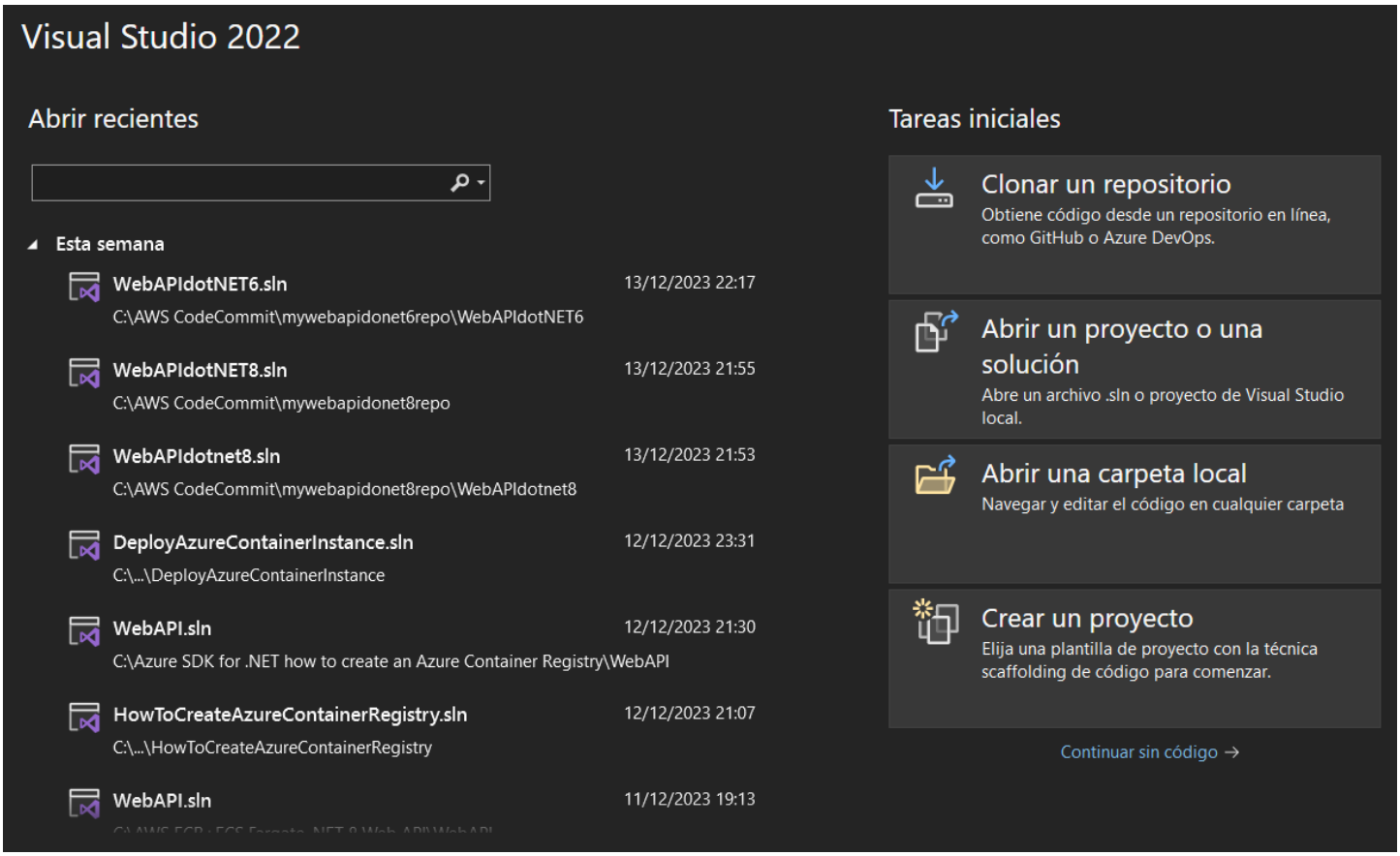
GithubActions: How to create .NET 7 Web API Docker image and Upload it to AWS ECS Elastic Container Service

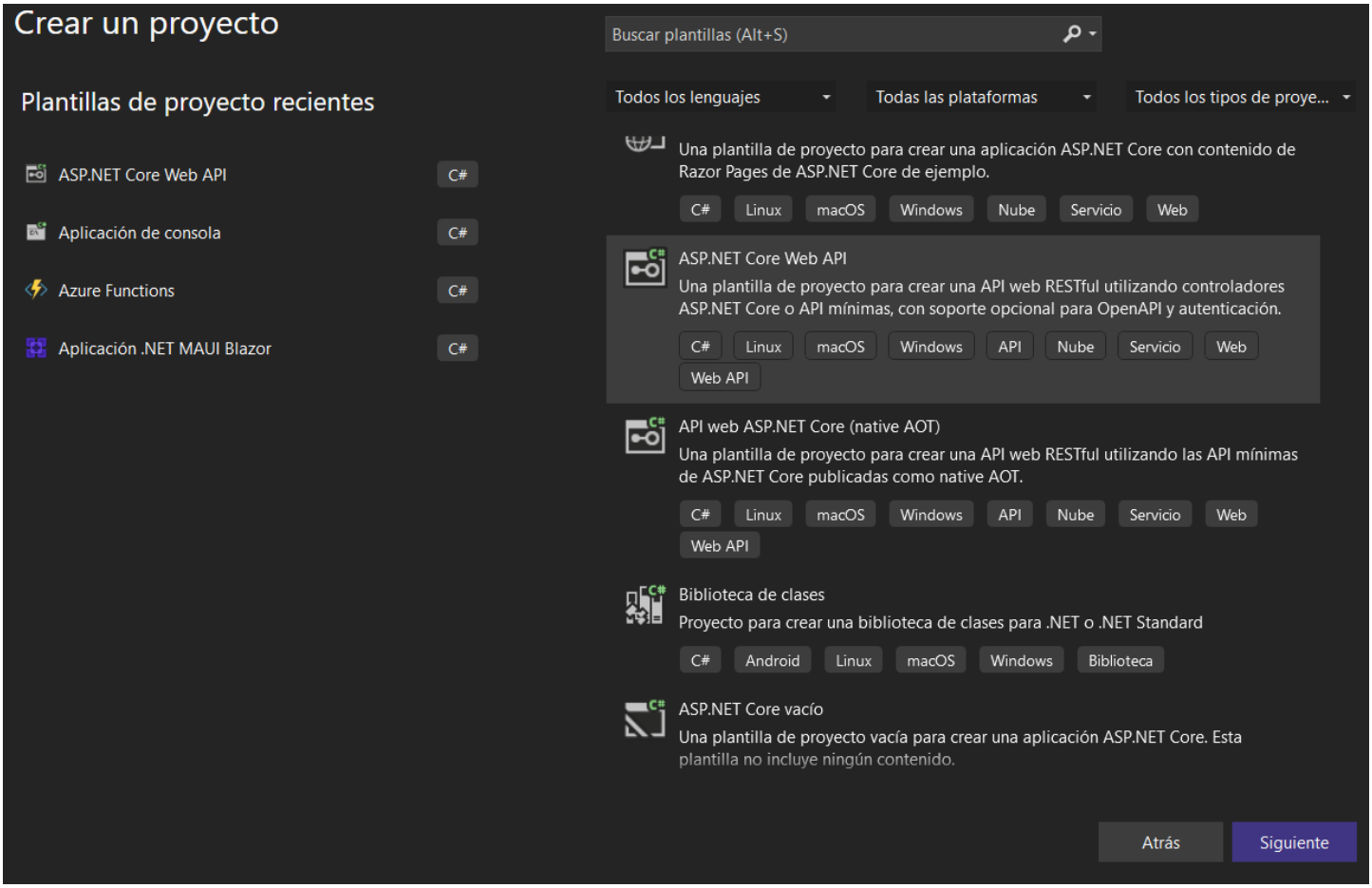
1. Create a .NET 7 Web API application in Visual Studio

