Contents

- Azure Data Studio documentation
- Download Azure Data Studio
- Release notes
- Overview
  - Azure Data Studio
- Quickstarts
  - Connect & query SQL Server
  - Connect & query Azure SQL Database
  - Connect & query Azure Synapse Analytics
  - Connect & query PostgreSQL
- Tutorials
  - T-SQL Editor
  - Sample insight: slow queries
  - Sample insight: table space usage
  - Build a custom insight
  - Backup & restore
- Concepts
  - Extensions
    - Extensions overview
    - Azure Arc extension
    - Data Virtualization extension
    - Kusto extension
    - Machine Learning extension
      - Machine Learning extension overview
      - Manage packages in database
      - Make predictions
      - Import and view models
    - Managed Instance extension
    - PostgreSQL extension

- PowerShell extension
- SandDance extension
- Schema Compare extension
- SQL Database Projects extension
  - SQL Database Projects extension overview
  - Getting Started with the SQL Database Projects extension
  - Build and Publish a Project
  - Build a Project from the Command Line
- SQL Server Agent
- SQL Server Central Management Servers extension
- SQL Server dacpac extension
- SQL Server Import
- SQL Server Profiler

Extension Authoring

- Extension authoring
- Extensibility overview
- Extensibility APIs
- Keymap extension
- Dashboard extension
- Notebook extension
- Jupyter Book extension
- Wizard extension

Integrated terminal (PowerShell, Bash)

Preview features

Server groups

Source control

How-to guides

Notebooks

- Notebooks overview
- Use SQL Kernel
- Use Python Kernel
- Use Kusto Kernel

- Use Kqlmagic
- Parameterization
  - Papermill Parameterization
  - URI Parameterization
  - Run with Parameters
- Deployment
  - Deploy Azure SQL Database
  - Deploy Azure SQL Edge
  - Deploy SQL Server on Azure Virtual Machines
- Dashboards & insight widgets
- Code snippets
- Explorer Azure SQL resources
- Keyboard shortcuts
- Usage data collection
- Windows authentication (Kerberos)
- Workspace & user settings

References

- SQL tools
- Azure Developer tools

Resources

- FAQ
- Troubleshooting
- Azure Data Studio GitHub repo >
- Report issues & make suggestions >
- SQL Server documentation >

Download and install Azure Data Studio

6/18/2021 • 8 minutes to read • Edit Online

Azure Data Studio is a cross-platform database tool for data professionals using on-premises and cloud data platforms on Windows, macOS, and Linux.

Azure Data Studio offers a modern editor experience with IntelliSense, code snippets, source control integration, and an integrated terminal. It's engineered with the data platform user in mind, with the built-in charting of query result sets and customizable dashboards.

Use Azure Data Studio to query, design, and manage your databases and data warehouses, wherever they are - on your local computer or in the cloud.

For more information about Azure Data Studio, visit What is Azure Data Studio.

Download Azure Data Studio

Azure Data Studio 1.30.0 is the latest general availability (GA) version. If you have a previous GA version of Azure Data Studio installed, installing Azure Data Studio 1.28.0 upgrades it to the latest version.

- Release number: 1.30.0
- Release date: June 17, 2021

PLATFORM	DOWNLOAD
Windows	User Installer (recommended) System Installer .zip
macOS	.zip
Linux	.deb .rpm .tar.gz

> **NOTE**
> Azure Data Studio currently does not support the ARM architecture.

If you have comments or suggestions or want to report issues, the best way to contact the Azure Data Studio team is to file an issue on the Azure Data Studio feedback page.

Install Azure Data Studio

Windows install

> **IMPORTANT**
>
> Beginning with SQL Server Management Studio (SSMS) 18.7, Azure Data Studio is automatically installed alongside SSMS. Users of SQL Server Management Studio are now able to benefit from the innovations and features in Azure Data Studio. Azure Data Studio is a cross-platform and open-source desktop tool for your environments, whether in the cloud, on-premises, or hybrid.
>
> To learn more about Azure Data Studio, check out What is Azure Data Studio or the FAQ.

This release of Azure Data Studio includes a standard Windows Installer experience and a .zip file.

We recommend the *user installer* because it doesn't require administrator privileges, simplifying installs and upgrades. The user installer doesn't require Administrator privileges as the location is under your user Local AppData (LOCALAPPDATA) folder. The user installer also provides a smoother background update experience. For more information, see User setup for Windows.

User Installer (recommended)

1. Download and run the Azure Data Studio *user* installer for Windows.
2. Start the Azure Data Studio app.

System Installer

1. Download and run the Azure Data Studio *system* installer for Windows.
2. Start the Azure Data Studio app.

.zip file

1. Download Azure Data Studio .zip for Windows.
2. Browse to the downloaded file and extract it.
3. Run `\azuredatastudio-windows\azuredatastudio.exe`

Unattended install for Windows

You can also install Azure Data Studio using a command prompt script.

If you want to install Azure Data Studio in the background with no GUI prompts, and you're on the Windows platform, then follow the steps below.

1. Launch the command prompt with elevated permissions.
2. Type the command below in the command prompt.

```
<path where the azuredatastudio-windows-user-setup-x.xx.x.exe file is located> /VERYSILENT
/MERGETASKS=!runcode>
```

Example:

```
%systemdrive%\azuredatastudio-windows-user-setup-1.24.0.exe /VERYSILENT /MERGETASKS=!runcode
```

> **NOTE**
>
> The example also works with the system installer file.
>
> ```
> <path where the azuredatastudio-windows-setup-x.xx.x.exe file is located> /VERYSILENT
> /MERGETASKS=!runcode>
> ```

You can also pass */SILENT* instead of */VERYSILENT* to see the setup UI.

3. If all goes well, you can see Azure Data Studio installed.

macOS install

1. Download Azure Data Studio for macOS.
2. To expand the contents of the zip, double-select it.
3. To make Azure Data Studio available in the *Launchpad*, drag *Azure Data Studio.app* to the *Applications* folder.

Linux install

.deb Installation

1. Download Azure Data Studio for Linux by using the deb file.
2. To extract the file, open a new Terminal window and follow the below commands.

```
cd ~
sudo dpkg -i ./Downloads/azuredatastudio-linux-<version string>.deb
```

3. To launch Azure Data Studio

```
azuredatastudio
```

> **NOTE**
>
> You may have missing dependencies. Use the following commands to install these dependencies.
>
> ```
> sudo apt-get install libunwind8
> ```

.rpm Installation

1. Download Azure Data Studio for Linux by using the rpm file.
2. To extract the file, open a new Terminal window and follow the below commands.

```
cd ~
yum install ./Downloads/azuredatastudio-linux-<version string>.rpm
```

3. To launch Azure Data Studio

```
azuredatastudio
```

> **NOTE**
>
> You may have missing dependencies. Use the following commands to install these dependencies.
>
> ```
> yum install libXScrnSaver
> ```

tar.gz Installation

1. Download Azure Data Studio for Linux by using the .tar.gz file.
2. To extract the file, open a new Terminal window and follow the below commands.

   ```
   cd ~
   cp ~/Downloads/azuredatastudio-linux-<version string>.tar.gz ~
   tar -xvf ~/azuredatastudio-linux-<version string>.tar.gz
   echo 'export PATH="$PATH:~/azuredatastudio-linux-x64"' >> ~/.bashrc
   source ~/.bashrc
   ```

3. To launch Azure Data Studio

   ```
   azuredatastudio
   ```

> **NOTE**
>
> You may have missing dependencies. Use the following commands to install these dependencies.
>
> ```
> sudo apt-get install libxss1 libgconf-2-4 libunwind8
> ```

Windows Subsystem for Linux (WSL)

1. Install **Azure Data Studio** for Windows. Then use the `azuredatastudio` command in a WSL terminal just as you would in a standard command prompt. By default, the application is stored in your AppData folder.
2. Start **Azure Data Studio** from the WSL command prompt. When using the default Windows installation, the application can be started using:

   ```
   '/mnt/c/Users/<your user name>/AppData/Local/Programs/Azure Data Studio/azuredatastudio.exe'
   ```

What's new

For details about the latest release of Azure Data Studio, see **Release notes for Azure Data Studio**.

Download Insiders build of Azure Data Studio

It's recommended to download the GA release of Azure Data Studio. However, if you want to try out the beta features and send feedback, you can download the Insiders build of Azure Data Studio.

Supported operating systems

Azure Data Studio runs on Windows, macOS, and Linux and is supported on the following platforms:

Windows operating systems

- Windows 10 (64-bit)

- Windows 8.1 (64-bit)
- Windows 8 (64-bit)
- Windows 7 (SP1)
- Windows Server 2019
- Windows Server 2016
- Windows Server 2012 R2 (64-bit)
- Windows Server 2012 (64-bit)
- Windows Server 2008 R2 (64-bit)

macOS operating systems

- macOS 10.15 Catalina
- macOS 10.14 Mojave
- macOS 10.13 High Sierra
- macOS 10.12 Sierra
- macOS 11.1 Big Sur

Linux operating systems

- Red Hat Enterprise Linux (RHEL) 8.3
- Red Hat Enterprise Linux (RHEL) 8.2
- Red Hat Enterprise Linux (RHEL) 8.1
- Red Hat Enterprise Linux (RHEL) 8.0
- Red Hat Enterprise Linux (RHEL) 7.4
- Red Hat Enterprise Linux (RHEL) 7.3
- SUSE Linux Enterprise Server v12 SP2
- Ubuntu 20.04
- Ubuntu 18.04
- Ubuntu 16.04

> **NOTE**
>
> The versions of RHEL (7.3 and 7.4) are no longer supported by Red Hat. RHEL 7.3 support ended November 30, 2018, and RHEL 7.4 ended August 31, 2019.
>
> For details of the RHEL 7 Application Compatibility Guide, see: https://access.redhat.com/articles/rhel-abi-compatibility for more information.
>
> For details of the RHEL 8 Application Compatibility Guide, see: https://access.redhat.com/articles/rhel8-abi-compatibility.

Recommended System Requirements

RECOMMENDED/MINIMUM	CPU CORES	MEMORY/RAM
Recommended	4	8 GB
Minimum	2	4 GB

Check for updates

To check for the latest updates, select the gear icon on the bottom left of the window and select **Check for Updates**.

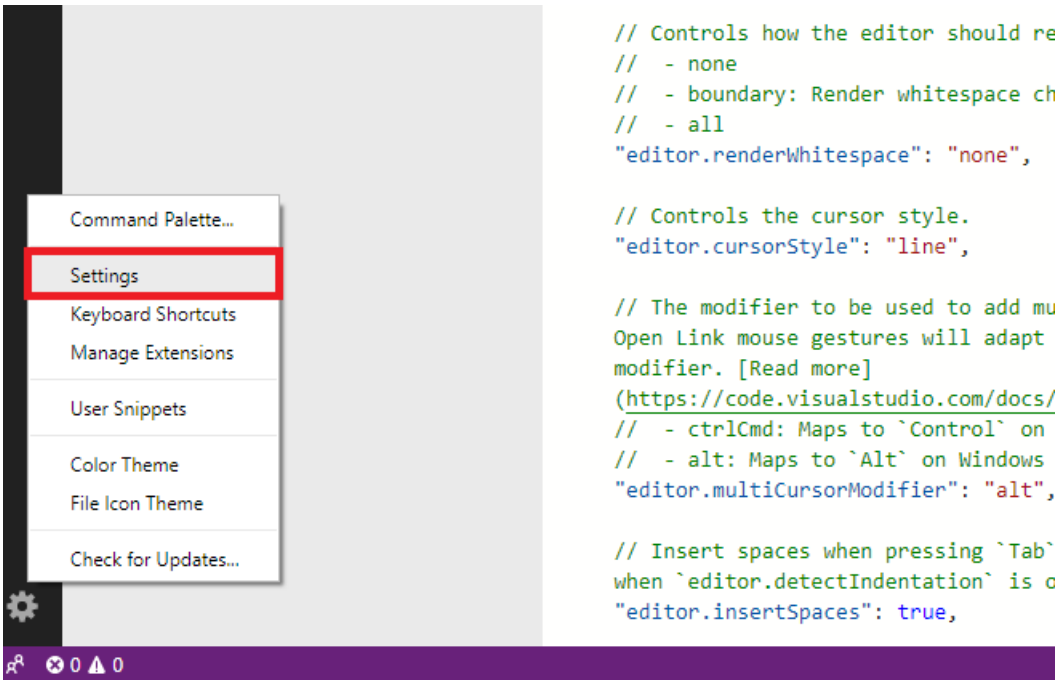
Offline environment updates can be applied by installing the latest version directly over a previously installed version. Uninstalling prior versions of Azure Data Studio isn't necessary. Instead, the installer updates a currently installed application if present.

Move user settings

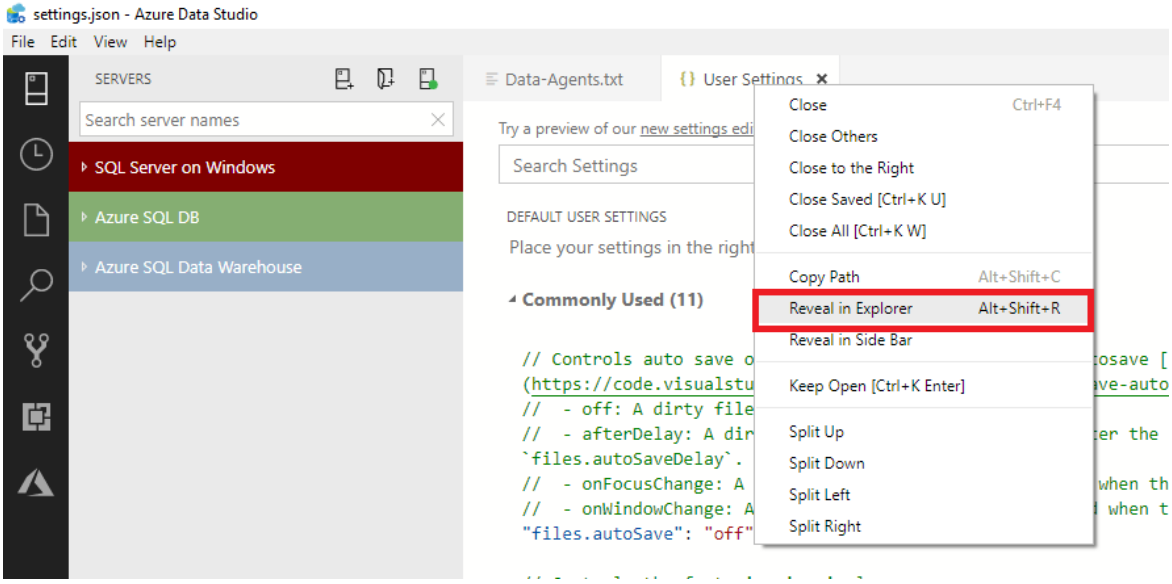
If you're updating SQL Operations Studio to Azure Data Studio and want to keep your settings, keyboard shortcuts, or code snippets, follow the steps below.

If you already have Azure Data Studio, or you've never installed or customized SQL Operations Studio, then you can ignore this section.

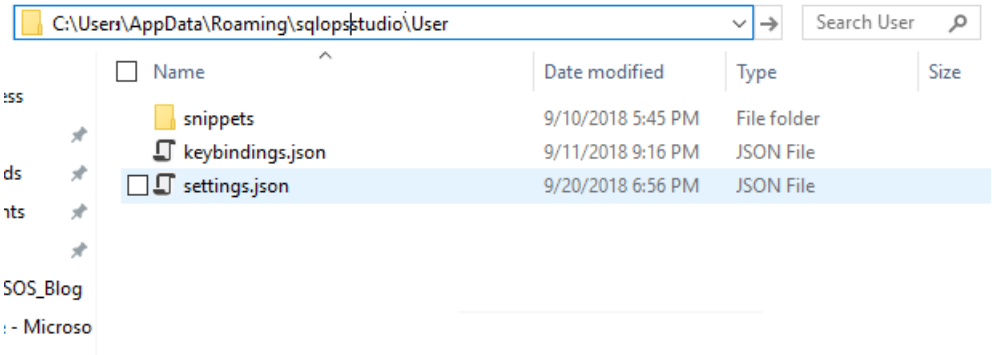
1. Open Settings by selecting the gear on the bottom left and selecting **Settings**.



2. Right-click the **User Settings** tab on top, and select **Reveal in Explorer**



3. Copy all files in this folder and save them in an easy-to-find location on your local drive, like your Documents folder.



4. In your new version of Azure Data Studio, follow steps 1-2, then for step 3, paste the contents you saved into the folder. You can also manually copy over the settings, key bindings, or snippets in their respective locations.

5. If overriding an existing installation, delete the old install directory before installation to avoid errors connecting to your Azure account for the resource explorer.

Uninstall Azure Data Studio from Windows

If you installed Azure Data Studio using the Windows Installer, then uninstall the same way you remove any Windows application.

If you installed Azure Data Studio with a .zip or other archive, then delete the files.

Uninstall Azure Data Studio from macOS

You can uninstall apps from the internet or disc on Mac by the below steps.

1. Select the **Finder icon** in the Dock, then select Applications in the Finder sidebar.

2. Then choose from one of te steps below:

   - If an app is in a folder, open the app's folder to check for an Uninstaller. If you see Uninstall [App] or [App] Uninstaller, double-select it, then follow the onscreen instructions.

   - If an app isn't in a folder or doesn't have an Uninstaller, drag the app from the Applications folder to the Trash (at the end of the Dock).

To uninstall apps, you downloaded from the App Store, use Launchpad.

Uninstall Azure Data Studio from Linux

Debian

You can uninstall Azure Data Studio under Debian or Ubuntu Linux.

To list installed software type:

```
dpkg --list
dpkg --list | less
dpkg --list | grep apache
```

To delete the software, enter:

```
sudo apt-get remove azuredatastudio
sudo apt-get remove apache
```

RedHat

Use rpm or yum command to delete Azure Data Studio.

To list the installed software type:

```
rpm -qa | less
rpm -qa azuredatastudio
yum list | less
yum list azuredatastudio
```

To get information about the azuredatastudio package, enter:

```
rpm -qa azuredatastudio
yum list azuredatastudio
```

To delete a package called azuredatastudio, enter:

```
rpm -e azuredatastudio
yum remove azuredatastudio
```

Next Steps

- Azure Data Studio release notes
- What is Azure Data Studio
- SQL tools
- Azure Data Architecture Guide
- SQL Server Blog

Get help for SQL tools

- All the ways to get help
- Submit an Azure Data Studio Git issue
- Contribute to Azure Data Studio
- SQL Client Tools Forum
- SQL Server Data Tools - MSDN forum
- Support options for business users

Contribute to SQL documentation

Did you know that you could edit the content yourself? If you do so, not only will our documentation improve, but you'll also be credited as a contributor to the page.

- How to contribute to SQL Server Documentation

Microsoft Privacy Statement and usage data collection.

Release notes for Azure Data Studio

6/28/2021 • 28 minutes to read • Edit Online

This article provides details about updates, improvements, and bug fixes for the current and previous versions of Azure Data Studio.

Current Azure Data Studio release

 **Download and install the latest release!**

June 2021

Azure Data Studio 1.30.0 is the latest general availability (GA) release.

- Release number: 1.30.0
- Release date: June 17, 2021

What's new in 1.30.0

NEW ITEM	DETAILS
Results Grid	Added filtering/sorting feature for query result grid in query editor and notebook, the feature can be invoked from the column headers. note that this feature is only available when you enable the preview features.
Results Grid	Added a status bar item to show summary of the selected cells if there are multiple numeric values
Notebooks	Added new book icon
Notebooks	Notebook URI Handler File Support
Python	Updated Python to 3.8.10

Bug fixes in 1.30.0

NEW ITEM	DETAILS
Notebooks	Fixed WYWIWYG Table cell adding new line in table cell
Notebooks	Fixed issue that Kusto notebook does not change kernels properly

For a full list of bug fixes addressed for the May 2021 release, visit the bugs and issues list on GitHub.

Known issues in 1.30.0

For a list of the current known issues, visit the issues list on GitHub.

Azure Data Studio feedback

You can reference Azure Data Studio feedback for other known issues and to provide feedback to the product team.

Previous Azure Data Studio releases and updates

AZURE DATA STUDIO RELEASE	BUILD NUMBER	RELEASE DATE	HOTFIX
May 2021	1.29.0	May 19, 2021	N/A
April 2021	1.28.0	April 15, 2021	N/A
March 2021	1.27.0	March 17, 2021	N/A
February 2021	1.26.0	February 18, 2021	N/A
December 2020	1.27.0	December 9, 2020	hotfix
November 2020	1.24.0	November 12, 2020	N/A
October 2020	1.23.0	October 14, 2020	N/A
September 2020	1.22.0	September 22, 2020	hotfix
August 2020	1.21.0	August 12, 2020	N/A
July 2020	1.20.0	July 15, 2020	hotfix
June 2020	1.19.0	June 15, 2020	N/A
May 2020	1.18.0	May 20, 2020	hotfix
April 2020	1.17.0	April 27, 2020	hotfix
March 2020	1.16.0	March 18, 2020	N/A
February 2020	1.15.0	February 13, 2020	hotfix
December 2019	1.14.0	December 19, 2019	hotfix
November 2019	1.13.0	November 4, 2019	hotfix
October 2019	1.12.0	October 2, 2019	hotfix 1 hotfix 2
September 2019	1.11.0	September 10, 2019	N/A
August 2019	1.10.0	August 15, 2019	N/A
July 2019	1.9.0	July 11, 2019	N/A
June 2019	1.8.0	June 6, 2019	N/A
May 2019	1.7.0	May 8, 2019	N/A
April 2019	1.6.0	April 18, 2019	N/A

AZURE DATA STUDIO RELEASE	BUILD NUMBER	RELEASE DATE	HOTFIX
March 2019	1.5.1	March 18, 2019	hotfix
February 2019	1.4.5	February 13, 2019	N/A
January 2019	1.3.8	January 09, 2019	hotfix

Download the previous release of Azure Data Studio.

> **NOTE**
> All previous versions of Azure Data Studio are not supported.

May 2021

Azure Data Studio 1.29.0 is the latest general availability (GA) release.

- Release number: 1.29.0
- Release date: May 19, 2021

What's new in 1.29.0

NEW ITEM	DETAILS
Notebooks	Added run with parameters action. Learn more here

Bug fixes in 1.29.0

NEW ITEM	DETAILS
Database	Fixed an issue with the title not changing when the database connection changes.
Extensions	Fixed an issue with no longer prompting the user to install 3rd party extensions.
General Azure Data Studio	Fixed an issue with the account button in the sidebar getting stuck loading.
General Azure Data Studio	Fixed an issue with the *Run With Parameters* silently failing if the parameters cell is empty or invalid.
General Azure Data Studio	Fixed an issue with the *Run With Parameters* not handling multiple parameters on the same line.
General Azure Data Studio	Fixed an issue with being unable to connect to Azure.
General Azure Data Studio	Fixed an issue with the wrong line number showing up in the output.
General Azure Data Studio	Fixed an issue when receiving an error when connecting to an Azure server in the Azure viewlet.
General Azure Data Studio	Fixed an issue with the loading spinner not being animated.

NEW ITEM	DETAILS
General Azure Data Studio	Fixed an issue with the links inserted in the split view/MD view not being inserted as a list item.
General Azure Data Studio	Fixed an issue with the *Issue Reporter* being blank.
Extensions	Fixed an issue with the extensions view having many filter options that aren't applicable to Azure Data Studio.
General Azure Data Studio	Fixed an issue with getting the Azure subscriptions API failing across Azure Data Studio.
General Azure Data Studio	Fixed an issue with the wrong event prefix.
General Azure Data Studio	Fixed an issue with converting the default tables from Excel, Word, and OneNote into markdown tables.
Notebooks	Fixed an issue with selecting notebooks from the notebooks viewlet recentering the viewlet vertically.
Notebooks	Fixed an issue with not connecting to SQL Server from SQL Notebook.
Notebooks	Fixed an issue with the notebook icons being sized incorrectly.
PowerShell	Fixed an issue with using the Cloud Shell (PowerShell).
SQL Database Project	Fixed an issue with showing an unnecessary horizontal scrollbar in the *create project from database* dialog SQL Database Project dashboard update.

For a full list of bug fixes addressed for the May 2021 release, visit the bugs and issues list on GitHub.

Known issues in 1.29.0

For a list of the current known issues, visit the issues list on GitHub.

April 2021

April 15, 2021 / version: 1.28.0

What's new in 1.28.0

NEW ITEM	DETAILS
Extension update	Kusto (KQL)
Extension update	MachineLearning
Extension update	SchemaCompare
Extension update	SQLDatabaseProjects
Notebook features	Added *Add Notebook* and *Remove Notebook commands*

Bug fixes in 1.28.0

For the list of the bug fixes addressed for the April 2021 release, visit the bugs and issues list on GitHub.

Known issues in 1.28.0

For the list of known issues, visit the issues list on GitHub.

March 2021

March 17, 2021 / version: 1.27.0

CHANGE	DETAILS
Bug Fixes	For a complete list of fixes, see Bugs and issues on GitHub.
Extension(s) update	Dacpac SQLDatabaseProjects
New Notebook features	Added create book dialog

February 2021

February 18, 2021 / version: 1.26.0

CHANGE	DETAILS
Bug Fixes	For a complete list of fixes see Bugs and issues on GitHub.
Extension(s) update	Dacpac Kusto (KQL) MachineLearning Profiler SchemaCompare SQLDatabaseProjects
New Azure Arc features	Multiple data controllers now supported New connection dialog options like the *kube config file* Postgres dashboard enhancements
New Notebook features	Improved Jupyter server start-up time by 50% on Windows Added support to edit Jupyter Books through right-click Added URI notebook parameterization support and added notebook parameterization documentation

December 2020 (hotfix)

February 10, 2021 / version: 1.25.3

CHANGE	DETAILS
Fix bug #13899	Scrolling to the appropriate cross-reference links in Notebooks
Upgrade Electron to incorporate important bug fixes	N/A

December 2020

December 9, 2020 / version: 1.25.0

CHANGE	DETAILS
Bug Fixes	For a complete list of fixes see Bugs and issues on GitHub.
Database Projects extension update	Added workspaces and improved sidebar

November 2020

November 12, 2020 / version: 1.24.0

CHANGE	DETAILS
Bug Fixes	For a complete list of fixes see Bugs and issues on GitHub.
Connection dialog	Added new browse tab for connection dialog.
Extension(s) update	Released update to Postgres extension.
New notebook features	Added new features to SQL to notebook support. Added new features to Notebook parameterization support. Added new features to results streaming for SQL Notebooks.
Python installation	PROSE package has been removed from default Python installation.

Known issues (1.24.0)

NEW ITEM	DETAILS	WORKAROUND
Azure Arc extension	Known Issue: The "Script to Notebook" button for Arc MIAA & PG deployments doesn't do field validation before scripting the notebook. This means that if users enter the password wrong in the password confirm inputs then they may end up with a notebook that has the wrong value for the password.	The "Deploy" button works as expected though so users should use that instead.
Object Explorer	Releases of Azure Data Studio before 1.24.0 have a breaking change in object explorer because of the engine's changes related to Azure Synapse Analytics serverless SQL pool.	To continue utilizing object explorer in Azure Data Studio with Azure Synapse Analytics serverless SQL pool, you need to use Azure Data Studio 1.24.0 or later.

You can reference Azure Data Studio feedback for other known issues and to provide feedback to the product team.

October 2020

October 14, 2020 / version: 1.23.0

CHANGE	DETAILS
Azure SQL Edge	Support for Azure SQL Edge objects.
Bug Fixes	For a complete list of fixes see Bugs and issues on GitHub.
Databases	Support for same database reference.
Extension updates	Azure Arc azdata Machine Learning Kusto (KQL) Schema Compare SQL Assessment SQL Database Projects SQL Server Import
New deployment features	Added Azure SQL DB and VM deployments.
PowerShell	Added PowerShell kernel results streaming support.

September 2020 (hotfix)

September 30, 2020 / version: 1.22.1

CHANGE	DETAILS
Resolved bugs and issues	For a complete list of fixes see Bugs and issues on GitHub.

September 2020

September 22, 2020 / version: 1.22.0

CHANGE	DETAILS
New notebook features	• Supports brand new text cell editing experience based on rich text formatting and seamless conversion to markdown, also known as WYSIWYG toolbar (What You See Is What You Get) • Supports Kusto kernel • Supports pinning of notebooks • Added support for new version of Jupyter Books • Improved Jupyter Shortcuts • Introduced perf loading improvements
SQL Database Projects extension	The SQL Database Projects extension brings project-based database development to Azure Data Studio. In this preview release, SQL projects can be created and published from Azure Data Studio.

CHANGE	DETAILS
Kusto (KQL) extension	Brings native Kusto experiences in Azure Data Studio for data exploration and data analytics against massive amount of real-time streaming data stored in Azure Data Explorer. This preview release supports connecting and browsing Azure Data Explorer clusters, writing KQL queries and authoring notebooks with Kusto kernel.
Azure Arc extension	Users can try out Azure Arc public preview through Azure Data Studio. This includes: • Deploy data controller • Deploy Postgres • Deploy Managed Instance for Azure Arc • Connect to data controller • Access data service dashboards • Azure Arc Jupyter Book
Deployment options	• Azure SQL Database Edge (Edge will require Azure SQL Edge Deployment Extension)
SQL Server Import extension GA	Announcing the GA of the SQL Server Import extension, features no longer in preview. This extension facilitates importing csv/txt files. Learn more about the extension in this article.
Resolved bugs and issues	For a complete list of fixes see Bugs and issues, on GitHub.

August 2020

August 12, 2020 / version: 1.21.0

CHANGE	DETAILS
New notebook features	• Move cell locations • Convert cells to Text Cell or Code cell
Jupyter Books picker	Users can now choose Jupyter Books from GitHub releases and open seamlessly in Azure Data Studio
Search added to Notebooks Viewlet	Users can easily search through content across their notebooks and Jupyter Books
Resolved bugs and issues	For a complete list of fixes see Bugs and issues, on GitHub.

July 2020 (hotfix)

July 17, 2020 / version: 1.20.1

CHANGE	DETAILS
Fix bug #11372 Object Explorer drag-and-drop table incorrectly wraps table names	#11372
Fix bug #11356 Dark theme is now the default theme	#11356

Known Issue

- Some users have reported connection errors from the new Microsoft.Data.SqlClient v2.0.0 included in this release. Users have found following these instructions to successfully connect

July 2020

July 15, 2020 / version: 1.20.0

CHANGE	DETAILS
Added new Feature Tour	From welcome page and command palette, users can now launch a feature tour to get a walkthrough of commonly used features including Connections Viewlet, Notebooks viewlet, and Extensions Marketplace
New notebook features	• Header support in Markdown Toolbar • Side-by-side Markdown Preview in Text Cells
Drag and Drop Columns and Tables in Query Editor	Users can now directly drag and drop columns and tables from connections viewlet to query editor
Added Azure Account icon to Activity Bar	Users can now easily see where to sign in to Azure
Resolved bugs and issues	For a complete list of fixes see Bugs and issues, on GitHub.

June 2020

June 15, 2020 / version: 1.19.0

CHANGE	DETAILS
Added Azure Data Studio to Azure portal Integration	Users can now directly launch to Azure portal from an Azure SQL Database connection, Azure Postgres, and more.
New notebook features	• New Notebook toolbar • New Edit Cell toolbar • Python dependencies wizard UX updates • Improved spacing across notebooks

CHANGE	DETAILS
Announcing SQL Assessment API extension	This extension adds SQL Server best-practice assessment in Azure Data Studio. It exposes SQL Assessment API, which was previously available for use in PowerShell SqlServer module and SMO only, to let you evaluate your SQL Server instances and receive for them recommendations by SQL Server Team. Learn more about SQL Assessment API and what it's capable of in this article.
Machine Learning Extension improvements	Now supports Azure SQL Managed Instance.
Data Virtualization extension improvements	Now supports MongoDB and Teradata
Postgres extension bug fixes	Fixed Azure MFA
Resolved bugs and issues	For a complete list of fixes see Bugs and issues, on GitHub.

May 2020 (hotfix)

May 27, 2020 / version: 1.18.1

CHANGE	DETAILS
Fix bug #10538 "Run Current Query" keybind no longer behaving as expected	#10538
Fix bug #10537 Unable to open new or existing sql files on v1.18	#10537

May 2020

May 20, 2020 / version: 1.18.0

CHANGE	DETAILS
Announcing Redgate SQL Prompt extension	This extension lets you manage formatting styles directly within Azure Data Studio, so you can create and edit your styles without leaving the IDE.
Announcing Machine Learning Extension	This extension enables you to: • Manage Python and R packages with SQL Server machine learning services with Azure Data Studio. • Use ONNX model to make predictions in Azure SQL Edge. • View ONNX models in an Azure SQL Edge database. • Import ONNX models from a file or Azure Machine Learning into Azure SQL Edge database. • Create a notebook to run experiments.

CHANGE	DETAILS
New notebook features	• Added new Python Dependencies Wizard to make it easier to install Python dependencies • Added underline support for Markdown Toolbar
Parameterization for Always Encrypted	Allows you to run queries that insert, update, or filter by encrypted database columns.
Resolved bugs and issues	For a complete list of fixes see Bugs and issues, on GitHub.

April 2020 (hotfix)

April 30, 2020 / version: 1.17.1

CHANGE	DETAILS
Fix bug #10197 Can't connect via MFA	#10197

April 2020

April 27, 2020 / version: 1.17.0

CHANGE	DETAILS
Improved welcome page	UI update on the welcome page to make it easier to see common actions and highlighting extensions.
New notebook features	• Added Markdown toolbar when editing text cells to help write with Markdown • Revamped Jupyter Books viewlet to become a Notebooks viewlet where you can manage Jupyter Books and notebooks together • Added support for persisting charts when saving a notebook • Added support for KQL magic in Python notebooks
Improved dashboards	Dashboards throughout Azure Data Studio have been updated with latest design patterns, including an actions toolbar. This also applies to many extensions.
Added Cloud Shell integration in the Azure view.	
Support for Always Encrypted and Always Encrypted with secure enclaves.	
Resolved bugs and issues	For a complete list of fixes see Bugs and issues, on GitHub.

CHANGE	DETAILS
Resolved bugs and issues	For a complete list of fixes see Bugs and issues, on GitHub.

March 2020

March 18, 2020 / version: 1.16.0

CHANGE	DETAILS
Added charting support in SQL Notebooks	When running a SQL query in a code cell, users can now create and save charts.
Added Create Jupyter Book experience	Users can now create their own Jupyter Books using a notebook.
Added Azure AD support for Postgres extension	
Fixed many accessibility bugs	List of accessibility bugs
VS Code merges to 1.42	This release includes updates to VS Code from the 3 previous VS Code releases. Read their release notes to learn more.
Resolved bugs and issues	For a complete list of fixes see Bugs and issues, on GitHub.

February (hotfix)

February 19, 2020 / version: 1.15.1

CHANGE	DETAILS
Fix bug #9149 Show active connections	#9149
Fix bug #9061 Edit Data grid doesn't properly resize when showing or hiding SQL Pane	#9061

February 2020

February 13, 2020 / version: 1.15.0

CHANGE	DETAILS

CHANGE	DETAILS
New Azure Sign-in improvement	Added improved Azure Sign-in experience, including removal of copy/paste of device code to make a more seamless connected experience.
Find in Notebook support	Users can now use Ctrl+F in a notebook. Find in Notebook support searches line by line through both code and text cells.
VS Code merges from 1.38 to 1.42	This release includes updates to VS Code from the 3 previous VS Code releases. Read their release notes to learn more.
Fix for the "white/blank screen" issue reported by many users.	
Resolved bugs and issues	For a complete list of fixes see Bugs and issues, on GitHub.

Known Issue

- Users on macOS Catalina will need to right-click Azure Data Studio and then click open.

December 2019 (hotfix)

December 26, 2019 / version: 1.14.1

CHANGE	DETAILS
Fix bug #8747 OE Expansion fails	#8747

December 2019

December 19, 2019 / version: 1.14.0

CHANGE	DETAILS
Changed attach to connection dropdown in Notebooks to only list the currently active connection	#8129
Added bigdatacluster.ignoreSslVerification setting to allow ignoring TLS/SSL verification errors when connecting to a BDC	#8582
Allow changing default language flavor for offline query editors	#8419
GA status for Big Data Cluster/SQL 2019 features	#8269
Resolved bugs and issues	For a complete list of fixes see Bugs and issues, on GitHub.

CHANGE	DETAILS

November 2019 (hotfix)

November 15, 2019 / version: 1.13.1

CHANGE	DETAILS
Fix bug #8210 Copy/Paste results are out of order	

November 2019

November 4, 2019 / version: 1.13.0

CHANGE	DETAILS
New SQL Server 2019 support	• Deploy SQL Server 2019 big data cluster with BDC Deploy wizard • Manage cluster health with controller dashboard • Manage HDFS access control lists using Security ACLs Dialog • Add mounts using HDFS Tiering Dialog • Troubleshoot using built-in Jupyter Book, SQL Server 2019 guide • Renamed to SQL vNext extension Data virtualization extension • Added Teradata and Mongo support in External Table Wizard
New notebook features	• Announcing PowerShell notebooks • Announcing collapsible code cells • Perf improvements in Notebooks • View the full list of improvements here
Announcing Jupyter Books	Jupyter Books are a collection of notebooks and markdown files organized in a table of contents.
New SQL Server Deploy wizard	Now includes support for deploying: • SQL Server 2019 on Windows • SQL Server 2017 on Windows • SQL Server 2019 on Docker • SQL Server 2017 on Docker
Announcing GA of Schema Compare extension	• SQLCMD mode • Localization support • Accessibility fixes • Security bugs

CHANGE	DETAILS
Announcing GA of SQL Server Dacpac extension	• Localization support • Accessibility fixes • Security bugs
Announcing Visual Studio IntelliCode extension	Visual Studio IntelliCode now supports SQL, which allows for smarter suggestions of reserved keywords.
Resolved bugs and issues	For a complete list of fixes see Bugs and issues, on GitHub.

October 2019 (hotfix 2)

October 11, 2019 / version: 1.12.2

CHANGE	DETAILS
Disable automatically starting the EH in inspect mode	

October 2019 (hotfix)

October 8, 2019 / version: 1.12.1

CHANGE	DETAILS
Fixed issue for quotes and backslashes in Notebooks to escape correctly.	

October 2019

October 2, 2019 / version: 1.12.0

CHANGE	DETAILS
Release of Query History extension	The SQL History extension saves all past queries executed in an Azure Data Studio session and lists them in execution order. Users can see open the query, execute the query, delete the query, pause query history, or delete all query history entries.
New Copy/Paste Results	We have added more ways to copy/paste results from the results grid.
Update to PowerShell extension	

CHANGE	DETAILS
Resolved bugs and issues	For a complete list of fixes see Bugs and issues, on GitHub.

Known Issues

- Notebooks
  - 7080 Rare Case when Notebook is Serialized Incorrectly

September 2019

September 10, 2019 / version: 1.11.0

CHANGE	DETAILS
Enable SQLCMD Mode	Query editor now supports toggling of SQLCMD mode to write and edit queries as SQLCMD scripts
Community Extension: Query Editor Boost	Query Editor Boost is an open-source extension focused on enhancing the Azure Data Studio query editor for users who are frequently writing queries. • Save the current query as a snippet • Switch databases using Ctrl+U • New Query from template • View the full list of improvements here
Notebook Improvements	• Performance improvements for supporting larger notebook files • View the full list of improvements here
Visual Studio Code August Release Merge 1.38	Latest improvements can be found here.
Resolved bugs and issues	For a complete list of fixes see Bugs and issues, on GitHub.

Known Issues

- Notebooks
  - 7080 Rare Case when Notebook is Serialized Incorrectly

August 2019

August 15, 2019 / version: 1.10.0

CHANGE	DETAILS
Release of SandDance 1.3.1 extension	• Smart chart detection • 3D Visualizations • Data filtering

CHANGE	DETAILS
Notebook Improvements	• Add code or text cell in-line • Added ability to right-click SQL results grid to save result as CSV, JSON, etc. • Improvement to notebook loading performance for loading JSON faster • View the full list of improvements here
SQL Server 2019 Support	This release includes support for extra SQL Server 2019 Big Data Cluster features including: • Reduced time taken to load table and column information on the object-mapping page. • Fixed a bug with loading existing database scoped credentials on the connection details page. • Increased default sample size used for PROSE parsing.
Dacpac extension now supports Azure AD	
Visual Studio Code July Release Merge 1.37	Latest improvements can be found here.
Resolved bugs and issues	For a complete list of fixes see Bugs and issues, on GitHub.

July 2019

July 11, 2019 / version: 1.9.0

CHANGE	DETAILS
Release of SentryOne Plan Explorer extension	Our valued Microsoft partner, SentryOne, will be shipping their SentryOne Plan Explorer extension for Azure Data Studio. This is a free extension, which provides enhanced plan diagrams for queries run in Azure Data Studio, with optimized layout algorithms and intuitive color-coding to help quickly identify the most expensive operators affecting query performance. To learn more about the extension, check out SentryOne's blog post here.
New Features coming to Schema Compare	• Schema Compare File Support (.SCMP) • Cancel Schema Compare Support • Complete changes can be found here
Notebook Improvements	• Plotly Python Support • Open Notebook from Browser • Python Package Management Dialog • Performance and Markdown Enhancements • Keyboard Shortcuts Update • Bug Fixes and Minor Features can be found here

CHANGE	DETAILS
SQL Server 2019 Support	This release includes support for extra SQL Server 2019 Big Data Cluster features including: • Service Endpoints table within the Management Dashboard that lists all key services in the cluster. • Cluster Status Notebook shows how you can query & troubleshoot cluster status across all services and pods.
Updated Language Packs Available	There are now 10 language packs available in the Extension Manager marketplace. Simply, search for the specific language using the extension marketplace and install. Once you install the selected language, Azure Data Studio will prompt you to restart with the new language.
SQL Server Profiler Update	The SQL Server Profile extension has been updated to include new features including: • Filtering by Database Name • Copy & Paste Support • Save/Load Filter A full list of improvements for SQL Server Profiler Extension can be found here.
Visual Studio Code May Release Merge 1.35	Latest improvements can be found here.
Resolved bugs and issues	In previous releases of Azure Data Studio, if a user database was selected when connecting from the Connection dialog, the resulting Object Explorer entry was scoped entirely to that single database. Beginning in this release, that behavior is being changed so that server level properties are also shown in the object explorer. For a complete list of fixes see Bugs and issues, on GitHub.

June 2019

June 6, 2019 / version: 1.8.0

CHANGE	DETAILS
Release of Central Management Servers (CMS) extension	Central Management Servers store a list of instances of SQL Server that is organized into one or more central management server groups. Users can connect to their own existing CMS servers and manage their servers like adding and removing servers. To learn more, you can read here
Release of Database Administration Tool Extensions for Windows	This extension launches two of the most used experiences in SQL Server Management Studio from Azure Data Studio. Users can right-click on many different objects (such as Databases, Tables, Columns, Views, and more) and select Properties to view the SSMS Properties Dialog for that object. In addition, users can right-click on a database and select Generate Scripts to launch the well-known SSMS Generate Scripts Wizard.

CHANGE	DETAILS
Schema Compare Improvements	• Added Exclude/Include Options • Generate Script opens script after being generated • Removed double scroll bars • Formatting and layout improvements • Complete changes can be found here
Moved Messages section to own tab	When users ran SQL queries, results and messages were on stacked panels. Now they are in separate tabs in one panel like in SSMS.
SQL Notebook Improvements	• Users can now choose to use their own Python 3 or Anaconda installs in notebooks • Multiple Stabilities + fit/finish fixes • View the full list of improvements here
Visual Studio Code April Release Merge 1.34	Latest improvements can be found here
Resolved bugs and issues.	See Bugs and issues, on GitHub.

Known Issues

- Database Administration Tool Extensions for Windows
  - Can't launch properties from disconnected server node
  - Can't launch properties for Azure servers
  - Not all objects have property dialogs
  - Dialogs take a long time to start up
  - Errors launching servers with some types of connections (such as Azure AD)
- Notebooks
  - 5838 Allow users to use system Python for Notebooks
- Schema Compare
  - 5804 Schema Compare tasks show default cancel context menu, which doesn'thing

May 2019

May 8, 2019 / version: 1.7.0

CHANGE	DETAILS
Release of Schema Compare extension	Schema Compare is a well-known feature in SQL Server Data Tools (SSDT), and its primary use case is to compare and visualize the differences between databases and .dacpac files and to execute actions to make them the same.
Moved Task view to Output Window	Users can now view the status of long running tasks like Backup, Restore, and Schema Compare in the Task view in Output window
Added Welcome page	• Links to common actions like New Query, New File, New Notebook • Links to documentation and GitHub

CHANGE	DETAILS
SQL Notebook Improvements	• Markdown rendering improvements, including better support for notes and tables • Usability improvements to the toolbar • Markdown links for trusted notebooks no longer require Cmd/Ctrl + click and can be clicked directly • Improvements in cleaning up Jupyter processes after closing notebooks and reducing errors when starting multiple notebooks concurrently • Improvements to SQL notebook connections to ensure errors don't occur when running 2 notebooks against the same database • Improvements to notebook autoscrolling to the currently executing cell when clicking the Run Cells button from the toolbar • General stability and performance improvements
Resolved bugs and issues.	See Bugs and issues, on GitHub.

April 2019

April 18, 2019 / version: 1.6.0

CHANGE	DETAILS
Renamed **Servers** tab to **Connections**	
Moved Azure Resource Explorer as an Azure viewlet under Connections	Users can now view their Azure SQL instances through Azure viewlet in the Connections view and expand to view objects under each server or database.
SQL Notebook Improvements	• Added button on toolbar to clear output for all cells • Added button on toolbar to run all cells • Fixed connection name instead of server name (if set) in the Attach to dropdown • Fix for images in markdown not rendering when using relative image paths • Improved functionality in notebook grids by adding double-click auto resize column size and improved mousewheel support • Improvements to error handling and python install resiliency when installing python through notebooks • Improvements to "select all" functionality when selecting notebook cells • Improvements to notebook connections to prevent closing a notebook and impacting an object explorer connection • Improved notebook experience to display a message to the user when notebook disconnected and needs a connection to run cells • Improved support for unsaved notebooks to rehydrate in Azure Data Studio when Azure Data Studio is started again
Resolved bugs and issues.	See Bugs and issues, on GitHub.

CHANGE	DETAILS

March 2019 (Hotfix)

March 22, 2019 / version: 1.5.2 / Hotfix release

CHANGE	DETAILS
Fixed a few issues discovered in 1.5.1.	See March Hotfix Release, on GitHub. • Fixed issue where user couldn't close notebook opened from the "Open Notebook" task in the Dashboard • Fixed issue where Notebook JSON has extra } after save • Fixed issue where notebook grids weren't responding to theme changes • Fixed issue where full notebook path was shown in the tab header. Now only the filename is shown.

March 2019

March 18, 2019 / version: 1.5.1

CHANGE	DETAILS
Added PostgreSQL extension for Azure Data Studio	Supported features: • Connection Dialog • Object Explorer • Query Editor • Charting • Dashboards • Snippets • Edit Data • Notebooks
Added SQL Notebooks	Added SQL Kernel support to built-in Notebook viewer: • Supports T-SQL • Support PGSQL
Added PowerShell Extension	Brings over the PowerShell extension experience from VS Code.
Added SQL Server dacpac extension	Removes Data-Tier Application Wizard from SQL Server Import extension into a new extension.
Added Community extension QueryPlan.show	Adds integration support to visualize query plans
Updated SQL Server 2019 Preview extension	• Jupyter Notebook support, specifically Python3, and Spark kernels, have moved into the core Azure Data Studio tool. • Bug fixes to External Data Wizard

CHANGE	DETAILS
Resolved bugs and issues.	See Bugs and issues, on GitHub.

Known Issues

- #4427: Clicking Run on Cell Before Kernel is Ready for Spark Results in Fatal Error **Workaround:** Wait until kernels are loaded until running any cells
- #4493: Azure Data Studio launched from SSMS using SQL auth - prompts user for password **Workaround:** Use Windows Auth for now.
- #4494: Unable to install SQL notebook feature
  Workaround: Follow workaround steps here.
- #4503: Azure Data Studio can't be Opened Directly from DownloAzure Data Studio Folder (Mac)
  Workaround: Restart computer after unzipping the app. Will be investigated.
- #4539: Notebook Save As loses connection context
  Workaround: Will be fixed in next release.
- #4458: Dacpac Extract crashes SqlToolsService if invalid version is used
  Workaround: Restart Azure Data Studio and ensure correct version is used.
- New Notebook and Open Notebook icons are lost
  Workaround: The legacy connection type is deprecated. We recommend connecting to the SQL Server endpoint and you'll get all the actions (New Notebook, Spark Job) as expected.

February 2019

February 13, 2019 / version: 1.4.5

CHANGE	DETAILS
Added **Admin pack for SQL Server** extension pack.	This makes it easier to install SQL Server admin-related extensions. This includes: • SQL Server Agent • SQL Server Profiler • SQL Server Import
Added filtering extended event support in Profiler extension.	
Added Save as XML feature that can save T-SQL results as XML.	
Added Data-Tier Application Wizard improvements.	• Added Generate script button • Added view to give warnings of possible data loss during deployment.
Updates to the SQL Server 2019 Preview extension.	See Data Virtualization extension.
Results streaming enabled by default for long running queries.	
Resolved bugs and issues.	See Bugs and issues, on GitHub.

January 2019 (Hotfix)

January 16, 2019 / version: 1.3.9 / Hotfix release

CHANGE	DETAILS
Fixed a few issues discovered in 1.3.8.	See January Hotfix Release, on GitHub. For detailed information, see: • Change Log, on GitHub. • Releases, on GitHub.

January 2019

January 09, 2019 / version: 1.3.8

CHANGE	DETAILS
Added a new user installer for Windows.	Unlike the existing system installer, the new user installer doesn't require administrator privileges. This also enables an easier upgrade experience for non-administrators.
Added Azure Active Directory Authentication support.	
Announcing Idera SQL DM Performance Insights (preview).	
Data-Tier Application Wizard support in SQL Server Import extension.	
Update to the SQL Server 2019 Preview extension.	See Data Virtualization extension.
SQL Server Profiler improvements.	
Results Streaming for large queries (preview).	
Community extensions: sp_executesql to sql and New Database.	
Resolved bugs and issues.	See Bugs and issues, on GitHub.

Next steps

See one of the following quickstarts to get started:

- Download Azure Data Studio
- Connect & Query SQL Server
- Connect & Query Azure SQL Database
- Connect & Query Azure Synapse Analytics

Contribute to Azure Data Studio:

- https://github.com/Microsoft/azuredatastudio

What is Azure Data Studio?

3/8/2021 • 4 minutes to read • Edit Online

Azure Data Studio is a cross-platform database tool for data professionals using on-premises and cloud data platforms on Windows, macOS, and Linux.

Azure Data Studio offers a modern editor experience with IntelliSense, code snippets, source control integration, and an integrated terminal. It's engineered with the data platform user in mind, with built-in charting of query result sets and customizable dashboards.

The source code for Azure Data Studio and its data providers is available on GitHub under a source code EULA that provides rights to modify and use the software, but not to redistribute it or host it in a cloud service. For more information, see Azure Data Studio FAQ.

Download and Install Azure Data Studio

SQL code editor with IntelliSense

Azure Data Studio offers a modern, keyboard-focused SQL coding experience that makes your everyday tasks easier with built-in features, such as multiple tab windows, a rich SQL editor, IntelliSense, keyword completion, code snippets, code navigation, and source control integration (Git). Run on-demand SQL queries, view and save results as text, JSON, or Excel. Edit data, organize your favorite database connections, and browse database objects in a familiar object browsing experience. To learn how to use the SQL editor, see Use the SQL editor to create database objects.

Smart SQL code snippets

SQL code snippets generate the proper SQL syntax to create databases, tables, views, stored procedures, users, logins, roles, and to update existing database objects. Use smart snippets to quickly create copies of your database for development or testing purposes, and to generate and execute CREATE and INSERT scripts.

Azure Data Studio also provides functionality to create custom SQL code snippets. To learn more, see Create and use code snippets.

Customizable Server and Database Dashboards

Create rich customizable dashboards to monitor and quickly troubleshoot performance bottlenecks in your databases. To learn about insight widgets, and database (and server) dashboards, see Manage servers and databases with insight widgets.

Connection management (server groups)

Server groups provide a way to organize connection information for the servers and databases you work with. For details, see Server groups.

Integrated Terminal

Use your favorite command-line tools (for example, Bash, PowerShell, sqlcmd, bcp, and ssh) in the Integrated Terminal window right within the Azure Data Studio user interface. To learn about the integrated terminal, see Integrated terminal.

Extensibility and extension authoring

Enhance the Azure Data Studio experience by extending the functionality of the base installation. Azure Data Studio provides extensibility points for data management activities, and support for extension authoring.

To learn about extensibility in Azure Data Studio, see Extensibility. To learn about authoring extensions, see Extension authoring.

Feature comparison with SQL Server Management Studio (SSMS)

Use Azure Data Studio if you:

- Are mostly editing or executing queries.
- Need the ability to quickly chart and visualize result sets.
- Can execute most administrative tasks via the integrated terminal using sqlcmd or PowerShell.
- Have minimal need for wizard experiences.
- Do not need to do deep administrative or platform related configuration.
- Need to run on macOS or Linux.

Use SQL Server Management Studio if you:

- Are doing complex administrative or platform configuration.
- Are doing security management, including user management, vulnerability assessment, and configuration of security features.
- Need to make use of performance tuning advisors and dashboards.
- Use database diagrams and table designers.
- Need access to Registered Servers.
- Make use of live query stats or client statistics.

Shell features

FEATURE	AZURE DATA STUDIO	SSMS
Azure Sign-In	Yes	Yes
Dashboard	Yes	
Extensions	Yes	
Integrated Terminal	Yes	
Object Explorer	Yes	Yes
Object Scripting	Yes	Yes
Project System	Yes	
Select from Table	Yes	Yes
Source Code Control	Yes	
Task Pane	Yes	

FEATURE	AZURE DATA STUDIO	SSMS
Themes, including Dark Mode	Yes	
Azure Resource Explorer	Preview	
Generate Scripts Wizard		Yes
Object Properties		Yes
Table Designer		Yes

Query Editor

FEATURE	AZURE DATA STUDIO	SSMS
Chart Viewer	Yes	
Export Results to CSV, JSON, XLSX	Yes	
Results to File		Yes
Results to Text		Yes
IntelliSense	Yes	Yes
Snippets	Yes	Yes
Show Plan	Preview	Yes
Client Statistics		Yes
Live Query Stats		Yes
Query Options		Yes
Spatial Viewer		Yes
SQLCMD	Yes	Yes

Operating System Support

FEATURE	AZURE DATA STUDIO	SSMS
Windows	Yes	Yes
macOS	Yes	
Linux	Yes	

Data Engineering

FEATURE	AZURE DATA STUDIO	SSMS
External Data Wizard	Preview	
HDFS Integration	Preview	
Notebooks	Preview	

Database Administration

FEATURE	AZURE DATA STUDIO	SSMS
Backup / Restore	Yes	Yes
Flat File Import	Yes	Yes
SQL Agent	Preview	Yes
SQL Profiler	Preview	Yes
Always On		Yes
Always Encrypted		Yes
Copy Data Wizard		Yes
Data Tuning Advisor		Yes
Database Diagrams		Yes
Error Log Viewer		Yes
Maintenance Plans		Yes
Multi-Server Query		Yes
Policy Based Management		Yes
PolyBase		Yes
Query Store		Yes
Registered Servers		Yes
Replication		Yes
Security Management		Yes
Service Broker		Yes
SQL Assessment	Preview	Yes

FEATURE	AZURE DATA STUDIO	SSMS
SQL Mail		Yes
Template Explorer		Yes
Vulnerability Assessment		Yes
XEvent Management		Yes

Database Development

FEATURE	AZURE DATA STUDIO	SSMS
Import\Export DACPAC	Yes	Yes
SQL Projects	Preview	
Schema Compare	Yes	

Next steps

- Download and Install Azure Data Studio
- Azure Data Studio FAQ
- Connect and query SQL Server
- Connect and query Azure SQL Database

Get help for SQL tools

- All the ways to get help
- Submit an Azure Data Studio Git issue
- Contribute to Azure Data Studio
- SQL Client Tools Forum
- SQL Server Data Tools - MSDN forum
- Support options for business users

Contribute to SQL documentation

Did you know that you could edit the content yourself? If you do so, not only will our documentation improve, but you'll also be credited as a contributor to the page.

- How to contribute to SQL Server Documentation

Quickstart: Use Azure Data Studio to connect and query SQL Server

3/5/2021 • 2 minutes to read • Edit Online

This quickstart shows how to use Azure Data Studio to connect to SQL Server, and then use Transact-SQL (T-SQL) statements to create the *TutorialDB* used in Azure Data Studio tutorials.

Prerequisites

To complete this quickstart, you need Azure Data Studio, and access to SQL Server.

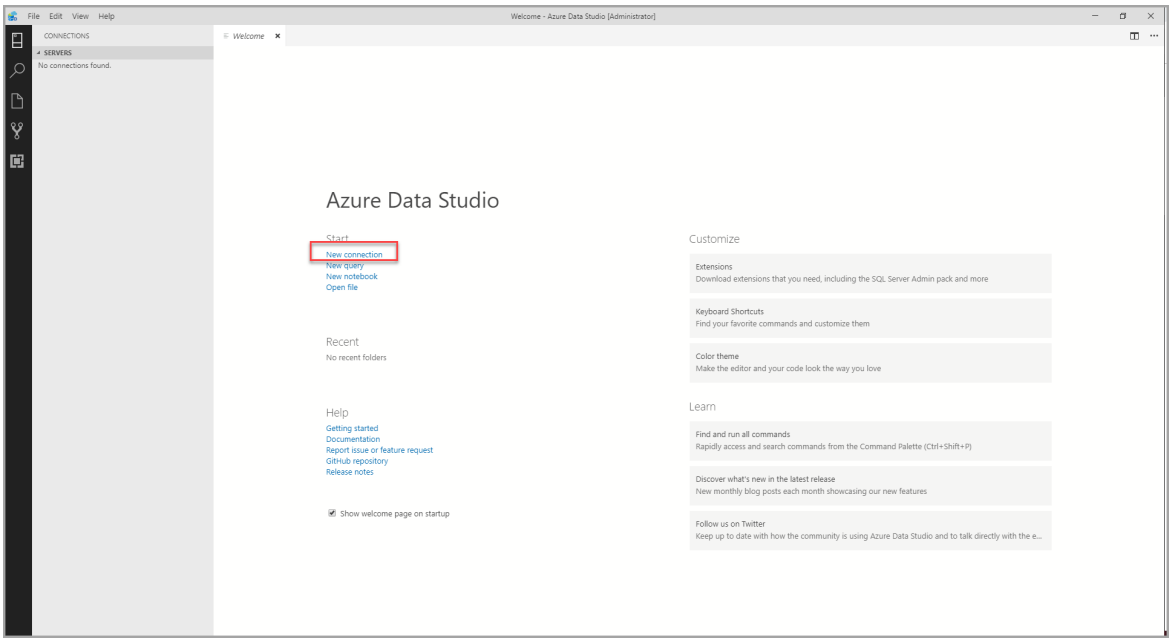
- Install Azure Data Studio.

If you don't have access to a SQL Server, select your platform from the following links (make sure you remember your SQL Login and Password!):

- Windows - Download SQL Server 2017 Developer Edition
- macOS - Download SQL Server 2017 on Docker
- Linux - Download SQL Server 2017 Developer Edition - You only need to follow the steps up to *Create and Query Data*.

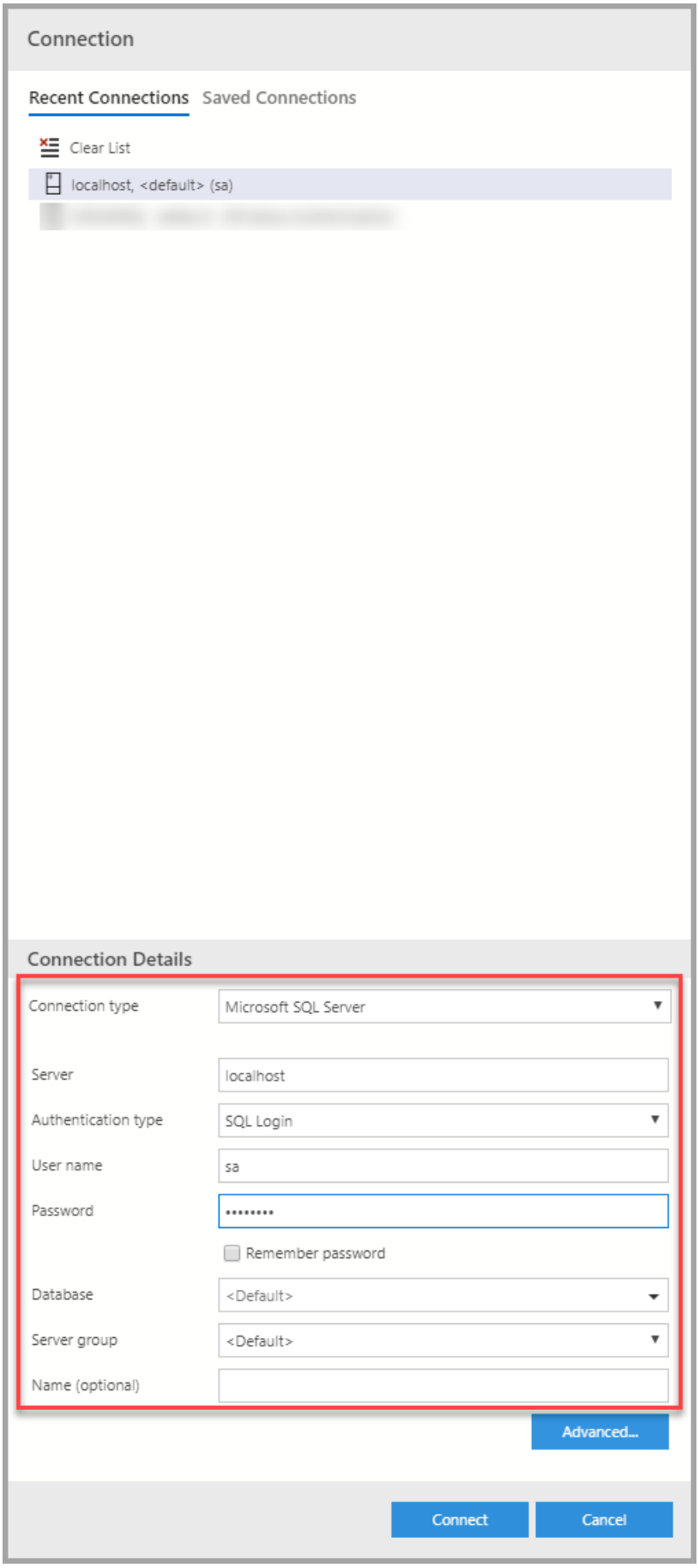
Connect to a SQL Server

1. Start **Azure Data Studio**.
2. The first time you run Azure Data Studio the **Welcome** page should open. If you don't see the **Welcome** page, select **Help** > **Welcome**. Select **New Connection** to open the **Connection** pane:



3. This article uses *SQL Login*, but *Windows Authentication* is supported. Fill in the fields as follows:
   - **Server Name:** Enter server name here. For example, localhost.
   - **Authentication Type:** SQL Login
   - **User name:** User name for the SQL Server

- **Password:** Password for the SQL Server
- **Database Name:** <Default>
- **Server Group:** <Default>



Create a database

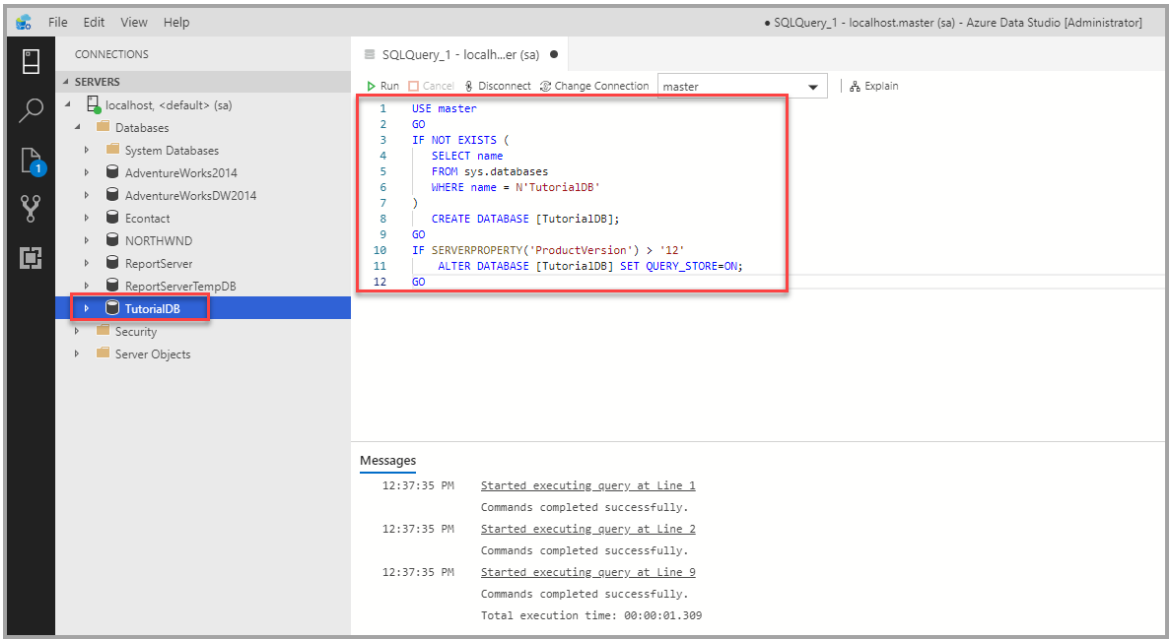
The following steps create a database named **TutorialDB**:

1. Right-click on your server, **localhost**, and select **New Query**.

2. Paste the following snippet into the query window: and then select **Run**.

```
USE master
GO
IF NOT EXISTS (
 SELECT name
 FROM sys.databases
 WHERE name = N'TutorialDB'
)
 CREATE DATABASE [TutorialDB];
GO
IF SERVERPROPERTY('ProductVersion') > '12'
 ALTER DATABASE [TutorialDB] SET QUERY_STORE=ON;
GO
```

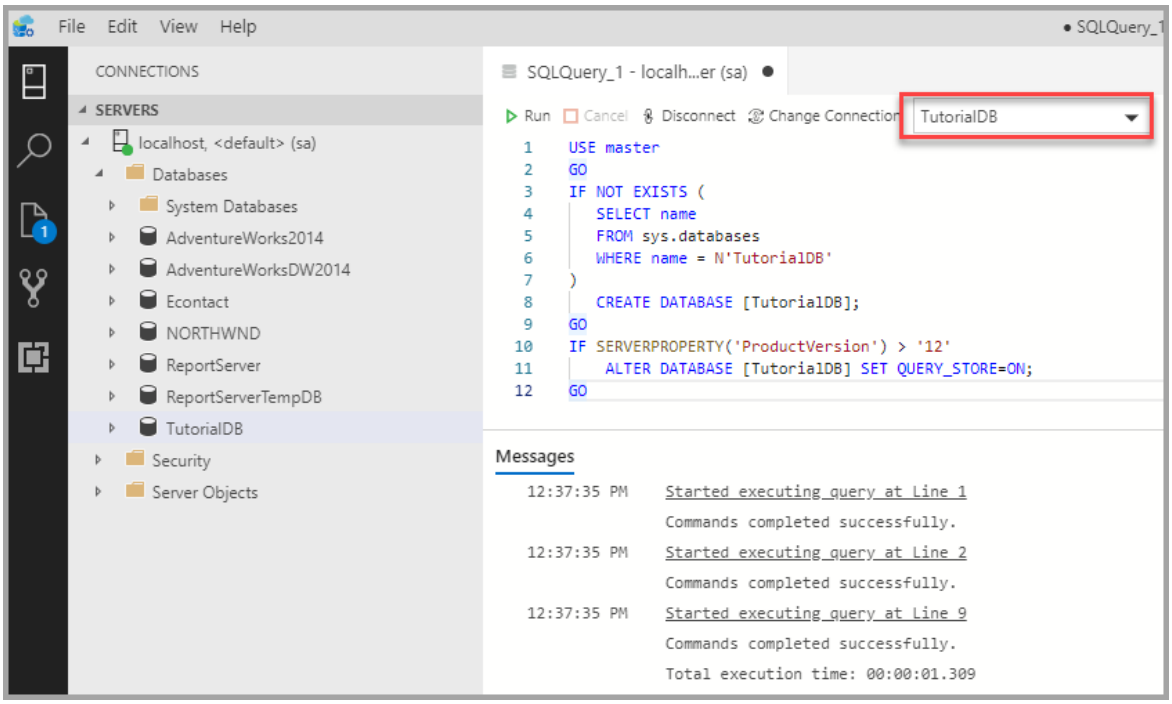
After the query completes, the new **TutorialDB** appears in the list of databases. If you don't see it, right-click the **Databases** node and select **Refresh**.



Create a table

The query editor is still connected to the *master* database, but we want to create a table in the *TutorialDB* database.

1. Change the connection context to **TutorialDB**:



2. Paste the following snippet into the query window and click **Run**:

> **NOTE**
>
> You can append this too, or overwrite the previous query in the editor. Note that clicking **Run** executes only the query that is selected. If nothing is selected, clicking **Run** executes all queries in the editor.

```
-- Create a new table called 'Customers' in schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.Customers', 'U') IS NOT NULL
 DROP TABLE dbo.Customers;
GO
-- Create the table in the specified schema
CREATE TABLE dbo.Customers
(
 CustomerId int NOT NULL PRIMARY KEY, -- primary key column
 Name nvarchar(50) NOT NULL,
 Location nvarchar(50) NOT NULL,
 Email nvarchar(50) NOT NULL
);
GO
```

After the query completes, the new **Customers** table appears in the list of tables. You might need to right-click the **TutorialDB** > **Tables** node and select **Refresh**.

Insert rows

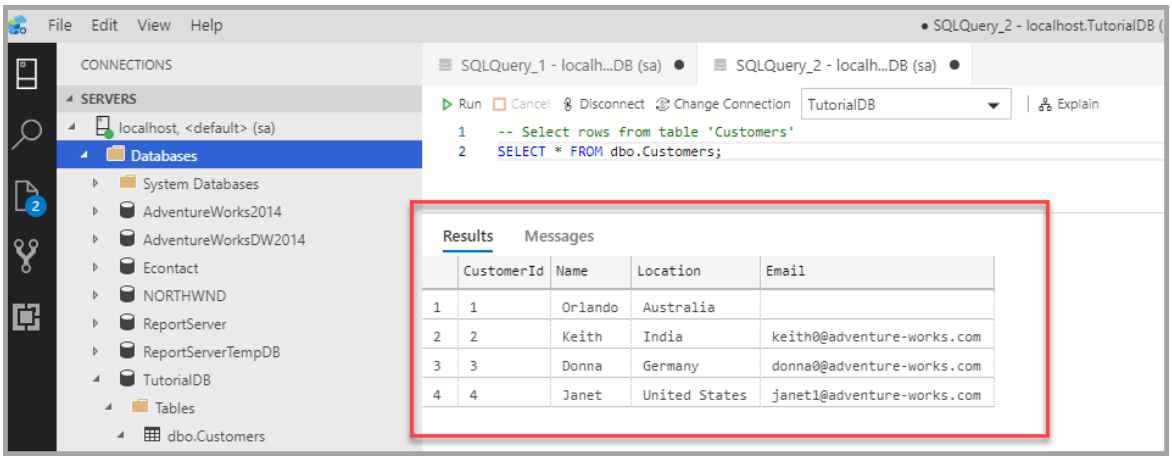
- Paste the following snippet into the query window and click **Run**:

```
-- Insert rows into table 'Customers'
INSERT INTO dbo.Customers
 ([CustomerId], [Name], [Location], [Email])
VALUES
 ( 1, N'Orlando', N'Australia', N''),
 ( 2, N'Keith', N'India', N'keith0@adventure-works.com'),
 ( 3, N'Donna', N'Germany', N'donna0@adventure-works.com'),
 ( 4, N'Janet', N'United States', N'janet1@adventure-works.com')
GO
```

View the data returned by a query

- Paste the following snippet into the query window and click **Run**:

```
-- Select rows from table 'Customers'
SELECT * FROM dbo.Customers;
```



Next steps

Now that you've successfully connected to SQL Server and run a query try out the Code editor tutorial.

Quickstart: Use Azure Data Studio to connect and query Azure SQL database

3/5/2021 • 3 minutes to read • Edit Online

In this quickstart, you'll use Azure Data Studio to connect to an Azure SQL Database server. You'll then run Transact-SQL (T-SQL) statements to create and query the TutorialDB database, which is used in other Azure Data Studio tutorials.

Prerequisites

To complete this quickstart, you need Azure Data Studio, and an Azure SQL Database server.

- Install Azure Data Studio

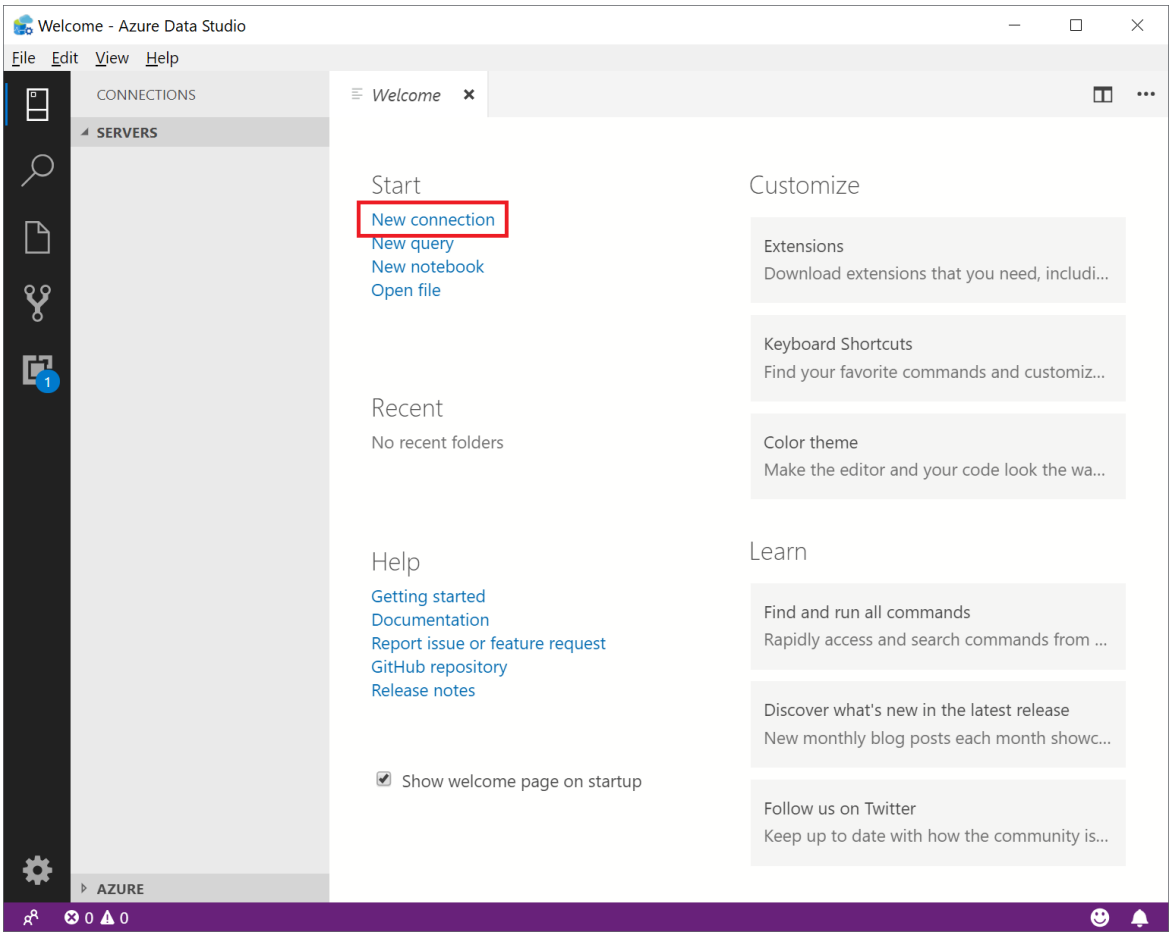
If you don't have an Azure SQL server, complete one of the following Azure SQL Database quickstarts. Remember the fully qualified server name and sign in credentials for later steps:

- Create DB - Portal
- Create DB - CLI
- Create DB - PowerShell

Connect to your Azure SQL Database server

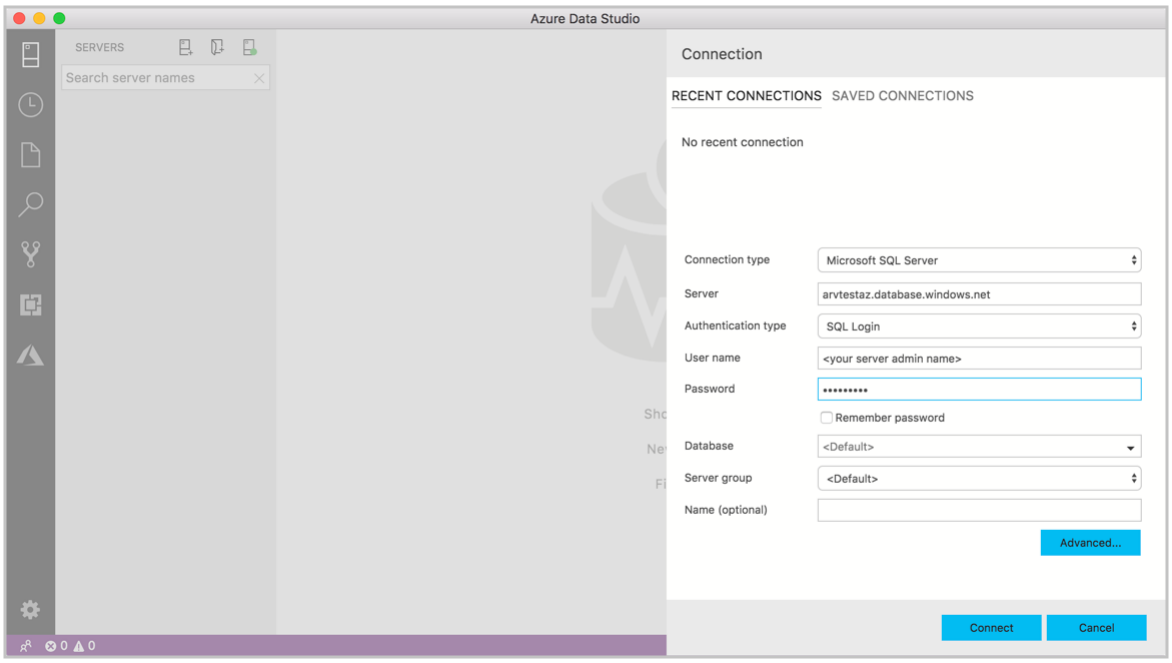
Use Azure Data Studio to establish a connection to your Azure SQL Database server.

1. The first time you run Azure Data Studio the **Welcome** page should open. If you don't see the **Welcome** page, select **Help** > **Welcome**. Select **New Connection** to open the **Connection** pane:



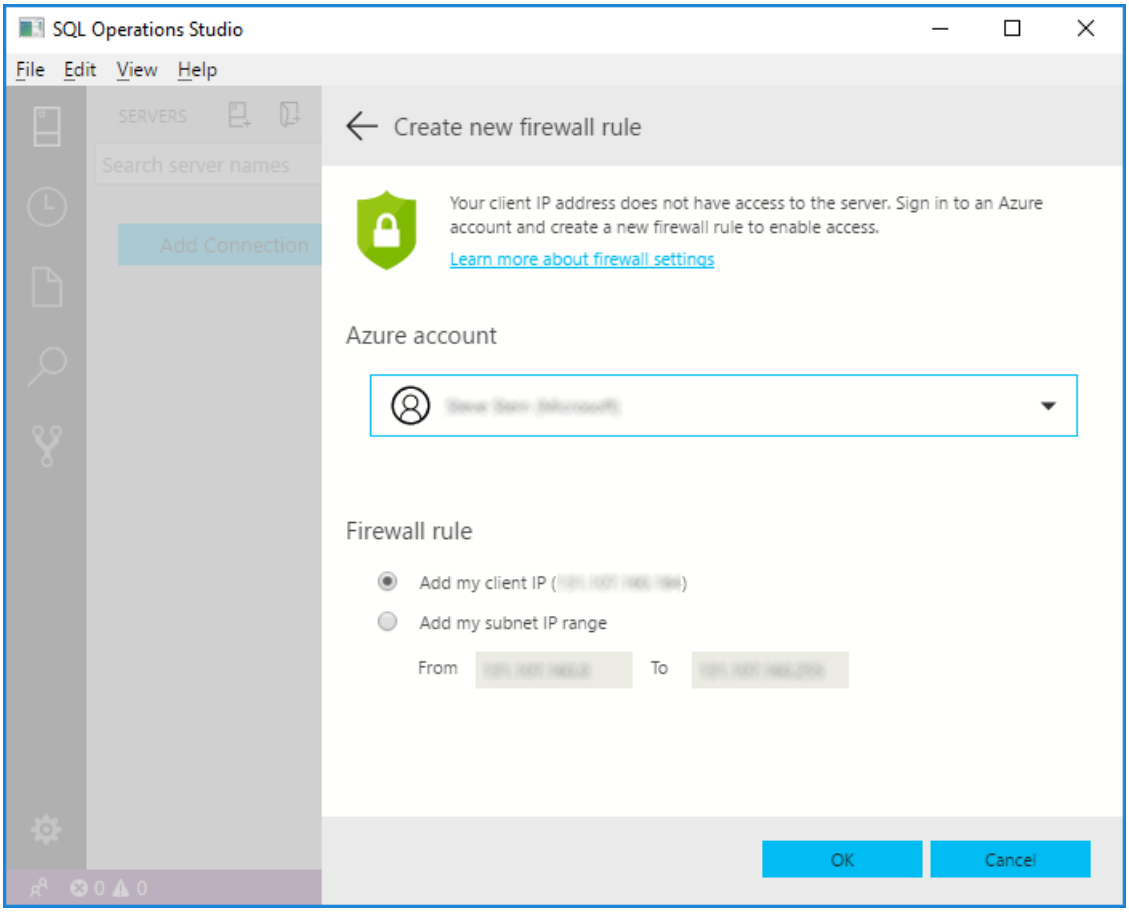
2. This article uses SQL sign-in, but also supports Windows authentication. Fill in the following fields using the server name, user name, and password for your Azure SQL server:

SETTING	SUGGESTED VALUE	DESCRIPTION
Server name	The fully qualified server name	Something like: **servername.database.windows.net**.
Authentication	SQL Login	This tutorial uses SQL Authentication.
User name	The server admin account user name	The user name from the account used to create the server.
Password (SQL Login)	The server admin account password	The password from the account used to create the server.
Save Password?	Yes or No	Select **Yes** if you don't want to enter the password each time.
Database name	*leave blank*	You're only connecting to the server here.
Server Group	Select	You can set this field to a specific server group you created.



3. Select **Connect**.

4. If your server doesn't have a firewall rule allowing Azure Data Studio to connect, the **Create new firewall rule** form opens. Complete the form to create a new firewall rule. For details, see Firewall rules.



After successfully connecting, your server opens in the **SERVERS** sidebar.

Create the tutorial database

The next sections create the TutorialDB database that's used in other Azure Data Studio tutorials.

1. Right-click on your Azure SQL server in the **SERVERS** sidebar and select **New Query**.

2. Paste this SQL into the query editor.

```
IF NOT EXISTS (
   SELECT name
   FROM sys.databases
   WHERE name = N'TutorialDB'
)
CREATE DATABASE [TutorialDB]
GO

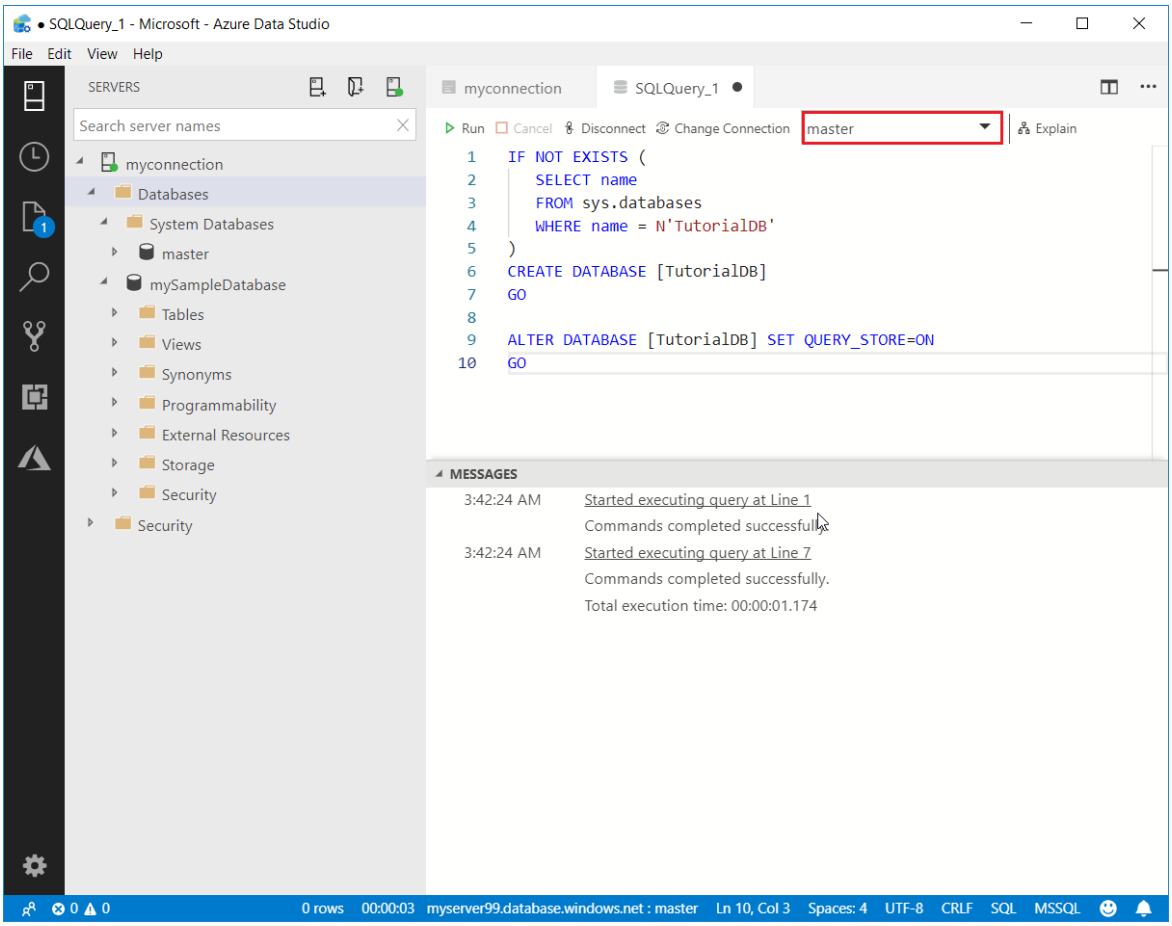
ALTER DATABASE [TutorialDB] SET QUERY_STORE=ON
GO
```

3. From the toolbar, select **Run**. Notifications appear in the **MESSAGES** pane showing query progress.

Create a table

The query editor is connected to the **master** database, but we want to create a table in the **TutorialDB** database.

1. Connect to the **TutorialDB** database.



2. Create a `Customers` table.

   Replace the previous query in the query editor with this one and select **Run**.

```
-- Create a new table called 'Customers' in schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.Customers', 'U') IS NOT NULL
DROP TABLE dbo.Customers
GO
-- Create the table in the specified schema
CREATE TABLE dbo.Customers
(
   CustomerId        INT    NOT NULL   PRIMARY KEY, -- primary key column
   Name      [NVARCHAR](50)  NOT NULL,
   Location  [NVARCHAR](50)  NOT NULL,
   Email     [NVARCHAR](50)  NOT NULL
);
GO
```

Insert rows into the table

Replace the previous query with this one and select **Run**.

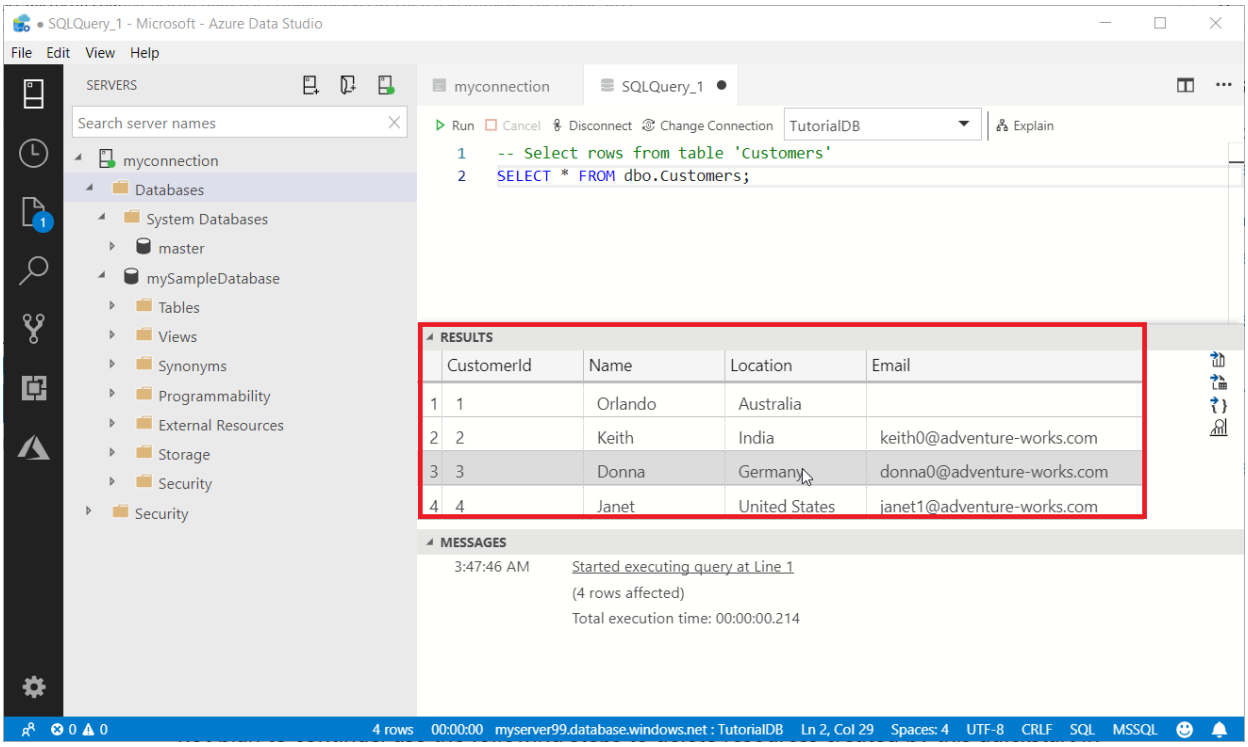
```
-- Insert rows into table 'Customers'
INSERT INTO dbo.Customers
   ([CustomerId],[Name],[Location],[Email])
VALUES
   ( 1, N'Orlando', N'Australia', N''),
   ( 2, N'Keith', N'India', N'keith0@adventure-works.com'),
   ( 3, N'Donna', N'Germany', N'donna0@adventure-works.com'),
   ( 4, N'Janet', N'United States', N'janet1@adventure-works.com')
GO
```

View the result

Replace the previous query with this one and select **Run**.

```
-- Select rows from table 'Customers'
SELECT * FROM dbo.Customers;
```

The query results display:



Clean up resources

Later quickstart articles build upon the resources created here. If you plan to work through these articles, be sure not to delete these resources. Otherwise, in the Azure portal, delete the resources you no longer need. For details, see Clean up resources.

Next steps

Now that you've successfully connected to an Azure SQL database and run a query, try the Code editor tutorial.

Quickstart: Use Azure Data Studio to connect and query data using a dedicated SQL pool in Azure Synapse Analytics

3/5/2021 • 3 minutes to read • Edit Online

This quickstart shows connecting to a dedicated SQL pool in Azure Synapse Analytics using Azure Data Studio.

Prerequisites

To complete this quickstart, you need Azure Data Studio, and a dedicated SQL pool in Azure Synapse Analytics.

- Install Azure Data Studio.

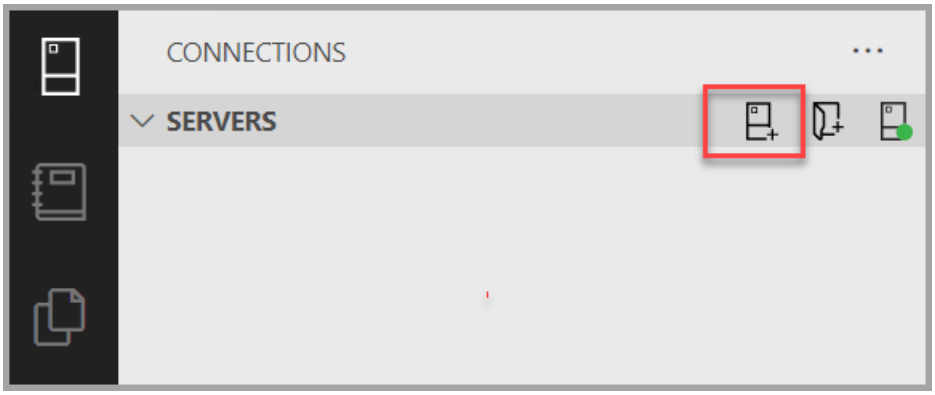
If you don't already have a dedicated SQL pool, see Create a dedicated SQL pool.

