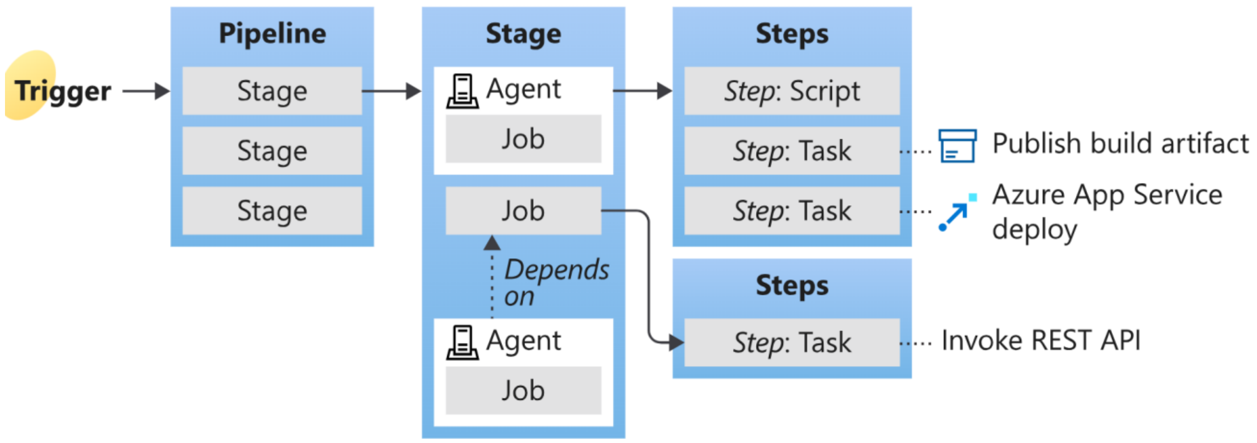
Azure DevOps Pipelines: Tasks, Jobs, Stages

Introduction

When starting in Azure DevOps Pipelines one can immediately become inundated with terminology that may seem foreign or question what pipeline structure will lead to the most flexibility and streamlining of their build/deployment process. This post will focus on the hierarchy of Tasks->Jobs->Stages. My intent will be to cover Environments and Variables in a follow up post and then templating in one after that.



- A [trigger] tells a pipeline to run.
- A [pipeline] is made up of one or more [stages]. A pipeline can deploy to one or more [environments].
- A [stage] is a way of organizing [jobs] in a pipeline and each stage can have one or more jobs.
- Each [job] runs on one [agent]. A job can also be agentless.
- Each [agent] runs a job that contains one or more [steps].
- A [step] can be a [task] or [script] and is the smallest building block of a pipeline.
- A [task] is a prepackaged script that performs an action, such as invoking a REST API or publishing a build artifact.
- An [artifact] is a collection of files or packages published by a [run].

What is a Pipeline?

An Azure DevOps Pipeline is the next generation of what is today referred to as Classic Build and Releases. I have heard them called YAML Pipelines, Multi-Stage Pipelines, and just Pipelines. They have been around now for a few years; however, with the maturity and introduction of new features coming from the product team the general direction seems to be towards adopting pipelines. Perhaps none more telling than the option to "disable classic pipelines" in an Azure Devops Organization.

At the end of the day a pipeline is defined in YAML and used to build and/or deploy your code. Some of the features of YAML pipelines are:

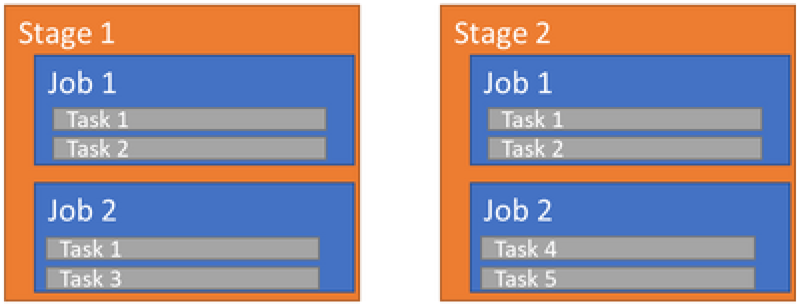
- Defined in Source Control via YAML Files
- Can be templated
- Reference resource in other repositories (even GitHub)
- Easily run jobs in parallel
- Self-Hosted or MS Hosted Agents (Machines) for execution

For more information feel free to check out Microsoft's official documentation.

If coming from a GitHub background this may sound familiar to you as it is comparable to GitHub Actions. Here is a breakdown of terminology between the two. For more feel free to check out this comparison document from GitHub

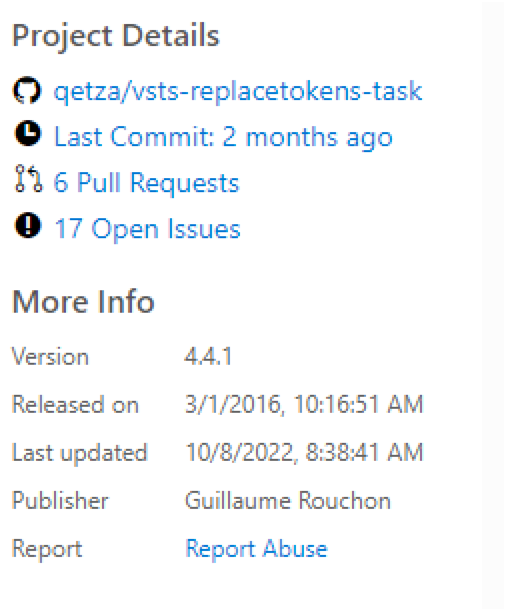
Azure DevOps Pipeline	**GitHub Action**
pipeline	workflow
stages	stages (separate workflow files)
jobs	jobs
tasks	uses

Here is an example of how one environment could utilize stages, jobs, tasks:



Tasks

Sometimes referred to as "steps", tasks are the basic building block to any pipeline. A task is the individual utility/method/execution that is being ran. One cannot have a pipeline without a task. A pipeline needs to run something. For some examples feel free to check out the complete list of Pipeline tasks available from Microsoft. Still can't find what you need? The Azure DevOps Marketplace provide plenty of other tasks which have been created and published from the open-source community. One word of advice if looking to download and install a task from the Marketplace is to ensure the extension is still being actively maintained. An easy way to check this would be click on the project details like the one below:



Project Details

qetza/vsts-replacetokens-task

Last Commit: 2 months ago

6 Pull Requests

17 Open Issues

More Info

Version	4.4.1
Released on	3/1/2016, 10:16:51 AM
Last updated	10/8/2022, 8:38:41 AM
Publisher	Guillaume Rouchon
Report	Report Abuse

Can see here that this particular task, Replace Tokens, was last updated 2 months ago from this writing.

When designing tasks, order matters. Tasks should be constructed so that they are ran in sequential order. In other words, one shouldn't be running an MSBuild task before installing any needed DotnetSDKs.

The majority of the time if the pipeline involves building an application there should be a PublishPipelineArtifact. In general, this task will upload an artifact (code, package, zip file) to the pipeline. This will not only ensure continuity for deployments across environments; however, it will also provide an audit trail and ability to roll back. There is no storage cost associated with this, just double check your artifact retention policies.

Jobs

The next step in the Pipeline hierarchy is the Job. The job is a grouping of tasks. A job may or may not be dependent on other jobs. This is a key concept to understand as it easily allows you to run jobs in parallel to significantly increase your pipeline efficiency.

This is achieved by the job being dispatched to an agent (machine) to run the tasks. Thus, it is important to scope jobs such that they are can be ran in parallel. Going back to my earlier example I may have a job that is building a dotnet code containing the tasks required to package and publish the code and a second job responsible for copying my Infrastructure as Code (IaC) components as pipeline artifacts. Splitting the jobs in this away ensures that agent time is maximized.

Another item to note is the job will download the artifact(s) produced in the build stage of the pipeline. This helps to understand how future jobs retrieve the artifact to be deployed.

A few key considerations when we are talking about jobs:

- Job names must be unique in a given stage....more on that in a minute
- Variables can be loaded to the scope of the job

- Every pipeline must have at least one job
- The agent OS can be defined at the job level
- By default, jobs are run in parallel. Can override by dependsOn[job name]
- Must have at least one job with no dependencies
- Deployment jobs are a key concept
- Azure DevOps Environments are a deployment job property. Next post forthcoming.

Stages

Next on the hierarchal chain is stages. Stages are the level above jobs. A stage is required; however, there is a default if one is not provided. A stage can contain one to many jobs and acts as a logical grouping of related jobs.

An important component to understand is that Azure DevOps, out of the box, provide an easy way to rerun Pipelines at the stage level. This means our stages should be constructed with jobs that make sense to re-run together. Usually, customers want to naturally associate a stage with a customer environment. I'd advise fighting this temptation.

It may make sense at first to do this; however, it can break down and isn't a one size fits all approach. If dealing with the cloud, I usually advise to start with a combination of customer environment/region. Think if my application spans three Azure regions and those are all in the same stage when clicking redeploy stage am I always going to want to redeploy across all three of my regions? Maybe not all the time and if so, we can click redeploy on all the individual stages. An additional scenario could be pipelines that deploy to both on prem and the cloud. Perhaps we'd like to split that as two different stages?

Some other notes on stages:

- Variables can be scoped at the stage level
- Can have dependencies
- Stage names must be unique to the pipeline

Conclusion

The architecture and thought process behind designing and developing tasks, jobs, and stages for your Azure Pipelines are paramount to be successful in a highly mature environment.