Lab Module 5b: GitHub: Application Deployment to Kubernetes

> Estimated Duration: 120 minutes

- Lab Module 5b: GitHub: Application Deployment to Kubernetes
  - Exercise: Configure GitHub
    - Task 1 - Create a GitHub account
    - Task 2 - Create a ssh key
    - Task 3 - Configure your GitHub SSH Key
  - Exercise: Create a GitHub Repository
    - Task 1 - Create a local git repo for the MT3Chained-Web folder
    - Task 2 - Create the GitHub repo for the local MT3Chained-Web repo
  - Exercise: Create the Secrets for the GitHub repo
    - Task 1 - Create a service principal and grant it access to the AKS resource group
    - Task 2 - Create the GitHub Action Secret to manage the AKS cluster and the Azure Key Vault
    - Task 3 - Create the GitHub Actions Secrets for the Azure Container Registry Authentication
  - Exercise: Create a Basic CI/CD Pipeline with GitHub Actions
    - Task 1 - Create the GitHub Action
    - Task 2 - Add in the workflowyaml the latest image tag
  - Exercise: Add a Kubernetes Deploy Step to the Workflow YAML
    - Task 1 - Deploy the new workflow yaml
  - Exercise: A Complex Microservices CI/CD Pipelines Using Helm and Azure Key Vault
    - Background
    - Task 1 - Create repos for the remaining microservices
    - Task 2 - Initialize git, check in the source code and push it to GitHub (For all the repos, excluding the NodeJS step and web repo)
    - Task 3 - Add the same secrets you added in the Exercise: Create the Secrets for the GitHub repo
    - Task 4 - Create a GitHub Token
    - Task 5 - Create a new GitHub Action for the step repos

- Task 6 - Configure the service principal to access the Key Vault created in Lab 4, Exercise 1, Task 2 to be used by your workflows
- Task 7 - Update the Build Pipelines to store image tags in Azure Key Vault
- Task 8 - Build the Pipelines for the Web and NodeJS microservice
- Task 9 - Build a single Release Pipelines that uses Helm to update the AKS cluster
- Task 10 - Push the changes

Exercise: Configure GitHub

Use this configuration to prepare GitHub for either one of the following exercises.

Use the same ACR, Azure Key Vault and AKS cluster you created for **Lab 4**.

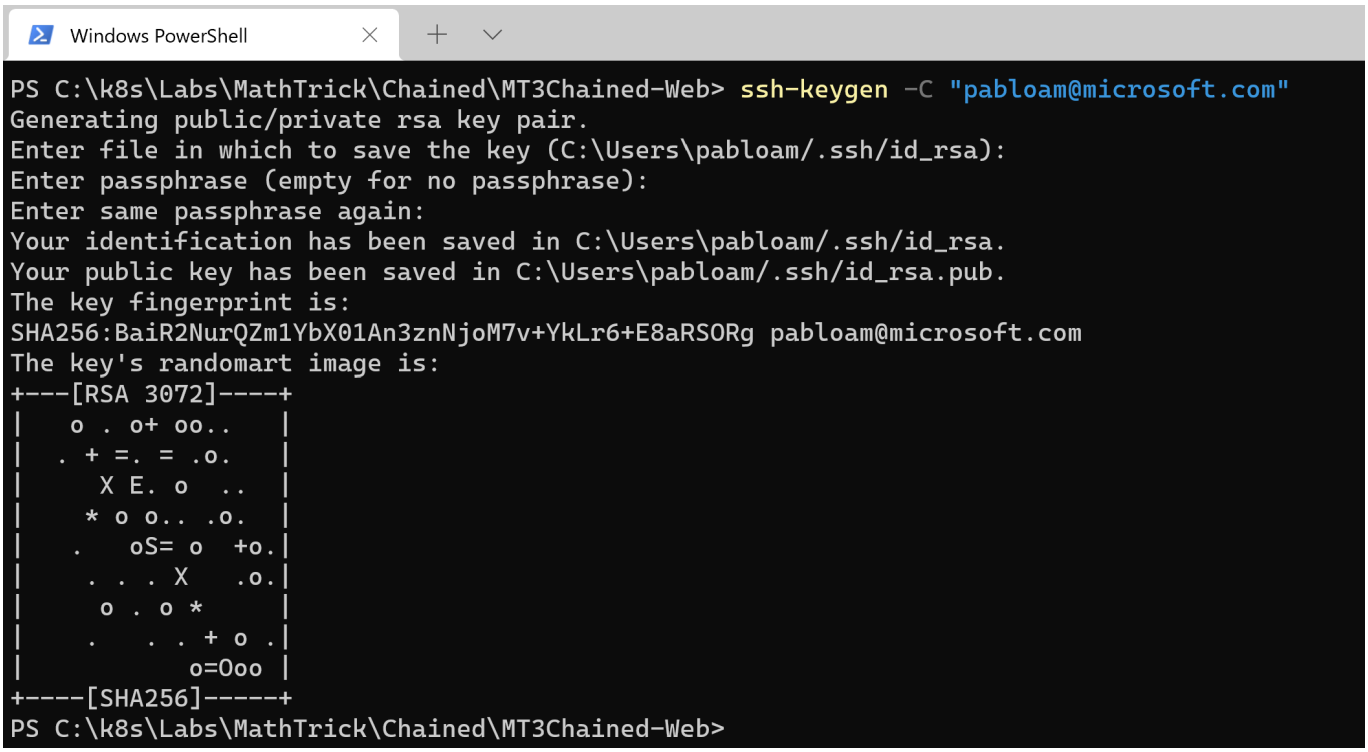
Task 1 - Create a GitHub account

1. If you don't already have access to a GitHub account, navigate to: https://github.com
2. Click on sign up.
3. Use an email of your choice to create an account.
4. Login to your new account.

Task 2 - Create a ssh key

1. Generate a SSH key (use your email) and **DO NOT ENTER A PASSPHRASE**.

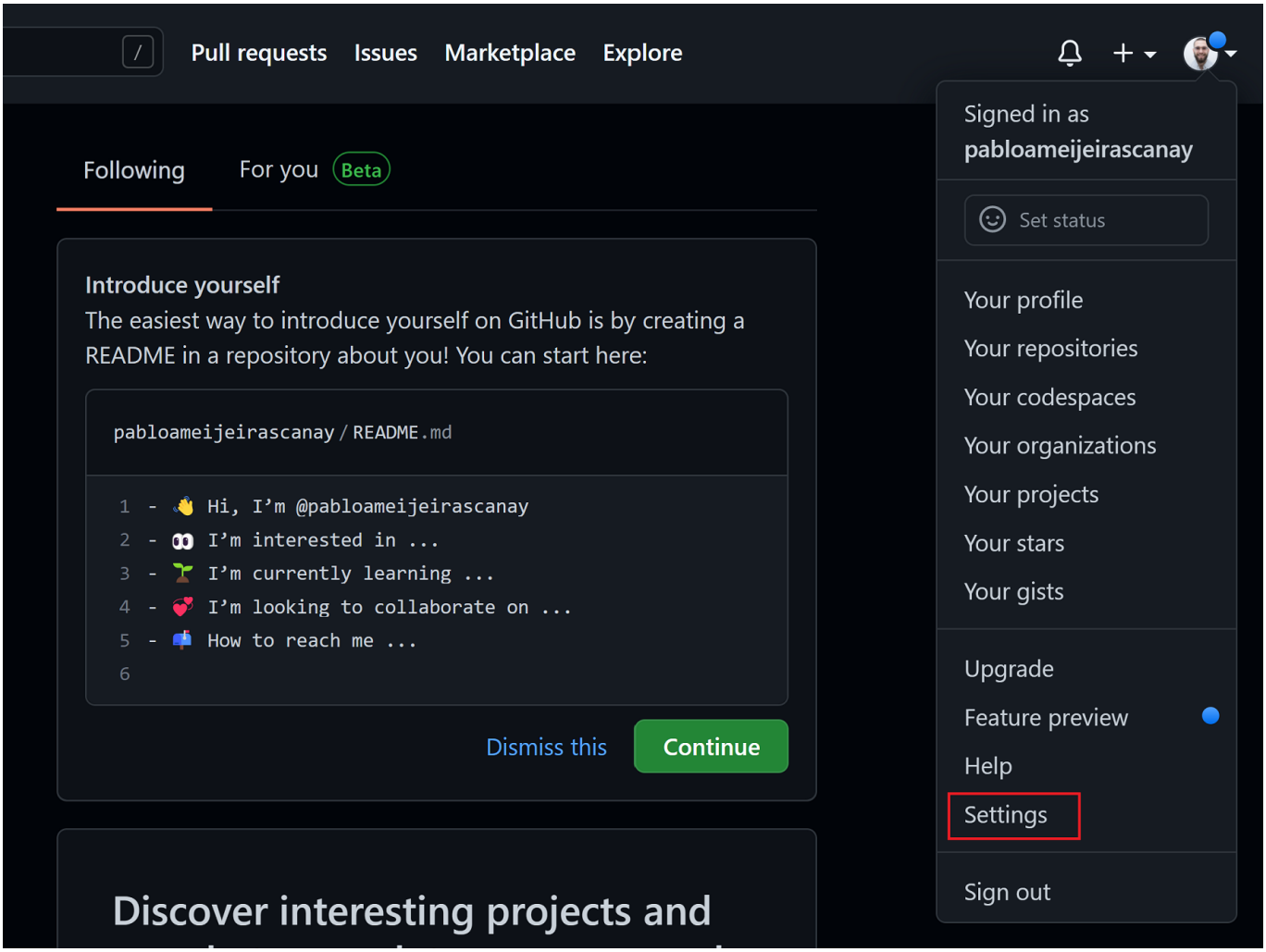
```
ssh-keygen -C "myemail@mycompany.com"
```



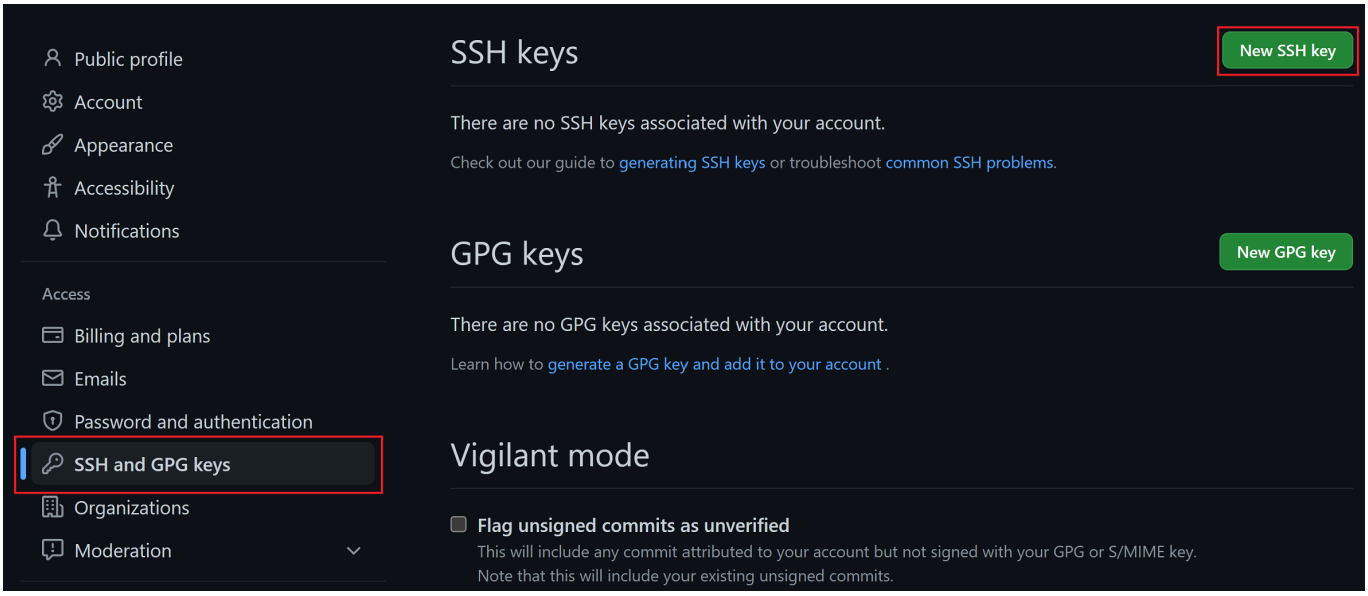
Task 3 - Configure your GitHub SSH Key

1. If you haven't already done so, login to your GitHub account.

2. Select your avatar in the upper right of the user interface and then click on **Settings** in the menu that appears.



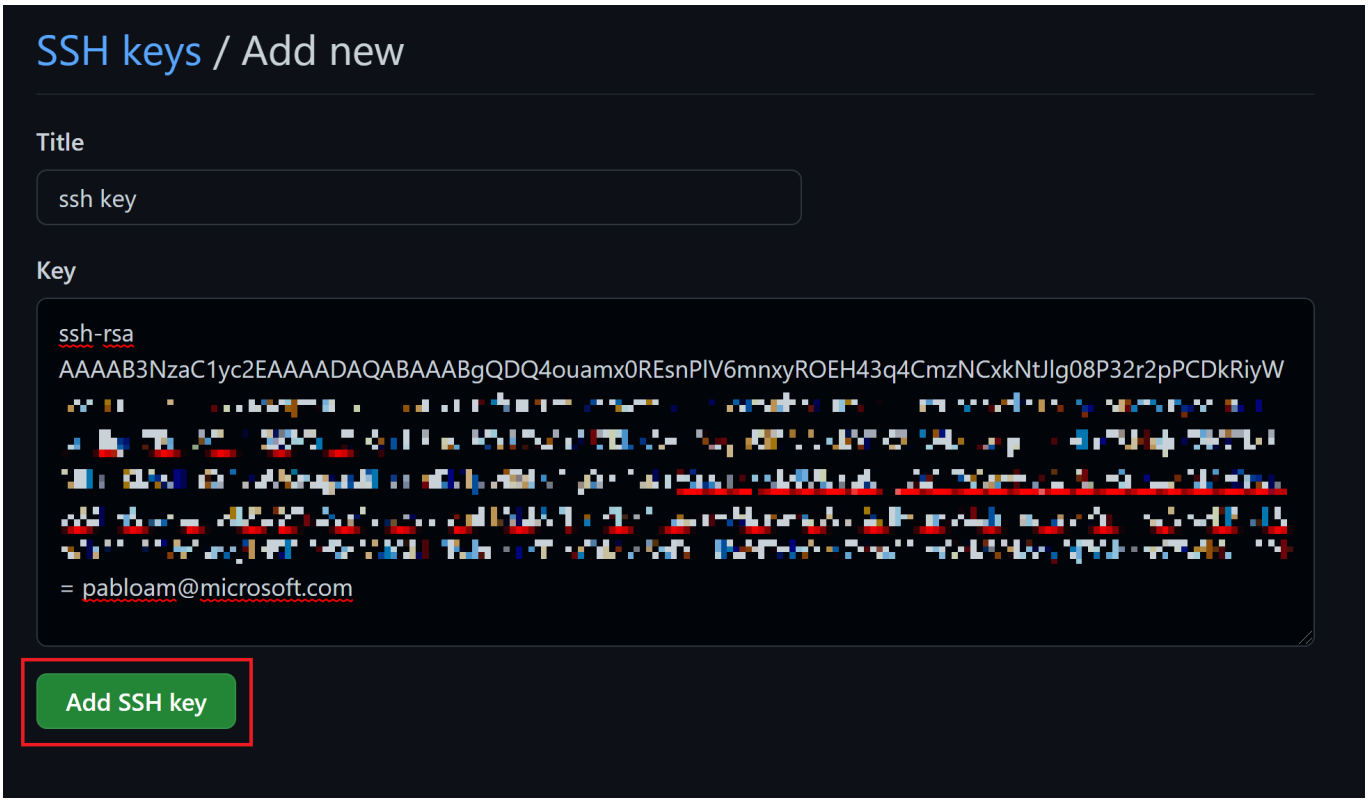
3. Click on **SSH and GPG keys** and **New SSH key**.



4. Copy the content of the public key (for example, id_rsa.pub) that you generated in your local.

```
cat C:\Users\pabloam\.ssh\id_rsa.pub
```

5. Name it as ssh key in the **Title section** and paste the content in the **Key** section. Finally, click on **Add SSH key**.



Exercise: Create a GitHub Repository

Task 1 - Create a local git repo for the MT3Chained-Web folder

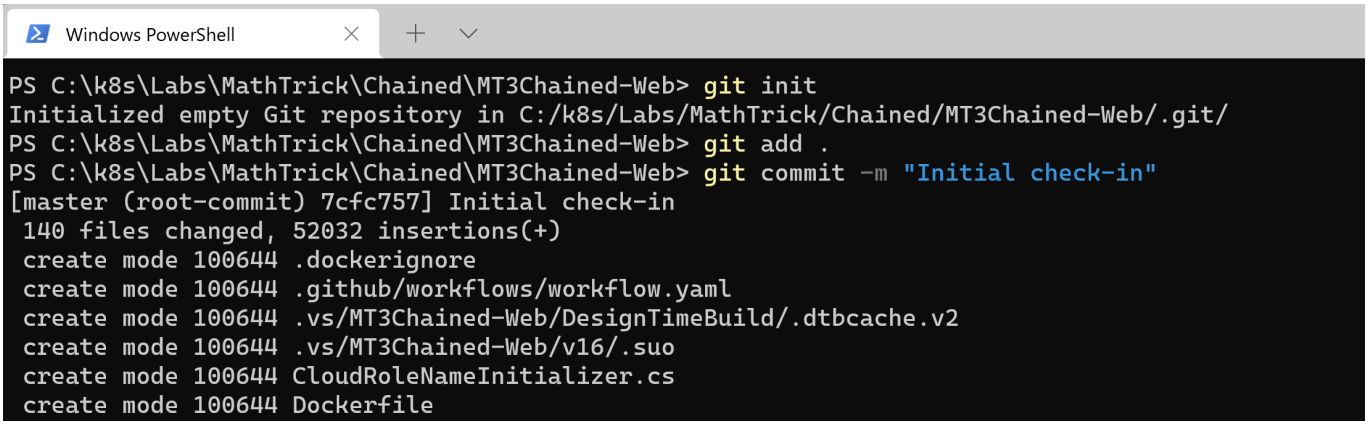
1. Open the Windows Terminal and change into the **C:\k8s\Labs\MathTrick\Chained\MT3Chained-Web** folder.

```
cd C:\k8s\Labs\MathTrick\Chained\MT3Chained-Web
```

2. Initialize git and check in the source code.

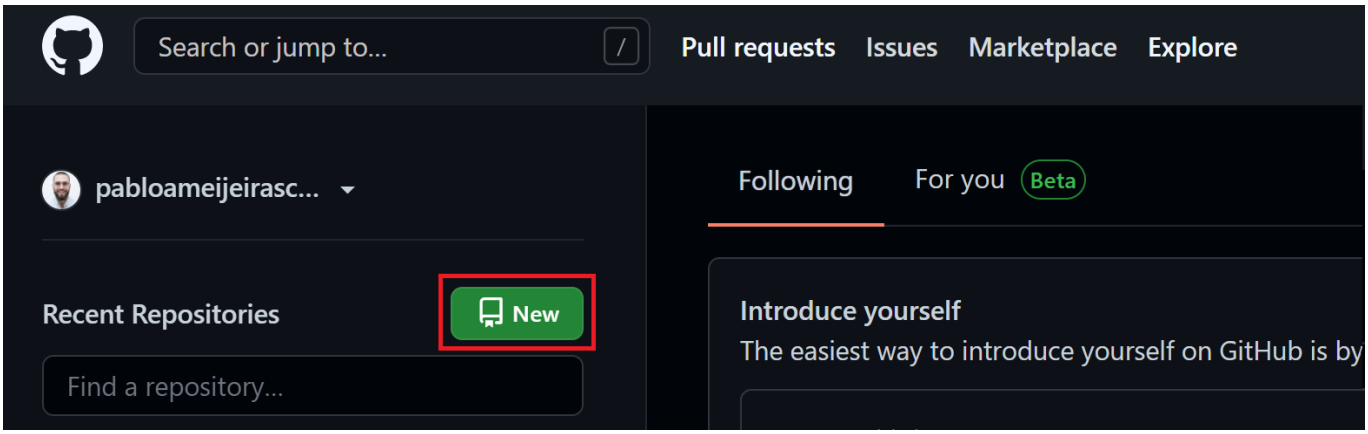
```
git init
git add .
git commit -m "Initial check-in"
```

The shell should look like this:



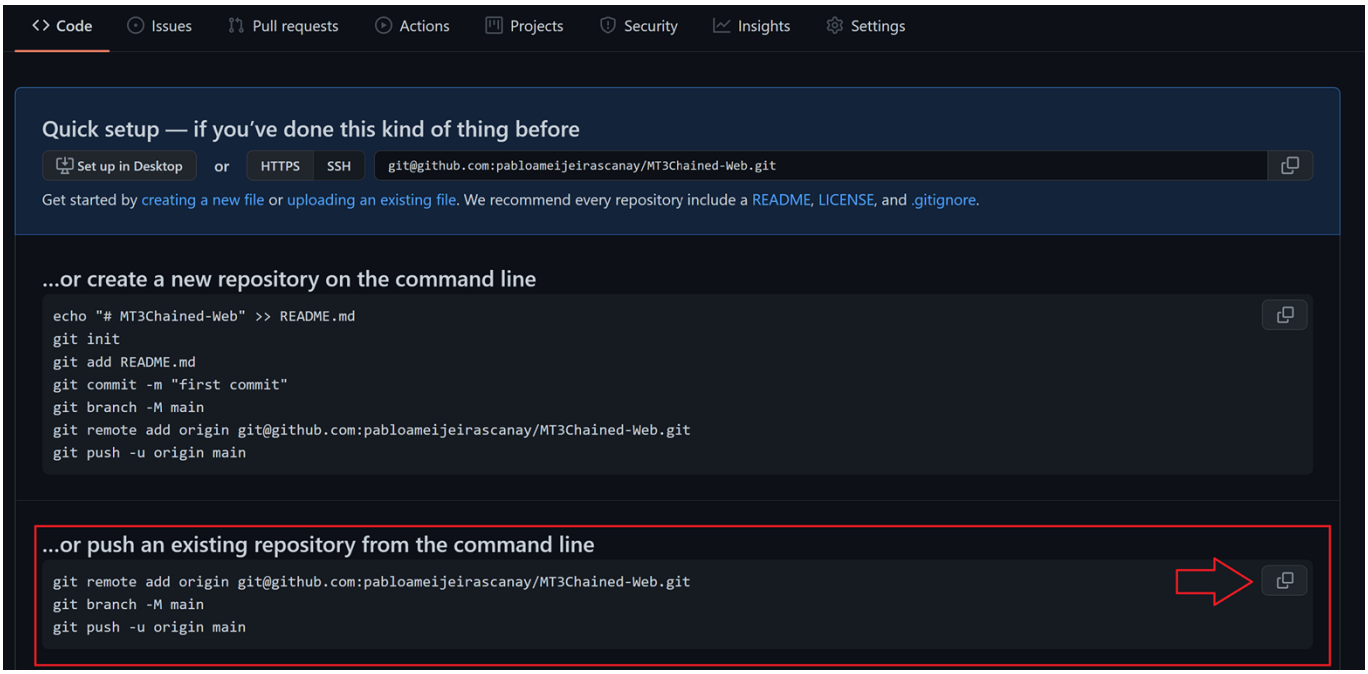
Task 2 - Create the GitHub repo for the local MT3Chained-Web repo

