**Introduction**

Do you want to improve your release process? Maybe you want to increase the frequency of your releases. Or maybe you want to increase the quality of your software. Perhaps you want to get rid of some manual processes and move toward a more automated approach. Whatever your goals, Azure DevOps can help.

**Learning objectives**

In this module, you will:

* Learn about value stream maps (VSMs)
* Use a VSM to get a sense of where the release process needs improvement
* Understand how the VSM gives you a good starting point to discuss your current DevOps practices

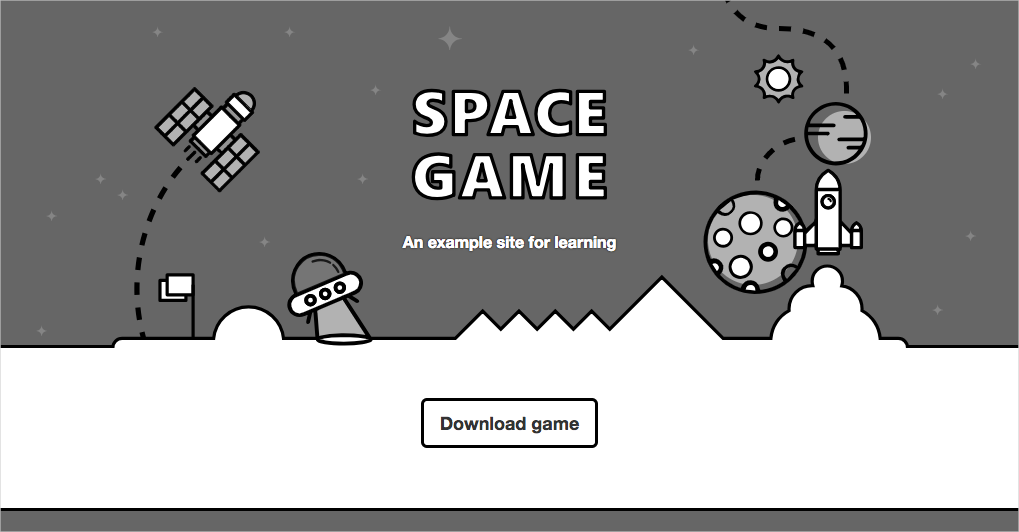
**Meet the team**

DevOps has many features and tools to help a team collaborate and improve its processes. Your journey through DevOps begins with an introduction to our fictitious software team members, who are discovering that they need to improve their release process.

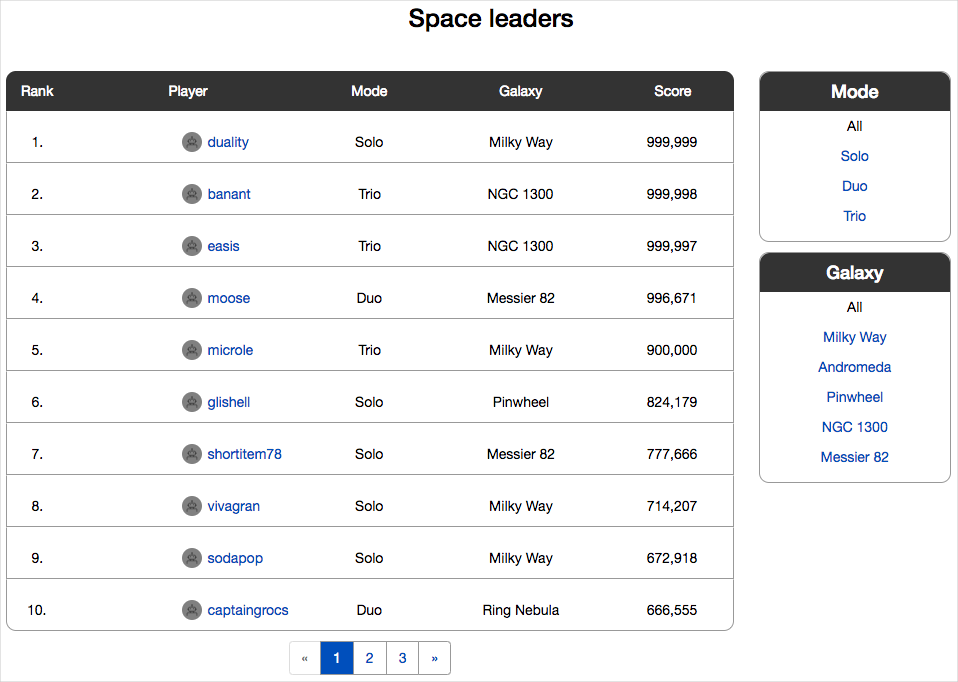
Tailspin Toys, or Tailspin for short, is a video game company. Tailspin hosts its game servers and websites in an on-premises datacenter. The company just celebrated the release of a new racing game. They'll be releasing a space shooter game, called *Space Game*, in the coming months.

