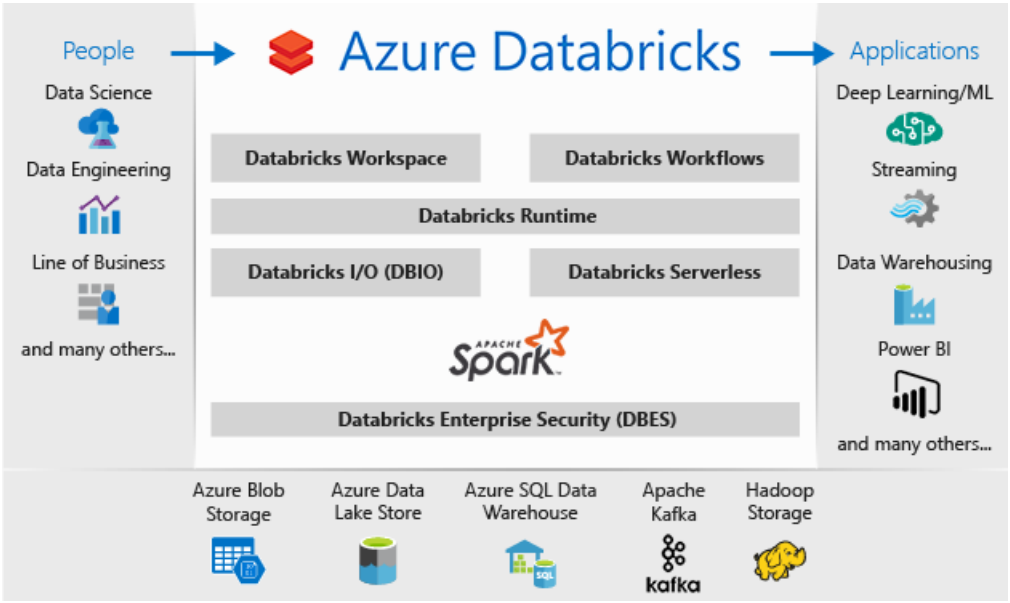
Contents

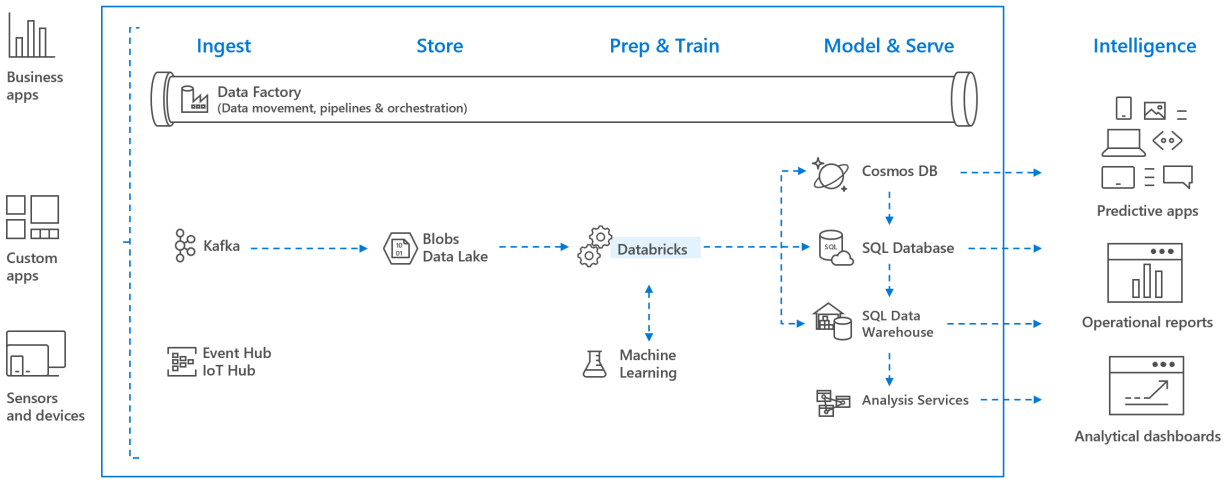
What is Azure Databricks?

12/12/2019 • 3 minutes to read • Edit Online

Azure Databricks is an Apache Spark-based analytics platform optimized for the Microsoft Azure cloud services platform. Designed with the founders of Apache Spark, Databricks is integrated with Azure to provide one-click setup, streamlined workflows, and an interactive workspace that enables collaboration between data scientists, data engineers, and business analysts.

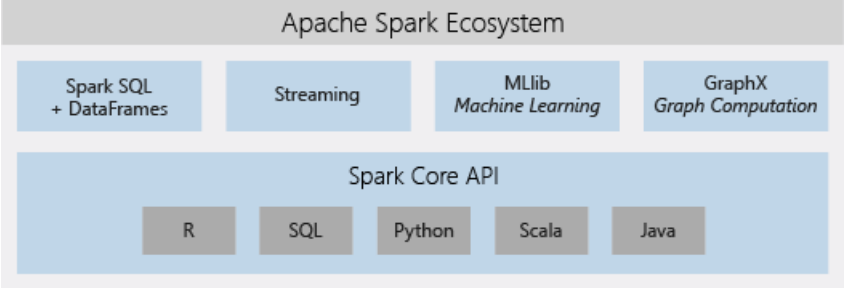


Azure Databricks is a fast, easy, and collaborative Apache Spark-based analytics service. For a big data pipeline, the data (raw or structured) is ingested into Azure through Azure Data Factory in batches, or streamed near real-time using Kafka, Event Hub, or IoT Hub. This data lands in a data lake for long term persisted storage, in Azure Blob Storage or Azure Data Lake Storage. As part of your analytics workflow, use Azure Databricks to read data from multiple data sources such as Azure Blob Storage, Azure Data Lake Storage, Azure Cosmos DB, or Azure SQL Data Warehouse and turn it into breakthrough insights using Spark.



Apache Spark-based analytics platform

Azure Databricks comprises the complete open-source Apache Spark cluster technologies and capabilities. Spark in Azure Databricks includes the following components:



- **Spark SQL and DataFrames**: Spark SQL is the Spark module for working with structured data. A DataFrame is a distributed collection of data organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python.
- **Streaming**: Real-time data processing and analysis for analytical and interactive applications. Integrates with HDFS, Flume, and Kafka.
- **MLlib**: Machine Learning library consisting of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as underlying optimization primitives.
- **GraphX**: Graphs and graph computation for a broad scope of use cases from cognitive analytics to data exploration.
- **Spark Core API**: Includes support for R, SQL, Python, Scala, and Java.

Apache Spark in Azure Databricks

Azure Databricks builds on the capabilities of Spark by providing a zero-management cloud platform that includes:

- Fully managed Spark clusters
- An interactive workspace for exploration and visualization
- A platform for powering your favorite Spark-based applications

Fully managed Apache Spark clusters in the cloud

Azure Databricks has a secure and reliable production environment in the cloud, managed and supported by Spark experts. You can:

- Create clusters in seconds.
- Dynamically autoscale clusters up and down, including serverless clusters, and share them across teams.
- Use clusters programmatically by using the REST APIs.
- Use secure data integration capabilities built on top of Spark that enable you to unify your data without centralization.
- Get instant access to the latest Apache Spark features with each release.

Databricks Runtime

The Databricks Runtime is built on top of Apache Spark and is natively built for the Azure cloud.

With the **Serverless** option, Azure Databricks completely abstracts out the infrastructure complexity and the need for specialized expertise to set up and configure your data infrastructure. The Serverless option helps data scientists iterate quickly as a team.

For data engineers, who care about the performance of production jobs, Azure Databricks provides a Spark engine that is faster and performant through various optimizations at the I/O layer and processing layer (Databricks I/O).

Workspace for collaboration

Through a collaborative and integrated environment, Azure Databricks streamlines the process of exploring data, prototyping, and running data-driven applications in Spark.

- Determine how to use data with easy data exploration.
- Document your progress in notebooks in R, Python, Scala, or SQL.
- Visualize data in a few clicks, and use familiar tools like Matplotlib, ggplot, or d3.
- Use interactive dashboards to create dynamic reports.
- Use Spark and interact with the data simultaneously.

Enterprise security

Azure Databricks provides enterprise-grade Azure security, including Azure Active Directory integration, role-based controls, and SLAs that protect your data and your business.

- Integration with Azure Active Directory enables you to run complete Azure-based solutions using Azure Databricks.
- Azure Databricks roles-based access enables fine-grained user permissions for notebooks, clusters, jobs, and data.
- Enterprise-grade SLAs.

Integration with Azure services

Azure Databricks integrates deeply with Azure databases and stores: SQL Data Warehouse, Cosmos DB, Data Lake Store, and Blob Storage.

Integration with Power BI

Through rich integration with Power BI, Azure Databricks allows you to discover and share your impactful insights quickly and easily. You can use other BI tools as well, such as Tableau Software via JDBC/ODBC cluster endpoints.

Next steps

- Quickstart: Run a Spark job on Azure Databricks
- Work with Spark clusters
- Work with notebooks
- Create Spark jobs

Quickstart: Run a Spark job on Azure Databricks using the Azure portal

12/23/2019 • 5 minutes to read • Edit Online

In this quickstart, you use the Azure portal to create an Azure Databricks workspace with an Apache Spark cluster. You run a job on the cluster and use custom charts to produce real-time reports from Boston safety data.

Prerequisites

- Azure subscription - create one for free

Sign in to the Azure portal

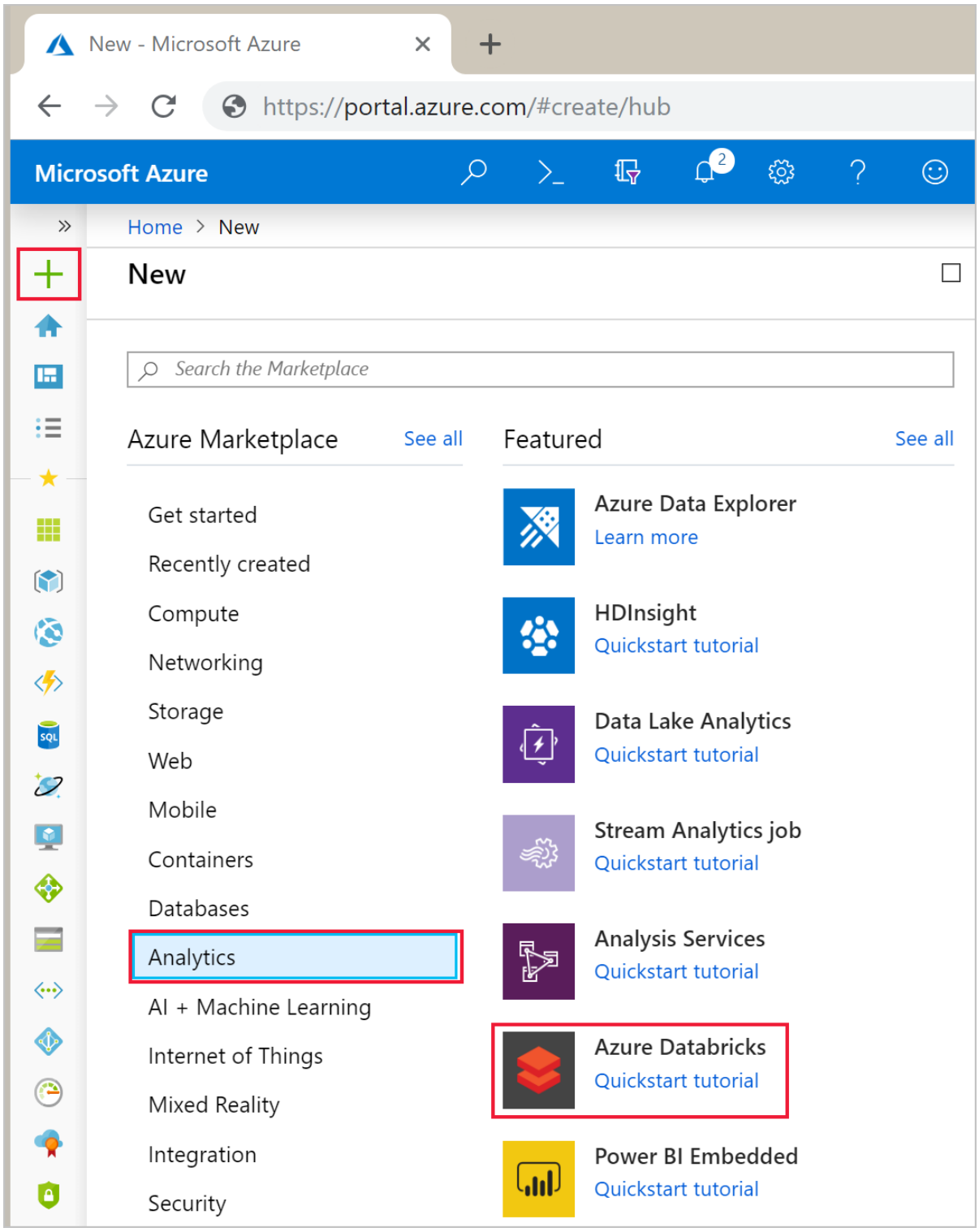
Sign in to the Azure portal.

> **NOTE**
>
> This tutorial cannot be carried out using **Azure Free Trial Subscription**. If you have a free account, go to your profile and change your subscription to **pay-as-you-go**. For more information, see Azure free account. Then, remove the spending limit, and request a quota increase for vCPUs in your region. When you create your Azure Databricks workspace, you can select the **Trial (Premium - 14-Days Free DBUs)** pricing tier to give the workspace access to free Premium Azure Databricks DBUs for 14 days.

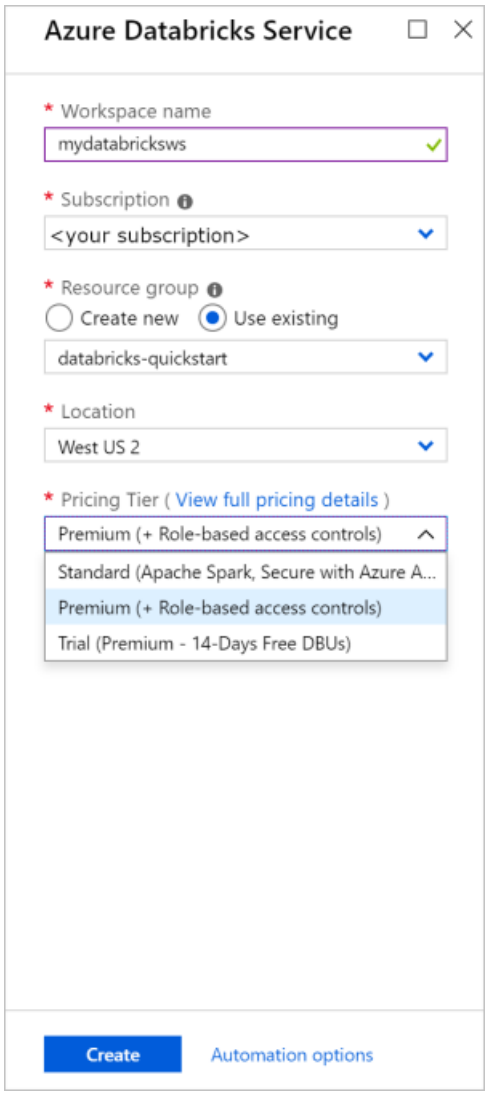
Create an Azure Databricks workspace

In this section, you create an Azure Databricks workspace using the Azure portal.

1. In the Azure portal, select **Create a resource** > **Analytics** > **Azure Databricks**.



2. Under **Azure Databricks Service**, provide the values to create a Databricks workspace.

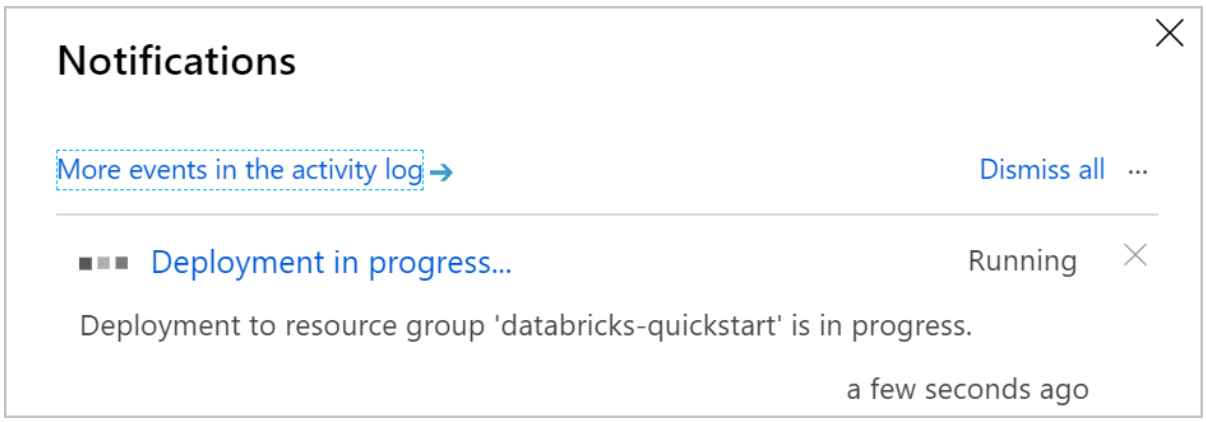


Provide the following values:

PROPERTY	DESCRIPTION
Workspace name	Provide a name for your Databricks workspace
Subscription	From the drop-down, select your Azure subscription.
Resource group	Specify whether you want to create a new resource group or use an existing one. A resource group is a container that holds related resources for an Azure solution. For more information, see Azure Resource Group overview.
Location	Select **West US 2**. For other available regions, see Azure services available by region.
Pricing Tier	Choose between **Standard**, **Premium**, or **Trial**. For more information on these tiers, see Databricks pricing page.
Virtual Network	Choose to deploy an Azure Databricks workspace in your own Virtual Network (VNet). For more information, see Deploy Azure Databricks in your Azure Virtual Network (VNet Injection).

Select **Create**.

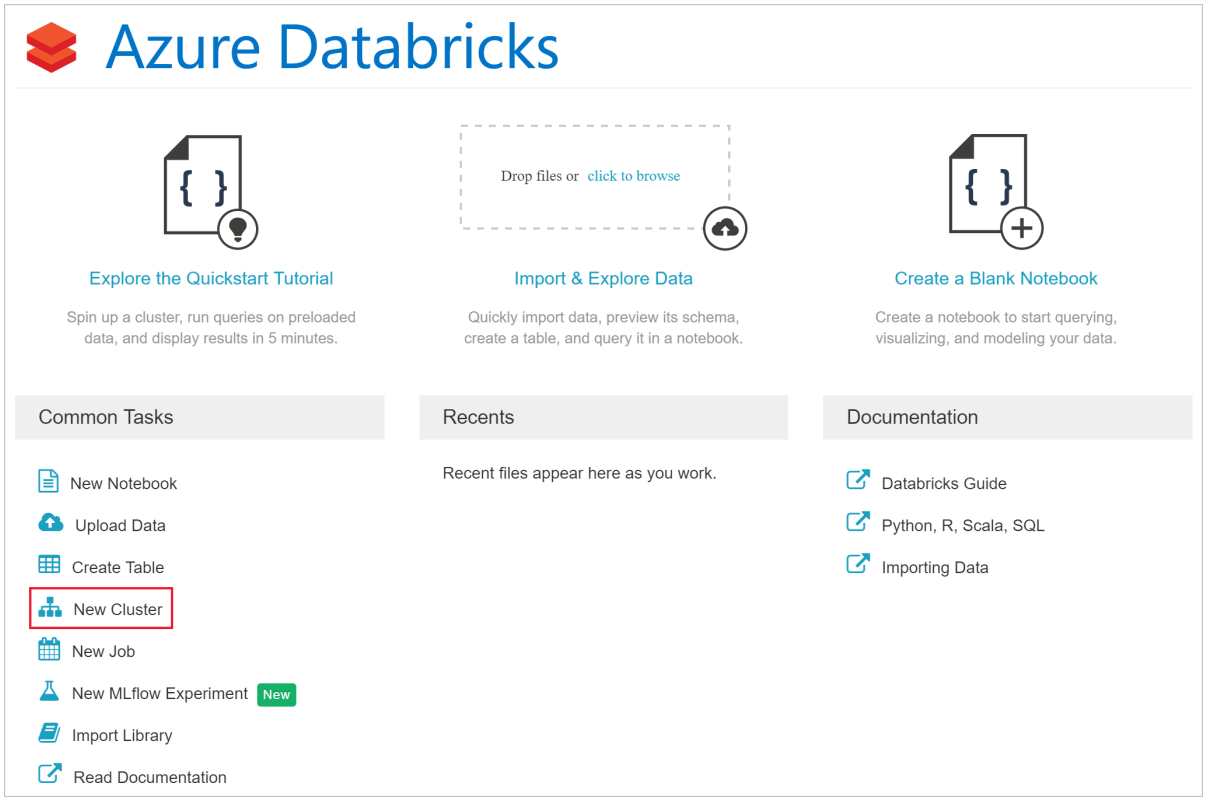
3. The workspace creation takes a few minutes. During workspace creation, you can view the deployment status in **Notifications**.



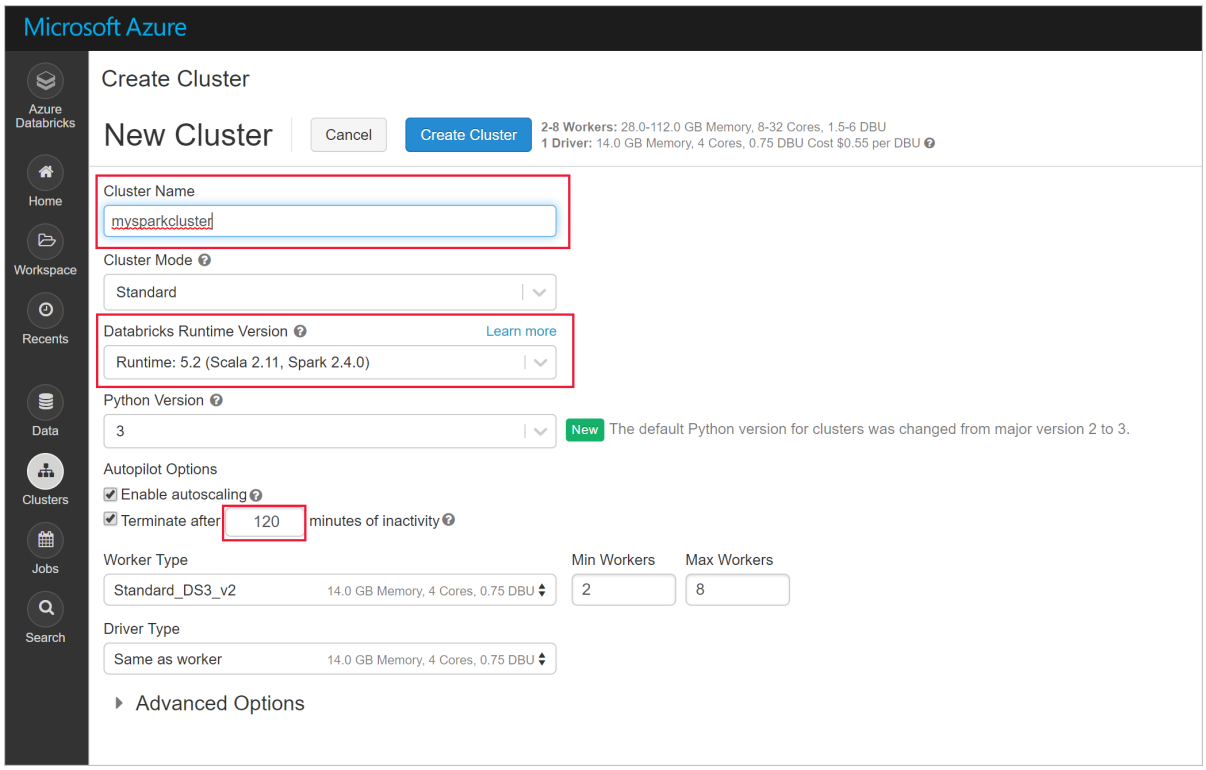
Create a Spark cluster in Databricks

> **NOTE**
> To use a free account to create the Azure Databricks cluster, before creating the cluster, go to your profile and change your subscription to **pay-as-you-go**. For more information, see Azure free account.

1. In the Azure portal, go to the Databricks workspace that you created, and then click **Launch Workspace**.
2. You are redirected to the Azure Databricks portal. From the portal, click **New Cluster**.



3. In the **New cluster** page, provide the values to create a cluster.



Accept all other default values other than the following:

- Enter a name for the cluster.
- For this article, create a cluster with **5.3** runtime.
- Make sure you select the **Terminate after __ minutes of inactivity** checkbox. Provide a duration (in minutes) to terminate the cluster, if the cluster is not being used.

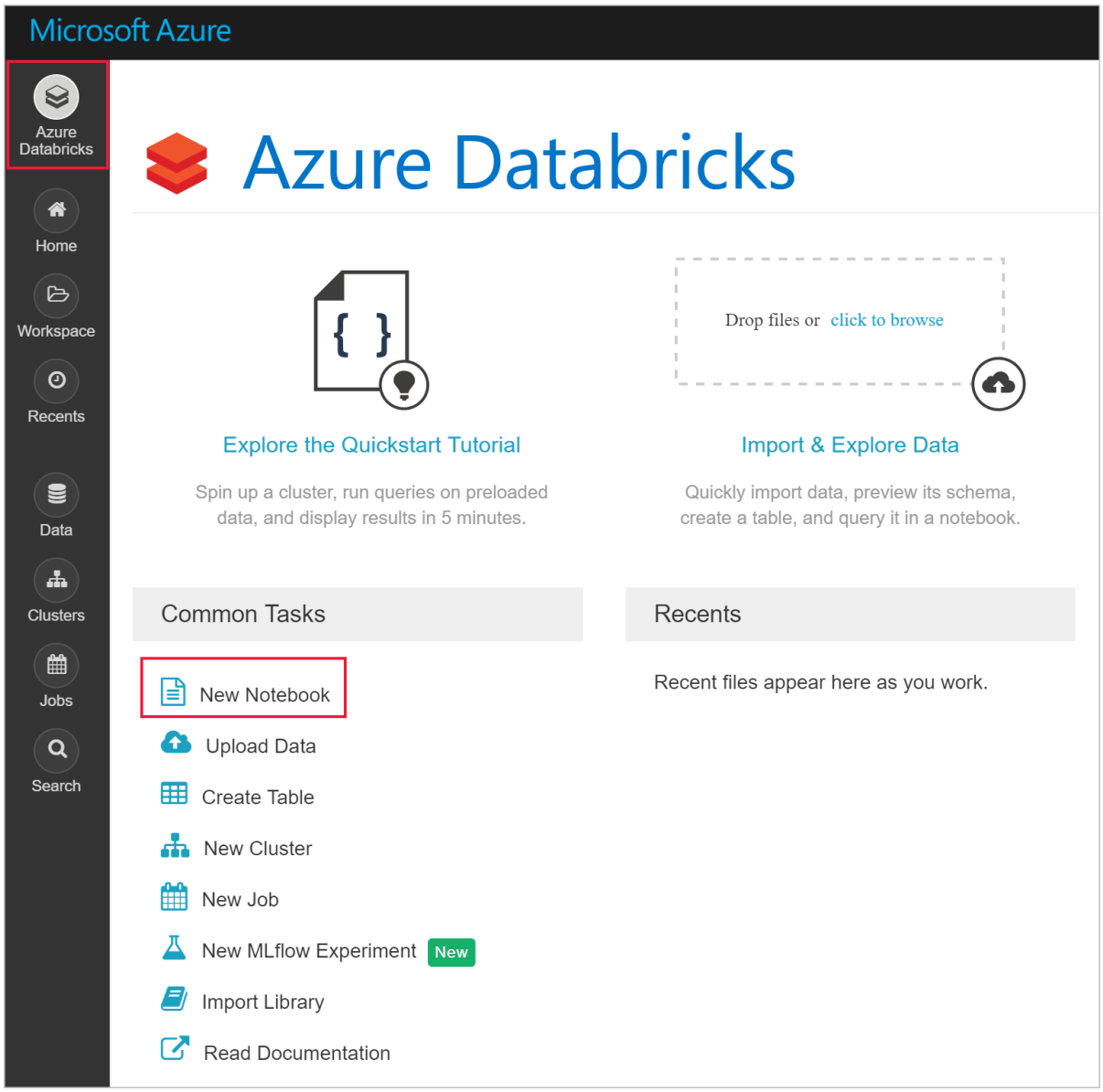
  Select **Create cluster**. Once the cluster is running, you can attach notebooks to the cluster and run Spark jobs.

For more information on creating clusters, see Create a Spark cluster in Azure Databricks.

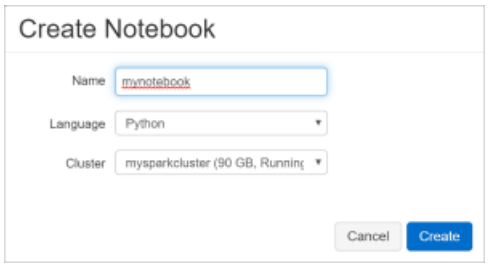
Run a Spark SQL job

Perform the following tasks to create a notebook in Databricks, configure the notebook to read data from an Azure Open Datasets, and then run a Spark SQL job on the data.

1. In the left pane, select **Azure Databricks**. From the **Common Tasks**, select **New Notebook**.



2. In the **Create Notebook** dialog box, enter a name, select **Python** as the language, and select the Spark cluster that you created earlier.



   Select **Create**.

3. In this step, create a Spark DataFrame with Boston Safety Data from Azure Open Datasets, and use SQL to query the data.

   The following command sets the Azure storage access information. Paste this PySpark code into the first cell and use **Shift+Enter** to run the code.

```
blob_account_name = "azureopendatastorage"
blob_container_name = "citydatacontainer"
blob_relative_path = "Safety/Release/city=Boston"
blob_sas_token = r"?st=2019-02-26T02%3A34%3A32Z&se=2119-02-27T02%3A34%3A00Z&sp=rl&sv=2018-03-
28&sr=c&sig=XlJVWA7fMXCSxCKqJm8psMOh0W4h7cSYO28coRqF2fs%3D"
```

The following command allows Spark to read from Blob storage remotely. Paste this PySpark code into the next cell and use **Shift+Enter** to run the code.

```
wasbs_path = 'wasbs://%s@%s.blob.core.windows.net/%s' % (blob_container_name, blob_account_name,
blob_relative_path)
spark.conf.set('fs.azure.sas.%s.%s.blob.core.windows.net' % (blob_container_name, blob_account_name),
blob_sas_token)
print('Remote blob path: ' + wasbs_path)
```

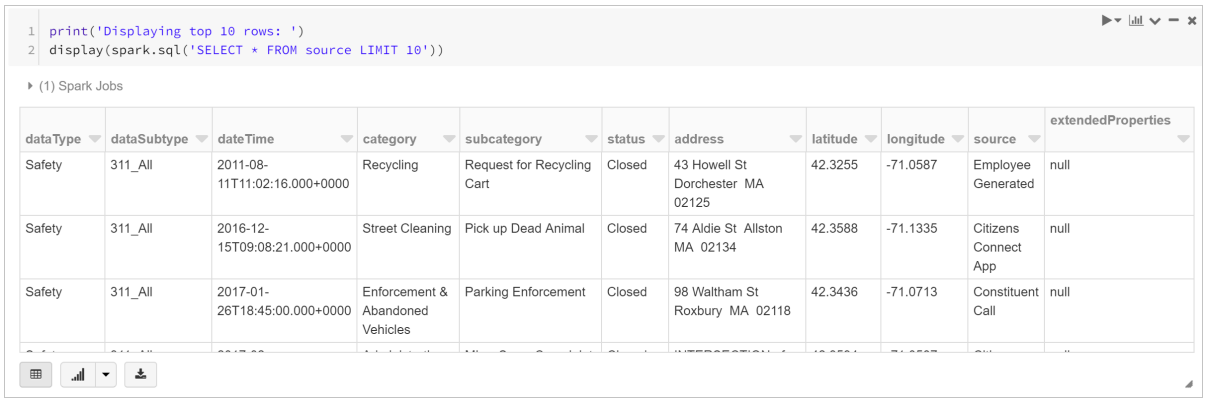
The following command creates a DataFrame. Paste this PySpark code into the next cell and use **Shift+Enter** to run the code.

```
df = spark.read.parquet(wasbs_path)
print('Register the DataFrame as a SQL temporary view: source')
df.createOrReplaceTempView('source')
```

4. Run a SQL statement return the top 10 rows of data from the temporary view called **source**. Paste this PySpark code into the next cell and use **Shift+Enter** to run the code.

```
print('Displaying top 10 rows: ')
display(spark.sql('SELECT * FROM source LIMIT 10'))
```

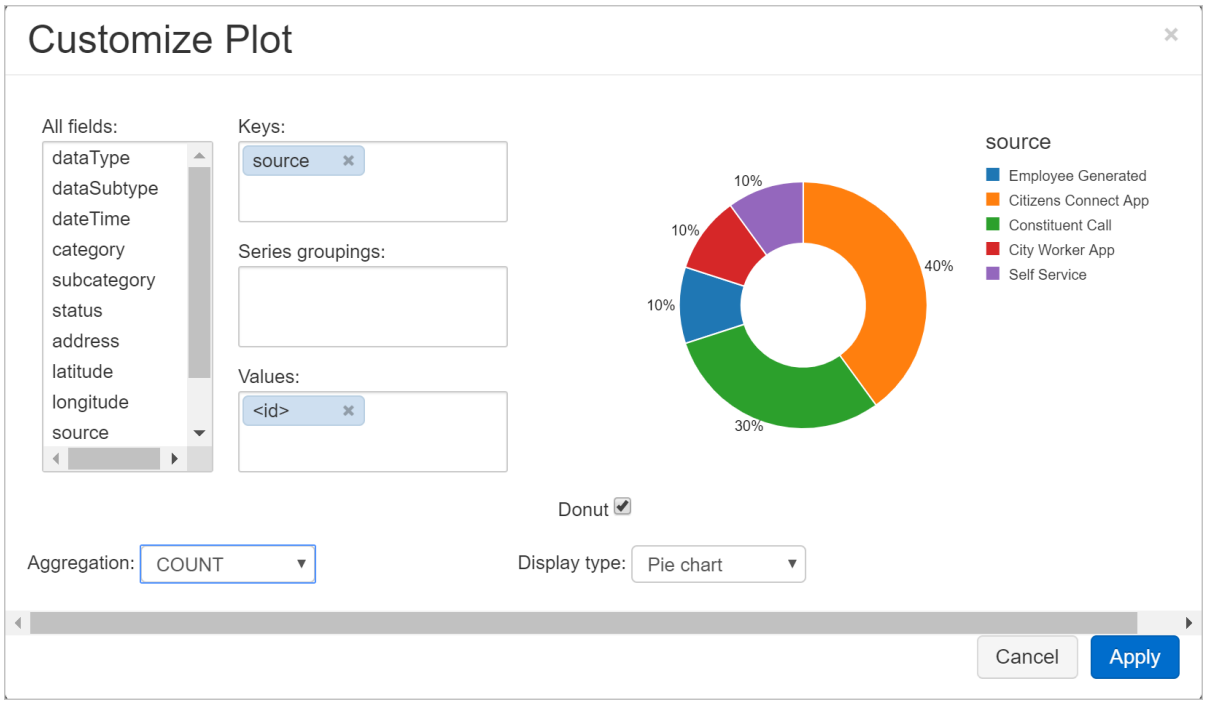
5. You see a tabular output like shown in the following screenshot (only some columns are shown):



dataType	dataSubtype	dateTime	category	subcategory	status	address	latitude	longitude	source	extendedProperties
Safety	311_All	2011-08-11T11:02:16.000+0000	Recycling	Request for Recycling Cart	Closed	43 Howell St Dorchester MA 02125	42.3255	-71.0587	Employee Generated	null
Safety	311_All	2016-12-15T09:08:21.000+0000	Street Cleaning	Pick up Dead Animal	Closed	74 Aldie St Allston MA 02134	42.3588	-71.1335	Citizens Connect App	null
Safety	311_All	2017-01-26T18:45:00.000+0000	Enforcement & Abandoned Vehicles	Parking Enforcement	Closed	98 Waltham St Roxbury MA 02118	42.3436	-71.0713	Constituent Call	null

6. You now create a visual representation of this data to show how many safety events are reported using the Citizens Connect App and City Worker App instead of other sources. From the bottom of the tabular output, select the **Bar chart** icon, and then click **Plot Options**.

7. In **Customize Plot**, drag-and-drop values as shown in the screenshot.

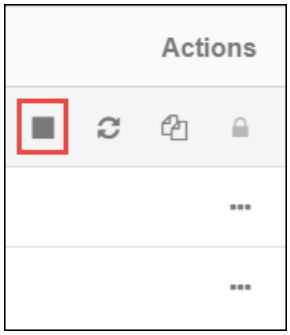


- Set **Keys** to **source**.
- Set **Values** to **<\id>**.
- Set **Aggregation** to **COUNT**.
- Set **Display type** to **Pie chart**.

  Click **Apply**.

Clean up resources

After you have finished the article, you can terminate the cluster. To do so, from the Azure Databricks workspace, from the left pane, select **Clusters**. For the cluster you want to terminate, move the cursor over the ellipsis under **Actions** column, and select the **Terminate** icon.



If you do not manually terminate the cluster it will automatically stop, provided you selected the **Terminate after __ minutes of inactivity** checkbox while creating the cluster. In such a case, the cluster automatically stops, if it has been inactive for the specified time.

Next steps

In this article, you created a Spark cluster in Azure Databricks and ran a Spark job using data from Azure Open Datasets. You can also look at Spark data sources to learn how to import data from other data sources into Azure Databricks. Advance to the next article to learn how to perform an ETL operation (extract, transform, and load data) using Azure Databricks.

Extract, transform, and load data using Azure Databricks

Quickstart: Run a Spark job on Azure Databricks using the Azure Resource Manager template

12/23/2019 • 7 minutes to read • Edit Online

In this quickstart, you use an Azure Resource Manager template to create an Azure Databricks workspace with an Apache Spark cluster. You run a job on the cluster and use custom charts to produce real-time reports from free/paid usage based on demographics.

Prerequisites

- Azure subscription - create one for free

Sign in to the Azure portal

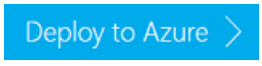
Sign in to the Azure portal.

> **NOTE**
>
> This tutorial cannot be carried out using **Azure Free Trial Subscription**. If you have a free account, go to your profile and change your subscription to **pay-as-you-go**. For more information, see Azure free account. Then, remove the spending limit, and request a quota increase for vCPUs in your region. When you create your Azure Databricks workspace, you can select the **Trial (Premium - 14-Days Free DBUs)** pricing tier to give the workspace access to free Premium Azure Databricks DBUs for 14 days.

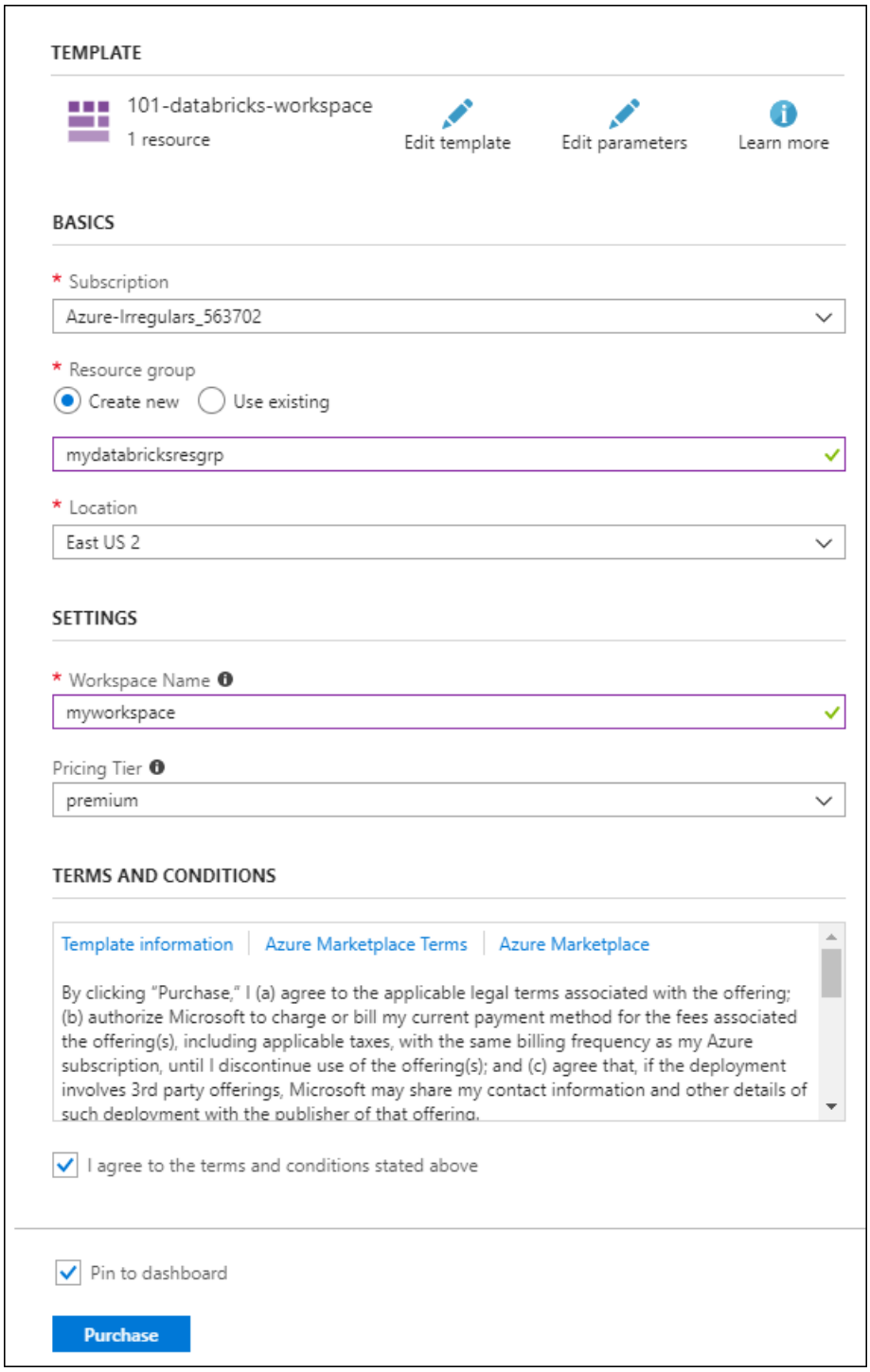
Create an Azure Databricks workspace

In this section, you create an Azure Databricks workspace using the Azure Resource Manager template.

1. Click the following image to open the template in the Azure portal.

2. Provide the required values to create your Azure Databricks workspace



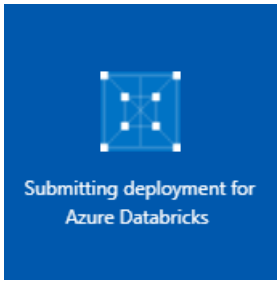
Provide the following values:

PROPERTY	DESCRIPTION
Subscription	From the drop-down, select your Azure subscription.
Resource group	Specify whether you want to create a new resource group or use an existing one. A resource group is a container that holds related resources for an Azure solution. For more information, see Azure Resource Group overview.
Location	Select **East US 2**. For other available regions, see Azure services available by region.
Workspace name	Provide a name for your Databricks workspace

PROPERTY	DESCRIPTION
Pricing Tier	Choose between **Standard** or **Premium**. For more information on these tiers, see Databricks pricing page.

3. Select **I agree to the terms and conditions stated above**, select **Pin to dashboard**, and then click **Purchase**.

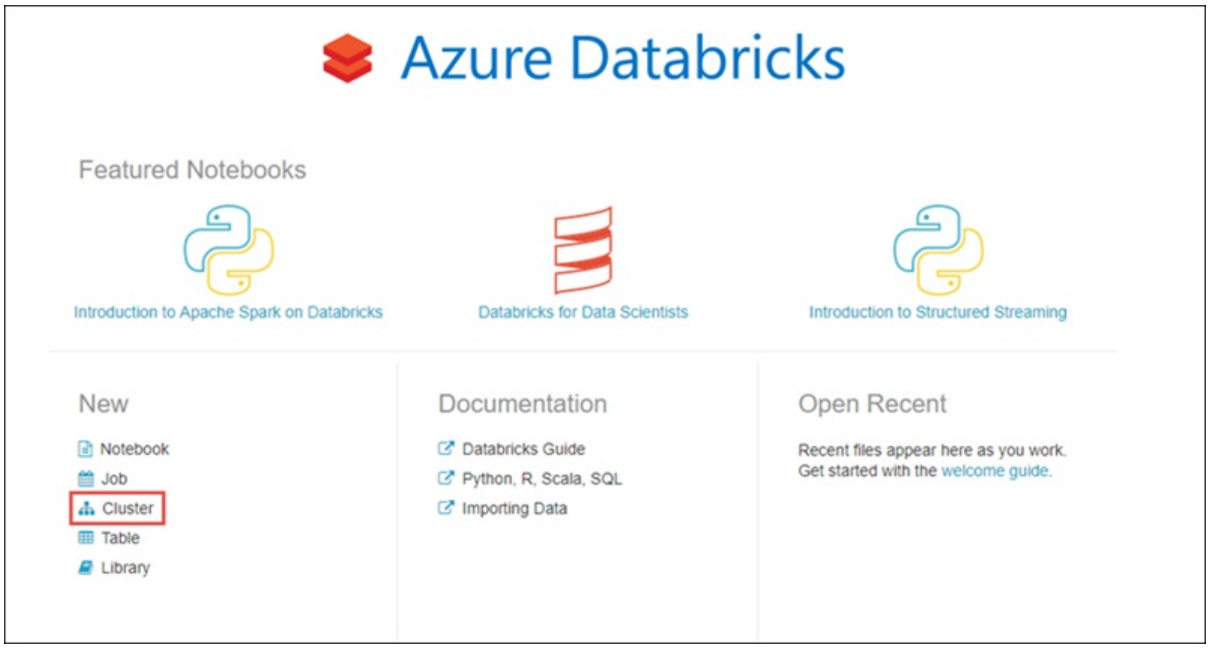
4. The workspace creation takes a few minutes. During workspace creation, the portal displays the **Submitting deployment for Azure Databricks** tile on the right side. You may need to scroll right on your dashboard to see the tile. There is also a progress bar displayed near the top of the screen. You can watch either area for progress.



