Microsoft Power Platform

Microsoft Power Platform

Dev in a day

Lab 05 Application lifecycle management/ December 2022

Table of Contents

Lab Scenario

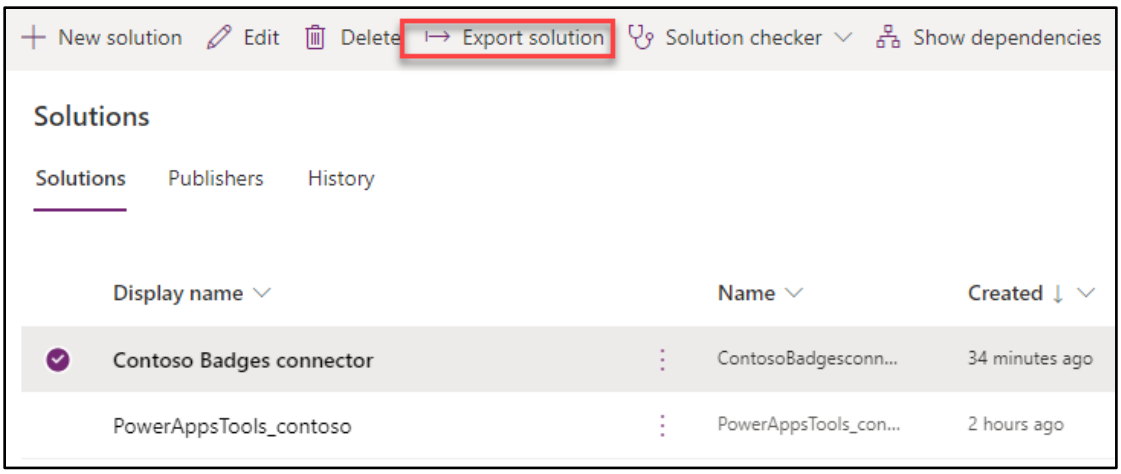
Working as part of the PrioritZ fusion team you will be configuring GitHub Actions using the Power Platform Build Tools to automate the team's deployments.

Exercise 1 – Promote Solution to Test Environment

In this exercise, you will manually export the export the Contoso Badges connector solution from the Dev environment and import it to Test environment. You will be doing this promotion manually for this solution so you see how it can be done. However, in a real project it could all be automated using the build tasks you will use in the remaining parts of this lab.

Task 1: Export solution.

1. Navigate to Power Apps maker portal and make sure you are in your dev environment.
2. Select **Solutions**.
3. Select the **Contoso Badges connector** solution and click **Export solution**.

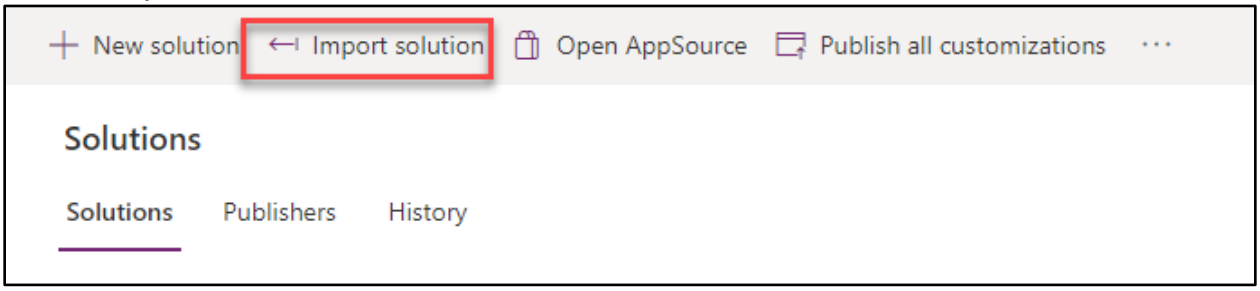


4. Click **Publish** and wait for the publishing to complete.
5. Click **Next**.
6. Select **Managed** and click **Export**.
7. Wait for the solution to be exported.
8. Click **Download**.

Note: In the interest of time, we've only added steps to export the solution as managed. It is best practice to also export the unmanaged solution and keep it available for future edits if needed. The steps to follow are essentially the same regardless of the exported solution being managed, or unamanged.

Task 2: Import solution

1. Navigate to [Power Apps maker portal]() and select your **Test** environment.
2. Select **Solutions**.
3. Click **Import solution**.



4. Click **Browse**.

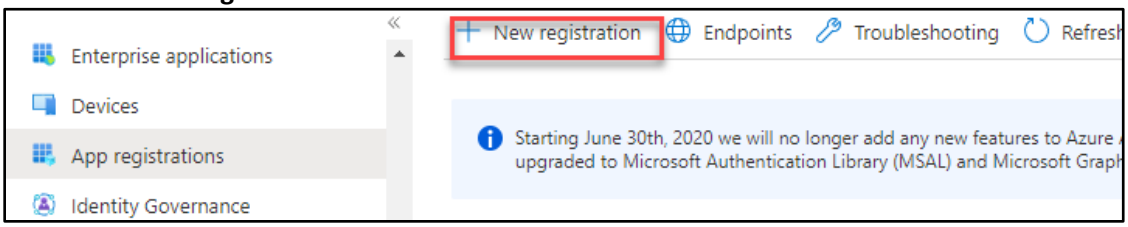
5. Select the solution you exported from the Dev environment and click **Open**.
6. Click **Next**.
7. Click **Import** and wait for the import to complete.
8. The solution should import successfully.
9. Do not navigate away from this page.

Exercise 2 – Configure a Service Principal

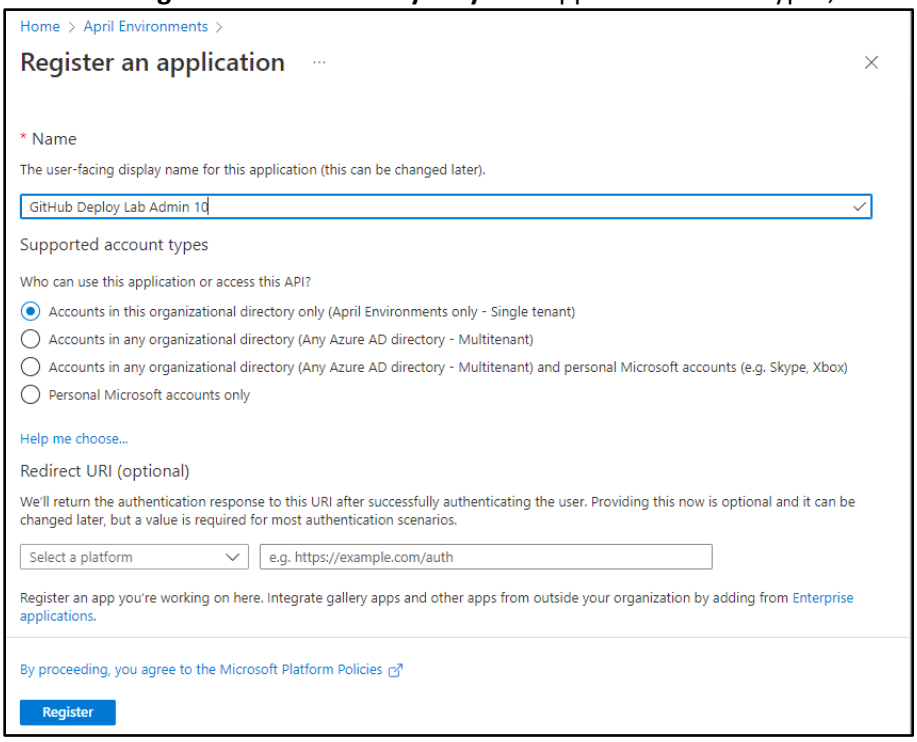
In this exercise, you will create service principal. The service principal will be used by the GitHub workflow actions, so they don't execute under your individual user identity.