We run Visual Studio 2022

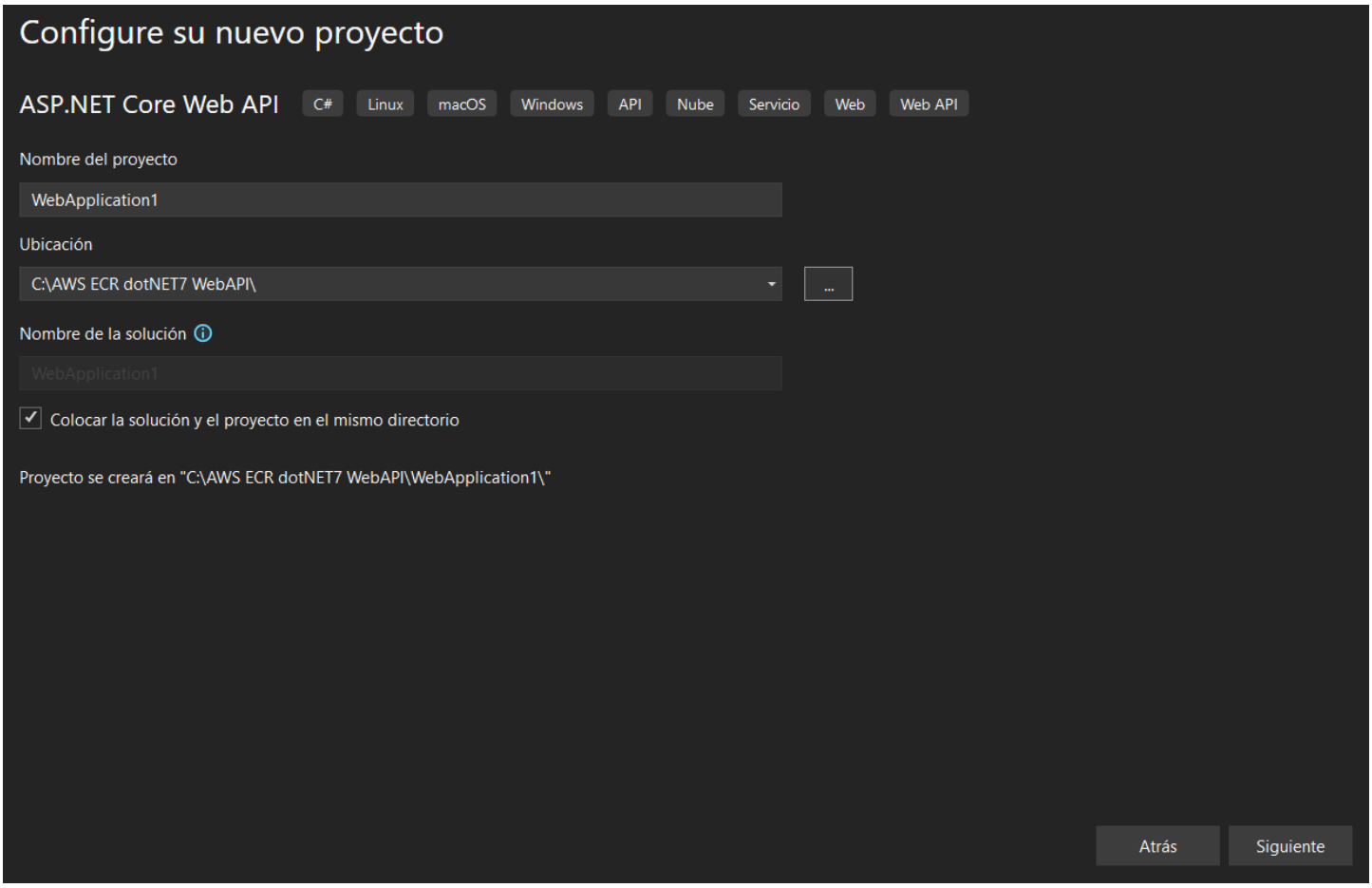
We create a new project



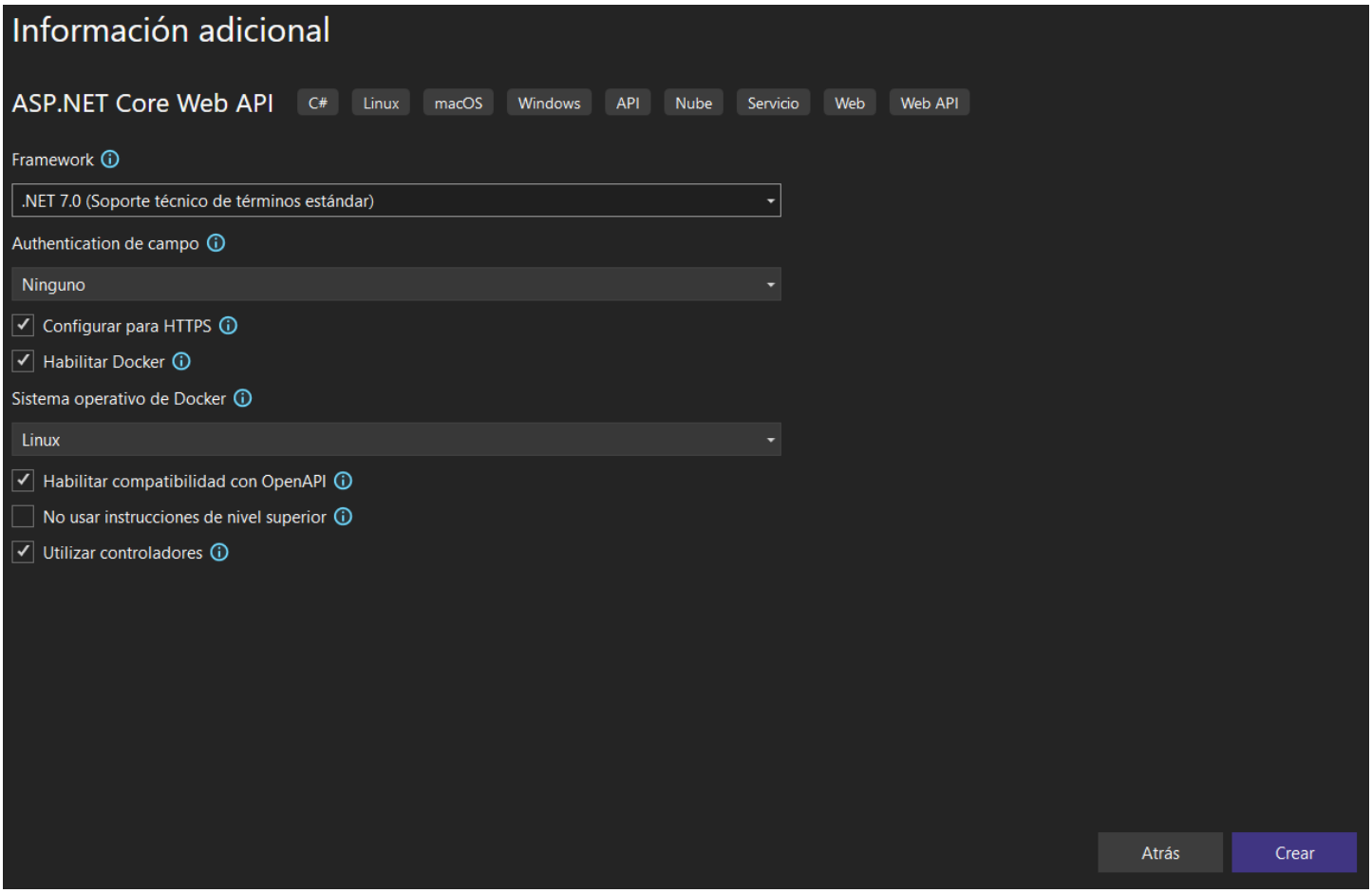
We select the projec template "ASP.NET Core Web API"



