**Create a release pipeline**

Advanced

Administrator

Developer

DevOps Engineer

Security Engineer

Security Operations Analyst

Service Adoption Specialist

Solution Architect

Technology Manager

Azure

Azure Artifacts

Azure Boards

Azure Cloud Services

Azure DevOps

Azure Pipelines

Azure Repos

Azure Test Plans

GitHub

This module describes Azure Pipelines capabilities, build, and release tasks.

**Learning objectives**

By the end of this module, you're able to:

* 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.

[**Start**](https://learn.microsoft.com/en-us/training/modules/create-release-pipeline-devops/1-introduction/)Add

**Prerequisites**

None

**This module is part of these learning paths**

* [AZ-400: Design and implement a release strategy](https://learn.microsoft.com/training/paths/az-400-design-implement-release-strategy/)

**Module assessment**

Assess your understanding of this module. Sign in and answer all questions correctly to earn a pass designation on your profile.

[**Take the module assessment**](https://learn.microsoft.com/training/modules/create-release-pipeline-devops/12-knowledge-check/)

* [Introduction](https://learn.microsoft.com/en-us/training/modules/create-release-pipeline-devops/1-introduction)1 min
* [Describe Azure DevOps release pipeline capabilities](https://learn.microsoft.com/en-us/training/modules/create-release-pipeline-devops/2-describe-azure-devops-capabilities)3 min
* [Explore release pipelines](https://learn.microsoft.com/en-us/training/modules/create-release-pipeline-devops/3-explore-release-pipelines)3 min
* [Explore artifact sources](https://learn.microsoft.com/en-us/training/modules/create-release-pipeline-devops/4-explore-artifact-sources)4 min
* [Choose the appropriate artifact source](https://learn.microsoft.com/en-us/training/modules/create-release-pipeline-devops/5-choose-appropriate-artifact-source)3 min
* [Examine considerations for deployment to stages](https://learn.microsoft.com/en-us/training/modules/create-release-pipeline-devops/6-examine-considerations-for-deployment-to-stages)3 min
* [Explore build and release tasks](https://learn.microsoft.com/en-us/training/modules/create-release-pipeline-devops/7-explore-build-release-tasks)4 min
* [Explore custom build and release tasks](https://learn.microsoft.com/en-us/training/modules/create-release-pipeline-devops/8-explore-custom-build-release-tasks)3 min
* [Explore release jobs](https://learn.microsoft.com/en-us/training/modules/create-release-pipeline-devops/9-explore-release-jobs)4 min
* [Understand database deployment task](https://learn.microsoft.com/en-us/training/modules/create-release-pipeline-devops/10-understand-database-deployment-task)3 min
* [Configure Pipelines as Code with YAML](https://learn.microsoft.com/en-us/training/modules/create-release-pipeline-devops/11-configure-pipelines-code-with-yaml)1 hr
* [Knowledge check](https://learn.microsoft.com/en-us/training/modules/create-release-pipeline-devops/12-knowledge-check)5 min
* [Summary](https://learn.microsoft.com/en-us/training/modules/create-release-pipeline-devops/13-summary)1 min

**Introduction**

Completed100 XP

* 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.

**Next unit: Describe Azure DevOps release pipeline capabilities**

**Describe Azure DevOps release pipeline capabilities**

Completed100 XP

* 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.

Expand 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. |
| 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. |