Task 1: Create app registration

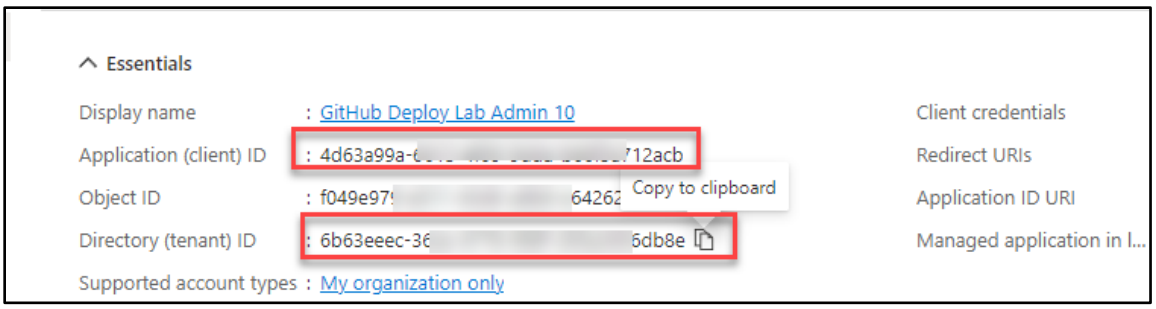
1. Navigate to https://portal.azure.com/
2. Select **Azure active directory**.
3. Select **App registrations**.
4. Click **+ New registration**.



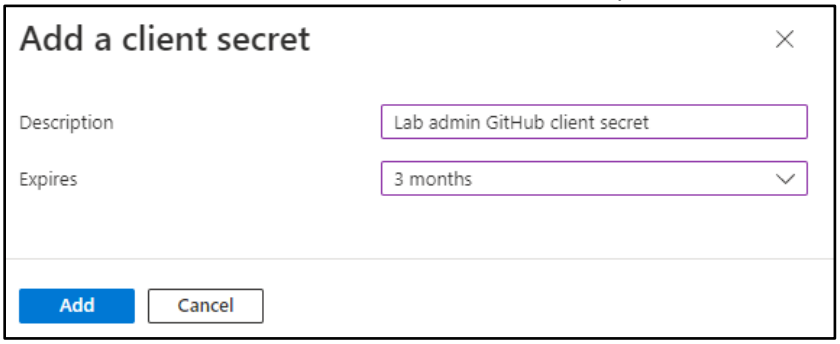
5. Enter **GitHub Deploy <Your Name>** e.g., **GitHub Deploy Lab Admin 1** for Name, select **Accounts in this organizational directory only** for Supported account types, and click **Register**.



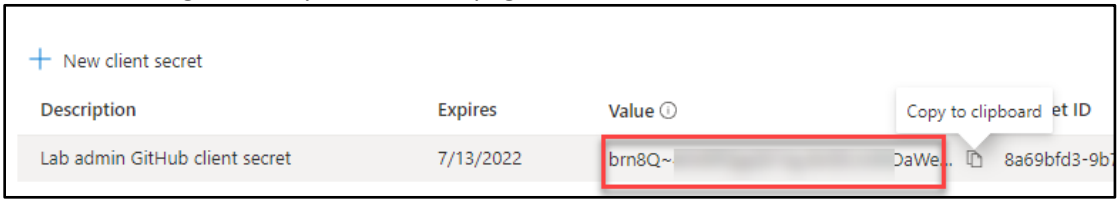
6. Copy the **Application (client) ID** and the Directory (tenant) ID. Keep the ids in a notepad, you will need them in future steps.



7. Select **Certificates and secrets**.
8. Click **+ New client secret**.
9. Enter **Lab admin GitHub client secret** for Description, select **3 months** for Expires, and click **Add**.



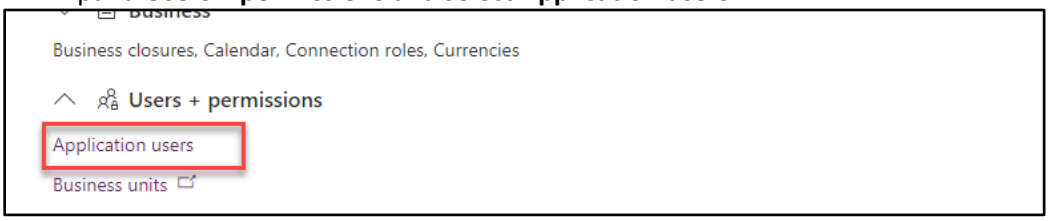
10. Copy the **Value** and save it on a notepad, you will need this value in future step. The value will not show again after you leave this page.



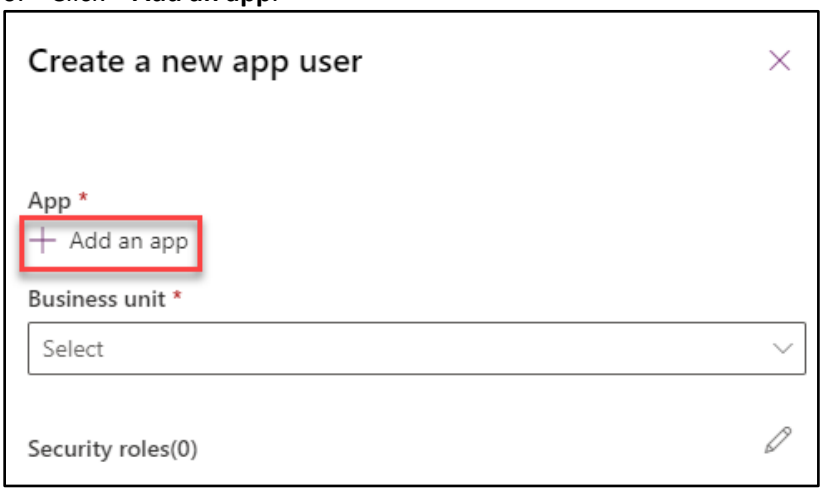
Task 2: Create app user in Dataverse

In this task, you will be registering the app you created in Azure Active Directory into the dev and test Dataverse environments. You will also be assigning a security role that will allow the service principal to deploy solutions.

