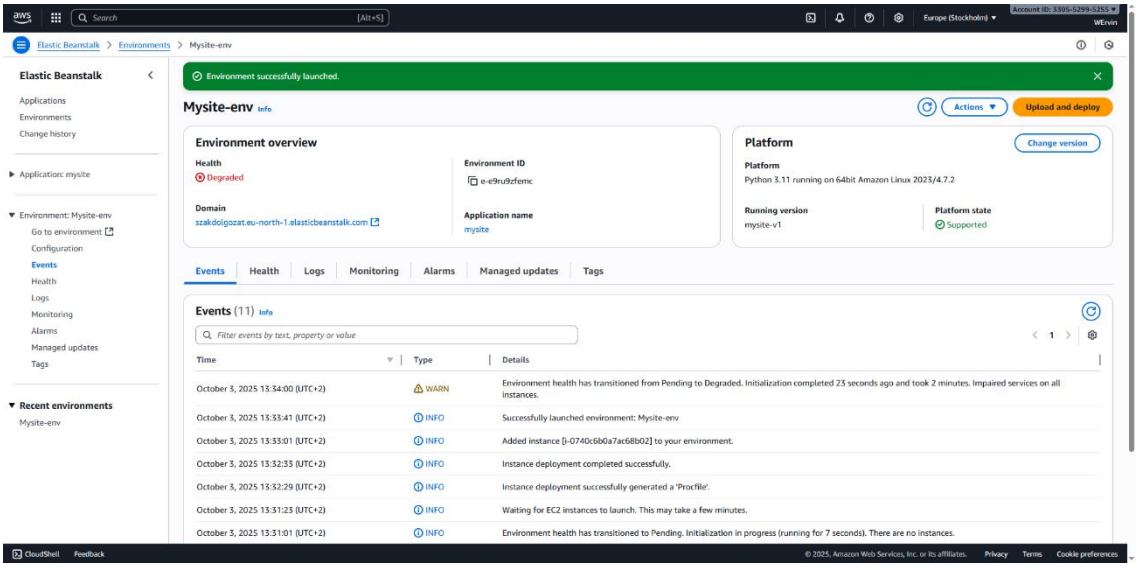


Step 32: EB Environment Launch and Verification



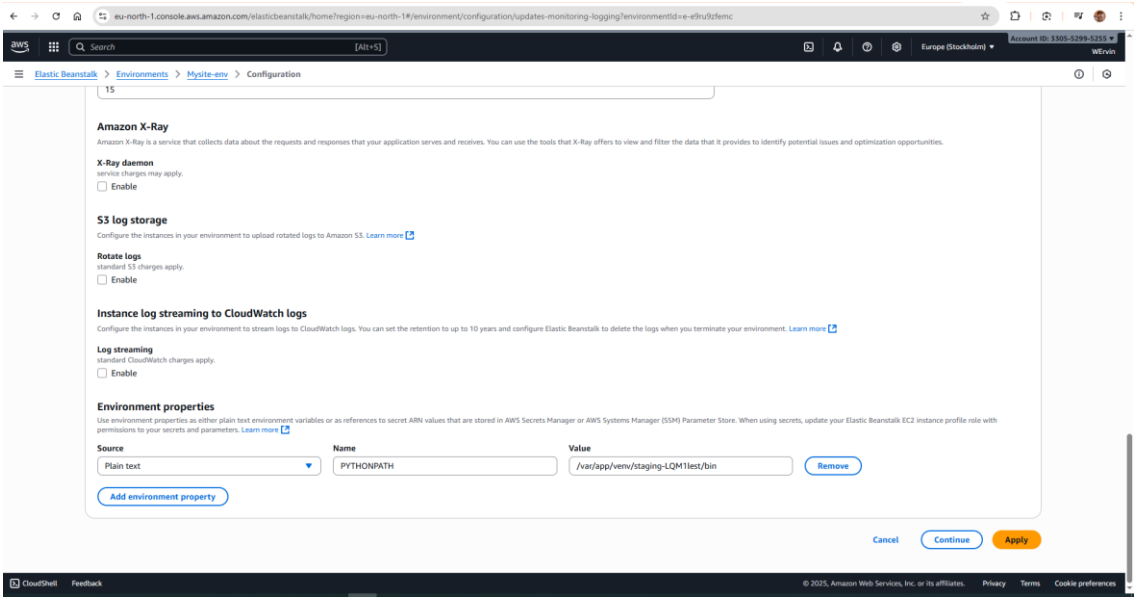
When the Health tab reaches a green "OK" status, the Domain link becomes immediately accessible.

Step 33: Health Check Troubleshooting



This status almost always indicates that the Elastic Beanstalk (EB) health check is receiving a 400, 404, or 5xx error instead of a 200 OK response for the root (/) URL.

Step 34: Adding Environment Properties



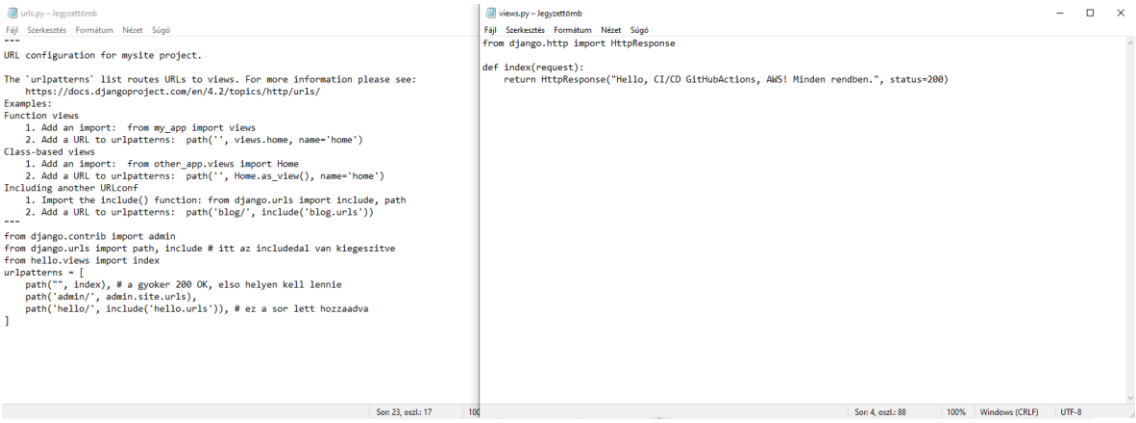
Elastic Beanstalk > Environments > Mysite_env > Configuration > (bottom) Environment properties > Add

Key: DJANGO_SETTINGS_MODULE → Value: mysite.settings

(opctional) Key: DEBUG → Value: False

(optional) Key: SECRET_KEY → Value: A12345678b

Step 35: Local File Modification for Health Check



```
"""
URL configuration for mysite project.

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/4.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path, include # itt az includedal van kiegeszitve
from hello.views import index
urlpatterns = [
    path("", index), # a gyoker 200 OK, also helyen kell lennie
    path('admin/', admin.site.urls),
    path('hello/', include('hello.urls')), # ez a sor lett hozzaadva
]
```

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello, CI/CD GitHubActions, AWS! Minden rendben.", status=200)
```

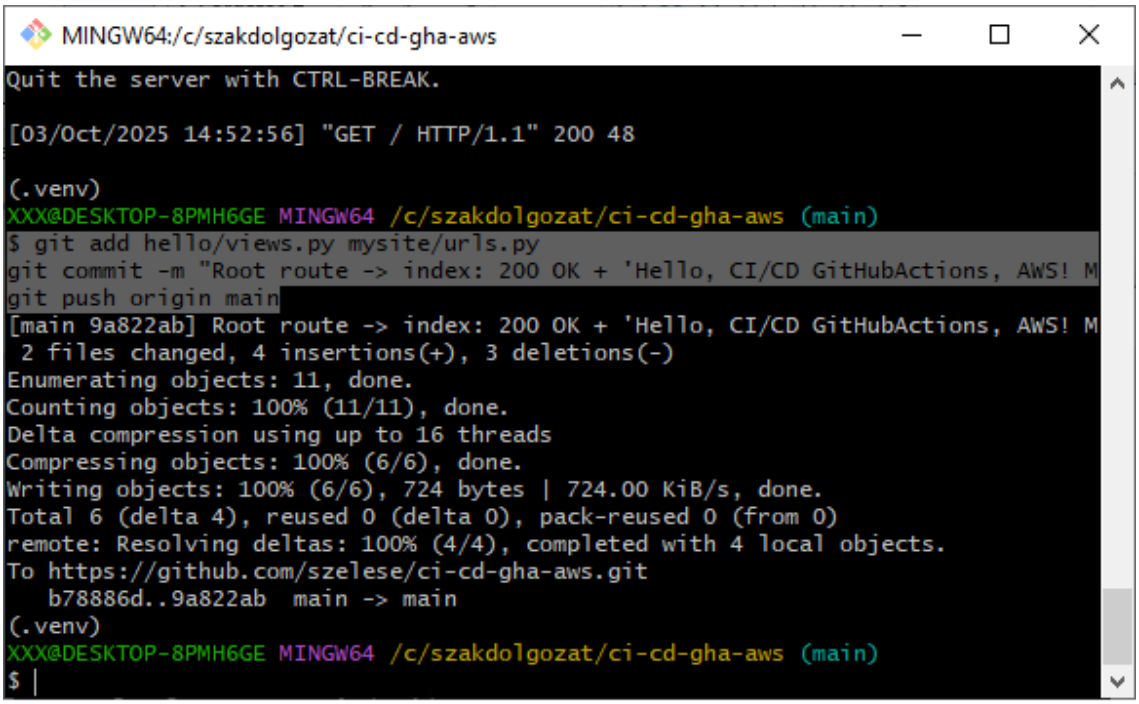
hello/views.py :

```
from django.http import HttpResponse
def index(request):
    return HttpResponse("Hello, CI/CD GitHubActions, AWS! ALL OK.", status=200)
```

mysite/urls.py :

```
path("", index), # a gyoker 200 OK, elso helyen kell lennie
python manage.py runserver # local test if everything is OK:
zip -r deploy.zip . \
  -x '.git/*' '.venv/*' '__pycache__/*' '*.pyc' 'deploy.zip'
```

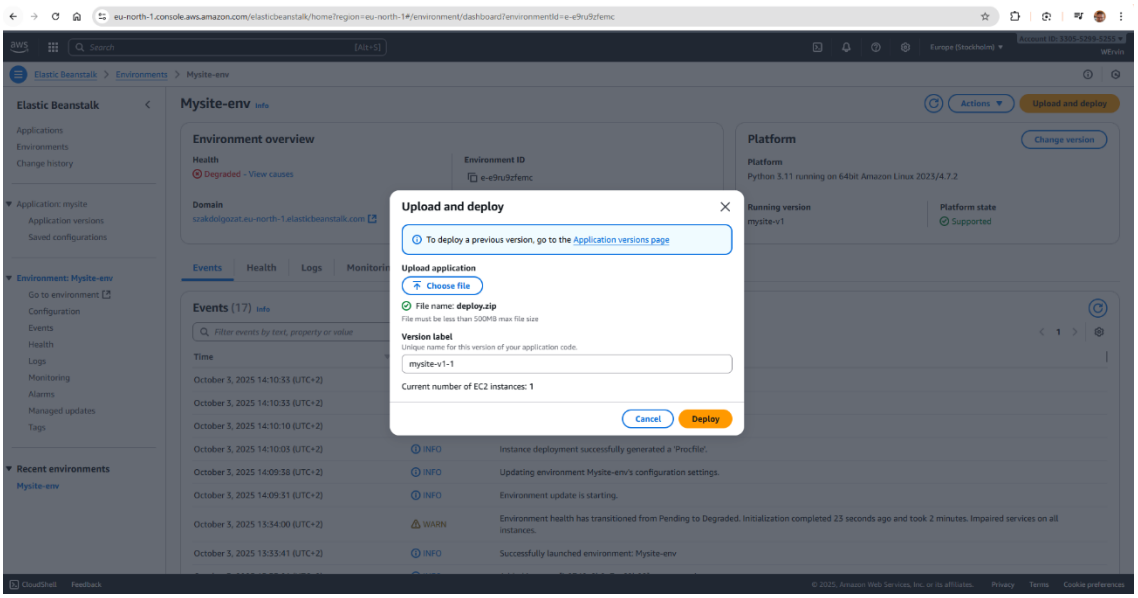
Step 36: GitBash Commit and Push




