Authenticate as a service principal to run a Microsoft Fabric notebook from Azure DevOps

Published by Kevin Chant on January 31, 2025

Reading Time: 6 minutes

In this post I want to share one way that you can authenticate as a service principal to run a Microsoft Fabric notebook from Azure DevOps.

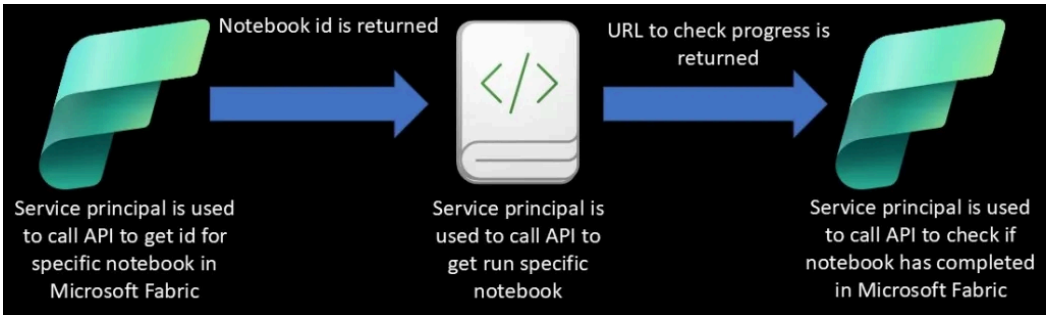


Diagram highlighting different API calls

Some of you may recall that I previously covered how to run a Microsoft Fabric notebook from Azure DevOps.

I decided to published a newer version of the aforementioned post to amplify the fact that the REST API that runs a notebook on demand now supports service principals.

As you can see in the updated text which can be found in the Microsoft article.

> Service principal authentication is available for Notebook CRUD API and Job scheduler API, meaning you can use service principal to do the CRUD operations and trigger/cancel notebook runs, and get

> the run status. You need to add the service principal to the workspace with the appropriate role.

To be honest, I am really excited about this update, and I am sure a lot of you reading this post are as well.

By the end of this post, you will know one way that you can use a service principal run a Microsoft Fabric notebook from Azure DevOps. Which you can customize how you see fit to align with any security policies your company has. Along the way I share plenty of links.

Tip: To solely see that the service principal works you can go straight to the section that covers the second task and from there the test results section.

My Microsoft Fabric notebook

I decided to work with the same notebook that I used in my post about unit tests on Microsoft Fabric items.

You can find out more details about it in that post. However below, is an overview of what it does since I added a bit more to it for my post that covers how to run a Microsoft Fabric notebook from Azure DevOps.

- First it issues the %%configure magic command to set the default Lakehouse.
- It then installs pytest.
- Afterwards, it imports pytest and pandas.
- It then inserts data into spark dataframes and then converts them to pandas dataframes.
- Once done, it creates a function that tests for missing values.
- It then runs the assertion test.

Azure DevOps preparation to authenticate as a service principal

In Azure DevOps I worked with two variable groups. One for sensitive values and another for non-sensitive values. Which contain the below variables:

- servicePrincipalId – The clientid of a service principal.
- servicePrincipalKey – The secret of a service principal.
- tenantId – Tenant id of the tenant hosting Microsoft Fabric.

- resourceUrl – Main API URL to authenticate against for a token, in this case “https://api.fabric.microsoft.com”.
- workspaceId – Id value of workspace that contains notebook.
- notebookName – Name of the notebook.

As you can see, this time around I solely work with the service principal.

With these variables in place I can create the below three tasks in a YAML pipeline in Azure DevOps.

First task: Get notebook id in Azure DevOps

Just like before I decided to create one job in my YAML pipeline with multiple tasks. With my first task calling an API to return the id value of a specific notebook.

So, I first authenticated with the service principal and returned a token I can use to call the API with the below code.

```
$SecureStringPwd = ConvertTo-SecureString $(servicePrincipalKey) -
AsPlainText -Force
$pscredential = New-Object -TypeName
System.Management.Automation.PSCredential -ArgumentList
$(servicePrincipalId), $SecureStringPwd

Connect-AzAccount -ServicePrincipal -Credential $pscredential -Tenant
$(tenantId)

# Get authentication
$fabricToken = (Get-AzAccessToken -ResourceUrl $(resourceUrl)).Token
```

Afterwards, I called the API.

```
$fabricToken = (Get-AzAccessToken -ResourceUrl $(resourceUrl)).Token

$apiUrl =
"https://api.fabric.microsoft.com/v1/workspaces/$(workspaceId)/noteboo
ks"
$headers = @{
"Authorization" = "Bearer $fabricToken"
"Content-Type"  = "application/json"
}

$list = Invoke-RestMethod -Uri $apiUrl -Headers $headers -Method GET
$notebookId = ($list.value | Where-Object {$_.displayName -eq
"$(notebookName)"}).id
```

```
Write-Host "##vso[task.setvariable variable=notebookId;]$notebookId"

pwsh: true
```

I performed this with the Invoke-RestMethod PowerShell cmdlet. Due to its flexibility. As you can see, I also stated pwsh to be true at the end of the task. Which allows me to work with newer parameters supported by this cmdlet.

