Azure DevOps Pipelines:
If Expressions and Conditions

Introduction

At this stage in the series we've talked about tasks, jobs, stages, how to template them, and how to leverage environments in variables. This post will attempt to cover some basics around using if and conditions in your YAML Pipelines.

For the full series check out the series on the Microsoft Health and Life Sciences Blog.

If Expressions

If expressions are simple and easy enough in YAML pipelines, they are a powerful tool. In my experience I have leveraged if expressions to:

- Conditionally load templates
- Dynamically load variables
- Enable same pipeline for CI/CD

The key to unlocking their power is the understanding that an if expression will evaluate at pipeline compilation. This means the pipeline has to leverage known values to apply the logic within. We should not use an if expression when relying on the output of another task/job, the status of another job, or a variable that is updated during pipeline execution.

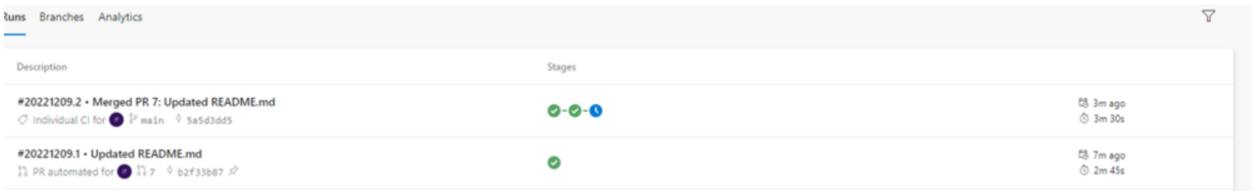
The other side of this, since the statement is evaluated at pipeline compilation time, is that we will not load any unnecessary templates into our pipelines. As opposed to conditions, which will we cover next, templates will not appear in the expanded pipeline YAML file. This leads to a cleaner and more secure experience since only what will be executed will appear in the pipeline logs.

One common scenario I leverage if statements in my YAML pipelines is for CI builds. Typically, I like to leverage the same pipeline for my CI as my CD. This means one pipeline that will only load deployment stages if the source branch is main. Feel free to switch this branch name for any condition your organization may like to use. The if expression for the outlined activity will leverage the built in variable 'Build.SourceBranch'.

```
stages:
- template: stages/terraform_build_stage.yml@templates
  parameters:
    environmentObjects: ${{ parameters.environmentObjects }}
    templateFileName: ${{ parameters.templateFileName }}
    serviceName: ${{ parameters.serviceName }}
- ${{ if eq(variables['Build.SourceBranch'], 'refs/heads/main')}}:
  - template: stages/terraform_apply_stage.yml@templates
```

Here you can see we load a template for the Terraform Build stage every time the pipeline is triggered. However, only if the source branch is main will a deployment occur. This is opposed to a PR build whose source branch will be the branch the PR is based off of, thus only running the CI pieces.

Here is an example illustrating the visual difference between a CI and a CD pipeline execution using the same definition that includes the if expression



Description	Stages	
#20221209.2 • Merged PR 7: Updated README.md Individual CI for main 5a5d3dd5	●-●-●	3m ago 3m 30s
#20221209.1 • Updated README.md PR automated for 7 b2f33b87	●	7m ago 2m 45s

Pipeline example showing the ability to dynamically load stages

Conditions

A condition is actually a key word defined in the schema of any stage, job, or step. It’s not always documented; however, it is available.

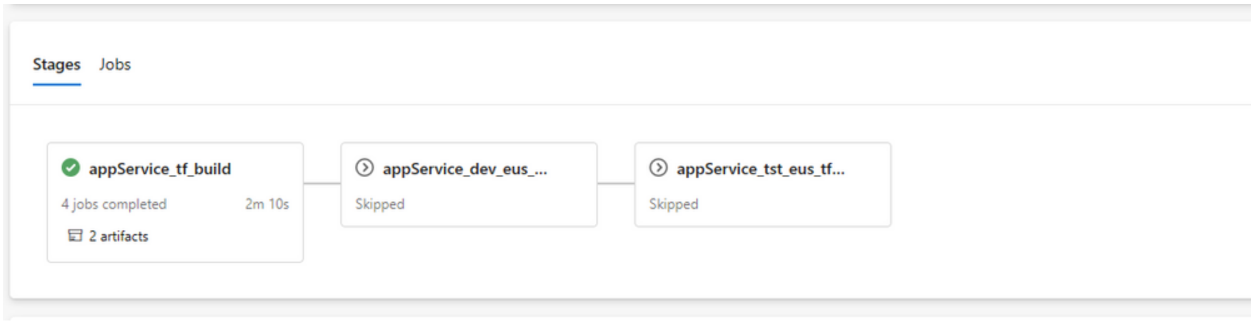
Microsoft defines conditions as:
You can specify the conditions under which each stage, job, or step runs. By default, a job or stage runs if it doesn't depend on any other job or stage, or if *all* of the jobs or stages it depends on have completed and succeeded. This includes not only direct dependencies, but their dependencies as well, computed recursively. By default, a step runs if nothing in its job has failed yet and the step immediately preceding it has finished. You can customize this behavior by forcing a stage, job, or step to run even if a previous dependency fails or by specifying a custom condition.

An important piece to understand is that every stage, job, step has the condition field defaulted to succeeded(). This default condition is configured to be in place and references to the preceding stage/job/task.

This is important to understand as any attempt override this condition, say add a condition to only run a task if the branch has a specific name pattern, will replace the succeeded() default. This would most likely have unintended consequences, so as a good practices if overwriting the condition one should include succeeded() to ensure the previous stage/job/task ran successfully prior to execution.

I have an example of this that was featured in the Microsoft DevOps Community updates on [Dynamically Retain Azure DevOps Pipelines.]() Essentially an optional stage that would run, if the pipeline went to the production stage, and attach a retention to the pipeline for auditing and rollback purposes.

For visual purposes let's look at the CI/CD pipeline and instead of using the if statement, let's use a condition and see what happens:



Pipeline example where condition is not met so stages are skipped.

Since the stages loaded into the pipeline and the condition will be evaluated at pipeline execution, the condition wasn't met, so the stages were skipped. Personally, I find this a bit of a headache, visually, to keep track of. I prefer not loading the stages/jobs/tasks if they won't be needed. Additionally, one can download the pipeline logs and see what all was skipped. Again, this could lead to confusion.

When to use each

Conditions should be leveraged when requiring to evaluate the status of something that has been ran or loaded into the template. Remember that if expressions will dynamically insert templates or variables into a pipeline.
This means if expressions can only evaluate information that is static and available at time of task/job/stage execution. They will not know about which jobs have succeeded, failed or unaware of any variables that may have been created as part of a proceeding task/job/stage.

My own personal pattern is to default leveraging if expressions first. This allows for a cleaner UI and a simpler approach when managing pipelines. This is due to only loading the necessary information into the pipeline vs load everything and evaluate as it goes. The flipside; however, is more complicated pipelines may require additional conditional operators and thus the condition attribute is more appropriate.

Conclusion

Leveraging both if expressions and YAML conditions each have their place and benefit within Azure DevOps. They both can offer the ability to run/load a task/job/stage based on a given criteria. Thus, better utilizing pipelines in an organization's environment.