**Exercise - Set up your environment**

In this section, you make sure that your Azure DevOps organization is set up to complete the rest of this module. You also create the Azure App Service environments that you'll deploy to.

To meet these objectives, you:

* Add a user to ensure that Azure DevOps can connect to your Azure subscription.
* Set up an Azure DevOps project for this module.
* In Azure Boards, move the work item for this module to the **Doing** column.
* Make sure that your project is set up locally so that you can push changes to the pipeline.
* Create the Azure App Service environments by using the Azure CLI in Azure Cloud Shell.
* Create pipeline variables that define the names of your App Service environments.
* Create a service connection that enables Azure Pipelines to securely access your Azure subscription.
* Perform an initial deployment.

**Add a user to Azure DevOps**

To complete this module, you need your own Azure subscription . You can get started with Azure for free.

Although you don't need an Azure subscription to work with Azure DevOps, here you'll use Azure DevOps to deploy to resources that exist in your Azure subscription. To simplify the process, use the same Microsoft account to sign in to both your Azure subscription and your Azure DevOps organization.

If you use different Microsoft accounts to sign in to Azure and Azure DevOps, then add a user to your DevOps organization under the Microsoft account that you use to sign in to Azure. For more information, see Add users to your organization or project . When you add the user, choose the **Basic** access level.

Then sign out of Azure DevOps and sign in again under the Microsoft account that you use to sign in to your Azure subscription.

**Get the Azure DevOps project**

Here you make sure that your Azure DevOps organization is set up to complete the rest of this module. You set it up by running a template that creates a project in Azure DevOps.

The modules in this learning path are part of a progression. You follow the Tailspin web team through their DevOps journey. For learning purposes, each module has an associated Azure DevOps project.

**Run the template**

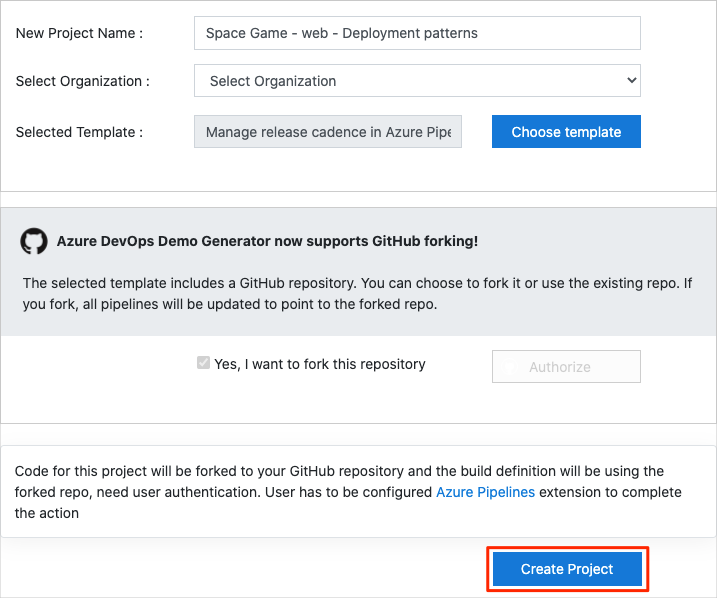
Run a template that sets up your Azure DevOps organization.

Run the template by clicking on the link below

https://azuredevopsdemogenerator.azurewebsites.net/?name=manage-release-cadence

On the Azure DevOps Demo Generator site, follow these steps to run the template:

1. Select **Sign In** and accept the usage terms.
2. On the **Create New Project** page, select your Azure DevOps organization. Then enter a project name, such as *Space Game - web - Deployment patterns*.



1. Select **Yes, I want to fork this repository**. Then select **Authorize**.
2. Select **Create Project**.

The template takes a few moments to run.

1. Select **Navigate to project** to go to your project in Azure DevOps.

**Important**

The **Clean up your Azure DevOps environment** page in this module contains important cleanup steps. Cleaning up helps ensure that you don't run out of free build minutes. Be sure to follow the cleanup steps even if you don't complete this module.

**Set your project's visibility**

Your fork of the *Space Game* repository on GitHub is initially public. The Azure DevOps template creates a project that's initially private.