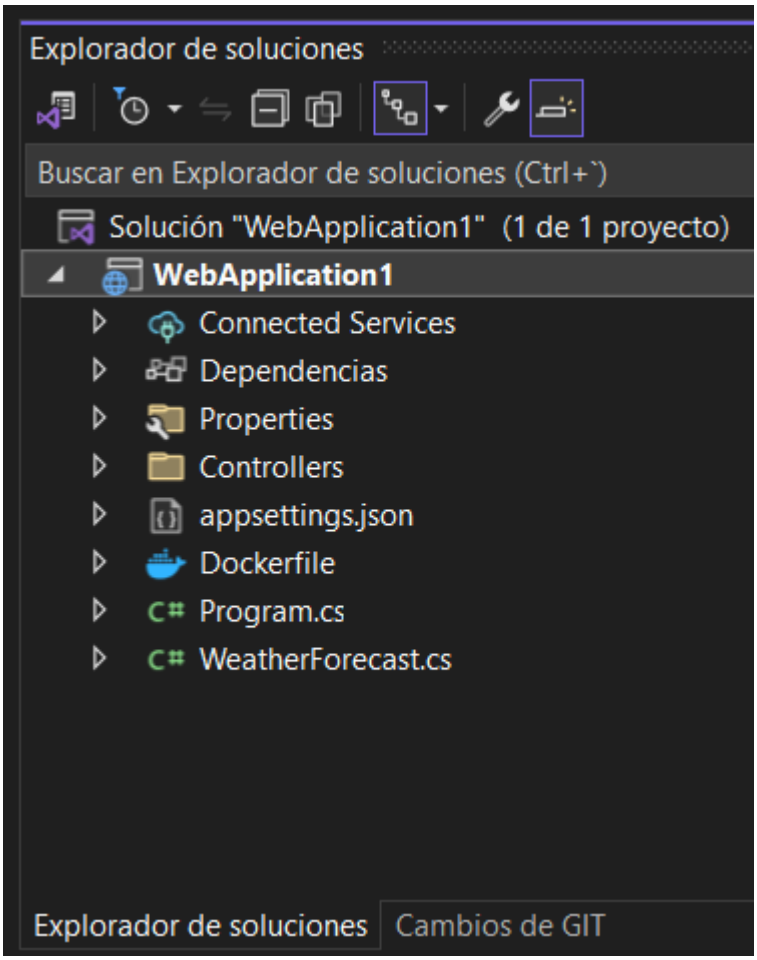
We set the project name and location

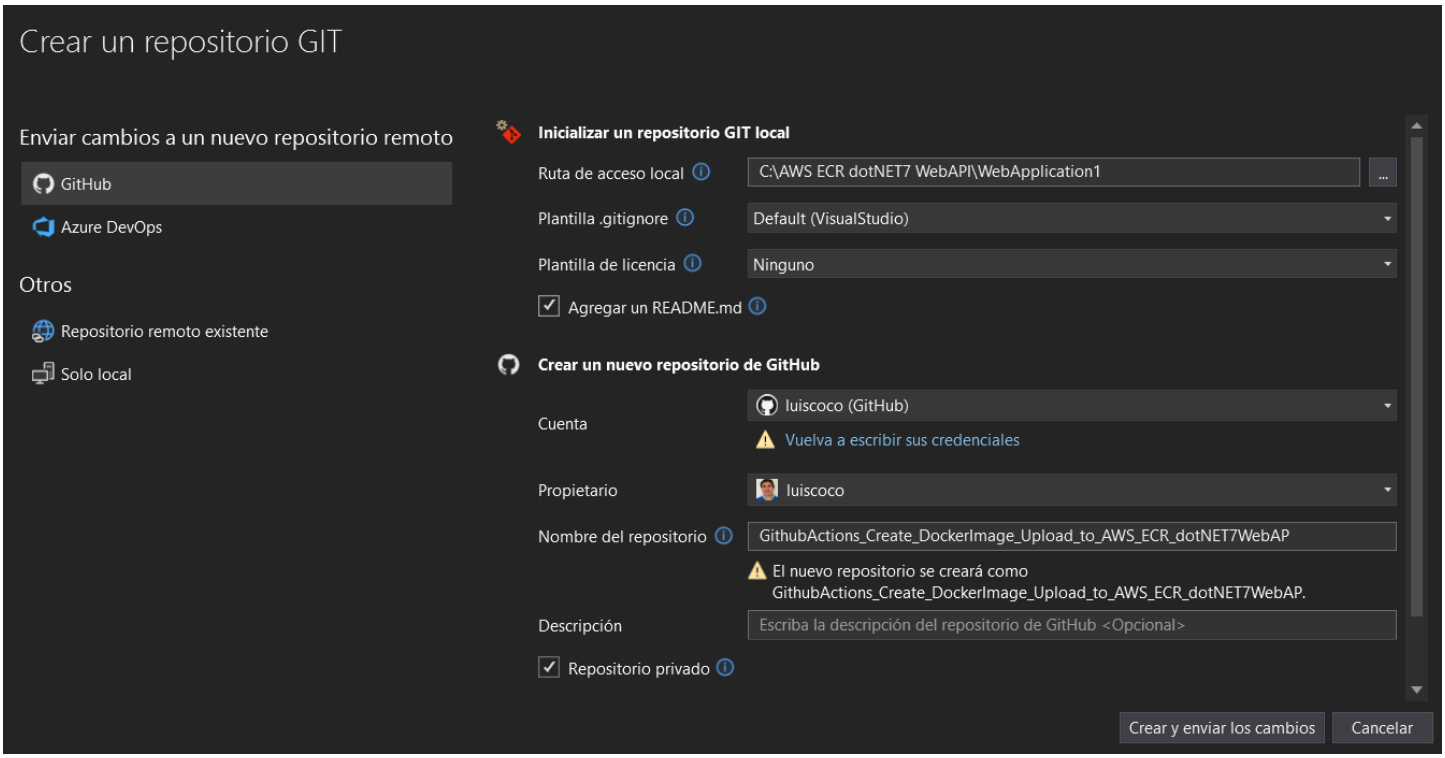


We select the project main options: .NET 7 framework, enable Docker, configure HTTPS, use OpenAPI and user controllers



