SQL Server vNext CTP1

Contents

[Overview 2](#_Toc500853799)

[Setup 3](#_Toc500853800)

[Setup Overview 3](#_Toc500853801)

[Kubernetes Cluster Setup 3](#_Toc500853802)

[Deploy on Minikube 3](#_Toc500853803)

[Step 1: Prerequisites 3](#_Toc500853804)

[Step 3: Install Minikube 4](#_Toc500853805)

[Step 4: Create a Minikube cluster 4](#_Toc500853806)

[Step 5: Disable automatic checkpoint If Using Windows 10/Hyper-V 4](#_Toc500853807)

[Deploy on Azure Container Services (ACS) 4](#_Toc500853808)

[Step 1: PreRequesites 4](#_Toc500853809)

[Step 2: Create SSH Keys 5](#_Toc500853810)

[Step 3: Login to Azure 5](#_Toc500853811)

[Step 4: Create a resource group 5](#_Toc500853812)

[Step 5: Create a SERVICE PRINCIPAL 5](#_Toc500853813)

[Step 6: Create a Kubernetes cluster 6](#_Toc500853814)

[Step 7: Connect to the cluster 6](#_Toc500853815)

[Deploy SQL Server vNext CTP1 in Kubernetes cluster 7](#_Toc500853816)

[Step 1: Verify K8S cluster Config 7](#_Toc500853817)

[Step 2: Clone the Deployment Project from GitHub 7](#_Toc500853818)

[Step 3: Configure persistent storage claim for ACS 7](#_Toc500853819)

[Step 4: Configure Number of Replicas 9](#_Toc500853820)

[Step 5: Deployment 9](#_Toc500853821)

[Step 6: Getting the IP Address and Port to Connect To 11](#_Toc500853822)

[CREATE Compute Pool 11](#_Toc500853823)

[Head Node Setup 12](#_Toc500853824)

[Spark Streaming to Compute Pool 12](#_Toc500853825)

[Query from Head Node 13](#_Toc500853826)

[Join High Volume Data and High Value Data 13](#_Toc500853827)