Remember the server name, and login credentials!

Connect to your dedicated SQL pool

Use Azure Data Studio to establish a connection to your Azure Synapse Analytics server.

1. The first time you run Azure Data Studio the **Connection** page should open. If you don't see the **Connection** page, select **Add Connection**, or the **New Connection** icon in the **SERVERS** sidebar:



2. This article uses *SQL Login*, but *Windows Authentication* is also supported. Fill in the fields as follows using the server name, user name, and password for *your* Azure SQL server:

SETTING	SUGGESTED VALUE	DESCRIPTION
Server name	The fully qualified server name	For example the name should look like to this: **sqlpoolservername.database.windows.net**.
Authentication	SQL Login	SQL Authentication is used in this tutorial.
User name	The server admin account	This is the account that you specified when you created the server.

SETTING	SUGGESTED VALUE	DESCRIPTION
Password (SQL Login)	The password for your server admin account	This is the password that you specified when you created the server.
Save Password?	Yes or No	Select Yes if you don't want to enter the password each time.
Database name	*leave blank*	The name of the database to which to connect.
Server Group	Select	If you created a server group, you can set to a specific server group.

3. If your server doesn't have a firewall rule allowing Azure Data Studio to connect, the **Create new firewall rule** form opens. Complete the form to create a new firewall rule. For details, see Firewall rules.
4. After successfully connecting your server opens in the *Servers* sidebar.

Create a database in your dedicated SQL pool

1. Right-click on your server, in the object explorer and select **New Query.**
2. Paste the following snippet into the query editor and select **Run**:

```
IF NOT EXISTS (
   SELECT name
   FROM sys.databases
   WHERE name = N'TutorialDB'
)
CREATE DATABASE [TutorialDB] (EDITION = 'datawarehouse', SERVICE_OBJECTIVE='DW100');
GO

ALTER DATABASE [TutorialDB] SET QUERY_STORE=ON
GO
```

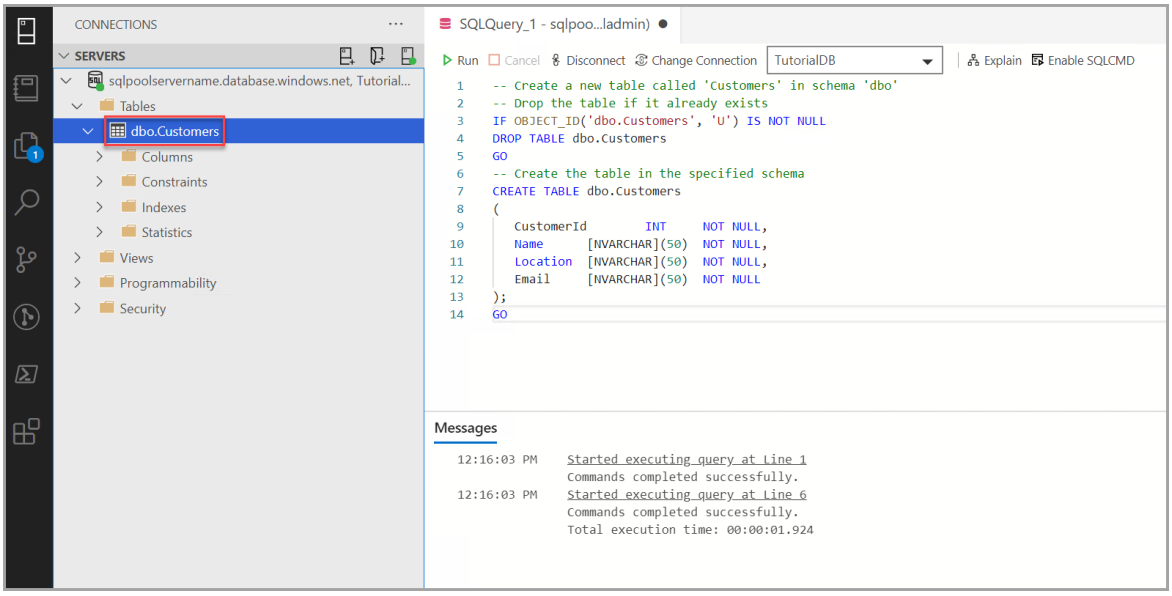
Create a table

The query editor is still connected to the *master* database, but we want to create a table in the *TutorialDB* database.

1. Change the connection context to **TutorialDB**:
2. Paste the following snippet into the query editor and select **Run**:

> **NOTE**
>
> You can append this to, or overwrite the previous query in the editor. Note that selecting **Run** executes only the query that is selected. If nothing is selected, selecting **Run** executes all queries in the editor.

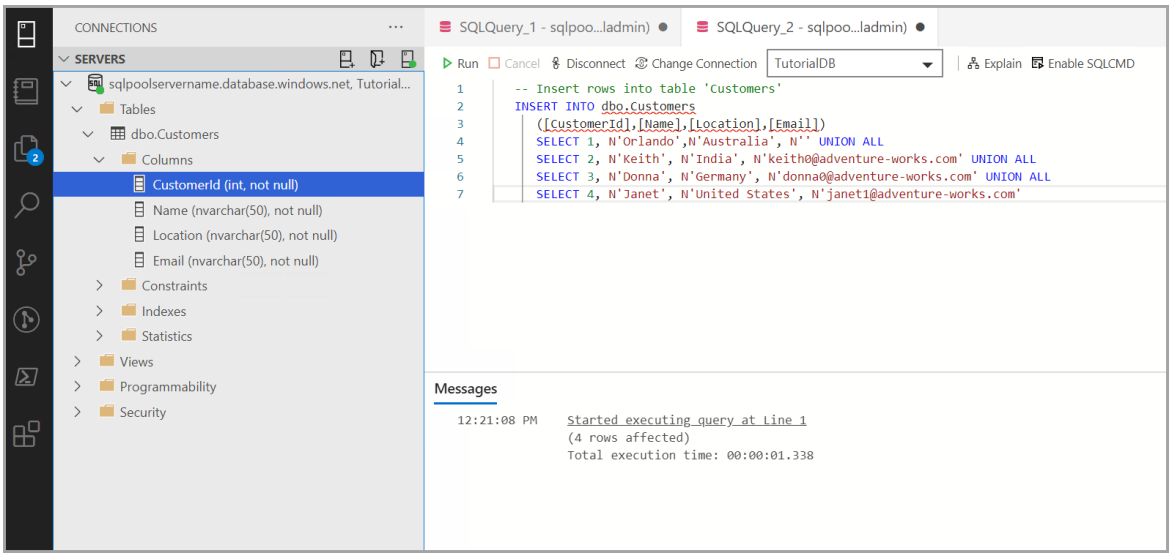
```
-- Create a new table called 'Customers' in schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.Customers', 'U') IS NOT NULL
DROP TABLE dbo.Customers
GO
-- Create the table in the specified schema
CREATE TABLE dbo.Customers
(
   CustomerId        INT     NOT NULL,
   Name      [NVARCHAR](50)  NOT NULL,
   Location  [NVARCHAR](50)  NOT NULL,
   Email     [NVARCHAR](50)  NOT NULL
);
GO
```



Insert rows

1. Paste the following snippet into the query editor and select **Run**:

```
-- Insert rows into table 'Customers'
INSERT INTO dbo.Customers
   ([CustomerId],[Name],[Location],[Email])
   SELECT 1, N'Orlando',N'Australia', N'' UNION ALL
   SELECT 2, N'Keith', N'India', N'keith0@adventure-works.com' UNION ALL
   SELECT 3, N'Donna', N'Germany', N'donna0@adventure-works.com' UNION ALL
   SELECT 4, N'Janet', N'United States', N'janet1@adventure-works.com'
```

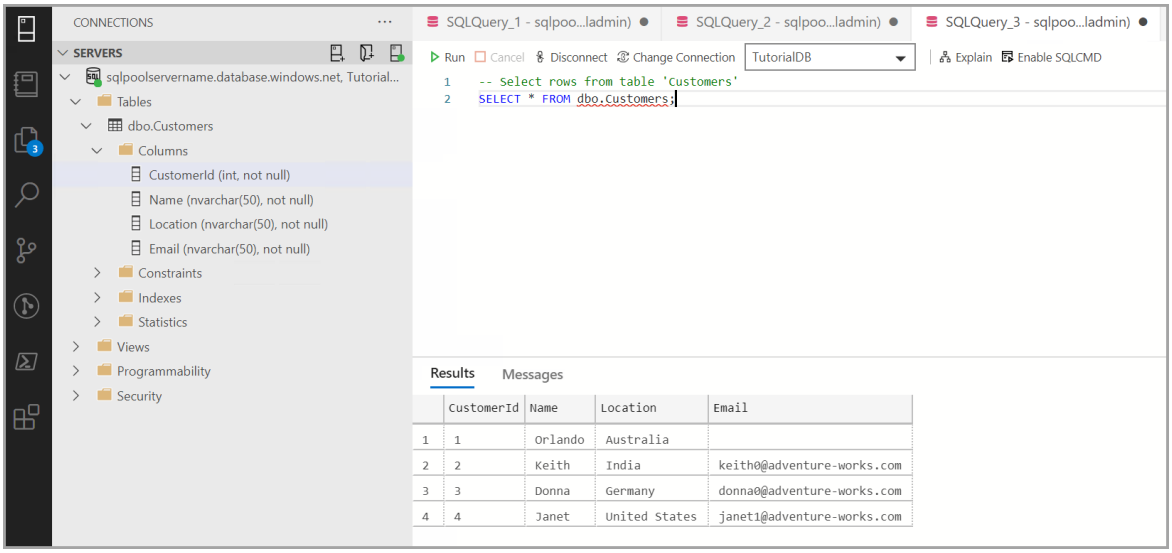


View the result

1. Paste the following snippet into the query editor and select **Run**:

```
-- Select rows from table 'Customers'
SELECT * FROM dbo.Customers;
```

2. The results of the query are displayed:



Clean up resources

If you don't plan to continue working with the sample databases created in this article, then delete the resource group.

Next steps

For more information, visit Connecting to Synapse SQL with Azure Data Studio.

Now that you've successfully connected to an Azure Synapse Analytics and ran a query, try out the Code editor tutorial.

Quickstart: Use Azure Data Studio to connect and query PostgreSQL

3/5/2021 • 2 minutes to read • Edit Online

This quickstart shows how to use Azure Data Studio to connect to PostgreSQL, and then use SQL statements to create the database *tutorialdb* and query it.

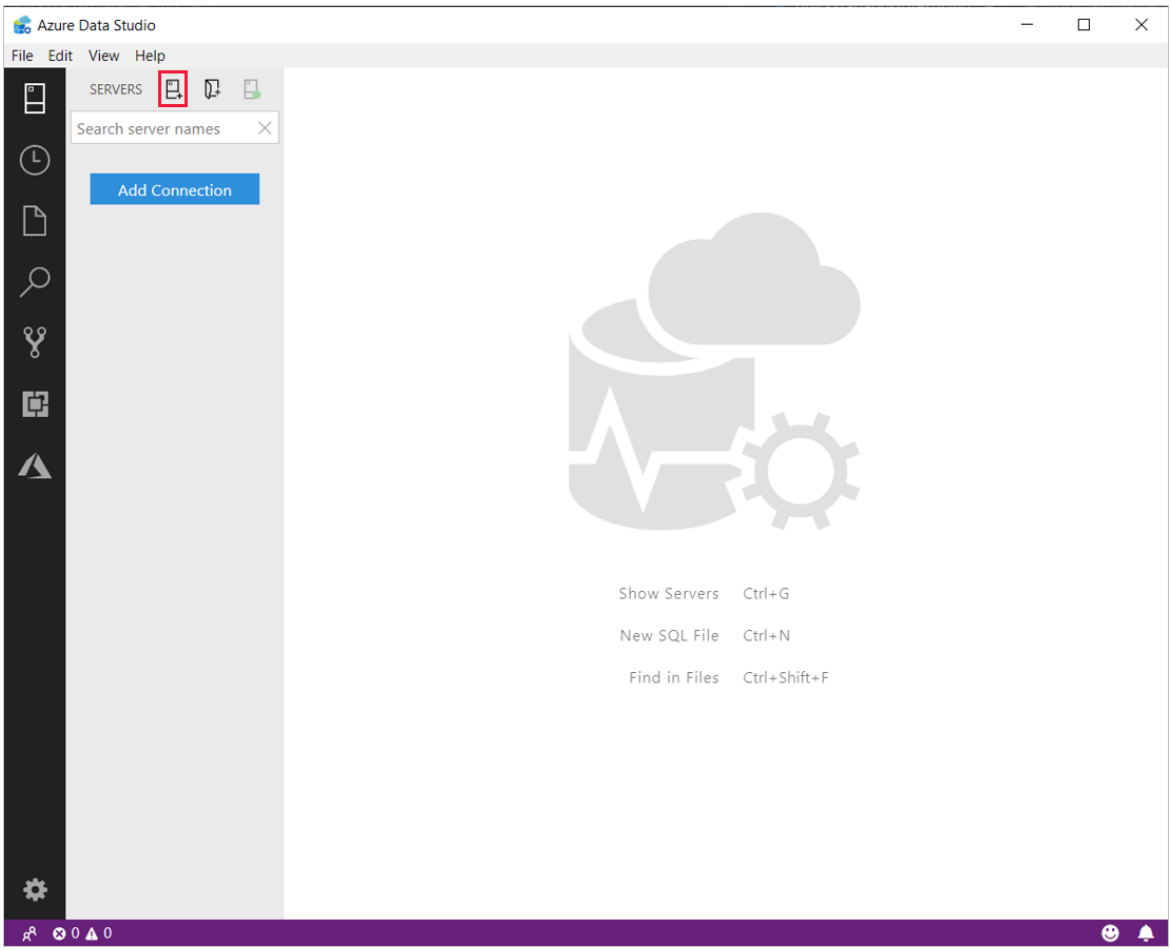
Prerequisites

To complete this quickstart, you need Azure Data Studio, the PostgreSQL extension for Azure Data Studio, and access to a PostgreSQL server.

- Install Azure Data Studio.
- Install the PostgreSQL extension for Azure Data Studio.
- Install PostgreSQL. (Alternatively, you can create a Postgres database in the cloud using az postgres up).

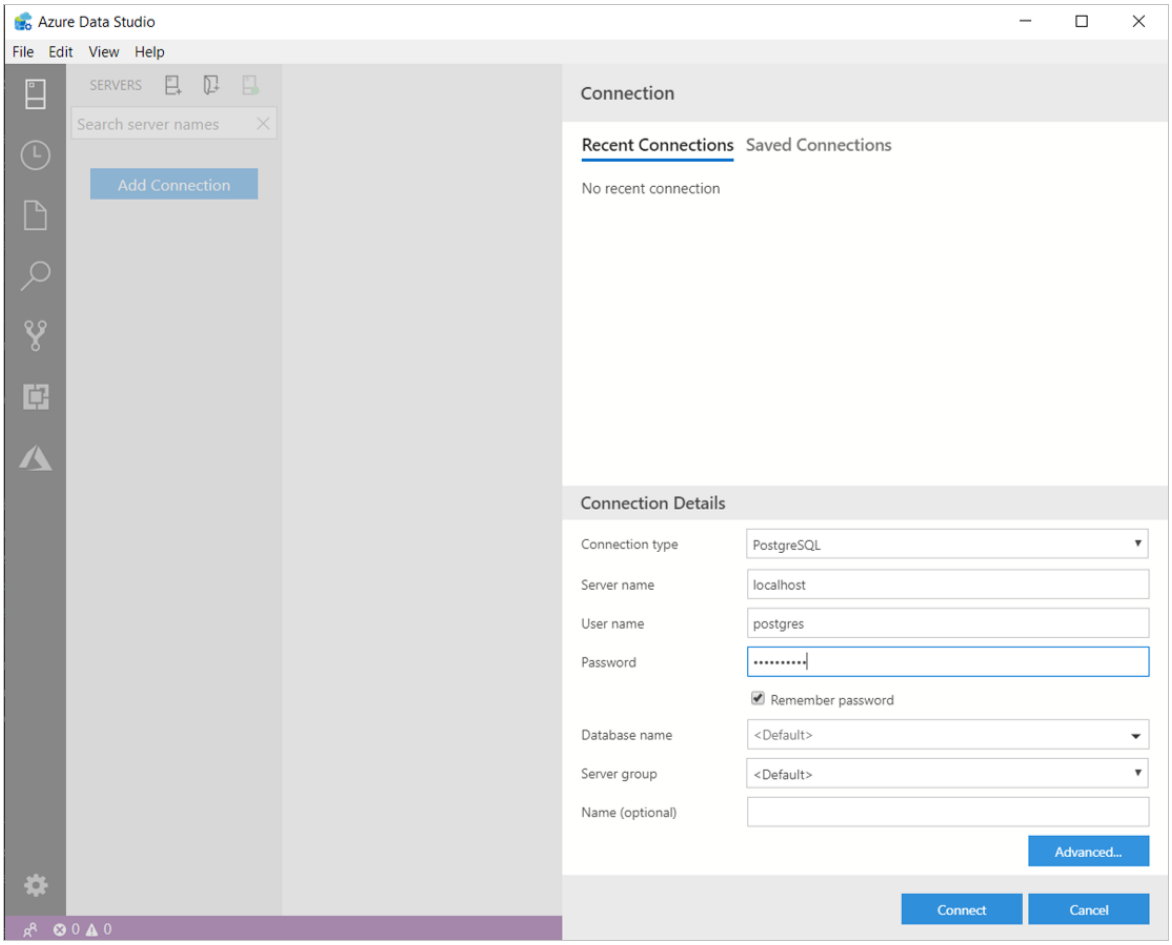
Connect to PostgreSQL

1. Start **Azure Data Studio**.
2. The first time you start Azure Data Studio the **Connection** dialog opens. If the **Connection** dialog doesn't open, click the **New Connection** icon in the **SERVERS** page:



3. In the form that pops up, go to **Connection type** and select **PostgreSQL** from the drop-down.

4. Fill in the remaining fields using the server name, user name, and password for your PostgreSQL server.



SETTING	EXAMPLE VALUE	DESCRIPTION
Server name	localhost	The fully qualified server name
User name	postgres	The user name you want to log in with.
Password (SQL Login)	*password*	The password for the account you are logging in with.
Password	*Check*	Check this box if you don't want to enter the password each time you connect.
Database name	<Default>	Fill this if you want the connection to specify a database.
Server group	<Default>	This option lets you assign this connection to a specific server group you create.
Name (optional)	*leave blank*	This option lets you specify a friendly name for your server.

5. Select **Connect**.

After successfully connecting, your server opens in the **SERVERS** sidebar.

Create a database

The following steps create a database named **tutorialdb**:

1. Right-click on your PostgreSQL server in the **SERVERS** sidebar and select **New Query**.
2. Paste this SQL statement in the query editor that opens up.

```
CREATE DATABASE tutorialdb;
```

3. From the toolbar select **Run** to execute the query. Notifications appear in the **MESSAGES** pane to show query progress.

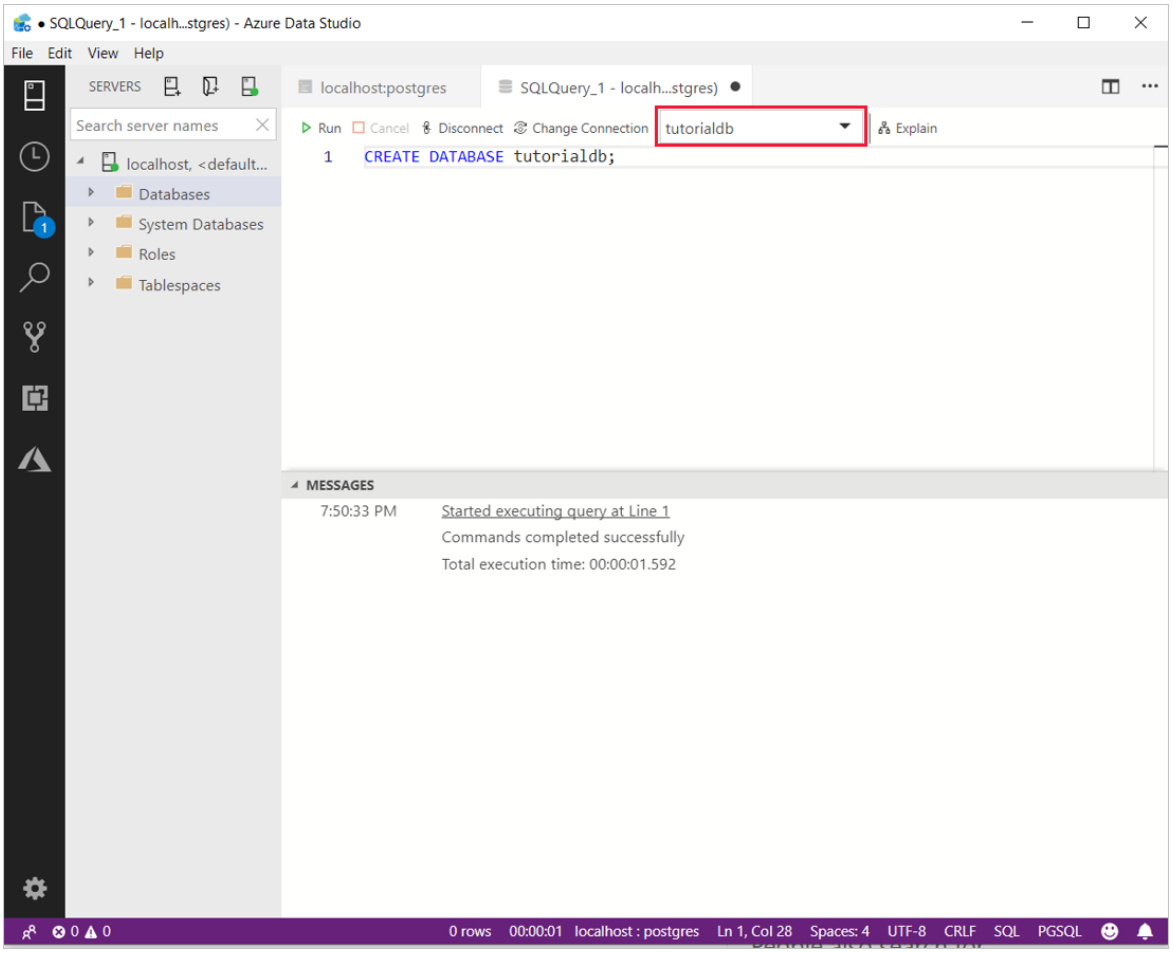
> **TIP**
> You can use **F5** on your keyboard to execute the statement instead of using **Run**.

After the query completes, right-click **Databases** and select **Refresh** to see **tutorialdb** in the list under the **Databases** node.

Create a table

The following steps create a table in the **tutorialdb**:

1. Change the connection context to **tutorialdb** using the drop-down in the query editor.



2. Paste the following SQL statement into the query editor and click **Run**.

> **NOTE**
>
> You can either append this or overwrite the existing query in the editor. Clicking **Run** executes only the query that is highlighted. If nothing is highlighted, clicking **Run** executes all queries in the editor.

```
-- Drop the table if it already exists
DROP TABLE IF EXISTS customers;
-- Create a new table called 'customers'
CREATE TABLE customers(
    customer_id SERIAL PRIMARY KEY,
    name VARCHAR (50) NOT NULL,
    location VARCHAR (50) NOT NULL,
    email VARCHAR (50) NOT NULL
);
```

Insert rows

Paste the following snippet into the query window and click **Run**:

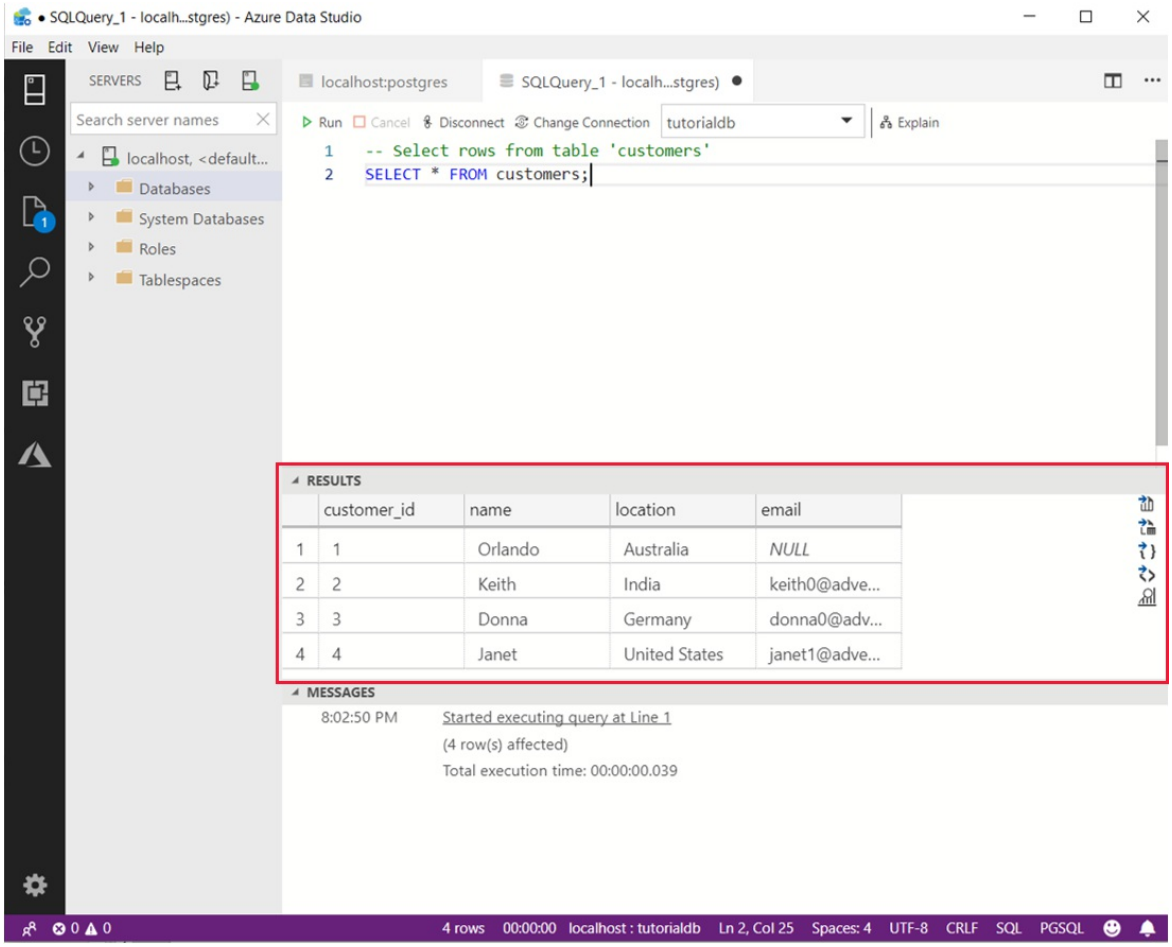
```
-- Insert rows into table 'customers'
INSERT INTO customers
    (customer_id, name, location, email)
 VALUES
   ( 1, 'Orlando', 'Australia', ''),
   ( 2, 'Keith', 'India', 'keith0@adventure-works.com'),
   ( 3, 'Donna', 'Germany', 'donna0@adventure-works.com'),
   ( 4, 'Janet', 'United States','janet1@adventure-works.com');
```

Query the data

1. Paste the following snippet into the query editor and click **Run**:

   ```
   -- Select rows from table 'customers'
   SELECT * FROM customers;
   ```

2. The results of the query are displayed:



Next Steps

Learn about the scenarios available for Postgres in Azure Data Studio.

Tutorial: Use the Transact-SQL editor to create database objects - Azure Data Studio

11/2/2020 • 5 minutes to read • Edit Online

Creating and running queries, stored procedures, scripts, etc. are the core tasks of database professionals. This tutorial demonstrates the key features in the T-SQL editor to create database objects.

In this tutorial, you learn how to use Azure Data Studio to:

- Search database objects
- Edit table data
- Use snippets to quickly write T-SQL
- View database object details using *Peek Definition* and *Go to Definition*

Prerequisites

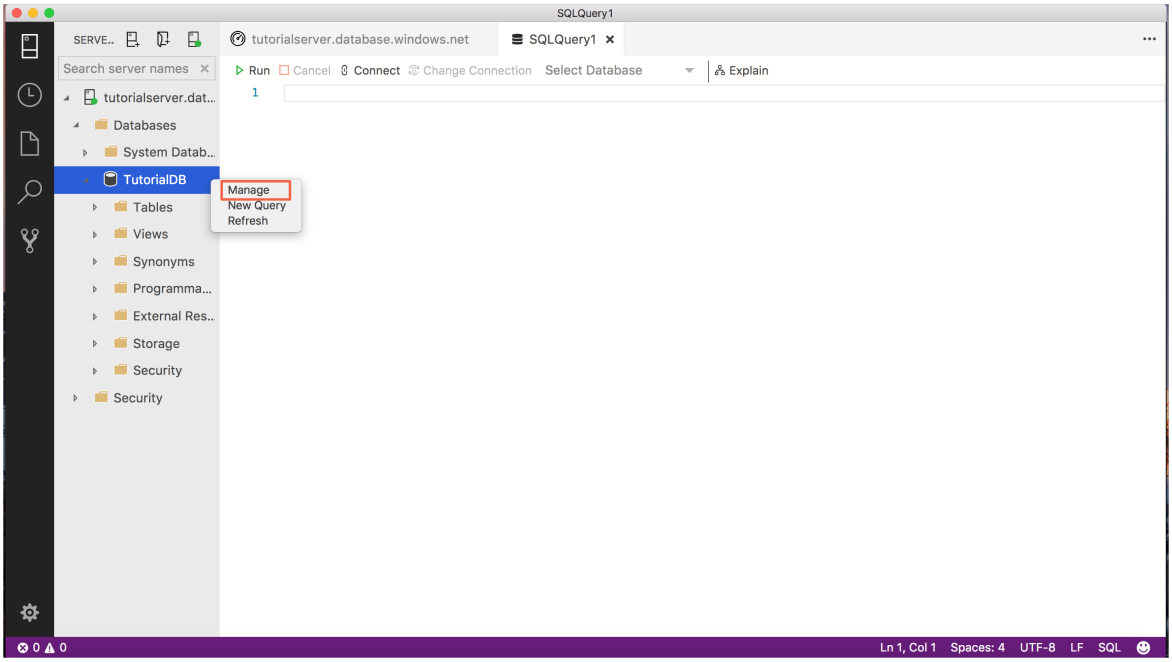
This tutorial requires the SQL Server or Azure SQL Database *TutorialDB*. To create the *TutorialDB* database, complete one of the following quickstarts:

- Connect and query SQL Server using Azure Data Studio
- Connect and query Azure SQL Database using Azure Data Studio

Quickly locate a database object and perform a common task

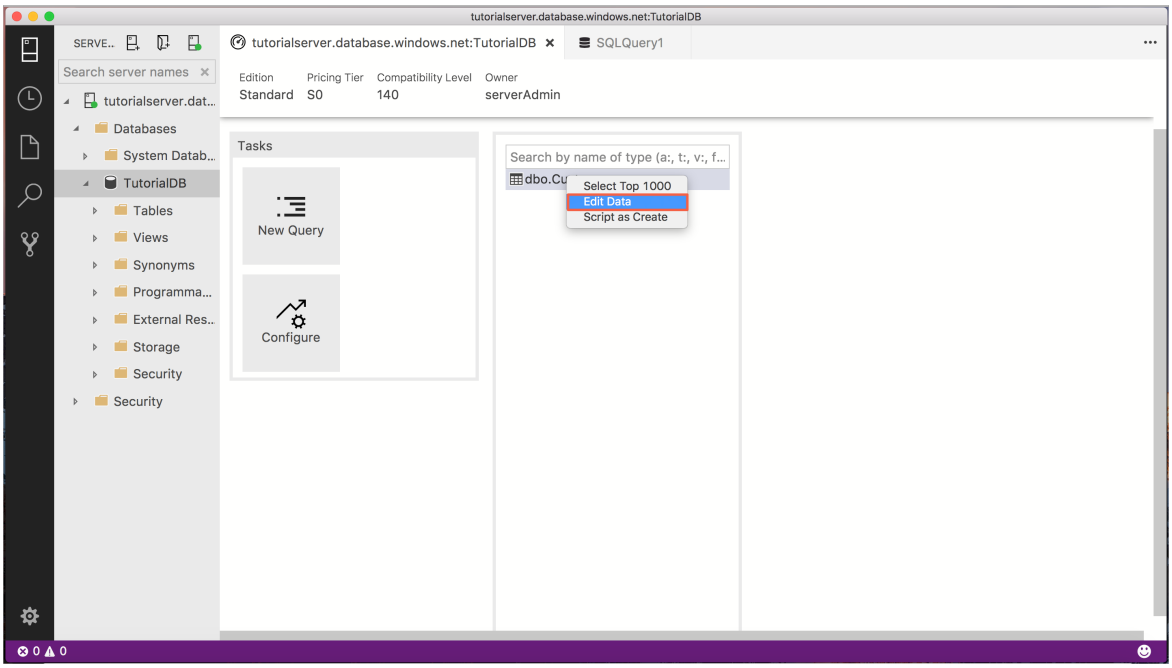
Azure Data Studio provides a search widget to quickly find database objects. The results list provides a context menu for common tasks relevant to the selected object, such as *Edit Data* for a table.

1. Open the SERVERS sidebar (**Ctrl+G**), expand **Databases**, and select **TutorialDB**.
2. Open the *TutorialDB Dashboard* by right-clicking **TutorialDB** and selecting **Manage** from the context menu:

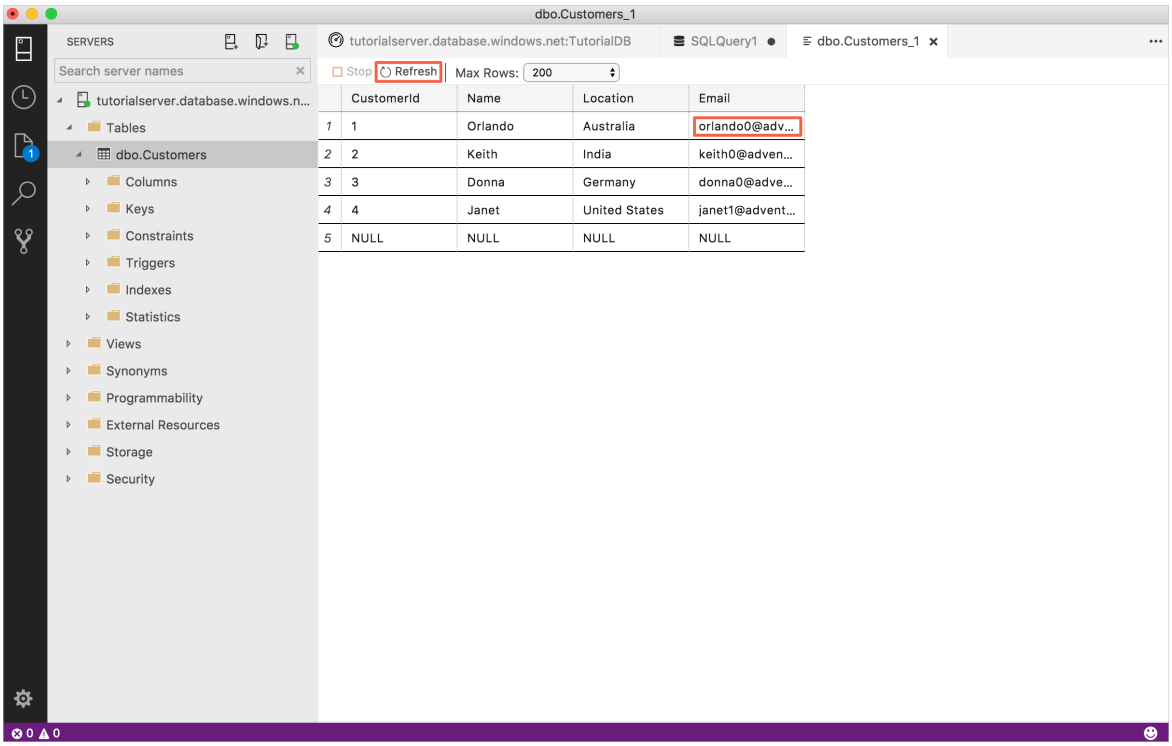


3. On the dashboard, right-click **dbo.Customers** (in the search widget) and select **Edit Data**.

> **TIP**
>
> For databases with many objects, use the search widget to quickly locate the table, view, etc. that you're looking for.



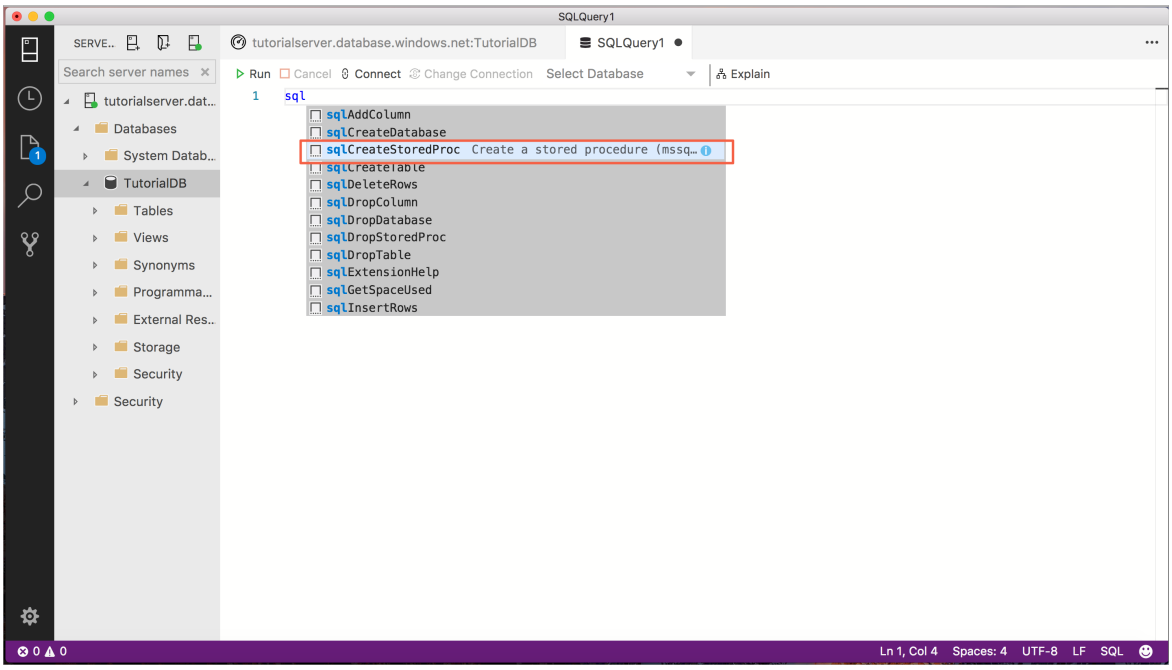
4. Edit the **Email** column in the first row, type *orlando0@adventure-works.com*, and press **Enter** to save the change.



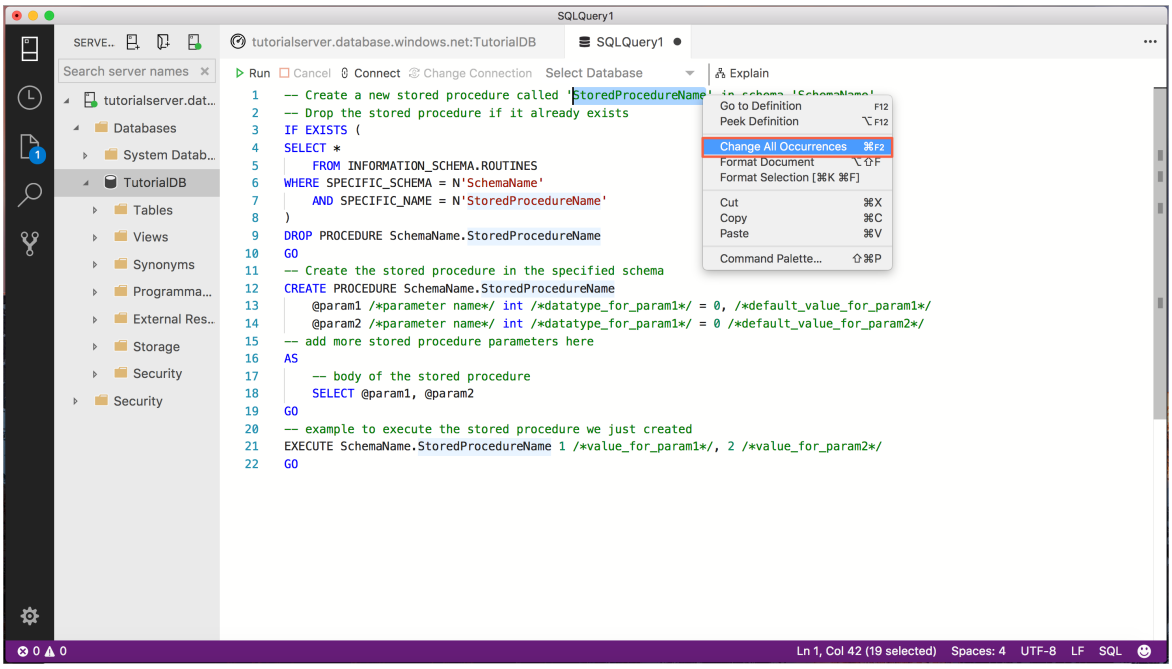
Use T-SQL snippets to create stored procedures

Azure Data Studio provides many built-in T-SQL snippets for quickly creating statements.

1. Open a new query editor by pressing **Ctrl+N**.
2. Type **sql** in the editor, arrow down to **sqlCreateStoredProcedure**, and press the *Tab* key (or *Enter*) to load the create stored procedure snippet.



3. The create stored procedure snippet has two fields set up for quick edit, *StoredProcedureName* and *SchemaName*. Select *StoredProcedureName*, right-click, and select **Change All Occurrences**. Now type *getCustomer* and all *StoredProcedureName* entries change to *getCustomer*.



4. Change all occurrences of *SchemaName* to *dbo*.

5. The snippet contains placeholder parameters and body text that needs updating. The *EXECUTE* statement also contains placeholder text because it doesn't know how many parameters the procedure will have. For this tutorial update the snippet so it looks like the following code:

```
-- Create a new stored procedure called 'getCustomer' in schema 'dbo'
-- Drop the stored procedure if it already exists
IF EXISTS (
SELECT *
FROM INFORMATION_SCHEMA.ROUTINES
WHERE SPECIFIC_SCHEMA = N'dbo'
AND SPECIFIC_NAME = N'getCustomer'
)
DROP PROCEDURE dbo.getCustomer
GO
-- Create the stored procedure in the specified schema
CREATE PROCEDURE dbo.getCustomer
@ID int
-- add more stored procedure parameters here
AS
-- body of the stored procedure
SELECT  c.CustomerId,
c.Name,
c.Location,
c.Email
FROM dbo.Customers c
WHERE c.CustomerId = @ID
FOR JSON PATH

GO
-- example to execute the stored procedure we just created
EXECUTE dbo.getCustomer 1
GO
```

6. To create the stored procedure and give it a test run, press **F5**.

The stored procedure is now created, and the **RESULTS** pane displays the returned customer in JSON. To see formatted JSON, click the returned record.

Use Peek Definition

Azure Data Studio provides the ability to view an objects definition using the peek definition feature. This section creates a second stored procedure and uses peek definition to see what columns are in a table to quickly create the body of the stored procedure.

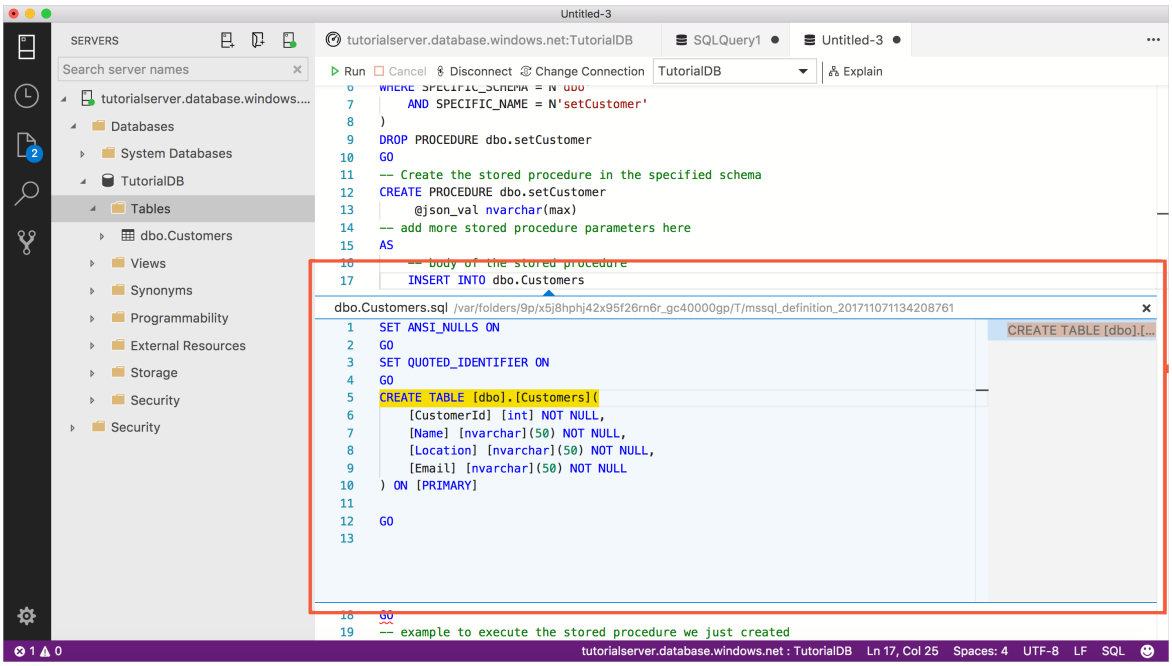
1. Open a new editor by pressing **Ctrl+N**.
2. Type *sql* in the editor, arrow down to *sqlCreateStoredProcedure*, and press the *Tab* key (or *Enter*) to load the create stored procedure snippet.
3. Type in *setCustomer* for *StoredProcedureName* and *dbo* for *SchemaName*
4. Replace the @param placeholders with the following parameter definition:

```
@json_val nvarchar(max)
```

5. Replace the body of the stored procedure with the following code:

```
INSERT INTO dbo.Customers
```

6. In the *INSERT* line you just added, right-click **dbo.Customers** and select **Peek Definition**.



7. The table definition appears so you can quickly see what columns are in the table. Refer to the column list to easily complete the statements for your stored procedure. Finish creating the INSERT statement you added previously to complete the body of the stored procedure, and close the peek definition window:

```
INSERT INTO dbo.Customers (CustomerId, Name, Location, Email)
    SELECT CustomerId, Name, Location, Email
    FROM OPENJSON (@json_val)
    WITH(   CustomerId int,
            Name nvarchar(50),
            Location nvarchar(50),
            Email nvarchar(50)
 )
```

8. Delete (or comment out) the *EXECUTE* command at the bottom of the query.

9. The entire statement should look like the following code:

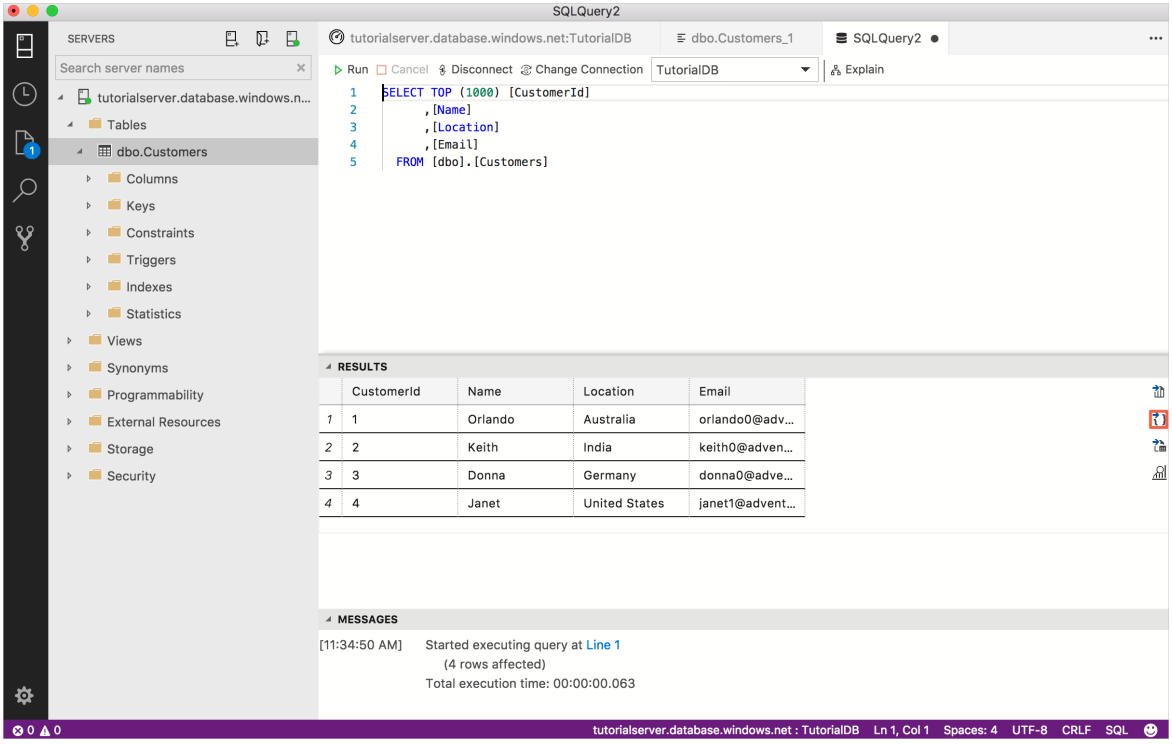
```
-- Create a new stored procedure called 'setCustomer' in schema 'dbo'
-- Drop the stored procedure if it already exists
IF EXISTS (
SELECT *
    FROM INFORMATION_SCHEMA.ROUTINES
    WHERE SPECIFIC_SCHEMA = N'dbo'
    AND SPECIFIC_NAME = N'setCustomer'
)
DROP PROCEDURE dbo.setCustomer
GO
-- Create the stored procedure in the specified schema
CREATE PROCEDURE dbo.setCustomer
    @json_val nvarchar(max)
AS
    -- body of the stored procedure
    INSERT INTO dbo.Customers (CustomerId, Name, Location, Email)
    SELECT CustomerId, Name, Location, Email
    FROM OPENJSON (@json_val)
    WITH(   CustomerId int,
            Name nvarchar(50),
            Location nvarchar(50),
            Email nvarchar(50)
    )
GO
```

10. To create the *setCustomer* stored procedure, press **F5**.

Use save query results as JSON to test the setCustomer stored procedure

The *setCustomer* stored procedure created in the previous section requires JSON data be passed into the *@json_val* parameter. This section shows how to get a properly formatted bit of JSON to pass into the parameter so you can test the stored procedure.

1. In the **SERVERS** sidebar right-click the *dbo.Customers* table and click **SELECT TOP 1000 Rows**.
2. Select the first row in the results view, make sure the entire row is selected (click the number 1 in the left-most column), and select **Save as JSON**.
3. Change the folder to a location you'll remember so you can delete the file later (for example desktop) and click **Save**. The JSON formatted file opens.



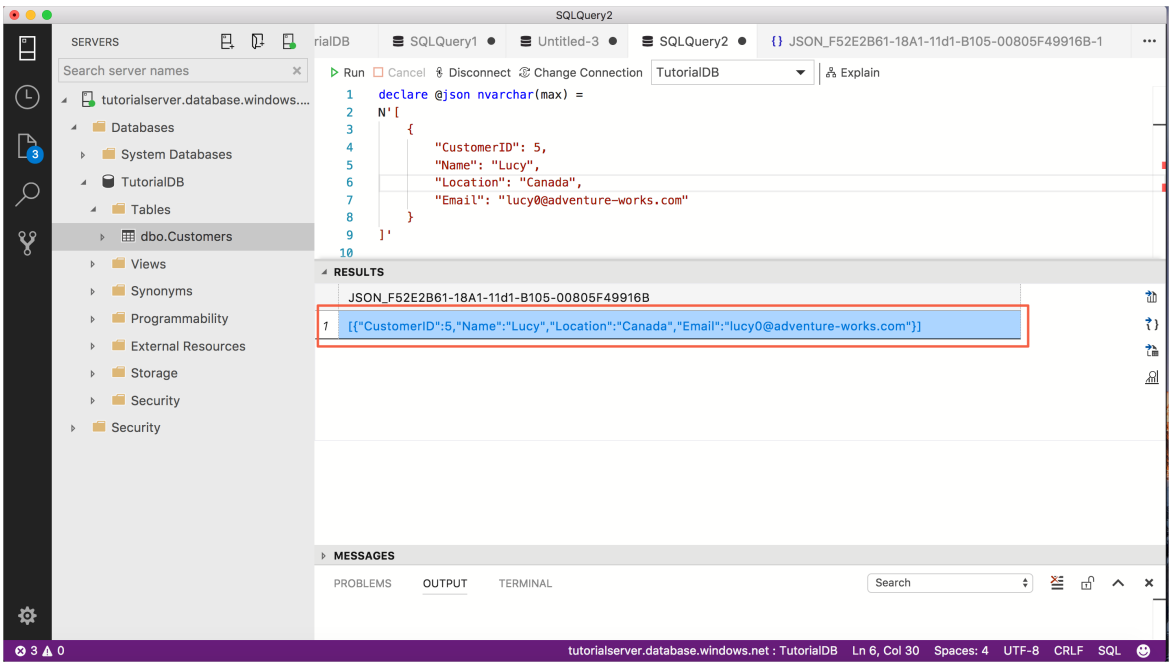
4. Select the JSON data in the editor and copy it.
5. Open a new editor by pressing **Ctrl+N**.
6. The previous steps show how you can easily get the properly formatted data to complete the call to the *setCustomer* procedure. You can see the following code uses the same JSON format with new customer details so we can test the *setCustomer* procedure. The statement includes syntax to declare the parameter and run the new get and set procedures. You can paste the copied data from the previous section and edit it so it is the same as the following example, or simply paste the following statement into the query editor.

```
-- example to execute the stored procedure we just created
declare @json nvarchar(max) =
N'[
    {
        "CustomerId": 5,
        "Name": "Lucy",
        "Location": "Canada",
        "Email": "lucy0@adventure-works.com"
    }
]'

EXECUTE dbo.setCustomer @json_val = @json
GO

EXECUTE dbo.getCustomer @ID = 5
```

7. Execute the script by pressing **F5**. The script inserts a new customer and returns the new customer's information in JSON format. Click the result to open a formatted view.



Next steps

In this tutorial, you learned how to:

- Quick search schema objects
- Edit table data
- Writing T-SQL script using snippets
- Learn about database object details using Peek Definition and Go to Definition

To learn how to enable the **five slowest queries** widget, complete the next tutorial:

Enable the slow queries sample insight widget

Tutorial: Add the *five slowest queries* sample widget to the database dashboard

11/2/2020 • 2 minutes to read • Edit Online

This tutorial demonstrates the process of adding one of the built-in Azure Data Studio sample widgets to the *database dashboard* to quickly view a database's five slowest queries. You also learn how to view the details of the slow queries and query plans using Azure Data Studio features. During this tutorial, you learn how to:

- Enable Query Store on a database
- Add a pre-built insight widget to the database dashboard
- View details about the database's slowest queries
- View query execution plans for the slow queries

Azure Data Studio includes several insight widgets out-of-the-box. This tutorial shows how to add the *query-data-store-db-insight* widget, but the steps are basically the same for adding any widget.

Prerequisites

This tutorial requires the SQL Server or Azure SQL Database *TutorialDB*. To create the *TutorialDB* database, complete one of the following quickstarts:

- Connect and query SQL Server using Azure Data Studio
- Connect and query Azure SQL Database using Azure Data Studio

Turn on Query Store for your database

The widget in this example requires *Query Store* to be enabled.

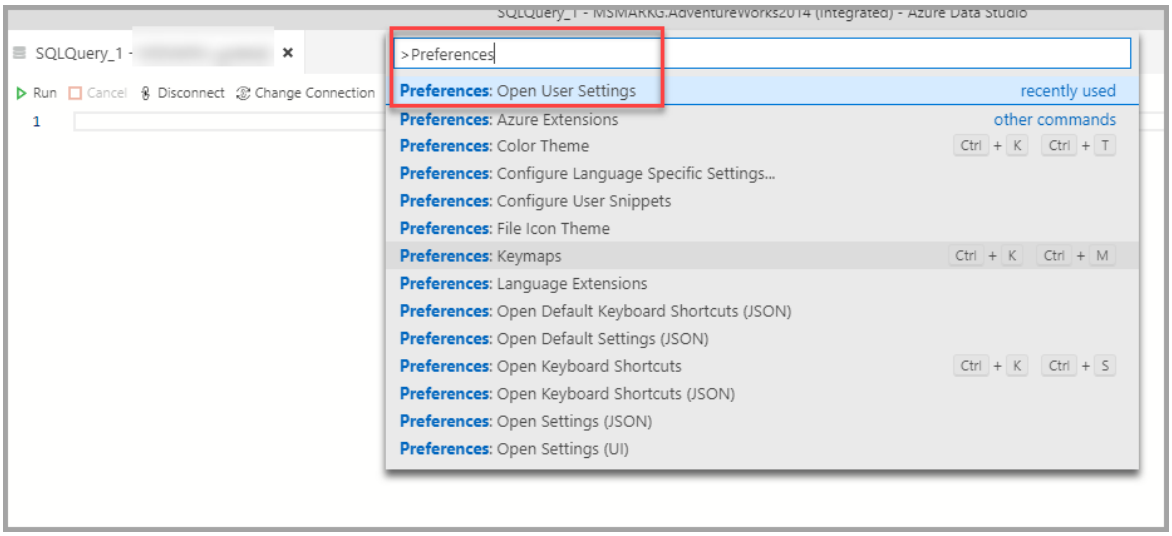
1. Right-click the **TutorialDB** database (in the **SERVERS** sidebar) and select **New Query**.
2. Paste the following Transact-SQL (T-SQL) statement in the query editor, and click **Run**:

```
ALTER DATABASE TutorialDB SET QUERY_STORE = ON
```

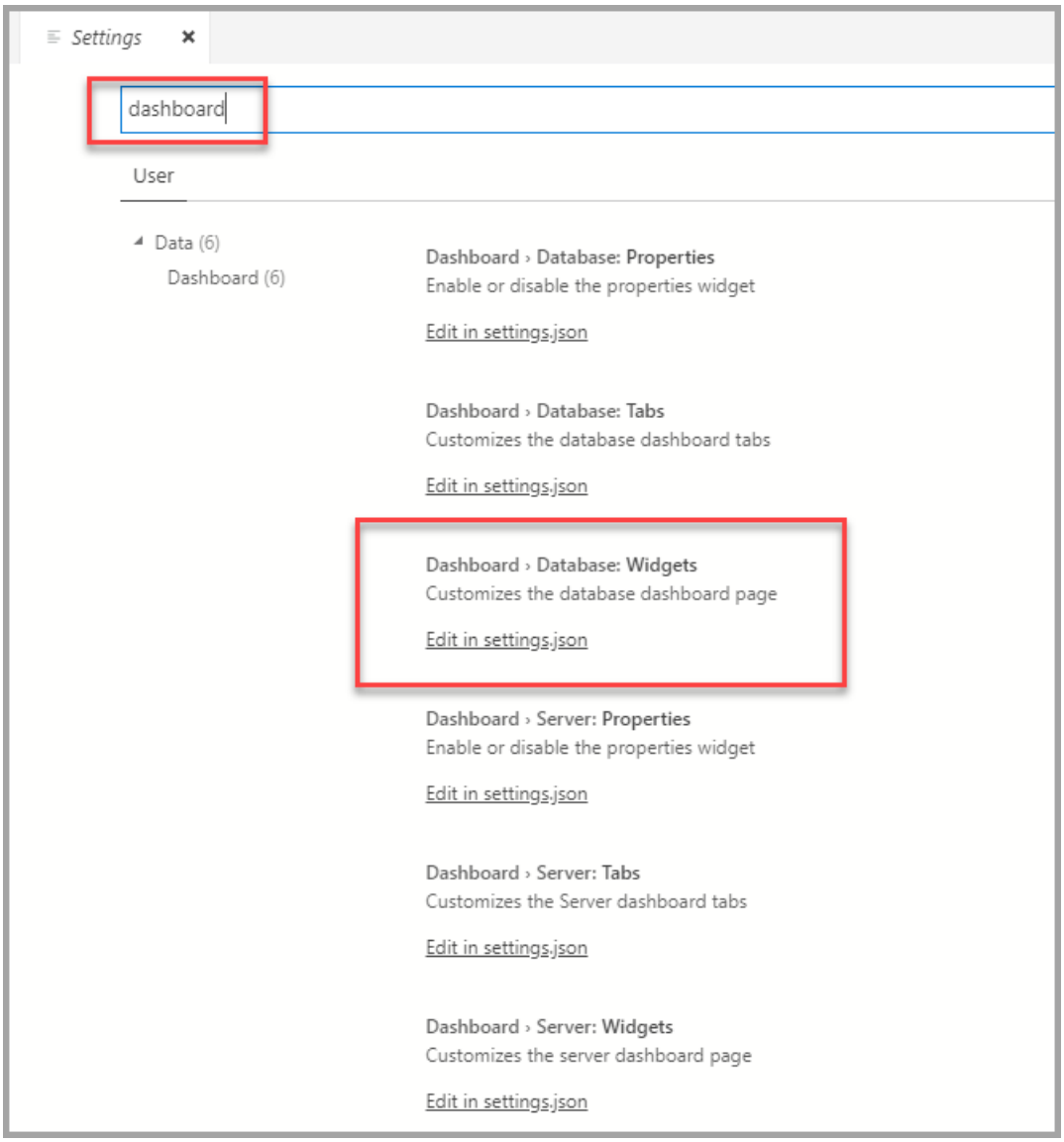
Add the slow queries widget to your database dashboard

To add the *slow queries widget* to your dashboard, edit the *dashboard.database.widgets* setting in your *User Settings* file.

1. Open *User Settings* by pressing **Ctrl+Shift+P** to open the *Command Palette*.
2. Type *settings* in the search box and select **Preferences: Open User Settings**.



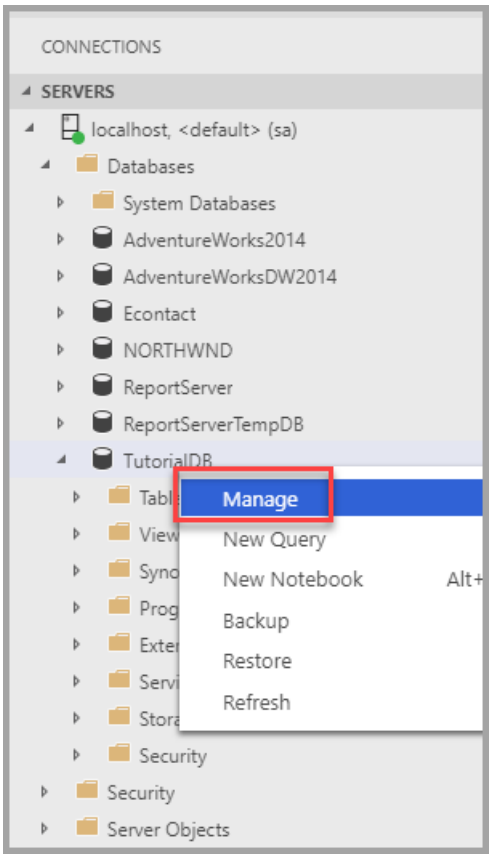
3. Type *dashboard* in the settings search box and locate **dashboard.database.widgets**, and then click *edit in settings.json*.



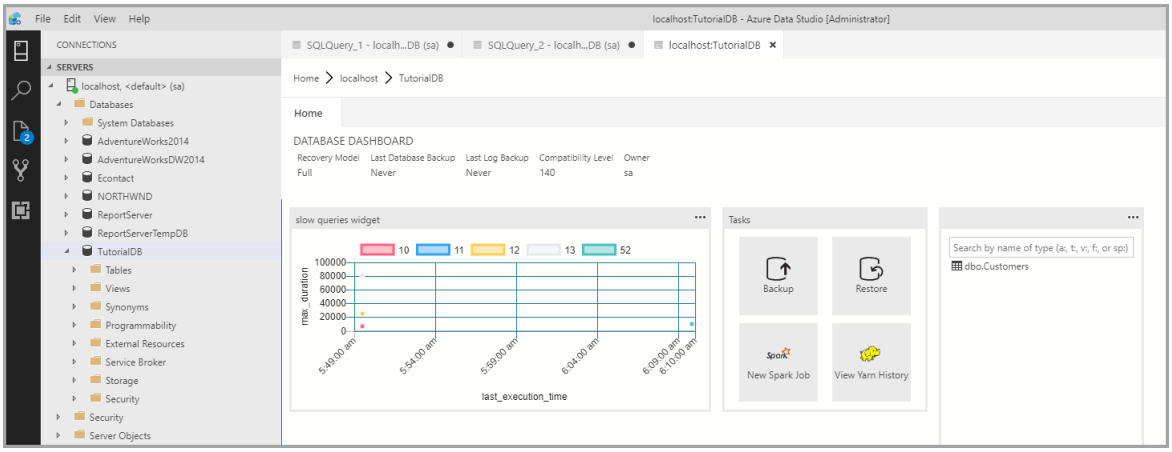
4. In settings.json, add the following code below:

```
"dashboard.database.widgets": [
    {
        "name": "slow queries widget",
        "gridItemConfig": {
            "sizex": 2,
            "sizey": 1
        },
        "widget": {
            "query-data-store-db-insight": null
        }
    },
    {
        "name": "Tasks",
        "gridItemConfig": {
            "sizex": 1,
            "sizey": 1
        },
        "widget": {
            "tasks-widget": {}
        }
    },
    {
        "gridItemConfig": {
            "sizex": 1,
            "sizey": 2
        },
        "widget": {
            "explorer-widget": {}
        }
    }
]
```

5. Press **Ctrl+S** to save the modified **User Settings**.
6. Open the *Database dashboard* by navigating to **TutorialDB** in the **SERVERS** sidebar, right-click, and select **Manage**.



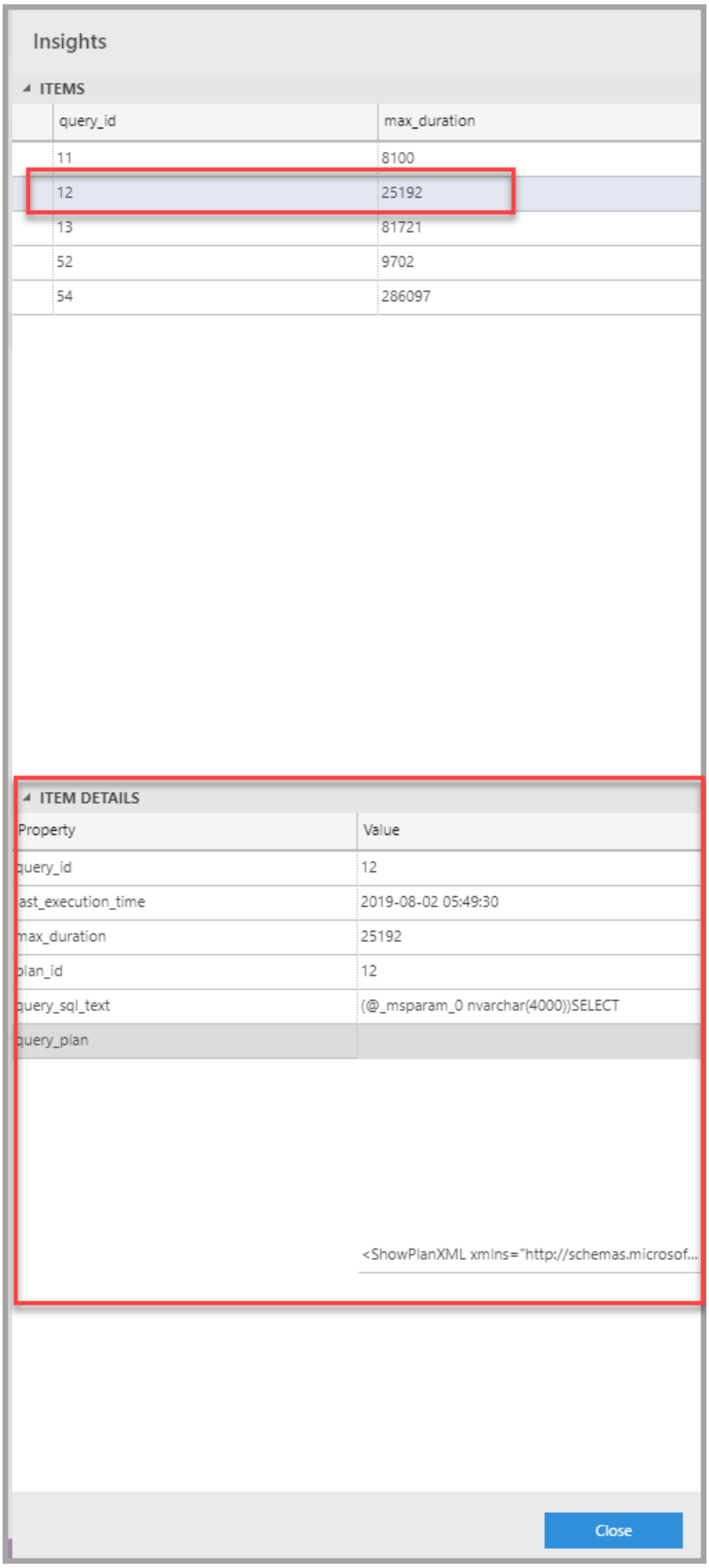
7. The insight widget appears on the dashboard:



View insight details for more information

1. To view additional information for an insight widget, click the ellipses (...) in the upper right, and select **Show Details**.

2. To show more details for an item, select any item in **Chart Data** list.



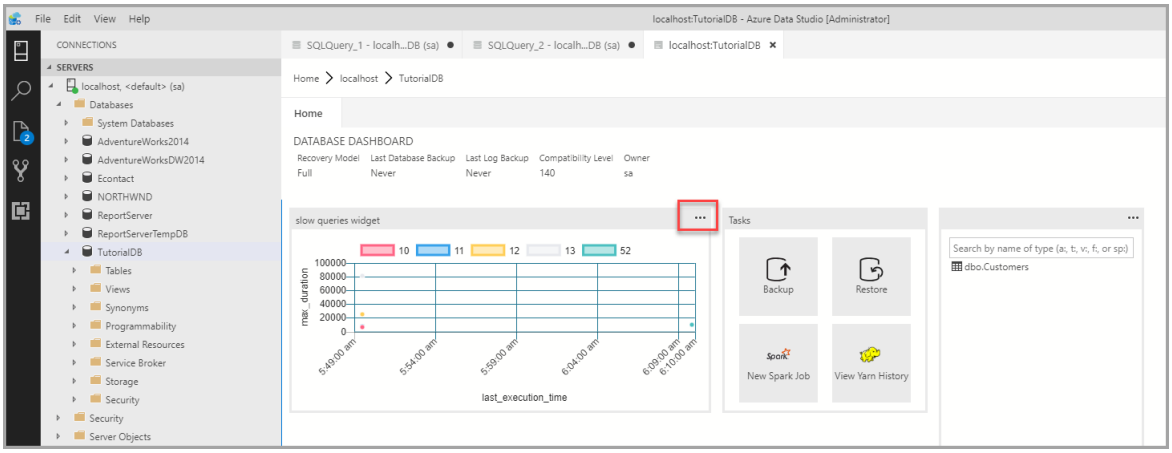
3. Close the **Insights** pane.

View the query plan

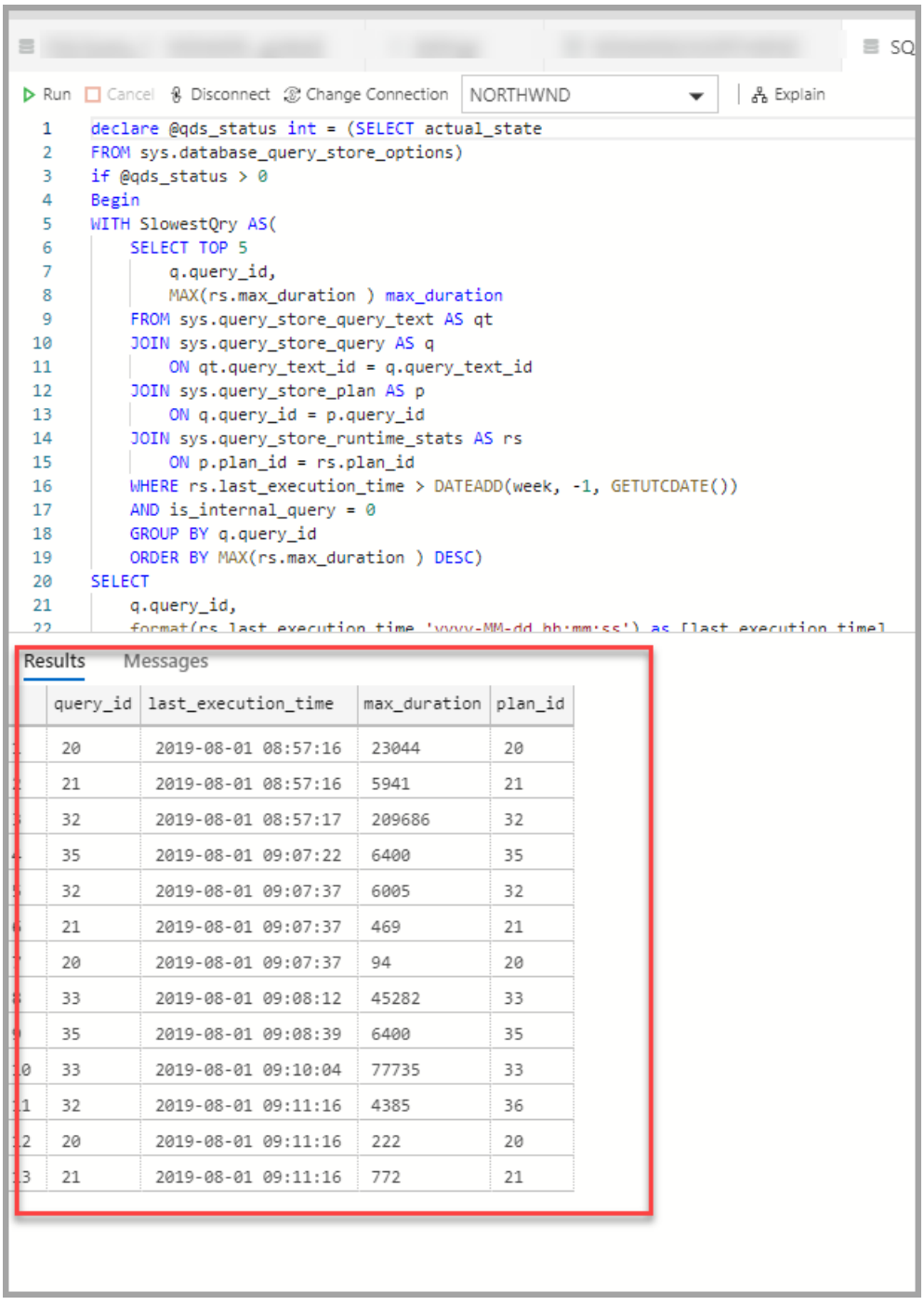
1. Right-click on the **TutorialDB** database and select *Manage*

2. From the *slow query widget* To view additional information for an insight widget, click the ellipses (...) in

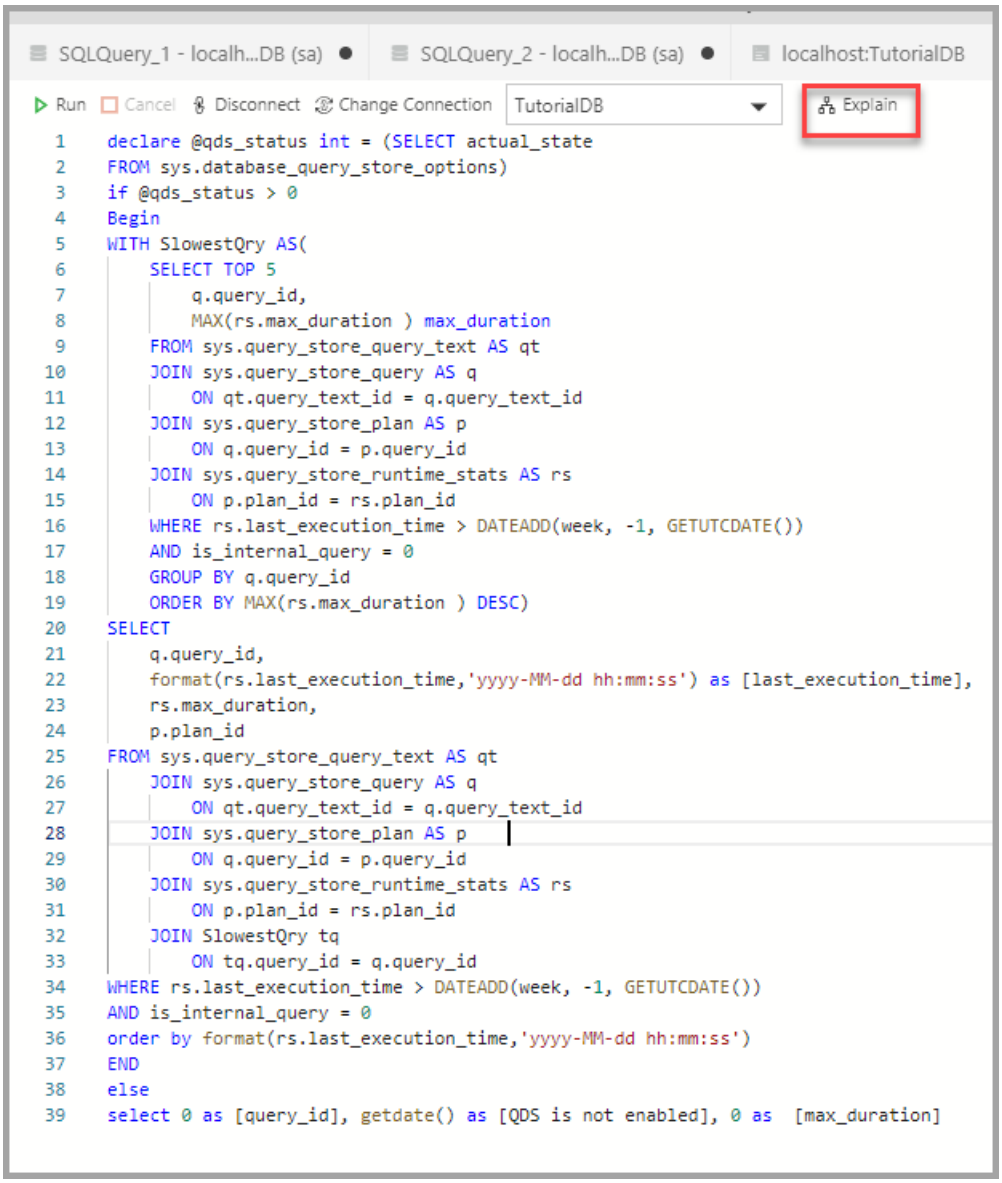
the upper right, and select **Run Query**.



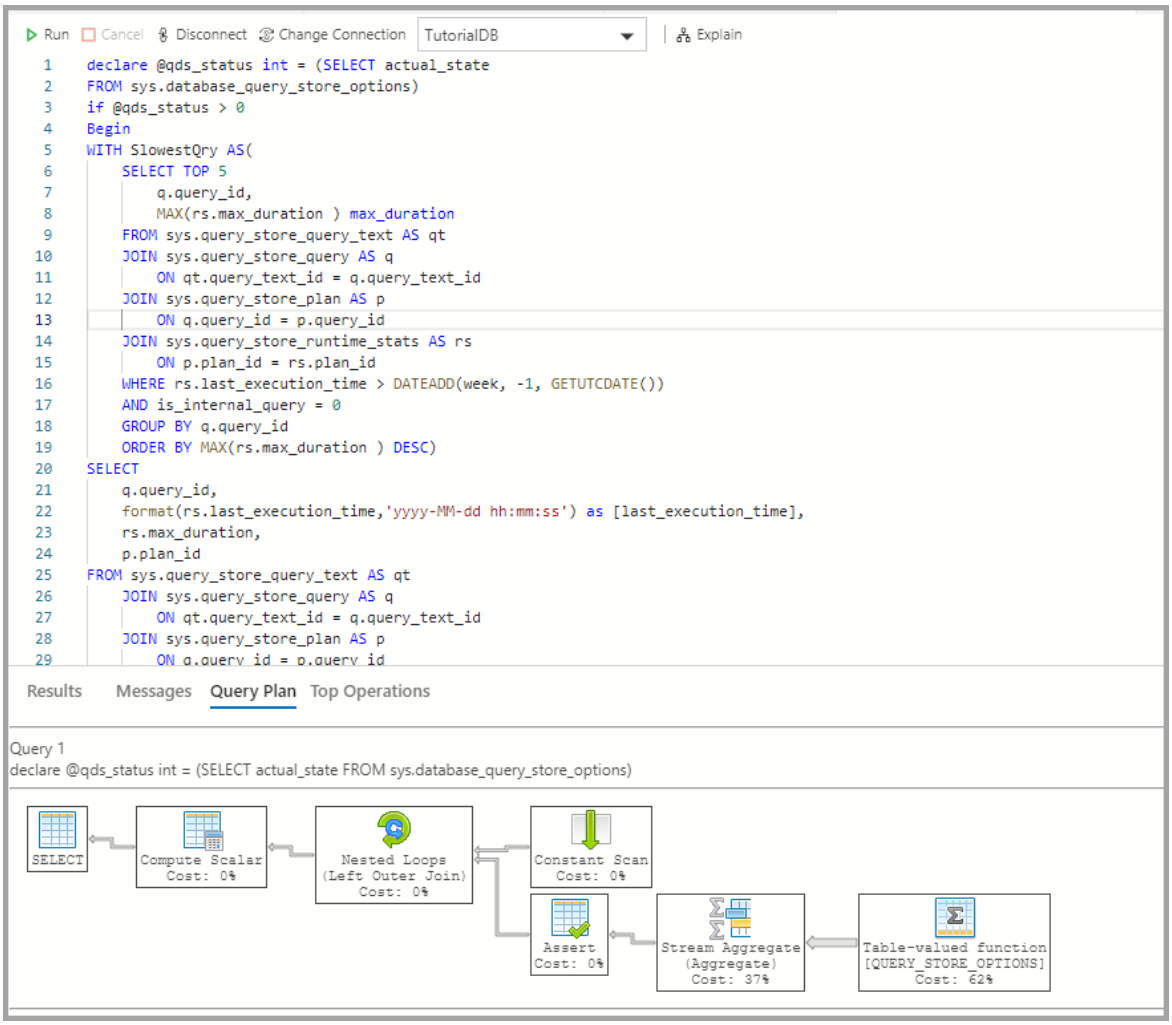
3. Now you should see a new query window with the results.



4. Click **Explain**.



5. View the query's execution plan:



Next steps

In this tutorial, you learned how to:

- Enable Query Store on a database
- Add an insight widget to the database dashboard
- View details about the database's slowest queries
- View query execution plans for the slow queries

To learn how to enable the **table space usage** sample insight, complete the next tutorial:

Enable the table space sample insight widget

Tutorial: Enable the table space usage sample insight widget using Azure Data Studio

11/2/2020 • 2 minutes to read • Edit Online

This tutorial demonstrates how to enable an insight widget on the database dashboard, providing an at-a-glance view about the space usage for all tables in a database. During this tutorial, you learn how to:

- Quickly turn on an insight widget using a built-in insight widget example
- View the details of table space usage
- Filter data and view label detail on an insight chart

Prerequisites

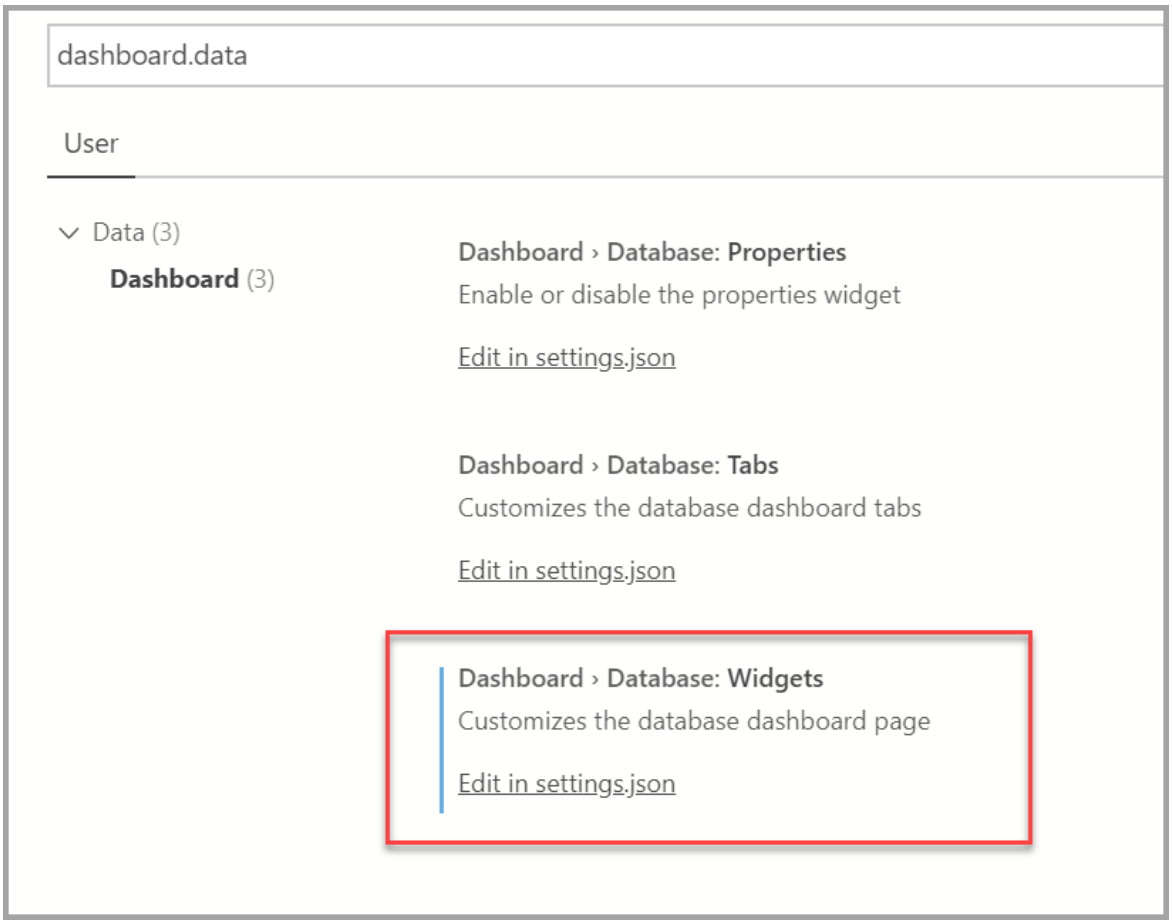
This tutorial requires the SQL Server or Azure SQL Database *TutorialDB*. To create the *TutorialDB* database, complete one of the following quickstarts:

- Connect and query SQL Server using Azure Data Studio
- Connect and query Azure SQL Database using Azure Data Studio

Turn on a management insight on Azure Data Studio's database dashboard

Azure Data Studio has a built-in sample widget to monitor the space used by tables in a database.

1. Open *User Settings* by pressing **Ctrl+Shift+P** to open the *Command Palette*.
2. Type *settings* in the search box and select **Preferences: Open User Settings**.
3. Type *dashboard* in Settings Search input box and locate **dashboard.database.widgets**.
4. To customize the **dashboard.database.widgets** settings, you need to edit the **dashboard.database.widgets** entry in the **USER SETTINGS** section.

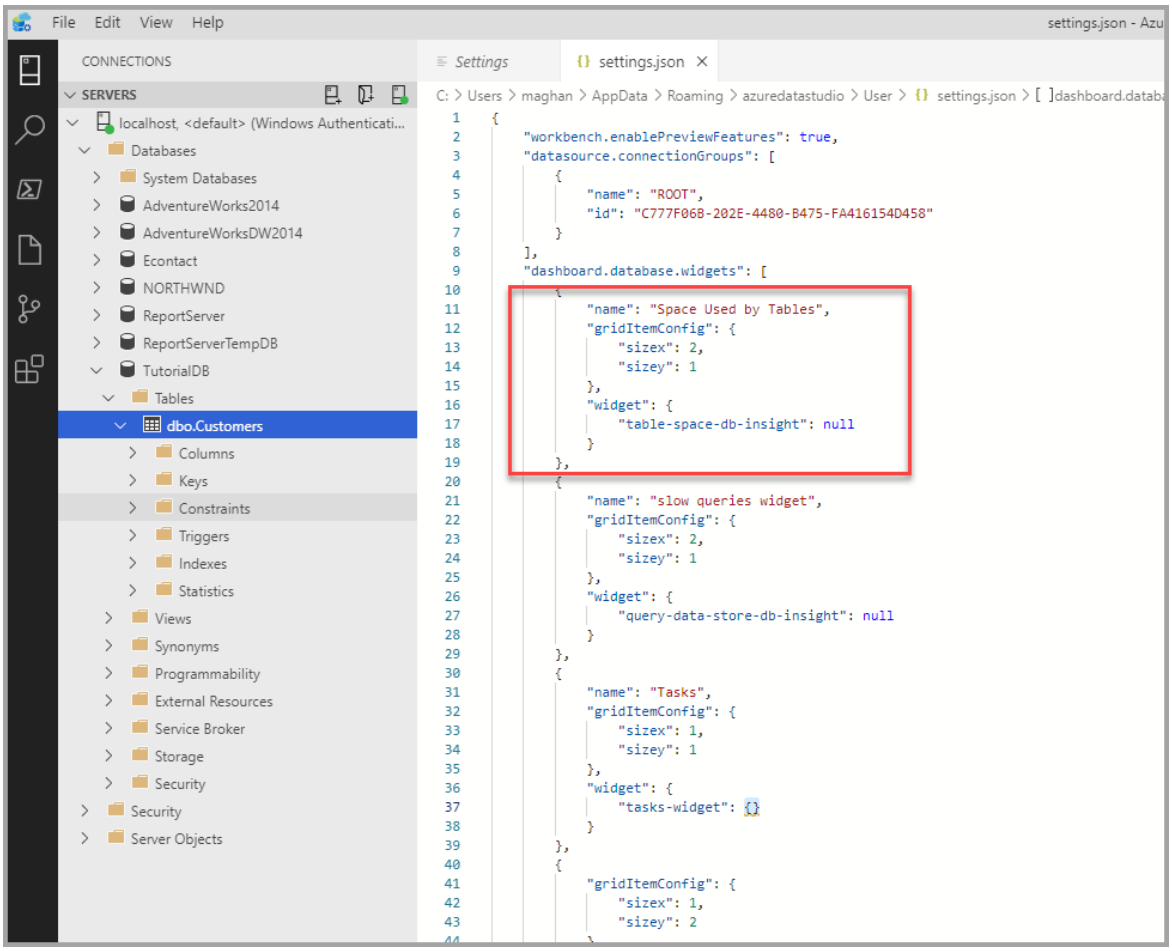


If there is no **dashboard.database.widgets** in the **USER SETTINGS** section, hover over the **dashboard.database.widgets** text in the DEFAULT SETTINGS column and click the *gear* icon that appears to the left of the text and click **Copy as Setting JSON**. If the pop-up says **Replace in Settings**, don't click it! Go to the **USER SETTINGS** column to the right and locate the **dashboard.database.widgets** section and advance to the next step.

