

Table of Contents

[About SQL Server on Linux](#)

[Overview](#)

[Release notes](#)

[What's new?](#)

[New and updated articles](#)

[Editions and supported features](#)

[FAQ](#)

[Quickstarts](#)

[Install & Connect - Red Hat](#)

[Install & Connect - SUSE](#)

[Install & Connect - Ubuntu](#)

[Run & Connect - Docker](#)

[Provision a SQL VM in Azure](#)

[Run & Connect - Cloud](#)

[Tutorials](#)

[1_Migrate from Windows](#)

[2_Migrate from Oracle](#)

[3_Migrate to Docker](#)

[4_Create a job](#)

[5_Setup AD Authentication](#)

[6_Create failover cluster instance](#)

[iSCSI](#)

[NFS](#)

[SMB](#)

[7_Deploy a Pacemaker cluster](#)

[8_Create and configure availability groups](#)

[9_Configure in Kubernetes for high availability](#)

[Concepts](#)

[Install](#)

- [Install SQL Server](#)
- [Install SQL Server tools](#)
- [Install SQL Server Agent](#)
- [Install SQL Server Full-Text Search](#)
- [Install SQL Server Integration Services](#)
- [Configure repositories](#)
- [Configure](#)
 - [Configure with mssql-conf](#)
 - [Environment variables](#)
 - [Configure Docker containers](#)
 - [Customer Feedback](#)
- [Develop](#)
 - [Connectivity libraries](#)
 - [Use Visual Studio Code](#)
 - [Use SSMS](#)
 - [Use SSDT](#)
- [Manage](#)
 - [Use SSMS to manage](#)
 - [Use PowerShell to manage](#)
 - [Use log shipping](#)
 - [Use DB Mail and email alerts](#)
 - [Configure multiple subnets for availability](#)
- [Migrate](#)
 - [Export and import a BACPAC from Windows](#)
 - [Migrate with SQL Server Migration Assistant](#)
 - [Bulk copy with bcp](#)
 - [Extract, transform, load](#)
 - [Limitations and known issues](#)
 - [Configure SSIS](#)
 - [Schedule SSIS packages](#)
 - [Configure business continuity](#)
 - [Availability basics](#)

- [Backup and restore](#)
- [Failover cluster instance](#)
- [Availability groups](#)
- [Security](#)
 - [Get started with security features](#)
 - [Active Directory authentication](#)
 - [Encrypting connections](#)
- [Performance](#)
 - [Best practices](#)
 - [Get started with performance features](#)
- [Samples](#)
 - [Unattended install](#)
 - [Red Hat Enterprise Linux \(RHEL\)](#)
 - [SUSE Linux Enterprise Server \(SLES\)](#)
 - [Ubuntu](#)
- [Resources](#)
 - [Troubleshoot](#)
 - [SQL Server Documentation](#)
- [Partners](#)
 - [Monitoring](#)
 - [High availability and disaster recovery](#)
 - [Management](#)
 - [Development](#)
- [DBA Stack Exchange](#)
- [Stack Overflow](#)
- [MSDN Forums](#)
- [Submit feedback](#)
- [Reddit](#)

SQL Server on Linux

2/23/2018 • 1 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

SQL Server 2017 now runs on Linux. It's the same SQL Server database engine, with many similar features and services regardless of your operating system.

Install

To get started, install SQL Server on Linux using one of the following quickstarts:

- [Install on Red Hat Enterprise Linux](#)
- [Install on SUSE Linux Enterprise Server](#)
- [Install on Ubuntu](#)
- [Run on Docker](#)
- [Provision a SQL VM in Azure](#)

NOTE

Docker itself runs on multiple platforms, which means that you can run the Docker image on Linux, Mac, and Windows.

Connect

After installation, connect to the SQL Server instance on your Linux machine. You can connect locally or remotely and with a variety of tools and drivers. The quickstarts demonstrate how to use the `sqlcmd` command-line tool.

Other tools include the following:

TOOL	TUTORIAL
Visual Studio Code (VS Code)	Use VS Code with SQL Server on Linux
SQL Server Management Studio (SSMS)	Use SSMS on Windows to connect to SQL Server on Linux
SQL Server Data Tools (SSDT)	Use SSDT with SQL Server on Linux

Explore

SQL Server 2017 has the same underlying database engine on all supported platforms, including Linux. So many existing features and capabilities operate the same way on Linux. This area of the documentation exposes some of these features from a Linux perspective. It also calls out areas that have unique requirements on Linux.

If you are already familiar with SQL Server, review the [Release notes](#) for general guidelines and known issues for this release. Then look at [what's new for SQL Server on Linux](#) as well as [what's new for SQL Server 2017 overall](#).

TIP

For answers to frequently asked questions, see the [SQL Server on Linux FAQ](#).

ⓘ Get Help

- [UserVoice - Suggestion to improve SQL Server?](#)
- [Stack Overflow \(tag sql-server\) - ask SQL development questions](#)
- [Setup and Upgrade - MSDN Forum](#)
- [SQL Server Data Tools - MSDN forum](#)
- [Reddit - general discussion about SQL Server](#)
- [Microsoft SQL Server License Terms and Information](#)
- [Support options for business users](#)
- [Contact Microsoft](#)

Release notes for SQL Server 2017 on Linux

2/23/2018 • 12 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The following release notes apply to SQL Server 2017 running on Linux. This article is broken into sections for each release. The GA release has detailed supportability and known issues listed. Each Cumulative Update (CU) release has a link to a support article describing the CU changes as well as links to the Linux package downloads.

Supported platforms

PLATFORM	FILE SYSTEM	INSTALLATION GUIDE
Red Hat Enterprise Linux 7.3 or 7.4 Workstation, Server, and Desktop	XFS or EXT4	Installation guide
SUSE Enterprise Linux Server v12 SP2	XFS or EXT4	Installation guide
Ubuntu 16.04LTS	XFS or EXT4	Installation guide
Docker Engine 1.8+ on Windows, Mac, or Linux	N/A	Installation guide

TIP

For more information, review the [system requirements](#) for SQL Server on Linux. For the latest support policy for SQL Server 2017, see the [Technical support policy for Microsoft SQL Server](#).

Tools

Most existing client tools that target SQL Server can seamlessly target SQL Server running on Linux. Some tools might have a specific version requirement to work well with Linux. For a full list of SQL Server tools, see [SQL Tools and Utilities for SQL Server](#).

Release history

The following table lists the release history for SQL Server 2017.

RELEASE	VERSION	RELEASE DATE
CU4	14.0.3022.28	2-2018
CU3	14.0.3015.40	1-2018
CU2	14.0.3008.27	11-2017
CU1	14.0.3006.16	10-2017

RELEASE	VERSION	RELEASE DATE
GA	14.0.1000.169	10-2017

How to install cumulative updates

If you have configured the Cumulative Update repository, then you will get the latest cumulative update of SQL Server packages when you perform new installations. The Cumulative Update repository is the default for all package installation articles for SQL Server on Linux. For more information about repository configuration, see [Configure repositories for SQL Server on Linux](#).

If you are updating existing SQL Server packages, run the appropriate update command for each package to get the latest cumulative update. For specific update instructions for each package, see the following installation guides:

- [Install SQL Server package](#)
- [Install Full-text Search package](#)
- [Install SQL Server Integration Services](#)
- [Enable SQL Server Agent](#)

CU4 (February 2018)

This is the Cumulative Update 4 (CU4) release of SQL Server 2017. The SQL Server engine version for this release is 14.0.3022.28. For information about the fixes and improvements in this release, see <https://support.microsoft.com/en-us/help/4056498>.

Package details

For manual or offline package installations, you can download the RPM and Debian packages with the information in the following table:

NOTE

As of CU4, SQL Server Agent is no longer installed as a separate package. It is installed with the Engine package and must be enabled to use.

PACKAGE	PACKAGE VERSION	DOWNLOADS
Red Hat RPM package	14.0.3022.28-2	Engine RPM package High Availability RPM package Full-text Search RPM package SSIS package
SLES RPM package	14.0.3022.28-2	mssql-server Engine RPM package High Availability RPM package Full-text Search RPM package
Ubuntu 16.04 Debian package	14.0.3022.28-2	Engine Debian package High Availability Debian package Full-text Search Debian package SSIS package

CU3 (January 2018)

This is the Cumulative Update 3 (CU3) release of SQL Server 2017. The SQL Server engine version for this release is 14.0.3015.40. For information about the fixes and improvements in this release, see <https://support.microsoft.com/en-us/help/4052987>.

Package details

For manual or offline package installations, you can download the RPM and Debian packages with the information in the following table:

PACKAGE	PACKAGE VERSION	DOWNLOADS
Red Hat RPM package	14.0.3015.40-1	Engine RPM package High Availability RPM package Full-text Search RPM package SQL Server Agent RPM package SSIS package
SLES RPM package	14.0.3015.40-1	mssql-server Engine RPM package High Availability RPM package Full-text Search RPM package SQL Server Agent RPM package
Ubuntu 16.04 Debian package	14.0.3015.40-1	Engine Debian package High Availability Debian package Full-text Search Debian package SQL Server Agent Debian package SSIS package

CU2 (November 2017)

This is the Cumulative Update 2 (CU2) release of SQL Server 2017. The SQL Server engine version for this release is 14.0.3008.27. For information about the fixes and improvements in this release, see <https://support.microsoft.com/help/4052574>.

Package details

For manual or offline package installations, you can download the RPM and Debian packages with the information in the following table:

PACKAGE	PACKAGE VERSION	DOWNLOADS
Red Hat RPM package	14.0.3008.27-1	Engine RPM package High Availability RPM package Full-text Search RPM package SQL Server Agent RPM package SSIS package
SLES RPM package	14.0.3008.27-1	mssql-server Engine RPM package High Availability RPM package Full-text Search RPM package SQL Server Agent RPM package
Ubuntu 16.04 Debian package	14.0.3008.27-1	Engine Debian package High Availability Debian package Full-text Search Debian package SQL Server Agent Debian package SSIS package

CU1 (October 2017)

This is the Cumulative Update 1 (CU1) release of SQL Server 2017. The SQL Server engine version for this release is 14.0.3006.16. For information about the fixes and improvements in this release, see <https://support.microsoft.com/help/KB4053439>.

Package details

For manual or offline package installations, you can download the RPM and Debian packages with the information in the following table:

PACKAGE	PACKAGE VERSION	DOWNLOADS
Red Hat RPM package	14.0.3006.16-3	Engine RPM package High Availability RPM package Full-text Search RPM package SQL Server Agent RPM package SSIS package
SLES RPM package	14.0.3006.16-3	mssql-server Engine RPM package High Availability RPM package Full-text Search RPM package SQL Server Agent RPM package
Ubuntu 16.04 Debian package	14.0.3006.16-3	Engine Debian package High Availability Debian package Full-text Search Debian package SQL Server Agent Debian package SSIS package

GA (October 2017)

This is the General Availability (GA) release of SQL Server 2017. The SQL Server engine version for this release is 14.0.1000.169.

Package details

Package details and download locations for the RPM and Debian packages are listed in the following table. Note that you do not need to download these packages directly if you use the steps in the following installation guides:

- [Install SQL Server package](#)
- [Install Full-text Search package](#)
- [Install SQL Server Agent package](#)
- [Install SQL Server Integration Services](#)

PACKAGE	PACKAGE VERSION	DOWNLOADS
Red Hat RPM package	14.0.1000.169-2	Engine RPM package High Availability RPM package Full-text Search RPM package SQL Server Agent RPM package SSIS package
SLES RPM package	14.0.1000.169-2	mssql-server Engine RPM package High Availability RPM package Full-text Search RPM package SQL Server Agent RPM package

PACKAGE	PACKAGE VERSION	DOWNLOADS
Ubuntu 16.04 Debian package	14.0.1000.169-2	Engine Debian package High Availability Debian package Full-text Search Debian package SQL Server Agent Debian package SSIS package

Unsupported features & services

The following features and services are not available on Linux at the time of the GA release. The support of these features will be increasingly enabled over time.

AREA	UNSUPPORTED FEATURE OR SERVICE
Database engine	Transactional replication
	Merge replication
	Stretch DB
	Polybase
	Distributed query with 3rd-party connections
	System extended stored procedures (XP_CMDSHELL, etc.)
	Filetable, FILESTREAM
	CLR assemblies with the EXTERNAL_ACCESS or UNSAFE permission set
	Buffer Pool Extension
SQL Server Agent	Subsystems: CmdExec, PowerShell, Queue Reader, SSIS, SSAS, SSRS
	Alerts
	Log Reader Agent
	Change Data Capture
High Availability	Managed Backup
	Database mirroring
Security	Extensible Key Management
	AD Authentication for Linked Servers
	AD Authentication for Availability Groups (AGs)

AREA	UNSUPPORTED FEATURE OR SERVICE
	3rd party AD tools (Centrify, Vintela, Powerbroker)
Services	SQL Server Browser
	SQL Server R services
	StreamInsight
	Analysis Services
	Reporting Services
	Data Quality Services
	Master Data Services
	Distributed Transaction Coordinator (DTC)

Known issues

The following sections describe known issues with the General Availability (GA) release of SQL Server 2017 on Linux.

General

- Upgrades to the GA release of SQL Server 2017 are supported only from CTP 2.1 or higher.
- The length of the hostname where SQL Server is installed needs to be 15 characters or less.
 - **Resolution:** Change the name in /etc/hostname to something 15 characters long or less.
- Manually setting the system time backwards in time will cause SQL Server to stop updating the internal system time within SQL Server.
 - **Resolution:** Restart SQL Server.
- Only single instance installations are supported.
 - **Resolution:** If you want to have more than one instance on a given host, consider using VMs or Docker containers.
- SQL Server Configuration Manager can't connect to SQL Server on Linux.
- The default language of the **sa** login is English.
 - **Resolution:** Change the language of the **sa** login with the **ALTER LOGIN** statement.

Databases

- The master database cannot be moved with the mssql-conf utility. Other system databases can be moved with mssql-conf.
- When restoring a database that was backed up on SQL Server on Windows, you must use the **WITH MOVE** clause in the Transact-SQL statement.
- Distributed transactions requiring the Microsoft Distributed Transaction Coordinator service are not supported on SQL Server running on Linux. SQL Server to SQL Server linked servers are supported unless they involve the DTC. For more information, see [Distributed transactions requiring the Microsoft Distributed Transaction Coordinator service are not supported on SQL Server running on Linux](#).