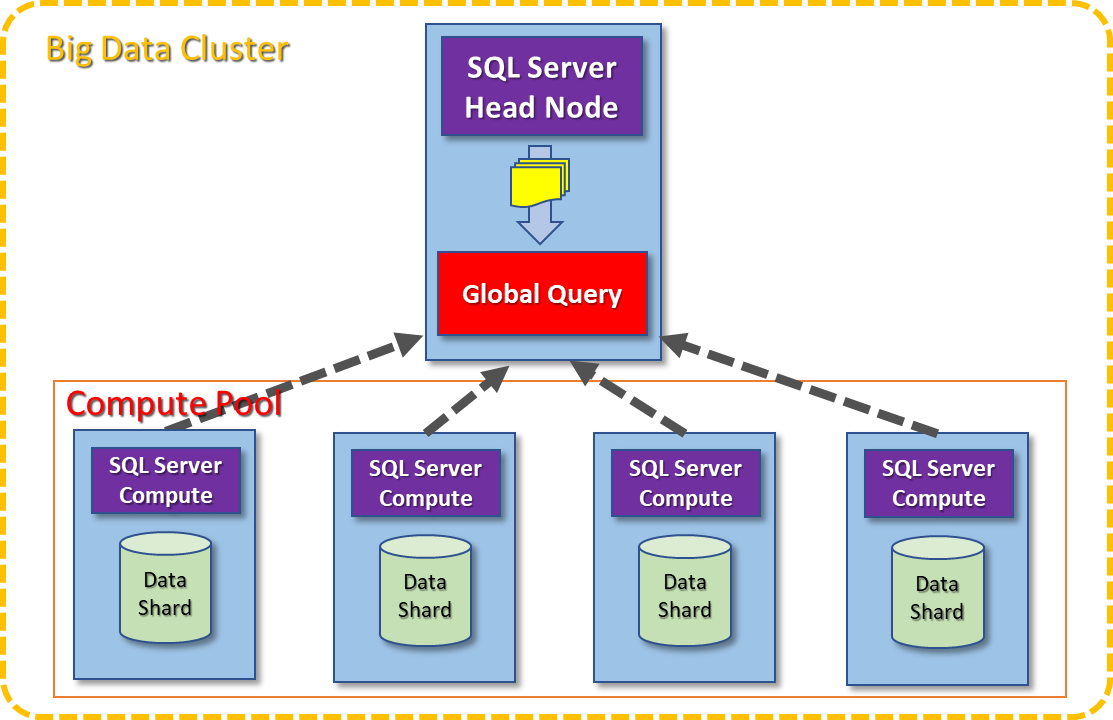
# Overview

SQL Server vNext CTP1 enables you to integrate your “**high-value**” relational data in SQL Server with your “**high-volume**” data in big data environments such as Hadoop.

CTP1 allows you to create and deploy a “**compute pool**” that consists of many “**SQL Server compute instances**” in your Hadoop cluster. You can then ingest your high-volume data from Spark streaming jobs into the SQL Server compute instances by partitioning the data and spreading the partitions across the SQL Server compute instances in the compute pool.

Once the high-volume data is stored in partitions in the SQL Server compute instances on the Hadoop cluster, you can create an “**external table**” in a “**SQL Server head node**” that represents the high-volume data that resides in the partitions stored in the SQL Server compute instances in your Hadoop cluster. This external table can be queried in the head node just like any other table, but in this case a “**fan-out query**” will be *simultaneously* executed against *each* of the SQL Server compute instances to query the partitioned data. The results of these queries will be brought back to the head node and you can optionally choose to join the results of the high-volume data fan-out query with the results of a high-value data query in the SQL Server head node.

The diagram below shows a 5-node configuration (1 head node + 4 compute instances) in a Hadoop cluster with SQL Server and Spark running on each of the instances.



This document will walk you through:

* Setup – deploy and configure SQL Server head node & compute pool on Kubernetes
* Stream – initiate Spark streaming jobs to write high-volume data into SQL Server compute pool
* Query – query high-volume data in SQL Server compute pool and combine with high-value data

# Setup

## Setup Overview

SQL Server vNext CTP1 can be deployed as docker containers on a Kubernetes cluster. This is an overview of the setup and configuration steps:

* Setup Kubernetes cluster on a single VM, cluster of VMs or in Azure Container Services
* Deploy SQL Server vNext CTP1 in Kubernetes cluster
* Configure SQL Server Head Node

## Kubernetes Cluster Setup

If you already have a Kubernetes cluster, then you can skip directly to the [deployment step](#_Deploy_SQL_Server). This section assumes a basic understanding of Kubernetes concepts. For detailed information on Kubernetes, see the [Kubernetes documentation](https://kubernetes.io/docs/home/).

You can choose to deploy Kubernetes in any of three ways:

* [Minikube – a single-node Kubernetes cluster in a VM](#_Deploy_on_Minikube)
* [Azure Container Services](#_Deploy_on_Azure) – a managed Kubernetes cluster in Azure
* Multiple VMs – Please refer to “K8S Deployment on Multiple VMs” document in GitHub for instructions

## Deploy on Minikube

**(Skip this section if you plan on using Azure Container Service or your own Kubernetes cluster)**

Minikube is a tool that makes it easy to run Kubernetes on a single machine like a laptop or a desktop. Minikube runs a single-node Kubernetes cluster inside a VM on your laptop for users looking to try out Kubernetes or develop with it day-to-day. **To run a Minikube cluster for CTP1, it is recommended that your machine have at least 32 GB of RAM**. If the machine has insufficient memory, then modify the cluster configuration such that only 3 instances are created – 1 head node & 2 compute instances.

