AZ-400: Design and implement a release strategy

Introduction to continuous delivery

Introduction

Completed

- 1 minute

This module introduces continuous delivery concepts and their implementation in a traditional IT development cycle.

Learning objectives

After completing this module, students and professionals can:

- Explain continuous delivery (CD).
- Implement continuous delivery in your development cycle.
- Understand releases and deployment.
- Identify project opportunities to apply CD.

Prerequisites

- Understanding of what DevOps is and its concepts.
- Familiarity with version control principles is helpful but isn't necessary.
- Beneficial to have experience in an organization that delivers software.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Explore traditional IT development cycle

Completed

- 2 minutes

A few years ago, IT was a facilitating department. IT was there to support the business users, and because time had proven that developed software had bad quality by default, software changes were a risk.

The resolution for this "quality problem" was to keep changes under strict control.

The department that became responsible for controlling the changes became the IT(-Pro) department.

In the past and today, the IT(-Pro) department is responsible for the stability of the systems, while the development department is responsible for creating new value.

This split brings many companies into a problematic situation. Development departments are motivated to deliver value as soon as possible to keep their customers happy.

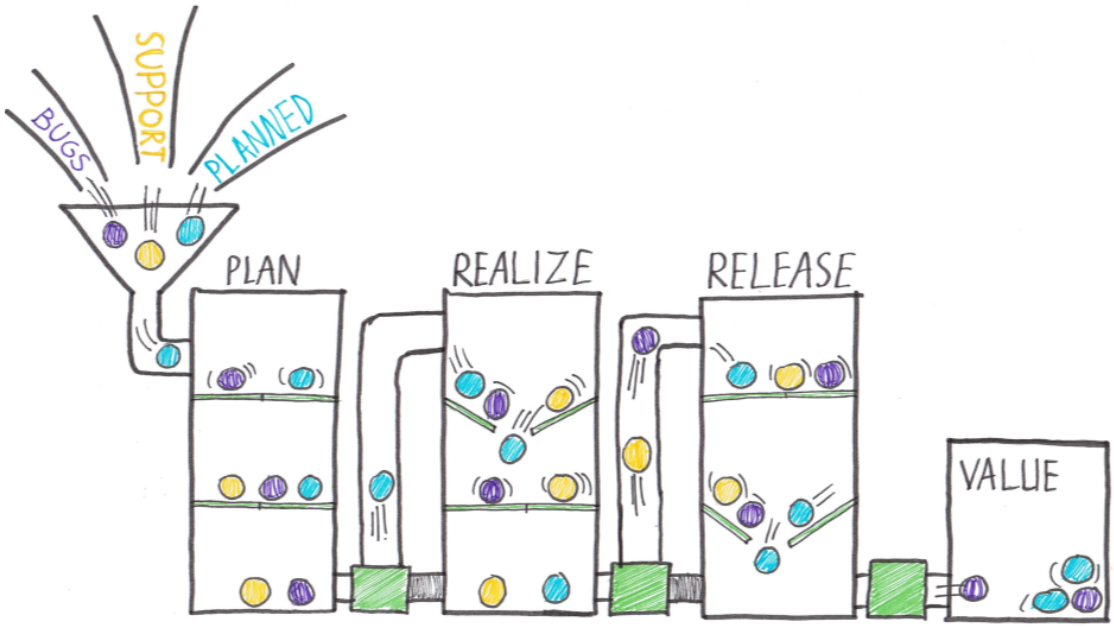
On the other hand, IT is motivated to change nothing because change is a risk, and they're responsible for eliminating the risks and keeping everything stable. And what do we get out of it? Long release cycles.

Silo-based development

Long release cycles, numerous tests, code freezes, night and weekend work, and many people ensure that everything works.

But the more we change, the more risk it leads to, and we're back at the beginning, on many occasions resulting in yet another document or process that should be followed.

It's what I call silo-based development.



If we look at this picture of a traditional, silo-based value stream, we see Bugs and Unplanned work, necessary updates or support work, and planned (value-adding) work, all added to the teams' backlog.

Everything is planned, and the first "gate" can be opened. Everything drops to the next phase. All the work, and so all the value, moves in piles to the next stage.

It moves from the Plan phase to a Realize phase where all the work is developed, tested, and documented, and from here, it moves to the release phase.

All the value is released at the same time. As a result, the release takes a long time.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

What is continuous delivery?

Completed

- 2 minutes

Continuous delivery (CD) is a set of processes, tools, and techniques for rapid, reliable, and continuous software development and delivery.

It means that continuous delivery goes beyond the release of software through a pipeline. The pipeline is a crucial component and the focus of this course, but continuous delivery is more.

To explain a bit more, look at the eight principles of continuous delivery:

- The process for releasing/deploying software must be repeatable and reliable.
- Automate everything!
- If something is difficult or painful, do it more often.
- Keep everything in source control.
- Done means "released."
- Build quality in!
- Everybody has responsibility for the release process.
- Improve continuously.

If we want to fulfill these eight principles, we can see that an automated pipeline doesn't suffice.

To deploy more often, we need to reconsider our:

- Software architecture (monoliths are hard to deploy).
- Testing strategy (manual tests don't scale well).
- Organization (separated business and IT departments don't work smoothly), and so forth.

This module will focus on the release management part of continuous delivery but be aware of the other changes you might find.

Find the next bottleneck in your process, solve it, learn from it, and repeat it forever.

Continuous delivery is an enabler for DevOps. DevOps focuses on organizations and bringing people together to build and run their software products.

Continuous delivery is a practice. It's being able to deliver software on-demand. Not necessarily 1000 times a day. Deploying every code change to production is what we call continuous deployment.

To do it, we need automation and a strategy.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Move to continuous delivery

Completed

- 3 minutes

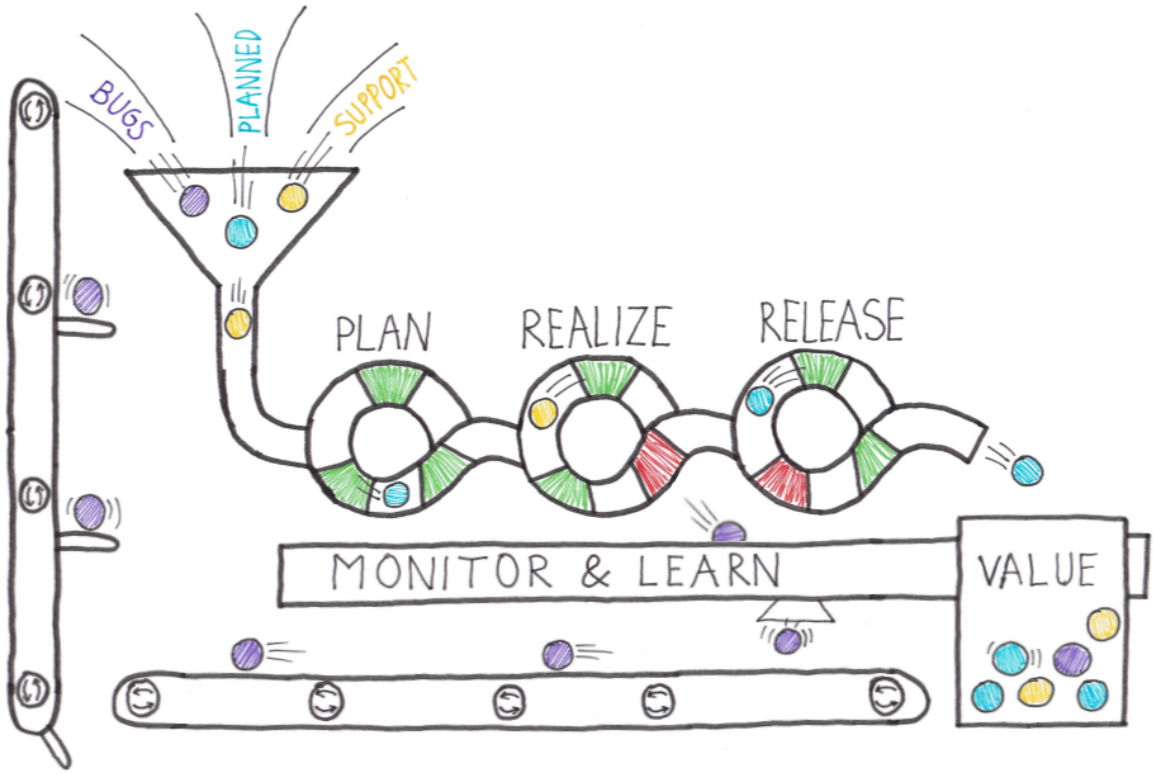
Times have changed, and we need to deal with a new normal. Our customers demand working software, and they wanted it yesterday.

If we can't deliver, they go to a competitor. And competition is fierce. With the Internet, we always have global competition.

Our competitors on our stack deliver a best-of-breed tool for one aspect of the software we built.

We need to deliver fast, and our product must be good. And we should do this with our cheap software production and high quality.

To achieve this, we need something like Continuous Delivery.



We need to move towards a situation where the value isn't piled up and released all at once but flows through a pipeline.

Just like in the picture, a piece of work is a marble. And only one part of the work can flow through the pipeline at once.

So, work must be prioritized in the right way. As you can see, the pipeline has green and red outlets.

We want to have these feedback loops or quality gates in place. A feedback loop can be different things:

- A unit test to validate the code.
- An automated build to validate the sources.
- An automated test on a Test environment.
- Some monitor on a server.
- Usage instrumentation in the code.

If one of the feedback loops is red, the marble can't pass the outlet and will end up in the Monitor and Learn tray.

It's where the learning happens. The problem is analyzed and solved so it's green the next time a marble passes the outlet.

Every workflow goes through the pipeline until it ends in the value tray.

The more that is automated, the faster value flows through the pipeline.

Companies want to move toward Continuous Delivery.

- They see the value.
- They hear their customers.
- Companies wish to deliver their products as fast as possible.
- Quality should be higher.
- The move to production should be faster.
- Technical Debt should be lower.

The introduction of Agile and Scrum was a great way to improve your software development practices.

Last year around 80% of companies claimed that they adopted Scrum as a software development practice.

Using Scrum, many teams can produce a working piece of software after a sprint of maybe two or three weeks.

But creating working software isn't the same as delivering working software.

The result is that all "done" increments are waiting to be delivered in the next release, which is coming in a few months.

We see now that Agile teams within a non-agile company are stuck in a delivery funnel.

The bottleneck is no longer the production of working software, but the problem has become the delivery of working software.

The finished product is waiting to be delivered to the customers to get business value, but it doesn't happen.

Continuous delivery needs to solve this problem.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Understand releases and deployments

Completed

- 4 minutes

One of the essential steps in moving software more quickly to production is changing how we deliver software to production.

It's common to have teams that need to do overtime on the weekend to install and release new software in our industry.

It's caused by the fact that we have two parts of the release process bolted together. As soon as we deploy new software, we also release new features to the end users.

The best way to safely move your software to production while maintaining stability is by separating these two concerns. So, we separate deployments from our release.

It can also be phrased as separating your functional release from your technical release (deployment).

What is a release, and what is a deployment?

When we talk about releases and deployments, we see that commonly used tools deal differently with the terminology as we did in the previous chapter.

To understand the concepts and the technical implementation in many tools, you need to know how tool vendors define the difference between a release and a deployment.

A release is a package or container containing a versioned set of artifacts specified in a release pipeline in your CI/CD process.

It includes a snapshot of all the information required to carry out all the tasks and actions in the release pipeline, such as:

- The stages or environments.
- The tasks for each one.
- The values of task parameters and variables.
- The release policies such as triggers, approvers, and release queuing options.

There can be multiple releases from one release pipeline (or release process).

Deployment is the action of running the tasks for one stage, which results in a tested and deployed application and other activities specified for that stage.

Starting a release starts each deployment based on the settings and policies defined in the original release pipeline.

There can be multiple deployments of each release, even for one stage.

When a release deployment fails for a stage, you can redeploy the same release to that stage.

See also Releases in Azure Pipelines .

Separating technical releases from functional releases

When we want to separate the technical and functional release, we need to start with our software itself.

The software needs to be built so that new functionality or features can be hidden from end users while it's running.

A common way to do this is the use of Feature Toggles. The simplest form of a Feature Toggle is an if statement that either executes or doesn't execute a certain piece of code.

By making the if-statement configurable, you can implement the Feature Toggle. We'll talk about Feature Toggles in Module 3 in more detail.

See also: Explore how to progressively expose your features in production for some or all users .

Once we've prepared our software, we must ensure that the installation won't expose any new or changed functionality to the end user.

When the software has been deployed, we need to watch how the system behaves. Does it act the same as it did in the past?

If the system is stable and operates the same as before, we can decide to flip a switch. It might reveal one or more features to the end user or change a set of routines part of the system.

The whole idea of separating deployment from release (exposing features with a switch) is compelling and something we want to incorporate in our Continuous Delivery practice.

It helps us with more stable releases and better ways to roll back when we run into issues when we have a new feature that produces problems.

We switch it off again and then create a hotfix. By separating deployment from the release of a feature, you make the opportunity to deploy any time of the day since the new software won't affect the system that already works.

Topics you might want to discuss are:

- Does your organization need Continuous Delivery?
- Do you use Agile/Scrum?
    - Is everybody involved or only the Dev departments?
- Can you deploy your application multiple times per day? Why or why not?
- What is the main bottleneck for Continuous Delivery in your organization?
    - The Organization
    - Application Architecture
    - Skills
    - Tooling
    - Tests
    - Other things?

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Understand release process versus release

Completed

- 2 minutes

Before diving into high-quality release pipelines, we must consider the difference between release and release processes. Or, when you talk about tooling, a release pipeline.

We start with defining a release process or release pipeline. The release pipeline contains all the steps you walk through when you move your artifact from one of the artifact sources discussed earlier through the stages or environments.

The stage can be a development stage, a test stage, a production stage, or a stage where a specific user can access the application.

Part of your pipeline is the people who approve the release or the deployment to a specific stage. Also, triggers or schedules on which the releases execute, and the release gates, the automatic approvals of the process.

The release itself is something different. The release is an instance of the release pipeline. You can compare it with object instantiation.

In Object Orientation, a class contains the blueprint or definition of an object. But the object itself is an instance of that blueprint.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Knowledge check

Completed

- 4 minutes

Choose the best response for each question. Then select **Check your answers** .

Check your knowledge

1.

Which of the following choices isn't considered a feedback loop or quality gates during the Continuous Delivery?

○

An automated test on a Test environment.

○

An automated build to validate the sources.

○

An automated Azure Boards and repository integration.

2.

Which of the following choices best describe Continuous Delivery?

○

Continuous delivery (CD) triggers automated testing for every code change.

○

Continuous delivery (CD) is a set of processes, tools, and techniques for rapid, reliable, and continuous software development and delivery.

○

Continuous delivery (CD) is the process of automating the build and testing of code every time a team member commits changes to version control.

3.

Which of the following choices is a definition for a technical release (deployment)?

○

Deployment is the action of running the tasks for one stage, which results in a tested and deployed application and other activities specified for that stage.

Deployment is a package or container containing a versioned set of artifacts specified in a release pipeline in your CI/CD process.

Deployment is a construct that holds a versioned set of artifacts specified in a CI/CD pipeline.

4.

Which of the following choices is a container with versioned artifacts, pipeline, approvals, stages, variables?

Deployment.

Build.

Release.

Check your answers

You must answer all questions before checking your work.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Summary

Completed

- 1 minute

This module introduces continuous delivery concepts and their implementation in a traditional IT development cycle.

You learned how to describe the benefits and usage of:

- Explain continuous delivery (CD).
- Implement continuous delivery in your development cycle.

- Understand releases and deployment.
- Identify project opportunities to apply CD.

Learn more

- What is Continuous Delivery? - Azure DevOps | Microsoft Docs .
- Continuous Delivery vs. Continuous Deployment | Microsoft Azure .
- Release pipelines - Azure Pipelines | Microsoft Docs .
- Define a Classic release pipeline - Azure Pipelines | Microsoft Docs .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Create a release pipeline

Introduction

Completed

- 1 minute

This module describes Azure Pipelines capabilities, build and release tasks, and multi-configuration and multi-agent differences.

Learning objectives

After completing this module, students and professionals can:

- Explain the terminology used in Azure DevOps and other Release Management Tooling.
- Describe what a Build and Release task is, what it can do, and some available deployment tasks.
- Implement release jobs.
- Differentiate between multi-agent and multi-configuration release jobs.

Prerequisites

- Understanding of what DevOps is and its concepts.
- Familiarity with version control principles is helpful but isn't necessary.
- Beneficial to have experience in an organization that delivers software.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Describe Azure DevOps release pipeline capabilities

Completed

- 3 minutes

Azure DevOps has extended support for pipelines as code (also called YAML pipelines) for continuous deployment and started introducing various release management capabilities into pipelines as code.

The existing UI-based release management solution in Azure DevOps is referred to as classic release.

You'll find a list of capabilities and availability in YAML pipelines vs. classic build and release pipelines in the following table.

Feature	YAML	Classic Build	Classic Release	Notes
Agents	Yes	Yes	Yes	Specifies a required resource on which the pipeline runs.
Approvals	Yes	No	Yes	Defines a set of validations required before completing a deployment stage.
Artifacts	Yes	Yes	Yes	Supports publishing or consuming different package types.
Caching	Yes	Yes	No	Reduces build time by allowing outputs or downloaded dependencies from one run to be reused in later runs. In Preview, available with Azure Pipelines only.
Conditions	Yes	Yes	Yes	Specifies conditions to be met before running a job.
Container jobs	Yes	No	No	Specifies jobs to run in a container.
Demands	Yes	Yes	Yes	Ensures pipeline requirements are met before running a pipeline stage. Requires self-hosted agents.
Dependencies	Yes	Yes	Yes	Specifies a requirement that must be met to run the next job or stage.
Deployment groups	Yes	No	Yes	Defines a logical set of deployment target machines.
Deployment group jobs	No	No	Yes	Specifies a job to release to a deployment group.
Deployment jobs	Yes	No	No	Defines the deployment steps. Requires Multi-stage pipelines experience.
Environment	Yes	No	No	Represents a collection of resources targeted for deployment. Available with Azure Pipelines only.
Gates	No	No	Yes	Supports automatic collection and evaluation of external health signals before completing a release stage. Available with Azure Pipelines only.
Jobs	Yes	Yes	Yes	Defines the execution sequence of a set of steps.
Service connections	Yes	Yes	Yes	Enables a connection to a remote service that is required to execute tasks in a job.
Service containers	Yes	No	No	Enables you to manage the lifecycle of a containerized service.
Stages	Yes	No	Yes	Organizes jobs within a pipeline.
Task groups	No	Yes	Yes	Encapsulates a sequence of tasks into a single reusable task. If using YAML, see templates.

Feature	YAML	Classic Build	Classic Release	Notes
Tasks	Yes	Yes	Yes	Defines the building blocks that make up a pipeline.
Templates	Yes	No	No	Defines reusable content, logic, and parameters.
Triggers	Yes	Yes	Yes	Defines the event that causes a pipeline to run.
Variables	Yes	Yes	Yes	Represents a value to be replaced by data to pass to the pipeline.
Variable groups	Yes	Yes	Yes	Use to store values that you want to control and make available across multiple pipelines.

Continue

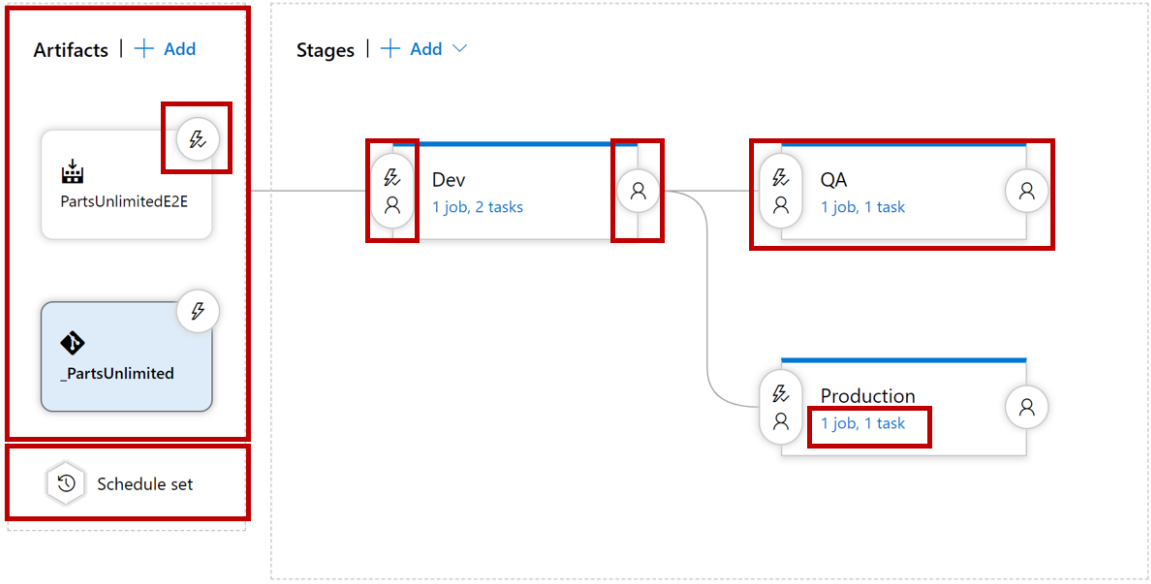
Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Explore release pipelines

Completed

- 3 minutes

A release pipeline takes artifacts and releases them through stages and finally into production.



Let us quickly walk through all the components step by step.

The first component in a release pipeline is an artifact:

- Artifacts can come from different sources.

- The most common source is a package from a build pipeline.
- Another commonly seen artifact source is, for example, source control.

Furthermore, a release pipeline has a trigger: the mechanism that starts a new release.

A trigger can be:

- A manual trigger, where people start to release by hand.
- A scheduled trigger, where a release is triggered based on a specific time.
- A continuous deployment trigger, where another event triggers a release. For example, a completed build.

Another vital component of a release pipeline is stages or sometimes called environments. It's where the artifact will be eventually installed. For example, the artifact contains the compiled website installed on the webserver or somewhere in the cloud. You can have many stages (environments); part of the release strategy is finding the appropriate combination of stages.

Another component of a release pipeline is approval.

People often want to sign a release before installing it in the environment.

In more mature organizations, this manual approval process can be replaced by an automatic process that checks the quality before the components move on to the next stage.

Finally, we have the tasks within the various stages. The tasks are the steps that need to be executed to install, configure, and validate the installed artifact.

In this part of the module, we'll detail all the release pipeline components and talk about what to consider for each element.

The components that make up the release pipeline or process are used to create a release. There's a difference between a release and the release pipeline or process. The release pipeline is the blueprint through which releases are done. We'll cover more of it when discussing the quality of releases and releases processes.

See also Release pipelines .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Explore artifact sources

Completed

- 4 minutes

What is an artifact? An artifact is a deployable component of your application. These components can then be deployed to one or more environments.

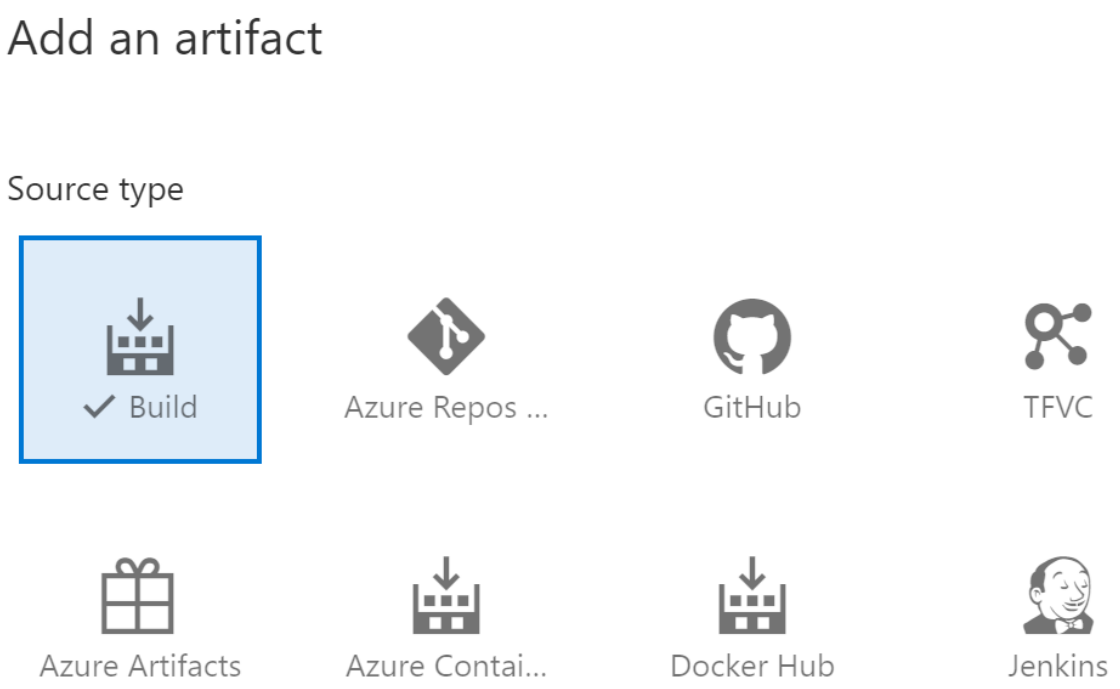
In general, the idea about build and release pipelines and Continuous Delivery is to build once and deploy many times.

It means that an artifact will be deployed to multiple environments. The artifact should be a stable package if you want to achieve it.

The configuration is the only thing you want to change when deploying an artifact to a new environment.

The contents of the package should never change. It's what we call immutability . We should be 100% sure that the package that we build, the artifact, remains unchanged.

How do we get an artifact? There are different ways to create and retrieve artifacts, and not every method is appropriate for every situation.



The most common way to get an artifact within the release pipeline is to use a build artifact.

The build pipeline compiles, tests, and eventually produces an immutable package stored in a secure place (storage account, database, and so on).

The release pipeline then uses a secure connection to this secured place to get the build artifact and do extra actions to deploy it to an environment.

The significant advantage of using a build artifact is that the build produces a versioned artifact.

The artifact is linked to the build and gives us automatic traceability. We can always find the sources that made this artifact. Another possible artifact source is version control.

We can directly link our version control to our release pipeline.

The release is related to a specific commit in our version control system. With that, we can also see which version of a file or script is eventually installed. In this case, the version doesn't come from the build but from version control.

Consideration for choosing a version control artifact instead of a build artifact can be that you only want to deploy one specific file. If you don't need to run more actions before using this file in your release pipeline, creating a versioned package (build artifact) containing only one file doesn't make sense.

Helper scripts that do actions to support the release process (clean up, rename, string actions) are typically good candidates to get from version control.

Another possibility of an artifact source can be a network share containing a set of files. However, you should be aware of the possible risk. The risk is that you aren't 100% sure that the package you're going to deploy is the same package that was put on the network share. If other people can also access the network share, the package might be compromised. Therefore, this option won't be sufficient to prove integrity in a regulated environment (banks, insurance companies).

Finally, container registries are a rising star regarding artifact sources. Container registries are versioned repositories where container artifacts are stored. Pushing a versioned container to the content repository and consuming that same version within the release pipeline has more or less the same advantages as using a build artifact stored in a safe location.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Choose the appropriate artifact source

Completed

- 3 minutes

When you use a release pipeline to deploy your packages to production, you need traceability.

That means you want to know where the package that you're deploying originates from.

It's essential to understand that the sources that you built and checked into your version control are precisely the same as the sources that you're going to deploy to the various environments that are going to be used by your customers.

Primarily when you work in a regulated environment like a bank or an insurance company, auditors ask you to provide traceability to sources that you deployed to prove the integrity of the package.

Another crucial aspect of your artifacts is auditability. You want to know who changed that line of code and who triggered the build that produces the artifact deployed.

A proper mechanism to make sure you can provide the correct traceability and auditability is using immutable packages.

It isn't something that you can buy, but something that you need to implement yourself.

Using a build pipeline that produces a package stored in a location that humans can't access, you ensure the sources are unchanged throughout the whole-release process. It's an essential concept of release pipelines.

You identify an immutable package by giving it a version so that you can refer to it at a later stage. Versioning strategy is a complex concept and isn't in the scope of this module.

Still, having a unique identification number or label attached to the package and ensuring that this number or label cannot be changed or modified afterward ensures traceability and auditability from source code to production.

Read more about Semantic Versioning .

Choosing the right artifact source is tightly related to the requirements you have about traceability and auditability.

If you need an immutable package (containing multiple files) that can never be changed and be traced, a build artifact is the best choice.

If it's one file, you can directly link to source control.

You can also point at a disk or network share, but it implies some risk-concerning auditability and immutability. Can you ensure the package never changed?

See also Release artifacts and artifact sources .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Exercise - select an artifact source

Completed

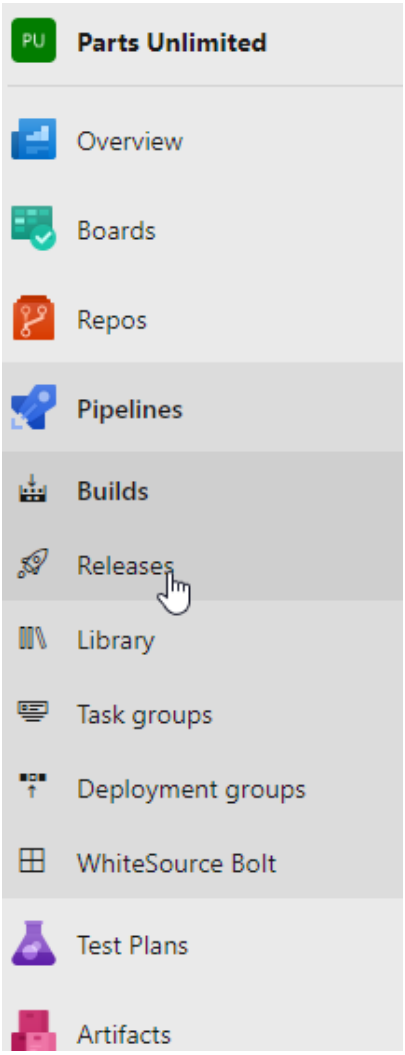
- 30 minutes

In this exercise, you'll investigate Artifact Sources.

Steps

Let's look at how to work with one or more artifact sources in the release pipeline.

1. In the Azure DevOps environment, open the **Parts Unlimited** project, then from the main menu, select **Pipelines** , then select **Releases** .



2. In the main screen area, select **New pipeline** .

No release pipelines found

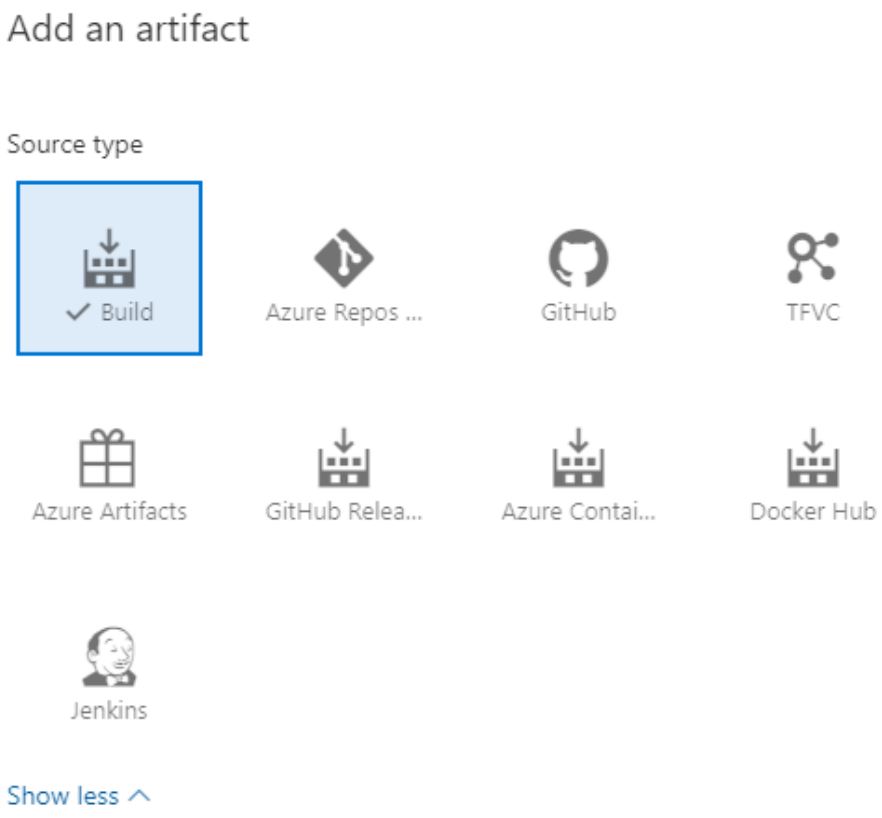
Automate your release process in a few easy steps with a new pipeline

New pipeline

3. In the **Select a template** pane, see the available templates, but then select the **Empty job** option at the top. It's because we're going to focus on selecting an artifact source.

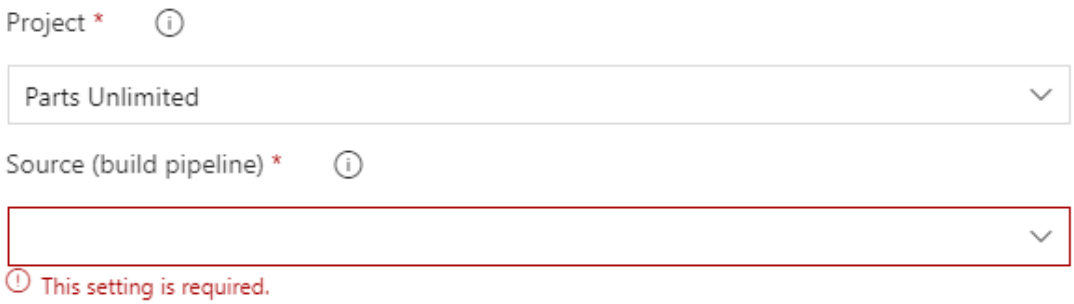
4. In the **Artifacts** section, select **+Add an artifact** .

5. See the available options in the **Add an artifact** pane, and select the option to see **more artifact types** , so that you can see all the available artifact types:

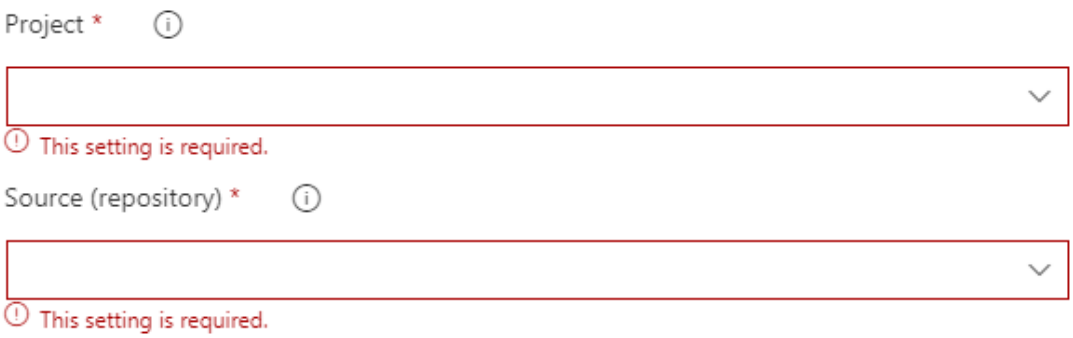


While we're in this section, let's briefly look at the available options.

6. Select **Build** and see the parameters required. This option is used to retrieve artifacts from an Azure DevOps Build pipeline. Using it requires a project name and a build pipeline name. (Projects can have multiple build pipelines). It's the option that we'll use shortly.

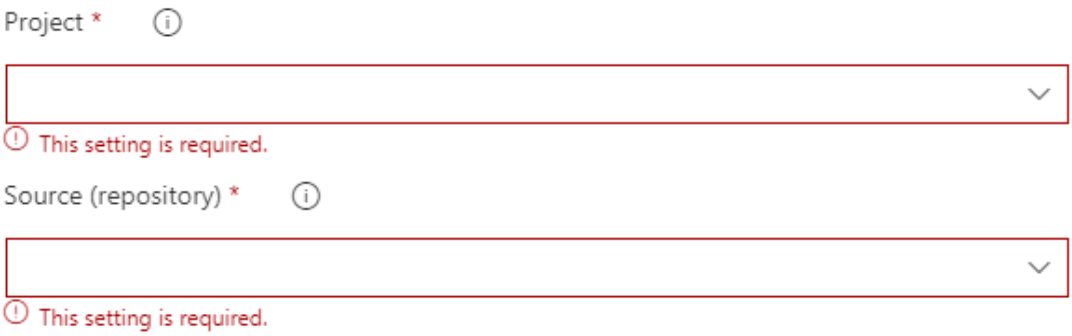


7. Select **Azure Repository** and see the parameters required. It requires a project name and asks you to select the source repository.



