**Exercise - Wait for approval only when the database schema changes**

Tim and Andy are happy with the pipeline and the manual approval of schema changes. But every time the pipeline runs, it halts to wait for DBA approval even if the database schema hasn't changed. They want the pipeline to halt only when the database schema changes.

In this section, you modify the pipeline to halt only when the database schema changes. To do so, you write a variable to the pipeline that flags the schema change. You then modify the DBAVerificationApply stage to run only when this variable is set.

**Fetch the branch from GitHub**

Here you fetch the schema-changes branch from GitHub. Then you *check out*, or switch to, that branch.

This branch contains the *Space Game* web project. The project includes changes to the website code. The code expects the Profiles table to have a favoriteMap column. The **Profile** page will display this new column's data as **Favorite Galaxy**. The project also includes the Azure Pipelines configuration that you created earlier.

In the *Tailspin.SpaceGame.Database* project, the Profiles table has changed. The favoriteMap column has been added.

1. In Visual Studio Code, open the integrated terminal.
2. Run the following git commands. The commands fetch the schema-changes branch from the Microsoft repository and switch to that branch.

**Bash**

git fetch upstream schema-changes

git checkout -b schema-changes upstream/schema-changes

1. Open the dbo folder in the database project. This folder includes the scripts for each of the tables. Open *Profile.sql* to see the new column, favoriteMap.

**Run the pipeline**

1. Use an empty commit to trigger the pipeline to run. Then push the change to GitHub.

**Bash**

git commit --allow-empty -m "Trigger pipeline for DBA approval"

git push origin schema-changes

1. Watch the pipeline and wait for the manual approval of the database schema.

When the pipeline stops for approval, select the DBAVerificationScript stage. Look at the change script that was created. This time you see an ALTER TABLE SQL statement that adds the new column to the Profiles table.

1. Go back to the pipeline. In the DBAVerificationApply stage, select the **Waiting** button. Select **Review** > **Approve**.
2. Wait for the pipeline to finish deploying to each App Service environment.
3. Before you can try the new website, you need to add data to the new column:
   1. Navigate back to your Azure portal  and select **SQL Databases**.
   2. Select your database, **tailspindatabase**.
   3. Select **Query editor** and sign in.
   4. In your local schema-changes branch, you see *FavoriteMapData.sql*. This SQL script adds a favorite galaxy string to each profile by filling in the favoriteMap column. Copy the contents of this file.
   5. Paste the file contents into **Query 1**. Select **Run** to populate the new table.
   6. Verify that the queries ran successfully.
   7. Select **New Query**.
   8. In **Query 2**, add the following T-SQL code:

**SQL**

SELECT \* FROM dbo.Profiles

Verify that the favoriteMap column is populated with data.

1. Navigate to one of your host names, for example, **tailspin-space-game-web-dev-1234.azurewebsites.net**. Select a player to see the new data on the profile.

The website displaying a profile that shows the player's favorite galaxy.

**Tim:** I think the meeting went well. We managed to get the DBA involved, and they're a tough customer. Score one for DevOps. But I can see a problem here. Every application change triggers this pipeline to run. We won't have schema changes every time. But the pipeline will stop and wait for approval even if the change file contains no changes. How can we fix that problem?

**Mara:** Look at you getting into DevOps! I have an idea. We already have a change script in the pipeline. Maybe we can check that script for keywords that signal a change. Then we can trigger the DBAVerificationApply stage only if the file contains those keywords.

**Andy:** That solution could work. We'll need a pipeline variable to check as a triggering condition.

**Check for database changes**

Here you change the pipeline to skip the DBAVerificationApply stage if the database needs no changes.

**Check for keywords in the generated script**

In this section, you add to the PowerShell script that created the change file. You check to see if a database change is pending. You'll know there's a change if the file contains the words **CREATE**, **ALTER**, or **DROP**.

Don't copy this script yet. Soon you'll replace the entire contents of the *azure-pipelines.yml* file as you did in the previous exercise.