### Step 1: Prerequisites

* VT-x or AMD-v virtualization must be enabled in your computer’s BIOS.
* If you are on Windows, install [Git for Windows](https://git-for-windows.github.io/). If you are on Linux or Mac, install [git](https://git-scm.com/book/en/v2/Getting-Started-Installing-Git).
* Install [kubectl](https://kubernetes.io/docs/tasks/tools/install-kubectl/)
* If you do not already have a hypervisor installed, install one now.
* For OS X, install [xhyve driver](https://git.k8s.io/minikube/docs/drivers.md), [VirtualBox](https://www.virtualbox.org/wiki/Downloads), or [VMware Fusion](https://www.vmware.com/products/fusion).
* For Linux, install [VirtualBox](https://www.virtualbox.org/wiki/Downloads) or [KVM](http://www.linux-kvm.org/).
* For Windows, install [VirtualBox](https://www.virtualbox.org/wiki/Downloads) or [Hyper-V](https://msdn.microsoft.com/en-us/virtualization/hyperv_on_windows/quick_start/walkthrough_install).
  + If you do not have an external switch configured in hyper-v, then create one that has external network access. See how to [create external switch in hyper-v for minikube](https://blogs.msdn.microsoft.com/wasimbloch/2017/01/23/setting-up-kubernetes-on-windows10-laptop-with-minikube/).

### Step 3: Install Minikube

Install Minikube according to the instructions for the [latest release](https://github.com/kubernetes/minikube/releases).

### Step 4: Create a Minikube cluster

The command below creates a minikube cluster in a Hyper-V VM with 8 CPUs, 28 GB memory, and disk size of 100GB. The disk size is not reserved space. It will grow to that size on disk as needed. We recommend not changing the disk space to something less than 100GB as we ran into problems with this in testing. This also specifies the hyper-v switch with external access explicitly.

**Change the parameters such as --memory as needed depending on your available hardware and which hypervisor you are using. Make sure the --hyper-v virtual-switch parameter value matches the name you used when creating your virtual switch.**

minikube start --vm-driver="hyperv" --cpus 8 --memory 28672 --disk-size 100g --hyperv-virtual-switch "External"

If you are using Minikube with VirtualBox the command would look like this:

minikube start --cpus 8 --memory 28672 --disk-size 100g

### Step 5: Disable automatic checkpoint If Using Windows 10/Hyper-V

On Windows 10, automatic checkpoint is enabled on a VM. Execute the command below in PowerShell to disable automatic checkpoint on the VM.

Set-VM -Name minikube -CheckpointType Disabled

## Deploy on Azure Container Services (ACS)

**(Skip this section if you plan on using Minikube or your own Kubernetes cluster)**

Azure Container Service makes it simple to create, configure, and manage a cluster of virtual machines that are preconfigured with a Kubernetes cluster to run containerized applications. This enables you to use your existing skills or draw upon a large and growing body of community expertise, to deploy and manage container-based applications on Microsoft Azure.

This section describes the steps to deploy Kubernetes on Azure Container Services using Azure CLI. If you don't have an Azure subscription, create a [free account](https://azure.microsoft.com/free/?WT.mc_id=A261C142F) before you begin.

### Step 1: PreRequesites

* If you are on Windows, install [Git for Windows](https://git-for-windows.github.io/). If you are on Linux or Mac, install [git](https://git-scm.com/book/en/v2/Getting-Started-Installing-Git).
* Install [kubectl](https://kubernetes.io/docs/tasks/tools/install-kubectl/)
* This section requires that you be running the Azure CLI version 2.0.4 or later. If you need to install or upgrade, see [Install Azure CLI 2.0](https://docs.microsoft.com/en-us/cli/azure/install-azure-cli). Run az --version to find the version if needed.

### Step 2: Create SSH Keys

#### Windows

If you are on Windows, follow these steps to create SSH keys:

Open the **Git Bash shell** from the Start menu.

