Evaluation Guide: SQL Server on Linux

Contents

Using this Guide

This guide gives you the necessary information to experience SQL Server built on Linux. The objective is to help you build an evaluation environment and walk through the tutorials described in this guide to learn about the core capabilities and features of SQL Server on Linux.

Run and Connect to SQL Server on Linux

View Service Status

SQL Server runs on Linux as a service that is started as a default after installation.

> Note: these commands are not appropriate for checking the status of SQL Server running in a container. Use `$ sudo docker ps` instead.

Use the following command to view the status of the SQL Server service:

```
$ systemctl status mssql-server
```

Stop and Start the Service

Use the following command to stop the SQL Server service:

```
$ sudo systemctl stop mssql-server
```

Use the following command to start the SQL Server service:

```
$ sudo systemctl start mssql-server
```

Retrieve the IP Address

You can connect to your SQL Server instance remotely from your tool of choice using the machine's IP address.

On Ubuntu, retrieve the IP address with the `ifconfig` command:

```
$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0F:20:CF:8B:42
          inet addr:217.149.127.10  Bcast:217.149.127.63  Mask:255.255.255.192
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2472694671 errors:1 dropped:0 overruns:0 frame:0
          TX packets:44641779 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1761467179 (1679.7 Mb)  TX bytes:2870928587 (2737.9 Mb)
          Interrupt:28
```

In Docker on Mac OS, run the following command to get the IP address of the VM that hosts Docker:

```
$ docker-machine ip <Docker VM name>
217.149.127.10
```

Setup a Sample Database

Install AdventureWorks

To get started with a sample database, download AdventureWorks2014.zip (Note that the version of AdventureWorks for SQL Server 2016 is not supported for SQL Server on Linux at this time because it exercises features that are not yet supported. Additionally, SQL Server on Linux only supports restoring databases that are backed up from SQL Server version 2016 CTP3 or lower at this time.)

Extract the zip file and place AdventureWorks.bak in the following directory: **/var/opt/mssql/data**

Execute the T-SQL statements below to create the AdventureWorks database. Note that the virtual "C:\" path maps to the working directory (/var/opt/mssql) for SQL Server in this preview release.

For example, "C:**\data\AdventureWorks2014.bak**" is the way to specify a path to "/var/opt/mssql/**data/AdventureWorks2014.bak**". We will be addressing this path idiosyncrasy in a future release.

```
USE [master]

RESTORE DATABASE AdventureWorks
FROM DISK = 'C:\data\AdventureWorks2014.bak'
WITH MOVE 'AdventureWorks2014_Data' TO 'C:\data\AdventureWorks2014_Data.mdf',
MOVE 'AdventureWorks2014_Log' TO 'C:\data\AdventureWorks2014_Log.ldf'
```

Basic Tasks

Query Tables

The data types, schemas, stored procedures and functions in AdventureWorks are documented here.

Execute the simple query below to list all departments and the groups they belong go:

```
SELECT Name, GroupName
FROM HumanResources.Department
```

Execute a query with a join to look up the Product and Product Subcategory associated with ProductID 800:

```
SELECT Product.ProductID, Product.Name as Name, Product.ProductNumber,
ProductSubcategory.Name as ProductSubcategory
FROM Production.Product
JOIN Production.ProductSubcategory
ON ProductSubcategory.ProductSubcategoryID = Product.ProductSubcategoryID
```

```
WHERE ProductID = 800
```

Execute a Stored Procedure

Execute the following query to run the `uspgetBillOfMaterials` stored procedure, which returns a recursive list of components used to manufacture ProductID 800:

```
USE AdventureWorks;
GO
DECLARE @CheckDate DATETIME;
SET @CheckDate = GETDATE();
EXEC [AdventureWorks].[dbo].[uspGetBillOfMaterials] 800, @CheckDate;
```

Update a Record

Update a row in the WorkOrder table to reflect that one quantity of ProductID 800 failed inspection and had to be scrapped.

```
Update Production.WorkOrder
SET ScrappedQty = 1
WHERE WorkOrderID = 70370
```

Bulk Insert

The Bulk Insert feature allows you to import records into a table or view from a local data file.

To test this feature, create a new directory /var/opt/mssql/**Samples** and a new file **SampleLocations.txt** with the following content:

1,Subassembly
2,Storage
3,Assembly
4,Final Assembly
5,Paint Shop

Linux and Windows handle newlines differently and SQL Server on Linux only recognizes newline characters in the Windows format this preview release. If you authored the SampleLocations.txt file in Linux, download the dos2unix package and use it to create a version of the file that will be interpreted correctly by SQL Server.

```
$ apt-get install dos2unix
…
$ unix2dos –n SampleLocations.txt WinSampleLocations.txt
```

Create a new table in tempdb for Location data:

```
use tempdb

CREATE TABLE Locations (
```

```
LocationID int,
Location varchar(255)
)
```

Use Bulk Insert to populate the Locations table with data from your text file:

```
BULK INSERT Locations FROM 'C:\Samples\WinSampleLocations.txt' WITH
(FIELDTERMINATOR = ',')
```

Security

Server Audit

SQL Server audits allow you to track and log events that occur on the Database Engine. The steps below show you how to define an audit. After the audit is created and enabled, the target will receive entries.

If you haven't already created a /var/opt/mssql/**Samples** directory, create a new directory and make sure that SQL Server has access to it and everything else in the mssql working directory:

```
$ mkdir Samples
$ chown -R mssql:mssql .
```

Create an audit and define the target file:

```
USE [master]

CREATE SERVER AUDIT [TestAudit]
TO FILE
(    FILEPATH = 'C:\Samples'
,MAXSIZE = 0 MB
,MAX_ROLLOVER_FILES = 2147483647
,RESERVE_DISK_SPACE = OFF
)
WITH
(    QUEUE_DELAY = 1000
,ON_FAILURE = CONTINUE
,AUDIT_GUID = 'e53eb508-875d-495b-a3cb-15af580e7e3b'
)
```

Enable the audit:

```
ALTER SERVER AUDIT [TestAudit] WITH (STATE = ON)
```

Map the database to the audit with a server audit specification:

```
CREATE SERVER AUDIT SPECIFICATION [TestServerAuditSpecification]
FOR SERVER AUDIT [TestAudit]
ADD (SCHEMA_OBJECT_ACCESS_GROUP)
WITH (STATE = ON)
```

Note you can also create a database audit specification, which is scoped to an individual database rather than the whole server. For a complete list of the types of actions and groups of actions that can be audited, see SQL Server Audit Action Groups and Actions.

Read the audit events:

```
SELECT * FROM fn_get_audit_file('C:\Samples\TestAudit*.sqlaudit', null, null)
```

Configure Row-Level Security

Row-Level Security enables you to restrict access to rows in a database based on the user executing a query. This feature is useful for scenarios like ensuring that customers can only access their own data or that workers can only access data that is pertinent to their department.

The steps below walk through setting up two Users with different row-level access to the Sales.SalesOrderHeader table.

Create two user accounts to test the row level security.

```
CREATE USER Manager WITHOUT LOGIN;
CREATE USER SalesPerson280 WITHOUT LOGIN
```

Grant read access on the Sales.SalesOrderHeader table to both users.

```
GRANT SELECT ON Sales.SalesOrderHeader TO Manager;
GRANT SELECT ON Sales.SalesOrderHeader TO SalesPerson280;
```

Create a new schema and inline table-valued function. The function returns 1 when a row in the SalesPersonID column matches the ID of a SalesPerson login or if the user executing the query is the Manager user.

```
CREATE SCHEMA Security;

GO

CREATE FUNCTION Security.fn_securitypredicate(@SalesPersonID AS int)
    RETURNS TABLE
WITH SCHEMABINDING
AS
    RETURN SELECT 1 AS fn_securitypredicate_result
```

```
WHERE ('SalesPerson' + CAST(@SalesPersonId as VARCHAR(16)) = USER_NAME()) OR
(USER_NAME() = 'Manager');
```

Create a security policy adding the function as both a filter and a block predicate on the table:

```
CREATE SECURITY POLICY SalesFilter
ADD FILTER PREDICATE Security.fn_securitypredicate(SalesPersonID)
ON Sales.SalesOrderHeader,
ADD BLOCK PREDICATE Security.fn_securitypredicate(SalesPersonID)
ON Sales.SalesOrderHeader
WITH (STATE = ON);
```

Execute the following to query the SalesOrderHeader table as each user. Verify that SalesPerson280 only sees the 95 rows from their own sales and that the Manager can see all the rows in the table.

```
EXECUTE AS USER = 'SalesPerson280';
SELECT * FROM Sales.SalesOrderHeader;
REVERT;

EXECUTE AS USER = 'Manager';
SELECT * FROM Sales.SalesOrderHeader;
REVERT;
```

Alter the security policy to disable the policy. Now both users can access all rows.

```
ALTER SECURITY POLICY SalesFilter
WITH (STATE = OFF);
```

Enable Always Encrypted

Always Encrypted is a security feature that protects sensitive data by encrypting it inside the client application. SQL Server only handles the encrypted data and never has access to the encryption keys or the plaintext values. Even if the SQL Server instance or the host machine is compromised, all an attacker would get is a ciphertext of sensitive data.

To enable Always Encryped, walk through the Getting Started with Always Encrypted blog post to create a table with Always Encryped. Note that this tutorial requires SQL Server Management Studio and Visual Studio on Windows.

Enable Dynamic Data Masking

Dynamic Data Masking enables you to limit the exposure of sensitive data to users of an application by fully or partially masking certain columns.

Use an ALTER TABLE statement to add a masking function to the EmailAddress column in the Person.EmailAddress table:

```
ALTER TABLE Person.EmailAddress
ALTER COLUMN EmailAddress ADD MASKED WITH (FUNCTION = 'email()');
```

Create a new user TestUser with SEELCT permission on the table, then execute a query as TestUser to view the masked data:

```
CREATE USER TestUser WITHOUT LOGIN;
GRANT SELECT ON Person.EmailAddress TO TestUser;

EXECUTE AS USER = 'TestUser';
SELECT * FROM Person.EmailAddress
REVERT;
```

Verify that the masking function changes the email address in the first record from

1	1	**ken0@adventure-works.com**	8A1901E4-671B-431A-871C-EADB2942E9EE	2009-01-07 00:00:00.000

into

1	1	**kXXX@XXXX.com**	8A1901E4-671B-431A-871C-EADB2942E9EE	2009-01-07 00:00:00.000

Performance

Create a Columnstore Index

A columnstore index is a technology for storing and querying large stores of data in a columnar data format, called a columnstore.

Add a Columnstore index to the SalesOrderDetail table by executing the T-SQL below:

```
CREATE NONCLUSTERED COLUMNSTORE INDEX [IX_SalesOrderDetail_ColumnStore]
ON Sales.SalesOrderDetail
(UnitPrice, OrderQty, ProductID)
```

Execute the following query that will use the Columnstore Index to scan the table:

```
SELECT ProductID, SUM(UnitPrice) SumUnitPrice, AVG(UnitPrice) AvgUnitPrice,
SUM(OrderQty) SumOrderQty, AVG(OrderQty) AvgOrderQty
FROM Sales.SalesOrderDetail
GROUP BY ProductID
ORDER BY ProductID
```

Verify that the Columnstore Index was used by looking up the object_id for the Columnstore index and confirming that it appears in the usage stats for the SalesOrderDetail table:

```
SELECT * FROM sys.indexes WHERE name = 'IX_SalesOrderDetail_ColumnStore'
```

```
GO

SELECT *
FROM sys.dm_db_index_usage_stats
WHERE database_id = DB_ID('AdventureWorks')
AND object_id = OBJECT_ID('AdventureWorks.Sales.SalesOrderDetail');
```

Use In-Memory OLTP

SQL Server provides In-Memory OLTP features that can greatly improve the performance of application systems. This section of the Evaluation Guide will walk you through the steps to create a memory-optimized table stored in memory and a natively compiled stored procedure that can access the table without needing to be compiled or interpreted.

Configure Database for In-Memory OLTP

1. It's recommended to set the database to a compatibility level of at least 130 to use In-Memory OLTP. Use the query below to check the current compatibility level of AdventureWorks:

```
USE AdventureWorks

SELECT d.compatibility_level
    FROM sys.databases as d
    WHERE d.name = Db_Name();
```

If necessary, update the level to 130:

```
ALTER DATABASE CURRENT
    SET COMPATIBILITY_LEVEL = 130;
```

2. When a transaction involves both a disk-based table and a memory-optimized table, it's essential that the memory-optimized portion of the transaction operate at the transaction isolation level named SNAPSHOT. To reliably enforce this level for memory-optimized tables in a cross-container transaction, execute the following:

```
ALTER DATABASE CURRENT SET MEMORY_OPTIMIZED_ELEVATE_TO_SNAPSHOT=ON
```

3. Before you can create a memory-optimized table you must first create a Memory Optimized FILEGROUP and a container for data files:

```
ALTER DATABASE AdventureWorks ADD FILEGROUP AdventureWorks_mod CONTAINS
memory_optimized_data

ALTER DATABASE AdventureWorks ADD FILE (NAME='AdventureWorks_mod',
FILENAME='c:\data\AdventureWorks_mod') TO FILEGROUP AdventureWorks_mod
```

Create a Memory-Optimized Table

The primary store for memory-optimized tables is main memory and so unlike disk-based tables, data does not need to be read in from disk into memory buffers. To create a memory-optimized table, use the MEMORY_OPTIMIZED = ON clause.

1. Execute the following query to create the memory-optimized table dbo.ShoppingCart. As a default, the data will be persisted on disk for durability purposes (Note that DURABILITY can also be set to persist the schema only).

```
  CREATE TABLE dbo.ShoppingCart (
    ShoppingCartId INT IDENTITY(1,1) PRIMARY KEY NONCLUSTERED,
    UserId INT NOT NULL INDEX ix_UserId NONCLUSTERED HASH WITH
(BUCKET_COUNT=1000000),
    CreatedDate DATETIME2 NOT NULL,
    TotalPrice MONEY
    ) WITH (MEMORY_OPTIMIZED=ON)
  GO
```

2. Insert some records into the table:

```
INSERT dbo.ShoppingCart VALUES (8798, SYSDATETIME(), NULL)
INSERT dbo.ShoppingCart VALUES (23, SYSDATETIME(), 45.4)
INSERT dbo.ShoppingCart VALUES (80, SYSDATETIME(), NULL)
INSERT dbo.ShoppingCart VALUES (342, SYSDATETIME(), 65.4)
```

Natively compiled Stored Procedure

SQL Server supports natively compiled stored procedures that access memory-optimized tables. The T-SQL statements are compiled to machine code and stored as native DLLs, enabling faster data access and more efficient query execution than traditional T-SQL. Stored procedures that are marked with NATIVE_COMPILATION are natively compiled.

1. Execute the following script to create a natively compiled stored procedure that inserts a large number of records into the ShoppingCart table:

```
CREATE PROCEDURE dbo.usp_InsertSampleCarts @InsertCount int
 WITH NATIVE_COMPILATION, SCHEMABINDING
 AS
 BEGIN ATOMIC
 WITH (TRANSACTION ISOLATION LEVEL = SNAPSHOT, LANGUAGE = N'us_english')

  DECLARE @i int = 0

  WHILE @i < @InsertCount
  BEGIN
    INSERT INTO dbo.ShoppingCart VALUES (1, SYSDATETIME() , NULL)
    SET @i += 1
```

```
  END

END
```

2. Insert 1,000,000 rows:

```
EXEC usp_InsertSampleCarts 1000000
```

3. Verify the rows have been inserted:

```
SELECT COUNT(*) FROM dbo.ShoppingCart
```

Learn More About In-Memory OLTP

Detailed quick start: https://msdn.microsoft.com/library/mt694156.aspx

Migrating to In-Memory OLTP: https://msdn.microsoft.com/en-us/library/dn247639.aspx

Optimizing temp table and table variable scenarios: https://msdn.microsoft.com/en-us/library/mt718711.aspx

Monitoring memory: https://msdn.microsoft.com/en-us/library/dn465869.aspx

General In-Memory OLTP documentation: https://msdn.microsoft.com/en-us/library/dn133186.aspx

Use Query Store

Query Store collects detailed performance information about queries, execution plans, and runtime statistics.

Query Store is not active by default and can be enabled with ALTER DATABASE:

```
ALTER DATABASE AdventureWorks SET QUERY_STORE = ON;
```

Run the following query to return information about queries and plans in the query store:

```
SELECT Txt.query_text_id, Txt.query_sql_text, Pl.plan_id, Qry.*
FROM sys.query_store_plan AS Pl
JOIN sys.query_store_query AS Qry
    ON Pl.query_id = Qry.query_id
JOIN sys.query_store_query_text AS Txt
    ON Qry.query_text_id = Txt.query_text_id ;
```

Query Dynamic Management Views

Dynamic management views return server state information that can be used to monitor the health of a server instance, diagnose problems, and tune performance.

To query the dm_os_wait stats dynamic management view:

```
SELECT wait_type, wait_time_ms
FROM sys.dm_os_wait_stats;
```

Lifecycle Management

Detach and Attach Database

The data and transaction log files of a database can be detached and then reattached to the same or another instance of SQL Server, useful for upgrading or moving the database.

Detaching a database will remove it from the instance of SQL Server but leave the database and its data and log files intact. To detach the AdventureWorks database:

```
USE [master]

EXEC master.dbo.sp_detach_db @dbname = N'AdventureWorks'
```

To reattach the database:

```
USE [master]

CREATE DATABASE [AdventureWorks] ON
( FILENAME = 'C:\data\AdventureWorks2014_Data.mdf' ),
( FILENAME = 'C:\data\AdventureWorks2014_Log.ldf' )
 FOR ATTACH
```

Backup, Drop, and Restore Database

Issue a full backup of the AdventureWorks database (this command will create the Backup directory for you if it doesn't exist):

```
USE [master]

BACKUP DATABASE AdventureWorks TO DISK = 'C:\Backup\AdventureWorks.bak'
```

Drop the AdventureWorks database by executing the T-SQL below:

```
USE [master]

DROP DATABASE [AdventureWorks]
```

Restore AdventureWorks from backup by executing the T-SQL below. Note that at this time SQL Server on Linux only supports restoring databases that are backed up from SQL Server version 2016 CTP3 or lower.

```
USE [master]

RESTORE DATABASE [AdventureWorks] FROM DISK = N'C:\Backup\AdventureWorks.bak'
```

Note: Make sure that both the owner and group of the database are "mssql" for SQL Server engine to recognize the files.

Development and Management

Using SQL Server Management Studio (SSMS)

You can use SSMS running on your Windows computer to connect to and work with SQL Server running anywhere, including SQL Server on Linux. You can download the latest version of SSMS from here: https://msdn.microsoft.com/en-us/library/mt238290.aspx and follow the SSMS tutorial here: https://msdn.microsoft.com/en-us/library/bb934498.aspx

Note that we're still hard at work to provide a great user experience when you use SSMS with SQL Server on Linux. As such, you may receive unexpected error messages fin SSMS and a number of features are not yet supported as documented here: https://www.yammer.com/bpdtechadvisors/#/files/59098321

Using SQL Server Data Tools (SSDT)

You can use SSDT running on your Windows computer to connect to and work with SQL Server running anywhere, including SQL Server on Linux. You can download the latest version of SSDT from here: https://msdn.microsoft.com/en-us/library/mt204009.aspx and learn how to create databases, packages and data models by following the tutorial here: https://msdn.microsoft.com/library/hh272702(v=vs.103).aspx

Connect with Open Source Frameworks

Walk through the tutorials on Connecting with Open Source Drivers to connect to your SQL Server instance with code written in Node.js, Python, or Ruby on a Linux client. In these examples, update the connection string to point to your AdventureWorks database on SQL Server on Linux instead of using Azure SQL Database.

Using the MSSQL extension (prototype) for Visual Studio Code

The MSSQL extension is a proof of concept that adds MSSQL support to Visual Studio Code and lets you:

- Connect to SQL Server running on-premises or in the cloud, Azure SQL Database and Azure SQL Data Warehouse
- Type T-SQL statements, scripts and queries in the Visual Studio Code editor window
- Run T-SQL scripts to see results in a stylized HTML grid alongside the T-SQL editor
- Easily view and switch between multiple result sets

First, go here to download and install Visual Studio Code for your platform. Then install the MSSQL extension from the VS Code Marketplace, or execute "ext install vscode-mssql" directly in the VS Code Quick Open window. Make sure to enable the extension after installing it.

Using the extension

1. *Add connections:* Open VS Code User settings (`File->Preferences->User Settings`) or Workspace settings (`File->Preferences->Workspace Settings`). This will open the settings.json file. If you start typing "mssql" inside the json object, VS Code will autocomplete with suggestions – select the "connections" option and complete in the template with your server connection parameters.
2. *Change language mode to SQL:* Open a `.sql` file or press `Ctrl+K M` (`Cmd+K M` on Mac) switch the language mode of the active editor to SQL. This will provide basic IntelliSense functionality when typing SQL commands.
3. *Connect to a database:* Press `F1` to open the command palette, type `mssql` then click `Connect to a database` and follow the prompts.
4. *Use the T-SQL editor:* Type T-SQL statements in the editor. Type the word `sql` to see a list of code snippets you can tweak & reuse.
5. *Run T-SQL statements:* Select some T-SQL statements in the editor and press `Ctrl+Shift+e` (`Cmd+Shift+e` on Mac) to execute them and display results. The entire contents of the editor are executed if there is no selection. The results will be shown in a panel to the right of the code editor window.

Head over to Github for the extension source code, advanced options, bug tracking, and feature requests.

Using SqlPackage to create and deploy SQL Server databases and DACPAC packages

SQL Package is a command-line utility that helps for the extraction and deployment of database snapshots and other data migration tasks. This tool makes use of the DACPAC format, a self-contained unit of SQL Server deployment which allows the migration of data in a portable package.

SqlPackage Location

Sqlpackage can be found in an executable file that is in the following location:

```
/opt/mssql/bin/sqlpackage
```

This file can be executed using the following pattern:

```
./sqlpackage /Action:{action} {parameters}
```

```
./sqlpackage /Action:Extract /SourceServerName:localhost
/SourceDatabaseName:your_database /TargetFile:”/path/to/your/file.dacpac”
/SourceUser:”sa”
```

Use the following command to extract a DACPAC file from an existing database with SqlPackage

Use the following command to extract a DACPAC file from an existing database with SqlPackage:

```
./sqlpackage /Action:Extract /SourceServerName:"localhost/your_server"
/SourceDatabaseName:"your_database" /SourceUser:"your_username"
/SourcePassword:"your_password"
/TargetFile:"/absolute/path/to/your/target/file.dacpac"
```

Using the Extract action, you can provide the SourceServerName, SourceDatabaseName, SourceUser and SourcePassword to connect and extract the DACPAC file to a local TargetFile. Note: The "/p:Storage=Memory" option is the only one supported at this moment.

Extract a DACPAC file from an existing database into another database

Use the following command to extract a DACPAC file from an existing database and deploy it to another database server.

```
./sqlpackage /Action:Extract /SourceServerName:"localhost/your_server"
/SourceDatabaseName:"your_database" /SourceUser:"your_username"
/SourcePassword:"your_password" /TargetServerName:"target_server"
/TargetDatabaseName:"target_database" /TargetUser:"target_username"
/TargetPassword:"target_password"
```

Using the Extract action, you can provide the SourceServerName, SourceDatabaseName, SourceUser and SourcePassword to connect and extract the DACPAC file. Afterwards, SqlPackage will connect to a TargetServerName using the TargetDatabaseName, TargetUser, TargetPassword. Note: The "/p:Storage=Memory" option is the only one supported at this moment.

Using SMB to Create a Shared Backups Directory in Linux

You can use Samba to create a shared folder on the Linux. This allows the files to be shared on the network through the SMB protocol. This will also allow Windows hosts to get access remote to the files using File Explorer.

Setting up Samba on Ubuntu

Follow these steps to install Samba in your Ubuntu machine:

```
$ sudo apt-get update
```

```
$ sudo apt-get install samba
```

After installing Samba, you will need to add a local user to Samba and assign a new password to it:

```
$ sudo smbpasswd -a <username>
```

You are going to use this account when accessing the shared folder from a remote location.

Setting Up the Backups Directory for SQL Server

For this example, we will create our SQL Server backups at the */var/opt/mssql/backups* and mirror this directory in the */tmp/mssql/backups* directory. Create the directories using the following commands:

```
$ sudo mkdir /var/opt/mssql/backups /tmp/mssql /tmp/mssql/backups
```

Add the */tmp/mssql/backups* directory to the Samba configuration file, and adding the user that was created in the previous section, by editing the */etc/samba/smb.conf* file and adding the following lines at the end:

```
[mssql_backups]
path = /tmp/mssql/backups
valid users = <username>, …
read only = no
```

After changing the configuration file, make sure to restart the *smbd* service:

```
$ sudo service smbd restart
```

This will make the */tmp/mssql/backups* directory accessible for a SMB client, including Windows File Explorer. You can access this location by typing \\<your_linux_host_IP>\mssql_backups on a File Explorer window.

Backup a Database from SQL Server on Linux

Create a backup of your database using the following T-SQL statement:

```
BACKUP DATABASE your_database TO DISK = 'C:\backups\your_database.bak'
```

This will create a .bak file in the ***/var/opt/mssql/backups*** directory. Since this directory is in a location that can't be accessed as a non-root user, you will need to copy or sync this to your mapped backups directory. Use any of the following commands to do so:

```
# Using the cp command.
$ sudo cp /var/opt/mssql/backups/* /tmp/mssql/backups/
$ sudo chown mssql /tmp/mssql/backups/*
$ sudo chgrp mssql /tmp/mssql/backups/*

# Using the rsync command.
$ rsync -azvv /var/opt/mssql/backups/ /tmp/mssql/backups/
```

After these files have been copied, you will now be able to access them from a Windows File Browser or a smbclient using the following location: \\<your_linux_host_IP>\mssql_backups

Data Management in Docker Containers

Mapping a Data Volume from a Docker Container

When running a Docker container, it is possible to specify a volume that will map a directory within the container to a destination in the host that is running Docker. This can be done to persist the data that a Docker container running SQL Server on Linux has generated. Otherwise, the data would be lost every time a SQL Server container gets executed.

In order to map a volume, you need to specify the "-v" parameter for the Docker run command:

```
# Command parameters
$ docker run -v <host-destination>:<container-source> mssql-server

#Example
$ docker run -v /tmp/mssql/:/var/opt/mssql/ mssql-server
```

This will copy the contents of **/var/opt/mssql/** from the container into **/tmp/mssql/** in the host. This directory mapping will include both the data and log directories from SQL Server on Linux in Docker. If these directories are mapped in the host, the data from the container will be persisted in every run of the containers.

Mapping a Data Volume from a Docker Container into a Samba Shared Folder

After configuring a shared directory following the steps outlined above, this directory can be mapped to the Docker container, so that the data files can be shared from the container.

Create the shared folder and add it to the configuration file under */etc/samba/smb.conf*:

```
[mssql_container_data]
path = /tmp/mssql/data/
valid users = <username>, …
read only = no
```

Map the shared folder to the container data location on the "docker run" command:

```
# Command parameters
```

```
$ docker run -v /tmp/mssql/:/var/opt/mssql/ [...] mssql-server
```

This will make the data directory available from a SMB client under the path specified above.
\\<your_linux_host_IP>\mssql_container_data