The team that you'll be working with builds websites to support new game titles. These websites provide information about the game, ways to get it, and leaderboards that show top scores. Each website must go live the same day the game is released, which requires coordination among the teams and puts some extra pressure on the web team.

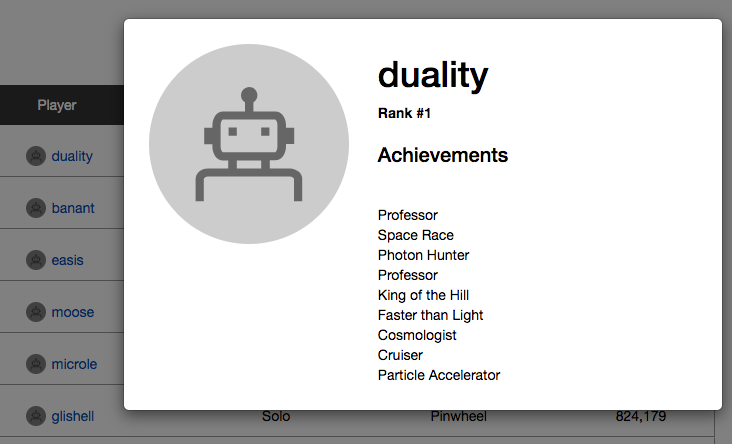
The *Space Game* website is a .NET Core app written in C# that's deployed to Linux. The website isn't finished yet, but here's what it looks like right now:



And here's what the leaderboard looks like:



You can filter the leaderboard by mode or by game map. You can also select a player's name to see their profile and game achievements:



**Note**

Although the game and the website aren't finished yet, you can **check out the website now** to get a sense of how it works.

Here are your team members:

Andy is the development lead who's been working with computers since he was a kid. He enjoys working on personal coding projects in his spare time. Andy always wishes he had more spare time.



Amita is in QA. She's calm, which helps with some temperamental developers. She's good at organizing and setting priorities and lives to find edge cases.



Tim is in operations. He likes practical solutions and he's very cautious (although some people might use the word "paranoid"), which makes sense because he's the person who gets the 3 A.M. call when something goes wrong.



Irwin is the product manager. He's been in the video game industry for decades. Irwin acts friendly towards the development teams, but everyone knows he favors a tight schedule over people. Irwin has a relatively fixed mindset, but if there's anything that can help teams get games to market faster with less effort, he's all ears.



Mara is new. She just joined Tailspin as a developer and reports to Andy. She joined Tailspin because she likes games and she thought a smaller company would have lots of opportunity for innovation. She's a big fan of DevOps.



**Good morning**

Irwin, the team's product manager, has called everyone into a meeting, and he's in a bad mood. The leaderboard for the racing game was just updated with several new features and he showed it at a local gaming group. Players' reactions were disappointing, to say the least. He reads off a list of the top problems:

* Some features work correctly for only some game modes.
* Updating the leaderboard takes too long, even with a small number of players.
* Multiple scores per player show up as multiple players.
* The new ranking feature returns incorrect results.
* There's no way to group scores according to a specific date or game session.
* It took months to produce the new release (and it's broken).

He demands, "How long before these problems are fixed?"

**Andy thinks**: I bet it takes me a month to write that code.

**Amita thinks**: It'll take me at least a week to test this and I can't start until Andy is finished, and he always wants to sneak in new code.

**Tim thinks**: It'll take me at least a week to set up the environments and deploy this to production. I can't start until Amita is finished, and she's never willing to call something a release candidate.

**Mara wonders**: Was taking this job a mistake?

Andy looks around at his teammates and says, "We'll get back to you."

The first step to setting up a DevOps practice is to assess your current process. This means analyzing:

* Your existing artifacts, such as deployment packages and NuGet, as well as your container repositories.
* Your existing test management tools.
* Your existing work management tools.
* Recommending migration and integration strategies.

Let's do that with the Tailspin team and see how DevOps can help.

After Irwin the product manager leaves, Amita says, "We need help. I don't know when these fixes are due, but I do know it's soon. We're not set up for a fast turnaround. Plus, the new *Space Game* website will have to wait until we get this mess solved — and that game is coming up fast."

Andy looks at Mara. "This is a lot to take in during your first few weeks."

"That's okay," Mara answers. "Maybe you can explain to me how things work around here. How does a game move from dev to production?"