Run the following commands

mkdir sqlvnextkeys

cd sqlvnextkeys

ssh-keygen -t rsa -b 2048 -f ./id\_rsa -N youpassword

#### Linux

If you are on Linux or MacOS, follow these steps to create SSH keys:

Run the following commands at the terminal:

mkdir ~/sqlvnextkeys

cd ~/sqlvnextkeys

ssh-keygen -t rsa -b 2048 -f ./id\_rsa -N yourpassword

### Step 3: Login to Azure

**For the remainder of these steps if you are on Windows please use PowerShell.**

At the command prompt run the following command and follow the prompts to login to your Azure subscription:

az login

### Step 4: Create a resource group

Create a resource group with the [az group create](https://docs.microsoft.com/en-us/cli/azure/group) command. An Azure resource group is a logical group in which Azure resources are deployed and managed. The following example creates a resource group named *sqlvnextgroup* in the *westus* location.

az group create --name **sqlvnextgroup** --location westus2

Note: We recommend using only the following locations: ukwest, uksouth, westcentralus, westus2, canadaeast, canadacentral, westindia, southindia, centralindia

### Step 5: Create a SERVICE PRINCIPAL

Create a service principal that Kubernetes will be used to interact with Azure APIs to manage resources within the service.

You will first need to determine your subscription ID:

az account show

Your subscription ID is the ID property value in the JSON that is returned by the above command. Copy that and replace the <subscriptionID> placeholder in the next command that is used to create the service principal. Make sure you create the service principal in the same resource group that you created above.

az ad sp create-for-rbac --role="Contributor" --scopes="/subscriptions**/<subscriptionID>**/resourceGroups/**sqlvnextgroup**"

**Save the JSON that is returned for future reference later.**

### Step 6: Create a Kubernetes cluster

**Wait for 1 minute for the newly created service principal to be available for use in your resource group.**

Create a Kubernetes cluster in Azure Container Service with the [az acs create](https://docs.microsoft.com/en-us/cli/azure/acs) command. The following example creates a cluster named *sqlvnextcluster* with one Linux master node and three Linux agent nodes. Make sure you create the ACS cluster in the same resource group that you created above. You can increase or decrease the default agent count by adding --agent-count <n> to the [az acs create](https://docs.microsoft.com/en-us/cli/azure/acs) command where <n> is the number of agent nodes you want to have.

az acs create --orchestrator-type kubernetes --resource-group **sqlvnextgroup** --name **sqlvnextcluster** --ssh-key-value=**~/sqlvnextkeys/id\_rsa.pub** --agent-vm-size=Standard\_DS2\_v2--master-vm-size=Standard\_DS2\_v2--master-storage-profile=ManagedDisks--agent-storage-profile**=**ManagedDisks --agent-count=2

After several minutes, the command completes and returns JSON-formatted information about the cluster. **Save the JSON that is returned for future reference later.**

### Step 7: Connect to the cluster

To manage a Kubernetes cluster, use [kubectl](https://kubernetes.io/docs/user-guide/kubectl/), the Kubernetes command-line client. To install it locally, you can use the [az acs kubernetes install-cli](https://docs.microsoft.com/en-us/cli/azure/acs/kubernetes) command.

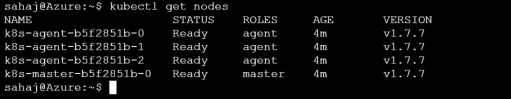
To configure kubectl to connect to your Kubernetes cluster, run the [az acs kubernetes get-credentials](https://docs.microsoft.com/en-us/cli/azure/acs/kubernetes) command. This step downloads credentials and configures the Kubernetes CLI to use them.

az acs kubernetes get-credentials --resource-group=**sqlvnextgroup** --name **sqlvnextcluster** --ssh-key-file=**~/sqlvnextkeys/id\_rsa**

You will be asked to enter the password that you used when you created the SSH keys above.

To verify the connection to your cluster, use the [kubectl get](https://kubernetes.io/docs/user-guide/kubectl/v1.6/) command to return a list of the cluster nodes.

kubectl get nodes



# Deploy SQL Server vNext CTP1 in Kubernetes cluster

Now that there is a Kubernetes cluster setup using Minikube, ACS or VMs, you can proceed with the deployment for SQL Server vNext CTP1. This step can be repeated to recreate the environment if needed.

**Before starting the deployment, you need the credentials to the private docker registry containing SQL Server vNext CTP1 images. This should be provided to you by your PM buddy. If you have not already received them, please contact your PM buddy.**

### Step 1: Verify K8S cluster Config

Execute the below kubectl command to view the cluster configuration. Ensure that kubectl is pointed to the correct cluster context.

kubectl config view

### Step 2: Clone the Deployment Project from GitHub

First, in PowerShell or a cmd window, navigate to the directory that you want to contain the deployment project files.

Then, run the following command to clone the files from GitHub to your machine:

git clone https://github.com/annashres/sqleapvnext.git

### Step 3: Configure persistent storage claim for ACS

**(Skip this step if you are using Minikube)**

To deploy SQL Server vNext CTP1 to a Kubernetes cluster in ACS, you will need to update ss-master.yaml and ss-node.yaml files.

Navigate to the **ctp1\kube\deployment\mssql-compute-pool\config** folder.

Open **ss-master.yaml** file and add following lines in **three** places:

**annotations:**

**volume.beta.kubernetes.io/storage-class: "managed-premium"**

|  |
| --- |
| apiVersion: apps/v1beta1  kind: StatefulSet  <…>  spec:  serviceName: service-master  <…>  volumeClaimTemplates:  - metadata:  name: pvc  **annotations:**  **volume.beta.kubernetes.io/storage-class: "managed-premium"**  spec:  accessModes: ["ReadWriteOnce"]  <…>  - metadata:  name: pvc-hdfs  **annotations:**  **volume.beta.kubernetes.io/storage-class: "managed-premium"**  spec:  accessModes: ["ReadWriteOnce"]  <…>  - metadata:  name: pvc-jobserver  **annotations:**  **volume.beta.kubernetes.io/storage-class: "~~default~~managed-premium"**  spec:  accessModes: ["ReadWriteOnce"]  <…> |

Open **ss-node.yaml** file and add following lines in **two** places:

**annotations:**

**volume.beta.kubernetes.io/storage-class: "managed-premium"**

|  |
| --- |
| apiVersion: apps/v1beta1  kind: StatefulSet  metadata:  name: mssql-compute-pool-node  <…>  volumeClaimTemplates:  - metadata:  name: pvc  **annotations:**  **volume.beta.kubernetes.io/storage-class: "managed-premium"**  spec:  accessModes: ["ReadWriteOnce"]  <…>  - metadata:  name: pvc-hdfs  **annotations:**  **volume.beta.kubernetes.io/storage-class: "managed-premium"**  spec:  accessModes: ["ReadWriteOnce"]  <…> |

### Step 4: Configure Number of Replicas

The default configuration will deploy 2 SQL Server compute instances. If you are running on a Kubernetes cluster with sufficient resources, then the number of compute pool instances to be deployed can be increased. Open **ss-node.yaml** file and modify the **replicas** value to your desired number of SQL Server compute instances:

|  |
| --- |
| apiVersion: apps/v1beta1  kind: StatefulSet  metadata:  name: mssql-compute-pool-node  spec:  serviceName: service-node  selector:  matchLabels:  app: mssql-compute-pool-node  replicas: **2**  … |

### Step 5: Deployment

The deployment commands are slightly different if using Windows or Linux. Choose the steps below depending on which operating system you are using.

#### Windows

Using a **CMD** window (**not PowerShell**), navigate to the **ctp1\kube\deployment\mssql-compute-pool\cmd-scripts** folder. Initialize the following environment variables first. You can set the MSSQL passwords to whatever you like but make sure they are sufficiently complex and don’t use the ! & or ‘ characters. Enter the Docker username and password that are available in the EAP Portal.