8. Select **GitHub** and see the parameters required. The **Service** is a connection to the GitHub repository. It can be authorized by either OAuth or by using a GitHub personal access token. You also need to select the source repository.

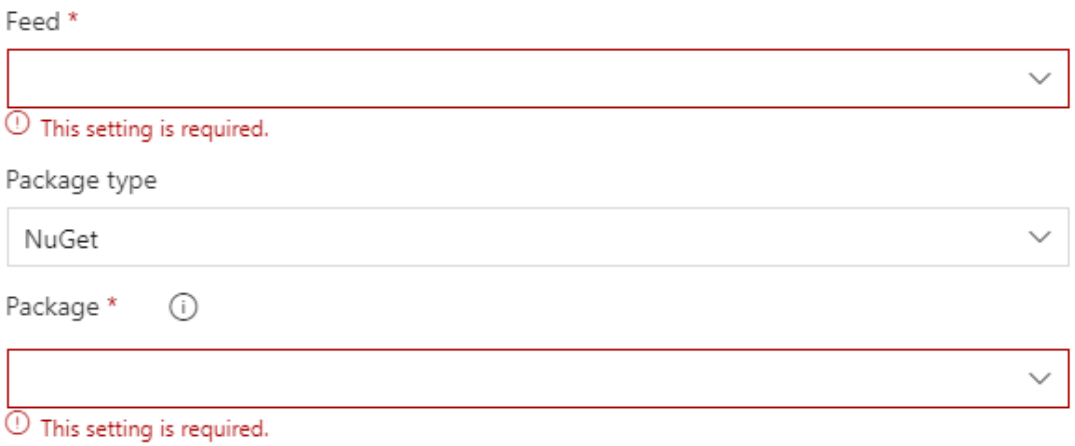
9. Select **TFVC** and see the parameters required. It also requires a project name and asks you to select the source repository.



Note

A release pipeline can have more than one set of artifacts as input. A typical example is when you also need to consume a package from a feed and your project source.

10. Select **Azure Artifacts** and see the parameters required. It requires you to identify the feed, package type, and package.

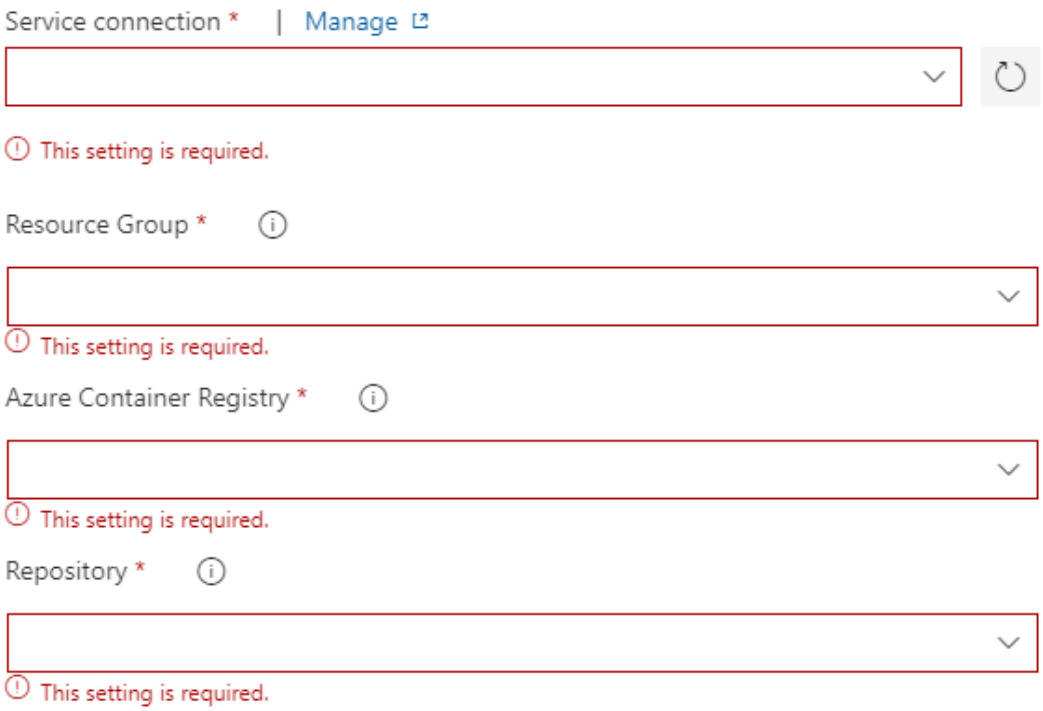


11. Select **GitHub Release** and see the parameters required. It requires a service connection and the source repository.

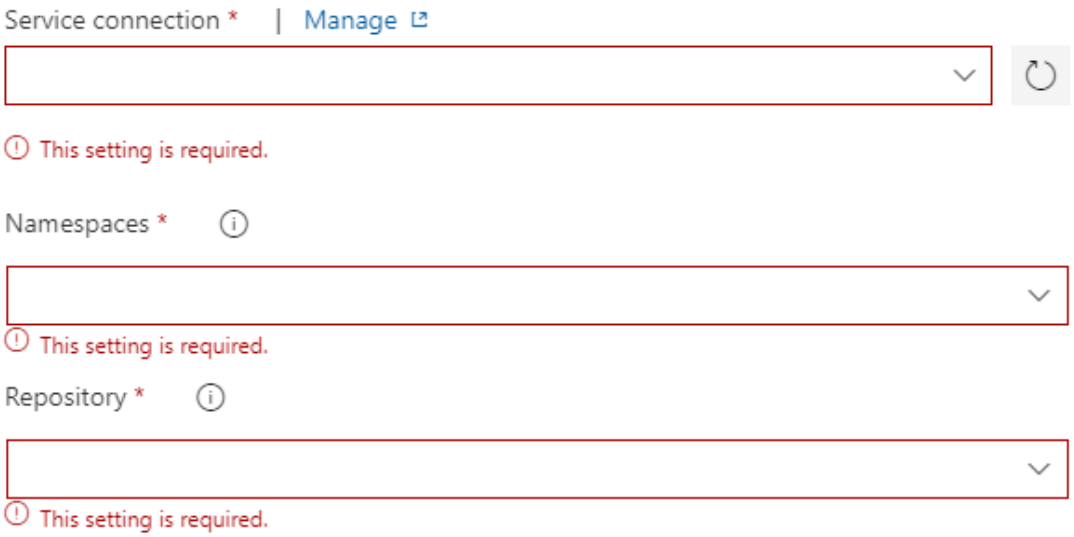
Note

We'll discuss service connections later in the module.

12. Select **Azure Container Registry** and see the parameters required. Again, it requires a secure service connection, and the Azure Resource Group details that the container registry is located. It allows you to provide all your Docker containers directly into your release pipeline.



13. Select **Docker Hub** and see the parameters required. This option would be helpful if your containers are stored in Docker Hub rather than in an Azure Container Registry. After choosing a secure service connection, you need to select the namespace and the repository.

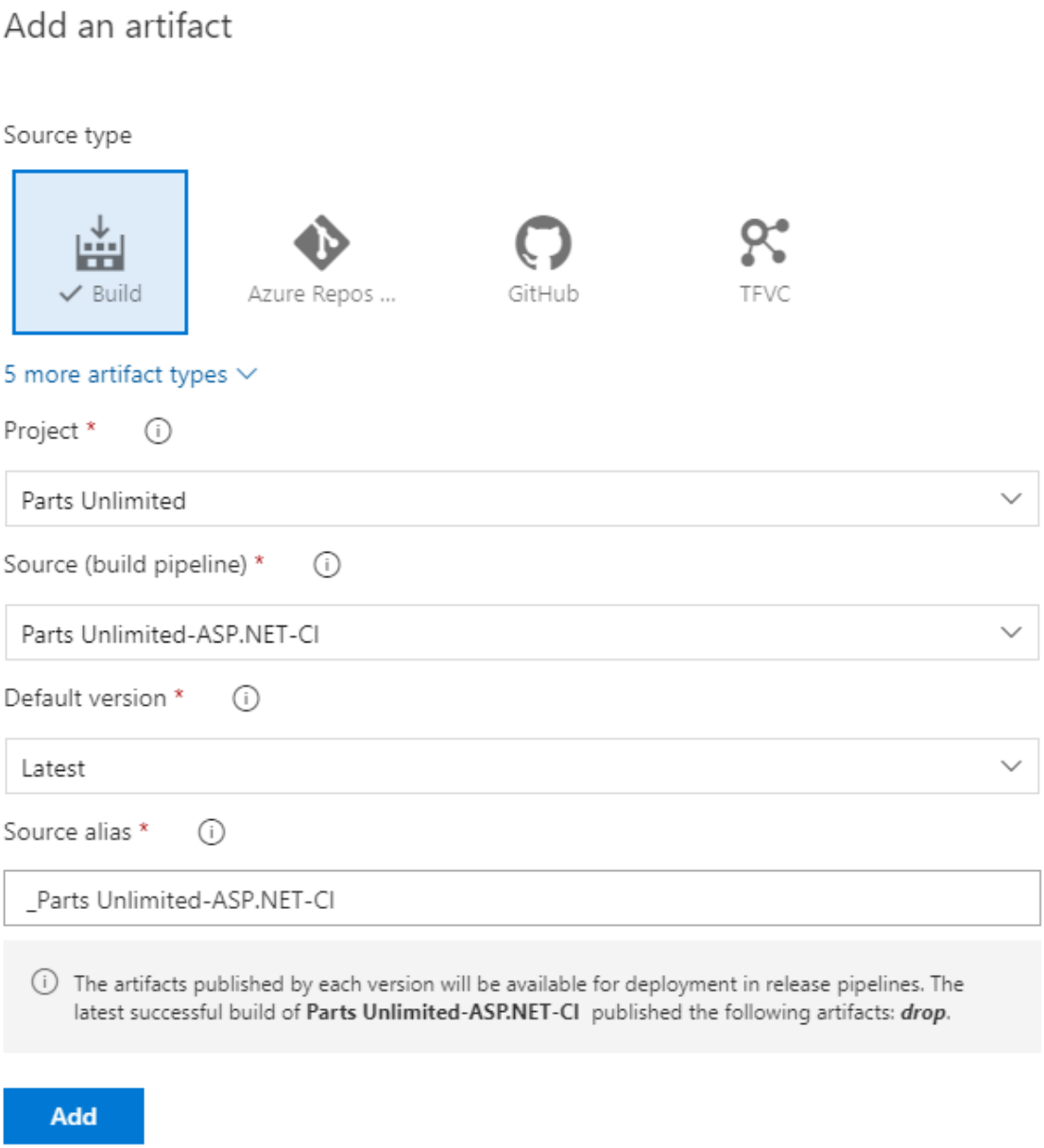


14. Finally, select **Jenkins** and see the parameters required. You don't need to get all your artifacts from Azure. You can retrieve them from a Jenkins build. So, if you have a Jenkins Server in your infrastructure, you can use the build artifacts from there directly in your Azure Pipelines.

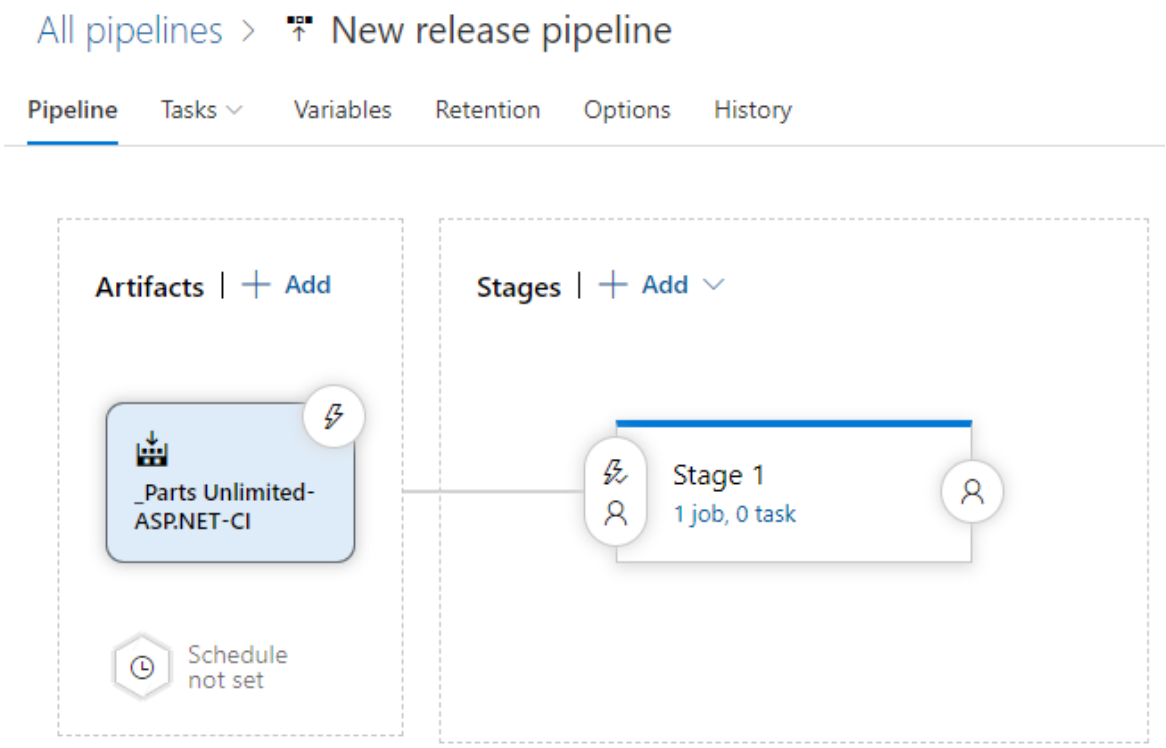
Configuring the build artifact

Let's return to adding our Build output as the artifact source.

15. Select the **Build** source type again. See that the Project should show the current project. From the **Source (build pipeline)** drop-down list, select **Parts Unlimited-ASP.NET-CI** . Take a record of the default values for the other options, and then select **Add** .



We've now added the artifacts that we'll need for later walkthroughs.



16. To save the work, select **Save** , then in the Save dialog box, select **OK** .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Examine considerations for deployment to stages

Completed

- 3 minutes

When you have a clear view of the different stages you'll deploy, you need to think about when you want to deploy to these stages.

As we mentioned in the introduction, Continuous Delivery is about deploying multiple times a day and can deploy on-demand.

When we define our cadence, questions that we should ask ourselves are:

- Do we want to deploy our application?
- Do we want to deploy multiple times a day?
- Can we deploy to a stage? Is it used?

For example, a tester testing an application during the day might not want to deploy a new version of the app during the test phase.

Another example is when your application incurs downtime, you don't want to deploy when users use the application.

The frequency of deployment, or cadence, differs from stage to stage.

A typical scenario we often see is continuous deployment during the development stage.

Every new change ends up there once it's completed and builds.

Deploying to the next phase doesn't always occur multiple times but only at night.

When designing your release strategy, choose your triggers carefully and consider the required release cadence.

Some things we need to take into consideration are:

- What is your target environment?
- Does one team use it, or do multiple teams use it?
    - If a single team uses it, you can deploy it frequently. Otherwise, it would be best if you were a bit more careful.
- Who are the users? Do they want a new version multiple times a day?
- How long does it take to deploy?
- Is there downtime? What happens to performance? Are users affected?

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Exercise - set up stages

Completed

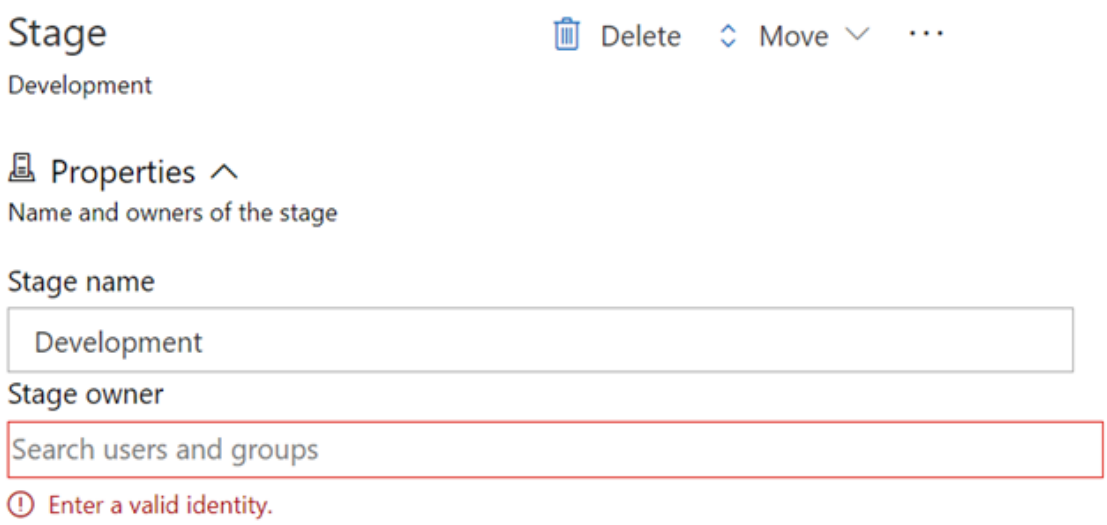
- 30 minutes

In this demonstration, you'll investigate Stages.

Steps

Look at the other section in our release pipeline: Stages.

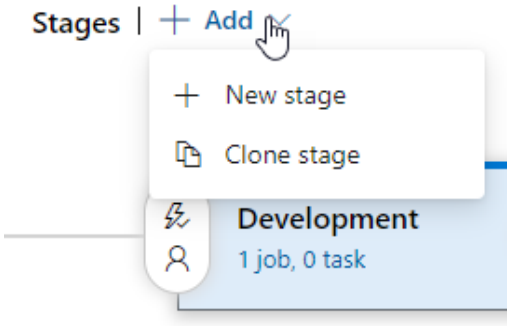
1. Click on **Stage 1,** and in the Stage properties pane, set **Stage name** to **Development** and close the pane.



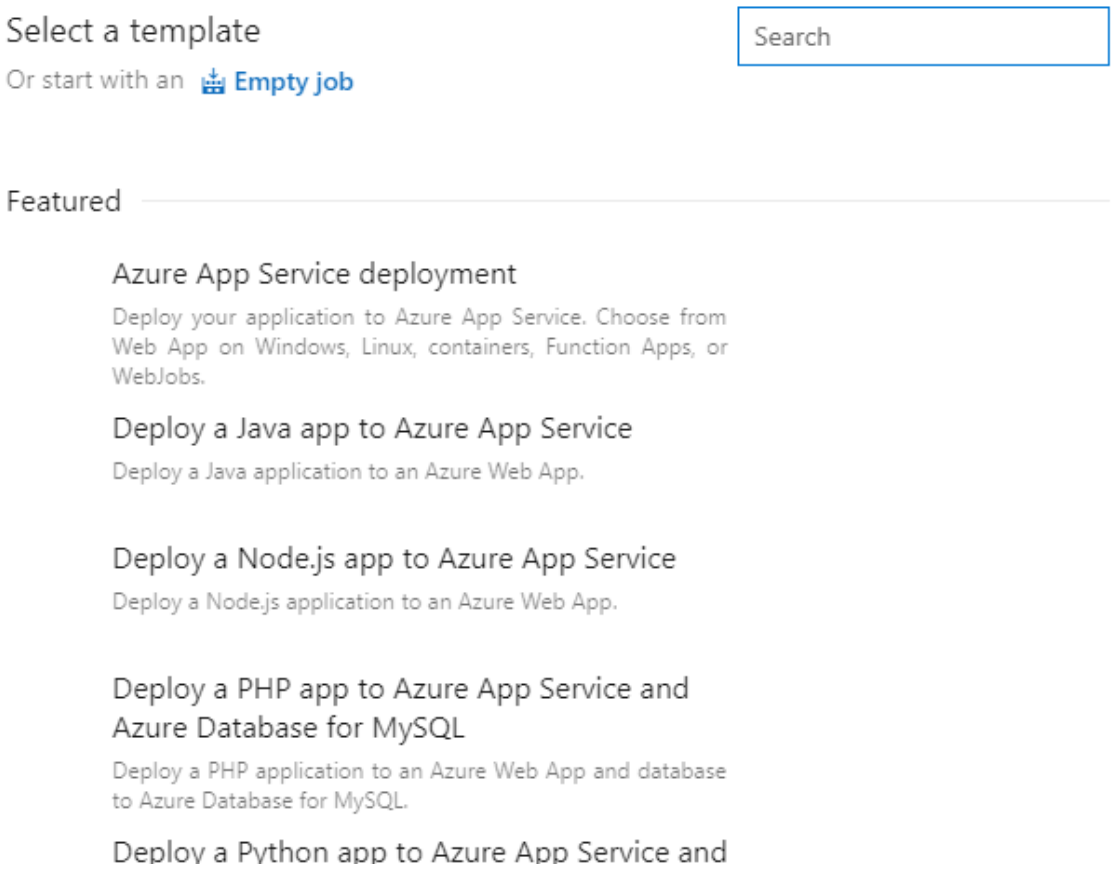
Note

Stages can be based on templates. For example, you might deploy a web application using node.js or Python. For this walkthrough that won't matter because we're focusing on defining a strategy.

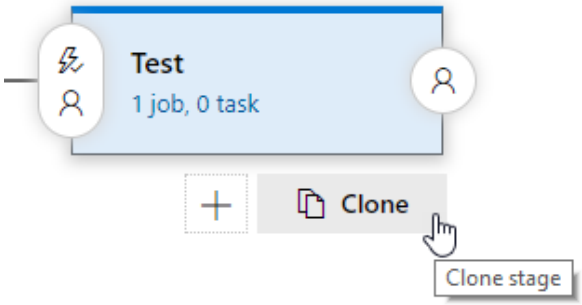
2. To add a second stage, click **+Add** in the Stages section and note the available options. You have a choice to create a new stage or clone an existing stage. Cloning a stage can help minimize the number of parameters that need to be configured. But for now, click **New stage** .



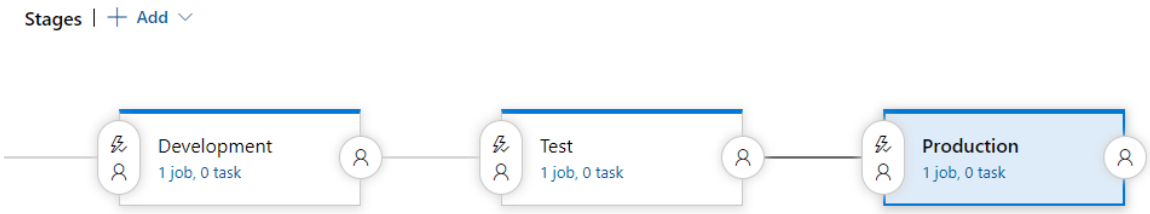
3. When the **Select a template** pane appears, scroll down to see the available templates. We don't need any of these, so click **Empty job** at the top, then in the Stage properties pane, set **Stage name** to **Test** , and then close the pane.



4. Hover over the **Test** stage and notice that two icons appear below. These are the same options that were available in the menu drop-down that we used before. Click the **Clone** icon to clone the stage to a new stage.



5. Click on the **Copy of Test** stage, and in the stage properties pane, set **Stage name** to **Production** and close the pane.



We've now defined a traditional deployment strategy. Each stage contains a set of tasks, and we'll look at those tasks later in the course.

Note

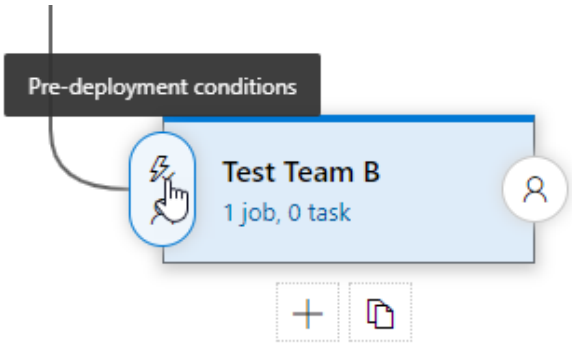
The same artifact sources move through each of the stages.

The lightning bolt icon on each stage shows that we can set a trigger as a pre-deployment condition. The person icon on both ends of a stage shows that we can have pre and post-deployment approvers.

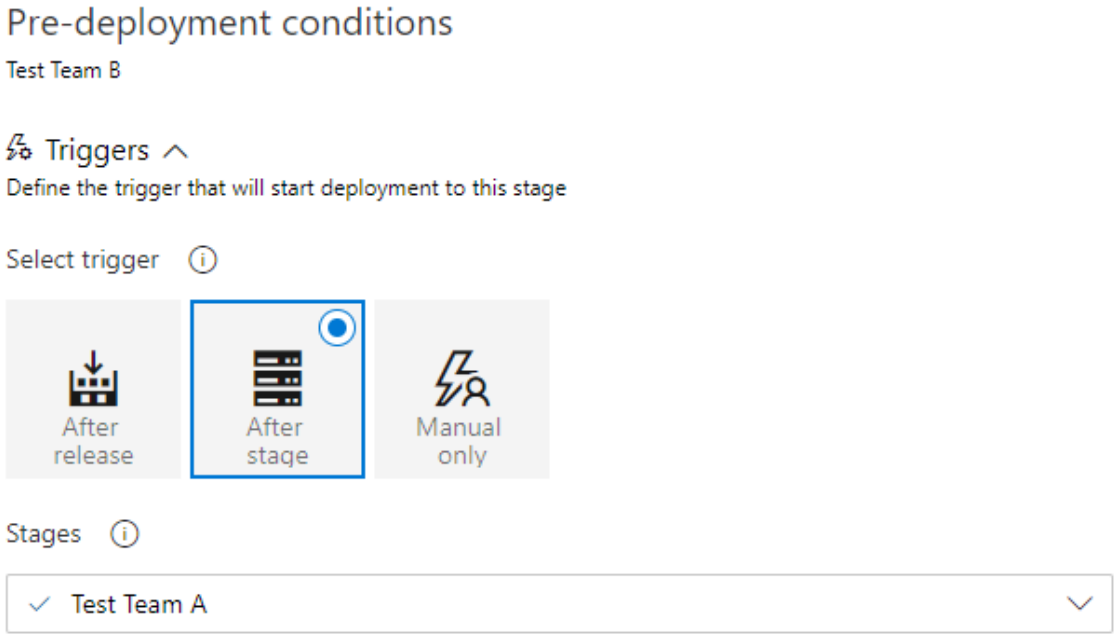
Concurrent stages

You'll notice that now, we have all the stages one after each other in a sequence. It's also possible to have concurrent stages. Let's see an example.

6. Click the **Test** stage, and on the stage properties pane, set the **Stage name** to **Test Team A** and close the pane.
7. Hover over the **Test Team A** stage and click the **Clone** icon that appears to create a new cloned stage.
8. Click the **Copy of Test Team A** stage, and on the stage properties pane, set **Stage name** to **Test Team B** and close the pane.
9. Click the **Pre-deployment conditions** icon (that is, the lightning bolt) on **Test Team B** to open the pre-deployment settings.



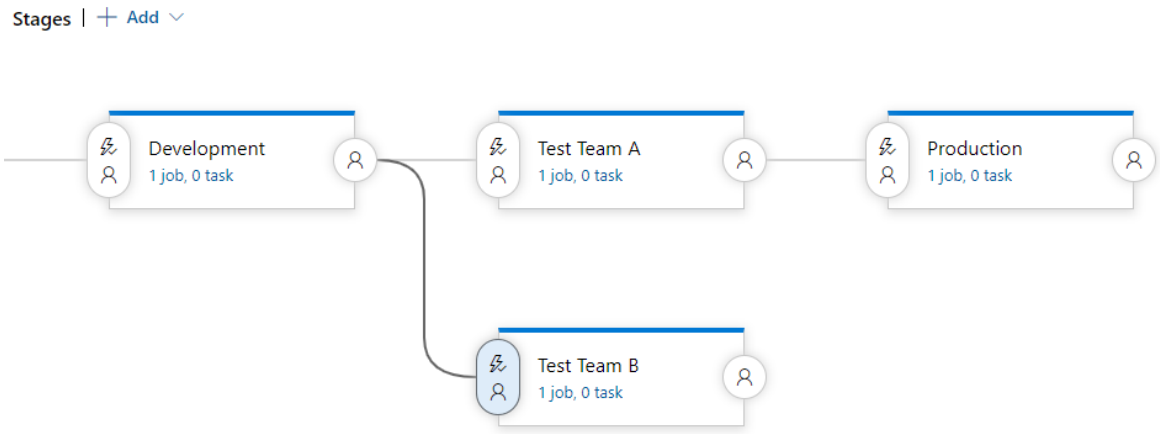
10. In the Pre-deployment conditions pane, the stage can be triggered in three different ways:



The stage can immediately follow Release. (That is how the Development stage is currently configured). It can require manual triggering. Or, more commonly, it can follow another stage. Now, it's following **Test Team A,** but that's not what we want.

11. Choose Development from the **Stages** drop-down list**,** uncheck **Test Team A,** and then close the pane.

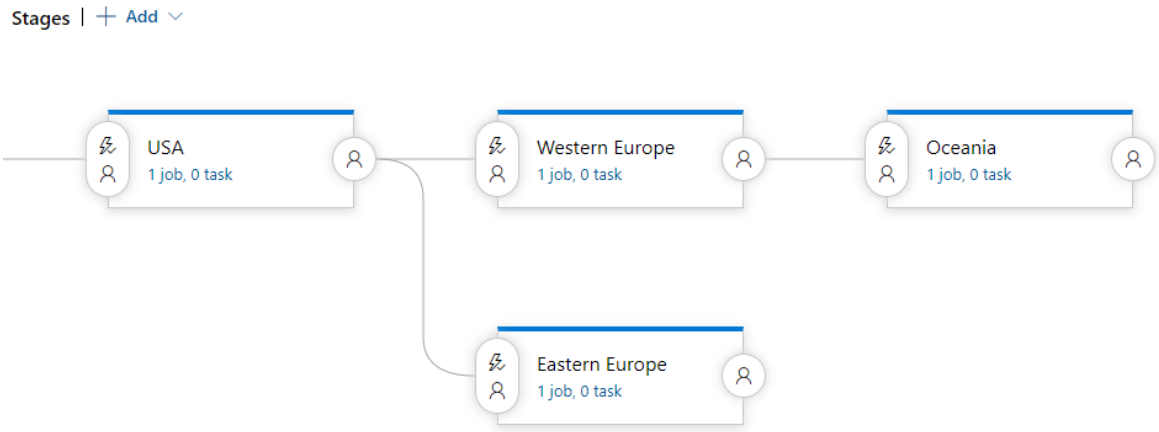
    We now have two concurrent Test stages.



Stage versus Environment

You may have wondered why these items are called **Stages** and not **Environments** .

In the current configuration, we're using them for different environments. But it's not always the case. Here's a deployment strategy based upon regions instead:



Azure Pipelines are configurable and support a wide variety of deployment strategies. The name **Stages** is a better fit than **Environment** even though the stages can be used for environments.

Let's give the pipeline a better name and save the work.

12. At the top of the screen, hover over the **New release pipeline** name and click it to edit the name when a pencil appears. Type **Release to all environments** as the name and hit enter or click elsewhere on the screen.

13. For now, save the environment-based release pipeline you've created by clicking **Save.** Then, **click OK** in the Save dialog box.

Continue

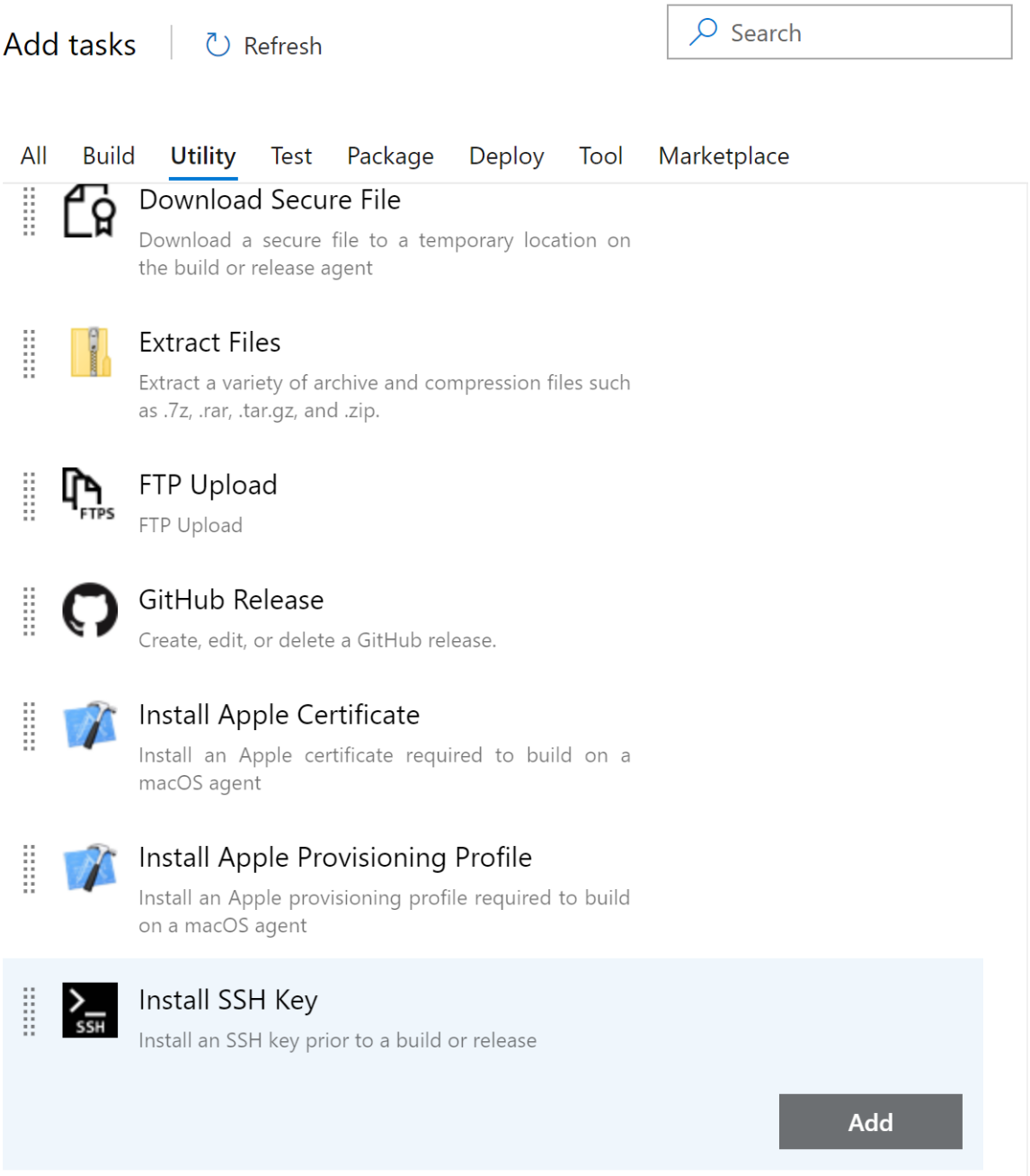
Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Explore build and release tasks

Completed

- 4 minutes

A build and release platform requires executing any number of repeatable actions during the build process. Tasks are units of executable code used to do selected actions in a specified order.



Add steps to specify what you want to build. The tests you want to run and all the other steps needed to complete the build process.

There are steps for building, testing, running utilities, packaging, and deploying.

If a task isn't available, you can find numerous community tasks in the marketplace.

Jenkins, Azure DevOps, and Atlassian have an extensive marketplace where other tasks can be found.

Links

For more information, see also:

- Task types & usage
- Tasks for Azure
- Atlassian marketplace
- Jenkins Plugins

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Explore custom build and release tasks

Completed

- 3 minutes

Instead of using out-of-the-box tasks, a command line, or a shell script, you can also use your custom build and release task.

By creating your tasks, the tasks are available publicly or privately to everyone you share them with.

Creating your task has significant advantages.

