**Azure Pipelines**

Azure Pipelines is a fully-featured continuous integration (CI) and continuous delivery (CD) service that works with your preferred Git provider and can deploy to most major cloud services, including Azure services.

Azure Pipelines enables you to configure and automate your build and delivery tools and environments in YAML (as Infrastructure-as-Code) or through the visual designer in your Azure DevOps web portal at [https://dev.azure.com](https://dev.azure.com/).

**Why you should use CI and CD for your project**

Continuous Integration is all about automating tests and builds for your project; it helps to catch bugs or problems early in the development cycle, which makes them easier and faster to fix. Items known as "artifacts" are produced from CI systems and used by the **Continuous Delivery** release pipelines to drive automatic deployments.

Continuous Delivery is all about automatically deploying and testing code in multiple stages to help drive quality. **Continuous Integration** systems produce the deployable artifacts, including infrastructure and apps, then automated release pipelines consume these artifacts to release new versions and fixes to the target of your choice.

**How to use Azure Pipelines**

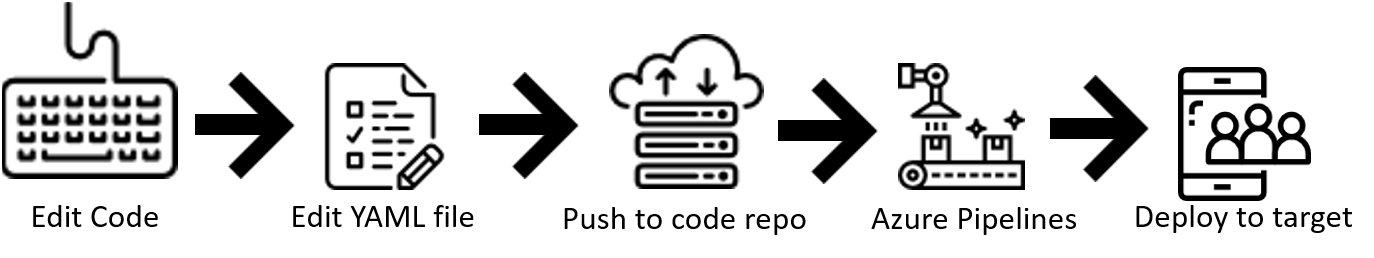
You can either use [YAML](https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started-yaml?view=vsts) to define your pipelines or use the [visual designer](https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started-designer?view=vsts) to do the same.

When you use YAML, you define your pipeline mostly in code (a YAML file) alongside the rest of the code for your app. When you use the visual designer, you define a **build pipeline** to build and test your code, and then to publish artifacts. You also define a **release pipeline** to consume and deploy those artifacts to deployment targets.

**Use Azure Pipelines with YAML**

You can configure your pipelines in a YAML file that exists alongside your code.

1. Configure Azure Pipelines to use your Git repo.
2. Edit your azure-pipelines.yml file to define your build.
3. Push your code to your version control repository, this will kick off the default trigger to build and deploy, and monitor the results.
4. Your code is now updated, built, tested, and packaged and can be deployed to any target.



### Benefits of using YAML

* The pipeline is versioned with your code and follows the same branching structure as your code, so you get validation of your changes through code reviews in pull requests and branch build policies.
* Every branch you use can modify the build policy by modifying the azure-pipelines.yml file.
* If a change to the build process causes a break or results in an unexpected outcome, you can much more easily identify the issue because the change is in version control with the rest of your codebase.

### Use Azure Pipelines in the visual designer

You can create and configure your build and release pipelines in the Azure DevOps Services web portal with the visual designer.

1. Configure Azure Pipelines to use your Git repo.
2. Use the Azure Pipelines visual designer to create and configure your build and release pipelines.
3. Push your code to your version control repository which triggers your pipeline, running any tasks such as building or testing code.
4. The build creates an artifact that is used by the rest of your pipeline, running any tasks such as deploying to staging or production.
5. Your code is now updated, built, tested, and packaged and can be deployed to any target.

**Benefits of using the visual designer**

The visual designer is great for users that are new to the world of CI and CD.

* The visual representation of the pipelines makes it easier to get started.
* The visual designer is located in the same hub as the build results, making it easier to switch back and forth and make changes if needed.

**Agents**

To build your code or deploy your software you need at least one agent.

## **Microsoft-hosted agents**

If your pipelines are in Azure Pipelines, then you've got a convenient option to build and deploy using a **Microsoft-hosted agent**. With Microsoft-hosted agents, maintenance and upgrades are taken care of for you. Each time you run a pipeline, you get a fresh virtual machine. The virtual machine is discarded after one use.

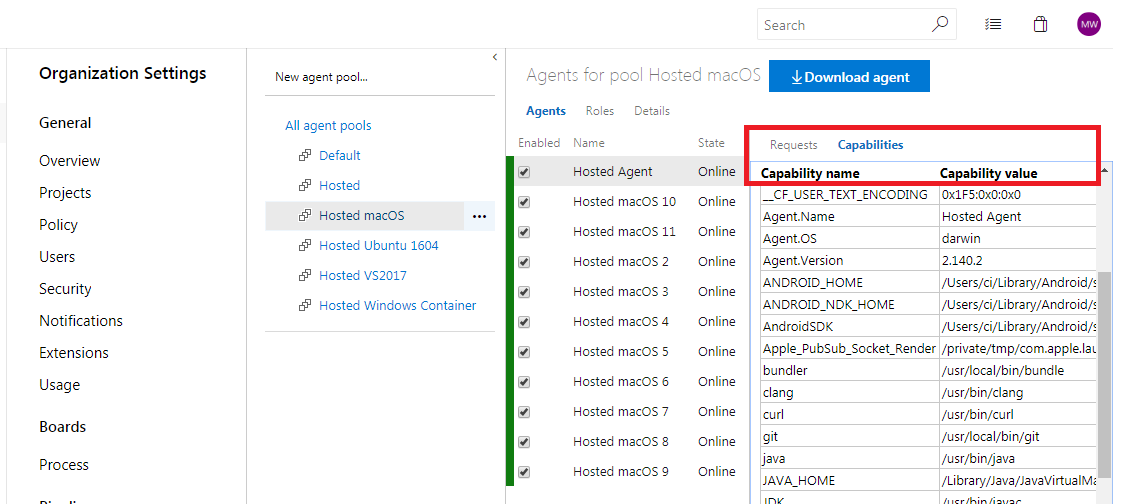
## **Self-hosted agents**

An agent that you set up and manage on your own to run build and deployment jobs is a **self-hosted agent**. Self-hosted agents give you more control to install dependent software needed for your builds and deployments.

## **Capabilities**

Every agent has a set of capabilities that indicate what it can do. Capabilities are name-value pairs that are either automatically discovered by the agent software, in which case they are called **system capabilities**, or those that you define, in which case they are called **user capabilities**.

When you author a build or release pipeline, or when you queue a build or deployment, you specify certain **demands** of the agent. The system sends the job only to agents that have capabilities matching the demands [specified in the pipeline](https://docs.microsoft.com/en-us/azure/devops/pipelines/build/options?view=vsts). As a result, agent capabilities allow you to direct builds and deployments to specific agents.



## **Communication**

### Communication with Azure Pipelines

The agent communicates with Azure Pipelines or TFS to determine which job it needs to run, and to report the logs and job status. This communication is always initiated by the agent.

Here is a common communication pattern between the agent and Azure Pipelines or TFS.

1. The user registers an agent with Azure Pipelines or TFS by adding it to an [agent pool](https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/pools-queues?view=vsts). You need to be an [agent pool administrator](https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/pools-queues?view=vsts#security) to register an agent in that agent pool. Once the registration is complete, the agent downloads a listener OAuth token and uses it to listen to the job queue.
2. Periodically, the agent checks to see if a new job request has been posted for it in the job queue in Azure Pipelines/TFS. When a job is available, the agent downloads the job as well as a job-specific OAuth token. This token is generated by Azure Pipelines/TFS for the scoped identity [specified in the pipeline](https://docs.microsoft.com/en-us/azure/devops/pipelines/build/options?view=vsts). That token is short lived and is used by the agent to access resources (e.g., source code) or modify resources (e.g., upload test results) on Azure Pipelines or TFS within that job.
3. Once the job is completed, the agent discards the job-specific OAuth token and goes back to checking if there is a new job request using the listener OAuth token.