One other key point I want to highlight is that the returned notebookId value is added as a new variable in my pipeline.

Second task: Authenticate as a service principal to run Microsoft Fabric notebook from Azure DevOps

As you can see below, this is the part where everything changes. Because this time around I **authenticated as the service principal** and got a bearer token with the below code.

```
$SecureStringPwd = ConvertTo-SecureString $(servicePrincipalKey) -
AsPlainText -Force
$pscredential = New-Object -TypeName
System.Management.Automation.PSCredential -ArgumentList
$(servicePrincipalId), $SecureStringPwd

Connect-AzAccount -ServicePrincipal -Credential $pscredential -Tenant
$(tenantId)
```

I then built up the required parameters and called the API with Invoke-RestMethod.

```
$startnotebookUrl =
"https://api.fabric.microsoft.com/v1/workspaces/$(workspaceId)/items/$
(notebookId)/jobs/instances?jobType=RunNotebook"

$headers = @{
"Authorization" = "Bearer $fabricToken"
 "Content-Type"  = "application/json"
}

Invoke-RestMethod -Uri $startnotebookUrl -Headers $headers -Method
POST -ResponseHeadersVariable headers
```

On a side note, you can also build up the same configuration as the Spark session configuration magic command in URL for the API call. Providing you the flexibility to specify various settings like additional jar files for libraries dynamically.

You can see a good example of this in the documentation that covers how to run a notebook on demand.

Another key point I want to highlight about the above code is that I added the -ResponseHeadersVariable parameter. So that I can get a returning URL which allows me to call an API to check the progress of the notebook.

I added the returned value as a variable in the Azure DevOps pipeline so that I can call it in the next task.

```
$statusUrl = $headers.location[0]

Write-Host "##vso[task.setvariable variable=statusUrl;]$statusUrl"
```

You can read about this in-detail in the guide to working with REST APIs and PowerShell's Invoke-RestMethod.

Third task: Checking the progress of the Microsoft Fabric notebook in Azure DevOps

Finally, I created a task that polls to check the progress of the Microsoft Fabric notebook whilst it is running.

Because I can use a service principal with the API that checks the progress of the running notebook, I authenticated with a service principal again.

```
Install-Module -Name Az.Accounts -AllowClobber -Force

$SecureStringPwd = ConvertTo-SecureString $(servicePrincipalKey) -
AsPlainText -Force
$pscredential = New-Object -TypeName
System.Management.Automation.PSCredential -ArgumentList
$(servicePrincipalId), $SecureStringPwd

Connect-AzAccount -ServicePrincipal -Credential $pscredential -Tenant
$(tenantId)
```

```
# Get authentication
$fabricToken = (Get-AzAccessToken -ResourceUrl $(resourceUrl)).Token
```

From there, I built up the header for my API call before calling the API to check the current status of the notebook in Microsoft Fabric.

```
$headers = @{
"Authorization" = "Bearer $fabricToken"
"Content-Type"  = "application/json"
}

# Write-Host $(statusUrl)

$status = (Invoke-RestMethod -Uri $(statusUrl) -Headers
$headers).status
```

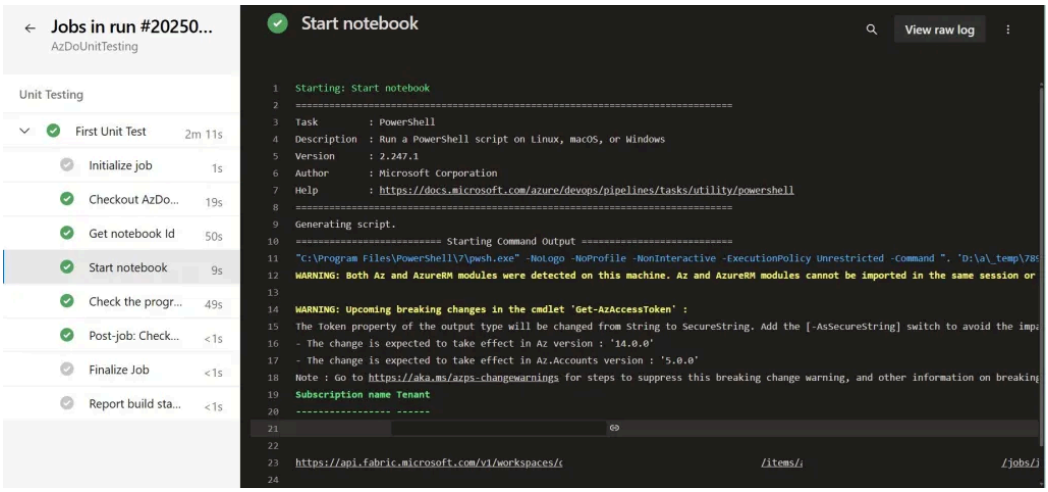
I then configured the below logic. Which polls for the status of the running notebook every ten seconds and will inform me if the notebook has either completed of failed.

```
while ($status -ne 'completed' -and $status -ne 'Failed') {
Write-Host "Current status: $status. Waiting for notebook to be
completed..."
Start-Sleep -Seconds 10
$status = (Invoke-RestMethod -Uri $(statusUrl) -Headers
$headers).status
}

if ($status -eq 'completed') {
  Write-Output "Notebook has completed running."
} else {
  Write-Output "Notebook has failed, probably due to an assertion
failure. Check the stdout log for assertion failure."

  exit 1
}


pwsh: true
```