```
git add hello/views.py mysite/urls.py
git commit -m "Root route -> index: 200 OK + 'Hello, CI/CD GitHubActions, AWS! Minden rendben.'"
git push origin main
```

Technical Note: Frequent small commits and pushes are central to the CI/CD process. Every git push command automatically triggers the GitHub Actions CI process (ci.yml), ensuring the system tests itself immediately after every modification.

Step 37: Manual Upload to Elastic Beanstalk



AWS Console → Elastic Beanstalk → Environments → Mysite-env

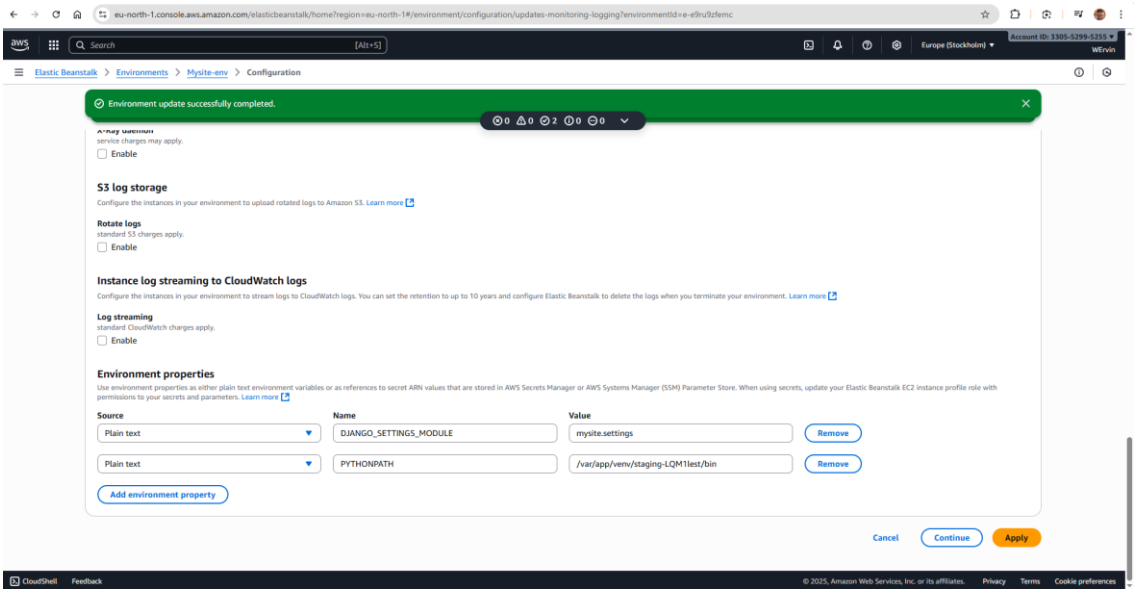
Top right corner: Upload and deploy.

Choose file → select the updated deploy.zip.

Version label: pl. mysite-v1-1 → Deploy.

Step 38: Gunicorn Configuration and Troubleshooting

Failed Attempt: Modifying /var/app/current due to PYTHONPATH issues did not resolve the error

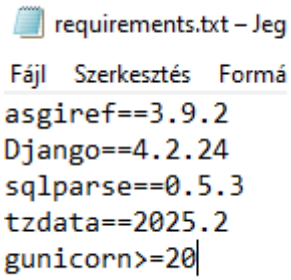


Log file analysis revealed an ImportError: No module named 'mysite/wsgi'

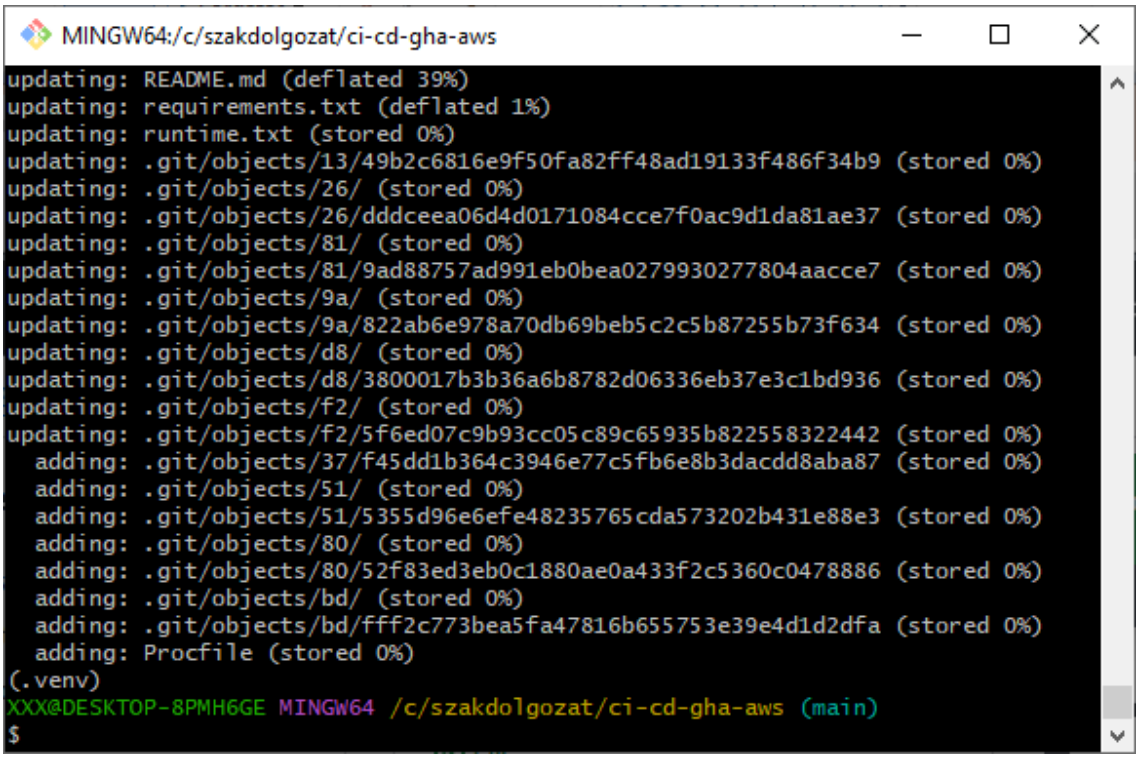
Diagnosis: Gunicorn requires module names to use dots rather than slashes. The correct entry point is mysite.wsgi:application.

Currently, EB attempts to launch it as mysite/wsgi, which causes a 502 Bad Gateway error. These troubleshooting steps demonstrate practical error handling and testing.

Step 39: Updating Requirements

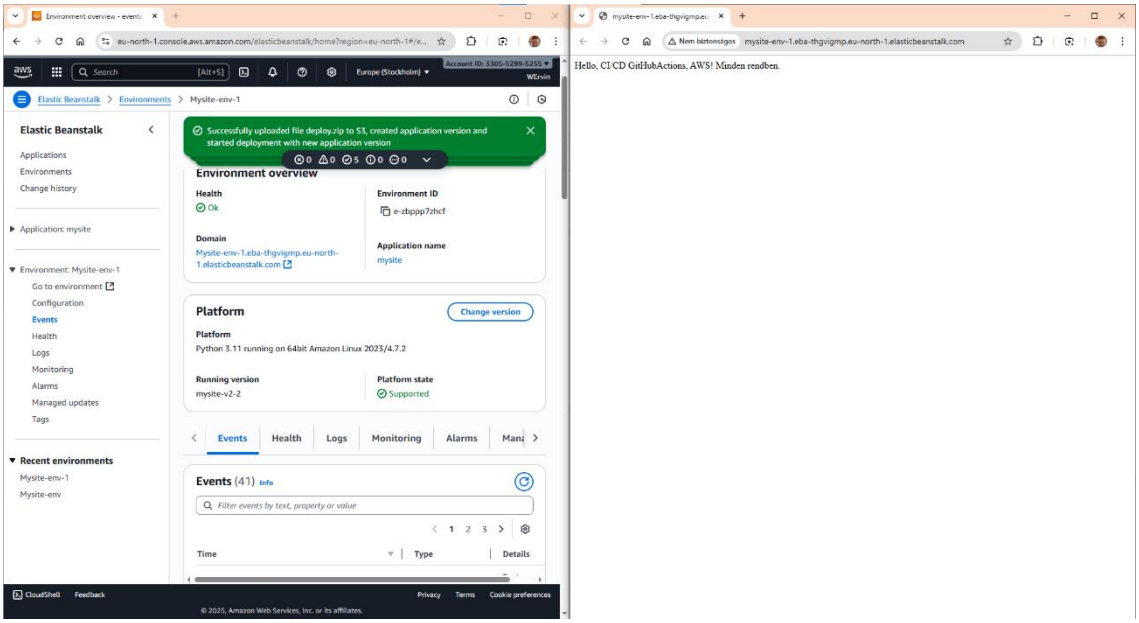


Step 40: Creating the Procfile and Finalizing Push