The payload of the messages exchanged between the agent and Azure Pipelines/TFS are secured using asymmetric encryption. Each agent has a public-private key pair, and the public key is exchanged with the server during registration. The server uses the public key to encrypt the payload of the job before sending it to the agent. The agent decrypts the job content using its private key.

### Communication to deploy to target servers

When you use the agent to deploy artifacts to a set of servers, it must have "line of sight" connectivity to those servers. The Microsoft-hosted agent pools, by default, have connectivity to Azure websites and servers running in Azure.

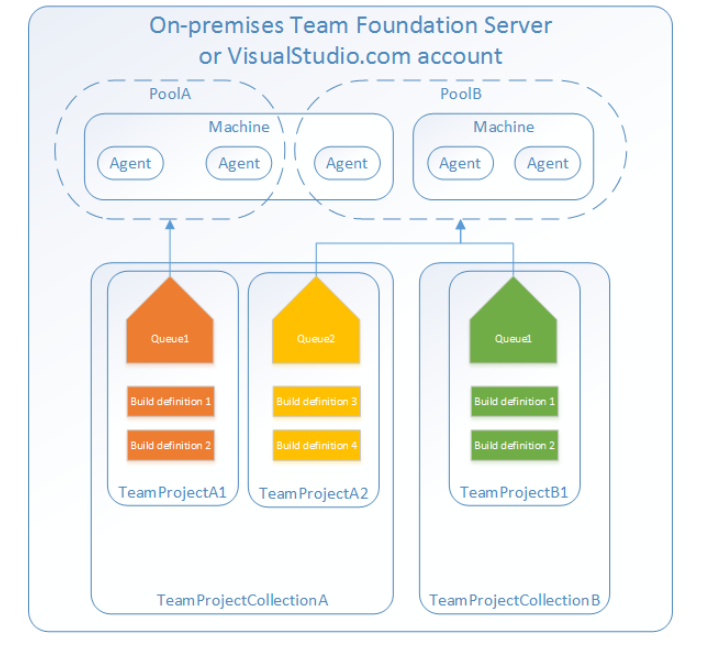
If your on-premises environments do not have connectivity to a Microsoft-hosted agent pool (which is typically the case due to intermediate firewalls), you'll need to manually configure a self-hosted agent on on-premises computer(s). The agents must have connectivity to the target on-premises environments, and access to the Internet to connect to Azure Pipelines or Team Foundation Server, as shown in the following schematic.

# **Agent pools**

Instead of managing each [agent](https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/agents?view=vsts) individually, you organize agents into **agent pools**. An agent pool defines the sharing boundary for all agents in that pool.

A **project agent pool** provides access to an **organization agent pool**. When you create a build or release pipeline, you specify which pool it uses.

To share an agent pool with multiple projects, in each of those projects, you create a project agent pool pointing to an organization agent pool. While multiple pools across projects can use the same organization agent pool, multiple pools within a project cannot use the same organization agent pool. Also, each project agent pool can use only one organization agent pool.



## **Default agent pools**

The following organization agent pools are provided by default:

* **Default** pool: Use it to register [self-hosted agents](https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/agents?view=vsts) that you've set up.
* **Hosted Ubuntu 1604** pool (Azure Pipelinese only): Enables you to build and release on Linux machines without having to configure a self-hosted Linux agent. Agents in this pool do not run in a container, but the Docker tools are available for you to use if you want to run [container jobs](https://docs.microsoft.com/en-us/azure/devops/pipelines/process/container-phases?view=vsts).
* **Hosted Linux** pool (Azure Pipelines only): Enables you to build and release on Linux machines without having to configure a self-hosted Linux agent. The agents in this pool run on an Ubuntu Linux host inside the **[vsts-agent-docker](https://github.com/Microsoft/vsts-agent-docker)**[container](https://github.com/Microsoft/vsts-agent-docker). Note: this pool has been superceded by the Hosted Ubuntu 1604 pool. It will be removed from the service on December 1, 2018. Learn more about [*migrating*](https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/hosted?view=vsts#hosted-linux-preview-pool-deprecation).
* **Hosted macOS** pool (Azure Pipelines only): Enables you to build and release on macOS without having to configure a self-hosted macOS agent. This option affects where your data is stored. [Learn more](https://www.microsoft.com/en-us/trustcenter/privacy/vsts-location)
* **Hosted VS2017** pool (Azure Pipelines only): The **Hosted VS2017** pool is another built-in pool in Azure Pipelines. Machines in this pool have Visual Studio 2017 installed on Windows Server 2016 operating system. For a complete list of software installed on these machines, see [Microsoft-hosted agents](https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/hosted?view=vsts).
* **Hosted** pool (Azure Pipelines only): The **Hosted** pool is the built-in pool that is a collection of Microsoft-hosted agents. For a complete list of software installed on Microsoft-hosted agents, see [Microsoft-hosted agents](https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/hosted?view=vsts).
* **Hosted Windows Container** pool (Azure Pipelines only): Enabled you to build and release inside [Windows containers](https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/). Unless you're building using containers, Windows builds should run in the **Hosted VS2017** or **Hosted** pools.

Each of these Microsoft-hosted organization agent pools is exposed to each project through a corresponding project agent pool.

# **Microsoft-hosted agents**

With Microsoft-hosted agents, maintenance and upgrades are taken care of for you. Each time you run a pipeline, you get a fresh virtual machine. The virtual machine is discarded after one use.

## **Use a Microsoft-hosted agent**

The Microsoft-hosted agent pool provides 5 virtual machine images to choose from:

* Ubuntu 16.04 (ubuntu-16.04)
* Visual Studio 2017 on Windows Server 2016 (vs2017-win2016)
* macOS 10.13 (macOS-10.13)
* Windows Server 1803 (win1803) - for running Windows containers
* Visual Studio 2015 on Windows Server 2012R2 (vs2015-win2012r2)

Microsoft-hosted agents do not offer:

* The ability to log on.
* The ability to [drop artifacts to a UNC file share](https://docs.microsoft.com/en-us/azure/devops/pipelines/build/artifacts?view=vsts#unc-file-share).
* The ability to run [XAML builds](https://msdn.microsoft.com/en-us/library/ms181709%28v=vs.120%29.aspx).
* Potential performance advantages that you might get by using self-hosted agents which might start and run builds faster. [Learn more](https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/agents?view=vsts#private-agent-performance-advantages)

If Microsoft-hosted agents don't meet your needs, then you can [deploy your own self-hosted agents](https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/agents?view=vsts#install).

| **If your development team uses...** | **...then choose this image...** | **...or pool in web designer** |
| --- | --- | --- |
| Docker containers | ubuntu-16.04 or win1803 | Hosted Ubuntu 1604 or Hosted Windows Container |
| Development tools on Ubuntu | ubuntu-16.04 | Hosted Ubuntu 1604 |
| Development tools on macOS | macOS-10.13 (see notes below) | Hosted macOS |
| .NET Core | ubuntu-1604 or vs2017-win2016 | Hosted Ubuntu 1604 or Hosted VS2017 |
| Visual Studio 2017 | vs2017-win2016 | Hosted VS2017 |
| Visual Studio 2015 | vs2015-win2012r2 | Hosted |

# **Deploy an agent on Windows**

## **Prepare permissions**

## **Decide which user you'll use**

As a one-time step, you must register the agent. Someone with permission to [administer the agent queue](https://docs.microsoft.com/en-us/azure/devops/organizations/security/about-security-roles?view=vsts#agent-queue-security-roles) must complete these steps. The agent will not use this person's credentials in everyday operation, but they're required to complete registration

#### **Authenticate with a personal access token (PAT)**

1. Sign in with the user account you plan to use in either your Azure DevOps organization (https://{your-organization}.visualstudio.com) or your Team Foundation Server web portal (https://{your-server}:8080/tfs/).
2. From your home page, open your profile. Go to your security details.
3. Create a personal access token.
4. For the scope select **Agent Pools (read, manage)** and make sure all the other boxes are cleared. If it's a [deployment group](https://docs.microsoft.com/en-us/azure/devops/pipelines/release/deployment-groups/index?view=vsts) agent, for the scope select **Deployment group (read, manage)** and make sure all the other boxes are cleared.
5. Copy the token. You'll use this token when you configure the agent.