5. In the **dashboard.database.widgets** section, add the following lines:

```
{
    "name": "Space Used by Tables",
    "gridItemConfig": {
        "sizex": 2,
        "sizey": 1
    },
    "widget": {
        "table-space-db-insight": null
    }
},
```

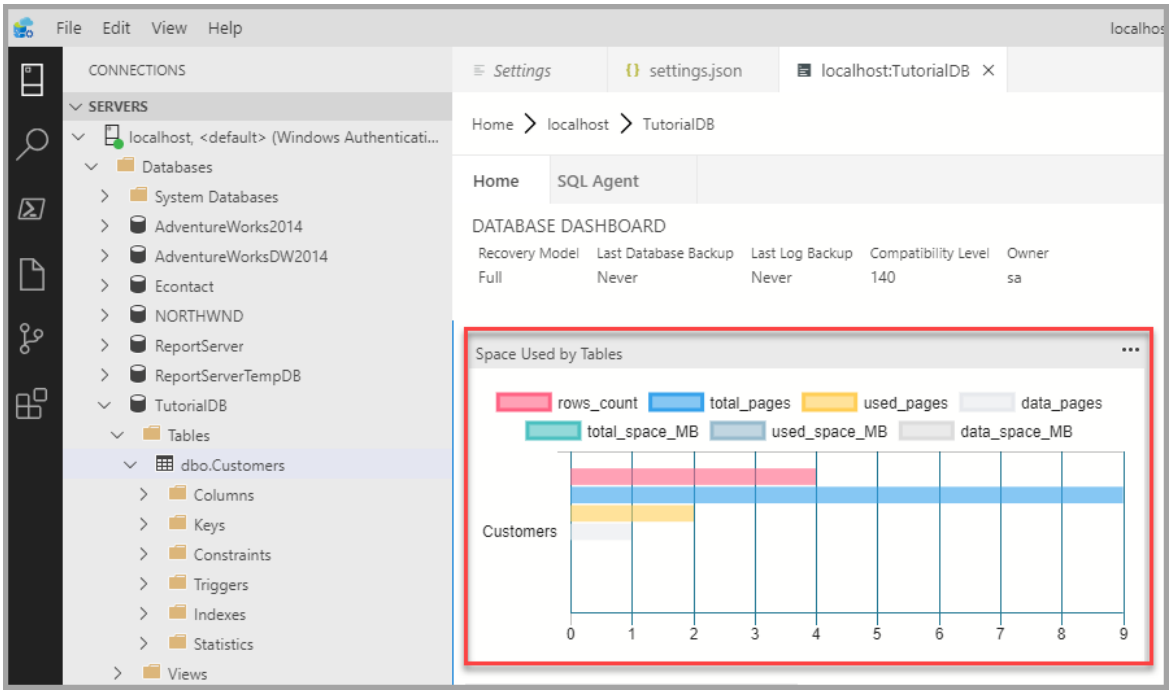
The **dashboard.database.widgets** section should look similar to the following image:



6. Press **Ctrl+S** to save the settings.

7. Open database dashboard by right-clicking **TutorialDB** and click **Manage**.

8. View the *table space* insight widget as shown in the following image:



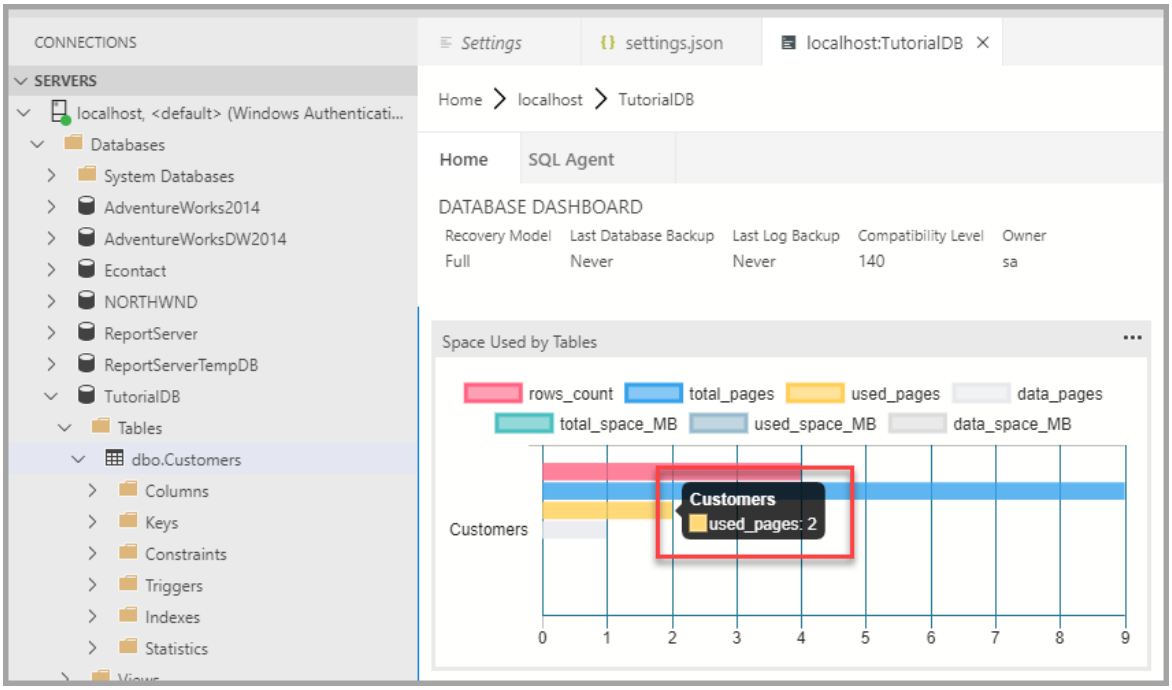
Working with the insight chart

Azure Data Studio's insight chart provides filtering and mouse-hover details. To try out the following steps:

1. Click and toggle the *row_count* legend on the chart. Azure Data Studio shows and hides data series as

you toggle a legend on or off.

2. Hover the mouse pointer over the chart. Azure Data Studio shows more information about the data series label and its value as shown in the following screenshot.



Next steps

In this tutorial, you learned how to:

- Quickly turn on an insight widget using a built-in insight widget sample.
- View the details of table space usage.
- Filter data and view label detail on an insight chart

To learn how to build a custom insight widget, complete the next tutorial:

Build a custom insight widget

Tutorial: Build a custom insight widget

11/2/2020 • 2 minutes to read • Edit Online

This tutorial demonstrates how to use your own insight queries to build custom insight widgets.

During this tutorial you learn how to:

- Run your own query and view it in a chart
- Build a custom insight widget from the chart
- Add the chart to a server or database dashboard
- Add details to your custom insight widget

Prerequisites

This tutorial requires the SQL Server or Azure SQL Database *TutorialDB*. To create the *TutorialDB* database, complete one of the following quickstarts:

- Connect and query SQL Server using Azure Data Studio
- Connect and query Azure SQL Database using Azure Data Studio

Run your own query and view the result in a chart view

In this step, run a sql script to query the current active sessions.

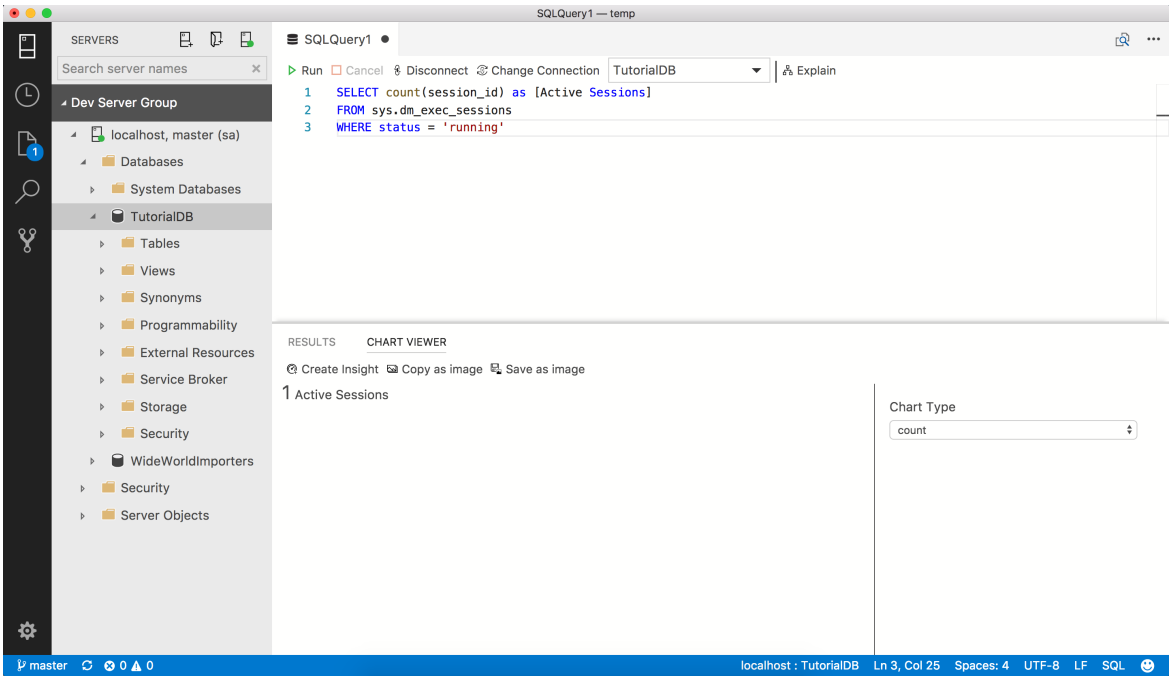
1. To open a new editor, press **Ctrl+N**.
2. Change the connection context to **TutorialDB**.
3. Paste the following query into the query editor:

```
SELECT count(session_id) as [Active Sessions]
FROM sys.dm_exec_sessions
WHERE status = 'running'
```

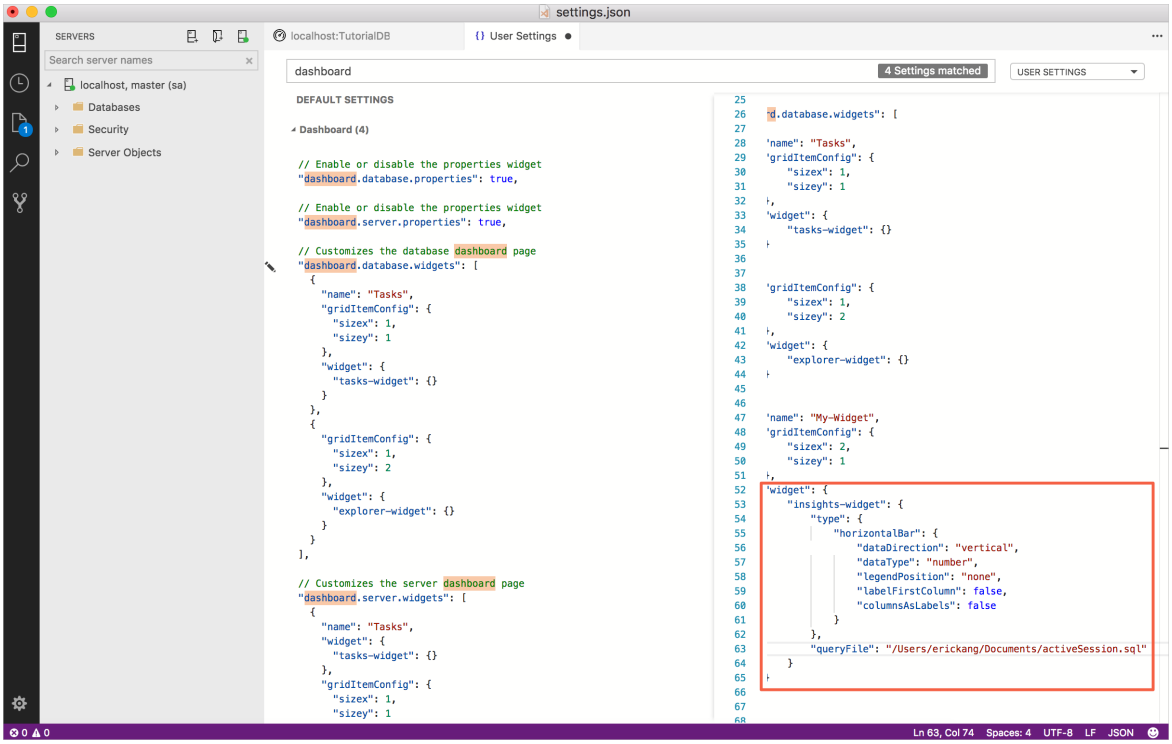
4. Save the query in the editor to a *.sql file. For this tutorial, save the script as *activeSession.sql*.
5. To execute the query, press **F5**.
6. After the query results are displayed, click **View as Chart**, then click the **Chart Viewer** tab.
7. Change **Chart Type** to **count**. These settings render a count chart.

Add the custom insight to the database dashboard

1. To open the insight widget configuration, click **Create Insight** on *Chart Viewer*.



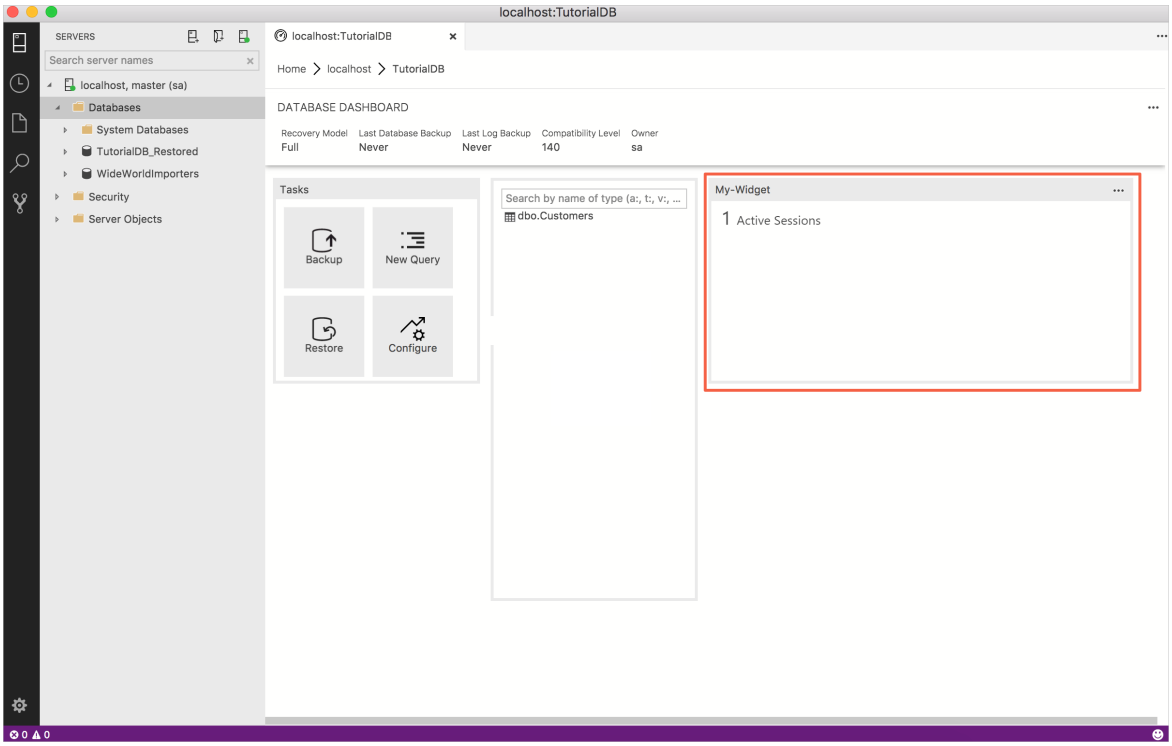
2. Copy the insight configuration (the JSON data).
3. Press **Ctrl+Comma** to open *User Settings*.
4. Type *dashboard* in *Search Settings*.
5. Click **Edit** for *dashboard.database.widgets*.



6. Paste the insight configuration JSON into *dashboard.database.widgets*. Database dashboard settings looks like the following:

```
"dashboard.database.widgets": [
    {
        "name": "My-Widget",
        "gridItemConfig": {
            "sizex": 2,
            "sizey": 1
        },
        "widget": {
            "insights-widget": {
                "type": {
                    "count": {
                        "dataDirection": "vertical",
                        "dataType": "number",
                        "legendPosition": "none",
                        "labelFirstColumn": false,
                        "columnsAsLabels": false
                    }
                },
                "queryFile": "{your file folder}/activeSession.sql"
            }
        }
    }
]
```

7. Save the *User Settings* file and open the *TutorialDB* database dashboard to see the active sessions widget:



Add details to custom insight

1. To open a new editor, press **Ctrl+N**.
2. Change the connection context to **TutorialDB**.
3. Paste the following query into the query editor:

```
 SELECT session_id AS [SID], login_time AS [Login Time], host_name AS [Host Name], program_name AS
[Program Name], login_name AS [Login Name]
 FROM sys.dm_exec_sessions
 WHERE status = 'running'
```

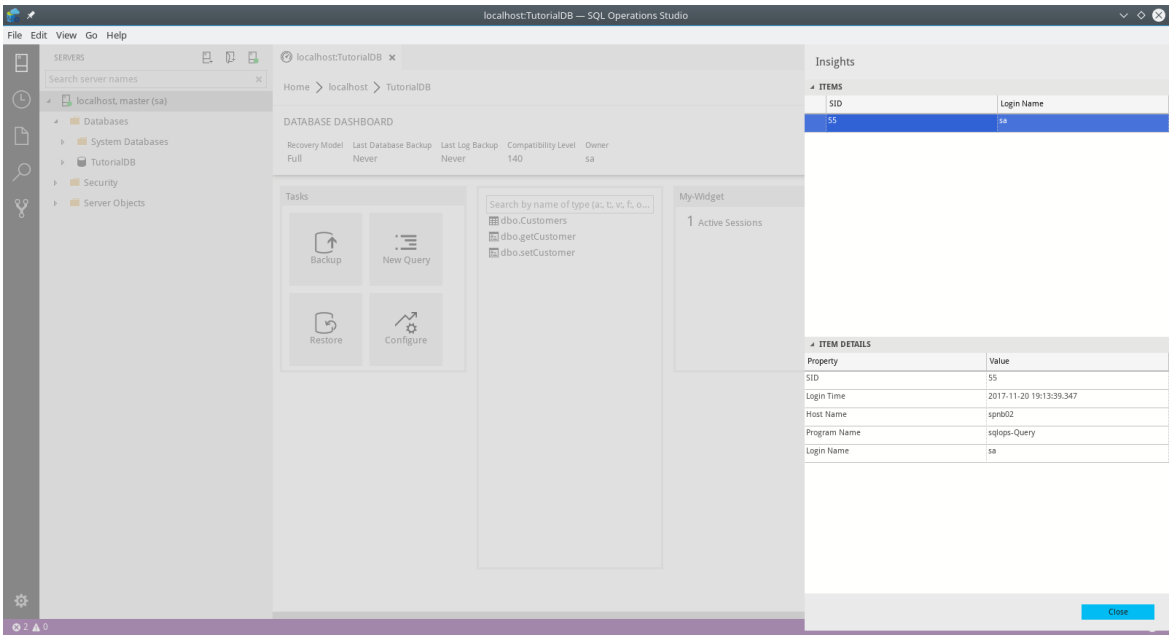
4. Save the query in the editor to a *.sql file. For this tutorial, save the script as *activeSessionDetail.sql*.

5. Press **Ctrl+Comma** to open *User Settings*.

6. Edit the existing *dashboard.database.widgets* node in your settings file:

```
"dashboard.database.widgets": [
    {
        "name": "My-Widget",
        "gridItemConfig": {
            "sizex": 2,
            "sizey": 1
        },
        "widget": {
            "insights-widget": {
                "type": {
                    "count": {
                        "dataDirection": "vertical",
                        "dataType": "number",
                        "legendPosition": "none",
                        "labelFirstColumn": false,
                        "columnsAsLabels": false
                    }
                },
                "queryFile": "{your file folder}/activeSession.sql",
                "details": {
                    "queryFile": "{your file folder}/activeSessionDetail.sql",
                    "label": "SID",
                    "value": "Login Name"
                }
            }
        }
    }
]
```

7. Save the *User Settings* file and open the *TutorialDB* database dashboard. Click the ellipsis (...) button next to *My-Widget* to show the details:



Next steps

In this tutorial, you learned how to:

- Run your own query and view it in a chart

- Build a custom insight widget from the chart
- Add the chart to a server or database dashboard
- Add details to your custom insight widget

To learn how to backup and restore databases, complete the next tutorial:

Backup and restore databases.

Tutorial: Back up and restore databases using Azure Data Studio

11/2/2020 • 2 minutes to read • Edit Online

In this tutorial, you learn how to use Azure Data Studio to:

- Back up a database.
- View the backup status.
- Generate the script used to perform the backup.
- Restore a database.
- View the status of the restore task.

Prerequisites

This tutorial requires SQL Server *TutorialDB*. To create the TutorialDB database, complete the following quickstart:

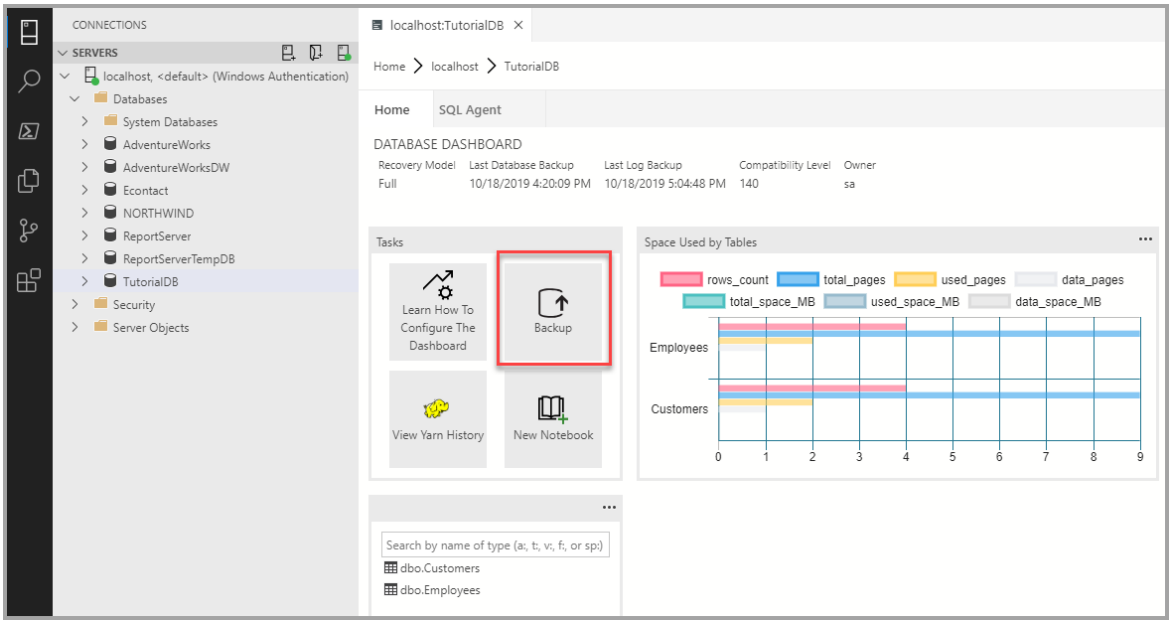
- Use Azure Data Studio to connect and query SQL Server

This tutorial requires a connection to a SQL Server database. Azure SQL Database has automated backups, so Azure Data Studio doesn't perform Azure SQL Database backup and restore. For more information, see Learn about automatic SQL Database backups.

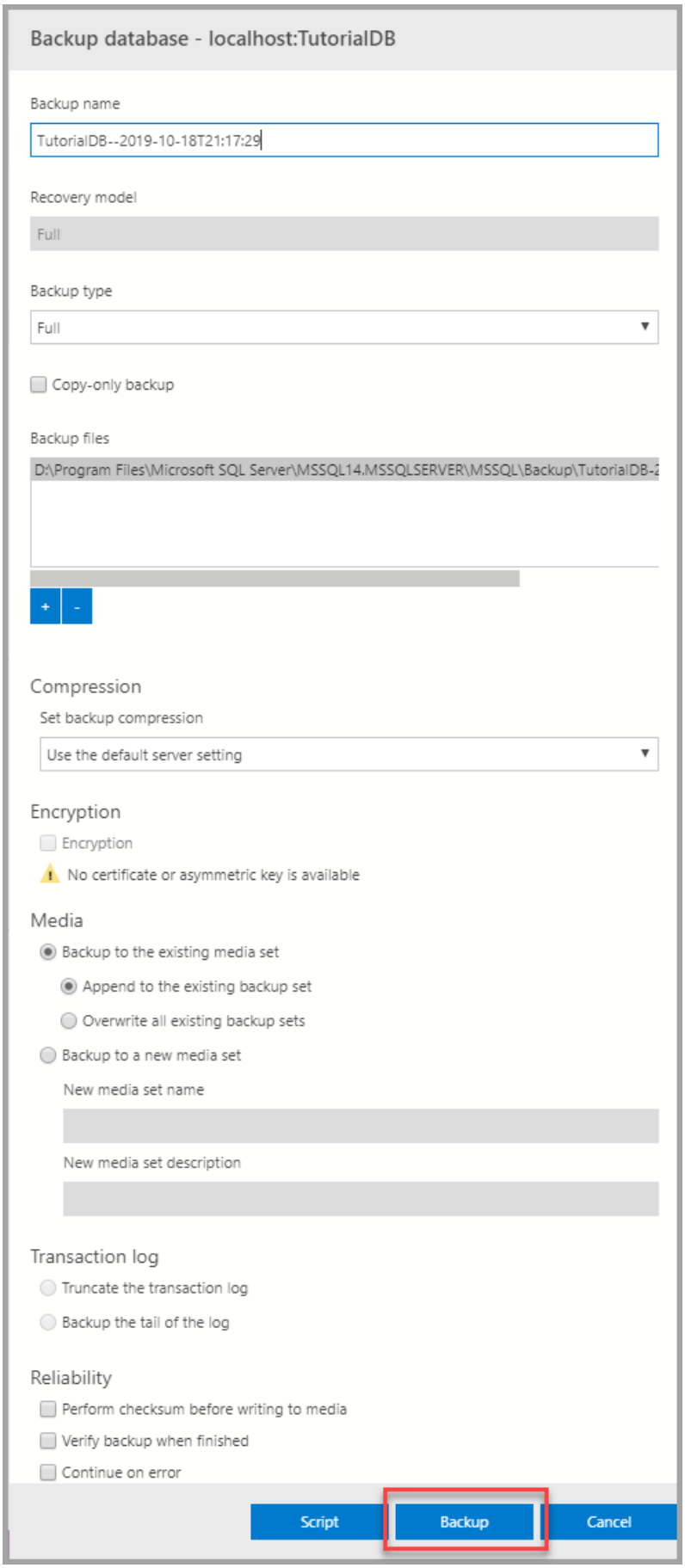
Back up a database

1. Open the TutorialDB database dashboard by opening the **SERVERS** sidebar. Then select **Ctrl+G**, expand **Databases**, right-click **TutorialDB**, and select **Manage**.

2. Open the **Backup database** dialog box by selecting **Backup** on the **Tasks** widget.



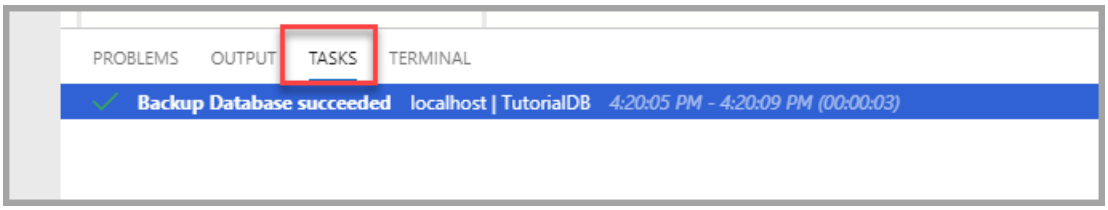
3. This tutorial uses the default backup options, so select **Backup**.



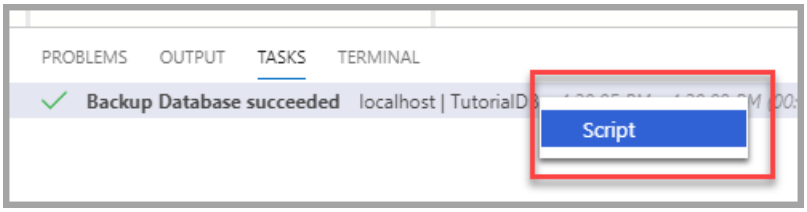
After you select **Backup**, the **Backup database** dialog box disappears and the backup process begins.

View the backup status and the backup script

1. The **Task History** pane appears, or select **Ctrl+T** to open it.

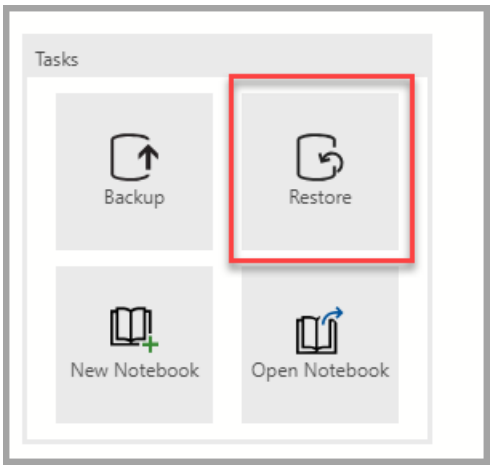


2. To view the backup script in the editor, right-click **Backup Database succeeded** and select **Script**.

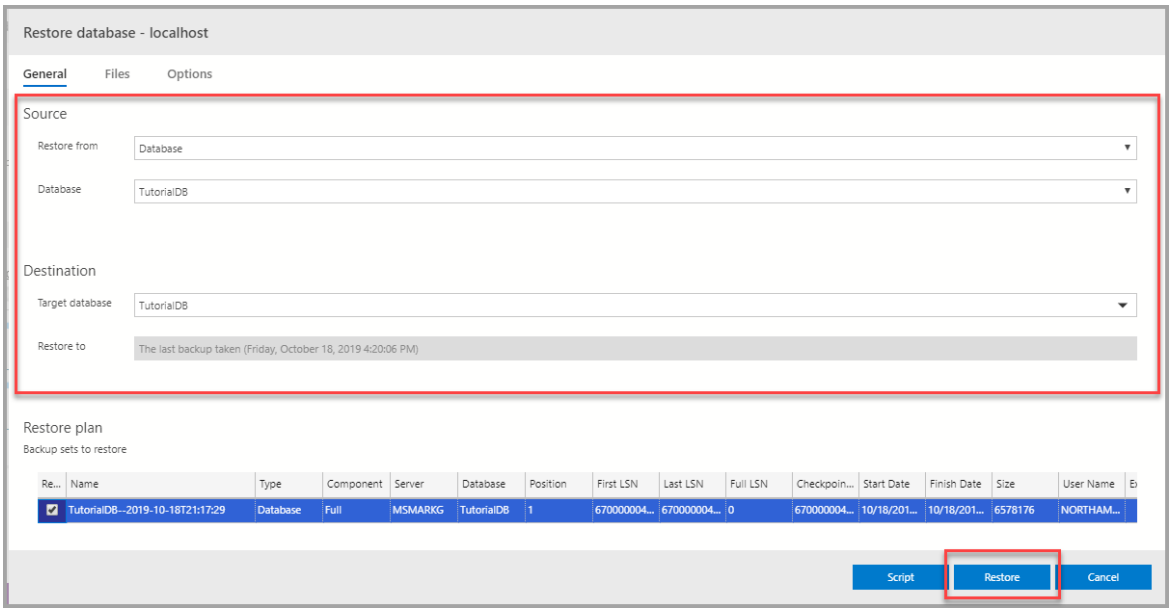


Restore a database from a backup file

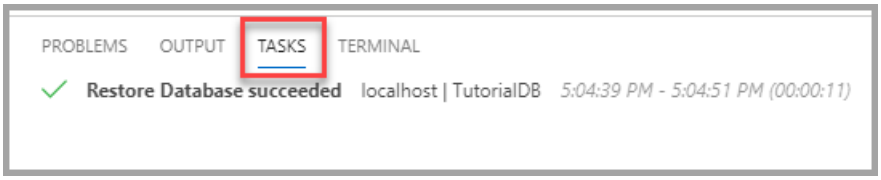
1. Open the **SERVERS** sidebar by selecting **Ctrl+G**. Then right-click your server, and select **Manage**.
2. Open the **Restore database** dialog box by selecting **Restore** on the **Tasks** widget.



3. Select **Backup file** in the **Restore from** box.
4. Select the ellipses (...) in the **Backup file path** box, and select the latest backup file for *TutorialDB*.
5. Enter **TutorialDB_Restored** in the **Target database** box in the **Destination** section to restore the backup file to a new database. Then select **Restore**.



6. To view the status of the restore operation, select **Ctrl+T** to open the **Task History**.



Extend the functionality of Azure Data Studio

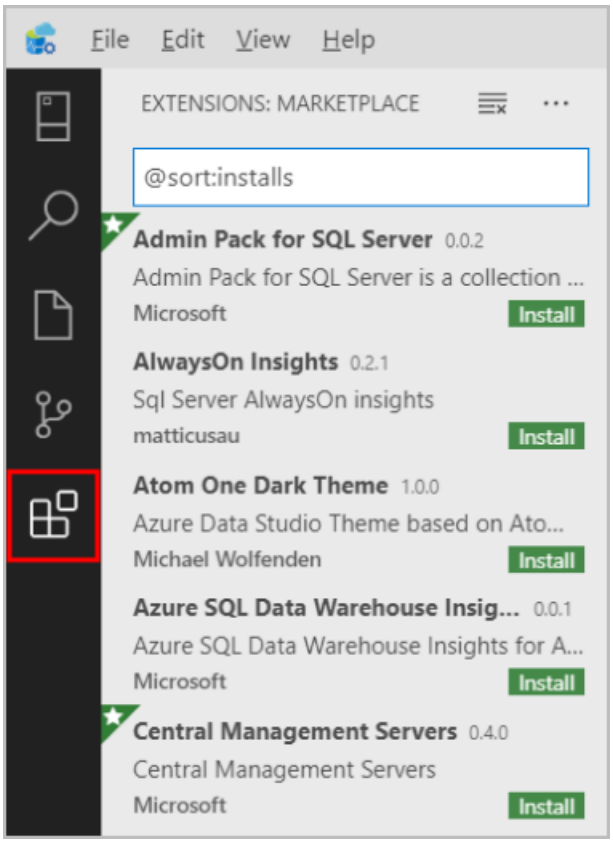
11/2/2020 • 4 minutes to read • Edit Online

Extensions in Azure Data Studio provide an easy way to add more functionality to the base Azure Data Studio installation.

Extensions are provided by the Azure Data Studio team (Microsoft), as well as the third-party community (you!). For more information about creating extensions, see Extension authoring.

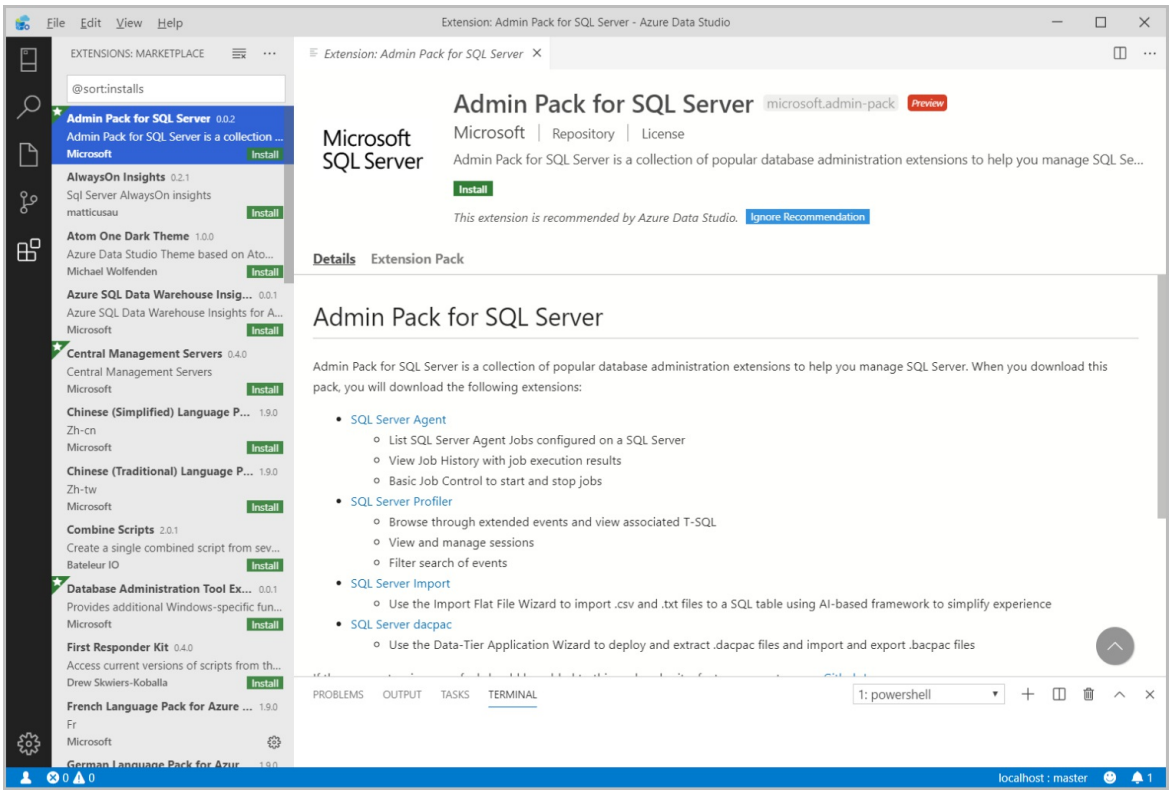
Add Azure Data Studio extensions

1. Access the available extensions by selecting the Extensions Icon, or by selecting **Extensions** in the **View** menu. You can use the **View: Show Extensions** command, available in the **Command Palette** (F1 or `Ctrl+Shift+P`).



   You can also quickly access the extensions manager by pressing `Ctrl+Shift+X` (Windows/Linux) or `Command+Shift+X` (Mac).

2. Select an available extension to view its details.



3. Select the extension you want and **Install** it.

4. Once installed, **Reload** to enable the extension in Azure Data Studio (only required when installing an extension for the first time).

If you're having problems accessing the Extensions Manager on Azure Data Studio, you can download the extension you need on our GitHub Wiki.

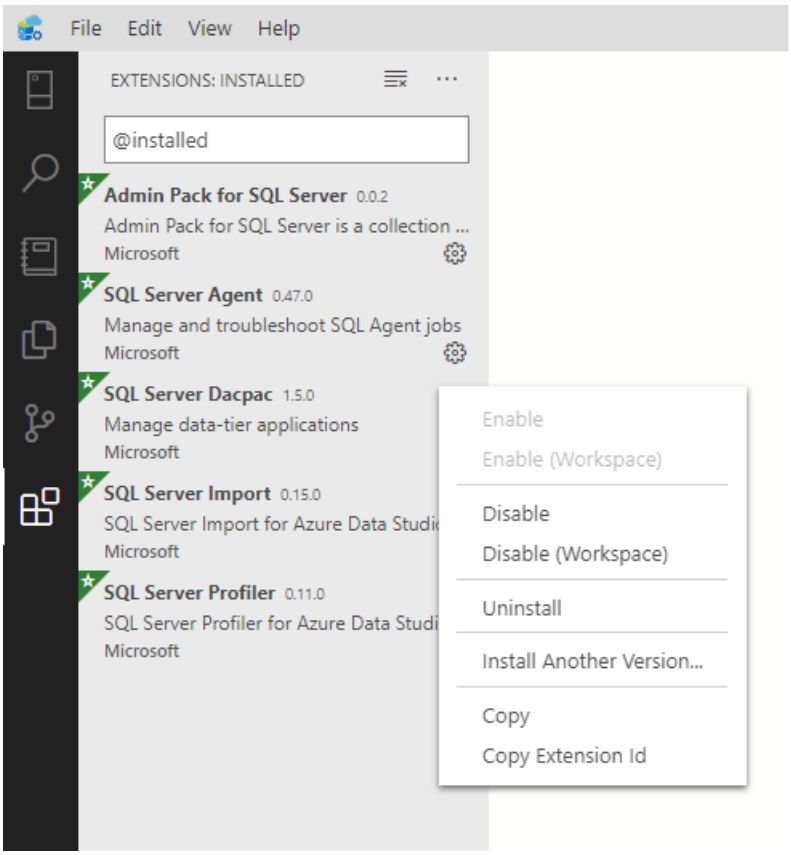
Manage extensions

List installed extensions

The default Extensions view shows the extensions that are currently enabled, all extensions that are recommended for you, and a collapsed container of all currently disabled extensions. The **Extensions: Show Installed Extensions** command, available in the **Command Palette** or the **More Actions** `(...)` drop-down menu, shows a list of all installed extensions, including disabled extensions.

Uninstall an extension

To uninstall an extension, click the gear icon on the right of an extension entry and choose **Uninstall** from the drop-down menu. This uninstalls the selected extension and will prompt you to reload Azure Data Studio.

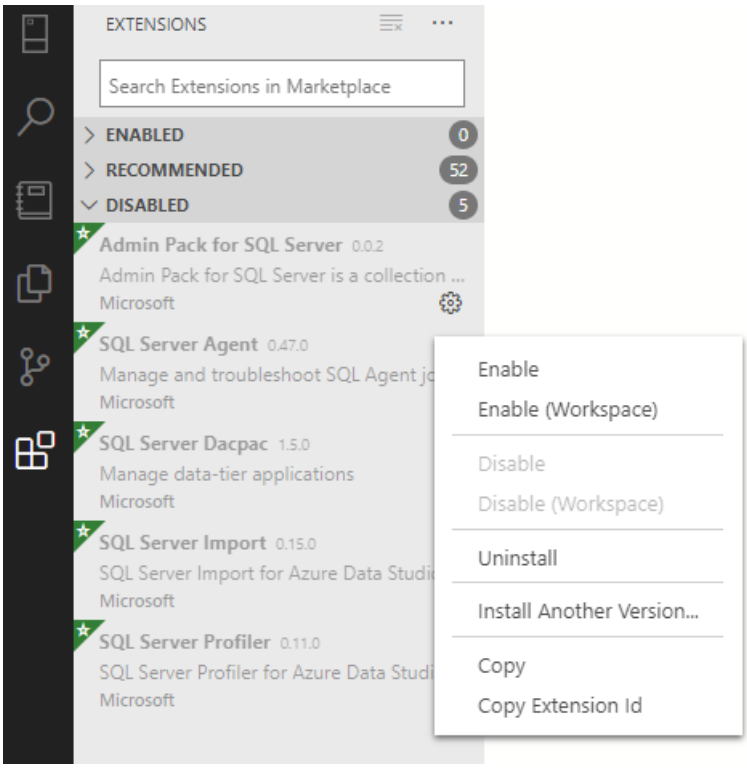


Disable an extension

You may temporarily disable an extension instead of permanently removing an extension. You can either disable an extension across all Azure Data Studio sessions (**Disable**) or just for your current Workspace (**Disable (Workspace)**). You can also disable all of your currently installed extensions through the **Command Palette** with the commands **Extensions: Disable All Extensions** and **Extensions: Disable All Extensions (Workspace)**.

Enable an extension

If an extension has been disabled, it will be in the **Disabled** section of the extension list and marked as *Disabled*. You can re-enable it with the **Enable** or **Enable (Workspace)** commands in the drop-down menu. The **Command Palette** also lets you enable all extensions with the commands **Extensions: Enable All Extensions** and **Extensions: Enable All Extensions (Workspace)**.



Updating an extension

Azure Data Studio automatically checks for and installs updates for any of your installed extensions. If you would like to turn off the auto-update feature, you can disable auto-update with the **Extensions: Disable Auto Updating Extensions** command.

To manually update an extension, you can check for extension updates with the **Extensions: Show Outdated Extensions** command which searches through your extension list using the `@outdated` filter. This will show any available updates for all currently installed extensions. Click the **Update** button on an outdated extension and the update will be installed. You will then be prompted to reload Azure Data Studio. You can also update all of your outdated extensions simultaneously with the **Extensions: Update All Extensions** command.

The **Extensions: Check for Extensions Updates** command is another way to check which of your extensions have updates available.

Install from a VSIX

You can manually install an Azure Data Studio extension packaged in a `.vsix` file using the **Install from VSIX** command in the Extensions view command drop-down, or the **Extensions: Install from VSIX command** in the Command Palette and point to the extension's `.vsix` file.

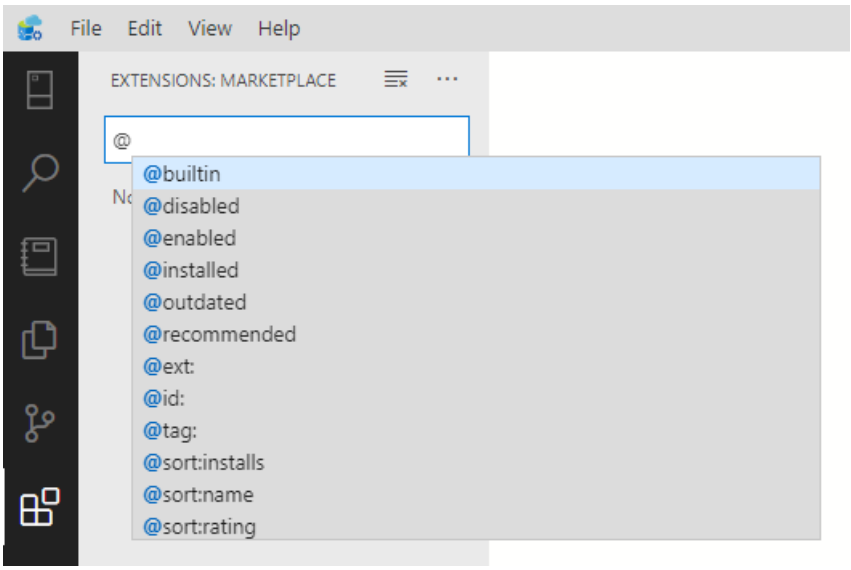
Access installed Azure Data Studio extensions

Each extension enhances your experience in Azure Data Studio in a different way. As a result, the entry point for extensions can vary. Refer to your installed extension's individual documentation for information on how its features can be accessed once it's installed.

Extensions view filters

The Extensions view search box supports filters to help you find and manage extensions. The commands **Show Installed Extensions** and **Show Recommended Extensions** use filters such as `@installed` and `@recommended` in the search box.

You can see a complete listing of all filters and sort commands by typing @ in the extensions search box and navigating through the suggestions:



Here are the Extensions view filters:

- `@builtin` - Show extensions that come with Azure Data Studio. Grouped by type (Programming Languages, Themes, etc.).
- `@disabled` - Show disabled installed extensions.
- `@enabled` - Show enabled installed extensions. Extensions can be individually enabled/disabled.
- `@installed` - Show installed extensions.
- `@outdated` - Show outdated installed extensions. A newer version is available on the Marketplace.
- `@recommended` - Show recommended extensions. Grouped as Workspace specific or general use.
- `@category` - Show extensions belonging to specified category. Below are a few of supported categories. For a complete list, type @category and follow the options in the suggestion list:
  - `@category:themes`
  - `@category:formatters`
  - `@category:snippets` These filters can be combined as well. For example, `@installed @category:themes` displays all installed themes.

If no filter is provided, the Extensions view displays currently installed and recommended extensions.

Sorting

You can sort extensions with the `@sort` filter, which can take the following values:

- `installs` - Sort by extension gallery installation count, in descending order.
- `rating` - Sort by extension gallery rating (1-5 stars), in descending order.
- `name` - Sort alphabetically by extension name.

Common questions

Where are extensions installed?

Extensions are installed in a per user extensions folder. Depending on your platform, the location is in the following folder:

- Windows `%USERPROFILE%\.azuredatastudio\extensions`
- macOS `~/.azuredatastudio/extensions`
- Linux `~/.azuredatastudio/extensions`

Azure Arc extension for Azure Data Studio (Preview)

11/2/2020 • 2 minutes to read • Edit Online

The Azure Arc extension (preview) is an extension for creating and managing Azure Arc data services resources.

Key actions include:

- Create a resource
  - Data Controller
  - SQL Managed Instance for Azure Arc
  - PostgreSQL for Azure Arc
- Manage a resource
  - View Data Controller dashboard
  - View SQL Managed Instance for Azure Arc dashboard
  - View PostgreSQL for Azure Arc dashboard
- Run Azure Arc Jupyter Book

Install the extension

- Install the **Azure Data CLI** extension from the gallery.
- Install the **Azure Arc** extension from the gallery.
- Restart Azure Data Studio

Sign in with Azure account

1. Select Accounts on bottom left
2. Select Add Account
3. This action will launch a browser. Sign in to your Azure Account.

Create a resource

This extension supports deployment of Azure Arc data controllers, Postgres for Azure Arc, and SQL Managed Instance for Azure Arc. Deployments can be done through the built-in Deployment wizard.

1. Select Connections viewlet on left Activity Bar
2. Select the three dots and select **New Deployment**
3. Follow prompts to create a new Azure Arc resource.

Manage a resource

After deploying an Azure Arc data controller from azdata, Azure portal, or Azure Data Studio, you can connect to it from Azure Data Studio.

1. Open Connections viewlet on the left activity bar.
2. Expand **Azure Arc controllers**
3. Select **Connect controller**
4. Fill out parameters and connect.

Once connected, you can view the resources deployed on the data controller. You can then right-click and select

Manage to access the resource dashboard.

These dashboards will provide additional information about the resource, including option to open in Azure portal.

Next steps

To learn more about Azure Arc data services, check our documentation.

Data Virtualization extension for Azure Data Studio

4/27/2021 • 7 minutes to read • Edit Online

The Data Virtualization extension for Azure Data Studio provides support for the External Table Wizard with ODBC data sources.

Install the Data Virtualization extension

To install the Data Virtualization extension, visit Extend the functionality of Azure Data Studio.

Changes in release 1.0

- Extension renamed to Data Virtualization.
- Create External Table wizard:
  - Included guided notebooks for virtualization MongoDB and Teradata sources.
  - Added dialog to fill out variables in MongoDB and Teradata virtualization notebooks.

Changes in release 0.16

- Create External Table wizard:
  - Improved error handling when loading tables and views on object-mapping page.

Changes in release 0.15

- Create External Table wizard:
  - Reduced time taken to load table and column information on the object-mapping page.
  - Fixed a bug with loading existing database scoped credentials on the connection details page.
- Create External Table from CSV Files Wizard:
  - Increased default sample size used for PROSE parsing.

Changes in release 0.14.1

- Support for CTP 3.1 data source support

Changes in release 0.12.1

- The **SQL Server big data cluster** connection type has been removed in this release. All functionality previously available from the SQL Server big data cluster connection is now available in the SQL Server connection.
- HDFS browsing can be found under the **Data Services** folder
- For notebooks, the PySpark and other big data kernels work when connected to the SQL Server master instance in your SQL Server big data cluster.
- Create External Table wizard:
  - Support for creating External Table using existing External Data Source.
  - Performance improvements across the wizard.
  - Improved handling of object names with special characters. In some cases, these caused the wizard to fail
  - Reliability improvements for the Object-Mapping page.

- Removed system databases - `DWConfiguration`, `DWDiagnostics`, `DWQueue` - from the databases dropdown.
- Support for setting the External File Format object's name in the **Create External Table from CSV Files** wizard.
- Added a refresh button to the first page of the **Create External Table from CSV Files** wizard.

Release Notes (v0.11.0)

- Jupyter Notebook support, specifically support for the Python3 and Spark kernels, has been moved into Azure Data Studio. This extension is no longer required in order to use Notebooks.
- Multiple bug fixes in the External Data wizards:
  - Oracle type mappings have been updated to match changes shipped in SQL Server 2019 CTP 2.3.
  - Fixed an issue where new schemas typed into the table-mapping controls were being lost.
  - Fixed an issue where checking a Database node in the table-mappings didn't result in all tables and views being checked.

Release Notes (v0.10.2)

SQL Server 2019 support

Support for SQL Server 2019 has been updated. After connecting to a SQL Server Big Data Cluster instance, a new *Data Services* folder appears in the explorer tree. The folder has launch points for actions such as opening a new notebook against the connection, submitting Spark jobs, and working with HDFS. Some actions, such as *Create External Data* over an HDFS file/folder, the *SQL Server 2019* extension must be installed.

Notebook support

We have made significant updates to the notebook user interface. Our focus is on making it easy to read notebooks that are shared with you. This meant removing all outline boxes around cells unless selected or hovered, adding hover support for easy cell-level actions without need to select a cell, and clarifying execution state by adding execution count, an animated *stop running* button and more. We also added keyboard shortcuts for *New Notebook* (`Ctrl+Shift+N`), *Run Cell* (`F5`), *New Code Cell* (`Ctrl+Shift+C`), *New Text Cell* (`Ctrl+Shift+T`). We aim to have all key actions launchable by shortcut so let us know what you're missing!

Other improvements and fixes include:

- The *SQL Server 2019* extension now prompts users to pick an install directory for Python dependencies. It also no longer includes Python in the `.vsix file`, reducing overall extension size. The Python dependencies support Spark and Python3 kernels.
- Support for launching a new notebook from the command line has been added. Launch with the arguments `--command=notebook.command.new --server=myservername` should open a new notebook and connect to this server.
- Performance fixes for notebooks with a large code length in cells. If code cells are over 250 lines, a scrollbar is added.
- Improved `.ipynb` file support. Version 3 or higher is now supported.

  > **NOTE**
  > Saving files updates to version 4 or higher.

- The `notebook.enabled` user setting has been removed now that the built-in Notebook viewer is stable.
- High Contrast theme is now supported with a number of fixes to object layout in this case.

- Fixed #3680 where outputs sometimes showed a number of `,,,` characters incorrectly.
- Fixed #3602 Editor disappears for cells after navigating away from Azure Data Studio.
- Support has been added to use Grid views for the `application/vnd.dataresource+json` output MIME type. This means many notebooks that use this (for example by setting `pd.options.display.html.table_schema` in a Python notebook) have nicer tabular outputs.

Known issues

- When opening a Notebook, the install python dialog appears. Canceling this install results in the Kernels and Attach To dropdowns not showing expected values. The workaround is to complete the Python installation.
- When a notebook is opened with a kernel that isn't supported, the kernels and *attach to* dropdowns causes Azure Data Studio to stop responding. Close Azure Data Studio and ensure you use a kernel that is supported (Python3, Spark | R, Spark | Scala, PySpark, PySpark3).
- Spark UI link fails when using PySpark3 or other Spark kernels against the SQL Server endpoint. As a workaround, select on Spark UI from the Dashboard, or connect using the SQL Server big data cluster connection type as this has the correct Spark UI hyperlink.

Extensibility improvements

A number of improvements that help extenders were added in this release.

- A new `ObjectExplorerNodeProvider` API allows extensions to contribute folders under SQL Server or other Connection nodes. This is how the `Data Services` node is added under SQL Server 2019 instances but could be used to add Monitoring or other folders easily to the UI.
- Two new context key values are available to help show/hide contributions to the dashboard.
  - `mssql:iscluster` indicates if this is a SQL Server 2019 Big Data Cluster
  - `mssql:servermajorversion` has the server version (15 for SQL Server 2019, 14 for SQL Server 2017, and so on). This can help if features should only be shown for SQL Server 2017 or greater, for example.

Release Notes (v0.8.0)

Notebooks:

- Adding cells before / after existing cells are now supported by selecting the "More Actions" cell button
- **Add New Connection** option has been added to the connections in the "Attach To" dropdown
- A **Reinstall Notebook Dependencies** command has been added to assist with Python package updates, and solve cases where install was halted partway through by closing the application. This can be run from the command palette (use `Ctrl/Cmd+Shift+P` and type `Reinstall Notebook Dependencies`)
- The PROSE python package has been updated to 1.1.0 and includes a number of bug fixes. Use the **Reinstall Notebook Dependencies** command to update this package
- A **Clear Output** command is now supported by selecting the **More Actions** cell button
- Fixed the following customer reported issues:
  - Notebook session couldn't start on Windows due to PATH issues
  - Notebook couldn't be started from the root folder of a drive, such as C:\ or D:\
  - #2820 Unable to edit notebooks created from ADS in VS Code
  - Spark UI link now works when running a Spark kernel
  - Renamed "Managed Packages" to "Install Packages"

Create External Data:

- Error messages are copyable and have been separated into a summary and detailed view for easier
- Improved UI layout and improved reliability and error handling

- Fixed the following customer reported issues:
  - Tables with invalid column mappings are shown as disabled and a warning explains the error

Release Notes (v0.7.2)

- Azure Resource Explorer is now built into Azure Data Studio and has been removed from this extension. Thank you for your feedback on this!
- Improved performance of notebooks with many Markdown cells.
- Autosize code cells in Notebook. This still has a minimum size based on the cell toolbar.
- Notify user when installing Notebook dependencies. On Windows in particular this can take a long time, so notifications are now shown in the Tasks view.
- Support reinstalling Notebook dependencies. This is useful if the user previously closed Azure Data Studio partway through installation.
- Support canceling cell execution in Notebook.
- Improved reliability when using Create External Data wizard, specifically when connection errors occur.
- Block use of Create External Data wizard if PolyBase isn't enabled or running in the target server.
- Spelling and naming fixes related to SQL Server 2019 and Create External Data.
- Removed a large number of errors from the Azure Data Studio debug console.

Kusto (KQL) extension for Azure Data Studio (Preview)

11/2/2020 • 3 minutes to read • Edit Online

The Kusto (KQL) extension for Azure Data Studio enables you to connect and query to Azure Data Explorer clusters.

Users can write and run KQL queries and author notebooks with the Kusto kernel complete with IntelliSense.

By enabling the native Kusto (KQL) experience in Azure Data Studio, data engineers, data scientists, and data analysts can quickly observe trends and anomalies against massive amounts of data stored in Azure Data Explorer.

This extension is currently in preview.

Prerequisites

If you don't have an Azure subscription, create a free Azure account before you begin.

The following prerequisites are also required:

- Azure Data Studio installed.
- An Azure Data Explorer cluster and database.

Install the Kusto (KQL) extension

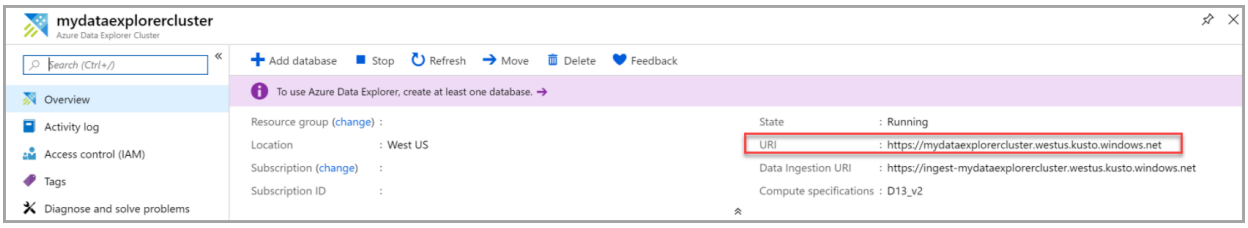
To install the Kusto (KQL) extension in Azure Data Studio, follow the steps below.

1. Open the extensions manager in Azure Data Studio. You can either select the extensions icon or select **Extensions** in the View menu.
2. Type in *Kusto* in the search bar.
3. Select the **Kusto (KQL)** extension and view its details.
4. Select **Install**.

How to connect to an Azure Data Explorer cluster

Find your Azure Data Explorer cluster

Find your Azure Data Explorer cluster in the Azure portal, then find the URI for the cluster.



However, you can get started immediately using the *help.kusto.windows.net* cluster.

For this article, we're using data from the help.kusto.windows.net cluster for samples.

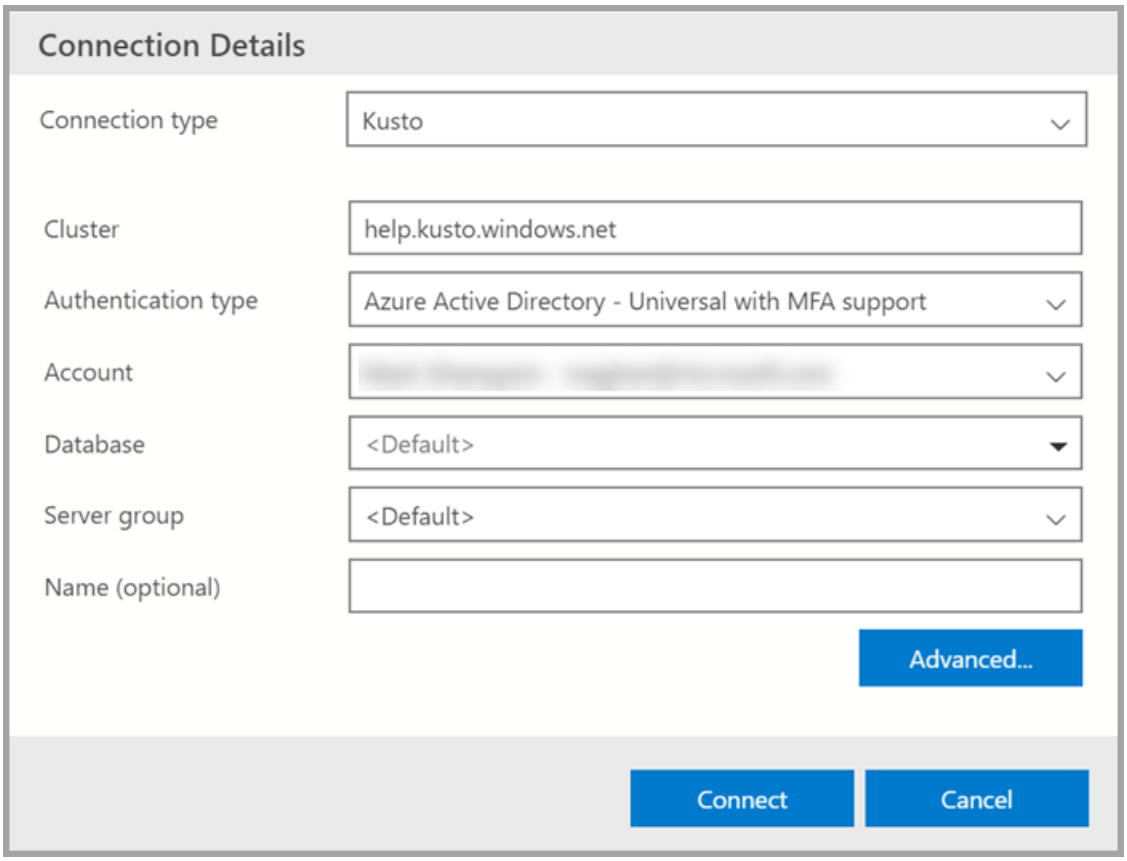
Connection details

To set up an Azure Data Explorer cluster to connect to, follow the steps below.

1. Select **New connection** from the **Connections** pane.
2. Fill in the **Connection Details** information.
    a. For **Connection type**, select *Kusto*.
    b. For **Cluster**, enter in your Azure Data Explorer cluster.

> **NOTE**
> When entering the cluster name, don't include the `https://` prefix or a trailing `/`.

    c. For **Authentication Type**, use the default - *Azure Active Directory - Universal with MFA account*.
    d. For **Account**, use your account information.
    e. For **Database**, use *Default*.
    f. For **Server Group**, use *Default*.
        a. You can use this field to organize your servers in a specific group.
    g. For **Name (optional)**, leave blank.
        a. You can use this field to give your server an alias.



How to query an Azure Data Explorer database in Azure Data Studio

Now that you have set up a connection to your Azure Data Explorer cluster, you can query your database(s) using Kusto (KQL).

To create a new query tab, you can either select **File > New Query**, use *Ctrl + N*, or right-click the database and select **New Query**.

Once you have your new query tab open, then enter your Kusto query.

Here are some samples of KQL queries:

```
StormEvents
| limit 1000
```

```
StormEvents
| where EventType == "Waterspout"
```

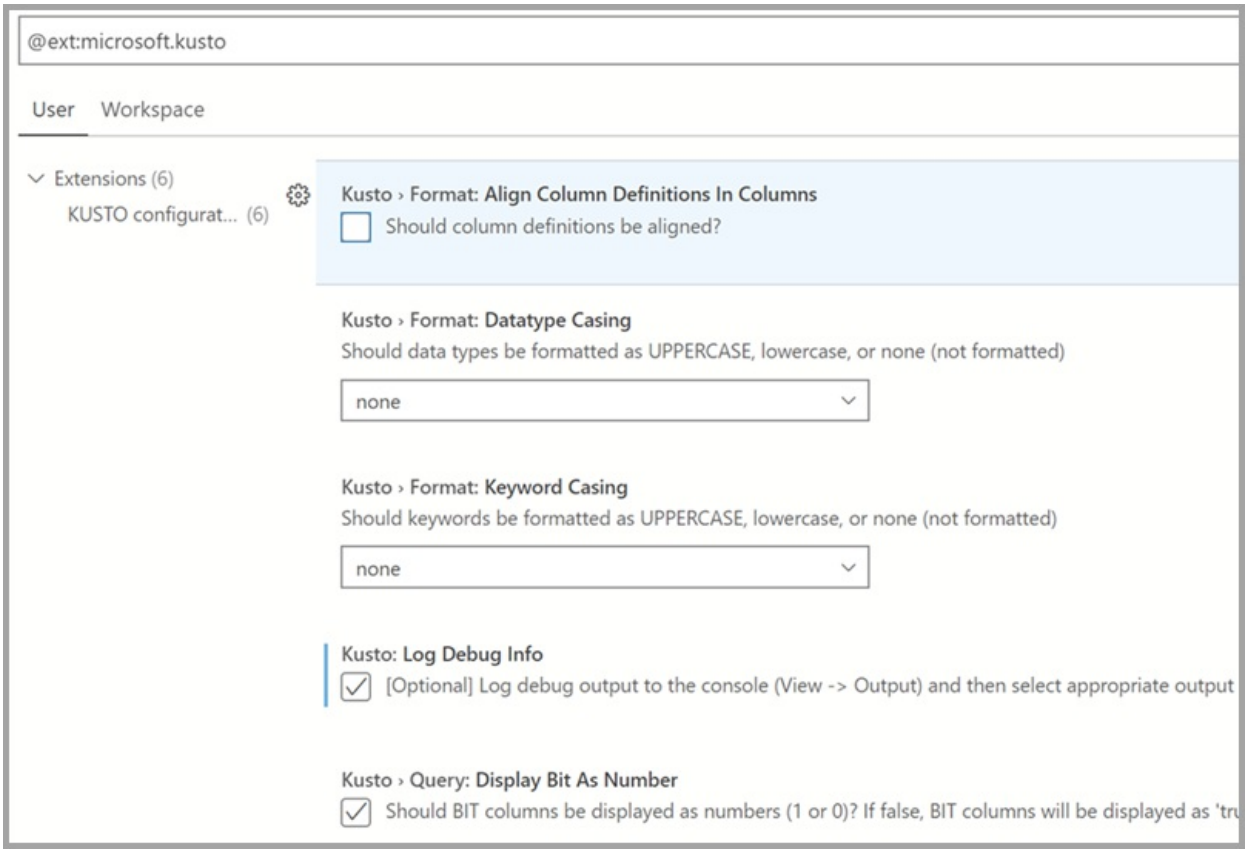
For more information about writing KQL queries, visit Write queries for Azure Data Explorer

View extension settings

To change the settings for the Kusto extension, follow the steps below.

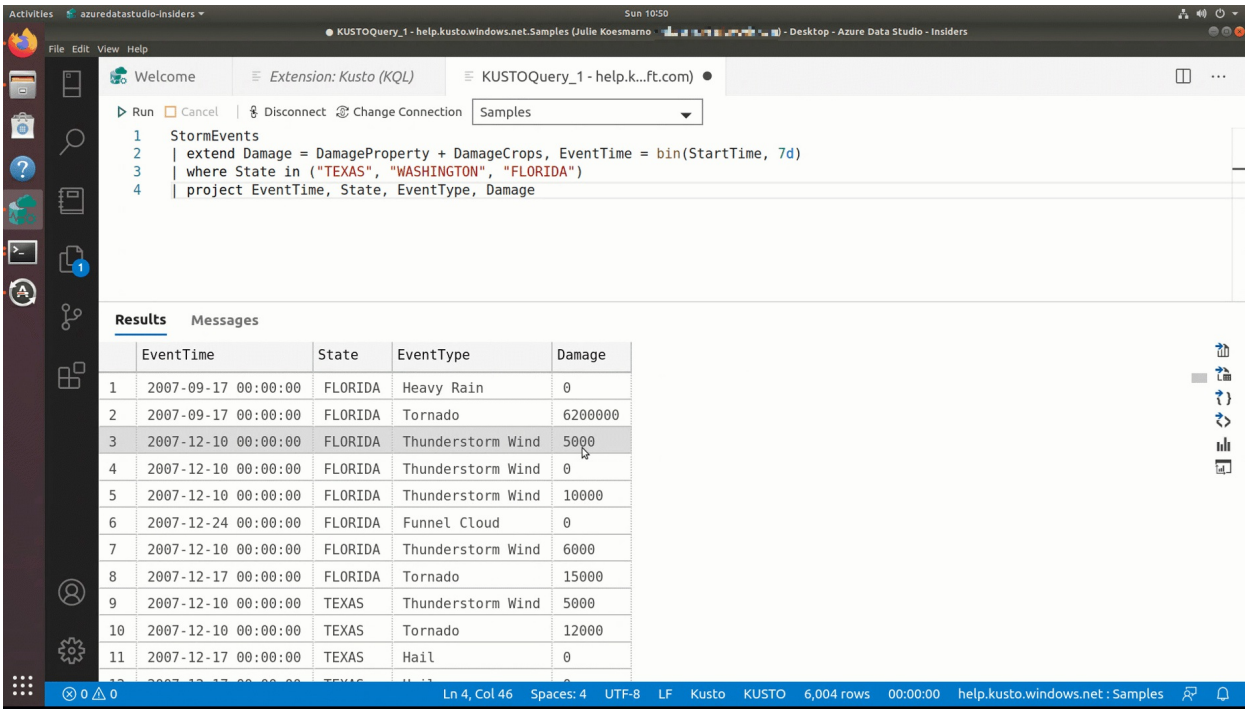
1. Open the extension manager in Azure Data Studio. You can either select the extensions icon or select **Extensions** in the View menu.
2. Find the **Kusto (KQL)** extension.
3. Select the **Manage** icon.
4. Select the **Extension Settings** icon.

The extensions settings look like this:



SandDance visualization

The SandDance extension with the Kusto (KQL) extension in Azure Data Studio bring rich interactive visualization together. From the KQL query result set, select the **Visualizer** button to launch SandDance.



Known issues

DETAILS	WORKAROUND
In Kusto notebook, Changing a database connection on a saved alias connection is stuck after an error in code cell execution	Close and reopen the Notebook, then connect to the right cluster with the database
In Kusto Notebook, changing a database connection on a non-saved alias connection doesn't work	Create a new connection from Connection Viewlet and save it with an alias. Then create a new notebook and connect to the newly saved connection)
In Kusto Notebook, the database dropdown isn't populated when creating a new ADX connection	Create a new connection from Connection Viewlet and save it with an alias. Then create a new notebook and connect to the newly saved connection)

You can file a feature request to provide feedback to the product team.
You can file a bug to provide feedback to the product team.

Next steps

- Create and run a Kusto notebook
- Kqlmagic notebook in Azure Data Studio
- SQL to Kusto cheat sheet
- What is Azure Data Explorer?
- Using SandDance visualizations

Machine Learning extension for Azure Data Studio (Preview)

6/29/2021 • 3 minutes to read • Edit Online

The Machine Learning extension for Azure Data Studio enables you to manage packages, import machine learning models, make predictions, and create notebooks to run experiments for your SQL databases. This extension is currently in preview.

Prerequisites

The following prerequisites need to be installed on the computer you run Azure Data Studio.

- Python 3. Once you have installed Python, you need to specify the local path to a Python installation under Extension Settings. If you have used a Python kernel notebook in Azure Data Studio, the extension will use the path from the notebook by default.
- Microsoft ODBC driver 17 for SQL Server for Windows, macOS, or Linux.
- R 3.5 (optional). Other version than 3.5 is currently not supported. Once you have installed R 3.5, you need to enable R and specify the local path to an R installation under Extension Settings. This is only required if you want to manage R packages in your database.

Trouble installing Python 3 from within ADS?

If you attempt to install Python 3 but get an error about TLS/SSL, add these two, optional components:

sample error:

```
$: ~/0.0.1/bin/python3 -m pip install --user "jupyter>=1.0.0" --extra-index-url https://prose-python-packages.azurewebsites.net
WARNING: pip is configured with locations that require TLS/SSL, however the ssl module in Python is not available.
Looking in indexes: https://pypi.org/simple, https://prose-python-packages.azurewebsites.net
Requirement already satisfied: jupyter
```

install these:

- Homebrew (optional). Install homebrew, then run `brew update` from the command line.
- *openssl* (optional). Next run `brew install openssl`.

Install the extension

To install the Machine Learning extension in Azure Data Studio, follow the steps below.

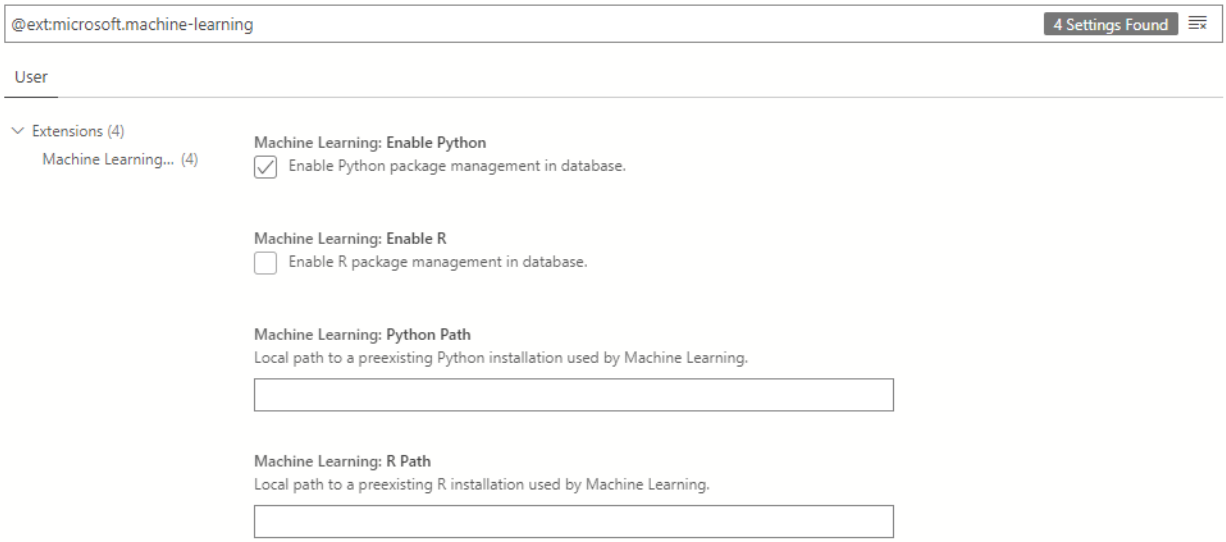
1. Open the extensions manager in Azure Data Studio. You can either select the extensions icon or select Extensions in the View menu.
2. Select the **Machine Learning** extension and view its details.
3. Select **Install**.
4. Select **Reload** to enable the extension. This is only required the first time you install an extension).

Extension settings

To change the settings for the Machine Learning extension, follow the steps below.

1. Open the extension manager in Azure Data Studio. You can either select the extensions icon or select Extensions in the View menu.
2. Find the **Machine Learning** extension under **enabled** extensions.
3. Select on the **Manage** icon.
4. Select on the **Extension Settings** icon.

The extensions settings look like this:



Enable Python

To use the Machine Learning extension as well as the Python package management in your database, follow the steps below.

> **IMPORTANT**
> The Machine Learning extension requires Python to be enabled and configured to most functionality to work, even if you do not wish to use the Python package management in database functionality.

1. Ensure that **Machine Learning: Enable Python** is enabled. This setting is enabled by default.
2. Provide the path to your pre-existing Python installation under **Machine Learning: Python Path**. This can either be the full path to the Python executable or the folder the executable is in. If you have used a Python kernel notebook in Azure Data Studio, the extension will use the path from the notebook by default.

Enable R

To use the Machine Learning extension for R package management in your database, follow the steps below.

1. Ensure that **Machine Learning: Enable R** is enabled. This setting is disabled by default.
2. Provide the path to your pre-existing R installation under **Machine Learning: R Path**. This has to be the full path to the R executable.

Use the Machine Learning extension

To use the Machine Learning extension in Azure Data Studio, follow the steps below.

1. Open the **Connections** viewlet in Azure Data Studio.
2. Right Select on your server and select **Manage**.
3. Select **Machine Learning** in the left side menu under **General**.

Follow the links under **Next steps** to see how you can use the Machine Learning extension for manage packages, make predictions, and import models in your database.

Next steps

- Manage packages in database
- Make predictions
- Import or view models
- Notebooks in Azure Data Studio
- SQL machine learning documentation
- Machine learning and AI with ONNX in SQL Edge (preview)

Manage packages in database with Machine Learning extension for Azure Data Studio (Preview)

6/29/2021 • 2 minutes to read • Edit Online

Learn how to manage Python or R packages in your database with the Machine Learning extension for Azure Data Studio.

> **IMPORTANT**
>
> Manage packages in database with the Machine Learning extension currently only supports Machine Learning Services on SQL Server 2019.

Prerequisites

- Install and configure Machine Learning extension for Azure Data Studio. You need to specify the Python or R installation paths in the Extension Settings for the package management to work.
- The **sqlmlutils** package. If the package is not already installed, the Machine Learning extension will prompt you to install this.
- For SQL Server on Linux, Windows authentication is not supported. Therefore, you need to use SQL authentication to connect to your SQL Server on Linux.

Manage Python packages

You can install and uninstall Python packages with the Machine Learning extension.

Install new Python package

Follow the steps below to install Python packages in your database.

1. Select **Manage packages in database**.
2. If you're asked to install **sqlmlutils**, select **Yes**.
3. Under the **Installed** tab, select **Python** under the **Package Type** and select your database under **Location**.
4. Select the **Add new** tab.
5. Enter the Python package you like to install in **Search Python packages** and select **Search**.
6. Verify the package is listed under **Package Name** and the desired version is listed under **Package Version**.
7. Select **Install**.
8. Select **Close** and verify that the package was installed successfully under **Tasks**.

Uninstall a Python package

Follow the steps below to uninstall Python packages in your database.

> **NOTE**
> You can only uninstall packages that have been installed in your database. It is not possible to uninstall a package that has been pre-installed with SQL Server Machine Learning Services.

1. Select **Manage packages in database**.
2. If you're asked to install **sqlmlutils**, select **Yes**.
3. Under the **Installed** tab, select **Python** under the **Package Type** and select your database under **Location**.
4. Select the package(s) you would like to uninstall.
5. Select **Uninstall selected packages**.
6. Select **Close** and verify that the package was installed successfully under **Tasks**.

Manage R packages

You can install and uninstall R packages with the Machine Learning extension.

Install new R package

Follow the steps below to install Python packages in your database.

1. Select **Manage packages in database**.
2. If asked to install **sqlmlutils**, select **Yes**.
3. Under the **Installed** tab, select **R** under the **Package Type** and select your database under **Location**.
4. Select the **Add new** tab.
5. Enter the R package you like to install in **Search R packages** and select **Search**.
6. Verify the package is listed under **Package Name** and the desired version is listed under **Package Version**.
7. Select **Install**.
8. Select **Close** and verify that the package was installed successfully under **Tasks**.

Uninstall an R package

Follow the steps below to uninstall R packages in your database.

> **NOTE**
> You can only uninstall packages that have been installed in your database. It is not possible to uninstall a package that has been pre-installed with SQL Server Machine Learning Services.

1. Select **Manage packages in database**.
2. If asked to install **sqlmlutils**, select **Yes**.
3. Under the **Installed** tab, select **R** under the **Package Type** and select your database under **Location**.
4. Select the package(s) you would like to uninstall.
5. Select **Uninstall selected packages**.

6. Select **Close** and verify that the package was installed successfully under **Tasks**.

Next steps

- Machine Learning extension in Azure Data Studio
- Make predictions
- Import or view models
- Notebooks in Azure Data Studio
- SQL machine learning documentation

Make predictions with Machine Learning extension for Azure Data Studio (Preview)

6/29/2021 • 2 minutes to read • Edit Online

Learn how to use the Machine Learning extension for Azure Data Studio to make predictions with an ONNX model in your database. The extension will generate a T-SQL script using PREDICT to make predictions on the dataset stored in your table with a model that is previously imported, resides in a local file, or from Azure Machine Learning.

> **IMPORTANT**
>
> Make predictions with the Machine Learning extension currently only supports Machine Learning Services in Azure SQL Managed Instance and Azure SQL Edge with ONNX.

Prerequisites

- Install and configure Machine Learning extension for Azure Data Studio. You need to specify the Python installation paths in the Extension Settings.
- The **onnxruntime**, **mlflow**, and **mlflow-dbstore** Python packages. If the packages are not already installed, the Machine Learning extension will prompt you to install them.

Make predictions from ONNX model

Follow the steps below to use an ONNX model to make predictions.

1. Select on **Make predictions**.
2. If you're asked to install **onnxruntime**, **mlflow**, and **mlflow-dbstore**, select **Yes**.
3. Choose where your model is located and select **Next**. You can use:
   - **Imported models**. Choose this to use a model that is already stored in your database. Choose the **Model database** and **Model table** where your model is located, select the model you want to use, and select **Next**.
   - **File upload**. Choose this to use a model from a file. Select the model file under **Source files** and select **Next**.
   - **Azure Machine Learning**. Choose this to use a model from Azure Machine Learning. First, **Sign in to Azure**. Then select your **Azure account**, **Azure subscription**, **Azure resource group**, and **Azure ML workspace**. Select the model you want to use and select **Next**.
4. Map the source data to your model.
   - Select the **Source database** and **Source table** containing the data set for which you want to apply the prediction.
   - Map the columns under **Model Input mapping** and **Model output**. The extension will automatically map columns that have the same name and data type.
5. Select **Predict**.

Azure Data Studio will create a new T-SQL query with the PREDICT, which you can use to make predictions on your data.

Next steps

- Machine Learning extension in Azure Data Studio
- Manage packages in database
- Import or view models
- Notebooks in Azure Data Studio
- SQL machine learning documentation
- Machine Learning Services in Azure SQL Managed Instance
- Machine learning and AI with ONNX in SQL Edge (Preview)

Import or view models with Machine Learning extension for Azure Data Studio (Preview)

6/29/2021 • 2 minutes to read • Edit Online

Learn how to use the Machine Learning extension for Azure Data Studio to import an ONNX model or view already imported models in your database.

> **IMPORTANT**
>
> Import and view in a database with the Machine Learning extension currently only supports Machine Learning Services in Azure SQL Managed Instance and Azure SQL Edge with ONNX.

Prerequisites

- Install and configure Machine Learning extension for Azure Data Studio. You need to specify the Python installation paths in the Extension Settings.
- The **onnxruntime**, **mlflow**, and **mlflow-dbstore** Python packages. If the packages are not already installed, the Machine Learning extension will prompt you to install them.

View models

Follow the steps below to view ONNX models that are stored in your database.

1. Select **Import or view models**.
2. If you're asked to install **onnxruntime**, **mlflow**, and **mlflow-dbstore**, select **Yes**.
3. Select the **Models database** and **Models table** where your models are stored in.

This will show a list of your models. You can edit the model name and description, or delete a model from the list.

Import a new model

Follow the steps below to import an ONNX model in your database.

1. Select **Import or view models**.
2. If you're asked to install **onnxruntime**, **mlflow**, and **mlflow-dbstore**, select **Yes**.
3. Select **Import models**.
4. Select the **Models database** you want to store the imported model in.
5. Select the **Models table** you want to store the imported model in. You can either choose an **Existing table** or create a **New table**. Select **Next**.
6. Select where your model is located and Select **Next**. You can use:
   - **File upload**. Choose this to use a model from a file. Select the model file under **Source files** and Select **Next**.
   - **Azure Machine Learning**. Choose this to use a model from Azure Machine Learning. First, **Sign in**

to Azure. Then select your **Azure account**, **Azure subscription**, **Azure resource group**, and **Azure ML workspace**. Select the model you want to use and Select **Next**.

7. Enter the model **Name** and **Description** and Select **Deploy** to store the model in your database.

> **NOTE**
>
> The Machine Learning extension is currently in preview. Therefore, the table schema where the models are stored might change in the future.

Next steps

- Machine Learning extension in Azure Data Studio
- Manage packages in database
- Make predictions
- SQL machine learning documentation
- Machine Learning Services in Azure SQL Managed Instance
- Machine learning and AI with ONNX in SQL Edge (Preview)

Azure SQL Managed Instance dashboard for Azure Data Studio (Preview)

3/5/2021 • 3 minutes to read • Edit Online

The Azure SQL Managed Instance extension provides a dashboard for working with an Azure SQL Managed Instance in Azure Data Studio. This extension provides the following features:

- Shows SQL Managed Instance properties, including vCores and used storage
- Monitors CPU and storage usage for previous two hours
- Displays configuration warnings and tuning recommendations
- Shows state of database replicas
- Displays filtered error logs

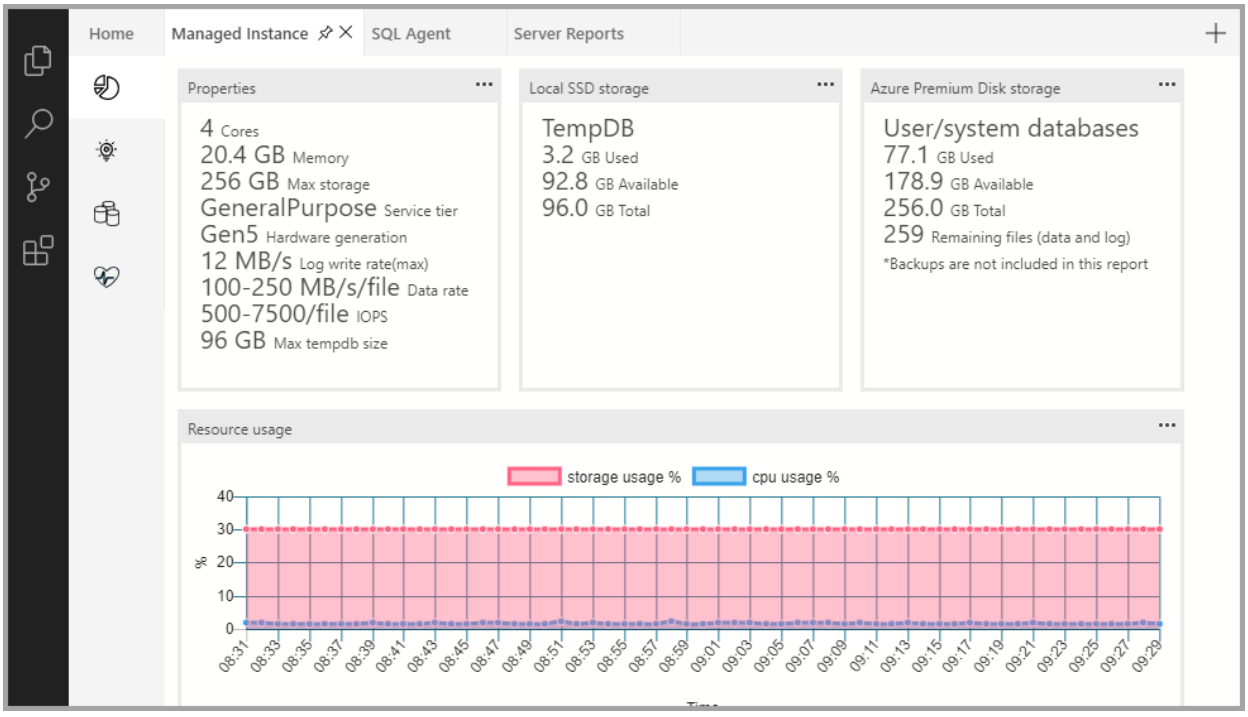
Install

You can install the official release of this extension. Follow the steps in the Azure Data Studio documentation. In the **Extensions** pane, search for "Managed Instance", and install it there. After it's installed, you'll get notified automatically about any future extension updates.

With the extension installed, you'll see a **Managed Instance** tab in Azure Data Studio. Here you can find information specific for your managed instance.

Properties

The extension displays technical characteristics and some resource usage of your managed instance.



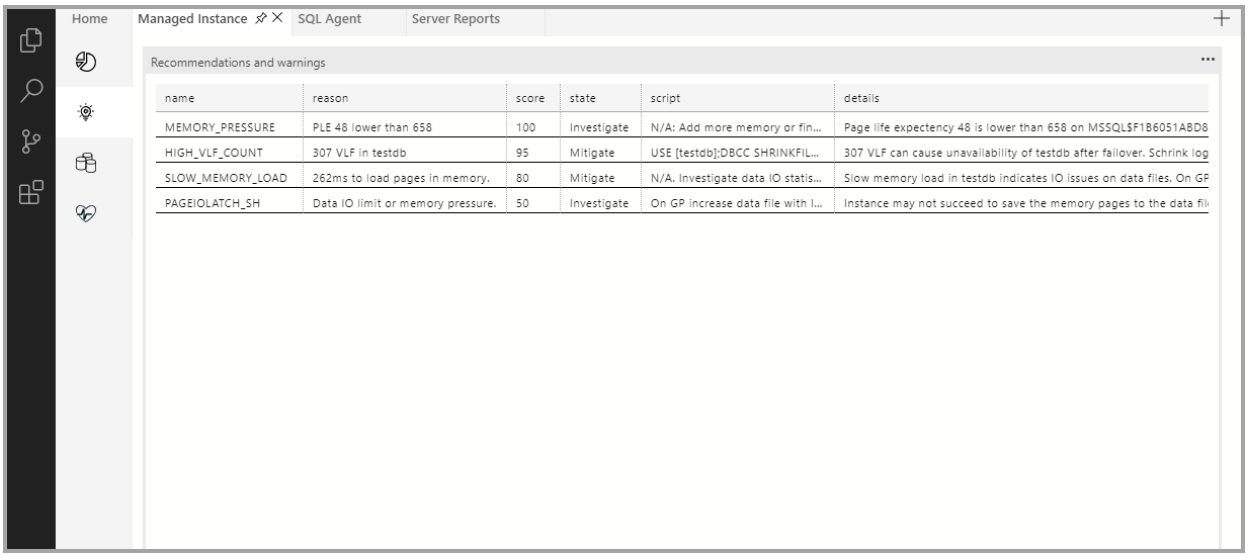
The top pane shows the following details:

- **Properties**. Get basic information about your managed instance, including available vCores, memory, and storage. Also find your current service tier, hardware generation, and IO characteristics such as instance log write throughput or file I/O throughput characteristics.

- **Local SSD storage**. On the general-purpose service tier, **TempDB** files are stored locally. On the business-critical service tier, *all* database files are placed on local SSD storage. In this section, you can see how much space on the local storage is used by your managed instance.
- **Azure Premium Disk Storage**. If you have the general-purpose service tier, both user and system database files are placed on Azure Premium storage. In this section, you can see the amount of data used, the number of files, and the available storage. On the business-critical service tier, this section is empty.
- **Resource usage**. View the percentage of storage and CPU that your managed instance used over the previous two hours. This way, you can increase the instance size if it's nearing the limit.

Recommendations

When you select the second pane in the **Managed Instance** tab, you get recommendations and alerts to help optimize your managed instance.



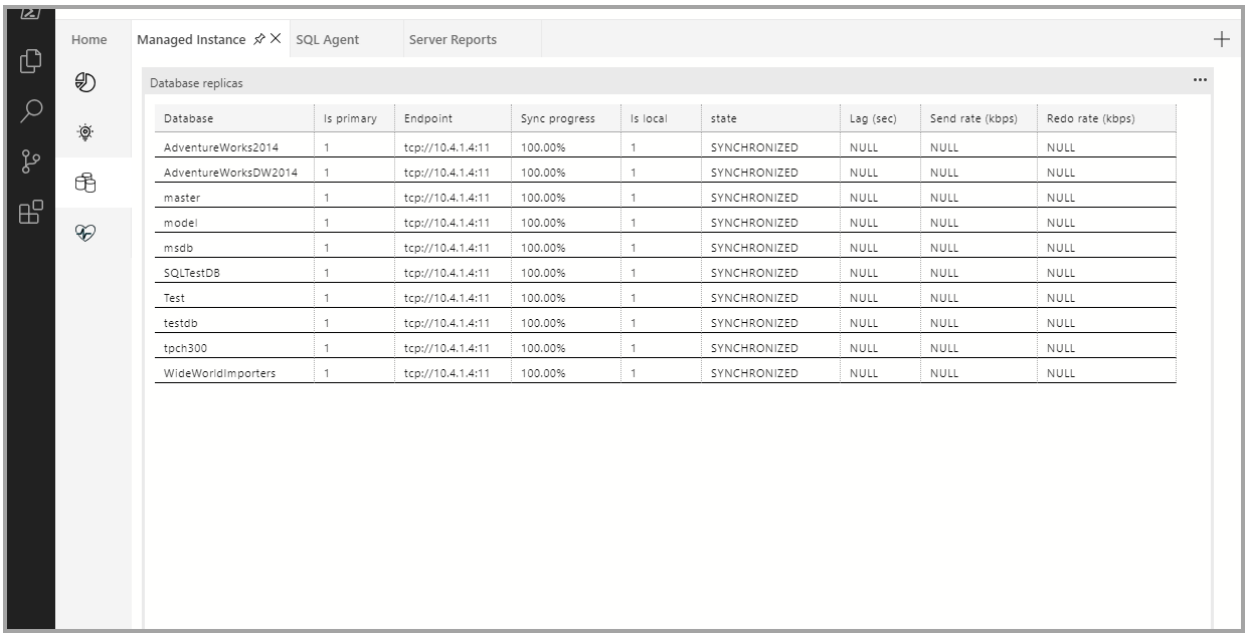
You might see some of the following recommendations:

- **Reaching storage space limit**. Either delete unnecessary data or increase instance storage size. Databases that reach storage limit might fail to process even read queries.
- **Reaching instance throughput limit**. Notifies you when you're loading near the limit of your service tier: 22 MB/s for general-purpose or 48 MB/s for business-critical. Be aware that your managed instance will limit your load to ensure that backups can be taken.
- **Memory pressure**. Low page life expectancy or numerous `PAGEIOLATCH` wait statistics might indicate that your instance is evicting pages from the memory and constantly trying to load more pages from disk.
- **Log file limits**. If your log files approach the file I/O limits on the general-purpose service tier, you might need to increase the log file size to get better performance.
- **Data file limits**. If your data files approach the file I/O limits on the general-purpose service tier, you might need to increase file size to get better performance. This issue might cause memory pressure and slow down backups.
- **Availability issues**. A high number of virtual log files can affect performance. If there's a process failure, such issues might result in longer database recovery on the general-purpose service tier.