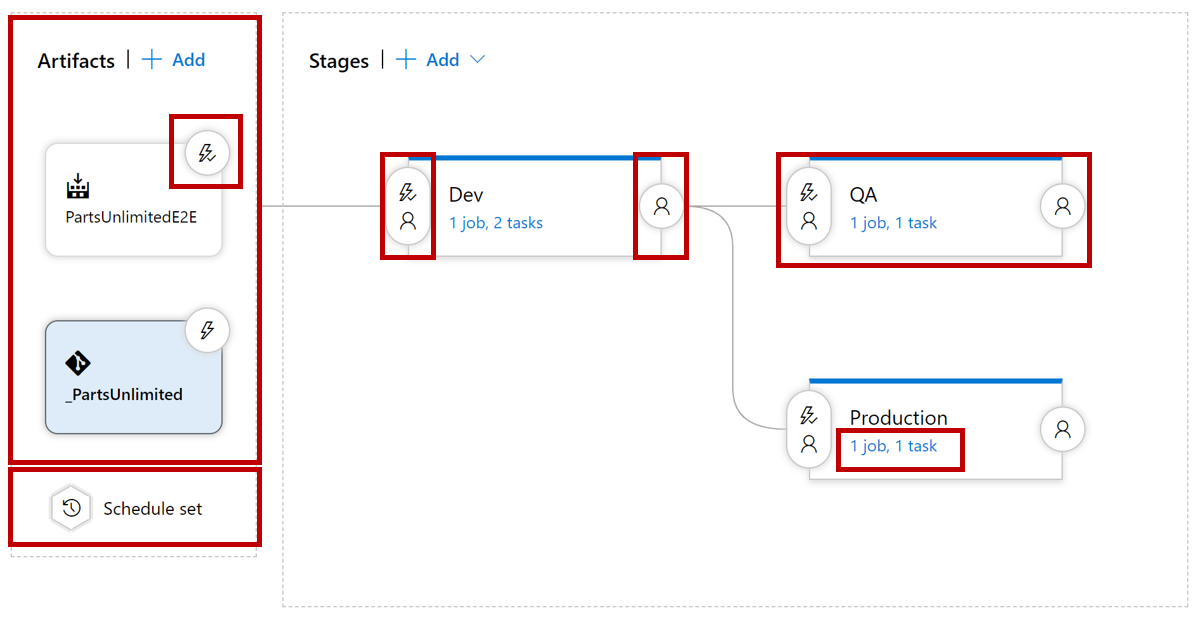
**Next unit: Explore release pipelines**

**Explore release pipelines**

Completed100 XP

* 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](https://learn.microsoft.com/en-us/azure/devops/pipelines/release).

**Next unit: Explore artifact sources**

**Explore artifact sources**

Completed100 XP

* 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.

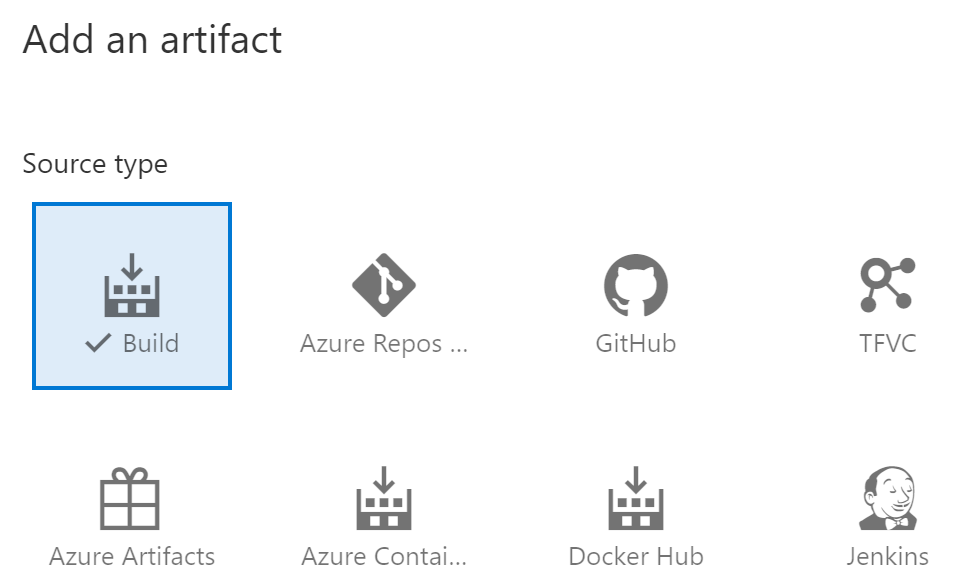
In general, the idea about build and release pipelines and Continuous Delivery is to build once and deploy many times.

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](https://learn.microsoft.com/en-us/azure/devops/artifacts/artifacts-key-concepts). 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.

**Next unit: Choose the appropriate artifact source**

**Choose the appropriate artifact source**

Completed100 XP

* 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](https://semver.org/).

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](https://learn.microsoft.com/en-us/azure/devops/pipelines/release/artifacts).