1. Go to the GitHub principal page and Select **New** from the left side.



2. Create a new repository called "MT3Chained-Web". **CHECK "Private"** and click on **Create Repository** button.

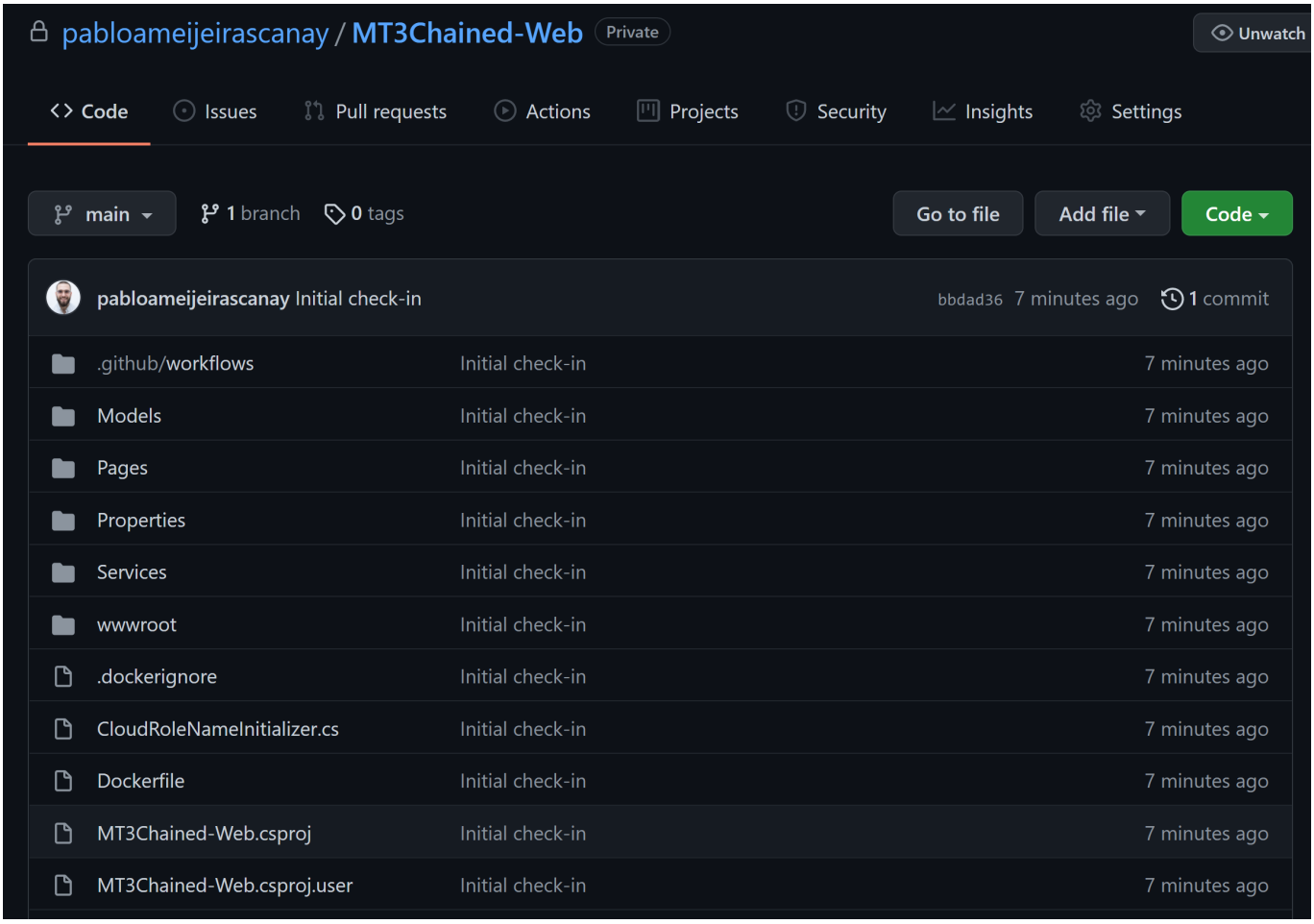
3. In the next window copy the following commands.



4. Go back to your Windows Terminal and paste those contents and enter yes.

```
PS C:\k8s\Labs\MathTrick\Chained\MT3Chained-Web> git remote add origin git@github.com:pabloameijeirascanay/MT3Chained-Web.git
PS C:\k8s\Labs\MathTrick\Chained\MT3Chained-Web> git branch -M main
PS C:\k8s\Labs\MathTrick\Chained\MT3Chained-Web> git push -u origin main
The authenticity of host 'github.com (140.82.121.4)' can't be established.
ED25519 key fingerprint is SHA256:+DiY3wvvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvCOqU.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
Enumerating objects: 97, done.
Counting objects: 100% (97/97), done.
Delta compression using up to 8 threads
Compressing objects: 100% (92/92), done.
Writing objects: 100% (97/97), 937.45 KiB | 1.86 MiB/s, done.
Total 97 (delta 15), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (15/15), done.
To github.com:pabloameijeirascanay/MT3Chained-Web.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
PS C:\k8s\Labs\MathTrick\Chained\MT3Chained-Web>
```

5. Refresh the GitHub page, you should see your local files.



Exercise: Create the Secrets for the GitHub repo

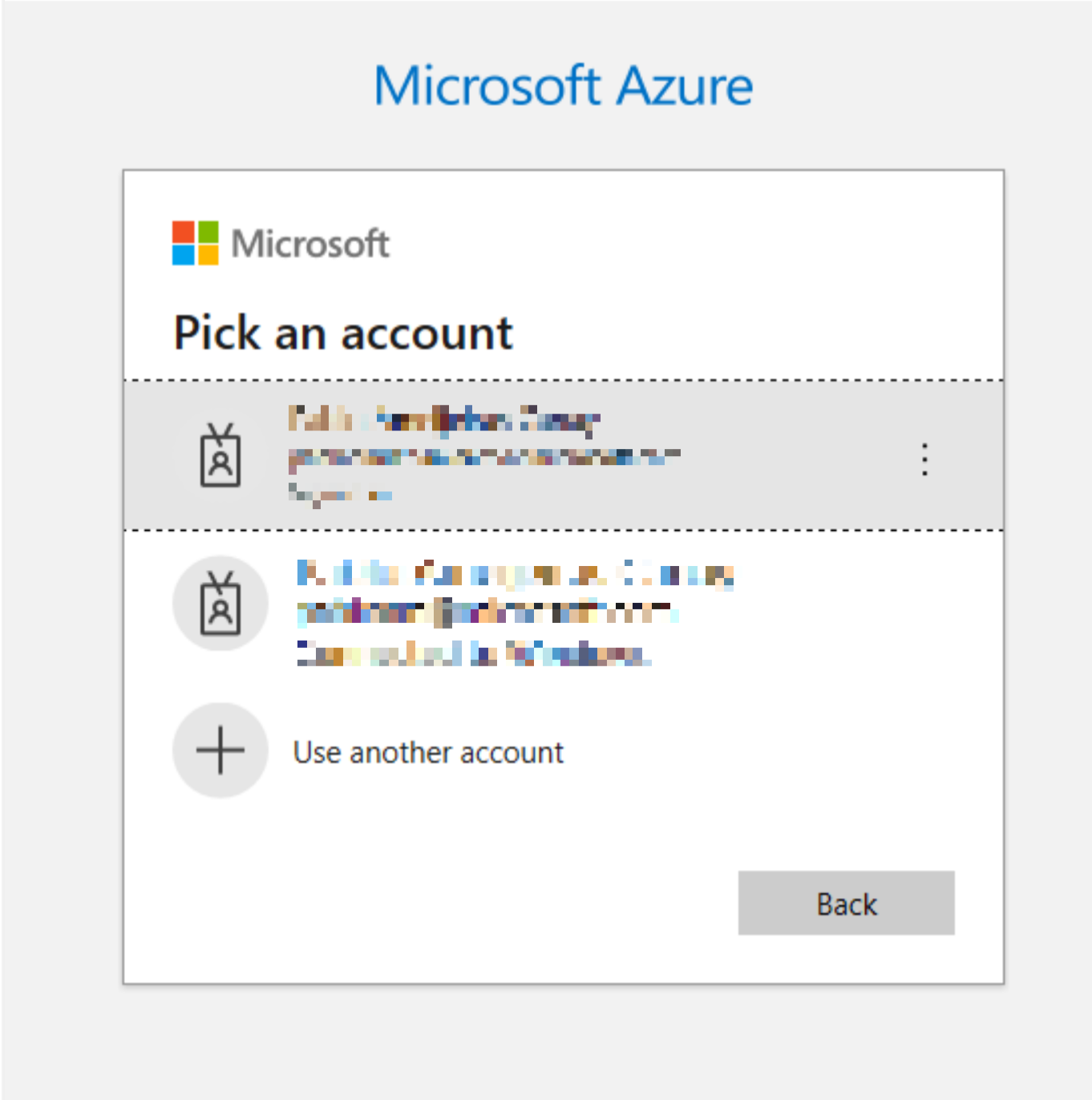
Task 1 - Create a service principal and grant it access to the AKS resource group

We will create the secrets with Service Principal values, so first we must create a Service Principal.

1. Login to Azure with PowerShell.

```
Connect-AzAccount
```

2. Use your user email and password.



3. Create a Service Principal and assign it the Contributor role in your resource group (AKS Resource Group). You just have to copy the following Azure PowerShell code and paste it in the Windows Terminal.

```
$resourceGroupId = (Get-AzResourceGroup -Name $AKS_RESOURCE_GROUP).ResourceId

$SP_NAME="sp-aks-$($YOUR_INITIALS)$SUFFIX"

Write-Host "Service Principal Name:" $SP_NAME

$azureContext = Get-AzContext

$servicePrincipal = New-AzADServicePrincipal `
```

```
  -DisplayName $SP_NAME `
  -Role Contributor `
  -Scope $resourceGroupId

  $output = @{
     clientId = $($servicePrincipal.ApplicationId)
     clientSecret = $([System.Net.NetworkCredential]::new('',
  $servicePrincipal.Secret).Password)
     subscriptionId = $($azureContext.Subscription.Id)
     tenantId = $($azureContext.Tenant.Id)
  }

  $output | ConvertTo-Json
```

4. The script output should look like this:

```
PS C:\k8s\Labs\MathTrick\Chained\MT3Chained-Web> $output | ConvertTo-Json
{
    "subscriptionId":  "[redacted]",
    "tenantId":  "[redacted]",
    "clientId":  "[redacted]",
    "clientSecret":  "[redacted]"
}
PS C:\k8s\Labs\MathTrick\Chained\MT3Chained-Web>
```

5. Copy the script output in a .txt file, since it will be used in the next steps.

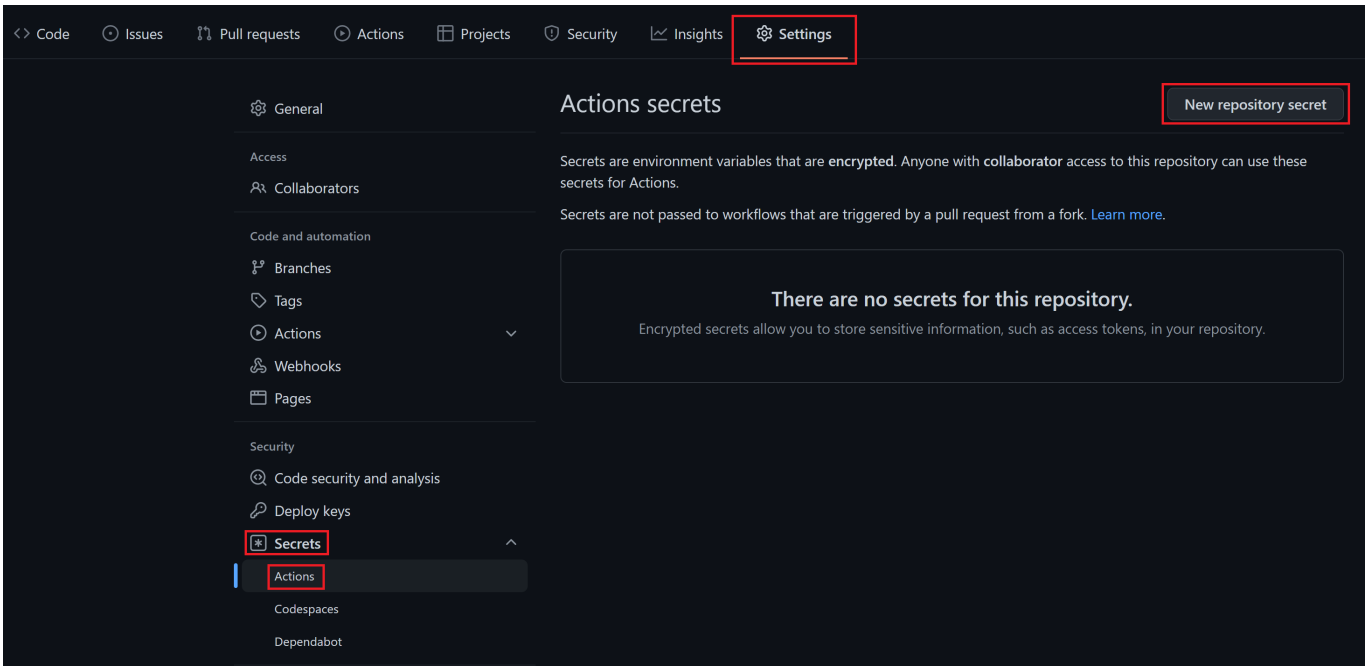
You've created the service principal. Next, create secrets in the GitHub Repository.

Task 2 - Create the GitHub Action Secret to manage the AKS cluster and the Azure Key Vault

1. In your browser, navigate to your GitHub repository (**MT3Chained-Web**).

2. Select Settings > Secrets > Actions.

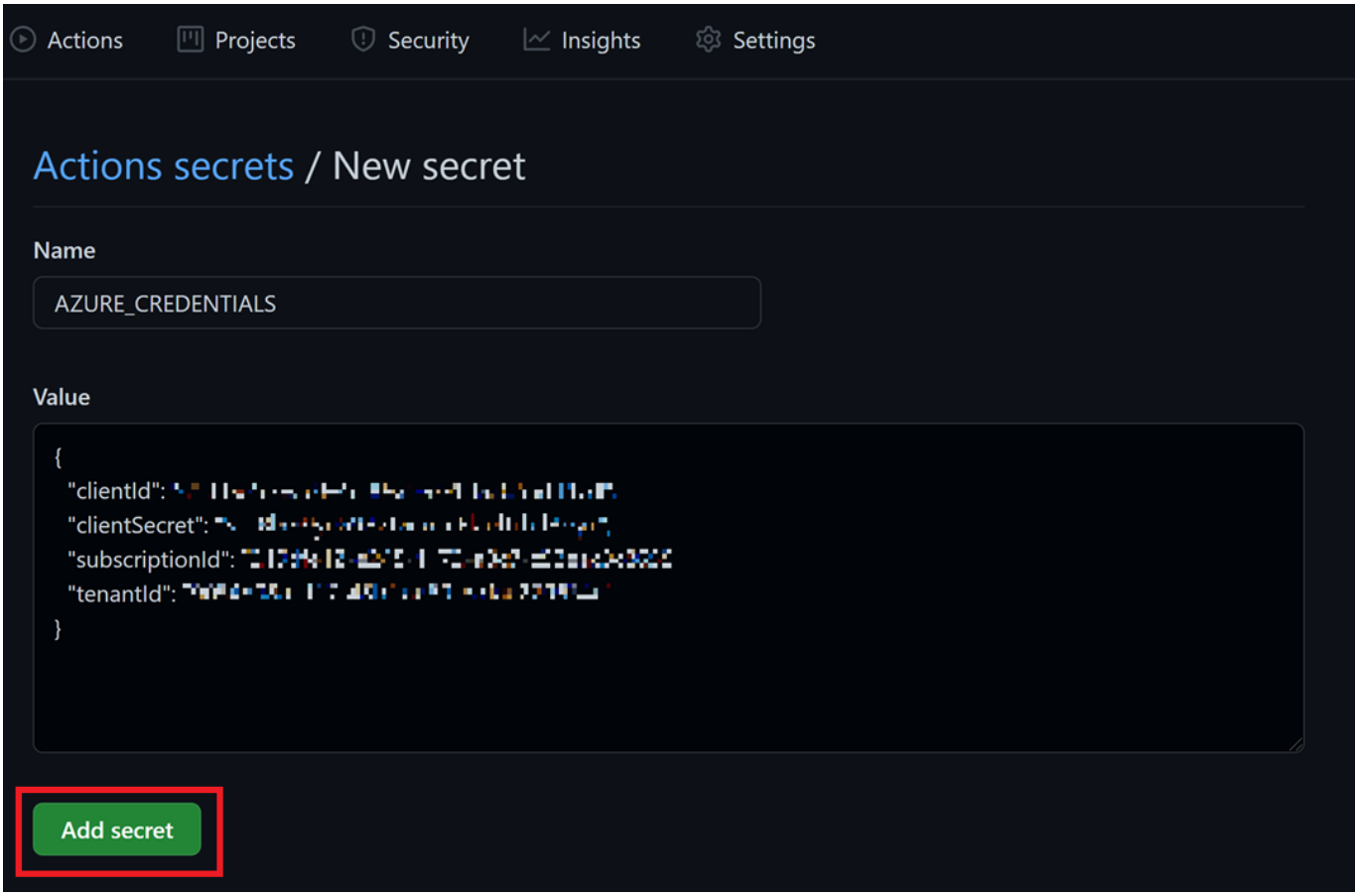
3. Select New repository secret.



4. Name the secret as **AZURE_CREDENTIALS**.

5. In the **Value** field, paste the JSON object that you copied in the .txt file in the previous section.

6. Click **Add secret**.



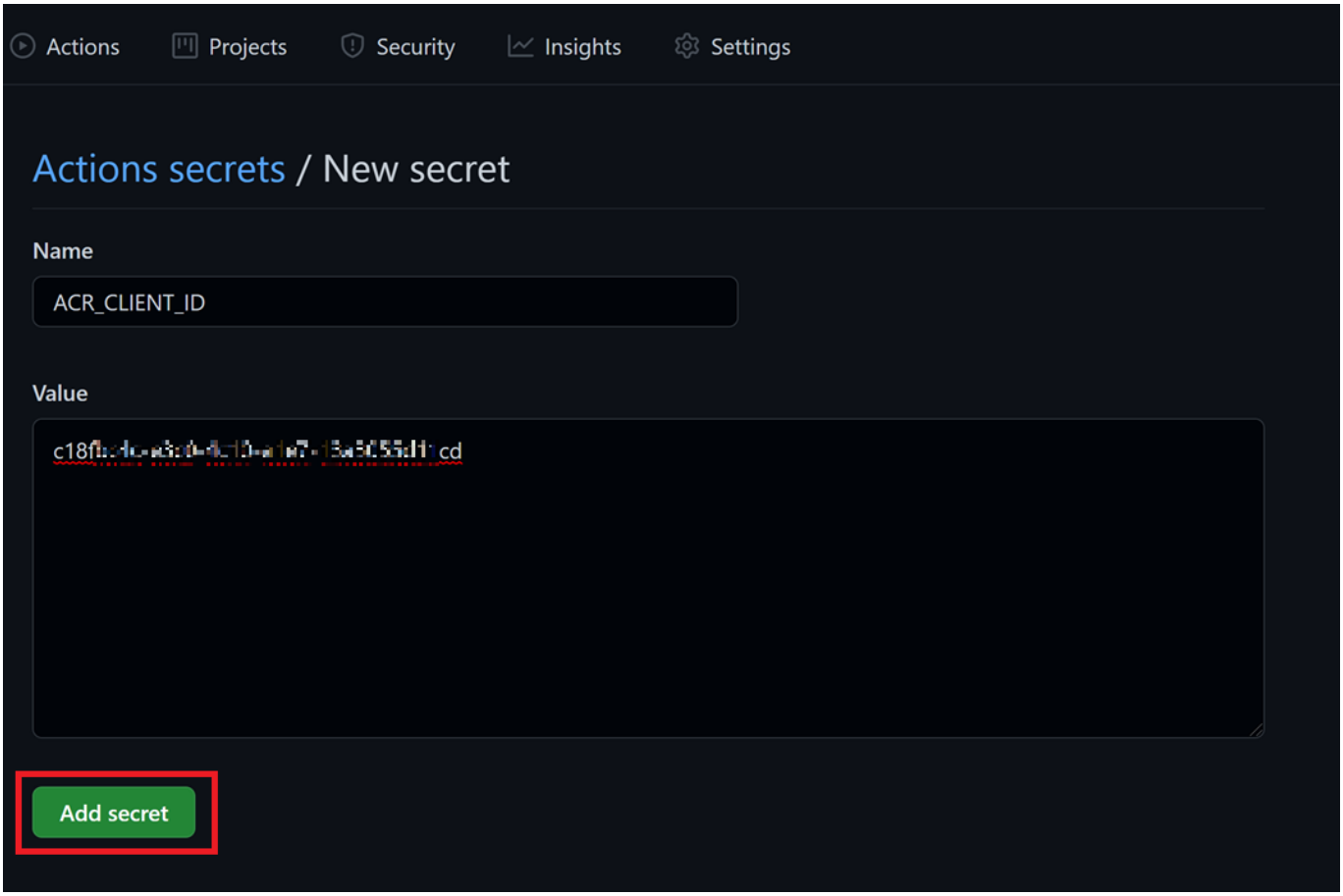
Task 3 - Create the GitHub Actions Secrets for the Azure Container Registry Authentication

1. Select New repository secret.

2. Name the secret as **ACR_CLIENT_ID**.

3. In the **Value** field, paste the **"clientId"** value of the JSON object that you copied in the previous section in the .txt file.

4. Select **Add secret**.

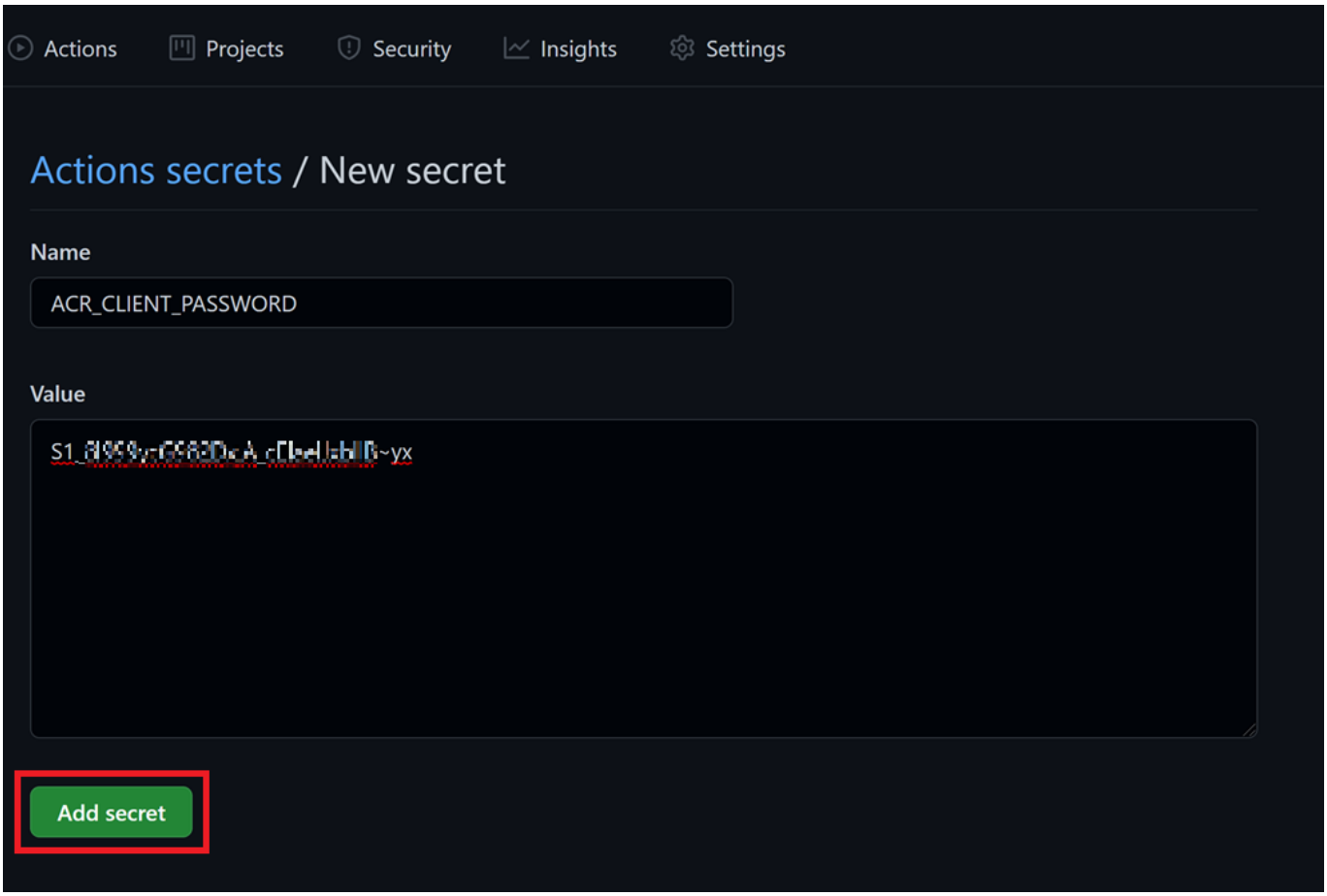


5. Select New repository secret.

6. Name the secret as **ACR_CLIENT_PASSWORD**.

7. In the **Value** field, paste the **"clientSecret"** value of the JSON object that you copied in the previous section in the .txt file.