A public GitHub repository is accessible to everyone, whereas a private repository is accessible to you and the people you share it with. In both cases, only collaborators can commit changes to a GitHub repository.

A project on Azure DevOps works the same way. Users who aren't signed in to the service have read-only access to public projects. Private projects require users to be granted access to the project and signed in to access the services.

For learning purposes, you don't need to change any of these settings right now. But for your own projects, you need to decide what visibility and access you want to provide to others. For example, if your project is open source, you might make both your GitHub repository and your Azure DevOps project public. If your project is closed source, you would likely make both your GitHub repository and your Azure DevOps project private.

Later, you can use these resources to help you decide which option is best for your project:

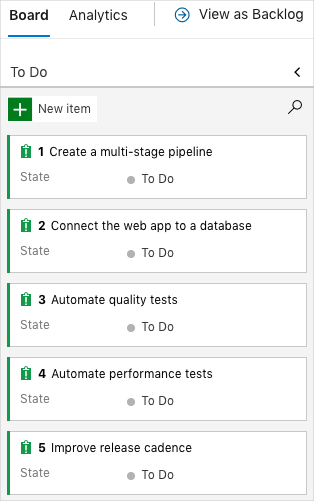
* What is a public project?
* Quickstart: Change the project visibility, public or private
* Setting repository visibility

**Move the work item to Doing**

In this part, in Azure Boards you assign yourself a work item that relates to this module. You also move the work item to the **Doing** state. In practice, your team would create work items at the start of each *sprint*, or work iteration.

Assigning work in this way gives you a checklist to work from. It gives your team visibility into what you're working on and how much work is left. It also helps the team enforce limits on work in progress (WIP) so that the team doesn't take on too much work at one time.

Recall that the team settled on these top issues for the current sprint:



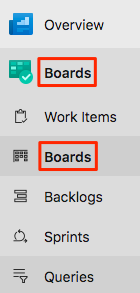
**Note**

Within an Azure DevOps organization, work items are numbered sequentially. In your project, the number for each work item might not match what you see here.

Here you move the fifth item, **Improve release cadence**, to the **Doing** column. Then you assign yourself to the work item. **Improve release cadence** relates to choosing a deployment pattern that enables you to quickly release changes to your users.

To set up the work item:

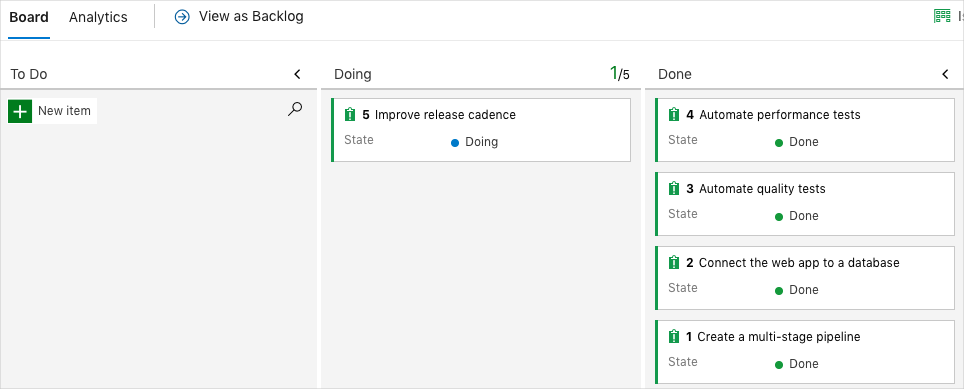
1. In Azure DevOps, go to **Boards**, and then select **Boards** from the menu.



1. At the bottom of the **Improve release cadence** work item, select the down arrow. Then assign the work item to yourself.

Assigning the work item to yourself

1. Move the work item from the **To Do** column to the **Doing** column.



At the end of this module, you move the card to the **Done** column after you complete the task.

**Set up the project locally**

Here you load the *Space Game* project in Visual Studio Code, configure Git, clone your repository locally, and set the upstream remote so that you can download starter code.

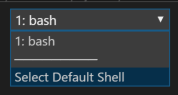
**Note**

If you're already set up with the **mslearn-tailspin-spacegame-web-deploy** project locally, you can move to the next section.

