# YAML(Azure Pipelines) Code Reviews

## Style Guide

Developers should follow the YAML schema reference.

## Code Analysis / Linting

The most popular YAML linter is YAML extension. This extension provides YAML validation, document outlining, auto-completion, hover support and formatter features.

## VS Code Extensions

There is an Azure Pipelines for VS Code extension to add syntax highlighting and autocompletion for Azure Pipelines YAML to VS Code. It also helps you set up continuous build and deployment for Azure WebApps without leaving VS Code.

## YAML in Azure Pipelines Overview

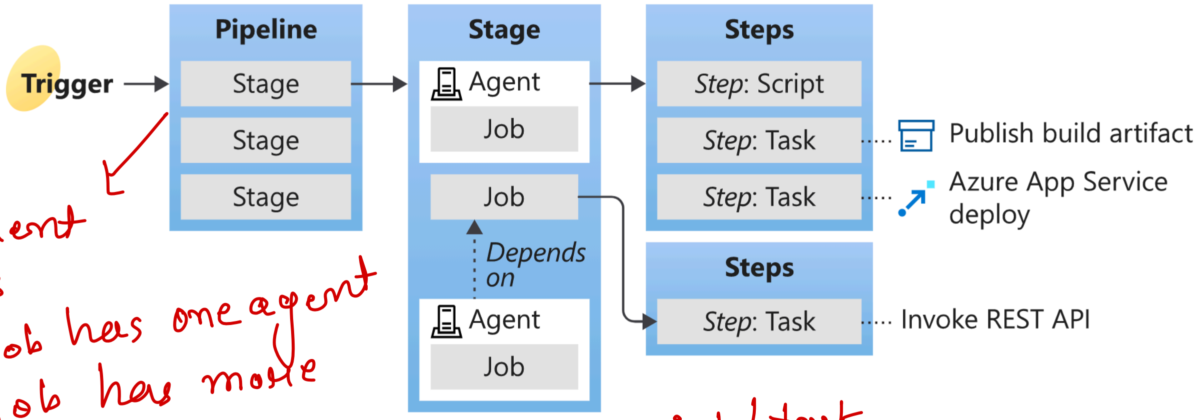
When the pipeline is triggered, before running the pipeline, there are a few phases such as Queue Time, Compile Time and Runtime where variables are interpreted by their runtime expression syntax.

When the pipeline is triggered, all nested YAML files are expanded to run in Azure Pipelines. This checklist contains some tips and tricks for reviewing all nested YAML files.

These documents may be useful when reviewing YAML files:

- Azure Pipelines YAML documentation.
- Pipeline run sequence
- Key concepts for new Azure Pipelines

**Key concepts overview**



- different jobs
- Each job has one agent
- Each job has more than one steps
- step can be script / task

- 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 pre-packaged 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.

## Code Review Checklist

In addition to the Code Review Checklist you should also look for these Azure Pipelines YAML specific code review items.

### Pipeline Structure

□ The steps are well understood and components are easily identifiable. Ensure that there is a proper description `displayName:` for every step in the pipeline.

□ Steps/stages of the pipeline are checked in Azure Pipelines to have more understanding of components.

□ In case you have complex nested YAML files, The pipeline in Azure Pipelines is edited to find trigger root file.

- □ All the template file references are visited to ensure a small change does not cause breaking changes, changing one file may affect multiple pipelines
- □ Long inline scripts in YAML file are moved into script files

## YAML Structure

- □ Re-usable components are split into separate YAML templates.
- □ Variables are separated per environment stored in templates or variable groups.
- □ Variable value changes in `Queue Time`, `Compile Time` and `Runtime` are considered.
- □ Variable syntax values used with `Macro Syntax`, `Template Expression Syntax` and `Runtime Expression Syntax` are considered.
- □ Variables can change during the pipeline, Parameters cannot.
- □ Unused variables/parameters are removed in pipeline.
- □ Does the pipeline meet with stage/job `Conditions` criteria?

## Permission Check & Security

- □ Secret values shouldn't be printed in pipeline, `issecret` is used for printing secrets for debugging
- □ If pipeline is using variable groups in Library, ensure pipeline has access to the variable groups created.
- □ If pipeline has a remote task in other repo/organization, does it have access?
- □ If pipeline is trying to access a secure file, does it have the permission?
- □ If pipeline requires approval for environment deployments, Who is the approver?
- □ Does it need to keep secrets and manage them, did you consider using Azure KeyVault?

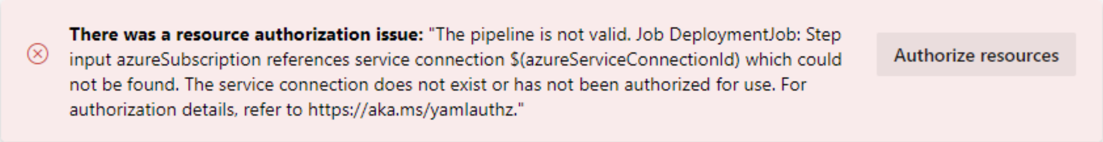
## Troubleshooting Tips

- Consider Variable Syntax with Runtime Expressions in the pipeline. Here is a nice sample to understand Expansion of variables.

- When we assign variable like below it won't set during initialize time, it'll assign during runtime, then we can retrieve some errors based on when template runs.

```
- task: AzureWebApp@1
  displayName: 'Deploy Azure Web App : $(webAppName)'
  inputs:
    azureSubscription: '$(azureServiceConnectionId)'
    appName: '$(webAppName)'
    package: $(Pipeline.Workspace)/drop/Application$(Build.BuildId).zip
    startUpCommand: 'gunicorn --bind=0.0.0.0 --workers=4 app:app'
```

Error:



**There was a resource authorization issue:** "The pipeline is not valid. Job DeploymentJob: Step input azureSubscription references service connection $(azureServiceConnectionId) which could not be found. The service connection does not exist or has not been authorized for use. For authorization details, refer to https://aka.ms/yamlauthz."

After passing these variables as parameter, it loads values properly.

```
  - template: steps-deployment.yaml
    parameters:
      azureServiceConnectionId: ${{ variables.azureServiceConnectionId  }}
      webAppName: ${{ variables.webAppName  }}
```

```
- task: AzureWebApp@1
  displayName: 'Deploy Azure Web App :${{ parameters.webAppName }}'
  inputs:
    azureSubscription: '${{ parameters.azureServiceConnectionId }}'
    appName: '${{ parameters.webAppName }}'
    package: $(Pipeline.Workspace)/drop/Application$(Build.BuildId).zip
    startUpCommand: 'gunicorn --bind=0.0.0.0 --workers=4 app:app'
```

- Use `issecret` for printing secrets for debugging

```
echo "##vso[task.setvariable variable=token;issecret=true]${token}"
```

Last update: August 26, 2024