**Next unit: Examine considerations for deployment to stages**

**Examine considerations for deployment to stages**

Completed100 XP

* 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:

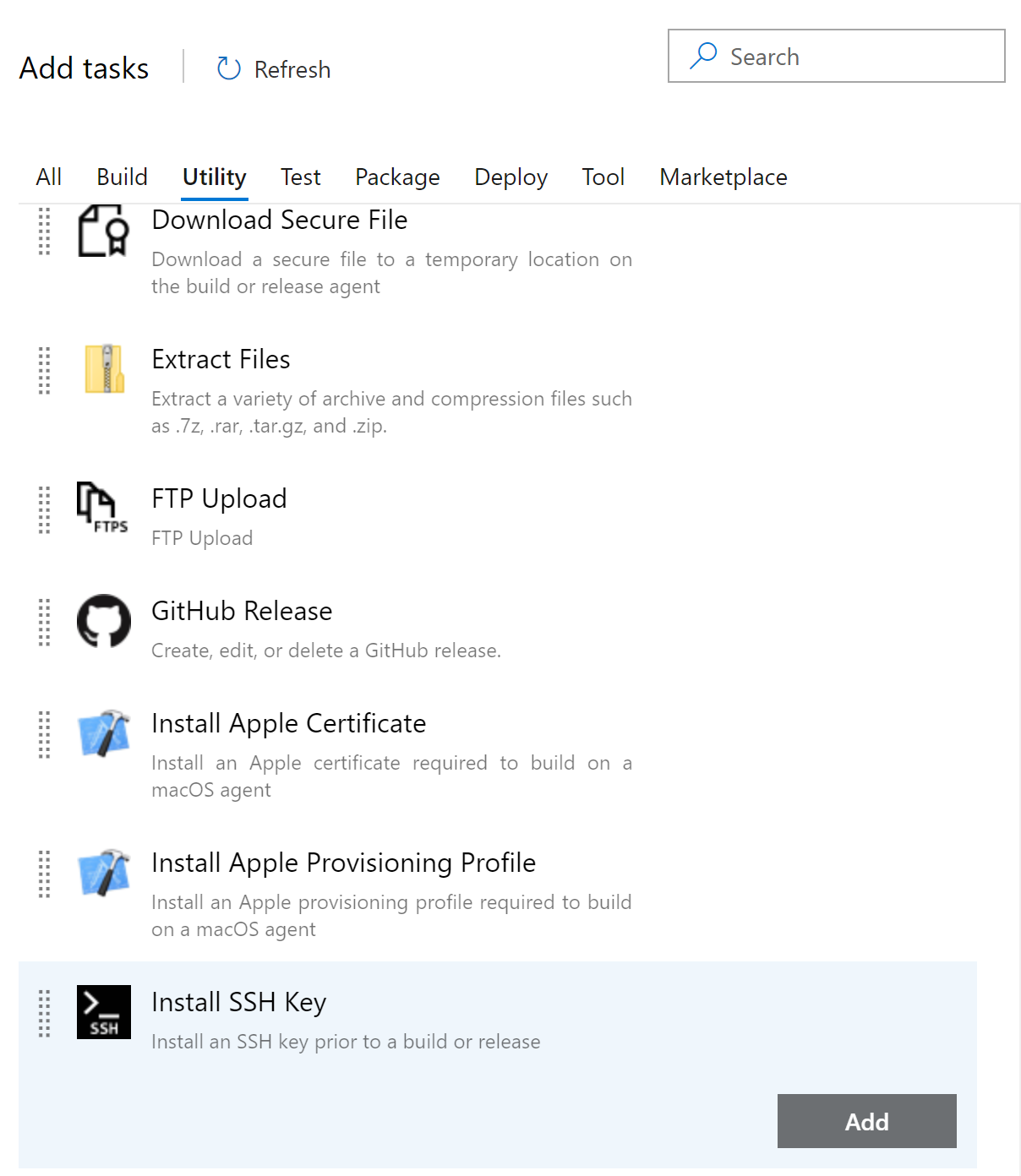
* What is your target environment?
* Does one team use it, or do multiple teams use it?
  + If a single team uses it, you can deploy it frequently. Otherwise, it would be best if you were a bit more careful.
* Who are the users? Do they want a new version multiple times a day?
* How long does it take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

**Next unit: Explore build and release tasks**

**Explore build and release tasks**

Completed100 XP

* 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](https://learn.microsoft.com/en-us/azure/devops/pipelines/process/tasks)
* [Tasks for Azure](https://github.com/microsoft/azure-pipelines-tasks)
* [Atlassian marketplace](https://marketplace.atlassian.com/addons/app/bamboo/trending)
* [Jenkins Plugins](https://plugins.jenkins.io/)

**Next unit: Explore custom build and release tasks**

**Explore custom build and release tasks**

Completed100 XP

* 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](https://learn.microsoft.com/en-us/azure/devops/extend/develop/add-build-task).