**PowerShell**

$containsWord = $file | %{$\_ -match "CREATE" -or $\_ -match "ALTER" -or $\_ -match "DROP"}

**Pass a pipeline variable change between stages**

If the keyword search returns a match, create a variable in the pipeline variable group. Set its value to True. If the variable is in the pipeline variable group, all stages can see it. Currently, when you set a new variable in a stage, the variable is scoped to only that stage.

Here you use a PowerShell library that was created by Donovan Brown. The library, called VSTeam , uses the REST API for Azure DevOps Services  to access Azure DevOps programmatically.

This script copies all variables out of the variable group. It adds a new variable named schemaChanged. It also updates the variable group with the complete set of variables.

Again, don't copy this script yet. Soon you'll replace the contents of the entire *azure-pipelines.yml* file as you did in the previous exercise.

**PowerShell**

if ($containsWord -contains $true) {

Install-Module VSTeam -Scope CurrentUser -Force

Set-VSTeamAccount –Account $(Acct) -PersonalAccessToken $(PAT)

$methodParameters = @{

ProjectName = "$(System.TeamProject)"

Name = "Release"}

$vg = Get-VSTeamVariableGroup @methodParameters

$vars = @{}

$vg.variables | Get-Member -MemberType \*Property | %{$vars.($\_.Name) = $vg.variables.($\_.Name)}

$varName = "schemaChanged"

$vars.$varName= @{}

$vars.$varName.value = "True"

$vars.$varName.isSecret = $false

$methodParameters = @{

id = $vg.id

ProjectName = "$(System.TeamProject)"

Name = "Release"

Description = ""

Type = "Vsts"

Variables = $vars}

Update-VSTeamVariableGroup @methodParameters}

In the DBAVerificationApply stage, you follow the same pattern. But instead of creating a variable, you remove it. Removing the variable ensures that the DBAVerificationApply stage doesn't run unless the variable is present.

**PowerShell**

Install-Module VSTeam -Scope CurrentUser -Force

Set-VSTeamAccount –Account $(Acct) -PersonalAccessToken $(PAT)

$methodParameters = @{

ProjectName = "$(System.TeamProject)"

Name = "Release"}

$vg = Get-VSTeamVariableGroup @methodParameters

$vars = @{}

$vg.variables | Get-Member -MemberType \*Property | %{$vars.($\_.Name) = $vg.variables.($\_.Name)}

$vars.Remove("schemaChanged")

$methodParameters = @{

id = $vg.id

ProjectName = "$(System.TeamProject)"

Name = "Release"

Description = ""

Type = "Vsts"

Variables = $vars}

Update-VSTeamVariableGroup @methodParameters

**Modify the pipeline to use a condition**

Here you add a condition to the DBAVerificationApply stage. The condition checks that the new variable is set to True. The variable doesn't appear if the database schema hasn't changed. In that case, this stage is skipped.

Again, don't copy this script yet. Soon you'll replace the contents of the entire *azure-pipelines.yml* file as you did in the previous exercise.

**yml**

- stage: DBAVerificationApply

variables:

- group: 'Release'

displayName: 'Apply database schema changes'

dependsOn: DBAVerificationScript

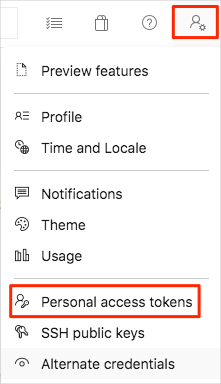
condition: and(succeeded('DBAVerificationScript'), eq(variables['schemaChanged'], True))

**Change the pipeline**

The VSTeam library needs to access your Azure DevOps organization, so it requires authentication. Here you create a personal access token. You add the token to your pipeline variables so that the VSTeam library can authenticate calls to Azure DevOps.

From Azure DevOps:

1. Select your profile from the upper-right corner, then select **Personal access tokens**.



1. Select **+ New Token**.
2. For the name, enter *Database Changes*.
3. Under **Scopes**, select **Full access**.
4. Select **Create**.
5. Copy the token to a safe place.

**Important**

Be sure to save the token copy now. It will never again be shown in plain text.