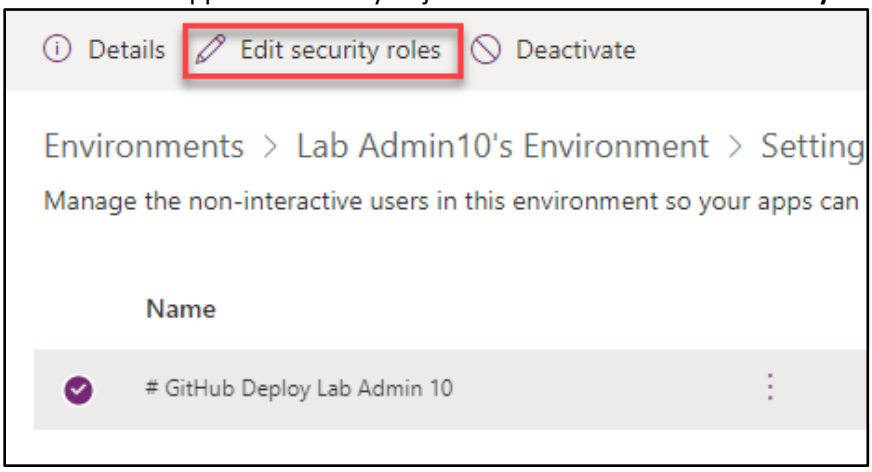
1. Navigate to Power Platform admin center
2. Select **Environments**.
3. Select your **Dev** environment and click **Settings**.
4. Expand **Users + permissions** and select **Application users**.



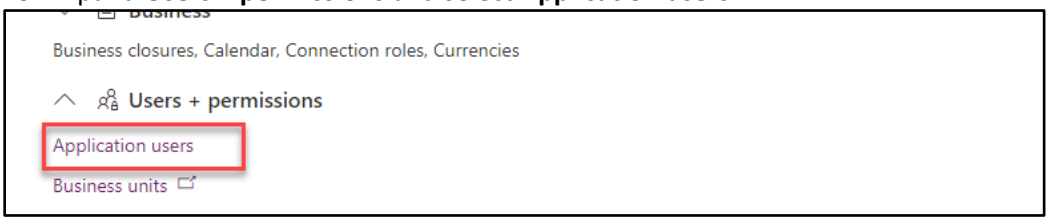
5. Click **+ New app user**.
6. Click **+ Add an app**.



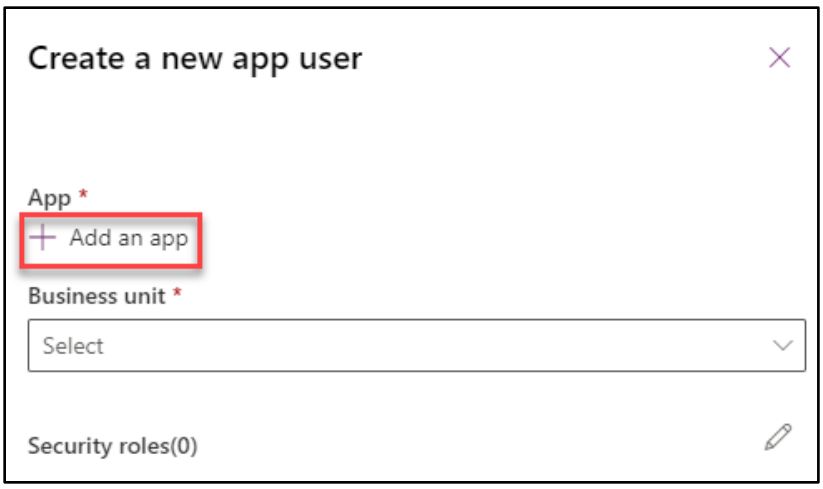
7. Select application registration you created and click **Add**.
8. Select your Business unit and click **Create**. There should only be one unless you created more.
9. Select the application user you just created and click **Edit security roles**.



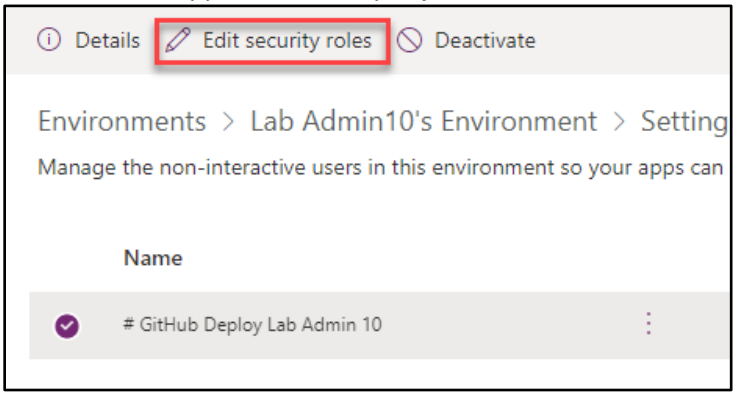
10. Select **System administrator** and click **Save**.
11. Select **Environments** again.
12. Select your **Test** environment and click **Settings**.
13. Expand **Users + permissions** and select **Application users**.



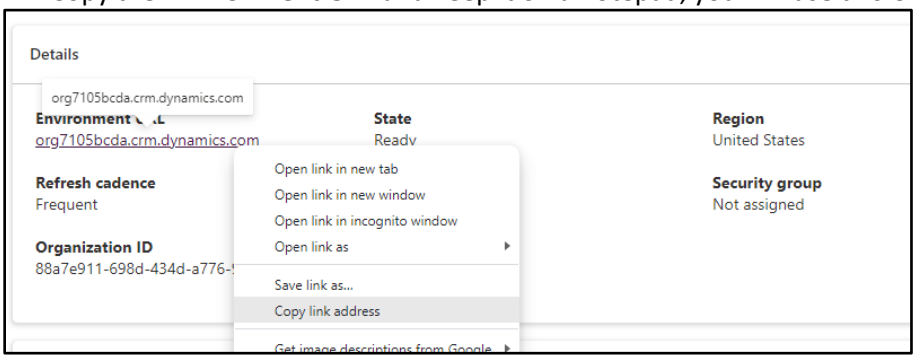
14. Click **+ New app user**.
15. Click **+ Add an app**.