Periodically review these recommendations, investigate the root causes, and take action to correct any issues. The SQL Managed Instance extension provides scripts you can run to mitigate some of the reported issues.

Replicas

The third pane in the **Managed Instance** tab shows you the state of database replicas in your managed instance.

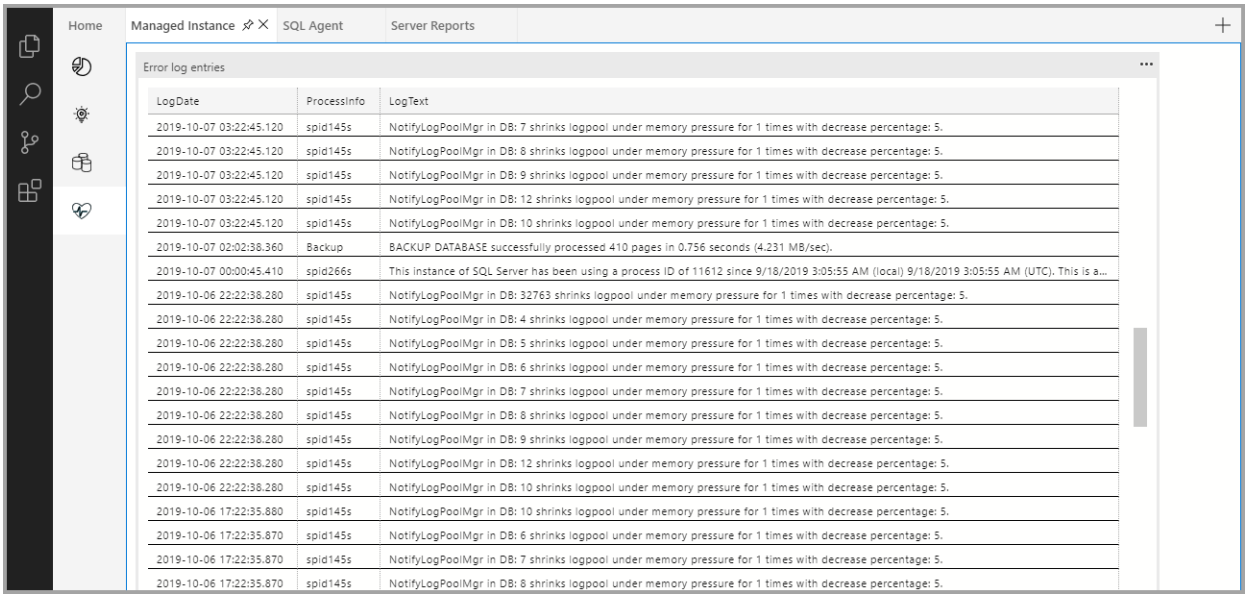


Database	Is primary	Endpoint	Sync progress	Is local	state	Lag (sec)	Send rate (kbps)	Redo rate (kbps)
AdventureWorks2014	1	tcp://10.4.1.4:11	100.00%	1	SYNCHRONIZED	NULL	NULL	NULL
AdventureWorksDW2014	1	tcp://10.4.1.4:11	100.00%	1	SYNCHRONIZED	NULL	NULL	NULL
master	1	tcp://10.4.1.4:11	100.00%	1	SYNCHRONIZED	NULL	NULL	NULL
model	1	tcp://10.4.1.4:11	100.00%	1	SYNCHRONIZED	NULL	NULL	NULL
msdb	1	tcp://10.4.1.4:11	100.00%	1	SYNCHRONIZED	NULL	NULL	NULL
SQLTestDB	1	tcp://10.4.1.4:11	100.00%	1	SYNCHRONIZED	NULL	NULL	NULL
Test	1	tcp://10.4.1.4:11	100.00%	1	SYNCHRONIZED	NULL	NULL	NULL
testdb	1	tcp://10.4.1.4:11	100.00%	1	SYNCHRONIZED	NULL	NULL	NULL
tpch300	1	tcp://10.4.1.4:11	100.00%	1	SYNCHRONIZED	NULL	NULL	NULL
WideWorldImporters	1	tcp://10.4.1.4:11	100.00%	1	SYNCHRONIZED	NULL	NULL	NULL

On the general-purpose service tier, every database has a single (primary) replica. On a business-critical tier instance, every database has one primary and three secondary replicas, one of which is used for read-only workloads. On the **Replicas** pane, you can monitor the synchronization process and verify that all secondary replicas are synchronized with the primary replica.

Logs

The fourth pane of **Managed Instance** shows the most recent and relevant SQL error log entries.



LogDate	ProcessInfo	LogText
2019-10-07 03:22:45.120	spid145s	NotifyLogPoolMgr in DB: 7 shrinks logpool under memory pressure for 1 times with decrease percentage: 5.
2019-10-07 03:22:45.120	spid145s	NotifyLogPoolMgr in DB: 8 shrinks logpool under memory pressure for 1 times with decrease percentage: 5.
2019-10-07 03:22:45.120	spid145s	NotifyLogPoolMgr in DB: 9 shrinks logpool under memory pressure for 1 times with decrease percentage: 5.
2019-10-07 03:22:45.120	spid145s	NotifyLogPoolMgr in DB: 12 shrinks logpool under memory pressure for 1 times with decrease percentage: 5.
2019-10-07 03:22:45.120	spid145s	NotifyLogPoolMgr in DB: 10 shrinks logpool under memory pressure for 1 times with decrease percentage: 5.
2019-10-07 02:02:38.360	Backup	BACKUP DATABASE successfully processed 410 pages in 0.756 seconds (4.231 MB/sec).
2019-10-07 00:00:45.410	spid266s	This instance of SQL Server has been using a process ID of 11612 since 9/18/2019 3:05:55 AM (local) 9/18/2019 3:05:55 AM (UTC). This is a...
2019-10-06 22:22:38.280	spid145s	NotifyLogPoolMgr in DB: 32763 shrinks logpool under memory pressure for 1 times with decrease percentage: 5.
2019-10-06 22:22:38.280	spid145s	NotifyLogPoolMgr in DB: 4 shrinks logpool under memory pressure for 1 times with decrease percentage: 5.
2019-10-06 22:22:38.280	spid145s	NotifyLogPoolMgr in DB: 5 shrinks logpool under memory pressure for 1 times with decrease percentage: 5.
2019-10-06 22:22:38.280	spid145s	NotifyLogPoolMgr in DB: 6 shrinks logpool under memory pressure for 1 times with decrease percentage: 5.
2019-10-06 22:22:38.280	spid145s	NotifyLogPoolMgr in DB: 7 shrinks logpool under memory pressure for 1 times with decrease percentage: 5.
2019-10-06 22:22:38.280	spid145s	NotifyLogPoolMgr in DB: 8 shrinks logpool under memory pressure for 1 times with decrease percentage: 5.
2019-10-06 22:22:38.280	spid145s	NotifyLogPoolMgr in DB: 9 shrinks logpool under memory pressure for 1 times with decrease percentage: 5.
2019-10-06 22:22:38.280	spid145s	NotifyLogPoolMgr in DB: 12 shrinks logpool under memory pressure for 1 times with decrease percentage: 5.
2019-10-06 22:22:38.280	spid145s	NotifyLogPoolMgr in DB: 10 shrinks logpool under memory pressure for 1 times with decrease percentage: 5.
2019-10-06 17:22:35.880	spid145s	NotifyLogPoolMgr in DB: 10 shrinks logpool under memory pressure for 1 times with decrease percentage: 5.
2019-10-06 17:22:35.870	spid145s	NotifyLogPoolMgr in DB: 6 shrinks logpool under memory pressure for 1 times with decrease percentage: 5.
2019-10-06 17:22:35.870	spid145s	NotifyLogPoolMgr in DB: 7 shrinks logpool under memory pressure for 1 times with decrease percentage: 5.
2019-10-06 17:22:35.870	spid145s	NotifyLogPoolMgr in DB: 8 shrinks logpool under memory pressure for 1 times with decrease percentage: 5.

Although your managed instance generates a large number of log entries, most of them are internal/system information. Also, some log entries show physical database names (`GUID` values) instead of actual logical database names.

The SQL Managed Instance extension filters out unnecessary log entries based on the Dimitri Furman method. The extension also displays actual logical file names instead of physical names.

Reporting problems

If you experience any problems with the SQL Managed Instance extension, go to the Extension GitHub project and report your issue.

Code of conduct

This project has adopted the Microsoft Open Source Code of Conduct.

For more information, see the Code of Conduct FAQ or contact opencode@microsoft.com with any additional questions or comments.

Next steps

For more information, visit the GitHub project.

PostgreSQL extension (Preview)

11/2/2020 • 2 minutes to read • Edit Online

The PostgreSQL extension (preview) enables you to connect to, query, and develop for Postgres using the capabilities of Azure Data Studio.

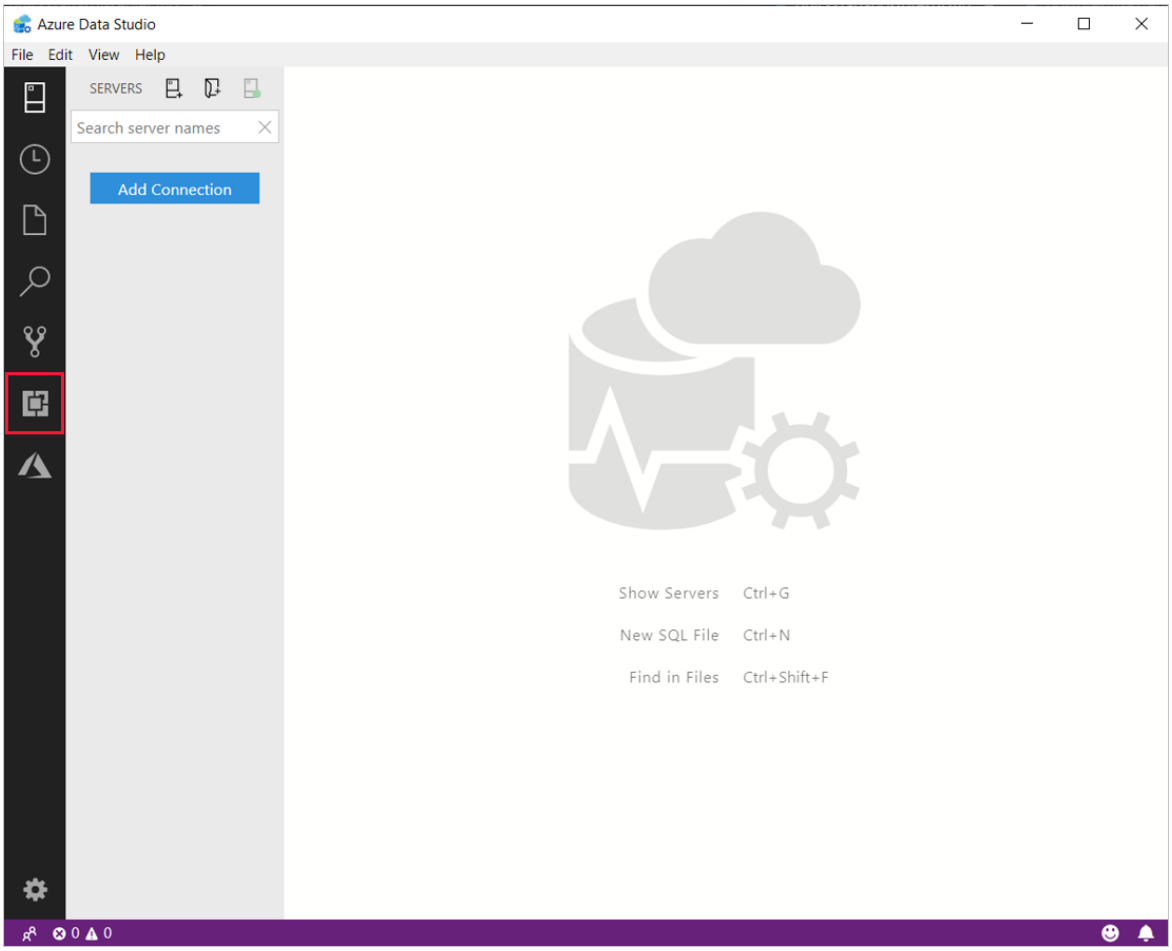
The Azure Data Studio functionality available for PostgreSQL includes:

- Connection manager & query editor
- Customizable dashboards & insight widgets
- Code snippets
- Integrated terminal
- Keyboard shortcuts
- Source control integration
- Workspace & user settings

Install the PostgreSQL extension (preview)

If you don't already have Azure Data Studio installed, see its install instructions.

1. Select the extensions icon from the sidebar in Azure Data Studio.



2. Type 'postgresql' into the search bar. Select the PostgreSQL extension.

3. Select **Install**. Once installed, select **Reload** to activate the extension in Azure Data Studio.

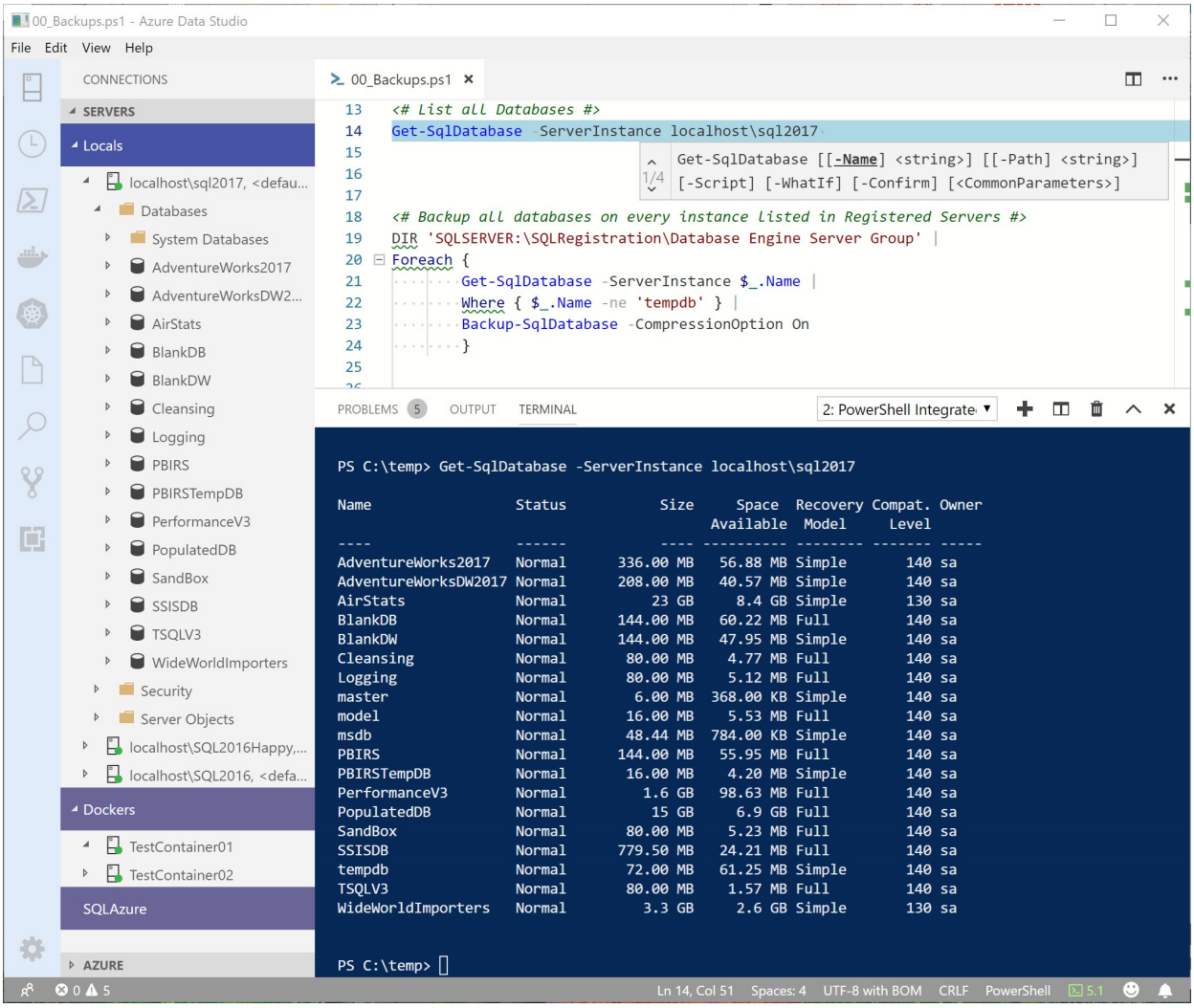
Next steps

Learn how to connect and query Postgres from Azure Data Studio.

PowerShell Editor Support for Azure Data Studio

11/2/2020 • 6 minutes to read • Edit Online

This extension provides rich PowerShell editor support in Azure Data Studio. Now you can write and debug PowerShell scripts using the excellent IDE-like interface that Azure Data Studio provides.



Features

- Syntax highlighting
- Code snippets
- IntelliSense for cmdlets and more
- Rule-based analysis provided by PowerShell Script Analyzer
- Go to Definition of cmdlets and variables
- Find References of cmdlets and variables
- Document and workspace symbol discovery
- Run selected selection of PowerShell code using `F8`
- Launch online help for the symbol under the cursor using `Ctrl+F1`
- Basic interactive console support!

Installing the Extension

You can install the official release of the PowerShell extension by following the steps in the Azure Data Studio documentation. In the Extensions pane, search for "PowerShell" extension and install it there. You will get notified automatically about any future extension updates!

You can also install a VSIX package from our Releases page and install it through the command line:

```
azuredatastudio --install-extension PowerShell-<version>.vsix
```

Platform support

- **Windows 7 through 10** with Windows PowerShell v3 and higher, and PowerShell Core
- **Linux** with PowerShell Core (all PowerShell-supported distributions)
- **macOS** with PowerShell Core

Read the FAQ for answers to common questions.

Installing PowerShell Core

If you are running Azure Data Studio on macOS or Linux, you may also need to install PowerShell Core.

PowerShell Core is an Open Source project on GitHub. For more information on installing PowerShell Core on macOS or Linux platforms, see the following articles:

- Installing PowerShell Core on Linux
- Installing PowerShell Core on macOS

Example Scripts

There are some example scripts in the extension's `examples` folder that you can use to discover PowerShell editing and debugging functionality. Check out the included README.md file to learn more about how to use them.

This folder can be found at the following path:

```
$HOME/.azuredatastudio/extensions/ms-vscode.PowerShell-<version>/examples
```

or if you're using the preview version of the extension

```
$HOME/.azuredatastudio/extensions/ms-vscode.powershell-preview-<version>/examples
```

To open/view the extension's examples in Azure Data Studio, run the following code from your PowerShell command prompt:

```
azuredatastudio (Get-ChildItem $Home\.azuredatastudio\extensions\ms-vscode.PowerShell-*\examples)[-1]
```

Creating and opening files

To create and open a new file inside the editor, use the New-EditorFile from within the PowerShell Integrated Terminal.

```
PS C:\temp> New-EditorFile ExportData.ps1
```

This command works for any file type, not just PowerShell files.

```
PS C:\temp> New-EditorFile ImportData.py
```

To open one or more files in Azure Data Studio, use the `Open-EditorFile` command.

```
Open-EditorFile ExportData.ps1, ImportData.py
```

No focus on console when executing

For those users who are used to working with SSMS, you're used to being able to execute a query, and then being able to re-execute it again without having to switch back to the query pane. In this case, the default behavior of the code editor may feel strange to you. To keep the focus in the editor when you execute with F8 change the following setting:

```
"powershell.integratedConsole.focusConsoleOnExecute": false
```

The default is `true` for accessibility purposes.

Be aware this setting will prevent the focus from changing to the console, even when you use a command that explicitly calls for input, like `Get-Credential`.

SQL PowerShell Examples

In order to use these examples (below), you need to install the SqlServer module from the PowerShell Gallery.

```
Install-Module -Name SqlServer
```

> **NOTE**
> With version `21.1.18102` and up, the `SqlServer` module supports PowerShell Core 6.2 and up, in addition to Windows PowerShell.

In this example, we use the `Get-SqlInstance` cmdlet to Get the Server SMO objects for ServerA & ServerB. The default output for this command will include the Instance name, version, Service Pack, & CU Update Level of the instances.

```
Get-SqlInstance -ServerInstance ServerA, ServerB
```

Here is a sample of what that output will look like:

```
Instance Name             Version    ProductLevel UpdateLevel  HostPlatform HostDistribution
-------------             -------    ------------ -----------  ------------ ----------------
ServerA                   13.0.5233  SP2          CU4          Windows      Windows Server 2016 Datacenter
ServerB                   14.0.3045  RTM          CU12         Linux        Ubuntu
```

The `SqlServer` module contains a Provider called `SQLRegistration` which allows you to programatically access the following types of saved SQL Server connections:

- Database Engine Server (Registered Servers)
- Central Management Server (CMS)

- Analysis Services
- Integration Services
- Reporting Services

In the following example, we will do a `dir` (alias for `Get-ChildItem`) to get the list of all SQL Server instances listed in your Registered Servers file.

```
dir 'SQLSERVER:\SQLRegistration\Database Engine Server Group' -Recurse
```

Here is a sample of what that output could look like:

```
Mode Name
---- ----
-    ServerA
-    ServerB
-    localhost\SQL2017
-    localhost\SQL2016Happy
-    localhost\SQL2017
```

For many operations that involve a database, or objects within a database, the `Get-SqlDatabase` cmdlet can be used. If you supply values for the both the `-ServerInstance` and `-Database` parameters, only that one database object will be retrieved. However, if you specify only the `-ServerInstance` parameter, a full list of all databases on that instance will be returned.

Here is a sample of what that output will look like:

```
Name                 Status           Size     Space  Recovery Compat. Owner
                                           Available  Model     Level
----                 ------           ---- ---------- -------- ------- -----
AdventureWorks2017   Normal      336.00 MB   57.01 MB Simple       140 sa
master               Normal        6.00 MB  368.00 KB Simple       140 sa
model                Normal       16.00 MB    5.53 MB Full         140 sa
msdb                 Normal       48.44 MB    1.70 MB Simple       140 sa
PBIRS                Normal      144.00 MB   55.95 MB Full         140 sa
PBIRSTempDB          Normal       16.00 MB    4.20 MB Simple       140 sa
SSISDB               Normal      325.06 MB   26.21 MB Full         140 sa
tempdb               Normal       72.00 MB   61.25 MB Simple       140 sa
WideWorldImporters   Normal         3.2 GB     2.6 GB Simple       130 sa
```

This next example uses the `Get-SqlDatabase` cmdlet to retrieve a list of all databases on the ServerB instance, then presents a grid/table (using the `Out-GridView` cmdlet) to select which databases should be backed up. Once the user clicks on the "OK" button, only the highlighted databases will be backed up.

```
Get-SqlDatabase -ServerInstance ServerB |
Out-GridView -PassThru |
Backup-SqlDatabase -CompressionOption On
```

This example, again, gets list of all SQL Server instances listed in your Registered Servers file, then calls the `Get-SqlAgentJobHistory` which reports every failed SQL Agent Job since Midnight, for each SQL Server instance listed.

```
dir 'SQLSERVER:\SQLRegistration\Database Engine Server Group' -Recurse |
WHERE {$_.Mode -ne 'd' } |
FOREACH {
    Get-SqlAgentJobHistory -ServerInstance  $_.Name -Since Midnight -OutcomesType Failed
}
```

In this example, we will do a `dir` (alias for `Get-ChildItem`) to get the list of all SQL Server instances listed in your Registered Servers file, and then use the `Get-SqlDatabase` cmdlet to get a list of Databases for each of those instances.

```
dir 'SQLSERVER:\SQLRegistration\Database Engine Server Group' -Recurse |
WHERE { $_.Mode -ne 'd' } |
FOREACH {
    Get-SqlDatabase -ServerInstance $_.Name
}
```

Here is a sample of what that output will look like:

```
Name                 Status           Size  Space     Recovery Compat. Owner
                                           Available Model    Level
----                 ------           ---- ---------- -------- ------- -----
AdventureWorks2017   Normal      336.00 MB   57.01 MB Simple       140 sa
master               Normal        6.00 MB  368.00 KB Simple       140 sa
model                Normal       16.00 MB    5.53 MB Full         140 sa
msdb                 Normal       48.44 MB    1.70 MB Simple       140 sa
PBIRS                Normal      144.00 MB   55.95 MB Full         140 sa
PBIRSTempDB          Normal       16.00 MB    4.20 MB Simple       140 sa
SSISDB               Normal      325.06 MB   26.21 MB Full         140 sa
tempdb               Normal       72.00 MB   61.25 MB Simple       140 sa
WideWorldImporters   Normal         3.2 GB     2.6 GB Simple       130 sa
```

Reporting Problems

If you experience any problems with the PowerShell Extension, see the troubleshooting docs for information on diagnosing and reporting issues.

Security Note

For any security issues, see here.

Contributing to the Code

Check out the development documentation for more details on how to contribute to this extension!

Maintainers

- Keith Hill - @r_keith_hill
- Tyler Leonhardt - @TylerLeonhardt
- Rob Holt

License

This extension is licensed under the MIT License. For details on the third-party binaries that we include with releases of this project, see the third-party notices file.

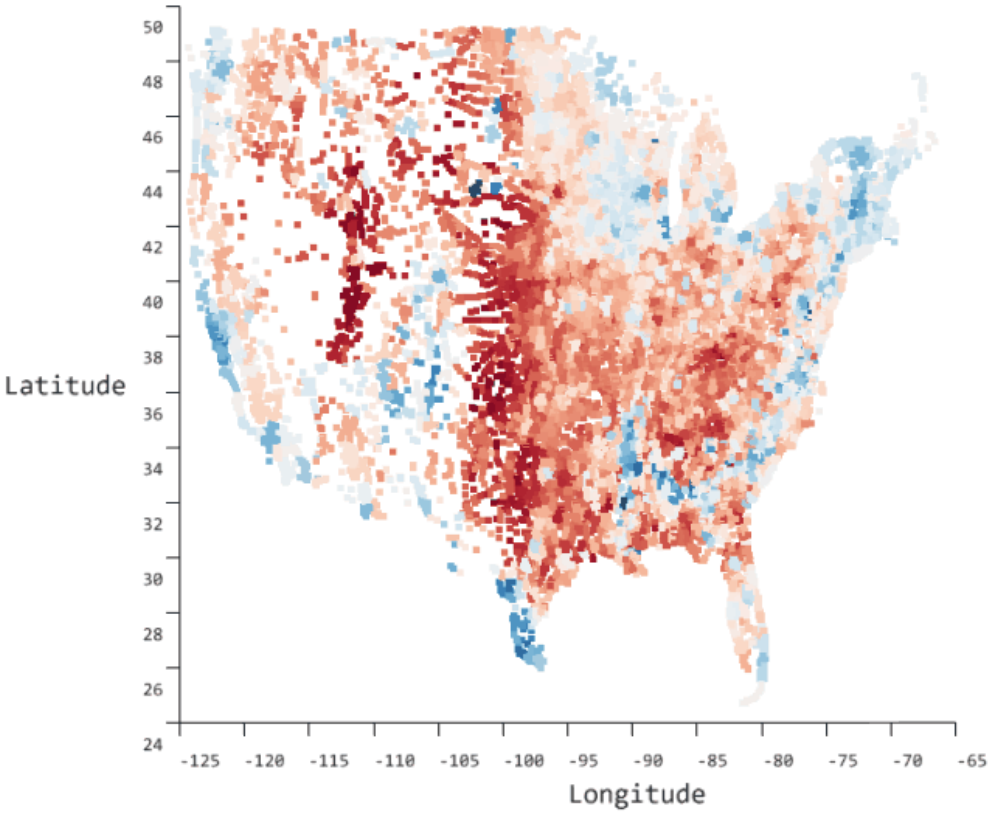
Code of Conduct

This project has adopted the Microsoft Open Source Code of Conduct. For more information, see the Code of Conduct FAQ or contact opencode@microsoft.com with any additional questions or comments.

SandDance for Azure Data Studio (Preview)

11/2/2020 • 2 minutes to read • Edit Online

Azure Data Studio now offers a way to create quick visualizations for your data. This extension is helpful when you are trying to look at the data and understand what's going on. We use a technology called SandDance from Microsoft Research, which can generate in-place visualizations of the data.



By using easy-to-understand views, SandDance helps you find insights about your data, which in turn help you tell stories supported by data, build cases based on evidence, test hypotheses, dig deeper into surface explanations, support decisions for purchases, or relate data into a wider, real world context.

SandDance uses unit visualizations, which apply a one-to-one mapping between rows in your database and marks on the screen. Smooth animated transitions between views help you to maintain context as you interact with your data.

Usage

View .csv or .tsv files

This includes local files or files on HDFS in your SQL Server 2019 Big Data Cluster.

Starting from the File menu, use Open Folder or [Ctrl+K Ctrl+O] to open the directory containing the .CSV file. Next, from within the Explorer panel, Right-click on the .csv or .tsv file and choose *View in SandDance*.

Right-click on a .csv or .tsv file in HDFS if you are connected to SQL Server 2019 Big Data Cluster and choose *View in SandDance*.

View SQL query results

Starting from a SQL query window, execute a query to get a results grid. Click the Visualizer icon on the right side of the Results pane.

Known Issues

Currently, we do not cap the row count that is visualized. However, memory consumption goes up proportionally to the number of rows, so we recommend that the data set or view is limited to around 100k rows.

See known issues

Release Notes

1.0.0

Initial release of azdata-sanddance

Next Steps

To learn more, visit the GitHub repo.

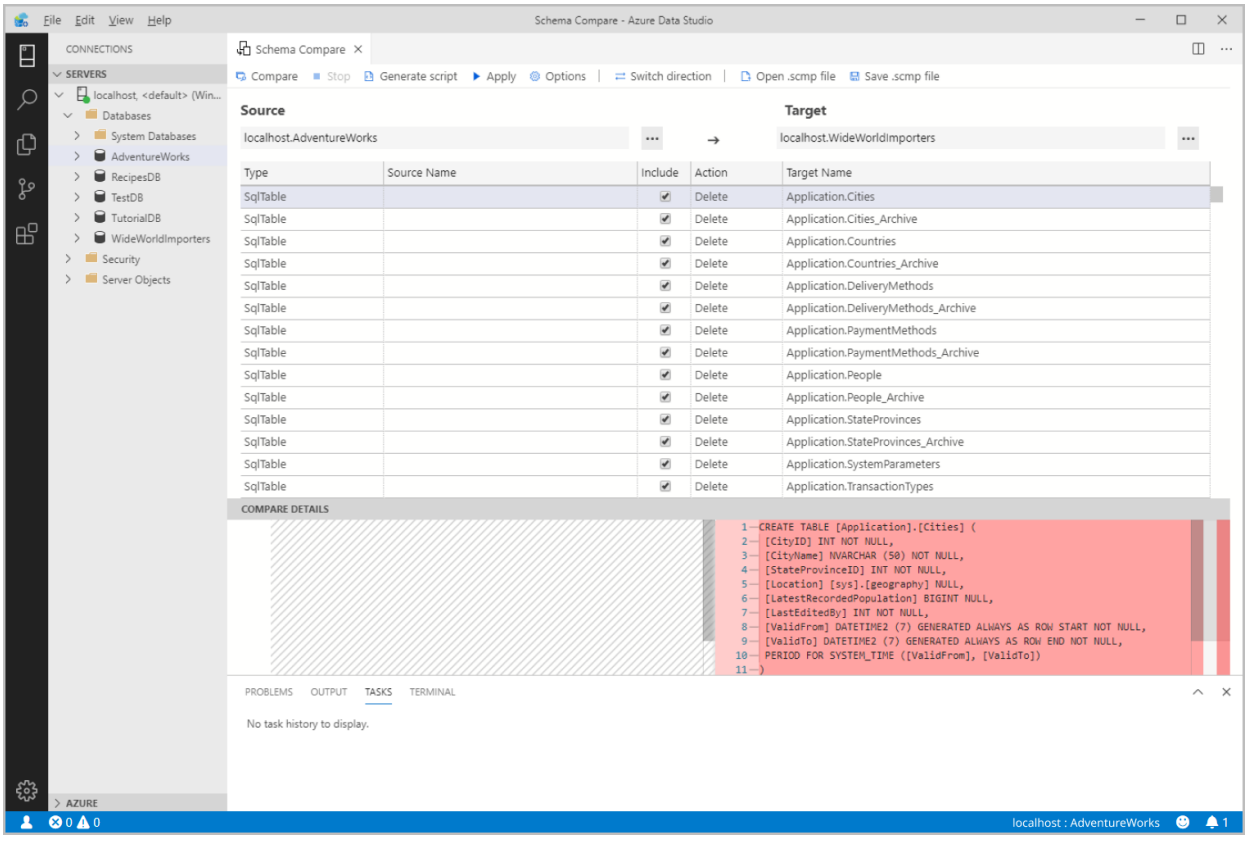
Schema Compare extension

11/2/2020 • 2 minutes to read • Edit Online

The Schema Compare extension provides an easy-to-use experience to compare two database definitions and apply the differences from the source to the target.

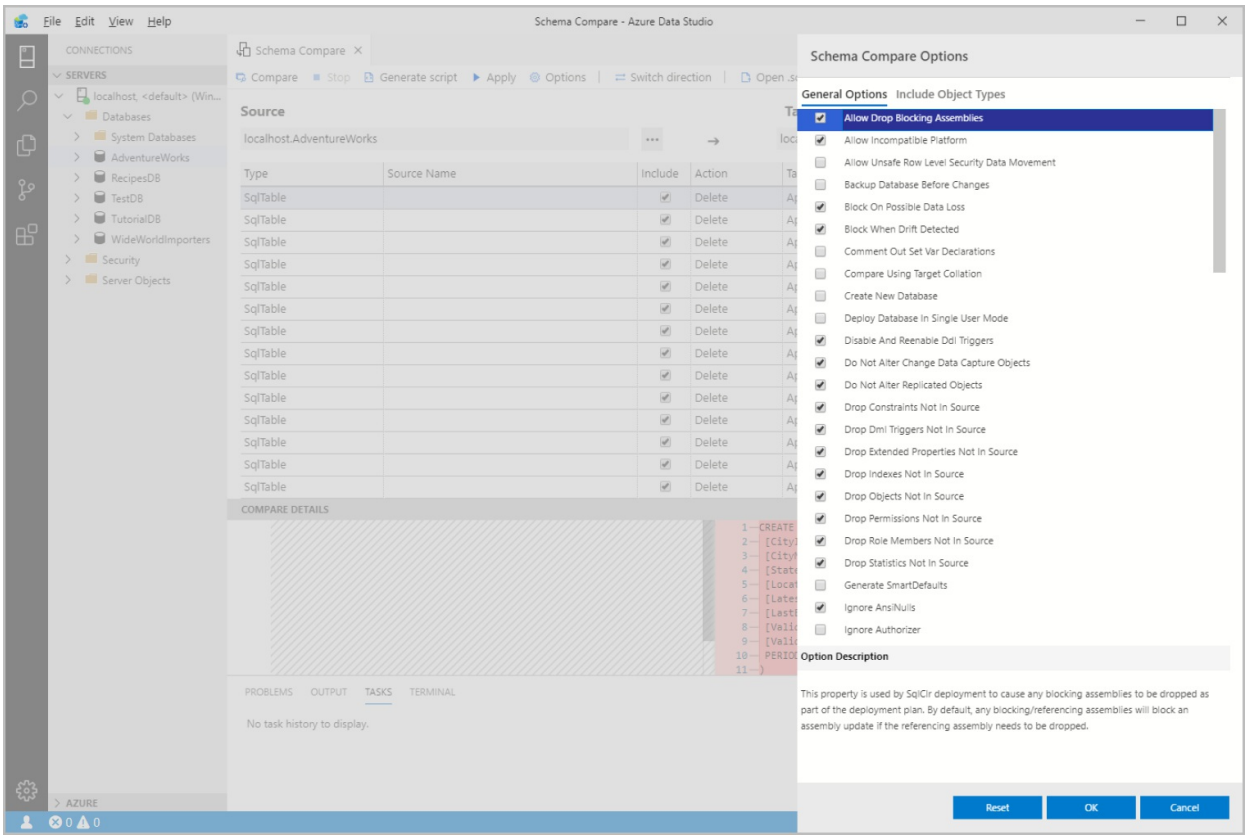
Features

- Compare schemas of two dacpac files or databases
- View results as a set of actions that must be taken against the target for it to match the source
- Selectively exclude actions listed in results
- Set options that control the scope of the comparison
- Apply changes to target or generate a script with the same effect
- Save the comparison



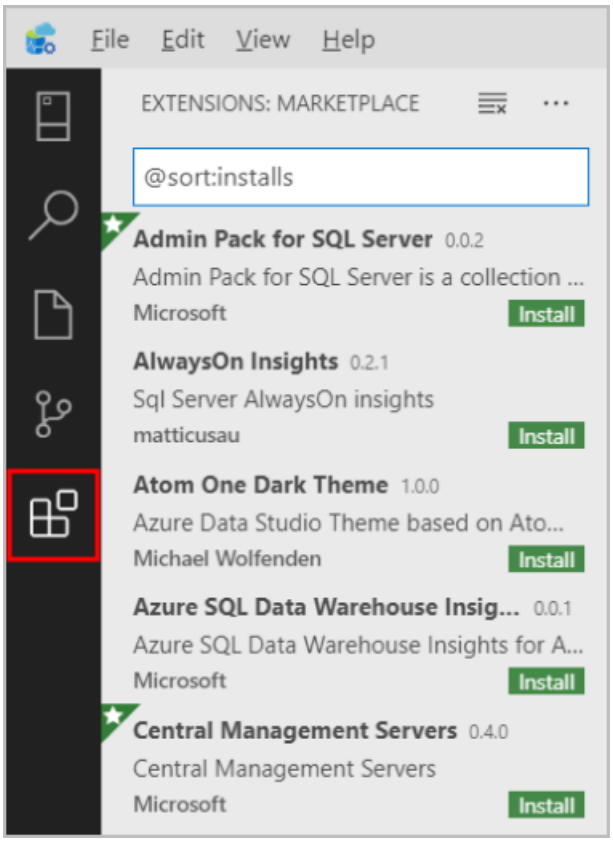
Why Would I Use the Schema Compare Extension?

It can be tedious to manually manage and synchronize different database versions. The Schema Compare extension simplifies the process of comparing databases and gives you full control when synchronizing them — you can selectively filter specific differences and categories of differences before applying the changes. The Schema Compare extension is a reliable tool that saves you time and code.



Install the Extension

1. Select the Extensions Icon to view the available extensions.

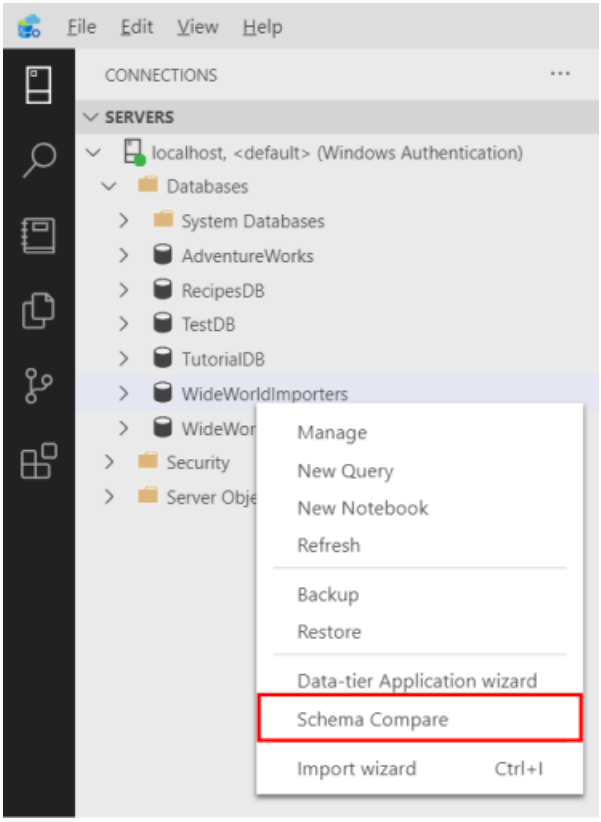


2. Search for the **Schema Compare** extension and select it to view its details. Select **Install** to add the extension.

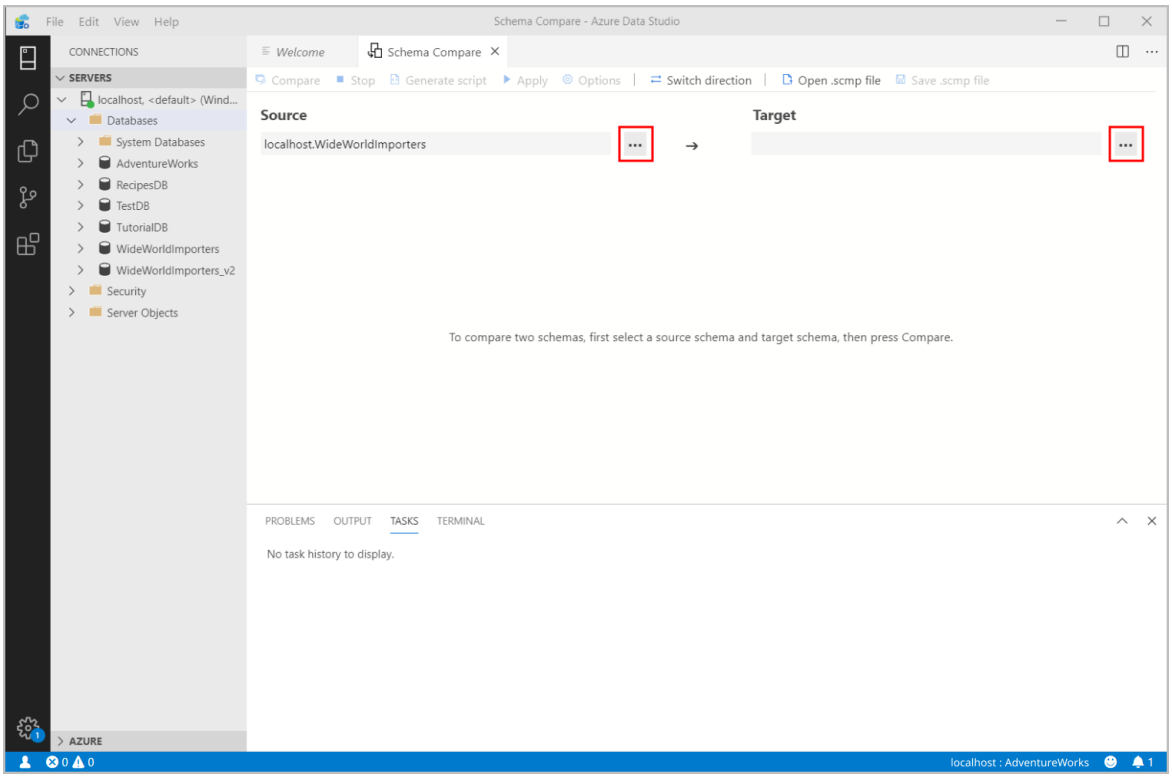
3. Once installed, **Reload** to enable the extension in Azure Data Studio (only required when installing an extension for the first time).

Launch a Schema Compare

1. To open the Schema Compare dialog, **right-click** a database in the Object Explorer and Select **Schema Compare**. The database you select is set as the Source database in the comparison.



2. Select one of the ellipses (...) to change the Source and Target of your Schema Compare and Select OK.



3. To customize your comparison, Select the **Options** button in the toolbar.

4. Select **Compare** to view the results of the comparison.

Next steps

To learn more about Schema Compare, visit Use Schema Compare to Compare Different Database Definitions.
Report issues and feature requests here.

SQL Database Projects extension (Preview)

4/22/2021 • 3 minutes to read • Edit Online

The SQL Database Projects extension (preview) is an extension for developing SQL databases in a project-based development environment.

Features

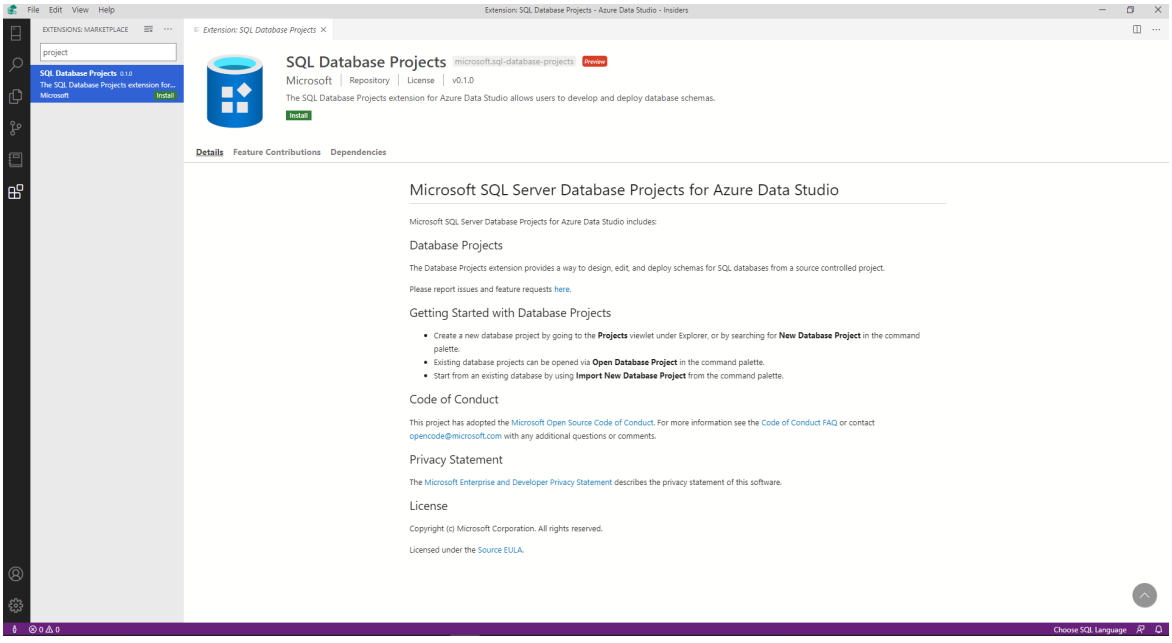
- Create a new blank project or a project from a connected database.
- Open a Project previously created in Azure Data Studio or in SQL Server Data Tools.
- Edit project by adding or removing objects (tables, views, stored procedures) or custom scripts in the project.
- Organize files/scripts in folders.
- Add references to system databases or user dacpac.
- Build single project.
- Deploy single project.
- Load connection details (SQL Windows authentication) and SQLCMD variables from deployment profile.

Watch this short 10-minute video for an introduction to the SQL Database Projects extension in Azure Data Studio:

Installation

1. Open the extensions manager to access the available extensions. To do so, either select the extensions icon or select **Extensions** in the **View** menu.

2. Identify the *SQL Database Projects* extension by typing all or part of the name in the extension search box. Select an available extension to view its details.



3. Select the extension you want and **Install** it.

4. Select **Reload** to enable the extension (only required the first time you install an extension).

5. Select the files icon from the activity bar or select **Explorer** from the **View** menu. A new viewlet for **Projects** is now available.

> **NOTE**
> The .NET Core SDK is required for project build functionality and you will be prompted to install the .NET Core SDK if it cannot be detected by the extension. The .NET Core SDK (v3.1 or higher) can be downloaded and installed from https://dotnet.microsoft.com/download/dotnet-core/3.1.

> **NOTE**
> It is recommended to install the Schema Compare extension alongside the SQL Database Projects extension for full functionality.

.NET Core SDK

The .NET Core SDK is required for project build functionality and you will be prompted to install the .NET Core SDK if it cannot be detected by the extension. The .NET Core SDK (v3.1 or higher) can be downloaded and installed from https://dotnet.microsoft.com/download/dotnet-core/3.1.

Previous installations of the .NET Core SDK may be below the minimum required version and are unsupported for use with the SQL Database Projects extension. If you would like to check currently installed versions of the dotnet SDK, open a terminal and run the following command.

```
dotnet --list-sdks
```

Unsupported .NET Core SDK versions may result in error messages such as:

- `error MSB4018: The "SqlBuildTask" task failed unexpectedly.`
- ```
  error MSB4018: System.TypeInitializationException: The type initializer for 'SqlSchemaModelStaticState'
  threw an exception. ---> System.IO.FileNotFoundException: Could not load file or assembly
  'System.Runtime, Version=4.2.2.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a'. The system cannot
  find the file specified. [c:\Users\ .sqlproj]_
  ```
  (where the linked non-existing file has an unmatched closing square bracket)

Known limitations

- Loading files as link is not supported in Azure Data Studio viewlet today, however the files will be loaded at the top level in tree and build will incorporate these files as expected.
- SQLCLR objects in project are not supported in .NET Core version of DacFx.
- Tasks (build/publish) are not user-defined.
- Publish targets defined by DacFx.
- WSL environment support is limited.

Workspace

SQL database projects in Azure Data Studio are contained within a logical workspace. A workspace manages the folder(s) visible in the Explorer pane as well as the project(s) visible in the Project pane. Adding and removing projects from a workspace can be accomplished through the Azure Data Studio interface in the Projects pane. However, the settings for a workspace can be manually edited in the `.code-workspace` file if necessary.

In the example `.code-workspace` file below, the `folders` array lists all folders included in the Explorer pane and
  ```

the `dataworkspace.projects` array within `settings` lists all the SQL projects included in the Projects pane.

```
{
 "folders": [
  {
   "path": "."
  },
  {
   "name": "WideWorldImportersDW",
   "path": "..\\WideWorldImportersDW"
  }
 ],
 "settings": {
  "dataworkspace.projects": [
   "AdventureWorksLT.sqlproj",
   "..\\WideWorldImportersDW\\WideWorldImportersDW.sqlproj"
  ]
 }
}
```

Next steps

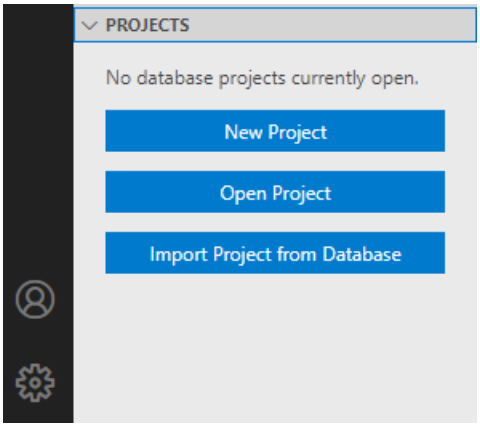
- Getting Started with the SQL Database Projects extension
- Build and Publish a project with SQL Database Projects extension for Azure Data Studio

Getting started with the SQL Database Projects extension (Preview)

11/2/2020 • 2 minutes to read • Edit Online

This article describes three ways to get started with the SQL Database Projects extension:

1. Create a new database project by going to the **Projects** viewlet under Explorer, or by searching for **New Database Project** in the command palette.
2. Existing database projects can be opened via **Open Database Project** in the command palette.
3. Start from an existing database by using **Import New Database Project** from the command palette.



Create an empty database project

In the **Projects** viewlet under **Explorer**, select the **New Project** button and enter a project name in the text input that appears. In the "Select a Folder" dialog that appears, select a directory for the project's folder, .sqlproj file, and other contents to reside in. The empty project is opened and visible in the **Projects** viewlet for editing.

Open an existing project

In the **Projects** viewlet, select the **Open Project** button and open an existing *.sqlproj* file from the file picker that appears. Existing projects can originate from Azure Data Studio or Visual Studio SQL Server Data Tools.

The existing project is opened and its contents are visible in the **Projects** viewlet for editing.

Create a database project from an existing database

In the **Project** viewlet, select the **Import Project from Database** button and connect to a SQL Server. Once the connection is established, select a database from the list available databases and set the name of the project.

Finally, select a target structure of the extraction. The new project is opened and contains SQL scripts for the contents of the selected database.

Build and publish

Deploying the database project is achieved in the SQL Database Projects extension for Azure Data Studio by building the project into a data-tier application file (DACPAC) and publishing to a supported platform. For more on this process, see Build and Publish a Project.

Schema compare

The SQL Database Projects extension interacts with the Schema Compare extension, if installed, to compare the contents of a project to a dacpac or existing database. The resulting schema comparison can be used to view and apply the differences from source to target.

Next steps

- Build and Publish a project with SQL Database Projects extension for Azure Data Studio

Build and Publish a project

3/5/2021 • 2 minutes to read • Edit Online

The build process in the SQL Database Projects extension (preview) for Azure Data Studio allows for *dacpac* creation in Windows, macOS, and Linux environments. The project can be deployed to a local or cloud environment with the publish process.

Prerequisites

- Install and configure SQL Database Projects extension for Azure Data Studio.

Build a Database Project

In the **Projects** viewlet under **Explorer**, right-click the *.sqlproj* root node and select **Build**.

The output pane will automatically appear with the output from the build process. A successful build will conclude with the message:

```
... exited with code: 0
```

Publish a Database Project

After a project is successfully compiled through the build process, the database can be published to a SQL Server instance. To publish a database project, in the **Projects** viewlet under **Explorer**, right-click the *.sqlproj* root node and select **Publish**.

In the **Publish Database** dialog that appears, specify a server connection and the database name to be created.

Next steps

- SQL Database Projects extension for Azure Data Studio
- Build SQL database projects from command line

Build a database project from command line

6/24/2021 • 2 minutes to read • Edit Online

While the SQL Database Project extension (preview) for Azure Data Studio provides a graphical user interface to build a database project, a command line build experience is also available for Windows, macOS, and Linux environments. This article outlines the prerequisites and syntax needed to build a SQL project to dacpac from the command line.

Prerequisites

1. Install and configure SQL Database Projects extension for Azure Data Studio.

2. The following .NET Core dlls and the target file `Microsoft.Data.Tools.Schema.SqlTasks.targets` are required to build a SQL database project from the command line from all platforms supported by the Azure Data Studio extension for SQL Database Projects. These files are created by the extension during the first build completed in the Azure Data Studio interface and placed in the extension's folder under `BuildDirectory`. For example, on Linux, these files are placed in `~\.azuredatastudio\extensions\microsoft.sql-database-projects-x.x.x\BuildDirectory\`. Copy these 11 files to a new and accessible folder or note their location. This location will be referred to as `DotNet Core build folder` in this document.

   - Microsoft.Data.SqlClient.dll
   - Microsoft.Data.Tools.Schema.Sql.dll
   - Microsoft.Data.Tools.Schema.SqlTasks.targets
   - Microsoft.Data.Tools.Schema.Tasks.Sql.dll
   - Microsoft.Data.Tools.Utilities.dll
   - Microsoft.SqlServer.Dac.dll
   - Microsoft.SqlServer.Dac.Extensions.dll
   - Microsoft.SqlServer.TransactSql.ScriptDom.dll
   - Microsoft.SqlServer.Types.dll
   - System.ComponentModel.Composition.dll
   - System.IO.Packaging.dll

3. If the project was created in Azure Data Studio - skip ahead to Build the project from the command line. If the project was created in SQL Server Data Tools (SSDT), open the project in the Azure Data Studio SQL Database project extension. Opening the project in Azure Data Studio automatically updates the `sqlproj` file with three edits, noted below for your information:

   a. Import conditions

```
<Import Condition="'$(NetCoreBuild)' == 'true'"
Project="$(NETCoreTargetsPath)\Microsoft.Data.Tools.Schema.SqlTasks.targets"/>
<Import Condition="'$(NetCoreBuild)' != 'true' AND '$(SQLDBExtensionsRefPath)' != ''"
Project="$(SQLDBExtensionsRefPath)\Microsoft.Data.Tools.Schema.SqlTasks.targets"/>
<Import Condition="'$(NetCoreBuild)' != 'true' AND '$(SQLDBExtensionsRefPath)' == ''"
Project="$(MSBuildExtensionsPath)\Microsoft\VisualStudio\v$(VisualStudioVersion)\SSDT\Microsoft.Data.
Tools.Schema.SqlTasks.targets"/>
```

   b. Package reference

```
<ItemGroup>
    <PackageReference Condition="'$(NetCoreBuild)' == 'true'"
Include="Microsoft.NETFramework.ReferenceAssemblies" Version="1.0.0" PrivateAssets="All"/>
</ItemGroup>
```

c. Clean target, necessary for supporting dual editing in SQL Server Data Tools (SSDT) and Azure Data Studio

```
<Target Name="AfterClean">
    <Delete Files="$(BaseIntermediateOutputPath)\project.assets.json"/>
</Target>
```

Build the project from the command line

From the full .NET folder, use the following command:

```
dotnet build "<sqlproj file path>" /p:NetCoreBuild=true /p:NETCoreTargetsPath="<DotNet Core build folder>"
```

For example, from `/usr/share/dotnet` on Linux:

```
dotnet build "/home/myuser/Documents/DatabaseProject1/DatabaseProject1.sqlproj" /p:NetCoreBuild=true
/p:NETCoreTargetsPath="/home/myuser/.azuredatastudio-insiders/extensions/microsoft.sql-database-projects-
0.1.2/BuildDirectory"
```

Next steps

- SQL Database Projects extension for Azure Data Studio
- Publish SQL database projects

SQL Server Agent extension (Preview)

11/2/2020 • 2 minutes to read • Edit Online

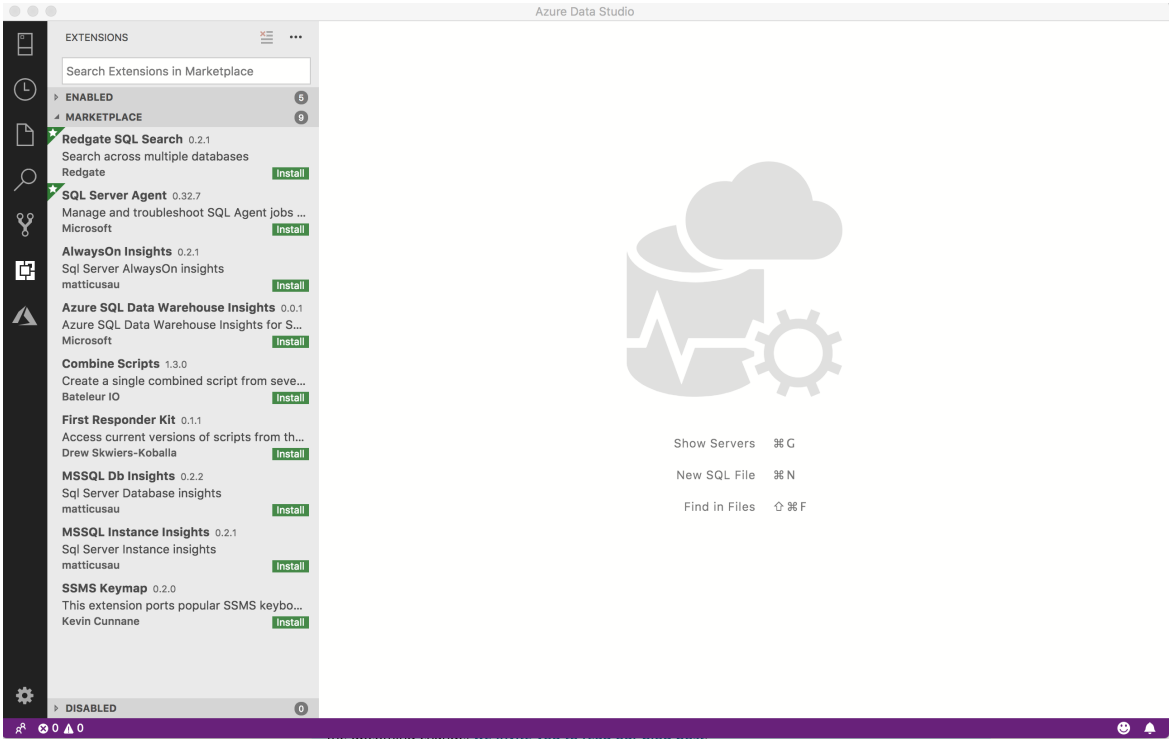
The SQL Server Agent extension (preview) is an extension for managing and troubleshooting SQL Agent jobs and configuration. This extension is currently in preview.

Key actions include:

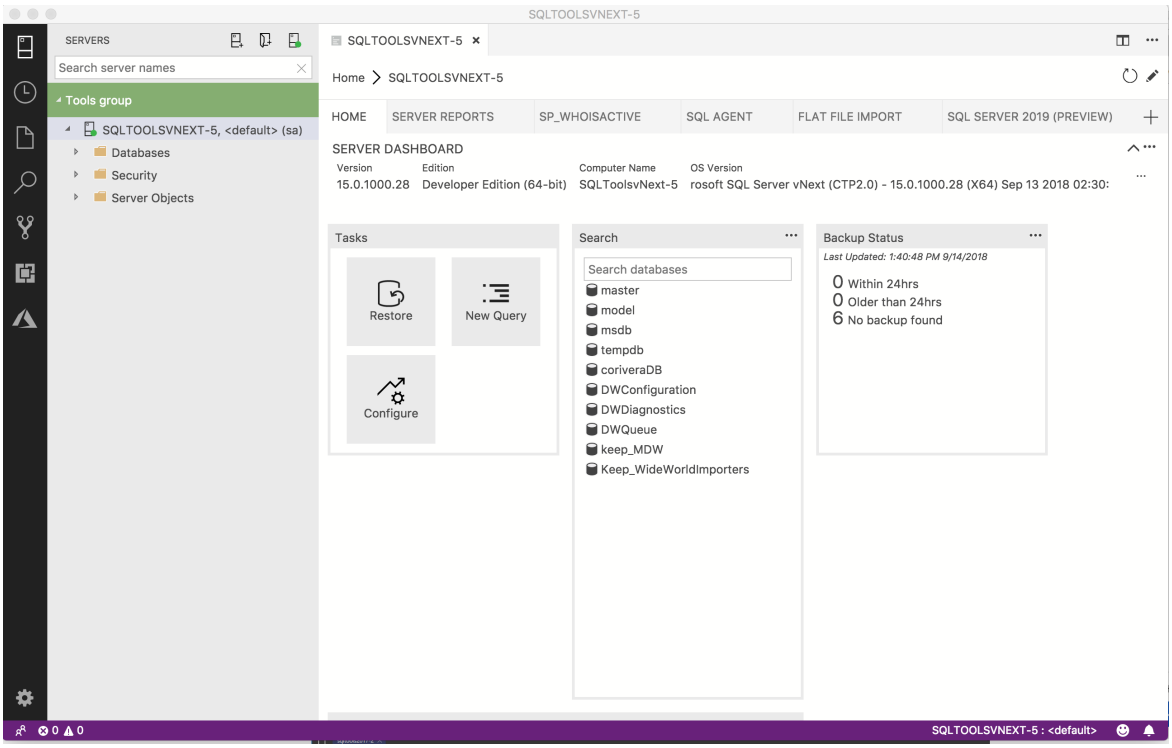
- List SQL Server Agent Jobs Configured on a SQL Server
- View Job History with job execution results
- Basic Job Control to start and stop jobs

Install the SQL Server Agent extension

1. To open the extensions manager and access the available extensions, select the extensions icon, or select **Extensions** in the **View** menu.
2. Select an available extension to view its details.

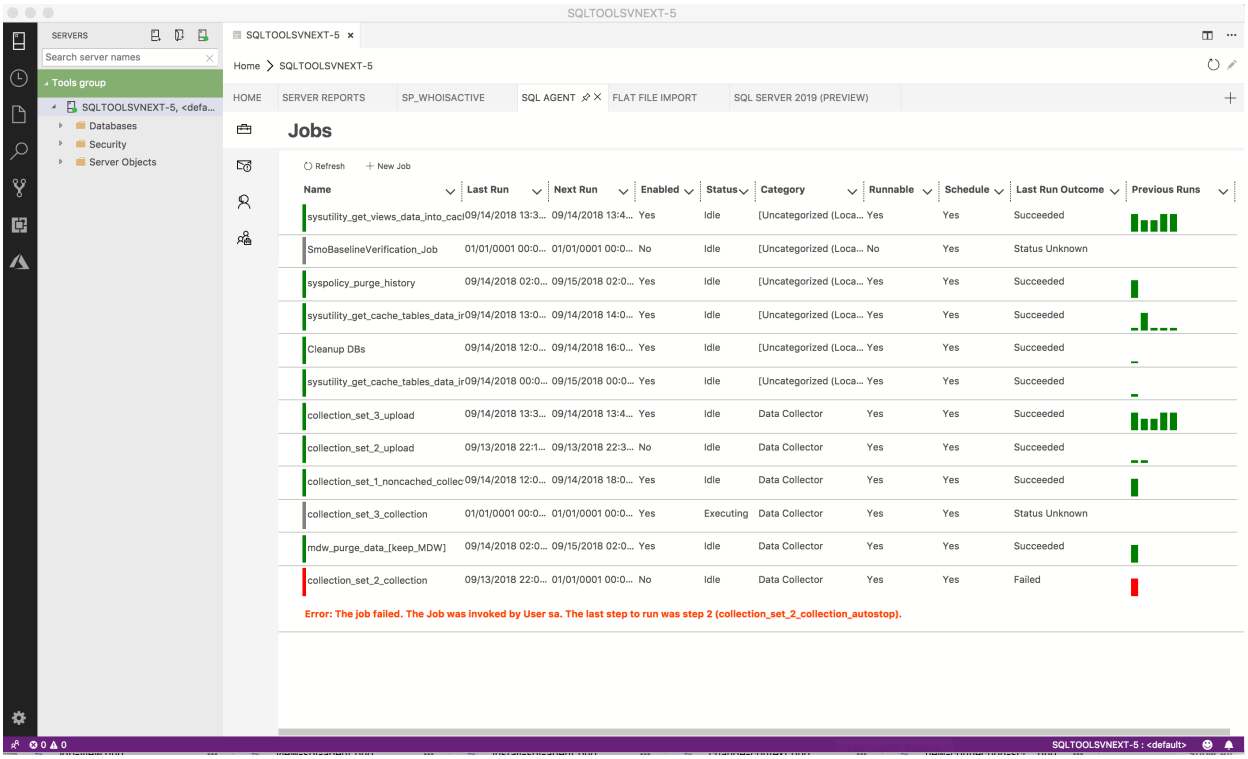


3. Select the extension you want and **Install** it.
4. Select **Reload** to enable the extension (only required the first time you install an extension).
5. Navigate to your management dashboard by right-clicking your server or database and selecting **Manage**.
6. Installed extensions appear as tabs on your management dashboard:



View jobs

When you connect to the SQL Server Agent extension, the first thing you see is a list of all your Agent jobs.



Next steps

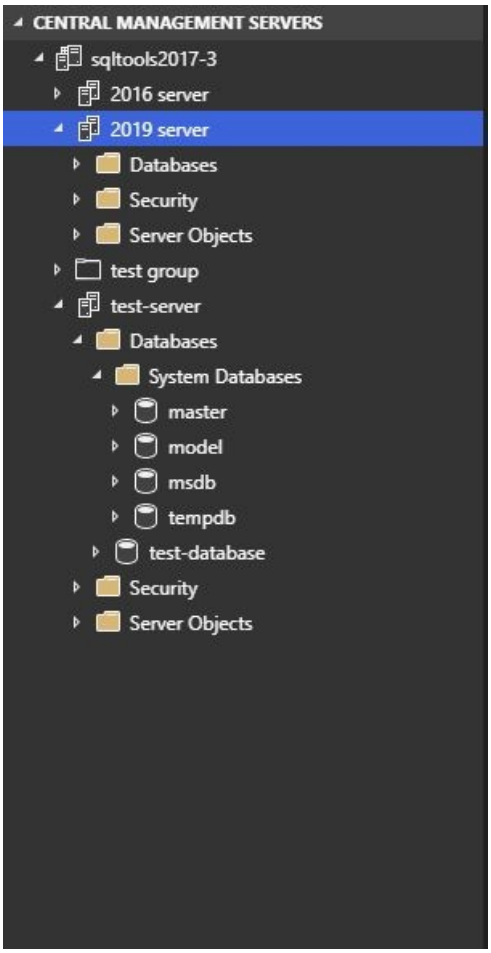
To learn more about SQL Server Agent, check our documentation.

SQL Server Central Management Servers extension (Preview)

11/2/2020 • 2 minutes to read • Edit Online

The Central Management Servers extension allows users to store a list of instances of SQL Server that is organized into one or more groups. Actions that are taken using a CMS group act on all servers in the server group.

This experience is currently in its initial preview. Report issues and feature requests here.



Install the SQL Server Central Management Servers extension

1. To open the extensions manager and access the available extensions, select the extensions icon, or select **Extensions** in the **View** menu.
2. Select an available extension to view its details.
3. Select the extension you want (SQL Server Central Management Servers) and **Install** it.

How do I start Central Management Servers?

Central Management Servers can be viewed by clicking on the Connections icon (Ctrl/Cmd + G). The first time you download the extension, the CMS view will be minimized, and you can open it by click on **Central Management Servers**

Next steps

To learn more conceptually about Central Management Servers, you can read more here.

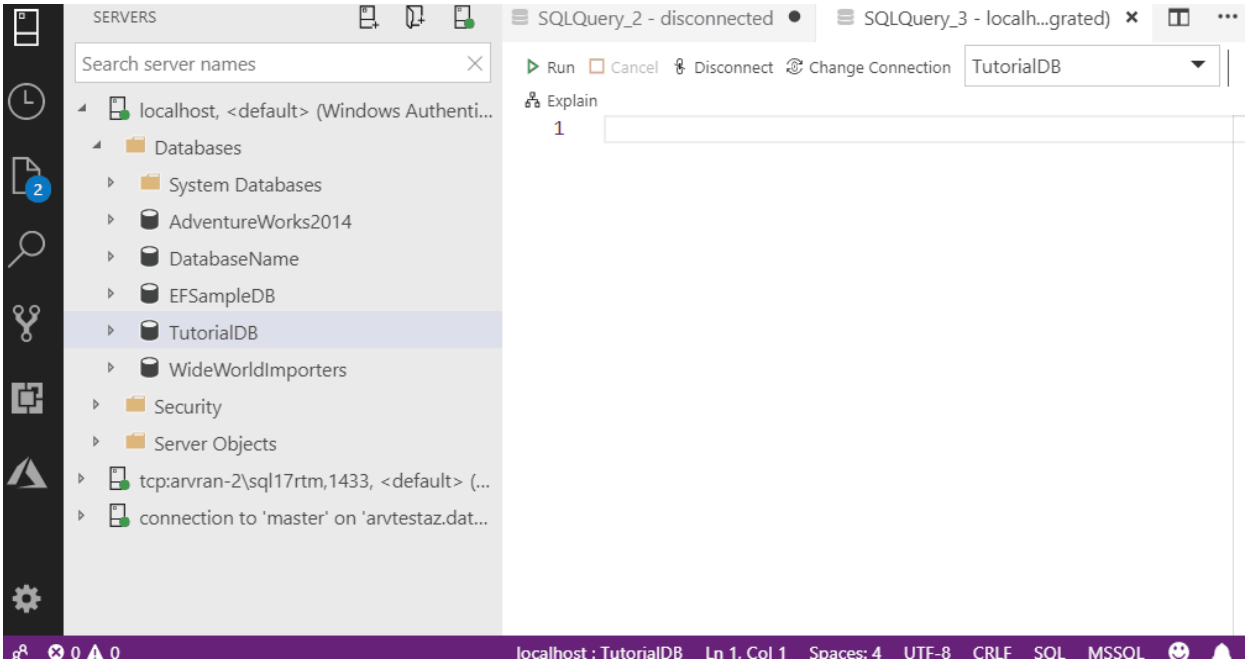
SQL Server dacpac extension

11/2/2020 • 2 minutes to read • Edit Online

The Data-tier Application Wizard provides an easy-to-use wizard experience to deploy and extract dacpac files and import and export bacpac files.

Features

- Deploy a dacpac to a SQL Server instance
- Extract a SQL Server instance to a dacpac
- Create a database from a bacpac
- Export schema and data to a bacpac

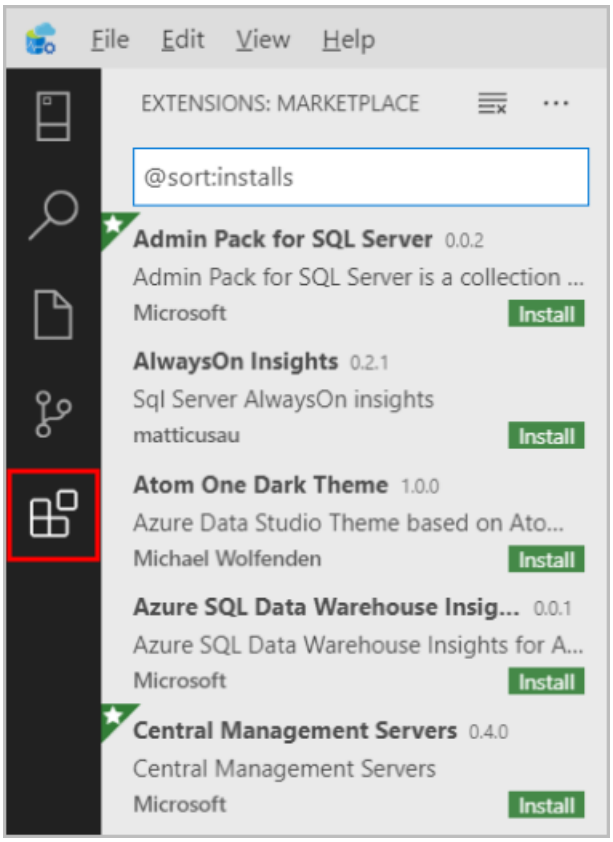


Why would I use the Data-tier Application Wizard?

The wizard makes it easier to manage dacpac and bacpac files, which simplifies the development and deployment of data-tier elements that support your application. To learn more about using Data-tier applications, check out our documentation.

Install the extension

1. Select the Extensions Icon to view the available extensions.

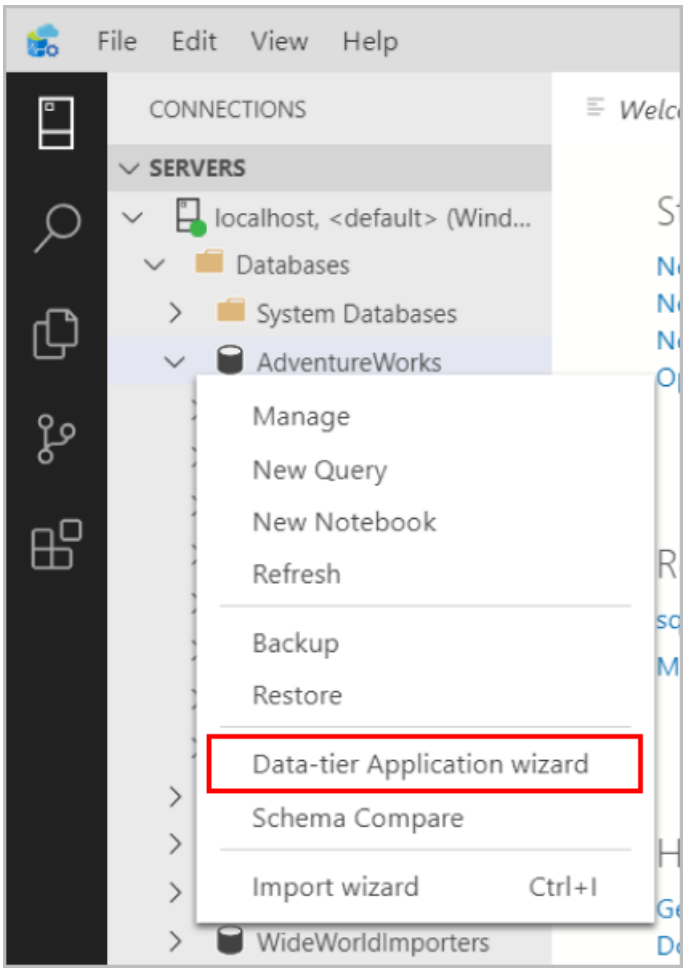


2. Search for the **SQL Server dacpac** extension and select it to view its details. select **Install** to add the extension.

3. Once installed, **Reload** to enable the extension in Azure Data Studio (only required when installing an extension for the first time).

Launch the Data-tier Application Wizard

To launch the wizard, right-click the Databases folder or right-click a specific database in the Object Explorer. Then, select **Data-tier Application Wizard**.



Next steps

To learn more about dacpacs, check out our documentation. Please report issues and feature requests here.

SQL Server Import extension

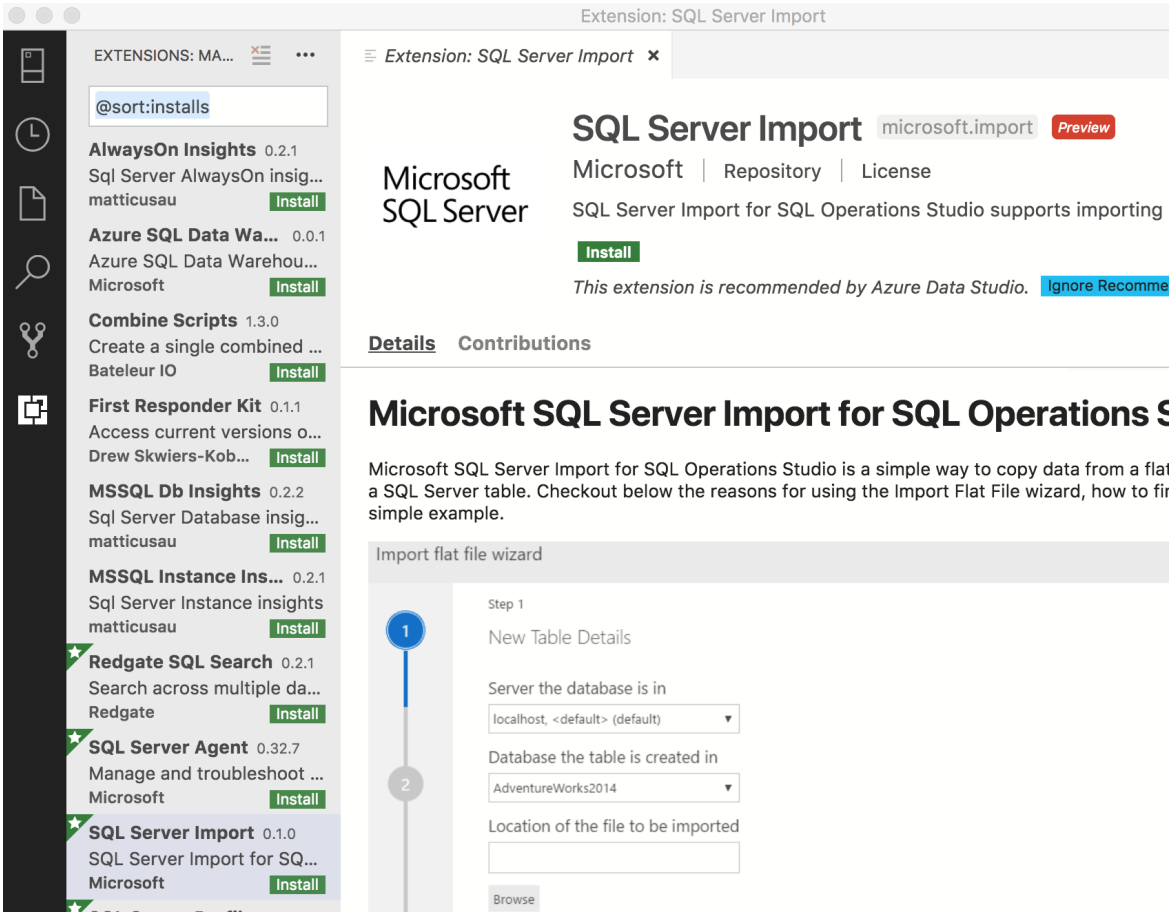
11/2/2020 • 2 minutes to read • Edit Online

The SQL Server Import extension converts .txt and .csv files into a SQL table. This wizard utilizes a Microsoft Research framework known as Program Synthesis using Examples (PROSE) to intelligently parse the file with minimal user input. it's a powerful framework for data wrangling, and it's the same technology that powers Flash Fill-in Microsoft Excel

To learn more about the SSMS version of this feature, you can read this article.

Install the SQL Server Import extension

1. To open the extensions manager and access the available extensions, select the extensions icon, or select **Extensions** in the **View** menu.
2. Select an available extension to view its details.

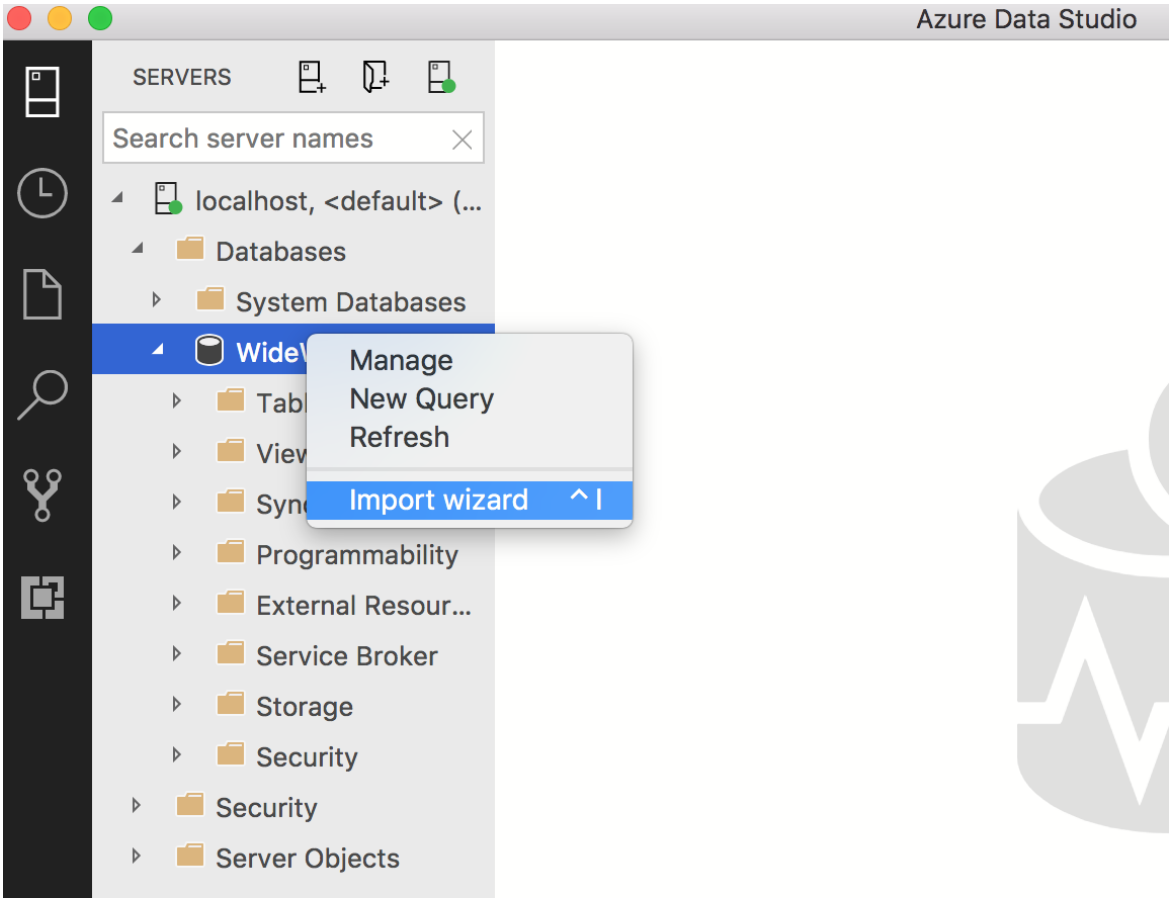


3. Select the extension you want and **Install** it.
4. Select **Reload** to enable the extension (only required the first time you install an extension).

Start Import Wizard

1. To start SQL Server Import, first make a connection to a server in the Servers tab.
2. After you make a connection, drill down to the target database that you want to import a file into a SQL table.

3. Right-click on the database and select **Import Wizard**.

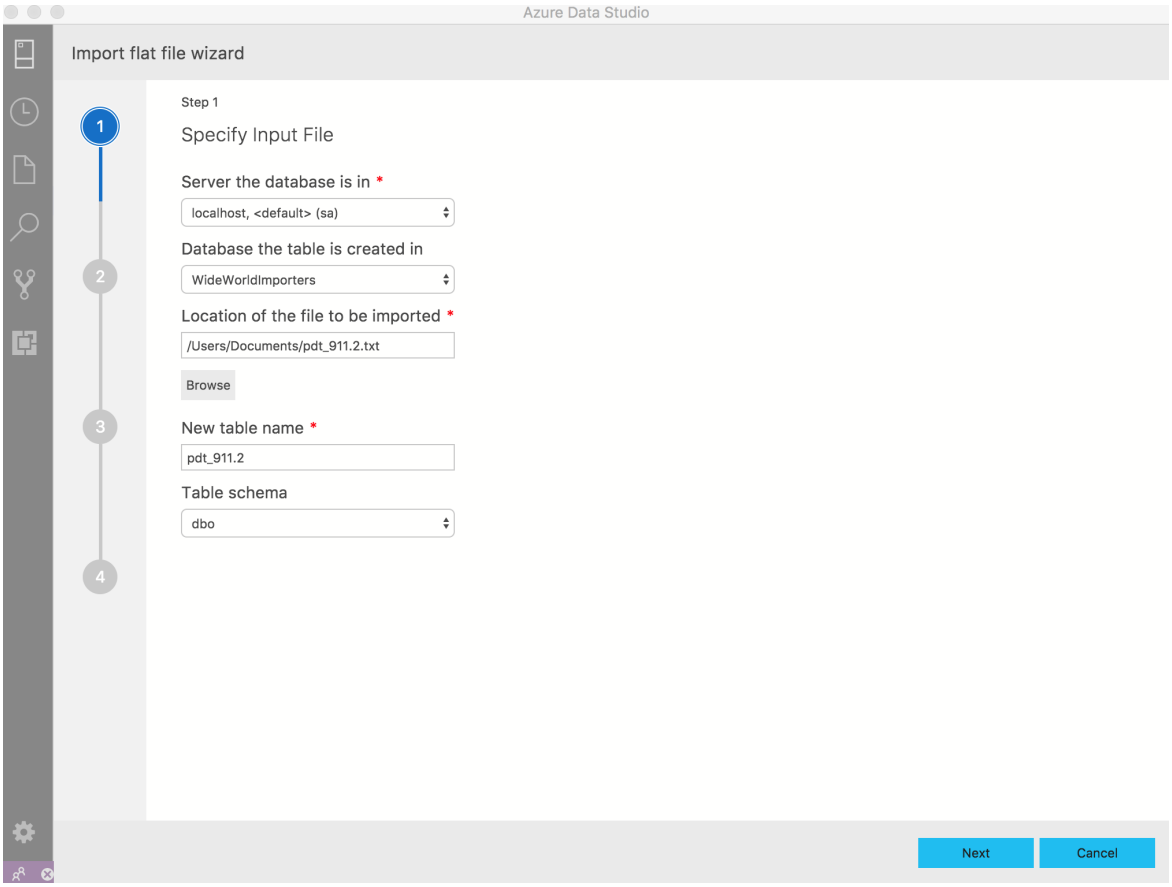


Importing a file

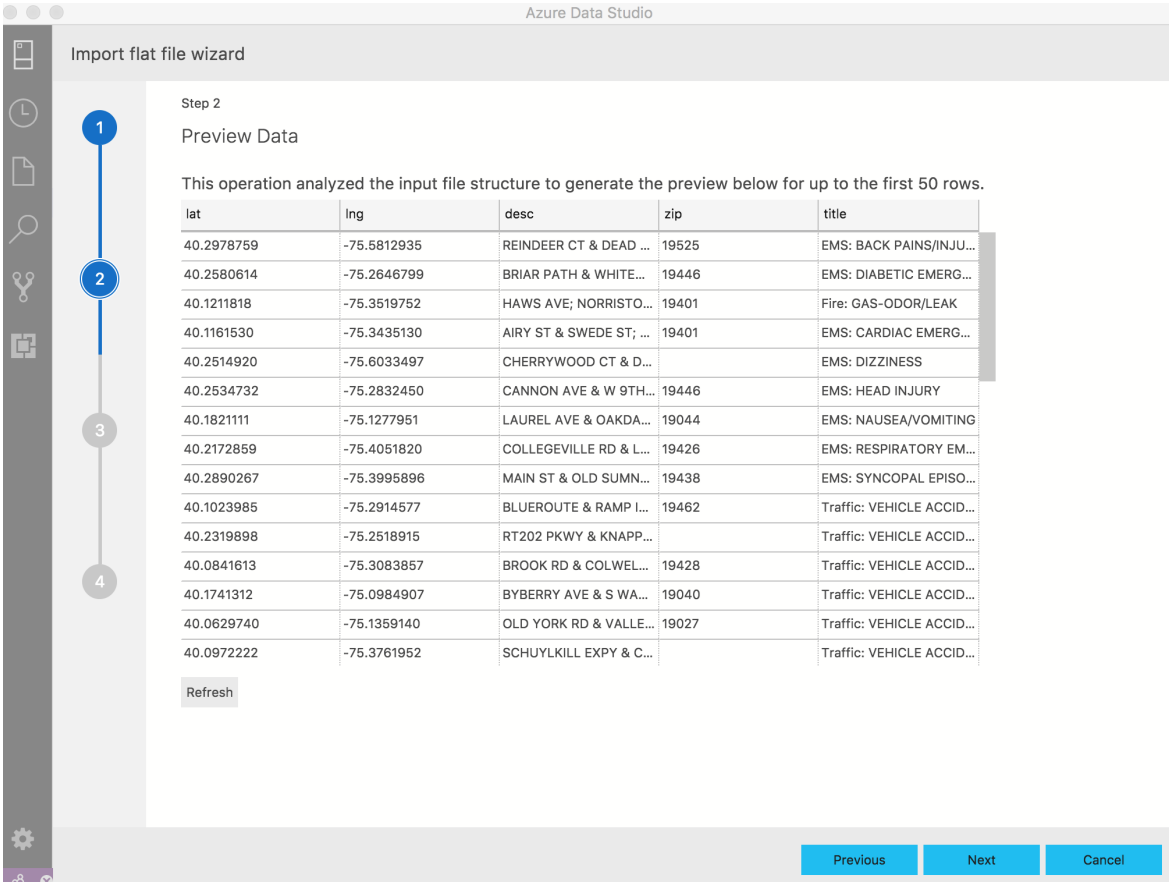
1. When you Right-click to launch the wizard, the server and database are already autofilled. If there are other active connections, you can select in the dropdown.

   Select a file by selecting **Browse.** It should autofill the table name based on the file name, but you can also change it yourself.

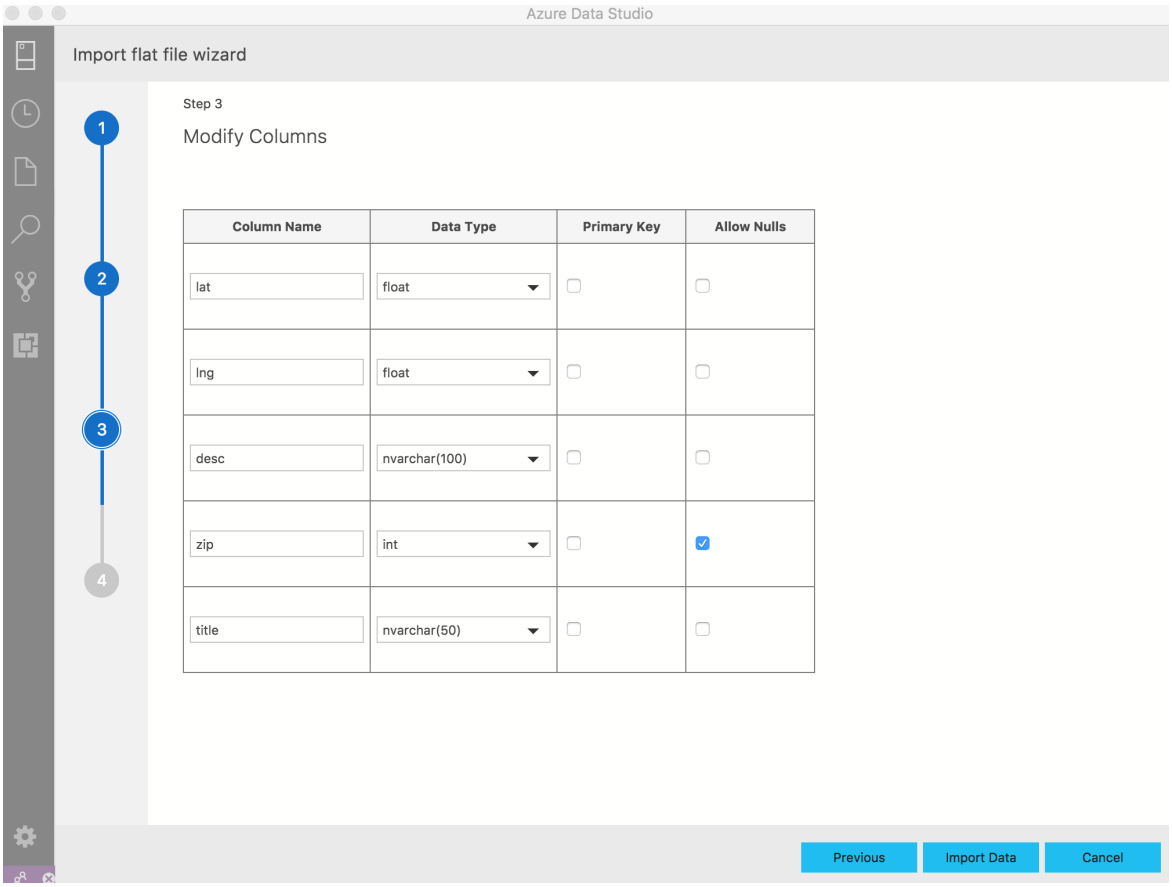
   By default, the schema will be dbo but you can change it. select **Next** to proceed.



2. The wizard will generate a preview based on the first 50 rows. There's no additional action on this page other than verifying the data looks accurate. select **Next** to proceed.

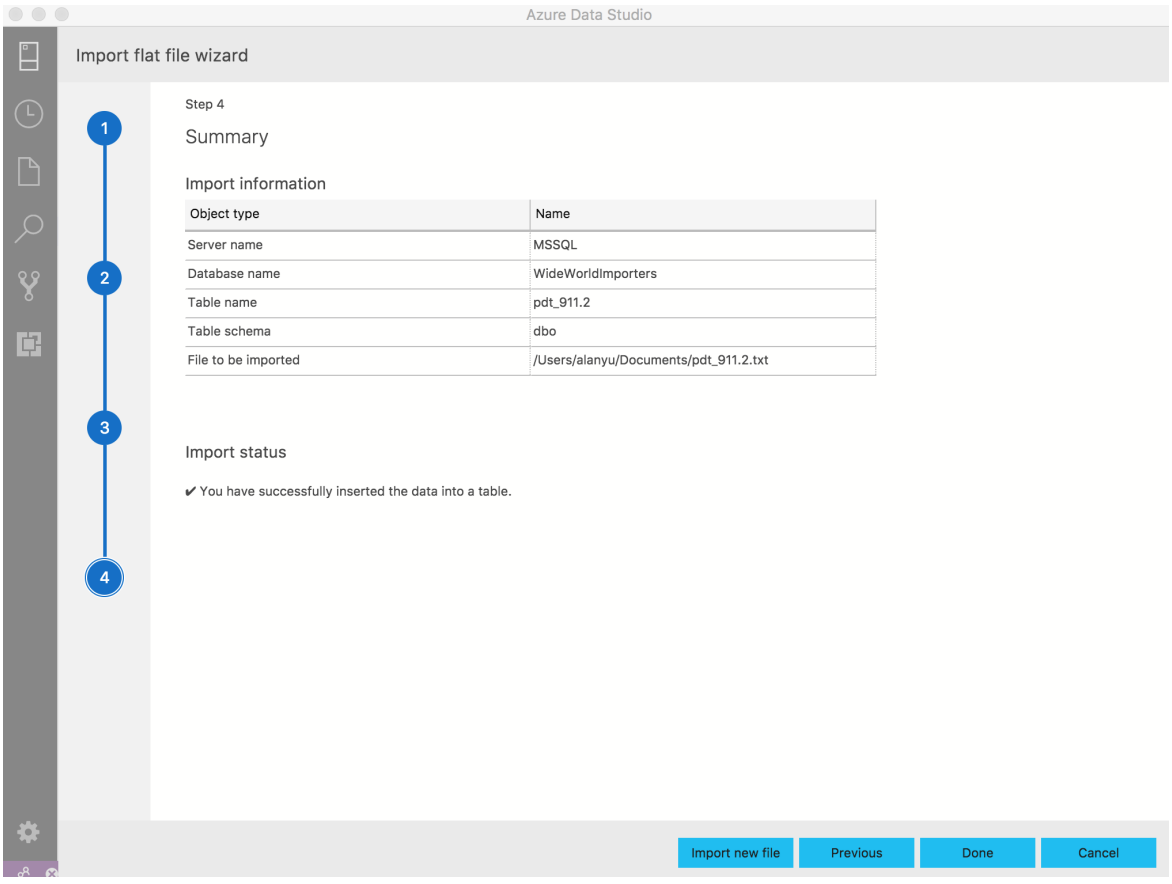


3. On this page, you can make changes to column name, data type, whether it's a primary key, or to allow nulls. You can make as many changes as you like. select **Import Data** to proceed.



4. This page gives a summary of the actions chosen. You can also see whether your table inserted successfully or not.

   You can either select **Done, Previous** if you need to make changes, or **Import new file** to quickly import another file.



5. Verify if your table successfully imported by refreshing your target database or running a SELECT query on the table name.

Next steps

- To learn more about the Import Wizard, read the blog post.
- To learn more about PROSE, read the documentation.

SQL Server Profiler extension (Preview)

6/28/2021 • 2 minutes to read • Edit Online

The SQL Server Profiler extension (preview) provides a simple SQL Server tracing solution similar to SQL Server Management Studio (SSMS) Profiler except built using Extended Events. SQL Server Profiler is easy to use and has good default values for the most common tracing configurations. The UX is optimized for browsing through events and viewing the associated Transact-SQL (T-SQL) text. The SQL Server Profiler for Azure Data Studio also assumes good default values for collecting T-SQL execution activities with an easy to use UX. This extension is currently in preview.

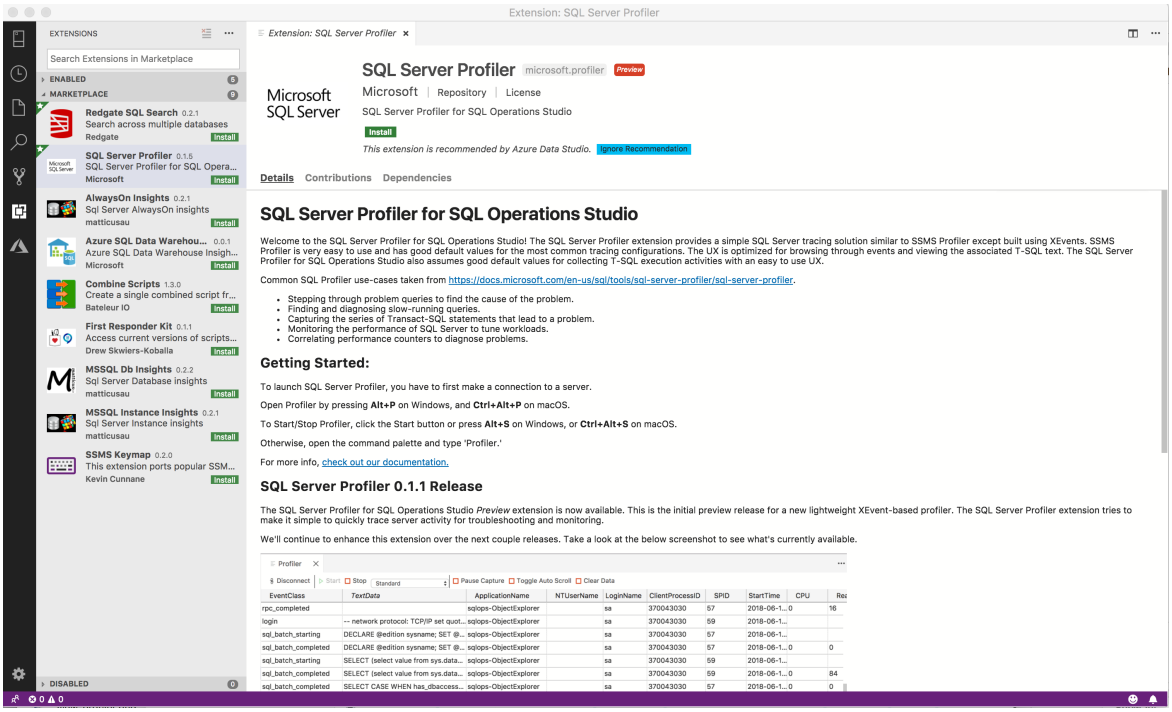
Common SQL Profiler use-cases:

- Stepping through problem queries to find the cause of the problem.
- Finding and diagnosing slow-running queries.
- Capturing the series of Transact-SQL statements that lead to a problem.
- Monitoring the performance of SQL Server to tune workloads.
- Correlating performance counters to diagnose problems.

Install the SQL Server Profiler extension

1. To open the extensions manager and access the available extensions, select the extensions icon, or select **Extensions** in the **View** menu.

2. Select an available extension to view its details.



3. Select the extension you want and **Install** it.

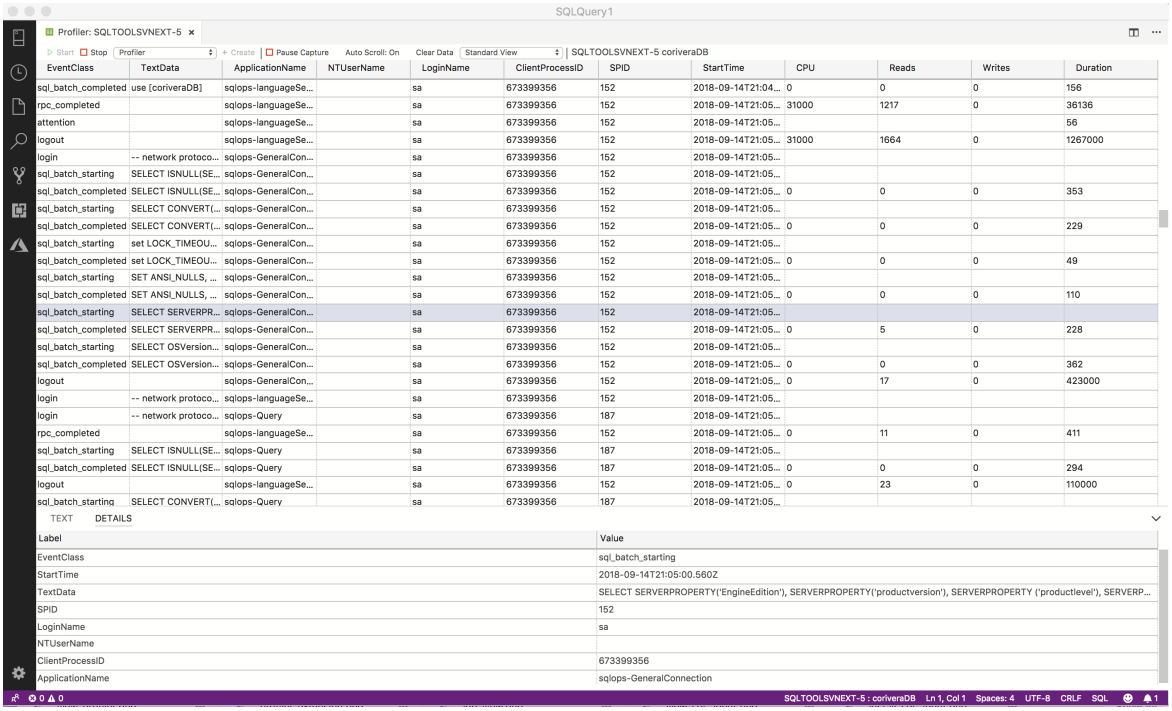
4. Select **Reload** to enable the extension (only required the first time you install an extension).

Start Profiler

1. To start Profiler, first make a connection to a server in the Servers tab.

2. After you make a connection, type **Alt + P** to launch Profiler.

3. To start Profiler, type **Alt + S**. You can now start seeing Extended Events.



4. To stop Profiler, type **Alt + S**. This hotkey is a toggle.

Next steps

To learn more about Profiler and extended events, see Extended Events.

Extend functionality by creating Azure Data Studio extensions

3/5/2021 • 2 minutes to read • Edit Online

Extensions in Azure Data Studio provide an easy way to add more functionality to the base Azure Data Studio installation.

Extensions are provided by the Azure Data Studio team (Microsoft), as well as the third-party community (you!).

Author an extension

If you're interested in extending Azure Data Studio, you can create your own extension and publish it to the extensions gallery.

Write an extension

Prerequisites

To develop an extension, you need Node.js installed and available in your `$PATH`. Node.js includes npm, the Node.js Package Manager, which is used to install the extension generator.

To create your new extension, you can use the Azure Data Studio extension generator. The Yeoman extension generator is a beneficial starting point for extension projects. To start the generator, enter the following command in a command prompt:

```
npm install -g yo generator-azuredatastudio
yo azuredatastudio
```

For an in-depth guide on how to get started with your extension template, see keymap extension, which walks you through the creation of an extension.

Extensibility references

To learn about Azure Data Studio extensibility, see Extensibility overview. You can also see examples of how to use the API in existing samples.

Debug an extension

You can debug your new extension by using the Visual Studio Code extension Azure Data Studio Debug.

To debug your extension:

1. Open your extension with Visual Studio Code.
2. Install the Azure Data Studio Debug extension.
3. Select **F5**, or select the **Debug** icon and then select **Start**.
4. A new instance of Azure Data Studio starts in a special mode (Extension Development Host). This new instance is now aware of your extension.

Create an extension package

After writing your extension, you need to create a VSIX package that installs in Azure Data Studio. You can use vscode-vsce (Visual Studio Code Extensions) to create the VSIX package.