1. Add two variables to the **Release** variable group.
   1. Navigate to your **Space Game - web - Database** project. Select **Pipelines**.
   2. Under **Pipelines**, select **Library**.
   3. Select **Release**. Add the following variables:

|  |  |
| --- | --- |
|  | |
| Variable name | **Example value** |
| Acct | The name of your organization in Azure DevOps |
| PAT | Your personal access token |

* 1. Select the lock icon next to the value for **PAT** to ensure that this value is encrypted.
  2. Near the top of the page, select **Save** to save your variables to the pipeline.

1. Open the *azure-pipelines.yml* file that you got when you switched to the schema-changes branch.
2. Copy the following new pipeline. Use it to replace the code in the *azure-pipelines.yml* file.

**yml**

trigger:

- '\*'

variables:

buildConfiguration: 'Release'

stages:

- stage: 'Build'

displayName: 'Build the web application'

jobs:

- job: 'Build'

displayName: 'Build job'

pool:

vmImage: 'ubuntu-18.04'

demands:

- npm

variables:

wwwrootDir: 'Tailspin.SpaceGame.Web/wwwroot'

dotnetSdkVersion: '3.1.300'

steps:

- task: UseDotNet@2

displayName: 'Use .NET Core SDK $(dotnetSdkVersion)'

inputs:

version: '$(dotnetSdkVersion)'

- task: Npm@1

displayName: 'Run npm install'

inputs:

verbose: false

- script: './node\_modules/.bin/node-sass $(wwwrootDir) --output $(wwwrootDir)'

displayName: 'Compile Sass assets'

- task: gulp@1

displayName: 'Run gulp tasks'

- script: 'echo "$(Build.DefinitionName), $(Build.BuildId), $(Build.BuildNumber)" > buildinfo.txt'

displayName: 'Write build info'

workingDirectory: $(wwwrootDir)

- task: DotNetCoreCLI@2

displayName: 'Restore project dependencies'

inputs:

command: 'restore'

projects: '\*\*/\*.csproj'

- task: DotNetCoreCLI@2

displayName: 'Build the project - $(buildConfiguration)'

inputs:

command: 'build'

arguments: '--no-restore --configuration $(buildConfiguration)'

projects: '\*\*/\*.csproj'

- task: DotNetCoreCLI@2

displayName: 'Publish the project - $(buildConfiguration)'

inputs:

command: 'publish'

projects: '\*\*/\*.csproj'

publishWebProjects: false

arguments: '--no-build --configuration $(buildConfiguration) --output $(Build.ArtifactStagingDirectory)/$(buildConfiguration)'

zipAfterPublish: true

- publish: '$(Build.ArtifactStagingDirectory)'

artifact: drop

- job: BuildDacpac

pool:

vmImage: 'windows-2019'

steps:

- task: DotNetCoreCLI@2

displayName: 'Restore project dependencies'

inputs:

command: 'restore'

projects: '\*\*/\*.csproj'

- task: VSBuild@1

displayName: 'Build the database project'

inputs:

project: '\*\*/\*.sqlproj'

- task: CopyFiles@2

displayName: 'Copy dacpac file to staging directory'

inputs:

contents: |

Tailspin.SpaceGame.Database/bin/\*\*/\*.dacpac

targetFolder: '$(Build.StagingDirectory)'

- task: PublishBuildArtifacts@1

displayName: 'Publish Artifact'

inputs:

pathToPublish: '$(Build.ArtifactStagingDirectory)'

artifactName: dropDacpac

condition: succeededOrFailed()

- stage: DBAVerificationScript

displayName: 'Script database schema changes'

dependsOn: Build

jobs:

- deployment: DBAVerificationScript

pool:

vmImage: 'windows-2019'

variables:

- group: 'Release'

environment: 'dbaverificationscript'

strategy:

runOnce:

deploy:

steps:

- download: current

artifact: dropDacpac

patterns: '\*\*/\*'