**Next unit: Explore release jobs**

**Explore release jobs**

Completed100 XP

* 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.

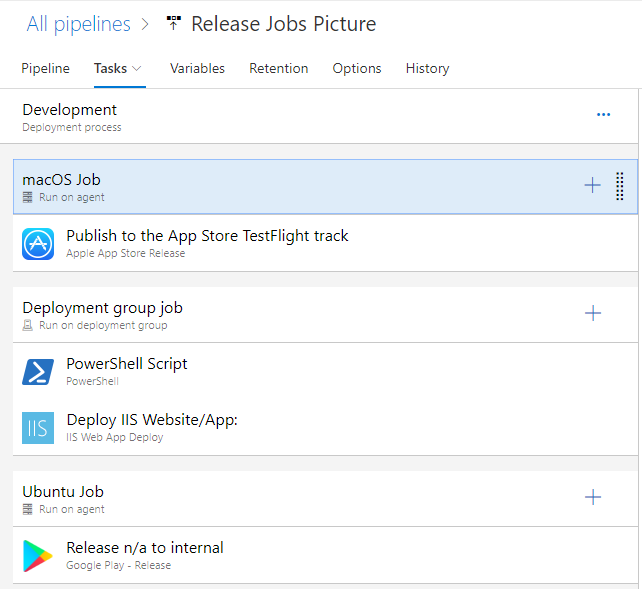
The backend might be deployed from a Windows machine.

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](https://learn.microsoft.com/en-us/azure/devops/pipelines/process/phases).

**Next unit: Understand database deployment task**

**Understand database deployment task**

Completed100 XP

* 3 minutes

Integrating database deployment tasks into CI/CD Azure Pipelines and GitHub Actions workflows is meant to automate provisioning and updating of databases alongside applications which data they host. This integration facilitates seamless coordination between application and database lifecycles, reducing the risk of errors and inconsistencies that may result from manual deployments. Automating database provisioning facilitates faster release cycles, reduces the possibility of human errors, and improves collaboration between development, operations, and database teams.

**Considerations**

Including data components in automated software deployment introduces additional considerations regarding maintaining data integrity and preventing potential data loss. To address and mitigate these risks, take into account the following provisions:

* Data separation: Maintain a clear separation between database schema changes and data manipulation operations. Database schema changes, such as table modifications or schema updates, should be managed separately from data manipulation operations, such as seeding initial data or importing and exporting datasets.
* Data preservation: Implement strategies to preserve existing data during pipeline or workflow redeployment. This may involve backing up critical data before deploying changes or using data migration scripts to transfer data between environments without overwriting existing data.
* Idempotent operations: Ensure that database deployment operations are idempotent, which means that they can be safely rerun multiple times resulting always in the same outcome. Idempotency should eliminate the possibility of creating duplicate schema objects and or records.
* Versioning and rollback: Maintain version control over database schema changes, data migrations, and deployment scripts to track changes and facilitate rollback procedures. Use source control repositories to manage database scripts, apply version tags or labels to database changes, and implement rollback strategies that revert database changes.
* Testing and validation: Perform thorough testing and validation of database changes in a staging or testing environment before deploying to production. Use automated testing frameworks, database unit tests, and integration tests to verify the database deployments and ensure compatibility with existing data.
* Monitoring and alerting: Implement monitoring and alerting mechanisms to detect anomalies, errors, or performance issues during database deployments. Monitor database health metrics, track audit logs, and configure alerts triggered by any unexpected behavior or failures.

The fundamental concepts and activities are similar between Azure Pipelines and GitHub Actions workflows. At a high level, incorporating deployment of databases into Azure Pipelines involves several implementation tasks:

* Creating database deployment scripts to define the database schema, seed data, and apply any additional configurations. The scripts should reside in the source control repository to facilitate change tracking and versioning.
* Creating database connection strings and storing them securely by as secret variables or Azure key vault secrets. This also requires granting pipelines access to the secrets.
* Using Azure DevOps tasks or third-party extensions to execute database deployment scripts as part of the pipeline.
* Configuring the pipeline to deploy any necessary dependencies, such as SQL Server instances, along with required tools and utilities.
* Including database testing tasks in the pipeline to validate database changes and ensure that deployments are successful. This typically involves running database unit tests, integration tests, or data validation checks.
* Implementing rollback and recovery mechanisms in the pipeline to handle deployment failures or unexpected errors. This may include creating database snapshots, backups, or transactional rollback scripts to revert changes in case of issues.