- Certain algorithms (cipher suites) for Transport Layer Security (TLS) do not work properly with SQL Server on Linux. This results in connection failures when attempting to connect to SQL Server, as well as problems establishing connections between replicas in high availability groups.
 - **Resolution:** Modify the **mssql.conf** configuration script for SQL Server on Linux to disable problematic cipher suites, by doing the following:

1. Add the following to `/var/opt/mssql/mssql.conf`.

```
[network]
tlsciphers= AES256-GCM-SHA384:AES128-GCM-SHA256:AES256-SHA256:AES128-SHA256:AES256-
SHA:AES128-SHA: !ECDHE-RSA-AES128-GCM-SHA256: !ECDHE-RSA-AES256-GCM-SHA384: !ECDHE-ECDSA-
AES256-GCM-SHA384: !ECDHE-ECDSA-AES128-GCM-SHA256: !ECDHE-ECDSA-AES256-SHA384: !ECDHE-
ECDSA-AES128-SHA256: !ECDHE-ECDSA-AES256-SHA: !ECDHE-ECDSA-AES128-SHA: !ECDHE-RSA-AES256-
SHA384: !ECDHE-RSA-AES128-SHA256: !ECDHE-RSA-AES256-SHA: !ECDHE-RSA-AES128-SHA: !DHE-RSA-
AES256-GCM-SHA384: !DHE-RSA-AES128-GCM-SHA256: !DHE-RSA-AES256-SHA: !DHE-RSA-AES128-
SHA: !DHE-DSS-AES256-SHA256: !DHE-DSS-AES128-SHA256: !DHE-DSS-AES256-SHA: !DHE-DSS-AES128-
SHA: !DHE-DSS-DES-CBC3-SHA: !NULL-SHA256: !NULL-SHA
```

NOTE

In the preceding code, `!` negates the expression. This tells OpenSSL to not use the following cipher suite.

2. Restart SQL Server with the following command.

```
sudo systemctl restart mssql-server
```

- SQL Server 2014 databases on Windows that use In-memory OLTP cannot be restored on SQL Server 2017 on Linux. To restore a SQL Server 2014 database that uses in-memory OLTP, first upgrade the databases to SQL Server 2016 or SQL Server 2017 on Windows before moving them to SQL Server on Linux via backup/restore or detach/attach.
- User permission **ADMINISTER BULK OPERATIONS** is not supported on Linux at this time.

Networking

Features that involve outbound TCP connections from the `sqlservr` process, such as linked servers or Availability Groups, might not work if both the following conditions are met:

1. The target server is specified as a hostname and not an IP address.
2. The source instance has IPv6 disabled in the kernel. To verify if your system has IPv6 enabled in the kernel, all the following tests must pass:

- `cat /proc/cmdline` will print the boot cmdline of the current kernel. The output must not contain `ipv6.disable=1`.
- The `/proc/sys/net/ipv6/` directory must exist.
- A C program that calls `socket(AF_INET6, SOCK_STREAM, IPPROTO_IP)` should succeed - the syscall must return an `fd != -1` and not fail with `EAFNOSUPPORT`.

The exact error depends on the feature. For linked servers, this manifests as a login timeout error. For Availability Groups, the `ALTER AVAILABILITY GROUP JOIN` DDL on the secondary will fail after 5 minutes with a download configuration timeout error.

To work around this issue, do one of the following:

1. Use IPs instead of hostnames to specify the target of the TCP connection.
2. Enable IPv6 in the kernel by removing `ipv6.disable=1` from the boot cmdline. The way to do this depends on the Linux distribution and the bootloader, such as grub. If you do want IPv6 to be disabled, you can still disable it by setting `net.ipv6.conf.all.disable_ipv6 = 1` in the `sysctl` configuration (eg `/etc/sysctl.conf`). This will still prevent the system's network adapter from getting an IPv6 address, but allow the sqlservr features to work.

Network File System (NFS)

If you use **Network File System (NFS)** remote shares in production, note the following support requirements:

- Use NFS version **4.2 or higher**. Older versions of NFS do not support required features, such as fallocate and sparse file creation, common to modern file systems.
- Locate only the **/var/opt/mssql** directories on the NFS mount. Other files, such as the SQL Server system binaries, are not supported.
- Ensure that NFS clients use the 'nolock' option when mounting the remote share.

Localization

- If your locale is not English (en_us) during setup, you must use UTF-8 encoding in your bash session/terminal. If you use ASCII encoding, you might see an error similar to the following:

```
UnicodeEncodeError: 'ascii' codec can't encode character u'\xf1' in position 8: ordinal not in range(128)
```

If you cannot use UTF-8 encoding, run setup using the MSSQL_LCID environment variable to specify your language choice.

```
sudo MSSQL_LCID=<LcidValue> /opt/mssql/bin/mssql-conf setup
```

- When running mssql-conf setup, and performing a non-English installation of SQL Server, incorrect extended characters are displayed after the localized text, "Configuring SQL Server...". Or, for non-Latin based installations, the sentence might be missing completely. The missing sentence should display the following localized string: "The licensing PID was successfully processed. The new edition is [<Name> edition]". This string is output for information purposes only, and the next SQL Server Cumulative Update will address this for all languages. This does not affect the successful installation of SQL Server in any way.

Full-Text Search

- Not all filters are available with this release, including filters for Office documents. For a list of supported filters, see [Install SQL Server Full-Text Search on Linux](#).

SQL Server Integration Services (SSIS)

- The **mssql-server-is** package is not supported on SUSE in this release. It is currently supported on Ubuntu and on Red Hat Enterprise Linux (RHEL).
- With SSIS on Linux CTP 2.1 Refresh and later, SSIS packages can use ODBC connections on Linux. This functionality has been tested with the SQL Server and the MySQL ODBC drivers, but is also expected to work with any Unicode ODBC driver that observes the ODBC specification. At design time, you can provide either a DSN or a connection string to connect to the ODBC data; you can also use Windows authentication. For more info, see the [blog post announcing ODBC support on Linux](#).
- The following features are not supported in this release when you run SSIS packages on Linux:
 - SSIS Catalog database
 - Scheduled package execution by SQL Agent

- Windows Authentication
- Third-party components
- Change Data Capture (CDC)
- SSIS Scale Out
- Azure Feature Pack for SSIS
- Hadoop and HDFS support
- Microsoft Connector for SAP BW

For a list of built-in SSIS components that are not currently supported, or that are supported with limitations, see [Limitations and known issues for SSIS on Linux](#).

For more info about SSIS on Linux, see the following articles:

- [Blog post announcing SSIS support for Linux](#).
- [Install SQL Server Integration Services \(SSIS\) on Linux](#)
- [Extract, transform, and load data on Linux with SSIS](#)

SQL Server Management Studio (SSMS)

The following limitations apply to SSMS on Windows connected to SQL Server on Linux.

- Maintenance plans are not supported.
- Management Data Warehouse (MDW) and the data collector in SSMS are not supported.
- SSMS UI components that have Windows Authentication or Windows event log options do not work with Linux. You can still use these features with other options, such as SQL logins.
- Number of log files to retain cannot be modified.

Next steps

To get started, see the following quickstarts:

- [Install on Red Hat Enterprise Linux](#)
- [Install on SUSE Linux Enterprise Server](#)
- [Install on Ubuntu](#)
- [Run on Docker](#)
- [Provision a SQL VM in Azure](#)
- [Run & Connect - Cloud](#)

For answers to frequently asked questions, see the [SQL Server on Linux FAQ](#).

What's new for SQL Server 2017 on Linux

2/23/2018 • 1 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse

Parallel Data Warehouse

This article describes the major features and services available for SQL Server 2017 running on Linux.

NOTE

In addition to these capabilities in this article, cumulative updates are released at regular intervals after the GA release. These cumulative updates provide many improvements and fixes. For information about the latest CU release, see <http://aka.ms/sql2017cu>. For package downloads and known issues, see the [Release notes](#).

SQL Server Database Engine

- Enabled the core SQL Server Database Engine capabilities.
- Support for native Linux paths.
- IPV6 support.
- Support for database files on NFS.
- Enabled [Transparent Layer Security \(TLS\)](#) encryption.
- Enabled [Active Directory Authentication](#).
- [Availability Groups functionality](#) for high availability.
- [Full-text Search](#) support.

SQL Server Agent

- Enabled [SQL Server Agent](#) support for the following tasks:
 - [Transact-SQL jobs](#)
 - [DB mail](#)
 - [Log shipping](#)

SQL Server Integration Services (SSIS)

- Ability to run SSIS packages on Linux. For more information, see [Configure SQL Server Integration Services on Linux with ssis-conf](#).

Other improvements

- Command-line configuration tool, [mssql-conf](#).
- Unattended installation support with [environment variables](#).
- Cross-platform [Visual Studio Code mssql-server extension](#).
- Cross-platform script generator, [mssql-scripter](#).
- Cross-platform Dynamic Management View (DMV) monitor, [DBFS tool](#).

Next steps

To install SQL Server on Linux, use one of the following tutorials:

- [Install on Red Hat Enterprise Linux](#)
- [Install on SUSE Linux Enterprise Server](#)
- [Install on Ubuntu](#)
- [Run on Docker](#)
- [Provision a SQL VM in Azure](#)

To see other improvements introduced in SQL Server 2017, see [What's New in SQL Server 2017](#).

TIP

For answers to frequently asked questions, see the [SQL Server on Linux FAQ](#).

ⓘ Get Help

- [UserVoice - Suggestion to improve SQL Server?](#)
- [Stack Overflow \(tag sql-server\) - ask SQL development questions](#)
- [Setup and Upgrade - MSDN Forum](#)
- [SQL Server Data Tools - MSDN forum](#)
- [Reddit - general discussion about SQL Server](#)
- [Microsoft SQL Server License Terms and Information](#)
- [Support options for business users](#)
- [Contact Microsoft](#)

New and Recently Updated: SQL Server on Linux docs

2/14/2018 • 6 min to read • [Edit Online](#)

Nearly every day Microsoft updates some of its existing articles on its [Docs.Microsoft.com](#) documentation website. This article displays excerpts from recently updated articles. Links to new articles might also be listed.

This article is generated by a program that is rerun periodically. Occasionally an excerpt can appear with imperfect formatting, or as markdown from the source article. Images are never displayed here.

Recent updates are reported for the following date range and subject:

- *Date range of updates:* **2017-12-03** -to- **2018-02-03**
- *Subject area:* **Microsoft SQL Server on Linux.**

New Articles Created Recently

The following links jump to new articles that have been added recently.

1. [Configure multiple-subnet Always On Availability Groups and failover cluster instances](#)
2. [Create and configure an availability group for SQL Server on Linux](#)
3. [Deploy a Pacemaker cluster for SQL Server on Linux](#)
4. [SQL Server on Linux Frequently Asked Questions \(FAQ\)](#)
5. [SQL Server availability basics for Linux deployments](#)
6. [Configure a SQL Server container in Kubernetes for high availability](#)

Updated Articles with Excerpts

This section displays the excerpts of updates gathered from articles that have recently experienced a large update.

The excerpts displayed here appear separated from their proper semantic context. Also, sometimes an excerpt is separated from important markdown syntax that surrounds it in the actual article. Therefore these excerpts are for general guidance only. The excerpts only enable you to know whether your interests warrant taking the time to click and visit the actual article.

For these and other reasons, do not copy code from these excerpts, and do not take as exact truth any text excerpt. Instead, visit the actual article.

Compact List of Articles Updated Recently

This compact list provides links to all the updated articles that are listed in the Excerpts section.

1. [Always On Availability Groups on Linux](#)
2. [Extract, transform, and load data on Linux with SSIS](#)

1. Always On Availability Groups on Linux

Updated: 2018-01-31 ([Next](#))

Automatic failover of an AG is possible when the following conditions are met:

- The primary and the secondary replica are set to synchronous data movement.
- The secondary has a state of synchronized (not synchronizing), meaning the two are at the same data point.
- The cluster type is set to External. Automatic failover is not possible with a cluster type of None.
- The `sequence_number` of the secondary replica to become the primary has the highest sequence number – in other words, the secondary replica's `sequence_number` matches the one from the original primary replica.

If these conditions are met and the server hosting the primary replica fails, the AG will change ownership to a synchronous replica. The behavior for synchronous replicas (of which there can be three total: one primary and two secondary replicas) can further be controlled by `required_synchronized_secondaries_to_commit`. This works with AGs on both Windows and Linux, but is configured completely differently. On Linux, the value is configured automatically by the cluster on the AG resource itself.

Configuration-only replica and quorum

Also new in SQL Server 2017 as of CU1 is a configuration-only replica. Because Pacemaker is different than a WSFC, especially when it comes to quorum and requiring STONITH, having just a two-node configuration will not work when it comes to an AG. For an FCI, the quorum mechanisms provided by Pacemaker can be fine, because all FCI failover arbitration happens at the cluster layer. For an AG, arbitration under Linux happens in SQL Server, where all the metadata is stored. This is where the configuration-only replica comes into play.

Without anything else, a third node and at least one synchronized replica would be required. This would not work for SQL Server Standard, since it can only have two replicas participating in an AG. The configuration-only replica stores the AG configuration in the master database, same as the other replicas in the AG configuration. The configuration-only replica does not have the user databases participating in the AG. The configuration data is sent synchronously from the primary. This configuration data is then used during failovers, whether they are automatic or manual.

2. Extract, transform, and load data on Linux with SSIS

Updated: 2018-01-31 ([Previous](#))

```
...
SSIS_PACKAGE_DECRYPT=test /opt/ssis/bin/dtexec /f package.dtsx
...
```

1. Specify the `/de[crypt]` option to enter the password interactively, as shown in the following example:

```
/opt/ssis/bin/dtexec /f package.dtsx /de
Enter decryption password:
```

2. Specify the `/de` option to provide the password on the command line, as shown in the following example.

This method is not recommended because it stores the decryption password with the command in the command history.

```
opt/ssis/bin/dtexec /f package.dtsx /de test

Warning: Using /De[crypt] <password> may store decryption password in command history.

You can use /De[crypt] instead to enter interactive mode,
or use environment variable SSIS_PACKAGE_DECRYPT to set decryption password.
```

Design packages

Connect to ODBC data sources. With SSIS on Linux CTP 2.1 Refresh and later, SSIS packages can use ODBC connections on Linux. This functionality has been tested with the SQL Server and the MySQL ODBC drivers, but is also expected to work with any Unicode ODBC driver that observes the ODBC specification. At design time, you can provide either a DSN or a connection string to connect to the ODBC data; you can also use Windows authentication. For more info, see the [blog post announcing ODBC support on Linux](#).

Similar articles about new or updated articles

This section lists very similar articles for recently updated articles in other subject areas, within our public GitHub.com repository: [MicrosoftDocs/sql-docs](#).