"That's a great question," says Andy. I'm not sure we can give you a simple answer, but let's try."

The team decides to go to a coffee shop to relax and have an informal discussion. Together, they'll try to figure out why they're having so many problems.

Over coffee, Mara listens and tries to take notes. There's a lot of information and it's not organized. Her overall thoughts about the team are:

* They use a waterfall approach. Management sets the priorities. Developers write code and hand the build off to QA. QA tests and then hands off to ops for deployment.
* Waterfall could be acceptable for a small team, but here the goals aren't always clear and they seem to change frequently.
* Testing is delayed until late in the process. That means it's harder and more expensive to fix bugs and make changes.
* There's no clear definition of what "done" means. Each team member has their own idea. There's no overall business goal that everyone agrees on.
* Some code is in a centralized version-control system. Many tools and scripts exist only on network file shares.
* There are many manual processes.
* Communication is haphazard and depends on email, Word docs, and spreadsheets.
* Feedback is also infrequent and inconsistent.
* On the plus side, the team seems to get along, and they want to make things better.

When she looks at her pile of notes, Mara knows she needs to organize all this information. Organizing it will make it easier to evaluate the processes. She's convinced a DevOps approach will solve many of the team's problems, but she needs a way to present her case to the team.

A DevOps practice often begins with an understanding of your existing processes. From there, you can evaluate what's working well, what's not, and focus on what to fix first.



Mara asks, "Have any of you ever performed a *value stream mapping* exercise?" Andy rolls his eyes, Amita sighs, and Tim says, "We don't need more paperwork."

Mara says, "I get it. Leave it to me."

Glad to let the newbie handle it, everyone heads back to work.

# Assess process efficiency with value stream maps

* 8 minutes

Creating a value stream map, or VSM, helps you analyze your current release cycle process. The purpose of a VSM is to visually show where in the process a team creates value and where there's waste. The goal, of course, is to arrive at a process that delivers maximum value to the customer with minimum waste. A VSM can help you pinpoint those areas that either don't contribute any value or that actually reduce the value of the product.

Let's see how Tailspin measures up.

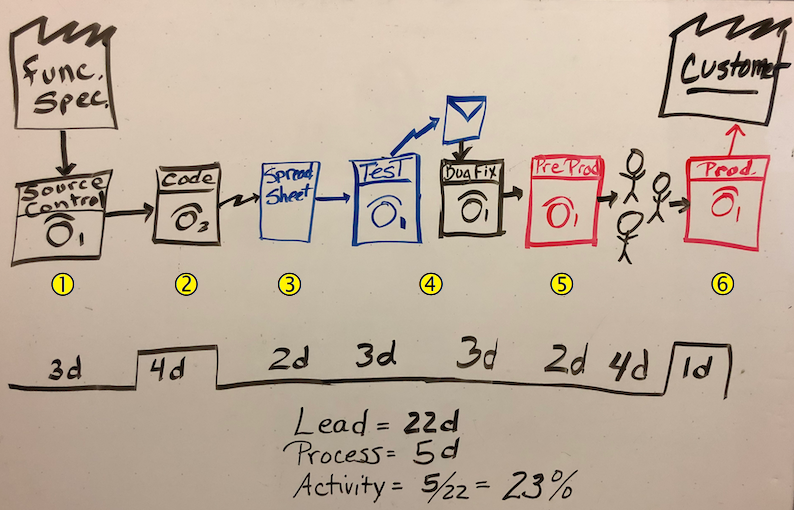
Mara, who's new to the team, is going to create a VSM to help her understand the existing process. With a VSM, she'll get a sense of where the team fits into the DevOps maturity model. As it turns out, more mature teams typically release faster, with greater confidence, and with fewer bugs than less mature teams.

Mara knows she doesn't understand everything yet. So she's going to create a quick VSM on the whiteboard in the meeting room. There will be some gaps and unanswered questions, but that's okay. It's a start. When she's done as much as she can, she'll share it with the team. The VSM will give everyone a common starting point for identifying the first steps toward improving how Tailspin develops and releases its websites.

Let's take a look at her map.

## Understand the current process

Mara gathers the team in the meeting room to present her VSM.



**Mara:** A VSM helps us measure where a process has value to the customer and where it's eating up time without producing any value. Our map begins on the upper left with the functional specification for the software. We'll follow just one feature to see how it moves through our current release cycle.

Some people roll their eyes, but Mara presses on.

### Development processes

Creating a new feature currently starts with creating a label in source control Callout 1. We have one person who can create labels, and that's Andy. We request a label by email. We use a centralized version control system, so Andy waits until all the existing code is checked in and stable before he creates the label. After the label is created, we get an email saying we can begin work. This process takes up to three days and has no value to the customer. Things with no value to the customer should take as little time as possible.

Coding a feature takes about four days for one person once we get access to all the files we need Callout 2. We have to be on the corporate network in order to access source control. This time has value to the customer. They want this feature.

### Test processes

After we decide that we have a stable build, we update a spreadsheet to tell Amita that there's a build ready for testing and where to find it Callout 3. It takes her two days to get notified.

Amita manually tests the build Callout 4. This process gets longer as the codebase grows. For now, let's say three days. She then emails Andy with bug reports. Testing doesn't add value, but it's necessary.

Andy then has to take time to triage the bugs and assign work Callout 4. It can take another three days for Andy to understand the issues and get them to the right developers.

### Operations processes

When Amita approves a build, she hands it off to Tim. Tim needs to deploy this build to the pre-production servers for more testing. Often, the pre-production servers are out of sync with the latest patches and updates that are needed to run the website. It takes Tim about two days to deploy to pre-production and run some tests. Again, while deploying to pre-production doesn't add value, it's necessary Callout 5.

After a build is ready for production, leadership needs to approve the release before it can be deployed. This happens in a meeting. It takes four days to get leadership to meet and review the release.

Eventually, Tim deploys the feature and the feature makes it to the customer here on the upper right of the VSM. If the production server configuration has drifted so it's out of sync with pre-production, Tim first needs to update it, and this takes one day Callout 6.

### Calculate the customer value metrics

Now we can look at the key performance metrics and see how we measure up.

Total lead time is the time it takes for a feature to make it to the customer. Process time is the time spent on a feature that has value to the customer. The activity ratio is process time divided by total lead time:

Activity ratio=Process timeTotal lead timeActivity ratio=Process timeTotal lead time

This is our **efficiency**. Multiply efficiency by 100 to get a percentage. The result is greater than 0% and typically less than 100%. A higher percentage indicates greater efficiency.

Substitute our numbers and we get:

Activity ratio = 5 days22 days=.23Activity ratio = 5 days22 days=.23

Multiply the result by 100% and you get **23%**.

As you can see, we have a lot of room for improvement. And taking 22 days to develop a feature is too long.

**Tim:** So how does this help us?

## Where do we go from here?

**Mara:** It helps to see where we are now so that we can pinpoint the areas where there's waste. We want to minimize the time we spend that has no value to the customer. I believe we can really improve our efficiency by adopting a DevOps approach. For one thing, we can automate a lot of these steps, and that will definitely cut down on the time.

I'm not suggesting we drop our current processes, but I think we can work toward a more efficient process in small increments without disrupting what we currently have in place.

Let's look at just a couple of areas where we can improve.

**Andy:** We might as well start at the beginning. It takes me a long time to get a label on the code so we can start the new feature. I have to walk around to the developers and ask them to check in what they have so we can build and test. If you can figure out how to speed that up, you'll have my attention.

Also, I noticed that you don't have time in there for the build itself. That takes half a day right now. It would be nice to see that time improve.

**Amita:** And dev doesn't always update the spreadsheet to let me know there's a new build that needs testing. It would save time if there was some way to make sure that part gets done.

**Mara:** Great! I think DevOps can help us out with all of these concerns.

**Andy:** We don't have time to change the process now. You heard Irwin. We're in crisis mode here!

**Mara:** I understand. For now, let's do what we always do. But we can all think about our part in the process and how we can improve. We can start making small changes in parallel to our current processes. That will let us see if DevOps will help us without disrupting what we have. I'll keep this map and the performance metrics. If we end up adopting Azure DevOps practices, then we can revisit the numbers. Maybe we can update the VSM at some point.

# Summary

* 4 minutes

As you can see, Mara and her team face a number of challenges. Although releases happen eventually, Mara feels they can happen much more frequently and efficiently.

Mara hopes she can convince the team it's at least worth testing out a DevOps approach. Perhaps they can apply a few DevOps practices as they finish up work on the Space Game website.

## What is DevOps?

At this point, we haven't yet defined DevOps. We'll do that in the next module. But for now, think of DevOps as something that brings together people, processes and products, automating software delivery to provide continuous value to your users.

Azure DevOps is a suite of services that spans the entire DevOps life cycle. Azure DevOps starts with planning and goes all the way through deployment and monitoring. If you already have some pieces in place, you can select which services you want to use. Azure DevOps integrates with many tools, such as Jenkins.