The specific of these implementation tasks vary considerably depending on the target database technology. The following section examines these specifics in the context of deploying Azure SQL services.

**SQL Server and Azure SQL-specific techniques and tools**

When working with SQL Server or Azure SQL Database in CI/CD pipelines, there are numerous tools and techniques you can leverage, including the following ones:

* SQL Server Data Tools (SSDT): a development toolset that enables database developers to build, debug, maintain, and refactor database schemas and objects in Visual Studio. You can use SSDT projects to manage database schema changes and generate deployment scripts for SQL Server or Azure SQL Database.
* SQLPackage.exe: a command-line utility included in SSDT that simplifies automating deployments of SQL Server databases using Data-tier Application Component (DACPAC) files. You can use SQLPackage.exe in CI/CD pipelines to deploy database changes to SQL Server or Azure SQL Database.
* Azure DevOps Tasks: built-in tasks such as Azure SQL Database Deployment and SqlAzureDacpacDeployment that allow you to deploy database changes directly from DACPAC files to Azure SQL Database. These tasks integrate database deployment into CI/CD pipelines without requiring custom scripting.
* Azure Command-Line Interface (CLI): a utility that allows you to interact with Azure resources, including Azure SQL Database, from the command line. You can use Azure CLI commands in CI/CD pipelines to automate database provisioning, configuration, and deployment tasks, such as creating database servers, deploying DACPAC files, and executing SQL scripts.
* SQL Server Management Studio (SSMS): a graphical tool provided by Microsoft for managing SQL Server databases. While primarily used for manual administration tasks, SSMS can also generate deployment scripts and automate database deployments using the Generate Scripts wizard or the Deploy Database to Microsoft Azure SQL Database feature.
* Entity Framework (EF): a programmatic technique that supports the Object-Relational Mapper (ORM) functionality. You can leverage EF-based migration capability to manage database schema changes and automate database migrations in CI/CD pipelines. EF migrations enable you to define changes to your database schema using code-first migrations and apply them automatically during application startup.
* Custom scripts: PowerShell, Bash, or SQL scripts that automate database deployment tasks in CI/CD pipelines. These scripts can use SQLCMD or Azure PowerShell modules to execute SQL scripts, deploy DACPAC files, or interact with Azure SQL Database programmatically.

**Next unit: Configure Pipelines as Code with YAML**

**Configure Pipelines as Code with YAML**

Completed100 XP

* 60 minutes

**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](https://learn.microsoft.com/en-us/azure/devops/server/compatibility).
* **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](https://learn.microsoft.com/en-us/azure/devops/organizations/accounts/create-organization).
* 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 ID tenant associated with the Azure subscription. For details, refer to [List Azure role assignments using the Azure portal](https://learn.microsoft.com/en-us/azure/active-directory/roles/manage-roles-portal) and [View and assign administrator roles in Microsoft Entra ID](https://learn.microsoft.com/en-us/azure/active-directory/roles/manage-roles-portal#view-my-roles).

**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.

[Screenshot of a launch button, which will take you to the lab.](https://go.microsoft.com/fwlink/?linkid=2270114)

**Next unit: Knowledge check**

**Check your knowledge**

Top of Form

**1.**

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

 Correct. There are three different types of jobs you can run. Multi-configuration, Multi-agent, and None (Tasks will run on a single agent).

**2.**

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

Correct. The agent can only execute one job at the same time.

**3.**

**Which of the following choices describes the job type correctly?**

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

Submit answers

Bottom of Form

**Next unit: Summary**

**Summary**

Completed100 XP

* 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 Learn](https://learn.microsoft.com/en-us/azure/devops/pipelines/release).
* [Build and Release Tasks - Azure Pipelines | Microsoft Learn](https://learn.microsoft.com/en-us/azure/devops/pipelines/process/tasks).
* [Jobs in Azure Pipelines and TFS - Azure Pipelines | Microsoft Learn](https://learn.microsoft.com/en-us/azure/devops/pipelines/process/phases).
* [Configure and pay for parallel jobs - Azure DevOps | Microsoft Learn](https://learn.microsoft.com/en-us/azure/devops/pipelines/licensing/concurrent-jobs).

“”” <https://learn.microsoft.com/en-us/training/modules/create-release-pipeline-devops/13-summary#completion> “””