- You get access to variables that are otherwise not accessible.
- You can use and reuse a secure endpoint to a target server.
- You can safely and efficiently distribute across your whole organization.
- Users don't see implementation details.

For more information, see Add a build or release task .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Explore release jobs

Completed

- 4 minutes

You can organize your build or release pipeline into jobs. Every build or deployment pipeline has at least one job.

A job is a series of tasks that run sequentially on the same target. It can be a Windows server, a Linux server, a container, or a deployment group.

A release job is executed by a build/release agent. This agent can only run one job at the same time.

You specify a series of tasks you want to run on the same agent during your job design.

When the build or release pipeline is triggered at runtime, each job is dispatched to its target as one or more.

A scenario that speaks to the imagination, where Jobs plays an essential role, is the following.

Assume that you built an application with a backend in .NET, a front end in Angular, and a native IOS mobile App. It might be developed in three different source control repositories triggering three other builds and delivering three other artifacts.

The release pipeline brings the artifacts together and wants to deploy the backend, frontend, and Mobile App all together as part of one release.

The deployment needs to take place on different agents.

If an IOS app needs to be built and distributed from a Mac, the angular app is hosted on Linux, so best deployed from a Linux machine.

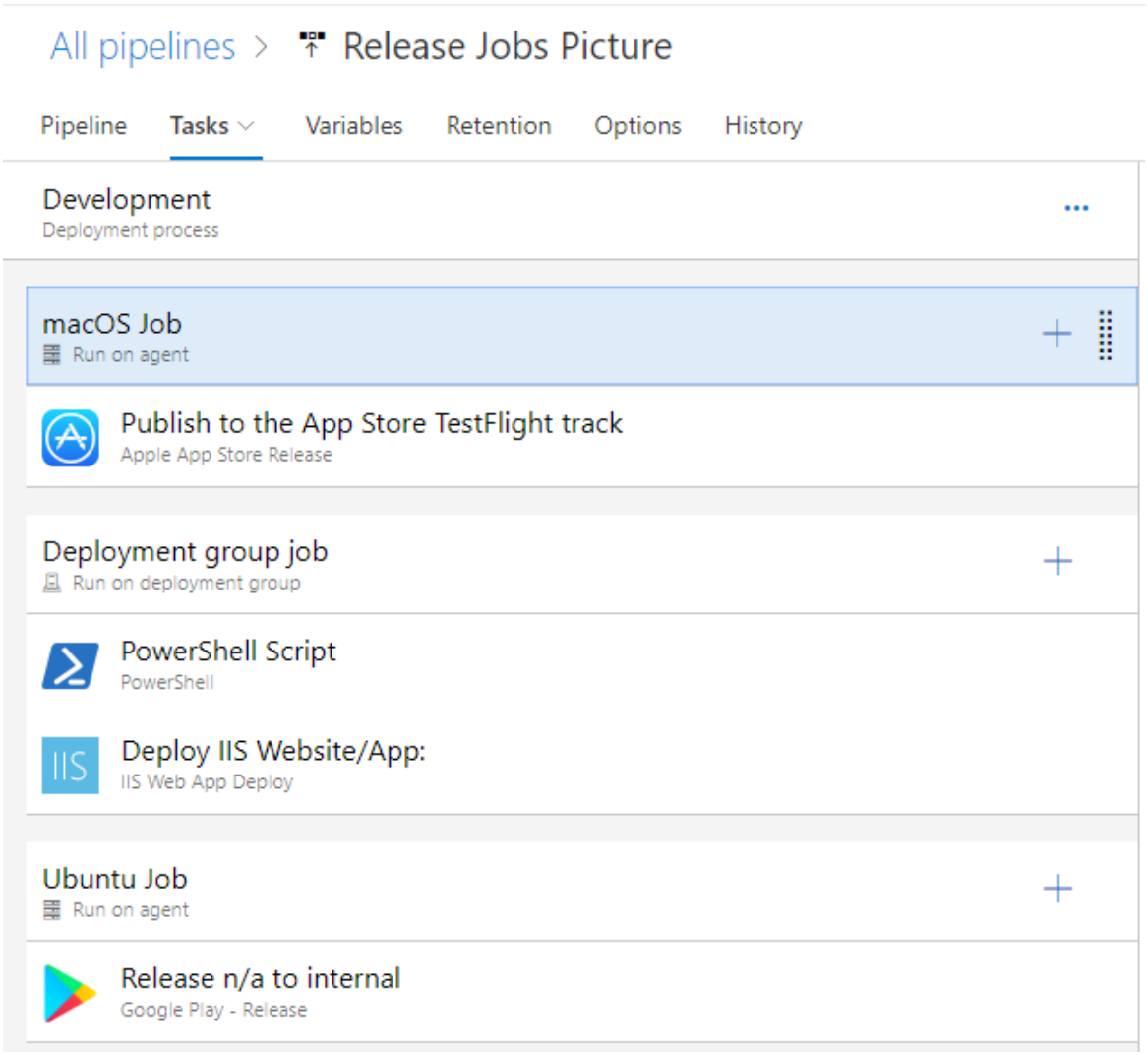
The backend might be deployed from a Windows machine.

Because you want all three deployments to be part of one pipeline, you can define multiple Release Jobs targeting the different agents, servers, or deployment groups.

By default, jobs run on the host machine where the agent is installed.

It's convenient and typically well suited for projects just beginning to adopt continuous integration (CI).

Over time, you may want more control over the stage where your tasks run.



For more information, see Jobs in Azure Pipelines .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Configure Pipelines as Code with YAML

Completed

- 60 minutes

Estimated time: 60 minutes.

Lab files: none.

Scenario

Many teams prefer to define their build and release pipelines using YAML. This allows them to access the same pipeline features as those using the visual designer but with a markup file that can be managed like any other source file. YAML build definitions can be added to a project by simply adding the corresponding files to the repository's root. Azure DevOps also provides default templates for popular project types and a YAML designer to simplify the process of defining build and release tasks.

Objectives

After completing this lab, you'll be able to:

- Configure CI/CD pipelines as code with YAML in Azure DevOps.

Requirements

- This lab requires **Microsoft Edge** or an Azure DevOps-supported browser .
- **Set up an Azure DevOps organization:** If you don't already have an Azure DevOps organization that you can use for this lab, create one by following the instructions available at Create an organization or project collection .
- Identify an existing Azure subscription or create a new one.
- Verify that you have a Microsoft account or a Microsoft Entra account with the Owner role in the Azure subscription and the Global Administrator role in the Microsoft Entra tenant associated with the Azure subscription. For details, refer to List Azure role assignments using the Azure portal and View and assign administrator roles in Microsoft Entra ID .

Exercises

During this lab, you'll complete the following exercises:

- Exercise 0: Configure the lab prerequisites.
- Exercise 1: Configure CI/CD Pipelines as Code with YAML in Azure DevOps.
- Exercise 2: Configure Environment settings for CI/CD Pipelines as Code with YAML in Azure DevOps.
- Exercise 3: Remove the Azure lab resources.

Launch Exercise

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Knowledge check

Completed

- 5 minutes

Choose the best response for each question. Then select **Check your answers** .

Check your knowledge

1.

Which of the following choices is a job type you can run?

○

Multi-configuration.

○

Multi-job.

○

Multi-pool.

2.

Which of the following choices is how many deployment jobs can be run concurrently by a single agent?

○

3.

○

5.

○

1.

3.

Which of the following choices describes the job type correctly?

○

None: Tasks will run on a single agent or more agents in parallel.

○

Multi-configuration: Run the same set of tasks with a single configuration per agent.

Multi-agent: Run the same set of tasks on multiple agents using the specified number of agents.

Check your answers

You must answer all questions before checking your work.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Summary

Completed

- 1 minute

This module described Azure Pipelines capabilities, build and release tasks, and multi-configuration and multi-agent differences.

You learned how to describe the benefits and usage of:

- Explain the terminology used in Azure DevOps and other Release Management Tooling.
- Describe what a Build and Release task is, what it can do, and some available deployment tasks.
- Implement release jobs.
- Differentiate between multi-agent and multi-configuration release jobs.

Learn more

- Release pipelines - Azure Pipelines | Microsoft Docs .
- Build and Release Tasks - Azure Pipelines | Microsoft Docs .
- Jobs in Azure Pipelines and TFS - Azure Pipelines | Microsoft Docs .
- Configure and pay for parallel jobs - Azure DevOps | Microsoft Docs .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Explore release recommendations

Introduction

Completed

- 1 minute

This module explores the critical release strategy recommendations organizations must consider when designing automated deployments.

It explains how to define components of a release pipeline and artifact sources, creates approval, and configure release gates.

Learning objectives

After completing this module, students and professionals can:

- Explain things to consider when designing your release strategy.
- Define the components of a release pipeline and use artifact sources.
- Create a release approval plan.
- Implement release gates.

Prerequisites

- Understanding of what DevOps is and its concepts.
- Familiarity with version control principles is helpful but isn't necessary.
- Beneficial to have experience in an organization that delivers software.
- For some exercises, you need to create an Azure DevOps Organization and a Team Project. If you don't have it, see Create an organization - Azure DevOps .
- If you already have your organization created, use the Azure DevOps Demo Generator and create a new Team Project called "Parts Unlimited" using the template "PartsUnlimited." Or feel free to create a blank project. See Create a project - Azure DevOps
- Complete the previous module's walkthroughs.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Understand the delivery cadence and three types of triggers

Completed

- 1 minute

Release and stages make use of triggers. There are three types of triggers we recognize.

Continuous deployment trigger

You can set up this trigger on your release pipeline. Once you do that, your release pipeline will trigger every time a build completes and creates a new release.

Scheduled triggers

It allows you to set up a time-based manner to start a new release—for example, every night at 3:00 AM or 12:00 PM. You can have one or multiple daily schedules, but it will always run at this specific time.

Manual trigger

With a manual trigger, a person or system triggers the release based on a specific event. When it's a person, it probably uses some UI to start a new release. When it's an automated process, some events will likely occur. You can trigger the release from another system using the automation engine, which is usually part of the release management tool.

For more information, see also:

- Release and Stage triggers .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Exercise - select your delivery and deployment cadence

Completed

- 30 minutes

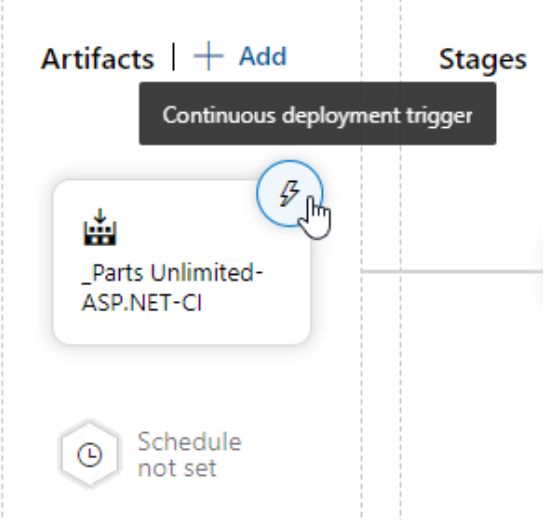
In this exercise, you'll investigate Delivery Cadence.

Steps

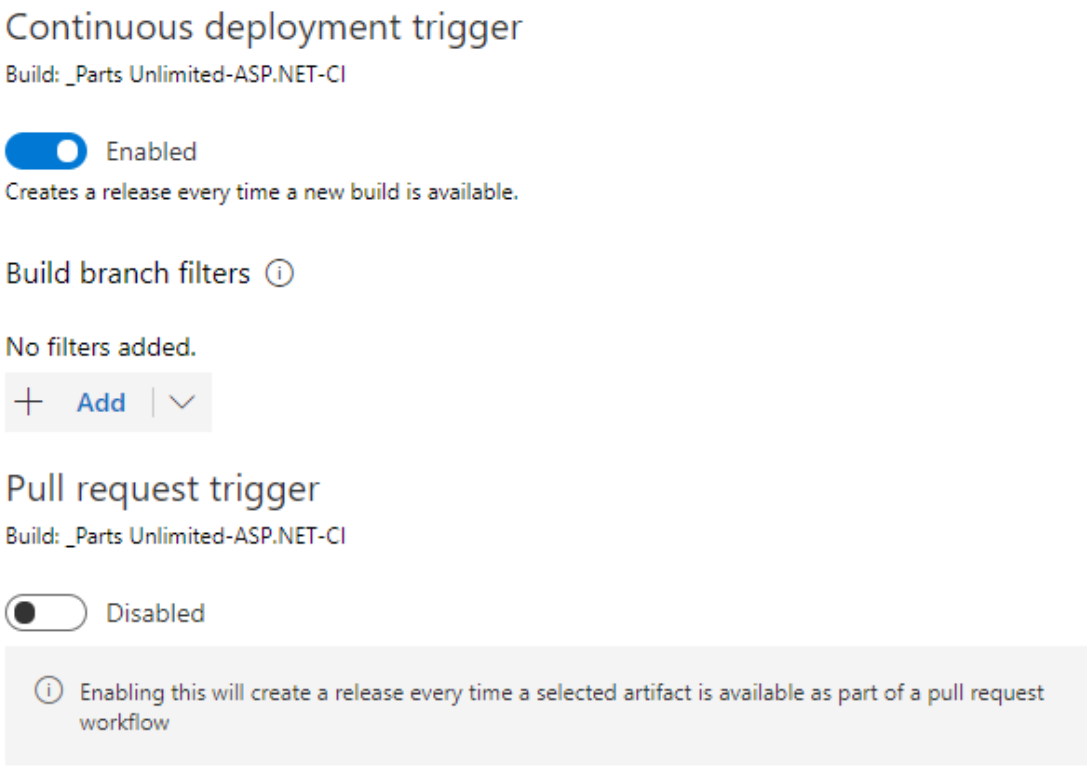
Let's look at when our release pipeline is used to create deployments. Mainly, it will involve the use of triggers.

When we refer to deployment, we refer to each stage. Each stage can have its triggers that determine when the deployment occurs.

1. Click the lightning bolt on the **_Parts Unlimited-ASP.NET-CI** artifact.



2. In the Continuous Deployment trigger pane, click the **Disabled** option to enable continuous deployment. It will then say **Enabled** .



Once it's selected, every time a build completes, deployment of the release pipeline will start.

Note

You can filter which branches affect it; for example, you could choose the main branch or a particular feature branch.

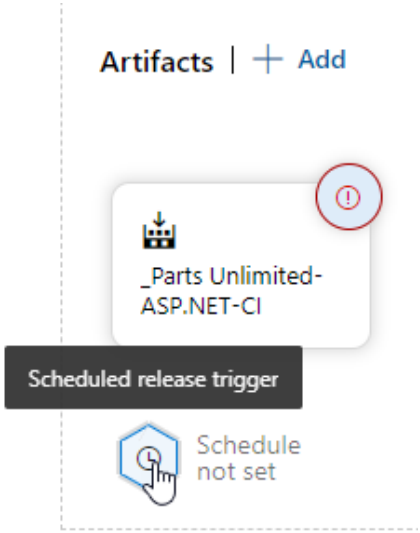
Scheduled deployments

You might not want to have a deployment start every time a build completes.

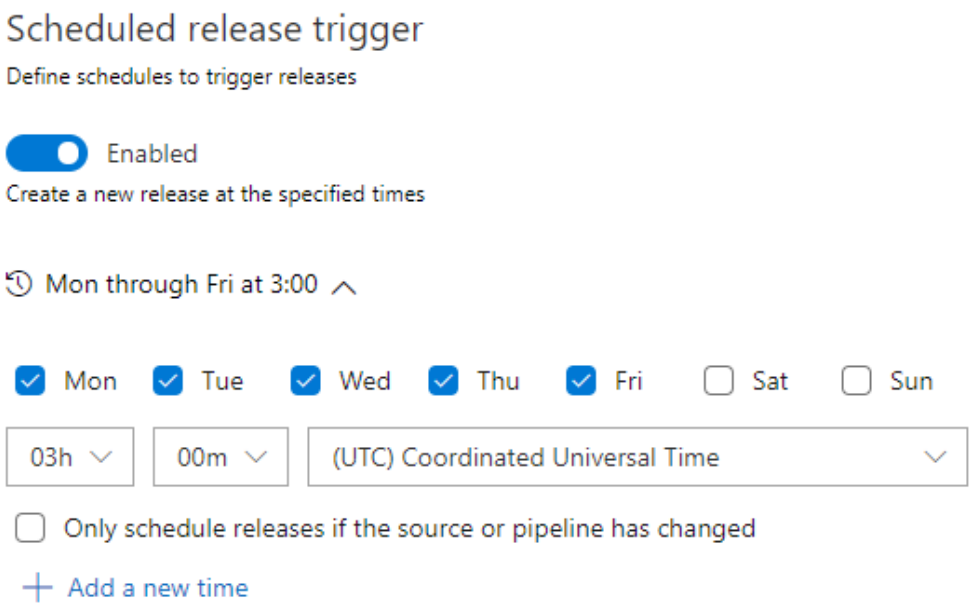
It might be disruptive to testers downstream if it was happening too often.

Instead, it might make sense to set up a deployment schedule.

3. Click on the **Scheduled release trigger** icon to open its settings.



4. In the Scheduled release trigger pane, click the **Disabled** option to enable the scheduled release. It will then say **Enabled** and extra options will appear.



You can see in the screenshot that a deployment using the release pipeline would now occur each weekday at 3 AM.

For example, it might be convenient when you share a stage with testers who work during the day.

You don't want to constantly deploy new versions to that stage while they're working. This setting would create a clean, fresh environment for them at 3 AM each weekday.

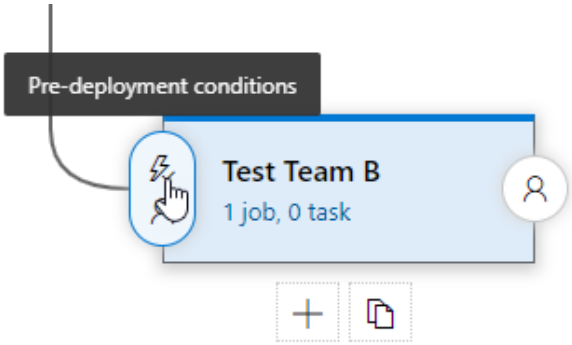
Note

The default timezone is UTC. You can change it to suit your local timezone, as it might be easier to work with when creating schedules.

5. For now, we don't need a scheduled deployment. Click the **Enabled** button again to turn off the scheduled release trigger and close the pane.

Pre-deployment triggers

6. Click the lightning bolt on the **Development** stage to open the pre-deployment conditions.



Note

Both artifact filters and schedules can be set at the pre-deployment for each stage rather than just at the artifact configuration level.

Deployment to any stage doesn't happen automatically unless you have chosen to allow that.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Explore release approvals

Completed

- 4 minutes

As we've described in the introduction, Continuous Delivery is all about delivering on-demand.

But, as we discussed in the differences between release and deployment, delivery, or deployment, it's only the technical part of the Continuous Delivery process.

It's all about how you can technically install the software on an environment, but it doesn't say anything about the process that needs to be in place for a release.

Release approvals don't control *how* but control *if* you want to deliver multiple times a day.

Manual approvals also suit a significant need. Organizations that start with Continuous Delivery often lack a certain amount of trust.

They don't dare to release without manual approval. After a while, when they find that the approval doesn't add value and the release always succeeds, the manual approval is often replaced by an automatic check.

Things to consider when you're setting up a release approval are:

- What do we want to achieve with the approval? Is it an approval that we need for compliance reasons? For example, we need to adhere to the four-eyes principle to get out SOX compliance. Or Is it an approval that we need to manage our dependencies? Or is it an approval that needs to be in place purely because we need a sign-out from an authority like Security Officers or Product Owners.
- Who needs to approve? We need to know who needs to approve the release. Is it a product owner, Security officer, or just someone that isn't the one that wrote the code? It's essential because the approver is part of the process. They're the ones that can delay the process if not available. So be aware of it.
- When do you want to approve? Another essential thing to consider is when to approve. It's a direct relationship with what happens after approval. Can you continue without approval? Or is everything on hold until approval is given. By using scheduled deployments, you can separate approval from deployment.

Although manual approval is a great mechanism to control the release, it isn't always helpful.

On many occasions, the check can be done at an earlier stage.

For example, it's approving a change that has been made in Source Control.

Scheduled deployments have already solved the dependency issue.

You don't have to wait for a person in the middle of the night. But there's still a manual action involved.

If you want to eliminate manual activities but still want control, you start talking about automatic approvals or release gates.

- Release approvals and gates overview .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Exercise - set up manual approvals

Completed

- 30 minutes

In this exercise, you'll investigate Manual Approval.

Steps

Let's now look at when our release pipeline needs manual approval before the deployment of a stage starts or manual approval that the deployment is completed as expected.

While DevOps is all about automation, manual approvals are still helpful. There are many scenarios where they're needed. For example, a product owner might want to sign out a release before it moves to production.

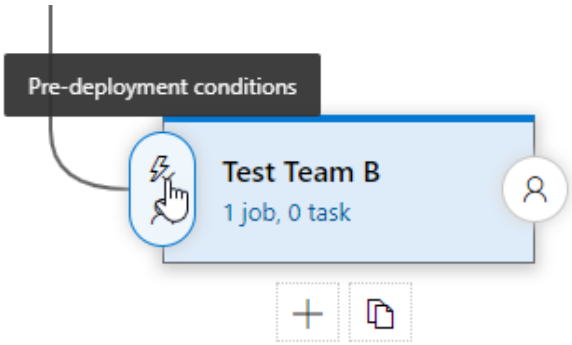
Or the scrum team wants to ensure that no new software is deployed to the test environment before someone signs off on it because they might need to find an appropriate time if it's constantly in use.

This can help to gain trust in the DevOps processes within the business.

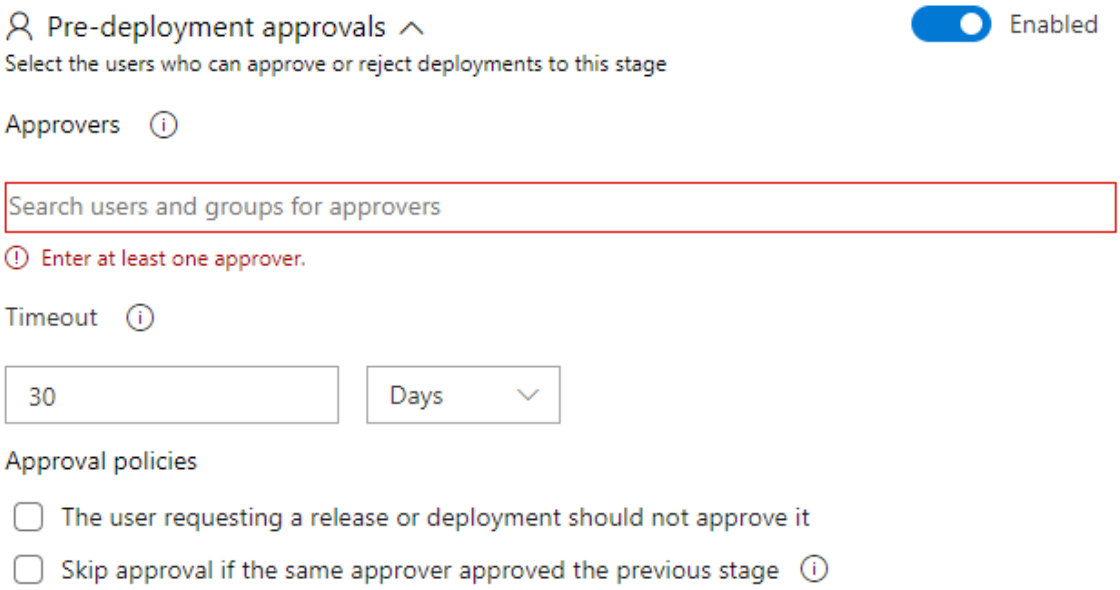
Even if the process is later automated, people might still want manual control until they become comfortable with the processes. Explicit manual approvals can be a great way to achieve that.

Let's try one.

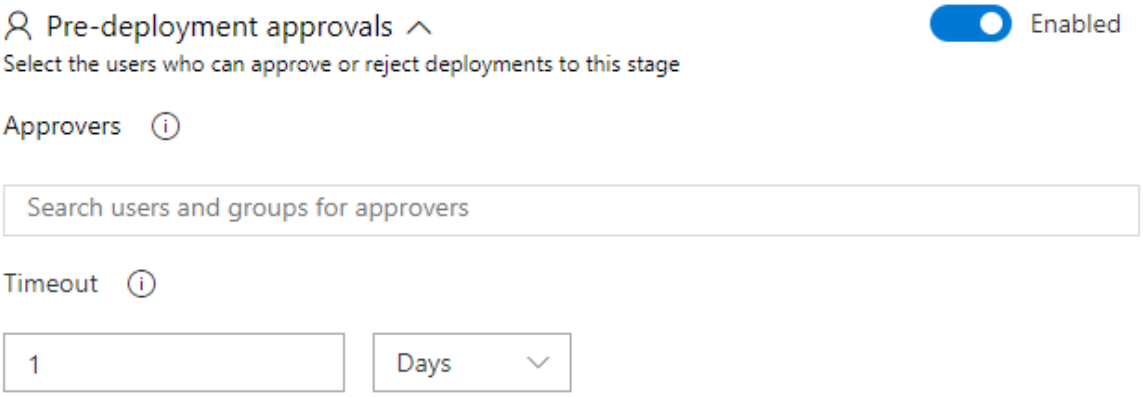
1. Click the pre-deployment conditions icon for the **Development** stage to open the settings.



2. Click the **Disabled** button in the Pre-deployment approvals section to enable it.



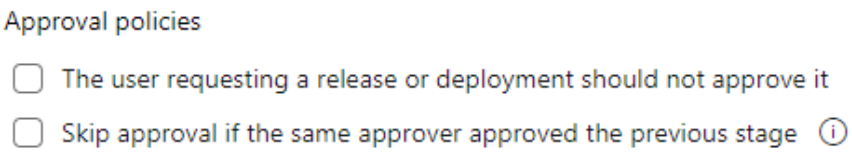
3. In the **Approvers** list, find your name and select it. Then set the **Timeout** to **1 Day** .



Note

Approvers is a list, not just a single value. If you add more than one person to the list, you can also choose if they need to approve in sequence or if either or both approvals are required.

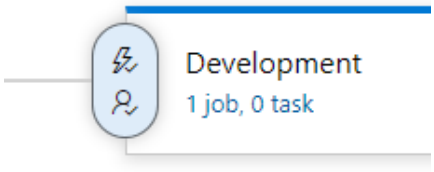
4. Take record of the approver policy options that are available:



It's prevalent not to allow a user who requests a release or deployment to approve it.

We're the only approver in this case, so we'll leave that unchecked.

5. Close the Pre-deployment conditions pane and notice a checkmark beside the person in the icon.

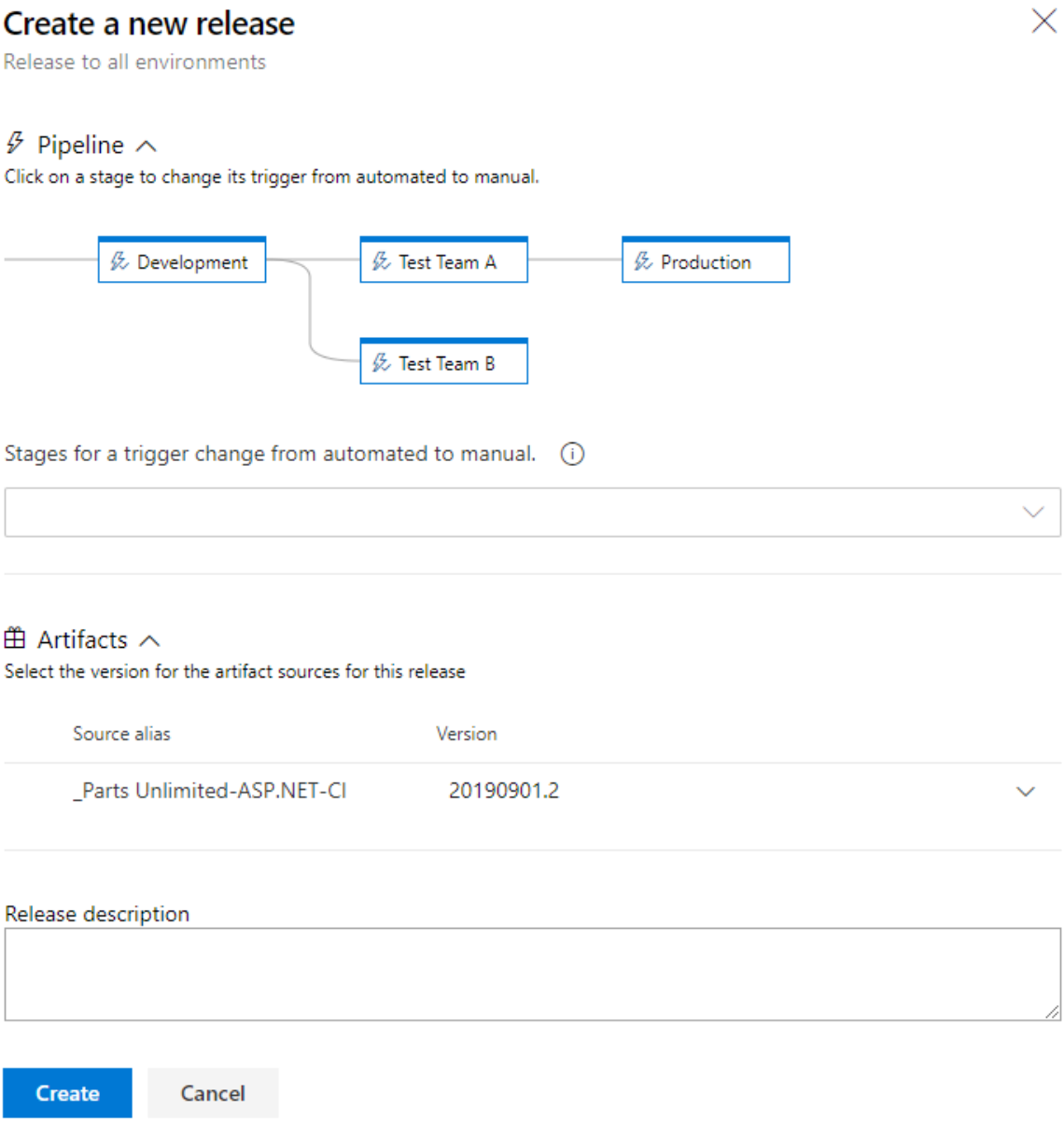


Test the approval

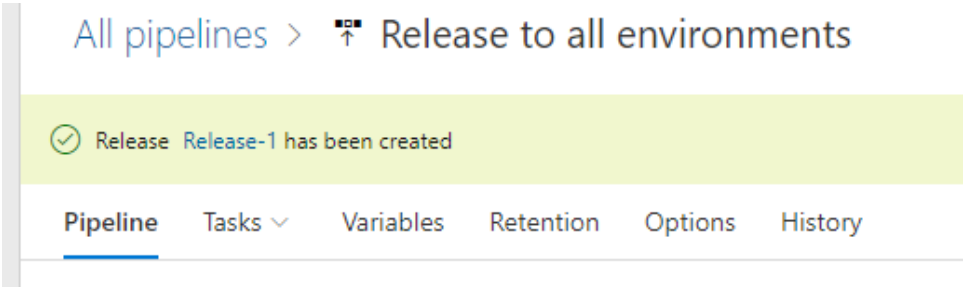
Now it's time to see what happens when approval is required.

6. Click **Save** to save the work, and **OK** on the popup window.
7. Click **Create release** to start the release process.

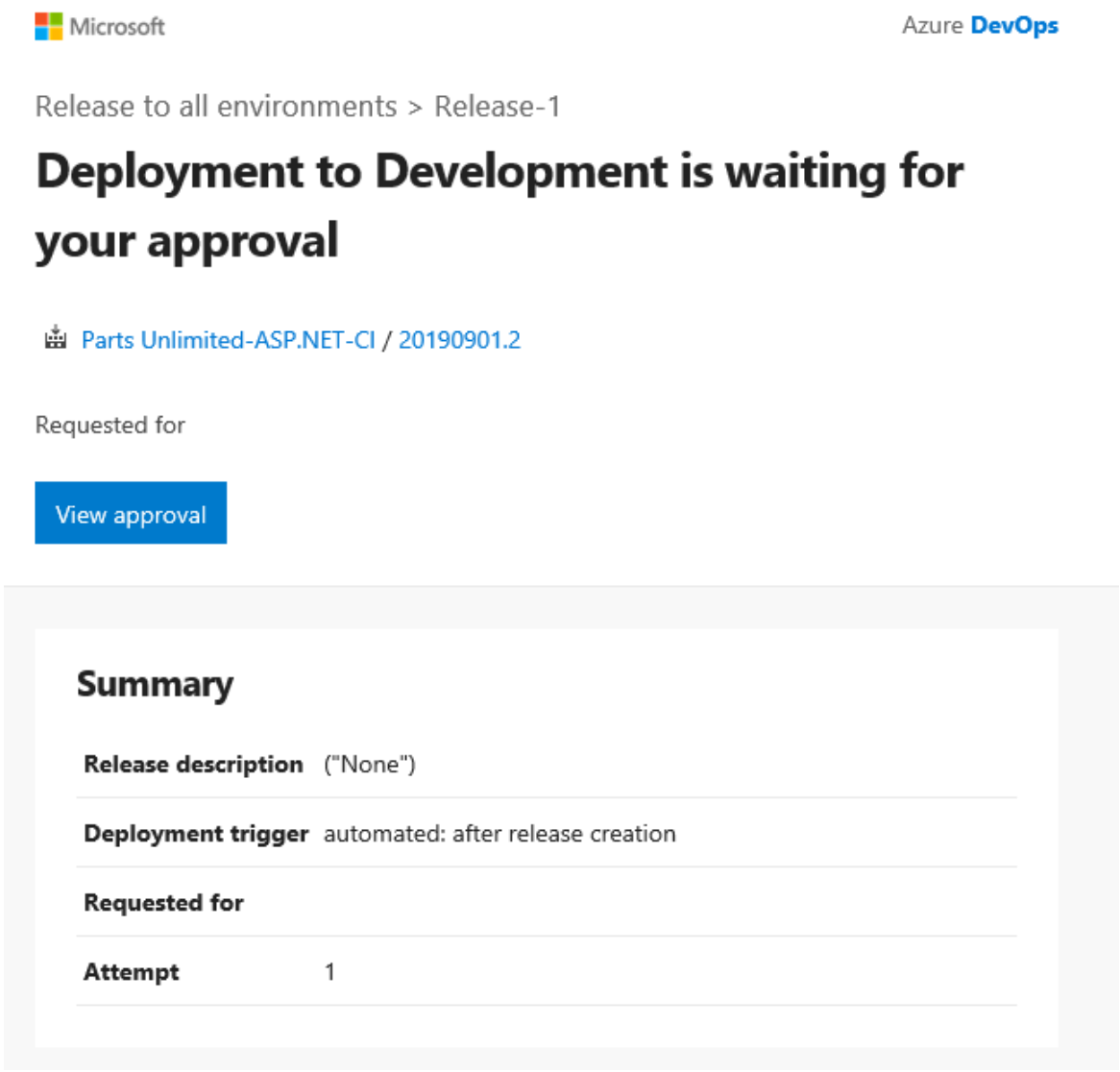
8. See the available options in the **Create a new release** pane, then click **Create** .