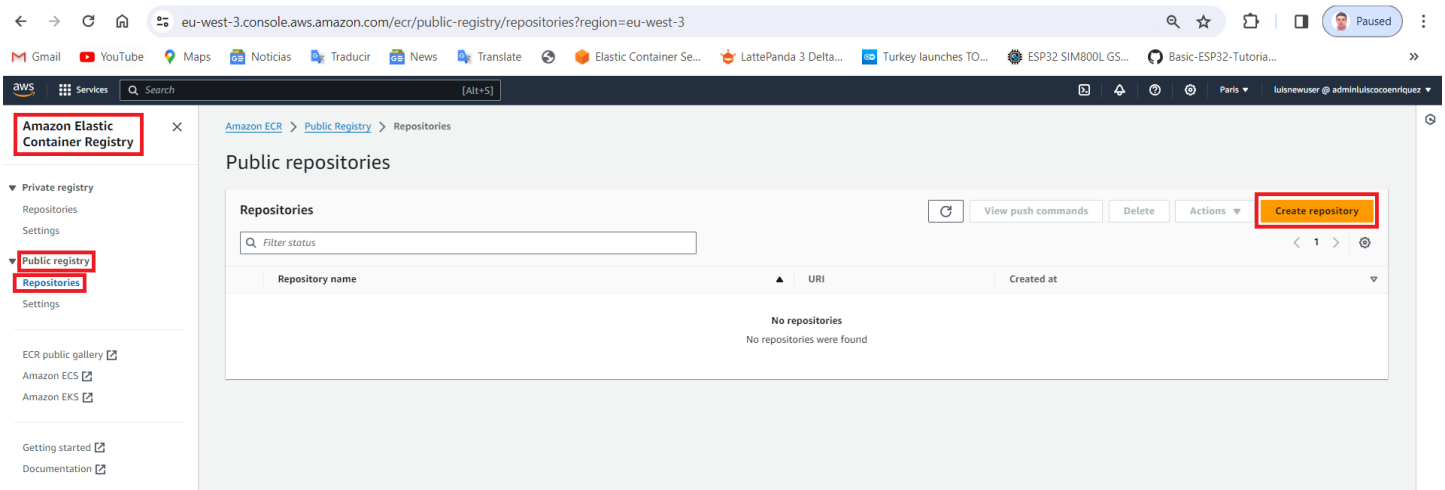
2. Upload your .NET 7 Web API to a Github repo