Subject areas that *do* have new or recently updated articles

- New + Updated (1+3): [Advanced Analytics for SQL docs](#)
- New + Updated (0+1): [Analytics Platform System for SQL docs](#)
- New + Updated (0+1): [Connect to SQL docs](#)
- New + Updated (0+1): [Database Engine for SQL docs](#)
- New + Updated (12+1): [Integration Services for SQL docs](#)
- New + Updated (6+2): [Linux for SQL docs](#)
- New + Updated (15+0): [PowerShell for SQL docs](#)
- New + Updated (2+9): [Relational Databases for SQL docs](#)
- New + Updated (1+0): [Reporting Services for SQL docs](#)
- New + Updated (1+1): [SQL Operations Studio docs](#)
- New + Updated (1+1): [Microsoft SQL Server docs](#)
- New + Updated (0+1): [SQL Server Data Tools \(SSDT\) docs](#)
- New + Updated (1+2): [SQL Server Management Studio \(SSMS\) docs](#)
- New + Updated (0+2): [Transact-SQL docs](#)

Subject areas that *do not* have any new or recently updated articles

- New + Updated (0+0): [Data Migration Assistant \(DMA\) for SQL docs](#)
- New + Updated (0+0): [ActiveX Data Objects \(ADO\) for SQL docs](#)
- New + Updated (0+0): [Analysis Services for SQL docs](#)
- New + Updated (0+0): [Data Quality Services for SQL docs](#)
- New + Updated (0+0): [Data Mining Extensions \(DMX\) for SQL docs](#)
- New + Updated (0+0): [Master Data Services \(MDS\) for SQL docs](#)
- New + Updated (0+0): [Multidimensional Expressions \(MDX\) for SQL docs](#)
- New + Updated (0+0): [ODBC \(Open Database Connectivity\) for SQL docs](#)
- New + Updated (0+0): [Samples for SQL docs](#)
- New + Updated (0+0): [SQL Server Migration Assistant \(SSMA\) docs](#)
- New + Updated (0+0): [Tools for SQL docs](#)

- New + Updated (0+0): **XQuery for SQL** docs

Editions and supported features of SQL Server 2017 on Linux

2/14/2018 • 8 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article provides details of features supported by the various editions of SQL Server 2017 on Linux. For editions and supported features of SQL Server on Windows, see [SQL Server 2017 - Windows](#).

Installation requirements vary based on your application needs. The different editions of SQL Server accommodate the unique performance, runtime, and price requirements of organizations and individuals. The SQL Server components that you install also depend on your specific requirements. The following sections help you understand how to make the best choice among the editions and components available in SQL Server.

For the latest release notes and what's new information, see the following:

- [SQL Server on Linux release notes](#)
- [What's new in SQL Server on Linux](#)

For a list of SQL Server features not available on Linux, see [Unsupported features and services](#).

Try SQL Server!

[Download SQL Server 2017](#)

SQL Server editions

The following table describes the editions of SQL Server.

SQL SERVER EDITION	DEFINITION
Enterprise	The premium offering, SQL Server Enterprise edition delivers comprehensive high-end datacenter capabilities with blazing-fast performance enabling high service levels for mission-critical workloads.
Standard	SQL Server Standard edition delivers basic data management for departments and small organizations to run their applications and supports common development tools for on-premise and cloud — enabling effective database management with minimal IT resources.
Web	SQL Server Web edition is a low total-cost-of-ownership option for Web hosts and Web VAPs to provide scalability, affordability, and manageability capabilities for small to large scale Web properties.
Developer	SQL Server Developer edition lets developers build any kind of application on top of SQL Server. It includes all the functionality of Enterprise edition, but is licensed for use as a development and test system, not as a production server. SQL Server Developer is an ideal choice for people who build and test applications.

SQL SERVER EDITION	DEFINITION
Express edition	Express edition is the entry-level, free database and is ideal for learning and building desktop and small server data-driven applications. It is the best choice for independent software vendors, developers, and hobbyists building client applications. If you need more advanced database features, SQL Server Express can be seamlessly upgraded to other higher end versions of SQL Server.

Using SQL Server with client/server applications

You can install just the SQL Server client components on a computer that is running client/server applications that connect directly to an instance of SQL Server. A client components installation is also a good option if you administer an instance of SQL Server on a database server, or if you plan to develop SQL Server applications.

SQL Server components

SQL Server 2017 on Linux supports the SQL Server database engine. The following table describes the features in the database engine.

SERVER COMPONENTS	DESCRIPTION
SQL Server Database Engine	SQL Server Database Engine includes the Database Engine, the core service for storing, processing, and securing data, replication, full-text search, tools for managing relational and XML data, and in database analytics integration.

Developer, Enterprise Core, and Evaluation editions

For features supported by Developer, Enterprise Core, and Evaluation editions, see features listed for the SQL Server Enterprise edition in the following tables.

The Developer edition continues to support only 1 client for [SQL Server Distributed Replay](#).

Scale limits

FEATURE	ENTERPRISE	STANDARD	WEB	EXPRESS
Maximum compute capacity used by a single instance - SQL Server Database Engine ¹	Operating system maximum	Limited to lesser of 4 sockets or 24 cores	Limited to lesser of 4 sockets or 16 cores	Limited to lesser of 1 socket or 4 cores
Maximum compute capacity used by a single instance - Analysis Services or Reporting Services	Operating system maximum	Limited to lesser of 4 sockets or 24 cores	Limited to lesser of 4 sockets or 16 cores	Limited to lesser of 1 socket or 4 cores
Maximum memory for buffer pool per instance of SQL Server Database Engine	Operating System Maximum	128 GB	64 GB	1410 MB

FEATURE	ENTERPRISE	STANDARD	WEB	EXPRESS
Maximum memory for Columnstore segment cache per instance of SQL Server Database Engine	Unlimited memory	32 GB	16 GB	352 MB
Maximum memory-optimized data size per database in SQL Server Database Engine	Unlimited memory	32 GB	16 GB	352 MB
Maximum relational database size	524 PB	524 PB	524 PB	10 GB

¹ Enterprise edition with Server + Client Access License (CAL) based licensing (not available for new agreements) is limited to a maximum of 20 cores per SQL Server instance. There are no limits under the Core-based Server Licensing model. For more information, see [Compute capacity limits by edition of SQL Server](#).

RDBMS high availability

FEATURE	ENTERPRISE	STANDARD	WEB	EXPRESS
Log shipping	Yes	Yes	Yes	No
Backup compression	Yes	Yes	No	No
Database snapshot	Yes	No	No	No
Always On failover cluster instance ¹	Yes	Yes	No	No
Always On availability groups ²	Yes	No	No	No
Basic availability groups ³	No	Yes	No	No
Minimum replica commit availability group	Yes	Yes	No	No
Clusterless availability group	Yes	Yes	No	No
Online page and file restore	Yes	No	No	No
Online indexing	Yes	No	No	No
Resumable online index rebuilds	Yes	No	No	No

FEATURE	ENTERPRISE	STANDARD	WEB	EXPRESS
Online schema change	Yes	No	No	No
Fast recovery	Yes	No	No	No
Mirrored backups	Yes	No	No	No
Hot add memory and CPU	Yes	No	No	No
Encrypted backup	Yes	Yes	No	No
Hybrid backup to Windows Azure (backup to URL)	Yes	Yes	No	No

¹ On Enterprise edition, the number of nodes is the operating system maximum. On Standard edition there is support for two nodes.

² On Enterprise edition, provides support for up to 8 secondary replicas - including 2 synchronous secondary replicas.

³ Standard edition supports basic availability groups. A basic availability group supports two replicas, with one database. For more information about basic availability groups, see [Basic Availability Groups](#).

RDBMS scalability and performance

FEATURE	ENTERPRISE	STANDARD	WEB	EXPRESS
Columnstore ¹	Yes	Yes	Yes	Yes
Large object binaries in clustered columnstore indexes	Yes	Yes	Yes	Yes
Online non-clustered columnstore index rebuild	Yes	No	No	No
In-Memory OLTP ¹	Yes	Yes	Yes	Yes
Persistent Main Memory	Yes	Yes	Yes	Yes
Table and index partitioning	Yes	Yes	Yes	Yes
Data compression	Yes	Yes	Yes	Yes
Resource Governor	Yes	No	No	No
Partitioned Table Parallelism	Yes	No	No	No

FEATURE	ENTERPRISE	STANDARD	WEB	EXPRESS
NUMA Aware and Large Page Memory and Buffer Array Allocation	Yes	No	No	No
IO Resource Governance	Yes	No	No	No
Delayed Durability	Yes	Yes	Yes	Yes
Automatic Tuning	Yes	No	No	No
Batch Mode Adaptive Joins	Yes	No	No	No
Batch Mode Memory Grant Feedback	Yes	No	No	No
Interleaved Execution for Multi-Statement Table Valued Functions	Yes	Yes	Yes	Yes
Bulk insert improvements	Yes	Yes	Yes	Yes

¹ In-Memory OLTP data size and Columnstore segment cache are limited to the amount of memory specified by edition in the Scale Limits section. The max degrees of parallelism is limited. The degrees of process parallelism (DOP) for an index build is limited to 2 DOP for the Standard edition and 1 DOP for the Web and Express editions. This refers to columnstore indexes created over disk-based tables and memory-optimized tables.

RDBMS security

FEATURE	ENTERPRISE	STANDARD	WEB	EXPRESS
Row-level security	Yes	Yes	Yes	Yes
Always Encrypted	Yes	Yes	Yes	Yes
Dynamic data masking	Yes	Yes	Yes	Yes
Basic auditing	Yes	Yes	Yes	Yes
Fine grained auditing	Yes	Yes	Yes	Yes
Transparent database encryption	Yes	No	No	No
User-defined roles	Yes	Yes	Yes	Yes
Contained databases	Yes	Yes	Yes	Yes

FEATURE	ENTERPRISE	STANDARD	WEB	EXPRESS
Encryption for backups	Yes	Yes	No	No

RDBMS manageability

FEATURE	ENTERPRISE	STANDARD	WEB	EXPRESS
Dedicated admin connection	Yes	Yes	Yes	Yes with trace flag
PowerShell scripting support	Yes	Yes	Yes	Yes
Support for data-tier application component operations - extract, deploy, upgrade, delete	Yes	Yes	Yes	Yes
Policy automation (check on schedule and change)	Yes	Yes	Yes	No
Performance data collector	Yes	Yes	Yes	No
Standard performance reports	Yes	Yes	Yes	No
Plan guides and plan freezing for plan guides	Yes	Yes	Yes	No
Direct query of indexed views (using NOEXPAND hint)	Yes	Yes	Yes	Yes
Automatic indexed views maintenance	Yes	Yes	Yes	No
Distributed partitioned views	Yes	No	No	No
Parallel indexed operations	Yes	No	No	No
Automatic use of indexed view by query optimizer	Yes	No	No	No
Parallel consistency check	Yes	No	No	No

FEATURE	ENTERPRISE	STANDARD	WEB	EXPRESS
SQL Server Utility Control Point	Yes	No	No	No

Programmability

FEATURE	ENTERPRISE	STANDARD	WEB	EXPRESS
JSON	Yes	Yes	Yes	Yes
Query Store	Yes	Yes	Yes	Yes
Temporal	Yes	Yes	Yes	Yes
Native XML support	Yes	Yes	Yes	Yes
XML indexing	Yes	Yes	Yes	Yes
MERGE & UPSERT capabilities	Yes	Yes	Yes	Yes
Date and Time datatypes	Yes	Yes	Yes	Yes
Internationalization support	Yes	Yes	Yes	Yes
Full-text and semantic search	Yes	Yes	Yes	Yes
Specification of language in query	Yes	Yes	Yes	Yes
Service Broker (messaging)	Yes	Yes	No (Client only)	No (Client only)
Transact-SQL endpoints	Yes	Yes	Yes	No
Graph	Yes	Yes	Yes	Yes

¹ Scale out with multiple compute nodes requires a head node.

Integration Services

For info about the Integration Services (SSIS) features supported by the editions of SQL Server, see [Integration Services features supported by the editions of SQL Server](#).

Spatial and location services

FEATURE NAME	ENTERPRISE	STANDARD	WEB	EXPRESS
Spatial indexes	Yes	Yes	Yes	Yes
Planar and geodetic datatypes	Yes	Yes	Yes	Yes
Advanced spatial libraries	Yes	Yes	Yes	Yes
Import/export of industry-standard spatial data formats	Yes	Yes	Yes	Yes

Next steps

[Editions and supported features for SQL Server 2017 - Windows](#)

[Editions and supported features for SQL Server 2016 - Windows](#)

[Editions and supported features for SQL Server 2014 - Windows](#)

[Installation for SQL Server](#)

[Product Specifications for SQL Server](#)

SQL Server on Linux Frequently Asked Questions (FAQ)

2/23/2018 • 5 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The following sections provide common questions and answers for SQL Server running on Linux.

General Questions

1. What Linux platforms are supported?

SQL Server is currently supported on Red Hat Enterprise Server, SUSE Linux Enterprise Server, and Ubuntu. It also runs in a container with Docker. For the latest information about the supported versions, see [Supported platforms](#).

2. Will SQL Server on Linux work on other platforms?

You can possibly install and run SQL Server on other distributions of Linux. For example, CentOS is closely related to Red Hat Enterprise Server, so you might be able to install the RPM SQL Server packages. This might be true for other closely related distributions as well. The main issue is testing and support. SQL Server has only been tested and is only supported on Red Hat Enterprise Linux, SUSE Linux Enterprise Server, and Ubuntu.

3. What SQL Server features are supported on Linux?

For a complete list of supported features and known issues, see the [Release notes](#).

4. What is the support policy for SQL Server?

To understand the support policy, review the [Technical Support Policy for SQL Server](#).

5. I am coming from a Windows SQL Server background. Are there resources to help learn how to use SQL Server on Linux?

The [quickstarts](#) provide step-by-step instructions on how to install SQL Server on Linux and run Transact-SQL queries. Other tutorials provide additional instructions on using SQL Server on Linux. For a third-party list of tips, see the [MSSQLTIPS list of SQL Server on Linux Tips](#).

Installation

1. How do I get SQL Server installed on my Linux servers?

Microsoft maintains package repositories for installing SQL Server and supports installation via native package managers such as yum, zypper, and apt. To quickly install, see one of the [quickstarts](#).

2. Can I install SQL Server on the Linux Subsystem for Windows 10?

No. Linux running on Windows 10 is currently not a supported platform for SQL Server and related tools.

3. Which Linux file systems can SQL Server 2017 use for data files?

Currently SQL Server on Linux supports ext4 and XFS. Support for other file systems will be added as needed in the future.

4. Can I download the installation packages to install SQL Server offline?

Yes. For more information, see the package download links in the [Release notes](#). Also, review the [instructions for offline installations](#).

5. Can I perform an unattended installation of SQL Server on Linux?

Yes. For a discussion of unattended installation, see [Installation guidance for SQL Server on Linux](#). See the sample scripts for [Red Hat](#), [SUSE Linux Enterprise Server](#), and [Ubuntu](#). You can also review [this sample script](#) created by the SQL Server Customer Advisory Team.

Tools

1. Can I use the SQL Server Management Studio client on Windows to access SQL Server on Linux?

Yes, you can use all your existing tools that run on Windows to access SQL Server on Linux. These include tools from Microsoft such as SQL Server Management Studio (SSMS), SQL Server Data Tools (SSDT), and OSS and third-party tools.

