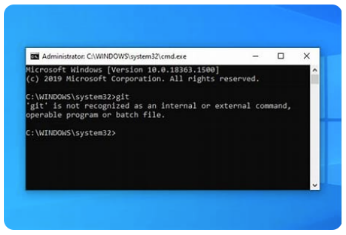
Do this lab and template it if possible – save it out of vm

1. AZ400-DesigningandImplementingMicrosoftDevOpsSolutions (microsoftlearning.github.io)
    a. Task 6: Working with Pull Requests – in text below links WIT
2. Describe working with Git locally - Training | Microsoft Learn
3. Lab for Work Item Types WIT – o2?
4. Monitor 12
5. Master TOC AZ400-DesigningandImplementingMicrosoftDevOpsSolutions (microsoftlearning.github.io)

Setup the environment – Sandbox Time – this is not complete – do things at xtreme first

Make sure you have the following items: - git account not provided

- A free GitHub: Let's build from here · GitHub account where you can create a repository.
- An Azure DevOps organization. Create one for free. If your team already has one, then make sure you're an administrator of the Azure DevOps project that you want to use.
- An ability to run pipelines on Microsoft-hosted agents. To use Microsoft-hosted agents, your Azure DevOps organization must have access to Microsoft-hosted parallel jobs. You can either purchase a parallel job or you can request a free grant.



```
Microsoft Windows [Version 10.0.18363.1500]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>git
'git' is not recognized as an internal or external command,
operable program or batch file.

C:\WINDOWS\system32>
```

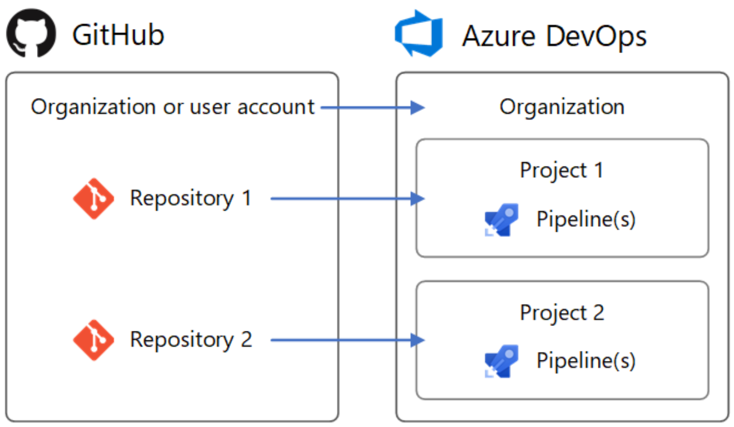
The error message "the term git is not recognized as a cmdlet" can occur when the Git path is not set correctly or the variables changes are not updated [1] [2]. To solve the problem, you can exit Command Prompt and open it again as administrator [1]. Another solution is to use the Git installation GUI to automatically create the Path variables for you [2]. If the Git module is missing or corrupt, you can use "get-module" in PowerShell to see if the module is present and correct [3]. If you are trying to run Git on PowerShell and get the error, you can re-run the Git-Bash Installation Setup to add the option to "Let git run from the command line or 3rd party tools" or add the Git path to the Path Environment Variable [4].

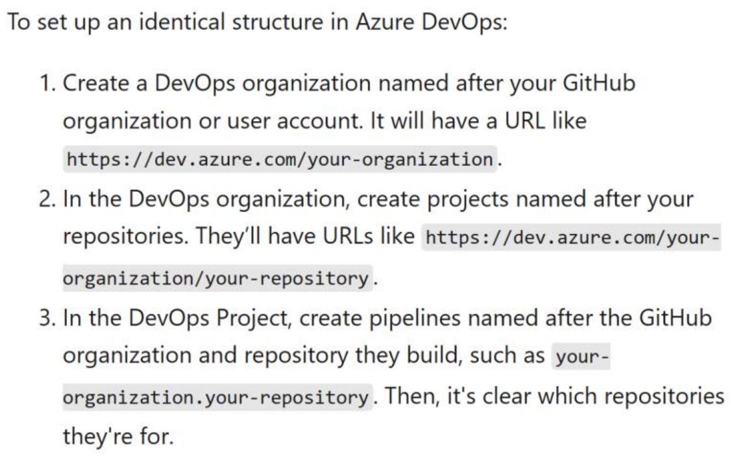
May need to download and install or run Git to fix path issues

Azure DevOps can reflect your GitHub structure with:

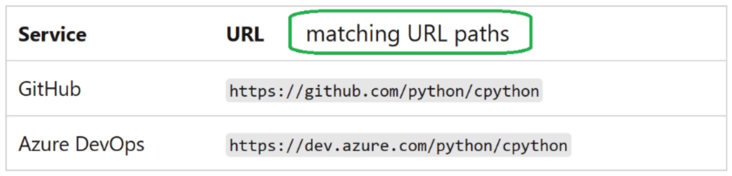
- A DevOps organization for your GitHub organization or user account
- DevOps Projects for your GitHub repositories
- A Git repository is the . git/ folder inside a project.

Plan your organizational structure - Azure DevOps | Microsoft Learn




To set up an identical structure in Azure DevOps:

1. Create a DevOps organization named after your GitHub organization or user account. It will have a URL like `https://dev.azure.com/your-organization`.
2. In the DevOps organization, create projects named after your repositories. They'll have URLs like `https://dev.azure.com/your-organization/your-repository`.
3. In the DevOps Project, create pipelines named after the GitHub organization and repository they build, such as `your-organization.your-repository`. Then, it's clear which repositories they're for.



Service	**URL** matching URL paths
GitHub	`https://github.com/python/cpython`
Azure DevOps	`https://dev.azure.com/python/cpython`

Kanban (Azure) Boards

1. About Kanban boards - Azure Boards | Microsoft Learn
2. Visual interactive space for you and your team to plan and show progress from idea to completion.
3. Each column represents a work stage.
4. Each card represents a work item and supports quick status updates through drag-and-drop.
5. What is Azure Boards? Tools to manage software development projects. - Azure Boards | Microsoft Learn
6. Track progress and individual items on the Taskboard - Azure Boards | Microsoft Learn
    a. In your daily Scrum meetings, your team can view progress made to backlog items and tasks from the sprint Taskboard.
7. Add tasks or other child work items to checklists - Azure Boards | Microsoft Learn

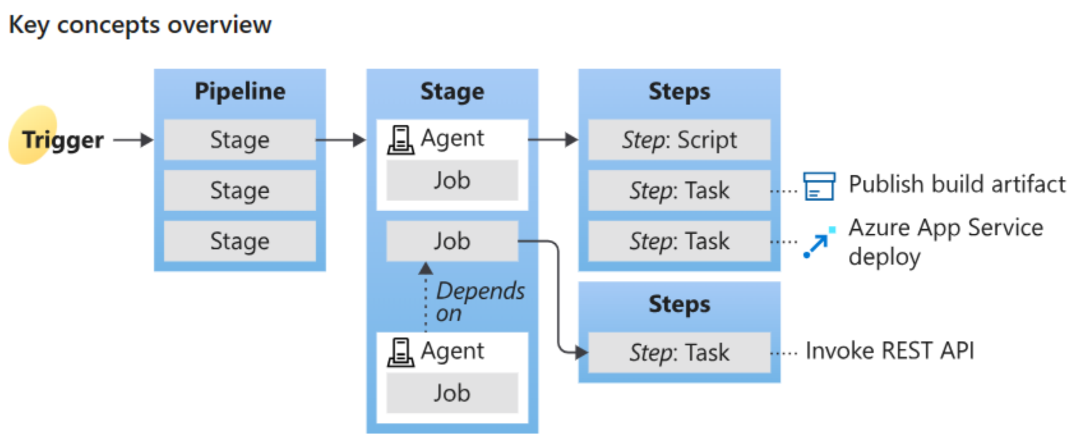
Yaml

1. Describe the anatomy of a pipeline - Training | Microsoft Learn

Extra detail

2. azure-pipelines-yaml/templates at master · microsoft/azure-pipelines-yaml · GitHub
3. YAML pipeline editor guide - Azure Pipelines | Microsoft Learn
    a. Use Task Assistant
4. Azure DevOps Release Notes - Preview fully parsed YAML document without running the pipeline | Microsoft Learn
5. Reduce the number of email notifications from pull requests, you can now create a custom notification subscription for pull requests that are created or updated in **draft state**.
6. steps.download definition | Microsoft Learn
7. Downloads artifacts associated with the current run or from another Azure Pipeline that is associated as a pipeline resource.

Hello Pipeline



- A trigger tells a Pipeline to run.
- A pipeline is made up of one or more stages.
- A pipeline can deploy to one or more environments.
- A stage is a way of organizing jobs in a pipeline
- Each stage can have one or more jobs.
- Can be used to mark separation of concerns (for example, Build, QA, and production)
- Each job runs on one agent. A job can also be agentless.
- Each agent runs a job that contains one or more steps.
- **The step can be a task or script and is the smallest building block of a pipeline.
- A task is a pre-packaged script that performs an action, such as invoking a REST API or publishing a build artifact.

- An artifact is a collection of files or packages published by a run.

Azure Pipelines New User Guide - Key concepts - Azure Pipelines | Microsoft Learn

- A pipeline is a workflow that defines how your test, build, and deployment steps are run, made up of one or more Stages.
- A Run represents one execution of a pipeline. It collects the logs associated with running the steps and the results of running tests.

What is Azure Pipelines? - Azure Pipelines | Microsoft Learn

- Azure Pipelines automatically builds and tests code projects. It supports all major languages and project types and combines continuous integration, continuous delivery, and continuous testing to build, test, and deliver your code to any destination.
- In Azure Pipelines jobs run on the host machine where the agent is installed using a Microsoft-hosted agent.
- Each time you run a pipeline; you get a fresh virtual machine for each job in the pipeline. The virtual machine is discarded after one job.
- For more control over the context where your tasks run. YAML pipelines offer container jobs.

List of Items

1. Create your first pipeline - Azure Pipelines | Microsoft Learn
   a. Jobs in Azure Pipelines and TFS - Azure Pipelines | Microsoft Learn
   b. Container Jobs in Azure Pipelines and TFS - Azure Pipelines | Microsoft Learn
   c. Make changes to your pipeline, select it in Pipelines and Edit azure-pipelines.yml file.
2. Build GitHub repositories - Azure Pipelines | Microsoft Learn
   a. Configure the integration between GitHub and Azure Pipelines.
   b. Azure Pipelines can automatically build and validate every pull request and commit to your GitHub repository.
3. Tutorial: Create and run your first GitLab CI/CD pipeline | GitLab
4. Tutorial: Create a CI/CD pipeline for your existing code by using Azure DevOps Starter | Microsoft Learn

Create workflow files, trigger workflows, and find workflow logs.

1. Deployment jobs - Azure Pipelines | Microsoft Learn
   a. A deployment job is a collection of steps that are run sequentially against the environment.
   b. Yaml syntax - lifecycle hooks
   c. `runOnce` is the simplest deployment strategy wherein all the lifecycle hooks, namely `preDeploy` `deploy`, `routeTraffic`, and `postRouteTraffic`, are executed once. Then, either `on: success` or `on: failure` is executed.

Other "Containers" to help or hurt your management style and more Yaml

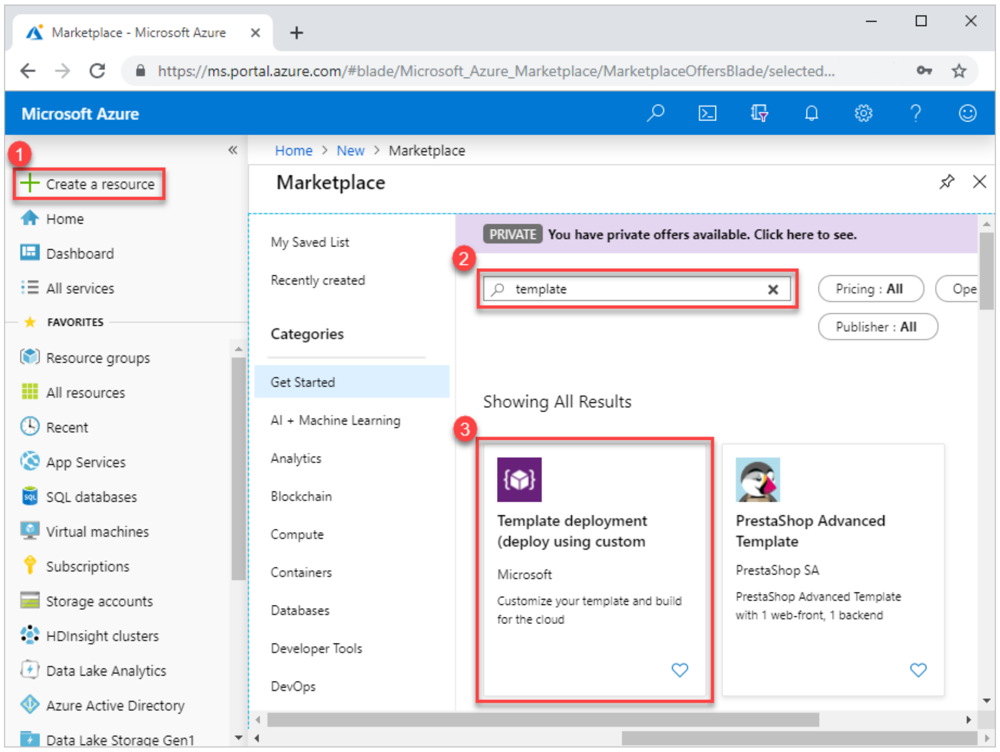
2. Create target environment - Azure Pipelines | Microsoft Learn

   a. An environment is a collection of resources that you can target with deployments from a pipeline.
   b. Typical examples of environment names are Dev, Test, QA, Staging, and Production.
   c. Kubernetes resource and virtual machine resource types are currently supported.
3. Artifacts in Azure Pipelines - Azure Pipelines | Microsoft Learn
   a. In conjunction with Azure Pipelines to deploy packages, publish build artifacts, or integrate files between your pipeline stages to build, test, or deploy your application.
   b. Publish and download build artifacts - Azure Pipelines | Microsoft Learn – Yaml
   c. Continuous integration systems produce deployable artifacts, which include infrastructure and apps.
   d. Used by the continuous delivery release pipelines to drive automatic deployments, new versions and fixes to existing systems.
4. Provision agents for deployment groups - Azure Pipelines | Microsoft Learn
   a. Deployment groups make it easy to define logical groups of target machines for deployment, and install the required agent on each machine.
5. Library for Azure Pipelines - Azure Pipelines | Microsoft Learn
   a. A collection of build and release assets for an Azure DevOps project that can be used in multiple build and release pipelines of the project.
6. space

eShopOnWeb and Parts Unlimited

ARM Templates

1. Deploy resources with Azure portal - Azure Resource Manager | Microsoft Learn
2. Export template in Azure portal - Azure Resource Manager | Microsoft Learn
3. To deploy a customized template through the portal, select Create a resource, search for template. and then select Template deployment.
4. Select Create.