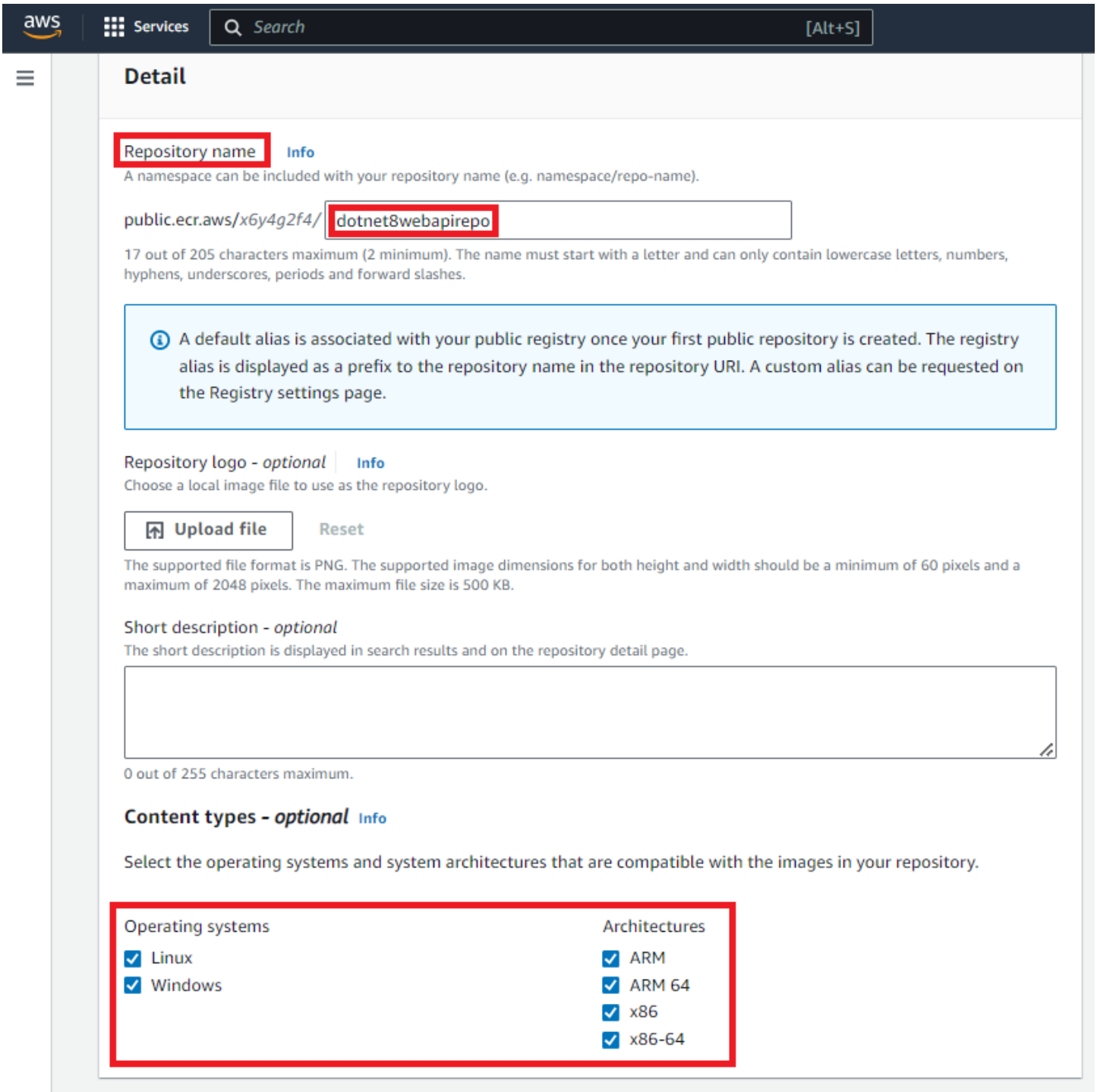


3. Create a AWS ECR Public repo for storing the Docker image

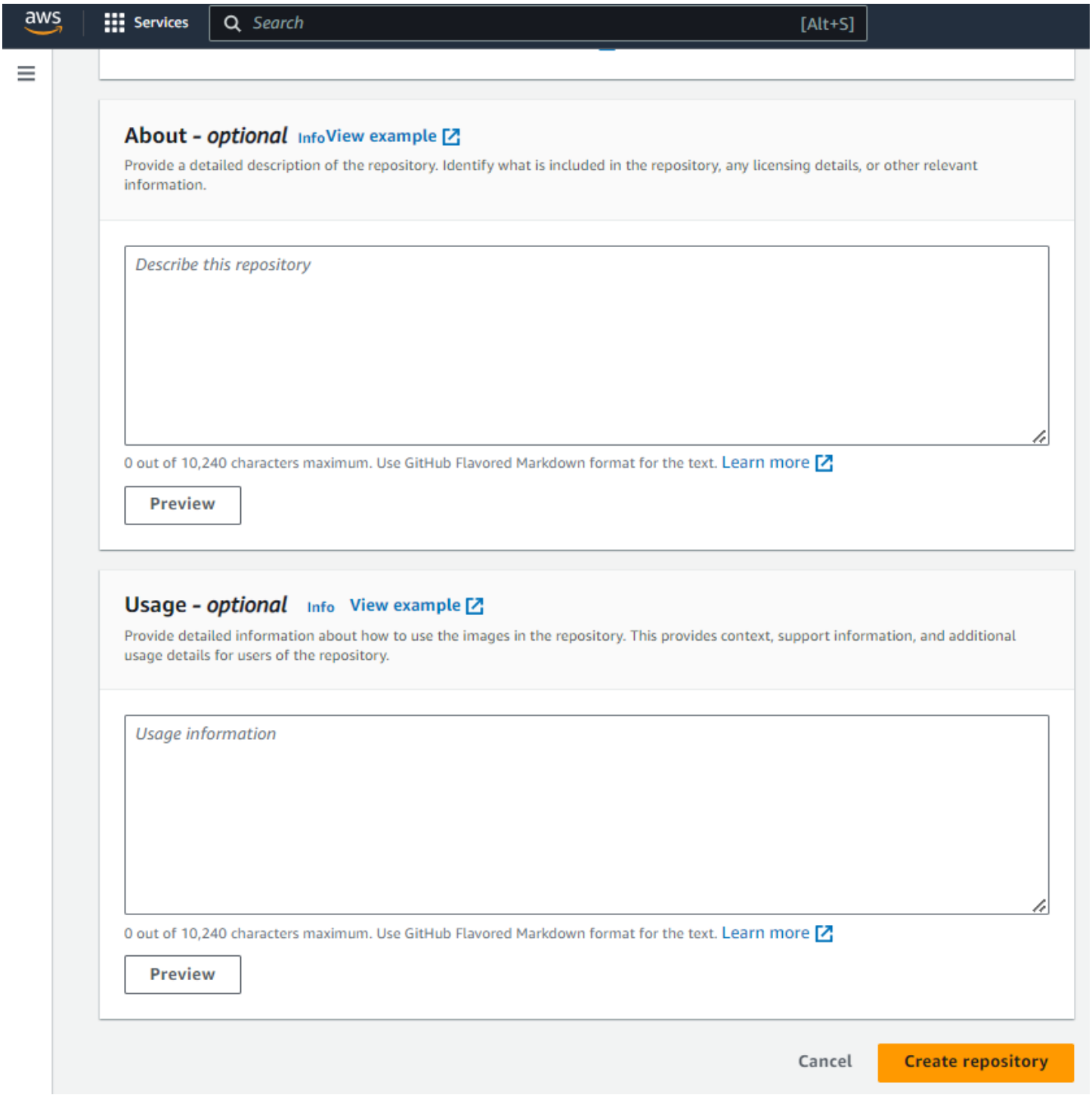
Navigate to the AWS ECR service and create a new Public repo



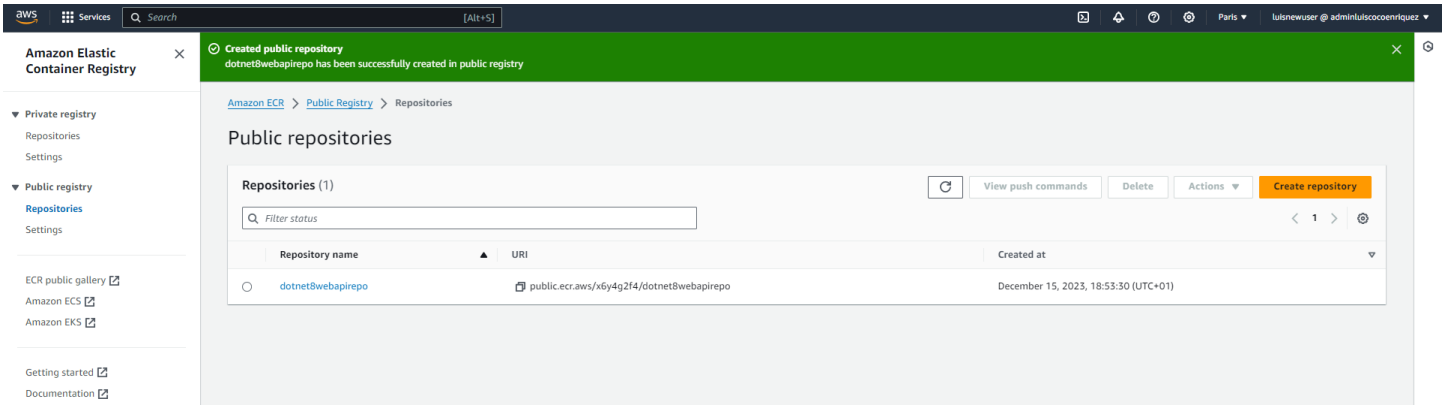
We set the repository name, and the container compatible operating system and architecture



We press the **Create repository** button



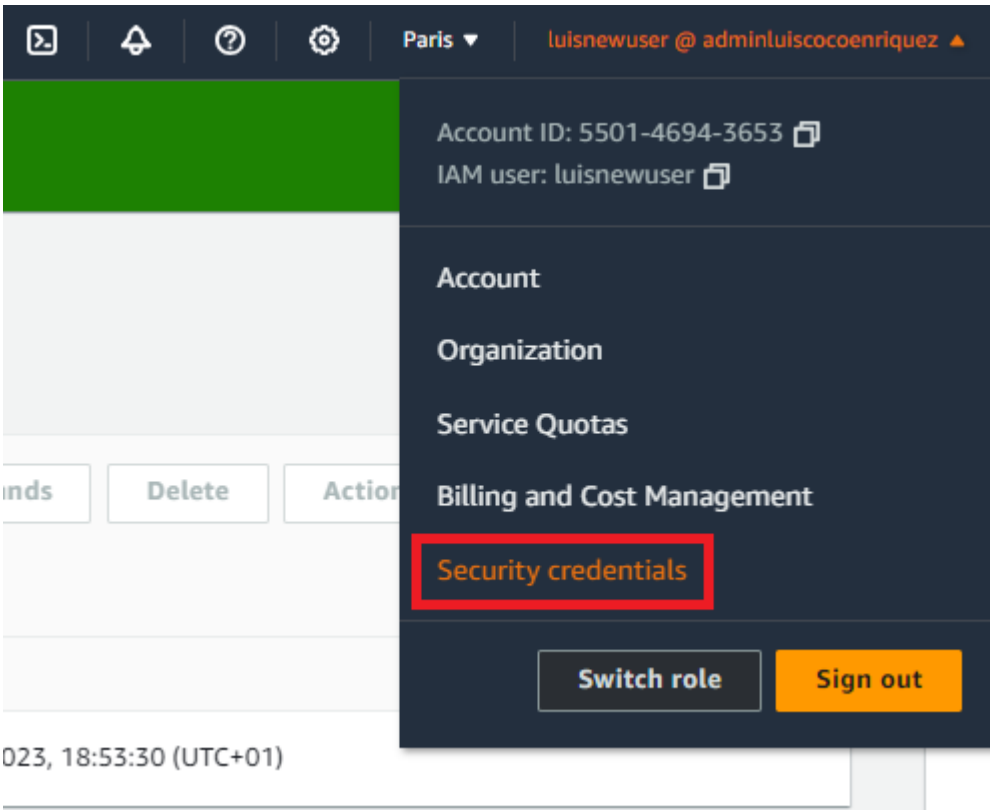
See the new repo in the list



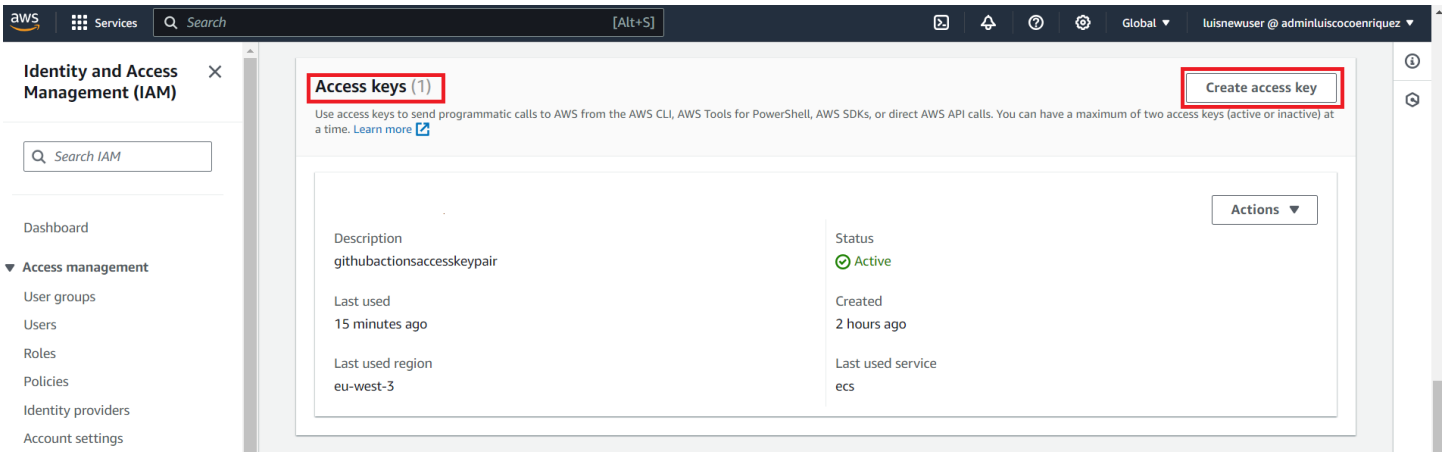
4. Create the Github Secrets to store the AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY

We start this section in **AWS** creating an access key and a secret key

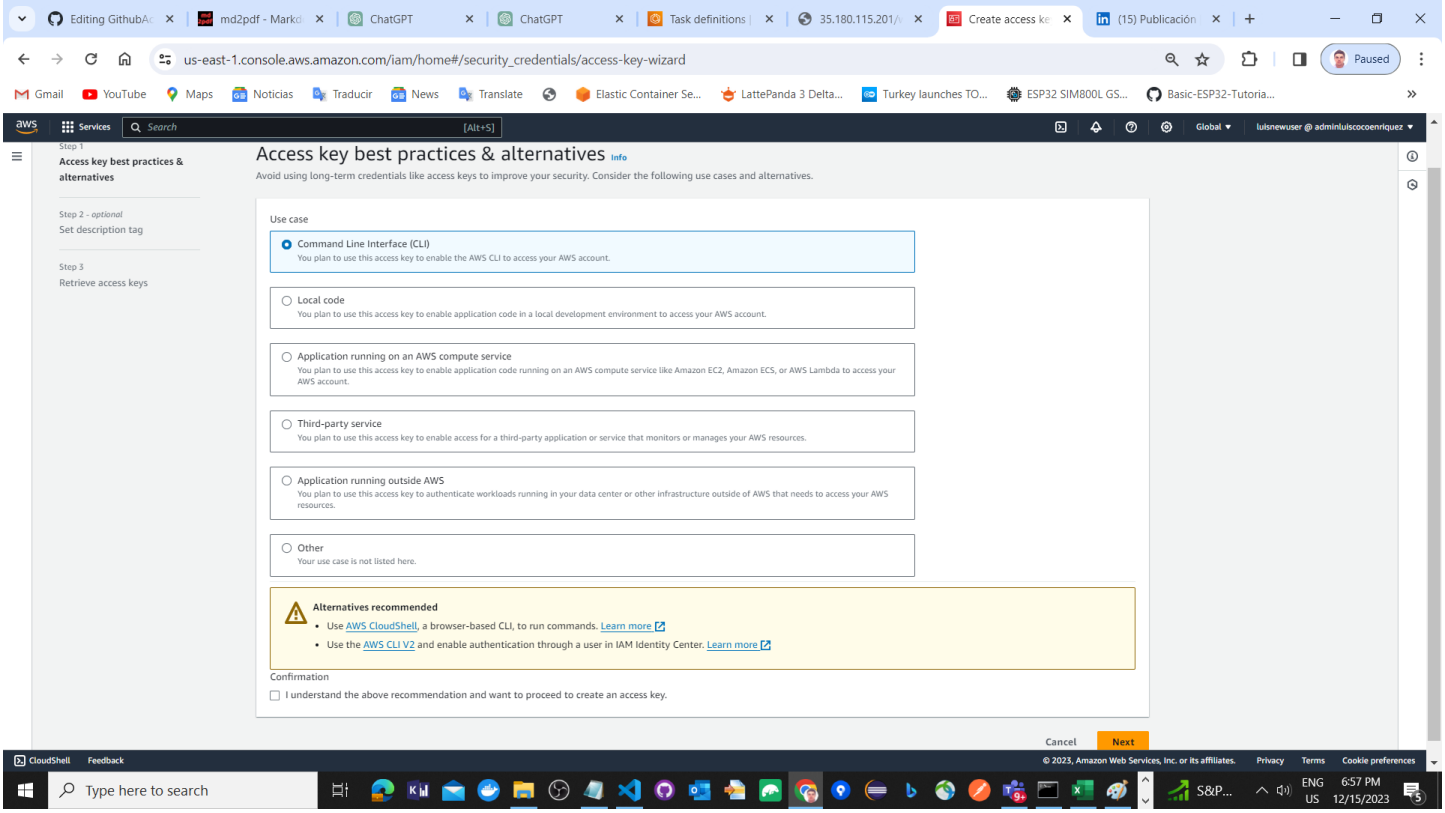
We select the **Security Credentials** option in the top right menu



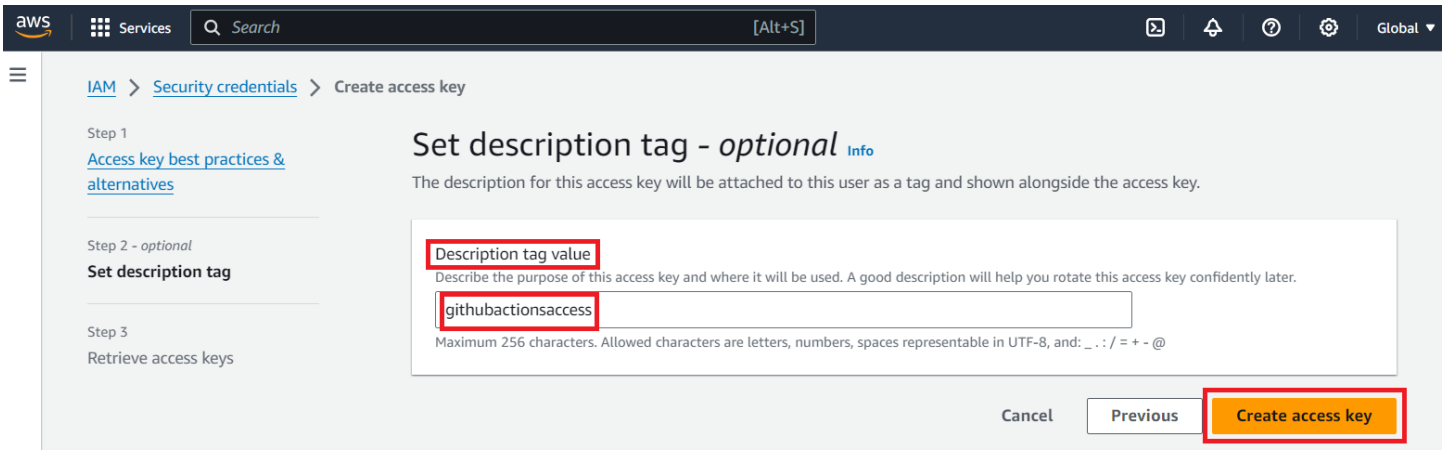
We click on the **Create access key** button



