PowerShell SQL Deployment – Part 1: PowerShell

By Andrew Hinkle

[*DogFoodCon*](https://dogfoodcon.com/) *2019 is October 3rd – 4th.* [*Ben Miller*](https://dogfoodcon.com/speakers/#sz-speaker-9d0a0d9d-bef7-40b6-a81a-ff09cdefa74e) *will be presenting* [*[DevOps] DBAs working Smarter not Harder with PowerShell*](https://dogfoodcon.com/sessions/#sz-session-126790) *and* [*[SQL/BI] Inside the DBAtools PowerShell Module*](https://dogfoodcon.com/sessions/#sz-session-126789) *. I interviewed Ben Miller regarding his presentation and have included a summary at the end of this article.*

Azure build definitions provide the capabilities to get code from source control, compile the code, run its unit tests, and stage the code, along with many other features. Azure release definitions take the staged code and replace tokens with environment specific values, run UI tests like Selenium, and much more. Jonathan Danylko showed us how to apply the same concept for SQL by [Deploying Databases using Azure DevOps Pipelines](https://www.danylkoweb.com/Blog/deploying-databases-using-azure-devops-pipelines-NQ) with [DACPACs](https://docs.microsoft.com/en-us/sql/relational-databases/data-tier-applications/data-tier-applications?view=sql-server-2017).

An alternative is to deploy your SQL files via PowerShell. You may create as many SQL files as you want following a simple naming convention to enforce the scripts are applied in the order you want. The files are plain text, so you may review them before deployment.

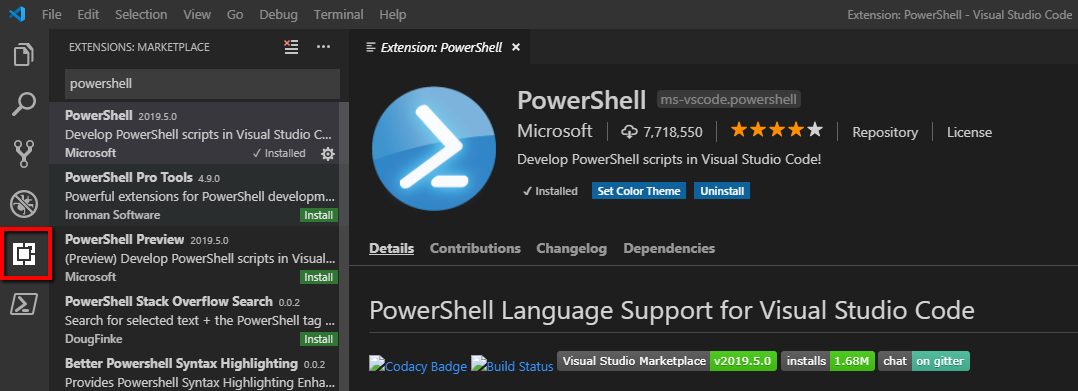
In this article I'll walk you through the steps of setting up Visual Studio Code with PowerShell and the SQL Server module. Then you'll run a simple Invoke-Sqlcmd query against your SQL Server instance containing a database. I'll provide a PowerShell module that wraps up the Invoke-Sqlcmd to handle an array of sql files with output messaging. Finally, we'll create a script that wraps the call to the module with exception handling and output messaging targeting Azure. In Part 2 I'll cover the next steps of creating a build and release definition, add release definition variables for each environment that will be supplied as parameters to the PowerShell script, and test.