5. 

Sort of beginning from repo forward…

Required for labs

- Create an organization or project collection.
- Git for Windows download page download it, and install it
- Visual Studio Code download page
- C# extension installation page

01 lab Working with Git

AZ400-DesigningandImplementingMicrosoftDevOpsSolutions (microsoftlearning.github.io)

- Clone an existing repository.
- Save work with commits.
- Review history of changes.
- Work with branches by using Visual Studio Code.

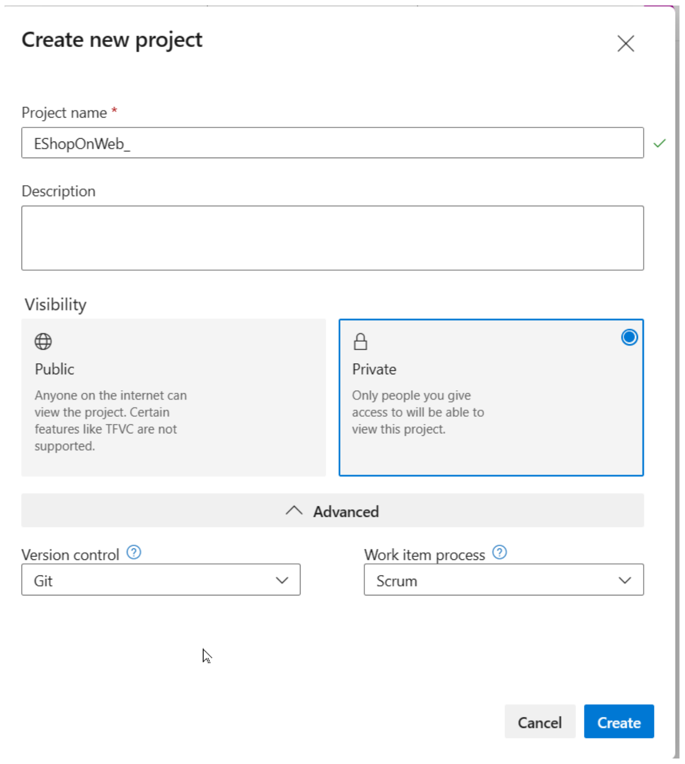
Exercise 0: Configure the lab prerequisites

In this exercise, you will set up the prerequisites for the lab, which consist of a new Azure DevOps project with a repository based on the eShopOnWeb.

Task 1: (skip if done) Create and configure the team project

In this task, you will create an **eShopOnWeb** Azure DevOps project to be used by several labs.

1. On your lab computer, in a browser window open your Azure DevOps organization. Click on **New Project**. Give your project the name **eShopOnWeb** and choose **Scrum** on the **Work Item process** dropdown. Click on **Create**.



Create new project

Project name *: EShopOnWeb_

Description:

Visibility

Public: Anyone on the internet can view the project. Certain features like TFVC are not supported.

[✓ Private: Only people you give access to will be able to view this project.]

Advanced

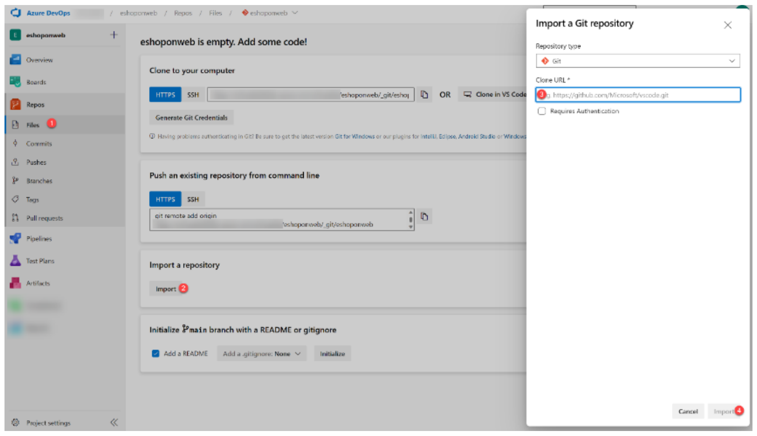
Version control: Git

Work item process: Scrum

Task 2: (skip if done) Import eShopOnWeb Git Repository

In this task you will import the eShopOnWeb Git repository that will be used by several labs.

1. On your lab computer, in a browser window open your Azure DevOps organization and the previously created **eShopOnWeb** project. Click on **Repos>Files** , **Import**. On the **Import a Git Repository** window, paste the following URL https://github.com/MicrosoftLearning/eShopOnWeb.git and click on **Import**:



2. 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 3: Configure Git and Visual Studio Code

In this task, you will install and configure Git and Visual Studio Code, including configuring the Git credential helper to securely store the Git credentials used to communicate with Azure DevOps. If you have already implemented these prerequisites, you can proceed directly to the next task.

1. On the lab computer, open **Visual Studio Code**.
2. In the Visual Studio Code interface, from the main menu, select **Terminal | New Terminal** to open the **TERMINAL** pane.
3. Make sure that the current Terminal is running **PowerShell** by checking if the drop-down list at the top right corner of the **TERMINAL** pane shows **1: powershell**
   Note: To change the current Terminal shell to **PowerShell** click the drop-down list at the top right corner of the **TERMINAL** pane and click **Select Default Shell**. At the top of the Visual Studio Code window select your preferred terminal shell **Windows PowerShell** and click the plus sign on the right-hand side of the drop-down list to open a new terminal with the selected default shell.

4. In the **TERMINAL** pane, run the following command below to configure the credential helper.

   CodeCopy

```
git config --global credential.helper wincred
```

5. In the **TERMINAL** pane, run the following commands to configure a user name and email for Git commits (replace the placeholders in braces with your preferred user name and email eliminating the < and > symbols):

CodeCopy

```
git config --global user.name "<John Doe>"
git config --global user.email <johndoe@example.com>
```

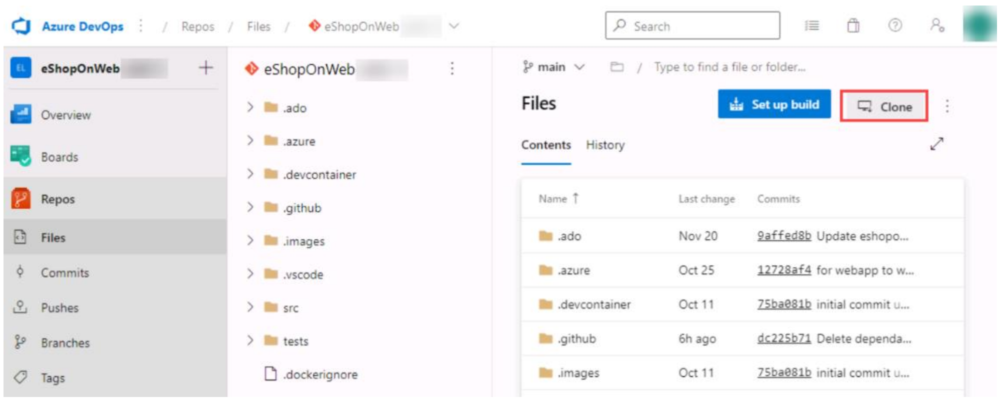
Exercise 1: Clone an existing repository

In this exercise, you use Visual Studio Code to clone the Git repository you provisioned as part of the previous exercise.

Task 1: Clone an existing repository

In this task, you will step through the process of cloning a Git repository by using Visual Studio Code.

1. Switch to the the web browser displaying your Azure DevOps organization with the **eShopOnWeb** project you generated in the previous exercise.
2. In the vertical navigational pane of the Azure DevOps portal, select the **Repos** icon.
3. In the upper right corner of the **eShopOnWeb** repository pane, click **Clone**.



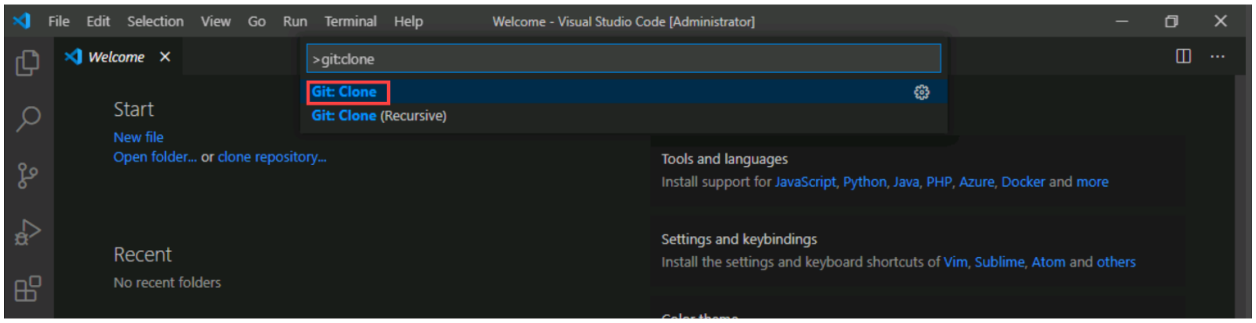
Name	Last change	Commits
.ado	Nov 20	9affed8b Update eshopo...
.azure	Oct 25	12728af4 for webapp to w...
.devcontainer	Oct 11	75ba081b initial commit u...
.github	6h ago	dc225b71 Delete dependa...
.images	Oct 11	75ba081b initial commit u...

Note: Getting a local copy of a Git repo is called *cloning*. Every mainstream development tool supports this and will be able to connect to Azure Repos to pull down the latest source to work with.

4. On the **Clone Repository** panel, with the **HTTPS** Command line option selected, click the **Copy to clipboard** button next to the repo clone URL.
   Note: You can use this URL with any Git-compatible tool to get a copy of the codebase.

5. Close the **Clone Repository** panel.
6. Switch to **Visual Studio Code** running on your lab computer.
7. Click the **View** menu header and, in the drop-down menu, click **Command Palette**.
   Note: The Command Palette provides an easy and convenient way to access a wide variety of tasks, including those implemented as 3rd party extensions. You can use the keyboard shortcut **Ctrl+Shift+P** or **F1** to open it.

8. At the Command Palette prompt, run the **Git: Clone** command.


File Edit Selection View Go Run Terminal Help
Welcome - Visual Studio Code [Administrator]
Welcome
>git:clone
Git: Clone
Git: Clone (Recursive)
Start
New file
Open folder... or clone repository...
Recent
No recent folders
Tools and languages
Install support for JavaScript, Python, Java, PHP, Azure, Docker and more
Settings and keybindings
Install the settings and keyboard shortcuts of Vim, Sublime, Atom and others

Note: To see all relevant commands, you can start by typing **Git**.

9. In the **Provide repository URL or pick a repository source** text box, paste the repo clone URL you copied earlier in this task and press the **Enter** key.
10. Within the **Select Folder** dialog box, navigate to the C: drive, create a new folder named **Git**, select it, and then click **Select Repository Location**.
11. When prompted, log in to your Azure DevOps account.
12. After the cloning process completes, once prompted, in the Visual Studio Code, click **Open** to open the cloned repository.
    Note: You can ignore warnings you might receive regarding problems with loading of the project. The solution may not be in the state suitable for a build, but we're going to focus on working with Git, so building the project is not required.

Exercise 2: Save work with commits

In this exercise, you will step through several scenarios that involve the use of Visual Studio Code to stage and commit changes.

When you make changes to your files, Git will record the changes in the local repository. You can select the changes that you want to commit by staging them. Commits are always made against your local Git repository, so you don't have to worry about the commit being perfect or ready to share with others. You can make more commits as you continue to work and push the changes to others when they are ready to be shared.

Git commits consists of the following:

- The file(s) changed in the commit. Git keeps the contents of all file changes in your repo in the commits. This keeps it fast and allows intelligent merging.
- A reference to the parent commit(s). Git manages your code history using these references.
- A message describing a commit. You give this message to Git when you create the commit. It's a good idea to keep this message descriptive, but to the point.

Task 1: Commit changes

In this task, you will use Visual Studio Code to commit changes.