**Open the integrated terminal**

Visual Studio Code comes with an integrated terminal. Here you can both edit files and work from the command line.

1. Start Visual Studio Code.
2. On the **View** menu, select **Terminal**.
3. In the drop-down list, select **bash**:



In the terminal window, you can choose any shell that's installed on your system. For example, you can choose Bash, Zsh, or PowerShell.

Here you'll use Bash. Git for Windows provides Git Bash, which makes it easy to run Git commands.

**Note**

On Windows, if you don't see **bash** listed as an option, make sure you've installed **Git** and then restart Visual Studio Code.

If you still don't see the **bash** option, see **Integrated Terminal** to manually configure your terminal settings.

1. Run the cd command to navigate to the directory where you want to work. Choose your home directory (~) or a different directory if you want.

**Bash**

cd ~

**Configure Git**

If you're new to Git and GitHub, first run a few commands to associate your identity with Git and authenticate with GitHub. For more information, see Set up Git .

At a minimum, you need to complete the following steps. Run the commands from the integrated terminal.

1. Set your username .
2. Set your commit email address .
3. Cache your GitHub password .

**Note**

If you already use two-factor authentication with GitHub, **create a personal access token**. When you're prompted, use your token in place of your password.

Treat your access token like a password. Keep it in a safe place.

**Set up your project in Visual Studio Code**

In the Build applications with Azure DevOps  learning path, you forked and then cloned a Git repository. The repository contains the source code for the *Space Game* website. Your fork was connected to your projects in Azure DevOps so that the build runs when you push changes to GitHub.

**Important**

In this learning path, we switch to a different Git repository, **mslearn-tailspin-spacegame-web-deploy**. When you ran the template to set up your Azure DevOps project, the process forked the repository automatically for you.

In this part, you clone your fork locally so that you can change and build out your pipeline configuration.

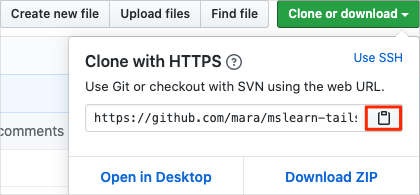
**Clone your fork locally**

You now have a copy of the *Space Game* web project in your GitHub account. Now you'll download, or *clone*, a copy to your computer so you can work with it.

A clone, just like a fork, is a copy of a repository. When you clone a repository, you can make changes, verify that they work as you expect, and then upload those changes to GitHub. You can also synchronize your local copy with changes that other authenticated users have made to the GitHub copy of your repository.

To clone the *Space Game* web project to your computer:

1. Go to your fork of the *Space Game* web project (**mslearn-tailspin-spacegame-web-deploy**) on GitHub .
2. Select **Clone or download**. Then select the button next to the URL to copy the URL to your clipboard:



1. In Visual Studio Code, go to the terminal window.
2. In the terminal, move to the directory where you want to work. Choose your home directory (~) or a different directory if you want.

**Bash**

**cd ~**

1. Run the git clone command. Replace the URL that's shown here with the contents of your clipboard:

**Bash**

**git clone https://github.com/your-name/mslearn-tailspin-spacegame-web-deploy.git**

1. Move to the mslearn-tailspin-spacegame-web-deploy directory. This is the root directory of your repository.

**Bash**

**cd mslearn-tailspin-spacegame-web-deploy**

**Set the upstream remote**

A *remote* is a Git repository where team members collaborate. It's like a repository on GitHub.

Run this git remote command to list your remotes:

**Bash**

**git remote -v**

You see that you have both fetch (download) and push (upload) access to your repository:

OutputCopy

origin https://github.com/username/mslearn-tailspin-spacegame-web-deploy.git (fetch)

origin https://github.com/username/mslearn-tailspin-spacegame-web-deploy.git (push)

*Origin* specifies your repository on GitHub. When you fork code from another repository, the original remote (the one you forked from) is often named *upstream*.

