# Complete Azure DevOps CI/CD guide for your Azure Synapse based Data Platform — part I

[Complete Azure DevOps CI/CD guide for your Azure Synapse based Data Platform — part I | by Koen Leijsten | Cmotions | Medium](https://medium.com/cmotions/complete-azure-devops-ci-cd-guide-for-your-azure-synapse-based-data-platform-part-i-132a542bca96)

This guide includes code snippets which you can use yourself. Credits to my colleagues at Delegate who have contributed a lot to this domain. If you would like to check out more extensive CI/CD code or contribute yourself check out: <https://github.com/atc-net/atc-snippets/tree/main/azure-cli/synapse/Publish>.

You have setup your synapse workspace and are busy developing your data platform, there are already some reports that make use of that data platform. You receive some complaints from the analysts that your development work is causing problems for the reports depending on the output of the data platform. You conclude that it is time to separate the development environment from the production environment. You want a professional approach and automate deployment to production in a structured way. Continuous Integration and Continuous Development (CI/CD) is the perfect way to automate your deployment process, but you don’t know where to start.

No worries, this guide will help you from beginning to end with implementing CI/CD for your Azure Synapse based data platform. In this first part we will take a deep dive into CI/CD with Azure DevOps and in the second part we will highlight some of the best practices.

## Azure DevOps CI/CD process for your Synapse data platform

We start with a high-level overview of the CI/CD architecture in figure 1. This is the architecture that we are going to setup, but before we can set up this architecture we need to make some assumptions.

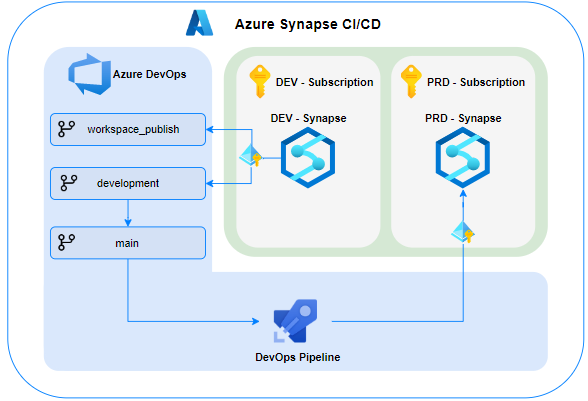


Figure 1: CI/CD architecture for Azure Synapse with Azure DevOps

* You have an organisation, project and repository ready to go in Azure DevOps in the same tenant as your Synapse workspace.
* You have one development Azure Synapse workspace and one empty production Azure Synapse workspace. In this guide we are going to assume that these are in different subscriptions, but this does not have to be the case (although it is recommended).
* You have two service principals (app registrations) in your Tenant, one for each of the subscriptions. Find out more about Application and service principal objects in the [Microsoft documentation](https://learn.microsoft.com/en-us/azure/active-directory/develop/app-objects-and-service-principals).
* These service principals need to have Contributor permissions on the resource group in order to create azure resources in their dedicated subscription.
* An Azure Active Directory (Azure AD) administrator must install the Azure DevOps Synapse Workspace Deployment Agent extension in your Azure DevOps organization. You can find out how you can install extensions in the following [link](https://learn.microsoft.com/en-us/azure/devops/marketplace/install-extension?view=azure-devops&tabs=browser).

In order to continue with this guide, the assumptions need to be met or at least take them in consideration while reading the guide.

## Connect your Azure Synapse workspace to an Azure DevOps repository

Before we can start implementing CI/CD we need to make sure that the Azure Synapse workspace is connected to your Azure DevOps repository, which will result in the DevOps branches shown in figure 1. The advantage of setting up the connection between Azure Synapse and Azure DevOps is that we can use Git for source control and versioning. We can set this up by going through the following steps.