- task: SqlAzureDacpacDeployment@1

displayName: Generate schema change script

inputs:

azureSubscription: 'Resource Manager - Tailspin - Space Game'

authenticationType: 'server'

serverName: '$(servername).database.windows.net'

databaseName: '$(databasename)'

sqlUsername: '$(adminlogin)'

sqlPassword: '$(adminPassword)'

deployType: 'DacpacTask'

deploymentAction: 'Script'

dacpacFile: '$(Pipeline.Workspace)/dropDacpac/Tailspin.SpaceGame.Database/bin/Debug/Tailspin.SpaceGame.Database.dacpac'

ipDetectionMethod: 'AutoDetect'

- task: PowerShell@2

displayName: Show Auto Generated SQL Script

inputs:

targetType: 'inline'

script: |

Write-Host "Auto Generated SQL Update Script:"

Get-Content d:\a\1\s\GeneratedOutputFiles\$(databasename)\_Script.sql | foreach {Write-Output $\_}

- task: PowerShell@2

displayName: Show change script and check for schema changes

inputs:

targetType: 'inline'

script: |

# Print the schema change script

Write-Host "Auto Generated SQL Update Script:"

Get-Content d:\a\1\s\GeneratedOutputFiles\$(databasename)\_Script.sql | foreach {Write-Output $\_}

# Check for schema changes

$file = Get-Content "d:\a\1\s\GeneratedOutputFiles\$(databasename)\_Script.sql"

$containsWord = $file | %{$\_ -match "CREATE" -or $\_ -match "ALTER" -or $\_ -match "DROP"}

if ($containsWord -contains $true) {

Install-Module VSTeam -Scope CurrentUser -Force

Set-VSTeamAccount –Account $(Acct) -PersonalAccessToken $(PAT)

$methodParameters = @{

ProjectName = "$(System.TeamProject)"

Name = "Release"}

$vg = Get-VSTeamVariableGroup @methodParameters

$vars = @{}

$vg.variables | Get-Member -MemberType \*Property | %{$vars.($\_.Name) = $vg.variables.($\_.Name)}

$varName = "schemaChanged"

$vars.$varName= @{}

$vars.$varName.value = "True"

$vars.$varName.isSecret = $false

$methodParameters = @{

id = $vg.id

ProjectName = "$(System.TeamProject)"

Name = "Release"

Description = ""

Type = "Vsts"

Variables = $vars}

Update-VSTeamVariableGroup @methodParameters}

- stage: DBAVerificationApply

variables:

- group: 'Release'

displayName: 'Apply database schema changes'

dependsOn: DBAVerificationScript

condition: and(succeeded('DBAVerificationScript'), eq(variables['schemaChanged'], True))

jobs:

- deployment: DBAVerificationApply

pool:

vmImage: 'windows-2019'

variables:

- group: 'Release'

environment: 'dbaverificationapply'

strategy:

runOnce:

deploy:

steps:

- download: current

artifact: dropDacpac

patterns: '\*\*/\*'

- task: SqlAzureDacpacDeployment@1

displayName: 'Deploy SQL schema'

inputs:

azureSubscription: 'Resource Manager - Tailspin - Space Game'

authenticationType: 'server'

serverName: '$(servername).database.windows.net'

databaseName: '$(databasename)'

sqlUsername: '$(adminlogin)'

sqlPassword: '$(adminPassword)'

deployType: 'DacpacTask'

deploymentAction: 'Publish'

dacpacFile: '$(Pipeline.Workspace)/dropDacpac/Tailspin.SpaceGame.Database/bin/Debug/Tailspin.SpaceGame.Database.dacpac'

ipDetectionMethod: 'AutoDetect'

- task: PowerShell@2

displayName: 'Remove "schemaChanged" pipeline variable'

inputs:

targetType: 'inline'

script: |

Install-Module VSTeam -Scope CurrentUser -Force

Set-VSTeamAccount –Account $(Acct) -PersonalAccessToken $(PAT)

$methodParameters = @{

ProjectName = "$(System.TeamProject)"

Name = "Release"}

$vg = Get-VSTeamVariableGroup @methodParameters

$vars = @{}

$vg.variables | Get-Member -MemberType \*Property | %{$vars.($\_.Name) = $vg.variables.($\_.Name)}