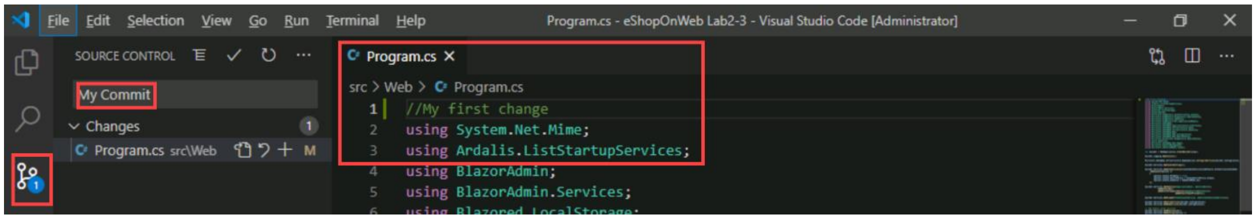
1. In the Visual Studio Code window, at the top of the vertical toolbar, select the **EXPLORER** tab, navigate to the **/eShopOnWeb/src/Web/Program.cs** file and select it. This will automatically display its content in the details pane.
2. On the first line add the following comment:

C#Copy

```
// My first change
```

Note: It doesn't really matter what the comment is since the goal is just to make a change.

3. Press **Ctrl+S** to save the change.
4. In the Visual Studio Code window, select the **SOURCE CONTROL** tab to verify that Git recognized the latest change to the file residing in the local clone of the Git repository.
5. With the **SOURCE CONTROL** tab selected, at the top of the pane, in the textbox, type **My commit** as the commit message and press **Ctrl+Enter** to commit it locally.


File Edit Selection View Go Run Terminal Help
Program.cs - eShopOnWeb Lab2-3 - Visual Studio Code [Administrator]
SOURCE CONTROL
My Commit
Changes 1
Program.cs src\Web M
Program.cs
src > Web > Program.cs

```
//My first change
using System.Net.Mime;
using Ardalis.ListStartupServices;
using BlazorAdmin;
using BlazorAdmin.Services;
using Blazored.LocalStorage;
```

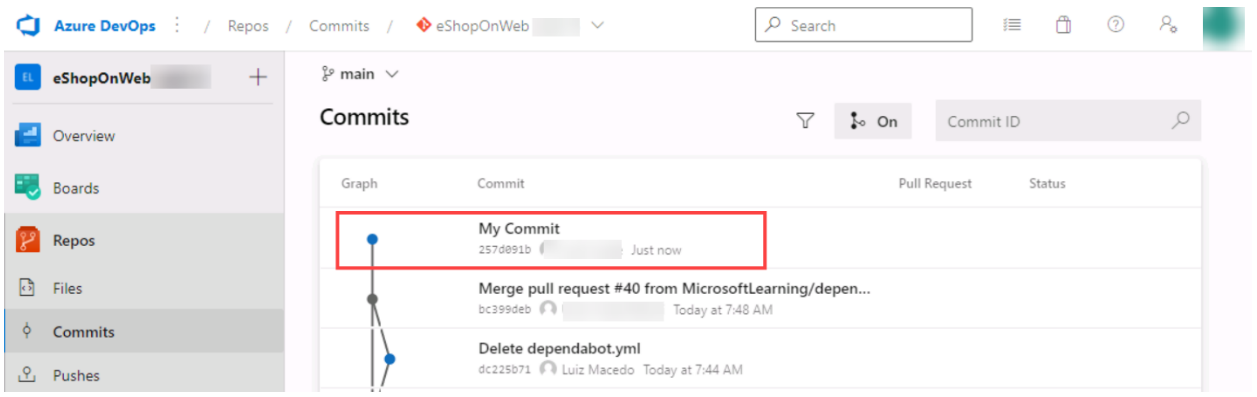
6. If prompted whether you would like to automatically stage your changes and commit them directly, click **Always**.
Note: We will discuss **staging** later in the lab.

7. In the lower left corner of the Visual Studio Code window, to the right of the **main** label, note the **Synchronize Changes** icon of a circle with two vertical arrows pointing in the opposite directions and the number **1** next to the arrow pointing up. Click the icon and, if prompted, whether to proceed, click **OK** to push and pull commits to and from **origin/main**.

Task 2: Review commits

In this task, you will use the Azure DevOps portal to review commits.

1. Switch to the web browser window displaying the Azure DevOps interface.
2. In the vertical navigational pane of the Azure DevOps portal, in the **Repos** section, select **Commits**.
3. Verify that your commit appears at the top of list.



My Commit
257d091b Just now

Merge pull request #40 from MicrosoftLearning/depen...
bc399deb Today at 7:48 AM

Delete dependabot.yml
dc225b71 Luiz Macedo Today at 7:44 AM

Task 3: Stage changes

In this task, you will explore the use of staging changes by using Visual Studio Code. Staging changes allows you to selectively add certain files to a commit while passing over the changes made in other files.

1. Switch back to the **Visual Studio Code** window.
2. Update the open **Program.cs** class by changing the first comment with the following, and saving the file.

C#Copy

```
//My second change
```

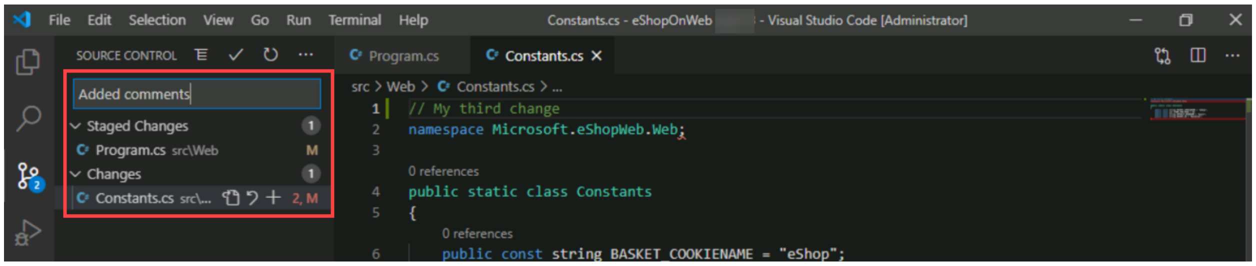
3. In the Visual Studio Code window, switch back the **EXPLORER** tab, navigate to the **/eShopOnWeb/src/Web/Constants.cs** file and select it. This will automatically display its content in the details pane.
4. Add to the **Constants.cs** file a comment on the first line and save the file.

C#Copy

```
// My third change
```

5. In the Visual Studio Code window, switch to the **SOURCE CONTROL** tab, hover the mouse pointer over the **Program.cs** entry, and click the plus sign on the right side of that entry.
Note: This stages the change to the **Program.cs** file only, preparing it for commit without **Constants.cs**.

6. With the **SOURCE CONTROL** tab selected, at the top of the pane, in the textbox, type **Added comments** as the commit message.


File Edit Selection View Go Run Terminal Help
Constants.cs - eShopOnWeb - Visual Studio Code [Administrator]
SOURCE CONTROL
Added comments
Staged Changes 1
Program.cs src\Web M
Changes 1
Constants.cs src\... 2, M
Program.cs
Constants.cs
src > Web > Constants.cs > ...

```
// My third change
namespace Microsoft.eShopWeb.Web;

0 references
public static class Constants
{
    0 references
    public const string BASKET_COOKIENAME = "eShop";
```

7. At the top of the **SOURCE CONTROL** tab, click the ellipsis symbol, in the drop-down menu, select **Commit** and, in the cascading menu, select **Commit Staged**.
8. In the lower left corner of the Visual Studio Code window, click the **Synchronize Changes** button to synchronize the committed changes with the server and, if prompted, whether to proceed, click **OK** to push and pull commits to and from **origin/main**.
Note: Note that since only the staged change was committed, the other change is still pending to be synchronized.

Exercise 3: Review history

In this exercise, you will use the Azure DevOps portal to review history of commits.

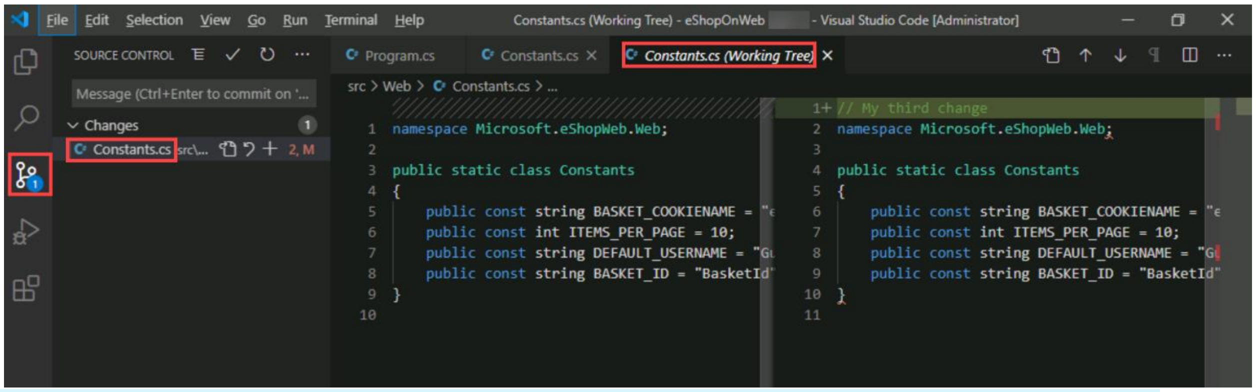
Git uses the parent reference information stored in each commit to manage a full history of your development. You can easily review this commit history to find out when file changes were made and determine differences between versions of your code using the terminal or from one of the many available Visual Studio Code extensions. You can also review changes by using the Azure DevOps portal.

Git's use of the **Branches and Merges** feature works through pull requests, so the commit history of your development doesn't necessarily form a straight, chronological line. When you use history to compare versions, think in terms of file changes between two commits instead of file changes between two points in time. A recent change to a file in the master branch may have come from a commit created two weeks ago in a feature branch that was merged yesterday.

Task 1: Compare files

In this task, you will step through commit history by using the Azure DevOps portal.

1. With the **SOURCE CONTROL** tab of the Visual Studio Code window open, select **Constants.cs** representing the non-staged version of the file.

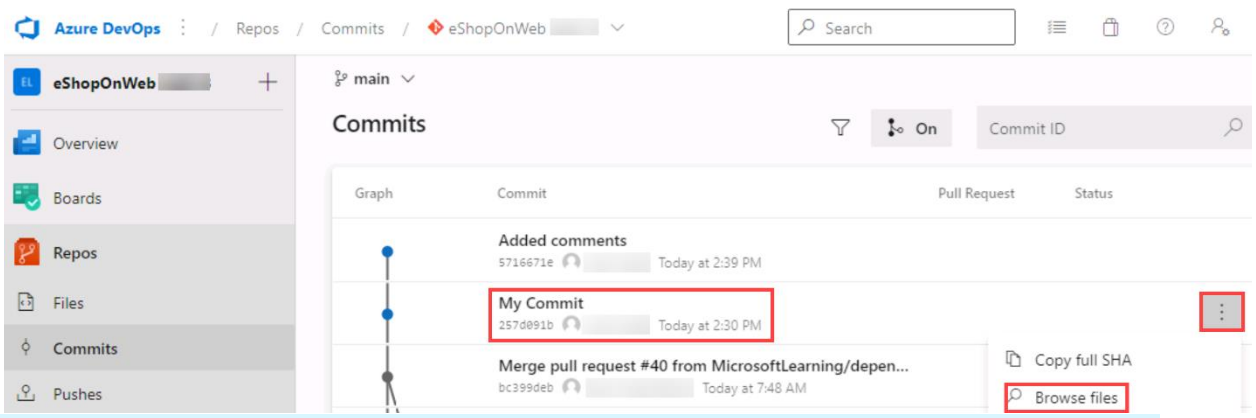

File Edit Selection View Go Run Terminal Help
Constants.cs (Working Tree) - eShopOnWeb [redacted] - Visual Studio Code [Administrator]
SOURCE CONTROL
Message (Ctrl+Enter to commit on '...
Changes 1
Constants.cs src\... 2, M
Program.cs
Constants.cs
Constants.cs (Working Tree)
src > Web > Constants.cs > ...

```
namespace Microsoft.eShopWeb.Web;

public static class Constants
{
    public const string BASKET_COOKIENAME = "E
    public const int ITEMS_PER_PAGE = 10;
    public const string DEFAULT_USERNAME = "Gu
    public const string BASKET_ID = "BasketId"
}
```

```
// My third change
namespace Microsoft.eShopWeb.Web;

public static class Constants
{
    public const string BASKET_COOKIENAME = "E
    public const int ITEMS_PER_PAGE = 10;
    public const string DEFAULT_USERNAME = "Gu
    public const string BASKET_ID = "BasketId"
}
```

Note: A comparison view is opened to enable you to easily locate the changes you've made. In this case, it's just the one comment.

2. Switch to the web browser window displaying the **Commits** pane of the **Azure DevOps** portal to review the source branches and merges. These provide a convenient way to visualize when and how changes were made to the source.
3. Scroll down to the **My commit** entry (pushed before) and hover the mouse pointer over it to reveal the ellipsis symbol on the right side.
4. Click the ellipsis, in the dropdown menu, select **Browse Files**, and review the results.



Graph	Commit	Pull Request	Status
	Added comments 5716671e [redacted] Today at 2:39 PM		
	My Commit 257d091b [redacted] Today at 2:30 PM		
	Merge pull request #40 from MicrosoftLearning/depen... bc399deb [redacted] Today at 7:48 AM		

Note: This view represents the state of the source corresponding to the commit, allowing you to review and download each of source files.

Exercise 4: Work with branches

In this exercise, you will step through scenarios that involve branch management by using Visual Studio Code and the Azure DevOps portal.