Testing that authenticating as a service principal to Microsoft Fabric notebooks really works

I prepared for my first test by disabling the final cell in my notebook which runs the assertion test. Afterwards, I ran the pipeline in Azure DevOps.

Which appeared to work. I then selected the “Start notebook” task from the job history for certainty and could see that it started okay.

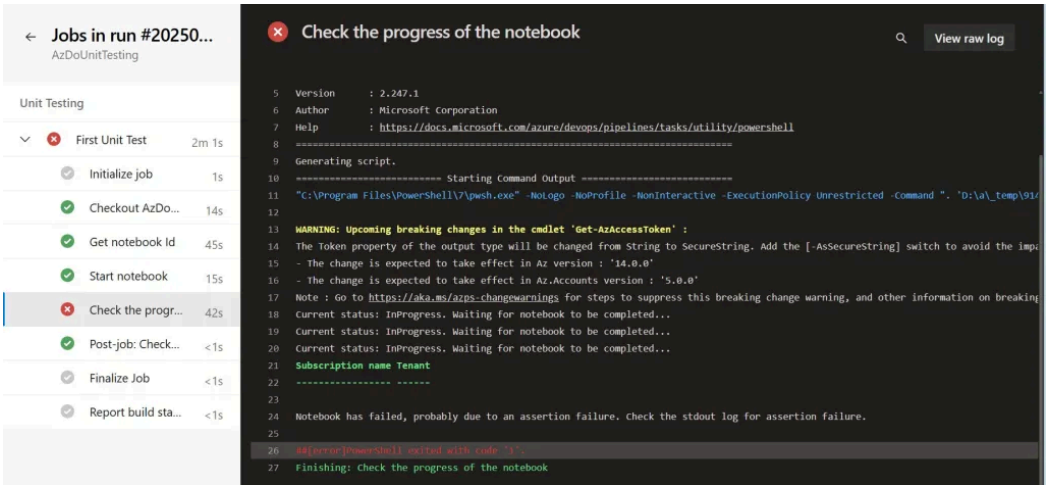


Viewing job history

For peace of mind, I checked the final task which polls the status of the notebook. Which also worked as expected.

I finally verified it by checking the recent runs for the notebook in Microsoft Fabric. Which showed it had completed okay.

My final test was to check that it would detect that an assertion test failed. Because it is good practice to test for failures. So, I enabled the final cell in the notebook which performs the assertion test before running the pipeline again.



Failed task in Azure DevOps

I verified this in Microsoft Fabric by selecting the recent runs of the notebook. In order to view its history. All three of the below runs took place **after** service principal authentication was configured.

Final words about authenticating as a service principal to run a Microsoft Fabric notebook

It has been a long time coming, but finally we can authenticate as a service principal to run a Microsoft Fabric notebook through REST APIs.

I am really excited about this recent update, and I am sure some of you are as well.

Of course, if you have any comments or queries about this post feel free to reach out to me.

Published in Azure DevOps Microsoft Fabric

Azure DevOps Microsoft Fabric service principals

Previous Post
Four ways to monitor a semantic model refresh in Microsoft Fabric

Next Post
Thoughts about the Azure Data Studio retirement announcement

5 Comments

Srikanth

Thank you for the code snippets and description. We will try from end and post my observations here

---

February 1, 2025 | Reply

Happy together paths to test semantic models in Microsoft Fabric feature workspaces with BPA - Kevin Chant

[…] though I covered previously that you can authenticate as a service principal to run a Microsoft Fabric notebook from Azure DevOps there a lot of things to consider if you want to attempt to run the BPA through semantic link in […]

Initial tests of fabric-cicd - Kevin Chant

[…] Deploying the Lakehouses inspired me to test fabric-cicd parameterization. Because I often need to change the default Lakehouses within notebooks. Like I mentioned in my previous post about authenticating as a service principal to run a notebook. […]

Jonathan Bateman

This is a fantastic article about running a fabric notebook using azure devops ci/cd pipeline. I was wondering how you setup your service principal. Following your guide step by step, I always end up with a PrincipalNotSupportedType.

February 28, 2025 | Reply

Kevin Chant (author)

When do you get that error message?

March 1, 2025 | Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name*

Email*

Website

Notify me of follow-up comments by email.

Notify me of new posts by email.

Post Comment