# Assumptions

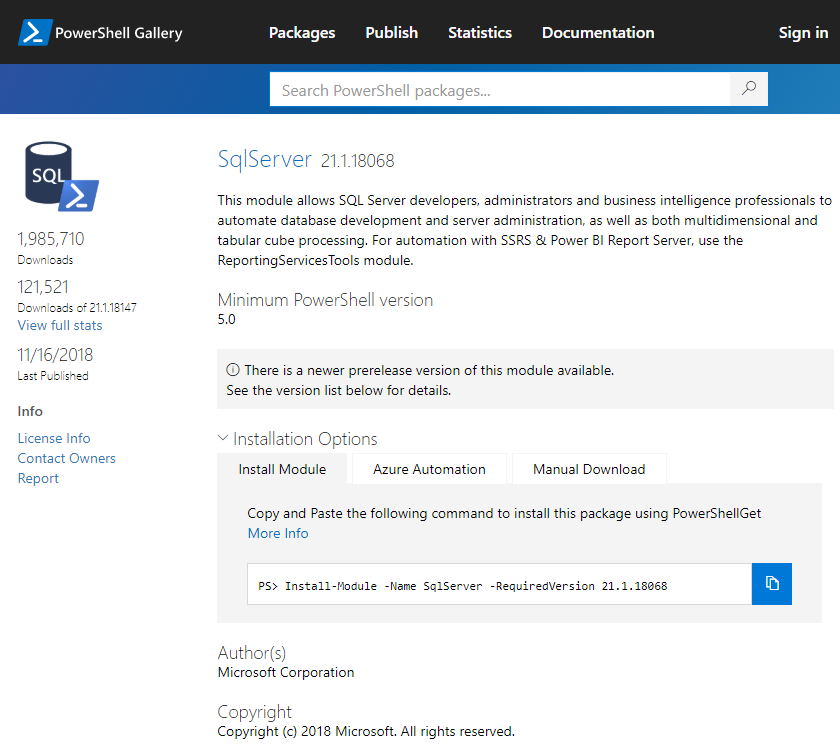
1. Your Windows account has permissions to create stored procedures on a SQL Server database

# Setup

1. Install Visual Studio Code
   1. <https://code.visualstudio.com/>
   2. Avoid most permission issues by running Visual Studio Code as an Administrator
2. Install Visual Studio Code > PowerShell Extension
   1. Run Visual Studio Code as an Administrator > Extensions > Search for "powershell" > Click Install



1. Install PowerShell > SQL Server module
   1. <https://www.powershellgallery.com/packages/SqlServer/21.1.18068>



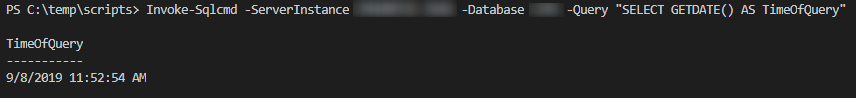
* 1. "[New Features and Best Practices of SQL Server PowerShell](https://www.mssqltips.com/sqlservertip/4993/new-features-and-best-practices-of-sql-server-powershell/)" is a great tutorial on installing the PowerShell SQL Server module. That article goes into great detail, so the following is the short version of what I did.
     1. I did run into some issues while trying to install the module beyond what was mentioned here that you may read in my [notes](https://github.com/penblade/Tips/blob/master/Tips.PowerShellSQLDeployment/docs/Tips.PowerShell.SQL.Deployment.Notes.docx)
  2. Copy the install module command and run it in the Visual Studio Code > PowerShell extension
  3. Install-Module -Name SqlServer -RequiredVersion 21.1.18068
     1. I was prompted to install the latest NuGet PackageProvider, which I answered yes to.

# Invoke-Sqlcmd

Execute a Query to verify you're setup correctly. The primary function that I've used in this process is [Invoke-Sqlcmd](https://docs.microsoft.com/en-us/powershell/module/sqlserver/invoke-sqlcmd?view=sqlserver-ps). Let's try it now.

1. In Visual Studio Code > PowerShell run the following commands
   1. Replace $ServerInstance with the SQL Server instance name, which may be as simple as the computer name depending on how it was setup. When you open SQL Server Management Studio and connect to a server instance, that's the name you use.
   2. Replace $Database with the database name.
   3. Make sure you are using actual hyphen "-" and not the bigger dash "–" that MS Word and other text editors like to replace it with.





# Taking it further

With this command alone you can build a process around it that will string together any number of PowerShell commands to prep your environments, run your queries, and clean up after yourself. The following sections are files with their contents. I created the following folders and placed all of the SQL files (.sql) in the sql folder and the PowerShell scripts (.ps1, .psm1) in the scripts folder to test them.

1. C:\temp\scripts
2. C:\temp\sql

I chose to separate the concerns of applying the SQL script into a reusable module. I isolated the gathering of the SQL files and Azure exception handling in the DeploySql PowerShell script. You can choose to combine the module and script into a single script. I like the flexibility to custom the exception handling to the deployment pipeline without affecting the core script.

# 00.Setup.sql



# 01.Deploy.sql



# ##.\*.SQL

Add as many SQL statements to deploy as you need.