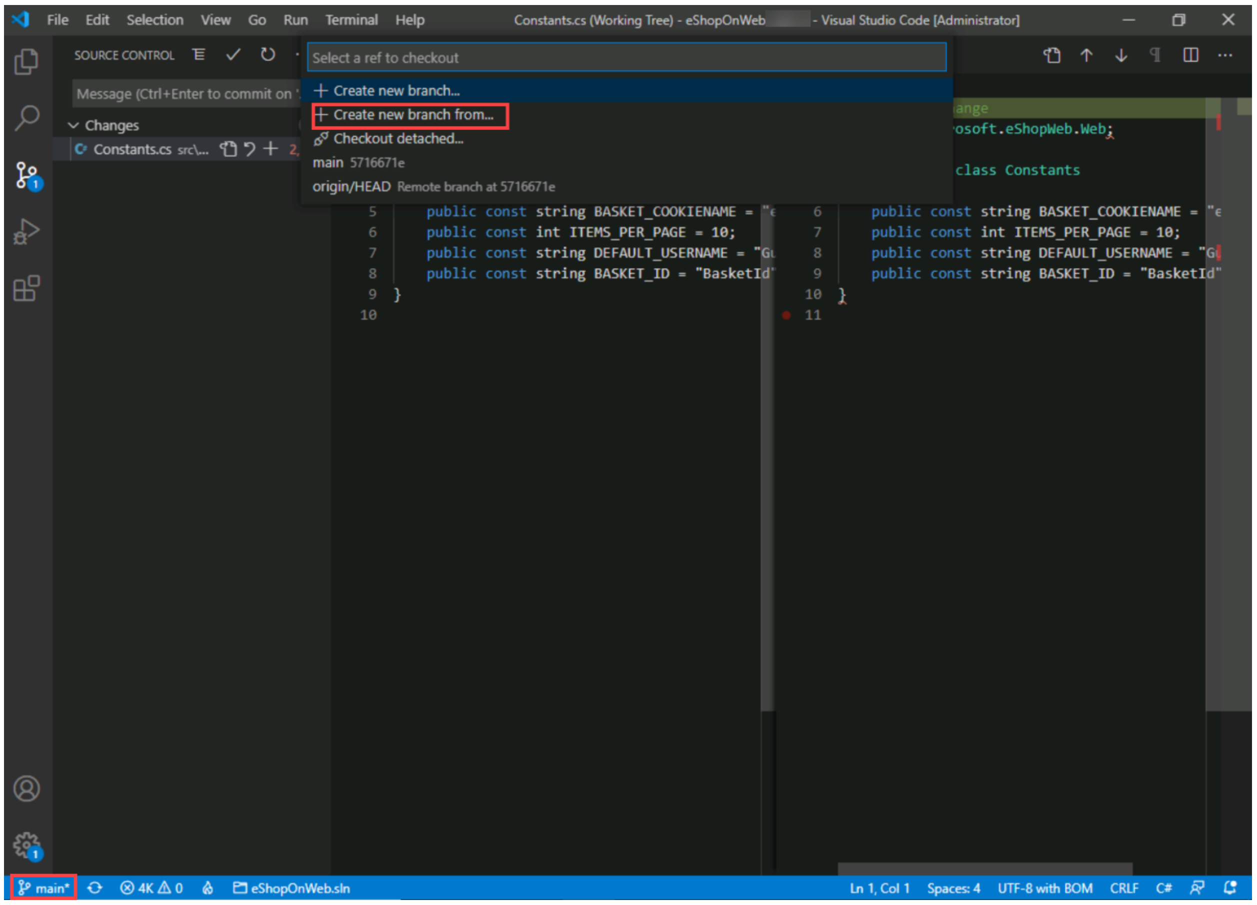
You can manage in your Azure DevOps Git repo from the **Branches** view of **Azure Repos** in the Azure DevOps portal. You can also customize the view to track the branches you care most about so you can stay on top of changes made by your team.

Committing changes to a branch will not affect other branches and you can share branches with others without having to merge the changes into the main project. You can also create new branches to isolate changes for a feature or a bug fix from your master branch and other work. Since the branches are lightweight, switching between branches is quick and easy. Git does not create multiple copies of your source when working with branches, but rather uses the history information stored in commits to recreate the files on a branch when you start working on it. Your Git workflow should create and use branches for managing features and bugfixes. The rest of the Git workflow, such as sharing code and reviewing code with pull requests, all work through branches. Isolating work in branches makes it very simple to change what you are working on by simply changing your current branch.

Task 1: Create a new branch in your local repository

In this task, you will create a branch by using Visual Studio Code.

1. Switch to **Visual Studio Code** running on your lab computer.
2. With the **SOURCE CONTROL** tab selected, in the lower left corner of the Visual Studio Code window, click **main**.
3. In the pop-up window, select **+ Create new branch from…**.


File Edit Selection View Go Run Terminal Help
Constants.cs (Working Tree) - eShopOnWeb - Visual Studio Code [Administrator]
SOURCE CONTROL
Select a ref to checkout
Message (Ctrl+Enter to commit on
Changes
Constants.cs src\...
+ Create new branch...
+ Create new branch from...
Checkout detached...
main 5716671e
origin/HEAD Remote branch at 5716671e
main*
4K 0
eShopOnWeb.sln
Ln 1, Col 1 Spaces: 4 UTF-8 with BOM CRLF C#

```
    public const string BASKET_COOKIENAME = "
    public const int ITEMS_PER_PAGE = 10;
    public const string DEFAULT_USERNAME = "G
    public const string BASKET_ID = "BasketId"
}
```

4. In the **Branch name** textbox, type **dev** to specify the new branch and press **Enter**.

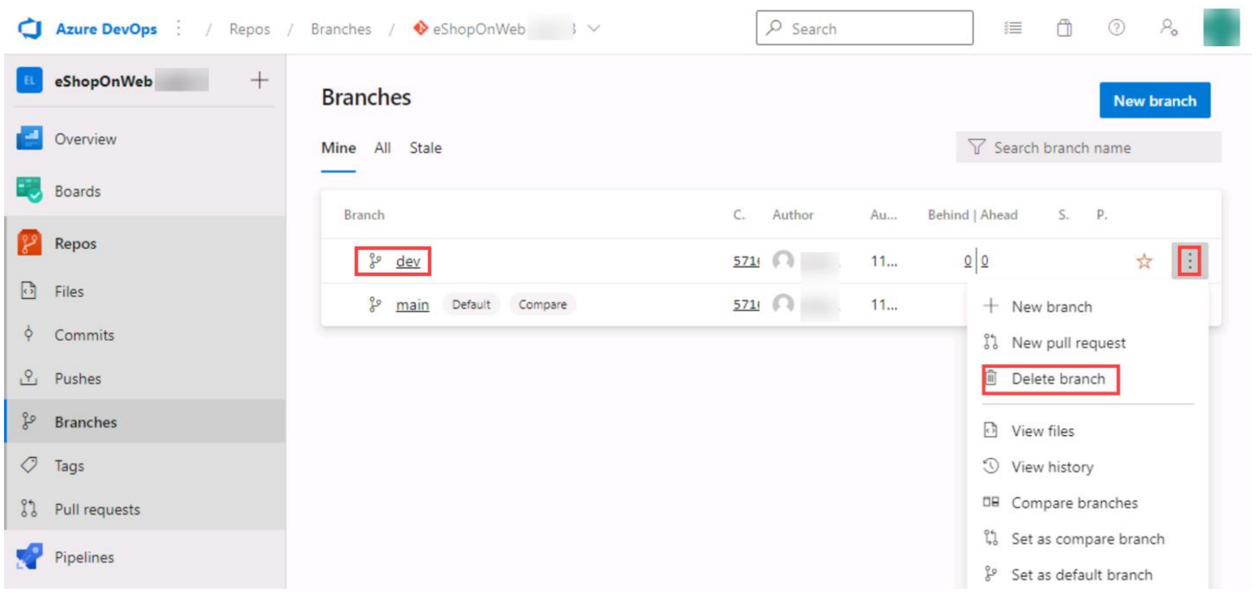
5. In the **Select a ref to create the 'dev' branch from** textbox, select **main** as the reference branch.
   Note: At this point, you are automatically switched to the **dev** branch.

Task 2: Delete a branch

In this task, you will use the Visual Studio Code to work with a branch created in the previous task.

Git keeps track of which branch you are working on and makes sure that, when you check out a branch, your files match the most recent commit on that branch. Branches let you work with multiple versions of the source code in the same local Git repository at the same time. You can use Visual Studio Code to publish, check out and delete branches.

1. In the **Visual Studio Code** window, with the **SOURCE CONTROL** tab selected, in the lower left corner of the Visual Studio Code window, click the **Publish changes** icon (directly to the right of the **dev** label representing your newly created branch).
2. Switch to the web browser window displaying the **Commits** pane of the **Azure DevOps** portal and select **Branches**.
3. On the **Mine** tab of the **Branches** pane, verify that the list of branches includes **dev**.
4. Hover the mouse pointer over the **dev** branch entry to reveal the ellipsis symbol on the right side.
5. Click the ellipsis, in the pop-up menu, select **Delete branch**, and, when prompted for confirmation, click **Delete**.



6. Switch back to the **Visual Studio Code** window and, with the **SOURCE CONTROL** tab selected, in the lower left corner of the Visual Studio Code window, click the **dev** entry. This will display the existing branches in the upper portion of the Visual Studio Code window.
7. Verify that now there are two **dev** branches listed.
   Note: The local (**dev**) branch is listed because it's existence is not affected by the deletion of the branch in the remote repository. The server (**origin/dev**) is listed because it hasn't been pruned.

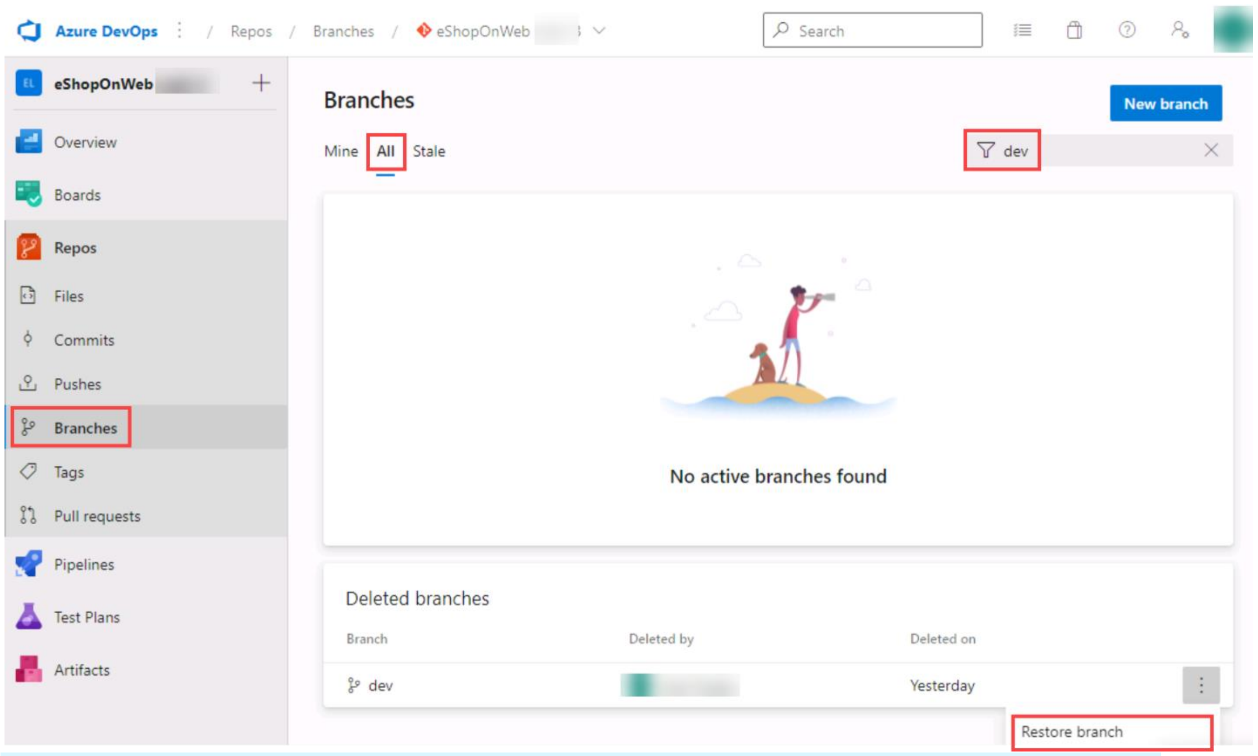
8. In the list of branches select the **main** branch to check it out.
9. Press **Ctrl+Shift+P** to open the **Command Palette**.
10. At the **Command Palette** prompt, start typing **Git: Delete** and select **Git: Delete Branch** when it becomes visible.
11. Select the **dev** entry in the list of branches to delete.
12. In the lower left corner of the Visual Studio Code window, click the **main** entry again. This will display the existing branches in the upper portion of the Visual Studio Code window.
13. Verify that the local **dev** branch no longer appears in the list, but the remote **origin/dev** is still there.
14. Press **Ctrl+Shift+P** to open the **Command Palette**.
15. At the **Command Palette** prompt, start typing **Git: Fetch** and select **Git: Fetch (Prune)** when it becomes visible.
    Note: This command will update the origin branches in the local snapshot and delete those that are no longer there.
    Note: You can check in on exactly what these tasks are doing by selecting the **Output** window in the lower right part bottom of the Visual Studio Code window. If you don't see the Git logs in the output console, make sure to select **Git** as the source.

16. In the lower left corner of the Visual Studio Code window, click the **main** entry again.
17. Verify that the **origin/dev** branch no longer appears in the list of branches.

Task 3: Restore a branch

In this task, you will use the Azure DevOps portal restore the branch you deleted in the previous task.

1. Go to the web browser displaying the **Mine** tab of the **Branches** pane in the Azure DevOps portal.
2. On the **Mine** tab of the **Branches** pane, select the **All** tab.
3. On the **All** tab of the **Branches** pane, in the **Search branch name** text box, type **dev**.
4. Review the **Deleted branches** section containing the entry representing the newly deleted branch.
5. In the **Deleted branches** section, hover the mouse pointer over the **dev** branch entry to reveal the ellipsis symbol on the right side.
6. Click the ellipsis, in the pop-up menu and select **Restore branch**.



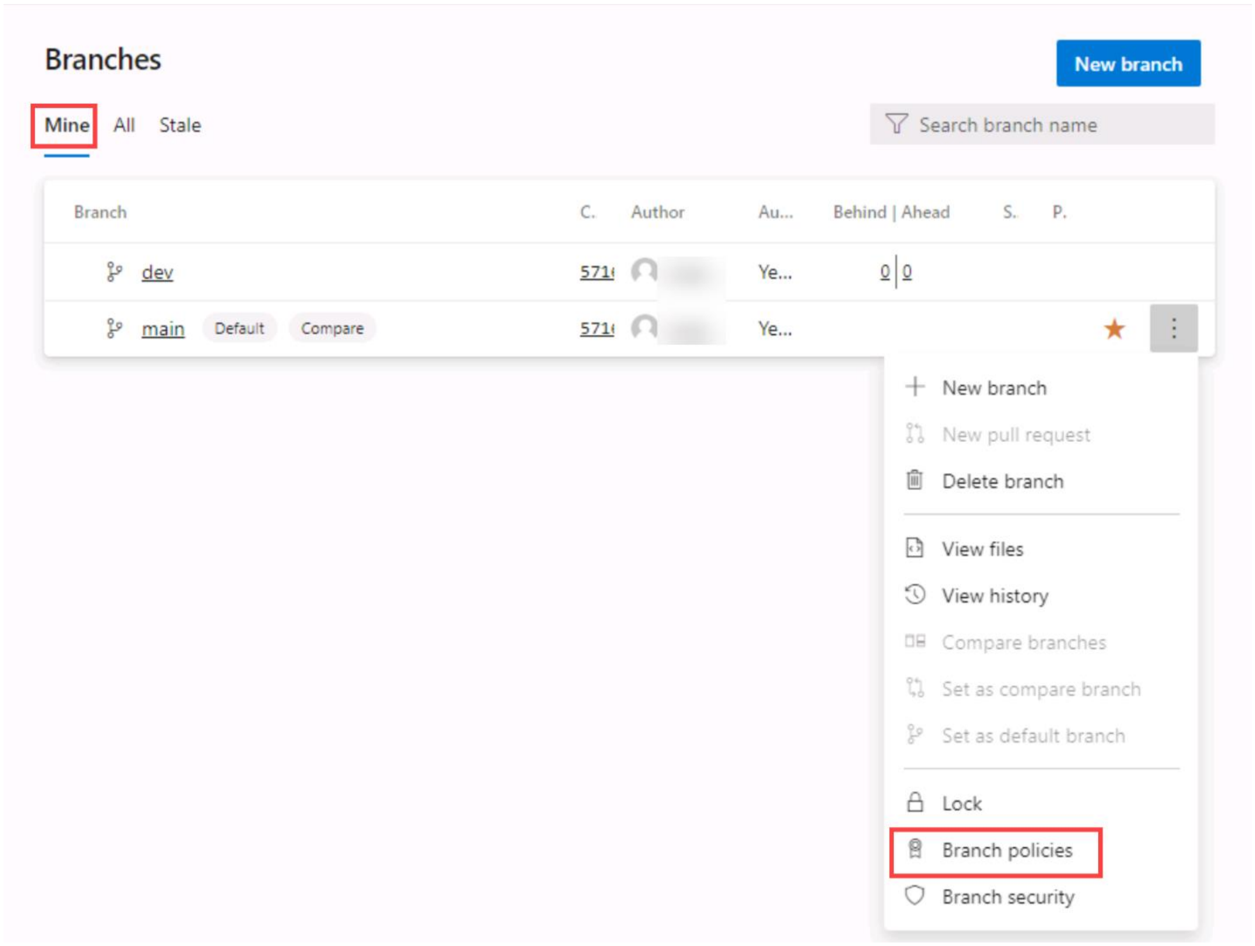
Note: You can use this functionality to restore a deleted branch as long as you know its exact name.

Task 4: Branch Policies

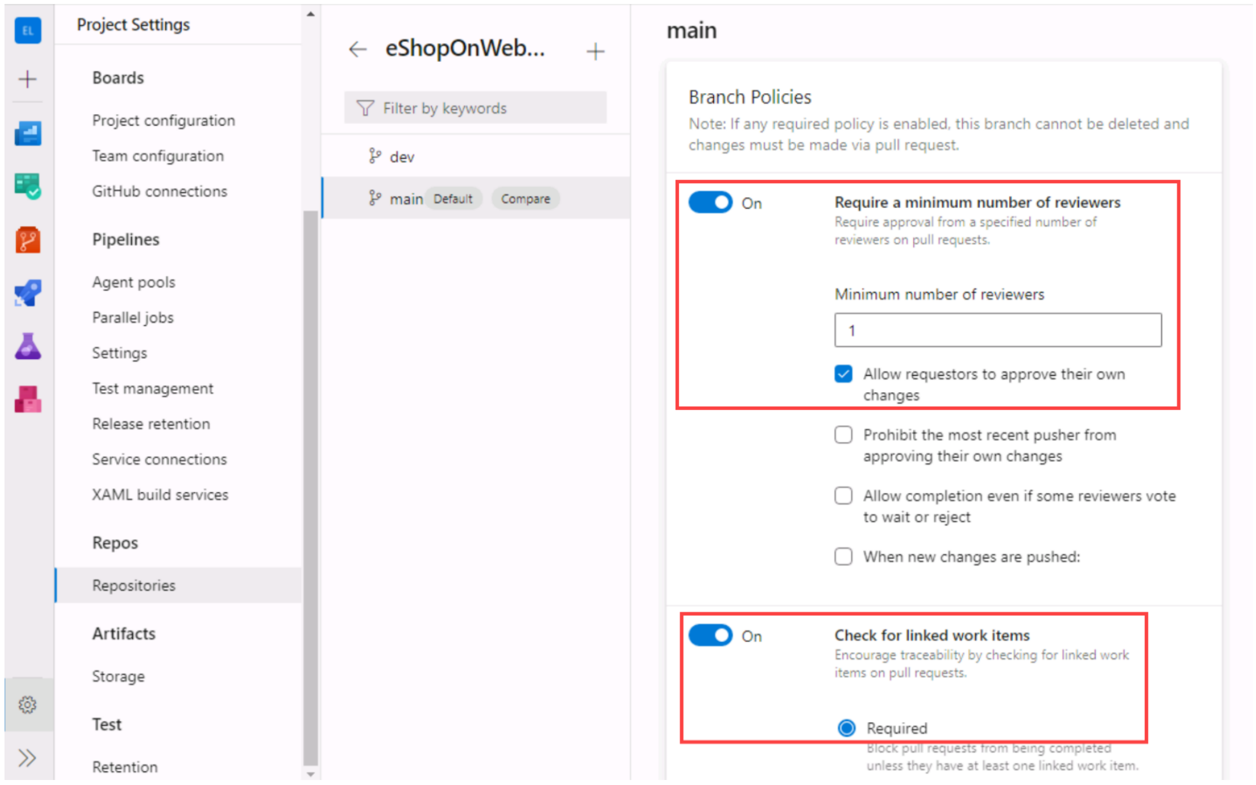
In this task, you will use the Azure DevOps portal to add policies to the main branch and only allow changes using Pull Requests that comply with the defined policies. You want to ensure that changes in a branch are reviewed before they are merged.

For simplicity we will work directly on the web browser repo editor (working directly in origin), instead of using the local clone in VS code (recommended for real scenarios).

1. Switch to the web browser displaying the **Mine** tab of the **Branches** pane in the Azure DevOps portal.
2. On the **Mine** tab of the **Branches** pane, hover the mouse pointer over the **main** branch entry to reveal the ellipsis symbol on the right side.
3. Click the ellipsis and, in the pop-up menu, select **Branch Policies**.



4. On the **main** tab of the repository settings, enable the option for **Require minimum number of reviewers**. Add **1** reviewer and check the box **Allow requestors to approve their own changes**(as you are the only user in your project for the lab)
5. On the **main** tab of the repository settings, enable the option for **Check for linked work items** and leave it with **Required** option.



main

Branch Policies

Note: If any required policy is enabled, this branch cannot be deleted and changes must be made via pull request.

On **Require a minimum number of reviewers**
Require approval from a specified number of reviewers on pull requests.

Minimum number of reviewers: 1

☑ Allow requestors to approve their own changes

☐ Prohibit the most recent pusher from approving their own changes

☐ Allow completion even if some reviewers vote to wait or reject

☐ When new changes are pushed:

On **Check for linked work items**
Encourage traceability by checking for linked work items on pull requests.

● Required
Block pull requests from being completed unless they have at least one linked work item.

Task 5: Testing branch policy

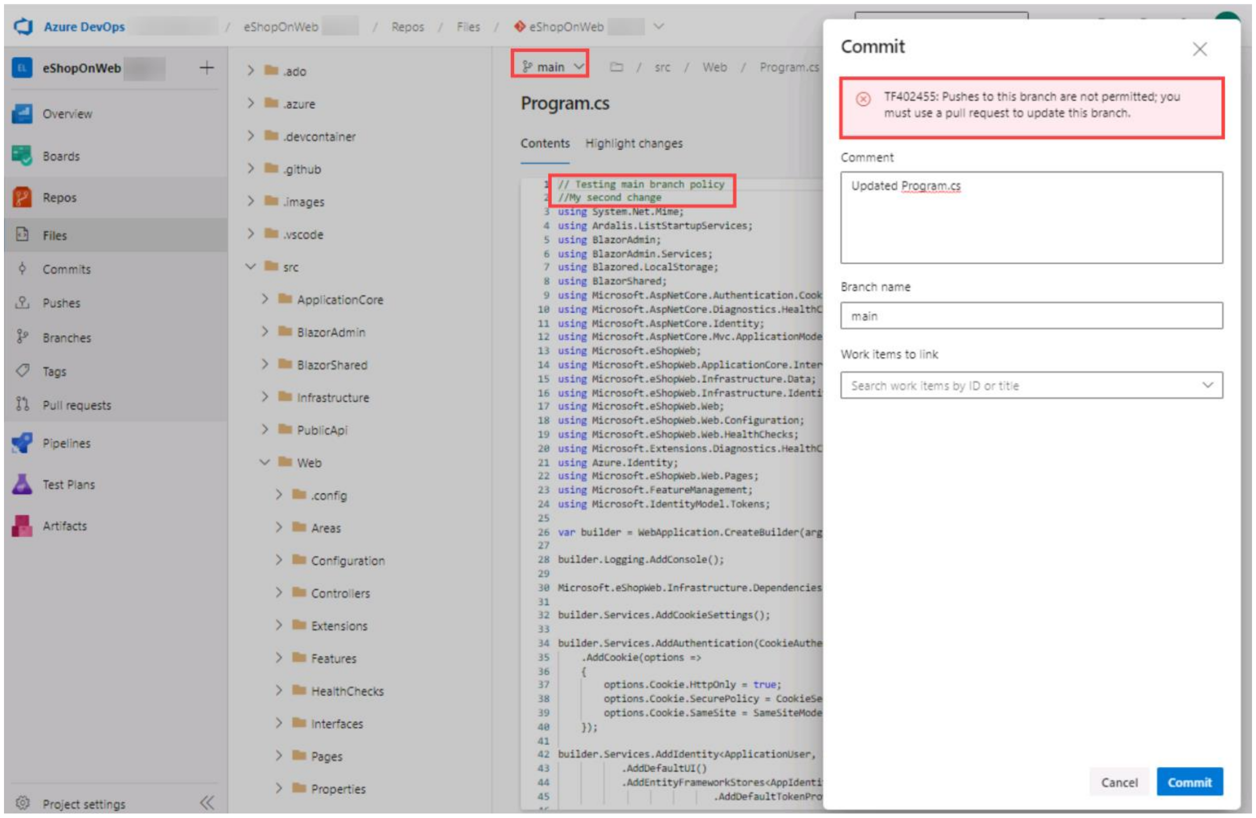
In this task, you will use the Azure DevOps portal to test the policy and create your first Pull Request.

1. In the vertical navigational pane of the of the Azure DevOps portal, in the **Repos>Files**, make sure the **main** branch is selected (dropdown above shown content).
2. To make sure policies are working, try making a change and committing it on the **main** branch, navigate to the **/eShopOnWeb/src/Web/Program.cs** file and select it. This will automatically display its content in the details pane.
3. On the first line add the following comment:

C#Copy

```
// Testing main branch policy
```

4. Click on **Commit > Commit**. You will see a warning: changes to the main branch can only be done using a Pull Request.



Program.cs

```
// Testing main branch policy
//My second change
using System.Net.Mime;
using Ardalis.ListStartupServices;
using BlazorAdmin;
using BlazorAdmin.Services;
using Blazored.LocalStorage;
using BlazorShared;
using Microsoft.AspNetCore.Authentication.Cook
using Microsoft.AspNetCore.Diagnostics.HealthC
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc.ApplicationMode
using Microsoft.eShopWeb;
using Microsoft.eShopWeb.ApplicationCore.Inter
using Microsoft.eShopWeb.Infrastructure.Data;
using Microsoft.eShopWeb.Infrastructure.Identi
using Microsoft.eShopWeb.Web;
using Microsoft.eShopWeb.Web.Configuration;
using Microsoft.eShopWeb.Web.HealthChecks;
using Microsoft.Extensions.Diagnostics.HealthC
using Azure.Identity;
using Microsoft.eShopWeb.Web.Pages;
using Microsoft.FeatureManagement;
using Microsoft.IdentityModel.Tokens;

var builder = WebApplication.CreateBuilder(arg

builder.Logging.AddConsole();

Microsoft.eShopWeb.Infrastructure.Dependencies

builder.Services.AddCookieSettings();

builder.Services.AddAuthentication(CookieAuthe
    .AddCookie(options =>
    {
        options.Cookie.HttpOnly = true;
        options.Cookie.SecurePolicy = CookieSe
        options.Cookie.SameSite = SameSiteMode
    });

builder.Services.AddIdentity<ApplicationUser,
        .AddDefaultUI()
        .AddEntityFrameworkStores<AppIdenti
                .AddDefaultTokenPro
```

Commit

TF402455: Pushes to this branch are not permitted; you must use a pull request to update this branch.

Comment: Updated Program.cs

Branch name: main

Work items to link: Search work items by ID or title

5. Click on **Cancel** to skip the commit.

Task 6: Working with Pull Requests

In this task, you will use the Azure DevOps portal to create a Pull Request, using the **dev** branch to merge a change into the protected **main** branch.

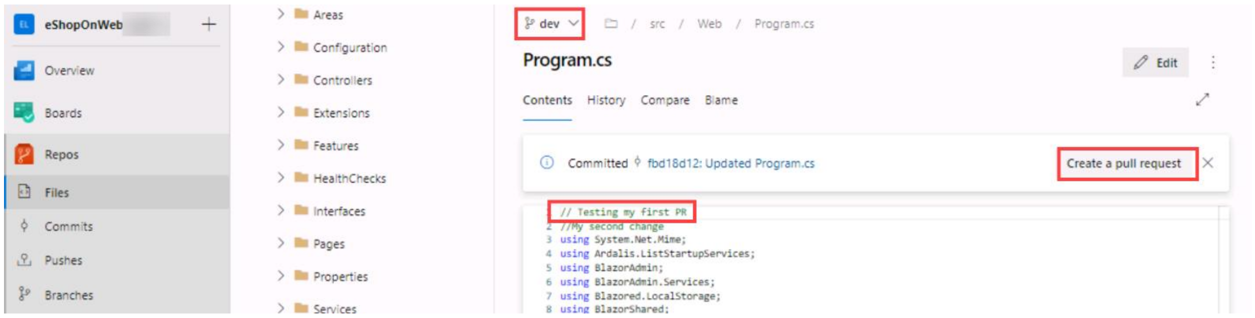
**An Azure DevOps work item with be linked to the changes to be able to trace pending work with code activity.

1. In the vertical navigational pane of the of the Azure DevOps portal, in the **Boards** section, select **Work Items**.
2. Click on **+ New Work Item > Product Backlog Item**. In title field, write **Testing my first PR** and click on **Save**.
3. Now go back to the vertical navigational pane of the of the Azure DevOps portal, in the **Repos>Files**, make sure the **dev** branch is selected.
4. Navigate to the **/eShopOnWeb/src/Web/Program.cs** file and make the following change on the first line:

C#Copy

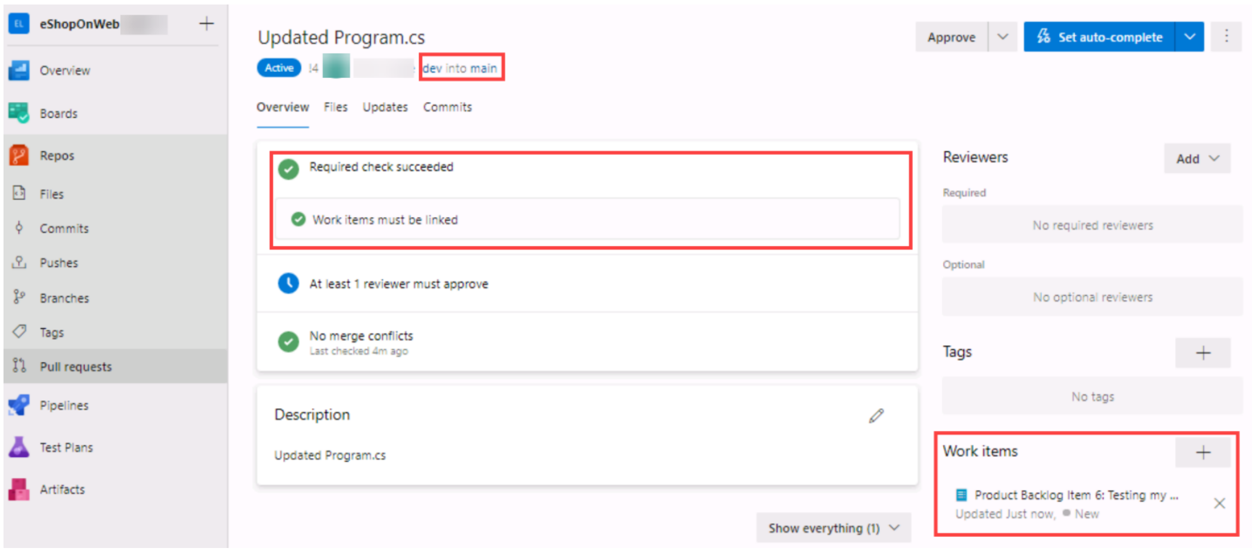
```
// Testing my first PR
```

5. Click on **Commit > Commit** (leave default commit message). This time the commit works, **dev** branch has no policies.
6. A message will pop-up, proposing to create a Pull Request (as you **dev** branch is now ahead in changes, compared to **main**). Click on **Create a Pull Request**.



```
// Testing my first PR
//My second change
using System.Net.Mime;
using Ardalis.ListStartupServices;
using BlazorAdmin;
using BlazorAdmin.Services;
using Blazored.LocalStorage;
using BlazorShared;
```

7. In the **New pull request** tab, leave defaults and click on **Create**.
8. The Pull Request will show some failed/pending requirements, based on the policies applied to our target **main** branch.
   - Proposed changes should have a work item linked
   - At least 1 user should review and approve the changes.
9. On the right side options, click on the **+** button next to **Work Items**. Link the previously created work item to the Pull Request by clicking on it. You will see one of the requirements changes status.



10. Next, open the **Files** tab to review the proposed changes. In a more complete Pull Request, you would be able to review files one by one (marked as reviewed) and open comments for lines that may not be clear (hovering the mouse over the line number gives you an option to post a comment).

11. Go back to the **Overview** tab, and on the top-right click on **Approve**. All the requirements will change to green. Now you can click on **Complete**.
12. On the **Complete Pull Request** tab, multiple options will be given before completing the merge:
    - **Merge Type**: 4 merge types are offered, you can review them here or observing the given animations. Choose **Merge (no fast forward)**.
    - **Post-complete options**:
        - Check **Complete associated work item…**. It will move associated PBI to **Done** state.
13. Click on **Complete Merge**

Task 7: Applying tags

The product team has decided that the current version of the site should be released as v1.1.0-beta.

1. In the vertical navigational pane of the of the Azure DevOps portal, in the **Repos** section, select **Tags**.
2. In the **Tags** pane, click **New tag**.
3. In the **Create a tag** panel, in the **Name** text box, type **v1.1.0-beta**, in the **Based on** drop-down list leave the **main** entry selected, in the **Description** text box, type **Beta release v1.1.0** and click **Create**.
   Note: You have now tagged the repository at this release (the latest commit gets linked to the tag). You could tag commits for a variety of reasons and Azure DevOps offers the flexibility to edit and delete them, as well as manage their permissions.

Endlab 01

02 lab: Azure Boards & Work Item Types (WIT)

[Agile plan and portfolio management with Azure Boards - Training | Microsoft Learn]

In this lab, you'll learn about the agile planning and portfolio management tools and processes provided by Azure Boards and how they can help you quickly plan, manage, and track work across your entire team. You'll explore the product backlog, sprint backlog, and task boards that can track the flow of work during an iteration. We'll also look at the enhanced tools in this release to scale for larger teams and organizations.

Objectives

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

- Manage teams, areas, and iterations.
- Manage work items.
- Manage sprints and capacity.
- Customize Kanban boards.
- Define dashboards.
- Customize team process.

2. Module 01: Get started on a DevOps transformation journey'

1.

2.

3.

4.

5.

6.

7.

8.

9.

10.

11. In the left pane, click "Policies" under "Security" and turn on "Third-party application access via OAuth" under "Application connection policies" and turn on "Allow public projects" under "Security policies".

Note If you already created this project during previous labs, this exercise can be skipped.

In this exercise, you will set up the prerequisites for the lab, which consist of a new Azure DevOps project with a repository based on the eShopOnWeb https://github.com/MicrosoftLearning/eShopOnWeb.

3. Module 01: Get started on a DevOps transformation journey'

Task 1: *Create and configure* **eShopOnWeb**

New Project. Give your project the name **eShopOnWeb**. Define **Private** as Visibility option.

Click **Advanced** and specify **Scrum** as **Work Item Process**. Click on **Create**.

4. Module 01: Get started on a DevOps transformation journey'

Exercise 1: *Manage Agile project*

In this exercise, you will use Azure Boards to perform a number of common agile planning and portfolio management tasks, including management of teams, areas, iterations, work items, sprints and capacity, customizing Kanban boards, defining dashboards, and customizing team processes.

Task 1: *Manage teams, areas, and iterations*

Each new project is configured with a default team, which name matches the project name. You have the option of creating additional teams. Each team can be granted access to a suite of Agile tools and team assets.

1. Verify that the web browser displays your Azure DevOps organization with the **EShopOnWeb** project you generated in the previous exercise.

2. Click the cogwheel icon labeled **Project settings** located in the lower left corner of the page to open the **Project settings** page.

3. In the **General** section, select the **Teams** tab. There is already a default team in this project, **EShopOnWeb Team** but you'll create a new one for this lab. Click **New Team**.

4. On the **Create a new team** pane, in the **Team name** textbox, type **EShop-Web**, leave other settings with their default values, and click **Create**.

5. In the list of **Teams**, select the newly created team to view its details.
   Note: By default, the new team has only you as its member. You can use this view to manage such functionality as team membership, notifications, and dashboards.

6. Click **Iterations and Area Paths** link **at the top of the EShop-Web** page to start defining the schedule and scope of the team.

7. At the top of the **Boards** pane, select the **Iterations** tab and then click **+ Select iteration(s)**.

8. Select **EShopOnWeb\Sprint 1** and click **Save and close**. Note that this first sprint will show up in the list of iterations, but the Dates are not set yet.

9. Select **Sprint 1** and click the **ellipsis (...)**. From the context menu, select **Edit**.
   Note: Specify the Start Date as the first work day of last week, and count 3 full work weeks for each sprint. For example, if March 6 is the first work day of the sprint, it goes until March 24th. Sprint 2 starts on March 27, which is 3 weeks out from March 6.

10. Repeat the previous step to add **Sprint 2** and **Sprint 3**. You could say that we are currently in the 2nd week of the first sprint.

12. Back on the **Boards** pane, at the top of the pane, select the **Areas** tab. You will find there an automatically generated area with the name matching the name of the team.

13. Click the ellipsis symbol (...) next to the **default area** entry and, in the dropdown list, select **Include sub areas**.

Note: The default setting for all teams is to exclude sub-area paths. We will change it to include sub-areas so that the team gets visibility into all of the work items from all teams. Optionally, the management team could also choose to not include sub-areas, which automatically removes work items from their view as soon as they are assigned to one of the teams.

6. Module 01: Get started on a DevOps transformation journey'

Task 2: *Manage work items*

Work items are the workhorse of modern projects. In this task you'll focus on using various work items to set up the plan to extend the Parts Unlimited site with a product training section. While it can be daunting to build out such a substantial part of a company's offering, Azure DevOps and the Scrum process make it very manageable.

1. In the vertical navigational pane of the Azure DevOps portal, select the **Boards** icon and, select **Work Items**.
   Note: There are many ways to create work items

2. On the **Work Items** window, click on **+ New Work Item > Epic**.

3. In the **Enter title** textbox, type **Product training**.

4. In the upper left corner, select the **Unassigned** entry and, in the dropdown list, select your user account in order to assign the new work item to yourself.

5. Next to the **Area** entry, select the **eShopOnWeb** entry and, in the dropdown list, select **EShop-WEB**. This will set the **Area** to **eShopOnWeb\EShop-WEB**.

6. Next to the **Iteration** entry, select the **eShopOnWeb** entry and, in the dropdown list, select **Sprint 2**. This will set the **Iteration** to **eShopOnWeb\Sprint 2**.

7. Click **Save** to finalize your changes. **Do not close it**.
   Note: Ordinarily you would want to fill out as much information as possible, but this is sufficient for the purposes of this lab.

Note: The work item form includes all the associated information and history for how it has been handled since creation. One of the key areas is the **Related Work**.

☐

8. In the **Related work** section on the lower right-side, select the **Add link** entry and, in the dropdown list, select **New item**.

☐

9. On the **Add link** panel, in the **Link Type** dropdown list, select **Child**. Next, in the **Work item type** dropdown list, select **Feature**, in the **Title** textbox, type **Training dashboard** and click **OK**.
   Note: On the **Training dashboard** panel, note that the assignment, **Area**, and **Iteration** are already set to the same values as the epic that the feature is based on. In addition, the feature is automatically linked to the parent item it was created from.

☐

10. On the **Training dashboard** panel, click **Save & Close**.

☑

11. In the vertical navigation pane of the Azure DevOps portal, in the list of the **Boards** items, select **Boards**.

☐

12. On the **Boards** panel, select the **EShop-WEB boards** entry. This will open the board for that particular team.

☐

13. On the **Boards** panel, in the upper right corner, select the **Backlog items** entry and, in the dropdown list, select **Features**.
    Note: This will make it easy to add tasks and other work items to the features.

☐

14. Hover with the mouse pointer over the rectangle representing the **Training dashboard** feature. This will reveal the ellipsis symbol (...) in its upper right corner.

☐

15. Click the ellipsis (...) icon and, in the dropdown list, select **Add Product Backlog Item**.

☐

16. In the textbox of the new product backlog item, type **As a customer, I want to view new tutorials** and press the **Enter** key to save the entry.

Note: This creates a new product backlog item (PBI) work item that is a child of the feature and shares its area and iteration.

17. Repeat the previous step to add two more PBIs designed to enable the customer to see their recently viewed tutorials and to request new tutorials named, respectively, **As a customer, I want to see tutorials I recently viewed** and **As a customer, I want to request new tutorials**.

18. On the **Boards** panel, in the upper right corner, select the **Features** entry and, in the dropdown list, select **Backlog items**.
    Note: Backlog items have a state that defines where they are relative to being completed. While you could open and edit the work item using the form, it's easier to just drag cards on the board.

19. Then on the **Board** tab of the **EShop-WEB** panel in Backlog items, drag the first work item named **As a customer, I want to view new tutorials** from the **New** to **Approved** stage.
    Note: You can also expand work item cards to get to conveniently editable details.

20. Hover with the mouse pointer over the rectangle representing the work item you moved to the **Approved** stage. This will reveal the down facing caret symbol.

21. Click the down facing caret symbol to expand the work item card, select the **Unassigned** entry, and in the list of user accounts, select your account to assign the moved PBI to yourself.

22. On the **Board** tab of the **EShop-WEB** panel, drag the second work item named **As a customer, I want to see tutorials I recently viewed** from the **New** to the **Committed** stage.

23. On the **Board** tab of the **EShop-WEB** panel, drag the third work item named **As a customer, I want to request new tutorials** from the **New** to the **Done** stage.

24. On the **Board** tab of the **EShop-WEB** pane, at the top of the pane, click **View as Backlog** to display the tabular form.
    Note: You can use the plus sign directly under the **Backlog** tab label of the **EShop-WEB** panel to view nested tasks under these work items.
    Note: You can use the second plus sign directly left to the first backlog item to add a new task to it.

25. On the **Backlog** tab of the **EShop-WEB** pane, in the upper left corner of the pane, click the second plus sign from the top, the one next to the first work item. This will display the **NEW TASK** panel.

26. At the top of the **NEW TASK** panel, in the **Enter title** textbox, type **Add page for most recent tutorials**.

27. On the **NEW TASK** panel, in the **Remaining Work** textbox, type **5**.

28. On the **NEW TASK** panel, in the **Activity** dropdown list, select **Development**.

29. On the **NEW TASK** panel, click **Save & Close**.

30. Repeat the last five steps to add another task named **Optimize data query for most recent tutorials**. Set its **Remaining Work** to **3** and its **Activity** to **Design**. Click **Save & Close** once completed.

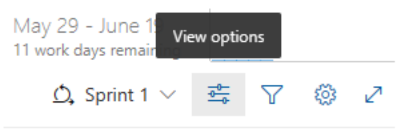
7. Module 01: Get started on a DevOps transformation journey'

Task 3: *Manage sprints and capacity*

The sprint backlog should contain all the information the team needs to successfully plan and complete work within the time allotted. Before planning the sprint, you'd want to have created, prioritized, and estimated the backlog and defined the sprints.

1. In the vertical navigational pane of the Azure DevOps portal, select the **Boards** icon and, in the list of the **Boards** items, select **Sprints**.

2. On the **Taskboard** tab of the **Sprints** view, in the toolbar, on the right-hand side, select the **View options** symbol (directly to the left of the funnel icon) and, in the **View options** dropdown list, select the **Work details** entry. Select Sprint 2



Note: The current sprint has a pretty limited scope. There are two tasks in the **To do** stage. At this point, neither task has been assigned. Both show a numeric value to the right of **Unassigned** entry representing the remaining work estimate.

3. In the rectangle representing the **Add page for most recent tutorials**, click the **Unassigned** entry and, in the list of user accounts, select your account to assign the task to yourself.

4. Select the **Capacity** tab of the **Sprints** view.
   Note: This view enables you to define what activities a user can take on and at what level of capacity.

5. On the **Capacity** tab of the **Sprints** view, directly under the **Activity** label, in the **Unassigned** dropdown list, select **Development** and, in the **Capacity per day** textbox, type **1**.
   Note: This represents 1 hour of development work per day. Note that you can add additional activities per user in the case they do more than just development.
   Note: You're going to take some vacation.

6. On the **Capacity** tab of the **Sprints** view, directly next to the entry representing your user account, in the **Days off** column, click the **0 days** entry. This will display a panel where you can set your days off.

7. In the displayed panel, use the calendar view to set your vacation to span five work days during the current sprint (within the next three weeks) and, once completed, click **OK**.

8. Back on the **Capacity** tab of the **Sprints** view, click **Save**.

9. Select the **Taskboard** tab of the **Sprints** view.
   Note: The **Work details** panel has been updated to reflect your available bandwidth. The actual number displayed in the **Work details** panel might vary, but your total sprint capacity will be

equal to the number of working days remaining till the end of the sprint, since you allocated 1 hour per day. Take a note of this value since you will use it in the upcoming steps.
Note: One convenient feature of the boards is that you can easily update key data in-line. It's a good practice to regularly update the **Remaining Work** estimate to reflect the amount of time expected for each task. Let's say you've reviewed the work for the **Add page for most recent tutorials** task and found that it will actually take longer than originally expected.

☐

10. On the **Taskboard** tab of the **Sprints** view, in the square box representing the **Add page for most recent tutorials**, set the estimated number of hours to **14**, to match your total capacity for this sprint, which you identified in the previous step.
Note: This automatically expands the **Development** and your personal capacities to their maximum. Since they're large enough to cover the assigned tasks, they stay green. However, the overall **Team** capacity is exceeded due to the additional 3 hours required by the **Optimize data query for most recent tutorials** task.
Note: One way to resolve this capacity issue would be to move the task to a future iteration.

☐

11. On the **Taskboard** tab of the **Sprints** view, in the toolbar, on the right-hand side, select the **View options** symbol (directly to the left of the funnel icon) and, in the **View options** dropdown list, select the **People** entry.
Note: This adjusts your view such that you can review the progress of tasks by person instead of by backlog item.
Note: There is also a lot of customization available.

☐

12. Click the **Configure team settings** cogwheel icon (directly to the right of the funnel icon).

☑

13. On the **Settings** panel, select the **Styles** tab, click **+ Styling rule**, under the **Rule name** label, in the **Name** textbox, type **Development**, and, in the **Card color** dropdown list, select the green rectangle.
Note: This will color all cards green if they meet the rule criteria set directly below, in the **Rule criteria** section.

☐

14. In the **Rule criteria** section, in the **Field** dropdown list, select **Activity**, in the **Operator** dropdown list, select **=**, and, in the **Value** dropdown list, select **Development**.
Note: This will set all cards assigned to **Development** activities green.

☐

15. On the **Settings** panel, select the **Backlogs** tab.
Note: Entries on this tab allow you to set the levels available for navigation. Epics are not included by default, but you could change that.

☐

16. On the **Settings** panel, select the **Working days** tab.
    Note: Entries on this tab allow you to specify the **Working days** the team follows. This applies to capacity and burndown calculations.

☐

17. On the **Settings** panel, select the **Working with bugs** tab.
    Note: Entries on this tab allow you to specify how bugs are presented on the board.

☑

18. On the **Settings** panel, click **Save and close** to save the styling rule.
    Note: The task associated with **Development** is now green and very easy to identify.
    Save and Close

8. Module 01: Get started on a DevOps transformation journey'

Task 4: *Customize Kanban boards*

☐

1. In the vertical navigational pane of the Azure DevOps portal, in the list of the **Boards** items, select **Boards**.

☐

2. On the **Boards** panel, click the **Configure team settings** cogwheel icon (directly to the right of the funnel icon).
   Note: The team is emphasizing work done with data, so there is special attention paid to any task associated with accessing or storing data.

☐

3. On the **Settings** panel, select the **Tag colors** tab, click **+ Tag color**, in the **Tag** textbox, type **data** and leave the default color in place.
   Note: Whenever a backlog item or bug is tagged with **data**, that tag will be highlighted.

☐

4. On the **Settings** panel, select the **Annotations** tab.
   Note: You can specify which **Annotations** you would like included on cards to make them easier to read and navigate. When an annotation is enabled, the child work items of that type are easily accessible by clicking the visualization on each card.

☐

5. On the **Settings** panel, select the **Tests** tab.
   Note: The **Tests** tab enables you to configure how tests appear and behave on the cards.

☐

6. On the **Settings** panel, click **Save and close** to save the styling rule.

7. On the **Board** tab of the **EShop-WEB** panel, right-click the Work Item representing the **As a customer, I want to view new tutorials** backlog item and select **Open**.

8. On the **As a customer, I want to view new tutorials** panel, at the top of the panel, to the right of the **0 comments** entry, click **Add tag**.

9. In the resulting textbox, type **data** and press the **Enter** key.

10. Repeat the previous step to add the **ux** tag.

11. On the **As a customer, I want to view new tutorials** panel, click **Save & Close**.
    Note: The two tags are now visible on the card, with the **data** tag highlighted in yellow as configured.

12. On the **Boards** panel, click the **Configure team settings** cogwheel icon (directly to the right of the funnel icon).

13. On the **Settings** panel, select the **Columns** tab.
    Note: This section allows you to add new stages to the workflow.

14. Click **+ Column**, under the **Column name** label, in the **Name** textbox, type **QA Approved** and, in the **WIP limit** textbox, type **1**
    Note: The Work in progress limit of 1 indicates that only one work item should be in this stage at a time. You would ordinarily set this higher, but there are only two work items to demonstrate the feature.

15. On the **Settings** panel, on the **Columns** tab, drag and drop the newly created tab between **Committed** and **Done**.

16. On the **Settings** panel, click **Save and close**.
    Note: Verify that you now see the new stage in the workflow.

17. **Refresh** the **Boards portal**, so the **QA Approved**4 column is visible in the Kanban board view now.

18. Drag the **As a customer, I want to see tutorials I recently viewed** work item from the **Committed** stage into the **QA Approved** stage.

19. Drag the **As a customer, I want to view new tutorials** work item from the **Approved** stage into the **QA Approved** stage.
    Note: The stage now exceeds its **WIP** limit and is colored red as a warning.

20. Move the **As a customer, I want to see tutorials I recently viewed** backlog item back to **Committed**.

21. On the **Boards** panel, click the **Configure team settings** cogwheel icon (directly to the right of the funnel icon).

22. On the **Settings** panel, return to the **Columns** tab and select the **QA Approved** tab.

    Note: A lag often exists between when work gets moved into a column and when work starts.

    - To counter that lag and reveal the actual state of work in progress, you can turn on split columns.
    - When split, each column contains two sub-columns: **Doing** and **Done**. Split columns let your team implement a pull model.
    - Without split columns, teams push work forward, to signal that they've completed their stage of work. However, pushing it to the next stage doesn't necessarily mean that a team member immediately starts work on that item.

23. On the **QA Approved** tab, enable the **Split column into doing and done** checkbox to create two separate columns.
    Note: As your team updates the status of work as it progresses from one stage to the next, it helps that they agree on what **done** means.

By specifying the **Definition of done** criteria for each Kanban column, you help share the essential tasks to complete before moving an item into a downstream stage.

24. On the **QA Approved** tab, at the bottom of the panel, in the **Definition of done** textbox, type **Passes **all** tests**.

25. On the **Settings** panel, click **Save and close**.
    Note: The **QA Approved** stage now has **Doing** and **Done** columns. You can also click the informational symbol (with letter **i** in a circle) next to the column header to read the **Definition of done**.

26. On the **Boards** panel, click the **Configure team settings** cogwheel icon (directly to the right of the funnel icon).
    Note: Your Kanban board supports your ability to visualize the flow of work as it moves from new to done. When you add **swimlanes**, you can also visualize the status of work that supports different service-level classes. You can create a swimlane to represent any other dimension that supports your tracking needs.

27. On the **Settings** panel, select the **Swimlanes** tab.

28. On the **Swimlanes** tab, click **+ Swimlane**, directly under the **Swimlane name** label, in the **Name** textbox, type **Expedite**.

29. On the **Settings** panel, click **Save and close**.

30. Back on the **Board** tab of the **Boards** panel, drag and drop the **Committed** work item onto the **QA Approved | Doing** stage of the **Expedite** swimlane so that it gets recognized as having priority when QA bandwidth becomes available.
    Note: If you would like to review a more sophisticated board with many more work items, on the **Board** tab of the **Boards** panel, in the upper left corner, select **EShop-WEB** and, in the dropdown list of teams, select the **eShopOnWeb Team**. This board provides a playground for you to experiment with and review the results.