```
printf "web: gunicorn mysite.wsgi:application --bind 127.0.0.1:8000\n" > Procfile
git add Procfile hello/views.py mysite/urls.py requirements.txt
git commit -m "Add Procfile (Gunicorn) + root health endpoint"
git push origin main
```

Step 41: Deployment Verification

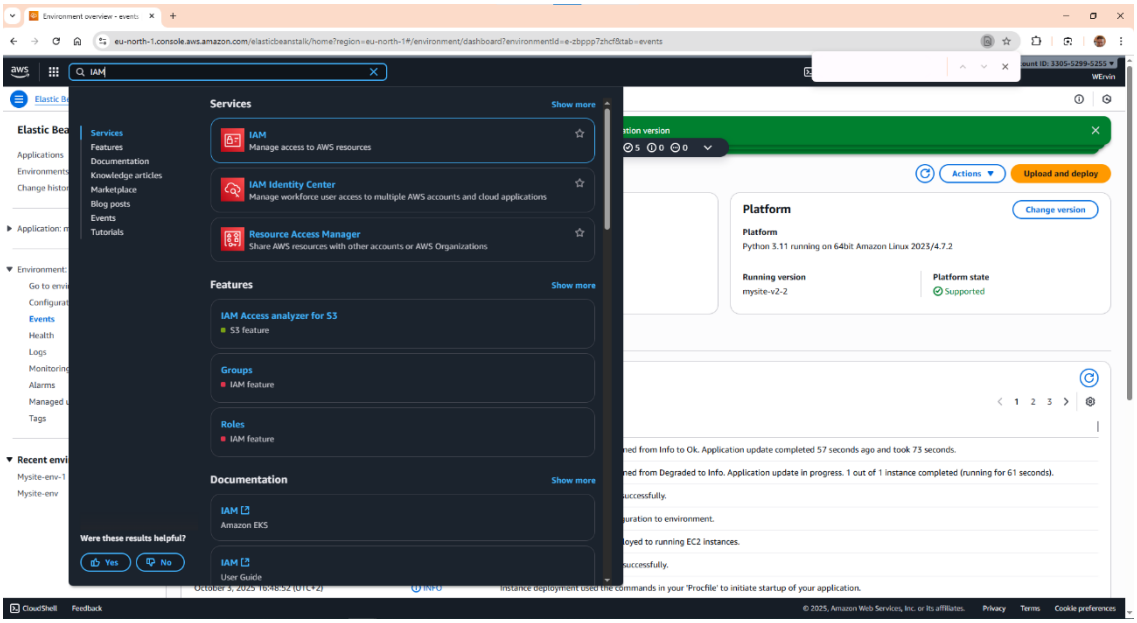


The deployment on AWS Elastic Beanstalk is successful; the mysite-env-1 environment status is OK.

Running version: mysite-v2-2 (Python 3.11, AL2023).

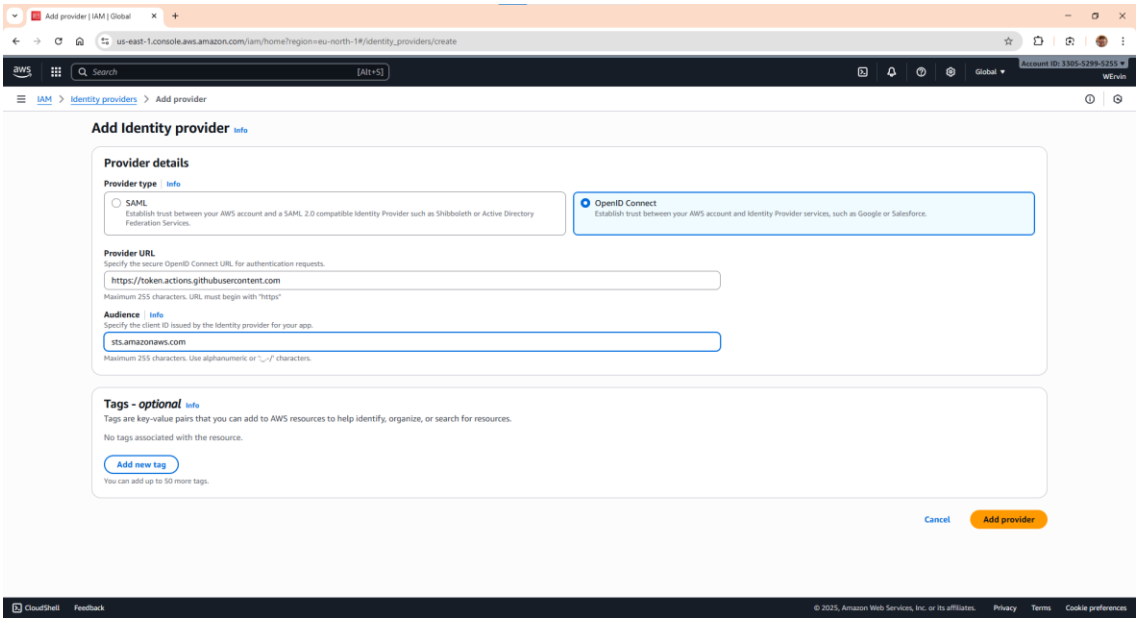
The domain endpoint now returns a 200 OK response: ‘Hello, CI/CD GitHubActions, AWS! All OK!’

Step 42: Opening IAM



Type IAM (Identity and Access Management) into the search bar and sign in

Step 43: Adding an Identity Provider



Identify provider -> Add provider

OPENID connect

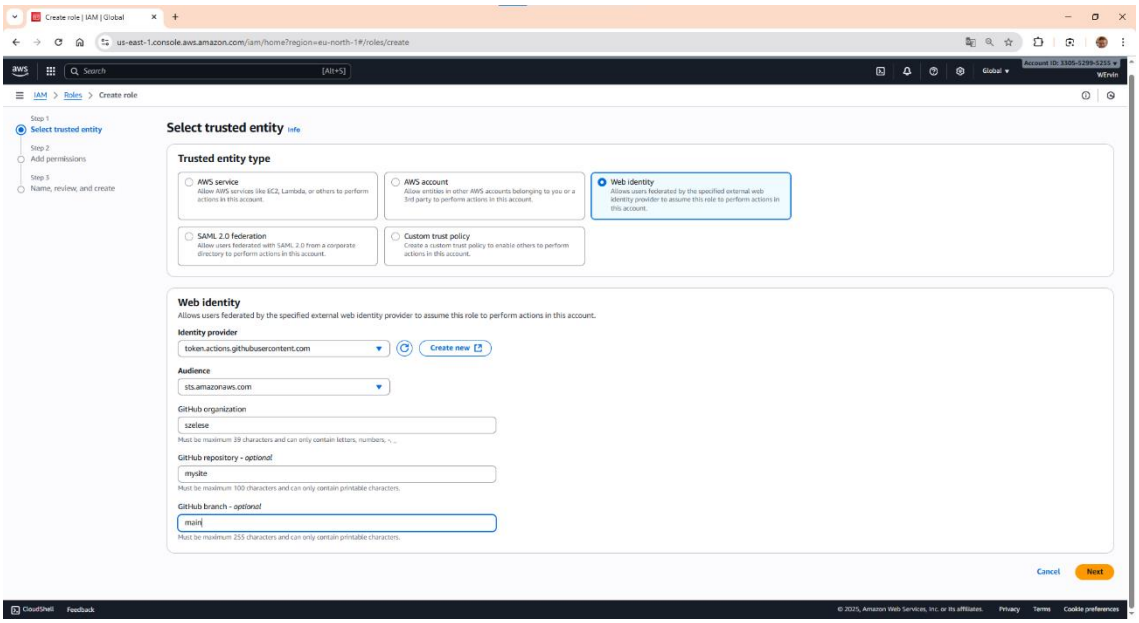
Provider URL:

https://token.actions.githubusercontent.com

Audience:

sts.amazonaws.com

Step 44: Configuring the Trusted Entity (Step 1)



IAM/roles/create role:

WEB Identity

Identity provider: token.actions.githubusercontent.com

Audience: sts.amazonaws.com

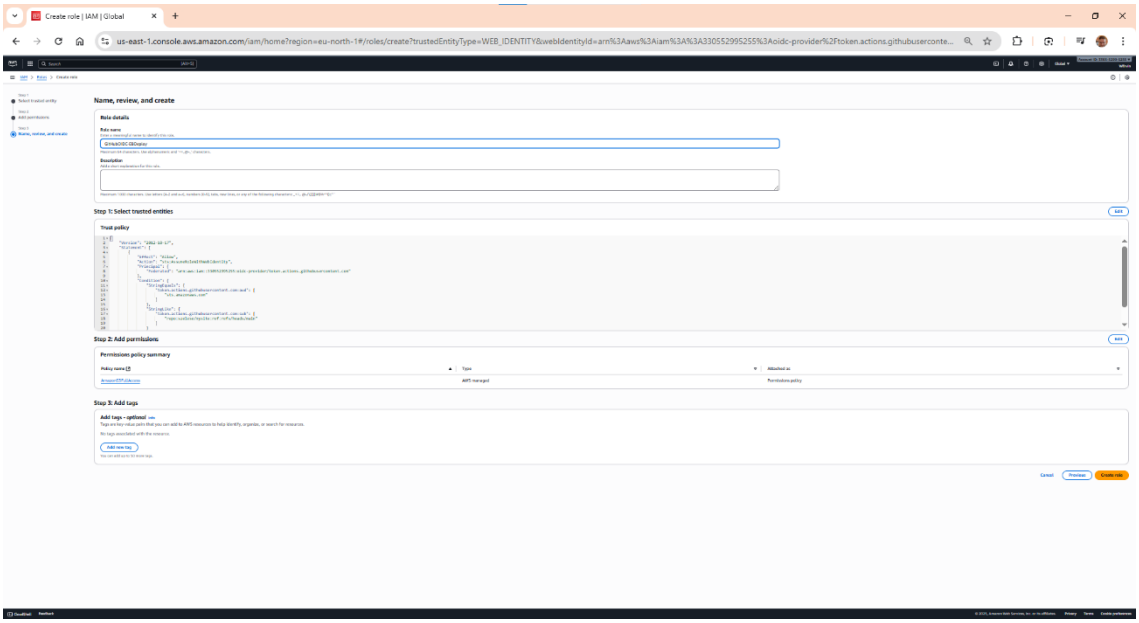
GitHub organization: YourGitHubName # your own github name

GitHub repository: mysite # the repository name for deployment

GitHub branch: main # usually

Security Note: While the repository and branch fields are optional, specifying them is recommended for better security.

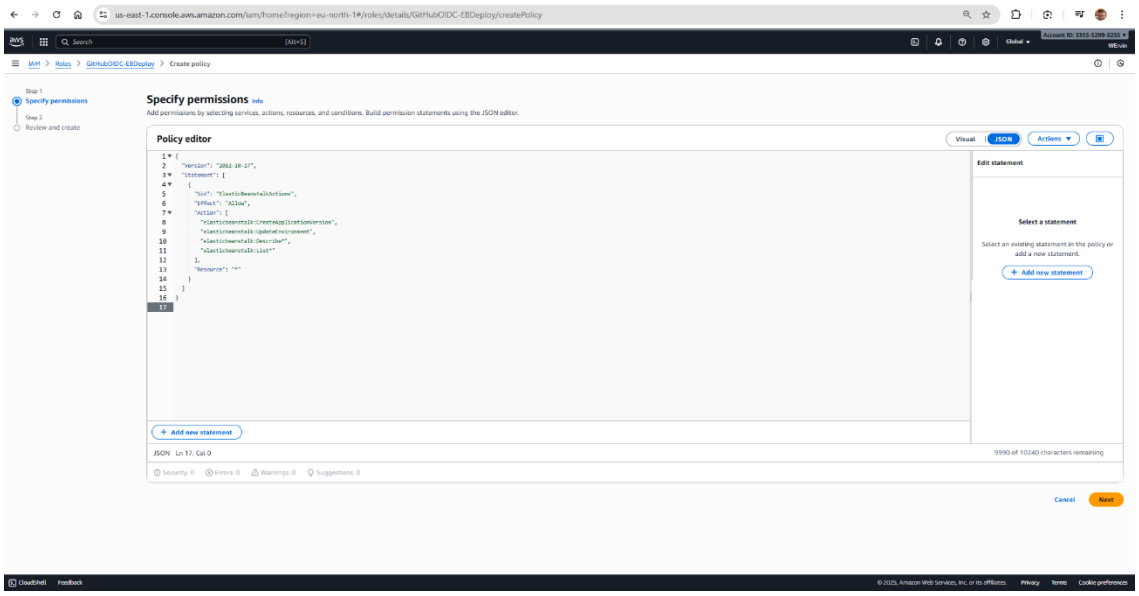
Step 45: Naming the Role (Step 3)



In the Role name field, enter: GitHubOIDC-EBDeploy.

Click Create role.

Step 46: Configuring Permissions



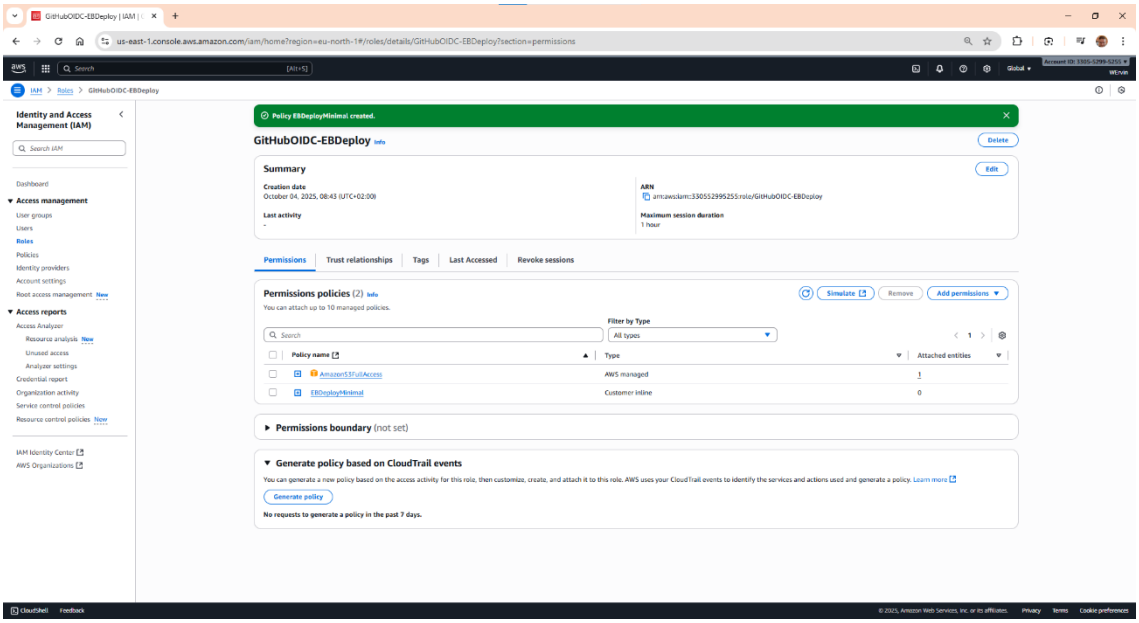
IAM->Roles->GitHubOIDC-EBDeploy->open

Permissions->Add permissions->Create inline policy->JSON

Enter the permission policy as shown in the project screenshots.->NEXT

Név: EBDeployMinimal -> Create policy

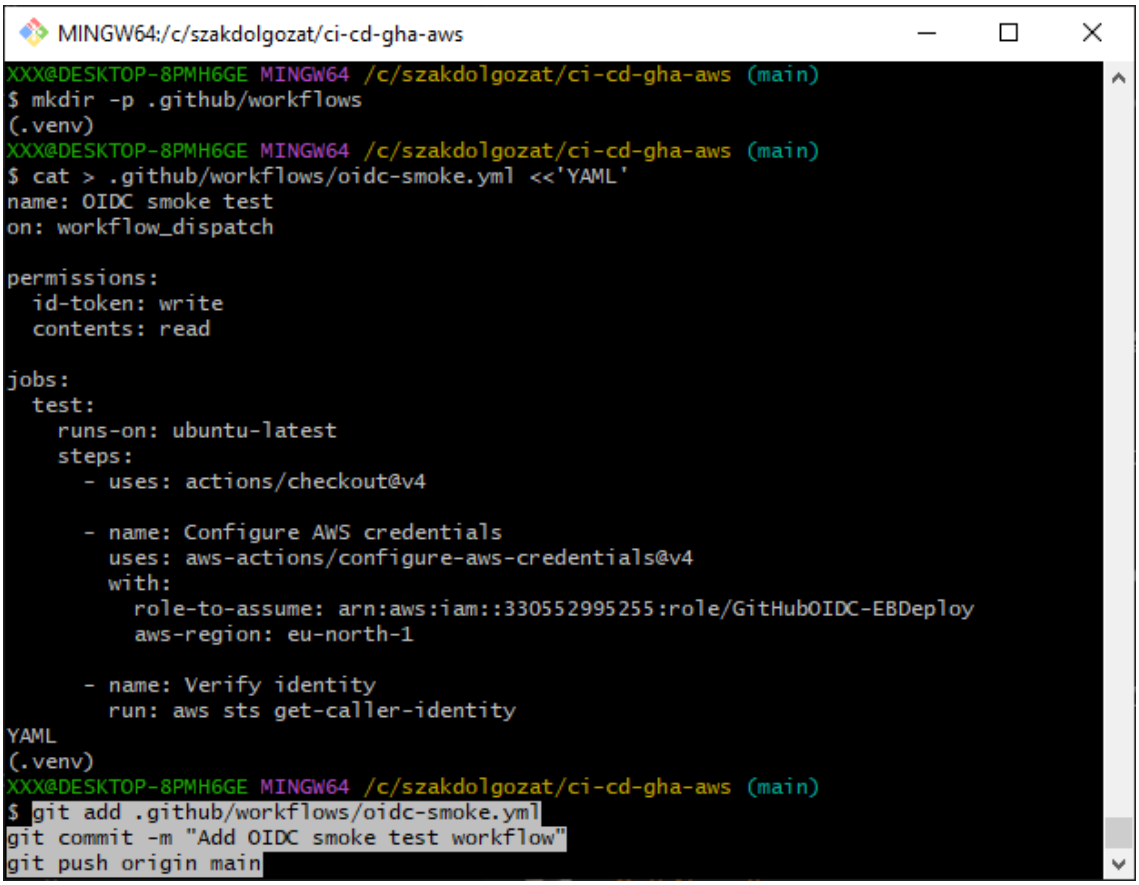
Step 47: Policy Verification and ARN Copying



Verify that the Permission policies section displays both expected policies.

Copy the Role ARN for later use: arn:aws:iam::330552995255:role/GitHubOIDC-EBDeploy .

Step 48: GitBash and commit & push

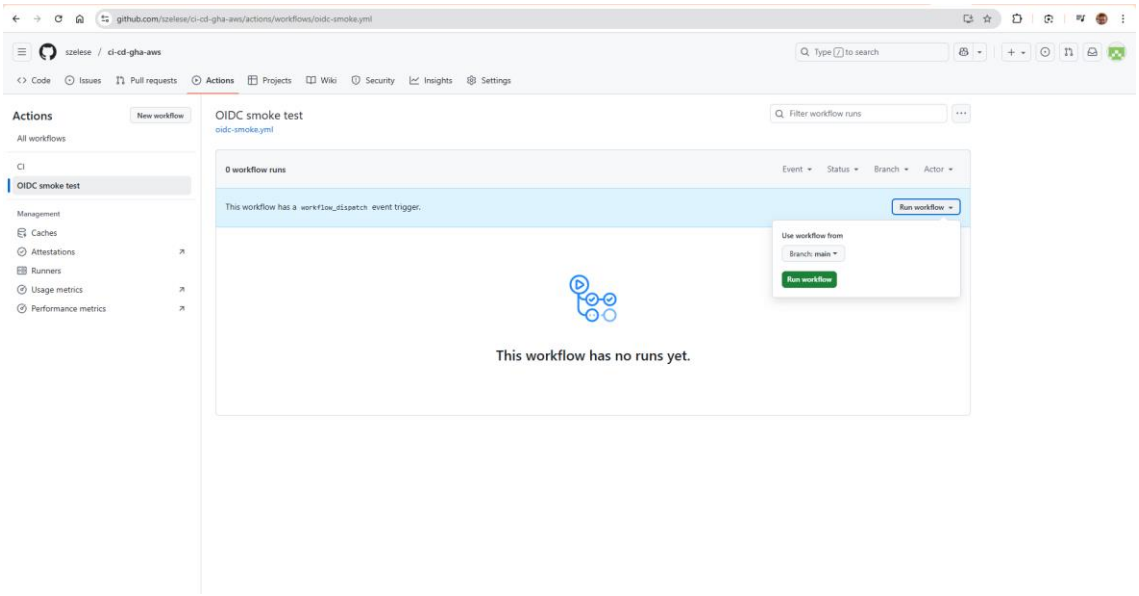