1. Navigate to your Azure Synapse Workspace and select “manage” from the left-side menu.
2. Select “Git configuration” and “configure”.
3. Select the option “Azure DevOps Git” from the dropdown menu in “Repository type”
4. Choose the correct tenant which contains the Azure DevOps organisation and project.
5. Choose your DevOps organisation, project and repository which you already setup.
6. Create a “Collaboration branch” which will be your development branch.
7. The “Publish branch” will contain the JSON file which defines your Synapse workspace and will be updated when you press publish (we will come back to this later on).
8. Make sure the box beneath “Import existing resources” is checked. This will make sure that everything that you have built up to this point is pushed to the repository.
9. Select “Apply”

You now have successfully connected your Synapse workspace to your Azure DevOps workspace and we can continue with setting up your Azure DevOps environment.

## Create service connections in Azure DevOps to your Azure subscriptions

In order to automatically deploy the Synapse workspace from development to production we need to authenticate and authorize. We will be doing this using Azure service connections which make use of service principals in Azure (in figure 1 these are the lines between the Synapse workspaces and the DevOps resources). We already have a service principal with the sufficient rights for each subscription. The following steps will describe how we can access and use these service principals in Azure DevOps.

1. Go to your project settings at the bottom in the left-side menu of Azure DevOps.
2. Navigate to “service connections” beneath the header “Pipelines” and select “New service connection” in the top right corner.
3. Choose the option “Azure Resource Manager”.
4. Next choose the option “Service principal (manual)”.
5. Select the option “Subscription” for the scope level and fill in the correct subscription id. You can find the subscription Id in the Azure Portal under subscriptions.
6. Fill in all the required fields. Remember we are going to create a service connection for both the development and production requirement, so make sure you select the corresponding service principal.
7. Select Verify to test if the connection works.
8. If the connection works, give a name to the service connection, make sure to include the environment in the name (so development or production).
9. To finish up select “Verify and save”.

We need to repeat these steps twice, once for development and once for production. When you went through the steps for both development and production, you will be able to authenticate and authorize automatically when deploying to your Synapse workspace.

## Create your first Azure DevOps Pipeline

Now we can continue creating our Azure DevOps Pipeline. We will first create a starter pipeline which will serve as the starting point for our deployment pipeline. We will create this starter pipeline using the Azure DevOps UI, which will result in a YAML file. We will expand this YAML file during the rest of this guide.

1. Log into your Azure DevOps environment and go to the repository where you connected your Synapse workspace.
2. Go to the main branch.
3. Select “Pipelines” in the left-side menu and select “New pipeline” in the top-right corner.
4. Select “Azure Repos Git YAML” and select your repository.
5. Configure your pipeline as a “Starter pipeline”.
6. You will now see the YAML file, select “Save and run”.

Your starter pipeline will now run. You can click on “Job”, now you will see the steps that the YAML file is performing. Let’s take a closer look at the YAML file.

trigger:  
- main  
   
pool:  
 vmImage: ubuntu-latest  
   
steps:  
- script: echo Hello, world!  
 displayName: 'Run a one-line script'  
   
- script: |  
 echo Add other tasks to build, test, and deploy your project.  
 echo See https://aka.ms/yaml  
 displayName: 'Run a multi-line script'

Firstly, the trigger of the pipeline is defined, the pipeline will trigger when the main branch is updated. Secondly virtual machine (VM) on which the pipeline runs is specified, the pipeline runs on a Linux VM with the latest ubuntu distribution. Next, the steps of the pipeline are defined. The starter pipeline contains two steps which both run an inline shell (Bash) script. Step one prints “Hello, world!” and the second step prints two strings in the shell. You can see that the second step contains “displayName” parameter which is used to give a step a descriptive name. When you run the pipeline you will see a step with the name “Run a multi-line script”.

## Deploy a Synapse workspace with your Azure DevOps pipeline

Now that we have a starting point for our deployment pipeline we can start configuring the YAML file to our needs. We will start with deploying our development Synapse workspace to our production Synapse workspace. This will only be a “simple” copy paste action on which we can build.