2. Is there a tool like SSMS that runs on Linux?

The new Microsoft SQL Operations Studio (preview) is a cross-platform tool for managing SQL Server. For more information, see [What is Microsoft SQL Operations Studio \(preview\)](#).

3. Are commands like sqlcmd and bcp available on Linux?

Yes, [sqlcmd](#) and [bcp](#) are natively available on Linux, macOS, and Windows. In addition, use the new [mssql-scripter](#) command-line tool on Linux, macOS, or Windows to generate T-SQL scripts for your SQL database running anywhere. Also, see the preview release for [mssql-cli](#).

4. Is it possible to view Activity Monitor when connected through SSMS on Windows for an instance running on Linux?

Yes, you can use SSMS on Windows to connect remotely, and use tools/ features such as Activity Monitor commands on a Linux instance.

5. What tools are available to monitor SQL Server performance on Linux?

You can use [system dynamic management views \(DMVs\)](#) to collect various types of information about SQL Server, including Linux process information. You can use [Query Store](#) to improve query performance. Other tools, such as the built-in [Performance Dashboard](#), work remotely in SQL Server Management Studio (SSMS) from Windows.

Administration

1. Has Microsoft created an app like the SQL Server Configuration Manager on Linux?

Yes, there is a configuration tool for SQL Server on Linux: [mssql-conf](#).

2. Does SQL Server on Linux support multiple instances on the same host?

We recommend running multiple containers on a host to have multiple distinct instances. Each container needs to listen on a different port. For more information, see [Run multiple SQL Server containers](#).

3. Is Active Directory Authentication supported on Linux?

Yes. For more information, see [Active Directory Authentication with SQL Server on Linux](#).

4. Are Always On and clustering supported in Linux?

Failover clustering and high availability on Linux are achieved with Pacemaker on Linux. For more

information, see [Business continuity and database recovery - SQL Server on Linux](#).

5. Is it possible to configure replication from Linux to Windows and vice versa?

Read-scale replicas can be used between Windows and Linux for one-way data replication.

6. Is it possible to migrate existing databases in older versions of SQL Server from Windows to Linux?

Yes, there are [several methods](#) of achieving this.

7. Can I migrate my data from Oracle and other database engines to SQL Server on Linux?

Yes. SSMA supports migration from several types of database engines: Microsoft Access, DB2, MySQL, Oracle, and SAP ASE (formerly SAP Sybase ASE). For an example of how to use SSMA, see [Migrate an Oracle schema to SQL Server 2017 on Linux with the SQL Server Migration Assistant](#).

8. What permissions are required for SQL Server files?

All files in the `/var/opt/mssql` file folder should be owned by the **mssql** user and belong to the **mssql** group. Both the **mssql** user and group should have read-write permissions of all files and directories. Note the following special scenarios involving file and directory permissions:

- Permissions for mssql owner and group are required for mounted network shares that are used to store SQL Server files.
- If you locate database files or backups in a non-default directory, you must also set permissions for that directory.
- If you change the default root umask from 0022, SQL Server configuration fails after installation. You must then manually apply required permissions to SQL Server startup account.

9. Can I change the ownership of SQL Server files and directories from the installed mssql account and group?

We do not support changing the ownership of SQL Server directory and files from the default installation. The mssql account and group is specifically used for SQL Server and has no interactive login access.

ⓘ Get Help

- [UserVoice - Suggestion to improve SQL Server?](#)
- [Stack Overflow \(tag sql-server\) - ask SQL development questions](#)
- [Setup and Upgrade - MSDN Forum](#)
- [SQL Server Data Tools - MSDN forum](#)
- [Reddit - general discussion about SQL Server](#)
- [Microsoft SQL Server License Terms and Information](#)
- [Support options for business users](#)
- [Contact Microsoft](#)

Quickstart: Install SQL Server and create a database on Red Hat

2/23/2018 • 6 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Linux only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

In this quickstart, you first install SQL Server 2017 on Red Hat Enterprise Linux (RHEL) 7.3+. Then connect with **sqlcmd** to create your first database and run queries.

TIP

This tutorial requires user input and an internet connection. If you are interested in the [unattended](#) or [offline](#) installation procedures, see [Installation guidance for SQL Server on Linux](#).

Prerequisites

You must have a RHEL 7.3 or 7.4 machine with **at least 2 GB** of memory.

To install Red Hat Enterprise Linux on your own machine, go to <http://access.redhat.com/products/red-hat-enterprise-linux/evaluation>. You can also create RHEL virtual machines in Azure. See [Create and Manage Linux VMs with the Azure CLI](#), and use `--image RHEL` in the call to `az vm create`.

For other system requirements, see [System requirements for SQL Server on Linux](#).

Install SQL Server

To configure SQL Server on RHEL, run the following commands in a terminal to install the **mssql-server** package:

IMPORTANT

If you have previously installed a CTP or RC release of SQL Server 2017, you must first remove the old repository before registering one of the GA repositories. For more information, see [Change repositories from the preview repository to the GA repository](#).

1. Download the Microsoft SQL Server Red Hat repository configuration file:

```
sudo curl -o /etc/yum.repos.d/mssql-server.repo https://packages.microsoft.com/config/rhel/7/mssql-server-2017.repo
```

NOTE

This is the Cumulative Update (CU) repository. For more information about your repository options and their differences, see [Configure repositories for SQL Server on Linux](#).

2. Run the following commands to install SQL Server:

```
sudo yum install -y mssql-server
```

3. After the package installation finishes, run **mssql-conf setup** and follow the prompts to set the SA password and choose your edition.

```
sudo /opt/mssql/bin/mssql-conf setup
```

TIP

If you are trying SQL Server 2017 in this tutorial, the following editions are freely licensed: Evaluation, Developer, and Express.

NOTE

Make sure to specify a strong password for the SA account (Minimum length 8 characters, including uppercase and lowercase letters, base 10 digits and/or non-alphanumeric symbols).

4. Once the configuration is done, verify that the service is running:

```
systemctl status mssql-server
```

5. To allow remote connections, open the SQL Server port on the firewall on RHEL. The default SQL Server port is TCP 1433. If you are using **FirewallD** for your firewall, you can use the following commands:

```
sudo firewall-cmd --zone=public --add-port=1433/tcp --permanent  
sudo firewall-cmd --reload
```

At this point, SQL Server is running on your RHEL machine and is ready to use!

Install the SQL Server command-line tools

To create a database, you need to connect with a tool that can run Transact-SQL statements on the SQL Server. The following steps install the SQL Server command-line tools: [sqlcmd](#) and [bcp](#).

1. Download the Microsoft Red Hat repository configuration file.

```
sudo curl -o /etc/yum.repos.d/msprod.repo https://packages.microsoft.com/config/rhel/7/prod.repo
```

2. If you had a previous version of **mssql-tools** installed, remove any older unixODBC packages.

```
sudo yum remove unixODBC-utf16 unixODBC-utf16-devel
```

3. Run the following commands to install **mssql-tools** with the unixODBC developer package.

```
sudo yum install -y mssql-tools unixODBC-devel
```

4. For convenience, add `/opt/mssql-tools/bin/` to your **PATH** environment variable. This enables you to run the tools without specifying the full path. Run the following commands to modify the **PATH** for both login

sessions and interactive/non-login sessions:

```
echo 'export PATH="$PATH:/opt/mssql-tools/bin"' >> ~/.bash_profile
echo 'export PATH="$PATH:/opt/mssql-tools/bin"' >> ~/.bashrc
source ~/.bashrc
```

TIP

Sqlcmd is just one tool for connecting to SQL Server to run queries and perform management and development tasks.

Other tools include:

- [SQL Server Operations Studio \(Preview\)](#)
- [SQL Server Management Studio](#)
- [Visual Studio Code](#).
- [mssql-cli \(Preview\)](#)

Connect locally

The following steps use **sqlcmd** to locally connect to your new SQL Server instance.

1. Run **sqlcmd** with parameters for your SQL Server name (-S), the user name (-U), and the password (-P).

In this tutorial, you are connecting locally, so the server name is `localhost`. The user name is `SA` and the password is the one you provided for the SA account during setup.

```
sqlcmd -S localhost -U SA -P '<YourPassword>'
```

TIP

You can omit the password on the command line to be prompted to enter it.

TIP

If you later decide to connect remotely, specify the machine name or IP address for the **-S** parameter, and make sure port 1433 is open on your firewall.

2. If successful, you should get to a **sqlcmd** command prompt: `1>`.

3. If you get a connection failure, first attempt to diagnose the problem from the error message. Then review the [connection troubleshooting recommendations](#).

Create and query data

The following sections walk you through using **sqlcmd** to create a new database, add data, and run a simple query.

Create a new database

The following steps create a new database named `TestDB`.

1. From the **sqlcmd** command prompt, paste the following Transact-SQL command to create a test database:

```
CREATE DATABASE TestDB
```

2. On the next line, write a query to return the name of all of the databases on your server:

```
SELECT Name from sys.Databases
```

3. The previous two commands were not executed immediately. You must type **GO** on a new line to execute the previous commands:

```
GO
```

Insert data

Next create a new table, **Inventory**, and insert two new rows.

1. From the **sqlcmd** command prompt, switch context to the new **TestDB** database:

```
USE TestDB
```

2. Create new table named **Inventory**:

```
CREATE TABLE Inventory (id INT, name NVARCHAR(50), quantity INT)
```

3. Insert data into the new table:

```
INSERT INTO Inventory VALUES (1, 'banana', 150); INSERT INTO Inventory VALUES (2, 'orange', 154);
```

4. Type **GO** to execute the previous commands:

```
GO
```

Select data

Now, run a query to return data from the **Inventory** table.

1. From the **sqlcmd** command prompt, enter a query that returns rows from the **Inventory** table where the quantity is greater than 152:

```
SELECT * FROM Inventory WHERE quantity > 152;
```

2. Execute the command:

```
GO
```

Exit the sqlcmd command prompt

To end your **sqlcmd** session, type **QUIT**:

```
QUIT
```

Connect from Windows

SQL Server tools on Windows connect to SQL Server instances on Linux in the same way they would connect to any remote SQL Server instance.

If you have a Windows machine that can connect to your Linux machine, try the same steps in this topic from a Windows command-prompt running **sqlcmd**. Just verify that you use the target Linux machine name or IP address rather than localhost, and make sure that TCP port 1433 is open. If you have any problems connecting from Windows, see [connection troubleshooting recommendations](#).

For other tools that run on Windows but connect to SQL Server on Linux, see:

- [SQL Server Management Studio \(SSMS\)](#)
- [Windows PowerShell](#)
- [SQL Server Data Tools \(SSDT\)](#)

Additional resources

For other installation scenarios, see the following resources:

Upgrade	Learn how to upgrade an existing installation of SQL Server on Linux
Uninstall	Uninstall SQL Server on Linux
Unattended install	Learn how to script the installation without prompts
Offline install	Learn how to manually download the packages for offline installation

To explore other ways to connect and manage SQL Server, explore the following tools:

Visual Studio Code	A cross-platform GUI code editor that runs Transact-SQL statements with the mssql extension.
SQL Server Operations Studio	A cross-platform GUI database management utility.
mssql-cli	A cross-platform command-line interface for running Transact-SQL commands.
SQL Server Management Studio	A Windows-based GUI database management utility that can connect to and manage SQL Server instances on Linux.

To learn more about writing Transact-SQL statements and queries, see [Tutorial: Writing Transact-SQL Statements](#).

TIP

For answers to frequently asked questions, see the [SQL Server on Linux FAQ](#).

Next steps

[Explore the tutorials for SQL Server on Linux](#)

Quickstart: Install SQL Server and create a database on SUSE Linux Enterprise Server

2/23/2018 • 6 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Linux only) ✖ Azure SQL Database ✖ Azure SQL Data Warehouse ✖ Parallel Data Warehouse

In this quickstart, you first install SQL Server 2017 on SUSE Linux Enterprise Server (SLES) v12 SP2. Then connect with **sqlcmd** to create your first database and run queries.

TIP

This tutorial requires user input and an internet connection. If you are interested in the [unattended](#) or [offline](#) installation procedures, see [Installation guidance for SQL Server on Linux](#).

Prerequisites

You must have a SLES v12 SP2 machine with **at least 2 GB** of memory. The file system must be **XFS** or **EXT4**. Other file systems, such as **BTRFS**, are unsupported.

To install SUSE Linux Enterprise Server on your own machine, go to <https://www.suse.com/products/server>. You can also create SLES virtual machines in Azure. See [Create and Manage Linux VMs with the Azure CLI](#), and use `--image SLES` in the call to `az vm create`.

NOTE

At this time, the [Windows Subsystem for Linux](#) for Windows 10 is not supported as an installation target.

For other system requirements, see [System requirements for SQL Server on Linux](#).

Install SQL Server

To configure SQL Server on SLES, run the following commands in a terminal to install the **mssql-server** package:

IMPORTANT

If you have previously installed a CTP or RC release of SQL Server 2017, you must first remove the old repository before registering one of the GA repositories. For more information, see [Change repositories from the preview repository to the GA repository](#).

1. Download the Microsoft SQL Server SLES repository configuration file:

```
sudo zypper addrepo -fc https://packages.microsoft.com/config/sles/12/mssql-server-2017.repo
```

NOTE

This is the Cumulative Update (CU) repository. For more information about your repository options and their differences, see [Configure repositories for SQL Server on Linux](#).

2. Refresh your repositories.

```
sudo zypper --gpg-auto-import-keys refresh
```

3. Run the following commands to install SQL Server:

```
sudo zypper install -y mssql-server
```

4. After the package installation finishes, run **mssql-conf setup** and follow the prompts to set the SA password and choose your edition.

```
sudo /opt/mssql/bin/mssql-conf setup
```

TIP

If you are trying SQL Server 2017 in this tutorial, the following editions are freely licensed: Evaluation, Developer, and Express.

NOTE

Make sure to specify a strong password for the SA account (Minimum length 8 characters, including uppercase and lowercase letters, base 10 digits and/or non-alphanumeric symbols).

5. Once the configuration is done, verify that the service is running:

```
systemctl status mssql-server
```

6. If you plan to connect remotely, you might also need to open the SQL Server TCP port (default 1433) on your firewall. If you are using the SuSE firewall, you need to edit the **/etc/sysconfig/SuSEfirewall2** configuration file. Modify the **FW_SERVICES_EXT_TCP** entry to include the SQL Server port number.

```
FW_SERVICES_EXT_TCP="1433"
```

At this point, SQL Server is running on your SLES machine and is ready to use!

Install the SQL Server command-line tools

To create a database, you need to connect with a tool that can run Transact-SQL statements on the SQL Server. The following steps install the SQL Server command-line tools: [sqlcmd](#) and [bcp](#).

