SQL Server vNext CTP1

Contents

[Overview 2](#_Toc500516383)

[Setup Overview 3](#_Toc500516384)

[Kubernetes Cluster Setup 3](#_Toc500516385)

[Deploy on Minikube 3](#_Toc500516386)

[Deploy on Azure Container Services (ACS or AKS) 4](#_Toc500516387)

[Step 1: Launch Azure Cloud Shell 4](#_Toc500516388)

[Step 2: Create a resource group 5](#_Toc500516389)

[Step 3: Create a Kubernetes cluster 5](#_Toc500516390)

[Connect to the cluster 7](#_Toc500516391)

[Deploy SQL Server vNext CTP1 in Kubernetes cluster 7](#_Toc500516392)

[Verify K8S cluster Config 7](#_Toc500516393)

[Configure persistent storage claim 7](#_Toc500516394)

[Configure Number of Replicas 9](#_Toc500516395)

[Deployment from Windows 9](#_Toc500516396)

[Deployment from Linux 10](#_Toc500516397)

[Exposing mssql-compute-pool-master to public IP (ACS and AKS only) 10](#_Toc500516398)

[Compute Pool Setup 11](#_Toc500516399)

[Head Node Setup 11](#_Toc500516400)

[Spark Streaming to Compute Pool 12](#_Toc500516401)

[Query from Head Node 12](#_Toc500516402)

[Bring your own Data 12](#_Toc500516403)

[Bring your own data into Compute Pool 12](#_Toc500516404)

[Bring your high value data into Head Node 14](#_Toc500516405)

[Troubleshooting 16](#_Toc500516406)

[Dynamic Management Views 16](#_Toc500516407)

[dm\_compute\_pools 16](#_Toc500516408)

[dm\_compute\_pool\_node\_status 17](#_Toc500516409)

[dm\_compute\_pool\_jobs 17](#_Toc500516410)

[dm\_compute\_pool\_table\_status 18](#_Toc500516411)

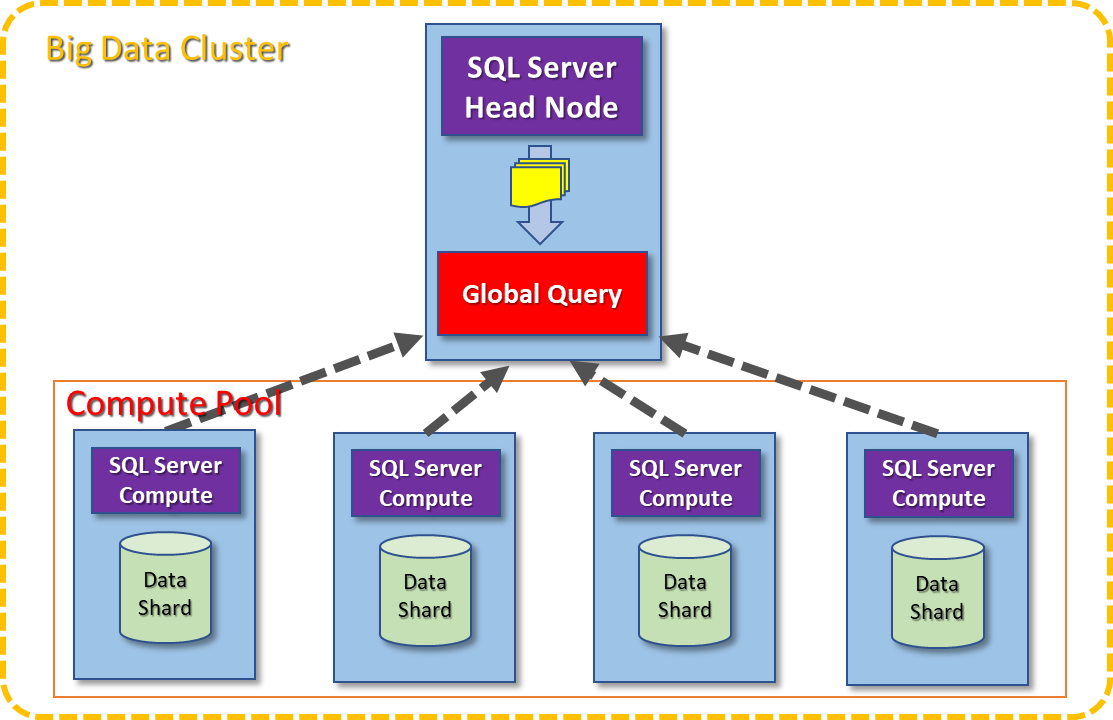
[sp\_compute\_pool\_log 18](#_Toc500516412)

# 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. Polybase technology provides external table mechanism to query data that resides in your Hadoop cluster. Additionally, a Polybase scale-out group can be used to run fan out queries against your external data. This leverages multiple SQL Server instances that can be used to execute Polybase queries.