We will start with adding some parameters to the file.

parameters:  
- name: subscriptionPrd  
 type: string  
 default: service-connection-prd  
   
- name: subscriptionDev  
 type: string  
 default: service-connection-dev  
   
- name: resourceGroupNamePrd  
 type: string  
 default: resource-group-prd  
   
- name: resourceGroupNameDev  
 type: string  
 default: resource-group-dev  
   
- name: synapseWorkspaceNamePrd  
 type: string  
 default: service-connection-prd  
   
- name: synapseWorkspaceNameDev  
 type: string  
 default: service-connection-dev

We start with the parameters for the service connections to both the development and production subscriptions. These will make sure we can authenticate and authorize. Next, we have the names for our development and production resource groups and Synapse workspaces.

We talked about the workspace\_publish branch in the previous section, we will now further elaborate on this branch. Our development Synapse workspace is defined in the workspace\_publish branch which is automatically created when you connect your Synapse workspace to Azure DevOps. This branch contains two JSON files; “TemplateForWorkspace.json” and “TemplateParametersForWorkspace.json”. The first file contains the complete definition of our Synapse workspace and the second file contains global parameters in the Synapse workspace. We will be adjusting the parameter file later on, for now we will be doing a simple deployment where we just copy the development environment to the production environment.

resources:  
 repositories:  
 - repository: PublishBranch  
 type: git  
 name: 'name of your repository'  
 ref: workspace\_publish  
   
steps:  
   
- checkout: PublishBranch  
 path: PublishBranch

We need these files in order to deploy to our production environment. Therefore, we need to make sure that we can access the files during deployment. In the code above we add the “workspace\_publish” branch as resource to our pipeline. In next we checkout the “the workspace\_publish” branch so that we can make changes to the files if needed, we will come back to this later on.

Now we can finally implement the most important step, deploying the Synapse workspace in our production environment.

#Deploy synapse production environment  
- task: Synapse workspace deployment@2  
 continueOnError: false  
 displayName: 'Deploy Synapse Workspace'  
 inputs:  
 operation: 'deploy'  
 TemplateFile: '$(Agent.BuildDirectory)/PublishBranch/main/TemplateForWorkspace.json'  
 ParametersFile: '$(Agent.BuildDirectory)/PublishBranch/TemplateWorkspaceParameters.json'  
 azureSubscription: '${{ parameters.subscriptionPrd }}'  
 ResourceGroupName: '${{ parameters.resourceGroupNamePrd }}'  
 TargetWorkspaceName: '${{ parameters.synapseWorkspaceNamePrd}}'  
 DeleteArtifactsNotInTemplate: true

This task will execute our Synapse workspace deployment. First, we specify that this task must not be executed if we get any errors in the previous steps, to make sure we don’t deploy faulty code. In the input parameter we specify the definition of our Synapse workspace. In the “Templatefile” parameter we will refer to the “TemplateForWorkspace.json” file in our checked out “publishBranch”. We will do the same for “ParametersFile”. Because we are deploying to our production Synapse workspace we will authenticate and authorize with our production service connection in the “azureSubscription” parameter. To make sure our development and production environment stay aligned we include the “DeleteArtifactsNotInTemplate” parameter and set it to true. This setting will delete everything in the production Synapse workspace that is not defined in the “TemplateForWorkspace.json” file that we deploy.

When we now run the Azure DevOps Pipeline we will deploy our development Synapse workspace to our production Synapse workspace in an automated way. We will continue with parameterizing our deployment to production.

## Parameterizing the deployment to production

Because our development Synapse workspace is in a different environment than our production Synapse workspace we need to make some adjustments to our parameters. For example, we have a connection string to a storage account. The connection string will be different in the production environment, because we want to connect to a different storage account. The “TemplateParametersForWorkspace.json” file contains key value pairs where we need to replace the value. We will use the adjusted “TemplateParametersForWorkspace.json” file with the replaced values in our deployment step.

Let’s walk through the process. First we are going to create a new folder in our main branch root and name it “deploy”. In this folder we are going to create a JSON file named “synapse-parameters.json”. This file contains the key value pairs of the parameters we would like to replace in our “TemplateParametersForWorkspace.json” file, for example:

{   
 "workspaceName": "synapse\_workspace-prd",   
}

Next we create a PowerShell file in the “deploy” folder and name it “initialize-synapse-parameters.ps1”. This file contains the following code:

function Initialize-SynapseParameters {  
 param (  
 [Parameter(Mandatory = $true)]  
 [ValidateNotNullOrEmpty()]  
 [string]  
 $SynapseParameterJsonPath,  
   
 [Parameter(Mandatory = $true)]  
 [ValidateNotNullOrEmpty()]  
 [string]  
 $OutputPath,  
   
 [Parameter(Mandatory = $true)]  
 [ValidateNotNullOrEmpty()]  
 [string]  
 $ParameterJsonPath  
 )  
   
 # Get the Synapse generated Parameters file  
 $synapseParameters = Get-Content -Raw $SynapseParameterJsonPath | ConvertFrom-Json -AsHashTable  
   
 Write-Host "Parameterize using jsonfile: $ParameterJsonPath" -ForegroundColor Yellow  
 $parameterUpdates = Get-Content -Raw $ParameterJsonPath | ConvertFrom-Json -AsHashTable  
   
 foreach ($parameter in $parameterUpdates.GetEnumerator()) {  
 $synapseParameters.parameters.$($parameter.Name).value = $parameter.Value  
 }  
   
 # New re-parameterized file ready for prod  
 Write-Host "Saved parameterized workspace file at: $OutputPath"  
 $synapseParameters | ConvertTo-Json | Out-File "$($OutputPath)"  
}  
   
Initialize-SynapseParameters `  
 -OutputPath $OutputPath `  
 -SynapseWorkspaceParameterJsonPath $SynapseParameterJsonPath `  
 -ParameterJsonPath $ParameterJsonPath

This PowerShell file contains a function that will replace the values based on their key in the “TemplateParametersForWorkspace.json” file and output it to the specified path. We are going to call this function in our DevOps pipeline with YAML code. But first we need to make sure we can access the PowerShell file we just created. Therefore we need to checkout the main branch which we will add underneath the checkout of the “PublishBranch” we did earlier.

resources:  
 repositories:  
 - repository: PublishBranch  
 type: git  
 name: 'name of your repository'  
 ref: workspace\_publish  
   
steps:  
   
- checkout: PublishBranch  
 path: PublishBranch  
   
- checkout: self  
 path: main

Now that we are able to access the files in the main branch, we can also access the PowerShell file and call the function with the YAML code below.

#Parameterize Workspace Parameter File  
- task: PowerShell@2  
 displayName: 'Parameterize Workspace Parameter File'  
 inputs:  
 targetType: inline  
 script: |  
 cd $(Agent.BuildDirectory)/main  
 ./deploy/publish-synapse-parameters.ps1 `  
 -OutputPath '$(Agent.BuildDirectory)/main/CustomWorkspaceParameters.json' `  
 -SynapseParameterJsonPath '$(Agent.BuildDirectory)/PublishBranch/TemplateParametersForWorkspace.json' `  
 -ParameterJsonPath '$(Agent.BuildDirectory)/main/deploy/synapse-parameters.json'

This YAML code first changes the directory to our checked-out main branch and then runs the PowerShell script we just created. We specify the parameters as follows:

* OutputPath: we set the path for our output file with the replaced values.
* SynapseParameterJsonPath: this points to the “TemplateParametersForWorkspace.json” file.
* ParameterJsonPath: we set the path to the “synapse-parameters.json” file which contains the new values for the production environment.

All of this results in a new “CustomWorkspaceParameters.json” file which contains the new values for the production environment that we can use for the Synapse deployment step. Instead of referring to the “TemplateParametersForWorkspace.json” file we can now refer to the “CustomWorkspaceParameters.json” file when deploying to production.

## Summary

Congratulations! You are now able to deploy your Synapse based data platform in an automated way using Azure DevOps. We talked about setting up your Synapse workspace and Azure DevOps environments, doing a simple deployment and customizing your deployment using parametrization. These are the first important steps for setting up your CI/CD process.

In the next part we will dig even deeper and highlight some of the specific use cases that we can add to make our CI/CD process and make it even more automated and dynamic.