1. Add the Microsoft SQL Server repository to Zypper.

```
sudo zypper addrepo -fc https://packages.microsoft.com/config/sles/12/prod.repo
sudo zypper --gpg-auto-import-keys refresh
```

2. Install **mssql-tools** with the unixODBC developer package.

```
sudo zypper install -y mssql-tools unixODBC-devel
```

3. For convenience, add `/opt/mssql-tools/bin/` to your **PATH** environment variable. This enables you to run the tools without specifying the full path. Run the following commands to modify the **PATH** for both login sessions and interactive/non-login sessions:

```
echo 'export PATH="$PATH:/opt/mssql-tools/bin"' >> ~/.bash_profile
echo 'export PATH="$PATH:/opt/mssql-tools/bin"' >> ~/.bashrc
source ~/.bashrc
```

TIP

Sqlcmd is just one tool for connecting to SQL Server to run queries and perform management and development tasks. Other tools include:

- [SQL Server Operations Studio \(Preview\)](#)
- [SQL Server Management Studio](#)
- [Visual Studio Code](#).
- [mssql-cli \(Preview\)](#)

Connect locally

The following steps use **sqlcmd** to locally connect to your new SQL Server instance.

1. Run **sqlcmd** with parameters for your SQL Server name (-S), the user name (-U), and the password (-P). In this tutorial, you are connecting locally, so the server name is `localhost`. The user name is `SA` and the password is the one you provided for the SA account during setup.

```
sqlcmd -S localhost -U SA -P '<YourPassword>'
```

TIP

You can omit the password on the command line to be prompted to enter it.

TIP

If you later decide to connect remotely, specify the machine name or IP address for the **-S** parameter, and make sure port 1433 is open on your firewall.

2. If successful, you should get to a **sqlcmd** command prompt: `1>`.

3. If you get a connection failure, first attempt to diagnose the problem from the error message. Then review the [connection troubleshooting recommendations](#).

Create and query data

The following sections walk you through using **sqlcmd** to create a new database, add data, and run a simple query.

Create a new database

The following steps create a new database named `TestDB`.

- From the **sqlcmd** command prompt, paste the following Transact-SQL command to create a test database:

```
CREATE DATABASE TestDB
```

- On the next line, write a query to return the name of all of the databases on your server:

```
SELECT Name from sys.Databases
```

- The previous two commands were not executed immediately. You must type `GO` on a new line to execute the previous commands:

```
GO
```

Insert data

Next create a new table, `Inventory`, and insert two new rows.

- From the **sqlcmd** command prompt, switch context to the new `TestDB` database:

```
USE TestDB
```

- Create new table named `Inventory`:

```
CREATE TABLE Inventory (id INT, name NVARCHAR(50), quantity INT)
```

- Insert data into the new table:

```
INSERT INTO Inventory VALUES (1, 'banana', 150); INSERT INTO Inventory VALUES (2, 'orange', 154);
```

- Type `GO` to execute the previous commands:

```
GO
```

Select data

Now, run a query to return data from the `Inventory` table.

- From the **sqlcmd** command prompt, enter a query that returns rows from the `Inventory` table where the quantity is greater than 152:

```
SELECT * FROM Inventory WHERE quantity > 152;
```

- Execute the command:

```
GO
```

Exit the `sqlcmd` command prompt

To end your `sqlcmd` session, type `QUIT`:

```
QUIT
```

Connect from Windows

SQL Server tools on Windows connect to SQL Server instances on Linux in the same way they would connect to any remote SQL Server instance.

If you have a Windows machine that can connect to your Linux machine, try the same steps in this topic from a Windows command-prompt running `sqlcmd`. Just verify that you use the target Linux machine name or IP address rather than localhost, and make sure that TCP port 1433 is open. If you have any problems connecting from Windows, see [connection troubleshooting recommendations](#).

For other tools that run on Windows but connect to SQL Server on Linux, see:

- [SQL Server Management Studio \(SSMS\)](#)
- [Windows PowerShell](#)
- [SQL Server Data Tools \(SSDT\)](#)

Additional resources

For other installation scenarios, see the following resources:

Upgrade	Learn how to upgrade an existing installation of SQL Server on Linux
Uninstall	Uninstall SQL Server on Linux
Unattended install	Learn how to script the installation without prompts
Offline install	Learn how to manually download the packages for offline installation

To explore other ways to connect and manage SQL Server, explore the following tools:

Visual Studio Code	A cross-platform GUI code editor that run Transact-SQL statements with the <code>mssql</code> extension.
SQL Server Operations Studio	A cross-platform GUI database management utility.
mssql-cli	A cross-platform command-line interface for running Transact-SQL commands.
SQL Server Management Studio	A Windows-based GUI database management utility that can connect to and manage SQL Server instances on Linux.

To learn more about writing Transact-SQL statements and queries, see [Tutorial: Writing Transact-SQL Statements](#).

TIP

For answers to frequently asked questions, see the [SQL Server on Linux FAQ](#).

Next steps

[Explore the tutorials for SQL Server on Linux](#)

Quickstart: Install SQL Server and create a database on Ubuntu

2/23/2018 • 6 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

In this quickstart, you first install SQL Server 2017 on Ubuntu 16.04. Then connect with **sqlcmd** to create your first database and run queries.

TIP

This tutorial requires user input and an internet connection. If you are interested in the [unattended](#) or [offline](#) installation procedures, see [Installation guidance for SQL Server on Linux](#).

Prerequisites

You must have a Ubuntu 16.04 machine with **at least 2 GB** of memory.

To install Ubuntu on your own machine, go to <http://www.ubuntu.com/download/server>. You can also create Ubuntu virtual machines in Azure. See [Create and Manage Linux VMs with the Azure CLI](#).

NOTE

At this time, the [Windows Subsystem for Linux](#) for Windows 10 is not supported as an installation target.

For other system requirements, see [System requirements for SQL Server on Linux](#).

Install SQL Server

To configure SQL Server on Ubuntu, run the following commands in a terminal to install the **mssql-server** package.

IMPORTANT

If you have previously installed a CTP or RC release of SQL Server 2017, you must first remove the old repository before registering one of the GA repositories. For more information, see [Change repositories from the preview repository to the GA repository](#).

1. Import the public repository GPG keys:

```
wget -qO- https://packages.microsoft.com/keys/microsoft.asc | sudo apt-key add -
```

2. Register the Microsoft SQL Server Ubuntu repository:

```
sudo add-apt-repository "$(wget -qO- https://packages.microsoft.com/config/ubuntu/16.04/mssql-server-2017.list)"
```

NOTE

This is the Cumulative Update (CU) repository. For more information about your repository options and their differences, see [Configure repositories for SQL Server on Linux](#).

3. Run the following commands to install SQL Server:

```
sudo apt-get update  
sudo apt-get install -y mssql-server
```

4. After the package installation finishes, run **mssql-conf setup** and follow the prompts to set the SA password and choose your edition.

```
sudo /opt/mssql/bin/mssql-conf setup
```

TIP

If you are trying SQL Server 2017 in this tutorial, the following editions are freely licensed: Evaluation, Developer, and Express.

NOTE

Make sure to specify a strong password for the SA account (Minimum length 8 characters, including uppercase and lowercase letters, base 10 digits and/or non-alphanumeric symbols).

5. Once the configuration is done, verify that the service is running:

```
systemctl status mssql-server
```

6. If you plan to connect remotely, you might also need to open the SQL Server TCP port (default 1433) on your firewall.

At this point, SQL Server is running on your Ubuntu machine and is ready to use!

Install the SQL Server command-line tools

To create a database, you need to connect with a tool that can run Transact-SQL statements on the SQL Server. The following steps install the SQL Server command-line tools: [sqlcmd](#) and [bcp](#).

1. Import the public repository GPG keys:

```
wget -qO- https://packages.microsoft.com/keys/microsoft.asc | sudo apt-key add -
```

2. Register the Microsoft Ubuntu repository:

```
sudo add-apt-repository "$(wget -qO- https://packages.microsoft.com/config/ubuntu/16.04/prod.list)"
```

3. Update the sources list and run the installation command with the unixODBC developer package:

```
sudo apt-get update  
sudo apt-get install -y mssql-tools unixodbc-dev
```

4. For convenience, add `/opt/mssql-tools/bin/` to your **PATH** environment variable. This enables you to run the tools without specifying the full path. Run the following commands to modify the **PATH** for both login sessions and interactive/non-login sessions:

```
echo 'export PATH="$PATH:/opt/mssql-tools/bin"' >> ~/.bash_profile  
echo 'export PATH="$PATH:/opt/mssql-tools/bin"' >> ~/.bashrc  
source ~/.bashrc
```

TIP

sqlcmd is just one tool for connecting to SQL Server to run queries and perform management and development tasks. Other tools include:

- [SQL Server Operations Studio \(Preview\)](#)
- [SQL Server Management Studio](#)
- [Visual Studio Code.](#)
- [mssql-cli \(Preview\)](#)

Connect locally

The following steps use **sqlcmd** to locally connect to your new SQL Server instance.

1. Run **sqlcmd** with parameters for your SQL Server name (-S), the user name (-U), and the password (-P). In this tutorial, you are connecting locally, so the server name is `localhost`. The user name is `SA` and the password is the one you provided for the SA account during setup.

```
sqlcmd -S localhost -U SA -P '<YourPassword>'
```

TIP

You can omit the password on the command line to be prompted to enter it.

TIP

If you later decide to connect remotely, specify the machine name or IP address for the **-S** parameter, and make sure port 1433 is open on your firewall.

2. If successful, you should get to a **sqlcmd** command prompt: `1>`.
3. If you get a connection failure, first attempt to diagnose the problem from the error message. Then review the [connection troubleshooting recommendations](#).

Create and query data

The following sections walk you through using **sqlcmd** to create a new database, add data, and run a simple query.

Create a new database

The following steps create a new database named `TestDB`.

1. From the **sqlcmd** command prompt, paste the following Transact-SQL command to create a test database:

```
CREATE DATABASE TestDB
```

2. On the next line, write a query to return the name of all of the databases on your server:

```
SELECT Name from sys.Databases
```

3. The previous two commands were not executed immediately. You must type `GO` on a new line to execute the previous commands:

```
GO
```

Insert data

Next create a new table, `Inventory`, and insert two new rows.

1. From the **sqlcmd** command prompt, switch context to the new `TestDB` database:

```
USE TestDB
```

2. Create new table named `Inventory`:

```
CREATE TABLE Inventory (id INT, name NVARCHAR(50), quantity INT)
```

3. Insert data into the new table:

```
INSERT INTO Inventory VALUES (1, 'banana', 150); INSERT INTO Inventory VALUES (2, 'orange', 154);
```

4. Type `GO` to execute the previous commands:

```
GO
```

Select data

Now, run a query to return data from the `Inventory` table.

1. From the **sqlcmd** command prompt, enter a query that returns rows from the `Inventory` table where the quantity is greater than 152:

```
SELECT * FROM Inventory WHERE quantity > 152;
```

2. Execute the command:

```
GO
```

Exit the sqlcmd command prompt

To end your **sqlcmd** session, type `QUIT`:

```
QUIT
```

Connect from Windows

SQL Server tools on Windows connect to SQL Server instances on Linux in the same way they would connect to any remote SQL Server instance.

If you have a Windows machine that can connect to your Linux machine, try the same steps in this topic from a Windows command-prompt running **sqlcmd**. Just verify that you use the target Linux machine name or IP address rather than localhost, and make sure that TCP port 1433 is open. If you have any problems connecting from Windows, see [connection troubleshooting recommendations](#).

For other tools that run on Windows but connect to SQL Server on Linux, see:

- [SQL Server Management Studio \(SSMS\)](#)
- [Windows PowerShell](#)
- [SQL Server Data Tools \(SSDT\)](#)

Additional resources

For other installation scenarios, see the following resources:

Upgrade	Learn how to upgrade an existing installation of SQL Server on Linux
Uninstall	Uninstall SQL Server on Linux
Unattended install	Learn how to script the installation without prompts
Offline install	Learn how to manually download the packages for offline installation

To explore other ways to connect and manage SQL Server, explore the following tools:

Visual Studio Code	A cross-platform GUI code editor that runs Transact-SQL statements with the <code>mssql</code> extension.
SQL Server Operations Studio	A cross-platform GUI database management utility.
mssql-cli	A cross-platform command-line interface for running Transact-SQL commands.
SQL Server Management Studio	A Windows-based GUI database management utility that can connect to and manage SQL Server instances on Linux.

To learn more about writing Transact-SQL statements and queries, see [Tutorial: Writing Transact-SQL Statements](#).

TIP

For answers to frequently asked questions, see the [SQL Server on Linux FAQ](#).

Next steps

[Explore the tutorials for SQL Server on Linux](#)

Quickstart: Run the SQL Server 2017 container image with Docker

3/7/2018 • 7 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

In this quickstart, you use Docker to pull and run the SQL Server 2017 container image, [mssql-server-linux](#). Then connect with **sqlcmd** to create your first database and run queries.

This image consists of SQL Server running on Linux based on Ubuntu 16.04. It can be used with the Docker Engine 1.8+ on Linux or on Docker for Mac/Windows.

NOTE

This quick start specifically focuses on using the [mssql-server-linux](#) image. The Windows image is not covered, but you can learn more about it on the [mssql-server-windows-developer Docker Hub page](#).

Prerequisites

- Docker Engine 1.8+ on any supported Linux distribution or Docker for Mac/Windows. For more information, see [Install Docker](#).
- Minimum of 2 GB of disk space
- Minimum of 2 GB of RAM
- [System requirements for SQL Server on Linux](#).

Pull and run the container image

1. Pull the SQL Server 2017 Linux container image from Docker Hub.

```
sudo docker pull microsoft/mssql-server-linux:2017-latest
```

```
docker pull microsoft/mssql-server-linux:2017-latest
```

The previous command pulls the latest SQL Server 2017 container image. If you want to pull a specific image, you add a colon and the tag name (for example, `microsoft/mssql-server-linux:2017-GA`). To see all available images, see [the mssql-server-linux Docker hub page](#).

For the bash commands in this article, `sudo` is used. On MacOS, `sudo` might not be required. On Linux, if you do not want to use `sudo` to run Docker, you can configure a **docker** group and add users to that group. For more information, see [Post-installation steps for Linux](#).

2. To run the container image with Docker, you can use the following command from a bash shell (Linux/macOS) or elevated PowerShell command prompt.

```
sudo docker run -e 'ACCEPT_EULA=Y' -e 'MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>' \
-p 1401:1433 --name sql1 \
-d microsoft/mssql-server-linux:2017-latest
```

```
docker run -e "ACCEPT_EULA=Y" -e "MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>" `  
-p 1401:1433 --name sql1 `  
-d microsoft/mssql-server-linux:2017-latest
```

NOTE

The password should follow the SQL Server default password policy, otherwise the container can not setup SQL server and will stop working. By default, the password must be at least 8 characters long and contain characters from three of the following four sets: Uppercase letters, Lowercase letters, Base 10 digits, and Symbols. You can examine the error log by executing the [docker logs](#) command.

NOTE

By default, this creates a container with the Developer edition of SQL Server 2017. The process for running production editions in containers is slightly different. For more information, see [Run production container images](#).