16. Select application registration you created and click **Add**.
17. Select your Business unit and click **Create**.
18. Select the application user you just created and click **Edit security roles**.



19. Select **System administrator** and click **Save**.
20. Select **Environments** and click to open the **Dev** environment.
21. Copy the **Environment URL** and keep it on a notepad, you will use this URL in future steps.



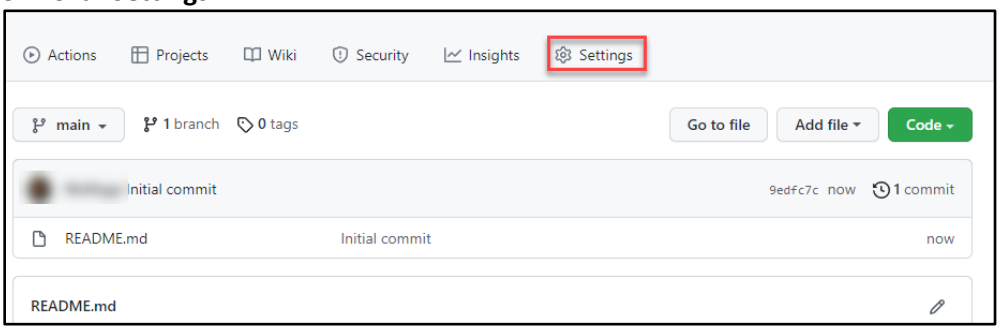
22. Select **Environments** and click to open the **Test** environment.
23. Copy the **Environment URL** and keep it on a notepad, you will this URL in future steps.

Exercise 3 – Create a GitHub Repo

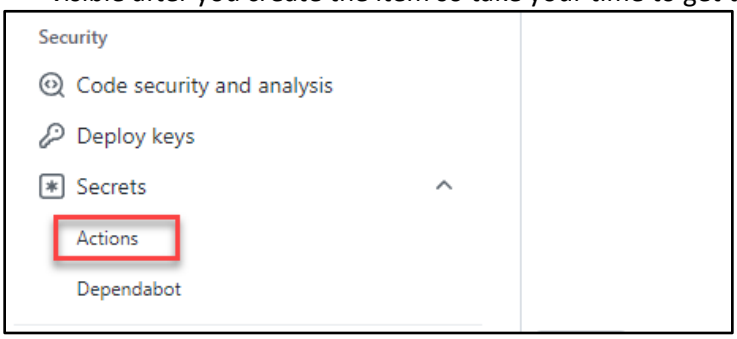
In this exercise, you will create a GitHub repository and add repository secrets.

Task 1: Create repository

1. Navigate to https://github.com/
2. Sign in or signup for GitHub.
3. Click **New repository**.
4. Enter **PrioritZ** for Repository name, select **Public**, check the **Add a README file**, and click **Create repository**.
5. Click **Settings**.

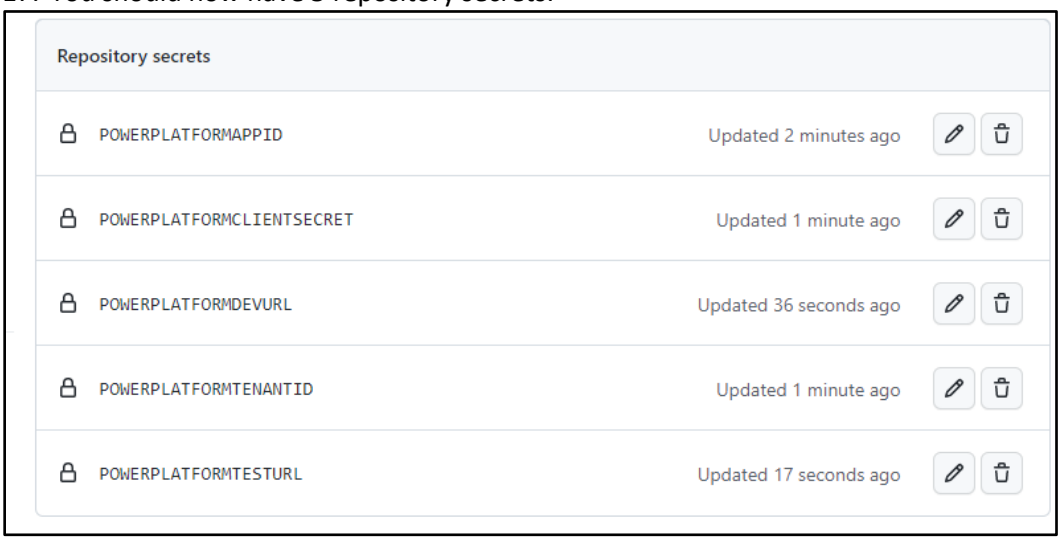


6. Go to the **Security** section, expand **Secrets** and select **Actions**. The values you provide will not be visible after you create the item so take your time to get the values correct.



7. Click **New repository secret**.
8. Enter **PowerPlatformAppID** for Name and paste the **Application (client) ID** from your notepad in the **Value** field and click **Add secret**.
9. Click **New repository secret** again.
10. Enter **PowerPlatformClientSecret** for Name and paste the secret **Value** from your notepad in the **Value** field and click **Add secret**.
11. Click **New repository secret** again.
12. Enter **PowerPlatformTenantID** for Name and paste the secret **Tenant ID** from your notepad in the **Value** field and click **Add secret**.
13. Click **New repository secret** again.
14. Enter **PowerPlatformDevUrl** for Name and paste the secret **Dev environment URL** from your notepad in the **Value** field and click **Add secret**.
15. Click **New repository secret** one more time.
16. Enter **PowerPlatformTestUrl** for Name and paste the secret **Test environment URL** from your notepad in the **Value** field and click **Add secret**.

17. You should now have **5** repository secrets.



18. Do not navigate away from this page.

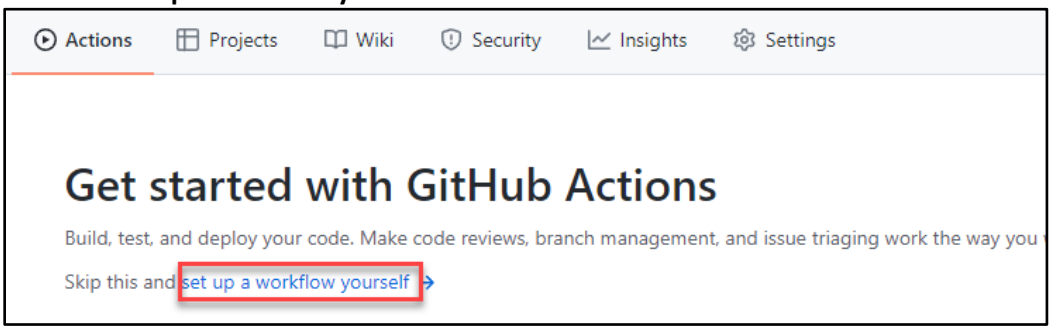
Exercise 4 – Export and Branch

In this exercise, you will setup a workflow action, and add steps that will export the solution from dev environment and create a new branch.