8. Click **Add secret**.



Exercise: Create a Basic CI/CD Pipeline with GitHub Actions

Task 1 - Create the GitHub Action

1. To create a GitHub Action, you need to have a **workflow.yaml** in the **C:\k8s\Labs\MathTrick\Chained\MT3Chained-Web\.github\workflows** directory. Now there is a file **workflow-web1.yaml** in the **C:\k8s\Labs\Module5\workflows** directory with the following content:

> This workflow build, tag, and push the web image to your Azure Container Registry.

```yaml
name: Math Trick 3 - Chained-Web-CI

on:
  push:
    branches:
      - main

env:
  APP_NAME: <appName>
  LOGIN_SERVER: <acrServerName>

jobs:

  build:
    runs-on: ubuntu-latest
    steps:
```

```
      - name: Check Out Repo
        uses: actions/checkout@v2

      # Connect to Azure Container Registry (ACR)
      - name: ACR login
        uses: azure/docker-login@v1
        with:
          login-server: ${{ env.LOGIN_SERVER }}
          username: ${{ secrets.ACR_CLIENT_ID }}
          password: ${{ secrets.ACR_CLIENT_PASSWORD }}

      # Container build and push to a Azure Container Registry (ACR)
      - name: Build, tag, and push image to ACR
        uses: docker/build-push-action@v2
        with:
          context: ./
          file: Dockerfile
          push: true
          tags: |
            ${{ env.LOGIN_SERVER }}/${{ env.APP_NAME }}:${{ github.run_number }}
          ## <latestTag> ##
```

> Take in to account that all the workflow.yaml created in this lab will run whenever you push a change in the main branch of the GitHub repository.

```
on:
  push:
    branches:
      - main
```

2. Obtain the Azure Container Registry Server Name.

```
$ACR_SERVER_NAME=(az acr show -n $ACR_NAME --query loginServer -o tsv)
```

3. Create the **.github\workflows** folder in the path **C:\k8s\Labs\MathTrick\Chained\MT3Chained-Web**.

```
New-Item -Path "C:\k8s\Labs\MathTrick\Chained\MT3Chained-Web\.github\workflows" -
ItemType Directory
```

4. Replace the **workflow-web1.yaml** file with the correct values and create the right file **workflow.yaml** in the right path.

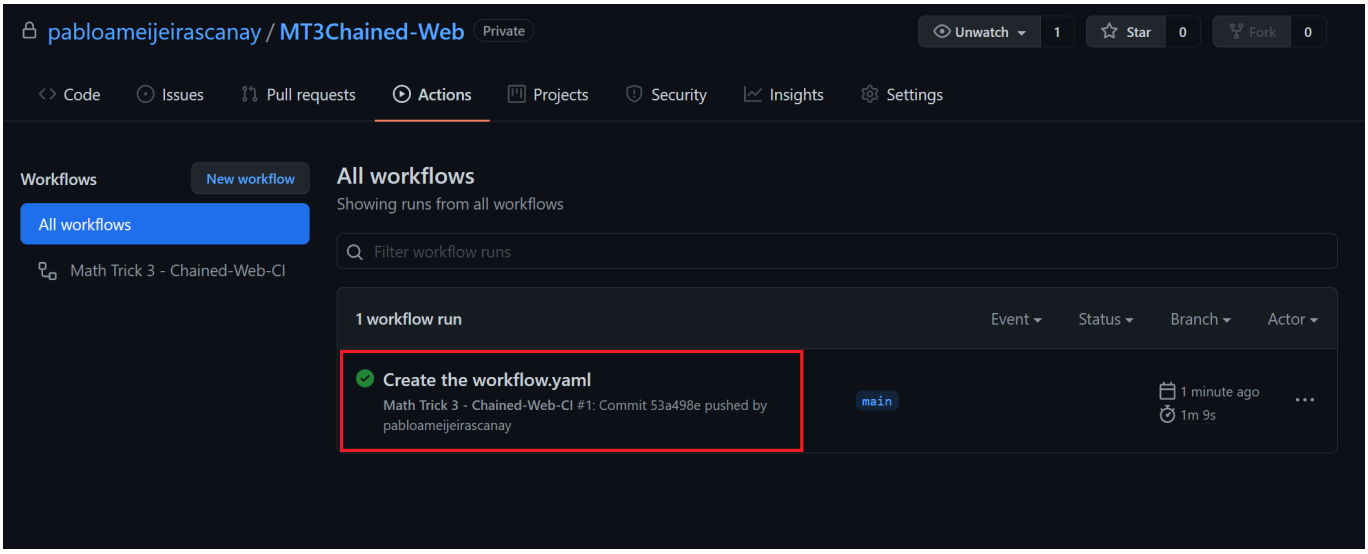
```
$Path1="C:\k8s\Labs\Module5\workflows"
$Path2="C:\k8s\Labs\MathTrick\Chained\MT3Chained-Web\.github\workflows"
```

```
$original_file = (Join-Path $Path1 workflow-web1.yaml)
$destination_file = (Join-Path $Path2 workflow.yaml)
(Get-Content $original_file) | Foreach-Object {
    $_ -replace "<appName>", "mt3chained-web" `
       -replace "<acrServerName>", $ACR_SERVER_NAME
    } | Set-Content $destination_file
```

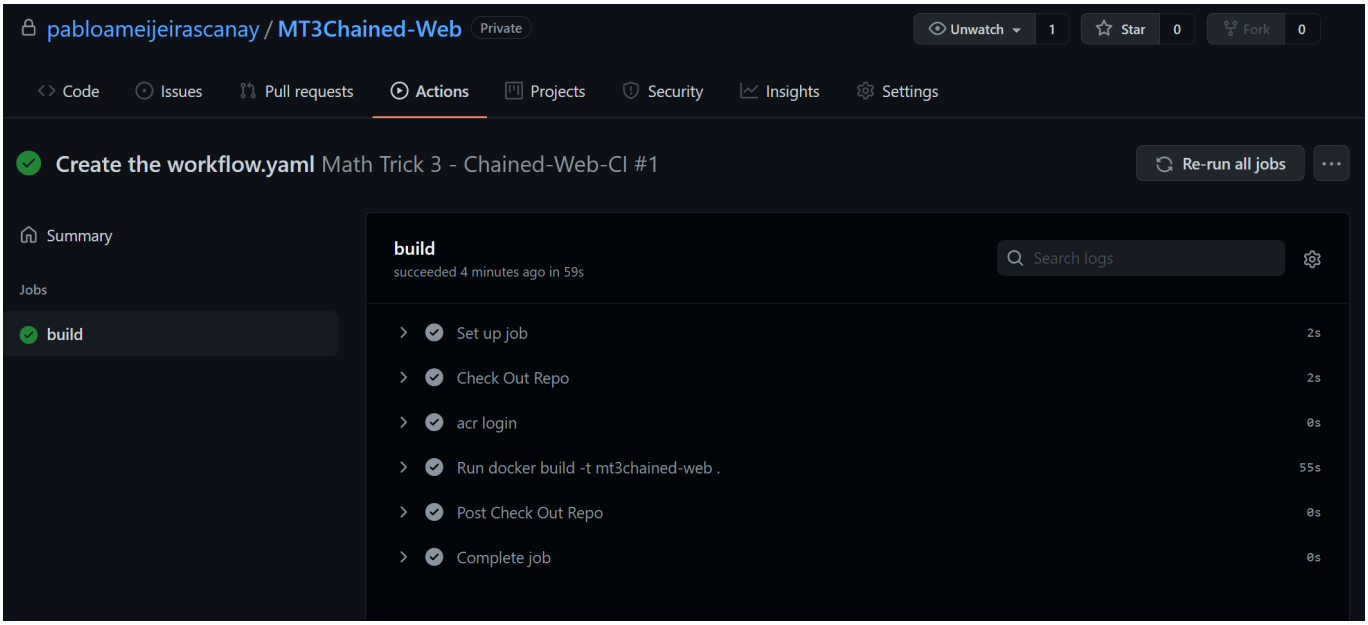
5. Push the changes to your GitHub Repository.

```
git add .
git commit -m "Create the workflow.yaml"
git push
```

6. Now you can see your GitHub Action in the Actions tab of your Repository.

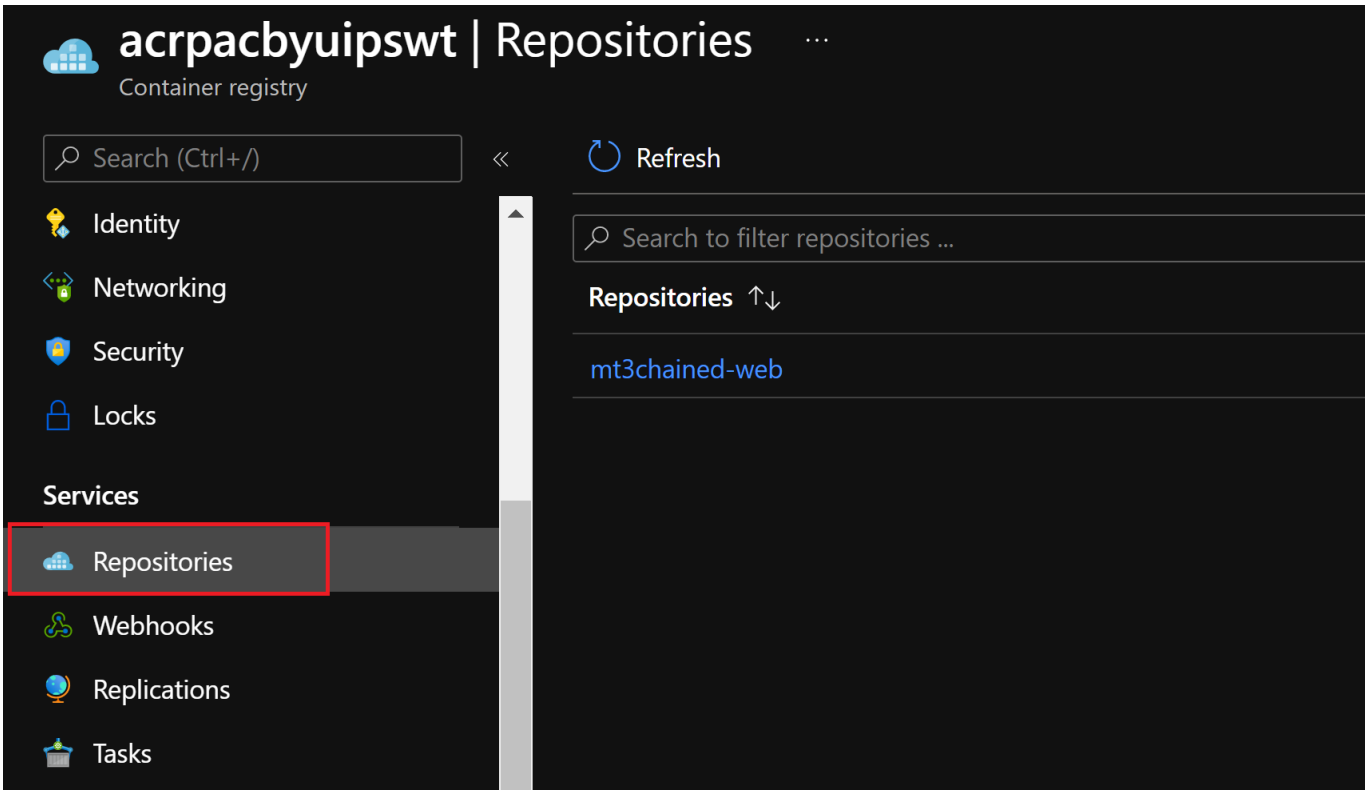


7. Click on **Create the workflow.yaml** and then click on build, you can see all the steps of your job.

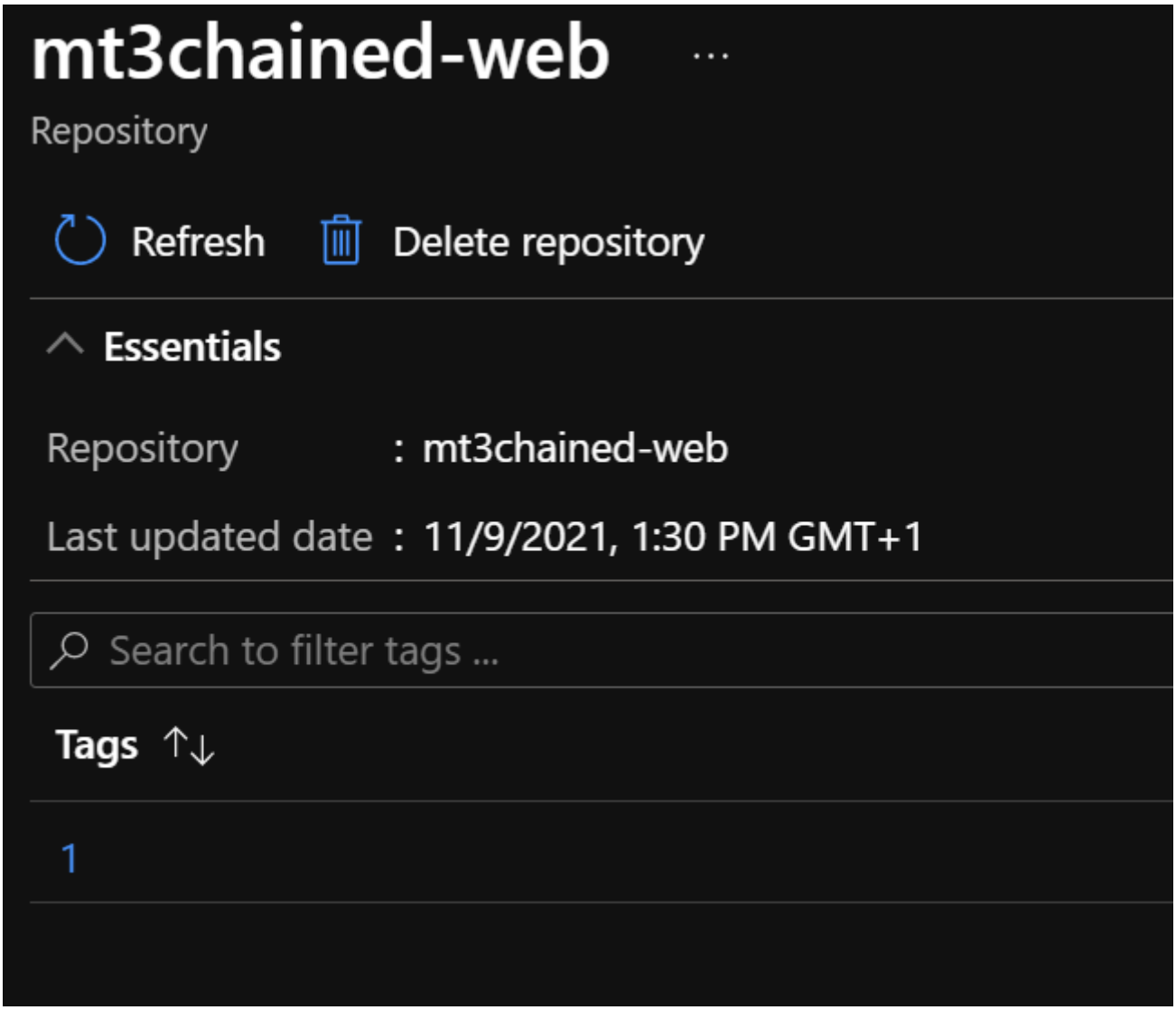


> This GitHub Action build an image with a Dockerfile and push that image to your Azure Container Registry, the name of the image is specified in the Environment variable **APP_NAME: mt3chained-web**, and the image tag is the **github.run_number**.

8. Go to the Azure Portal and check your Azure Container Registry Repositories.



9. Click on the **mt3chained-web** Repository and you can see your tag.



Task 2 - Add in the workflowyaml the latest image tag

1. Open the Windows Terminal and run te following command to add the lastest tag.

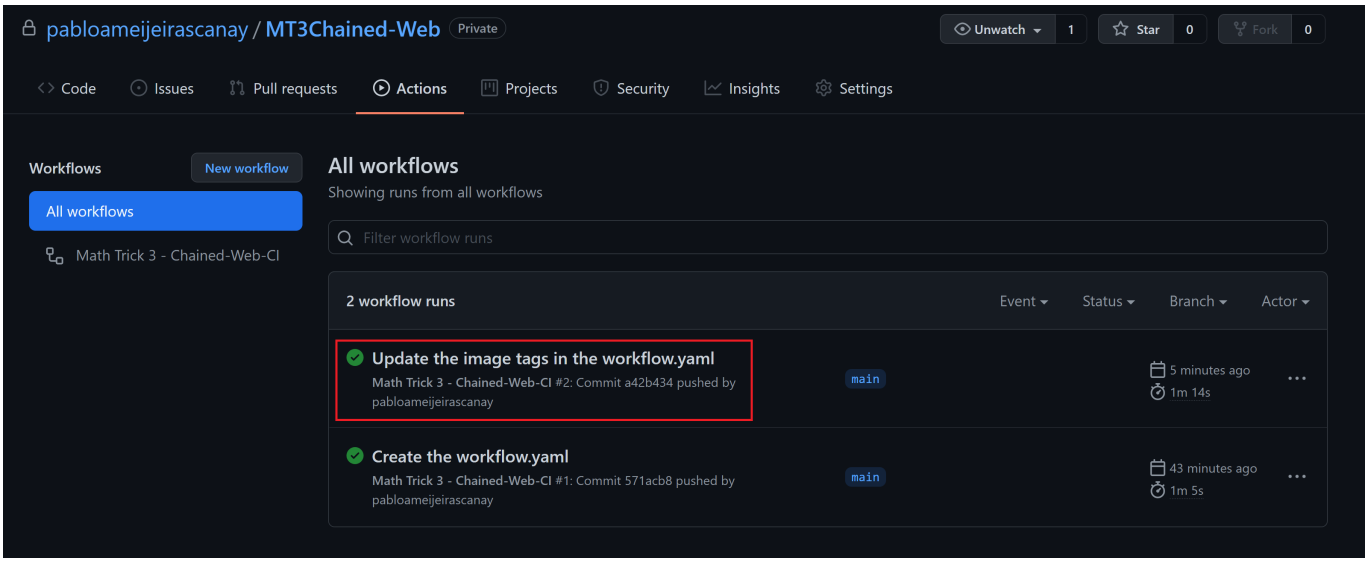
```
(Get-Content (Join-Path $Path2 workflow.yaml)).replace("## <latestTag> ##", '  ${{ env.LOGIN_SERVER }}/${{ env.APP_NAME }}:latest') | Set-Content (Join-Path $Path2 workflow.yaml)
```

> NOTE: This script replaces the string **## <latestTag> ##** in the workflow.yaml with the right code to add the latest image tag.