The following table provides a description of the parameters in the previous `docker run` example:

PARAMETER	DESCRIPTION
<code>-e 'ACCEPT_EULA=Y'</code>	Set the ACCEPT_EULA variable to any value to confirm your acceptance of the End-User Licensing Agreement . Required setting for the SQL Server image.
<code>-e 'MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>'</code>	Specify your own strong password that is at least 8 characters and meets the SQL Server password requirements . Required setting for the SQL Server image.
<code>-p 1401:1433</code>	Map a TCP port on the host environment (first value) with a TCP port in the container (second value). In this example, SQL Server is listening on TCP 1433 in the container and this is exposed to the port, 1401, on the host.
<code>--name sql1</code>	Specify a custom name for the container rather than a randomly generated one. If you run more than one container, you cannot reuse this same name.
<code>microsoft/mssql-server-linux:2017-latest</code>	The SQL Server 2017 Linux container image.

- To view your Docker containers, use the `docker ps` command.

```
sudo docker ps -a
```

```
docker ps -a
```

You should see output similar to the following screenshot:

```
PS C:\WINDOWS\system32> docker ps -a  
CONTAINER ID IMAGE CREATED STATUS PORTS NAMES  
0075848e1f48 microsoft/mssql-server-linux 5 seconds ago Up 3 seconds 0.0.0.0:1401->1433/tcp sqlcontainer1
```

- If the **STATUS** column shows a status of **Up**, then SQL Server is running in the container and listening on the port specified in the **PORTS** column. If the **STATUS** column for your SQL Server container shows

Exited, see the [Troubleshooting section of the configuration guide](#).

The `-h` (host name) parameter is also useful, but it is not used in this tutorial for simplicity. This changes the internal name of the container to a custom value. This is the name you'll see returned in the following Transact-SQL query:

```
SELECT @@SERVERNAME,
    SERVERPROPERTY('ComputerNamePhysicalNetBIOS'),
    SERVERPROPERTY('MachineName'),
    SERVERPROPERTY('ServerName')
```

Setting `-h` and `--name` to the same value is a good way to easily identify the target container.

Change the SA password

The **SA** account is a system administrator on the SQL Server instance that gets created during setup. After creating your SQL Server container, the `MSSQL_SA_PASSWORD` environment variable you specified is discoverable by running `echo $MSSQL_SA_PASSWORD` in the container. For security purposes, change your SA password.

1. Choose a strong password to use for the SA user.
2. Use `docker exec` to run **sqlcmd** to change the password using Transact-SQL. Replace `<YourStrong!Passw0rd>` and `<YourNewStrong!Passw0rd>` with your own password values.

```
sudo docker exec -it sql1 /opt/mssql-tools/bin/sqlcmd \
-S localhost -U SA -P '<YourStrong!Passw0rd>' \
-Q 'ALTER LOGIN SA WITH PASSWORD="<YourNewStrong!Passw0rd>"'
```

```
docker exec -it sql1 /opt/mssql-tools/bin/sqlcmd \
-S localhost -U SA -P "<YourStrong!Passw0rd>" \
-Q "ALTER LOGIN SA WITH PASSWORD='<YourNewStrong!Passw0rd>'"
```

Connect to SQL Server

The following steps use the SQL Server command-line tool, **sqlcmd**, inside the container to connect to SQL Server.

1. Use the `docker exec -it` command to start an interactive bash shell inside your running container. In the following example `sql1` is name specified by the `--name` parameter when you created the container.

```
sudo docker exec -it sql1 "bash"
```

```
docker exec -it sql1 "bash"
```

2. Once inside the container, connect locally with sqlcmd. Sqlcmd is not in the path by default, so you have to specify the full path.

```
/opt/mssql-tools/bin/sqlcmd -S localhost -U SA -P '<YourNewStrong!Passw0rd>'
```

TIP

You can omit the password on the command-line to be prompted to enter it.

3. If successful, you should get to a **sqlcmd** command prompt: `1>`.

Create and query data

The following sections walk you through using **sqlcmd** and Transact-SQL to create a new database, add data, and run a simple query.

Create a new database

The following steps create a new database named `TestDB`.

1. From the **sqlcmd** command prompt, paste the following Transact-SQL command to create a test database:

```
CREATE DATABASE TestDB
```

2. On the next line, write a query to return the name of all of the databases on your server:

```
SELECT Name from sys.Databases
```

3. The previous two commands were not executed immediately. You must type `GO` on a new line to execute the previous commands:

```
GO
```

Insert data

Next create a new table, `Inventory`, and insert two new rows.

1. From the **sqlcmd** command prompt, switch context to the new `TestDB` database:

```
USE TestDB
```

2. Create new table named `Inventory`:

```
CREATE TABLE Inventory (id INT, name NVARCHAR(50), quantity INT)
```

3. Insert data into the new table:

```
INSERT INTO Inventory VALUES (1, 'banana', 150); INSERT INTO Inventory VALUES (2, 'orange', 154);
```

4. Type `GO` to execute the previous commands:

```
GO
```

Select data

Now, run a query to return data from the `Inventory` table.

- From the **sqlcmd** command prompt, enter a query that returns rows from the **Inventory** table where the quantity is greater than 152:

```
SELECT * FROM Inventory WHERE quantity > 152;
```

- Execute the command:

```
GO
```

Exit the sqlcmd command prompt

- To end your **sqlcmd** session, type **QUIT**:

```
QUIT
```

- To exit the interactive command-prompt in your container, type **exit**. Your container continues to run after you exit the interactive bash shell.

Connect from outside the container

You can also connect to the SQL Server instance on your Docker machine from any external Linux, Windows, or macOS tool that supports SQL connections.

The following steps use **sqlcmd** outside of your container to connect to SQL Server running in the container. These steps assume that you already have the SQL Server command-line tools installed outside of your container. The same principals apply when using other tools, but the process of connecting is unique to each tool.

- Find the IP address for the machine that hosts your container. On Linux, use **ifconfig** or **ip addr**. On Windows, use **ipconfig**.
- Run **sqlcmd** specifying the IP address and the port mapped to port 1433 in your container. In this example, that is port 1401 on the host machine.

```
sqlcmd -S 10.3.2.4,1401 -U SA -P '<YourNewStrong!Passw0rd>'
```

```
sqlcmd -S 10.3.2.4,1401 -U SA -P "<YourNewStrong!Passw0rd>"
```

- Run Transact-SQL commands. When finished, type **QUIT**.

Other common tools to connect to SQL Server include:

- [Visual Studio Code](#)
- [SQL Server Management Studio \(SSMS\) on Windows](#)
- [SQL Server Operations Studio \(Preview\)](#)
- [mssql-cli \(Preview\)](#)

Remove your container

If you want to remove the SQL Server container used in this tutorial, run the following commands:

```
sudo docker stop sql1
sudo docker rm sql1
```

```
docker stop sql1
docker rm sql1
```

WARNING

Stopping and removing a container permanently deletes any SQL Server data in the container. If you need to preserve your data, [create and copy a backup file out of the container](#) or use a [container data persistence technique](#).

Docker demo

After you have tried using the SQL Server container image for Docker, you might want to know how Docker is used to improve development and testing. The following video shows how Docker can be used in a continuous integration and deployment scenario.

Next steps

For a tutorial on how to restore database backup files into a container, see [Restore a SQL Server database in a Linux Docker container](#). To explore other scenarios, such as running multiple containers, data persistence, and troubleshooting, see [Configure SQL Server 2017 container images on Docker](#).

Also, check out the [mssql-docker GitHub repository](#) for resources, feedback, and known issues.

Quickstart: Run the SQL Server 2017 in the cloud

2/23/2018 • 1 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

In this quickstart, you will install SQL Server 2017 on Red Hat Enterprise Linux (RHEL), SUSE Linux Enterprise Server (SLES), or Ubuntu in the cloud of your choice. Go to [Provision a Linux SQL Server virtual machine in the Azure portal](#) to run SQL Server on Linux in Azure.

- > [!NOTE]
- > If you choose to run a paid edition of SQL Server then you need to bring your own license (BYOL)

Amazon Web Services

1. Create a Linux AMI with at least 2 GB of memory from the marketplace
 - [RHEL 7.3+](#)
 - [SLES v12 SP2](#)
 - [Ubuntu 16.04](#)
2. Connect to the AMI with ssh
3. Follow the quickstart for the Linux distribution you chose:
 - [RHEL](#)
 - [SLES](#)
 - [Ubuntu](#)
4. Configure for remote connections:
 - Open the [Amazon EC2 console](#)
 - In the navigation pane, choose **Security Groups**.
 - Choose **Inbound, Edit, Add Rule**
 - Add an inbound rule to allow traffic on the port on which SQL Server listens (default TCP port 1433)

Digital Ocean

1. Log in to the [control panel](#) and click create a droplet
2. Choose a Ubuntu 16.04 droplet with at least 2 GB of memory
3. Connect to the droplet with ssh
4. Follow the [Ubuntu quickstart](#)
5. Configure for remote connections:
 - At the top of the Control Panel, follow the **Networking** link and then select **Firewalls**
 - Add an inbound rule to allow traffic on the port on which SQL Server listens (default TCP port 1433)

Google Cloud Platform

1. Create a Linux image with at least 2 GB of memory from the Cloud Launcher
 - [RHEL 7.3+](#)
 - [SLES v12 SP2](#)
 - [Ubuntu 16.04](#)

2. Connect to the image with ssh
3. Follow the quickstart for the Linux distribution you chose:
 - [RHEL](#)
 - [SLES](#)
 - [Ubuntu](#)
4. Configure for remote connections:
 - Go to the [Firewall Rules](#)
 - Add an inbound rule to allow traffic on the port on which SQL Server listens (default tcp: 1433)

Migrate a SQL Server database from Windows to Linux using backup and restore

2/14/2018 • 5 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

SQL Server's backup and restore feature is the recommended way to migrate a database from SQL Server on Windows to SQL Server 2017 on Linux. In this tutorial, you will walk through the steps required to move a database to Linux with backup and restore techniques.

- Create a backup file on Windows with SSMS
- Install a Bash shell on Windows
- Move the backup file to Linux from the Bash shell
- Restore the backup file on Linux with Transact-SQL
- Run a query to verify the migration

You can also create a SQL Server Always On Availability Group to migrate a SQL Server database from Windows to Linux. See [sql-server-linux-availability-group-cross-platform](#).

Prerequisites

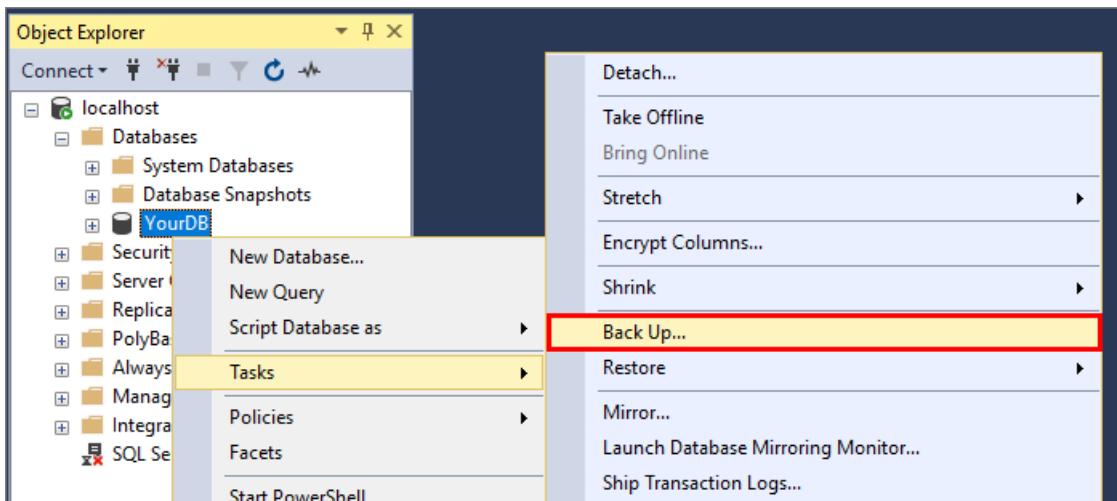
The following prerequisites are required to complete this tutorial:

- Windows machine with the following:
 - [SQL Server](#) installed.
 - [SQL Server Management Studio](#) installed.
 - Target database to migrate.
- Linux machine with the following installed:
 - SQL Server 2017 ([RHEL](#), [SLES](#), or [Ubuntu](#)) with command-line tools.

Create a backup on Windows

There are several ways to create a backup file of a database on Windows. The following steps use SQL Server Management Studio (SSMS).

1. Start **SQL Server Management Studio** on your Windows machine.
2. In the connection dialog, enter **localhost**.
3. In Object Explorer, expand **Databases**.
4. Right-click your target database, select **Tasks**, and then click **Back Up....**



5. In the **Backup Up Database** dialog, verify that **Backup type** is **Full** and **Back up to** is **Disk**. Note name and location of the file. For example, a database named **YourDB** on SQL Server 2016 has a default backup path of `C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\MSSQL\Backup\YourDB.bak`.
6. Click **OK** to back up your database.

NOTE

Another option is to run a Transact-SQL query to create the backup file. The following Transact-SQL command performs the same actions as the previous steps for a database called **YourDB**:

```
BACKUP DATABASE [YourDB] TO DISK =
N'C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\MSSQL\Backup\YourDB.bak'
WITH NOFORMAT, NOINIT, NAME = N'YourDB-Full Database Backup',
SKIP, NOREWIND, NOUNLOAD, STATS = 10
GO
```

Install a Bash shell on Windows

To restore the database, you must first transfer the backup file from the Windows machine to the target Linux machine. In this tutorial, we move the file to Linux from a Bash shell (terminal window) running on Windows.

1. Install a Bash shell on your Windows machine that supports the **scp** (secure copy) and **ssh** (remote login) commands. Two examples include:
 - The [Windows Subsystem for Linux](#) (Windows 10)
 - The Git Bash Shell (<https://git-scm.com/downloads>)
2. Open a Bash session on Windows.

Copy the backup file to Linux

1. In your Bash session, navigate to the directory containing your backup file. For example:

```
cd 'C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\MSSQL\Backup\'
```

2. Use the **scp** command to transfer the file to the target Linux machine. The following example transfers **YourDB.bak** to the home directory of **user1** on the Linux server with an IP address of **192.0.2.9**:

```
scp YourDB.bak user1@192.0.2.9:/
```

```
MINGW32:/c/Program Files/Microsoft SQL Server/MSSQL13.MSSQLSERVER/MSSQL/Backup
azureadmin@sqlwin MINGW32 ~
$ cd 'C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\MSSQL\Backup\' 

azureadmin@sqlwin MINGW32 /c/Program Files/Microsoft SQL Server/MSSQL13.MSSQLSERVER/
MSSQL/Backup
$ ls
YourDB.bak

azureadmin@sqlwin MINGW32 /c/Program Files/Microsoft SQL Server/MSSQL13.MSSQLSERVER/
MSSQL/Backup
$ scp YourDB.bak user1@192.0.2.9:/
The authenticity of host '192.0.2.9(192.0.2.9)' can't be established.
ECDSA key fingerprint is SHA256: [REDACTED].
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.0.2.9' (ECDSA) to the list of known hosts.
Password:
YourDB.bak                                         100% 8960KB   7.4MB/s   00:01
```

TIP

There are alternatives to using `scp` for file transfer. One is to use [Samba](#) to configure an SMB network share between Windows and Linux. For a walkthrough on Ubuntu, see [How to Create a Network Share Via Samba](#). Once established, you can access it as a network file share from Windows, such as `\machinenameorip\share`.