9. In the upper left of the screen, you can see that a release has been created.

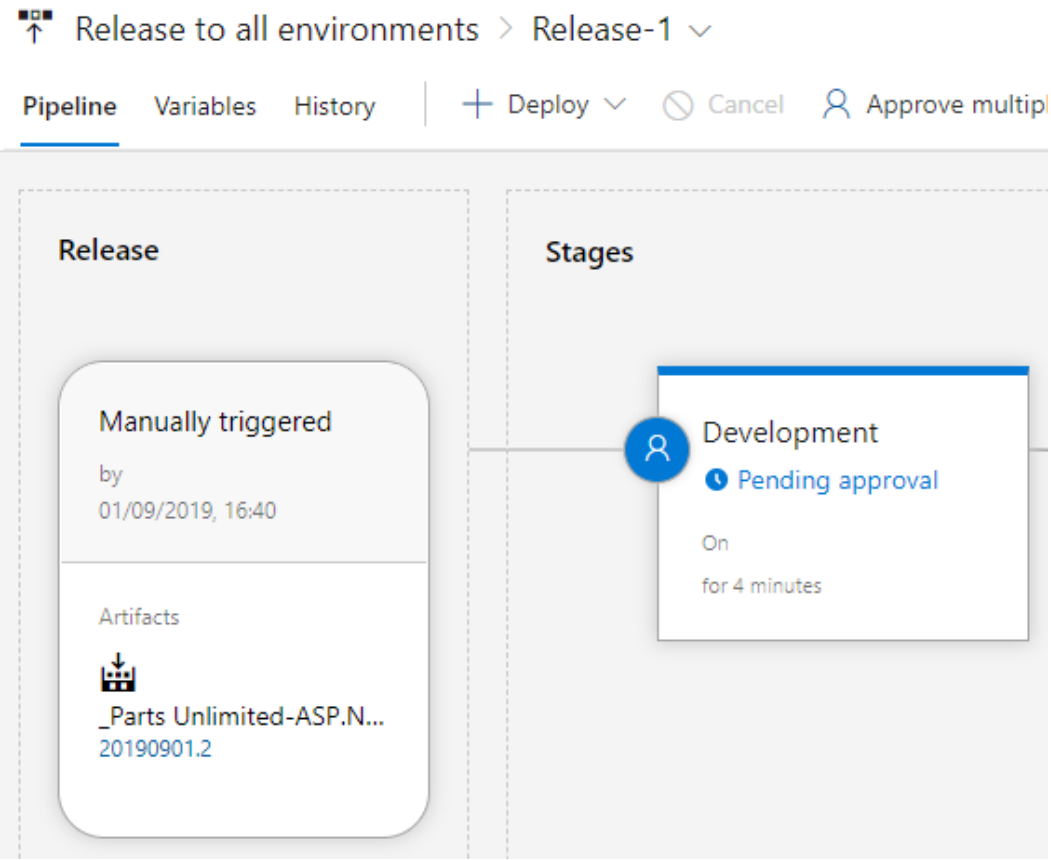


10. An email should have been received at this point, indicating that approval is required.



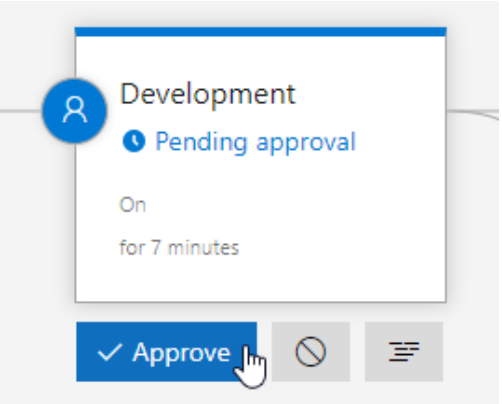
At this point, you could click the link in the email, but instead, we'll navigate within Azure DevOps to see what's needed.

11. Click on the **Release 1 Created** link (or whatever number it is for you) in the area we looked at in Step 9. We're then taken to a screen that shows the status of the release.



You can see that a release has been manually triggered and that the Development stage is waiting for approval. As an approver, you can now do that approval.

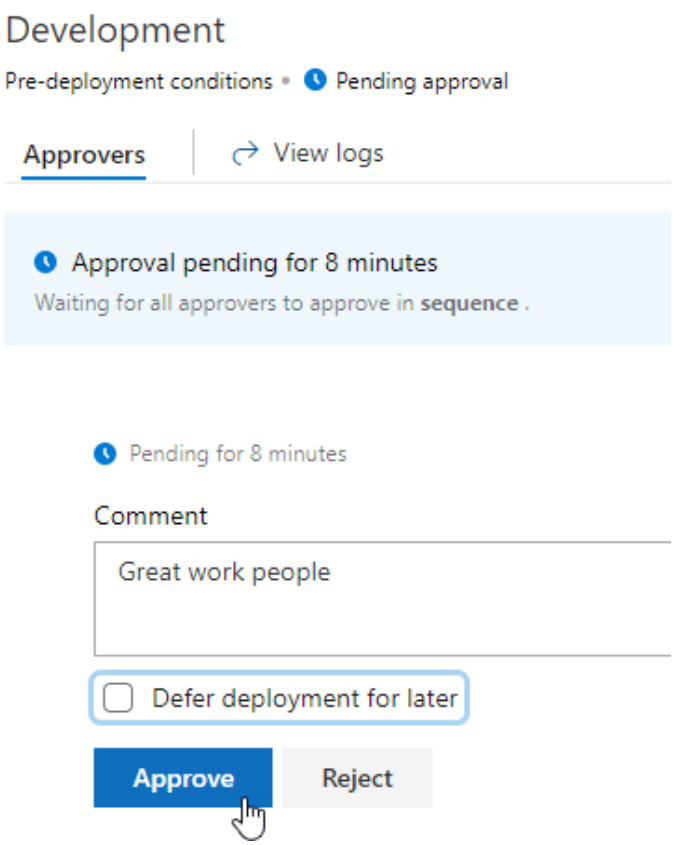
12. Hover over the **Development** stage and click the **Approve** icon that appears.



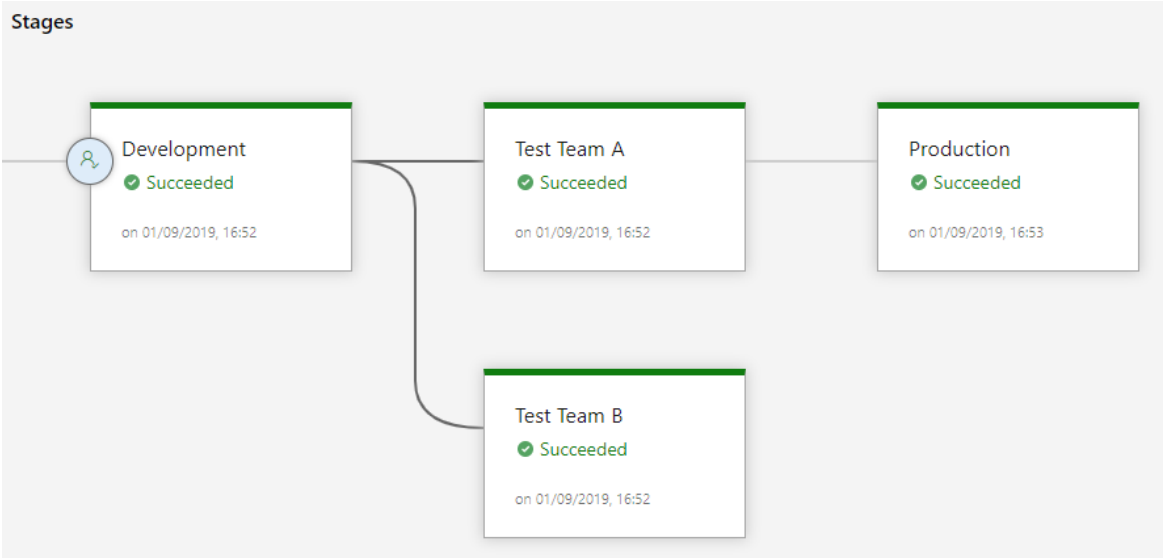
Note

Options to cancel the deployment or to view the logs are also provided at this point.

13. In the Development approvals window, add a comment and click **Approve** .



The deployment stage will then continue. Watch as each stage proceeds and succeeds.



Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Explore release gates

Completed

- 4 minutes

Release gates give you more control over the start and completion of the deployment pipeline.

They're often set up as pre-deployment and post-deployment conditions.

In many organizations, there are so-called dependency meetings.

It's a planning session where the release schedule of dependent components is discussed.

Think of downtime of a database server or an update of an API.

It takes much time and effort, and the only thing needed is a signal if the release can continue.

Instead of having this meeting, you can create a mechanism where people press a button on a form when the release can't advance.

When the release starts, it checks the state of the gate by calling an API. If the "gate" is open, we can continue. Otherwise, we'll stop the release.

By using scripts and APIs, you can create your release gates instead of manual approval. Or at least extending your manual approval.

Other scenarios for automatic approvals are, for example.

- Incident and issues management. Ensure the required status for work items, incidents, and issues. For example, ensure that deployment only occurs if no bugs exist.
- Notify users such as legal approval departments, auditors, or IT managers about a deployment by integrating with approval collaboration systems such as Microsoft Teams or Slack and waiting for the approval to complete.
- Quality validation. Query metrics from tests on the build artifacts such as pass rate or code coverage and only deploy within required thresholds.
- Security scan on artifacts. Ensure security scans such as anti-virus checking, code signing, and policy checking for build artifacts have been completed. A gate might start the scan and wait for it to finish or check for completion.
- User experience relative to baseline. Using product telemetry, ensure the user experience hasn't regressed from the baseline state. The experience level before the deployment could be considered a baseline.
- Change management. Wait for change management procedures in a system such as ServiceNow complete before the deployment occurs.
- Infrastructure health. Execute monitoring and validate the infrastructure against compliance rules after deployment or wait for proper resource use and a positive

security report.

In short, approvals and gates give you more control over the start and completion of the deployment pipeline.

They can usually be set up as pre-deployment and post-deployment conditions, including waiting for users to approve or reject deployments manually and checking with other automated systems until specific requirements are verified.

Also, you can configure a manual intervention to pause the deployment pipeline and prompt users to carry out manual tasks, then resume or reject the deployment.

To find out more about Release Approvals and Gates, check these documents.

- Release approvals and gates overview .
- Release Gates .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Use release gates to protect quality

Completed

- 3 minutes

A quality gate is the best way to enforce a quality policy in your organization. It's there to answer one question: can I deliver my application to production or not?

A quality gate is located before a stage that is dependent on the outcome of a previous stage. A quality gate was typically something that a QA department monitored in the past.

They had several documents or guidelines, and they verified if the software was of a good enough quality to move on to the next stage.

When we think about Continuous Delivery, all manual processes are a potential bottleneck.

We need to reconsider the notion of quality gates and see how we can automate these checks as part of our release pipeline.

By using automatic approval with a release gate, you can automate the approval and validate your company's policy before moving on.

Many quality gates can be considered.

- No new blocker issues.
- Code coverage on new code greater than 80%.
- No license violations.

- No vulnerabilities in dependencies.
- No further technical debt was introduced.
- Is the performance not affected after a new release?
- Compliance checks
    - Are there work items linked to the release?
    - Is the release started by someone else as the one who commits the code?

Defining quality gates improves the release process, and you should always consider adding them.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Control Deployments using Release Gates

Completed

- 75 minutes

Estimated time: 75 minutes.

Lab files: none.

Scenario

This lab covers the configuration of the deployment gates and details how to use them to control the execution of Azure Pipelines. To illustrate their implementation, you'll configure a release definition with two environments for an Azure Web App. You'll deploy to the Canary environment only when there are no blocking bugs for the app and mark the Canary environment complete only when there are no active alerts in Application Insights of Azure Monitor.

A release pipeline specifies the end-to-end release process for an application to be deployed across various environments. Deployments to each environment are fully automated by using jobs and tasks. Ideally, you don't want new updates to the applications to be simultaneously exposed to all users. It's a best practice to expose updates in a phased manner, that is, expose them to a subset of users, monitor their usage, and expose them to other users based on the experience of the initial set of users.

Approvals and gates enable you to control the start and completion of the deployments in a release. You can wait for users to approve or reject deployments with approvals manually. Using release gates, you can specify application health criteria to be met before the release is promoted to the following environment. Before or after any environment deployment, all the specified gates are automatically evaluated until they pass or reach your defined timeout period and fail.

Gates can be added to an environment in the release definition from the pre-deployment conditions or the post-deployment conditions panel. Multiple gates can be added to the environment conditions to ensure all the inputs are successful for the release.

As an example:

- Pre-deployment gates ensure no active issues in the work item or problem management system before deploying a build to an environment.
- Post-deployment gates ensure no incidents from the app's monitoring or incident management system after being deployed before promoting the release to the following environment.

There are four types of gates included by default in every account.

- Invoke Azure Function: Trigger the execution of an Azure Function and ensures a successful completion.
- Query Azure Monitor alerts: Observe the configured Azure Monitor alert rules for active alerts.
- Invoke REST API: Make a call to a REST API and continues if it returns a successful response.
- Query work items: Ensure the number of matching work items returned from a query is within a threshold.

Objectives

After completing this lab, you'll be able to:

- Configure release pipelines.
- Configure release gates.
- Test release gates.

Requirements

- This lab requires **Microsoft Edge** or an Azure DevOps-supported browser .
- **Set up an Azure DevOps organization:** If you don't already have an Azure DevOps organization that you can use for this lab, create one by following the instructions available at Create an organization or project collection .
- Identify an existing Azure subscription or create a new one.
- Verify that you have a Microsoft or Microsoft Entra account with the Contributor or the Owner role in the Azure subscription. For details, refer to List Azure role assignments using the Azure portal .

Exercises

During this lab, you'll complete the following exercises:

- Exercise 0: Configure the lab prerequisites.

- Exercise 1: Configure the build pipeline.
- Exercise 2: Configure the release pipeline.
- Exercise 3: Configure release gates.
- Exercise 4: Test release gates.
- Exercise 5: Remove the Azure lab resources.

Launch Exercise

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Knowledge check

Completed

- 5 minutes

Choose the best response for each question. Then select **Check your answers** .

Check your knowledge

1.

Which of the following choices isn't a release trigger?

○

A manual trigger.

○

A continuous deployment trigger.

○

A feature trigger.

2.

Which of the following choices can you use to prevent deployment in Azure DevOps when a security testing tool finds a compliance problem?

○

Work Item.

○

Release Gate.

Manual trigger.

3.

Which of the following choices is a logical boundary in your release pipeline at which you can pause the pipeline and perform various checks?

Stage.

Trigger.

Quality Gate.

Check your answers

You must answer all questions before checking your work.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Summary

Completed

- 1 minute

This module explored the critical release strategy recommendations that organizations must consider when designing automated deployments.

It explained how to define components of a release pipeline and artifact sources, create approves, and configure release gates.

You learned how to describe the benefits and usage of:

- Explain things to consider when designing your release strategy.
- Define the components of a release pipeline and use artifact sources.
- Create a release approval plan.

- Implement release gates.

Learn more

- How Microsoft plans efficient workloads with DevOps - Azure DevOps | Microsoft Docs .
- Release engineering app development - Azure Architecture Center | Microsoft Docs .
- How Microsoft develops modern software with DevOps - Azure DevOps | Microsoft Docs .
- Control deployments by using approvals - Azure Pipelines | Microsoft Docs .
- Control deployments by using gates - Azure Pipelines | Microsoft Docs .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Provision and test environments

Introduction

Completed

- 1 minute

This module details target environment provisioning, service connections creation process, and test infrastructure setup. Learn how to configure functional test automation and run availability tests.

Learning objectives

After completing this module, students and professionals can:

- Provision and configure target environment.
- Deploy to an environment securely using a service connection.
- Configure functional test automation and run availability tests.
- Setup test infrastructure.

Prerequisites

- Understanding of what DevOps is and its concepts.
- Familiarity with version control principles is helpful but isn't necessary.
- Beneficial to have experience in an organization that delivers software.
- You must create an Azure DevOps Organization and a Team Project for some exercises. If you don't have it yet, see: Create an Organization - Azure DevOps .
    - If your organization is already created, use the Azure DevOps Demo Generator and create a new Team Project called "Parts Unlimited" using the template "PartsUnlimited". Or feel free to create a blank project. See Create a Project - Azure DevOps .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Provision and configure target environments

Completed

- 5 minutes

The release pipeline deploys software to a target environment. But it isn't only the software that will be deployed with the release pipeline.

If you focus on Continuous Delivery, Infrastructure as Code and spinning up Infrastructure as part of your release pipeline is essential.

When we focus on the deployment of the Infrastructure, we should first consider the differences between the target environments that we can deploy to:

- On-Premises servers.
- Cloud servers or Infrastructure as a Service (IaaS). For example, Virtual machines or networks.
- Platform as a Service (PaaS) and Functions as a Service (FaaS). For example, Azure SQL Database in both PaaS and serverless options.
- Clusters.
- Service Connections.

Let us dive a bit further into these different target environments and connections.

On-premises servers

In most cases, when you deploy to an on-premises server, the hardware and the operating system are already in place. The server is already there and ready.

In some cases, empty, but most of the time not. In this case, the release pipeline can only focus on deploying the application.

You might want to start or stop a virtual machine (Hyper-V or VMware).

The scripts you use to start or stop the on-premises servers should be part of your source control and delivered to your release pipeline as a build artifact.

Using a task in the release pipeline, you can run the script that starts or stops the servers.

To take it one step further and configure the server, you should look at technologies like PowerShell Desired State Configuration(DSC).

The product will maintain your server and keep it in a particular state. When the server changes its state, you can recover the changed configuration to the original configuration.

Integrating a tool like PowerShell DSC into the release pipeline is no different from any other task you add.

Infrastructure as a service

When you use the cloud as your target environment, things change slightly. Some organizations lift and shift from their on-premises servers to cloud servers.

Then your deployment works the same as an on-premises server. But when you use the cloud to provide you with Infrastructure as a Service (IaaS), you can use the power of the cloud to start and create servers when needed.

It's where Infrastructure as Code (IaC) starts playing a significant role.

Creating a script or template can make a server or other infrastructural components like a SQL server, a network, or an IP address.

By defining a template or using a command line and saving it in a script file, you can use that file in your release pipeline tasks to execute it on your target cloud.

The server (or another component) will be created as part of your pipeline. After that, you can run the steps to deploy the software.

Technologies like Azure Resource Manager are great for creating Infrastructure on demand.

Platform as a Service

When you move from Infrastructure as a Service (IaaS) to Platform as a Service (PaaS), you'll get the Infrastructure from the cloud you're running on.

For example: In Azure, you can create a Web application. The cloud arranges the server, the hardware, the network, the public IP address, the storage account, and even the web server.

The user only needs to take care of the web application on this Platform.

You only need to provide the templates instructing the cloud to create a WebApp. Functions as a Service(FaaS) or Serverless technologies are the same.

In Azure, it's called Azure Functions. You only deploy your application, and the cloud takes care of the rest. However, you must instruct the Platform (the cloud) to create a placeholder where your application can be hosted.

You can define this template in Azure Resource Manager. You can use the Azure CLI or command-line tools.

In all cases, the Infrastructure is defined in a script file and lives alongside the application code in source control.

Clusters

Finally, you can deploy your software to a cluster. A cluster is a group of servers that host high-scale applications.

When you run an Infrastructure as a Service cluster, you must create and maintain the cluster. It means that you need to provide the templates to create a cluster.

You must also ensure you roll out updates, bug fixes, and patches to your cluster. It's comparable with Infrastructure as a Service.

When you use a hosted cluster, you should consider it a Platform as a Service. You instruct the cloud to create the cluster, and you deploy your software to the cluster.

When you run a container cluster, you can use the container cluster technologies like AKS.

Service connections

When a pipeline needs resource access, you must often create service connections.

Summary

Whatever the technology you choose to host your application, your Infrastructure's creation or configuration should be part of your release pipeline and source control repository.

Infrastructure as Code is a fundamental part of Continuous Delivery, allowing you to create servers and environments on demand.

Links

- Desired State Configuration Overview.
- Azure Functions.
- Azure Resource Manager.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Exercise - set up service connections

Completed

- 30 minutes

In this exercise, you'll investigate Service Connections.

Note

To follow along with this walkthrough, you'll need an existing Azure subscription containing an existing storage account.

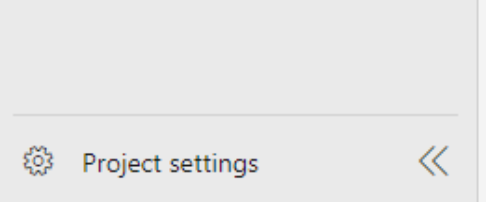
Steps

You can set up a service connection to environments to create a secure and safe connection to the environment you want to deploy.

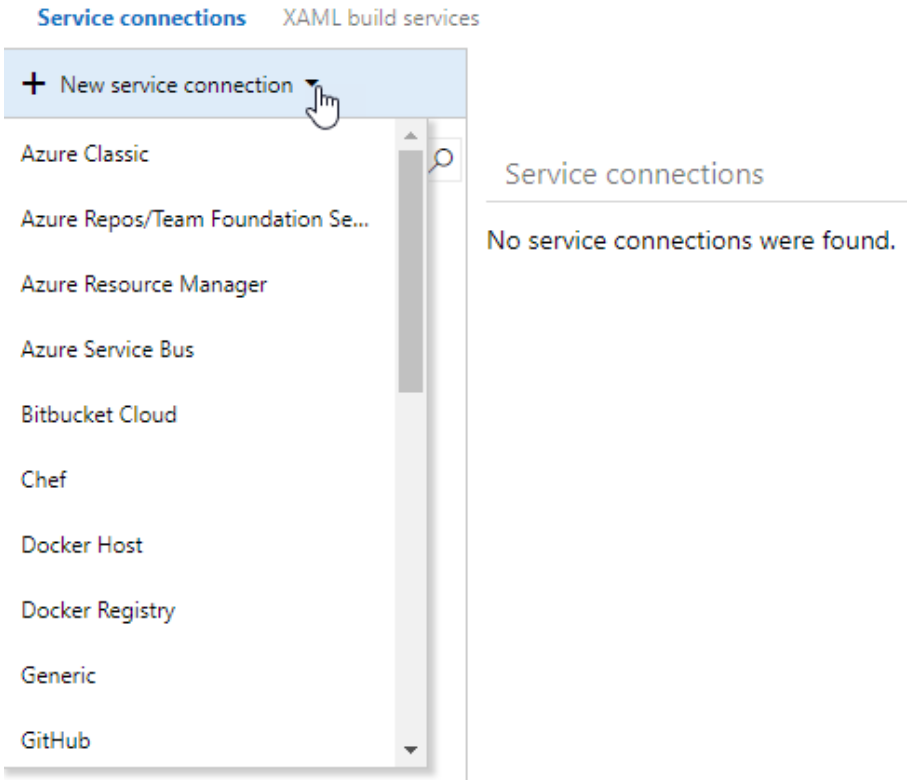
Service connections are also used to get resources from other places in a secure manner.

For example, you might need to get your source code from GitHub. Let's look at configuring a service connection to Azure in this case.

1. From the main menu in the **Parts Unlimited** project, click **Project settings** at the bottom of the screen.



2. In the Project Settings pane, from the **Pipelines** section, click **Service Connections** . Click the drop-down beside **+New service connection** .



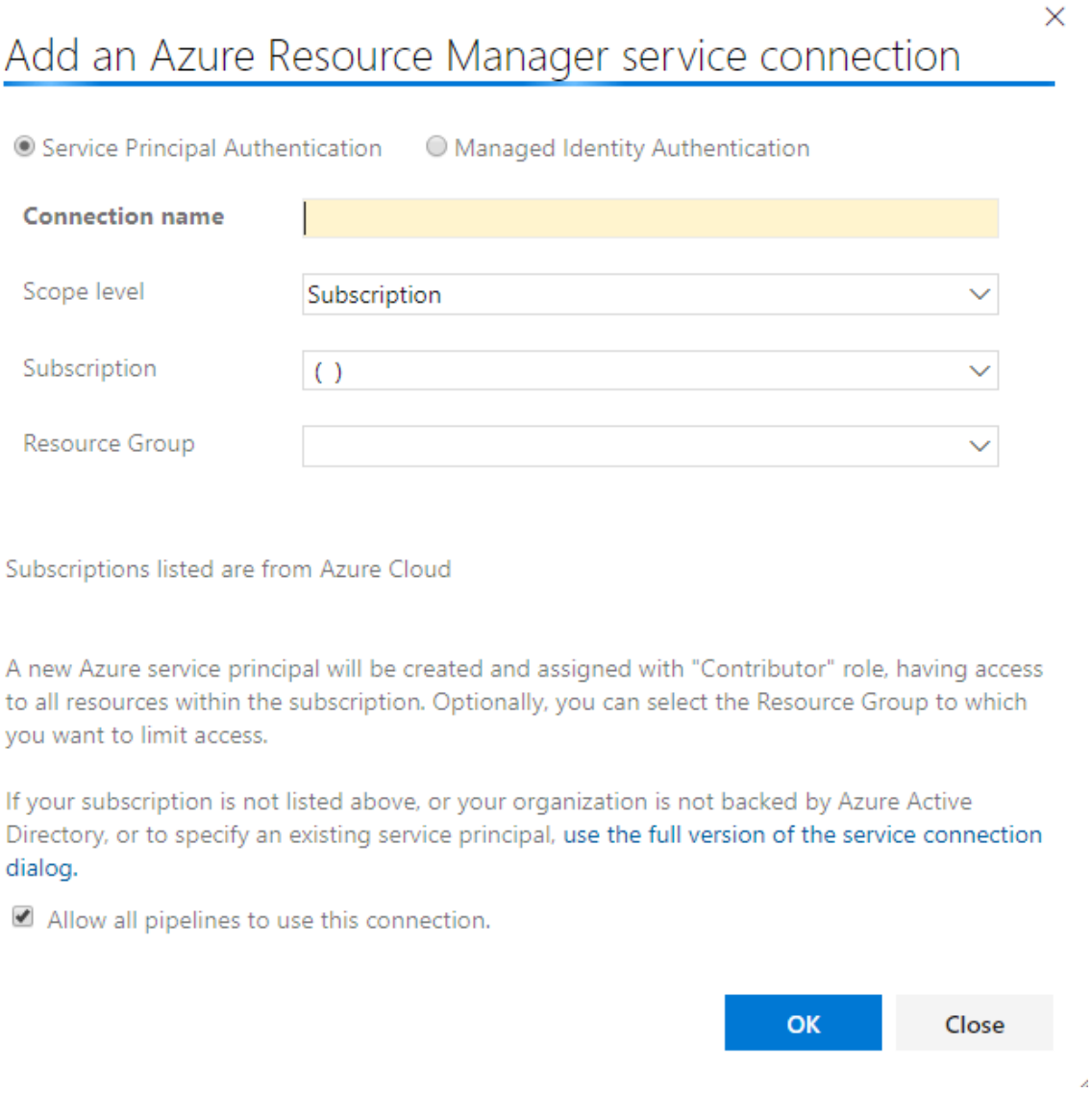
As you can see, there are many types of service connections. You can create a connection to:

- Apple App Store.
- Docker Registry

- Bitbucket.
- Azure Service bus.

In this case, we want to deploy a new Azure resource, so we'll use the Azure Resource Manager option.

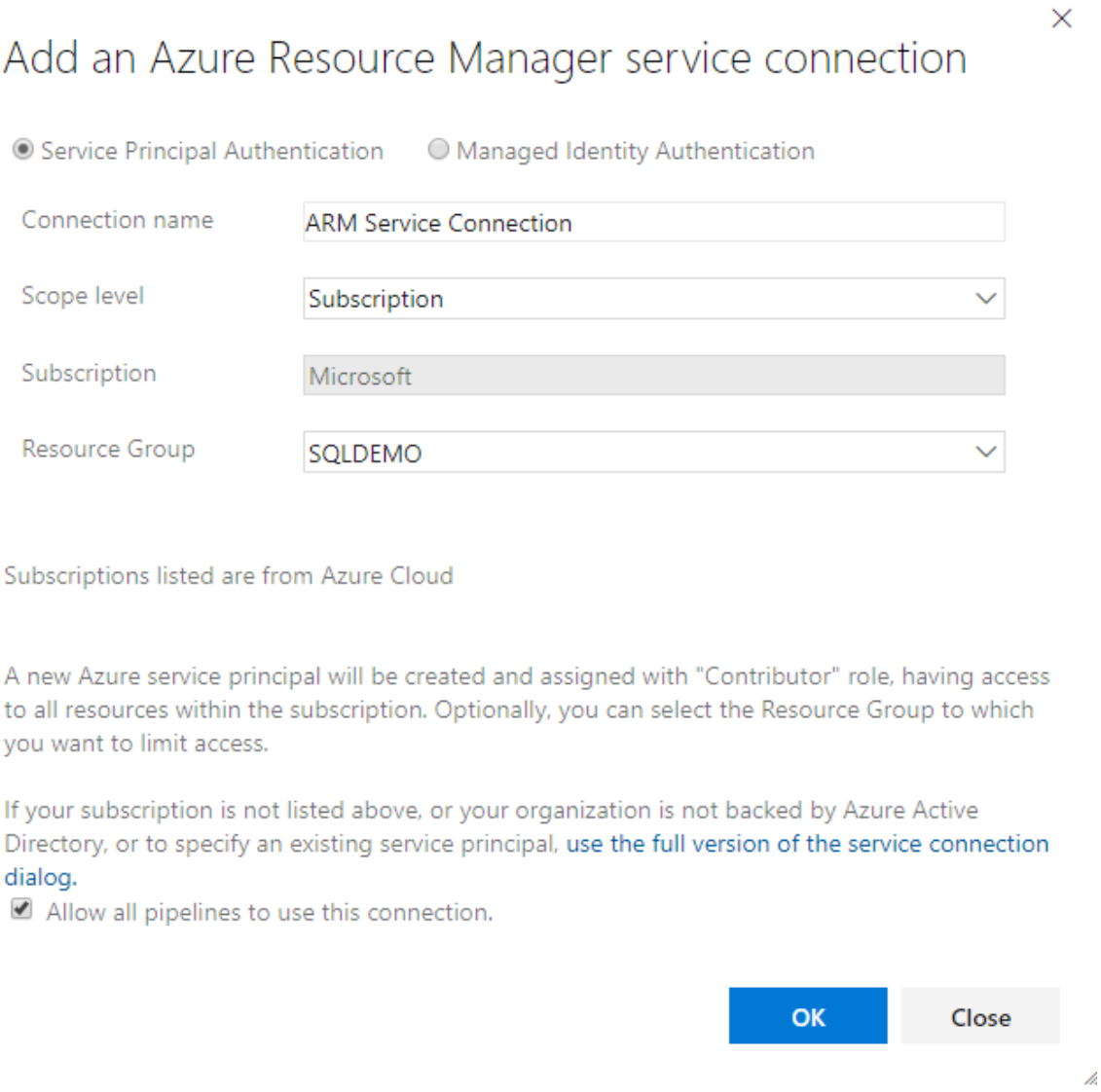
3. Click **Azure Resource Manager** to add a new service connection.



4. Set the **Connection name** to **Azure Resource Manager Service Connection** , click on an Azure **Subscription** , then select an existing **Resource Group** .

   Note

   You might be prompted to sign-in Azure at this point. If so, sign-in first.



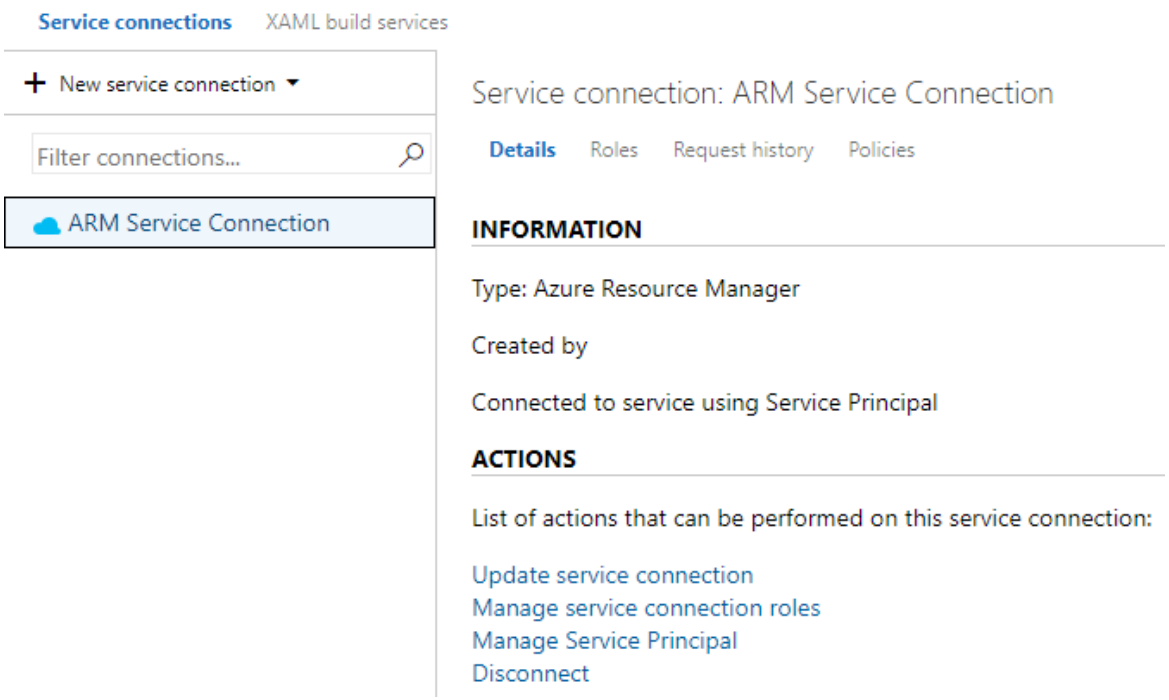
Notice that what we are creating is a **Service Principal** . We'll be using the Service Principal for authenticating to Azure. At the top of the window, there's an option to set up Managed Identity Authentication instead.

The Service Principal is a service account with only permissions in the specific subscription and resource group. It makes it a safe way to connect from the pipeline.

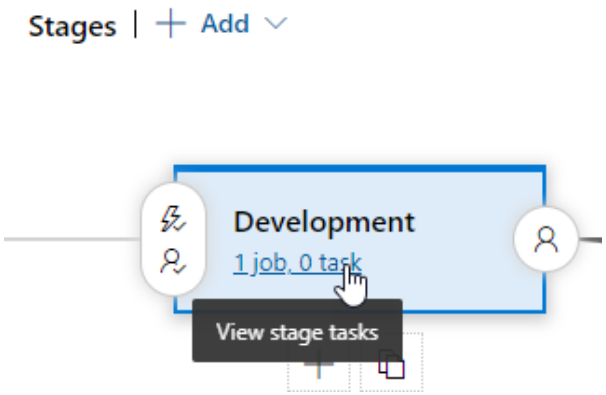
Important

When you create a service connection with Azure, the service principal gets a contributor role to the subscription or resource group. It's not enough to upload data to blob storage for the service principal. You must explicitly add the service principal to the Storage Blob Data Contributor role. Otherwise, the release gets failed with an authorization permission mismatch error.

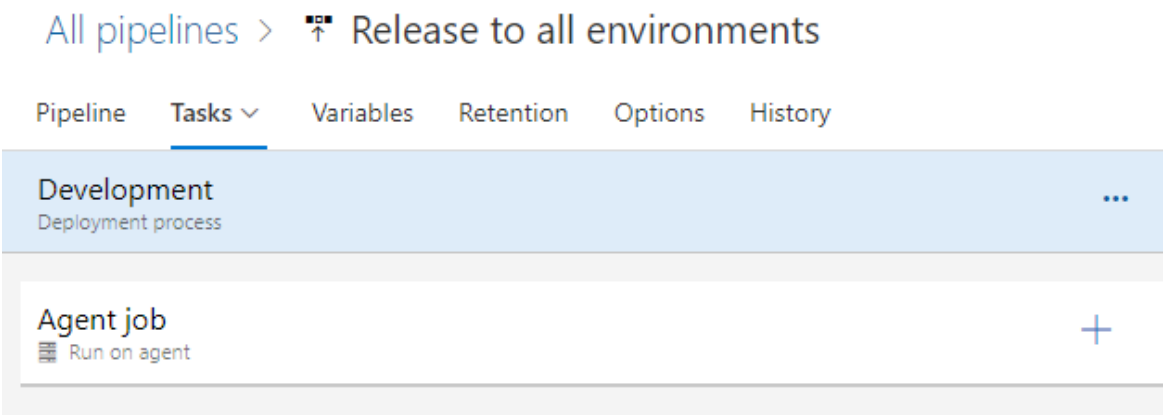
5. Click **OK** to create it. It will then be shown in the list.