SQL Server vNext CTP1 now allows you to create and define a compute pool that consists of many SQL Server nodes in your big data cluster. This provides the ability to dynamically deploy SQL Server nodes into your big data cluster, ingest data from Spark streaming jobs into the compute nodes and query the data from the head node via an external table mechanism.

The diagram below shows the stand-alone 5 node configuration (1 head node + 4 compute nodes) in a Hadoop cluster with SQL & Spark running on each nodes.



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 into SQL Server compute pool
* Query – query stream data in SQL Server compute pool and combine with high-value data
* Consume – visualize data with BI tools

# Setup Overview

SQL Server vNext CTP1 can be deployed as docker containers on a Kubernetes cluster. This section describes 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 environment then you can skip directly to [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 the 3 environments (click on item below to jump to appropriate section):

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

## Deploy on Minikube

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 minikube cluster with CTP1, it is recommended that your machine has at least 32 GB of RAM. If the machine has insufficient memory, then modify the cluster configuration such that only 3 nodes are created – 1 head node & 2 compute nodes.

1. **Prerequisites**

VT-x or AMD-v virtualization must be enabled in your computer’s BIOS.

1. **Install a Hypervisor**

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).

1. **Configure external switch in Hypervisor**

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/).

1. **Install kubectl**

kubectl is the Kubernetes command-line tool to deploy and manage Kubernetes. Using kubectl, you can inspect cluster resources, create, delete, and update components, and look at your new cluster and bring up example apps.

[Install kubectl](https://kubernetes.io/docs/tasks/tools/install-kubectl/)

1. **Install Minikube**
   1. Install Minikube according to the instructions for the [latest release](https://github.com/kubernetes/minikube/releases).
2. **Create a Minikube cluster**

Below command creates a minikube cluster with 16 CPUs, 80 GB memory, and disk size of 100GB. This also specifies the hyper-v switch with external access explicitly. Change the parameters such as –memory as needed depending on your available hardware.

minikube start --vm-driver="hyperv" --cpus 16 --memory 80920 --disk-size 100g --hyperv-virtual-switch "Extrernal Switch"

1. **Disable automatic checkpoint**

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

powershell -Command "Set-VM -Name minikube -CheckpointType Disabled -AutomaticCheckpointsEnabled $false"

## Deploy on Azure Container Services (ACS or AKS)

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: Launch Azure Cloud Shell

The Azure Cloud Shell is a free Bash shell that you can run directly within the Azure portal. It has the Azure CLI preinstalled and configured to use with your account. Click the **Cloud Shell** button on the menu in the upper-right of the [Azure portal](https://portal.azure.com/).

[Cloud Shell](https://portal.azure.com/)

If you choose to install and use the CLI locally, this section requires that you are running the Azure CLI version 2.0.4 or later. Run az --version to find the version. If you need to install or upgrade, see [Install Azure CLI 2.0](https://docs.microsoft.com/en-us/cli/azure/install-azure-cli).

### Step 2: 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 *k8acsResourceGroup* in the *westus* location.

az group create --name k8acsResourceGroup --location westus

### Step 3: Create a Kubernetes cluster

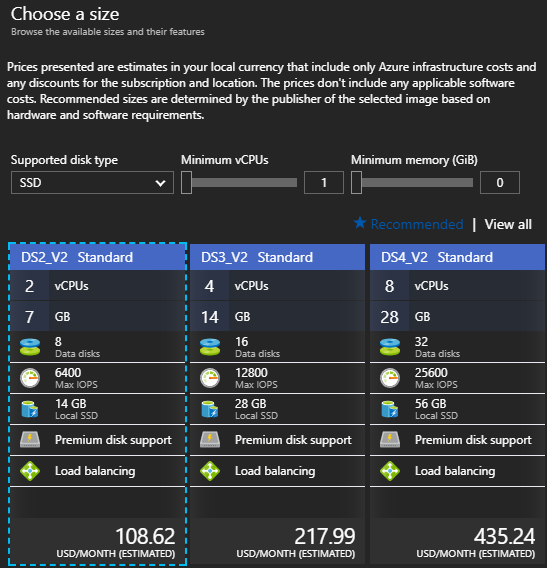
Create a Kubernetes cluster in Azure Container Service with the [az acs create](https://docs.microsoft.com/en-us/cli/azure/acs) command. The following example creates a cluster named *arisK8Cluster* with one Linux master node and three Linux agent nodes.

