# Introduction to Continuous Delivery

# Introduction

In the Build applications with Azure DevOps  learning path, you helped the Tailspin Toys team use Microsoft Azure DevOps to plan and build a continuous integration pipeline for the Space Game website.

The Tailspin team's big release is approaching. The team can use Azure DevOps to build and test their code. But how can they quickly deploy the application to an environment that's available to their users?

In this module, you'll continue your journey with the Tailspin team as they set up a continuous delivery (CD) pipeline for the Space Game website.

## Learning objectives

In this module, you'll:

* Learn what continuous delivery is, why it's important, and what tools you can use.
* Create a basic release pipeline that deploys a web application to Azure App Service.
* Examine pipeline analytics to understand the health and history of your releases.

## Prerequisites

The modules in this learning path and previous learning paths form a progression.

To follow the progression from the beginning, be sure to first complete these learning paths:

* Evolve your DevOps practices
* Build applications with Azure DevOps

If you want to start with this learning path, set up a development environment on your Windows, macOS, or Linux system. You need:

* An Azure DevOps organization .
* An Azure subscription .
* A GitHub  account.
* Visual Studio Code .
* .NET Core 3.1 SDK .
* Git .

You can get started with Azure and Azure DevOps for free. You don't need an Azure subscription to work with Azure DevOps. But here you'll use Azure DevOps to deploy to Azure resources in your Azure subscription.

You'll use this environment to complete the exercises in this module and future modules. You can also use this environment to apply your new skills to your own projects.

**Note**

Keep in mind that you can use Azure DevOps to build and deploy almost any kind of application written in any language. In this module, you'll be working with a .NET Core application written in C#.

You don't need to be an expert in .NET or C# to complete this module. You can apply the patterns you learn here to your own projects that use your favorite programming languages and frameworks.

## Meet the team

You met the Space Game web team at Tailspin Toys in previous modules. As a refresher, here's who you'll work with in this module.



Andy is the development lead.



Amita is in QA.



Tim is in operations.



Mara just joined as a developer and reports to Andy.



Irwin is the product manager.

Mara has prior experience with DevOps. She's helping the team adopt a more streamlined process by using Azure DevOps.

# What is continuous delivery?

Here you follow the Tailspin team as they discuss how a continuous delivery (CD) pipeline can help them with their upcoming release.

The Tailspin team is starting to feel better about their build process. They have an automated process running on Azure Pipelines, which means the build environment is stable. Amita knows immediately when she needs to test an artifact. She finds fewer bugs because Andy and Mara have started to add unit tests and code quality tests. Life is looking good. Let's check in with the team.

## Morning meeting

The team is in the meeting room waiting for Irwin, the product manager, who says he wants to talk to them. They look forward to telling him about their progress. But when Irwin walks in, he doesn't look happy. He starts talking right away.

**Irwin:** I had a meeting this morning with the management team. They want to know why we're taking so long to release our games and websites. Our closest competitors get new features and new games out there much faster than we do. We need to speed things up. I'm not alerting just you; I'm alerting all the teams. What can we do to help your team deploy faster?

Irwin looks around. Andy clears his throat.

**Andy:** This is a little sudden, but we're a bit ahead of you. We've been automating how we build our websites. Maybe now it's time to extend our automation to our release process.

**Irwin:** How would you do that?

**Mara:** We created an automated build pipeline by using Azure Pipelines. It builds an artifact that Amita can test. We could also build a continuous delivery, or CD, pipeline.

**Irwin:** What's a CD pipeline?

Mara begins to explain but is interrupted when Irwin's cell phone beeps. Irwin reads a text message and mutters under his breath.

**Irwin:** I'm sorry, but this is urgent. I have to go. Why don't you all figure out this CD business and get back to me soon?

Andy looks around at his team.

**Andy:** Coffee?

Andy and the rest of the team head to the coffee shop to create a plan.

## How will the team deploy faster?

The team is meeting over coffee. Irwin, the product manager, has told them that they need to start releasing new versions of the website faster. He has left it up to them to figure out how to do it.

**Andy:** So, as I told Irwin just before he ran off, I think it's time to build a CD pipeline. Automation is the only way to release faster.

Mara nods. Amita and Tim look unhappy.

**Tim:** I don't really know what CD is, but if it means losing control of my production environment, I won't do it.

**Amita:** If it means all the tests are automated, what am I supposed to do? It's not only about my job. I don't think we should sacrifice the user experience just to meet some arbitrary deadline.

Amita starts crumbling her scone into little pieces.

**Andy:** Tim, you won't lose control over your production environment. And Amita, you're correct, we should never sacrifice the user experience. Let's talk a bit about what CD is so we all understand each other. Mara, want to try?

**Mara:** Sure.

## What is continuous delivery?

**Mara:** To me, CD and DevOps are inseparable. Remember that we defined DevOps as the union of people, process, and products to enable continuous delivery of value to our end users.

CD by itself is a set of processes, tools, and techniques that enable rapid, reliable, and continuous delivery of software. So CD isn't only about setting up a pipeline, although that part is important. CD is about setting up a working environment where:

* We have a reliable and repeatable process for releasing and deploying software.
* We automate as much as possible.
* We don't put off doing something that's difficult or painful. Instead, we do it more often so that we figure out how to make it routine.
* We keep everything in source control.
* We all agree that done means released.
* We build quality into the process. Quality is never an afterthought.
* We're all responsible for the release process. We no longer work in silos.
* We always try to improve.

We've already put many of these ideas into place, and we all agree they've improved how we work. CD is an extension of what we've already started.

## Why do I need continuous delivery?

CD helps software teams deliver reliable software updates to their customers at a rapid cadence. CD also helps ensure that both customers and stakeholders have the latest features and fixes quickly.

Let's continue to listen in on the team as they talk this out.

**Andy:** Thanks, Mara. We need CD because, as we all know, the world has changed. New features are being released faster. Updates and bug fixes need to be available right away. It isn't just our management that wants to speed up our releases. Management is simply reacting to the demands of our customers. If customers can't get what they want from us, they'll go somewhere else.

The old way of doing things, where we all worked in our separate silos and didn't talk to each other, is too slow. And honestly, it's frustrating for all of us.

We need to work together if we're going to succeed. All of us have seen how automation gives us a reliable, repeatable, and faster process to build artifacts. CD will bring those benefits to an entire release pipeline.

Amita, I think your life has gotten better since we added tests to the build pipeline. You aren't getting regression bugs. You're also getting automatic updates that tell you when builds are ready.

Tim, I'm going to ask you to be patient. I know you haven't seen much benefit yet, but I promise you, you will. When you're home on the weekends instead of here because we keep crashing your production environment, you'll be happier. Can I ask you both to give CD a chance?

Amita and Tim look at each other and nod.

**Tim:** Let's get started.

**Andy:** Thanks, everyone. I'm going to propose that Mara and I put together a simple proof of concept (POC). I think everything will be a lot easier to understand if you can see a CD pipeline in action.

Tim laughs.

**Tim:** Don't let me stop you from doing all the work!

**Amita:** Good luck, you two.

The team leaves Andy and Mara to work out the details.

## How does continuous delivery compare to right-click publishing?

Many development tools provide ways to publish your application directly to some target environment, such as Microsoft Internet Information Services (IIS) or Azure. For example, you can Publish an ASP.NET Core app to Azure by using Visual Studio . This process is sometimes called right-click publishing.

Right-click publishing is a great way to quickly build a prototype. For example, you might right-click publish your application to Azure so that you can share a new idea with your team. But this technique has limitations.

Continuous delivery provides a consistent way for you and your team to continuously test, deploy, and monitor your application each time you check in your code. When you right-click publish your application, there's no guarantee that the code was properly tested or will behave as expected under real-world usage.

## How does continuous delivery compare to continuous deployment?

In the DevOps community, you might hear the terms continuous delivery and continuous deployment.

## What continuous delivery tools can I use?

After the meeting ends, Andy and Mara plan next steps. They use Azure Pipelines to build their software. They want to consider what tools, including Azure Pipelines, are available to help them with their release process.

**Mara:** Where do you want to start?

**Andy:** First we need to agree on our release management tool. Let's make sure the tool we choose:

* Supports our version control system.
* Can deploy to multiple environments so we can test and validate our work.
* Makes it easy to define our deployment tasks.
* Is easy to extend.

**Mara:** Azure DevOps integrates with several other continuous integration (CI) and CD solutions. Many solutions are out there, and we're not invested in any of them. If we were, it would make sense to use that one. Popular CI and CD systems include Jenkins, Circle CI, GitLab, Travis CI, and Azure Pipelines.

These tools have similarities, but each also has its particular strengths. Some of these tools are open source, some are free, and some you have to pay for. They each also provide built-in integrations with other software tools.

For example, Jenkins is open source. It has many plug-ins, and many companies use it. You can run Circle CI in the cloud or on-premises. I think we would need to customize it. GitLab is a single application for the entire software development life cycle. It might be bigger than we want right now. We can keep using Azure Pipelines.

**Mara:** My vote is to stay with Azure Pipelines.

**Andy:** I agree. Azure Pipelines has worked great for us so far, and we don't have to learn another new technology.

**Mara:** Great. Let's get started on the pipeline details.

Andy and Mara move to a conference room to plan their CD pipeline.

# Plan a release pipeline by using Azure Pipelines

In this section, you follow along with Andy and Mara as they plan a basic CD pipeline that runs on Azure Pipelines.

When it's done, they'll demo it to the rest of the team. The pipeline will serve as a POC that they'll improve on and expand as they learn more and get feedback from Tim and Amita.

Andy and Mara are sitting in a conference room, ready to get started.

**Andy:** I've never built a CD pipeline before. Have you?

**Mara:** I've worked with a few of the CD pipeline tools we discussed earlier, but I haven't set one up.

**Andy:** Because we're both new to this process, maybe we should start with the automated pipeline we already have: our build pipeline.

**Mara:** Great minds think alike. We could probably extend our existing build configuration. But how do we turn our build pipeline into a CD pipeline?

## What are the parts of a basic CD pipeline?

A basic CD pipeline contains a trigger to get the process going and at least one stage, or deployment phase. A stage is made up of jobs. A job is a series of steps that defines how to build, test, or deploy your software.

Let's follow along with Andy and Mara as they plan their POC.

**Andy:** Here's what I think we need to start.

Andy draws a diagram on the whiteboard.

**Andy:** We already have the  build artifact. It's the .zip file that our existing build pipeline creates. But how do we deploy it to a  live environment?

**Mara:** In the CD pipelines I worked with, we defined the deployment process in stages, such as building the artifact and deploying the artifact to the various testing and production environments. Each stage breaks down into one or more jobs. Each job breaks down into tasks, just like the ones we use in our existing build pipeline.

## What is a pipeline stage?

A stage is a part of the pipeline that can run independently and be triggered by different mechanisms. A mechanism might be the success of the previous stage, a schedule, or even a manual trigger. You'll learn more about these mechanisms in the next module.

**Mara:** I think that deciding on our stages is a great start and pretty straightforward. Let's define a stage as a major division in a pipeline. Every stage is independent of every other stage. We could have a stage that builds the app and another stage that runs tests. There are many possibilities. Because we want to keep it simple, how about we start with two stages?

Mara updates the diagram on the whiteboard.

**Mara:** We've already defined the tasks for the  build stage in our pipeline. Our  deployment stage can be similar, including tasks that deploy the build to an environment.

The question is, where should we deploy the artifact?

## What is an environment?

You've likely used the term environment to refer to where your application or service is running. For example, your production environment might be where your end users access your application.

Following this example, your production environment might be:

* A physical machine or virtual machine (VM).
* A containerized environment, such as Kubernetes.
* A managed service, such as Azure App Service.
* A serverless environment, such as Azure Functions.

An artifact is deployed to an environment. Azure Pipelines makes it easy to deploy to almost any kind of environment, whether it's on-premises or in the cloud.

In Azure Pipelines, the term environment has a second meaning. Here, an environment is an abstract representation of your deployment environment, such as a Kubernetes cluster, an App Service instance, or a virtual machine.

An Azure Pipelines environment records the deployment history to help you identify the source of changes. By using Azure Pipelines environments, you can also define security checks and ways to control how an artifact is promoted from one stage of a pipeline to another. What an environment includes depends on what you want to do with the artifact. An environment where you want to test the artifact will probably be defined differently than one where you want to deploy the artifact for your end users.

One way to define an Azure Pipelines environment is with a YAML file. Your YAML file includes an environment section, which specifies the Azure Pipelines environment where you'll deploy your artifact.

As you plan your release pipeline, you'll need to decide where your application or service will run. Let's listen in and see what Andy and Mara decide.

**Andy:** At a high level, what type of environment do we want? Do we want to deploy on-premises or to the cloud?

**Mara:** We could ask Tim to create a VM for us in the lab. But he's always running out of hardware. It'll be fast and easy to set up a POC ourselves if we use the cloud.

**Andy:** I agree. But there are so many cloud options to consider, and we can use Azure Pipelines to deploy to any of them. Which should we try?

**Mara:** The teams that develop our games use Azure to host some of their back-end systems. They set it up quickly and seem to like it. I think we should stick with Azure for our cloud.

**Andy:** OK. That makes sense! But Azure provides so many compute options. Which should we pick?

Andy lists these options on the whiteboard:

* Virtual machines
* Containers
* Azure App Service
* Serverless computing

**Note**

You'll find more information on each of these compute options at the end of this module.

**Mara:** I know containers and serverless computing are popular right now. Compared to VMs, they're both lightweight in terms of resources. They're also easy to replace and scale out. Both are interesting, but I'm nervous about learning two new technologies at the same time. I'd rather concentrate just on building the pipeline.

**Andy:** I'm with you. That leaves VMs or App Service. I think VMs would be a better choice if we were moving a line-of-business app, one that requires full access to some particular environment, to the cloud. We're not doing anything that significant.

**Mara:** That leaves App Service, which would be my choice. It's designed to work with Azure DevOps, and it comes with advantages. It's a platform-as-a-service (PaaS) environment for web apps, so it takes a lot of the burden off of us. We won't have to worry about infrastructure. It also comes with security features and lets us perform load balancing and automatic scaling.

**Andy:** App Service sounds like what we need. Let's use App Service. We're creating just a proof of concept anyway. We can always change the compute option if we want to try something else later.

## How does Azure Pipelines perform deployment steps?

To deploy your software, Azure Pipelines first needs to authenticate with the target environment. Azure Pipelines provides different authentication mechanisms. The one you use depends on the target environment you're deploying to. You'll find more information about these mechanisms at the end of this module.

**Andy:** We have our build artifact, and we know we'll build and deploy in stages of the pipeline. We've also defined the target environment for our deployment. That's App Service. My question now is, how does Azure Pipelines authenticate with App Service? I know this will be one of Tim's concerns. We need to ensure the process is secure.

After a bit of research, Andy and Mara come up with the general steps that allow Azure Pipelines to deploy to App Service:

1. Specify the target deployment environment in the pipeline configuration.
2. Provide a way for Azure Pipelines to authenticate access to that environment.
3. Use Azure Pipelines tasks to deploy the build artifact to that environment.

**Mara:** According to our research, we need to create a service connection to specify the target environment and authenticate access to it. After we define the service connection, it will be available for all of our tasks to use. Then we need to use the built-in tasks DownloadPipelineArtifact@2  and AzureWebApp@1 .

**Andy:** I have the tasks here. Let's see what they do for us.

### Download pipeline artifacts

In the previous learning path, you published build artifacts to the pipeline. These artifacts were .zip files that contained the Space Game website as a build package. Amita manually downloaded this package and installed it in her test environment.

To deploy a build artifact from the pipeline, you need a way to download it from the pipeline to the agent. You use the DownloadPipelineArtifact@2 task to download artifacts.

This task requires a few inputs. The ones we need here are:

* buildType, which specifies whether we want the artifacts from the current build or a specific build. For now, we want to deploy the current build.
* artifact, which specifies the name of the artifact to download. We need this input to specify the name of the .zip file.

This example downloads the latest package named "drop" from the pipeline to the build agent:

ymlCopy

- task: DownloadPipelineArtifact@2

inputs:

buildType: current

artifact: 'drop'

The download task is a shortcut for the DownloadPipelineArtifact@2 task. Here's an example that uses the download task to download the same artifact from the pipeline:

ymlCopy

- download: current

artifact: drop

### Deploy a web application to App Service

You use the AzureWebApp@1 task to deploy a web application to App Service. This task works with several programming languages and frameworks, including ASP.NET, ASP.NET Core, PHP, Java, Python, Go, and Node.js. We use this task to do the deployment. To use it, though, we have to have App Service running on our Azure subscription.

This task also requires a few inputs:

* azureSubscription is the service connection we talked about earlier. We need this input to authenticate with the target environment.
* appName specifies the name of our App Service instance.
* package specifies where on the build agent to find the package to deploy.

This example uses AzureWebApp@1 to deploy the contents of MyPackage.zip to an App Service instance named "MyWebApp":

ymlCopy

- task: AzureWebApp@1

inputs:

azureSubscription: 'MyServiceConnection'

appName: 'MyWebApp'

package: '\*\*/MyPackage.zip'

The azureSubscription part specifies the service connection that's named MyServiceConnection. You'll work with service connections shortly.

### What are jobs and strategies?

Your existing build pipeline defines a build agent, pipeline variables, and the tasks needed to build your software.

The deployment part of your pipeline contains these same elements. Your deployment configuration typically also defines one or more jobs, a pipeline environment, and strategies. You learned about pipeline environments earlier.

Here's an example configuration that you'll run later in this module. This configuration deploys the Space Game website to App Service.

ymlCopy

- stage: 'DeployDev'

displayName: 'Deploy to dev environment'

dependsOn: Build

jobs:

- deployment: Deploy

pool:

vmImage: 'ubuntu-18.04'

environment: dev

variables:

- group: 'Release Pipeline'

strategy:

runOnce:

deploy:

steps:

- download: current

artifact: drop

- task: AzureWebApp@1

displayName: 'Azure App Service Deploy: website'

inputs:

azureSubscription: 'Resource Manager - Tailspin - Space Game'

appName: '$(WebAppName)'

package: '$(Pipeline.Workspace)/drop/$(buildConfiguration)/\*.zip'

#### Jobs

A job is a series of steps, or tasks, that run sequentially as a unit. Every pipeline stage has one job by default, even when that stage doesn't use the job keyword.

A job can run in an agent pool, on a container, or directly on the Azure DevOps server. The example job shown here runs on a Microsoft-hosted Ubuntu agent.

You can specify the conditions under which each job runs. The example job shown here doesn't define any conditions. By default, a job runs if it doesn't depend on any other job or if all of the jobs that it does depend on have finished successfully.

You can also run jobs in parallel or sequentially. Using your existing build pipeline as an example, you can use parallel jobs to build your software on Windows, Linux, and macOS agents simultaneously.

A deployment job is a special type of job that plays an important role in your deployment stages. Deployment jobs record the status of your deployments in Azure Pipelines, providing you with an audit trail. Deployment jobs also help you define your deployment strategy, which we'll do shortly.

#### Strategies

A strategy defines how your application is rolled out. You'll learn more about strategies such as blue-green and canary in a future module. For now, you'll use the runOnce strategy to download the Space Game package from the pipeline and deploy it to App Service.

### How does Azure Pipelines connect to Azure?

To deploy your app to an Azure resource, such as a virtual machine or App Service, you need a service connection. A service connection provides secure access to your Azure subscription by using one of two methods:

* Service principal authentication
* Managed identities for Azure resources

You can learn more about these security models at the end of this module, but in short:

* A service principal is an identity with a limited role that can access Azure resources. Think of a service principal as a service account that can do automated tasks on your behalf.
* Managed identities for Azure resources are a feature of Azure Active Directory (Azure AD). Managed identities simplify the process of working with service principals. Because managed identities exist on the Azure AD tenant, Azure infrastructure can automatically authenticate the service and manage the account for you.

Managed identities simplify the process of working with service principals. But in this module you use service principal authentication because a service connection can automatically discover your Azure resources and assign the appropriate service principal roles for you.

## The plan

Andy and Mara are ready to begin. They're going to:

* Build on their existing Azure Pipelines build configuration.
* Define a build stage that creates the artifact.
* Define a deployment stage that deploys the artifact to App Service.

**Andy:** Is this drawing correct? We use  Azure Pipelines to deploy to  App Service. To do that, we take the  build artifact as the input to the  deployment stage. The tasks in the deployment stage  download the artifact and use a service connection to  deploy the artifact to App Service.

**Mara:** That about sums it up. Let's get started.