### Confirm the user has permission

1. Open a browser and navigate to the **Agent pools** tab for your Azure Pipelines organization or TFS server:
2. Click the pool on the left side of the page and then click **Roles**.
3. If the user account you're going to use is not shown, then get an administrator to add it.

## **Download and configure the agent**

1. Log on to the machine using the account for which you've prepared permissions as explained above.
2. In your web browser, sign in to Azure Pipelines or TFS, and navigate to the **Agent pools** tab:
3. Click **Download agent**.
4. On the **Get agent** dialog box, click **Windows**.
5. Click the **Download** button.
6. Follow the instructions on the page to download the agent.
7. Unpack the agent into the directory of your choice. Then run config.cmd.

### Server URL and authentication

When setup asks for your server URL, for Azure DevOps Services, answer https://dev.azure.com/{your-organization}.

When setup asks for your authentication type, choose **PAT**. Then paste the [PAT token you created](https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/v2-windows?view=vsts#permissions) into the command prompt window.

### Choose interactive or service mode

## **Run the agent**

If you configured the agent to run interactively, to run it:

.\run.cmd

If you configured the agent to run as a service, it starts automatically. You can view and control the agent running status from the services snap-in. Run services.msc and look for "Azure Pipelines Agent (name of your agent)".

# **Run a self-hosted agent behind a web proxy**

To enable the agent to run behind a web proxy, pass --proxyurl, --proxyusername and --proxypassword during agent configuration.

./config.cmd --proxyurl http://127.0.0.1:8888 --proxyusername "myuser" --proxypassword "mypass"

**Builds**

**Artifacts**

Artifacts are the files that you want your build to produce. Artifacts can be nearly anything your team needs to test or deploy your app.

Your CD release pipeline picks up the .ZIP (ASP.NET or Node.js) or .WAR (Java) web deployment file. Your changes are automatically deployed to a test environment in Azure.

## **Examples**

Here are some examples of how to publish artifacts from the **Tasks** tab of your build pipeline.

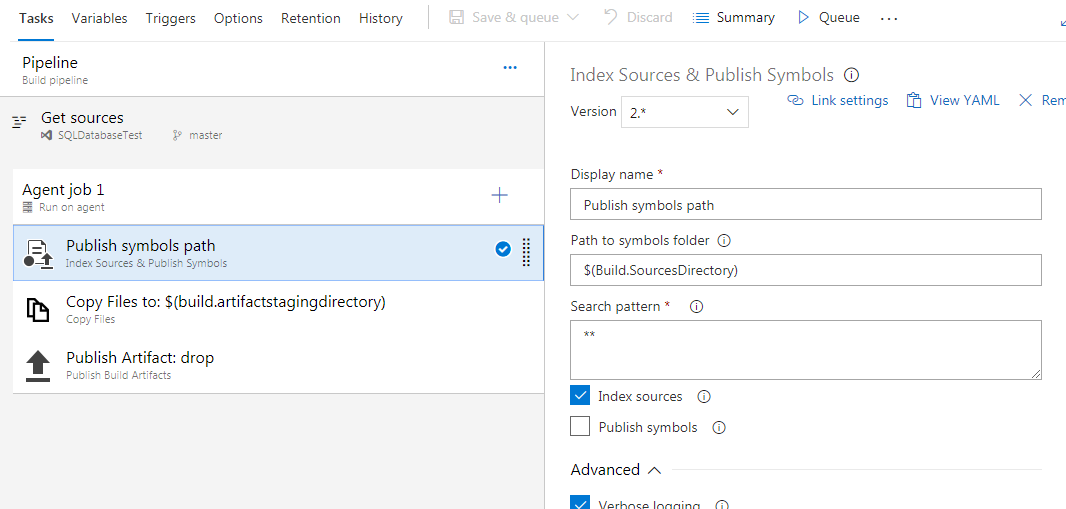
### Publish a README.md file

**Utility: Publish Build Artifacts**

**Publish Symbol Paths**

Path to publish $(Build.SourcesDirectory)/README.md

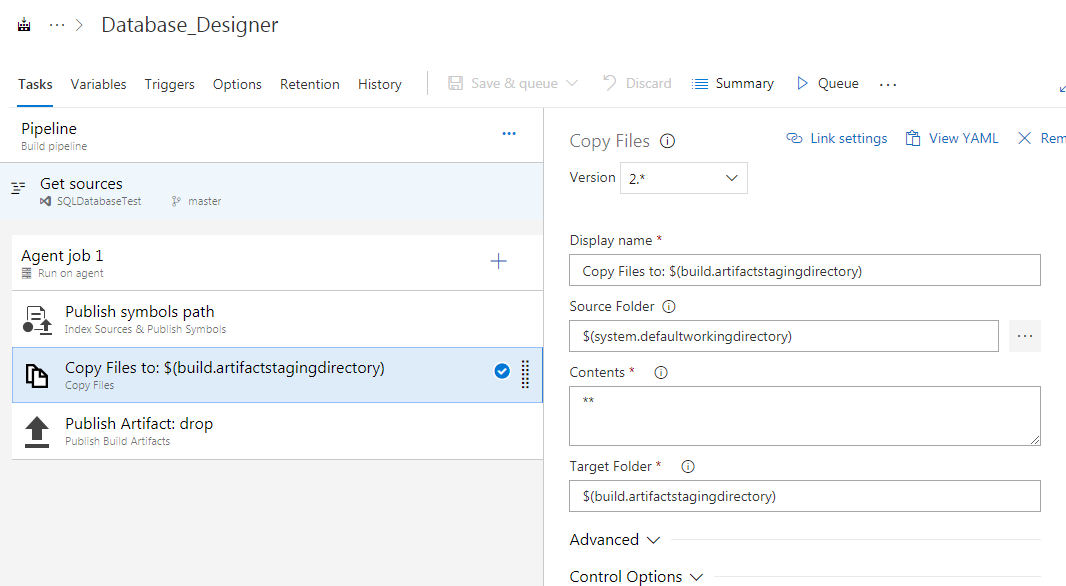
Artifact name drop



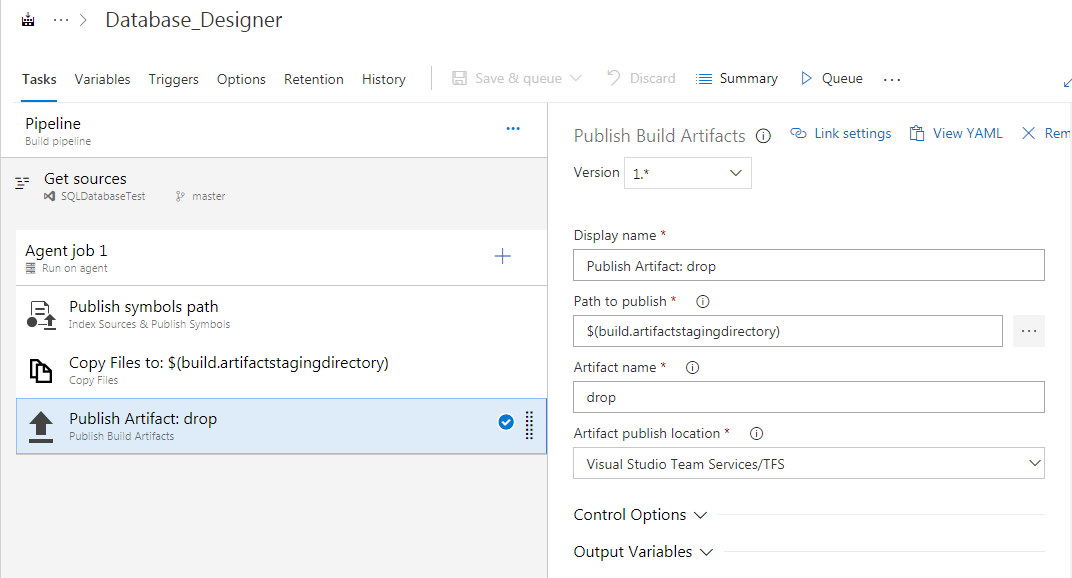
Copy Files to Build Artifacts Staging Directory