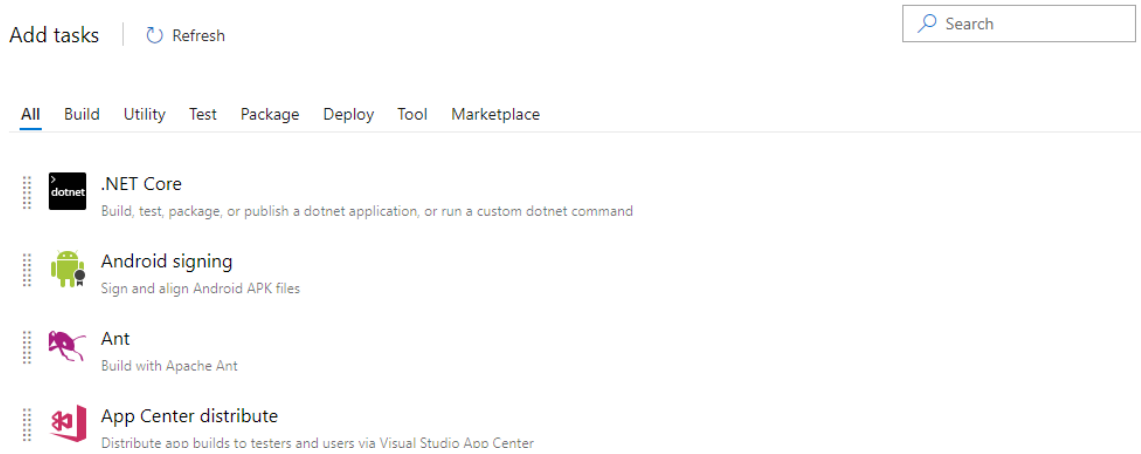
6. Click **Pipelines, Releases** , and **Edit** in the main Parts Unlimited menu to see the release pipeline. Click the link to **View stage tasks** .



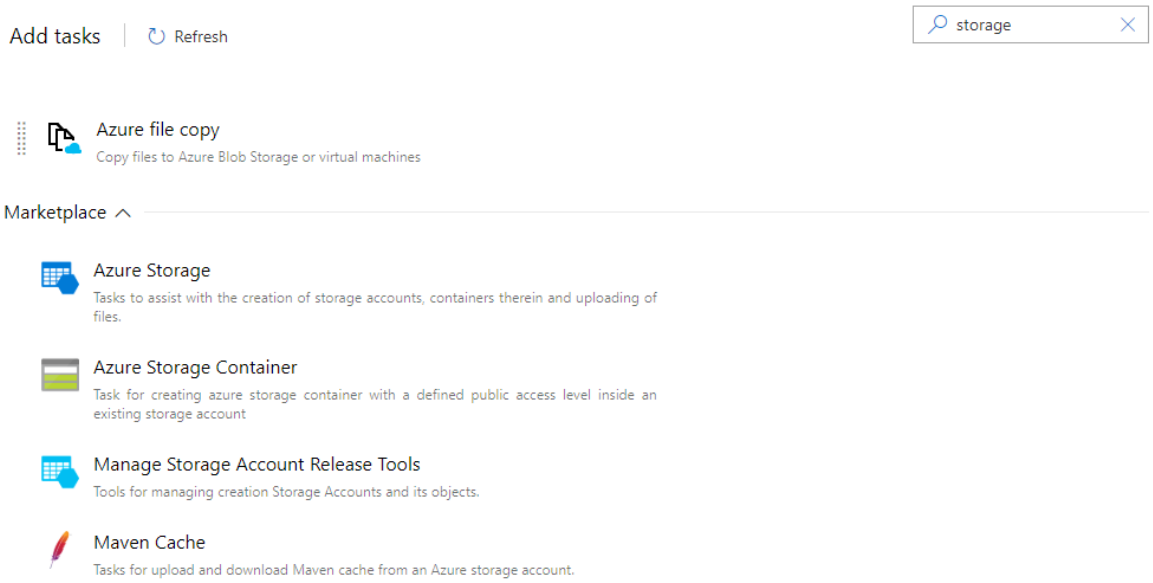
The current list of tasks is then shown. Because we started with an empty template, there are no tasks yet. Each stage can execute many tasks.



7. Click the + sign to the right of the **Agent job** to add a new task. See the available list of task types.

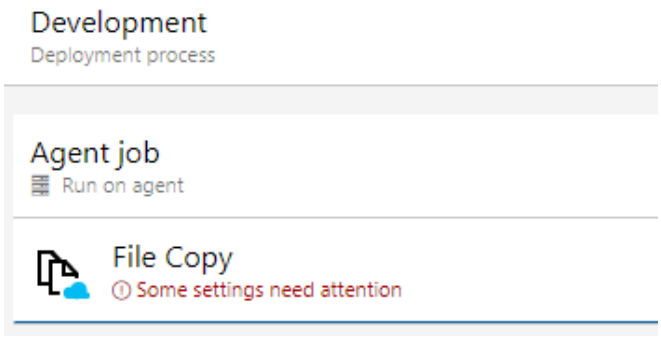


8. In the **Search** box, enter the word **storage** and see the list of storage-related tasks. These include standard tasks and tasks available from the Marketplace.

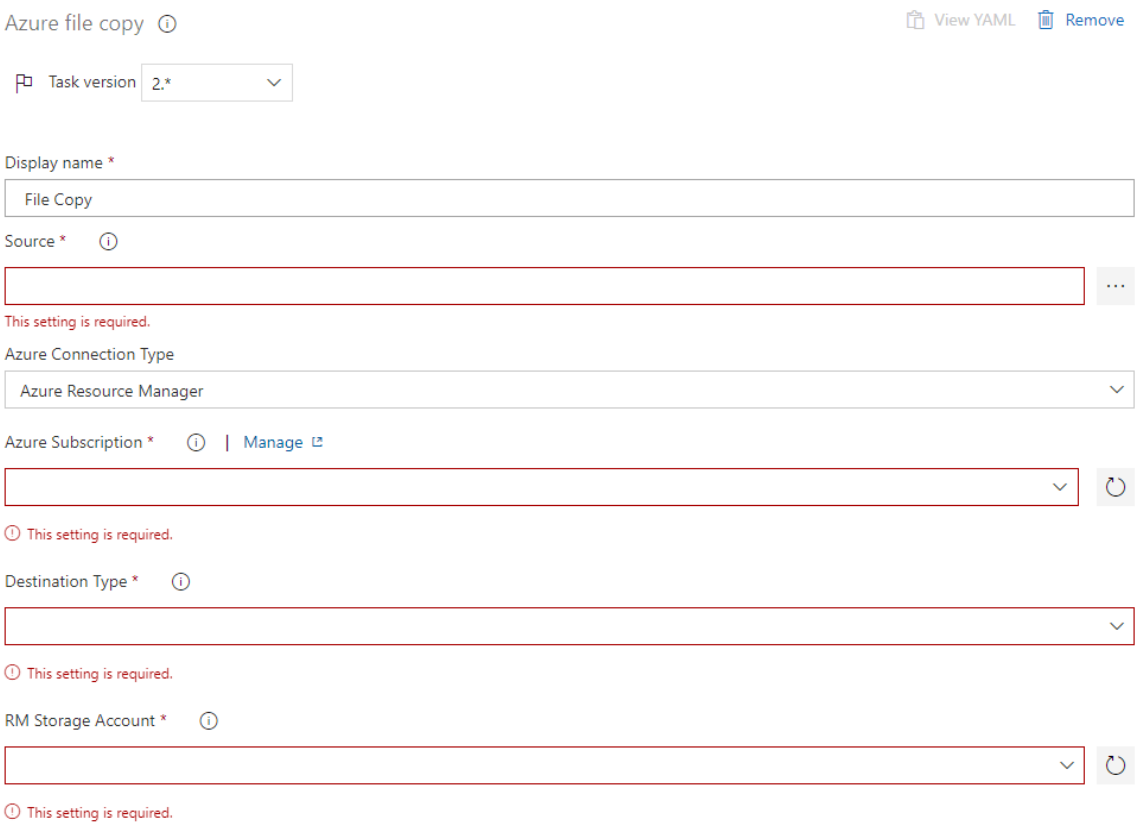


We'll use the Azure file copy task to copy one of our source files to a storage account container.

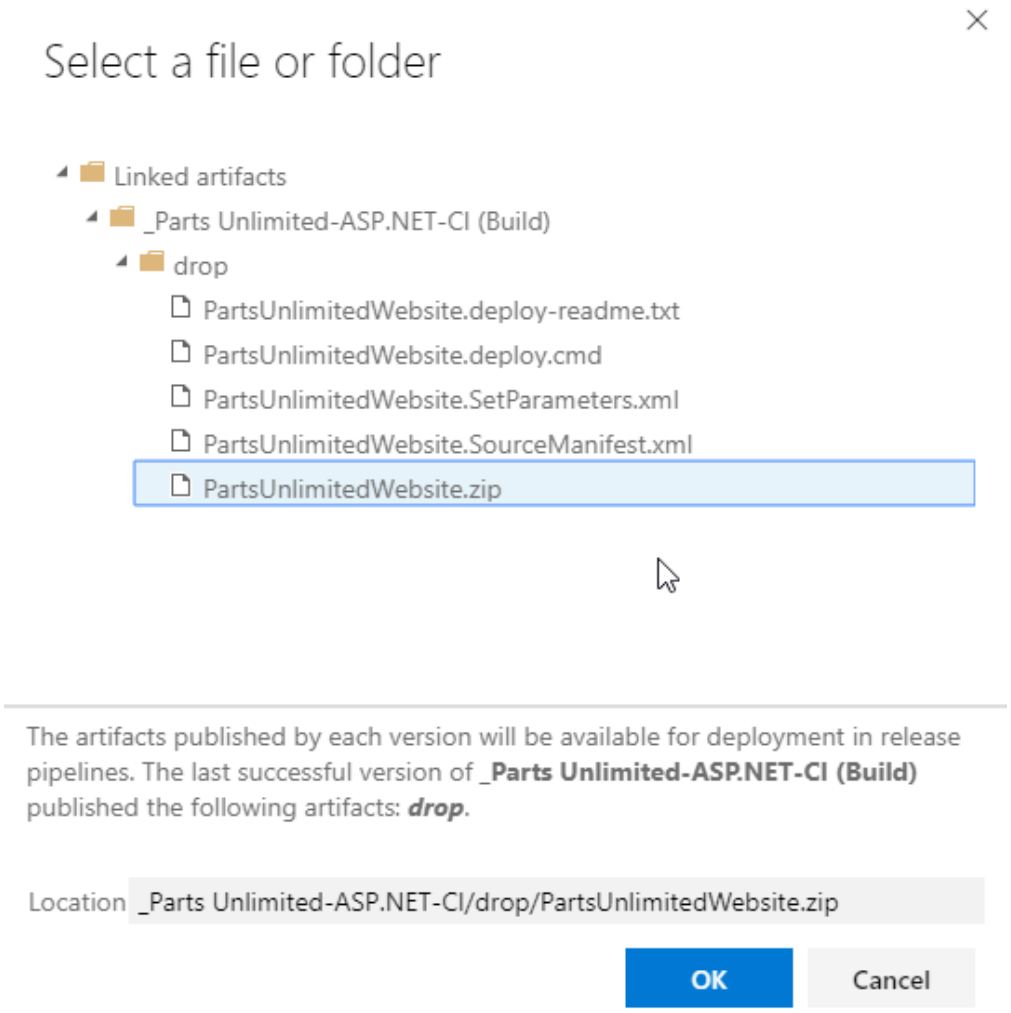
9. Hover over the **Azure file copy** task type and click **Add** when it appears. The task will be added to the stage but requires further configuration.



10. Click the **File Copy** task to see the required settings. Select the latest task version.

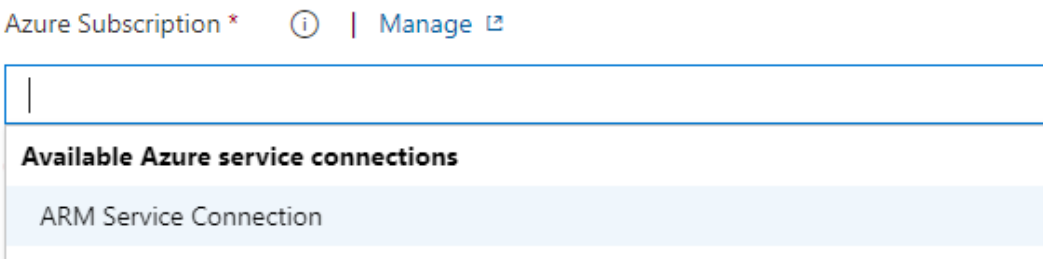


11. Set the **Display Name** to **Backup website zip file** , click the ellipsis beside **Source, locate** the file as follows, and click **OK** to select it.



We then need to provide details on connecting to the Azure subscription. The easiest and most secure way is to use our new Service Connection.

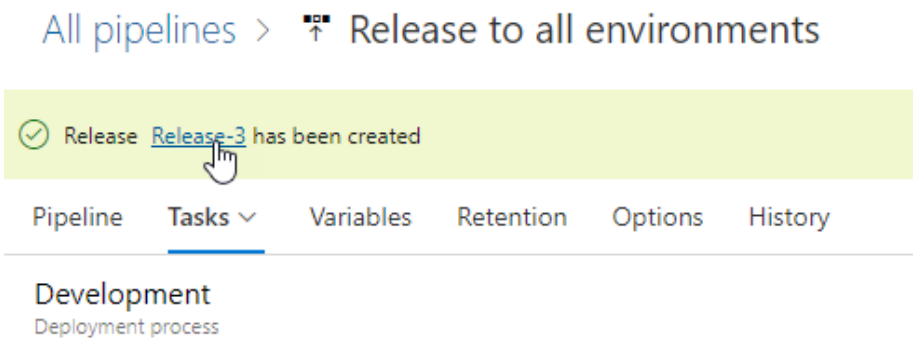
12. Find and select the **Azure Resource Manager Service Connection** we created from the **Azure Subscription** drop-down list.



13. From the **Destination Type** drop-down list, select **Azure Blob** , and from the **RM Storage Account** and **Container Name** , select the storage account, enter the container's name, then click **Save** at the top of the screen, **OK** .

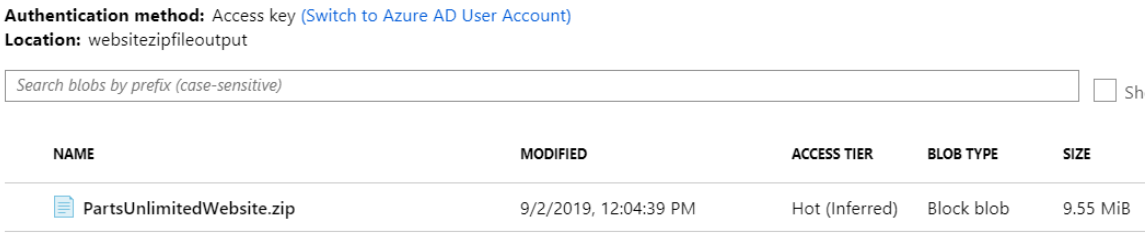
14. To test the task, click **Create release** , and in the **Create a new release** pane, click **Create** .

15. Click the new release to view the details.



16. On the release page, approve the release so that it can continue.

17. Once the **Development** stage has been completed, you should see the file in the Azure storage account.



A key advantage of using service connections is that this type of connection is managed in a single place within the project settings. It doesn't involve connection details spread throughout the pipeline tasks.

[Continue]

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Configure automated integration and functional test automation

Completed

- 6 minutes

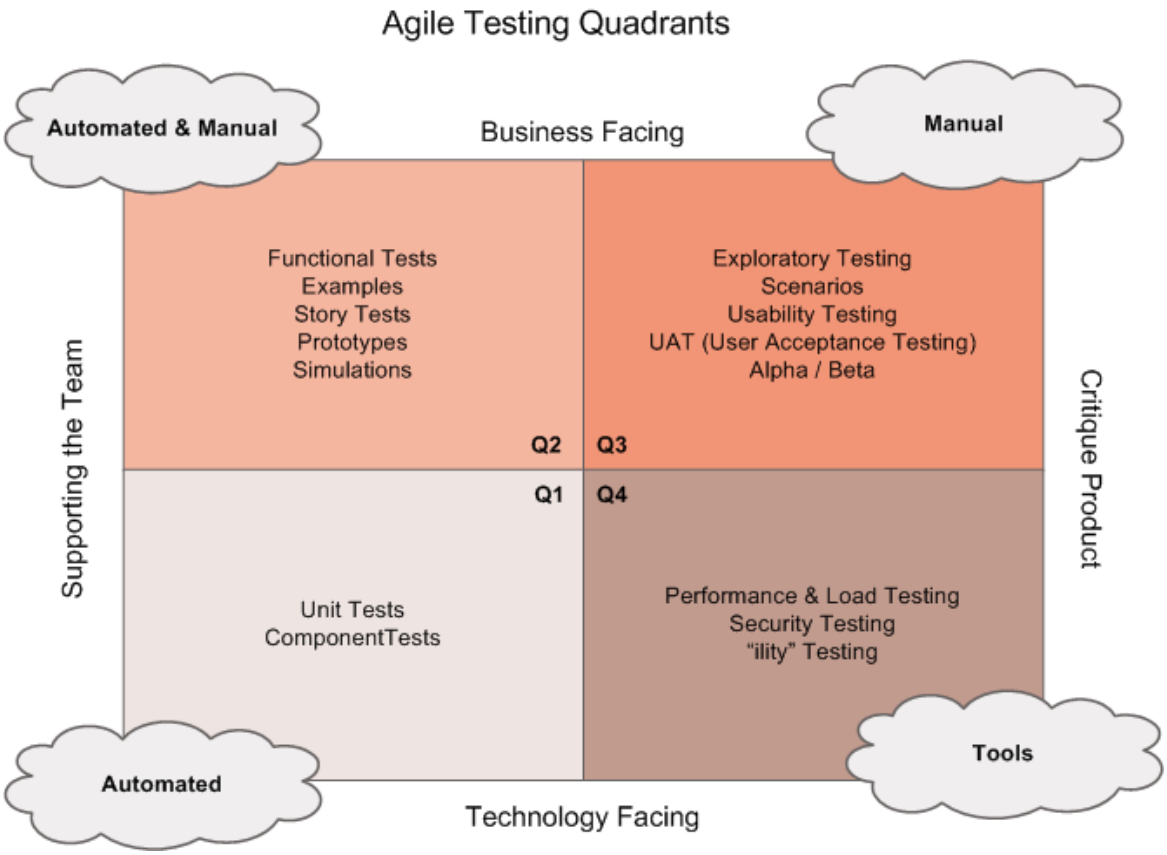
The first thing that comes to mind about Continuous Delivery is that everything needs to be automated.

Otherwise, you can't deploy multiple times a day. But how to deal with testing, then?

Many companies still have a broad suite of manual tests to be run before delivering to production. Somehow these tests need to run every time a new release is created.

Instead of automating all your manual tests into automated UI tests, you need to rethink your testing strategy.

Lisa Crispin describes in her book Agile Testing that you can divide your tests into multiple categories.



Source: https://lisacrispin.com/2011/11/08/using-the-agile-testing-quadrants

We can make four quadrants where each side of the square defines our targets with our tests.

- Business facing - the tests are more functional and often executed by end users of the system or by specialized testers that know the problem domain well.
- Supporting the Team - it helps a development team get constant feedback on the product to find bugs quickly and deliver a quality build-in product.
- Technology facing - the tests are rather technical and non-meaningful to business people. They're typical tests written and executed by the developers in a development team.
- Critique Product - tests that validate a product's workings on its functional and non-functional requirements.

Now we can place different test types we see in the other quadrants. For example, we can put Unit tests, Component tests, and System or integration tests in the first quadrant.

We can place functional tests, Story tests, prototypes, and simulations in quadrant two. These tests support the team in delivering the correct functionality and are business-facing since they're more functional.

In quadrant three, we can place tests like exploratory, usability, acceptance, etc.

We place performance, load, security, and other non-functional requirements tests in quadrant four.

Looking at these quadrants, specific tests are easy to automate or automated by nature. These tests are in quadrants 1 and 4. Tests that are automatable but mostly not automated by nature are the tests in quadrant 2. Tests that are the hardest to automate are in quadrant 3.

We also see that the tests that can't be automated or are hard to automate are tests that can be executed in an earlier phase and not after release.

We call shift-left, where we move the testing process towards the development cycle.

We need to automate as many tests as possible and test them.

A few of the principles we can use are:

- Tests should be written at the lowest level possible.
- Write once, and run anywhere, including the production system.
- The product is designed for testability.
- Test code is product code; only reliable tests survive.
- Test ownership follows product ownership.

By testing at the lowest level possible, you'll find many tests that don't require infrastructure or applications to be deployed.

We can use the pipeline to execute the tests that need an app or infrastructure. We can run scripts or use specific test tools to perform tests within the pipeline.

On many occasions, you execute these external tools from the pipeline, like Owasp ZAP, SpecFlow, or Selenium.

You can use test functionality from a platform like Azure on other occasions. For example, Availability or Load Tests executed from within the cloud platform.

When you want to write your automated tests, choose the language that resembles the language from your code.

In most cases, the application developers should also write the test, so it makes sense to use the same language. For example, write tests for your .NET application in .NET and your Angular application in Angular.

The build and release agent can handle it to execute Unit Tests or other low-level tests that don't need a deployed application or infrastructure.

When you need to do tests with a UI or other specialized functionality, you need a Test agent to run the test and report the results. Installation of the test agent then needs to be done upfront or as part of the execution of your pipeline.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

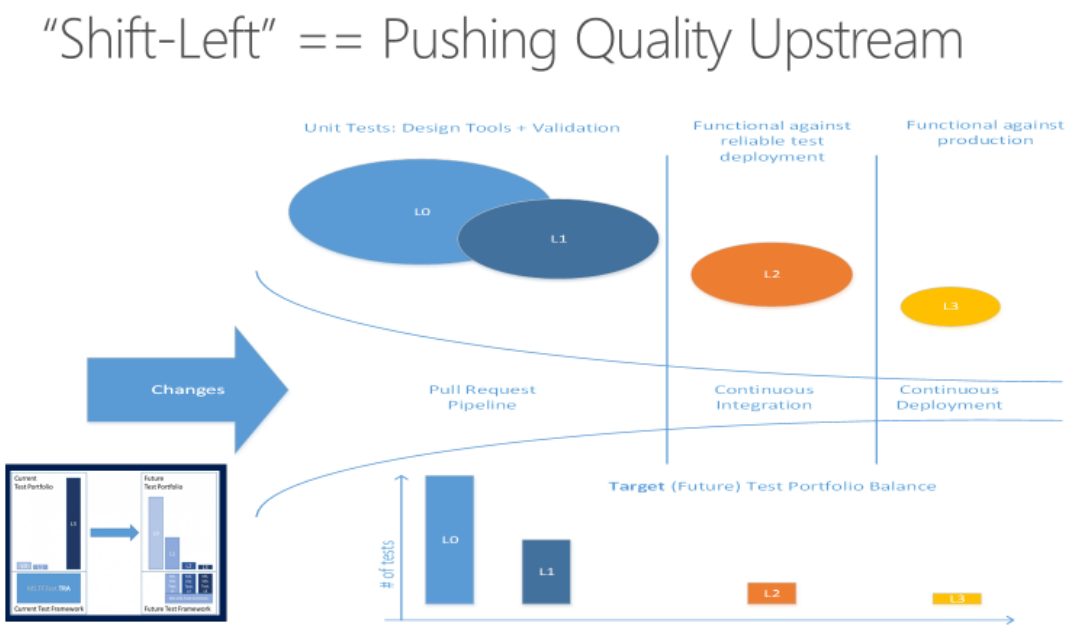
Understand Shift-left

Completed

- 4 minutes

The goal for shifting left is to move quality upstream by performing tests early in the pipeline. It represents the phrase "fail fast, fail often" combining test and process improvements reduces the time it takes for tests to be run and the impact of failures later on.

The idea is to ensure that most of the testing is complete before merging a change into the main branch.

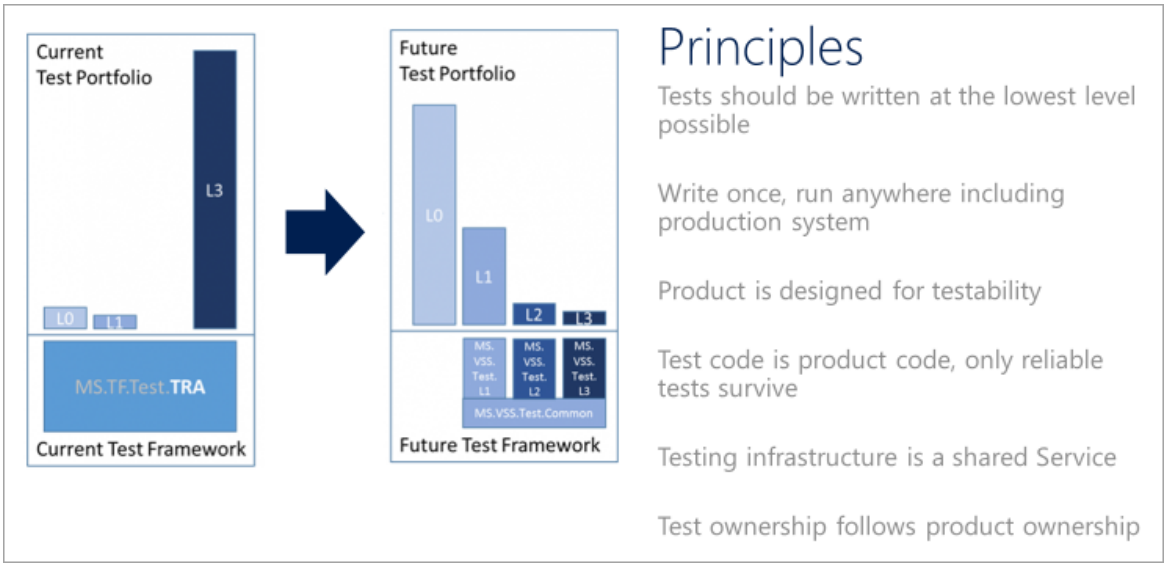


Many teams find their test takes too long to run during the development lifecycle.

As projects scale, the number and nature of tests will grow substantially, taking hours or days to run the complete test.

They get pushed further until they're run at the last possible moment, and the benefits intended to be gained from building those tests aren't realized until long after the code has been committed.

There are several essential principles that DevOps teams should adhere to in implementing any quality vision.



Other important characteristics to take into consideration:

- **Unit tests:** These tests need to be fast and reliable.
  - One team at Microsoft runs over 60,000 unit tests in parallel in less than 6 minutes, intending to get down to less than a minute.
- **Functional tests:** Must be independent.
- **Defining a test taxonomy** is an essential aspect of DevOps. The developers should understand the suitable types of tests in different scenarios.
  - **L0** tests are a broad class of fast in-memory unit tests. It's a test that depends on code in the assembly under test and nothing else.
  - **L1** tests might require assembly plus SQL or the file system.
  - **L2** tests are functional tests run against testable service deployments. It's a functional test category requiring a service deployment but may have critical service dependencies stubbed out.
  - **L3** tests are a restricted class of integration tests that run against production. They require a complete product deployment.

Check the case study in shifting left at Microsoft: Shift left to make testing fast and reliable .

For more information, see:

- Shift right to test in production .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Set up and run availability tests

Completed

- 2 minutes

After you've deployed your web app or website to any server, you can set up tests to monitor its availability and responsiveness.

It's helpful to check if your application is still running and gives a healthy response.

Some applications have specific Health endpoints that an automated process can check. The Health endpoint can be an HTTP status or a complex computation that uses and consumes crucial parts of your application.

For example, you can create a Health endpoint that queries the database. This way, you can check that your application is still accessible, but also the database connection is verified.

You can create your framework to create availability tests (ping test) or use a platform that can do it for you.

Azure has the functionality to develop Availability tests. You can use these tests in the pipeline and as release gates.

In Azure, you can set up availability tests for any HTTP or HTTPS endpoint accessible from the public internet.

You don't have to add anything to the website you're testing. It doesn't even have to be your site: you could try a REST API service you depend on.

There are two types of availability tests:

- URL ping test: a simple test that you can create in the Azure portal. You can check the URL and check the response and status code of the response.
- Multi-step web test: Several HTTP calls that are executed in sequence.

For more information, see also:

- Creating an Application Insights Web Test and Alert Programmatically .
- Monitor the availability of any website .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Explore Azure Load Testing

Completed

- 4 minutes

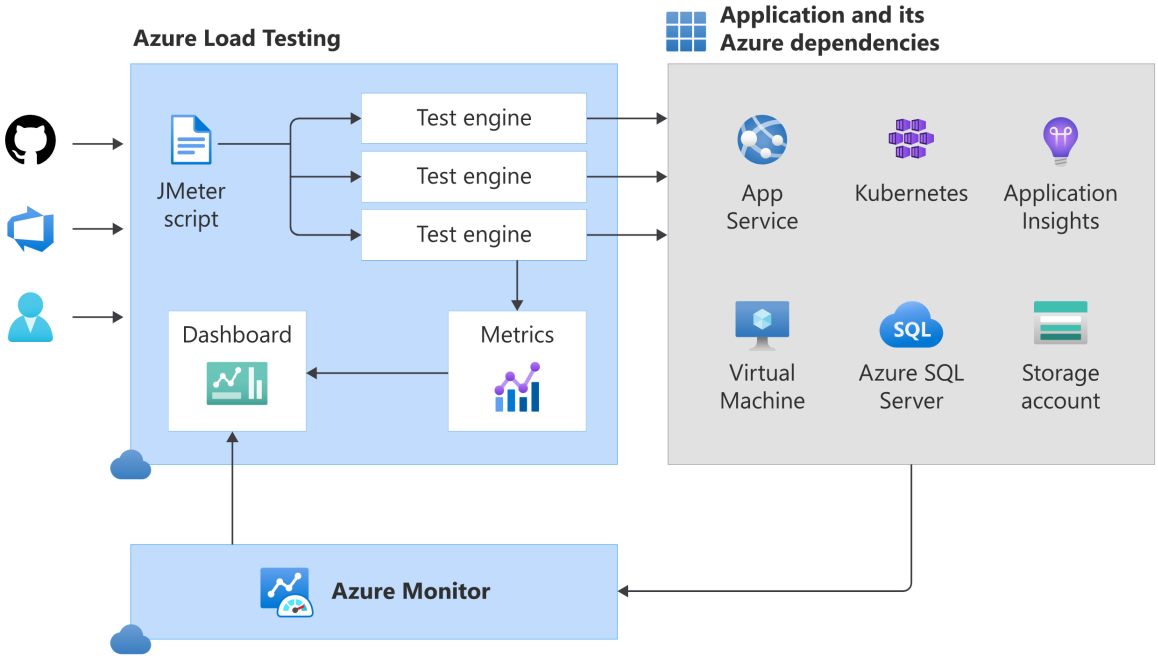
Azure Load Testing Preview is a fully managed load-testing service that enables you to generate a high-scale load.

The service simulates your applications' traffic, helping you optimize application performance, scalability, or capacity.

You can create a load test using existing test scripts based on Apache JMeter. Azure Load Testing abstracts the infrastructure to run your JMeter script and load test your application.

Azure Load Testing collects detailed resource metrics for Azure-based applications to help you identify performance bottlenecks across your Azure application components.

You can automate regression testing by running load tests as part of your continuous integration and continuous deployment (CI/CD) workflow.



Note

The overview image shows how Azure Load Testing uses Azure Monitor to capture metrics for app components. Learn more about the supported Azure resource types .

You can automatically run a load test at the end of each sprint or in a staging environment to validate a release candidate build.

You can trigger Azure Load Testing from Azure Pipelines or GitHub Actions workflows.

Get started with adding load testing to your Azure Pipelines CI/CD workflow , or use our Azure Load Testing GitHub action .

For more information about the Azure Load Testing preview, see:

- What is Azure Load Testing?
- Tutorial: Use a load test to identify performance bottlenecks .
- Tutorial: Set up automated load testing .
- Learn about the key concepts for Azure Load Testing .
- Quickstart: Create and run a load test with Azure Load Testing .
- Tutorial: Identify performance regressions with Azure Load Testing and GitHub Actions - Azure Load Testing .
- Configure Azure Load Testing for high-scale load tests - Azure Load Testing .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Set up and run functional tests

Completed

- 60 minutes

Estimated time: 60 minutes.

Lab files: none.

Scenario

Selenium is a portable open source software-testing framework for web applications. It can operate on almost every operating system. It supports all modern browsers and multiple languages, including .NET (C#) and Java.

This lab will teach you how to execute Selenium test cases on a C# web application as part of the Azure DevOps Release pipeline.

Objectives

After completing this lab, you'll be able to:

- Configure a self-hosted Azure DevOps agent.
- Configure the release pipeline.
- Trigger build and release.
- Run tests in Chrome and Firefox.

Requirements

- This lab requires **Microsoft Edge** or an Azure DevOps-supported browser .
- **Set up an Azure DevOps organization:** If you don't already have an Azure DevOps organization that you can use for this lab, create one by following the instructions available at Create an organization or project collection .
- Identify an existing Azure subscription or create a new one.
- Verify that you have a Microsoft account or a Microsoft Entra account with the Contributor or the Owner role in the Azure subscription. For details, refer to List Azure role assignments using the Azure portal .

Exercises

During this lab, you'll complete the following exercises:

- Exercise 0: Configure the lab prerequisites.
- Exercise 1: Implement Selenium tests by using a self-hosted Azure DevOps agent.
- Exercise 2: Remove the Azure lab resources.

Launch Exercise

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Knowledge check

Completed

- 5 minutes

Choose the best response for each question. Then select **Check your answers** .

Check your knowledge

1.

Which of the following choices isn't a technology or tool for configuring a target server during the deployment?

○

PowerShell Desired State Configuration (DSC).

○

Chef.

○

Azure App Configuration.

2.

Which of the following choices do you need to create when a pipeline needs access to resources?

○

Service Hook.

○

Service Connection.

○

Agent Pool.

3.

Which of the following choices is recommended to authenticate to Azure with Service Connections?

User and Password.

Personal Access Token.

Service Principal.

Check your answers

You must answer all questions before checking your work.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Summary

Completed

- 1 minute

This module detailed target environment provisioning, service connections creation process, and test infrastructure setup. You learned how to configure functional test automation and run availability tests.

You learned how to describe the benefits and usage of:

- Provision and configure target environment.
- Deploy to an environment securely using a service connection.
- Configure functional test automation and run availability tests.
- Set up test infrastructure.

Learn more

- Create target environment - Azure Pipelines | Microsoft Docs .

- Integrate DevTest Labs environments into Azure Pipelines - Azure DevTest Labs | Microsoft Docs .
- Connect to Microsoft Azure - Azure Pipelines | Microsoft Docs .
- Service connections in Azure Pipelines - Azure Pipelines | Microsoft Docs .
- Run Functional Tests task - Azure Pipelines | Microsoft Docs .
- Monitor availability with URL ping tests - Azure Monitor | Microsoft Docs .
- Testing your app and Azure environment - Azure Architecture Center | Microsoft Docs .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Manage and modularize tasks and templates

Introduction

Completed

- 1 minute

This module describes the creation of task and variable groups, creating custom build and release tasks, and using release variables and stage variables in your pipeline.

Learning objectives

After completing this module, students and professionals can:

- Use and manage task and variable groups.
- Use release variables and stage variables in your release pipeline.
- Use variables in release pipelines.
- Create custom build and release tasks.

Prerequisites

- Understanding of what DevOps is and its concepts.
- Familiarity with version control principles is helpful but isn't necessary.
- Beneficial to have experience in an organization that delivers software.
- For some exercises, you need to create an Azure DevOps Organization and a Team Project. If you don't have it yet, see: Create an organization - Azure DevOps .
    - If you already have your organization created, use the Azure DevOps Demo Generator and create a new Team Project called "Parts Unlimited" using the template "PartsUnlimited." Or feel free to create a blank project. See Create a project - Azure DevOps .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Examine task groups

Completed

- 1 minute

A task group allows you to encapsulate a sequence of tasks, already defined in a build or a release pipeline, into a single reusable task that can be added to a build or release pipeline,

just like any other task.

You can choose to extract the parameters from the encapsulated tasks as configuration variables and abstract the rest of the task information.

Task groups are a way to standardize and centrally manage deployment steps for all your applications.

When you include a task group in your definitions and then make a change centrally to the task group, the change is automatically reflected in all the definitions that use the task group.

There's no need to change each one individually.

For more information, see Task groups for builds and releases .

Note

Note: Task Groups aren't currently supported in YAML. Use templates instead. See Template References .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Exercise - create and manage task groups

Completed

- 4 minutes

In this exercise, you'll investigate Task Groups.