```
mkdir -p .github/workflows
```

Create the file with the specified content.

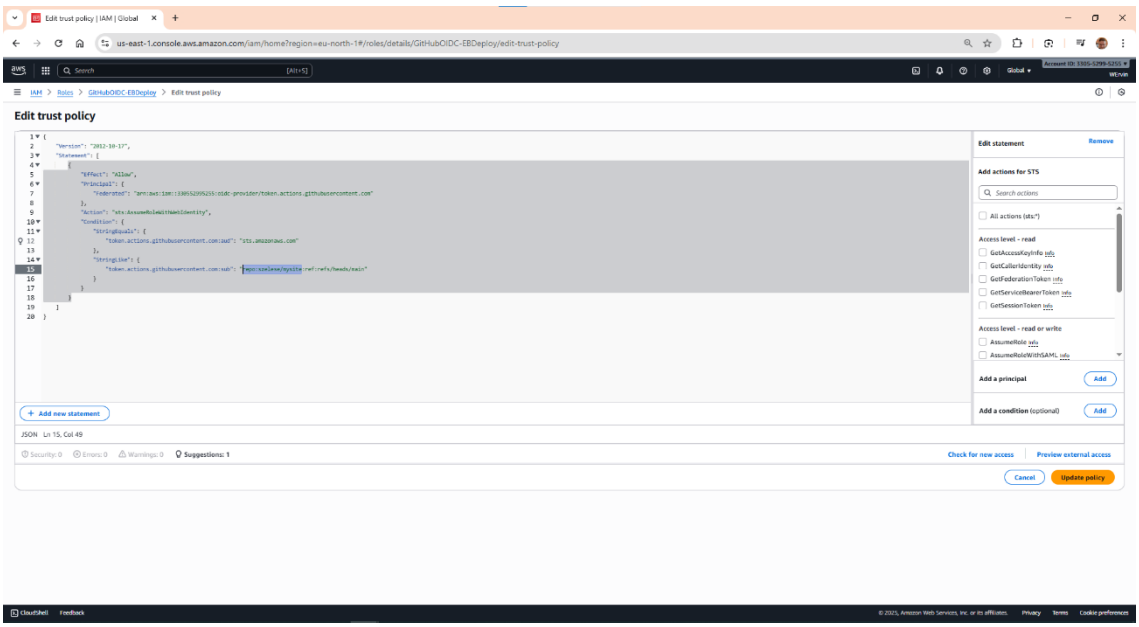
```
# commit&push
git add .github/workflows/oidc-smoke.yml
git commit -m "Add OIDC smoke test workflow"
git push origin main
```

Step 49: Running the Smoke Test



GitHub -> Actions ->OIDC smoke test -> Branch -> Run workflow

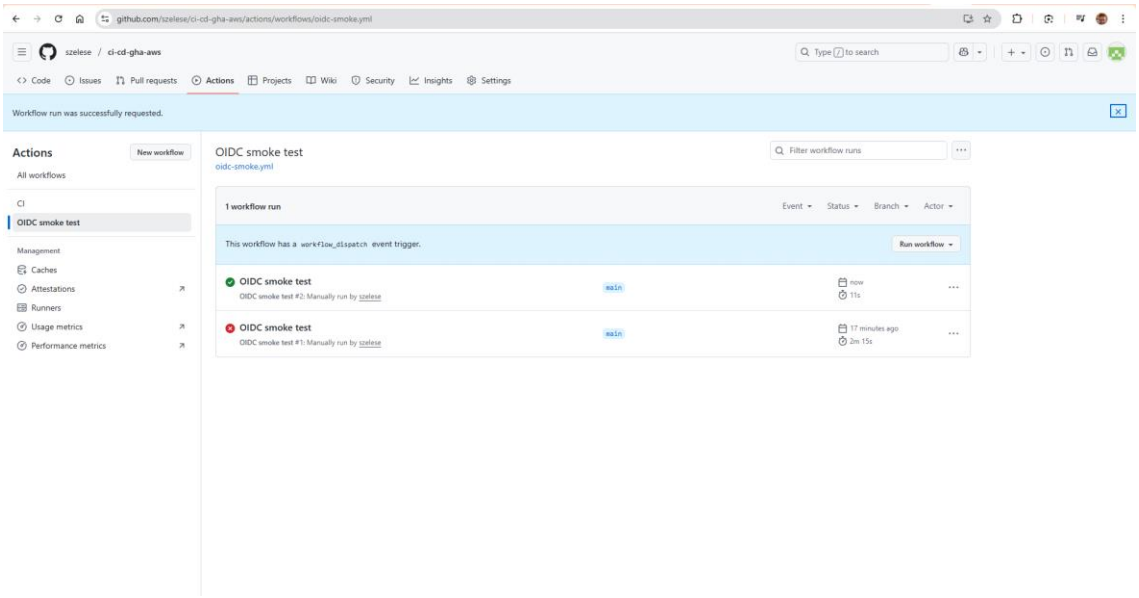
Step 50: Verification and Debugging



The default repository configuration points to the wrong location.

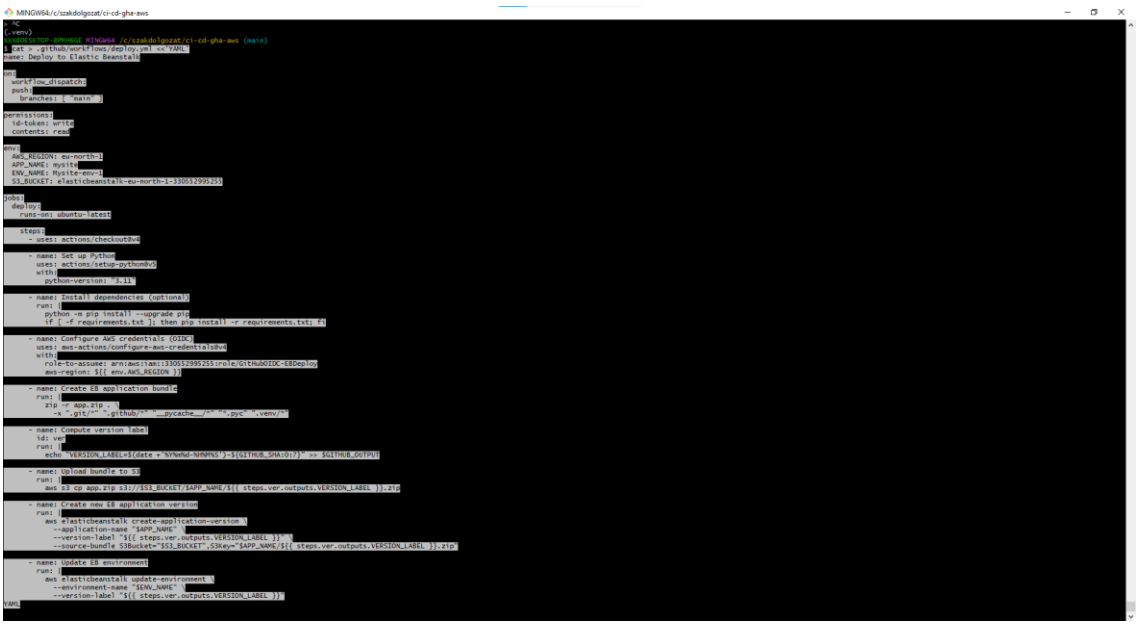
The actual repository is szelese/ci-cd-gha-aws; correct this so the smoke test can run properly. Note: Change to your repoName/projectName

Step 51: Re-running the Smoke Test

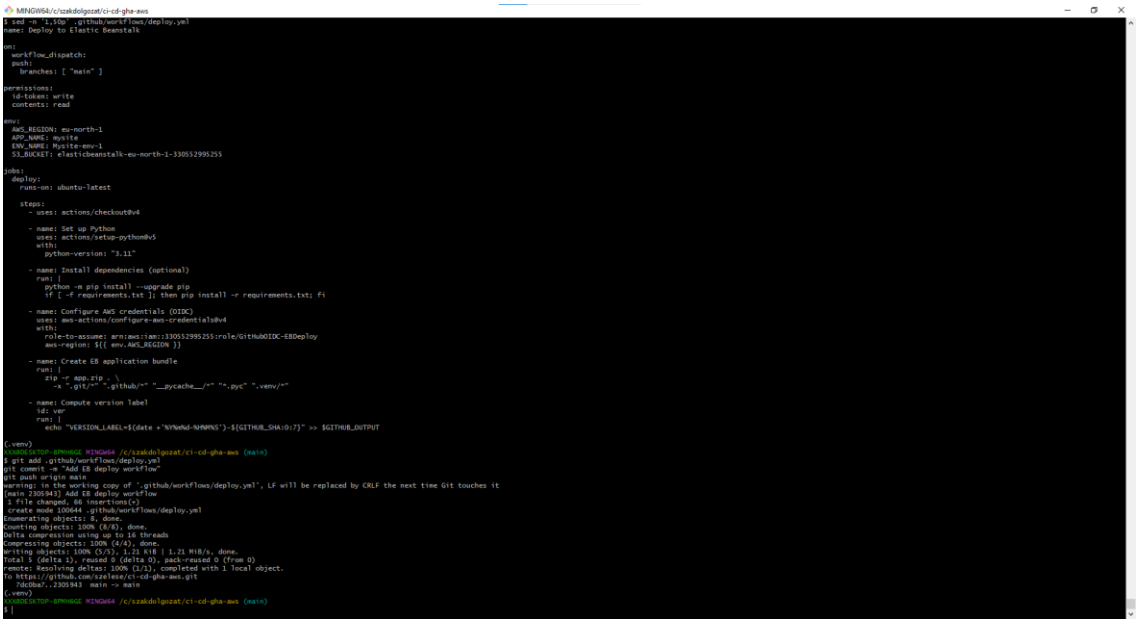


Successful OIDC smoke test between GitHub Actions and the AWS IAM role—the connection is verified and working (indicated by a green checkmark).

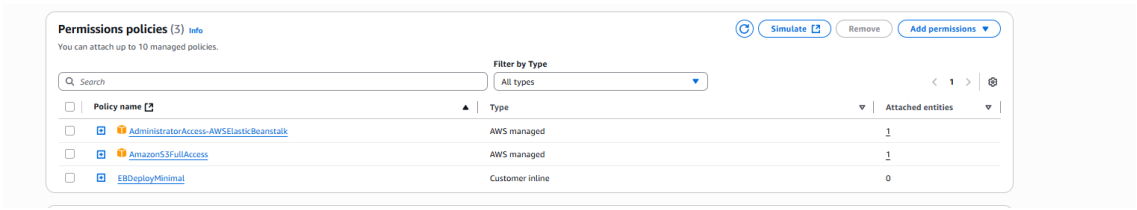
Step 52: Creating the Deploy File



Step 53: Verification, commit & push



Step 54: Manual Run Error and Permission Fix



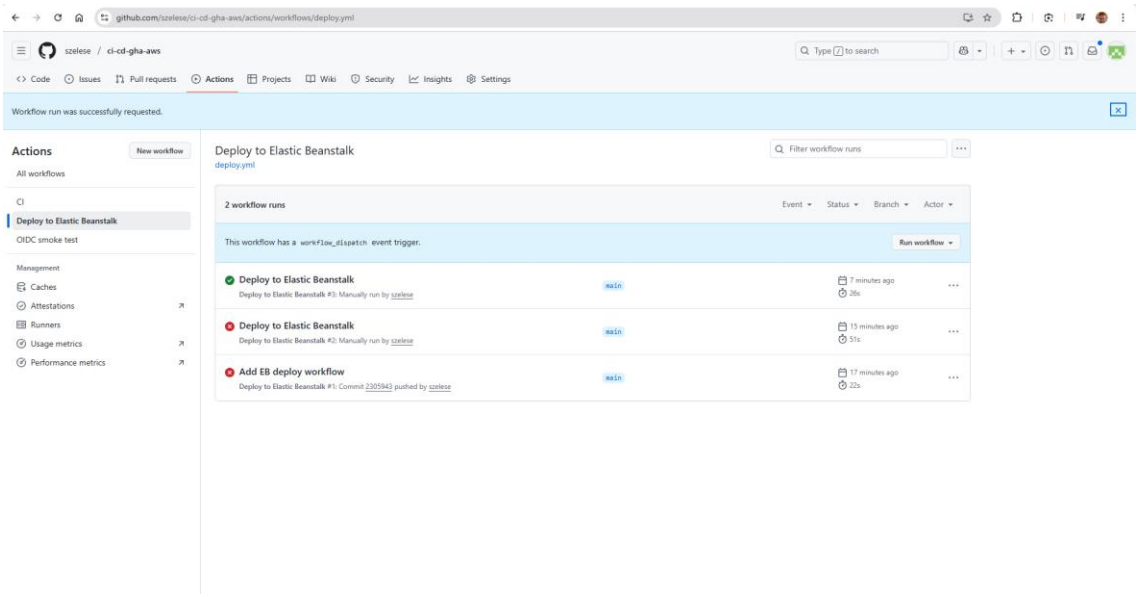
AWS Console → IAM → Roles → GitHubOIDC-EBDeploy.

Permissions tab → Add permissions → Attach policies.

AdministratorAccess-AWSElasticBeanstalk

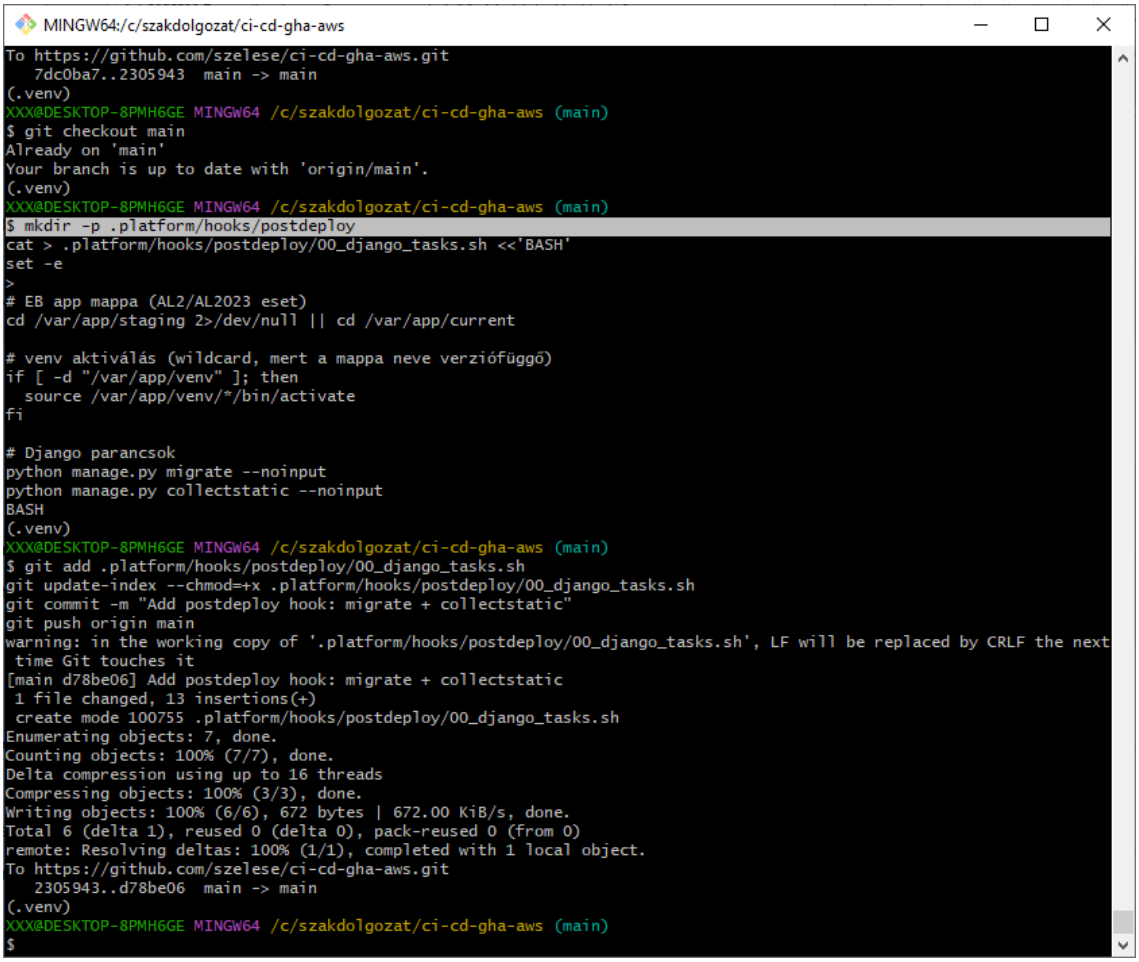
Add permissions

Step 55: GitHub Actions -> Re-run jobs



As shown in the logs, the most recent "Deploy to Elastic Beanstalk" run is now green—the CI→CD chain is fully functional.

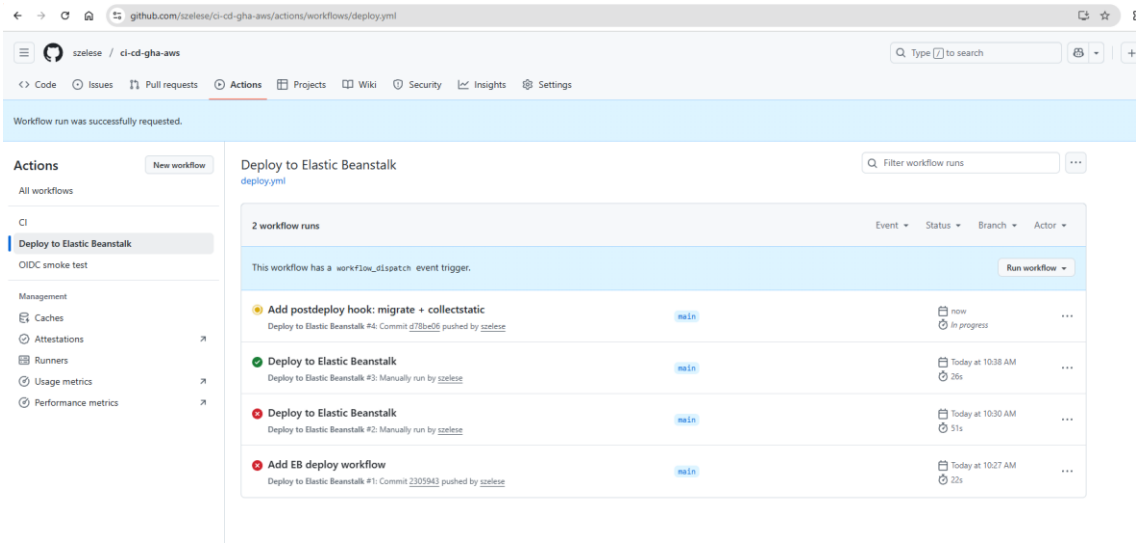
Step 56: Post-deploy Hook (migrate + collectstatic)



Technical Note: While not strictly required for operation, it is highly recommended to automate migrations, static file collection, and server-side secret management.

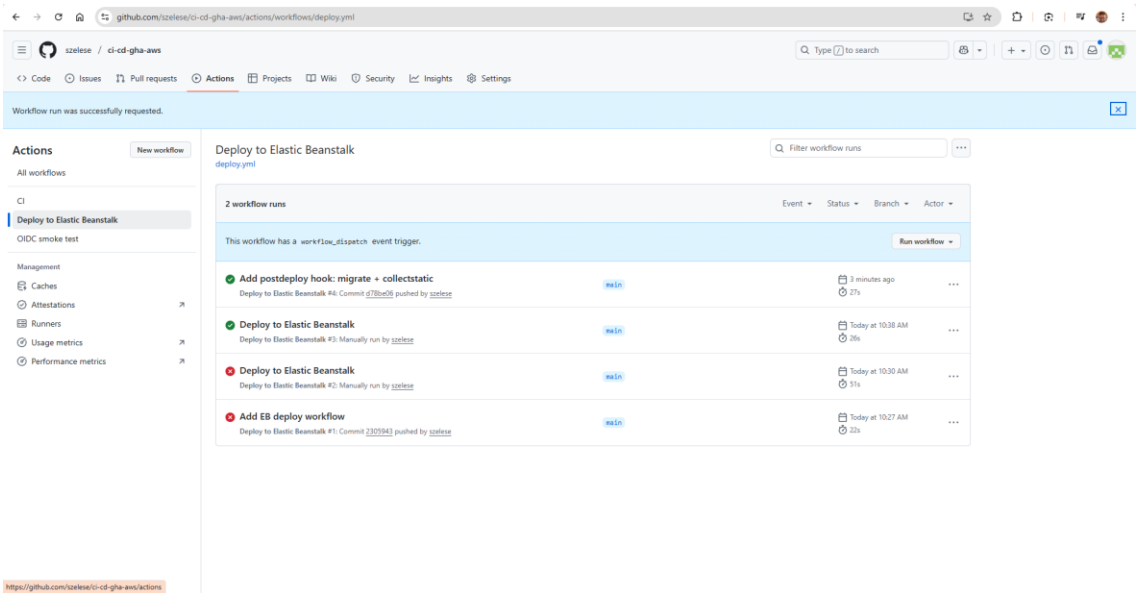
```
mkdir -p .platform/hooks/postdeploy
cat > .platform/hooks/postdeploy/00_django_tasks.sh <<'BASH'
git add .platform/hooks/postdeploy/00_django_tasks.sh
git update-index --chmod=+x .platform/hooks/postdeploy/00_django_tasks.sh
git commit -m "Add postdeploy hook: migrate + collectstatic"
git push origin main
```

Step 57: GitHub Actions Status

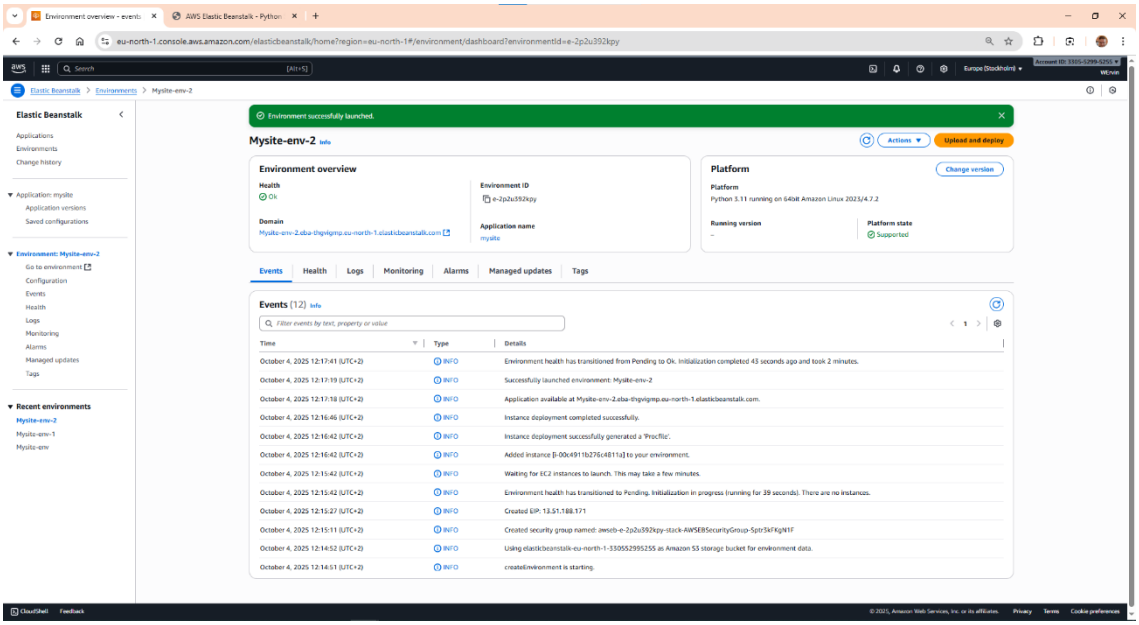


After the push and commit, GitHub Actions automatically begins execution.

Step 58: Successful Run

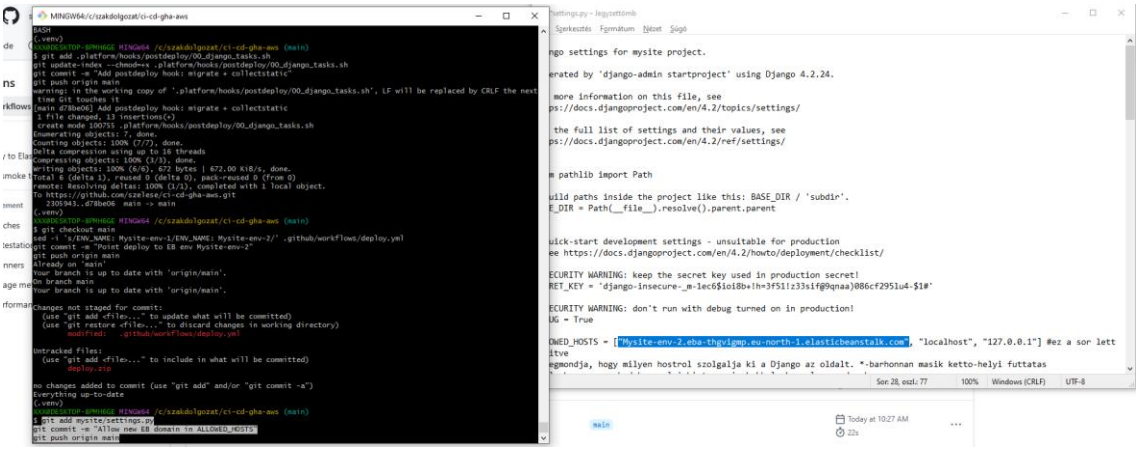


Step 59: Creating a New Server: mysite-env-2



Technical Note: For security reasons, mysite-env-1 was later placed into a Suspended state.

Step 60: Adjusting the Deploy Workflow for the New Environment



Modify settings.py to restrict ALLOWED_HOSTS to the specific domain:

Mysite-env-2.eba-thgvigmp.eu-north-1.elasticbeanstalk.com.

Note: use your own repo details.

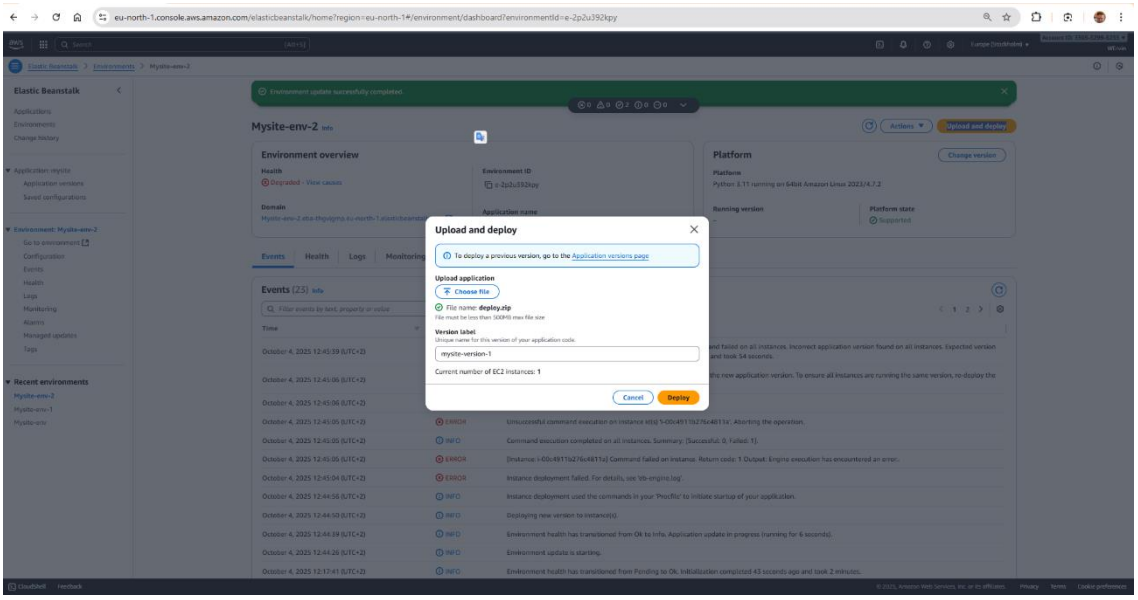
Commit and push the security update.

Step 61: Local vs Repo Environment Synchronization

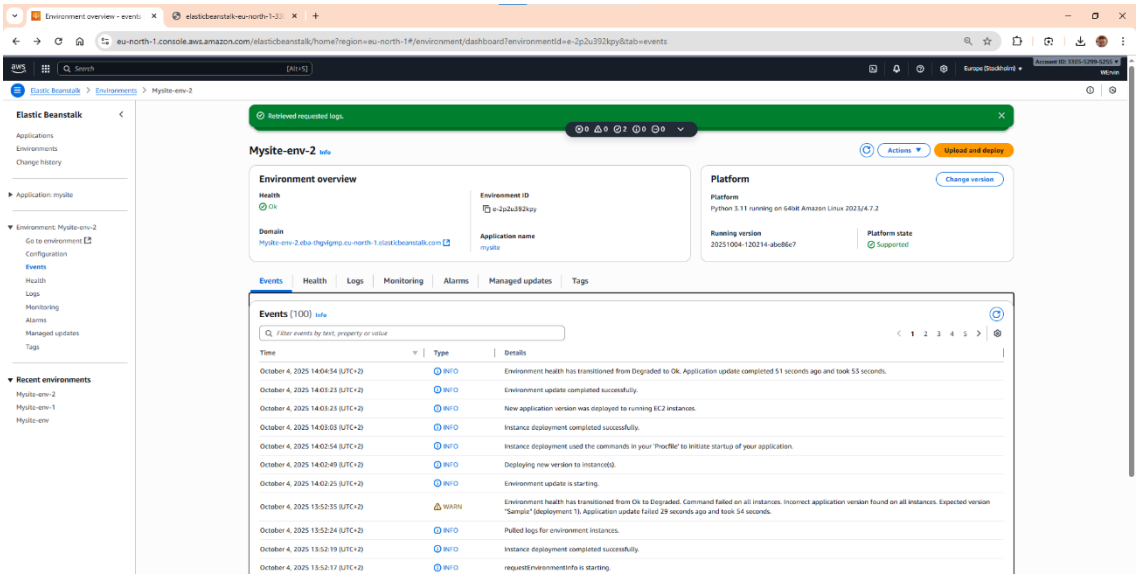
Locally the target is Mysite-env-2, but the repository still references environment 1; correct this via commit and push:


```
git add .github/workflows/deploy.yml
git commit -m "Switch ENV_NAME to Mysite-env-2"
git push origin main
```

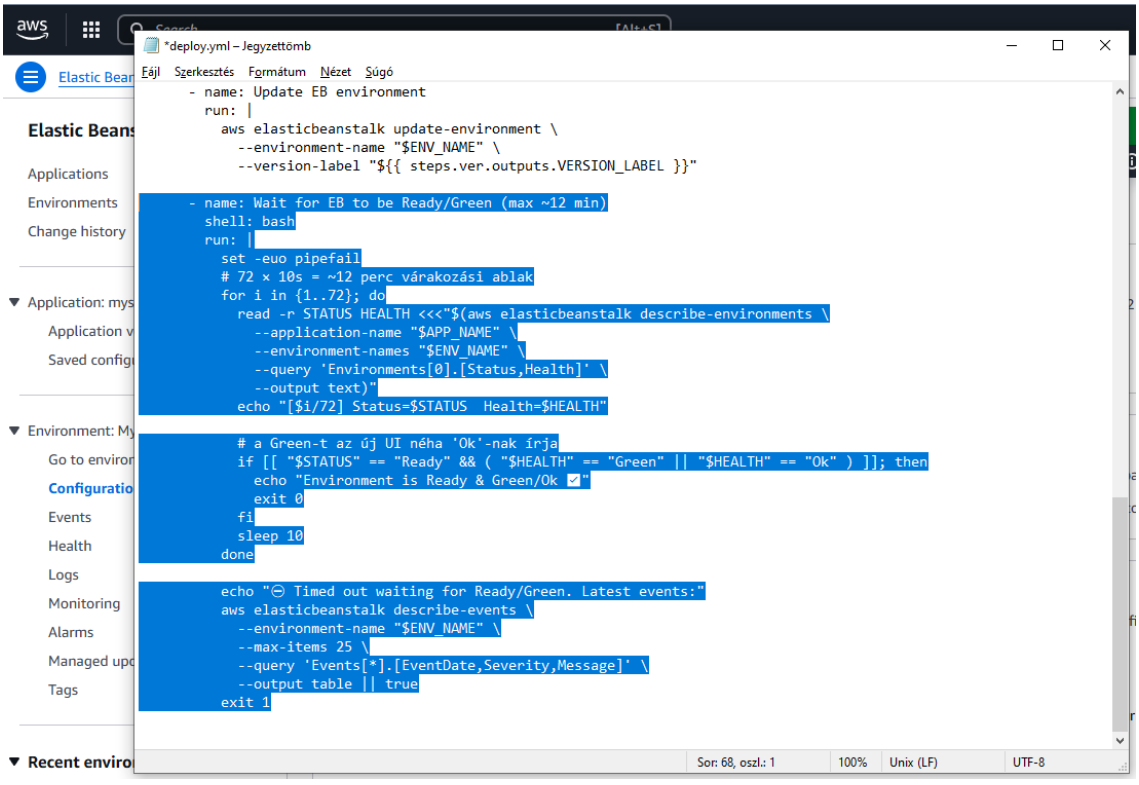
Step 62: End of Manual Deployment



Step 63: New Server Stability



Step 64: Troubleshooting and Prerequisites



The following configuration ensures the deployment workflow only completes successfully if the Elastic Beanstalk (EB) environment reaches a Ready + Green/Ok status. If this state is not reached, the workflow fails (marked in red) and outputs the most

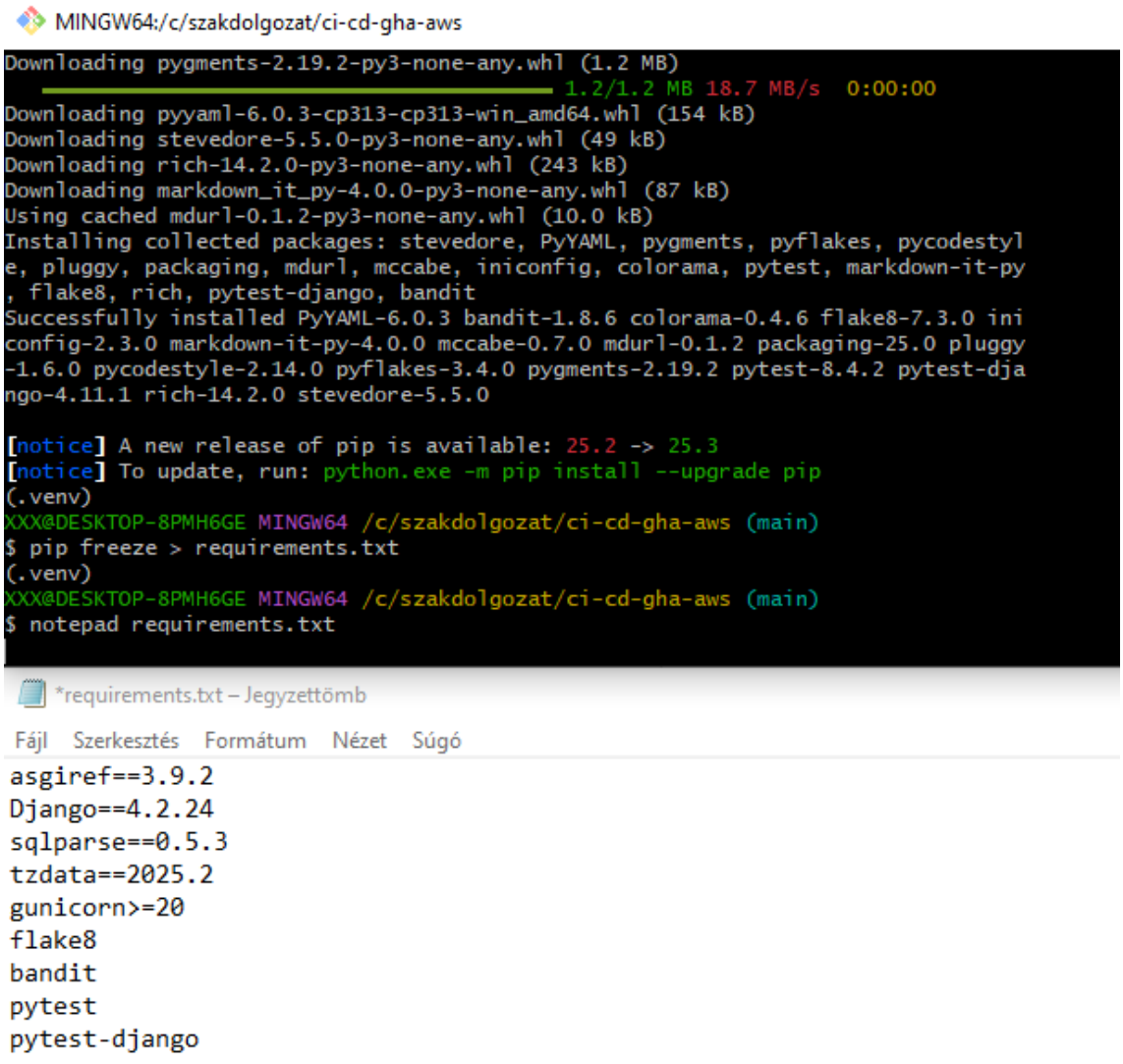
recent EB events. This serves as a simple diagnostic check and a prerequisite ensuring the server only goes live when the system is stable. Commit&push the changes GitBash:

```
git checkout main
git add .github/workflows/deploy.yml
git commit -m "Deploy: wait until EB is Ready & Green"
git push origin main
```

Step 65: CI Expansion: flake8, bandit, pytest

To strengthen the CI pipeline, install essential quality assurance tools: flake8 (style guide enforcement), bandit (security vulnerability scanning), and pytest-django (unit testing). Add these dependencies to requirements.txt to ensure the GitHub Actions runner installs them.

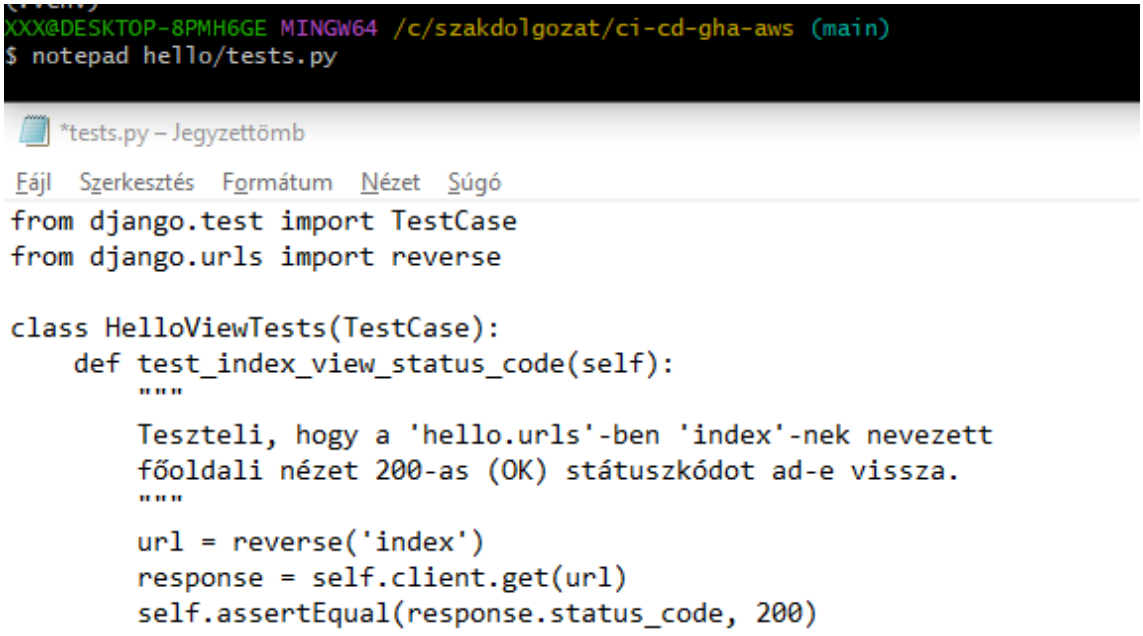
Technical Note: Instead of using pip freeze > requirements.txt (which can include unnecessary packages), manually append the new dependencies to the end of the file to maintain a clean environment.



```
pip install flake8 bandit pytest pytest-django
notepad requirements.txt #manuálisan beírni
# pip freeze > requirements.txt
```

Step 66: Basic Unit Test Creation

This test verifies that the application's main page (index view) loads successfully by checking for a 200 OK status code.

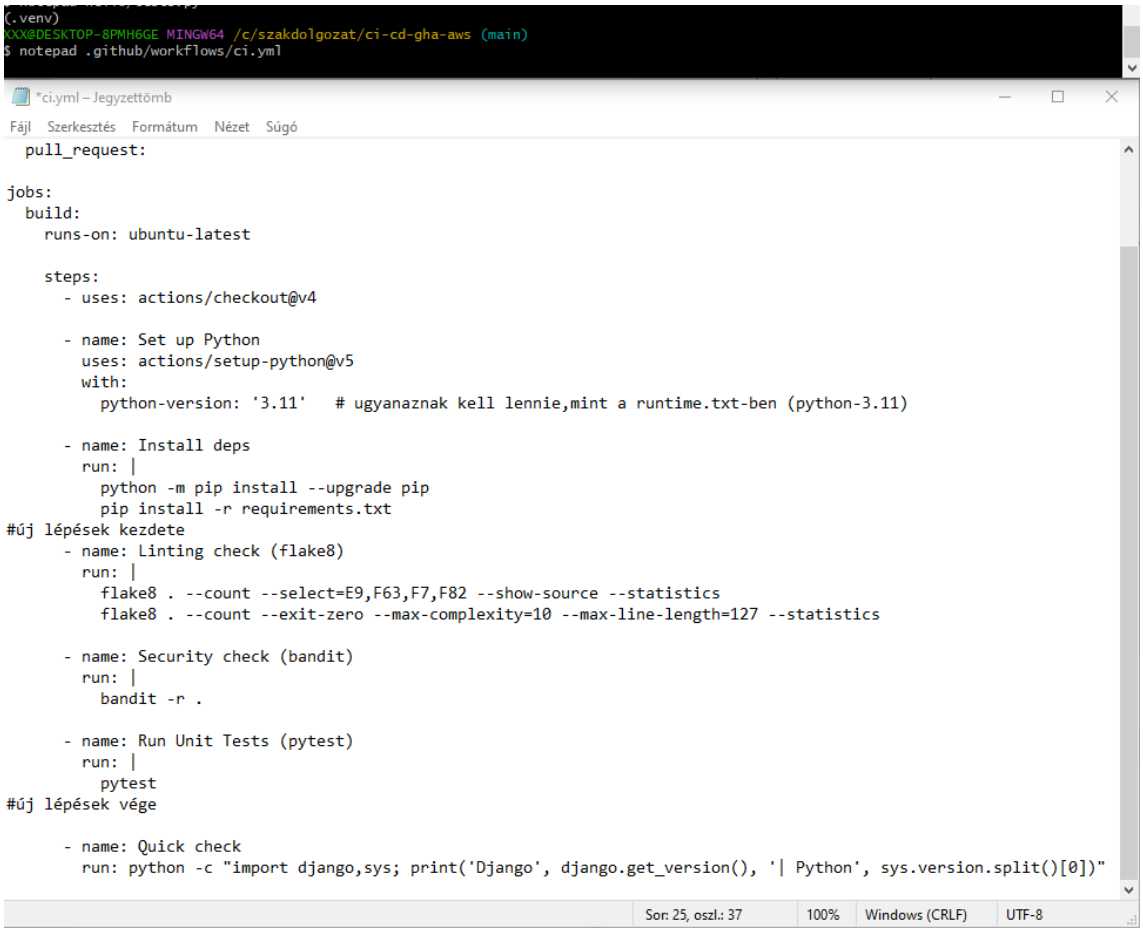


notepad hello/tests.py # insert the TestCase code

67. lépés: a ci.yml workflow kibővítése

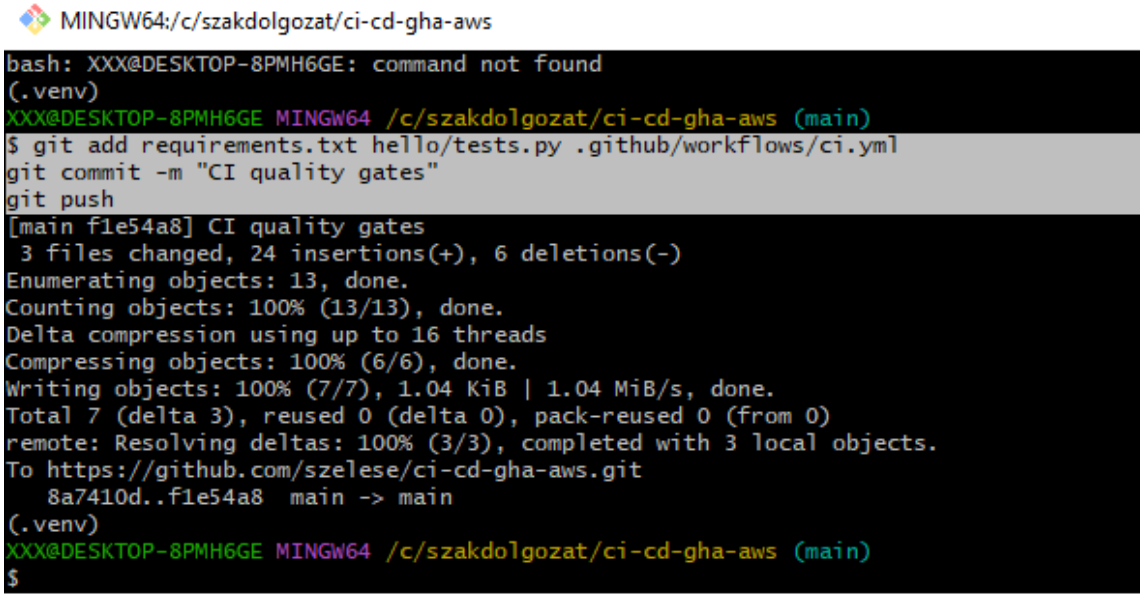
Modify .github/workflows/ci.yml by inserting three new steps before the 'Quick check': 'Linting check (flake8)', 'Security check (bandit)', and 'Run Unit Tests (pytest)'. These steps function as quality gates: if any check finds an error, the pipeline terminates, preventing faulty code from being integrated.

This completes the CI portion of the pipeline.



notepad .github/workflows/ci.yml

Step 68: Commit & Push

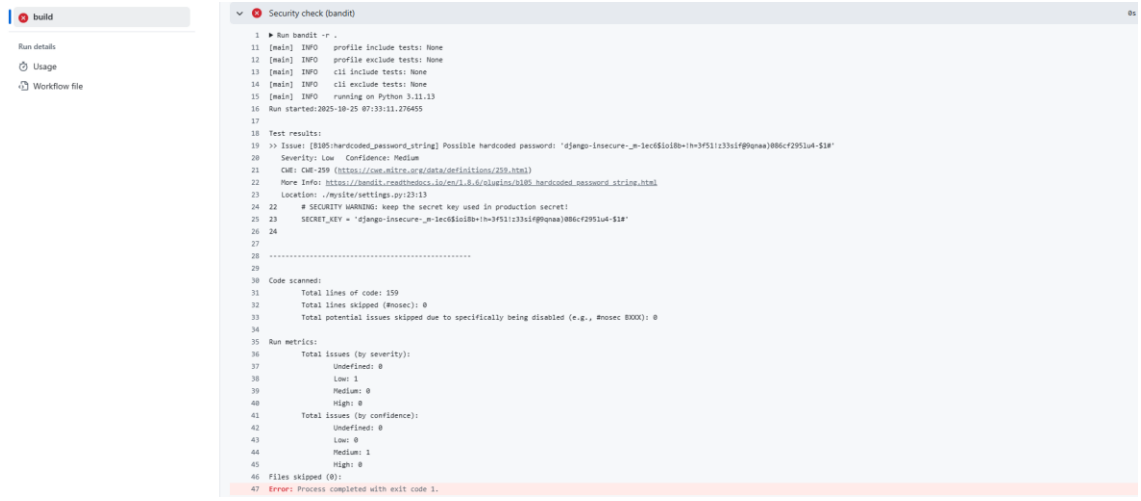