az acs create --orchestrator-type kubernetes --resource-group k8acsResourceGroup --name arisK8Cluster --generate-ssh-keys

You 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. After several minutes, the command completes and returns json formatted information about the cluster.

##### Using AKS instead of ACS

[AKS](https://azure.microsoft.com/en-us/blog/introducing-azure-container-service-aks-managed-kubernetes-and-azure-container-registry-geo-replication/) is a managed Kubernetes service in Azure. The VMs used for the cluster should be based on an image that supports premium disks. See screenshot below that shows how to determine the correct VM name with premium disk support.



To provision a cluster using AKS, run following commands in Azure CLI.

|  |
| --- |
| az provider register -n Microsoft.ContainerService  az provider register -n Microsoft.Compute  az provider register -n Microsoft.Network  az group create -n k8aksResourceGroup -l westus2  az aks create -g k8aksResourceGroup \  -n arisK8sCluster \  --agent-count 3 \  --agent-vm-size DS2\_V2 \  --generate-ssh-keys |

Change **--agent-vm-size** value to a desired VM size or --agent-count if needed

### Connect to the cluster

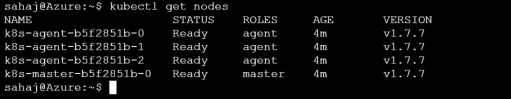
To manage a Kubernetes cluster, use [kubectl](https://kubernetes.io/docs/user-guide/kubectl/), the Kubernetes command-line client. If you're using Azure CloudShell, kubectl is already installed. If you want to install it locally, you can use the [az acs kubernetes install-cli](https://docs.microsoft.com/en-us/cli/azure/acs/kubernetes) command.

To configure kubectl to connect to your Kubernetes cluster, run the [az aks 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 aks get-credentials --resource-group=k8aksResourceGroup --name=arisK8sCluster

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 we have a Kubernetes cluster setup using either minikube or ACS or VMs, we can proceed with the deployment for SQL Server vNext CTP1. After this point, you would have a configuration for the K8S cluster in your machine that will be used further by the SQL Server vNext CTP1 deployment script. This step can be repeated multiple times to recreate the environment. Before starting the deployment, you need the credentials to the private docker registry containing SQL Server vNext CTP1 image(s).

Before proceeding with the deployment, make sure you have the following information:

***Docker credentials to the private repository. This was sent to in the CTP1 private preview information email.***

## 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

## Configure persistent storage claim

For minikube deployment, no change is required to configure persistence. To deploy SQL Server vNext CTP1 to Kubernetes cluster in AKS, you will need to update ss-master.yaml and ss-node.yaml files.

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

#### Persistent storage claim for master node

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

**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"]  <…> |

#### Persistent storage claim for data node

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

**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"]  <…> |

## Configure Number of Replicas

The default configuration will deploy 2 compute pool nodes. If you are running on a K8S cluster with sufficient resources then the number of compute pool nodes to be deployed can be modified. Open **ss-node.yaml** file and modify the **replicas** value to your desired number of compute pool nodes:

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

## Deployment from Windows

Using a **CMD** windows (not PowerShell), navigate to the **kube\deployment\mssql-compute-pool\cmd-scripts** folder. Initialize the following environment variables first:

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

SET DOCKER\_PASSWORD=<your\_password>

SET DOCKER\_EMAIL=<your\_email>

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

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

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

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

Execute **deploy-mssql-cluster.CMD** script. Note the IP address of the head node that will be printed at the end of a successful deployment. You will use this IP address & port number**31433** to connect to SQL Server running as the head node.

## Deployment from Linux

Navigate to the **kube\deployment\mssql-compute-pool\sh-scripts** folder. Initialize the following environment variables first:

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

export DOCKER\_PASSWORD=<your\_password>

export DOCKER\_EMAIL=<your\_email>

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

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

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

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

Execute **deploy-mssql-cluster** script. Note the IP address of the head node that will be printed at the end of a successful deployment. You will use this IP address & port number **31433** to connect to SQL Server running as the head node.