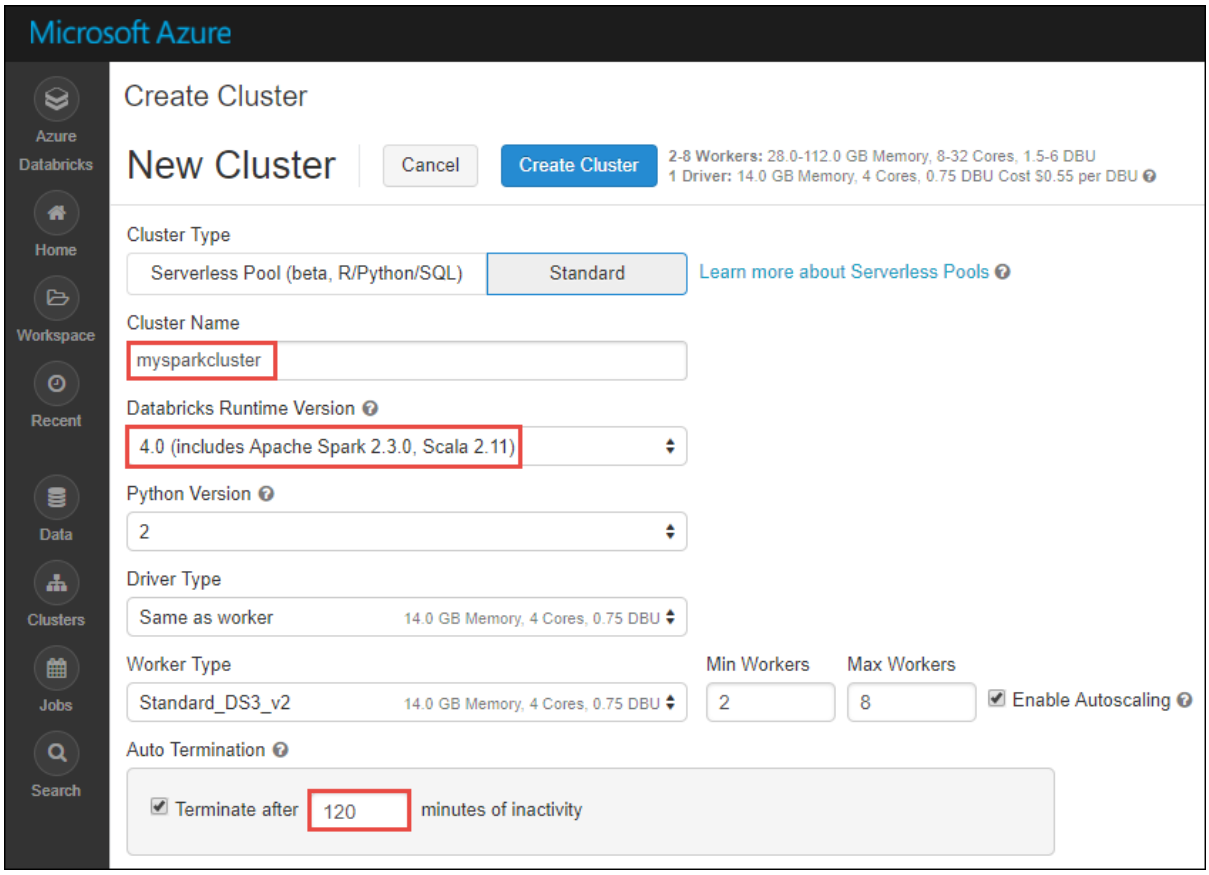
Create a Spark cluster in Databricks

1. In the Azure portal, go to the Databricks workspace that you created, and then click **Launch Workspace**.

2. You are redirected to the Azure Databricks portal. From the portal, click **Cluster**.



3. In the **New cluster** page, provide the values to create a cluster.



Accept all other default values other than the following:

- Enter a name for the cluster.
- For this article, create a cluster with **4.0** runtime.
- Make sure you select the **Terminate after __ minutes of inactivity** checkbox. Provide a duration (in minutes) to terminate the cluster, if the cluster is not being used.

Select **Create cluster**. Once the cluster is running, you can attach notebooks to the cluster and run Spark jobs.

For more information on creating clusters, see Create a Spark cluster in Azure Databricks.

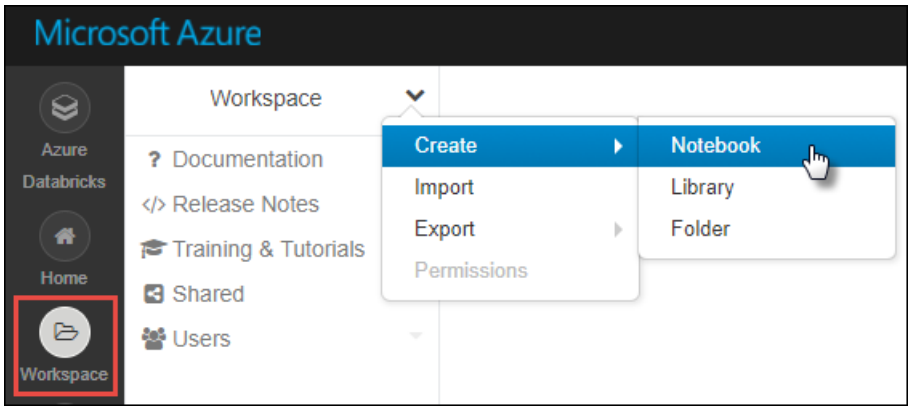
Run a Spark SQL job

Before you begin with this section, you must complete the following prerequisites:

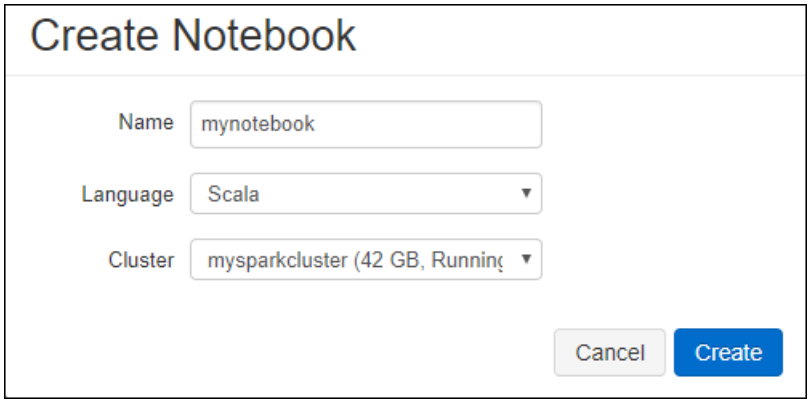
- Create an Azure Blob storage account.
- Download a sample JSON file from GitHub.
- Upload the sample JSON file to the Azure Blob storage account you created. You can use Microsoft Azure Storage Explorer to upload files.

Perform the following tasks to create a notebook in Databricks, configure the notebook to read data from an Azure Blob storage account, and then run a Spark SQL job on the data.

1. In the left pane, click **Workspace**. From the **Workspace** drop-down, click **Create**, and then click **Notebook**.



2. In the **Create Notebook** dialog box, enter a name, select **Scala** as the language, and select the Spark cluster that you created earlier.



Click **Create**.

3. In this step, associate the Azure Storage account with the Databricks Spark cluster. There are two ways to accomplish this association. You can mount the Azure Storage account to the Databricks Filesystem (DBFS), or directly access the Azure Storage account from the application you create.

> **IMPORTANT**
>
> This article uses the approach to **mount the storage with DBFS**. This approach ensures that the mounted storage gets associated with the cluster filesystem itself. Hence, any application accessing the cluster is able to use the associated storage as well. The direct-access approach is limited to the application from where you configure the access.
>
> To use the mounting approach, you must create a Spark cluster with Databricks runtime version **4.0**, which is what you chose in this article.

In the following snippet, replace `{YOUR CONTAINER NAME}`, `{YOUR STORAGE ACCOUNT NAME}`, and `{YOUR STORAGE ACCOUNT ACCESS KEY}` with the appropriate values for your Azure Storage account. Paste the snippet in an empty cell in the notebook and then press SHIFT + ENTER to run the code cell.

- **Mount the storage account with DBFS (recommended)**. In this snippet, the Azure Storage account path is mounted to `/mnt/mypath`. So, in all future occurrences where you access the Azure Storage account you don't need to give the full path. You can just use `/mnt/mypath`.

```
dbutils.fs.mount(
  source = "wasbs://{YOUR CONTAINER NAME}@{YOUR STORAGE ACCOUNT NAME}.blob.core.windows.net/",
  mountPoint = "/mnt/mypath",
  extraConfigs = Map("fs.azure.account.key.{YOUR STORAGE ACCOUNT NAME}.blob.core.windows.net" ->
"{YOUR STORAGE ACCOUNT ACCESS KEY}"))
```

- **Directly access the storage account**

  ```
  spark.conf.set("fs.azure.account.key.{YOUR STORAGE ACCOUNT NAME}.blob.core.windows.net", "{YOUR STORAGE ACCOUNT ACCESS KEY}")
  ```

  For information about how to retrieve the storage account access keys, see Manage storage account access keys.

> **NOTE**
> You can also use Azure Data Lake Store with a Spark cluster on Azure Databricks. For instructions, see Use Data Lake Store with Azure Databricks.

4. Run a SQL statement to create a temporary table using data from the sample JSON data file, **small_radio_json.json**. In the following snippet, replace the placeholder values with your container name and storage account name. Paste the snippet in a code cell in the notebook, and then press SHIFT + ENTER. In the snippet, `path` denotes the location of the sample JSON file that you uploaded to your Azure Storage account.

   ```
   %sql
   DROP TABLE IF EXISTS radio_sample_data;
   CREATE TABLE radio_sample_data
   USING json
   OPTIONS (
    path "/mnt/mypath/small_radio_json.json"
   )
   ```

   Once the command successfully completes, you have all the data from the JSON file as a table in Databricks cluster.

   The `%sql` language magic command enables you to run a SQL code from the notebook, even if the notebook is of another type. For more information, see Mixing languages in a notebook.

5. Let's look at a snapshot of the sample JSON data to better understand the query that you run. Paste the following snippet in the code cell and press **SHIFT + ENTER**.

   ```
   %sql
   SELECT * from radio_sample_data
   ```

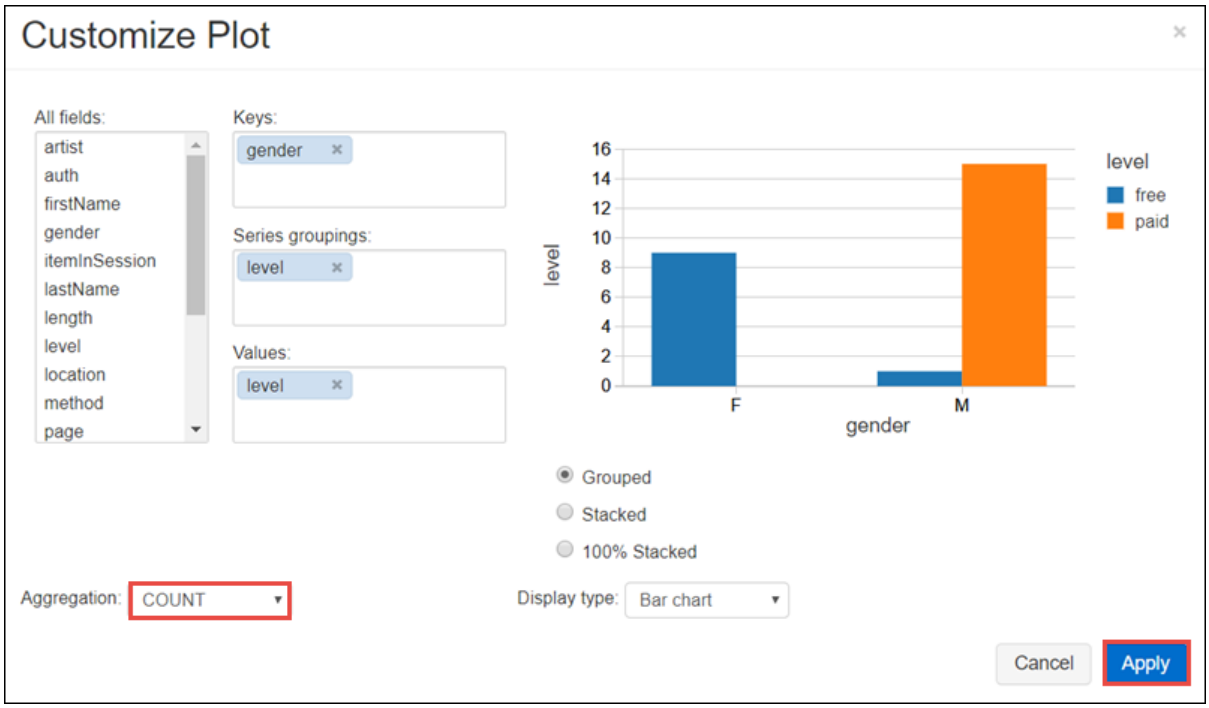
6. You see a tabular output like shown in the following screenshot (only some columns are shown):

artist	auth	firstName	gender	itemInSession	lastName	length	level
El Arrebato	Logged In	Annalyse	F	2	Montgomery	234.57914	free
Creedence Clearwater Revival	Logged In	Dylann	M	9	Thomas	340.87138	paid
Gorillaz	Logged In	Liam	M	11	Watts	246.17751	paid
null	Logged In	Tess	F	0	Townsend	null	free
Otis Redding	Logged In	Margaux	F	2	Smith	135.57506	free

Among other details, the sample data captures the gender of the audience of a radio channel (column name, **gender**) and whether their subscription is free or paid (column name, **level**).

7. You now create a visual representation of this data to show for each gender, how many users have free accounts and how many are paid subscribers. From the bottom of the tabular output, click the **Bar chart** icon, and then click **Plot Options**.

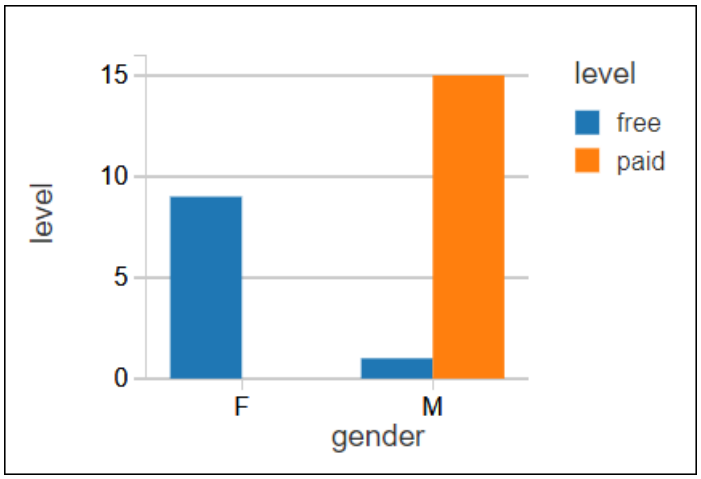
8. In **Customize Plot**, drag-and-drop values as shown in the screenshot.



- Set **Keys** to **gender**.
- Set **Series groupings** to **level**.
- Set **Values** to **level**.
- Set **Aggregation** to **COUNT**.

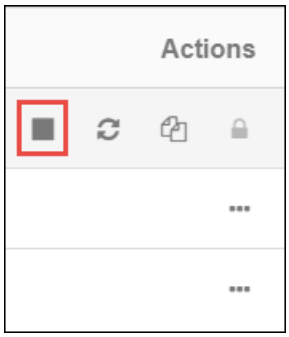
Click **Apply**.

9. The output shows the visual representation as depicted in the following screenshot:



Clean up resources

After you have finished the article, you can terminate the cluster. To do so, from the Azure Databricks workspace, from the left pane, select **Clusters**. For the cluster you want to terminate, move the cursor over the ellipsis under **Actions** column, and select the **Terminate** icon.



If you do not manually terminate the cluster it will automatically stop, provided you selected the **Terminate after __ minutes of inactivity** checkbox while creating the cluster. In such a case, the cluster automatically stops, if it has been inactive for the specified time.

Next steps

In this article, you created a Spark cluster in Azure Databricks and ran a Spark job using data in Azure storage. You can also look at Spark data sources to learn how to import data from other data sources into Azure Databricks. You can also look at the Resource Manager template to Create an Azure Databricks workspace with custom VNET address. For the JSON syntax and properties to use in a template, see Microsoft.Databricks/workspaces template reference.

Advance to the next article to learn how to perform an ETL operation (extract, transform, and load data) using Azure Databricks.

Extract, transform, and load data using Azure Databricks

Quickstart: Create an Azure Databricks workspace in your own Virtual Network

12/6/2019 • 4 minutes to read • Edit Online

The default deployment of Azure Databricks creates a new virtual network that is managed by Databricks. This quickstart shows how to create an Azure Databricks workspace in your own virtual network instead. You also create an Apache Spark cluster within that workspace.

For more information about why you might choose to create an Azure Databricks workspace in your own virtual network, see [Deploy Azure Databricks in your Azure Virtual Network (VNet Injection)](/databricks/administration-guide/cloud-configurations/azure/vnet-inject).

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

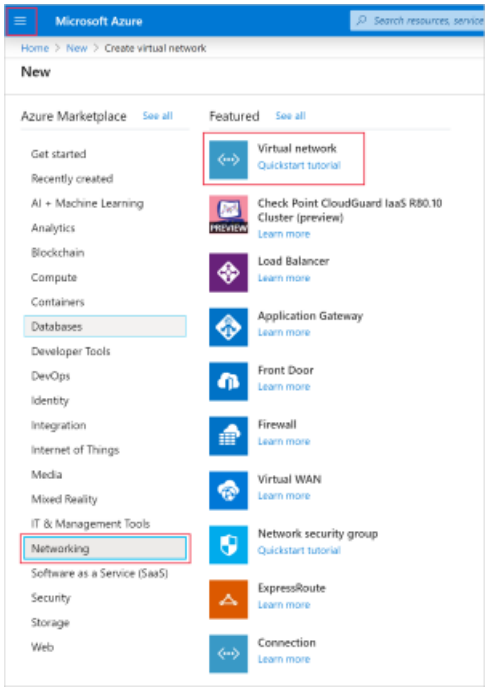
Sign in to the Azure portal

Sign in to the Azure portal.

> **NOTE**
>
> This tutorial cannot be carried out using **Azure Free Trial Subscription**. If you have a free account, go to your profile and change your subscription to **pay-as-you-go**. For more information, see Azure free account. Then, remove the spending limit, and request a quota increase for vCPUs in your region. When you create your Azure Databricks workspace, you can select the **Trial (Premium - 14-Days Free DBUs)** pricing tier to give the workspace access to free Premium Azure Databricks DBUs for 14 days.

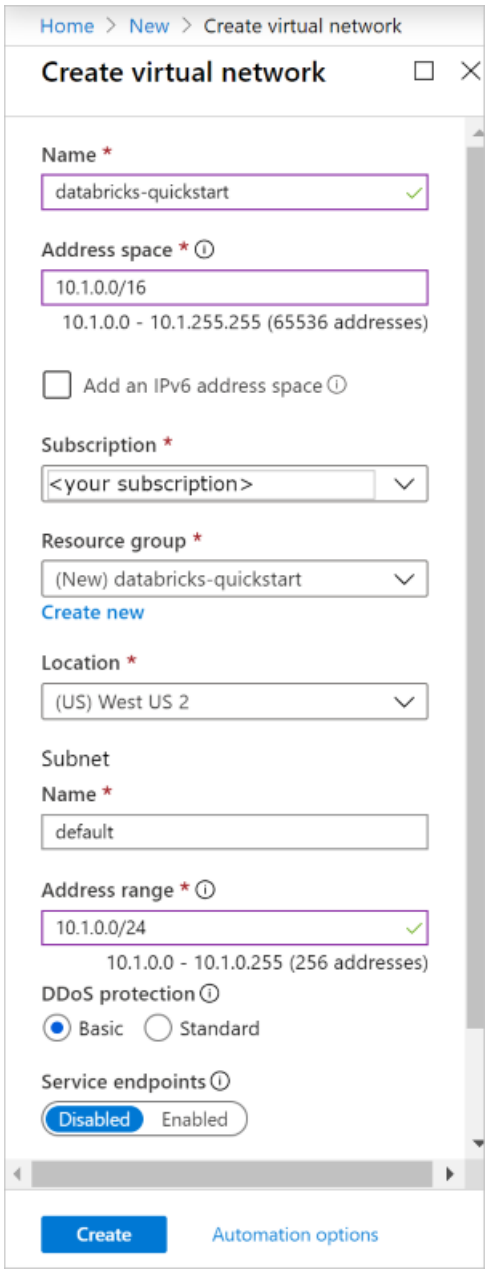
Create a virtual network

1. From the Azure portal menu, select **Create a resource**. Then select **Networking > Virtual network**.

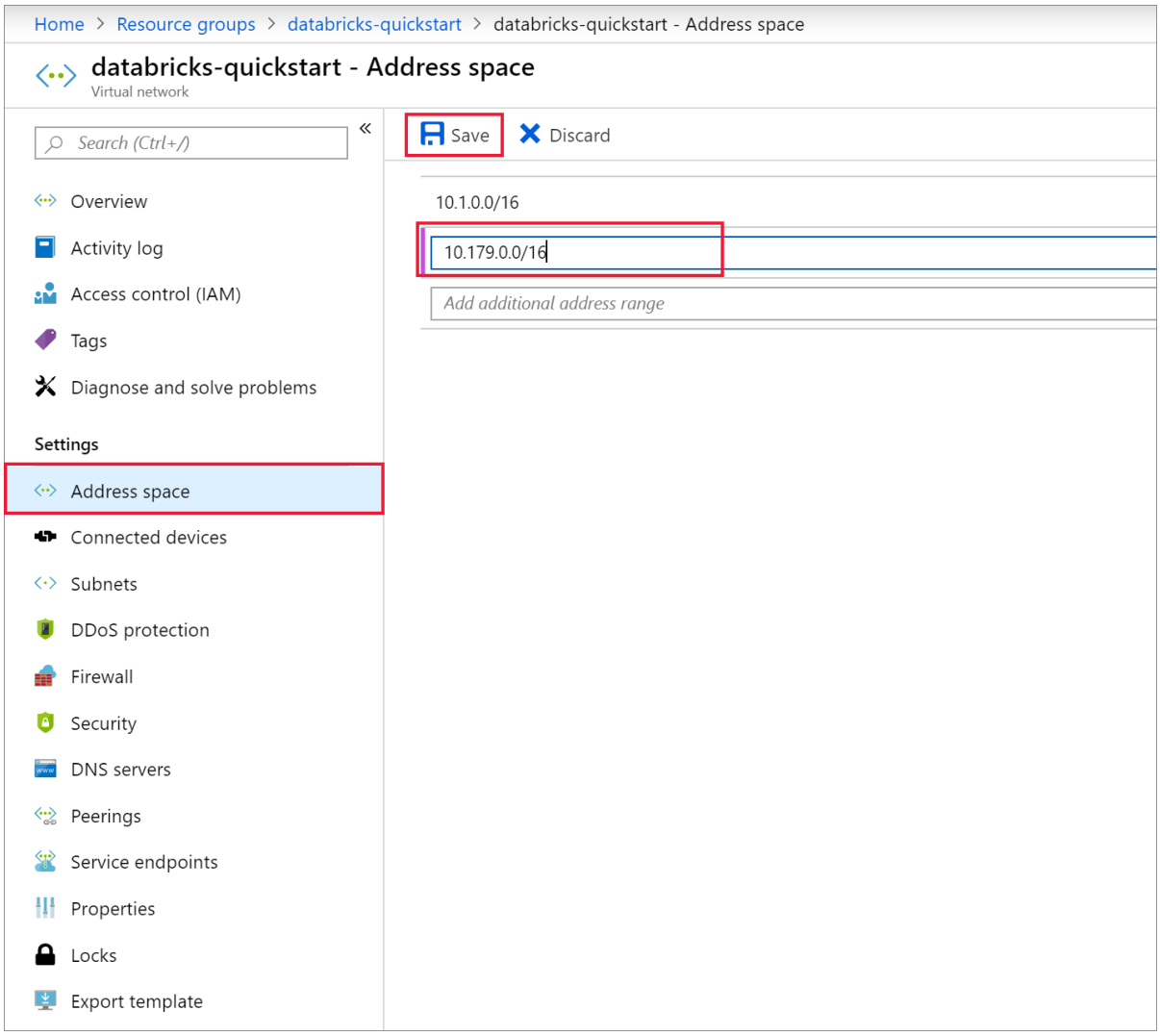


2. Under **Create virtual network**, apply the following settings:

SETTING	SUGGESTED VALUE	DESCRIPTION
Name	databricks-quickstart	Select a name for your virtual network.
Address space	10.1.0.0/16	The virtual network's address range in CIDR notation. The CIDR range must be between /16 and /24
Subscription	<Your subscription>	Select the Azure subscription that you want to use.
Resource group	databricks-quickstart	Select **Create New** and enter a new resource group name for your account.
Location	<Select the region that is closest to your users>	Select a geographic location where you can host your virtual network. Use the location that's closest to your users.
Subnet name	default	Select a name for the default subnet in your virtual network.
Subnet Address range	10.1.0.0/24	The subnet's address range in CIDR notation. It must be contained by the address space of the virtual network. The address range of a subnet which is in use can't be edited.

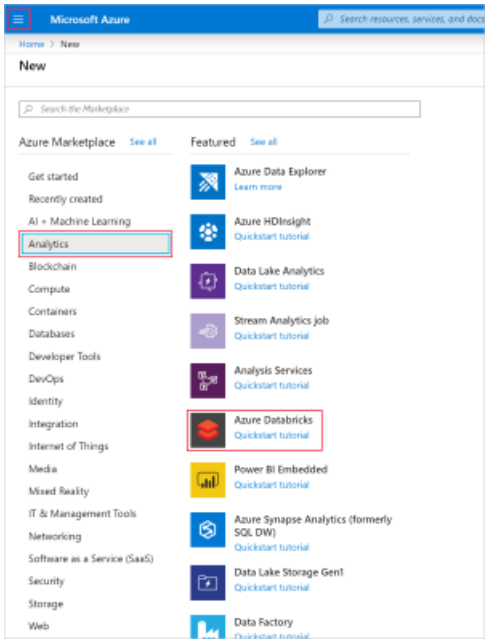


3. Once the deployment is complete, navigate to your virtual network and select **Address space** under **Settings**. In the box that says *Add additional address range*, insert `10.179.0.0/16` and select **Save**.



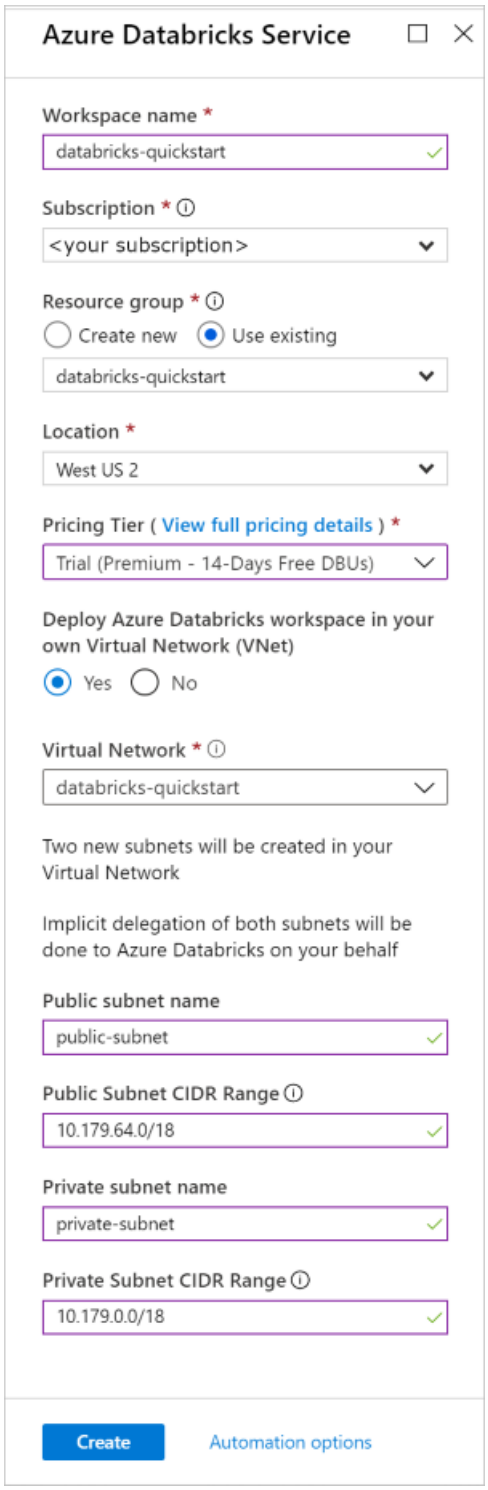
Create an Azure Databricks workspace

1. From the Azure portal menu, select **Create a resource**. Then select **Analytics** > **Databricks**.

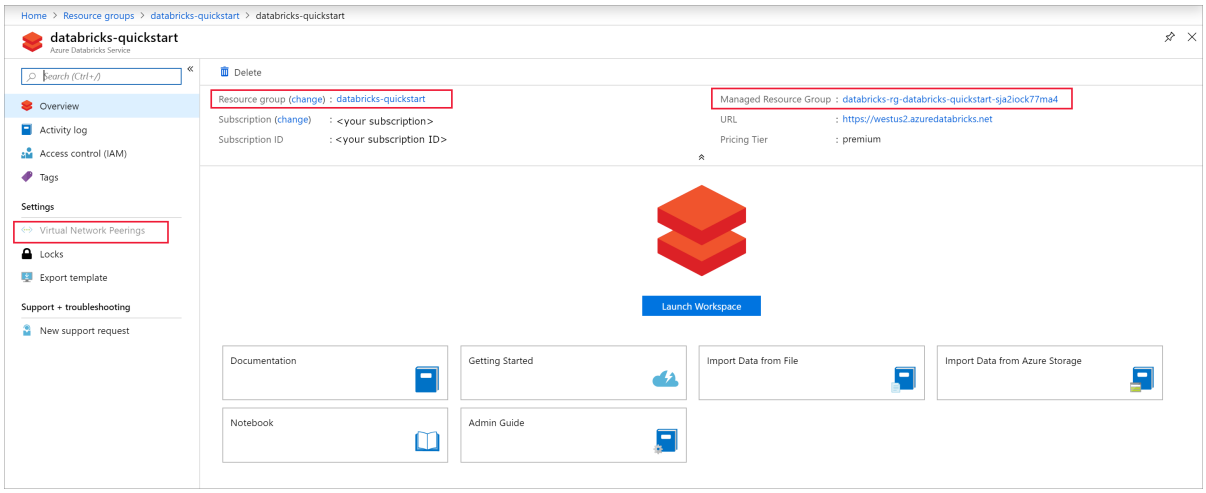


2. Under **Azure Databricks Service**, apply the following settings:

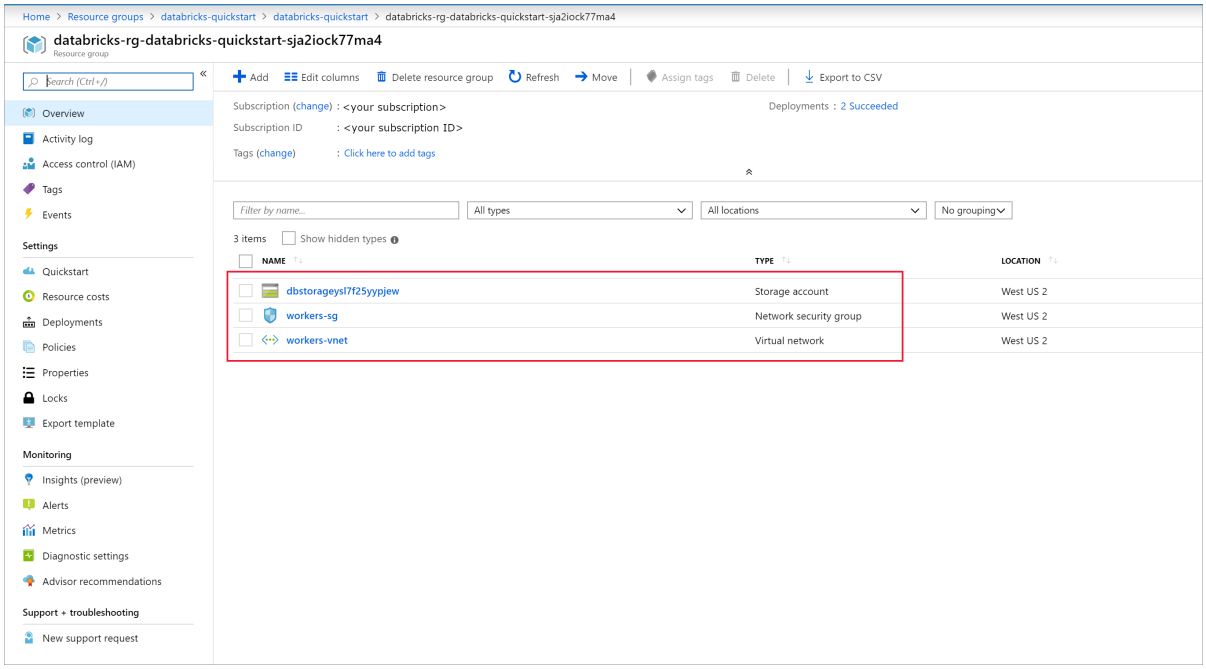
SETTING	SUGGESTED VALUE	DESCRIPTION
Workspace name	databricks-quickstart	Select a name for your Azure Databricks workspace.
Subscription	<Your subscription>	Select the Azure subscription that you want to use.
Resource group	databricks-quickstart	Select the same resource group you used for the virtual network.
Location	<Select the region that is closest to your users>	Choose the same location as your virtual network.
Pricing Tier	Choose between Standard or Premium.	For more information on pricing tiers, see the Databricks pricing page.
Deploy Azure Databricks workspace in your Virtual Network (VNet)	Yes	This setting allows you to deploy an Azure Databricks workspace in your virtual network.
Virtual Network	databricks-quickstart	Select the virtual network you created in the previous section.
Public Subnet Name	public-subnet	Use the default public subnet name.
Public Subnet CIDR Range	10.179.64.0/18	Use a CIDR range up to and including /26.
Private Subnet Name	private-subnet	Use the default private subnet name.
Private Subnet CIDR Range	10.179.0.0/18	Use a CIDR range up to and including /26.



3. Once the deployment is complete, navigate to the Azure Databricks resource. Notice that virtual network peering is disabled. Also notice the resource group and managed resource group in the overview page.



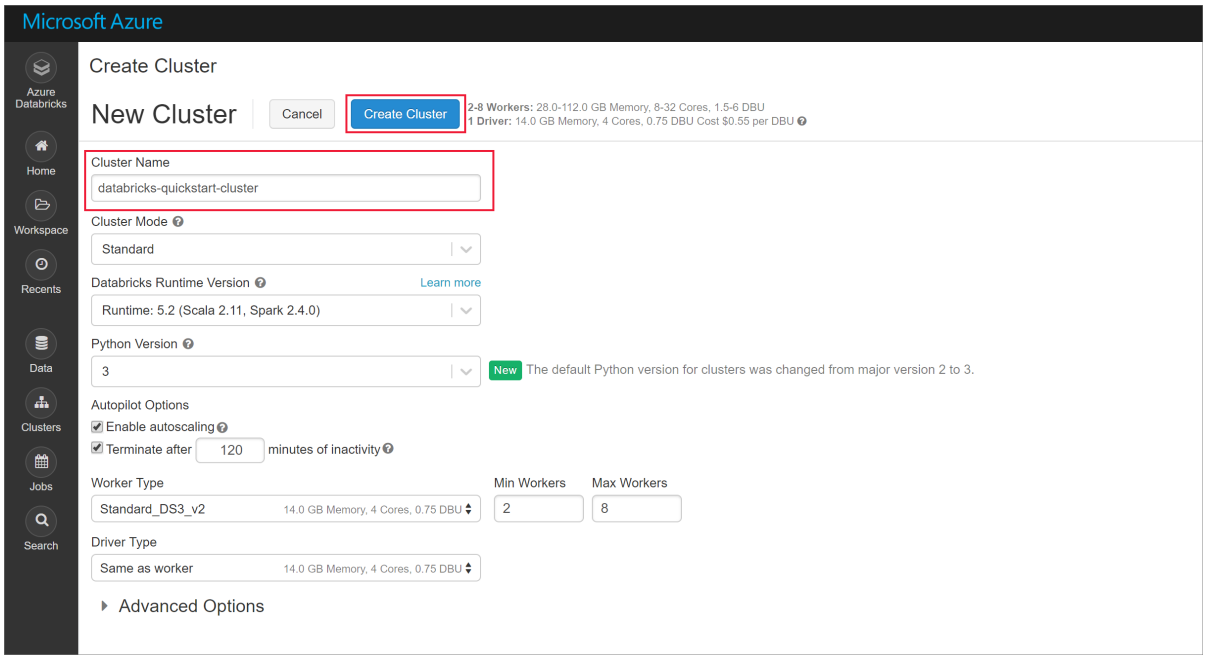
The managed resource group is not modifiable, and it is not used to create virtual machines. You can only create virtual machines in the resource group you manage.



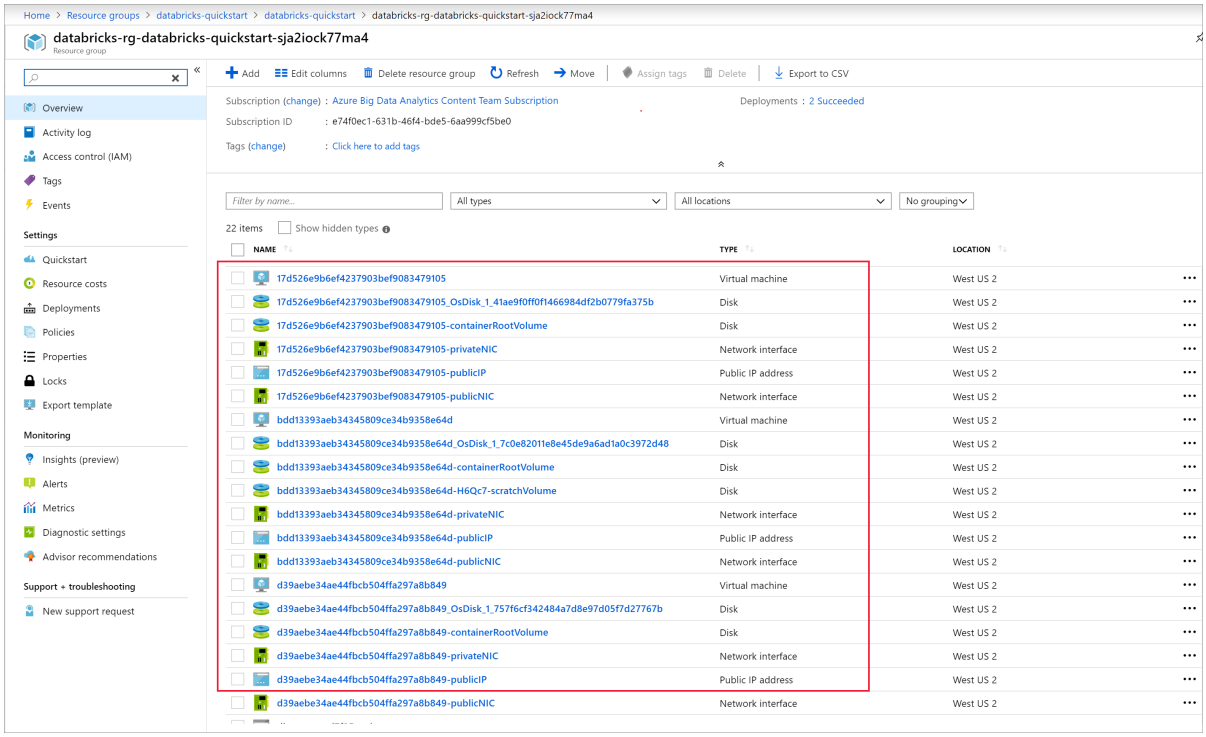
Create a cluster

> **NOTE**
>
> To use a free account to create the Azure Databricks cluster, before creating the cluster, go to your profile and change your subscription to **pay-as-you-go**. For more information, see Azure free account.

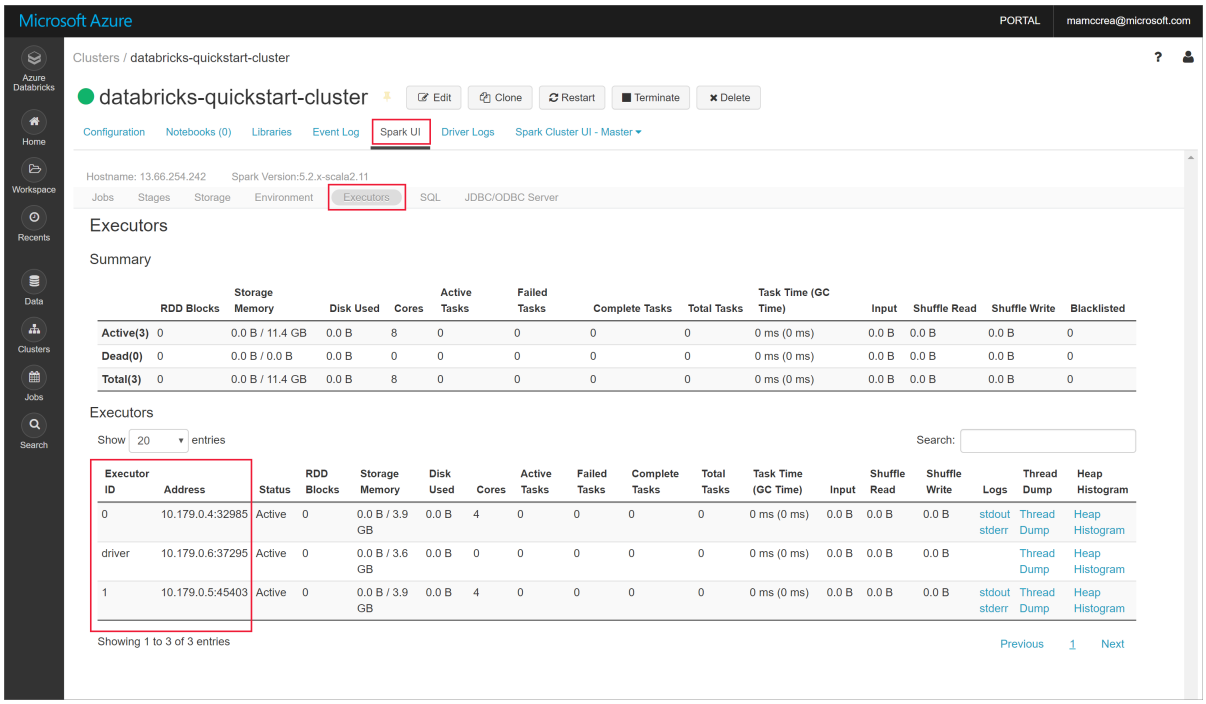
1. Return to your Azure Databricks service and select **Launch Workspace** on the **Overview** page.
2. Select **Clusters** > **+ Create Cluster**. Then create a cluster name, like *databricks-quickstart-cluster*, and accept the remaining default settings. Select **Create Cluster**.



3. Once the cluster is running, return to the managed resource group in the Azure portal. Notice the new virtual machines, disks, IP Address, and network interfaces. A network interface is created in each of the public and private subnets with IP addresses.



4. Return to your Azure Databricks workspace and select the cluster you created. Then navigate to the **Executors** tab on the **Spark UI** page. Notice that the addresses for the driver and the executors are in the private subnet range. In this example, the driver is 10.179.0.6 and executors are 10.179.0.4 and 10.179.0.5. Your IP addresses could be different.



Clean up resources

After you have finished the article, you can terminate the cluster. To do so, from the Azure Databricks workspace, from the left pane, select **Clusters**. For the cluster you want to terminate, move the cursor over the ellipsis under **Actions** column, and select the **Terminate** icon. This stops the cluster.

If you do not manually terminate the cluster it will automatically stop, provided you selected the **Terminate after __ minutes of inactivity** checkbox while creating the cluster. In such a case, the cluster automatically stops, if it has been inactive for the specified time.

If you do not wish to reuse the cluster, you can delete the resource group you created in the Azure portal.

Next steps

In this article, you created a Spark cluster in Azure Databricks that you deployed to a virtual network. Advance to the next article to learn how to query a SQL Server Linux Docker container in the virtual network using JDBC from an Azure Databricks notebook.

Query a SQL Server Linux Docker container in a virtual network from an Azure Databricks notebook

Tutorial: Query a SQL Server Linux Docker container in a virtual network from an Azure Databricks notebook

11/7/2019 • 5 minutes to read • Edit Online

This tutorial teaches you how to integrate Azure Databricks with a SQL Server Linux Docker container in a virtual network.