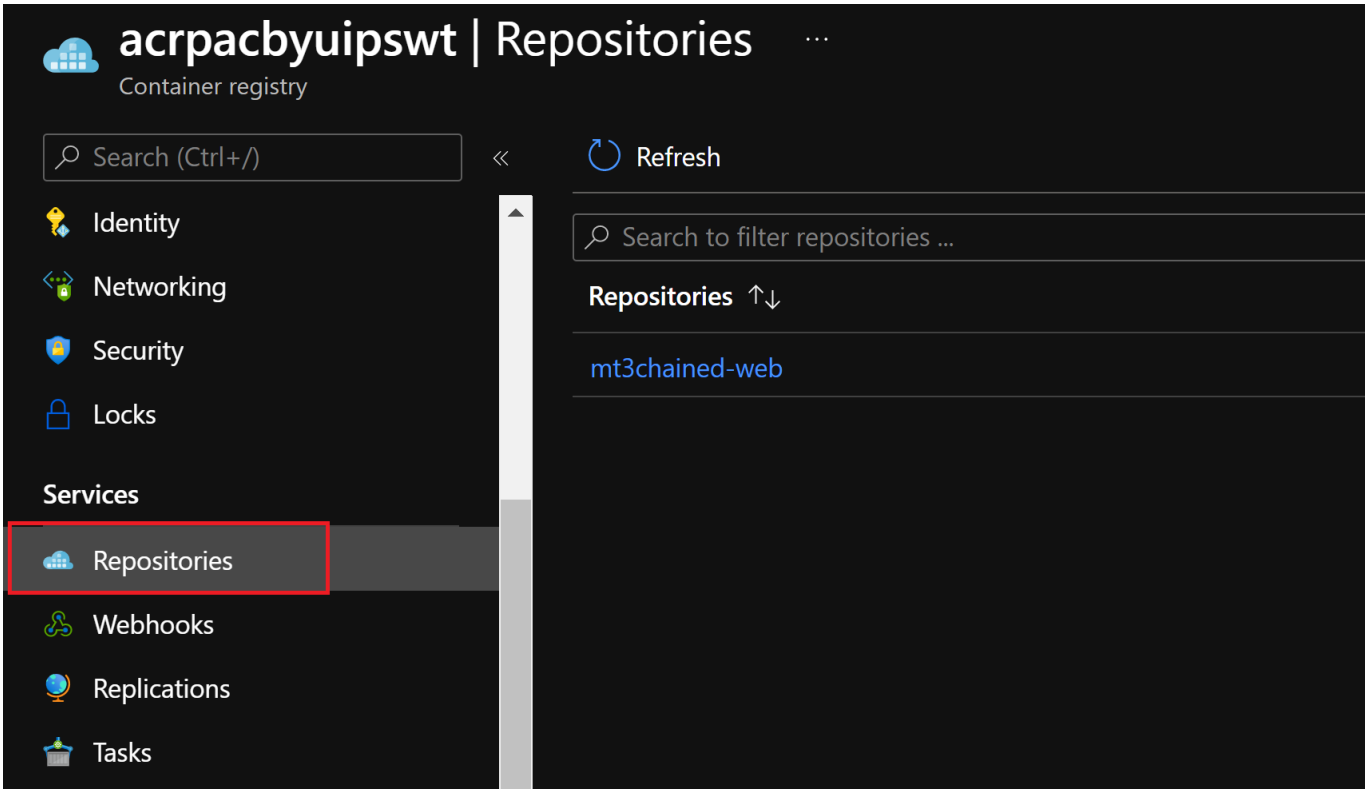
2. Push the changes to your GitHub Repository.

```
git add .
git commit -m "Update the image tags in the workflow.yaml"
git push
```

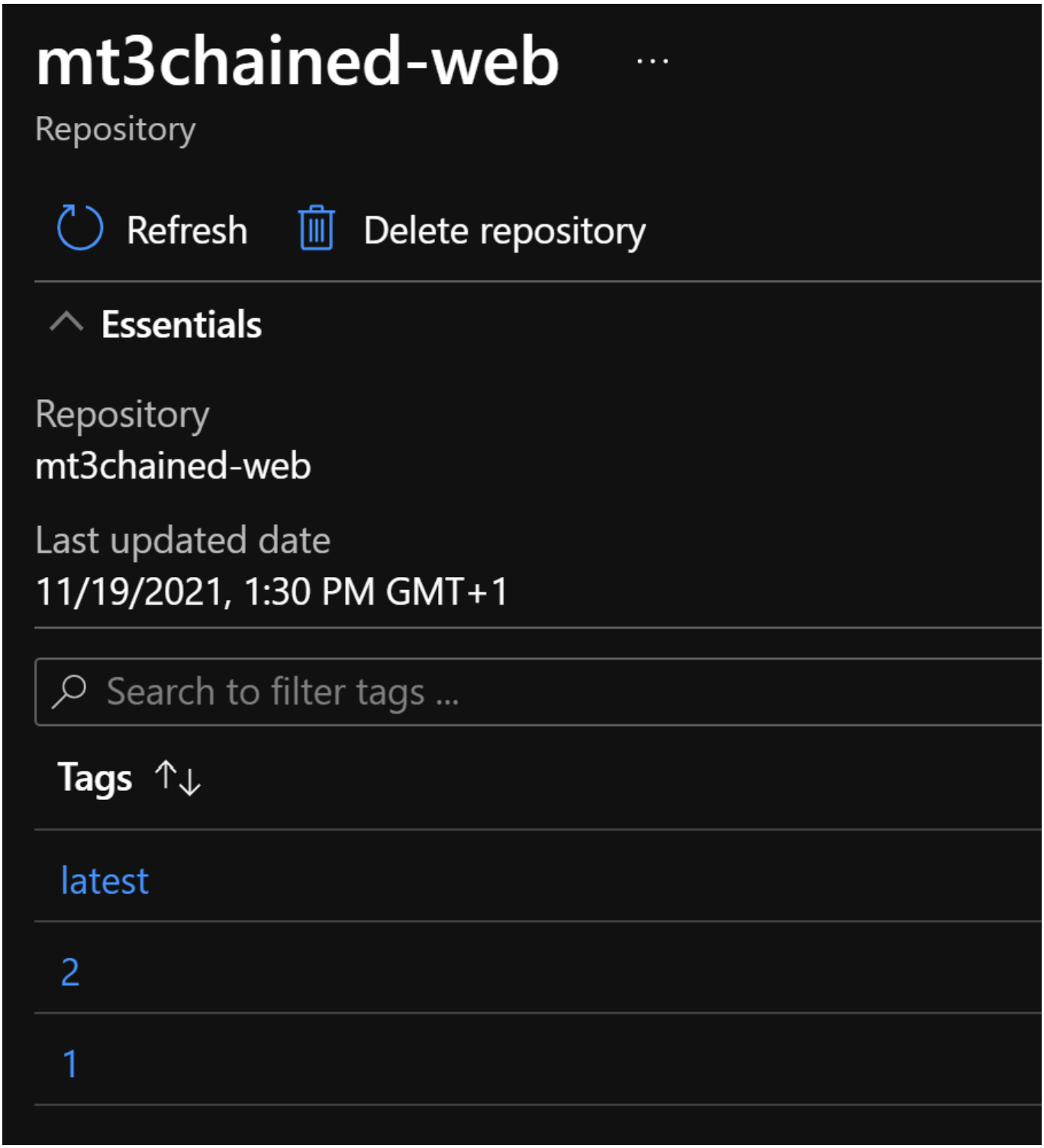
3. Now you can see your new workflow run in the Actions tab of your GitHub Repository.



4. Go to the Azure Portal and check your Azure Container Registry Repositories.



5. Click on the **mt3chained-web** Repository and you can see the two new tags (the tag **2** appears because you run the workflow.yaml again, so it creates a new **github.run_number**).



Exercise: Add a Kubernetes Deploy Step to the Workflow YAML

Task 1 - Deploy the new workflow yaml

1. To create a the new GitHub Action, you need to upgrade the **workflow.yaml** content in the **C:\k8s\Labs\MathTrick\Chained\MT3Chained-Web\.github\workflows** directory. Now there is a file **workflow-web2.yaml** in the **C:\k8s\Labs\Module5\workflows** directory with the following content:

> This workflow does the same as the last one, but with one more step to deploy the app in the AKS cluster.

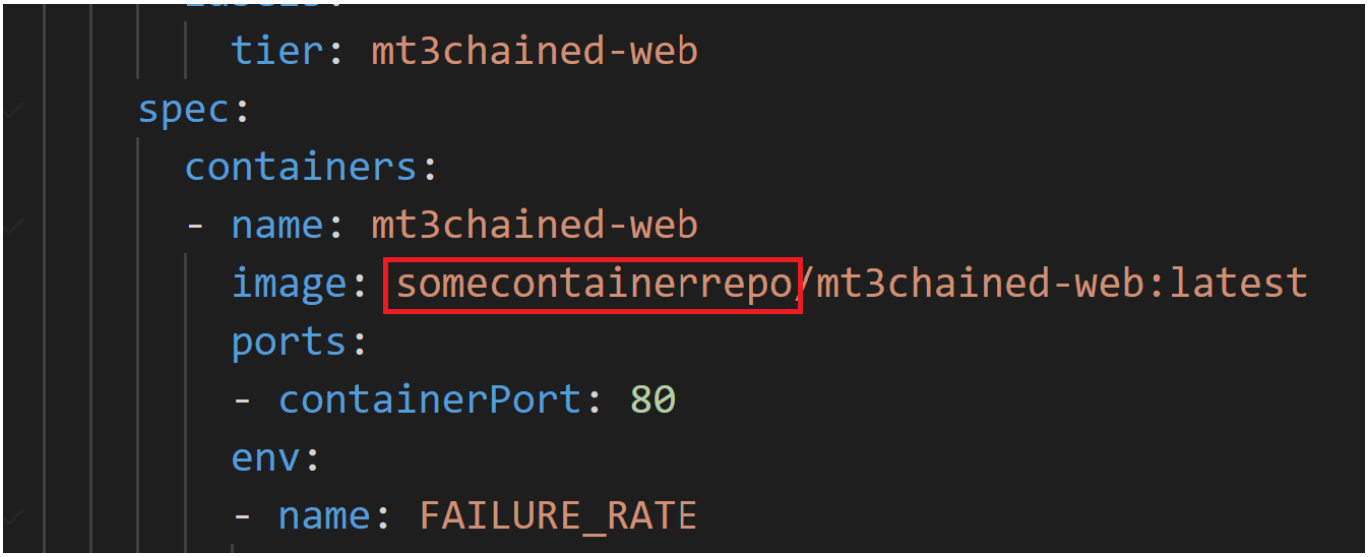
```yaml
name: Math Trick 3 - Chained-Web-CI

on:
  push:
    branches:
      - main

env:
  APP_NAME: <appName>
  LOGIN_SERVER: <acrServerName>
  CLUSTER_NAME: <clusterName>
  CLUSTER_RESOURCE_GROUP: <clusterResourceGroup>
  NAMESPACE: <namespaceName>

jobs:

  build-and-deploy:
    runs-on: ubuntu-latest
    steps:

      - name: Check Out Repo
        uses: actions/checkout@v2

      # Connect to Azure Container Registry (ACR)
      - name: ACR login
        uses: azure/docker-login@v1
        with:
          login-server: ${{ env.LOGIN_SERVER }}
          username: ${{ secrets.ACR_CLIENT_ID }}
          password: ${{ secrets.ACR_CLIENT_PASSWORD }}

      # Container build and push to a Azure Container Registry (ACR)
      - name: Build, tag, and push image to ACR
        uses: docker/build-push-action@v2
        with:
          context: ./
          file: Dockerfile
          push: true
          tags: |
            ${{ env.LOGIN_SERVER }}/${{ env.APP_NAME }}:${{ github.run_number }}

      # Set the target Azure Kubernetes Service (AKS) cluster.
      - uses: azure/aks-set-context@v1
        with:
          creds: '${{ secrets.AZURE_CREDENTIALS }}'
          cluster-name: ${{ env.CLUSTER_NAME }}
          resource-group: ${{ env.CLUSTER_RESOURCE_GROUP }}

      # Create namespace if doesn't exist
      - run: |
          kubectl create namespace ${{ env.NAMESPACE }} --dry-run -o json |
kubectl apply -f -
      # Deploy app to AKS
```

```
      - uses: azure/k8s-deploy@v1
        with:
          manifests: |
            k8s-deployment.yaml
            k8s-service.yaml
          images: |
            ${{ env.LOGIN_SERVER }}/${{ env.APP_NAME }}:${{ github.run_number }}
          namespace: ${{ env.NAMESPACE }}
```

2. Copy the next script in the terminal to replace the **workflow-web2.yaml** file with the right values and copy the content to the right file called **(workflow.yaml)** in the right path **(C:\k8s\Labs\MathTrick\Chained\MT3Chained-Web\.github\workflows)**.

```
$Path1="C:\k8s\Labs\Module5\workflows"
$Path2="C:\k8s\Labs\MathTrick\Chained\MT3Chained-Web\.github\workflows"
$original_file = (Join-Path $Path1 workflow-web2.yaml)
$destination_file = (Join-Path $Path2 workflow.yaml)
(Get-Content $original_file) | Foreach-Object {
    $_ -replace "<appName>", "mt3chained-web" `
       -replace "<acrServerName>", $ACR_SERVER_NAME `
       -replace "<clusterName>", $AKS_NAME `
       -replace "<clusterResourceGroup>", $AKS_RESOURCE_GROUP `
       -replace "<namespaceName>", "default"
    } | Set-Content $destination_file
```

3. Now replace in the manifest **k8s-deployment.yaml** the server name of your ACR.



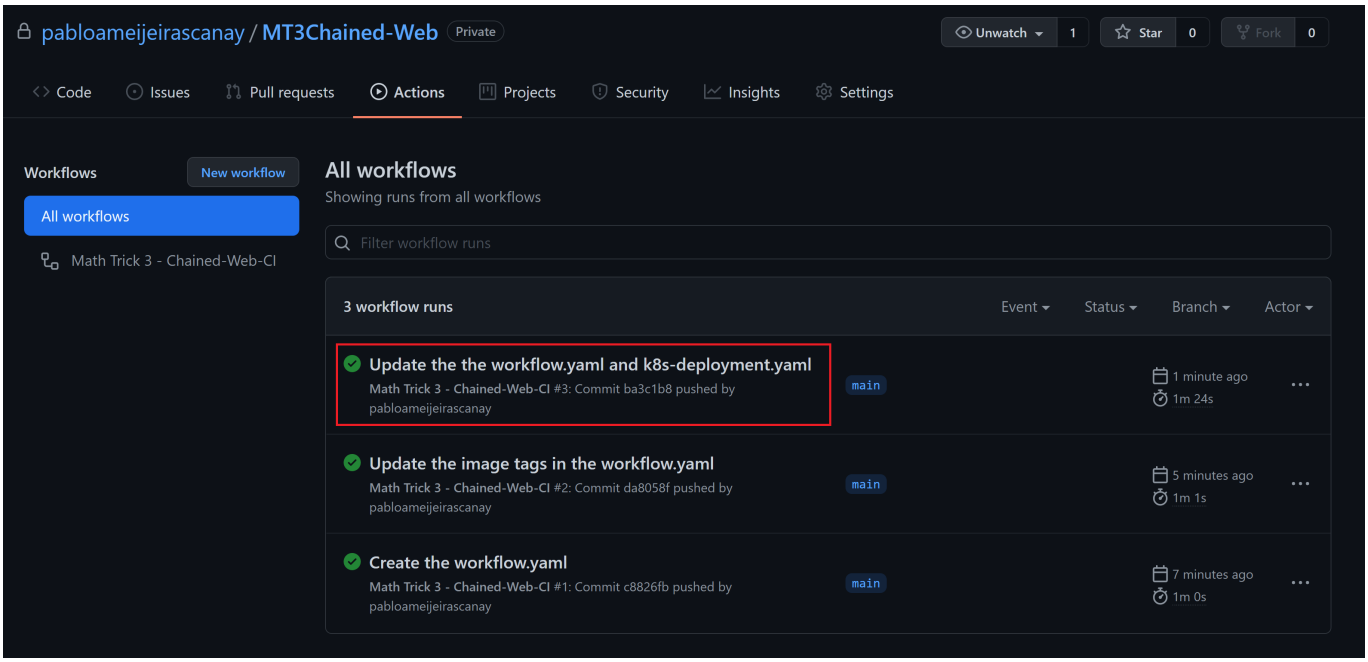
Copy the next script in your Windows Terminal.

```
$Path="C:\k8s\Labs\MathTrick\Chained\MT3Chained-Web"
(Get-Content (Join-Path $Path k8s-deployment.yaml)).replace("somecontainerrepo",
$ACR_SERVER_NAME) | Set-Content (Join-Path $Path k8s-deployment.yaml)
```

4. Push the changes to your GitHub Repository.

```
git add .
git commit -m "Update the the workflow.yaml and k8s-deployment.yaml"
git push
```

5. Now you can see your new workflow run in the Actions tab of your GitHub Repository.



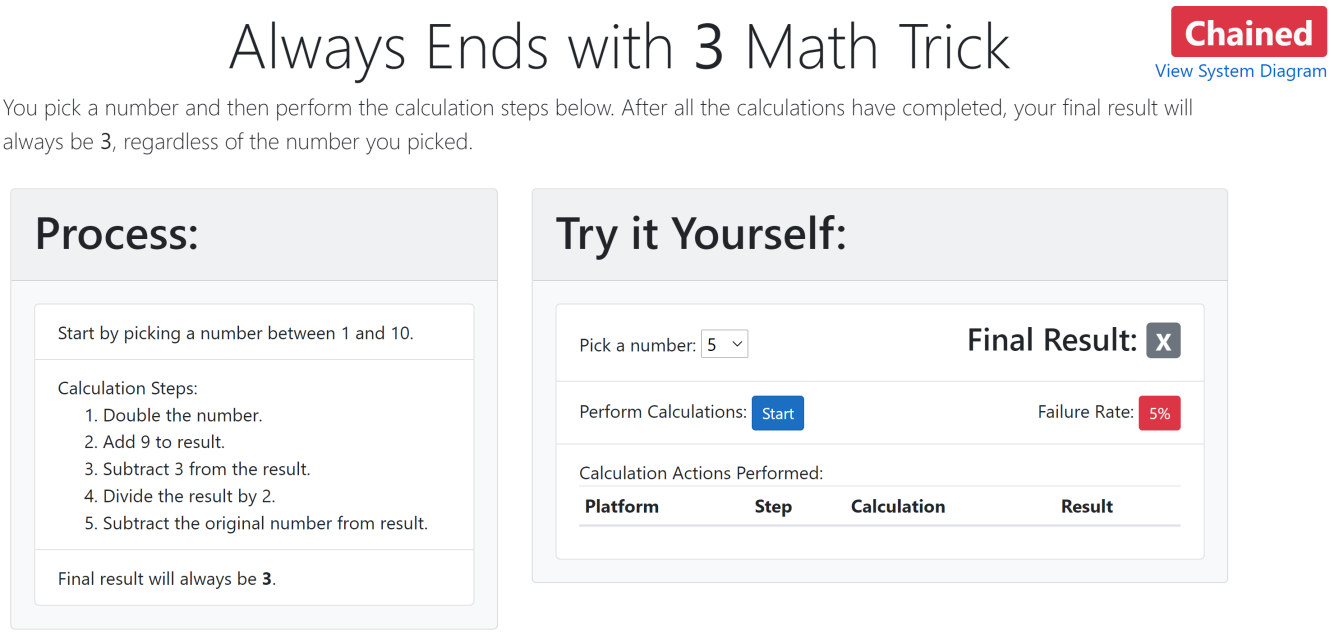
6. Copy the next command in the terminal to check your cluster for the expected resources:

```
kubectl get all
```

7. The output should look like this:

```
PS C:\k8s\Labs\MathTrick\Chained\MT3Chained-Web> kubectl get all
NAME                                 READY   STATUS    RESTARTS   AGE
pod/mt3chained-web-6c7c494988-wcq4d   1/1     Running   0          2m57s

NAME                     TYPE           CLUSTER-IP    EXTERNAL-IP    PORT(S)        AGE
service/kubernetes       ClusterIP      10.0.0.1      <none>         443/TCP        16m
service/mt3chained-web   LoadBalancer   10.0.37.23    51.105.235.0   80:31791/TCP   2m56s

NAME                            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mt3chained-web   1/1     1            1           2m57s

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/mt3chained-web-6c7c494988   1         1         1       2m57s
PS C:\k8s\Labs\MathTrick\Chained\MT3Chained-Web>
```

8. Once the EXTERNAL-IP for the "service/mt3chained-web" has been assigned a value (you may have to wait a few minutes), open a browser and enter that IP address in your address bar:



9. Verify that the correct image was deployed

```
kubectl get pods -o jsonpath="{.items[*].spec.containers[*].image}"
```

You should see the full name of the image:

```
acrpacyowrfzcd.azurecr.io/mt3chained-web:3
```

10. Return to the Windows Terminal in the MT3Chained-Web folder.

11. Create a simple file just to trigger a change in the main branch.

```
echo "Hello" > text.txt
```

12. Commit and push your changes.

```
git add .
git commit -m "Add a test file"
git push
```

13. Wait a few minutes and verify the deployed container in AKS has automatically updated. Verify what image was deployed

```
kubectl get pods -o jsonpath="{.items[*].spec.containers[*].image}"
```

You should see the full name of the image:

```
acrpacyowrfzcd.azurecr.io/mt3chained-web:4
```

CONGRATULATIONS! You just built a full CI/CD pipeline from code to Kubernetes.

Exercise: A Complex Microservices CI/CD Pipelines Using Helm and Azure Key Vault

Background

Before creating the Git repos for the calculation step microservices, it's important to understand how the code projects of these microservices are structured. Most of the .Net microservices access a common library called **MathTrickCore**. This is a separate solution, but is referenced by .Net "step" projects (except the NodeJS and Python microservices).

The file structure contains the following hierarchy:

```
..\MathTrick
    \Chained
      \MT3Chained-Step1
        \MT3Chained-Step1.csproj
      \MT3Chained-Step2
        \MT3Chained-Step2.csproj
      ...
    \Gateway
      \MT3Gateway-Step1
        \MT3Gateway-Step1.csproj
      \MT3Gateway-Step2
        \MT3Gateway-Step2.csproj
      ...
    \MathTrickCore
      \MathTrickCore.csproj
```

Notice how the **MathTrickCore** is located 2 levels higher than any of the microservices projects. This is an important consideration when working with the **Dockerfile**. Docker files cannot access folder higher up in a hierarchy than the current folder (ie. Dockerfiles do not support references like ..\..\file). That means for build process to work, the Dockerfile has to be run at the parent level (**\MathTrick**) so it can use that parent folder as its *context*:

```
## Executed in the parent folder: .\Labs\MathTrick>
docker build -t "myacr.azurecr.io/mt3chained-step1:latest" --file
"./Chained/MT3Chained-Step1/Dockerfile" .
```

To facilitate this type of build, the **Dockerfile** is setup to copy all of its files from the parent folder:

```
...

FROM mcr.microsoft.com/dotnet/sdk:5.0 AS build
WORKDIR /src
COPY ["./Chained/MT3Chained-Step1/MT3Chained-Step1.csproj", "./Chained/MT3Chained-
Step1/"]
COPY ["./MathTrickCore/MathTrickCore.csproj", "./MathTrickCore/"]
RUN dotnet restore "./Chained/MT3Chained-Step1/MT3Chained-Step1.csproj"
COPY . .

FROM build AS publish
WORKDIR "/src/."
RUN dotnet publish "./Chained/MT3Chained-Step1/MT3Chained-Step1.csproj" -c Release
-o /app/publish
...
```

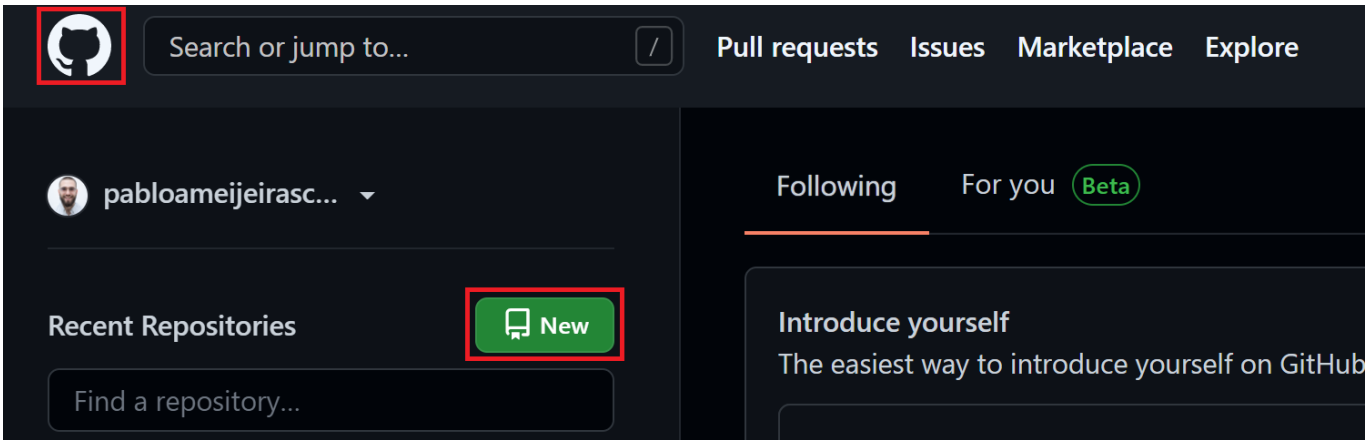
When you add all of these microservices to your GitHub Account, the services will be in different repos, but most will reference the shared project. You'll have to keep this file structure in mind when you check out multiple repos when building the Docker images.

> This structure does not apply to the Web UI project and the non-.Net microservices. Those projects are self-contained.