## Exposing mssql-compute-pool-master to public IP (ACS and AKS only)

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

Navigate to kube\deployment\mssql-compute-pool\config folder and run following command:

kubectl apply -f svc-master-lb.yaml

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

Wait until **external-ip** value is assigned to the service then connect SQL Server instance using the IP at port 1433.

# Compute Pool Setup

Connect to the head node using the IP address / port number (31433) obtained at the end of the deployment script. You can now configure a compute pool on the sample database (high\_value\_data) on the head node. Execute the batch below to create a compute pool that consists of 2 nodes. If you modified the number of compute pool nodes then change the value of **@cluster\_node\_count** variable appropriately.

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'

-- 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 nodes to receive data from a Spark streaming job and query the data from the head node. For this demo, we will derive the schema of the table in SQL Server from a sample file that resides on HDFS volume on the newly setup cluster. Execute the batch below to create an external table (airlinedata) on the head node and tables on the compute node to hold data from Spark streaming job:

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

--    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 nodes. 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 nodes

--

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:

-- View data via fan-out queries

--

SELECT count(\*) FROM airlinedata

SELECT TOP 10 \* FROM airlinedata

GO

Now, that you have successfully the basic scenario you can execute the script “**prepare\_high\_value\_db\_view.sql**” script under the **samples\tsql-scripts** folder in the CTP1 private preview repo.

# Bring your own Data

### Bring your own data into Compute Pool

Httpfs is started when deployment is completed. The service port is set at 32000 (default port 14000 but we mapped it to 32000). Once the deployment completes, we can use the following command to access hdfs.

To list file under hdfs:///airlinedata:

**curl "http://<host\_ip\_address>:32000/webhdfs/v1/airlinedata/?user.name=root&op=liststatus’** **http://<host\_ip\_address>:32000/webhdfs/v1/airlinedata/?user.name=root&op=liststatus’**

To put a new file ext1.csv from local directory to airlinedata directory (Content-Type parameter is required):

**curl -i -X PUT -T ext1.csv ‘http://<host\_ip\_address>:32000/webhdfs/v1/airlinedata/ext1.csv?user.name=root&op=create&data=true’ -H ‘Content-Type: application/octet-stream’**

To get a file from airlinedata directory:

**curl ‘**[**http://** **<host\_ip\_address>:32000/webhdfs/v1/airlinedata/ext1.csv?user.name=root&op=open**](http://10.135.9.28:32000/webhdfs/v1/airlinedata/ext1.csv?user.name=root&op=open)**’ > ext1.csv.cop0079**

To create a directory “test” under hdfs:///:

**curl -i -X PUT ‘http://<host\_ip\_address>:32000/webhdfs/v1/test?op=mkdirs&user.name=root’**

Possible commands that we can submit through httpfs are listed below:

    OPEN(HTTP\_GET),

    GETFILESTATUS(HTTP\_GET),

    LISTSTATUS(HTTP\_GET),

    GETHOMEDIRECTORY(HTTP\_GET),

    GETCONTENTSUMMARY(HTTP\_GET),

    GETFILECHECKSUM(HTTP\_GET),

    GETFILEBLOCKLOCATIONS(HTTP\_GET),

    INSTRUMENTATION(HTTP\_GET),

    GETACLSTATUS(HTTP\_GET),

    APPEND(HTTP\_POST),

    CONCAT(HTTP\_POST),

    CREATE(HTTP\_PUT),

    MKDIRS(HTTP\_PUT),

    RENAME(HTTP\_PUT),

    SETOWNER(HTTP\_PUT),

    SETPERMISSION(HTTP\_PUT),

    SETREPLICATION(HTTP\_PUT),

    SETTIMES(HTTP\_PUT),

    MODIFYACLENTRIES(HTTP\_PUT),

    REMOVEACLENTRIES(HTTP\_PUT),

    REMOVEDEFAULTACL(HTTP\_PUT),

    REMOVEACL(HTTP\_PUT),

    SETACL(HTTP\_PUT),

    DELETE(HTTP\_DELETE),

    SETXATTR(HTTP\_PUT),

    GETXATTRS(HTTP\_GET),

    REMOVEXATTR(HTTP\_PUT),

    LISTXATTRS(HTTP\_GET);

### Bring your high value data into Head Node

To bring an existing SQL Server database into the Head Node, we recommend backup and restore.

1. **Backup your existing SQL Server database**

$ sqlcmd -S <server\_name> -U <username> -P <password> -Q "BACKUP DATABASE <db\_name> TO DISK='<path>\<db\_name>.bak'"

1. **Get list of pods from Kubernetes cluster**

Note the name of the pod with <compute\_pool\_name>-master-0. This is the SQL Server head node.

$ kubectl get pods

NAME READY STATUS RESTARTS AGE

mssql-compute-pool-master-0 2/2 Running 0 1h

mssql-compute-pool-node-0 2/2 Running 0 1h

mssql-compute-pool-node-1 2/2 Running 0 1h

mssql-compute-pool-node-2 2/2 Running 0 1h

mssql-compute-pool-node-3 2/2 Running 0 1h

mssql-compute-pool-node-4 2/2 Running 0 1h

mssql-compute-pool-node-5 2/2 Running 0 1h

mssql-compute-pool-node-6 2/2 Running 0 1h

mssql-compute-pool-node-7 2/2 Running 0 1h

1. **Copy backup file to the SQL Server container in the master pod of the Kubernetes cluster**

$kubectl cp <path>\<db\_name>.bak mssql-compute-pool-master-0:/tmp/ -c mssql-compute-pool-data

1. **Verify that the backup file was copied to the pod container**

$ kubectl exec -it mssql-compute-pool-master-0 -c mssql-compute-pool-data -- bin/bash

root@mssql-compute-pool-master-0:/# cd tmp

root@mssql-compute-pool-master-0:/tmp# ls

<db\_name>.bak

root@mssql-compute-pool-master-0:/tmp# exit

exit

1. **Find the Head node SQL Server IP address**

$ kubectl get services

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S)

AGE

kubernetes ClusterIP 10.0.0.1 <none> 443/TCP

2d

service-master ClusterIP None <none> 8088/TCP,50070/TCP,50075/TCP,50020/TCP,9000/TCP,50010/TCP,8030/TCP,8031/TCP,8032/TCP,8033/TCP,8040/TCP,8042/TCP,8080/TCP,1433/TCP,9995/TCP,8090/TCP,14000/TCP 12m

service-master-nodeport LoadBalancer 10.0.111.236 40.84.230.63 31000:31912/TCP,30001:31035/TCP,30002:30723/TCP,31433:31651/TCP,30003:31311/TCP,30004:31219/TCP,32000:30335/TCP 12m

service-node ClusterIP None <none> 8088/TCP,50070/TCP,50075/TCP,50020/TCP,9000/TCP,50010/TCP,8030/TCP,8031/TCP,8032/TCP,8033/TCP,8040/TCP,8042/TCP,1433/TCP 8m

1. **Restore the database backup to Head node SQL Server**

$ sqlcmd -S <head\_node\_ip> -U <username> -P <password> -Q "RESTORE FILELISTONLY FROM DISK='/tmp/<db\_name>.bak'"

LogicalName PhysicalName Type FileGroupName Size MaxSize FileId CreateLSN DropLSN UniqueId ReadOnlyLSN ReadWriteLSN BackupSizeInBytes SourceBlockSize FileGroupId LogGroupGUID DifferentialBaseLSN DifferentialBaseGUID IsReadOnly IsPresent TDEThumbprint SnapshotUrl

-------------------------------------------------------------------------------------------------------------------------------- ---------------------------------------------

FlightDemo C:\Demo\sqlservr\data\FlightDemo.mdf D PRIMARY 8388608 35184372080640 1 0 0 C5EABDE0-49D0-4FF5-A80A-BC50C45B5321 0 0 3604480 4096 1 NULL 0 00000000-0000-0000-0000-000000000000 0 1 NULL NULL

FlightDemo\_log C:\Demo\sqlservr\data\FlightDemo\_log.ldf L NULL 8388608 2199023255552 2 0 0 F45ABB16-1BDC-48F8-97C6-AE4B41347600 0 0 0 4096 0 NULL 0 00000000-0000-0000-0000-000000000000 0 1 NULL NULL

(2 rows affected)

$ sqlcmd -S <head\_node\_ip> -U <username> -P <password> -Q "RESTORE DATABASE <db\_name> FROM DISK='/tmp/<db\_name>.bak' WITH MOVE '<db\_name>' TO '/tmp/<db\_name>.mdf', MOVE '<db\_name>\_log' TO '/tmp/<db\_name>.ldf'"

Processed 440 pages for database 'FlightDemo', file 'FlightDemo' on file 1.

Processed 3 pages for database 'FlightDemo', file 'FlightDemo\_log' on file 1.

Converting database 'FlightDemo' from version 857 to the current version 888.

Database 'FlightDemo' running the upgrade step from version 857 to version 858.

Database 'FlightDemo' running the upgrade step from version 858 to version 859.

Database 'FlightDemo' running the upgrade step from version 859 to version 860.

Database 'FlightDemo' running the upgrade step from version 860 to version 861.

Database 'FlightDemo' running the upgrade step from version 861 to version 862.

Database 'FlightDemo' running the upgrade step from version 862 to version 863.

Database 'FlightDemo' running the upgrade step from version 863 to version 864.

Database 'FlightDemo' running the upgrade step from version 864 to version 865.

Database 'FlightDemo' running the upgrade step from version 865 to version 866.

Database 'FlightDemo' running the upgrade step from version 866 to version 867.

Database 'FlightDemo' running the upgrade step from version 867 to version 868.

Database 'FlightDemo' running the upgrade step from version 868 to version 869.

Database 'FlightDemo' running the upgrade step from version 869 to version 875.

Database 'FlightDemo' running the upgrade step from version 875 to version 876.

Database 'FlightDemo' running the upgrade step from version 876 to version 877.

Database 'FlightDemo' running the upgrade step from version 877 to version 878.

Database 'FlightDemo' running the upgrade step from version 878 to version 879.

Database 'FlightDemo' running the upgrade step from version 879 to version 880.

Database 'FlightDemo' running the upgrade step from version 880 to version 881.

Database 'FlightDemo' running the upgrade step from version 881 to version 882.

Database 'FlightDemo' running the upgrade step from version 882 to version 883.

Database 'FlightDemo' running the upgrade step from version 883 to version 884.

Database 'FlightDemo' running the upgrade step from version 884 to version 885.

Database 'FlightDemo' running the upgrade step from version 885 to version 886.

Database 'FlightDemo' running the upgrade step from version 886 to version 887.

Database 'FlightDemo' running the upgrade step from version 887 to version 888.

RESTORE DATABASE successfully processed 443 pages in 0.456 seconds (7.581 MB/sec).

1. **Setup compute pool stored procedures**

Execute “**high-value-db-configuration\compute\_pool\_ddl\_install.SQL**” script

1. **Setup supportability stored procedures**

Execute “**high-value-db-configuration\supportability.SQL**” script

# Troubleshooting

## Dynamic Management Views

The sample database (high\_value\_data) created by the deployment script already contains the dynamic management views for troubleshooting. The following new dynamic management views are available for troubleshooting.

### dm\_compute\_pools

This DMV shows the number of nodes in the compute pool

|  |  |
| --- | --- |
| Column | Description |
| Name | Compute pool name |
| Nodes | Number of compute node in a compute pool |

Example usage

SELECT \* FROM dm\_compute\_pools ()

### dm\_compute\_pool\_node\_status

This DMV returns the status of all compute nodes in the compute pool.

|  |  |
| --- | --- |
| Column | Description |
| Node\_id | Computer node id |
| State | State of the compute node |
| Connection\_time |  |
| version | Version of sql server running on the compute node |

Example usage

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

SELECT \* FROM dm\_compute\_pool\_node\_status (@compute\_pool\_name)

### dm\_compute\_pool\_jobs

|  |  |
| --- | --- |
| This DMV returns information about the spark job that was submitted by T-SQL command. Column | Description |
| Duration | Duration that spark job ran |
| Class\_path | Spark job class name |
| Start\_time | Spark job execution start time |
| Context | Spark job context name |
| Status | Status of spark job |
| Job\_id | Spark job id |
| Result\_message | Spark job result message |

Example usage

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

SELECT \* FROM dm\_compute\_pool\_jobs(@compute\_pool\_name)

### dm\_compute\_pool\_table\_status

This DMV returns number of rows in compute node external tables.

|  |  |
| --- | --- |
| Column | Description |
| Node\_id | Compute node id |
| Rows | Number of row in an external table in the compute node |

Example usage

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

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

SELECT \* FROM dm\_compute\_pool\_table\_status (@compute\_pool\_name, @table\_name)

## sp\_compute\_pool\_log

This store procedure shows the log of all the compute pool operations. You can use this stored procedure to troubleshoot the commands between head node & compute nodes and spark job server.