Example -

(Example D:\a\1\s\.git\config to D:\a\1\a\.git\config



Publish Artifact



Artifact publish location: Azure Artifacts/TFS (**TFS 2018 RTM and older**: Artifact type: Server)

## **Pipeline - Default agent pool**

Select the [pool](https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/pools-queues?view=vsts) that's attached to the pool that contains the agents you want to run this pipeline.

## **Agent Job - Allow scripts to access the OAuth token**

Select this check box if you want to enable your script to use the build pipeline OAuth token.

# **Build pipeline options**

## **Build number format :**

In YAML, this property is called name. If not specified, your completed build is given a unique integer as its name.

$(TeamProject)\_$(BuildDefinitionName)\_$(SourceBranchName)\_$(Date:yyyyMMdd)$(Rev:.r)

Use **$(Rev:.r)** to ensure that every completed build has a unique name. When a build is completed, if nothing else in the build number has changed, the Rev integer value is incremented by one.

## **Variables**

You can also use user-defined and predefined variables that have a scope of "All" in your build number format. For example, if you've defined My.Variable, you could specify the following build number format:

$(Build.DefinitionName)\_$(Build.DefinitionVersion)\_$(Build.RequestedFor)\_$(Build.BuildId)\_$(My.Variable)

## **Create a work item on failure**

If the build pipeline fails, you can automatically create a work item to track getting the problem fixed. You can specify the work item type.

You can also select if you want to assign the work item to the requestor. For example, if this is a CI build, and a team member checks in some code that breaks the build, then the work item is assigned to that person.

**Additional Fields:** You can set the value of work item fields. For example:

| **Field** | **Value** |
| --- | --- |
| System.Title | Build $(Build.BuildNumber) failed |
| System.Reason | Build failure |

## **Build job authorization scope**

Specify the authorization scope for a build job. Select:

* **Project Collection** if the build needs access to multiple projects.
* **Current Project** if you want to restrict this build to have access only the resources in the current project.

## **Build job timeout in minutes**

Specify the maximum time a build job is allowed to execute on an agent before being canceled by the server. Leave it empty or at zero if you want the job to never be canceled by the server.

## **Build job cancel timeout in minutes**

Specify the maximum time a build job is allowed to respond after the user cancels the build. You can specify a value from 1 to 60 minutes.

Set this value to allow sufficient time for tasks to complete in cases where you've specified to **Run this task** as **Even if a previous task has failed, even if the build was cancelled** or as **Custom conditions** that allow a task to [always run](https://docs.microsoft.com/en-us/azure/devops/pipelines/process/conditions?view=vsts#always) after cancellation.

## **Demands**

Use demands to make sure that the capabilities your build needs are present on the build agents that run it.

Some build tasks won't run unless one or more demands are met by the build agent. For example, the [Visual Studio Build](https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/build/visual-studio-build?view=vsts) task demands that msbuild and visualstudio are installed on the build agent. If your build [includes tasks](https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/index?view=vsts) that have demands, they are listed first.

**Manually entered demands**

You might need to use on-premises build agents with special capabilities. For example, your build pipeline requires SpecialSoftware.

Add the demand to your build pipeline.

| **Name** | **Type** |
| --- | --- |
| SpecialSoftware | exists |

# **Build pipeline source repositories**

### Clean the local repo on the agent

You can perform different forms of cleaning the working directory of your self-hosted agent before a build runs.

In general, for faster performance of your self-hosted agents, don't clean the repo. In this case, to get the best performance, make sure you're also building incrementally. For example, if you're building Visual Studio projects, make sure to clear the **Clean** check box of the Visual Studio Build or MSBuild task.

**Sources**: The build pipeline performs an undo of any changes in $(Build.SourcesDirectory).

**Sources and output directory**: Same operation as **Sources** option above, plus: Deletes and recreates $(Build.BinariesDirectory)

**Sources directory**: Deletes and recreates $(Build.SourcesDirectory). This results in initializing a new, local Git repository for every build.

**All build directories**: Deletes and recreates $(Agent.BuildDirectory).

### Report build status

You've got the option to give your team a view of the build status from your remote source repository.

### Checkout submodules

Select if you want to download files from [submodules](https://git-scm.com/book/en/v2/Git-Tools-Submodules). You can either choose to get the immediate submodules or all submodules nested to any depth of recursion.

### Checkout files from LFS

Select if you want to download files from [large file storage (LFS)](https://docs.microsoft.com/en-us/azure/devops/repos/git/manage-large-files?view=vsts).

### Don't sync sources (TFS 2017 and newer only)

Use this option if you want to skip fetching new commits.

### Shallow fetch

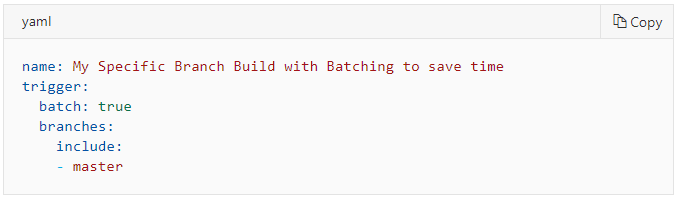
Select if you want to limit how far back in history to download. Effectively this results in git fetch --depth=n

# **Build pipeline triggers**

YAML builds are configured by default with a CI trigger on all branches.



If you set batch to true, when a build is running, the system waits until the build is completed, then queues another build of all changes that have not yet been built.



### Git Filters

If your repository is Git then you can specify the branches where you want to trigger builds.

#### **Path filters in Azure Pipelines and Team Foundation Services (TFS)**

If your Git repo is in Azure Repos or TFS, you can also specify path filters to reduce the set of files that you want to trigger a build.

## **Pull request validation**

For Git-based repos, you can specify branches to include and exclude. Select a branch name from the dropdown and choose "Include" or "Exclude" as appropriate. For included branches, a build will be triggered on each push to a PR targeting that branch.

# **Predefined build variables**

Variables give you a convenient way to get key bits of data into various parts of your build pipeline.

## System.Debug

For more detailed logs to debug pipeline problems, define System.Debug and set it to true.