```
npm install -g vsce
cd myExtensionName
vsce package
# The myExtensionName.vsix file has now been generated
```

With a VSIX package, you can share your extension locally and privately by sharing the .vsix file and using the command **Extensions: Install From VSIX File** from the command palette to install the extension in Azure Data Studio.

Publish an extension

To publish your new extension to Azure Data Studio:

1. Add your extension to the extensions gallery.
2. We currently don't have support to host third-party extensions. Instead of downloading the extension, Azure Data Studio has the option to browse to a download page. To set a download page for your extension, set the value of the asset **Microsoft.AzureDataStudio.DownloadPage**.
3. Create a PR against release/extensions branch.
4. Send a review request to the team.

Your extension will be reviewed and added to the extensions gallery.

Publish extension updates

The process to publish updates is similar to publishing the extension. Make sure the version is updated in package.json.

Next steps

See one of the following extension authoring tutorials for step-by-step instructions on how to get started:

- Keymap extension tutorial
- Dashboard extension tutorial
- Notebook extension tutorial
- Jupyter Book extension tutorial
- Wizard extension tutorial

Azure Data Studio extensibility

3/5/2021 • 3 minutes to read • Edit Online

Azure Data Studio has several extensibility mechanisms to customize the user experience and make those customizations available to the entire user community. The core Azure Data Studio platform is built upon Visual Studio Code, so most of the Visual Studio Code extensibility APIs are available. Additionally, we've provided additional extensibility points for data management-specific activities.

Some of the key extensibility points are:

- Visual Studio Code extensibility APIs
- Azure Data Studio extension authoring tools
- Manage Dashboard tab panel contributions
- Insights with Actions experiences
- Azure Data Studio extensibility APIs
- Custom Data Provider APIs

Visual Studio Code extensibility APIs

Because the core Azure Data Studio platform is built upon Visual Studio Code, details about the Visual Studio Code extensibility APIs are found in the Extension Authoring and Extension API documentation on the Visual Studio Code website.

> **NOTE**
> Azure Data Studio releases are aligned with a recent version of VS Code, however the included VS Code engine may not be the current VS Code release. For example, in November 2020 the VS Code engine in Azure Data Studio is 1.48 and the current VS Code version is 1.51. The error message "Unable to install extension '' as it is not compatible with VS Code '' when installing an extension is caused by an extension that has a later VS Code engine version defined in the package manifest (`package.json`). You can verify the VS Code engine version in your Azure Data Studio through the **Help** menu under **About**.

Manage Dashboard tab panel contributions

For details, see Contribution Points and Context Variables.

Azure Data Studio extensibility APIs

For details, see Extensibility APIs.

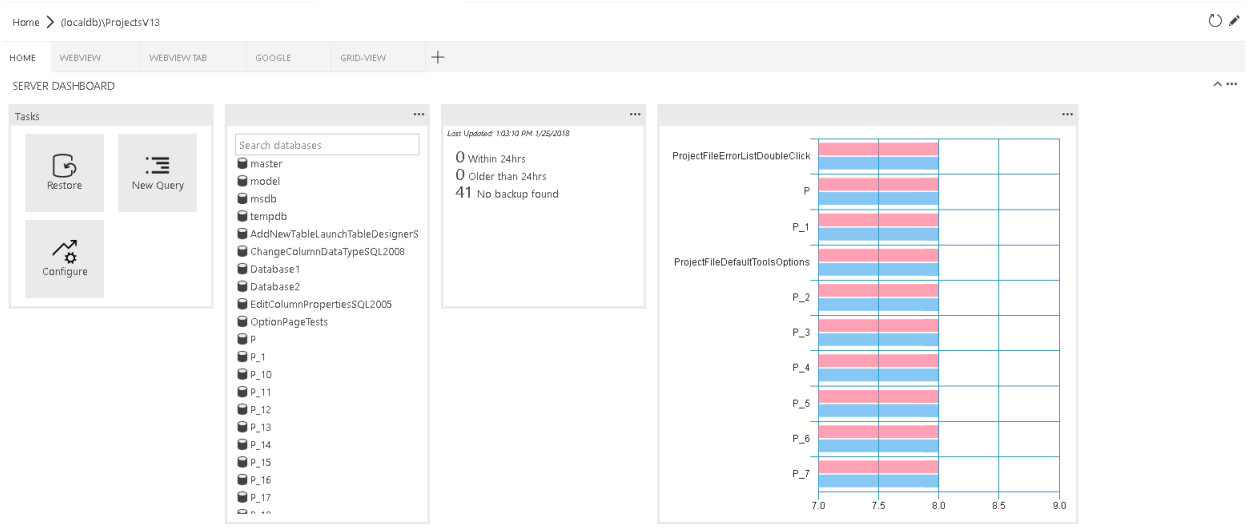
Contribution points

This section covers the various contribution points that are defined in the package.json extension manifest.

The IntelliSense is supported inside azuredatastudio.

Dashboard contribution points

Contribute a tab, container, and/or insight widget to the dashboard.



`dashboard.tabs`

Dashboard.tabs creates the tab sections inside the dashboard page. It expects an object or an array of objects.

```
"dashboard.tabs": [
{
 "id": "test-tab1",
 "title": "Test 1",
 "description": "The test 1 displays a list of widgets.",
 "when": "connectionProvider == 'MSSQL' && !mssql:iscloud",
 "alwaysShow": true,
 "container": {
  ...
 }
}
]
```

`dashboard.containers`

Instead of specifying dashboard container inline (inside the dashboard.tab). You can register containers using dashboard.containers. It accepts an object or an array of the object.

```
"dashboard.containers": [
{
 "id": "innerTab1",
 "container": {
  ...
 }
},
{
 "id": "innerTab2",
 "container": {
    ...
 }
}
]
```

To refer to registered container, specify the id of the container

```
"dashboard.tabs": [
{
 ...
 "container": {
  "innerTab1": {
  }
 }
}
]
```

`dashboard.insights`

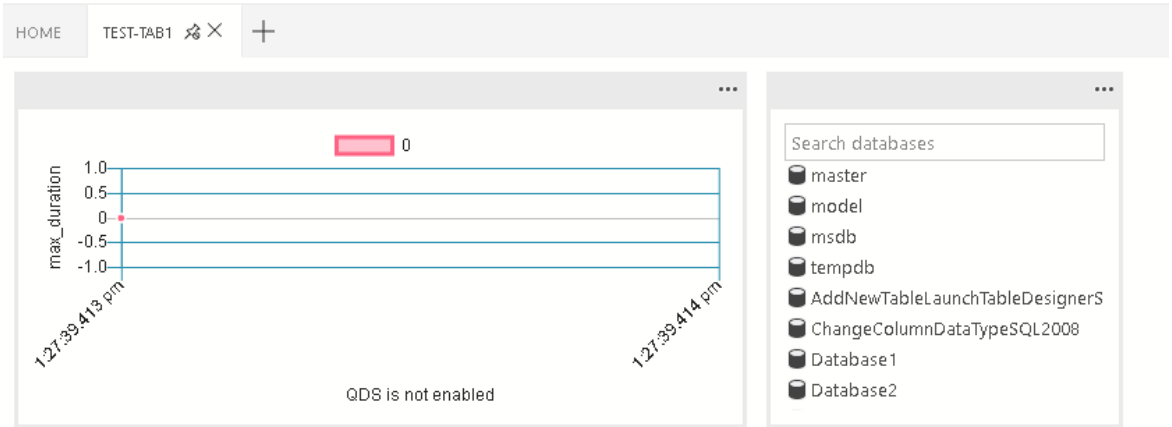
You can register insights using dashboard.insights. This is similar to Tutorial: Build a custom insight widget

```
"dashboard.insights": {
"id": "my-widget"
"type": {
 "count": {
  "dataDirection": "vertical",
  "dataType": "number",
  "legendPosition": "none",
          "labelFirstColumn": false,
  "columnsAsLabels": false
       }
   },
   "queryFile": "{your file folder}/activeSession.sql"
}
```

Dashboard container types

There are currently four supported container types:

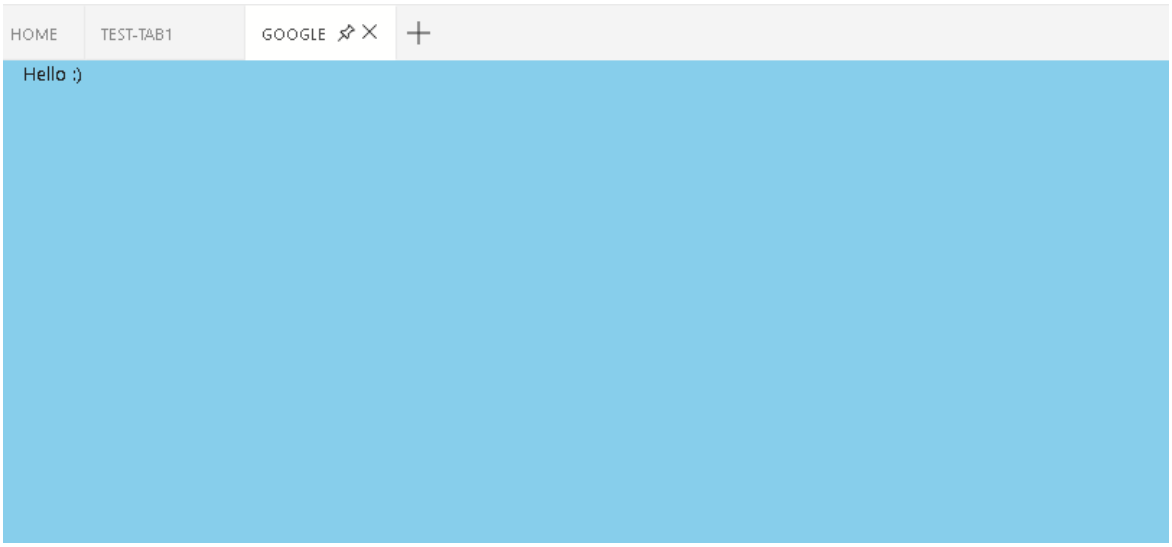
1. `widgets-container`



The list of widgets that will be displayed in the container. It's a flow layout. It accepts the list of widgets.

```
"container": {
 "widgets-container": [
 {
  "widget": {
   "query-data-store-db-insight": {
   }
  }
 },
 {
  "widget": {
   "explorer-widget": {
   }
  }
 }
  ]
}
```

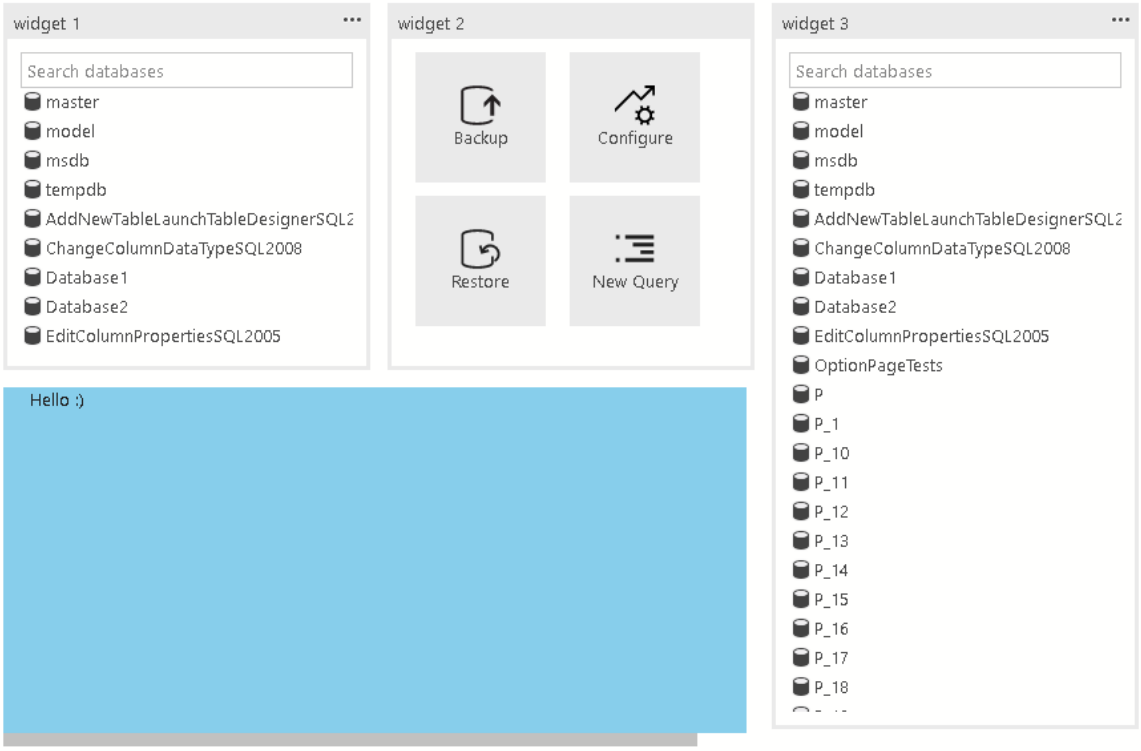
2. `webview-container`



The webview will be displayed in the entire container. It expects webview id to be the same is tab ID

```
"container": {
 "webview-container": null
}
```

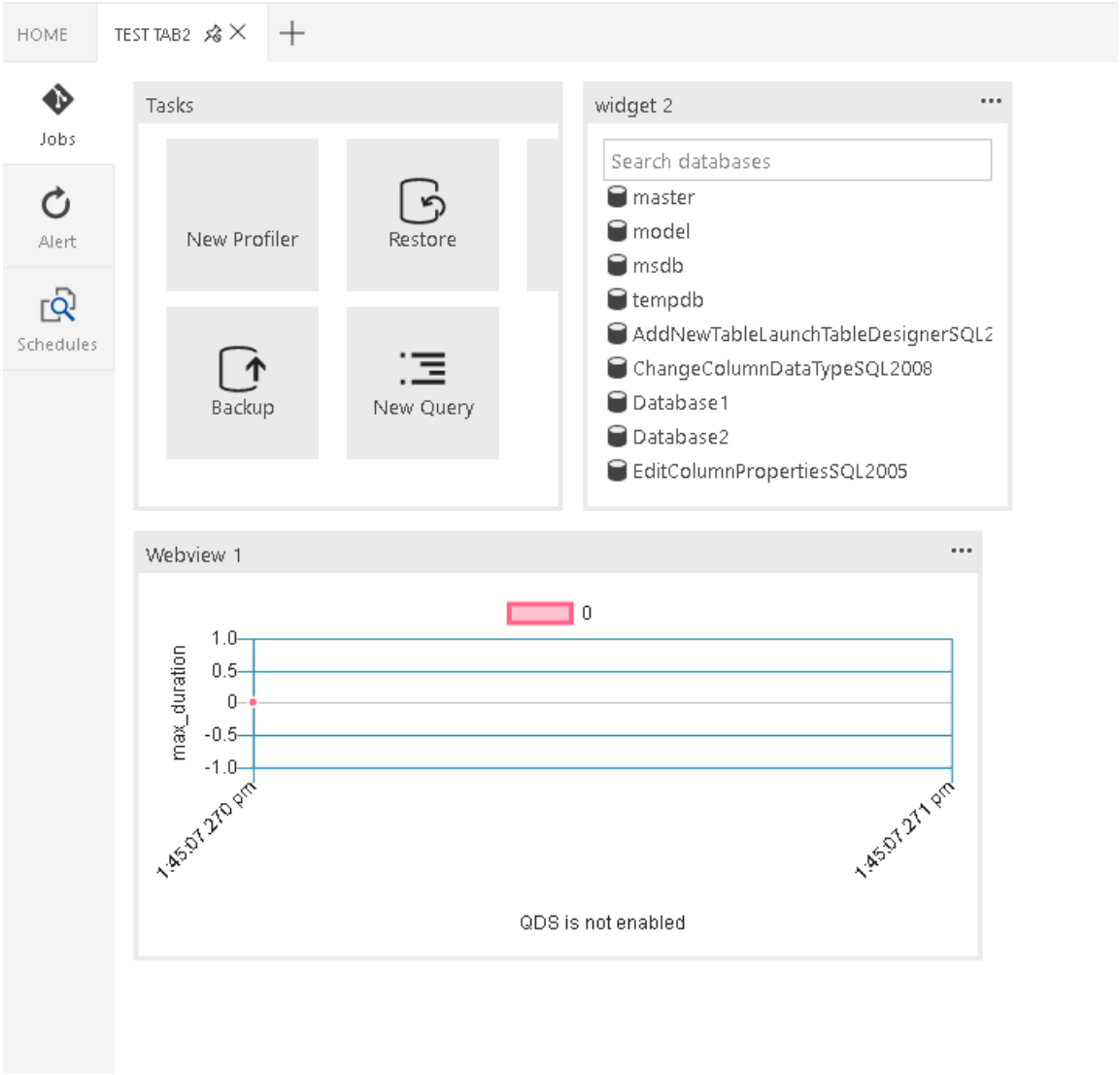
3. `grid-container`



The list of widgets or webviews that will be displayed in the grid layout

```
 "container": {
  "grid-container": [
    {
     "name": "widget 1",
     "widget": {
      "explorer-widget": {
      }
     },
     "row":0,
     "col":0
    },
    {
     "name": "widget 2",
     "widget": {
      "tasks-widget": {
       "backup",
       "restore",
       "configureDashboard",
       "newQuery"
      }
     },
     "row":0,
     "col":1
    },
    {
     "name": "Webview 1",
     "webview": {
      "id": "google"
     },
     "row":1,
     "col":0,
     "colspan":2
    },
    {
     "name": "widget 3",
     "widget": {
      "explorer-widget": {}
     },
     "row":0,
     "col":3,
     "rowspan":2
    },
]
```

4. `nav-section`



The navigation section will be displayed in the container

```
"container": {
 "nav-section": [
  {
   "id": "innerTab1",
   "title": "inner-tab1",
   "icon": {
    "light": "./icons/tab1Icon.svg",
    "dark": "./icons/tab1Icon_dark.svg"
   },
   "container": {
    ...
   }
  },
  {
   "id": "innerTab2",
   "title": "inner-tab2",
   "icon": {
    "light": "./icons/tab2Icon.svg",
    "dark": "./icons/tab2Icon_dark.svg"
   },
   "container": {
    ...
   }
  }
 ]
}
```

Context variables

For general information about context in Visual Studio Code and subsequently Azure Data Studio, see Extensibility.

In Azure Data Studio, we have specific context around database connections available for extensions.

Dashboard

In dashboard, we provide the following context variables:

CONTEXT VARIABLE	DESCRIPTION
`connectionProvider`	A string of the identifier for the provider of the current connection. Ex. `connectionProvider == 'MSSQL'`.
`serverName`	A string of the server name of the current connection. Ex. `serverName == 'localhost'`.
`databaseName`	A string of the database name of the current connection. Ex. `databaseName == 'master'`.
`connection`	The full connection profile object for the current connection (IConnectionProfile)
`dashboardContext`	A string of the context of the page the dashboard is currently on. Either 'database' or 'server'. Ex. `dashboardContext == 'database'`

Azure Data Studio extensibility APIs

11/2/2020 • 4 minutes to read • Edit Online

Azure Data Studio provides an API that extensions can use to interact with other parts of Azure Data Studio, such as Object Explorer. These APIs are available from the `src/sql/azdata.d.ts` file and are described below.

Connection Management

`azdata.connection`

Top-level Functions

- `getCurrentConnection(): Thenable<azdata.connection.Connection>` Gets the current connection based on the active editor or Object Explorer selection.
- `getActiveConnections(): Thenable<azdata.connection.Connection[]>` Gets a list of all the user's connections that are active. Returns an empty list if there are no such connections.
- `getCredentials(connectionId: string): Thenable<{ [name: string]: string }>` Gets a dictionary containing the credentials associated with a connection. These would otherwise be returned as part of the options dictionary under a `azdata.connection.Connection` object but get stripped from that object.

`Connection`

- `options: { [name: string]: string }` The dictionary of connection options
- `providerName: string` The name of the connection provider (e.g. "MSSQL")
- `connectionId: string` The unique identifier for the connection

Example Code

```
> let connection = azdata.connection.getCurrentConnection();
connection: {
 providerName: 'MSSQL',
 connectionId: 'd97bb63a-466e-4ef0-ab6f-00cd44721dcc',
 options: {
  server: 'mairvine-sql-server',
  user: 'sa',
  authenticationType: 'sqlLogin',
  ...
 },
 ...
}
> let credentials = azdata.connection.getCredentials(connection.connectionId);
credentials: {
 password: 'abc123'
}
```

Object Explorer

`azdata.objectexplorer`

Top-level Functions

- `getNode(connectionId: string, nodePath?: string): Thenable<azdata.objectexplorer.ObjectExplorerNode>` Get an Object Explorer node corresponding to the given connection and path. If no path is given, it

returns the top-level node for the given connection. If there is no node at the given path, it returns `undefined`. Note: The `nodePath` for an object is generated by the SQL Tools Service backend and is difficult to construct by hand. Future API improvements will allow you to get nodes based on metadata you provide about the node, such as name, type and schema.

- `getActiveConnectionNodes(): Thenable<azdata.objectexplorer.ObjectExplorerNode>` Get all active Object Explorer connection nodes.
- `findNodes(connectionId: string, type: string, schema: string, name: string, database: string, parentObjectNames: string[]): Thenable<azdata.objectexplorer.ObjectExplorerNode[]>` Find all Object Explorer nodes that match the given metadata. The `schema`, `database`, and `parentObjectNames` arguments should be `undefined` when they are not applicable. `parentObjectNames` is a list of non-database parent objects, from highest to lowest level in Object Explorer, that the desired object is under. For example, when searching for a column "column1" that belongs to a table "schema1.table1" and database "database1" with connection ID `connectionId`, call `findNodes(connectionId, 'Column', undefined, 'column1', 'database1', ['schema1.table1'])`. Also see the list of types that Azure Data Studio supports by default for this API call.

ObjectExplorerNode

- `connectionId: string` The id of the connection that the node exists under
- `nodePath: string` The path of the node, as used for a call to the `getNode` function.
- `nodeType: string` A string representing the type of the node
- `nodeSubType: string` A string representing the subtype of the node
- `nodeStatus: string` A string representing the status of the node
- `label: string` The label for the node as it appears in Object Explorer
- `isLeaf: boolean` Whether the node is a leaf node and therefore has no children
- `metadata: azdata.ObjectMetadata` Metadata describing the object represented by this node
- `errorMessage: string` Message shown if the node is in an error state
- `isExpanded(): Thenable<boolean>` Whether the node is currently expanded in Object Explorer
- `setExpandedState(expandedState: vscode.TreeItemCollapsibleState): Thenable<void>` Set whether the node is expanded or collapsed. If the state is set to None, the node will not be changed.
- `setSelected(selected: boolean, clearOtherSelections?: boolean): Thenable<void>` Set whether the node is selected. If `clearOtherSelections` is true, clear any other selections when making the new selection. If it is false, leave any existing selections. `clearOtherSelections` defaults to true when `selected` is true and false when `selected` is false.
- `getChildren(): Thenable<azdata.objectexplorer.ObjectExplorerNode[]>` Get all the child nodes of this node. Returns an empty list if there are no children.
- `getParent(): Thenable<azdata.objectexplorer.ObjectExplorerNode>` Get the parent node of this node. Returns undefined if there is no parent.

Example Code

```
private async interactWithOENode(selectedNode: azdata.objectexplorer.ObjectExplorerNode): Promise<void> {
 let choices = ['Expand', 'Collapse', 'Select', 'Select (multi)', 'Deselect', 'Deselect (multi)'];
 if (selectedNode.isLeaf) {
  choices[0] += ' (is leaf)';
  choices[1] += ' (is leaf)';
 } else {
  let expanded = await selectedNode.isExpanded();
  if (expanded) {
   choices[0] += ' (is expanded)';
  } else {
   choices[1] += ' (is collapsed)';
  }
 }
 let parent = await selectedNode.getParent();
 if (parent) {
  choices.push('Get Parent');
 }
 let children = await selectedNode.getChildren();
 children.forEach(child => choices.push(child.label));
 let choice = await vscode.window.showQuickPick(choices);
 let nextNode: azdata.objectexplorer.ObjectExplorerNode = undefined;
 if (choice === choices[0]) {
  selectedNode.setExpandedState(vscode.TreeItemCollapsibleState.Expanded);
 } else if (choice === choices[1]) {
  selectedNode.setExpandedState(vscode.TreeItemCollapsibleState.Collapsed);
 } else if (choice === choices[2]) {
  selectedNode.setSelected(true);
 } else if (choice === choices[3]) {
  selectedNode.setSelected(true, false);
 } else if (choice === choices[4]) {
  selectedNode.setSelected(false);
 } else if (choice === choices[5]) {
  selectedNode.setSelected(false, true);
 } else if (choice === 'Get Parent') {
  nextNode = parent;
 } else {
  let childNode = children.find(child => child.label === choice);
  nextNode = childNode;
 }
 if (nextNode) {
  let updatedNode = await azdata.objectexplorer.getNode(nextNode.connectionId, nextNode.nodePath);
  this.interactWithOENode(updatedNode);
 }
}

vscode.commands.registerCommand('mssql.objectexplorer.interact', () => {
 azdata.objectexplorer.getActiveConnectionNodes().then(activeConnections => {
  vscode.window.showQuickPick(activeConnections.map(connection => connection.label + ' ' +
connection.connectionId)).then(selection => {
   let selectedNode = activeConnections.find(connection => connection.label + ' ' + connection.connectionId
=== selection);
   this.interactWithOENode(selectedNode);
  });
 });
});
```

Proposed APIs

We have added proposed APIs to allow extensions to display custom UI in dialogs, wizards, and document tabs, among other capabilities. See the proposed API types file for more documentation, though be aware that these APIs are subject to change at any time. Examples of how to use some of these APIs can be found in the "sqlservices" sample extension.

Create an Azure Data Studio Keymap extension

3/5/2021 • 4 minutes to read • Edit Online

This tutorial demonstrates how to create a new Azure Data Studio extension. The extension creates familiar SSMS key bindings in Azure Data Studio.

In this article you learn how to:

- Create an extension project
- Install the extension generator
- Create your extension
- Add custom key-bindings to your extension
- Test your extension
- Package your extension
- Publish your extension to the marketplace

Prerequisites

Azure Data Studio is built on the same framework as Visual Studio Code, so extensions for Azure Data Studio are built using Visual Studio Code. To get started, you need the following components:

- Node.js installed and available in your `$PATH`. Node.js includes npm, the Node.js Package Manager, which is used to install the extension generator.
- Visual Studio Code to debug the extension.
- The Azure Data Studio Debug extension (optional). This lets you test your extension without needing to package and install it into Azure Data Studio.
- Ensure `azuredatastudio` is in your path. For Windows, make sure you choose the `Add to Path` option in setup.exe. For Mac or Linux, run **Install 'azuredatastudio' command in PATH** from the Command Palette in Azure Data Studio.

Install the extension generator

To simplify the process of creating extensions, we've built an extension generator using Yeoman. To install it, run the code in the command prompt below:

```
npm install -g yo generator-azuredatastudio
```

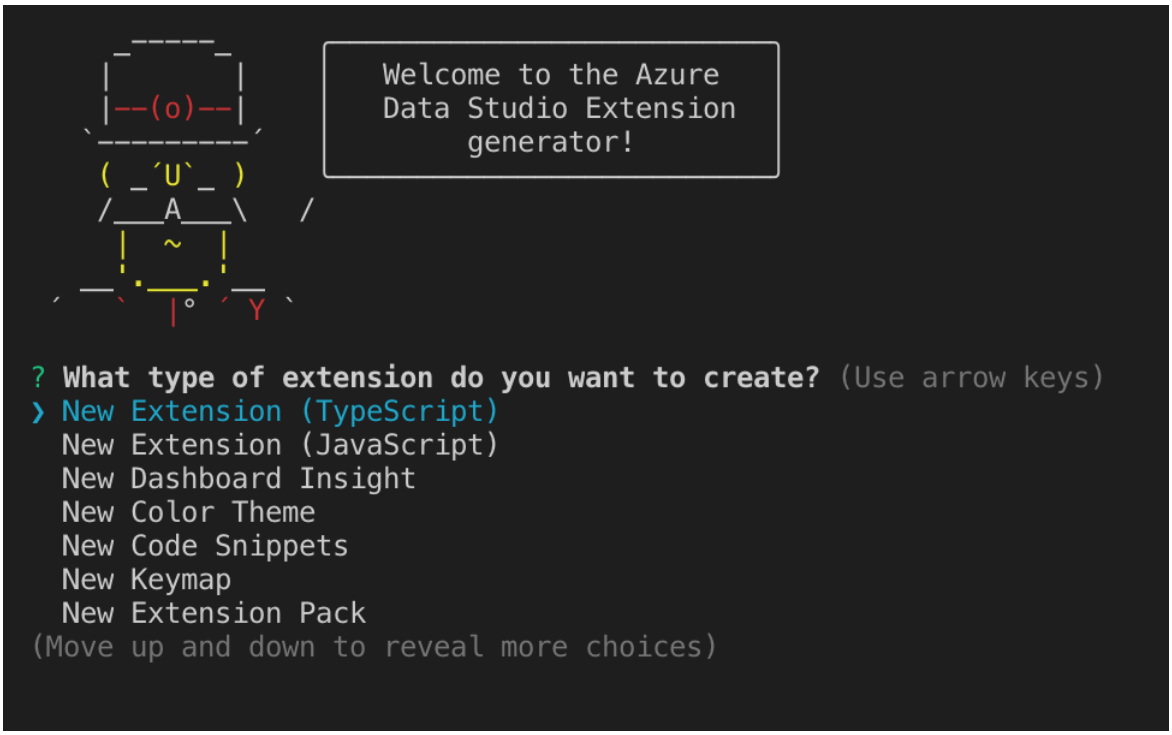
Create your keymap extension

To create an extension:

1. Launch the extension generator with the following command:

   `yo azuredatastudio`

2. Choose **New Keymap** from the list of extension types:



3. Follow the steps to fill in the extension name (for this tutorial, use **ssmskeymap2**), and add a description.

Completing the previous steps creates a new folder. Open the folder in Visual Studio Code and you're ready to create your own key binding extension!

Add a keyboard shortcut

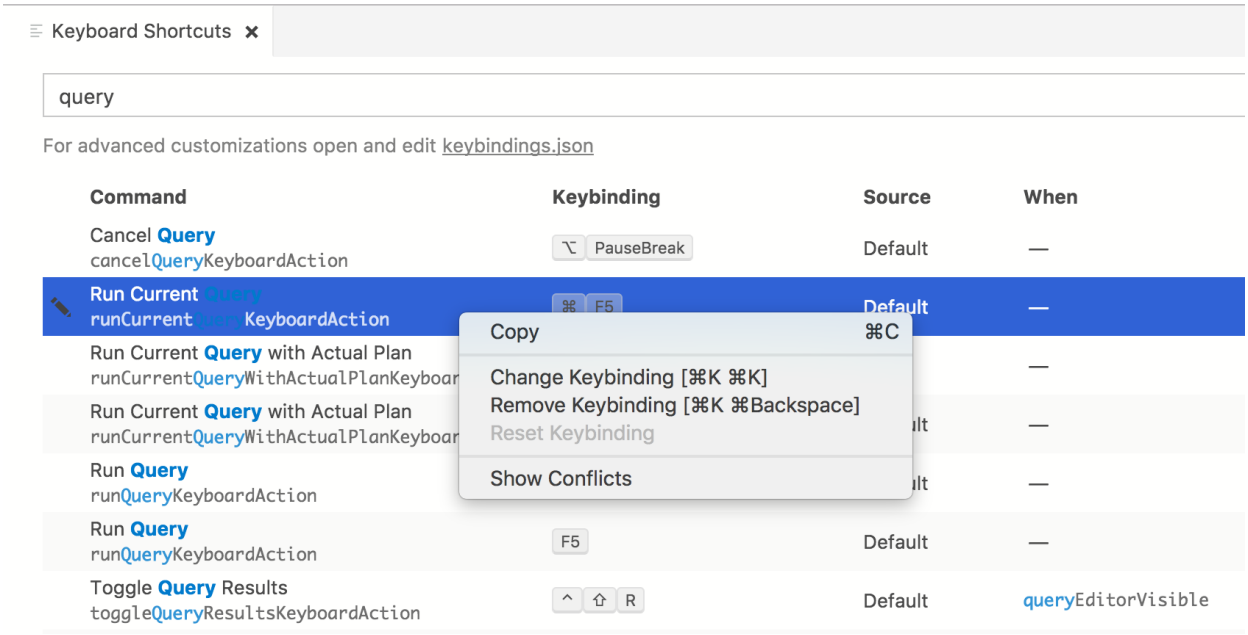
Step 1: Find the shortcuts to replace

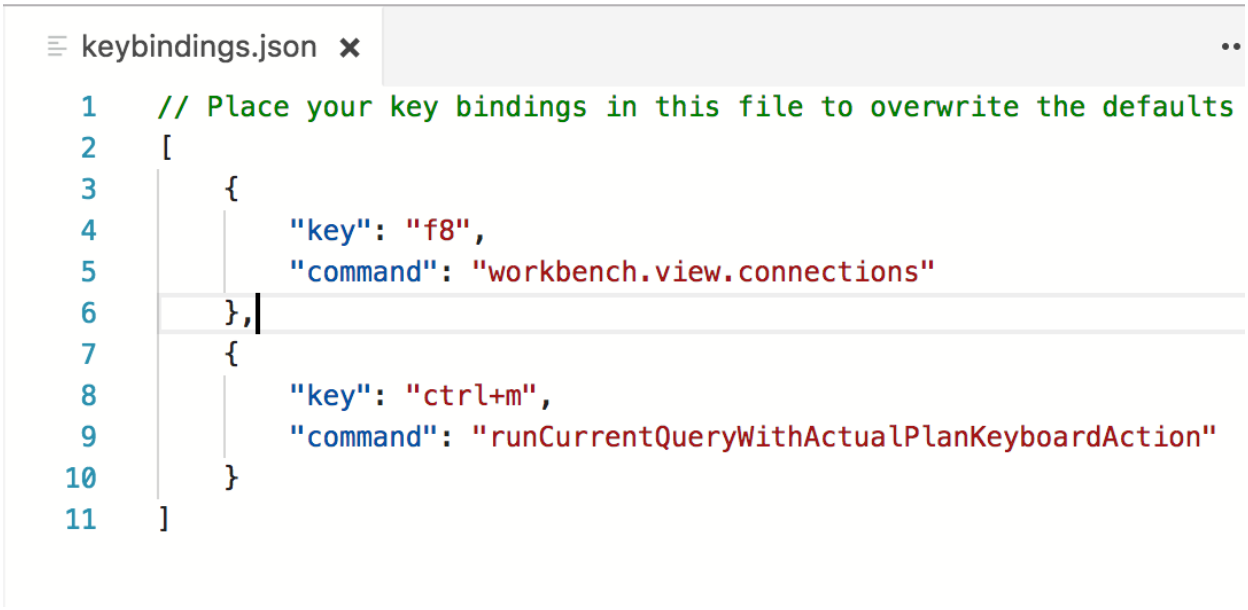
Now that we have our extension ready to go, add some SSMS keyboard shortcuts (or keybindings) into Azure Data Studio. I used Andy Mallon's Cheatsheet and RedGate's keyboard shortcuts list for inspiration.

The top things I saw missing were:

- Run a query with the actual execution plan enabled. This is **Ctrl+M** in SSMS and doesn't have a binding in Azure Data Studio.
- Having **CTRL+SHIFT+E** as a second way of running a query. User feedback indicated that this was missing.
- Having **ALT+F1** run `sp_help`. We added this in Azure Data Studio but since that binding was already in use, we mapped it to **ALT+F2** instead.
- Toggle full screen (**SHIFT+ALT+ENTER**).
- **F8** to show **Object Explorer / Servers view**.

It's easy to find and replace thesekey bindings. Run *Open Keyboard Shortcuts* to show the **Keyboard Shortcuts** tab in Azure Data Studio, search for *query* and then choose **Change Key binding**. Once you're done changing the keybinding, you can see the updated mapping in the keybindings.json file (run *Open Keyboard Shortcuts* to see it).





Step 2: Add shortcuts to the extension

To add shortcuts to the extension, open the *package.json* file (in the extension) and replace the `contributes` section with the code below:

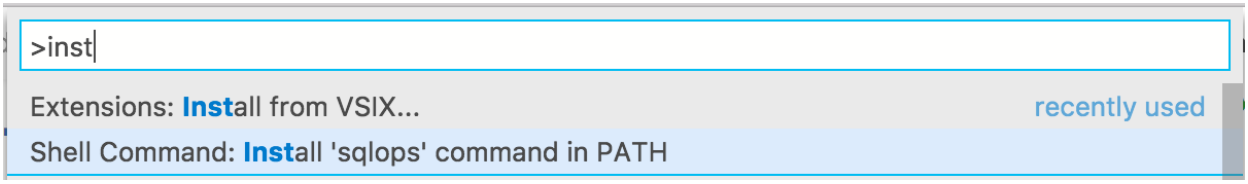
```
"contributes": {
  "keybindings": [
    {
      "key": "shift+cmd+e",
      "command": "runQueryKeyboardAction"
    },
    {
      "key": "ctrl+cmd+e",
      "command": "workbench.view.explorer"
    },
    {
      "key": "alt+f1",
      "command": "workbench.action.query.shortcut1"
    },
    {
      "key": "shift+alt+enter",
      "command": "workbench.action.toggleFullScreen"
    },
    {
      "key": "f8",
      "command": "workbench.view.connections"
    },
    {
      "key": "ctrl+m",
      "command": "runCurrentQueryWithActualPlanKeyboardAction"
    }
  ]
}
```

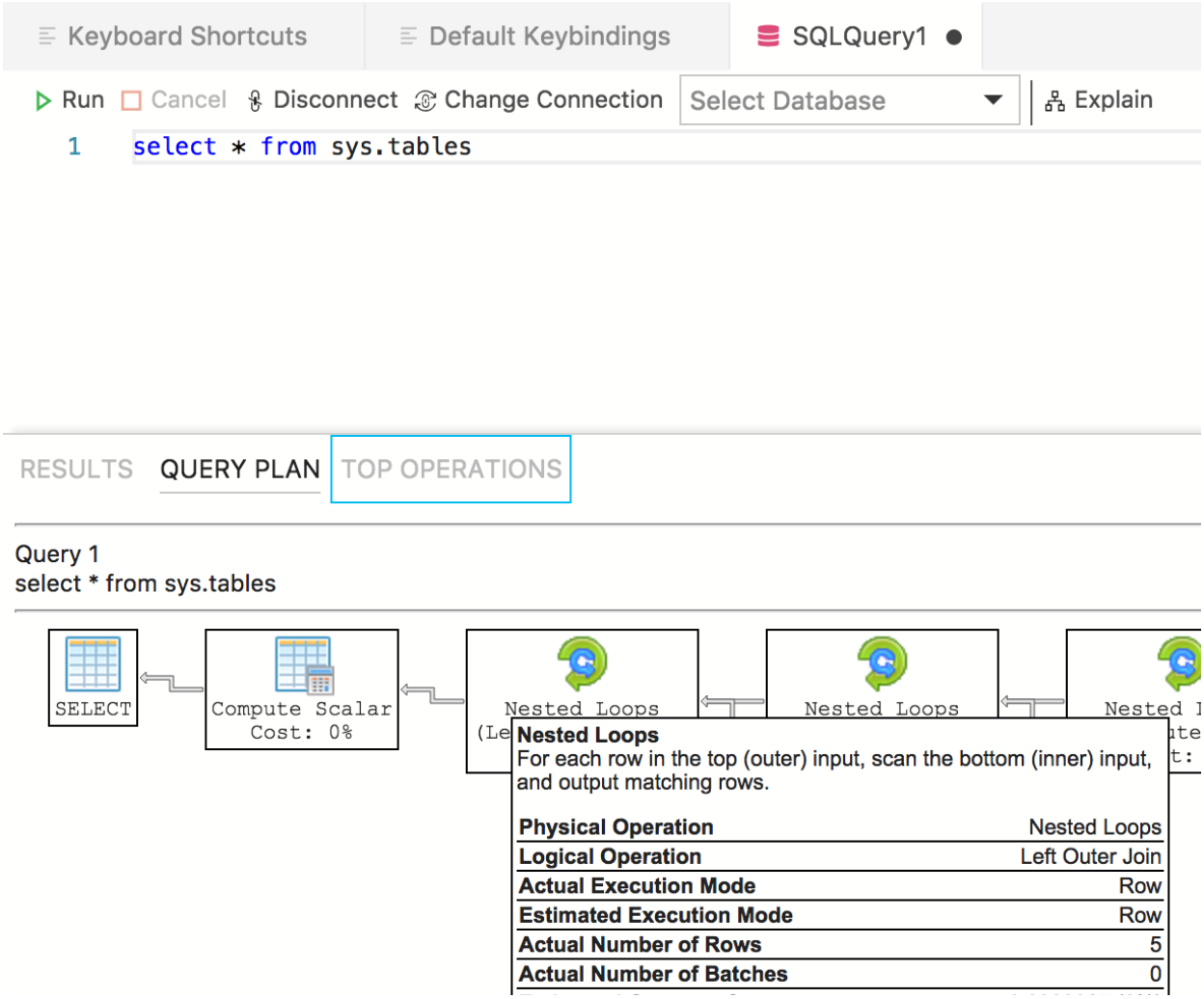
Test your extension

Ensure `azuredatastudio` is in your PATH by running the Install azuredatastudio command in PATH command in Azure Data Studio.

Ensure the Azure Data Studio Debug extension is installed in Visual Studio Code.

Select **F5** to launch Azure Data Studio in debug mode with the extension running:





Key maps are one of the quickest extensions to create, so your new extension should now be successfully working and ready to share.

Package your extension

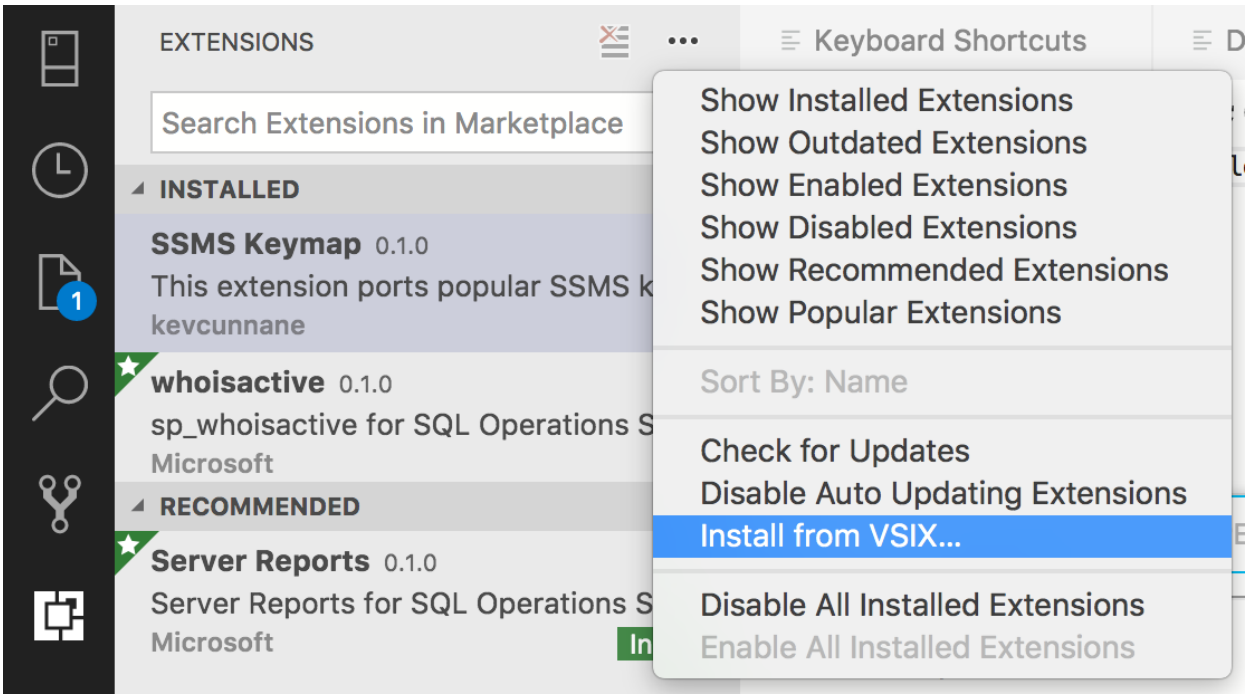
To share with others, you need to package the extension into a single file. This can be published to the Azure Data Studio extension Marketplace, or shared among your team or community. To do this, you need to install another npm package from the command line:

```
npm install -g vsce
```

Navigate to the base directory of the extension, and run `vsce package`. I had to add in a couple of extra lines to stop the *vsce* tool from complaining:

```
"repository": {
    "type": "git",
    "url": "https://github.com/kevcunnane/ssmskeymap.git"
},
"bugs": {
    "url": "https://github.com/kevcunnane/ssmskeymap/issues"
},
```

Once this was done, my ssmskeymap-0.1.0.vsix file was created and ready to install and share with the world!



Publish your extension to the Marketplace

The Azure Data Studio extension Marketplace is under construction, but the current process is to host the extension VSIX somewhere (for example, a GitHub Release page) then submit a PR updating this JSON file with your extension info.

Next steps

In this tutorial, you learned how to:

- Create an extension project
- Install the extension generator
- Create your extension
- Add custom key-bindings to your extension
- Test your extension
- Package your extension
- Publish your extension to the marketplace

We hope after reading this you'll be inspired to build your own extension for Azure Data Studio. We have support for Dashboard Insights (pretty graphs that run against your SQL Server), a number of SQL-specific APIs, and a huge existing set of extension points inherited from Visual Studio Code.

If you have an idea but are not sure how to get started, open an issue or tweet at the team: azuredatastudio.

You can always refer to the Visual Studio Code extension guide because it covers all the existing APIs and patterns.

To learn how to work with T-SQL in Azure Data Studio, complete the T-SQL Editor tutorial:

Use the Transact-SQL editor to create database objects

Create an Azure Data Studio dashboard extension

3/5/2021 • 4 minutes to read • Edit Online

This tutorial demonstrates how to create a new Azure Data Studio dashboard extension. The extension contributes to the Azure Data Studio connection dashboard, so you can extend the functionality of Azure Data Studio in a manner easily visible to users.

In this article you learn how to:

- Install the extension generator.
- Create your extension.
- Contribute to the dashboard in your extension.
- Test your extension.
- Package your extension.
- Publish your extension to the marketplace.

Prerequisites

Azure Data Studio is built on the same framework as Visual Studio Code, so extensions for Azure Data Studio are built by using Visual Studio Code. To get started, you need the following components:

- Node.js installed and available in your `$PATH`. Node.js includes npm, the Node.js Package Manager, which is used to install the extension generator.
- Visual Studio Code to debug the extension.
- The Azure Data Studio Debug extension (optional). The Debug extension lets you test your extension without needing to package and install it in Azure Data Studio.
- Ensure `azuredatastudio` is in your path. For Windows, make sure you choose the **Add to Path** option in setup.exe. For Mac or Linux, run **Install 'azuredatastudio' command in PATH** from the Command Palette in Azure Data Studio.

Install the extension generator

To simplify the process of creating extensions, we've built an extension generator using Yeoman. To install it, run the following command from the command prompt:

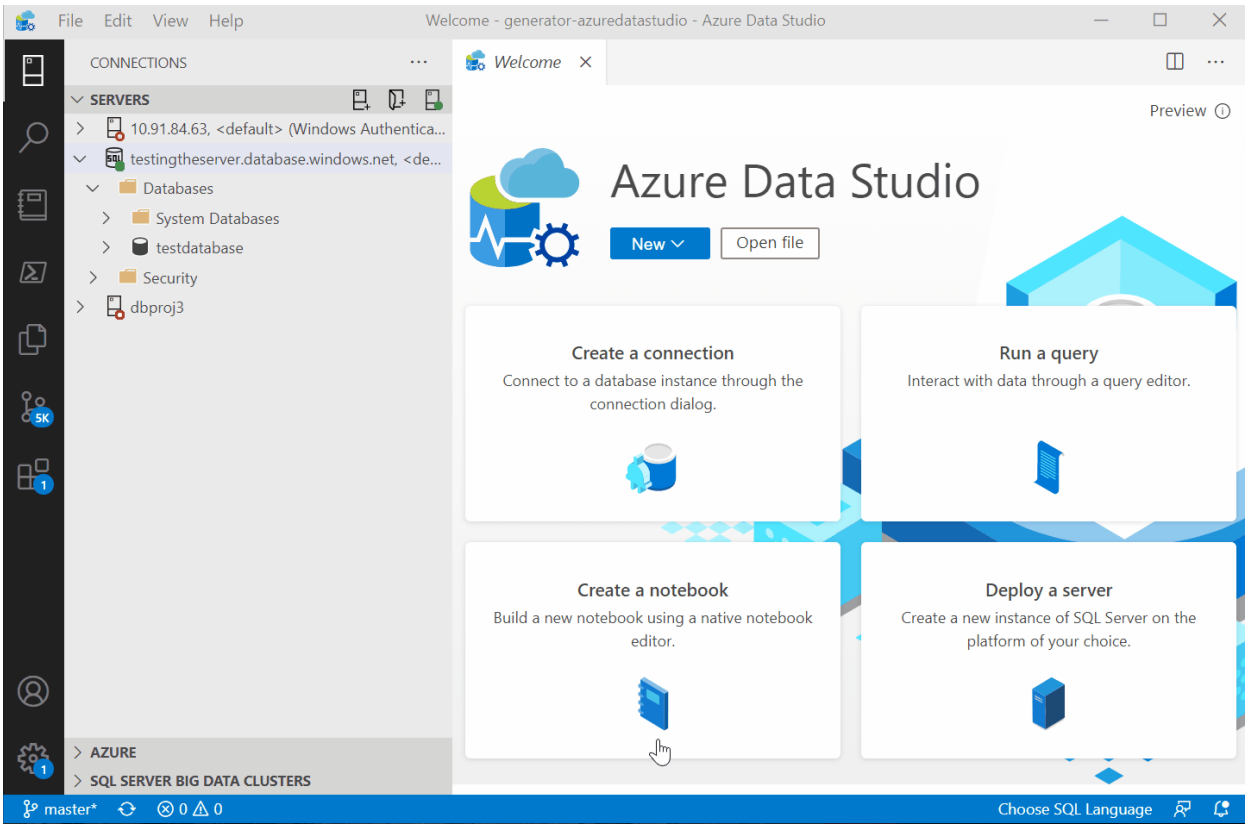
```
npm install -g yo generator-azuredatastudio
```

Create your dashboard extension

Introduction to the dashboard

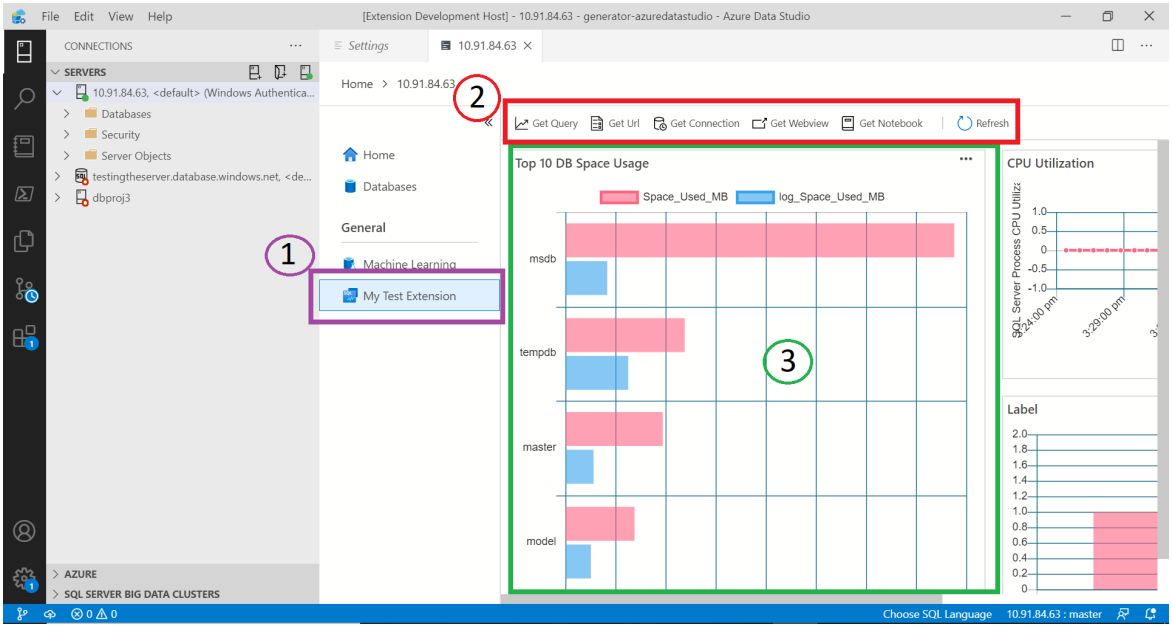
The Azure Data Studio connection dashboard is a powerful tool that summarizes and provides insight into a user's connections.

There are two variations of the dashboard. The *server dashboard* summarizes the entire server, and the *database dashboard* summarizes an individual database. You can access either dashboard by right-clicking a server or a database in the **Connections** viewlet of Azure Data Studio, and selecting **Manage**.



There are three key contribution points for extensions to add functionality to the dashboard:

1. **Full Dashboard tab**: A separate tab in the dashboard for your extension. Can be added to either a server or database dashboard. Customizable with widgets, a toolbar, and a navigation section.
2. **Homepage Actions**: Action buttons at the top of the connection toolbar.
3. **Widgets**: Graphs that run against your SQL Server.



Run the extension generator

To create an extension:

1. Start the extension generator with the following command:

   ```
   yo azuredatastudio
   ```

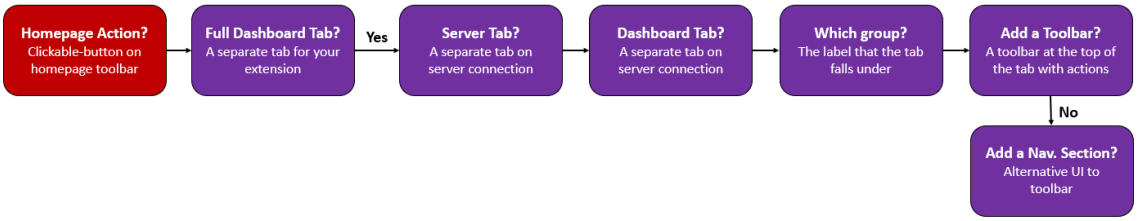
2. Choose **New Dashboard** from the list of extension types.

3. Fill in the prompts, as shown, to create an extension that contributes a tab to the server dashboard.

```
     _-----_
    |       |     ╭──────────────╮
    |--(o)--|     │   Welcome to the Azure   │
   `---------´    │   Data Studio Extension  │
    ( _´U`_ )     │        Generator!        │
    /___A___\   /  ╰──────────────╯
     |  ~  |
   __'.___.'__
 ´   `  |° ´ Y `

? What type of extension do you want to create? New Dashboard
? Add a homepage action? No
? Add a full dashboard tab? Yes
? Add the dashboard tab on server? Yes
? Add the dashboard tab on database? No
? Which group to place the tab? General
? Add a dashboard toolbar? Yes
? What's the display name of your extension? My Test Extension
? What's the publisher name for your extension? Microsoft
? What's the unique identifier for your extension? (Appears as Microsoft.your-identifier) my-test-extension
? What's the description of your extension? an extension to test the dashboard
```

There are many prompts, so here's a little more information on what each question means:



Completing the previous steps creates a new folder. Open the folder in Visual Studio Code, and you're ready to create your own dashboard extension.

Run the extension

Let's see what the dashboard template gives us by running the extension. Before you run it, ensure that the **Azure Data Studio Debug extension** is installed in Visual Studio Code.

Select **F5** in Visual Studio Code to launch Azure Data Studio in Debug mode with the extension running. Then, you can see how this default template contributes to the dashboard.

Next, we look at how to modify this default dashboard.

Develop the dashboard

The most important file to get started with extension development is `package.json`. This file is the manifest file, where the dashboard contributions are registered. Note the `dashboard.tabs`, `dashboard.insights`, and `dashboard.containers` sections.

Here are some changes to try out:

- Play around with the insights types, which include bar, horizontalBar, and timeSeries.
- Write your own queries to run against your SQL Server connection.
- See this sample insight tutorial or this tutorial for specific insight tutorials.

Package your extension

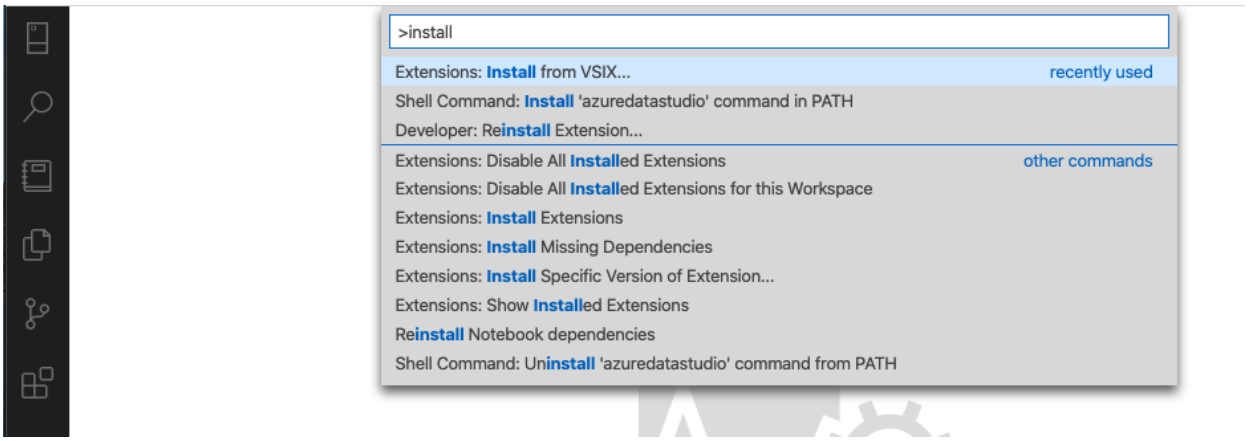
To share with others, you need to package the extension into a single file. Your extension can be published to the Azure Data Studio extension marketplace or shared with your team or community. To do this step, you need to install another npm package from the command line.

```
npm install -g vsce
```

Edit the `README.md` file to your liking. Then go to the base directory of the extension, and run `vsce package`. You can optionally link a repository with your extension or continue without one. To add one, add a similar line to your `package.json` file.

```
"repository": {
    "type": "git",
    "url": "https://github.com/anjalia/my-test-extension.git"
}
```

After these lines are added, a `my-test-extension-0.0.1.vsix` file is created and ready to install in Azure Data Studio.



Publish your extension to the marketplace

The Azure Data Studio extension marketplace is under construction. The current process is to host the extension VSIX somewhere, for example, on a GitHub release page. Then you submit a pull request that updates this JSON file with your extension information.

Next steps

In this tutorial, you learned how to:

- Install the extension generator.
- Create your extension.
- Contribute to the dashboard in your extension.
- Test your extension.
- Package your extension.
- Publish your extension to the marketplace.

We hope that after reading this article you're inspired to build your own extension for Azure Data Studio. We have support for Dashboard Insights (attractive graphs that run against your SQL Server), a number of SQL-specific APIs, and a huge existing set of extension points inherited from Visual Studio Code.

If you have an idea but aren't sure how to get started, open an issue or tweet the team at azuredatastudio.

For more information, the Visual Studio Code extension guide covers all the existing APIs and patterns.

To learn how to work with T-SQL in Azure Data Studio, complete the T-SQL Editor tutorial:

Use the Transact-SQL editor to create database objects

Create a Jupyter Notebook extension

3/5/2021 • 5 minutes to read • Edit Online

This tutorial demonstrates how to create a new Jupyter Notebook Azure Data Studio extension. The extension ships a sample Jupyter Notebook that can be opened and run in Azure Data Studio.

In this article you learn how to:

- Create an extension project.
- Install the extension generator.
- Create your notebook extension.
- Run your extension.
- Package your extension.
- Publish your extension to the marketplace.

APIs used

- `azdata.nb.showNotebookDocument`

Extension use cases

There are a few different reasons why you would create a notebook extension:

- Share interactive documentation
- Save and have constant access to that notebook
- Provide coding problems for users to follow along with
- Version and keep track of notebook updates

Prerequisites

Azure Data Studio is built on the same framework as Visual Studio Code, so extensions for Azure Data Studio are built by using Visual Studio Code. To get started, you need the following components:

- Node.js installed and available in your `$PATH`. Node.js includes npm, the Node.js Package Manager, which is used to install the extension generator.
- Visual Studio Code to debug the extension.
- Ensure `azuredatastudio` is in your path. For Windows, make sure you choose the **Add to Path** option in setup.exe. For Mac or Linux, run **Install 'azuredatastudio' command in PATH** from the Command Palette in Azure Data Studio.

Install the extension generator

To simplify the process of creating extensions, we've built an extension generator using Yeoman. To install it, run the following command from the command prompt:

```
npm install -g yo generator-azuredatastudio
```

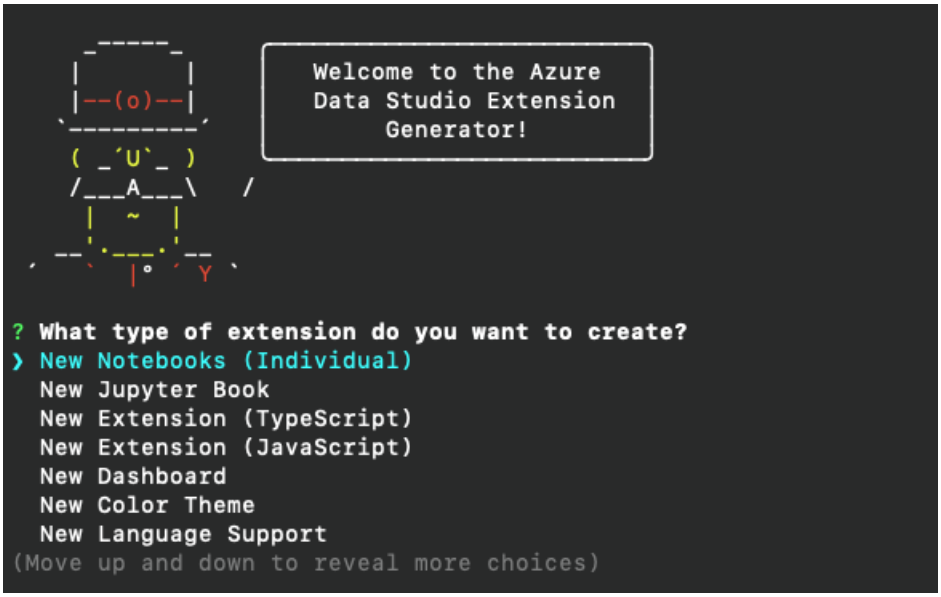
Create your extension

To create an extension:

1. Start the extension generator with the following command:

   ```
   yo azuredatastudio
   ```

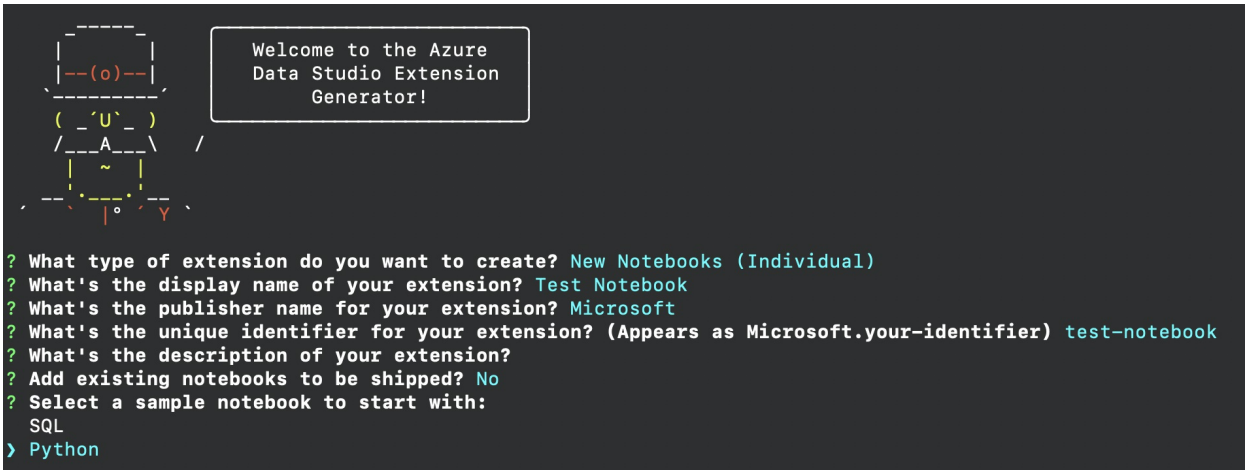
2. Choose **New Notebooks (Individual)** from the list of extension types.



3. Follow the steps to fill in the extension name. For this tutorial, use **Test Notebook**. Then fill in a publisher name. For this tutorial, use **Microsoft**. Finally, add a description.

Now, this is where some branching exists. You can either add Jupyter Notebooks that you've already created or you can use sample notebooks provided to you through the generator.

For this tutorial, we'll use a sample Python notebook:

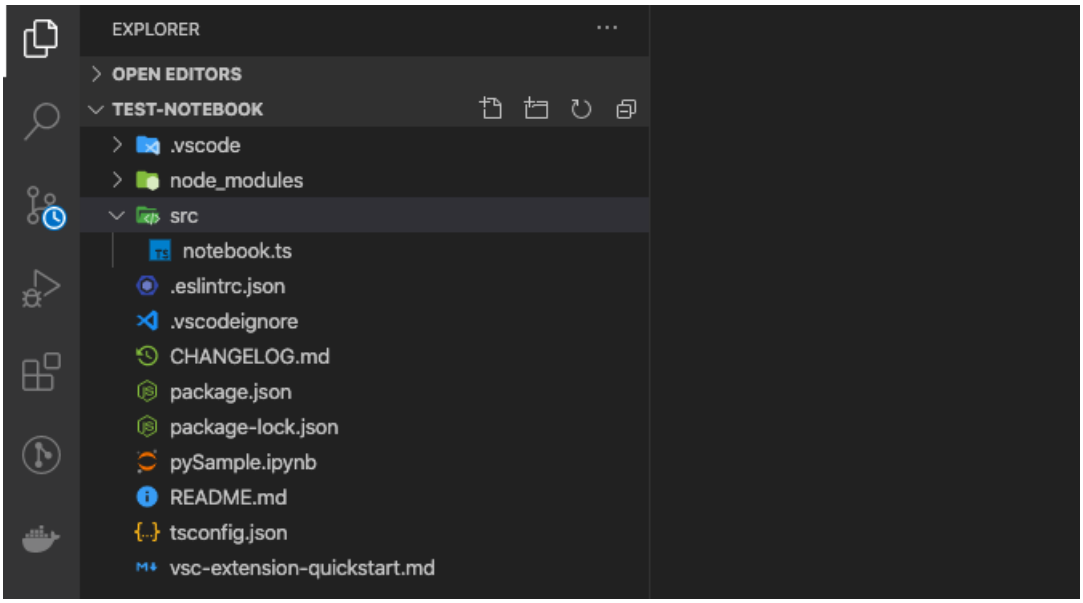


If you have notebooks that you're interested in shipping, answer that you have existing notebooks you want to ship. Provide the absolute file path where all of your notebooks or markdown files live.

Completing the previous steps creates a new folder with the sample notebook. Open the folder in Visual Studio Code, and you're ready to ship your new notebook extension.

Understand your extension

This is what your project should currently look like:



The `vsc-extension-quickstart.md` file provides you with a reference of the important files. The `README.md` file is where you can provide documentation for your new extension. Note the `package.json`, `notebook.ts`, and `pySample.ipynb` files.

If there are any files or folders that you don't want to publish, you can include their names in the `.vscodeignore` file.

Let's take a look at `notebook.ts` to understand what our newly formed extension is doing.

```
// This function is called when you run the command `Launch Notebooks: Test Notebook` from the
// command palette in Azure Data Studio. If you want any additional functionality
// to occur when you launch the book, add it to the activate function.
export function activate(context: vscode.ExtensionContext) {
    context.subscriptions.push(vscode.commands.registerCommand('launchNotebooks.test-notebook', () => {
        let notebooksToDisplay: Array<string> = processNotebooks();
        notebooksToDisplay.forEach(name => {
            azdata.nb.showNotebookDocument(vscode.Uri.file(name));
        });
    }));

    // Add other code here if you want to register another command.
}
```

This is the main function in `notebook.ts` that's called whenever we run our extension through the command **Launch Notebooks: Test Notebook**. We create our new command by using the `vscode.commands.registerCommand` API. The following definition inside the braces is the code that runs each time we call our command. For each notebook that's found from our `processNotebooks` function, we open it up in Azure Data Studio by using `azdata.nb.showNotebookDocument`.

The `package.json` file also plays an important role in registering our command **Launch Notebooks: Test Notebook**.

```
"activationEvents": [
  "onCommand:launchNotebooks.test-notebook"
 ],
 "main": "./out/notebook.js",
 "contributes": {
  "commands": [
   {
    "command": "launchNotebooks.test-notebook",
    "title": "Launch Notebooks: Test Notebook"
   }
  ]
 }
```

We have an activation event for the command, and we've also added specific contribution points. These contribution points show up in the extension marketplace, where extensions are published, when users are looking at your extension. If you want to add more commands, be sure to add them to the `activationEvents` field. For more options, see Activation events.

Package your extension

To share with others, you need to package the extension into a single file. Your extension can be published to the Azure Data Studio extension marketplace or shared with your team or community. To do this step, you need to install another npm package from the command line.

```
npm install -g vsce
```

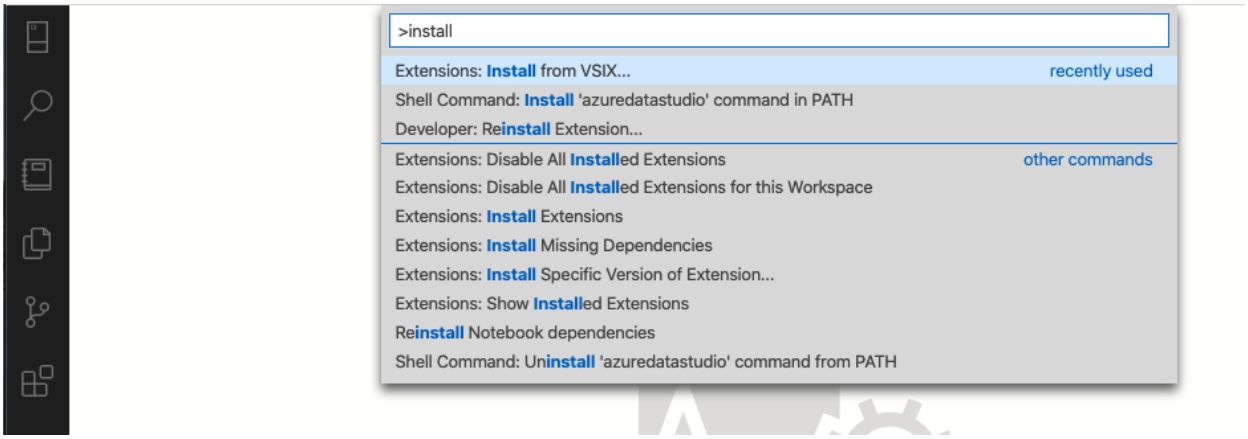
Edit the `README.md` file to your liking. Then go to the base directory of the extension, and run `vsce package`. You can optionally link a repository with your extension or continue without one. To add one, add a similar line to your `package.json` file.

```
"repository": {
    "type": "git",
    "url": "https://github.com/laurajjiang/testnotebook.git"
}
```

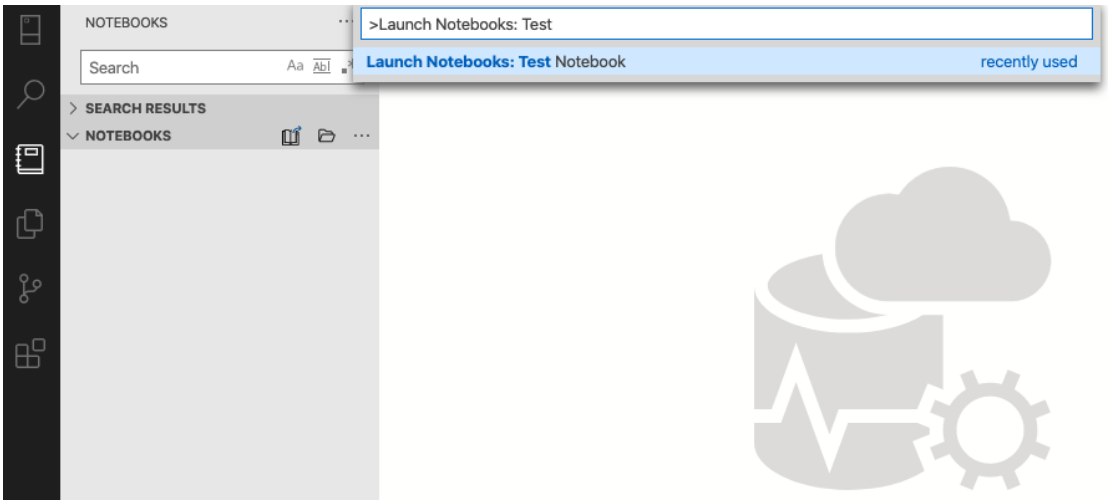
After these lines are added, a `my test-notebook-0.0.1.vsix` file is created and ready to install and share with the world.

Run your extension

To run and test your extension, open Azure Data Studio and open the command palette by selecting **Ctrl+Shift+P**. Find the command **Extensions: Install from VSIX**, and go to the folder that contains your new extension.



Your extension should now show up in your extension panel in Azure Data Studio. Open the command palette again, and you'll find the new command that we created with our extension **Launch Book: Test Book**. Upon running, it should open the Jupyter Book that we packaged with our extension.



Congratulations! You built and can now ship your first Jupyter Notebook extension.

Publish your extension to the marketplace

The Azure Data Studio extension marketplace is under construction. To publish, host the extension VSIX somewhere, for example, on a GitHub release page. Then submit a pull request that updates this JSON file with your extension information.

Next steps

In this tutorial, you learned how to:

- Create an extension project.
- Install the extension generator.
- Create your notebook extension.
- Create your extension.
- Package your extension.
- Publish your extension to the marketplace.

We hope that after reading this article you'll be inspired to build your own extension for Azure Data Studio.

If you have an idea but aren't sure how to get started, open an issue or tweet the team at azuredatastudio.

For more information, the Visual Studio Code extension guide covers all the existing APIs and patterns.

Create a Jupyter Book extension

3/5/2021 • 6 minutes to read • Edit Online

This tutorial demonstrates how to create a new Jupyter Book Azure Data Studio extension. The extension ships a sample Jupyter Book that can be opened and run in Azure Data Studio.

In this article you learn how to:

- Create an extension project.
- Install the extension generator.
- Create your Jupyter Book extension.
- Run your extension.
- Package your extension.
- Publish your extension to the marketplace.

APIs used

- `bookTreeView.openBook`

Extension use cases

There are a few different reasons why you would create a Jupyter Book extension:

- Share organized and sectioned interactive documentation
- Share a full book (similar to an e-book but distributed through Azure Data Studio)
- Version and keep track of Jupyter Book updates

The main difference between a Jupyter Book and a notebook extension is that a Jupyter Book provides you with organization. Tens of notebooks can be split into different chapters in a Jupyter Book, but the notebook extension is intended to ship a small number of individual notebooks.

Prerequisites

Azure Data Studio is built on the same framework as Visual Studio Code, so extensions for Azure Data Studio are built by using Visual Studio Code. To get started, you need the following components:

- Node.js installed and available in your `$PATH`. Node.js includes npm, the Node.js Package Manager, which is used to install the extension generator.
- Visual Studio Code to make any changes to your extension and debug the extension.
- Ensure `azuredatastudio` is in your path. For Windows, make sure you choose the **Add to Path** option in setup.exe. For Mac or Linux, run **Install 'azuredatastudio' command in PATH** from the Command Palette in Azure Data Studio.

Install the extension generator

To simplify the process of creating extensions, we've built an extension generator using Yeoman. To install it, run the following command from the command prompt:

```
npm install -g yo generator-azuredatastudio
```

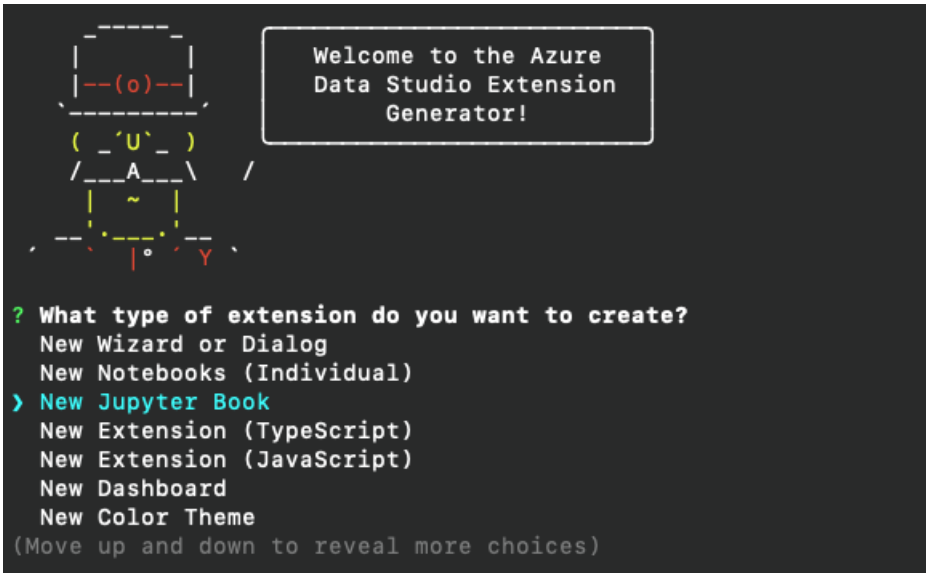
Create your extension

To create an extension:

1. Start the extension generator with the following command:

   ```
   yo azuredatastudio
   ```

2. Choose **New Jupyter Book** from the list of extension types.

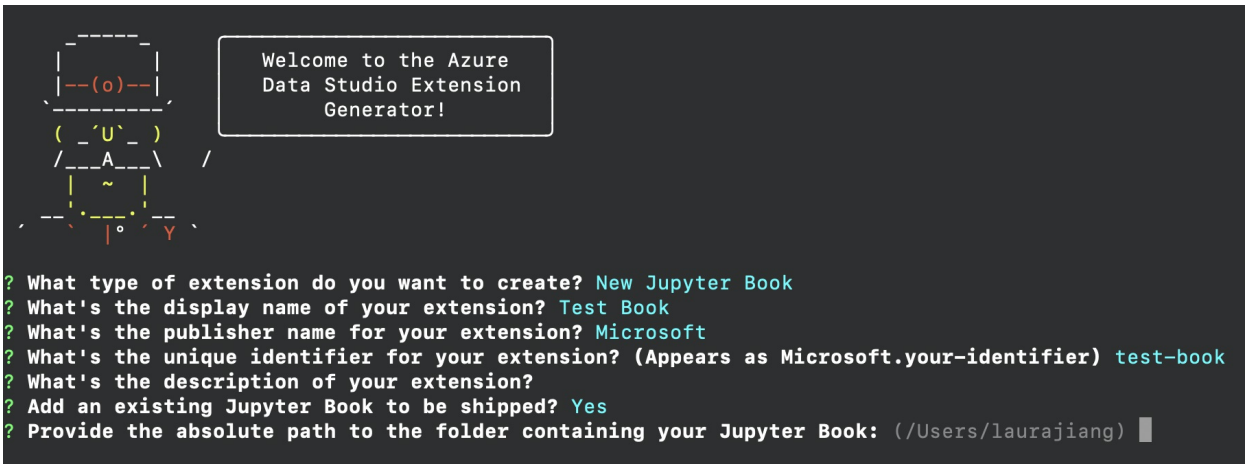


3. Follow the steps to fill in the extension name. For this tutorial, use **Test Book**. Then fill in a publisher name. For this tutorial, use **Microsoft**. Finally, add a description.

You select to either provide an existing Jupyter Book, use a provided sample book, or work to create a new Jupyter Book. All three options are shown.

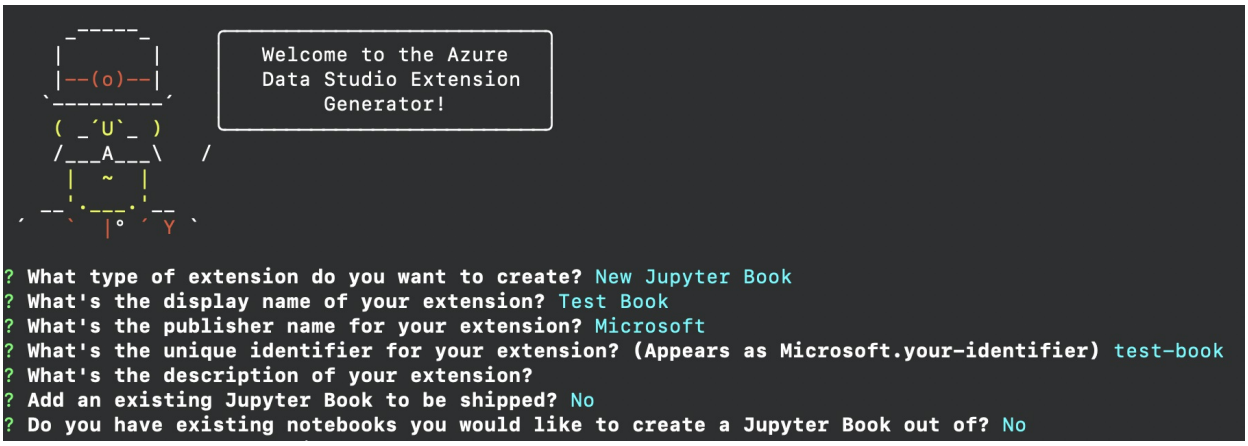
Provide an existing book

If you want to ship a book that you've already created, provide the absolute file path to the folder where your book contents live. You can then be ready to move on to learning about the extension and shipping it.



Use the sample book

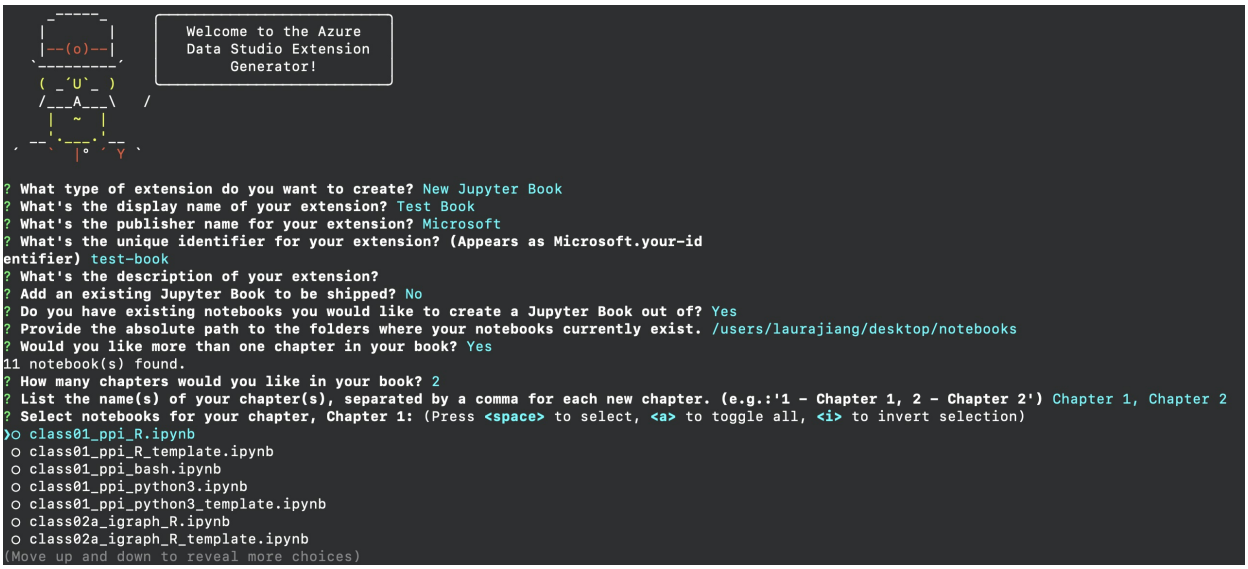
If you don't have an existing book or notebooks, you can use the provided sample in the generator.



The sample book demonstrates what a simple Jupyter Book looks like. If you want to learn about customizing a Jupyter Book, see the following section on how to create a new book with existing notebooks.

Create a new book

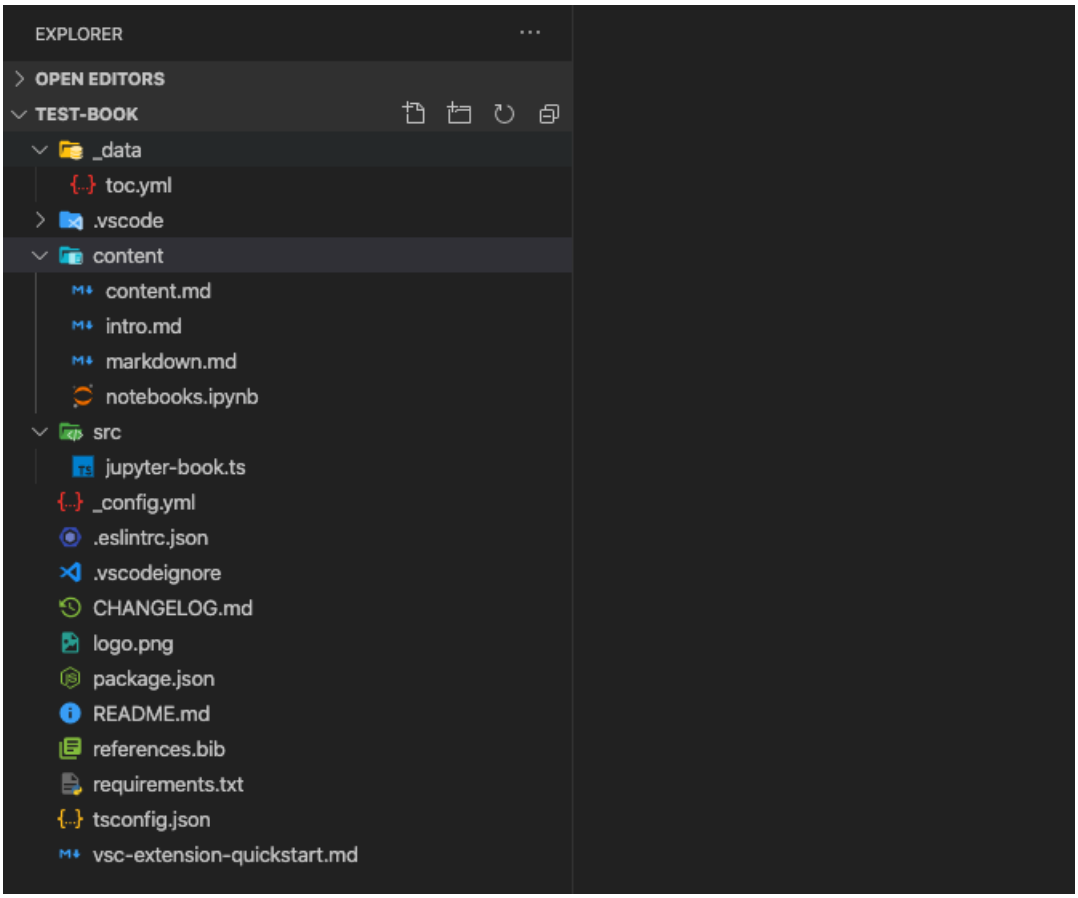
If you have notebooks that you want to package into a Jupyter Book, you can. The generator asks if you want chapters in your book, and if so, how many and their titles. You can see what the selection process looks like here. Use the Spacebar to select which notebooks you want to place in each chapter.



Completing the previous steps creates a new folder with your new Jupyter Book. Open the folder in Visual Studio Code, and you're ready to ship your Jupyter Book extension.

Understand your extension

This is what your project should currently look like:



The `vsc-extension-quickstart.md` file provides you with a reference of the important files. The `README.md` file is where you can provide documentation for your new extension. Note the `package.json`, `jupyter-book.ts`, `content`, and `toc.yml` files. The `content` folder holds all notebook or markdown files. The `toc.yml` structures your Jupyter Book and is autogenerated if you opted to create a custom Jupyter Book through the extension generator.

If you created a book by using the generator and opted for chapters in your book, your folder structure would look a little different. Instead of your markdown and Jupyter Notebook files living in the `content` folder, there would be subfolders that correspond to the titles that you chose for your chapters.

If there are any files or folders that you don't want to publish, you can include their names in the `.vscodeignore` file.

Let's take a look at `jupyter-book.ts` to understand what our newly formed extension is doing.

```
// This function is called when you run the command `Launch Book: Test Book` from the
// command palette in Azure Data Studio. If you want any additional functionality
// to occur when you launch the book, add it to the activate function.
export function activate(context: vscode.ExtensionContext) {
    context.subscriptions.push(vscode.commands.registerCommand('launchBook.test-book', () => {
        processNotebooks();
    }));

    // Add other code here if you want to register another command.
}
```

The `activate` function is the main action of your extension. Any commands that you want to register should appear inside the `activate` function, similar to our `launchBook.test-book` command. Inside the `processNotebooks` function, we find our extension folder that holds our Jupyter Book and call `bookTreeView.openBook` by using our extension's folder as a parameter.

The `package.json` file also plays an important role in registering our extension's command.

```
"activationEvents": [
  "onCommand:launchBook.test-book"
 ],
 "main": "./out/notebook.js",
 "contributes": {
  "commands": [
   {
    "command": "launchBook.test-book",
    "title": "Launch Book: Test Book"
   }
  ]
 }
```

The activation event, `onCommand`, triggers the function that we registered when we invoke the command. There are a few other activation events that are possible for additional customization. For more information, see Activation events.

Package your extension

To share with others, you need to package the extension into a single file. Your extension can be published to the Azure Data Studio extension marketplace or shared with your team or community. To do this step, you need to install another npm package from the command line.

```
npm install -g vsce
```

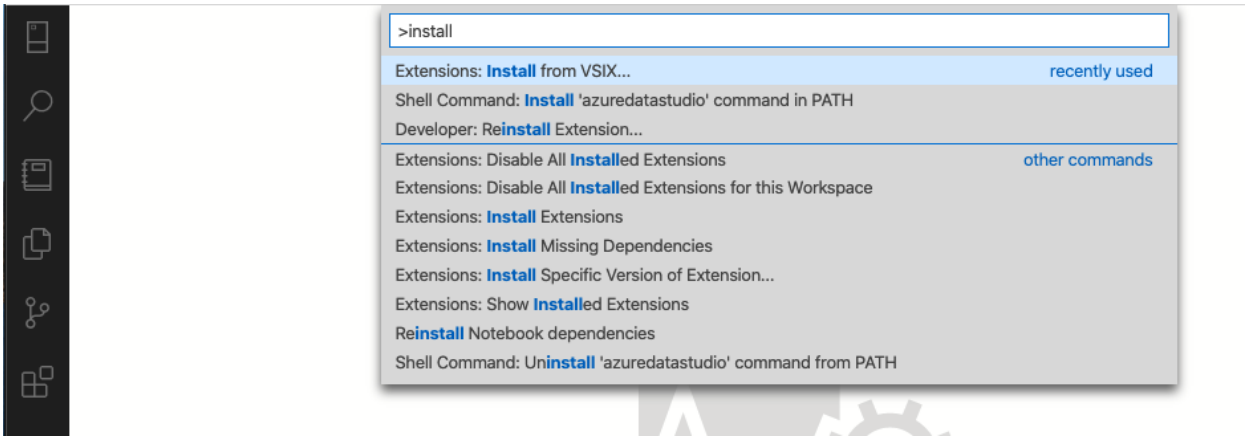
Edit the `README.md` file to your liking. Then go to the base directory of the extension, and run `vsce package`. You can optionally link a repository with your extension or continue without one. To add one, add a similar line to your `package.json` file.

```
"repository": {
    "type": "git",
    "url": "https://github.com/laurajjiang/testbook.git"
}
```

After these lines are added, a `my test-book-0.0.1.vsix` file is created and ready to install in Azure Data Studio.

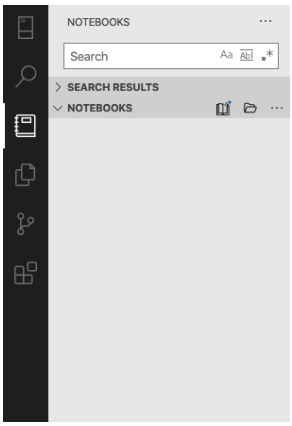
Run your extension

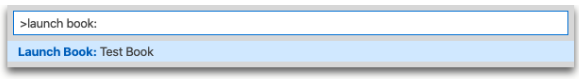
To run and test your extension, open Azure Data Studio and open the command palette by selecting **Ctrl+Shift+P**. Find the command **Extensions: Install from VSIX**, and go to the folder that contains your new extension. It should now show up in your extension panel in Azure Data Studio.



Open the command palette again, and find the command that we registered, **Launch Book: Test Notebook**.

Upon running, it should open the Jupyter Book that we packaged with our extension.





Congratulations! You built and can now ship your first Jupyter Book extension. For more information on Jupyter Books, see Books with Jupyter.

Publish your extension to the Marketplace

The Azure Data Studio extension marketplace is under construction. To publish, host the extension VSIX somewhere, for example, on a GitHub release page. Submit a pull request that updates this JSON file with your extension information.

Next steps

In this tutorial, you learned how to:

- Create an extension project.
- Install the extension generator.
- Create your Jupyter Book extension.
- Package your extension.
- Publish your extension to the marketplace.

We hope that after reading this article you'll have ideas on Jupyter Books that you'd like to share with the Azure Data Studio community.

If you have an idea but aren't sure how to get started, open an issue or tweet the team at azuredatastudio.

For more information, the Visual Studio Code extension guide covers all the existing APIs and patterns.

Create an Azure Data Studio Wizard extension

3/5/2021 • 4 minutes to read • Edit Online

This tutorial demonstrates how to create a new **Azure Data Studio Wizard extension**. The extension contributes a wizard to interact with users in Azure Data Studio.

In this article you learn how to:

- Install the extension generator
- Create your extension
- Add a custom wizard to your extension
- Test your extension
- Package your extension
- Publish your extension to the marketplace

Prerequisites

Azure Data Studio is built on the same framework as Visual Studio Code, so extensions for Azure Data Studio are built using Visual Studio Code. To get started, you need the following components:

- Node.js installed and available in your `$PATH`. Node.js includes npm, the Node.js Package Manager, which is used to install the extension generator.
- Visual Studio Code to debug the extension.
- The Azure Data Studio Debug extension (optional). This lets you test your extension without needing to package and install it into Azure Data Studio.
- Ensure `azuredatastudio` is in your path. For Windows, make sure you choose the `Add to Path` option in setup.exe. For Mac or Linux, run **Install 'azuredatastudio' command in PATH** from the Command Palette in Azure Data Studio.

Install the extension generator

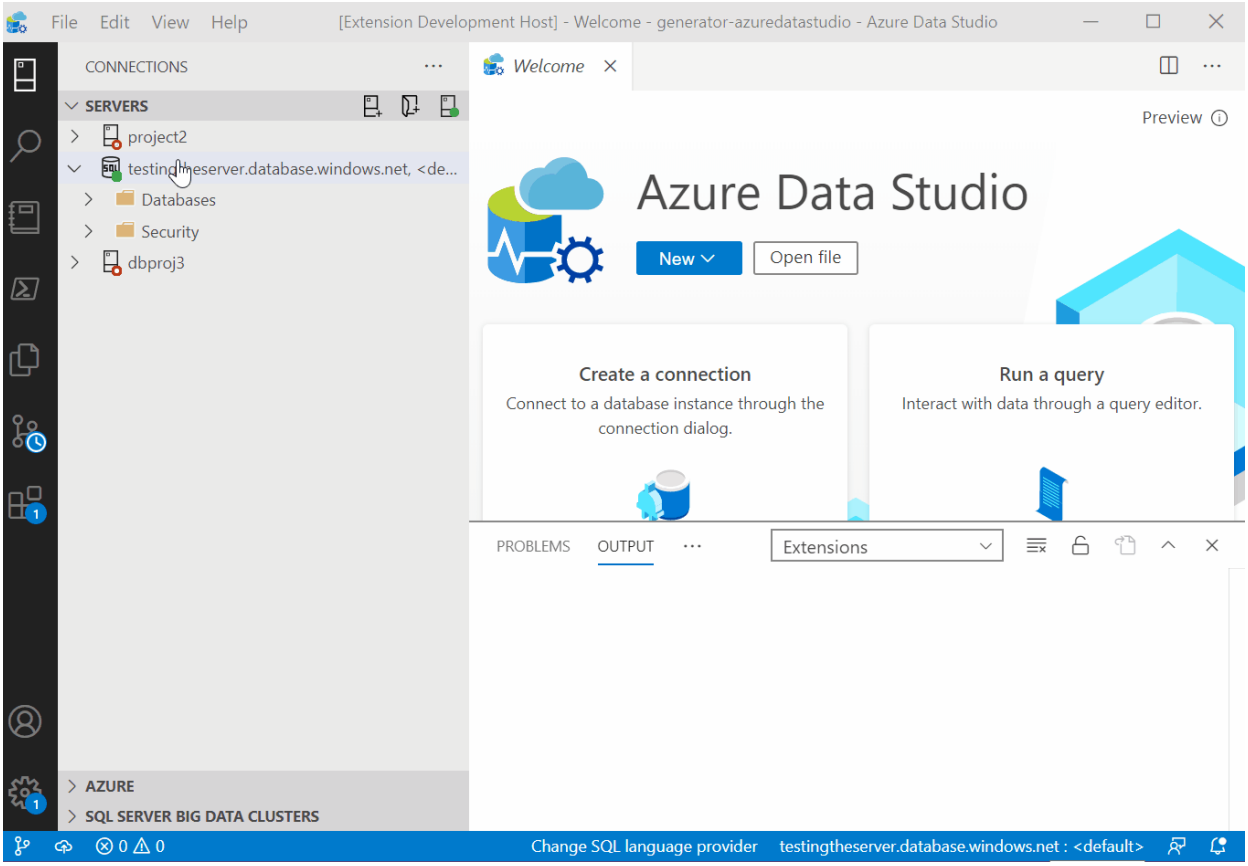
To simplify the process of creating extensions, we've built an extension generator using Yeoman. To install it, run the following from the command prompt:

```
npm install -g yo generator-azuredatastudio
```

Create your wizard extension

Introduction to wizards

Wizards are a user interface type that present step-by-step pages for users to fill in, in order to accomplish a task. Common examples include software setup wizards and troubleshooting wizards. For example:



Because wizards are often helpful when working with data and extending the functionality of Azure Data Studio, Azure Data Studio offers APIs to create your own custom wizards. We will be walking through how to generate a Wizard template and modify it to create your own custom wizard.

Run the extension generator

To create an extension:

1. Launch the extension generator with the following command:

   ```
   yo azuredatastudio
   ```

2. Choose **New Wizard or Dialog** from the list of extension types. Then select **Wizard**, followed by the **Getting Started Template**

3. Follow the steps to fill in the extension name (for this tutorial, use **My Test Extension**), and add a description.

```
     _-----_
    |       |     ╭──────────────────╮
    |--(o)--|     │  Welcome to the  │
   `---------´    │  Azure Data Studio │
    ( _´U`_ )     │  Extension Generator! │
    /___A___\   /  ╰──────────────────╯
     |  ~  |
   __'.___.'__
 ´   `  |° ´ Y `

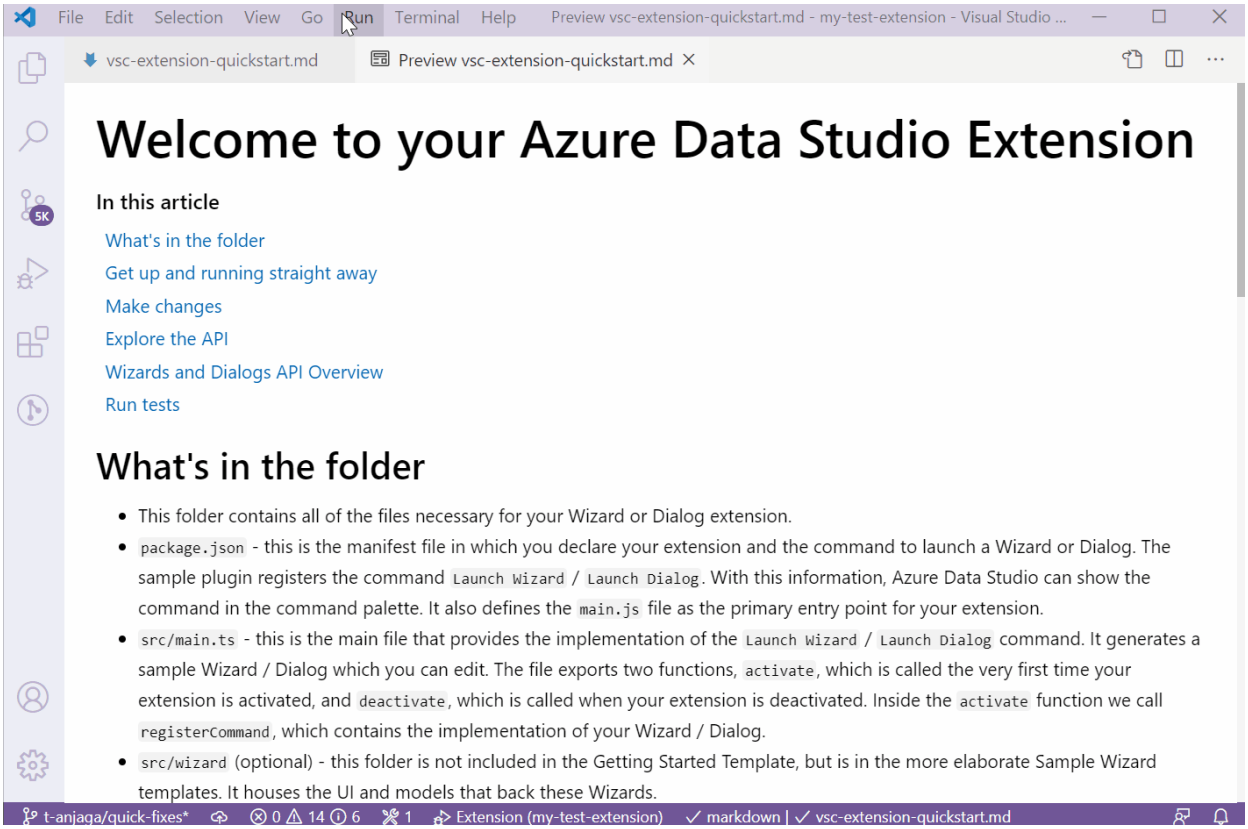
? What type of extension do you want to create? New Wizard or Dialog
? Do you want to create a Wizard or a Dialog Extension? Wizard
? Choose a Wizard Template: Getting Started Template
? What's the display name of your extension? My Test Extension
? What's the publisher name for your extension? Microsoft
? What's the unique identifier for your extension? (Appears as Microsoft.your-identifier) my-test-extension
? What's the description of your extension? my test extension!
? Initialize a git repository? (Y/n) n
```

Completing the previous steps creates a new folder. Open the folder in Visual Studio Code and you're ready to create your own wizard extension!

Run the extension

Let's see what the wizard template gives us by running the extension. Before running, ensure that the **Azure Data Studio Debug extension** is installed in Visual Studio Code.

Select **F5** in VS Code to launch Azure Data Studio in debug mode with the extension running. Then, in Azure Data Studio, run the **Launch Wizard** command from the Command Palette (Ctr+Shift+P) in the new window. This will launch the default wizard that this extension contributes:



Next, we will look at how to modify this default wizard.

Develop the wizard

The most important files to get started with extension development are `package.json`, `src/main.ts`, and `vsc-extension-quickstart.md`:

- `package.json`: This is the manifest file, where the **Launch Wizard** command is registered. This is also where `main.ts` is declared the main program entry point.
- `main.ts`: Contains the code to add UI elements to the Wizard, like pages, text, and buttons
- `vsc-extension-quickstart.md`: Contains technical documentation that may be a helpful reference when developing

Let's make a change to the wizard: we'll add a 4th, blank page. Modify `src/main.ts` as shown below, and you should see an additional page show up when you launch the wizard.

```
//---------------------------- Creating the Wizard ------------------------------------//

// The command has been defined in the package.json file
// This method defines what occurs when the command is entered in the ADS command palette
context.subscriptions.push(vscode.commands.registerCommand('my-test-extension.launchWizard', () => {
    // Create wizard
    let wizard : azdata.window.Wizard = azdata.window.createWizard('Sample Wizard');
    // Create wizard pages
    let page1 : azdata.window.WizardPage = azdata.window.createWizardPage('Sample Page 1');
    let page2 : azdata.window.WizardPage = azdata.window.createWizardPage('Sample Page 2');
    let page3 : azdata.window.WizardPage = azdata.window.createWizardPage('Sample Page 3');
    let page4 : azdata.window.WizardPage = azdata.window.createWizardPage('Sample Page 4');
    // Populate pages with content
    registerContentPage1(page1);
    registerContentPage2(page2);
    registerContentPage3(page3);

    page4.registerContent(async (view) => {
        // await view.initializeModel();
    });

    wizard.pages = [page1, page2, page3, page4]; // set wizard's content field to the wizard pages
    wizard.generateScriptButton.hidden = true;
    wizard.open(); // open wizard
}));
```

Once you are familiar with the template, here are some additional ideas to try:

- Add a button with a width of 300 to your new page
- Add a flex component to put your button in
- Add an action to your button. For example, when the button is clicked, launch a file-opening dialog or open a query editor.

Package your extension

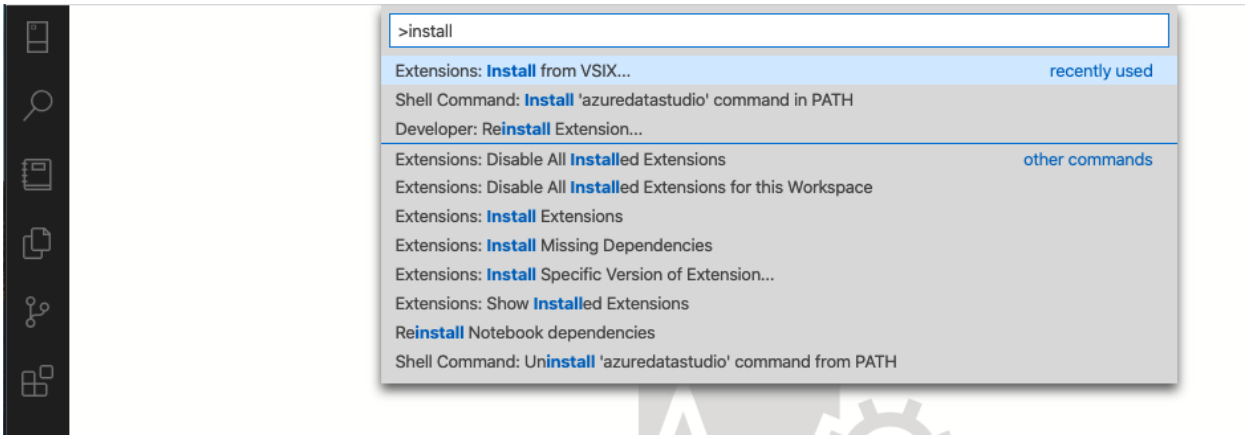
To share with others you need to package the extension into a single file. This can be published to the Azure Data Studio extension marketplace, or shared among your team or community. To do this, you need to install another npm package from the command line:

```
npm install -g vsce
```

Edit the `README.md` to your liking, then navigate to the base directory of the extension, and run `vsce package`. You can optionally link a repository with your extension or continue without one. To add one, add a similar line to your `package.json` file.

```
"repository": {
    "type": "git",
    "url": "https://github.com/anjalia/my-test-extension.git"
}
```

After these lines were added, a my-test-extension-0.0.1.vsix file was created and ready to install in Azure Data Studio.



Publish your extension to the marketplace

The Azure Data Studio extension marketplace is not totally implemented yet, but the current process is to host the extension VSIX somewhere (for example, a GitHub Release page) then submit a PR updating this JSON file with your extension info.

Next steps

In this tutorial, you learned how to:

- Install the extension generator
- Create your extension
- Add a custom wizard to your extension
- Test your extension
- Package your extension
- Publish your extension to the marketplace

We hope after reading this you'll be inspired to build your own extension for Azure Data Studio. We have support for Dashboard Insights (pretty graphs that run against your SQL Server), a number of SQL-specific APIs, and a huge existing set of extension points inherited from Visual Studio Code.

If you have an idea but are not sure how to get started, please open an issue or tweet at the team: azuredatastudio.

You can always refer to the Visual Studio Code extension guide because it covers all the existing APIs and patterns.

To learn how to work with T-SQL in Azure Data Studio, complete the T-SQL Editor tutorial:

Use the Transact-SQL editor to create database objects

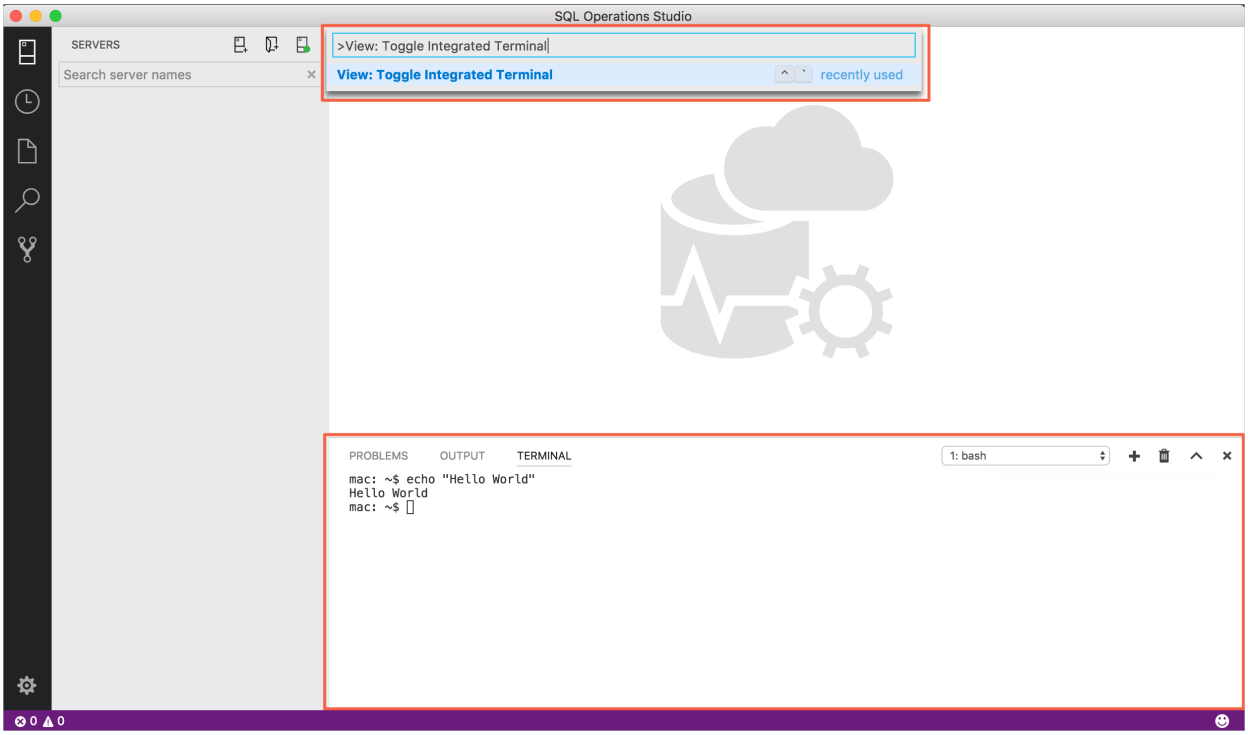
Integrated Terminal

11/2/2020 • 4 minutes to read • Edit Online

In Azure Data Studio, you can open an integrated terminal, initially starting at the root of your workspace. This can be convenient as you don't have to switch windows or alter the state of an existing terminal to perform a quick command-line task.

To open the terminal:

- Use the **Ctrl+`** keyboard shortcut with the backtick character.
- Use the **View | Integrated Terminal** menu command.
- From the **Command Palette** (**Ctrl+Shift+P**), use the **View:Toggle Integrated Terminal** command.

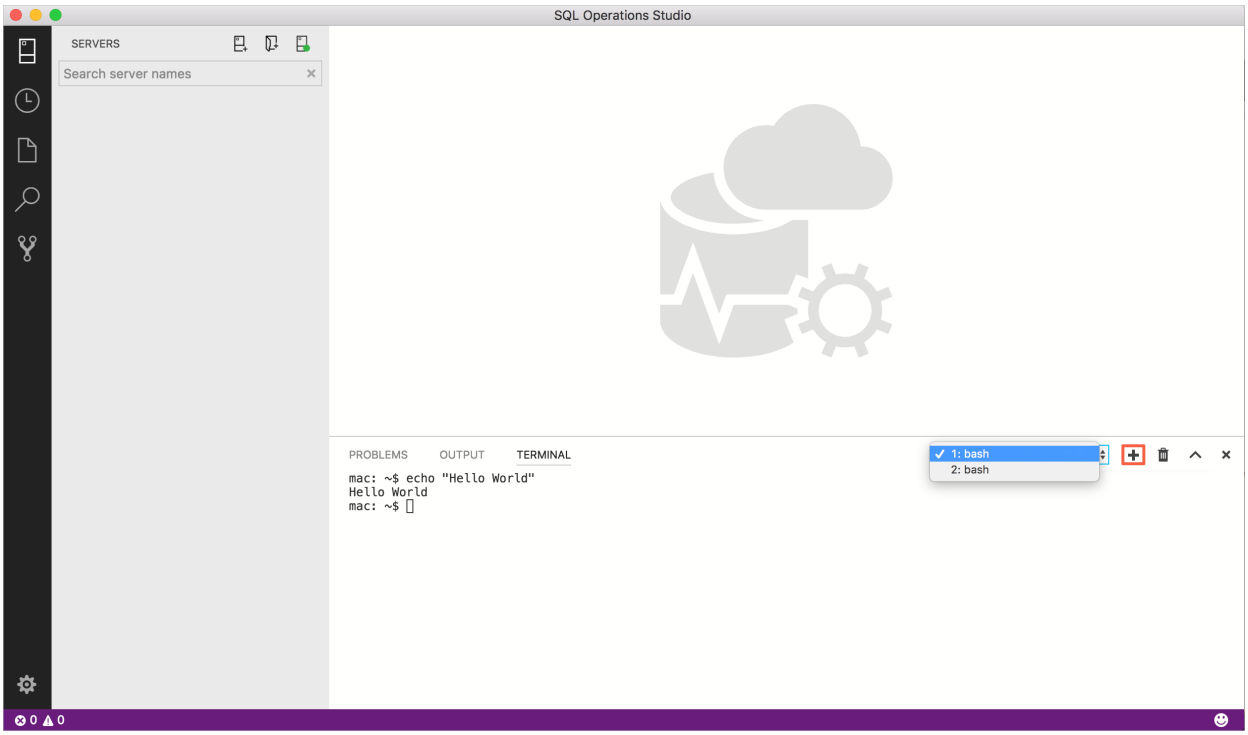


NOTE

You can still open an external shell with the Explorer **Open in Command Prompt** command (**Open in Terminal** on Mac or Linux) if you prefer to work outside Azure Data Studio.

Managing Multiple Terminals

You can create multiple terminals open to different locations and easily navigate between them. Terminal instances can be added by hitting the plus icon on the top-right of the **TERMINAL** panel or by triggering the **Ctrl+Shift+`** command. This creates another entry in the dropdown list that can be used to switch between them.



Remove terminal instances by pressing the trash can button.

> **TIP**
> If you use multiple terminals extensively, you can add key bindings for the `focusNext`, `focusPrevious` and `kill` commands outlined in the Key Bindings section to allow navigation between them using only the keyboard.

Configuration

The shell used defaults to `$SHELL` on Linux and macOS, PowerShell on Windows 10 and cmd.exe on earlier versions of Windows. These can be overridden manually by setting `terminal.integrated.shell.*` in settings. Arguments can be passed to the terminal shell on Linux and macOS using the `terminal.integrated.shellArgs.*` settings.

Windows

Correctly configuring your shell on Windows is a matter of locating the right executable and updating the setting. Below are a list of common shell executables and their default locations:

```
// 64-bit cmd if available, otherwise 32-bit
"terminal.integrated.shell.windows": "C:\\Windows\\sysnative\\cmd.exe"
// 64-bit PowerShell if available, otherwise 32-bit
"terminal.integrated.shell.windows": "C:\\Windows\\sysnative\\WindowsPowerShell\\v1.0\\powershell.exe"
// Git Bash
"terminal.integrated.shell.windows": "C:\\Program Files\\Git\\bin\\bash.exe"
// Bash on Ubuntu (on Windows)
"terminal.integrated.shell.windows": "C:\\Windows\\sysnative\\bash.exe"
```

> **NOTE**
> To be used as an integrated terminal, the shell executable must be a console application so that `stdin/stdout/stderr` can be redirected.

TIP

The integrated terminal shell is running with the permissions of Azure Data Studio. If you need to run a shell command with elevated (administrator) or different permissions, you can use platform utilities such as `runas.exe` within a terminal.

Shell arguments

You can pass arguments to the shell when it is launched.

For example, to enable running bash as a login shell (which runs `.bash_profile`), pass in the `-l` argument (with double quotes):

```
// Linux
"terminal.integrated.shellArgs.linux": ["-l"]
```

Terminal Display Settings

You can customize the integrated terminal font and line height with the following settings:

- `terminal.integrated.fontFamily`
- `terminal.integrated.fontSize`
- `terminal.integrated.lineHeight`

Terminal Key Bindings

The **View: Toggle Integrated Terminal** command is bound to **Ctrl+`** to quickly toggle the integrated terminal panel in and out of view.

Below are the keyboard shortcuts to quickly navigate within the integrated terminal:

KEY	COMMAND
Ctrl+`	Show integrated terminal
Ctrl+Shift+`	Create new terminal
Ctrl+Up	Scroll up
Ctrl+Down	Scroll down
Ctrl+PageUp	Scroll page up
Ctrl+PageDown	Scroll page down
Ctrl+Home	Scroll to top
Ctrl+End	Scroll to bottom
Ctrl+K	Clear the terminal

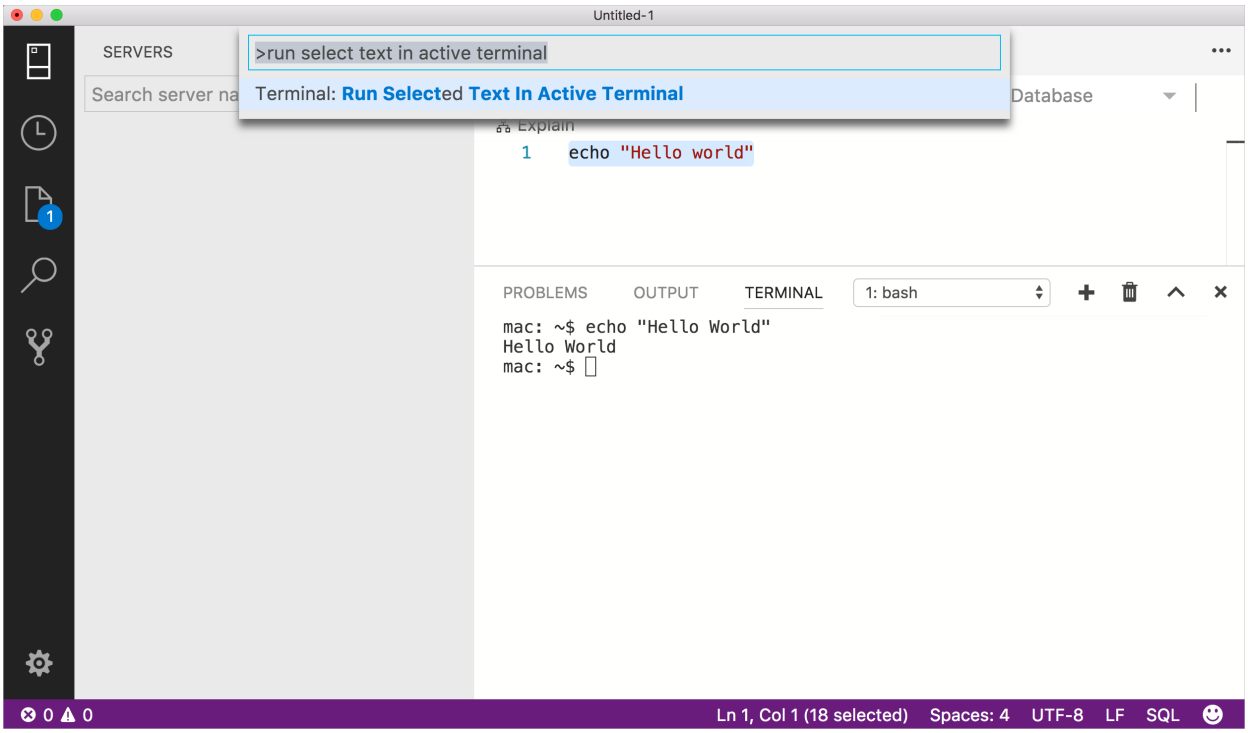
Other terminal commands are available and can be bound to your preferred keyboard shortcuts.

They are:

- `workbench.action.terminal.focus`: Focus the terminal. This is like toggle but focuses the terminal instead of hiding it, if it is visible.
- `workbench.action.terminal.focusNext`: Focuses the next terminal instance.
- `workbench.action.terminal.focusPrevious`: Focuses the previous terminal instance.
- `workbench.action.terminal.kill`: Remove the current terminal instance.
- `workbench.action.terminal.runSelectedText`: Run the selected text in the terminal instance.
- `workbench.action.terminal.runActiveFile`: Run the active file in the terminal instance.

Run Selected Text

To use the `runSelectedText` command, select text in an editor and run the command **Terminal: Run Selected Text in Active Terminal** via the **Command Palette** (**Ctrl+Shift+P**). The terminal attempts to run the selected text:



If no text is selected in the active editor, the line that the cursor is on is run in the terminal.

Copy & Paste

The keybindings for copy and paste follow platform standards:

- Linux: **Ctrl+Shift+C** and **Ctrl+Shift+V**
- Mac: **Cmd+C** and **Cmd+V**
- Windows: **Ctrl+C** and **Ctrl+V**

Find

The Integrated Terminal has basic find functionality that can be triggered with **Ctrl+F**.

If you want **Ctrl+F** to go to the shell instead of launching the Find widget on Linux and Windows, you need to remove the keybinding like so:

```
{ "key": "ctrl+f", "command": "-workbench.action.terminal.focusFindWidget",
                    "when": "terminalFocus" },
```

Rename terminal sessions

Integrated Terminal sessions can now be renamed using the **Terminal: Rename** (

`workbench.action.terminal.rename`) command. The new name is displayed in the terminal selection drop-down.

Forcing key bindings to pass through the terminal

While focus is in the integrated terminal, many key bindings won't work because the keystrokes are passed to and consumed by the terminal itself. The `terminal.integrated.commandsToSkipShell` setting can be used to get around this. It contains an array of command names whose key bindings skip processing by the shell and instead be processed by the Azure Data Studio key binding system. By default this includes all terminal key bindings in addition to a select few commonly used key bindings.

Preview features in Azure Data Studio

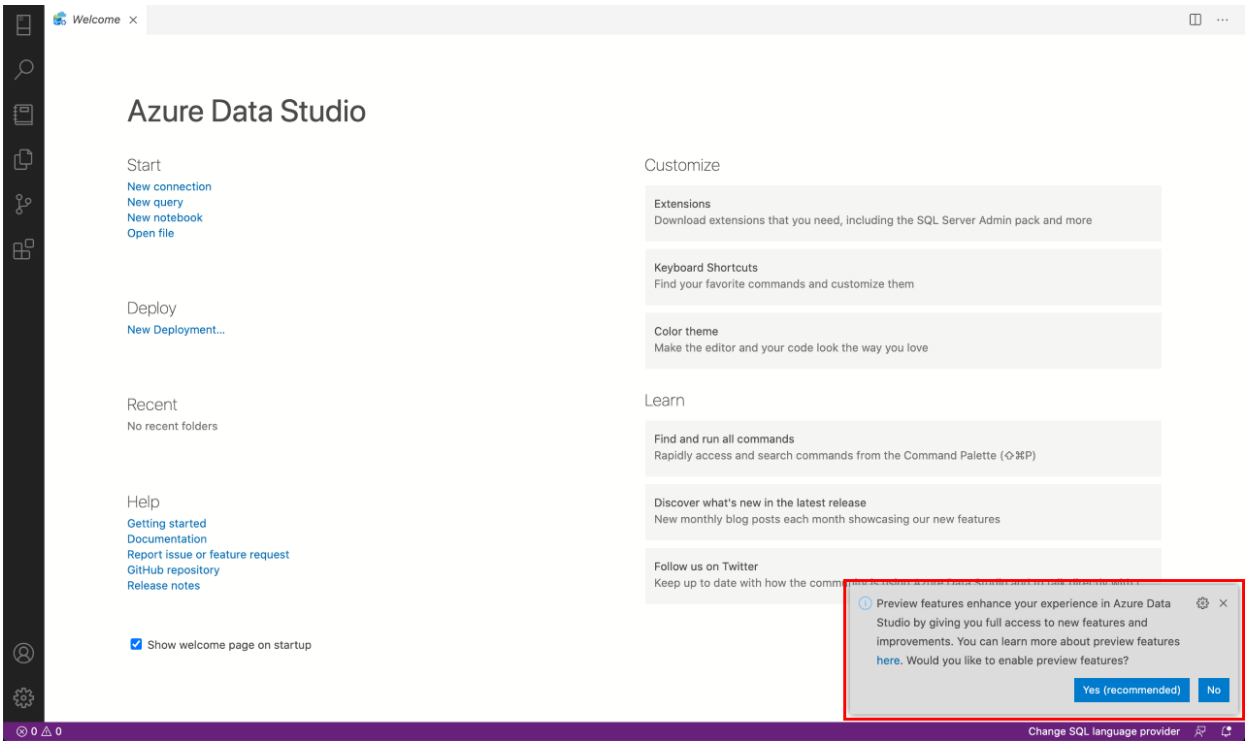
3/5/2021 • 2 minutes to read • Edit Online

In Azure Data Studio, new features and improvements are often first released as preview features before they're made generally available (GA). The amount of time a feature remains in preview can vary based on user feedback, quality checks, and long-term road maps. By enabling preview features, you get full access to Azure Data Studio features and the chance to provide early feedback.

How do I enable preview features?

On first launch

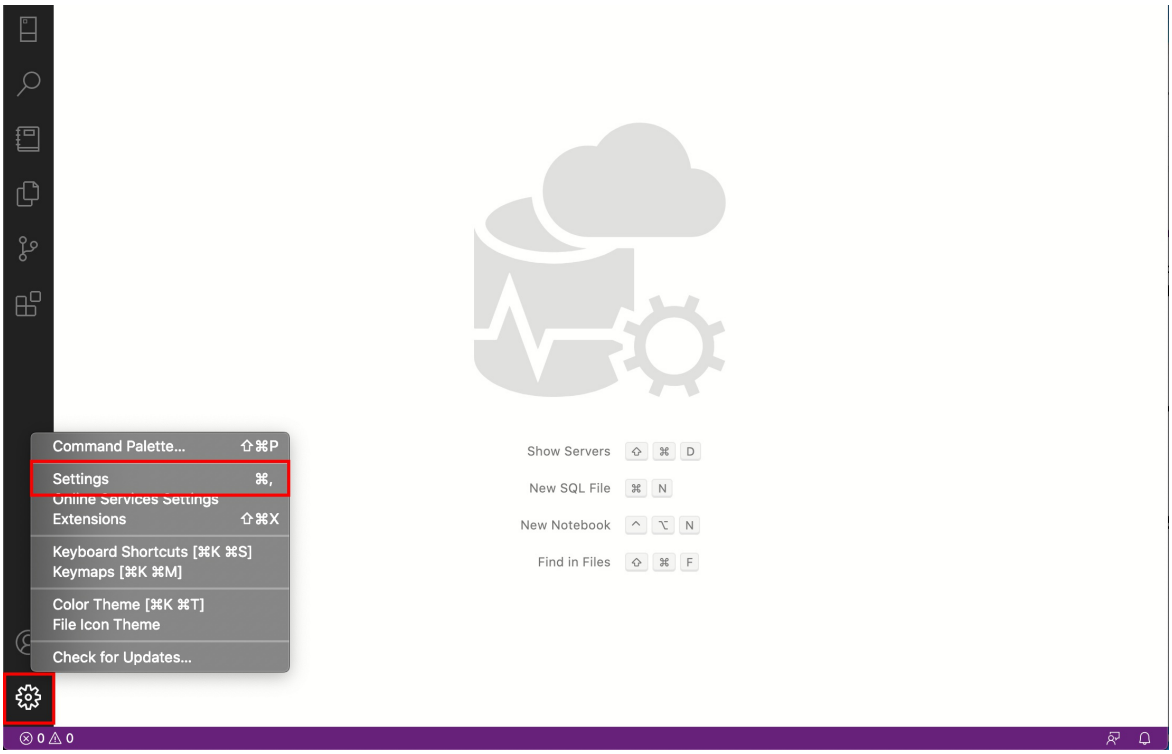
If you're a new user, you can opt into preview features when you launch Azure Data Studio for the first time. On startup, a toast notification will appear in the bottom-right corner of the screen that gives you the option to enable or disable preview features. Select **Yes (recommended)** to enable preview features.



In Settings

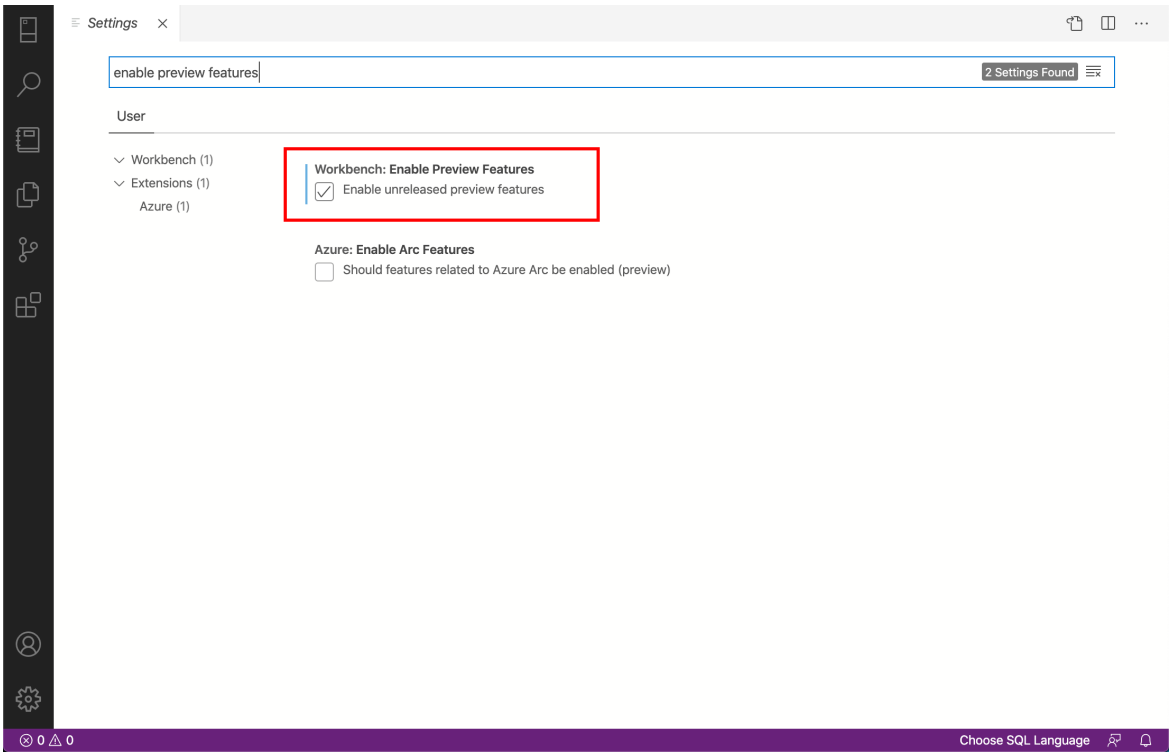
You can enable or disable preview features anytime in your Settings.

1. Select the **Gear** icon in the bottom-left corner and then select **Settings** from the context menu. The Settings tab will open.



2. Type "enable preview features" in the search bar.

3. To enable preview features, check the checkbox for **Enable unreleased preview features** under **Workbench: Enable Preview Features**. To disable preview features, clear the checkbox.



List of preview features in Azure Data Studio

General features in preview

- Azure portal integration
- Backup / Restore
- Deployments
  - SQL Edge
  - SQL Server Big Data Cluster

    - SQL Server container image
    - SQL Server on Windows
- Feature tour
- SQLCMD mode
- New Welcome page

Notebook features in preview

- Dotnet interactive support
- Markdown toolbar
- New Notebook toolbar
- Notebook kernels
    - Kusto
    - PowerShell
    - PySpark
    - Python
    - Spark | Scala
    - Spark | R
    - SQL
- Open Notebook from browser
- Pinned Notebooks
- Python dependencies wizard

First-party extensions in preview

- Admin Pack for SQL Server
- Azure Synapse Analytics Insights
- Central Management Servers
- Database Administration Tool Extensions for Windows
- Kusto
- Language packs
- PostgreSQL
- PowerShell
- Query History
- SandDance for Azure Data Studio
- Server Reports
- SQL Assessment
- SQL Server Agent
- SQL Server Profiler
- Machine Learning
- Managed Instance Dashboard
- Visual Studio IntelliCode
- whoisactive

Next steps

- Azure Data Studio

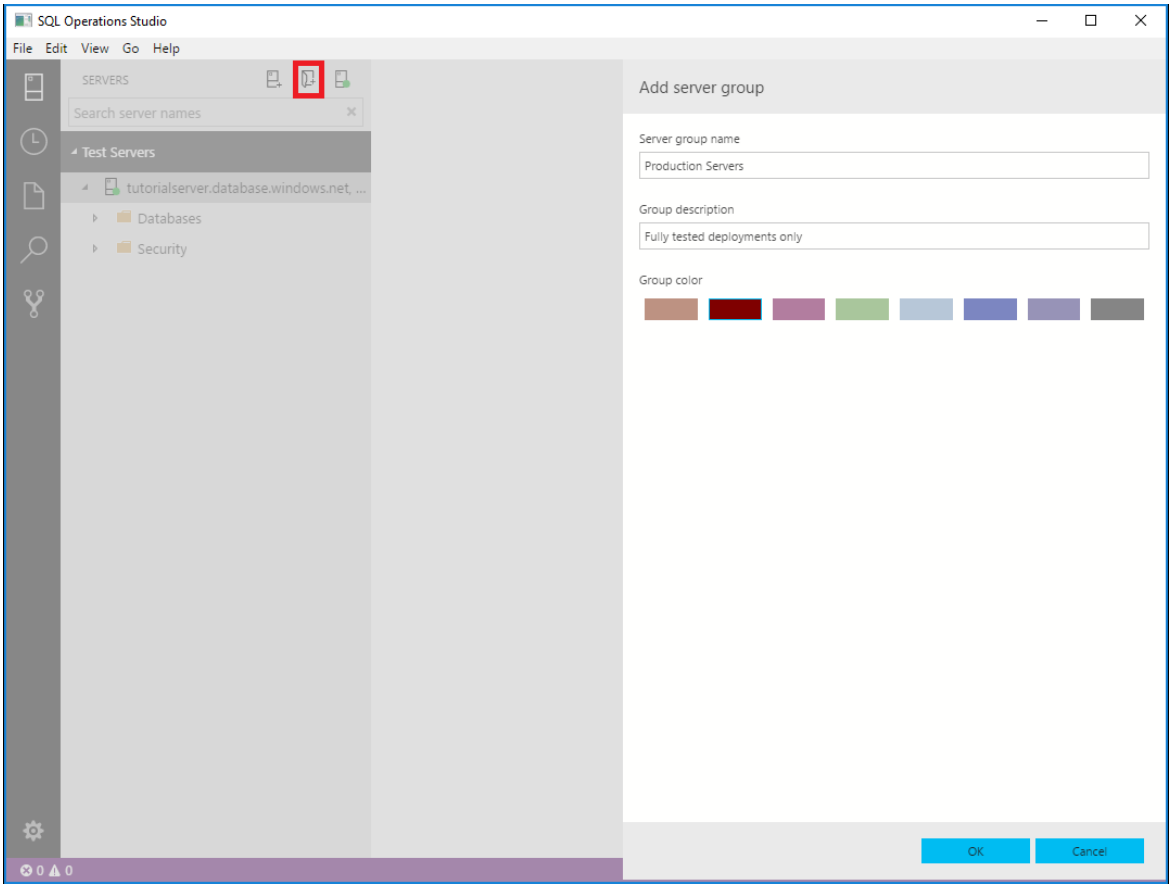
Server groups in Azure Data Studio

11/2/2020 • 2 minutes to read • Edit Online

Server groups provide a way to organize your connections to the servers and databases you work with. When you create server groups, the configuration details are saved into *User Settings*.

Create and edit server groups

1. Click **New Server Group** at the top of the *SERVERS* sidebar.
2. Enter a group name and select a color for the group. Optionally, add a description.



To edit an existing server group, right-click the group, and select **Edit Server Group**.

To edit available server group colors, edit the *Server Groups* values in User Settings.

> **TIP**
> You can drag and drop servers between different Server Groups.

Additional resources

- Workspace and User settings

Source control in Azure Data Studio

11/2/2020 • 2 minutes to read • Edit Online

Azure Data Studio supports Git for version/source control.

Git support in Azure Data Studio

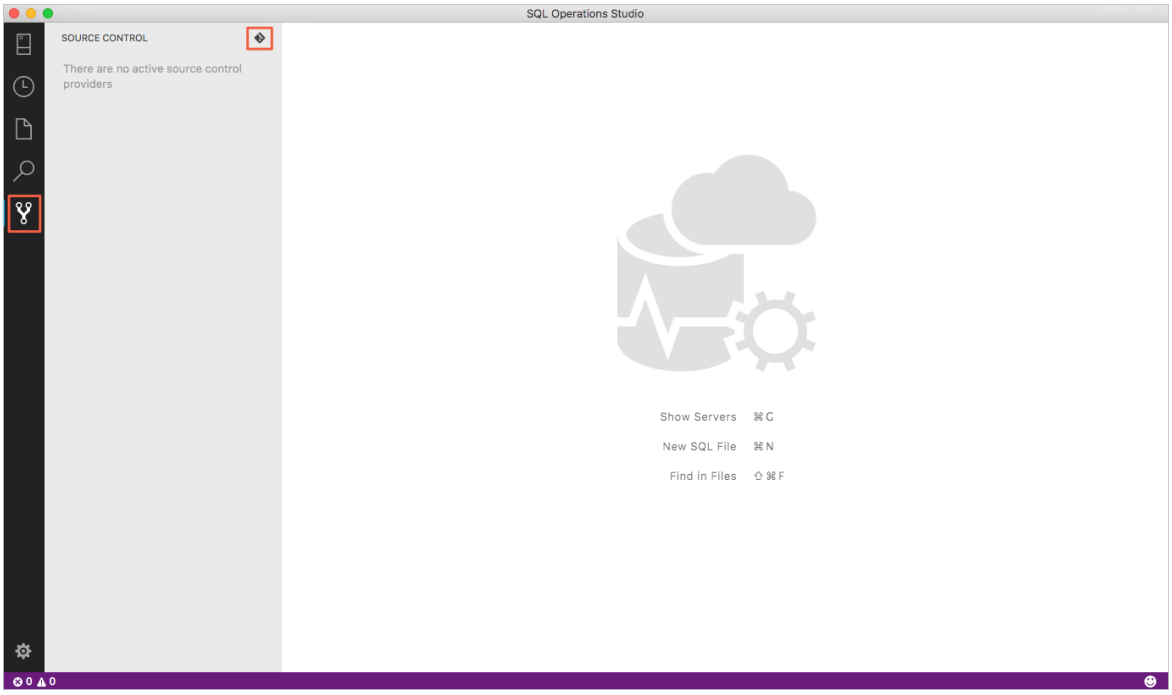
Azure Data Studio ships with a Git source control manager (SCM), but you still need to install Git (version 2.0.0 or later) before these features are available.

Open an existing Git repository

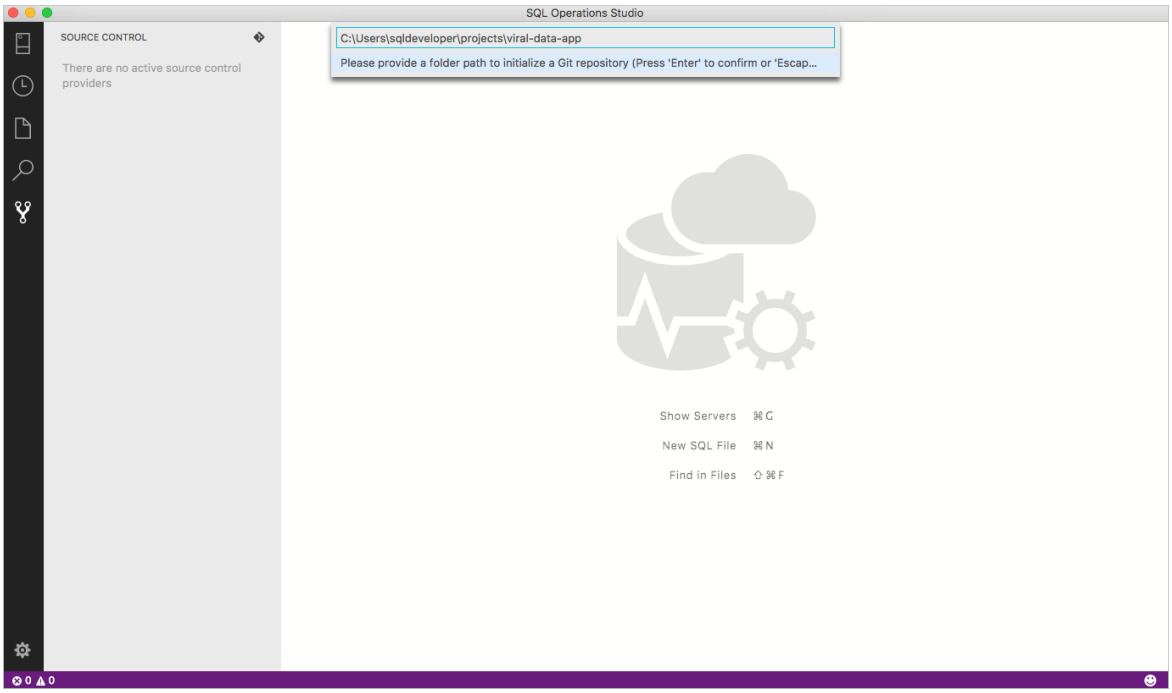
1. Under the **File** menu, select **Open Folder...**
2. Browse to the folder that contains your files tracked by git, and select **Select Folder**. Subfolders in your local repository are okay to select here.

Initialize a new git repository

1. Select **Source Control**, then select the git icon.



2. Enter the path to the folder you want to initialize as a Git repository and press **Enter**.



Working with Git repositories

Azure Data Studio inherits its Git implementation from VS Code, but doesn't currently support additional SCM providers. For the details about working with Git after you open or initialize a repository, see Git support in VS Code.

Additional resources

- Git documentation

Use Jupyter Notebooks in Azure Data Studio

11/2/2020 • 4 minutes to read • Edit Online

Applies to: SQL Server 2019 (15.x)

Jupyter Notebook is an open-source web application that allows you to create and share documents containing live code, equations, visualizations, and narrative text. Usage includes data cleaning and transformation, numerical simulation, statistical modeling, data visualization, and machine learning.

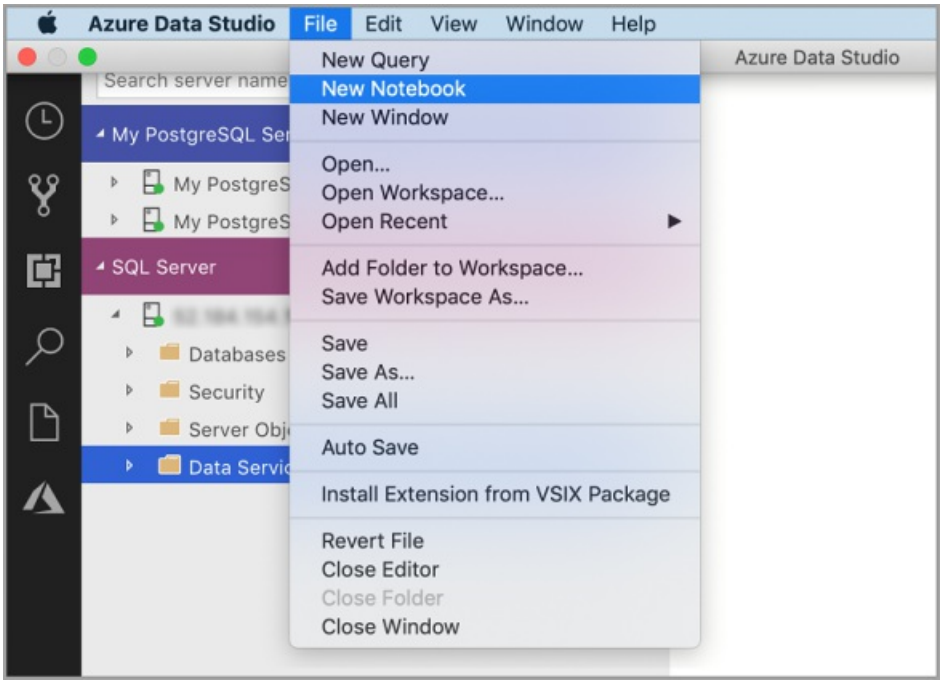
This article describes how to create a new notebook in the latest release of Azure Data Studio and how to start authoring your own notebooks using different kernels.

Watch this short 5-minute video for an introduction to notebooks in Azure Data Studio:

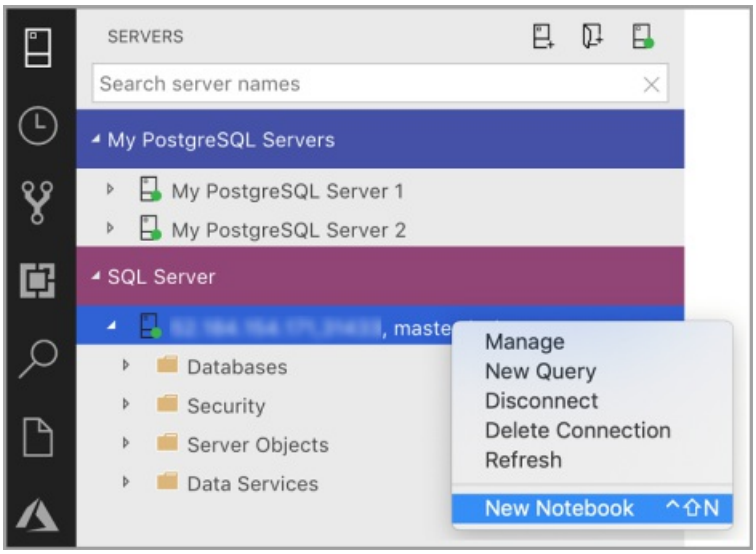
Create a notebook

There are multiple ways to create a new notebook. In each case, a new file named `Notebook-1.ipynb` opens.

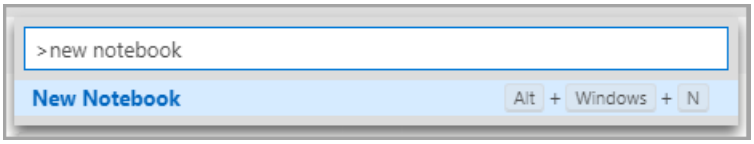
- Go to the **File Menu** in Azure Data Studio and select **New Notebook**.



- Right-click a **SQL Server** connection and select **New Notebook**.



- Open the command palette (**Ctrl+Shift+P**), type "new notebook", and select the **New Notebook** command.



Connect to a kernel

Azure Data Studio notebooks support a number of different kernels, including SQL Server, Python, PySpark, and others. Each kernel supports a different language in the code cells of your notebook. For example, when connected to the SQL Server kernel, you can enter and run T-SQL statements in a notebook code cell.

Attach to provides the context for the kernel. For example, if you're using SQL Kernel, then you can attach to any of your SQL Server instances. If you're using Python3 Kernel you attach to **localhost** and you can use this kernel for your local Python development.

SQL Kernel can also be used to connect to PostgreSQL server instances. If you're a PostgreSQL developer and want to connect the notebooks to your PostgreSQL Server, then download the **PostgreSQL extension** in the Azure Data Studio extension Marketplace and connect to the PostgreSQL server.

If you're connected to SQL Server 2019 big data cluster, the default **Attach to** is the end point of the cluster. You can submit Python, Scala, and R code using the Spark compute of the cluster.

KERNEL	DESCRIPTION
SQL Kernel	Write SQL Code targeted at your relational database.
PySpark3 and PySpark Kernel	Write Python code using Spark compute from the cluster.
Spark Kernel	Write Scala and R code using Spark compute from the cluster.
Python Kernel	Write Python code for local development.

For more information on specific kernels, see:

- Create and run a SQL Server notebook
- Create and run a Python notebook
- Kqlmagic extension in Azure Data Studio - this extends the capabilities of the Python kernel

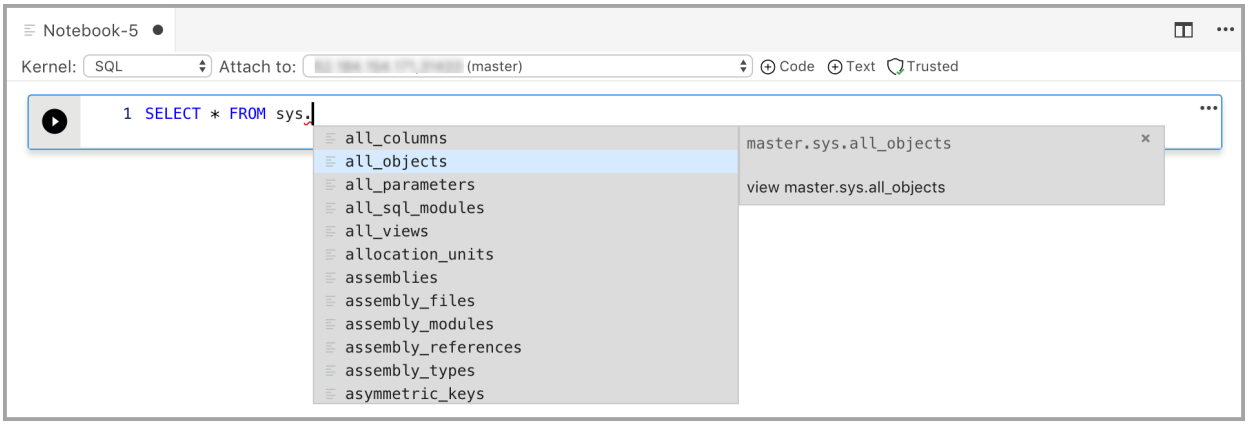
Add a code cell

Code cells allow you to run code interactively within the notebook.

Add a new code cell by clicking the **+Cell** command in the toolbar and selecting **Code cell**. A new code cell is added after the currently selected cell.

Enter code in the cell for the selected kernel. For example, if you're using the SQL kernel, you can enter T-SQL commands in the code cell.

Entering code with the SQL kernel is similar to a SQL query editor. The code cell supports a modern SQL coding experience with built-in features such as a rich SQL editor, IntelliSense, and built-in code snippets. Code snippets allow you to generate the proper SQL syntax to create databases, tables, views, stored procedures, and to update existing database objects. Use code snippets to quickly create copies of your database for development or testing purposes and to generate and execute scripts.

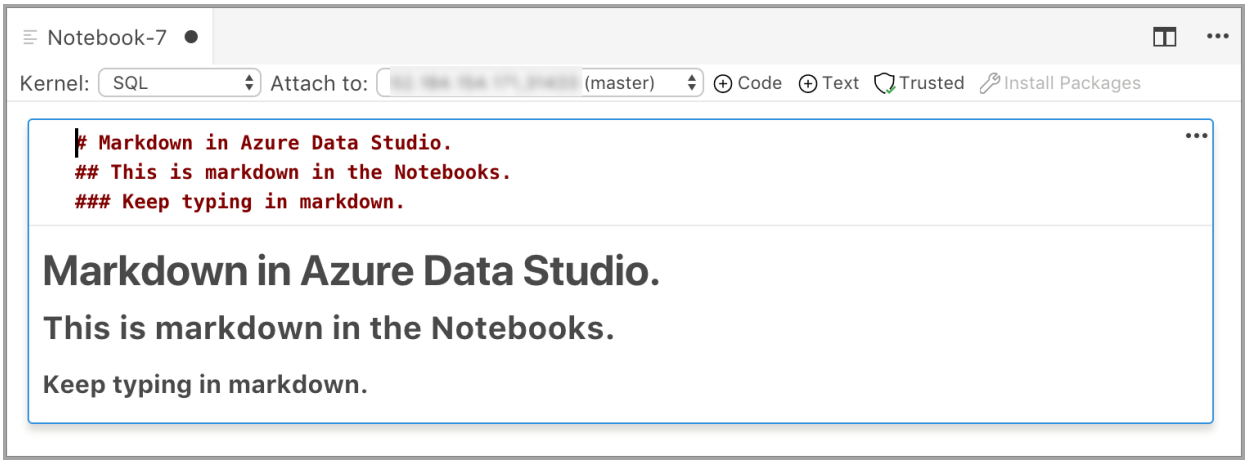


Add a text cell

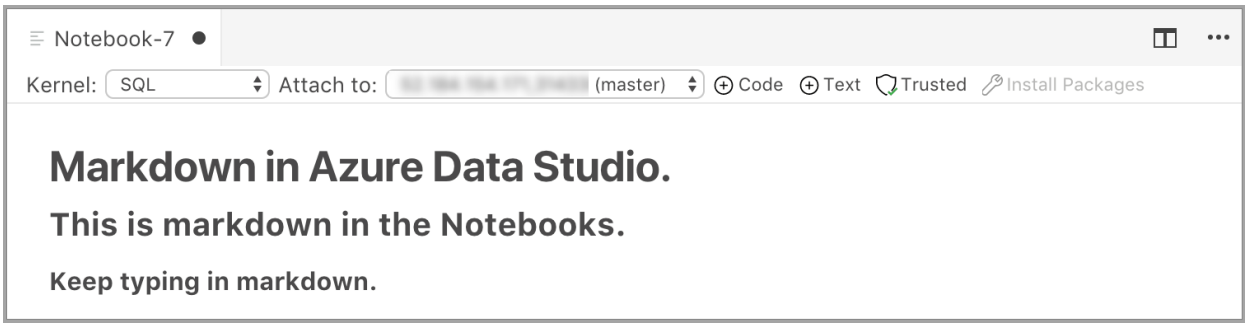
Text cells allow you to document your code by adding Markdown text blocks in between code cells.

Add a new text cell by clicking the **+Cell** command in the toolbar and selecting **Text cell**.

The cell starts in edit mode in which you can type Markdown text. As you type, a preview is shown below.



Selecting outside the text cell shows the Markdown text.



If you click in the text cell again, it changes to edit mode.

Run a cell

To run a single cell, click **Run cell** (the round black arrow) to the left of the cell or select the cell and press F5. You can run all cells in the notebook by clicking **Run all** in the toolbar - the cells are run one at a time and execution stops if an error is encountered in a cell.

Results from the cell are shown below the cell. You can clear the results of all the executed cells in the notebook by selecting the **Clear Results** button in the toolbar.

Save a notebook

To save a notebook, do one of the following.

- Type Ctrl+S
- Select **Save** from the **File** menu
- Select **Save As...** from the **File** menu
- Select **Save All** from the **File** menu - this saves all open notebooks
- In the command palette, enter **File: Save**

Notebooks are saved as `.ipynb` files.

Trusted and Non Trusted

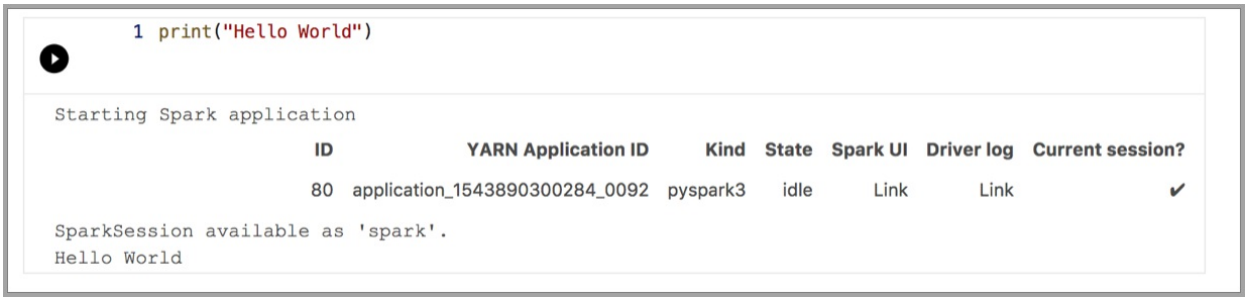
The notebooks open in Azure Data Studio are defaulted to **Trusted**.

If you open a notebook from some other source, it opens in **Non-Trusted** mode and then you can make it **Trusted**.

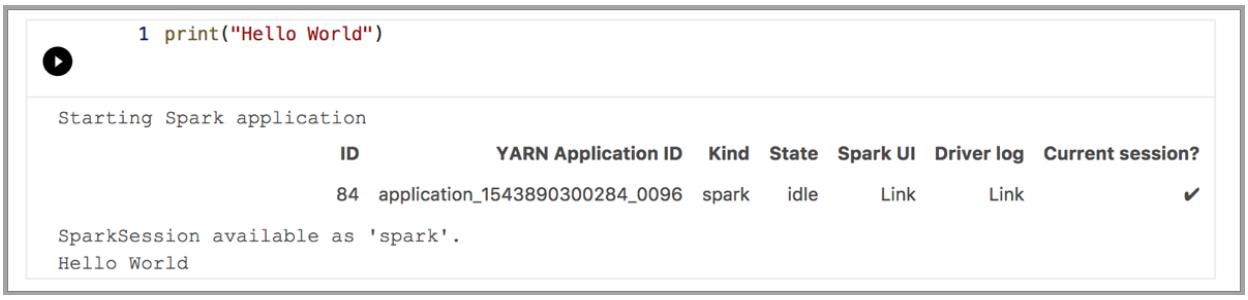
Examples

The following examples demonstrate using different kernels to run a simple "Hello World" command. Select the kernel, enter the example code in a cell, and click **Run cell**.

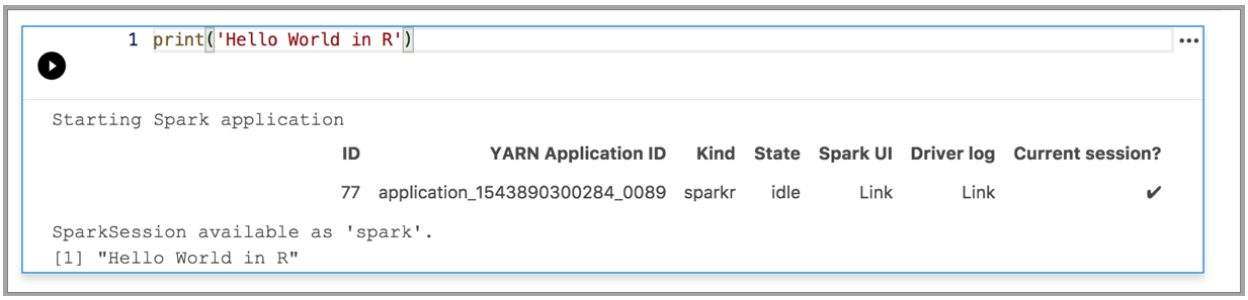
Pyspark



Spark | Scala language



Spark | R language



Python 3

Next steps

- Create and run a SQL Server notebook.
- Create and run a Python notebook
- Run Python and R scripts in Azure Data Studio notebooks with SQL Server Machine Learning Services.
- Deploy SQL Server big data cluster with Azure Data Studio notebook.
- Manage SQL Server Big Data Clusters with Azure Data Studio notebooks.
- Run a sample notebook using Spark.

Create and run a SQL Server notebook

11/2/2020 • 2 minutes to read • Edit Online

Applies to: SQL Server 2019 (15.x)

This tutorial demonstrates how to create and run a notebook in Azure Data Studio using SQL Server.

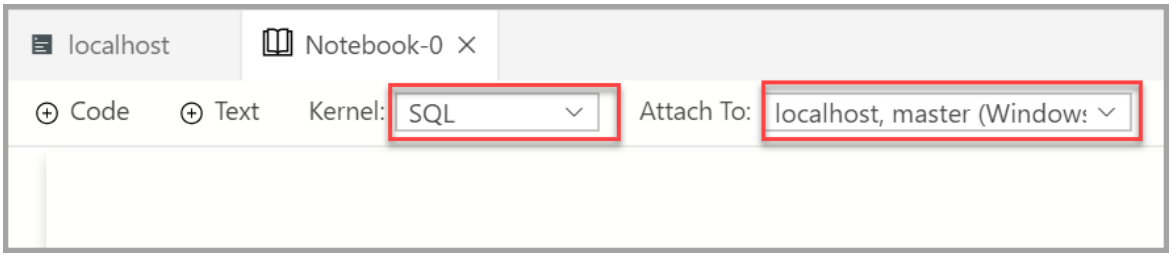
Prerequisites

- Azure Data Studio installed
- SQL Server installed
  - Windows
  - Linux

Create a notebook

The following steps show how to create a notebook file in Azure Data Studio:

1. In Azure Data Studio, connect to your SQL Server.
2. Select under the **Connections** in the **Servers** window. Then select **New Notebook**.
3. Wait for the **Kernel** and the target context (**Attach to**) to be populated. Confirm that the **Kernel** is set to **SQL**, and set **Attach to** for your SQL Server (in this example it's *localhost*).



You can save the notebook using the **Save** or **Save as...** command from the **File** menu.

To open a notebook, you can use the **Open file...** command in the **File** menu, select **Open file** on the **Welcome** page, or use the **File: Open** command from the command palette.

Change the SQL connection

To change the SQL connection for a notebook:

1. Select the **Attach to** menu from the notebook toolbar and then select **Change Connection**.

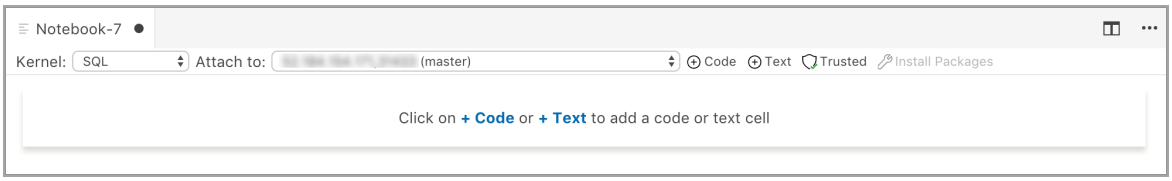
2. Now you can either select a recent connection server or enter new connection details to connect.

Run a code cell

You can create cells containing SQL code that you can run in place by clicking the **Run cell** button (the round black arrow) to the left of the cell. The results are shown in the notebook after the cell finishes running.

For example:

1. Add a new code cell by selecting the **+Code** command in the toolbar.

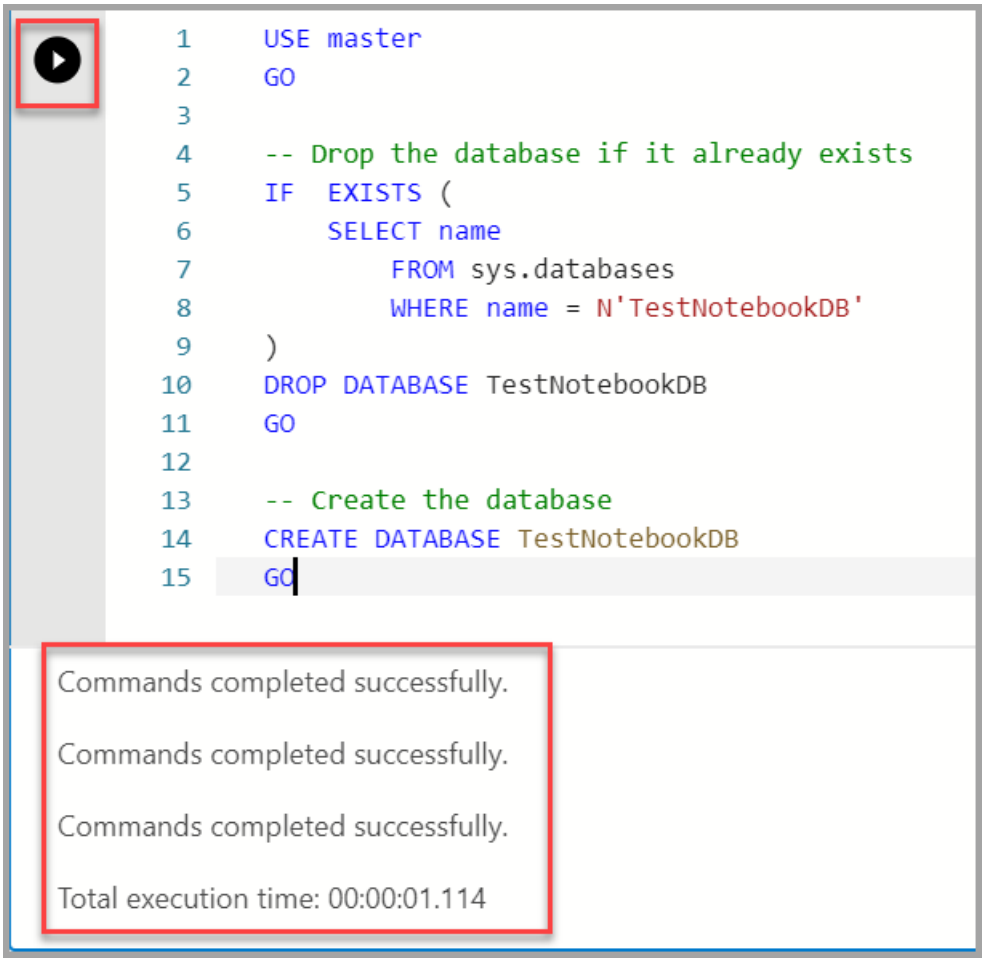


2. Copy and paste the following example into the cell and click **Run cell**. This example creates a new database.

```
USE master
GO

-- Drop the database if it already exists
IF  EXISTS (
        SELECT name
        FROM sys.databases
        WHERE name = N'TestNotebookDB'
   )
DROP DATABASE TestNotebookDB
GO

-- Create the database
CREATE DATABASE TestNotebookDB
GO
```



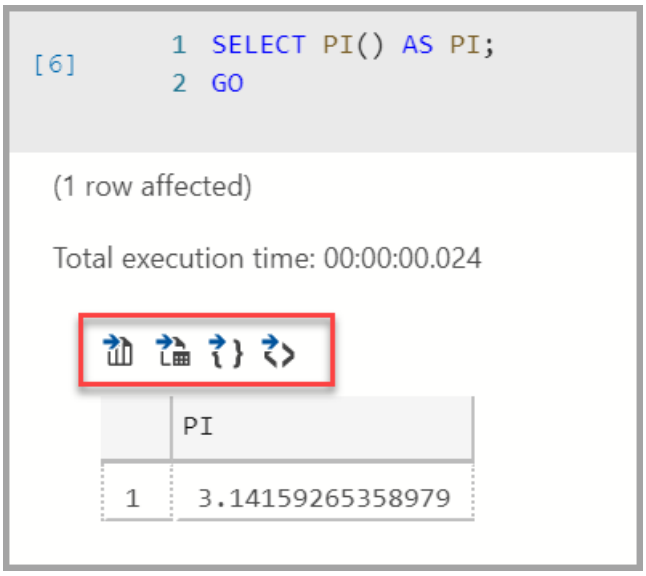
Save the result

If you run a script that returns a result, you can save that result in different formats using the toolbar displayed above the result.

- Save As CSV
- Save As Excel
- Save As JSON
- Save As XML

For example, the following code returns the result of PI.

```
SELECT PI() AS PI;
GO
```



Next steps

Learn more about notebooks:

- How to use notebooks in Azure Data Studio
- Create and run a Python notebook
- Run a sample notebook using Spark

Create and run a Python notebook

11/2/2020 • 2 minutes to read • Edit Online

Applies to: SQL Server 2019 (15.x)

This tutorial demonstrates how to create and run a notebook in Azure Data Studio using the Python kernel.

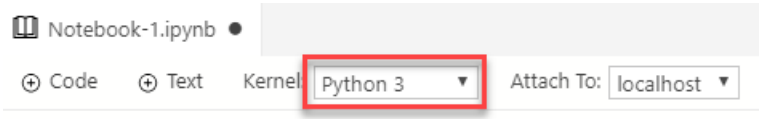
Prerequisites

- Azure Data Studio installed

Create a notebook

The following steps show how to create a notebook file in Azure Data Studio:

1. Open Azure Data Studio. If you're prompted to connect to a SQL Server, you may connect or click **Cancel**.
2. Select **New Notebook** in the **File** menu.
3. Select **Python 3** for the **Kernel**. **Attach to** is set to "localhost".



You can save the notebook using the **Save** or **Save as...** command from the **File** menu.

To open a notebook, you can use the **Open file...** command in the **File** menu, select **Open file** on the **Welcome** page, or use the **File: Open** command from the command palette.

Change the Python kernel

The first time you connect to the Python kernel in a notebook, the **Configure Python for Notebooks** page is displayed. You can select either:

- **New Python installation** to install a new copy of Python for Azure Data Studio, or
- **Use existing Python installation** to specify the path to an existing Python installation for Azure Data Studio to use

To view the location and version of the active Python kernel, create a code cell and run the following Python commands:

```
import os
import sys
print(sys.version_info)
print(os.path.dirname(sys.executable))
```

To connect to a different installation of Python:

1. From the **File** menu, select **Preferences** and then **Settings**.
2. Scroll to **Notebook configuration** under **Extensions**.
3. Under **Use Existing Python**, uncheck the option "Local path to a preexisting python installation used by Notebooks."

4. Restart Azure Data Studio.

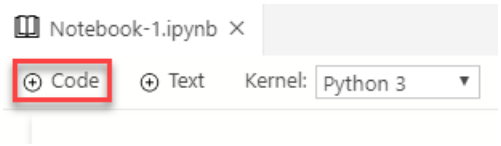
When Azure Data Studio starts and you connect to the Python kernel, the **Configure Python for Notebooks** page is displayed, You can choose to create a new Python installation or specify a path to an existing installation.

Run a code cell

You can create cells containing SQL code that you can run in place by clicking the **Run cell** button (the round black arrow) to the left of the cell. The results are shown in the notebook after the cell finishes running.

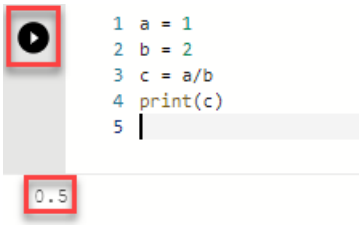
For example:

1. Add a new Python code cell by selecting the **+Code** command in the toolbar.



2. Copy and paste the following example into the cell and click **Run cell**. This example does simple math and the result appears below.

```
a = 1
b = 2
c = a/b
print(c)
```



Next steps

Learn more about notebooks:

- Extend Python with Kqlmagic
- How to use notebooks in Azure Data Studio
- Create and run a SQL Server notebook

Create and run a Kusto (KQL) notebook (Preview)

11/2/2020 • 2 minutes to read • Edit Online

This article shows you how to create and run an Azure Data Studio notebook using the Kusto (KQL) extension, connecting to an Azure Data Explorer cluster.

With the Kusto (KQL) extension, you can change the kernel option to **Kusto**.

This feature is currently in preview.

Prerequisites

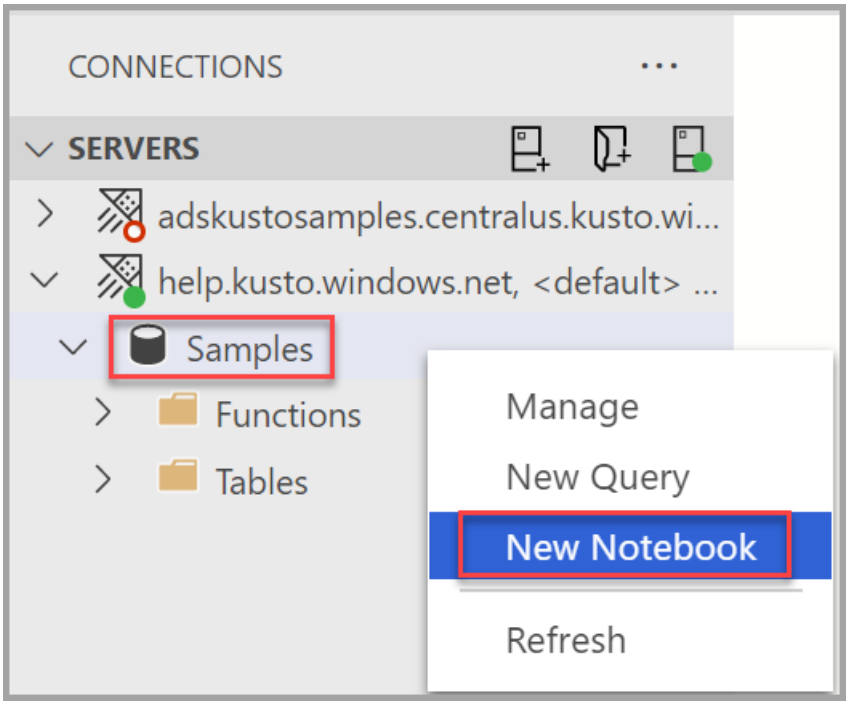
If you don't have an Azure subscription, create a free Azure account before you begin.

- An Azure Data Explorer cluster with a database that you can connect to.
- Azure Data Studio.
- Kusto (KQL) extension for Azure Data Studio.

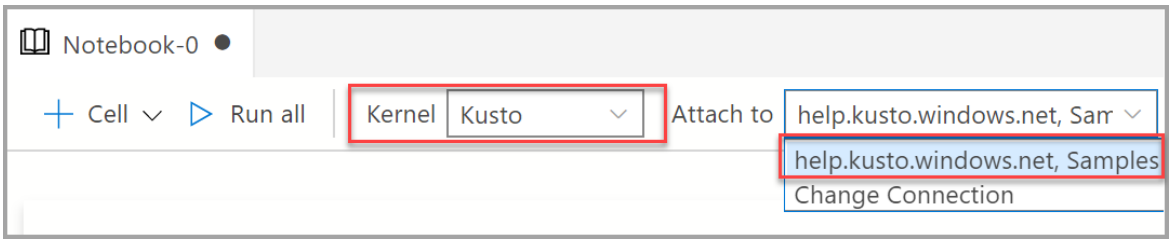
Create a Kusto (KQL) notebook

The following steps show how to create a notebook file in Azure Data Studio:

1. In Azure Data Studio, connect to your Azure Data Explorer cluster.
2. Navigate to the **Connections** pane and under the **Servers** window, right-click the Kusto database and select *New Notebook*. You can also go to **File** > **New Notebook**.



3. Select *Kusto* for the **Kernel**. Confirm that the **Attach to** menu is set to the cluster name and database. For this article, we use the help.kusto.windows.net cluster with the Samples database data.



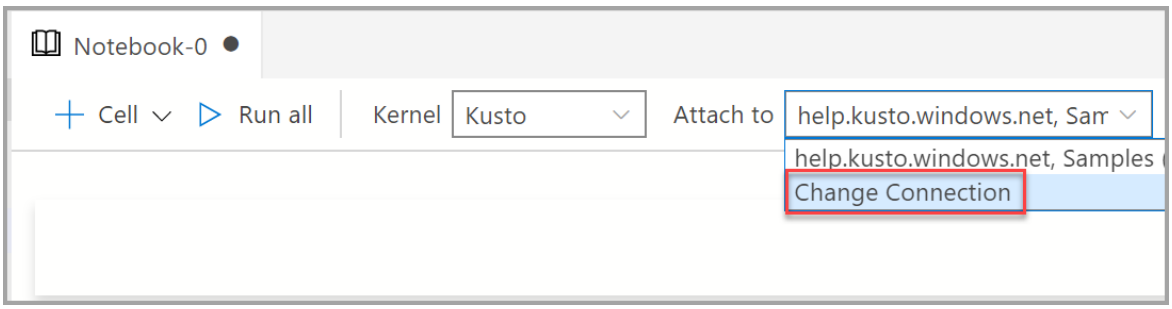
You can save the notebook using the **Save** or **Save as...** command from the **File** menu.

To open a notebook, you can use the **Open file...** command in the **File** menu, select **Open file** on the **Welcome** page, or use the **File: Open** command from the command palette.

Change the connection

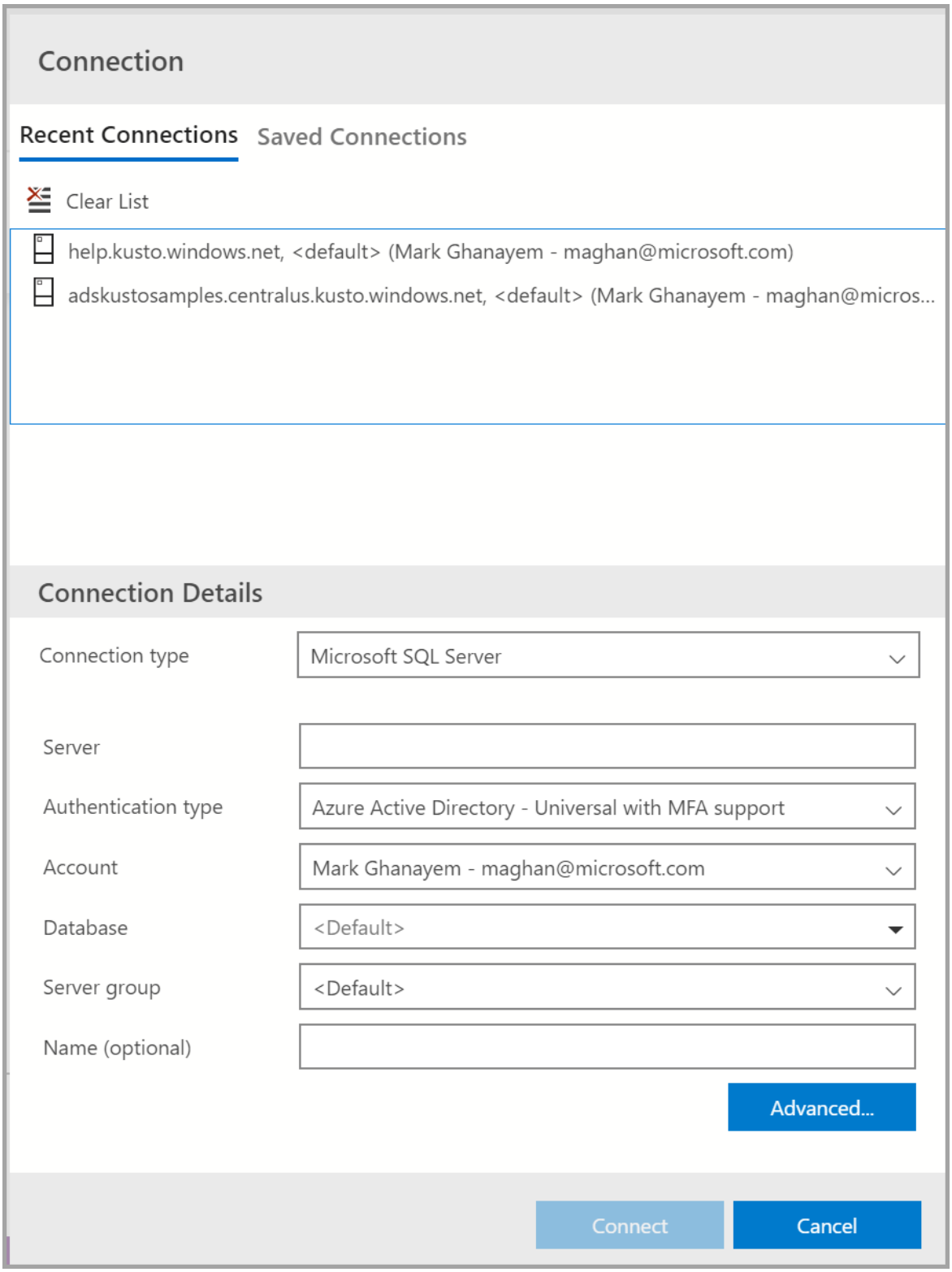
To change the Kusto connection for a notebook:

1. Select the **Attach to** menu from the notebook toolbar and then select **Change Connection**.



> **NOTE**
> Ensure that the database value is populated. Kusto notebooks require to have the database specified.

2. Now you can either select a recent connection server or enter new connection details to connect.



NOTE

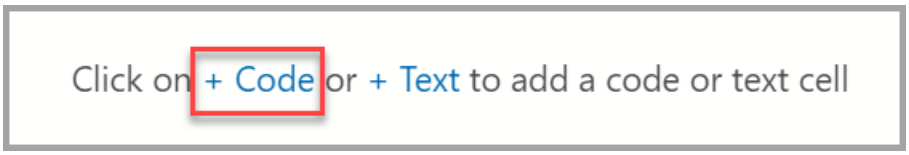
Specify the cluster name without the `https://`.

Run a code cell

You can create cells containing KQL queries that you can run in place by selecting the **Run cell** button to the cell's left. The results are shown in the notebook after the cell runs.

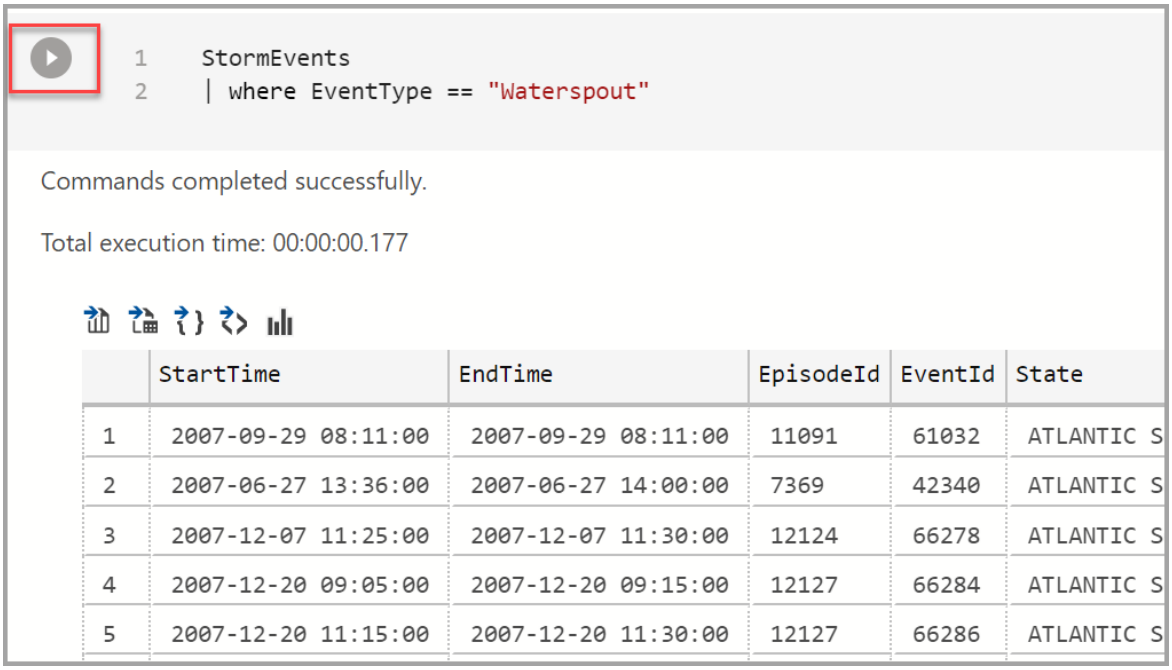
For example:

1. Add a new code cell by selecting the **+Code** command in the toolbar.



2. Copy and paste the following example into the cell and select **Run cell**. This example queries the StormEvents data for a specific event type.

```
StormEvents
| where EventType == "Waterspout"
```



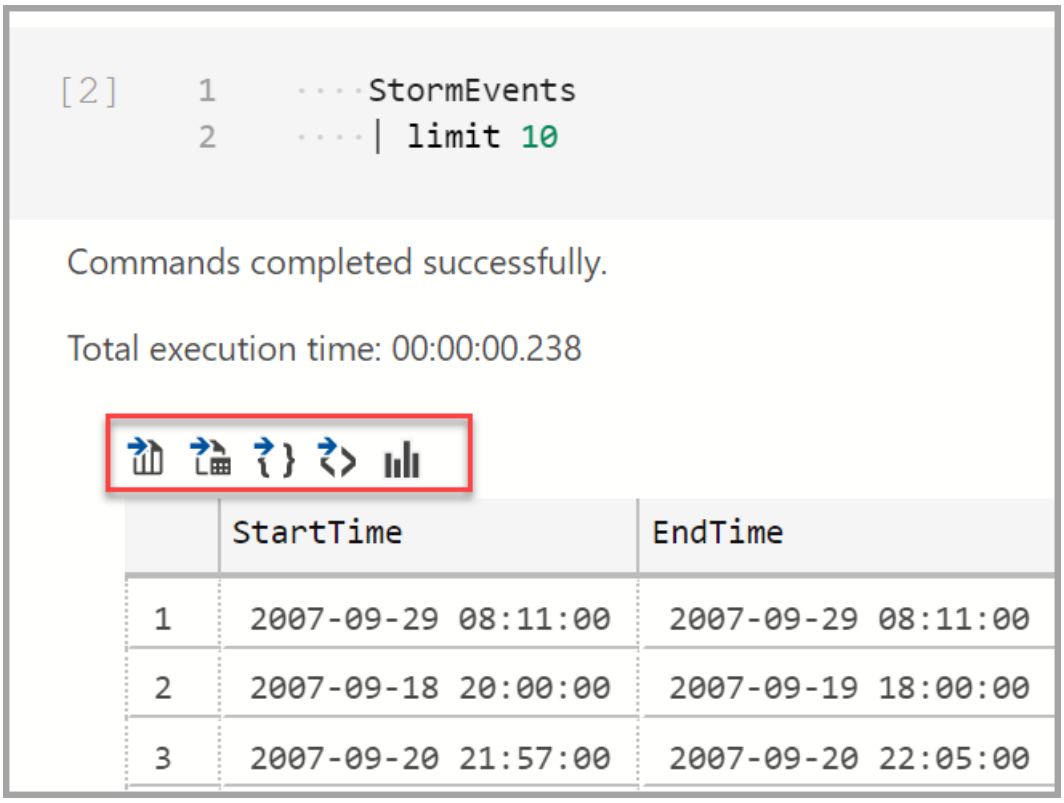
	StartTime	EndTime	EpisodeId	EventId	State
1	2007-09-29 08:11:00	2007-09-29 08:11:00	11091	61032	ATLANTIC S
2	2007-06-27 13:36:00	2007-06-27 14:00:00	7369	42340	ATLANTIC S
3	2007-12-07 11:25:00	2007-12-07 11:30:00	12124	66278	ATLANTIC S
4	2007-12-20 09:05:00	2007-12-20 09:15:00	12127	66284	ATLANTIC S
5	2007-12-20 11:15:00	2007-12-20 11:30:00	12127	66286	ATLANTIC S

Save the result or show chart

If you run a script that returns a result, you can save that result in different formats using the toolbar displayed above the result.

- Save As CSV
- Save As Excel
- Save As JSON
- Save As XML
- Show Chart

```
StormEvents
| limit 10
```



Known issues

DETAILS	WORKAROUND
Query result only shows column headers.	N/A

You can file a feature request to provide feedback to the product team.
You can file a bug to provide feedback to the product team.

Next steps

Learn more about notebooks:

- Kusto (KQL) extension for Azure Data Studio
- How to use notebooks in Azure Data Studio
- Create and run a Python notebook
- Create and run a SQL Server notebook

Create and run a notebook with Kqlmagic

6/28/2021 • 4 minutes to read • Edit Online

Kqlmagic is a command that extends the capabilities of the Python kernel in **Azure Data Studio notebooks**. You can combine Python and **Kusto query language (KQL)** to query and visualize data using rich Plot.ly library integrated with `render` commands. Kqlmagic brings you the benefit of notebooks, data analysis, and rich Python capabilities all in the same location. Supported data sources with Kqlmagic include **Azure Data Explorer**, **Application Insights**, and **Azure Monitor logs**.

This article shows you how to create and run a notebook in Azure Data Studio using the Kqlmagic extension for an Azure Data Explorer cluster, an Application Insights log, and Azure Monitor logs.

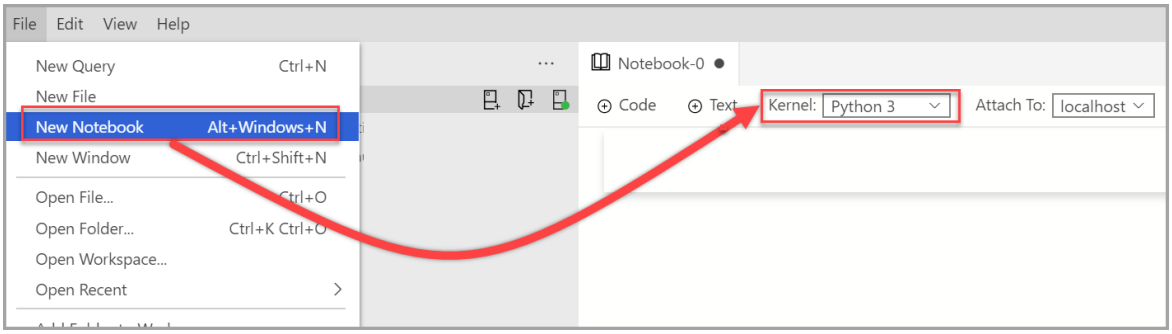
Prerequisites

- Azure Data Studio
- Python

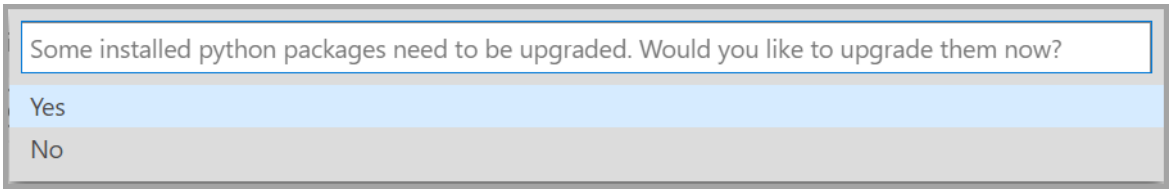
Install and set up Kqlmagic in a notebook

The steps in this section all run within an Azure Data Studio notebook.

1. Create a new notebook and change the **Kernel** to *Python 3*.



2. You may be prompted to upgrade your Python packages when your packages need updating.

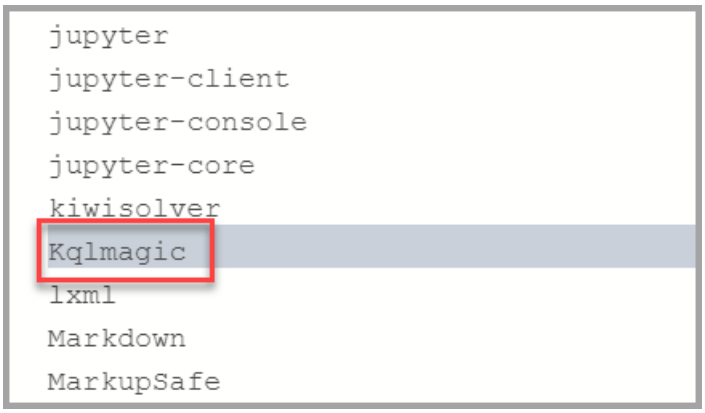


3. Install Kqlmagic:

```
import sys
!{sys.executable} -m pip install Kqlmagic --no-cache-dir --upgrade
```

Verify it's installed:

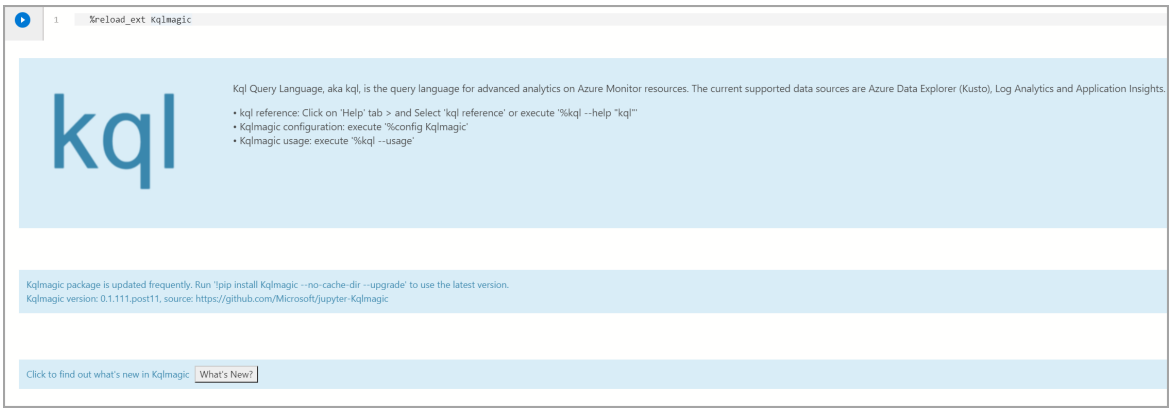
```
import sys
!{sys.executable} -m pip list
```



4. Load Kqlmagic:

```
%reload_ext Kqlmagic
```

> **NOTE**
>
> If this step fails, then close the file and reopen it.



5. You can test if Kqlmagic is loaded properly by browsing the help documentation or by checking for the version.

```
%kql --help "help"
```

> **NOTE**
>
> If `Samples@help` is asking for a password, then you can leave it blank and press **Enter**.

```
1 %kql --help "help"
```

Overview
Help command is a tool to get more information on a topics that are relevant to Kqlmagic. t usage: %kql --help "topic"

Topics

- **usage** - How to use Kqlmagic.

-**config** - Lists Kqlmagic default options. The same as --config without parameters.

- **faq** - Reference to Kqlmagic FAQ
- **conn** - Lists the available connection string variation, and how their are used to authenticatie to data sources.
- **query** / **kql** - Reference to resources Kusto Query language, aka kql, documentation
- **options** - Lists the available options, and their behavior impact on the submit query command.
- **commands** - Lists the available commands, and what they do.
- **proxies** - How to use Kqlmagic via proxies.

To see which version of Kqlmagic is installed, run the command below.

```
%kql --version
```

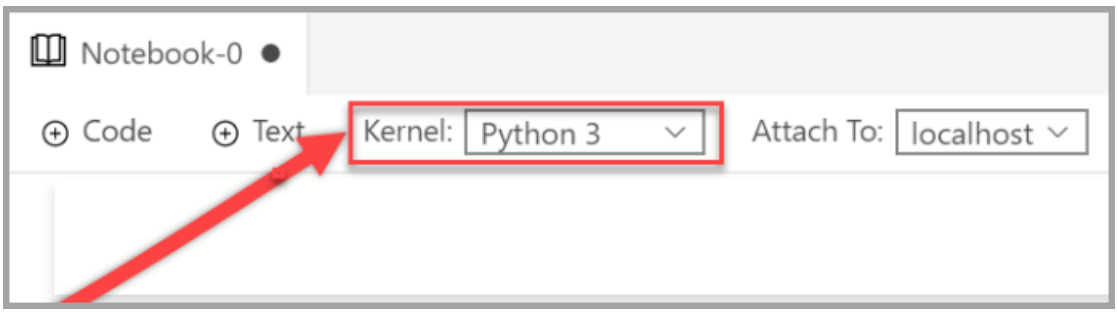
Kqlmagic with an Azure Data Explorer cluster

This section explains how to run data analysis using Kqlmagic with an Azure Data Explorer cluster.

Load and authenticate Kqlmagic for Azure Data Explorer

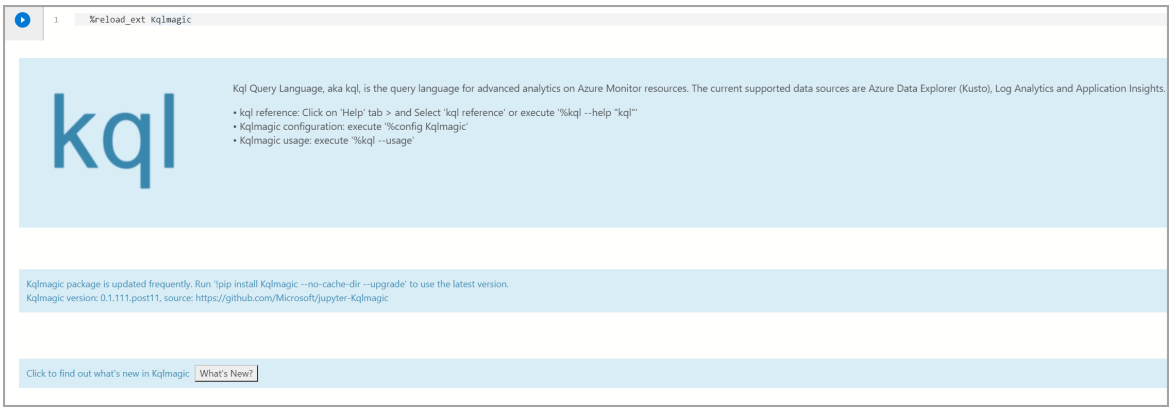
> **NOTE**
> Every time you create a new notebook in Azure Data Studio you must load the Kqlmagic extension.

1. Verify the **Kernel** is set to *Python3*.



2. Load Kqlmagic:

```
%reload_ext Kqlmagic
```

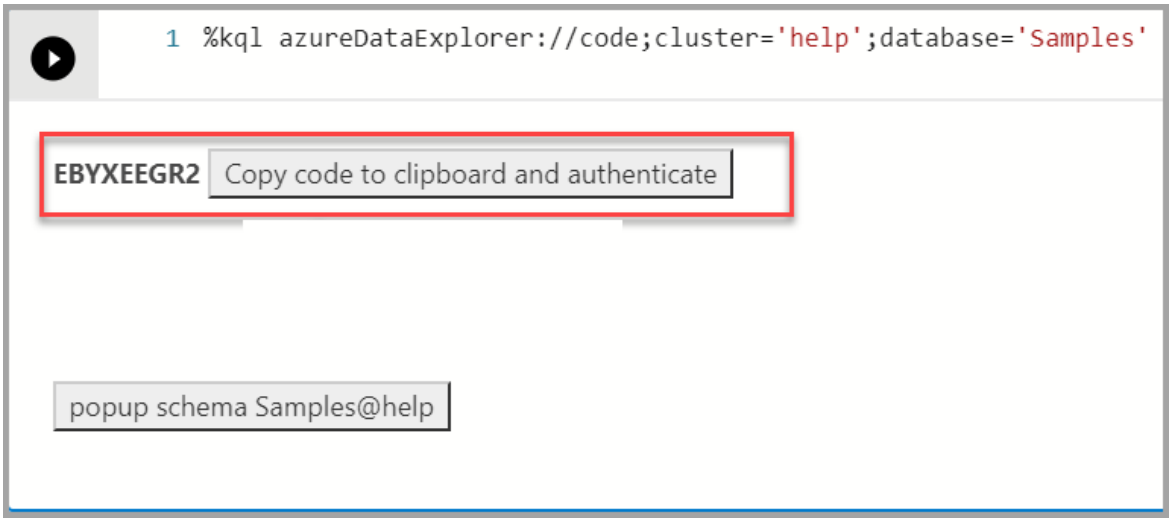


3. Connect to the cluster and authenticate:

```
%kql azureDataExplorer://code;cluster='help';database='Samples'
```

> **NOTE**
>
> If you are using your own ADX cluster, you must include the region in the connection string as follows:
> `%kql azuredataexplorer://code;cluster='mycluster.westus';database='mykustodb'`

You use Device Login to authenticate. Copy the code from the output and select **authenticate** which opens a browser where you need to paste the code. Once you authenticate successfully, you can come back to Azure Data Studio to continue with the rest of the script.



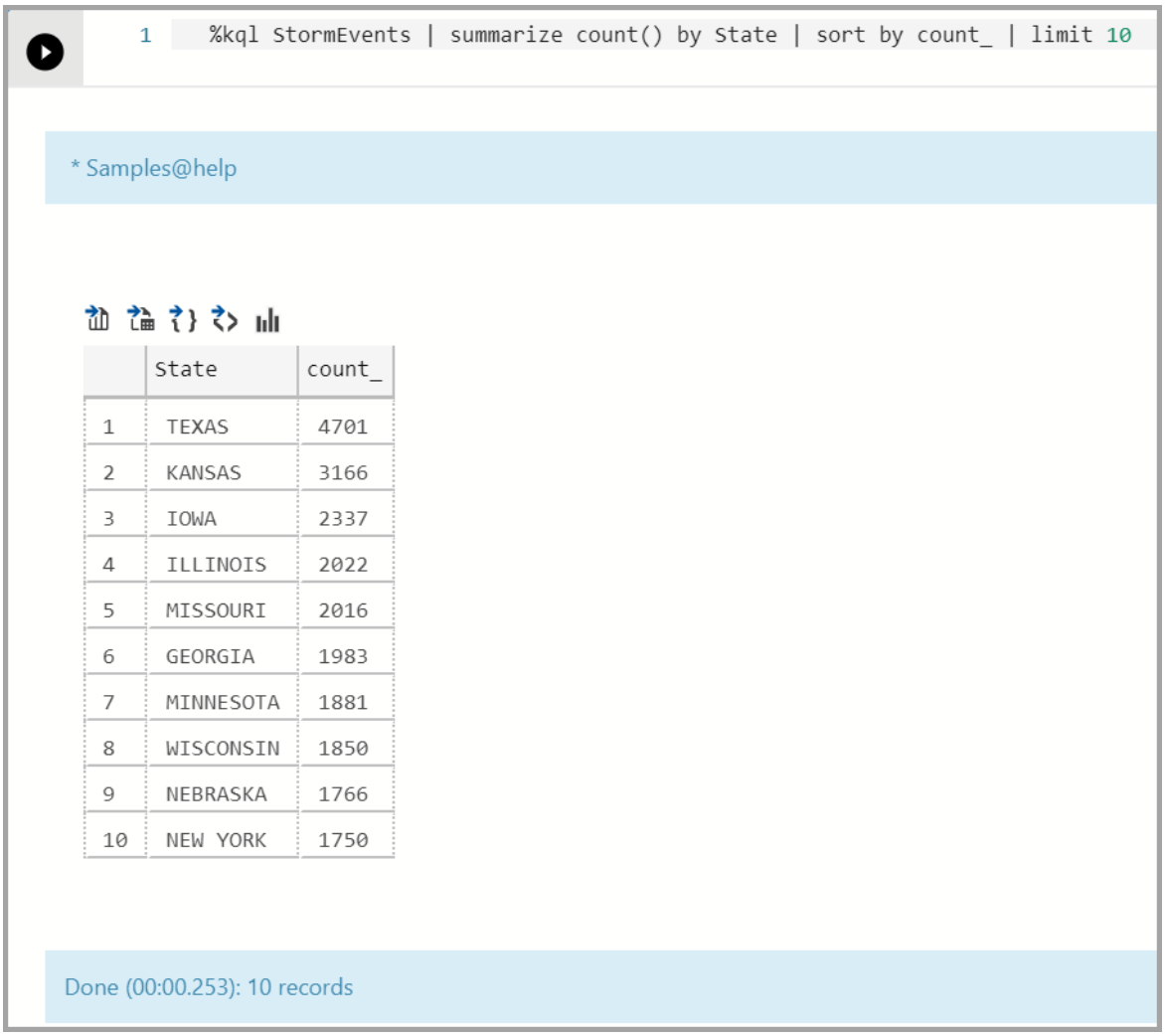
Query and visualize for Azure Data Explorer

Query data using the render operator and visualize data using the ploy.ly library. This query and visualization supplies an integrated experience that uses native KQL.

1. Analyze top 10 storm events by state and frequency:

```
%kql StormEvents | summarize count() by State | sort by count_ | limit 10
```

If you're familiar with the Kusto Query Language (KQL), you can type the query after `%kql`.

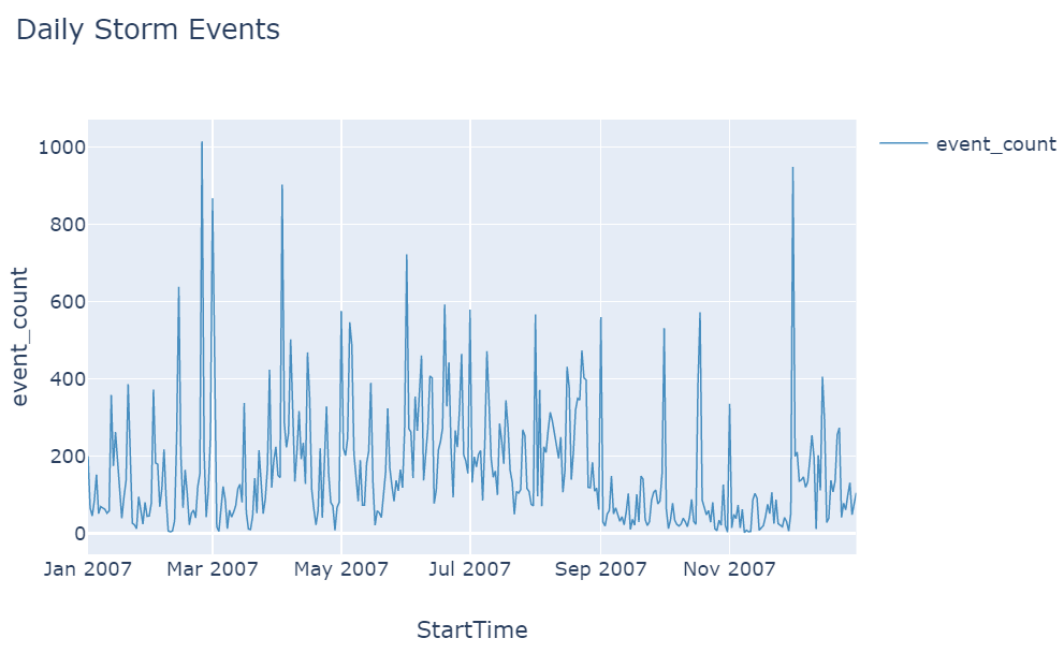


2. Visualize a timeline chart:

```
%kql StormEvents \
| summarize event_count=count() by bin(StartTime, 1d) \
| render timechart title= 'Daily Storm Events'
```

```
%kql StormEvents \
| summarize event_count=count() by bin(StartTime, 1d) \
| render timechart title= 'Daily Storm Events'
```

```
* Samples@help
```



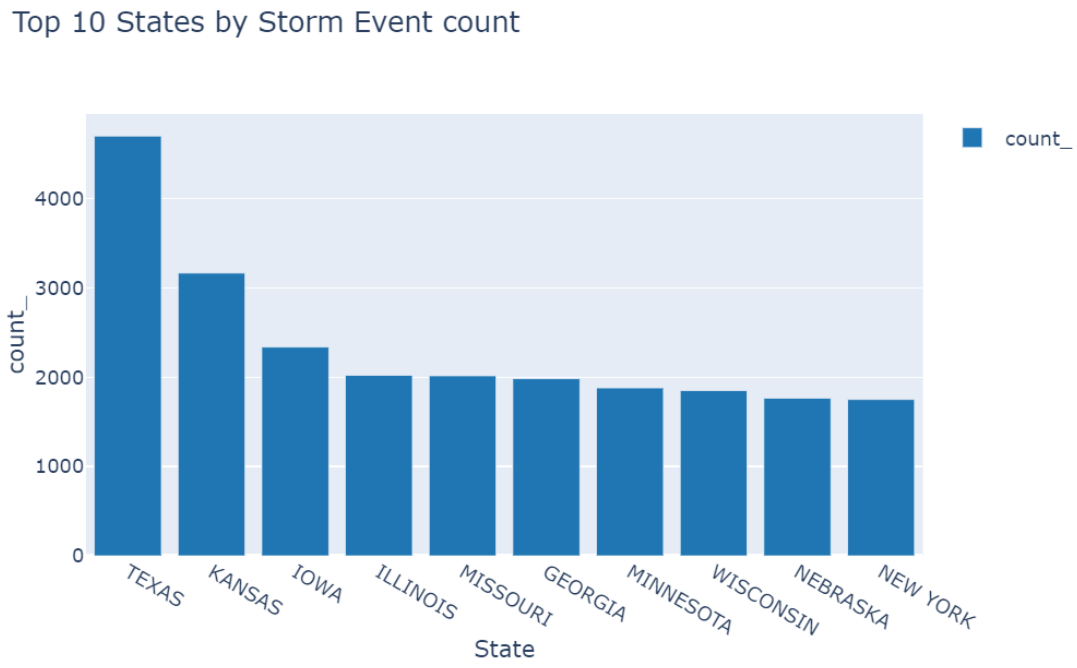

```
Done (00:00.251): 365 records
```

3. Multiline Query sample using `%%kql`.

```
%%kql
StormEvents
| summarize count() by State
| sort by count_
| limit 10
| render columnchart title='Top 10 States by Storm Event count'
```

```
%%kql
    StormEvents
    | summarize count() by State
    | sort by count_
    | limit 10
    | render columnchart title='Top 10 States by Storm Event count'
```

* Samples@help

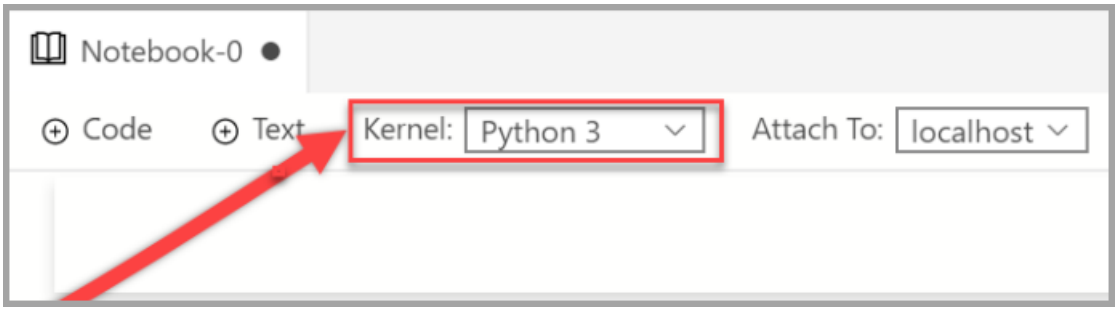


Done (00:01.199): 10 records

Kqlmagic with Application Insights

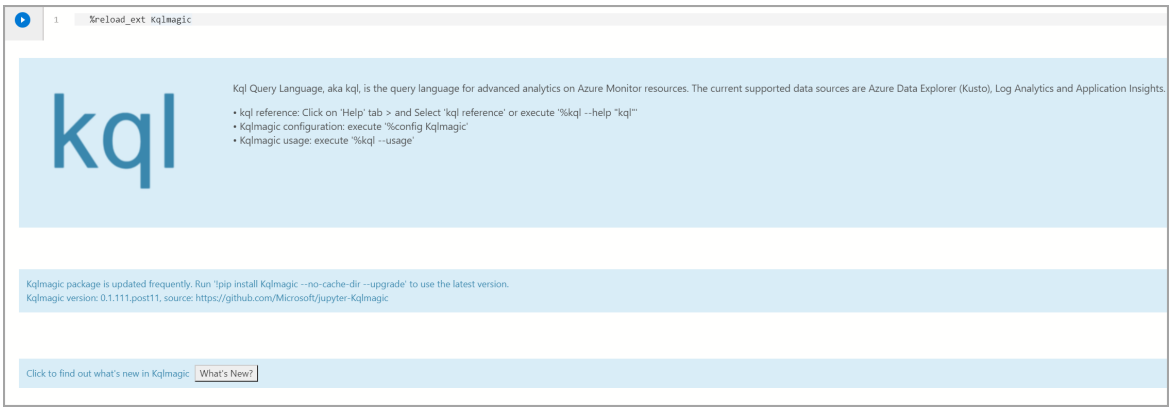
Load and authenticate Kqlmagic for Application Insights

1. Verify the **Kernel** is set to *Python3*.



2. Load Kqlmagic:

```
%reload_ext Kqlmagic
```



> **NOTE**
>
> Every time you create a new notebook in Azure Data Studio you must load the Kqlmagic extension.

3. Connect and authenticate.

   First, you must generate an API key for your Application Insights resource. Then, use the Application ID and API key to connect to Application Insights from the notebook:

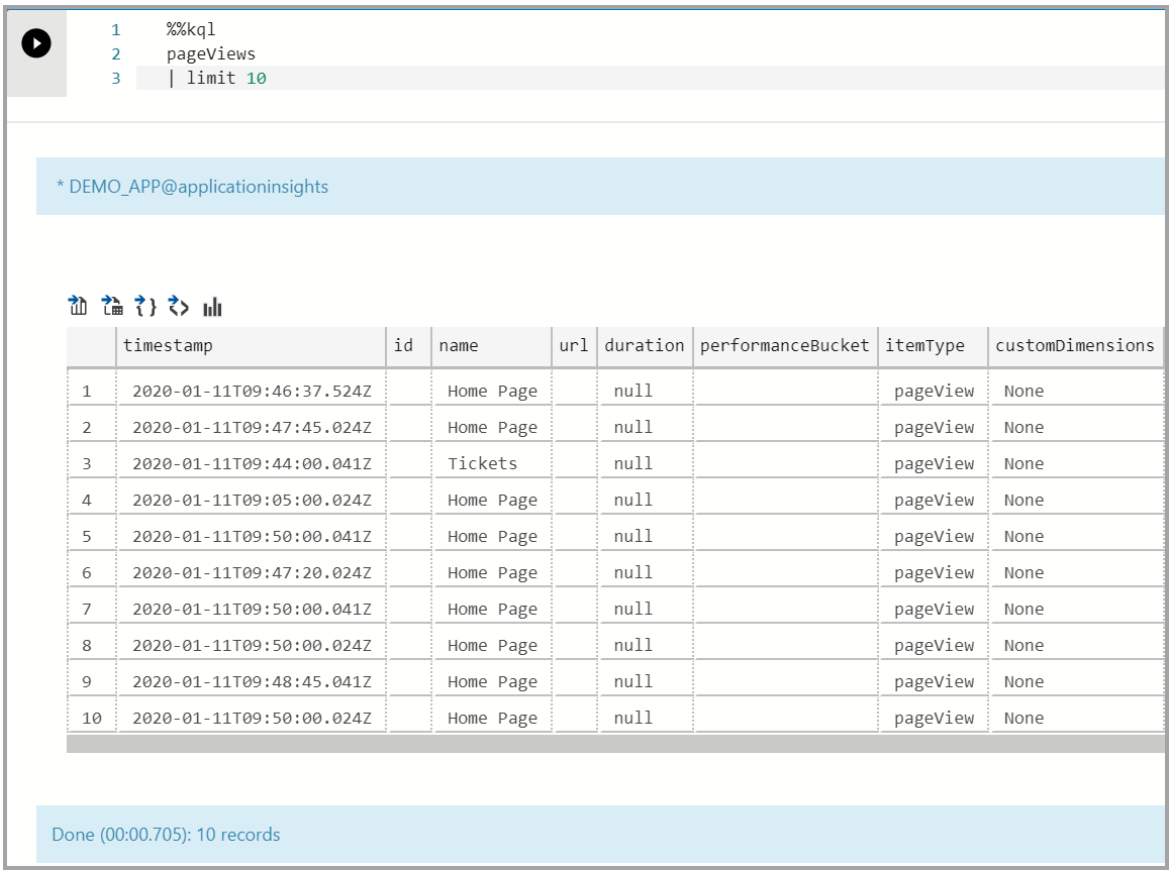
```
%kql appinsights://appid='DEMO_APP';appkey='DEMO_KEY'
```

Query and visualize for Application Insights

Query data using the render operator and visualize data using the ploy.ly library. This query and visualization supplies an integrated experience that uses native KQL.

1. Show Page Views:

```
%%kql
pageViews
| limit 10
```



```
%%kql
pageViews
| limit 10
```

* DEMO_APP@applicationinsights

	timestamp	id	name	url	duration	performanceBucket	itemType	customDimensions
1	2020-01-11T09:46:37.524Z		Home Page		null		pageView	None
2	2020-01-11T09:47:45.024Z		Home Page		null		pageView	None
3	2020-01-11T09:44:00.041Z		Tickets		null		pageView	None
4	2020-01-11T09:05:00.024Z		Home Page		null		pageView	None
5	2020-01-11T09:50:00.041Z		Home Page		null		pageView	None
6	2020-01-11T09:47:20.024Z		Home Page		null		pageView	None
7	2020-01-11T09:50:00.041Z		Home Page		null		pageView	None
8	2020-01-11T09:50:00.024Z		Home Page		null		pageView	None
9	2020-01-11T09:48:45.041Z		Home Page		null		pageView	None
10	2020-01-11T09:50:00.024Z		Home Page		null		pageView	None

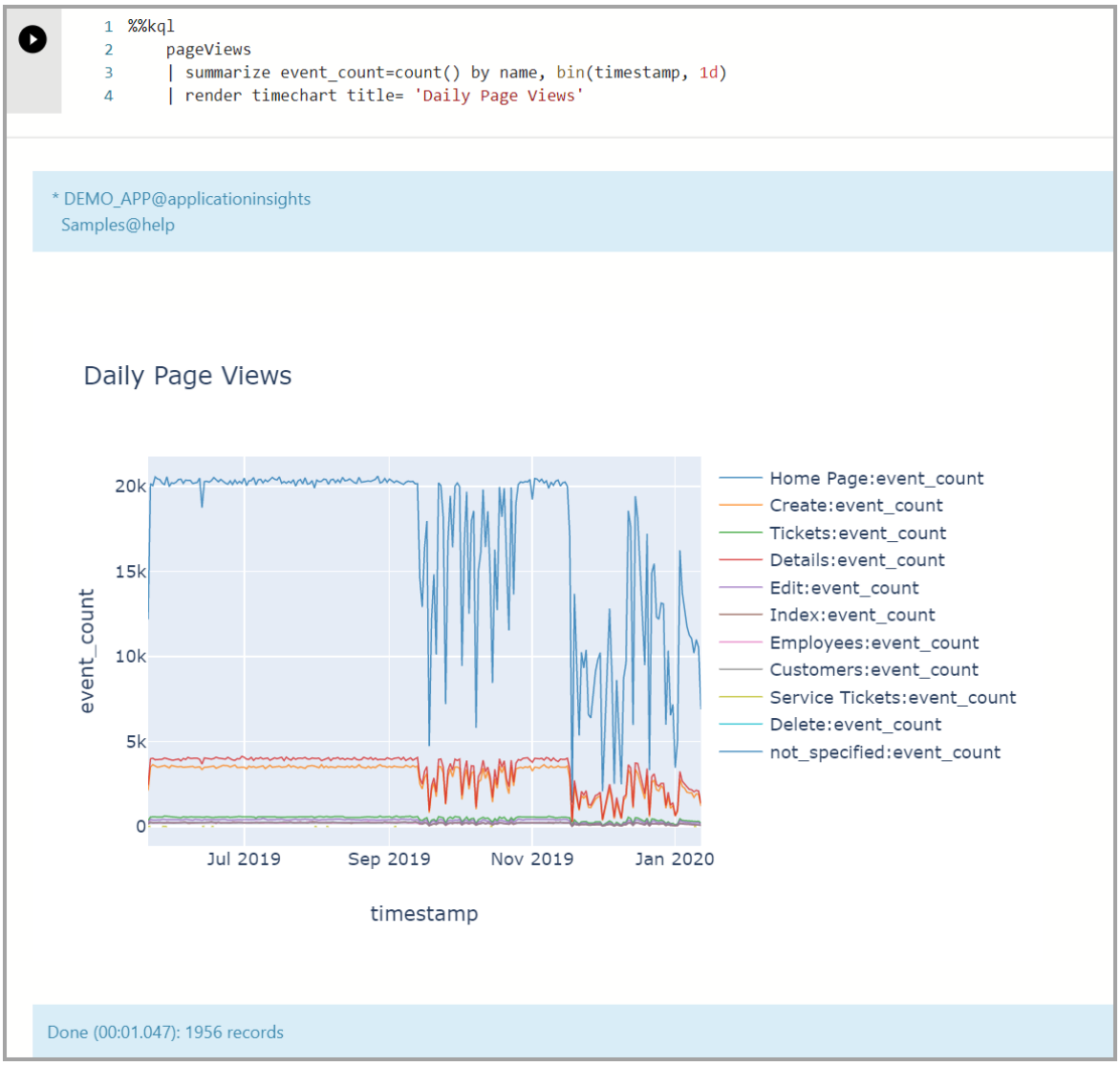
Done (00:00.705): 10 records

NOTE

Use your mouse to drag on an area of the chart to zoom in to the specific date(s).

2. Show Page views in a timeline chart:

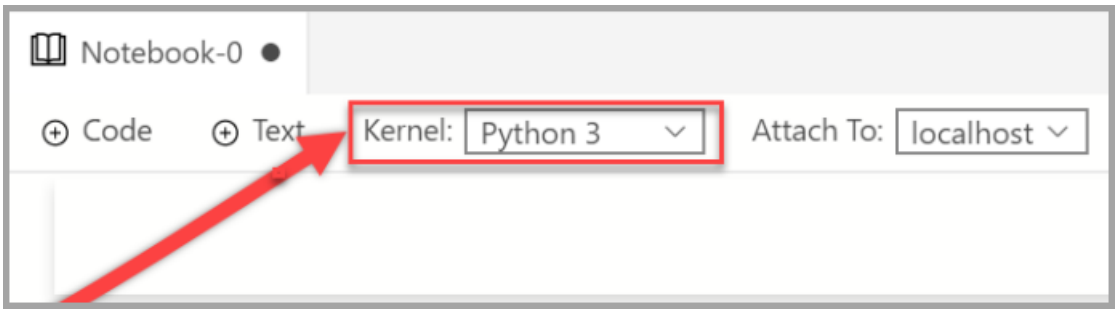
```
%%kql
pageViews
| summarize event_count=count() by name, bin(timestamp, 1d)
| render timechart title= 'Daily Page Views'
```



Kqlmagic with Azure Monitor logs

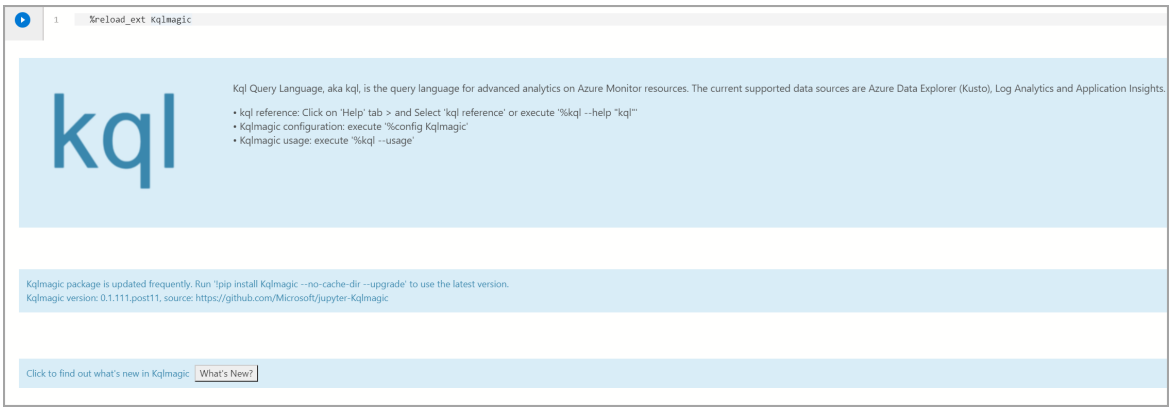
Load and authenticate Kqlmagic for Azure Monitor logs

1. Verify the **Kernel** is set to *Python3*.



2. Load Kqlmagic:

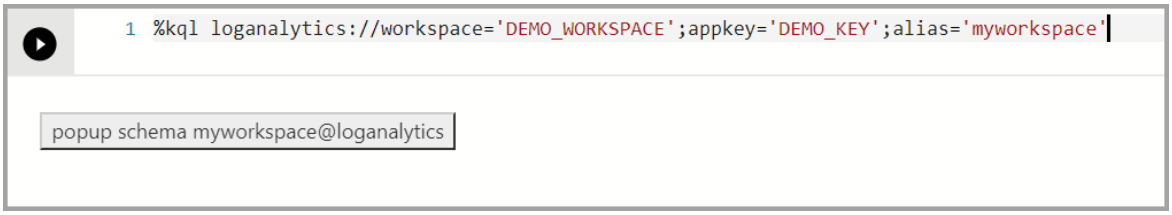
```
%reload_ext Kqlmagic
```



> **NOTE**
>
> Every time you create a new notebook in Azure Data Studio you must load the Kqlmagic extension.

3. Connect and authenticate:

```
%kql loganalytics://workspace='DEMO_WORKSPACE';appkey='DEMO_KEY';alias='myworkspace'
```



Query and visualize for Azure Monitor Logs

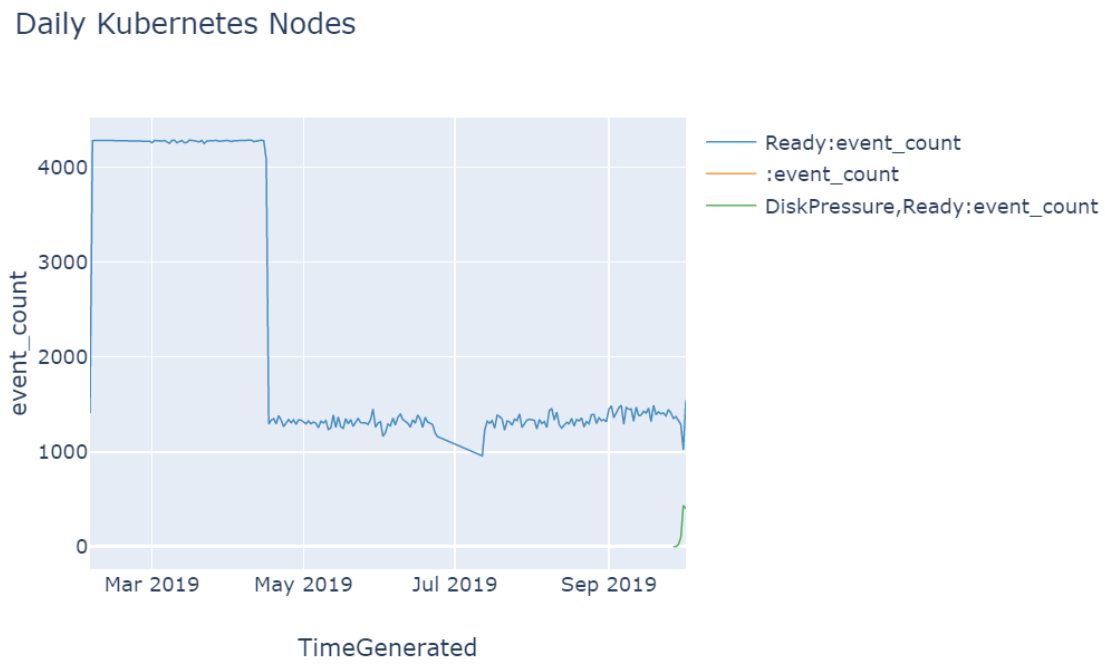
Query data using the render operator and visualize data using the ploy.ly library. This query and visualization supplies an integrated experience that uses native KQL.

1. View a timeline chart:

```
%%kql
KubeNodeInventory
| summarize event_count=count() by Status, bin(TimeGenerated, 1d)
| render timechart title= 'Daily Kubernetes Nodes'
```

```
%%kql
    KubeNodeInventory
    | summarize event_count=count() by Status, bin(TimeGenerated, 1d)
    | render timechart title= 'Daily Kubernetes Nodes'
```

```
DEMO_APP@applicationinsights
Samples@help
* myworkspace@loganalytics
```




```
Done (00:00.840): 234 records
```

Next steps

Learn more about notebooks and Kqlmagic:

- Kusto (KQL) extension for Azure Data Studio (Preview)
- Create and run a Kusto (KQL) notebook (Preview)
- Use a Jupyter Notebook and Kqlmagic extension to analyze data in Azure Data Explorer
- Extension (Magic) to Jupyter Notebook and Jupyter lab, that enable notebook experience working with Kusto, Application Insights, and LogAnalytics data.
- Kqlmagic
- How to use notebooks in Azure Data Studio

Create a Parameterized Notebook with Papermill

6/30/2021 • 2 minutes to read • Edit Online

Parameterization is the ability to execute the same notebook with different parameters.

This article shows you how to create and run a parameterized notebook in Azure Data Studio using the python kernel.

> **NOTE**
> Currently parameterization can be used with Python, PySpark, PowerShell, and .Net Interactive Kernels.

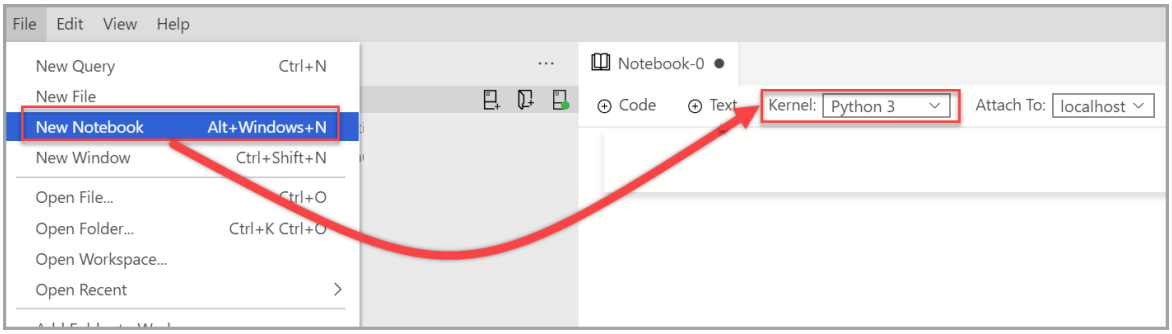
Prerequisites

- Azure Data Studio
- Python

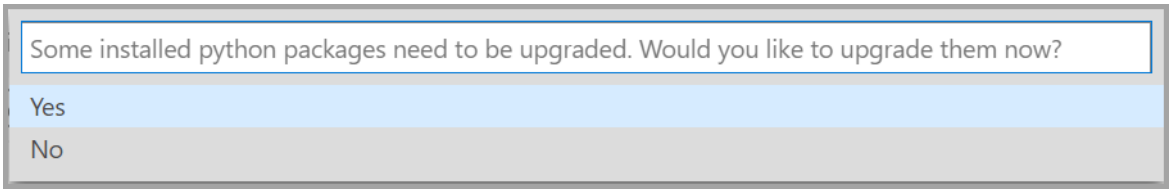
Install and set up Papermill in Azure Data Studio

The steps in this section all run within an Azure Data Studio notebook.

1. Create a new notebook and change the **Kernel** to *Python 3*.



2. You may be prompted to upgrade your Python packages when your packages need updating.



3. Install Papermill:

```
import sys
!{sys.executable} -m pip install papermill --no-cache-dir --upgrade
```

Verify it's installed:

```
import sys
!{sys.executable} -m pip list
```

```
oset                    0.1.3
packaging               20.4
pandas                  1.1.4
pandocfilters           1.4.3
papermill               2.2.2
paramiko                2.7.2
parso                   0.7.1
pathspec                0.8.1
pexpect                 4.8.0
```

4. You can test if Papermill is loaded properly by checking the version of Papermill.

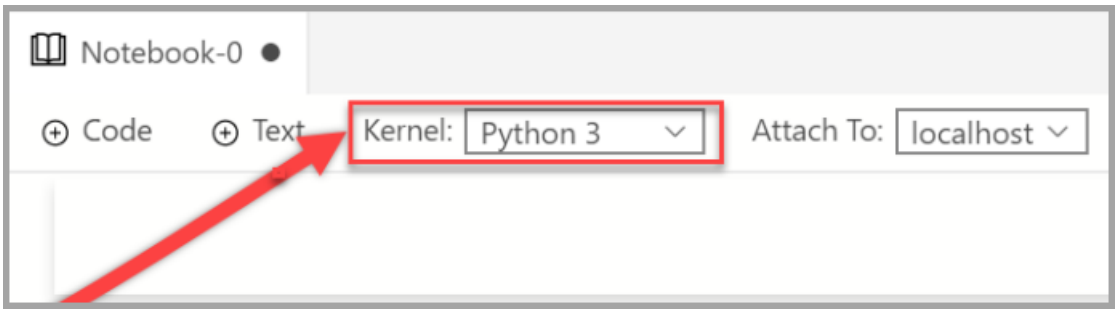
```
import papermill
papermill
```

```
<module 'papermill' from
'/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-
packages/papermill/__init__.py'>
```

Set up a parameterized notebook

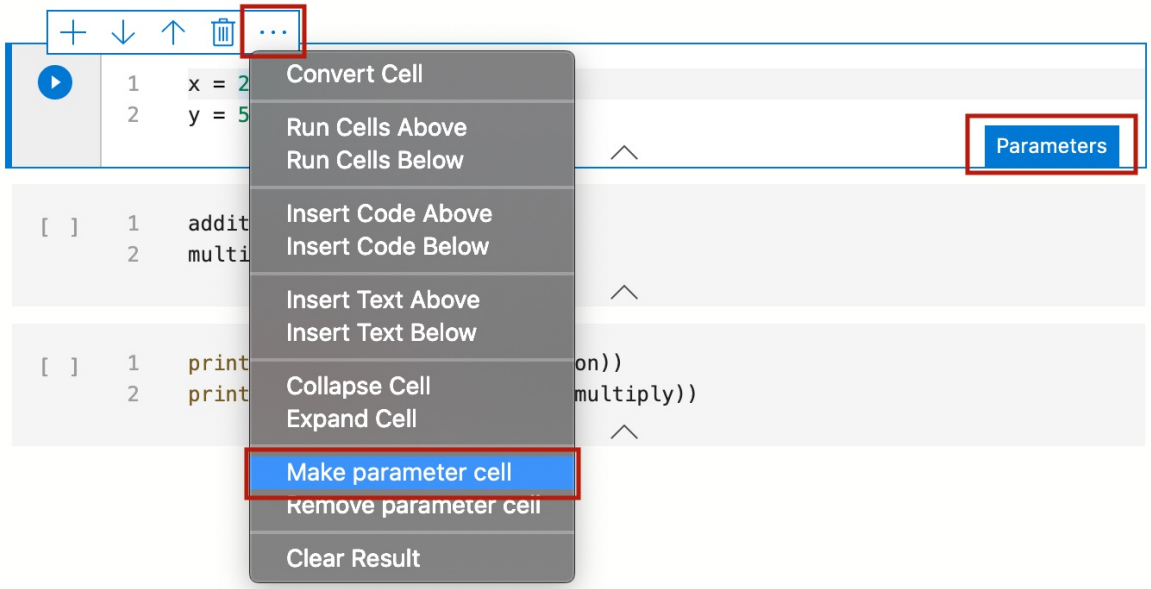
To open the below notebook example in Azure Data Studio, visit [GitHub] (https://github.com/microsoft/sql-server-samples/blob/master/samples/applications/azure-data-studio/parameterization.ipynb and follow along.

1. Verify the **Kernel** is set to *Python3*.



2. Create a New Code Cell and Tag as **Parameters Cell**.

```
x = 2.0
y = 5.0
```



3. Add other cells to test different parameters.

```
addition = x + y
multiply = x * y
```

```
print("Addition: " + str(addition))
print("Multiplication: " + str(multiply))
```

Cells in Example Input Notebook:

```
x = 2.0
y = 5.0
```

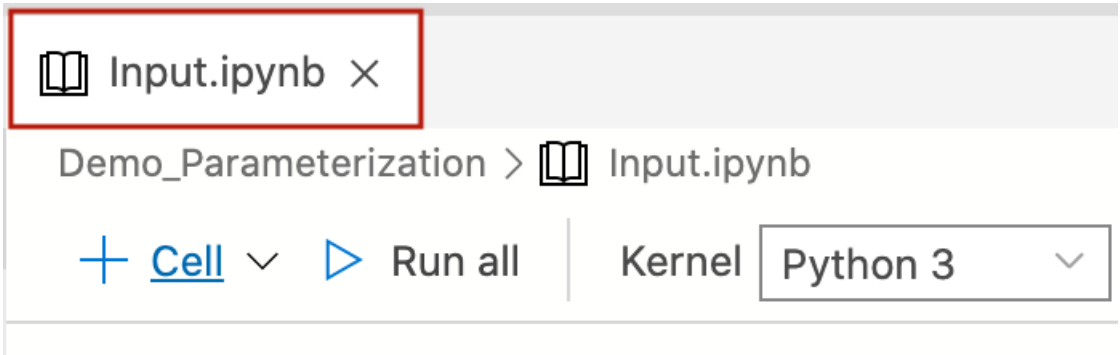
Parameters

```
addition = x + y
multiply = x * y
```

```
print("Addition: " + str(addition))
print("Multiplication: " + str(multiply))
```

```
Addition: 7.0
Multiplication: 10.0
```

4. Save notebook as **Input.ipynb**.



How to execute Papermill notebook

Papermill can be executed two ways:

- Command Line Interface (CLI)
- Python API

Parameterized CLI execution

To execute a notebook using the CLI, enter the papermill command in the terminal with the input notebook, location for output notebook, and options.

> **NOTE**
> Papermill Command Line Interface Documentation can be found here.

1. Execute Input Notebook with new parameters.

   ```
   papermill Input.ipynb Output.ipynb -p x 10 -p y 20
   ```

   This executes the Input Notebook with new values for parameters **x** and **y**.

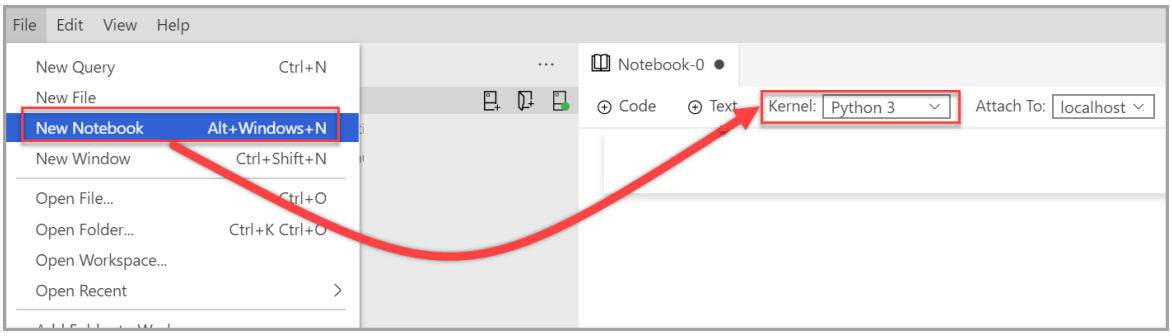
2. After execution view the new output parameterized notebook.

   You can note that there's a new cell labeled **# Injected-Parameters** containing the new parameter values passed in via CLI.

Parameterized Python API execution

> **NOTE**
> Papermill Python API Documentation can be found here.

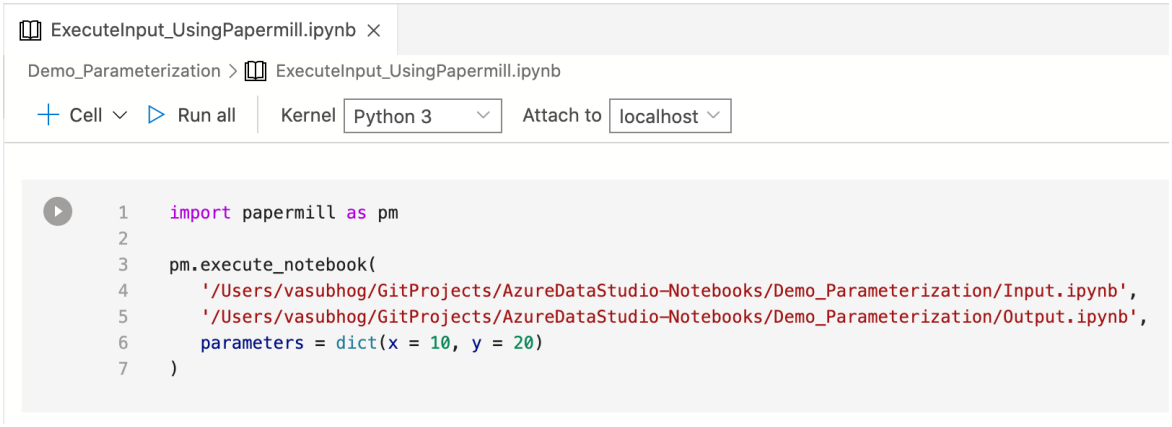
1. Create a new notebook and change the **Kernel** to *Python 3*.



2. Add a new code cell and use papermill to use the execute method.

```
import papermill as pm

pm.execute_notebook(
'/Users/vasubhog/GitProjects/AzureDataStudio-Notebooks/Demo_Parameterization/Input.ipynb',
'/Users/vasubhog/GitProjects/AzureDataStudio-Notebooks/Demo_Parameterization/Output.ipynb',
parameters = dict(x = 10, y = 20)
)
```



3. After execution view the new output parameterized notebook.

   You can note that there's a new cell labeled **# Injected-Parameters** containing the new parameter values passed in via CLI.

```
[1]  1  x = 2.0
     2  y = 5.0
```

Parameters

```
[2]  1  # Injected-Parameters
     2  x = 10
     3  y = 20
     4
```

```
     1  addition = x + y
     2  multiply = x * y
```

```
[4]  1  print("Addition: " + str(addition))
     2  print("Multiplication: " + str(multiply))
```

```
Addition: 30
Multiplication: 200
```

Next steps

Learn more about notebooks and Parameterization:

- How to use notebooks in Azure Data Studio
- Papermill Parameterization Docs
- URI Parameterization
- Run with Parameters

Create a Parameterized Notebook with the Notebook URI

6/30/2021 • 2 minutes to read • Edit Online

Parameterization is the ability to execute the same notebook with different parameters.

This article shows you how to create and run a parameterized notebook in Azure Data Studio with the python kernel.

> **NOTE**
> Currently parameterization can be used with Python, PySpark, PowerShell, and .Net Interactive Kernels.

Prerequisites

- Azure Data Studio
- Python

URI Parameterization

URI parameterization programmatically adds parameters to the query of the ADS URI to open the notebook in ADS with new parameters.

Azure Data Studio Notebook URI supports HTTPS/HTTP/FILE URI schema and follows the format:

```
_azuredatastudio:\/\/microsoft.notebook\/open?url=_
```

The format to pass in parameters with the ADS Notebook URI is as follows:

```
_azuredatastudio:\/\/microsoft.notebook\/open?url=LinkToNotebook_***?x=1&y=2***
```

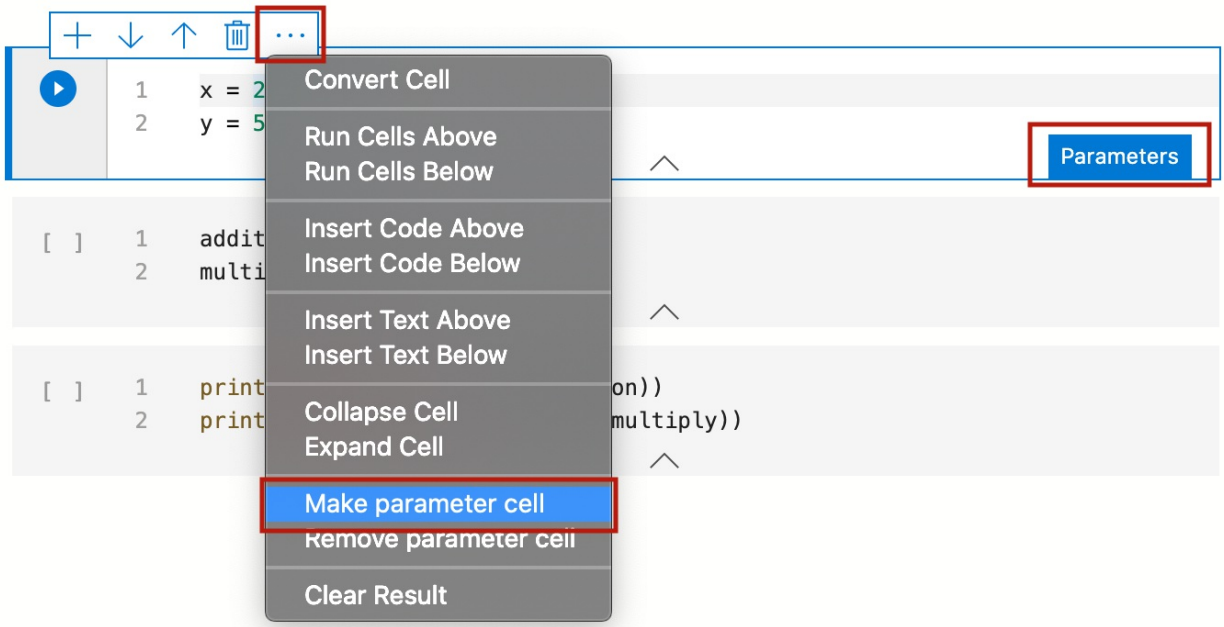
In the URI query, use & to indicate a new parameter to be injected.

URI Parameterization Example

To open the below notebook example in Azure Data Studio, visit GitHub and follow along.

Below is the contents and structure of the notebook, you must use a notebook that has a cell tagged with parameters.

Tag a code cell in Azure Data Studio as **Parameters Cell**.



Below is the contents of the notebook:

```
x = 2.0
y = 5.0
```

```
addition = x + y
multiply = x * y
```

```
print("Addition: " + str(addition))
print("Multiplication: " + str(multiply))
```

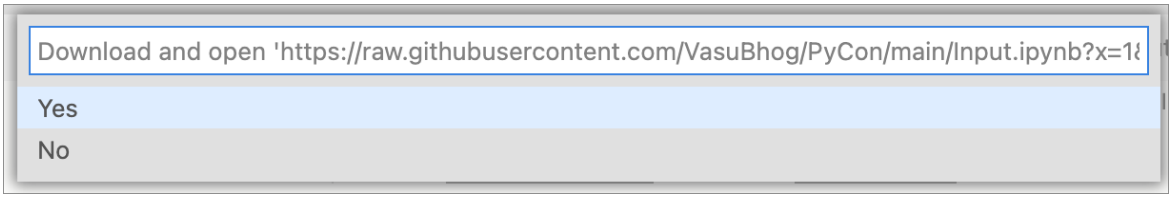
1. We can either use the search bar of any browser or a markdown cell to open up the notebook URI link.

   Below is the notebook URI for parameterizing the notebook with new x and y values:

   ```
   _azuredatastudio://microsoft.notebook/open?url=https://raw.githubusercontent.com/microsoft/sql-server-samples/master/samples/applications/azure-data-studio/parameterization.ipynb_**?x=10&y=20**
   ```

   When opening the link from the web browser, you're prompted to open the notebook in Azure Data Studio. Select **Open Azure Data Studio**.

2. Then you're prompted to download and open the notebook with new parameters.

   Once you select **Yes**, view the new parameterized notebook and **run all cells** to see the new output.

   You can note that there's a new cell labeled **# Injected-Parameters** containing the new parameter values passed in.

Next steps

Learn more about notebooks and Parameterization:

- How to use notebooks in Azure Data Studio
- Papermill Parameterization
- Run with Parameters

Create a Parameterized Notebook using Run with Parameters Action

6/30/2021 • 2 minutes to read • Edit Online

Parameterization is the ability to execute the same notebook with different parameters.

This article shows you how to create and run a parameterized notebook in Azure Data Studio with the python kernel.

> **NOTE**
> Currently parameterization can be used with Python, PySpark, PowerShell, and .Net Interactive Kernels.

Prerequisites

- Azure Data Studio
- Python

Run with Parameters Action

The `Run with Parameters` notebook action enables users to quickly set new parameters for their notebook by allowing the user to input new parameters from the UI.

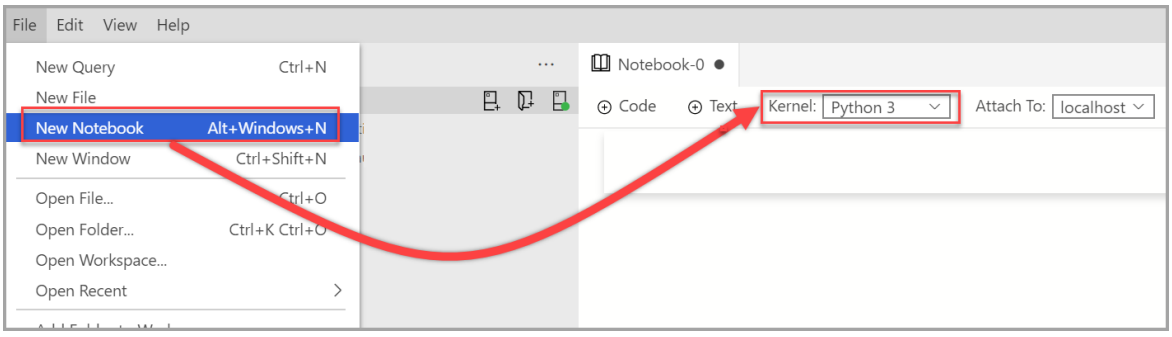
> **NOTE**
> The parameter cell has to be formatted with each new parameter on a new line.

Set up a notebook for parameterization in Azure Data Studio

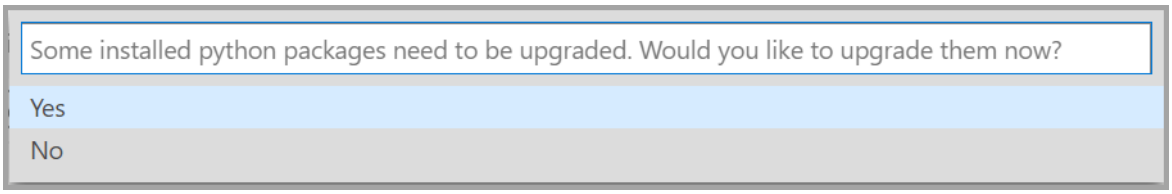
To open the below notebook example in Azure Data Studio, visit [GitHub] (https://github.com/microsoft/sql-server-samples/blob/master/samples/applications/azure-data-studio/parameterization.ipynb and follow along.

The steps in this section all run within an Azure Data Studio notebook.

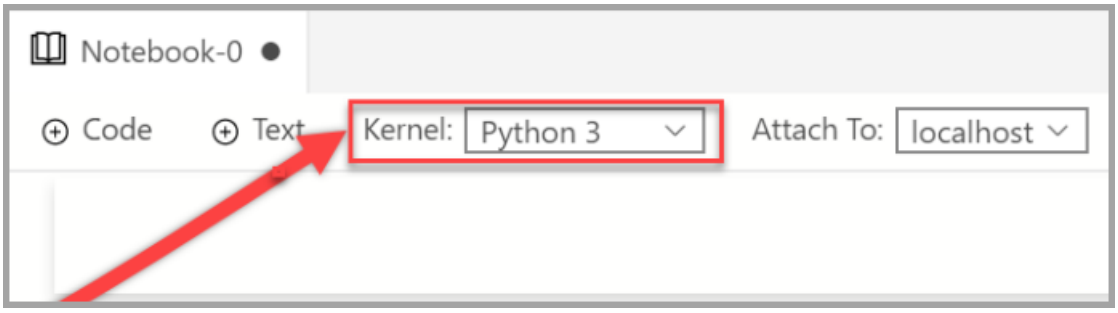
1. Create a new notebook and change the **Kernel** to **Python 3**.



2. You may be prompted to upgrade your Python packages when your packages need updating.

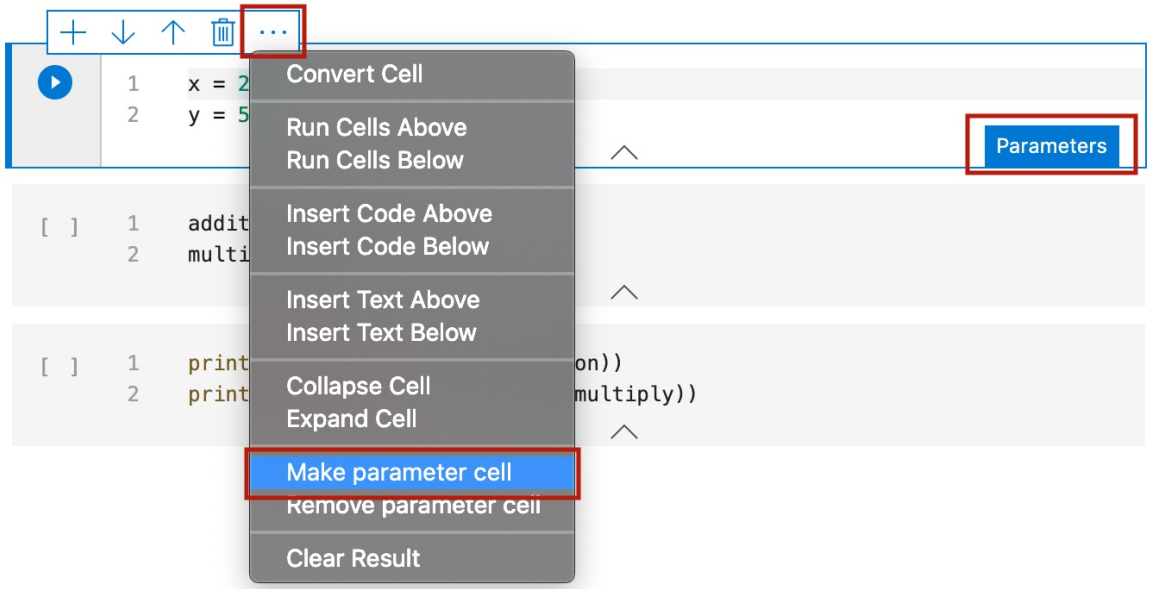


3. Verify the **Kernel** is set to **Python3**.



4. Create a New Code Cell and Tag as **Parameters Cell**.

```
x = 2.0
y = 5.0
```



5. Add other cells to test different parameters.