Move the backup file before restoring

At this point, the backup file is on your Linux server in your user's home directory. Before restoring the database to SQL Server, you must place the backup in a subdirectory of **/var/opt/mssql**.

1. In the same Windows Bash session, connect remotely to your target Linux machine with `ssh`. The following example connects to the Linux machine **192.0.2.9** as user **user1**.

```
ssh user1@192.0.2.9
```

You are now running commands on the remote Linux server.

2. Enter super user mode.

```
sudo su
```

3. Create a new backup directory. The `-p` parameter does nothing if the directory already exists.

```
mkdir -p /var/opt/mssql/backup
```

4. Move the backup file to that directory. In the following example, the backup file resides in the home directory of `user1`. Change the command to match the location and file name of your backup file.

```
mv /home/user1/YourDB.bak /var/opt/mssql/backup/
```

5. Exit super user mode.

```
exit
```

Restore your database on Linux

To restore the database backup, you can use the **RESTORE DATABASE** Transact-SQL (TSQL) command.

NOTE

The following steps use the **sqlcmd** tool. If you haven't install SQL Server Tools, see [Install SQL Server command-line tools on Linux](#).

1. In the same terminal, launch **sqlcmd**. The following example connects to the local SQL Server instance with the **SA** user. Enter the password when prompted, or specify the password by adding the **-P** parameter.

```
sqlcmd -S localhost -U SA
```

2. At the **>1** prompt, enter the following **RESTORE DATABASE** command, pressing ENTER after each line (you cannot copy and paste the entire multi-line command at once). Replace all occurrences of **YourDB** with the name of your database.

```
RESTORE DATABASE YourDB
FROM DISK = '/var/opt/mssql/backup/YourDB.bak'
WITH MOVE 'YourDB' TO '/var/opt/mssql/data/YourDB.mdf',
MOVE 'YourDB_Log' TO '/var/opt/mssql/data/YourDB_Log.ldf'
GO
```

You should get a message the database is successfully restored.

3. Verify the restoration by listing all of the databases on the server. The restored database should be listed.

```
SELECT Name FROM sys.Databases
GO
```

4. Run other queries on your migrated database. The following command switches context to the **YourDB** database and selects rows from one of its tables.

```
USE YourDB
SELECT * FROM YourTable
GO
```

5. When you are done using **sqlcmd**, type **exit**.

6. When you are done working in the remote **ssh** session, type **exit** again.

Next steps

In this tutorial, you learned how to back up a database on Windows and move it to a Linux server running SQL Server 2017. You learned how to:

- Use SSMS and Transact-SQL to create a backup file on Windows
- Install a Bash shell on Windows
- Use **scp** to move backup files from Windows to Linux
- Use **ssh** to remotely connect to your Linux machine
- Relocate the backup file to prepare for restore
- Use **sqlcmd** to run Transact-SQL commands
- Restore the database backup with the **RESTORE DATABASE** command

- Run the query to verify the migration

Next, explore other migration scenarios for SQL Server on Linux.

[Migrate databases to SQL Server on Linux](#)

Migrate an Oracle schema to SQL Server 2017 on Linux with the SQL Server Migration Assistant

12/20/2017 • 2 min to read • [Edit Online](#)

This tutorial uses SQL Server Migration Assistant (SSMA) for Oracle on Windows to convert the Oracle sample **HR** schema to [SQL Server 2017 on Linux](#).

- Download and install SSMA on Windows
- Create an SSMA project to manage the migration
- Connect to Oracle
- Run a migration report
- Convert the sample HR schema
- Migrate the data

Prerequisites

- An instance of Oracle 12c (12.2.0.1.0) with the **HR** schema installed
- A working instance of SQL Server on Linux

NOTE

The same steps can be used to target SQL Server on Windows, but you must select Windows in the **Migrate To** project setting.

Download and install SSMA for Oracle

There are several editions of SQL Server Migration Assistant available, depending on your source database. Download the current version of [SQL Server Migration Assistant for Oracle](#) and install it using the instructions found on the download page.

NOTE

At this time, the **SSMA for Oracle Extension Pack** is not supported on Linux, but it is not necessary for this tutorial.

Create and set-up project

Use the following steps to create a new SSMA project:

1. Open SSMA for Oracle and choose **New Project** from the **File** menu.
2. Give the project a name.
3. Choose "SQL Server 2017 (Linux) - Preview" in the **Migrate To** field.

SSMA for Oracle does not use the Oracle sample schemas by default. To enable the HR schema, use the following steps:

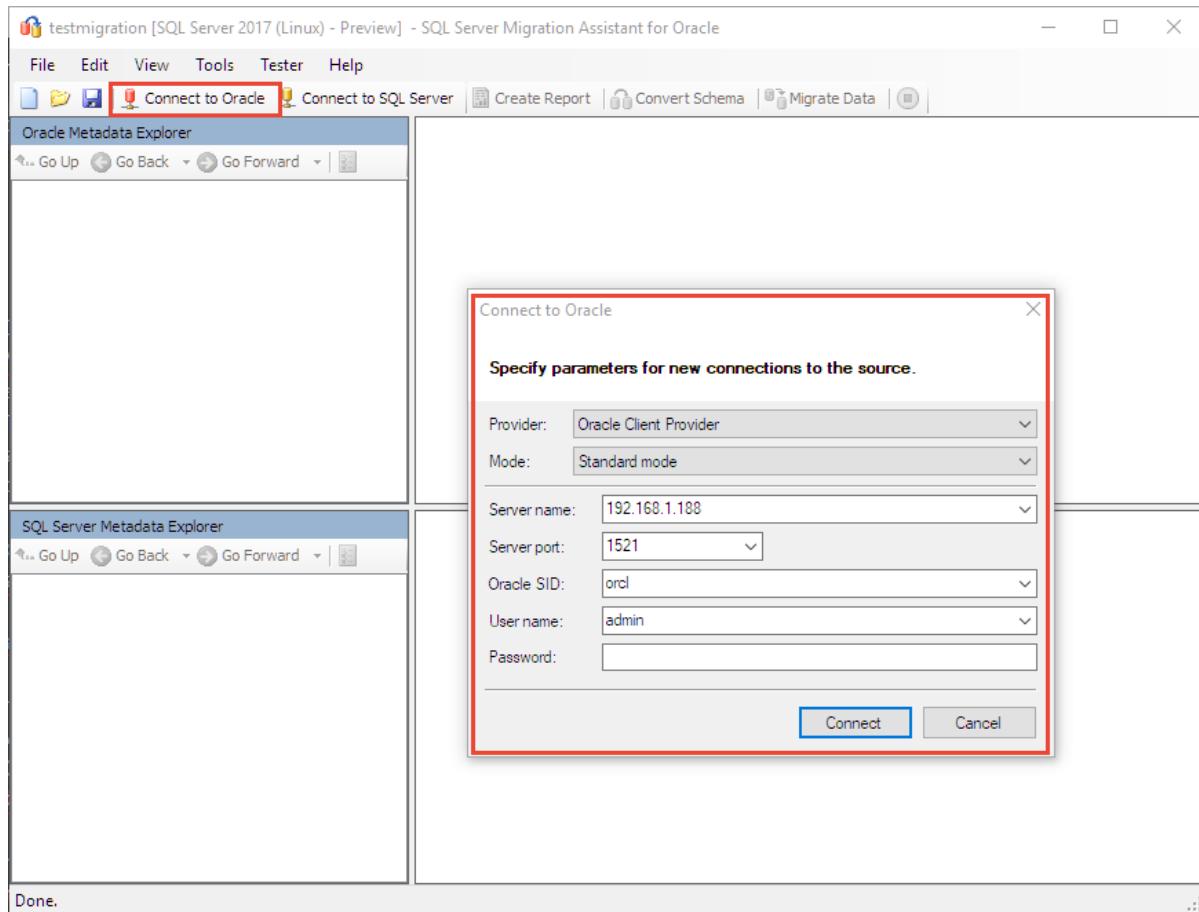
1. In SSMA, select the **Tools** menu.
2. Select **Default Project Settings**, and then choose **Loading System Objects**.

3. Make sure **HR** is checked, and choose **OK**.

Connect to Oracle

Next connect SSMA to Oracle.

1. On the toolbar, click **Connect to Oracle**.
2. Enter the server name, port, Oracle SID, user name, and password.

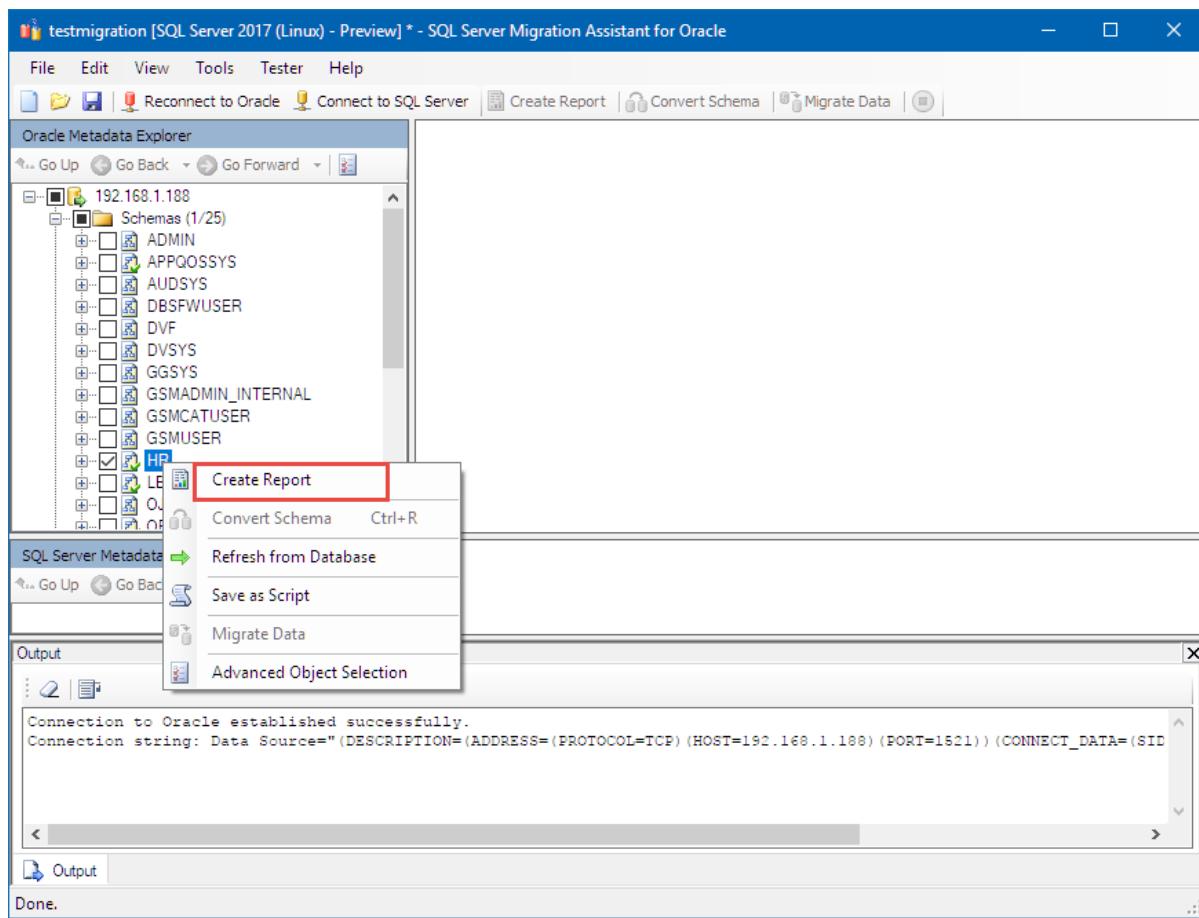


3. Then click **Connect**. In a few moments, SSMA for Oracle connects to your database and reads its metadata.

Create a report

Use the following steps to generate a migration report.

1. In the **Oracle Metadata Explorer**, expand your server's node.
2. Expand **Schemas**, right-click **HR**, and select **Create Report**.



3. A new browser window opens with a report that lists all of the warnings and errors associated with the conversion.

NOTE

You don't need to do anything with that list for this tutorial. If you perform these steps for your own Oracle database, you should review the report to address any important conversion problems for your database.

The screenshot shows the Microsoft SQL Server Migration Assistant interface. On the left, there's a tree view of the HR schema containing Procedures, Sequences, Tables, and Views. The main area displays two tables: 'Conversion statistics' and 'Objects by categories'. The 'Conversion statistics' table shows the count of statements converted and not converted across various types like ALL, argument, block-statement, etc. The 'Objects by categories' table shows the total number of objects and those with errors for categories like procedure, sequence, table, index, trigger, view, and schema. At the bottom, there are tabs for Errors, Warnings (5), and Info, along with a message about total estimated manual conversion time.

Statement Type	Total	Converted	Not converted
ALL	105	100 %	0
argument	5	100 %	0
block-statement	2	100 %	0
check-constraint	2	100 %	0
column	51	100 %	0
create-statement	5	100 %	0
foreign-key	10	100 %	0
if-statement	1	100 %	0
index	11	100 %	0
insert-statement	1	100 %	0
primary-key	7	100 %	0

Object type	Total	With errors
procedure	2	0
sequence	3	0
table	7	0
index	11	0
trigger	2	0
view	1	0
schema	1	0

Errors | Warnings (5) | Info Navigate by Errors Total estimated manual conversion time: 0 hr(s)

[x] O2SS0017: SQL Server Migration Assistant for Oracle Warning message: WITH READ ONLY(1)
[x] O2SS0356: SQL Server Migration Assistant for Oracle Warning message: Conversion from NUMBER datatype can cause data loss(4)

Connect to SQL Server

Next choose **Connect to SQL Server** and enter the appropriate connection information. If you use a database name that doesn't already exist, SSMA for Oracle creates it for you.