## Agent variables

Agent.BuildDirectory, Agent.HomeDirectory, Agent.MachineName etc

## Build variables

Build.ArtifactStagingDirectory, Build.BuildNumber

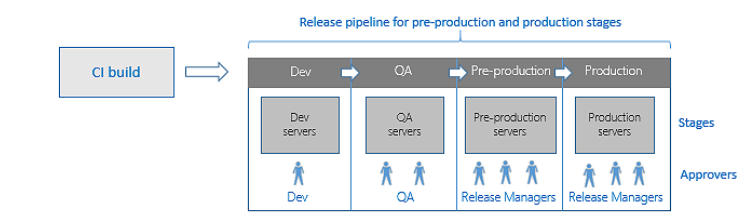
## System variables

System.DefaultWorkingDirectory, System.DefinitionId

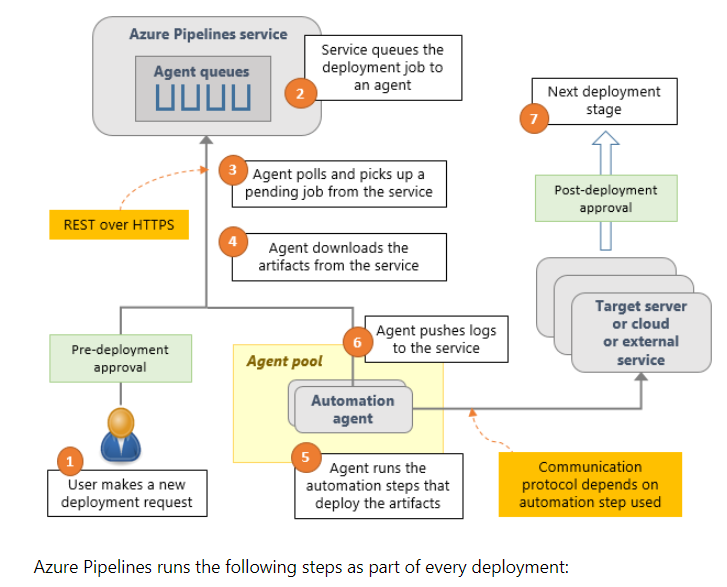
**Releases**

# **What is Azure Pipelines release service?**

**Azure Pipelines releases is** an essential element of DevOps CI/CD that helps your team **continuously deliver** software to your customers at a faster pace and with lower risk. You can **fully automate** the testing and delivery of your software in multiple stages all the way to production, or set up semi-automated processes with **approvals** and **on-demand deployments**.



# **How does Azure Pipelines release service work?**



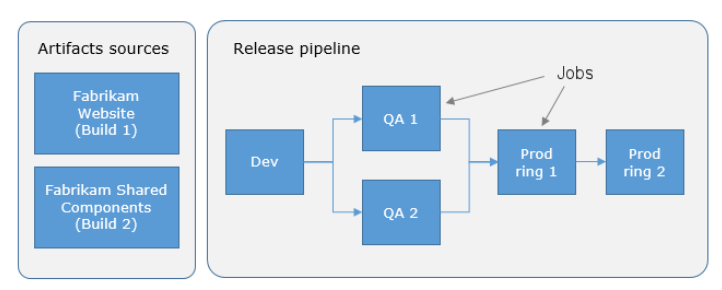
# **Release pipelines and release names**

You start using Azure Pipelines releases by authoring a release pipeline for your application. To author a release pipeline, you must specify the [artifacts](https://docs.microsoft.com/en-us/azure/devops/pipelines/release/artifacts?view=vsts) that make up the application and the **release pipeline**.

An **artifact** is a deployable component of your application. It is typically produced through a Continuous Integration or a build pipeline.

You define the **release pipeline** using [stages](https://docs.microsoft.com/en-us/azure/devops/pipelines/release/environments?view=vsts), and restrict deployments into or out of an stage using [approvals](https://docs.microsoft.com/en-us/azure/devops/pipelines/release/approvals/index?view=vsts). You define the automation in each stage using [jobs](https://docs.microsoft.com/en-us/azure/devops/pipelines/process/phases?view=vsts) and [tasks](https://docs.microsoft.com/en-us/azure/devops/pipelines/process/tasks?view=vsts). You use [variables](https://docs.microsoft.com/en-us/azure/devops/pipelines/release/variables?view=vsts) to generalize your automation and [triggers](https://docs.microsoft.com/en-us/azure/devops/pipelines/release/triggers?view=vsts) to control when the deployments should be kicked off automatically.

An example of a release pipeline that can be modeled through a release pipeline in shown below:



# **Release names**

The first release is named **Release-1**, the next release is **Release-2**, and so on. You can change this naming scheme by editing the release name format mask. In the **Options** tab of a release pipeline, edit the **Release name format** property.

Current Format is : Release-$(rev:r)

# **Release stages, queuing policies, and options**

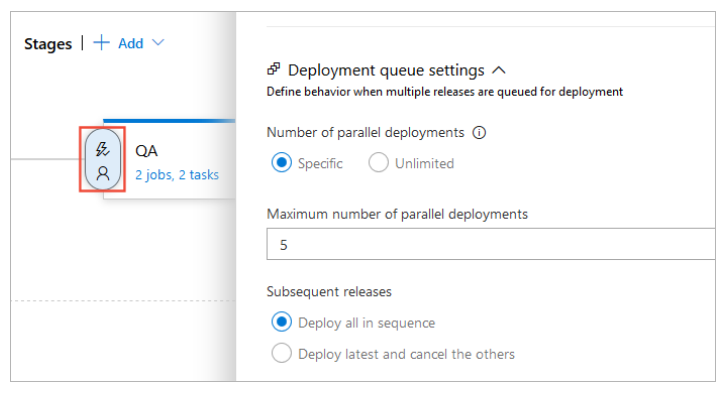
A **stage** is a logical and independent entity that represents where you want to deploy a release generated from a release pipeline.

First, a stage in a release pipeline is a **logical** entity. It can represent any physical or real stage that you need. For example, the deployment in a stage may be to a collection of servers, a cloud, or multiple clouds.

Second, you must be able to deploy to a stage **independently** of other stages in the pipeline. For example, your pipeline might consist of two stages A and B, and Azure Pipelines could deploy Release 2 to A and Release 1 to B.

## **Queuing policies**

In some cases, you may be generating builds more quickly than they can be deployed. Alternatively, you may configure multiple [agents](https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/agents?view=vsts) and, for example, be creating releases from the same release pipeline for deployment of different artifacts. In such cases, it's useful to be able to control how multiple releases are queued into a stage. **Queuing policies** give you that control.



**Example**

Releases **R1**, **R2**, **...**, **R5** of a single release pipeline are created in quick succession due to new builds being produced rapidly. Assume that the first stage in this pipeline is named **QA** and has both pre-deployment and post-deployment approvers defined.

* If you do not specify a limit for the number of parallel deployments, all five approval requests will be sent out as soon as the releases are created. If the approvers grant approval for all of the releases, they will all be deployed to the **QA** stage in parallel.
* If you specify a limit and **Deploy all in sequence**, and the limit has already been reached, the pre-deployment approval for release **R1** will be sent out first. After this approval is completed, the deployment of release **R1** to the **QA** stage begins. Next, a request for post-deployment approval is sent out for release **R1**. It is only after this post-deployment approval is completed that execution of release **R2** begins and its pre-deployment approval is sent out. The process continues like this for all of the releases in turn.
* If you specify a limit and **Deploy latest and cancel the others**, and the limit has already been reached, releases **R2**, **R3**, and **R4** will be skipped, and the pre-deployment approval for **R5** in the **QA** stage will be sent out immediately after the post-deployment approval for release **R1** is completed.

# **Deployment groups**

A deployment group is a logical set of deployment target machines that have agents installed on each one. Deployment groups represent the physical environments; for example, "Dev", "Test", "UAT", and "Production".