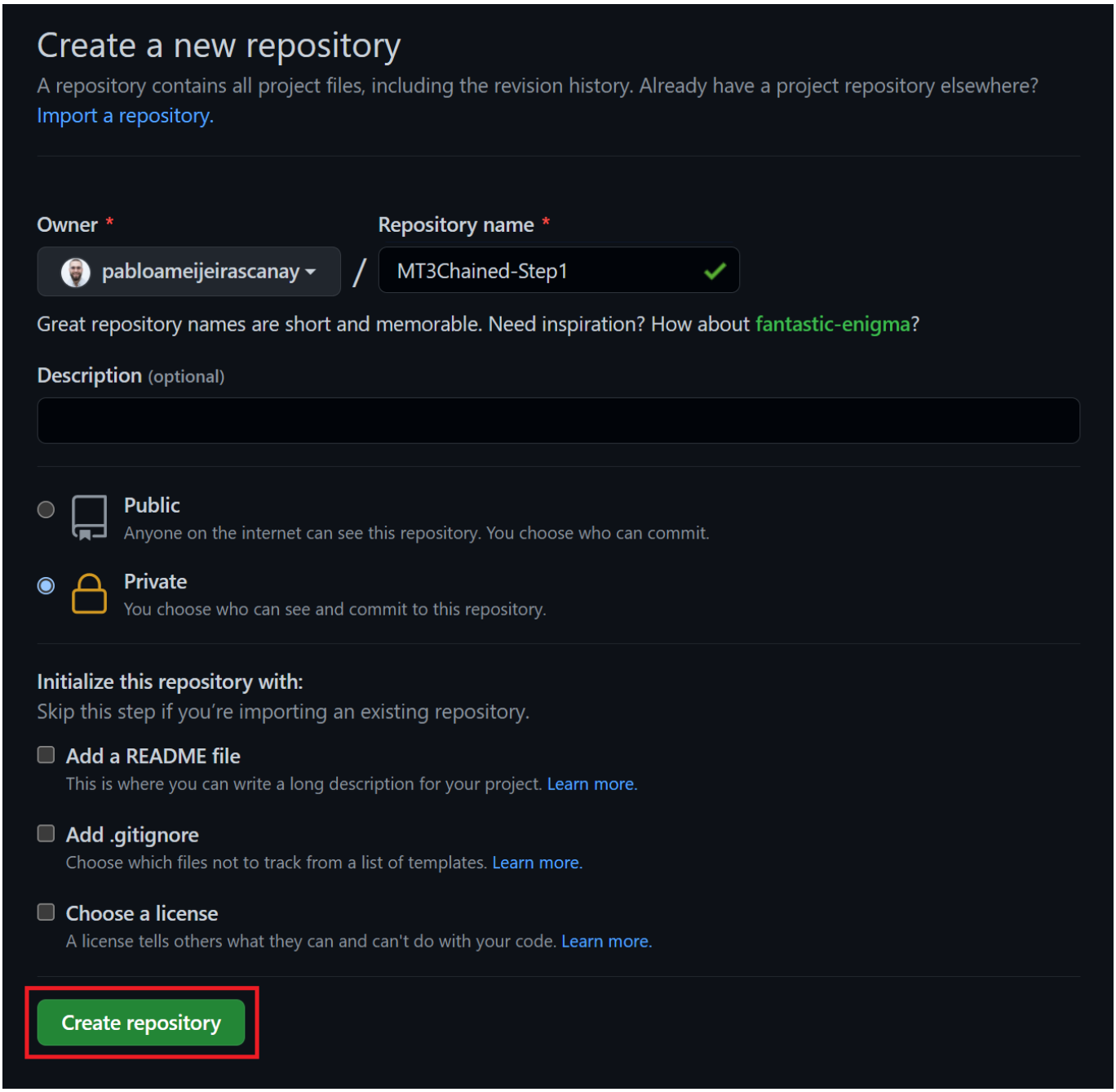
Task 1 - Create repos for the remaining microservices

In the previous exercise, you created a repository for the Web microservice. In this exercise, you'll create repositories for each of the remaining services.

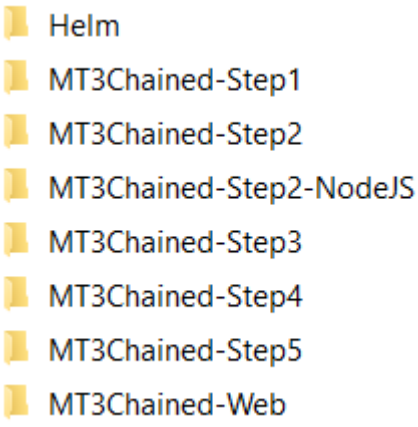
1. Open GitHub and click on the GitHub avatar to go home, then click on **New**.



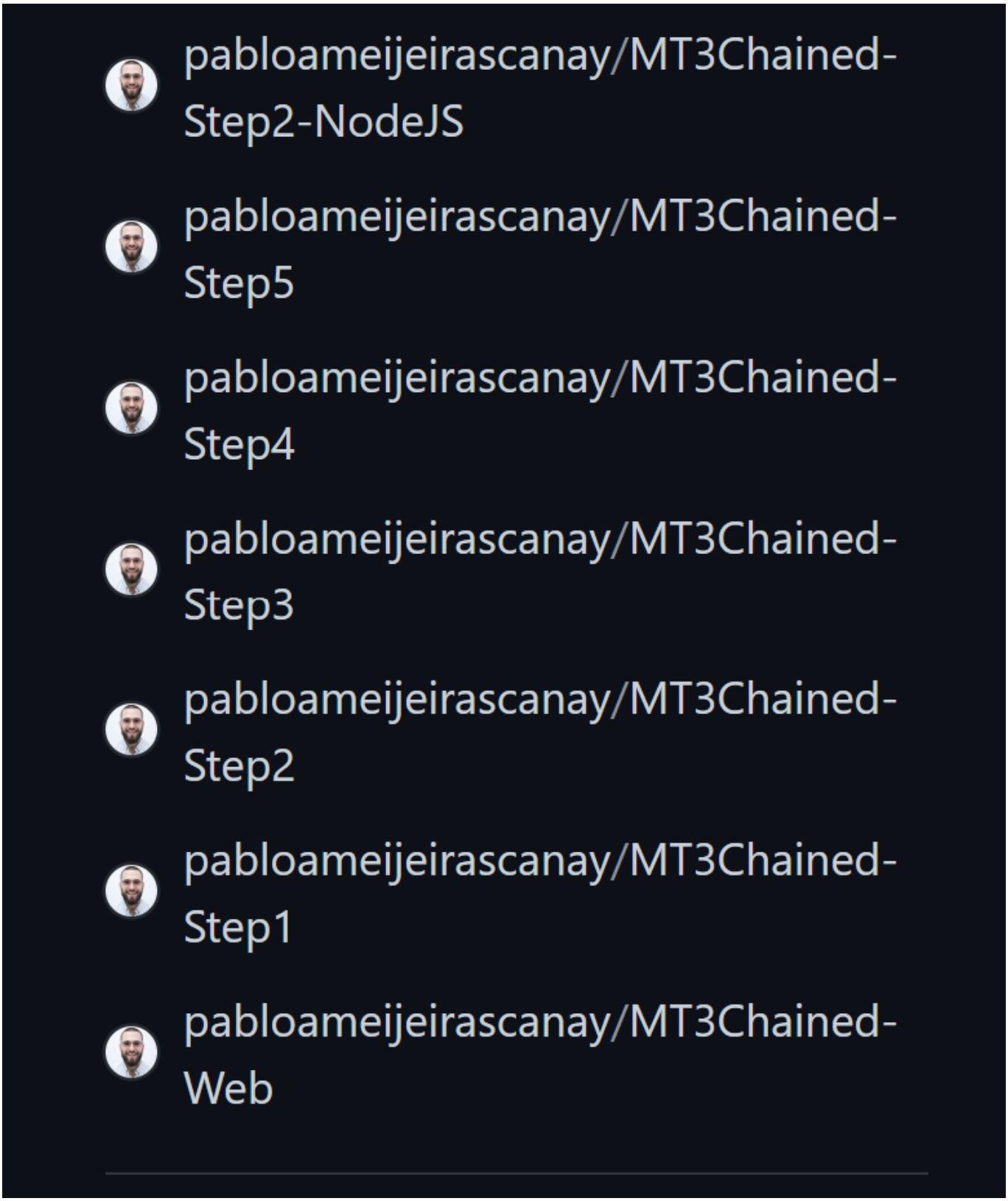
2. Create a new repository called "MT3Chained-Step1". **CHECK "Private"** and click the **Create Repository** button.



3. Repeat the process for the other services, including the Node JS service, but excluding the *Helm* folder (for now).



4. When you're done, your list should look like this:



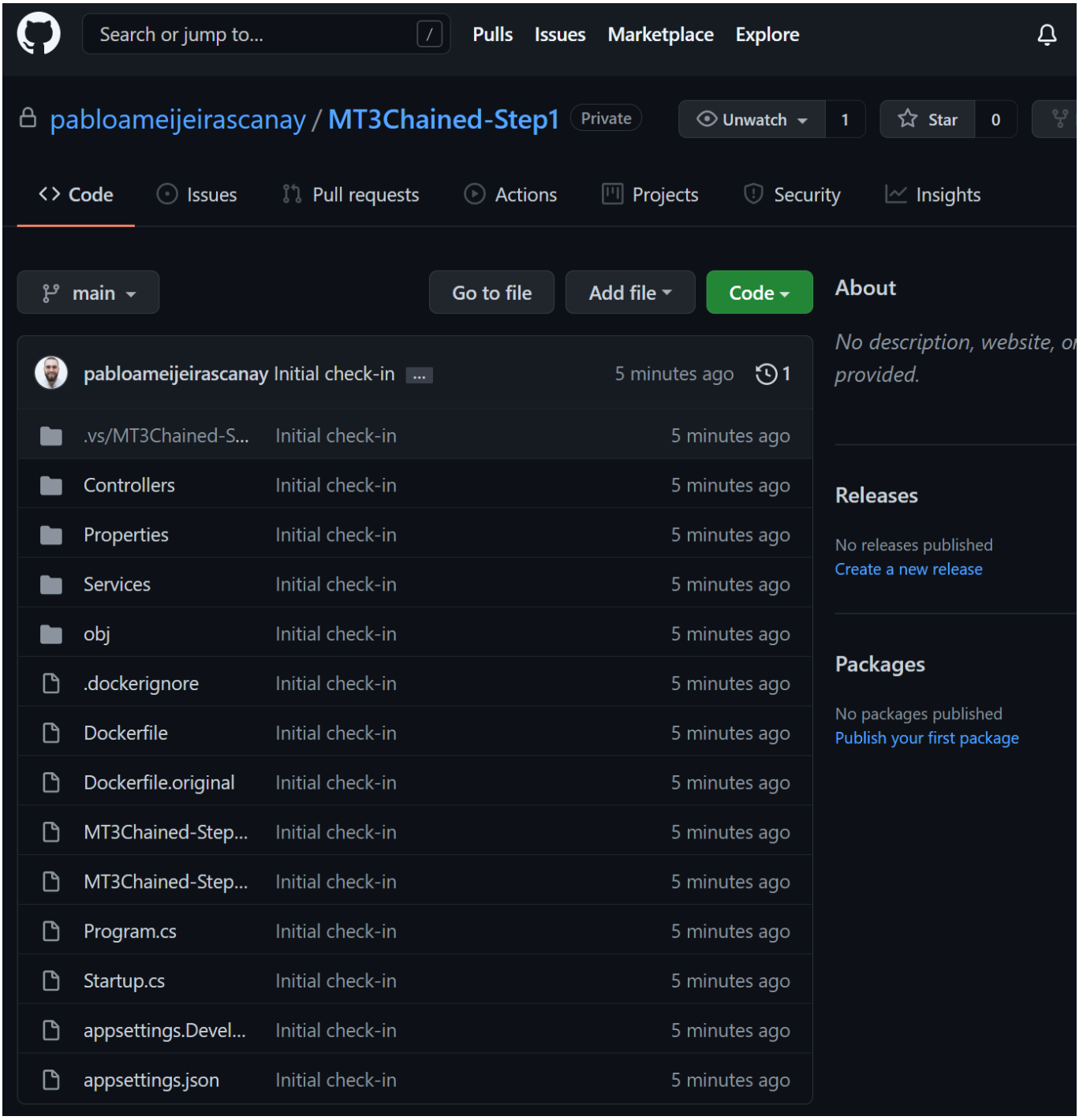
Task 2 - Initialize git, check in the source code and push it to GitHub (For all the repos, excluding the NodeJS step and web repo)

1. Initialize git, check in the source code and push it to GitHub for all step repos. **IMPORTANT: REPLACE THE *$GITHUB_ALIAS* VARIABLE WITH YOUR GITHUB ALIAS**.

```
$GITHUB_ALIAS="ENTER YOUR GITHUB ALIAS"
```

```
ForEach ($i in 1..5) {
  cd "C:\k8s\Labs\MathTrick\Chained\MT3Chained-Step$i"
  Write-Host "=========================MT3Chained-
Step$i=========================" -ForeGround Green
  git init
  git add .
  git commit -m "Initial check-in"
  git remote add origin "git@github.com:$GITHUB_ALIAS/MT3Chained-Step$i.git"
  git branch -M main
  git push -u origin main
}
```

2. Return back to the GitHub. Select for example the **MT3Chained-Step1** repo, you should see all your files (the same for all the step repos).



3. Create two new repos called "**Math Trick 3 - Chained**" and "**MathTrickCore**" following the same steps for the others that you created. Use the "**Math Trick 3 - Chained**" repo to store the contents of the "**Helm**" folder and use the "**MathTrickCore**" repo to store the contents of the "**MathTrickCore**" folder.

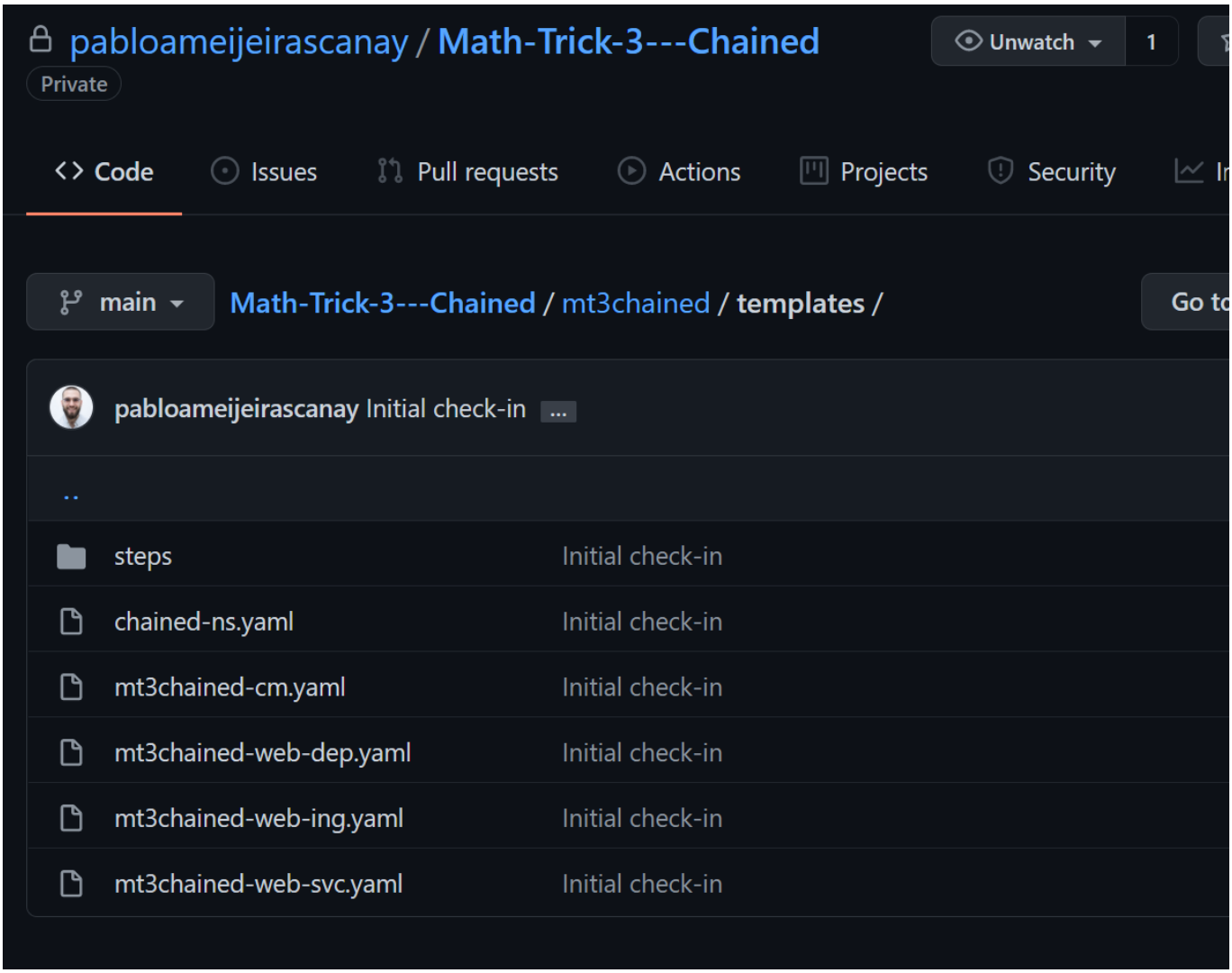
4. Initialize git, check in the source code and push it to GitHub for the **Math-Trick-3---Chained** repo.

```
cd "C:\k8s\Labs\MathTrick\Chained\Helm"
git init
git add .
git commit -m "Initial check-in"
git remote add origin git@github.com:$GITHUB_ALIAS/Math-Trick-3---Chained.git
git branch -M main
git push -u origin main
```

5. Initialize git, check in the source code and push it to GitHub for the **MathTrickCore** repo.

```
cd "C:\k8s\Labs\MathTrick\MathTrickCore"
git init
git add .
git commit -m "Initial check-in"
git remote add origin git@github.com:$GITHUB_ALIAS/MathTrickCore.git
git branch -M main
git push -u origin main
```

6. Now, for example you should see all the Helm files in the Math-Trick-3---Chained repo (the same for the MathTrickCore repo)



Task 3 - Add the same secrets you added in the Exercise: Create the Secrets for the GitHub repo

1. For the *step* and *Step2-NodeJS* repos create the **ACR_CLIENT_ID** and **ACR_CLIENT_PASSWORD** secrets for the ACR authentication and **AZURE_CREDENTIALS** secret for the Azure Key Vault Authentication (Then we will assign the access policies to the Service Principal created). You can use the following links if you don't remember how to create the secrets:

Task 2 - Create the GitHub Action Secret to manage the AKS cluster and the Azure Key Vault

Task 3 - Create the GitHub Actions Secrets for the Azure Container Registry Authentication

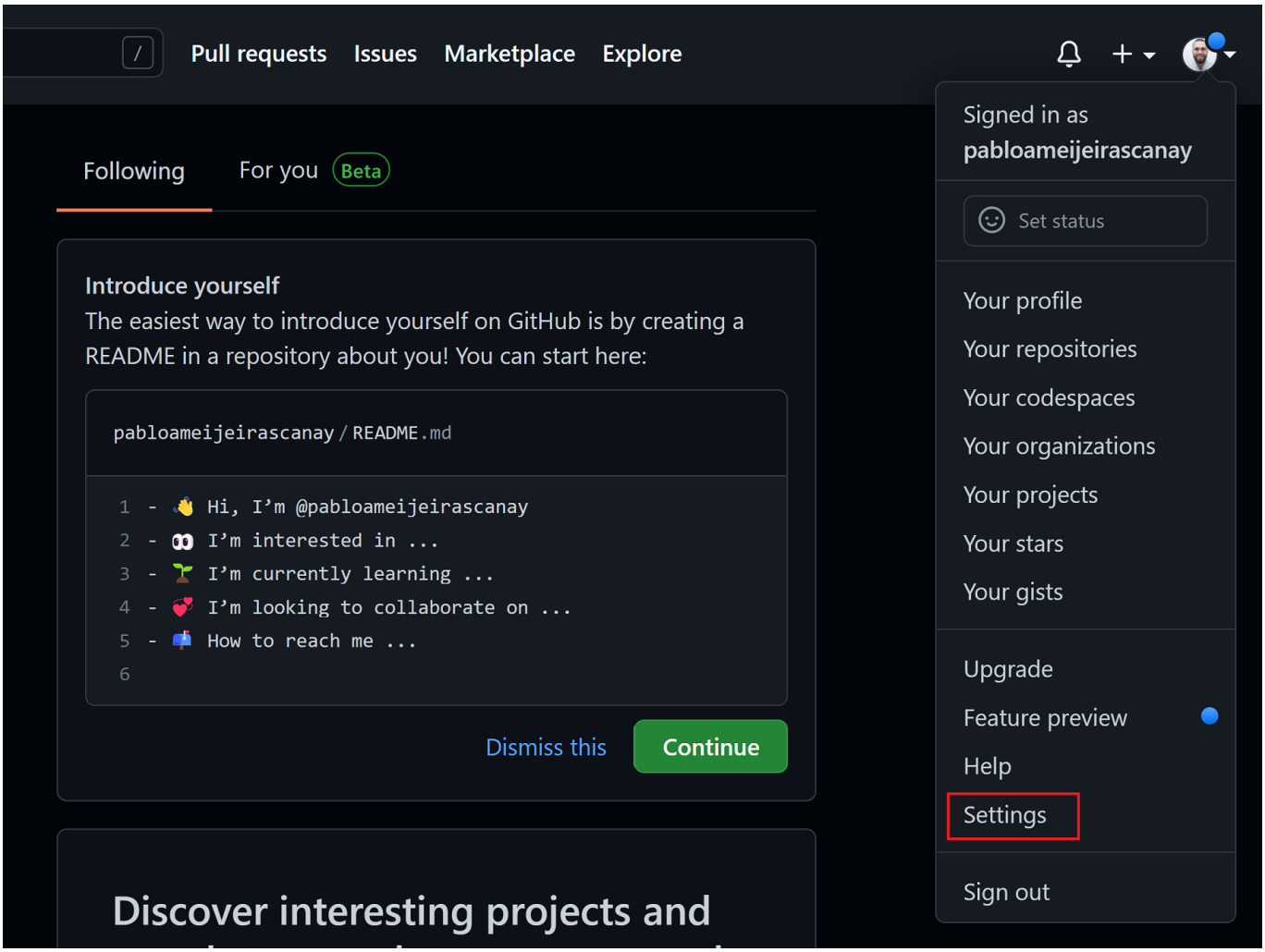
2. For the *Math-Trick-3---Chained* repo create the **AZURE_CREDENTIALS** secret for the Azure Key Vault and AKS Authentication. You can use the following link if you don't remember how to create the secret:

Task 2 - Create the GitHub Action Secret to manage the AKS cluster and the Azure Key Vault