9. Module 01: Get started on a DevOps transformation journey'

Task 5: *Customize team process*

In this task we'll create a custom Scrum-based process. The process will include a backlog item field designed to track to a proprietary PartsUnlimited ticket ID.
In Azure DevOps, you customize your work tracking experience through a process. A process defines the building blocks of the work item tracking system as well as other sub-systems you access through Azure DevOps.
Whenever you create a team project, you select the process which contains the building blocks you want for your project.
Azure DevOps supports two process types. The first, the core system processes (Scrum, Agile, and CMMI) are read-only, so you cannot customize them.
The second type, inherited processes, you create based on core system processes, with the option of customizing their settings.
All processes are shared within the same organization. That is, one or more team projects can reference a single process. Instead of customizing a single team project, you customize a process. Changes made to the process automatically update all team projects that reference that process. Once you've created an inherited process, you can customize it, create team projects based on it, and migrate existing team projects to reference it. The Git team project can't be customized until it's migrated to an inherited process.

1. On the Azure DevOps page, click the **Azure DevOps** logo in the top left corner to navigate to the account root page.

2. In the left bottom corner of the page, click **Organization settings**.

3. In the **Organization Settings** vertical menu, in the **Boards** section, select **Process**.

4. On the **All processes** pane, to the right of the **Scrum** entry, select the ellipsis symbol (...) and, in the dropdown menu, select **Create inherited process**.

5. In the **Create inherited process from Scrum** panel, in the **Process name (required)** textbox, type **Customized Scrum** and click **Create process**.

6. Back on the **All processes** pane, click the **Customized Scrum** entry.
   Note: You may need to refresh the browser for the new process to become visible.

7. On the **All processes > Customized Scrum** pane, select **Product Backlog Item**.

8. On the **All processes > Customized Scrum > Product Backlog Item** pane, click **New field**.

9. On the **Add a field to Product Backlog Item** panel, on the **Definition** tab, in the **Create a field** section, in the **Name** textbox, type **EShop Ticket ID**.

10. On the **Add a field to Product Backlog Item** panel, click **Layout**.

11. On the **Add a field to Product Backlog Item** panel, on the **Layout** tab, in the **Label** textbox, type **Ticket ID**, select the **Create a new group** option, in the **Group** textbox, type **EShopOnWeb**, and click **Add field**.
    Note: Now that the customized process has been configured, let's switch to the eShopOnWeb project to use it.

12. Return to the **All processes** root using the breadcrumb path at the top of the **All processes > Customized Scrum > Product Backlog Item** pane.

13. On the **All processes** pane, select the **Scrum** entry.
    Note: Our current project uses **Scrum**.

14. On the **All processes > Scrum** pane, select the **Projects** tab.

15. In the list of projects, in the row containing the **eShopOnWeb** entry, select the ellipsis symbol (...) and then select **Change process**.

16. On the **Change the project process** pane, in the **Select a target process** dropdown list, select the **Customized Scrum** process, click **Save** and then click **Close**.

17. Click the **Azure DevOps** logo in the top left corner to return to the account root page.

18. On the **Projects** tab, select the entry representing the **eShopOnWeb** project.

19. In the vertical menu on the left side of the **eShopOnWeb** page, select **Boards** and ensure that the **Work Items** pane is displayed.

20. In the list of work items, click the first backlog item.

21. Verify that you now have the **Ticket ID** field under the **PartsUnlimited** group, which was defined during the process customization. You can treat this like any other text field.
    Note: Once the work item is saved, Azure DevOps will also save the new custom information so that it will be available for queries and through the rest of Azure DevOps.

10. Module 01: Get started on a DevOps transformation journey'

Exercise 2 (optional) : *Define dashboards*

In this task, you will step through the process of creating dashboards and their core components. Dashboards allow teams to visualize status and monitor progress across the project

1. In the vertical navigational pane of the Azure DevOps portal, select the **Overview** icon and, in the list of the **Overview** items, select **Dashboards**.

2. If necessary, on the **Dashboards** pane, in the upper left corner, in the **eShopOnWeb Team** section, select **eShopOnWeb Team - Overview** and review the existing dashboard.

3. On the **Dashboards** pane, select the drop-down menu next to the **eShopOnWeb Team - Overview** title, and select **+ New dashboard**.

4. On the **Create a dashboard** pane, in the **Name** textbox, type **Product training**, in the **Team** dropdown list, select the **EShop-WEB** team, and click **Create**.

5. On the new dashboard pane, click **Add a widget**.

6. On the **Add Widget** panel, in the **Search** textbox, type **sprint** to find existing widgets that focus on sprints. In the list of results, select **Sprint Overview** and click **Add**.

7. In the rectangle representing the newly added widget, click the **Settings** cogwheel icon and review the **Configuration** pane.
   Note: The customization level will vary by widget.

8. On the **Configuration** pane, click **Close** without making any changes.

9. Back on the **Add Widget** pane, in the **Search** textbox, type **sprint** again to find existing widgets that focus on sprints. In the list of results, select **Sprint Capacity** and click **Add**.

10. In the **Dashboard** view, at the top of the pane, click **Done Editing**.
    Note: You can now review two important aspects of your current sprint on your custom dashboard.
    Note: Another way of customizing dashboards is to generate charts based on work item queries, which you can share to a dashboard.

11. In the vertical navigational pane of the Azure DevOps portal, select the **Boards** icon and, in the list of the **Boards** items, select **Queries**.

12. On the **Queries** pane, click **+ New query**.

13. On the **Editor** tab of **Queries > My Queries** pane, in the **Value** dropdown list of the **Work Item Type** row, select **Task**.

14. On the **Editor** tab of **Queries > My Queries** pane, in the second row, in the **Field** column, select **Area Path** and, in the corresponding **Value** dropdown list, select **eShopOnWeb\EShop-WEB**.

15. Click **Save query**.

16. In the **New query** panel, in the **Enter name** textbox, type **Web tasks**, in the **Folder** dropdown list, select **Shared Queries**, and click **OK**.

17. Select the **Charts** tab and click **+ New chart**.

18. On the **Configure Chart** panel, in the **Name** textbox, type **Web tasks - By assignment**, in the **Group by** dropdown list, select **Assigned To**, and click **OK** to save the changes.
    Note: You can now add this chart to a dashboard.

☑ Endlab 02

03 Lab: Enabling Continuous Integration with Azure Pipelines

- Include build validation as part of a Pull Request.
- Configure CI pipeline as code with YAML.
- AZ400-DesigningandImplementingMicrosoftDevOpsSolutions (microsoftlearning.github.io)

Exercise 1: Include build validation as part of a Pull Request

In this exercise, you will include build validation to validate a Pull Request.

Task 1: Import the YAML build definition

In this task, you will import the YAML build definition that will be used as a Branch Policy to validate the pull requests.

Let's start by importing the build pipeline named eshoponweb-ci-pr.yml.

1. Go to **Pipelines>Pipelines**
2. Click on **Create Pipeline** or **New Pipeline** button
3. Select **Azure Repos Git (YAML)**
4. Select the **eShopOnWeb** repository
5. Select **Existing Azure Pipelines YAML File**
6. Select the **/.ado/eshoponweb-ci-pr.yml** file then click on **Continue**

   The build definition consists of the following tasks:

   - **DotNet Restore**: With NuGet Package Restore you can install all your project's dependency without having to store them in source control.
   - **DotNet Build**: Builds a project and all of its dependencies.
   - **DotNet Test**: .Net test driver used to execute unit tests.
   - **DotNet Publish**: Publishes the application and its dependencies to a folder for deployment to a hosting system. In this case, it's **Build.ArtifactStagingDirectory**.
7. Click the **Save** button to save the pipeline definition
8. Your pipeline will take a name based on the project name. Let's **rename** it for identifying the pipeline better. Go to **Pipelines>Pipelines** and click on the recently created pipeline. Click on the ellipsis and **Rename/Remove** option. Name it **eshoponweb-ci-pr** and click on **Save**.

Task 2: Branch Policies

In this task, you will add policies to the main branch and only allow changes using Pull Requests that comply with the defined policies. You want to ensure that changes in a branch are reviewed before they are merged.

1. Go to **Repos>Branches** section.
2. On the **Mine** tab of the **Branches** pane, hover the mouse pointer over the **main** branch entry to reveal the ellipsis symbol on the right side.
3. Click the ellipsis and, in the pop-up menu, select **Branch Policies**.
4. On the **main** tab of the repository settings, enable the option for **Require minimum number of reviewers**. Add **1** reviewer and check the box **Allow requestors to approve their own changes**(as you are the only user in your project for the lab)
5. On the **main** tab of the repository settings, in the **Build Validation** section, click **+** (Add a new build policy) and in the **Build pipeline** list, select **eshoponweb-ci-pr** then click **Save**.

Task 3: Working with Pull Requests

In this task, you will use the Azure DevOps portal to create a Pull Request, using a new branch to merge a change into the protected **main** branch.

1. Navigate to the **Repos** section in the eShopOnWeb navigation and click **Branches**.
2. Create a new branch named **Feature01** based on the **main** branch.
3. Click **Feature01* and navigate to the **/eShopOnWeb/src/Web/Program.cs** file as part of the **Feature01** branch and make the following change on the first line:

   C#Copy

```
// Testing my PR
```

4. Click on **Commit > Commit** (leave default commit message).
5. A message will pop-up, proposing to create a Pull Request (as your **Feature01** branch is now ahead in changes, compared to **main**). Click on **Create a Pull Request**.
6. In the **New pull request** tab, leave defaults and click on **Create**.
7. The Pull Request will show some pending requirements, based on the policies applied to the target **main** branch.
   - At least 1 user should review and approve the changes.
   - Build validation, you will see that the build **eshoponweb-ci-pr** was triggered automatically

8. After all validations are successful, on the top-right click on **Approve**. Now from the **Set auto-complete** dropdown you can click on **Complete**.
9. On the **Complete Pull Request** tab, click on **Complete Merge**

Exercise 2: Configure CI Pipeline as Code with YAML

In this exercise, you will configure CI Pipeline as code with YAML.

Task 1: Import the YAML build definition for CI

Let's start by importing the CI pipeline named eshoponweb-ci.yml.

1. Go to **Pipelines>Pipelines**.
2. Click on **New Pipeline** button.
3. Select **Azure Repos Git (YAML)**.
4. Select the **eShopOnWeb** repository.
5. Select **Existing Azure Pipelines YAML File**.
6. Select the **/.ado/eshoponweb-ci.yml** file then click on **Continue**.

   The CI definition consists of the following tasks:

   - **DotNet Restore**: With NuGet Package Restore you can install all your project's dependency without having to store them in source control.
   - **DotNet Build**: Builds a project and all of its dependencies.
   - **DotNet Test**: .Net test driver used to execute unit tests.
   - **DotNet Publish**: Publishes the application and its dependencies to a folder for deployment to a hosting system. In this case, it's **Build.ArtifactStagingDirectory**.
   - **Publish Artifact - Website**: Publish the app artifact (created in the previous step) and make it available as a pipeline artifact.
   - **Publish Artifact - Bicep**: Publish the infrastructure artifact (Bicep file) and make it available as a pipeline artifact.

Task 2: Enable Continuous Integration

The default build pipeline definition doesn't enable Continuous Integration.

1. Now, you need to replace the **trigger: none** code with the following code:

   CodeCopy

```
trigger:
  branches:
    include:
    - main
  paths:
    include:
    - src/web/*
```

This will automatically trigger the build pipeline if any change is made to the main branch and the web application code (the src/web folder).

Since you enabled Branch Policies, you need to pass by a Pull Request in order to update your code.

2. Click the **Save** button (not **Save and run**) to save the pipeline definition.
3. Select **Create a new branch for this commit**
4. Keep the default branch name and **Start a pull request** checked.
5. Click on **Save**
6. Your pipeline will take a name based on the project name. Let's **rename** it for identifying the pipeline better. Go to **Pipelines>Pipelines** and click on the recently created pipeline. Click on the ellipsis and **Rename/Remove** option. Name it **eshoponweb-ci** and click on **Save**.
7. Go to **Repos > Pull Requests**
8. Click on the existing pull request
9. After all validations are successful, on the top-right click on **Approve**. Now you can click on **Complete**.
10. On the **Complete Pull Request** tab, Click on **Complete Merge**

Task 3: Test the CI pipeline

In this task, you will create a Pull Request, using a new branch to merge a change into the protected **main** branch and automatically trigger the CI pipeline.

1. Navigate to the **Repos** section
2. Create a new branch named **Feature02** based on the **main** branch
3. Click the new **Feature02** branch
4. Navigate to the **/eShopOnWeb/src/Web/Program.cs** file and remove the first line:

C#Copy

```
// Testing my PR
```

5. Click on **Commit > Commit** (leave default commit message).
6. A message will pop-up, proposing to create a Pull Request (as your **Feature02** branch is now ahead in changes, compared to **main**).
7. Click on **Create a Pull Request**.
8. In the **New pull request** tab, leave defaults and click on **Create**.
9. The Pull Request will show some pending requirements, based on the policies applied to the target **main** branch.
10. After all validations are successful, on the top-right click on **Approve**. Now from the **Set auto-complete** dropdown you can click on **Complete**.
11. On the **Complete Pull Request** tab, Click on **Complete Merge**
12. Go back to **Pipelines>Pipelines**, you will notice that the build **eshoponweb-ci** was triggered automatically after the code was merged.
13. Click on the **eshoponweb-ci** build then select the last run.
14. After its successful execution, click on **Related > Published** to check the published artifacts:
    - Bicep: the infrastructure artifact.
    - Website: the app artifact.

EndLab03

Monitor

Test

1. Add, run, and update inline tests in Azure Boards and Azure DevOps - Azure Boards | Microsoft Learn
2. Define inline tests, or a set of manual tests cases, for a backlog item from your Kanban board. Not only can you add tests, you can run them and update their status.

MS Online labs:

Version Controlling with Git in Visual Studio Code and Azure DevOps | Azure DevOps Hands-on-Labs (azuredevopslabs.com)

Agile plan and portfolio management with Azure Boards - Training | Microsoft Learn