**Introduction**

* 1 minute

In [Create a release pipeline with Azure Pipelines](https://docs.microsoft.com/en-us/learn/modules/create-release-pipeline), you built a basic release pipeline that deploys an ASP.NET Core application to Azure App Service. Although this process covers the needs for basic applications, modern solutions often require a combination of applications, services, and other components.

Azure provides great support for traditional app deployments, such as to virtual machines or application services. Another option you might consider is to deploy your app by using a container. A *container* app is one that's packaged and published as a single artifact that can be deployed with all of its dependencies to run in an isolated environment.

In this module, you'll join the Tailspin Toys web team as they explore one way to use containers in Azure. You'll learn how to update your CI/CD pipeline to build a Docker container, publish the container to Azure Container Registry, and deploy the container to App Service.

While this module focuses on the core tasks that are required to build and deploy your container app, it's important to understand that all of the other features of Azure Pipelines are still available for Docker container deployments. You can integrate testing, define multiple stages, and perform other tasks just like you would for your existing applications. We omit these tasks here to keep things focused.

**Learning objectives**

In this module, you'll:

* Create Azure resources to support a Docker container web application.
* Update a basic release pipeline to build, publish, and deploy a Docker container.
* Monitor the build, release, and deployment of the project.

**Prerequisites**

The modules in this learning path form a progression. To follow the progression from the beginning, be sure to first complete these learning paths:

* [Evolve your DevOps practices](https://docs.microsoft.com/en-us/learn/paths/evolve-your-devops-practices)
* [Build applications with Azure DevOps](https://docs.microsoft.com/en-us/learn/paths/build-applications-with-azure-devops)

We also recommend you start at the beginning of this learning path: [Deploy applications with Azure DevOps](https://docs.microsoft.com/en-us/learn/paths/deploy-applications-with-azure-devops). This module picks up where [Create a release pipeline in Azure Pipelines](https://docs.microsoft.com/en-us/learn/modules/create-release-pipeline/) leaves off.

**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.

This module also assumes you have basic familiarity with Docker, although that knowledge is not required to complete it. If you are new to the topic, it is recommended that you complete the [Introduction to Docker containers](https://docs.microsoft.com/en-us/learn/modules/intro-to-docker-containers/) module first.

**Why are containers important?**

* 7 minutes

In this part, you follow the Tailspin team as they discuss some much-needed improvements to their DevOps process. In this scenario, the team uses Docker to containerize their web application. The team then updates their CI/CD pipeline to support it.

**It's been a few rough weeks**

The past few weeks have been a challenging time at Tailspin. Teams struggle to meet deadlines for a number of reasons, and there has been concern over productivity across the company. Andy has called some key stakeholders from the Space Game website team together to gather feedback for an upcoming presentation to management.

**Andy:** Thanks for stopping by. I know the last few weeks have been rough for everyone, but I have some good news. Management is holding an offsite tomorrow to hear proposals on changes we can make to improve performance. They've invited me to present a case study on our DevOps successes and said they're also open to any other ideas we might have. I was hoping we could use this meeting as an opportunity to brainstorm. Who wants to go first?

*Everyone looks at Amita. She has been especially frustrated lately.*

**Amita:** I'll go first. As you know, I test for multiple teams, and it can be challenging since each team uses their own technology stack. And even when they use the same underlying platforms, like .NET or Java, they often target different versions. I feel like I sometimes spend half of my day simply getting test environments in a state where they can run the code I need to test. When something doesn't work, it's hard to tell whether there's a bug in the code or if I accidentally configured version 4.2.3 instead of 4.3.2.

*Andy writes "Dependency versioning challenges for QA" on the whiteboard.*

**Tim:** I'd like to add operations to that frustration. We have a few teams that have unique version requirements, so we have to publish their apps on their own virtual machines just to make sure their version and component requirements don't conflict with our other apps. Besides the overhead involved in maintaining the extra set of VMs, it also costs us more than it would if those apps could run side by side.

*Andy writes "Overhead due to solving app isolation with VMs" on the whiteboard.*

**Mara:** I have something from the development side. A few weeks ago I was working on the peer-to-peer update system and had it all working on my machine. But when I handed it off for deployment, it didn't work in production. I had forgotten that I needed to open port 315 as part of the service. It took us over a day of troubleshooting to realize what was going on. Once we opened that up in production, things worked as expected.

*Andy writes "Configuration inconsistencies between deployment stages" on the whiteboard.*

**Andy:** I think this conversation is a good start. Let me research these issues and see what I can come up with. Here are the concerns that I heard:

* Dependency versioning challenges for QA
* Overhead due to solving app isolation with VMs
* Configuration inconsistencies between deployment stages

**Putting it all together (in one container)**

The next morning, Andy calls a meeting to present a new idea to the team.

**Andy:** I spoke with some colleagues yesterday about the challenges we're facing, and they made some interesting suggestions. The one I'm excited to try out is Docker. It's a technology for packaging entire applications as containers.

**Amita:** What's a container? Is that like a *.zip* file?

**Andy:** Not exactly. It's more like a lightweight virtual machine designed to run directly on the host operating system. When you build your project, the output is a container that includes your software along with its dependencies. However, it's not a complete virtualized system, so it can spin up in a little as less than one second.

**Tim:** How does it handle security and isolation?

**Andy:** Security and isolation are handled by the host operating system. When your container runs in a host process, the container is isolated from the other processes on that same host machine. This isolation allows your container to load whatever versions of components it needs, regardless of what other containers are doing. It also means you can easily run multiple containers on the same host simultaneously.

**Amita:** That sounds great for the production environment, but does it solve the challenges we're facing earlier in the pipeline?

**Andy:** Absolutely! Instead of shipping source code or a set of binaries, the entire container becomes the artifact. That means that when Mara is developing, her debugging sessions run locally against the container hosted on her machine. When Amita tests, she tests against a copy of that same container, which already includes all the required versions of its dependencies. When Tim manages our production environment, the containers he monitors are standalone copies of the same containers developed by Mara and tested by Amita.

**Mara:** How hard is it to develop a container application? Do we have to make significant changes to our existing code?

**Andy:** Containers are more of a packaging and deployment technology. They don't impact the fundamental software we're writing. We can just instruct our tools to produce a Docker container at the end of the build. Then, when we debug, the application will run out of that local container instead of our local web server. In fact, tools like Visual Studio even let you switch between debug environments like Docker and IIS Express to give us the flexibility we need. I actually forked our web site project last night and converted it to build as a Docker container to test out the process. I only needed to add some basic container configuration; I didn't need to change any of our existing code.

**Mara:** That's great to know. I bet we can even update the release pipeline in Azure Pipelines from your fork to build and deploy the Docker version.

**Andy:** You read my mind.

**What is Docker?**

Docker is a technology for automating the packaging and deployment of portable, self-sufficient containers. Docker containers can be run anywhere a Docker host is found, whether on a development machine, a departmental server, an enterprise datacenter, or in the cloud. Azure provides multiple ways to run container-based applications, including App Service or as part of clusters managed with orchestration technologies like Kubernetes.

The Tailspin team selected Docker containers for this scenario because it met all their needs:

* Dependency versioning challenges for QA

Applications are packaged as containers that bring the correct versions of their dependencies with them.

* Overhead due to solving app isolation with VMs

Many isolated containers can be run on the same host with benefits over virtual machines including faster startup time to greater resource efficiency.

* Configuration inconsistencies between DevOps stages

Containers ship with manifests that automate configuration requirements, such as which ports need to be exposed.

Adopting Docker containers can be a key step on the path towards a microservices architecture. We'll discuss that more later on.