```
git add requirements.txt hello/tests.py .github/workflows/ci.yml
git commit -m "CI quality gates"
git push
```

Step 69: Verification of Expanded CI Pipeline

After pushing, GitHub Actions automatically triggers the expanded pipeline.

Observation: The run failed at the 'Security check (bandit)' step.



Step 70: Modifying mysite/settings.py

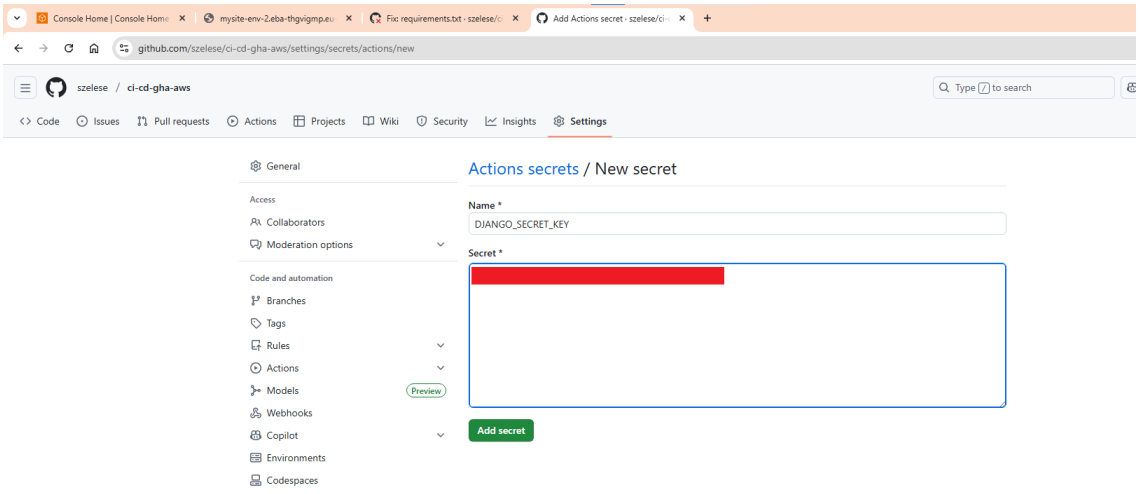
```
XXX@DESKTOP-8PMH6GE MINGW64 /c/szakdolgozat/ci-cd-gha-aws (main)
$ notepad mysite/settings.py
```

```
Generated by 'django-admin startproject' using Django 4.2.24.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/4.2/ref/settings/
"""

from pathlib import Path
import os
from django.core.management.utils import get_random_secret_key

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = os.environ.get("DJANGO_SECRET_KEY", get_random_secret_key())
```

Diagnosis: Bandit flagged the hardcoded SECRET_KEY.

Replace the static key with logic that prioritizes the DJANGO_SECRET_KEY environment variable, falling back to a randomly generated key if the variable is missing.

Step 71: Setting GitHub Repository Secret

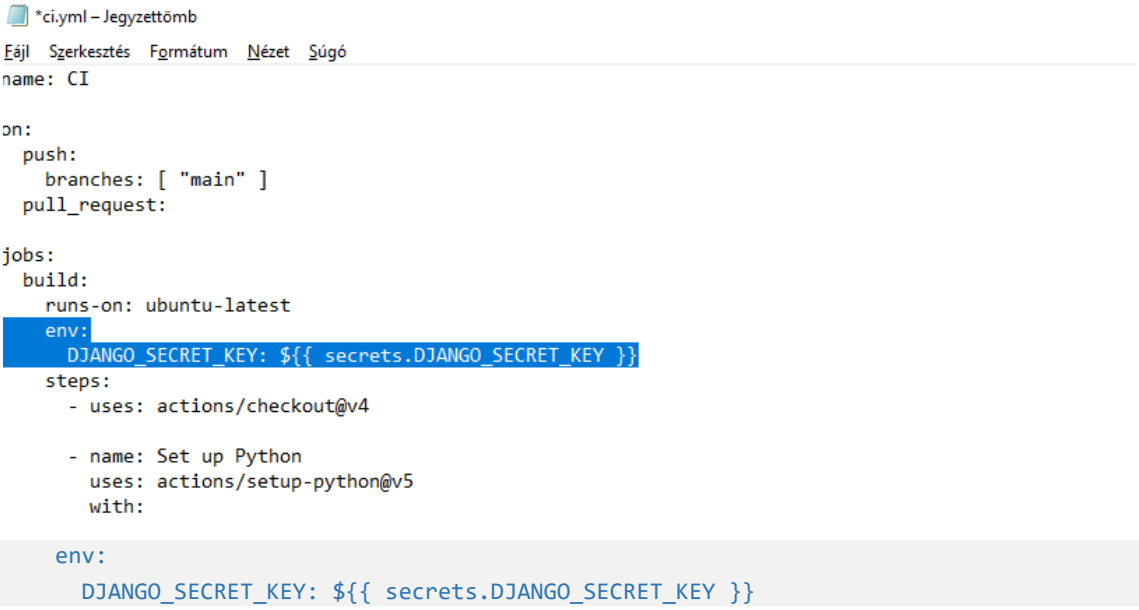


Settings/Secrets and Variables/Actions/New repository secret

Name: DJANGO_SECRET_KEY

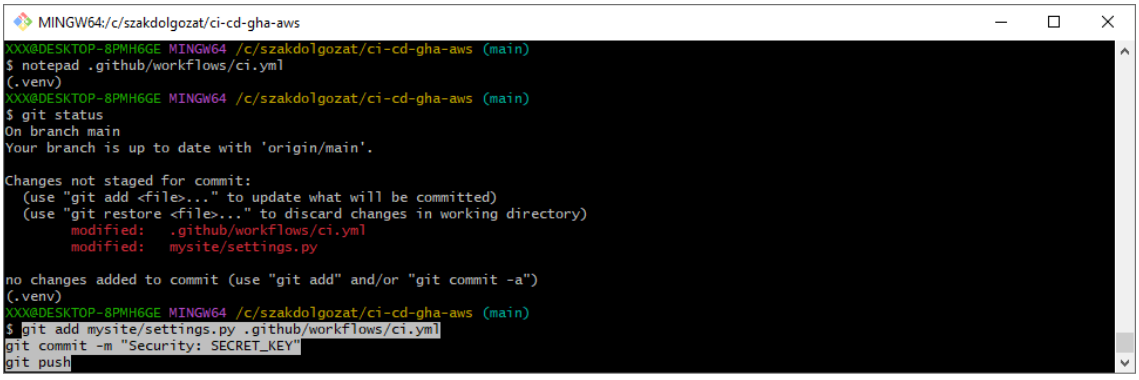
Secret: **********

Step 72: Updating the Workflow File



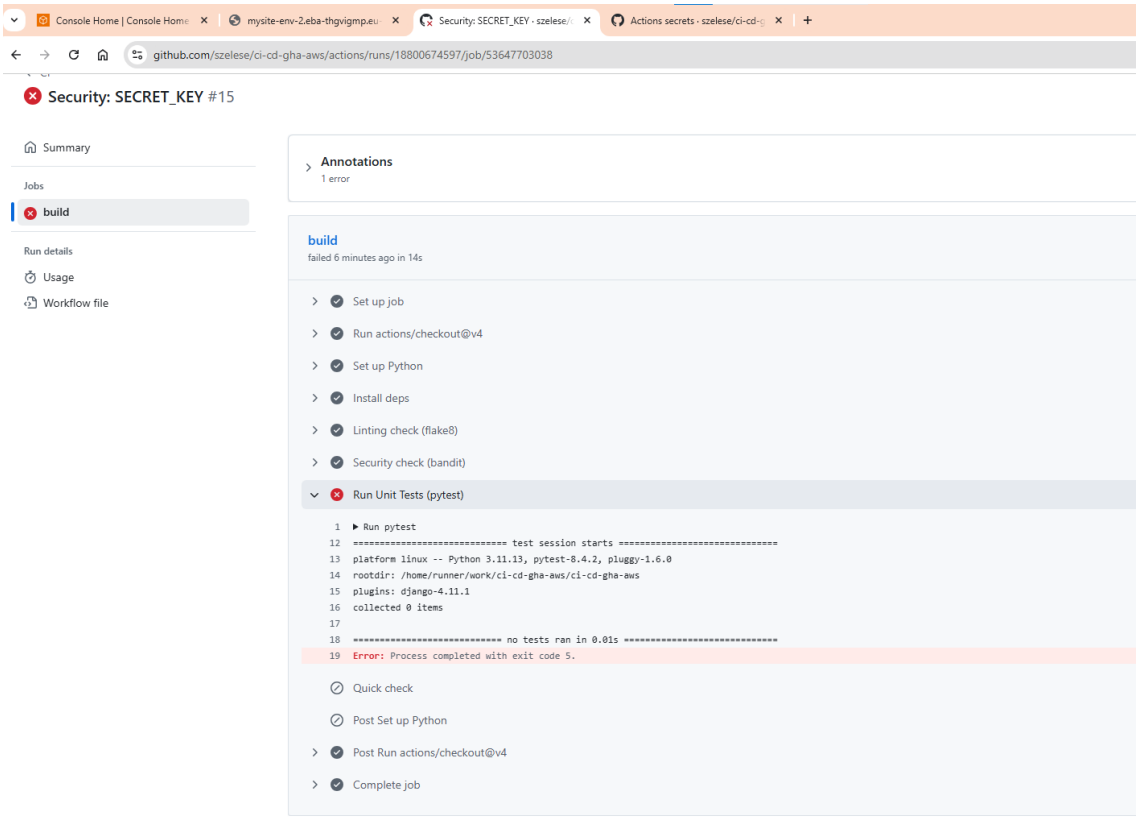

```
    env:
      DJANGO_SECRET_KEY: ${{ secrets.DJANGO_SECRET_KEY }}
```

Step 73: Commit & Push



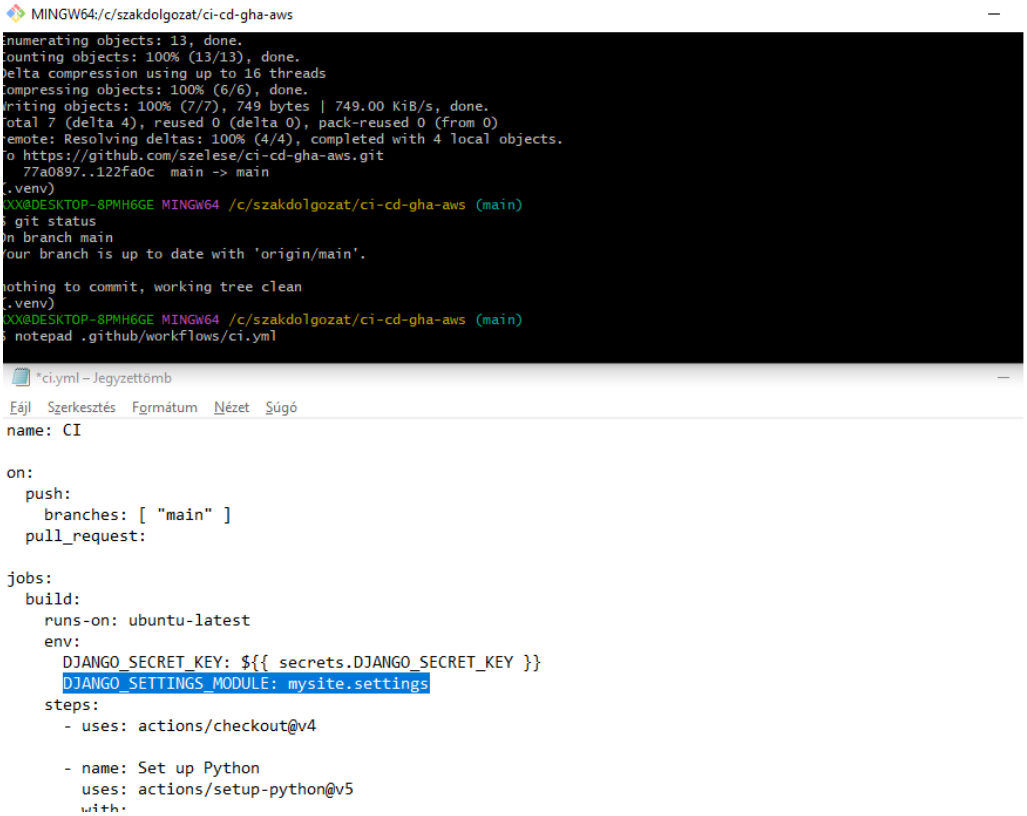

```
git add mysite/settings.py .github/workflows/ci.yml
git commit -m "Security: SECRET_KEY"
git push
```

Step 74: Unit Test Failure (Pytest Configuration)



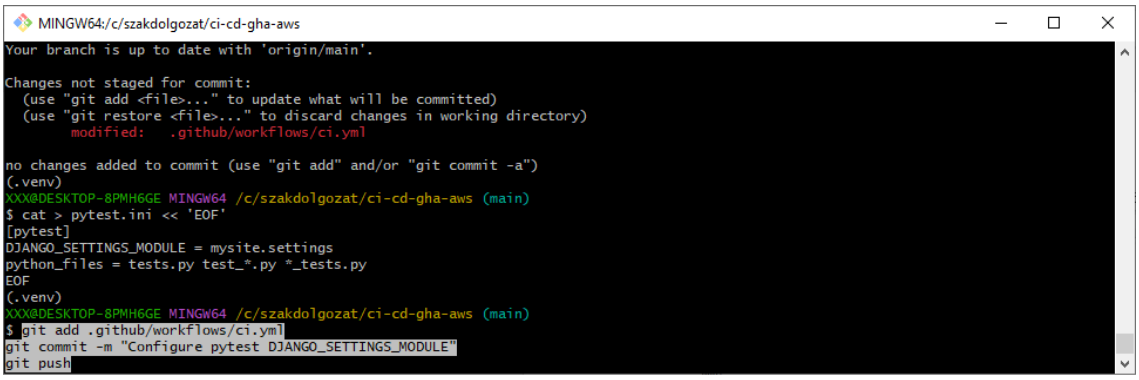
Observation: The run failed again because Pytest could not find the tests or identify the Django settings automatically.

Step 75: Setting DJANGO_SETTINGS_MODULE in CI



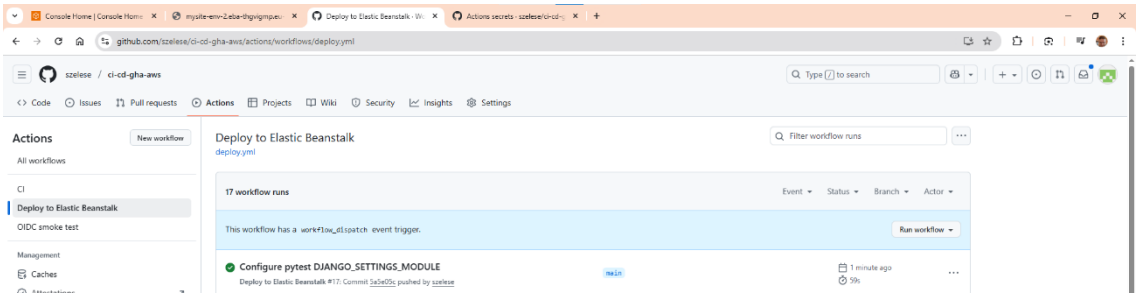

```
DJANGO_SETTINGS_MODULE: mysite.settings
...
      - name: Run Unit Tests (pytest)
        run: |
          pytest --ds=mysite.settings
```

Step 76: Creating pytest.ini and Commit & Push



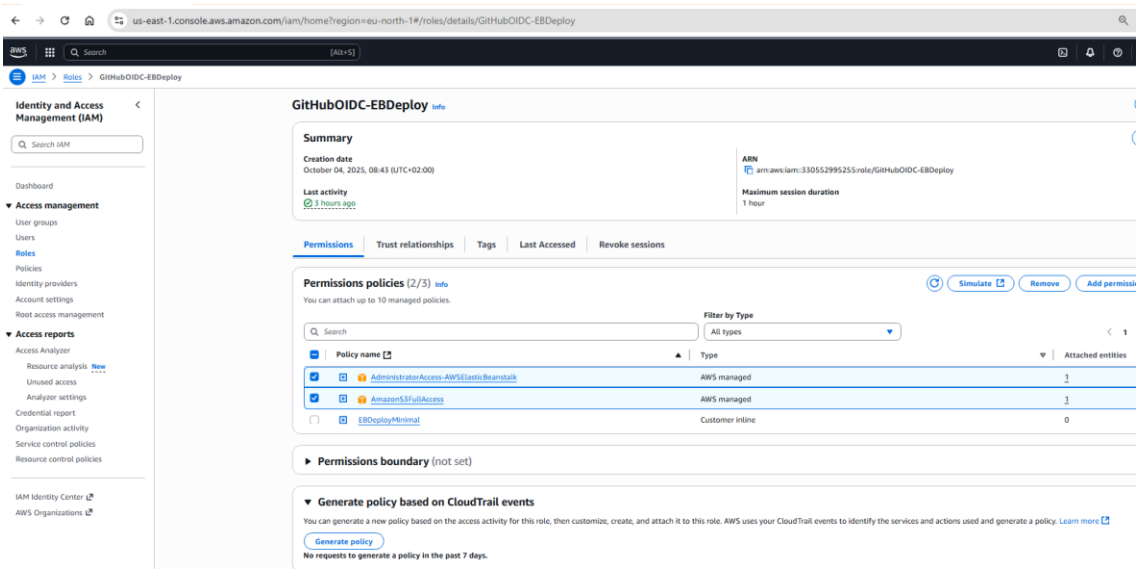

```
cat > pytest.ini << 'EOF'
[pytest]
DJANGO_SETTINGS_MODULE = mysite.settings
python_files = tests.py test_*.py *_tests.py
EOF
git add pytest.ini .github/workflows/ci.yml
git commit -m "Configure pytest DJANGO_SETTINGS_MODULE"
git push
```

Step 77: Verification of Successful CI Pipeline



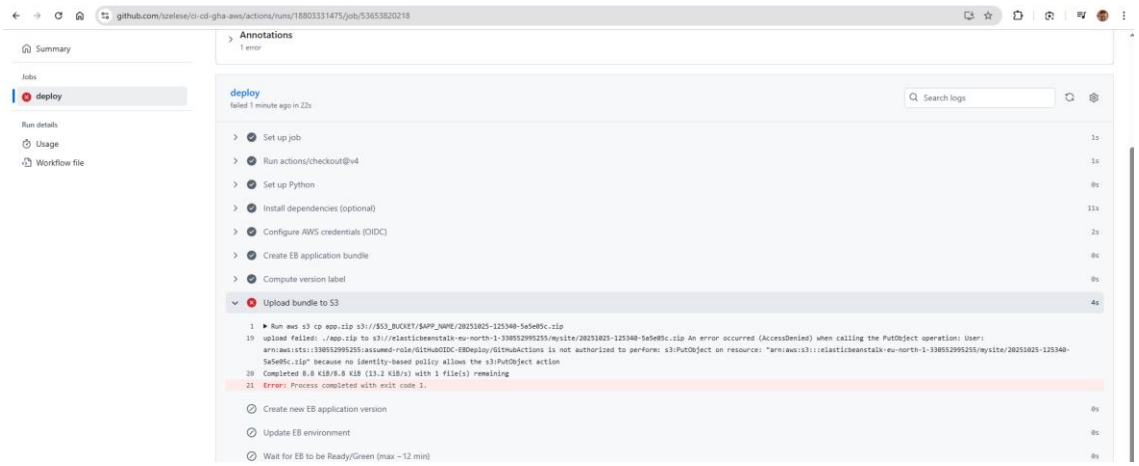
The GitHub Actions pipeline now runs successfully, passing all quality gates.

Step 78: Admin Policy Correction



To adhere to the Principle of Least Privilege, remove the "lazy" administrative policies: AmazonS3FullAccess and AdministratorAccess-AWSElasticBeanstalk.

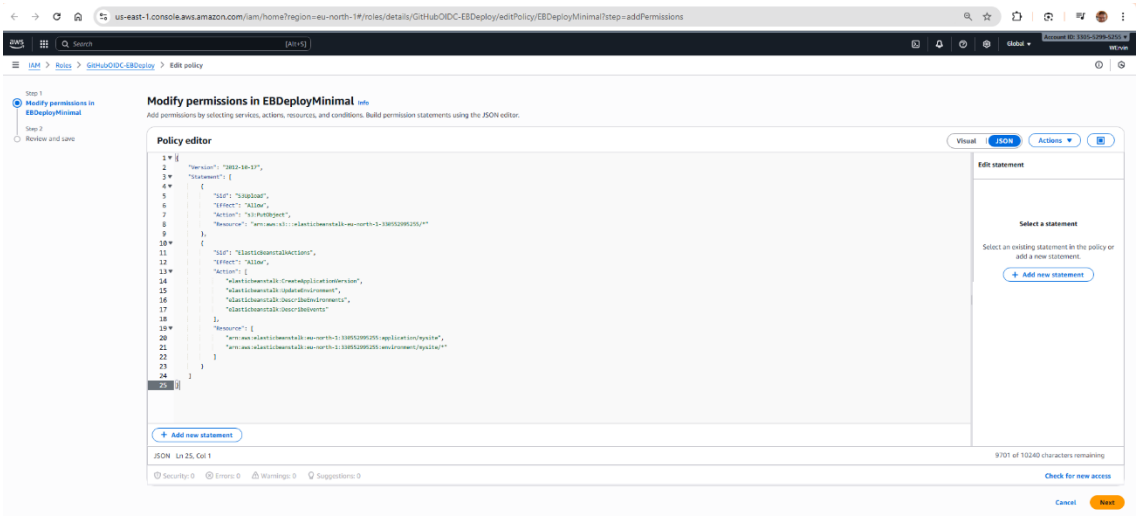
Step 79: Iterative Debugging Cycle



Repeatedly trigger the deploy.yml workflow to identify precisely which permissions are missing.

Observation: The first failure indicates s3:PutObject (file upload) is unauthorized.

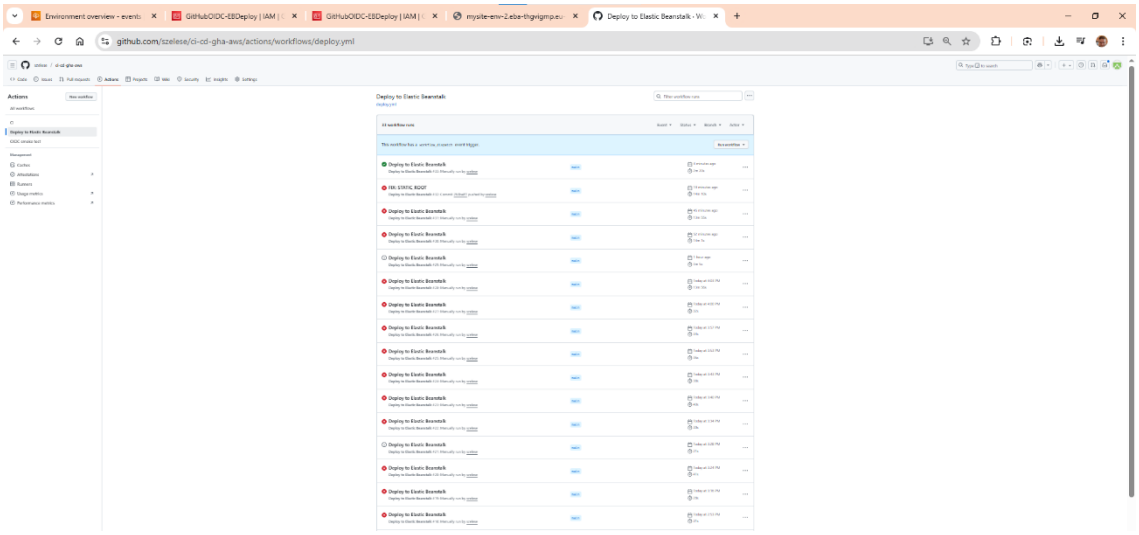
Step 80: Policy Modification



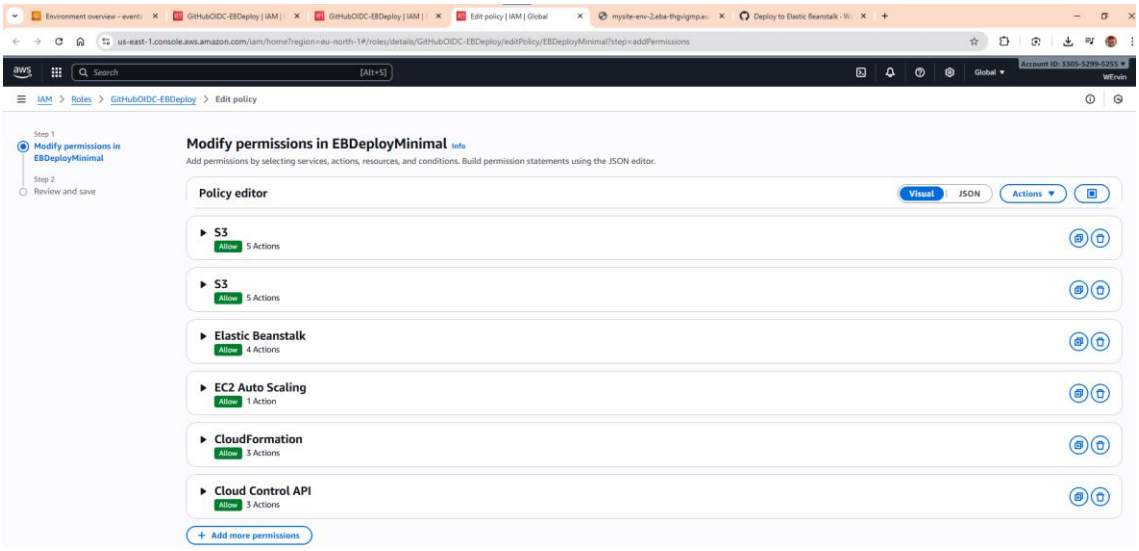
Manually update the EBDeployMinimal inline policy with the missing permission identified in the previous step.

Save the policy and trigger the deployment workflow again.

Step 81: Successful Deployment After 15 Iterations



Step 82: Final Minimal Policy Configuration




```
JSON:
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "S3ObjectAccess",
         "Effect": "Allow",
         "Action": [
            "s3:PutObject",
            "s3:GetObject",
            "s3:GetObjectAcl",
            "s3:PutObjectAcl",
            "s3:DeleteObject"
         ],
         "Resource": "arn:aws:s3:::elasticbeanstalk-eu-north-1-330552995255/*"
      },
      {
         "Sid": "S3BucketAccess",
         "Effect": "Allow",
         "Action": [
            "s3:CreateBucket",
            "s3:PutBucketPolicy",
            "s3:PutBucketOwnershipControls",
            "s3:ListBucket",
            "s3:GetBucketPolicy"
         ],
         "Resource": "arn:aws:s3:::elasticbeanstalk-eu-north-1-330552995255"
      },
      {
         "Sid": "ElasticBeanstalkCloudformationAutoscalingAccess",
         "Effect": "Allow",
         "Action": [
            "elasticbeanstalk:CreateApplicationVersion",
```

```
            "elasticbeanstalk:UpdateEnvironment",
            "elasticbeanstalk:DescribeEnvironments",
            "elasticbeanstalk:DescribeEvents",
            "cloudformation:GetTemplate",
            "cloudformation:DescribeStackResource",
            "cloudformation:DescribeStackResources",
            "autoscaling:DescribeAutoScalingGroups"
         ],
         "Resource": [
            "arn:aws:elasticbeanstalk:eu-north-1:330552995255:application/mysite",
            "arn:aws:elasticbeanstalk:eu-north-1:330552995255:environment/mysite/*",
            "arn:aws:elasticbeanstalk:eu-north-
1:330552995255:applicationversion/mysite/*",
            "arn:aws:cloudformation:eu-north-1:330552995255:stack/awseb-*",
            "arn:aws:autoscaling:eu-north-
1:330552995255:autoScalingGroup:*:autoScalingGroupName/*"
         ]
      }
   ]
}
```

Step 83: Enforcing Sequential CI→CD Flow
Analysis of runtimes showed that CI and Deployment workflows were previously triggering in parallel upon a push.

Modify deploy.yml to use the workflow_run event.

This configuration forces the Deployment to start only if the CI workflow has completed with a success conclusion.

ci-cd-gha-aws > .github > workflows > deploy.yml

```yaml
name: Deploy to Elastic Beanstalk

on:
  workflow_run:
    workflows: ["CI"]
    types:
      - completed

permissions:
  id-token: write
  contents: read

env:
  AWS_REGION: eu-north-1
  APP_NAME: mysite
  ENV_NAME: Mysite-env-2
  S3_BUCKET: elasticbeanstalk-eu-north-1-330552995255

jobs:
  deploy:
    if: ${{ github.event.workflow_run.conclusion == 'success' }}
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v4
        with:
          ref: ${{ github.event.workflow_run.head_sha }}
          fetch-depth: 0

      - name: Set up Python
        uses: actions/setup-python@v5
```

```
commit & push
git add .github/workflows/deploy.yml
```

```
git commit -m " CI quality gates enforced before deployment (sequential CI→CD flow)"
git push
```

Summary

This project successfully established a fully functional CI/CD pipeline that automates code integration, testing, and deployment to AWS Elastic Beanstalk.

Secure authentication via AWS IAM OIDC replaces static credentials.

The application remains in a stable, "Healthy" state in production, demonstrating the effectiveness of automated DevOps practices.