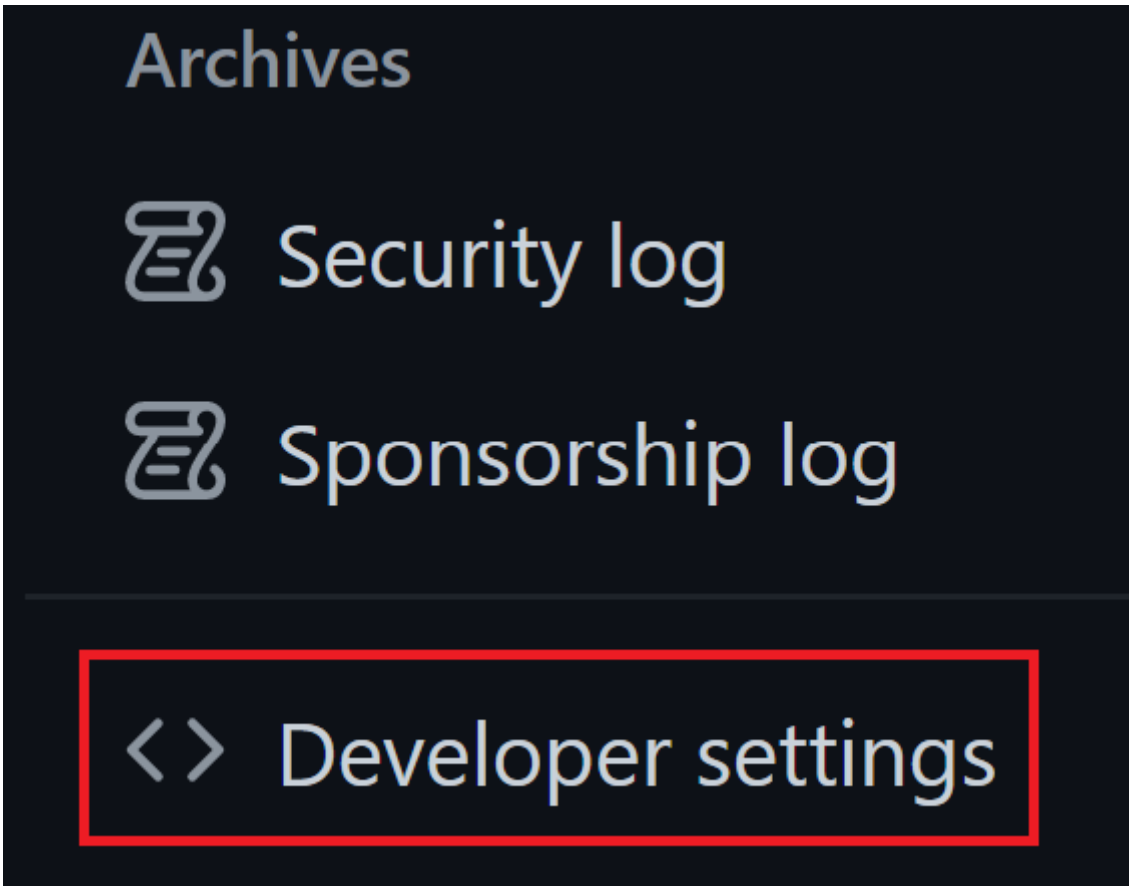
Task 4 - Create a GitHub Token

Create a github token to use in the workflow.yaml of each step repos to access the *MathTrickCore* repo; and also for step, NodeJS and web repos to trigger the workflow of the *Math-Trick-3---Chained* when there is a change (it will be used in the last steps of the lab)

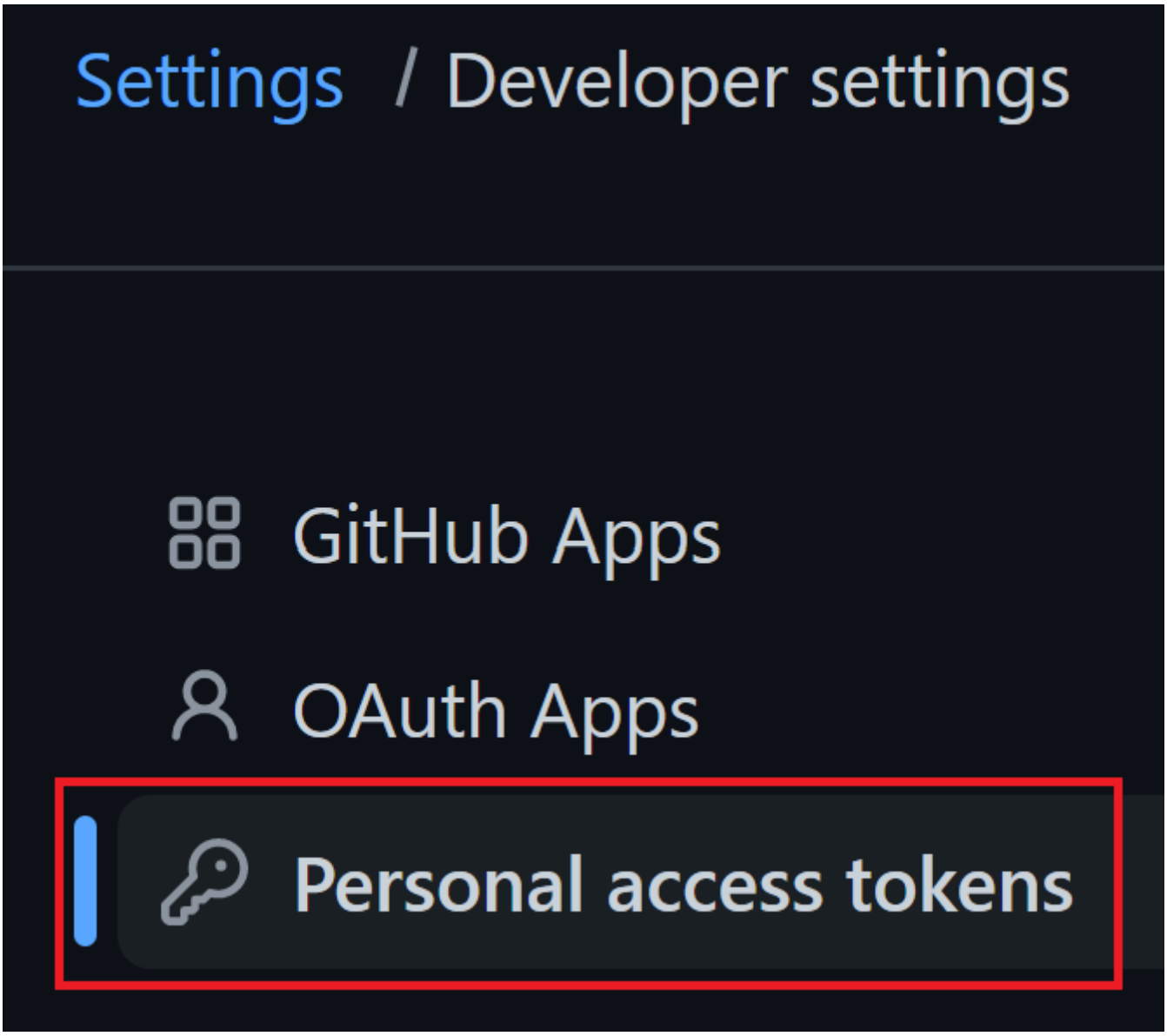
1. In the upper-right corner of any page, click on your profile photo, then click Settings.



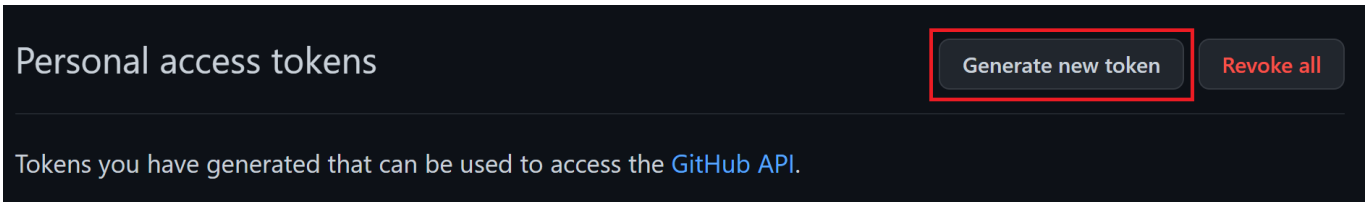
2. In the left sidebar, click Developer settings.



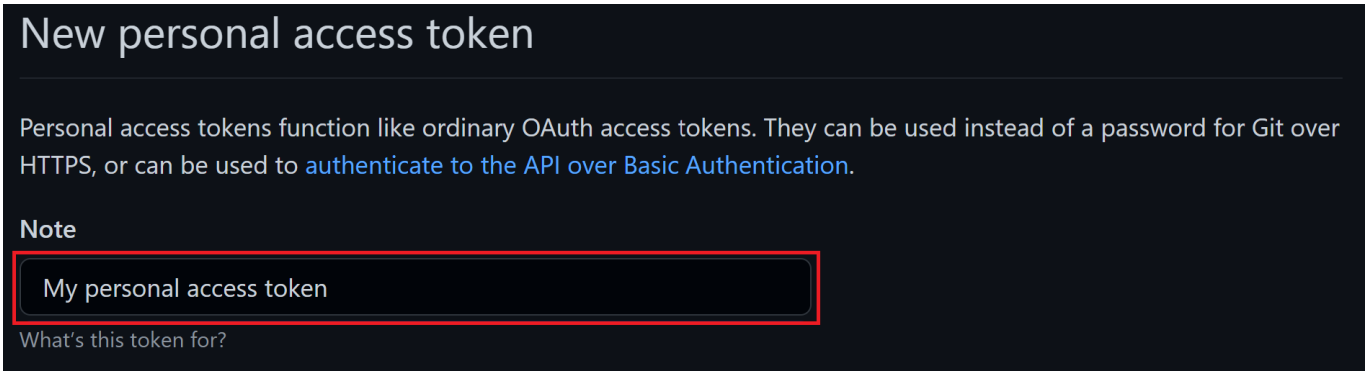
3. In the left sidebar, click Personal access tokens.



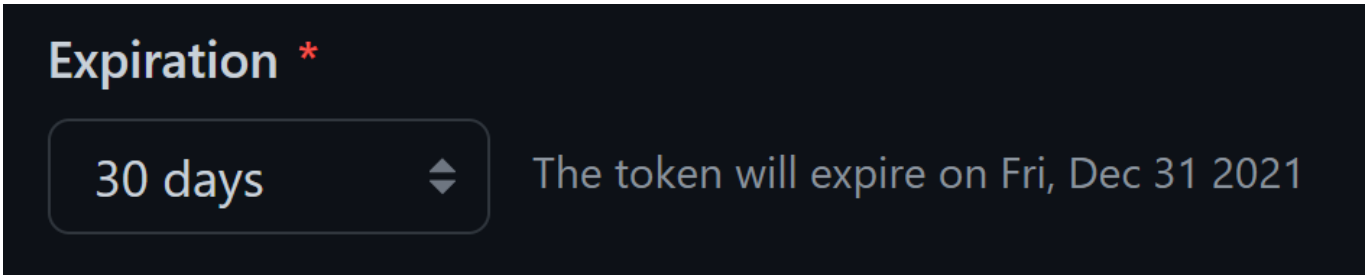
4. Click Generate new token.



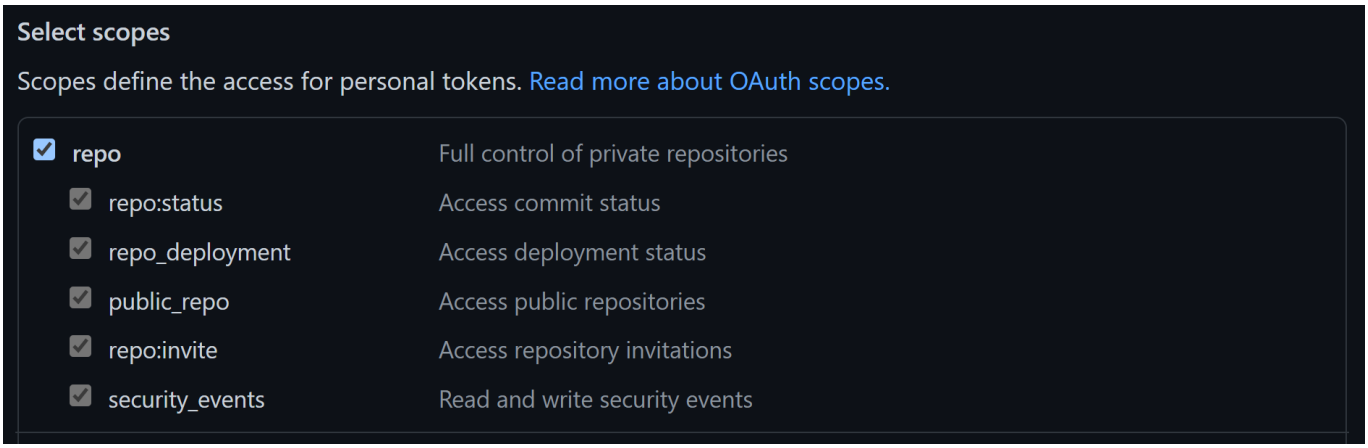
5. Give your token a descriptive name.



6. To give your token an expiration, select the Expiration drop-down menu, then click a default or use the calendar picker.



7. Select the scopes, or permissions, you'd like to grant this token. To use your token to access other repositories, select repo.

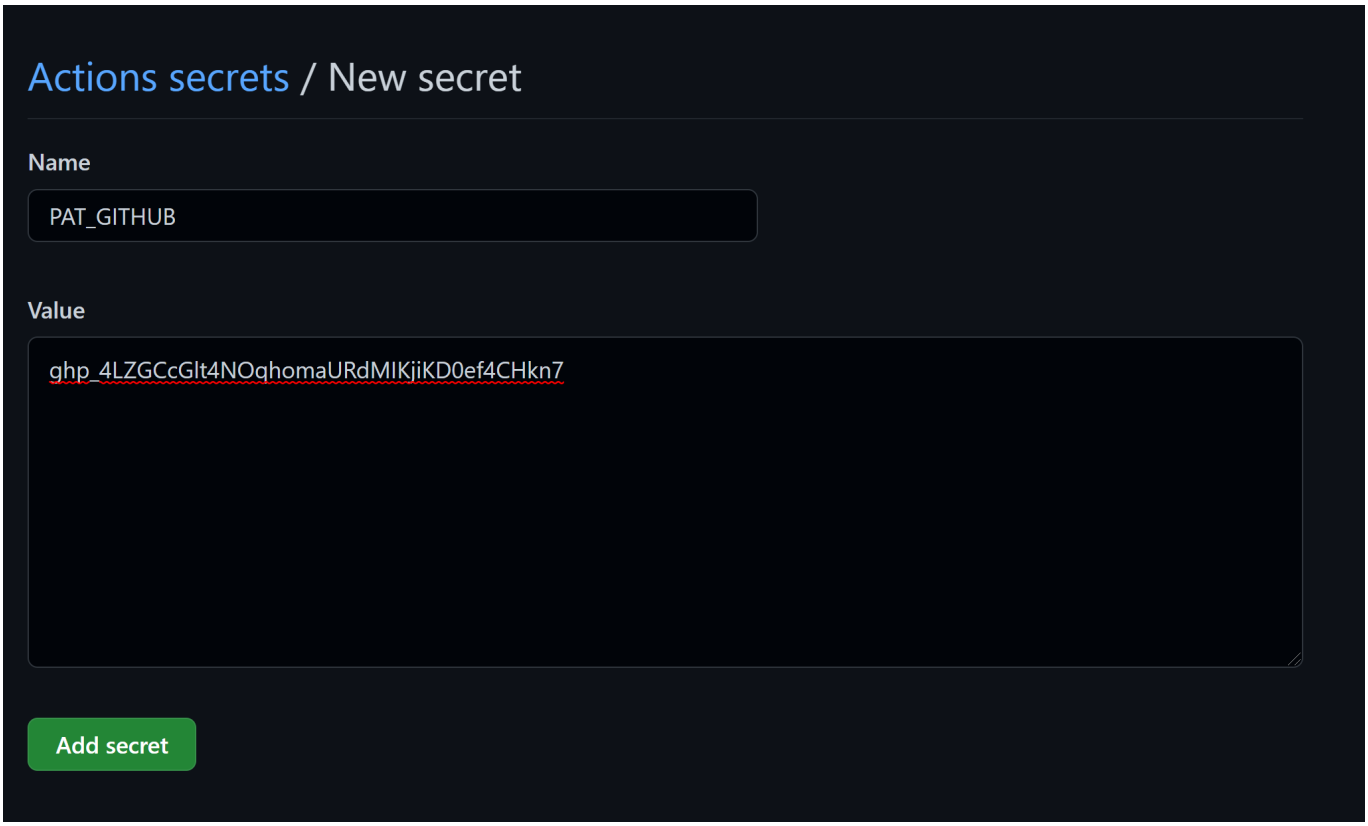


8. Click Generate token.

9. Copy the token in a .txt file, it will be used to create a secret.

10. Create a Action Secret for all the ***step*** repos (including the **NodeJS step** and the **web** repo). Not all of them require the secret at this point, but they will in later Tasks. **Do not create the secret for *MathTrickCore* and the *Math-Trick-3---Chained* repos, as those projects don't need it.**

11. For example how to create the **PAT_GITHUB** secret for the *MT3Chained-Step1* repo, just copy the token which was created before and paste it in the Value section.



IMPORTANT: Repeat this secret creation for all the ***step*** repos (including the **NodeJS step** and the **web** repo).

Task 5 - Create a new GitHub Action for the step repos

Create the **workflow.yaml** in the the directory of all the step repos **\.github\workflows** from the **workflow-steps1.yaml** file, in the directory **C:\k8s\Labs\Module5\workflows**, with the following content:

> This workflow does the same as the first one for the web repo, but with one more step to access the *MathTrickCore* repo.

```
name: <githubRepoName>

on:
  push:
    branches:
      - main

env:
  APP_NAME: <appName>
  LOGIN_SERVER: <acrServerName>

jobs:

  build:
    runs-on: ubuntu-latest
    steps:

      - name: Check-out Repo
        uses: actions/checkout@v2
```

```
        with:
          path: Chained/<githubRepoName>

      - name: Check-out MathTrickCore
        uses: actions/checkout@v2
        with:
          repository: <githubAlias>/MathTrickCore
          path: MathTrickCore
          token: ${{ secrets.PAT_GITHUB }}

      # Connect to Azure Container Registry (ACR)
      - name: ACR login
        uses: azure/docker-login@v1
        with:
          login-server: ${{ env.LOGIN_SERVER }}
          username: ${{ secrets.ACR_CLIENT_ID }}
          password: ${{ secrets.ACR_CLIENT_PASSWORD }}
      # Container build and push to a Azure Container Registry (ACR)
      - name: Build, tag, and push image to ACR
        uses: docker/build-push-action@v2
        with:
          context: ./
          file: Chained/<githubRepoName>/Dockerfile
          push: true
          tags: |
            ${{ env.LOGIN_SERVER }}/${{ env.APP_NAME }}:${{ github.run_number }}
            ${{ env.LOGIN_SERVER }}/${{ env.APP_NAME }}:latest
```

1. Create the **.github\workflows** folder in the step repos.

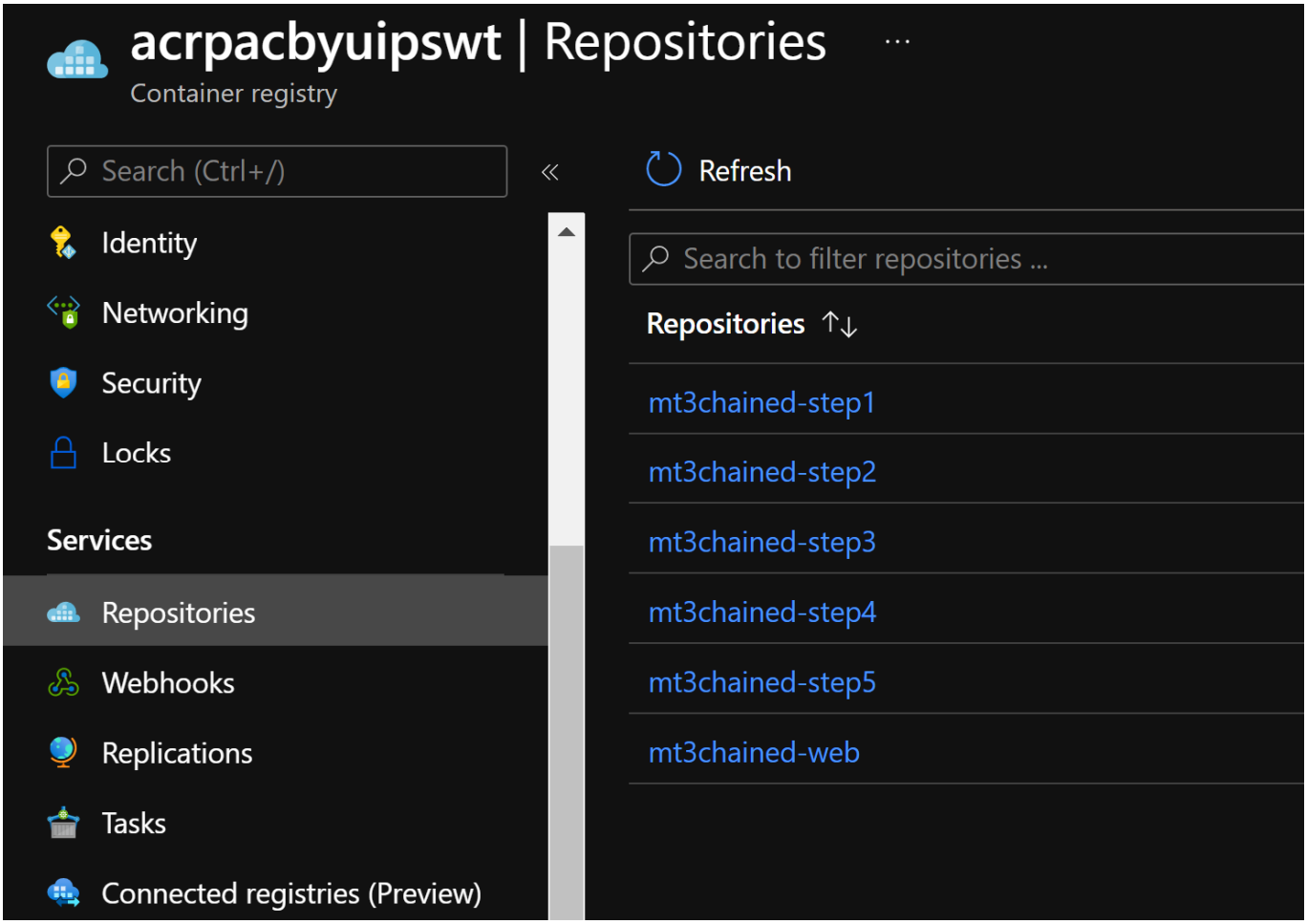
```
ForEach ($i in 1..5) {
  New-Item -Path "C:\k8s\Labs\MathTrick\Chained\MT3Chained-
Step$i\.github\workflows" -ItemType Directory
}
```

2. Copy the following Azure PowerShell code in the Windows Terminal to replace the **workflow-steps1.yaml** file in the path **C:\k8s\Labs\Module5\workflows** with the correct values and give it the correct file name **(workflow.yaml)** copying it in **\.github\workflows** folder of each step repo (excluding the NodeJS step and web repo).

```
ForEach ($i in 1..5) {
  $PathTemplate="C:\k8s\Labs\Module5\workflows"
  $PathWorkflow="C:\k8s\Labs\MathTrick\Chained\MT3Chained-
Step$i\.github\workflows"
  $original_file = (Join-Path $PathTemplate workflow-steps1.yaml)
  $destination_file = (Join-Path $PathWorkflow workflow.yaml)

  (Get-Content $original_file) | Foreach-Object {
    $_ -replace "<appName>", "mt3chained-step$i" `
```

```
        -replace "<githubRepoName>", "MT3Chained-Step$i" `
        -replace "<acrServerName>", $ACR_SERVER_NAME `
        -replace "<githubAlias>", $GITHUB_ALIAS
    } | Set-Content $destination_file

  # Commit and push workflow
  cd "C:\k8s\Labs\MathTrick\Chained\MT3Chained-Step$i\"
  Write-Host "=========================MT3Chained-
Step$i=========================" -ForeGround Green
  git add .
  git commit -m "Add workflow.yaml"
  git push
}
```

3. After all the builds have completed, verify all the images have been created in the ACR.



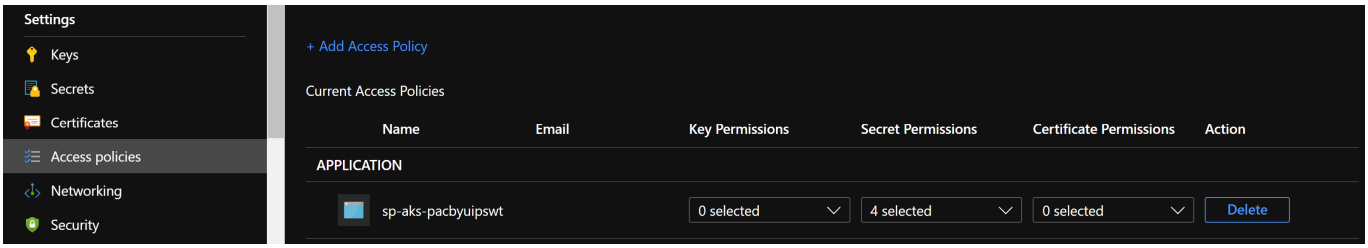
Task 6 - Configure the service principal to access the Key Vault created in Lab 4, Exercise 1, Task 2 to be used by your workflows

You created an Azure Key Vault in the previous module and set it's name in the **$KV_NAME** variable.

