# Elucidate the concept of pipelines in Azure DevOps

## Introduction

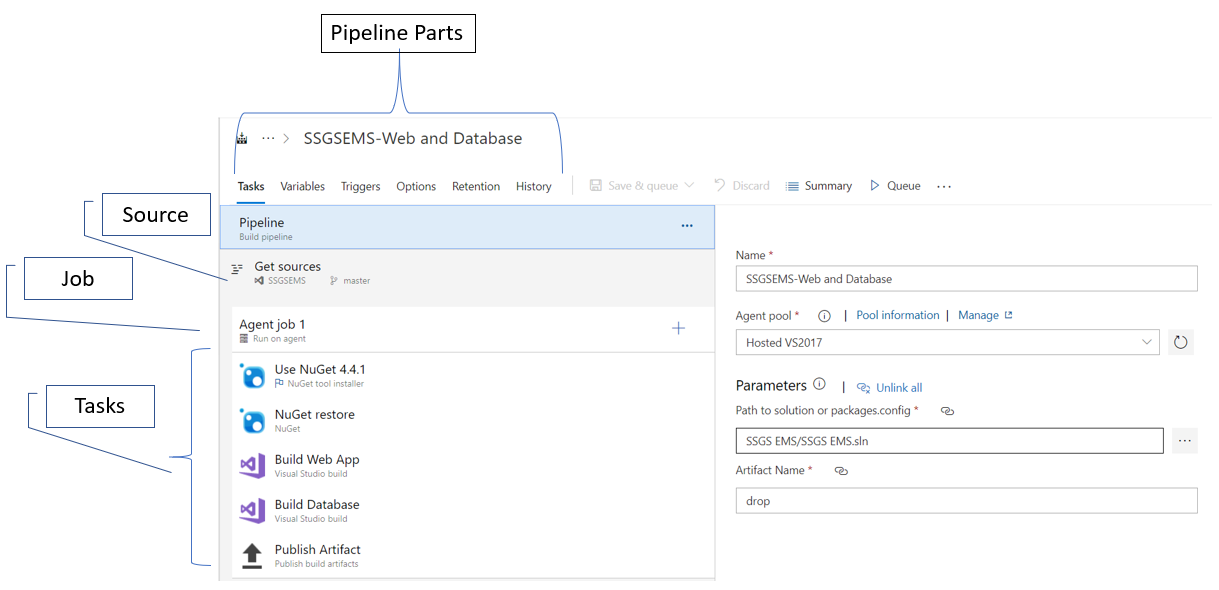
Pipelines is one of the services offered by Azure DevOps. It offers the automated continuous integration (CI) and continuous deployment (CD) features. The pipelines word has also been used loosely for the workflow or ordered set of actions within the same scope of CI / CD. To confound the mater more, Microsoft also used the word in relation to the licensing and pricing of build and release management service, which again, deals with CI / CD. These multiple usage of the same word over the years, has shrouded this concept in a fog of misconceptions. In this article, I am going to clear this fog to make the term understandable to every user of Azure DevOps.

Azure Pipelines is an important service offered by Azure DevOps set of services. It allows us to configure and run the workflows related to server-side build and automated deployment. One of the important features of this service is automation of triggers for running build workflow and deployment workflow. This enables the Continuous Integration – Continuous Deployment (CICD) that is an important part of DevOps. Let’s understand some terms related to Azure Pipelines.

## Terms related to Azure Pipelines

Pipeline (Build Pipeline) – This is a specific use of word ‘pipeline’, under a generic concept of Azure Pipelines, to define various facets of the workflow that builds a deliverable output from code. It defines following:

* The ordered set of actions to be executed – Although core of these actions is to get the code from version control and compile it, it may also optionally involve following:
  + Restore NuGet packages that are used in the project during development.
  + Run any unit tests that may be included in the solution.
  + Analyze and report issues in the code. Third party tools like SonarQUBE may also be used for this.
  + Create the report of the build that is run.
  + Link code and the work items to the build that is run.
  + Create a bug or any other work item on failure of the build.
  + Package the output and upload it for distribution.
* The source of the code – This has to be under some version control like git or Team Foundation Version Control. It can be an external Version Control like Atlassian BitBucket or GitHub or any other third-party installation of Git.
* Agent to be used to run this build – We will discuss the agent selection later in the article.
* The artifact (sharable output) that will be created
* The solution from where the NuGet configuration will be available – NuGet packages are added to the solution and the project as needed by the developers. These NuGet packages are not stored under the Version Control where the code of the project is. The components that are referenced under these NuGet packages are required at the time of compilation. List of such added NuGet packages is kept either in the solution file or packages.config file under the source control. Build engine restores the packages as per this list before the compilation starts.
* Variables to be used – We can use variables that are predefined or those that we define. Examples of some built in (predefined) variables are BuildPlatform (Any CPU, x86 etc.), BuildConfiguration (release or debug), system.teamProject etc. We can create variables to store user names, password etc.
* Automated Triggers that will be used – These triggers include
  + Continuous Integration (CI) – Every time code repository is updated by way of git push or check-in; this build is triggered. To reduce the number of times this trigger should get fired, we can set some filters to it. Team Project, branch or a folder under a branch can be listed to reduce the scope of code that is watched for changes by this trigger.
  + Scheduled – We can set one or multiple schedules at which day and time in the week the build for which the schedule is set will get automatically executed. We can also specify whether such a build should be executed even though there is no change in the code compared to last such scheduled build.
  + Gated Check-in – This trigger is specific to TFVC. It provides a build of the code that you are attempting to check-in, with the remaining files in the project that are of latest version. This ensures buildability of your code even before the check-in is successful. You can take decision about check-in upon the result of this build.
  + Pull Request Related – This is build trigger similar to Gated Check-in trigger but specific to Git. When a new Pull Request is created, it does the build of the changed code with the remaining code of the project from latest commits that are pushed. The result of this build is shown to the reviewer for making decision regarding acceptance or rejection of the build.
* Options like – whether to create a work item if build fails and if yes then which type of work item to be created, update linked work item at the end of the build, format of build id etc.
* Retention policy to define how long the build output will be retained – We can set this in two different terms:
  + Number of days that the build output will be stored. After those number of days, the output of the builds prior to that will be deleted.
  + Minimum number of builds that will be retained. This overrides on the retention policy that is based upon number of days, so that minimum specified number of builds will be retained even if they cross number of days threshold for deletion.

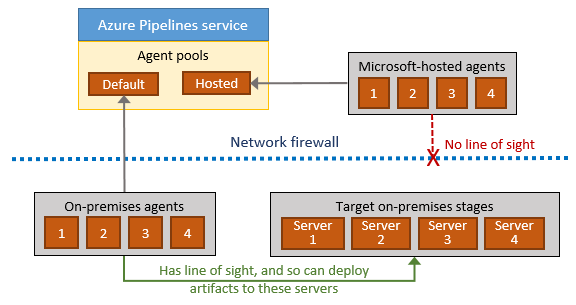


This is the definition that will be used time and again to run the build on the selected agent. We can define a pipeline in the form of a workflow that is designed in the Classic Designer as shown in the figure or we can also define it in a YAML file. Details of YAML for defining builds are available at <https://www.dotnetcurry.com/devops/1522/azure-devops-yaml-ci-cd-pipelines>.

Let us now drill deeper into the concept of agent.

Agent is a software on an identified computer that can run the workflow of build pipeline. It has a capability to also run the deployment pipelines and the test runs too. Agents are of two types based upon the provider.

1. Hosted Agents – Provided by Microsoft for use only during the run of the pipeline. These are on VMs or containers hosted by Microsoft in Azure. We cannot access them directly but view the artifacts that are created at the end of the workflow run. Hosted agents can be used in the builds which do not require any non-standard tools to be used as a part of the workflow because we do not have permission to install any such tools on the hosted agent. In case of deployments to any publicly available environments, that are accessible from the internet, we can use hosted agents for running the deployment workflow.
2. On-Premises Agents – These are the agents that are installed on our computers in our own network. We can install such agents on multiple machines if we want to run the parallel builds or releases. If we want to use any custom tools like JetBrain Resharper or SonarQube as part of the build workflow then we should use such on-premises agent. We should also use it when we want to deploy some software component on the environment which is hosted on our own datacenter and is not accessible from cloud.

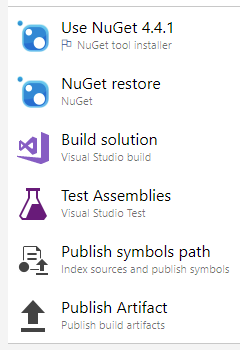


It is important to know difference between the types of agents, for learning the licensing and pricing of pipelines. Licensing and pricing will be covered later in the article.

Source of code to be built has to be compulsorily in a version control. It can be git or TFVC repositories in Azure DevOps itself. It can be git in other services like GitHub or Atlassian Bitbucket or any other. As long as the version control is accessible from cloud, that is acceptable.

Activity / Step / Task – These words denote smallest building blocks of the pipeline. A task encapsulates a script. Activity or Step is either a task or an independent script. Microsoft provides number of tasks that are built-in. Most of the common activities like get source, compile, unit test, copy files, run code analysis, create artifact etc. are covered in those. There are many tasks created by third-party software vendors like SonarCloud, Whitesource, Jenkins, Terraforms etc. are available from Visual Studio Marketplace. We can create and publish our own custom extension for Azure Pipelines to use if what we require is not available in the built-in tasks or from marketplace.

Job – This is a very important concept in pipelines. A job is a set of activities (steps) that are grouped and are to be executed on an agent. These activities are in a specific order. In a pipeline, a job is assigned to an agent. When we start creating a pipeline, we are allowed to select a template from a list of available pre-created templates. Each template consists of a job that has predefined set of tasks in line with the type of selected template. For example, if the template for ASP.NET Web Application is selected for build, then following tasks are automatically added in the created job:

* Use NuGet 4.4.1 – Installs specific version of NuGet package manager on the agent if it does not exist
* NuGet Restore – Restores the NuGet packages as defined in the Solution or packages.config file
* Build Solution – Compile the dependencies and then the web application. It also packages the output in a Web Deploy format and places it in a folder from where it can be published as an artifact.
* Test Assemblies – If any unit tests are defined in the solution then those are executed. Outcome of this test run does also affect the outcome of build.
* Publish symbols – If you have a symbols server defined then the .pdb files that are created as a part of the build are published to that symbols server
* Publish Artifact – the output that is packaged and stored at a specific folder, denoted usually by a variable Build.ArtifactStaginDirectory are published so that it is available for deployment.

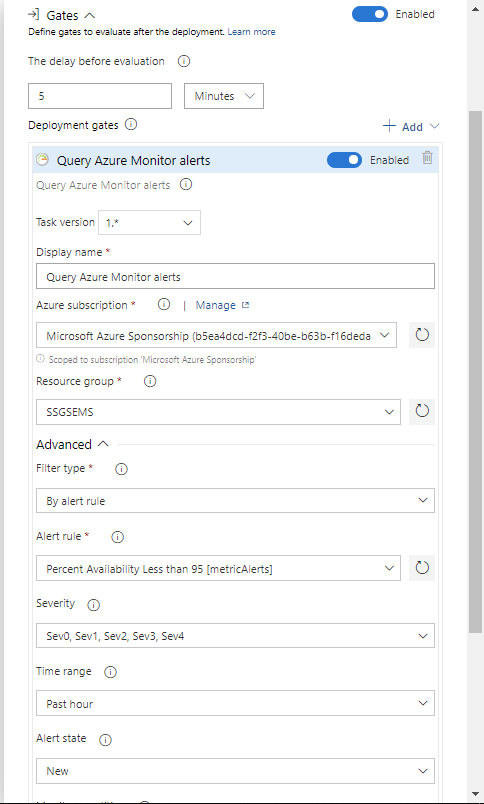
With a default license of Azure DevOps, for an account, only one job can run at a time.

Stage – It is a logical boundary in the pipeline to denote actions to be done for a specific purpose e.g. Testing, Staging, Production etc. A stage may contain one or multiple jobs in it. Each stage has a defined trigger which may depend upon a release getting created or deployment to some other stage getting completed. These are called the Continuous Deployment triggers. More about the concept of Release, later in the article.

Environment – Although a stage and environment are very similar in practice, an environment defines physical resources like machines, containers, application services etc. that are used as the target of a deployment. Usually a stage may target a specific environment but that is not a compulsion.

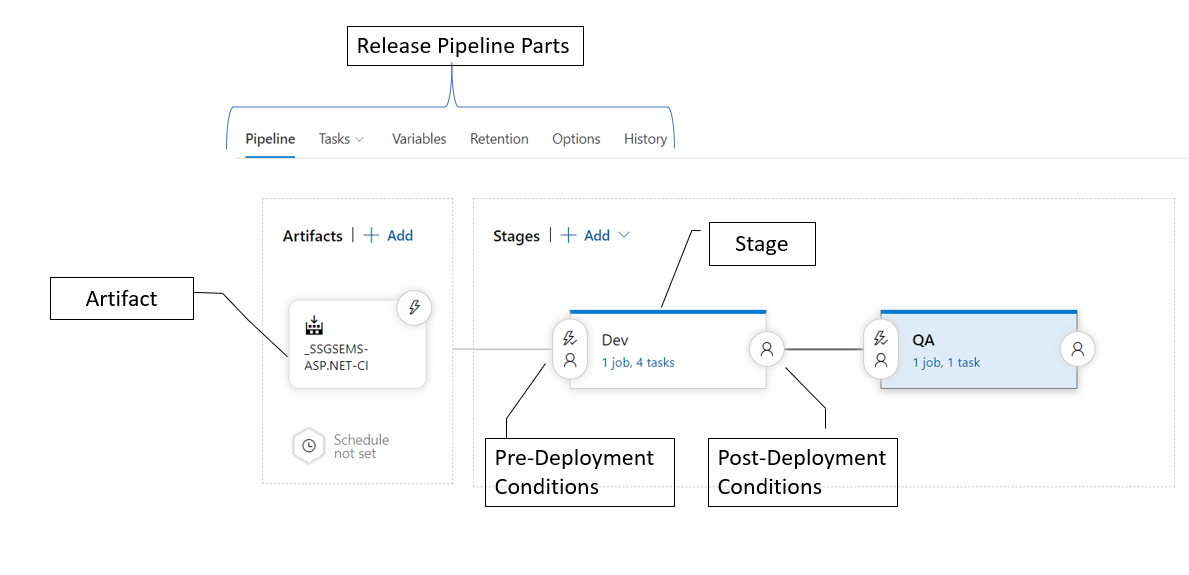
Pipeline (Release) – This defines the process of deployment. It is made up of one or multiple stages. Stages may get run serially (one after the other) or parallelly. Release pipeline also defines following:

* Artifact(s) to be deployed with their version – We may have one or multiple artifacts associated with the release and the deployment process.
* Release trigger – This may be manual or automated (Continuous Deployment) trigger.
* Deployment triggers for each stage – Each stage mat have a manual or automated trigger. Automated trigger may be based upon a release being created or completion of deployment on some other stage.
* Pre-deployment approvers list for each stage – This ensures that after the deployment is triggered, actual deployment starts only after an approval from a defined competent authority is given. For example, when deployment to QA stage is triggered, the QA Lead in the team may not want the deployment to QA environment to be started. It may be because the testing of earlier build is not yet completed. Once that is satisfactorily completed then the QA Lead will approve the deployment to be started after which the actual process of deployment of new build will start. There may be multiple approvers listed and option for any or all with specific order of approvals can be selected.
* Post-deployment approvers list for each stage – This is to ensure that quality of built artifact is satisfactory and it can be deployed to next stage. For example, when a build is satisfactorily smoke tested on Dev environment and when sufficient development on the desired feature is achieved then only the Dev Lead in the team will approve that build to be deployed to next stage, which may be QA stage. Once such approval is given then the deployment to next stage will be triggered.
* Release level variables and stage level variables with their values – Variables allow us to reuse some values in different stages. For example, while doing deployment to a VM, we may need Admin User Name and Admin Password to be used by the task. These values may be needed in multiple different tasks. We can create variables and if necessary their values can be hidden (and kept in Azure Key Vault). If a variable is defined at the Release level then its value can be used in all the stages. If that variable is needed to have different value in each stage then we can define variables at the stage level. Their names can be same, if defined for different stages. To learn more about storing variables in Azure Key Vault, please refer to - <https://www.dotnetcurry.com/devops/1507/azure-key-vault-secrets-pipelines>
* Automated gates for approvals – This is a mechanism to automate the check and approval as part of the post-deployment scenario. Such gates can be
  + Some query in Azure DevOps Boards to check if there is any bug related to the feature under development. If there is no such bug then the gate is opened.
  + Call to an Azure Function or an external web service – It evaluates custom parameters and based upon the results further deployment may be allowed.
  + Query Azure Monitor Alerts – If an alert with a filter condition is raised then further deployment is not allowed. For example, if an alert against a specific rule has been raised and is in New stare, then that closes the gate.



* + A compliance policy against a resource in Azure is not being followed then further deployment is not allowed.

This list of parts of release pipeline is not an exhaustive list but covers main parts of the release pipeline definition.



Run – Execution of a build pipeline with specific version of code or of a release pipeline with an output of a specific run of the build pipeline. Output of a specific run of the build pipeline is also called an Artifact.

Release – This is a concept which links a build with the deployment workflow. It is a process where an artifact of a specific run of the build is linked with the run of the deployment. When a release is created, the artifact version identified with a build id is earmarked for deployment in the run but it is not mandatory to trigger that deployment. As part of continuous deployment (CD) we automate the triggers for creating a release and then starting the deployment process. Although both are separate triggers, if we enable both of them, then when an artifact is ready, it triggers the creation of release and then triggers the deployment process as defined in various stages.

Now that these concepts are clearly defined, let us declutter the word Pipeline as it was used in the licensing and pricing of Azure DevOps. As I mentioned earlier, it is necessary to specify whether the pipeline that you are referring to is going to build an artifact or is going to deploy a built artifact. I prefer to specify it as build pipeline or a release pipeline to disambiguate those two concepts. Let us also understand the licensing and pricing of Azure Pipelines where another use of word pipeline is made.

## Azure Pipeline Pricing

Azure Pipelines pricing is based upon whether you are using hosted agents or on-premises agent. Until some time ago, it used to be defined as per pipeline price. This created another meaning for the word pipeline in the whole context and more confusion prevailed. To reduce the confusion, Microsoft has removed the word pipeline in this context and now the pricing is based upon per parallel jobs that you want to use. When you purchase Azure DevOps, along with Azure Boards, Azure Repos and Azure Artifacts (limited to 2GB) you get Azure Pipelines with following limits:

1. 1 Microsoft-hosted job (on hosted agents) with 1800 minutes per month for CI/CD and 1 self-hosted job with unlimited minutes per month.
2. If you want to use the same job beyond first 1800 minutes, you need to pay approximately USD 40 per month.
3. You can also buy additional hosted jobs to run those in parallel at the price of approximately USD 40 per month per job.
4. If you want to install and use additional parallel agents on premises then you need to pay approximately USD 15 per month per job.

Note: Prices mentioned here are indicative only. For actual prices in your region please access the page <https://azure.microsoft.com/en-us/pricing/details/devops/azure-devops-services/> that pertains to your region.

## Summary

In the context of Azure Pipelines, the word ‘pipeline’ is used in different places. It is used as a workflow for build, multiple workflow branches in release and deployment. It was also used in the licensing and pricing terminology of Azure Pipelines. In this article I have cleared the multiple meaning of word ‘pipeline’ that is used in the context of Azure Pipelines so that we can use it without any ambiguity.