SET DOCKER\_USERNAME=<your\_user\_name>

SET DOCKER\_PASSWORD=<your\_password>

SET DOCKER\_EMAIL=<your\_email>

SET MSSQL\_SA\_PASSWORD=<sa\_password\_of\_sql\_instances>

SET MSSQL\_IMPORT\_PASSWORD=<import\_account\_password\_of\_sql\_instances>

SET MSSQL\_EXTERNAL\_PASSWORD=<external\_account\_password\_of\_sql\_instances>

NOTE: Characters ! & ' cannot be used in the MSSQL passwords.

If you are using **ACS** run this command:

deploy-**mssql**-**cluster**.cmd

If you are using **Minikube** run this command:

deploy-mssql-cluster-**local**.cmd

You can ignore the initial error messages about various resources not being found.

**This step can take several minutes to run.**

While the command window is showing the status ‘waiting for **master** node to be running’, you can check the deployment status by running this command:

kubectl describe pod mssql-compute-pool-**master**-0

While the command window is showing the status ‘waiting for **compute** node to be running’, you can check the deployment status by running this command:DOCKER

kubectl describe pod mssql-compute-pool-**node**-0

#### Linux

Navigate to the **ctp1/kube/deployment/mssql-compute-pool/sh-scripts** folder. Initialize the following environment variables first. You can set the MSSQL passwords to whatever you like but make sure they are sufficiently complex and don’t use the ! & or ‘ characters. Enter the Docker username and password that are available in the EAP Portal.

export DOCKER\_USERNAME=<your\_user\_name>

export DOCKER\_PASSWORD=<your\_password>

export DOCKER\_EMAIL=<your\_email>

export MSSQL\_SA\_PASSWORD=<sa\_password\_of\_sql\_instances>

export MSSQL\_IMPORT\_PASSWORD=<import\_account\_password\_of\_sql\_instances>

export MSSQL\_EXTERNAL\_PASSWORD=<external\_account\_password\_of\_sql\_instances>

NOTE: Characters ! & ' cannot be used in the MSSQL passwords.

Run this command to enable running the scripts:

chmod a+x -R ../sh-scripts

If you are using ACS run this command:

deploy-**mssql**-**cluster**

If you are using Minikube run this command:

deploy-mssql-cluster-**local**

You can ignore the initial error messages about various resources not being found.

**This step can take several minutes to run.**

While the command window is showing the status ‘waiting for **master** node to be running’, you can check the deployment status by running this command:

kubectl describe pod mssql-compute-pool-**master**-0

While the command window is showing the status ‘waiting for **compute** node to be running’, you can check the deployment status by running this command:

kubectl describe pod mssql-compute-pool-**node**-0

There will probably be multiple compute pool pods number 0-n. You can run the above command to see the status of each of them as they are deploying by changing the number at the end.

### Step 6: Getting the IP Address and Port to Connect To

#### Using ACS

Azure provides Azure LoadBalancer service to ACS. To expose SQL Server vNext CTP1 instance to a public IP, enable Kubernetes service using the LoadBalancer.

Navigate to **ctp1\kube\deployment\mssql-compute-pool\config** folder and run following commands:

kubectl apply -f svc-master-lb.yaml

kubectl get service mssql-compute-pool-svc-lb

Look for the **External-IP** value that is assigned to the service. Then, connect to the SQL Server head node instance using the IP address at port **31433**.

#### Using Minikube

If you are using **Minikube,** you need to run the following command to get the IP address you need to connect to.

minikube status

You will use this IP address & port number **31433** to connect to the SQL Server head node.

# CREATE Compute Pool

Connect to the head node using the IP address / port number (31433) obtained at the end of the deployment script. You can now create a compute pool on the head node. Execute the batch below to create a compute pool that consists of 2 instances. If you modified the number of compute instances in the ss-node.yml above, then change the value of **@cluster\_node\_count** variable to match.