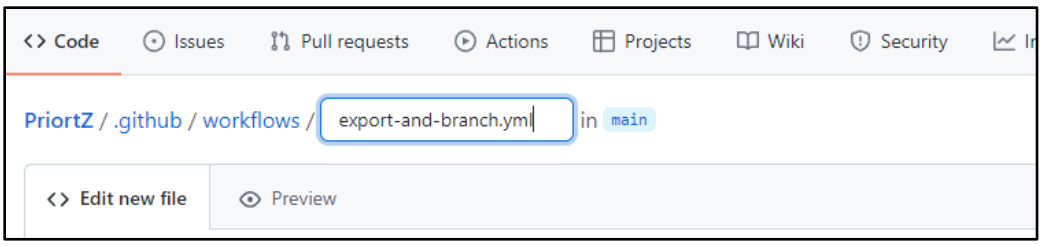
Task 1: Export and branch

In this task you will create the workflow definition using the YAML provided. The action YAML uses two space indentation so follow that carefully as you build the workflow definition. If in doubt, review the indentation shown in the images.

1. Select the **Actions** tab.
2. Click **set up a workflow yourself**.

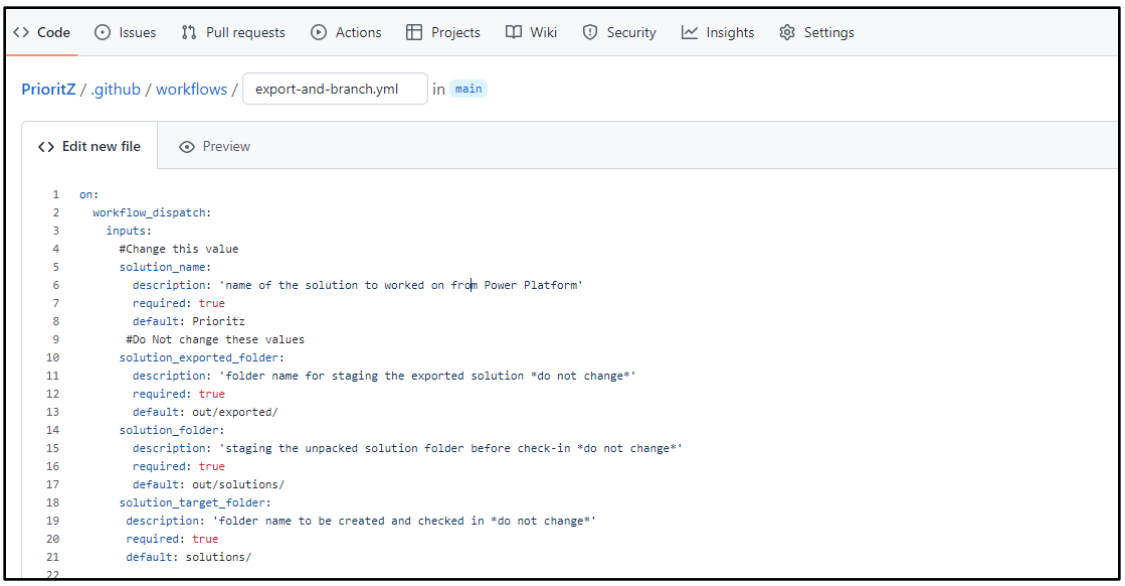


3. Change the file name to it **export-and-branch.yml**.



4. Add the below YAML snippet. This defines the action trigger and some input parameters that could be changed when manually running the action.

```
on:
  workflow_dispatch:
    inputs:
      #Change this value
      solution_name:
        description: 'name of the solution to worked on from Power Platform'
        required: true
        default: Prioritz
       #Do Not change these values
      solution_exported_folder:
        description: 'folder name for staging the exported solution *do not
change*'
        required: true
        default: out/exported/
      solution_folder:
        description: 'staging the unpacked solution folder before check-in *do
not change*'
        required: true
        default: out/solutions/
      solution_target_folder:
       description: 'folder name to be created and checked in *do not change*'
       required: true
       default: solutions/
```



5. Setup the workflow. Add the below YAML snippet after the last snippet. This sets up the jobs and identifies the first job as export-from-dev. This also defines the steps with the first one checking out the current main branch content.

```
jobs:
  export-from-dev:
    runs-on: windows-latest
    steps:
    - uses: actions/checkout@v2
      with:
        lfs: true
```

6. Next you will export both an unmanaged and managed solution file from your dev environment.
7. Export the unmanaged solution. Add below snippet after the last snippet.

```
    - name: export-solution action
      uses: microsoft/powerplatform-actions/export-solution@v0
      with:
        environment-url: ${{secrets.PowerPlatformDevUrl}}
        app-id: ${{secrets.PowerPlatformAppID}}
        client-secret: ${{ secrets.PowerPlatformClientSecret }}
        tenant-id: ${{secrets.PowerPlatformTenantID}}
        solution-name: ${{ github.event.inputs.solution_name }}
        solution-output-file: ${{
github.event.inputs.solution_exported_folder}}/${{
github.event.inputs.solution_name }}.zip
```

8. Export the managed solution. Add below snippet after the last snippet.

```
    - name: export-managed-solution action
      uses: microsoft/powerplatform-actions/export-solution@v0
      with:
        environment-url: ${{secrets.PowerPlatformDevUrl}}
        app-id: ${{secrets.PowerPlatformAppID}}
        client-secret: ${{ secrets.PowerPlatformClientSecret }}
        tenant-id: ${{secrets.PowerPlatformTenantID}}
        solution-name: ${{ github.event.inputs.solution_name }}
        solution-output-file: ${{
github.event.inputs.solution_exported_folder}}/${{
github.event.inputs.solution_name }}_managed.zip
        managed: true
```

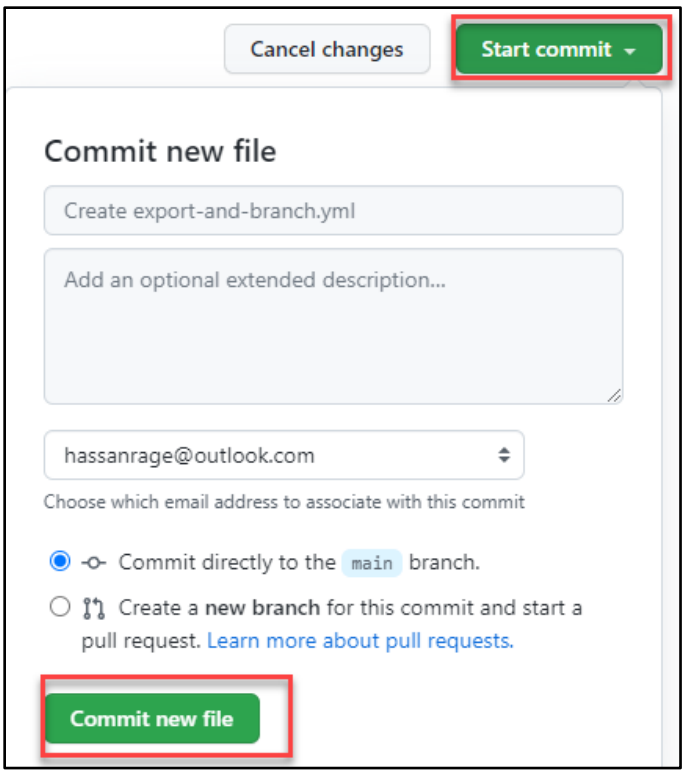
9. The solution files are compressed files and don't version control well. Using unpack you will expand the solution files into a set of files that can be easily checked into the repo.
10. Unpack solution file. Add below snippet after the last snippet.

```
    - name: unpack-solution action
      uses: microsoft/powerplatform-actions/unpack-solution@v0
      with:
        solution-file: ${{ github.event.inputs.solution_exported_folder}}/${{
github.event.inputs.solution_name }}.zip
        solution-folder: ${{ github.event.inputs.solution_folder}}/${{
github.event.inputs.solution_name }}
        solution-type: 'Both'
```

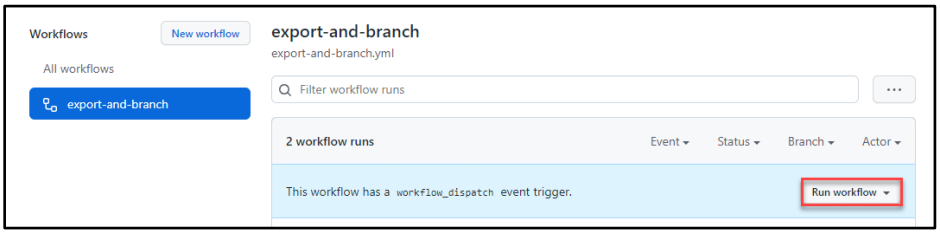
11. Branch and prepare for pull. Add below snippet after the last snippet.

```
    - name: branch-solution, prepare it for a PullRequest
      uses: microsoft/powerplatform-actions/branch-solution@v0
      with:
        solution-folder: ${{ github.event.inputs.solution_folder}}/${{
github.event.inputs.solution_name }}
        solution-target-folder: ${{
github.event.inputs.solution_target_folder}}/${{
github.event.inputs.solution_name }}
        repo-token: ${{ secrets.GITHUB_TOKEN }}
        allow-empty-commit: true
```

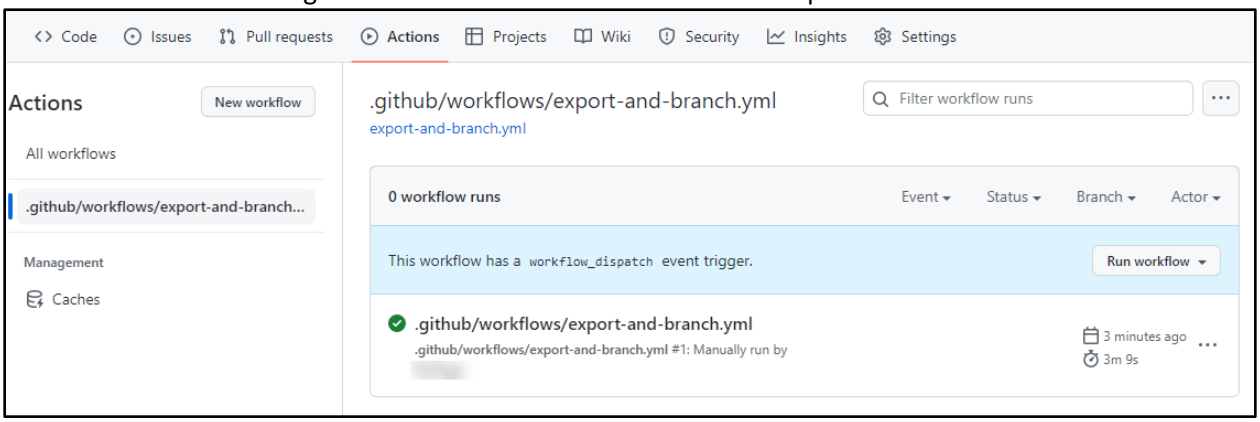
12. Click **Start commit** and then click **Commit new file**.



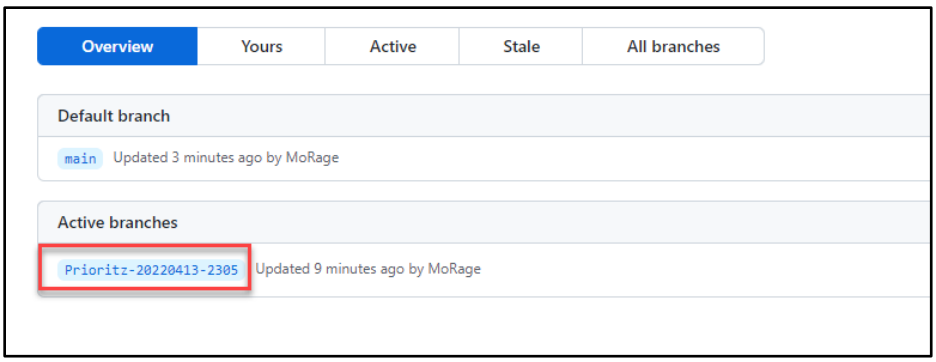
13. Select the **Actions** tab and select the workflow you created.
14. Click **Run workflow.**



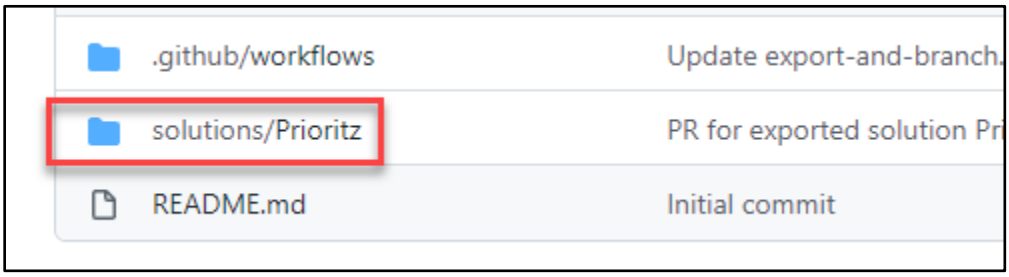
15. Click **Run workflow** again and wait for the workflow run to complete.



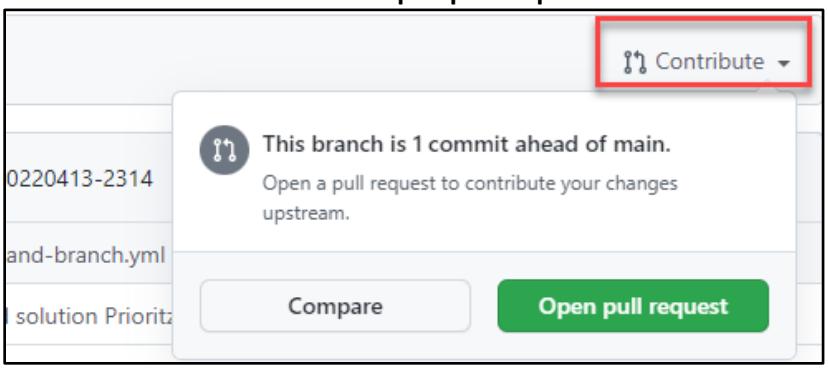
16. Select the **Code** tab.
17. Select **Branches**. You should see two branches.
18. Click to open the branch that was created by the workflow action.



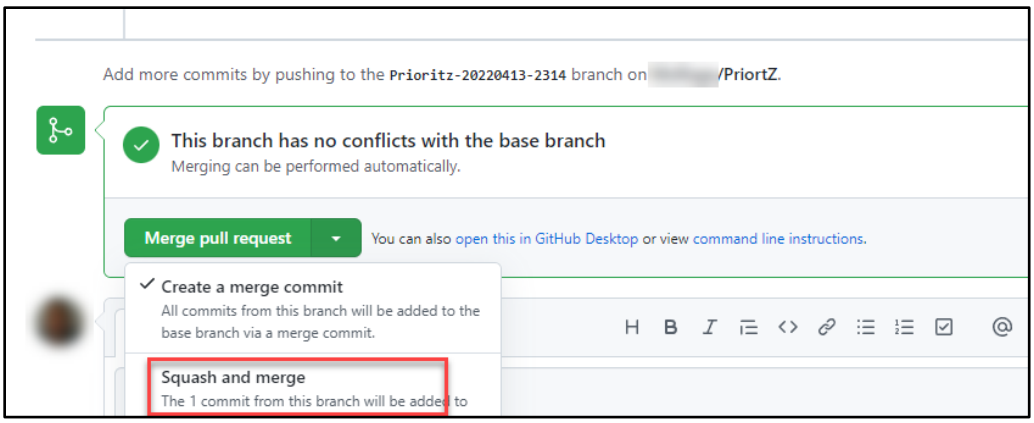
19. You should see solution folder.



20. Click **Contribute** and select **Open pull request**.



21. Add description if you like and then click **Create pull request**.
22. You should now see the pull request summary. Confirm that the branch has no conflicts with the main branch and that the changes can be merged into the main branch automatically.
23. Click on the chevron button next to the **Merge pull request** button and select **Squash and merge**.



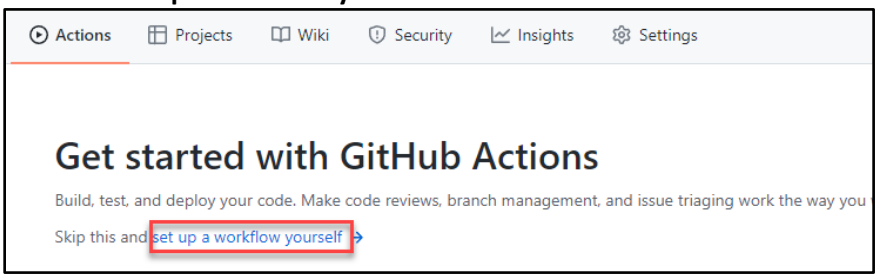
24. Click **Squash and merge**.
25. Click **Confirm squash and merge**.
26. The pull request should get merged successfully.
27. Do not navigate away from this page.

Exercise 5 – Release to Test

In this exercise, you will create a workflow action and add steps that will release the solution you exported to the test environment.

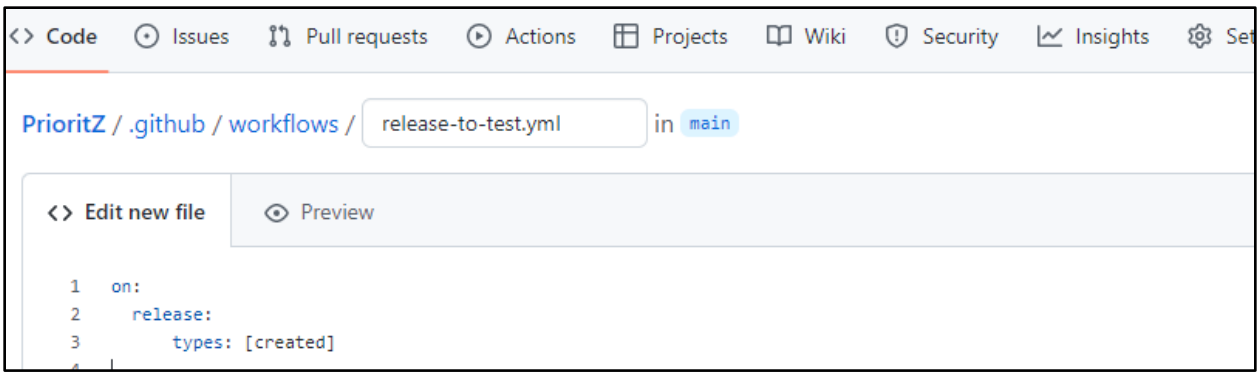
Task 1: Create workflow

1. Select the **Actions** tab.
2. Click **New workflow**.
3. Click **set up a workflow yourself**.



4. Change the file name to it **release-to-test.yml**.
5. Add the following trigger. This will trigger on creation of a new release.

```
on:
  release:
      types: [created]
```



6. Define constants. Add the below YAML snippet.

```
env:
  solution_name: Prioritz
  solution_source_folder: solutions
  solution_outbound_folder: out/solutions
  solution_release_folder: out/release
```

7. Add job and steps. Add the below YAML snippet.

```
jobs:
  convert-to-managed:
   runs-on: windows-latest

   steps:
   - uses: actions/checkout@v2
     with:
       lfs: true
```