In this tutorial, you learn how to:

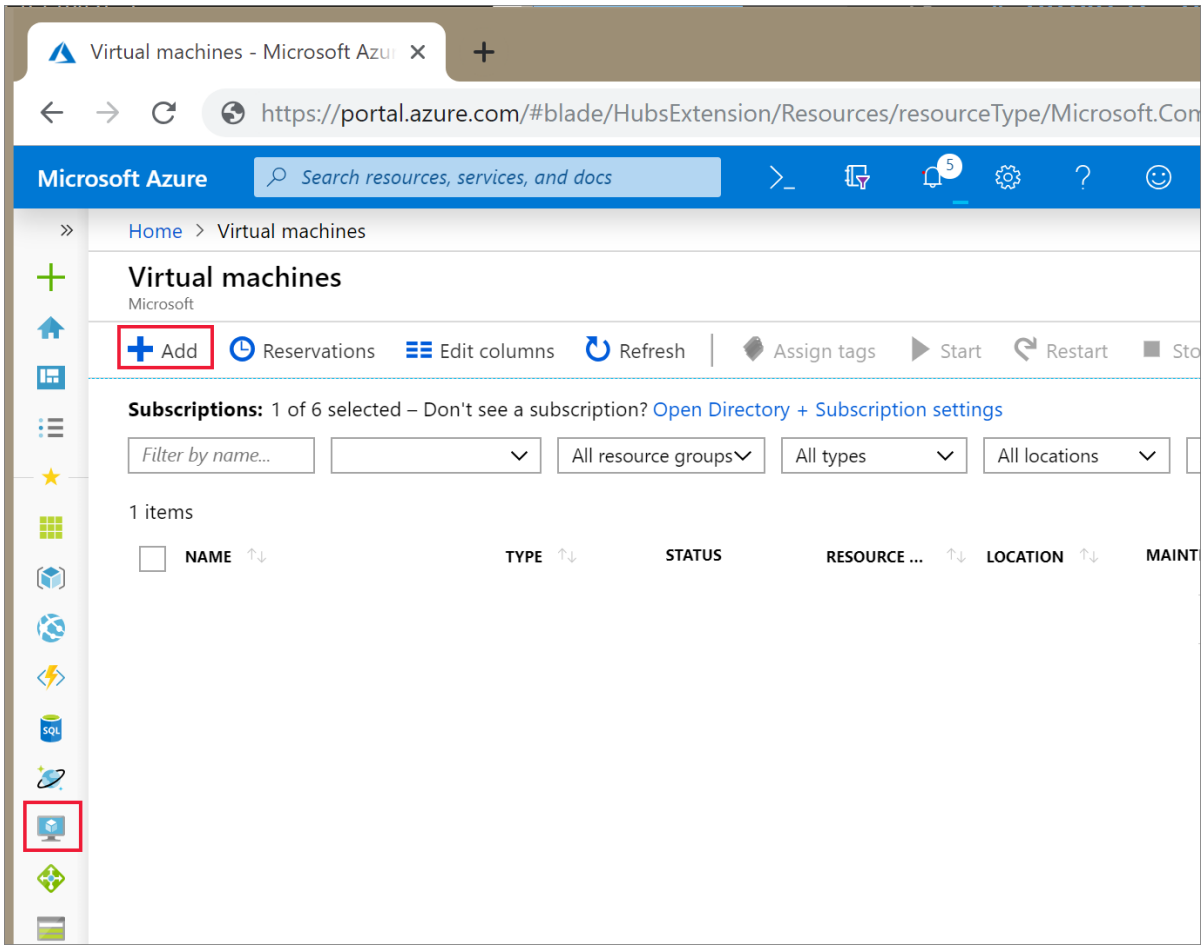
- Deploy an Azure Databricks workspace to a virtual network
- Install a Linux virtual machine in a public network
- Install Docker
- Install Microsoft SQL Server on Linux docker container
- Query the SQL Server using JDBC from a Databricks notebook

Prerequisites

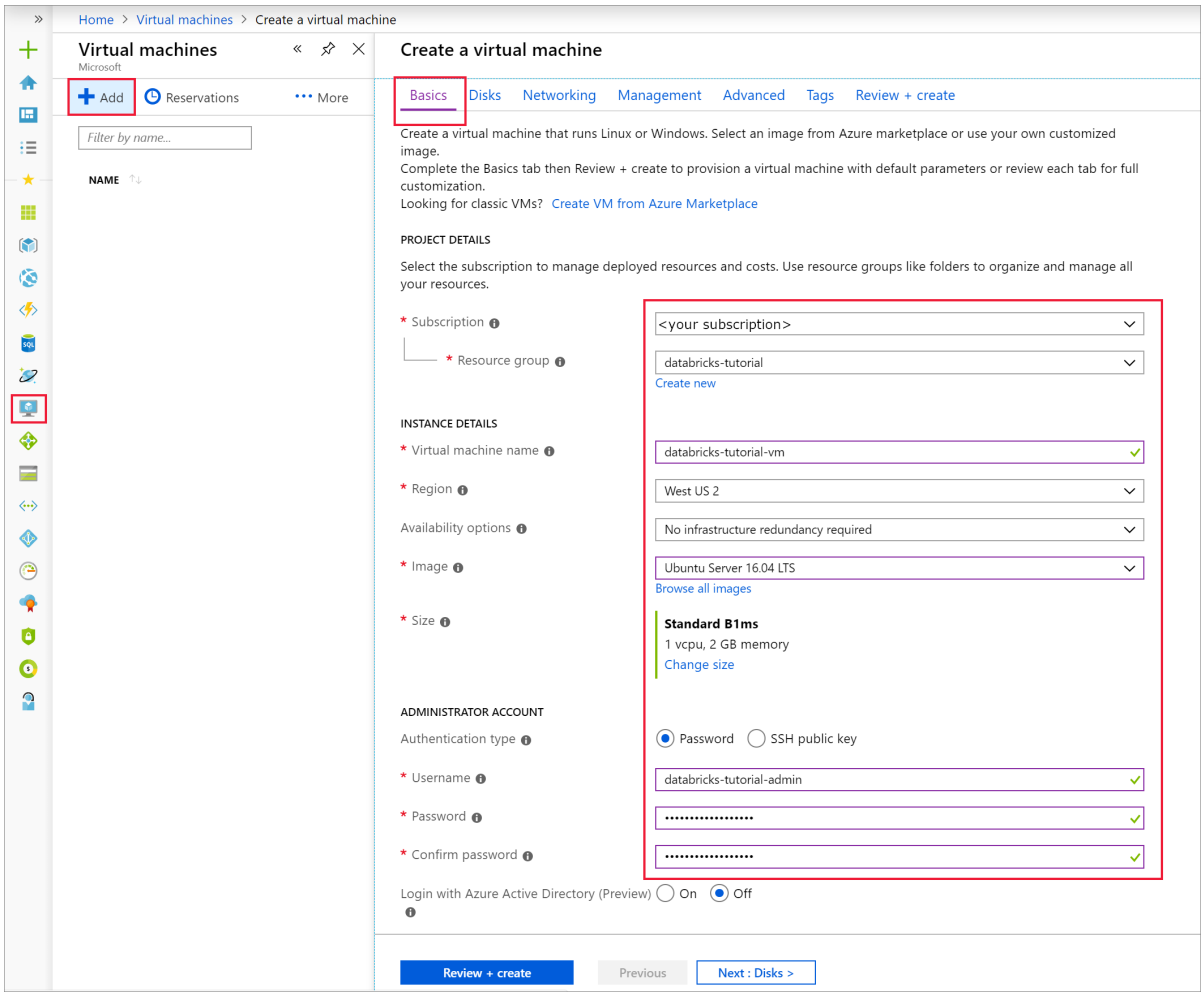
- Create a Databricks workspace in a virtual network.
- Install Ubuntu for Windows.
- Download SQL Server Management Studio.

Create a Linux virtual machine

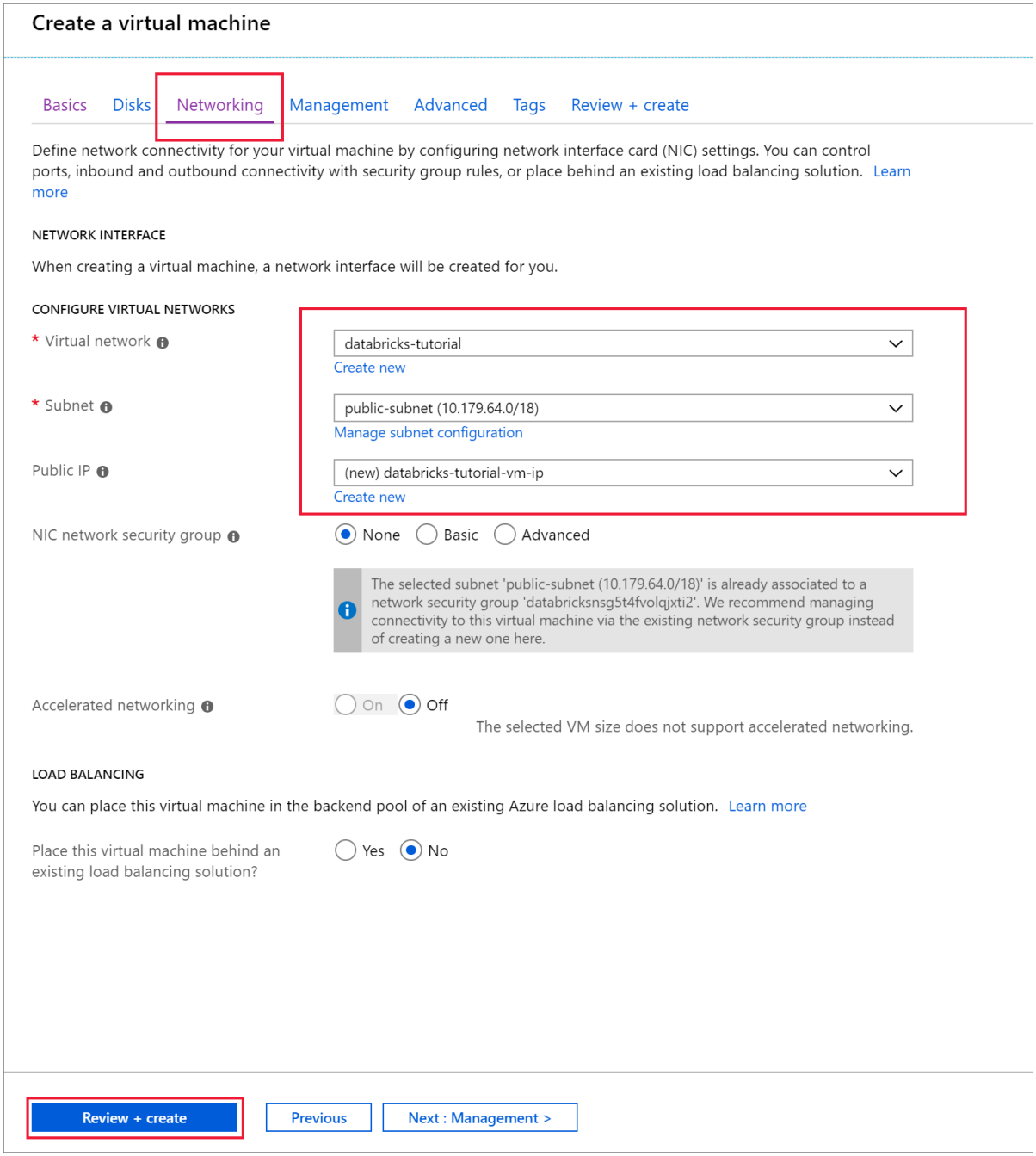
1. In the Azure portal, select the icon for **Virtual Machines**. Then, select **+ Add**.



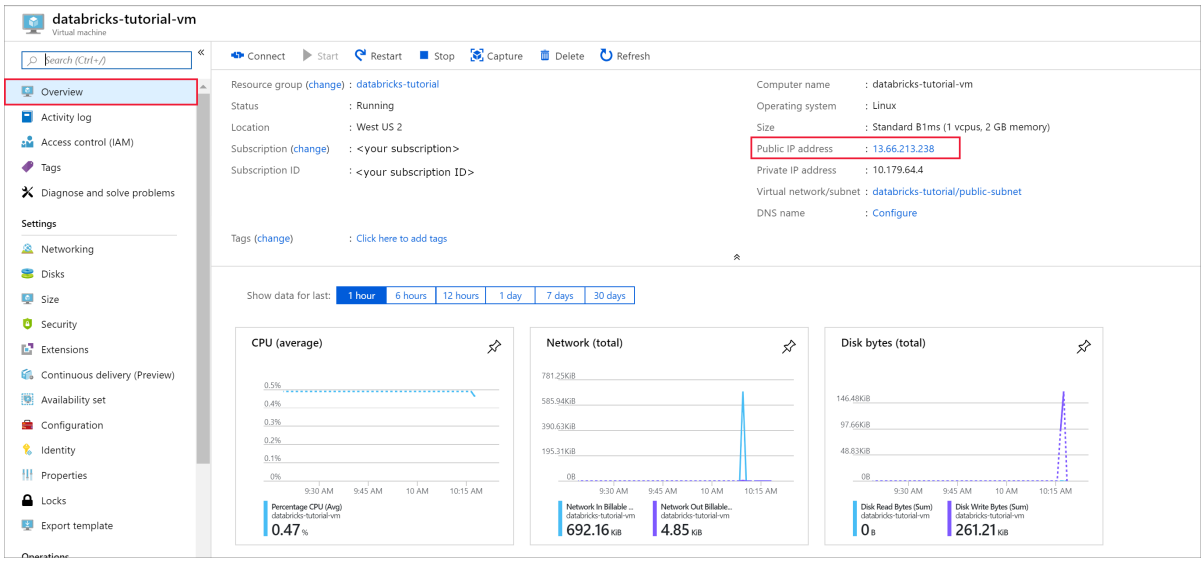
2. On the **Basics** tab, Choose Ubuntu Server 18.04 LTS and change the VM size to B2s. Choose an administrator username and password.



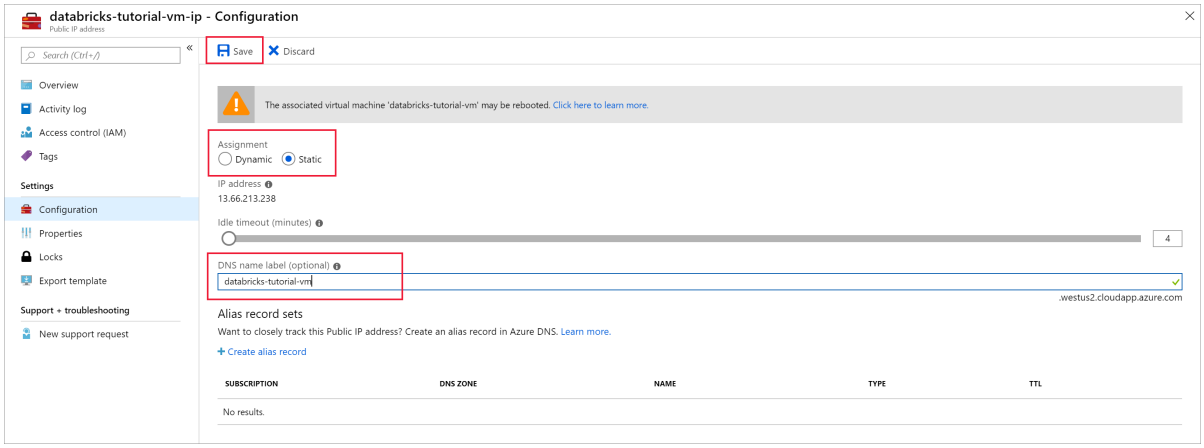
3. Navigate to the **Networking** tab. Choose the virtual network and the public subnet that includes your Azure Databricks cluster. Select **Review + create**, then **Create** to deploy the virtual machine.



4. When the deployment is complete, navigate to the virtual machine. Notice the Public IP address and Virtual network/subnet in the **Overview**. Select the **Public IP Address**



5. Change the **Assignment** to **Static** and enter a **DNS name label**. Select **Save**, and restart the virtual machine.

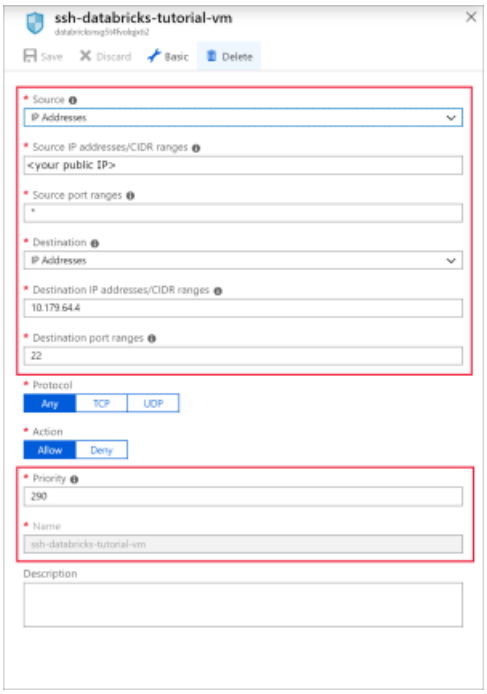


6. Select the **Networking** tab under **Settings**. Notice that the network security group that was created during the Azure Databricks deployment is associated with the virtual machine. Select **Add inbound port rule**.

7. Add a rule to open port 22 for SSH. Use the following settings:

SETTING	SUGGESTED VALUE	DESCRIPTION
Source	IP Addresses	IP Addresses specifies that incoming traffic from a specific source IP Address will be allowed or denied by this rule.
Source IP addresses	<your public ip>	Enter the your public IP address. You can find your public IP address by visiting bing.com and searching for **"my IP"**.
Source port ranges	*	Allow traffic from any port.
Destination	IP Addresses	IP Addresses specifies that outgoing traffic for a specific source IP Address will be allowed or denied by this rule.

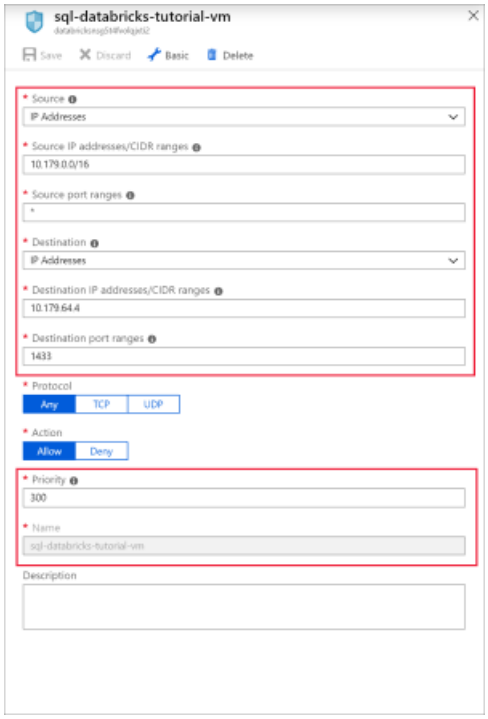
SETTING	SUGGESTED VALUE	DESCRIPTION
Destination IP addresses	<your vm public ip>	Enter your virtual machine's public IP address. You can find this on the **Overview** page of your virtual machine.
Destination port ranges	22	Open port 22 for SSH.
Priority	290	Give the rule a priority.
Name	ssh-databricks-tutorial-vm	Give the rule a name.



8. Add a rule to open port 1433 for SQL with the following settings:

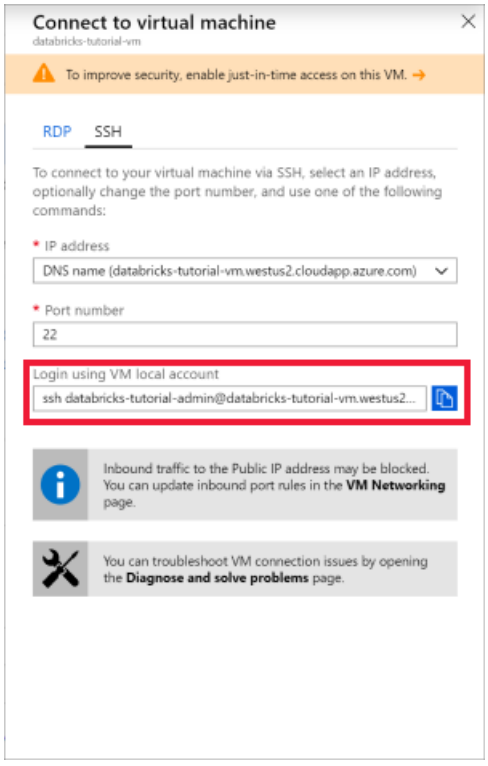
SETTING	SUGGESTED VALUE	DESCRIPTION
Source	Any	Source specifies that incoming traffic from a specific source IP Address will be allowed or denied by this rule.
Source port ranges	*	Allow traffic from any port.
Destination	IP Addresses	IP Addresses specifies that outgoing traffic for a specific source IP Address will be allowed or denied by this rule.
Destination IP addresses	<your vm public ip>	Enter your virtual machine's public IP address. You can find this on the **Overview** page of your virtual machine.
Destination port ranges	1433	Open port 22 for SQL Server.
Priority	300	Give the rule a priority.

SETTING	SUGGESTED VALUE	DESCRIPTION
Name	sql-databricks-tutorial-vm	Give the rule a name.

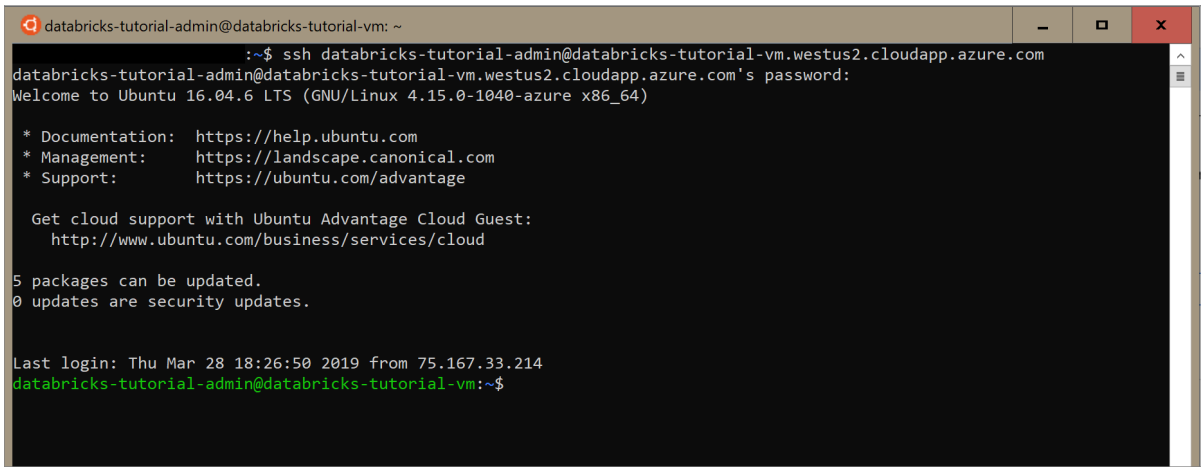


Run SQL Server in a Docker container

1. Open Ubuntu for Windows, or any other tool that will allow you to SSH into the virtual machine. Navigate to your virtual machine in the Azure portal and select **Connect** to get the SSH command you need to connect.



2. Enter the command in your Ubuntu terminal and enter the admin password you created when you configured the virtual machine.



3. Use the following command to install Docker on the virtual machine.

   ```
   sudo apt-get install docker.io
   ```

   Verify the install of Docker with the following command:

   ```
   sudo docker --version
   ```

4. Install the image.

   ```
   sudo docker pull mcr.microsoft.com/mssql/server:2017-latest
   ```

   Check the images.

   ```
   sudo docker images
   ```

5. Run the container from the image.

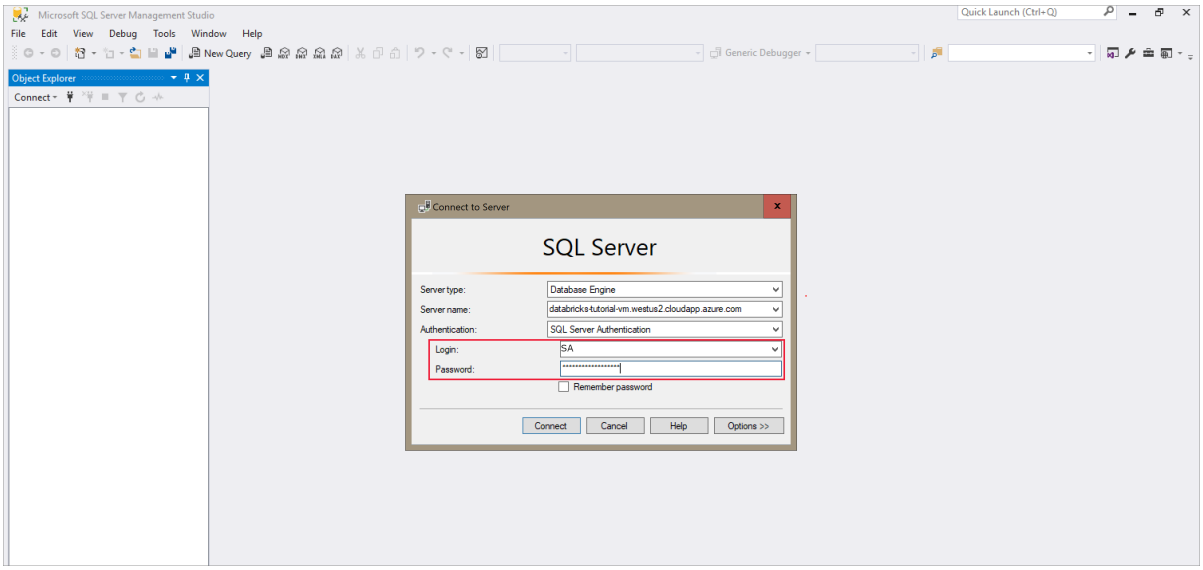
   ```
   sudo docker run -e 'ACCEPT_EULA=Y' -e 'SA_PASSWORD=Password1234' -p 1433:1433 --name sql1  -d
   mcr.microsoft.com/mssql/server:2017-latest
   ```

   Verify that the container is running.

   ```
   sudo docker ps -a
   ```

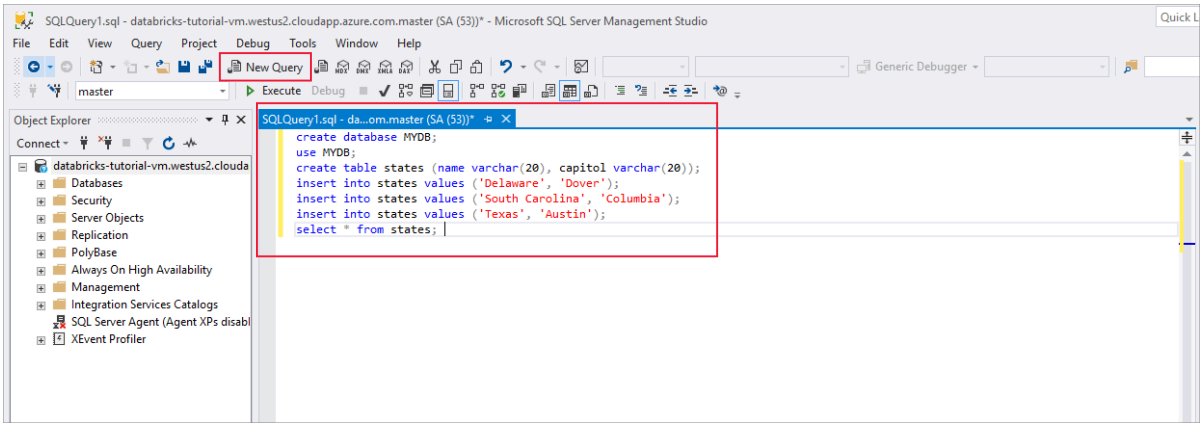
Create a SQL database

1. Open SQL Server Management Studio and connect to the server using the server name and SQL Authentication. The sign in username is **SA** and the password is the password set in the Docker command. The password in the example command is `Password1234`.



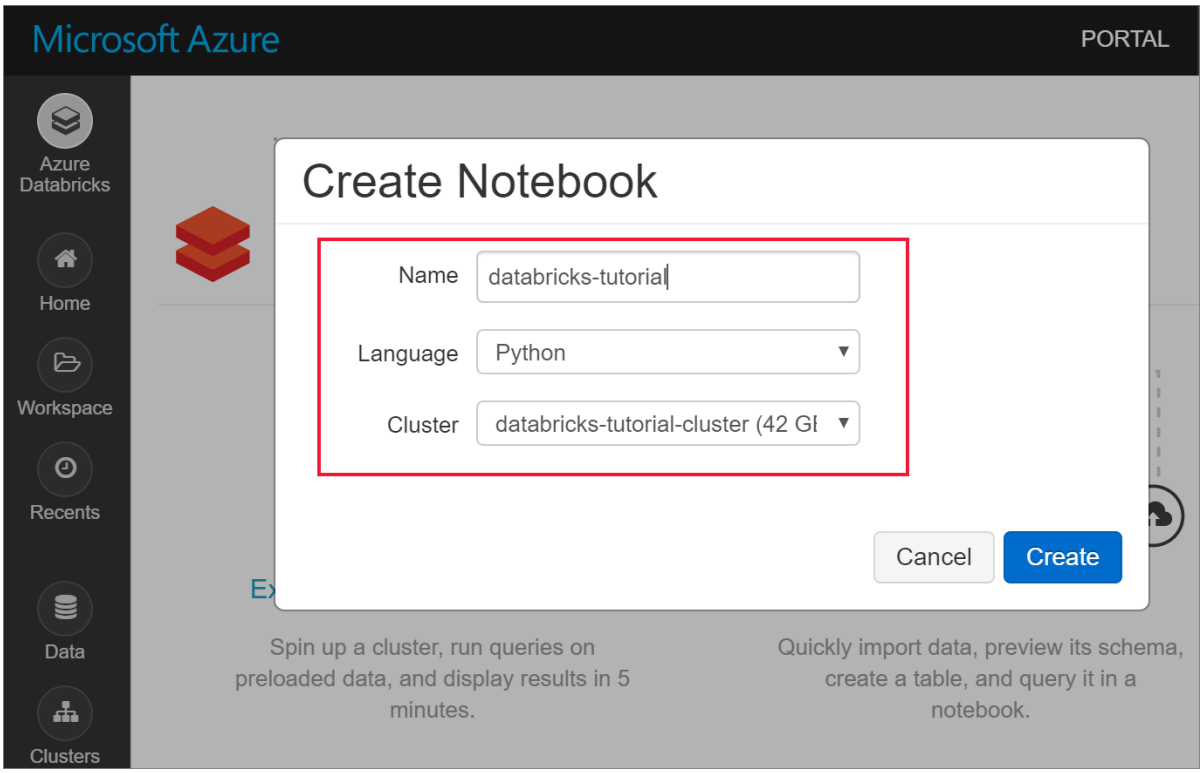
2. Once you've successfully connected, select **New Query** and enter the following code snippet to create a database, a table, and insert some records in the table.

```
CREATE DATABASE MYDB;
GO
USE MYDB;
CREATE TABLE states(Name VARCHAR(20), Capitol VARCHAR(20));
INSERT INTO states VALUES ('Delaware','Dover');
INSERT INTO states VALUES ('South Carolina','Columbia');
INSERT INTO states VALUES ('Texas','Austin');
SELECT * FROM states
GO
```



Query SQL Server from Azure Databricks

1. Navigate to your Azure Databricks workspace and verify that you created a cluster as part of the prerequisites. Then, select **Create a Notebook**. Give the notebook a name, select *Python* as the language, and select the cluster you created.



2. Use the following command to ping the internal IP Address of the SQL Server virtual machine. This ping should be successful. If not, verify that the container is running, and review the network security group (NSG) configuration.

```
%sh
ping 10.179.64.4
```

You can also use the nslookup command to review.

```
%sh
nslookup databricks-tutorial-vm.westus2.cloudapp.azure.com
```

3. Once you've successfully pinged the SQL Server, you can query the database and tables. Run the following python code:

```
jdbcHostname = "10.179.64.4"
jdbcDatabase = "MYDB"
userName = 'SA'
password = 'Password1234'
jdbcPort = 1433
jdbcUrl = "jdbc:sqlserver://{0}:{1};database={2};user={3};password={4}".format(jdbcHostname, jdbcPort,
jdbcDatabase, userName, password)

df = spark.read.jdbc(url=jdbcUrl, table='states')
display(df)
```

Clean up resources

When no longer needed, delete the resource group, the Azure Databricks workspace, and all related resources. Deleting the job avoids unnecessary billing. If you're planning to use the Azure Databricks workspace in future, you can stop the cluster and restart it later. If you are not going to continue to use this Azure Databricks workspace, delete all resources you created in this tutorial by using the following steps:

1. From the left-hand menu in the Azure portal, click **Resource groups** and then click the name of the

resource group you created.

2. On your resource group page, select **Delete**, type the name of the resource to delete in the text box, and then select **Delete** again.

Next steps

Advance to the next article to learn how to extract, transform, and load data using Azure Databricks.

Tutorial: Extract, transform, and load data by using Azure Databricks

Tutorial: Access Azure Blob Storage from Azure Databricks using Azure Key Vault

12/5/2019 • 5 minutes to read • Edit Online

This tutorial describes how to access Azure Blob Storage from Azure Databricks using secrets stored in a key vault.

In this tutorial, you learn how to:

- Create a storage account and blob container
- Create an Azure Key Vault and add a secret
- Create an Azure Databricks workspace and add a secret scope
- Access your blob container from Azure Databricks

Prerequisites

- Azure subscription - create one for free

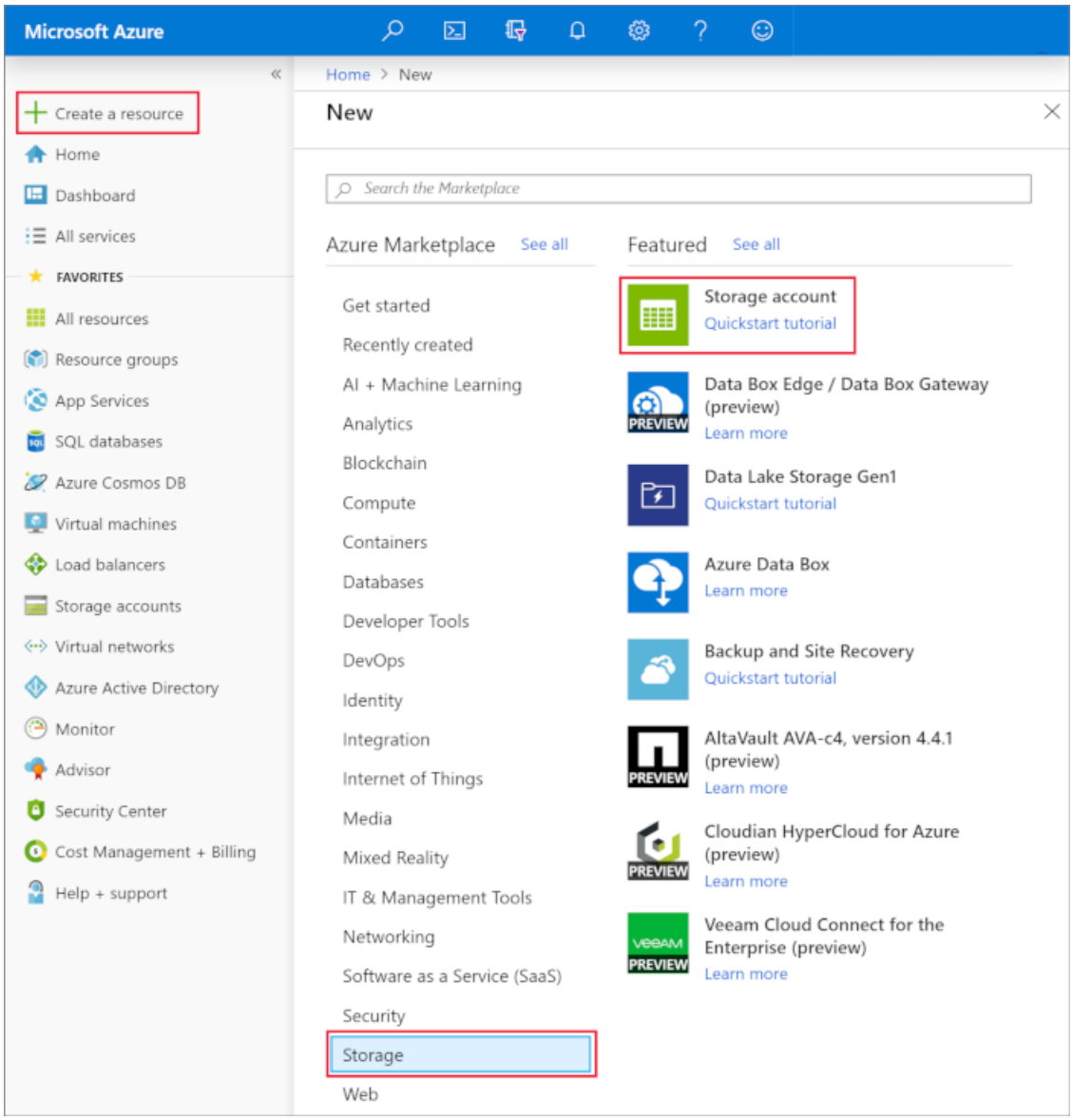
Sign in to the Azure portal

Sign in to the Azure portal.

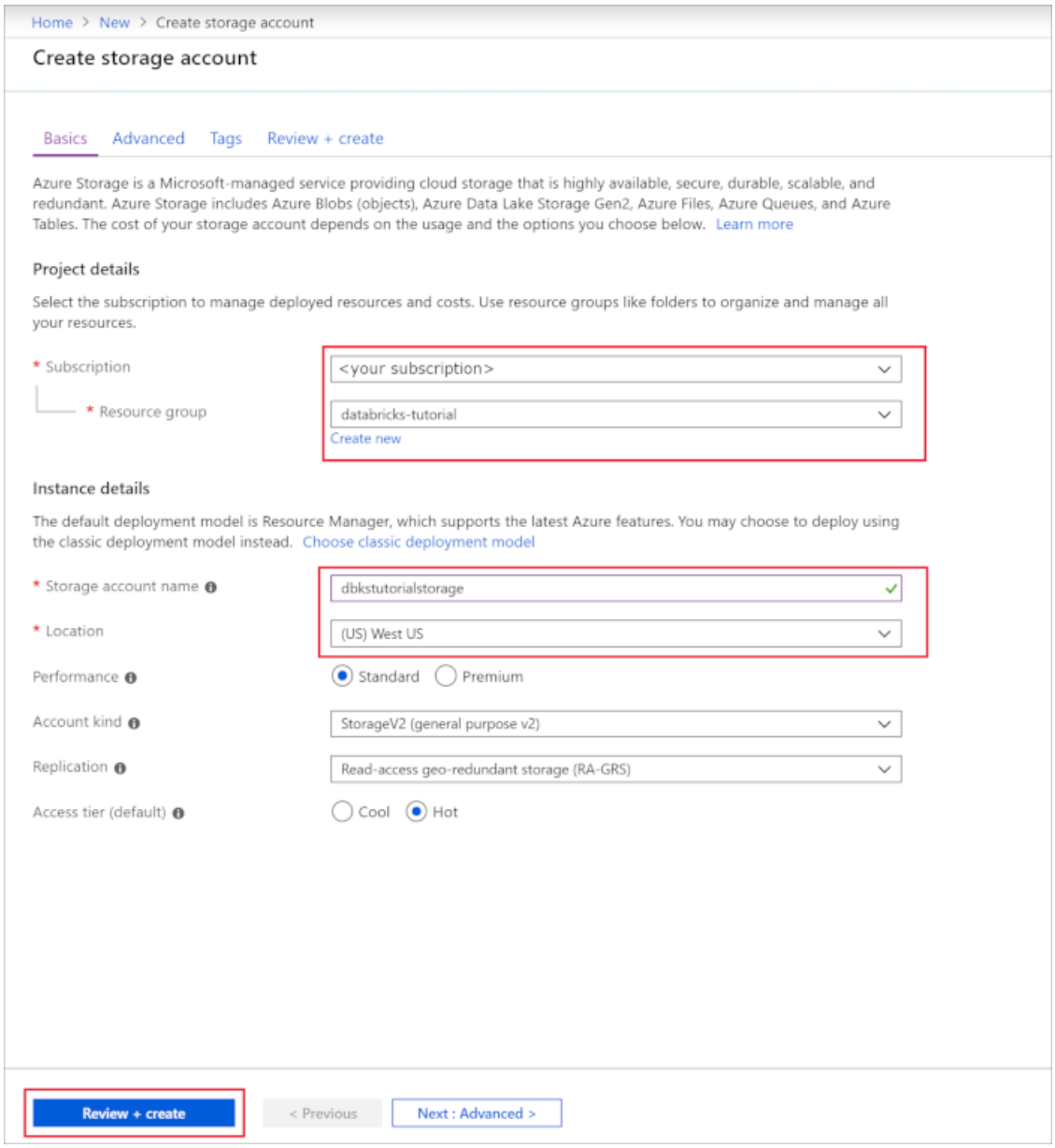
> **NOTE**
>
> This tutorial cannot be carried out using **Azure Free Trial Subscription**. If you have a free account, go to your profile and change your subscription to **pay-as-you-go**. For more information, see Azure free account. Then, remove the spending limit, and request a quota increase for vCPUs in your region. When you create your Azure Databricks workspace, you can select the **Trial (Premium - 14-Days Free DBUs)** pricing tier to give the workspace access to free Premium Azure Databricks DBUs for 14 days.

Create a storage account and blob container

1. In the Azure portal, select **Create a resource** > **Storage**. Then select **Storage account**.

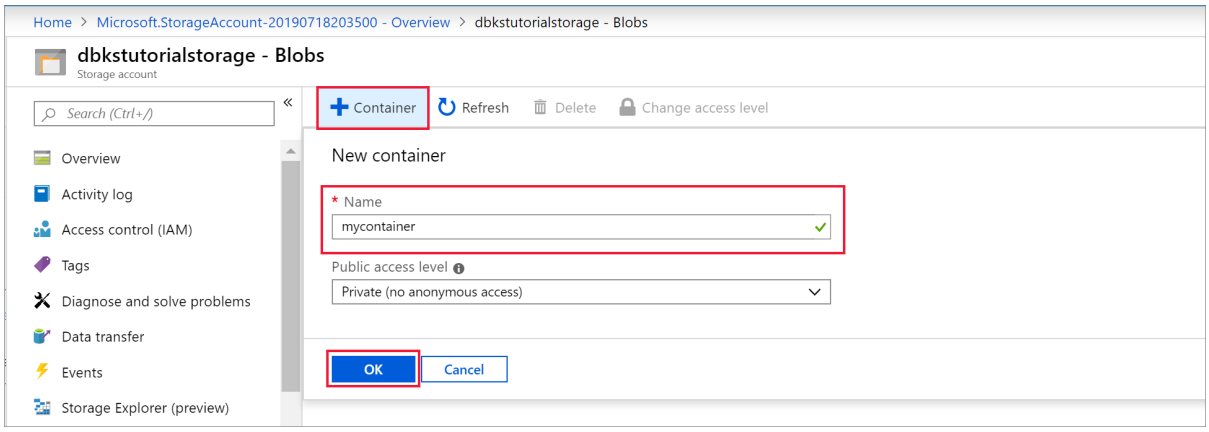


2. Select your subscription and resource group, or create a new resource group. Then enter a storage account name, and choose a location. Select **Review + Create**.

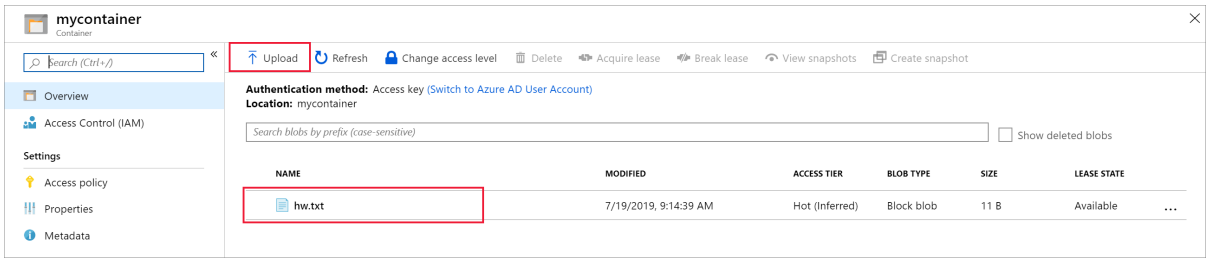


3. If the validation is unsuccessful, address the issues and try again. If the validation is successful, select **Create** and wait for the storage account to be created.

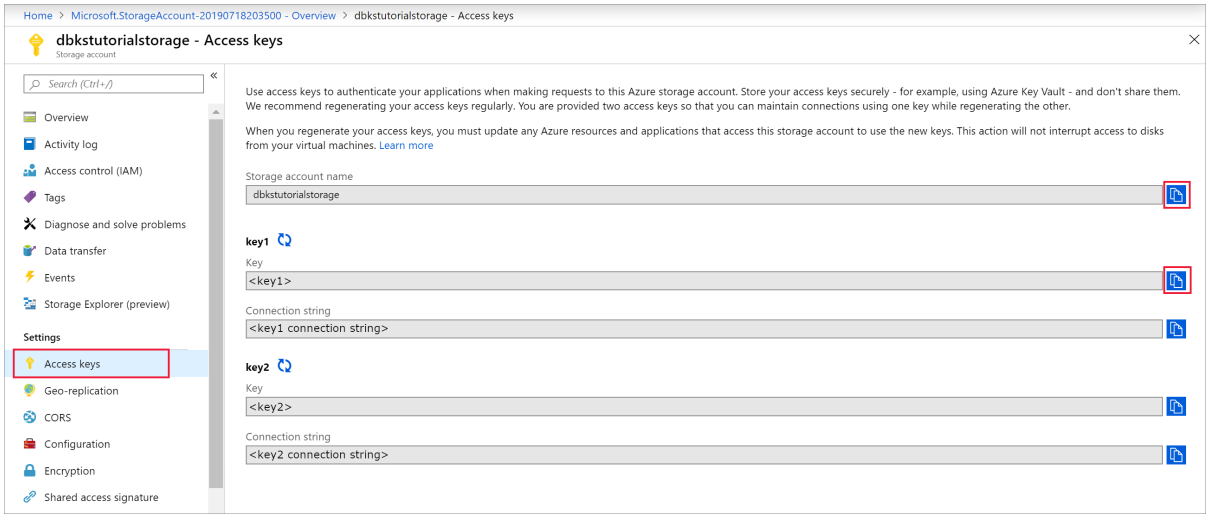
4. Navigate to your newly created storage account and select **Blobs** under **Services** on the **Overview** page. Then select **+ Container** and enter a container name. Select **OK**.



5. Locate a file you want to upload to your blob storage container. If you don't have a file, use a text editor to create a new text file with some information. In this example, a file named **hw.txt** contains the text "hello world." Save your text file locally and upload it to your blob storage container.