$vars.Remove("schemaChanged")

$methodParameters = @{

id = $vg.id

ProjectName = "$(System.TeamProject)"

Name = "Release"

Description = ""

Type = "Vsts"

Variables = $vars}

Update-VSTeamVariableGroup @methodParameters

- stage: 'Dev'

displayName: 'Deploy to the dev environment'

dependsOn:

- DBAVerificationScript

- DBAVerificationApply

condition: |

and

(

succeeded('DBAVerificationScript'),

in(dependencies.DBAVerificationApply.result, 'Succeeded', 'Skipped')

)

jobs:

- deployment: Deploy

pool:

vmImage: 'ubuntu-18.04'

environment: dev

variables:

- group: Release

strategy:

runOnce:

deploy:

steps:

- download: current

artifact: drop

- task: AzureWebApp@1

displayName: 'Azure App Service Deploy: website'

inputs:

azureSubscription: 'Resource Manager - Tailspin - Space Game'

appName: '$(WebAppNameDev)'

package: '$(Pipeline.Workspace)/drop/$(buildConfiguration)/\*.zip'

- stage: 'Test'

displayName: 'Deploy to the test environment'

dependsOn: Dev

condition: succeeded('Dev')

jobs:

- deployment: Deploy

pool:

vmImage: 'ubuntu-18.04'

environment: test

variables:

- group: 'Release'

strategy:

runOnce:

deploy:

steps:

- download: current

artifact: drop

- task: AzureWebApp@1

displayName: 'Azure App Service Deploy: website'

inputs:

azureSubscription: 'Resource Manager - Tailspin - Space Game'

appName: '$(WebAppNameTest)'

package: '$(Pipeline.Workspace)/drop/$(buildConfiguration)/\*.zip'

- stage: 'Staging'

displayName: 'Deploy to the staging environment'

dependsOn: Test

condition: succeeded('Test')

jobs:

- deployment: Deploy

pool:

vmImage: 'ubuntu-18.04'

environment: staging

variables:

- group: 'Release'

strategy:

runOnce:

deploy:

steps:

- download: current

artifact: drop

- task: AzureWebApp@1

displayName: 'Azure App Service Deploy: website'

inputs:

azureSubscription: 'Resource Manager - Tailspin - Space Game'

appName: '$(WebAppNameStaging)'

package: '$(Pipeline.Workspace)/drop/$(buildConfiguration)/\*.zip'

This pipeline adds to the PowerShell script. It checks the generated SQL script for the keywords **CREATE**, **ALTER**, and **DROP**. If any of these words are found, the script creates a variable named schemaChanged in the pipeline variable group. Then a condition is added to the DBAVerificationApply stage to check for this variable. If this variable is True, a change needs approval. If the variable isn't present, the script contains no changes and this stage is skipped because the condition fails.

1. Add and commit *azure-pipelines.yml* to your branch. Then push the changes to your GitHub remote, origin. This step causes the pipeline to run.

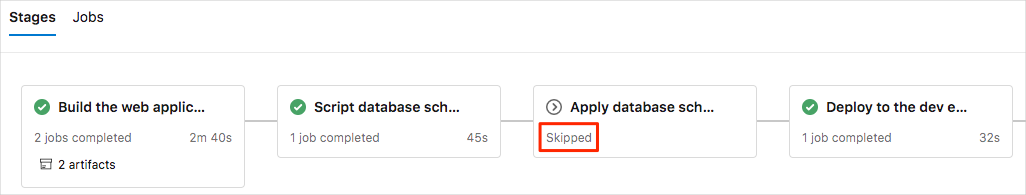
**Bash**

git add azure-pipelines.yml

git commit -m "Add condition for database approval stage"

git push origin schema-changes

1. Navigate back to Azure DevOps and watch the pipeline run. You see that the DBAVerificationApply stage is skipped because the schema didn't change.



**Recommended practices**

In this exercise, you created a variable to use as a condition for a stage. In practice, the variable must be unique for each pipeline run because you might be running this pipeline in parallel with other changes.