# 99.TearDown.sql



# COMPANYNAME.DevOps.Sql.psm1





# DeploySql.ps1





# Run the PowerShell script

In Visual Studio Code > PowerShell run the following command. Remember to adjust the ServerInstance, Database, and SqlFolder appropriately.

.\DeploySql -DevOpsSqlModule "C:\temp\scripts\COMPANYNAME.DevOps.Sql.psm1" -ServerInstance SERVERINSTANCE1 -Database DATABASE1 -SqlFolder "C:\temp\sql\DATABASE1\"

# 

# Next Steps

Given these example PowerShell scripts research the steps and learn what each command does. Adjust them to meet your needs and get excited that you deployed SQL from PowerShell!

In Part 2 I'll cover the next steps of creating a build and release definition, add release definition variables for each environment that will be supplied as parameters to the PowerShell script, and test.

# References

1. [New Features and Best Practices of SQL Server PowerShell](https://www.mssqltips.com/sqlservertip/4993/new-features-and-best-practices-of-sql-server-powershell/)
2. [PowerShell Invoke-SQLCmd outputs DataTables you can INSERT into SQL Server](https://www.mssqltips.com/sqlservertip/4678/powershell-invokesqlcmd-outputs-datatables-you-can-insert-into-sql-server/)
3. [Navigate SQL Server PowerShell Paths](https://docs.microsoft.com/en-us/sql/powershell/navigate-sql-server-powershell-paths?view=sql-server-2017)
4. [PowerShell Documentation and Comments](https://poshcode.gitbooks.io/powershell-practice-and-style/Style-Guide/Documentation-and-Comments.html)

# DogFoodCon 2019

[DogFoodCon](https://dogfoodcon.com/) 2019 is October 3rd – 4th. [Ben Miller](https://dogfoodcon.com/speakers/#sz-speaker-9d0a0d9d-bef7-40b6-a81a-ff09cdefa74e) will be presenting [[DevOps] DBAs working Smarter not Harder with PowerShell](https://dogfoodcon.com/sessions/#sz-session-126790) and [[SQL/BI] Inside the DBAtools PowerShell Module](https://dogfoodcon.com/sessions/#sz-session-126789). Ben Miller walked me through the entire process he follows to deploy SQL to the hundreds of databases he services. I was thoroughly impressed with the attention to detail to ensure the stability, accuracy, and reliability he invested into his deployment process. I highly recommend attending his session if you have any interest in deploying SQL in an automated manner. Whether you prefer to deploy DACPAC or SQL files you'll learn from the insights he'll provide in this session.

From Ben Miller's site <https://dbaduck.com/about-me/>:

Ben has been a member of the SQL Server Community for over 17 years now. He is currently working at [MaritzCX](https://www.maritzcx.com/) revolutionizing the SQL Server environment and loving it. He also consults as well. He is a SQL Server MVP and Microsoft Certified Master (MCM) and has spent time in the field using SQL Server since 1997. He has worked at various companies throughout the US as well as at Microsoft for 7 years. He is passionate about SQL Server and automation and integration and uses SMO and PowerShell regularly.

By attending his session you'll learn the following:

1. Pros/Cons of SQLCompare and Database Projects
2. Shred the DACPAC into deployment files by categories in a specific order enforced by Microsoft standards
   1. This section is the primary focus
   2. Why/How to shred the DACPAC
   3. Standards to enforce a reliable/stable deployment experience
3. PowerShell SQL Deployments
   1. What is covered here, plus additional details Ben Miller

|  |
| --- |
| [DBADuck – Ben Miller](https://dbaduck.com/)[Insight Digital Innovation's Twitter Page](http://www.twitter.com/DBAduck) [Jeff McKenzie's LinkedIn Profile](http://www.linkedin.com/in/dbaduck) |

# Conclusion

I've demonstrated how to setup Visual Studio Code with the PowerShell extension and Sql Server module and verified the setup by executing a simple query. You have learned how to deploy multiple SQL files with exception handling and Azure output messaging. Next steps include Part 2 where you'll learn how to setup an Azure Build and Release definitions to deploy SQL through your DevOps pipeline. Next steps also include attending Ben Miller's sessions at DogFoodCon 2019.

Were you able to run the Invoke-Sqlcmd statement? Were you able to deploy any SQL scripts to your database? Do you deploy your SQL scripts via DACPAC or SQL files? Have you automated the process or is it fairly manual? Do you deploy your SQL in a different way? Share in the comments below.