https://az400-webapp-aran.azurewebsites.net/

Implementing GitHub Actions for CI/CD

Lab overview

In this lab, you'll learn how to implement a GitHub Action workflow that deploys an Azure web app.

Objectives

After you complete this lab, you will be able to:

* Implement a GitHub Action workflow for CI/CD.
* Explain the basic characteristics of GitHub Action workflows.

Exercise 0: Import eShopOnWeb to your GitHub Repository

In this exercise, you will import the existing [**https://github.com/MicrosoftLearning/eShopOnWeb**](urn:gd:lg:a:send-vm-keys) repository code to your own GitHub private repo.

The repository is organized the following way: - **.ado** folder contains Azure DevOps YAML pipelines - **.devcontainer** folder container setup to develop using containers (either locally in VS Code or GitHub Codespaces) - **.azure** folder contains Bicep&ARM infrastructure as code templates used in some lab scenarios. - **.github** folder container YAML GitHub workflow definitions. - **src** folder contains the .NET 6 website used on the lab scenarios.

Task 1: Create a public repository in GitHub and import eShopOnWeb

In this task, you will create an empty public GitHub repository and import the existing [**https://github.com/MicrosoftLearning/eShopOnWeb**](urn:gd:lg:a:send-vm-keys)/'. repository.

1. From the lab computer, start a web browser, navigate to the [**https://github.com/**](urn:gd:lg:a:send-vm-keys), sign in using your account and click on **New** to create new repository.
2. On the **Create a new repository** page, click on **Import a repository** link (below the page title).

**Note**: you can also open the import website directly at [**https://github.com/new/import**](urn:gd:lg:a:send-vm-keys)

1. On the **Import your project to GitHub** page:

| **Field** | **Value** |
| --- | --- |
| Your old repository’s clone URL | [**https://github.com/MicrosoftLearning/eShopOnWeb**](urn:gd:lg:a:send-vm-keys) |
| Owner | Your account alias |
| Repository Name | eShopOnWeb |
| Privacy | **Public** |

1. Click on **Begin Import** and wait for your repository to be ready.
2. On the repository page, go to **Settings**, click on **Actions > General** and choose the option **Allow all actions and reusable workflows**. Click on **Save**.

Exercise 1: Setup your GitHub Repository and Azure access

In this exercise, you will create an Azure Service Principal to authorize GitHub accessing your Azure subscription from GitHub Actions. You will also setup the GitHub workflow that will build, test and deploy your website to Azure.

Task 1: Create an Azure Service Principal and save it as GitHub secret

In this task, you will create the Azure Service Principal used by GitHub to deploy the desired resources. As an alternative, you could also use **[OpenID connect in Azure](https://docs.github.com/actions/deployment/security-hardening-your-deployments/configuring-openid-connect-in-azure)**, as a secretless authentication mechanism.

1. On your lab computer, in a browser window, open the Azure Portal [**https://portal.azure.com/**](urn:gd:lg:a:send-vm-keys) and sign in using [**aria\_ansariHJU8J@gdcs0.com**](urn:gd:lg:a:send-vm-keys) and password [**yirVsptNGQS9IXux**](urn:gd:lg:a:send-vm-keys).
2. In the portal, look for **Resource Groups** and click on it.
3. Click on **+ Create** to create a new Resource Group for the exercise.
4. On the **Create a resource group** tab, give the following name to your Resource Group: **rg-az400-eshoponweb-NAME** (replace NAME for some unique alias). Click on **Review+Create > Create**.\*\*\*\*
5. In the Azure Portal, open the **Cloud Shell** (next to the search bar).