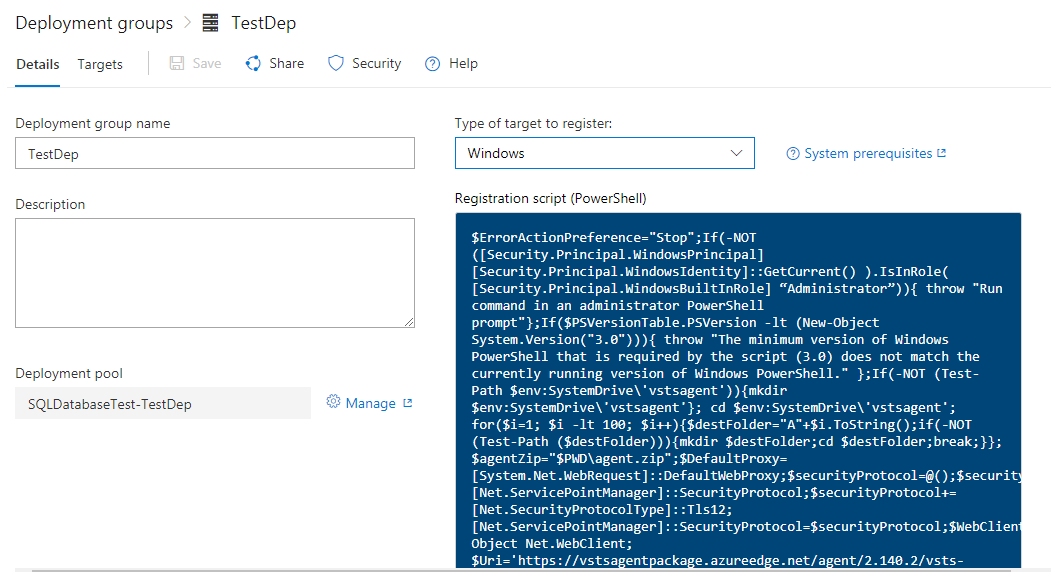
When authoring an Azure Pipelines or TFS Release pipeline, you can specify the deployment targets for a [job](https://docs.microsoft.com/en-us/azure/devops/pipelines/process/phases?view=vsts) using a deployment group. This makes it easy to define [parallel execution](https://docs.microsoft.com/en-us/azure/devops/pipelines/process/phases?view=vsts#parallelexec) of deployment tasks.

## **Create a deployment group**

You define groups on the **Deployment Groups** tab of the **Azure Pipelines** section, and install the agent on each server in the group. After you prepare your target servers, they appear in the **Deployment Groups** tab.

## **Deploy agents to a deployment group**

Every target machine in the deployment group requires the build and release agent to be installed. You can do this using the script that is generated in the **Deployment Groups** tab of **Azure Pipelines**. You can choose the type of agent to suit the target operating system and platform; such as Windows and Linux.



If the target machines are Azure VMs, you can quickly and easily prepare them by by installing the **Azure Pipelines Agent** Azure VM extension on each of the VMs, or by using the **Azure Resource Group Deployment** task in your release pipeline to create a deployment group dynamically.

## **Run the installation script on the target servers**

1. In the **Deployment groups** tab of **Azure Pipelines**, choose **+New** to create a new group.
2. Enter a name for the group, and optionally a description, then choose **Create**.
3. In the **Register machines using command line** section of the next page, select the target machine operating system.
4. Choose **Use a personal access token in the script for authentication**. [Learn more](https://go.microsoft.com/fwlink/?linkid=844181).
5. Choose **Copy the script to clipboard**.
6. Log onto each target machine in turn using the account with the [appropriate permissions](https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/v2-windows?view=vsts#permissions) and:
   * Open an Administrator PowerShell command prompt, paste in the script you copied, then execute it to register the machine with this group.
   * If you get an error when running the script that a secure channel could not be created, execute this command at the Administrator PowerShell prompt:

[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12

* + When prompted to configure tags for the agent, press Y and enter any tags you will use to identify subsets of the machines in the group for partial deployments.

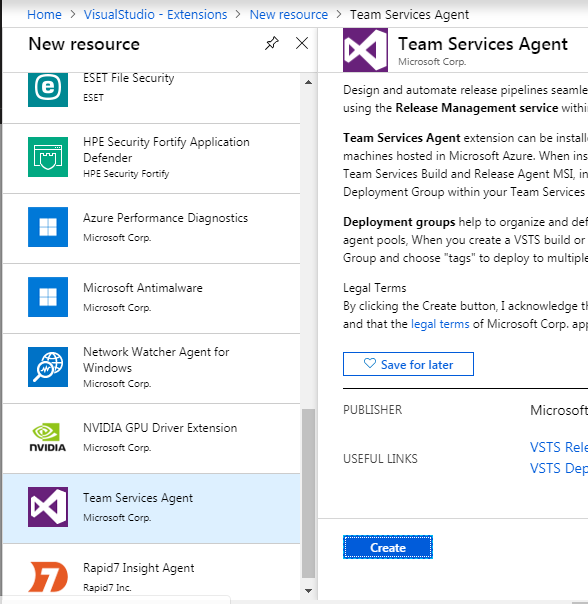
Tags you assign allow you to limit deployment to specific servers when the deployment group is used in a [**Run on machine group** job](https://docs.microsoft.com/en-us/azure/devops/pipelines/process/deployment-group-phases?view=vsts).

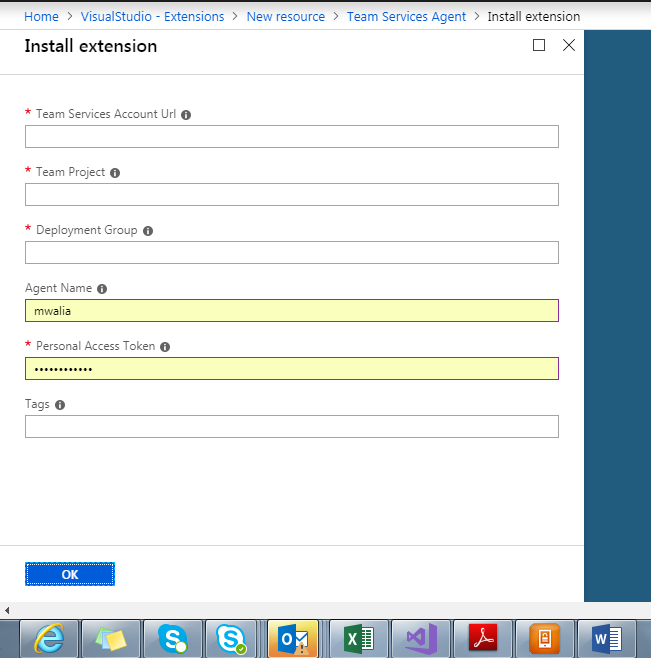
* + When prompted for the user account, press Return to accept the defaults.
  + Wait for the script to finish with the message Service vstsagent.{organization-name}.{computer-name} started successfully.

1. In the **Deployment groups** page of **Azure Pipelines**, open the **Machines** tab and verify that the agents are running. If the tags you configured are not visible, refresh the page.

## **Install the Azure Pipelines Agent Azure VM extension**

1. In the **Deployment groups** tab of **Azure Pipelines**, choose **+New** to create a new group.
2. Enter a name for the group, and optionally a description, then choose **Create**.
3. In the Azure portal, for each VM that will be included in the deployment group open the **Extension** blade, choose **+ Add** to open the **New resource** list, and select **Azure Pipelines Agent**.