Select Use case **Command Line Interface CLI**, check "I understand the above recommendation and want to proceed to create an access key." and press Next button



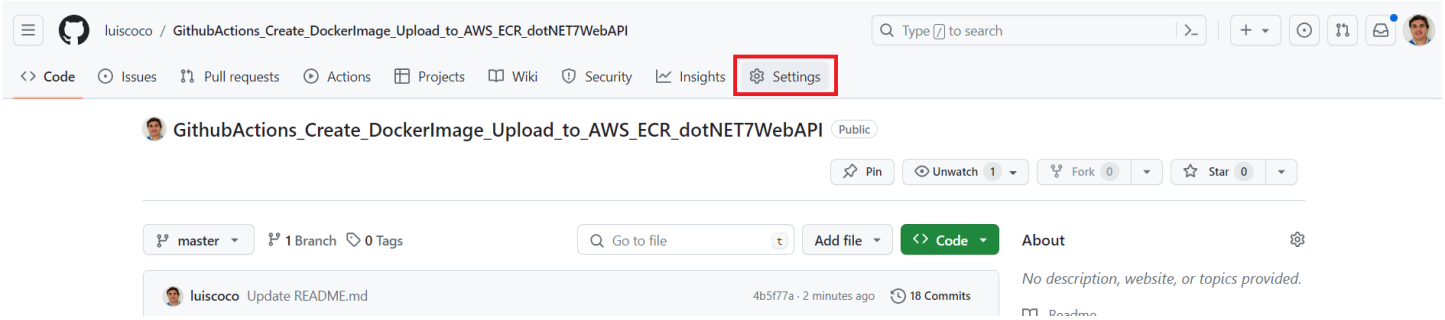
We input the description tag and press the **Create access** key button



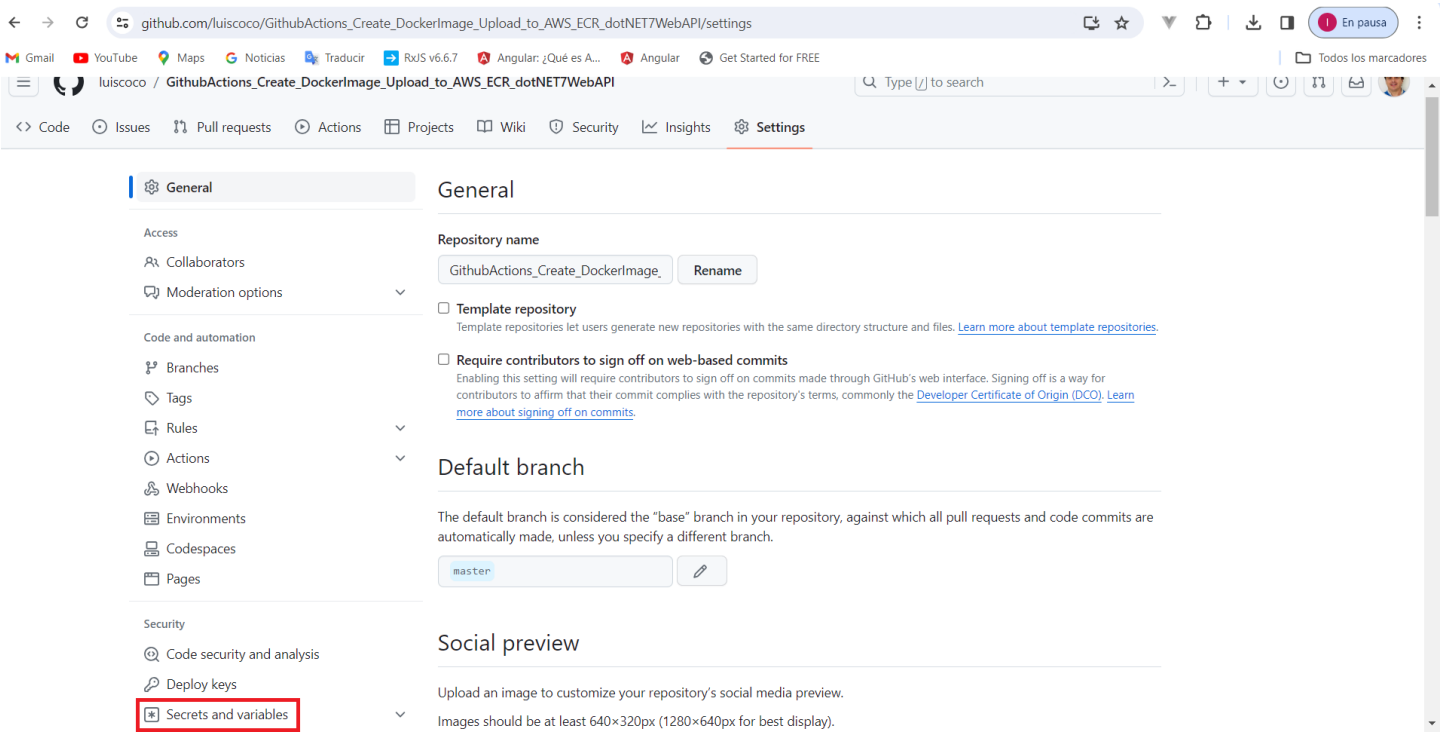
We click on the **Download .csv file** and and Excel file will be donwloaded to our laptop with the access key and secret key

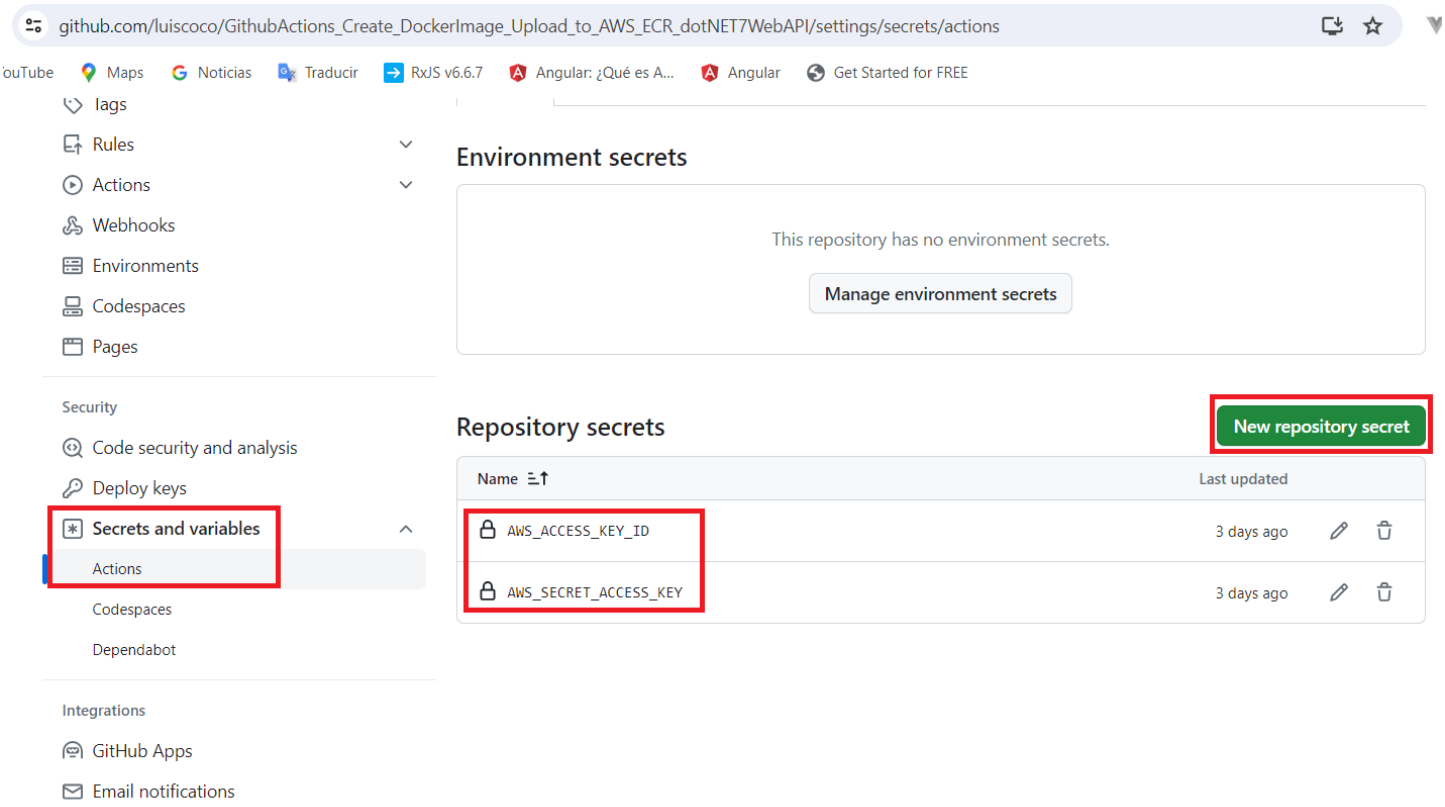
Go to you **Github repo** and create two secrets for stroing the access key and secret key

Press the **Settings** button



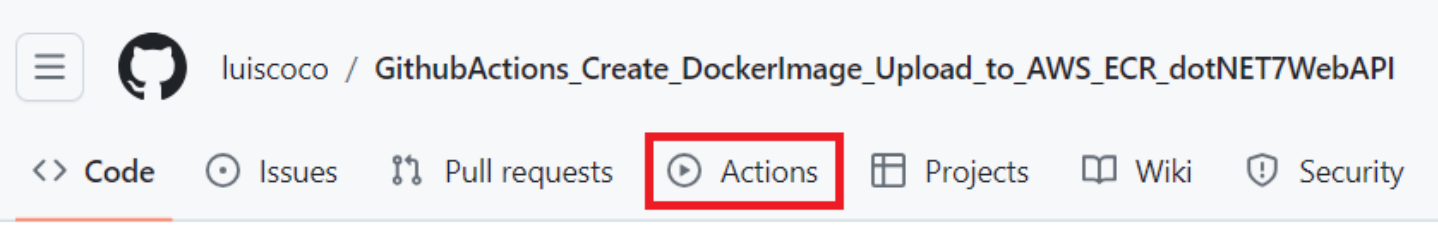
We select **Secrets and variables** and we store **AWS_ACCESS_KEY_ID** and **AWS_SECRET_ACCESS_KEY** in two github repository secrets





5. Create the Github actions workflow

In the application Github repo click on the **Actions** button



We input the **main.yml** file for executing the github action workflow to create a .NET 7 Web API Docker image and upload it to AWS ECR

```yaml
name: Build and Push Docker Image

on:
  push:
    branches: [ master ]

env:
  ECR_REGISTRY: public.ecr.aws/x6y4g2f4  # Your AWS ECR Registry
  IMAGE_NAME: dotnet8webapirepo  # Replace with your image name
  AWS_REGION: us-east-1  # Replace with your AWS region

jobs:
  build_and_push:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2

    - name: Set up Docker Buildx
      uses: docker/setup-buildx-action@v1

    - name: Set up AWS CLI
      uses: aws-actions/configure-aws-credentials@v1
      with:
        aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
        aws-region: ${{ env.AWS_REGION }}

    - name: Login to Amazon ECR
      run: |
        aws ecr-public get-login-password --region ${{ env.AWS_REGION }} | docker login --user
    # New step to delete existing images
    - name: Delete existing images in ECR repository
      run: |
        aws ecr-public describe-images --repository-name ${{ env.IMAGE_NAME }} --region ${{ en

    - name: Build, tag, and push image to Amazon ECR
      run: |
        docker build -t ${{ env.ECR_REGISTRY }}/${{ env.IMAGE_NAME }}:latest .
        docker push ${{ env.ECR_REGISTRY }}/${{ env.IMAGE_NAME }}:latest
```

We press "**Commit changes...**" button

6. How deploy to AWS ECS the Docker image stored in my ECR repo

We create a **ECS Cluster**:

```
aws ecs create-cluster --cluster-name myCluster
```

Create a new file task-definition.json and place it in a local folder: **C:/AWS Task Definition/task-definition.json**

This is the **task-definition.json** file source code

```
{
    "family": "myDotnetApp",
    "executionRoleArn": "arn:aws:iam::550146943653:role/ecsTaskExecutionRole",
    "networkMode": "awsvpc",
    "containerDefinitions": [
        {
            "name": "dotnet8webapi",
            "image": "public.ecr.aws/x6y4g2f4/dotnet8webapirepo:latest",
            "cpu": 1024,
            "memory": 3072,
            "essential": true,
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80
                }
            ],
            "environment": [
                {
                    "name": "ASPNETCORE_ENVIRONMENT",
                    "value": "Development"
                }
            ]
        }
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "cpu": "1024",
    "memory": "3072"
}
```

We create a new **Task Definition**:

```
aws ecs register-task-definition --cli-input-json file://"C:/AWS Task Definition/task-definiti
```

We get my AWS Account default **VPC** name:

```
aws ec2 describe-vpcs ^
  --filters "Name=isDefault,Values=true" ^
  --query "Vpcs[].VpcId" ^
  --output text
```

For example the VPC name output is: **vpc-07d51d92f73354c0b**

With that VPC name we can retrieve the attached **Subnets** names:

```
aws ec2 describe-subnets ^
  --filters "Name=vpc-id,Values=vpc-07d51d92f73354c0b" ^
  --query "Subnets[].{ID:SubnetId,Name:Tags[?Key=='Name']|[0].Value}"
```

For example the Subnet output is: **subnet-0df35048d0d30e90f**

We also get the **Security Group** name with this command:

```
aws ec2 describe-security-groups ^
  --filters "Name=vpc-id,Values=vpc-07d51d92f73354c0b" "Name=group-name,Values=default" ^
  --query "SecurityGroups[].{ID:GroupId,Name:GroupName}" ^
  --output text
```

For example the Security Group output is: **sg-051b9197846af4fe0**

We create a new **Service** with the "aws ecs create-service" command

Do not forget to set the **subnetId** and the **securityGroupId**

```
aws ecs create-service ^
  --cluster myCluster ^
  --service-name myDotnetService ^
  --task-definition myDotnetApp ^
  --launch-type FARGATE ^
  --desired-count 1 ^
  --network-configuration "awsvpcConfiguration={subnets=[subnet-0df35048d0d30e90f],securityGro
```

For accessing the endpotint click on the Public IP address and add the controller name

```
http://IPAddress/controlername
```