**Note**: if this is the first time you open the Cloud Shell, you need to configure the [**persistent storage**](https://learn.microsoft.com/en-us/azure/cloud-shell/persisting-shell-storage#create-new-storage)

1. Make sure the terminal is running in **Bash** mode and execute the following command, replacing **SUBSCRIPTION-ID** and **RESOURCE-GROUP** with your own identifiers (both can be found on the **Overview** page of the Resorce Group):
2. az ad sp create-for-rbac --name GH-Action-eshoponweb --role contributor --scopes /subscriptions/SUBSCRIPTION-ID/resourceGroups/RESOURCE-GROUP --sdk-auth

**Note**: this command will create a Service Principal with Contributor access to the Resource Group created before. This way we make sure GitHub Actions will only have the permissions needed to interact only with this Resource Group (not the rest of the subscription)

1. The command will output a JSON object, you will later keep it as a GitHub secret for the workflow, copy it. The JSON contains the identifiers used to authenticate against Azure in the name of an Azure AD application identity (service principal).
2. {
3. "clientId": "<GUID>",
4. "clientSecret": "<GUID>",
5. "subscriptionId": "<GUID>",
6. "tenantId": "<GUID>",
7. (...)
8. }
9. In a browser window, go back to your **eShopOnWeb** GitHub repository.
10. On the repository page, go to **Settings**, click on **Secrets > Actions**. Click on **New repository secret**
    * Name : **AZURE\_CREDENTIALS**
    * Secret: **paste the previously copied JSON object**
11. Click on **Add secret**. Now GitHub Actions will be able to reference the service principal, using the repository secret.

Task 2: Modify and execute the GitHub workflow

In this task, you will modify the given GitHub workflow and execute it to deploy the solution in your own subscription.

1. In a browser window, go back to your **eShopOnWeb** GitHub repository.
2. On the repository page, go to **Code** and open the following file: **eShopOnWeb/.github/workflows/eshoponweb-cicd.yml**. This workflow defines the CI/CD process for the given .NET 6 website code.
3. Uncomment the **on** section (delete "#"). The workflow triggers with every push to the main branch and also offers manual triggering ("workflow\_dispatch").
4. In the **env** section, make the following changes:
   * Replace **NAME** in **RESOURCE-GROUP** variable. It should be the same resource group created in previous steps.
   * eastus
   * Replace **YOUR-SUBS-ID** in **SUBSCRIPTION-ID**.
   * Replace **NAME** in **WEBAPP-NAME** with some unique alias. It will be used to create a globally unique website using Azure App Service.
5. Read the workflow carefully, comments are provided to help understand.
6. Click on **Start Commit** and **Commit Changes** leaving defaults (changing the main branch). The workflow will get automatically executed.

Task 3: Review GitHub Workflow execution

In this task, you will review the GitHub workflow execution:

1. In a browser window, go back to your **eShopOnWeb** GitHub repository.
2. On the repository page, go to **Actions**, you will see the workflow setup before executing. Click on it.
3. Wait for the workflow to finish. From the **Summary** you can see the two workflow jobs, the status and Artifacts retained from the execution. You can click in each job to review logs.
4. In a browser window, go back to the Azure Portal (https://portal.azure.com/). Open the resource group created before. You will see that the GitHub Action, using a bicep template, has created an Azure App Service Plan + App Service. You can see the published website opening the App Service and clicking **Browse**.

(OPTIONAL) Task 4: Add manual approval pre-deploy using GitHub Environments

In this task, you will use GitHub environments to ask for manual approval before executing the actions defined on the deploy job of your workflow.

1. On the repository page, go to **Code** and open the following file: **eShopOnWeb/.github/workflows/eshoponweb-cicd.yml**.
2. In the **deploy** job section, you can find a reference to an **enviroment** called **Development**.
3. On the repository page, go to **Settings**, open **Environments** and click **New environment**.
4. Give it **Development** name and click on **Configure Environment**.
5. In the **Configure Development** tab, check the option **Required Reviewers** and your GitHub account as a reviewer. Click on **Save protection rules**.
6. Now lets test the protection rule. On the repository page, go to **Actions**, click on **eShopOnWeb Build and Test** workflow and click on **Run workflow>Run workflow** to execute manually.
7. Click on the started execution of the workflow and wait for **buildandtest** job to finish. You will see a review request when **deploy** job is reached.
8. Click on **Review deployments**, check **Development** and click on **Approve and deploy**.
9. Workflow will follow the **deploy** job execution and finish.

**Review**

In this lab, you implemented a GitHub Action workflow that deploys an Azure Web App.