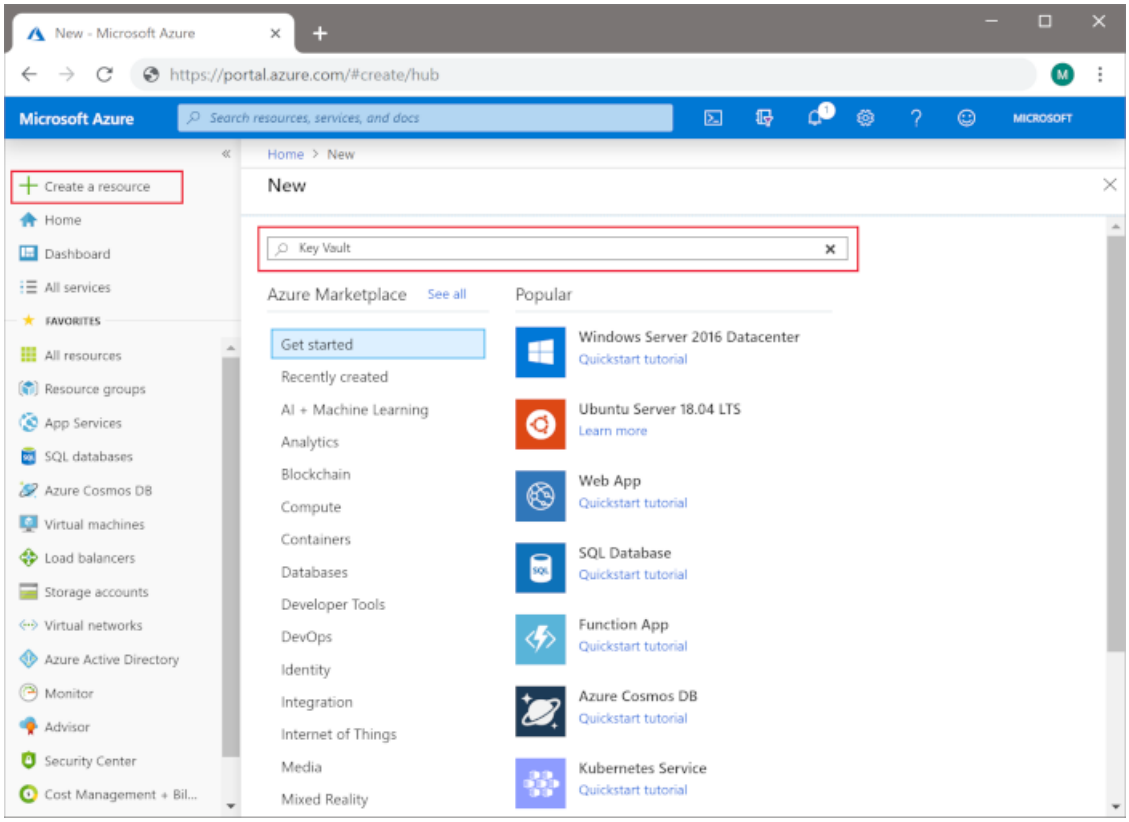


6. Return to your storage account and select **Access keys** under **Settings**. Copy **Storage account name** and **key 1** to a text editor for later use in this tutorial.

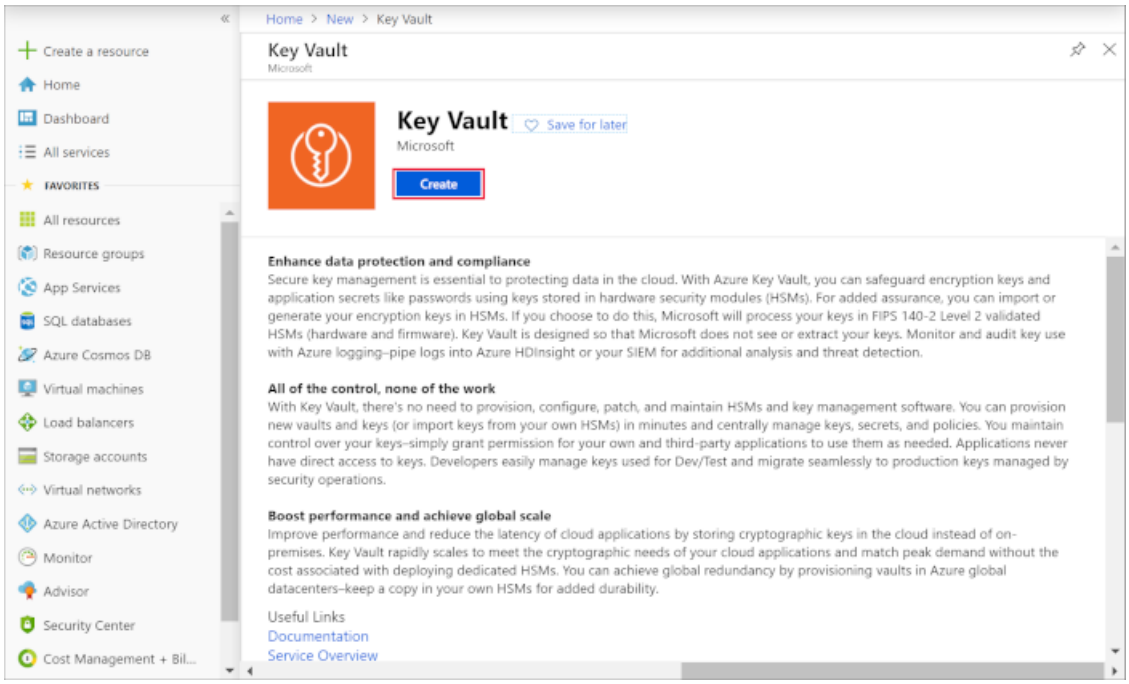


Create an Azure Key Vault and add a secret

1. In the Azure portal, select **Create a resource** and enter **Key Vault** in the search box.

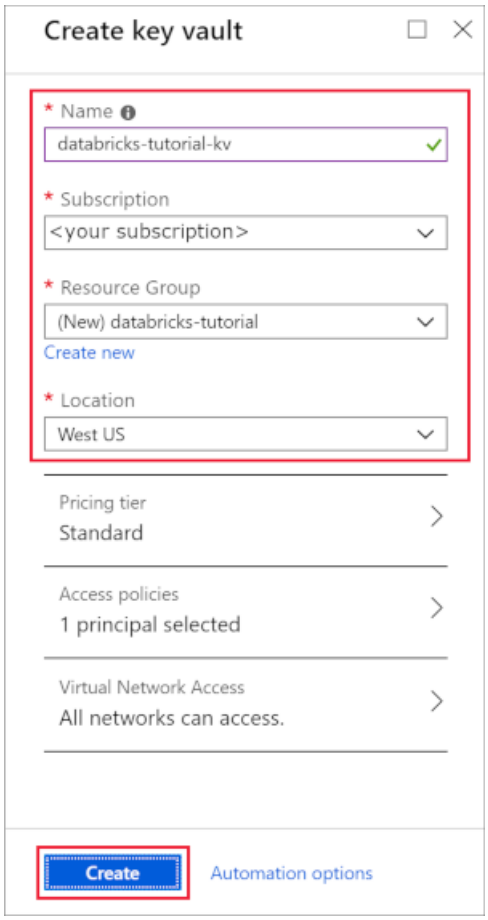


2. The Key Vault resource is automatically selected. Select **Create**.



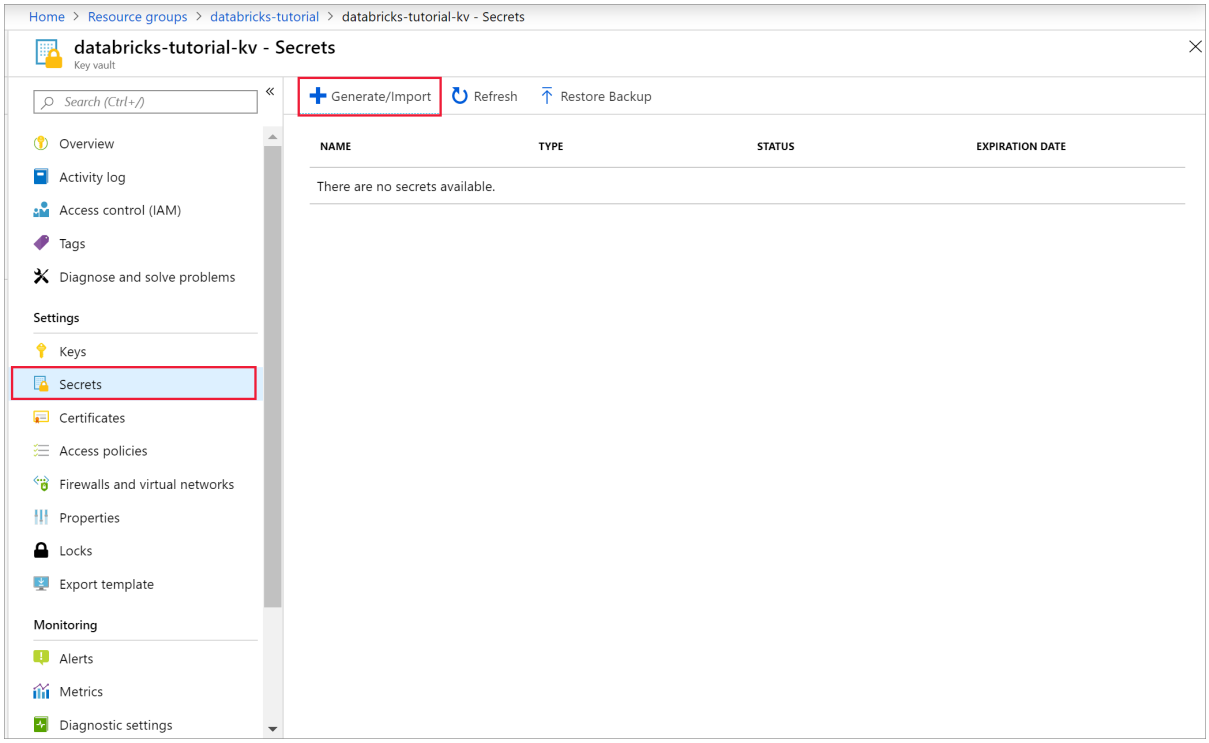
3. On the **Create key vault** page, enter the following information, and keep the default values for the remaining fields:

PROPERTY	DESCRIPTION
Name	A unique name for your key vault.
Subscription	Choose a subscription.
Resource group	Choose a resource group or create a new one.
Location	Choose a location.



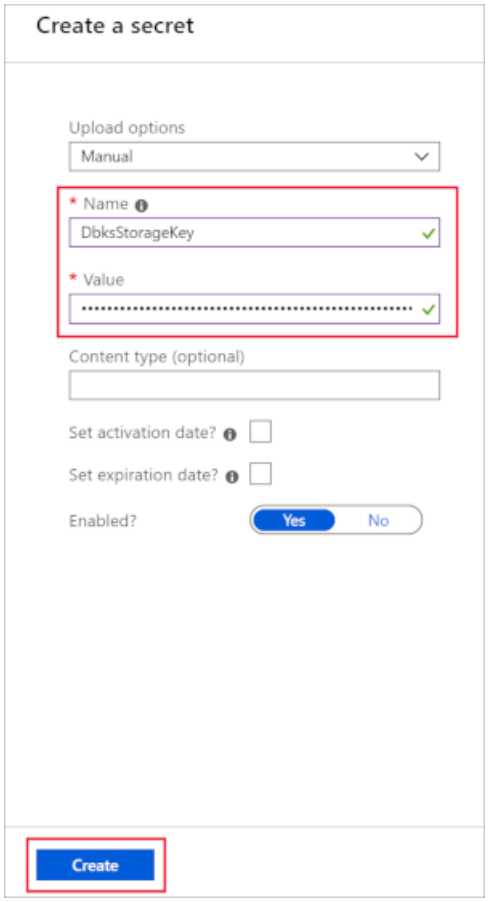
4. After providing the information above, select **Create**.

5. Navigate to your newly created key vault in the Azure portal and select **Secrets**. Then, select **+ Generate/Import**.

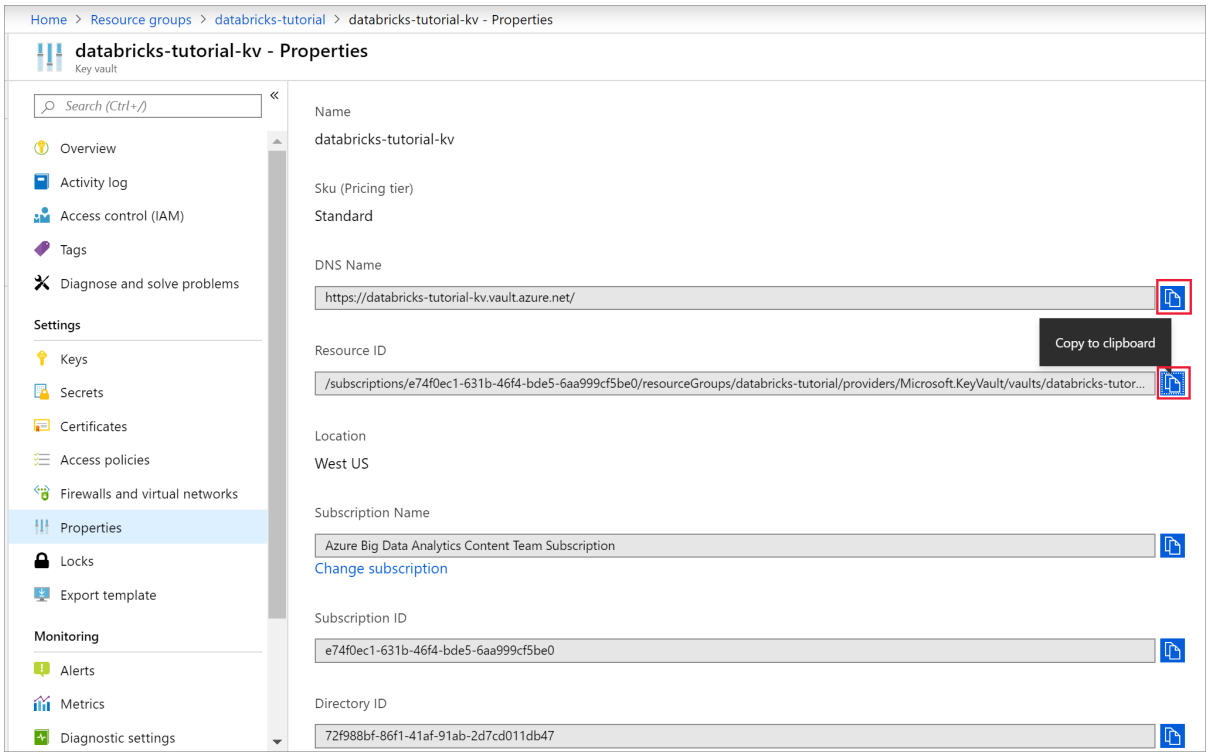


6. On the **Create a secret** page, provide the following information, and keep the default values for the remaining fields:

PROPERTY	VALUE
Upload options	Manual
Name	Friendly name for your storage account key.
Value	key1 from your storage account.

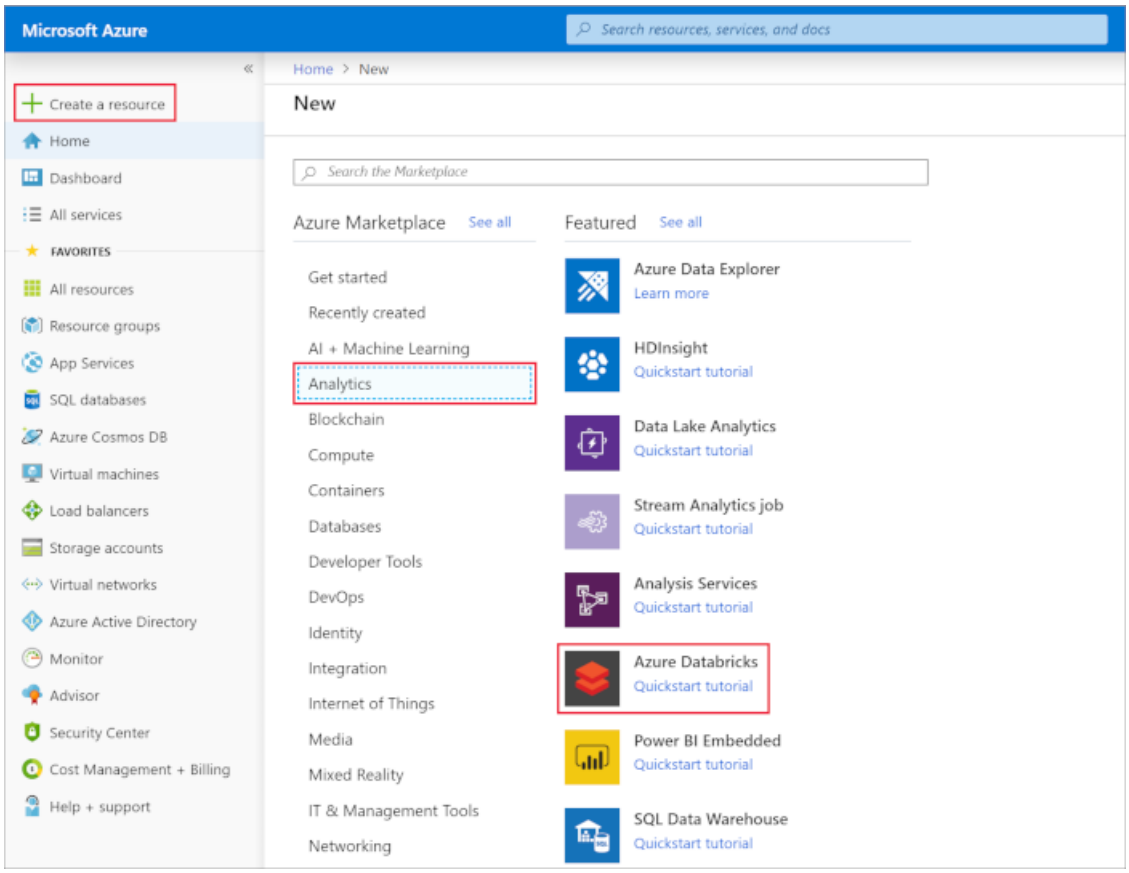


7. Save the key name in a text editor for use later in this tutorial, and select **Create**. Then, navigate to the **Properties** menu. Copy the **DNS Name** and **Resource ID** to a text editor for use later in the tutorial.



Create an Azure Databricks workspace and add a secret scope

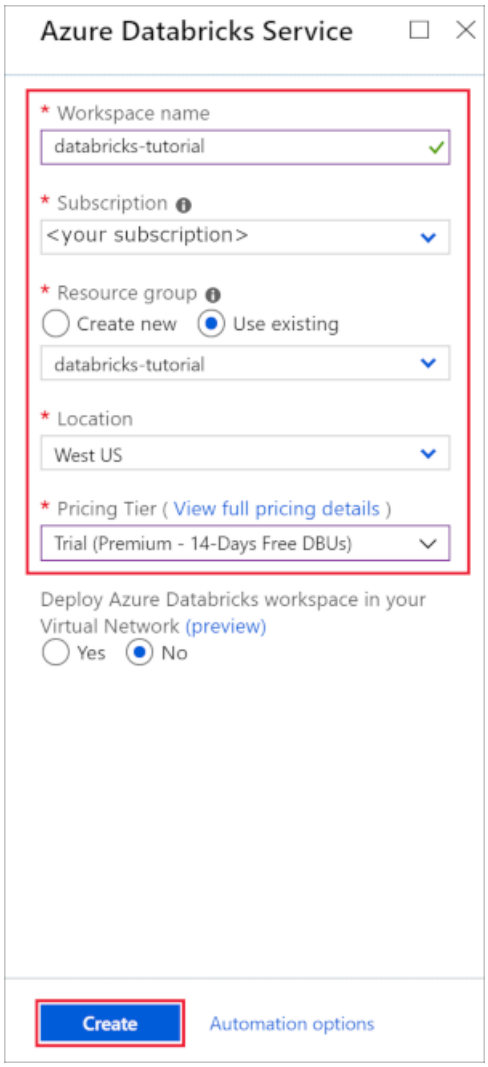
1. In the Azure portal, select **Create a resource** > **Analytics** > **Azure Databricks**.



2. Under **Azure Databricks Service**, provide the following values to create a Databricks workspace.

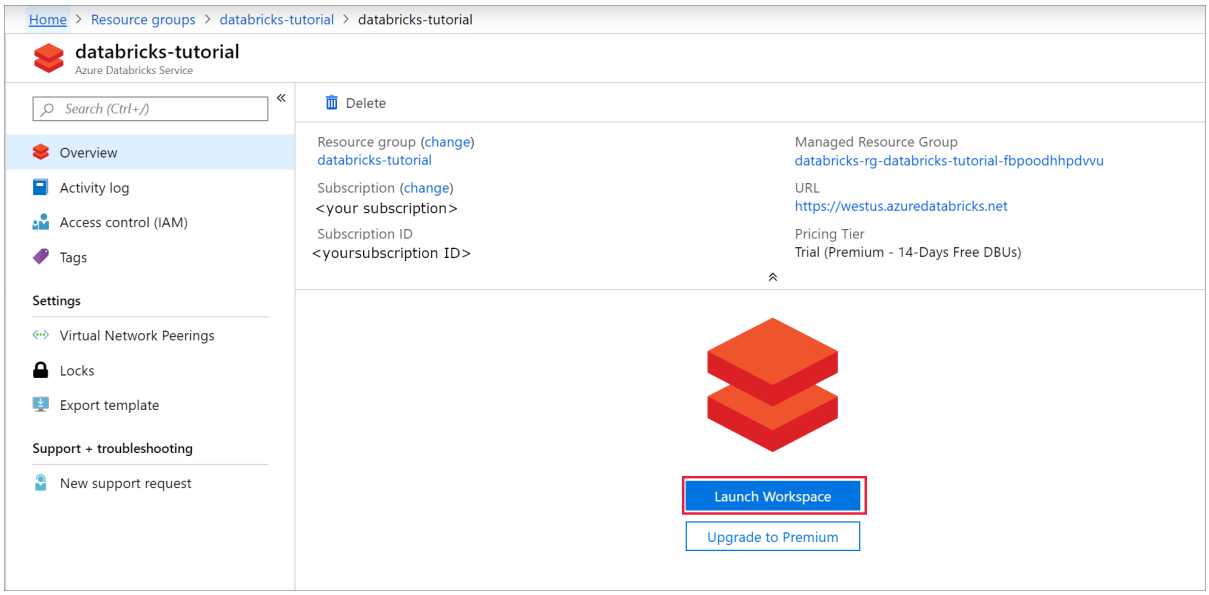
PROPERTY	DESCRIPTION
Workspace name	Provide a name for your Databricks workspace

PROPERTY	DESCRIPTION
Subscription	From the drop-down, select your Azure subscription.
Resource group	Select the same resource group that contains your key vault.
Location	Select the same location as your Azure Key Vault. For all available regions, see Azure services available by region.
Pricing Tier	Choose between **Standard** or **Premium**. For more information on these tiers, see Databricks pricing page.



Select **Create**.

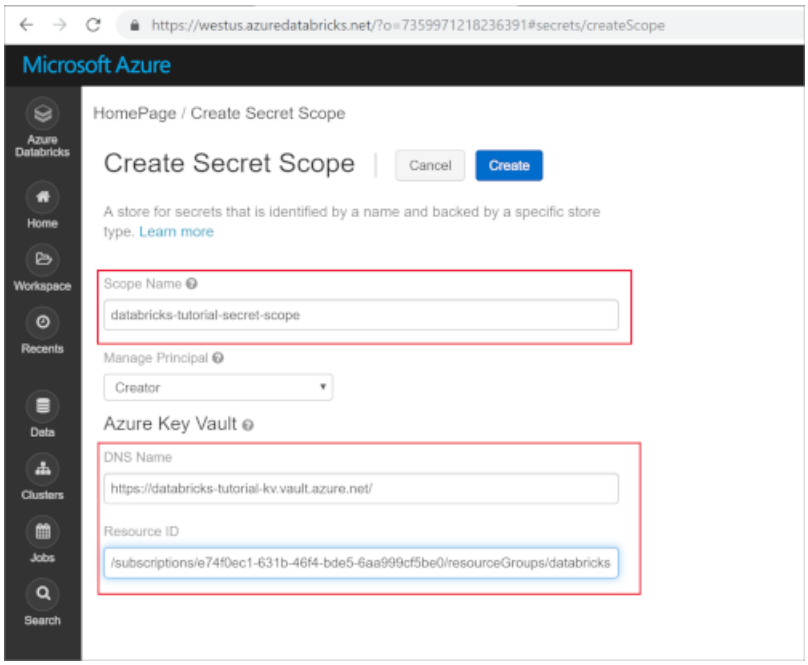
3. Navigate to your newly created Azure Databricks resource in the Azure portal and select **Launch Workspace**.



4. Once your Azure Databricks workspace is open in a separate window, append **#secrets/createScope** to the URL. The URL should have the following format:

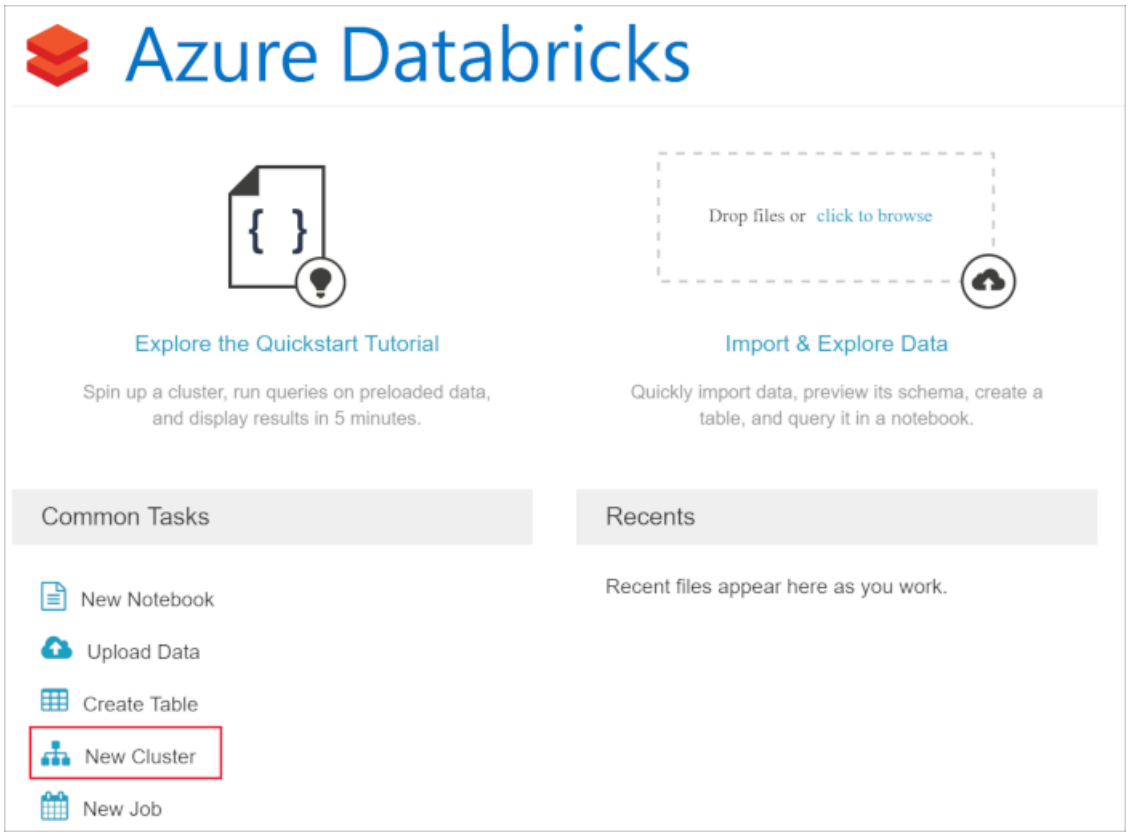
   https://<\location>.azuredatabricks.net/?o=<\orgID>#secrets/createScope.

5. Enter a scope name, and enter the Azure Key Vault DNS name and Resource ID you saved earlier. Save the scope name in a text editor for use later in this tutorial. Then, select **Create**.

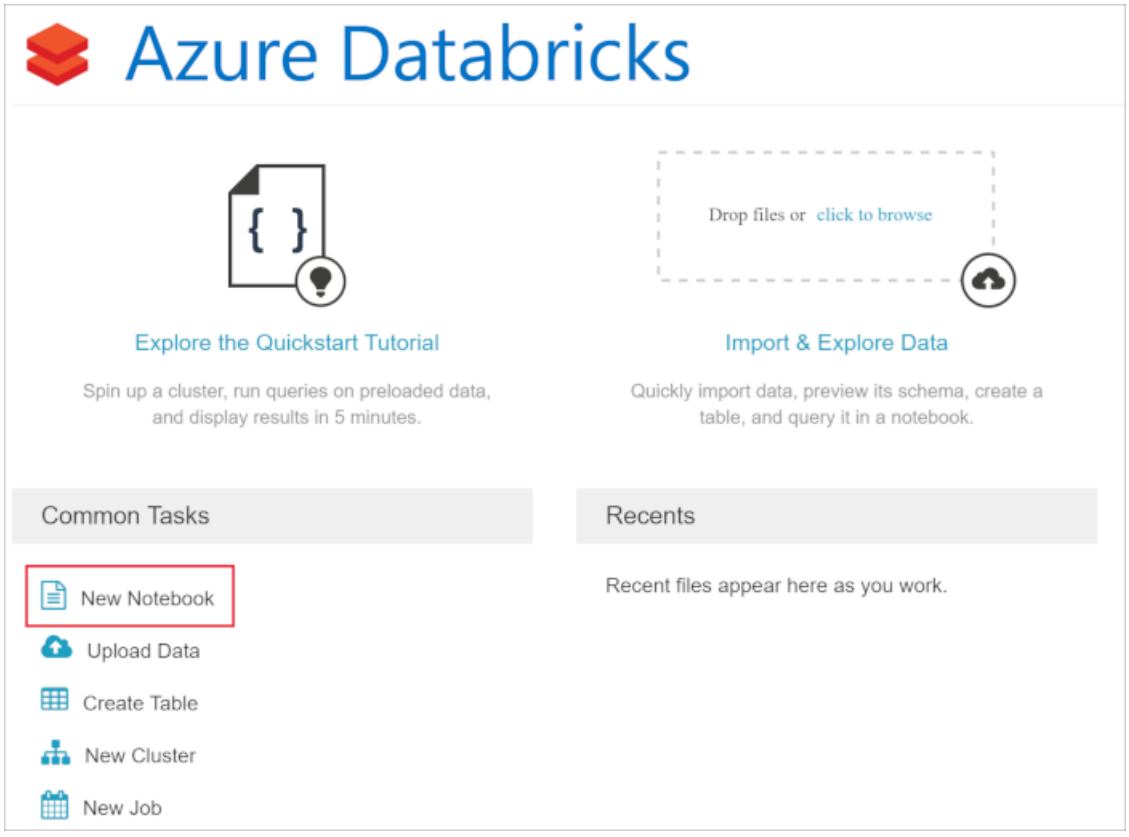


Access your blob container from Azure Databricks

1. From the home page of your Azure Databricks workspace, select **New Cluster** under **Common Tasks**.



2. Enter a cluster name and select **Create cluster**. The cluster creation takes a few minutes to complete.
3. Once the cluster is created, navigate to the home page of your Azure Databricks workspace, select **New Notebook** under **Common Tasks**.



4. Enter a notebook name, and set the language to Python. Set the cluster to the name of the cluster you created in the previous step.
5. Run the following command to mount your blob storage container. Remember to change the values for the following properties:

- your-container-name
- your-storage-account-name
- mount-name
- config-key
- scope-name
- key-name

```
dbutils.fs.mount(
source = "wasbs://<your-container-name>@<your-storage-account-name>.blob.core.windows.net",
mount_point = "/mnt/<mount-name>",
extra_configs = {"<conf-key>":dbutils.secrets.get(scope = "<scope-name>", key = "<key-name>")})
```

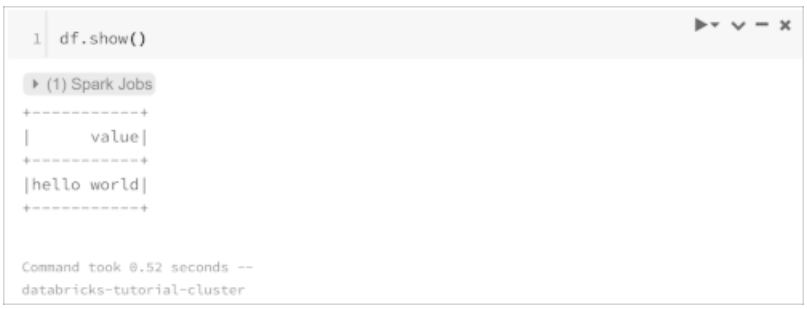
- **mount-name** is a DBFS path representing where the Blob Storage container or a folder inside the container (specified in source) will be mounted.
- **conf-key** can be either `fs.azure.account.key.<\your-storage-account-name>.blob.core.windows.net` or `fs.azure.sas.<\your-container-name>.<\your-storage-account-name>.blob.core.windows.net`
- **scope-name** is the name of the secret scope you created in the previous section.
- **key-name** is the name of they secret you created for the storage account key in your key vault.

6. Run the following command to read the text file in your blob storage container to a dataframe. Change the values in the command to match your mount name and file name.

```
df = spark.read.text("mnt/<mount-name>/<file-name>")
```

7. Use the following command to display the contents of your file.

```
df.show()
```



8. To unmount your blob storage, run the following command:

```
dbutils.fs.unmount("/mnt/<mount-name>")
```

9. Notice that once the mount has been unmounted, you can no longer read from your blob storage account.

Clean up resources

If you're not going to continue to use this application, delete your entire resource group with the following steps:

1. From the left-hand menu in Azure portal, select **Resource groups** and navigate to your resource group.
2. Select **Delete resource group** and type your resource group name. Then select **Delete**.

Next steps

Advance to the next article to learn how to implement a VNet injected Databricks environment with a Service Endpoint enabled for Cosmos DB.

Tutorial: Implement Azure Databricks with a Cosmos DB endpoint

Tutorial: Implement Azure Databricks with a Cosmos DB endpoint

12/2/2019 • 4 minutes to read • Edit Online

This tutorial describes how to implement a VNet injected Databricks environment with a Service Endpoint enabled for Cosmos DB.

In this tutorial you learn how to:

- Create an Azure Databricks workspace in a virtual network
- Create a Cosmos DB service endpoint
- Create a Cosmos DB account and import data
- Create an Azure Databricks cluster
- Query Cosmos DB from an Azure Databricks notebook

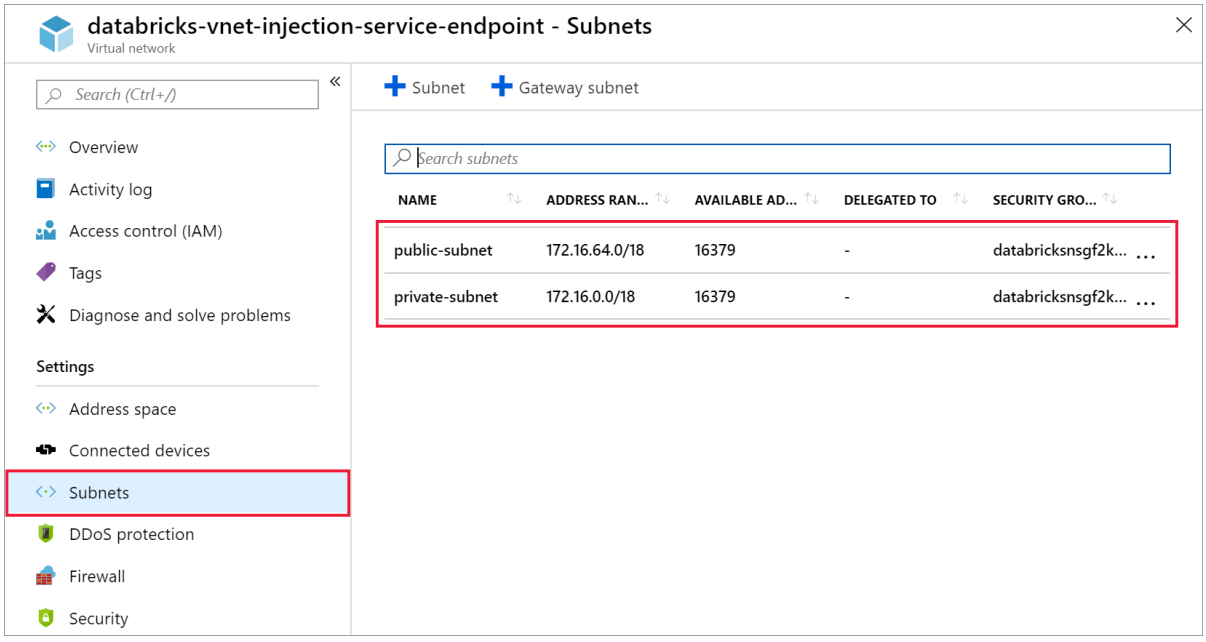
Prerequisites

Before you start, do the following:

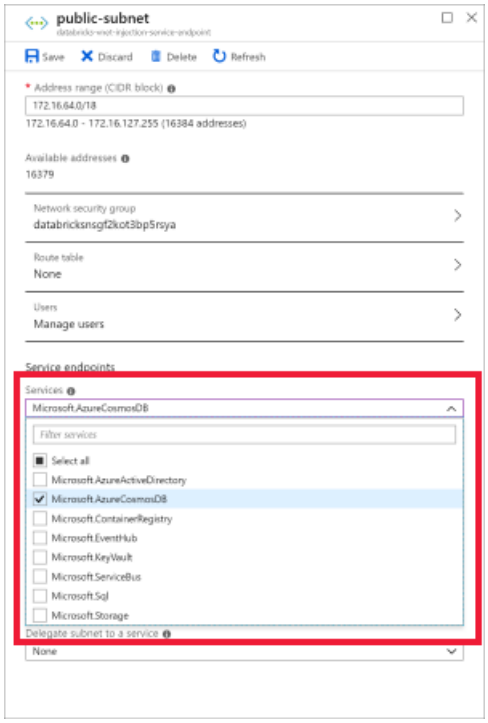
- Create an Azure Databricks workspace in a virtual network.
- Download the Spark connector.
- Download sample data from the NOAA National Centers for Environmental Information. Select a state or area and select **Search**. On the next page, accept the defaults and select **Search**. Then select **CSV Download** on the left side of the page to download the results.
- Download the pre-compiled binary of the Azure Cosmos DB Data Migration Tool.

Create a Cosmos DB service endpoint

1. Once you have deployed an Azure Databricks workspace to a virtual network, navigate to the virtual network in the Azure portal. Notice the public and private subnets that were created through the Databricks deployment.



2. Select the *public-subnet* and create a Cosmos DB service endpoint. Then **Save**.

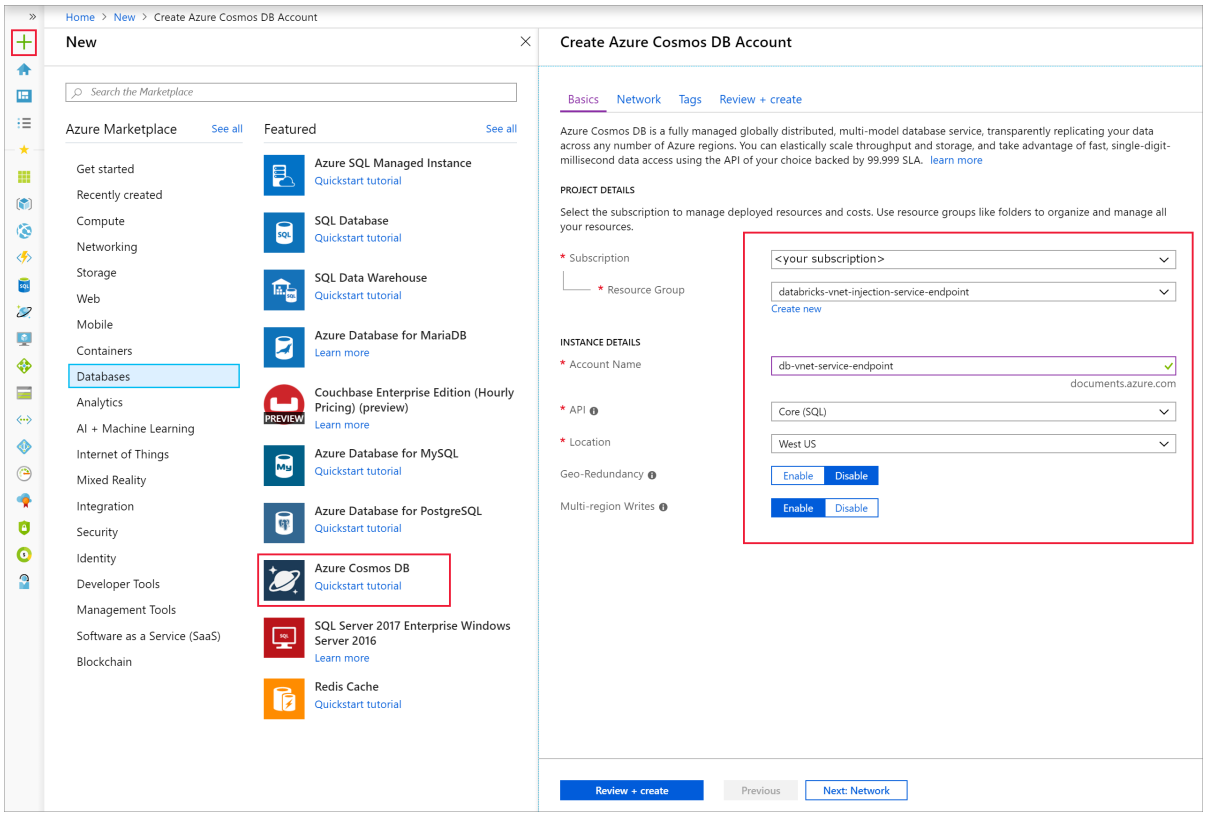


Create a Cosmos DB account

1. Open the Azure portal. On the upper-left side of the screen, select **Create a resource > Databases > Azure Cosmos DB**.

2. Fill out the **Instance Details** on the **Basics** tab with the following settings:

SETTING	VALUE
Subscription	*your subscription*
Resource Group	*your resource group*
Account Name	db-vnet-service-endpoint
API	Core (SQL)
Location	West US
Geo-Redundancy	Disable
Multi-region Writes	Enable

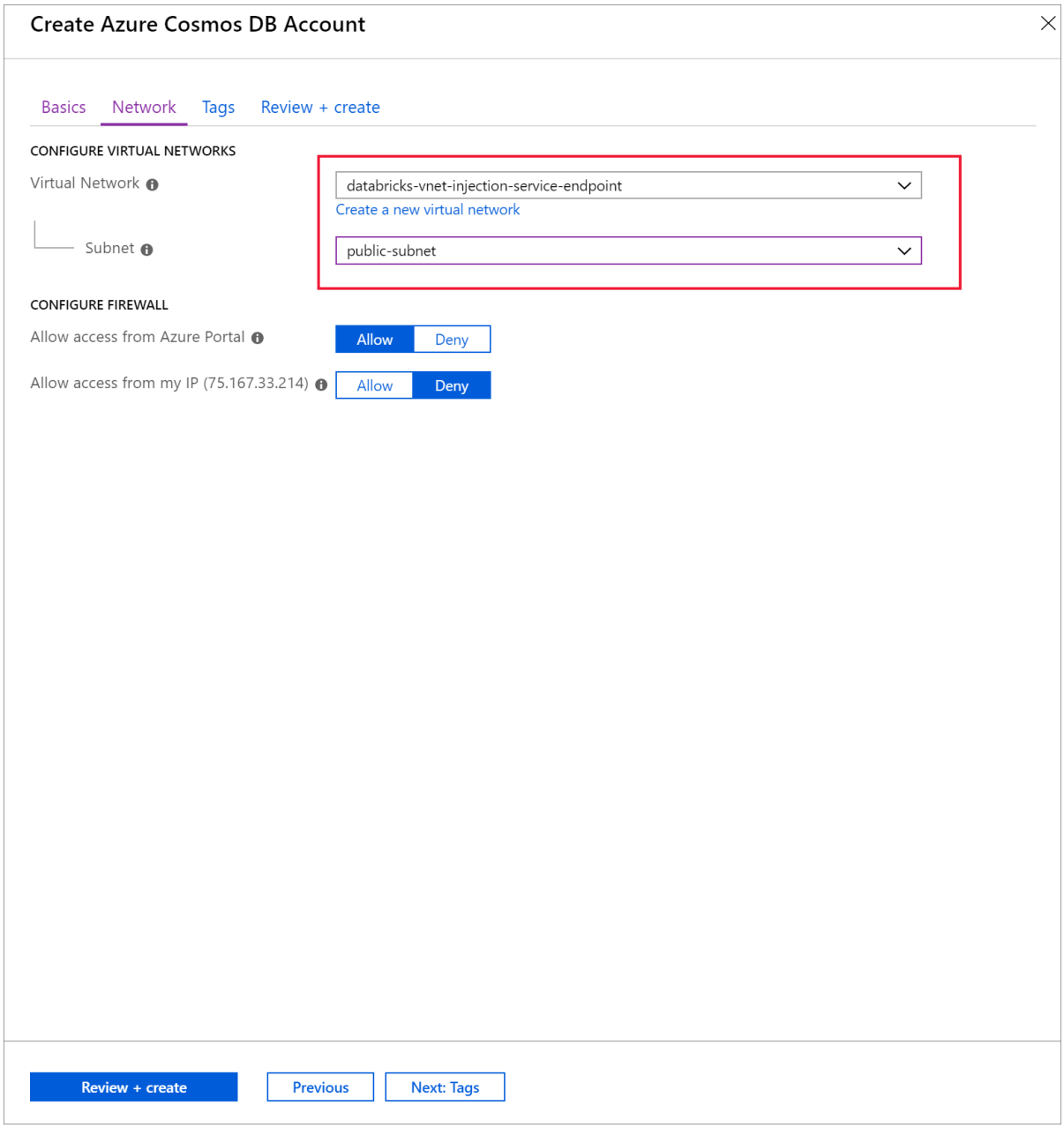


3. Select the **Network** tab and configure your virtual network.

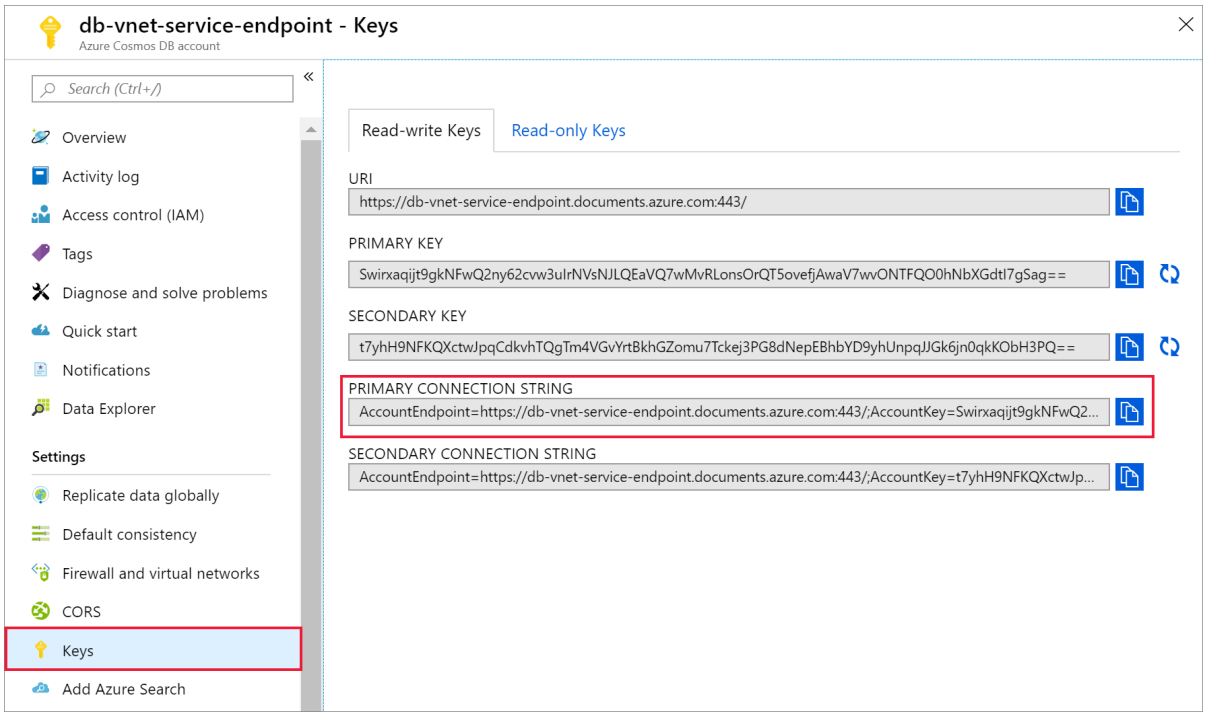
   a. Choose the virtual network you created as a prerequisite, and then select *public-subnet*. Notice that *private-subnet* has the note '*Microsoft AzureCosmosDB' endpoint is missing*'. This is because you only enabled the Cosmos DB service endpoint on the *public-subnet*.

   b. Ensure you have **Allow access from Azure portal** enabled. This setting allows you to access your Cosmos DB account from the Azure portal. If this option is set to **Deny**, you will receive errors when attempting to access your account.

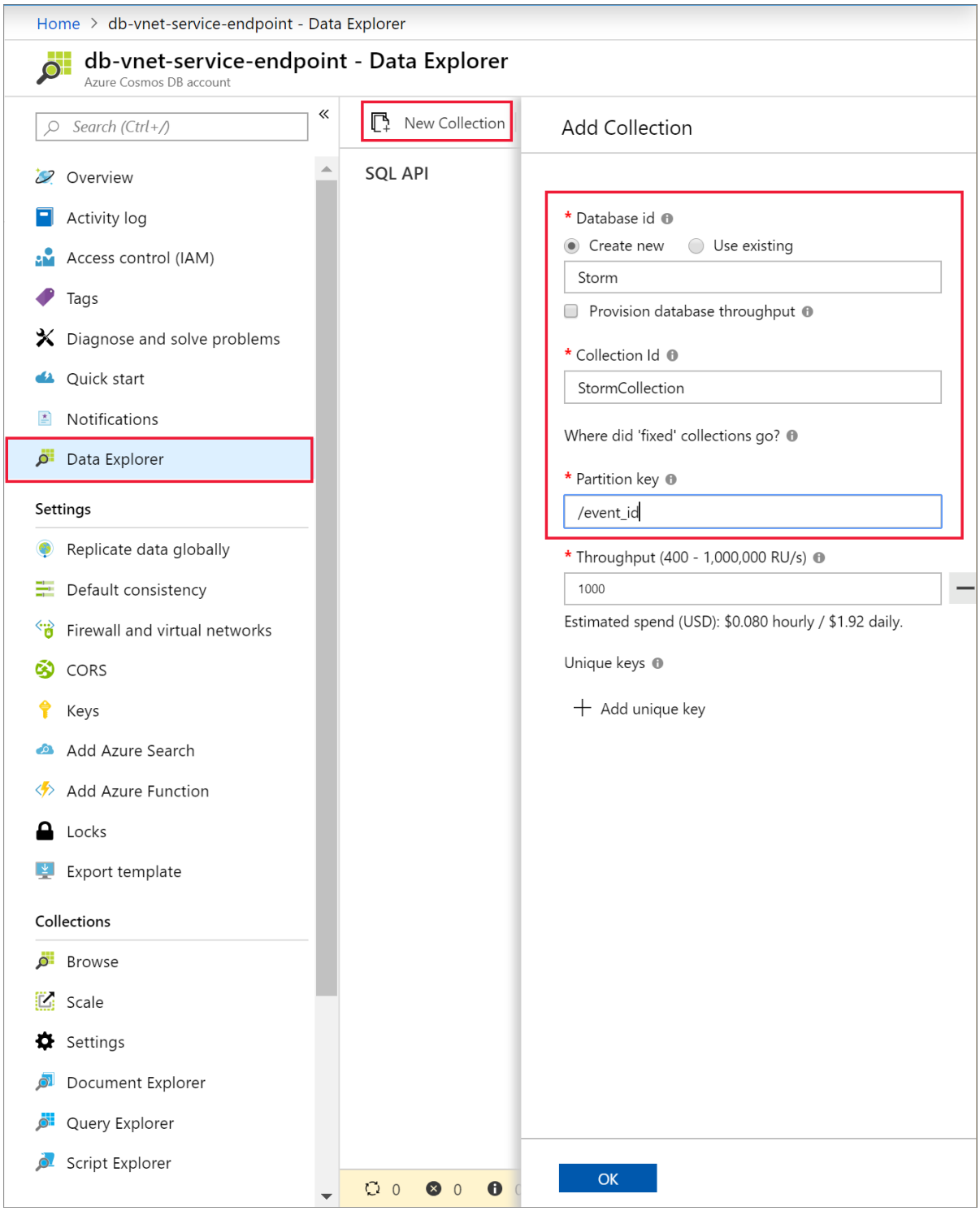
> **NOTE**
>
> It is not necessary for this tutorial, but you can also enable *Allow access from my IP* if you want the ability to access your Cosmos DB account from your local machine. For example, if you are connecting to your account using the Cosmos DB SDK, you need to enable this setting. If it is disabled, you will receive "Access Denied" errors.



4. Select **Review + Create**, and then **Create** to create your Cosmos DB account inside the virtual network.
5. Once your Cosmos DB account has been created, navigate to **Keys** under **Settings**. Copy the primary connection string and save it in a text editor for later use.

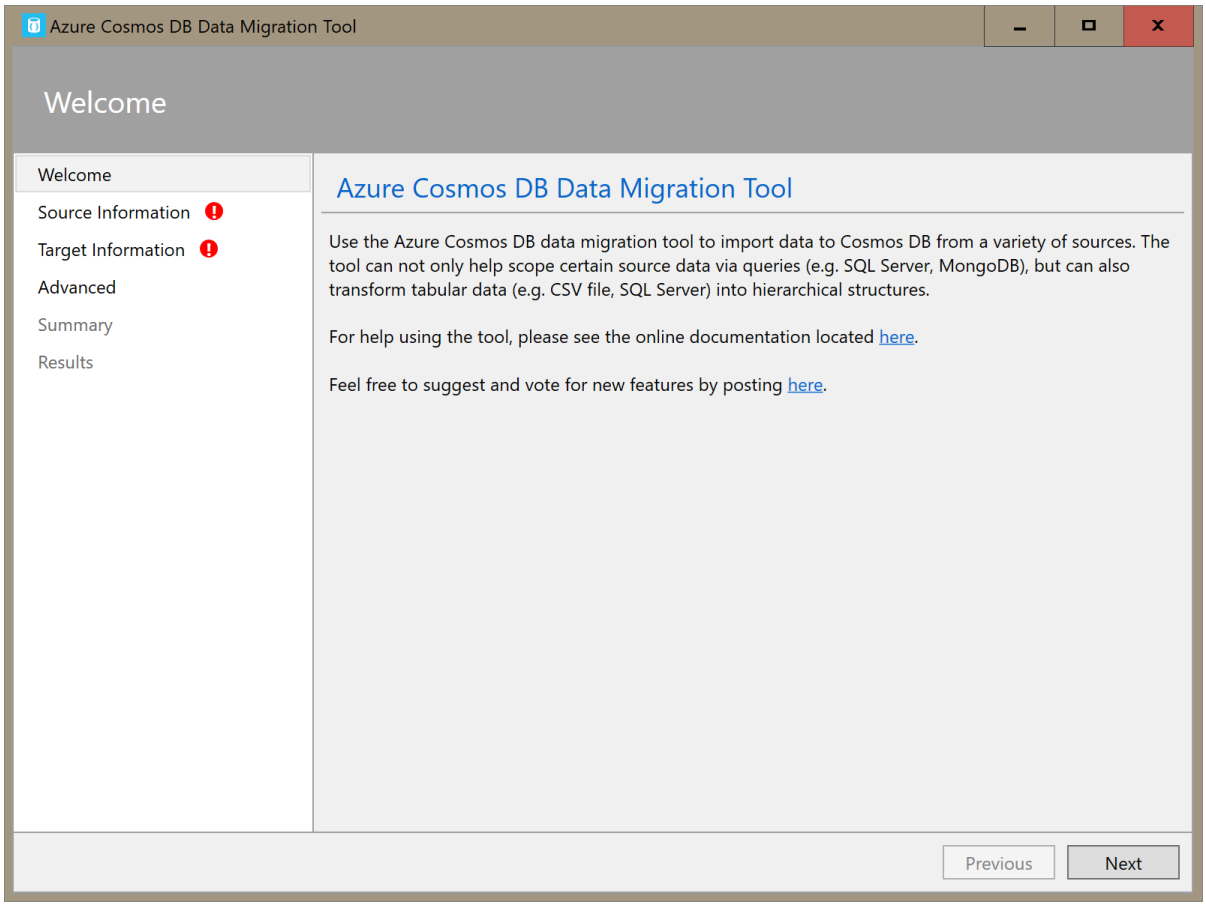


6. Select **Data Explorer** and **New Collection** to add a new database and collection to your Cosmos DB account.

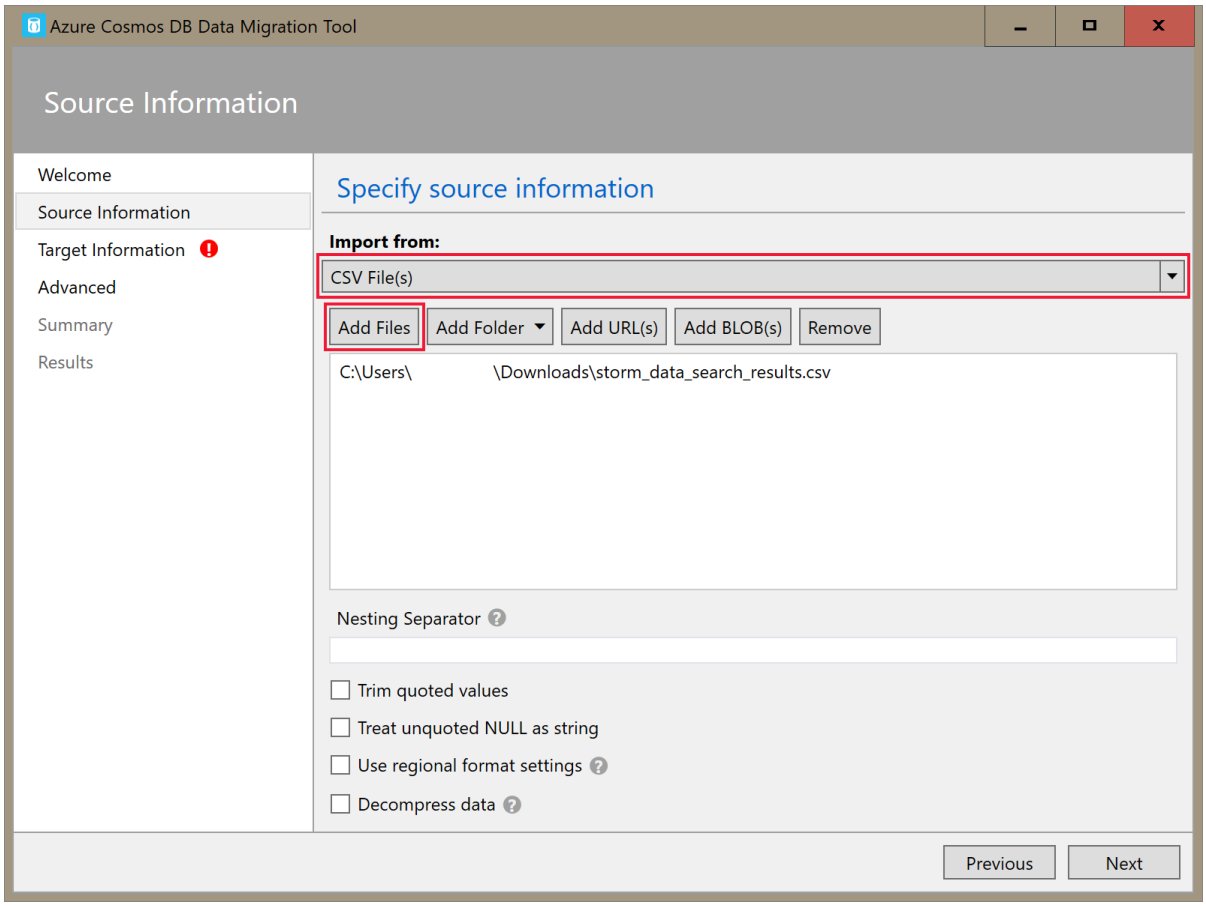


Upload data to Cosmos DB

1. Open the graphical interface version of the data migration tool for Cosmos DB, **Dtui.exe**.

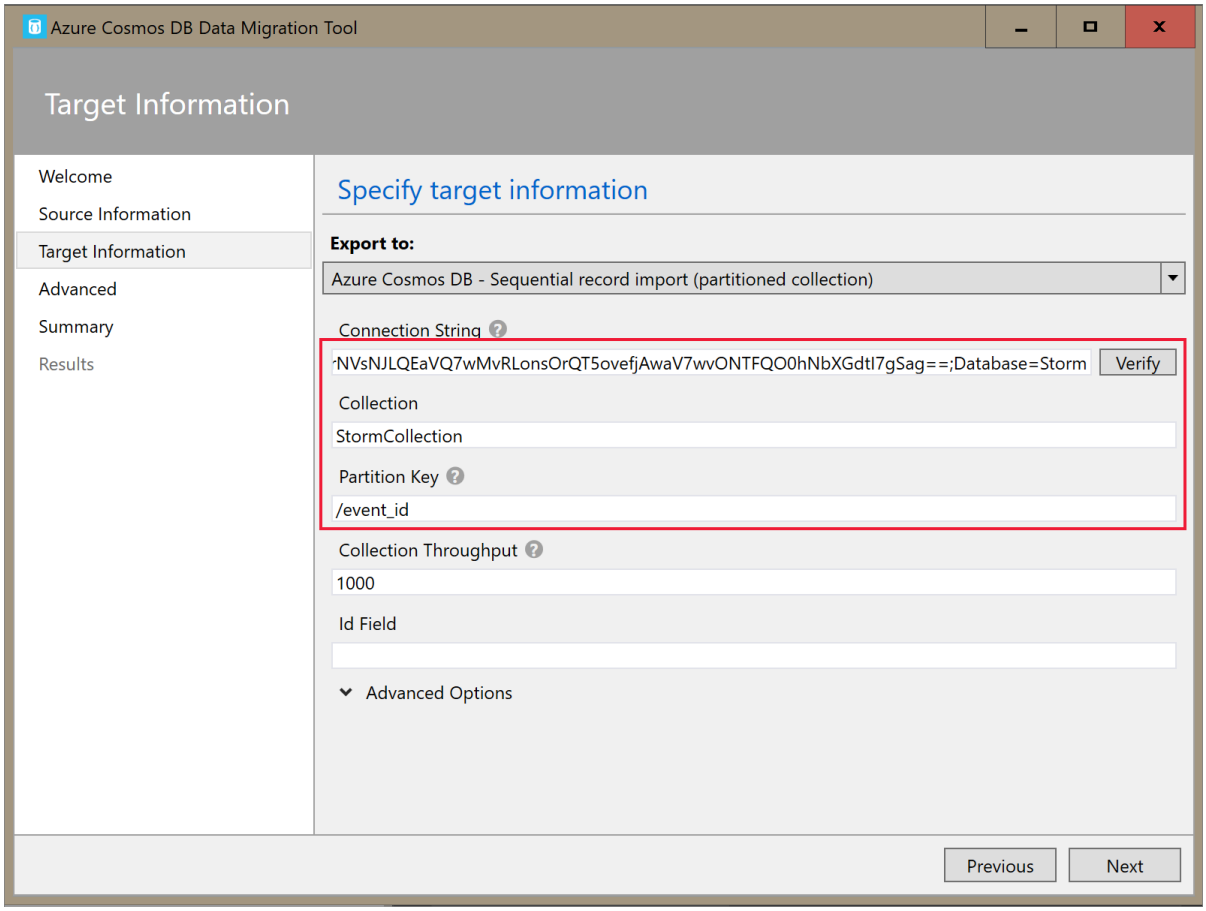


2. On the **Source Information** tab, select **CSV File(s)** in the **Import from** dropdown. Then select **Add Files** and add the storm data CSV you downloaded as a prerequisite.



3. On the **Target Information** tab, input your connection string. The connection string format is `AccountEndpoint=<URL>;AccountKey=<key>;Database=<database>`. The AccountEndpoint and AccountKey are included in the primary connection string you saved in the previous section. Append

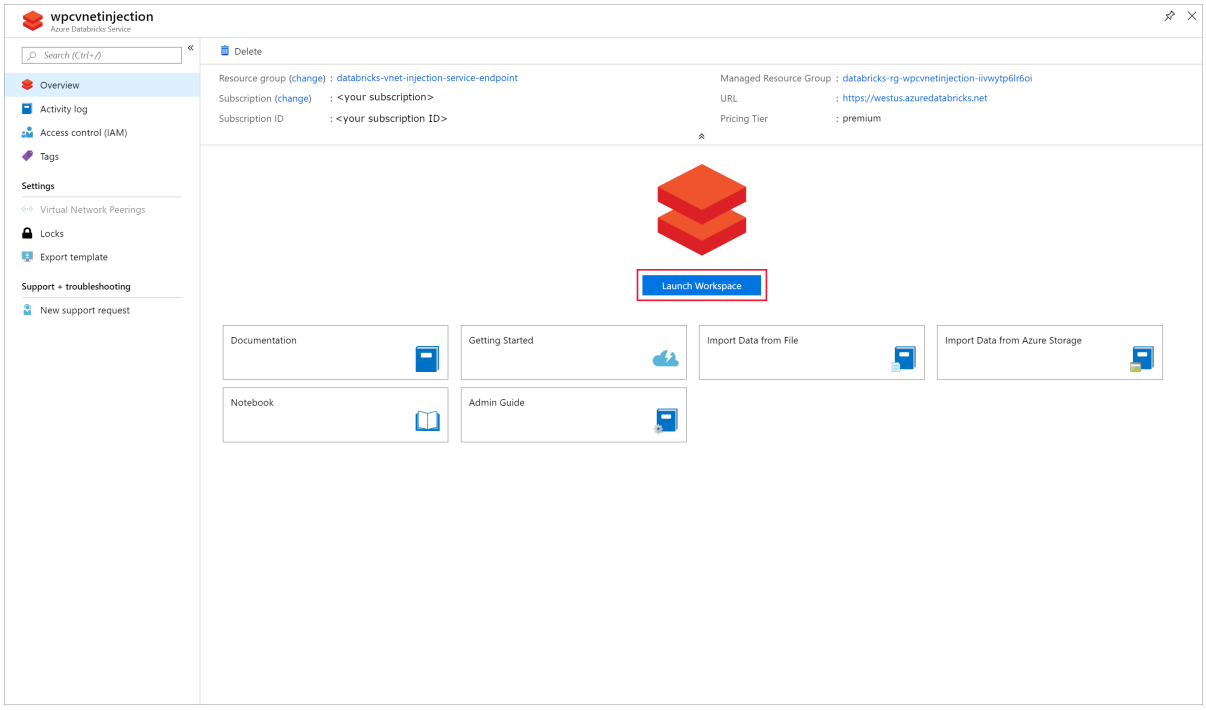
`Database=<your database name>` to the end of the connection string, and select **Verify**. Then, add the Collection name and partition key.



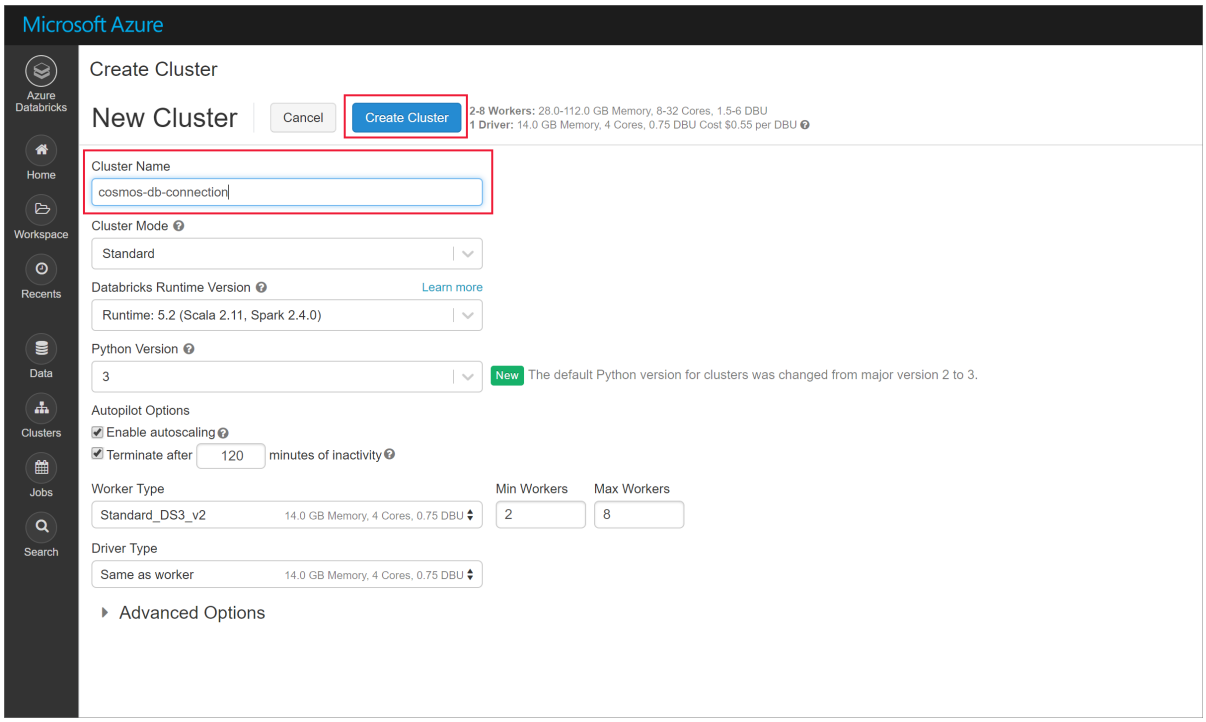
4. Select **Next** until you get to the Summary page. Then, select **Import**.

Create a cluster and add library

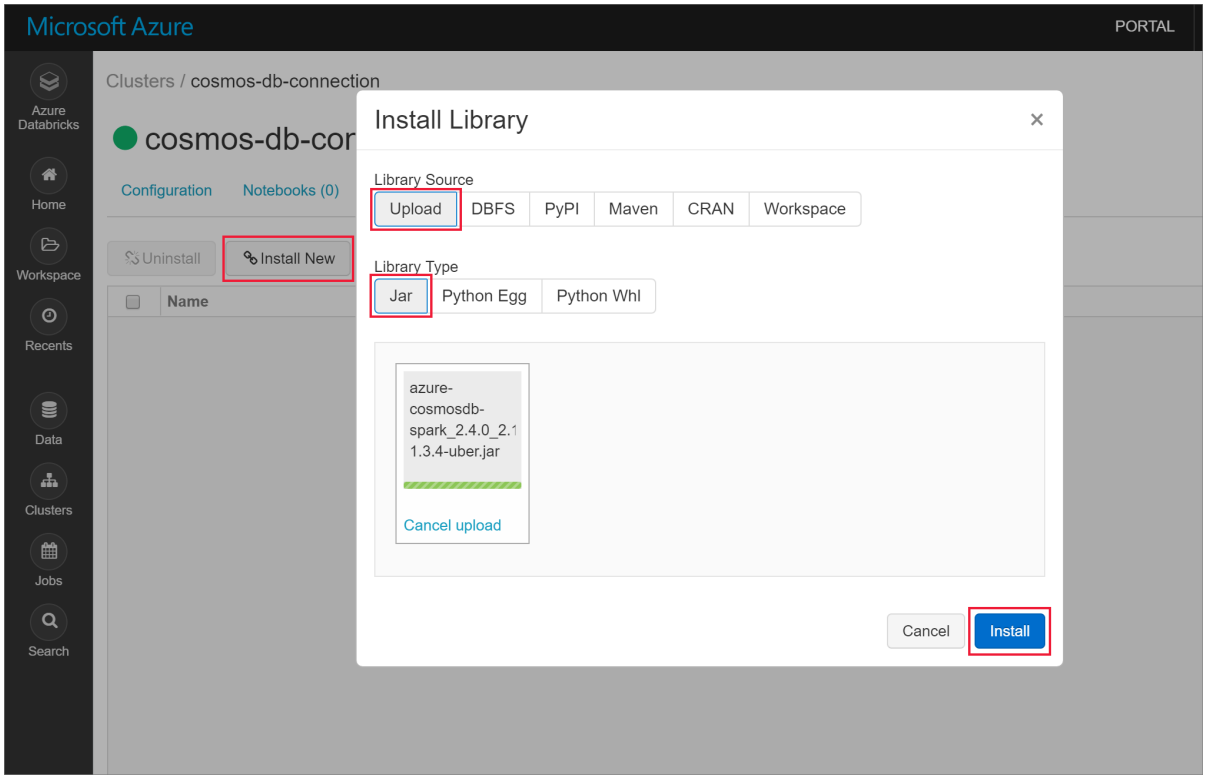
1. Navigate to your Azure Databricks service in the Azure portal and select **Launch Workspace**.



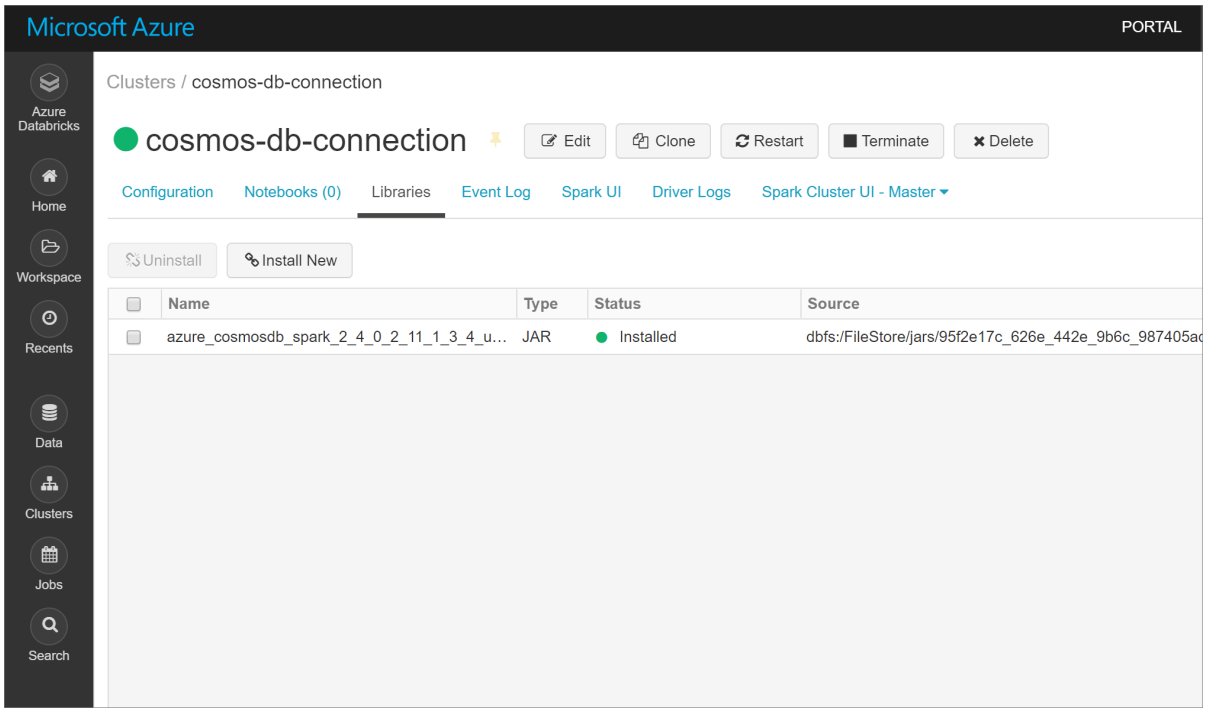
2. Create a new cluster. Choose a Cluster Name and accept the remaining default settings.



3. After your cluster is created, navigate to the cluster page and select the **Libraries** tab. Select **Install New** and upload the Spark connector jar file to install the library.

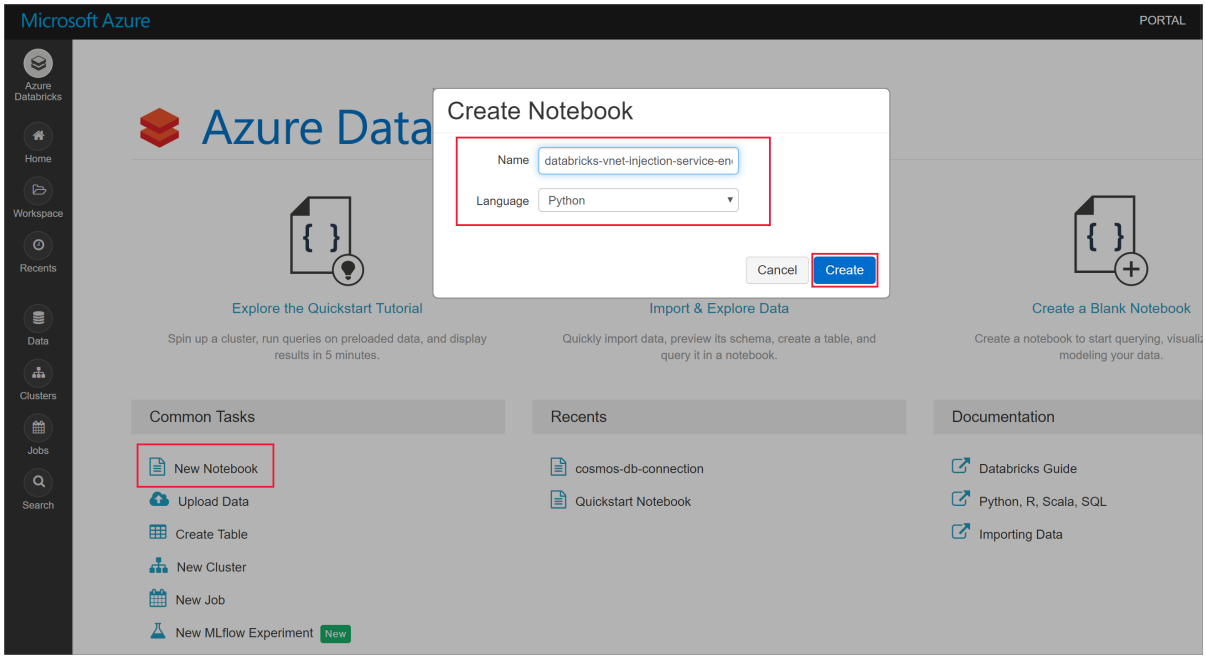


You can verify that the library was installed on the **Libraries** tab.



Query Cosmos DB from a Databricks notebook

1. Navigate to your Azure Databricks workspace and create a new python notebook.



2. Run the following python code to set the Cosmos DB connection configuration. Change the **Endpoint**, **Masterkey**, **Database**, and **Collection** accordingly.

```
connectionConfig = {
  "Endpoint" : "https://<your Cosmos DB account name.documents.azure.com:443/",
  "Masterkey" : "<your Cosmos DB primary key>",
  "Database" : "<your database name>",
  "preferredRegions" : "West US 2",
  "Collection": "<your collection name>",
  "schema_samplesize" : "1000",
  "query_pagesize" : "200000",
  "query_custom" : "SELECT * FROM c"
}
```

3. Use the following python code to load the data and create a temporary view.

```
users = spark.read.format("com.microsoft.azure.cosmosdb.spark").options(**connectionConfig).load()
users.createOrReplaceTempView("storm")
```

4. Use the following magic command to execute a SQL statement that returns data.

```
%sql
select * from storm
```

You have successfully connected your VNet-injected Databricks workspace to a service-endpoint enabled Cosmos DB resource. To read more about how to connect to Cosmos DB, see Azure Cosmos DB Connector for Apache Spark.

Clean up resources

When no longer needed, delete the resource group, the Azure Databricks workspace, and all related resources. Deleting the job avoids unnecessary billing. If you're planning to use the Azure Databricks workspace in future, you can stop the cluster and restart it later. If you are not going to continue to use this Azure Databricks workspace, delete all resources you created in this tutorial by using the following steps:

1. From the left-hand menu in the Azure portal, click **Resource groups** and then click the name of the resource group you created.
2. On your resource group page, select **Delete**, type the name of the resource to delete in the text box, and then select **Delete** again.

Next steps

In this tutorial, you've deployed an Azure Databricks workspace to a virtual network, and used the Cosmos DB Spark connector to query Cosmos DB data from Databricks. To learn more about working with Azure Databricks in a virtual network, continue to the tutorial for using SQL Server with Azure Databricks.

Tutorial: Query a SQL Server Linux Docker container in a virtual network from an Azure Databricks notebook

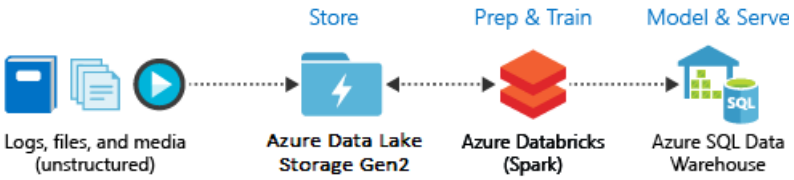
Tutorial: Extract, transform, and load data by using Azure Databricks

12/23/2019 • 12 minutes to read • Edit Online

In this tutorial, you perform an ETL (extract, transform, and load data) operation by using Azure Databricks. You extract data from Azure Data Lake Storage Gen2 into Azure Databricks, run transformations on the data in Azure Databricks, and load the transformed data into Azure SQL Data Warehouse.

The steps in this tutorial use the SQL Data Warehouse connector for Azure Databricks to transfer data to Azure Databricks. This connector, in turn, uses Azure Blob Storage as temporary storage for the data being transferred between an Azure Databricks cluster and Azure SQL Data Warehouse.

The following illustration shows the application flow:



This tutorial covers the following tasks:

- Create an Azure Databricks service.
- Create a Spark cluster in Azure Databricks.
- Create a file system in the Data Lake Storage Gen2 account.
- Upload sample data to the Azure Data Lake Storage Gen2 account.
- Create a service principal.
- Extract data from the Azure Data Lake Storage Gen2 account.
- Transform data in Azure Databricks.
- Load data into Azure SQL Data Warehouse.

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

> **NOTE**
>
> This tutorial cannot be carried out using **Azure Free Trial Subscription**. If you have a free account, go to your profile and change your subscription to **pay-as-you-go**. For more information, see Azure free account. Then, remove the spending limit, and request a quota increase for vCPUs in your region. When you create your Azure Databricks workspace, you can select the **Trial (Premium - 14-Days Free DBUs)** pricing tier to give the workspace access to free Premium Azure Databricks DBUs for 14 days.

Prerequisites

Complete these tasks before you begin this tutorial:

- Create an Azure SQL data warehouse, create a server-level firewall rule, and connect to the server as a server admin. See Quickstart: Create and query an Azure SQL data warehouse in the Azure portal.
- Create a master key for the Azure SQL data warehouse. See Create a database master key.
- Create an Azure Blob storage account, and a container within it. Also, retrieve the access key to access the

storage account. See Quickstart: Upload, download, and list blobs with the Azure portal.

- Create an Azure Data Lake Storage Gen2 storage account. See Quickstart: Create an Azure Data Lake Storage Gen2 storage account.
- Create a service principal. See How to: Use the portal to create an Azure AD application and service principal that can access resources.

  There's a couple of specific things that you'll have to do as you perform the steps in that article.

  - When performing the steps in the Assign the application to a role section of the article, make sure to assign the **Storage Blob Data Contributor** role to the service principal in the scope of the Data Lake Storage Gen2 account. If you assign the role to the parent resource group or subscription, you'll receive permissions-related errors until those role assignments propagate to the storage account.

    If you'd prefer to use an access control list (ACL) to associate the service principal with a specific file or directory, reference Access control in Azure Data Lake Storage Gen2.

  - When performing the steps in the Get values for signing in section of the article, paste the tenant ID, app ID, and password values into a text file. You'll need those soon.
- Sign in to the Azure portal.

Gather the information that you need

Make sure that you complete the prerequisites of this tutorial.

Before you begin, you should have these items of information:

✔ The database name, database server name, user name, and password of your Azure SQL Data warehouse.

✔ The access key of your blob storage account.

✔ The name of your Data Lake Storage Gen2 storage account.

✔ The tenant ID of your subscription.

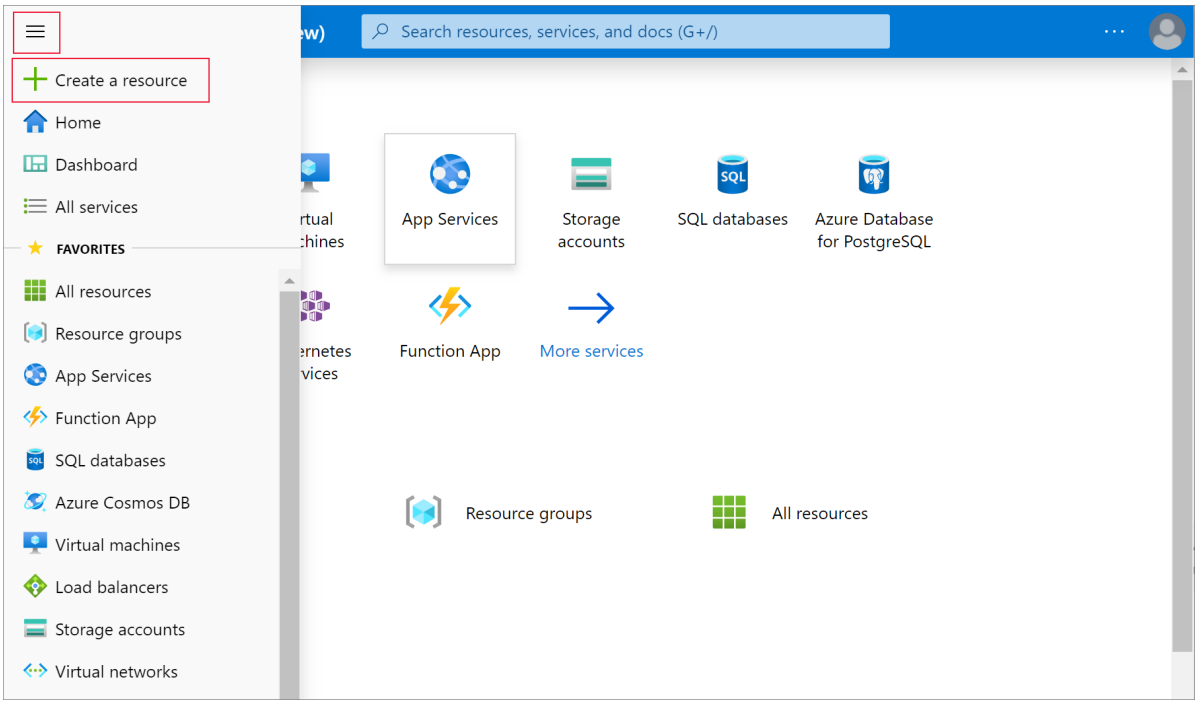
✔ The application ID of the app that you registered with Azure Active Directory (Azure AD).

✔ The authentication key for the app that you registered with Azure AD.

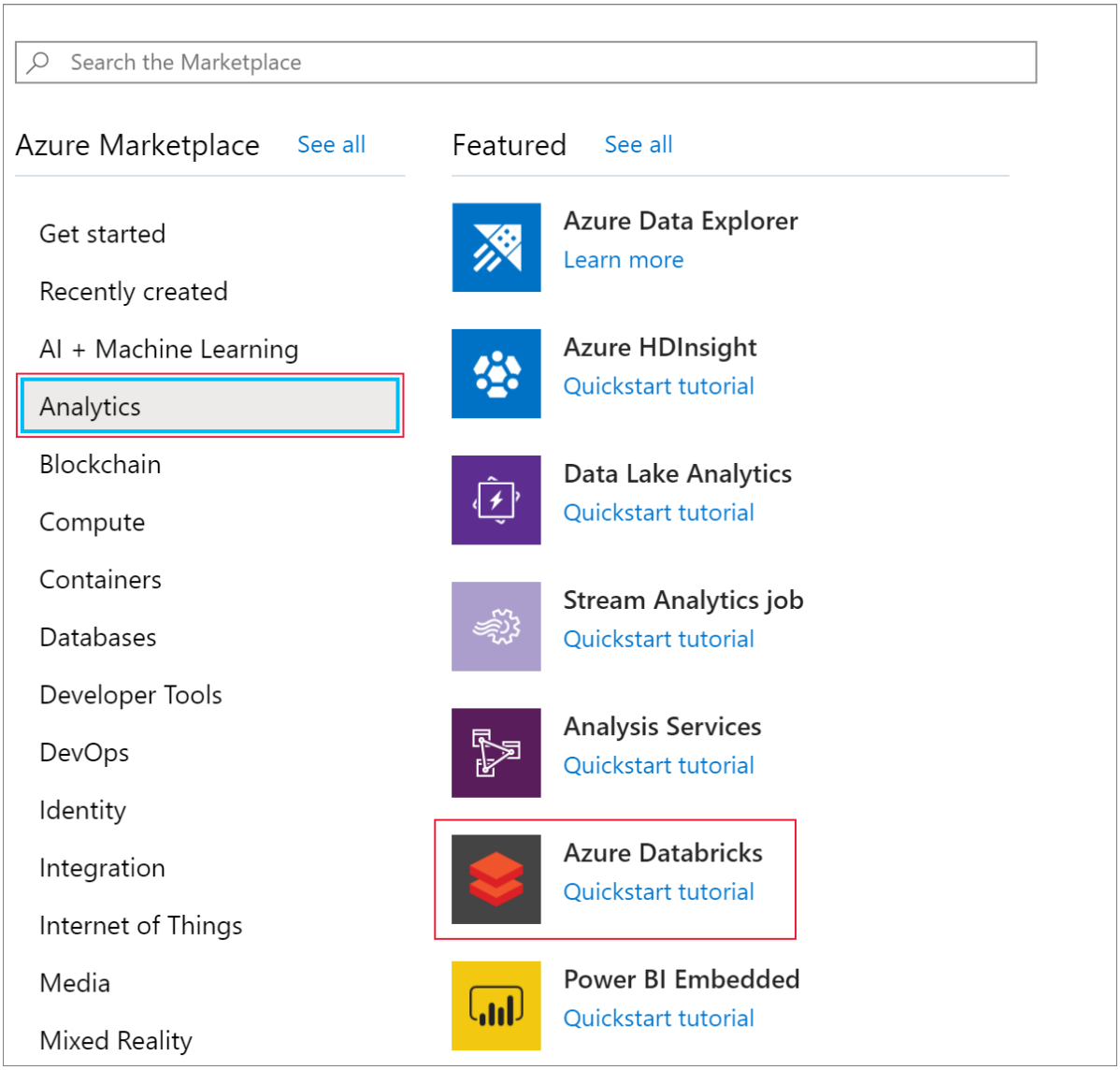
Create an Azure Databricks service

In this section, you create an Azure Databricks service by using the Azure portal.

1. From the Azure portal menu, select **Create a resource**.



Then, select **Analytics** > **Azure Databricks**.



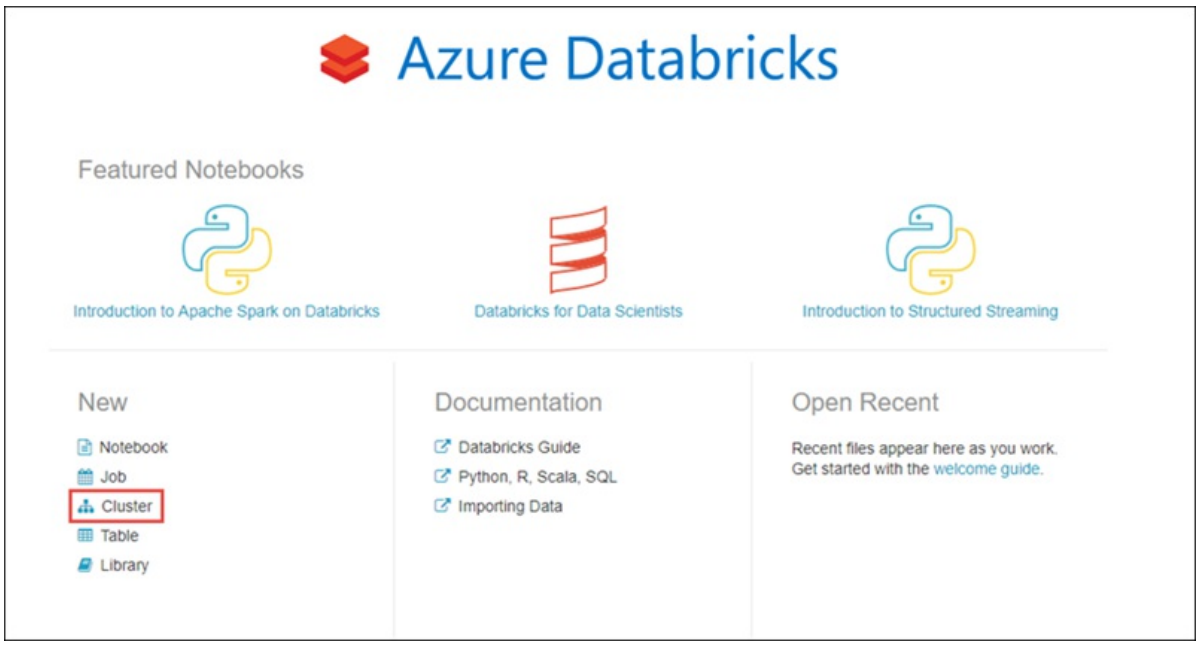
2. Under **Azure Databricks Service**, provide the following values to create a Databricks service:

PROPERTY	DESCRIPTION
Workspace name	Provide a name for your Databricks workspace.
Subscription	From the drop-down, select your Azure subscription.
Resource group	Specify whether you want to create a new resource group or use an existing one. A resource group is a container that holds related resources for an Azure solution. For more information, see Azure Resource Group overview.
Location	Select **West US 2**. For other available regions, see Azure services available by region.
Pricing Tier	Select **Standard**.

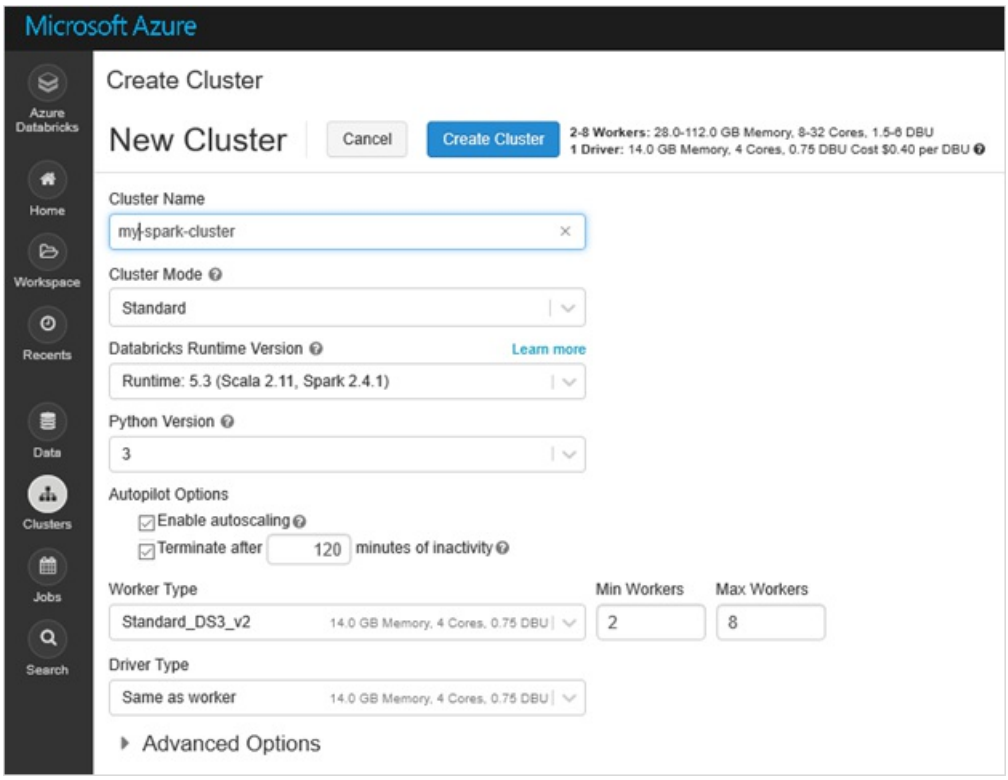
3. The account creation takes a few minutes. To monitor the operation status, view the progress bar at the top.
4. Select **Pin to dashboard** and then select **Create**.

Create a Spark cluster in Azure Databricks

1. In the Azure portal, go to the Databricks service that you created, and select **Launch Workspace**.
2. You're redirected to the Azure Databricks portal. From the portal, select **Cluster**.



3. In the **New cluster** page, provide the values to create a cluster.



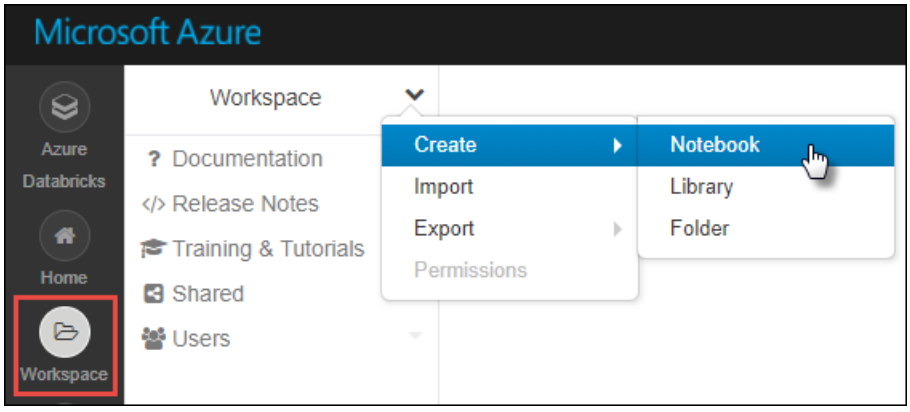
4. Fill in values for the following fields, and accept the default values for the other fields:

   - Enter a name for the cluster.
   - Make sure you select the **Terminate after __ minutes of inactivity** check box. If the cluster isn't being used, provide a duration (in minutes) to terminate the cluster.
   - Select **Create cluster**. After the cluster is running, you can attach notebooks to the cluster and run Spark jobs.

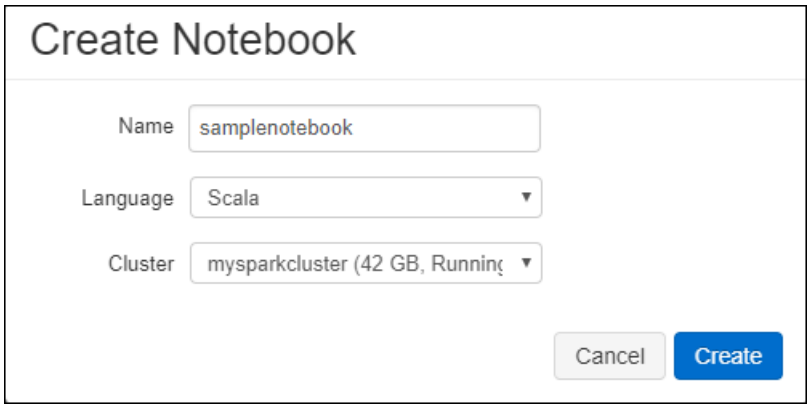
Create a file system in the Azure Data Lake Storage Gen2 account

In this section, you create a notebook in Azure Databricks workspace and then run code snippets to configure the storage account

1. In the Azure portal, go to the Azure Databricks service that you created, and select **Launch Workspace**.
2. On the left, select **Workspace**. From the **Workspace** drop-down, select **Create** > **Notebook**.



3. In the **Create Notebook** dialog box, enter a name for the notebook. Select **Scala** as the language, and then select the Spark cluster that you created earlier.



4. Select **Create**.

5. The following code block sets default service principal credentials for any ADLS Gen 2 account accessed in the Spark session. The second code block appends the account name to the setting to specify credentials for a specific ADLS Gen 2 account. Copy and paste either code block into the first cell of your Azure Databricks notebook.

   Session configuration

```
val appID = "<appID>"
val password = "<password>"
val tenantID = "<tenant-id>"

spark.conf.set("fs.azure.account.auth.type", "OAuth")
spark.conf.set("fs.azure.account.oauth.provider.type",
"org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider")
spark.conf.set("fs.azure.account.oauth2.client.id", "<appID>")
spark.conf.set("fs.azure.account.oauth2.client.secret", "<password>")
spark.conf.set("fs.azure.account.oauth2.client.endpoint", "https://login.microsoftonline.com/<tenant-
id>/oauth2/token")
spark.conf.set("fs.azure.createRemoteFileSystemDuringInitialization", "true")
```

   Account configuration

```
val storageAccountName = "<storage-account-name>"
val appID = "<app-id>"
val password = "<password>"
val fileSystemName = "<file-system-name>"
val tenantID = "<tenant-id>"

spark.conf.set("fs.azure.account.auth.type." + storageAccountName + ".dfs.core.windows.net", "OAuth")
spark.conf.set("fs.azure.account.oauth.provider.type." + storageAccountName + ".dfs.core.windows.net",
"org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider")
spark.conf.set("fs.azure.account.oauth2.client.id." + storageAccountName + ".dfs.core.windows.net", ""
+ appID + "")
spark.conf.set("fs.azure.account.oauth2.client.secret." + storageAccountName + ".dfs.core.windows.net",
"" + password + "")
spark.conf.set("fs.azure.account.oauth2.client.endpoint." + storageAccountName +
".dfs.core.windows.net", "https://login.microsoftonline.com/" + tenantID + "/oauth2/token")
spark.conf.set("fs.azure.createRemoteFileSystemDuringInitialization", "true")
dbutils.fs.ls("abfss://" + fileSystemName  + "@" + storageAccountName + ".dfs.core.windows.net/")
spark.conf.set("fs.azure.createRemoteFileSystemDuringInitialization", "false")
```

6. In this code block, replace the `<app-id>`, `<password>`, `<tenant-id>`, and `<storage-account-name>` placeholder values in this code block with the values that you collected while completing the prerequisites of this tutorial. Replace the `<file-system-name>` placeholder value with whatever name you want to give the file system.

- The `<app-id>`, and `<password>` are from the app that you registered with active directory as part of creating a service principal.
- The `<tenant-id>` is from your subscription.
- The `<storage-account-name>` is the name of your Azure Data Lake Storage Gen2 storage account.

7. Press the **SHIFT + ENTER** keys to run the code in this block.

Ingest sample data into the Azure Data Lake Storage Gen2 account

Before you begin with this section, you must complete the following prerequisites:

Enter the following code into a notebook cell:

```
%sh wget -P /tmp
https://raw.githubusercontent.com/Azure/usql/master/Examples/Samples/Data/json/radiowebsite/small_radio_json.j
son
```

In the cell, press **SHIFT + ENTER** to run the code.

Now in a new cell below this one, enter the following code, and replace the values that appear in brackets with the same values you used earlier:

```
dbutils.fs.cp("file:///tmp/small_radio_json.json", "abfss://" + fileSystemName + "@" + storageAccountName +
".dfs.core.windows.net/")
```

In the cell, press **SHIFT + ENTER** to run the code.

Extract data from the Azure Data Lake Storage Gen2 account

1. You can now load the sample json file as a data frame in Azure Databricks. Paste the following code in a new cell. Replace the placeholders shown in brackets with your values.

   ```
   val df = spark.read.json("abfss://<file-system-name>@<storage-account-
   name>.dfs.core.windows.net/small_radio_json.json")
   ```

2. Press the **SHIFT + ENTER** keys to run the code in this block.

3. Run the following code to see the contents of the data frame:

   ```
   df.show()
   ```

   You see an output similar to the following snippet:

```
+---------------------+---------+---------+------+-------------+----------+---------+-------+----------
----------+------+--------+-------------+---------+--------------------+------+-------------+------+
|               artist|     auth|firstName|gender|itemInSession|  lastName|   length|  level|
location|method|    page| registration|sessionId|                song|status|           ts|userId|
+---------------------+---------+---------+------+-------------+----------+---------+-------+----------
----------+------+--------+-------------+---------+--------------------+------+-------------+------+
| El Arrebato         |Logged In| Annalyse|     F|            2|Montgomery|234.57914| free  |  Killeen-
Temple, TX|   PUT|NextSong|1384448062332|     1879|Quiero Quererte Q...|   200|1409318650332|   309|
| Creedence Clearwa...|Logged In|   Dylann|     M|            9|    Thomas|340.87138| paid  |
Anchorage, AK|   PUT|NextSong|1400723739332|       10|        Born To Move|   200|1409318653332|    11|
| Gorillaz            |Logged In|     Liam|     M|           11|     Watts|246.17751| paid  |New York-
Newark-J...|   PUT|NextSong|1406279422332|     2047|               DARE|   200|1409318685332|   201|
...
...
```

You have now extracted the data from Azure Data Lake Storage Gen2 into Azure Databricks.

Transform data in Azure Databricks

The raw sample data **small_radio_json.json** file captures the audience for a radio station and has a variety of columns. In this section, you transform the data to only retrieve specific columns from the dataset.

1. First, retrieve only the columns **firstName**, **lastName**, **gender**, **location**, and **level** from the dataframe that you created.

```
val specificColumnsDf = df.select("firstname", "lastname", "gender", "location", "level")
specificColumnsDf.show()
```

You receive output as shown in the following snippet:

```
+---------+----------+------+--------------------+-----+
|firstname|  lastname|gender|            location|level|
+---------+----------+------+--------------------+-----+
| Annalyse|Montgomery|     F|  Killeen-Temple, TX| free|
|   Dylann|    Thomas|     M|       Anchorage, AK| paid|
|     Liam|     Watts|     M|New York-Newark-J...| paid|
|     Tess|  Townsend|     F|Nashville-Davidso...| free|
|  Margaux|     Smith|     F|Atlanta-Sandy Spr...| free|
|     Alan|     Morse|     M|Chicago-Napervill...| paid|
|Gabriella|   Shelton|     F|San Jose-Sunnyval...| free|
|   Elijah|  Williams|     M|Detroit-Warren-De...| paid|
|  Margaux|     Smith|     F|Atlanta-Sandy Spr...| free|
|     Tess|  Townsend|     F|Nashville-Davidso...| free|
|     Alan|     Morse|     M|Chicago-Napervill...| paid|
|     Liam|     Watts|     M|New York-Newark-J...| paid|
|     Liam|     Watts|     M|New York-Newark-J...| paid|
|   Dylann|    Thomas|     M|       Anchorage, AK| paid|
|     Alan|     Morse|     M|Chicago-Napervill...| paid|
|   Elijah|  Williams|     M|Detroit-Warren-De...| paid|
|  Margaux|     Smith|     F|Atlanta-Sandy Spr...| free|
|     Alan|     Morse|     M|Chicago-Napervill...| paid|
|   Dylann|    Thomas|     M|       Anchorage, AK| paid|
|  Margaux|     Smith|     F|Atlanta-Sandy Spr...| free|
+---------+----------+------+--------------------+-----+
```

2. You can further transform this data to rename the column **level** to **subscription_type**.

```
val renamedColumnsDF = specificColumnsDf.withColumnRenamed("level", "subscription_type")
renamedColumnsDF.show()
```

You receive output as shown in the following snippet.

```
+---------+----------+------+--------------------+-----------------+
|firstname|  lastname|gender|            location|subscription_type|
+---------+----------+------+--------------------+-----------------+
| Annalyse|Montgomery|     F|  Killeen-Temple, TX|             free|
|   Dylann|    Thomas|     M|       Anchorage, AK|             paid|
|     Liam|     Watts|     M|New York-Newark-J...|             paid|
|     Tess|  Townsend|     F|Nashville-Davidso...|             free|
|  Margaux|     Smith|     F|Atlanta-Sandy Spr...|             free|
|     Alan|     Morse|     M|Chicago-Napervill...|             paid|
|Gabriella|   Shelton|     F|San Jose-Sunnyval...|             free|
|   Elijah|  Williams|     M|Detroit-Warren-De...|             paid|
|  Margaux|     Smith|     F|Atlanta-Sandy Spr...|             free|
|     Tess|  Townsend|     F|Nashville-Davidso...|             free|
|     Alan|     Morse|     M|Chicago-Napervill...|             paid|
|     Liam|     Watts|     M|New York-Newark-J...|             paid|
|     Liam|     Watts|     M|New York-Newark-J...|             paid|
|   Dylann|    Thomas|     M|       Anchorage, AK|             paid|
|     Alan|     Morse|     M|Chicago-Napervill...|             paid|
|   Elijah|  Williams|     M|Detroit-Warren-De...|             paid|
|  Margaux|     Smith|     F|Atlanta-Sandy Spr...|             free|
|     Alan|     Morse|     M|Chicago-Napervill...|             paid|
|   Dylann|    Thomas|     M|       Anchorage, AK|             paid|
|  Margaux|     Smith|     F|Atlanta-Sandy Spr...|             free|
+---------+----------+------+--------------------+-----------------+
```

Load data into Azure SQL Data Warehouse

In this section, you upload the transformed data into Azure SQL Data Warehouse. You use the Azure SQL Data Warehouse connector for Azure Databricks to directly upload a dataframe as a table in a SQL data warehouse.

As mentioned earlier, the SQL Data Warehouse connector uses Azure Blob storage as temporary storage to upload data between Azure Databricks and Azure SQL Data Warehouse. So, you start by providing the configuration to connect to the storage account. You must already have already created the account as part of the prerequisites for this article.

1. Provide the configuration to access the Azure Storage account from Azure Databricks.

   ```
   val blobStorage = "<blob-storage-account-name>.blob.core.windows.net"
   val blobContainer = "<blob-container-name>"
   val blobAccessKey =  "<access-key>"
   ```

2. Specify a temporary folder to use while moving data between Azure Databricks and Azure SQL Data Warehouse.

   ```
   val tempDir = "wasbs://" + blobContainer + "@" + blobStorage +"/tempDirs"
   ```

3. Run the following snippet to store Azure Blob storage access keys in the configuration. This action ensures that you don't have to keep the access key in the notebook in plain text.

   ```
   val acntInfo = "fs.azure.account.key."+ blobStorage
   sc.hadoopConfiguration.set(acntInfo, blobAccessKey)
   ```

4. Provide the values to connect to the Azure SQL Data Warehouse instance. You must have created a SQL data warehouse as a prerequisite. Use the fully qualified server name for **dwServer**. For example, `<servername>.database.windows.net`.

```
//SQL Data Warehouse related settings
val dwDatabase = "<database-name>"
val dwServer = "<database-server-name>"
val dwUser = "<user-name>"
val dwPass = "<password>"
val dwJdbcPort =  "1433"
val dwJdbcExtraOptions =
"encrypt=true;trustServerCertificate=true;hostNameInCertificate=*.database.windows.net;loginTimeout=30;
"
val sqlDwUrl = "jdbc:sqlserver://" + dwServer + ":" + dwJdbcPort + ";database=" + dwDatabase + ";user="
+ dwUser+";password=" + dwPass + ";$dwJdbcExtraOptions"
val sqlDwUrlSmall = "jdbc:sqlserver://" + dwServer + ":" + dwJdbcPort + ";database=" + dwDatabase +
";user=" + dwUser+";password=" + dwPass
```

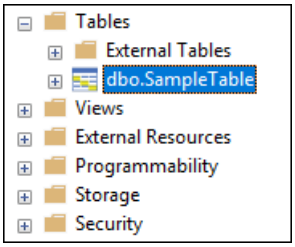
5. Run the following snippet to load the transformed dataframe, **renamedColumnsDF**, as a table in a SQL data warehouse. This snippet creates a table called **SampleTable** in the SQL database.

```
spark.conf.set(
    "spark.sql.parquet.writeLegacyFormat",
    "true")

renamedColumnsDF.write.format("com.databricks.spark.sqldw").option("url",
sqlDwUrlSmall).option("dbtable", "SampleTable")       .option(
"forward_spark_azure_storage_credentials","True").option("tempdir", tempDir).mode("overwrite").save()
```

> **NOTE**
>
> This sample uses the `forward_spark_azure_storage_credentials` flag, which causes SQL Data Warehouse to access data from blob storage using an Access Key. This is the only supported method of authentication.
>
> If your Azure Blob Storage is restricted to select virtual networks, SQL Data Warehouse requires Managed Service Identity instead of Access Keys. This will cause the error "This request is not authorized to perform this operation."

6. Connect to the SQL database and verify that you see a database named **SampleTable**.

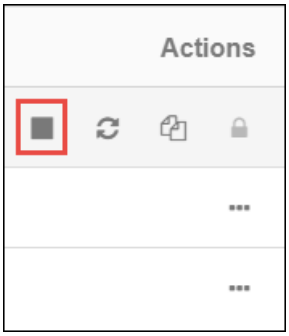


7. Run a select query to verify the contents of the table. The table should have the same data as the **renamedColumnsDF** dataframe.

	firstname	lastname	gender	location	subscription_type
1	Annalyse	Montgomery	F	Killeen-Temple, TX	free
2	Dylann	Thomas	M	Anchorage, AK	paid
3	Liam	Watts	M	New York-Newark-Jersey City, NY-NJ-PA	paid
4	Tess	Townsend	F	Nashville-Davidson--Murfreesboro--Franklin, TN	free
5	Margaux	Smith	F	Atlanta-Sandy Springs-Roswell, GA	free
6	Alan	Morse	M	Chicago-Naperville-Elgin, IL-IN-WI	paid
7	Gabriella	Shelton	F	San Jose-Sunnyvale-Santa Clara, CA	free
8	Elijah	Williams	M	Detroit-Warren-Dearborn, MI	paid
9	Margaux	Smith	F	Atlanta-Sandy Springs-Roswell, GA	free
10	Tess	Townsend	F	Nashville-Davidson--Murfreesboro--Franklin, TN	free
11	Alan	Morse	M	Chicago-Naperville-Elgin, IL-IN-WI	paid
12	Liam	Watts	M	New York-Newark-Jersey City, NY-NJ-PA	paid
13	Liam	Watts	M	New York-Newark-Jersey City, NY-NJ-PA	paid
14	Dylann	Thomas	M	Anchorage, AK	paid
15	Alan	Morse	M	Chicago-Naperville-Elgin, IL-IN-WI	paid
16	Elijah	Williams	M	Detroit-Warren-Dearborn, MI	paid
17	Margaux	Smith	F	Atlanta-Sandy Springs-Roswell, GA	free
18	Alan	Morse	M	Chicago-Naperville-Elgin, IL-IN-WI	paid
19	Dylann	Thomas	M	Anchorage, AK	paid
20	Margaux	Smith	F	Atlanta-Sandy Springs-Roswell, GA	free

Clean up resources

After you finish the tutorial, you can terminate the cluster. From the Azure Databricks workspace, select **Clusters** on the left. For the cluster to terminate, under **Actions**, point to the ellipsis (...) and select the **Terminate** icon.



If you don't manually terminate the cluster, it automatically stops, provided you selected the **Terminate after __ minutes of inactivity** check box when you created the cluster. In such a case, the cluster automatically stops if it's been inactive for the specified time.

Next steps

In this tutorial, you learned how to:

- Create an Azure Databricks service
- Create a Spark cluster in Azure Databricks
- Create a notebook in Azure Databricks
- Extract data from a Data Lake Storage Gen2 account
- Transform data in Azure Databricks
- Load data into Azure SQL Data Warehouse

Advance to the next tutorial to learn about streaming real-time data into Azure Databricks using Azure Event Hubs.

Stream data into Azure Databricks using Event Hubs

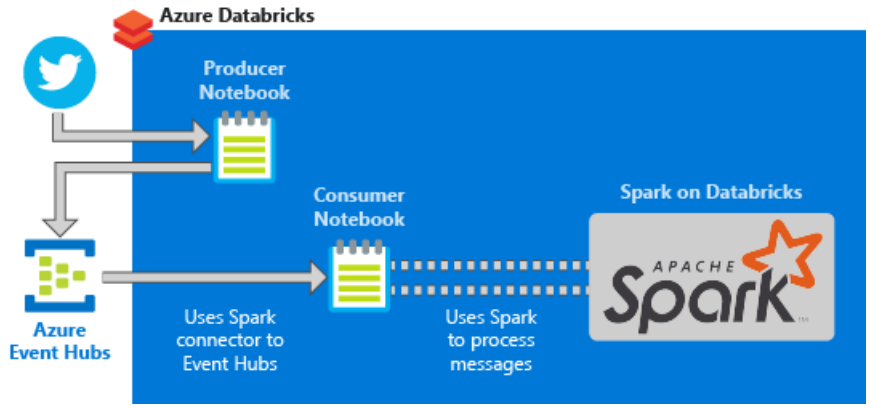
Tutorial: Stream data into Azure Databricks using Event Hubs

12/23/2019 • 12 minutes to read • Edit Online

In this tutorial, you connect a data ingestion system with Azure Databricks to stream data into an Apache Spark cluster in near real-time. You set up data ingestion system using Azure Event Hubs and then connect it to Azure Databricks to process the messages coming through. To access a stream of data, you use Twitter APIs to ingest tweets into Event Hubs. Once you have the data in Azure Databricks, you can run analytical jobs to further analyze the data.

By the end of this tutorial, you would have streamed tweets from Twitter (that have the term "Azure" in them) and read the tweets in Azure Databricks.

The following illustration shows the application flow:



This tutorial covers the following tasks:

- Create an Azure Databricks workspace
- Create a Spark cluster in Azure Databricks
- Create a Twitter app to access streaming data
- Create notebooks in Azure Databricks
- Attach libraries for Event Hubs and Twitter API
- Send tweets to Event Hubs
- Read tweets from Event Hubs

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

> **NOTE**
>
> This tutorial cannot be carried out using **Azure Free Trial Subscription**. If you have a free account, go to your profile and change your subscription to **pay-as-you-go**. For more information, see Azure free account. Then, remove the spending limit, and request a quota increase for vCPUs in your region. When you create your Azure Databricks workspace, you can select the **Trial (Premium - 14-Days Free DBUs)** pricing tier to give the workspace access to free Premium Azure Databricks DBUs for 14 days.

Prerequisites

Before you start with this tutorial, make sure to meet the following requirements:

- An Azure Event Hubs namespace.
- An Event Hub within the namespace.
- Connection string to access the Event Hubs namespace. The connection string should have a format similar to `Endpoint=sb://<namespace>.servicebus.windows.net/;SharedAccessKeyName=<key name>;SharedAccessKey=<key value>`.
- Shared access policy name and policy key for Event Hubs.

You can meet these requirements by completing the steps in the article, Create an Azure Event Hubs namespace and event hub.

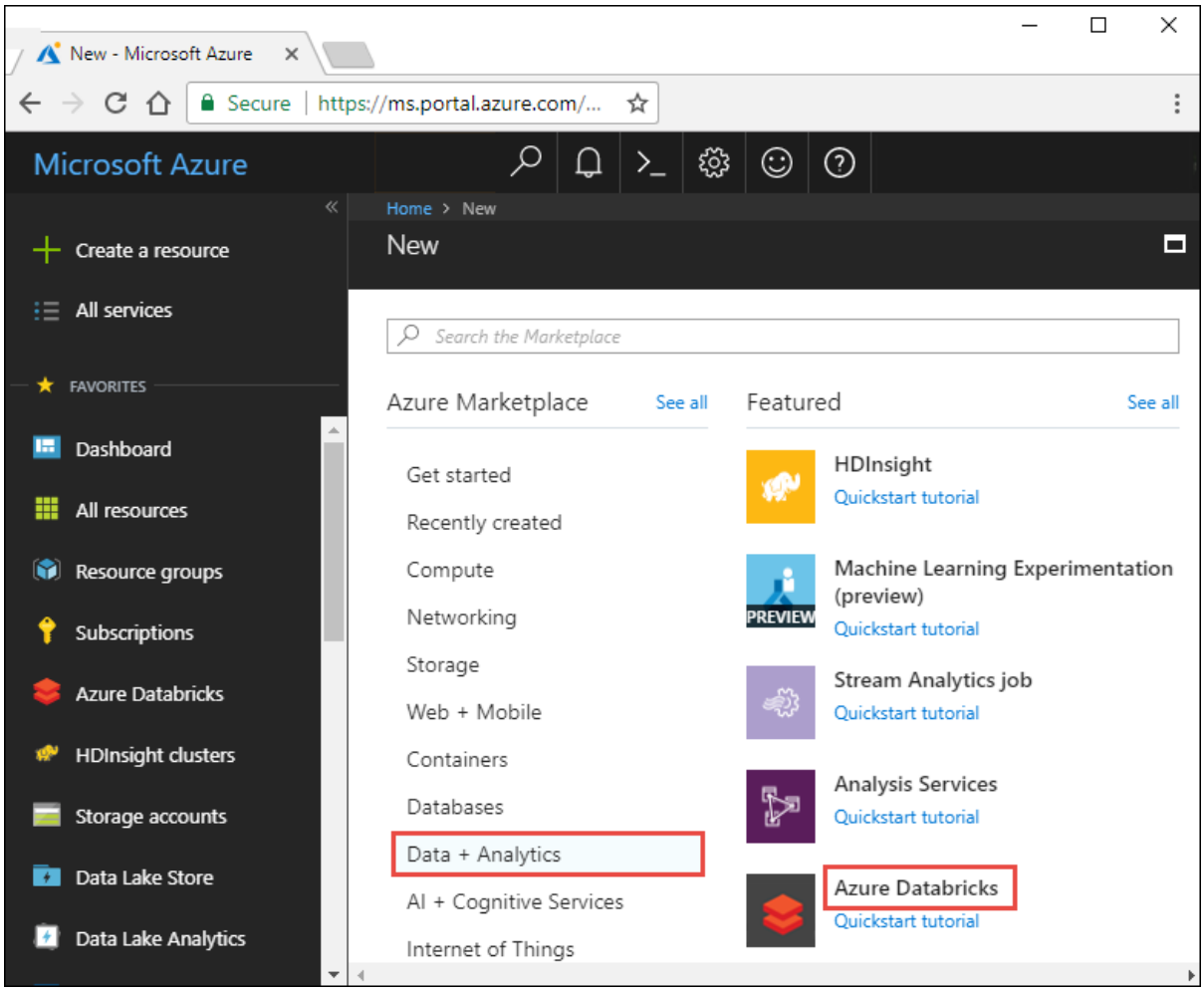
Sign in to the Azure portal

Sign in to the Azure portal.

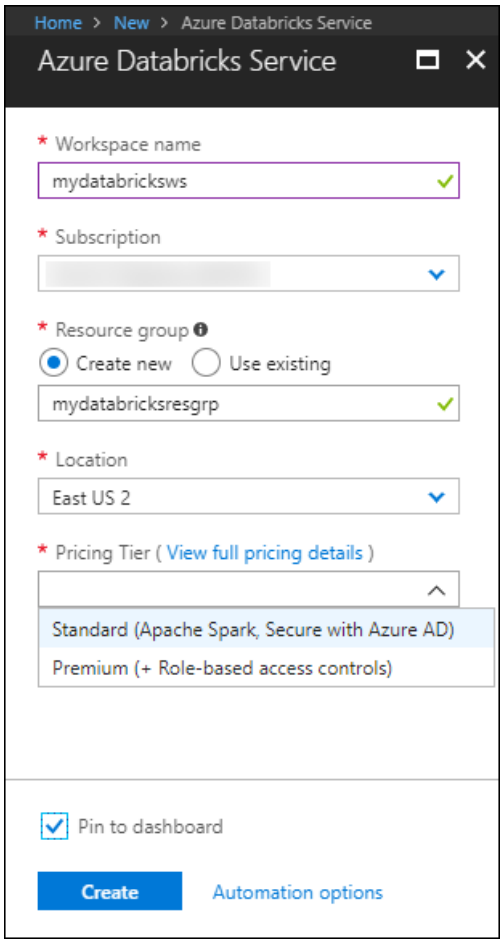
Create an Azure Databricks workspace

In this section, you create an Azure Databricks workspace using the Azure portal.

1. In the Azure portal, select **Create a resource** > **Data + Analytics** > **Azure Databricks**.



2. Under **Azure Databricks Service**, provide the values to create a Databricks workspace.

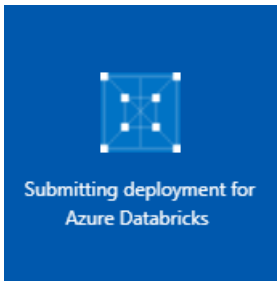


Provide the following values:

PROPERTY	DESCRIPTION
Workspace name	Provide a name for your Databricks workspace
Subscription	From the drop-down, select your Azure subscription.
Resource group	Specify whether you want to create a new resource group or use an existing one. A resource group is a container that holds related resources for an Azure solution. For more information, see Azure Resource Group overview.
Location	Select **East US 2**. For other available regions, see Azure services available by region.
Pricing Tier	Choose between **Standard** or **Premium**. For more information on these tiers, see Databricks pricing page.

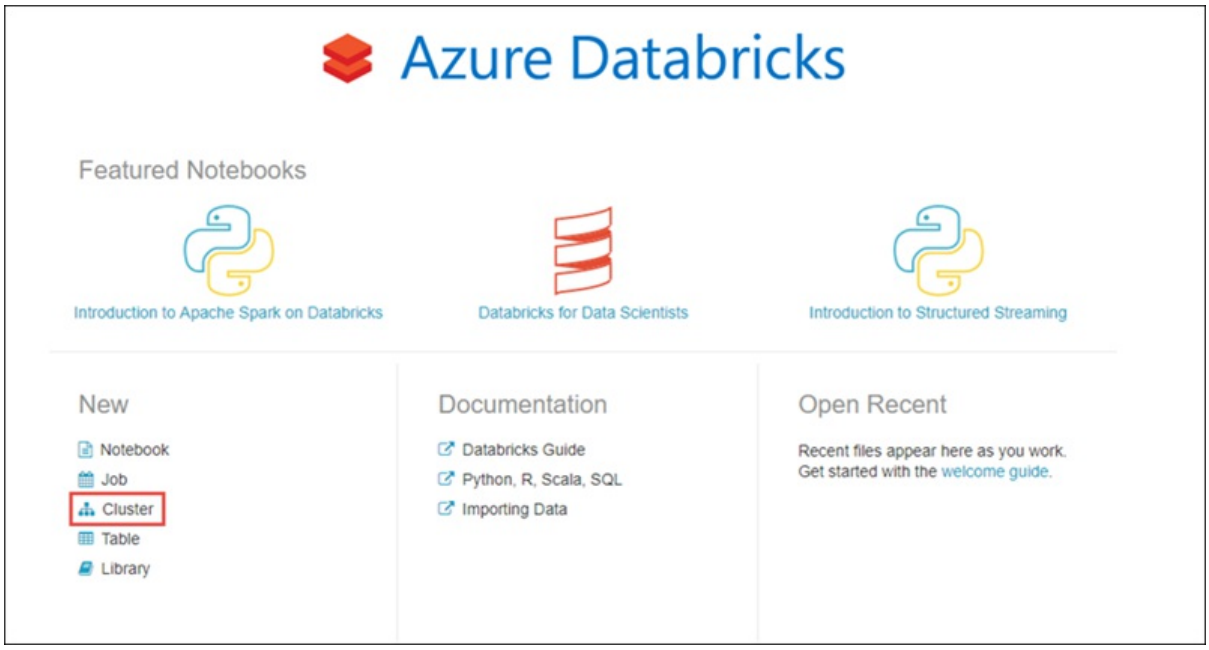
Select **Pin to dashboard** and then select **Create**.

3. The account creation takes a few minutes. During account creation, the portal displays the **Submitting deployment for Azure Databricks** tile on the right side. You may need to scroll right on your dashboard to see the tile. There is also a progress bar displayed near the top of the screen. You can watch either area for progress.

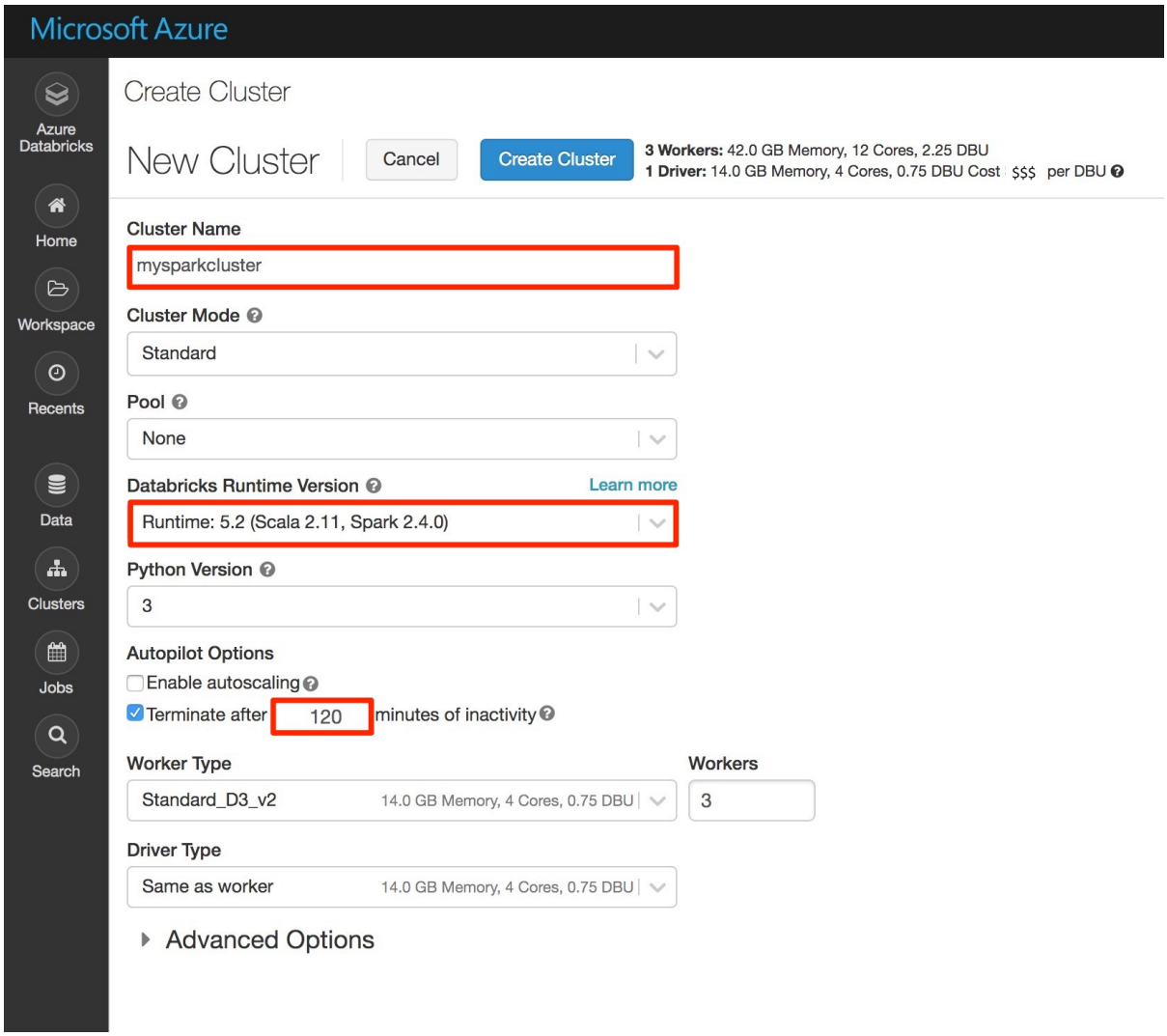


Create a Spark cluster in Databricks

1. In the Azure portal, go to the Databricks workspace that you created, and then select **Launch Workspace**.
2. You are redirected to the Azure Databricks portal. From the portal, select **Cluster**.



3. In the **New cluster** page, provide the values to create a cluster.



Accept all other default values other than the following:

- Enter a name for the cluster.
- For this article, create a cluster with **6.0** runtime.
- Make sure you select the **Terminate after __ minutes of inactivity** checkbox. Provide a duration (in minutes) to terminate the cluster, if the cluster is not being used.

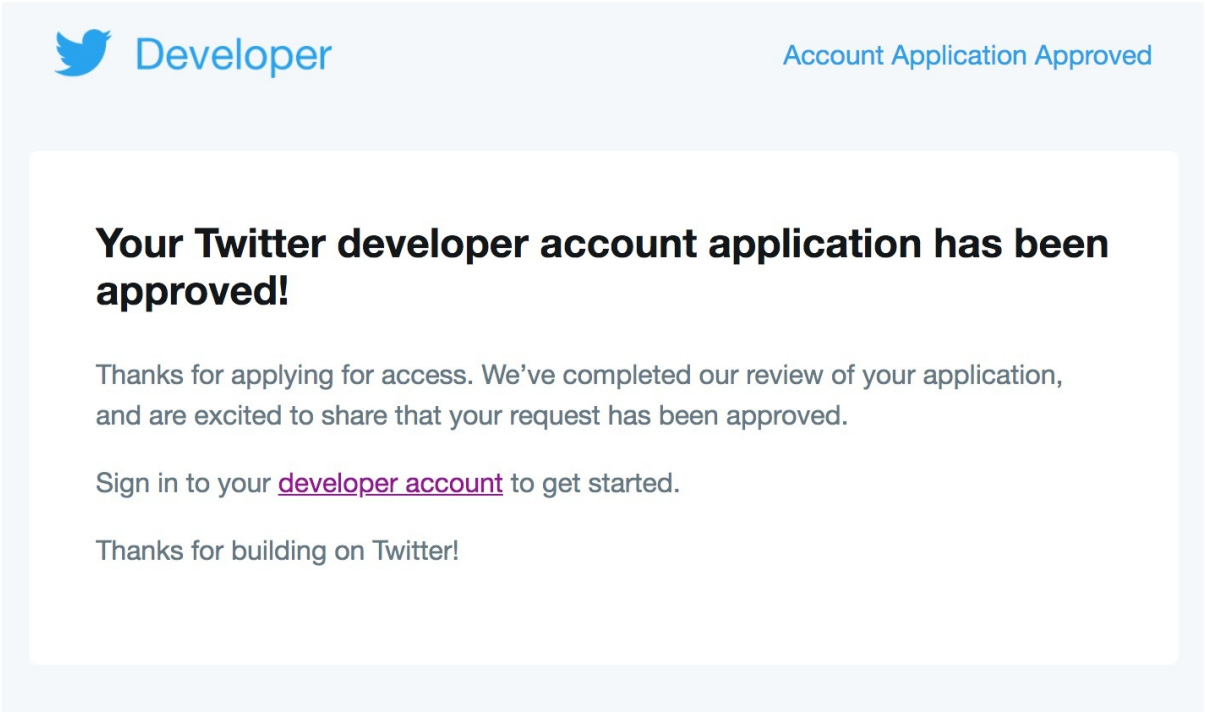
Select cluster worker and driver node size suitable for your technical criteria and budget.

Select **Create cluster**. Once the cluster is running, you can attach notebooks to the cluster and run Spark jobs.

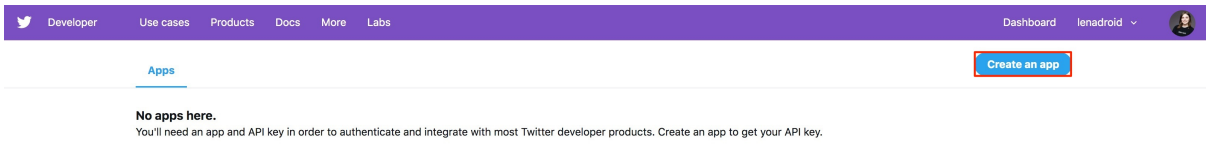
Create a Twitter application

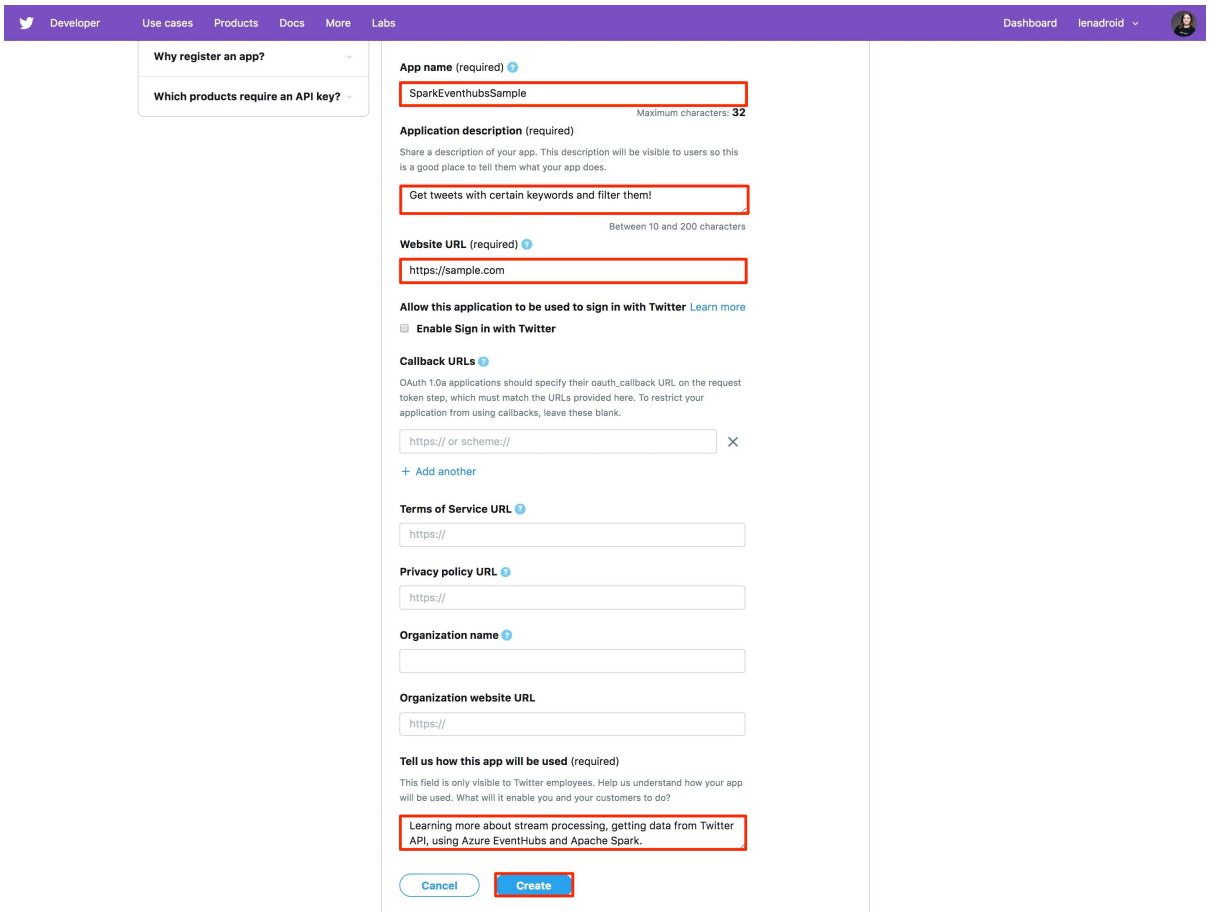
To receive a stream of tweets, you create an application in Twitter. Follow the instructions create a Twitter application and record the values that you need to complete this tutorial.

1. From a web browser, go to Twitter For Developers, and select **Create an app**. You might see a message saying that you need to apply for a Twitter developer account. Feel free to do so, and after your application has been approved you should see a confirmation email. It could take several days to be approved for a developer account.

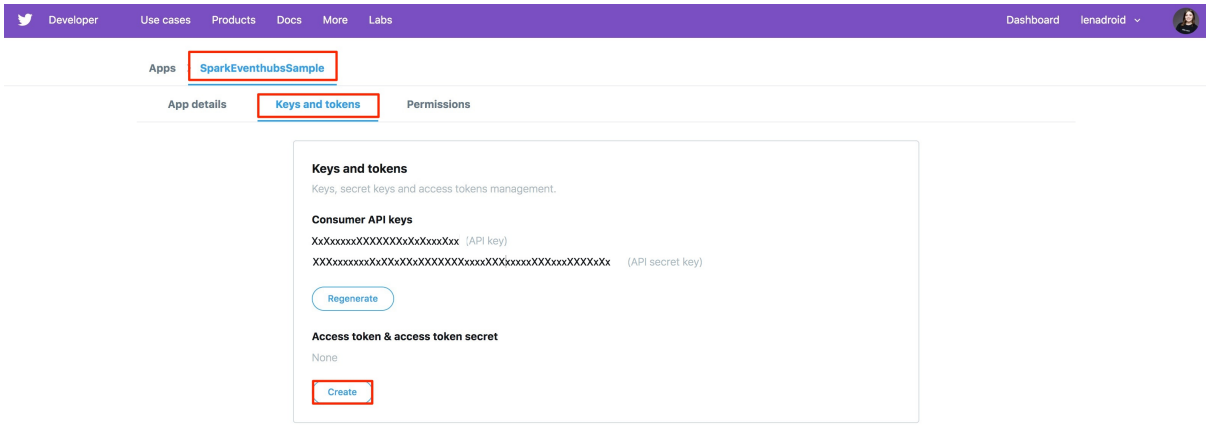


2. In the **Create an application** page, provide the details for the new app, and then select **Create your Twitter application**.





3. In the application page, select the **Keys and Tokens** tab and copy the values for **Consumer API Key** and **Consumer API Secret Key**. Also, select **Create** under **Access Token and Access Token Secret** to generate the access tokens. Copy the values for **Access Token** and **Access Token Secret**.

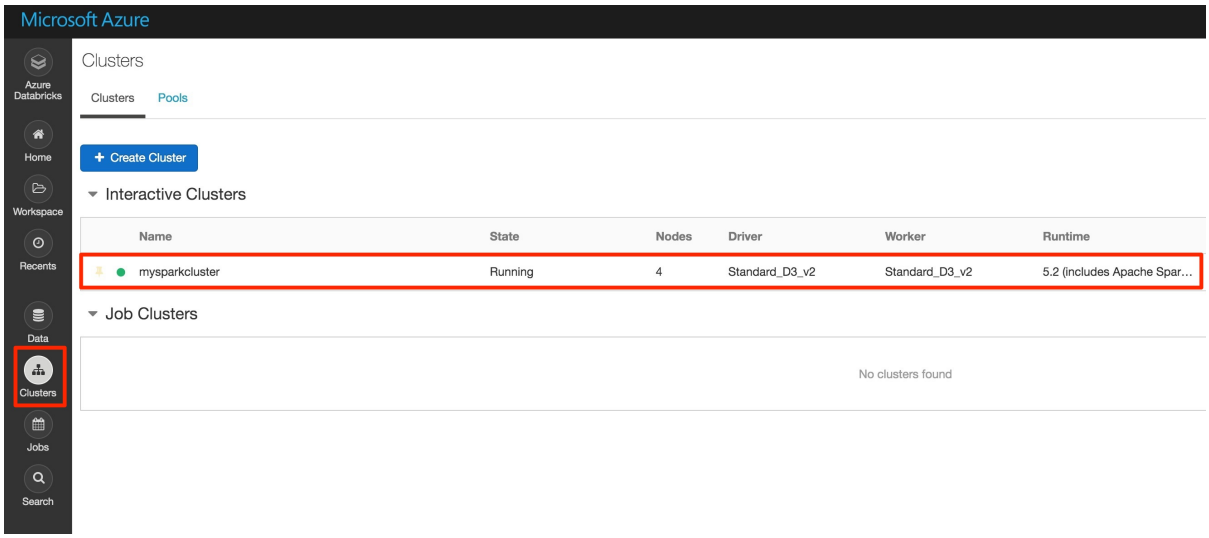


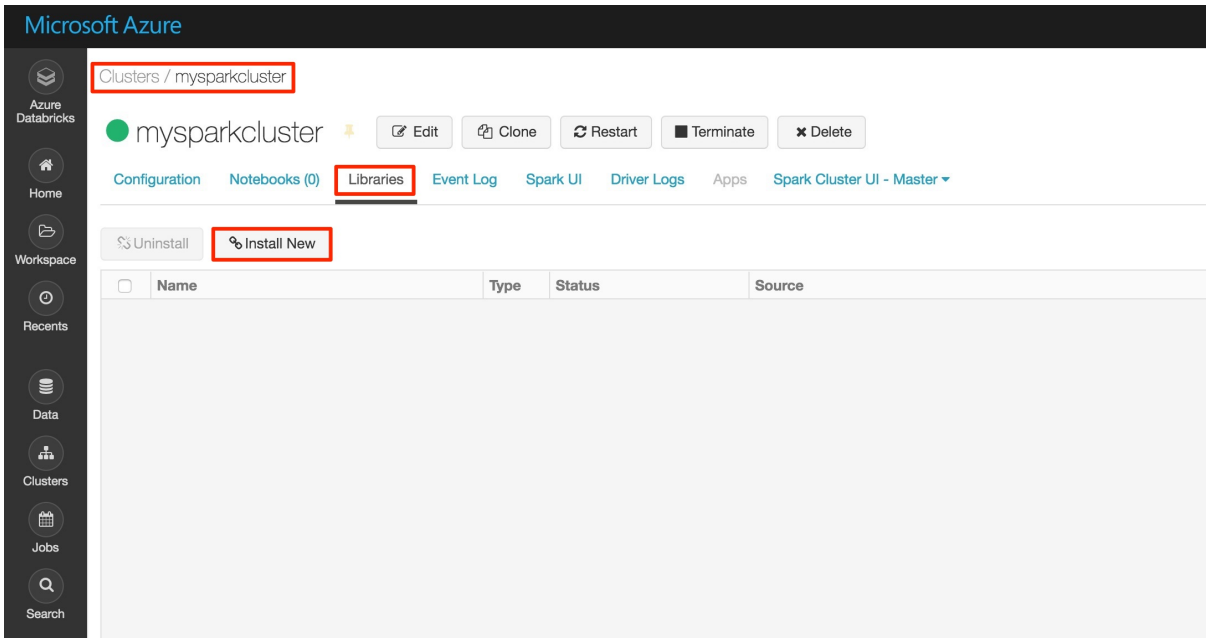
Save the values that you retrieved for the Twitter application. You need the values later in the tutorial.

Attach libraries to Spark cluster

In this tutorial, you use the Twitter APIs to send tweets to Event Hubs. You also use the Apache Spark Event Hubs connector to read and write data into Azure Event Hubs. To use these APIs as part of your cluster, add them as libraries to Azure Databricks and associate them with your Spark cluster. The following instructions show how to add a library.

1. In the Azure Databricks workspace, select **Clusters**, and choose your existing Spark cluster. Within the cluster menu, choose **Libraries** and click **Install New**.



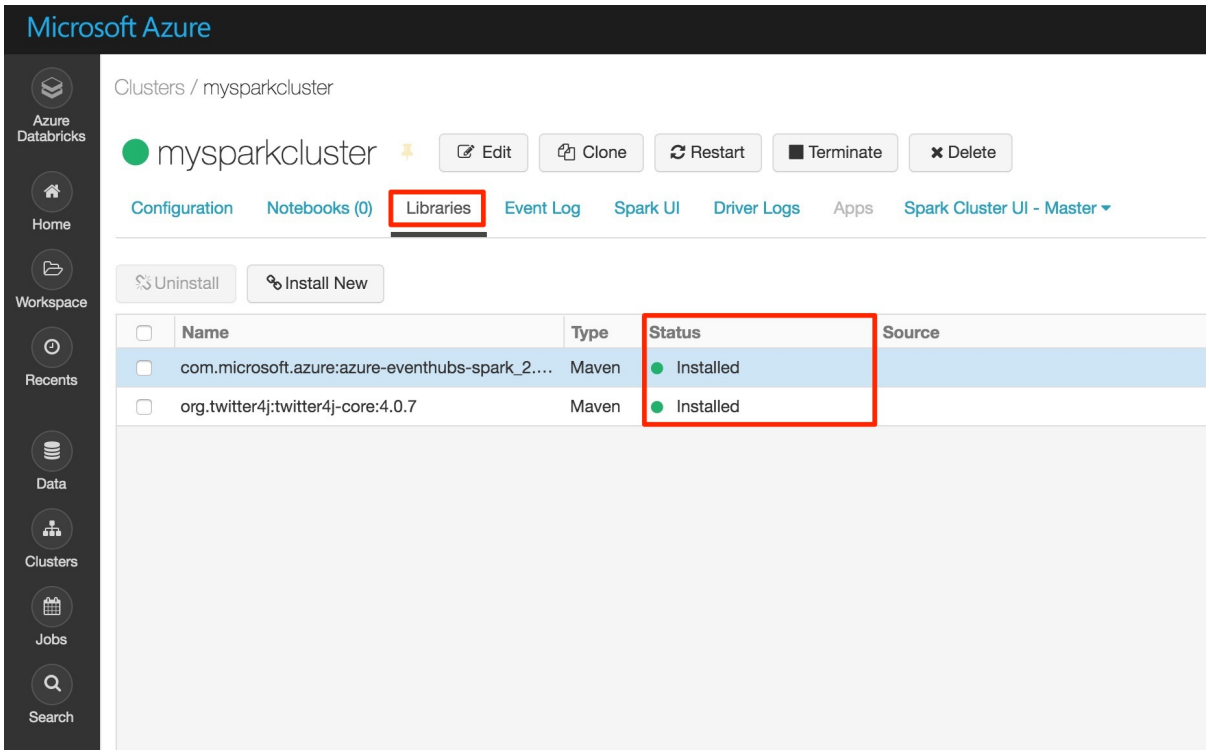


2. In the New Library page, for **Source** select **Maven**. Individually enter the following coordinates for the Spark Event Hubs connector and the Twitter API into **Coordinates**.

   - Spark Event Hubs connector - `com.microsoft.azure:azure-eventhubs-spark_2.11:2.3.12`
   - Twitter API - `org.twitter4j:twitter4j-core:4.0.7`

3. Select **Install**.

4. In the cluster menu, make sure both libraries are installed and attached properly.



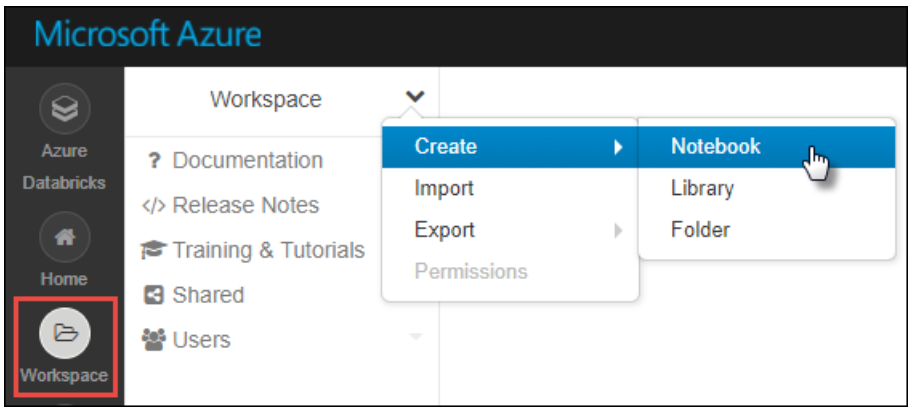
5. Repeat these steps for the Twitter package, `twitter4j-core:4.0.7`.

Create notebooks in Databricks

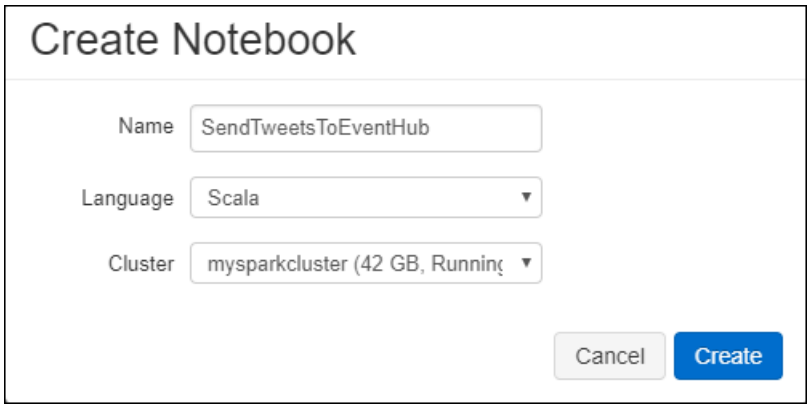
In this section, you create two notebooks in Databricks workspace with the following names:

- **SendTweetsToEventHub** - A producer notebook you use to get tweets from Twitter and stream them to Event Hubs.
- **ReadTweetsFromEventHub** - A consumer notebook you use to read the tweets from Event Hubs.

1. In the left pane, select **Workspace**. From the **Workspace** drop-down, select **Create** > **Notebook**.



2. In the **Create Notebook** dialog box, enter **SendTweetsToEventHub**, select **Scala** as the language, and select the Spark cluster that you created earlier.



Select **Create**.

3. Repeat the steps to create the **ReadTweetsFromEventHub** notebook.

Send tweets to Event Hubs

In the **SendTweetsToEventHub** notebook, paste the following code, and replace the placeholders with values for your Event Hubs namespace and Twitter application that you created earlier. This notebook streams tweets with the keyword "Azure" into Event Hubs in real time.

> **NOTE**
> Twitter API has certain request restrictions and quotas. If you are not satisfied with standard rate limiting in Twitter API, you can generate text content without using Twitter API in this example. To do that, set variable **dataSource** to `test` instead of `twitter` and populate the list **testSource** with preferred test input.

```
import scala.collection.JavaConverters._
import com.microsoft.azure.eventhubs._
import java.util.concurrent._
import scala.collection.immutable._
import scala.concurrent.Future
import scala.concurrent.ExecutionContext.Implicits.global

val namespaceName = "<EVENT HUBS NAMESPACE>"
val eventHubName = "<EVENT HUB NAME>"
val sasKeyName = "<POLICY NAME>"
val sasKey = "<POLICY KEY>"
val connStr = new ConnectionStringBuilder()
            .setNamespaceName(namespaceName)
            .setEventHubName(eventHubName)
            .setSasKeyName(sasKeyName)
            .setSasKey(sasKey)

val pool = Executors.newScheduledThreadPool(1)
val eventHubClient = EventHubClient.create(connStr.toString(), pool)

def sleep(time: Long): Unit = Thread.sleep(time)

def sendEvent(message: String, delay: Long) = {
  sleep(delay)
  val messageData = EventData.create(message.getBytes("UTF-8"))
  eventHubClient.get().send(messageData)
  System.out.println("Sent event: " + message + "\n")
}

// Add your own values to the list
val testSource = List("Azure is the greatest!", "Azure isn't working :(", "Azure is okay.")

// Specify 'test' if you prefer to not use Twitter API and loop through a list of values you define in
```

```
`testSource`
    // Otherwise specify 'twitter'
    val dataSource = "test"

    if (dataSource == "twitter") {

      import twitter4j._
      import twitter4j.TwitterFactory
      import twitter4j.Twitter
      import twitter4j.conf.ConfigurationBuilder

      // Twitter configuration!
      // Replace values below with you

      val twitterConsumerKey = "<CONSUMER API KEY>"
      val twitterConsumerSecret = "<CONSUMER API SECRET>"
      val twitterOauthAccessToken = "<ACCESS TOKEN>"
      val twitterOauthTokenSecret = "<TOKEN SECRET>"

      val cb = new ConfigurationBuilder()
        cb.setDebugEnabled(true)
        .setOAuthConsumerKey(twitterConsumerKey)
        .setOAuthConsumerSecret(twitterConsumerSecret)
        .setOAuthAccessToken(twitterOauthAccessToken)
        .setOAuthAccessTokenSecret(twitterOauthTokenSecret)

      val twitterFactory = new TwitterFactory(cb.build())
      val twitter = twitterFactory.getInstance()

      // Getting tweets with keyword "Azure" and sending them to the Event Hub in realtime!
      val query = new Query(" #Azure ")
      query.setCount(100)
      query.lang("en")
      var finished = false
      while (!finished) {
        val result = twitter.search(query)
        val statuses = result.getTweets()
        var lowestStatusId = Long.MaxValue
        for (status <- statuses.asScala) {
          if(!status.isRetweet()){
            sendEvent(status.getText(), 5000)
          }
          lowestStatusId = Math.min(status.getId(), lowestStatusId)
        }
        query.setMaxId(lowestStatusId - 1)
      }

    } else if (dataSource == "test") {
      // Loop through the list of test input data
      while (true) {
        testSource.foreach {
          sendEvent(_,5000)
        }
      }

    } else {
      System.out.println("Unsupported Data Source. Set 'dataSource' to \"twitter\" or \"test\"")
    }

    // Closing connection to the Event Hub
    eventHubClient.get().close()
```

To run the notebook, press **SHIFT + ENTER**. You see an output like the snippet below. Each event in the output is a tweet that is ingested into the Event Hubs containing the term "Azure".

```
Sent event: @Microsoft and @Esri launch Geospatial AI on Azure https://t.co/VmLUCiPm6q via @geoworldmedia
#geoai #azure #gis #ArtificialIntelligence

Sent event: Public preview of Java on App Service, built-in support for Tomcat and OpenJDK
https://t.co/7vs7cKtvah
#cloudcomputing #Azure

Sent event: 4 Killer #Azure Features for #Data #Performance https://t.co/kpIb7hFO2j by @RedPixie

Sent event: Migrate your databases to a fully managed service with Azure SQL Database Managed Instance | #Azure
| #Cloud https://t.co/sJHXN4trDk

Sent event: Top 10 Tricks to #Save Money with #Azure Virtual Machines https://t.co/F2wshBXdoz #Cloud

...
...
```

Read tweets from Event Hubs

In the **ReadTweetsFromEventHub** notebook, paste the following code, and replace the placeholder with values for your Azure Event Hubs that you created earlier. This notebook reads the tweets that you earlier streamed into Event Hubs using the **SendTweetsToEventHub** notebook.

```
    import org.apache.spark.eventhubs._
    import com.microsoft.azure.eventhubs._

    // Build connection string with the above information
    val namespaceName = "<EVENT HUBS NAMESPACE>"
    val eventHubName = "<EVENT HUB NAME>"
    val sasKeyName = "<POLICY NAME>"
    val sasKey = "<POLICY KEY>"
    val connStr = new com.microsoft.azure.eventhubs.ConnectionStringBuilder()
                .setNamespaceName(namespaceName)
                .setEventHubName(eventHubName)
                .setSasKeyName(sasKeyName)
                .setSasKey(sasKey)

    val customEventhubParameters =
      EventHubsConf(connStr.toString())
      .setMaxEventsPerTrigger(5)

    val incomingStream = spark.readStream.format("eventhubs").options(customEventhubParameters.toMap).load()

    incomingStream.printSchema

    // Sending the incoming stream into the console.
    // Data comes in batches!
    incomingStream.writeStream.outputMode("append").format("console").option("truncate",
false).start().awaitTermination()
```

You get the following output:

```
root
 |-- body: binary (nullable = true)
 |-- offset: long (nullable = true)
 |-- seqNumber: long (nullable = true)
 |-- enqueuedTime: long (nullable = true)
 |-- publisher: string (nullable = true)
 |-- partitionKey: string (nullable = true)

-------------------------------------------
Batch: 0
-------------------------------------------
+------+------+--------------+---------------+---------+------------+
|body  |offset|sequenceNumber|enqueuedTime   |publisher|partitionKey|
+------+------+--------------+---------------+---------+------------+
|[50 75 62 6C 69 63 20 70 72 65 76 69 65 77 20 6F 66 20 4A 61 76 61 20 6F 6E 20 41 70 70 20 53 65 72 76 69 63
65 2C 20 62 75 69 6C 74 2D 69 6E 20 73 75 70 70 6F 72 74 20 66 6F 72 20 54 6F 6D 63 61 74 20 61 6E 64 20 4F 70
65 6E 4A 44 4B 0A 68 74 74 70 73 3A 2F 2F 74 2E 63 6F 2F 37 76 73 37 63 4B 74 76 61 68 20 0A 23 63 6C 6F 75 64
63 6F 6D 70 75 74 69 6E 67 20 23 41 7A 75 72 65]                              |0     |0             |2018-03-09
05:49:08.86 |null     |null        |
|[4D 69 67 72 61 74 65 20 79 6F 75 72 20 64 61 74 61 62 61 73 65 73 20 74 6F 20 61 20 66 75 6C 6C 79 20 6D 61
6E 61 67 65 64 20 73 65 72 76 69 63 65 20 77 69 74 68 20 41 7A 75 72 65 20 53 51 4C 20 44 61 74 61 62 61 73 65
20 4D 61 6E 61 67 65 64 20 49 6E 73 74 61 6E 63 65 20 7C 20 23 41 7A 75 72 65 20 7C 20 23 43 6C 6F 75 64 20 68
74 74 70 73 3A 2F 2F 74 2E 63 6F 2F 73 4A 48 58 4E 34 74 72 44 6B]            |168   |1             |2018-03-09
05:49:24.752|null     |null        |
+------+------+--------------+---------------+---------+------------+

-------------------------------------------
Batch: 1
-------------------------------------------
...
...
```

Because the output is in a binary mode, use the following snippet to convert it into string.

```
    import org.apache.spark.sql.types._
    import org.apache.spark.sql.functions._

    // Event Hub message format is JSON and contains "body" field
    // Body is binary, so we cast it to string to see the actual content of the message
    val messages =
      incomingStream
      .withColumn("Offset", $"offset".cast(LongType))
      .withColumn("Time (readable)", $"enqueuedTime".cast(TimestampType))
      .withColumn("Timestamp", $"enqueuedTime".cast(LongType))
      .withColumn("Body", $"body".cast(StringType))
      .select("Offset", "Time (readable)", "Timestamp", "Body")

    messages.printSchema

    messages.writeStream.outputMode("append").format("console").option("truncate",
false).start().awaitTermination()
```

The output now resembles the following snippet:

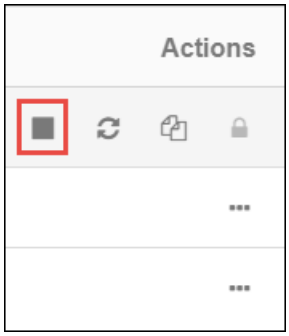
```
root
 |-- Offset: long (nullable = true)
 |-- Time (readable): timestamp (nullable = true)
 |-- Timestamp: long (nullable = true)
 |-- Body: string (nullable = true)

-------------------------------------------
Batch: 0
-------------------------------------------
+------+----------------+----------+-------+
|Offset|Time (readable)  |Timestamp |Body
+------+----------------+----------+-------+
|0     |2018-03-09 05:49:08.86 |1520574548|Public preview of Java on App Service, built-in support for Tomcat
and OpenJDK
https://t.co/7vs7cKtvah
#cloudcomputing #Azure          |
|168   |2018-03-09 05:49:24.752|1520574564|Migrate your databases to a fully managed service with Azure SQL
Database Managed Instance | #Azure | #Cloud https://t.co/sJHXN4trDk    |
|0     |2018-03-09 05:49:02.936|1520574542|@Microsoft and @Esri launch Geospatial AI on Azure
https://t.co/VmLUCiPm6q via @geoworldmedia #geoai #azure #gis #ArtificialIntelligence|
|176   |2018-03-09 05:49:20.801|1520574560|4 Killer #Azure Features for #Data #Performance
https://t.co/kpIb7hFO2j by @RedPixie                                                |
+------+----------------+----------+-------+
-------------------------------------------
Batch: 1
-------------------------------------------
...
...
```

That's it! Using Azure Databricks, you have successfully streamed data into Azure Event Hubs in near real-time. You then consumed the stream data using the Event Hubs connector for Apache Spark. For more information on how to use the Event Hubs connector for Spark, see the connector documentation.

Clean up resources

After you have finished running the tutorial, you can terminate the cluster. To do so, from the Azure Databricks workspace, from the left pane, select **Clusters**. For the cluster you want to terminate, move the cursor over the ellipsis under **Actions** column, and select the **Terminate** icon.



If you do not manually terminate the cluster it will automatically stop, provided you selected the **Terminate after __ minutes of inactivity** checkbox while creating the cluster. In such a case, the cluster will automatically stop if it has been inactive for the specified time.

Next steps

In this tutorial, you learned how to:

- Create an Azure Databricks workspace
- Create a Spark cluster in Azure Databricks
- Create a Twitter app to generate streaming data

- Create notebooks in Azure Databricks
- Add libraries for Event Hubs and Twitter API
- Send tweets to Event Hubs
- Read tweets from Event Hubs

Advance to the next tutorial to learn about performing sentiment analysis on the streamed data using Azure Databricks and Cognitive Services API.

Sentiment analysis on streaming data using Azure Databricks

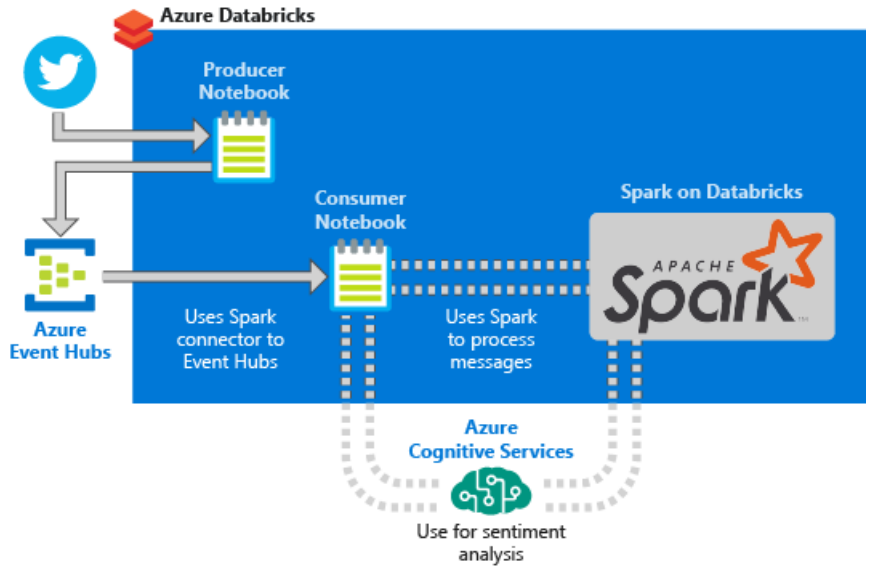
Tutorial: Sentiment analysis on streaming data using Azure Databricks

12/23/2019 • 17 minutes to read • Edit Online

In this tutorial, you learn how to run sentiment analysis on a stream of data using Azure Databricks in near real time. You set up data ingestion system using Azure Event Hubs. You consume the messages from Event Hubs into Azure Databricks using the Spark Event Hubs connector. Finally, you use Cognitive Service APIs to run sentiment analysis on the streamed data.

By the end of this tutorial, you would have streamed tweets from Twitter that have the term "Azure" in them and ran sentiment analysis on the tweets.

The following illustration shows the application flow:



This tutorial covers the following tasks:

- Create an Azure Databricks workspace
- Create a Spark cluster in Azure Databricks
- Create a Twitter app to access streaming data
- Create notebooks in Azure Databricks
- Attach libraries for Event Hubs and Twitter API
- Create a Cognitive Services account and retrieve the access key
- Send tweets to Event Hubs
- Read tweets from Event Hubs
- Run sentiment analysis on tweets

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

> **NOTE**
>
> This tutorial cannot be carried out using **Azure Free Trial Subscription**. If you have a free account, go to your profile and change your subscription to **pay-as-you-go**. For more information, see Azure free account. Then, remove the spending limit, and request a quota increase for vCPUs in your region. When you create your Azure Databricks workspace, you can select the **Trial (Premium - 14-Days Free DBUs)** pricing tier to give the workspace access to free Premium Azure Databricks DBUs for 14 days.

Prerequisites

Before you start with this tutorial, make sure to meet the following requirements:

- An Azure Event Hubs namespace.
- An Event Hub within the namespace.
- Connection string to access the Event Hubs namespace. The connection string should have a format similar to `Endpoint=sb://<namespace>.servicebus.windows.net/;SharedAccessKeyName=<key name>;SharedAccessKey=<key value>`.
- Shared access policy name and policy key for Event Hubs.

You can meet these requirements by completing the steps in the article, Create an Azure Event Hubs namespace and event hub.

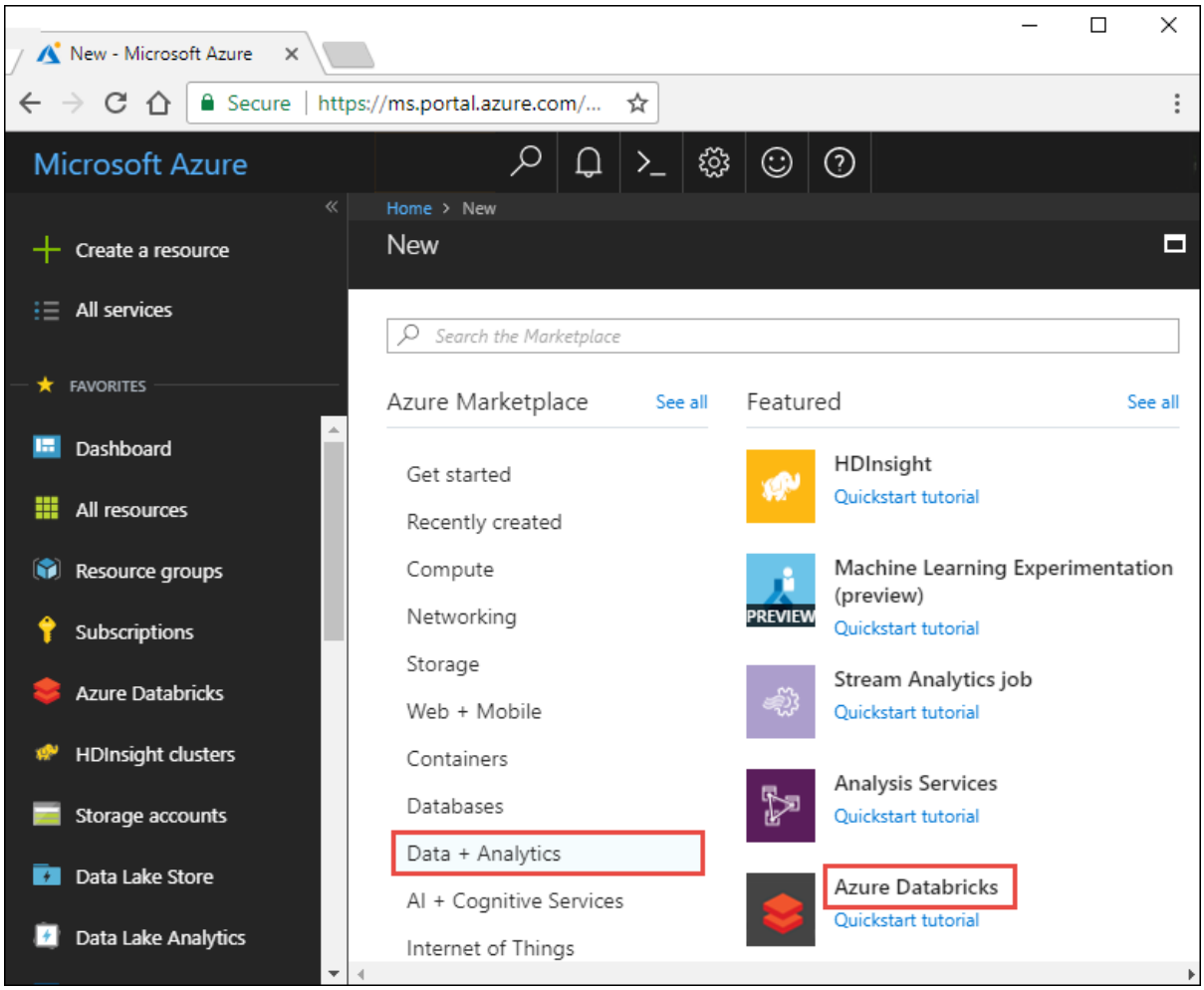
Sign in to the Azure portal

Sign in to the Azure portal.

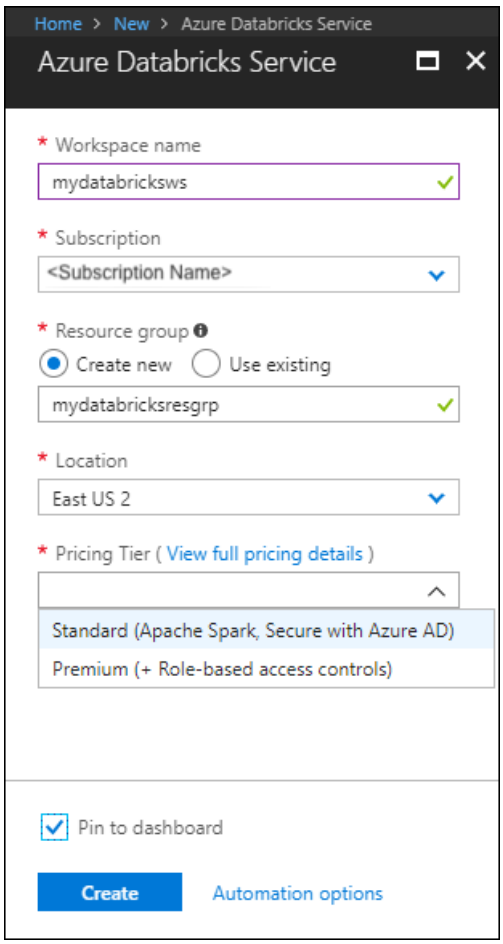
Create an Azure Databricks workspace

In this section, you create an Azure Databricks workspace using the Azure portal.

1. In the Azure portal, select **Create a resource** > **Data + Analytics** > **Azure Databricks**.



2. Under **Azure Databricks Service**, provide the values to create a Databricks workspace.

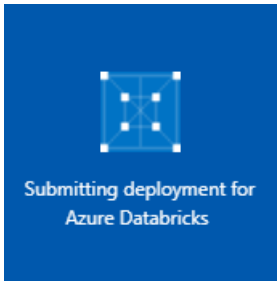


Provide the following values:

PROPERTY	DESCRIPTION
Workspace name	Provide a name for your Databricks workspace
Subscription	From the drop-down, select your Azure subscription.
Resource group	Specify whether you want to create a new resource group or use an existing one. A resource group is a container that holds related resources for an Azure solution. For more information, see Azure Resource Group overview.
Location	Select **East US 2**. For other available regions, see Azure services available by region.
Pricing Tier	Choose between **Standard** or **Premium**. For more information on these tiers, see Databricks pricing page.

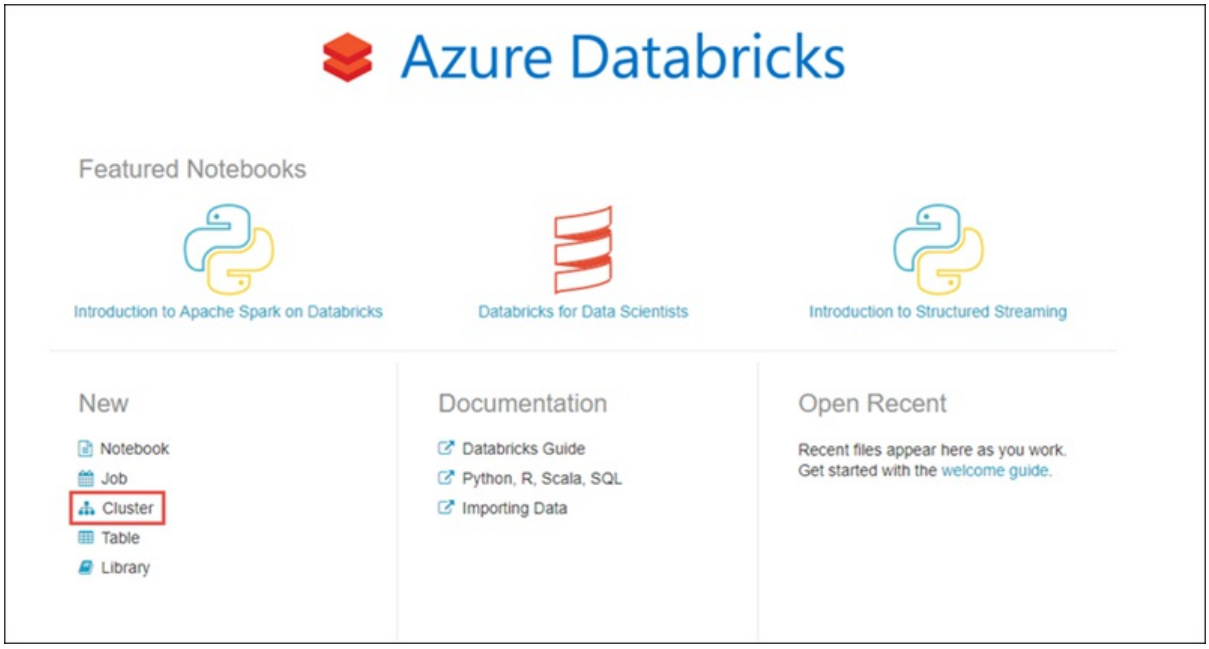
Select **Pin to dashboard** and then select **Create**.

3. The account creation takes a few minutes. During account creation, the portal displays the **Submitting deployment for Azure Databricks** tile on the right side. You may need to scroll right on your dashboard to see the tile. There is also a progress bar displayed near the top of the screen. You can watch either area for progress.

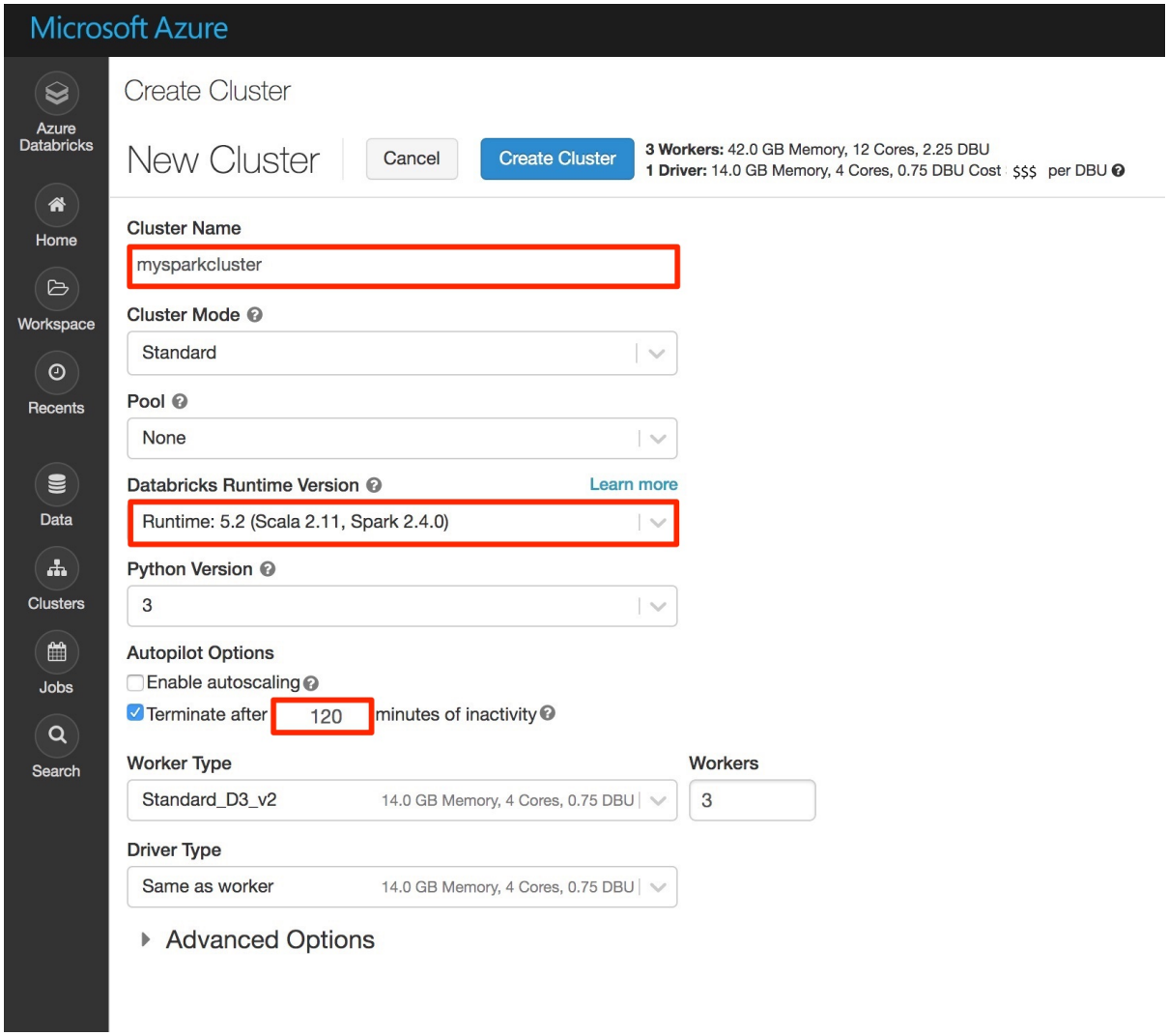


Create a Spark cluster in Databricks

1. In the Azure portal, go to the Databricks workspace that you created, and then select **Launch Workspace**.
2. You are redirected to the Azure Databricks portal. From the portal, select **Cluster**.



3. In the **New cluster** page, provide the values to create a cluster.



Accept all other default values other than the following:

- Enter a name for the cluster.
- For this article, create a cluster with **6.0** runtime.
- Make sure you select the **Terminate after __ minutes of inactivity** checkbox. Provide a duration (in minutes) to terminate the cluster, if the cluster is not being used.

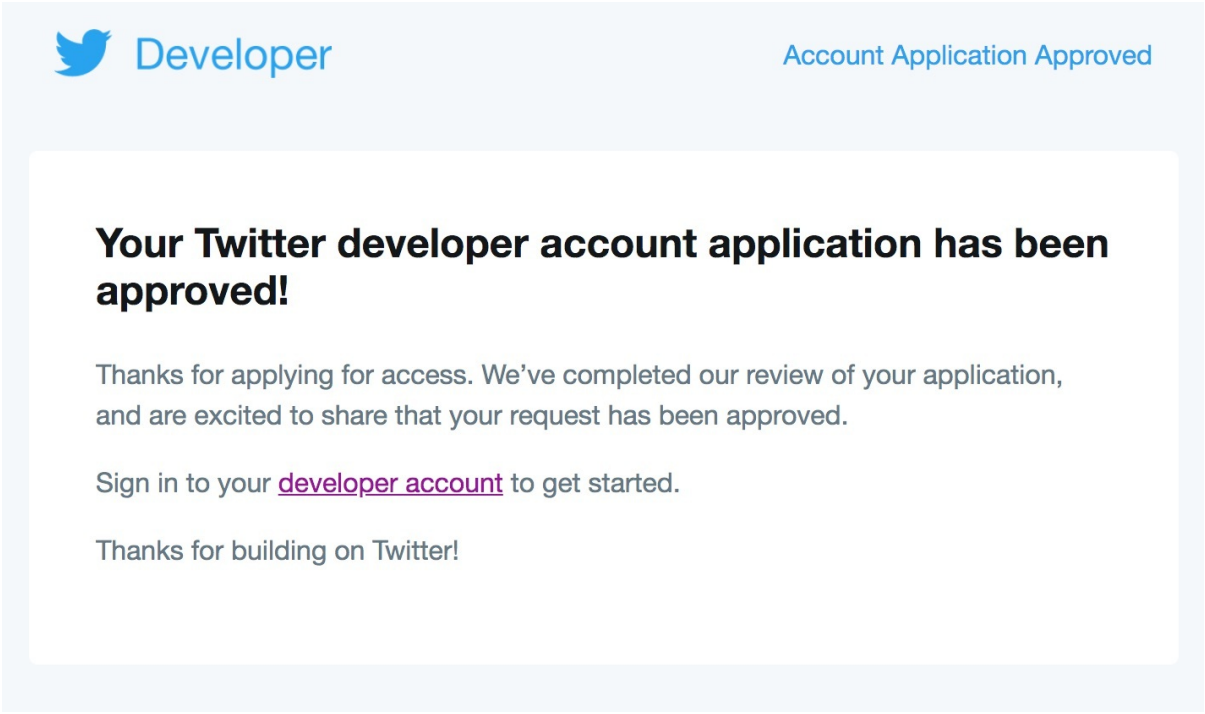
Select cluster worker and driver node size suitable for your technical criteria and budget.

Select **Create cluster**. Once the cluster is running, you can attach notebooks to the cluster and run Spark jobs.

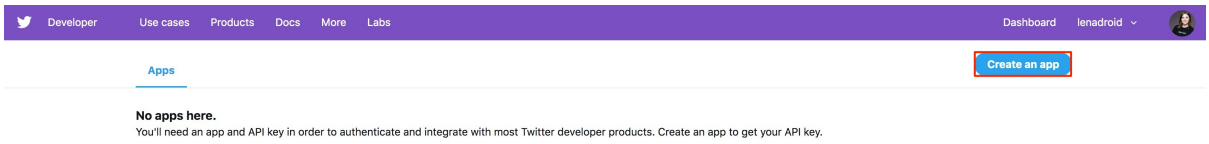
Create a Twitter application

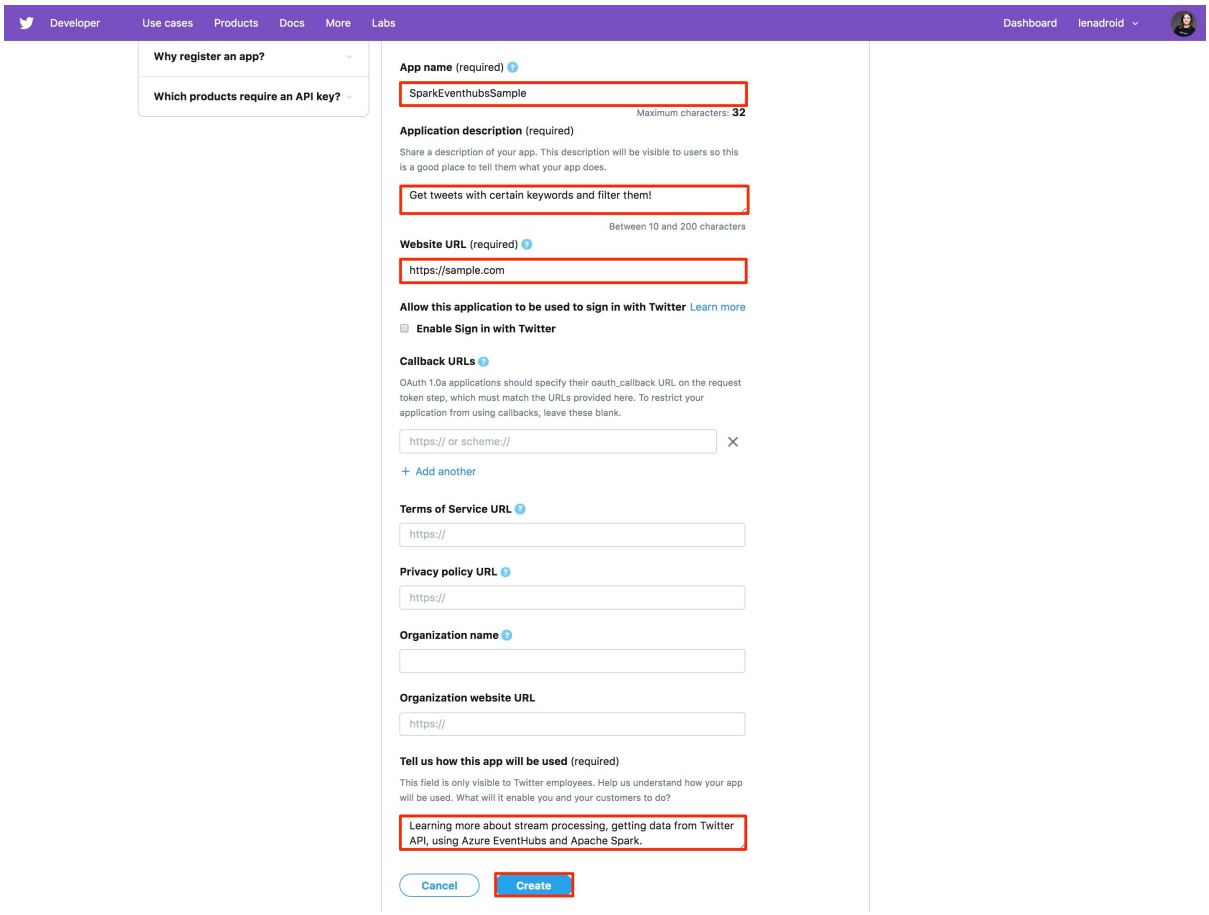
To receive a stream of tweets, you create an application in Twitter. Follow the instructions create a Twitter application and record the values that you need to complete this tutorial.

1. From a web browser, go to Twitter For Developers, and select **Create an app**. You might see a message saying that you need to apply for a Twitter developer account. Feel free to do so, and after your application has been approved you should see a confirmation email. It could take several days to be approved for a developer account.

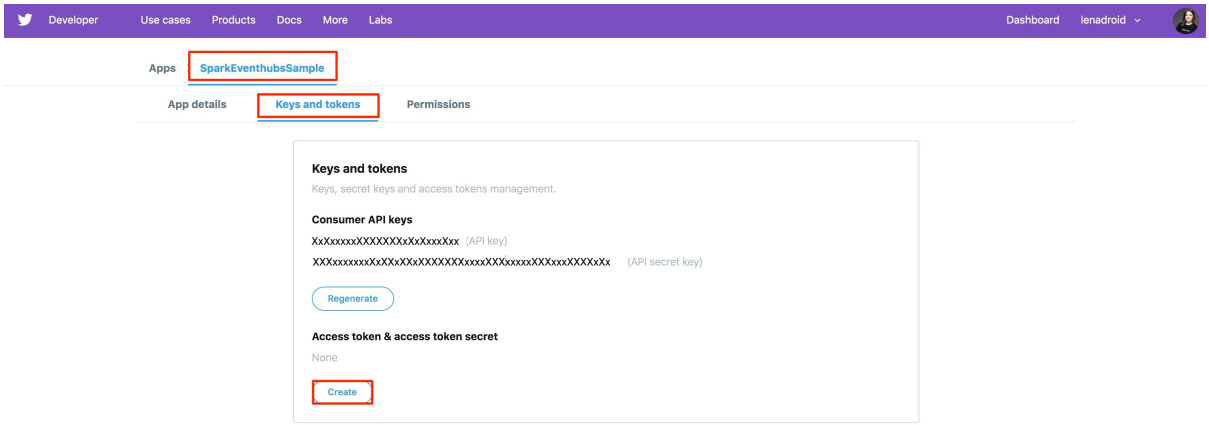


2. In the **Create an application** page, provide the details for the new app, and then select **Create your Twitter application**.





3. In the application page, select the **Keys and Tokens** tab and copy the values for **Consumer API Key** and **Consumer API Secret Key**. Also, select **Create** under **Access Token and Access Token Secret** to generate the access tokens. Copy the values for **Access Token** and **Access Token Secret**.

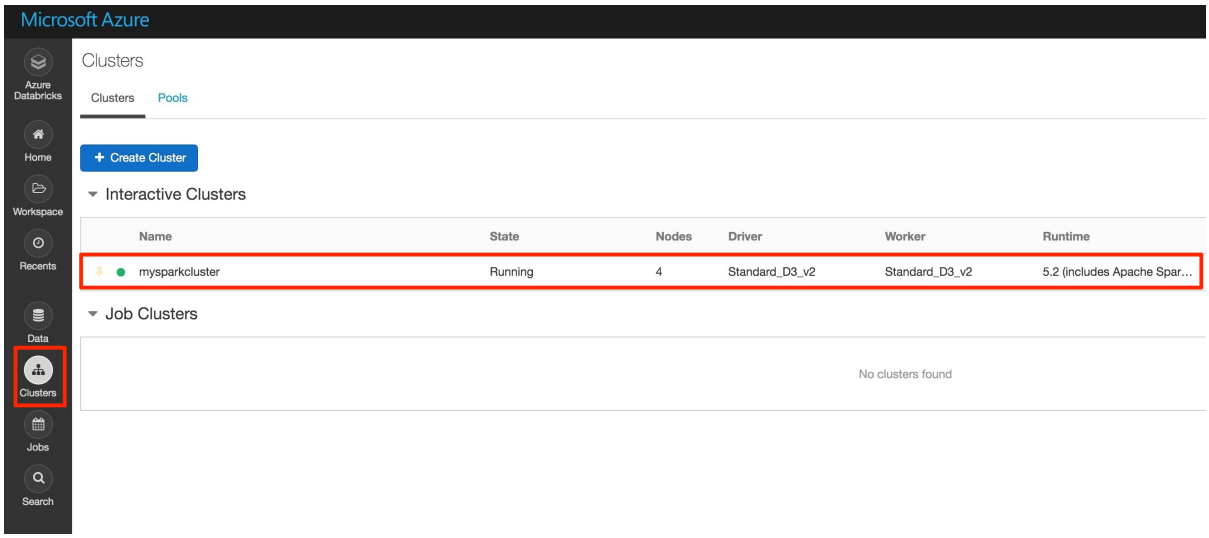


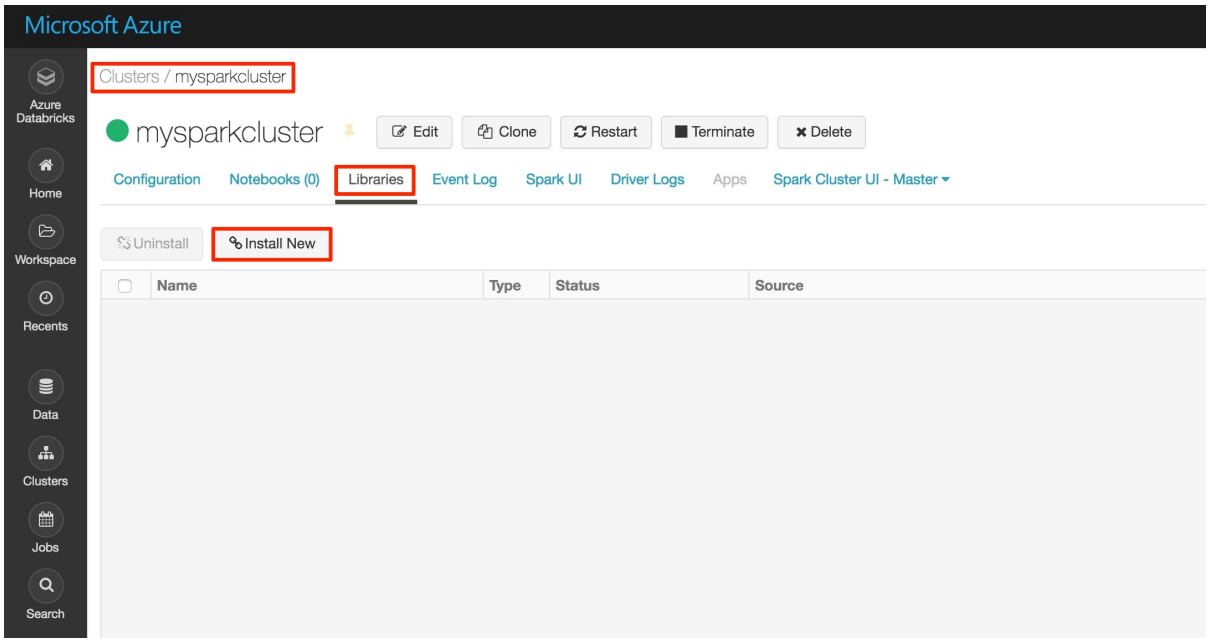
Save the values that you retrieved for the Twitter application. You need the values later in the tutorial.

Attach libraries to Spark cluster

In this tutorial, you use the Twitter APIs to send tweets to Event Hubs. You also use the Apache Spark Event Hubs connector to read and write data into Azure Event Hubs. To use these APIs as part of your cluster, add them as libraries to Azure Databricks and associate them with your Spark cluster. The following instructions show how to add a library.

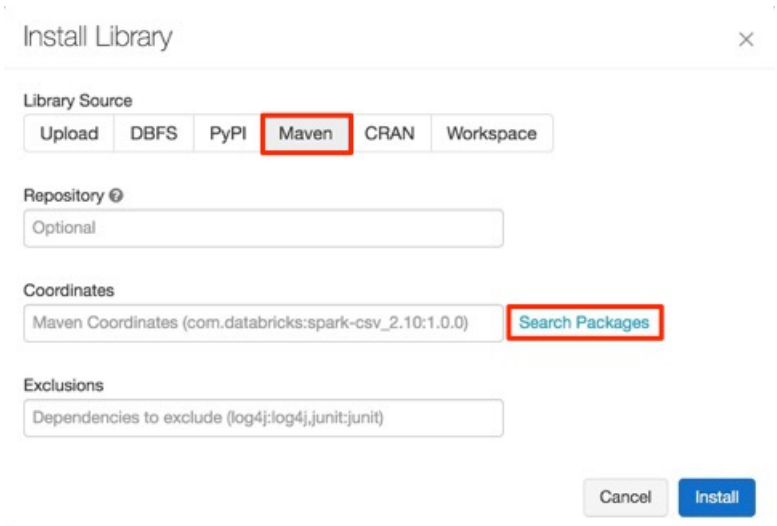
1. In the Azure Databricks workspace, select **Clusters**, and choose your existing Spark cluster. Within the cluster menu, choose **Libraries** and click **Install New**.

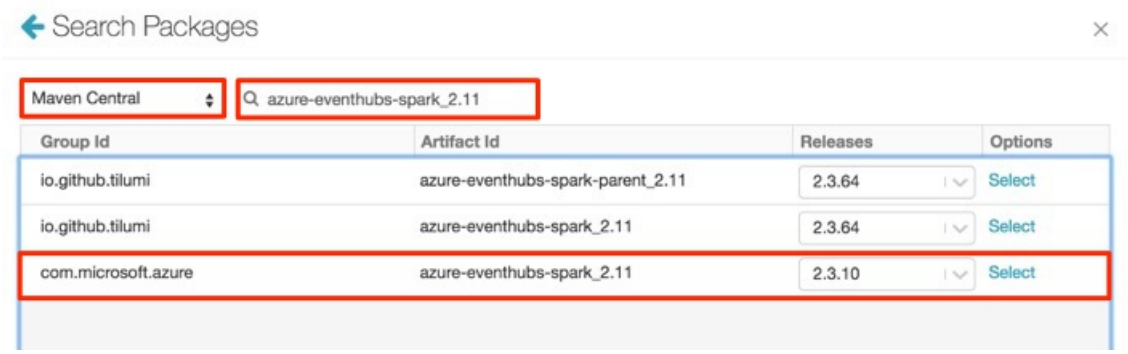




2. In the New Library page, for **Source** select **Maven**. For **Coordinate**, click **Search Packages** for the package you want to add. Here is the Maven coordinates for the libraries used in this tutorial:

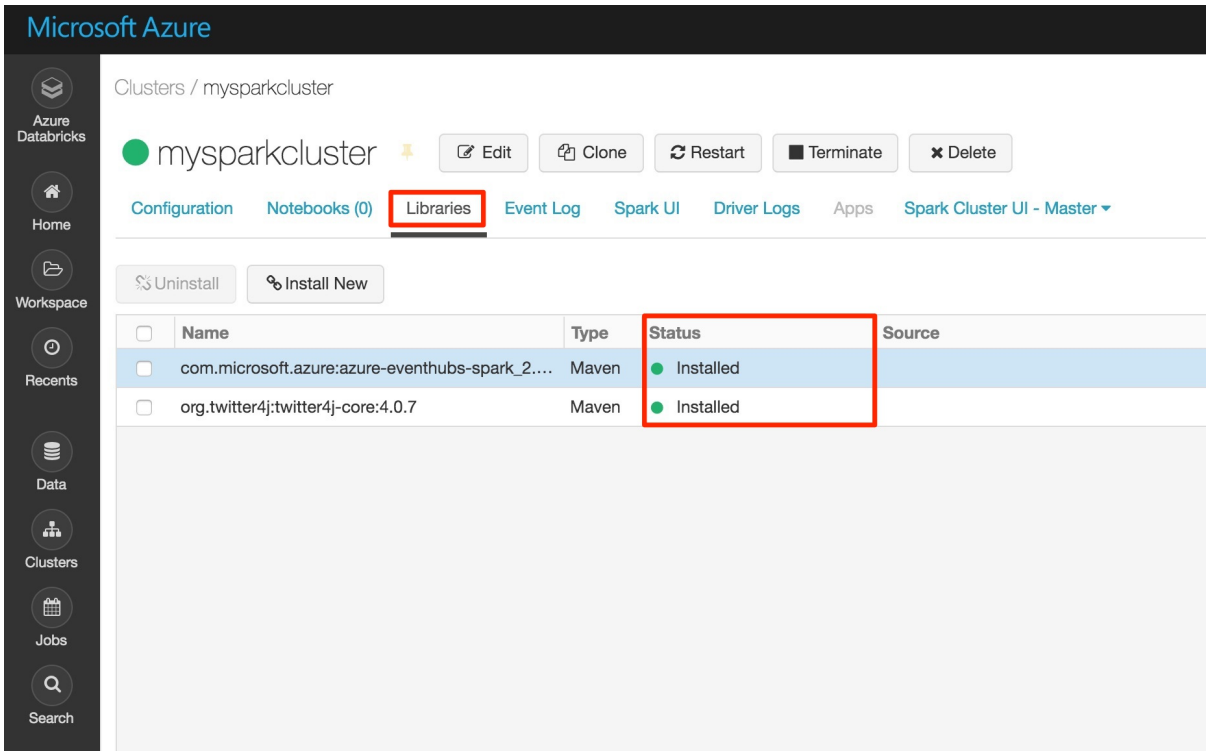
   - Spark Event Hubs connector - `com.microsoft.azure:azure-eventhubs-spark_2.11:2.3.10`
   - Twitter API - `org.twitter4j:twitter4j-core:4.0.7`





3. Select **Install**.

4. In the cluster menu, make sure both libraries are installed and attached properly.



5. Repeat these steps for the Twitter package, `twitter4j-core:4.0.7`.

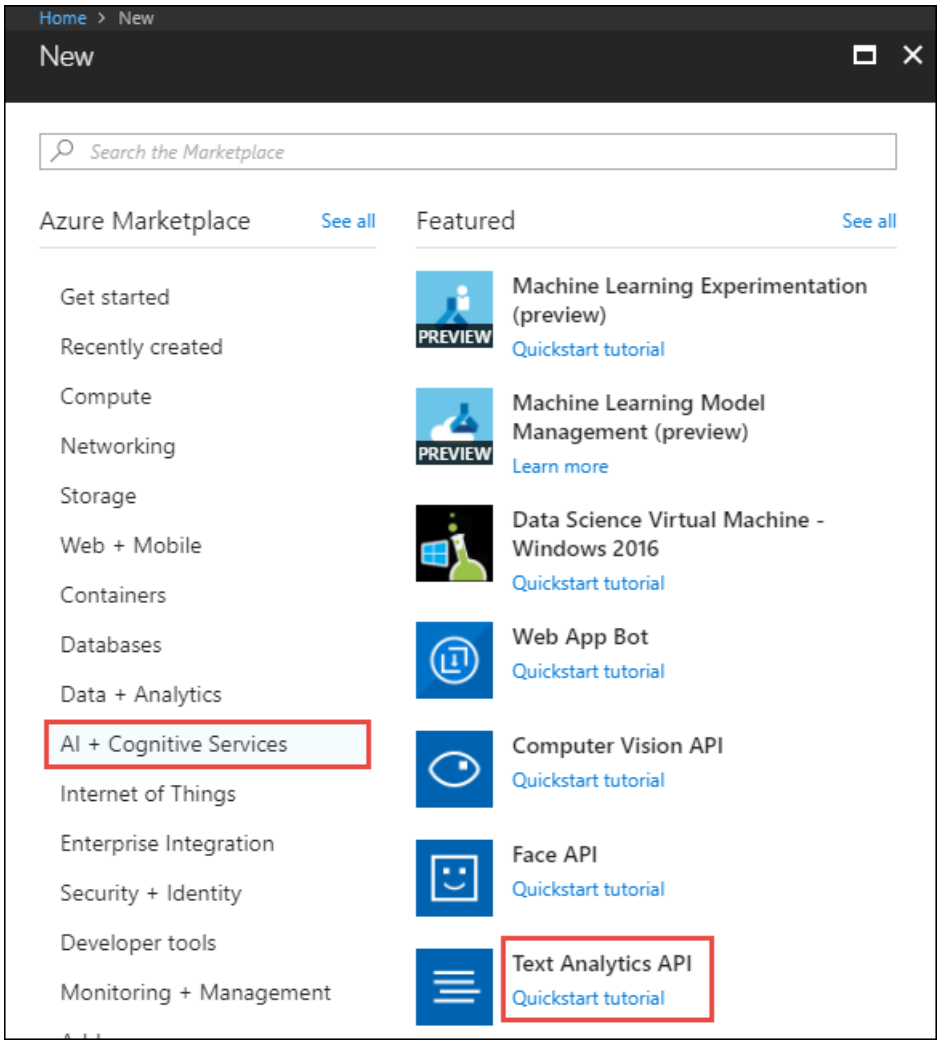
Get a Cognitive Services access key

In this tutorial, you use the Azure Cognitive Services Text Analytics APIs to run sentiment analysis on a stream of tweets in near real time. Before you use the APIs, you must create a Azure Cognitive Services account on Azure and retrieve an access key to use the Text Analytics APIs.

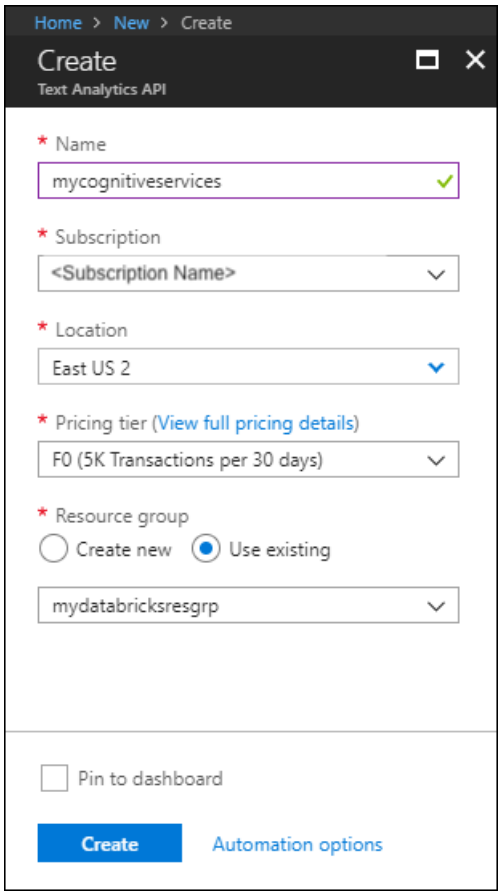
1. Sign in to the Azure portal.

2. Select **+ Create a resource**.

3. Under Azure Marketplace, Select **AI + Cognitive Services** > **Text Analytics API**.



4. In the **Create** dialog box, provide the following values:

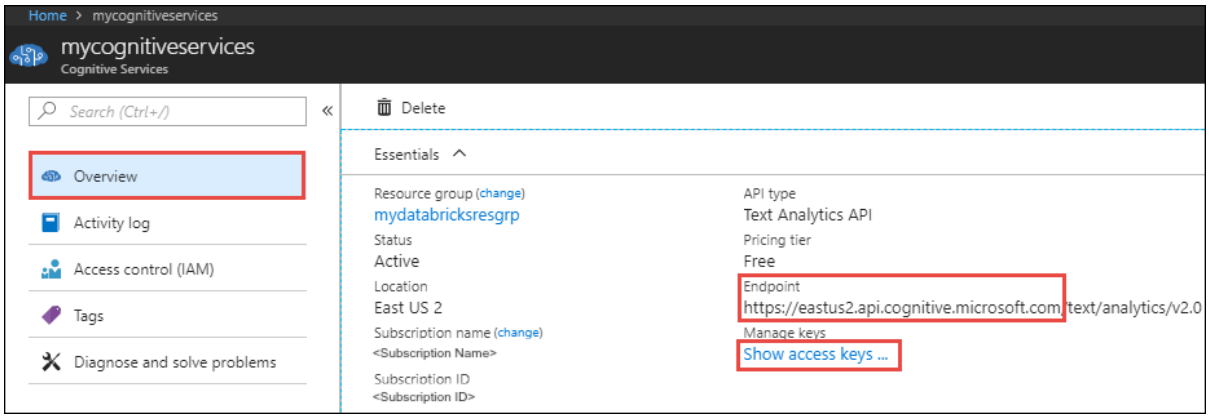


- Enter a name for the Cognitive Services account.

- Select the Azure subscription under which the account is created.
- Select an Azure location.
- Select a pricing tier for the service. For more information about Cognitive Services pricing, see pricing page.
- Specify whether you want to create a new resource group or select an existing one.

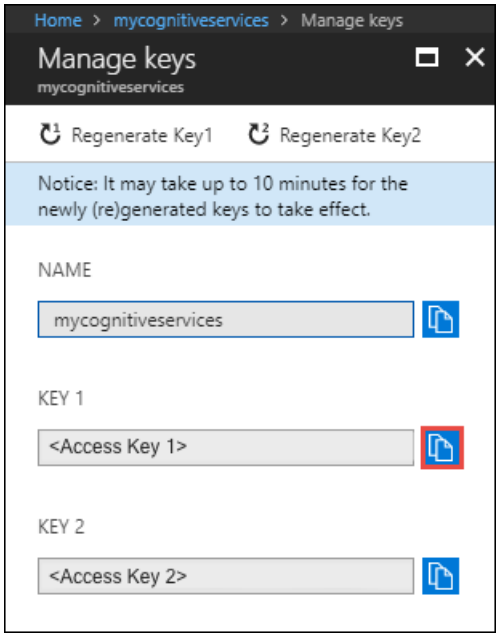
  Select **Create**.

5. After the account is created, from the **Overview** tab, select **Show access keys**.



   Also, copy a part of the endpoint URL, as shown in the screenshot. You need this URL in the tutorial.

6. Under **Manage keys**, select the copy icon against the key you want to use.



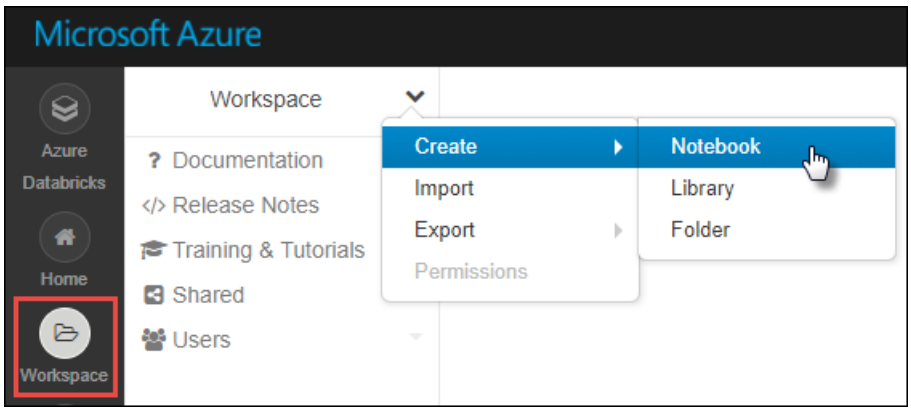
7. Save the values for the endpoint URL and the access key, you retrieved in this step. You need it later in this tutorial.

Create notebooks in Databricks

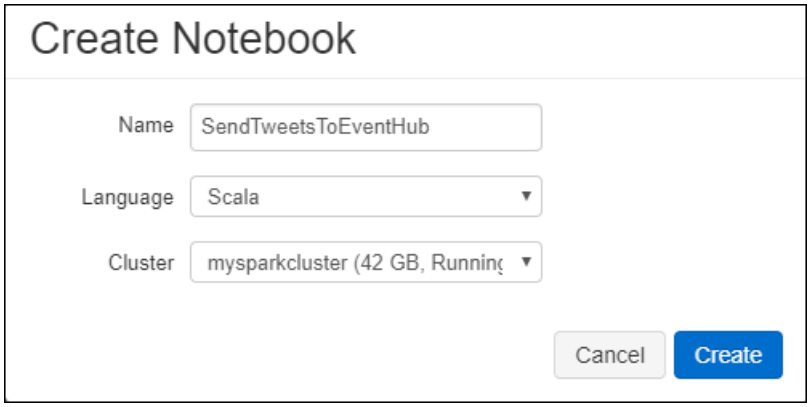
In this section, you create two notebooks in Databricks workspace with the following names

- **SendTweetsToEventHub** - A producer notebook you use to get tweets from Twitter and stream them to Event Hubs.
- **AnalyzeTweetsFromEventHub** - A consumer notebook you use to read the tweets from Event Hubs and run sentiment analysis.

1. In the left pane, select **Workspace**. From the **Workspace** drop-down, select **Create**, and then select **Notebook**.



2. In the **Create Notebook** dialog box, enter **SendTweetsToEventHub**, select **Scala** as the language, and select the Spark cluster that you created earlier.



   Select **Create**.

3. Repeat the steps to create the **AnalyzeTweetsFromEventHub** notebook.

Send tweets to Event Hubs

In the **SendTweetsToEventHub** notebook, paste the following code, and replace the placeholders with values for your Event Hubs namespace and Twitter application that you created earlier. This notebook streams tweets with the keyword "Azure" into Event Hubs in real time.

> **NOTE**
>
> Twitter API has certain request restrictions and quotas. If you are not satisfied with standard rate limiting in Twitter API, you can generate text content without using Twitter API in this example. To do that, set variable **dataSource** to `test` instead of `twitter` and populate the list **testSource** with preferred test input.

```
import scala.collection.JavaConverters._
import com.microsoft.azure.eventhubs._
import java.util.concurrent._
import scala.collection.immutable._
import scala.concurrent.Future
import scala.concurrent.ExecutionContext.Implicits.global

val namespaceName = "<EVENT HUBS NAMESPACE>"
val eventHubName = "<EVENT HUB NAME>"
val sasKeyName = "<POLICY NAME>"
val sasKey = "<POLICY KEY>"
val connStr = new ConnectionStringBuilder()
```

```
                .setNamespaceName(namespaceName)
                .setEventHubName(eventHubName)
                .setSasKeyName(sasKeyName)
                .setSasKey(sasKey)

    val pool = Executors.newScheduledThreadPool(1)
    val eventHubClient = EventHubClient.create(connStr.toString(), pool)

    def sleep(time: Long): Unit = Thread.sleep(time)

    def sendEvent(message: String, delay: Long) = {
      sleep(delay)
      val messageData = EventData.create(message.getBytes("UTF-8"))
      eventHubClient.get().send(messageData)
      System.out.println("Sent event: " + message + "\n")
    }

    // Add your own values to the list
    val testSource = List("Azure is the greatest!", "Azure isn't working :(", "Azure is okay.")

    // Specify 'test' if you prefer to not use Twitter API and loop through a list of values you define in
`testSource`
    // Otherwise specify 'twitter'
    val dataSource = "test"

    if (dataSource == "twitter") {

      import twitter4j._
      import twitter4j.TwitterFactory
      import twitter4j.Twitter
      import twitter4j.conf.ConfigurationBuilder

      // Twitter configuration!
      // Replace values below with you

      val twitterConsumerKey = "<CONSUMER API KEY>"
      val twitterConsumerSecret = "<CONSUMER API SECRET>"
      val twitterOauthAccessToken = "<ACCESS TOKEN>"
      val twitterOauthTokenSecret = "<TOKEN SECRET>"

      val cb = new ConfigurationBuilder()
        cb.setDebugEnabled(true)
        .setOAuthConsumerKey(twitterConsumerKey)
        .setOAuthConsumerSecret(twitterConsumerSecret)
        .setOAuthAccessToken(twitterOauthAccessToken)
        .setOAuthAccessTokenSecret(twitterOauthTokenSecret)

      val twitterFactory = new TwitterFactory(cb.build())
      val twitter = twitterFactory.getInstance()

      // Getting tweets with keyword "Azure" and sending them to the Event Hub in realtime!
      val query = new Query(" #Azure ")
      query.setCount(100)
      query.lang("en")
      var finished = false
      while (!finished) {
        val result = twitter.search(query)
        val statuses = result.getTweets()
        var lowestStatusId = Long.MaxValue
        for (status <- statuses.asScala) {
          if(!status.isRetweet()){
            sendEvent(status.getText(), 5000)
          }
          lowestStatusId = Math.min(status.getId(), lowestStatusId)
        }
        query.setMaxId(lowestStatusId - 1)
      }

    } else if (dataSource == "test") {
```

```
    // Loop through the list of test input data
    while (true) {
      testSource.foreach {
        sendEvent(_,5000)
      }
    }

  } else {
    System.out.println("Unsupported Data Source. Set 'dataSource' to \"twitter\" or \"test\"")
  }

  // Closing connection to the Event Hub
  eventHubClient.get().close()
```

To run the notebook, press **SHIFT + ENTER**. You see an output like the snippet below. Each event in the output is a tweet that is ingested into the Event Hubs containing the term "Azure".

```
Sent event: @Microsoft and @Esri launch Geospatial AI on Azure https://t.co/VmLUCiPm6q via @geoworldmedia
#geoai #azure #gis #ArtificialIntelligence

Sent event: Public preview of Java on App Service, built-in support for Tomcat and OpenJDK
https://t.co/7vs7cKtvah
#cloudcomputing #Azure

Sent event: 4 Killer #Azure Features for #Data #Performance https://t.co/kpIb7hFO2j by @RedPixie

Sent event: Migrate your databases to a fully managed service with Azure SQL Database Managed Instance | #Azure
| #Cloud https://t.co/sJHXN4trDk

Sent event: Top 10 Tricks to #Save Money with #Azure Virtual Machines https://t.co/F2wshBXdoz #Cloud

...
...
```

Read tweets from Event Hubs

In the **AnalyzeTweetsFromEventHub** notebook, paste the following code, and replace the placeholder with values for your Azure Event Hubs that you created earlier. This notebook reads the tweets that you earlier streamed into Event Hubs using the **SendTweetsToEventHub** notebook.

```
    import org.apache.spark.eventhubs._
    import com.microsoft.azure.eventhubs._

    // Build connection string with the above information
    val namespaceName = "<EVENT HUBS NAMESPACE>"
    val eventHubName = "<EVENT HUB NAME>"
    val sasKeyName = "<POLICY NAME>"
    val sasKey = "<POLICY KEY>"
    val connStr = new com.microsoft.azure.eventhubs.ConnectionStringBuilder()
                .setNamespaceName(namespaceName)
                .setEventHubName(eventHubName)
                .setSasKeyName(sasKeyName)
                .setSasKey(sasKey)

    val customEventhubParameters =
      EventHubsConf(connStr.toString())
      .setMaxEventsPerTrigger(5)

    val incomingStream = spark.readStream.format("eventhubs").options(customEventhubParameters.toMap).load()

    incomingStream.printSchema

    // Sending the incoming stream into the console.
    // Data comes in batches!
    incomingStream.writeStream.outputMode("append").format("console").option("truncate",
false).start().awaitTermination()
```

You get the following output:

```
root
 |-- body: binary (nullable = true)
 |-- offset: long (nullable = true)
 |-- seqNumber: long (nullable = true)
 |-- enqueuedTime: long (nullable = true)
 |-- publisher: string (nullable = true)
 |-- partitionKey: string (nullable = true)

-------------------------------------------
Batch: 0
-------------------------------------------
+------+------+--------------+---------------+---------+------------+
|body  |offset|sequenceNumber|enqueuedTime   |publisher|partitionKey|
+------+------+--------------+---------------+---------+------------+
|[50 75 62 6C 69 63 20 70 72 65 76 69 65 77 20 6F 66 20 4A 61 76 61 20 6F 6E 20 41 70 70 20 53 65 72 76 69 63
65 2C 20 62 75 69 6C 74 2D 69 6E 20 73 75 70 70 6F 72 74 20 66 6F 72 20 54 6F 6D 63 61 74 20 61 6E 64 20 4F 70
65 6E 4A 44 4B 0A 68 74 74 70 73 3A 2F 2F 74 2E 63 6F 2F 37 76 73 37 63 4B 74 76 61 68 20 0A 23 63 6C 6F 75 64
63 6F 6D 70 75 74 69 6E 67 20 23 41 7A 75 72 65]                              |0     |0             |2018-03-09
05:49:08.86 |null     |null        |
|[4D 69 67 72 61 74 65 20 79 6F 75 72 20 64 61 74 61 62 61 73 65 73 20 74 6F 20 61 20 66 75 6C 6C 79 20 6D 61
6E 61 67 65 64 20 73 65 72 76 69 63 65 20 77 69 74 68 20 41 7A 75 72 65 20 53 51 4C 20 44 61 74 61 62 61 73 65
20 4D 61 6E 61 67 65 64 20 49 6E 73 74 61 6E 63 65 20 7C 20 23 41 7A 75 72 65 20 7C 20 23 43 6C 6F 75 64 20 68
74 74 70 73 3A 2F 2F 74 2E 63 6F 2F 73 4A 48 58 4E 34 74 72 44 6B]            |168   |1             |2018-03-09
05:49:24.752|null     |null        |
+------+------+--------------+---------------+---------+------------+

-------------------------------------------
Batch: 1
-------------------------------------------
...
...
```

Because the output is in a binary mode, use the following snippet to convert it into string.

```
    import org.apache.spark.sql.types._
    import org.apache.spark.sql.functions._

    // Event Hub message format is JSON and contains "body" field
    // Body is binary, so we cast it to string to see the actual content of the message
    val messages =
      incomingStream
      .withColumn("Offset", $"offset".cast(LongType))
      .withColumn("Time (readable)", $"enqueuedTime".cast(TimestampType))
      .withColumn("Timestamp", $"enqueuedTime".cast(LongType))
      .withColumn("Body", $"body".cast(StringType))
      .select("Offset", "Time (readable)", "Timestamp", "Body")

    messages.printSchema

    messages.writeStream.outputMode("append").format("console").option("truncate",
false).start().awaitTermination()
```

The output now resembles the following snippet:

```
root
 |-- Offset: long (nullable = true)
 |-- Time (readable): timestamp (nullable = true)
 |-- Timestamp: long (nullable = true)
 |-- Body: string (nullable = true)

-------------------------------------------
Batch: 0
-------------------------------------------
+------+----------------+----------+-------+
|Offset|Time (readable)  |Timestamp |Body
+------+----------------+----------+-------+
|0     |2018-03-09 05:49:08.86 |1520574548|Public preview of Java on App Service, built-in support for Tomcat
and OpenJDK
https://t.co/7vs7cKtvah
#cloudcomputing #Azure          |
|168   |2018-03-09 05:49:24.752|1520574564|Migrate your databases to a fully managed service with Azure SQL
Database Managed Instance | #Azure | #Cloud https://t.co/sJHXN4trDk    |
|0     |2018-03-09 05:49:02.936|1520574542|@Microsoft and @Esri launch Geospatial AI on Azure
https://t.co/VmLUCiPm6q via @geoworldmedia #geoai #azure #gis #ArtificialIntelligence|
|176   |2018-03-09 05:49:20.801|1520574560|4 Killer #Azure Features for #Data #Performance
https://t.co/kpIb7hFO2j by @RedPixie                                                |
+------+----------------+----------+-------+
-------------------------------------------
Batch: 1
-------------------------------------------
...
...
```

You have now streamed data from Azure Event Hubs into Azure Databricks at near real time using the Event Hubs connector for Apache Spark. For more information on how to use the Event Hubs connector for Spark, see the connector documentation.

Run sentiment analysis on tweets

In this section, you run sentiment analysis on the tweets received using the Twitter API. For this section, you add the code snippets to the same **AnalyzeTweetsFromEventHub** notebook.

Start by adding a new code cell in the notebook and paste the code snippet provided below. This code snippet defines data types for working with the Language and Sentiment API.

```
import java.io._
import java.net._
import java.util._

case class Language(documents: Array[LanguageDocuments], errors: Array[Any]) extends Serializable
case class LanguageDocuments(id: String, detectedLanguages: Array[DetectedLanguages]) extends Serializable
case class DetectedLanguages(name: String, iso6391Name: String, score: Double) extends Serializable

case class Sentiment(documents: Array[SentimentDocuments], errors: Array[Any]) extends Serializable
case class SentimentDocuments(id: String, score: Double) extends Serializable

case class RequestToTextApi(documents: Array[RequestToTextApiDocument]) extends Serializable
case class RequestToTextApiDocument(id: String, text: String, var language: String = "") extends Serializable
```

Add a new code cell and paste the snippet provided below. This snippet defines an object that contains functions to call the Text Analysis API to run language detection and sentiment analysis. Make sure you replace the placeholder `<PROVIDE ACCESS KEY HERE>` with the value you retrieved for your Cognitive Services account.

```
import javax.net.ssl.HttpsURLConnection
import com.google.gson.Gson
import com.google.gson.GsonBuilder
import com.google.gson.JsonObject
import com.google.gson.JsonParser
import scala.util.parsing.json._

object SentimentDetector extends Serializable {

    // Cognitive Services API connection settings
    val accessKey = "<PROVIDE ACCESS KEY HERE>"
    val host = "https://cognitive-docs.cognitiveservices.azure.com/"
    val languagesPath = "/text/analytics/v2.1/languages"
    val sentimentPath = "/text/analytics/v2.1/sentiment"
    val languagesUrl = new URL(host+languagesPath)
    val sentimenUrl = new URL(host+sentimentPath)
    val g = new Gson

    def getConnection(path: URL): HttpsURLConnection = {
        val connection = path.openConnection().asInstanceOf[HttpsURLConnection]
        connection.setRequestMethod("POST")
        connection.setRequestProperty("Content-Type", "text/json")
        connection.setRequestProperty("Ocp-Apim-Subscription-Key", accessKey)
        connection.setDoOutput(true)
        return connection
    }

    def prettify (json_text: String): String = {
        val parser = new JsonParser()
        val json = parser.parse(json_text).getAsJsonObject()
        val gson = new GsonBuilder().setPrettyPrinting().create()
        return gson.toJson(json)
    }

    // Handles the call to Cognitive Services API.
    def processUsingApi(request: RequestToTextApi, path: URL): String = {
        val requestToJson = g.toJson(request)
        val encoded_text = requestToJson.getBytes("UTF-8")
        val connection = getConnection(path)
        val wr = new DataOutputStream(connection.getOutputStream())
        wr.write(encoded_text, 0, encoded_text.length)
        wr.flush()
        wr.close()

        val response = new StringBuilder()
        val in = new BufferedReader(new InputStreamReader(connection.getInputStream()))
        var line = in.readLine()
```

```
        while (line != null) {
            response.append(line)
            line = in.readLine()
        }
        in.close()
        return response.toString()
    }

    // Calls the language API for specified documents.
    def getLanguage (inputDocs: RequestToTextApi): Option[Language] = {
        try {
            val response = processUsingApi(inputDocs, languagesUrl)
            // In case we need to log the json response somewhere
            val niceResponse = prettify(response)
            // Deserializing the JSON response from the API into Scala types
            val language = g.fromJson(niceResponse, classOf[Language])
            if (language.documents(0).detectedLanguages(0).iso6391Name == "(Unknown)")
                return None
            return Some(language)
        } catch {
            case e: Exception => return None
        }
    }

    // Calls the sentiment API for specified documents. Needs a language field to be set for each of them.
    def getSentiment (inputDocs: RequestToTextApi): Option[Sentiment] = {
        try {
            val response = processUsingApi(inputDocs, sentimenUrl)
            val niceResponse = prettify(response)
            // Deserializing the JSON response from the API into Scala types
            val sentiment = g.fromJson(niceResponse, classOf[Sentiment])
            return Some(sentiment)
        } catch {
            case e: Exception => return None
        }
    }
}
```

Add another cell to define a Spark UDF (User-defined function) that determines sentiment.

```
// User Defined Function for processing content of messages to return their sentiment.
val toSentiment =
    udf((textContent: String) =>
        {
            val inputObject = new RequestToTextApi(Array(new RequestToTextApiDocument(textContent,
textContent)))
            val detectedLanguage = SentimentDetector.getLanguage(inputObject)
            detectedLanguage match {
                case Some(language) =>
                    if(language.documents.size > 0) {
                        inputObject.documents(0).language =
language.documents(0).detectedLanguages(0).iso6391Name
                        val sentimentDetected = SentimentDetector.getSentiment(inputObject)
                        sentimentDetected match {
                            case Some(sentiment) => {
                                if(sentiment.documents.size > 0) {
                                    sentiment.documents(0).score.toString()
                                }
                                else {
                                    "Error happened when getting sentiment: " + sentiment.errors(0).toString
                                }
                            }
                            case None => "Couldn't detect sentiment"
                        }
                    }
                    else {
                        "Error happened when getting language" + language.errors(0).toString
                    }
                case None => "Couldn't detect language"
            }
        }
    )
```

Add a final code cell to prepare a dataframe with the content of the tweet and the sentiment associated with the tweet.

```
// Prepare a dataframe with Content and Sentiment columns
val streamingDataFrame = incomingStream.selectExpr("cast (body as string) AS Content").withColumn("Sentiment",
toSentiment($"Content"))

// Display the streaming data with the sentiment
streamingDataFrame.writeStream.outputMode("append").format("console").option("truncate",
false).start().awaitTermination()
```

You should see an output like the following snippet:

```
-------------------------------------------
Batch: 0
-------------------------------------------
+--------------------------------+------------------+
|Content                         |Sentiment         |
+--------------------------------+------------------+
|Public preview of Java on App Service, built-in support for Tomcat and OpenJDK
https://t.co/7vs7cKtvah   #cloudcomputing #Azure          |0.7761918306350708|
|Migrate your databases to a fully managed service with Azure SQL Database Managed Instance | #Azure | #Cloud
https://t.co/sJHXN4trDk    |0.8558163642883301|
|@Microsoft and @Esri launch Geospatial AI on Azure https://t.co/VmLUCiPm6q via @geoworldmedia #geoai #azure
#gis #ArtificialIntelligence|0.5               |
|4 Killer #Azure Features for #Data #Performance https://t.co/kpIb7hFO2j by @RedPixie
|0.5               |
+--------------------------------+------------------+
```

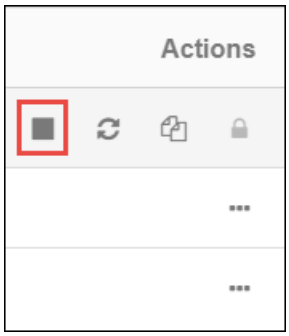
A value closer to **1** in the **Sentiment** column suggests a great experience with Azure. A value closer to **0** suggests

issues that users faced while working with Microsoft Azure.

That's it! Using Azure Databricks, you have successfully streamed data into Azure Event Hubs, consumed the stream data using the Event Hubs connector, and then ran sentiment analysis on streaming data in near real time.

Clean up resources

After you have finished running the tutorial, you can terminate the cluster. To do so, from the Azure Databricks workspace, from the left pane, select **Clusters**. For the cluster you want to terminate, move the cursor over the ellipsis under **Actions** column, and select the **Terminate** icon.



If you do not manually terminate the cluster it will automatically stop, provided you selected the **Terminate after __ minutes of inactivity** checkbox while creating the cluster. In such a case, the cluster will automatically stop if it has been inactive for the specified time.

Next steps

In this tutorial, you learned how to use Azure Databricks to stream data into Azure Event Hubs and then read the streaming data from Event Hubs in real time. You learned how to:

- Create an Azure Databricks workspace
- Create a Spark cluster in Azure Databricks
- Create a Twitter app to access streaming data
- Create notebooks in Azure Databricks
- Add and attach libraries for Event Hubs and Twitter API
- Create a Microsoft Cognitive Services account and retrieve the access key
- Send tweets to Event Hubs
- Read tweets from Event Hubs
- Run sentiment analysis on tweets

Advance to the next tutorial to learn about performing machine learning tasks using Azure Databricks.

Machine Learning using Azure Databricks