USE high\_value\_data

GO

DECLARE @compute\_pool\_name NVARCHAR(max) = 'mssql-compute-pool'

DECLARE @cluster\_node\_count INT = 2

PRINT 'STEP 1: Initialize compute cluster'

-- sp\_compute\_pool\_create will do the following:

-- 1. setup up 'map' management

-- 2. create credential for head node --> compute node authentication

-- 3. create compute node databases and configure 'map'

-- 4. create EXTERNAL DATASOURCE on head node for compute cluster

--

EXEC sp\_compute\_pool\_create @compute\_pool\_name, @cluster\_node\_count

GO

# Head Node Setup

After successful creation of the compute pool, you can now configure the compute instances to receive data from a Spark streaming job and query the data from the head node. For this example, we will derive the schema of the table in SQL Server from a sample file that resides on an HDFS volume on the newly created cluster. Execute the batch below to create an external table (airlinedata) on the head node and tables on the compute instances to hold data from the Spark streaming job:

PRINT 'STEP 2: Derive table schema from sample file (using Spark) and create SQL Server table schema'

-- sp\_compute\_pool\_create\_table will do the following:

--    1. Submit Resource Negotiator request to submit Spark job to derive T/SQL schema

--    2. Create TABLE on each compute node

--    3. Create EXTERNAL TABLE on the head node

--

DECLARE @compute\_pool\_name NVARCHAR(max) = 'mssql-compute-pool'

DECLARE @table\_name NVARCHAR(max) = 'airlinedata'

DECLARE @sample\_file\_name NVARCHAR(max) = 'hdfs:///airlinedata/airlinedata\_sample.csv'

EXEC sp\_compute\_pool\_create\_table @compute\_pool\_name, @table\_name, @sample\_file\_name

GO

# Spark Streaming to Compute Pool

Now, you can start streaming data from a Spark streaming job into the compute instances. Execute the batch below to start a Spark streaming job that will stream data into the table airlinedata on each compute node:

PRINT 'STEP 3: Start/Stop Spark stream (Submit Spark job to Resource Negotiator)'

--  Submit Spark job to ingest data from specified folder (files) into compute instances

--

DECLARE @compute\_pool\_name NVARCHAR(max) = 'mssql-compute-pool'

DECLARE @table\_name NVARCHAR(max) = 'airlinedata'

DECLARE @source\_folder NVARCHAR(max) = 'hdfs:///airlinedata'

EXEC sp\_compute\_pool\_start\_import @compute\_pool\_name, @table\_name, @source\_folder;

GO

# Query from Head Node

After the streaming job has been successfully started, you can now query the data from the head node. Execute the batch below to run some sample queries. If you run the query a few times you should see that the number of rows in the results is increasing as the data is continuously ingested into the compute instances via the Spark streaming job.

-- View data via fan-out queries

--

SELECT count(\*) FROM airlinedata

SELECT TOP 10 \* FROM airlinedata

GO

# Join High Volume Data and High Value Data

Now that you have populated the high-volume streaming data into the compute instances, you can execute the script “**prepare\_high\_value\_db\_view.sql**” script under the /**samples-tsql** folder to create some sample high-value data in the head node.

The last step is a to execute a query to join the high-volume data in the compute instances represented by the newly created view AirlineEngineSensorDataNorm with the high-value data in AirlineEngines, AircraftRegistration, and FlightRoutes.

-- Join high-value data with the high-volume data in compute pool

--

SELECT E.AircraftRegistration, F.Origin, F.Destination, A.\*

FROM AirlineEngineSensorDataNorm AS A

JOIN high\_value\_data.dbo.AirlineEngines AS E

ON E.EngineId = A.EngineId

JOIN high\_value\_data.dbo.FlightRoutes AS F

ON F.AircraftRegistration = E.AircraftRegistration AND F.EngineId = E.EngineId

WHERE A.EngineId IN (9, 48);

GO