1. Create an Access Policy to allow the service principal to read and write secrets to the Key Vault, so the GitHub Action workflows can access to the Key Vault through the **AZURE_CREDENTIALS** secret.

```
$OBJ_ID=$(az ad sp show --id $servicePrincipal.ApplicationId --query objectId -o
tsv)
az keyvault set-policy --name $KV_NAME `
                       --resource-group $AKS_RESOURCE_GROUP `
                       --object-id $OBJ_ID `
                       --secret-permissions delete get list set
```

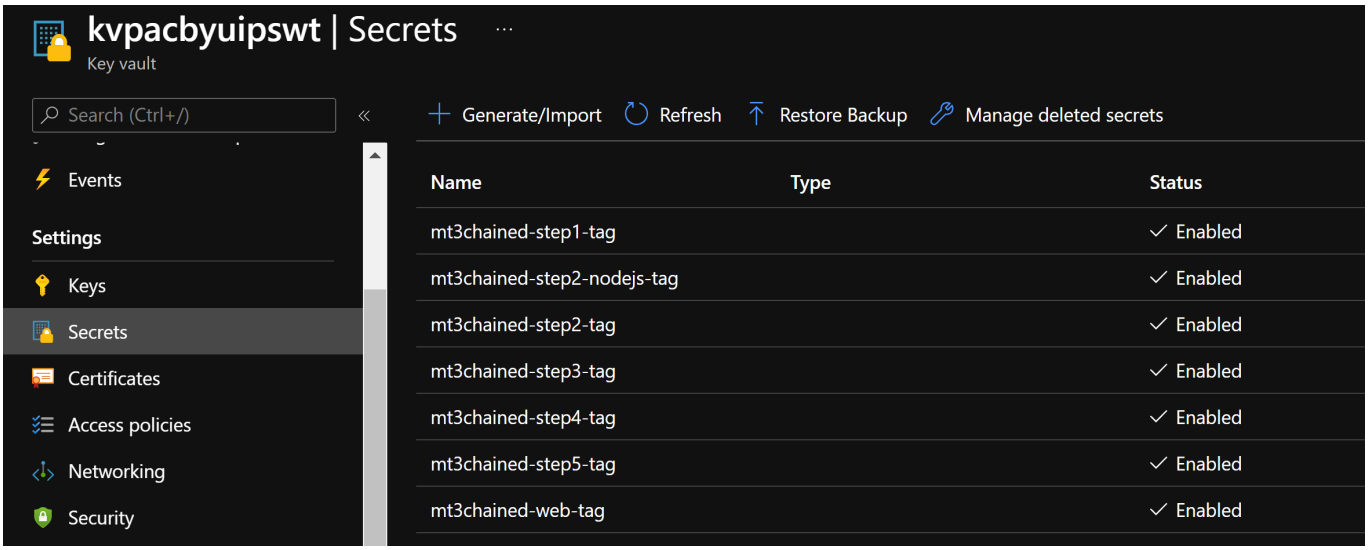
2. Open the Azure Portal, go to the Key Vault and click on **Access policies** to verify the access policy was created



3. Create the default image tags secrets for all the images. Set the default tags to *latest* so you can use them until they're replaced with the build numbers in the pipelines

```
az keyvault secret set --vault-name $KV_NAME --name "mt3chained-step1-tag" --value
"latest"
az keyvault secret set --vault-name $KV_NAME --name "mt3chained-step2-tag" --value
"latest"
az keyvault secret set --vault-name $KV_NAME --name "mt3chained-step2-nodejs-tag"
--value "latest"
az keyvault secret set --vault-name $KV_NAME --name "mt3chained-step3-tag" --value
"latest"
az keyvault secret set --vault-name $KV_NAME --name "mt3chained-step4-tag" --value
"latest"
az keyvault secret set --vault-name $KV_NAME --name "mt3chained-step5-tag" --value
"latest"
az keyvault secret set --vault-name $KV_NAME --name "mt3chained-web-tag" --value
"latest"
```

4. On the Key Vault in the Azure Portal, click on **Secrets** to verify they were all created (you may need to click *Refresh*)



Task 7 - Update the Build Pipelines to store image tags in Azure Key Vault

1. Edit the build workflow.yaml create in Task 5.

2. Add 2 env variables.

```
  KV_NAME: <keyVaultName>
  SECRET_NAME: <appName>-tag
```

> Notice the *secretTagName* will match the secret name created in the previous task.

3. Add another task to the end of your workflow. The workflow will execute the same **az keyvault secret set** command that you executed above, except the *value* will be set to the current build tag (**github.run_number**).

```
      - name: Azure login
        uses: azure/login@v1
        with:
          creds: ${{ secrets.AZURE_CREDENTIALS }}
      - run: |
          az keyvault secret set --vault-name ${{ env.KV_NAME }} --name ${{
env.SECRET_NAME }} --value ${{ github.run_number }}
```

4. Create the new **workflow.yaml** in the the directory of all the step repos **\.github\workflows** from the **workflow-steps2.yaml** file, in the directory **C:\k8s\Labs\Module5\workflows**, with the following content:

```
name: <githubRepoName>

on:
  push:
    branches:
```

```
      - main

env:
  APP_NAME: <appName>
  LOGIN_SERVER: <acrServerName>
  KV_NAME: <keyVaultName>
  SECRET_NAME: <appName>-tag

jobs:

  build:
    runs-on: ubuntu-latest
    steps:

      - name: Check-out Repo
        uses: actions/checkout@v2
        with:
          path: Chained/<githubRepoName>

      - name: Check-out MathTrickCore
        uses: actions/checkout@v2
        with:
          repository: <githubAlias>/MathTrickCore
          path: MathTrickCore
          token: ${{ secrets.PAT_GITHUB }}

      # Connect to Azure Container Registry (ACR)
      - name: ACR login
        uses: azure/docker-login@v1
        with:
          login-server: ${{ env.LOGIN_SERVER }}
          username: ${{ secrets.ACR_CLIENT_ID }}
          password: ${{ secrets.ACR_CLIENT_PASSWORD }}
      # Container build and push to a Azure Container Registry (ACR)
      - name: Build, tag, and push image to ACR
        uses: docker/build-push-action@v2
        with:
          context: ./
          file: Chained/<githubRepoName>/Dockerfile
          push: true
          tags: |
            ${{ env.LOGIN_SERVER }}/${{ env.APP_NAME }}:${{ github.run_number }}
            ${{ env.LOGIN_SERVER }}/${{ env.APP_NAME }}:latest
      - name: Azure login
        uses: azure/login@v1
        with:
          creds: ${{ secrets.AZURE_CREDENTIALS }}
      - run: |
          az keyvault secret set --vault-name ${{ env.KV_NAME }} --name ${{
env.SECRET_NAME }} --value ${{ github.run_number }}
```

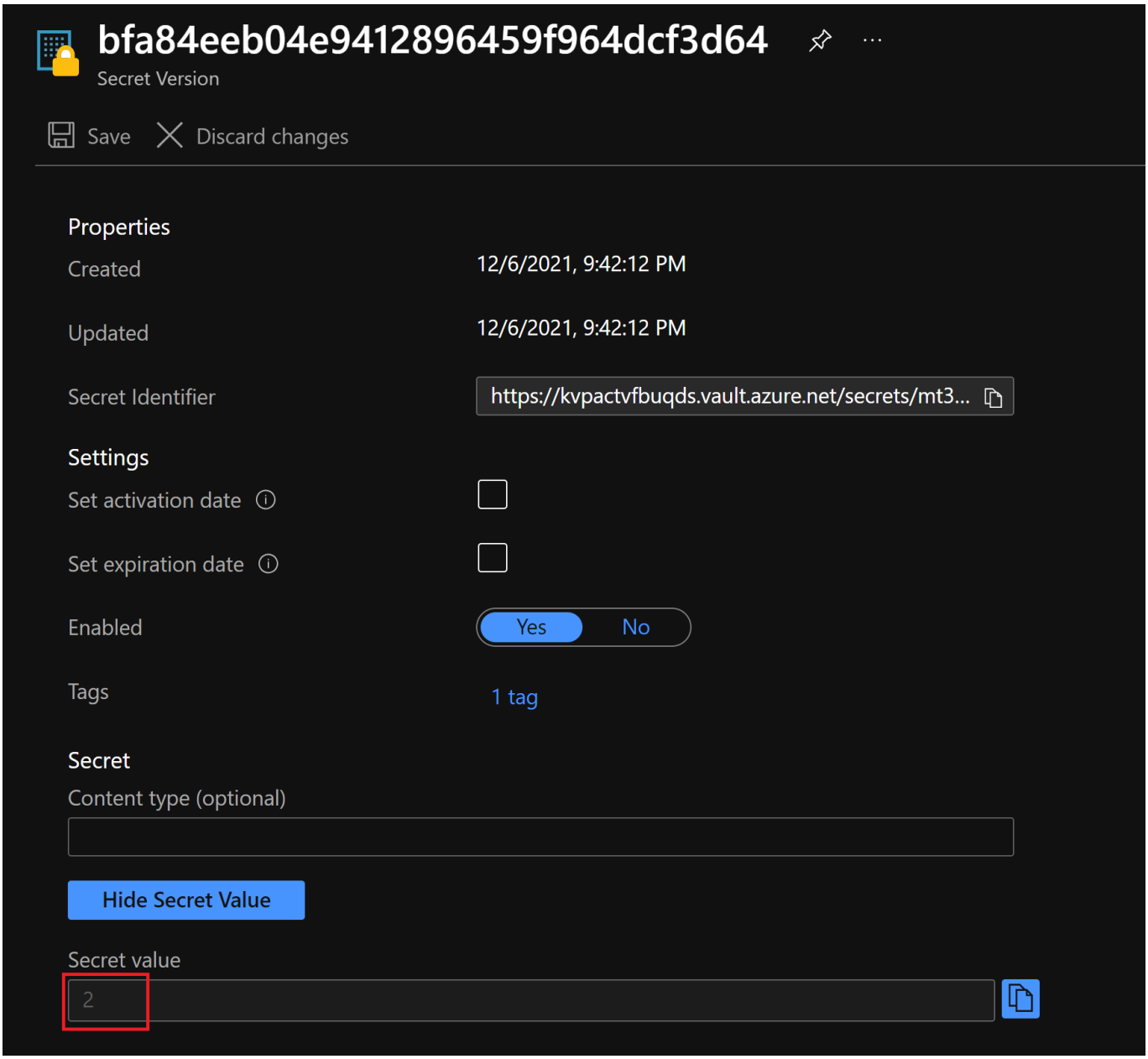
5. Copy the following Azure PowerShell code in the Windows Terminal to replace the **workflow-steps2.yaml** file in the path **C:\k8s\Labs\Module5\workflows** with the correct values and replace the file **(workflow.yaml)** in **\.github\workflows** folder of each step repo (excluding the NodeJS step and web repo).

```
ForEach ($i in 1..5) {
  $PathTemplate="C:\k8s\Labs\Module5\workflows"
  $PathWorkflow="C:\k8s\Labs\MathTrick\Chained\MT3Chained-
Step$i\.github\workflows"
  $original_file = (Join-Path $PathTemplate workflow-steps2.yaml)
  $destination_file = (Join-Path $PathWorkflow workflow.yaml)

  (Get-Content $original_file) | Foreach-Object {
    $_ -replace "<appName>", "mt3chained-step$i" `
       -replace "<githubRepoName>", "MT3Chained-Step$i" `
       -replace "<acrServerName>", $ACR_SERVER_NAME `
       -replace "<keyVaultName>", $KV_NAME `
       -replace "<githubAlias>", $GITHUB_ALIAS
    } | Set-Content $destination_file

  # Commit and push workflow
  cd "C:\k8s\Labs\MathTrick\Chained\MT3Chained-Step$i\"
  Write-Host "==========================MT3Chained-
Step$i==========================" -ForeGround Green
  git add .
  git commit -m "Add workflow.yaml"
  git push
}
```

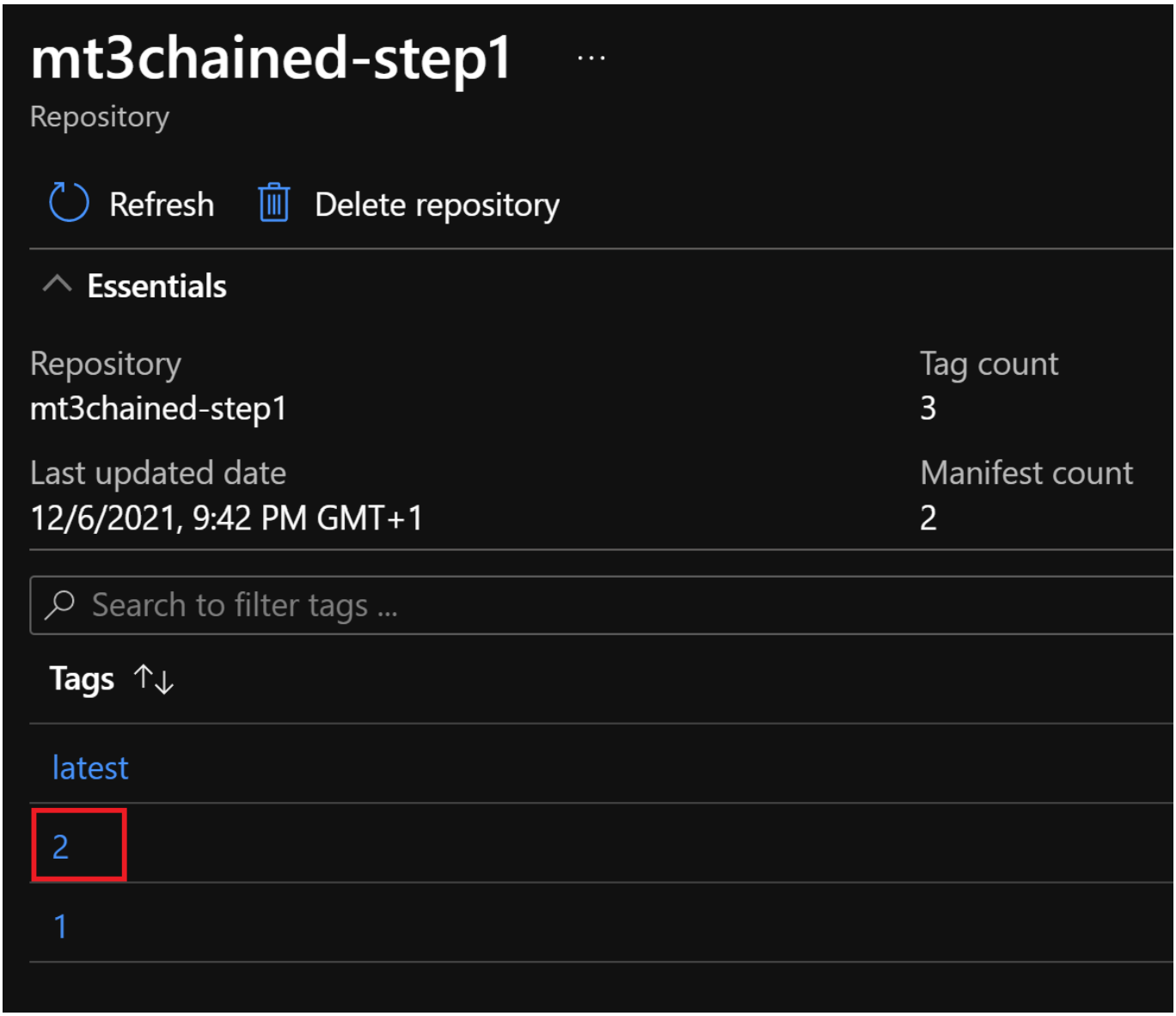
6. When the build completes successfully, go to the Azure Portal and look up the secrets in the key vault, click on the *mt3chained-step1-tag* secret for example.

7. Click on the link below the **CURRENT VERSION** of the secret. Click the **Show Secret Value** button

8. The latest build tag should be in the value of the secret.



9. Which should match the last *github.run_number* tag in the ACR repository *mt3chained-step1*.



Task 8 - Build the Pipelines for the Web and NodeJS microservice

1. The process is similar to the one followed in Task 7 above. We will create a workflow for the *MT3Chained Web* and the *MT3Chained-Step2-NodeJS* repos. This workflow's structure will be simpler than the structure of the ***step*** repositories, since these are self contained projects and don't require the obtention of dependencies of the *MathTrickCore* repo.

2. To create the new **workflow.yaml** in the the directory **\.github\workflows** of *MT3Chained-Web* and the *MT3Chained-Step2-NodeJS* repos you will use the **workflow-web3.yaml** file, in the directory **C:\k8s\Labs\Module5\workflows**. It has 3 main steps: checkout, login into Azure, and image build and upload, and finally store the tag as a secret in the Azure Key Vault.

```
name: <githubRepoName>

on:
  push:
    branches:
      - main

env:
```

```
  APP_NAME: <appName>
  LOGIN_SERVER: <acrServerName>
  KV_NAME: <azureKeyVaultName>
  SECRET_NAME: <secretName>
jobs:

  build:
    runs-on: ubuntu-latest
    steps:

      - name: Check-out Repo
        uses: actions/checkout@v2

      # Connect to Azure Container Registry (ACR)
      - name: ACR login
        uses: azure/docker-login@v1
        with:
          login-server: ${{ env.LOGIN_SERVER }}
          username: ${{ secrets.ACR_CLIENT_ID }}
          password: ${{ secrets.ACR_CLIENT_PASSWORD }}
      # Container build and push to a Azure Container Registry (ACR)
      - name: Build, tag, and push image to ACR
        uses: docker/build-push-action@v2
        with:
          push: true
          tags: |
            ${{ env.LOGIN_SERVER }}/${{ env.APP_NAME }}:${{ github.run_number }}
            ${{ env.LOGIN_SERVER }}/${{ env.APP_NAME }}:latest

      - name: Azure login
        uses: azure/login@v1
        with:
          creds: ${{ secrets.AZURE_CREDENTIALS }}
      - run: |
          az keyvault secret set --vault-name ${{ env.KV_NAME }} --name ${{
env.SECRET_NAME }} --value ${{ github.run_number }}
```

3. Copy the following Azure PowerShell code in the Windows Terminal to replace the **workflow-web3.yaml** file in the path **C:\k8s\Labs\Module5\workflows** with the correct values and replace the **(workflow.yaml)** file with the new content in the **\.github\workflows** folder of web repo.

```
$Path1="C:\k8s\Labs\Module5\workflows"
$Path2="C:\k8s\Labs\MathTrick\Chained\MT3Chained-Web\.github\workflows"
$original_file = (Join-Path $Path1 workflow-web3.yaml)
$destination_file = (Join-Path $Path2 workflow.yaml)
(Get-Content $original_file) | Foreach-Object {
    $_ -replace "<appName>", "mt3chained-web" `
       -replace "<githubRepoName>", "MT3Chained-Web" `
       -replace "<acrServerName>", $ACR_SERVER_NAME `
       -replace "<azureKeyVaultName>", $KV_NAME `
```

```
        -replace "<secretName>", "mt3chained-step1-web-tag"
    } | Set-Content $destination_file
```

4. Commit the changes and push them to the GitHub repository. This will also trigger the workflow execution.

```
cd "C:\k8s\Labs\MathTrick\Chained\MT3Chained-Web"
git add .
git commit -m "Update workflow.yaml"
git push
```

5. We'll repeat the same process for the *MT3Chained-Step2-NodeJS* repo. The definition of the workflow is basically the same. But first we need to create the **\.github\workflows** folder of NodeJS repo.

```
New-Item -Path "C:\k8s\Labs\MathTrick\Chained\MT3Chained-Step2-
NodeJS\.github\workflows" -ItemType Directory
```

6. Now, run the following script:

```
$Path1="C:\k8s\Labs\Module5\workflows"
$Path2="C:\k8s\Labs\MathTrick\Chained\MT3Chained-Step2-NodeJS\.github\workflows"
$original_file = (Join-Path $Path1 workflow-web3.yaml)
$destination_file = (Join-Path $Path2 workflow.yaml)
(Get-Content $original_file) | Foreach-Object {
    $_ -replace "<appName>", "mt3chained-step2-nodejs" `
       -replace "<githubRepoName>", "MT3Chained-Step2-NodeJS" `
       -replace "<acrServerName>", $ACR_SERVER_NAME `
       -replace "<azureKeyVaultName>", $KV_NAME `
       -replace "<secretName>", "mt3chained-step2-nodejs-tag"
    } | Set-Content $destination_file
```

7. Init and Commit the changes and push them to the GitHub repository (As you can see we have some steps more as this repo wasn't initialized before). This will also trigger the workflow execution.

```
cd "C:\k8s\Labs\MathTrick\Chained\MT3Chained-Step2-NodeJS"
git init
git add .
git commit -m "Initial check-in"
git remote add origin git@github.com:pabloameijeirascanay/MT3Chained-Step2-
NodeJS.git
git branch -M main
git push -u origin main
```

8. Once the execution is over, verify the images in your Azure Container Registry and the tags in your Azure Key Vault.

9. When all the workflows have run at least once. You can verify their status by selecting "Actions" tab and checking the execution log.

Task 9 - Build a single Release Pipelines that uses Helm to update the AKS cluster

Now that all of the microservice images are automatically built, pushed and their tags are saved, it's time to create a workflow that is triggered automatically and updates the AKS cluster when **ANY** microservice is modified. This will require a small change in all the previously added workflows, as well as a workflow that contains a special trigger.

GitHub does not support triggering child workflows from outside of a repository. This means that, we either bundle the entire solution in a single repository, which would be undesirable, or we'll have to find a way to achieve the same behavior. Fortunately, this can be done by using a feature from the GitHub API called workflow_dispatch. Let's see how to add this event to our previous pipelines.

1. The use of the GitHub API requires us to authenticate ourselves to perform the required request. This means we need a personal access token available in the repositories that will trigger a deployment when updated. In Task 4 we have already taken care of this step but make sure that all our **step** repositories (Including NodeJS) and the **web** repository have the PAT_GITHUB secret created. This step is mandatory.

2. Once we have the required secrets in place, we have to add the following step at the end of each of our existing workflows.

```
      # Issue workflow dispatch event to trigger Helm chart deployment
      - name: Invoke workflow in another repo with inputs
        uses: benc-uk/workflow-dispatch@v1
        with:
          workflow: Math-Trick-3---Chained
          repo: <githubAlias>/Math-Trick-3---Chained
          token: ${{ secrets.PAT_GITHUB }}
```

The code shown above uses an action to perform the API request that will trigger the workflow_dispatch event for the required repository and workflow. The `workflow` and `repo` parameters determine where the event will be sent. The `token` parameter should be passed the PAT_GITHUB secret we have defined for our repositories.