We'll go deeper into Azure DevOps in future modules. You can also check out these resources:

* Azure DevOps
* DevOps Resource Center

## Appendix: General recommendations for migrating and consolidating tools to Azure DevOps

The following sections are optional. They provide additional information beyond what's covered in this module.

You've seen how analyzing all your existing artifacts and tools is the first step toward designing a DevOps strategy. Once you've done that, here are a few recommendations for developing migration and integration strategies you can use if you're using Azure DevOps.

### Designing an authentication and access strategy

Azure DevOps Services uses enterprise-grade authentication. You can use either a Microsoft account or Azure Active Directory (Azure AD) to protect and secure your data. Many client applications such as Visual Studio or Visual Studio code, natively support authentication by either Microsoft Accounts or Azure AD. Eclipse can also support this, if you install a Team Explorer Everywhere plugin.

When you need a non-Microsoft tool to integrate directly with Azure DevOps Services and the tools don't directly support a Microsoft account or an Azure AD account for authentication, you can still use them by setting up personal access tokens. These tokens can be set up using Git credential managers or you can create them manually.

Personal access tokens are also useful when you need to establish access in command line tools, or in tools and tasks in build pipelines and when calling REST-based APIs, because you don't have a UI popping out to perform the authentication. When access is no longer required you can then just revoke the personal access token.

Azure DevOps is pre-configured with default security groups. Default permissions are assigned to the default security groups. But you can also configure access at the organization level, the collection level and at the project or object level. In the organization settings in Azure DevOps, you can configure app access policies. Based on your security policies, you might allow alternate authentication methods, allow third party applications to access via OAuth or even allow anonymous access to some projects.

For even tighter control, you can set conditional access to Azure DevOps. This offers simple ways to help secure resources when using Azure Active Directory for authentication. Conditional access policies such as multifactor authentication can help to minimize the risk of compromised credentials. As part of a conditional access policy you might require security group membership, a particular location or network identity, a specific operating system, a managed device, or other criteria.

### Migrating and integrating artifact repositories

While you can continue to work with your existing artifact repositories in their current locations when using Azure DevOps, there are advantages to migrating them. For NuGet, Azure DevOps Services provides hosted NuGet feeds as a service. By using this service, you can often eliminate the dependencies on on-premises resources such as file shares and locally hosted instances of NuGet.Server. The feeds can also be consumed by any continuous integration system that supports authenticated NuGet feeds.

### Designing a licensing strategy

Azure DevOps has many different licensing policies designed to suit many different situations. To get a quick overview, go to Pricing for Azure DevOps  to see what's available.

There is also a large number of both free and paid extensions, many of which you use to integrate third-party applications. Depending upon your current configuration, you might also decide to implement Group-Based Licensing. This makes it easier to manage licenses in Azure DevOps. If you're planning to make this change though, it's important to avoid a situation where migration to Group-Based Licensing results in users temporarily losing their currently assigned licenses.

In addition to Azure DevOps Licensing, you need to ensure you have appropriate licenses for any third party tools that you need. Also, you need to make sure that any open-source tools that you have included have licenses that are compatible with how you intend to use them.

### Migrating and integrating source control

Similarly to artifacts, you might consider migrating your source control to Azure DevOps by using Azure Repos. However, moving your team from a centralized version control system to a distributed system like Git involves much more than learning the new commands. A successful migration to Git requires that you understand the differences in how file history and branches are handled.

The Azure DevOps team recommends the following process:

1. Evaluate the tools and processes you're using.
2. Select a branching strategy for Git.
3. Decide how to migrate history – or if you even want to.
4. Maintain your old version control system.
5. Remove binaries and executables from source control.
6. Train your team in the concepts and practice of Git.
7. Perform the actual migration to Git.

### Migrating and integrating work management tools

Migrating from other work management tools to Azure DevOps takes considerable planning. Most work management tools are highly configurable by the end user. This means that there might not be a tool available that will perform the migration without further configuration. One example is Jira.

In the Visual Studio Marketplace, Solidify  offers a tool for Jira to Azure DevOps migration. It does the migration in two phases. Jira issues are exported to files and then the files are imported to Azure DevOps.

If you decide to try to write the migration code yourself, the following blog post provides sample code that might help you to get started: Migrate Your Project From Jira to Azure DevOps

### Migrating and integrating test tools

Azure Test Plans  are used to track manual testing for sprints and milestones. This allows you to track when that testing is complete.

Apache JMeter  is open source software written in Java and designed to load test functional behavior and measure performance.

Pester  is a tool that can be used to automate the testing of PowerShell code.

SoapUI  is another testing framework for SOAP and REST testing.