```
addition = x + y
multiply = x * y
```

```
print("Addition: " + str(addition))
print("Multiplication: " + str(multiply))
```

Cells in Example Input Notebook:

```
[1]  1  x = 2.0
     2  y = 5.0
```

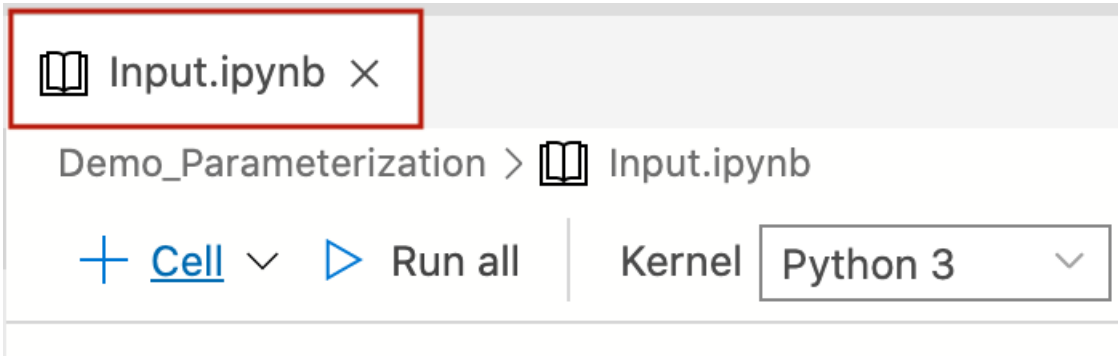
Parameters

```
[2]  1  addition = x + y
     2  multiply = x * y
```

```
[3]  1  print("Addition: " + str(addition))
     2  print("Multiplication: " + str(multiply))
```

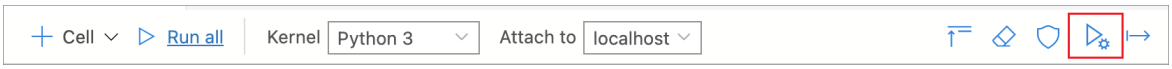
```
Addition: 7.0
Multiplication: 10.0
```

6. Save notebook as **Input.ipynb**.

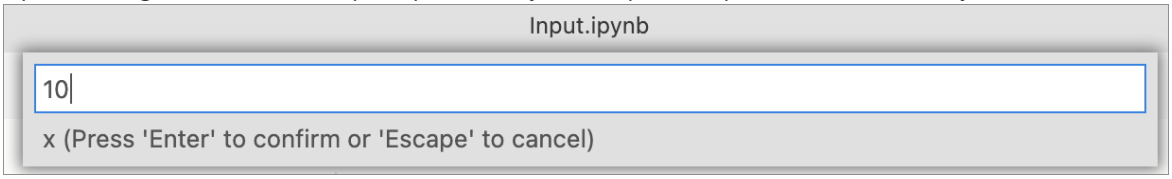


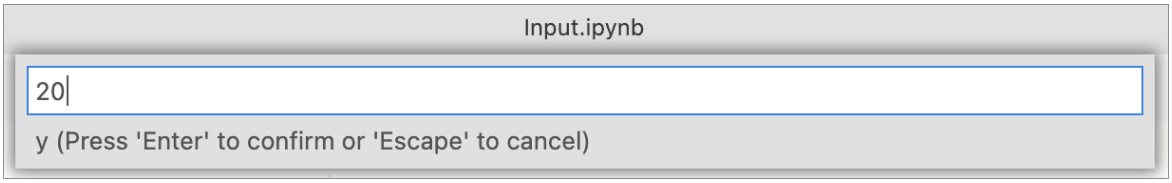
How to Run the Notebook with Parameters

1. On the notebook toolbar, select the `Run with Parameters` action.



2. Upon clicking the action, a new prompt will ask you to input new parameters for x and y.





3. After, entering the new parameters view the new parameterized notebook and run all cells to see the new output.

   You can note that there's a new cell labeled **# Injected-Parameters** containing the new parameter values passed in.

Next steps

Learn more about notebooks and parameterization:

- How to use notebooks in Azure Data Studio
- Papermill Parameterization
- URI Parameterization

Create an Azure SQL Database using Azure Data Studio

11/2/2020 • 2 minutes to read • Edit Online

You can create an Azure SQL Database using Azure Data Studio through the deployment wizard and notebooks.

Pre-requisites

- Azure Data Studio is installed
- An active Azure account and subscription. If you do not have one, create a free account.

Use the deployment wizard

Follow these steps to use the deployment wizard, which will guide you through the required settings in a simple UI experience.

First, find and select Azure SQL Database in the deployment wizard.

1. In Azure Data Studio, select the Connections viewlet on the left-side navigation.
2. Select the **...** button at the top of the Connections panel and choose **New Deployment...**
3. In the deployment wizard, select the **Azure SQL Database** tile and check the terms acceptance checkbox
4. In the Resource type dropdown, select what you would like to create - either a database, database server, or elastic pool. If you don't have any SQL databases in Azure, we recommend starting by creating a database.
5. Select **Create in Azure portal** if you chose to create a server or pool or select **Select** if you chose to create a database.

If you chose to create a database, follow the steps below.

1. Sign in to your Azure account if you have not already. You can refresh your connection if you have issues on this page of the wizard.
2. Select your desired subscription and server. Then select **Next**.
3. Enter a database name.
4. Enter a firewall rule name and the IP range of your local client machine running Azure Data Studio. This is so you can connect to the server and database from ADS right after it has been created.
5. Select **Next**.
6. Review the parameters you have entered and then select **Script to Notebook**.

Once the Notebook opens, you can review the content and the code and make changes if you like. In particular you can change the compute and storage settings from the default if you have specific performance or cost preferences. However, be aware that any changes you make to the Notebook could potentially cause validation errors.

The last step is to select **Run all** to run all cells in the Notebook. Once this completes you should have a fully created and running SQL Database that you can connect to and query from ADS.

Next steps

After you've created your database, server, or pool, you can connect and query your database in Azure Data Studio.

Deploy Azure SQL Edge with Azure Data Studio (Preview)

11/2/2020 • 3 minutes to read • Edit Online

Azure SQL Edge is a relational database engine optimized for IoT and Azure IoT Edge deployments. It provides capabilities to create a high-performance data storage and processing layer for IoT applications and solutions. This article shows you how to deploy an Azure SQL Edge instance with Azure Data Studio and the deployment scenarios that are supported with the deployment wizard.

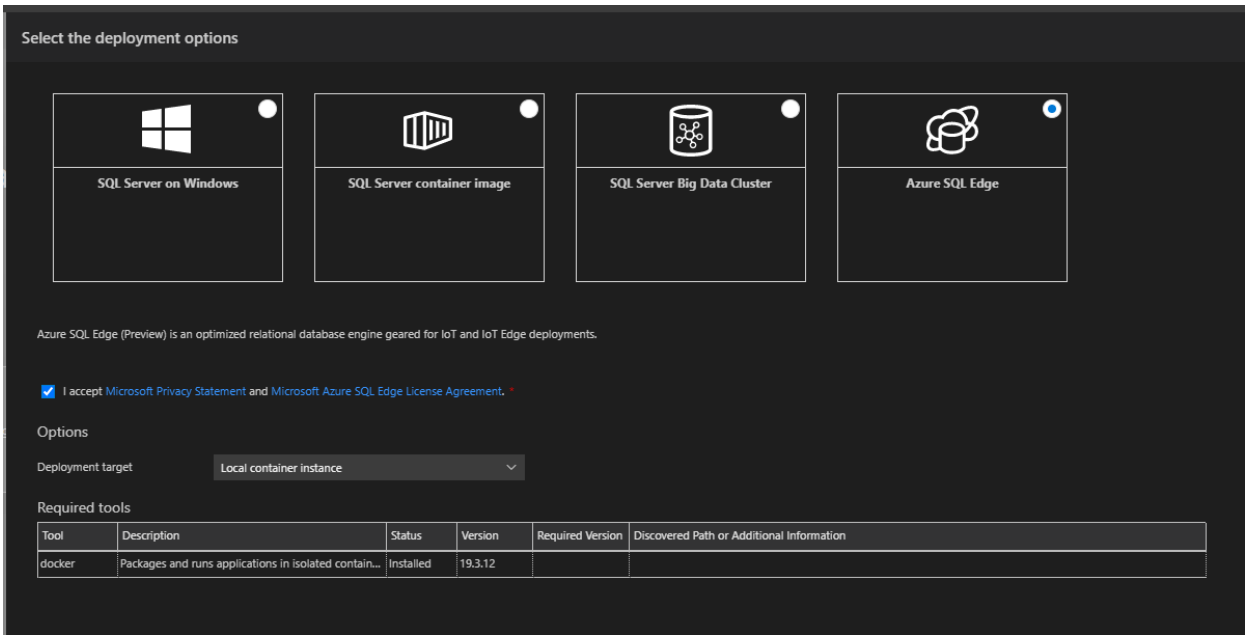
The following scenarios are supported by the deployment wizard in Azure Data Studio:

- Local container instance
- Remote container instance
- New Azure IoT Hub and VM
- Existing Device of an Azure IoT Hub
- Multiple Devices of an Azure IoT Hub

REQUIRED TOOLS	`DOCKER`	`AZURE CLI`
Local container instance	x	
Remote container instance		
New Azure IoT Hub and VM		x
Existing Device of an Azure IoT Hub		x
Multiple Devices of an Azure IoT Hub		x

> **NOTE**
> Azure SQL Edge deployment (preview) is available through the extension "Azure SQL Edge Deployment Extension" while these features are in preview. To make the features available to you, please install the extension in Azure Data Studio.

To begin a deployment in Azure Data Studio, open the menu in the **Connections** view and select the **New Deployment...** option. For all Azure SQL Edge scenarios, select **Azure SQL Edge** on the following screen and the desired scenario from the **Deployment target** drop-down. The deployment wizard generates a TSQL notebook that can be executed to complete the deployment. It should be noted that connection information, including the SQL admin password, is contained in the notebook by default.



Local container instance

Azure SQL Edge can be deployed to a Docker container on the localhost by specifying the container name, `sa` user password, and the host port for Azure SQL Edge connectivity. After completing the deployment, several links are available at the bottom of the notebook for further actions:

- **Select here to connect to the Azure SQL Edge instance**: Creates a connection in Azure Data Studio to the new Azure SQL Edge instance
- **Open the terminal window**: Opens a terminal (existing or new) in Azure Data Studio
- **Stop the container**: Copies a command into the terminal that stops the container when executed by the user
- **Remove the container**: Copies a command into the terminal that removes the container when executed by the user

Remote container instance

Azure SQL Edge can be deployed to a Docker container on a remote host with Docker installed by specifying the container information and the target/host machine information. After completing the deployment, a connection link is available at the bottom of the notebook. Because of the remote container host environment, to stop or remove the container, commands must be copied to execute in a remote shell.

New Azure IoT Hub and VM

The Azure SQL Edge deployment wizard can create several Azure resources needed to deploy an edge-enabled virtual machine (VM) connected to an Azure IoT hub. An active Azure subscription is required. This deployment creates:

- Resource group (if current resource group isn't selected)
- Network security group
- Virtual machine
- Static public IP address

Optionally, a dacpac file can be zipped in a folder and deployed to the new Azure SQL Edge instance as a part of the process. If a dacpac file is provided, an Azure Blob Storage account is created in the same resource group.

Existing Device of an Azure IoT Hub

If you have an existing IoT hub and a connected device, Azure SQL Edge can be deployed to the device based on the resource group, IoT hub name, and the device ID. The IP address provided during the deployment wizard is utilized to generate a quick connect link at the bottom of the notebook.

Optionally, a dacpac file can be zipped in a folder and deployed to the new Azure SQL Edge instance as a part of the process. If a dacpac file is provided, an Azure Blob Storage account is created in the same resource group.

Multiple Devices of an Azure IoT Hub

If you have an existing IoT hub and connected devices, Azure SQL Edge can be deployed to the device based on the resource group, IoT hub name, and a target condition to select device(s). The IP address provided during the deployment wizard is utilized to generate a quick connect link at the bottom of the notebook.

Optionally, a dacpac file can be zipped in a folder and deployed to the new Azure SQL Edge instance as a part of the process. If a dacpac file is provided, an Azure Blob Storage account is created in the same resource group.

Next steps

- Learn more about Azure SQL Edge

Create SQL Server on Azure Virtual Machines using Azure Data Studio

3/5/2021 • 2 minutes to read • Edit Online

You can create an SQL virtual machine (VM) using Azure Data Studio through the deployment wizard and notebooks.

Pre-requisites

- Azure Data Studio is installed
- An active Azure account and subscription. If you don't have one, create a free account.

Use the deployment wizard

Follow these steps to use the deployment wizard, which will guide you through the required settings in a simple UI experience.

First, find, and select Azure SQL VM in the deployment wizard.

1. In Azure Data Studio, select the Connections viewlet on the left-side navigation.
2. Select the **...** button at the top of the Connections panel and choose **New Deployment.**
3. In the deployment wizard, select the **Azure SQL VM** tile and check the terms acceptance checkbox
4. If prompted to, install the required tools and then select the **Select** button at the bottom.

Next, enter all of the required parameters in the Azure SQL VM wizard.

1. Sign in to your Azure account if you've ven't already. You can refresh your connection if you've issues on this page of the wizard.
2. Select your desired subscription, resource group, and region. Then select **Next**.
3. Enter a unique Virtual Machine name and your username and password credentials.
4. Select your preferred image, SKU, and version, and then select your preferred VM size. You can learn more about available VM sizes to help you make your selection. Then select **Next**.
5. Either select an existing virtual network from the dropdown, or check the **New virtual network** checkbox to enter a name for a new virtual network.
6. Do the same for selecting or creating a subnet and public IP address.
7. If you would like to connect to your VM via Remote Desktop (RDP), check the **Enable RDP inbound port** checkbox. Then select **Next**.
8. Enter your preferred SQL Server settings. The recommendation for testing out this experience is to set SQL connectivity to **Public (internet)**, enter port 1433, and enable SQL authentication with your preferred username and password. Then select **Next**.
9. Review the parameters you've entered and then select **Script to Notebook**.

Once the Notebook opens, you can review the content and the code and make changes if you like. However making changes isn't recommended since it could cause validation errors.

The last step is to select **Run all** to run all cells in the Notebook. Once this completes, you should have a fully created and running:

- An Azure virtual machine
- A SQL virtual machine
- A Virtual Network, subnet, and public IP address
- A network security group and a network interface
- Access to RDP into the VM
- Access to a variety of SQL Server manageability features to easily control the Backup schedule, patching schedule, the SQL Server edition and licensing, the storage performance configurations, and much more.

Next steps

To learn more about how to migrate your data to the new SQL VM, see the following article.

Migrate a database to a SQL VM

For other information about using SQL Server in Azure, see SQL Server on Azure Virtual Machines and the Frequently Asked Questions.

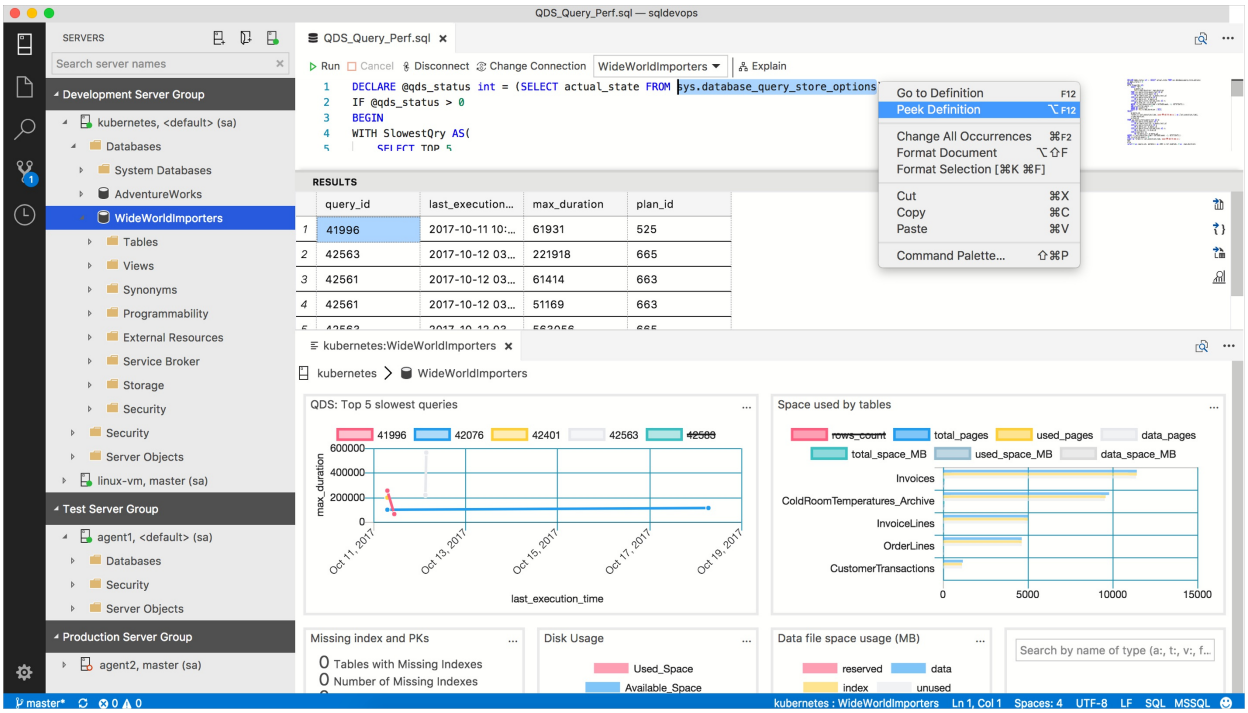
Manage servers and databases with Insight widgets in Azure Data Studio

11/2/2020 • 2 minutes to read • Edit Online

Insight widgets take the Transact-SQL (T-SQL) queries you use to monitor servers & databases and turns them into insightful visualizations.

Insights are customizable charts and graphs that you add to server and database monitoring dashboards. View at-a-glance insights of your servers and databases, then drill into more details, and launch management actions that you define.

You can build awesome server and database management dashboards similar to the following example:



To jump in and start creating different types of insight widgets, check out the following tutorials:

- Build a custom insight widget
- *Enable built-in insight widgets*
  - Enable the performance monitoring insight
  - Enable the table space usage insight

SQL Queries

Azure Data Studio tries to avoid introducing yet another language or heavy user interface so it tries to use T-SQL as much as possible with minimal JSON configuration. Configuring insight widgets with T-SQL leverages the countless number of existing sources of useful T-SQL queries that can be turned into insightful widgets.

Insight widgets are composed of one or two T-SQL queries:

- *Insight widget query* is mandatory, and is the query that returns the data that appears in the widget.
- *Insight details query* is only required if you are creating an insight details page.

An insight widget query defines a dataset that renders a count, chart, or graph. Insight details query is used to

list relevant insight detail information in a tabular format in the insight details panel.

Azure Data Studio executes insight widget queries and maps the query result set to a chart's dataset then renders it. When users open up an insight's details, it executes the insight details query and prints out the result in a grid view within the dialog.

The basic idea is to write a T-SQL query in a way so it can be used as a dataset of a count, chart, and graph widget.

Summary

The T-SQL query and its result set determine the insight widget behavior. Writing a query for a chart type or mapping a right chart type for existing query is the key consideration to build an effective insight widget.

Additional resources

- Query Editor

Create and use code snippets to quickly create Transact-SQL (T-SQL) scripts in Azure Data Studio

3/5/2021 • 2 minutes to read • Edit Online

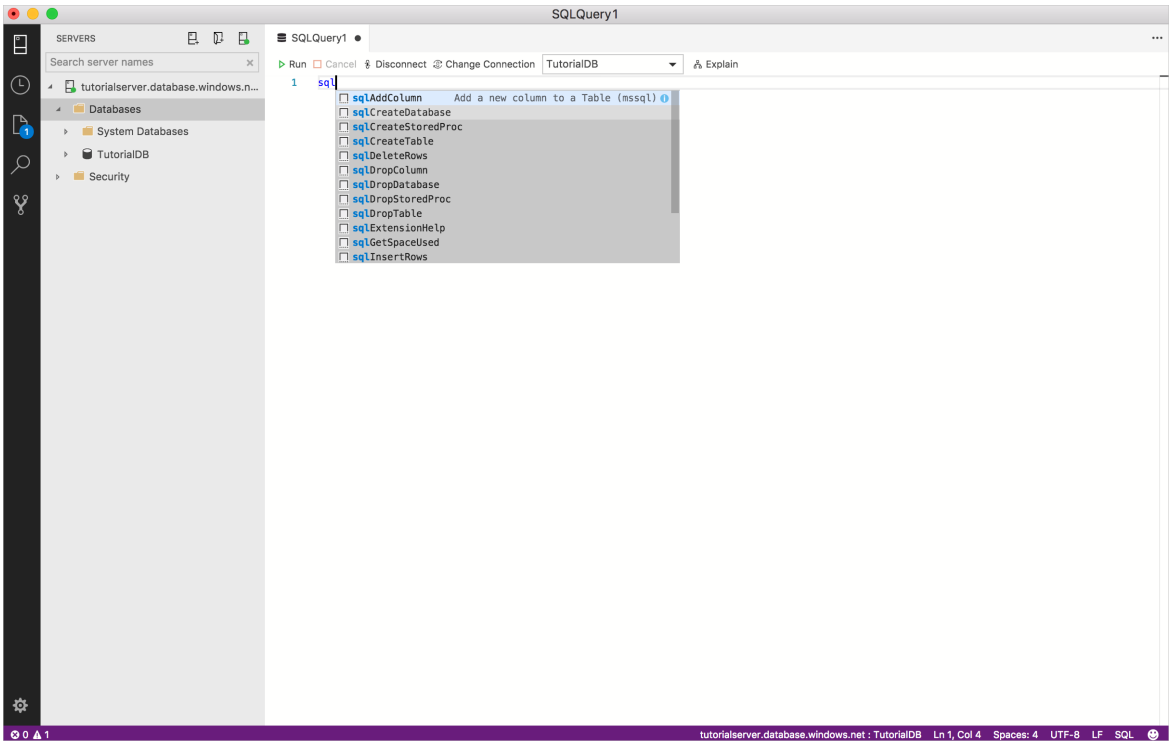
Code snippets in Azure Data Studio are templates that make it easy to create databases and database objects.

Azure Data Studio provides several T-SQL snippets to assist you with quickly generating the proper syntax.

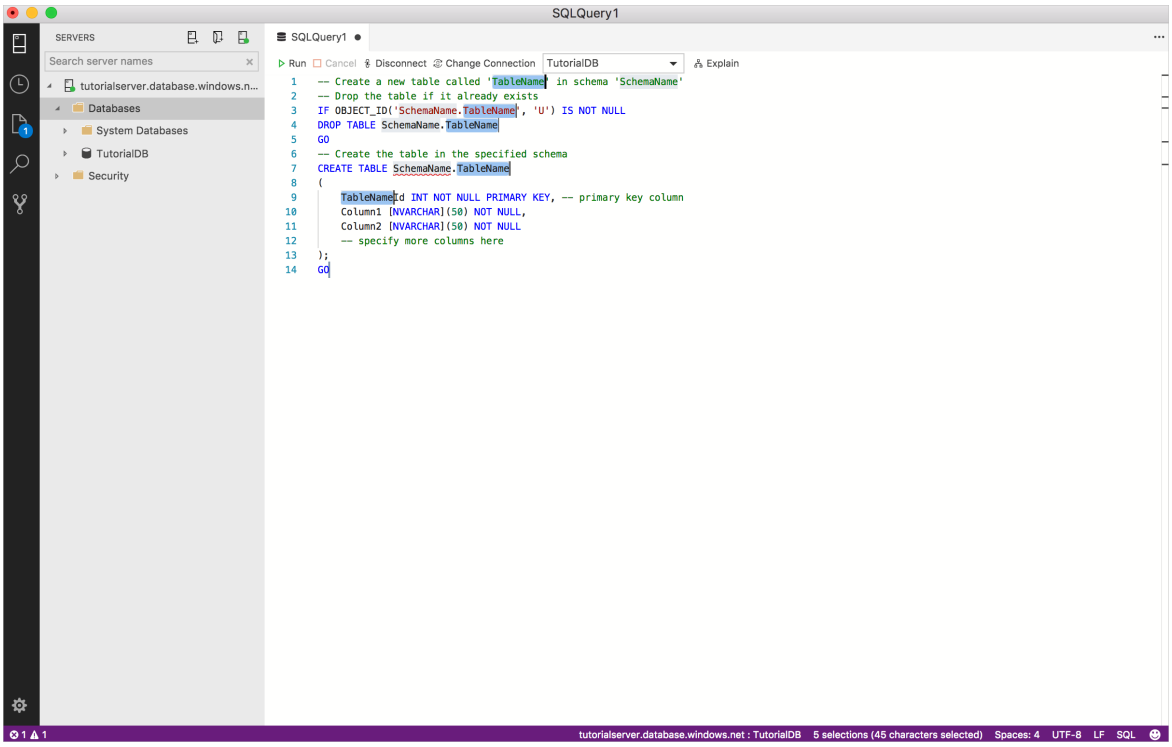
User-defined code snippets can also be created.

Using built-in T-SQL code snippets

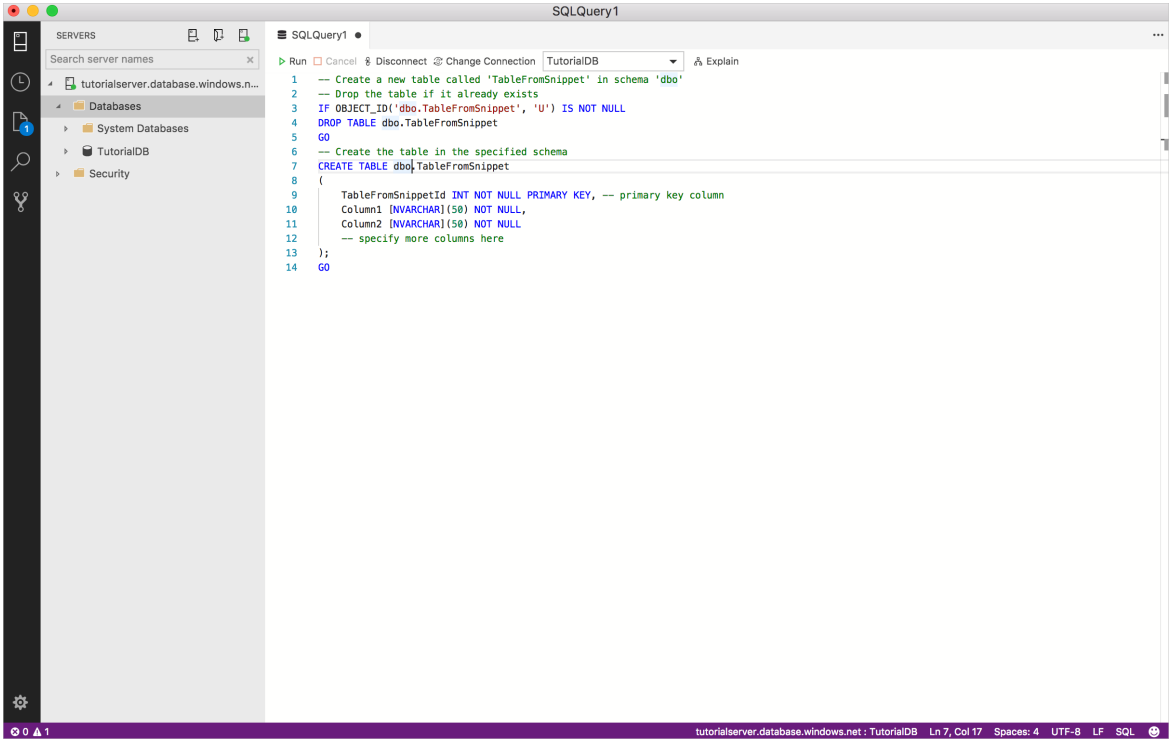
1. To access the available snippets, type *sql* in the query editor to open the list:



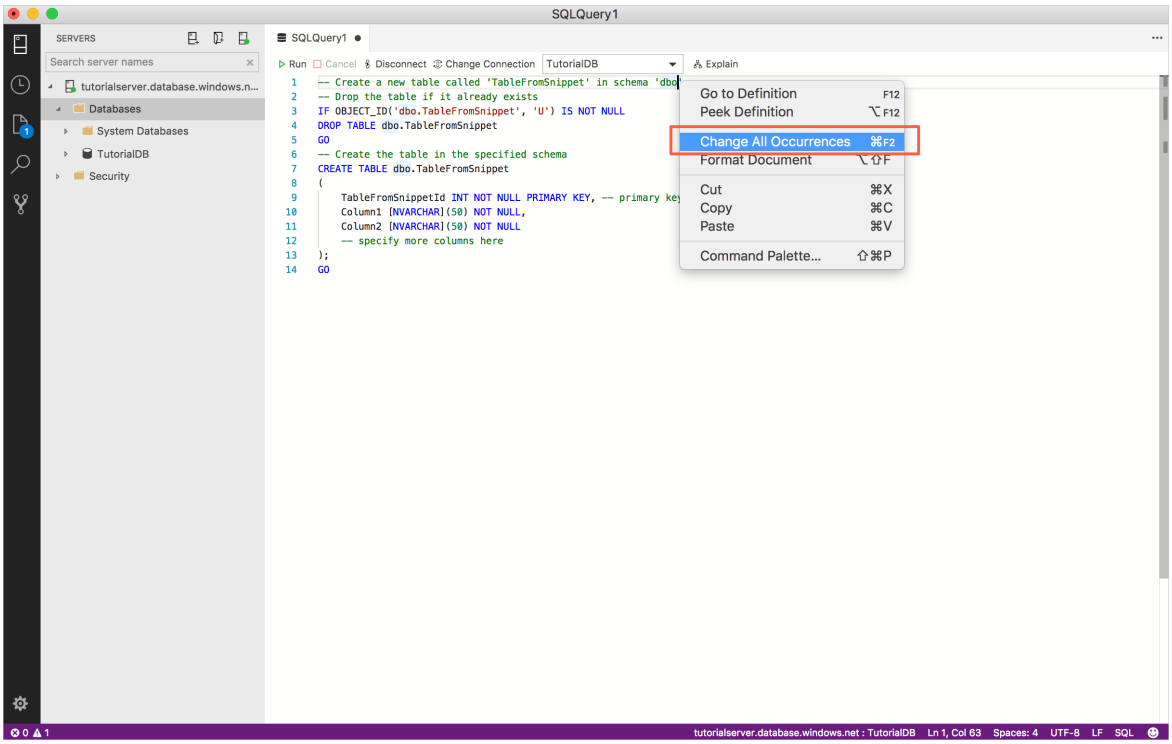
2. Select the snippet you want to use, and it generates the T-SQL script. For example, select *sqlCreateTable*.



3. Update the highlighted fields with your specific values. For example, replace *TableName* and *Schema* with the values for your database:



If the field you want to change is no longer highlighted (this happens when moving the cursor around the editor), right-click the word you want to change, and select **Change all occurrences**:

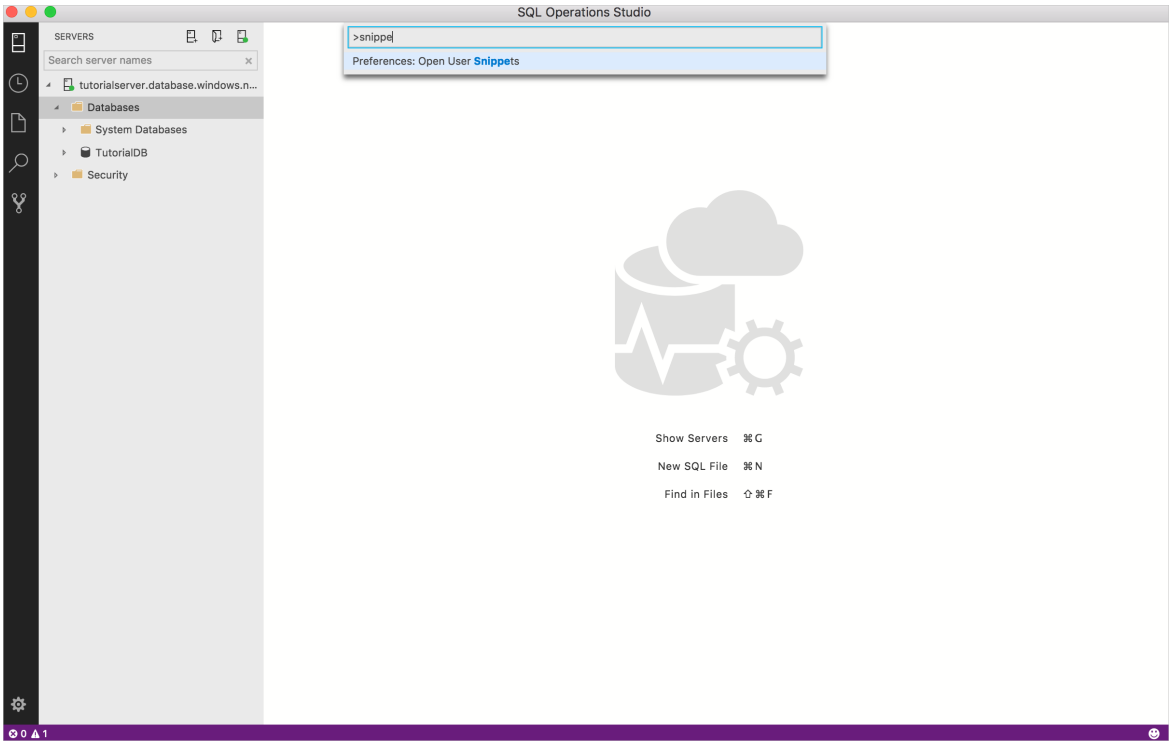


4. Update or add any additional T-SQL you need for the selected snippet. For example, update *Column1*, *Column2*, and add more columns.

Creating SQL code snippets

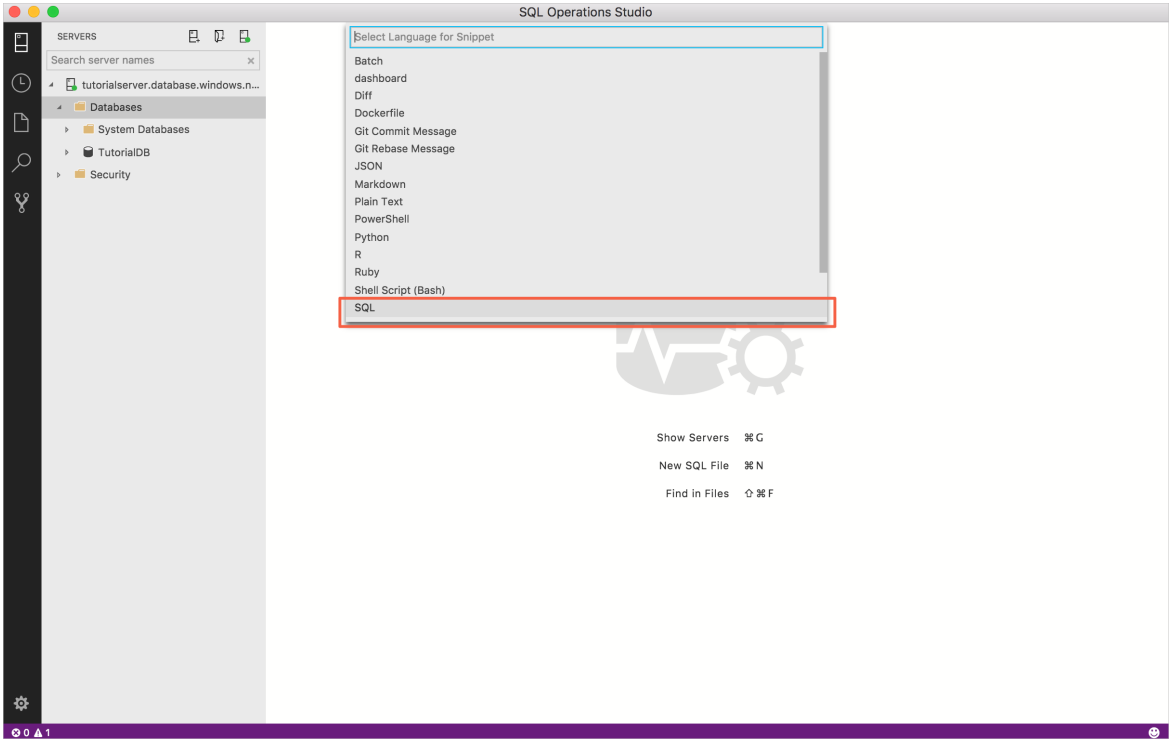
You can define your own snippets. To open up the SQL snippet file for editing:

1. Open the *Command Palette* (**Shift+Ctrl+P**), and type *snip*, and select **Preferences: Open User Snippets**:



2. Select **SQL**:

> **NOTE**
>
> Azure Data Studio inherits its code snippet functionality from Visual Studio Code so this article specifically discusses using SQL snippets. For more detailed information, see Creating your own snippets in the Visual Studio Code documentation.



3. Paste the following code into *sql.json*:

```
{
 "Select top 5": {
"prefix": "sqlSelectTop5",
"body": "SELECT TOP 5 * FROM ${1:TableName}",
"description": "User-defined snippet example 1"
},
"Create Table snippet":{
"prefix": "sqlCreateTable2",
"body": [
"-- Create a new table called '${1:TableName}' in schema '${2:SchemaName}'",
"-- Drop the table if it already exists",
"IF OBJECT_ID('$2.$1', 'U') IS NOT NULL",
"DROP TABLE $2.$1",
"GO",
"-- Create the table in the specified schema",
"CREATE TABLE $2.$1",
"(",
"$1Id INT NOT NULL PRIMARY KEY, -- primary key column",
"Column1 [NVARCHAR](50) NOT NULL,",
"Column2 [NVARCHAR](50) NOT NULL",
"-- specify more columns here",
");",
"GO"
],
   "description": "User-defined snippet example 2"
   }
   }
```

4. Save the sql.json file.

5. Open a new query editor window by clicking **Ctrl+N**.

6. Type **sql**, and you see the two user snippets you just added; *sqlCreateTable2* and *sqlSelectTop5*.

Select one of the new snippets and give it a test run!

Next steps

For information about the SQL editor, see Code editor tutorial.

Explore and manage Azure SQL resources with Azure Resource Explorer

11/2/2020 • 2 minutes to read • Edit Online

In this document, you learn how you can explore and manage Azure SQL Server, Azure SQL database, and Azure SQL Managed Instance resources through Azure Resource Explorer in Azure Data Studio.

> **NOTE**
>
> The Azure Resource Explorer is supported in SQL Server 2019. After that, you can install the extension through extension manager or through **File** > **Install Package from VSIX Package**.

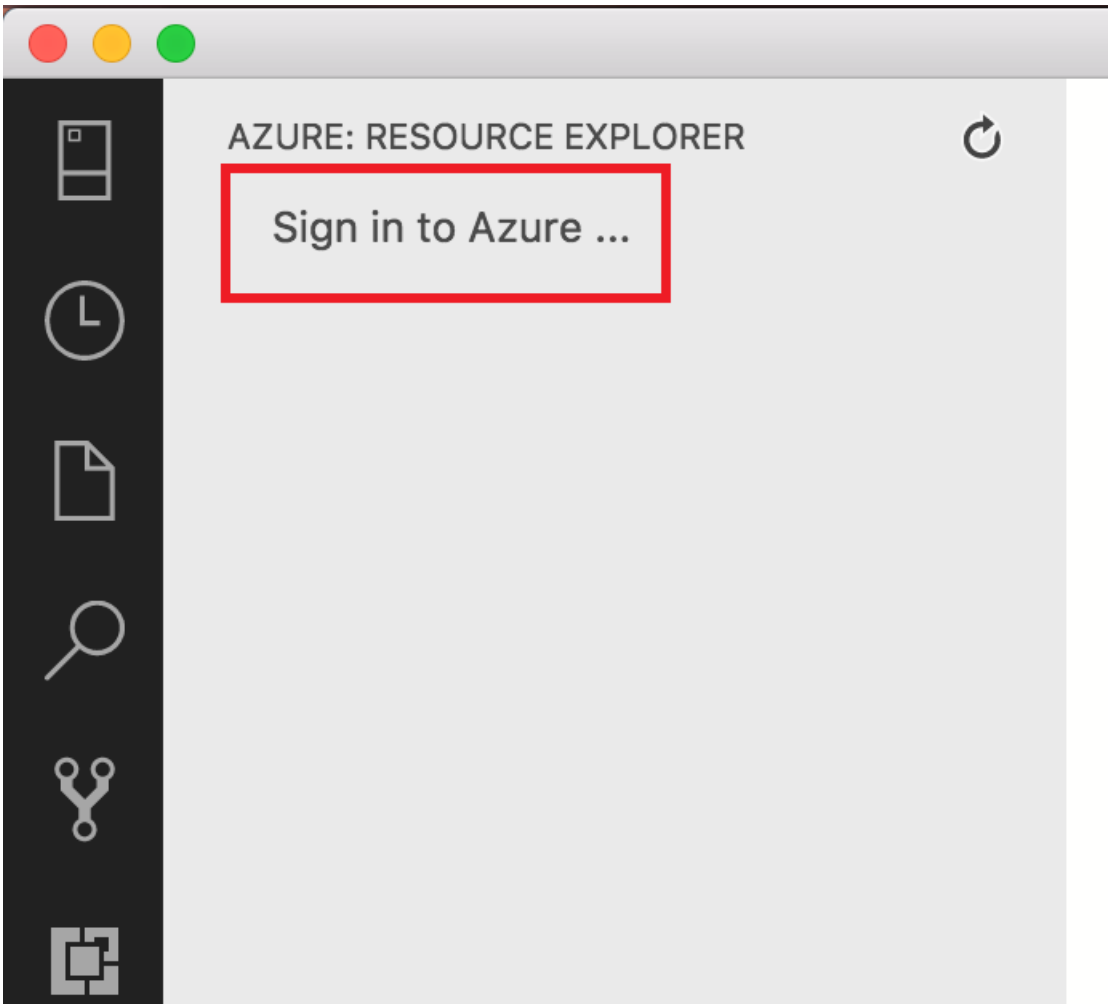
Connect to Azure

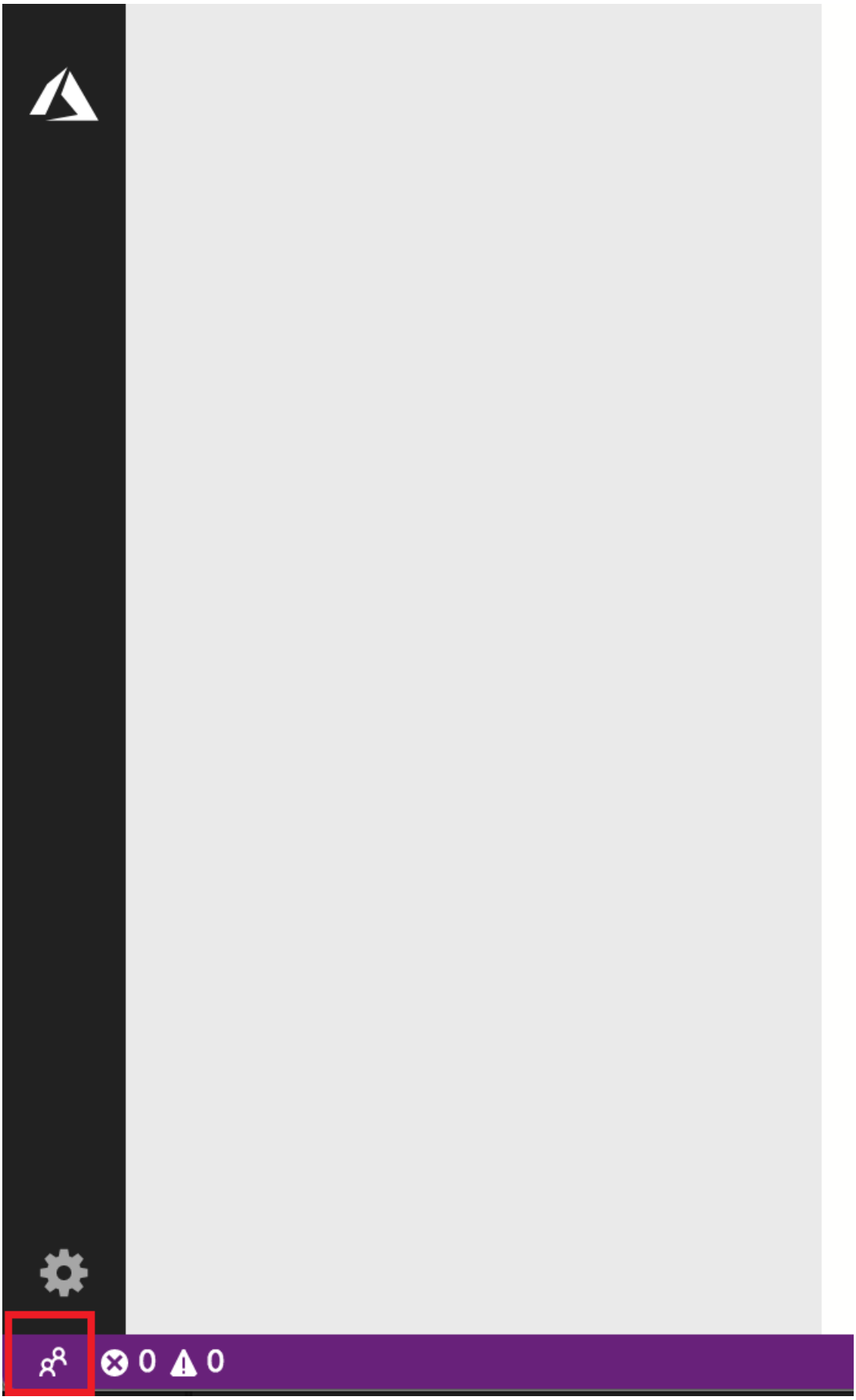
After installing the SQL plugin, an Azure icon appears in the left menu bar. Click the icon to open Azure Resource Explorer. If you don't see the Azure icon, right click the left menu bar, and select **Azure Resource Explorer**.

Add an Azure account

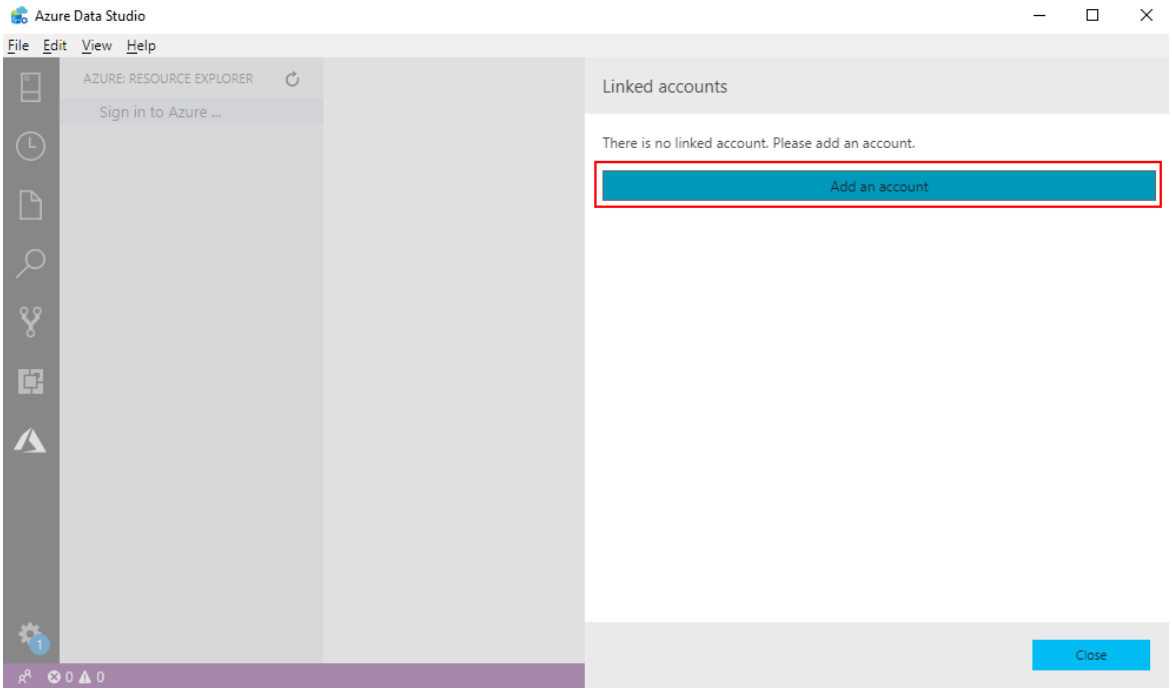
To view the SQL resources associated with an Azure account, you must first add the account to Azure Data Studio.

1. Open **Linked Accounts** dialog through the account management icon on the left bottom, or through **Sign in to Azure...** link in Azure Resource Explorer.

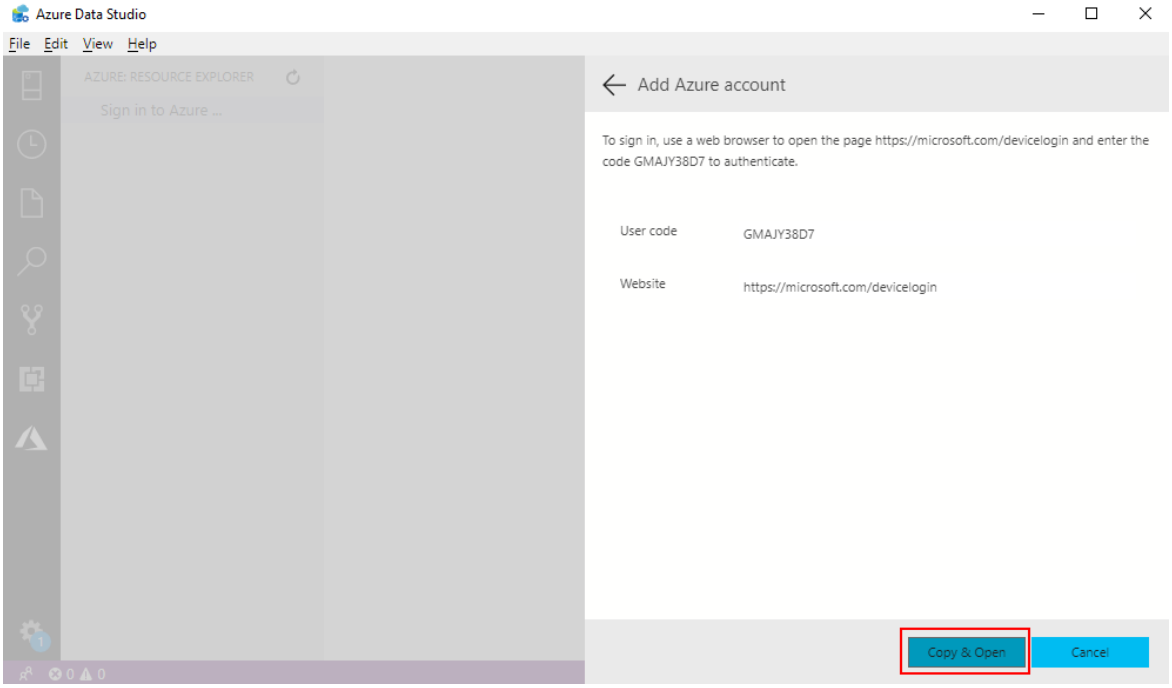




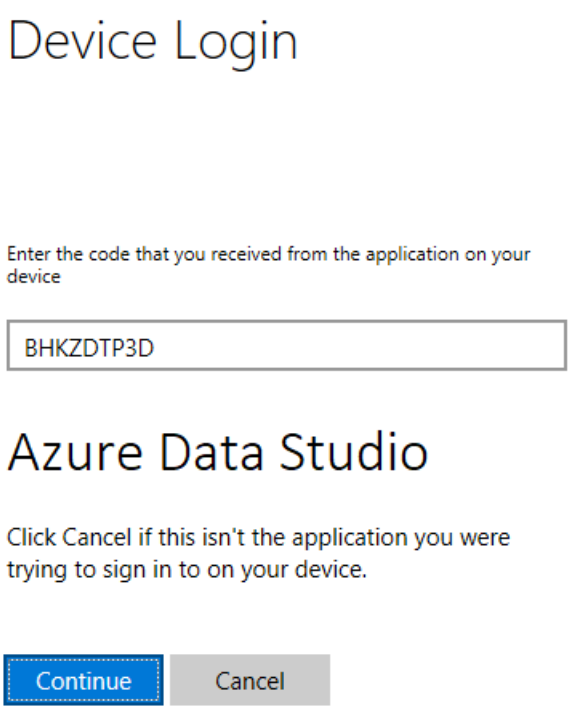
2. In the **Linked Accounts** dialog, click **Add an account**.



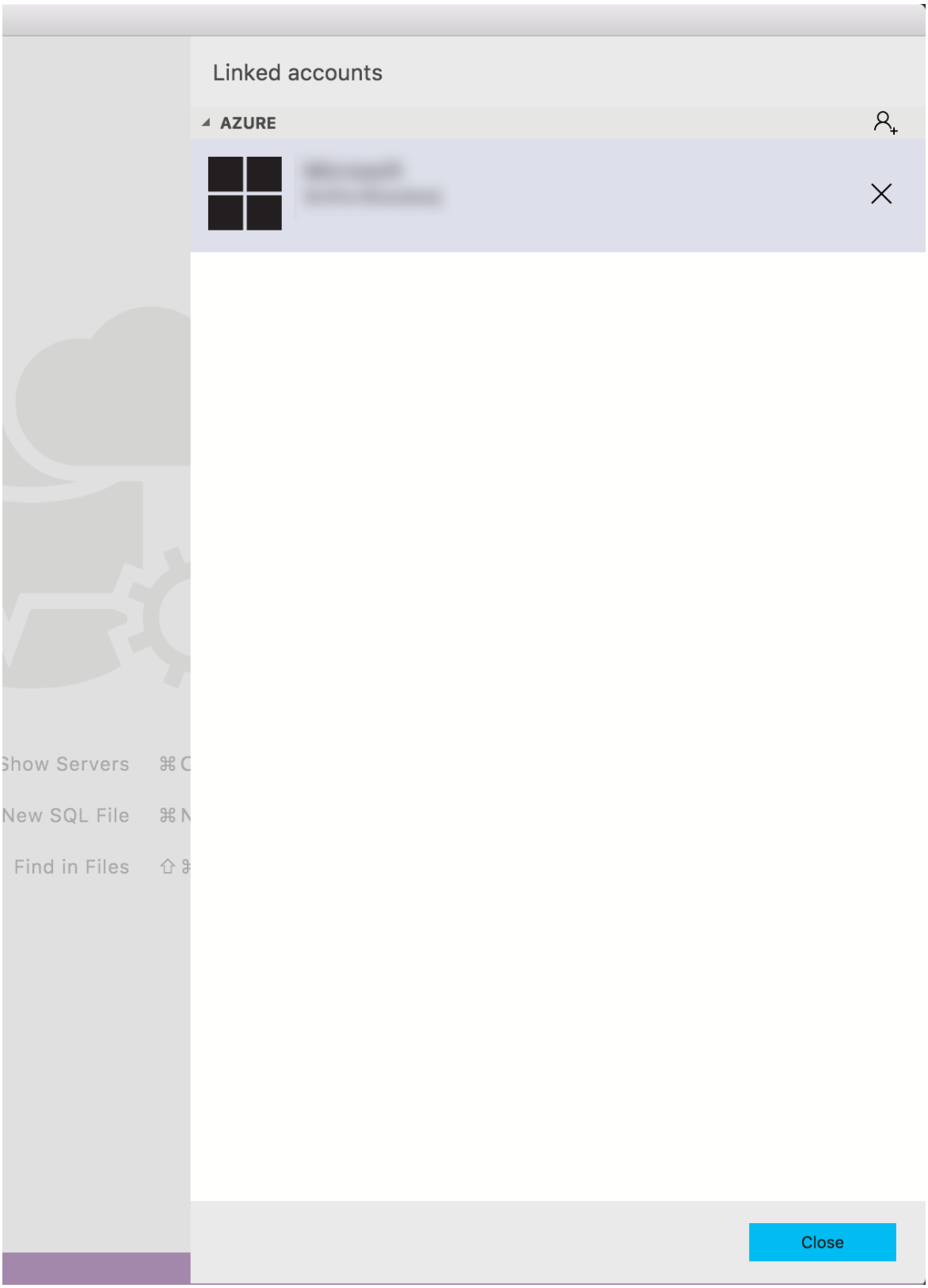
3. Click **Copy and Open** to open the browser for authentication.



4. Paste the **User code** in the web page and click **Continue** to authenticate.

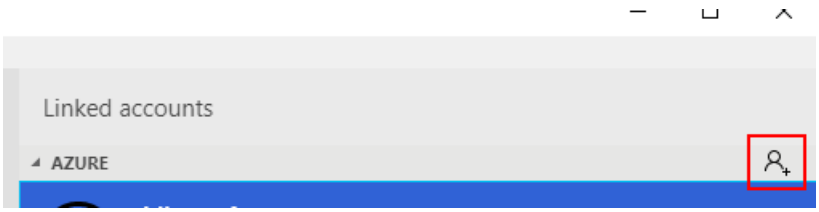


5. In Azure Data Studio you should now find the logged in Azure account in **Linked Accounts** dialog.



Add more Azure accounts

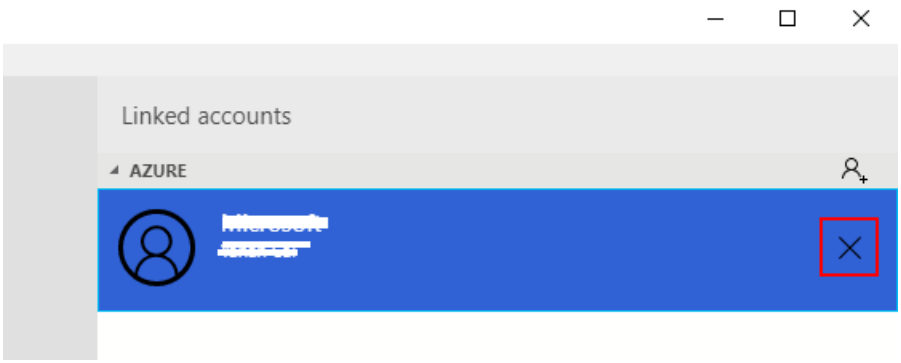
Multiple Azure accounts are supported in Azure Data Studio. To add more Azure accounts, click the button on the right top of **Linked Accounts** dialog and follow the same steps with Add an Azure account section to add more Azure accounts.



Remove an Azure account

To remove an existing logged in Azure account:

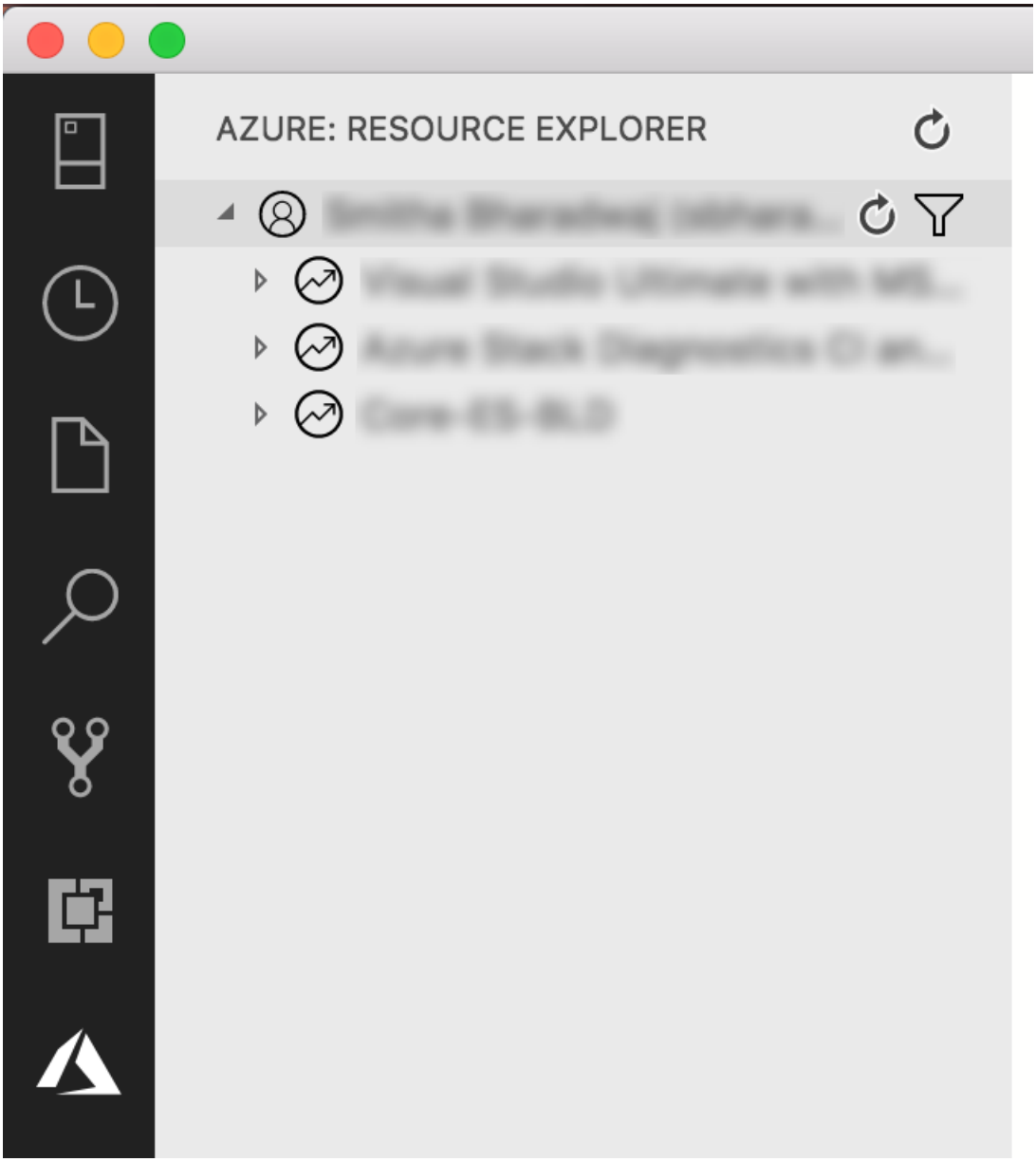
1. Open **Linked Accounts** dialog through the account management icon on the left bottom.
2. Click the **X** button at the right of the Azure account to remove it.

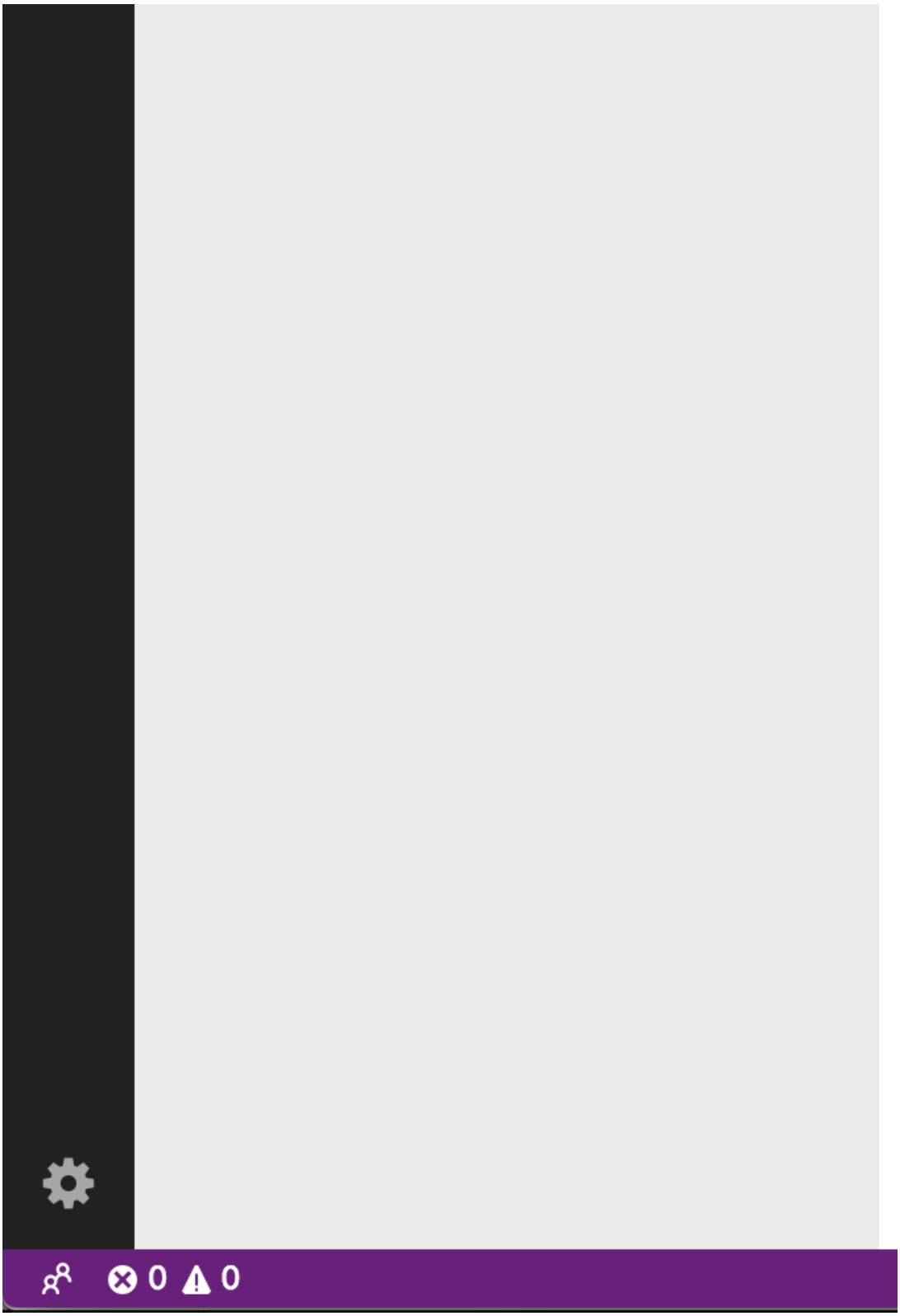


Filter subscription

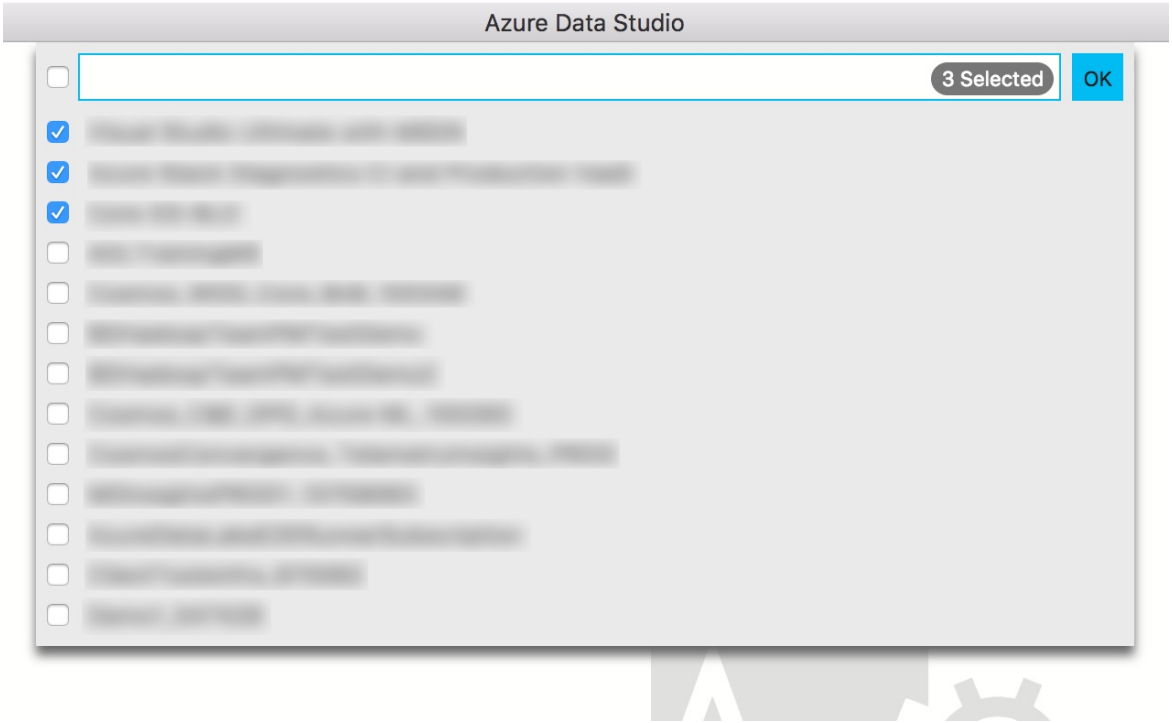
Once logged in to an Azure account, all subscriptions associated with that Azure account display in Azure Resource Explorer. You can filter subscriptions for each Azure account.

1. Click the **Select Subscription** button at right of the Azure account.





2. Select the check boxes for the account subscriptions you want to browse and then click **OK**.



Explore Azure SQL resources

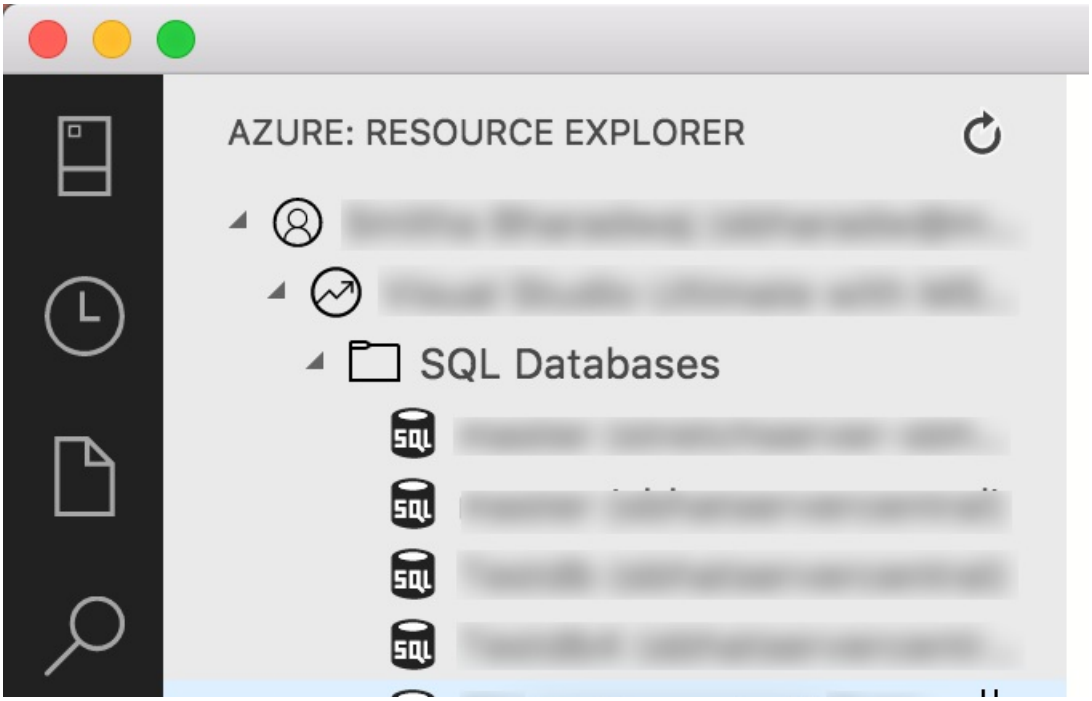
To navigate an Azure SQL resource in Azure Resource Explorer, expand the Azure accounts and resource type group.

Azure Resource Explorer supports Azure SQL Server, Azure SQL Database and Azure SQL Managed Instance currently.

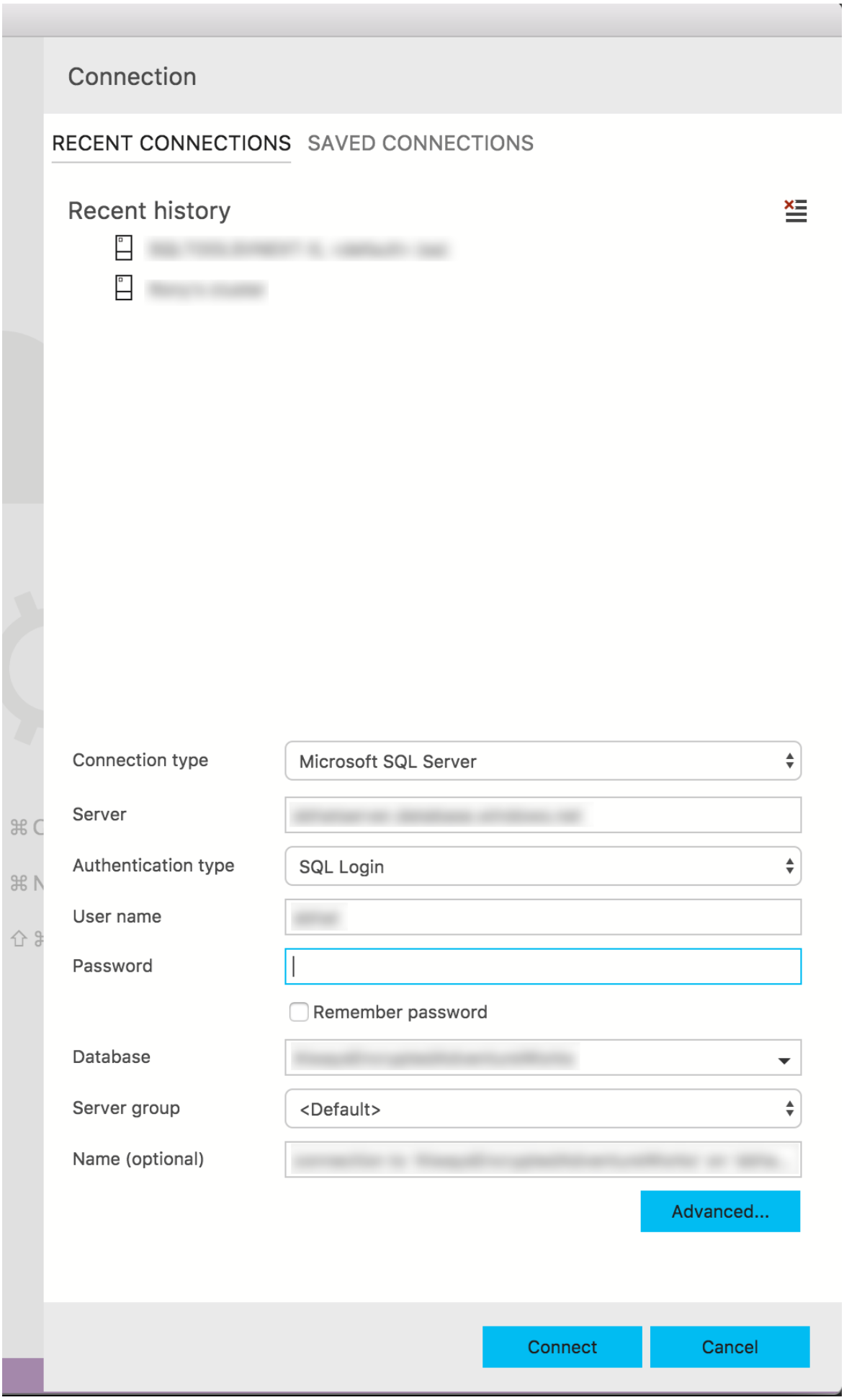
Connect to Azure SQL resources

Azure Resource Explorer provide quick access that helps you connect to SQL Servers and databases for query and management.

1. Explore the SQL resource you would like to connect with from the tree view.
2. Right click the resource and select **Connect**, you can also find the connect button at the right of the resource.



3. In the opened **Connection** dialog, enter your password and click **Connect**.



4. The **Servers** window automatically opens with the new connected SQL server/database after connection succeeds.

Next steps

- Use Azure Data Studio to connect and query Azure SQL database
- Use Azure Data Studio to connect and query data in Azure Synapse Analytics

Keyboard shortcuts in Azure Data Studio

11/2/2020 • 2 minutes to read • Edit Online

This article provides the steps to quickly view, edit, and create keyboard shortcuts in Azure Data Studio.

Because Azure Data Studio inherits its key binding functionality from Visual Studio Code, detailed information about advanced customizations, using different keyboard layouts, etc., is in the Key Bindings for Visual Studio Code article. Some key binding features may not be available (for example, Keymap extensions are not supported in Azure Data Studio).

Open the Keyboard Shortcuts editor

To view all currently defined keyboard shortcuts:

Open the **Keyboard Shortcuts** editor from the **File** menu: **File** > **Preferences** > **Keyboard Shortcuts** (**Azure Data Studio** > **Preferences** > **Keyboard Shortcuts** on Mac).

In addition to displaying current key bindings, the **Keyboard Shortcuts** editor lists the available commands that do not have keyboard shortcuts defined. The **Keyboard Shortcuts** editor enables you to easily change, remove, reset, and define new key bindings.

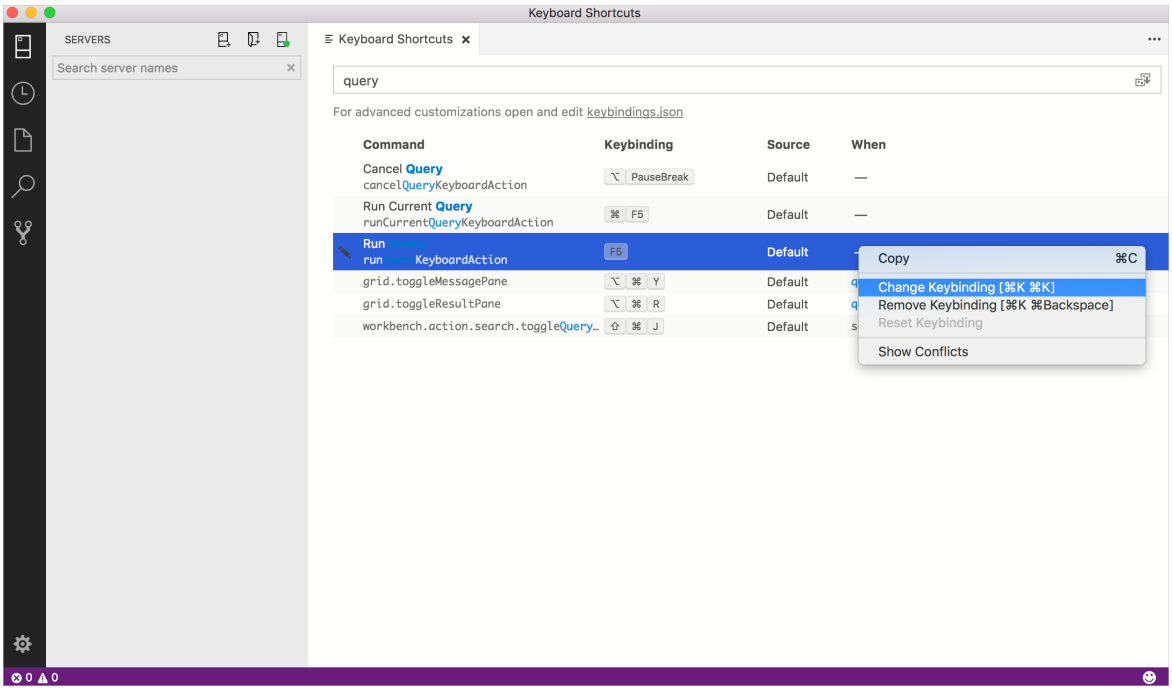
Edit existing keyboard shortcuts

To change the key binding for an existing keyboard shortcut:

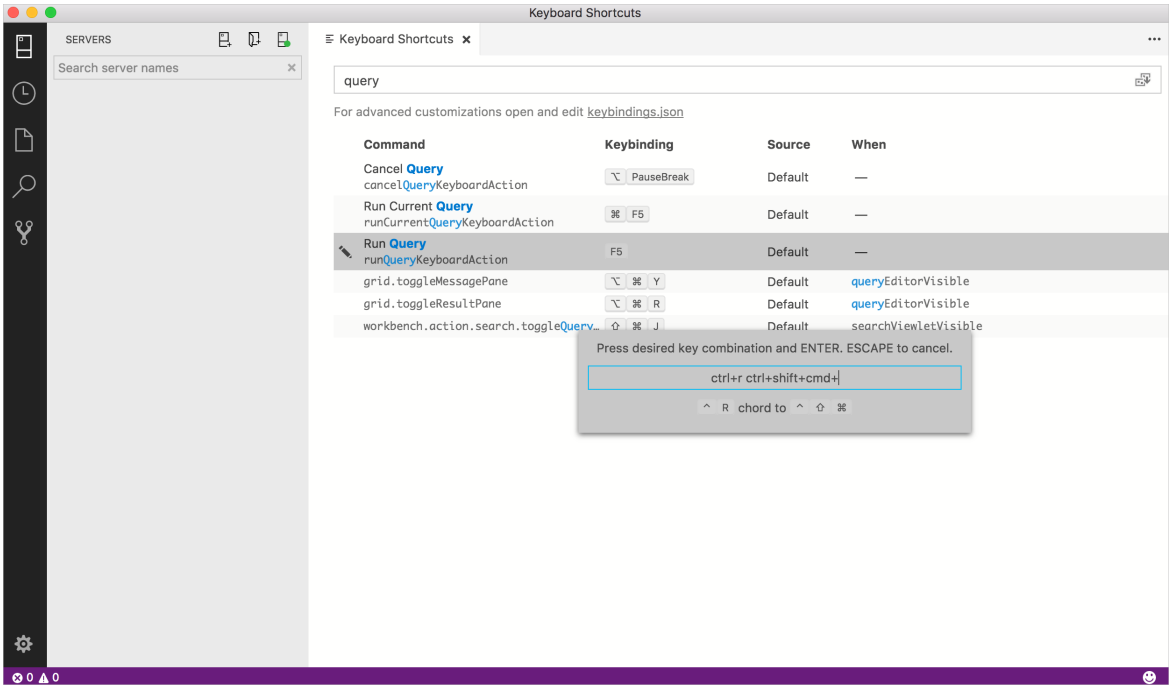
1. Locate the keyboard shortcut you want to change by using the search box or scrolling through the list.

   > **TIP**
   > Search by key, by command, by source, etc. to return all relevant keyboard shortcuts.

2. Right-click the desired entry and select **Change Key binding**



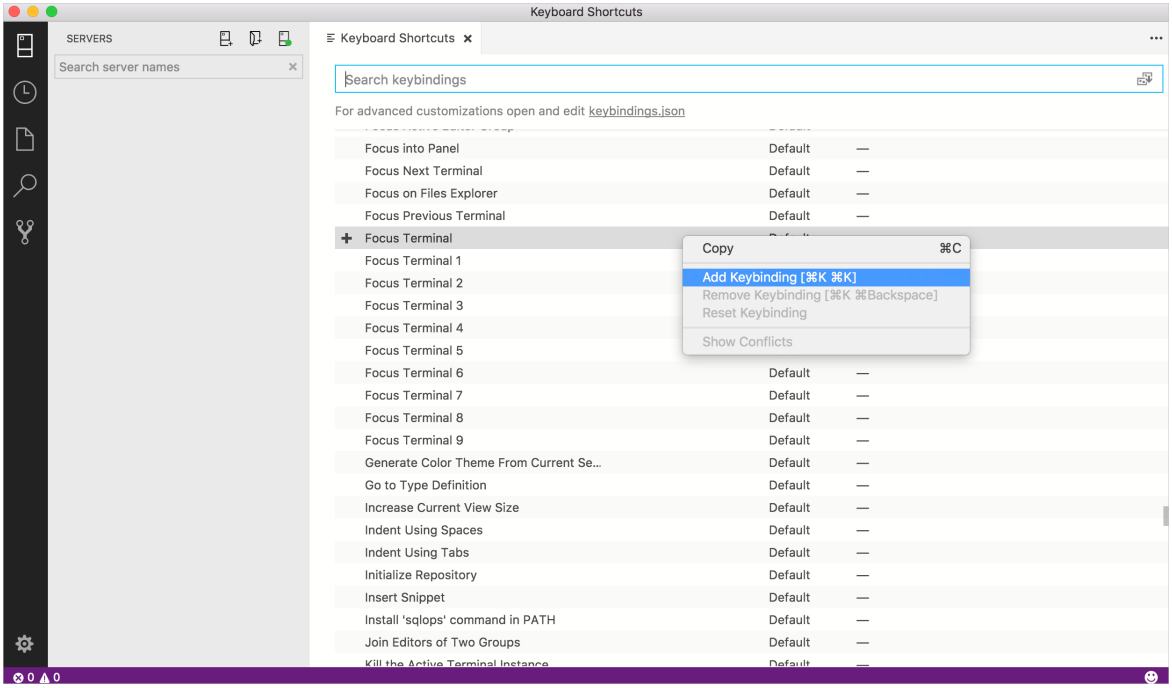
3. Press the desired combination of keys, then press **Enter** to save it.



Create new keyboard shortcuts

To create new keyboard shortcuts:

1. Right-click a command that doesn't have any key binding and select **Add Key binding**.



2. Press the desired combination of keys, then press **Enter** to save it.

Enable or disable usage data collection for Azure Data Studio

5/18/2021 • 2 minutes to read • Edit Online

Azure Data Studio contains Internet-enabled features that can collect and send anonymous feature usage and diagnostic data to Microsoft.

Azure Data Studio may collect standard computer, use, and performance information that may be transmitted to Microsoft and analyzed to improve the quality, security, and reliability of Azure Data Studio.

Azure Data Studio doesn't collect your name or address, but Azure Data Studio gathers data that helps approximate a single user for diagnostic purposes (based on a hash of the network adapter NIC).

Several updates made to Azure Data Studio to help ensure data privacy.

- Making it more accessible to opt-out of telemetry collection by placing a notification in the product for all existing and new users.
- Reviewing and classifying the telemetry that we send.
- Ensuring that we have valid data retention policies in place for any data we collect, for example, crash dumps.

For details, see the Microsoft Privacy Statement, and SQL Server Privacy supplement.

Audit feature usage and diagnostic data

To see feature usage data that is collected by Azure Data Studio, follow the steps below:

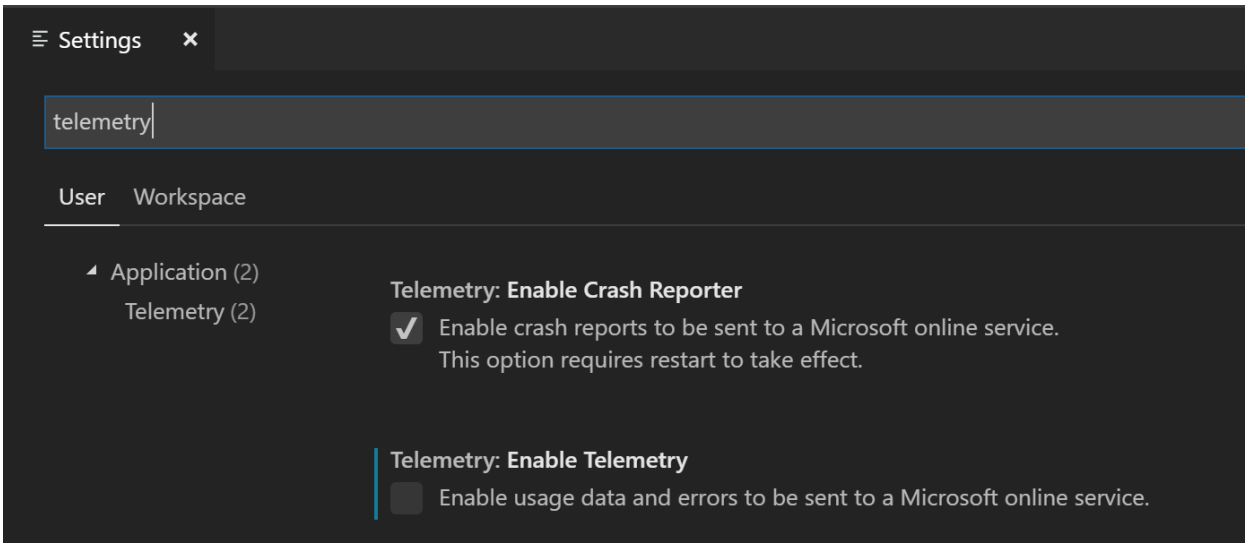
1. Launch Azure Data Studio.
2. Open the command palette and choose the **Developer: Set Log Level...** command
3. Select **Trace** from the options
4. Open the Output panel (Ctrl+Shift+U)
5. Select **Log (Telemetry)** from the dropdown
6. Select **View**, then Select **Output** in the main menu to show the **Output** window.
7. When the **Output** window is visible, choose **Log (Window)** in the **Show output from:** menu.

When tracing telemetry events, the events are also logged to a local file telemetry.log, which you can view using the **Developer: Open Log File...** command and choosing **Telemetry** from the dropdown.

Disable telemetry reporting

To not send usage data to Microsoft, you can set the `telemetry.enableTelemetry` user settings to `false`.

Go to **File** > **Preferences** > **Settings** (macOS: **Code** > **Preferences** > **Settings**) and search for `telemetry`, and uncheck the **Telemetry: Enable Telemetry** setting. With this setting, you silence all telemetry events from Azure Data Studio going forward. Telemetry information may have been collected and sent up until the point when you disable the setting.



If you use the JSON editor for your settings, add the following line:

```
"telemetry.enableTelemetry": false
```

Disable crash reporting

Azure Data Studio collects data about any crashes that occur and sends it to Microsoft to help improve our products and services.

If you don't want to send, crash data to Microsoft, you can change the `enable-crash-reporter` runtime argument to `false`.

1. Open the Command Palette (`kb(workbench.action.showCommands)`).
2. Run the **Preferences: Configure Runtime Arguments** command.
3. This command opens a `argv.json` file to configure runtime arguments.
4. Edit `"enable-crash-reporter": false`.
5. Restart Azure Data Studio.

Extensions and telemetry

Azure Data Studio lets you add features to the product by installing Microsoft and third-party extensions. These extensions may be collecting their own usage data and are not controlled by the `telemetry.enableTelemetry` setting. Consult the specific extension's documentation to learn about its telemetry reporting and whether it can be disabled.

More resources

- Workspace and User settings
- GDPR section of the Service Trust portal
- GDPR section of the Microsoft Trust Center

See also

- Configure usage and diagnostic data collection for SQL Server
- Local audit for SQL Server usage and diagnostic data collection

Connect Azure Data Studio to SQL Server using Kerberos

6/28/2021 • 3 minutes to read • Edit Online

Azure Data Studio supports connecting to SQL Server by using Kerberos.

To use integrated authentication (Windows Authentication) on macOS or Linux, you need to set up a *Kerberos ticket* that links your current user to a Windows domain account.

Prerequisites

To get started, you need:

- Access to a Windows domain-joined machine to query your Kerberos domain controller.
- SQL Server should be configured to allow Kerberos authentication. For the client driver running on Unix, integrated authentication is supported only by using Kerberos. For more information, see Using Kerberos integrated authentication to connect to SQL Server. There should be service principal names (SPNs) registered for each instance of SQL Server you're trying to connect to. For more information, see Register a Service Principal Name for Kerberos Connections.

Check if SQL Server has a Kerberos setup

Sign in to the host machine of SQL Server. From the Windows command prompt, use `setspn -L %COMPUTERNAME%` to list all the SPNs for the host. Verify there are entries that begin with MSSQLSvc/HostName.Domain.com. These entries mean that SQL Server has registered an SPN and is ready to accept Kerberos authentication.

If you don't have access to the host of the SQL Server instance, then from any other Windows OS joined to the same Active Directory, you could use the command `setspn -L <SQLSERVER_NETBIOS>`, where *<SQLSERVER_NETBIOS>* is the computer name of the host of the SQL Server instance.

Get the Kerberos Key Distribution Center

Find the Kerberos Key Distribution Center (KDC) configuration value. Run the following command on a Windows computer that's joined to your Active Directory domain.

Start `cmd.exe` and run `nltest`.

```
nltest /dsgetdc:DOMAIN.COMPANY.COM (where "DOMAIN.COMPANY.COM" maps to your domain's name)

Sample Output
DC: \\dc-33.domain.company.com
Address: \\2111:4444:2111:33:1111:ecff:ffff:3333
...
The command completed successfully
```

Copy the DC name that's the required KDC configuration value. In this case, it's dc-33.domain.company.com.

Join your OS to the Active Directory domain controller

Ubuntu

```
sudo apt-get install realmd krb5-user software-properties-common python-software-properties packagekit
```

Edit the `/etc/network/interfaces` file so that your Active Directory domain controller's IP address is listed as dns-nameserver. For example:

```
<...>
# The primary network interface
auth eth0
iface eth0 inet dhcp
dns-nameservers **<AD domain controller IP address>**
dns-search **<AD domain name>**
```

> **NOTE**
> The network interface (eth0) might differ for different machines. To find out which one you're using, run ifconfig and copy the interface that has an IP address and transmitted and received bytes.

After editing this file, restart the network service:

```
sudo ifdown eth0 && sudo ifup eth0
```

Now check that your `/etc/resolv.conf` file contains a line like the following one:

```
nameserver **<AD domain controller IP address>**
```

```
sudo realm join contoso.com -U 'user@CONTOSO.COM' -v
<...>
* Success
```

RedHat Enterprise Linux

```
sudo yum install realmd krb5-workstation
```

Edit the `/etc/sysconfig/network-scripts/ifcfg-eth0` file (or other interface config file as appropriate) so that your Active Directory domain controller's IP address is listed as a DNS server:

```
<...>
PEERDNS=no
DNS1=**<AD domain controller IP address>**
```

After editing this file, restart the network service:

```
sudo systemctl restart network
```

Now check that your `/etc/resolv.conf` file contains a line like the following one:

```
nameserver **<AD domain controller IP address>**
```

```
sudo realm join contoso.com -U 'user@CONTOSO.COM' -v
<...>
* Success
```

Configure KDC in krb5.conf with macOS

This section discusses the Kerberos configuration file.

Edit the `/etc/krb5.conf` file in an editor of your choice. Configure the following keys:

```
sudo vi /etc/krb5.conf

[libdefaults]
  default_realm = DOMAIN.COMPANY.COM

[realms]
DOMAIN.COMPANY.COM = {
   kdc = dc-33.domain.company.com
}
```

Then save the krb5.conf file and exit.

> **NOTE**
> The domain must be in ALL CAPS.

Test the ticket granting ticket retrieval

Get a Ticket Granting Ticket (TGT) from KDC.

```
kinit username@DOMAIN.COMPANY.COM
```

View the available tickets by using klist. If the kinit was successful, you should see a ticket.

```
klist

krbtgt/DOMAIN.COMPANY.COM@ DOMAIN.COMPANY.COM.
```

Connect by using Azure Data Studio

1. Create a new connection profile.
2. Select **Windows Authentication** as the authentication type.
3. Complete the connection profile, and select **Connect**.

After successfully connecting, your server appears in the **SERVERS** sidebar.

Modify User and Workspace Settings

5/6/2021 • 2 minutes to read • Edit Online

it's easy to configure Azure Data Studio to your liking through settings. Nearly every part of Azure Data Studio's editor, user interface, and functional behavior has options you can modify.

Azure Data Studio provides two different scopes for settings:

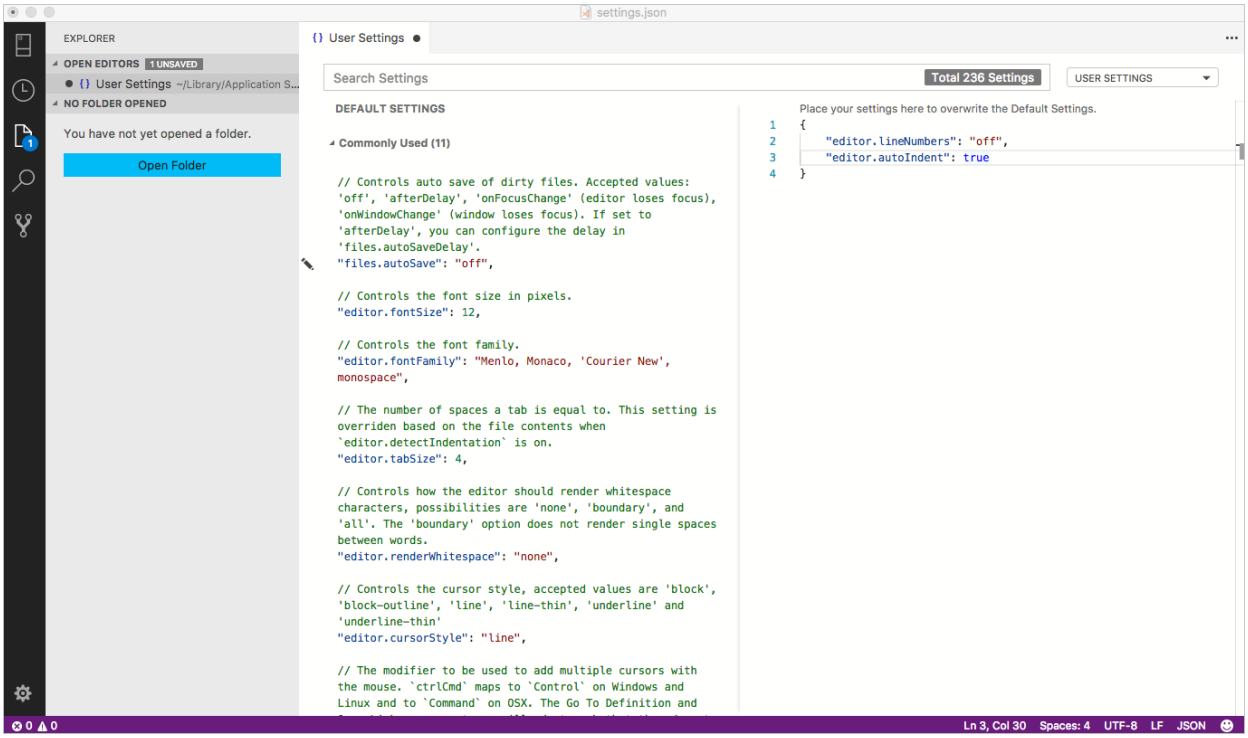
- **User** - These settings apply globally to any instance of Azure Data Studio you open.
- **Workspace** - Workspace settings are settings specific to a folder on your computer, and are only available when the folder is open in the Explorer sidebar. Settings defined on this scope override the user scope.

Creating User and Workspace Settings

The menu command **File** > **Preferences** > **Settings** (**Code** > **Preferences** > **Settings** on Mac) provides the entry point to configure user and workspace settings. You're provided with a list of Default Settings. Copy any setting that you want to change to the appropriate `settings.json` file. The tabs on the right let you switch quickly between the user and workspace settings files.

You can also open the user and workspace settings from the **Command Palette** (**Ctrl+Shift+P**) with **Preferences: Open User Settings** and **Preferences: Open Workspace Settings** or use the keyboard shortcut (**Ctrl+,**).

The following example disables line numbers in the editor and configures lines of code to be indented automatically.



Changes to settings are reloaded by Azure Data Studio after the modified `settings.json` file is saved.

> **NOTE**
>
> Workspace settings are useful for sharing project-specific settings across a team.

Settings File Locations

Depending on your platform, the user settings file is located here:

- **Windows** `%APPDATA%\azuredatastudio\User\settings.json`
- **Mac** `$HOME/Library/Application Support/azuredatastudio/User/settings.json`
- **Linux** `$HOME/.config/azuredatastudio/User/settings.json`

The workspace setting file is located under the `.Azure Data Studio` folder in your project.

Hot Exit

Azure Data Studio remembers unsaved changes to files when you exit by default. In Visual Studio Code, this is the same as the hot exit feature.

By default, hot exit's off. Enable hot exit by editing the `files.hotExit` setting. For details, see Hot Exit (in the Visual Studio Code documentation).

Tab color

To simplify identifying what connections you're working with, open tabs in the editor can have their colors set to match the color of the Server Group the connection belongs to. By default, tab colors are off by default. Enable tab colors by editing the `sql.tabColorMode` setting.

Additional resources

Because Azure Data Studio inherits its user and workspace settings functionality from Visual Studio Code, detailed information about settings is in the Settings for Visual Studio Code article.

SQL tools overview

3/5/2021 • 3 minutes to read • Edit Online

Applies to: SQL Server (all supported versions) Azure SQL Database Azure SQL Managed Instance Azure Synapse Analytics Parallel Data Warehouse

To manage your database, you need a tool. Whether your databases run in the cloud, on Windows, on macOS, or on Linux, your tool doesn't need to run on the same platform as the database.

You can view the links to the different SQL tools in the following tables.

> **NOTE**
> To download SQL Server, see Install SQL Server.

Recommended tools

The following tools provide a graphical user interface (GUI).

TOOL	DESCRIPTION	OPERATING SYSTEM
 Azure Data Studio	A light-weight editor that can run on-demand SQL queries, view and save results as text, JSON, or Excel. Edit data, organize your favorite database connections, and browse database objects in a familiar object browsing experience.	**Windows** **macOS** **Linux**
 SQL Server Management Studio (SSMS)	Manage a SQL Server instance or database with full GUI support. Access, configure, manage, administer, and develop all components of SQL Server, Azure SQL Database, and Azure Synapse Analytics. Provides a single comprehensive utility that combines a broad group of graphical tools with a number of rich script editors to provide access to SQL for developers and database administrators of all skill levels.	**Windows**
 SQL Server Data Tools (SSDT)	A modern development tool for building SQL Server relational databases, Azure SQL databases, Analysis Services (AS) data models, Integration Services (IS) packages, and Reporting Services (RS) reports. With SSDT, you can design and deploy any SQL Server content type with the same ease as you would develop an application in Visual Studio.	**Windows**

TOOL	DESCRIPTION	OPERATING SYSTEM
 Visual Studio Code	The mssql extension for Visual Studio Code is the official SQL Server extension that supports connections to SQL Server and rich editing experience for T-SQL in Visual Studio Code. Write T-SQL scripts in a light-weight editor.	Windows macOS Linux

Command-line tools

The tools below are the main command-line tools.

TOOL	DESCRIPTION	OPERATING SYSTEM
bcp	The **b**ulk **c**opy **p**rogram utility (**bcp**) bulk copies data between an instance of Microsoft SQL Server and a data file in a user-specified format.	Windows macOS Linux
mssql-cli (preview)	**mssql-cli** is an interactive command-line tool for querying SQL Server. Also, query SQL Server with a command-line tool that features IntelliSense, syntax high-lighting, and more.	Windows macOS Linux
mssql-conf	**mssql-conf** configures SQL Server running on Linux.	Linux
mssql-scripter (preview)	**mssql-scripter** is a multi-platform command-line experience for scripting SQL Server databases.	Windows macOS Linux
sqlcmd	**sqlcmd** utility lets you enter Transact-SQL statements, system procedures, and script files at the command prompt.	Windows macOS Linux
sqlpackage	**sqlpackage** is a command-line utility that automates several database development tasks.	Windows macOS Linux
SQL Server PowerShell	**SQL Server PowerShell** provides cmdlets for working with SQL.	Windows macOS Linux

Migration and other tools

These tools are used to migrate, configure, and provide other features for SQL databases.

TOOL	DESCRIPTION
Configuration Manager	Use SQL Server Configuration Manager to configure SQL Server services and configure network connectivity. Configuration Manager runs on Windows

TOOL	DESCRIPTION
Database Experimentation Assistant	Use Database Experimentation Assistant to evaluate a targeted version of SQL for a given workload.
Data Migration Assistant	The Data Migration Assistant tool helps you upgrade to a modern data platform by detecting compatibility issues that can impact database functionality in your new version of SQL Server or Azure SQL Database.
Distributed Replay	Use the Distributed Replay feature to help you assess the impact of future SQL Server upgrades. Also use Distributed Replay to help assess the impact of hardware and operating system upgrades, and SQL Server tuning.
ssbdiagnose	The ssbdiagnose utility reports issues in Service Broker conversations or the configuration of Service Broker services.
SQL Server Migration Assistant	Use SQL Server Migration Assistant to automate database migration to SQL Server from Microsoft Access, DB2, MySQL, Oracle, and Sybase.

If you're looking for additional tools that aren't mentioned on this page, see SQL Command Prompt Utilities and Download SQL Server extended features and tools

Azure Data Studio Troubleshooting

3/5/2021 • 2 minutes to read • Edit Online

Azure Data Studio tracks issues and feature requests using on a GitHub repository issue tracker for the `azuredatastudio` repository.

If you've experienced any issue

Report issues to GitHub Issue Tracker and let us know any details that will help reproduce the error. Include any log information from the log file.

Writing good bug reports and feature requests

File a single issue per problem and feature request.

- Don't enumerate multiple bugs or feature requests in the same issue.
- Don't add your issue as a comment to an existing issue unless it's for the identical input. Many issues look similar, but have different causes.

The more information you can provide, the more likely someone will be successful reproducing the issue and finding a fix.

Include the following information with each issue:

- Version of Azure Data Studio
- Reproducible steps (1... 2... 3...) and what you expected versus what you actually saw.
- Images, animations, or a link to a video. Images and animations illustrate repro-steps but don't replace them.
- A code snippet that demonstrates the issue or a link to a code repository we can easily pull down onto our machine to recreate the issue.

> **NOTE**
> Because we need to copy and paste the code snippet, including a code snippet as a media file (i.e. .gif) is not sufficient.

- Errors in the Dev Tools Console (Help | Toggle Developer Tools)

Please remember to do the following:

- Search the issue repository to see if there exists a duplicate.
- Simplify your code around the issue so we can better isolate the problem.

Don't feel bad if we can't reproduce the issue and ask for more information!

How to set the logging level

Azure Data Studio

Run the `Developer: Set Log Level...` command to select the log level for the current session. This value is NOT persisted over multiple sessions - so if you restart Azure Data Studio it will revert back to the default `Info` level.

If you want to enable debug logging for startup then set the log level to `Debug` and run the `Developer: Reload Window` command

MSSQL (Built-In Extension)

If the `Mssql: Log Debug Info` user setting is set to true, then debug log info will be sent to the `MSSQL` output channel.

The `Mssql: Tracing Level` user setting is used to control the verbosity of the logging.

Debug log location

From Azure Data Studio, run the `Developer: Open Logs Folder` command to open the path to the logs. There's many different types of log files that write there, a few of the commonly used ones are:

1. `renderer#.log` (for example, renderer1.log) - this file is the log file for the main process.
2. `telemetry.log` - When the log level is set to `Trace` this file will contain the telemetry events sent by Azure Data Studio
3. `exthost#/exthost.log` - Log file for the extension host process (this is only the process itself, not the extensions running inside it)
4. `exthost#/Microsoft.mssql` - Logs for the mssql extension, which contains much of the core logic for MSSQL-related features
   - sqltools.log is the log for SQL Tools Service
5. `exthost#/output_logging_#######` - these folders contain the messages displayed in the `Output` panel in Azure Data Studio. Each file is named `#-<Channel Name>` so for example the `Notebooks` output channel may output to a file named `3-Notebooks.log`.

If you are asked to provide logs, zip up the entire folder to ensure that the correct logs are included.

Next Steps

- Report an issue
- What is Azure Data Studio