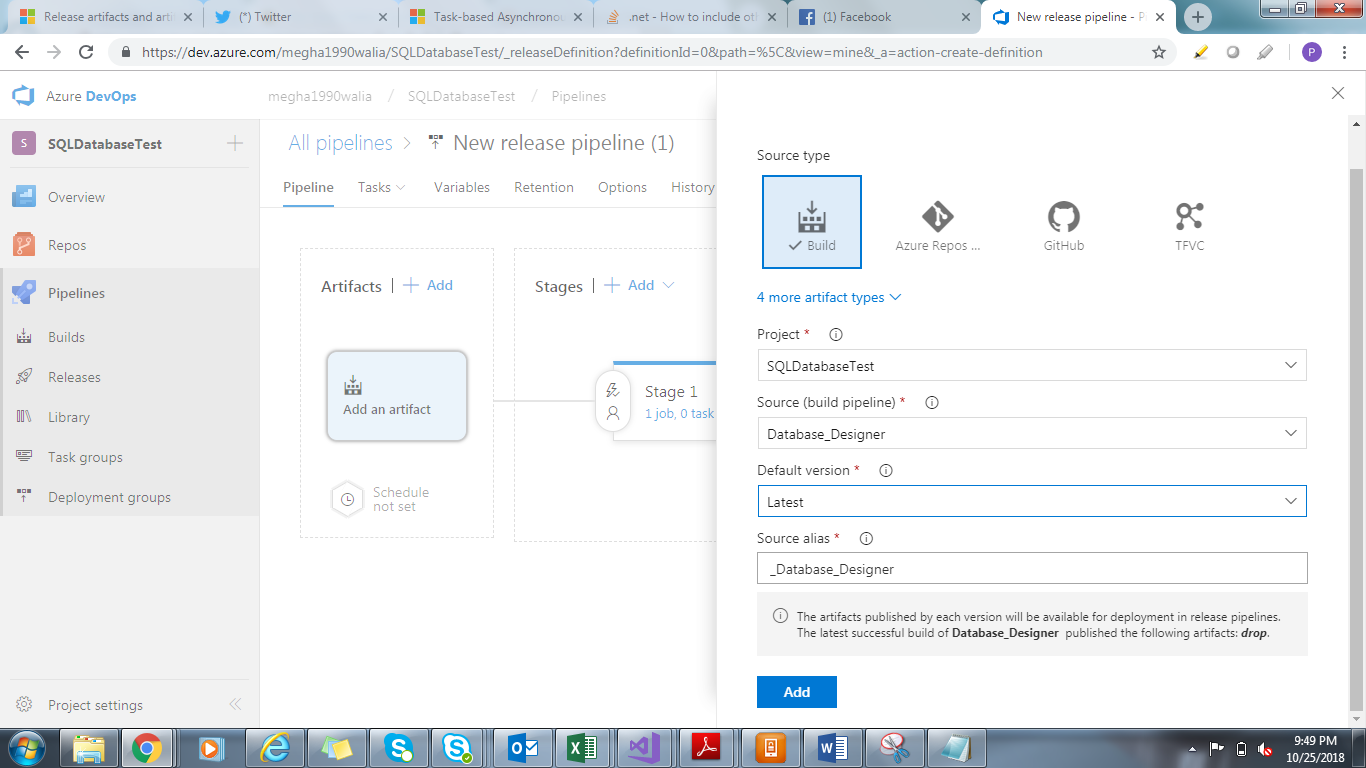


1. In the **Install extension** blade, specify the name of the Azure Pipelines subacription to use. For example, if the URL is https://dev.azure.com/contoso, just specify **contoso**.
2. Specify the project name and the deployment group name.
3. Optionally, specify a name for the agent. If not specified, it uses the VM name appended with -DG.
4. Enter the [Personal Access Token (PAT)](https://go.microsoft.com/fwlink/?linkid=844181) to use for authentication against Azure Pipelines.
5. Optionally, specify a comma-separated list of tags that will be configured on the agent. Tags are not case-sensitive, and each must no more than 256 characters.
6. Choose **OK** to begin installation of the agent on this VM.
7. Add the extension to any other VMs you want to include in this deployment group.

# **Release artifacts and artifact sources**

An **artifact** is a deployable component of your application. Azure Pipelines can deploy artifacts that are produced by a [wide range of artifact sources](https://docs.microsoft.com/en-us/azure/devops/pipelines/release/artifacts?view=vsts#sources), and stored in different types of artifact repositories.

When **authoring a release pipeline**, you link the appropriate **artifact sources** to your release pipeline. For example, you might link an Azure Pipelines build pipeline or a Jenkins project to your release pipeline.



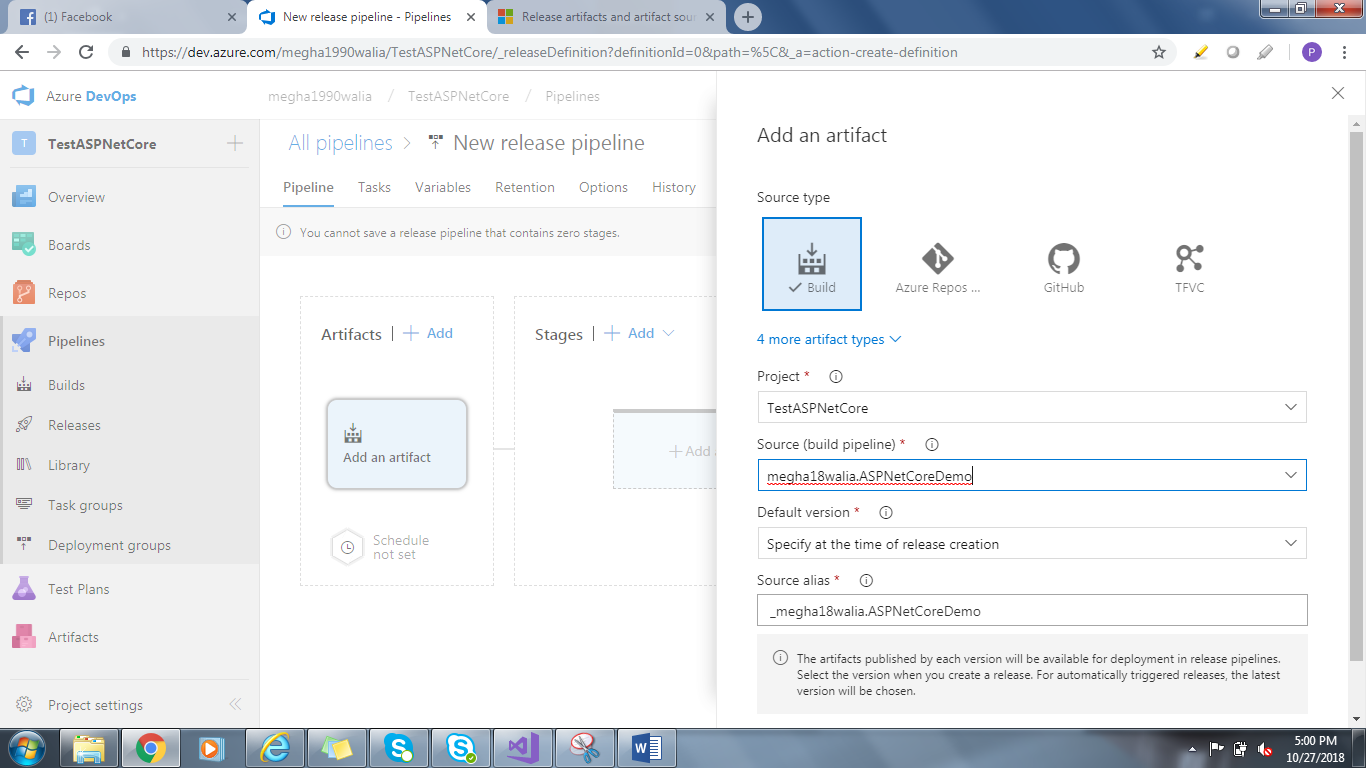
## **Artifact Features**

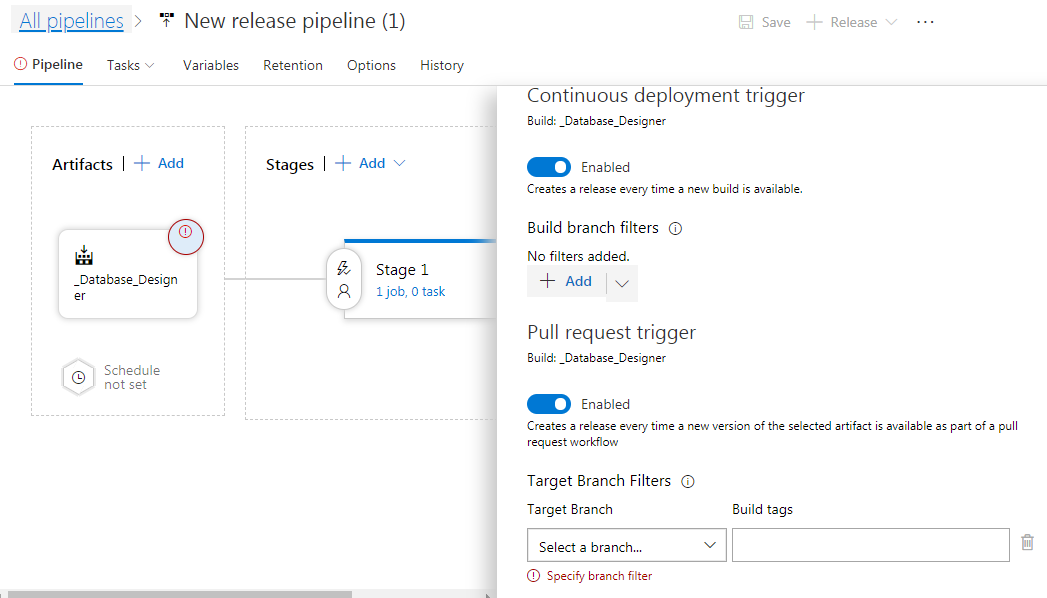
1. **Auto-trigger releases**.
2. **Trigger conditions**.
3. **Artifact versions**.
4. **Artifact variables**. Every artifact that is part of a release has metadata associated with it, exposed to [tasks](https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/index?view=vsts) through [variables](https://docs.microsoft.com/en-us/azure/devops/pipelines/release/artifacts?view=vsts#art-vars).
5. **Artifact download**. Whenever a release is deployed to a stage, by default Azure Pipelines automatically downloads all the artifacts in that release to the [agent](https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/agents?view=vsts) where the deployment job runs.

## **Artifact download**

When you deploy a release to a stage, the versioned artifacts from each of the sources are, by default, downloaded to the automation agent so that tasks running within that stage can deploy these artifacts. The artifacts downloaded to the agent are not deleted when a release is completed. However, when you initiate the next release, the downloaded artifacts are deleted and replaced with the new set of artifacts.

A new unique folder in the agent is created for every release pipeline when you initiate a release, and the artifacts are downloaded into that folder. The $(System.DefaultWorkingDirectory) variable maps to this folder.





## **Artifact source alias**

To ensure the uniqueness of every artifact download, each artifact source linked to a release pipeline is automatically provided with a specific download location known as the source alias. This location can be accessed through the variable:

$(System.DefaultWorkingDirectory)\[source alias]

This uniqueness also ensures that, if you later rename a linked artifact source in its original location (for example, rename a build pipeline in Azure Pipelines or a project in Jenkins), you don't need to edit the task properties because the download location defined in the agent does not change.

**Triggers**

## **Release, branch, and stage triggers**

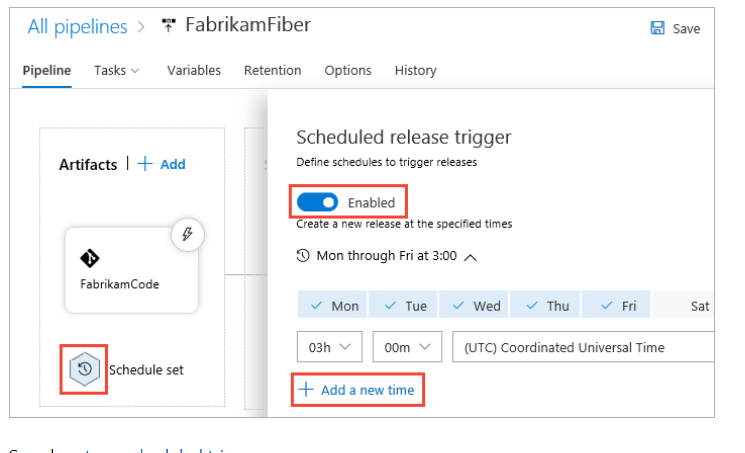
You can configure when releases should be created, and when those releases should be deployed to stages, in your DevOps CI/CD processes.

## **Continuous deployment triggers**

Click on the Continuous Deployment Trigger and then Continuous Deployment can be enabled.

## **Scheduled release triggers**

If you want to create and start a release at specific times, define one or more scheduled release triggers. Choose the schedule icon in the **Artifacts** section of your pipeline and enable scheduled release triggers.

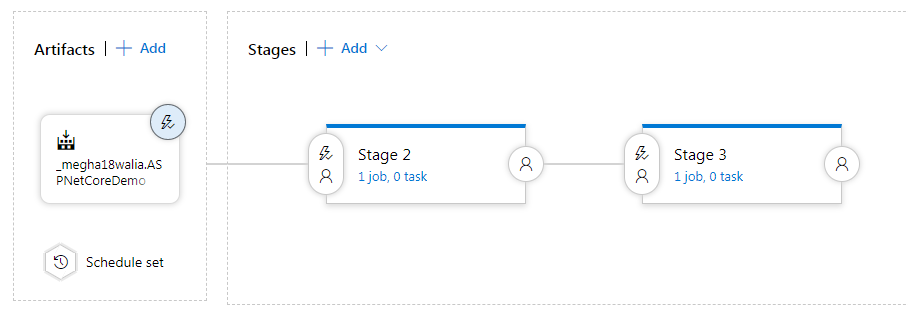


## **Pull request triggers**

You can configure a pull request trigger that will create a new release when a pull request uploads a new version of the artifact. Enable the trigger and add the branches targetted by pull requests that you want to activate this trigger.

## **Stage triggers**

You can choose to have the deployment to each stage triggered automatically when a release is created by a continuous deployment trigger,

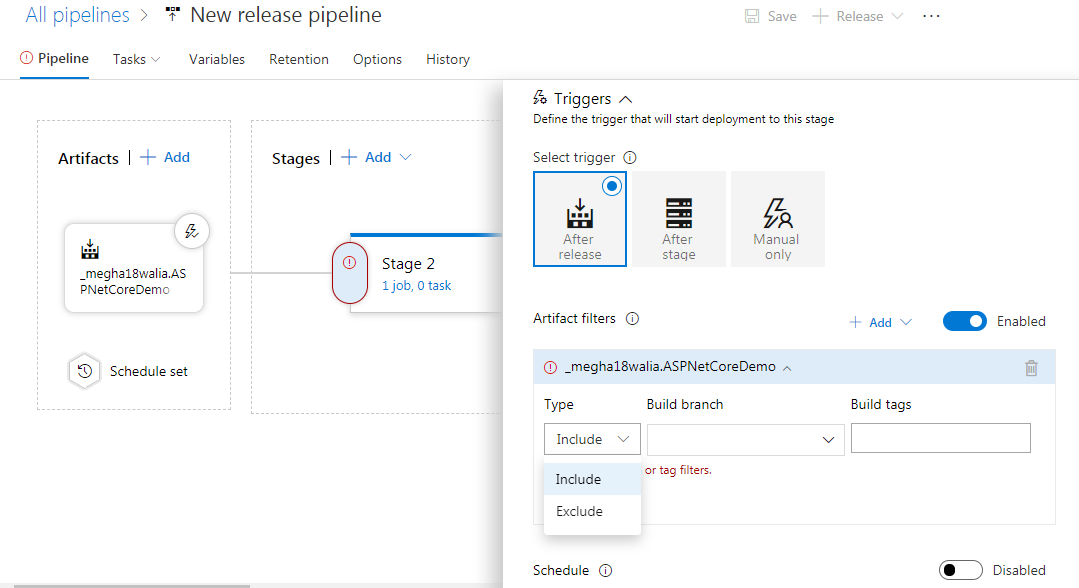


**The result of deploying to a previous stage in the pipeline**.

Use this setting if you want the release to be first deployed and validated in another stage(s) before it is deployed to this stage. For example, you can set up a linear pipeline where a release is deployed first to the **Test** and **QA** stages. Then, if these two deployments succeed, it will be deployed to a **Staging** stage.

**Filters based on the artifacts**.

You can add one or more filters for each artifact linked to the release pipeline, and specify if you want to include or exclude particular branches of the code.



**A predefined schedule**.

Enable Schedule