Run this git remote add command to create a remote named *upstream* that points to the Microsoft repository:

**Bash**

**git remote add upstream https://github.com/MicrosoftDocs/mslearn-tailspin-spacegame-web-deploy.git**

Run git remote again to see the changes:

**Bash**

**git remote -v**

You see that you still have both fetch (download) access and push (upload) access to your repository. You also now have fetch access to the Microsoft repository:

OutputCopy

origin https://github.com/username/mslearn-tailspin-spacegame-web-deploy.git (fetch)

origin https://github.com/username/mslearn-tailspin-spacegame-web-deploy.git (push)

upstream https://github.com/MicrosoftDocs/mslearn-tailspin-spacegame-web-deploy.git (fetch)

**Open the project**

In Visual Studio Code, your terminal window points to the root directory of the *Space Game* web project. You'll now open the project to view its structure and work with files.

1. On the **File** menu, select **Open**.
2. Navigate to the root directory of the *Space Game* web project. If you need a reminder of the full path, you can run the pwd command in the terminal window to see the path.

You see the directory and file tree.

**Note**

You might need to open the integrated terminal again after you open the folder.

You're now set up to work with the *Space Game* source code and your Azure Pipelines configuration from your local development environment.

**Create the Azure App Service environments**

Here you create the environments that define the pipeline stages. You create one App Service instance that corresponds to each stage: *Dev*, *Test*, and *Staging*.

In previous modules, you used the Azure CLI to create your App Service instances. Here you'll do the same.

**Important**

You need your own Azure subscription to complete the exercises in this module.

**Bring up Cloud Shell through the Azure portal**

1. Go to the Azure portal  and sign in.
2. On the menu, select **Cloud Shell**. When you're prompted, select the **Bash** experience.

**Select an Azure region**

Here you specify the default *region*, or geographic location, where your Azure resources will be created.

1. In Cloud Shell, run the following az account list-locations command to list the regions that are available from your Azure subscription.

**Azure CLI**

az account list-locations \

--query "[].{Name: name, DisplayName: displayName}" \

--output table

1. From the **Name** column in the output, choose a region that's close to you. For example, choose **eastasia** or **westus2**.
2. Run az configure to set your default region. Replace **<REGION>** with the name of the region that you chose.

**Azure CLI**

az configure --defaults location=<REGION>

This example sets **westus2** as the default region:

**Azure CLI**

az configure --defaults location="South India"

**Create the App Service instances**

Here you create App Service instances for the three stages that you'll deploy to: *Dev*, *Test*, and *Staging*. You'll add a deployment slot to *Staging* later in this module.

**Note**

For learning purposes, here you use the default network settings. These settings make your site accessible from the internet. In practice, you could configure an Azure virtual network that places your website in a network that's not routable from the internet but that only your team can access. Later, when you're ready, you could reconfigure your network to make the website available to your users.

1. In Cloud Shell, generate a random number that makes your web app's domain name unique:

**Bash**

webappsuffix=$RANDOM

1. Run the following az group create command to create a resource group that's named *tailspin-space-game-rg*.

**Azure CLI**

az group create --name tailspin-space-game-rg

1. Run the following commands to create two App Service plans.

**Azure CLI**

az appservice plan create \

--name tailspin-space-game-test-asp \

--resource-group tailspin-space-game-rg \

--sku B1

az appservice plan create \

--name tailspin-space-game-prod-asp \

--resource-group tailspin-space-game-rg \

--sku P1V2

**Important**

If the **B1** SKU isn't available as part of your Azure subscription, then **choose a different plan**, such as **S1** (**Standard**).

Recall that an App Service plan defines the CPU, memory, and storage resources that are provided for your web app to run.

The first command specifies the **B1** plan. This plan runs on the **Basic** tier. The App Service environments use this plan for the *Dev* and *Test* stages. The **B1 Basic** plan is intended for apps that have low-traffic requirements, such as in a development or test environment.

The second command specifies the **P1V2** plan, which runs on the **Premium** tier. This plan is used for *Staging*. The **Premium P1V2** plan is for production workloads. It runs on dedicated virtual machine instances.

The **Basic** plan tier offers no additional deployment slots. However, the **Premium** plan does. That's why you need the **Premium** plan for your *Staging* environment.

1. Run the following az webapp create commands to create the three App Service instances, one for each of the *Dev*, *Test*, and *Staging* environments.