3. Copy the following Azure PowerShell code in the Windows Terminal to replace the **workflow-steps3.yaml** file in the path **C:\k8s\Labs\Module5\workflows** with the correct values and replace the **(workflow.yaml)** content in **\.github\workflows** folder of all the step repos.

```
ForEach ($i in 1..5) {
  $PathTemplate="C:\k8s\Labs\Module5\workflows"
  $PathWorkflow="C:\k8s\Labs\MathTrick\Chained\MT3Chained-
Step$i\.github\workflows"
```

```
  $original_file = (Join-Path $PathTemplate workflow-steps3.yaml)
  $destination_file = (Join-Path $PathWorkflow workflow.yaml)

  (Get-Content $original_file) | Foreach-Object {
    $_ -replace "<appName>", "mt3chained-step$i" `
       -replace "<githubRepoName>", "MT3Chained-Step$i" `
       -replace "<acrServerName>", $ACR_SERVER_NAME `
       -replace "<keyVaultName>", $KV_NAME `
       -replace "<githubAlias>", $GITHUB_ALIAS
    } | Set-Content $destination_file
}
```

4. Copy the following Azure PowerShell code in the Windows Terminal to replace the **workflow-web4.yaml** file in the path **C:\k8s\Labs\Module5\workflows** with the correct values and replace the **(workflow.yaml)** content in **\.github\workflows** folder of web repo.

```
$Path1="C:\k8s\Labs\Module5\workflows"
$Path2="C:\k8s\Labs\MathTrick\Chained\MT3Chained-Web\.github\workflows"
$original_file = (Join-Path $Path1 workflow-web4.yaml)
$destination_file = (Join-Path $Path2 workflow.yaml)
(Get-Content $original_file) | Foreach-Object {
    $_ -replace "<appName>", "mt3chained-web" `
       -replace "<githubRepoName>", "MT3Chained-Web" `
       -replace "<acrServerName>", $ACR_SERVER_NAME `
       -replace "<azureKeyVaultName>", $KV_NAME `
       -replace "<secretName>", "mt3chained-web-tag" `
       -replace "<githubAlias>", $GITHUB_ALIAS
    } | Set-Content $destination_file
```

5. We'll repeat the same process for the *MT3Chained-Step2-NodeJS* repo as the web repo above. The definition of the workflow is basically the same.

```
$Path1="C:\k8s\Labs\Module5\workflows"
$Path2="C:\k8s\Labs\MathTrick\Chained\MT3Chained-Step2-NodeJS\.github\workflows"
$original_file = (Join-Path $Path1 workflow-web4.yaml)
$destination_file = (Join-Path $Path2 workflow.yaml)
(Get-Content $original_file) | Foreach-Object {
    $_ -replace "<appName>", "mt3chained-step2-nodejs" `
       -replace "<githubRepoName>", "MT3Chained-Step2-NodeJS" `
       -replace "<acrServerName>", $ACR_SERVER_NAME `
       -replace "<azureKeyVaultName>", $KV_NAME `
       -replace "<secretName>", "mt3chained-step2-nodejs-tag" `
       -replace "<githubAlias>", $GITHUB_ALIAS
    } | Set-Content $destination_file
```

DO NOT PUSH THE CHANGES YET, AS WE STILL NEED TO ADD THE WORKFLOW THAT WILL RECEIVE THE EVENTS

6. Once the code repositories are ready, we need to create a workflow that will issue the deployment of our Helm chart. This workflow must be triggered in two circumstances. The first one would be the reception of the workflow_dispatch event, which would correspond to a change made to one of the subprojects (subrepos: step repos, NodeJS and web repo) that are part of the chart. The second is when changes to the Helm repository itself are performed. To do this, we have to make a slight modification to the on section of our workflow:

```
on:
  workflow_dispatch: ~
  push:
    branches: [main]
```

This new definition will cause the workflow to execute following the rules we have previously explained.

7. This repository will connect to our Azure KeyVault and fetch the secrets corresponding to the versions of each of the apps in our solution. These secrets will be used to perform a variable replacement of the entries in the values.yaml file in the chart. We need two steps to perform these tasks. The convenience and versatility of GitHub actions and the workflow syntax can be seen again in this brief example. The chosen actions eliminate the need to manually perform the obtention of the secrets and the replacement of the values in the .yaml code, providing an easy to use alternative for common tasks.

```
      # Get secrets
      - name: Get Keyvault secrets
        uses: Azure/get-keyvault-secrets@v1
        with:
          keyvault: ${{ env.KV_NAME }}
          secrets: 'mt3chained-step1-tag,mt3chained-step2-nodejs-tag,mt3chained-
step2-tag,mt3chained-step3-tag,mt3chained-step4-tag,mt3chained-step5-
tag,mt3chained-web-tag'
        id: versiontags

      - name: 'Set version tags'
        uses: microsoft/variable-substitution@v1
        with:
          files: mt3chained/values.yaml
        env:
          tags.mt3chainedweb: ${{ steps.versiontags.outputs.mt3chained-web-tag }}
          tags.mt3chainedstep1: ${{ steps.versiontags.outputs.mt3chained-step1-tag
}}
          tags.mt3chainedstep2: ${{ steps.versiontags.outputs.mt3chained-step2-tag
}}
          tags.mt3chainedstep2nodejs: ${{ steps.versiontags.outputs.mt3chained-
step2-nodejs-tag }}
          tags.mt3chainedstep3: ${{ steps.versiontags.outputs.mt3chained-step3-tag
}}
          tags.mt3chainedstep4: ${{ steps.versiontags.outputs.mt3chained-step4-tag
}}
          tags.mt3chainedstep5: ${{ steps.versiontags.outputs.mt3chained-step5-tag
}}
```

8. Before we attempt to deploy our chart, we need to set the AKS context.

```
      - name: Azure Kubernetes set context
        uses: Azure/aks-set-context@v1
        with:
          creds: ${{ secrets.AZURE_CREDENTIALS }}
          resource-group: ${{ env.AKS_RESOURCE_GROUP }}
          cluster-name: ${{ env.AKS_NAME }}
```

9. Once our AKS context is set, we must install Helm.

```
      - name: Helm tool installer
        uses: Azure/setup-helm@v1
```

10. Finally, we just have to run the Helm command to install/upgrade a release.

```
      - name: Deploy
        run: |
          helm upgrade --install chained mt3chained/
```

11. The complete workflow for the Helm repository will look like this:

```
name: Math-Trick-3---Chained

on:
  workflow_dispatch: ~
  push:
    branches: [main]

env:
  KV_NAME: <azureKeyVaultName>
  AKS_NAME: <aksName>
  AKS_RESOURCE_GROUP: <aksResourceGroup>

jobs:

  deploy-to-aks:
    runs-on: ubuntu-latest
    steps:

      - name: 'Checkout'  # Checkout the repository code.
        uses: 'actions/checkout@v1'

      # Connect to Azure
```

```
      - name: Azure login
        uses: azure/login@v1
        with:
          creds: ${{ secrets.AZURE_CREDENTIALS }}

      # Get secrets
      - name: Get Keyvault secrets
        uses: Azure/get-keyvault-secrets@v1
        with:
          keyvault: ${{ env.KV_NAME }}
          secrets: 'mt3chained-step1-tag,mt3chained-step2-nodejs-tag,mt3chained-
step2-tag,mt3chained-step3-tag,mt3chained-step4-tag,mt3chained-step5-
tag,mt3chained-web-tag'
        id: versiontags

      - name: 'Set version tags'
        uses: microsoft/variable-substitution@v1
        with:
          files: mt3chained/values.yaml
        env:
          tags.mt3chainedweb: ${{ steps.versiontags.outputs.mt3chained-web-tag }}
          tags.mt3chainedstep1: ${{ steps.versiontags.outputs.mt3chained-step1-tag
}}
          tags.mt3chainedstep2: ${{ steps.versiontags.outputs.mt3chained-step2-tag
}}
          tags.mt3chainedstep2nodejs: ${{ steps.versiontags.outputs.mt3chained-
step2-nodejs-tag }}
          tags.mt3chainedstep3: ${{ steps.versiontags.outputs.mt3chained-step3-tag
}}
          tags.mt3chainedstep4: ${{ steps.versiontags.outputs.mt3chained-step4-tag
}}
          tags.mt3chainedstep5: ${{ steps.versiontags.outputs.mt3chained-step5-tag
}}

      - name: Azure Kubernetes set context
        uses: Azure/aks-set-context@v1
        with:
          creds: ${{ secrets.AZURE_CREDENTIALS }}
          resource-group: ${{ env.AKS_RESOURCE_GROUP }}
          cluster-name: ${{ env.AKS_NAME }}

      - name: Helm tool installer
        uses: Azure/setup-helm@v1

      - name: Deploy
        run: |
          helm upgrade --install chained mt3chained/
```

12. But first we need to create the **\.github\workflows** folder of Helm repo.

```
New-Item -Path "C:\k8s\Labs\MathTrick\Chained\Helm\.github\workflows" -ItemType
Directory
```

13. Copy the following Azure PowerShell code in the Windows Terminal to replace the **workflow-helm.yaml** file in the path **C:\k8s\Labs\Module5\workflows** with the correct values and give it the correct file name **(workflow.yaml)** copying it in **\.github\workflows** folder of Helm repo.

```
$Path1="C:\k8s\Labs\Module5\workflows"
$Path2="C:\k8s\Labs\MathTrick\Chained\Helm\.github\workflows"
$original_file = (Join-Path $Path1 workflow-helm.yaml)
$destination_file = (Join-Path $Path2 workflow.yaml)
(Get-Content $original_file) | Foreach-Object {
    $_ -replace "<azureKeyVaultName>", $KV_NAME `
       -replace "<aksName>", $AKS_NAME `
       -replace "<aksResourceGroup>", $AKS_RESOURCE_GROUP
    } | Set-Content $destination_file
```

14. Copy the following Azure PowerShell code in the Windows Terminal to replace the **values.yaml** file in the path **C:\k8s\Labs\MathTrick\Chained\Helm\mt3chained** with your correct Azure Container Registry name.

```
$Path="C:\k8s\Labs\MathTrick\Chained\Helm\mt3chained"
(Get-Content (Join-Path $Path values.yaml)).replace("kizacr.azurecr.io",
$ACR_SERVER_NAME) | Set-Content (Join-Path $Path values.yaml)
```

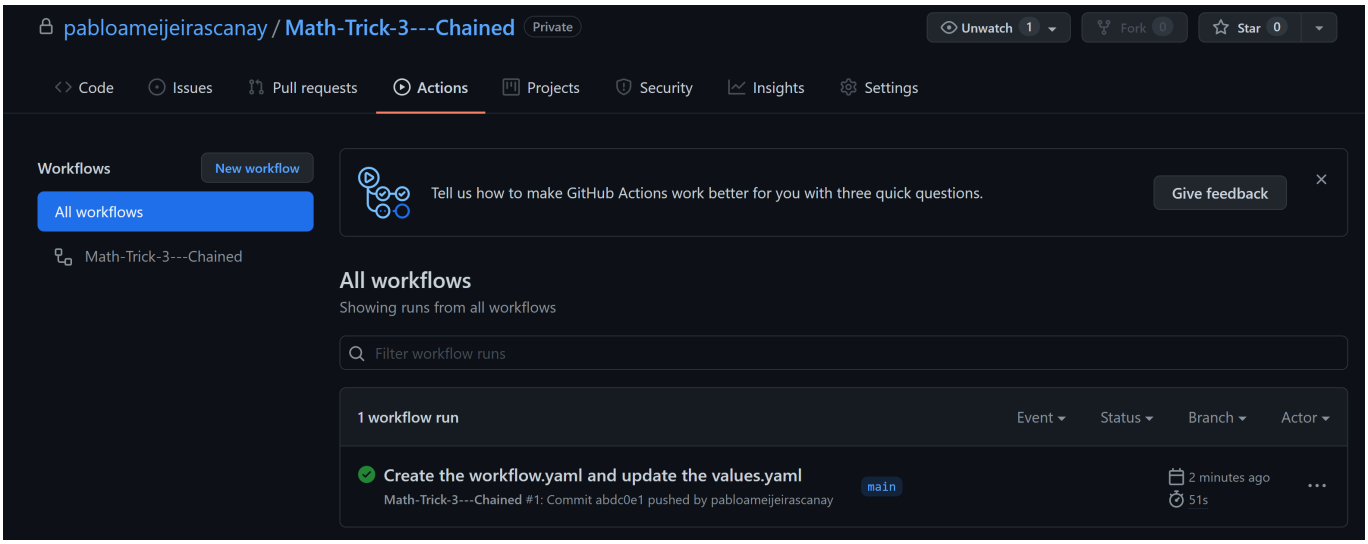
Task 10 - Push the changes

1. Once the Helm workflow is ready, we'll commit and push the changes. This will trigger the execution of the workflow.

```
cd "C:\k8s\Labs\MathTrick\Chained\Helm"
git add .
git commit -m "Create the workflow.yaml and update the values.yaml"
git push
```

> Note that the execution of this workflow requires the docker images for all the projects to be present in the ACR. This means at least one correct execution of the workflow for each of the projects is required.

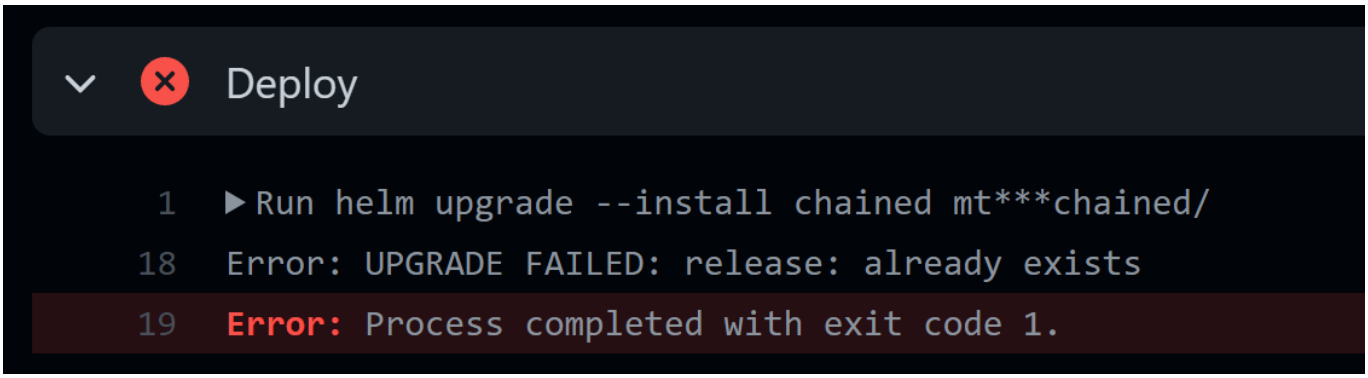
2. Wait to see the Helm github action finished.



3. Now, we can push the changes for the ***step***, **NodeJS** and ***web*** repos. From this point onwards, any change to these projects will also cause the deployment of the chart. Copy the following script in your Windows Terminal:

```
ForEach ($i in 1..5) {
  cd "C:\k8s\Labs\MathTrick\Chained\MT3Chained-Step$i\"
  Write-Host "=========================MT3Chained-
Step$i=========================" -ForeGround Green
  git add .
  git commit -m "Add workflow.yaml"
  git push
  Start-Sleep -s 60
}

cd "C:\k8s\Labs\MathTrick\Chained\MT3Chained-Step2-NodeJS\"
git add .
git commit -m "Add workflow.yaml"
git push
Start-Sleep -s 60

cd "C:\k8s\Labs\MathTrick\Chained\MT3Chained-Web\"
git add .
git commit -m "Add workflow.yaml"
git push
```

> The command *Start-Sleep -s 60* is just to avoid the following error:



> This happens because each of the above pushes will create a trigger for the workflow in the *Math-Trick-3---Chained* repo, and without the *Start-Sleep -s 60* command there would be 7 *helm upgrade --install chained mt3chained/* running at the same time.

---

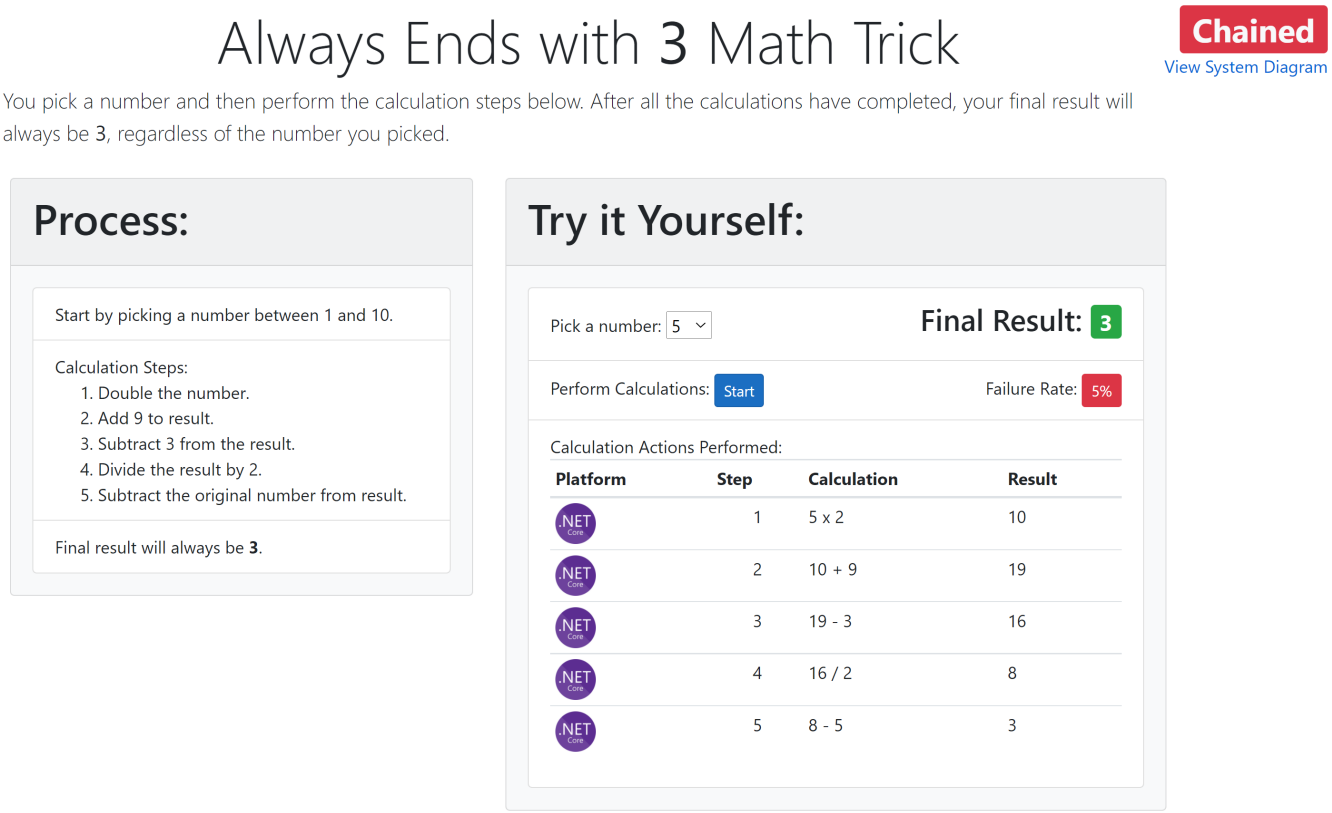
4. Assuming the pipeline succeeds, check the cluster to verify everything has been deployed and is running.

```
kubectl get all -n chained
```

```
PS C:\k8s\Labs\MathTrick\Chained\Helm> kubectl get all -n chained
NAME                                        READY   STATUS    RESTARTS   AGE
pod/mt3chained-step1-dep-5c5ccbcf47-f5wv9   1/1     Running   0          2m5s
pod/mt3chained-step2-dep-84f665f4d9-csgkz   1/1     Running   0          2m5s
pod/mt3chained-step3-dep-67f846d9b-h9fzn    1/1     Running   0          2m5s
pod/mt3chained-step4-dep-67f59c6fdb-plfwz   1/1     Running   0          2m5s
pod/mt3chained-step5-dep-cb4fdb848-j2kvl    1/1     Running   0          2m5s
pod/mt3chained-web-dep-77dc995794-s7skh     1/1     Running   0          2m5s

NAME                            TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)        AGE
service/mt3chained-step1-svc    ClusterIP      10.0.237.19    <none>          5010/TCP       2m5s
service/mt3chained-step2-svc    ClusterIP      10.0.100.2     <none>          5020/TCP       2m5s
service/mt3chained-step3-svc    ClusterIP      10.0.190.221   <none>          5030/TCP       2m5s
service/mt3chained-step4-svc    ClusterIP      10.0.94.127    <none>          5040/TCP       2m5s
service/mt3chained-step5-svc    ClusterIP      10.0.35.172    <none>          5050/TCP       2m5s
service/mt3chained-web-svc      LoadBalancer   10.0.62.1      20.93.209.221   80:30386/TCP   2m5s

NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mt3chained-step1-dep    1/1     1            1           2m6s
deployment.apps/mt3chained-step2-dep    1/1     1            1           2m6s
deployment.apps/mt3chained-step3-dep    1/1     1            1           2m6s
deployment.apps/mt3chained-step4-dep    1/1     1            1           2m6s
deployment.apps/mt3chained-step5-dep    1/1     1            1           2m6s
deployment.apps/mt3chained-web-dep      1/1     1            1           2m6s

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/mt3chained-step1-dep-5c5ccbcf47   1         1         1       2m6s
replicaset.apps/mt3chained-step2-dep-84f665f4d9   1         1         1       2m6s
replicaset.apps/mt3chained-step3-dep-67f846d9b    1         1         1       2m6s
replicaset.apps/mt3chained-step4-dep-67f59c6fdb   1         1         1       2m6s
replicaset.apps/mt3chained-step5-dep-cb4fdb848    1         1         1       2m6s
replicaset.apps/mt3chained-web-dep-77dc995794     1         1         1       2m6s
PS C:\k8s\Labs\MathTrick\Chained\Helm>
```

5. Open a browser and enter the EXTERNAL-IP address listed for the Load Balancer service.

6. Click the Start button to ensure all the services are connected.



7. Verify that the Helm pipeline used all the tags from the Key Vault and replaced all the tags successfully, including the ACR.

```
helm get values chained -a
```

```
PS C:\k8s\Labs\MathTrick\Chained\MT3Chained-Web> helm get values chained -a
COMPUTED VALUES:
namespace: chained
platform: dotnet
repo: acrpactqkyvnju.azurecr.io
tags:
  mt3chainedstep1: "3"
  mt3chainedstep2: "3"
  mt3chainedstep2nodejs: "2"
  mt3chainedstep3: "3"
  mt3chainedstep4: "3"
  mt3chainedstep5: "3"
  mt3chainedweb: "6"
PS C:\k8s\Labs\MathTrick\Chained\MT3Chained-Web>
```

8. Finally, confirm that the entire process is working correctly by adding a test file in any microservice and check it in.

```
## Change into any of the microservice folders
echo "Test" > pipeline.txt
git add .
```

```
git commit -m "Pipeline test"
git push
```

9. Wait a few minutes to ensure that **ONLY** that effected Pod was automatically replaced with the new version.

```
kubectl get all -n chained
```

```
PS C:\k8s\Labs\MathTrick\Chained\MT3Chained-Step1> kubectl get all -n chained
NAME                                        READY   STATUS    RESTARTS   AGE
pod/mt3chained-step1-dep-896fdddf8-wjq42    1/1     Running   0          29s
pod/mt3chained-step2-dep-84f665f4d9-csgkz   1/1     Running   0          31m
pod/mt3chained-step3-dep-67f846d9b-h9fzn    1/1     Running   0          31m
pod/mt3chained-step4-dep-67f59c6fdb-plfwz   1/1     Running   0          31m
pod/mt3chained-step5-dep-cb4fdb848-j2kvl    1/1     Running   0          31m
pod/mt3chained-web-dep-77dc995794-s7skh     1/1     Running   0          31m
```

CONGRATULATIONS!!! You now have a "practical" pipeline you can use as a template for complex microservices applications.