Note

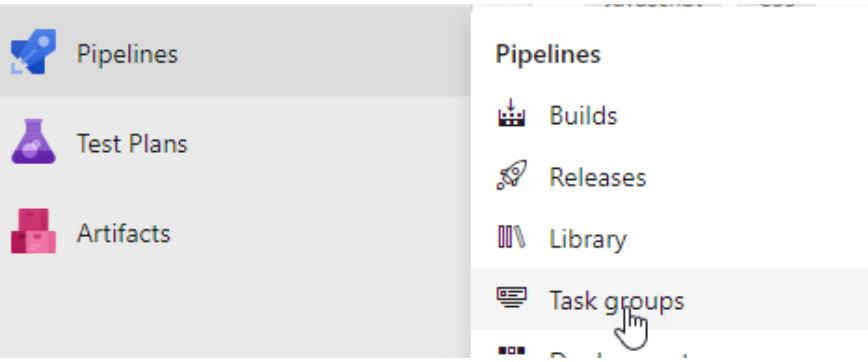
Before starting this walkthrough, ensure you've done the steps in the prerequisites section and the previous exercises.

Steps

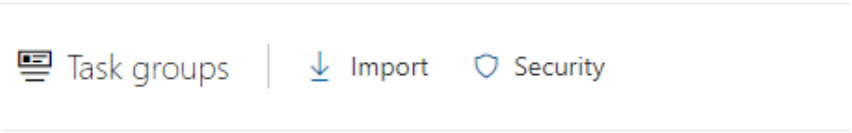
Let's now look at how a release pipeline can reuse groups of tasks.

It's common to reuse a group of tasks in more than one stage within a pipeline or in different pipelines.

1. In the main menu for the **Parts Unlimited** project, click **Pipelines,** then click **Task groups** .

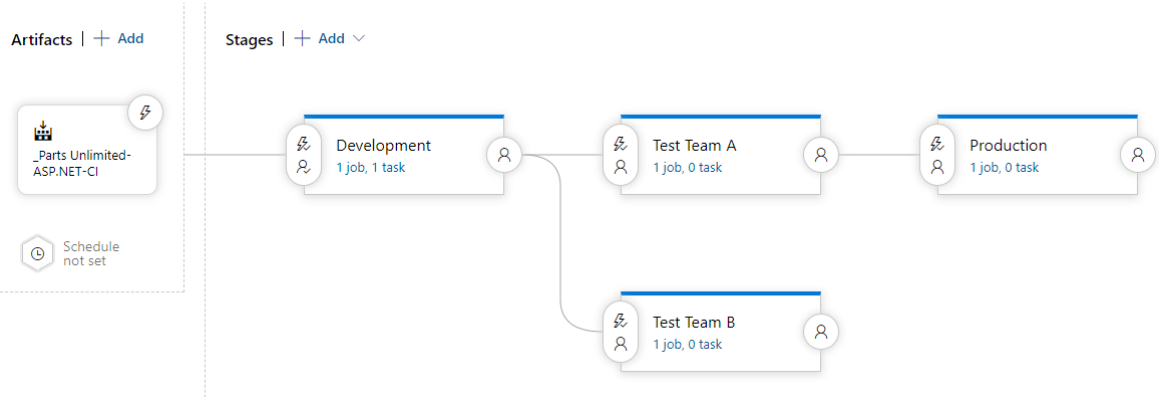


You'll notice that you don't currently have any task groups defined.

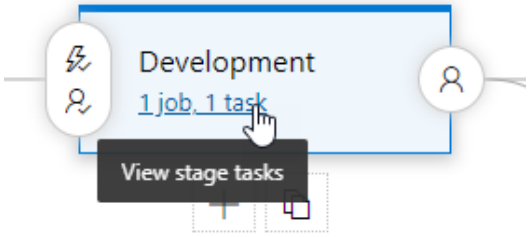


There's an option to import task groups, but the most common way to create a task group is directly within the release pipeline, so let's do that.

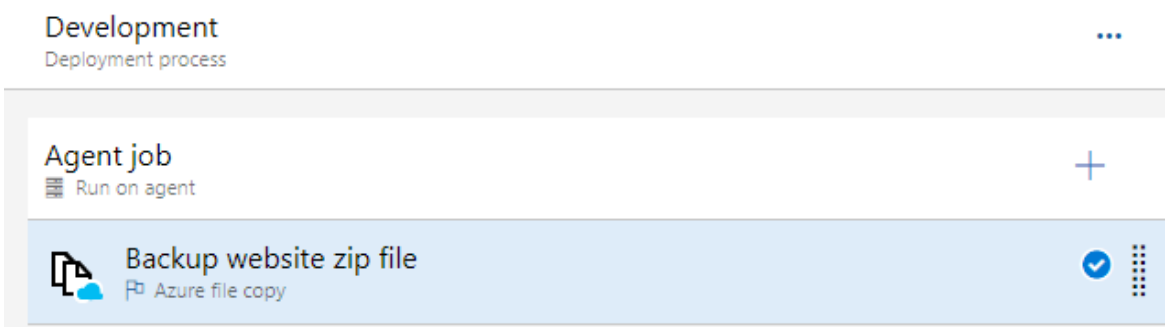
2. Click **Pipelines,** click **Releases** and click **Edit** to open the pipeline we worked on in the main menu.



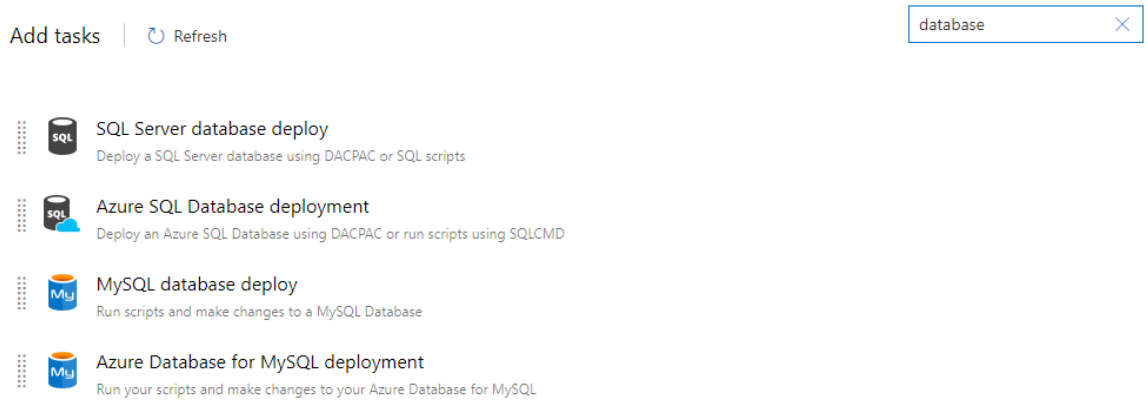
3. The **Development** stage currently has a single task. We'll add another task to that stage. Click the **View stage tasks** link to open the stage editor.



You can see that there's currently one task.

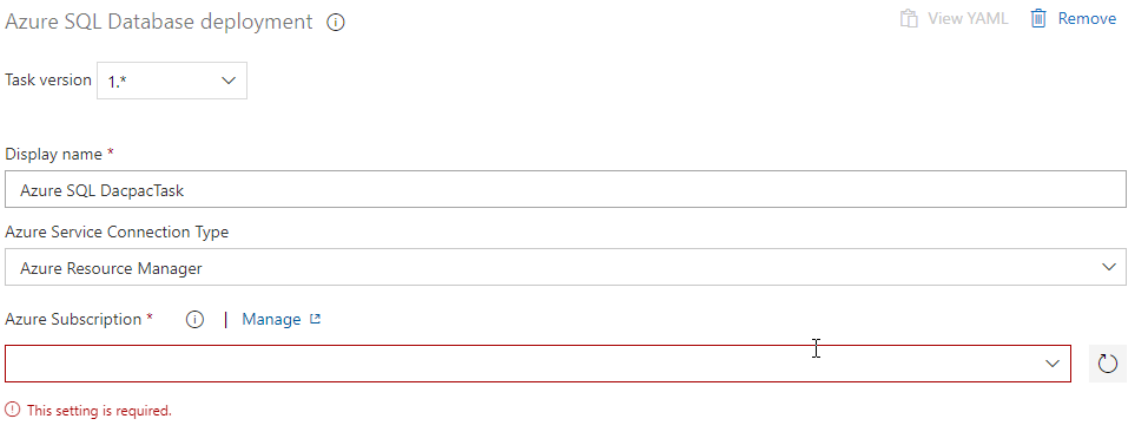


4. Click the + sign to the right of the **Agent job** line to add a new task, in the **Search** box, type **database** .



We'll add a task to deploy an Azure SQL Database.

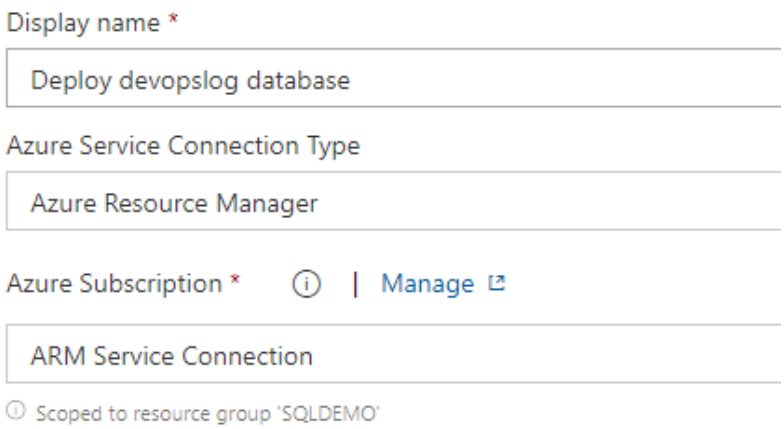
5. Hover over the **Azure SQL Database Deployment** option and click **Add** . Click the **Azure SQL DacpacTask** when it appears in the list to open the settings pane.



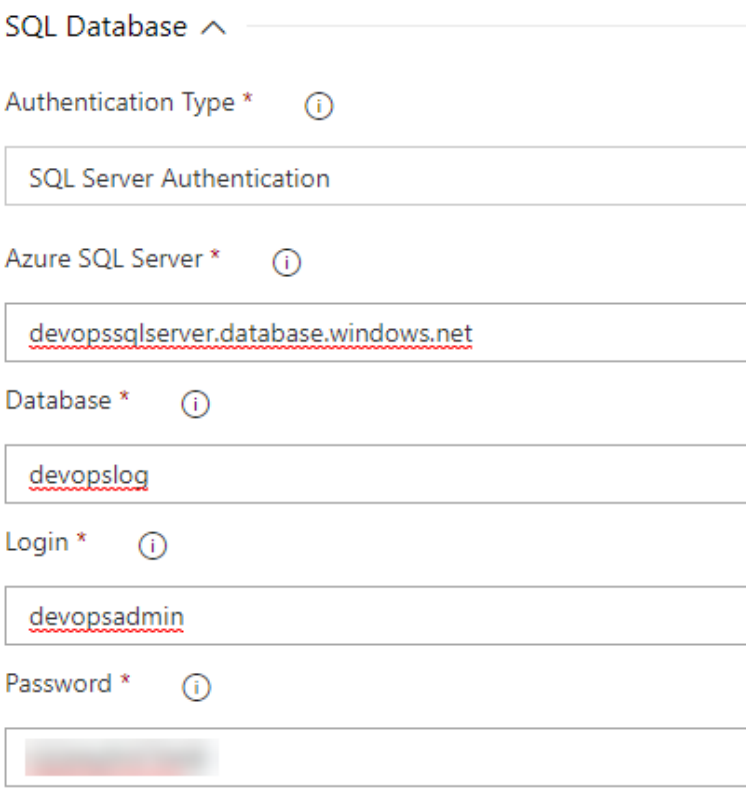
6. Set the **Display name** to **Deploy devopslog database** , and from the **Azure Subscriptions** drop-down list, click **ARM Service Connection** .

   Note

   We can reuse our service connection here.

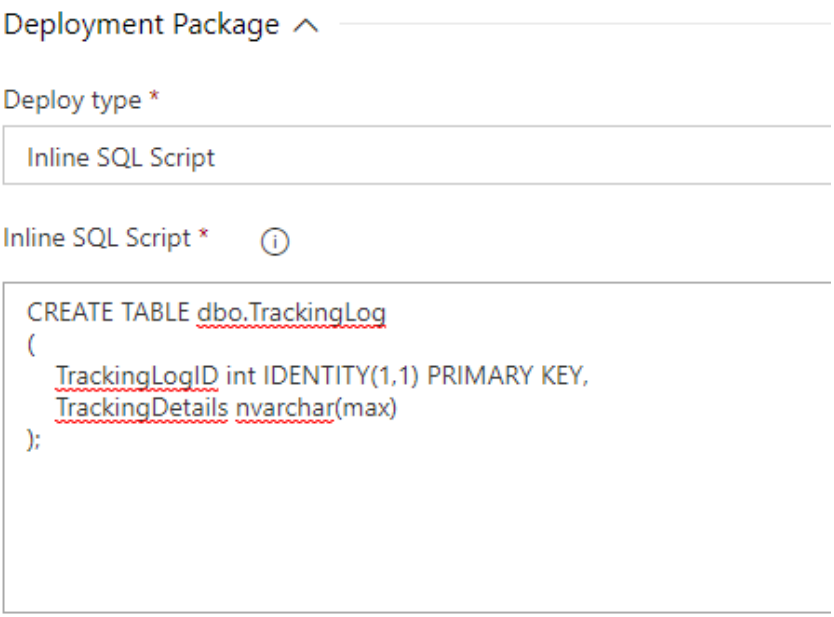


7. In the **SQL Database** section, set a unique name for the SQL Server, set the **Database** to **devopslog** , set the **Login** to **devopsadmin** , and set any suitable password.



8. In the **Deployment Package** section, set the **Deploy type** to **Inline SQL Script** , set the **Inline SQL Script** to:

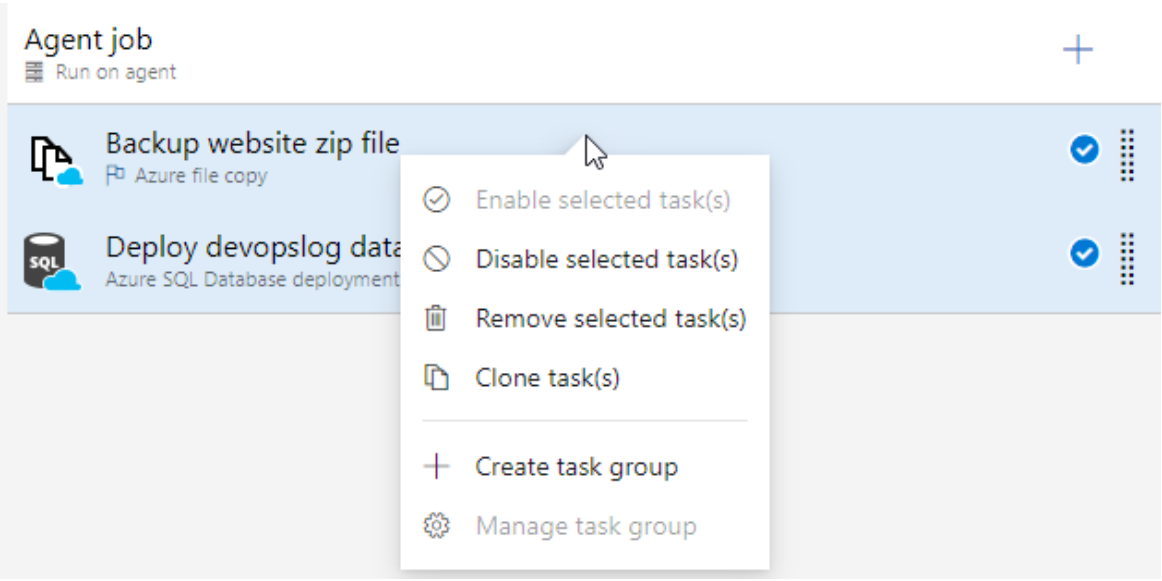
```
CREATE TABLE dbo.TrackingLog
(
    TrackingLogID int IDENTITY(1,1) PRIMARY KEY,
    TrackingDetails nvarchar(max)
);
```



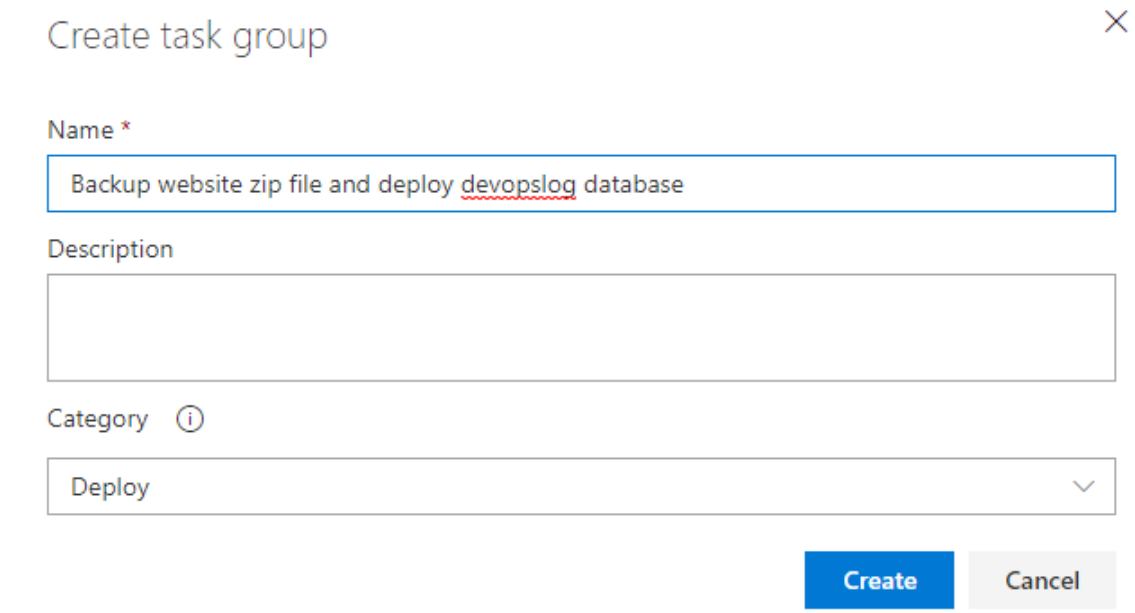
9. Click **Save** then **OK** to save the work.

   Now that we have two tasks let's use them to create a task group.

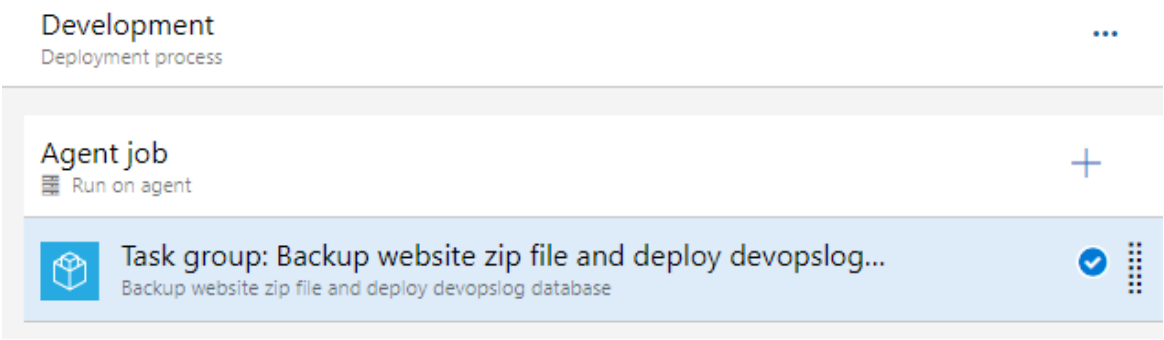
10. Click to select the **Backup website zip file** task and select the **Deploy devopslog database** task, then right-click either task.



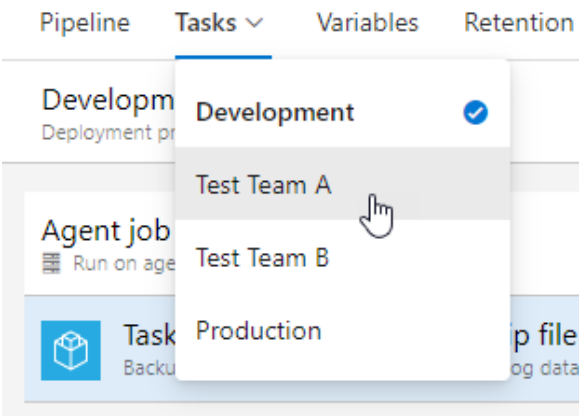
11. Click **Create task group** , then in the **Create task group** window, set **Name** to **Backup website zip file and deploy devopslog** . Click the **Category** drop-down list to see the available options. Ensure that **Deploy** is selected, and click **Create** .



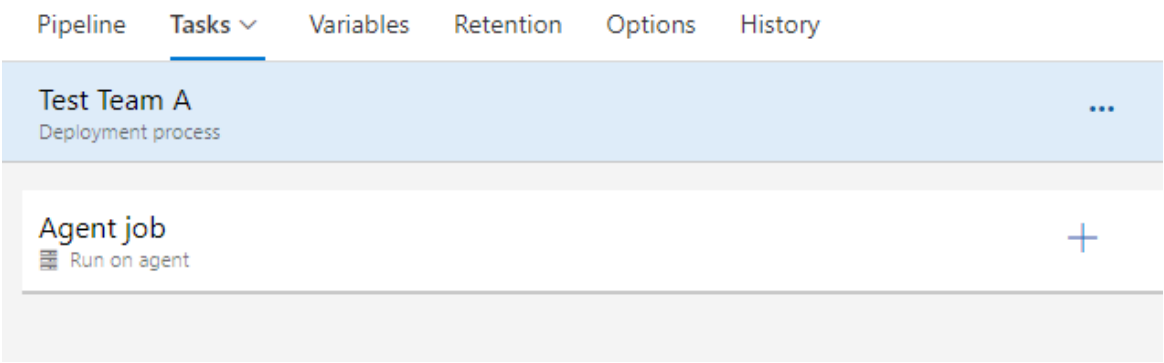
The individual tasks have now disappeared from the list of tasks, and the new task group appears instead.



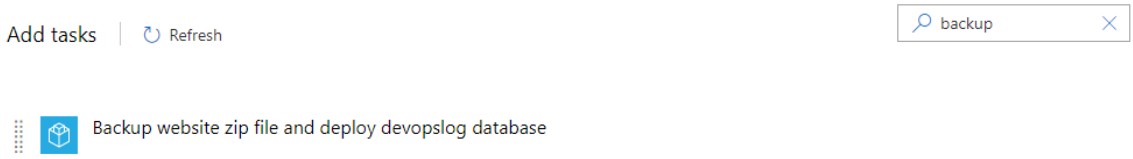
12. From the **Task** drop-down list, select the **Test Team A** stage.



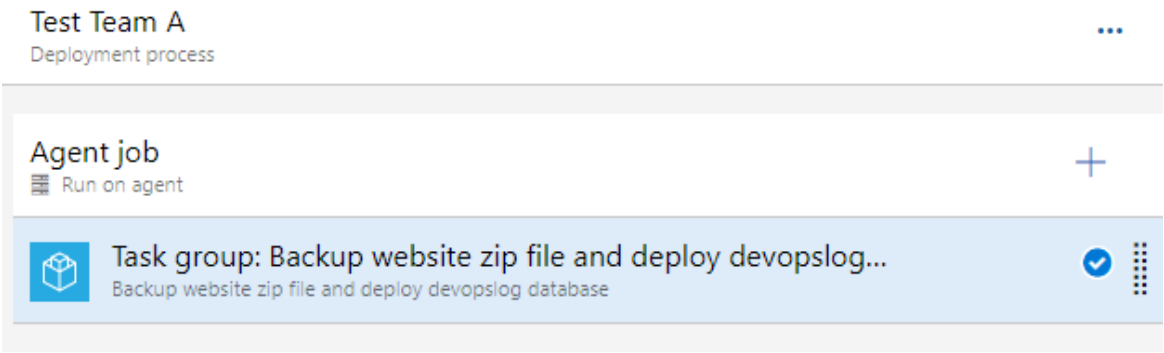
There are currently no tasks on the stage.



13. Click the + sign to the right of the **Agent job** to add a new task. In the **Search** box, type **backup** and notice that the new task group appears like any other task.



14. Hover on the task group and click **Add** when it appears.



Task groups allow for each reuse of a set of tasks and limit the number of places where edits need to occur.

Walkthrough cleanup

1. Click **Remove** to remove the task group from the **Test Team A** stage.
2. From the **Tasks** drop-down list, select the **Development** stage. Again click **Remove** to remove the task group from the **Development** stage.
3. Click **Save,** then **OK** .

[Continue]

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Explore variables in release pipelines

Completed

- 2 minutes

Variables give you a convenient way to get critical bits of data into various parts of the pipeline.

As the name suggests, the contents of a variable may change between releases, stages of jobs of your pipeline.

The system predefines some variables, and you're free to add your own as well.

The variable's scope is the most important thing you need to think about when using variables in the release pipeline.

You can imagine that a variable containing the target server's name may vary between a Development environment and a Test Environment.

Within the release pipeline, you can use variables in different scopes and different ways.

For more information, see Release variables and debugging .

Predefined variables

When running your release pipeline, you always need variables that come from the agent or context of the release pipeline.

For example, the agent directory where the sources are downloaded, the build number or build ID, the agent's name, or any other information.

This information is accessible in predefined variables that you can use in your tasks.

Release pipeline variables

Choose a release pipeline variable when you need to use the same value across all the stages and tasks in the release pipeline, and you want to change the value in a single place.

Stage variables

Share values across all the tasks within one specific stage by using stage variables.

Use a stage-level variable for values that vary from stage to stage (and are the same for all the tasks in a stage).

Variable groups

Share values across all the definitions in a project by using variable groups. We'll cover variable groups later in this module.

Normal and secret variables

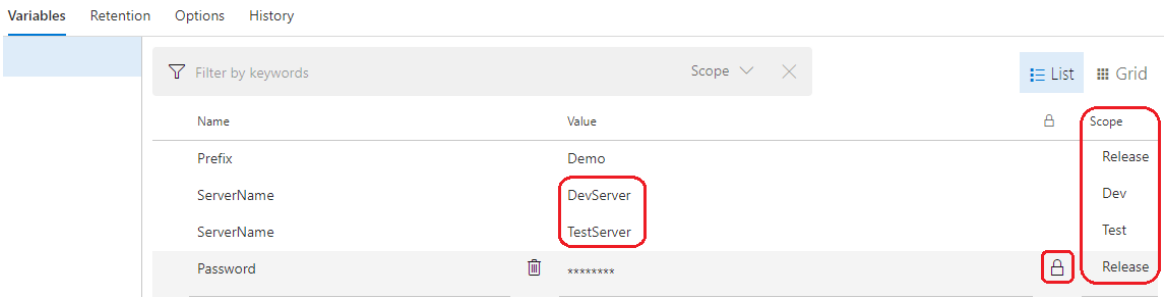
Because the pipeline tasks are executed on an agent, variable values are passed to the various tasks using environment variables.

The task knows how to read it. You should be aware that a variable contains clear text and can be exposed to the target system.

When you use the variable in the log output, you can also see the variable's value.

When the pipeline has finished, the values will be cleared.

You can mark a variable in the release pipeline as secret. This way, the secret is hidden from the log output. It's beneficial when writing a password or other sensitive information.



Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Understand variable groups

Completed

- 1 minute

A variable group stores values that you want to make available across multiple builds and release pipelines.

Examples

- Store the username and password for a shared server.
- Store a share connection string.
- Store the geolocation of an application.

- Store all settings for a specific application.

For more information, see Variable Groups for Azure Pipelines .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Exercise - create and manage variable groups

Completed

- 3 minutes

In this exercise, you'll investigate Variable Groups.

Note

Before starting this walkthrough, ensure you've done the steps in the prerequisites section and the previous activities.

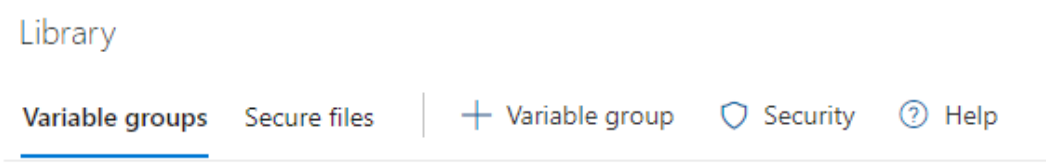
Steps

Let's now look at how a release pipeline can use predefined variables, called Variable Groups.

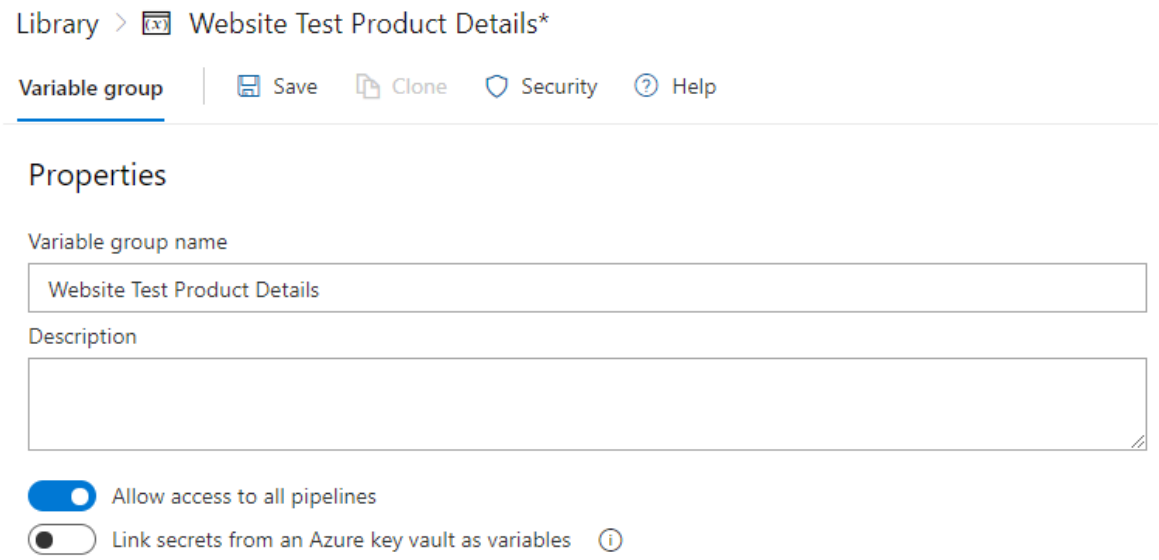
Like how we used task groups, variable groups provide a convenient way to avoid redefining many variables when defining stages within pipelines and even when working across multiple pipelines.

Let's create a variable group and see how it can be used.

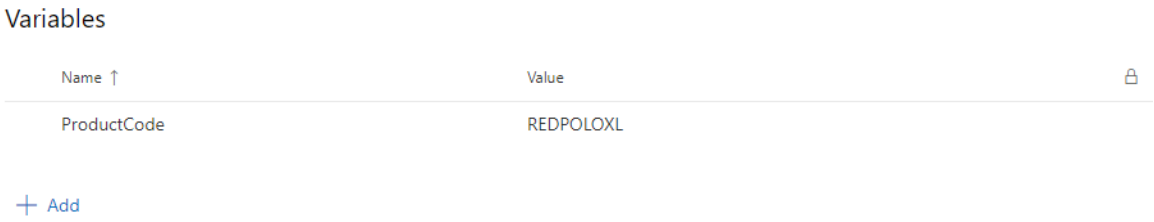
1. On the main menu for the **Parts Unlimited** project, click **Pipelines** , then click **Library** . There are currently no variable groups in the project.



2. Click **+ Variable group** to start creating a variable group. Set **Variable group name** to **Website Test Product Details** .



3. In the **Variables** section, click **+Add** , enter **Name** , enter **ProductCode** , and in **Value** , enter **REDPOLOXL** .



You can see an extra column that shows a lock. It allows you to have variable values that are locked and not displayed in the configuration screens.

While it's often used for values like passwords, notice an option to link secrets from an Azure key vault as variables.

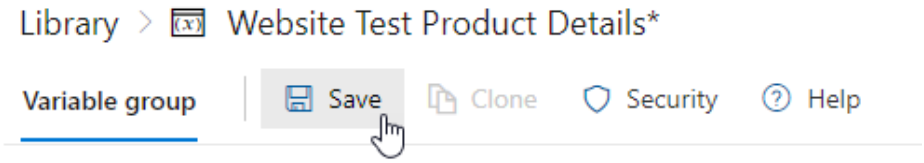
It would be a preferable option for variables that provide credentials that need to be secured outside the project.

In this example, we're just providing details of a product used in testing the website.

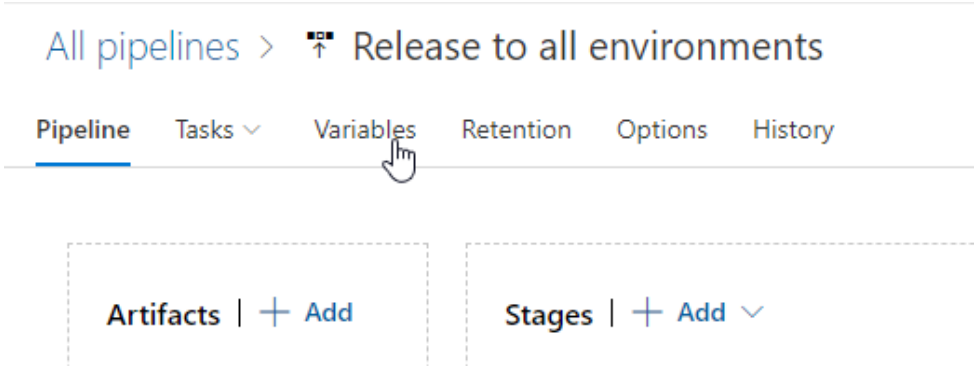
4. Add another variable called **Quantity** with a value of **12** .

5. Add another variable called **SalesUnit** with a value of **Each** .

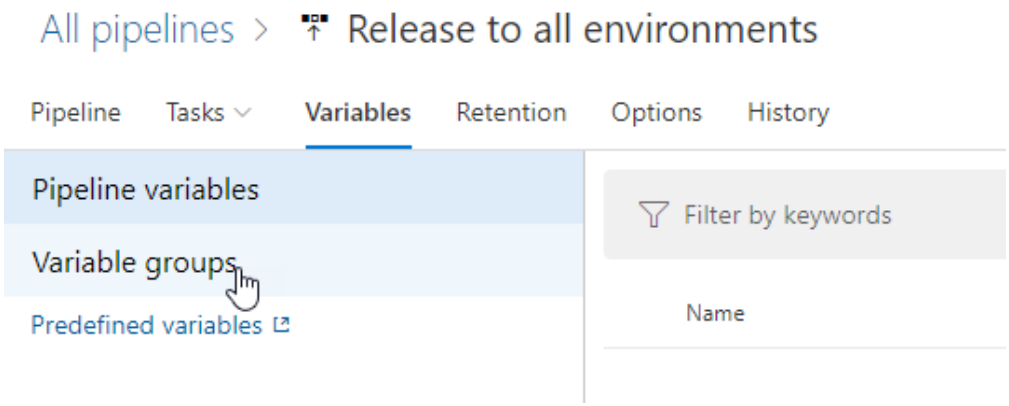
6. Click **Save** to save the new variable group.



7. On the main menu, click **Pipelines** , click Releases and click **Edit** to return to editing the release pipeline we have been working on. From the top menu, click **Variables** .



8. In the left-hand pane, click **Variable Groups** .



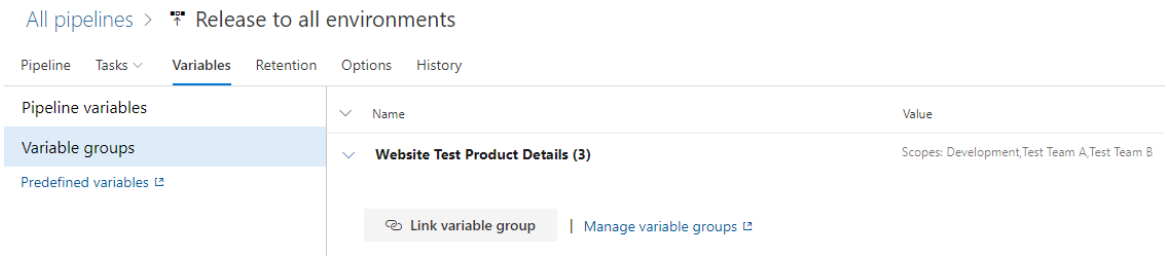
Variable groups are linked to pipelines rather than being directly added to them.

