

Contents

Azure Data Explorer Documentation

Overview

[What is Azure Data Explorer?](#)

Quickstarts

[Create cluster and database - Azure portal](#)

[Ingest sample data](#)

[Query data with Web UI](#)

Tutorials

[E2E programmatic Blob ingestion](#)

[C#](#)

[Python](#)

[Ingest and query monitoring data](#)

[Visualize data in Power BI](#)

Concepts

[Data ingestion overview](#)

[Data visualization overview](#)

How-to guides

[Create cluster and database](#)

[Azure CLI](#)

[PowerShell](#)

[C#](#)

[Python](#)

[Azure Resource Manager template](#)

Ingest data

[Event Hub](#)

[Azure portal](#)

[Python](#)

[C#](#)

[Azure Resource Manager template](#)

[Event Grid](#)

[Azure portal](#)

[Python](#)

[C#](#)

[Azure Resource Manager template](#)

[IoT Hub](#)

[Azure portal](#)

[Python](#)

[C#](#)

[Azure Resource Manager template](#)

[One-click ingestion](#)

[Streaming ingestion](#)

[Kafka](#)

[Python](#)

[Node SDK](#)

[.NET Standard SDK](#)

[Logstash](#)

[Copy data using Azure Data Factory](#)

[Query data](#)

[Write queries](#)

[Query data in Azure Data Lake](#)

[Query data in Azure Monitor](#)

[Debug KQL inline Python](#)

[using SDKs](#)

[Python](#)

[Time Series Analysis and Machine Learning](#)

[Time series analysis](#)

[Anomaly detection and forecasting](#)

[Machine learning](#)

[Visualize data](#)

[Power BI](#)

[Best practices](#)

- connector
- imported query
- SQL query
- Excel
 - connector
 - blank query
- Grafana
- ODBC connector
- Tableau
- Sisense
- Redash
- Manage
 - Cluster
 - Select cluster VM SKU
 - Manage cluster horizontal scaling
 - Manage cluster vertical scaling
 - Manage cluster security
 - Check cluster health
 - Database
 - Follower databases
 - Manage database permissions
 - Policies
 - C#
 - Python
- Data
 - Delete data
 - Deal with duplicate data
- Deploy
 - Deploy your cluster to your virtual network
- Monitor
 - Use diagnostic logs to monitor ingestion
 - Use metrics to monitor cluster health

[Integrate with other tools](#)

[Azure Data Factory](#)

[Integration with Data Factory](#)

[Bulk copy data using Azure Data Factory template](#)

[Run control commands using Azure Data Factory](#)

[Apache Spark Connector](#)

[Connect from Databricks using Python](#)

[Analyze data in Jupyter Notebooks & Kqlmagic](#)

[Azure Pipelines](#)

[Troubleshoot](#)

[Creating a cluster](#)

[Connecting to a cluster](#)

[Working with databases and tables](#)

[Reference](#)

[Query language](#)

[Management commands](#)

[APIs](#)

[Tools](#)

[Concepts](#)

[Resources](#)

[Blog](#)

[Forums](#)

[Stack Overflow](#)

[Microsoft Tech Community](#)

[MSDN forum](#)

[Product feedback](#)

[Pricing & billing](#)

[Pricing page](#)

[Cost estimator](#)

[Reserved capacity](#)

[Service updates](#)

What is Azure Data Explorer?

4/4/2019 • 3 minutes to read • [Edit Online](#)

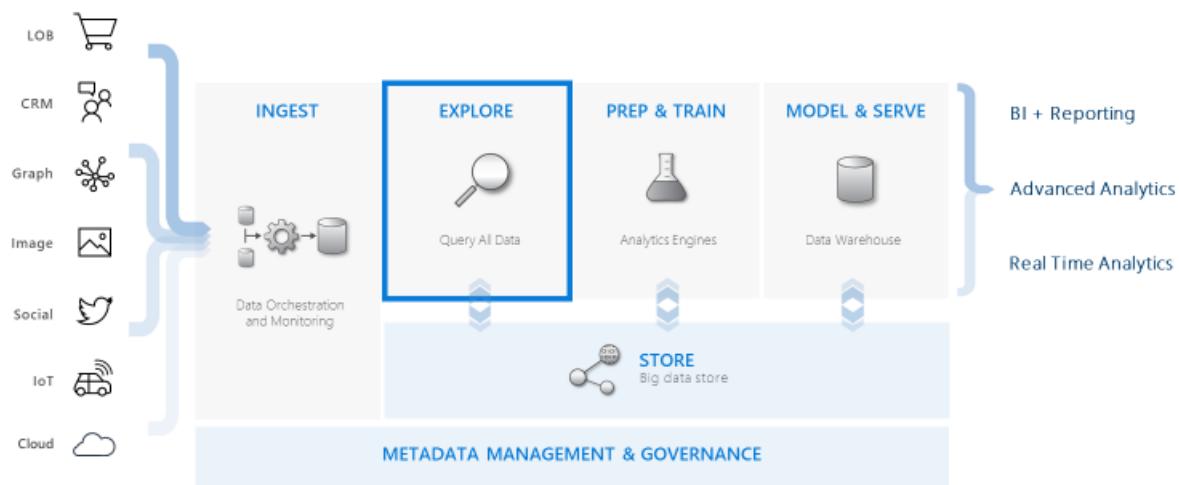
Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. It helps you handle the many data streams emitted by modern software, so you can collect, store, and analyze data. Azure Data Explorer is ideal for analyzing large volumes of diverse data from any data source, such as websites, applications, IoT devices, and more. This data is used for diagnostics, monitoring, reporting, machine learning, and additional analytics capabilities. Azure Data Explorer makes it simple to ingest this data and enables you to perform complex ad hoc queries on the data in seconds.

What makes Azure Data Explorer unique?

- Scales quickly to terabytes of data, in minutes, allowing rapid iterations of data exploration to discover relevant insights.
- Offers an innovative query language, optimized for high performance data analytics.
- Supports analysis of high volumes of heterogeneous data (structured and unstructured).
- Provides the ability to build and deploy exactly what you need by combining with other services to supply an encompassing, powerful, and interactive data analytics solution.

Data warehousing workflow

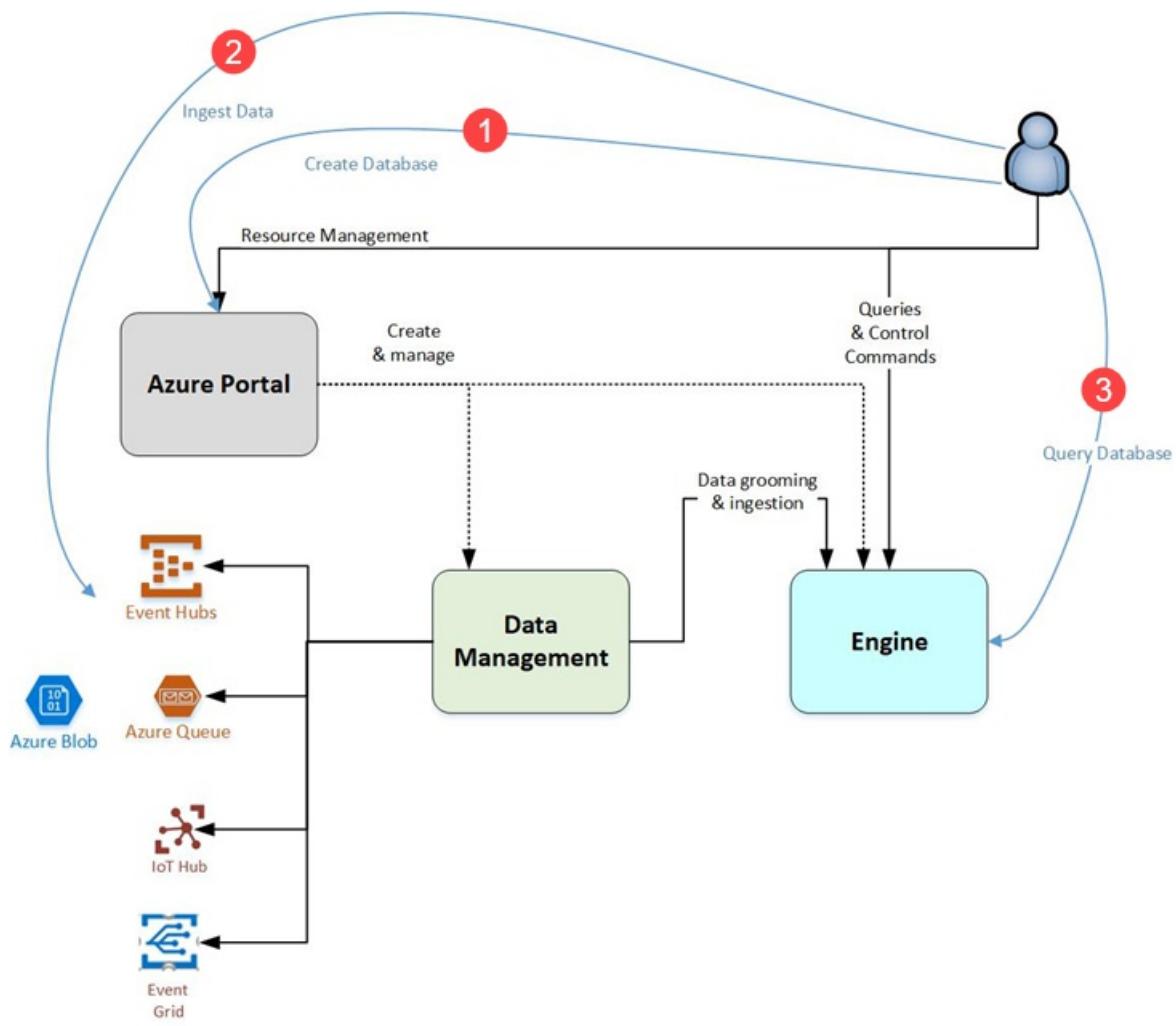
Azure Data Explorer integrates with other major services to provide an end-to-end solution that includes data collection, ingestion, storage, indexing, querying, and visualization. It has a pivotal role in the data warehousing flow by executing the **EXPLORE** step of the flow on terabytes of diverse raw data.



Azure Data Explorer supports several ingestion methods, including connectors to common services like Event Hub, programmatic ingestion using SDKs, such as .NET and Python, and direct access to the engine for exploration purposes. Azure Data Explorer integrates with analytics and modeling services for additional analysis and visualization of data.

Azure Data Explorer flow

The following diagram shows the different aspects of working with Azure Data Explorer.



Work in Azure Data Explorer generally follows this pattern:

- Create database:** Create a *cluster* and then create one or more *databases* in that cluster. [Quickstart: Create an Azure Data Explorer cluster and database](#)
- Ingest data:** Load data into database tables so that you can run queries against it. [Quickstart: Ingest data from Event Hub into Azure Data Explorer](#)
- Query database:** Use our web application to run, review, and share queries and results. It is available in the Azure portal and as a stand-alone application. In addition, you can send queries programmatically (using an SDK) or to a REST API endpoint. [Quickstart: Query data in Azure Data Explorer](#)

Query experience

A query in Azure Data Explorer is a read-only request to process data and return the results of this processing, without modifying the data or metadata. You continue refining your queries until you have completed your analysis. Azure Data Explorer makes this process easy because of its very fast ad hoc query experience.

Azure Data Explorer handles large amounts of structured, semi-structured (JSON-like nested types) and unstructured (free-text) data equally well. It allows you to search for specific text terms, locate particular events, and perform metric-style calculations on structured data. Azure Data Explorer bridges the worlds of unstructured text logs and structured numbers and dimensions by extracting values in runtime from free-form text fields. Data exploration is simplified by combining fast text indexing, column store, and time series operations.

Azure Data Explorer capabilities are extended by other services built on its powerful query language, including [Azure Monitor logs](#), [Application Insights](#), [Time Series Insights](#), and [Windows Defender Advanced Threat Protection](#).

Feedback

We would be thrilled to hear your feedback regarding Azure Data Explorer and its query language at:

- Ask questions
 - [Stack Overflow](#)
 - [Microsoft Tech Community](#)
 - [MSDN](#)
- Make product suggestions in [User Voice](#)

Next steps

[Quickstart: Create an Azure Data Explorer cluster and database](#)

[Quickstart: Ingest data from Event Hub into Azure Data Explorer](#)

[Quickstart: Query data in Azure Data Explorer](#)

Quickstart: Create an Azure Data Explorer cluster and database

9/26/2019 • 4 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. To use Azure Data Explorer, you first create a cluster, and create one or more databases in that cluster. Then you ingest (load) data into a database so that you can run queries against it. In this quickstart, you create a cluster and a database.

If you don't have an Azure subscription, create a [free Azure account](#) before you begin.

Sign in to the Azure portal

Sign in to the [Azure portal](#).

Create a cluster

Create an Azure Data Explorer cluster with a defined set of compute and storage resources in an Azure resource group.

1. Select the **Create a resource** button (+) in the upper-left corner of the portal.
2. Search for *Azure Data Explorer*.

The screenshot shows the Microsoft Azure Marketplace search interface. On the left, there's a sidebar with various service categories like Compute, Networking, Storage, Web, Mobile, Containers, Databases, Analytics, AI + Machine Learning, Internet of Things, Integration, Security, Identity, Developer Tools, Management Tools, Software as a Service (SaaS), and Blockchain. A red box highlights the 'Marketplace' button at the top of this sidebar. The main area has a search bar at the top with the text 'Azure Data Explorer'. Below the search bar are three dropdown filters: 'Pricing' (set to 'All'), 'Operating System' (set to 'All'), and 'Publisher' (set to 'All'). A red box highlights the search bar and these filters. Below the filters, the results section is titled 'Results' and shows two items:

NAME	PUBLISHER	CATEGORY
Azure Data Explorer	Microsoft	Analytics
Azure Security Center	Microsoft	Security

3. Under **Azure Data Explorer**, at the bottom of the screen, select **Create**.

4. Fill out the basic cluster details with the following information.

Create an Azure Data Explorer Cluster

* Basics Tags Review + create

PROJECT DETAILS

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

* Subscription
 └─ * Resource group ⓘ

CLUSTER DETAILS

* Cluster name ⓘ
 * Region ⓘ
 Availability zones ⓘ
 * Compute specifications ([View full pricing details](#))

Review + create **Next : Tags >**

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
Subscription	Your subscription	Select the Azure subscription that you want to use for your cluster.
Resource group	Your resource group	Use an existing resource group or create a new resource group.
Cluster name	A unique cluster name	Choose a unique name that identifies your cluster. The domain name <code>[region].kusto.windows.net</code> is appended to the cluster name you provide. The name can contain only lowercase letters and numbers. It must contain from 4 to 22 characters.
Region	West US or West US 2	Select <code>West US</code> or <code>West US 2</code> (if using availability zones) for this quickstart. For a production system, select the region that best meets your needs.

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
Availability zones	1, 2, and/or 3	Place the cluster instances in various availability zones in the same region (optional). Azure Availability Zones are unique physical locations within the same Azure region. They protect an Azure Data Explorer cluster and data from partial region failure. The cluster nodes are created, by default, in the same data center. By selecting several availability zones you can eliminate a single point of failure and ensure high availability. Deployment to availability zones is possible only during cluster creation and can't be modified at a later date.
Compute specifications	D13_v2	Select the lowest price specification for this quickstart. For a production system, select the specification that best meets your needs.

5. Select **Review + create** to review your cluster details, and **Create** to provision the cluster. Provisioning typically takes about 10 minutes.
6. When the deployment is complete, select **Go to resource**.

Microsoft.AzureKusto - Overview

Deployment

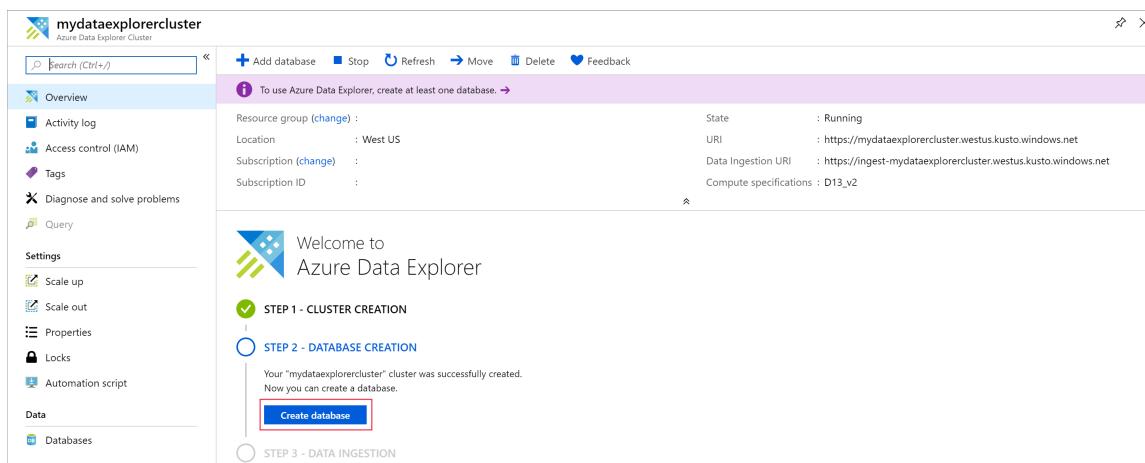
The screenshot shows the Microsoft Azure Kusto Overview page. At the top, there is a search bar labeled "Search (Ctrl+/" and action buttons for "Delete", "Cancel", "Redeploy", and "Refresh". On the left, a sidebar menu includes "Overview" (selected), "Inputs", "Outputs", and "Template". The main content area displays a message: "Your deployment is complete" with a green checkmark icon. Below this, there is a "Go to resource" button with a red border. To the right of the button, deployment details are listed: "Deployment name: Microsoft.AzureKusto", "Subscription:", and "Resource group:". Further down, "DEPLOYMENT DETAILS" are provided with a download link, showing start time (2/20/2019, 2:20:10 PM), duration (9 minutes 11 seconds), and correlation ID (ab40e65e-d4ba-43af-9b25-fc3e3bd961f4). At the bottom, a table lists resources with columns: RESOURCE, TYPE, STATUS, and OPERATI... (partially visible). One row is shown: "mydataexpl" (RESOURCE), "Microsoft..." (TYPE), "OK" (STATUS), and "Operation c" (OPERATI...).

RESOURCE	TYPE	STATUS	OPERATI...
mydataexpl	Microsoft...	OK	Operation c

Create a database

You're now ready for the second step in the process: database creation.

1. On the **Overview** tab, select **Create database**.



The screenshot shows the Azure Data Explorer Cluster Overview page for 'mydataexplorercluster'. The cluster is listed with the following details:

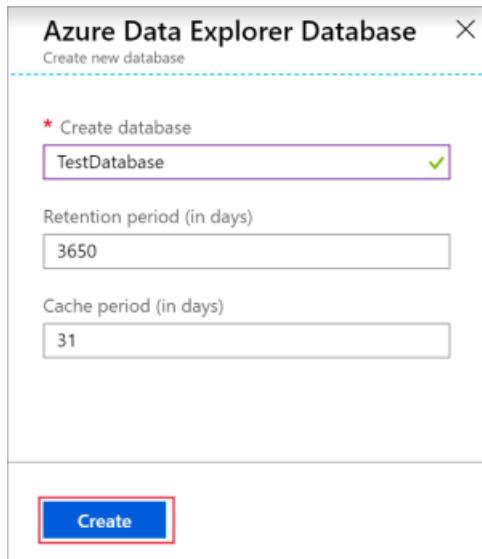
- Resource group: mydataexplorercluster
- Location: West US
- Subscription: (change)
- Subscription ID: (change)
- State: Running
- URI: https://mydataexplorercluster.westus.kusto.windows.net
- Data Ingestion URI: https://ingest-mydataexplorercluster.westus.kusto.windows.net
- Compute specifications: D13_v2

The main content area displays a welcome message and a step-by-step guide:

- STEP 1 - CLUSTER CREATION** (Completed)
- STEP 2 - DATABASE CREATION** (In Progress)
- STEP 3 - DATA INGESTION** (Not Started)

A message indicates that the cluster was successfully created, and a red-bordered **Create database** button is visible.

2. Fill out the form with the following information.



The 'Azure Data Explorer Database' dialog box is open, showing the 'Create new database' section. The 'Create database' field is populated with 'TestDatabase'. The 'Retention period (in days)' field is set to '3650'. The 'Cache period (in days)' field is set to '31'. A red-bordered **Create** button is at the bottom.

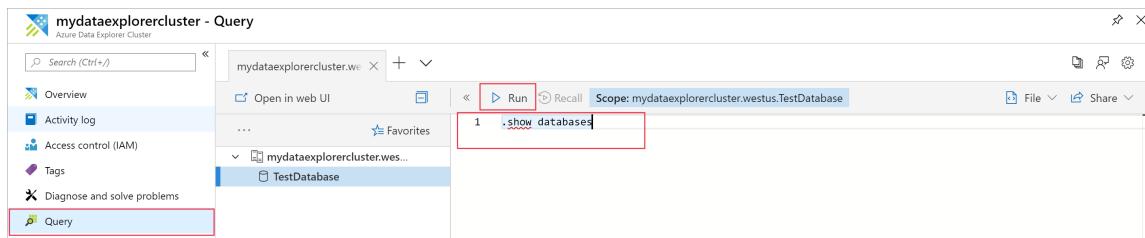
SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
Database name	TestDatabase	The database name must be unique within the cluster.
Retention period	3650	The time span (in days) for which it's guaranteed that the data is kept available to query. The time span is measured from the time that data is ingested.
Cache period	31	The time span (in days) for which to keep frequently queried data available in SSD storage or RAM, rather than in longer-term storage.

3. Select **Create** to create the database. Creation typically takes less than a minute. When the process is complete, you're back on the cluster **Overview** tab.

Run basic commands in the database

Now that you have a cluster and database, you can run queries and commands. You don't have any data in the database yet, but you can still see how the tools work.

1. Under your cluster, select **Query**. Paste the command `.show databases` into the query window, then select **Run**.



The result set shows **TestDatabase**, the only database in the cluster.

2. Paste the command `.show tables` into the query window and select **Run**.

This command returns an empty result set because you don't have any tables yet. You add a table in the next article in this series.

Stop and restart the cluster

You can stop and restart a cluster depending on business needs.

1. To stop the cluster, at the top of the **Overview** tab, select **Stop**.

When the cluster is stopped, data is not available for queries, and you can't ingest new data.

2. To restart the cluster, at the top of the **Overview** tab, select **Start**.

When the cluster is restarted, it takes about 10 minutes for it to become available (like when it was originally provisioned). It takes additional time for data to load into the hot cache.

Clean up resources

If you plan to follow other quickstarts and tutorials, keep the resources you created. Otherwise, clean up your resource group, to avoid incurring costs.

1. In the Azure portal, select **Resource groups** on the far left, and then select the resource group that contains your Data Explorer cluster.
2. Select **Delete resource group** to delete the entire resource group. If using an existing resource group, you can choose to only delete the Data Explorer cluster.

Next steps

[Quickstart: Ingest data from Event Hub into Azure Data Explorer](#)

Quickstart: Ingest sample data into Azure Data Explorer

12/10/2019 • 2 minutes to read • [Edit Online](#)

This article shows you how to ingest (load) sample data into an Azure Data Explorer database. There are [several ways to ingest data](#); this article focuses on a basic approach that is suitable for testing purposes.

NOTE

You already have this data if you completed [Ingest data using the Azure Data Explorer Python library](#).

Prerequisites

[A test cluster and database](#)

Ingest data

The **StormEvents** sample data set contains weather-related data from the [National Centers for Environmental Information](#).

1. Sign in to <https://dataexplorer.azure.com>.
2. In the upper-left of the application, select **Add cluster**.
3. In the **Add cluster** dialog box, enter your cluster URL in the form
`https://<ClusterName>.<Region>.kusto.windows.net/`, then select **Add**.
4. Paste in the following command, and select **Run** to create a StormEvents table.

```
.create table StormEvents (StartTime: datetime, EndTime: datetime, EpisodeId: int, EventId: int, State: string, EventType: string, InjuriesDirect: int, InjuriesIndirect: int, DeathsDirect: int, DeathsIndirect: int, DamageProperty: int, DamageCrops: int, Source: string, BeginLocation: string, EndLocation: string, BeginLat: real, BeginLon: real, EndLat: real, EndLon: real, EpisodeNarrative: string, EventNarrative: string, StormSummary: dynamic)
```

5. Paste in the following command, and select **Run** to ingest data into StormEvents table.

```
.ingest into table StormEvents  
h'https://kustosamplefiles.blob.core.windows.net/samplefiles/StormEvents.csv?st=2018-08-  
31T22%3A02%3A25Z&se=2020-09-01T22%3A02%3A00Z&sp=r&sv=2018-03-  
28&sr=b&sig=LQIbomcKI80oz425hWtjeq6d61uEaq21UVX7YrM61N4%3D' with (ignoreFirstRecord=true)
```

6. After ingestion completes, paste in the following query, select the query in the window, and select **Run**.

```
StormEvents  
| sort by StartTime desc  
| take 10
```

The query returns the following results from the ingested sample data.

Table 1 Stats

StartTime	EndTime	Episode...	EventId	State	EventType
2007-12-31T23:53:00Z	2007-12-31T23:53:00Z	12,037	65,838	CALIFORNIA	High Wind
2007-12-31T23:53:00Z	2007-12-31T23:53:00Z	12,037	65,839	CALIFORNIA	High Wind
2007-12-31T22:30:00Z	2007-12-31T23:59:00Z	12,950	71,588	MICHIGAN	Winter Storm
2007-12-31T22:30:00Z	2007-12-31T23:59:00Z	12,950	71,589	MICHIGAN	Winter Storm
2007-12-31T22:30:00Z	2007-12-31T23:59:00Z	12,950	71,590	MICHIGAN	Winter Storm
2007-12-31T20:30:00Z	2007-12-31T23:59:00Z	12,950	71,586	MICHIGAN	Winter Storm
2007-12-31T20:30:00Z	2007-12-31T23:59:00Z	12,950	71,587	MICHIGAN	Winter Storm
2007-12-31T19:00:00Z	2007-12-31T23:59:00Z	12,994	71,915	ALASKA	High Wind
2007-12-31T19:00:00Z	2007-12-31T23:59:00Z	12,994	71,917	ALASKA	High Wind
2007-12-31T18:30:00Z	2007-12-31T23:59:00Z	13,007	71,979	CALIFORNIA	High Wind

Next steps

- [Azure Data Explorer data ingestion](#) to learn more about ingestion methods.
- [Quickstart: Query data in Azure Data Explorer Web UI](#).
- [Write queries with Kusto Query Language](#).

Quickstart: Query data in Azure Data Explorer Web UI

7/10/2019 • 4 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Azure Data explorer provides a web application that enables you to run and share queries. The application is available in the Azure portal and as a stand-alone web application. In this article, you work in the stand-alone version, which enables you to connect to multiple clusters and to share deep links to your queries.

If you don't have an Azure subscription, create a [free Azure account](#) before you begin.

Prerequisites

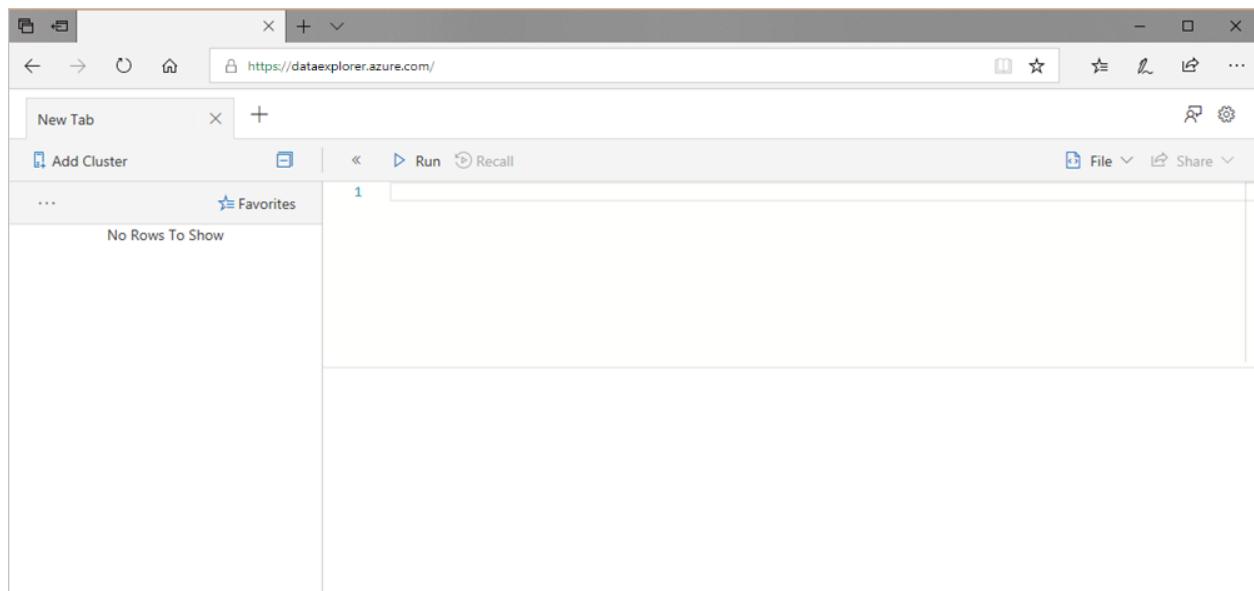
In addition to an Azure subscription, you need a [test cluster and database](#) to complete this quickstart.

Sign in to the application

Sign in to [the application](#).

Add clusters

When you first open the application, there are no connections.



You must add a connection to at least one cluster before you can start running queries. In this section, you add connections to the Azure Data Explorer *help cluster* that we have set up to aid learning, and to the test cluster you created in a previous quickstart.

1. In the upper-left of the application, select **Add cluster**.
2. In the **Add cluster** dialog box, enter the URI, then select **Add**.

You may use the help cluster URI, <https://help.kusto.windows.net>. If you have your own cluster, provide the URI of your cluster. For example, <https://mydataexplorercluster.westus.kusto.windows.net> as in the following image:

3. In the left pane, you should now see the **help** cluster. Expand the **Samples** database so that you can see the sample tables that you have access to.

We use the **StormEvents** table later in this quickstart, and in other Azure Data Explorer articles.

Now add the test cluster you created.

1. Select **Add cluster**.
2. In the **Add cluster** dialog box, enter your test cluster URL in the form
`https://<ClusterName>.<Region>.kusto.windows.net/`, then select **Add**.

In the example below, you see the **help** cluster and a new cluster, **docscluster.westus** (full URL is
`https://docscluster.westus.kusto.windows.net/`).

Run queries

You can now run queries against either cluster that you're connected to (assuming you have data in your test cluster). We'll focus on the **help** cluster.

1. In the left pane, under the **help** cluster, select the **Samples** database.
2. Copy and paste the following query into the query window. At the top of the window, select **Run**.

```
StormEvents
| sort by StartTime desc
| take 10
```

This query returns the ten newest records in the **StormEvents** table. The left side of the result should look like the following table.

Table 1 Stats

StartTime	EndTime	Episode...	EventId	State	EventType
2007-12-31T23:53:00Z	2007-12-31T23:53:00Z	12,037	65,838	CALIFORNIA	High Wind
2007-12-31T23:53:00Z	2007-12-31T23:53:00Z	12,037	65,839	CALIFORNIA	High Wind
2007-12-31T22:30:00Z	2007-12-31T23:59:00Z	12,950	71,588	MICHIGAN	Winter Storm
2007-12-31T22:30:00Z	2007-12-31T23:59:00Z	12,950	71,589	MICHIGAN	Winter Storm
2007-12-31T22:30:00Z	2007-12-31T23:59:00Z	12,950	71,590	MICHIGAN	Winter Storm
2007-12-31T20:30:00Z	2007-12-31T23:59:00Z	12,950	71,586	MICHIGAN	Winter Storm
2007-12-31T20:30:00Z	2007-12-31T23:59:00Z	12,950	71,587	MICHIGAN	Winter Storm
2007-12-31T19:00:00Z	2007-12-31T23:59:00Z	12,994	71,915	ALASKA	High Wind
2007-12-31T19:00:00Z	2007-12-31T23:59:00Z	12,994	71,917	ALASKA	High Wind
2007-12-31T18:30:00Z	2007-12-31T23:59:00Z	13,007	71,979	CALIFORNIA	High Wind

The following image shows the state that the application should now be in, with clusters added, and a query with results.

The screenshot shows the Azure Data Explorer interface. On the left, the navigation pane displays a tree structure with 'help.Samples' selected. Under 'Samples', there are several items: 'Functions', 'ForecastExample', 'StormEvents', 'tsa_example_data', and 'docscluster.westus'. In the main pane, a query is being run:

```
1 StormEvents
| sort by StartTime desc
| take 10
```

The results of the query are displayed in a table titled 'Table 1 Stats'.

StartTime	EndTime	EpisodId	EventId
2007-12-31T23:53:00Z	2007-12-31T23:53:00Z	12,037	65,838
2007-12-31T23:53:00Z	2007-12-31T23:53:00Z	12,037	65,839
2007-12-31T23:53:00Z	2007-12-31T23:53:00Z	12,037	65,839
2007-12-31T23:53:00Z	2007-12-31T23:53:00Z	12,037	65,838
2007-12-31T22:30:00Z	2007-12-31T23:59:00Z	12,950	71,589
2007-12-31T22:30:00Z	2007-12-31T23:59:00Z	12,950	71,590

3. Copy and paste the following query into the query window, below the first query. Notice how it's not formatted on separate lines like the first query.

```
StormEvents | sort by StartTime desc | project StartTime, EndTime, State, EventType, DamageProperty, EpisodeNarrative | take 10
```

4. Click the new query in the window, which selects the query. Press Shift+Alt+F to format the query, so it looks like the following.

```
StormEvents
| sort by StartTime desc
| project StartTime, EndTime, State, EventType, DamageProperty, EpisodeNarrative
| take 10
```

5. Press Shift+Enter, which is a shortcut to run a query.

This query returns the same records as the first one, but includes only the columns specified in the `project` statement. The result should look like the following table.

StartTime	EndTime	State	EventType	DamageProperty	EpisodeNarrative
2007-12-31T23:53:00Z	2007-12-31T23:53:00Z	CALIFORNIA	High Wind	0	One last round of offshore win...
2007-12-31T23:53:00Z	2007-12-31T23:53:00Z	CALIFORNIA	High Wind	0	One last round of offshore win...
2007-12-31T22:30:00Z	2007-12-31T23:59:00Z	MICHIGAN	Winter Storm	0	Heavy snow moved into southe...
2007-12-31T22:30:00Z	2007-12-31T23:59:00Z	MICHIGAN	Winter Storm	0	Heavy snow moved into southe...
2007-12-31T22:30:00Z	2007-12-31T23:59:00Z	MICHIGAN	Winter Storm	0	Heavy snow moved into southe...
2007-12-31T20:30:00Z	2007-12-31T23:59:00Z	MICHIGAN	Winter Storm	0	Heavy snow moved into southe...
2007-12-31T20:30:00Z	2007-12-31T23:59:00Z	MICHIGAN	Winter Storm	0	Heavy snow moved into southe...
2007-12-31T19:00:00Z	2007-12-31T23:59:00Z	ALASKA	High Wind	0	On the morning of 12/31 a stor...
2007-12-31T19:00:00Z	2007-12-31T23:59:00Z	ALASKA	High Wind	0	On the morning of 12/31 a stor...
2007-12-31T18:30:00Z	2007-12-31T23:59:00Z	CALIFORNIA	High Wind	0	Offshore flow resulted in high ...

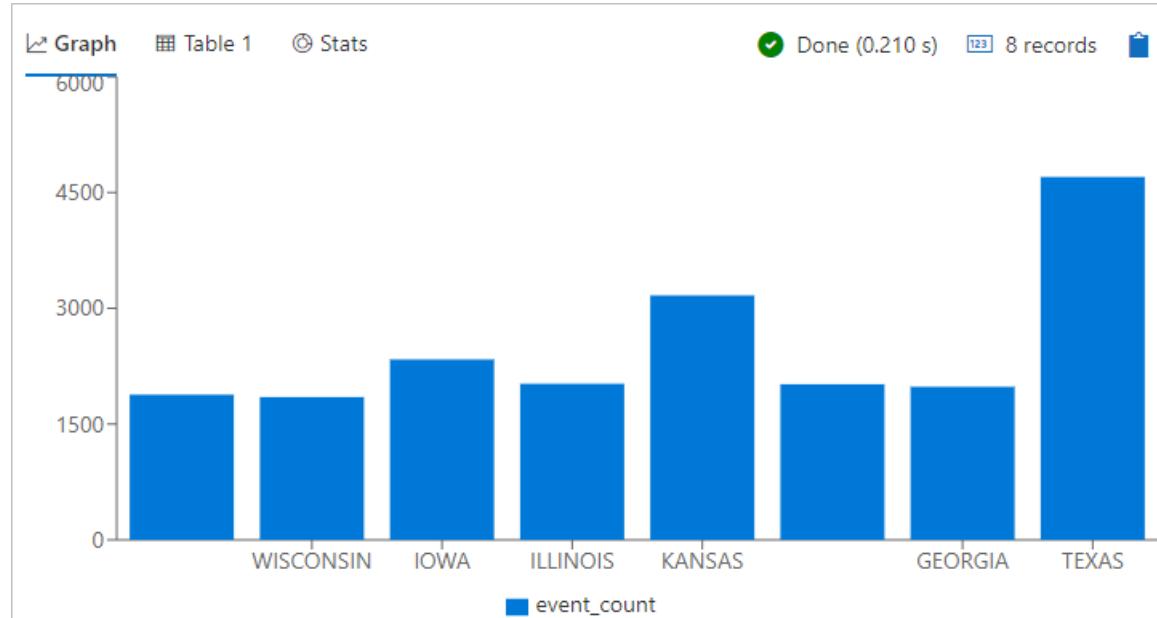
- At the top of the query window, select **Recall**.

The query window now shows the result set from the first query without having to rerun the query. Often during analysis, you run multiple queries, and **Recall** enables you to revisit the results of previous queries.

- Let's run one more query to see a different type of output.

```
StormEvents
| summarize event_count=count(), mid = avg(BeginLat) by State
| sort by mid
| where event_count > 1800
| project State, event_count
| render columnchart
```

The result should look like the following chart.



Work with the table grid

Now you've seen how basic queries work, let's look at how you can use the table grid to customize results and do further analysis.

- Rerun the first query. Mouse-over the **State** column, select the menu, and select **Group by State**.

State	
CALIFORNIA	
CALIFORNIA	
MICHIGAN	
MICHIGAN	
MICHIGAN	
MICHIGAN	
ALASKA	
ALASKA	
CALIFORNIA	

2. In the grid, expand **California** to see records for that state.

Table 1 Stats						
Group	StartTime	EndTime	EpisodeId	EventId	State	EventType
▼ CALIFORNIA (3)						
	2007-12-31T23:53:00Z	2007-12-31T23:53:00Z	12,037	65,838	CALIFORNIA	High Wind
	2007-12-31T23:53:00Z	2007-12-31T23:53:00Z	12,037	65,839	CALIFORNIA	High Wind
	2007-12-31T18:30:00Z	2007-12-31T23:59:00Z	13,007	71,979	CALIFORNIA	High Wind
➤ MICHIGAN (5)						
➤ ALASKA (2)						

This type of grouping can be helpful when doing exploratory analysis.

3. Mouse-over the **Group** column, then select **Reset columns**.

Group	Time
▼ CALIFORNIA	
➤ MICHIGAN	
➤ ALASKA	

This returns the grid to its original state.

4. Run the following query.

```
StormEvents
| sort by StartTime desc
| where DamageProperty > 5000
| project StartTime, State, EventType, DamageProperty, Source
| take 10
```

5. On the right side of the grid, select **Columns** to see the tool panel.

This panel functions similarly to the pivot table field list in Excel, enabling you to do more analysis in the grid itself.

6. Select **Pivot Mode**, then drag columns as follows: **State** to **Row groups**; **DamageProperty** to **Values**; and **EventType** to **Column labels**.

The result should look like the following pivot table.

Table 1 Stats					
	Hail	Heavy Snow	Lightning	Thunderstorm Wind	Winter Weather
Group	sum(DamageProperty)	sum(DamageProperty)	sum(DamageProperty)	sum(DamageProperty)	sum(DamageProperty)
VERMONT (2)	20,000				
GEORGIA (1)				25,000	
ALABAMA (2)	45,000				
TENNESSEE (1)		100,000			
MISSISSIPPI (1)		45,000			
NEW HAMPSHIRE (1)					30,000
TEXAS (2)	75,000				50,000

Notice how Vermont and Alabama each have two events under the same category, while Texas has two events under different categories. Pivot tables enable you to quickly spot things like this; they are a great

tool for quick analysis.

Share queries

Many times, you want to share the queries you create. You can provide a deep link so that other users with access to the cluster can run the queries.

1. In the query window, select the first query you copied in.
2. At the top of the query window, select **Share**.
3. Select **Link, query to clipboard**.
4. Copy the link and query to a text file.
5. Paste the link into a new browser window. The result should look like the following after the query runs.

The screenshot shows the Azure Data Explorer interface. At the top, there is a code editor containing a query:

```
1  StormEvents
2  | sort by StartTime desc
3  | take 10
```

Below the code editor is a table titled "Table 1". The table has four columns: StartTime, EndTime, EpisodeId, and EventId. The data consists of 10 records, each with a timestamp between December 31, 2007, and January 1, 2008, and IDs ranging from 12,037 to 13,007. The table includes standard data grid controls like sorting arrows and a "Columns" dropdown. At the bottom right of the table, it says "Done (0.620 s)" and "10 records".

StartTime	EndTime	EpisodeId	EventId
2007-12-31T23:53:00Z	2007-12-31T23:53:00Z	12,037	65,838
2007-12-31T23:53:00Z	2007-12-31T23:53:00Z	12,037	65,839
2007-12-31T22:30:00Z	2007-12-31T23:59:00Z	12,950	71,588
2007-12-31T22:30:00Z	2007-12-31T23:59:00Z	12,950	71,589
2007-12-31T22:30:00Z	2007-12-31T23:59:00Z	12,950	71,590
2007-12-31T20:30:00Z	2007-12-31T23:59:00Z	12,950	71,586
2007-12-31T20:30:00Z	2007-12-31T23:59:00Z	12,950	71,587
2007-12-31T19:00:00Z	2007-12-31T23:59:00Z	12,994	71,915
2007-12-31T19:00:00Z	2007-12-31T23:59:00Z	12,994	71,917
2007-12-31T18:30:00Z	2007-12-31T23:59:00Z	13,007	71,979

Provide feedback

Data Explorer is currently in preview, and we welcome feedback on your experience. You can do this now or wait until you've spent more time with it.

1. In the upper-right of the application, select the feedback icon:
2. Enter your feedback, then select **Submit**.

Clean up resources

You didn't create any resources in this quickstart, but if you'd like to remove one or both clusters from the application, right-click the cluster and select **Remove connection**.

Next steps

[Write queries for Azure Data Explorer](#)

End-to-end blob ingestion into Azure Data Explorer through C#

11/4/2019 • 6 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and scalable data exploration service for log and telemetry data. This article gives you an end-to-end example of how to ingest data from Azure Blob storage into Azure Data Explorer.

You'll learn how to programmatically create a resource group, a storage account and container, an event hub, and an Azure Data Explorer cluster and database. You'll also learn how to programmatically configure Azure Data Explorer to ingest data from the new storage account.

Prerequisites

If you don't have an Azure subscription, create a [free Azure account](#) before you begin.

Install C# NuGet

- Install [Microsoft.Azure.Management.kusto](#).
- Install [Microsoft.Azure.Management.ResourceManager](#).
- Install [Microsoft.Azure.Management.EventGrid](#).
- Install [Microsoft.Azure.Storage.Blob](#).
- Install [Microsoft.Rest.ClientRuntime.Azure.Authentication](#) for authentication.

Authentication

To run the following example, you need an Azure Active Directory (Azure AD) application and service principal that can access resources. To create a free Azure AD application and add role assignment at the subscription level, see [Create an Azure AD application](#). You also need the directory (tenant) ID, application ID, and client secret.

Azure Resource Manager template

In this article, you use an Azure Resource Manager template to create a resource group, a storage account and container, an event hub, and an Azure Data Explorer cluster and database. Save the following content in a file with the name `template.json`. You'll use this file to run the code example.

```
{  
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",  
    "contentVersion": "1.0.0.0",  
    "parameters": {  
        "eventHubNamespaceName": {  
            "type": "string",  
            "metadata": {  
                "description": "Specifies a the event hub Namespace name."  
            }  
        },  
        "eventHubName": {  
            "type": "string",  
            "metadata": {  
                "description": "Specifies a event hub name."  
            }  
        },  
        "storageAccountType": {  
            "type": "string"  
        }  
    },  
    "resources": [  
        {  
            "type": "Microsoft.EventHub/eventHubs",  
            "name": "[parameters('eventHubName')]",  
            "apiVersion": "2017-04-01",  
            "location": "West US",  
            "dependsOn": [  
                "[resourceId('Microsoft.EventHub/namespaces', parameters('eventHubNamespaceName'))]"  
            ],  
            "properties": {  
                "partitionCount": 1,  
                "status": "Enabled"  
            }  
        },  
        {  
            "type": "Microsoft.DataExplorer/clusters",  
            "name": "cluster1",  
            "apiVersion": "2018-06-01",  
            "location": "West US",  
            "dependsOn": [  
                "[resourceId('Microsoft.EventHub/namespaces', parameters('eventHubNamespaceName'))]"  
            ],  
            "properties": {  
                "eventHubName": "[parameters('eventHubName')]",  
                "maxDegreeOfParallelism": 1,  
                "minDegreeOfParallelism": 1  
            }  
        },  
        {  
            "type": "Microsoft.DataExplorer/databases",  
            "name": "database1",  
            "apiVersion": "2018-06-01",  
            "location": "West US",  
            "dependsOn": [  
                "[resourceId('Microsoft.DataExplorer/clusters', 'cluster1')]"  
            ],  
            "properties": {  
                "clusterName": "cluster1",  
                "maxDegreeOfParallelism": 1,  
                "minDegreeOfParallelism": 1  
            }  
        },  
        {  
            "type": "Microsoft.Storage/storageAccounts",  
            "name": "storageaccount1",  
            "apiVersion": "2018-06-01",  
            "location": "West US",  
            "dependsOn": [  
                "[resourceId('Microsoft.DataExplorer/databases', 'database1')]"  
            ],  
            "properties": {  
                "blobEndpoint": "https://storageaccount1.blob.core.windows.net/",  
                "containerSasTtl": 3600,  
                "leaseSasTtl": 3600,  
                "fileEndpoint": "https://storageaccount1.file.core.windows.net/",  
                "httpSasTtl": 3600,  
                "httpsSasTtl": 3600,  
                "queueEndpoint": "https://storageaccount1.queue.core.windows.net/",  
                "snapshotSasTtl": 3600,  
                "storageType": "Standard_LRS"  
            }  
        }  
    ]  
}
```

```
        "type": "string",
        "defaultValue": "Standard_LRS",
        "allowedValues": ["Standard_LRS", "Standard_GRS", "Standard_ZRS", "Premium_LRS"],
        "metadata": {
            "description": "Storage Account type"
        }
    },
    "storageAccountName": {
        "type": "string",
        "defaultValue": "[concat('storage', uniqueString(resourceGroup().id))]",
        "metadata": {
            "description": "Name of the storage account to create"
        }
    },
    "containerName": {
        "type": "string",
        "defaultValue": "[concat('storagecontainer', uniqueString(resourceGroup().id))]",
        "metadata": {
            "description": "Name of the container in storage account to create"
        }
    },
    "eventHubSku": {
        "type": "string",
        "allowedValues": ["Basic", "Standard"],
        "defaultValue": "Standard",
        "metadata": {
            "description": "Specifies the messaging tier for service Bus namespace."
        }
    },
    "kustoClusterName": {
        "type": "string",
        "defaultValue": "[concat('kusto', uniqueString(resourceGroup().id))]",
        "metadata": {
            "description": "Name of the cluster to create"
        }
    },
    "kustoDatabaseName": {
        "type": "string",
        "defaultValue": "kustodb",
        "metadata": {
            "description": "Name of the database to create"
        }
    },
    "location": {
        "type": "string",
        "defaultValue": "[resourceGroup().location]",
        "metadata": {
            "description": "Location for all resources."
        }
    },
    "variables": {},
    "resources": [
        {
            "apiVersion": "2017-04-01",
            "type": "Microsoft.EventHub/namespaces",
            "name": "[parameters('eventHubNamespaceName')]",
            "location": "[parameters('location')]",
            "sku": {
                "name": "[parameters('eventHubSku')]",
                "tier": "[parameters('eventHubSku')]",
                "capacity": 1
            },
            "properties": {
                "isAutoInflateEnabled": false,
                "maximumThroughputUnits": 0
            }
        },
        {
            "apiVersion": "2017-04-01",
            "type": "Microsoft.EventHub/namespaces/partitions"
        }
    ]
}
```

```

    "type": "Microsoft.EventHub/namespaces/eventhubs",
    "name": "[concat(parameters('eventHubNamespaceName'), '/', parameters('eventHubName'))]",
    "location": "[parameters('location')]",
    "dependsOn": ["[resourceId('Microsoft.EventHub/namespaces',
parameters('eventHubNamespaceName'))]"],
    "properties": {
        "messageRetentionInDays": 7,
        "partitionCount": 1
    }
}, {
    "type": "Microsoft.Storage/storageAccounts",
    "name": "[parameters('storageAccountName')]",
    "location": "[parameters('location')]",
    "apiVersion": "2018-07-01",
    "sku": {
        "name": "[parameters('storageAccountType')]"
    },
    "kind": "StorageV2",
    "resources": [
        {
            "name": "[concat('default/', parameters('containerName'))]",
            "type": "blobServices/containers",
            "apiVersion": "2018-07-01",
            "dependsOn": [
                "[parameters('storageAccountName')]"
            ],
            "properties": {
                "publicAccess": "None"
            }
        }
    ],
    "properties": {}
}, {
    "name": "[parameters('kustoClusterName')]",
    "type": "Microsoft.Kusto/clusters",
    "sku": {
        "name": "Standard_D13_v2",
        "tier": "Standard",
        "capacity": 2
    },
    "apiVersion": "2019-09-07",
    "location": "[parameters('location')]",
    "tags": {
        "Created By": "GitHub quickstart template"
    }
}, {
    "name": "[concat(parameters('kustoClusterName'), '/', parameters('kustoDatabaseName'))]",
    "type": "Microsoft.Kusto/clusters/databases",
    "apiVersion": "2019-09-07",
    "location": "[parameters('location')]",
    "dependsOn": ["[resourceId('Microsoft.Kusto/clusters', parameters('kustoClusterName'))]"],
    "properties": {
        "softDeletePeriodInDays": 365,
        "hotCachePeriodInDays": 31
    }
}
]
}

```

Code example

The following code example gives you a step-by-step process that results in data ingestion into Azure Data Explorer.

You first create a resource group. You also create Azure resources such as a storage account and container, an

event hub, and an Azure Data Explorer cluster and database. You then create an Azure Event Grid subscription, along with a table and column mapping, in the Azure Data Explorer database. Finally, you create the data connection to configure Azure Data Explorer to ingest data from the new storage account.

```
var tenantId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx";//Directory (tenant) ID
var clientId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx";//Application ID
var clientSecret = "xxxxxxxxxxxxxx";//Client secret
var subscriptionId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx";
string location = "West Europe";
string locationSmallCase = "westeurope";
string azureResourceTemplatePath = @"xxxxxxxx\template.json";//Path to the Azure Resource Manager template
JSON from the previous section

string deploymentName = "e2eexample";
string resourceGroupName = deploymentName + "resourcegroup";
string eventHubName = deploymentName + "eventhub";
string eventHubNamespaceName = eventHubName + "ns";
string storageAccountName = deploymentName + "storage";
string storageContainerName = deploymentName + "storagecontainer";
string eventGridSubscriptionName = deploymentName + "eventgrid";
string kustoClusterName = deploymentName + "kustocluster";
string kustoDatabaseName = deploymentName + "kustodatabase";
string kustoTableName = "Events";
string kustoColumnMappingName = "Events_CSV_Mapping";
string kustoDataConnectionName = deploymentName + "kustoeventgridconnection";

var serviceCreds = await ApplicationTokenProvider.LoginSilentAsync(tenantId, clientId, clientSecret);
var resourceManagementClient = new ResourceManagementClient(serviceCreds);
Console.WriteLine("Step 1: Create a new resource group in your Azure subscription to manage all the resources for using Azure Data Explorer.");
resourceManagementClient.SubscriptionId = subscriptionId;
await resourceManagementClient.ResourceGroups.CreateOrUpdateAsync(resourceGroupName,
    new ResourceGroup() { Location = locationSmallCase });

Console.WriteLine(
    "Step 2: Create a Blob Storage, a container in the Storage account, an Event Hub, an Azure Data Explorer cluster, and database by using an Azure Resource Manager template.");
var parameters = $"\"{{\\\"eventHubNamespaceName\\\":{{\\\"value\\\":\\\"{eventHubNamespaceName}\\\"}}},\\\"eventHubName\\\":{{\\\"value\\\":\\\"{eventHubName}\\\"}},\\\"storageAccountName\\\":{{\\\"value\\\":\\\"{storageAccountName}\\\"}},\\\"kustoClusterName\\\":{{\\\"value\\\":\\\"{kustoClusterName}\\\"}},\\\"kustoDatabaseName\\\":{{\\\"value\\\":\\\"{kustoDatabaseName}\\\"}}}";
string template = File.ReadAllText(azureResourceTemplatePath, Encoding.UTF8);
await resourceManagementClient.Deployments.CreateOrUpdateAsync(resourceGroupName, deploymentName,
    new Deployment(new DeploymentProperties(DeploymentMode.Incremental, template: template,
    parameters: parameters)));

Console.WriteLine(
    "Step 3: Create an Event Grid subscription to publish blob events created in a specific container to an Event Hub.");
var eventGridClient = new EventGridManagementClient(serviceCreds)
{
    SubscriptionId = subscriptionId
};
string storageResourceId =
$"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/{storageAccountName}";
string eventHubResourceId =
$"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.EventHub/namespaces/{eventHubNamespaceName}/eventhubs/{eventHubName}";
await eventGridClient.EventSubscriptions.CreateOrUpdateAsync(storageResourceId, eventGridSubscriptionName,
    new EventSubscription()
    {
        Destination = new EventHubEventSubscriptionDestination(eventHubResourceId),
        Filter = new EventSubscriptionFilter()
        {
            SubjectBeginsWith = $"/blobServices/default/containers/{storageContainerName}",
            IncludedEventTypes = new List<string>(){ "Microsoft.Storage.BlobCreated" }
        }
    }
}
```

```

        }

    });

Console.WriteLine("Step 4: Create a table (with three columns: EventTime, EventId, and EventSummary) and column mapping in your Azure Data Explorer database.");
var kustoUri = $"https://{{kustoClusterName}}.{{locationSmallCase}}.kusto.windows.net";
var kustoConnectionStringBuilder = new KustoConnectionStringBuilder(kustoUri)
{
    InitialCatalog = kustoDatabaseName,
    FederatedSecurity = true,
    ApplicationClientId = clientId,
    ApplicationKey = clientSecret,
    Authority = tenantId
};

using (var kustoClient = KustoClientFactory.CreateCslAdminProvider(kustoConnectionStringBuilder))
{
    var command =
        CslCommandGenerator.GenerateTableCreateCommand(
            kustoTableName,
            new[]
            {
                Tuple.Create("EventTime", "System.DateTime"),
                Tuple.Create("EventId", "System.Int32"),
                Tuple.Create("EventSummary", "System.String"),
            });
    kustoClient.ExecuteControlCommand(command);

    command = CslCommandGenerator.GenerateTableCsvMappingCreateCommand(
        kustoTableName,
        kustoColumnMappingName,
        new[]
        {
            new CsvColumnMapping { ColumnName = "EventTime", CslDataType="dateTime", Ordinal = 0 },
            new CsvColumnMapping { ColumnName = "EventId", CslDataType="int", Ordinal = 1 },
            new CsvColumnMapping { ColumnName = "EventSummary", CslDataType="string", Ordinal = 2 },
        });
    kustoClient.ExecuteControlCommand(command);
}

Console.WriteLine("Step 5: Add an Event Grid data connection. Azure Data Explorer will automatically ingest the data when new blobs are created.");
var kustoManagementClient = new KustoManagementClient(serviceCreds)
{
    SubscriptionId = subscriptionId
};
await kustoManagementClient.DataConnections.CreateOrUpdateAsync(resourceGroupName, kustoClusterName,
    kustoDatabaseName, dataConnectionName: kustoDataConnectionName, new
EventGridDataConnection(storageResourceId, eventHubResourceId, consumerGroup: "$Default", location: location,
    tableName:kustoTableName, mappingRuleName: kustoColumnMappingName, dataFormat: "csv"));

```

SETTING	FIELD DESCRIPTION
tenantId	Your tenant ID. It's also known as a directory ID.
subscriptionId	The subscription ID that you use for resource creation.
clientId	The client ID of the application that can access resources in your tenant.
clientSecret	The client secret of the application that can access resources in your tenant.

Test the code example

1. Upload a file into the storage account.

```
string storageConnectionString =
    "DefaultEndpointsProtocol=https;AccountName=xxxxxxxxxxxxxx;AccountKey=xxxxxxxxxxxxxx;EndpointSuffix=core
.windows.net";
var cloudStorageAccount = CloudStorageAccount.Parse(storageConnectionString);
CloudBlobClient blobClient = cloudStorageAccount.CreateCloudBlobClient();
CloudBlobContainer container = blobClient.GetContainerReference(storageContainerName);
CloudBlockBlob blockBlob = container.GetBlockBlobReference("test.csv");
var blobContent = @"2007-01-01 00:00:00.000000,2592,Several trees down
2007-01-01 00:00:00.000000,4171,Winter Storm";
await blockBlob.UploadTextAsync(blobContent);
```

SETTING	FIELD DESCRIPTION
storageConnectionString	The connection string of the programmatically created storage account.

2. Run a test query in Azure Data Explorer.

```
var kustoUri = $"https://{{kustoClusterName}}.{{locationSmallCase}}.kusto.windows.net";
var kustoConnectionStringBuilder = new KustoConnectionStringBuilder(kustoUri)
{
    InitialCatalog = kustoDatabaseName,
    FederatedSecurity = true,
    ApplicationClientId = clientId,
    ApplicationKey = clientSecret,
    Authority = tenantId
};
using (var kustoClient = KustoClientFactory.CreateCslQueryProvider(kustoConnectionStringBuilder))
{
    var query = $"{{kustoTableName}} | take 10";
    using (var reader = kustoClient.ExecuteQuery(query) as DataTableReader2)
    {// Print the contents of each of the result sets.
        while (reader.Read())
        {
            Console.WriteLine($"{reader[0]}, {reader[1]}, {reader[2]}");
        }
    }
}
```

Clean up resources

To delete the resource group and clean up resources, use the following command:

```
await resourceManagementClient.ResourceGroups.DeleteAsync(resourceGroupName);
```

Next steps

- To learn about other ways to create a cluster and database, see [Create an Azure Data Explorer cluster and database](#).
- To learn more about ingestion methods, see [Azure Data Explorer data ingestion](#).
- To learn about the web application, see [Quickstart: Query data in the Azure Data Explorer web UI](#).
- [Write queries](#) with Kusto Query Language.

End-to-end blob ingestion into Azure Data Explorer through Python

11/4/2019 • 6 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and scalable data exploration service for log and telemetry data. This article gives you an end-to-end example of how to ingest data from Azure Blob storage into Azure Data Explorer.

You'll learn how to programmatically create a resource group, a storage account and container, an event hub, and an Azure Data Explorer cluster and database. You'll also learn how to programmatically configure Azure Data Explorer to ingest data from the new storage account.

Prerequisites

If you don't have an Azure subscription, create a [free Azure account](#) before you begin.

Install the Python package

To install the Python package for Azure Data Explorer (Kusto), open a command prompt that has Python in its path. Run these commands:

```
pip install azure-common
pip install azure-mgmt-resource
pip install azure-mgmt-kusto
pip install azure-mgmt-eventgrid
pip install azure-storage-blob
```

Authentication

To run the following example, you need an Azure Active Directory (Azure AD) application and service principal that can access resources. To create a free Azure AD application and add role assignment at the subscription level, see [Create an Azure AD application](#). You also need the directory (tenant) ID, application ID, and client secret.

Azure Resource Manager template

In this article, you use an Azure Resource Manager template to create a resource group, a storage account and container, an event hub, and an Azure Data Explorer cluster and database. Save the following content in a file with the name `template.json`. You'll use this file to run the code example.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "eventHubNamespaceName": {
            "type": "string",
            "metadata": {
                "description": "Specifies a the event hub Namespace name."
            }
        },
        "eventHubName": {
            "type": "string",
            "metadata": {
                "description": "Specifies a event hub name."
            }
        }
    }
}
```

```
        },
    },
    "storageAccountType": {
        "type": "string",
        "defaultValue": "Standard_LRS",
        "allowedValues": ["Standard_LRS", "Standard_GRS", "Standard_ZRS", "Premium_LRS"],
        "metadata": {
            "description": "Storage Account type"
        }
    },
    "storageAccountName": {
        "type": "string",
        "defaultValue": "[concat('storage', uniqueString(resourceGroup().id))]",
        "metadata": {
            "description": "Name of the storage account to create"
        }
    },
    "containerName": {
        "type": "string",
        "defaultValue": "[concat('storagecontainer', uniqueString(resourceGroup().id))]",
        "metadata": {
            "description": "Name of the container in storage account to create"
        }
    },
    "eventHubSku": {
        "type": "string",
        "allowedValues": ["Basic", "Standard"],
        "defaultValue": "Standard",
        "metadata": {
            "description": "Specifies the messaging tier for service Bus namespace."
        }
    },
    "kustoClusterName": {
        "type": "string",
        "defaultValue": "[concat('kusto', uniqueString(resourceGroup().id))]",
        "metadata": {
            "description": "Name of the cluster to create"
        }
    },
    "kustoDatabaseName": {
        "type": "string",
        "defaultValue": "kustodb",
        "metadata": {
            "description": "Name of the database to create"
        }
    },
    "location": {
        "type": "string",
        "defaultValue": "[resourceGroup().location]",
        "metadata": {
            "description": "Location for all resources."
        }
    },
    "variables": {},
    "resources": [
        {
            "apiVersion": "2017-04-01",
            "type": "Microsoft.EventHub/namespaces",
            "name": "[parameters('eventHubNamespaceName')]",
            "location": "[parameters('location')]",
            "sku": {
                "name": "[parameters('eventHubSku')]",
                "tier": "[parameters('eventHubSku')]",
                "capacity": 1
            },
            "properties": {
                "isAutoInflateEnabled": false,
                "maximumThroughputUnits": 0
            }
        }
    ]
}
```

```

        }
    },
    "apiVersion": "2017-04-01",
    "type": "Microsoft.EventHub/namespaces/eventhubs",
    "name": "[concat(parameters('eventHubNamespaceName'), '/', parameters('eventHubName'))]",
    "location": "[parameters('location')]",
    "dependsOn": "[ resourceId('Microsoft.EventHub/namespaces', parameters('eventHubNamespaceName')) ]",
    "properties": {
        "messageRetentionInDays": 7,
        "partitionCount": 1
    }
},
{
    "type": "Microsoft.Storage/storageAccounts",
    "name": "[parameters('storageAccountName')]",
    "location": "[parameters('location')]",
    "apiVersion": "2018-07-01",
    "sku": {
        "name": "[parameters('storageAccountType')]"
    },
    "kind": "StorageV2",
    "resources": [
        {
            "name": "[concat('default/', parameters('containerName'))]",
            "type": "blobServices/containers",
            "apiVersion": "2018-07-01",
            "dependsOn": [
                "[parameters('storageAccountName')]"
            ],
            "properties": {
                "publicAccess": "None"
            }
        }
    ],
    "properties": {}
},
{
    "name": "[parameters('kustoClusterName')]",
    "type": "Microsoft.Kusto/clusters",
    "sku": {
        "name": "Standard_D13_v2",
        "tier": "Standard",
        "capacity": 2
    },
    "apiVersion": "2019-09-07",
    "location": "[parameters('location')]",
    "tags": {
        "Created By": "GitHub quickstart template"
    }
},
{
    "name": "[concat(parameters('kustoClusterName'), '/', parameters('kustoDatabaseName'))]",
    "type": "Microsoft.Kusto/clusters/databases",
    "apiVersion": "2019-09-07",
    "location": "[parameters('location')]",
    "dependsOn": "[ resourceId('Microsoft.Kusto/clusters', parameters('kustoClusterName')) ]",
    "properties": {
        "softDeletePeriodInDays": 365,
        "hotCachePeriodInDays": 31
    }
}
]
}

```

Code example

The following code example gives you a step-by-step process that results in data ingestion into Azure Data

Explorer.

You first create a resource group. You also create Azure resources such as a storage account and container, an event hub, and an Azure Data Explorer cluster and database. You then create an Azure Event Grid subscription, along with a table and column mapping, in the Azure Data Explorer database. Finally, you create the data connection to configure Azure Data Explorer to ingest data from the new storage account.

```
from azure.common.credentials import ServicePrincipalCredentials
from azure.mgmt.resource import ResourceManagementClient
from azure.mgmt.resource.resources.models import DeploymentMode
import os.path
import json
from azure.kusto.data.request import KustoClient, KustoConnectionStringBuilder
from azure.mgmt.eventgrid import EventGridManagementClient
from azure.mgmt.kusto import KustoManagementClient
from azure.mgmt.kusto.models import EventGridDataConnection

#Directory (tenant) ID
tenant_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
#Application ID
client_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
#Client secret
client_secret = "xxxxxxxxxxxxxx"
subscription_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
location = "West Europe"
location_small_case = "westeurope"
#Path to the Azure Resource Manager template JSON from the previous section
azure_resource_template_path = "xxxxxxxx/template.json";

deployment_name = 'e2eexample'
resource_group_name = deployment_name + "resourcegroup"
event_hub_name = deployment_name + "eventhub"
event_hub_namespace_name = event_hub_name + "ns"
storage_account_name = deployment_name + "storage"
storage_container_name = deployment_name + "storagecontainer"
event_grid_subscription_name = deployment_name + "eventgrid"
kusto_cluster_name = deployment_name + "kustocluster"
kusto_database_name = deployment_name + "kustodatabase"
kusto_table_name = "Events"
kusto_column_mapping_name = "Events_CSV_Mapping"
kusto_data_connection_name = deployment_name + "kustoeventgridconnection"

credentials = ServicePrincipalCredentials(
    client_id=client_id,
    secret=client_secret,
    tenant=tenant_id
)
resource_client = ResourceManagementClient(credentials, subscription_id)

print('Step 1: Create a new resource group in your Azure subscription to manage all the resources for using Azure Data Explorer.')
resource_client.resource_groups.create_or_update(
    resource_group_name,
    {
        'location': location_small_case
    }
)

print('Step 2: Create a Blob Storage, a container in the Storage account, an Event Hub, an Azure Data Explorer cluster, and database by using an Azure Resource Manager template.')
#Read the Azure Resource Manager template
with open(azure_resource_template_path, 'r') as template_file_fd:
    template = json.load(template_file_fd)

parameters = {
    'eventHubNamespaceName': event_hub_namespace_name,
```

```

        'eventHubName': event_hub_name,
        'storageAccountName': storage_account_name,
        'containerName': storage_container_name,
        'kustoClusterName': kusto_cluster_name,
        'kustoDatabaseName': kusto_database_name
    }
parameters = {k: {'value': v} for k, v in parameters.items()}
deployment_properties = {
    'mode': DeploymentMode.incremental,
    'template': template,
    'parameters': parameters
}

#Returns an instance of LROPoller; see https://docs.microsoft.com/python/api/msrest/msrest.polling.lropoller?
view=azure-python
poller = resource_client.deployments.create_or_update(
    resource_group_name,
    deployment_name,
    deployment_properties
)
poller.wait()

print('Step 3: Create an Event Grid subscription to publish blob events created in a specific container to an
Event Hub.')
event_client = EventGridManagementClient(credentials, subscription_id)
storage_resource_id =
'/subscriptions/{}/resourceGroups/{}/providers/Microsoft.Storage/storageAccounts{}'.format(subscription_id,
resource_group_name, storage_account_name)
event_hub_resource_id =
'/subscriptions/{}/resourceGroups/{}/providers/Microsoft.EventHub/namespaces/{}/eventhubs{}'.format(subscription_id,
resource_group_name, event_hub_namespace_name, event_hub_name)
event_client.event_subscriptions.create_or_update(storage_resource_id, event_grid_subscription_name, {
    'destination': {
        'endpointType': 'EventHub',
        'properties': {
            'resourceId': event_hub_resource_id
        }
    },
    "filter": {
        "subjectBeginsWith": "/blobServices/default/containers{}".format(storage_container_name),
        "includedEventTypes": ["Microsoft.Storage.BlobCreated"],
        "advancedFilters": []
    }
})
}

print('Step 4: Create a table (with three columns: EventTime, EventId, and EventSummary) and column mapping in
your Azure Data Explorer database.')
kusto_uri = "https://{}.{}.kusto.windows.net".format(kusto_cluster_name, location_small_case)
database_name = kusto_database_name
kusto_connection_string_builder =
KustoConnectionStringBuilder.with_aad_application_key_authentication(connection_string=kusto_uri,
aad_app_id=client_id, app_key=client_secret, authority_id=tenant_id)
kusto_client = KustoClient(kusto_connection_string_builder)
create_table_command = ".create table " + kusto_table_name + " (EventTime: datetime, EventId: int,
EventSummary: string)"
kusto_client.execute_mgmt(database_name, create_table_command)

create_column_mapping_command = ".create table " + kusto_table_name + " ingestion csv mapping '" +
kusto_column_mapping_name \
        + "''' '[{"Name":"EventTime","datatype":"datetime","Ordinal":0},
{"Name":"EventId","datatype":"int","Ordinal":1},{"Name":"EventSummary","datatype":"string","Ordinal":2}]''''"
kusto_client.execute_mgmt(database_name, create_column_mapping_command)

print('Step 5: Add an Event Grid data connection. Azure Data Explorer will automatically ingest the data when
new blobs are created.')
kusto_management_client = KustoManagementClient(credentials, subscription_id)
data_connections = kusto_management_client.data_connections

```

```
#Returns an instance of LROPoller; see https://docs.microsoft.com/python/api/msrest/msrest.polling.lropoller?view=azure-python
poller = data_connections.create_or_update(resource_group_name=resource_group_name,
cluster_name=kusto_cluster_name, database_name=kusto_database_name,
data_connection_name=kusto_data_connection_name,

parameters=EventGridDataConnection(storage_account_resource_id=storage_resource_id,
event_hub_resource_id=event_hub_resource_id, consumer_group="$Default", location=location,
table_name=kusto_table_name, mapping_rule_name=kusto_column_mapping_name, data_format="csv"))
poller.wait()
```

SETTING	FIELD DESCRIPTION
tenant_id	Your tenant ID. It's also known as a directory ID.
subscription_id	The subscription ID that you use for resource creation.
client_id	The client ID of the application that can access resources in your tenant.
client_secret	The client secret of the application that can access resources in your tenant.

Test the code example

1. Upload a file into the storage account.

```
account_key = "xxxxxxxxxxxxxx"
block_blob_service = BlockBlobService(account_name=storage_account_name, account_key=account_key)
blob_name = "test.csv"
blob_content = """2007-01-01 00:00:00.000000,2592,Several trees down
2007-01-01 00:00:00.000000,4171,Winter Storm"""
block_blob_service.create_blob_from_text(container_name=storage_container_name, blob_name=blob_name,
text=blob_content)
```

SETTING	FIELD DESCRIPTION
account_key	The access key of the programmatically created storage account.

2. Run a test query in Azure Data Explorer.

```
kusto_uri = "https://{}.{}.kusto.windows.net".format(kusto_cluster_name, location_small_case)
kusto_connection_string_builder =
KustoConnectionStringBuilder.with_aad_application_key_authentication(connection_string=kusto_uri,
aad_app_id=client_id, app_key=client_secret, authority_id=tenant_id)
kusto_client = KustoClient(kusto_connection_string_builder)
query = "{} | take 10".format(kusto_table_name)
response = kusto_client.execute_query(kusto_database_name, query)
print(response.primary_results[0].rows_count)
```

Clean up resources

To delete the resource group and clean up resources, use the following command:

```
#Returns an instance of LROPoller; see https://docs.microsoft.com/python/api/msrest/msrest.polling.lropoller?  
view=azure-python  
poller = resource_client.resource_groups.delete(resource_group_name=resource_group_name)  
poller.wait()
```

Next steps

- To learn about other ways to create a cluster and database, see [Create an Azure Data Explorer cluster and database](#).
- To learn more about ingestion methods, see [Azure Data Explorer data ingestion](#).
- To learn about the web application, see [Quickstart: Query data in the Azure Data Explorer web UI](#).
- [Write queries with Kusto Query Language](#).

Tutorial: Ingest and query monitoring data in Azure Data Explorer

12/10/2019 • 13 minutes to read • [Edit Online](#)

This tutorial will teach you how to ingest data from diagnostic and activity logs to an Azure Data Explorer cluster without writing code. With this simple ingestion method, you can quickly begin querying Azure Data Explorer for data analysis.

In this tutorial, you'll learn how to:

- Create tables and ingestion mapping in an Azure Data Explorer database.
- Format the ingested data by using an update policy.
- Create an [event hub](#) and connect it to Azure Data Explorer.
- Stream data to an event hub from Azure Monitor [diagnostic metrics and logs](#) and [activity logs](#).
- Query the ingested data by using Azure Data Explorer.

NOTE

Create all resources in the same Azure location or region.

Prerequisites

- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.
- [An Azure Data Explorer cluster and database](#). In this tutorial, the database name is *TestDatabase*.

Azure Monitor data provider: diagnostic metrics and logs and activity logs

View and understand the data provided by the Azure Monitor diagnostic metrics and logs and activity logs below. You'll create an ingestion pipeline based on these data schemas. Note that each event in a log has an array of records. This array of records will be split later in the tutorial.

Examples of diagnostic metrics and logs and activity logs

Azure diagnostic metrics and logs and activity logs are emitted by an Azure service and provide data about the operation of that service.

- [Diagnostic metrics](#)
- [Diagnostic logs](#)
- [Activity logs](#)

Example

Diagnostic metrics are aggregated with a time grain of 1 minute. Following is an example of an Azure Data Explorer metric-event schema on query duration:

```
{
  "records": [
    {
      "count": 14,
      "total": 0,
      "minimum": 0,
      "maximum": 0,
      "average": 0,
      "resourceId": "/SUBSCRIPTIONS/<subscriptionID>/RESOURCEGROUPS/<resource-group>/PROVIDERS/MICROSOFT.KUSTO/CLUSTERS/<cluster-name>",
      "time": "2018-12-20T17:00:00.000000Z",
      "metricName": "QueryDuration",
      "timeGrain": "PT1M"
    },
    {
      "count": 12,
      "total": 0,
      "minimum": 0,
      "maximum": 0,
      "average": 0,
      "resourceId": "/SUBSCRIPTIONS/<subscriptionID>/RESOURCEGROUPS/<resource-group>/PROVIDERS/MICROSOFT.KUSTO/CLUSTERS/<cluster-name>",
      "time": "2018-12-21T17:00:00.000000Z",
      "metricName": "QueryDuration",
      "timeGrain": "PT1M"
    }
  ]
}
```

Set up an ingestion pipeline in Azure Data Explorer

Setting up an Azure Data Explorer pipeline involves several steps, such as [table creation and data ingestion](#). You can also manipulate, map, and update the data.

Connect to the Azure Data Explorer Web UI

In your Azure Data Explorer *TestDatabase* database, select **Query** to open the Azure Data Explorer Web UI.

Resource group (change)	Retention period (in days)
<your resource group>	3650 days

Location	Cache period (in days)
West US	31 days

Subscription (change)	
<your subscription>	

Subscription ID	
<your subscription ID>	

Create the target tables

The structure of the Azure Monitor logs isn't tabular. You'll manipulate the data and expand each event to one or more records. The raw data will be ingested to an intermediate table named *ActivityLogsRawRecords* for activity logs and *DiagnosticRawRecords* for diagnostic metrics and logs. At that time, the data will be manipulated and expanded. Using an update policy, the expanded data will then be ingested into the *ActivityLogs* table for activity logs, *DiagnosticMetrics* for diagnostic metrics and *DiagnosticLogs* for diagnostic logs. This means that you'll need to create two separate tables for ingesting activity logs and three separate tables for ingesting diagnostic metrics and logs.

Use the Azure Data Explorer Web UI to create the target tables in the Azure Data Explorer database.

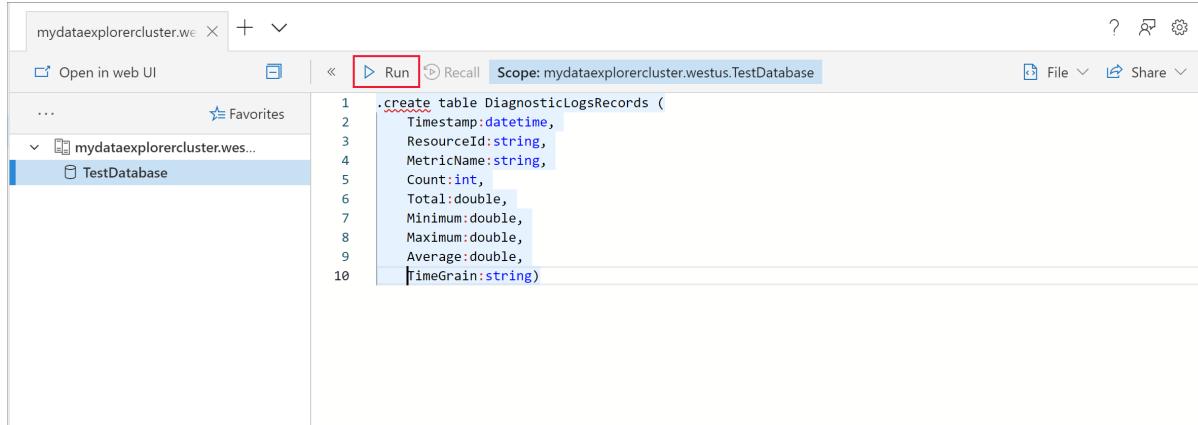
- Diagnostic metrics
- Diagnostic logs
- Activity logs

Create tables for the diagnostic metrics

1. In the *TestDatabase* database, create a table named *DiagnosticMetrics* to store the diagnostic metrics records. Use the following `.create table` control command:

```
.create table DiagnosticMetrics (Timestamp:datetime, ResourceId:string, MetricName:string, Count:int, Total:double, Minimum:double, Maximum:double, Average:double, TimeGrain:string)
```

2. Select **Run** to create the table.



3. Create the intermediate data table named *DiagnosticRawRecords* in the *TestDatabase* database for data manipulation using the following query. Select **Run** to create the table.

```
.create table DiagnosticRawRecords (Records:dynamic)
```

4. Set zero **retention policy** for the intermediate table:

```
.alter-merge table DiagnosticRawRecords policy retention softdelete = 0d
```

Create table mappings

Because the data format is `json`, data mapping is required. The `json` mapping maps each json path to a table column name.

- Diagnostic metrics / Diagnostic logs
- Activity logs

Map diagnostic metrics and logs to the table

To map the diagnostic metric and log data to the table, use the following query:

```
.create table DiagnosticRawRecords ingestion json mapping 'DiagnosticRawRecordsMapping'
'[{"column":"Records","path":("$.records")}]'
```

Create the update policy for metric and log data

- Diagnostic metrics
- Diagnostic logs
- Activity logs

Create data update policy for diagnostics metrics

1. Create a [function](#) that expands the collection of diagnostic metric records so that each value in the collection receives a separate row. Use the `mv-expand` operator:

```
.create function DiagnosticMetricsExpand() {
    DiagnosticRawRecords
    | mv-expand events = Records
    | where isnitempty(events.metricName)
    | project
        Timestamp = todatetime(events['time']),
        ResourceId = tostring(events.resourceId),
        MetricName = tostring(events.metricName),
        Count = toint(events['count']),
        Total = todouble(events.total),
        Minimum = todouble(events.minimum),
        Maximum = todouble(events.maximum),
        Average = todouble(events.average),
        TimeGrain = tostring(events.timeGrain)
}
```

2. Add the [update policy](#) to the target table. This policy will automatically run the query on any newly ingested data in the *DiagnosticRawRecords* intermediate data table and ingest its results into the *DiagnosticMetrics* table:

```
.alter table DiagnosticMetrics policy update @'[{"Source": "DiagnosticRawRecords", "Query": "DiagnosticMetricsExpand()", "IsEnabled": "True"}]'
```

Create an Azure Event Hubs namespace

Azure diagnostic settings enable exporting metrics and logs to a storage account or to an event hub. In this tutorial, we'll route the metrics and logs via an event hub. You'll create an Event Hubs namespace and an event hub for the diagnostic metrics and logs in the following steps. Azure Monitor will create the event hub *insights-operational-logs* for the activity logs.

1. Create an event hub by using an Azure Resource Manager template in the Azure portal. To follow the rest of the steps in this article, right-click the **Deploy to Azure** button, and then select **Open in new window**. The **Deploy to Azure** button takes you to the Azure portal.



2. Create an Event Hubs namespace and an event hub for the diagnostic logs.

The screenshot shows the 'Create an EventHubs namespace, Event Hub, & consumer group' wizard in the Azure portal. The 'SETTINGS' section is expanded, showing the following configuration:

- Namespace Name:** AzureMonitoringData
- Eventhub Sku:** Standard
- Sku Capacity:** 1
- Event Hub Name:** DiagnosticLogsData
- Consumer Group Name:** adxpipeline
- Location:** [resourceGroup()].location

The 'Purchase' button at the bottom is highlighted with a red box.

- Fill out the form with the following information. For any settings not listed in the following table, use the default values.

SETTING	SUGGESTED VALUE	DESCRIPTION
Subscription	<i>Your subscription</i>	Select the Azure subscription that you want to use for your event hub.
Resource group	<i>test-resource-group</i>	Create a new resource group.
Location	Select the region that best meets your needs.	Create the Event Hubs namespace in the same location as other resources.
Namespace name	<i>AzureMonitoringData</i>	Choose a unique name that identifies your namespace.
Event hub name	<i>DiagnosticData</i>	The event hub sits under the namespace, which provides a unique scoping container.
Consumer group name	<i>adxpipeline</i>	Create a consumer group name. Consumer groups enable multiple consuming applications to each have a separate view of the event stream.

Connect Azure Monitor metrics and logs to your event hub

Now you need to connect your diagnostic metrics and logs and your activity logs to the event hub.

- Diagnostic metrics / Diagnostic logs
- Activity logs

Connect diagnostic metrics and logs to your event hub

Select a resource from which to export metrics. Several resource types support exporting diagnostic data, including Event Hubs namespace, Azure Key Vault, Azure IoT Hub, and Azure Data Explorer clusters. In this tutorial, we'll use an Azure Data Explorer cluster as our resource, we'll review query performance metrics and ingestion results logs.

1. Select your Kusto cluster in the Azure portal.
2. Select **Diagnostic settings**, and then select the **Turn on diagnostics** link.

The screenshot shows the Azure portal interface for managing an Azure Data Explorer cluster named 'mydataexplorercluster'. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Query, Settings (Scale up, Scale out, Properties, Locks, Automation script), Data (Databases), Monitoring (Alerts, Metrics), and Diagnostic settings. The 'Diagnostic settings' link is highlighted with a red box. The main content area is titled 'mydataexplorercluster - Diagnostic settings' and includes a 'Turn on diagnostics' section with a red box around it. The section describes collecting data for 'AllMetrics'.

3. The **Diagnostics settings** pane opens. Take the following steps:

- a. Give your diagnostics log data the name *ADXExportedData*.
- b. Under **LOG**, select both **SucceededIngestion** and **FailedIngestion** check boxes.
- c. Under **METRIC**, select the **Query performance** check box.
- d. Select the **Stream to an event hub** check box.
- e. Select **Configure**.

Diagnostics settings

- Name: ADXExportedData
- Stream to an event hub: checked
- Event hub: Configure
- Metric: AllMetrics

Select event hub

- Subscription: <your subscription>
- Select event hub namespace: AzureMonitoringData
- Select event hub name (optional): diagnosticlogdata
- Select event hub policy name: RootManageSharedAccessKey

- In the **Select event hub** pane, configure how to export data from diagnostic logs to the event hub you created:
 - In the **Select event hub namespace** list, select **AzureMonitoringData**.
 - In the **Select event hub name** list, select **DiagnosticData**.
 - In the **Select event hub policy name** list, select **RootManagerSharedAccessKey**.
 - Select **OK**.
- Select **Save**.

See data flowing to your event hubs

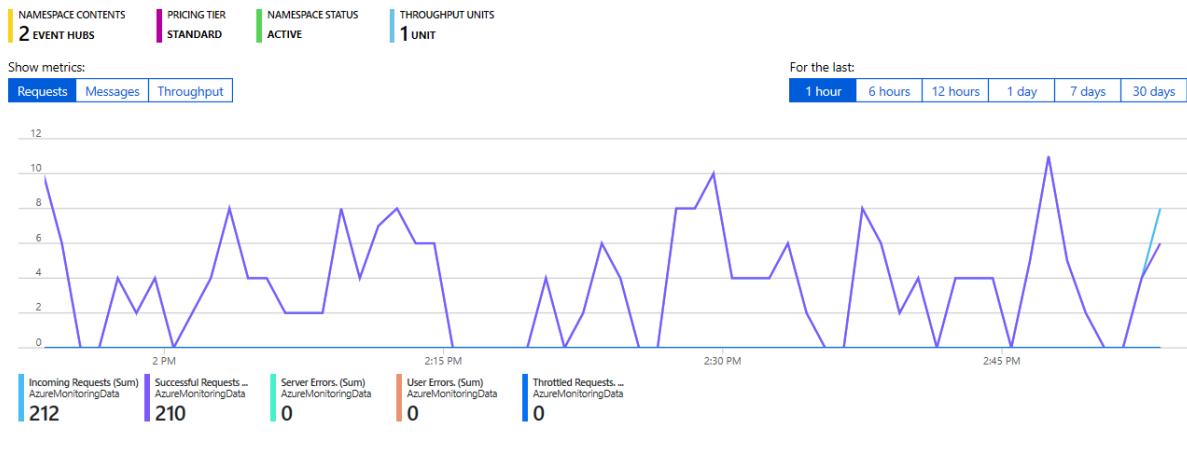
- Wait a few minutes until the connection is defined, and the activity-log export to the event hub is finished. Go to your Event Hubs namespace to see the event hubs you created.

AzureMonitoringData - Event Hubs

Event Hubs

NAME	STATUS	MESSAGE RETENTION	PARTITION COUNT
diagnosticlogdata	Active	7	4
insights-operational-logs	Active	7	4

- See data flowing to your event hub:



Connect an event hub to Azure Data Explorer

Now you need to create the data connections for your diagnostic metrics and logs and activity logs.

Create the data connection for diagnostic metrics and logs and activity logs

1. In your Azure Data Explorer cluster named *kustodocs*, select **Databases** in the left menu.
2. In the **Databases** window, select your *TestDatabase* database.
3. In the left menu, select **Data ingestion**.
4. In the **Data ingestion** window, click **+ Add Data Connection**.
5. In the **Data connection** window, enter the following information:

Data connection

Create data connection

Select connection type
Event Hub

Data source

* Data connection name ⓘ
DiagnosticsLogsConnection ✓

* Subscription
<your subscription> ✓

* Event Hub namespace
AzureMonitoringData ✓

* Event Hub
diagnosticlogdata ✓

* Consumer group ⓘ
adxpipeline ✓

Target table

My data includes routing info ⓘ
 No Yes

Table ⓘ
DiagnosticLogsRawRecords

Data format
JSON ✓

Column mapping ⓘ
DiagnosticLogsRawRecordsMapping

- Diagnostic metrics / Diagnostic logs

- Activity logs

1. Use the following settings in the **Data Connection** window:

Data source:

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
Data connection name	DiagnosticsLogsConnection	The name of the connection you want to create in Azure Data Explorer.
Event hub namespace	AzureMonitoringData	The name you chose earlier that identifies your namespace.
Event hub	DiagnosticData	The event hub you created.
Consumer group	adxpipeline	The consumer group defined in the event hub you created.

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION

Target table:

There are two options for routing: *static* and *dynamic*. For this tutorial, you'll use static routing (the default), where you specify the table name, the data format, and the mapping. Leave **My data includes routing info** unselected.

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
Table	<i>DiagnosticRawRecords</i>	The table you created in the <i>TestDatabase</i> database.
Data format	JSON	The format used in the table.
Column mapping	<i>DiagnosticRawRecordsMapping</i>	The mapping you created in the <i>TestDatabase</i> database, which maps incoming JSON data to the column names and data types of the <i>DiagnosticRawRecords</i> table.

2. Select **Create**.

Query the new tables

You now have a pipeline with data flowing. Ingestion via the cluster takes 5 minutes by default, so allow the data to flow for a few minutes before beginning to query.

- [Diagnostic metrics](#)
- [Diagnostic logs](#)
- [Activity logs](#)

Query the diagnostic metrics table

The following query analyzes query duration data from diagnostic metric records in Azure Data Explorer:

```
DiagnosticMetrics
| where Timestamp > ago(15m) and MetricName == 'QueryDuration'
| summarize avg(Average)
```

Query results:

	avg_Average
	00:06.156

Next steps

- Learn to write many more queries on the data you extracted from Azure Data Explorer by using [Write queries for Azure Data Explorer](#).

- Monitor Azure Data Explorer ingestion operations using diagnostic logs
- Use metrics to monitor cluster health

Tutorial: Visualize data from Azure Data Explorer in Power BI

11/13/2019 • 5 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Power BI is a business analytics solution that lets you visualize your data and share the results across your organization. In this tutorial, you first learn how to render visuals in Azure Data Explorer. You then connect to Azure Data Explorer with Power BI, build a report based on sample data, and publish the report to the Power BI service.

If you don't have an Azure subscription, create a [free Azure account](#) before you begin. If you're not signed up for Power BI Pro, [sign up for a free trial](#) before you begin.

In this tutorial, you learn how to:

- Render visuals in Azure Data Explorer
- Connect to Azure Data Explorer in Power BI Desktop
- Work with data in Power BI Desktop
- Create a report with visuals
- Publish and share the report

Prerequisites

In addition to Azure and Power BI subscriptions, you need the following to complete this tutorial:

- [A test cluster and database](#)
- [The StormEvents sample data](#). The StormEvents sample data set contains weather-related data from the [National Centers for Environmental Information](#).
- [Power BI Desktop](#) (select **DOWNLOAD FREE**)

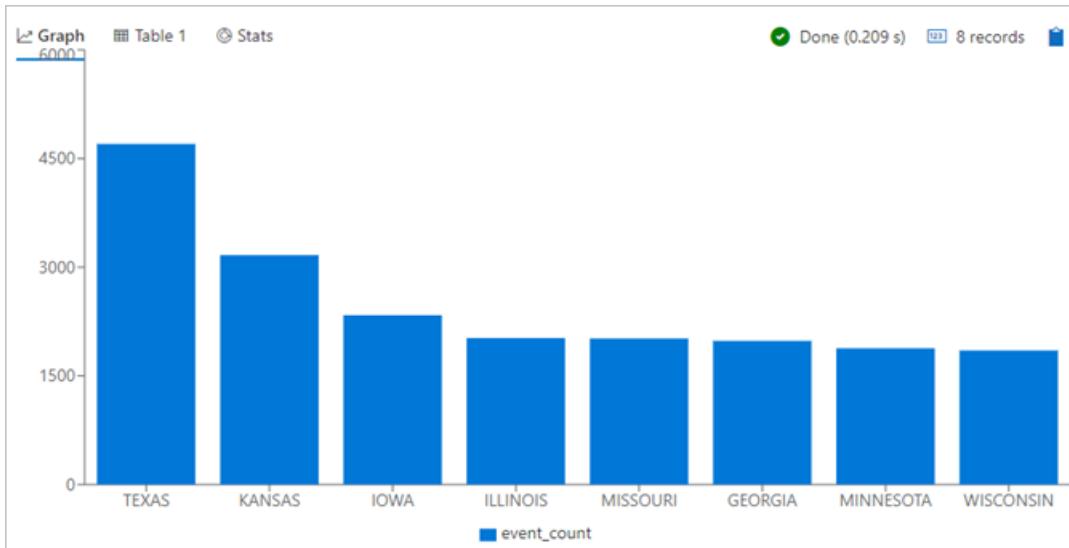
Render visuals in Azure Data Explorer

Before jumping into Power BI, let's look at how to render visuals in Azure Data Explorer. This is great for some quick analysis.

1. Sign in to <https://dataexplorer.azure.com>.
2. In the left pane, select the test database that contains the StormEvents sample data.
3. Paste the following query into the right window, and select **Run**.

```
StormEvents
| summarize event_count=count() by State
| where event_count > 1800
| project State, event_count
| sort by event_count
| render columnchart
```

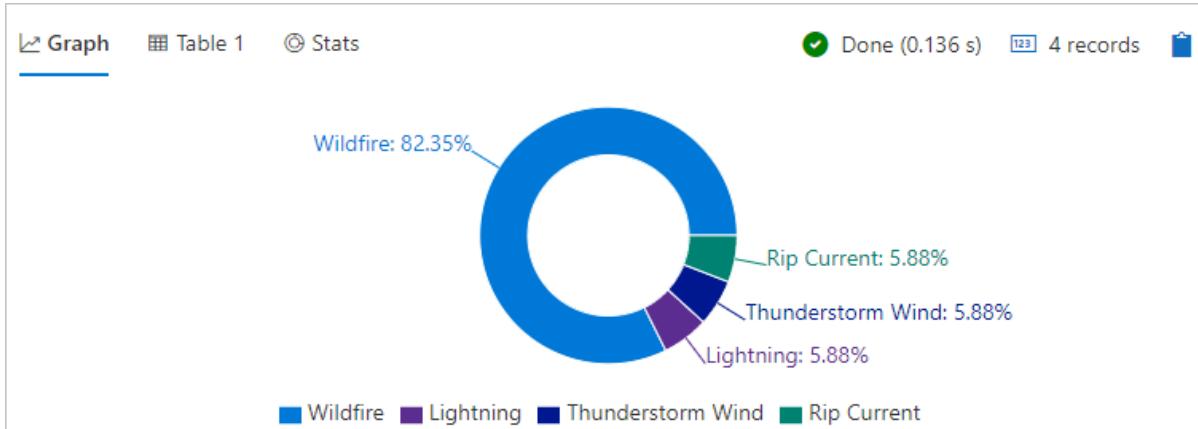
This query counts weather events by state. It then renders a column chart for all states that have more than 1800 weather events.



4. Paste the following query into the right window, and select **Run**.

```
StormEvents
| where State == "WASHINGTON" and StartTime >= datetime(2007-07-01) and StartTime <= datetime(2007-07-31)
| summarize StormCount = count() by EventType
| render piechart
```

This query counts weather events by type for the month of July in the state of Washington. It then renders a pie chart showing the percentage of each event type.

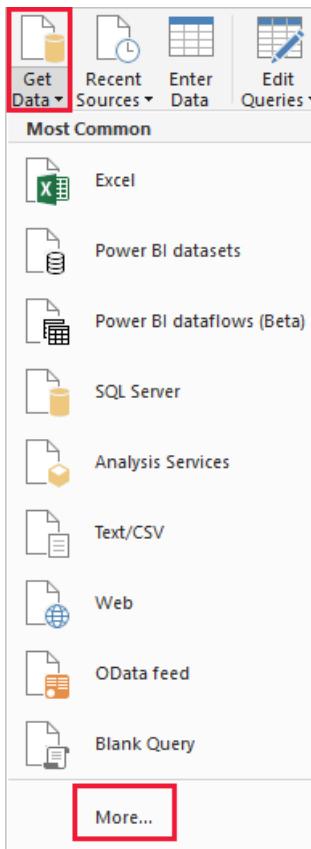


It's now time to look at Power BI, but there's a lot more you can do with visuals in Azure Data Explorer.

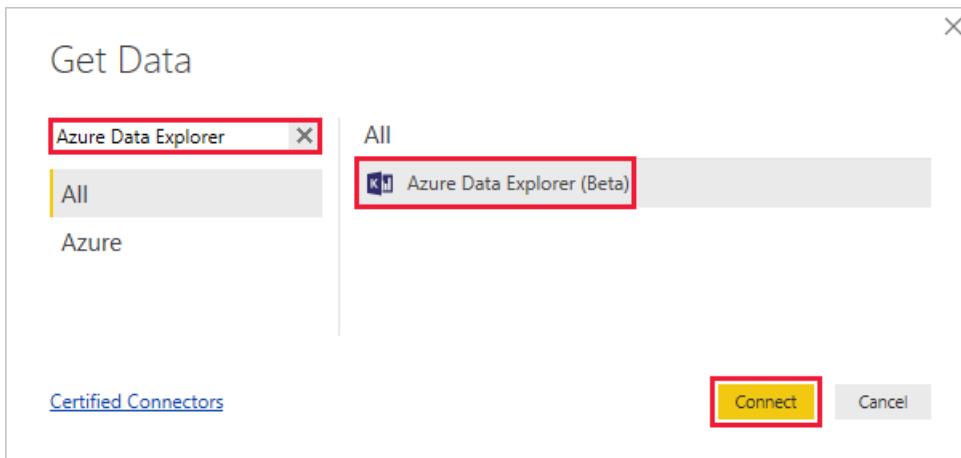
Connect to Azure Data Explorer

Now you connect to Azure Data Explorer in Power BI Desktop.

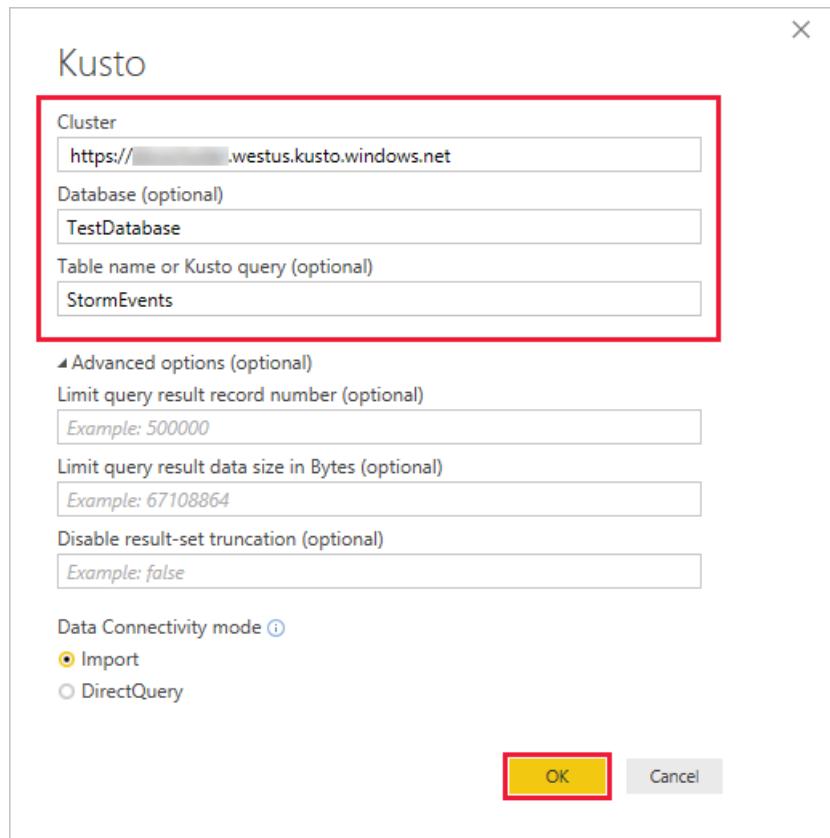
1. In Power BI Desktop on the **Home** tab, select **Get Data** then **More**.



2. Search for *Azure Data Explorer*, select **Azure Data Explorer (Beta)**, then **Connect**.



3. On the **Preview connector** screen, select **Continue**.
4. On the next screen, enter the name of your test cluster and database. Cluster should be in the form `https://<ClusterName>.<Region>.kusto.windows.net`. Enter *StormEvents* for the name of the table. Leave all other options with default values, and select **OK**.



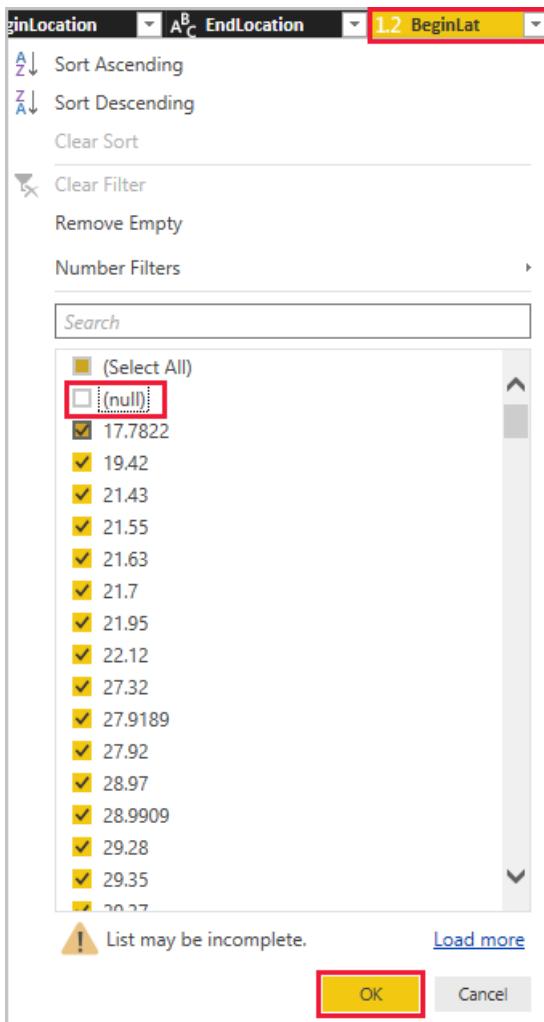
5. On the data preview screen, select **Edit**.

The table opens in Power Query Editor, where you can edit rows and columns before importing the data.

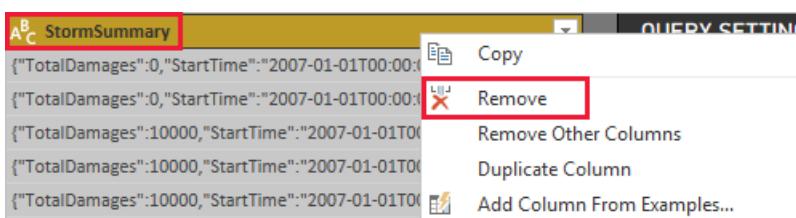
Work with data in Power BI Desktop

Now that you have a connection to Azure Data Explorer, you edit the data in Power Query Editor. You drop rows with null values in the **BeginLat** column and drop the **StormSummary** JSON column entirely. These are simple operations, but you can also perform complex transformations when importing data.

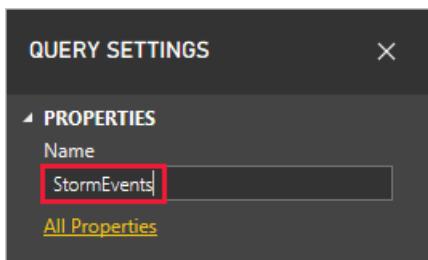
1. Select the arrow for the **BeginLat** column, clear the **null** check box, then select **OK**.



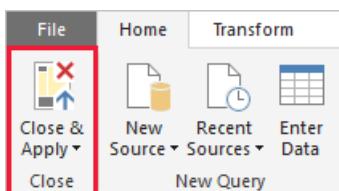
- Right-click the **StormSummary** column header, then select **Remove**.



- In the **QUERY SETTINGS** pane, change the name from *Query1* to *StormEvents*.



- On the **Home** tab of the ribbon, select **Close and apply**.

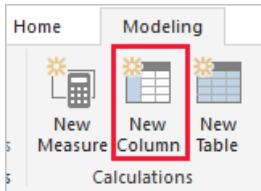


Power Query applies your changes, then imports the sample data into a *data model*. The next few steps show you how to enrich that model. Again, this is just a simple example to give an idea of what's possible.

5. On the left side of the main window, select the data view.



6. On the **Modeling** tab of the ribbon, select **New column**.



7. Enter the following Data Analysis Expressions (DAX) formula into the formula bar, then press Enter.

```
DurationHours = DATEDIFF(StormEvents[StartTime], StormEvents[EndTime], hour)
```



This formula creates the column *DurationHours* that calculates how many hours each weather event lasted. You use this column in a visual in the next section.

8. Scroll to the right side of the table to see the column.

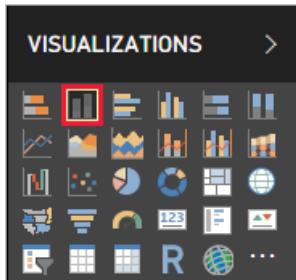
Create a report with visuals

Now that the data is imported and you've improved the data model, it's time to build a report with visuals. You add a column chart based on event duration and a map that shows crop damage.

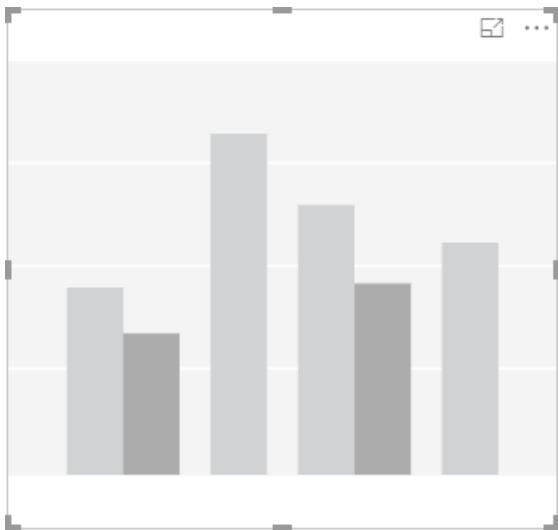
1. On the left side of the window, select the report view.



2. In the **VISUALIZATIONS** pane, select the clustered column chart.



A blank chart is added to the canvas.

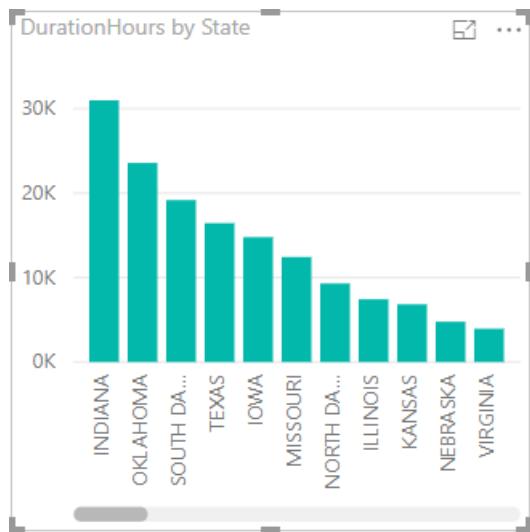


3. In the **FIELDS** list, select **DurationHours** and **State**.

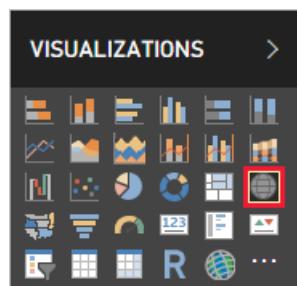
The screenshot shows the 'Fields' list for a dataset named 'StormEvents'. The list contains numerous fields, each preceded by a checkbox and a summation symbol (Σ). Two specific fields are highlighted with red boxes: 'DurationHours' and 'State'. Both of these fields have a checked checkbox and a small yellow square icon to their left, indicating they are selected for the analysis.

- ☐ Σ BeginLat
- ☐ BeginLocation
- ☐ Σ BeginLon
- ☐ Σ DamageCrops
- ☐ Σ DamagePropre...
- ☐ Σ DeathsDirect
- ☐ Σ DeathsIndirect
- Σ DurationHours
- ☐ Σ EndLat
- ☐ EndLocation
- ☐ Σ EndLon
- ☐ EndTime
- ☐ Σ EpisodeId
- ☐ EpisodeNarrat...
- ☐ Σ EventId
- ☐ EventNarrative
- ☐ EventType
- ☐ Σ InjuriesDirect
- ☐ Σ InjuriesIndirect
- ☐ Source
- ☐ StartTime
- State

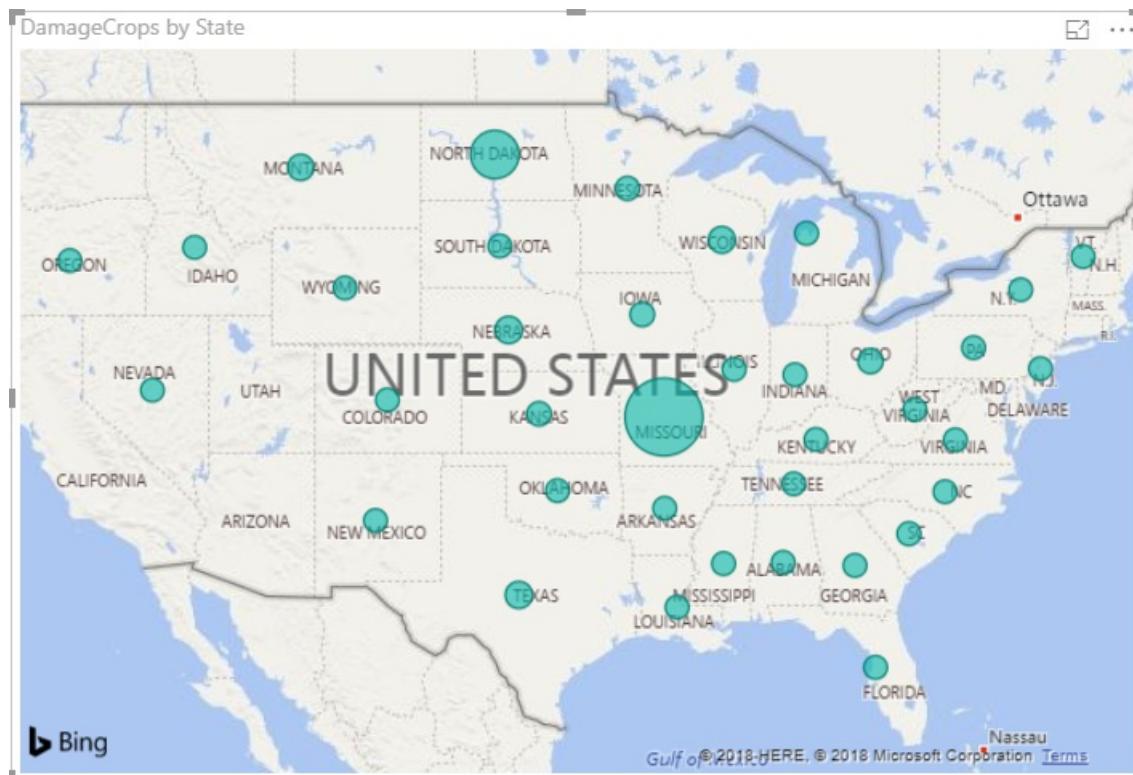
You now have a chart that shows the total hours of weather events by state over the course of a year.



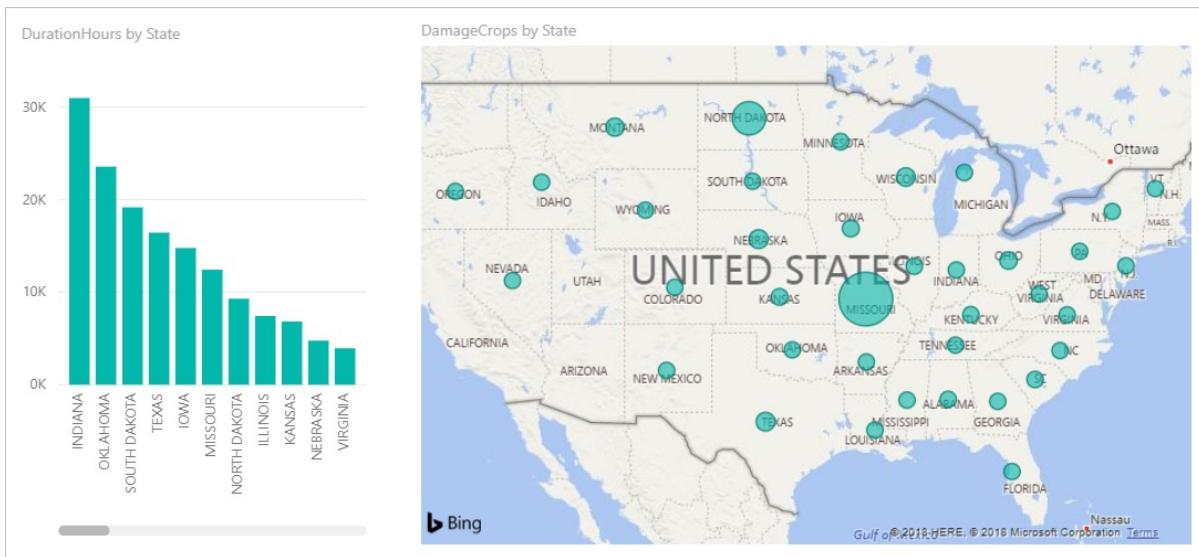
4. Click anywhere on the canvas outside the column chart.
5. In the **VISUALIZATIONS** pane, select the map.



6. In the **FIELDS** list, select **CropDamage** and **State**. Resize the map so you can see the US states clearly.



- The size of the bubbles represents the dollar value of crop damage. Mouse over the bubbles to see details.
7. Move and resize the visuals so you have a report that looks like the following image.

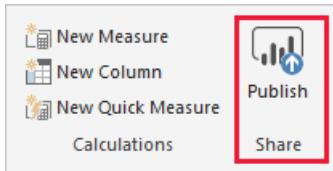


- Save the report with the name *storm-events.pbix*.

Publish and share the report

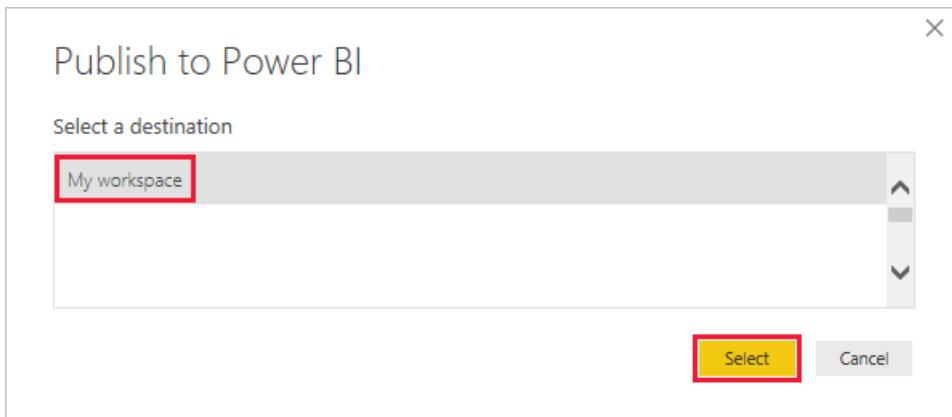
Until this point, the work you've done in Power BI has all been local, using Power BI Desktop. Now you publish the report to the Power BI service where you can share it with others.

- In Power BI Desktop, on the **Home** tab of the ribbon, select **Publish**.

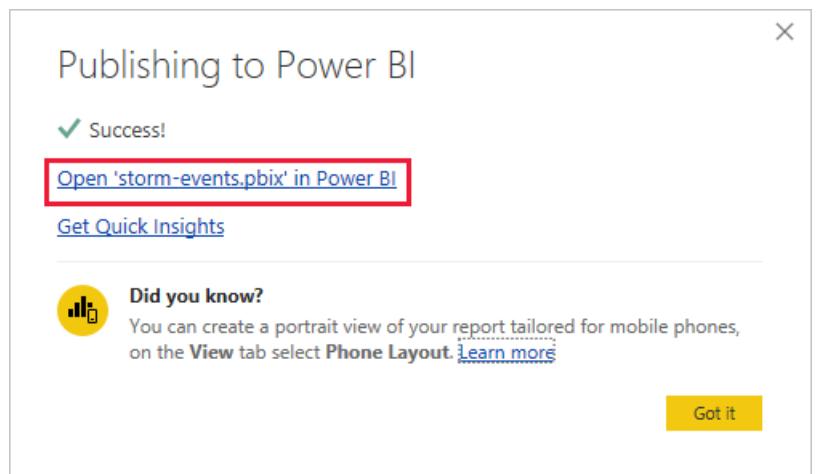


- If you're not already signed in to Power BI, go through the sign-in process.

- Select **My workspace**, then **Select**.



- When publishing is finished, select **Open storm-events.pbix in Power BI**.

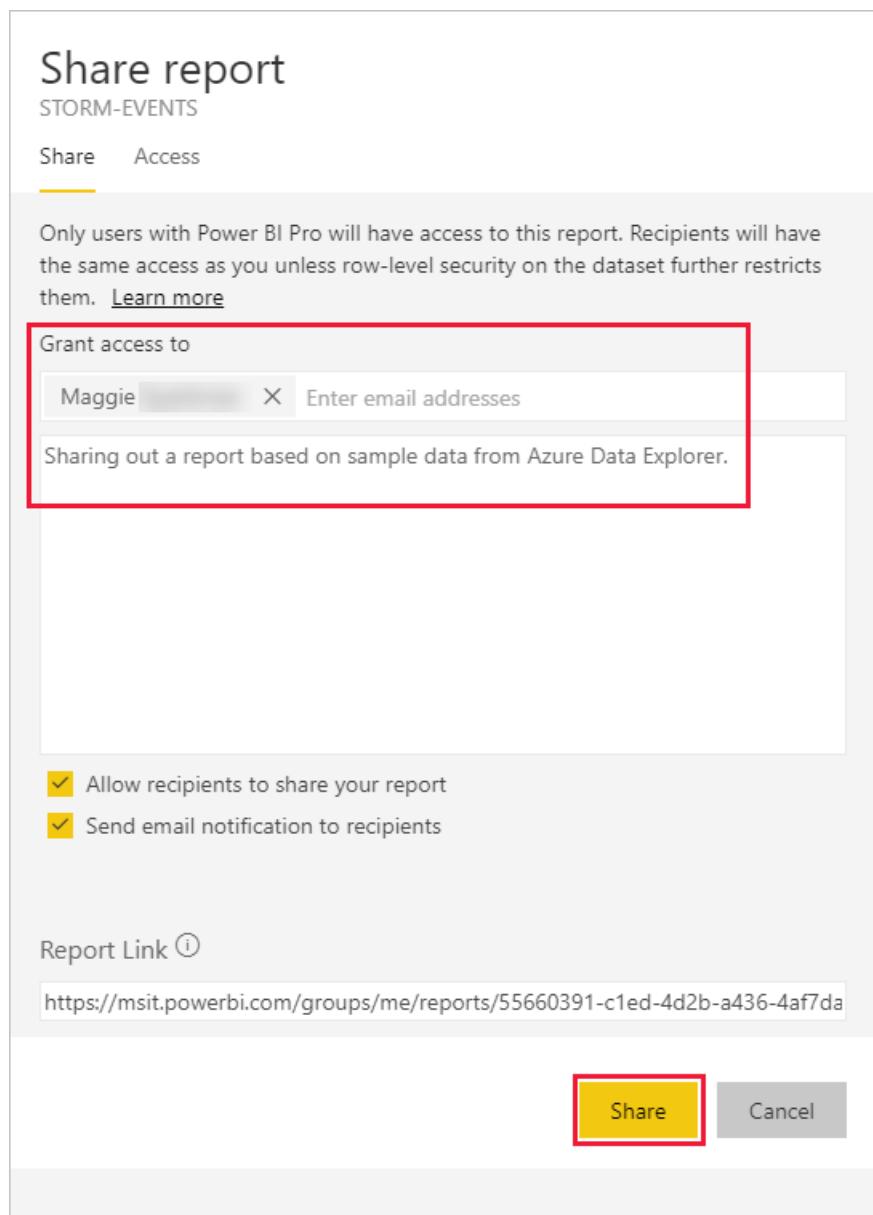


The report opens in the service, with the same visuals and layout you defined in Power BI Desktop.

5. In the upper-right corner of the report, select **Share**.



6. In the **Share report** screen, add a colleague in your organization, add a note, then select **Share**.

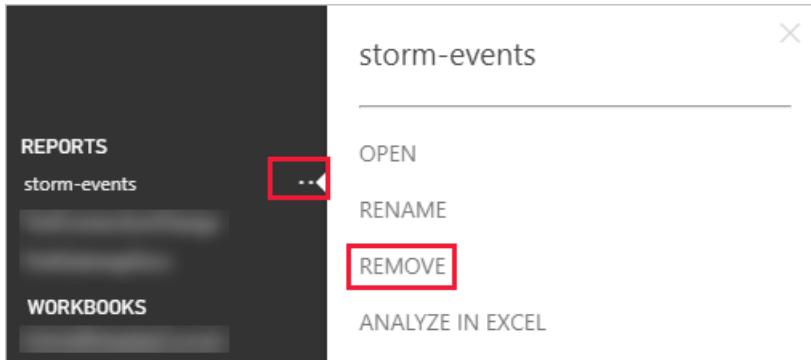


If your colleague has appropriate permissions, they can access the report that you shared.

Clean up resources

If you don't want to keep the report you created, simply delete the *storm-events.pbix* file. If you want to remove the report you published, follow these steps.

1. Under **My workspace**, scroll down to **REPORTS** and find **storm-events**.
2. Select the ellipsis (...) next to **storm-events**, then select **REMOVE**.



3. Confirm the removal.

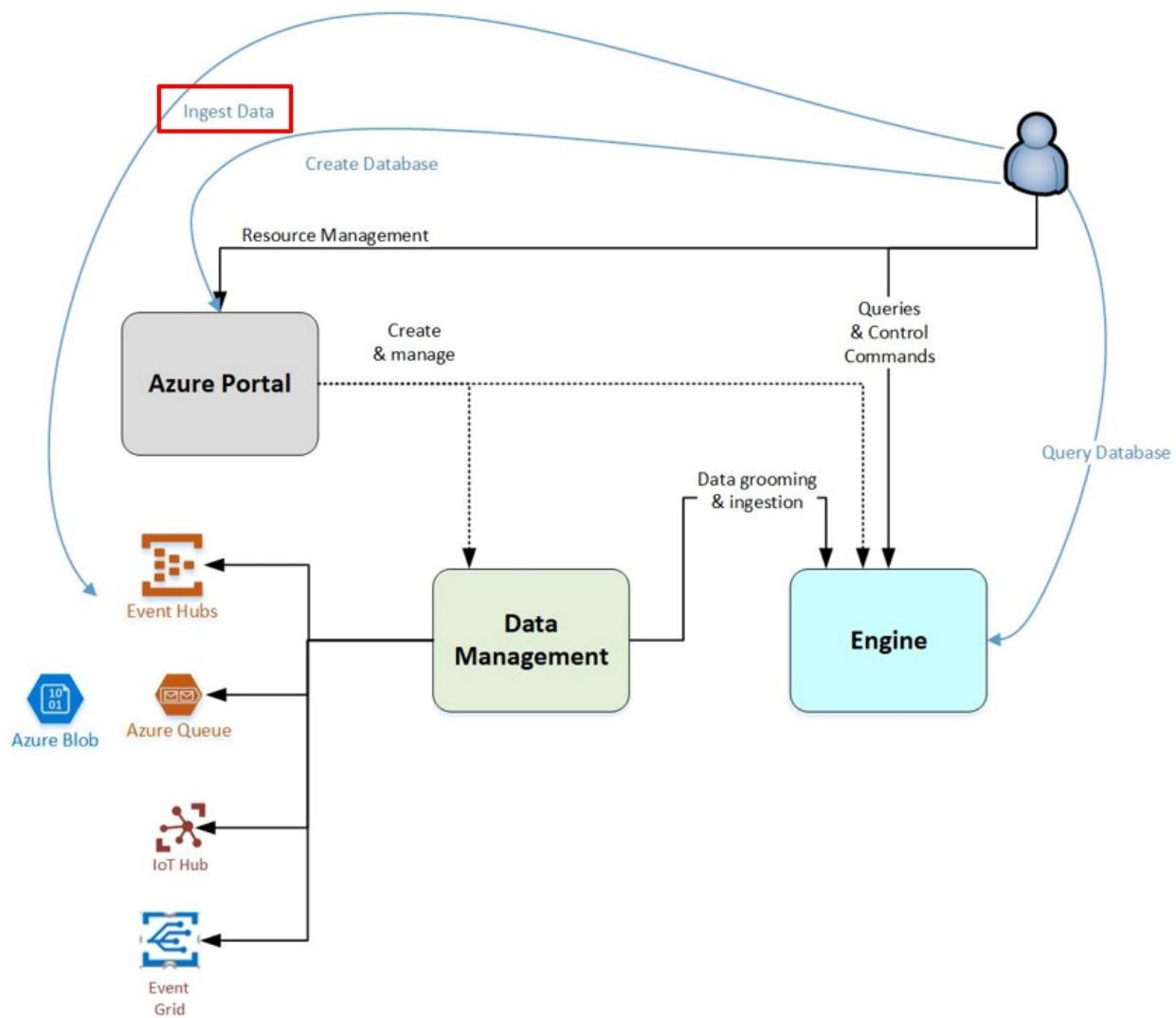
Next steps

[Write queries](#)

Azure Data Explorer data ingestion

11/14/2019 • 4 minutes to read • [Edit Online](#)

Data ingestion is the process used to load data records from one or more sources to create or update a table in Azure Data Explorer. Once ingested, the data becomes available for query. The diagram below shows the end-to-end flow for working in Azure Data Explorer, including data ingestion.



The Azure Data Explorer data management service, which is responsible for data ingestion, provides the following functionality:

1. **Data pull:** Pull data from external sources (Event Hubs) or read ingestion requests from an Azure Queue.
2. **Batching:** Batch data flowing to the same database and table to optimize ingestion throughput.
3. **Validation:** Preliminary validation and format conversion if necessary.
4. **Data manipulation:** Matching schema, organizing, indexing, encoding and compressing the data.
5. **Persistence point in the ingestion flow:** Manage ingestion load on the engine and handle retries upon transient failures.
6. **Commit the data ingest:** Makes the data available for query.

Ingestion methods

Azure Data Explorer supports several ingestion methods, each with its own target scenarios, advantages, and disadvantages. Azure Data Explorer offers pipelines and connectors to common services, programmatic ingestion using SDKs, and direct access to the engine for exploration purposes.

Ingestion using pipelines, connectors, and plugins

Azure Data Explorer currently supports:

- Event Grid pipeline, which can be managed using the management wizard in the Azure portal. For more information, see [Ingest Azure Blobs into Azure Data Explorer](#).
- Event Hub pipeline, which can be managed using the management wizard in the Azure portal. For more information, see [Ingest data from Event Hub into Azure Data Explorer](#).
- Logstash plugin, see [Ingest data from Logstash to Azure Data Explorer](#).
- Kafka connector, see [Ingest data from Kafka into Azure Data Explorer](#).

Ingestion using integration services

- Azure Data Factory (ADF), a fully managed data integration service for analytic workloads in Azure, to copy data to and from Azure Data Explorer using [supported data stores and formats](#). For more information, see [Copy data from Azure Data Factory to Azure Data Explorer](#).

Programmatic ingestion

Azure Data Explorer provides SDKs that can be used for query and data ingestion. Programmatic ingestion is optimized for reducing ingestion costs (COGs), by minimizing storage transactions during and following the ingestion process.

Available SDKs and open-source projects:

Kusto offers client SDK that can be used to ingest and query data with:

- [Python SDK](#)
- [.NET SDK](#)
- [Java SDK](#)
- [Node SDK](#)
- [REST API](#)

Programmatic ingestion techniques:

- Ingesting data through the Azure Data Explorer data management service (high-throughput and reliable ingestion):

Batch ingestion (provided by SDK): the client uploads the data to Azure Blob storage (designated by the Azure Data Explorer data management service) and posts a notification to an Azure Queue. Batch ingestion is the recommended technique for high-volume, reliable, and cheap data ingestion.

- Ingesting data directly into the Azure Data Explorer engine (most appropriate for exploration and prototyping):
 - **Inline ingestion:** control command (.ingest inline) containing in-band data is intended for ad hoc testing purposes.
 - **Ingest from query:** control command (.set, .set-or-append, .set-or-replace) that points to query results is used for generating reports or small temporary tables.

- **Ingest from storage:** control command (.ingest into) with data stored externally (for example, Azure Blob Storage) allows efficient bulk ingestion of data.

Latency of different methods:

METHOD	LATENCY
Inline ingestion	Immediate
Ingest from query	Query time + processing time
Ingest from storage	Download time + processing time
Queued ingestion	Batching time + processing time

Processing time depends on the data size, less than a few seconds. Batching time defaults to 5 minutes.

Choosing the most appropriate ingestion method

Before you start to ingest data, you should ask yourself the following questions.

- Where does my data reside?
- What is the data format, and can it be changed?
- What are the required fields to be queried?
- What is the expected data volume and velocity?
- How many event types are expected (reflected as the number of tables)?
- How often is the event schema expected to change?
- How many nodes will generate the data?
- What is the source OS?
- What are the latency requirements?
- Can one of the existing managed ingestion pipelines be used?

For organizations with an existing infrastructure that are based on a messaging service like Event Hub and IoT Hub, using a connector is likely the most appropriate solution. Queued ingestion is appropriate for large data volumes.

Supported data formats

For all ingestion methods other than ingest from query, format the data so that Azure Data Explorer can parse it.

- The supported data formats are: TXT, CSV, TSV, TSVE, PSV, SCSV, SOH, JSON (line-separated, multi-line), Avro, and Parquet.
- Supports ZIP and GZIP compression.

NOTE

When data is being ingested, data types are inferred based on the target table columns. If a record is incomplete or a field cannot be parsed as the required data type, the corresponding table columns will be populated with null values.

Ingestion recommendations and limitations

- The effective retention policy of ingested data is derived from the database's retention policy. See [retention](#)

[policy](#) for details. Ingesting data requires **Table ingestor** or **Database ingestor** permissions.

- Ingestion supports a maximum file size of 5 GB. The recommendation is to ingest files between 100 MB and 1 GB.

Schema mapping

Schema mapping helps bind source data fields to destination table columns.

- [CSV Mapping](#) (optional) works with all ordinal-based formats. It can be performed using the ingest command parameter or [pre-created on the table](#) and referenced from the ingest command parameter.
- [JSON Mapping](#) (mandatory) and [Avro mapping](#) (mandatory) can be performed using the ingest command parameter. They can also be [pre-created on the table](#) and referenced from the ingest command parameter.

Next steps

[Ingest data from Event Hub into Azure Data Explorer](#)

[Ingest data using Event Grid subscription into Azure Data Explorer](#)

[Ingest data from Kafka into Azure Data Explorer](#)

[Ingest data using the Azure Data Explorer Python library](#)

[Ingest data using the Azure Data Explorer Node library](#)

[Ingest data using the Azure Data Explorer .NET Standard SDK \(Preview\)](#)

[Ingest data from Logstash to Azure Data Explorer](#)

Data visualization with Azure Data Explorer

7/2/2019 • 2 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data that is used to build complex analytics solutions for vast amounts of data. Azure Data Explorer integrates with various visualization tools, so you can visualize your data and share the results across your organization. This data can be transformed into actionable insights to make an impact on your business.

Data visualization and reporting is a critical step in the data analytics process. Azure Data Explorer supports many BI services so you can use the one that best fits your scenario and budget.

Kusto query language visualizations

The Kusto query language [render operator](#) offers various visualizations such as tables, pie charts, and bar charts to depict query results. Query visualizations are helpful in anomaly detection and forecasting, machine learning, and more.

Power BI

Azure Data Explorer provides the capability to connect to [Power BI](#) using various methods:

- [Built-in native Power BI connector](#)
- [Query import from Azure Data Explorer into Power BI](#)
- [SQL query](#)

Microsoft Excel

Azure Data Explorer provides the capability to connect to [Microsoft Excel](#) using the built-in native Excel connector, or import a query from Azure Data Explorer into Excel.

Grafana

Grafana provides an Azure Data Explorer plugin that enables you to visualize data from Azure Data Explorer. You set up [Azure Data Explorer as a data source for Grafana, and then visualize the data](#).

ODBC connector

Azure Data Explorer provides an [Open Database Connectivity \(ODBC\) connector](#) so any application that supports ODBC can connect to Azure Data Explorer.

Tableau

Azure Data Explorer provides the capability to connect to [Tableau](#) using the [ODBC connector](#) and then [visualize the data in Tableau](#).

Qlik

Azure Data Explorer provides the capability to connect to [Qlik](#) using the [ODBC connector](#) and then create Qlik Sense dashboards and visualize the data. Using the following video, you can learn to visualize Azure Data Explorer data with Qlik.

Sisense

Azure Data Explorer provides the capability to connect to [Sisense](#) using the JDBC connector. You [set up Azure Data Explorer as a data source for Sisense, and then visualize the data.](#)

Create an Azure Data Explorer cluster and database by using Azure CLI

9/26/2019 • 3 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast, fully managed data analytics service for real-time analysis on large volumes of data streaming from applications, websites, IoT devices, and more. To use Azure Data Explorer, you first create a cluster, and create one or more databases in that cluster. Then you ingest (load) data into a database so that you can run queries against it. In this article, you create a cluster and a database by using Azure CLI.

Prerequisites

To complete this article, you need an Azure subscription. If you don't have one, [create a free account](#) before you begin.

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select Try It in the upper-right corner of a code block. Selecting Try It doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com , or select the Launch Cloud Shell button to open Cloud Shell in your browser.	
Select the Cloud Shell button on the top-right menu bar in the Azure portal .	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.
4. Select **Enter** to run the code.

If you choose to install and use the Azure CLI locally, this article requires the Azure CLI version 2.0.4 or later. Run `az --version` to check your version. If you need to install or upgrade, see [Install the Azure CLI](#).

Configure the CLI parameters

The following steps are not required if you're running commands in Azure Cloud Shell. If you're running the CLI locally, follow these steps to sign in to Azure and to set your current subscription:

1. Run the following command to sign in to Azure:

```
az login
```

2. Set the subscription where you want your cluster to be created. Replace `MyAzureSub` with the name of the Azure subscription that you want to use:

```
az account set --subscription MyAzureSub
```

Create the Azure Data Explorer cluster

1. Create your cluster by using the following command:

```
az kusto cluster create --name azureclitest --sku D11_v2 --resource-group testrg
```

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
name	<i>azureclitest</i>	The desired name of your cluster.
sku	<i>D13_v2</i>	The SKU that will be used for your cluster.
resource-group	<i>testrg</i>	The resource group name where the cluster will be created.

There are additional optional parameters that you can use, such as the capacity of the cluster.

2. Run the following command to check whether your cluster was successfully created:

```
az kusto cluster show --name azureclitest --resource-group testrg
```

If the result contains `provisioningState` with the `Succeeded` value, then the cluster was successfully created.

Create the database in the Azure Data Explorer cluster

1. Create your database by using the following command:

```
az kusto database create --cluster-name azureclitest --name clidatabase --resource-group testrg --soft-delete-period P365D --hot-cache-period P31D
```

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
cluster-name	<i>azureclitest</i>	The name of your cluster where the database will be created.
name	<i>clidatabase</i>	The name of your database.
resource-group	<i>testrg</i>	The resource group name where the cluster will be created.

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
soft-delete-period	P365D	Signifies the amount of time that data will be kept available to query. See retention policy for more information.
hot-cache-period	P31D	Signifies the amount of time that data will be kept in cache. See cache policy for more information.

- Run the following command to see the database that you created:

```
az kusto database show --name clidatabase --resource-group testrg --cluster-name azureclitest
```

You now have a cluster and a database.

Clean up resources

- If you plan to follow our other articles, keep the resources you created.
- To clean up resources, delete the cluster. When you delete a cluster, it also deletes all the databases in it. Use the following command to delete your cluster:

```
az kusto cluster delete --name azureclitest --resource-group testrg
```

Next steps

- [Ingest data using the Azure Data Explorer Python library](#)

Create an Azure Data Explorer cluster and database by using PowerShell

9/26/2019 • 3 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast, fully managed data analytics service for real-time analysis on large volumes of data streaming from applications, websites, IoT devices, and more. To use Azure Data Explorer, you first create a cluster, and create one or more databases in that cluster. Then you ingest (load) data into a database so that you can run queries against it. In this article, you create a cluster and a database by using PowerShell. You can run PowerShell cmdlets and scripts on Windows, Linux, or in [Azure Cloud Shell](#) with [Az.Kusto](#) to create and configure Azure Data Explorer clusters and databases.

Prerequisites

NOTE

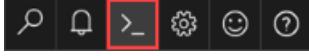
This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

If you don't have an Azure subscription, [create a free account](#) before you begin.

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select Try It in the upper-right corner of a code block. Selecting Try It doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com , or select the Launch Cloud Shell button to open Cloud Shell in your browser.	
Select the Cloud Shell button on the top-right menu bar in the Azure portal .	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.

4. Select **Enter** to run the code.

If you choose to install and use the Azure CLI locally, this article requires the Azure CLI version 2.0.4 or later. Run `az --version` to check your version. If you need to install or upgrade, see [Install the Azure CLI](#).

Configure parameters

The following steps are not required if you're running commands in Azure Cloud Shell. If you're running the CLI locally, follow steps 1 & 2 to sign in to Azure and to set your current subscription:

1. Run the following command to sign in to Azure:

```
Connect-AzAccount
```

2. Set the subscription where you want your cluster to be created:

```
Set-AzContext -SubscriptionId "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
```

3. When running Azure CLI locally or in the Azure Cloud Shell, you need to install the Az.Kusto module on your device:

```
Install-Module -Name Az.Kusto
```

Create the Azure Data Explorer cluster

1. Create your cluster by using the following command:

```
New-AzKustoCluster -ResourceGroupName testrg -Name mykustocluster -Location 'Central US' -Sku D13_v2 -Capacity 10
```

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
Name	<i>mykustocluster</i>	The desired name of your cluster.
Sku	<i>D13_v2</i>	The SKU that will be used for your cluster.
ResourceGroupName	<i>testrg</i>	The resource group name where the cluster will be created.

There are additional optional parameters that you can use, such as the capacity of the cluster.

2. Run the following command to check whether your cluster was successfully created:

```
Get-AzKustoCluster -Name mykustocluster -ResourceGroupName testrg
```

If the result contains `provisioningState` with the `Succeeded` value, then the cluster was successfully created.

Create the database in the Azure Data Explorer cluster

1. Create your database by using the following command:

```
New-AzKustoDatabase -ResourceGroupName testrg -ClusterName mykustocluster -Name mykustodatabase -  
SoftDeletePeriod 3650:00:00:00 -HotCachePeriod 3650:00:00:00
```

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
ClusterName	<i>mykustocluster</i>	The name of your cluster where the database will be created.
Name	<i>mykustodatabase</i>	The name of your database.
ResourceGroupName	<i>testrg</i>	The resource group name where the cluster will be created.
SoftDeletePeriod	<i>3650:00:00:00</i>	The amount of time that data will be kept available to query.
HotCachePeriod	<i>3650:00:00:00</i>	The amount of time that data will be kept in cache.

- Run the following command to see the database that you created:

```
Get-AzKustoDatabase -ClusterName mykustocluster -ResourceGroupName testrg -Name mykustodatabase
```

You now have a cluster and a database.

Clean up resources

- If you plan to follow our other articles, keep the resources you created.
- To clean up resources, delete the cluster. When you delete a cluster, it also deletes all the databases in it. Use the following command to delete your cluster:

```
Remove-AzKustoCluster -ResourceGroupName testrg -Name mykustocluster
```

Next steps

- [Additional Az.Kusto commands](#)
- [Ingest data using the Azure Data Explorer .NET Standard SDK \(Preview\)](#)

Create an Azure Data Explorer cluster and database by using C#

12/2/2019 • 3 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast, fully managed data analytics service for real-time analysis on large volumes of data streaming from applications, websites, IoT devices, and more. To use Azure Data Explorer, you first create a cluster, and create one or more databases in that cluster. Then you ingest (load) data into a database so that you can run queries against it. In this article, you create a cluster and a database by using C#.

Prerequisites

- If you don't have Visual Studio 2019 installed, you can download and use the [free Visual Studio 2019 Community Edition](#). Make sure that you enable **Azure development** during the Visual Studio setup.
- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.

Install C# Nuget

- Install the [Azure Data Explorer \(Kusto\) nuget package](#).
- Install the [Microsoft.IdentityModel.Clients.ActiveDirectory nuget package](#) for authentication.

Authentication

For running the examples in this article, we need an Azure AD Application and service principal that can access resources. Check [create an Azure AD application](#) to create a free Azure AD Application and add role assignment at the subscription scope. It also shows how to get the `Directory (tenant) ID`, `Application ID`, and `Client Secret`.

Create the Azure Data Explorer cluster

1. Create your cluster by using the following code:

```

var tenantId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx";//Directory (tenant) ID
var clientId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx";//Application ID
var clientSecret = "xxxxxxxxxxxxxx"; //Client Secret
var subscriptionId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx";
var authenticationContext = new AuthenticationContext($"https://login.windows.net/{tenantId}");
var credential = new ClientCredential(clientId, clientSecret);
var result = await authenticationContext.AcquireTokenAsync(resource:
"https://management.core.windows.net/", clientCredential: credential);

var credentials = new TokenCredentials(result.AccessToken, result.AccessTokenType);

var kustoManagementClient = new KustoManagementClient(credentials)
{
    SubscriptionId = subscriptionId
};

var resourceGroupName = "testrg";
var clusterName = "mykustocluster";
var location = "Central US";
var skuName = "Standard_D13_v2";
var tier = "Standard";
var capacity = 5;
var sku = new AzureSku(skuName, tier, capacity);
var cluster = new Cluster(location, sku);
await kustoManagementClient.Clusters.CreateOrUpdateAsync(resourceGroupName, clusterName, cluster);

```

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
clusterName	<i>mykustocluster</i>	The desired name of your cluster.
skuName	<i>Standard_D13_v2</i>	The SKU that will be used for your cluster.
tier	<i>Standard</i>	The SKU tier.
capacity	<i>number</i>	The number of instances of the cluster.
resourceGroupName	<i>testrg</i>	The resource group name where the cluster will be created.

NOTE

Create a cluster is a long running operation, so it's highly recommended to use `CreateOrUpdateAsync`, instead of `CreateOrUpdate`.

- Run the following command to check whether your cluster was successfully created:

```
kustoManagementClient.Clusters.Get(resourceGroupName, clusterName);
```

If the result contains `ProvisioningState` with the `Succeeded` value, then the cluster was successfully created.

Create the database in the Azure Data Explorer cluster

- Create your database by using the following code:

```

var hotCachePeriod = new TimeSpan(3650, 0, 0, 0);
var softDeletePeriod = new TimeSpan(3650, 0, 0, 0);
var databaseName = "mykustodatabase";
var database = new Database(location: location, softDeletePeriod: softDeletePeriod, hotCachePeriod:
hotCachePeriod);

await kustoManagementClient.Databases.CreateOrUpdateAsync(resourceGroupName, clusterName,
databaseName, database);

```

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
clusterName	<i>mykustocluster</i>	The name of your cluster where the database will be created.
databaseName	<i>mykustodatabase</i>	The name of your database.
resourceGroupName	<i>testrg</i>	The resource group name where the cluster will be created.
softDeletePeriod	<i>3650:00:00:00</i>	The amount of time that data will be kept available to query.
hotCachePeriod	<i>3650:00:00:00</i>	The amount of time that data will be kept in cache.

- Run the following command to see the database that you created:

```
kustoManagementClient.Databases.Get(resourceGroupName, clusterName, databaseName);
```

You now have a cluster and a database.

Clean up resources

- If you plan to follow our other articles, keep the resources you created.
- To clean up resources, delete the cluster. When you delete a cluster, it also deletes all the databases in it. Use the following command to delete your cluster:

```
kustoManagementClient.Clusters.Delete(resourceGroupName, clusterName);
```

Next steps

- [Ingest data using the Azure Data Explorer .NET Standard SDK \(Preview\)](#)

Create an Azure Data Explorer cluster and database by using Python

10/7/2019 • 3 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast, fully managed data analytics service for real-time analysis on large volumes of data streaming from applications, websites, IoT devices, and more. To use Azure Data Explorer, you first create a cluster, and create one or more databases in that cluster. Then you ingest (load) data into a database so that you can run queries against it. In this article, you create a cluster and a database by using Python.

Prerequisites

If you don't have an Azure subscription, create a [free Azure account](#) before you begin.

Install Python package

To install the Python package for Azure Data Explorer (Kusto), open a command prompt that has Python in its path. Run this command:

```
pip install azure-common  
pip install azure-mgmt-kusto
```

Authentication

For running the examples in this article, we need an Azure AD Application and service principal that can access resources. Check [create an Azure AD application](#) to create a free Azure AD Application and add role assignment at the subscription scope. It also shows how to get the `Directory (tenant) ID`, `Application ID`, and `Client Secret`.

Create the Azure Data Explorer cluster

1. Create your cluster by using the following command:

```

from azure.mgmt.kusto import KustoManagementClient
from azure.mgmt.kusto.models import Cluster, AzureSku
from azure.common.credentials import ServicePrincipalCredentials

#Directory (tenant) ID
tenant_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
#Application ID
client_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
#Client Secret
client_secret = "xxxxxxxxxxxxxx"
subscription_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
credentials = ServicePrincipalCredentials(
    client_id=client_id,
    secret=client_secret,
    tenant=tenant_id
)

location = 'Central US'
sku_name = 'Standard_D13_v2'
capacity = 5
tier = "Standard"
resource_group_name = 'testrg'
cluster_name = 'mykustocluster'
cluster = Cluster(location=location, sku=AzureSku(name=sku_name, capacity=capacity, tier=tier))

kustoManagementClient = KustoManagementClient(credentials, subscription_id)

cluster_operations = kustoManagementClient.clusters

poller = cluster_operations.create_or_update(resource_group_name, cluster_name, cluster)

```

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
cluster_name	<i>mykustocluster</i>	The desired name of your cluster.
sku_name	<i>Standard_D13_v2</i>	The SKU that will be used for your cluster.
tier	<i>Standard</i>	The SKU tier.
capacity	<i>number</i>	The number of instances of the cluster.
resource_group_name	<i>testrg</i>	The resource group name where the cluster will be created.

NOTE

Create a cluster is a long running operation. Method **create_or_update** returns an instance of LROPoller, see [LROPoller class](#) to get more information.

- Run the following command to check whether your cluster was successfully created:

```

cluster_operations.get(resource_group_name = resource_group_name, cluster_name= clusterName,
custom_headers=None, raw=False)

```

If the result contains `provisioningState` with the `Succeeded` value, then the cluster was successfully created.

Create the database in the Azure Data Explorer cluster

1. Create your database by using the following command:

```
from azure.mgmt.kusto.models import Database
from datetime import timedelta

softDeletePeriod = timedelta(days=3650)
hotCachePeriod = timedelta(days=3650)
databaseName="mykustodatabase"

database_operations = kusto_management_client.databases
_database = Database(location=location,
    soft_delete_period=softDeletePeriod,
    hot_cache_period=hotCachePeriod)

#Returns an instance of LROPoller, see
https://docs.microsoft.com/python/api/msrest/msrest.polling.lropoller?view=azure-python
poller = database_operations.create_or_update(resource_group_name = resource_group_name, cluster_name =
clusterName, database_name = databaseName, parameters = _database)
```

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
cluster_name	<i>mykustocluster</i>	The name of your cluster where the database will be created.
database_name	<i>mykustodatabase</i>	The name of your database.
resource_group_name	<i>testrg</i>	The resource group name where the cluster will be created.
soft_delete_period	<i>3650 days, 0:00:00</i>	The amount of time that data will be kept available to query.
hot_cache_period	<i>3650 days, 0:00:00</i>	The amount of time that data will be kept in cache.

2. Run the following command to see the database that you created:

```
database_operations.get(resource_group_name = resource_group_name, cluster_name = clusterName,
database_name = databaseName)
```

You now have a cluster and a database.

Clean up resources

- If you plan to follow our other articles, keep the resources you created.
- To clean up resources, delete the cluster. When you delete a cluster, it also deletes all the databases in it. Use the following command to delete your cluster:

```
cluster_operations.delete(resource_group_name = resource_group_name, cluster_name = clusterName)
```

Next steps

- [Ingest data using the Azure Data Explorer Python library](#)

Create an Azure Data Explorer cluster and database by using an Azure Resource Manager template

12/5/2019 • 3 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. To use Azure Data Explorer, you first create a cluster, and create one or more databases in that cluster. Then you ingest (load) data into a database so that you can run queries against it.

In this article, you create an Azure Data Explorer cluster and database by using an [Azure Resource Manager template](#). The article shows how to define which resources are deployed and how to define parameters that are specified when the deployment is executed. You can use this template for your own deployments, or customize it to meet your requirements. For information about creating templates, see [authoring Azure Resource Manager templates](#). For the JSON syntax and properties to use in a template, see [Microsoft.Kusto resource types](#).

If you don't have an Azure subscription, [create a free account](#) before you begin.

Azure Resource Manager template for cluster and database creation

In this article, you use an [existing quickstart template](#)

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "clusters_kustocluster_name": {
            "type": "string",
            "defaultValue": "[concat('kusto', uniqueString(resourceGroup().id))]",
            "metadata": {
                "description": "Name of the cluster to create"
            }
        },
        "databases_kustodb_name": {
            "type": "string",
            "defaultValue": "kustodb",
            "metadata": {
                "description": "Name of the database to create"
            }
        },
        "location": {
            "type": "string",
            "defaultValue": "[resourceGroup().location]",
            "metadata": {
                "description": "Location for all resources."
            }
        }
    },
    "variables": {},
    "resources": [
        {
            "name": "[parameters('clusters_kustocluster_name')]",
            "type": "Microsoft.Kusto/clusters",
            "sku": {
                "name": "Standard_D13_v2",
                "tier": "Standard",
                "capacity": 2
            },
            "apiVersion": "2019-09-07",
            "location": "[parameters('location')]",
            "tags": {
                "Created By": "GitHub quickstart template"
            }
        },
        {
            "name": "[concat(parameters('clusters_kustocluster_name'), '/', parameters('databases_kustodb_name'))]",
            "type": "Microsoft.Kusto/clusters/databases",
            "apiVersion": "2019-09-07",
            "location": "[parameters('location')]",
            "dependsOn": [
                "[resourceId('Microsoft.Kusto/clusters', parameters('clusters_kustocluster_name'))]"
            ],
            "properties": {
                "softDeletePeriodInDays": 365,
                "hotCachePeriodInDays": 31
            }
        }
    ]
}
```

To find more template samples, see [Azure Quickstart Templates](#).

Deploy the template and verify template deployment

You can deploy the Azure Resource Manager template by [using the Azure portal](#) or [using powershell](#).

Use the Azure portal to deploy the template and verify template deployment

1. To create a cluster and database, use the following button to start the deployment. Right-click and select **Open in new window**, so you can follow the rest of the steps in this article.



The **Deploy to Azure** button takes you to the Azure portal to fill out a deployment form.

The screenshot shows the Azure portal interface for deploying a template. On the left is a sidebar with various icons. The main area has a title bar "Home > Create a cluster a database" and a sub-header "Create a cluster a database" with "Azure quickstart template".

TEMPLATE section: Shows a template named "101-kusto-cluster-database" with "2 resources". It includes "Edit template" and "Edit param.." buttons, both of which are highlighted with red boxes.

BASICS section (highlighted with a red box): Contains fields for "Subscription" (dropdown), "Resource group" (dropdown with "Select a resource group" and "Create new" options), and "Location" (dropdown set to "(US) Central US").

SETTINGS section: Contains three fields with dynamic placeholder values: "Clusters_kustocluster_name" with "[concat('kusto', uniqueString(resourceGroup().id))]", "Databases_kustodb_name" with "kustodb", and "Location" with "[resourceGroup().location]".

TERMS AND CONDITIONS section: Includes links to "Template information", "Azure Marketplace Terms", and "Azure Marketplace". A terms and conditions agreement box is present, with the "I agree to the terms and conditions stated above" checkbox highlighted with a red box. A "Purchase" button at the bottom is also highlighted with a red box.

You can [edit and deploy the template in the Azure portal](#) by using the form.

2. Complete **BASICS** and **SETTINGS** sections. Select unique cluster and database names. It takes a few minutes to create an Azure Data Explorer cluster and database.
3. To verify the deployment, you open the resource group in the [Azure portal](#) to find your new cluster and database.

Use powershell to deploy the template and verify template deployment

Deploy the template using powershell

1. Select **Try it** from the following code block, and then follow the instructions to sign in to the Azure Cloud shell.

```
$projectName = Read-Host -Prompt "Enter a project name that is used for generating resource names"
.setLocation = Read-Host -Prompt "Enter the location (i.e. centralus)"
$resourceGroupName = "${projectName}rg"
$clusterName = "${projectName}cluster"
$parameters = @{}
$parameters.Add("clusters_kustocluster_name", $clusterName)
$templateUri = "https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-kusto-cluster-database/azuredeploy.json"
New-AzResourceGroup -Name $resourceGroupName -Location $location
New-AzResourceGroupDeployment -ResourceGroupName $resourceGroupName -TemplateUri $templateUri -TemplateParameterObject $parameters
Write-Host "Press [ENTER] to continue ..."
```

2. Select **Copy** to copy the PowerShell script.
3. Right-click the shell console, and then select **Paste**. It takes a few minutes to create an Azure Data Explorer cluster and database.

Verify the deployment using PowerShell

To verify the deployment, use the following Azure PowerShell script. If the Cloud Shell is still open, you don't need to copy/run the first line (Read-Host). For more information regarding managing Azure Data Explorer resources in PowerShell, read [Az.Kusto](#).

```
$projectName = Read-Host -Prompt "Enter the same project name that you used in the last procedure"

Install-Module -Name Az.Kusto
$resourceGroupName = "${projectName}rg"
$clusterName = "${projectName}cluster"

Get-AzKustoCluster -ResourceGroupName $resourceGroupName -Name $clusterName
Write-Host "Press [ENTER] to continue ..."
```

Clean up resources

When the Azure resources are no longer needed, clean up the resources you deployed by deleting the resource group.

Clean up resources using the Azure portal

Delete the resources in the Azure portal by following the steps in [clean up resources](#).

Clean up resources using PowerShell

If the Cloud Shell is still open, you don't need to copy/run the first line (Read-Host).

```
$projectName = Read-Host -Prompt "Enter the same project name that you used in the last procedure"
$resourceGroupName = "${projectName}rg"

Remove-AzResourceGroup -ResourceGroupName $resourceGroupName

Write-Host "Press [ENTER] to continue ..."
```

Next steps

[Ingest data into Azure Data Explorer cluster and database](#)

Ingest data from Event Hub into Azure Data Explorer

12/2/2019 • 6 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Azure Data Explorer offers ingestion (data loading) from Event Hubs, a big data streaming platform and event ingestion service. [Event Hubs](#) can process millions of events per second in near real-time. In this article, you create an event hub, connect to it from Azure Data Explorer and see data flow through the system.

Prerequisites

- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.
- [A test cluster and database](#).
- [A sample app](#) that generates data and sends it to an event hub. Download the sample app to your system.
- [Visual Studio 2019](#) to run the sample app.

Sign in to the Azure portal

Sign in to the [Azure portal](#).

Create an event hub

In this article, you generate sample data and send it to an event hub. The first step is to create an event hub. You do this by using an Azure Resource Manager template in the Azure portal.

1. To create an event hub, use the following button to start the deployment. Right-click and select **Open in new window**, so you can follow the rest of the steps in this article.



The **Deploy to Azure** button takes you to the Azure portal to fill out a deployment form.

The screenshot shows the Microsoft Azure portal interface. The title bar says "Create an EventHubs namespace, Event Hub, & consumer group". The left sidebar has a "TEMPLATES" section with a "201-event-hubs-create-event-hub-and-consumer-group" item selected. The main content area is titled "BASICS" and contains fields for "Subscription" (dropdown), "Resource group" (dropdown with "Select a resource group" and "Create new" link), and "Location" (dropdown with "West US"). To the right of the template list are three buttons: "Edit template", "Edit parameters", and "Learn more".

2. Select the subscription where you want to create the event hub, and create a resource group named *test-hub-rg*.

A dialog box for creating a new resource group. It has a red box around the "Create new" button. The "Name" field is filled with "test-hub-rg". The "OK" button is highlighted with a red box. At the bottom, there is a "Sku Capacity" dropdown set to "1".

3. Fill out the form with the following information.

BASICS

* Subscription

* Resource group

[Create new](#)

* Location

SETTINGS

* Namespace Name ⓘ

Eventhub SKU ⓘ

Sku Capacity ⓘ

* Event Hub Name ⓘ

* Consumer Group Name ⓘ

Location ⓘ

TERMS AND CONDITIONS

[Template information](#) | [Azure Marketplace](#)
[Terms](#) | [Azure Marketplace](#)

By clicking "Purchase," I (a) agree to the applicable legal terms associated with the offering; (b) authorize Microsoft to charge or bill my current payment method for the fees associated with the offering(s), including

I agree to the terms and conditions stated above

Purchase

Use defaults for any settings not listed in the following table.

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
Subscription	Your subscription	Select the Azure subscription that you want to use for your event hub.
Resource group	test-hub-rg	Create a new resource group.

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
Location	<i>West US</i>	Select <i>West US</i> for this article. For a production system, select the region that best meets your needs. Create the event hub namespace in the same Location as the Kusto cluster for best performance (most important for event hub namespaces with high throughput).
Namespace name	A unique namespace name	Choose a unique name that identifies your namespace. For example, <i>mytestnamespace</i> . The domain name <i>servicebus.windows.net</i> is appended to the name you provide. The name can contain only letters, numbers, and hyphens. The name must start with a letter, and it must end with a letter or number. The value must be between 6 and 50 characters long.
Event hub name	<i>test-hub</i>	The event hub sits under the namespace, which provides a unique scoping container. The event hub name must be unique within the namespace.
Consumer group name	<i>test-group</i>	Consumer groups enable multiple consuming applications to each have a separate view of the event stream.

4. Select **Purchase**, which acknowledges that you're creating resources in your subscription.
5. Select **Notifications** on the toolbar to monitor the provisioning process. It might take several minutes for the deployment to succeed, but you can move on to the next step now.



Create a target table in Azure Data Explorer

Now you create a table in Azure Data Explorer, to which Event Hubs will send data. You create the table in the cluster and database provisioned in **Prerequisites**.

1. In the Azure portal, navigate to your cluster then select **Query**.

2. Copy the following command into the window and select **Run** to create the table (TestTable) which will receive the ingested data.

```
.create table TestTable (TimeStamp: datetime, Name: string, Metric: int, Source:string)
```

TableName	Schema	DatabaseName	Folder
TestTable	{"Name":"TestTable","OrderedC...	TestDatabase	

3. Copy the following command into the window and select **Run** to map the incoming JSON data to the column names and data types of the table (TestTable).

```
.create table TestTable ingestion json mapping 'TestMapping'
[{"column": "TimeStamp", "path": "$.TimeStamp", "datatype": "datetime"}, {"column": "Name", "path": "$.name", "datatype": "string"}, {"column": "Metric", "path": "$.metric", "datatype": "int"}, {"column": "Source", "path": "$.source", "datatype": "string"}]
```

Connect to the event hub

Now you connect to the event hub from Azure Data Explorer. When this connection is in place, data that flows into the event hub streams to the test table you created earlier in this article.

1. Select **Notifications** on the toolbar to verify that the event hub deployment was successful.
2. Under the cluster you created, select **Databases** then **TestDatabase**.

mydataexplorercluster - Databases

Search (Ctrl+ /)

Add database Refresh Query Delete

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Query

Settings

Scale up

Scale out

Properties

Locks

Automation script

Data

Databases

DATABASE	SIZE	RETENTION PERIOD	CACHE PERIOD
TestDatabase	0 Bytes	3650 days	31 days

3. Select **Data ingestion** and **Add data connection**. Then fill out the form with the following information. Select **Create** when you are finished.

TestDatabase (mydataexplorercluster/TestDatabase) - Data ingestion

Search (Ctrl+ /)

Add data connection Refresh

Overview

Permissions

Query

Settings

Data ingestion

Properties

No results

Select connection type

Event Hub

Data source

* Data connection name ✓

Subscription

Event Hub namespace

Select an Event Hub namespace

Event Hub

Select an Event Hub

Consumer group

Select a consumer group

Event system properties

0 selected

Target table

My data includes routing info

No Yes

Table

Data format

JSON

* Column mapping ✓

Create

Data Source:

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
Data connection name	<i>test-hub-connection</i>	The name of the connection you want to create in Azure Data Explorer.
Event hub namespace	A unique namespace name	The name you chose earlier that identifies your namespace.
Event hub	<i>test-hub</i>	The event hub you created.
Consumer group	<i>test-group</i>	The consumer group defined in the event hub you created.
Event system properties	Select relevant properties	The Event Hub system properties . If there are multiple records per event message, the system properties will be added to the first one. When adding system properties, create or update table schema and mapping to include the selected properties.

Target table:

There are two options for routing the ingested data: *static* and *dynamic*. For this article, you use static routing, where you specify the table name, data format, and mapping. Therefore, leave **My data includes routing info** unselected.

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
Table	<i>TestTable</i>	The table you created in TestDatabase .
Data format	JSON	Supported formats are Avro, CSV, JSON, MULTILINE JSON, PSV, SOHSV, SCSV, TSV, TSVE and TXT. Supported compression options: GZip
Column mapping	<i>TestMapping</i>	The mapping you created in TestDatabase , which maps incoming JSON data to the column names and data types of TestTable . Required for JSON, MULTILINE JSON, or AVRO, and optional for other formats.

NOTE

- Select **My data includes routing info** to use dynamic routing, where your data includes the necessary routing information as seen in the [sample app](#) comments. If both static and dynamic properties are set, the dynamic properties override the static ones.
- Only events enqueued after you create the data connection are ingested.
- Enable GZip compression for static routing by opening a [support request in the Azure portal](#). Enable GZip compression for dynamic routing as seen in the [sample app](#).
- Avro format and event system properties aren't supported on compression payload.

Copy the connection string

When you run the [sample app](#) listed in Prerequisites, you need the connection string for the event hub namespace.

1. Under the event hub namespace you created, select **Shared access policies**, then **RootManageSharedAccessKey**.

The screenshot shows the 'Shared access policies' page for the 'test-hub-docs' Event Hubs Namespace. On the left, there's a sidebar with icons for Overview, Access control (IAM), Tags, Diagnose and solve problems, Events, and Settings. Under Settings, the 'Shared access policies' option is selected and highlighted with a red box. The main area shows a table with one row for 'RootManageSharedAccessKey'. A search bar at the top right and an 'Add' button are also visible.

2. Copy **Connection string - primary key**. You paste it in the next section.

The screenshot shows the 'Connection string' section for the 'RootManageSharedAccessKey' policy. It displays four fields: 'Primary key' (redacted), 'Secondary key' (redacted), 'Connection string-primary key' (redacted and highlighted with a red box), and 'Connection string-secondary key' (redacted). Each field has a small blue clipboard icon to its right.

Generate sample data

Use the [sample app](#) you downloaded to generate data.

1. Open the sample app solution in Visual Studio.
2. In the *program.cs* file, update the `connectionString` constant to the connection string you copied from the event hub namespace.

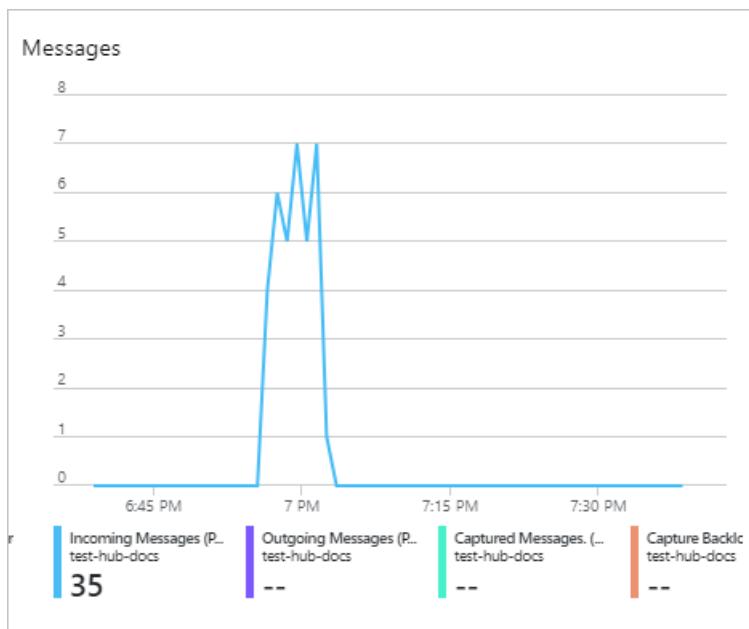
```
const string eventHubName = "test-hub";
// Copy the connection string ("Connection string-primary key") from your Event Hub namespace.
const string connectionString = @"<YourConnectionString>";
```

3. Build and run the app. The app sends messages to the event hub, and it prints out status every ten seconds.
4. After the app has sent a few messages, move on to the next step: reviewing the flow of data into your event hub and test table.

Review the data flow

With the app generating data, you can now see the flow of that data from the event hub to the table in your cluster.

1. In the Azure portal, under your event hub, you see the spike in activity while the app is running.



2. To check how many messages have made it to the database so far, run the following query in your test database.

```
TestTable
| count
```

3. To see the content of the messages, run the following query:

```
TestTable
```

The result set should look like the following:

TimeStamp	Name	Metric	Source
2018-09-18T18:19:17Z	name 0	0	EventHubMessage
2018-09-18T18:19:17Z	name 0	1	EventHubMessage
2018-09-18T18:19:17Z	name 0	2	EventHubMessage
2018-09-18T18:24:29Z	name 3	3	EventHubMessage
2018-09-18T18:24:29Z	name 3	4	EventHubMessage
2018-09-18T18:24:29Z	name 3	5	EventHubMessage
2018-09-18T18:29:39Z	name 6	6	EventHubMessage
2018-09-18T18:29:39Z	name 6	7	EventHubMessage
2018-09-18T18:29:39Z	name 6	8	EventHubMessage

NOTE

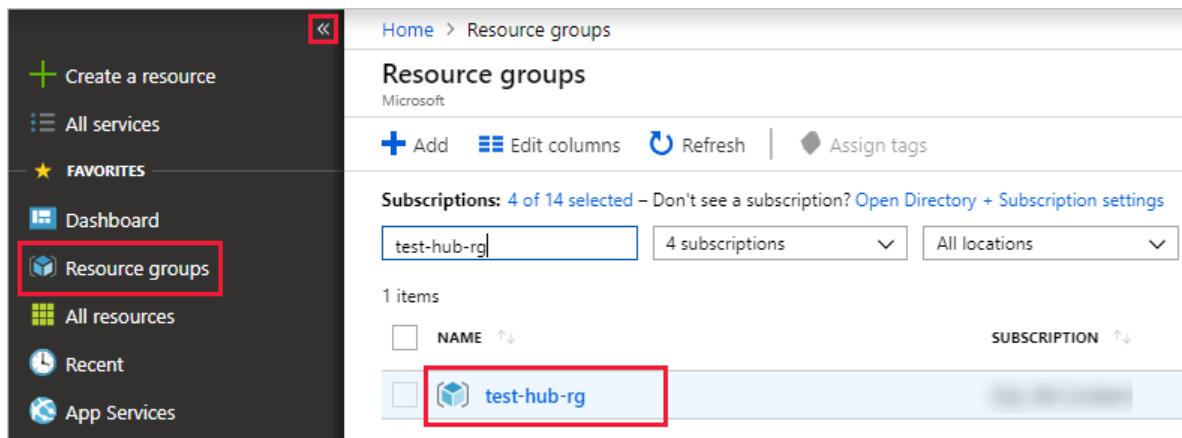
- Azure Data Explorer has an aggregation (batching) policy for data ingestion, designed to optimize the ingestion process. The policy is configured to 5 minutes or 500 MB of data, by default, so you may experience a latency. See [batching policy](#) for aggregation options.
- Event Hub ingestion includes Event Hub response time of 10 seconds or 1 MB.
- Configure your table to support streaming and remove the lag in response time. See [streaming policy](#).

Clean up resources

If you don't plan to use your event hub again, clean up **test-hub-rg**, to avoid incurring costs.

1. In the Azure portal, select **Resource groups** on the far left, and then select the resource group you created.

If the left menu is collapsed, select  to expand it.



The screenshot shows the Azure portal's Resource groups blade. On the left, there's a sidebar with links like 'Create a resource', 'All services', 'FAVORITES' (which includes 'Dashboard', 'Resource groups' [highlighted with a red box], 'All resources', 'Recent', and 'App Services'). The main area shows 'Resource groups' under 'Microsoft'. It has a search bar with 'test-hub-rg', a dropdown for '4 subscriptions', and a 'SUBSCRIPTION' dropdown. A table lists one item: 'test-hub-rg' with a blue cube icon, which is also highlighted with a red box.

2. Under **test-resource-group**, select **Delete resource group**.
3. In the new window, type the name of the resource group to delete (*test-hub-rg*), and then select **Delete**.

Next steps

- [Query data in Azure Data Explorer](#)

Create an Event Hub data connection for Azure Data Explorer by using Python

12/2/2019 • 2 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Azure Data Explorer offers ingestion (data loading) from Event Hubs, IoT Hubs, and blobs written to blob containers. In this article, you create an Event Hub data connection for Azure Data Explorer by using Python.

Prerequisites

- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.
- [Create a cluster and database](#)
- [Create table and column mapping](#)
- [Set database and table policies](#) (optional)
- [Create an Event Hub with data for ingestion](#).

Install Python package

To install the Python package for Azure Data Explorer (Kusto), open a command prompt that has Python in its path. Run the following command:

```
pip install azure-common  
pip install azure-mgmt-kusto
```

Authentication

To run the following example, you need an Azure Active Directory (Azure AD) application and service principal that can access resources. To create a free Azure AD application and add role assignment at the subscription level, see [Create an Azure AD application](#). You also need the directory (tenant) ID, application ID, and client secret.

Add an Event Hub data connection

The following example shows you how to add an Event Hub data connection programmatically. See [connect to the event hub](#) for adding an Event Hub data connection using the Azure portal.

```

from azure.mgmt.kusto import KustoManagementClient
from azure.mgmt.kusto.models import EventHubDataConnection
from azure.common.credentials import ServicePrincipalCredentials

#Directory (tenant) ID
tenant_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
#Application ID
client_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
#Client Secret
client_secret = "xxxxxxxxxxxxxx"
subscription_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
credentials = ServicePrincipalCredentials(
    client_id=client_id,
    secret=client_secret,
    tenant=tenant_id
)
kusto_management_client = KustoManagementClient(credentials, subscription_id)

resource_group_name = "testrg";
#The cluster and database that are created as part of the Prerequisites
cluster_name = "mykustocluster";
database_name = "mykustodatabase";
data_connection_name = "myeventhubconnect";
#The event hub that is created as part of the Prerequisites
event_hub_resource_id = "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxx/resourceGroups/xxxxxx/providers/Microsoft.EventHub/namespaces/xxxxxx/eventhubs/xxxxxx";
consumer_group = "$Default";
location = "Central US";
#The table and column mapping that are created as part of the Prerequisites
table_name = "StormEvents";
mapping_rule_name = "StormEvents_CSV_Mapping";
data_format = "csv";
#Returns an instance of LROPoller, check
https://docs.microsoft.com/python/api/msrest/msrest.polling.lropoller?view=azure-python
poller = kusto_management_client.data_connections.create_or_update(resource_group_name=resource_group_name,
cluster_name=cluster_name, database_name=database_name, data_connection_name=data_connection_name,
parameters=EventHubDataConnection(event_hub_resource_id=event_hub_resource_id, consumer_group=consumer_group,
location=location,
table_name=table_name,
mapping_rule_name=mapping_rule_name, data_format=data_format))

```

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
tenant_id	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx	Your tenant ID. Also known as directory ID.
subscriptionId	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx	The subscription ID that you use for resource creation.
client_id	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx	The client ID of the application that can access resources in your tenant.
client_secret	xxxxxxxxxxxxxx	The client secret of the application that can access resources in your tenant.
resource_group_name	testrg	The name of the resource group containing your cluster.
cluster_name	mykustocluster	The name of your cluster.

Setting	Suggested Value	Field Description
database_name	<i>mykustodatabase</i>	The name of the target database in your cluster.
data_connection_name	<i>myeventhubconnect</i>	The desired name of your data connection.
table_name	<i>StormEvents</i>	The name of the target table in the target database.
mapping_rule_name	<i>StormEvents_CSVMapping</i>	The name of your column mapping related to the target table.
data_format	<i>csv</i>	The data format of the message.
event_hub_resource_id	<i>Resource ID</i>	The resource ID of your Event Hub that holds the data for ingestion.
consumer_group	<i>\$Default</i>	The consumer group of your Event Hub.
location	<i>Central US</i>	The location of the data connection resource.

Clean up resources

To delete the data connection, use the following command:

```
kusto_management_client.data_connections.delete(resource_group_name=resource_group_name,
cluster_name=kusto_cluster_name, database_name=kusto_database_name,
data_connection_name=kusto_data_connection_name)
```

Create an Event Hub data connection for Azure Data Explorer by using C#

12/2/2019 • 2 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Azure Data Explorer offers ingestion (data loading) from Event Hubs, IoT Hubs, and blobs written to blob containers. In this article, you create an Event Hub data connection for Azure Data Explorer by using C#.

Prerequisites

- If you don't have Visual Studio 2019 installed, you can download and use the [free Visual Studio 2019 Community Edition](#). Make sure that you enable **Azure development** during the Visual Studio setup.
- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.
- [Create a cluster and database](#)
- [Create table and column mapping](#)
- [Set database and table policies](#) (optional)
- Create an [Event Hub with data for ingestion](#).

Install C# nuget

- Install the [Azure Data Explorer \(Kusto\) NuGet package](#).
- Install the [Microsoft.IdentityModel.Clients.ActiveDirectory NuGet package](#) for authentication.

Authentication

To run the following example, you need an Azure Active Directory (Azure AD) application and service principal that can access resources. To create a free Azure AD application and add role assignment at the subscription level, see [Create an Azure AD application](#). You also need the directory (tenant) ID, application ID, and client secret.

Add an Event Hub data connection

The following example shows you how to add an Event Hub data connection programmatically. See [connect to the event hub](#) for adding an Event Hub data connection using the Azure portal.

```

var tenantId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx";//Directory (tenant) ID
var clientId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx";//Application ID
var clientSecret = "xxxxxxxxxxxxxx"; //Client Secret
var subscriptionId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx";
var authenticationContext = new AuthenticationContext($"https://login.windows.net/{tenantId}");
var credential = new ClientCredential(clientId, clientSecret);
var result = await authenticationContext.AcquireTokenAsync(resource: "https://management.core.windows.net/",
clientCredential: credential);

var credentials = new TokenCredentials(result.AccessToken, result.AccessTokenType);

var kustoManagementClient = new KustoManagementClient(credentials)
{
    SubscriptionId = subscriptionId
};

var resourceGroupName = "testrg";
//The cluster and database that are created as part of the Prerequisites
var clusterName = "mykustocluster";
var databaseName = "mykustodatabase";
var dataConnectionName = "myeventhubconnect";
//The event hub that is created as part of the Prerequisites
var eventHubResourceId = "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxx-
xxxxxxxx/resourceGroups/xxxxxx/providers/Microsoft.EventHub/namespaces/xxxxxx/eventhubs/xxxxxx";
var consumerGroup = "$Default";
var location = "Central US";
//The table and column mapping are created as part of the Prerequisites
var tableName = "StormEvents";
var mappingRuleName = "StormEvents_CSV_Mapping";
var dataFormat = DataFormat.CSV;
await kustoManagementClient.DataConnections.CreateOrUpdateAsync(resourceGroupName, clusterName, databaseName,
dataConnectionName,
new EventHubDataConnection(eventHubResourceId, consumerGroup, location: location, tableName: tableName,
mappingRuleName: mappingRuleName, dataFormat: dataFormat));

```

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
tenantId	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx	Your tenant ID. Also known as directory ID.
subscriptionId	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx	The subscription ID that you use for resource creation.
clientId	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx	The client ID of the application that can access resources in your tenant.
clientSecret	xxxxxxxxxxxxxx	The client secret of the application that can access resources in your tenant.
resourceGroupName	testrg	The name of the resource group containing your cluster.
clusterName	mykustocluster	The name of your cluster.
databaseName	mykustodatabase	The name of the target database in your cluster.
dataConnectionName	myeventhubconnect	The desired name of your data connection.

Setting	Suggested Value	Field Description
tableName	<i>StormEvents</i>	The name of the target table in the target database.
mappingRuleName	<i>StormEvents_CSVMapping</i>	The name of your column mapping related to the target table.
dataFormat	<i>csv</i>	The data format of the message.
eventHubResourceId	<i>Resource ID</i>	The resource ID of your Event Hub that holds the data for ingestion.
consumerGroup	<i>\$Default</i>	The consumer group of your Event Hub.
location	<i>Central US</i>	The location of the data connection resource.

Clean up resources

To delete the data connection, use the following command:

```
kustoManagementClient.DataConnections.Delete(resourceGroupName, clusterName, databaseName,
dataConnectionName);
```

Create an Event Hub data connection for Azure Data Explorer by using Azure Resource Manager template

12/2/2019 • 2 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Azure Data Explorer offers ingestion (data loading) from Event Hubs, IoT Hubs, and blobs written to blob containers. In this article, you create an Event Hub data connection for Azure Data Explorer by using Azure Resource Manager template.

Prerequisites

- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.
- [Create a cluster and database](#)
- [Create a table and column mapping](#)
- [Create an event hub](#)

Azure Resource Manager template for adding an Event Hub data connection

The following example shows an Azure Resource Manager template for adding an Event Hub data connection. You can [edit and deploy the template in the Azure portal](#) by using the form.

```
{  
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",  
    "contentVersion": "1.0.0.0",  
    "parameters": {  
        "namespaces_eventhubsns_name": {  
            "type": "string",  
            "defaultValue": "eventhubns",  
            "metadata": {  
                "description": "Specifies the Event Hub Namespace name."  
            }  
        },  
        "EventHubs_eventhubdemo_name": {  
            "type": "string",  
            "defaultValue": "eventhubdemo",  
            "metadata": {  
                "description": "Specifies the Event Hub name."  
            }  
        },  
        "consumergroup_default_name": {  
            "type": "string",  
            "defaultValue": "$Default",  
            "metadata": {  
                "description": "Specifies the consumer group of the Event Hub."  
            }  
        },  
        "Clusters_kustocluster_name": {  
            "type": "string",  
            "defaultValue": "kustocluster",  
            "metadata": {  
                "description": "Specifies the name of the cluster"  
            }  
        },  
        "databases_kustodb_name": {  
            "type": "string",  
            "defaultValue": "kustodb",  
            "metadata": {  
                "description": "Specifies the name of the database"  
            }  
        }  
    }  
}
```

```

        "type": "string",
        "defaultValue": "kustodb",
        "metadata": {
            "description": "Specifies the name of the database"
        }
    },
    "tables_kustotable_name": {
        "type": "string",
        "defaultValue": "kustotable",
        "metadata": {
            "description": "Specifies the name of the table"
        }
    },
    "mapping_kustomapping_name": {
        "type": "string",
        "defaultValue": "kustomapping",
        "metadata": {
            "description": "Specifies the name of the mapping rule"
        }
    },
    "dataformat_type": {
        "type": "string",
        "defaultValue": "csv",
        "metadata": {
            "description": "Specifies the data format"
        }
    },
    "dataconnections_kustodc_name": {
        "type": "string",
        "defaultValue": "kustodc",
        "metadata": {
            "description": "Name of the data connection to create"
        }
    },
    "subscriptionId": {
        "type": "string",
        "defaultValue": "[subscription().subscriptionId]",
        "metadata": {
            "description": "Specifies the subscriptionId of the Event Hub"
        }
    },
    "resourceGroup": {
        "type": "string",
        "defaultValue": "[resourceGroup().name]",
        "metadata": {
            "description": "Specifies the resourceGroup of the Event Hub"
        }
    },
    "location": {
        "type": "string",
        "defaultValue": "[resourceGroup().location]",
        "metadata": {
            "description": "Location for all resources."
        }
    },
    "variables": {
    },
    "resources": [
        {
            "type": "Microsoft.Kusto/Clusters/Databases/DataConnections",
            "apiVersion": "2019-09-07",
            "name": "[concat(parameters('Clusters_kustocluster_name'), '/',
parameters('databases_kustodb_name'), '/', parameters('dataconnections_kustodc_name'))]",
            "location": "[parameters('location')]",
            "kind": "EventHub",
            "properties": {
                "eventHubResourceId": "[resourceId(parameters('subscriptionId'), parameters('resourceGroup'),
'Microsoft.EventHub/namespaces/eventhubs', parameters('namespaces_eventhubns_name'),
parameters('EventHubs_eventhubdemo_name'))]"
            }
        }
    ]
}

```

```
        "consumerGroup": "[parameters('consumergroup_default_name')]",
        "tableName": "[parameters('tables_kustotable_name')]",
        "mappingRuleName": "[parameters('mapping_kustomapping_name')]",
        "dataFormat": "[parameters('dataformat_type')]"
    }
}
]
}
```

Clean up resources

When the Azure resources are no longer needed, clean up the resources you deployed by deleting the resource group.

Clean up resources using the Azure portal

Delete the resources in the Azure portal by following the steps in [clean up resources](#).

Clean up resources using PowerShell

If the Cloud Shell is still open, you don't need to copy/run the first line (Read-Host).

```
$projectName = Read-Host -Prompt "Enter the same project name that you used in the last procedure"
$resourceGroupName = "${projectName}rg"

Remove-AzResourceGroup -ResourceGroupName $resourceGroupName

Write-Host "Press [ENTER] to continue ..."
```

Ingest blobs into Azure Data Explorer by subscribing to Event Grid notifications

12/2/2019 • 5 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and scalable data exploration service for log and telemetry data. It offers continuous ingestion (data loading) from blobs written to blob containers.

In this article, you learn how to set an [Azure Event Grid](#) subscription, and route events to Azure Data Explorer via an event hub. To begin, you should have a storage account with an event grid subscription that sends notifications to Azure Event Hubs. Then you'll create an Event Grid data connection and see the data flow throughout the system.

Prerequisites

- An Azure subscription. Create a [free Azure account](#).
- [A cluster and database](#).
- [A storage account](#).
- [An event hub](#).

Create an Event Grid subscription in your storage account

1. In the Azure portal, find your storage account.
2. Select **Events > Event Subscription**.

The screenshot shows the 'Create Event Subscription' page for 'Event Grid'. The 'Basic' tab is selected. The 'EVENT SUBSCRIPTION DETAILS' section includes a 'Name' field containing 'test-grid-connection' and an 'Event Schema' dropdown set to 'Event Grid Schema'. The 'TOPIC DETAILS' section shows 'Topic Type' as 'Storage account' and 'Topic Resource' as 'gridteststorage'. The 'EVENT TYPES' section has a checkbox for 'Subscribe to all event types' and a dropdown for 'Defined Event Types' showing 'Blob Created'. The 'ENDPOINT DETAILS' section shows 'Endpoint Type' as 'Event Hubs (change)' and 'Endpoint' as 'test-hub (change)'. A note at the bottom says 'Pick an event handler to receive your events.'

3. In the **Create Event Subscription** window within the **Basic** tab, provide the following values:

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
Name	<i>test-grid-connection</i>	The name of the event grid that you want to create.
Event Schema	<i>Event Grid schema</i>	The schema that should be used for the event grid.
Topic Type	<i>Storage account</i>	The type of event grid topic.
Topic Resource	<i>gridteststorage</i>	The name of your storage account.
Subscribe to all event types	<i>clear</i>	Don't get notified on all events.
Defined Event Types	<i>Blob created</i>	Which specific events to get notified for.
Endpoint Type	<i>Event hubs</i>	The type of endpoint to which you send the events.
Endpoint	<i>test-hub</i>	The event hub you created.

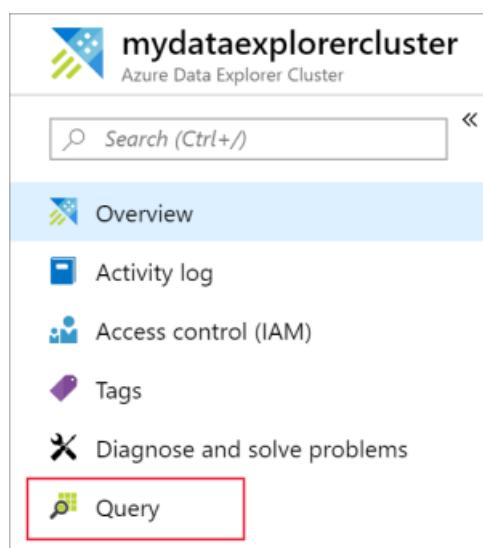
4. Select the **Filters** tab if you want to track files from a specific container. Set the filters for the notifications as follows:

- **Subject Begins With** field is the *literal* prefix of the blob container. As the pattern applied is *startswith*, it can span multiple containers. No wildcards are allowed. It *must* be set as follows:
`/blobServices/default/containers/ [container prefix]`
- **Subject Ends With** field is the *literal* suffix of the blob. No wildcards are allowed.

Create a target table in Azure Data Explorer

Create a table in Azure Data Explorer where Event Hubs will send data. Create the table in the cluster and database prepared in the prerequisites.

1. In the Azure portal, under your cluster, select **Query**.



- Copy the following command into the window and select **Run** to create the table (TestTable) that will receive the ingested data.

```
.create table TestTable (TimeStamp: datetime, Value: string, Source:string)
```

The screenshot shows the Azure Data Explorer interface. In the top navigation bar, there is a search bar and a favorites section. Below the navigation bar, the path is shown as kustodocs.WestUS/TestDatabase. The main area contains a code editor with the following SQL command:

```
1 .create table TestTable (TimeStamp: datetime, Value: string, Source:string)
2
3
4
5
```

Below the code editor, there is a table named "Table 1" with one row:

TableName	Schema	DatabaseName	Folder	DocString
TestTable	{"Name": "TestTable", "OrderedColumn...}	TestDatabase	null	null

- Copy the following command into the window and select **Run** to map the incoming JSON data to the column names and data types of the table (TestTable).

```
.create table TestTable ingestion json mapping 'TestMapping'
'[{"column":"TimeStamp","path":"$.TimeStamp"}, {"column":"Value","path":"$.Value"}, {"column":"Source","path":"$.Source"}]'
```

Create an Event Grid data connection in Azure Data Explorer

Now connect to the Event Grid from Azure Data Explorer, so that data flowing into the blob container is streamed to the test table.

- Select **Notifications** on the toolbar to verify that the event hub deployment was successful.
- Under the cluster you created, select **Databases > TestDatabase**.

The screenshot shows the Azure Data Explorer Databases blade for the cluster "mydataexplorercluster". On the left, there is a sidebar with various options: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Query, Settings (Scale up, Scale out, Properties, Locks, Automation script), and Data (Databases). The "Databases" option is highlighted with a red box. The main area displays a table of databases:

DATABASE	SIZE	RETENTION PERIOD	CACHE PERIOD
TestDatabase	0 Bytes	3650 days	31 days

- Select **Data ingestion > Add data connection**.



TestDatabase (mydataexplorercluster/TestDatabase) - Data in

Azure Data Explorer Database



Search (Ctrl+ /)

«



Add data connection



Refresh



Overview



Permissions



Query

Settings



Data ingestion



Properties



Search to filter items...



DATA CONNECTION ...

DATABASE

No results

4. Select the connection type: **Blob Storage**.
5. Fill out the form with the following information, and select **Create**.

Data connection

Select connection type
Blob storage

Data source

* Data connection name ⓘ
test-hub-connection ✓

* Storage account subscription
KUSTO_DEV_KUSTO_11

* Storage account
gridteststorage1

* Event Grid
test-grid-connection

Event Hub name ⓘ
test-hub

* Consumer group ⓘ
test-group

Target table

* Table ⓘ
TestTable ✓

* Data format
JSON

* Column mapping ⓘ
TestMapping ✓

Create

Data source:

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
Data connection name	<i>test-hub-connection</i>	The name of the connection that you want to create in Azure Data Explorer.
Storage account subscription	Your subscription ID	The subscription ID where your storage account resides.
Storage account	<i>gridteststorage</i>	The name of the storage account that you created previously.
Event Grid	<i>test-grid-connection</i>	The name of the event grid that you created.

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
Event Hub name	<i>test-hub</i>	The event hub that you created. This field is automatically filled when you pick an event grid.
Consumer group	<i>test-group</i>	The consumer group defined in the event hub that you created.

Target table:

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
Table	<i>TestTable</i>	The table you created in TestDatabase .
Data format	<i>JSON</i>	Supported formats are Avro, CSV, JSON, MULTILINE JSON, PSV, SOH, SCSV, TSV, and TXT. Supported compression options: Zip and GZip
Column mapping	<i>TestMapping</i>	The mapping you created in TestDatabase , which maps incoming JSON data to the column names and data types of TestTable .

Generate sample data

Now that Azure Data Explorer and the storage account are connected, you can create sample data and upload it to the blob storage.

We'll work with a small shell script that issues a few basic Azure CLI commands to interact with Azure Storage resources. This script creates a new container in your storage account, uploads an existing file (as a blob) to that container, and then lists the blobs in the container. You can use [Azure Cloud Shell](#) to execute the script directly in the portal.

Save the data into a file and upload it with this script:

```
{"TimeStamp": "1987-11-16 12:00", "Value": "Hello World", "Source": "TestSource"}
```

```

#!/bin/bash
### A simple Azure Storage example script

export AZURE_STORAGE_ACCOUNT=<storage_account_name>
export AZURE_STORAGE_KEY=<storage_account_key>

export container_name=<container_name>
export blob_name=<blob_name>
export file_to_upload=<file_to_upload>
export destination_file=<destination_file>

echo "Creating the container..."
az storage container create --name $container_name

echo "Uploading the file..."
az storage blob upload --container-name $container_name --file $file_to_upload --name $blob_name

echo "Listing the blobs..."
az storage blob list --container-name $container_name --output table

echo "Done"

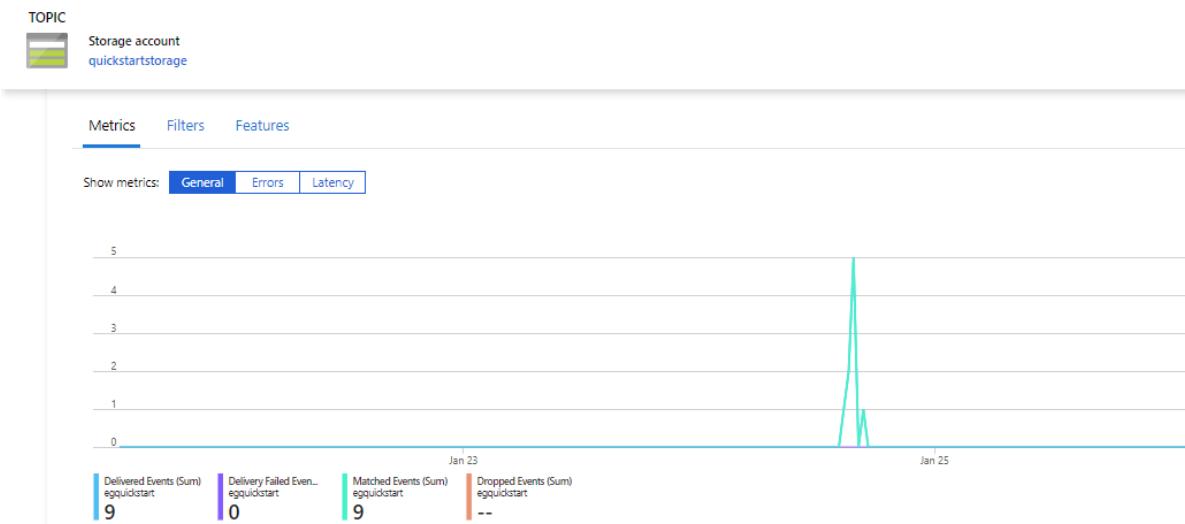
```

Review the data flow

NOTE

Azure Data Explorer has an aggregation (batching) policy for data ingestion designed to optimize the ingestion process. By default, the policy is configured to 5 minutes. You'll be able to alter the policy at a later time if needed. In this article you can expect a latency of a few minutes.

1. In the Azure portal, under your event grid, you see the spike in activity while the app is running.



2. To check how many messages have made it to the database so far, run the following query in your test database.

```

TestTable
| count

```

3. To see the content of the messages, run the following query in your test database.

```
TestTable
```

The result set should look like the following.

A screenshot of the Azure Data Explorer interface. At the top, there are navigation buttons: 'Run' and 'Recall'. Below them, a list shows two rows: '1 TestTable' and '2'. Underneath this, there are tabs: 'TestTable' (which is selected) and 'Stats'. A table below the tabs displays three columns: 'TimeStamp', 'Value', and 'Source'. The data row is: '1987-11-16 12:00:00.0000', 'Hello World', and 'TestSource'.

Clean up resources

If you don't plan to use your event grid again, clean up **test-hub-rg**, to avoid incurring costs.

1. In the Azure portal, select **Resource groups** on the far left, and then select the resource group you created.

If the left menu is collapsed, select to expand it.

A screenshot of the Azure portal's Resource groups page. On the left, a sidebar lists options: 'Create a resource', 'All services', 'FAVORITES' (with 'Dashboard' selected), 'Resource groups' (which is highlighted with a red box), 'All resources', 'Recent', and 'App Services'. The main area shows a table titled 'Resource groups' under the 'Microsoft' category. The table has columns: 'NAME' and 'SUBSCRIPTION'. One item is listed: 'test-hub-rg' (which is also highlighted with a red box). Above the table, there are filters: 'test-hub-rg' in the 'NAME' dropdown, '4 subscriptions' in the 'SUBSCRIPTIONS' dropdown, and 'All locations' in the 'LOCATION' dropdown.

2. Under **test-resource-group**, select **Delete resource group**.

3. In the new window, enter the name of the resource group to delete (*test-hub-rg*), and then select **Delete**.

Next steps

- [Query data in Azure Data Explorer](#)

Create an Event Grid data connection for Azure Data Explorer by using Python

12/2/2019 • 2 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Azure Data Explorer offers ingestion (data loading) from Event Hubs, IoT Hubs, and blobs written to blob containers. In this article, you create an Event Grid data connection for Azure Data Explorer by using Python.

Prerequisites

- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.
- [Create a cluster and database](#)
- [Create table and column mapping](#)
- [Set database and table policies](#) (optional)
- [Create a storage account with an Event Grid subscription](#).

Install Python package

To install the Python package for Azure Data Explorer (Kusto), open a command prompt that has Python in its path. Run the following command:

```
pip install azure-common  
pip install azure-mgmt-kusto
```

Authentication

To run the following example, you need an Azure Active Directory (Azure AD) application and service principal that can access resources. To create a free Azure AD application and add role assignment at the subscription level, see [Create an Azure AD application](#). You also need the directory (tenant) ID, application ID, and client secret.

Add an Event Grid data connection

The following example shows you how to add an Event Grid data connection programmatically. See [create an Event Grid data connection in Azure Data Explorer](#) for adding an Event Grid data connection using the Azure portal.

```

from azure.mgmt.kusto import KustoManagementClient
from azure.mgmt.kusto.models import EventGridDataConnection
from azure.common.credentials import ServicePrincipalCredentials

#Directory (tenant) ID
tenant_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
#Application ID
client_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
#Client Secret
client_secret = "xxxxxxxxxxxxxx"
subscription_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
credentials = ServicePrincipalCredentials(
    client_id=client_id,
    secret=client_secret,
    tenant=tenant_id
)
kusto_management_client = KustoManagementClient(credentials, subscription_id)

resource_group_name = "testrg";
#The cluster and database that are created as part of the Prerequisites
cluster_name = "mykustocluster";
database_name = "mykustodatabase";
data_connection_name = "myeventhubconnect";
#The event hub and storage account that are created as part of the Prerequisites
event_hub_resource_id = "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxx/resourceGroups/xxxxxx/providers/Microsoft.EventHub/namespaces/xxxxxx/eventhubs/xxxxxx"
storage_account_resource_id = "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxx/resourceGroups/xxxxxx/providers/Microsoft.Storage/storageAccounts/xxxxxx"
consumer_group = "$Default";
location = "Central US";
#The table and column mapping that are created as part of the Prerequisites
table_name = "StormEvents";
mapping_rule_name = "StormEvents_CSV_Mapping";
data_format = "csv";

#Returns an instance of LROPoller, check
https://docs.microsoft.com/python/api/msrest/msrest.polling.lropoller?view=azure-python
poller = kusto_management_client.data_connections.create_or_update(resource_group_name=resource_group_name,
cluster_name=cluster_name, database_name=database_name, data_connection_name=data_connection_name,
parameters=EventGridDataConnection(storage_account_resource_id=storage_account_resource_id,
event_hub_resource_id=event_hub_resource_id,
consumer_group=consumer_group,
table_name=table_name, location=location, mapping_rule_name=mapping_rule_name, data_format=data_format))

```

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
tenant_id	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx	Your tenant ID. Also known as directory ID.
subscription_id	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx	The subscription ID that you use for resource creation.
client_id	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx	The client ID of the application that can access resources in your tenant.
client_secret	xxxxxxxxxxxxxx	The client secret of the application that can access resources in your tenant.
resource_group_name	testrg	The name of the resource group containing your cluster.

Setting	Suggested Value	Field Description
cluster_name	<i>mykustocluster</i>	The name of your cluster.
database_name	<i>mykustodatabase</i>	The name of the target database in your cluster.
data_connection_name	<i>myeventhubconnect</i>	The desired name of your data connection.
table_name	<i>StormEvents</i>	The name of the target table in the target database.
mapping_rule_name	<i>StormEvents_CSVMapping</i>	The name of your column mapping related to the target table.
data_format	<i>csv</i>	The data format of the message.
event_hub_resource_id	<i>Resource ID</i>	The resource ID of your Event Hub where the Event Grid is configured to send events.
storage_account_resource_id	<i>Resource ID</i>	The resource ID of your storage account that holds the data for ingestion.
consumer_group	<i>\$Default</i>	The consumer group of your Event Hub.
location	<i>Central US</i>	The location of the data connection resource.

Clean up resources

To delete the data connection, use the following command:

```
kusto_management_client.data_connections.delete(resource_group_name=resource_group_name,
cluster_name=kusto_cluster_name, database_name=kusto_database_name,
data_connection_name=kusto_data_connection_name)
```

Create an Event Grid data connection for Azure Data Explorer by using C#

12/2/2019 • 2 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Azure Data Explorer offers ingestion (data loading) from Event Hubs, IoT Hubs, and blobs written to blob containers. In this article, you create an Event Grid data connection for Azure Data Explorer by using C#.

Prerequisites

- If you don't have Visual Studio 2019 installed, you can download and use the [free Visual Studio 2019 Community Edition](#). Make sure that you enable **Azure development** during the Visual Studio setup.
- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.
- [Create a cluster and database](#)
- [Create table and column mapping](#)
- [Set database and table policies](#) (optional)
- Create a [storage account with an Event Grid subscription](#).

Install C# nuget

- Install the [Azure Data Explorer \(Kusto\) NuGet package](#).
- Install the [Microsoft.IdentityModel.Clients.ActiveDirectory NuGet package](#) for authentication.

Authentication

To run the following example, you need an Azure Active Directory (Azure AD) application and service principal that can access resources. To create a free Azure AD application and add role assignment at the subscription level, see [Create an Azure AD application](#). You also need the directory (tenant) ID, application ID, and client secret.

Add an Event Grid data connection

The following example shows you how to add an Event Grid data connection programmatically. See [create an Event Grid data connection in Azure Data Explorer](#) for adding an Event Grid data connection using the Azure portal.

```

var tenantId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx";//Directory (tenant) ID
var clientId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx";//Application ID
var clientSecret = "xxxxxxxxxxxxxx";//Client Secret
var subscriptionId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx";
var authenticationContext = new AuthenticationContext($"https://login.windows.net/{tenantId}");
var credential = new ClientCredential(clientId, clientSecret);
var result = await authenticationContext.AcquireTokenAsync(resource: "https://management.core.windows.net/",
clientCredential: credential);

var credentials = new TokenCredentials(result.AccessToken, result.AccessTokenType);

var kustoManagementClient = new KustoManagementClient(credentials)
{
    SubscriptionId = subscriptionId
};

var resourceGroupName = "testrg";
//The cluster and database that are created as part of the Prerequisites
var clusterName = "mykustocluster";
var databaseName = "mykustodatabase";
var dataConnectionName = "myeventhubconnect";
//The event hub and storage account that are created as part of the Prerequisites
var eventHubResourceId = "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxx/resourceGroups/xxxxxx/providers/Microsoft.EventHub/namespaces/xxxxxx/eventhubs/xxxxxx";
var storageAccountResourceId = "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxx-
xxxxxxxx/resourceGroups/xxxxxx/providers/Microsoft.Storage/storageAccounts/xxxxxx";
var consumerGroup = "$Default";
var location = "Central US";
//The table and column mapping are created as part of the Prerequisites
var tableName = "StormEvents";
var mappingRuleName = "StormEvents_CSVM_Mapping";
var dataFormat = DataFormat.CSV;

await kustoManagementClient.DataConnections.CreateOrUpdateAsync(resourceGroupName, clusterName, databaseName,
dataConnectionName,
new EventGridDataConnection(storageAccountResourceId, eventHubResourceId, consumerGroup, tableName:
tableName, location: location, mappingRuleName: mappingRuleName, dataFormat: dataFormat));

```

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
tenantId	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx	Your tenant ID. Also known as directory ID.
subscriptionId	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx	The subscription ID that you use for resource creation.
clientId	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx	The client ID of the application that can access resources in your tenant.
clientSecret	xxxxxxxxxxxxxx	The client secret of the application that can access resources in your tenant.
resourceGroupName	testrg	The name of the resource group containing your cluster.
clusterName	mykustocluster	The name of your cluster.
databaseName	mykustodatabase	The name of the target database in your cluster.

Setting	Suggested Value	Field Description
dataConnectionName	<i>myeventhubconnect</i>	The desired name of your data connection.
tableName	<i>StormEvents</i>	The name of the target table in the target database.
mappingRuleName	<i>StormEvents_CSVMapping</i>	The name of your column mapping related to the target table.
dataFormat	<i>csv</i>	The data format of the message.
eventHubResourceId	<i>Resource ID</i>	The resource ID of your Event Hub where the Event Grid is configured to send events.
storageAccountResourceId	<i>Resource ID</i>	The resource ID of your storage account that holds the data for ingestion.
consumerGroup	<i>\$Default</i>	The consumer group of your Event Hub.
location	<i>Central US</i>	The location of the data connection resource.

Clean up resources

To delete the data connection, use the following command:

```
kustoManagementClient.DataConnections.Delete(resourceGroupName, clusterName, databaseName,
dataConnectionName);
```

Create an Event Grid data connection for Azure Data Explorer by using Azure Resource Manager template

12/2/2019 • 2 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Azure Data Explorer offers ingestion (data loading) from Event Hubs, IoT Hubs, and blobs written to blob containers. In this article, you create an Event Grid data connection for Azure Data Explorer by using Azure Resource Manager template.

Prerequisites

- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.
- [Create a cluster and database](#)
- [Create a table and column mapping](#)
- [Create an event hub](#)
- [Create a storage account with an Event Grid subscription.](#)

Azure Resource Manager template for adding an Event Grid data connection

The following example shows an Azure Resource Manager template for adding an Event Grid data connection. You can [edit and deploy the template in the Azure portal](#) by using the form.

```
{  
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",  
    "contentVersion": "1.0.0.0",  
    "parameters": {  
        "namespaces_eventhubns_name": {  
            "type": "string",  
            "defaultValue": "eventhubns",  
            "metadata": {  
                "description": "Specifies the Event Hub Namespace name."  
            }  
        },  
        "EventHubs_eventhubdemo_name": {  
            "type": "string",  
            "defaultValue": "eventhubdemo",  
            "metadata": {  
                "description": "Specifies the Event Hub name."  
            }  
        },  
        "consumergroup_default_name": {  
            "type": "string",  
            "defaultValue": "$Default",  
            "metadata": {  
                "description": "Specifies the consumer group of the Event Hub."  
            }  
        },  
        "StorageAccounts_storagedemo_name": {  
            "type": "string",  
            "defaultValue": "storagedemo",  
            "metadata": {  
                "description": "Specifies the storage account name"  
            }  
        },  
    },  
    "resources": [  
        {  
            "type": "Microsoft.EventGrid/eventHubTopics",  
            "name": "[parameters('EventHubs_eventhubdemo_name')]",  
            "apiVersion": "2018-01-01",  
            "location": "West US",  
            "properties": {  
                "topicName": "[parameters('EventHubs_eventhubdemo_name')]",  
                "topicType": "EventHub",  
                "topicProperties": {  
                    "topicLockPolicy": "None",  
                    "topicRetentionTime": "P1D",  
                    "topicTtl": "P1D",  
                    "topicThroughput": 1  
                },  
                "topicMetadata": {  
                    "topicConsumerGroup": "[parameters('consumergroup_default_name')]"  
                }  
            }  
        },  
        {  
            "type": "Microsoft.DataExplorer/accounts",  
            "name": "[parameters('namespaces_eventhubns_name')]",  
            "apiVersion": "2018-01-01",  
            "location": "West US",  
            "properties": {  
                "accountName": "[parameters('namespaces_eventhubns_name')]",  
                "accountType": "Standard",  
                "accountProperties": {  
                    "accountTtl": "P1D",  
                    "accountThroughput": 1  
                }  
            }  
        },  
        {  
            "type": "Microsoft.DataExplorer/accounts/providers",  
            "name": "[concat(parameters('namespaces_eventhubns_name'), '/EventHub')]",  
            "apiVersion": "2018-01-01",  
            "location": "West US",  
            "dependsOn": ["/subscriptions/  
                "Microsoft.DataExplorer/accounts/  
                    "[parameters('namespaces_eventhubns_name')]"  
            ]  
        },  
        {  
            "type": "Microsoft.DataExplorer/accounts/providers/partitions",  
            "name": "[concat(parameters('namespaces_eventhubns_name'), '/EventHub', '/EventHub')]",  
            "apiVersion": "2018-01-01",  
            "location": "West US",  
            "dependsOn": ["/subscriptions/  
                "Microsoft.DataExplorer/accounts/  
                    "[parameters('namespaces_eventhubns_name')]"  
            ],  
            "properties": {  
                "partitionName": "[parameters('EventHubs_eventhubdemo_name')]"  
            }  
        },  
        {  
            "type": "Microsoft.DataExplorer/accounts/providers/partitions/partitionEvents",  
            "name": "[concat(parameters('namespaces_eventhubns_name'), '/EventHub', '/EventHub', '/EventHub')]",  
            "apiVersion": "2018-01-01",  
            "location": "West US",  
            "dependsOn": ["/subscriptions/  
                "Microsoft.DataExplorer/accounts/  
                    "[parameters('namespaces_eventhubns_name')]"  
            ],  
            "properties": {  
                "partitionName": "[parameters('EventHubs_eventhubdemo_name')]",  
                "partitionEventName": "[parameters('consumergroup_default_name')]"  
            }  
        },  
        {  
            "type": "Microsoft.DataExplorer/accounts/providers/partitions/partitionEvents/partitionEventTopics",  
            "name": "[concat(parameters('namespaces_eventhubns_name'), '/EventHub', '/EventHub', '/EventHub', '/EventHub')]",  
            "apiVersion": "2018-01-01",  
            "location": "West US",  
            "dependsOn": ["/subscriptions/  
                "Microsoft.DataExplorer/accounts/  
                    "[parameters('namespaces_eventhubns_name')]"  
            ],  
            "properties": {  
                "partitionName": "[parameters('EventHubs_eventhubdemo_name')]",  
                "partitionEventName": "[parameters('consumergroup_default_name')]",  
                "topicName": "[parameters('EventHubs_eventhubdemo_name')]"  
            }  
        }  
    ]  
}
```

```
"Clusters_kustocluster_name": {
    "type": "string",
    "defaultValue": "kustocluster",
    "metadata": {
        "description": "Specifies the name of the cluster"
    }
},
"databases_kustedb_name": {
    "type": "string",
    "defaultValue": "kustedb",
    "metadata": {
        "description": "Specifies the name of the database"
    }
},
"tables_kustotable_name": {
    "type": "string",
    "defaultValue": "kustotable",
    "metadata": {
        "description": "Specifies the name of the table"
    }
},
"mapping_kustomapping_name": {
    "type": "string",
    "defaultValue": "kustomapping",
    "metadata": {
        "description": "Specifies the name of the mapping rule"
    }
},
"dataformat_type": {
    "type": "string",
    "defaultValue": "csv",
    "metadata": {
        "description": "Specifies the data format"
    }
},
"dataconnections_kustodc_name": {
    "type": "string",
    "defaultValue": "kustodc",
    "metadata": {
        "description": "Name of the data connection to create"
    }
},
"subscriptionId": {
    "type": "string",
    "defaultValue": "[subscription().subscriptionId]",
    "metadata": {
        "description": "Specifies the subscriptionId of the resources"
    }
},
"resourceGroup": {
    "type": "string",
    "defaultValue": "[resourceGroup().name]",
    "metadata": {
        "description": "Specifies the resourceGroup of the resources"
    }
},
"location": {
    "type": "string",
    "defaultValue": "[resourceGroup().location]",
    "metadata": {
        "description": "Location for all resources."
    }
},
"variables": {},
"resources": [
{
    "type": "Microsoft.Kusto/Clusters/Databases/DataConnections",
    "apiVersion": "2019-09-07",
    "name": "kustocluster1",
    "clusterName": "kustocluster1",
    "databaseName": "kustedb1",
    "tableName": "kustotable1",
    "mappingName": "kustomapping1",
    "dataFormat": "csv",
    "dataConnectionString": "kustodc1",
    "subscriptionId": "00000000-0000-0000-0000-000000000000",
    "resourceGroup": "kustouser1",
    "location": "East US"
}
]
```

```
        "name": "[concat(parameters('Clusters_kustocluster_name'), '/', parameters('databases_kustodb_name'), '/', parameters('dataconnections_kustodc_name'))]",  
        "location": "[parameters('location')]",  
        "kind": "EventGrid",  
        "properties": {  
            "storageAccountResourceId": "[resourceId(parameters('subscriptionId'), parameters('resourceGroup'), 'Microsoft.Storage/storageAccounts', parameters('StorageAccounts_storagedemo_name'))]",  
            "eventHubResourceId": "[resourceId(parameters('subscriptionId'), parameters('resourceGroup'), 'Microsoft.EventHub/namespaces/eventhubs', parameters('namespaces_eventhubsns_name'), parameters('EventHubs_eventhubdemo_name'))]",  
            "consumerGroup": "[parameters('consumergroup_default_name')]",  
            "tableName": "[parameters('tables_kustotable_name')]",  
            "mappingRuleName": "[parameters('mapping_kustomapping_name')]",  
            "dataFormat": "[parameters('dataformat_type')]"  
        }  
    }  
}  
]
```

Clean up resources

When the Azure resources are no longer needed, clean up the resources you deployed by deleting the resource group.

Clean up resources using the Azure portal

Delete the resources in the Azure portal by following the steps in [clean up resources](#).

Clean up resources using PowerShell

If the Cloud Shell is still open, you don't need to copy/run the first line (Read-Host).

```
$projectName = Read-Host -Prompt "Enter the same project name that you used in the last procedure"  
$resourceGroupName = "${projectName}rg"  
  
Remove-AzResourceGroup -ResourceGroupName $resourceGroupName  
  
Write-Host "Press [ENTER] to continue ..."
```

Ingest data from IoT Hub into Azure Data Explorer (Preview)

12/2/2019 • 7 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Azure Data Explorer offers ingestion (data loading) from IoT Hub, a big data streaming platform and IoT ingestion service.

Prerequisites

- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.
- Create [a test cluster and database](#) with database name *testdb*.
- [A sample app](#) and documentation for simulating a device.
- [Visual Studio 2019](#) to run the sample app.

Create an IoT Hub

This section describes how to create an IoT hub using the [Azure portal](#).

1. Sign in to the [Azure portal](#).
2. From the Azure homepage, select the **+ Create a resource** button, and then enter *IoT Hub* in the **Search the Marketplace** field.
3. Select **IoT Hub** from the search results, and then select **Create**.
4. On the **Basics** tab, complete the fields as follows:
 - **Subscription:** Select the subscription to use for your hub.
 - **Resource Group:** Select a resource group or create a new one. To create a new one, select **Create new** and fill in the name you want to use. To use an existing resource group, select that resource group. For more information, see [Manage Azure Resource Manager resource groups](#).
 - **Region:** Select the region in which you want your hub to be located. Select the location closest to you.
 - **IoT Hub Name:** Enter a name for your hub. This name must be globally unique. If the name you enter is available, a green check mark appears.

IMPORTANT

Because the IoT hub will be publicly discoverable as a DNS endpoint, be sure to avoid entering any sensitive or personally identifiable information when you name it.

The screenshot shows the 'Basics' step of the IoT hub creation wizard. It includes fields for Subscription (Contoso Subscription), Resource Group ((New) IoTHubResourceGroup), Region (West US), and IoT Hub Name (iot-hub-contoso-one). The 'Next: Size and scale' button is highlighted.

5. Select **Next: Size and scale** to continue creating your hub.

The screenshot shows the 'Size and scale' step of the IoT hub creation wizard. It includes a dropdown for Pricing and scale tier (S1: Standard tier), a slider for Number of S1 IoT Hub units (set to 1), and a list of features: Device-to-cloud-messages, Message routing, Cloud-to-device commands, IoT Edge, and Device management, all marked as Enabled. The 'Advanced Settings' section is expanded, showing a slider for Device-to-cloud partitions set to 4. Navigation buttons at the bottom include 'Review + create', '< Previous: Basics', and 'Automation options'.

This screen allows you to set the following values:

- **Pricing and scale tier:** Your selected tier. You can choose from several tiers, depending on how many features you want and how many messages you send through your solution per day. The free

tier is intended for testing and evaluation. It allows 500 devices to be connected to the hub and up to 8,000 messages per day. Each Azure subscription can create one IoT hub in the free tier.

- **IoT Hub units:** The number of messages allowed per unit per day depends on your hub's pricing tier. For example, if you want the hub to support ingress of 700,000 messages, you choose two S1 tier units. For details about the other tier options, see [Choosing the right IoT Hub tier](#).
 - **Advanced Settings > Device-to-cloud partitions:** This property relates the device-to-cloud messages to the number of simultaneous readers of the messages. Most hubs need only four partitions.
6. For this article, accept the default choices, and then select **Review + create** to review your choices. You see something similar to this screen.

The screenshot shows the 'Review + create' step in the Azure portal for creating an IoT hub. The top navigation bar includes 'Home', 'New', 'IoT Hub', and 'IoT hub'. The main title is 'IoT hub' under 'Microsoft'. Below the title, there are two tabs: 'Basics' and 'Size and scale', with 'Size and scale' currently selected. A red box highlights the 'Review + create' button, which is also selected. The 'Basics' section contains fields for Subscription (Contoso Subscription), Resource Group (IoTHubResourceGroup), Region (West US), and IoT Hub Name (iot-hub-contoso-one). The 'Size and Scale' section shows the selected tier (S1), number of units (1), messages per day (400,000), and cost per month (<price>). At the bottom, a red box highlights the 'Create' button, and the status message '« Previous: Size and scale' is visible.

7. Select **Create** to create your new hub. Creating the hub takes a few minutes.

Register a device to the IoT Hub

In this section, you use the Azure CLI to create a device identity for this article. Device IDs are case sensitive.

1. Open [Azure Cloud Shell](#).
2. In Azure Cloud Shell, run the following command to install the Microsoft Azure IoT Extension for Azure CLI:

```
az extension add --name azure-cli-iot-ext
```

3. Create a new device identity called `myDeviceId` and retrieve the device connection string with these commands:

```
az iot hub device-identity create --device-id myDeviceId --hub-name {Your IoT Hub name}
az iot hub device-identity show-connection-string --device-id myDeviceId --hub-name {Your IoT Hub name}
-o table
```

IMPORTANT

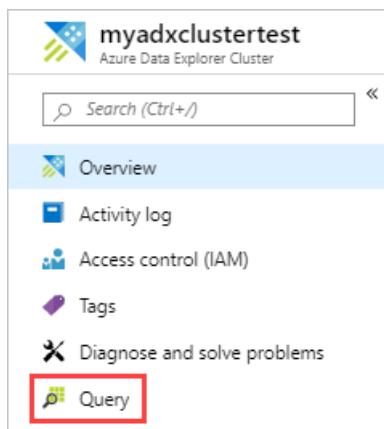
The device ID may be visible in the logs collected for customer support and troubleshooting, so make sure to avoid any sensitive information while naming it.

Make a note of the device connection string from the result. This device connection string is used by the device app to connect to your IoT Hub as a device.

Create a target table in Azure Data Explorer

Now you create a table in Azure Data Explorer to which IoT Hubs will send data. You create the table in the cluster and database provisioned in [Prerequisites](#).

1. In the Azure portal, navigate to your cluster and select **Query**.



2. Copy the following command into the window and select **Run** to create the table (TestTable) which will receive the ingested data.

```
.create table TestTable (temperature: real, humidity: real)
```

A screenshot of the Azure Data Explorer 'Query' editor. The command '.create table TestTable (temperature: real, humidity: real)' is entered. The 'Run' button is highlighted with a red box. The results show a single record for 'TestTable'.

3. Copy the following command into the window and select **Run** to map the incoming JSON data to the column names and data types of the table (TestTable).

```
.create table TestTable ingestion json mapping 'TestMapping'  
'[{"column":"humidity","path":("$.humidity","datatype":"real")},  
 {"column":"temperature","path":("$.temperature","datatype":"real")}]'
```

Connect Azure Data Explorer table to IoT hub

Now you connect to the IoT Hub from Azure Data Explorer. When this connection is complete, data that flows into the iot hub streams to the [target table you created](#).

1. Select **Notifications** on the toolbar to verify that the IoT Hub deployment was successful.
2. Under the cluster you created, select **Databases** then select the database that you created **testdb**.

The screenshot shows the Azure Data Explorer Cluster - PREVIEW interface. The left sidebar has a navigation menu with the following items:

- Home > myadxclustertest - Databases
- myadxclustertest - Databases (selected)
- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Query
- Settings
 - Scale up
 - Scale out
 - Security
 - Properties
 - Locks
 - Export template
- Data
 - Databases (selected)
 - Monitoring
 - Alerts
 - Metrics
 - Diagnostic settings
 - Support + troubleshooting
 - New support request

The main content area displays the "Databases" table with one row:

SEARCH TO FILTER ITEMS...	DATABASE	SIZE	RETENTION PERIOD	CACHE PERIOD
	testdb	5.67 KB	3650 days	31 days

3. Select **Data ingestion** and **Add data connection**. Then fill out the form with the following information.
Select **Create** when you're finished.

The screenshot shows the Azure Data Explorer interface for a database named 'testdb'. On the left, there's a sidebar with 'Overview', 'Permissions', 'Query', 'Settings' (which is expanded), and 'Data ingestion' (which is selected and highlighted with a red box). The main area shows a search bar and a list of data connections with the message 'No results'. A modal window titled 'Data connection' is open on the right, containing fields for 'Data connection name' (set to 'iotHub'), 'Subscription' (set to 'kustoiothub'), 'Shared Access Policy' (set to 'iothubowner'), 'Consumer group' (set to '\$Default'), 'Event system properties' (set to '0 selected'), 'Target table' (set to 'TestTable'), 'Data format' (set to 'JSON'), and 'Column mapping' (set to 'TestMapping'). A note at the bottom of the modal states: 'This is a preview version of IoT Hub data connection. This version does not support IoT Hub manual-failover. In such cases, please recreate the data connection.' The 'Create' button at the bottom right of the modal is also highlighted with a red box.

Data Source:

SETTING	FIELD DESCRIPTION
Data connection name	The name of the connection you want to create in Azure Data Explorer
IoT Hub	IoT Hub name
Shared access policy	The name of the shared access policy. Must have read permissions
Consumer group	The consumer group defined in the IoT Hub built-in endpoint
Event system properties	The IoT Hub event system properties . When adding system properties, create or update table schema and mapping to include the selected properties.

NOTE

In case of a [manual failover](#), you must recreate the data connection.

Target table:

There are two options for routing the ingested data: *static* and *dynamic*. For this article, you use static routing, where you specify the table name, data format, and mapping. Therefore, leave **My data includes routing info** unselected.

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
Table	<i>TestTable</i>	The table you created in testdb .
Data format	JSON	Supported formats are Avro, CSV, JSON, MULTILINE JSON, PSV, SOHSV, SCSV, TSV, TSVE, and TXT.
Column mapping	<i>TestMapping</i>	The mapping you created in testdb , which maps incoming JSON data to the column names and data types of testdb . Required for JSON, MULTILINE JSON, and AVRO, and optional for other formats.

NOTE

- Select **My data includes routing info** to use dynamic routing, where your data includes the necessary routing information as seen in the [sample app](#) comments. If both static and dynamic properties are set, the dynamic properties override the static ones.
- Only events enqueued after you create the data connection are ingested.

Generate sample data for testing

The simulated device application connects to a device-specific endpoint on your IoT hub and sends simulated temperature and humidity telemetry.

1. Download the sample C# project from <https://github.com/Azure-Samples/azure-iot-samples-csharp/archive/master.zip> and extract the ZIP archive.
2. In a local terminal window, navigate to the root folder of the sample C# project. Then navigate to the **iot-hub\Quickstarts\simulated-device** folder.
3. Open the **SimulatedDevice.cs** file in a text editor of your choice.

Replace the value of the `sConnectionString` variable with the device connection string from [Register a device to the IoT Hub](#). Then save your changes to **SimulatedDevice.cs** file.

4. In the local terminal window, run the following commands to install the required packages for simulated device application:

```
dotnet restore
```

5. In the local terminal window, run the following command to build and run the simulated device application:

```
dotnet run
```

The following screenshot shows the output as the simulated device application sends telemetry to your IoT hub:

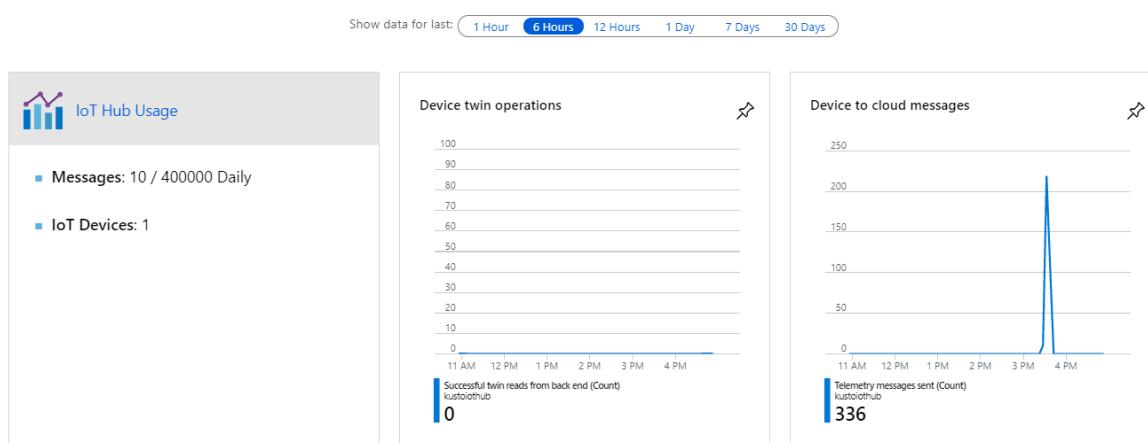
```
c:\repos\iot-hub-quickstarts-dotnet\simulated-device>dotnet run
IoT Hub Quickstarts #1 - Simulated device. Ctrl-C to exit.

04/04/2018 11:15:22 > Sending message: {"temperature":31.02260307922149,"humidity":66.922409537677851}
04/04/2018 11:15:23 > Sending message: {"temperature":25.782883018619792,"humidity":64.358296908605055}
04/04/2018 11:15:24 > Sending message: {"temperature":23.521925824937377,"humidity":63.064605203952922}
04/04/2018 11:15:25 > Sending message: {"temperature":29.148115233587156,"humidity":60.096141295552322}
04/04/2018 11:15:27 > Sending message: {"temperature":23.686750376916841,"humidity":61.101306500426169}
04/04/2018 11:15:28 > Sending message: {"temperature":31.85187338239135,"humidity":77.934687918952989}
04/04/2018 11:15:29 > Sending message: {"temperature":33.109382886024832,"humidity":62.153963494186272}
```

Review the data flow

With the app generating data, you can now see the data flow from the IoT hub to the table in your cluster.

1. In the Azure portal, under your IoT hub, you see the spike in activity while the app is running.



2. To check how many messages have made it to the database so far, run the following query in your test database.

```
TestTable
| count
```

3. To see the content of the messages, run the following query:

TestTable

The result set:

temperature	humidity
> 26.625	78.492
> 31.046	79.981
> 34.439	71.633
> 29.111	60.517
> 26.751	62.035
> 30.574	75.827
> 33.081	63.99
> 21.92	76.266
> 27.066	65.374
> 26.904	64.302

NOTE

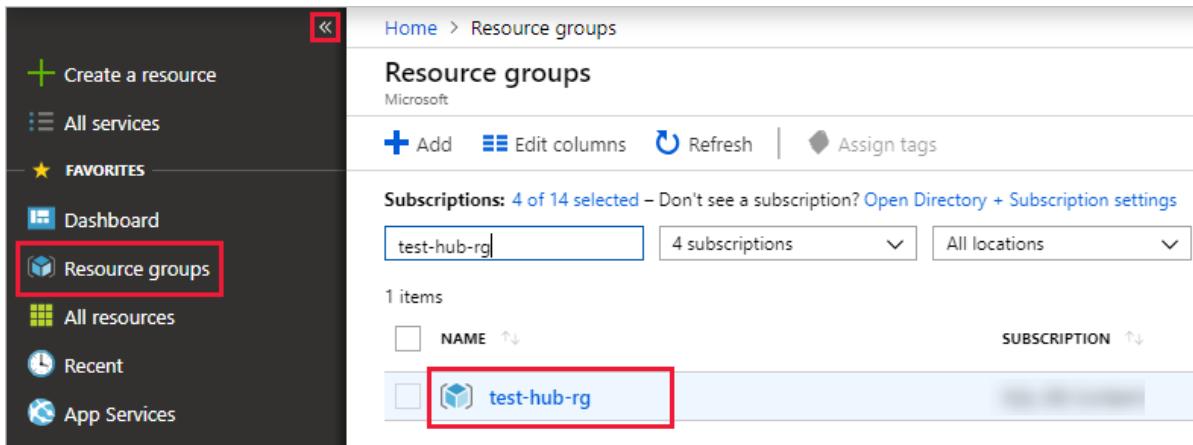
- Azure Data Explorer has an aggregation (batching) policy for data ingestion, designed to optimize the ingestion process. The policy is configured to 5 minutes or 500 MB of data, by default, so you may experience a latency. See [batching policy](#) for aggregation options.
- Configure your table to support streaming and remove the lag in response time. See [streaming policy](#).

Clean up resources

If you don't plan to use your IoT Hub again, clean up **test-hub-rg**, to avoid incurring costs.

1. In the Azure portal, select **Resource groups** on the far left, and then select the resource group you created.

If the left menu is collapsed, select  to expand it.



The screenshot shows the Azure portal's Resource groups blade. On the left, a sidebar lists 'Create a resource', 'All services', 'FAVORITES' (with 'test-hub-rg' listed), 'Dashboard', 'Resource groups' (which is highlighted with a red box), 'All resources', 'Recent', and 'App Services'. The main area shows 'Resource groups' under 'Microsoft'. It includes buttons for 'Add', 'Edit columns', 'Refresh', and 'Assign tags'. A search bar shows 'test-hub-rg'. Below it, there are filters for 'Subscriptions' (4 of 14 selected), '4 subscriptions', and 'All locations'. A table lists one item: 'test-hub-rg', which is also highlighted with a red box. The table has columns for 'NAME' and 'SUBSCRIPTION'.

2. Under **test-resource-group**, select **Delete resource group**.
3. In the new window, type the name of the resource group to delete (*test-hub-rg*), and then select **Delete**.

Next steps

- Query data in Azure Data Explorer

Create an IoT Hub data connection for Azure Data Explorer by using Python (Preview)

12/2/2019 • 2 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Azure Data Explorer offers ingestion (data loading) from Event Hubs, IoT Hubs, and blobs written to blob containers. In this article, you create an IoT Hub data connection for Azure Data Explorer by using Python.

Prerequisites

- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.
- [Create a cluster and database](#)
- [Create table and column mapping](#)
- [Set database and table policies](#) (optional)
- [Create an IoT Hub with a shared access policy configured.](#)

Install Python package

To install the Python package for Azure Data Explorer (Kusto), open a command prompt that has Python in its path. Run the following command:

```
pip install azure-common  
pip install azure-mgmt-kusto
```

Authentication

To run the following example, you need an Azure Active Directory (Azure AD) application and service principal that can access resources. To create a free Azure AD application and add role assignment at the subscription level, see [Create an Azure AD application](#). You also need the directory (tenant) ID, application ID, and client secret.

Add an IoT Hub data connection

The following example shows you how to add an IoT Hub data connection programmatically. See [connect Azure Data Explorer table to IoT Hub](#) for adding an IoT Hub data connection using the Azure portal.

```

from azure.mgmt.kusto import KustoManagementClient
from azure.mgmt.kusto.models import IoTHubDataConnection
from azure.common.credentials import ServicePrincipalCredentials

#Directory (tenant) ID
tenant_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
#Application ID
client_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
#Client Secret
client_secret = "xxxxxxxxxxxxxx"
subscription_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
credentials = ServicePrincipalCredentials(
    client_id=client_id,
    secret=client_secret,
    tenant=tenant_id
)
kusto_management_client = KustoManagementClient(credentials, subscription_id)

resource_group_name = "testrg";
#The cluster and database that are created as part of the Prerequisites
cluster_name = "mykustocluster";
database_name = "mykustodatabase";
data_connection_name = "myeventhubconnect";
#The IoT hub that is created as part of the Prerequisites
iot_hub_resource_id = "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxx/resourceGroups/xxxxxx/providers/Microsoft.Devices/IotHubs/xxxxxx";
shared_access_policy_name = "iothubforread";
consumer_group = "$Default";
location = "Central US";
#The table and column mapping that are created as part of the Prerequisites
table_name = "StormEvents";
mapping_rule_name = "StormEvents_CSV_Mapping";
data_format = "csv";

#Returns an instance of LROPoller, check
https://docs.microsoft.com/python/api/msrest/msrest.polling.lropoller?view=azure-python
poller = kusto_management_client.data_connections.create_or_update(resource_group_name=resource_group_name,
cluster_name=cluster_name, database_name=database_name, data_connection_name=data_connection_name,
parameters=IoTHubDataConnection(iot_hub_resource_id=iot_hub_resource_id,
shared_access_policy_name=shared_access_policy_name,
consumer_group=consumer_group,
table_name=table_name, location=location, mapping_rule_name=mapping_rule_name, data_format=data_format))

```

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
tenant_id	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx	Your tenant ID. Also known as directory ID.
subscriptionId	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx	The subscription ID that you use for resource creation.
client_id	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx	The client ID of the application that can access resources in your tenant.
client_secret	xxxxxxxxxxxxxx	The client secret of the application that can access resources in your tenant.
resource_group_name	testrg	The name of the resource group containing your cluster.
cluster_name	mykustocluster	The name of your cluster.

Setting	Suggested Value	Field Description
database_name	<i>mykustodatabase</i>	The name of the target database in your cluster.
data_connection_name	<i>myeventhubconnect</i>	The desired name of your data connection.
table_name	<i>StormEvents</i>	The name of the target table in the target database.
mapping_rule_name	<i>StormEvents_CSV_Mapping</i>	The name of your column mapping related to the target table.
data_format	<i>csv</i>	The data format of the message.
iot_hub_resource_id	<i>Resource ID</i>	The resource ID of your IoT hub that holds the data for ingestion.
shared_access_policy_name	<i>iothubforread</i>	The name of the shared access policy that defines the permissions for devices and services to connect to IoT Hub.
consumer_group	<i>\$Default</i>	The consumer group of your event hub.
location	<i>Central US</i>	The location of the data connection resource.

Clean up resources

To delete the data connection, use the following command:

```
kusto_management_client.data_connections.delete(resource_group_name=resource_group_name,
cluster_name=kusto_cluster_name, database_name=kusto_database_name,
data_connection_name=kusto_data_connection_name)
```

Create an IoT Hub data connection for Azure Data Explorer by using C# (Preview)

12/2/2019 • 2 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Azure Data Explorer offers ingestion (data loading) from Event Hubs, IoT Hubs, and blobs written to blob containers. In this article, you create an IoT Hub data connection for Azure Data Explorer by using C#.

Prerequisites

- If you don't have Visual Studio 2019 installed, you can download and use the [free Visual Studio 2019 Community Edition](#). Make sure that you enable **Azure development** during the Visual Studio setup.
- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.
- [Create a cluster and database](#)
- [Create table and column mapping](#)
- Set [database and table policies](#) (optional)
- Create an [IoT Hub with a shared access policy configured](#).

Install C# nuget

- Install the [Azure Data Explorer \(Kusto\) NuGet package](#).
- Install the [Microsoft.IdentityModel.Clients.ActiveDirectory NuGet package](#) for authentication.

Authentication

To run the following example, you need an Azure Active Directory (Azure AD) application and service principal that can access resources. To create a free Azure AD application and add role assignment at the subscription level, see [Create an Azure AD application](#). You also need the directory (tenant) ID, application ID, and client secret.

Add an IoT Hub data connection

The following example shows you how to add an IoT Hub data connection programmatically. See [connect Azure Data Explorer table to IoT Hub](#) for adding an IoT Hub data connection using the Azure portal.

```

var tenantId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx";//Directory (tenant) ID
var clientId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx";//Application ID
var clientSecret = "xxxxxxxxxxxxxx";//Client Secret
var subscriptionId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx";
var authenticationContext = new AuthenticationContext($"https://login.windows.net/{tenantId}");
var credential = new ClientCredential(clientId, clientSecret);
var result = await authenticationContext.AcquireTokenAsync(resource: "https://management.core.windows.net/", clientCredential: credential);

var credentials = new TokenCredentials(result.AccessToken, result.AccessTokenType);

var kustoManagementClient = new KustoManagementClient(credentials)
{
    SubscriptionId = subscriptionId
};

var resourceGroupName = "testrg";
//The cluster and database that are created as part of the Prerequisites
var clusterName = "mykustocluster";
var databaseName = "mykustodatabase";
var dataConnectionName = "myeventhubconnect";
//The IoT hub that is created as part of the Prerequisites
var iotHubResourceId = "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxx/resourceGroups/xxxxxx/providers/Microsoft.Devices/IotHubs/xxxxxx";
var sharedAccessPolicyName = "iothubforread";
var consumerGroup = "$Default";
var location = "Central US";
//The table and column mapping are created as part of the Prerequisites
var tableName = "StormEvents";
var mappingRuleName = "StormEvents_CSV_Mapping";
var dataFormat = DataFormat.CSV;

await kustoManagementClient.DataConnections.CreateOrUpdate(resourceGroupName, clusterName, databaseName, dataConnectionName,
    new IoTHubDataConnection(iotHubResourceId, consumerGroup, sharedAccessPolicyName, tableName: tableName, location: location, mappingRuleName: mappingRuleName, dataFormat: dataFormat));

```

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
tenantId	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx	Your tenant ID. Also known as directory ID.
subscriptionId	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx	The subscription ID that you use for resource creation.
clientId	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx	The client ID of the application that can access resources in your tenant.
clientSecret	xxxxxxxxxxxxxx	The client secret of the application that can access resources in your tenant.
resourceGroupName	testrg	The name of the resource group containing your cluster.
clusterName	mykustocluster	The name of your cluster.
databaseName	mykustodatabase	The name of the target database in your cluster.

Setting	Suggested Value	Field Description
dataConnectionName	<i>myeventhubconnect</i>	The desired name of your data connection.
tableName	<i>StormEvents</i>	The name of the target table in the target database.
mappingRuleName	<i>StormEvents_CSVMapping</i>	The name of your column mapping related to the target table.
dataFormat	<i>csv</i>	The data format of the message.
iotHubResourceId	<i>Resource ID</i>	The resource ID of your IoT hub that holds the data for ingestion.
sharedAccessPolicyName	<i>iothubforread</i>	The name of the shared access policy that defines the permissions for devices and services to connect to IoT Hub.
consumerGroup	<i>\$Default</i>	The consumer group of your event hub.
location	<i>Central US</i>	The location of the data connection resource.

Clean up resources

To delete the data connection, use the following command:

```
kustoManagementClient.DataConnections.Delete(resourceGroupName, clusterName, databaseName,
dataConnectionName);
```

Create an IoT Hub data connection for Azure Data Explorer by using Azure Resource Manager template

12/2/2019 • 2 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Azure Data Explorer offers ingestion (data loading) from Event Hubs, IoT Hubs, and blobs written to blob containers. In this article, you create an IoT Hub data connection for Azure Data Explorer by using Azure Resource Manager template.

Prerequisites

- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.
- [Create a cluster and database](#)
- [Create a table and column mapping](#)
- [Create an IoT Hub with a shared access policy configured.](#)

Azure Resource Manager template for adding an IoT Hub data connection

The following example shows an Azure Resource Manager template for adding an IoT Hub data connection. You can [edit and deploy the template in the Azure portal](#) by using the form.

```
{  
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",  
    "contentVersion": "1.0.0.0",  
    "parameters": {  
        "IoTHubs_iothubdemo_name": {  
            "type": "string",  
            "defaultValue": "iothubdemo",  
            "metadata": {  
                "description": "Specifies the IoT Hub name."  
            }  
        },  
        "iothubpolices_iothubowner_name": {  
            "type": "string",  
            "defaultValue": "iothubowner",  
            "metadata": {  
                "description": "Specifies the shared access policy name."  
            }  
        },  
        "consumergroup_default_name": {  
            "type": "string",  
            "defaultValue": "$Default",  
            "metadata": {  
                "description": "Specifies the consumer group of the IoT Hub."  
            }  
        },  
        "Clusters_kustocluster_name": {  
            "type": "string",  
            "defaultValue": "kustocluster",  
            "metadata": {  
                "description": "Specifies the name of the cluster"  
            }  
        },  
        "databases_kustodb_name": {  
            "type": "string",  
            "defaultValue": "kustodb",  
            "metadata": {  
                "description": "Specifies the name of the database"  
            }  
        }  
    }  
}
```

```

        "type": "string",
        "defaultValue": "kustodb",
        "metadata": {
            "description": "Specifies the name of the database"
        }
    },
    "tables_kustotable_name": {
        "type": "string",
        "defaultValue": "kustotable",
        "metadata": {
            "description": "Specifies the name of the table"
        }
    },
    "mapping_kustomapping_name": {
        "type": "string",
        "defaultValue": "kustomapping",
        "metadata": {
            "description": "Specifies the name of the mapping rule"
        }
    },
    "dataformat_type": {
        "type": "string",
        "defaultValue": "csv",
        "metadata": {
            "description": "Specifies the data format"
        }
    },
    "dataconnections_kustodc_name": {
        "type": "string",
        "defaultValue": "kustodc",
        "metadata": {
            "description": "Name of the data connection to create"
        }
    },
    "subscriptionId": {
        "type": "string",
        "defaultValue": "[subscription().subscriptionId]",
        "metadata": {
            "description": "Specifies the subscriptionId of the IoT Hub"
        }
    },
    "resourceGroup": {
        "type": "string",
        "defaultValue": "[resourceGroup().name]",
        "metadata": {
            "description": "Specifies the resourceGroup of the IoT Hub"
        }
    },
    "location": {
        "type": "string",
        "defaultValue": "[resourceGroup().location]",
        "metadata": {
            "description": "Location for all resources."
        }
    },
    "variables": {
    },
    "resources": [
        {
            "type": "Microsoft.Kusto/Clusters/Databases/DataConnections",
            "apiVersion": "2019-09-07",
            "name": "[concat(parameters('Clusters_kustocluster_name'), '/',
parameters('databases_kustodb_name'), '/', parameters('dataconnections_kustodc_name'))]",
            "location": "[parameters('location')]",
            "kind": "IoTHub",
            "properties": {
                "iotHubResourceId": "[resourceId(parameters('subscriptionId'), parameters('resourceGroup'),
'Microsoft.Devices/IotHubs', parameters('IotHubs_iothubdemo_name'))]",
                "consumerGroup": "[parameters('consumergroup_default_name')]"
            }
        }
    ]
}

```

```
        "sharedAccessPolicyName": "[parameters('iothubpolices_iothubowner_name')]",
        "tableName": "[parameters('tables_kustotable_name')]",
        "mappingRuleName": "[parameters('mapping_kustomapping_name')]",
        "dataFormat": "[parameters('dataformat_type')]"
    }
}
]
}
```

Clean up resources

When the Azure resources are no longer needed, clean up the resources you deployed by deleting the resource group.

Clean up resources using the Azure portal

Delete the resources in the Azure portal by following the steps in [clean up resources](#).

Clean up resources using PowerShell

If the Cloud Shell is still open, you don't need to copy/run the first line (Read-Host).

```
$projectName = Read-Host -Prompt "Enter the same project name that you used in the last procedure"
$resourceGroupName = "${projectName}rg"

Remove-AzResourceGroup -ResourceGroupName $resourceGroupName

Write-Host "Press [ENTER] to continue ..."
```

Use one-click ingestion to ingest data into Azure Data Explorer

12/10/2019 • 3 minutes to read • [Edit Online](#)

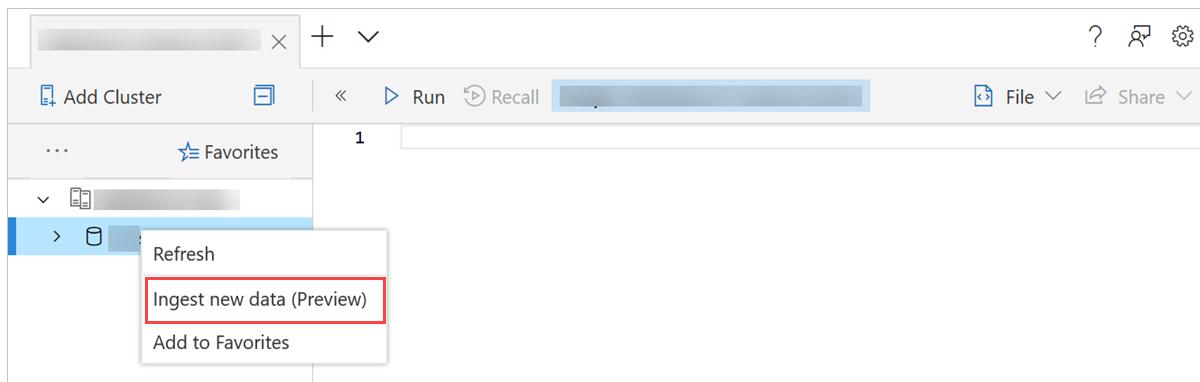
This article shows you how to use one-click ingestion for quick ingestion of a new table in either JSON or CSV format. The data can be ingested from storage or a local file into an existing table or a new table. Use the intuitive one-click wizard, and your data ingests within a few minutes. Then, you can edit the table and run queries using the Azure Data Explorer Web UI.

Prerequisites

- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.
- Sign in to [the application](#).
- Create [an Azure Data Explorer cluster and database](#).
- Sign in to the [Web UI](#) and [add a connection to your cluster](#).
- Create a source of data in Azure Storage.

Ingest new data

1. Right-click on the *database* or *table* row in left menu of the Web UI and select **Ingest new data (Preview)**.



2. In the **Ingest new data (Preview)** window, select the **Source** tab and complete the **Project Details**:

- For **Table**, select an existing table name from the drop-down menu or select **Create new** to make a new table.
- For **Ingestion type**, select either **from storage** or **from file**.
 - If you selected **from storage**, select **Link to storage** to add the URL. Use [Blob SAS URL](#) for private storage accounts.
 - If you selected **from file**, select **Browse** and drag the file into the box.
- Select **Edit schema** to view and edit your table column configuration.

Azure Data Explorer

Ingest new data (preview)

Ingest new data into table myadxclustertest.koreacentral > docstest > ADXdocs

1 Source **2 Schema**

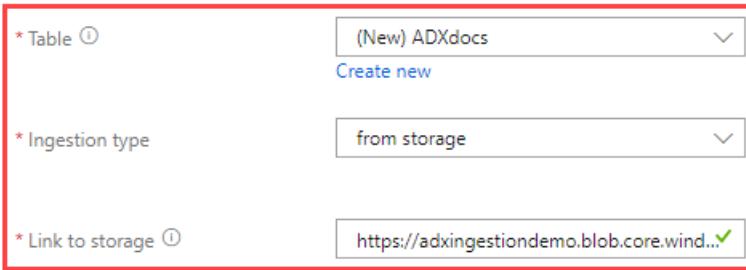
Project Details

* Table ① (New) ADXdocs
Create new

* Ingestion type from storage

* Link to storage ① https://adxitgestiondemo.blob.core.wind...

Previous **Edit schema**



TIP

If you select **Ingest new data (Preview)** on a *table* row, the selected table name will appear in the **Project Details**.

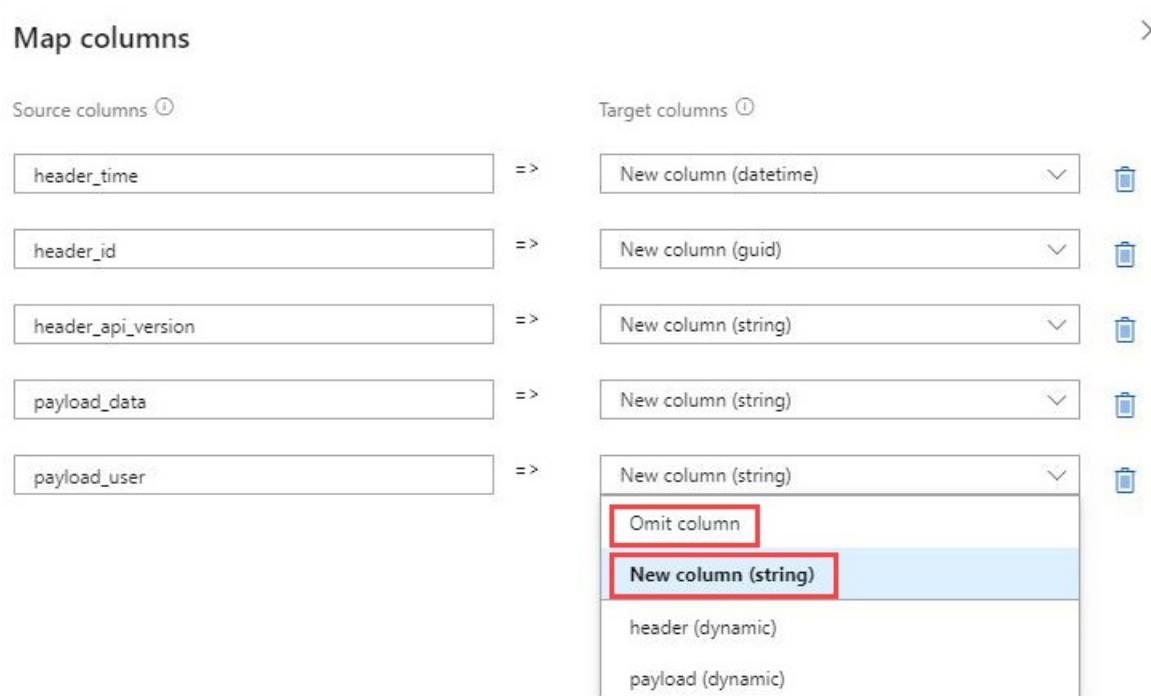
3. If you selected an existing table, the **Map columns** window opens to map source data columns to target table columns.

- Use **Omit column** to remove a target column from the table.
- Use **New column** to add a new column to the table.

Map columns

Source columns ①	=>	Target columns ①
header_time	=>	New column (datetime)
header_id	=>	New column (guid)
header_api_version	=>	New column (string)
payload_data	=>	New column (string)
payload_user	=>	New column (string) Omit column New column (string)
		header (dynamic) payload (dynamic)

Cancel **Update**



4. In the **Schema** tab:

- Select **Compression type** from the drop-down menu, and then select either **Uncompressed** or **GZip**.
- Select **Data format** from the drop-down menu, and then select **JSON**, **CSV**, **TSV**, **SCSV**, **SOHSV**, **TSVE**, or **PSV**.
 - When you select the **JSON** format, you must also select **JSON levels**, from 1 to 10. The levels affect the table column data depiction.
 - If you select a format other than JSON, you must select the check box **Include column names** to ignore the heading row of the file.
- **Mapping name** is set automatically but can be edited.
- If you selected an existing table, you can select **Map columns** to open the **Map columns** window.

The screenshot shows the 'Ingest new data (preview)' interface in Azure Data Explorer. The 'Schema' tab is selected. On the left, there are configuration options: 'Compression type' (set to 'Uncompressed'), 'Data format' (set to 'CSV'), and 'Includes column names' (checked). Below these is the 'Mapping name' field, which contains 'ADXdocs_mapping2'. At the bottom of this section is a blue 'Map columns' button, which is also highlighted with a red box. On the right, the 'Editor' pane displays a list of JSON documents. Each document has an '_api_version__v2__' field followed by a string value like 'pa'. There are 10 such entries. At the bottom of the interface are 'Previous' and 'Start ingestion' buttons.

5. Above the **Editor** pane, select the **v** button to open the editor. In the editor, you can view and copy the automatic queries generated from your inputs.

6. In table:

- Right-click on new column headers to **Change data type**, **Rename column**, **Delete column**, **Sort ascending**, or **Sort descending**. On existing columns, only data sorting is available.
- Double-click the new column name to edit.

7. Select **Start ingestion** to create a table and mapping and to begin data ingestion.

Azure Data Explorer

Ingest new data (preview)

Ingest new data into table myadxclustertest.koreacentral > docstest > ADXdocs

Source Schema

ADXdocs

Compression type: Uncompressed

Data format: JSON

JSON levels: 2

Mapping name: ADXdocs_mapping

Editor

```

1 // Create table command
2 /////////////////////////////////
3 .create table ['ADXdocs'] ([header_time]:datetime, [header_id]:guid,
   [header_api_version]:string, [payload_data]:string, [payload_user]:string)
4
5 // Create mapping command
6 /////////////////////////////////
7 .create table ['ADXdocs'] ingestion json mapping 'ADXdocs_mapping' [
  {"column": "header_time", "path": "$.header.time", "datatype": "datetime"},
  {"column": "header_id", "path": "$.header.id", "datatype": "guid"},
  {"column": "header_api_version", "path": "$.header.api_version", "datatype": "string"}
]

```

header_time (datetime)	header_id (guid)	header_api_version (string)	payload_data (string)
2018-08-24 09:42:15.0000	0944f542-a637-411b-94dd-887...	v2	NEEUGQSPKPD
2018-08-24 09:42:27.0000	09f7c3a2-27e0-4a9b-b00a-353...	v1	MSLAMKKSTOK
2018-08-24 09:42:47.0000	e0e4a6dd-3823-412f-ad0c-84b...	v1	QZWCBJKBPVE
2018-08-24 09:42:56.0000	e52cd01e-6984-4821-a4aa-a97...	v2	LEWDDGKXFGN
2018-08-24 09:43:14.0000	6a56af09-1610-4932-b596-84d...	v2	TQEEMYTKPEKOI
2018-08-24 09:43:37.0000	a5d0161f-ba96-407b-a10a-462...	v1	WABJKYNMPAR
2018-08-24 09:43:57.0000	a0250103-76a7-46aa-bac9-6eb...	v2	IERDQXNNJNLN
2018-08-24 09:44:11.0000	a774edf2-0f17-47a4-87f2-bd22...	v2	DOQHIEEJWUIE
2018-08-24 09:44:28.0000	23702971-372f-43ff-b10b-35bc...	v2	RWMOLFUNK
2018-08-24 09:44:56.0000	3eaf2cd4-8244-4f70-b3de-f823...	v1	ZXHJBTDKDHKT
2018-08-24 09:45:05.0000	ebbc83b8-5dbb-46ad-bb91-c8...	v1	VNYHIHRYYHAL
2018-08-24 09:45:15.0000	7b57c9c5-4aa4-4e68-93a5-645...	v1	YMNOHFNKTA
2018-08-24 09:45:15.0000	3ff9945-e776-4156-8825-16c9...	v2	PASJPMDFZFSDC
2018-08-24 09:45:39.0000	a38a3660-0dfb-498e-8253-c90...	v2	YTBVHDVKQLC
2018-08-24 09:45:42.0000	37e75fb-e78d-4478-bb6c-42d...	v1	KJBCPSAKJRGSA
2018-08-24 09:45:54.0000	95aa520a-64e1-4e7d-9dfe-d3c...	v2	UVUIYSAGLKAIC
2018-08-24 09:46:10.0000	03610479-aa33-4f46-8e02-90f...	v2	KWFKXKCCILGZ
2018-08-24 09:46:27.0000	514b6331-92ac-4e99-80c9-686...	v2	BZCVSWTXFTBF
2018-08-24 09:46:34.0000	90614bfb-d434-4396-b394-41...	v1	BJKIAXNSNVERI

[Previous](#) [Start ingestion](#)

Query data

- In the **Data ingestion completed** window, all three steps will be marked with green check marks if data ingestion completes successfully.

Azure Data Explorer

Data ingestion completed

- Create table ADXDocs
- Create mapping
- Data ingestion

```

1 .create table ['ADXDocs'] ([header]:dynamic, [payload]:dynamic)

```

Quick queries
Query sample
Number of rows

Tools
Drop commands

header	payload
> {"api_version": "v2", "time": "24-Aug-18 09:42:15", "id": "0944f542-a637-411b-94dd-887...", "user": "owild@fabrikam.c...	{"data": "NEEUGQSPKPDQPVFE", "user": "owild@fabrikam.c...

[Close](#)

2. Select the **v** button to open the query. Copy to the Web UI to edit the query.
3. The menu on the right contains **Quick queries** and **Tools** options.
 - **Quick queries** includes links to the Web UI with example queries.
 - **Tools** includes a link to **Drop commands** on the Web UI, which allow you to troubleshoot issues by running the relevant `.drop` commands.

TIP

Data may be lost by using `.drop` commands. Use them carefully.

Next steps

- [Query data in Azure Data Explorer Web UI](#)
- [Write queries for Azure Data Explorer using Kusto Query Language](#)

Streaming ingestion (Preview)

11/26/2019 • 2 minutes to read • [Edit Online](#)

Streaming ingestion is targeted for scenarios that require low latency with an ingestion time of less than 10 seconds for varied volume data. It's used for optimization of operational processing of many tables, in one or more databases where the stream of data into each table is relatively small (few records per second) but overall data ingestion volume is high (thousands of records per second).

Use the classic (bulk) ingestion instead of streaming ingestion when the amount of data grows to more than 1 MB per second per table. Read [Data ingestion overview](#) to learn more about the various methods of ingestion.

NOTE

Streaming ingestion doesn't support the following features:

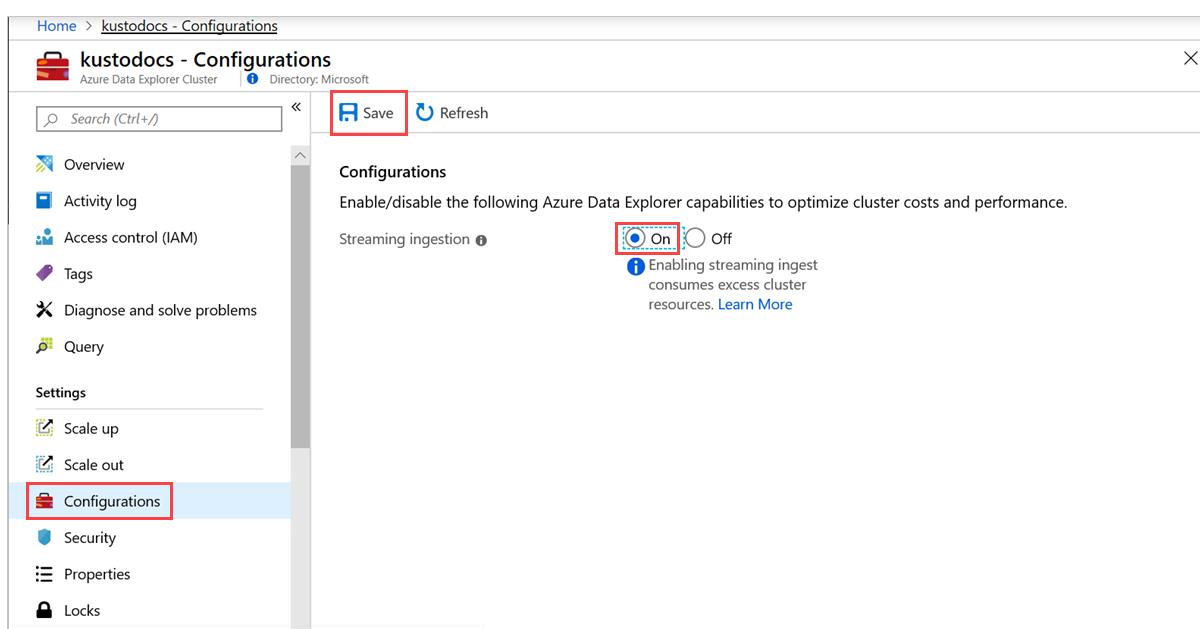
- [Database cursors](#).
- [Data mapping](#). Only [pre-created](#) data mapping is supported.

Prerequisites

- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.
- Sign in to the [Web UI](#).
- Create [an Azure Data Explorer cluster and database](#)

Enable streaming ingestion on your cluster

1. In the Azure portal, go to your Azure Data Explorer cluster. In **Settings**, select **Configurations**.
2. In the **Configurations** pane, select **On** to enable **Streaming ingestion**.
3. Select **Save**.



4. In the [Web UI](#), define [streaming ingestion policy](#) on table(s) or database(s) that will receive streaming data.

NOTE

- If the policy is defined at the database level, all tables in the database are enabled for streaming ingestion.
- The applied policy can reference only newly ingested data and not other tables in the database.

Use streaming ingestion to ingest data to your cluster

There are two supported streaming ingestion types:

- [Event Hub](#) used as a data source
- Custom ingestion requires you to write an application that uses one of Azure Data Explorer client libraries. See [streaming ingestion sample](#) for a sample application.

Choose the appropriate streaming ingestion type

	EVENT HUB	CUSTOM INGESTION
Data delay between ingestion initiation and the data available for query	longer delay	shorter delay
Development overhead	fast and easy setup, no development overhead	high development overhead for application to handle errors and ensure data consistency

Disable streaming ingestion on your cluster

WARNING

Disabling streaming ingestion may take a few hours.

1. Drop [streaming ingestion policy](#) from all relevant tables and databases. The streaming ingestion policy removal triggers streaming ingestion data movement from the initial storage to the permanent storage in the column store (extents or shards). The data movement can last between a few seconds to a few hours, depending on the amount of data in the initial storage and the CPU and memory utilization of the cluster.
2. In the Azure portal, go to your Azure Data Explorer cluster. In **Settings**, select **Configurations**.
3. In the **Configurations** pane, select **Off** to disable **Streaming ingestion**.
4. Select **Save**.

The screenshot shows the 'Configurations' page in the Azure Data Explorer Cluster. The left sidebar has a 'Configurations' section highlighted with a red box. The main area shows a 'Configurations' section with a sub-section for 'Streaming ingestion'. A radio button for 'Off' is selected and highlighted with a red box. At the top right are 'Save' and 'Refresh' buttons.

Limitations

- Streaming ingestion performance and capacity scales with increased VM and cluster sizes. Concurrent ingestions are limited to 6 ingestions per core. For example, for 16 core SKUs, such as D14 and L16, the maximal supported load is 96 concurrent ingestions. For 2 core SKUs, such as D11, the maximal supported load is 12 concurrent ingestions.
- The data size limitation per ingestion request is 4 MB.
- Schema updates, such as creation and modification of tables and ingestion mappings, may take up to 5 minutes for the streaming ingestion service.
- Enabling streaming ingestion on a cluster, even when data isn't ingested via streaming, uses part of the local SSD disk of the cluster machines for streaming ingestion data and reduces the storage available for hot cache.

Next steps

- [Query data in Azure Data Explorer](#)

Ingest data from Kafka into Azure Data Explorer

6/4/2019 • 2 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Azure Data Explorer offers ingestion (data loading) from Kafka. Kafka is a distributed streaming platform that allows building of real-time streaming data pipelines that reliably move data between systems or applications.

Prerequisites

- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.
- [A test cluster and database](#).
- [A sample app](#) that generates data and sends it to Kafka.
- [Visual Studio 2019](#) to run the sample app.

Kafka connector setup

Kafka Connect is a tool for scalable and reliable streaming of data between Apache Kafka and other systems. It makes it simple to quickly define connectors that move large collections of data into and out of Kafka. The ADX Kafka Sink serves as the connector from Kafka.

Bundle

Kafka can load a `.jar` as a plugin that will act as a custom connector. To produce such a `.jar`, we will clone the code locally and build using Maven.

Clone

```
git clone git://github.com/Azure/kafka-sink-azure-kusto.git
cd ./kafka-sink-azure-kusto/kafka/
```

Build

Build locally with Maven to produce a `.jar` complete with dependencies.

- JDK >= 1.8 [download](#)
- Maven [download](#)

Inside the root directory `kafka-sink-azure-kusto`, run:

```
mvn clean compile assembly:single
```

Deploy

Load plugin into Kafka. A deployment example using docker can be found at [kafka-sink-azure-kusto](#)

Detailed documentation on Kafka connectors and how to deploy them can be found at [Kafka Connect](#)

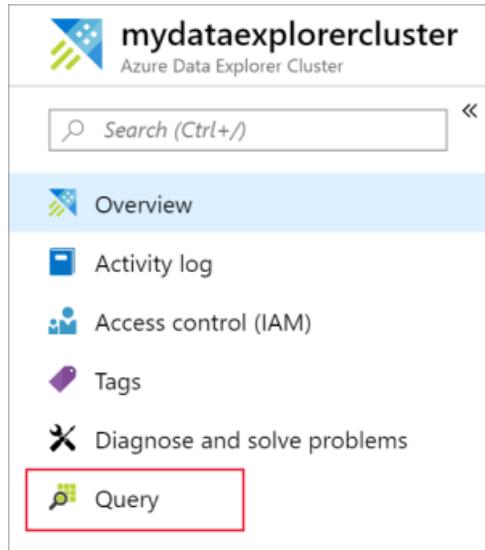
Example configuration

```
name=KustoSinkConnector
connector.class=com.microsoft.azure.kusto.kafka.connect.sink.KustoSinkConnector
kusto.sink.flush_interval_ms=300000
key.converter=org.apache.kafka.connect.storage.StringConverter
value.converter=org.apache.kafka.connect.storage.StringConverter
tasks.max=1
topics=test1
kusto.tables.topics_mapping=[{'topic': 'test1','db': 'daniel', 'table': 'TestTable','format': 'json',
'mapping':'TestMapping'}]
kusto.auth.authority=XXX
kusto.url=https://ingest-{mycluster}.kusto.windows.net/
kusto.auth.appid=XXX
kusto.auth.appkey=XXX
kusto.sink.tempdir=/var/tmp/
kusto.sink.flush_size=1000
```

Create a target table in ADX

Create a table in ADX to which Kafka can send data. Create the table in the cluster and database provisioned in the **Prerequisites**.

1. In the Azure portal, navigate to your cluster and select **Query**.



2. Copy the following command into the window and select **Run**.

```
.create table TestTable (TimeStamp: datetime, Name: string, Metric: int, Source:string)
```

The screenshot shows the Azure Data Explorer interface. On the left, there's a navigation pane with a tree view. Under 'docscluster.WestUS.TestDatabase', 'TestDatabase' is selected, revealing two tables: 'StormEvents' and 'TestTable'. The main area displays a table named 'Table 1' with one record. The table has columns: 'TableName', 'Schema', 'DatabaseName', and 'Folder'. The data row shows 'TestTable', '{"Name":"TestTable","OrderedC...', 'TestDatabase', and an empty folder icon. At the top, there are tabs for 'Run' (highlighted with a red box), 'Recall', and 'Scope: docscluster.WestUS.TestDatabase'. Below the table, there's a status bar indicating 'Done (0.668 s)' and '1 records'.

- Copy the following command into the window and select **Run**.

```
.create table TestTable ingestion json mapping 'TestMapping'
[{"column":"TimeStamp","path":("$.timeStamp","datatype":"datetime"),
 {"column":"Name","path":("$.name","datatype":"string"),
 {"column":"Metric","path":("$.metric","datatype":"int"),
 {"column":"Source","path":("$.source","datatype":"string")}]'
```

This command maps incoming JSON data to the column names and data types of the table (TestTable).

Generate sample data

Now that the Kafka cluster is connected to ADX, use the [sample app](#) you downloaded to generate data.

Clone

Clone the sample app locally:

```
git clone git://github.com/Azure/azure-kusto-samples-dotnet.git
cd ./azure-kusto-samples-dotnet/kafka
```

Run the app

- Open the sample app solution in Visual Studio.

- In the `Program.cs` file, update the `connectionString` constant to your Kafka connection string.

```
const string connectionString = @"<YourConnectionString>";
```

- Build and run the app. The app sends messages to the Kafka cluster, and it prints out its status every 10 seconds.
- After the app has sent a few messages, move on to the next step.

Query and review the data

- To make sure no errors occurred during ingestion:

```
.show ingestion failures
```

- To see the newly ingested data:

```
TestTable  
| count
```

3. To see the content of the messages:

```
TestTable
```

The result set should look like the following:

TestTable			
TimeStamp	Name	Metric	Source
2018-09-18T18:19:17Z	name 0	0	EventHubMessage
2018-09-18T18:19:17Z	name 0	1	EventHubMessage
2018-09-18T18:19:17Z	name 0	2	EventHubMessage
2018-09-18T18:24:29Z	name 3	3	EventHubMessage
2018-09-18T18:24:29Z	name 3	4	EventHubMessage
2018-09-18T18:24:29Z	name 3	5	EventHubMessage
2018-09-18T18:29:39Z	name 6	6	EventHubMessage
2018-09-18T18:29:39Z	name 6	7	EventHubMessage
2018-09-18T18:29:39Z	name 6	8	EventHubMessage

Next steps

- [Query data in Azure Data Explorer](#)

Ingest data using the Azure Data Explorer Python library

7/19/2019 • 4 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Azure Data Explorer provides two client libraries for Python: an [ingest library](#) and [a data library](#). These libraries enable you to ingest (load) data into a cluster and query data from your code. In this article, you first create a table and data mapping in a cluster. You then queue ingestion to the cluster and validate the results.

This article is also available as an [Azure Notebook](#).

Prerequisites

- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.
- [A cluster and database](#)
- [Python](#) installed on your development computer

Install the data and ingest libraries

Install *azure-kusto-data* and *azure-kusto-ingest*.

```
pip install azure-kusto-data
pip install azure-kusto-ingest
```

Add import statements and constants

Import classes from *azure-kusto-data*.

```
from azure.kusto.data.request import KustoClient, KustoConnectionStringBuilder
from azure.kusto.data.exceptions import KustoServiceError
from azure.kusto.data.helpers import dataframe_from_result_table
```

To authenticate an application, Azure Data Explorer uses your AAD tenant ID. To find your tenant ID, use the following URL, substituting your domain for *YourDomain*.

```
https://login.windows.net/<YourDomain>/.well-known/openid-configuration/
```

For example, if your domain is *contoso.com*, the URL is: <https://login.windows.net/contoso.com/.well-known/openid-configuration/>. Click this URL to see the results; the first line is as follows.

```
"authorization_endpoint": "https://login.windows.net/6babcaad-604b-40ac-a9d7-9fd97c0b779f/oauth2/authorize"
```

The tenant ID in this case is `6babcaad-604b-40ac-a9d7-9fd97c0b779f`. Set the values for *AAD_TENANT_ID*, *KUSTO_URI*, *KUSTO_INGEST_URI*, and *KUSTO_DATABASE* before running this code.

```

AAD_TENANT_ID = "<TenantId>"
KUSTO_URI = "https://<ClusterName>.<Region>.kusto.windows.net:443/"
KUSTO_INGEST_URI = "https://ingest-<ClusterName>.<Region>.kusto.windows.net:443/"
KUSTO_DATABASE = "<DatabaseName>"
```

Now construct the connection string. This example uses device authentication to access the cluster. You can also use [AAD application certificate](#), [AAD application key](#), and [AAD user and password](#).

You create the destination table and mapping in a later step.

```

KCSB_INGEST = KustoConnectionStringBuilder.with_aad_device_authentication(
    KUSTO_INGEST_URI, AAD_TENANT_ID)

KCSB_DATA = KustoConnectionStringBuilder.with_aad_device_authentication(
    KUSTO_URI, AAD_TENANT_ID)

DESTINATION_TABLE = "StormEvents"
DESTINATION_TABLE_COLUMN_MAPPING = "StormEvents_CSVMapping"
```

Set source file information

Import additional classes and set constants for the data source file. This example uses a sample file hosted on Azure Blob Storage. The **StormEvents** sample data set contains weather-related data from the [National Centers for Environmental Information](#).

```

from azure.storage.blob import BlockBlobService
from azure.kusto.ingest import KustoIngestClient, IngestionProperties, FileDescriptor, BlobDescriptor,
DataFormat, ReportLevel, ReportMethod

CONTAINER = "samplefiles"
ACCOUNT_NAME = "kustosamplefiles"
SAS_TOKEN = "?st=2018-08-31T22%3A02%3A25Z&se=2020-09-01T22%3A02%3A00Z&sp=r&sv=2018-03-
28&sr=b&sig=LQIbomcKI80oz425hWtjeq6d61uEaq21UVX7YrM61N4%3D"
FILE_PATH = "StormEvents.csv"
FILE_SIZE = 64158321    # in bytes

BLOB_PATH = "https://" + ACCOUNT_NAME + ".blob.core.windows.net/" + \
CONTAINER + "/" + FILE_PATH + SAS_TOKEN
```

Create a table on your cluster

Create a table that matches the schema of the data in the StormEvents.csv file. When this code runs, it returns a message like the following: *To sign in, use a web browser to open the page <https://microsoft.com/devicelogin> and enter the code F3W4VWZDM to authenticate.* Follow the steps to sign in, then return to run the next code block. Subsequent code blocks that make a connection require you to sign in again.

```

KUSTO_CLIENT = KustoClient(KCSB_DATA)
CREATE_TABLE_COMMAND = ".create table StormEvents (StartTime: datetime, EndTime: datetime, EpisodeId: int,
EventId: int, State: string, EventType: string, InjuriesDirect: int, InjuriesIndirect: int, DeathsDirect: int,
DeathsIndirect: int, DamageProperty: int, DamageCrops: int, Source: string, BeginLocation: string,
EndLocation: string, BeginLat: real, BeginLon: real, EndLat: real, EndLon: real, EpisodeNarrative: string,
EventNarrative: string, StormSummary: dynamic)"

RESPONSE = KUSTO_CLIENT.execute_mgmt(KUSTO_DATABASE, CREATE_TABLE_COMMAND)

dataframe_from_result_table(RESPONSE.primary_results[0])
```

Define ingestion mapping

Map incoming CSV data to the column names and data types used when creating the table. This maps source data fields to destination table columns

```
CREATE_MAPPING_COMMAND = """.create table StormEvents ingestion csv mapping 'StormEvents_CSV_Mapping'
'[{"Name":"StartTime","datatype":"datetime","Ordinal":0},
 {"Name":"EndTime","datatype":"datetime","Ordinal":1}, {"Name":"EpisodeId","datatype":"int","Ordinal":2},
 {"Name":"EventId","datatype":"int","Ordinal":3}, {"Name":"State","datatype":"string","Ordinal":4},
 {"Name":"EventType","datatype":"string","Ordinal":5}, {"Name":"InjuriesDirect","datatype":"int","Ordinal":6},
 {"Name":"InjuriesIndirect","datatype":"int","Ordinal":7}, {"Name":"DeathsDirect","datatype":"int","Ordinal":8},
 {"Name":"DeathsIndirect","datatype":"int","Ordinal":9},
 {"Name":"DamageProperty","datatype":"int","Ordinal":10}, {"Name":"DamageCrops","datatype":"int","Ordinal":11},
 {"Name":"Source","datatype":"string","Ordinal":12}, {"Name":"BeginLocation","datatype":"string","Ordinal":13},
 {"Name":"EndLocation","datatype":"string","Ordinal":14}, {"Name":"BeginLat","datatype":"real","Ordinal":16},
 {"Name":"BeginLon","datatype":"real","Ordinal":17}, {"Name":"EndLat","datatype":"real","Ordinal":18},
 {"Name":"EndLon","datatype":"real","Ordinal":19}, {"Name":"EpisodeNarrative","datatype":"string","Ordinal":20},
 {"Name":"EventNarrative","datatype":"string","Ordinal":21},
 {"Name":"StormSummary","datatype":"dynamic","Ordinal":22}]"""

RESPONSE = KUSTO_CLIENT.execute_mgmt(KUSTO_DATABASE, CREATE_MAPPING_COMMAND)

dataframe_from_result_table(RESPONSE.primary_results[0])
```

Queue a message for ingestion

Queue a message to pull data from blob storage and ingest that data into Azure Data Explorer.

```
INGESTION_CLIENT = KustoIngestClient(KCSB_INGEST)

# All ingestion properties are documented here: https://docs.microsoft.com/azure/kusto/management/data-ingest#ingestion-properties
INGESTION_PROPERTIES = IngestionProperties(database=KUSTO_DATABASE, table=DESTINATION_TABLE,
dataFormat=DataFormat.csv,
                                mappingReference=DESTINATION_TABLE_COLUMN_MAPPING,
additionalProperties={'ignoreFirstRecord': 'true'})
# FILE_SIZE is the raw size of the data in bytes
BLOB_DESCRIPTOR = BlobDescriptor(BLOB_PATH, FILE_SIZE)
INGESTION_CLIENT.ingest_from_blob(
    BLOB_DESCRIPTOR, ingestion_properties=INGESTION_PROPERTIES)

print('Done queuing up ingestion with Azure Data Explorer')
```

Query data that was ingested into the table

Wait for five to ten minutes for the queued ingestion to schedule the ingest and load the data into Azure Data Explorer. Then run the following code to get the count of records in the StormEvents table.

```
QUERY = "StormEvents | count"

RESPONSE = KUSTO_CLIENT.execute_query(KUSTO_DATABASE, QUERY)

dataframe_from_result_table(RESPONSE.primary_results[0])
```

Run troubleshooting queries

Sign in to <https://dataexplorer.azure.com> and connect to your cluster. Run the following command in your database to see if there were any ingestion failures in the last four hours. Replace the database name before

running.

```
.show ingestion failures  
| where FailedOn > ago(4h) and Database == "<DatabaseName>"
```

Run the following command to view the status of all ingestion operations in the last four hours. Replace the database name before running.

```
.show operations  
| where StartedOn > ago(4h) and Database == "<DatabaseName>" and Table == "StormEvents" and Operation ==  
"DataIngestPull"  
| summarize arg_max>LastUpdatedOn, * by OperationId
```

Clean up resources

If you plan to follow our other articles, keep the resources you created. If not, run the following command in your database to clean up the StormEvents table.

```
.drop table StormEvents
```

Next steps

- [Query data using Python](#)

Ingest data using the Azure Data Explorer Node library

6/4/2019 • 3 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Azure Data Explorer provides two client libraries for Node: an [ingest library](#) and a [data library](#). These libraries enable you to ingest (load) data into a cluster and query data from your code. In this article, you first create a table and data mapping in a test cluster. You then queue ingestion to the cluster and validate the results.

If you don't have an Azure subscription, create a [free Azure account](#) before you begin.

Prerequisites

In addition to an Azure subscription, you need the following to complete this article:

- [A test cluster and database](#)
- [Node.js](#) installed on your development computer

Install the data and ingest libraries

Install *azure-kusto-ingest* and *azure-kusto-data*

```
npm i azure-kusto-ingest azure-kusto-data
```

Add import statements and constants

Import classes from the libraries

```
const KustoConnectionStringBuilder = require("azure-kusto-data").KustoConnectionStringBuilder;
const KustoClient = require("azure-kusto-data").Client;
const KustoIngestClient = require("azure-kusto-ingest").IngestClient;
const IngestionProperties = require("azure-kusto-ingest").IngestionProperties;
const { DataFormat } = require("azure-kusto-ingest").IngestionPropertiesEnums;
const { BlobDescriptor } = require("azure-kusto-ingest").IngestionDescriptors;
```

To authenticate an application, Azure Data Explorer uses your Azure Active Directory tenant ID. To find your tenant ID, follow [Find your Office 365 tenant ID](#).

Set the values for `authorityId`, `kustoUri`, `kustoIngestUri` and `kustoDatabase` before running this code.

```
const cluster = "MyCluster";
const region = "westus";
const authorityId = "microsoft.com";
const kustoUri = `https://${cluster}.${region}.kusto.windows.net:443`;
const kustoIngestUri = `https://ingest-${cluster}.${region}.kusto.windows.net:443`;
const kustoDatabase = "Weather";
```

Now construct the connection string. This example uses device authentication to access the cluster. You can also

use Azure Active Directory application certificate, application key, and user and password.

You create the destination table and mapping in a later step.

```
const kcsbIngest = KustoConnectionStringBuilder.withAadDeviceAuthentication(kustoIngestUri, authorityId);
const kcsbData = KustoConnectionStringBuilder.withAadDeviceAuthentication(kustoUri, authorityId);
const destTable = "StormEvents";
const destTableMapping = "StormEvents_CSVM_Mapping";
```

Set source file information

Import additional classes and set constants for the data source file. This example uses a sample file hosted on Azure Blob Storage. The **StormEvents** sample data set contains weather-related data from the [National Centers for Environmental Information](#).

```
const container = "samplefiles";
const account = "kustosamplefiles";
const sas = "?st=2018-08-31T22%3A02%3A25Z&se=2020-09-01T22%3A02%3A00Z&sp=r&sv=2018-03-
28&sr=b&sig=LQIBomcKI8Ooz425hWtjeq6d61uEaq21UVX7YrM61N4%3D";
const filePath = "StormEvents.csv";
const blobPath = `https://${account}.blob.core.windows.net/${container}/${filePath}${sas}`;
```

Create a table on your test cluster

Create a table that matches the schema of the data in the `StormEvents.csv` file. When this code runs, it returns a message like the following: *To sign in, use a web browser to open the page <https://microsoft.com/devicelogin> and enter the code XXXXXXXXX to authenticate.* Follow the steps to sign in, then return to run the next code block. Subsequent code blocks that make a connection will require you to sign in again.

```
const kustoClient = new KustoClient(kcsbData);
const createTableCommand = ` .create table ${destTable} (StartTime: datetime, EndTime: datetime, EpisodeId: int, EventId: int, State: string, EventType: string, InjuriesDirect: int, InjuriesIndirect: int, DeathsDirect: int, DeathsIndirect: int, DamageProperty: int, DamageCrops: int, Source: string, BeginLocation: string, EndLocation: string, BeginLat: real, BeginLon: real, EndLat: real, EndLon: real, EpisodeNarrative: string, EventNarrative: string, StormSummary: dynamic)`;

kustoClient.executeMgmt(kustoDatabase, createTableCommand, (err, results) => {
  console.log(results.primaryResults[0][0].toString());
});
```

Define ingestion mapping

Map incoming CSV data to the column names and data types used when creating the table.

```

const createMappingCommand = ` .create table ${destTable} ingestion csv mapping '${destTableMapping}'
[{"Name":"StartTime","datatype":"datetime","Ordinal":0},
 {"Name":"EndTime","datatype":"datetime","Ordinal":1}, {"Name":"EpisodeId","datatype":"int","Ordinal":2},
 {"Name":"EventId","datatype":"int","Ordinal":3}, {"Name":"State","datatype":"string","Ordinal":4},
 {"Name":"EventType","datatype":"string","Ordinal":5}, {"Name":"InjuriesDirect","datatype":"int","Ordinal":6},
 {"Name":"InjuriesIndirect","datatype":"int","Ordinal":7}, {"Name":"DeathsDirect","datatype":"int","Ordinal":8},
 {"Name":"DeathsIndirect","datatype":"int","Ordinal":9},
 {"Name":"DamageProperty","datatype":"int","Ordinal":10}, {"Name":"DamageCrops","datatype":"int","Ordinal":11},
 {"Name":"Source","datatype":"string","Ordinal":12}, {"Name":"BeginLocation","datatype":"string","Ordinal":13},
 {"Name":"EndLocation","datatype":"string","Ordinal":14}, {"Name":"BeginLat","datatype":"real","Ordinal":16},
 {"Name":"BeginLon","datatype":"real","Ordinal":17}, {"Name":"EndLat","datatype":"real","Ordinal":18},
 {"Name":"EndLon","datatype":"real","Ordinal":19}, {"Name":"EpisodeNarrative","datatype":"string","Ordinal":20},
 {"Name":"EventNarrative","datatype":"string","Ordinal":21},
 {"Name":"StormSummary","datatype":"dynamic","Ordinal":22}]``;

kustoClient.executeMgmt(kustoDatabase, createMappingCommand, (err, results) => {
  console.log(results.primaryResults[0][0].toString());
});

```

Queue a message for ingestion

Queue a message to pull data from blob storage and ingest that data into Azure Data Explorer.

```

const defaultProps = new IngestionProperties(kustoDatabase, destTable, DataFormat.csv, null, destTableMapping,
{'ignoreFirstRecord': 'true'});
const ingestClient = new KustoIngestClient(kcsbIngest, defaultProps);
// All ingestion properties are documented here: https://docs.microsoft.com/azure/kusto/management/data-ingest#ingestion-properties

const blobDesc = new BlobDescriptor(blobPath, 10);
ingestClient.ingestFromBlob(blobDesc, null, (err) => {
  if (err) throw new Error(err);
});

```

Validate that table contains data

Validate that the data was ingested into the table. Wait for five to ten minutes for the queued ingestion to schedule the ingest and load the data into Azure Data Explorer. Then run the following code to get the count of records in the `StormEvents` table.

```

const query = `${destTable} | count`;

kustoClient.execute(kustoDatabase, query, (err, results) => {
  if (err) throw new Error(err);
  console.log(results.primaryResults[0][0].toString());
});

```

Run troubleshooting queries

Sign in to <https://dataexplorer.azure.com> and connect to your cluster. Run the following command in your database to see if there were any ingestion failures in the last four hours. Replace the database name before running.

```

.show ingestion failures
| where FailedOn > ago(4h) and Database == "<DatabaseName>"

```

Run the following command to view the status of all ingestion operations in the last four hours. Replace the database name before running.

```
.show operations
| where StartedOn > ago(4h) and Database == "<DatabaseName>" and Operation == "DataIngestPull"
| summarize arg_max>LastUpdatedOn, * by OperationId
```

Clean up resources

If you plan to follow our other articles, keep the resources you created. If not, run the following command in your database to clean up the `StormEvents` table.

```
.drop table StormEvents
```

Next steps

- [Write queries](#)

Ingest data using the Azure Data Explorer .NET Standard SDK (Preview)

6/4/2019 • 4 minutes to read • [Edit Online](#)

Azure Data Explorer (ADX) is a fast and highly scalable data exploration service for log and telemetry data. ADX provides two client libraries for .NET Standard: an [ingest library](#) and a [data library](#). These libraries enable you to ingest (load) data into a cluster and query data from your code. In this article, you first create a table and data mapping in a test cluster. You then queue an ingestion to the cluster and validate the results.

Prerequisites

- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.
- [A test cluster and database](#)

Install the ingest library

```
Install-Package Microsoft.Azure.Kusto.Ingest.NETStandard
```

Authentication

To authenticate an application, Azure Data Explorer uses your AAD tenant ID. To find your tenant ID, use the following URL, substituting your domain for *YourDomain*.

```
https://login.windows.net/<YourDomain>/.well-known/openid-configuration/
```

For example, if your domain is *contoso.com*, the URL is: <https://login.windows.net/contoso.com/.well-known/openid-configuration/>. Click this URL to see the results; the first line is as follows.

```
"authorization_endpoint": "https://login.windows.net/6babcaad-604b-40ac-a9d7-9fd97c0b779f/oauth2/authorize"
```

The tenant ID in this case is `6babcaad-604b-40ac-a9d7-9fd97c0b779f`.

This example uses an AAD user and password for authentication to access the cluster. You can also use AAD application certificate and AAD application key. Set the your values for `tenantId`, `user`, and `password` before running this code.

```
var tenantId = "<TenantId>";
var user = "<User>";
var password = "<Password>";
```

Construct the connection string

Now construct the connection string. You create the destination table and mapping in a later step.

```

var kustoUri = "https://<ClusterName>.<Region>.kusto.windows.net:443/";
var database = "<DatabaseName>";

var kustoConnectionStringBuilder =
    new KustoConnectionStringBuilder(kustoUri)
{
    FederatedSecurity = true,
    InitialCatalog = database,
    UserID = user,
    Password = password,
    Authority = tenantId
};

```

Set source file information

Set the path for the source file. This example uses a sample file hosted on Azure Blob Storage. The **StormEvents** sample data set contains weather-related data from the [National Centers for Environmental Information](#).

```

var blobPath = "https://kustosamplefiles.blob.core.windows.net/samplefiles/StormEvents.csv?st=2018-08-
31T22%3A02%3A25Z&se=2020-09-01T22%3A02%3A00Z&sp=r&sv=2018-03-
28&sr=b&sig=LQIbomcKI8Ooz425hWtjeq6d61uEaq21UVX7YrM61N4%3D";

```

Create a table on your test cluster

Create a table named `StormEvents` that matches the schema of the data in the `StormEvents.csv` file.

```

var table = "StormEvents";
using (var kustoClient = KustoClientFactory.CreateCs1AdminProvider(kustoConnectionStringBuilder))
{
    var command =
        Cs1CommandGenerator.GenerateTableCreateCommand(
            table,
            new[]
            {
                Tuple.Create("StartTime", "System.DateTime"),
                Tuple.Create("EndTime", "System.DateTime"),
                Tuple.Create("EpisodeId", "System.Int32"),
                Tuple.Create("EventId", "System.Int32"),
                Tuple.Create("State", "System.String"),
                Tuple.Create("EventType", "System.String"),
                Tuple.Create("InjuriesDirect", "System.Int32"),
                Tuple.Create("InjuriesIndirect", "System.Int32"),
                Tuple.Create("DeathsDirect", "System.Int32"),
                Tuple.Create("DeathsIndirect", "System.Int32"),
                Tuple.Create("DamageProperty", "System.Int32"),
                Tuple.Create("DamageCrops", "System.Int32"),
                Tuple.Create("Source", "System.String"),
                Tuple.Create("BeginLocation", "System.String"),
                Tuple.Create("EndLocation", "System.String"),
                Tuple.Create("BeginLat", "System.Double"),
                Tuple.Create("BeginLon", "System.Double"),
                Tuple.Create("EndLat", "System.Double"),
                Tuple.Create("EndLon", "System.Double"),
                Tuple.Create("EpisodeNarrative", "System.String"),
                Tuple.Create("EventNarrative", "System.String"),
                Tuple.Create("StormSummary", "System.Object"),
            });
    kustoClient.ExecuteControlCommand(command);
}

```

Define ingestion mapping

Map the incoming CSV data to the column names used when creating the table. Provision a [CSV column mapping object](#) on that table

```
var tableMapping = "StormEvents_CSVMapping";
using (var kustoClient = KustoClientFactory.CreateCslAdminProvider(kustoConnectionStringBuilder))
{
    var command =
        CslCommandGenerator.GenerateTableCsvMappingCreateCommand(
            table,
            tableMapping,
            new[]
            {
                new CsvColumnMapping { ColumnName = "StartTime", Ordinal = 0 },
                new CsvColumnMapping { ColumnName = "EndTime", Ordinal = 1 },
                new CsvColumnMapping { ColumnName = "EpisodeId", Ordinal = 2 },
                new CsvColumnMapping { ColumnName = "EventId", Ordinal = 3 },
                new CsvColumnMapping { ColumnName = "State", Ordinal = 4 },
                new CsvColumnMapping { ColumnName = "EventType", Ordinal = 5 },
                new CsvColumnMapping { ColumnName = "InjuriesDirect", Ordinal = 6 },
                new CsvColumnMapping { ColumnName = "InjuriesIndirect", Ordinal = 7 },
                new CsvColumnMapping { ColumnName = "DeathsDirect", Ordinal = 8 },
                new CsvColumnMapping { ColumnName = "DeathsIndirect", Ordinal = 9 },
                new CsvColumnMapping { ColumnName = "DamageProperty", Ordinal = 10 },
                new CsvColumnMapping { ColumnName = "DamageCrops", Ordinal = 11 },
                new CsvColumnMapping { ColumnName = "Source", Ordinal = 12 },
                new CsvColumnMapping { ColumnName = "BeginLocation", Ordinal = 13 },
                new CsvColumnMapping { ColumnName = "EndLocation", Ordinal = 14 },
                new CsvColumnMapping { ColumnName = "BeginLat", Ordinal = 15 },
                new CsvColumnMapping { ColumnName = "BeginLon", Ordinal = 16 },
                new CsvColumnMapping { ColumnName = "EndLat", Ordinal = 17 },
                new CsvColumnMapping { ColumnName = "EndLon", Ordinal = 18 },
                new CsvColumnMapping { ColumnName = "EpisodeNarrative", Ordinal = 19 },
                new CsvColumnMapping { ColumnName = "EventNarrative", Ordinal = 20 },
                new CsvColumnMapping { ColumnName = "StormSummary", Ordinal = 21 },
            });
    kustoClient.ExecuteControlCommand(command);
}
```

Queue a message for ingestion

Queue a message to pull data from blob storage and ingest that data into ADX.

```

var ingestUri = "https://ingest-<ClusterName>.<Region>.kusto.windows.net:443/";
var ingestConnectionStringBuilder =
    new KustoConnectionStringBuilder(ingestUri)
{
    FederatedSecurity = true,
    InitialCatalog = database,
    UserID = user,
    Password = password,
    Authority = tenantId
};

using (var ingestClient = KustoIngestFactory.CreateQueuedIngestClient(ingestConnectionStringBuilder))
{
    var properties =
        new KustoQueuedIngestionProperties(database, table)
    {
        Format = DataSourceFormat.csv,
        CSVMappingReference = tableMapping,
        IgnoreFirstRecord = true
    };

    ingestClient.IngestFromSingleBlob(blobPath, deleteSourceOnSuccess: false, ingestionProperties:
properties);
}

```

Validate data was ingested into the table

Wait for five to ten minutes for the queued ingestion to schedule the ingest and load the data into ADX. Then run the following code to get the count of records in the `StormEvents` table.

```

using (var cslQueryProvider = KustoClientFactory.CreateCslQueryProvider(kustoConnectionStringBuilder))
{
    var query = $"{table} | count";

    var results = cslQueryProvider.ExecuteQuery<long>(query);
    Console.WriteLine(results.Single());
}

```

Run troubleshooting queries

Sign in to <https://dataexplorer.azure.com> and connect to your cluster. Run the following command in your database to see if there were any ingestion failures in the last four hours. Replace the database name before running.

```

.show ingestion failures
| where FailedOn > ago(4h) and Database == "<DatabaseName>"

```

Run the following command to view the status of all ingestion operations in the last four hours. Replace the database name before running.

```

.show operations
| where StartedOn > ago(4h) and Database == "<DatabaseName>" and Operation == "DataIngestPull"
| summarize arg_max>LastUpdatedOn, * by OperationId

```

Clean up resources

If you plan to follow our other articles, keep the resources you created. If not, run the following command in your

database to clean up the `StormEvents` table.

```
.drop table StormEvents
```

Next steps

- [Write queries](#)

Ingest data from Logstash to Azure Data Explorer

6/4/2019 • 3 minutes to read • [Edit Online](#)

Logstash is an open source, server-side data processing pipeline that ingests data from many sources simultaneously, transforms the data, and then sends the data to your favorite "stash". In this article, you'll send that data to Azure Data Explorer, which is a fast and highly scalable data exploration service for log and telemetry data. You'll initially create a table and data mapping in a test cluster, and then direct Logstash to send data into the table and validate the results.

Prerequisites

- An Azure subscription. If you don't have one, create a [free Azure account](#) before you begin.
- An Azure Data Explorer [test cluster and database](#)
- Logstash version 6+ [Installation instructions](#)

Create a table

After you have a cluster and a database, it's time to create a table.

1. Run the following command in your database query window to create a table:

```
.create table logs (timestamp: datetime, message: string)
```

2. Run the following command to confirm that the new table `logs` has been created and that it's empty:

```
logs  
| count
```

Create a mapping

Mapping is used by Azure Data Explorer to transform the incoming data into the target table schema. The following command creates a new mapping named `basicmsg` that extracts properties from the incoming json as noted by the `path` and outputs them to the `column`.

Run the following command in the query window:

```
.create table logs ingestion json mapping 'basicmsg' '[{"column":"timestamp","path":"$.@timestamp"},  
 {"column":"message","path":"$.message"}]'
```

Install the Logstash output plugin

The Logstash output plugin communicates with Azure Data Explorer and sends the data to the service. Run the following command inside the Logstash root directory to install the plugin:

```
bin/logstash-plugin install logstash-output-kusto
```

Configure Logstash to generate a sample dataset

Logstash can generate sample events that can be used to test an end-to-end pipeline. If you're already using Logstash and have access to your own event stream, skip to the next section.

NOTE

If you're using your own data, change the table and mapping objects defined in the previous steps.

1. Edit a new text file that will contain the required pipeline settings (using vi):

```
vi test.conf
```

2. Paste the following settings that will tell Logstash to generate 1000 test events:

```
input {  
  stdin { }  
  generator {  
    message => "Test Message 123"  
    count => 1000  
  }  
}
```

This configuration also includes the `stdin` input plugin that will enable you to write more messages by yourself (be sure to use *Enter* to submit them into the pipeline).

Configure Logstash to send data to Azure Data Explorer

Paste the following settings into the same config file used in the previous step. Replace all the placeholders with the relevant values for your setup. For more information, see [Creating an AAD Application](#).

```
output {  
  kusto {  
    path => "/tmp/kusto/%{+YYYY-MM-dd-HH-mm-ss}.txt"  
    ingest_url => "https://ingest-<cluster name>.kusto.windows.net/"  
    app_id => "<application id>"  
    app_key => "<application key/secret>"  
    app_tenant => "<tenant id>"  
    database => "<database name>"  
    table => "<target table>" # logs as defined above  
    mapping => "<mapping name>" # basicmsg as defined above  
  }  
}
```

PARAMETER NAME	DESCRIPTION
path	The Logstash plugin writes events to temporary files before sending them to Azure Data Explorer. This parameter includes a path where files should be written and a time expression for file rotation to trigger an upload to the Azure Data Explorer service.
ingest_url	The Kusto endpoint for ingestion-related communication.
app_id, app_key, and app_tenant	Credentials required to connect to Azure Data Explorer. Be sure to use an application with ingest privileges.

PARAMETER NAME	DESCRIPTION
database	Database name to place events.
table	Target table name to place events.
mapping	Mapping is used to map an incoming event json string into the correct row format (defines which property goes into which column).

Run Logstash

We are now ready to run Logstash and test our settings.

1. In the Logstash root directory, run the following command:

```
bin/logstash -f test.conf
```

You should see information printed to the screen, and then the 1000 messages generated by our sample configuration. At this point, you can also enter more messages manually.

2. After a few minutes, run the following Data Explorer query to see the messages in the table you defined:

```
logs
| order by timestamp desc
```

3. Select Ctrl+C to exit Logstash

Clean up resources

Run the following command in your database to clean up the `logs` table:

```
.drop table logs
```

Next steps

- [Write queries](#)

Copy data to Azure Data Explorer by using Azure Data Factory

9/25/2019 • 7 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast, fully managed, data-analytics service. It offers real-time analysis on large volumes of data that stream from many sources, such as applications, websites, and IoT devices. With Azure Data Explorer, you can iteratively explore data and identify patterns and anomalies to improve products, enhance customer experiences, monitor devices, and boost operations. It helps you explore new questions and get answers in minutes.

Azure Data Factory is a fully managed, cloud-based, data-integration service. You can use it to populate your Azure Data Explorer database with data from your existing system. It can help you save time when you're building analytics solutions.

When you load data into Azure Data Explorer, Data Factory provides the following benefits:

- **Easy setup:** Get an intuitive, five-step wizard with no scripting required.
- **Rich data store support:** Get built-in support for a rich set of on-premises and cloud-based data stores. For a detailed list, see the table of [Supported data stores](#).
- **Secure and compliant:** Data is transferred over HTTPS or Azure ExpressRoute. The global service presence ensures that your data never leaves the geographical boundary.
- **High performance:** The data-loading speed is up to 1 gigabyte per second (GBps) into Azure Data Explorer. For more information, see [Copy activity performance](#).

In this article, you use the Data Factory Copy Data tool to load data from Amazon Simple Storage Service (S3) into Azure Data Explorer. You can follow a similar process to copy data from other data stores, such as:

- [Azure Blob storage](#)
- [Azure SQL Database](#)
- [Azure SQL Data Warehouse](#)
- [Google BigQuery](#)
- [Oracle](#)
- [File system](#)

Prerequisites

- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.
- [An Azure Data Explorer cluster and database](#).
- A source of data.

Create a data factory

1. Sign in to the [Azure portal](#).
2. In the left pane, select **Create a resource > Analytics > Data Factory**.

The screenshot shows the Azure Marketplace 'New' pane. On the left is a navigation sidebar with various service icons and names. A red box highlights the 'Create a resource' button at the top of the sidebar. Below it, under 'FAVORITES', is a section for 'Analytics' services. A red box highlights the 'Analytics' category. Within this category, the 'Data Factory' service is shown with its icon and name, also enclosed in a red box.

New

Home > New

Search the Marketplace

Azure Marketplace See all Featured See all

- Get started
- Recently created
- Compute
- Networking
- Storage
- Web
- Mobile
- Containers
- Databases
- Analytics**
- AI + Machine Learning
- Internet of Things
- Mixed Reality
- Integration
- Security
- Identity
- Developer Tools
- Management Tools
- Software as a Service (SaaS)
- Blockchain

Azure Data Explorer
Learn more

HDInsight
Quickstart tutorial

Data Lake Analytics
Quickstart tutorial

Stream Analytics job
Quickstart tutorial

Analysis Services
Quickstart tutorial

Azure Databricks
Quickstart tutorial

Power BI Embedded
Quickstart tutorial

SQL Data Warehouse
Quickstart tutorial

Data Lake Storage Gen1
Quickstart tutorial

Data Factory
Quickstart tutorial

3. In the **New data factory** pane, provide values for the fields in the following table:

New data factory

* Name i

* Subscription

* Resource Group i
 Create new Use existing

Version i

* Location i

Integrate with GIT source control to do collaboration, source control, change tracking, change difference, continuous integration and deployment etc

Enable GIT i

* GIT URL i

* Repo name i

* Branch Name i

* Root folder i

Create [Automation options](#)

SETTING	VALUE TO ENTER
Name	In the box, enter a globally unique name for your data factory. If you receive an error, <i>Data factory name "LoadADXDemo" is not available</i> , enter a different name for the data factory. For rules about naming Data Factory artifacts, see Data Factory naming rules .
Subscription	In the drop-down list, select the Azure subscription in which to create the data factory.
Resource Group	Select Create new , and then enter the name of a new resource group. If you already have a resource group, select Use existing .
Version	In the drop-down list, select V2 .
Location	In the drop-down list, select the location for the data factory. Only supported locations are displayed in the list. The data stores that are used by the data factory can exist in other locations or regions.

4. Select **Create**.
5. To monitor the creation process, select **Notifications** on the toolbar. After you've created the data factory,

select it.

The **Data Factory** pane opens.

The screenshot shows the Azure Data Factory pane. At the top, there's a header bar with a factory icon and the text '<your factory name> Data factory'. Below the header, there's a 'Delete' button and a 'Essentials' section with a dropdown arrow. The 'Essentials' section contains the following information:

Resource group <your resource group>	Type Data factory (V2)
Location EastUS	Subscription name <your subscription name>
Provisioning state Succeeded	Subscription id <your subscription id>
Getting started Quick start	

Below this is a 'Quick links' section with a 'Documentation' tile and an 'Author & Monitor' tile, which is highlighted with a red box. To the right of the 'Quick links' section is a 'Monitoring' section titled 'PipelineRuns'.

Monitoring

PipelineRuns

8 AM

SUCCEEDED PIPELIN... ⓘ - | FAILED PIPELINE RU... ⓘ -

6. To open the application in a separate pane, select the **Author & Monitor** tile.

Load data into Azure Data Explorer

You can load data from many types of [data stores](#) into Azure Data Explorer. This article discusses how to load data from Amazon S3.

You can load your data in either of the following ways:

- In the Azure Data Factory user interface, in the left pane, select the **Author** icon, as shown in the "Create a data factory" section of [Create a data factory by using the Azure Data Factory UI](#).
- In the Azure Data Factory Copy Data tool, as shown in [Use the Copy Data tool to copy data](#).

Copy data from Amazon S3 (source)

1. In the **Let's get started** pane, open the Copy Data tool by selecting **Copy Data**.

Let's get started



Create pipeline



Create pipeline from template



Copy Data



Configure SSIS Integration



Set up Code Repository

2. In the **Properties** pane, in the **Task name** box, enter a name, and then select **Next**.

The screenshot shows the 'Copy Data' task configuration in the Azure Data Factory interface. The left sidebar lists steps 1 through 6. Step 1, 'Properties', is selected and highlighted with a red box around its title. The main pane displays the 'Properties' configuration screen, which includes fields for 'Task name' (set to 'CopyfromAmazonS3ToADX'), 'Task description', and 'Task cadence or Task schedule' (set to 'Run once now'). At the bottom right of the main pane, the 'Next' button is highlighted with a red box.

3. In the **Source data store** pane, select **Create new connection**.

Copy Data

1 Properties
One time copy

2 Source
Connection
Dataset

3 Destination
Connection
Dataset

4 Settings

5 Summary

6 Deployment

Source data store

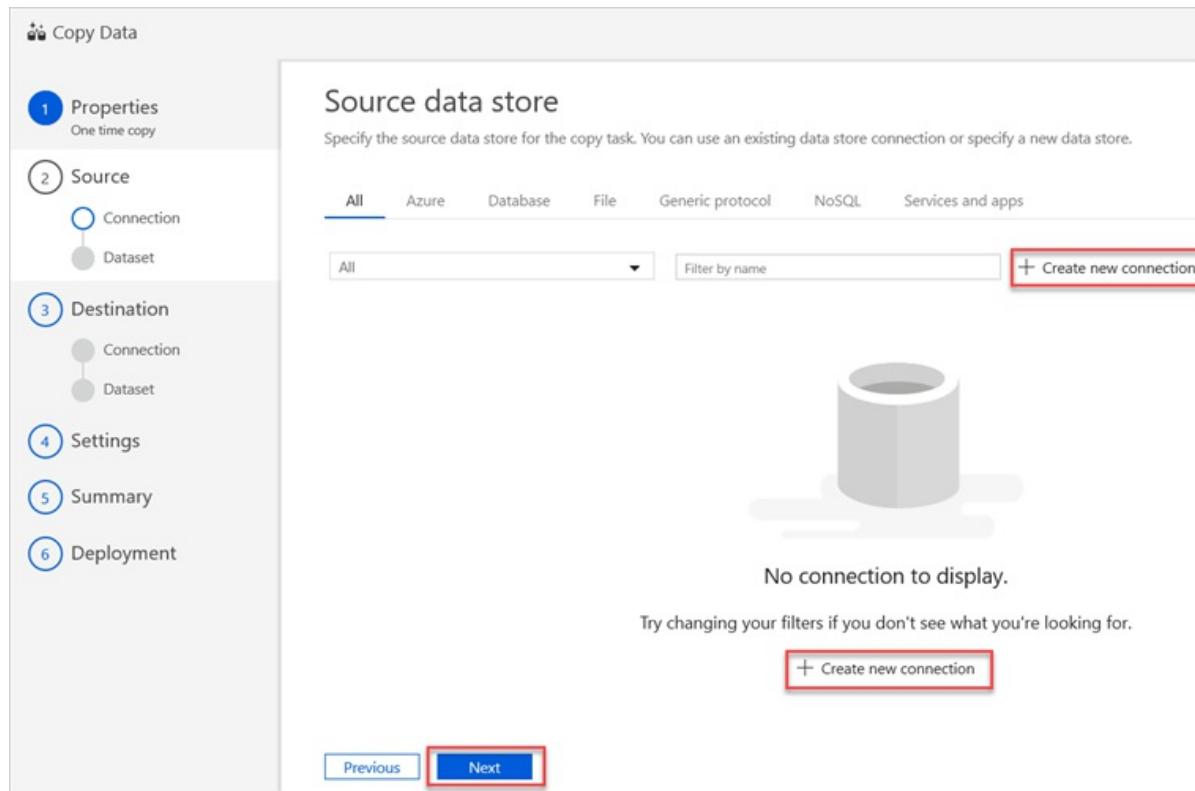
Specify the source data store for the copy task. You can use an existing data store connection or specify a new data store.

All Azure Database File Generic protocol NoSQL Services and apps

All Filter by name + Create new connection

No connection to display.
Try changing your filters if you don't see what you're looking for.
+ Create new connection

Previous Next



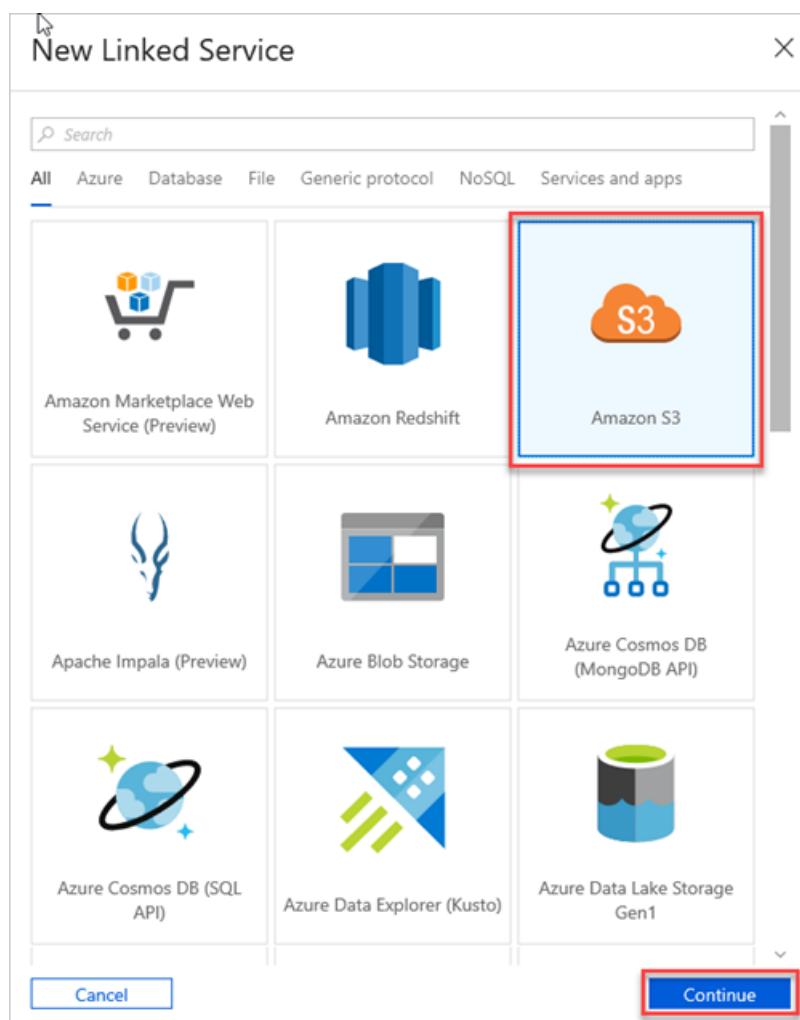
4. Select **Amazon S3**, and then select **Continue**.

New Linked Service

Search All Azure Database File Generic protocol NoSQL Services and apps

 Amazon Marketplace Web Service (Preview)	 Amazon Redshift	 Amazon S3
 Apache Impala (Preview)	 Azure Blob Storage	 Azure Cosmos DB (MongoDB API)
 Azure Cosmos DB (SQL API)	 Azure Data Explorer (Kusto)	 Azure Data Lake Storage Gen1

Cancel Continue



5. In the **New Linked Service (Amazon S3)** pane, do the following:

← New Linked Service (Amazon S3) X

Name *
AmazonS3_4ADX

Description

Connect via integration runtime *
AutoResolveIntegrationRuntime

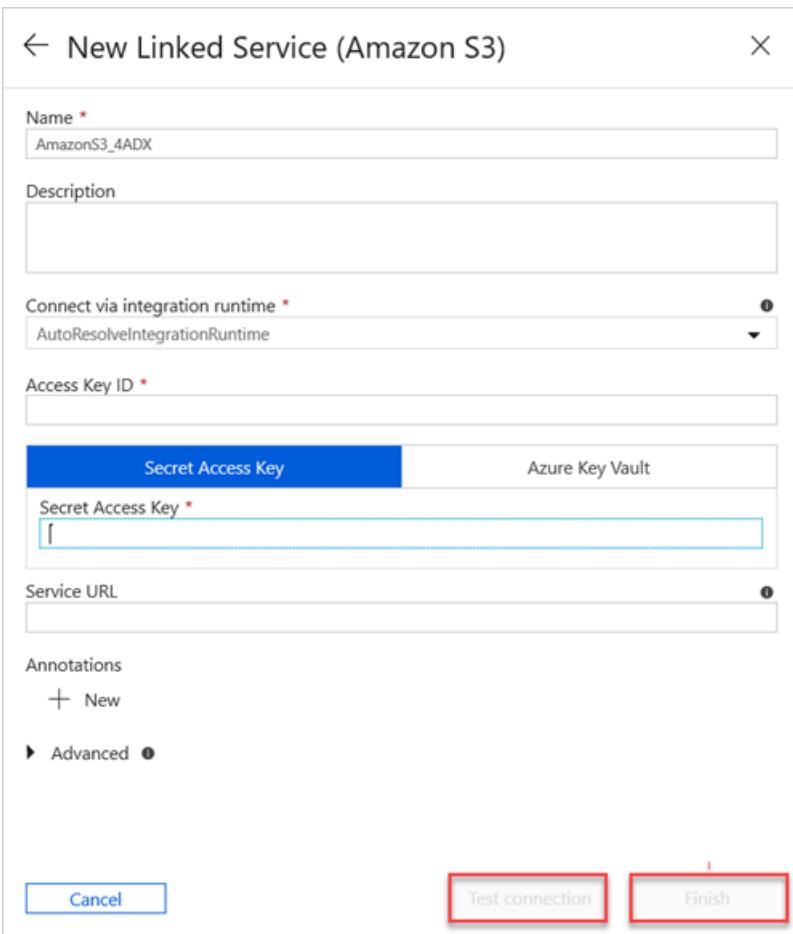
Access Key ID *

Secret Access Key Azure Key Vault
Secret Access Key *

Service URL

Annotations
+ New
► Advanced ⓘ

Cancel Test connection Finish



- a. In the **Name** box, enter the name of your new linked service.
- b. In the **Connect via integration runtime** drop-down list, select the value.
- c. In the **Access Key ID** box, enter the value.

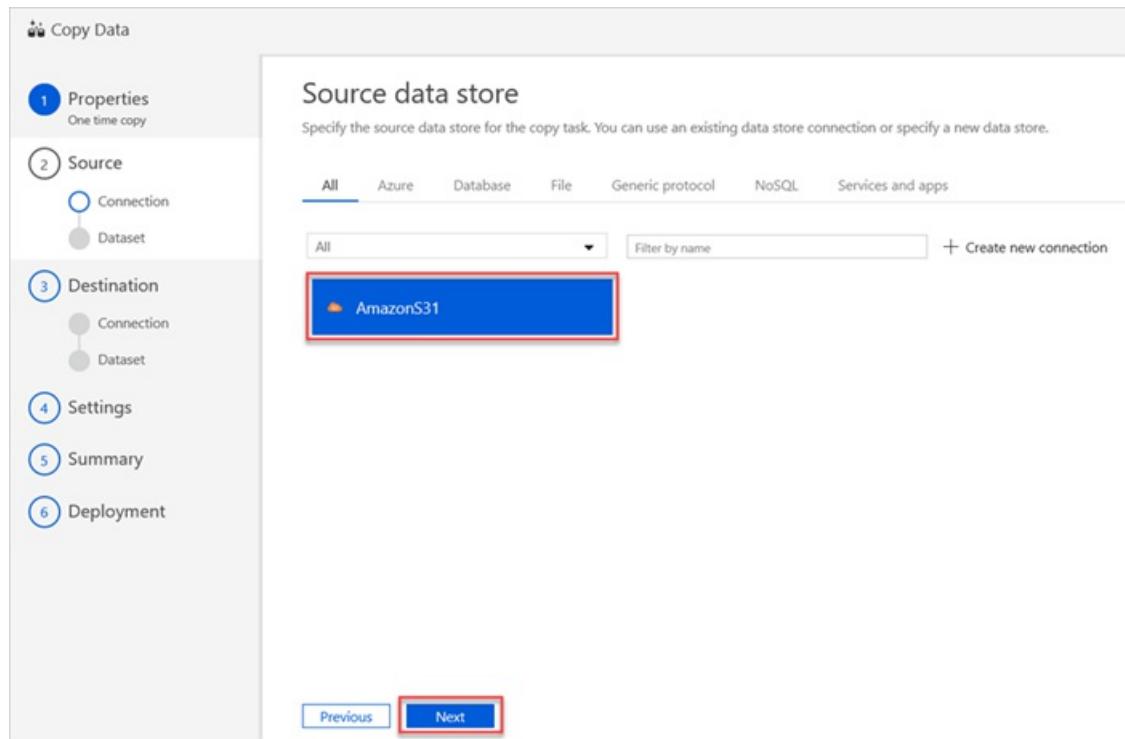
NOTE

In Amazon S3, to locate your access key, select your Amazon username on the navigation bar, and then select **My Security Credentials**.

- d. In the **Secret Access Key** box, enter a value.
- e. To test the linked service connection you created, select **Test Connection**.
- f. Select **Finish**.

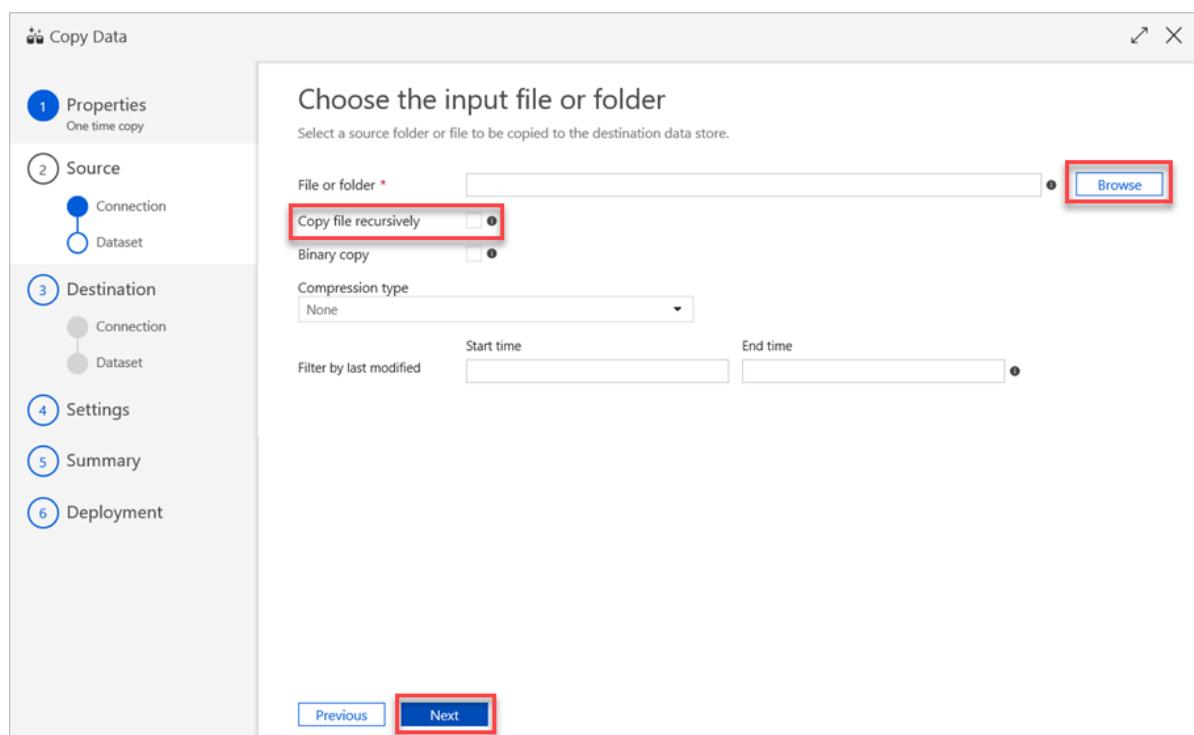
The **Source data store** pane displays your new AmazonS31 connection.

6. Select **Next**.

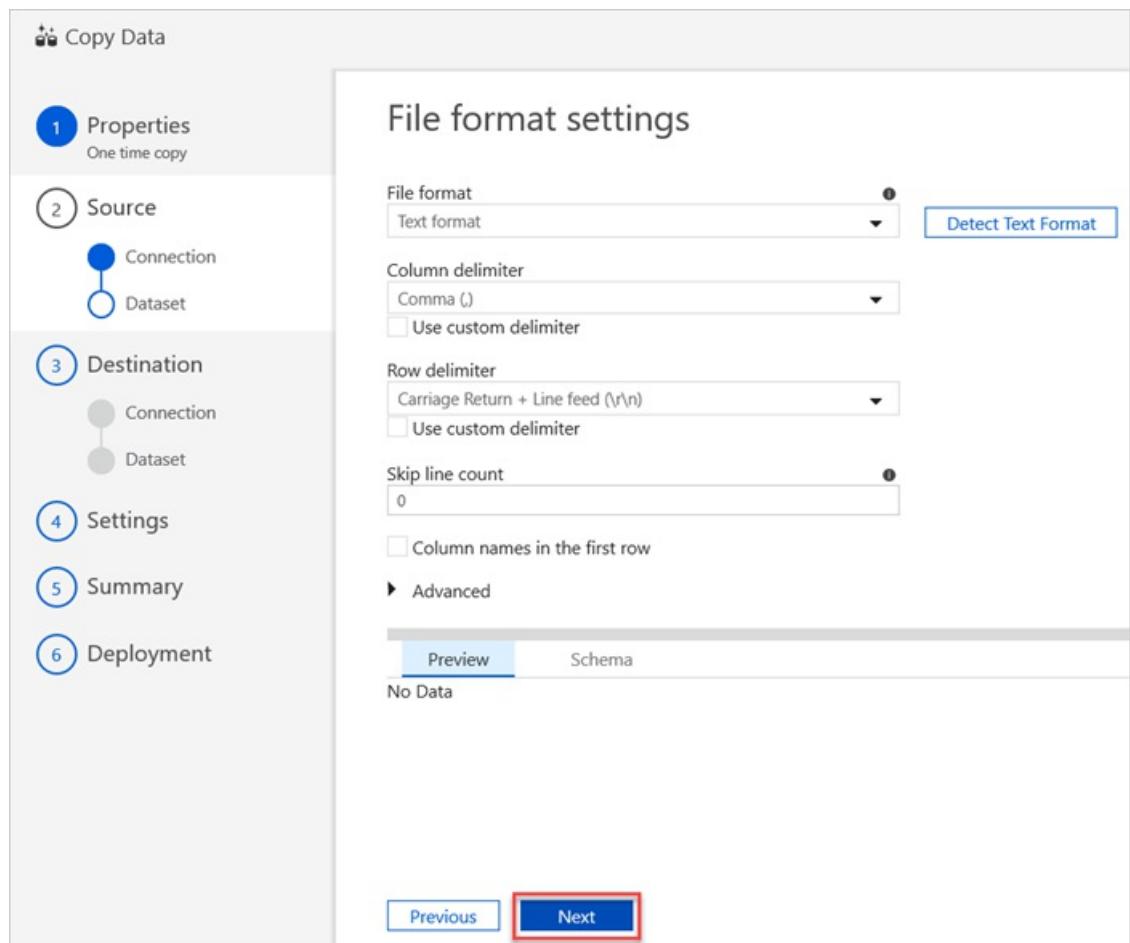


7. In the **Choose the input file or folder** pane, do the following:

- Browse to the file or folder that you want to copy, and then select it.
- Select the copy behavior that you want. Make sure that the **Binary copy** check box is cleared.
- Select **Next**.



8. In the **File format settings** pane, select the relevant settings for your file. and then select **Next**.



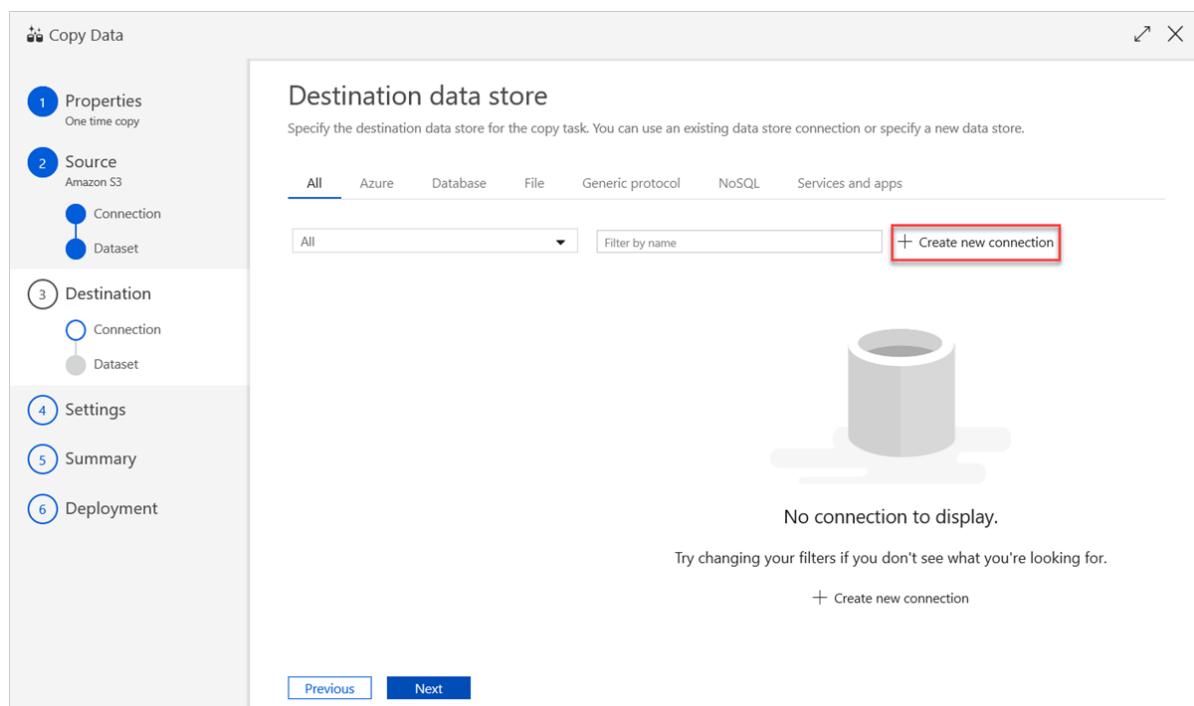
Copy data into Azure Data Explorer (destination)

The new Azure Data Explorer linked service is created to copy the data into the Azure Data Explorer destination table (sink) that's specified in this section.

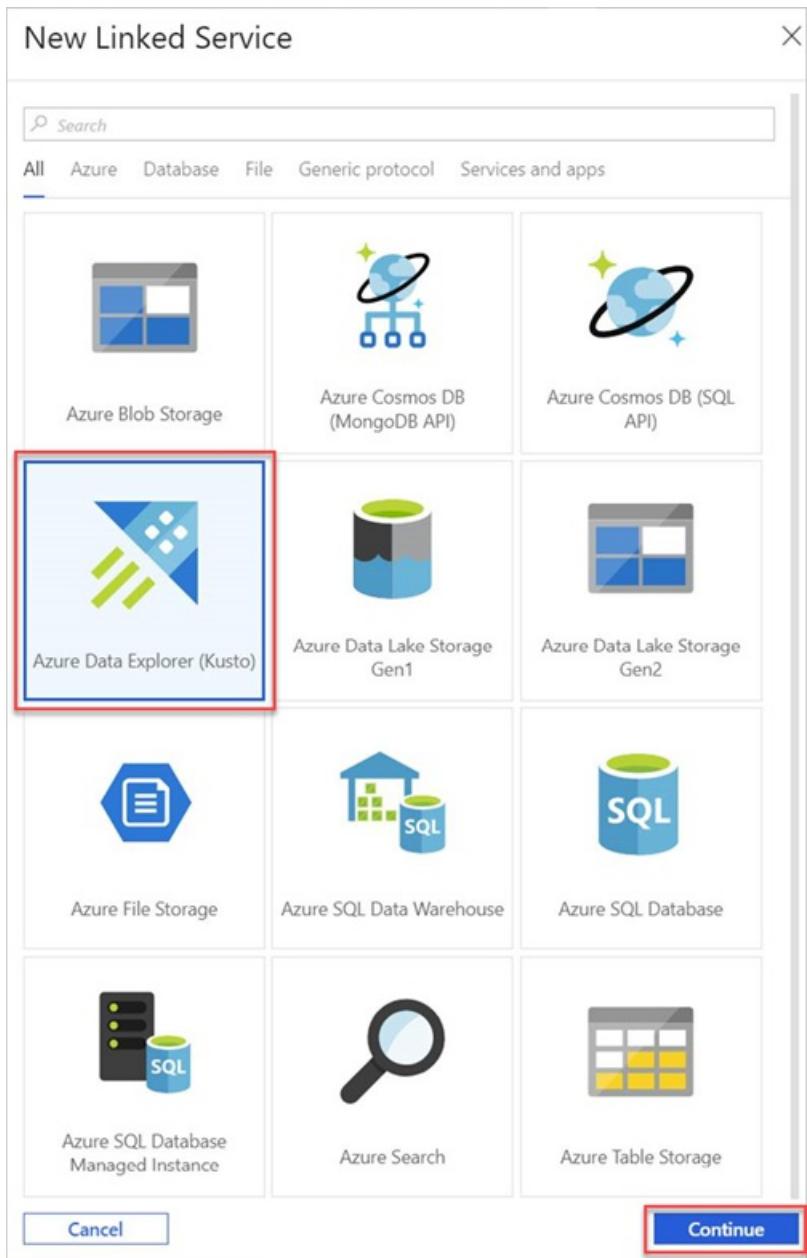
Create the Azure Data Explorer linked service

To create the Azure Data Explorer linked service, do the following:

1. To use an existing data store connection or specify a new data store, in the **Destination data store** pane, select **Create new connection**.



2. In the **New Linked Service** pane, select **Azure Data Explorer**, and then select **Continue**.



3. In the **New Linked Service (Azure Data Explorer)** pane, do the following:

← New Linked Service (Azure Data Explorer (Kusto)) X

Name *	AzureDataExplorer2				
Description					
Connect via integration runtime *	AutoResolveIntegrationRuntime				
Account selection method	<input checked="" type="radio"/> From Azure subscription <input type="radio"/> Enter manually				
Azure subscription	Select all				
Cluster *	No clusters found				
Tenant *					
Service principal ID *					
<table border="1"> <tr> <td style="background-color: #0078D4; color: white;">Service principal key</td> <td>Azure Key Vault</td> </tr> <tr> <td>Service principal key *</td> <td></td> </tr> </table>		Service principal key	Azure Key Vault	Service principal key *	
Service principal key	Azure Key Vault				
Service principal key *					
Database *	No Database				
<input type="checkbox"/> Edit					
Annotations	+ New				
► Advanced ●					
<input type="button" value="Cancel"/> <input type="button" value="Test connection"/> <input type="button" value="Finish"/>					

a. In the **Name** box, enter a name for the Azure Data Explorer linked service.

b. Under **Account selection method**, do one of the following:

- Select **From Azure subscription** and then, in the drop-down lists, select your **Azure subscription** and your **Cluster**.

NOTE

The **Cluster** drop-down control lists only clusters that are associated with your subscription.

- Select **Enter manually**, and then enter your **Endpoint**.

c. In the **Tenant** box, enter the tenant name.

d. In the **Service principal ID** box, enter the service principal ID.

e. Select **Service principal key** and then, in the **Service principal key** box, enter the value for the key.

f. In the **Database** drop-down list, select your database name. Alternatively, select the **Edit** check box, and then enter the database name.

g. To test the linked service connection you created, select **Test Connection**. If you can connect to your linked service, the pane displays a green checkmark and a **Connection successful** message.

h. Select **Finish** to complete the linked service creation.

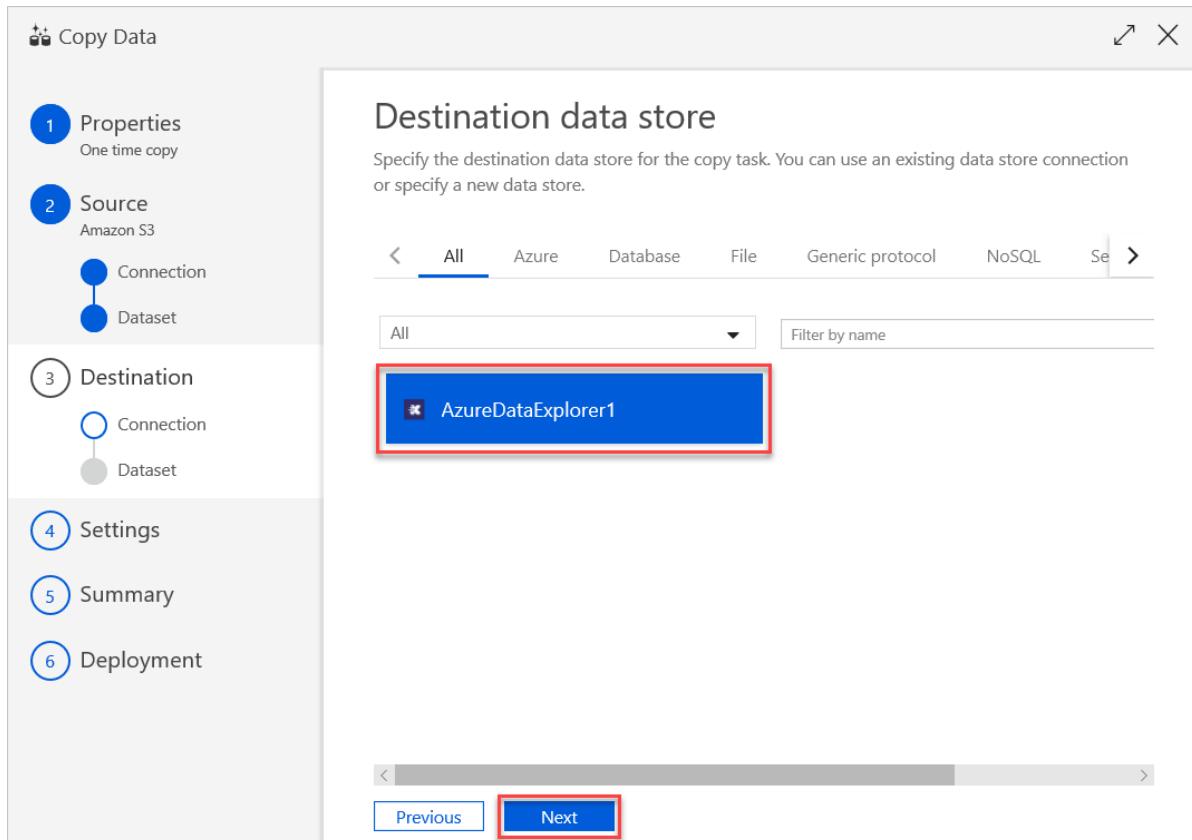
NOTE

The service principal is used by Azure Data Factory to access the Azure Data Explorer service. To create a service principal, go to [create an Azure Active Directory \(Azure AD\) service principal](#). Do not use the Azure Key Vault method.

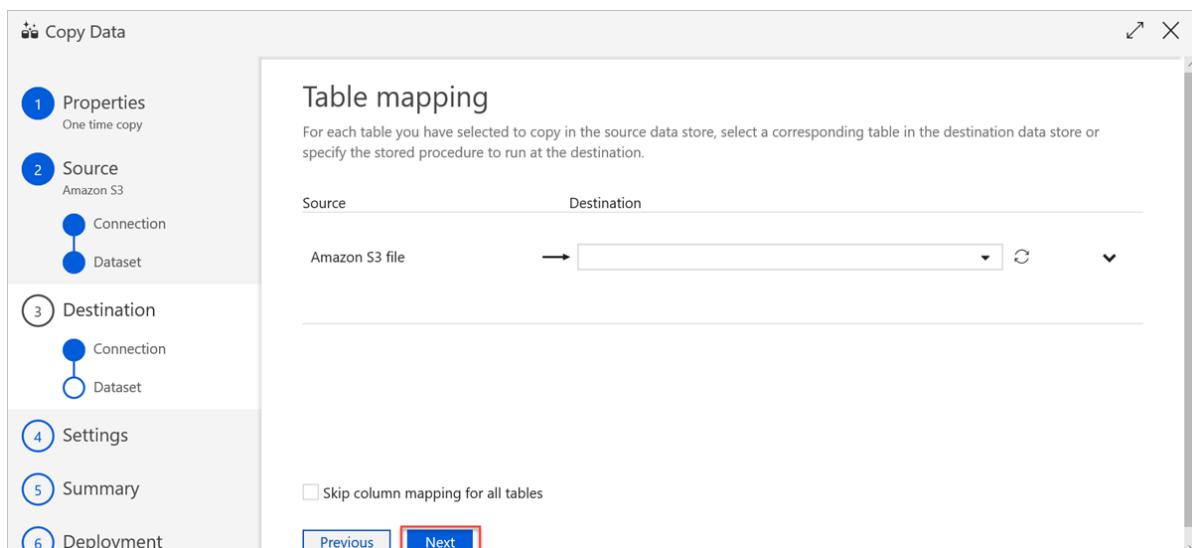
Configure the Azure Data Explorer data connection

After you've created the linked service connection, the **Destination data store** pane opens, and the connection you created is available for use. To configure the connection, do the following:

1. Select **Next**.



2. In the **Table mapping** pane, set the destination table name, and then select **Next**.



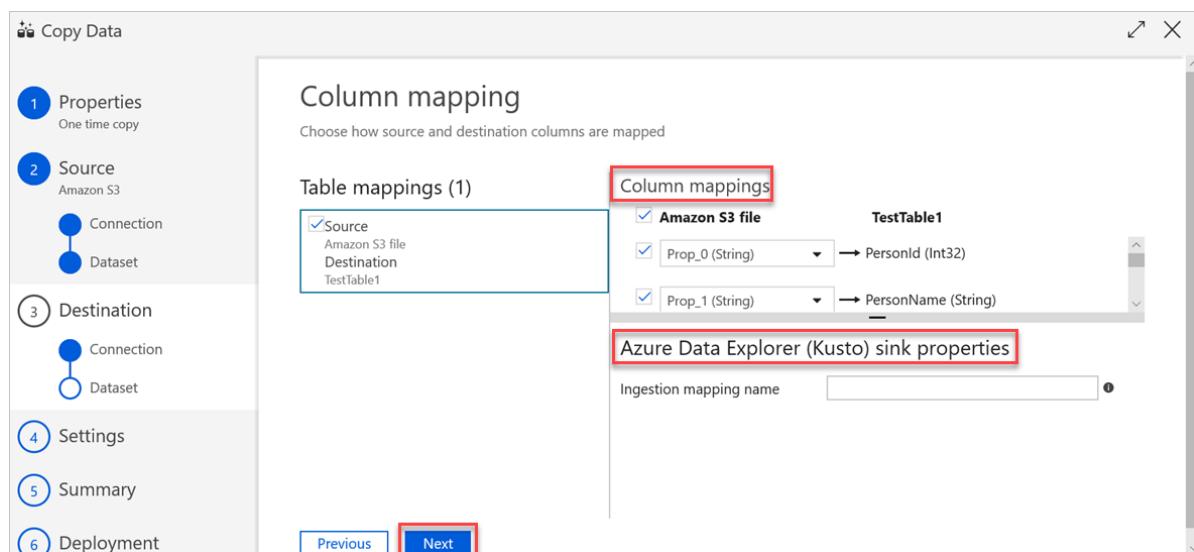
3. In the **Column mapping** pane, the following mappings take place:

a. The first mapping is performed by Azure Data Factory according to the [Azure Data Factory schema mapping](#). Do the following:

- Set the **Column mappings** for the Azure Data Factory destination table. The default mapping is displayed from source to the Azure Data Factory destination table.
- Cancel the selection of the columns that you don't need to define your column mapping.

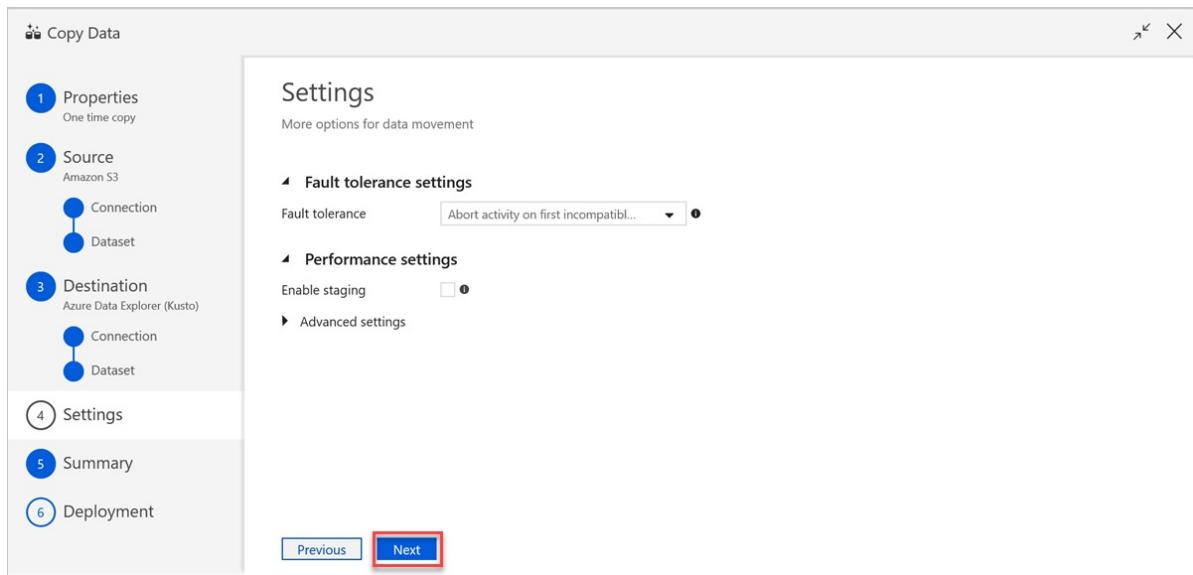
b. The second mapping occurs when this tabular data is ingested into Azure Data Explorer. Mapping is performed according to [CSV mapping rules](#). Even if the source data isn't in CSV format, Azure Data Factory converts the data into a tabular format. Therefore, CSV mapping is the only relevant mapping at this stage. Do the following:

- (Optional) Under **Azure Data Explorer (Kusto) sink properties**, add the relevant **Ingestion mapping name** so that column mapping can be used.
- If **Ingestion mapping name** isn't specified, the *by-name* mapping order that's defined in the **Column mappings** section will be used. If *by-name* mapping fails, Azure Data Explorer tries to ingest the data in a *by-column position* order (that is, it maps by position as the default).
- Select **Next**.

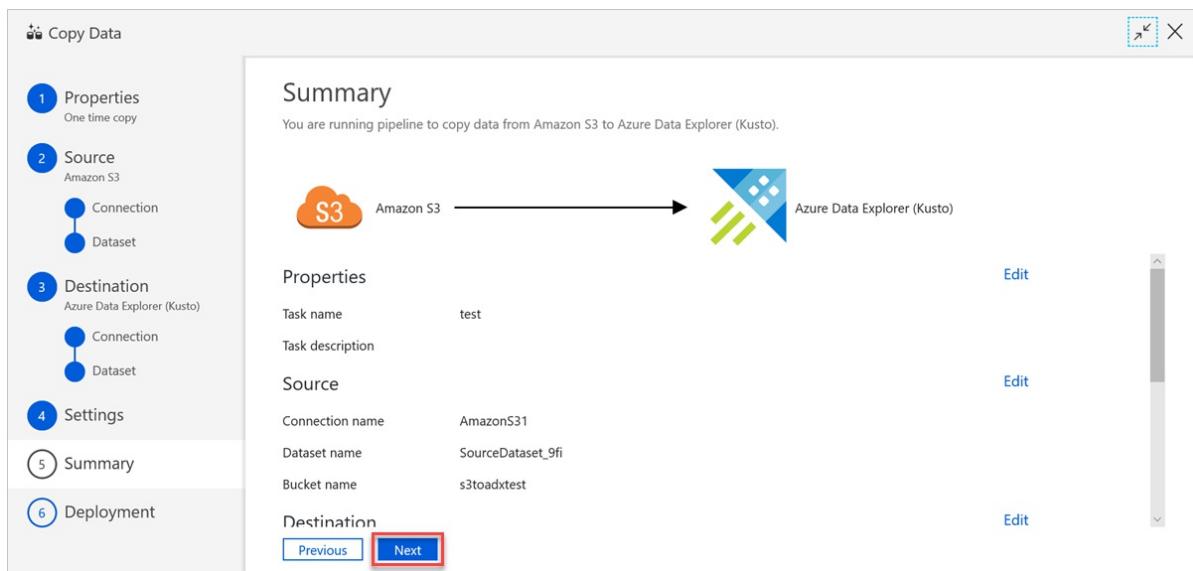


4. In the **Settings** pane, do the following:

- Under **Fault tolerance settings**, enter the relevant settings.
- Under **Performance settings**, **Enable staging** doesn't apply, and **Advanced settings** includes cost considerations. If you have no specific requirements, leave these settings as is.
- Select **Next**.

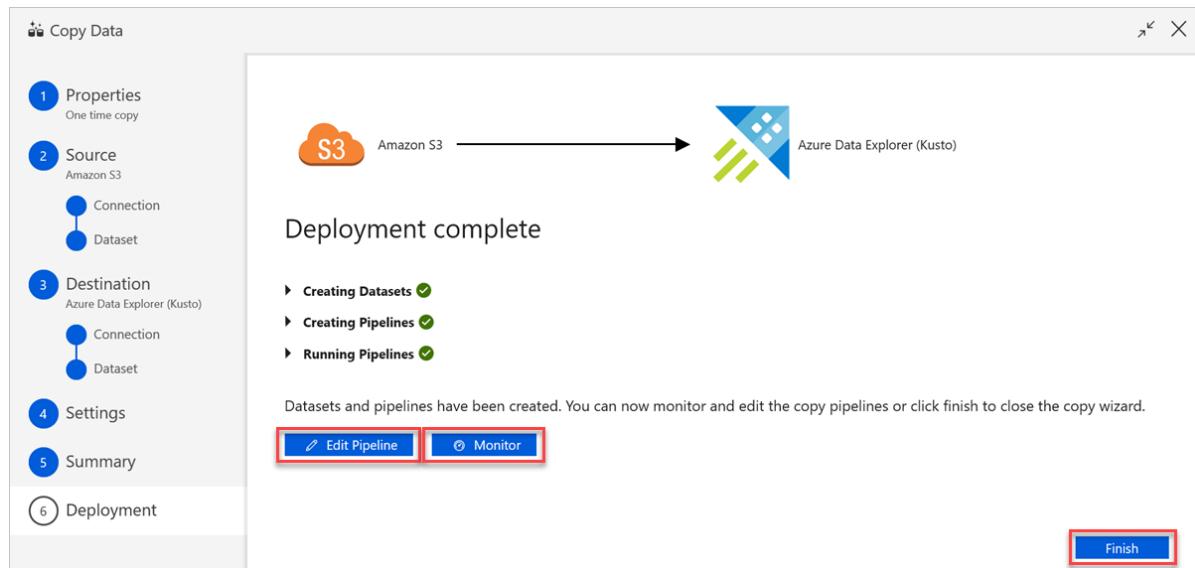


5. In the **Summary** pane, review the settings, and then select **Next**.



6. In the **Deployment complete** pane, do the following:

- To switch to the **Monitor** tab and view the status of the pipeline (that is, progress, errors, and data flow), select **Monitor**.
- To edit linked services, datasets, and pipelines, select **Edit Pipeline**.
- Select **Finish** to complete the copy data task.



Next steps

- Learn about the [Azure Data Explorer connector](#) in Azure Data Factory.
- Learn more about editing linked services, datasets, and pipelines in the [Data Factory UI](#).
- Learn about [Azure Data Explorer queries](#) for data querying.

Write queries for Azure Data Explorer

8/8/2019 • 22 minutes to read • [Edit Online](#)

In this article, you learn how to use the query language in Azure Data Explorer to perform basic queries with the most common operators. You also get exposure to some of the more advanced features of the language.

Prerequisites

You can run the queries in this article in one of two ways:

- On the Azure Data Explorer *help cluster* that we have set up to aid learning. [Sign in to the cluster](#) with an organizational email account that is a member of Azure Active directory.
- On your own cluster that includes the StormEvents sample data. For more information, see [Quickstart: Create an Azure Data Explorer cluster and database](#) and [Ingest sample data into Azure Data Explorer](#).

The StormEvents sample data set contains weather-related data from the [National Centers for Environmental Information](#).

Overview of the query language

A query language in Azure Data Explorer is a read-only request to process data and return results. The request is stated in plain text, using a data-flow model designed to make the syntax easy to read, author, and automate. The query uses schema entities that are organized in a hierarchy similar to SQL: databases, tables, and columns.

The query consists of a sequence of query statements, delimited by a semicolon (;), with at least one statement being a tabular expression statement, which is a statement that produces data arranged in a table-like mesh of columns and rows. The query's tabular expression statements produce the results of the query.

The syntax of the tabular expression statement has tabular data flow from one tabular query operator to another, starting with data source (for example, a table in a database, or an operator that produces data) and then flowing through a set of data transformation operators that are bound together through the use of the pipe (|) delimiter.

For example, the following query has a single statement, which is a tabular expression statement. The statement starts with a reference to a table called `StormEvents` (the database that host this table is implicit here, and part of the connection information). The data (rows) for that table are then filtered by the value of the `StartTime` column, and then filtered by the value of the `State` column. The query then returns the count of "surviving" rows.

[\[Click to run query\]](#)

```
StormEvents
| where StartTime >= datetime(2007-11-01) and StartTime < datetime(2007-12-01)
| where State == "FLORIDA"
| count
```

In this case, the result is:

COUNT

```
23
```

For more information see the [Query language reference](#).

Most common operators

The operators covered in this section are the building blocks to understanding queries in Azure Data Explorer. Most queries you write will include several of these operators.

To run queries on the help cluster: select **Click to run query** above each query.

To run queries on your own cluster:

1. Copy each query into the web-based query application, and then either select the query or place your cursor in the query.
2. At the top of the application, select **Run**.

count

count: Returns the count of rows in the table.

The following query returns the count of rows in the StormEvents table.

[Click to run query]

```
StormEvents | count
```

take

take: Returns up to the specified number of rows of data.

The following query returns five rows from the StormEvents table. The keyword *limit* is an alias for *take*.

[Click to run query]

```
StormEvents | take 5
```

TIP

There is no guarantee which records are returned unless the source data is sorted.

project

project: Selects a subset of columns.

The following query returns a specific set of columns.

[Click to run query]

```
StormEvents  
| take 5  
| project StartTime, EndTime, State, EventType, DamageProperty, EpisodeNarrative
```

where

where: Filters a table to the subset of rows that satisfy a predicate.

The following query filters the data by `EventType` and `State`.

[\[Click to run query\]](#)

```
StormEvents
| where EventType == 'Flood' and State == 'WASHINGTON'
| take 5
| project StartTime, EndTime, State, EventType, DamageProperty, EpisodeNarrative
```

sort

sort: Sorts the rows of the input table into order by one or more columns.

The following query sorts the data in descending order by `DamageProperty`.

[\[Click to run query\]](#)

```
StormEvents
| where EventType == 'Flood' and State == 'WASHINGTON'
| sort by DamageProperty desc
| take 5
| project StartTime, EndTime, State, EventType, DamageProperty, EpisodeNarrative
```

NOTE

The order of operations is important. Try putting `take 5` before `sort by`. Do you get different results?

top

top: Returns the first N records sorted by the specified columns.

The following query returns the same results as above with one less operator.

[\[Click to run query\]](#)

```
StormEvents
| where EventType == 'Flood' and State == 'WASHINGTON'
| top 5 by DamageProperty desc
| project StartTime, EndTime, State, EventType, DamageProperty, EpisodeNarrative
```

extend

extend: Computes derived columns.

The following query creates a new column by computing a value in every row.

[\[Click to run query\]](#)

```
StormEvents
| where EventType == 'Flood' and State == 'WASHINGTON'
| top 5 by DamageProperty desc
| extend Duration = EndTime - StartTime
| project StartTime, EndTime, Duration, State, EventType, DamageProperty, EpisodeNarrative
```

Expressions can include all the usual operators ($+$, $-$, $*$, $/$, $\%$), and there's a range of useful functions that you can call.

summarize

summarize: Aggregates groups of rows.

The following query returns the count of events by `state`.

[\[Click to run query\]](#)

```
StormEvents
| summarize event_count = count() by State
```

The **summarize** operator groups together rows that have the same values in the **by** clause, and then uses the aggregation function (such as **count**) to combine each group into a single row. So, in this case, there's a row for each state, and a column for the count of rows in that state.

There's a range of aggregation functions, and you can use several of them in one **summarize** operator to produce several computed columns. For example, you could get the count of storms in each state and the unique number of storms per state, then use **top** to get the most storm-affected states.

[\[Click to run query\]](#)

```
StormEvents
| summarize StormCount = count(), TypeOfStorms = dcount(EventType) by State
| top 5 by StormCount desc
```

The result of a **summarize** operation has:

- Each column named in **by**
- A column for each computed expression
- A row for each combination of **by** values

render

render: Renders results as a graphical output.

The following query displays a column chart.

[\[Click to run query\]](#)

```
StormEvents
| summarize event_count=count(), mid = avg(BeginLat) by State
| sort by mid
| where event_count > 1800
| project State, event_count
| render columnchart
```

The following query displays a simple time chart.

[\[Click to run query\]](#)

```
StormEvents
| summarize event_count=count() by bin(StartTime, 1d)
| render timechart
```

The following query counts events by the time modulo one day, binned into hours, and displays a time chart.

[\[Click to run query\]](#)

```
StormEvents
| extend hour = floor(StartTime % 1d , 1h)
| summarize event_count=count() by hour
| sort by hour asc
| render timechart
```

The following query compares multiple daily series on a time chart.

[\[Click to run query\]](#)

```
StormEvents
| extend hour= floor( StartTime % 1d , 1h)
| where State in ("GULF OF MEXICO","MAINE","VIRGINIA","WISCONSIN","NORTH DAKOTA","NEW JERSEY","OREGON")
| summarize event_count=count() by hour, State
| render timechart
```

NOTE

The **render** operator is a client-side feature rather than part of the engine. It's integrated into the language for ease of use. The web application supports the following options: barchart, columnchart, piechart, timechart, and linechart.

Scalar operators

This section covers some of the most important scalar operators.

bin()

bin(): Rounds values down to an integer multiple of a given bin size.

The following query calculates the count with a bucket size of one day.

[\[Click to run query\]](#)

```
StormEvents
| where StartTime > datetime(2007-02-14) and StartTime < datetime(2007-02-21)
| summarize event_count = count() by bin(StartTime, 1d)
```

case()

case(): Evaluates a list of predicates, and returns the first result expression whose predicate is satisfied, or the final **else** expression. You can use this operator to categorize or group data:

The following query returns a new column `deaths_bucket` and groups the deaths by number.

[\[Click to run query\]](#)

```
StormEvents
| summarize deaths = sum(DeathsDirect) by State
| extend deaths_bucket = case (
    deaths > 50, "large",
    deaths > 10, "medium",
    deaths > 0, "small",
    "N/A")
| sort by State asc
```

extract()

extract(): Gets a match for a regular expression from a text string.

The following query extracts specific attribute values from a trace.

[\[Click to run query\]](#)

```
let MyData = datatable (Trace: string) ["A=1, B=2, Duration=123.45, ...", "A=1, B=5, Duration=55.256, ..."];
MyData
| extend Duration = extract("Duration=([0-9.]+)", 1, Trace, typeof(real)) * time(1s)
```

This query uses a **let** statement, which binds a name (in this case `MyData`) to an expression. For the rest of the scope, in which the **let** statement appears (global scope or in a function body scope), the name can be used to refer to its bound value.

parse_json()

parse_json(): Interprets a string as a JSON value, and returns the value as dynamic. It is superior to using the **extractjson()** function when you need to extract more than one element of a compound JSON object.

The following query extracts the JSON elements from an array.

[\[Click to run query\]](#)

```
let MyData = datatable (Trace: string)
[ '{"duration": [{"value":118.0,"valcount":5.0,"min":100.0,"max":150.0,"stdDev":0.0}]} ];
MyData
| extend NewCol = parse_json(Trace)
| project NewCol.duration[0].value, NewCol.duration[0].valcount, NewCol.duration[0].min,
NewCol.duration[0].max, NewCol.duration[0].stdDev
```

The following query extracts the JSON elements.

[\[Click to run query\]](#)

```
let MyData = datatable (Trace: string)
[ '{"value":118.0,"valcount":5.0,"min":100.0,"max":150.0,"stdDev":0.0}' ];
MyData
| extend NewCol = parse_json(Trace)
| project NewCol.value, NewCol.valcount, NewCol.min, NewCol.max, NewCol.stdDev
```

The following query extracts the JSON elements with a dynamic data type.

[\[Click to run query\]](#)

```
let MyData = datatable (Trace: dynamic)
[dynamic({ "value":118.0,"counter":5.0,"min":100.0,"max":150.0,"stdDev":0.0})];
MyData
| project Trace.value, Trace.counter, Trace.min, Trace.max, Trace.stdDev
```

ago()

ago(): Subtracts the given timespan from the current UTC clock time.

The following query returns data for the last 12 hours.

[\[Click to run query\]](#)

```
//The first two lines generate sample data, and the last line uses  
//the ago() operator to get records for last 12 hours.  
print TimeStamp= range(now(-5d), now(), 1h), SomeCounter = range(1,121)  
| mv-expand TimeStamp, SomeCounter  
| where TimeStamp > ago(12h)
```

startofweek()

startofweek(): Returns the start of the week containing the date, shifted by an offset, if provided

The following query returns the start of the week with different offsets.

[\[Click to run query\]](#)

```
range offset from -1 to 1 step 1  
| project weekStart = startofweek(now(), offset), offset
```

This query uses the **range** operator, which generates a single-column table of values. See also: [startofday\(\)](#), [startofweek\(\)](#), [startofyear\(\)](#), [startofmonth\(\)](#), [endofday\(\)](#), [endofweek\(\)](#), [endofmonth\(\)](#), and [endofyear\(\)](#).

between()

between(): Matches the input that is inside the inclusive range.

The following query filters the data by a given date range.

[\[Click to run query\]](#)

```
StormEvents  
| where StartTime between (datetime(2007-07-27) .. datetime(2007-07-30))  
| count
```

The following query filters the data by a given date range, with the slight variation of three days (`3d`) from the start date.

[\[Click to run query\]](#)

```
StormEvents  
| where StartTime between (datetime(2007-07-27) .. 3d)  
| count
```

Tabular operators

Kusto has many tabular operators, some of which are covered in other sections of this article. Here we'll focus on **parse**.

parse

parse: Evaluates a string expression and parses its value into one or more calculated columns. There are three ways to parse: simple (the default), regex, and relaxed.

The following query parses a trace and extracts the relevant values, using a default of simple parsing. The expression (referred to as StringConstant) is a regular string value and the match is strict: extended columns must match the required types.

[\[Click to run query\]](#)

```

let MyTrace = datatable (EventTrace:string)
[
'Event: NotifySliceRelease (resourceName=PipelineScheduler, totalSlices=27, sliceNumber=23,
lockTime=02/17/2016 08:40:01Z, releaseTime=02/17/2016 08:40:01Z, previousLockTime=02/17/2016 08:39:01Z)',
'Event: NotifySliceRelease (resourceName=PipelineScheduler, totalSlices=27, sliceNumber=15,
lockTime=02/17/2016 08:40:00Z, releaseTime=02/17/2016 08:40:00Z, previousLockTime=02/17/2016 08:39:00Z)',
'Event: NotifySliceRelease (resourceName=PipelineScheduler, totalSlices=27, sliceNumber=20,
lockTime=02/17/2016 08:40:01Z, releaseTime=02/17/2016 08:40:01Z, previousLockTime=02/17/2016 08:39:01Z)',
'Event: NotifySliceRelease (resourceName=PipelineScheduler, totalSlices=27, sliceNumber=22,
lockTime=02/17/2016 08:41:01Z, releaseTime=02/17/2016 08:41:00Z, previousLockTime=02/17/2016 08:40:01Z),
'Event: NotifySliceRelease (resourceName=PipelineScheduler, totalSlices=27, sliceNumber=16,
lockTime=02/17/2016 08:41:00Z, releaseTime=02/17/2016 08:41:00Z, previousLockTime=02/17/2016 08:40:00Z)'
];
MyTrace
| parse EventTrace with * "resourceName=" resourceName ", totalSlices=" totalSlices:long * "sliceNumber=" sliceNumber:long * "lockTime=" lockTime ", releaseTime=" releaseTime:date ", * "previousLockTime=" previousLockTime:date ")*"
| project resourceName ,totalSlices , sliceNumber , lockTime , releaseTime , previousLockTime

```

The following query parses a trace and extracts the relevant values, using `kind = regex`. The StringConstant can be a regular expression.

[\[Click to run query\]](#)

```

let MyTrace = datatable (EventTrace:string)
[
'Event: NotifySliceRelease (resourceName=PipelineScheduler, totalSlices=27, sliceNumber=23,
lockTime=02/17/2016 08:40:01Z, releaseTime=02/17/2016 08:40:01Z, previousLockTime=02/17/2016 08:39:01Z)',
'Event: NotifySliceRelease (resourceName=PipelineScheduler, totalSlices=27, sliceNumber=15,
lockTime=02/17/2016 08:40:00Z, releaseTime=02/17/2016 08:40:00Z, previousLockTime=02/17/2016 08:39:00Z),
'Event: NotifySliceRelease (resourceName=PipelineScheduler, totalSlices=27, sliceNumber=20,
lockTime=02/17/2016 08:40:01Z, releaseTime=02/17/2016 08:40:01Z, previousLockTime=02/17/2016 08:39:01Z),
'Event: NotifySliceRelease (resourceName=PipelineScheduler, totalSlices=27, sliceNumber=22,
lockTime=02/17/2016 08:41:01Z, releaseTime=02/17/2016 08:41:00Z, previousLockTime=02/17/2016 08:40:01Z),
'Event: NotifySliceRelease (resourceName=PipelineScheduler, totalSlices=27, sliceNumber=16,
lockTime=02/17/2016 08:41:00Z, releaseTime=02/17/2016 08:41:00Z, previousLockTime=02/17/2016 08:40:00Z)'
];
MyTrace
| parse kind = regex EventTrace with "(.*?[a-zA-Z]*=" resourceName @", totalSlices=\s*\d+\s*.*?
sliceNumber=" sliceNumber:long ".*?(previous)?lockTime=" lockTime ".*?releaseTime=" releaseTime ".*?
previousLockTime=" previousLockTime:date "\\\")"
| project resourceName , sliceNumber , lockTime , releaseTime , previousLockTime

```

The following query parses a trace and extracts the relevant values, using `kind = relaxed`. The StringConstant is a regular string value and the match is relaxed: extended columns can partially match the required types.

[\[Click to run query\]](#)

```

let MyTrace = datatable (EventTrace:string)
[
    'Event: NotifySliceRelease (resourceName=PipelineScheduler, totalSlices=27, sliceNumber=23,
lockTime=02/17/2016 08:40:01Z, releaseTime=02/17/2016 08:40:01Z, previousLockTime=02/17/2016 08:39:01Z)',
    'Event: NotifySliceRelease (resourceName=PipelineScheduler, totalSlices=27, sliceNumber=15,
lockTime=02/17/2016 08:40:00Z, releaseTime=02/17/2016 08:40:00Z, previousLockTime=02/17/2016 08:39:00Z)',
    'Event: NotifySliceRelease (resourceName=PipelineScheduler, totalSlices=27, sliceNumber=20,
lockTime=02/17/2016 08:40:01Z, releaseTime=02/17/2016 08:40:01Z, previousLockTime=02/17/2016 08:39:01Z)',
    'Event: NotifySliceRelease (resourceName=PipelineScheduler, totalSlices=27, sliceNumber=22,
lockTime=02/17/2016 08:41:01Z, releaseTime=02/17/2016 08:41:00Z, previousLockTime=02/17/2016 08:40:01Z)',
    'Event: NotifySliceRelease (resourceName=PipelineScheduler, totalSlices=27, sliceNumber=16,
lockTime=02/17/2016 08:41:00Z, releaseTime=02/17/2016 08:41:00Z, previousLockTime=02/17/2016 08:40:00Z)'
];
MyTrace
| parse kind=relaxed "Event: NotifySliceRelease (resourceName=PipelineScheduler, totalSlices=NULL,
sliceNumber=23, lockTime=02/17/2016 08:40:01, releaseTime=NULL, previousLockTime=02/17/2016 08:39:01)" with
* "resourceName=" resourceName ", totalSlices=" totalSlices:long * "sliceNumber=" sliceNumber:long *
"lockTime=" lockTime ", releaseTime=" releaseTime:date "," * "previousLockTime=" previousLockTime:date ")"
*
| project resourceName ,totalSlices , sliceNumber , lockTime , releaseTime , previousLockTime

```

Time series analysis

make-series

make-series: aggregates together groups of rows like [summarize](#), but generates a (time) series vector per each combination of by values.

The following query returns a set of time series for the count of storm events per day. The query covers a three-month period for each state, filling missing bins with the constant 0:

[\[Click to run query\]](#)

```

StormEvents
| make-series n=count() default=0 on StartTime in range(datetime(2007-01-01), datetime(2007-03-31), 1d) by
State

```

Once you create a set of (time) series, you can apply series functions to detect anomalous shapes, seasonal patterns, and a lot more.

The following query extracts the top three states that had the most events in specific day:

[\[Click to run query\]](#)

```

StormEvents
| make-series n=count() default=0 on StartTime in range(datetime(2007-01-01), datetime(2007-03-31), 1d) by
State
| extend series_stats(n)
| top 3 by series_stats_n_max desc
| render timechart

```

For more information, review the full list of [series functions](#).

Advanced aggregations

We covered basic aggregations, like **count** and **summarize**, earlier in this article. This section introduces more advanced options.

top-nested

top-nested: Produces hierarchical top results, where each level is a drill-down based on previous level values.

This operator is useful for dashboard visualization scenarios, or when it is necessary to answer a question like the following: "Find the top-N values of K1 (using some aggregation); for each of them, find what are the top-M values of K2 (using another aggregation); ..."

The following query returns a hierarchical table with `State` at the top level, followed by `Sources`.

[\[Click to run query\]](#)

```
StormEvents
| top-nested 2 of State by sum(BeginLat),
  top-nested 3 of Source by sum(BeginLat),
    top-nested 1 of EndLocation by sum(BeginLat)
```

pivot() plugin

pivot() plugin: Rotates a table by turning the unique values from one column in the input table into multiple columns in the output table. The operator performs aggregations where they are required on any remaining column values in the final output.

The following query applies a filter and pivots the rows into columns.

[\[Click to run query\]](#)

```
StormEvents
| project State, EventType
| where State startswith "AL"
| where EventType has "Wind"
| evaluate pivot(State)
```

dcount()

dcount(): Returns an estimate of the number of distinct values of an expression in the group. Use `count()` to count all values.

The following query counts distinct `Source` by `State`.

[\[Click to run query\]](#)

```
StormEvents
| summarize Sources = dcount(Source) by State
```

dcountif()

dcountif(): Returns an estimate of the number of distinct values of the expression for rows for which the predicate evaluates to true.

The following query counts the distinct values of `Source` where `DamageProperty < 5000`.

[\[Click to run query\]](#)

```
StormEvents
| take 100
| summarize Sources = dcountif(Source, DamageProperty < 5000) by State
```

dcount_hll()

dcount_hll(): Calculates the `dcount` from HyperLogLog results (generated by `hll` or `hll_merge`).

The following query uses the HLL algorithm to generate the count.

[Click to run query]

```
StormEvents
| summarize hllRes = hll(DamageProperty) by bin(StartTime,10m)
| summarize hllMerged = hll_merge(hllRes)
| project dcount_hll(hllMerged)
```

arg_max()

arg_max(): Finds a row in the group that maximizes an expression, and returns the value of another expression (or * to return the entire row).

The following query returns the time of the last flood report in each state.

[Click to run query]

```
StormEvents
| where EventType == "Flood"
| summarize arg_max(StartTime, *) by State
| project State, StartTime, EndTime, EventType
```

makeset()

makeset(): Returns a dynamic (JSON) array of the set of distinct values that an expression takes in the group.

The following query returns all the times when a flood was reported by each state and creates an array from the set of distinct values.

[Click to run query]

```
StormEvents
| where EventType == "Flood"
| summarize FloodReports = makeset(StartTime) by State
| project State, FloodReports
```

mv-expand

mv-expand: Expands multi-value collection(s) from a dynamic-typed column so that each value in the collection gets a separate row. All the other columns in an expanded row are duplicated. It's the opposite of makelist.

The following query generates sample data by creating a set and then using it to demonstrate the **mv-expand** capabilities.

[Click to run query]

```
let FloodDataSet = StormEvents
| where EventType == "Flood"
| summarize FloodReports = makeset(StartTime) by State
| project State, FloodReports;
FloodDataSet
| mv-expand FloodReports
```

percentiles()

percentiles(): Returns an estimate for the specified **nearest-rank percentile** of the population defined by an expression. The accuracy depends on the density of population in the region of the percentile. Can be used only in the context of aggregation inside **summarize**.

The following query calculates percentiles for storm duration.

[\[Click to run query\]](#)

```
StormEvents
| extend duration = EndTime - StartTime
| where duration > 0s
| where duration < 3h
| summarize percentiles(duration, 5, 20, 50, 80, 95)
```

The following query calculates percentiles for storm duration by state and normalizes the data by five-minute bins (`5m`).

[\[Click to run query\]](#)

```
StormEvents
| extend duration = EndTime - StartTime
| where duration > 0s
| where duration < 3h
| summarize event_count = count() by bin(duration, 5m), State
| summarize percentiles(duration, 5, 20, 50, 80, 95) by State
```

Cross Dataset

This section covers elements that enable you to create more complex queries, join data across tables, and query across databases and clusters.

let

let: Improves modularity and reuse. The **let** statement allows you to break a potentially complex expression into multiple parts, each bound to a name, and compose those parts together. A **let** statement can also be used to create user-defined functions and views (expressions over tables whose results look like a new table). Expressions bound by a **let** statement can be of scalar type, of tabular type, or user-defined function (lambdas).

The following example creates a tabular type variable and uses it in a subsequent expression.

[\[Click to run query\]](#)

```
let LightningStorms =
StormEvents
| where EventType == "Lightning";
let AvalancheStorms =
StormEvents
| where EventType == "Avalanche";
LightningStorms
| join (AvalancheStorms) on State
| distinct State
```

join

join: Merge the rows of two tables to form a new table by matching values of the specified column(s) from each table. Kusto supports a full range of join types: **fullouter**, **inner**, **innerunique**, **leftanti**, **leftantisemi**, **leftouter**, **leftsemi**, **rightanti**, **rightantisemi**, **rightouter**, **rightsemi**.

The following example joins two tables with an inner join.

[\[Click to run query\]](#)

```

let X = datatable(Key:string, Value1:long)
[
    'a',1,
    'b',2,
    'b',3,
    'c',4
];
let Y = datatable(Key:string, Value2:long)
[
    'b',10,
    'c',20,
    'c',30,
    'd',40
];
X
| join kind=inner Y on Key

```

TIP

Use **where** and **project** operators to reduce the numbers of rows and columns in the input tables, before the join. If one table is always smaller than the other, use it as the left (piped) side of the join. The columns for the join match must have the same name. Use the **project** operator if necessary to rename a column in one of the tables.

serialize

serialize: Serializes the row set so you can use functions that require serialized data, like **row_number()**.

The following query succeeds because the data is serialized.

[\[Click to run query\]](#)

```

StormEvents
| summarize count() by State
| serialize
| extend row_number = row_number()

```

The row set is also considered as serialized if it's a result of: **sort**, **top**, or **range** operators, optionally followed by **project**, **project-away**, **extend**, **where**, **parse**, **mv-expand**, or **take** operators.

[\[Click to run query\]](#)

```

StormEvents
| summarize count() by State
| sort by State asc
| extend row_number = row_number()

```

Cross-database and cross-cluster queries

Cross-database and cross-cluster queries: You can query a database on the same cluster by referring it as `database("MyDatabase").MyTable`. You can query a database on a remote cluster by referring to it as `cluster("MyCluster").database("MyDatabase").MyTable`.

The following query is called from one cluster and queries data from `MyCluster` cluster. To run this query, use your own cluster name and database name.

```

cluster("MyCluster").database("Wiki").PageViews
| where Views < 2000
| take 1000;

```

User Analytics

This section includes elements and queries that demonstrate how easy it is to perform analysis of user behaviors in Kusto.

activity_counts_metrics plugin

activity_counts_metrics plugin: Calculates useful activity metrics (total count values, distinct count values, distinct count of new values, and aggregated distinct count). Metrics are calculated for each time window, then they are compared, and aggregated to and with all previous time windows.

The following query analyzes user adoption by calculating daily activity counts.

[Click to run query]

```
let start=datetime(2017-08-01);
let end=datetime(2017-08-04);
let window=1d;
let T = datatable(UserId:string, Timestamp:datetime)
[
    'A', datetime(2017-08-01),
    'D', datetime(2017-08-01),
    'J', datetime(2017-08-01),
    'B', datetime(2017-08-01),
    'C', datetime(2017-08-02),
    'T', datetime(2017-08-02),
    'J', datetime(2017-08-02),
    'H', datetime(2017-08-03),
    'T', datetime(2017-08-03),
    'T', datetime(2017-08-03),
    'J', datetime(2017-08-03),
    'B', datetime(2017-08-03),
    'S', datetime(2017-08-03),
    'S', datetime(2017-08-04),
];
T
| evaluate activity_counts_metrics(UserId, Timestamp, start, end,
window)
```

activity_engagement plugin

activity_engagement plugin: Calculates activity engagement ratio based on ID column over a sliding timeline window. **activity_engagement plugin** can be used for calculating DAU, WAU, and MAU (daily, weekly, and monthly active users).

The following query returns the ratio of total distinct users using an application daily compared to total distinct users using the application weekly, on a moving seven-day window.

[Click to run query]

```
// Generate random data of user activities
let _start = datetime(2017-01-01);
let _end = datetime(2017-01-31);
range _day from _start to _end step 1d
| extend d = tolong(_day - _start)/1d
| extend r = rand()+1
| extend _users=range(tolong(d*50*r), tolong(d*50*r+100*r-1), 1)
| mv-expand id=_users to typeof(long) limit 1000000
// Calculate DAU/WAU ratio
| evaluate activity_engagement(['id'], _day, _start, _end, 1d, 7d)
| project _day, Dau_Wau=activity_ratio*100
| render timechart
```

TIP

When calculating DAU/MAU, change the end date and the moving window period (OuterActivityWindow).

activity_metrics plugin

activity_metrics plugin: Calculates useful activity metrics (distinct count values, distinct count of new values, retention rate, and churn rate) based on the current period window vs. the previous period window.

The following query calculates the churn and retention rate for a given dataset.

[\[Click to run query\]](#)

```
// Generate random data of user activities
let _start = datetime(2017-01-02);
let _end = datetime(2017-05-31);
range _day from _start to _end step 1d
| extend d = tolong(_day - _start)/1d
| extend r = rand()+1
| extend _users=range(tolong(d*50*r), tolong(d*50*r+200*r-1), 1)
| mv-expand id=_users to typeof(long) limit 1000000
| where _day > datetime(2017-01-02)
| project _day, id
// Calculate weekly retention rate
| evaluate activity_metrics(['id'], _day, _start, _end, 7d)
| project _day, retention_rate*100, churn_rate*100
| render timechart
```

new_activity_metrics plugin

new_activity_metrics plugin: Calculates useful activity metrics (distinct count values, distinct count of new values, retention rate, and churn rate) for the cohort of new users. The concept of this plugin is similar to [activity_metrics plugin](#), but focuses on new users.

The following query calculates a retention and churn rate with a week-over-week window for the new users cohort (users that arrived on the first week).

[\[Click to run query\]](#)

```
// Generate random data of user activities
let _start = datetime(2017-05-01);
let _end = datetime(2017-05-31);
range Day from _start to _end step 1d
| extend d = tolong(Day - _start)/1d
| extend r = rand()+1
| extend _users=range(tolong(d*50*r), tolong(d*50*r+200*r-1), 1)
| mv-expand id=_users to typeof(long) limit 1000000
// Take only the first week cohort (last parameter)
| evaluate new_activity_metrics(['id'], Day, _start, _end, 7d, _start)
| project from_Day, to_Day, retention_rate, churn_rate
```

session_count plugin

session_count plugin: Calculates the count of sessions based on ID column over a timeline.

The following query returns the count of sessions. A session is considered active if a user ID appears at least once at a timeframe of 100-time slots, while the session look-back window is 41-time slots.

[\[Click to run query\]](#)

```

let _data = range Timeline from 1 to 9999 step 1
| extend __key = 1
| join kind=inner (range Id from 1 to 50 step 1 | extend __key=1) on __key
| where Timeline % Id == 0
| project Timeline, Id;
// End of data definition
_data
| evaluate session_count(Id, Timeline, 1, 10000, 100, 41)
| render linechart

```

funnel_sequence plugin

funnel_sequence plugin: Calculates the distinct count of users who have taken a sequence of states; shows the distribution of previous and next states that have led to or were followed by the sequence.

The following query shows what event happens before and after all Tornado events in 2007.

[\[Click to run query\]](#)

```

// Looking on StormEvents statistics:
// Q1: What happens before Tornado event?
// Q2: What happens after Tornado event?
StormEvents
| evaluate funnel_sequence(EpisodeId, StartTime, datetime(2007-01-01), datetime(2008-01-01), 1d,365d,
EventType, dynamic(['Tornado']))

```

funnel_sequence_completion plugin

funnel_sequence_completion plugin: Calculates the funnel of completed sequence steps within different time periods.

The following query checks the completion funnel of the sequence: `Hail -> Tornado -> Thunderstorm -> Wind` in "overall" times of one hour, four hours, and one day (`[1h, 4h, 1d]`).

[\[Click to run query\]](#)

```

let _start = datetime(2007-01-01);
let _end = datetime(2008-01-01);
let _windowSize = 365d;
let _sequence = dynamic(['Hail', 'Tornado', 'Thunderstorm', 'Wind']);
let _periods = dynamic([1h, 4h, 1d]);
StormEvents
| evaluate funnel_sequence_completion(EpisodeId, StartTime, _start, _end, _windowSize, EventType,
_sequence, _periods)

```

Functions

This section covers **functions**: reusable queries that are stored on the server. Functions can be invoked by queries and other functions (recursive functions are not supported).

NOTE

You cannot create functions on the help cluster, which is read-only. Use your own test cluster for this part.

The following example creates a function that takes a state name (`MyState`) as an argument.

```
.create function with (folder="Demo")
MyFunction (MyState: string)
{
    StormEvents
    | where State =~ MyState
}
```

The following example calls a function, which gets data for the state of Texas.

```
MyFunction ("Texas")
| summarize count()
```

The following example deletes the function that was created in the first step.

```
.drop function MyFunction
```

Next steps

[Kusto Query Language reference](#)

Query data in Azure Data Lake using Azure Data Explorer (Preview)

12/8/2019 • 6 minutes to read • [Edit Online](#)

Azure Data Lake Storage is a highly scalable and cost-effective data lake solution for big data analytics. It combines the power of a high-performance file system with massive scale and economy to help you speed your time to insight. Data Lake Storage Gen2 extends Azure Blob Storage capabilities and is optimized for analytics workloads.

Azure Data Explorer integrates with Azure Blob Storage and Azure Data Lake Storage Gen2, providing fast, cached, and indexed access to data in the lake. You can analyze and query data in the lake without prior ingestion into Azure Data Explorer. You can also query across ingested and uningested native lake data simultaneously.

TIP

The best query performance necessitates data ingestion into Azure Data Explorer. The capability to query data in Azure Data Lake Storage Gen2 without prior ingestion should only be used for historical data or data that is rarely queried. [Optimize your query performance in the lake](#) for best results.

Create an external table

NOTE

Currently supported storage accounts are Azure Blob Storage or Azure Data Lake Storage Gen2. Currently supported data formats are json, csv, tsv and txt.

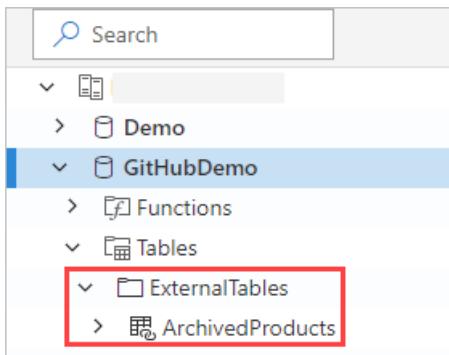
1. Use the `.create external table` command to create an external table in Azure Data Explorer. Additional external table commands such as `.show`, `.drop`, and `.alter` are documented in [External table commands](#).

```
.create external table ArchivedProducts(  
    Timestamp:datetime,  
    ProductId:long, ProductDescription:string)  
    kind=blob  
    partition by bin(Timestamp, 1d)  
    dataformat=csv (h@'http://storageaccount.blob.core.windows.net/container1;secretKey')  
    with (compressed = true)
```

NOTE

- Increased performance is expected with more granular partitioning. For example, queries over external tables with daily partitions, will have better performance than those queries with monthly partitioned tables.
- When you define an external table with partitions, the storage structure is expected to be identical. For example, if the table is defined with a DateTime partition in yyyy/MM/dd format (default), the URI storage file path should be `container1/yyyy/MM/dd/all_exported_blobs`.
- If the external table is partitioned by a datetime column, always include a time filter for a closed range in your query (for example, the query - `ArchivedProducts | where Timestamp between (ago(1h) .. 10m)` - should perform better than this (opened range) one - `ArchivedProducts | where Timestamp > ago(1h)`).

2. The external table is visible in the left pane of the Web UI



Create an external table with json format

You can create an external table with json format. For more information see [External table commands](#)

1. Use the `.create external table` command to create a table named *ExternalTableJson*:

```
.create external table ExternalTableJson (rownumber:int, rowguid:guid)
kind=blob
dataformat=json
(
    h@'http://storageaccount.blob.core.windows.net/container1;secretKey'
)
with
(
    docstring = "Docs",
    folder = "ExternalTables",
    namePrefix="Prefix"
)
```

2. Json format necessitates a second step of creating mapping to columns as shown below. In the following query, create a specific json mapping named *mappingName*:

```
.create external table ExternalTableJson json mapping "mappingName" '[{"column": "rownumber",
"datatype": "int", "path": "$.rownumber"}, {"column": "rowguid", "path": "$.rowguid"}]'
```

External table permissions

- The database user can create an external table. The table creator automatically becomes the table administrator.
- The cluster, database, or table administrator can edit an existing table.
- Any database user or reader can query an external table.

Query an external table

To query an external table, use the `external_table()` function, and provide the table name as the function argument. The rest of the query is standard Kusto query language.

```
external_table("ArchivedProducts") | take 100
```

TIP

Intellisense isn't currently supported on external table queries.

Query an external table with json format

To query an external table with json format, use the `external_table()` function, and provide both table name and mapping name as the function arguments. In the query below, if *mappingName* is not specified, a mapping that

you previously created will be used.

```
external_table('ExternalTableJson', 'mappingName')
```

Query external and ingested data together

You can query both external tables and ingested data tables within the same query. You [join](#) or [union](#) the external table with additional data from Azure Data Explorer, SQL servers, or other sources. Use a [let\(\) statement](#) to assign a shorthand name to an external table reference.

In the example below, *Products* is an ingested data table and *ArchivedProducts* is an external table that contains data in the Azure Data Lake Storage Gen2:

```
let T1 = external_table("ArchivedProducts") | where TimeStamp > ago(100d);
let T = Products; //T is an internal table
T1 | join T on ProductId | take 10
```

Query *TaxiRides* external table in the help cluster

The *TaxiRides* sample data set contains New York City taxi data from [NYC Taxi and Limousine Commission](#).

Create external table *TaxiRides*

NOTE

This section depicts the query used to create the *TaxiRides* external table in the *help* cluster. Since this table has already been created you can skip this section and perform [query *TaxiRides* external table data](#).

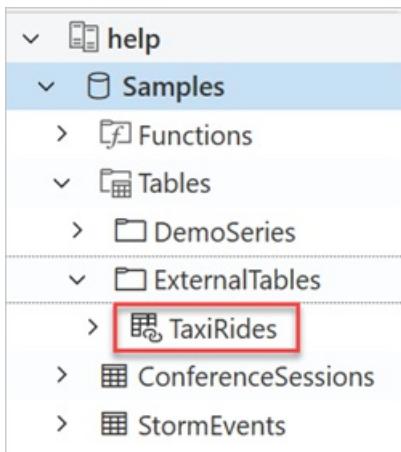
1. The following query was used to create the external table *TaxiRides* in the help cluster.

```

.create external table TaxiRides
(
    trip_id: long,
    vendor_id: string,
    pickup_datetime: datetime,
    dropoff_datetime: datetime,
    store_and_fwd_flag: string,
    rate_code_id: int,
    pickup_longitude: real,
    pickup_latitude: real,
    dropoff_longitude: real,
    dropoff_latitude: real,
    passenger_count: int,
    trip_distance: real,
    fare_amount: real,
    extra: real,
    mta_tax: real,
    tip_amount: real,
    tolls_amount: real,
    ehail_fee: real,
    improvement_surcharge: real,
    total_amount: real,
    payment_type: string,
    trip_type: int,
    pickup: string,
    dropoff: string,
    cab_type: string,
    precipitation: int,
    snow_depth: int,
    snowfall: int,
    max_temperature: int,
    min_temperature: int,
    average_wind_speed: int,
    pickup_nyct2010_gid: int,
    pickup_ctlabel: string,
    pickup_borocode: int,
    pickup_boroname: string,
    pickup_ct2010: string,
    pickup_boroct2010: string,
    pickup_cdeligibil: string,
    pickup_ntacode: string,
    pickup_ntaname: string,
    pickup_puma: string,
    dropoff_nyct2010_gid: int,
    dropoff_ctlabel: string,
    dropoff_borocode: int,
    dropoff_boroname: string,
    dropoff_ct2010: string,
    dropoff_boroct2010: string,
    dropoff_cdeligibil: string,
    dropoff_ntacode: string,
    dropoff_ntaname: string,
    dropoff_puma: string
)
kind=blob
partition by bin(pickup_datetime, 1d)
dataformat=csv
(
    h@'http://storageaccount.blob.core.windows.net/container1;secretKey'
)

```

2. The resulting table was created in the *help* cluster:



Query *TaxiRides* external table data

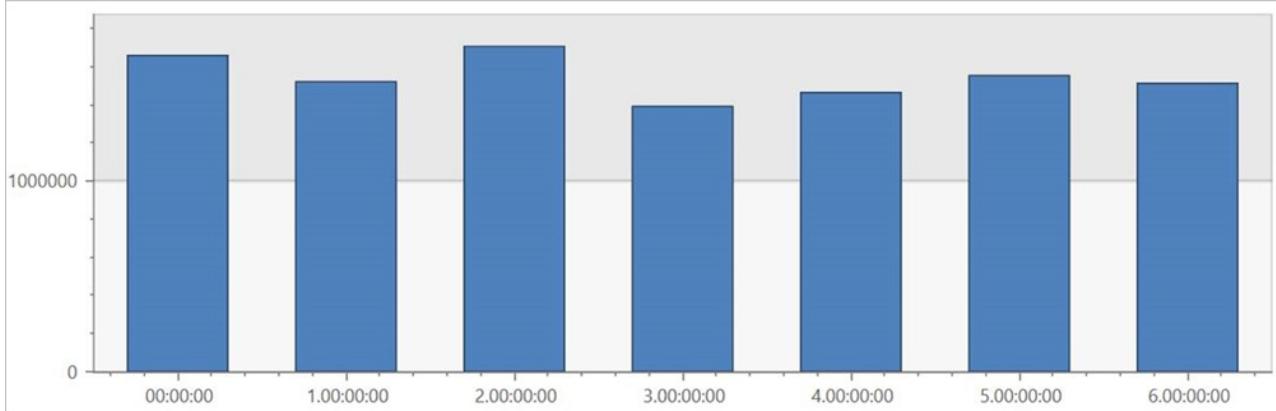
Sign in to <https://dataexplorer.azure.com/clusters/help/databases/Samples> to query the *TaxiRides* external table.

Query *TaxiRides* external table without partitioning

Run this query on the external table *TaxiRides* to depict rides for each day of the week, across the entire data set.

```
external_table("TaxiRides")
| summarize count() by dayofweek(pickup_datetime)
| render columnchart
```

This query shows the busiest day of the week. Since the data isn't partitioned, this query may take a long time to return results (up to several minutes).

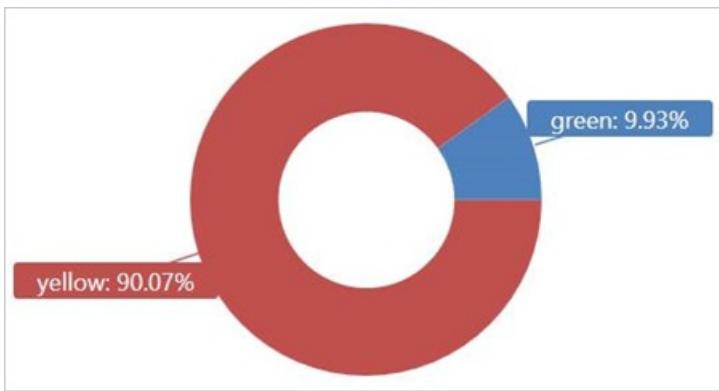


Query *TaxiRides* external table with partitioning

Run this query on the external table *TaxiRides* showing taxi cab types (yellow or green) used in January of 2017.

```
external_table("TaxiRides")
| where pickup_datetime between (datetime(2017-01-01) .. datetime(2017-02-01))
| summarize count() by cab_type
| render piechart
```

This query uses partitioning, which optimizes query time and performance. The query filters on a partitioned column (*pickup_datetime*) and returns results in a few seconds.



You can write additional queries to run on the external table *TaxiRides* and learn more about the data.

Optimize your query performance

Optimize your query performance in the lake by using the following best practices for querying external data.

Data format

Use a columnar format for analytical queries since:

- Only the columns relevant to a query can be read.
- Column encoding techniques can reduce data size significantly.

Azure Data Explorer supports Parquet and ORC columnar formats. Parquet format is suggested due to optimized implementation.

Azure region

Ascertain that external data resides in the same Azure region as your Azure Data Explorer cluster. This reduces cost and data fetch time.

File size

Optimal file size is hundreds of Mb (up to 1 Gb) per file. Avoid many small files that require unneeded overhead, such as slower file enumeration process and limited use of columnar format. Note that the number of files should be greater than the number of CPU cores in your Azure Data Explorer cluster.

Compression

Use compression to reduce the amount of data being fetched from the remote storage. For Parquet format, use the internal Parquet compression mechanism that compresses column groups separately, thus allowing you to read them separately. To validate use of compression mechanism, check that the files are named as follows: ".gz.parquet" or ".snappy.parquet" as opposed to ".parquet.gz").

Partitioning

Organize your data using "folder" partitions that enables the query to skip irrelevant paths. When planning partitioning consider file size and common filters in your queries such as timestamp or tenant ID.

VM size

Select VM SKUs with more cores and higher network throughput (memory is less important). For more information see [Select the correct VM SKU for your Azure Data Explorer cluster](#).

Next steps

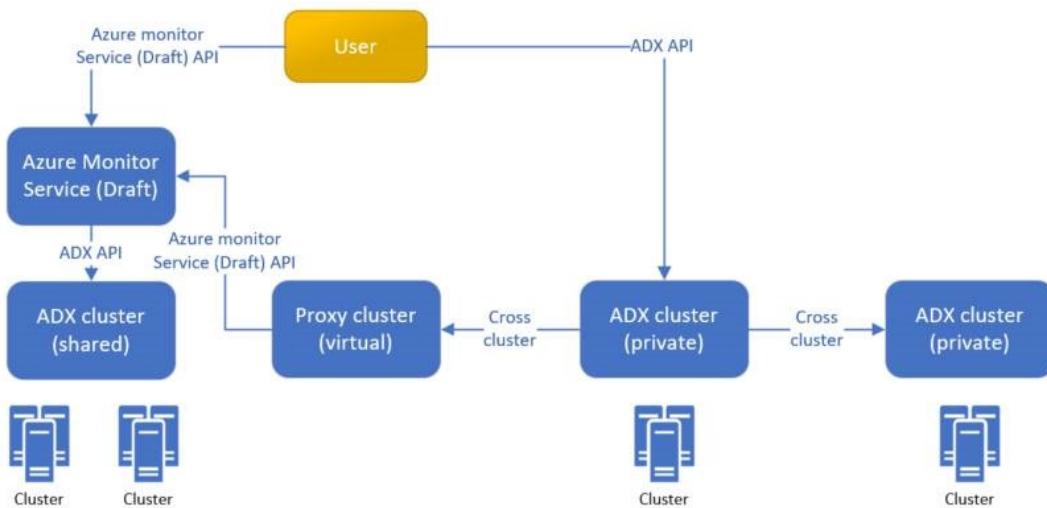
Query your data in the Azure Data Lake using Azure Data Explorer. Learn to [write queries](#) and derive additional insights from your data.

Query data in Azure Monitor using Azure Data Explorer (Preview)

9/12/2019 • 3 minutes to read • [Edit Online](#)

The Azure Data Explorer proxy cluster (ADX Proxy) is an entity that enables you to perform cross product queries between Azure Data Explorer, Application Insights (AI), and Log Analytics (LA) in the Azure Monitor service. You can map Azure Monitor Log Analytics workspaces or Application Insights apps as a proxy cluster. You can then query the proxy cluster using Azure Data Explorer tools and refer to it in a cross cluster query. The article shows how to connect to a proxy cluster, add a proxy cluster to Azure Data Explorer Web UI, and run queries against your AI apps or LA workspaces from Azure Data Explorer.

The Azure Data Explorer proxy flow:



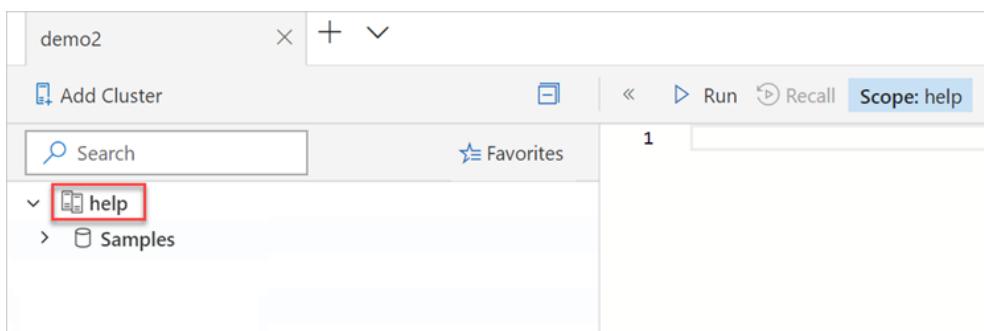
Prerequisites

NOTE

The ADX Proxy is in preview mode. To enable this feature, contact the [ADXProxy](#) team.

Connect to the proxy

1. Verify your Azure Data Explorer native cluster (such as *help* cluster) appears on the left menu before you connect to your Log Analytics or Application Insights cluster.



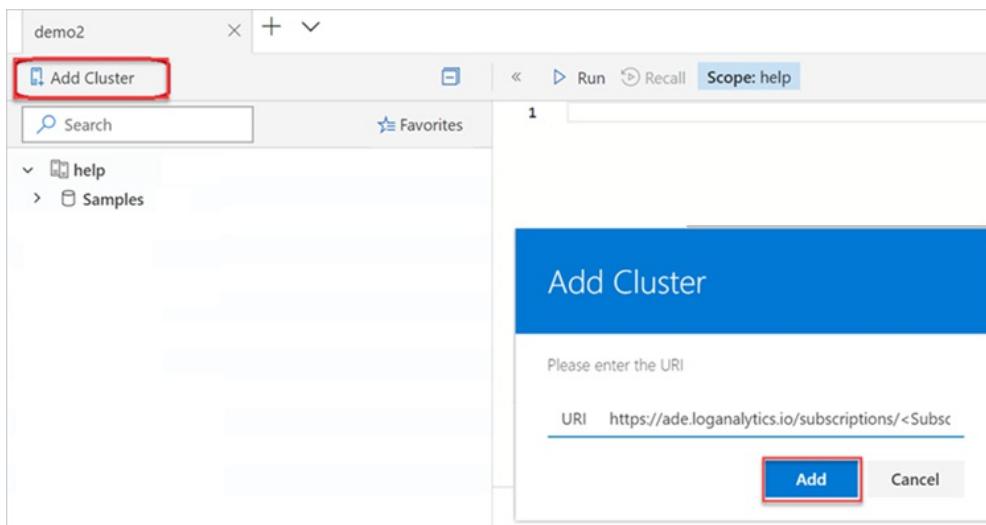
2. In the Azure Data Explorer UI (<https://dataexplorer.azure.com/clusters>), select **Add Cluster**.

3. In the **Add Cluster** window:

- Add the URL to the LA or AI cluster. For example:

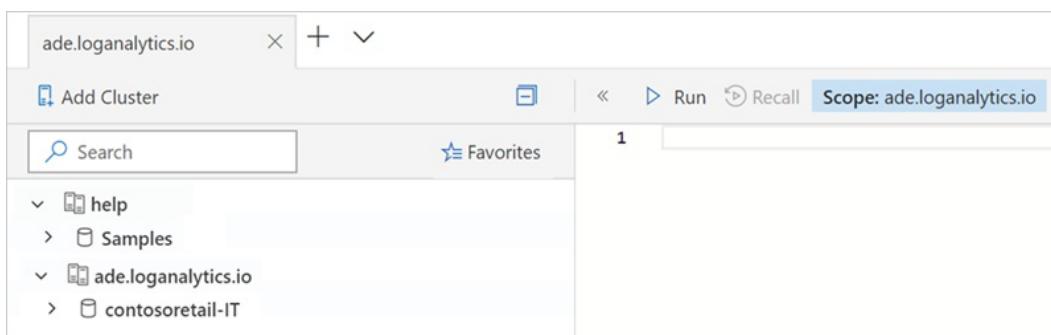
```
https://ade.loganalytics.io/subscriptions/<subscription-id>/resourcegroups/<resource-group-name>/providers/microsoft.operationalinsights/workspaces/<workspace-name>
```

- Select **Add**.



If you add a connection to more than one proxy cluster, give each a different name. Otherwise they'll all have the same name in the left pane.

- After the connection is established, your LA or AI cluster will appear in the left pane with your native ADX cluster.



Run queries

You can use Kusto Explorer, ADX web Explorer, Jupyter KqlMagic, or REST API to query the proxy clusters.

TIP

- Database name should have the same name as the resource specified in the proxy cluster. Names are case sensitive.
- In cross cluster queries, make sure that the naming of Application Insights apps and Log Analytics workspaces is correct.
 - If names contain special characters, they're replaced by URL encoding in the proxy cluster name.
 - If names include characters that don't meet [KQL identifier name rules](#), they are replaced by the dash - character.

Query against the native Azure Data Explorer cluster

Run queries on your Azure Data Explorer cluster (such as *StormEvents* table in *help* cluster). When running the query, verify that your native Azure Data Explorer cluster is selected in the left pane.

```
StormEvents | take 10 // Demonstrate query through the native ADX cluster
```

Query against your LA or AI cluster

When you run queries on your LA or AI cluster, verify that your LA or AI cluster is selected in the left pane.

```
Perf | take 10 // Demonstrate query through the proxy on the LA workspace
```

```
1 StormEvents | take 10 // Demonstrate query through the native ADX cluster
2
3 Perf | take 10 // Demonstrate query through the proxy on the LA workspace
```

Query your LA or AI cluster from the ADX proxy

When you run queries on your LA or AI cluster from the proxy, verify your ADX native cluster is selected in the left pane. The following example demonstrates a query of the LA workspace using the native ADX cluster

```
cluster('https://ade.loganalytics.io/subscriptions/<subscription-id>/resourcegroups/<resource-group-name>/providers/microsoft.operationalinsights工作空间名').database('<workspace-name>').Perf
| take 10
```

```
1 StormEvents | take 10 // Demonstrate query through the native ADX cluster
2
3 Perf | take 10 // Demonstrate query through the proxy on the LA workspace
4
5 cluster('https://ade.loganalytics.io/subscriptions/<subscription-id>/resourcegroups/<resource-group-name>/providers/microsoft.operationalinsights工作空间名').database('<workspace-name>').Perf
6 | take 10 // Demonstrate query of the LA workspace through the native DX cluster
7
8
```

Cross query of LA or AI cluster and the ADX cluster from the ADX proxy

When you run cross cluster queries from the proxy, verify your ADX native cluster is selected in the left pane. The following examples demonstrate combining ADX cluster tables (using `union`) with LA workspace.

```
union StormEvents, cluster('https://ade.loganalytics.io/subscriptions/<subscription-id>/resourcegroups/<resource-group-name>/providers/microsoft.operationalinsights工作空间名').database('<workspace-name>').Perf
| take 10
```

```
let CL1 = 'https://ade.loganalytics.io/subscriptions/<subscription-id>/resourcegroups/<resource-group-name>/providers/microsoft.operationalinsights工作空间名';
union <ADX table>, cluster(CL1).database('<workspace-name>').<table name>
```

```
1 StormEvents | take 10 // Demonstrate query through the native ADX cluster
2
3 Perf | take 10 // Demonstrate query through the proxy on the LA workspace
4
5 cluster('https://ade.loganalytics.io/subscriptions/<subscription-id>/resourcegroups/<resource-group-name>/providers/microsoft.operationalinsights工作空间名').database('<workspace-name>').Perf
6 | take 10 // Demonstrate query of the LA workspace through the native DX cluster
7
8 // IMPORTANT - before you run the cross cluster query - verify you are running the query while the ADX cluster is marked on the left view pane
9 union StormEvents, cluster('https://ade.loganalytics.io/subscriptions/<subscription ID>/resourcegroups/<resource-group-name>/providers/microsoft.operationalinsights工作空间名').database('<workspace name>').Perf
10 | take 10 // union tables from both the ADX cluster and the LA workspace
11
```

Using the `join` operator, instead of `union`, may require a `hint` to run it on an Azure Data Explorer native cluster (and not on the proxy).

Additional syntax examples

The following syntax options are available when calling the Application Insights (AI) or Log Analytics (LA) clusters:

SYNTAX DESCRIPTION	APPLICATION INSIGHTS	LOG ANALYTICS
Database within a cluster that contains only the defined resource in this subscription (recommended for cross cluster queries)	cluster(https://ade.applicationinsights.io/subscriptions/<subscription-id>/resourcegroups/<resource-group-name>/providers/microsoft.insights/components/<name>/providers/microsoft.operationalinsights工作空间名).database('<ai-app-name>')	cluster(https://ade.loganalytics.io/subscriptions/<subscription-id>/resourcegroups/<resource-group-name>/providers/microsoft.operationalinsights工作空间名).database('<workspace-name>')
Cluster that contains all apps/workspaces in this subscription	cluster(https://ade.applicationinsights.io/subscriptions/<subscription-id>)	cluster(https://ade.logalytics.io/subscriptions/<subscription-id>)

SYNTAX DESCRIPTION	APPLICATION INSIGHTS	LOG ANALYTICS
Cluster that contains all apps/workspaces in the subscription and are members of this resource group	cluster(https://ade.applicationinsights.io/subscription)resourcegroups/<resource-group-name>)	cluster(https://ade.loganalytics.io/subscriptions/<subscription-id>/resourcegroups/<resource-group-name>)
Cluster that contains only the defined resource in this subscription	cluster(https://ade.applicationinsights.io/subscription)resourcegroups/<resource-group-name>/providers/microsoft.insights/components/<name>)	cluster(https://ade.logalytics.io/subscriptions/<subscription-id>/resourcegroups/<resource-group-name>/providers/microsoft.operationalinsights/works)

Next steps

[Write queries](#)

Debug Kusto query language inline Python using VS Code

12/4/2019 • 2 minutes to read • [Edit Online](#)

Azure Data Explorer supports running Python code embedded in Kusto query language using the [python\(\)](#) plugin. The plugin runtime is hosted in a sandbox, an isolated and secure Python environment. The [python\(\)](#) plugin capability extends Kusto query language native functionalities with the huge archive of OSS Python packages. This extension enables you to run advanced algorithms, such as machine learning, artificial intelligence, statistical, and time series, as part of the query.

Kusto query language tools aren't convenient for developing and debugging Python algorithms. Therefore, develop the algorithm on your favorite Python-integrated development environment such as Jupyter, PyCharm, VS, or VS Code. When the algorithm is complete, copy and paste into KQL. To improve and streamline this workflow, Azure Data Explorer supports integration between Kusto Explorer or Web UI clients and VS Code for authoring and debugging KQL inline Python code.

NOTE

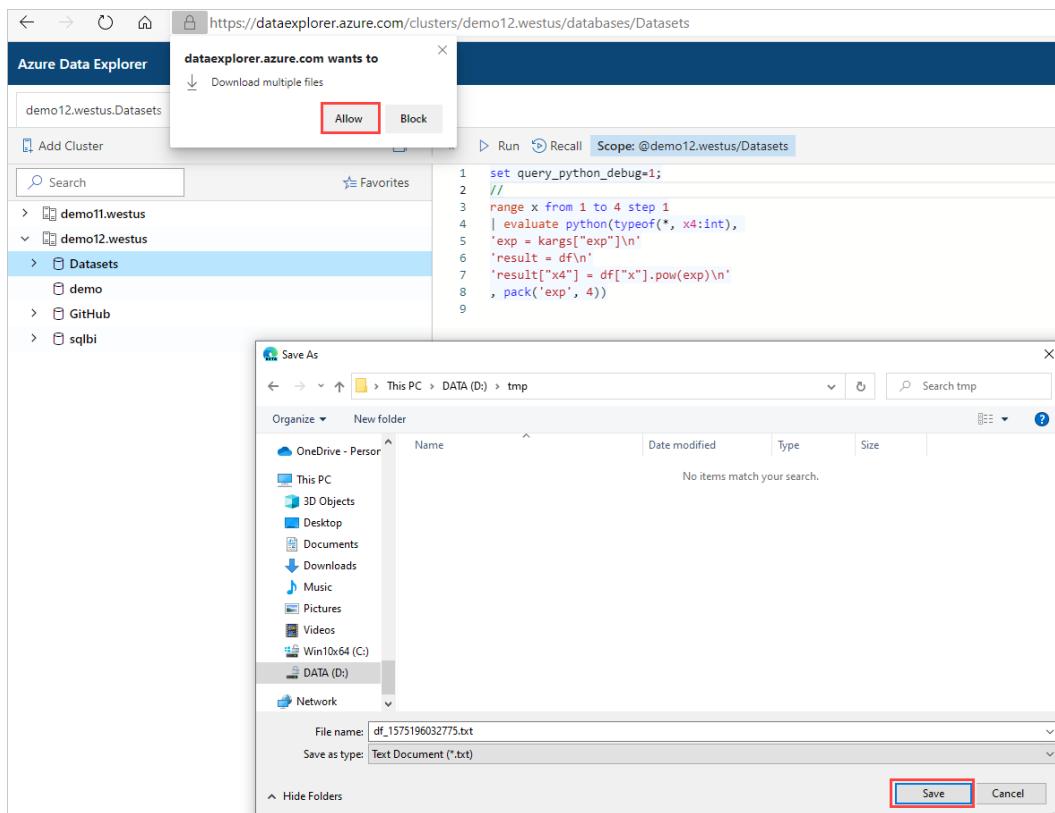
This workflow can only be used to debug relatively small input tables (up to few MB). Therefore, you may need to limit the input for debugging. If you need to process a large table, limit it for debugging using `| take`, `| sample`, or `where rand() < 0.x`.

Prerequisites

1. Install Python [Anaconda Distribution](#). In **Advanced Options**, select **Add Anaconda to my PATH environment variable**.
2. Install [Visual Studio Code](#)
3. Install [Python extension for Visual Studio Code](#).

Run your query in your client application

1. In your client application, prefix a query containing inline Python with `set query_python_debug;`
2. Run the query.
 - Kusto Explorer: VS Code is automatically launched with the `debug_python.py` script.
 - Kusto Web UI:
 - a. Download and save `debug_python.py`, `df.txt`, and `kargs.txt`. In window, select **Allow. Save files** in selected directory.



- b. Right-click `debug_python.py` and open with VS code. The `debug_python.py` script contains the inline Python code, from the KQL query, prefixed by the template code to initialize the input dataframe from `df.txt` and the dictionary of parameters from `kargs.txt`.
3. In VS code, launch the VS code debugger: **Debug > Start Debugging (F5)**, select **Python** configuration. The debugger will launch and automatically breakpoint to debug the inline code.

How does inline Python debugging in VS Code work?

1. The query is parsed and executed in the server until the required `| evaluate python()` clause is reached.
2. The Python sandbox is invoked but instead of running the code, it serializes the input table, the dictionary of parameters, and the code, and sends them back to the client.
3. These three objects are saved in three files: `df.txt`, `kargs.txt`, and `debug_python.py` in the selected directory (Web UI) or in the client `%TEMP%` directory (Kusto Explorer).
4. VS code is launched, preloaded with the `debug_python.py` file that contains a prefix code to initialize `df` and `kargs` from their respective files, followed by the Python script embedded in the KQL query.

Query example

1. Run the following KQL query in your client application:

```

range x from 1 to 4 step 1
| evaluate python(typeof(*, x4:int),
'exp = kargs["exp"]\n'
'result = df\n'
'result["x4"] = df["x"].pow(exp)\n'
, pack('exp', 4))

```

See the resulting table:

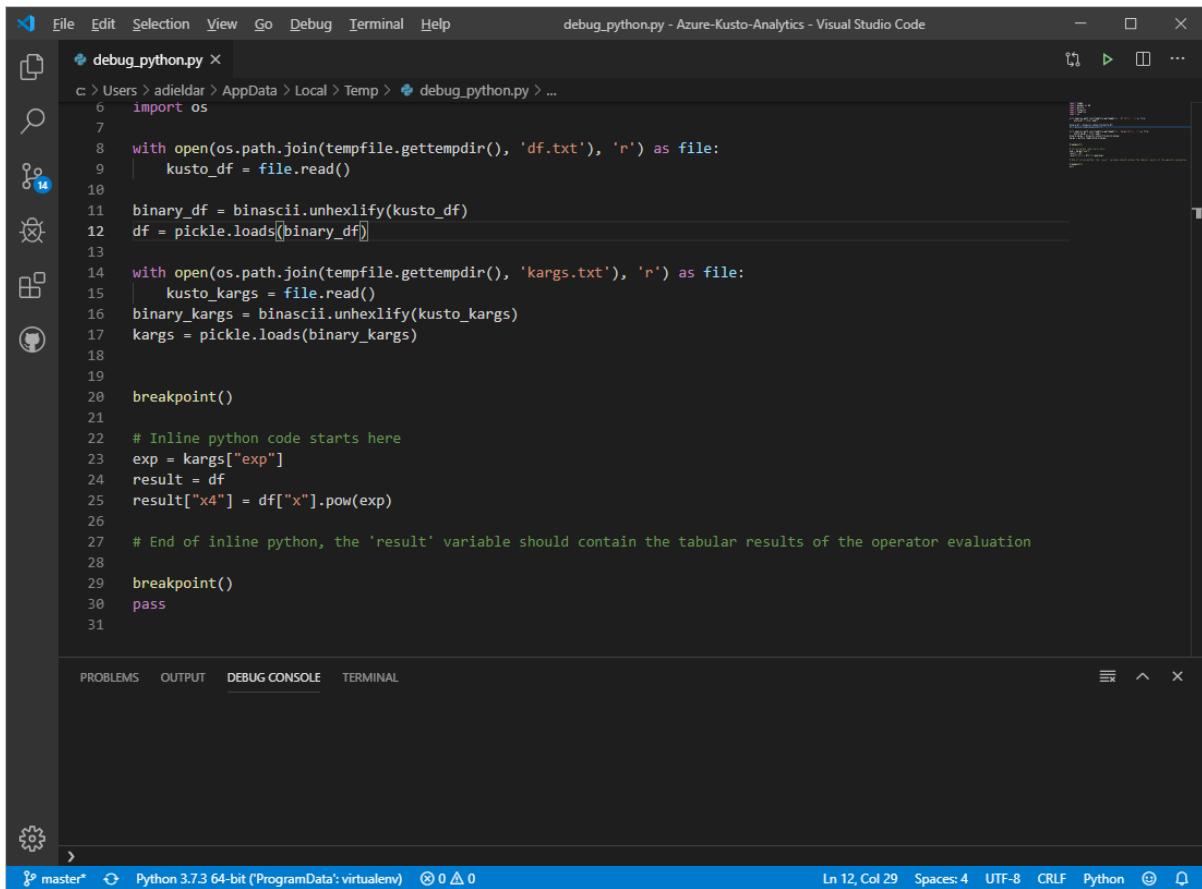
x	x4
1	1

X	X4
2	16
3	81
4	256

2. Run the same KQL query in your client application using `set query_python_debug;`:

```
set query_python_debug;
range x from 1 to 4 step 1
| evaluate python(typeof(*, x4:int),
'exp = kargs["exp"]\n'
'result = df\n'
'result["x4"] = df["x"].pow(exp)\n'
, pack('exp', 4))
```

3. VS Code is launched:



```
File Edit Selection View Go Debug Terminal Help debug_python.py - Azure-Kusto-Analytics - Visual Studio Code
debug_python.py ×
c > Users > adieldar > AppData > Local > Temp > debug_python.py > ...
6 import os
7
8 with open(os.path.join(tempfile.gettempdir(), 'df.txt'), 'r') as file:
9     kusto_df = file.read()
10
11 binary_df = binascii.unhexlify(kusto_df)
12 df = pickle.loads(binary_df)
13
14 with open(os.path.join(tempfile.gettempdir(), 'kargs.txt'), 'r') as file:
15     kusto_kargs = file.read()
16 binary_kargs = binascii.unhexlify(kusto_kargs)
17 kargs = pickle.loads(binary_kargs)
18
19
20 breakpoint()
21
22 # Inline python code starts here
23 exp = kargs["exp"]
24 result = df
25 result["x4"] = df["x"].pow(exp)
26
27 # End of inline python, the 'result' variable should contain the tabular results of the operator evaluation
28
29 breakpoint()
30 pass
31
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

master* Python 3.7.3 64-bit ('ProgramData':virtualenv) 0 △ 0 Ln 12, Col 29 Spaces:4 UTF-8 CRLF Python

4. VS Code debugs and prints 'result' dataframe in the debug console:

```
import os
with open(os.path.join(tempfile.gettempdir(), 'df.txt'), 'r') as file:
    kusto_df = file.read()
binary_df = binascii.unhexlify(kusto_df)
df = pickle.loads(binary_df)

with open(os.path.join(tempfile.gettempdir(), 'kargs.txt'), 'r') as file:
    kusto_kargs = file.read()
binary_kargs = binascii.unhexlify(kusto_kargs)
kargs = pickle.loads(binary_kargs)

breakpoint()

# Inline python code starts here
exp = kargs["exp"]
result = df
result["x4"] = df["x"].pow(exp)

# End of inline python, the 'result' variable should contain the tabular result
breakpoint()
pass
```

print(result)

x	x4
0	1
1	16
2	81
3	256

NOTE

There may be differences between the Python sandbox image and your local installation. [Check the sandbox image for specific packages by querying the plugin.](#)

Query data using the Azure Data Explorer Python library

8/6/2019 • 2 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Azure Data Explorer provides a [data client library for Python](#). This library enables you to query data from your code. In this article, you connect to a table on the *help cluster* that we have set up to aid learning. You then query a table on that cluster and return the results.

This article is also available as an [Azure Notebook](#).

Prerequisites

- An organizational email account that is a member of Azure Active Directory (AAD)
- [Python](#) installed on your development computer

Install the data library

Install *azure-kusto-data*.

```
pip install azure-kusto-data
```

Add import statements and constants

Import classes from the library, as well as *pandas*, a data analysis library.

```
from azure.kusto.data.request import KustoClient, KustoConnectionStringBuilder  
from azure.kusto.data.exceptions import KustoServiceError  
from azure.kusto.data.helpers import dataframe_from_result_table  
import pandas as pd
```

To authenticate an application, Azure Data Explorer uses your AAD tenant ID. To find your tenant ID, use the following URL, substituting your domain for *YourDomain*.

```
https://login.windows.net/<YourDomain>/.well-known/openid-configuration/
```

For example, if your domain is *contoso.com*, the URL is: <https://login.windows.net/contoso.com/.well-known/openid-configuration/>. Click this URL to see the results; the first line is as follows.

```
"authorization_endpoint": "https://login.windows.net/6babcaad-604b-40ac-a9d7-9fd97c0b779f/oauth2/authorize"
```

The tenant ID in this case is `6babcaad-604b-40ac-a9d7-9fd97c0b779f`. Set the value for `AAD_TENANT_ID` before running this code.

```
AAD_TENANT_ID = "<TenantId>"  
KUSTO_CLUSTER = "https://help.kusto.windows.net/"  
KUSTO_DATABASE = "Samples"
```

Now construct the connection string. This example uses device authentication to access the cluster. You can also use [AAD application certificate](#), [AAD application key](#), and [AAD user and password](#).

```
KCSB = KustoConnectionStringBuilder.with_aad_device_authentication(  
    KUSTO_CLUSTER)  
KCSB.authority_id = AAD_TENANT_ID
```

Connect to Azure Data Explorer and execute a query

Execute a query against the cluster and store the output in a data frame. When this code runs, it returns a message like the following: *To sign in, use a web browser to open the page <https://microsoft.com/devicelogin> and enter the code F3W4VWZDM to authenticate.* Follow the steps to sign-in, then return to run the next code block.

```
KUSTO_CLIENT = KustoClient(KCSB)  
KUSTO_QUERY = "StormEvents | sort by StartTime desc | take 10"  
  
RESPONSE = KUSTO_CLIENT.execute(KUSTO_DATABASE, KUSTO_QUERY)
```

Explore data in DataFrame

After you enter a sign in, the query returns results, and they are stored in a data frame. You can work with the results like you do any other data frame.

```
df = dataframe_from_result_table(RESPONSE.primary_results[0])  
df
```

You should see the top ten results from the StormEvents table.

Next steps

[Ingest data using the Azure Data Explorer Python library](#)

Time series analysis in Azure Data Explorer

12/3/2019 • 7 minutes to read • [Edit Online](#)

Azure Data Explorer (ADX) performs on-going collection of telemetry data from cloud services or IoT devices. This data can be analyzed for various insights such as monitoring service health, physical production processes, and usage trends. Analysis is done on time series of selected metrics to find a deviation in the pattern compared to its typical baseline pattern. ADX contains native support for creation, manipulation, and analysis of multiple time series. In this topic, learn how ADX is used to create and analyze **thousands of time series in seconds**, enabling near real-time monitoring solutions and workflows.

Time series creation

In this section, we'll create a large set of regular time series simply and intuitively using the `make-series` operator, and fill-in missing values as needed. The first step in time series analysis is to partition and transform the original telemetry table to a set of time series. The table usually contains a timestamp column, contextual dimensions, and optional metrics. The dimensions are used to partition the data. The goal is to create thousands of time series per partition at regular time intervals.

The input table `demo_make_series1` contains 600K records of arbitrary web service traffic. Use the command below to sample 10 records:

[\[Click to run query\]](#)

```
demo_make_series1 | take 10
```

The resulting table contains a timestamp column, three contextual dimensions columns, and no metrics:

	TimeStamp	BrowserVer	OsVer	Country/Region
	2016-08-25 09:12:35.4020000	Chrome 51.0	Windows 7	United Kingdom
	2016-08-25 09:12:41.1120000	Chrome 52.0	Windows 10	
	2016-08-25 09:12:46.2300000	Chrome 52.0	Windows 7	United Kingdom
	2016-08-25 09:12:46.5100000	Chrome 52.0	Windows 10	United Kingdom
	2016-08-25 09:12:46.5570000	Chrome 52.0	Windows 10	Republic of Lithuania
	2016-08-25 09:12:47.0470000	Chrome 52.0	Windows 8.1	India
	2016-08-25 09:12:51.3600000	Chrome 52.0	Windows 10	United Kingdom

	2016-08-25 09:12:51.6930000	Chrome 52.0	Windows 7	Netherlands
	2016-08-25 09:12:56.4240000	Chrome 52.0	Windows 10	United Kingdom
	2016-08-25 09:13:08.7230000	Chrome 52.0	Windows 10	India

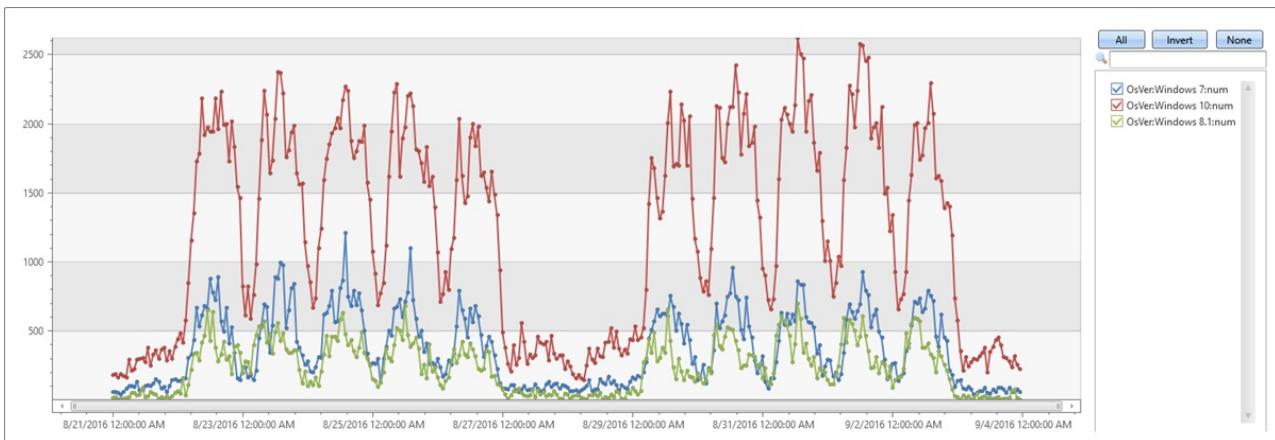
Since there are no metrics, we can only build a set of time series representing the traffic count itself, partitioned by OS using the following query:

[Click to run query]

```
let min_t = toscalar(demo_make_series1 | summarize min(TimeStamp));
let max_t = toscalar(demo_make_series1 | summarize max(TimeStamp));
demo_make_series1
| make-series num=count() default=0 on TimeStamp in range(min_t, max_t, 1h) by OsVer
| render timechart
```

- Use the `make-series` operator to create a set of three time series, where:
 - `num=count()` : time series of traffic
 - `range(min_t, max_t, 1h)` : time series is created in 1-hour bins in the time range (oldest and newest timestamps of table records)
 - `default=0` : specify fill method for missing bins to create regular time series. Alternatively use `series_fill_const()`, `series_fill_forward()`, `series_fill_backward()` and `series_fill_linear()` for changes
 - `byOsVer` : partition by OS
- The actual time series data structure is a numeric array of the aggregated value per each time bin. We use `render timechart` for visualization.

In the table above, we have three partitions. We can create a separate time series: Windows 10 (red), 7 (blue) and 8.1 (green) for each OS version as seen in the graph:



Time series analysis functions

In this section, we'll perform typical series processing functions. Once a set of time series is created, ADX supports a growing list of functions to process and analyze them which can be found in the [time series documentation](#). We will describe a few representative functions for processing and analyzing time series.

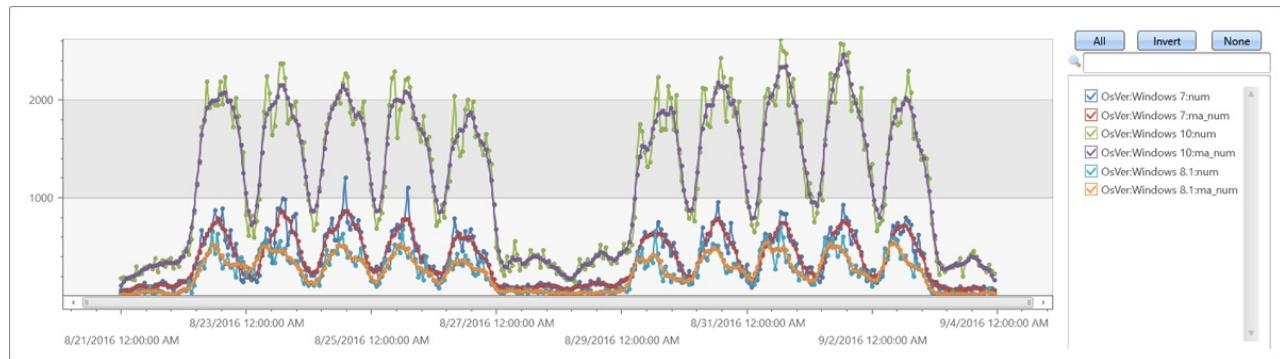
Filtering

Filtering is a common practice in signal processing and useful for time series processing tasks (for example, smooth a noisy signal, change detection).

- There are two generic filtering functions:
 - `series_fir()`: Applying FIR filter. Used for simple calculation of moving average and differentiation of the time series for change detection.
 - `series_iir()`: Applying IIR filter. Used for exponential smoothing and cumulative sum.
- Extend the time series set by adding a new moving average series of size 5 bins (named `ma_num`) to the query:

[Click to run query]

```
let min_t = toscalar(demo_make_series1 | summarize min(TimeStamp));
let max_t = toscalar(demo_make_series1 | summarize max(TimeStamp));
demo_make_series1
| make-series num=count() default=0 on TimeStamp in range(min_t, max_t, 1h) by OsVer
| extend ma_num=series_fir(num, repeat(1, 5), true, true)
| render timechart
```



Regression analysis

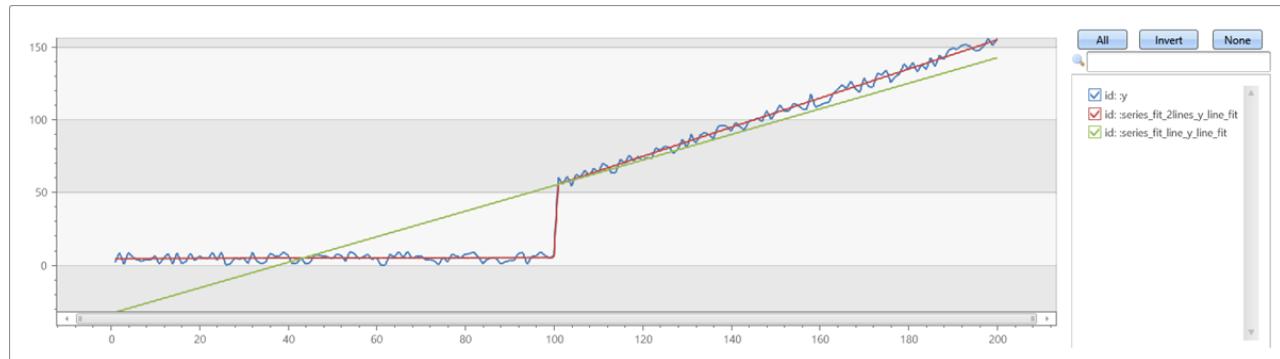
ADX supports segmented linear regression analysis to estimate the trend of the time series.

- Use `series_fit_line()` to fit the best line to a time series for general trend detection.
- Use `series_fit_2lines()` to detect trend changes, relative to the baseline, that are useful in monitoring scenarios.

Example of `series_fit_line()` and `series_fit_2lines()` functions in a time series query:

[Click to run query]

```
demo_series2
| extend series_fit_2lines(y), series_fit_line(y)
| render linechart with(xcolumn=x)
```



- Blue: original time series

- Green: fitted line
- Red: two fitted lines

NOTE

The function accurately detected the jump (level change) point.

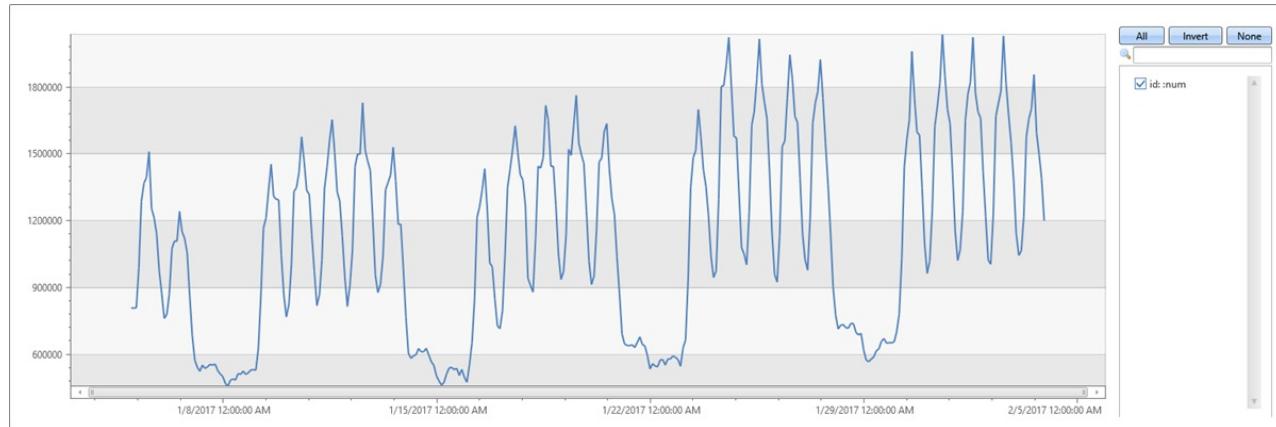
Seasonality detection

Many metrics follow seasonal (periodic) patterns. User traffic of cloud services usually contains daily and weekly patterns that are highest around the middle of the business day and lowest at night and over the weekend. IoT sensors measure in periodic intervals. Physical measurements such as temperature, pressure, or humidity may also show seasonal behavior.

The following example applies seasonality detection on one month traffic of a web service (2-hour bins):

[\[Click to run query\]](#)

```
demo_series3
| render timechart
```



- Use [series_periods_detect\(\)](#) to automatically detect the periods in the time series.
- Use [series_periods_validate\(\)](#) if we know that a metric should have specific distinct period(s) and we want to verify that they exist.

NOTE

It's an anomaly if specific distinct periods don't exist

[\[Click to run query\]](#)

```
demo_series3
| project (periods, scores) = series_periods_detect(num, 0., 14d/2h, 2) //to detect the periods in the time
series
| mv-expand periods, scores
| extend days=2h*todouble(periods)/1d
```

	periods	scores	days
	84	0.820622786055595	7

	12	0.764601405803502	1
--	----	-------------------	---

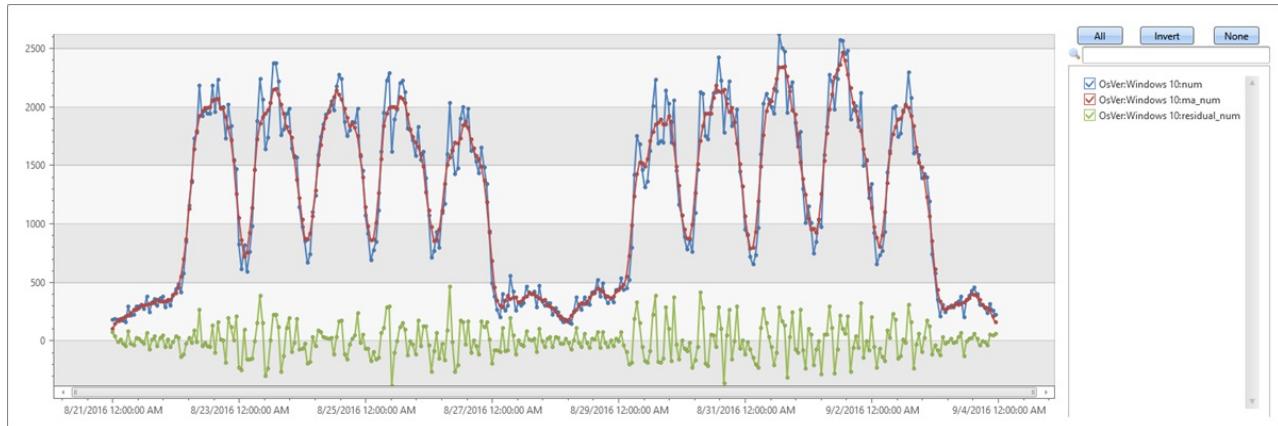
The function detects daily and weekly seasonality. The daily scores less than the weekly because weekend days are different from weekdays.

Element-wise functions

Arithmetic and logical operations can be done on a time series. Using `series_subtract()` we can calculate a residual time series, that is, the difference between original raw metric and a smoothed one, and look for anomalies in the residual signal:

[\[Click to run query\]](#)

```
let min_t = toscalar(demo_make_series1 | summarize min(TimeStamp));
let max_t = toscalar(demo_make_series1 | summarize max(TimeStamp));
demo_make_series1
| make-series num=count() default=0 on TimeStamp in range(min_t, max_t, 1h) by OsVer
| extend ma_num=series_fir(num, repeat(1, 5), true, true)
| extend residual_num=series_subtract(num, ma_num) //to calculate residual time series
| where OsVer == "Windows 10" // filter on Win 10 to visualize a cleaner chart
| render timechart
```



- Blue: original time series
- Red: smoothed time series
- Green: residual time series

Time series workflow at scale

The example below shows how these functions can run at scale on thousands of time series in seconds for anomaly detection. To see a few sample telemetry records of a DB service's read count metric over four days run the following query:

[\[Click to run query\]](#)

```
demo_many_series1
| take 4
```

	TIMESTAMP	Loc	anonOp	DB	DataRead

	2016-09-11 21:00:00.000000 0	Loc 9	5117853934049 630089	262	0
	2016-09-11 21:00:00.000000 0	Loc 9	5117853934049 630089	241	0
	2016-09-11 21:00:00.000000 0	Loc 9	- 8659983319411 49874	262	279862
	2016-09-11 21:00:00.000000 0	Loc 9	3719217345637 83410	255	0

And simple statistics:

[\[Click to run query\]](#)

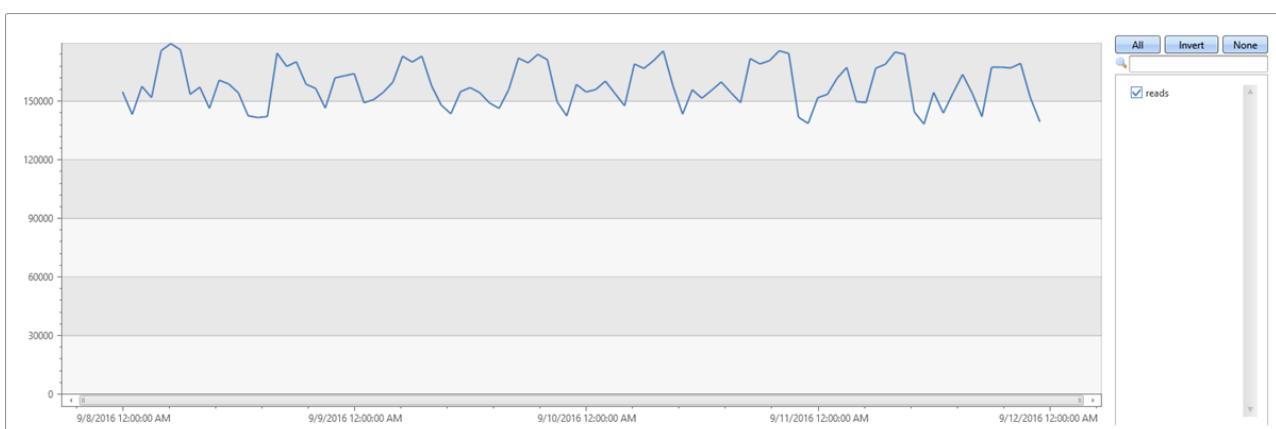
```
demo_many_series1
| summarize num=count(), min_t=min(TIMESTAMP), max_t=max(TIMESTAMP)
```

	num	min_t	max_t
	2177472	2016-09-08 00:00:00.0000000	2016-09-11 23:00:00.0000000

Building a time series in 1-hour bins of the read metric (total four days * 24 hours = 96 points), results in normal pattern fluctuation:

[\[Click to run query\]](#)

```
let min_t = toscalar(demo_many_series1 | summarize min(TIMESTAMP));
let max_t = toscalar(demo_many_series1 | summarize max(TIMESTAMP));
demo_many_series1
| make-series reads=avg(DataRead) on TIMESTAMP in range(min_t, max_t, 1h)
| render timechart with(ymin=0)
```



The above behavior is misleading, since the single normal time series is aggregated from thousands of different instances that may have abnormal patterns. Therefore, we create a time series per instance. An instance is defined

by Loc (location), anonOp (operation), and DB (specific machine).

How many time series can we create?

[Click to run query]

```
demo_many_series1  
| summarize by Loc, Op, DB  
| count
```

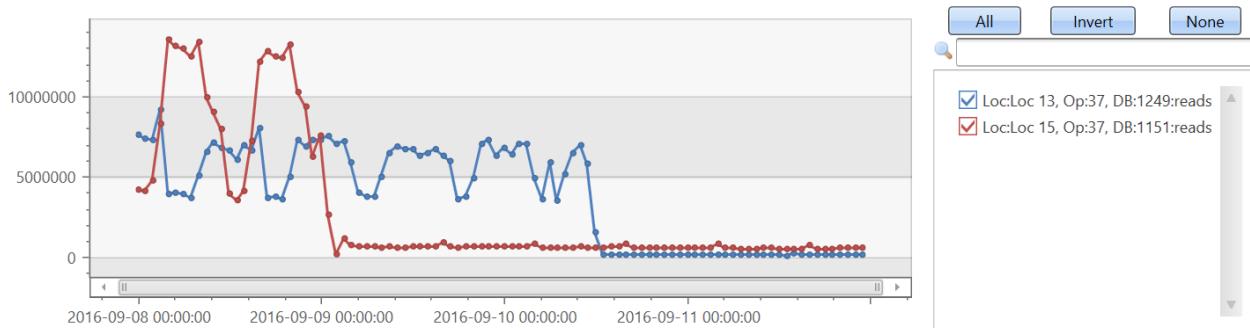
Count
18339

Now, we're going to create a set of 18339 time series of the read count metric. We add the `by` clause to the make-series statement, apply linear regression, and select the top two time series that had the most significant decreasing trend:

[Click to run query]

```
let min_t = toscalar(demo_many_series1 | summarize min(TIMESTAMP));  
let max_t = toscalar(demo_many_series1 | summarize max(TIMESTAMP));  
demo_many_series1  
| make-series reads=avg(DataRead) on TIMESTAMP in range(min_t, max_t, 1h) by Loc, Op, DB  
| extend (rsquare, slope) = series_fit_line(reads)  
| top 2 by slope asc  
| render timechart with(title='Service Traffic Outage for 2 instances (out of 18339)')
```

Service Traffic Outage for 2 instances (out of 18339)



Display the instances:

[Click to run query]

```
let min_t = toscalar(demo_many_series1 | summarize min(TIMESTAMP));  
let max_t = toscalar(demo_many_series1 | summarize max(TIMESTAMP));  
demo_many_series1  
| make-series reads=avg(DataRead) on TIMESTAMP in range(min_t, max_t, 1h) by Loc, Op, DB  
| extend (rsquare, slope) = series_fit_line(reads)  
| top 2 by slope asc  
| project Loc, Op, DB, slope
```

Loc	Op	DB	slope
-----	----	----	-------

	Loc 15	37	1151	-102743.910227889
	Loc 13	37	1249	-86303.2334644601

In less than two minutes, ADX analyzed close to 20,000 time series and detected two abnormal time series in which the read count suddenly dropped.

These advanced capabilities combined with ADX fast performance supply a unique and powerful solution for time series analysis.

Next steps

- Learn about [Time series anomaly detection and forecasting](#) in Azure Data Explorer.
- Learn about [Machine learning capabilities](#) in Azure Data Explorer.

Anomaly detection and forecasting in Azure Data Explorer

12/3/2019 • 5 minutes to read • [Edit Online](#)

Azure Data Explorer performs on-going collection of telemetry data from cloud services or IoT devices. This data is analyzed for various insights such as monitoring service health, physical production processes, usage trends, and load forecast. The analysis is done on time series of selected metrics to locate a deviation pattern of the metric relative to its typical normal baseline pattern. Azure Data Explorer contains native support for creation, manipulation, and analysis of multiple time series. It can create and analyze thousands of time series in seconds, enabling near real time monitoring solutions and workflows.

This article details the Azure Data Explorer time series anomaly detection and forecasting capabilities. The applicable time series functions are based on a robust well-known decomposition model, where each original time series is decomposed into seasonal, trend, and residual components. Anomalies are detected by outliers on the residual component, while forecasting is done by extrapolating the seasonal and trend components. The Azure Data Explorer implementation significantly enhances the basic decomposition model by automatic seasonality detection, robust outlier analysis, and vectorized implementation to process thousands of time series in seconds.

Prerequisites

Read [Time series analysis in Azure Data Explorer](#) for an overview of time series capabilities.

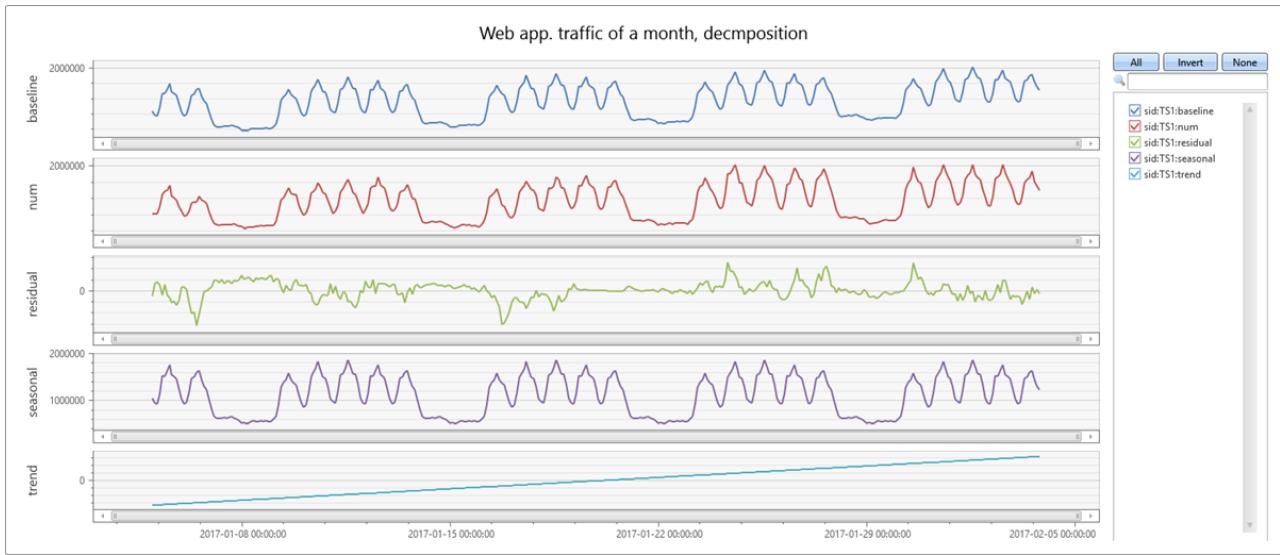
Time series decomposition model

Azure Data Explorer native implementation for time series prediction and anomaly detection uses a well-known decomposition model. This model is applied to time series of metrics expected to manifest periodic and trend behavior, such as service traffic, component heartbeats, and IoT periodic measurements to forecast future metric values and detect anomalous ones. The assumption of this regression process is that other than the previously known seasonal and trend behavior, the time series is randomly distributed. You can then forecast future metric values from the seasonal and trend components, collectively named baseline, and ignore the residual part. You can also detect anomalous values based on outlier analysis using only the residual portion. To create a decomposition model, use the function `series_decompose()`. The `series_decompose()` function takes a set of time series and automatically decomposes each time series to its seasonal, trend, residual, and baseline components.

For example, you can decompose traffic of an internal web service by using the following query:

[\[Click to run query\]](#)

```
let min_t = datetime(2017-01-05);
let max_t = datetime(2017-02-03 22:00);
let dt = 2h;
demo_make_series2
| make-series num=avg(num) on TimeStamp from min_t to max_t step dt by sid
| where sid == 'TS1'    // select a single time series for a cleaner visualization
| extend (baseline, seasonal, trend, residual) = series_decompose(num, -1, 'linefit') // decomposition of a
set of time series to seasonal, trend, residual, and baseline (seasonal+trend)
| render timechart with(title='Web app. traffic of a month, decomposition', ysplit=panels)
```



- The original time series is labeled **num** (in red).
- The process starts by auto detection of the seasonality by using the function `series_periods_detect()` and extracts the **seasonal** pattern (in purple).
- The seasonal pattern is subtracted from the original time series and a linear regression is run using the function `series_fit_line()` to find the **trend** component (in light blue).
- The function subtracts the trend and the remainder is the **residual** component (in green).
- Finally, the function adds the seasonal and trend components to generate the **baseline** (in blue).

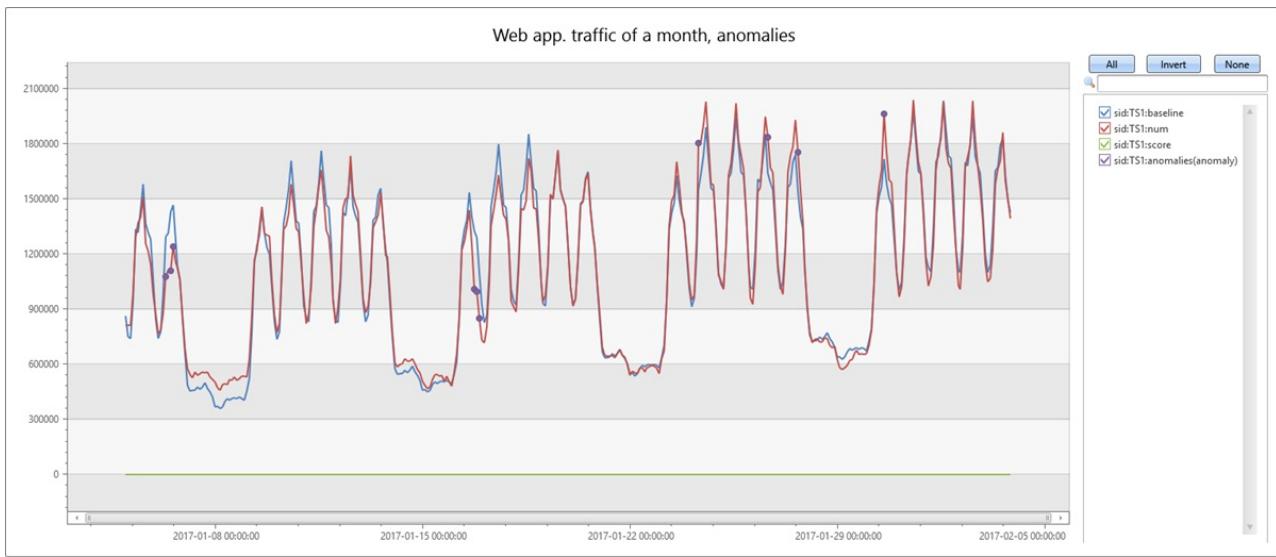
Time series anomaly detection

The function `series_decompose_anomalies()` finds anomalous points on a set of time series. This function calls `series_decompose()` to build the decomposition model and then runs `series_outliers()` on the residual component. `series_outliers()` calculates anomaly scores for each point of the residual component using Tukey's fence test. Anomaly scores above 1.5 or below -1.5 indicate a mild anomaly rise or decline respectively. Anomaly scores above 3.0 or below -3.0 indicate a strong anomaly.

The following query allows you to detect anomalies in internal web service traffic:

[\[Click to run query\]](#)

```
let min_t = datetime(2017-01-05);
let max_t = datetime(2017-02-03 22:00);
let dt = 2h;
demo_make_series2
| make-series num=avg(num) on TimeStamp from min_t to max_t step dt by sid
| where sid == 'TS1' // select a single time series for a cleaner visualization
| extend (anomalies, score, baseline) = series_decompose_anomalies(num, 1.5, -1, 'linefit')
| render anomalychart with(anomalycolumns=anomalies, title='Web app. traffic of a month, anomalies') //use "| render anomalychart with anomalycolumns=anomalies" to render the anomalies as bold points on the series charts.
```



- The original time series (in red).
- The baseline (seasonal + trend) component (in blue).
- The anomalous points (in purple) on top of the original time series. The anomalous points significantly deviate from the expected baseline values.

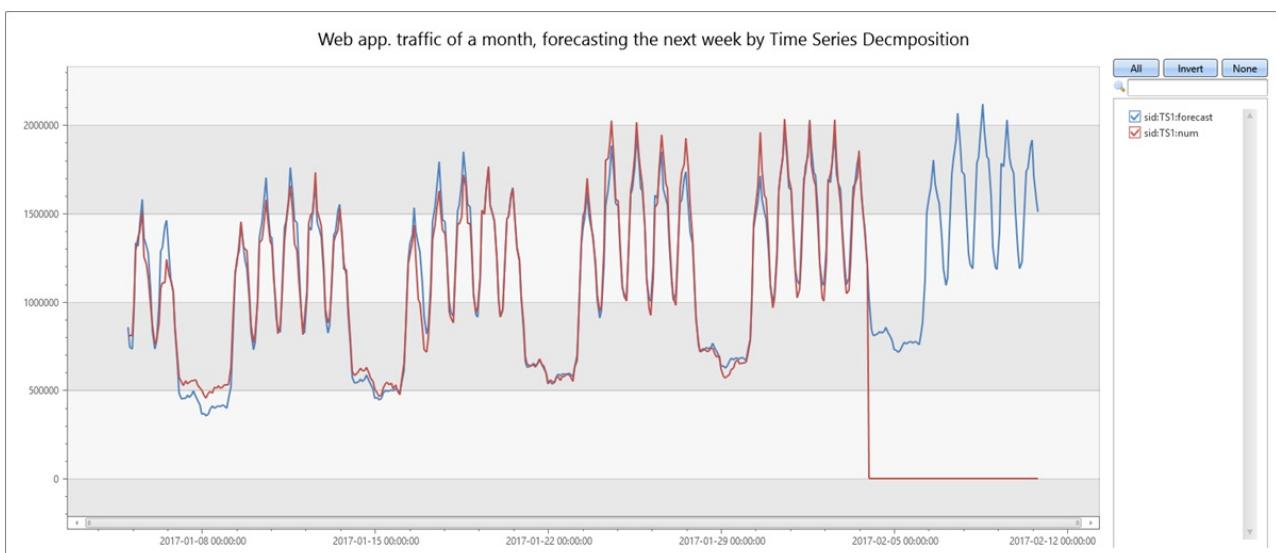
Time series forecasting

The function `series_decompose_forecast()` predicts future values of a set of time series. This function calls `series_decompose()` to build the decomposition model and then, for each time series, extrapolates the baseline component into the future.

The following query allows you to predict next week's web service traffic:

[\[Click to run query\]](#)

```
let min_t = datetime(2017-01-05);
let max_t = datetime(2017-02-03 22:00);
let dt = 2h;
let horizon=7d;
demo_make_series2
| make-series num=avg(num) on TimeStamp from min_t to max_t+horizon step dt by sid
| where sid == 'TS1' // select a single time series for a cleaner visualization
| extend forecast = series_decompose_forecast(num, toint(horizon/dt))
| render timechart with(title='Web app. traffic of a month, forecasting the next week by Time Series Decomposition')
```



- Original metric (in red). Future values are missing and set to 0, by default.
- Extrapolate the baseline component (in blue) to predict next week's values.

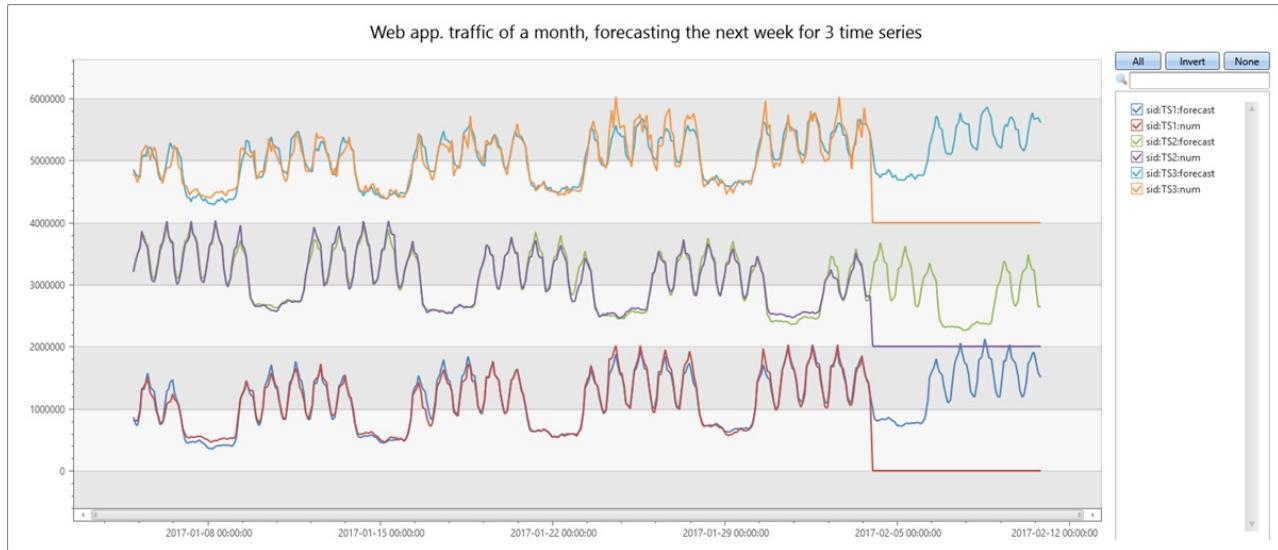
Scalability

Azure Data Explorer query language syntax enables a single call to process multiple time series. Its unique optimized implementation allows for fast performance, which is critical for effective anomaly detection and forecasting when monitoring thousands of counters in near real-time scenarios.

The following query shows the processing of three time series simultaneously:

[\[Click to run query\]](#)

```
let min_t = datetime(2017-01-05);
let max_t = datetime(2017-02-03 22:00);
let dt = 2h;
let horizon=7d;
demo_make_series2
| make-series num=avg(num) on TimeStamp from min_t to max_t+horizon step dt by sid
| extend offset=case(sid=='TS3', 4000000, sid=='TS2', 2000000, 0) // add artificial offset for easy
visualisation of multiple time series
| extend num=series_add(num, offset)
| extend forecast = series_decompose_forecast(num, toint(horizon/dt))
| render timechart with(title='Web app. traffic of a month, forecasting the next week for 3 time series')
```



Summary

This document details native Azure Data Explorer functions for time series anomaly detection and forecasting. Each original time series is decomposed into seasonal, trend and residual components for detecting anomalies and/or forecasting. These functionalities can be used for near real-time monitoring scenarios, such as fault detection, predictive maintenance, and demand and load forecasting.

Next steps

Learn about [Machine learning capabilities](#) in Azure Data Explorer.

Machine learning capability in Azure Data Explorer

12/3/2019 • 7 minutes to read • [Edit Online](#)

Azure Data Explorer, a Big Data analytics platform, is used to monitor service health, QoS, or malfunctioning devices for anomalous behavior using built-in [anomaly detection and forecasting](#) functions. Once an anomalous pattern is detected, a Root Cause Analysis (RCA) is performed to mitigate or resolve the anomaly.

The diagnosis process is complex and lengthy and done by domain experts. The process includes fetching and joining additional data from different sources for the same time frame, looking for changes in the distribution of values on multiple dimensions, charting additional variables, and other techniques based on domain knowledge and intuition. Since these diagnosis scenarios are common in Azure Data Explorer, machine learning plugins are available to make the diagnosis phase easier and shorten the duration of the RCA.

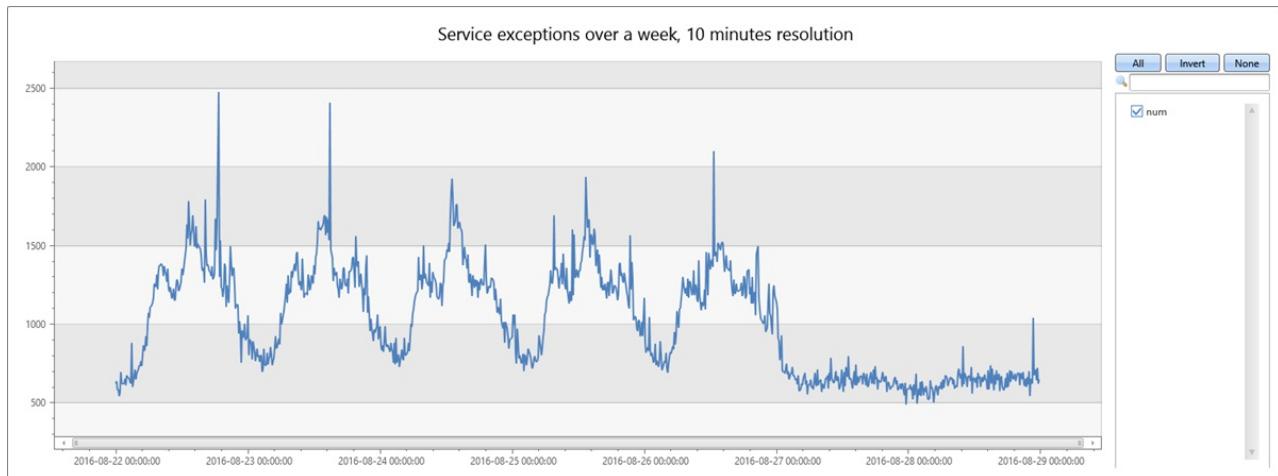
Azure Data Explorer has three Machine Learning plugins: `autocluster`, `basket`, and `diffpatterns`. All plugins implement clustering algorithms. The `autocluster` and `basket` plugins cluster a single record set and the `diffpatterns` plugin clusters the differences between two record sets.

Clustering a single record set

A common scenario includes a data set selected by a specific criteria such as time window that exhibits anomalous behavior, high temperature device readings, long duration commands, and top spending users. We would like an easy and fast way to find common patterns (segments) in the data. Patterns are a subset of the data set whose records share the same values over multiple dimensions (categorical columns). The following query builds and shows a time series of service exceptions over a week in ten-minute bins:

[\[Click to run query\]](#)

```
let min_t = toscalar(demo_clustering1 | summarize min(PreciseTimeStamp));
let max_t = toscalar(demo_clustering1 | summarize max(PreciseTimeStamp));
demo_clustering1
| make-series num=count() on PreciseTimeStamp from min_t to max_t step 10m
| render timechart with(title="Service exceptions over a week, 10 minutes resolution")
```

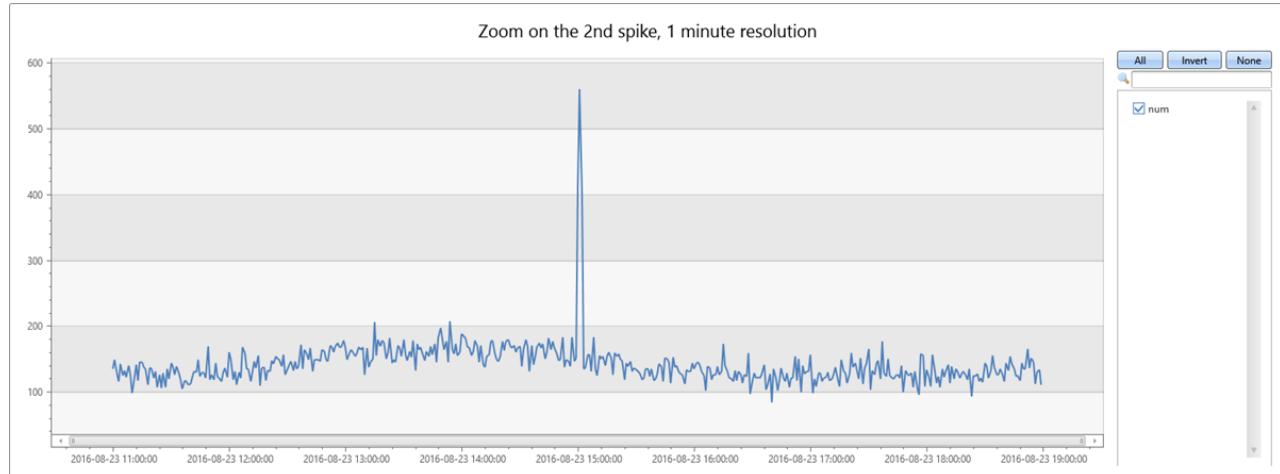


The service exception count correlates with the overall service traffic. You can clearly see the daily pattern, for business days of Monday to Friday, with a rise in service exception counts mid-day, and drops in counts during the night. Flat low counts are visible over the weekend. Exception spikes can be detected using [time series anomaly detection](#) in Azure Data Explorer.

The second spike in the data occurs on Tuesday afternoon. The following query is used to further diagnose this spike. Use the query to redraw the chart around the spike in higher resolution (eight hours in one-minute bins) to verify whether it's a sharp spike, and view its borders.

[\[Click to run query\]](#)

```
let min_t=datetime(2016-08-23 11:00);
demo_clustering1
| make-series num=count() on PreciseTimeStamp from min_t to min_t+8h step 1m
| render timechart with(title="Zoom on the 2nd spike, 1 minute resolution")
```



We see a narrow two-minute spike from 15:00 to 15:02. In the following query, count the exceptions in this two-minute window:

[\[Click to run query\]](#)

```
let min_peak_t=datetime(2016-08-23 15:00);
let max_peak_t=datetime(2016-08-23 15:02);
demo_clustering1
| where PreciseTimeStamp between(min_peak_t..max_peak_t)
| count
```

COUNT

972

In the following query, sample 20 exceptions out of 972:

[\[Click to run query\]](#)

```
let min_peak_t=datetime(2016-08-23 15:00);
let max_peak_t=datetime(2016-08-23 15:02);
demo_clustering1
| where PreciseTimeStamp between(min_peak_t..max_peak_t)
| take 20
```

PRECISETIMESTAMP	REGION	SCALEUNIT	DEPLOYMENTID	TRACEPOINT	SERVICEHOST
2016-08-23 15:00:08.7302460	scus	su5	9dbd1b161d5b4779a73cf19a7836ebd6	100005	00000000-0000-0000-0000-000000000000

PreciseTimestamp	Region	ScaleUnit	DeploymentID	TracePoint	ServiceHost
2016-08-23 15:00:09.9496584	scus	su5	9dbd1b161d5b4779a73cf19a7836ebd6	10007006	8d257da1-7a1c-44f5-9acd-f9e02ff507fd
2016-08-23 15:00:10.5911748	scus	su5	9dbd1b161d5b4779a73cf19a7836ebd6	100005	00000000-0000-0000-0000-000000000000
2016-08-23 15:00:12.2957912	scus	su5	9dbd1b161d5b4779a73cf19a7836ebd6	10007007	f855fcefbfe-405d-aaf8-9c5e2e43d862
2016-08-23 15:00:18.5955357	scus	su5	9dbd1b161d5b4779a73cf19a7836ebd6	10007006	9d390e07-417d-42eb-bebd-793965189a28
2016-08-23 15:00:20.7444854	scus	su5	9dbd1b161d5b4779a73cf19a7836ebd6	10007006	6e54c1c8-42d3-4e4e-8b79-9bb076ca71f1
2016-08-23 15:00:23.8694999	eus2	su2	89e2f62a73bb4ef d8f545aeae40d7e51	36109	19422243-19b9-4d85-9ca6-bc961861d287
2016-08-23 15:00:26.4271786	ncus	su1	e24ef436e02b4823ac5d5b1465a9401e	36109	3271bae4-1c5b-4f73-98ef-cc117e9be914
2016-08-23 15:00:27.8958124	scus	su3	90d3d2fc7ecc430c9621ece335651a01	904498	8cf38575-fca9-48ca-bd7c-21196f6d6765
2016-08-23 15:00:32.9884969	scus	su3	90d3d2fc7ecc430c9621ece335651a01	10007007	d5c7c825-9d46-4ab7-a0c1-8e2ac1d83ddb
2016-08-23 15:00:34.5061623	scus	su5	9dbd1b161d5b4779a73cf19a7836ebd6	1002110	55a71811-5ec4-497a-a058-140fb0d611ad
2016-08-23 15:00:37.4490273	scus	su3	90d3d2fc7ecc430c9621ece335651a01	10007006	f2ee8254-173c-477d-a1de-4902150ea50d
2016-08-23 15:00:41.2431223	scus	su3	90d3d2fc7ecc430c9621ece335651a01	103200	8cf38575-fca9-48ca-bd7c-21196f6d6765
2016-08-23 15:00:47.2983975	ncus	su1	e24ef436e02b4823ac5d5b1465a9401e	423690590	00000000-0000-0000-0000-000000000000
2016-08-23 15:00:50.5932834	scus	su5	9dbd1b161d5b4779a73cf19a7836ebd6	10007006	2a41b552-aa19-4987-8cdd-410a3af016ac

PRECISETIMESTAMP	REGION	SCALEUNIT	DEPLOYMENTID	TRACEPOINT	SERVICEHOST
2016-08-23 15:00:50.8259021	scus	su5	9dbd1b161d5b4779a73cf19a7836ebd6	1002110	0d56b8e3-470d-4213-91da-97405f8d005e
2016-08-23 15:00:53.2490731	scus	su5	9dbd1b161d5b4779a73cf19a7836ebd6	36109	55a71811-5ec4-497a-a058-140fb0d611ad
2016-08-23 15:00:57.0000946	eus2	su2	89e2f62a73bb4efd8f545aeae40d7e51	64038	cb55739e-4afe-46a3-970f-1b49d8ee7564
2016-08-23 15:00:58.2222707	scus	su5	9dbd1b161d5b4779a73cf19a7836ebd6	10007007	8215dcf6-2de0-42bd-9c90-181c70486c9c
2016-08-23 15:00:59.9382620	scus	su3	90d3d2fc7ecc430c9621ece335651a01	10007006	451e3c4c-0808-4566-a64d-84d85cf30978

Use autocluster() for single record set clustering

Even though there are less than a thousand exceptions, it's still hard to find common segments, as there are multiple values in each column. You can use `autocluster()` plugin to instantly extract a small list of common segments and find the interesting clusters within the spike's two minutes as seen in the following query:

[[Click to run query](#)]

```
let min_peak_t=datetime(2016-08-23 15:00);
let max_peak_t=datetime(2016-08-23 15:02);
demo_clustering1
| where PreciseTimeStamp between(min_peak_t..max_peak_t)
| evaluate autocluster()
```

SEGMENTID	COUNT	PERCENT	REGION	SCALEUNIT	DEPLOYMENTID	SERVICEHOST
0	639	65.7407407407407	eau	su7	b5d1d4df547d4a04ac15885617edba57	e7f60c5d-4944-42b3-922a-92e98a8e7dec
1	94	9.67078189300411	scus	su5	9dbd1b161d5b4779a73cf19a7836ebd6	
2	82	8.43621399176955	ncus	su1	e24ef436e02b4823ac5d5b1465a9401e	
3	68	6.99588477366255	scus	su3	90d3d2fc7ecc430c9621ece335651a01	

SEGMENTID	COUNT	PERCENT	REGION	SCALEUNIT	DEPLOYMENTID	SERVICEHOST
4	55	5.658436213 99177	weu	su4	be1d6d7ac95 74cbc9a22cb8 ee20f16fc	

You can see from the results above that the most dominant segment contains 65.74% of the total exception records and shares four dimensions. The next segment is much less common, contains only 9.67% of the records and shares three dimensions. The other segments are even less common.

Autocluster uses a proprietary algorithm for mining multiple dimensions and extracting interesting segments. "Interesting" means that each segment has significant coverage of both the records set and the features set. The segments are also diverged, meaning that each one is significantly different from the others. One or more of these segments may be relevant for the RCA process. To minimize segment review and assessment, autocluster extracts only a small segment list.

Use basket() for single record set clustering

You can also use the [basket\(\)](#) plugin as seen in the following query:

[\[Click to run query\]](#)

```
let min_peak_t=datetime(2016-08-23 15:00);
let max_peak_t=datetime(2016-08-23 15:02);
demo_clustering1
| where PreciseTimeStamp between(min_peak_t..max_peak_t)
| evaluate basket()
```

SEGMENTID	COUNT	PERCENT	REGION	SCALEUNIT	DEPLOYMENTID	TRACEPOINT	SERVICEHOST
0	639	65.7407407 407407	eau	su7	b5d1d4df5 47d4a04ac 15885617e dba57		e7f60c5d- 4944- 42b3-922a- 92e98a8e7 dec
1	642	66.0493827 160494	eau	su7	b5d1d4df5 47d4a04ac 15885617e dba57		
2	324	33.3333333 333333	eau	su7	b5d1d4df5 47d4a04ac 15885617e dba57	0	e7f60c5d- 4944- 42b3-922a- 92e98a8e7 dec
3	315	32.4074074 074074	eau	su7	b5d1d4df5 47d4a04ac 15885617e dba57	16108	e7f60c5d- 4944- 42b3-922a- 92e98a8e7 dec
4	328	33.7448559 670782				0	

SEGMENTID	COUNT	PERCENT	REGION	SCALEUNIT	DEPLOYMENTID	TRACEPOINT	SERVICEHOST
5	94	9.67078189 300411	scus	su5	9dbd1b161 d5b4779a7 3cf19a7836 ebd6		
6	82	8.43621399 176955	ncus	su1	e24ef436e0 2b4823ac5 d5b1465a9 401e		
7	68	6.99588477 366255	scus	su3	90d3d2fc7e cc430c9621 ece335651a 01		
8	167	17.1810699 588477	scus				
9	55	5.65843621 399177	weu	su4	be1d6d7ac 9574cbc9a2 2cb8ee20f1 6fc		
10	92	9.46502057 613169				10007007	
11	90	9.25925925 925926				10007006	
12	57	5.86419753 08642					00000000- 0000- 0000- 0000- 0000000000 000

Basket implements the Apriori algorithm for item set mining and extracts all segments whose coverage of the record set is above a threshold (default 5%). You can see that more segments were extracted with similar ones (for example, segments 0,1 or 2,3).

Both plugins are powerful and easy to use, but their significant limitation is that they cluster a single record set in an unsupervised manner (with no labels). It's therefore unclear whether the extracted patterns characterize the selected record set (the anomalous records) or the global record set.

Clustering the difference between two records sets

The `diffpatterns()` plugin overcomes the limitation of `autocluster` and `basket`. `Diffpatterns` takes two record sets and extracts the main segments that are different between them. One set usually contains the anomalous record set being investigated (one analyzed by `autocluster` and `basket`). The other set contains the reference record set (baseline).

In the query below, we use `diffpatterns` to find interesting clusters within the spike's two minutes, which are different than clusters within the baseline. We define the baseline window as the eight minutes before 15:00 (when the spike started). We also need to extend by a binary column (AB) specifying whether a specific record belongs to

the baseline or to the anomalous set. `Diffpatterns` implements a supervised learning algorithm, where the two class labels were generated by the anomalous versus the baseline flag (AB).

[\[Click to run query\]](#)

```
let min_peak_t=datetime(2016-08-23 15:00);
let max_peak_t=datetime(2016-08-23 15:02);
let min_baseline_t=datetime(2016-08-23 14:50);
let max_baseline_t=datetime(2016-08-23 14:58); // Leave a gap between the baseline and the spike to avoid the transition zone.
let splitime=(max_baseline_t+min_peak_t)/2.0;
demo_clustering1
| where (PreciseTimeStamp between(min_baseline_t..max_baseline_t)) or
(PreciseTimeStamp between(min_peak_t..max_peak_t))
| extend AB=iff(PreciseTimeStamp > splitime, 'Anomaly', 'Baseline')
| evaluate diffpatterns(AB, 'Anomaly', 'Baseline')
```

SEGMENTID	COUNTA	COUNTB	PERCENT A	PERCENT B	PERCENT DIFFAB	REGION	SCALEUNIT	DEPLOYMENTID	TRACEPOINT
0	639	21	65.74	1.7	64.04	eau	su7	b5d1d4 df547d4 a04ac15 885617e dba57	
1	167	544	17.18	44.16	26.97	scus			
2	92	356	9.47	28.9	19.43				100070 07
3	90	336	9.26	27.27	18.01				100070 06
4	82	318	8.44	25.81	17.38	ncus	su1	e24ef43 6e02b48 23ac5d5 b1465a9 401e	
5	55	252	5.66	20.45	14.8	weu	su4	be1d6d7 ac9574c bc9a22c b8ee20f 16fc	
6	57	204	5.86	16.56	10.69				

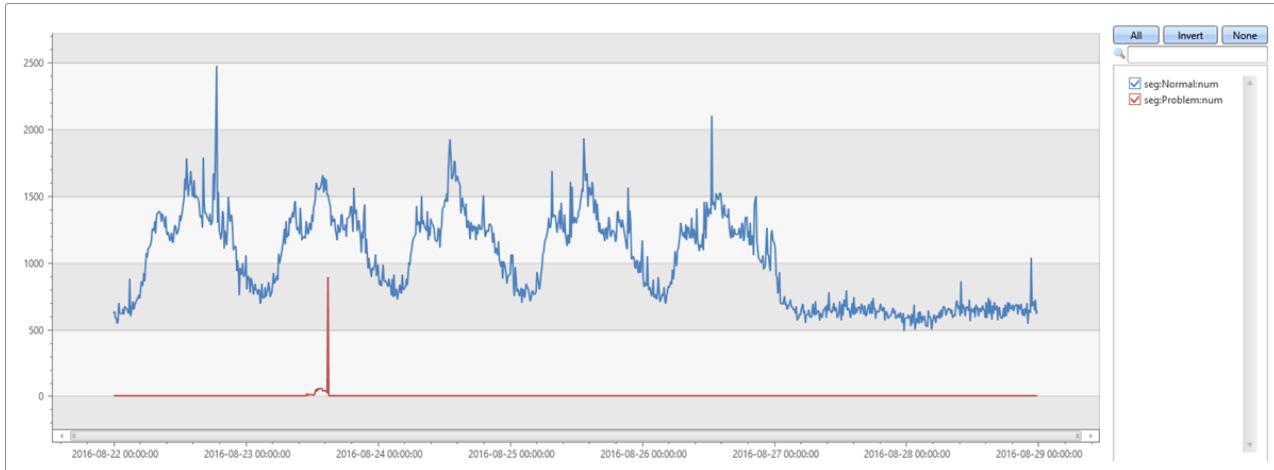
The most dominant segment is the same segment that was extracted by `autocluster`, its coverage on the two-minute anomalous window is also 65.74%. But its coverage on the eight-minute baseline window is only 1.7%. The difference is 64.04%. This difference seems to be related to the anomalous spike. You can verify this assumption by splitting the original chart into the records belonging to this problematic segment versus the other segments as seen in the query below:

[\[Click to run query\]](#)

```

let min_t = toscalar(demo_clustering1 | summarize min(PreciseTimeStamp));
let max_t = toscalar(demo_clustering1 | summarize max(PreciseTimeStamp));
demo_clustering1
| extend seg = iff(Region == "eau" and ScaleUnit == "su7" and DeploymentId ==
"b5d1d4df547d4a04ac15885617edba57"
and ServiceHost == "e7f60c5d-4944-42b3-922a-92e98a8e7dec", "Problem", "Normal")
| make-series num=count() on PreciseTimeStamp from min_t to max_t step 10m by seg
| render timechart

```



This chart allows us to see that the spike on Tuesday afternoon was because of exceptions from this specific segment, discovered by using the `diffpatterns` plugin.

Summary

The Azure Data Explorer Machine Learning plugins are helpful for many scenarios. The `autocluster` and `basket` implement unsupervised learning algorithm and are easy to use. `Diffpatterns` implements supervised learning algorithm and although more complex, it's more powerful in extracting differentiation segments for RCA.

These plugins are used interactively in ad-hoc scenarios and in automatic near real-time monitoring services. In Azure Data Explorer, time series anomaly detection is followed by a diagnosis process that is highly optimized to meet necessary performance standards.

Best practices for using Power BI to query and visualize Azure Data Explorer data

11/13/2019 • 5 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. [Power BI](#) is a business analytics solution that lets you visualize your data and share the results across your organization. Azure Data Explorer provides three options for connecting to data in Power BI. Use the [built-in connector](#), [import a query from Azure Data Explorer into Power BI](#), or use a [SQL query](#). This article supplies you with tips for querying and visualizing your Azure Data Explorer data with Power BI.

Best practices for using Power BI

When working with terabytes of fresh raw data, follow these guidelines to keep Power BI dashboards and reports snappy and updated:

- **Travel light** - Bring only the data that you need for your reports to Power BI. For deep interactive analysis, use the [Azure Data Explorer Web UI](#) that is optimized for ad-hoc exploration with the Kusto Query Language.
- **Composite model** - Use [composite model](#) to combine aggregated data for top-level dashboards with filtered operational raw data. You can clearly define when to use raw data and when to use an aggregated view.
- **Import mode versus DirectQuery mode** - Use **Import** mode for interaction of smaller data sets. Use **DirectQuery** mode for large, frequently updated data sets. For example, create dimension tables using **Import** mode since they're small and don't change often. Set the refresh interval according to the expected rate of data updates. Create fact tables using **DirectQuery** mode since these tables are large and contain raw data. Use these tables to present filtered data using Power BI [drillthrough](#).
- **Parallelism** – Azure Data Explorer is a linearly scalable data platform, therefore, you can improve the performance of dashboard rendering by increasing the parallelism of the end-to-end flow as follows:
 - Increase the number of [concurrent connections in DirectQuery in Power BI](#).
 - Use [weak consistency to improve parallelism](#). This may have an impact on the freshness of the data.
- **Effective slicers** – Use [sync slicers](#) to prevent reports from loading data before you're ready. After you structure the data set, place all visuals, and mark all the slicers, you can select the sync slicer to load only the data needed.
- **Use filters** - Use as many Power BI filters as possible to focus the Azure Data Explorer search on the relevant data shards.
- **Efficient visuals** – Select the most performant visuals for your data.

Tips for using the Azure Data Explorer connector for Power BI to query data

The following section includes tips and tricks for using Kusto query language with Power BI. Use the [Azure Data Explorer connector for Power BI](#) to visualize data

Complex queries in Power BI

Complex queries are more easily expressed in Kusto than in Power Query. They should be implemented as [Kusto functions](#), and invoked in Power BI. This method is required when using **DirectQuery** with `let` statements in your Kusto query. Because Power BI joins two queries, and `let` statements can't be used with the `join` operator, syntax errors may occur. Therefore, save each portion of the join as a Kusto function and allow Power BI to join these two functions together.

How to simulate a relative date-time operator

Power BI doesn't contain a *relative* date-time operator such as `ago()`. To simulate `ago()`, use a combination of `DateTime.FixedLocalNow()` and `#duration` Power BI functions.

Instead of this query using the `ago()` operator:

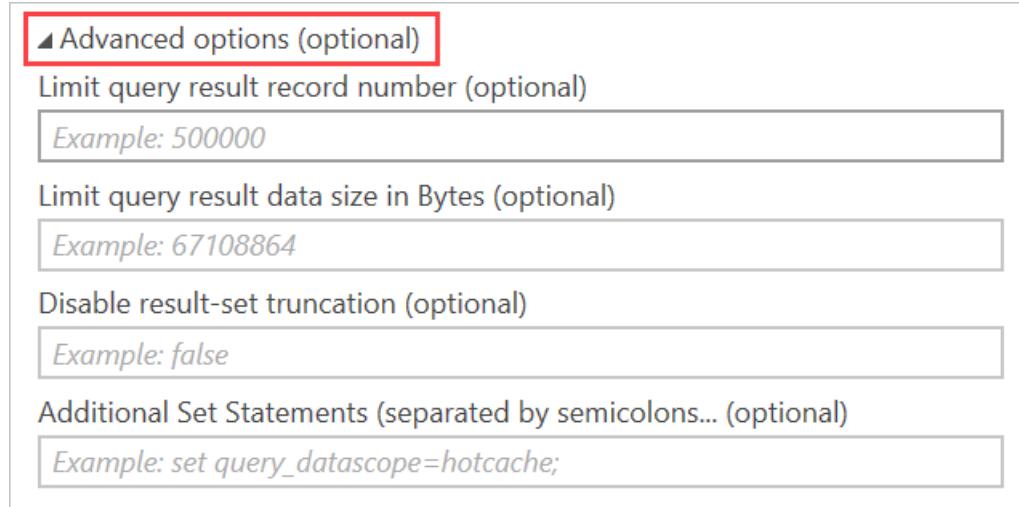
```
StormEvents | where StartTime > (now()-5d)  
StormEvents | where StartTime > ago(5d)
```

Use the following equivalent query:

```
let  
    Source = Kusto.Contents("help", "Samples", "StormEvents", []),  
    #"Filtered Rows" = Table.SelectRows(Source, each [StartTime] > (DateTime.FixedLocalNow() -  
#duration(5,0,0,0)))  
in  
    #"Filtered Rows"
```

Reaching Kusto query limits

Kusto queries return, by default, up to 500,000 rows or 64 MB, as described in [query limits](#). You can override these defaults by using **Advanced options** in the **Azure Data Explorer (Kusto)** connection window:



These options issue [set statements](#) with your query to change the default query limits:

- **Limit query result record number** generates a `set truncationmaxrecords`
- **Limit query result data size in Bytes** generates a `set truncationmaxsize`
- **Disable result-set truncation** generates a `set notruncation`

Using query parameters

You can use [query parameters](#) to modify your query dynamically.

Using a query parameter in the connection details

Use a query parameter to filter information in the query and optimize query performance.

In **Edit Queries** window, **Home > Advanced Editor**

- Find the following section of the query:

```
Source = Kusto.Contents("<Cluster>", "<Database>", "<Query>", [])
```

For example:

```
Source = Kusto.Contents("Help", "Samples", "StormEvents | where State == 'ALABAMA' | take 100", [])
```

- Replace the relevant part of the query with your parameter. Split the query into multiple parts, and concatenate them back using an ampersand (&), along with the parameter.

For example, in the query above, we'll take the `State == 'ALABAMA'` part, and split it to: `State == '` and `'` and we'll place the `State` parameter between them:

```
"StormEvents | where State == "" & State & " | take 100"
```

- If your query contains quotation marks, encode them correctly. For example, the following query:

```
"StormEvents | where State == "ALABAMA" | take 100"
```

will appear in the **Advanced Editor** as follows with two quotation marks:

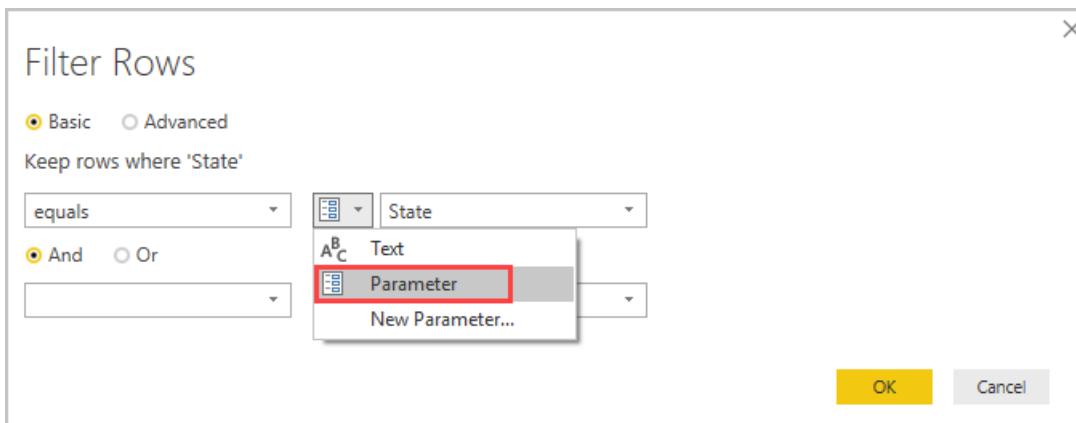
```
"StormEvents | where State == """ALABAMA"" | take 100"
```

It should be replaced with the following query with three quotation marks:

```
"StormEvents | where State == """ & State & """ | take 100"
```

Use a query parameter in the query steps

You can use a query parameter in any query step that supports it. For example, filter the results based on the value of a parameter.



Don't use Power BI data refresh scheduler to issue control commands to Kusto

Power BI includes a data refresh scheduler that can periodically issue queries against a data source. This mechanism shouldn't be used to schedule control commands to Kusto because Power BI assumes all queries are read-only.

Power BI can send only short (<2000 characters) queries to Kusto

If running a query in Power BI results in the following error: "DataSource.Error: Web.Contents failed to get

contents from..." the query is probably longer than 2000 characters. Power BI uses **PowerQuery** to query Kusto by issuing an HTTP GET request that encodes the query as part of the URI being retrieved. Therefore, Kusto queries issued by Power BI are limited to the maximum length of a request URI (2000 characters, minus small offset). As a workaround, you can define a [stored function](#) in Kusto, and have Power BI use that function in the query.

Next steps

[Visualize data using the Azure Data Explorer connector for Power BI](#)

Visualize data using the Azure Data Explorer connector for Power BI

11/13/2019 • 3 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Power BI is a business analytics solution that lets you visualize your data and share the results across your organization. Azure Data Explorer provides three options for connecting to data in Power BI: use the built-in connector, import a query from Azure Data Explorer, or use a SQL query. This article shows you how to use the built-in connector to get data and visualize it in a Power BI report. Using the Azure Data Explorer native connector for creating Power BI dashboards is straightforward. The Power BI connector supports [Import and Direct Query connectivity modes](#). You can build dashboards using **Import** or **DirectQuery** mode depending on the scenario, scale, and performance requirements.

Prerequisites

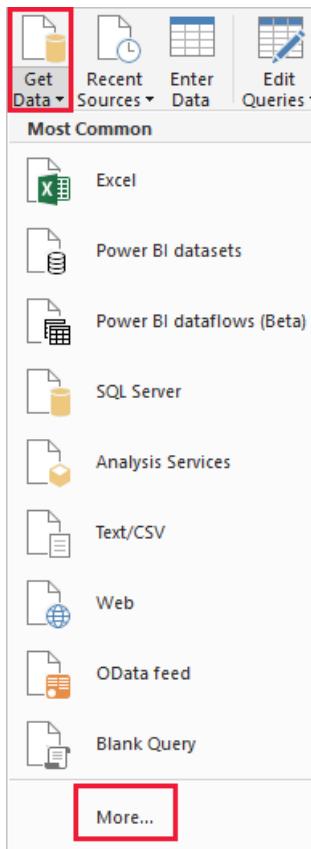
You need the following to complete this article:

- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.
- An organizational email account that is a member of Azure Active directory, so you can connect to the [Azure Data Explorer help cluster](#).
- [Power BI Desktop](#) (select **DOWNLOAD FREE**)

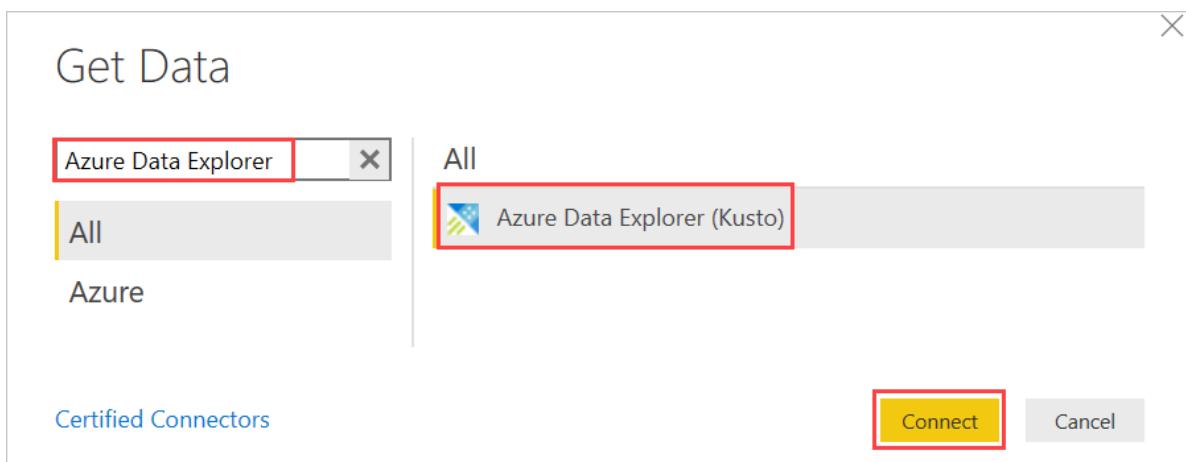
Get data from Azure Data Explorer

First, you connect to the Azure Data Explorer help cluster, then you bring in a subset of the data from the *StormEvents* table. The *StormEvents* sample data set contains weather-related data from the [National Centers for Environmental Information](#).

1. In Power BI Desktop, on the **Home** tab, select **Get Data** then **More**.



2. Search for **Azure Data Explorer**, select **Azure Data Explorer** then **Connect**.



3. On the **Azure Data Explorer (Kusto)** screen, fill out the form with the following information.

Azure Data Explorer (Kusto)

Cluster

<https://help.kusto.windows.net>

Database (optional)

Example: abc

Table name or Azure Data Explorer query (optional)

Example: Logs | where [Timestamp] > ago(1h)

▲ Advanced options (optional)

Limit query result record number (optional)

Example: 500000

Limit query result data size in Bytes (optional)

Example: 67108864

Disable result-set truncation (optional)

Example: false

Additional Set Statements (separated by semicolons... (optional)

Example: set query_datascope=hotcache;

Data Connectivity mode ⓘ

Import

DirectQuery

OK

Cancel

SETTING	VALUE	FIELD DESCRIPTION
Cluster	https://help.kusto.windows.net	The URL for the help cluster. For other clusters, the URL is in the form <code>https://<ClusterName>. <Region>.kusto.windows.net.</code>
Database	Leave blank	A database that is hosted on the cluster you're connecting to. We'll select this in a later step.
Table name	Leave blank	One of the tables in the database, or a query like <code>StormEvents take 1000</code> . We'll select this in a later step.
Advanced options	Leave blank	Options for your queries, such as result set size.
Data connectivity mode	<i>DirectQuery</i>	Determines whether Power BI imports the data or connects directly to the data source. You can use either option with this connector.

NOTE

In **Import** mode, data is moved to Power BI. In **DirectQuery** mode, data is queried directly from your Azure Data Explorer cluster.

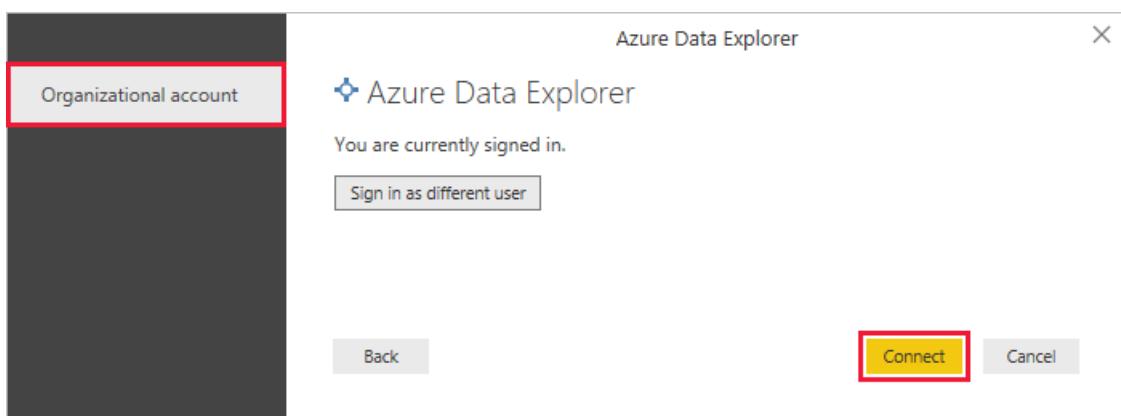
Use **Import** mode when:

- Your data set is small.
- You don't need near real-time data.
- Your data is already aggregated or you perform [aggregation in Kusto](#)

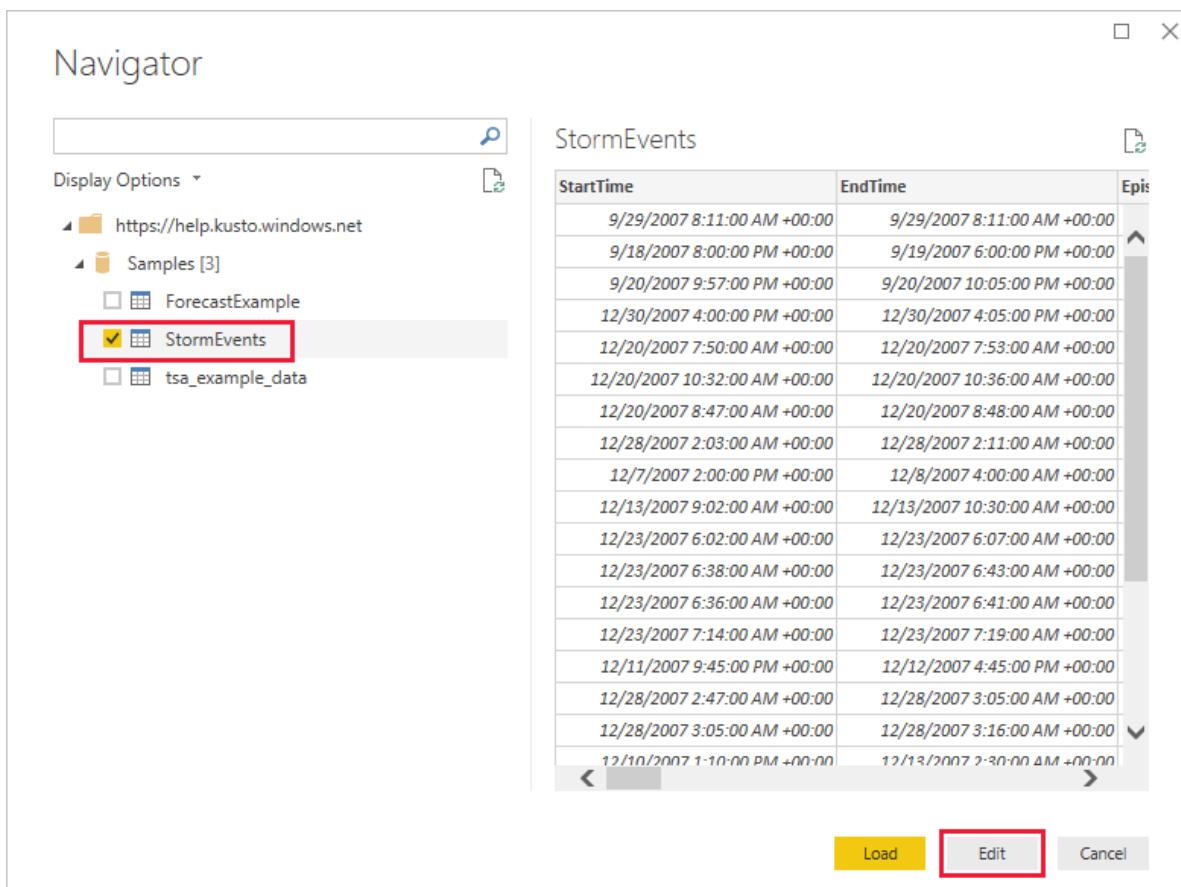
Use **DirectQuery** mode when:

- Your data set is very large.
- You need near real-time data.

4. If you don't already have a connection to the help cluster, sign in. Sign in with an organizational account, then select **Connect**.



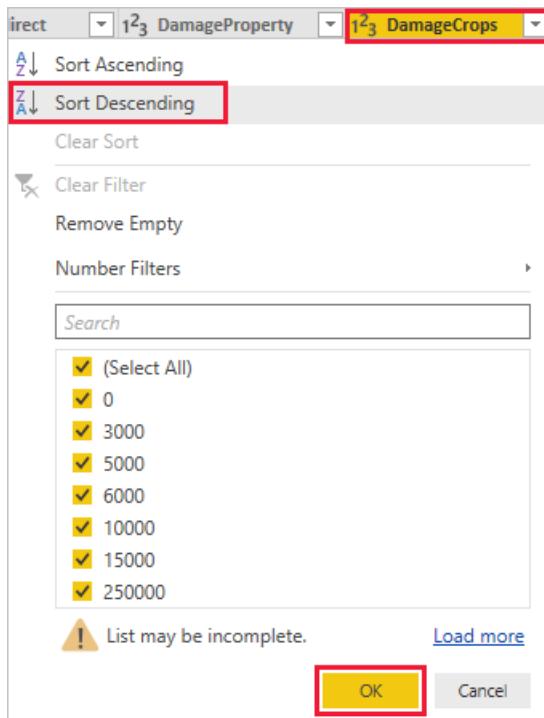
5. On the **Navigator** screen, expand the **Samples** database, select **StormEvents** then **Edit**.



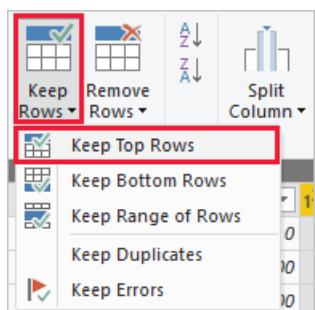
StartTime	EndTime	Episode
9/29/2007 8:11:00 AM +00:00	9/29/2007 8:11:00 AM +00:00	
9/18/2007 8:00:00 PM +00:00	9/19/2007 6:00:00 PM +00:00	
9/20/2007 9:57:00 PM +00:00	9/20/2007 10:05:00 PM +00:00	
12/30/2007 4:00:00 PM +00:00	12/30/2007 4:05:00 PM +00:00	
12/20/2007 7:50:00 AM +00:00	12/20/2007 7:53:00 AM +00:00	
12/20/2007 10:32:00 AM +00:00	12/20/2007 10:36:00 AM +00:00	
12/20/2007 8:47:00 AM +00:00	12/20/2007 8:48:00 AM +00:00	
12/28/2007 2:03:00 AM +00:00	12/28/2007 2:11:00 AM +00:00	
12/7/2007 2:00:00 PM +00:00	12/8/2007 4:00:00 AM +00:00	
12/13/2007 9:02:00 AM +00:00	12/13/2007 10:30:00 AM +00:00	
12/23/2007 6:02:00 AM +00:00	12/23/2007 6:07:00 AM +00:00	
12/23/2007 6:38:00 AM +00:00	12/23/2007 6:43:00 AM +00:00	
12/23/2007 6:36:00 AM +00:00	12/23/2007 6:41:00 AM +00:00	
12/23/2007 7:14:00 AM +00:00	12/23/2007 7:19:00 AM +00:00	
12/11/2007 9:45:00 PM +00:00	12/12/2007 4:45:00 PM +00:00	
12/28/2007 2:47:00 AM +00:00	12/28/2007 3:05:00 AM +00:00	
12/28/2007 3:05:00 AM +00:00	12/28/2007 3:16:00 AM +00:00	
12/10/2007 1:10:00 PM +00:00	12/13/2007 2:30:00 AM +00:00	

The table opens in Power Query Editor, where you can edit rows and columns before importing the data.

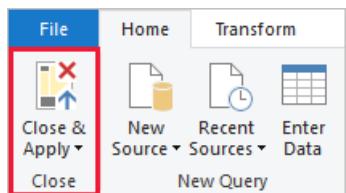
6. In the Power Query Editor, select the arrow next to the **DamageCrops** column then **Sort descending**.



7. On the **Home** tab, select **Keep Rows** then **Keep Top Rows**. Enter a value of **1000** to bring in the top 1000 rows of the sorted table.



8. On the **Home** tab, select **Close & Apply**.



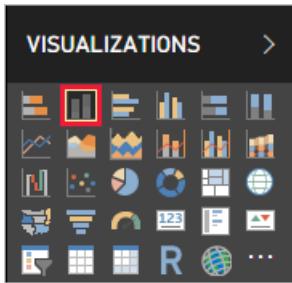
Visualize data in a report

Now that you have data in Power BI Desktop, you can create reports based on that data. You'll create a simple report with a column chart that shows crop damage by state.

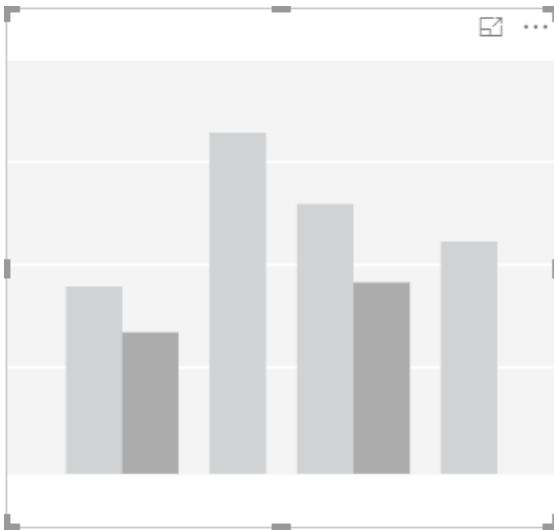
1. On the left side of the main Power BI window, select the report view.



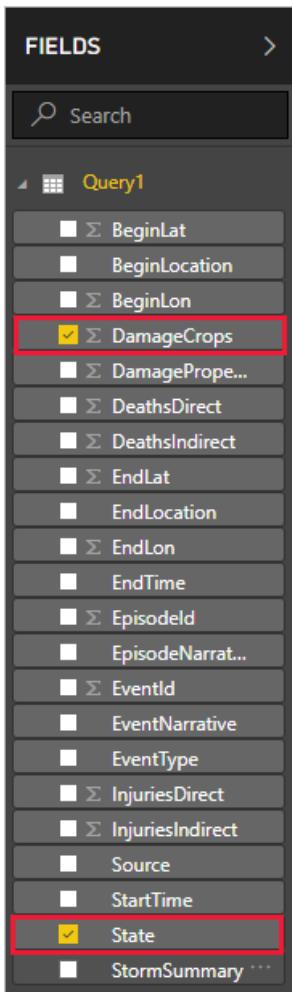
2. In the **VISUALIZATIONS** pane, select the clustered column chart.



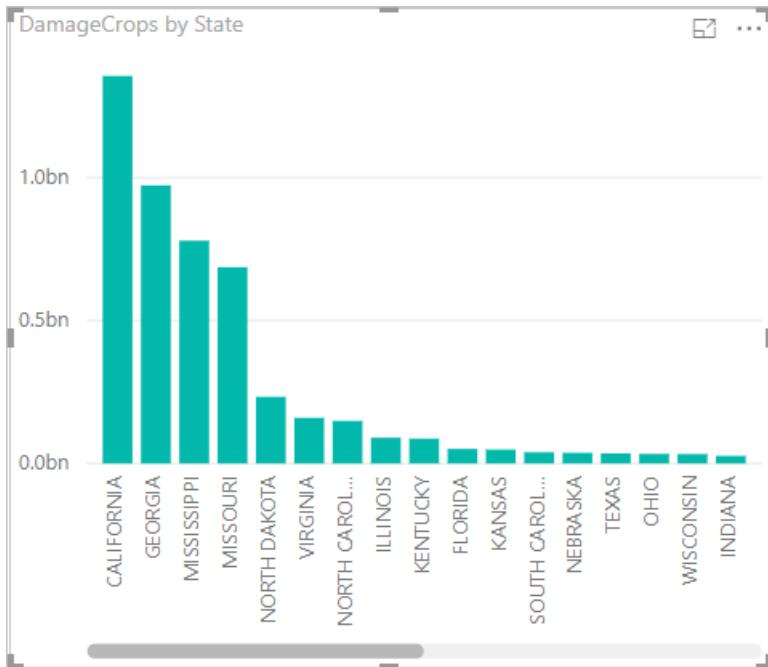
A blank chart is added to the canvas.



3. In the **FIELDS** list, select **DamageCrops** and **State**.



You now have a chart that shows the damage to crops for the top 1000 rows in the table.



4. Save the report.

Clean up resources

If you no longer need the report you created for this article, delete the Power BI Desktop (.pbix) file.

Next steps

[Tips for using the Azure Data Explorer connector for Power BI to query data](#)

Visualize data using a query imported into Power BI

7/11/2019 • 2 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Power BI is a business analytics solution that lets you visualize your data and share the results across your organization.

Azure Data Explorer provides three options for connecting to data in Power BI: use the built-in connector, import a query from Azure Data Explorer, or use a SQL query. This article shows you how to import a query so that you can get data and visualize it in a Power BI report.

If you don't have an Azure subscription, create a [free Azure account](#) before you begin.

Prerequisites

You need the following to complete this article:

- An organizational email account that is a member of Azure Active directory, so you can connect to the [Azure Data Explorer help cluster](#).
- [Power BI Desktop](#) (select **DOWNLOAD FREE**)
- [Azure Data Explorer desktop app](#)

Get data from Azure Data Explorer

First, you create a query in the Azure Data Explorer desktop app and export it for use in Power BI. Then, you connect to the Azure Data Explorer help cluster, and bring in a subset of the data from the *StormEvents* table. The *StormEvents* sample data set contains weather-related data from the [National Centers for Environmental Information](#).

1. In a browser, go to <https://help.kusto.windows.net/> to launch the Azure Data Explorer desktop app.
2. In the desktop app, copy the following query into the upper-right query window then run it.

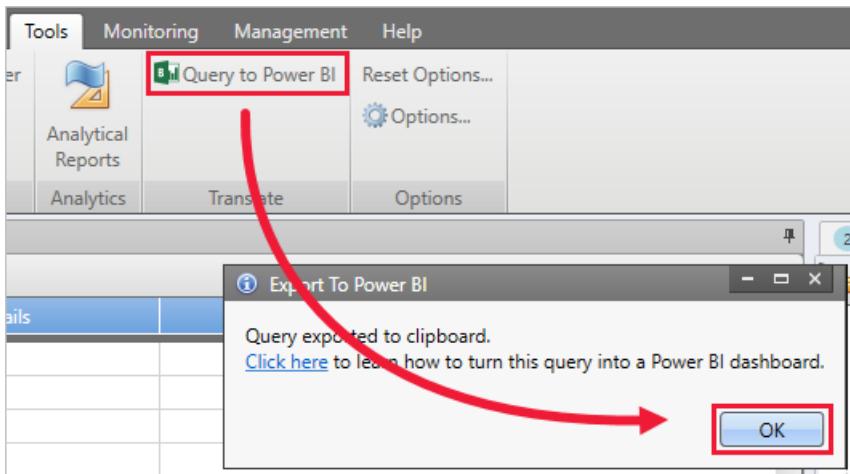
```
StormEvents
| sort by DamageCrops desc
| take 1000
```

The first few rows of the result set should look similar to the following image.

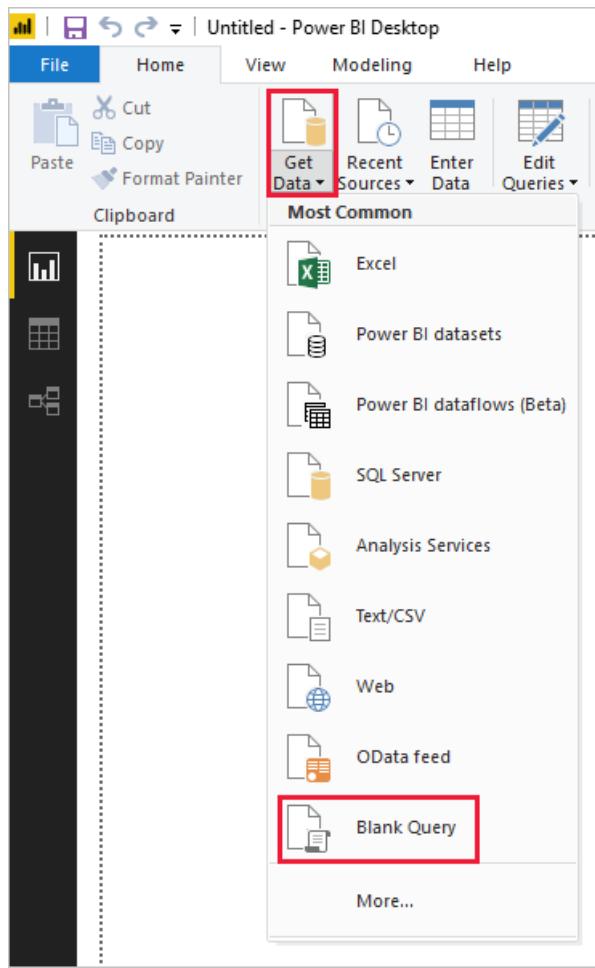
1 StormEvents
2 | sort by DamageCrops desc
3 | take 1000

StartTime	EndTime	Episodeld	EventId	State	EventType
2007-06-01 00:00:00.0000000	2007-06-30 23:59:00.0000000	7401	42508	MISSISSIPPI	Drought
2007-01-11 22:00:00.0000000	2007-01-24 10:00:00.0000000	1030	13721	CALIFORNIA	Frost/Freeze
2007-07-01 00:00:00.0000000	2007-07-03 15:45:00.0000000	12456	68350	MISSOURI	Flood
2007-09-01 00:00:00.0000000	2007-09-30 23:59:00.0000000	11187	61331	GEORGIA	Drought
2007-04-07 04:00:00.0000000	2007-04-09 09:00:00.0000000	5504	36585	GEORGIA	Frost/Freeze
2007-04-07 04:00:00.0000000	2007-04-09 09:00:00.0000000	5504	36584	GEORGIA	Frost/Freeze
2007-04-07 04:00:00.0000000	2007-04-09 09:00:00.0000000	5504	32414	GEORGIA	Frost/Freeze
2007-04-07 04:00:00.0000000	2007-04-09 09:00:00.0000000	5504	36604	GEORGIA	Frost/Freeze
2007-01-11 22:00:00.0000000	2007-01-24 10:00:00.0000000	1030	4407	CALIFORNIA	Frost/Freeze
2007-01-13 22:00:00.0000000	2007-01-14 09:00:00.0000000	2103	10237	CALIFORNIA	Frost/Freeze

3. On the **Tools** tab, select **Query to Power BI** then **OK**.



4. In Power BI Desktop, on the **Home** tab, select **Get Data** then **Blank query**.



5. In the Power Query Editor, on the **Home** tab, select **Advanced editor**.
6. In the **Advanced editor** window, paste the query you exported then select **Done**.

```

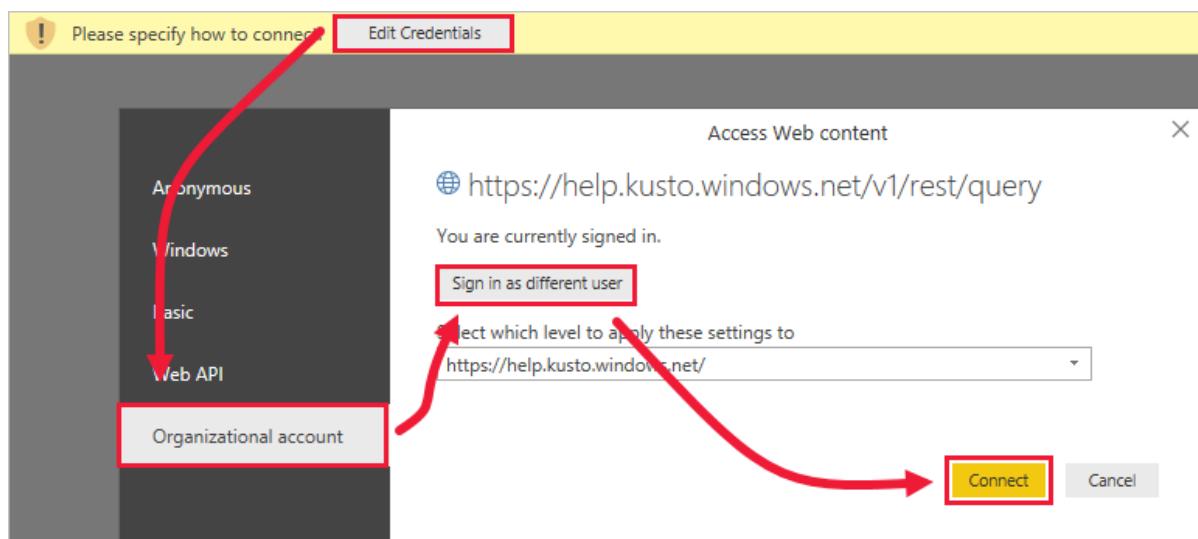
let KustoQuery =
    let Source = Json.Document(Web.Contents("https://help.kusto.windows.net:"))
    TypeMap = #table(
        { "DataType", "Type" },
        {
            { "Double", Double.Type },
            { "Int64", Int64.Type },
            { "Int32", Int32.Type },
            { "Int16", Int16.Type },
            { "UInt64", Number.Type },
            { "UInt32", Number.Type },
            { "UInt16", Number.Type },
            { "Byte", Byte.Type },
            { "Single", Single.Type },
            { "Decimal", Decimal.Type },
            { "TimeSpan", Duration.Type },
            { "DateTime", DateTimeZone.Type },
            { "String", Text.Type },
            { "Boolean", Logical.Type },
            { "SByte", Logical.Type },
            { "Guid", Text.Type }
        })
    in TypeMap
)

```

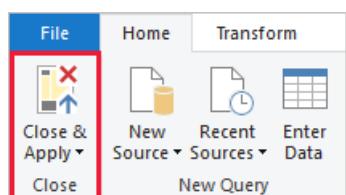
✓ No syntax errors have been detected.

Done **Cancel**

- In the main Power Query Editor window, select **Edit credentials**. Select **Organizational account**, sign in, then select **Connect**.



- On the **Home** tab, select **Close & Apply**.



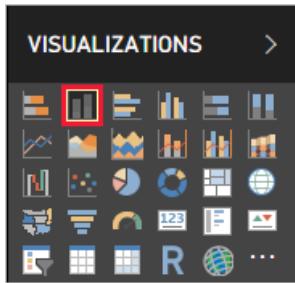
Visualize data in a report

Now that you have data in Power BI Desktop, you can create reports based on that data. You'll create a simple report with a column chart that shows crop damage by state.

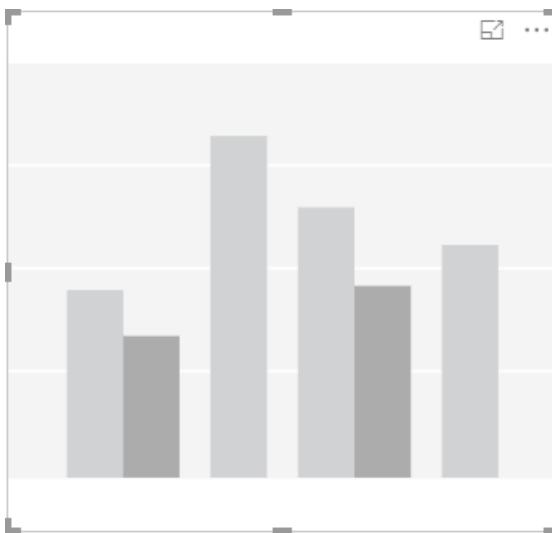
1. On the left side of the main Power BI window, select the report view.



2. In the **VISUALIZATIONS** pane, select the clustered column chart.



A blank chart is added to the canvas.



3. In the **FIELDS** list, select **DamageCrops** and **State**.

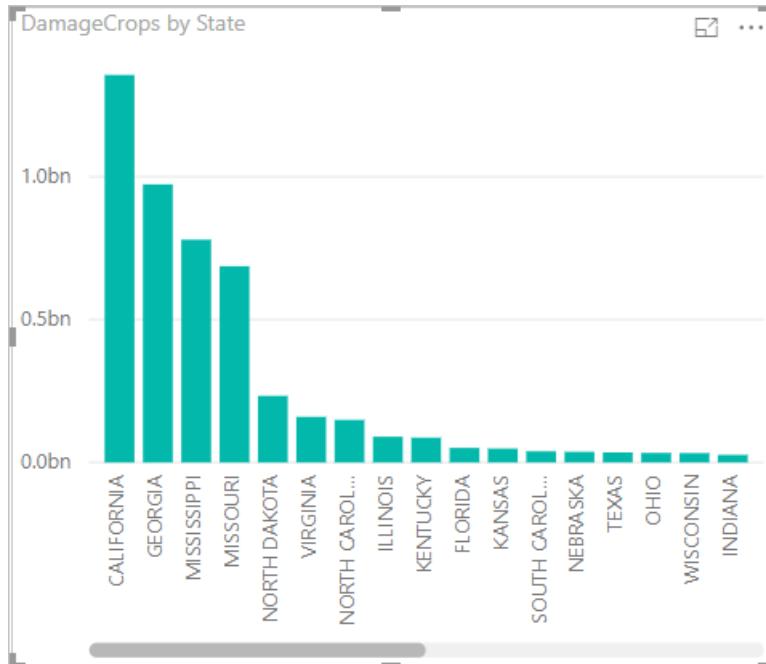
FIELDS

Search

Query1

- Σ BeginLat
- BeginLocation
- Σ BeginLon
- Σ DamageCrops
- Σ DamagePrope...
- Σ DeathsDirect
- Σ DeathsIndirect
- Σ EndLat
- EndLocation
- Σ EndLon
- EndTime
- Σ EpisodeId
- EpisodeNarrat...
- Σ EventId
- EventNarrative
- EventType
- Σ InjuriesDirect
- Σ InjuriesIndirect
- Source
- StartTime
- State
- StormSummary

You now have a chart that shows the damage to crops for the top 1000 rows in the table.



- Save the report.

Clean up resources

If you no longer need the report you created for this article, delete the Power BI Desktop (.pbix) file.

Next steps

Visualize data using the Azure Data Explorer connector for Power BI

Visualize data from Azure Data Explorer using a SQL query in Power BI

11/13/2019 • 2 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Power BI is a business analytics solution that lets you visualize your data and share the results across your organization.

Azure Data Explorer provides three options for connecting to data in Power BI: use the built-in connector, import a query from Azure Data Explorer, or use a SQL query. This article shows you how to use a SQL query to get data and visualize it in a Power BI report.

If you don't have an Azure subscription, create a [free Azure account](#) before you begin.

Prerequisites

You need the following to complete this article:

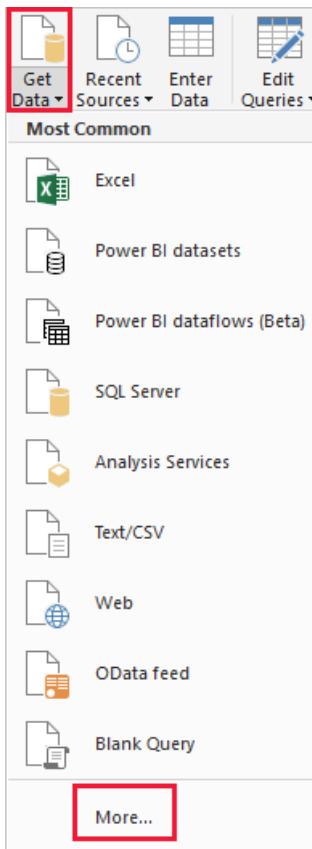
- An organizational email account that is a member of Azure Active Directory, so you can connect to the [Azure Data Explorer help cluster](#).
- [Power BI Desktop](#) (select **DOWNLOAD FREE**)

Get data from Azure Data Explorer

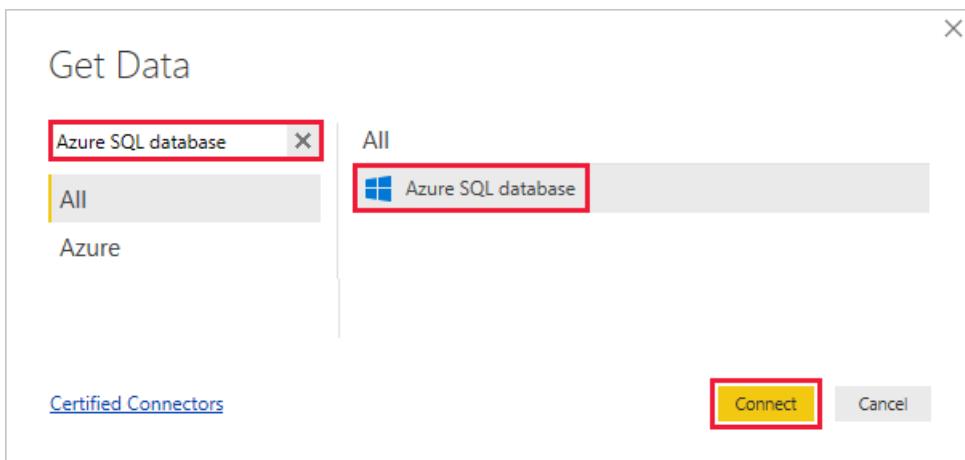
First, you connect to the Azure Data Explorer help cluster, then you bring in a subset of the data from the *StormEvents* table. The *StormEvents* sample data set contains weather-related data from the [National Centers for Environmental Information](#).

You typically use the native query language with Azure Data Explorer, but it also supports SQL queries, which you'll use here. Azure Data Explorer translates the SQL query into a native query for you.

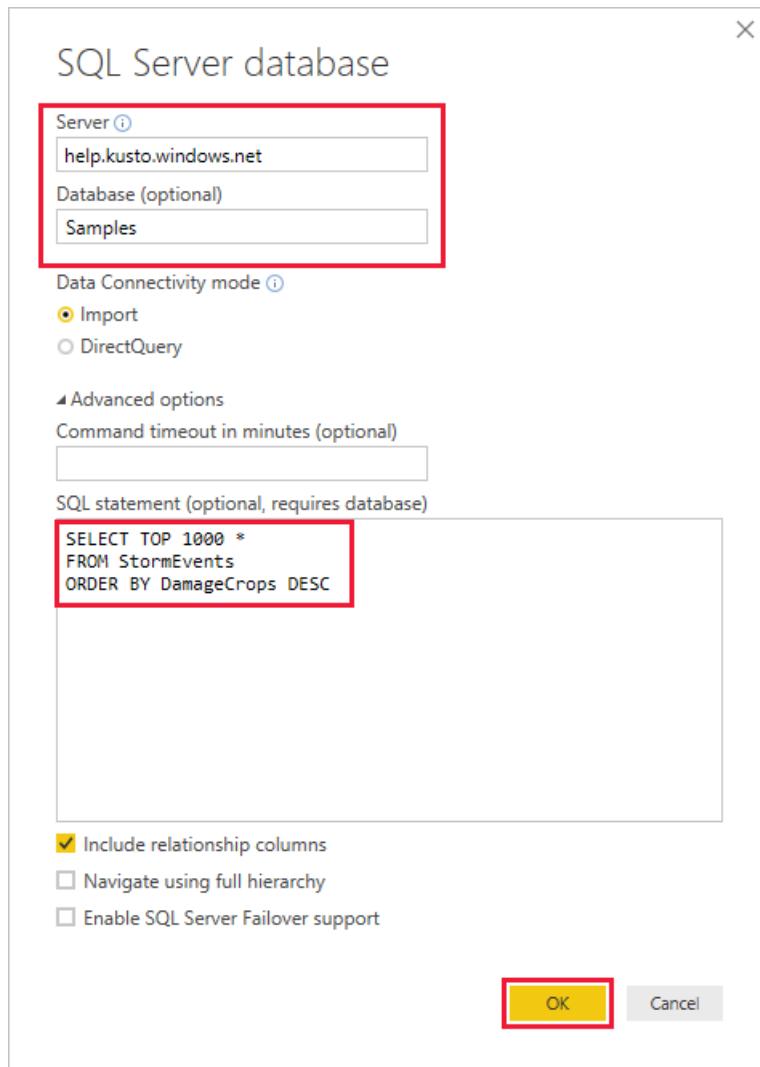
1. In Power BI Desktop, on the **Home** tab, select **Get Data** then **More**.



2. Search for *Azure SQL Database*, select **Azure SQL Database** then **Connect**.



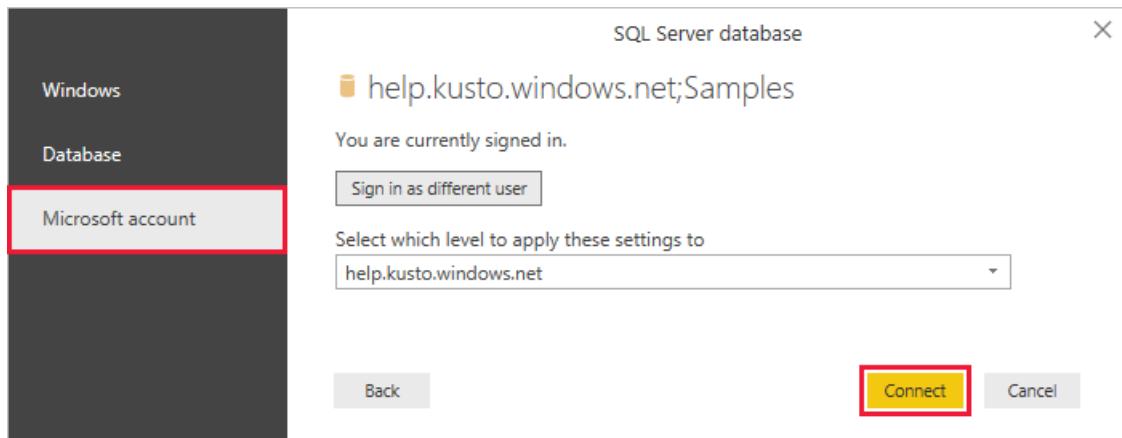
3. On the **SQL Server database** screen, fill out the form with the following information.



SETTING	VALUE	FIELD DESCRIPTION
Server	help.kusto.windows.net	The URL for the help cluster (without <code>https://</code>). For other clusters, the URL is in the form <code><ClusterName>.kusto.windows.net</code> .
Database	Samples	The sample database that is hosted on the cluster you're connecting to.
Data connectivity mode	Import	Determines whether Power BI imports the data or connects directly to the data source. You can use either option with this connector.
Command timeout	Leave blank	How long the query runs before it throws a timeout error.
SQL statement	Copy the query below this table	The SQL statement that Azure Data Explorer translates into a native query.
Other options	Leave as default values	Options don't apply to Azure Data Explorer clusters.

```
SELECT TOP 1000 *
FROM StormEvents
ORDER BY DamageCrops DESC
```

4. If you don't already have a connection to the help cluster, sign in. Sign in with a Microsoft account, then select **Connect**.



5. On the **help.kusto.windows.net: Samples** screen, select **Load**.

StartTime	EndTime	EpisodeId	EventId	State	EventType
6/1/2007 12:00:00 AM	6/30/2007 11:59:00 PM	7401	42508	MISSISSIPPI	Drought
1/11/2007 10:00:00 PM	1/24/2007 10:00:00 AM	1030	13721	CALIFORNIA	Frost/Freeze
7/1/2007 12:00:00 AM	7/3/2007 3:45:00 PM	12456	68350	MISSOURI	Flood
9/1/2007 12:00:00 AM	9/30/2007 11:59:00 PM	11187	61331	GEORGIA	Drought
4/7/2007 4:00:00 AM	4/9/2007 9:00:00 AM	5504	36585	GEORGIA	Frost/Freeze
4/7/2007 4:00:00 AM	4/9/2007 9:00:00 AM	5504	36584	GEORGIA	Frost/Freeze
4/7/2007 4:00:00 AM	4/9/2007 9:00:00 AM	5504	32414	GEORGIA	Frost/Freeze
4/7/2007 4:00:00 AM	4/9/2007 9:00:00 AM	5504	36604	GEORGIA	Frost/Freeze
1/11/2007 10:00:00 PM	1/24/2007 10:00:00 AM	1030	4407	CALIFORNIA	Frost/Freeze
1/13/2007 10:00:00 PM	1/14/2007 9:00:00 AM	2103	10237	CALIFORNIA	Frost/Freeze
1/13/2007 10:00:00 PM	1/14/2007 9:00:00 AM	2103	10239	CALIFORNIA	Frost/Freeze
1/13/2007 10:00:00 PM	1/14/2007 9:00:00 AM	2103	10238	CALIFORNIA	Frost/Freeze
1/13/2007 12:00:00 AM	1/18/2007 3:00:00 PM	2477	12530	CALIFORNIA	Frost/Freeze
1/14/2007 12:00:00 AM	1/18/2007 3:00:00 PM	2477	12533	CALIFORNIA	Frost/Freeze
2/17/2007 5:30:00 AM	2/17/2007 7:30:00 AM	2384	11766	FLORIDA	Frost/Freeze
1/14/2007 12:00:00 AM	1/18/2007 3:00:00 PM	2477	12529	CALIFORNIA	Frost/Freeze
1/11/2007 10:00:00 PM	1/24/2007 10:00:00 AM	1030	4408	CALIFORNIA	Frost/Freeze
1/14/2007 12:00:00 AM	1/18/2007 3:00:00 PM	2477	12534	CALIFORNIA	Frost/Freeze
7/15/2007 6:20:00 PM	7/15/2007 6:20:00 PM	7967	48490	NORTH DAKOTA	Hail
4/7/2007 10:00:00 PM	4/9/2007 7:00:00 AM	5196	30492	KANSAS	Frost/Freeze

At the bottom of the table are navigation arrows and buttons for 'Load' (highlighted with a yellow box), 'Edit', and 'Cancel'.

The table opens in the main Power BI window, in report view, where you can create reports based on the sample data.

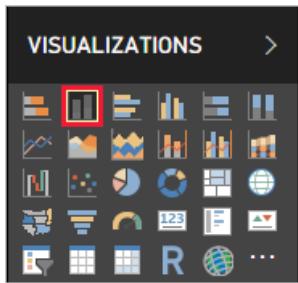
Visualize data in a report

Now that you have data in Power BI Desktop, you can create reports based on that data. You'll create a simple report with a column chart that shows crop damage by state.

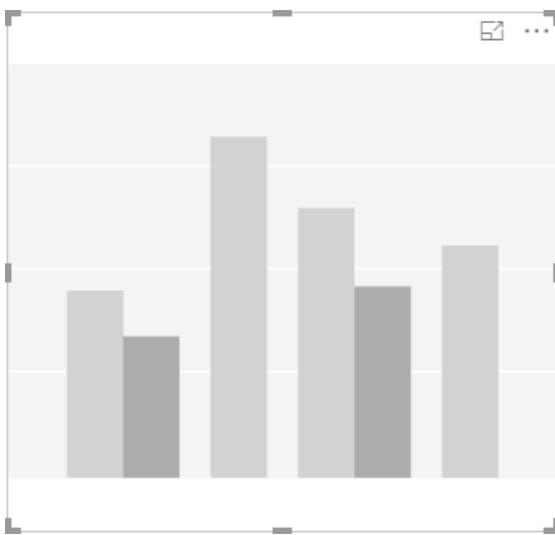
1. On the left side of the main Power BI window, select the report view.



2. In the **VISUALIZATIONS** pane, select the clustered column chart.



A blank chart is added to the canvas.



3. In the **FIELDS** list, select **DamageCrops** and **State**.

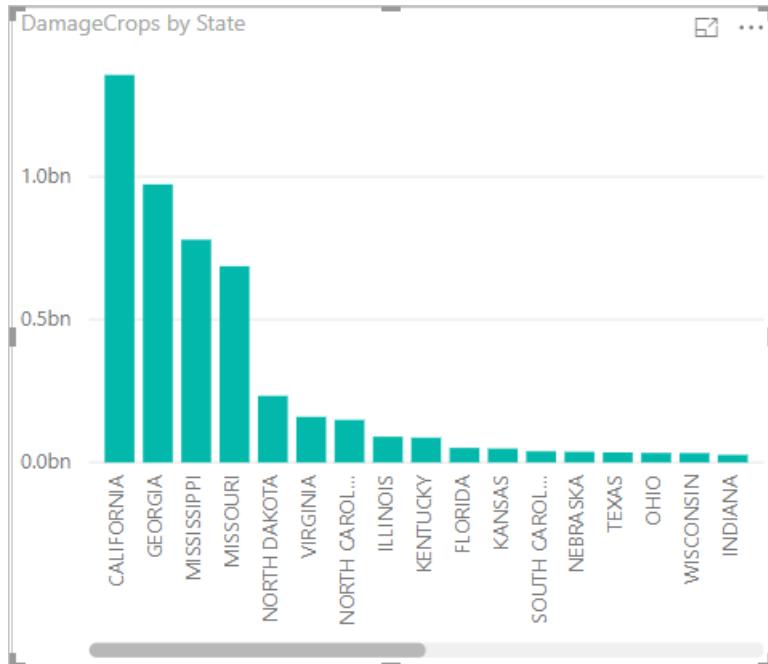
FIELDS

Search

Query1

- Σ BeginLat
- BeginLocation
- Σ BeginLon
- Σ DamageCrops
- Σ DamagePrope...
- Σ DeathsDirect
- Σ DeathsIndirect
- Σ EndLat
- EndLocation
- Σ EndLon
- EndTime
- Σ EpisodeId
- EpisodeNarrat...
- Σ EventId
- EventNarrative
- EventType
- Σ InjuriesDirect
- Σ InjuriesIndirect
- Source
- StartTime
- State
- StormSummary

You now have a chart that shows the damage to crops for the top 1000 rows in the table.



- Save the report.

Clean up resources

If you no longer need the report you created for this article, delete the Power BI Desktop (.pbix) file.

Next steps

Visualize data using the Azure Data Explorer connector for Power BI

Visualize data using the Azure Data Explorer connector for Excel

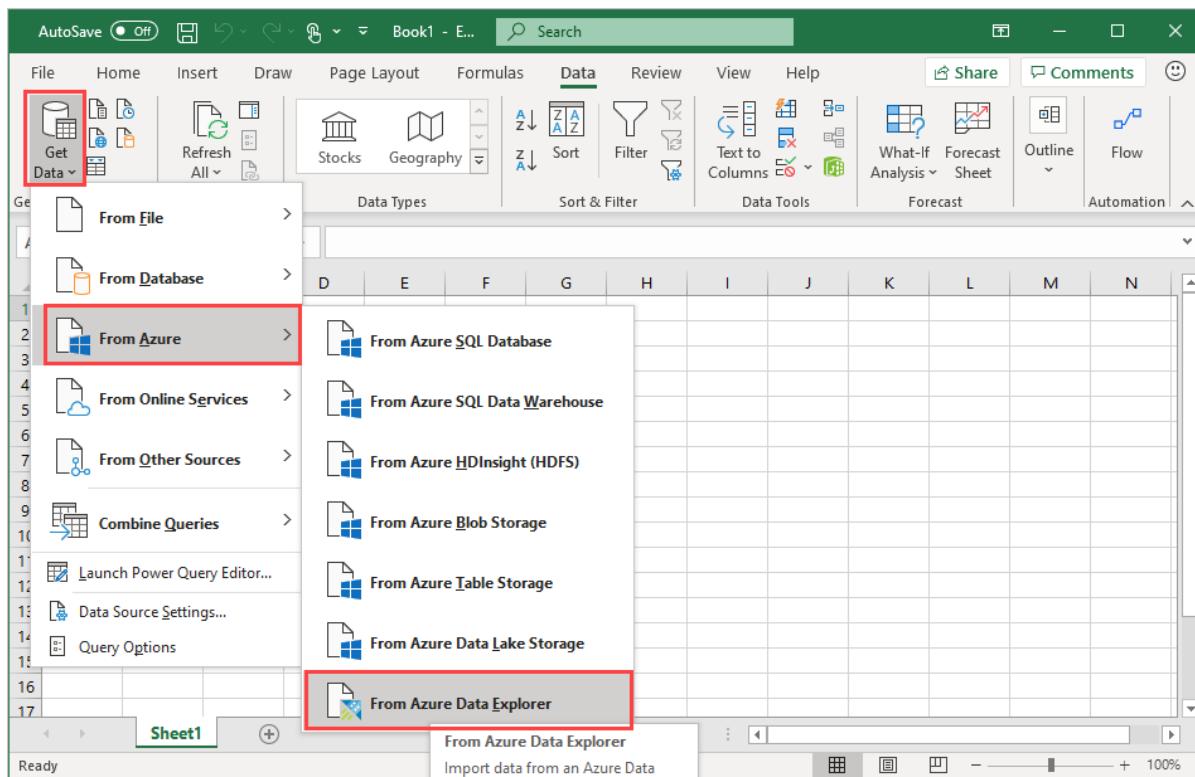
12/5/2019 • 2 minutes to read • [Edit Online](#)

Azure Data Explorer provides two options for connecting to data in Excel: use the native connector or import a query from Azure Data Explorer. This article shows you how to use the native connector in Excel and connect to the Azure Data Explorer cluster to get and visualize data.

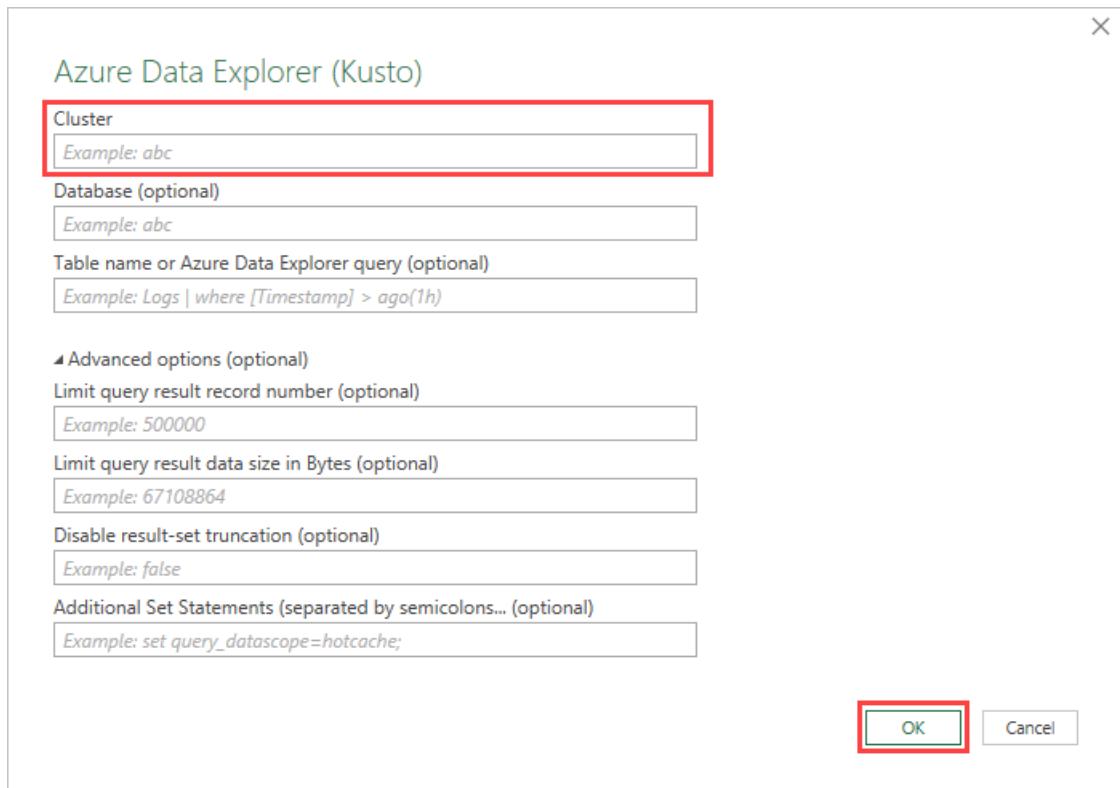
The Azure Data Explorer Excel native connector offers the ability to export query results to Excel. You can also add a KQL query as an Excel data source for additional calculations or visualizations.

Define Kusto query as an Excel data source and load the data to Excel

1. Open **Microsoft Excel**.
2. In the **Data** tab, select **Get Data > From Azure > From Azure Data Explorer**.



3. In the **Azure Data Explorer (Kusto)** window, complete the following fields and select **OK**.



FIELD	DESCRIPTION
Cluster	Name of cluster (mandatory)
Database	Name of database
Table name or Azure Data Explorer query	Name of table or Azure Data Explorer query

Advanced Options:

FIELD	DESCRIPTION
Limit query result record number	Limit the number of records loaded into excel
Limit query result data size (bytes)	Limit the data size
Disable result-set truncation	
Additional Set statements (separated by semicolons)	Add <code>set</code> statements to apply to data source

4. In the **Navigator** pane, navigate to the correct table. In the table preview pane, select **Transform Data** to make changes to your data or select **Load** to load it to Excel.

Navigator

The screenshot shows the Power BI Navigator interface. On the left, there's a navigation pane with a search bar, a 'Select multiple items' checkbox, 'Display Options' dropdown, 'Help [1]', and a 'Samples [6]' section. Under 'Samples [6]', 'ConferenceSessions', 'demo_make_series1', 'demo_many_series1', 'demo_series2', 'demo_series3', and 'StormEvents' are listed, with 'StormEvents' highlighted by a red box. The main area is titled 'StormEvents' and displays a table with the following data:

StartTime	EndTime	EpisodeId	EventId	State
9/29/2007 8:11:00 AM +00:00	9/29/2007 8:11:00 AM +00:00	11091	61032	ATLANTIC SOU
9/18/2007 8:00:00 PM +00:00	9/19/2007 6:00:00 PM +00:00	11074	60904	FLORIDA
9/20/2007 9:57:00 PM +00:00	9/20/2007 10:05:00 PM +00:00	11078	60913	FLORIDA
12/30/2007 4:00:00 PM +00:00	12/30/2007 4:05:00 PM +00:00	11749	64588	GEORGIA
12/20/2007 7:50:00 AM +00:00	12/20/2007 7:53:00 AM +00:00	12554	68796	MISSISSIPPI
12/20/2007 10:32:00 AM +00:00	12/20/2007 10:36:00 AM +00:00	12554	68814	MISSISSIPPI
12/20/2007 8:47:00 AM +00:00	12/20/2007 8:48:00 AM +00:00	12554	68834	MISSISSIPPI
12/28/2007 2:03:00 AM +00:00	12/28/2007 2:11:00 AM +00:00	12561	68846	MISSISSIPPI
12/7/2007 2:00:00 PM +00:00	12/8/2007 4:00:00 AM +00:00	13183	73241	AMERICAN SA
12/13/2007 9:02:00 AM +00:00	12/13/2007 10:30:00 AM +00:00	11780	64725	KENTUCKY
12/23/2007 6:02:00 AM +00:00	12/23/2007 6:07:00 AM +00:00	11781	64726	OHIO
12/23/2007 6:38:00 AM +00:00	12/23/2007 6:43:00 AM +00:00	11781	64727	OHIO
12/23/2007 6:36:00 AM +00:00	12/23/2007 6:41:00 AM +00:00	11781	64728	OHIO
12/23/2007 7:14:00 AM +00:00	12/23/2007 7:19:00 AM +00:00	11781	64729	OHIO
12/11/2007 9:45:00 PM +00:00	12/12/2007 4:45:00 PM +00:00	12826	70787	KANSAS
12/28/2007 2:47:00 AM +00:00	12/28/2007 3:05:00 AM +00:00	12561	68867	MISSISSIPPI
12/28/2007 3:05:00 AM +00:00	12/28/2007 3:16:00 AM +00:00	12561	68868	MISSISSIPPI
12/10/2007 1:10:00 PM +00:00	12/13/2007 2:30:00 AM +00:00	12068	65995	KENTUCKY
12/15/2007 1:00:00 PM +00:00	12/15/2007 3:00:00 PM +00:00	11895	65282	KENTUCKY
12/15/2007 12:00:00 PM +00:00	12/15/2007 6:00:00 PM +00:00	11895	65283	KENTUCKY
12/28/2007 3:30:00 AM +00:00	12/28/2007 3:31:00 AM +00:00	12561	68871	MISSISSIPPI
12/16/2007 2:30:00 AM +00:00	12/16/2007 2:35:00 AM +00:00	11747	64586	FLORIDA

At the bottom right are buttons for 'Load' (with a dropdown arrow), 'Transform Data' (highlighted with a red box), and 'Cancel'.

TIP

If **Database** and/or **Table name or Azure Data Explorer query** are already specified, the correct table preview pane will open automatically.

Analyze and visualize data in Excel

Once the data loads to excel and is available in your Excel sheet, you can analyze, summarize, and visualize the data by creating relationships and visuals.

1. In the **Table Design** tab, select **Summarize with PivotTable**. In the **Create PivotTable** window, select the relevant table, and select **OK**.

The screenshot shows the Microsoft Excel ribbon with the 'Table Design' tab selected. A 'Create PivotTable' dialog box is displayed over the worksheet. The 'Table/Range' dropdown is set to 'StormEvents'. The 'OK' button at the bottom right of the dialog box is highlighted with a red box.

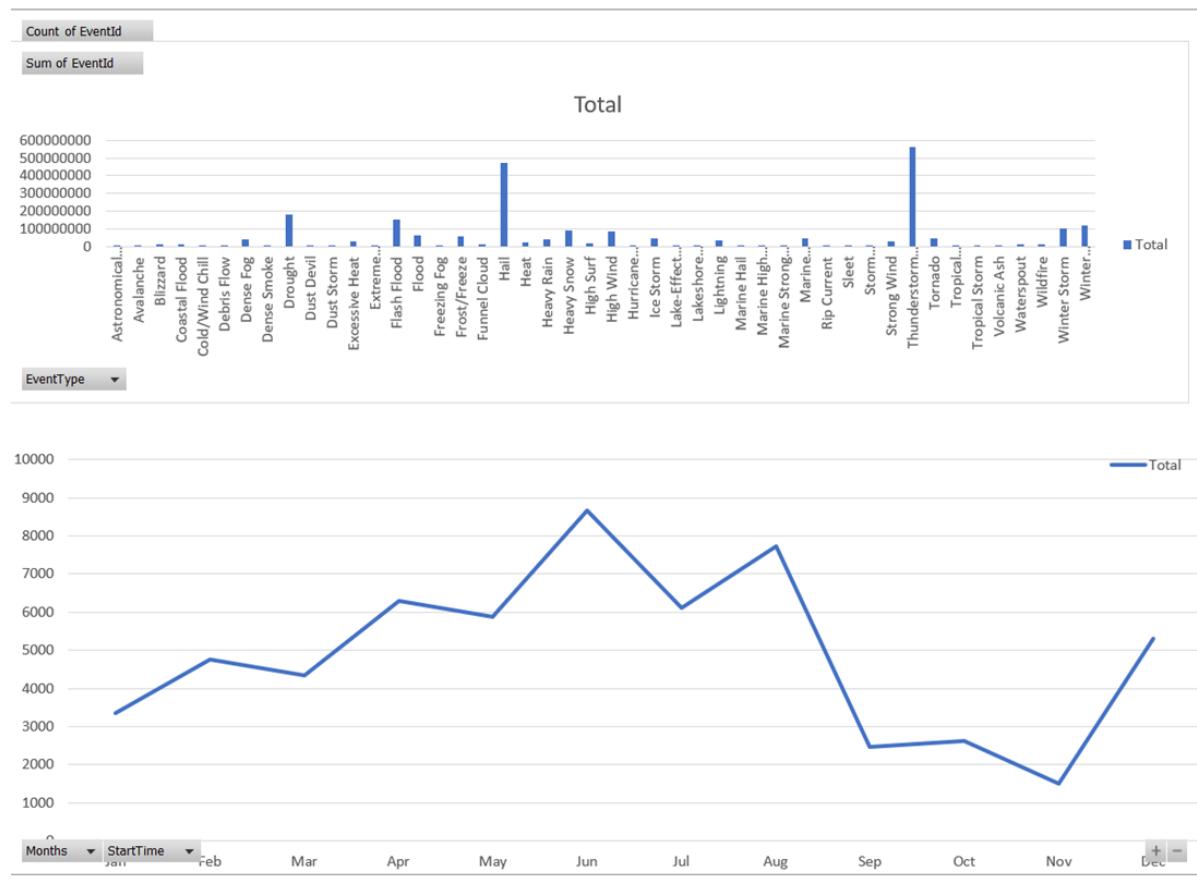
2. In the **PivotTable Fields** pane, select the relevant table columns to create summary tables. In the example below, **EventId** and **State** are selected.

The screenshot shows the Microsoft Excel ribbon with the 'PivotTable Analyze' tab selected. The 'PivotTable Fields' pane is open on the right side, showing fields for 'EventId', 'State', and 'EventType'. The main grid displays a pivot table with 'State' in the rows and 'Sum of EventId' in the values. The 'Sum of EventId' column shows values such as 58382873 for ALABAMA and 16479873 for HAWAII.

3. In the **PivotTable Analyze** tab, select **PivotChart** to create visuals based on the table.

The screenshot shows the Microsoft Excel ribbon with the 'PivotTable Analyze' tab selected. The 'Tools' section of the ribbon has the 'PivotChart' icon highlighted with a red box.

4. In the example below, use **Event Id**, **StartTime**, and **EventType** to view additional information about the weather events.



5. Create full dashboards to monitor your data.

Next steps

[Visualize data using an Azure Data Explorer Kusto query imported into Microsoft Excel](#)

Visualize data using an Azure Data Explorer Kusto query imported into Microsoft Excel

12/5/2019 • 2 minutes to read • [Edit Online](#)

Azure Data Explorer provides two options for connecting to data in Excel: use the native connector or import a query from Azure Data Explorer. This article shows you how to import a query from Azure Data Explorer to Excel to visualize data. Add Kusto query as an Excel data source to do additional calculations or visualizations on the data.

Prerequisites

- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.
- An organizational email account that is a member of Azure Active directory, so you can connect to the [Azure Data Explorer help cluster](#)
or
- Create a [test cluster and database](#) and sign in to [the Azure Data Explorer Web UI application](#).

Define Kusto query as an Excel data source

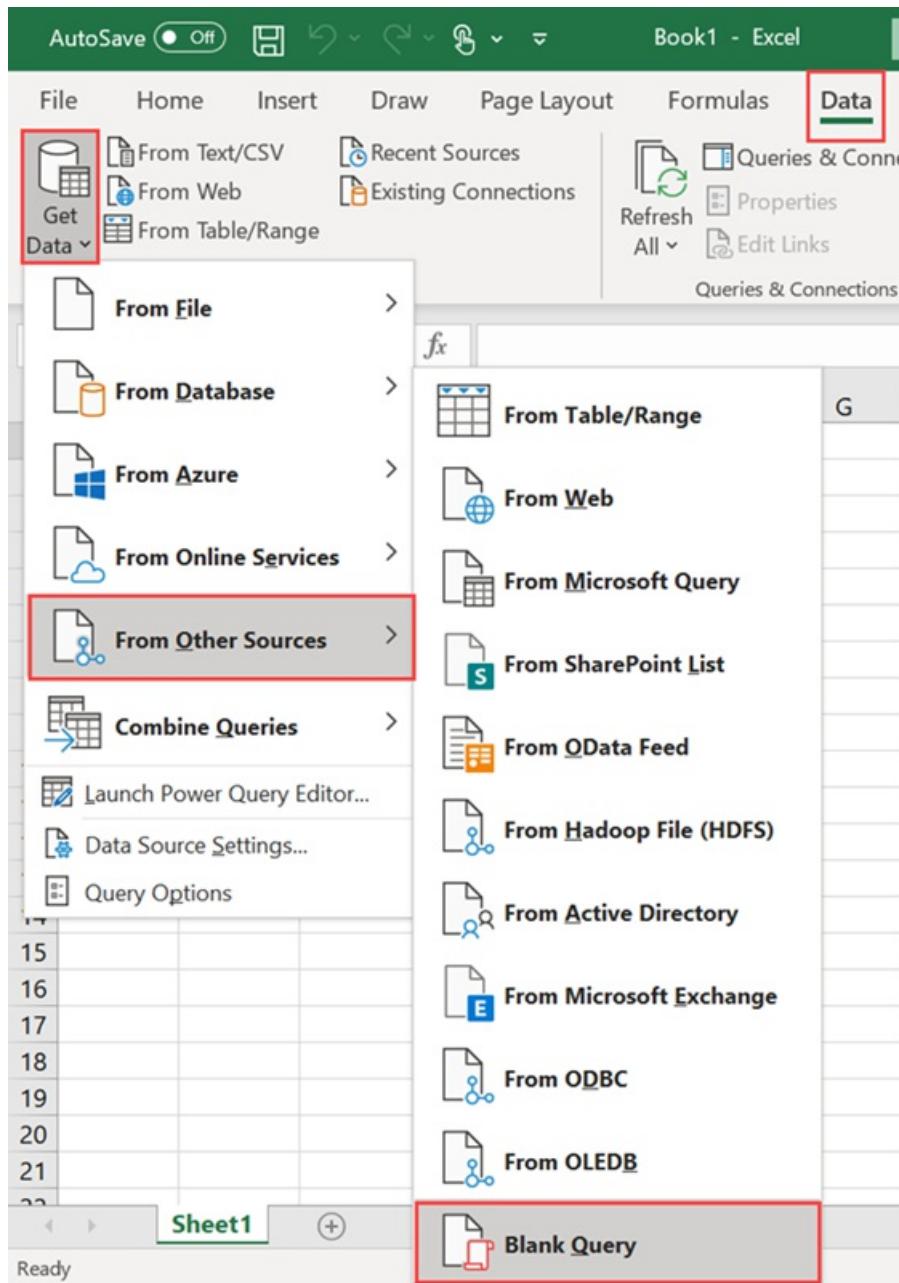
1. In [Azure Data Explorer Web UI](#), run the query and check the results.
2. Select the **Share** tab and select **Query to Power BI**.



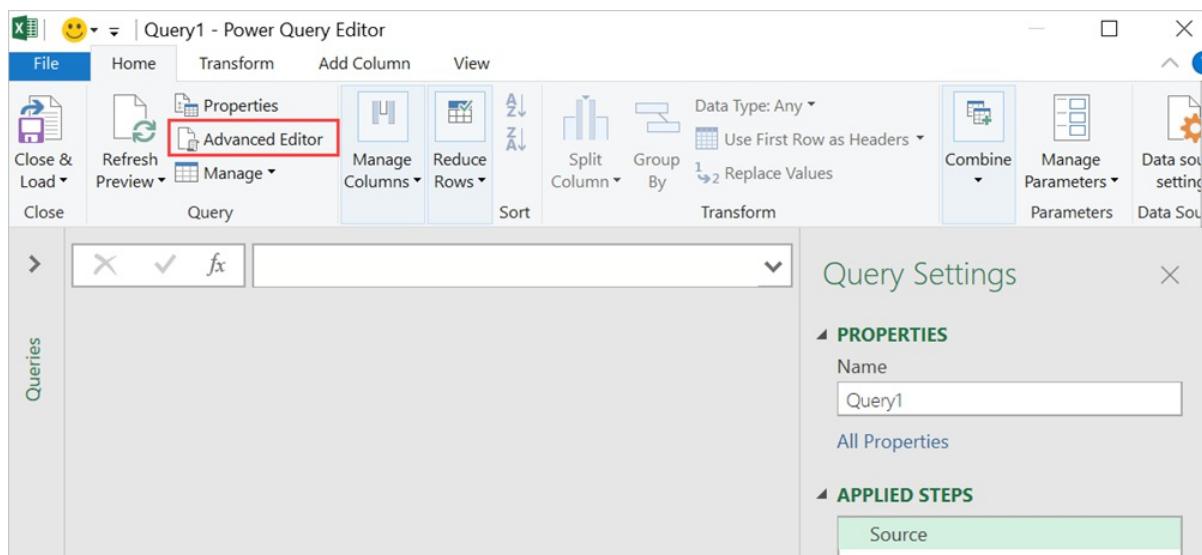
3. A window appears with the following notification:



4. Open **Microsoft Excel**.
5. In the **Data** tab, select **Get Data > From Other Sources > Blank Query**.



6. The **Power Query Editor** window opens. In the window, select **Advanced Editor**.



7. In the **Advanced Editor** window, paste the query you exported to the clipboard and select **Done**.

Query1

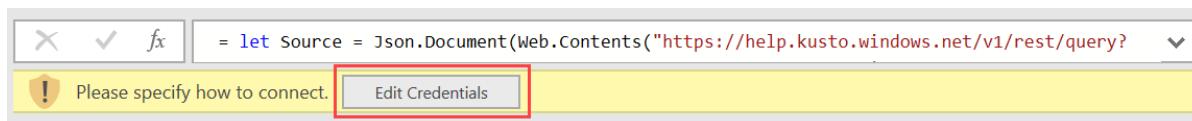
Display Options ?

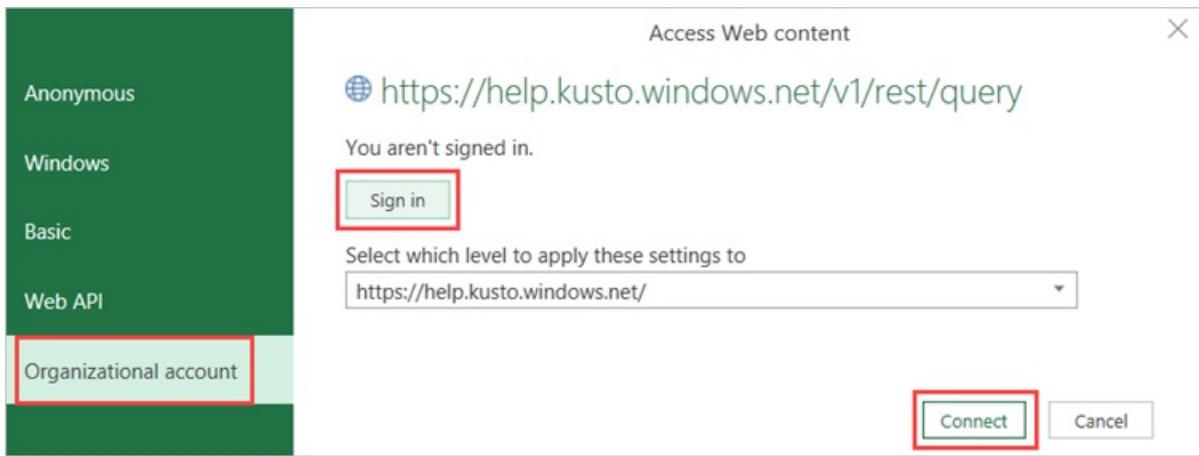
```

let KustoQuery =
    let Source = Json.Document(Web.Contents("https://help.kusto.windows.net/v1/rest/query?db=Sam")
    TypeMap = #table(
        { "DataType", "Type" },
        {
            { "Double", Double.Type },
            { "Int64", Int64.Type },
            { "Int32", Int32.Type },
            { "Int16", Int16.Type },
            { "UInt64", Number.Type },
            { "UInt32", Number.Type },
            { "UInt16", Number.Type },
            { "Byte", Byte.Type },
            { "Single", Single.Type },
            { "Decimal", Decimal.Type },
            { "TimeSpan", Duration.Type },
            { "DateTime", DateTimeZone.Type },
            { "String", Text.Type },
            { "Boolean", Logical.Type },
            { "SByte", Logical.Type },
            { "Guid", Text.Type }
        }),
    Exception = Source[Exceptions]?{0}?,
    Result = if (Exception <> null) then
        error Exception
    else
        let
            DataTable = Source[Tables]{0},
            Columns = Table.FromRecords(DataTable[Columns]),
            ColumnsWithType = Table.Join(Columns, {"DataType"}, TypeMap , {"DataType"}),
            TableRows = Table.FromRows(DataTable[Rows], Columns[ColumnName]),
            TypedTable = Table.TransformColumnTypes(TableRows, Table.ToList(ColumnsWithType), (c)
                in
                    TypedTable
            in
                Result
            in
                KustoQuery

```

✓ No syntax errors have been detected.

Done**Cancel**8. To authenticate, select **Edit Credentials**.9. Select **Organizational account** and **Sign in**. Complete the sign-in process and then select **Connect**.



Repeat the previous steps to add more queries. You can rename the queries to more meaningful names.

10. Select the **Close & Load** button to get your data into Excel.

11. Now your data is in Excel. Select the **Refresh** button to refresh the query.

Next steps

[Visualize data using the Azure Data Explorer connector for Excel](#)

Visualize data from Azure Data Explorer in Grafana

11/13/2019 • 5 minutes to read • [Edit Online](#)

Grafana is an analytics platform that enables you to query and visualize data, then create and share dashboards based on your visualizations. Grafana provides an Azure Data Explorer *plugin*, which enables you to connect to and visualize data from Azure Data Explorer. In this article, you learn to set up Azure Data Explorer as a data source for Grafana, and then visualize data from a sample cluster.

Use the following video, to learn how to use Grafana's Azure Data Explorer plugin, set up Azure Data Explorer as a data source for Grafana, and then visualize data.

Alternatively you can [configure the data source](#) and [visualize data](#) as detailed in the article below.

Prerequisites

You need the following to complete this article:

- [Grafana version 5.3.0 or later](#) for your operating system
- The [Azure Data Explorer plugin](#) for Grafana
- A cluster that includes the StormEvents sample data. For more information, see [Quickstart: Create an Azure Data Explorer cluster and database](#) and [Ingest sample data into Azure Data Explorer](#).

The StormEvents sample data set contains weather-related data from the [National Centers for Environmental Information](#).

Configure the data source

You perform the following steps to configure Azure Data Explorer as a data source for your dashboard tool. We'll cover these steps in more detail in this section:

1. Create an Azure Active Directory (Azure AD) service principal. The service principal is used by your dashboard tool to access the Azure Data Explorer service.
2. Add the Azure AD service principal to the *viewers* role in the Azure Data Explorer database.
3. Specify your dashboard tool connection properties based on information from the Azure AD service principal, then test the connection.

Create a service principal

You can create the service principal in the [Azure portal](#) or using the [Azure CLI](#) command-line experience.

Regardless of which method you use, after creation you get values for four connection properties that you'll use in later steps.

Azure portal

1. To create the service principal, follow the instructions in the [Azure portal documentation](#).

- a. In the [Assign the application to a role](#) section, assign a role type of **Reader** to your Azure Data Explorer cluster.
- b. In the [Get values for signing in](#) section, copy the three property values covered in the steps: **Directory ID** (tenant ID), **Application ID**, and **Password**.

2. In the Azure portal, select **Subscriptions** then copy the ID for the subscription in which you created the service principal.

SUBSCRIPTION	SUBSCRIPTION ID	MY ROLE	CURRENT C...	STATUS
Azure Team Subscr...	<Subscription Id>	Owner	Not available	Active

Azure CLI

1. Create a service principal. Set an appropriate scope and a role type of `reader`.

```
az ad sp create-for-rbac --name "https://{{UrlToYourDashboard}}:{PortNumber}" --role "reader" \
--scopes /subscriptions/{{SubID}}/resourceGroups/{{ResourceGroupName}}
```

For more information, see [Create an Azure service principal with Azure CLI](#).

2. The command returns a result set like the following. Copy the three property values: **appId**, **password**, and **tenant**.

```
{
  "appId": "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
  "displayName": "{UrlToYourDashboard}:{PortNumber}",
  "name": "https://{{UrlToYourDashboard}}:{PortNumber}",
  "password": "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
  "tenant": "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
}
```

3. Get a list of your subscriptions.

```
az account list --output table
```

Copy the appropriate subscription ID.

Name	CloudName	SubscriptionId	State	IsDefault
Visual Studio Enterprise	AzureCloud	<Subscription Id>	Enabled	True
Azure Team Subscription	AzureCloud	<Subscription Id>	Enabled	False

Add the service principal to the viewers role

Now that you have a service principal, you add it to the *viewers* role in the Azure Data Explorer database. You can perform this task under **Permissions** in the Azure portal, or under **Query** by using a management command.

Azure portal - Permissions

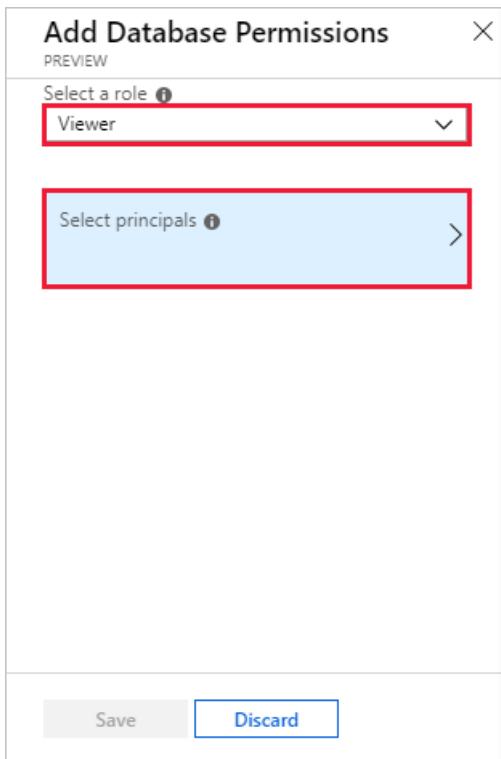
1. In the Azure portal, go to your Azure Data Explorer cluster.
2. In the **Overview** section, select the database with the StormEvents sample data.

The screenshot shows the Azure Data Explorer cluster overview page for 'docscluster'. The left sidebar has tabs for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Query, Settings, Scale out, Properties, Locks, and Automation script. The 'Overview' tab is selected. On the right, there's a summary of the resource group ('Resource group (change) docscluster'), location ('West US'), and subscription ('Subscription (change) <SubscriptionName>'). Below this, a list of databases is shown, starting with 'DATABASE' and then 'TestDatabase', which is highlighted with a red box.

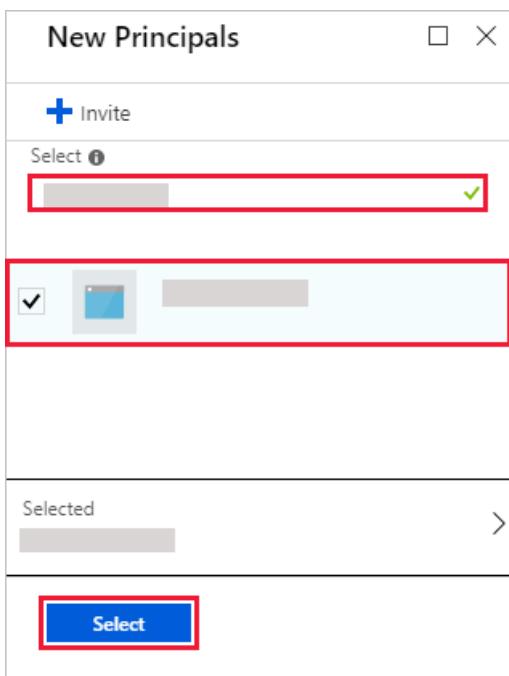
3. Select **Permissions** then **Add**.

The screenshot shows the 'TestDatabase - Permissions' page. The 'Permissions' tab is selected. At the top, there's a 'Role' dropdown set to '6 selected' and a '+ Add' button, which is highlighted with a red box. The main area shows '2 items (1 Admin, 1 User)' listed under 'NAME'.

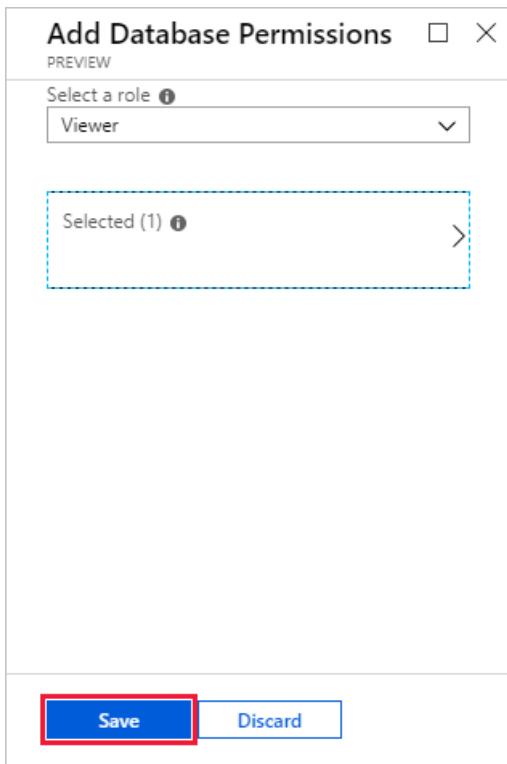
4. Under **Add database permissions**, select the **Viewer** role then **Select principals**.



5. Search for the service principal you created. Select the principal, then **Select**.

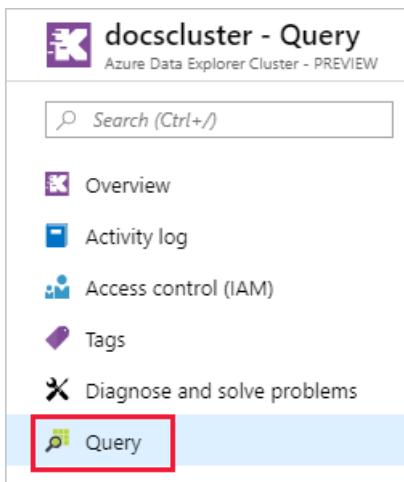


6. Select **Save**.



Management command - Query

1. In the Azure portal, go to your Azure Data Explorer cluster, and select **Query**.



2. Run the following command in the query window. Use the application ID and tenant ID from the Azure portal or CLI.

```
.add database {TestDatabase} viewers ('aadapp={ApplicationID};{TenantID}')
```

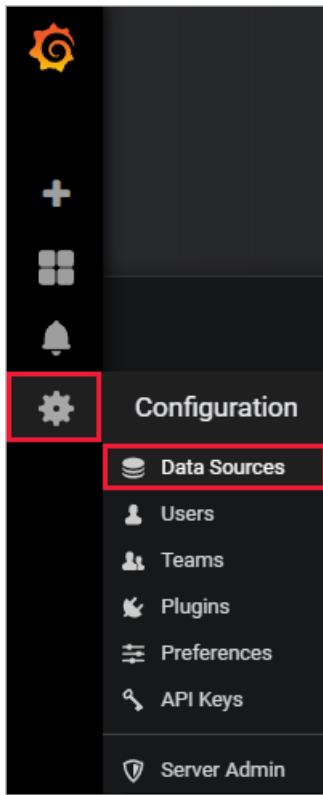
The command returns a result set like the following. In this example, the first row is for an existing user in the database, and the second row is for the service principal that was just added.

Role	PrincipalType	PrincipalDisplayName	PrincipalObjectId	PrincipalFQN
Database TestDatabase Admin	AAD User	<Principal display name>	<Principal object id>	<Principal FQN>
Database TestDatabase Viewer	AAD Application	<Principal display name>	<Principal object id>	<Principal FQN>

Specify properties and test the connection

With the service principal assigned to the *viewers* role, you now specify properties in your instance of Grafana, and test the connection to Azure Data Explorer.

1. In Grafana, on the left menu, select the gear icon then **Data Sources**.



2. Select **Add data source**.
3. On the **Data Sources / New** page, enter a name for the data source, then select the type **Azure Data Explorer Datasource**.

A screenshot of the 'Data Sources / New' configuration page. The 'Settings' tab is selected. The 'Name' field contains 'Test Azure Data Explorer'. The 'Type' dropdown is set to 'Graphite' and has 'Azure Data Explorer Datasource' selected. Other options in the dropdown include Cloudwatch, Elasticsearch, Grafana Logging, InfluxDB, Microsoft SQL Server, MySQL, OpenTSDB, PostgreSQL, Prometheus, Stackdriver, and TestData DB. The page also includes sections for 'HTTP', 'URL', 'Access', 'Auth', and 'Basic Auth'.

4. Enter the name of your cluster in the form `https://{{ClusterName}}.{{Region}}.kusto.windows.net`. Enter the other values from the Azure portal or CLI. See the table below the following image for a mapping.

Cluster URL	https://yourcluster.kusto.windows.net	
Subscription Id	XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX	
Tenant Id	XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX	
Client Id	XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX	
Client Secret		
Default Database		

GRAFANA UI	AZURE PORTAL	AZURE CLI
Subscription Id	SUBSCRIPTION ID	SubscriptionId
Tenant Id	Directory ID	tenant
Client Id	Application ID	appId
Client secret	Password	password

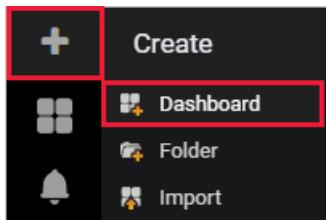
5. Select **Save & Test**.

If the test is successful, go to the next section. If you come across any issues, check the values you specified in Grafana, and review previous steps.

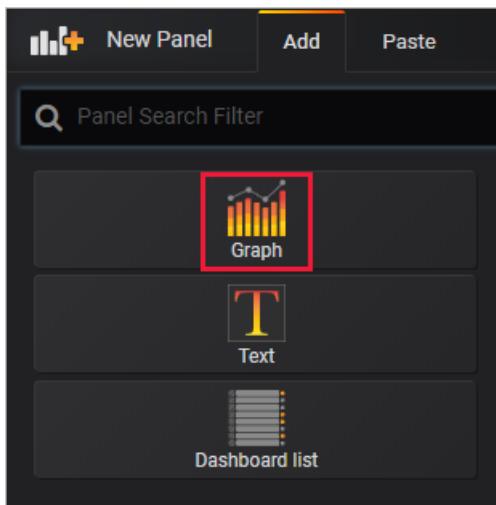
Visualize data

Now you've finished configuring Azure Data Explorer as a data source for Grafana, it's time to visualize data. We'll show a basic example here, but there's a lot more you can do. We recommend looking at [Write queries for Azure Data Explorer](#) for examples of other queries to run against the sample data set.

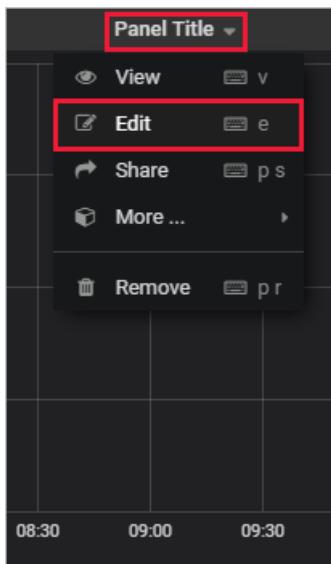
1. In Grafana, on the left menu, select the plus icon then **Dashboard**.



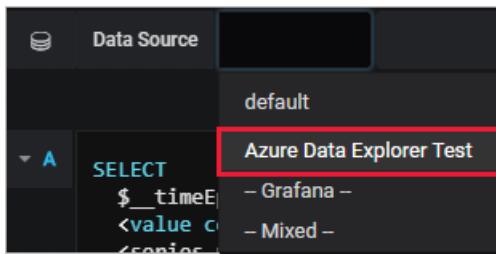
2. Under the **Add** tab, select **Graph**.



3. On the graph panel, select **Panel Title** then **Edit**.



4. At the bottom of the panel, select **Data Source** then select the data source that you configured.



5. In the query pane, copy in the following query then select **Run**. The query buckets the count of events by day for the sample data set.

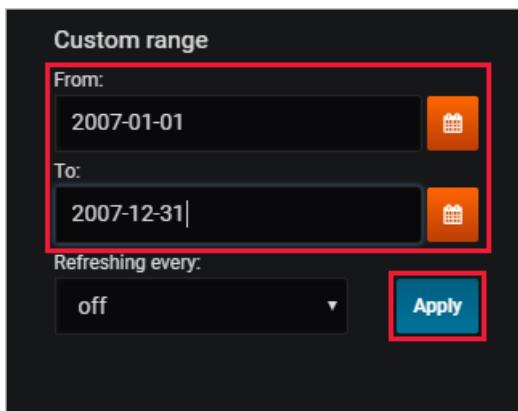
```
StormEvents
| summarize event_count=count() by bin(StartTime, 1d)
```

Run (Run Query: Shift+Enter, Trigger Suggestion: Ctrl+Space)

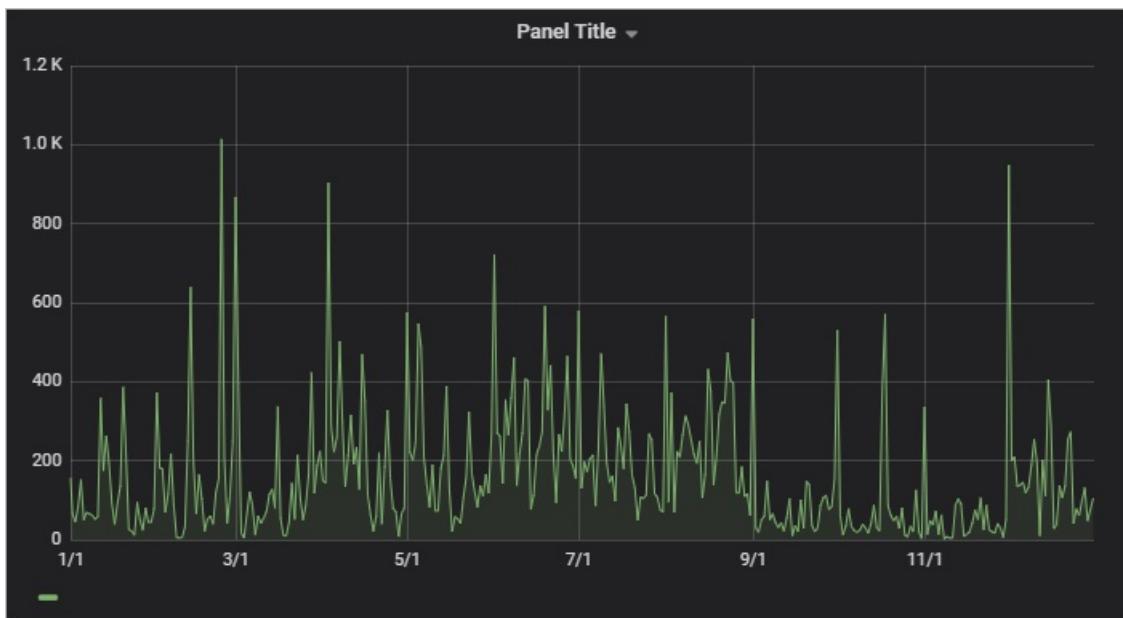
6. The graph doesn't show any results because it's scoped by default to data from the last six hours. On the top menu, select **Last 6 hours**.



7. Specify a custom range that covers 2007, the year included in our StormEvents sample data set. Select **Apply**.



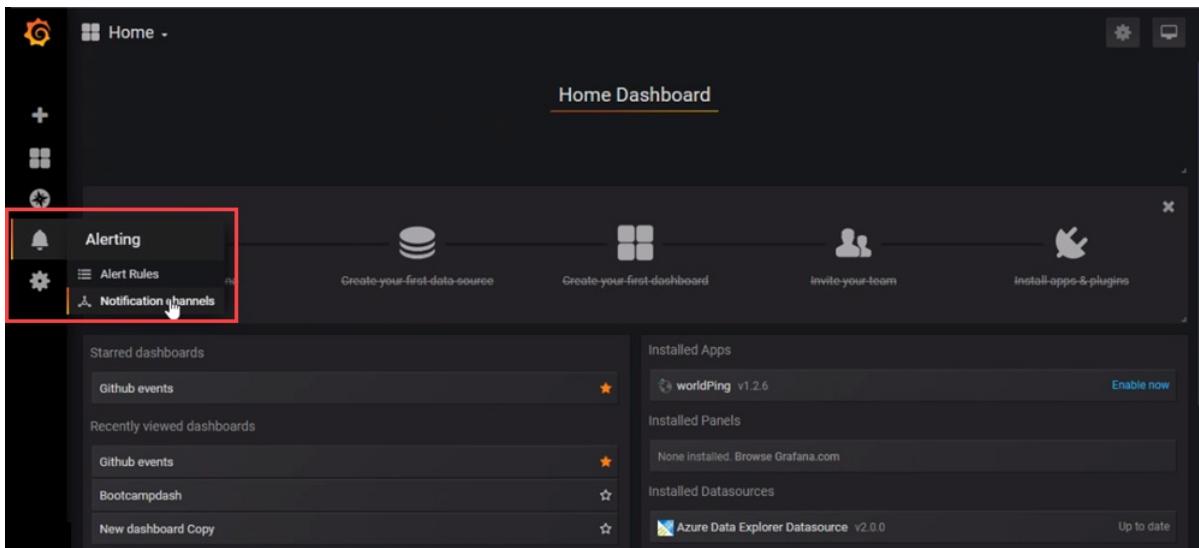
Now the graph shows the data from 2007, bucketed by day.



8. On the top menu, select the save icon:

Create Alerts

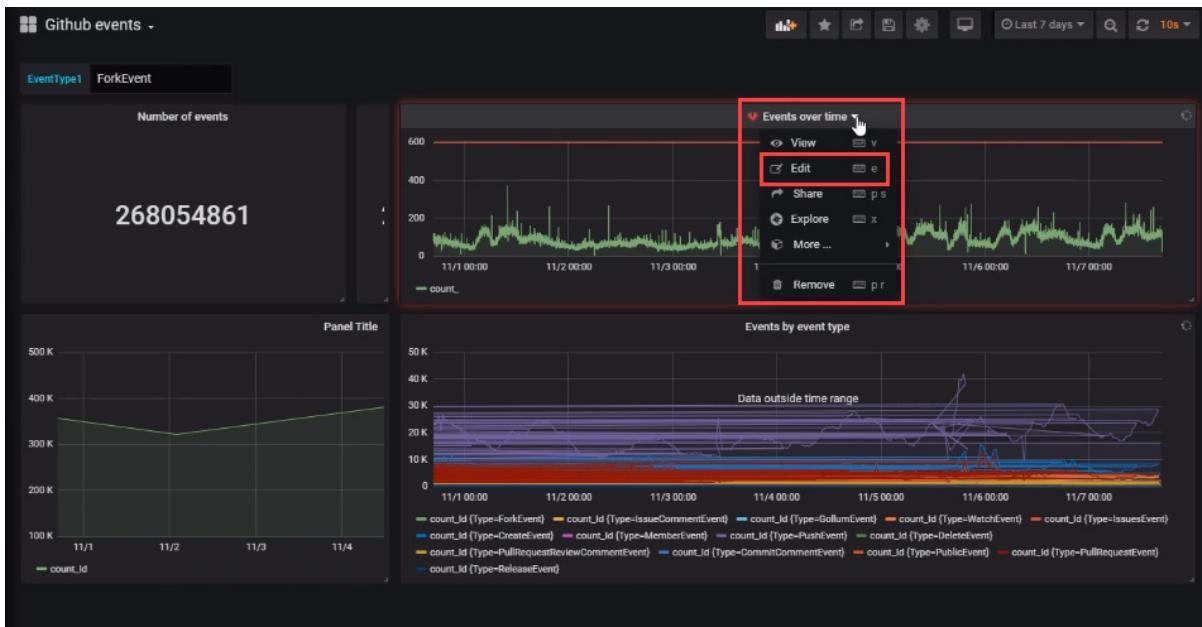
1. In Home Dashboard, select **Alerting > Notification channels** to create a new notification channel



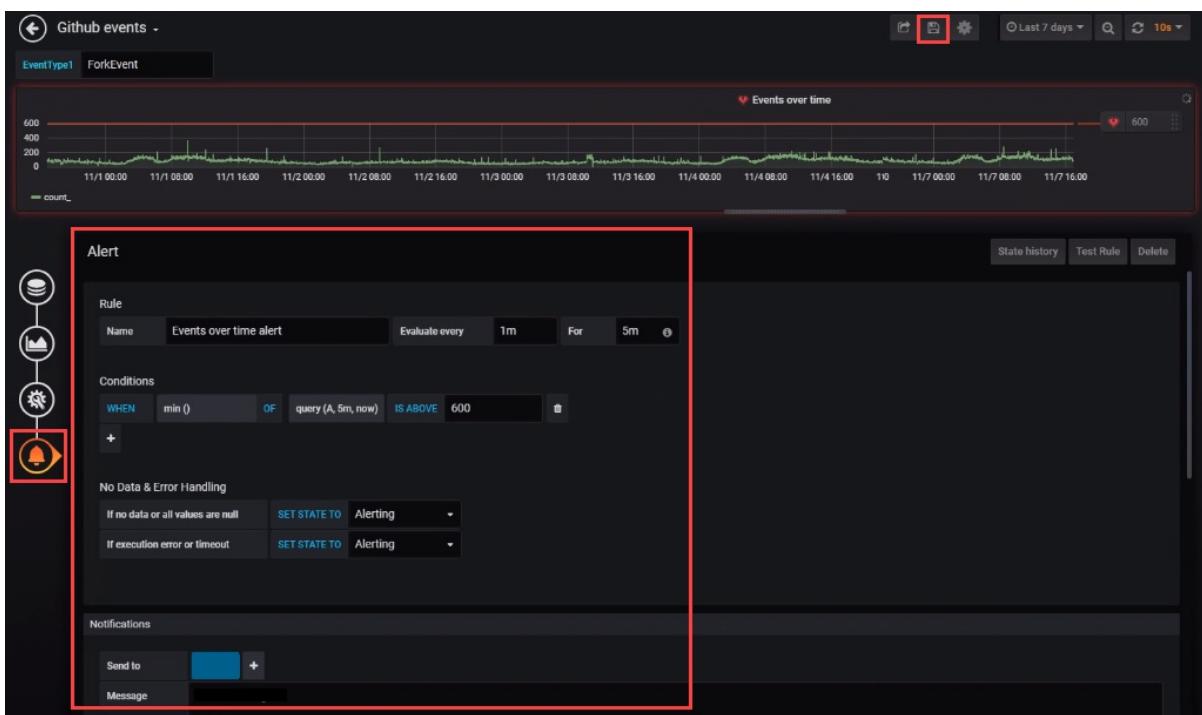
2. Create a new **Notification channel**, then **Save**.

This screenshot shows the 'New Notification Channel' configuration page. The 'Name' field is set to 'Azure Data Explorer channel' and the 'Type' field is set to 'Email'. The 'Save' button at the bottom-left is highlighted with a red box. Other settings include 'Default (send on all alerts)' (disabled), 'Include image' (enabled), 'Disable Resolve Message' (disabled), and 'Send reminders' (disabled). The 'Email addresses' section contains the email 'xxxx@microsoft.com'. A note says 'You can enter multiple email addresses using a ; separator'.

3. On the **Dashboard**, select **Edit** from the dropdown.



4. Select the alert bell icon to open the **Alert** pane. Select **Create Alert**. Complete the following properties in the **Alert** pane.



5. Select the **Save dashboard** icon to save your changes.

Next steps

- [Write queries for Azure Data Explorer](#)

Connect to Azure Data Explorer with ODBC

11/13/2019 • 2 minutes to read • [Edit Online](#)

Open Database Connectivity ([ODBC](#)) is a widely accepted application programming interface (API) for database access. Use ODBC to connect to Azure Data Explorer from applications that don't have a dedicated connector.

Behind the scenes, applications call functions in the ODBC interface, which are implemented in database-specific modules called *drivers*. Azure Data Explorer supports a subset of the SQL Server communication protocol ([MS-TDS](#)), so it can use the ODBC driver for SQL Server.

Using the following video, you can learn to create an ODBC connection.

Alternatively, you can [configure the ODBC data source](#) as detailed below.

In the article, you learn how to use the SQL Server ODBC driver, so you can connect to Azure Data Explorer from any application that supports ODBC.

Prerequisites

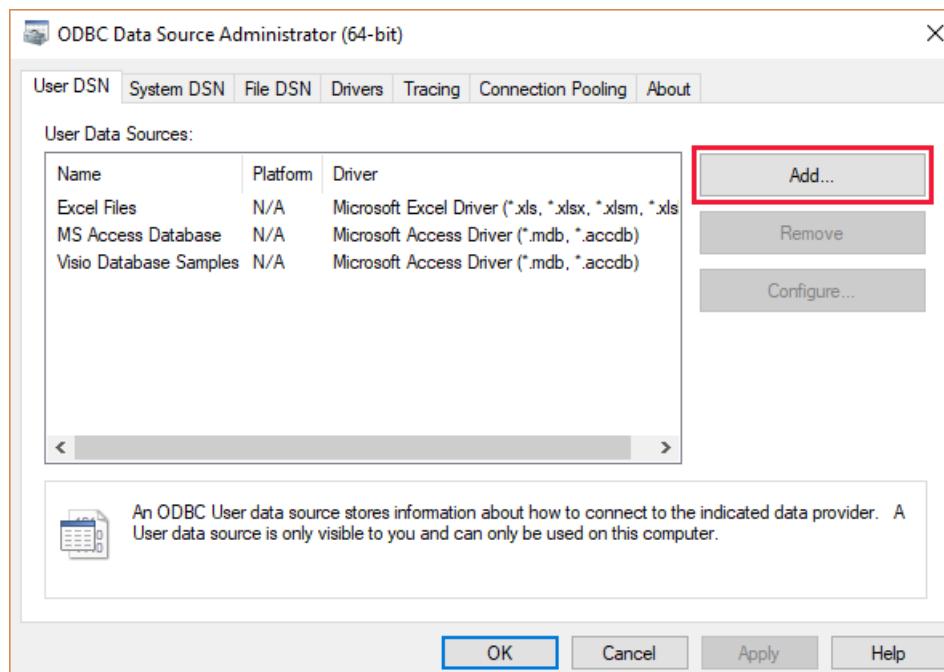
You need the following:

- [Microsoft ODBC Driver for SQL Server version 17.2.0.1 or later](#) for your operating system.

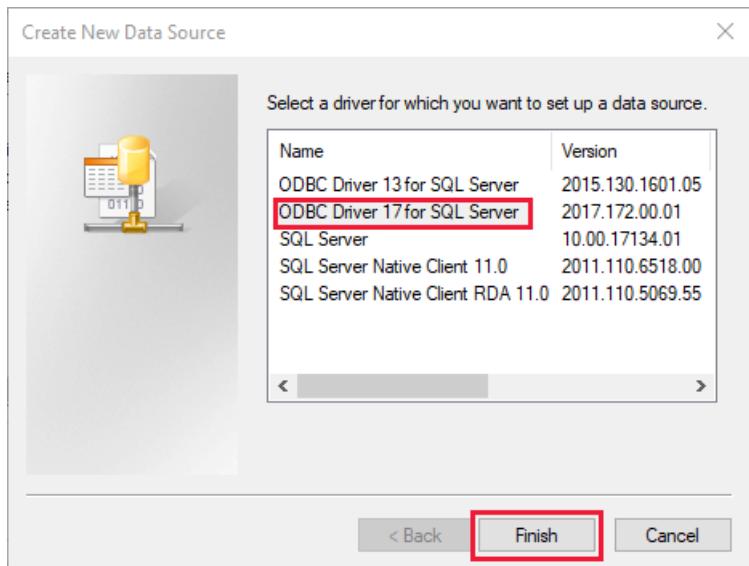
Configure the ODBC data source

Follow these steps to configure an ODBC data source using the ODBC driver for SQL Server.

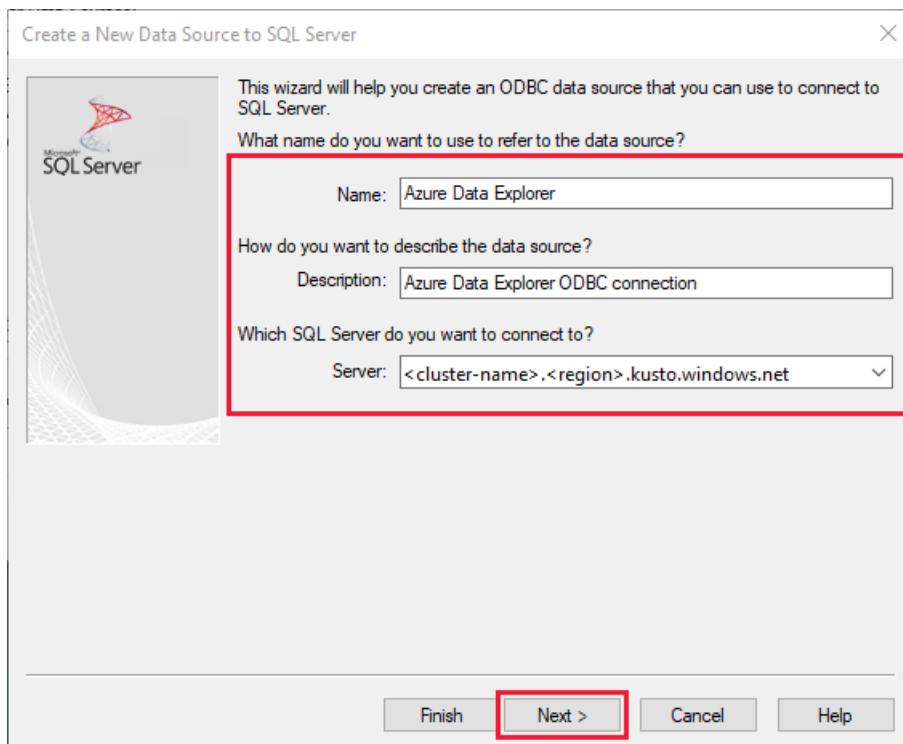
1. In Windows, search for *ODBC Data Sources*, and open the ODBC Data Sources desktop app.
2. Select **Add**.



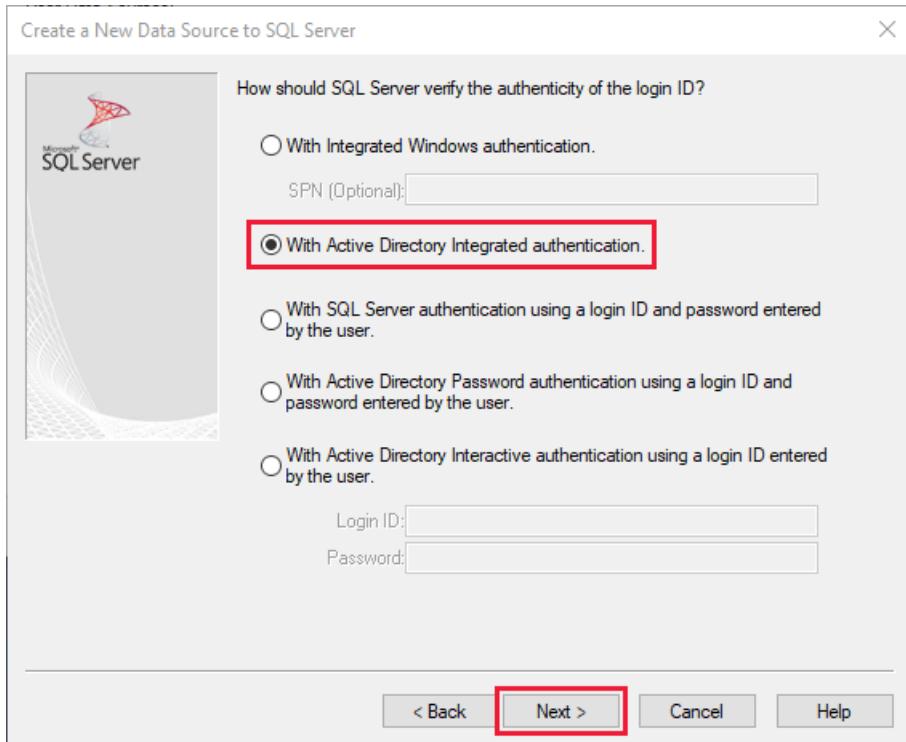
3. Select **ODBC Driver 17 for SQL Server** then **Finish**.



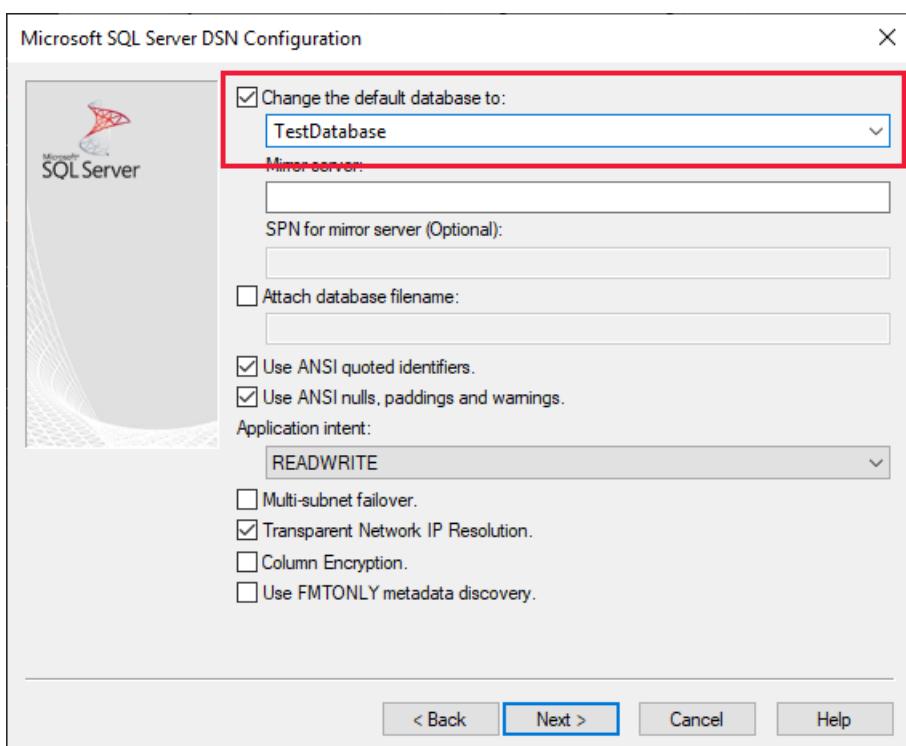
4. Enter a name and description for the connection and the cluster you want to connect to, then select **Next**.
The cluster URL should be in the form <*ClusterName*>.<*Region*>.kusto.windows.net.



5. Select **Active Directory Integrated** then **Next**.

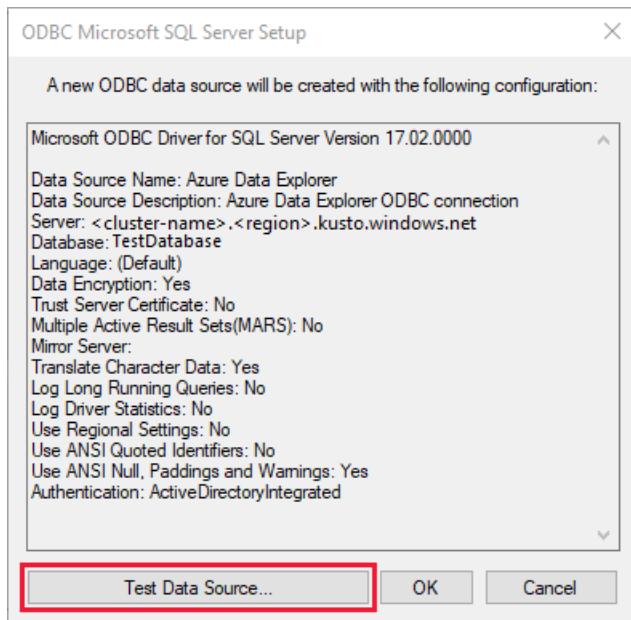


6. Select the database with the sample data then **Next**.

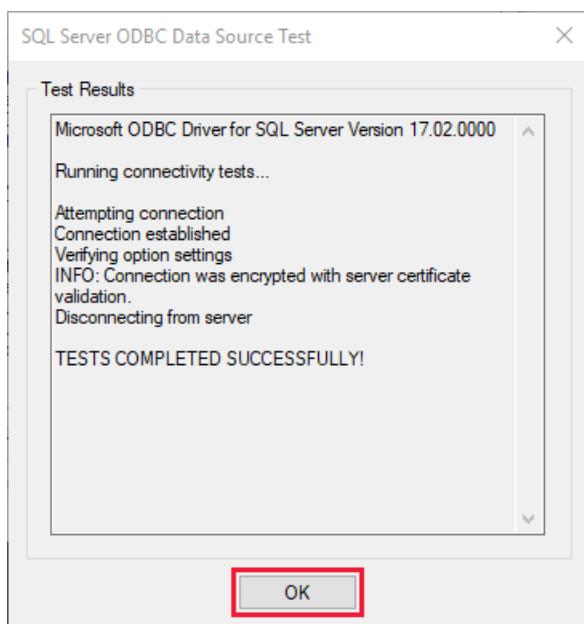


7. On the next screen, leave all options as defaults then select **Finish**.

8. Select **Test Data Source**.



9. Verify that the test succeeded then select **OK**. If the test didn't succeed, check the values that you specified in previous steps, and ensure you have sufficient permissions to connect to the cluster.



Next steps

- Connect to Azure Data Explorer from Tableau

Visualize data from Azure Data Explorer in Tableau

11/13/2019 • 2 minutes to read • [Edit Online](#)

Tableau is a visual analytics platform for business intelligence. To connect to Azure Data Explorer from Tableau and bring in data from a sample cluster, use the SQL Server Open Database Connectivity (ODBC) driver.

Prerequisites

You need the following to complete this article:

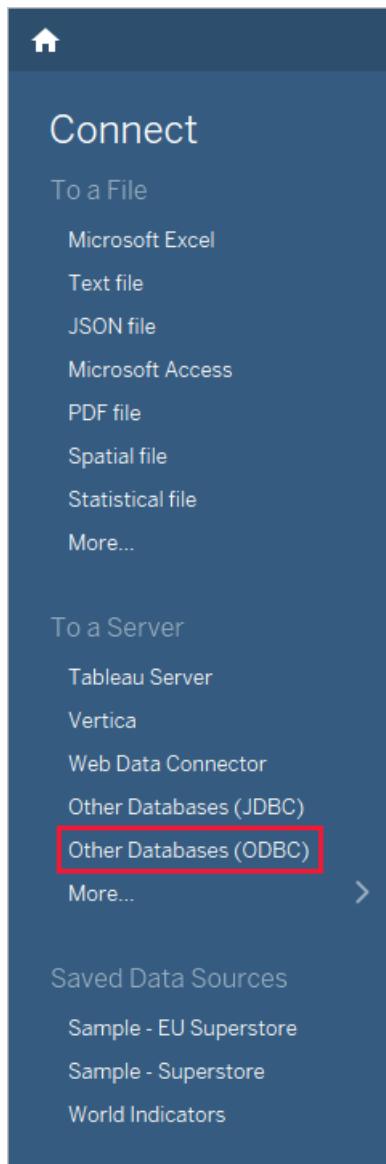
- [Connect to Azure Data Explorer with ODBC](#) using the SQL Server ODBC driver, to connect to Azure Data Explorer from Tableau.
- Tableau Desktop, full, or [trial](#) version.
- A cluster that includes the StormEvents sample data. For more information, see [Create an Azure Data Explorer cluster and database](#) and [Ingest sample data into Azure Data Explorer](#).

The StormEvents sample data set contains weather-related data from the [National Centers for Environmental Information](#).

Visualize data in Tableau

Once you've finished configuring ODBC, you can bring sample data into Tableau.

1. In Tableau Desktop, in the left menu, select **Other Databases (ODBC)**.



2. For **DSN**, select the data source you created for ODBC, then select **Sign In**.

Other Databases (ODBC)

Connect Using

Generic ODBC requires additional configuration for publishing to succeed. Select DSN (data source name) for cross-platform portability. A DSN with the same name must be configured on Tableau Server.

DSN: **Azure Data Explorer**

Driver: ODBC Driver 17 for SQL Server

Connect

Connection Attributes

Server: _____ Port: _____

Database: _____

Username: _____

Password: _____

Description=Azure Data Explorer ODBC connection;APP=Tableau 2018.3;WSID=MININT-O2AT5;QuotedId=No;Authentication=ActiveDirectoryIntegrated

String Extras:

Sign In

- For **Database**, select the database on your sample cluster, such as *TestDatabase*. For **Schema**, select *dbo*, and for **Table**, select the *StormEvents* sample table.

Tableau - Book1

File Data Server Window Help

Connections Add

Azure Data Explorer (ODBC)
Other Databases (ODBC)

Database TestDatabase

Schema dbo

Table StormEvents

Exact Contains Starts with

StormEvents ...StormEvents)

4. Tableau now shows the schema for the sample data. Select **Update Now** to bring the data into Tableau.

The screenshot shows the Tableau Data Source interface. On the left, there's a sidebar with 'Connections' (Azure Data Explorer (ODBC)), 'Database' (dropdown), 'Schema' (Select Schema dropdown), and 'Table' (StormEvents selected). Below these are 'Sort fields', 'Data source order', and checkboxes for 'Show aliases' and 'Show hidden fields'. At the bottom of the interface is a toolbar with 'Go to Worksheet' and tabs for 'Data Source' (selected) and 'Sheet1'.

In the main area, the title is 'StormEvents (dbo.StormEvents) (mblythe)'. It shows the schema for the 'StormEvents' table with columns: Start Time, End Time, Episode Id, Event Id, State, Event Type, Injuries Direct, Injuries Indirect, Deaths Direct, and Deaths Indirect. The 'Event Type' column has an 'Abc' icon above it. At the bottom right of the main area, there are two buttons: 'Update Now' (highlighted with a red box) and 'Automatically Update'.

When the data is imported, Tableau shows rows of data similar to the following image.

StormEvents Start Time	StormEvents End Time	# StormEvents Episode Id	# StormEvents Event Id	StormEvents State	Abc StormEvents Event Type	# StormEvents Injuries Direct	# StormEvents Injuries Indirect
1/1/2007 12:00:00 AM	1/1/2007 12:00:00 AM	2592	13208	NORTH CAROLINA	Thunderstorm Wind	0	0
1/1/2007 12:00:00 AM	1/1/2007 5:00:00 AM	4171	23358	WISCONSIN	Winter Storm	0	0
1/1/2007 12:00:00 AM	1/1/2007 5:00:00 AM	4171	23357	WISCONSIN	Winter Storm	0	0
1/1/2007 12:00:00 AM	1/1/2007 6:00:00 AM	1930	9494	NEW YORK	Winter Weather	0	0
1/1/2007 12:00:00 AM	1/1/2007 6:00:00 AM	1930	9488	NEW YORK	Winter Weather	0	0
1/1/2007 12:00:00 AM	1/1/2007 6:00:00 AM	1930	9487	NEW YORK	Winter Weather	0	0
1/1/2007 12:00:00 AM	1/1/2007 6:00:00 AM	1930	9485	NEW YORK	Winter Weather	0	0
1/1/2007 12:00:00 AM	1/1/2007 6:00:00 AM	1930	9486	NEW YORK	Winter Weather	0	0
1/1/2007 12:00:00 AM	1/1/2007 6:00:00 AM	1930	9493	NEW YORK	Winter Weather	0	0
1/1/2007 12:00:00 AM	1/1/2007 6:00:00 AM	1930	9489	NEW YORK	Winter Weather	0	0
1/1/2007 12:00:00 AM	1/1/2007 6:00:00 AM	1930	9490	NEW YORK	Winter Weather	0	0
1/1/2007 12:00:00 AM	1/1/2007 6:00:00 AM	1930	9491	NEW YORK	Winter Weather	0	0
1/1/2007 12:00:00 AM	1/1/2007 10:13:00 AM	765	3419	ALASKA	Blizzard	0	0
1/1/2007 12:00:00 AM	1/1/2007 12:00:00 PM	1979	12631	DELAWARE	Heavy Rain	0	0
1/1/2007 12:00:00 AM	1/18/2007 11:59:00 PM	2437	12689	OKLAHOMA	Drought	0	0

5. Now you can create visualizations in Tableau based on the data you brought in from Azure Data Explorer.

For more information, see [Tableau Learning](#).

Next steps

- [Write queries for Azure Data Explorer](#)

Visualize data from Azure Data Explorer in Sisense

5/29/2019 • 2 minutes to read • [Edit Online](#)

Sisense is an analytics business intelligence platform that enables you to build analytics apps that deliver highly interactive user experiences. The business intelligence and dashboard reporting software allows you to access and combine data in a few clicks. You can connect to structured and unstructured data sources, join tables from multiple sources with minimal scripting and coding, and create interactive web dashboards and reports. In this article, you'll learn how to set up Azure Data Explorer as a data source for Sisense, and visualize data from a sample cluster.

Prerequisites

You need the following to complete this article:

- [Download and install Sisense app](#)
- Create a cluster and database that includes the StormEvents sample data. For more information, see [Quickstart: Create an Azure Data Explorer cluster and database](#) and [Ingest sample data into Azure Data Explorer](#).

The StormEvents sample data set contains weather-related data from the [National Centers for Environmental Information](#).

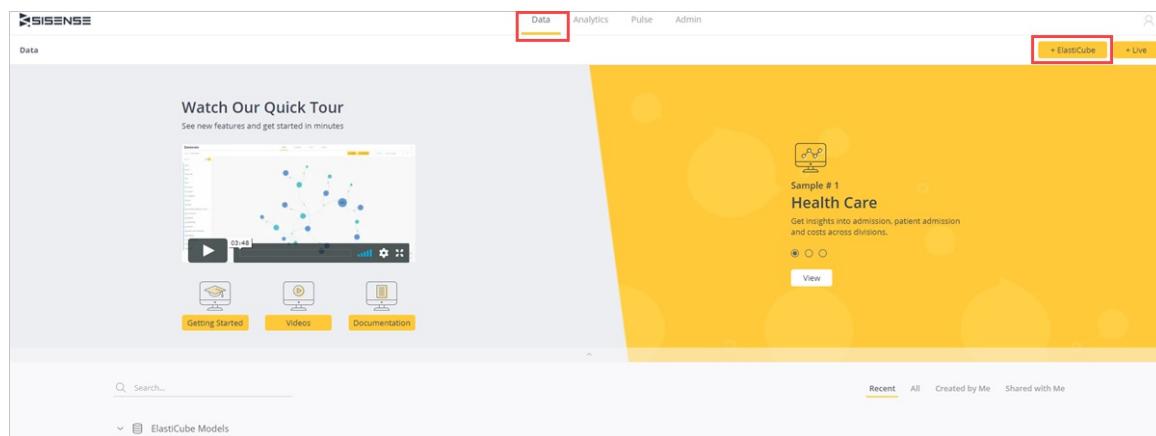
Connect to Sisense dashboards using Azure Data Explorer JDBC connector

1. Download and copy the latest versions of the following jar files to ..\Sisense\DataConnectors\jdbcdrivers\adx

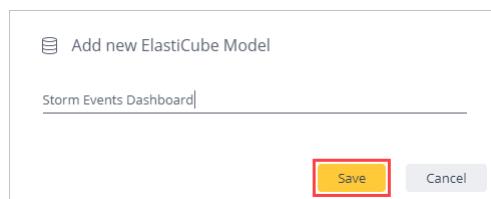
- activation-1.1.jar
- adal4j-1.6.0.jar
- commons-codec-1.10.jar
- commons-collections4-4.1.jar
- commons-lang3-3.5.jar
- gson-2.8.0.jar
- jcip-annotations-1.0-1.jar
- json-smart-1.3.1.jar
- lang-tag-1.4.4.jar
- mail-1.4.7.jar
- mssql-jdbc-7.2.1.jre8.jar
- nimbus-jose-jwt-7.0.1.jar
- oauth2-oidc-sdk-5.24.1.jar
- slf4j-api-1.7.21.jar

2. Open **Sisense** app.

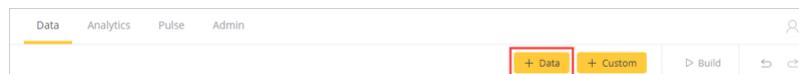
3. Select **Data** tab and select **+ElastiCube** to create a new ElastiCube model.



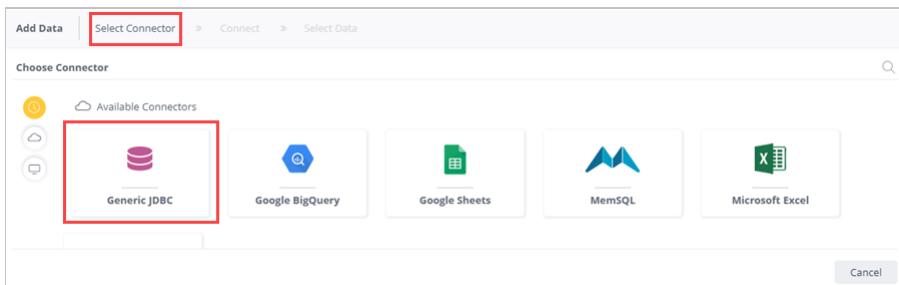
4. In **Add new ElastiCube Model**, name the ElastiCube model and **Save**.



5. Select **+ Data**.



6. In **Select Connector** tab, select **Generic JDBC** connector.



7. In the **Connect** tab, complete the following fields for the **Generic JDBC** connector and select **Next**.

FIELD	DESCRIPTION
Connection String	jdbc:sqlserver://<cluster_name.region>.kusto.windows.net:1433;database=<database_name>;encrypt=true;trustServerCertificate=false;hostNameInCertificate=*.kusto.windows.net;
JDBC JARs folder	..\Sisense\DataConnectors\jdbcdrivers\adx
Driver's Class Name	com.microsoft.sqlserver.jdbc.SQLServerDriver
User Name	AAD user name
Password	AAD user password

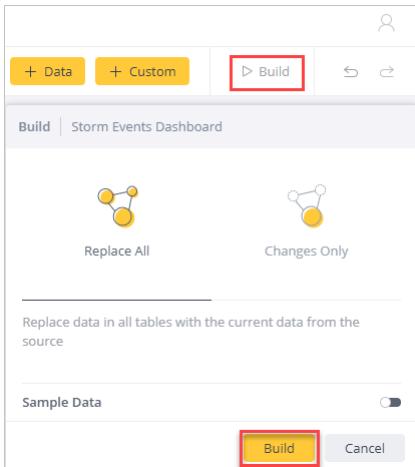
8. In the **Select Data** tab, search **Select Database** to select the relevant database to which you have permissions. In this example, select **test1**.

9. In **test** (database name) pane:

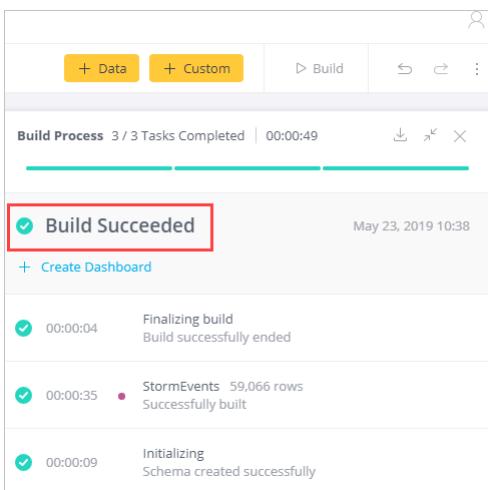
- Select the table name to preview the table and see the table column names. You can remove unnecessary columns.
- Select the check box of the relevant table to select that table.
- Select **Done**.

10. Select **Build** to build your dataset.

- In the Build window, select **Build**.

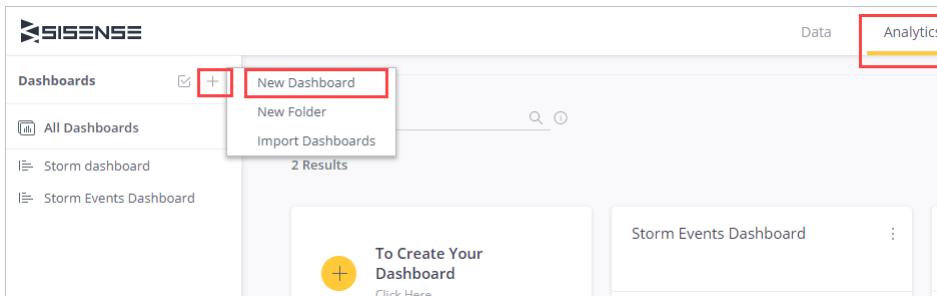


- Wait until build process is complete and then select **Build Succeeded**.

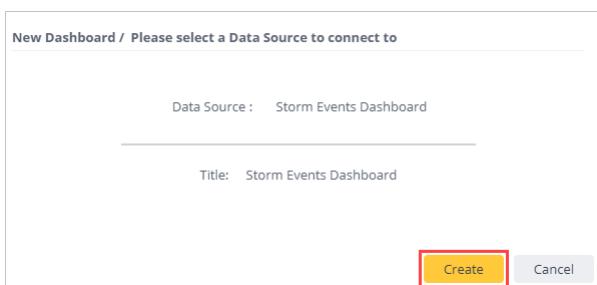


Create Sisense dashboards

- In **Analytics** tab, select + > **New Dashboard** to create dashboards on this dataset.



- Pick a dashboard and select **Create**.



- Under **New Widget**, select + **Select Data** to create a new widget.

New Widget

+ Select Data

You might be interested in:

Add Title

Type to search for fields

- # DeathsIndirect
- # EndLat
- A EndLocation
- # EndLon
- A EndTime
- # EpisodeId
- A EpisodeNarrative
- # EventId
- A EventNarrative
- A EventType
- # InjuriesDirect
- # InjuriesIndirect
- A Source
- A StartTime
- A State
- A StormSummary

Advanced Configuration

Cancel

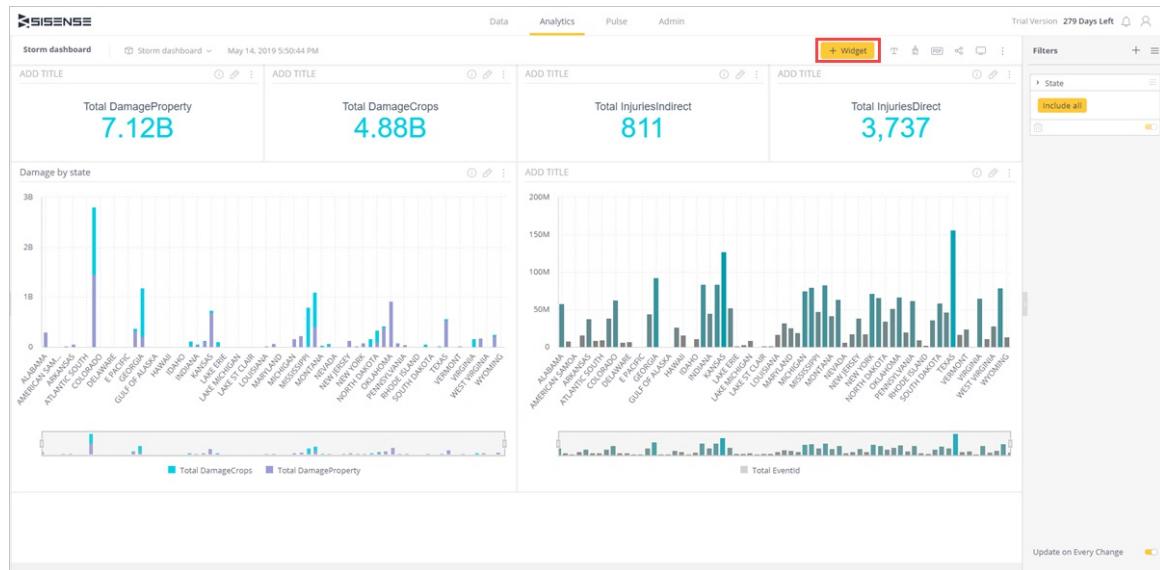
4. Select **+ Add More Data** to add additional columns to your graph.

Storm Events Dashboard | May 23, 2019 10:40:01 AM

New Widget

Total InjuriesDirect | + Add More Data...

5. Select **+ Widget** to create another widget. Drag and drop widgets to rearrange your dashboard.



You can now explore your data with visual analytics, build additional dashboards, and transform data into actionable insights to make an impact on your business.

Next steps

- [Write queries for Azure Data Explorer](#)

Visualize data from Azure Data Explorer in Redash

11/4/2019 • 4 minutes to read • [Edit Online](#)

Redash connects and queries your data sources, builds dashboards to visualize data and share them with peers. In this article, you learn how to set up Azure Data Explorer as a data source for Redash, and then visualize data.

Prerequisites

1. [Create cluster and database](#).
2. Ingest data as explained in [ingest sample data into Azure Data Explorer](#). For more ingestion options, see [ingestion overview](#).

Configure the data source

You perform the following steps to configure Azure Data Explorer as a data source for your dashboard tool. We'll cover these steps in more detail in this section:

1. Create an Azure Active Directory (Azure AD) service principal. The service principal is used by your dashboard tool to access the Azure Data Explorer service.
2. Add the Azure AD service principal to the *viewers* role in the Azure Data Explorer database.
3. Specify your dashboard tool connection properties based on information from the Azure AD service principal, then test the connection.

Create a service principal

You can create the service principal in the [Azure portal](#) or using the [Azure CLI](#) command-line experience. Regardless of which method you use, after creation you get values for four connection properties that you'll use in later steps.

Azure portal

1. To create the service principal, follow the instructions in the [Azure portal documentation](#).
 - a. In the [Assign the application to a role](#) section, assign a role type of **Reader** to your Azure Data Explorer cluster.
 - b. In the [Get values for signing in](#) section, copy the three property values covered in the steps: **Directory ID** (tenant ID), **Application ID**, and **Password**.
2. In the Azure portal, select **Subscriptions** then copy the ID for the subscription in which you created the service principal.

The screenshot shows the Azure portal's 'Subscriptions' page. On the left, there's a sidebar with various service icons and a 'Subscriptions' button highlighted with a red box. The main area displays a table of subscriptions. The columns are labeled: SUBSCRIPTION, SUBSCRIPTION ID, MY ROLE, CURRENT C..., and STATUS. The first row shows 'Azure Team Subscr...' with a placeholder '<Subscription Id>' in the 'SUBSCRIPTION ID' column. This row is also highlighted with a red box.

Azure CLI

1. Create a service principal. Set an appropriate scope and a role type of `reader`.

```
az ad sp create-for-rbac --name "https://{{UrlToYourDashboard}}:{PortNumber}" --role "reader" \
--scopes /subscriptions/{{SubID}}/resourceGroups/{{ResourceGroupName}}
```

For more information, see [Create an Azure service principal with Azure CLI](#).

2. The command returns a result set like the following. Copy the three property values: **appId**, **password**, and **tenant**.

```
{
  "appId": "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
  "displayName": "{{UrlToYourDashboard}}:{PortNumber}",
  "name": "https://{{UrlToYourDashboard}}:{PortNumber}",
  "password": "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
  "tenant": "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
}
```

3. Get a list of your subscriptions.

```
az account list --output table
```

Copy the appropriate subscription ID.

Name	CloudName	SubscriptionId	State	IsDefault
Visual Studio Enterprise	AzureCloud	<Subscription Id>	Enabled	True
Azure Team Subscription	AzureCloud	<Subscription Id>	Enabled	False

Add the service principal to the viewers role

Now that you have a service principal, you add it to the *viewers* role in the Azure Data Explorer database. You can perform this task under **Permissions** in the Azure portal, or under **Query** by using a management command.

Azure portal - Permissions

1. In the Azure portal, go to your Azure Data Explorer cluster.
2. In the **Overview** section, select the database with the StormEvents sample data.

The screenshot shows the Azure Data Explorer Cluster - PREVIEW interface. On the left, there's a sidebar with options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Query, Scale out, Properties, Locks, and Automation script. The 'Overview' tab is selected. On the right, it shows details for the 'Resource group (change) docscluster', 'Location West US', and 'Subscription (change) <SubscriptionName>'. Below that, it lists 'Databases from A-Z' under 'DATABASE' with 'TestDatabase' highlighted and surrounded by a red box. At the bottom of the main area, there's a search bar, an 'Add Database' button (also highlighted with a red box), a 'Stop' button, and a 'Refresh' button.

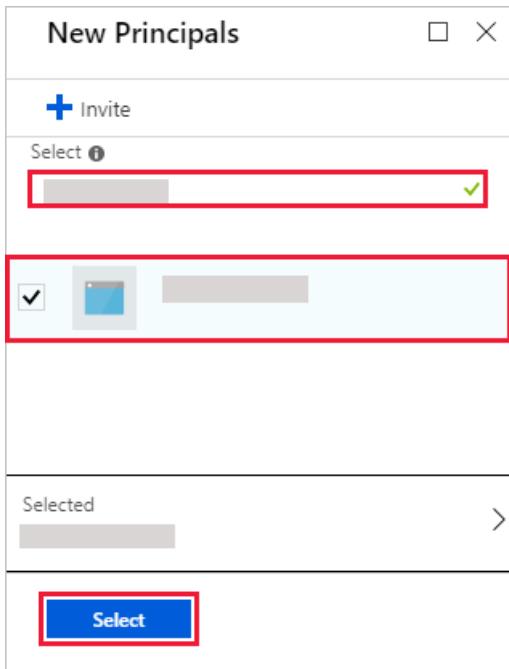
3. Select **Permissions** then **Add**.

The screenshot shows the 'TestDatabase - Permissions' page. The 'Permissions' tab is selected. On the right, there's a 'Role' dropdown set to '6 selected' with a note '2 items (1 Admin, 1 User)'. At the top, there's a search bar, an 'Add' button (highlighted with a red box), a 'Refresh' button, and a 'Remove' button. The sidebar on the left includes 'Overview', 'Permissions' (selected), and 'Query'.

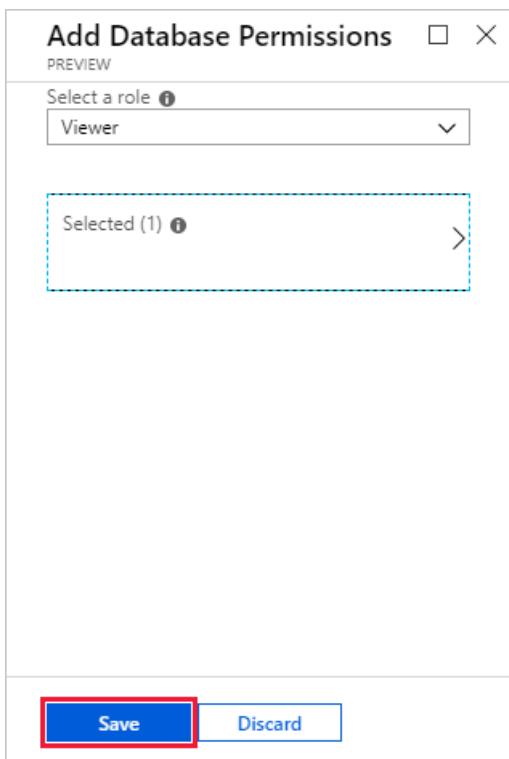
4. Under **Add database permissions**, select the **Viewer** role then **Select principals**.

The screenshot shows the 'Add Database Permissions' dialog. It has a 'Select a role' dropdown set to 'Viewer' and a 'Select principals' step highlighted with a red box. At the bottom, there are 'Save' and 'Discard' buttons.

5. Search for the service principal you created. Select the principal, then **Select**.



6. Select **Save**.



Management command - Query

1. In the Azure portal, go to your Azure Data Explorer cluster, and select **Query**.

The screenshot shows the Azure Data Explorer Cluster - PREVIEW interface. On the left, there is a navigation sidebar with the following items:

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Query** (highlighted with a red border)

- Run the following command in the query window. Use the application ID and tenant ID from the Azure portal or CLI.

```
.add database {TestDatabase} viewers ('aadapp={ApplicationID};{TenantID}')
```

The command returns a result set like the following. In this example, the first row is for an existing user in the database, and the second row is for the service principal that was just added.

Role	PrincipalType	PrincipalDisplayName	PrincipalObjectId	PrincipalFQN
Database TestDatabase Admin	AAD User	<Principal display name>	<Principal object id>	<Principal FQN>
Database TestDatabase Viewer	AAD Application	<Principal display name>	<Principal object id>	<Principal FQN>

Create Azure Data Explorer Connector in Redash

- Sign in to [Redash](#). Select **Get Started** to create an account.
- Under **Let's get started**, Select **Connect a Data Source**.

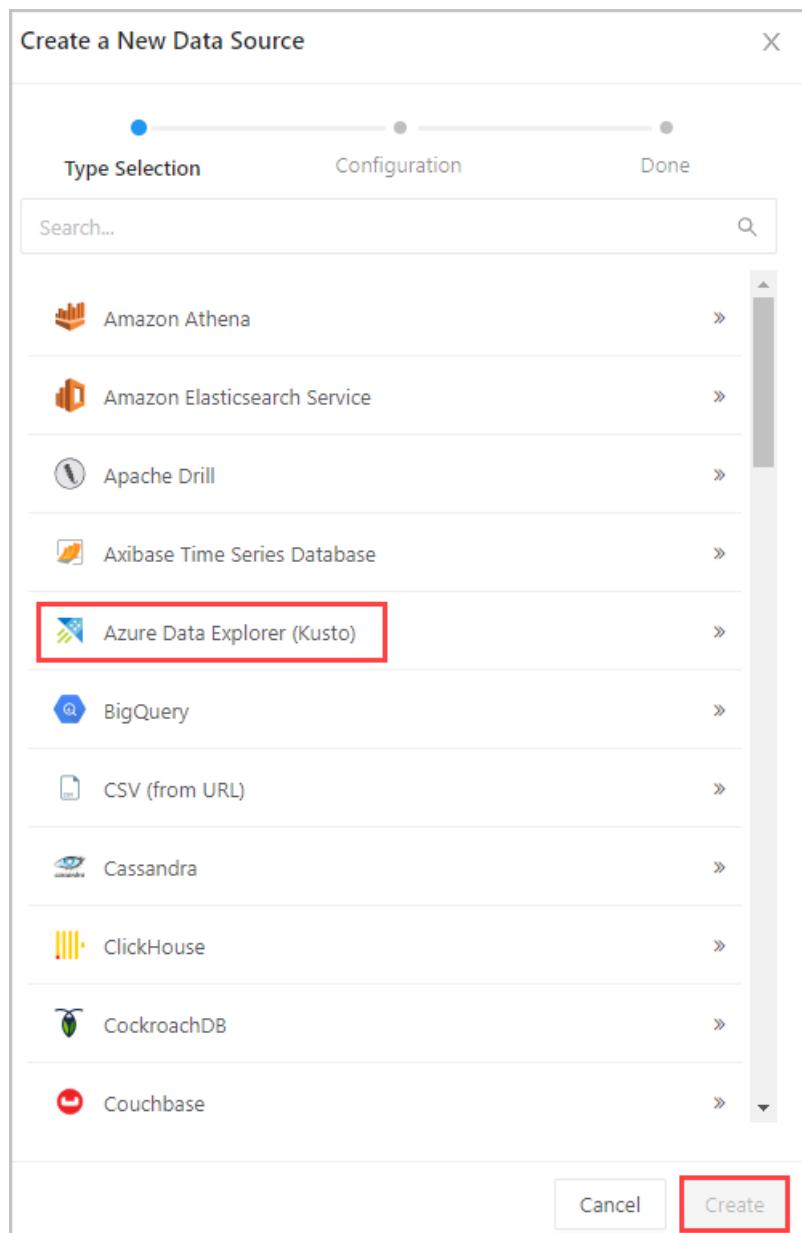
The screenshot shows the Redash 'Let's get started' page. At the top, there is a navigation bar with 'Dashboards', 'Queries', 'Alerts', and a 'Create' button. Below the navigation bar, a yellow banner says: 'We have sent an email with a confirmation link to your email address. Please follow the link to verify your email address. [Resend email](#)'.

On the right side, there is a box titled 'Let's get started' with the following steps:

1. [Connect a Data Source](#)
2. [Create your first Query](#)
3. [Create your first Dashboard](#)
4. [Invite your team members](#)

At the bottom of the box, it says 'Need more support? [See our Help](#)'.

- In **Create a New Data Source** window, select **Azure Data Explorer (Kusto)**, then select **Create**.



4. In **Azure Data Explorer (Kusto)** window, complete the following form and select **Create**.

Create a New Data Source X

Type Selection Configuration Done

Azure Data Explorer (Kusto)

* Name
Github connector ✓

* Cluster
https://demo12.westus.kusto.windows.net ✓

* Azure AD Client ID ✓

* Azure AD Client Secret ✓

* Azure AD Tenant Id ✓

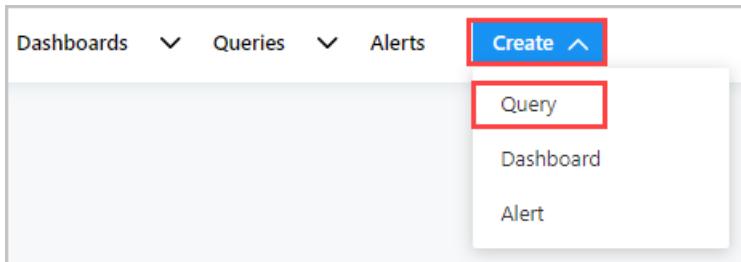
* Database
GitHub ✓

Previous Create

- In **Settings** window, select **Save** and **Test Connection** to test your **Azure Data Explorer (Kusto)** data source connection.

Create queries in Redash

- On top left of Redash, select **Create > Query**. Click on **New Query** and rename the query.



- Type your query in the top editing pane and select **Save** and **Execute**. Select **Publish** to publish query for future use.

The screenshot shows the 'New Query' interface. On the left, a dropdown menu is set to 'Github connector', with a red box highlighting it. Below the dropdown is a search bar labeled 'Search schema...'. To the right is a list of event types: CommitCommentEvent, CreateEvent, DeleteEvent, EventsFromLiveStream, FollowEvent, ForkEvent, GithubEvent, GollumEvent, IssueCommentEvent, IssuesEvent, MemberEvent, PublicEvent, PullRequestEvent, PullRequestReviewCommentEvent, PushEvent, ReleaseEvent, and WatchEvent. A red box highlights this list. On the right, the query editor contains the following code:

```
1 GithubEvent  
2 | summarize count () by Type|
```

Below the code are three small icons: {{}}, ⚙️, and ⚡. To the right are 'Save*' and 'Execute' buttons.

In the left pane, you can see the data source connection name (**Github connector** in our flow) in the drop-down menu, and the tables in the selected database.

3. View the query results in the bottom central pane. Create a visualization to go with the query by selecting the **New Visualization** button.

The screenshot shows the 'New Query' interface with the results table highlighted by a red box. The table has two columns: 'Type' and 'count_'. The data is as follows:

Type	count_
WatchEvent	99,456,940
PullRequestEvent	71,051,017
IssuesEvent	45,585,696
CreateEvent	173,667,880
PushEvent	666,191,662
DeleteEvent	30,984,087
IssueCommentEvent	89,038,491
GollumEvent	7,627,541
ForkEvent	35,304,190
PullRequestReviewCommentEvent	22,687,859

At the bottom of the table, it says '15 rows 1 second runtime' and 'Updated just now'. Above the table, there are two tabs: 'Table' (selected) and '+ New Visualization'. The '+ New Visualization' tab is highlighted with a red box.

4. In the visualization screen, select the **Visualization Type** and the relevant fields such as **X Column** and **Y Column**. **Save** the visualization.

The screenshot shows the 'Visualization Editor' interface. The top left contains fields for 'Visualization Type' (set to 'Chart') and 'Visualization Name' (set to 'Chart'). Below these are tabs for 'General', 'X Axis', 'Y Axis', 'Series', 'Colors', and 'Data Labels'. The 'X Axis' tab is active, showing 'Chart Type' set to 'Bar', 'X Column' set to 'Type', and 'Y Columns' set to 'count_'. To the right is a bar chart titled 'count_'. The X-axis labels include 'CommitCommentEvent', 'CreateEvent', 'DeleteEvent', 'FollowEvent', 'ForkEvent', 'GollumEvent', 'IssueCommentEvent', 'IssuesEvent', 'MemberEvent', 'PublicEvent', 'PullEvent', 'PullRequestEvent', 'PushEvent', 'PushEventReviewCommentEvent', 'ReleaseEvent', and 'WatchEvent'. The Y-axis ranges from 0 to 700M. A legend indicates a single series named 'count_'. At the bottom right of the editor are 'Cancel' and 'Save' buttons, with 'Save' highlighted by a red box.

Create a query using a parameter

1. **Create > Query** to create a new query. Add a parameter to it using `{()}` curly brackets. Select `{()}` to open **Add Parameter** window. You can also select the *settings icon* to modify the attributes of an existing parameter and open the `<parameter_name>` window.

The screenshot shows the Kibana Query editor. The top navigation bar includes 'Dashboards', 'Queries', 'Alerts', and a 'Create' button. The main area is titled 'Top 5 popular events'. On the left is a sidebar with a 'Github connector' dropdown and a 'Search schema...' input field. The central area contains a code editor with the following query:

```

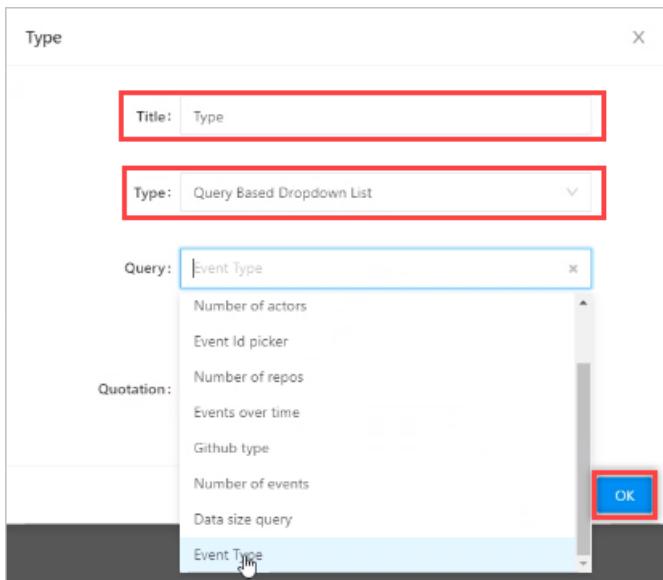
1 GithubEvent
2 | where Type == |
3 //| where Type in ((split('({Type})', ',')))
4 | summarize Events=count() by RepoName = tolower(tostring(Repo.name))
5 | top 5 by Events

```

A red box highlights the curly braces `{()}` in the query. Below the code editor is a 'Save' and 'Execute' button. To the right is a table titled 'Table' with the heading 'Top 5 Popular Repos'. The table lists five repositories with their event counts:

RepoName	Events
freetecodecamp/freetecodecamp	312,575
koorellasuresh/ukregiontest	262,493
microsoft/vscode	185,102
ron190/sql-injection	172,172
tensorflow/tensorflow	145,927

2. Name your parameter. Select **Type: Query Based Dropdown List** from dropdown menu. Select **OK**



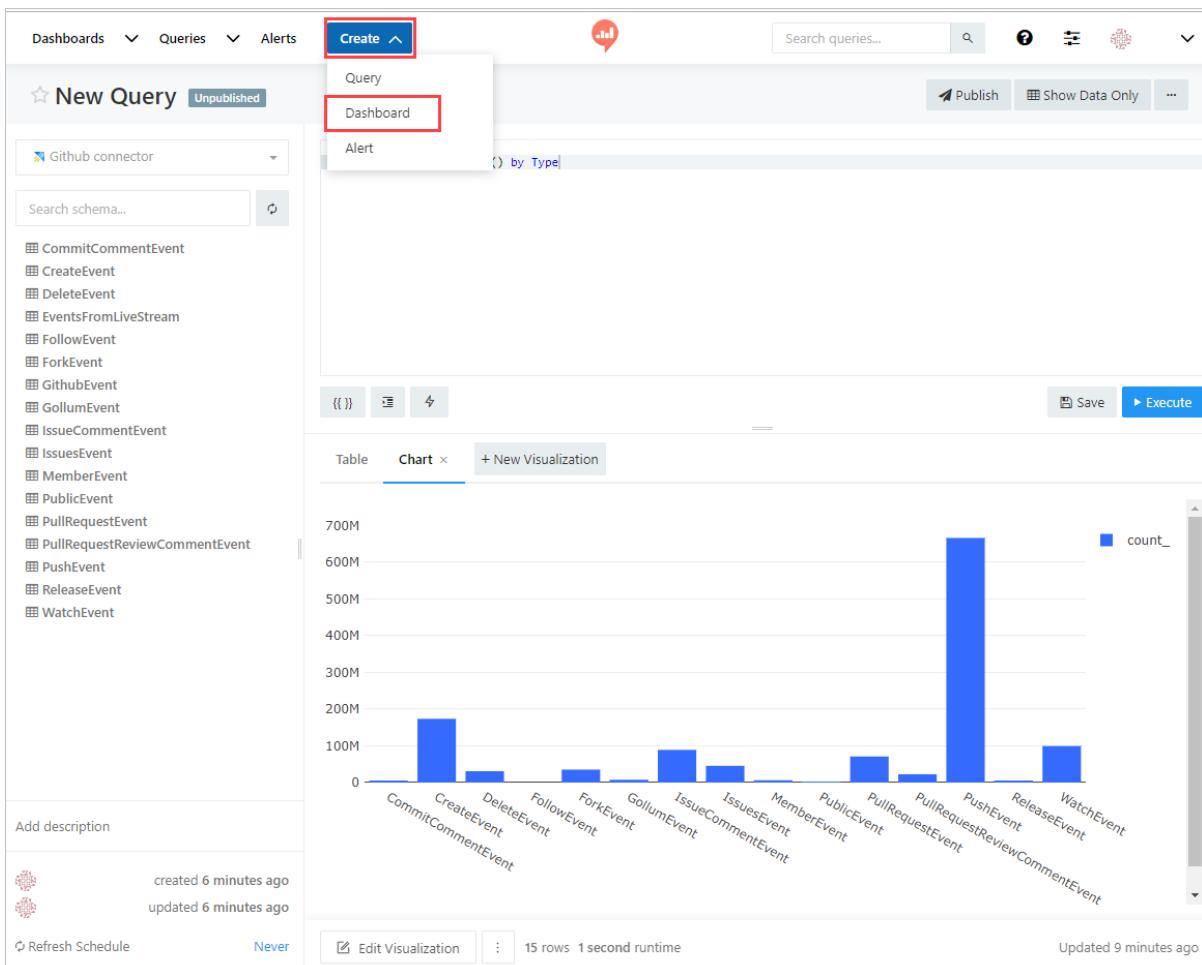
NOTE

The query uses multiple values, therefore you must include the following syntax

`| where Type in ((split('{{Type}}', ',')))`. For more information, see [in operator](#). This results in [multiple query parameter options in redash app](#)

Create a dashboard in Redash

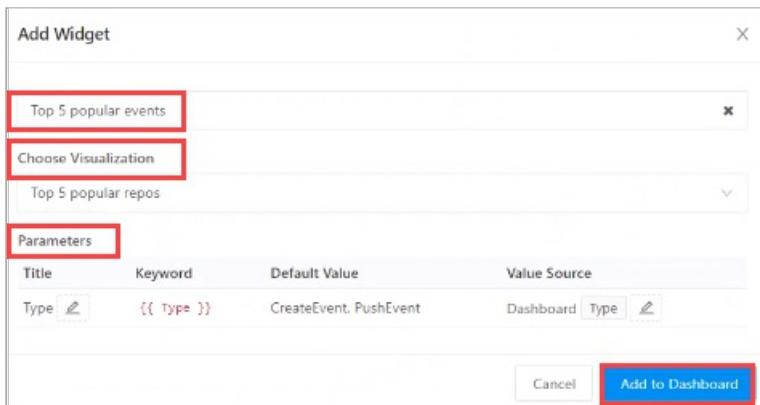
- To create your dashboard, **Create > Dashboard**. Alternatively, select existing dashboard, **Dashboards > select a dashboard from the list.**



- In **New Dashboard** window, name your dashboard and select **Save**. In **<Dashboard_name>** window,

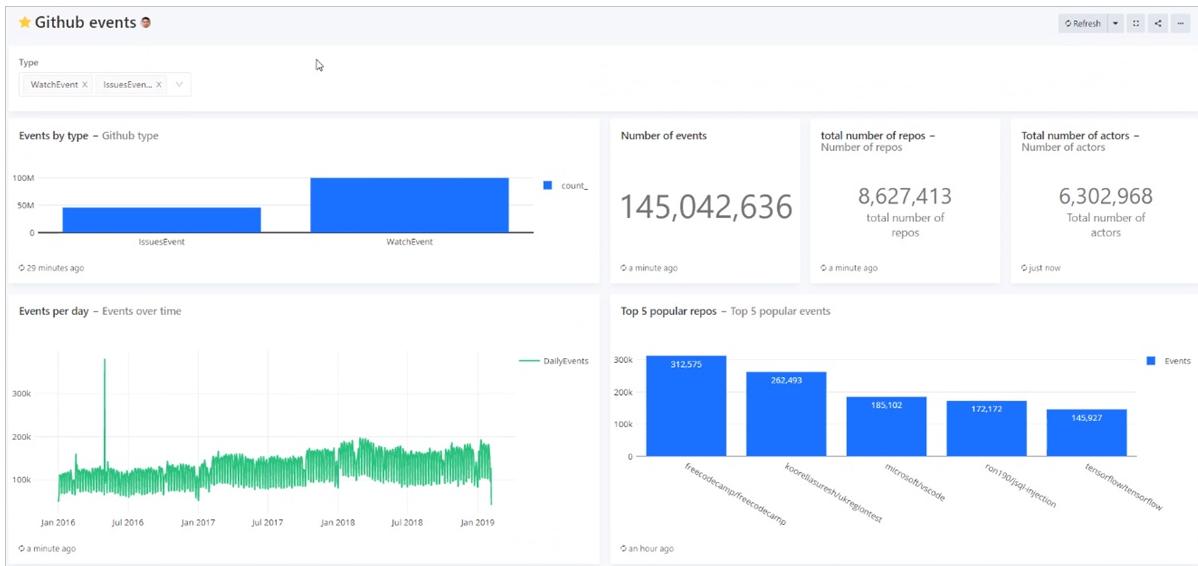
select **Add Widget** to create a new widget.

3. In **Add Widget** window, select query name, **Choose Visualization**, and **Parameters**. Select **Add to Dashboard**



4. Select **Done Editing** to complete dashboard creation.

5. In the dashboard edit mode, select **Use Dashboard Level Filters** to use the **Type** parameter previously defined.



Next steps

- [Write queries for Azure Data Explorer](#)

Select the correct VM SKU for your Azure Data Explorer cluster

8/30/2019 • 3 minutes to read • [Edit Online](#)

When you create a new cluster or optimize a cluster for a changing workload, Azure Data Explorer offers multiple virtual machine (VM) SKUs to choose from. The VMs have been carefully chosen to give you the most optimal cost for any workload.

The size and VM SKU of the data-management cluster are fully managed by the Azure Data Explorer service. They're determined by such factors as the engine's VM size and the ingestion workload.

You can change the VM SKU for the engine cluster at any time by [scaling up the cluster](#). It's best to start with the smallest SKU size that fits the initial scenario. Keep in mind that scaling up the cluster results in a downtime of up to 30 minutes while the cluster is re-created with the new VM SKU.

TIP

Compute [Reserved Instances \(RI\)](#) is applicable to the Azure Data Explorer cluster.

This article describes various VM SKU options and provides the technical details that can help you make the best choice.

Select a cluster type

Azure Data Explorer offers two types of clusters:

- **Production:** Production clusters contain two nodes for engine and data-management clusters and are operated under the Azure Data Explorer [SLA](#).
- **Dev/Test (no SLA):** Dev/Test clusters have a single D11 v2 node for the engine cluster and a single D1 node for the data-management cluster. This cluster type is the lowest cost configuration because of its low instance count and no engine markup charge. There's no SLA for this cluster configuration, because it lacks redundancy.

SKU types

When you create an Azure Data Explorer cluster, select the *optimal* VM SKU for the planned workload. You can choose from the following two Azure Data Explorer SKU families:

- **D v2:** The D SKU is compute-optimized and comes in two flavors:
 - The VM itself
 - The VM bundled with premium storage disks
- **Ls:** The L SKU is storage-optimized. It has a much greater SSD size than the similarly priced D SKU.

The key differences between the available SKU types are described in the following table:

ATTRIBUTE	D SKU	L SKU
Small SKUs	Minimal size is D11 with two cores	Minimal size is L4 with four cores

ATTRIBUTE	D SKU	L SKU
Availability	Available in all regions (the DS+PS version has more limited availability)	Available in a few regions
Cost per GB cache per core	High with the D SKU, low with the DS+PS version	Lowest with the Pay-As-You-Go option
Reserved Instances (RI) pricing	High discount (over 55 percent for a three-year commitment)	Lower discount (20 percent for a three-year commitment)

Select your cluster VM

To select your cluster VM, [configure vertical scaling](#).

With various VM SKU options to choose from, you can optimize costs for the performance and hot-cache requirements for your scenario.

- If you need the most optimal performance for a high query volume, the ideal SKU should be compute-optimized.
- If you need to query large volumes of data with relatively lower query load, the storage-optimized SKU can help reduce costs and still provide excellent performance.

Because the number of instances per cluster for the small SKUs is limited, it's preferable to use larger VMs that have greater RAM. More RAM is needed for some query types that put more demand on the RAM resource, such as queries that use `joins`. Therefore, when you're considering scaling options, we recommend that you scale up to a larger SKU rather than scale out by adding more instances.

VM options

The technical specifications for the Azure Data Explorer cluster VMs are described in the following table:

NAME	CATEGORY	SSD SIZE	CORES	RAM	PREMIUM STORAGE DISKS (1 TB)	MINIMUM INSTANCE COUNT PER CLUSTER	MAXIMUM INSTANCE COUNT PER CLUSTER
D11 v2	compute-optimized	75 GB	2	14 GB	0	1	8 (except for dev/test SKU, which is 1)
D12 v2	compute-optimized	150 GB	4	28 GB	0	2	16
D13 v2	compute-optimized	307 GB	8	56 GB	0	2	1,000
D14 v2	compute-optimized	614 GB	16	112 GB	0	2	1,000
DS13 v2 + 1 TB PS	storage-optimized	1 TB	8	56 GB	1	2	1,000
DS13 v2 + 2 TB PS	storage-optimized	2 TB	8	56 GB	2	2	1,000

Name	Category	SSD Size	Cores	RAM	Premium Storage Disks (1 TB)	Minimum Instance Count per Cluster	Maximum Instance Count per Cluster
DS14 v2 + 3 TB PS	storage-optimized	3 TB	16	112 GB	2	2	1,000
DS14 v2 + 4 TB PS	storage-optimized	4 TB	16	112 GB	4	2	1,000
L4s v1	storage-optimized	650 GB	4	32 GB	0	2	16
L8s v1	storage-optimized	1.3 TB	8	64 GB	0	2	1,000
L16s_1	storage-optimized	2.6 TB	16	128 GB	0	2	1,000

- You can view the updated VM SKU list per region by using the Azure Data Explorer [ListSkus API](#).
- Learn more about the [various SKUs](#).

Next steps

- You can [scale up or scale down](#) the engine cluster at any time by changing the VM SKU, depending on differing needs.
- You can [scale in or scale out](#) the size of the engine cluster to alter capacity, depending on changing demands.

Manage cluster horizontal scaling (scale out) in Azure Data Explorer to accommodate changing demand

12/9/2019 • 4 minutes to read • [Edit Online](#)

Sizing a cluster appropriately is critical to the performance of Azure Data Explorer. A static cluster size can lead to under-utilization or over-utilization, neither of which is ideal. Because demand on a cluster can't be predicted with absolute accuracy, it's better to *scale* a cluster, adding and removing capacity and CPU resources with changing demand.

There are two workflows for scaling an Azure Data Explorer cluster:

- Horizontal scaling, also called scaling in and out.
- [Vertical scaling](#), also called scaling up and down. This article explains the horizontal scaling workflow.

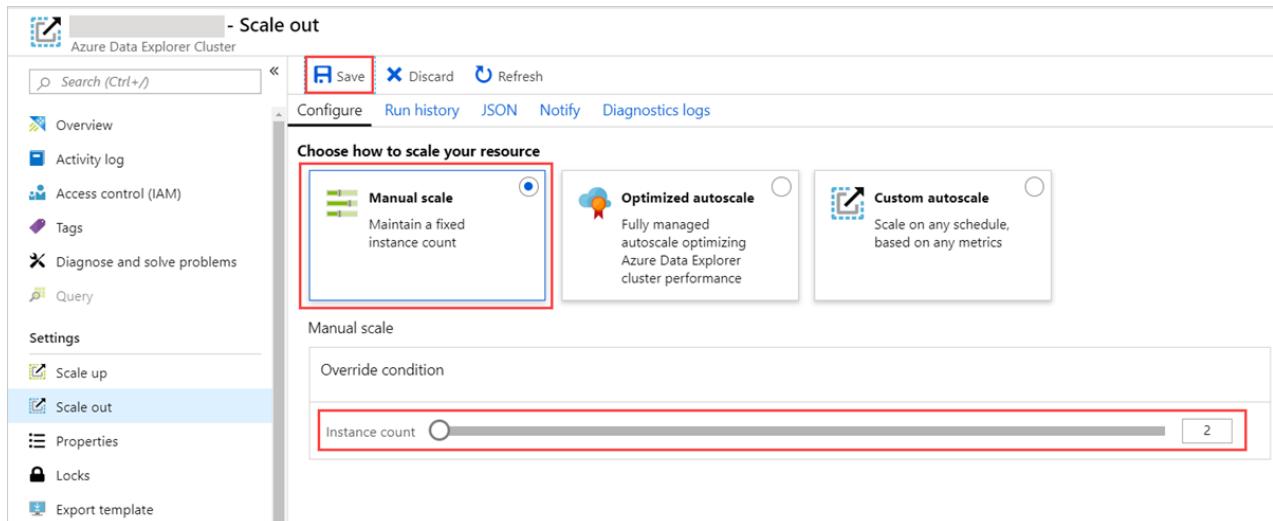
Configure horizontal scaling

By using horizontal scaling, you can scale the instance count automatically, based on predefined rules and schedules. To specify the autoscale settings for your cluster:

1. In the Azure portal, go to your Azure Data Explorer cluster resource. Under **Settings**, select **Scale out**.
2. In the **Scale out** window, select the autoscale method that you want: **Manual scale**, **Optimized autoscale**, or **Custom autoscale**.

Manual scale

Manual scale is the default setting during cluster creation. The cluster has a static capacity that doesn't change automatically. You select the static capacity by using the **Instance count** bar. The cluster's scaling remains at that setting until you make another change.



Optimized autoscale (preview)

Optimized autoscale is the recommended autoscale method. This method optimizes cluster performance and costs. If the cluster approaches a state of under-utilization, it will be scaled in. This action lowers costs but keeps performance level. If the cluster approaches a state of over-utilization, it will be scaled out to maintain optimal performance. To configure Optimized autoscale:

1. Select **Optimized autoscale**.

2. Select a minimum instance count and a maximum instance count. The cluster auto-scaling ranges between those two numbers, based on load.

3. Select **Save**.

The screenshot shows the Azure Data Explorer Cluster configuration interface for scaling out. The left sidebar has options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Query, Settings, Scale up, Scale out (which is selected), Properties, Locks, and Export template. The main area is titled '- Scale out' and shows 'Choose how to scale your resource'. It offers 'Manual scale' (Maintain a fixed instance count) and 'Optimized autoscale' (Fully managed autoscale optimizing Azure Data Explorer cluster performance, which is selected). Below that, 'Optimized autoscale' settings show 'Minimum instance count' as 2 and 'Maximum instance count' as 10. A note at the bottom says: 'This Azure Data Explorer cluster will be scaled intelligently between the minimum and maximum instance counts, based on load.'

Optimized autoscale starts working. Its actions are now visible in the Azure activity log of the cluster.

Logic of optimized autoscale

Scale out

When your cluster approaches a state of over-utilization, scale out to maintain optimal performance. Scale out will occur when:

- The number of cluster instances is below the maximum number of instances defined by the user.
- The cache utilization is high for over an hour.

NOTE

The scale out logic doesn't currently consider the ingestion utilization and CPU metrics. If those metrics are important for your use case, use [custom autoscale](#).

Scale in

When your cluster approaches a state of under-utilization, scale in to lower costs but maintain performance.

Multiple metrics are used to verify that it's safe to scale in the cluster. The following rules are evaluated daily for 7 days before scale in is performed:

- The number of instances is above 2 and above the minimum number of instances defined.
- To ensure that there's no overloading of resources, the following metrics must be verified before scale in is performed:
 - Cache utilization isn't high
 - CPU is below average
 - Ingestion utilization is below average
 - Streaming ingest utilization (if streaming ingest is used) isn't high
 - Keep alive events are above a defined minimum, processed properly, and on time.
 - No query throttling
 - Number of failed queries are below a defined minimum.

NOTE

The scale in logic currently requires a 7-day evaluation before implementation of optimized scale in. This evaluation takes place once every 24 hours. If a quick change is needed, use [manual scale](#).

Custom autoscale

By using custom autoscale, you can scale your cluster dynamically based on metrics that you specify. The following graphic shows the flow and steps to configure custom autoscale. More details follow the graphic.

1. In the **Autoscale setting name** box, enter a name, such as *Scale-out: cache utilization*.

The screenshot shows the Azure portal interface for configuring a custom autoscale setting. The main page has a 'Save' button highlighted with a red box (1). Below it, there's a 'Scale mode' section with two radio buttons: 'Scale based on a metric' (selected) and 'Scale to a specific instance count' (2). A note below says 'Scale out and scale in your instances based on metric. For example: 'Add a rule that above 70%'.

In the 'Rules' section (3), there's a '+ Add a rule' button. Below it, 'Instance limits' are set to Minimum 1, Maximum 1, and Default 1. A note says 'It is recommended to have at least one scale in rule'.

The 'Schedule' section (6) notes that this scale condition is executed when none of the other scale condition(s) match.

The 'Scale rule' dialog on the right contains the following settings:

- Criteria:** Time aggregation: Average, Metric name: Cache Utilization, 1 minute time grain.
- Action:** Operation: Increase count by 1, Instance count: 1, Cool down (minutes): 5.

A large red box highlights the 'Criteria' and 'Action' sections, and a blue box highlights the 'Add' button at the bottom right of the dialog (5).

2. For **Scale mode**, select **Scale based on a metric**. This mode provides dynamic scaling. You can also select **Scale to a specific instance count**.
3. Select **+ Add a rule**.
4. In the **Scale rule** section on the right, enter values for each setting.

Criteria

SETTING	DESCRIPTION AND VALUE
Time aggregation	Select an aggregation criteria, such as Average .

SETTING	DESCRIPTION AND VALUE
Metric name	Select the metric you want the scale operation to be based on, such as Cache Utilization .
Time grain statistic	Choose between Average , Minimum , Maximum , and Sum .
Operator	Choose the appropriate option, such as Greater than or equal to .
Threshold	Choose an appropriate value. For example, for cache utilization, 80 percent is a good starting point.
Duration (in minutes)	Choose an appropriate amount of time for the system to look back when calculating metrics. Start with the default of 10 minutes.

Action

SETTING	DESCRIPTION AND VALUE
Operation	Choose the appropriate option to scale in or scale out.
Instance count	Choose the number of nodes or instances you want to add or remove when a metric condition is met.
Cool down (minutes)	Choose an appropriate time interval to wait between scale operations. Start with the default of five minutes.

5. Select **Add**.

6. In the **Instance limits** section on the left, enter values for each setting.

SETTING	DESCRIPTION AND VALUE
Minimum	The number of instances that your cluster won't scale below, regardless of utilization.
Maximum	The number of instances that your cluster won't scale above, regardless of utilization.
Default	The default number of instances. This setting is used if there are problems with reading the resource metrics.

7. Select **Save**.

You've now configured horizontal scaling for your Azure Data Explorer cluster. Add another rule for vertical scaling. If you need assistance with cluster-scaling issues, [open a support request](#) in the Azure portal.

Next steps

- Monitor Azure Data Explorer performance, health, and usage with metrics
- Manage cluster vertical scaling for appropriate sizing of a cluster.

Manage cluster vertical scaling (scale up) in Azure Data Explorer to accommodate changing demand

7/15/2019 • 2 minutes to read • [Edit Online](#)

Sizing a cluster appropriately is critical to the performance of Azure Data Explorer. A static cluster size can lead to under-utilization or over-utilization, neither of which is ideal.

Since demand on a cluster can't be predicted with absolute accuracy, a better approach is to *scale* a cluster, adding and removing capacity and CPU resources with changing demand.

There are two workflows for scaling an Azure Data Explorer cluster:

- [Horizontal scaling](#), also called scaling in and out.
- Vertical scaling, also called scaling up and down.

This article explains the vertical scaling workflow:

Configure vertical scaling

1. In the Azure portal, go to your Azure Data Explorer cluster resource. Under **Settings**, select **Scale up**.
2. In the **Scale up** window, you will see a list of available SKUs for your cluster. For example, in the following figure, only four SKUs are available.

Select compute specifications							
Available plans							
The price is an estimate of the cluster's virtual machines and Azure Data Explorer service costs. Other costs are not included. Please see Azure calculator page for an estimate and the Azure Data Explorer pricing page for full pricing information.							
PAY ATTENTION: * The scale up process can take a few minutes. During that time your cluster will be suspended. * Scale down can harm your cluster performance.							
D11_V2	D12_V2	D13_V2	D14_V2	L4	L8	L16	
2 vCPUs \$0.185/h \$0.11/h	4 vCPUs \$0.371/h \$0.22/h	8 vCPUs \$0.747/h \$0.44/h	16 vCPUs \$1.625/h \$0.88/h	4 vCPUs \$0.374/h This SKU is not supported in your cluster's location	8 vCPUs \$0.748/h This SKU is not supported in your cluster's location	16 vCPUs \$1.736/h This SKU is not supported in your cluster's location	
75 GB Cache	150 GB Cache	307 GB Cache	614 GB Cache	650 GB Cache	1.3 TB Cache	2.7 TB Cache	
14 GB Ram	28 GB Ram	56 GB Ram	112 GB Ram	32 GB Ram	64 GB Ram	128 GB Ram	
USD/H (ESTIMATED)	USD/H (ESTIMATED)	USD/H (ESTIMATED)	USD/H (ESTIMATED)	USD/H (ESTIMATED)	USD/H (ESTIMATED)	USD/H (ESTIMATED)	USD/H (ESTIMATED)
0.30	0.59	1.18	2.36	0.59	1.19	2.38	

The SKUs are disabled because they're the current SKU, or they aren't available in the region where the cluster is located.

3. To change your SKU, select a new SKU and click **Select**.

NOTE

- The vertical scaling process can take a few minutes, and during that time your cluster will be suspended.
- Scaling down can harm your cluster performance.
- The price is an estimate of the cluster's virtual machines and Azure Data Explorer service costs. Other costs are not included. See Azure Data Explorer [cost estimator](#) page for an estimate and the Azure Data Explorer [pricing page](#) for full pricing information.

You've now configured vertical scaling for your Azure Data Explorer cluster. Add another rule for a horizontal

scaling. If you need assistance with cluster-scaling issues, [open a support request](#) in the Azure portal.

Next steps

- [Manage cluster horizontal scaling](#) to dynamically scale out the instance count based on metrics that you specify.
- Monitor your resource usage by following this article: [Monitor Azure Data Explorer performance, health, and usage with metrics](#).

Secure your cluster in Azure Data Explorer

8/30/2019 • 2 minutes to read • [Edit Online](#)

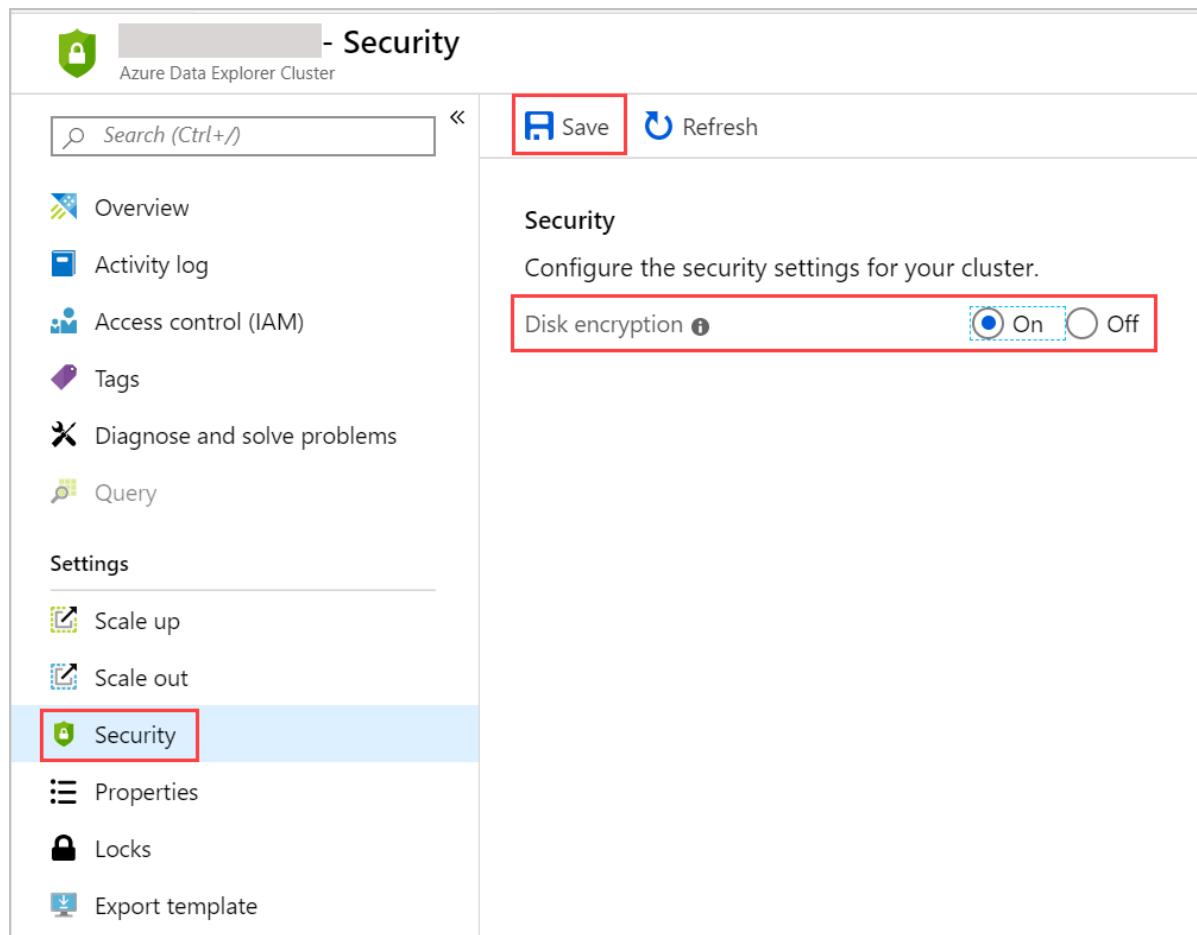
Azure Disk Encryption helps protect and safeguard your data to meet your organizational security and compliance commitments. It provides volume encryption for the OS and data disks of your cluster virtual machines. It also integrates with [Azure Key Vault](#) which allows us to control and manage the disk encryption keys and secrets, and ensure all data on the VM disks is encrypted at rest while in Azure Storage.

Your cluster security settings allow you to enable disk encryption on your cluster.

Enable encryption at rest

Enabling [encryption at rest](#) on your cluster provides data protection for stored data (at rest).

1. In the Azure portal, go to your Azure Data Explorer cluster resource. Under the **Settings** heading, select **Security**.



The screenshot shows the Azure Data Explorer Cluster - Security settings page. On the left, there's a sidebar with various options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Query, Scale up, Scale out, Security (which is highlighted with a red box), Properties, Locks, and Export template. The main area is titled 'Security' and contains the instruction 'Configure the security settings for your cluster.' Below this is a section for 'Disk encryption' with two radio buttons: 'On' (selected) and 'Off'. At the top right of the main area are 'Save' and 'Refresh' buttons.

2. In the **Security** window, select **On** for the **Disk encryption** security setting.
3. Select **Save**.

NOTE

Select **Off** to disable the encryption after it has been enabled.

Next steps

[Check cluster health](#)

Check the health of an Azure Data Explorer cluster

4/4/2019 • 2 minutes to read • [Edit Online](#)

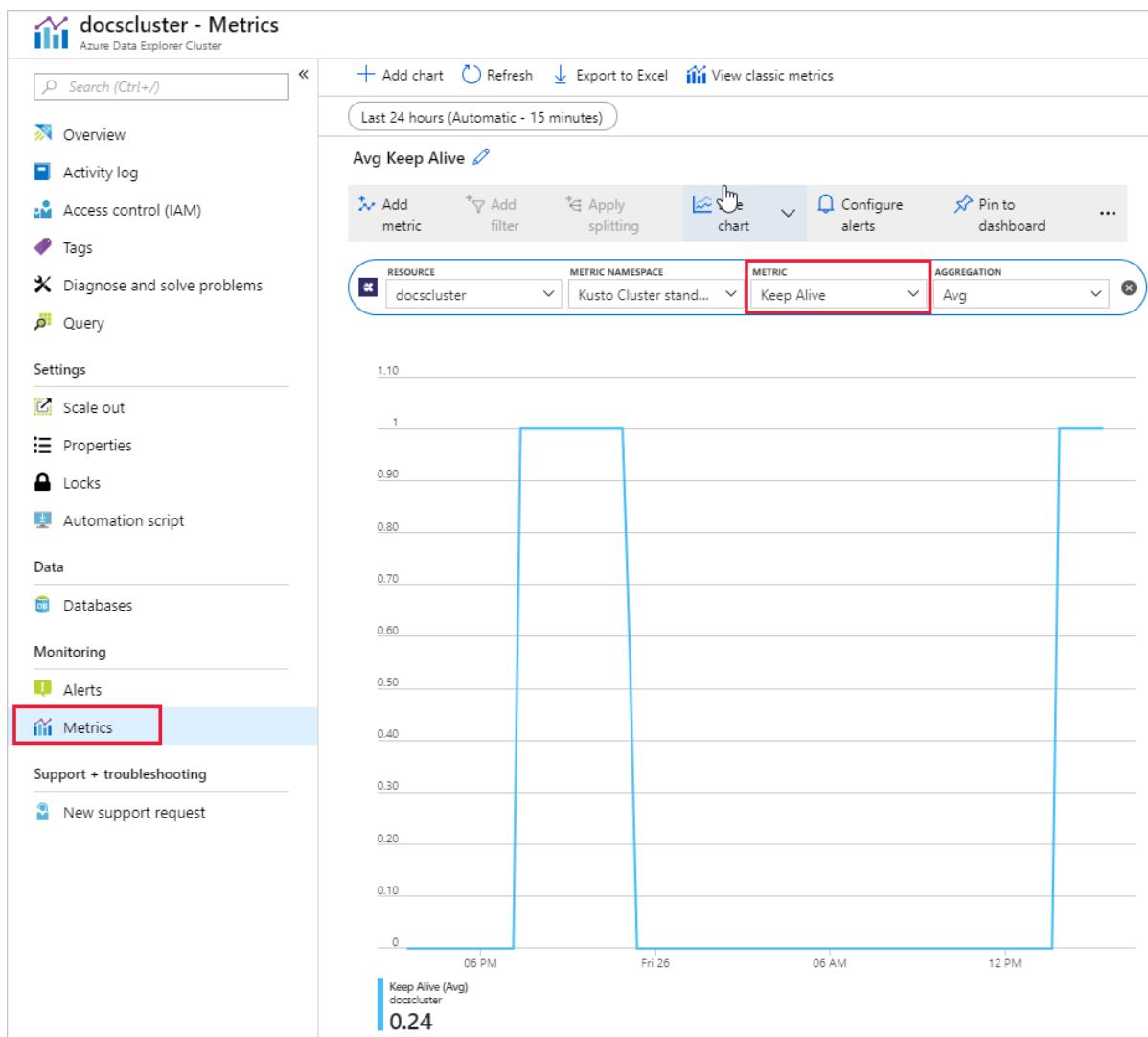
There are several factors that impact the health of an Azure Data Explorer cluster, including CPU, memory, and the disk subsystem. This article shows some basic steps you can take to gauge the health of a cluster.

1. Sign in to <https://dataexplorer.azure.com>.
2. In the left pane, select your cluster, and run the following command.

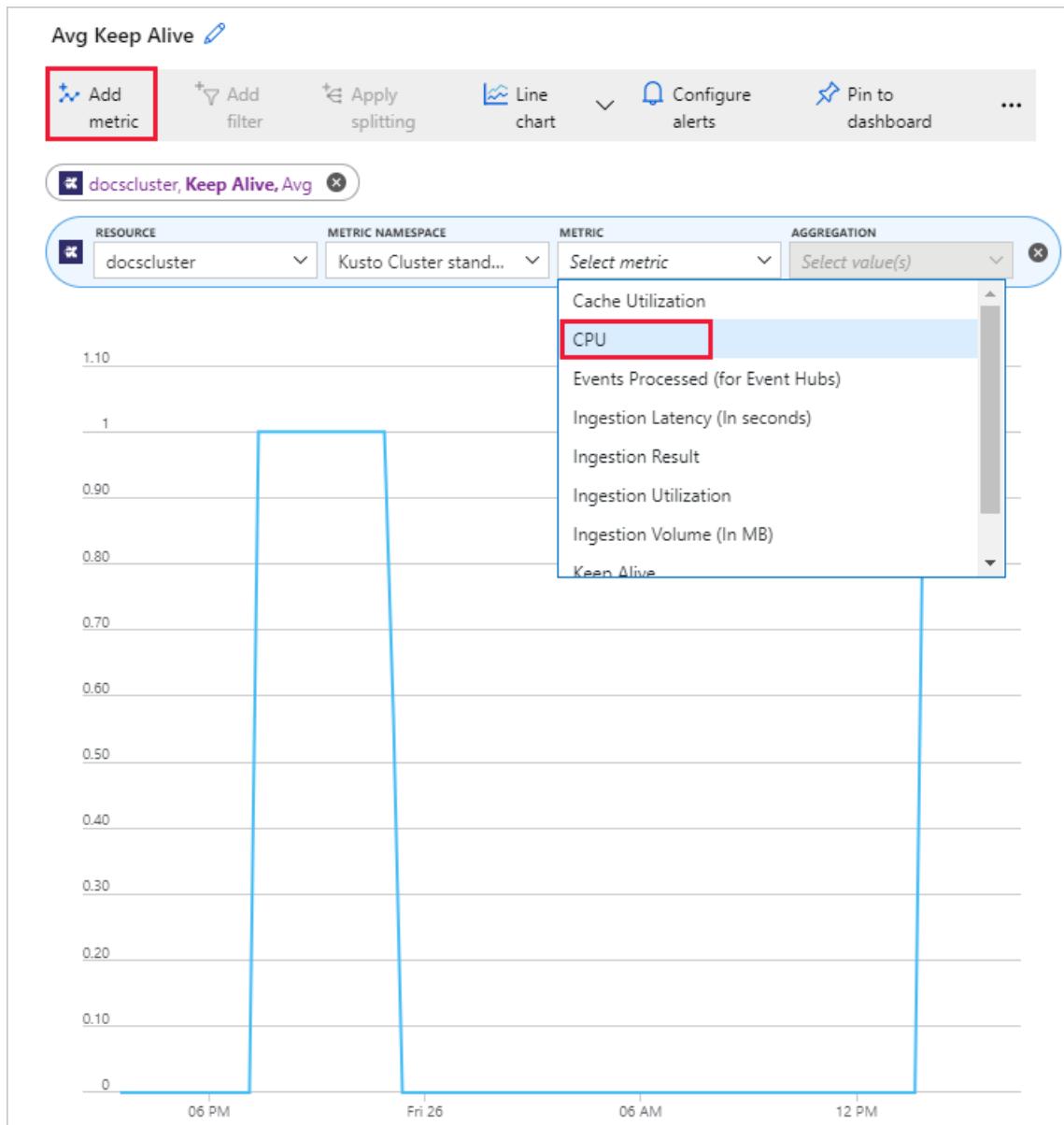
```
.show diagnostics  
| project IsHealthy
```

An output of 1 is healthy; an output of 0 is unhealthy.

3. Sign into the [Azure portal](#), and navigate to your cluster.
4. Under **Monitoring**, select **Metrics**, then select **Keep Alive**, as shown in the following image. An output close to 1 means a healthy cluster.



5. It's possible to add other metrics to the chart. Select the chart then **Add metric**. Select another metric - this example shows **CPU**.



6. If you need assistance diagnosing issues with the health of a cluster, please open a support request in the [Azure portal](#).

Use follower database to attach databases in Azure Data Explorer

12/2/2019 • 8 minutes to read • [Edit Online](#)

The **follower database** feature allows you to attach a database located in a different cluster to your Azure Data Explorer cluster. The **follower database** is attached in *read-only* mode, making it possible to view the data and run queries on the data that was ingested into the **leader database**. The follower database synchronizes changes in the leader databases. Due to the synchronization, there's a data lag of a few seconds to a few minutes in data availability. The length of the time lag depends on the overall size of the leader database metadata. The leader and follower databases use the same storage account to fetch the data. The storage is owned by the leader database. The follower database views the data without needing to ingest it. Since the attached database is a read-only database, the data, tables, and policies in the database can't be modified except for [caching policy](#), [principals](#), and [permissions](#). Attached databases can't be deleted. They must be detached by the leader or follower and only then they can be deleted.

Attaching a database to a different cluster using the follower capability is used as the infrastructure to share data between organizations and teams. The feature is useful to segregate compute resources to protect a production environment from non-production use cases. Follower can also be used to associate the cost of Azure Data Explorer cluster to the party that runs queries on the data.

Which databases are followed?

- A cluster can follow one database, several databases, or all databases of a leader cluster.
- A single cluster can follow databases from multiple leader clusters.
- A cluster can contain both follower databases and leader databases

Prerequisites

1. If you don't have an Azure subscription, [create a free account](#) before you begin.
2. [Create cluster and DB](#) for the leader and follower.
3. [Ingest data](#) to leader database using one of various methods discussed in [ingestion overview](#).

Attach a database

There are various methods you can use to attach a database. In this article, we discuss attaching a database using C# or an Azure Resource Manager template. To attach a database, you must have permissions on the leader cluster and the follower cluster. For more information about permissions, see [manage permissions](#).

Attach a database using C#

Needed NuGets

- Install [Microsoft.Azure.Management.kusto](#).
- Install [Microsoft.Rest.ClientRuntime.Azure.Authentication](#) for authentication.

Code Example

```

var tenantId = "xxxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxxx";//Directory (tenant) ID
var clientId = "xxxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxxx";//Application ID
var clientSecret = "xxxxxxxxxxxxxx"; //Client secret
var leaderSubscriptionId = "xxxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxxx";
var followerSubscriptionId = "xxxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxxx";

var serviceCreds = await ApplicationTokenProvider.LoginSilentAsync(tenantId, clientId, clientSecret);
var resourceManagementClient = new KustoManagementClient(serviceCreds){
    SubscriptionId = followerSubscriptionId
};

var followerResourceGroupName = "followerResouceGroup";
var leaderResourceGroup = "leaderResouceGroup";
var leaderClusterName = "leader";
var followerClusterName = "follower";
var attachedDatabaseConfigurationName = "adc";
var databaseName = "db"; // Can be specific database name or * for all databases
var defaultPrincipalsModificationKind = "Union";
var location = "North Central US";

AttachedDatabaseConfiguration attachedDatabaseConfigurationProperties = new AttachedDatabaseConfiguration()
{
    ClusterResourceId =
    $"{"/subscriptions/{leaderSubscriptionId}/resourceGroups/{leaderResourceGroup}/providers/Microsoft.Kusto/Clusters/{leaderClusterName}"},
    DatabaseName = databaseName,
    DefaultPrincipalsModificationKind = defaultPrincipalsModificationKind,
    Location = location
};

var attachedDatabaseConfigurations =
    resourceManagementClient.AttachedDatabaseConfigurations.CreateOrUpdate(followerResourceGroupName,
    followerClusterName, attachedDatabaseConfigurationName, attachedDatabaseConfigurationProperties);

```

Attach a database using Python

Needed Modules

```

pip install azure-common
pip install azure-mgmt-kusto

```

Code Example

```

from azure.mgmt.kusto import KustoManagementClient
from azure.mgmt.kusto.models import AttachedDatabaseConfiguration
from azure.common.credentials import ServicePrincipalCredentials
import datetime

#Directory (tenant) ID
tenant_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
#Application ID
client_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
#Client Secret
client_secret = "xxxxxxxxxxxxxx"
follower_subscription_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
leader_subscription_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
credentials = ServicePrincipalCredentials(
    client_id=client_id,
    secret=client_secret,
    tenant=tenant_id
)
kusto_management_client = KustoManagementClient(credentials, follower_subscription_id)

follower_resource_group_name = "followerResouceGroup"
leader_resouce_group_name = "leaderResouceGroup"
follower_cluster_name = "follower"
leader_cluster_name = "leader"
attached_database_Configuration_name = "adc"
database_name = "db" # Can be specific database name or * for all databases
default_principals_modification_kind = "Union"
location = "North Central US"
cluster_resource_id = "/subscriptions/" + leader_subscription_id + "/resourceGroups/" +
leader_resouce_group_name + "/providers/Microsoft.Kusto/Clusters/" + leader_cluster_name

attached_database_configuration_properties = AttachedDatabaseConfiguration(cluster_resource_id =
cluster_resource_id, database_name = database_name, default_principals_modification_kind =
default_principals_modification_kind, location = location)

#Returns an instance of LROPoller, see https://docs.microsoft.com/python/api/msrest/msrest.polling.lropoller?
view=azure-python
poller =
kusto_management_client.attached_database_configurations.create_or_update(follower_resource_group_name,
follower_cluster_name, attached_database_Configuration_name, attached_database_configuration_properties)

```

Attach a database using an Azure Resource Manager template

In this section, you learn how to attach a database by using an [Azure Resource Manager template](#).

```
{
"$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
"contentVersion": "1.0.0.0",
"parameters": {
  "followerClusterName": {
    "type": "string",
    "defaultValue": "",
    "metadata": {
      "description": "Name of the follower cluster."
    }
  },
  "attachedDatabaseConfigurationsName": {
    "type": "string",
    "defaultValue": "",
    "metadata": {
      "description": "Name of the attached database configurations to create."
    }
  },
  "databaseName": {
    "type": "string",
    "defaultValue": ""
  }
}
```

```

"metadata": {
    "description": "The name of the database to follow. You can follow all databases by using '*'."
},
"leaderClusterResourceId": {
    "type": "string",
    "defaultValue": "",
    "metadata": {
        "description": "Name of the leader cluster to create."
    }
},
"defaultPrincipalsModificationKind": {
    "type": "string",
    "defaultValue": "",
    "metadata": {
        "description": "The default principal modification kind."
    }
},
"location": {
    "type": "string",
    "defaultValue": "",
    "metadata": {
        "description": "Location for all resources."
    }
},
"variables": {},
"resources": [
{
    "name": "[parameters('followerClusterName')]",
    "type": "Microsoft.Kusto/clusters",
    "sku": {
        "name": "Standard_D13_v2",
        "tier": "Standard",
        "capacity": 2
    },
    "apiVersion": "2019-09-07",
    "location": "[parameters('location')]"
},
{
    "name": "[concat(parameters('followerClusterName'), '/',
parameters('attachedDatabaseConfigurationsName'))]",
    "type": "Microsoft.Kusto/clusters/attachedDatabaseConfigurations",
    "apiVersion": "2019-09-07",
    "location": "[parameters('location')]",
    "dependsOn": [
        "[resourceId('Microsoft.Kusto/clusters', parameters('followerClusterName'))]"
    ],
    "properties": {
        "databaseName": "[parameters('databaseName')]",
        "clusterResourceId": "[parameters('leaderClusterResourceId')]",
        "defaultPrincipalsModificationKind": "[parameters('defaultPrincipalsModificationKind')]"
    }
}
]
}

```

Deploy the template

You can deploy the Azure Resource Manager template by [using the Azure portal](#) or using powershell.

TEMPLATE



2 resources



Edit template



Edit paramet...



Learn more

BASICS

Subscription *

KUSTO_DEV_KUSTO_05



Resource group *

Select a resource group

[Create new](#)

Location *

(Europe) West Europe



SETTINGS

Follower Cluster Name ⓘ

Attached Database Configurations Name ⓘ

Database Name ⓘ

Leader Cluster Resource Id ⓘ

Default Principals Modification Kind ⓘ

Location ⓘ

SETTING	DESCRIPTION
Follower Cluster Name	The name of the follower cluster
Attached Database Configurations Name	The name of the attached database configurations object. The name must be unique at the cluster level.
Database Name	The name of the database to be followed. If you want to follow all the leader's databases, use '*'.
Leader Cluster Resource ID	The resource ID of the leader cluster.
Default Principals Modification Kind	The default principal modification kind. Can be <code>Union</code> , <code>Replace</code> or <code>None</code> . For more information about default principal modification kind, see principal modification kind control command .
Location	The location of all the resources. The leader and the follower must be in the same location.

Verify that the database was successfully attached

To verify that the database was successfully attached, find your attached databases in the [Azure portal](#).

1. Navigate to the follower cluster and select **Databases**
2. Search for new Read-only databases in the database list.

Home > follower4 - Databases

follower4 - Databases
Azure Data Explorer Cluster

Search (Ctrl+ /) + Add database Refresh Query Delete

Overview Activity log Access control (IAM) Tags Diagnose and solve problems Query Settings Scale up Scale out Configurations Security Properties Locks Export template Data Databases

DATABASE	SIZE	RETENTION PERIOD	CACHE PERIOD	DATABASE KIND	SHARED WITH OTHERS
db13	0 Bytes	3650 days	31 days	Read-Only	No
db1	0 Bytes	3650 days	31 days	Read-Only	No

Alternatively:

1. Navigate to the leader cluster and select **Databases**
2. Check that the relevant databases are marked as **SHARED WITH OTHERS > Yes**

Home > dorsi2 - Databases

dorsi2 - Databases
Azure Data Explorer Cluster

Search (Ctrl+ /) + Add database Refresh Query Delete

Overview Activity log Access control (IAM) Tags Diagnose and solve problems Query Settings Scale up Scale out Configurations Security Properties Locks Export template Data Databases

DATABASE	SIZE	RETENTION PERIOD	CACHE PERIOD	DATABASE KIND	SHARED WITH OTHERS
db1	0 Bytes	3650 days	31 days	Read-Write	Yes
db13	0 Bytes	3650 days	31 days	Read-Write	Yes

Detach the follower database using C#

Detach the attached follower database from the follower cluster

The follower cluster can detach any attached database as follows:

```

var tenantId = "xxxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxx";//Directory (tenant) ID
var clientId = "xxxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxx";//Application ID
var clientSecret = "xxxxxxxxxxxxxx";//Client secret
var leaderSubscriptionId = "xxxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxx";
var followerSubscriptionId = "xxxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxx";

var serviceCreds = await ApplicationTokenProvider.LoginSilentAsync(tenantId, clientId, clientSecret);
var resourceManagementClient = new KustoManagementClient(serviceCreds){
    SubscriptionId = followerSubscriptionId
};

var followerResourceGroupName = "testrg";
//The cluster and database that are created as part of the prerequisites
var followerClusterName = "follower";
var attachedDatabaseConfigurationsName = "adc";

resourceManagementClient.AttachedDatabaseConfigurations.Delete(followerResourceGroupName, followerClusterName,
attachedDatabaseConfigurationsName);

```

Detach the attached follower database from the leader cluster

The leader cluster can detach any attached database as follows:

```

var tenantId = "xxxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxx";//Directory (tenant) ID
var clientId = "xxxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxx";//Application ID
var clientSecret = "xxxxxxxxxxxxxx";//Client secret
var leaderSubscriptionId = "xxxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxx";
var followerSubscriptionId = "xxxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxx";

var serviceCreds = await ApplicationTokenProvider.LoginSilentAsync(tenantId, clientId, clientSecret);
var resourceManagementClient = new KustoManagementClient(serviceCreds){
    SubscriptionId = leaderSubscriptionId
};

var leaderResourceGroupName = "testrg";
var followerResourceGroupName = "followerResourceGroup";
var leaderClusterName = "leader";
var followerClusterName = "follower";
//The cluster and database that are created as part of the Prerequisites
var followerDatabaseDefinition = new FollowerDatabaseDefinition()
{
    AttachedDatabaseConfigurationName = "adc",
    ClusterResourceId =
    $"/subscriptions/{followerSubscriptionId}/resourceGroups/{followerResourceGroupName}/providers/Microsoft.Kusto/
Clusters/{followerClusterName}"
};

resourceManagementClient.Clusters.DetachFollowerDatabases(leaderResourceGroupName, leaderClusterName,
followerDatabaseDefinition);

```

Detach the follower database using Python

Detach the attached follower database from the follower cluster

The follower cluster can detach any attached database as follows:

```
from azure.mgmt.kusto import KustoManagementClient
from azure.common.credentials import ServicePrincipalCredentials
import datetime

#Directory (tenant) ID
tenant_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
#Application ID
client_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
#Client Secret
client_secret = "xxxxxxxxxxxxxx"
follower_subscription_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
credentials = ServicePrincipalCredentials(
    client_id=client_id,
    secret=client_secret,
    tenant=tenant_id
)
kusto_management_client = KustoManagementClient(credentials, follower_subscription_id)

follower_resource_group_name = "followerResourceGroup"
follower_cluster_name = "follower"
attached_database_configurationName = "adc"

#Returns an instance of LROPoller, see https://docs.microsoft.com/python/api/msrest/msrest.polling.lropoller?
view=azure-python
poller = kusto_management_client.attached_database_configurations.delete(follower_resource_group_name,
follower_cluster_name, attached_database_configurationName)
```

Detach the attached follower database from the leader cluster

The leader cluster can detach any attached database as follows:

```

from azure.mgmt.kusto import KustoManagementClient
from azure.mgmt.kusto.models import FollowerDatabaseDefinition
from azure.common.credentials import ServicePrincipalCredentials
import datetime

#Directory (tenant) ID
tenant_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
#Application ID
client_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
#Client Secret
client_secret = "xxxxxxxxxxxxxx"
follower_subscription_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
leader_subscription_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
credentials = ServicePrincipalCredentials(
    client_id=client_id,
    secret=client_secret,
    tenant=tenant_id
)
kusto_management_client = KustoManagementClient(credentials, follower_subscription_id)

follower_resource_group_name = "followerResourceGroup"
leader_resource_group_name = "leaderResourceGroup"
follower_cluster_name = "follower"
leader_cluster_name = "leader"
attached_database_configuration_name = "adc"
location = "North Central US"
cluster_resource_id = "/subscriptions/" + follower_subscription_id + "/resourceGroups/" +
follower_resource_group_name + "/providers/Microsoft.Kusto/Clusters/" + follower_cluster_name

#Returns an instance of LROPoller, see https://docs.microsoft.com/python/api/msrest/msrest.polling.lropoller?
view=azure-python
poller = kusto_management_client.clusters.detach_follower_databases(resource_group_name =
leader_resource_group_name, cluster_name = leader_cluster_name, cluster_resource_id = cluster_resource_id,
attached_database_configuration_name = attached_database_configuration_name)

```

Manage principals, permissions, and caching policy

Manage principals

When attaching a database, specify the **"default principals modification kind"**. The default is keeping the leader database collection of [authorized principals](#)

KIND	DESCRIPTION
Union	The attached database principals will always include the original database principals plus additional new principals added to the follower database.
Replace	No inheritance of principals from the original database. New principals must be created for the attached database.
None	The attached database principals include only the principals of the original database with no additional principals.

For more information about using control commands to configure the authorized principals, see [Control commands for managing a follower cluster](#).

Manage permissions

Managing read-only database permission is the same as for all database types. See [manage permissions in the](#)

[Azure portal](#).

Configure caching policy

The follower database administrator can modify the [caching policy](#) of the attached database or any of its tables on the hosting cluster. The default is keeping the leader database collection of database and table-level caching policies. You can, for example, have a 30 day caching policy on the leader database for running monthly reporting and a three day caching policy on the follower database to query only the recent data for troubleshooting. For more information about using control commands to configure the caching policy on the follower database or table, see [Control commands for managing a follower cluster](#).

Limitations

- The follower and the leader clusters must be in the same region.
- [Streaming ingestion](#) can't be used on a database that is being followed.
- You can't delete a database that is attached to a different cluster before detaching it.
- You can't delete a cluster that has a database attached to a different cluster before detaching it.
- You can't stop a cluster that has attached follower or leader database(s).

Next steps

- For information about follower cluster configuration, see [Control commands for managing a follower cluster](#).

Manage Azure Data Explorer database permissions

4/4/2019 • 2 minutes to read • [Edit Online](#)

Azure Data Explorer enables you to control access to databases and tables, using a *role-based access control* model. Under this model, *principals* (users, groups, and apps) are mapped to *roles*. Principals can access resources according to the roles they're assigned.

This article describes the available roles and how to assign principals to those roles using the Azure portal and Azure Data Explorer management commands.

Roles and permissions

Azure Data Explorer has the following roles:

ROLE	PERMISSIONS
Database admin	Can do anything in the scope of a particular database.
Database user	Can read all data and metadata in the database. Additionally, they can create tables (becoming the table admin for that table) and functions in the database.
Database viewer	Can read all data and metadata in the database.
Database ingestor	Can ingest data to all existing tables in the database, but not query the data.
Database monitor	Can execute '.show ...' commands in the context of the database and its child entities.
Table admin	Can do anything in the scope of a particular table.
Table ingestor	Can ingest data in the scope of a particular table, but not query the data.

Manage permissions in the Azure portal

1. Sign in to the [Azure portal](#).
2. Navigate to your Azure Data Explorer cluster.
3. In the **Overview** section, select the database where you want to manage permissions.

docscluster
Azure Data Explorer Cluster

Search (Ctrl+ /) <> Add Database Stop Refresh

Resource group (change)
docscluster

Location
West US

Subscription (change)
<SubscriptionName>

Subscription ID
<SubscriptionID>

Databases from A-Z

DATABASE

TestDatabase

4. Select **Permissions** then **Add**.

TestDatabase - Permissions

Search (Ctrl+ /) <> Add Refresh Remove

Overview

Permissions

Query

Role 6 selected

2 items (1 Admin, 1 User)

NAME

5. Under **Add database permissions**, select the role that you want to assign the principal to, then **Select principals**.

Add Database Permissions X

Select a role i
 ▼

Select principals i >

Save Discard

6. Look up the principal, select it, then **Select**.

New Principals □ X

+ Invite

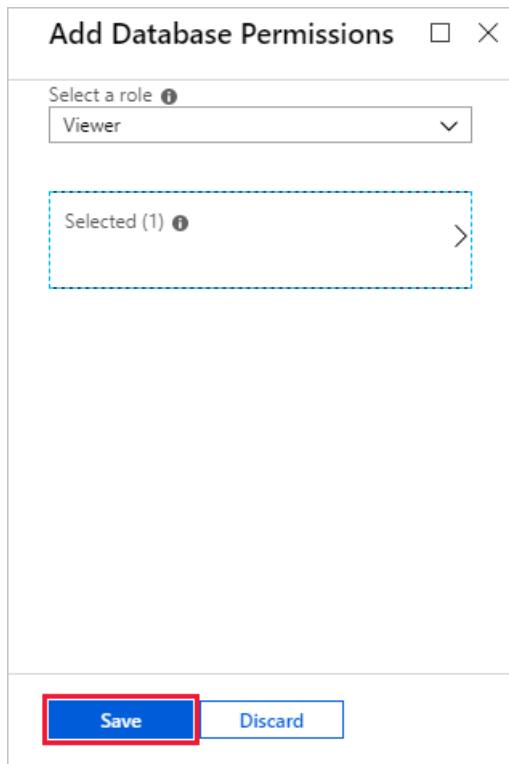
Select i
 ✓

 Maggie

Selected
Maggie >

Select

7. Select **Save**.



Manage permissions with management commands

1. Sign-in to <https://dataexplorer.azure.com>, and add your cluster if it's not already available.

2. In the left pane, select the appropriate database.

3. Use the `.add` command to assign principals to roles:

```
.add database databasename rolename ('aaduser | aadgroup=user@domain.com') . To add a user to the Database user role, run the following command, substituting your database name and user.
```

```
.add database <TestDatabase> users ('aaduser=<user@contoso.com>')
```

The output of the command shows the list of existing users and the roles they're assigned to in the database.

Next steps

[Write queries](#)

Create database and table policies for Azure Data Explorer by using C#

12/2/2019 • 3 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. In this article, you'll create database and table policies for Azure Data Explorer by using C#.

Prerequisites

- Visual Studio 2019. If you don't have Visual Studio 2019, you can download and use the [free Visual Studio Community 2019](#). Be sure to select **Azure development** during the Visual Studio setup.
- An Azure subscription. If you need to, you can create a [free Azure account](#) before you start.
- [A test cluster and database](#).
- [A test table](#).

Install C# NuGet

- Install the [Azure Data Explorer \(Kusto\) NuGet package](#).
- Install the [Microsoft.Azure.Kusto.Data.NETStandard NuGet package](#). (Optional, for changing table policies.)
- Install the [Microsoft.IdentityModel.Clients.ActiveDirectory NuGet package](#), for authentication.

Authentication

To run the examples in this article, you need an Azure Active Directory (Azure AD) application and service principal that can access resources. You can use the same Azure AD application for authentication from [a test cluster and database](#). If you want to use a different Azure AD application, see [create an Azure AD application](#) to create a free Azure AD application and add role assignment at the subscription scope. This article also shows how to get the `Directory (tenant) ID`, `Application ID`, and `Client secret`. You might need to add the new Azure AD application as a principal in the database. For more information, see [Manage Azure Data Explorer database permissions](#).

Alter database retention policy

Sets a retention policy with a 10-day soft-delete period.

```

var tenantId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx";//Directory (tenant) ID
var clientId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx";//Application ID
var clientSecret = "xxxxxxxxxxxxxx"; //Client secret
var subscriptionId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx";
var authenticationContext = new AuthenticationContext($"https://login.windows.net/{tenantId}");
var credential = new ClientCredential(clientId, clientSecret);
var result = await authenticationContext.AcquireTokenAsync(resource: "https://management.core.windows.net/",
clientCredential: credential);

var credentials = new TokenCredentials(result.AccessToken, result.AccessTokenType);

var kustoManagementClient = new KustoManagementClient(credentials)
{
    SubscriptionId = subscriptionId
};

var resourceGroupName = "testrg";
//The cluster and database that are created as part of the prerequisites
var clusterName = "mykustocluster";
var databaseName = "mykustodatabase";
await kustoManagementClient.Databases.UpdateAsync(resourceGroupName, clusterName, databaseName, new
DatabaseUpdate(softDeletePeriod: TimeSpan.FromDays(10)));

```

Alter database cache policy

Sets a cache policy for the database. The previous five days of data will be on the cluster SSD.

```

var tenantId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx";//Directory (tenant) ID
var clientId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx";//Application ID
var clientSecret = "xxxxxxxxxxxxxx"; //Client secret
var subscriptionId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx";
var authenticationContext = new AuthenticationContext($"https://login.windows.net/{tenantId}");
var credential = new ClientCredential(clientId, clientSecret);
var result = await authenticationContext.AcquireTokenAsync(resource: "https://management.core.windows.net/",
clientCredential: credential);

var credentials = new TokenCredentials(result.AccessToken, result.AccessTokenType);

var kustoManagementClient = new KustoManagementClient(credentials)
{
    SubscriptionId = subscriptionId
};

var resourceGroupName = "testrg";
//The cluster and database that are created as part of the prerequisites
var clusterName = "mykustocluster";
var databaseName = "mykustodatabase";
await kustoManagementClient.Databases.UpdateAsync(resourceGroupName, clusterName, databaseName, new
DatabaseUpdate(hotCachePeriod: TimeSpan.FromDays(5)));

```

Alter table cache policy

Sets a cache policy for the table. The previous five days of data will be on the cluster SSD.

```

var kustoUri = "https://<ClusterName>.<Region>.kusto.windows.net:443/";
var databaseName = "<DatabaseName>";
var tenantId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx";//Directory (tenant) ID
var clientId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx";//Application ID
var clientSecret = "xxxxxxxxxxxxxx";//Client secret
var tableName = "<TableName>"

var kustoConnectionStringBuilder =
    new KustoConnectionStringBuilder(kustoUri)
{
    FederatedSecurity = true,
    InitialCatalog = databaseName,
    ApplicationClientId = clientId,
    ApplicationKey = clientSecret,
    Authority = tenantId
};

using (var kustoClient = KustoClientFactory.CreateCslAdminProvider(kustoConnectionStringBuilder))
{
    //dataHotSpan and indexHotSpan should have the same value
    var hotSpan = TimeSpan.FromDays(5);
    var command1 = CslCommandGenerator.GenerateAlterTableCachingPolicyCommand(tableName,
        dataHotSpan: hotSpan, indexHotSpan: hotSpan);

    kustoClient.ExecuteControlCommand(command);
}

```

Add a new principal for the database

Adds a new Azure AD application as admin principal for the database.

```

var tenantId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx";//Directory (tenant) ID
var clientId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx";//Application ID
var clientSecret = "xxxxxxxxxxxxxx";//Client Secret
var clientIdToAdd = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx";//Application ID
var subscriptionId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx";
var authenticationContext = new AuthenticationContext($"https://login.windows.net/{tenantId}");
var credential = new ClientCredential(clientId, clientSecret);
var result = await authenticationContext.AcquireTokenAsync(resource: "https://management.core.windows.net/",
clientCredential: credential);

var credentials = new TokenCredentials(result.AccessToken, result.AccessTokenType);

var kustoManagementClient = new KustoManagementClient(credentials)
{
    SubscriptionId = subscriptionId
};

var resourceGroupName = "testrg";
//The cluster and database that are created as part of the prerequisites
var clusterName = "mykustocluster";
var databaseName = "mykustodatabase";
await kustoManagementClient.Databases.AddPrincipalsAsync(resourceGroupName, clusterName, databaseName,
    new DatabasePrincipalListRequest()
    {
        Value = new List<DatabasePrincipal>()
        {
            new DatabasePrincipal("Admin", "<database_principle_name>", "App", appId:
clientIdToAdd, tenantName:tenantId)
        }
    });

```

Next steps

- [Read more about database and table policies](#)

Create database and table policies for Azure Data Explorer by using Python

12/2/2019 • 2 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. In this article, you create database and table policies for Azure Data Explorer using Python.

Prerequisites

- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.
- [A test cluster and database](#)
- [A test table](#)

Install the data libraries

```
pip install azure-common
pip install azure-mgmt-kusto
pip install azure-kusto-data (Optional, for changing table's policies)
```

Authentication

For running the examples in this article, we need an Azure AD Application and service principal that can access resources. You may use the same Azure AD Application for authentication from [a test cluster and database](#). If you want to use a different Azure AD Application, see [create an Azure AD application](#) to create a free Azure AD Application and add role assignment at the subscription scope. It also shows how to get the `Directory (tenant) ID`, `Application ID`, and `Client Secret`. You may need to add the new Azure AD Application as a principal in the database, see [Manage Azure Data Explorer database permissions](#).

Alter database retention policy

Sets a retention policy with a 10 day soft-delete period.

```

from azure.mgmt.kusto import KustoManagementClient
from azure.mgmt.kusto.models import DatabaseUpdate
from azure.common.credentials import ServicePrincipalCredentials
import datetime

#Directory (tenant) ID
tenant_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
#Application ID
client_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
#Client Secret
client_secret = "xxxxxxxxxxxxxx"
subscription_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
credentials = ServicePrincipalCredentials(
    client_id=client_id,
    secret=client_secret,
    tenant=tenant_id
)
kusto_management_client = KustoManagementClient(credentials, subscription_id)

resource_group_name = "testrg";
#The cluster and database that are created as part of the Prerequisites
cluster_name = "mykustocluster";
database_name = "mykustodatabase";

#Returns an instance of LROPoller, see https://docs.microsoft.com/python/api/msrest/msrest.polling.lropoller?
view=azure-python
poller = kustoManagementClient.databases.update(resource_group_name=resource_group_name,
cluster_name=cluster_name, database_name=database_name,
parameters=DatabaseUpdate(soft_delete_period=datetime.timedelta(days=10)))

```

Alter database cache policy

Sets a cache policy for the database that the last five days of data will be on the cluster SSD.

```

from azure.mgmt.kusto import KustoManagementClient
from azure.mgmt.kusto.models import DatabaseUpdate
from azure.common.credentials import ServicePrincipalCredentials
import datetime

#Directory (tenant) ID
tenant_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
#Application ID
client_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
#Client Secret
client_secret = "xxxxxxxxxxxxxx"
subscription_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
credentials = ServicePrincipalCredentials(
    client_id=client_id,
    secret=client_secret,
    tenant=tenant_id
)
kusto_management_client = KustoManagementClient(credentials, subscription_id)

resource_group_name = "testrg";
#The cluster and database that are created as part of the Prerequisites
cluster_name = "mykustocluster";
database_name = "mykustodatabase";

#Returns an instance of LROPoller, see https://docs.microsoft.com/python/api/msrest/msrest.polling.lropoller?
view=azure-python
poller = kustoManagementClient.databases.update(resource_group_name=resource_group_name,
cluster_name=cluster_name, database_name=database_name,
parameters=DatabaseUpdate(hot_cache_period=datetime.timedelta(days=5)))

```

Alter table cache policy

Sets a cache policy for the table that the last five days of data will be on the cluster SSD.

```

from azure.kusto.data.request import KustoClient, KustoConnectionStringBuilder
kusto_uri = "https://<ClusterName>.<Region>.kusto.windows.net"
database_name = "<DatabaseName>"
#Directory (tenant) ID
tenant_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
#Application ID
client_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
#Client Secret
client_secret = "xxxxxxxxxxxxxx"
#Application ID
client_id_to_add = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx";

kusto_connection_string_builder =
KustoConnectionStringBuilder.with_aad_application_key_authentication(connection_string=kusto_uri,
aad_app_id=client_id, app_key=client_secret, authority_id=tenant_id)

#The table that is created as part of the Prerequisites
table_name = "<TableName>"
caching_policy = r'hot = 5d'
command = '.alter table {} policy caching '.format(table_name) + caching_policy
kusto_client.execute_mgmt(database_name, command)

```

Add a new principal for database

Add a new Azure AD application as admin principal for the database

```

from azure.mgmt.kusto import KustoManagementClient
from azure.mgmt.kusto.models import DatabasePrincipal
from azure.common.credentials import ServicePrincipalCredentials

#Directory (tenant) ID
tenant_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
#Application ID
client_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
#Client Secret
client_secret = "xxxxxxxxxxxxxx"
#Application ID
client_id_to_add = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx";
subscription_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
credentials = ServicePrincipalCredentials(
    client_id=client_id,
    secret=client_secret,
    tenant=tenant_id
)
kusto_management_client = KustoManagementClient(credentials, subscription_id)

resource_group_name = "testrg";
#The cluster and database that are created as part of the Prerequisites
cluster_name = "mykustocluster";
database_name = "mykustodatabase";
role = "Admin"
principle_name = "<database_principle_name>";
type_name = "App"
kustoManagementClient.databases.add_principals(resource_group_name=resource_group_name,
cluster_name=cluster_name, database_name=database_name,
value=[DatabasePrincipal(role=role, name=principle_name, type=type_name,
app_id=client_id_to_add, tenant_name=tenant_id)])

```

Next steps

- [Read more about database and table policies](#)

Delete data from Azure Data Explorer

4/4/2019 • 2 minutes to read • [Edit Online](#)

Azure Data Explorer supports several bulk delete approaches, which we cover in this article. It doesn't support per-record deletion in real time, because it's optimized for fast read access.

- If one or more tables is no longer needed, delete them using the drop table or drop tables command.

```
.drop table <TableName>  
  
.drop tables (<TableName1>, <TableName2>,...)
```

- If old data is no longer needed, delete it by changing the retention period at the database or table level.

Consider a database or table that is set for 90 days of retention. Business needs change, so now only 60 days of data is needed. In this case, delete the older data in one of the following ways.

```
.alter-merge database <DatabaseName> policy retention softdelete = 60d  
  
.alter-merge table <TableName> policy retention softdelete = 60d
```

For more information, see [Retention policy](#).

If you need assistance with data deletion issues, please open a support request in the [Azure portal](#).

Handle duplicate data in Azure Data Explorer

7/29/2019 • 4 minutes to read • [Edit Online](#)

Devices sending data to the Cloud maintain a local cache of the data. Depending on the data size, the local cache could be storing data for days or even months. You want to safeguard your analytical databases from malfunctioning devices that resend the cached data and cause data duplication in the analytical database. This topic outlines best practices for handling duplicate data for these types of scenarios.

The best solution for data duplication is preventing the duplication. If possible, fix the issue earlier in the data pipeline, which saves costs associated with data movement along the data pipeline and avoids spending resources on coping with duplicate data ingested into the system. However, in situations where the source system can't be modified, there are various ways to deal with this scenario.

Understand the impact of duplicate data

Monitor the percentage of duplicate data. Once the percentage of duplicate data is discovered, you can analyze the scope of the issue and business impact and choose the appropriate solution.

Sample query to identify the percentage of duplicate records:

```
let _sample = 0.01; // 1% sampling
let _data =
DeviceEventsAll
| where EventDateTime between (datetime('10-01-2018 10:00') .. datetime('10-10-2018 10:00'));
let _totalRecords = toscalar(_data | count);
_data
| where rand()<= _sample
| summarize recordsCount=count() by hash(DeviceId) + hash(EventId) + hash(StationId) // Use all dimensions
that make row unique. Combining hashes can be improved
| summarize duplicateRecords=countif(recordsCount > 1)
| extend duplicate_percentage = (duplicateRecords / _sample) / _totalRecords
```

Solutions for handling duplicate data

Solution #1: Don't remove duplicate data

Understand your business requirements and tolerance of duplicate data. Some datasets can manage with a certain percentage of duplicate data. If the duplicated data doesn't have major impact, you can ignore its presence. The advantage of not removing the duplicate data is no additional overhead on the ingestion process or query performance.

Solution #2: Handle duplicate rows during query

Another option is to filter out the duplicate rows in the data during query. The `arg_max()` aggregated function can be used to filter out the duplicate records and return the last record based on the timestamp (or another column). The advantage of using this method is faster ingestion since de-duplication occurs during query time. In addition, all records (including duplicates) are available for auditing and troubleshooting. The disadvantage of using the `arg_max` function is the additional query time and load on the CPU every time the data is queried. Depending on the amount of the data being queried, this solution may become non-functional or memory-consuming and will require switching to other options.

In the following example, we query the last record ingested for a set of columns that determine the unique records:

```
DeviceEventsAll  
| where EventDateTime > ago(90d)  
| summarize hint.strategy=shuffle arg_max(EventDateTime, *) by DeviceId, EventId, StationId
```

This query can also be placed inside a function instead of directly querying the table:

```
.create function DeviceEventsView  
{  
DeviceEventsAll  
| where EventDateTime > ago(90d)  
| summarize arg_max(EventDateTime, *) by DeviceId, EventId, StationId  
}
```

Solution #3: Filter duplicates during the ingestion process

Another solution is to filter duplicates during the ingestion process. The system ignores the duplicate data during ingestion into Kusto tables. Data is ingested into a staging table and copied into another table after removing duplicate rows. The advantage of this solution is that query performance improves dramatically as compared to the previous solution. The disadvantages include increased ingestion time and additional data storage costs.

Additionaly, this solution works only if duplications aren't ingested concurrently. If there are multiple concurrent ingestions containing duplicate records, all may be ingested since the deduplication process will not find any existing matching records in the table.

The following example depicts this method:

1. Create another table with the same schema:

```
.create table DeviceEventsUnique (EventDateTime: datetime, DeviceId: int, EventId: int, StationId: int)
```

2. Create a function to filter out the duplicate records by anti-joining the new records with the previously ingested ones.

```
.create function RemoveDuplicateDeviceEvents()  
{  
DeviceEventsAll  
| join hint.strategy=broadcast kind = anti  
    (  
        DeviceEventsUnique  
        | where EventDateTime > ago(7d) // filter the data for certain time frame  
        | limit 1000000 //set some limitations (few million records) to avoid choking-up the system during outage recovery  
    ) on DeviceId, EventId, StationId  
}
```

NOTE

Joins are CPU-bound operations and add an additional load on the system.

3. Set [Update Policy](#) on `DeviceEventsUnique` table. The update policy is activated when new data goes into the `DeviceEventsAll` table. The Kusto engine will automatically execute the function as new [extents](#) are created. The processing is scoped to the newly created data. The following command stitches the source table (`DeviceEventsAll`), destination table (`DeviceEventsUnique`), and the function `RemoveDuplicatesDeviceEvents` together to create the update policy.

```
.alter table DeviceEventsUnique policy update  
@'[{\"IsEnabled\": true, \"Source\": \"DeviceEventsAll\", \"Query\": \"RemoveDuplicateDeviceEvents()\","  
\"IsTransactional\": true, \"PropagateIngestionProperties\": true}]'
```

NOTE

Update policy extends the duration of ingestion since the data is filtered during ingestion and then ingested twice (to the `DeviceEventsAll` table and to the `DeviceEventsUnique` table).

4. (Optional) Set a lower data retention on the `DeviceEventsAll` table to avoid storing copies of the data.

Choose the number of days depending on the data volume and the length of time you want to retain data for troubleshooting. You can set it to `0d` days retention to save COGS and improve performance, since the data isn't uploaded to storage.

```
.alter-merge table DeviceEventsAll policy retention softdelete = 1d
```

Summary

Data duplication can be handled in multiple ways. Evaluate the options carefully, taking into account price and performance, to determine the correct method for your business.

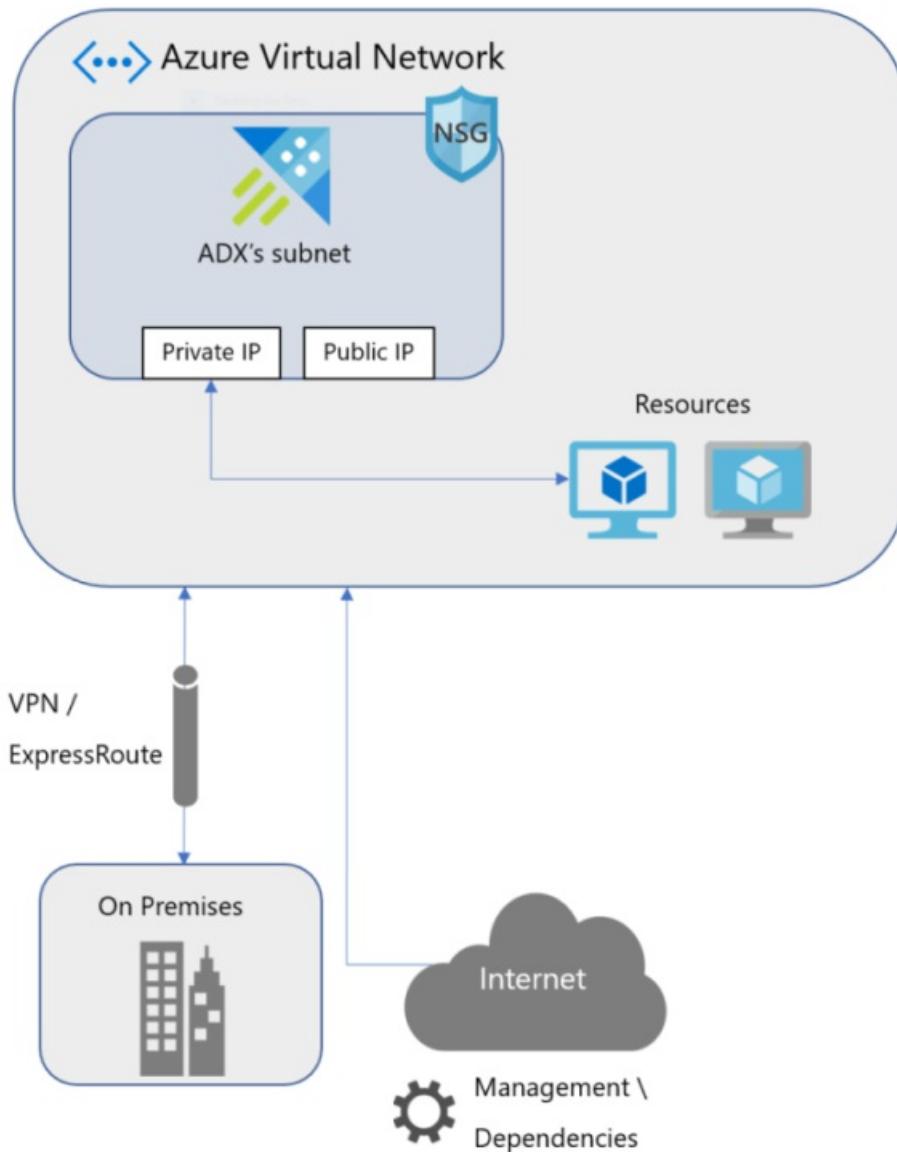
Next steps

[Write queries for Azure Data Explorer](#)

Deploy Azure Data Explorer into your Virtual Network (Preview)

12/4/2019 • 6 minutes to read • [Edit Online](#)

This article explains the resources that are present when you deploy an Azure Data Explorer cluster into a custom Azure Virtual Network. This information will help you deploy a cluster into a subnet in your Virtual Network (VNet). For more information on Azure Virtual Networks, see [What is Azure Virtual Network?](#)



Azure Data Explorer supports deploying a cluster into a subnet in your Virtual Network (VNet). This capability enables you to:

- Enforce [Network Security Group](#) (NSG) rules on your Azure Data Explorer cluster traffic.
- Connect your on-premises network to Azure Data Explorer cluster's subnet.
- Secure your data connection sources ([Event Hub](#) and [Event Grid](#)) with [service endpoints](#).

NOTE

The Virtual Network integration and deployment is in preview mode. To enable this feature, open a [support ticket](#).

Access your Azure Data Explorer cluster in your VNet

You can access your Azure Data Explorer cluster using the following IP addresses for each service (engine and data management services):

- **Private IP:** Used for accessing the cluster inside the VNet.
- **Public IP:** Used for accessing the cluster from outside the VNet for management and monitoring, and as a source address for outbound connections started from the cluster.

The following DNS records are created to access the service:

- `[clusternode].[geo-region].kusto.windows.net` (engine)
`ingest-[clusternode].[geo-region].kusto.windows.net` (data management) are mapped to the public IP for each service.
- `private-[clusternode].[geo-region].kusto.windows.net` (engine)
`private-ingest-[clusternode].[geo-region].kusto.windows.net` (data management) are mapped to the private IP for each service.

Plan subnet size in your VNet

The size of the subnet used to host an Azure Data Explorer cluster can't be altered after the subnet is deployed. In your VNet, Azure Data Explorer uses one private IP address for each VM and two private IP addresses for the internal load balancers (engine and data management). Azure networking also uses five IP addresses for each subnet. Azure Data Explorer provisions two VMs for the data management service. Engine service VMs are provisioned per user configuration scale capacity.

The total number of IP addresses:

USE	NUMBER OF ADDRESSES
Engine service	1 per instance
Data management service	2
Internal load balancers	2
Azure reserved addresses	5
Total	#engine_instances + 9

IMPORTANT

Subnet size must be planned in advance since it can't be changed after Azure Data Explorer is deployed. Therefore, reserve needed subnet size accordingly.

Service endpoints for connecting to Azure Data Explorer

[Azure Service Endpoints](#) enables you to secure your Azure multi-tenant resources to your virtual network.

Deploying Azure Data Explorer cluster into your subnet allows you to setup data connections with [Event Hub](#) or [Event Grid](#) while restricting the underlying resources for Azure Data Explorer subnet.

NOTE

When using EventGrid setup with [Storage](#) and [Event Hub], the storage account used in the subscription can be locked with service endpoints to Azure Data Explorer's subnet while allowing trusted Azure platform services in the [firewall configuration](#), but the Event Hub can't enable Service Endpoint since it doesn't support trusted [Azure platform services](#).

Dependencies for VNet deployment

Network Security Groups configuration

[Network Security Groups \(NSG\)](#) provide the ability to control network access within a VNet. Azure Data Explorer can be accessed using two endpoints: HTTPs (443) and TDS (1433). The following NSG rules must be configured to allow access to these endpoints for management, monitoring, and proper operation of your cluster.

Inbound NSG configuration

USE	FROM	TO	PROTOCOL
Management	ADX management addresses /AzureDataExplore rManagement(ServiceTag)	ADX subnet:443	TCP
Health monitoring	ADX health monitoring addresses	ADX subnet:443	TCP
ADX internal communication	ADX subnet: All ports	ADX subnet:All ports	All
Allow Azure load balancer inbound (health probe)	AzureLoadBalancer	ADX subnet:80,443	TCP

Outbound NSG configuration

USE	FROM	TO	PROTOCOL
Dependency on Azure Storage	ADX subnet	Storage:443	TCP
Dependency on Azure Data Lake	ADX subnet	AzureDataLake:443	TCP
EventHub ingestion and service monitoring	ADX subnet	EventHub:443,5671	TCP
Publish Metrics	ADX subnet	AzureMonitor:443	TCP
Azure Monitor configuration download	ADX subnet	Azure Monitor configuration endpoint addresses :443	TCP
Active Directory (if applicable)	ADX subnet	AzureActiveDirectory:443	TCP
Certificate authority	ADX subnet	Internet:80	TCP
Internal communication	ADX subnet	ADX Subnet:All Ports	All

USE	FROM	TO	PROTOCOL
Ports that are used for sql_request and http_request plugins	ADX subnet	Internet:Custom	TCP

Relevant IP addresses

Azure Data Explorer management IP addresses

REGION	ADDRESSES
Australia Central	20.37.26.134
Australia Central2	20.39.99.177
Australia East	40.82.217.84
Australia Southeast	20.40.161.39
BrazilSouth	191.233.25.183
Canada Central	40.82.188.208
Canada East	40.80.255.12
Central India	40.81.249.251
Central US	40.67.188.68
Central US EUAP	40.89.56.69
East Asia	20.189.74.103
East US	52.224.146.56
East US2	52.232.230.201
East US2 EUAP	52.253.226.110
France Central	40.66.57.91
France South	40.82.236.24
Japan East	20.43.89.90
Japan West	40.81.184.86
Korea Central	40.82.156.149
Korea South	40.80.234.9
North Central US	40.81.45.254

REGION	ADDRESSES
North Europe	52.142.91.221
South Africa North	102.133.129.138
South Africa West	102.133.0.97
South Central US	20.45.3.60
Southeast Asia	40.119.203.252
South India	40.81.72.110
UK South	40.81.154.254
UK West	40.81.122.39
West Central US	52.159.55.120
West Europe	51.145.176.215
West India	40.81.88.112
West US	13.64.38.225
West US2	40.90.219.23

Health monitoring addresses

REGION	ADDRESSES
Australia Central	191.239.64.128
Australia Central 2	191.239.64.128
Australia East	191.239.64.128
Australia Southeast	191.239.160.47
Brazil South	23.98.145.105
Canada Central	168.61.212.201
Canada East	168.61.212.201
Central India	23.99.5.162
Central US	168.61.212.201
Central US EUAP	168.61.212.201
East Asia	168.63.212.33

REGION	ADDRESSES
East US	137.116.81.189
East US 2	137.116.81.189
East US 2 EUAP	137.116.81.189
France Central	23.97.212.5
France South	23.97.212.5
Japan East	138.91.19.129
Japan West	138.91.19.129
Korea Central	138.91.19.129
Korea South	138.91.19.129
North Central US	23.96.212.108
North Europe	191.235.212.69
South Africa North	104.211.224.189
South Africa West	104.211.224.189
South Central US	23.98.145.105
South India	23.99.5.162
Southeast Asia	168.63.173.234
UK South	23.97.212.5
UK West	23.97.212.5
West Central US	168.61.212.201
West Europe	23.97.212.5
West India	23.99.5.162
West US	23.99.5.162
West US 2	23.99.5.162

Azure Monitor configuration endpoint addresses

REGION	ADDRESSES
Australia Central	52.148.86.165

REGION	ADDRESSES
Australia Central 2	52.148.86.165
Australia East	52.148.86.165
Australia Southeast	52.148.86.165
Brazil South	13.68.89.19
Canada Central	13.90.43.231
Canada East	13.90.43.231
Central India	13.71.25.187
Central US	52.173.95.68
Central US EUAP	13.90.43.231
East Asia	13.75.117.221
East US	13.90.43.231
East US 2	13.68.89.19
East US 2 EUAP	13.68.89.19
France Central	52.174.4.112
France South	52.174.4.112
Japan East	13.75.117.221
Japan West	13.75.117.221
Korea Central	13.75.117.221
Korea South	13.75.117.221
North Central US	52.162.240.236
North Europe	52.169.237.246
South Africa North	13.71.25.187
South Africa West	13.71.25.187
South Central US	13.84.173.99
South India	13.71.25.187

REGION	ADDRESSES
Southeast Asia	52.148.86.165
UK South	52.174.4.112
UK West	52.169.237.246
West Central US	52.161.31.69
West Europe	52.174.4.112
West India	13.71.25.187
West US	40.78.70.148
West US 2	52.151.20.103

ExpressRoute setup

Use ExpressRoute to connect on premises network to the Azure Virtual Network. A common setup is to advertise the default route (0.0.0.0/0) through the Border Gateway Protocol (BGP) session. This forces traffic coming out of the Virtual Network to be forwarded to the customer's premise network that may drop the traffic, causing outbound flows to break. To overcome this default, [User Defined Route \(UDR\)](#) (0.0.0.0/0) can be configured and next hop will be *Internet*. Since the UDR takes precedence over BGP, the traffic will be destined to the Internet.

Securing outbound traffic with firewall

If you want to secure outbound traffic using [Azure Firewall](#) or any virtual appliance to limit domain names, the following Fully Qualified Domain Names (FQDN) must be allowed in the firewall.

```
prod.warmpath.msftcloudes.com:443
production.diagnostics.monitoring.core.windows.net:443
graph.windows.net:443
*.update.microsoft.com:443
shavamanifestcdnprod1.azureedge.net:443
login.live.com:443
wdcp.microsoft.com:443
login.microsoftonline.com:443
azureprofilerfrontdoor.cloudapp.net:443
*.core.windows.net:443
*.servicebus.windows.net:443
shoebox2.metrics.nsatc.net:443
prod-dsts.dsts.core.windows.net:443
ocsp.msocsp.com:80
*.windowsupdate.com:80
ocsp.digicert.com:80
go.microsoft.com:80
dmd.metaservices.microsoft.com:80
www.msftconnecttest.com:80
crl.microsoft.com:80
www.microsoft.com:80
adl.windows.com:80
crl3.digicert.com:80
```

You also need to define the [route table](#) on the subnet with the [management addresses](#) and [health monitoring addresses](#) with next hop *Internet* to prevent asymmetric routes issues.

For example, for **West US** region, the following UDRs must be defined:

NAME	ADDRESS PREFIX	NEXT HOP
ADX_Management	13.64.38.225/32	Internet
ADX_Monitoring	23.99.5.162/32	Internet

Deploy Azure Data Explorer cluster into your VNet using an Azure Resource Manager template

To deploy Azure Data Explorer cluster into your virtual network, use the [Deploy Azure Data Explorer cluster into your VNet](#) Azure Resource Manager template.

This template creates the cluster, virtual network, subnet, network security group, and public IP addresses.

Monitor Azure Data Explorer ingestion operations using diagnostic logs (Preview)

11/18/2019 • 3 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast, fully managed data analytics service for real-time analysis on large volumes of data streaming from applications, websites, IoT devices, and more. To use Azure Data Explorer, you first create a cluster, and create one or more databases in that cluster. Then you ingest (load) data into a table in a database so that you can run queries against it. [Azure Monitor diagnostic logs](#) provide data about the operation of Azure resources. Azure Data Explorer uses diagnostic logs for insights on ingestion successes and failures. You can export operation logs to Azure Storage, Event Hub, or Log Analytics to monitor ingestion status. Logs from Azure Storage and Azure Event Hub can be routed to a table in your Azure Data Explorer cluster for further analysis.

Prerequisites

- If you don't have an Azure subscription, create a [free Azure account](#).
- Create a [cluster and database](#).

Sign in to the Azure portal

Sign in to the [Azure portal](#).

Set up diagnostic logs for an Azure Data Explorer cluster

Diagnostic logs can be used to configure the collection of the following log data:

- Successful ingestion operations: These logs have information about successfully completed ingestion operations.
- Failed ingestion operations: These logs have detailed information about failed ingestion operations including error details.

The data is then archived into a Storage account, streamed to an Event Hub, or sent to Log Analytics, as per your specifications.

Enable diagnostic logs

Diagnostic logs are disabled by default. To enable diagnostic logs, do the following steps:

1. In the [Azure portal](#), select the Azure Data Explorer cluster resource that you want to monitor.
2. Under **Monitoring**, select **Diagnostic settings**.

The screenshot shows the Azure portal interface for managing diagnostic settings. On the left, there's a navigation pane with sections like Overview, Activity log, Access control (IAM), Diagnose and solve problems, Query, Settings, Data, Monitoring, Alerts, Metrics, and Diagnostic settings. The 'Monitoring' section is currently selected. In the main content area, it shows the 'KUSTO_DEV_KUSTO_11 > test-resource-group > kustodocs' path. It lists 'Diagnostics settings' with columns for NAME, STORAGE ACCOUNT, EVENT HUB, LOG ANALYTIC, and EDIT SETTING. A message says 'No diagnostic settings defined'. Below this is a button labeled '+ Add diagnostic setting' which is also highlighted with a red box. Further down, it says 'Click 'Add Diagnostic setting' above to configure the collection of the following data:' followed by a bulleted list: Cluster health, Query performance, Ingestion health and performance, Export health and performance.

3. Select **Add diagnostic setting**.

4. In the **Diagnostics settings** window:

The screenshot shows the 'Diagnostics settings' configuration window. At the top, there are three buttons: Save (highlighted with a red box), Discard, and Delete. Below that is a required 'Name' field with a placeholder '(Required)' and a red border. Under the 'LOG' section, there are two checkboxes: 'Archive to a storage account' and 'Send to Log Analytics' (both highlighted with red boxes). Under the 'METRIC' section, there are several checkboxes for different types of monitoring data: Cluster health, Query performance, Ingestion health and performance, Export health and performance, and Streaming ingest.

- a. Select **Name** for your diagnostic setting.
- b. Select one or more targets: a Storage account, Event Hub, or Log Analytics.
- c. Select logs to be collected: `SucceededIngestion` or `FailedIngestion`.
- d. Select `metrics` to be collected (optional).
- e. Select **Save** to save the new diagnostic logs settings and metrics.
- f. Create a **New support request** in the Azure portal to request activation of diagnostic logs.

New settings will be set in a few minutes. Logs then appear in the configured archival target (Storage account, Event Hub, or Log Analytics).

Diagnostic logs schema

All [Azure Monitor diagnostic logs share a common top-level schema](#). Azure Data Explorer has unique properties for their own events. All logs are stored in a JSON format.

Ingestion logs schema

Log JSON strings include elements listed in the following table:

NAME	DESCRIPTION
time	Time of the report
resourceId	Azure Resource Manager resource ID
operationName	Name of the operation: 'MICROSOFT.KUSTO/CLUSTERS/INGEST/ACTION'
operationVersion	Schema version: '1.0'
category	Category of the operation. <code>SucceededIngestion</code> or <code>FailedIngestion</code> . Properties differ for successful operation or failed operation .
properties	Detailed information of the operation.

Successful ingestion operation log

Example:

```
{
  "time": "",
  "resourceId": "",
  "operationName": "MICROSOFT.KUSTO/CLUSTERS/INGEST/ACTION",
  "operationVersion": "1.0",
  "category": "SucceededIngestion",
  "properties": {
    "succeededOn": "2019-05-27 07:55:05.3693628",
    "operationId": "b446c48f-6e2f-4884-b723-92eb6dc99cc9",
    "database": "Samples",
    "table": "StormEvents",
    "ingestionSourceId": "66a2959e-80de-4952-975d-b65072fc571d",
    "ingestionSourcePath": "https://kustoingestionlogs.blob.core.windows.net/sampleddata/events8347293.json",
    "rootActivityId": "d0bd5dd3-c564-4647-953e-05670e22a81d"
  }
}
```

Properties of a successful operation diagnostic log

NAME	DESCRIPTION
succeededOn	Time of ingestion completion
operationId	Azure Data Explorer ingestion operation ID
database	Name of the target database
table	Name of the target table
ingestionSourceId	ID of the ingestion data source
ingestionSourcePath	Path of the ingestion data source or blob URI
rootActivityId	Activity ID

Failed ingestion operation log

Example:

```
{
  "time": "",
  "resourceId": "",
  "operationName": "MICROSOFT.KUSTO/CLUSTERS/INGEST/ACTION",
  "operationVersion": "1.0",
  "category": "FailedIngestion",
  "properties": {
    "failedOn": "2019-05-27 08:57:05.4273524",
    "operationId": "5956515d-9a48-4544-a514-cf4656fe7f95",
    "database": "Samples",
    "table": "StormEvents",
    "ingestionSourceId": "eee56f8c-2211-4ea4-93a6-be556e853e5f",
    "ingestionSourcePath": "https://kustoingestionlogs.blob.core.windows.net/sampleddata/events5725592.json",
    "rootActivityId": "52134905-947a-4231-afaf-13d9b7b184d5",
    "details": "Permanent failure downloading blob. URI: ..., permanentReason: Download_SourceNotFound, DownloadFailedException: 'Could not find file ...''",
    "errorCode": "Download_SourceNotFound",
    "failureStatus": "Permanent",
    "originatesFromUpdatePolicy": false,
    "shouldRetry": false
  }
}
```

Properties of a failed operation diagnostic log

NAME	DESCRIPTION
failedOn	Time of ingestion completion
operationId	Azure Data Explorer ingestion operation ID
database	Name of the target database
table	Name of the target table

NAME	DESCRIPTION
ingestionSourceId	ID of the ingestion data source
ingestionSourcePath	Path of the ingestion data source or blob URI
rootActivityId	Activity ID
details	Detailed description of the failure and error message
errorCode	Error code
failureStatus	<code>Permanent</code> or <code>Transient</code> . Retry of a transient failure may succeed.
originatesFromUpdatePolicy	True if failure originates from an update policy
shouldRetry	True if retry may succeed

Next steps

- [Tutorial: Ingest and query monitoring data in Azure Data Explorer](#)
- [Use metrics to monitor cluster health](#)

Monitor Azure Data Explorer performance, health, and usage with metrics

11/18/2019 • 3 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast, fully managed data analytics service for real-time analysis on large volumes of data streaming from applications, websites, IoT devices, and more. To use Azure Data Explorer, you first create a cluster, and create one or more databases in that cluster. Then you ingest (load) data into a database so that you can run queries against it. Azure Data Explorer metrics provide key indicators as to the health and performance of the cluster resources. Use the metrics that are detailed in this article to monitor Azure Data Explorer cluster health and performance in your specific scenario as standalone metrics. You can also use metrics as the basis for operational [Azure Dashboards](#) and [Azure Alerts](#).

Prerequisites

- If you don't have an Azure subscription, create a [free Azure account](#).
- Create a [cluster and database](#).

Sign in to the Azure portal

Sign in to the [Azure portal](#).

Using metrics

In your Azure Data Explorer cluster, select **Metrics** to open the metrics pane and begin analysis on your cluster.

The screenshot shows the Azure Data Explorer Cluster interface for the 'kustodocs' database. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Query, Settings, Scale up, Scale out, Properties, Locks, Export template, Data (Databases), Monitoring (Alerts, Metrics), Support + troubleshooting, and New support request. The 'Metrics' link in the Monitoring section is highlighted with a red box. The main area displays a table of databases with columns for DATABASE, SIZE, RETENTION PERIOD, and CACHE PERIOD. Below the table is a chart titled 'CPU' showing CPU usage over time, with a value of 0.02% displayed at the bottom. A time range selector at the bottom of the chart allows selecting from 1 hour to 30 days.

In the Metrics pane:

The screenshot shows the 'kustodocs - Metrics' page. The left sidebar is identical to the previous one. The main area shows a chart of 'Avg CPU' usage over time. A context menu is open at the top right of the chart area, with the following numbered steps highlighted with red boxes: 1. Metric dropdown menu; 2. 'Add metric' button; 3. 'New chart' button; 4. Time range selector 'Last 24 hours (Automatic - 15 minutes)'; 5. 'Add filter' and 'Apply splitting' buttons; 6. 'Pin to dashboard' button; and 7. 'New alert rule' button. The chart itself shows CPU usage spikes, with a value of 0.02% displayed at the bottom.

1. To create a metric chart, select **Metric** name and relevant **Aggregation** per metric as detailed below. The **Resource** and **Metric Namespace** pickers are pre-selected to your Azure Data Explorer cluster.

Metric	Unit	Aggregation	Metric Description
Cache utilization	Percent	Avg, Max, Min	<p>Percentage of allocated cache resources currently in use by the cluster. Cache refers to the size of SSD allocated for user activity according to the defined cache policy. An average cache utilization of 80% or less is a sustainable state for a cluster. If the average cache utilization is above 80%, the cluster should be scaled up to a storage optimized pricing tier or scaled out to more instances. Alternatively, adapt the cache policy (fewer days in cache). If cache utilization is over 100%, the size of data to be cached, according to the caching policy, is larger than the total size of cache on the cluster.</p>
CPU	Percent	Avg, Max, Min	<p>Percentage of allocated compute resources currently in use by machines in the cluster. An average CPU of 80% or less is sustainable for a cluster. The maximum value of CPU is 100% which means there are no additional compute resources to process data. When a cluster isn't performing well, check the maximum value of the CPU to determine if there are specific CPUs that are blocked.</p>
Events processed (for Event Hubs)	Count	Max, Min, Sum	<p>Total number of events read from event hubs and processed by the cluster. The events are split into events rejected and events accepted by the cluster engine.</p>
Ingestion latency	Seconds	Avg, Max, Min	<p>Latency of data ingested, from the time the data was received in the cluster until it's ready for query. The ingestion latency period depends on the ingestion scenario.</p>

METRIC	UNIT	AGGREGATION	METRIC DESCRIPTION
Ingestion result	Count	Count	Total number of ingestion operations that failed and succeeded. Use apply splitting to create buckets of success and fail results and analyze the dimensions (Value > Status).
Ingestion utilization	Percent	Avg, Max, Min	Percentage of actual resources used to ingest data from the total resources allocated, in the capacity policy, to perform ingestion. The default capacity policy is no more than 512 concurrent ingestion operations or 75% of the cluster resources invested in ingestion. Average ingestion utilization of 80% or less is a sustainable state for a cluster. Maximum value of ingestion utilization is 100%, which means all cluster ingestion ability is used and an ingestion queue may result.
Ingestion volume (in MB)	Count	Max, Min, Sum	The total size of data ingested to the cluster (in MB) before compression.
Keep alive	Count	Avg	Tracks the responsiveness of the cluster. A fully responsive cluster returns value 1 and a blocked or disconnected cluster returns 0.
Query duration	Seconds	Count, Avg, Min, Max, Sum	Total time until query results are received (doesn't include network latency).

Additional information regarding [supported Azure Data Explorer cluster metrics](#)

2. Select the **Add metric** button to see multiple metrics plotted in the same chart.
3. Select the **+ New chart** button to see multiple charts in one view.
4. Use the time picker to change the time range (default: past 24 hours).
5. Use **Add filter** and **Apply splitting** for metrics that have dimensions.
6. Select **Pin to dashboard** to add your chart configuration to the dashboards so that you can view it again.

7. Set **New alert rule** to visualize your metrics using the set criteria. The new alerting rule will include your target resource, metric, splitting, and filter dimensions from your chart. Modify these settings in the [alert rule creation pane](#).

Additional information on using the [Metrics Explorer](#).

Next steps

- [Tutorial: Ingest and query monitoring data in Azure Data Explorer](#)
- [Monitor Azure Data Explorer ingestion operations using diagnostic logs](#)
- [Quickstart: Query data in Azure Data Explorer](#)

Integrate Azure Data Explorer with Azure Data Factory

12/4/2019 • 11 minutes to read • [Edit Online](#)

Azure Data Factory (ADF) is a cloud-based data integration service that allows you to integrate different data stores and perform activities on the data. ADF allows you to create data-driven workflows for orchestrating and automating data movement and data transformation. Azure Data Explorer is one of the [supported data stores](#) in Azure Data Factory.

Azure Data Factory activities for Azure Data Explorer

Various integrations with Azure Data Factory are available for Azure Data Explorer users:

Copy activity

Azure Data Factory Copy activity is used to transfer data between data stores. Azure Data Explorer is supported as a source, where data is copied from Azure Data Explorer to any supported data store, and a sink, where data is copied from any supported data store to Azure Data Explorer. For more information, see [copy data to or from Azure Data Explorer using Azure Data Factory](#), and for a detailed walk-through see [load data from Azure Data Factory into Azure Data Explorer](#). Azure Data Explorer is supported by Azure IR (Integration Runtime), used when data is copied within Azure, and self-hosted IR, used when copying data from/to data stores located on-premises or in a network with access control, such as an Azure Virtual Network. For more information, see [which IR to use](#)

TIP

When using the copy activity and creating a **Linked Service** or a **Dataset**, select the data store **Azure Data Explorer (Kusto)** and not the old data store **Kusto**.

Lookup activity

The Lookup activity is used for executing queries on Azure Data Explorer. The result of the query will be returned as the output of the Lookup activity, and can be used in the next activity in the pipeline as described in the [ADF Lookup documentation](#).

In addition to the response size limit of 5,000 rows and 2 MB, the activity also has a query timeout limit of 1 hour.

Command activity

The Command activity allows the execution of Azure Data Explorer [control commands](#). Unlike queries, the control commands can potentially modify data or metadata. Some of the control commands are targeted to ingest data into Azure Data Explorer, using commands such as `.ingest` or `.set-or-append`) or copy data from Azure Data Explorer to external data stores using commands such as `.export`. For a detailed walk-through of the command activity, see [use Azure Data Factory command activity to run Azure Data Explorer control commands](#). Using a control command to copy data can, at times, be a faster and cheaper option than the Copy activity. To determine when to use the Command activity versus the Copy activity, see [select between Copy and Command activities when copying data](#).

Copy in bulk from a database template

The [Copy in bulk from a database to Azure Data Explorer by using the Azure Data Factory template](#) is a predefined Azure Data Factory pipeline. The template is used to create many pipelines per database or per table for faster data copying.

Select between Copy and Azure Data Explorer Command activities when copy data

This section will assist you in selecting the correct activity for your data copying needs.

When copying data from or to Azure Data Explorer, there are two available options in Azure Data Factory:

- Copy activity.
- Azure Data Explorer Command activity, which executes one of the control commands that transfer data in Azure Data Explorer.

Copy data from Azure Data Explorer

You can copy data from Azure Data Explorer using the copy activity or the `.export` command. The `.export` command executes a query, and then exports the results of the query.

See the following table for a comparison of the Copy activity and `.export` command for copying data from Azure Data Explorer.

	COPY ACTIVITY	.EXPORT COMMAND
Flow description	ADF executes a query on Kusto, processes the result, and sends it to the target data store. (ADX > ADF > sink data store)	ADF sends an <code>.export</code> control command to Azure Data Explorer, which executes the command, and sends the data directly to the target data store. (ADX > sink data store)
Supported target data stores	A wide variety of supported data stores	ADLSv2, Azure Blob, SQL Database
Performance	Centralized	<ul style="list-style-type: none">• Distributed (default), exporting data from multiple nodes concurrently• Faster and COGS efficient.
Server limits	<p>Query limits can be extended/disabled. By default, ADF queries contain:</p> <ul style="list-style-type: none">• Size limit of 500,000 records or 64 MB.• Time limit of 10 minutes.• <code>noTruncation</code> set to false.	<p>By default, extends or disables the query limits:</p> <ul style="list-style-type: none">• Size limits are disabled.• Server timeout is extended to 1 hour.• <code>MaxMemoryConsumptionPerIterator</code> and <code>MaxMemoryConsumptionPerQueryPerNode</code> are extended to max (5 GB, <code>TotalPhysicalMemory/2</code>).

TIP

If your copy destination is one of the data stores supported by the `.export` command, and if none of the Copy activity features is crucial to your needs, select the `.export` command.

Copying data to Azure Data Explorer

You can copy data to Azure Data Explorer using the copy activity or ingestion commands such as [ingest from query](#) (`.set-or-append`, `.set-or-replace`, `.set`, `.replace`), and [ingest from storage](#) (`.ingest`).

See the following table for a comparison of the Copy activity, and ingestion commands for copying data to Azure Data Explorer.

	COPY ACTIVITY	INGEST FROM QUERY .SET-OR-APPEND .SET-OR-REPLACE / .SET / .REPLACE	INGEST FROM STORAGE .INGEST
Flow description	ADF gets the data from the source data store, converts it into a tabular format, and does the required schema-mapping changes. ADF then uploads the data to Azure blobs, splits it into chunks, then downloads the blobs to ingest them into the ADX table. (Source data store > ADF > Azure blobs > ADX)	These commands can execute a query or a <code>.show</code> command, and ingest the results of the query into a table (ADX > ADX).	This command ingests data into a table by "pulling" the data from one or more cloud storage artifacts.
Supported source data stores	variety of options	ADLS Gen 2, Azure Blob, SQL (using the <code>sql_request</code> plugin), Cosmos (using the <code>cosmosdb_sql_request</code> plugin), and any other data store that provides HTTP or Python APIs.	Filesystem, Azure Blob Storage, ADLS Gen 1, ADLS Gen 2
Performance	Ingestions are queued and managed, which ensures small-size ingestions and assures high availability by providing load balancing, retries and error handling.	<ul style="list-style-type: none"> Those commands weren't designed for high volume data importing. Works as expected and cheaper. But for production scenarios and when traffic rates and data sizes are large, use the Copy activity. 	
Server Limits	<ul style="list-style-type: none"> No size limit. Max timeout limit: 1 hour per ingested blob. 	<ul style="list-style-type: none"> There's only a size limit on the query part, which can be skipped by specifying <code>noTruncation=true</code> Max timeout limit: 1 hour. 	<ul style="list-style-type: none"> No size limit. Max timeout limit: 1 hour.

TIP

- When copying data from ADF to Azure Data Explorer use the `ingest from query` commands.
- For large data sets (>1GB), use the Copy activity.

Required permissions

The following table lists the required permissions for various steps in the integration with Azure Data Factory.

Step	Operation	Minimum Level of Permissions	Notes
Create a Linked Service	Database navigation	<i>database viewer</i> The logged-in user using ADF should be authorized to read database metadata.	User can provide the database name manually.
	Test Connection	<i>database monitor or table ingestor</i> Service principal should be authorized to execute database level <code>.show</code> commands or table level ingestion.	<ul style="list-style-type: none"> TestConnection verifies the connection to the cluster, and not to the database. It can succeed even if the database doesn't exist. Table admin permissions aren't sufficient.
Creating a Dataset	Table navigation	<i>database monitor</i> The logged in user using ADF, must be authorized to execute database level <code>.show</code> commands.	User can provide table name manually.
Creating a Dataset or Copy Activity	Preview data	<i>database viewer</i> Service principal must be authorized to read database metadata.	
	Import schema	<i>database viewer</i> Service principal must be authorized to read database metadata.	When ADX is the source of a tabular-to-tabular copy, ADF will import schema automatically, even if the user didn't import schema explicitly.
ADX as Sink	Create a by-name column mapping	<i>database monitor</i> Service principal must be authorized to execute database level <code>.show</code> commands.	<ul style="list-style-type: none"> All mandatory operations will work with <i>table ingestor</i>. Some optional operations can fail.
	<ul style="list-style-type: none"> Create a CSV mapping on the table Drop the mapping 	<i>table ingestor or database admin</i> Service principal must be authorized to make changes to a table.	
	Ingest data	<i>table ingestor or database admin</i> Service principal must be authorized to make changes to a table.	

Step	Operation	Minimum level of permissions	Notes
ADX as source	Execute query	<i>database viewer</i> Service principal must be authorized to read database metadata.	
Kusto command		According to the permissions level of each command.	

Performance

If Azure Data Explorer is the source and you use the Lookup, copy, or command activity that contains a query where, refer to [query best practices](#) for performance information and [ADF documentation for copy activity](#).

This section addresses the use of copy activity where Azure Data Explorer is the sink. The estimated throughput for Azure Data Explorer sink is 11-13 MBps. The following table details the parameters influencing the performance of the Azure Data Explorer sink.

Parameter	Notes
Components geographical proximity	Place all components in the same region: <ul style="list-style-type: none">• source and sink data stores.• ADF integration runtime.• Your ADX cluster. Make sure that at least your integration runtime is in the same region as your ADX cluster.
Number of DIUs	1 VM for every 4 DIUs used by ADF. Increasing the DIUs will help only if your source is a file-based store with multiple files. Each VM will then process a different file in parallel. Therefore, copying a single large file will have a higher latency than copying multiple smaller files.
Amount and SKU of your ADX cluster	High number of ADX nodes will boost ingestion processing time.
Parallelism	To copy a very large amount of data from a database, partition your data and then use a ForEach loop that copies each partition in parallel or use the Bulk Copy from Database to Azure Data Explorer Template . Note: Settings > Degree of Parallelism in the Copy activity isn't relevant to ADX.
Data processing complexity	Latency varies according to source file format, column mapping, and compression.
The VM running your integration runtime	<ul style="list-style-type: none">• For Azure copy, ADF VMs and machine SKUs can't be changed.• For on-prem to Azure copy, determine that the VM hosting your self-hosted IR is strong enough.

Tips and common pitfalls

Monitor activity progress

- When monitoring the activity progress, the *Data written* property may be much larger than the *Data read* property because *Data read* is calculated according to the binary file size, while *Data written* is calculated according to the in-memory size, after data is de-serialized and decompressed.
- When monitoring the activity progress, you can see that data is written to the Azure Data Explorer sink. When querying the Azure Data Explorer table, you see that data hasn't arrived. This is because there are two stages when copying to Azure Data Explorer.
 - First stage reads the source data, splits it to 900-MB chunks, and uploads each chunk to an Azure Blob. The first stage is seen by the ADF activity progress view.
 - The second stage begins once all the data is uploaded to Azure Blobs. The Azure Data Explorer engine nodes download the blobs and ingest the data into the sink table. The data is then seen in your Azure Data Explorer table.

Failure to ingest CSV files due to improper escaping

Azure Data Explorer expects CSV files to align with [RFC 4180](#). It expects:

- Fields that contain characters that require escaping (such as " and new lines) should start and end with a " character, without whitespace. All " characters *inside* the field are escaped by using a double " character (""). For example, "Hello, ""World"" is a valid CSV file with a single record having a single column or field with the content Hello, "World".
- All records in the file must have the same number of columns and fields.

Azure Data Factory allows the backslash (escape) character. If you generate a CSV file with a backslash character using Azure Data Factory, ingestion of the file to Azure Data Explorer will fail.

Example

The following text values: Hello, "World"

ABC DEF

"ABC\DEF

"ABC DEF

Should appear in a proper CSV file as follows: "Hello, ""World"""

"ABC DEF"

""ABC DEF"

""ABC\DEF"

By using the default escape character (backslash), the following CSV won't work with Azure Data Explorer: "Hello, "World""

"ABC DEF"

""ABC DEF"

""ABC\DEF"

Nested JSON objects

When copying a JSON file to Azure Data Explorer, note that:

- Arrays aren't supported.
- If your JSON structure contains object data types, Azure Data Factory will flatten the object's child items, and try to map each child item to a different column in your Azure Data Explorer table. If you want the entire object item to be mapped to a single column in Azure Data Explorer:
 - Ingest the entire JSON row into a single dynamic column in Azure Data Explorer.
 - Manually edit the pipeline definition by using Azure Data Factory's JSON editor. In **Mappings**
 - Remove the multiple mappings that were created for each child item, and add a single mapping that maps your object type to your table column.
 - After the closing square bracket, add a comma followed by:

```
"mapComplexValuesToString": true .
```

Specify AdditionalProperties when copying to Azure Data Explorer

NOTE

This feature is currently available by manually editing the JSON payload.

Add a single row under the "sink" section of the copy activity as follows:

```
"sink": {  
    "type": "AzureDataExplorerSink",  
    "additionalProperties": "{\"tags\":\\\"[\\\\\"drop-by:account_FiscalYearID_2020\\\\\"]\\\"}"  
},
```

Escaping of the value may be tricky. Use the following code snippet as a reference:

```
static void Main(string[] args)  
{  
    Dictionary<string, string> additionalProperties = new Dictionary<string, string>();  
    additionalProperties.Add("ignoreFirstRecord", "false");  
    additionalProperties.Add("csvMappingReference", "Table1_mapping_1");  
    IEnumerable<string> ingestIfNotExists = new List<string> { "Part0001" };  
    additionalProperties.Add("ingestIfNotExists", JsonConvert.SerializeObject(ingestIfNotExists));  
    IEnumerable<string> tags = new List<string> { "ingest-by:Part0001", "ingest-by:IngestedByTest" };  
    additionalProperties.Add("tags", JsonConvert.SerializeObject(tags));  
    var additionalPropertiesForPayload = JsonConvert.SerializeObject(additionalProperties);  
    Console.WriteLine(additionalPropertiesForPayload);  
    Console.ReadLine();  
}
```

The printed value:

```
{"ignoreFirstRecord":"false","csvMappingReference":"Table1_mapping_1","ingestIfNotExists":  
["\"Part0001\""],"tags":["\"ingest-by:Part0001\",\"ingest-by:IngestedByTest\"]"}
```

Next steps

- Learn how to [copy data to Azure Data Explorer by using Azure Data Factory](#).
- Learn about using [Azure Data Factory template for bulk copy from database to Azure Data Explorer](#).
- Learn about using [Azure Data Factory command activity to run Azure Data Explorer control commands](#).
- Learn about [Azure Data Explorer queries](#) for data querying.

Copy in bulk from a database to Azure Data Explorer by using the Azure Data Factory template

9/25/2019 • 3 minutes to read • [Edit Online](#)

Azure Data Explorer is a fast, fully managed, data-analytics service. It offers real-time analysis on large volumes of data that stream from many sources, such as applications, websites, and IoT devices.

Azure Data Factory is a fully managed, cloud-based, data-integration service. You can use it to populate your Azure Data Explorer database with data from your existing system. And it can help you save time when you're building analytics solutions.

[Azure Data Factory templates](#) are predefined Data Factory pipelines. These templates can help you get started quickly with Data Factory and reduce development time on data integration projects.

You create the *Bulk Copy from Database to Azure Data Explorer* template by using *Lookup* and *ForEach* activities. For faster data copying, you can use the template to create many pipelines per database or per table.

IMPORTANT

Be sure to use the tool that's appropriate for the quantity of data you want to copy.

- Use the *Bulk Copy from Database to Azure Data Explorer* template to copy large amounts of data from databases such as SQL server and Google BigQuery to Azure Data Explorer.
- Use the [Data Factory Copy Data tool](#) to copy a few tables with small or moderate amounts of data into Azure Data Explorer.

Prerequisites

- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.
- [An Azure Data Explorer cluster and database](#).
- [Create a data factory](#).
- A source of data in a database.

Create ControlTableDataset

ControlTableDataset indicates what data will be copied from the source to the destination in the pipeline. The number of rows indicates the total number of pipelines that are needed to copy the data. You should define *ControlTableDataset* as part of the source database.

An example of the SQL Server source table format is shown in the following code:

```
CREATE TABLE control_table (
    PartitionId int,
    SourceQuery varchar(255),
    ADXTableName varchar(255)
);
```

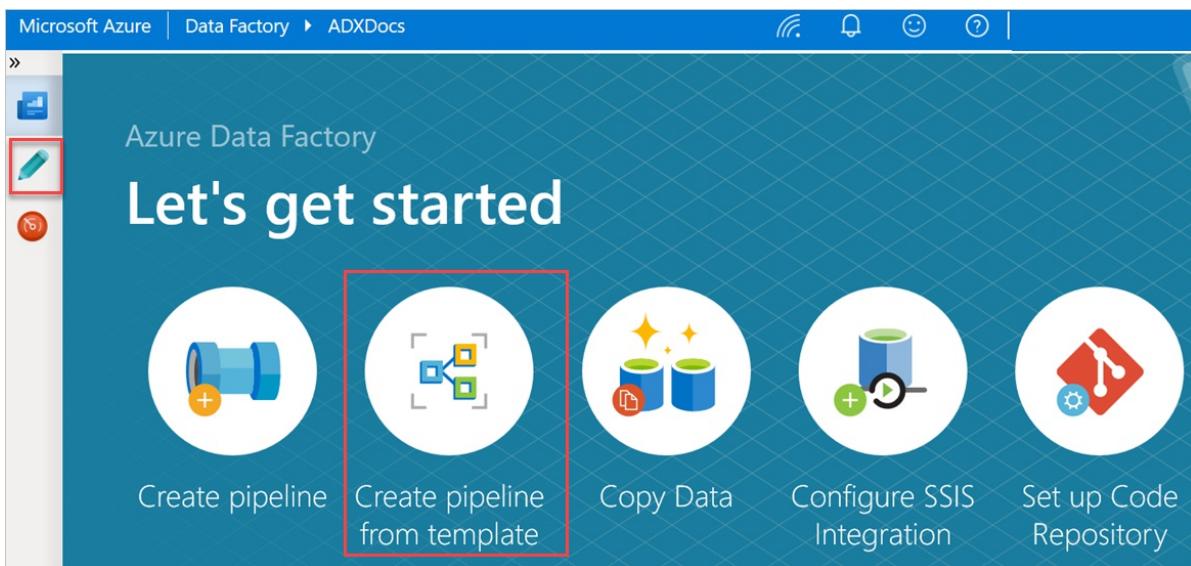
The code elements are described in the following table:

PROPERTY	DESCRIPTION	EXAMPLE
PartitionId	The copy order	1
SourceQuery	The query that indicates which data will be copied during the pipeline runtime	<pre>select * from table where lastmodifiedtime LastModifytime >= '2015-01-01 00:00:00'></pre>
ADXTableName	The destination table name	MyAdxTable

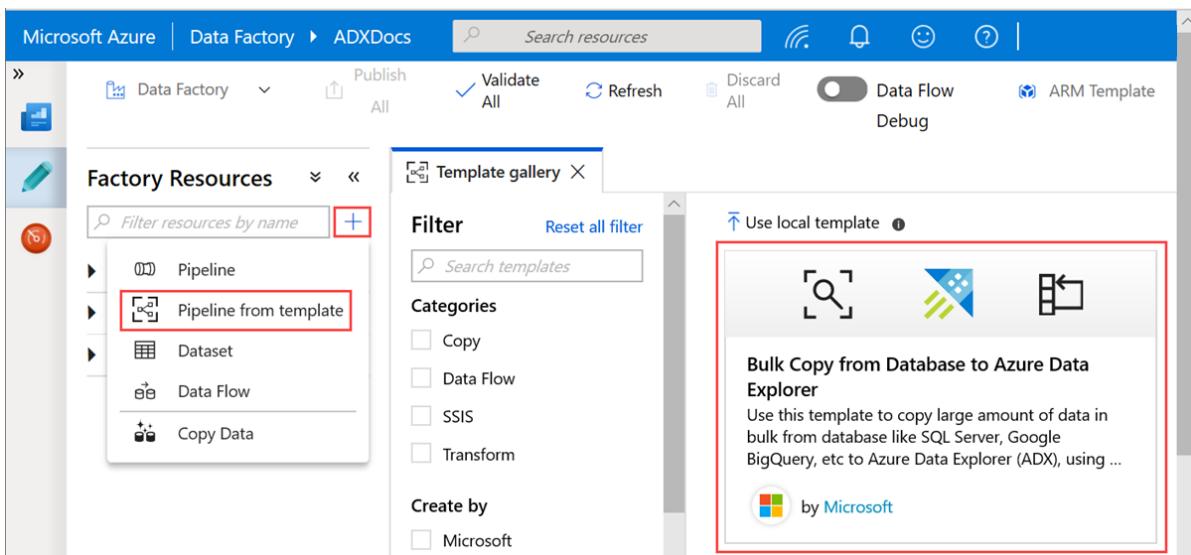
If your ControlTableDataset is in a different format, create a comparable ControlTableDataset for your format.

Use the Bulk Copy from Database to Azure Data Explorer template

1. In the **Let's get started** pane, select **Create pipeline from template** to open the **Template gallery** pane.



2. Select the **Bulk Copy from Database to Azure Data Explorer** template.



3. In the **Bulk Copy from Database to Azure Data Explorer** pane, under **User Inputs**, specify your datasets by doing the following:

- a. In the **ControlTableDataset** drop-down list, select the linked service to the control table that indicates what data is copied from the source to the destination and where it will be placed in the destination.

b. In the **SourceDataset** drop-down list, select the linked service to the source database.

c. In the **AzureDataExplorerTable** drop-down list, select the Azure Data Explorer table. If the dataset doesn't exist, [create the Azure Data Explorer linked service](#) to add the dataset.

d. Select **Use this template**.

The screenshot shows the 'Bulk Copy from Database to Azure Data Explorer' template configuration. On the left, there is a canvas with a 'Lookup' activity followed by a 'ForEach' activity. Below the canvas, there is descriptive text about the template and a link to documentation. On the right, there is a panel titled 'User Inputs' containing four entries:

- ControlTableDataset**: Dataset for control table, which is used to store the partition list of source data.
- SourceLinkedService ***: SQLServer (selected)
- SourceDataset**: Dataset for source.
- SourceLinkedService ***: SQLServer (selected)
- AzureDataExplorerTable**: Azure Data Explorer (Kusto) (selected)
- AzureDataExplorer ***: AzureDataExplorer (selected)

At the bottom, there are 'Cancel', 'Export template', and 'Use this template' buttons. The 'Use this template' button is highlighted with a red border.

4. Select an area in the canvas, outside the activities, to access the template pipeline. Select the **Parameters** tab to enter the parameters for the table, including **Name** (control table name) and **Default value** (column names).

The screenshot shows the Microsoft Azure Data Factory pipeline editor. A pipeline named 'CopyIntoADX' is selected in the 'Template gallery'. The pipeline consists of a 'Lookup' activity followed by a 'ForEach' activity. In the bottom pane, the 'Parameters' tab is selected, showing a table of parameters:

NAME	TYPE	DEFAULT VALUE
ControlTableName	String	control_table
ControlTableSchemaPartition	String	PartitionId
ControlTableSchemaSourceQ	String	SourceQuery
ControlTableSchemaADXTabl	String	ADXTableName

5. Under **Lookup**, select **GetPartitionList** to view the default settings. The query is automatically created.

6. Select the Command activity, **ForEachPartition**, select the **Settings** tab, and then do the following:

a. In the **Batch count** box, enter a number from 1 to 50. This selection determines the number of pipelines that run in parallel until the number of *ControlTableDataset* rows is reached.

b. To ensure that the pipeline batches run in parallel, *do not* select the **Sequential** check box.

The screenshot shows the Microsoft Azure Data Factory pipeline editor. On the left, the 'Factory Resources' sidebar lists 'Pipelines' (2), 'Datasets' (6), and 'Data Flows (Preview)' (0). The main workspace displays a pipeline named 'CopyIntoADX'. The pipeline consists of a 'Lookup' activity followed by a 'ForEach' activity. The 'ForEach' activity is expanded to show its child activity, 'ForEachPartition'. A red box highlights the 'ForEach' activity. Another red box highlights the 'Settings' tab of the 'ForEach' activity's properties pane, which shows the 'Sequential' checkbox is unchecked and the 'Batch count' is set to 5. The 'Items' field contains the expression '@activity('GetPartitionList').output.value'.

TIP

The best practice is to run many pipelines in parallel so that your data can be copied more quickly. To increase efficiency, partition the data in the source table and allocate one partition per pipeline, according to date and table.

7. Select **Validate All** to validate the Azure Data Factory pipeline, and then view the result in the **Pipeline Validation Output** pane.

The screenshot shows the Microsoft Azure Data Factory pipeline editor. The 'Validate All' button in the top toolbar is highlighted with a red box. The pipeline workspace shows the same 'CopyIntoADX' pipeline with its activities. The 'Pipeline Validation Output' pane on the right is also highlighted with a red box. It displays a green bar chart icon and the message 'Your Pipeline has been validated. No errors were found.'

8. If necessary, select **Debug**, and then select **Add trigger** to run the pipeline.

The screenshot shows the Microsoft Azure Data Factory pipeline editor. The 'Debug' and 'Add trigger' buttons in the top toolbar are highlighted with red boxes. The pipeline workspace shows the 'CopyIntoADX' pipeline with its activities. The 'ForEach' activity is expanded to show its child activity, 'ForEachPartition'.

You can now use the template to efficiently copy large amounts of data from your databases and tables.

Next steps

- Learn how to [copy data to Azure Data Explorer by using Azure Data Factory](#).
- Learn about the [Azure Data Explorer connector](#) in Azure Data Factory.
- Learn about [Azure Data Explorer queries](#) for data querying.

Use Azure Data Factory command activity to run Azure Data Explorer control commands

10/10/2019 • 5 minutes to read • [Edit Online](#)

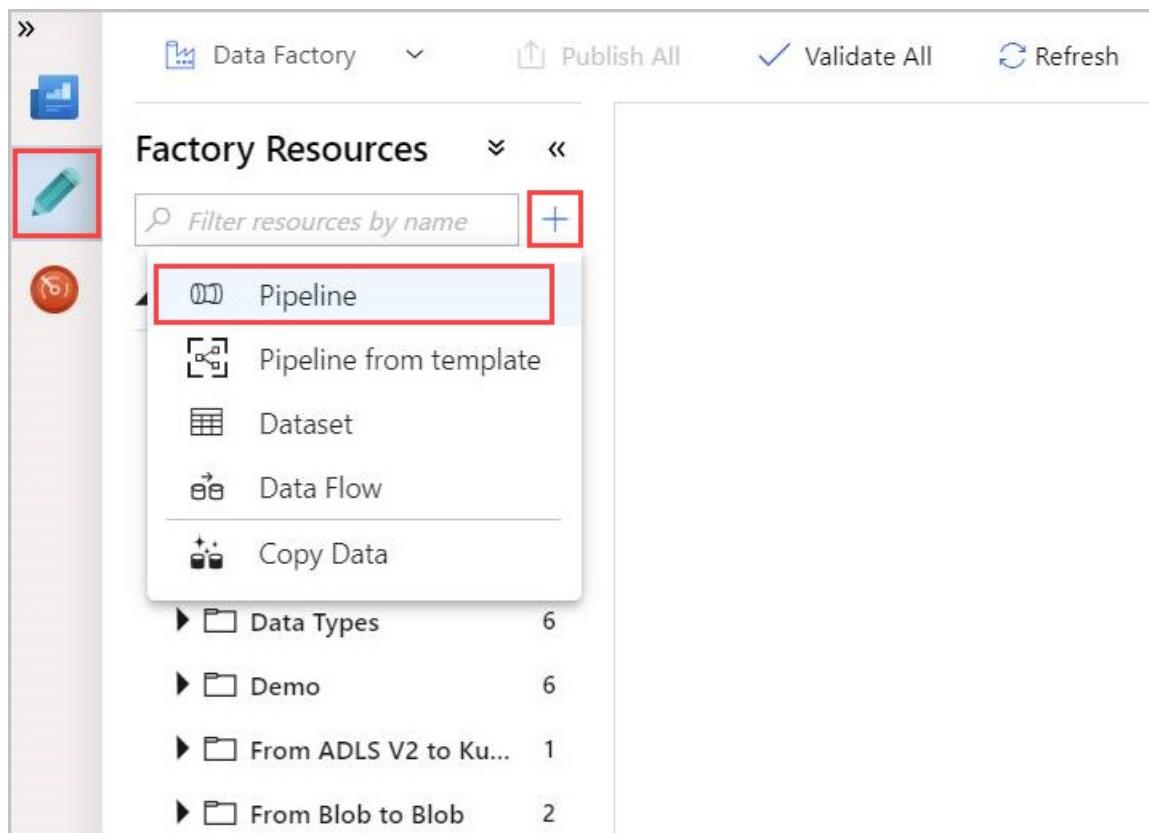
Azure Data Factory (ADF) is a cloud-based data integration service that allows you to perform a combination of activities on the data. Use ADF to create data-driven workflows for orchestrating and automating data movement and data transformation. The **Azure Data Explorer Command** activity in Azure Data Factory enables you to run [Azure Data Explorer control commands](#) within an ADF workflow. This article teaches you how to create a pipeline with a lookup activity and ForEach activity containing an Azure Data Explorer command activity.

Prerequisites

- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.
- [An Azure Data Explorer cluster and database](#)
- A source of data.
- [A data factory](#)

Create a new pipeline

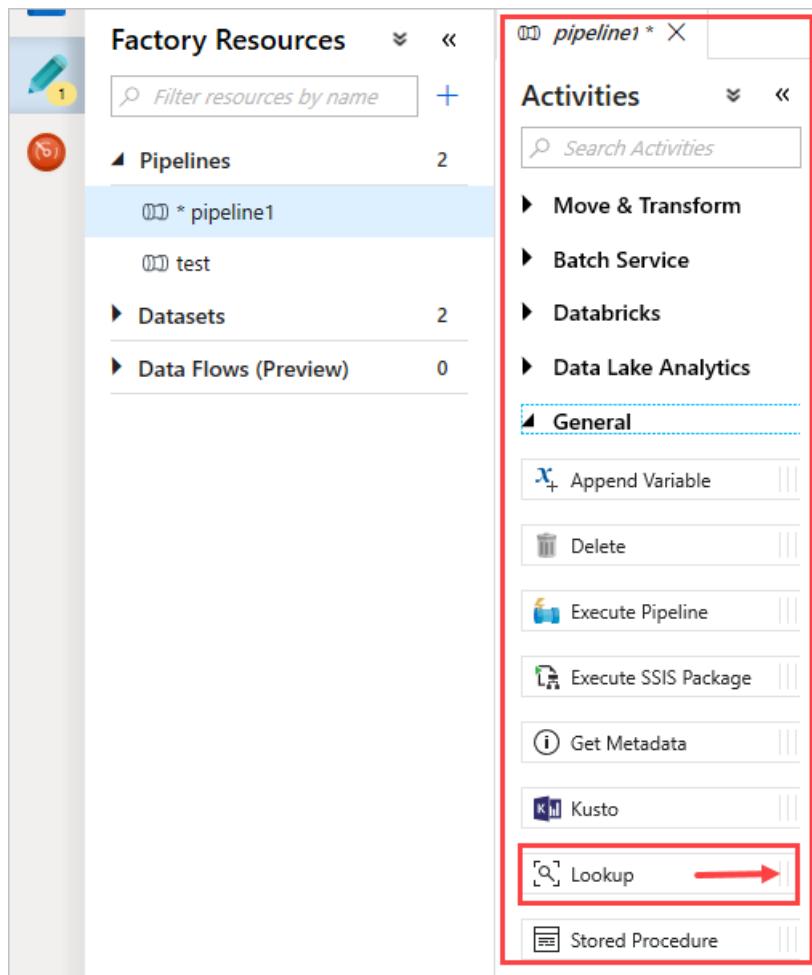
1. Select the **Author** pencil tool.
2. Create a new pipeline by selecting + and then select **Pipeline** from the drop-down.



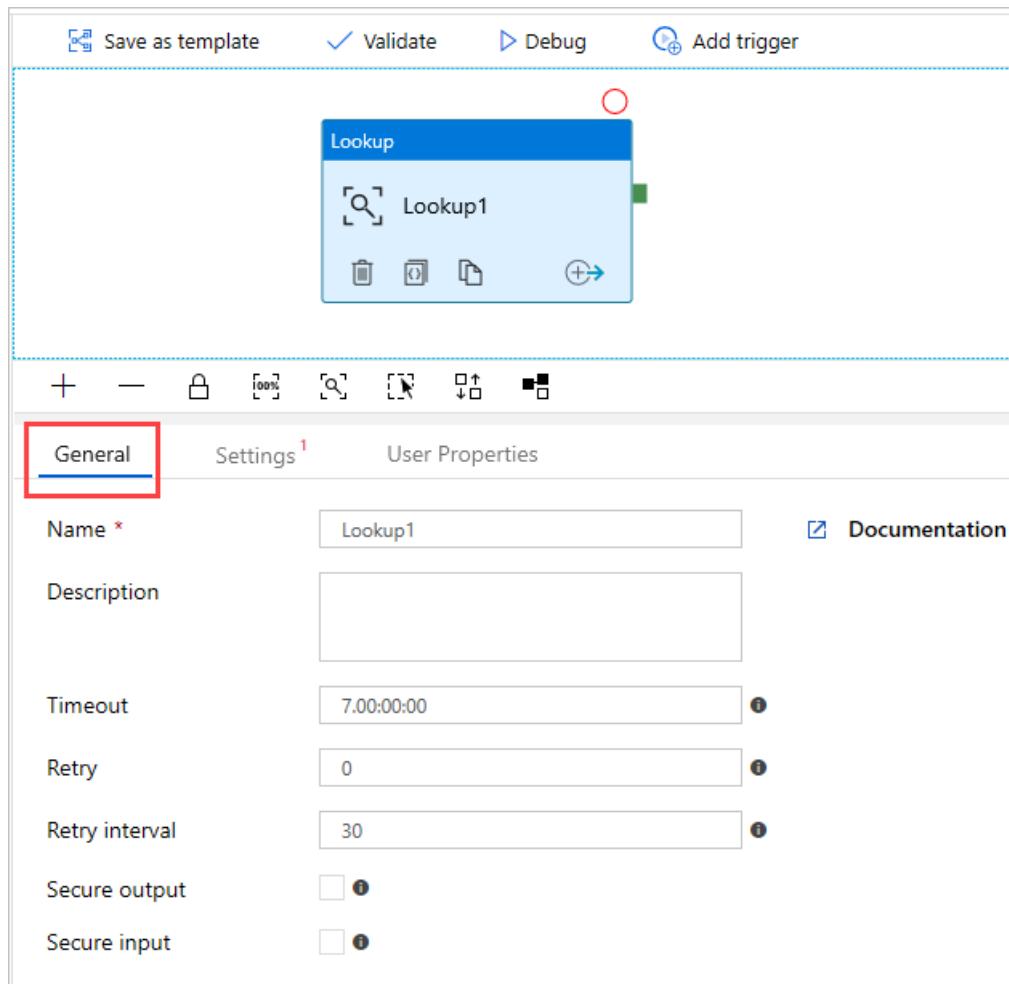
Create a Lookup activity

A [lookup activity](#) can retrieve a dataset from any Azure Data Factory-supported data sources. The output from Lookup activity can be used in a ForEach or other activity.

1. In the **Activities** pane, under **General**, select the **Lookup** activity. Drag and drop it into the main canvas on the right.



2. The canvas now contains the Lookup activity you created. Use the tabs below the canvas to change any relevant parameters. In **General**, rename the activity.

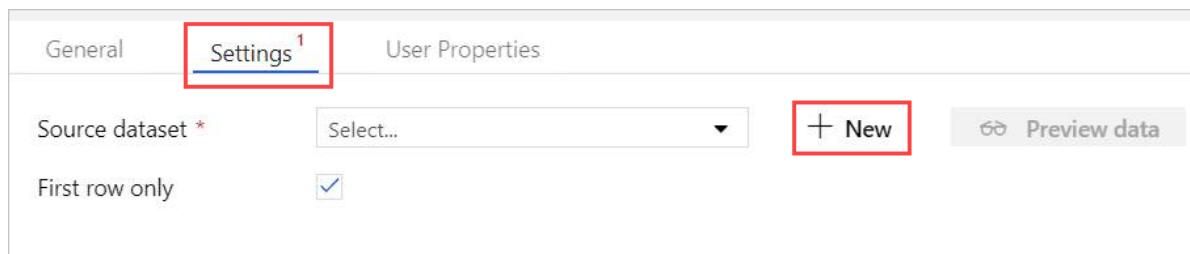


TIP

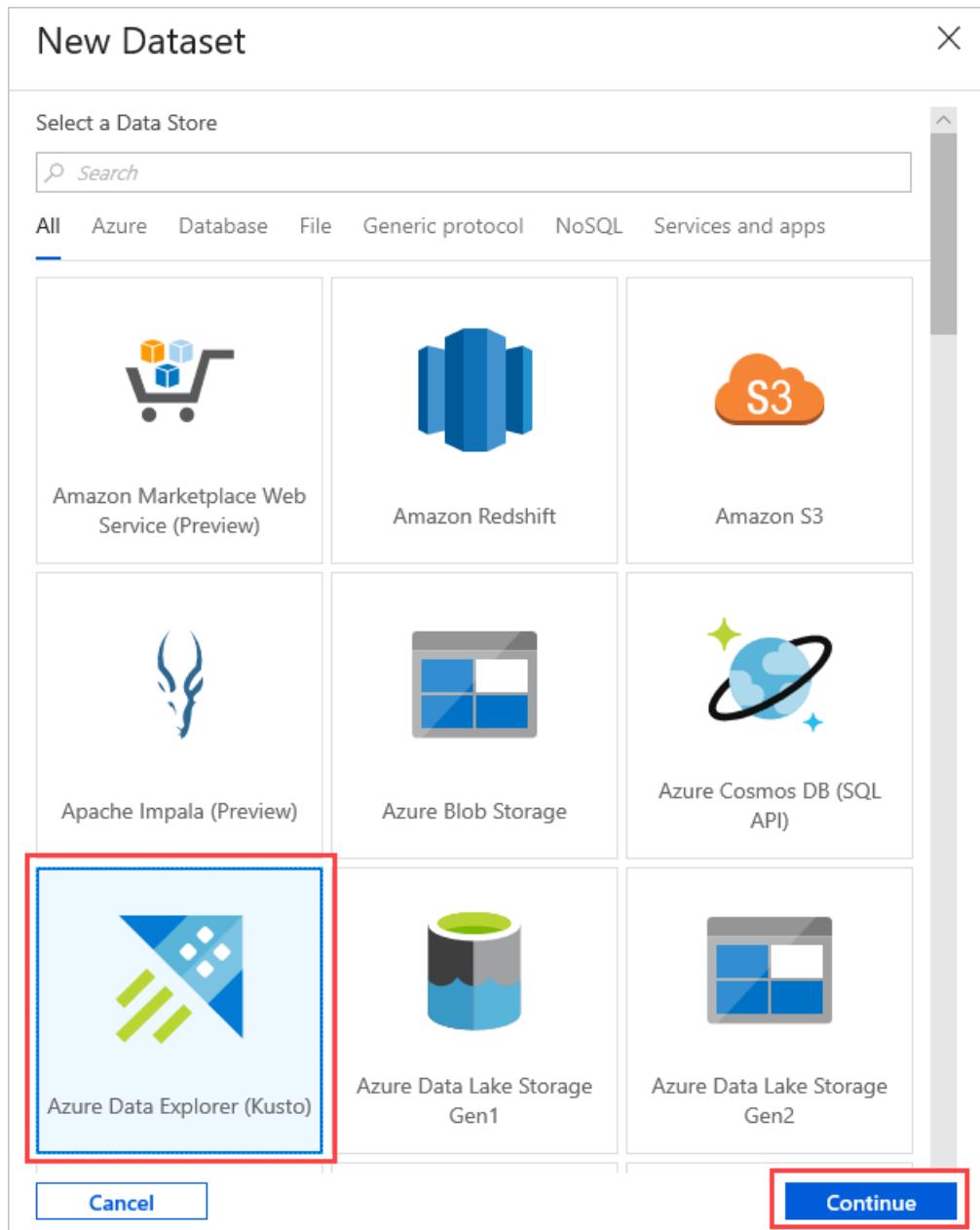
Click on the empty canvas area to view the pipeline properties. Use the **General** tab to rename the pipeline. Our pipeline is named *pipeline-4-docs*.

Create an Azure Data Explorer dataset in lookup activity

1. In **Settings**, select your pre-created Azure Data Explorer **Source dataset**, or select **+ New** to create a new dataset.



2. Select the **Azure Data Explorer (Kusto)** dataset from **New Dataset** window. Select **Continue** to add the new dataset.



3. The new Azure Data Explorer dataset parameters are visible in **Settings**. To update the parameters, select **Edit**.

The screenshot shows the 'Settings' tab for an Azure Data Explorer dataset. The 'General' tab is selected. The 'Source dataset' dropdown is set to 'AzureDataExplorerTable8' and is highlighted with a red border. The 'Query' field contains the text 'table1 | take 10'. Other settings include 'Query timeout' (00:10:00), 'No truncation' (unchecked), and 'First row only' (checked). The 'Edit' button is also highlighted with a red border.

4. The **AzureDataExplorerTable** new tab opens in the main canvas.

- Select **General** and edit the dataset name.
- Select **Connection** to edit the dataset properties.
- Select the **Linked service** from the drop-down, or select **+ New** to create a new linked service.

Azure Data Explorer (Kusto)
AzureDataExplorerTable1

General **Connection** Parameters

Linked service * Select... + New

Table **Preview data**

- When creating a new linked service, the **New Linked Service (Azure Data Explorer)** page opens:

New Linked Service (Azure Data Explorer (Kusto))

X

Name *

MyKustoLinkedService

Description

Connect via integration runtime *

AutoResolveIntegrationRuntime

Account selection method

From Azure subscription

Enter manually

Endpoint *

[REDACTED]

Tenant *

[REDACTED]

Service principal ID *

[REDACTED]

Service principal key

Azure Key Vault

Service principal key *

[REDACTED]

Database *

[REDACTED]

↻ ⓘ

Edit

Annotations

+ New

► Advanced ⓘ

✓ Connection successful

Cancel

Test connection

Finish

- Select **Name** for Azure Data Explorer linked service. Add **Description** if needed.
- In **Connect via integration runtime**, change current settings, if needed.
- In **Account selection method** select your cluster using one of two methods:
 - Select the **From Azure subscription** radio button and select your **Azure subscription** account. Then, select your **Cluster**. Note the drop-down will only list clusters that belong to the user.
 - Instead, select **Enter manually** radio button and enter your **Endpoint** (cluster URL).
- Specify the **Tenant**.
- Enter **Service principal ID**. The principal ID must have the adequate permissions, according to the permission level required by the command being used.
- Select **Service principal key** button and enter **Service Principal Key**.
- Select your **Database** from the dropdown menu. Alternatively, select **Edit** checkbox and enter your

database name.

- Select **Test Connection** to test the linked service connection you created. If you can connect to your setup, a green checkmark **Connection successful** will appear.
 - Select **Finish** to complete linked service creation.
6. Once you've set up a linked service, In **AzureDataExplorerTable > Connection**, add **Table** name. Select **Preview data**, to make sure that the data is presented properly.

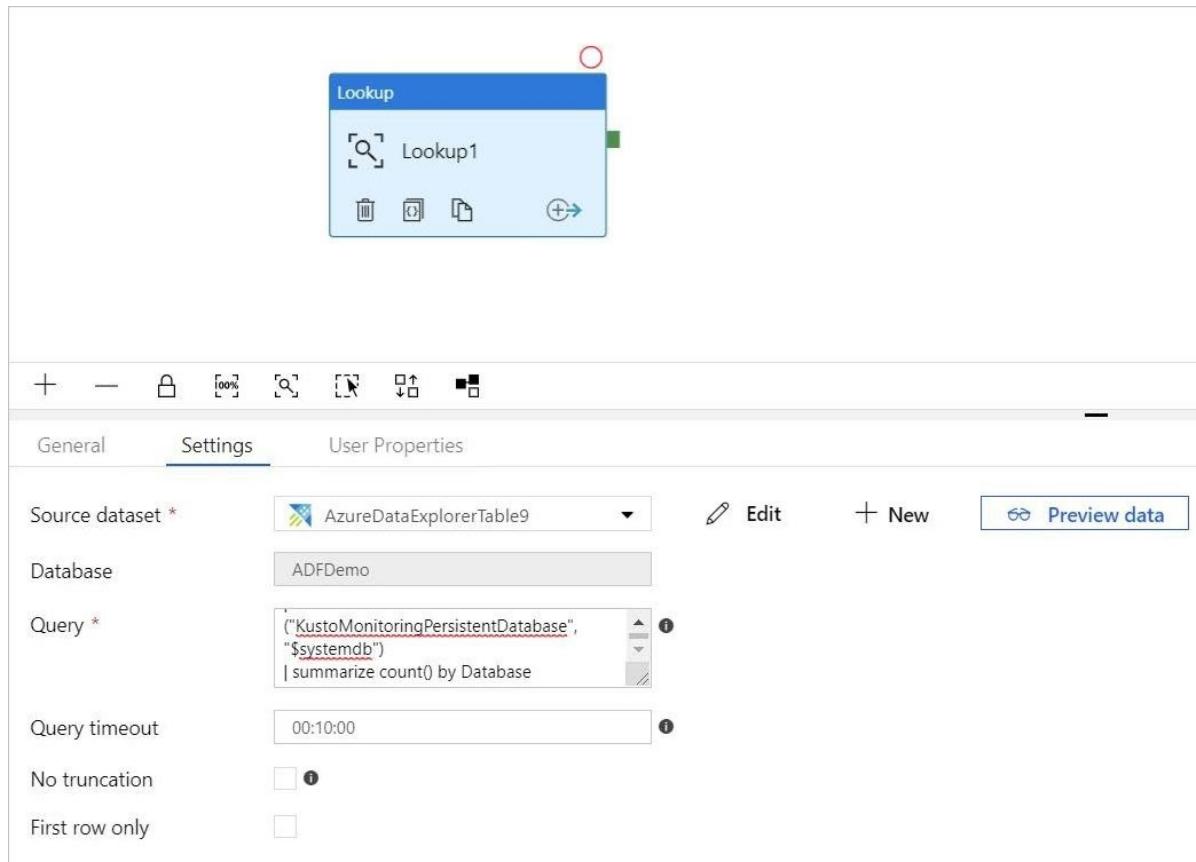
Your dataset is now ready, and you can continue editing your pipeline.

Add a query to your lookup activity

1. In **Pipeline-4-docs > Settings** add a query in **Query** text box, for example:

```
ClusterQueries  
| where Database !in ("KustoMonitoringPersistentDatabase", "$systemdb")  
| summarize count() by Database
```

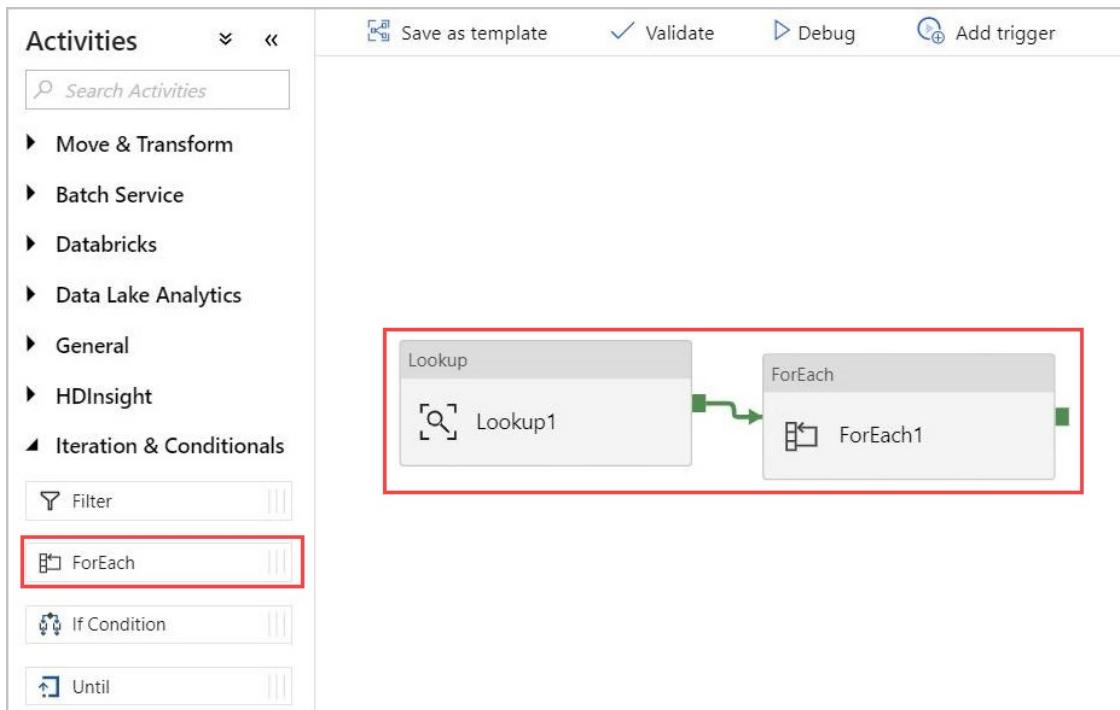
2. Change the **Query timeout** or **No truncation** and **First row only** properties, as needed. In this flow, we keep the default **Query timeout** and uncheck the checkboxes.



Create a For-Each activity

The **For-Each** activity is used to iterate over a collection and execute specified activities in a loop.

1. Now you add a For-Each activity to the pipeline. This activity will process the data returned from the Lookup activity.
- In the **Activities** pane, under **Iteration & Conditionals**, select the **ForEach** activity and drag and drop it into the canvas.
 - Draw a line between the output of the Lookup activity and the input of the ForEach activity in the canvas to connect them.



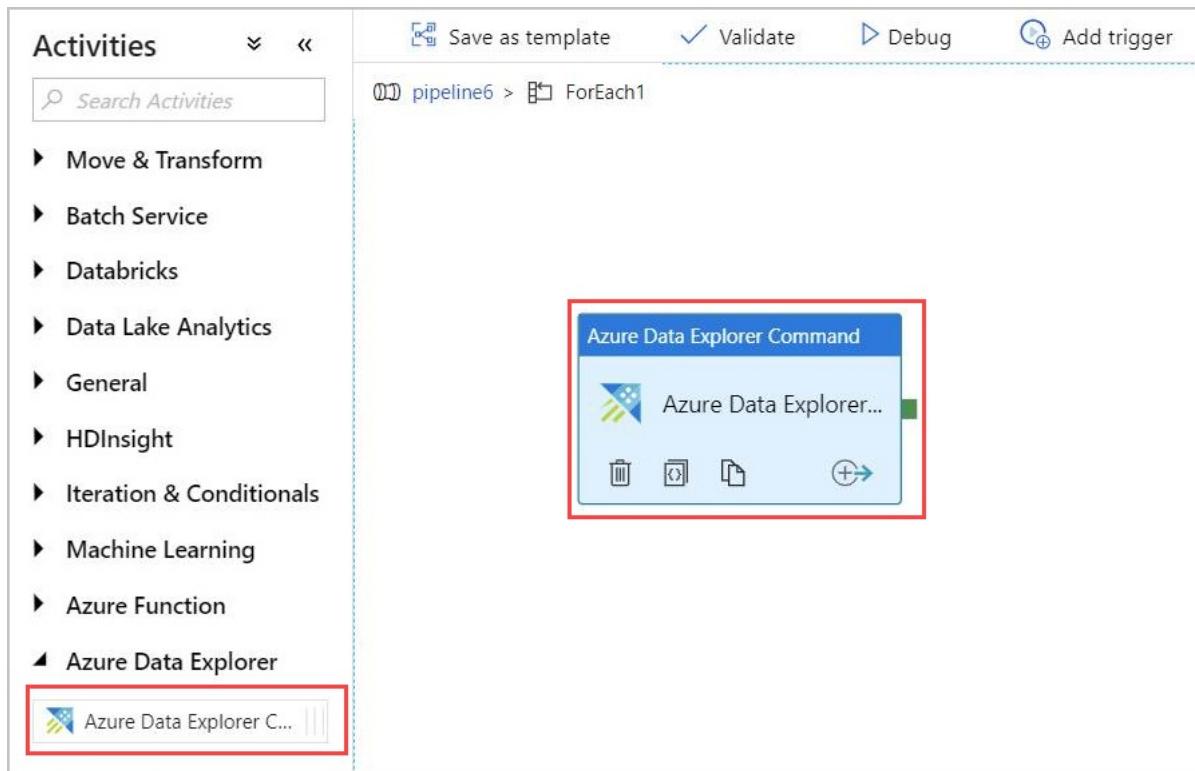
2. Select the ForEach activity in the canvas. In the **Settings** tab below:

- Check the **Sequential** checkbox for a sequential processing of the Lookup results, or leave it unchecked to create parallel processing.
- Set **Batch count**.
- In **Items**, provide the following reference to the output value: `@activity('Lookup1').output.value`

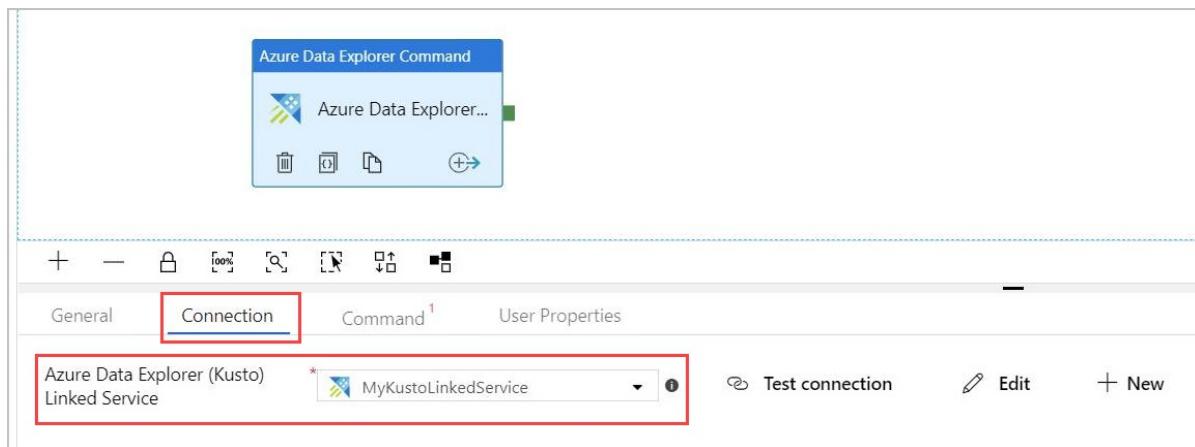


Create an Azure Data Explorer Command activity within the ForEach activity

1. Double-click the ForEach activity in the canvas to open it in a new canvas to specify the activities within ForEach.
2. In the **Activities** pane, under **Azure Data Explorer**, select the **Azure Data Explorer Command** activity and drag and drop it into the canvas.



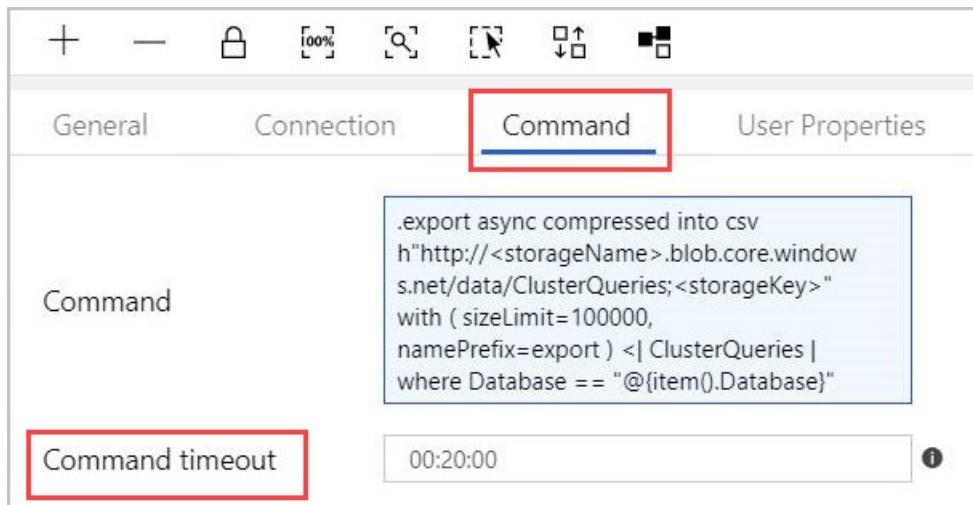
3. In the **Connection** tab, select the same Linked Service previously created.



4. In the **Command** tab, provide the following command:

```
.export
async compressed
into csv h"http://<storageName>.blob.core.windows.net/data/ClusterQueries;<storageKey>" with (
sizeLimit=100000,
namePrefix=export
)
<| ClusterQueries | where Database == "@{item().Database}"
```

The **Command** instructs Azure Data Explorer to export the results of a given query into a blob storage, in a compressed format. It runs asynchronously (using the `async` modifier). The query addresses the database column of each row in the Lookup activity result. The **Command timeout** can be left unchanged.

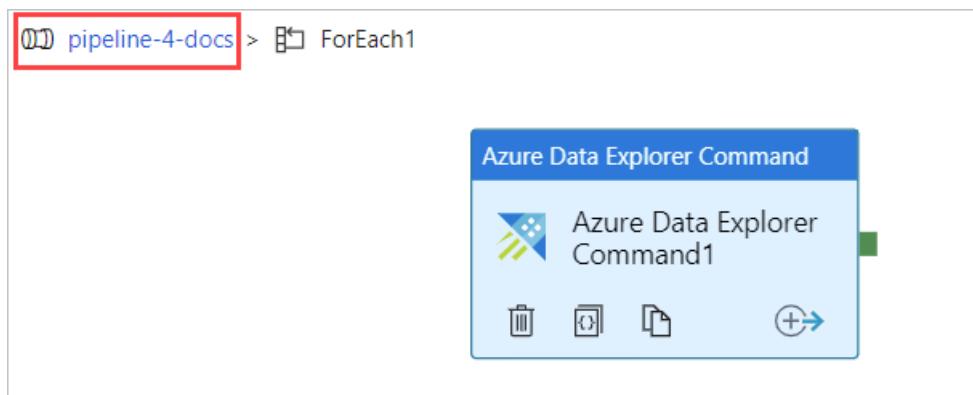


NOTE

The command activity has the following limits:

- Size limit: 1 MB response size
- Time limit: 20 minutes (default), 1 hour (maximum).
- If needed, you can append a query to the result using [AdminThenQuery](#), to reduce resulting size/time.

- Now the pipeline is ready. You can go back to the main pipeline view by clicking the pipeline name.



- Select **Debug** before publishing the pipeline. The pipeline progress can be monitored in the **Output** tab.

General	Parameters	Variables	Output
Pipeline Run ID: dbeedd93-9848-4a45-8097-49a6b24f4600			
Pipeline Progress			
NAME	TYPE	RUN START	DURATION
Azure Data Explorer Com...	AzureDataExplorerCom...	08/06/2019 12:25 PM	00:00:11
Azure Data Explorer Com...	AzureDataExplorerCom...	08/06/2019 12:25 PM	00:00:11
Azure Data Explorer Com...	AzureDataExplorerCom...	08/06/2019 12:25 PM	00:00:11
ForEach1	ForEach	08/06/2019 12:25 PM	00:00:13
Lookup1	Lookup	08/06/2019 12:25 PM	00:00:03
Actions			
Run ID			

- You can **Publish All** and then **Add trigger** to run the pipeline.

Control command outputs

The structure of the command activity output is detailed below. This output can be used by the next activity in the pipeline.

Returned value of a non-async control command

In a non-async control command, the structure of the returned value is similar to the structure of the Lookup

activity result. The `count` field indicates the number of returned records. A fixed array field `value` contains a list of records.

```
{  
  "count": "2",  
  "value": [  
    {  
      "ExtentId": "1b9977fe-e6cf-4cda-84f3-4a7c61f28ecd",  
      "ExtentSize": 1214.0,  
      "CompressedSize": 520.0  
    },  
    {  
      "ExtentId": "b897f5a3-62b0-441d-95ca-bf7a88952974",  
      "ExtentSize": 1114.0,  
      "CompressedSize": 504.0  
    }  
  ]  
}
```

Returned value of an async control command

In an async control command, the activity polls the operations table behind the scenes, until the async operation is completed or times-out. Therefore, the returned value will contain the result of `.show operations OperationId` for that given **OperationId** property. Check the values of **State** and **Status** properties, to verify successful completion of the operation.

```
{  
  "count": "1",  
  "value": [  
    {  
      "OperationId": "910deeeae-dd79-44a4-a3a2-087a90d4bb42",  
      "Operation": "TableSetOrAppend",  
      "NodeId": "",  
      "StartedOn": "2019-06-23T10:12:44.0371419Z",  
      "LastUpdatedOn": "2019-06-23T10:12:46.7871468Z",  
      "Duration": "00:00:02.7500049",  
      "State": "Completed",  
      "Status": "",  
      "RootActivityId": "f7c5aaaf-197b-4593-8ba0-e864c94c3c6f",  
      "ShouldRetry": false,  
      "Database": "MyDatabase",  
      "Principal": "<some principal id>",  
      "User": "<some User id>"  
    }  
  ]  
}
```

Next steps

- Learn about how to [copy data to Azure Data Explorer using Azure Data Factory](#).
- Learn about using [Azure Data Factory template for bulk copy from database to Azure Data Explorer](#).

Azure Data Explorer Connector for Apache Spark (Preview)

9/5/2019 • 5 minutes to read • [Edit Online](#)

Apache [Spark](#) is a unified analytics engine for large-scale data processing. Azure Data Explorer is a fast, fully managed data analytics service for real-time analysis on large volumes of data.

Azure Data Explorer connector for Spark implements data source and data sink for moving data across Azure Data Explorer and Spark clusters to use both of their capabilities. Using Azure Data Explorer and Apache Spark, you can build fast and scalable applications targeting data driven scenarios, such as machine learning (ML), Extract-Transform-Load (ETL), and Log Analytics. Writing to Azure Data Explorer can be done in batch and streaming mode. Reading from Azure Data Explorer supports column pruning and predicate pushdown, which reduces the volume of transferred data by filtering out data in Azure Data Explorer.

Azure Data Explorer Spark connector is an [open source project](#) that can run on any Spark cluster. The Azure Data Explorer Spark connector makes Azure Data Explorer a valid data store for standard Spark source and sink operations such as write, read and writeStream.

NOTE

Although some of the examples below refer to an [Azure Databricks](#) Spark cluster, Azure Data Explorer Spark connector does not take direct dependencies on Databricks or any other Spark distribution.

Prerequisites

- [Create an Azure Data Explorer cluster and database](#)
- Create a Spark cluster
- Install Azure Data Explorer connector library, and libraries listed in [dependencies](#) including the following [Kusto Java SDK](#) libraries:
 - [Kusto Data Client](#)
 - [Kusto Ingest Client](#)
- Pre-built libraries for [Spark 2.4, Scala 2.11](#)

How to build the Spark connector

Spark Connector can be built from [sources](#) as detailed below.

NOTE

This step is optional. If you are using pre-built libraries go to [Spark cluster setup](#).

Build prerequisites

- Java 1.8 SDK installed
- [Maven 3.x](#) installed
- Apache Spark version 2.4.0 or higher

TIP

2.3.x versions are also supported, but may require some changes in pom.xml dependencies.

For Scala/Java applications using Maven project definitions, link your application with the following artifact (latest version may differ):

```
<dependency>
  <groupId>com.microsoft.azure</groupId>
  <artifactId>spark-kusto-connector</artifactId>
  <version>1.0.0-Beta-02</version>
</dependency>
```

Build commands

To build jar and run all tests:

```
mvn clean package
```

To build jar, run all tests, and install jar to your local Maven repository:

```
mvn clean install
```

For more information, see [connector usage](#).

Spark cluster setup

NOTE

It is recommended to use the latest Azure Data Explorer Spark connector release when performing the following steps:

1. Set the following Spark cluster settings, based on Azure Databricks cluster using Spark 2.4 and Scala 2.11:

Microsoft Azure

Clusters / KustoDemoDS4_24

KustoDemoDS4_24

Edit Start Clone

Configuration Notebooks (0) Libraries Event Log Spark UI Driver Logs

Cluster Mode Standard

Databricks Runtime Version 5.2 (includes Apache Spark 2.4.0, Scala 2.11)

Python Version 3

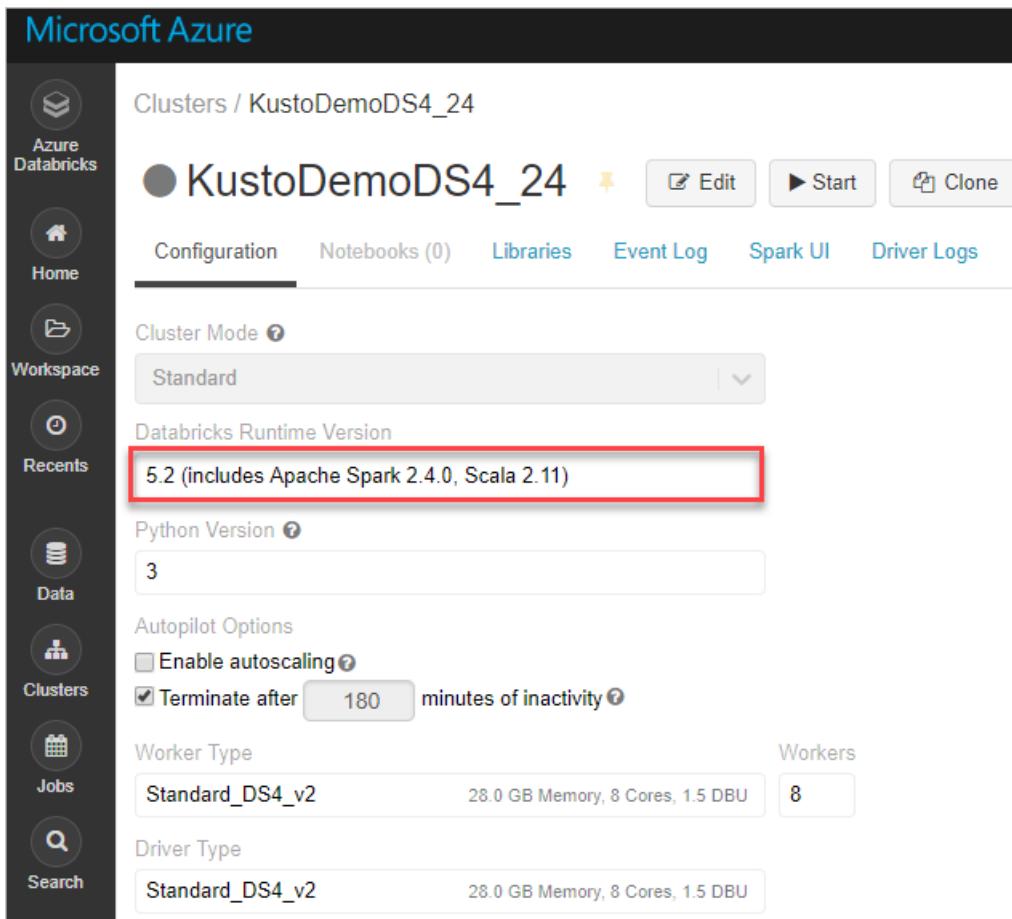
Autopilot Options

Enable autoscaling

Terminate after 180 minutes of inactivity

Worker Type Standard_DS4_v2 28.0 GB Memory, 8 Cores, 1.5 DBU Workers 8

Driver Type Standard_DS4_v2 28.0 GB Memory, 8 Cores, 1.5 DBU



2. Import the Azure Data Explorer connector library:

Microsoft Azure

Create Library

Library Source

Upload DBFS PyPI Maven CRAN

Library Type

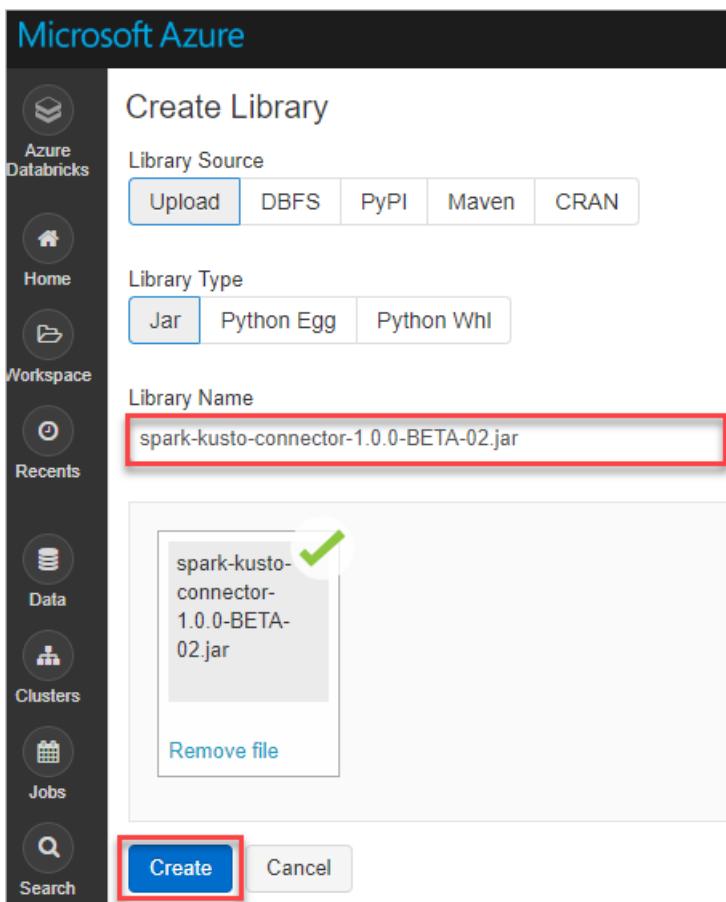
Jar Python Egg Python Whl

Library Name spark-kusto-connector-1.0.0-BETA-02.jar

spark-kusto-connector-1.0.0-BETA-02.jar

Remove file

Create Cancel



3. Add additional dependencies (not necessary if used from maven) :

The screenshot shows the 'Create Library' interface in Microsoft Azure Databricks. On the left, there's a sidebar with icons for Home, Workspace, Recents, Data, and Clusters. The main area has tabs for 'Upload', 'DBFS', 'PyPI', 'Maven' (which is highlighted with a red box), and 'CRAN'. Below these tabs is a 'Repository' dropdown set to 'Optional'. A 'Coordinates' input field contains 'Maven Coordinates (com.databricks.spark-csv_2.10:1.0.0)'. A 'Search' button is next to it. To the right is a 'Search Packages' section with a 'Maven Central' dropdown and a search bar containing 'kusto'. The results table lists several packages:

Group Id	Artifact Id	Releases	Options
com.microsoft.azure.kusto	kusto-data	1.0.0-BETA-04	Select
com.microsoft.azure.kusto	kusto-ingest	1.0.0-BETA-04	Select
com.microsoft.azure.kusto.v2019_01_21	azure-mgmt-kusto	1.0.0-beta	Select
com.microsoft.azure.kusto.v2018_09_07_preview	azure-mgmt-kusto	1.0.0-beta	Select

At the bottom left are 'Create' and 'Cancel' buttons, with 'Create' also highlighted with a red box.

TIP

The correct java release version for each Spark release is found [here](#).

4. Verify that all required libraries are installed:

The screenshot shows the 'Clusters / KustoDemoDS4_24' page in Microsoft Azure Databricks. The left sidebar includes icons for Home, Workspace, Recents, and Data. The main area shows a cluster named 'KustoDemoDS4_24' with a green status icon. Below it are tabs for Configuration, Notebooks (0), Libraries (which is selected and highlighted with a red box), Event Log, Spark UI, Driver Logs, and Spark Cluster UI - Master. Under the Libraries tab, there are buttons for 'Uninstall' and 'Install New'. A table lists installed packages:

Name	Type	Status	Source
com.microsoft.azure.kusto:kusto-data:1.0.0-BETA-04	Maven	Installed	
com.microsoft.azure.kusto:kusto-ingest:1.0.0-BETA-04	Maven	Installed	
com.microsoft.azure.azure-keyvault:1.2.1	Maven	Installed	
spark_kusto_connector_1.0.0_BETA_02.jar	JAR	Installed	dbfs:/FileStore/jars/0433e4eb_6a82_48d5_bt27_8fde4151307b-spark_kusto_connector_1.0.0_BETA_02-2312a.jar

Authentication

Azure Data Explorer Spark connector allows you to authenticate with Azure Active Directory (Azure AD) using an [Azure AD application](#), [Azure AD access token](#), [device authentication](#) (for non-production scenarios), or [Azure Key Vault](#). The user must install `azure-keyvault` package and provide application credentials to access the Key Vault resource.

Azure AD application authentication

Most simple and common authentication method. This method is recommended for Azure Data Explorer Spark connector usage.

PROPERTIES	DESCRIPTION
KUSTO_AAD_CLIENT_ID	Azure AD application (client) identifier.
KUSTO_AAD_AUTHORITY_ID	Azure AD authentication authority. Azure AD Directory (tenant) ID.
KUSTO_AAD_CLIENT_PASSWORD	Azure AD application key for the client.

Azure Data Explorer Privileges

The following privileges must be granted on an Azure Data Explorer Cluster:

- For reading (data source), Azure AD application must have *viewer* privileges on the target database, or *admin* privileges on the target table.
- For writing (data sink), Azure AD application must have *ingestor* privileges on the target database. It must also

have *user* privileges on the target database to create new tables. If the target table already exists, *admin* privileges on the target table can be configured.

For more information on Azure Data Explorer principal roles, see [role-based authorization](#). For managing security roles, see [security roles management](#).

Spark sink: Writing to Azure Data Explorer

1. Set up sink parameters:

```
val KustoSparkTestAppId = dbutils.secrets.get(scope = "KustoDemos", key = "KustoSparkTestAppId")
val KustoSparkTestAppKey = dbutils.secrets.get(scope = "KustoDemos", key = "KustoSparkTestAppKey")

val appId = KustoSparkTestAppId
val appKey = KustoSparkTestAppKey
val authorityId = "72f988bf-86f1-41af-91ab-2d7cd011db47" // Optional - defaults to microsoft.com
val cluster = "Sparktest.eastus2"
val database = "TestDb"
val table = "StringAndIntTable"
```

2. Write Spark DataFrame to Azure Data Explorer cluster as batch:

```
import com.microsoft.kusto.spark.datasink.KustoSinkOptions
val conf = Map(
    KustoSinkOptions.KUSTO_CLUSTER -> cluster,
    KustoSinkOptions.KUSTO_TABLE -> table,
    KustoSinkOptions.KUSTO_DATABASE -> database,
    KustoSinkOptions.KUSTO_AAD_CLIENT_ID -> appId,
    KustoSinkOptions.KUSTO_AAD_CLIENT_PASSWORD -> appKey,
    KustoSinkOptions.KUSTO_AAD_AUTHORITY_ID -> authorityId)

df.write
    .format("com.microsoft.kusto.spark.datasource")
    .options(conf)
    .save()
```

Or use the simplified syntax:

```
import com.microsoft.kusto.spark.datasink.SparkIngestionProperties
import com.microsoft.kusto.spark.sql.extension.SparkExtension._

val sparkIngestionProperties = Some(new SparkIngestionProperties()) // Optional, use None if not needed
df.write.kusto(cluster, database, table, conf, sparkIngestionProperties)
```

3. Write streaming data:

```

import org.apache.spark.sql.streaming.Trigger
import java.util.concurrent.TimeUnit
import java.util.concurrent.TimeUnit
import org.apache.spark.sql.streaming.Trigger

// Set up a checkpoint and disable codeGen. Set up a checkpoint and disable codeGen as a workaround for
// an known issue
spark.conf.set("spark.sql.streaming.checkpointLocation", "/FileStore/temp/checkpoint")
spark.conf.set("spark.sql_codegen.wholeStage","false") // Use in case a NullPointerException is thrown
inside codegen iterator

// Write to a Kusto table from a streaming source
val kustoQ = df
    .writeStream
    .format("com.microsoft.kusto.spark.datasink.KustoSinkProvider")
    .options(conf)
    .option(KustoSinkOptions.KUSTO_WRITE_ENABLE_ASYNC, "true") // Optional, better for streaming,
harder to handle errors
    .trigger(Trigger.ProcessingTime(TimeUnit.SECONDS.toMillis(10))) // Sync this with the
ingestionBatching policy of the database
    .start()

```

Spark source: Reading from Azure Data Explorer

- When reading small amounts of data, define the data query:

```

import com.microsoft.kusto.spark.datasource.KustoSourceOptions
import org.apache.spark.SparkConf
import org.apache.spark._
import com.microsoft.azure.kusto.data.ClientRequestProperties

val query = s"$table | where (ColB % 1000 == 0) | distinct ColA"
val conf: Map[String, String] = Map(
    KustoSourceOptions.KUSTO_AAD_CLIENT_ID -> appId,
    KustoSourceOptions.KUSTO_AAD_CLIENT_PASSWORD -> appKey
)

val df = spark.read.format("com.microsoft.kusto.spark.datasource").
    options(conf).
    option(KustoSourceOptions.KUSTO_QUERY, query).
    option(KustoSourceOptions.KUSTO_DATABASE, database).
    option(KustoSourceOptions.KUSTO_CLUSTER, cluster).
    load()

// Simplified syntax flavor
import com.microsoft.kusto.spark.sql.extension.SparkExtension._

val cpr: Option[ClientRequestProperties] = None // Optional
val df2 = spark.read.kusto(cluster, database, query, conf, cpr)
display(df2)

```

- When reading large amounts of data, transient blob storage must be provided. Provide storage container SAS key, or storage account name, account key, and container name. This step is only required for the current preview release of the Spark connector.

```

// Use either container/account-key/account name, or container SaS
val container = dbutils.secrets.get(scope = "KustoDemos", key = "blobContainer")
val storageAccountKey = dbutils.secrets.get(scope = "KustoDemos", key = "blobStorageAccountKey")
val storageAccountName = dbutils.secrets.get(scope = "KustoDemos", key = "blobStorageAccountName")
// val storageSas = dbutils.secrets.get(scope = "KustoDemos", key = "blobStorageSasUrl")

```

In the example above, we don't access the Key Vault using the connector interface. Alternatively, we use a simpler method of using the Databricks secrets.

3. Read from Azure Data Explorer:

```
val conf3 = Map(  
    KustoSourceOptions.KUSTO_AAD_CLIENT_ID -> appId,  
    KustoSourceOptions.KUSTO_AAD_CLIENT_PASSWORD -> appKey  
    KustoSourceOptions.KUSTO_BLOB_STORAGE_SAS_URL -> storageSas)  
val df2 = spark.read.kusto(cluster, database, "ReallyBigTable", conf3)  
  
val dfFiltered = df2  
    .where(df2.col("ColA").startsWith("row-2"))  
    .filter("ColB > 12")  
    .filter("ColB <= 21")  
    .select("ColA")  
  
display(dfFiltered)
```

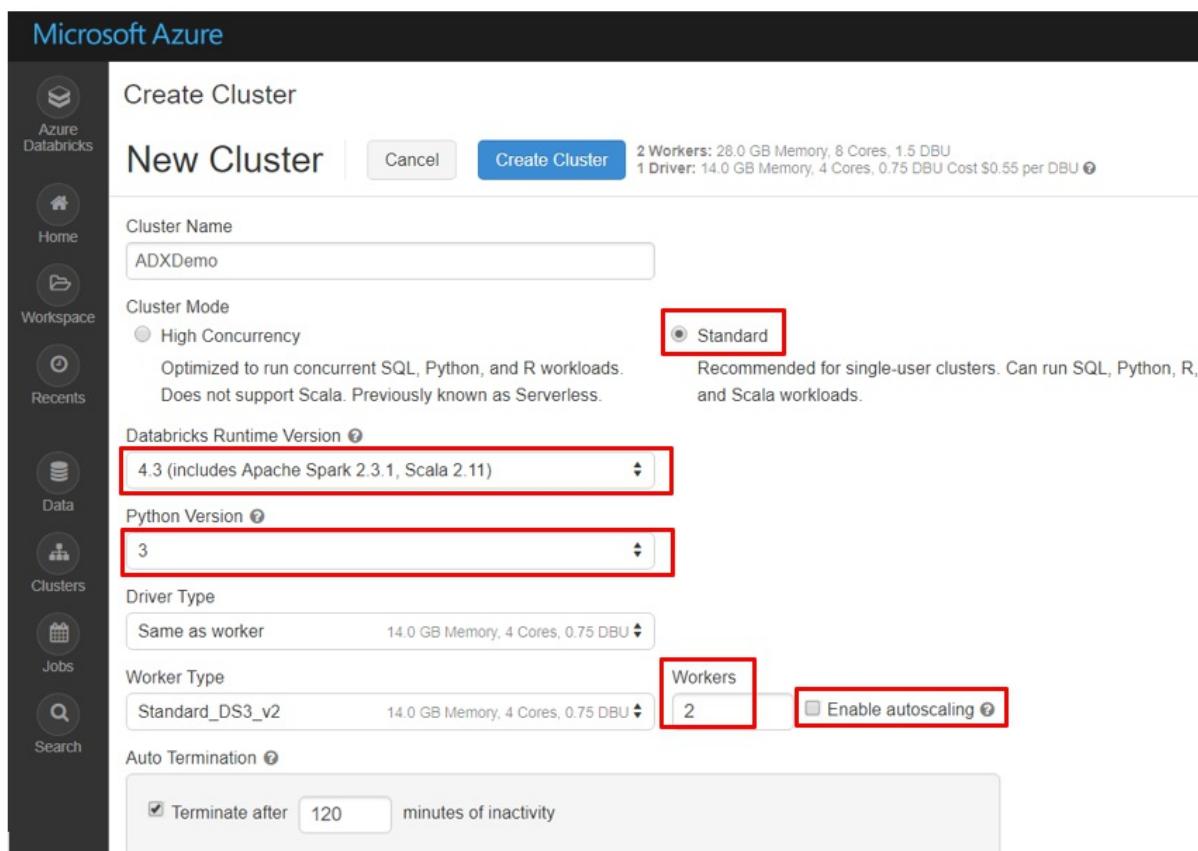
Connect to Azure Data Explorer from Azure Databricks by using Python

4/4/2019 • 2 minutes to read • [Edit Online](#)

Azure Databricks is an Apache Spark-based analytics platform that's optimized for the Microsoft Azure platform. This article shows you how to use a Python library in Azure Databricks to access data from Azure Data Explorer. There are several ways to authenticate with Azure Data Explorer, including a device login and an Azure Active Directory (Azure AD) app.

Prerequisites

- [Create an Azure Data Explorer cluster and database](#).
- [Create an Azure Databricks workspace](#). Under **Azure Databricks Service**, in the **Pricing Tier** drop-down list, select **Premium**. This selection enables you to use Azure Databricks secrets to store your credentials and reference them in notebooks and jobs.
- [Create a cluster](#) in Azure Databricks with the following specifications (minimum settings needed to run the sample notebooks):



Install the Python library on your Azure Databricks cluster

To install the [Python library](#) on your Azure Databricks cluster:

1. Go to your Azure Databricks workspace and [create a library](#).
2. [Upload a Python PyPI package or Python Egg](#).
 - Upload, install, and attach the library to your Databricks cluster.

- Enter the PyPi name: **azure-kusto-data**.

Connect to Azure Data Explorer by using a device login

Import a notebook by using the [Query-ADX-device-login](#) notebook. You can then connect to Azure Data Explorer by using your credentials.

Connect to ADX by using an Azure AD app

1. Create Azure AD app by [provisioning an Azure AD application](#).
2. Grant access to your Azure AD app in your Azure Data Explorer database as follows:

<pre>.set database <DB Name> users ('aadapp=<AAD App ID>;<AAD Tenant ID>') 'AAD App to connect Spark to ADX'</pre>	
DB Name	your database name
AAD App ID	your Azure AD app ID
AAD Tenant ID	your Azure AD tenant ID

Find your Azure AD tenant ID

To authenticate an application, Azure Data Explorer uses your Azure AD tenant ID. To find your tenant ID, use the following URL. Substitute your domain for *YourDomain*.

```
https://login.windows.net/<YourDomain>/.well-known/openid-configuration/
```

For example, if your domain is *contoso.com*, the URL is: <https://login.windows.net/contoso.com/.well-known/openid-configuration/>. Select this URL to see the results. The first line is as follows:

```
"authorization_endpoint": "https://login.windows.net/6babcaad-604b-40ac-a9d7-9fd97c0b779f/oauth2/authorize"
```

Your tenant ID is `6babcaad-604b-40ac-a9d7-9fd97c0b779f`.

Store and secure your Azure AD app ID and key

Store and secure your Azure AD app ID and key by using Azure Databricks [secrets](#) as follows:

1. [Set up the CLI](#).
2. [Install the CLI](#).
3. [Set up authentication](#).
4. Configure the [secrets](#) by using the following sample commands:

```
databricks secrets create-scope --scope adx
```

```
databricks secrets put --scope adx --key myaadappid
```

```
databricks secrets put --scope adx --key myaadappkey
```

```
databricks secrets list --scope adx
```

Import a notebook

Import a notebook by using the [Query-ADX-AAD-App](#) notebook to connect to Azure Data Explorer. Update the placeholder values with your cluster name, database name, and Azure AD tenant ID.

Use a Jupyter Notebook and Kqlmagic extension to analyze data in Azure Data Explorer

8/9/2019 • 3 minutes to read • [Edit Online](#)

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. [Jupyter Notebook](#) supports magic functions that extend the capabilities of the kernel by supporting additional commands. KQL magic is a command that extends the capabilities of the Python kernel in Jupyter Notebook so you can run Kusto language queries natively. You can easily combine Python and Kusto query language to query and visualize data using rich Plotly library integrated with `render` commands. Data sources for running queries are supported. These data sources include Azure Data Explorer, a fast and highly scalable data exploration service for log and telemetry data, as well as Azure Monitor logs and Application Insights. KQL magic also works with Azure Notebooks, Jupyter Lab, and Visual Studio Code Jupyter extension.

Prerequisites

- Organizational email account that is a member of Azure Active Directory (AAD).
- Jupyter Notebook installed on your local machine or use Azure Notebooks and clone the sample [Azure Notebook](#)

Install KQL magic library

1. Install KQL magic:

```
!pip install Kqlmagic --no-cache-dir --upgrade
```

NOTE

When using Azure Notebooks, this step is not required.

2. Load KQL magic:

```
%reload_ext Kqlmagic
```

NOTE

Change the Kernel version to Python 3.6 by clicking on Kernel > Change Kernel > Python 3.6

Connect to the Azure Data Explorer Help cluster

Use the following command to connect to the *Samples* database hosted on the *Help* cluster. For non-Microsoft AAD users, replace the tenant name `Microsoft.com` with your AAD Tenant.

```
%kql AzureDataExplorer://tenant="Microsoft.com";code;cluster='help';database='Samples'
```

Query and visualize

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. Kqlmagic supports most charts except `timepivot`, `pivotchart`, and `ladderchart`. Render is supported with all attributes except `kind`, `ysplit`, and `accumulate`.

Query and render piechart

```
%%kql
StormEvents
| summarize statecount=count() by State
| sort by statecount
| limit 10
| render piechart title="My Pie Chart by State"
```

Query and render timechart

```
%%kql
StormEvents
| summarize count() by bin(StartTime,7d)
| render timechart
```

NOTE

These charts are interactive. Select a time range to zoom into a specific time.

Customize the chart colors

If you don't like the default color palette, customize the charts using palette options. The available palettes can be found here: [Choose colors palette for your KQL magic query chart result](#)

1. For a list of palettes:

```
%kql --palettes -popup_window
```

2. Select the `cool` color palette and render the query again:

```
%%kql -palette_name "cool"
StormEvents
| summarize statecount=count() by State
| sort by statecount
| limit 10
| render piechart title="My Pie Chart by State"
```

Parameterize a query with Python

KQL magic allows for simple interchange between Kusto query language and Python. To learn more: [Parameterize your KQL magic query with Python](#)

Use a Python variable in your KQL query

You can use the value of a Python variable in your query to filter the data:

```
statefilter = ["TEXAS", "KANSAS"]
```

```
%%kql
let _state = statefilter;
StormEvents
| where State in (_state)
| summarize statecount=count() by bin(StartTime,1d), State
| render timechart title = "Trend"
```

Convert query results to Pandas DataFrame

You can access the results of a KQL query in Pandas DataFrame. Access the last executed query results by variable `_kql_raw_result_` and easily convert the results into Pandas DataFrame as follows:

```
df = _kql_raw_result_.to_dataframe()
df.head(10)
```

Example

In many analytics scenarios, you may want to create reusable notebooks that contain many queries and feed the results from one query into subsequent queries. The example below uses the Python variable `statefilter` to filter the data.

1. Run a query to view the top 10 states with maximum `DamageProperty` :

```
%%kql
StormEvents
| summarize max(DamageProperty) by State
| order by max_DamageProperty desc
| limit 10
```

2. Run a query to extract the top state and set it into a Python variable:

```
df = _kql_raw_result_.to_dataframe()
statefilter = df.loc[0].State
statefilter
```

3. Run a query using the `let` statement and the Python variable:

```
%%kql
let _state = statefilter;
StormEvents
| where State in (_state)
| summarize statecount=count() by bin(StartTime,1d), State
| render timechart title = "Trend"
```

4. Run the help command:

```
%kql --help "help"
```

TIP

To receive information about all available configurations use `%config KQLmagic`. To troubleshoot and capture Kusto errors, such as connection issues and incorrect queries, use `%config Kqlmagic.short_errors=False`

Next steps

Run the help command to explore the following sample notebooks that contain all the supported features:

- [Get started with KQL magic for Azure Data Explorer](#)
- [Get started with KQL magic for Application Insights](#)
- [Get started with KQL magic for Azure Monitor logs](#)
- [Parametrize your KQL magic query with Python](#)
- [Choose colors palette for your KQL magic query chart result](#)

Azure DevOps Task for Azure Data Explorer

5/29/2019 • 3 minutes to read • [Edit Online](#)

Azure DevOps Services provides development collaboration tools such as high-performance pipelines, free private Git repositories, configurable Kanban boards, and extensive automated and continuous testing capabilities. [Azure Pipelines](#) is an Azure DevOps capability that enables you to manage CI/CD to deploy your code with high-performance pipelines that work with any language, platform, and cloud. [Azure Data Explorer - Admin Commands](#) is the Azure Pipelines task that enables you to create release pipelines and deploy your database changes to your Azure Data Explorer databases. It's available for free in the [Visual Studio Marketplace](#).

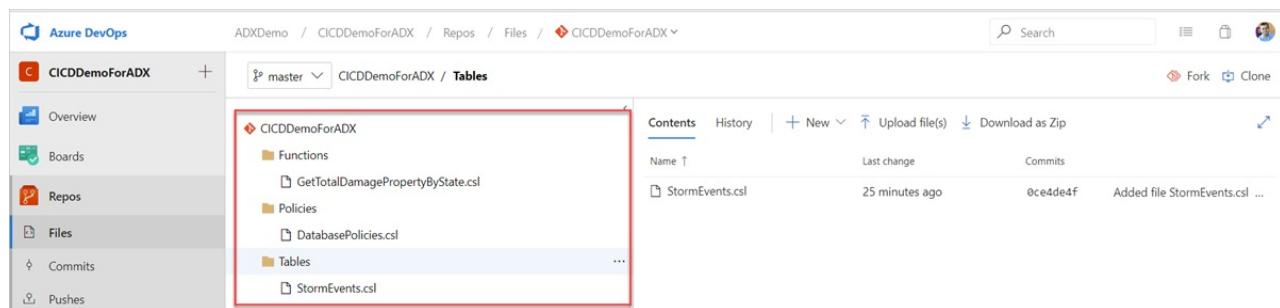
This document describes a simple example on the use of the **Azure Data Explorer – Admin Commands** task to deploy your schema changes to your database. For complete CI/CD pipelines, refer to [Azure DevOps documentation](#).

Prerequisites

- If you don't have an Azure subscription, create a [free Azure account](#) before you begin.
- Azure Data Explorer Cluster setup:
 - An [Azure Data Explorer cluster and database](#)
 - Create Azure Active Directory (Azure AD) app by [provisioning an Azure AD application](#).
 - Grant access to your Azure AD App on your Azure Data Explorer database by [managing Azure Data Explorer database permissions](#).
- Azure DevOps setup:
 - [Sign up for a free organization](#)
 - [Create an organization](#)
 - [Create a project in Azure DevOps](#)
 - [Code with Git](#)

Create folders

Create the following sample folders (*Functions*, *Policies*, *Tables*) in your Git repository. Copy the files from [here](#) into the respective folders as seen below and commit the changes. The sample files are provided to execute the following workflow.



TIP

When creating your own workflow, we recommend making your code idempotent. For example, use `.create-merge table` instead of `.create table`, and use `.create-or-alter` function instead of `.create` function.

Create a release pipeline

1. Sign in to your Azure DevOps organization.
2. Select **Pipelines > Releases** from left-hand menu and select **New pipeline**.

The screenshot shows the Azure DevOps interface. On the left, a sidebar menu has 'Pipelines' and 'Releases' highlighted with red boxes. The main area displays a message: 'No release pipelines found' with a call-to-action button 'New pipeline'.

3. The **New release pipeline** window opens. In the **Pipelines** tab, in the **Select a template** pane, select **Empty job**.

The screenshot shows the 'New release pipeline' configuration window. The 'Pipeline' tab is selected. In the 'Select a template' pane, the 'Empty job' option is highlighted with a red box. A list of featured templates is shown on the right.

Template	Description
Azure App Service deployment	Deploy your application to Azure App Service. Choose from Web App on Windows, Linux, containers, Function Apps, or WebJobs.
Deploy a Java app to Azure App Service	Deploy a Java application to an Azure Web App.
Deploy a Node.js app to Azure App Service	Deploy a Node.js application to an Azure Web App.
Deploy a PHP app to Azure App Service and Azure Database for MySQL	Deploy a PHP application to an Azure Web App and connect it to an Azure Database for MySQL.

4. Select **Stage** button. In **Stage** pane, add the **Stage name**. Select **Save** to save your pipeline.

The screenshot shows the 'New release pipeline' configuration window. The 'Stage' pane is open, showing a stage named 'Deploy to Pre-Production'. The 'Stage name' field contains the value 'Deploy to Pre-Production', which is highlighted with a red box. The 'Save' button at the top right of the configuration pane is also highlighted with a red box.

5. Select **Add an artifact** button. In the **Add an artifact** pane, select the repository where your code exists, fill out relevant information, and click **Add**. Select **Save** to save your pipeline.

6. In the **Variables** tab, select **+ Add** to create a variable for **Endpoint URL** that will be used in the task. Write the **Name** and the **Value** of the endpoint. Select **Save** to save your pipeline.

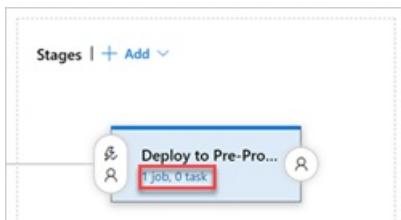
To find your **Endpoint_URL**, the overview page of your **Azure Data Explorer Cluster** in the Azure portal contains the Azure Data Explorer cluster URI. Construct the URI in the following format

`https://<Azure Data Explorer cluster URI>?DatabaseName=<DBName>`. For example,

`https://kustodocs.westus.kusto.windows.net?DatabaseName=SampleDB`

Create tasks to deploy

- In the **Pipeline** tab, click on **1 job, 0 task** to add tasks.



2. Create three tasks to deploy **Tables**, **Functions**, and **Policies**, in this order.
3. In the **Tasks** tab, select **+** by **Agent job**. Search for **Azure Data Explorer**. In **Marketplace**, install the **Azure Data Explorer – Admin Commands** extension. Then, select **Add** in **Run Azure Data Explorer Command**.

4. Click on **Kusto Command** on the left and update the task with the following information:

- **Display name:** Name of the task
- **File path:** In the **Tables** task, specify `/Tables/.csl` since the table creation files are in the `Table` folder.
- **Endpoint URL:** enter the `EndPoint URL` variable created in previous step.
- Select **Use Service Endpoint** and select **+ New**.

5. Complete the following information in the **Add Azure Data Explorer service connection** window:

SETTING	SUGGESTED VALUE
Connection name	Enter a name to identify this service endpoint

SETTING	SUGGESTED VALUE
Cluster Url	Value can be found in the overview section of your Azure Data Explorer Cluster in the Azure portal
Service Principal Id	Enter the AAD App ID (created as prerequisite)
Service Principal App Key	Enter the AAD App Key (created as prerequisite)
AAD tenant Id	Enter your AAD tenant (such as microsoft.com, contoso.com...)

Select **Allow all pipelines to use this connection** checkbox. Select **OK**.

Add Azure Data Explorer service connection

Connection name	SampleDB
Cluster Url	https://*****.westus2.kusto.windows.net
Service Principal Id	*****-*****-*****-*****
Service Principal App Key
AAD tenant Id	microsoft.com

[Learn More about ADX AAD authentication](#)

Allow all pipelines to use this connection.

OK **Close**

6. Repeat steps 1-5 another two times to deploy files from the *Functions* and *Policies* folders. Select **Save**. In the **Tasks** tab, see the three tasks created: **Deploy Tables**, **Deploy Functions**, and **Deploy Policies**.

Save + Release View releases ...

Deploy to Pre-Production

Agent job Run on agent

Deploy Tables Run Azure Data Explorer Command

Deploy Functions Run Azure Data Explorer Command

Deploy Policies Run Azure Data Explorer Command

Run Azure Data Explorer Command

Task version 1.*

Display name * Deploy Policies

Type File Path

Files **/Policies/*.sql

Single Kusto command per file

Wait for long Async Admin commands to complete (polling ".show operations <Guid>")

Wait Timeout 0

Endpoint URLs * \$(EndPoint_URL)

Use Service Endpoint

Service Endpoint SampleDB

7. Select **+ Release > Create release** to create a release.

All pipelines > New release pipeline

Save + Release View releases ...

Artifacts | + Add

Stages | + Add

Deploy to Pre-Pro... 1 job, 3 tasks

Create release

Create a draft release

8. In the **Logs** tab, check the deployment status is successful.

New release pipeline > Release-1 > Deploy to Pre-Production ✓ Succeeded

Pipeline Tasks Variables Logs Tests Deploy Cancel Refresh Download all logs Edit release ...

Deployment attempt #2 Succeeded

Agent job Succeeded

Agent job Pool: Hosted VS2017 · Agent: Hosted Agent Started: 4/18/2019, 12:09:28 AM ... 14s

Initialize job · succeeded 1s

Download Artifacts · succeeded 9s

Deploy Tables · succeeded 1s

Deploy Functions · succeeded <1s

Deploy Policies · succeeded <1s

Finalize Job · succeeded <1s

You have now completed creation of a release pipeline for deployment of three tasks to pre-production.

Troubleshoot: Failed cluster creation of Azure Data Explorer

4/4/2019 • 2 minutes to read • [Edit Online](#)

In the unlikely event that cluster creation fails in Azure Data Explorer, follow these steps.

1. Ensure you have adequate permissions. To create a cluster, you must be a member of the *Contributor* or *Owner* role for the Azure subscription. If necessary, work with your subscription administrator so they can add you to the appropriate role.
2. Ensure that there are no validation errors related to the cluster name you entered under **Create cluster** in the Azure portal.
3. Check the [Azure service health dashboard](#). Look for the status of Azure Data Explorer in the region where you're trying to create the cluster.

If the status isn't **Good** (green check mark), try creating the cluster after the status improves.

4. If you still need assistance solving your issue, please open a support request in the [Azure portal](#).

Troubleshoot: Failure to connect to a cluster in Azure Data Explorer

4/4/2019 • 2 minutes to read • [Edit Online](#)

If you're not able to connect to a cluster in Azure Data Explorer, follow these steps.

1. Ensure the connection string is correct. It should be in the form:

`https://<ClusterName>.<Region>.kusto.windows.net`, such as the following example:
`https://docscluster.westus.kusto.windows.net`.

2. Ensure you have adequate permissions. If you don't, you'll get a response of *unauthorized*.

For more information about permissions, see [Manage database permissions](#). If necessary, work with your cluster administrator so they can add you to the appropriate role.

3. Verify that the cluster hasn't been deleted: review the activity log in your subscription.
4. Check the [Azure service health dashboard](#). Look for the status of Azure Data Explorer in the region where you're trying to connect to a cluster.

If the status isn't **Good** (green check mark), try connecting to the cluster after the status improves.

5. If you still need assistance solving your issue, please open a support request in the [Azure portal](#).

Troubleshoot: Failure to create or delete a database or table in Azure Data Explorer

4/4/2019 • 2 minutes to read • [Edit Online](#)

In Azure Data Explorer, you regularly work with databases and tables. This article provides troubleshooting steps for issues that can come up.

Creating a database

1. Ensure you have adequate permissions. To create a database, you must be a member of the *Contributor* or *Owner* role for the Azure subscription. If necessary, work with your subscription administrator so they can add you to the appropriate role.
2. Ensure that there are no name validation errors for the database name. The name must be alphanumeric, with a maximum length of 260 characters.
3. Ensure that the database retention and caching values are within allowable ranges. Retention must be between 1 and 36500 days (100 years). Caching must be between 1 and the maximum value set for retention.

Deleting or renaming a database

Ensure you have adequate permissions. To delete or rename a database, you must be a member of the *Contributor* or *Owner* role for the Azure subscription. If necessary, work with your subscription administrator so they can add you to the appropriate role.

Creating a table

1. Ensure you have adequate permissions. To create a table, you must be a member of the *database admin* or *database user* role in the database or the *Contributor* or *Owner* role for the Azure subscription. If necessary, work with your subscription or cluster administrator so they can add you to the appropriate role.

For more information about permissions, see [Manage database permissions](#).

2. Ensure that a table with the same name doesn't already exist. If it exists, then you can: Create a table with a different name; rename the existing table (requires *table admin* role); or drop the existing table (requires *database admin* role). Use the following commands.

```
.drop table <TableName>  
  
.rename table <OldTableName> to <NewTableName>
```

Deleting or renaming a table

Ensure you have adequate permissions. To delete or rename a table, you must be a member of the *database admin* or *table admin* role in the database. If necessary, work with your subscription or cluster administrator so they can add you to the appropriate role.

For more information about permissions, see [Manage database permissions](#).

General guidance

1. Check the [Azure service health dashboard](#). Look for the status of Azure Data Explorer in the region where you're trying to work with a database or table.

If the status isn't **Good** (green check mark), try again after the status improves.

2. If you still need assistance solving your issue, please open a support request in the [Azure portal](#).

Next steps

[Check cluster health](#)

Prepay for Azure Data Explorer markup units with Azure Data Explorer reserved capacity

12/3/2019 • 3 minutes to read • [Edit Online](#)

Save money with Azure Data Explorer by prepaying for the markup units compared to pay-as-you-go prices. With Azure Data Explorer reserved capacity, you make a commitment for Azure Data Explorer use for a period of one or three years to get a significant discount on the Azure Data Explorer markup costs. To purchase Azure Data Explorer reserved capacity, you only need to specify the term, it will apply to all deployments of Azure Data Explorer in all regions.

By purchasing a reservation, you're pre-paying for the markup costs for a period of one or three years. As soon as you buy a reservation, the Azure Data Explorer markup charges that match the reservation attributes are no longer charged at the pay-as-you go rates. Azure Data Explorer clusters that are already running or ones that are newly deployed will automatically get the benefit. This reservation doesn't cover compute, networking, or storage charges associated with the clusters. Reserved capacity for these resources needs to be purchased separately. At the end of the reservation term, the billing benefit expires and the Azure Data Explorer markup units are billed at the pay-as-you go price. Reservations don't auto-renew. For pricing information, see the [Azure Data Explorer pricing page](#).

You can buy Azure Data Explorer reserved capacity in the [Azure portal](#). To buy Azure Data Explorer reserved capacity:

- You must be the owner of at least one Enterprise or Pay-As-You-Go subscription.
- For Enterprise subscriptions, **Add Reserved Instances** must be enabled in the [EA portal](#). Alternatively, if that setting is disabled, you must be an EA Admin on the subscription.
- For the Cloud Solution Provider (CSP) program, only the admin agents or sales agents can purchase Azure Data Explorer reserved capacity.

For details on how enterprise customers and Pay-As-You-Go customers are charged for reservation purchases, see:

- [Understand Azure reservation usage for your Enterprise enrollment](#)
- [Understand Azure reservation usage for your Pay-As-You-Go subscription](#).

Determine the right markup usage before purchase

The size of reservation should be based on the total number of Azure Data Explorer markup units used by the existing or soon-to-be-deployed Azure Data Explorer clusters. The number of markup units is equal to the number of Azure Data Explorer engine cluster cores in production (not including the dev/test SKU).

Buy Azure Data Explorer reserved capacity

1. Sign in to the [Azure portal](#).
2. Select **All services > Reservations > Purchase Now**. Select **Add**
3. In the **Select Product Type** pane, select **Azure Data Explorer** to purchase a new reservation for Azure Data Explorer markup units.
4. Select **Buy**
5. Fill in the required fields.

Select the product you want to purchase

Save on your Azure Data Explorer Markup costs with reserved capacity. The reservation discount only applies on the markup meter, other charges such as compute and storage are charged separately. After purchase, the reservation discount will automatically apply to the matching cluster. The discount applies hourly on the markup usage, unused reserved hours don't carry over. [Learn More](#)

Scope * Subscription *

Filter by name... Billing frequency : Select a value

Name	Term	Billing frequency
Azure Data Explorer	One Year	Upfront
Azure Data Explorer	One Year	Monthly
Azure Data Explorer	Three Years	Upfront
Azure Data Explorer	Three Years	Monthly

Monthly price for Data Explorer markup per core: 56.22 USD
29% Estimated savings

6. Review the cost of the Azure Data Explorer markup reserved capacity reservation in the **Costs** section.
7. Select **Purchase**.
8. Select **View this Reservation** to see the status of your purchase.

Cancellations and exchanges

If you need to cancel your Azure Data Explorer reserved capacity reservation, there may be a 12% early termination fee. Refunds are based on the lowest price of your purchase price or the current price of the reservation. Refunds are limited to \$50,000 per year. The refund you receive is the remaining pro-rated balance minus the 12% early termination fee. To request a cancellation, go to the reservation in the Azure portal and select **Refund** to create a support request.

If you need to change your Azure Data Explorer reserved capacity reservation to another term, you can exchange it for another reservation that is of equal or greater value. The term start date for the new reservation doesn't carry over from the exchanged reservation. The 1 or 3-year term starts from when you create the new reservation. To request an exchange, go to the reservation in the Azure portal, and select **Exchange** to create a support request.

For more information about how to exchange or refund reservations, see [Reservation exchanges and refunds](#).

Manage your reserved capacity reservation

The Azure Data Explorer markup units reservation discount is applied automatically to the number of markup units that match the Azure Data Explorer reserved capacity reservation scope and attributes.

NOTE

- You can update the scope of the Azure Data Explorer reserved capacity reservation through the [Azure portal](#), PowerShell, CLI or through the API.
- To learn how to manage the Azure Data Explorer reserved capacity reservation, see [manage Azure Data Explorer reserved capacity](#).

Next steps

To learn more about Azure Reservations, see the following articles:

- [What are Azure Reservations?](#)
- [Manage Azure Reservations](#)
- [Understand Azure Reservations discount](#)

- Understand reservation usage for your Pay-As-You-Go subscription
- Understand reservation usage for your Enterprise enrollment
- Azure Reservations in Partner Center Cloud Solution Provider (CSP) program

Need help? Contact us

If you have questions or need help, [create a support request](#).