**Azure CLI**

az webapp create \

--name tailspin-space-game-web-dev-$webappsuffix \

--resource-group tailspin-space-game-rg \

--plan tailspin-space-game-test-asp

az webapp create \

--name tailspin-space-game-web-test-$webappsuffix \

--resource-group tailspin-space-game-rg \

--plan tailspin-space-game-test-asp

az webapp create \

--name tailspin-space-game-web-staging-$webappsuffix \

--resource-group tailspin-space-game-rg \

--plan tailspin-space-game-prod-asp

Notice that here you apply the same App Service plan, **B1 Basic**, to the App Service instances for *Dev* and *Test*. You apply the App Service plan **Premium P1V2** to the App Service instance for *Staging*.

1. Run the following az webapp list command to list the host name and state of each App Service instance.

**Azure CLI**

az webapp list \

--resource-group tailspin-space-game-rg \

--query "[].{hostName: defaultHostName, state: state}" \

--output table

Note the host name for each running service. You'll need these host names later when you verify your work. Here's an example of the host names:

**Output**

HostName State

------------------------------------------------------- -------

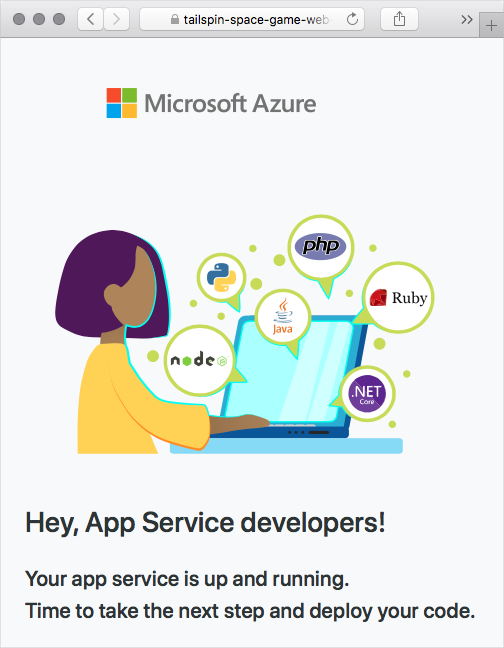
tailspin-space-game-web-dev-21017.azurewebsites.net Running

tailspin-space-game-web-test-21017.azurewebsites.net Running

tailspin-space-game-web-staging-21017.azurewebsites.net Running

1. As an optional step, go to one or more of the names. Verify that they're running and that the default home page appears.

You see this page in each environment:



**Important**

The **Clean up your Azure DevOps environment** page in this module contains important cleanup steps. Cleaning up helps ensure that you're not charged for Azure resources after you complete this module. Be sure to perform the cleanup steps even if you don't complete this module.

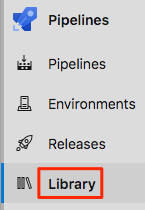
**Create pipeline variables in Azure Pipelines**

In Create a multistage pipeline by using Azure Pipelines , you added one variable for each of the App Service instances that correspond to the *Dev*, *Test*, and *Staging* stages in your pipeline. Here you do the same.

Each stage in your pipeline configuration uses these variables to identify which App Service instance to deploy to.

To add the variables:

1. In Azure DevOps, go to your **Space Game - web - Deployment patterns** project.
2. Under **Pipelines**, select **Library**.

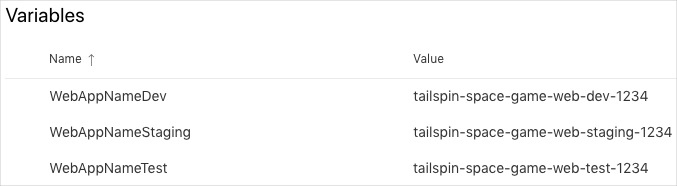


1. Select **+ Variable group**.
2. Under **Properties**, for the variable group name, enter *Release*.
3. Under **Variables**, select **+ Add**.
4. For the name of your variable, enter *WebAppNameDev*. As its value, enter the name of the App Service instance that corresponds to your *Dev* environment, such as *tailspin-space-game-web-dev-1234*.
5. Repeat steps 5 and 6 twice more to create variables for your *Test* and *Staging* environments, as shown in this table:

|  |  |
| --- | --- |
|  | |
| Variable name | **Example value** |
| *WebAppNameTest* | *tailspin-space-game-web-test-1234* |
| *WebAppNameStaging* | *tailspin-space-game-web-staging-1234* |

1. Be sure to replace each example value with the App Service instance that corresponds to your environment.
2. **Important**
3. Set the name of the App Service instance, not its host name. In this example, you would enter *tailspin-space-game-web-dev-1234* and not *tailspin-space-game-web-dev-1234.azurewebsites.net*.
4. Near the top of the page, select **Save** to save your variable to the pipeline.

Your variable group resembles this one:



**Create a service connection**

Here you create a service connection that enables Azure Pipelines to access your Azure subscription. Azure Pipelines uses this service connection to deploy the website to App Service. You created a similar service connection in the previous module.

**Important**

Make sure that you're signed in to both the Azure portal and Azure DevOps under the same Microsoft account.

1. In Azure DevOps, go to your **Space Game - web - Deployment patterns** project.
2. From the bottom corner of the page, select **Project settings**.
3. Under **Pipelines**, select **Service connections**.
4. Select **New service connection**, then choose **Azure Resource Manager**, then select **Next**.
5. Near the top of the page, **Service principal (automatic)**. Then select **Next**.
6. Fill in these fields:

| **TABLE 2** | |
| --- | --- |
| **Field** | **Value** |
| Scope level | **Subscription** |
| Subscription | Your Azure subscription |
| Resource Group | **tailspin-space-game-rg** |
| Service connection name | *Resource Manager - Tailspin - Space Game* |

1. During the process, you might be prompted to sign in to your Microsoft account.
2. Ensure that **Grant access permission to all pipelines** is selected.
3. Select **Save**.

Azure DevOps performs a test connection to verify that it can connect to your Azure subscription. If Azure DevOps can't connect, you have the chance to sign in a second time.

**Fetch the branch from GitHub**

Here you fetch the blue-green branch from GitHub. Then you *check out*, or switch to, that branch.

This branch contains the *Space Game* project that you worked with in the previous modules and an Azure Pipelines configuration to start with.

1. In Visual Studio Code, open the integrated terminal.
2. Run the following git commands to fetch a branch named blue-green from the Microsoft repository and to switch to that branch.

**Bash**

**git fetch upstream blue-green**

**git checkout -b blue-green upstream/blue-green**

The format of these commands enables you to get starter code from the Microsoft GitHub repository, known as upstream. Shortly, you'll push this branch up to your GitHub repository, known as origin.

1. As an optional step, open *azure-pipelines.yml* from Visual Studio Code. Familiarize yourself with the initial configuration.

The configuration resembles the ones that you created in the previous modules in this learning path. It builds only the application's **Release** configuration. For brevity, it omits the triggers, manual approvals, and tests that you set up in previous modules.

For learning purposes, this configuration promotes changes from any branch to *Dev*, *Test*, and *Staging*. A more robust approach might promote changes from only a release branch or master. You used this robust approach in the Create a multistage pipeline by using Azure Pipelines  module.

**Run the pipeline and see the deployed website**

Here you push the initial configuration to GitHub so that you're up to date with the team. Later, you'll add a slot to the *Staging* environment so that you can implement a blue-green deployment.

1. In Visual Studio Code, open the integrated terminal.
2. Run the following git commit command to add an empty entry to your commit history.

**Bash**

**git commit --allow-empty -m "Trigger the pipeline"**

This step is for learning purposes. It isn't typical. We provide starter code that you don't need to modify now. The --allow-empty flag ensures that the next step successfully pushes the branch to GitHub and triggers Azure Pipelines to run.

If you omitted this step, then the git push command that you run in the next step wouldn't take any action. So it wouldn't trigger Azure Pipelines to run.

1. Run the following git push command to upload the branch to your GitHub repository.

**Bash**

**git push origin blue-green**

1. In Azure Pipelines, go to the build. Then trace the build as it runs.
2. After the build finishes, go to the summary page.



You see that each stage of the pipeline succeeded.

1. Go to the URL that corresponds to each stage.

You see that the *Space Game* website successfully deployed to each App Service environment.