9. Click **Link variable group** , then in the **Link variable group** pane, click the **Website Test Product Details** variable group (notice that it shows you how many variables are contained). In the **Variable group scope** , select the **Development** , **Test Team A** , and **Test Team B** stages.

We need the test product for development and testing, but we don't need it in production. If required in all stages, we would have chosen **Release** for the Variable

group scope instead.

10. Click the **Link** to complete the link.



The variables contained in the variable group are now available for use within all stages except production, just the same way as any other variable.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Knowledge check

Completed

- 4 minutes

Choose the best response for each question. Then select **Check your answers** .

Check your knowledge

1.

Which of the following choices is correct about Azure Pipelines?

○

Azure Pipelines allow creating custom build/release tasks.

○

Azure Pipelines only allow native tasks or extensions from the marketplace.

○

Azure Pipelines automatically create tasks based on PowerShell.

2.

Which of the following choices should you create to store values that you want to make available across multiple builds and release pipelines?

Task Group.

Variable Group.

Deployment Group.

3.

Which of the following choices isn't an advantage of creating your own task?

You can safely and efficiently distribute across your organization.

You can use and reuse a secure endpoint to a target server.

Users can see the implementation details.

Check your answers

You must answer all questions before checking your work.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Summary

Completed

- 1 minute

This module described the creation of task and variable groups, creating custom build and release tasks, and using release variables and stage variables in your pipeline.

You learned how to describe the benefits and usage of:

- Use and manage task and variable groups.
- Use release variables and stage variables in your release pipeline.
- Use variables in release pipelines.
- Create custom build and release tasks.

Learn more

- Variable groups for Azure Pipelines - Azure Pipelines | Microsoft Docs .
- Define variables - Azure Pipelines | Microsoft Docs .
- Add a build or release task in an extension - Azure DevOps | Microsoft Docs .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Automate inspection of health

Introduction

Completed

- 1 minute

This module describes how to automate the inspection of health events, configure notifications, and set up service hooks to monitor pipelines.

Learning objectives

After completing this module, students and professionals can:

- Implement automated inspection of health.
- Create and configure events.
- Configure notifications.
- Create service hooks to monitor the pipeline.

Prerequisites

- Understanding of what DevOps is and its concepts.
- Familiarity with version control principles is helpful but isn't necessary.
- Beneficial to have experience in an organization that delivers software.
- You need to create an Azure DevOps Organization and a Team Project for some exercises. If you don't have it yet, see: Create an organization - Azure DevOps .
    - If you already have your organization created, use the Azure DevOps Demo Generator and create a new Team Project called "Parts Unlimited" using the template "PartsUnlimited." Or feel free to create a blank project. See Create a project - Azure DevOps .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Automate inspection of health

Completed

- 2 minutes

Inspection of the release pipeline and release is something you should consider from the start.

When you run multiple deployments a day, you want to:

- Stay informed.
- Know whether a release passed or failed.
- Know the quality of the release.
- Know details about the release and how it has been done.
- Stop releases when you detect something suspicious.
- Visualize some of these things on a dashboard.

You can do a few things to stay informed about your release pipeline. In the following chapters, we'll dive a bit deeper into these.

Release gates

Release gates allow the automatic collection of health signals from external services and then promote the release when all the signs are booming simultaneously or stop the deployment on timeout.

Typically, gates are connected with incident management, problem management, change management, monitoring, and external approval systems. Release gates are discussed in an upcoming module.

Events, subscriptions, and notifications

Events are raised when specific actions occur, like when a release is started or a build is completed.

A notification subscription is associated with a supported event type. The subscription ensures you get notified when a specific event occurs.

Notifications are usually emails that you receive when an event occurs to which you're subscribed.

Service hooks

Service hooks enable you to do tasks on other services when events happen in your Azure DevOps Services projects.

For example, create a card in Trello when a work item is created or send a push notification to your team's Slack when a build fails.

Service hooks can also be used in custom apps and services as a more efficient way to drive activities when events happen in your projects.

Reporting

Reporting is the most static approach to inspection but also the most evident in many cases.

Creating a dashboard that shows the status of your build and releases combined with team-specific information is, in many cases, a valuable asset to get insights.

Read more at About dashboards, charts, reports, & widgets .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Explore events and notifications

Completed

- 2 minutes

One of the first requests many people have when working in a system that does asynchronous actions is to get notifications or alerts. Why?

Because they don't want to open the application, log in and see if things changed repeatedly.

The ability to receive Alerts and notifications is a powerful mechanism to get notified about certain events in your system when they happen.

For example, when a build takes a while to complete, you probably don't want to stare at the screen until it has finished. But you want to know when it does.

Getting an email or another kind of notification instead is powerful and convenient. Another example is a system that needs to be monitored.

You want to get notified by the system in real time. By implementing a successful alert mechanism, you can use alerts to react to situations proactively before anybody is bothered by them.

Alerts

However, when you define alerts, you need to be careful. When you get alerts for every single event that happens in the system, your mailbox will quickly be flooded with numerous alerts.

The more alerts you get that aren't relevant, the more significant the change that people will never look at the alerts and notifications and will miss the important ones.

Target audience and delivery mechanism

When defining alerts or notifications, you need to think about the target audience. Who needs to react to the alerts? Don't send messages to people for information only. They'll

stop looking at it quickly.

Another thing to consider when defining alerts is the mechanism to deliver them. Do you want to send an email, or do you like to send a message in Slack for your team? Or do you want to call another system to do a particular action?

Within Azure DevOps, there are multiple ways to define your alerts. By using query and filter mechanisms, you can filter out specific alerts. For example, you only want to get notified for failed releases and not for successful ones.

Almost every action in the system raises an event to which you can subscribe. A subscription is personal or for your whole team. When you have made a subscription, you can select how you want the notification to be delivered.

For more information, see also:

- About notifications .
- Events, subscriptions, and notifications .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Explore service hooks

Completed

- 1 minute

Service hooks enable you to do tasks on other services when your Azure DevOps Services projects happen.

For example, create a card in Trello when a work item is created.

Or send a push notification to your team's mobile devices when a build fails.

Service hooks can also be used in custom apps and services.

It's a more efficient way to drive activities when events happen in your projects.

Azure DevOps includes built-in support for the following Service Hooks:

Build and release.	Collaborate	Customer support	Plan and track	Integrate
AppVeyor	Campfire	UserVoice	Trello	Azure Service Bus
Bamboo	Flowdock	Zendesk		Azure Storage
Jenkins	HipChat			Web Hooks
MyGet	Hubot			Zapier

Build and release. Collaborate Customer support Plan and track Integrate

Slack

This list will change over time.

To learn more about service hooks and how to use and create them, read Service Hooks in Azure DevOps .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Exercise - set up service hooks to monitor the pipeline

Completed

- 4 minutes

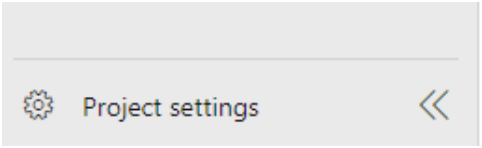
In this exercise, you'll investigate Service Hooks.

Steps

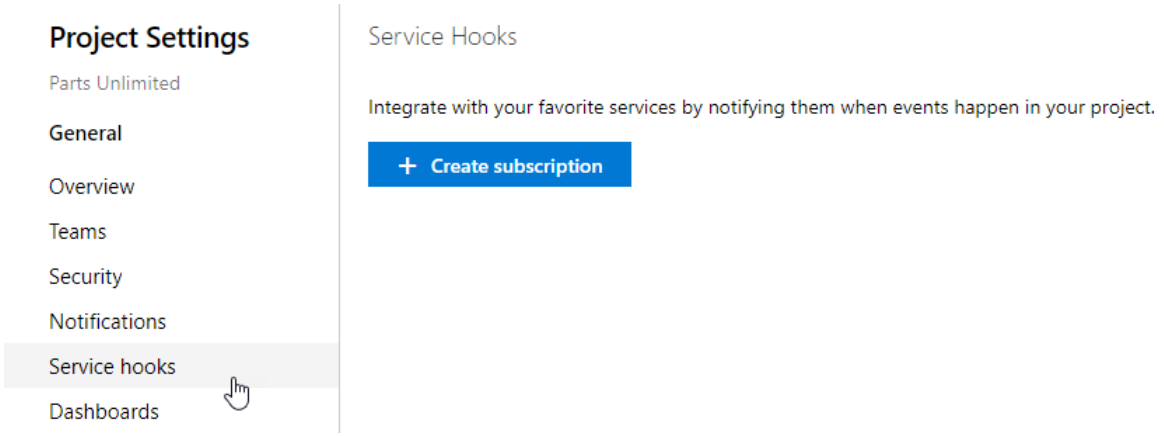
Let's now look at how a release pipeline can communicate with other services by using service hooks.

Azure DevOps can be integrated with a wide variety of other applications. It has built-in support for many applications and generic hooks for working with other applications. Let's look.

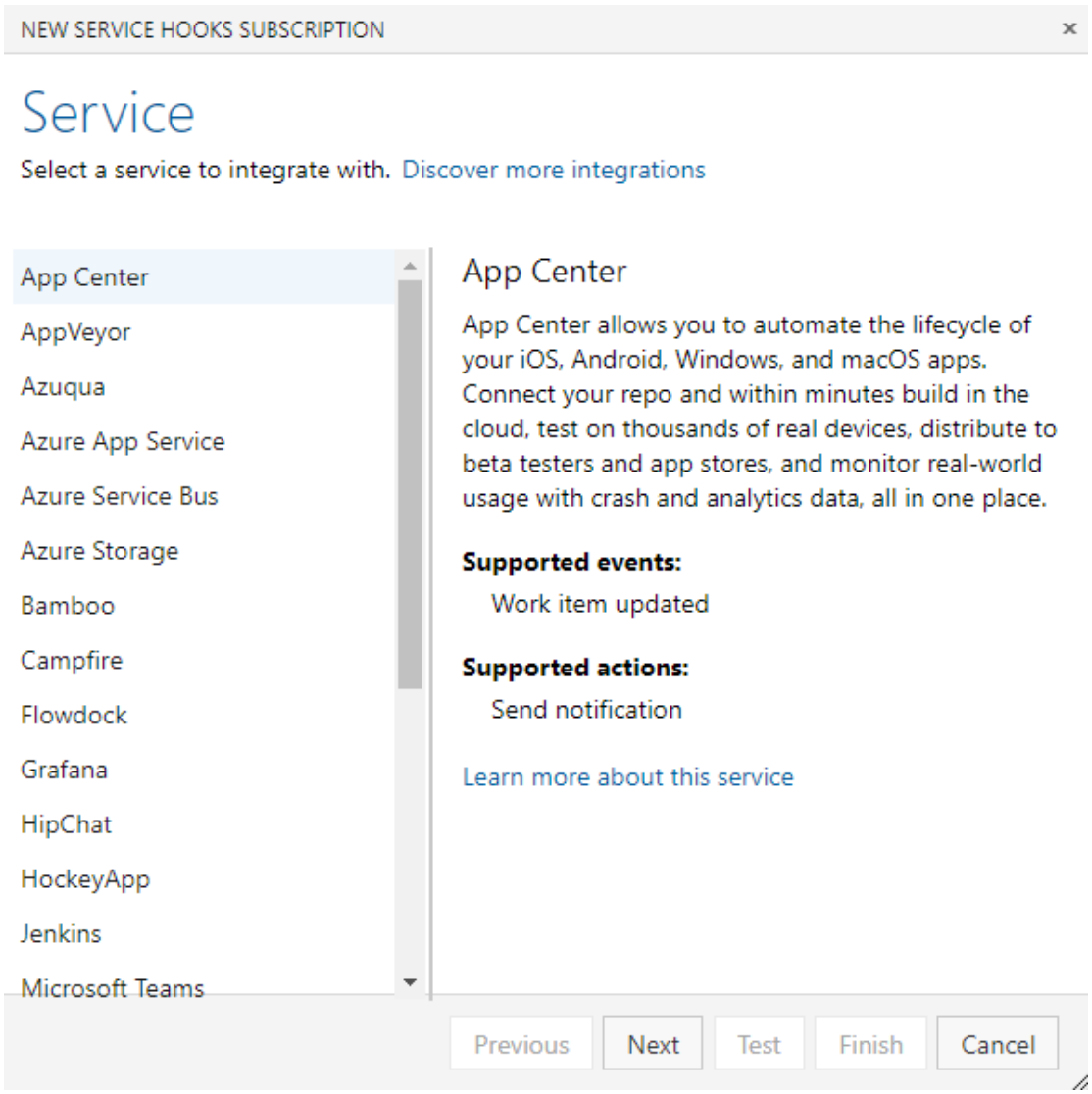
1. Below the main menu for the **Parts Unlimited** project, click **Project Settings** .



2. In the **Project settings** menu, click **Service hooks** .

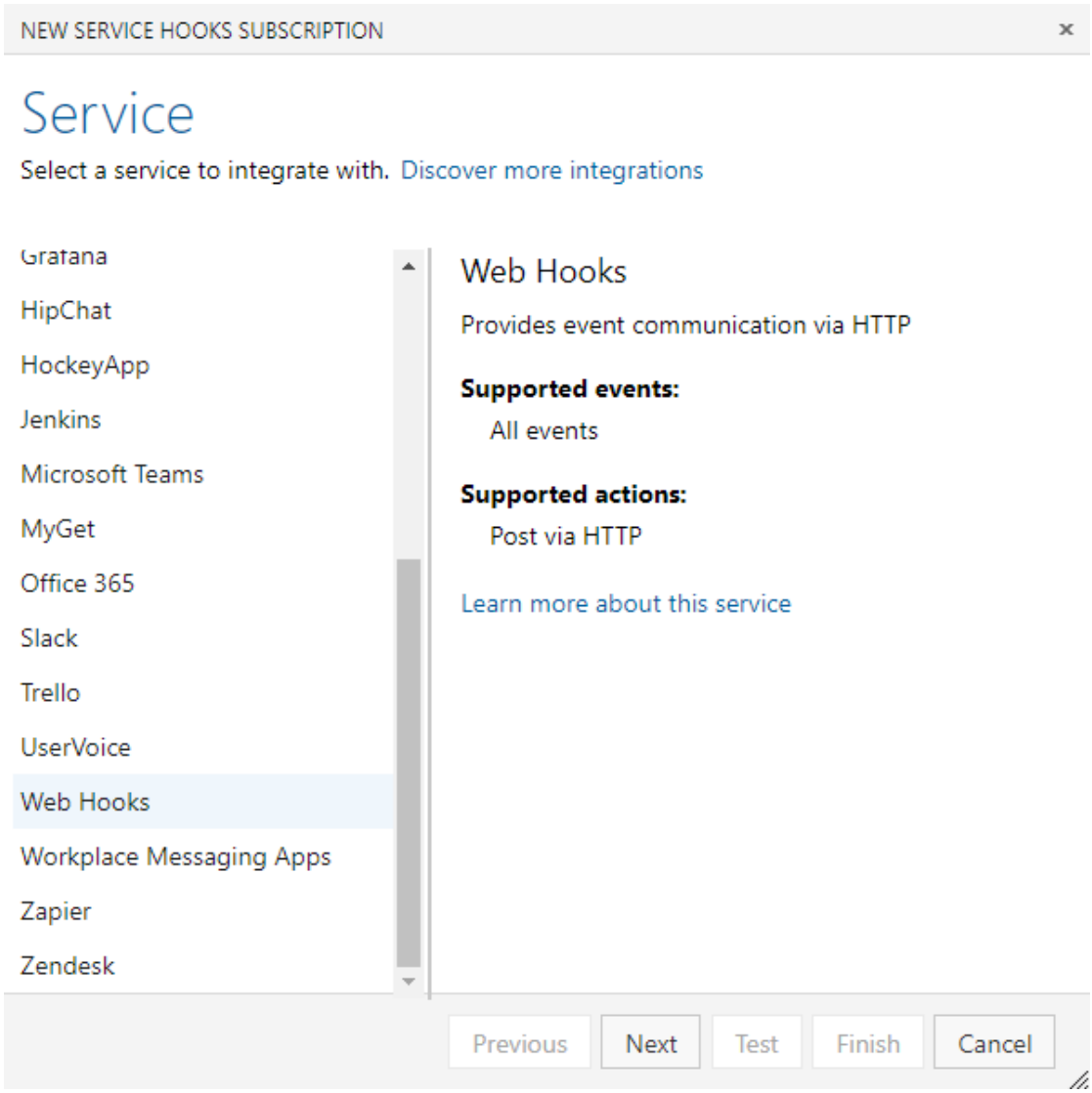


3. Click **+Create subscription** .



By using service hooks, we can notify other applications that an event has occurred within Azure DevOps. We could also send a message to a team in **Microsoft Teams** or **Slack** . We could also trigger an action in **Bamboo** or **Jenkins** .

4. Scroll to the bottom of the list of applications and click on **Web Hooks** .



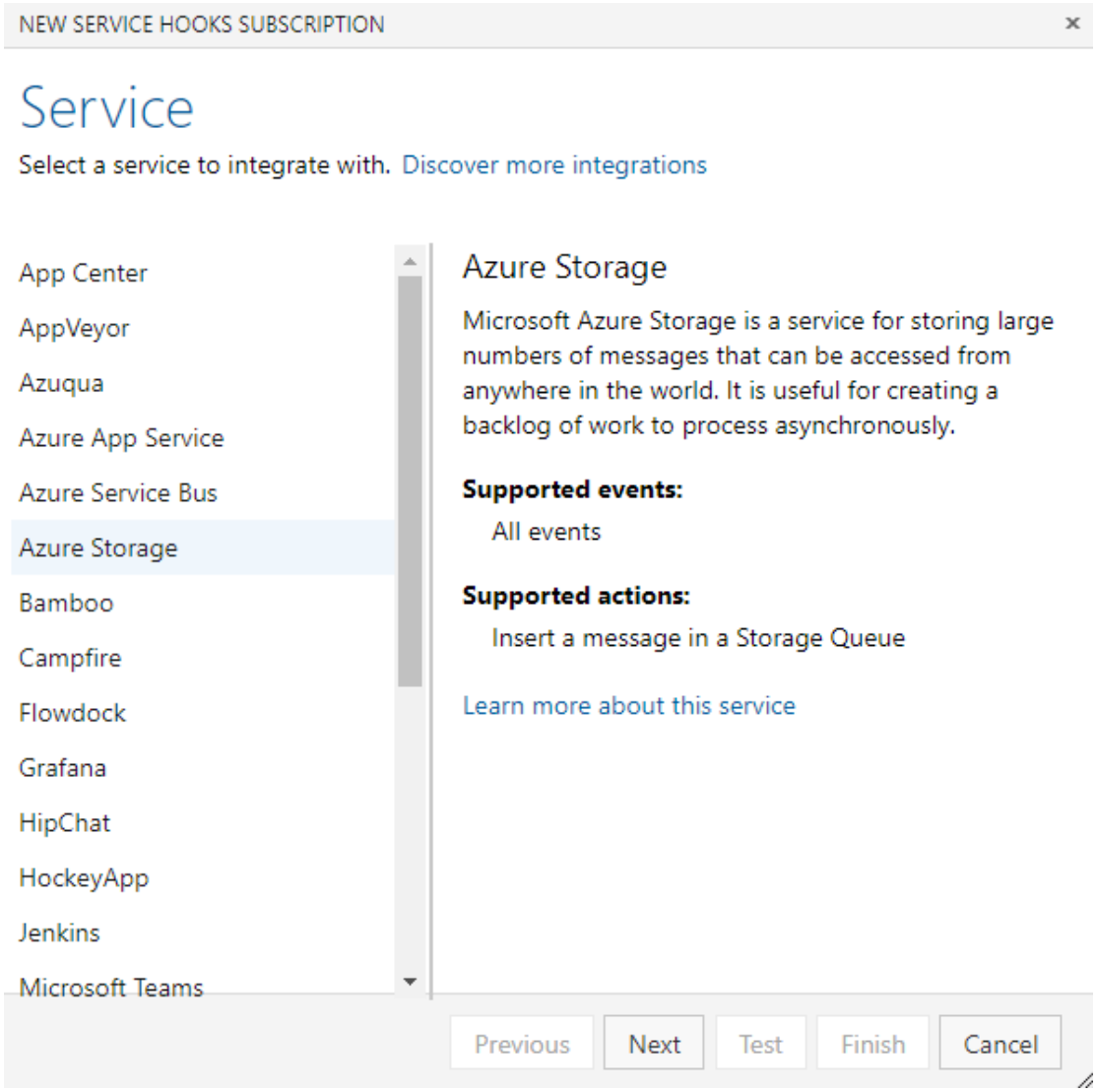
Suppose the application that you want to communicate with isn't in the list of available application hooks.

In that case, you can almost always use the **Web Hooks** option as a generic way to communicate. It allows you to make an HTTP POST when an event occurs.

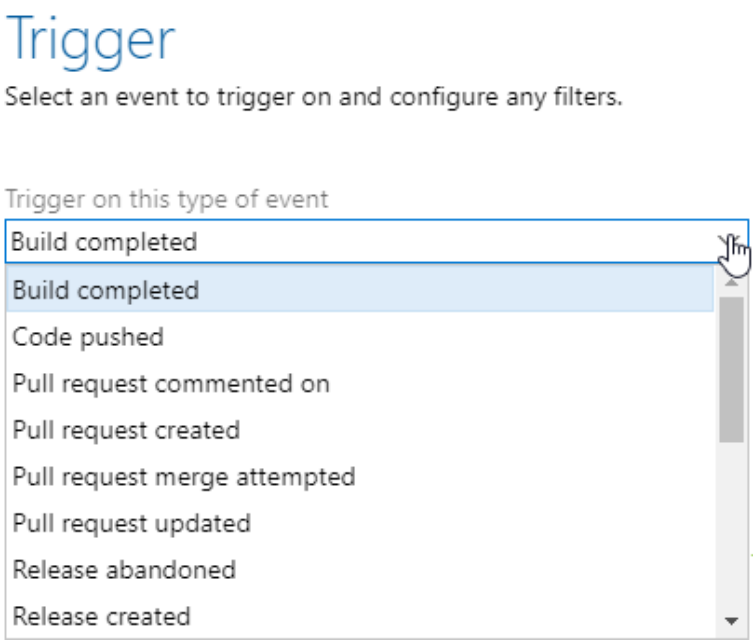
So, if, for example, you wanted to call an Azure Function or an Azure Logic App, you could use this option.

To demonstrate the basic process for calling web hooks, we'll write a message into a queue in the Azure Storage account that we have been using.

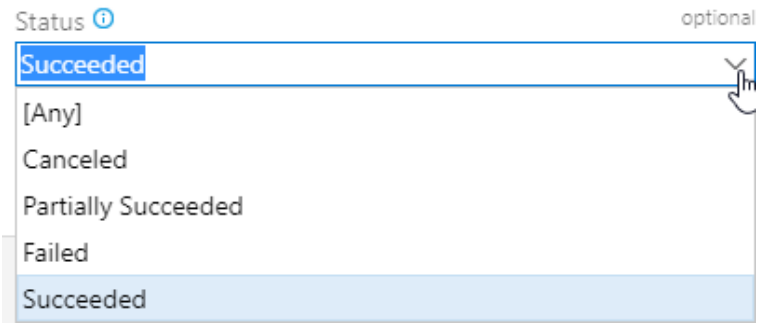
5. From the list of available applications, click **Azure Storage** .



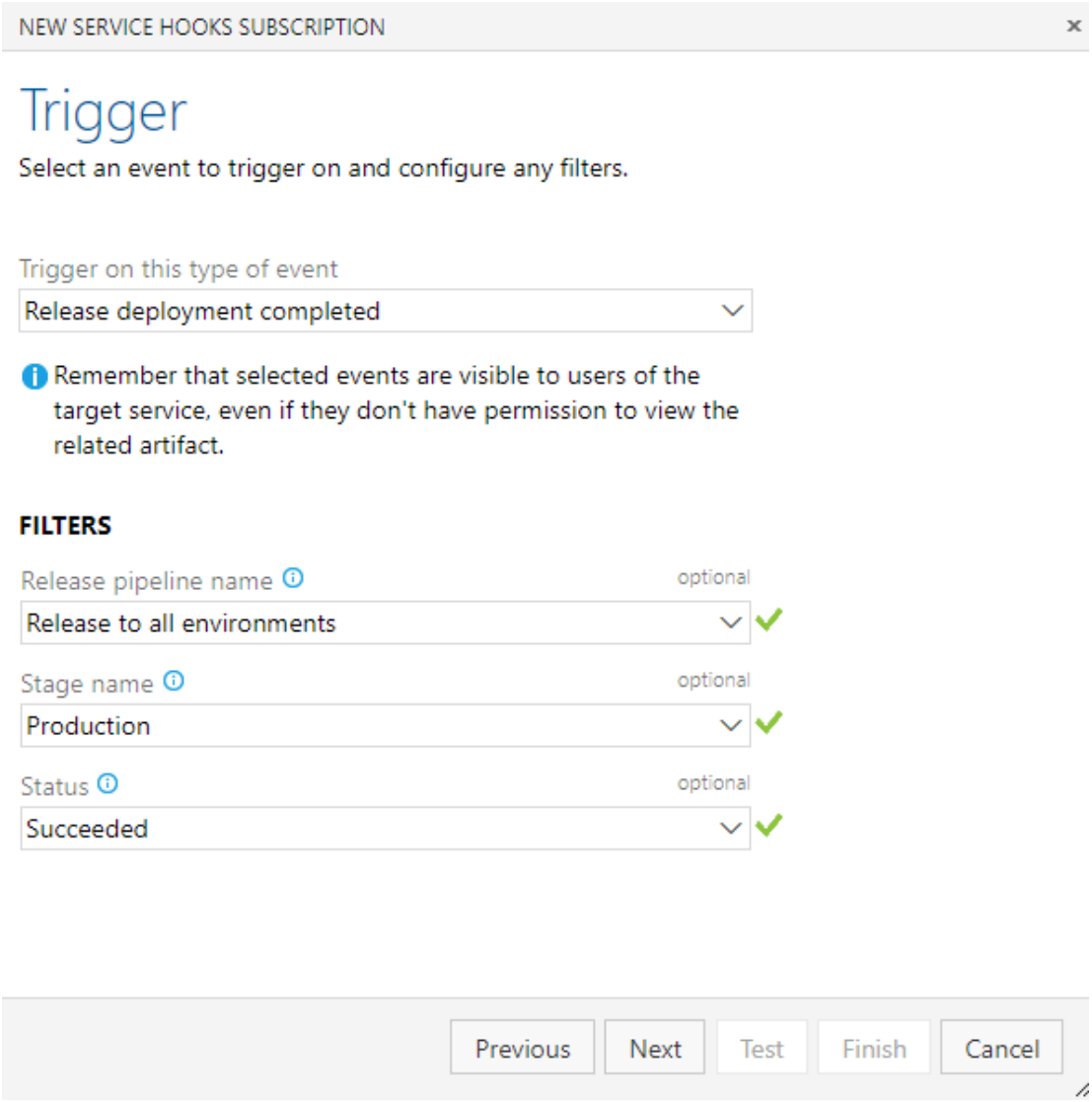
6. Click **Next** . On the **Trigger** page, we determine which event causes the service hook to be called. Click the drop-down for **Trigger on this type of event** to see the available event types.



7. Ensure that **Release deployment completed** is selected, then in the **Release pipeline name** select **Release to all environments** . For **Stage** , select **Production** . Drop down the list for **Status** and see the available options.

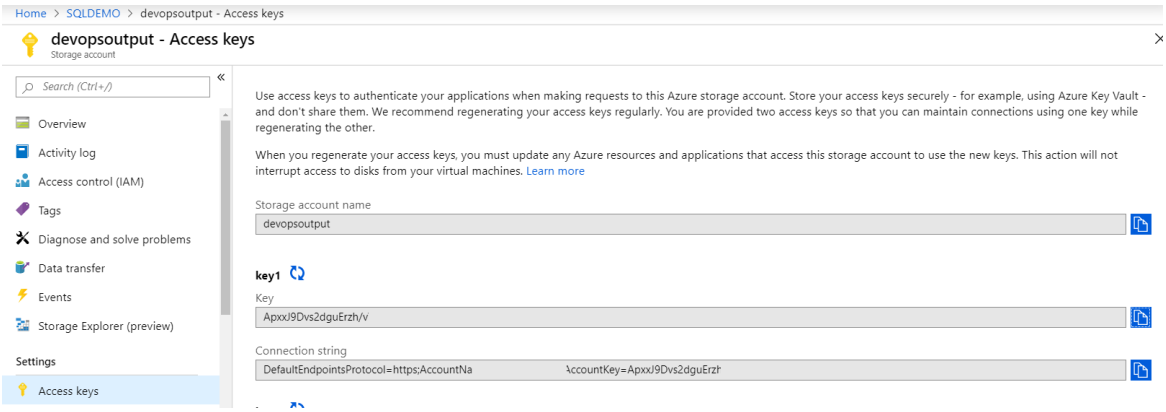


8. Ensure that **Succeeded** is selected, then click **Next** .

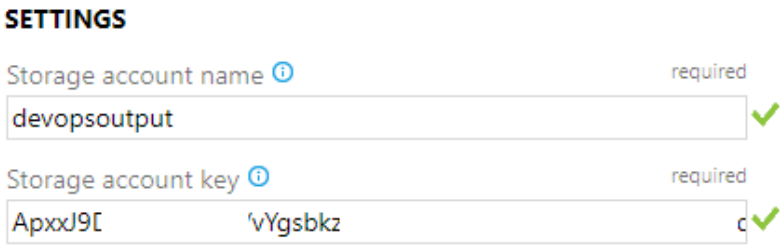


9. On the **Action** page, enter the name of your Azure storage account.

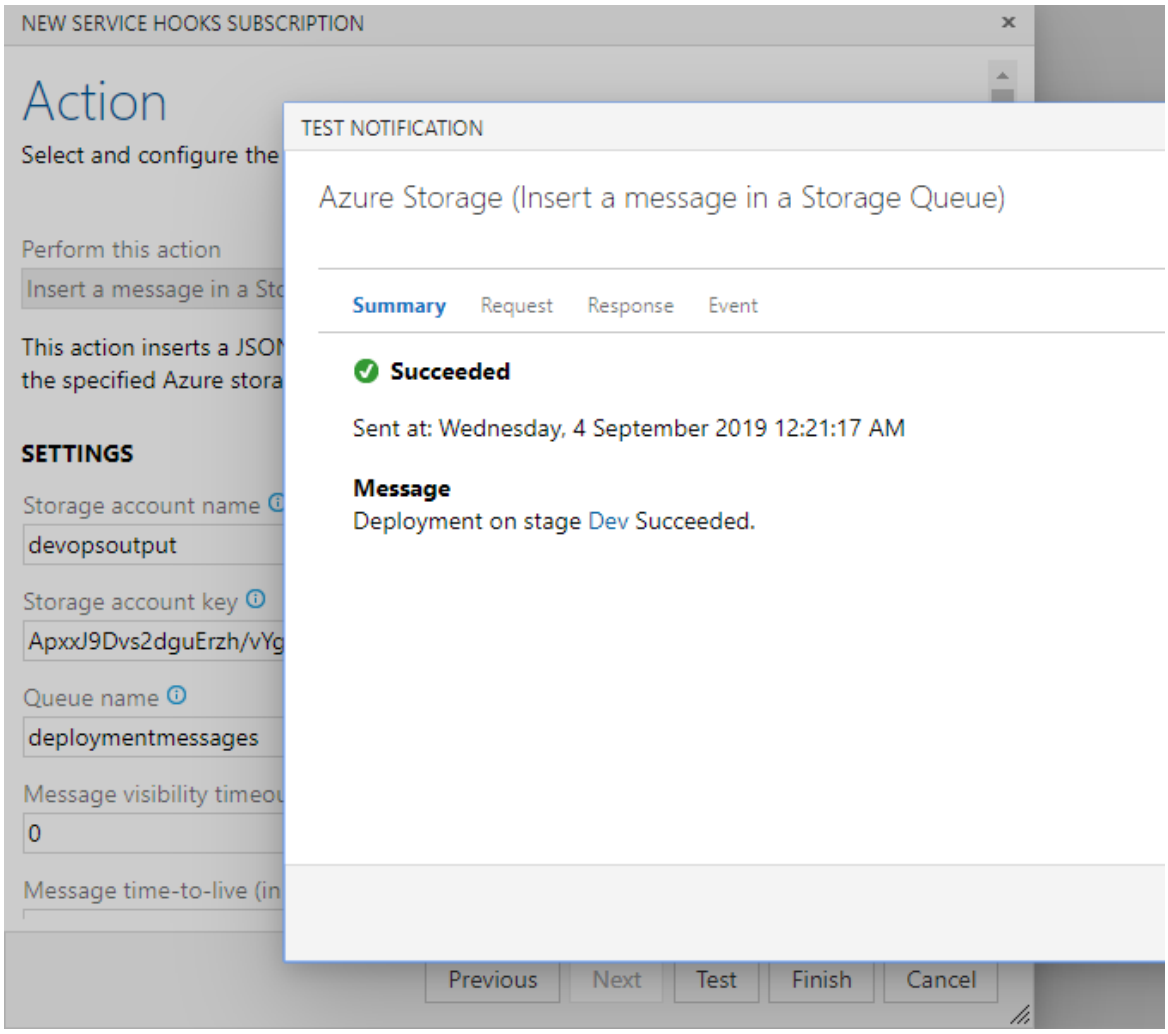
10. Open the Azure portal, and from the settings for the storage account, copy the value for Key in the **Access keys** section.



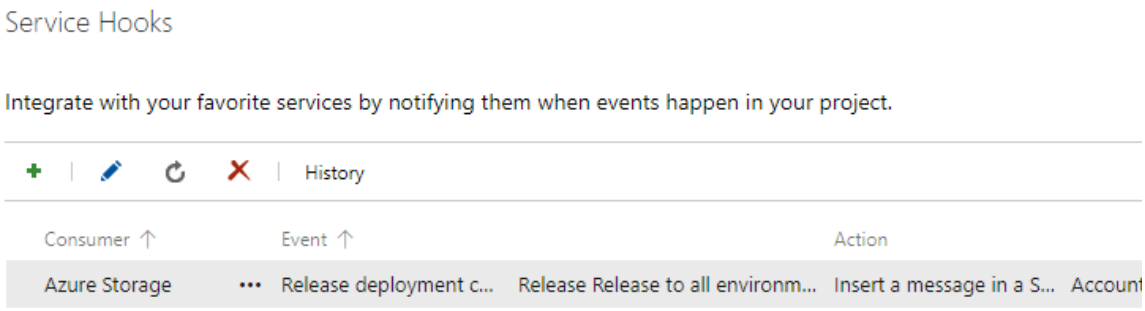
11. Back in the **Action** page in Azure DevOps, paste in the key.