8. Package managed solution. Add the below YAML snippet. This will take the individual files and put them in a compressed file that can be deployed.

```
   - name: Pack managed solution
     uses: microsoft/powerplatform-actions/pack-solution@v0
     with:
       solution-folder: ${{ env.solution_source_folder}}/${{ env.solution_name
}}
       solution-file: ${{ env.solution_outbound_folder}}/${{ env.solution_name
}}_managed.zip
       solution-type: Managed
```

9. Package unmanaged solution. Add the below YAML snippet.

```
   - name: Pack unmanaged solution
```

```
    uses: microsoft/powerplatform-actions/pack-solution@v0
    with:
      solution-folder: ${{ env.solution_source_folder}}/${{ env.solution_name
}}
      solution-file: ${{ env.solution_outbound_folder}}/${{ env.solution_name
}}_unmanaged.zip
      solution-type: Unmanaged
```

Edit new file Preview

```
17      steps:
18      - uses: actions/checkout@v2
19        with:
20          lfs: true
21
22      - name: Pack managed solution
23        uses: microsoft/powerplatform-actions/pack-solution@0.4.0
24        with:
25          solution-folder: ${{ env.solution_source_folder}}/${{ env.solution_name }}
26          solution-file: ${{ env.solution_outbound_folder}}/${{ env.solution_name }}_managed.zip
27          solution-type: Managed
28
29      - name: Pack unmanaged solution
30        uses: microsoft/powerplatform-actions/pack-solution@0.4.0
31        with:
32          solution-folder: ${{ env.solution_source_folder}}/${{ env.solution_name }}
33          solution-file: ${{ env.solution_outbound_folder}}/${{ env.solution_name }}_unmanaged.zip
34          solution-type: Unmanaged
35
```

```
14
15      steps:
16      - uses: actions/checkout@v2
17        with:
18          lfs: true
19
20      - name: Pack managed solution
21        uses: microsoft/powerplatform-actions/pack-solution@v0
22        with:
23          solution-folder: ${{ env.solution_source_folder}}/${{ env.solution_name }}
24          solution-file: ${{ env.solution_outbound_folder}}/${{ env.solution_name }}_managed.zip
25          solution-type: Managed
26
27      - name: Pack unmanaged solution
28        uses: microsoft/powerplatform-actions/pack-solution@v0
29        with:
30          solution-folder: ${{ env.solution_source_folder}}/${{ env.solution_name }}
31          solution-file: ${{ env.solution_outbound_folder}}/${{ env.solution_name }}_unmanaged.zip
32          solution-type: Unmanaged
33
```

10. Upload solution artifacts. Add the below YAML snippet.

```
  - name: Upload the unmanaged solution to GH artifact store
    uses: actions/upload-artifact@v2
    with:
      name: unmanagedSolutions
      path: ${{ env.solution_outbound_folder}}/${{ env.solution_name
}}_unmanaged.zip
```

```
  - name: Upload the managed solution to GH artifact store
    uses: actions/upload-artifact@v2
    with:
      name: managedSolutions
      path: ${{ env.solution_outbound_folder}}/${{ env.solution_name
}}_managed.zip
```

11. Release to staging. Add the below YAML snippet. This defines a second job to deploy.

```
  release-to-staging:
    needs: [ convert-to-managed ]
    runs-on: windows-latest

    steps:
    - uses: actions/checkout@v2
      with:
        lfs: true
```

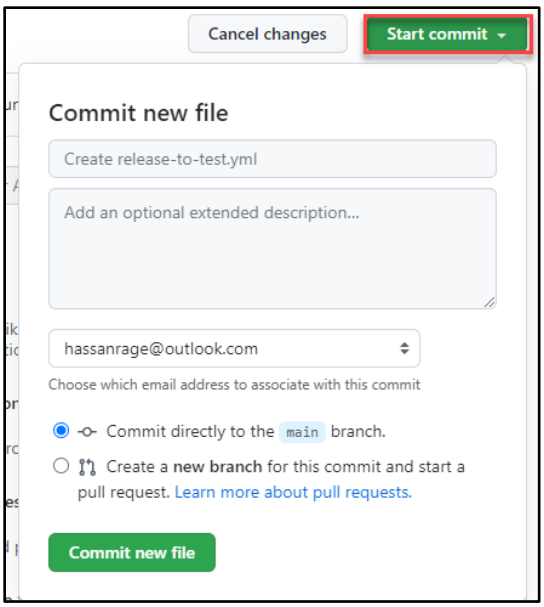
12. Download artifacts. Add the below YAML snippet.

```
    - name: Fetch the ready to ship solution from GH artifact store
      uses: actions/download-artifact@v2
      with:
        name: managedSolutions
        path: ${{ env.solution_release_folder}}
```

13. Import the managed solution to the test environment. Add the below YANL snippet.

```
    - name: Import solution to prod env
      uses: microsoft/powerplatform-actions/import-solution@v0
      with:
        environment-url: ${{secrets.PowerPlatformTestUrl}}
        app-id: ${{secrets.PowerPlatformAppID}}
        client-secret: ${{ secrets.PowerPlatformClientSecret }}
        tenant-id: ${{secrets.PowerPlatformTenantID}}
        solution-file: ${{ env.solution_release_folder}}/${{ env.solution_name
}}_managed.zip
        run-asynchronously: true
```

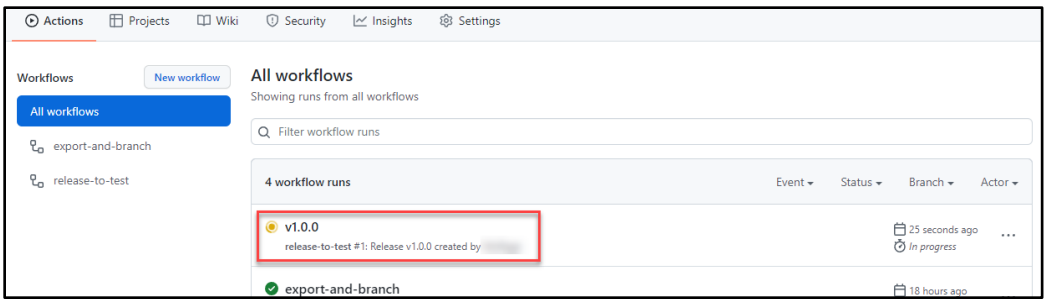
14. Click **Start commit** and then click **Commit new file**.



15. Select the **Code** tab.
16. Go to the **Releases** section and click **Create new release**.

17. Click on the **Choose a tag** button, enter **v1.0.0**, and select **+ Create new tag on publish**.

18. Click **Publish release**.
19. Select the **Actions** tab and monitor the workflow.



20. The release should complete successfully.
21. Check your test environment and you should see the solution deployed.