Integrating External Source Control with Azure Pipelines

Lab requirements

- **Set up an Azure DevOps organization:** If you don't already have an Azure DevOps organization that you can use for this lab, create one by following the instructions available at Create an organization or project collection.
- If you don't already have a GitHub account that you can use for this lab, follow instructions available at Signing up for a new GitHub account.

Lab overview

With the introduction of Azure DevOps, Microsoft offers developers a new continuous integration/continuous delivery (CI/CD) service called Azure Pipelines that enables you to build continuously, test, and deploy to any platform or cloud. It has cloud-hosted agents for Linux, macOS, and Windows; powerful workflows with native container support; and flexible deployments to Kubernetes, VMs, and serverless environments.

Azure Pipelines provides unlimited CI/CD minutes and 10 parallel jobs to every GitHub open source project for free. All open source projects run on the same infrastructure that our paying customers to use. That means you'll have the same fast performance and high quality of service. Many of the top open source projects are already using Azure Pipelines for CI/CD, such as Atom, CPython, Pipenv, Tox, Visual Studio Code, and TypeScript-and the list is growing every day.

Objectives

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

- Install Azure Pipelines from the GitHub Marketplace.
- Integrate a GitHub project with an Azure Pipeline.
- Track pull requests through the pipeline.

Instructions

Exercise 1: Getting started with Azure Pipelines

In this exercise, you will integrate a GitHub project with Azure DevOps by using the new Azure Pipelines integration from the Marketplace.

Task 1: Forking a GitHub repo and installing Azure Pipelines

In this task, you will fork a GitHub repo and install Azure Pipelines in your GitHub account.

1. On your lab computer, start a web browser, navigate to the GitHub actionsdemos/calculator site and, if you're not already signed into GitHub, sign in now.
2. On the **actionsdemos/calculator site** page, click **Fork**, to fork the repository to your own GitHub account. If prompted, select the account to fork the repository into.
3. On the page displaying the forked repo, at the top menu, click **Marketplace**.

   > **Note**: The **GitHub Marketplace** provides a variety of tools from Microsoft and 3rd parties that help you extend your project workflows.

4. In the **Search for apps and actions**, type **Azure Pipelines**, press the **Enter** key, and, in the list of results, click **Azure Pipelines**.
5. On the **Azure Pipelines** page, click **Read more** and read through the benefits of Azure Pipelines.

   > **Note**: The Azure Pipelines offering is free for anyone to use for public repositories, and free for a single build queue if you're using a private repository.

6. On the **Azure Pipelines** page, click **Install it for free**. If you have multiple **GitHub** accounts, select the one you forked the calculator to from the **Switch billing account** dropdown.
7. On the **Review your order** page, click **Complete order and begin installation**.
8. On the **Install Azure Pipelines** page, use the default option **All repositories** and click **Install**.
9. If prompted, authenticate with your GitHub password to continue.
10. When prompted, on the **Setup your Azure Pipelines project** page, firstly, click **Switch Directories** and ensure 'Default Directory' is selected. Then, in

the **Select your Azure DevOps organization** dropdown list, select your Azure DevOps account and click **Create a new project**.

11. When prompted, on the **Setup your Azure Pipelines project** page, in the **Project name** textbox, type **Integrating External Source Control with Azure Pipelines**, leave the **Project visibility** set to **Private**, and click **Continue**.
12. On the **Azure Pipelines by Microsoft would like permission to** page, click **Authorize Azure Pipelines**.

Task 2: Configuring your Azure Pipelines project

In this task, you will configure the Azure Pipelines project based on the fork of the GitHub repo you created in the previous task.

Note: You are now on the Azure DevOps site and need to set up your Azure Pipelines project.

1. In the Azure DevOps portal, click on **Pipelines** in the navigation panel on the left.
2. Click on the **Create pipeline** button at top right.
3. On the **Select a repository** pane of the **Pipelines** view in the Azure DevOps portal, select the fork of the GitHub calculator repository you created in the previous task.
4. On the **Configure your pipeline**, select **Node.js**.

   Note: The build pipeline is defined as **YAML**, a markup syntax well-suited to defining processes like this because it allows you to manage the configuration of the pipeline like any other file in the repo. It's a pretty simple template that identifies the pool to pull a VM from for building, the process to install Node.js for building, and the actual build itself.

5. On the **Review your pipeline YAML**, click **Save and run** to save the pipeline and queue a new build.
6. On the **Save and run** pane, accept the default settings and click **Save and run**.
7. On the **Summary** tab of the build job's pane, verify that the build completed successfully.

Task 3: Modifying a YAML build pipeline definition

In this task, you will modify the YAML build definition in the forked GitHub repository and track the build process triggered by your modification.

Note: While the default pipeline is a great start, it doesn't do everything we would like to have automated. For example, it would be great if it also ran our tests to confirm that the changes don't create bugs. Let's return to GitHub where we can edit the YAML by hand.

1. On the **Summary** tab of the build job's pane, next to the **Repository and version** label, right-click the entry representing the GitHub project repo hosting the fork you created earlier in this lab and select **Open link in new tab**. This will open a new browser tab, displaying the GitHub page with the content of the fork.
2. On the GitHub page displaying the content of the fork, locate and click the entry representing the file **azure-pipelines.yml**. This will automatically open the file and display its content.
3. On the **master/calculator/azure-pipelines.yml** page, in the upper right corner of the pane displaying the file content, click the **Edit this file** icon in the shape of a pencil.

   > **Note**: Our project already contains tests written using Mocha so we just need to execute them in our pipeline.