12. For **Queue name,** enter **deploymentmessages** , then click **Test** .



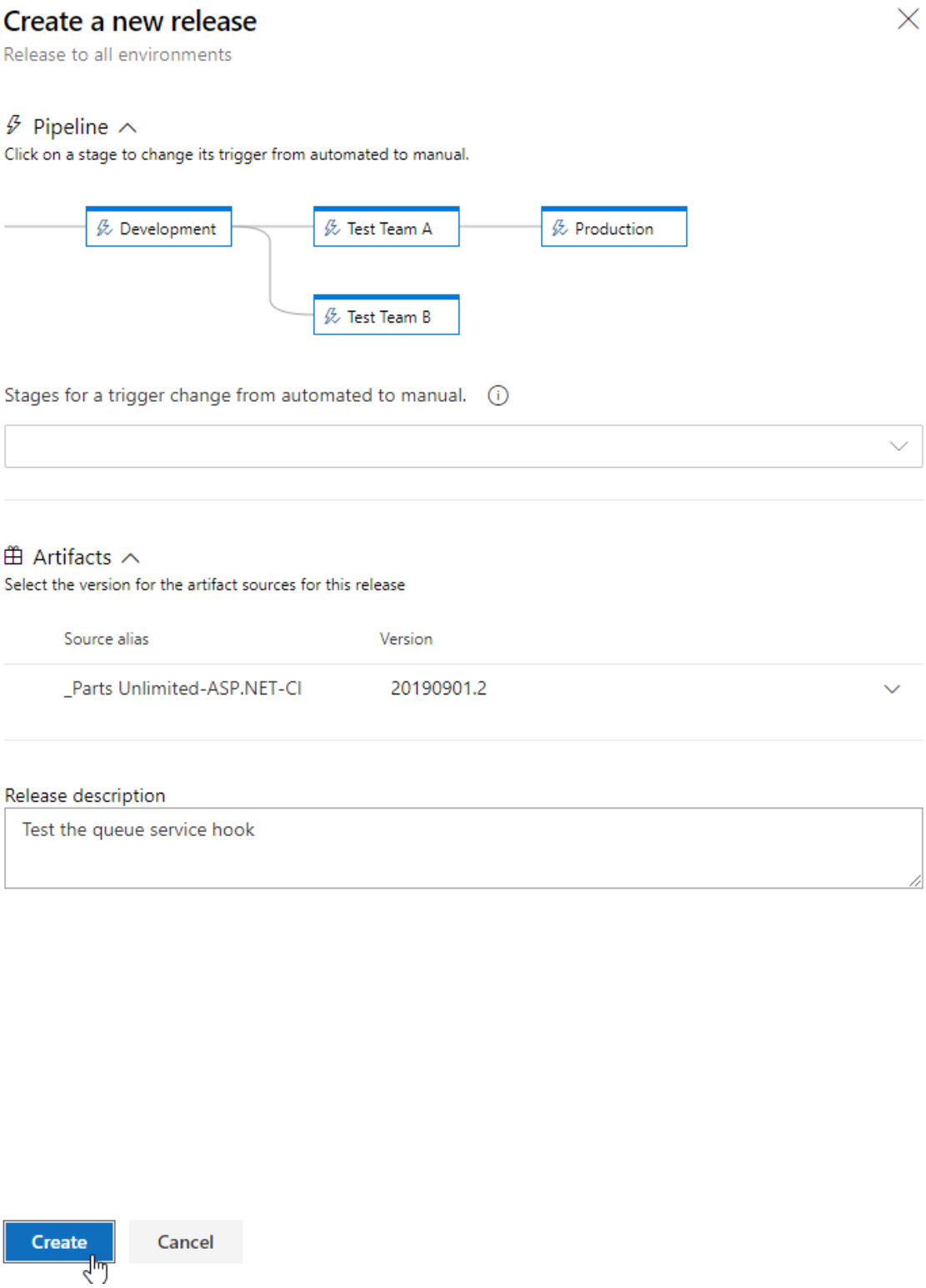
13. Make sure that the test succeeded, then click **Close** , and on the **Action** page, click **Finish** .



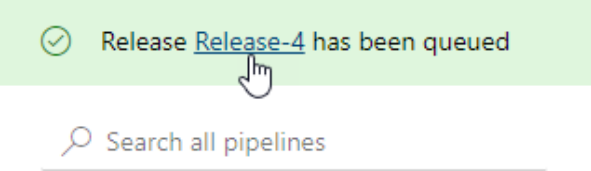
Create a release to test the service hook

Now that you've successfully added the service hook, it's time to test it.

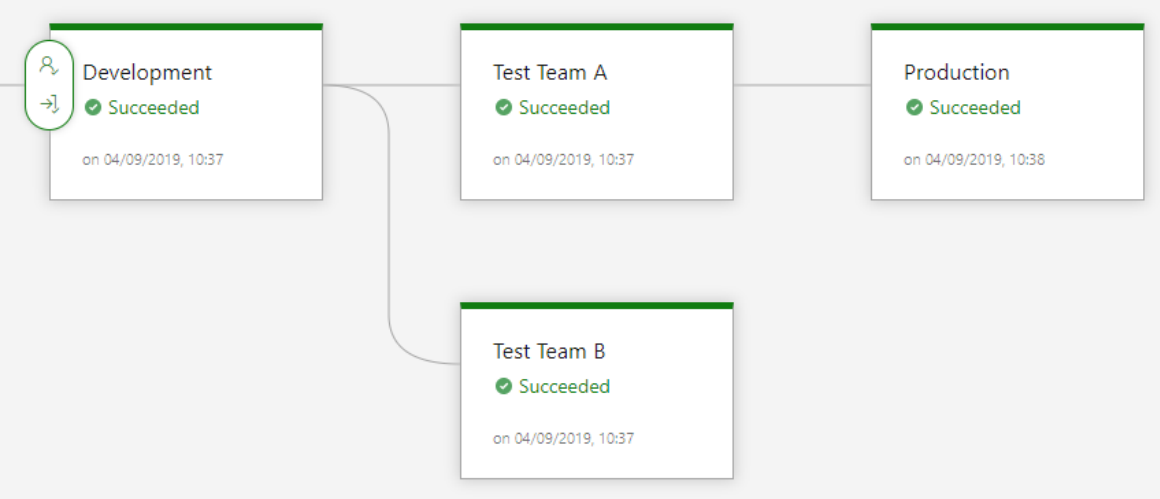
1. From the **Parts Unlimited** project's main menu, click Pipelines, click **Releases** , then click **Create release** , and in the **Create a new release** pane, enter **Test the queue service hook** for **Release description** , and click **Create** .



2. Click to view the release details.

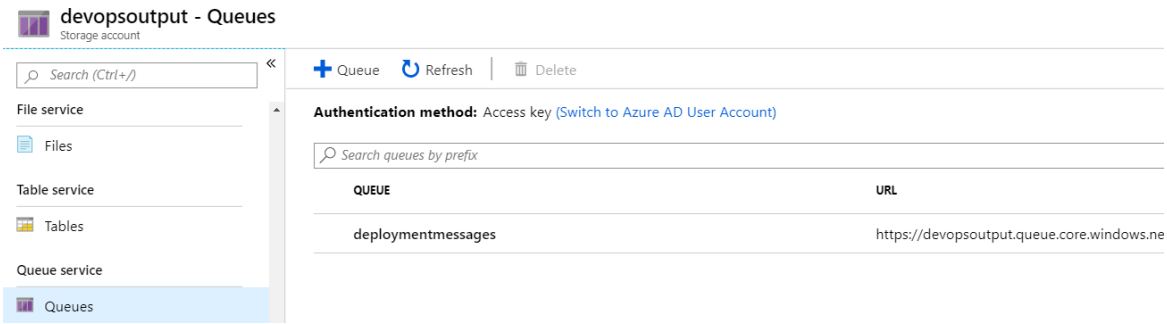


3. If the release is waiting for approval, click to approve it and wait for the release to complete successfully.

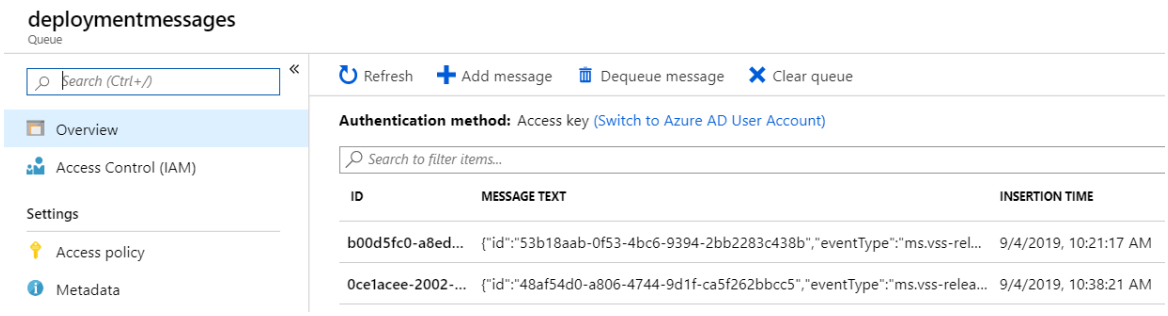


Check the queue contents

1. In the **Azure portal** , click **Queues** from the **Queue service** section in the blade for the storage account.



2. Click to open the **deploymentmessages** queue.



Note

If you have run multiple releases, you might have various messages.

3. Click the latest message (usually the bottom of the list) to open it and review the message properties, then close the **Message properties** pane.

You've successfully integrated this message queue with your Azure DevOps release pipeline.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Configure Azure DevOps notifications

Completed

- 2 minutes

After defining your target audience, you need to configure your notifications. Using Azure DevOps, you can help your team stay informed about your projects' activities.

You can configure notifications to be sent based on rules or subscriptions by default, out-of-the-box (OOB), created by you or the team or group administrator.

You can get a notification when the following items change:

- Work items.
- Code reviews.
- Pull requests.
- Source control files (TFVC or Git).
- Builds.
- Release.

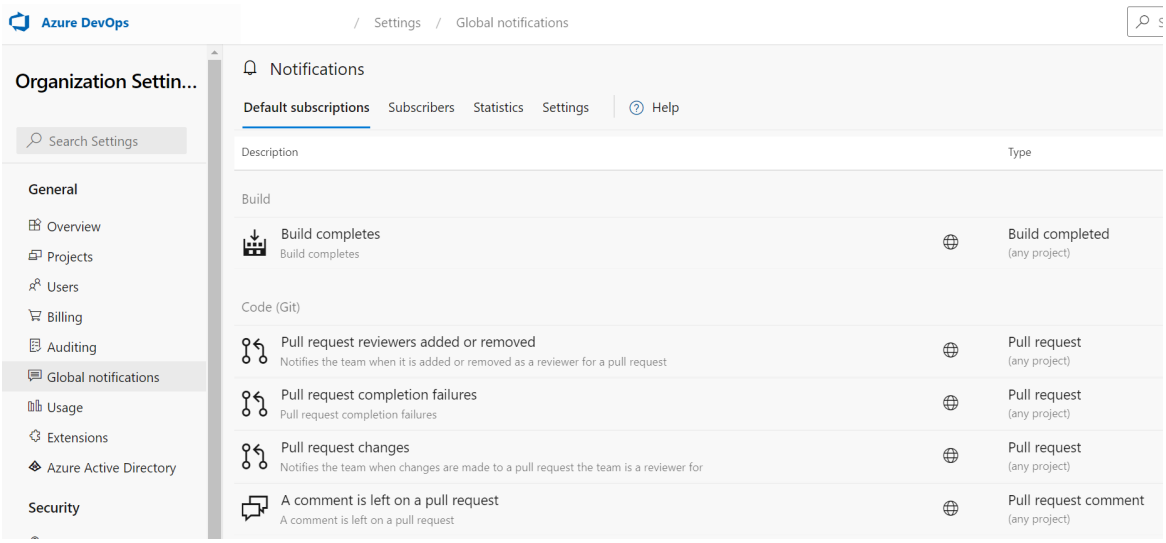
For example, you can get notified whenever your build completes, or your release fails.

There are four notification types that you can manage in Azure DevOps:

- Personal notifications.
- Team notifications.
- Project notifications.
- Global notifications.

For each notification, you have a set of specific steps to configure. The following steps show how to manage global notifications:

1. Open your Azure DevOps organization `https://dev.azure.com/**\{organization\}**/\_settings/organizationOverview .`
2. Click on Organization settings at the bottom left side.
3. Click on Global notifications under the General tab.



The Default subscriptions tab lists all default global subscriptions available. The globe icon on a notification subscription indicates the subscription is a default subscription. You can view all default notification subscriptions .

You can view and enable options available in the context menu (...) for each subscription.

Note

Only Project Collection Administrators can enable/disable any default subscription in this view. Project Collection Valid Users group can only view the details of the default subscription.

In the Subscribers tab, you can see users subscribed to each notification item. The Settings section shows the *Default delivery option* setting. All teams and groups inherit this setting.

You can see how to manage your personal notifications following manage your personal notifications .

For more information, see:

- Get started with notifications in Azure DevOps - Azure DevOps .
- Manage notifications for a team, project, organization, or collection - Azure DevOps .
- Events, subscriptions, and notifications - Azure DevOps.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Configure GitHub notifications

Completed

- 2 minutes

Github.com notifications provide updates about the activity that you've subscribed to. You can use the notifications inbox to customize, triage, and manage your updates.

You can choose to subscribe to notifications for:

- A conversation in a specific issue, pull request, or gist.
- All activity in a repository or team discussion.
- CI activity, such as the status of workflows in repositories set up with GitHub Actions.
- Repository issues, pull requests, releases, security alerts, or discussions (if enabled).

By default, you automatically watch all repositories you create and own by your personal account and subscribe to conversations when you have:

- Not disabled automatic watching for repositories or teams you've joined in your notification settings.
- Been assigned to an issue or pull request.
- Opened a pull request, issue, or created a team discussion post.
- Commented on a thread.
- Subscribed to a thread manually by clicking Watch or Subscribe.
- Had your username @mentioned .
- Changed the thread's state by closing an issue or merging a pull request.
- Had a team you're a member of @mentioned .

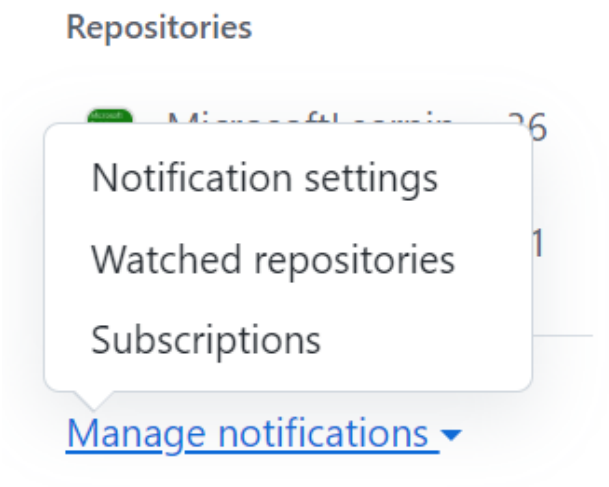
Tip

To unsubscribe from conversations, you can change your notification settings or directly unsubscribe or unwatch activity on GitHub.com. For more information, see " Managing your subscriptions ."

Notification settings

1. Click on the notification icon in the upper-right corner of any page.

2. Click on the notification settings under the list of repositories in the left sidebar.



3. On the notifications settings page, choose how you receive notifications.

Notifications

Choose how you receive notifications. These notification settings apply to the things you're watching.

Automatic watching

When you're given push access to a repository, automatically receive notifications for it.

- [ ] Automatically watch repositories

When you're added to or join a team, automatically receive notifications for that team's discussions.

- [x] Automatically watch teams

Participating

Notifications for the conversations you are participating in, or if someone cites you with an **@mention**.

- [x] Email
- [x] Web and Mobile

Watching

Notifications for all repositories, teams, or conversations you're watching.

- [x] Email
- [x] Web and Mobile

Dependabot alerts

When you're given access to Dependabot alerts, automatically receive notifications when a new vulnerability is found in one of your dependencies.

- [x] UI alerts
- [x] Command Line
- [x] Web

Receive security alert notifications via email

- [x] Email each time a vulnerability is found
- [x] Email a digest summary of vulnerabilities

Weekly security email digest

Actions

Notifications for workflow runs on repositories set up with GitHub Actions.

- [ ] Email
- [x] Web
- [x] Send notifications for failed workflows only

Organization alerts

When you are given admin permissions to an organization, automatically receive notifications when a new deploy key is added.

- [x] Email

Email notification preferences

Default notification email

Save

Choose which email updates you receive on conversations you're participating in or watching

- [x] Comments on Issues and Pull Requests
- [x] Pull Request reviews
- [x] Pull Request pushes
- [ ] Include your own updates

For more information, see:

- About notifications - GitHub Docs .
- Configuring notifications - GitHub Docs .
- Viewing your subscriptions - GitHub Docs .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Explore how to measure quality of your release process

Completed

- 2 minutes

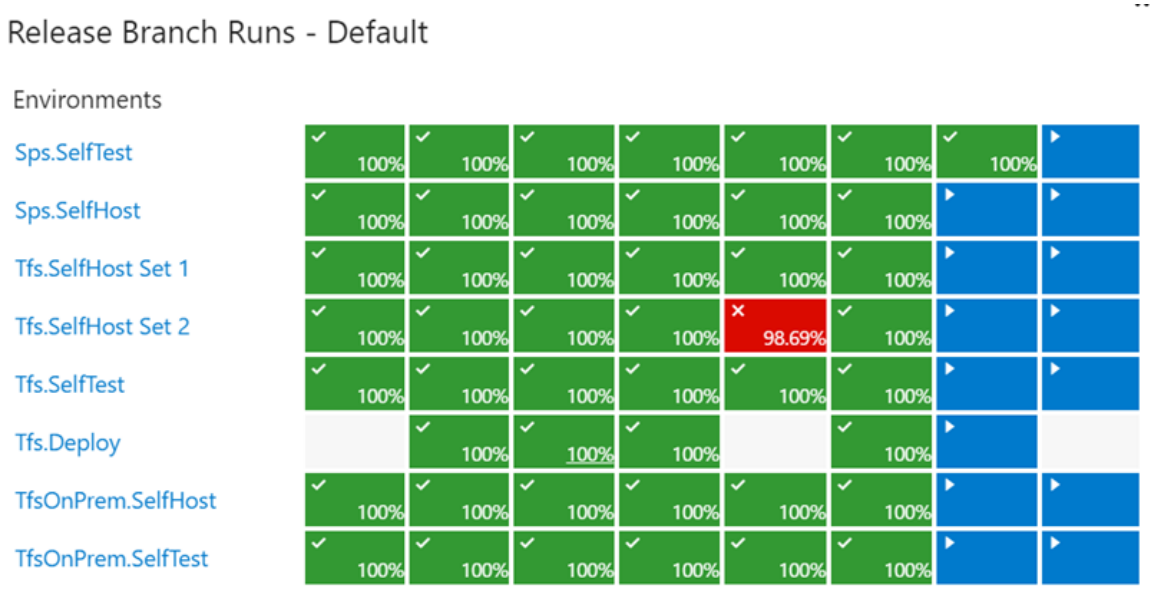
How do you measure the quality of your release process? The quality of your release process can't be measured directly because it's a process. What you can measure is how well your process works.

If your release process constantly changes, it might indicate something wrong with the process. If your releases continuously fail, and you regularly must update your release process to make it work, it might also suggest that something is wrong with your release process.

Maybe something is wrong with the schedule on which your release runs, and you notice that your release always fails on a particular day or at a specific time. Or your release always fails after the deployment to another environment. It might be an indication that some things are maybe dependent or related.

You can keep track of your release process quality by creating visualizations about the quality of all the releases following that same release process or release pipeline.

For example, we're adding a dashboard widget that shows you the status of every release.



The release also has a quality aspect, but it's tightly related to the quality of the deployment and package deployed. When we want to measure the quality of a release itself, we can do all kinds of checks within the pipeline.

You can execute all different types of tests like integration tests, load tests, or even UI tests while running your pipeline and checking the release's quality.

Using a quality gate is also a perfect way to check the quality of your release. There are many different quality gates. For example, a gate that monitors to check if everything is healthy on your deployment targets, work item gates that verify the quality of your requirements process.

You can add extra security and compliance checks. For example, do we follow the four-eyes principle, or do we have the proper traceability?

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Examine release notes and documentation

Completed

- 3 minutes

When you deploy a new release to a customer or install new software on your server, and you want to communicate what has been released to your customer, the usual way is to use release notes.

But where do the release notes come from? There are different ways to store your release notes.

Document store

An often-used way of storing release notes is by creating text files or documents in some document store. This way, the release notes are stored together with other documents.

The downside of this approach is that there's no direct connection between the release in the release management tool and the release notes that belong to this release.

Wiki

The most used way for customers is to store the release notes in a Wiki. For example:

- Confluence from Atlassian
- SharePoint Wiki
- SlimWiki
- Wiki in Azure DevOps

Release notes are created as a page in the wiki and by using hyperlinks. Relations can be associated with the build, the release, and the artifacts.

In the codebase

When you look at it, release notes belong strictly to the release of the features you implemented and your code. In that case, the best option might be to store release notes as part of your code repository.

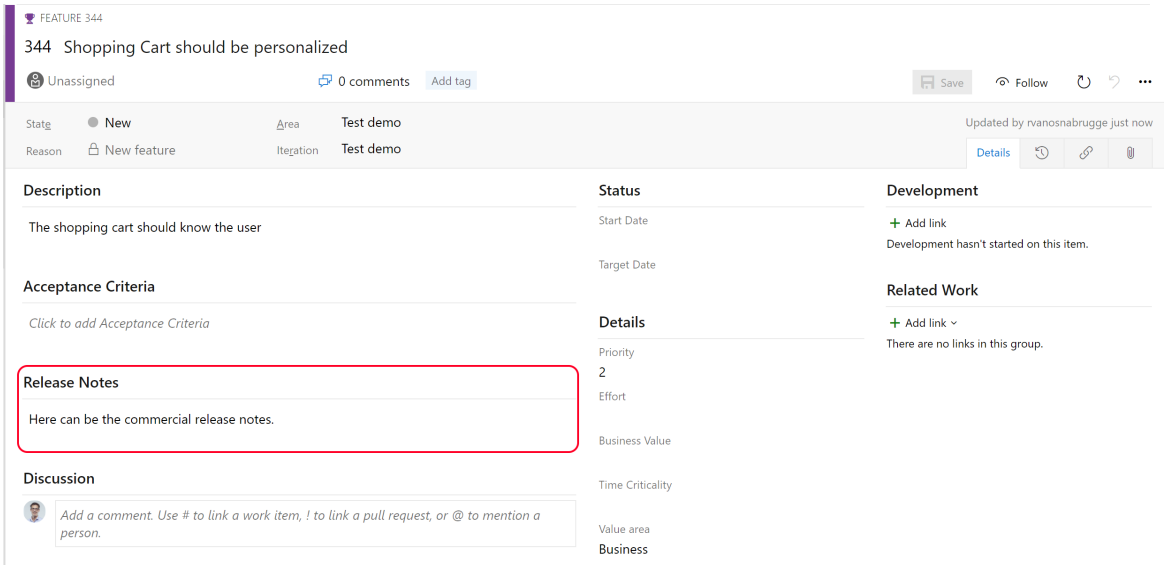
Once the team completes a feature, they or the product owner also write the release notes and save them alongside the code. This way, it becomes living documentation because the notes change with the rest of the code.

In a work item

Another option is to store your release notes as part of your work items. Work items can be Bugs, Tasks, Product Backlog Items, or User Stories.

You can create or use a different field within the work item to save release notes in work items. In this field, you type the publicly available release notes that will be communicated to the customer.

With a script or specific task in your build and release pipeline, you can generate the release notes and store them as artifacts or publish them to an internal or external website.



- Generate Release Notes Build Task.
- Wiki Updater Tasks.
- Atlassian Confluence.
- Azure DevOps Wiki.

There's a difference between functional and technical documentation. Also, a difference between documentation designing the product, primarily written upfront, and documentation describing the product afterward, like manuals or help files.

Storing technical documentation about your products in the design phase is done on a document-sharing portal, like SharePoint or Confluence.

Creating a wiki is a better and more modern way to store your documentation. Wiki's don't contain Documents, Presentations, or Spreadsheets but text files called Markdown Files.

These markdowns can refer to pictures, hold code samples, and be part of your code repository. Code repositories can deal well with text files. Changes and history can be easily tracked by using the native code tools.

However, the most significant advantage of using a Wiki instead of documents is that a Wiki is accessible to everyone in your team. People can work together on the documentation instead of waiting for each other when working on the same document by giving all the team members the correct permissions.

Manuals or documentation you release together with the product should be treated as source code. When the product changes and new functionality are added, the documentation needs to change.

You can store the documentation as part of your code repository or create a new repository containing your documentation. In any case, the documentation should be treated the same way as your source code. Create a documentation artifact in your build pipeline and deliver this artifact to the release pipeline.

The release pipeline can then deploy the documentation to a site or include it in the boxed product.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Examine considerations for choosing release management tools

Completed

- 5 minutes

When choosing the right Release Management tool, you should look at the possibilities of all the different components and map them to your needs.

Many tools are available in the marketplace, which we'll discuss in the next chapter. The most important thing to notice is that not every vendor or tool treats Release Management similarly.

The tools in the marketplace can be divided into two categories.

- Tools that can do Build and Continuous Integration and Deployment.
- Tools that can do Release Management.

In many cases, companies only require the deployment part of Release Management.

Many build or release tools can perform deployment or installation. Primarily because the technical aspect of the release is executing a script or running a program, Release Management, which requires approvals, quality gates, and different stages, needs a different kind of tool that tightly integrates with the build, and CI tools aren't the same.

Artifacts and artifact source

Artifacts can come from different sources. Treating your artifact as a versioned package needs to be stored somewhere before your release pipeline consumes it. Considerations for choosing your tool can be:

- Which Source Control systems are supported?
- Can you have one or more artifact sources in your release pipeline? In other words, can you combine artifacts from different sources into one release?
- Does it integrate with your build server?
- Does it support other build servers?
- Does it support container registries?
- How do you secure the connection to the artifact source?
- Can you extend the artifact sources?

Triggers and schedules

Triggers are an essential component in the pipeline. Triggers are required to start a release, but if you want multiple stages, create a deployment. Considerations for choosing your trigger can be:

- Does your system support Continuous Deployment triggers?
- Can you trigger releases from an API (for integration with other tools)?
- Can you schedule releases?
- Can you schedule and trigger each stage separately?

Approvals and gates

Starting a release using scripts, executables, or deployable artifacts doesn't differentiate between a Continuous Integration/Build tool and a Release Management tool. Adding a release approval workflow to the pipeline is the critical component that does make the difference. Considerations When it comes to approvals:

- Do you need approvals for your release pipeline?
- Is the approvers part of your company? Do they need a tool license?
- Do you want to use manual or automatic approvals? Or both?
- Can you approve an API (integration with other tools)
- Can you set up a workflow with approvers (optional and required)?
- Can you have different approvers per stage?
- Can you've more than one approver? Do you need them all to approve?
- What are the possibilities for automatic approval?
- Can you have a manual approval or step in your release pipeline?

Stages

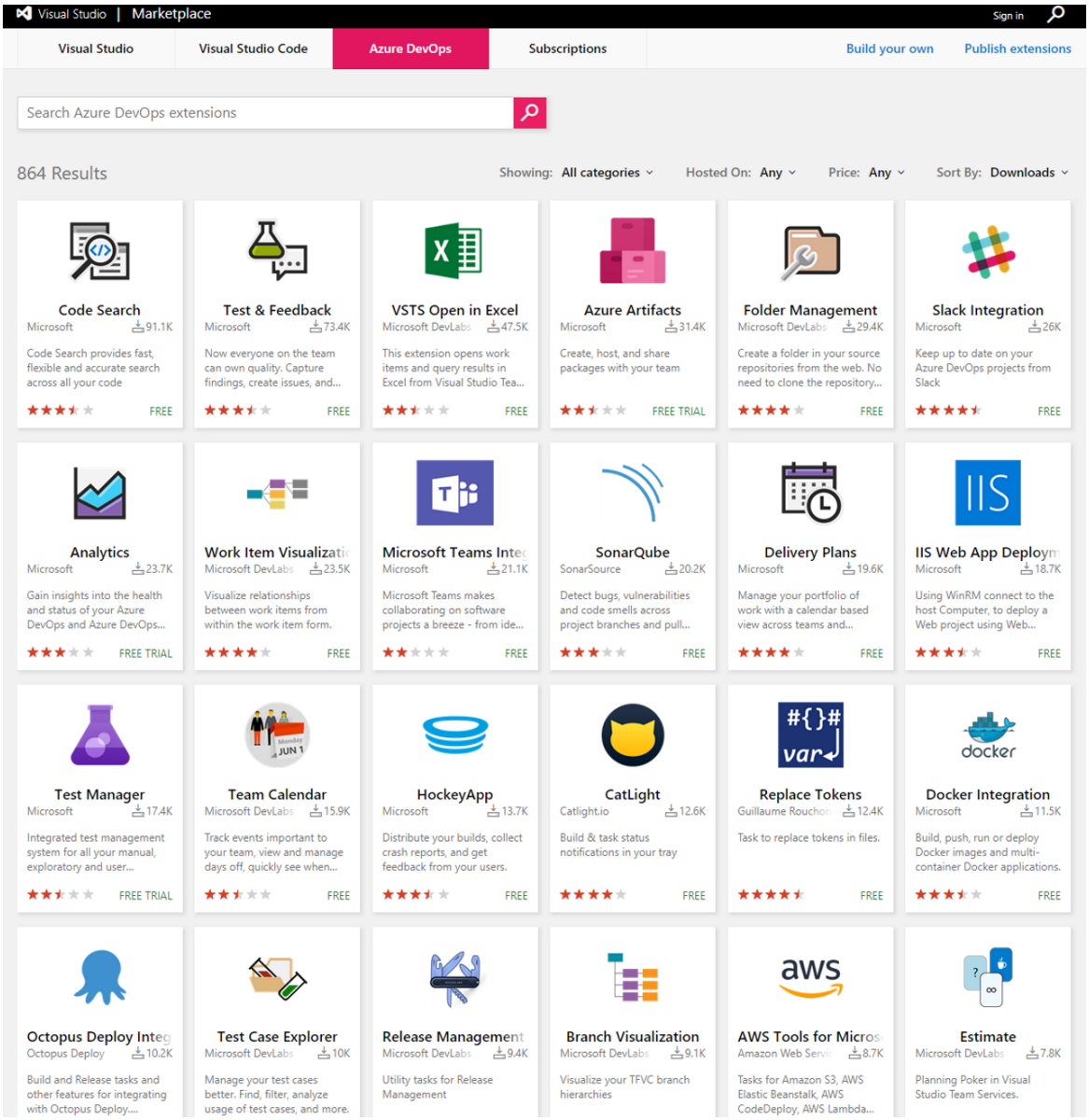
Running a Continuous Integration pipeline that builds and deploys your product is a commonly used scenario. But what if you want to deploy the same release to different environments? When choosing the right release management tool, you should consider the following things when it comes to stages (or environments)

- Can you use the same artifact to deploy to different stages?
- Can you differ the configuration between the stages?
- Can you have different steps for each stage?
- Can you follow the release between the stages?
- Can you track the artifacts/work items and source code between the stages?

Build and release tasks.

Finally, the work needs to be done within the pipeline. It isn't only about the workflow and orchestration; the code must also be deployed or installed. Things to consider when it comes to the execution of tasks.

- How do you create your steps? Is it running a script (bat, shell, PowerShell CLI), or are there specialized tasks?
- Can you create your tasks?
- How do tasks authenticate to secure sources?
- Can tasks run on multiple platforms?
- Can tasks be easily reused?
- Can you integrate with multiple environments? (Linux, Windows, Container Clusters, PaaS, Cloud)
- Can you control the tasks that are used in the pipeline?



Traceability, auditability, and security

One of the essential things in enterprises and companies that need to adhere to compliance frameworks is:

- Traceability.
- Auditability.
- Security.

Although it isn't explicitly related to a release pipeline, it's essential.

When it comes to compliance, three principles are fundamental:

- four-eyes principle
  - Does at least one other person review the deployed artifact?
  - Is the person that deploys another person the one that writes the code?
- Traceability
  - Can we see where the released software originates from (which code)?
  - Can we see the requirements that led to this change?
  - Can we follow the requirements through the code, build, and release?
- Auditability
  - Can we see who, when, and why the release process changed?
  - Can we see who, when, and why a new release has been deployed?

Security is vital in it. It isn't ok when people can do everything, including deleting evidence. Setting up the right roles, permissions, and authorization is essential to protect your system and pipeline.

When looking at an appropriate Release Management tool, you can consider the following:

- Does it integrate with your company's Active Directory?
- Can you set up roles and permissions?
- Is there a change history of the release pipeline itself?
- Can you ensure the artifact didn't change during the release?
- Can you link the requirements to the release?
- Can you link source code changes to the release pipeline?
- Can you enforce approval or the four-eyes principle?
- Can you see the release history and the people who triggered the release?

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Explore common release management tools

Completed

- 3 minutes

The following tools are mainstream in the current ecosystem. You'll find links to the product websites to explore the product and see if it fits the needs we described in the previous chapter.

- What Artifacts and Artifact sources does the tool support?
- What Triggers and Schedules?
- Does the tool support Approvals and gates?
- Can you define multiple stages?
- How do the Build and Release Tasks work?
- How does the tool deal with Traceability, Auditability, and Security?
- What is the Extensibility model?

Per tool is indicated if it's part of a more extensive suite. Integration with a bigger suite gives you many advantages regarding traceability, security, and auditability. Numerous integrations are already there out of the box.

GitHub Actions

GitHub Actions help you build, test, and deploy your code. You can implement continuous integration and continuous delivery (CI/CD) that allows you to make code reviews, branch management, and issue triaging work the way you want.

- Trigger workflows with various events.
- Configure environments to set rules before a job can proceed and to limit access to secrets.
- Use concurrency to control the number of deployments running at a time.

Links

- GitHub Actions .
- Understanding GitHub Actions .
- Essential features of GitHub Actions .
- Deploying with GitHub Actions .

Azure Pipelines

Azure Pipelines helps you implement a build, test, and deployment pipeline for any app.

Tutorials, references, and other documentation show you how to configure and manage the continuous integration and Continuous Delivery (CI/CD) for the app and platform of your choice.

- Hosted on Azure as a SaaS in multiple regions and available as an on-premises product.
- Complete Rest API for everything around Build and Release Management.
- Integration with many build and source control systems
    - GitHub.

    - Azure Repos.
    - Jenkins.
    - Bitbucket.
    - and so on.
- Cross-Platform support, all languages, and platforms.
- Rich marketplace with extra plugins, build tasks and release tasks, and dashboard widgets.
- Part of the Azure DevOps suite. Tightly integrated.
- Fully customizable.
- Manual approvals and Release Quality Gates supported.
- Integrated with (Azure) Active Directory.
- Extensive roles and permissions.

Links

- Azure Pipelines .
- Building and Deploying your Code with Azure Pipelines .

Jenkins

Jenkins's leading open-source automation server provides hundreds of plugins to support building, deploying, and automating any project.

- On-premises system. They're offered as SaaS by a third party.
- No part of a bigger suite.
- Industry-standard, especially in the full-stack space.
- Integrates with almost every source control system.
- A rich ecosystem of plugins.
- CI/Build tool with deployment possibilities.
- No release management capabilities.

Links

- Jenkins .
- Tutorial: Jenkins CI/CD to deploy an ASP.NET Core application to Azure Web App service .
- Azure Friday - Jenkins CI/CD with Service Fabric .

Circle CI

CircleCI's continuous integration and delivery platform help software teams rapidly release code with confidence by automating the build, test, and deploy process.

CircleCI offers a modern software development platform that lets teams ramp quickly, scale easily, and build confidently every day.

- CircleCI is a cloud-based system or an on-premises system.
- Rest API—you have access to projects, builds, and artifacts.
- The result of the build is going to be an artifact.
- Integration with GitHub and BitBucket.
- Integrates with various clouds.
- Not part of a bigger suite.
- Not fully customizable.

Links

- CircleCI .
- How to get started on CircleCI 2.0: CircleCI 2.0 Demo

GitLab Pipelines

GitLab helps teams automate the release and delivery of their applications to shorten the delivery lifecycle, streamline manual processes and accelerate team velocity.

With Continuous Delivery (CD) built into the pipeline, deployment can be automated to multiple environments like staging and production and support advanced features such as canary deployments.

Because the configuration and definition of the application are version controlled and managed, it's easy to configure and deploy your application on demand.

Link

- GitLab

Atlassian Bamboo

Bamboo is a continuous integration (CI) server that can automate the release management for a software application, creating a Continuous Delivery pipeline.

Link

- Atlassian Bamboo

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Knowledge check

Completed

- 4 minutes

Choose the best response for each question. Then select **Check your answers** .

Check your knowledge

1.

Which of the following choices allows automatic collection of health signals from external services and promotes the release when all the signals are successful?

○

Release gates.

○

Events.

○

Service Hooks.

2.

Which of the following choices are raised when certain actions occur, like when a release is started or a build completed?

○

Service Hooks.

○

Events.

○

Release gates.

3.

Which of the following choices are usually emails you receive when an action occurs to which you're subscribed?

○

Notifications.

○

Service Hooks.

Events.

Check your answers

You must answer all questions before checking your work.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Summary

Completed

- 1 minute

This module described how to automate the inspection of health events, configure notifications, and set up service hooks to monitor pipelines.

You learned how to describe the benefits and usage of:

- Implement automated inspection of health.
- Create and configure events.
- Configure notifications.
- Create service hooks to monitor the pipeline.

Learn more

- DevOps checklist - Azure Design Review Framework | Microsoft Docs .
- Events, subscriptions, and notifications - Azure DevOps | Microsoft Docs .
- Integrate with service hooks - Azure DevOps | Microsoft Docs .
- Build Quality Indicators report - Azure DevOps Server | Microsoft Docs .

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .