**Introduction**

* 1 minute

In [Automate Docker container deployments with Azure Pipelines](https://docs.microsoft.com/en-us/learn/modules/deploy-docker), you updated a release pipeline to build and publish an ASP.NET Core app as a Docker container, which you then deployed to Azure App Service. The migration to a container-based project afforded the product team several key benefits. However, the adoption of container technologies can open new doors future innovations.

As development organizations scale, so do the complexities of the solutions they deliver. Over time, different products and services take increasing dependence on each other. This can result in different development and operations requirements for different components within a given application. Refactoring those components as their own distinct microservices can improve architectural efficiency. A *microservice* is a small, autonomous service designed to be self-contained and to fulfill a specific business capability. Containers provide a great technical foundation for building and deploying these services, but this leads to a new question: how do we *manage* all of these containers floating around?

This is where orchestration technologies like Kubernetes come in. Kubernetes is a platform for managing containerized workloads and services. It's a great option for organizations that have a growing number of containers that need to be deployed, integrated, and monitored in any environment.

In this module, you join the Tailspin Toys web team as they explore one way to use Kubernetes on Azure. You learn how to update a CI/CD pipeline to build, publish, and deploy multiple Docker containers to a Kubernetes cluster.

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 container deployments to Kubernetes. 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 will:

* Understand the benefits of Kubernetes and when to use it.
* Create Azure resources to support a Kubernetes cluster.
* Update a basic release pipeline to build, publish, and deploy multiple Docker containers to the Kubernetes cluster.
* 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 [Automate Docker container deployments with Azure Pipelines](https://docs.microsoft.com/en-us/learn/modules/deploy-docker) 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 Kubernetes, although that knowledge is not required to complete it. If you are new to the topic, we recommend that you complete the [Administer containers in Azure learning path](https://docs.microsoft.com/en-us/learn/paths/administer-containers-in-azure/) learning path first. You'll also find additional resources at this end of this module.

**Why is container orchestration important?**

* 7 minutes

Here you follow the Tailspin team as they discuss how to best deliver on new directive driven by management. The team discusses how Kubernetes can support their migration towards a microservices architecture.

**The future is smaller**

Things are looking up at Tailspin. At a recent management offsite, Andy presented his team's recent successes with Azure DevOps, which were well received. Andy also presented a demo of the team’s recent proof-of-concept project using Docker containers. These demonstrations led to a series of productive conversations on the technical future of the organization. The next day, Andy returns to share the news with the Space Game web team.

**Andy:** Things went really well at my offsite presentation yesterday. Management is impressed with the work we've done so far, and has issued us a special assignment.

**Tim:** Uh oh. I've been around long enough to see a trap like that coming from a mile away.

**Andy:** No, this is a great opportunity for us. Management loved our Docker container demo and wants us to explore adopting a microservices architecture.

**Amita**: Microservices? Like apps for phones and watches?

**Andy**: No, microservices are typical apps, like our web app. The main difference is that instead of building and deploying a single monolithic app, we refactor any components that would be better maintained and managed as autonomous services. We then build those services to be good at what they do and deploy them to operate independently.

**Tim**: I'm not sure I like the sound of that. I'm already dealing with so many services across our environments. I don't know if I want more on my plate.

**Andy**: That's an understandable concern. Fortunately, there are some great tools for managing a multitude of containers in a given environment. We have been asked to spike out a multi-container solution for our web app that is orchestrated using Kubernetes. They also want to know how it will impact our DevOps process.

**Mara**: I've been reading up on Kubernetes. Azure has great support for it through Azure Kubernetes Service, and I know there is pipeline support for it in Azure DevOps.

**Amita**: This process sounds like it's going to get complex. How will it impact testing?

**Mara**: It shouldn't be a significant change. Kubernetes offers a way for us to deploy to different namespaces. This enables us to partition our deployments so that we can have entire environments dedicated to testing versus production. And since they all run in the same cluster and use the same containers, the testing experience should offer what we expect to see in production.

**Amita**: Is it going to be hard to keep track of what environment is where?

**Mara**: No, we can use Azure DevOps environments to do all of that. You'll be able to find out where each service is and how it got there by using the portal. It's all automated through the pipeline, so there will be nothing we have to manually keep track of. The only concern I have now is how much impact it will have on our development experience to build this out.

**Andy**: The good news is that the impact is minimal. Assuming we have our projects set up to build Docker containers, all we need to deploy to Kubernetes are some manifest files that describe the services and their deployments.

**Mara**: Have you thought about what we will refactor out as the second container? I know there have been several teams asking us to make our leaderboard available through a web API.

**Andy**: I'm one step ahead of you. I forked the Docker project last night and refactored the leaderboard data functionality into its own miroservice. This leaves us with one container for the web site and another for a leaderboard API. Both containers are configured to have their own public endpoints that we can share with anyone who wants to use the site or API regardless of what technology stack their app uses. If the load grows substantially for either, we can scale its containers independently.

**Mara**: This project sounds awesome! Let's get started on updating the release pipeline.

**What is Kubernetes?**

Kubernetes is a technology for orchestrating multi-container deployments. It provides a framework for running distributed systems in a declarative, responsive fashion. It automatically applies and enforces your deployment patterns to ensure that containers are deployed and run as intended. It also offers support for specialized release cadences, such as those using canary deployments.

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

* Complexity of multi-container deployments

Kubernetes is designed, first and foremost, to automate the processes around deploying and maintaining container deployments.

* Consistency across environments and stages

Just as containers ensure a consistent deployment for the apps they contain, Kubernetes ensures a consistent deployment for the containers a cluster manages.

* Azure DevOps support

Azure DevOps offers first-class support for working with Kubernetes.

* Ease of development

The impact of Kubernetes on a source project is comparable to that of adding Docker support, which is minimal and limited to declarative configuration.

Adopting Kubernetes drastically simplifies the process of adopting a microservices architecture that makes use of multiple Docker containers.