4. To add the test run, add the `npm test` command directly below the `npm run build` command, with the same indentation. In addition, update the `displayName` entry to `'npm install, build, and test'` to clearly indicate what each task of the build is doing:

```
  npm test
displayName: 'npm install, build, and test'
```

5. Scroll to the bottom of the page, replace the default commit message with **Adding npm test**, and click **Commit changes**.
6. Switch back to the browser tab displaying the **Azure DevOps** portal and use the breadcrumb navigation to navigate to the **Pipelines** pane of the **Pipelines** view.
7. Verify that the new build triggered by the update already appears on the **Recent** tab in the **Recently run pipelines** list. Click the entry corresponding to the pipeline, on the **Runs** tab, select the most recent run, and, in the **Jobs** section, click the **Job** entry.
8. On the pane displaying job details, click on individual tasks of the job and follow it through to completion.

Task 4: Proposing a change via GitHub pull request

In this task, you will propose an invalid change and review the results of a build triggered by a pull request.

1. Switch back to the browser tab displaying the GitHub page displaying the content of the **azure-pipelines.yml** file, navigate back to the page listing content of the forked repo and click **Go to file**.

2. At the **calculator/** prompt, type **arithmeticController.js** and, in the list of results click **api/controllers/arithmeticController.js**. This will automatically redirect the browser session to the **master/calculator/api/controllers/arithmeticController.js** page, displaying the content of that file.
3. On the **master/calculator/api/controllers/arithmeticController.js** page, in the upper right corner of the pane displaying the file content, click the **Edit this file** icon in the shape of a pencil.
4. Change the line `'add': function(a,b) { return +a + +b },` to `'add': function(a,b) { return a + b },`.

   Note: This is an incorrect change that would result in invalid outcome.

5. Scroll to the bottom of the page, replace the default commit message with **Modifying the add function**, select **Create a new branch**, set its name to **addition-cleanup**, and click **Propose file change**.
6. On the **Open a pull request** page, click **Create pull request** to initiate the process of getting your untested changes into production code.

   Note: Azure DevOps will detect the change and start the build pipeline. This will result in failed checks, triggering an update the GitHub UI.

   Note: Return to your original mindset of "project owner".

7. On the **Modifying the add function #1** pull request page, in the **All checks have failed** section, click **Details** to learn more.
8. Review the **ANNOTATIONS** section and click the link **View more details on Azure Pipelines** directly below it. This will open a new browser tab displaying the failed run of the job in the Azure DevOps portal.
9. On the failed job pane in the Azure portal, click the **Job** entry to display its details.
10. In the list of job tasks, click the **npm install, build, and test** task to view its output.
11. Locate the section that lists out failing tests.
12. Close the tab displaying the failed run of the job in the Azure DevOps portal.

Task 5: Using the broken pull request to improve the project

In this task, you will correct the invalid changes introduced in the pull request created in the previous task.

1. Return to the browser tab displaying the **Modifying the add function #1** GitHub page and return to the main page listing the content of the forked repo.

2. At the top of the pane containing the listing of the repo files, **Pull requests** and then click the entry representing the most recent pull request.
3. On the **Modifying the add function #1** GitHub page, click the **File changed** tab and review its content.

   Note: It appears that the changes were made by someone who didn't realize that the plus signs before each variable were necessary to coerce those variables to their number representations. By removing them, JavaScript interpreted the middle plus sign as the string concatenation operator, which explains why 21 + 21 = 2121 in the failed test.

4. On the **Modifying the add function #1** GitHub page, click the ellipsis symbol directly under the **Review changes** button and, in the drop-down menu, click **Edit file**.
5. Revert the original changes by adding the plus signs in front of the **a** and **b** variables, resulting in `'add': function(a,b) { return +a + +b },`. In addition, include a comment on the preceding line stating `// Using + operator to type cast variables as integers in order to prevent string concatenation`.
6. Scroll to the bottom of the page, replace the default commit message with **Fixing the add function**, ensure that the option to **Commit directly to the addition-cleanup branch** is selected, and click **Commit changes**.
7. On the **Modifying the add function #1** GitHub page, select the **Conversation** tab.

   Note: Azure DevOps will again detect the change and start the build pipeline. Wait for all checks to pass.

8. Once all checks have passed, click **Merge pull request** and then click **Confirm merge**.

Task 6: Adding a build status badge

In this task, you will add a build status badge to your GitHub repo.

Note: An important sign for a quality project is its build status badge. When someone finds a project that has a badge indicating that the project is currently in a successful build state, it's a sign that the project is maintained effectively.

1. Switch back to the browser tab displaying the **Azure DevOps** portal and use the breadcrumb navigation to navigate to the **Recent** tab of the **Pipelines** pane of the **Pipelines** view.
2. On the **Recent** tab in the **Recently run pipelines** list, click the entry corresponding to the pipeline you used in this lab.
3. On the pipeline pane, click the ellipsis symbol in the upper right corner and, in the dropdown list, select **Status badge**.

> **Note**: The **Status badge** UI provides a quick and easy way to reference the build status. Often, you'll want to use the provided URLs in your own dashboards, or you can use the Markdown snippet to add the status badge to locations such as Wiki pages.

4. On the **Status badge** pane, click the **Copy to clipboard** button for **Sample Markdown**.
5. Switch back to the browser tab displaying the GitHub page displaying the content of the forked repo and, if needed, click the **<> Code** tab.
6. In the list of repo files, click **README.md** and, on the **master/calculator/README.md** page, in the upper right corner of the pane displaying the file content, click the **Edit this file** icon in the shape of a pencil.
7. Add an extra line above line 6 and paste into it the content of Clipboard.
8. Scroll to the bottom of the page, replace the default commit message with **Add an Azure Pipelines status badge**, and click **Commit changes**.