The screenshot shows the SQL Server Migration Assistant for Oracle interface. The 'Connect to SQL Server' dialog box is open in the foreground, prompting for connection details: Server name (192.168.1.205), Server port (default), Database (hr), Authentication (SQL Server Authentication), User name (admin), and Password. The 'Encrypt Connection' and 'Trust Server Certificate' checkboxes are checked. In the background, the Oracle Metadata Explorer shows the HR schema, and the SQL Server Metadata Explorer shows the hr database. The Output window at the bottom shows the conversion progress and completion message.

testmigration [SQL Server 2017 (Linux) - Preview] * - SQL Server Migration Assistant for Oracle

File Edit View Tools Tester Help

Reconnect to Oracle Connect to SQL Server Create Report Convert Schema Migrate Data

Oracle Metadata Explorer

HR

SQL Server Metadata Explorer

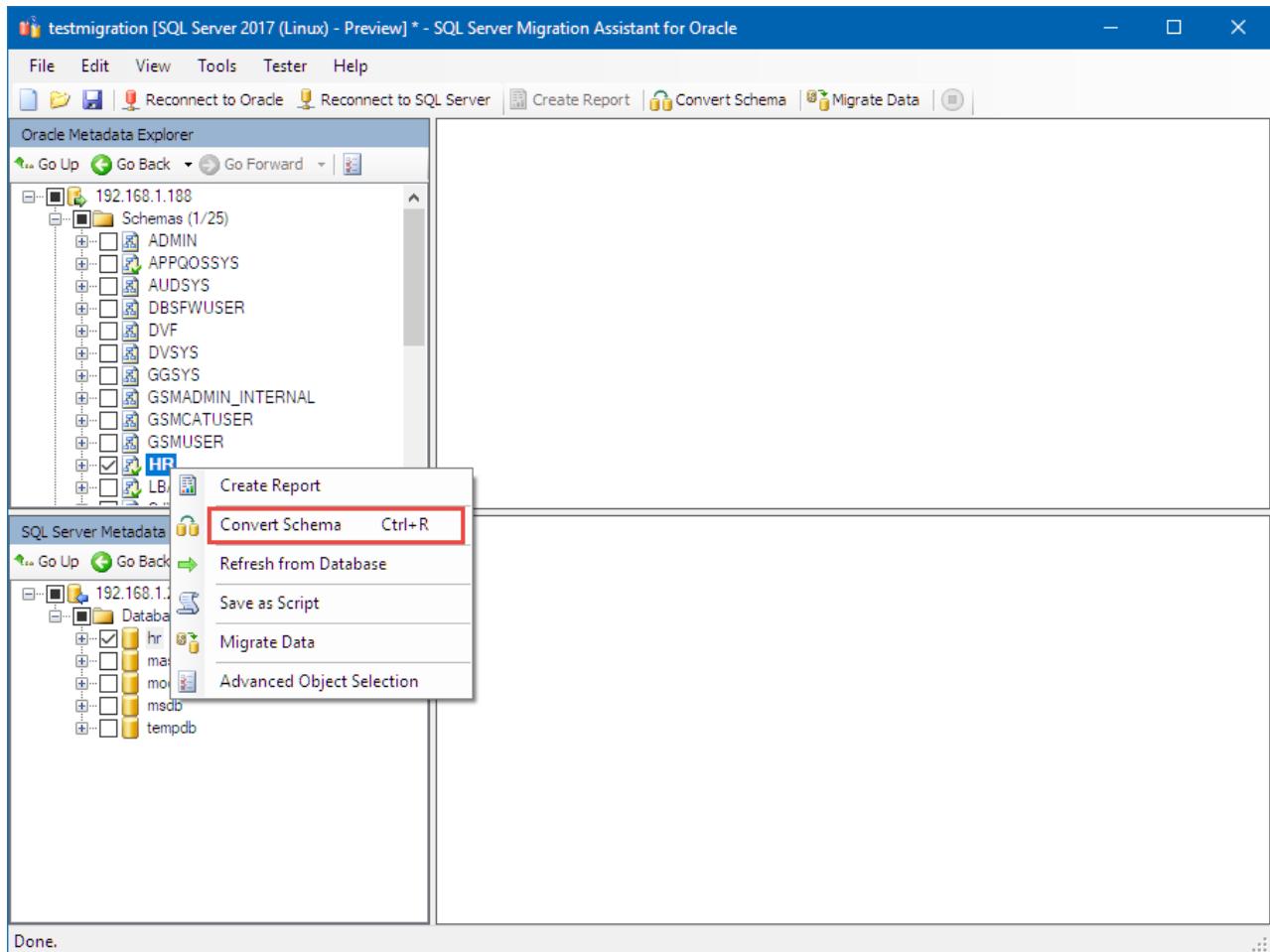
Output

Done.

Convert migration finished with 0 errors, 5 warnings, and 0 informational messages.

Convert Schema

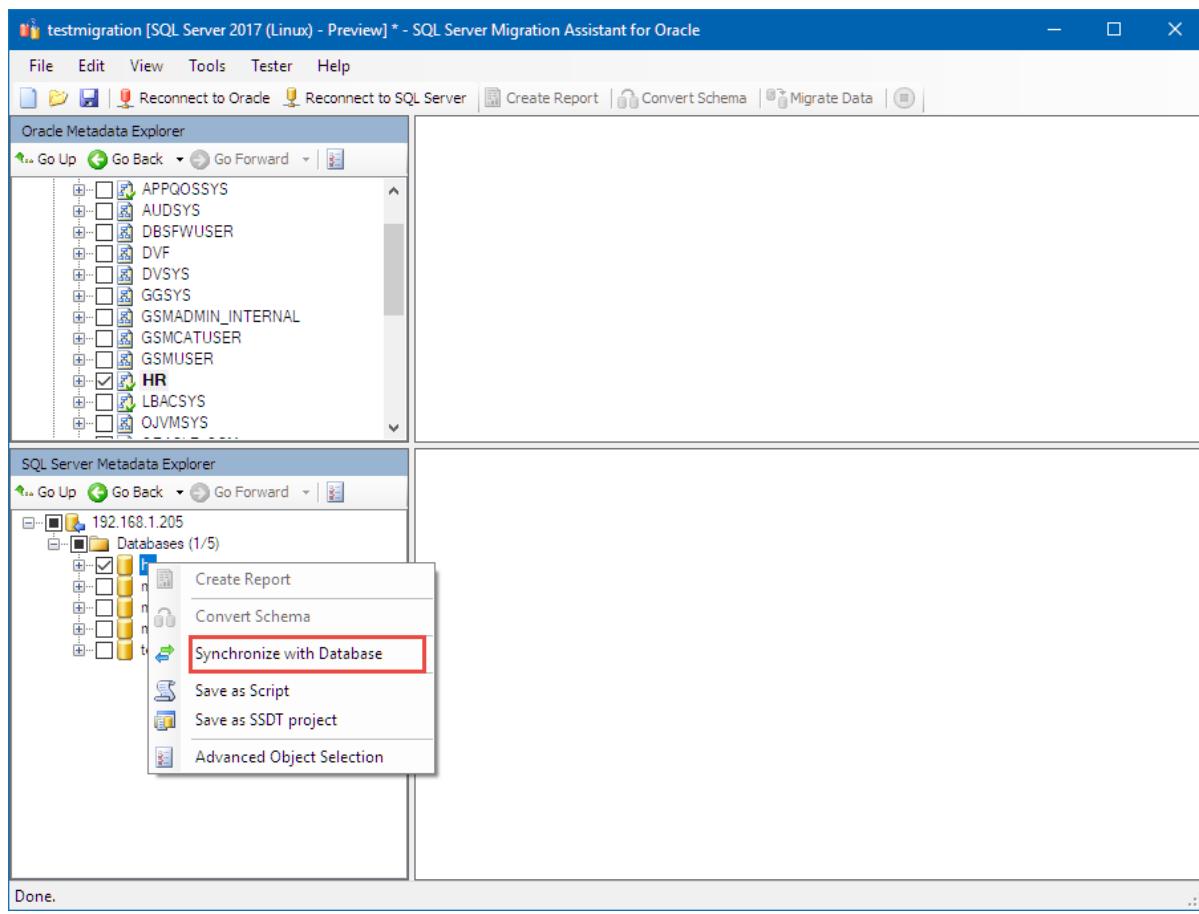
Right-click on **HR** in **Oracle Metadata Explorer**, and choose Convert Schema.



Synchronize Database

Next, synchronize your database.

1. Once the conversion is finished, use the **SQL Server Metadata Explorer** to go to the database you created in the previous step.
2. Right-click on your database, select **Synchronize with Database**, and then click OK.



Migrate data

The final step is to migrate your data.

1. In the **Oracle Metadata Explorer**, right-click on **HR**, and select **Migrate Data**.
2. The data migration step requires that you reenter your Oracle and SQL Server credentials.
3. When finished, review the data migration report, which should look similar to the following screenshot:

Status	From	To	Total Rows	Migrated Rows	Success Rate	Duration (DD:HH:MM:SS:MS)
Success	"HR"."COUNTRIES"	[hr].[HR].[COUNTRIES]	25	25	100.00 %	00:00:00:00:815
Success	"HR"."REGIONS"	[hr].[HR].[REGIONS]	4	4	100.00 %	00:00:00:00:525
Success	"HR"."LOCATIONS"	[hr].[HR].[LOCATIONS]	23	23	100.00 %	00:00:00:00:575
Success	"HR"."DEPARTMENTS"	[hr].[HR].[DEPARTMENT]	27	27	100.00 %	00:00:00:00:788
Success	"HR"."JOBS"	[hr].[HR].[JOBS]	19	19	100.00 %	00:00:00:00:645
Success	"HR"."EMPLOYEES"	[hr].[HR].[EMPLOYEES]	107	107	100.00 %	00:00:00:00:717
Success	"HR"."JOB_HISTORY"	[hr].[HR].[JOB_HISTORY]	10	10	100.00 %	00:00:00:00:685

Next steps

For a more complex Oracle schema, the conversion process would involve more time, testing, and possible changes to client applications. The purpose of this tutorial is to show how you can use SSMA for Oracle as a part of

your overall migration process.

In this tutorial, you learned how to:

- Install SSMA on Windows
- Create a new SSMA project
- Assess and run a migration from Oracle

Next, explore other ways to use SSMA:

[SQL Server Migration Assistant documentation](#)

Restore a SQL Server database in a Linux Docker container

2/14/2018 • 10 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Linux only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

This tutorial demonstrates how to move and restore a SQL Server backup file into a SQL Server 2017 Linux container image running on Docker.

- Pull and run the latest SQL Server 2017 Linux container image.
- Copy the World Wide Importers database file into the container.
- Restore the database in the container.
- Run Transact-SQL statements to view and modify the database.
- Backup the modified database.

Prerequisites

- Docker Engine 1.8+ on any supported Linux distribution or Docker for Mac/Windows. For more information, see [Install Docker](#).
- Minimum of 2 GB of disk space
- Minimum of 2 GB of RAM
- [System requirements for SQL Server on Linux](#).

Pull and run the container image

1. Open a bash terminal on Linux/Mac or an elevated PowerShell session on Windows.
2. Pull the SQL Server 2017 Linux container image from Docker Hub.

```
sudo docker pull microsoft/mssql-server-linux:2017-latest
```

```
docker pull microsoft/mssql-server-linux:2017-latest
```

TIP

Throughout this tutorial, docker command examples are given for both the bash shell (Linux/Mac) and PowerShell (Windows).

3. To run the container image with Docker, you can use the following command:

```
sudo docker run -e 'ACCEPT_EULA=Y' -e 'MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>' \
--name 'sql1' -p 1401:1433 \
-v sql1data:/var/opt/mssql \
-d microsoft/mssql-server-linux:2017-latest
```

```
docker run -e "ACCEPT_EULA=Y" -e "MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>" `  
    --name "sql1" -p 1401:1433 `  
    -v sql1data:/var/opt/mssql `  
    -d microsoft/mssql-server-linux:2017-latest
```

This command creates a SQL Server 2017 container with the Developer edition (default). SQL Server port **1433** is exposed on the host as port **1401**. The optional `-v sql1data:/var/opt/mssql` parameter creates a data volume container named **sql1data**. This is used to persist the data created by SQL Server.

NOTE

The process for running production SQL Server editions in containers is slightly different. For more information, see [Run production container images](#). If you use the same container names and ports, the rest of this walkthrough still works with production containers.

4. To view your Docker containers, use the `docker ps` command.

```
sudo docker ps -a
```

```
docker ps -a
```

5. If the **STATUS** column shows a status of **Up**, then SQL Server is running in the container and listening on the port specified in the **PORTS** column. If the **STATUS** column for your SQL Server container shows **Exited**, see the [Troubleshooting section of the configuration guide](#).

```
$ sudo docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
PORTS	NAMES			
941e1bdf8e1d	microsoft/mssql-server-linux	/bin/sh -c /opt/m..."	About an hour ago	Up
About an hour	0.0.0.0:1401->1433/tcp	sql1		

Change the SA password

The **SA** account is a system administrator on the SQL Server instance that gets created during setup. After creating your SQL Server container, the `MSSQL_SA_PASSWORD` environment variable you specified is discoverable by running `echo $MSSQL_SA_PASSWORD` in the container. For security purposes, change your SA password.

1. Choose a strong password to use for the SA user.
2. Use `docker exec` to run **sqlcmd** to change the password using Transact-SQL. Replace `<YourStrong!Passw0rd>` and `<YourNewStrong!Passw0rd>` with your own password values.

```
sudo docker exec -it sql1 /opt/mssql-tools/bin/sqlcmd \  
    -S localhost -U SA -P '<YourStrong!Passw0rd>' \  
    -Q 'ALTER LOGIN SA WITH PASSWORD="<YourNewStrong!Passw0rd>"'
```

```
docker exec -it sql1 /opt/mssql-tools/bin/sqlcmd \  
    -S localhost -U SA -P "<YourStrong!Passw0rd>" \  
    -Q "ALTER LOGIN SA WITH PASSWORD='<YourNewStrong!Passw0rd>'"
```

Copy a backup file into the container

This tutorial uses the [Wide World Importers sample database](#). Use the following steps to download and copy the Wide World Importers database backup file into your SQL Server container.

1. First, use **docker exec** to create a backup folder. The following command creates a **/var/opt/mssql/backup** directory inside the SQL Server container.

```
sudo docker exec -it sql1 mkdir /var/opt/mssql/backup
```

```
docker exec -it sql1 mkdir /var/opt/mssql/backup
```

2. Next, download the [WideWorldImporters-Full.bak](#) file to your host machine. The following commands navigate to the home/user directory and downloads the backup file as **wwi.bak**.

```
cd ~  
curl -L -o wwi.bak 'https://github.com/Microsoft/sql-server-samples/releases/download/wide-world-importers-v1.0/WideWorldImporters-Full.bak'
```

```
curl -OutFile "wwi.bak" "https://github.com/Microsoft/sql-server-samples/releases/download/wide-world-importers-v1.0/WideWorldImporters-Full.bak"
```

3. Use **docker cp** to copy the backup file into the container in the **/var/opt/mssql/backup** directory.

```
sudo docker cp wwi.bak sql1:/var/opt/mssql/backup
```

```
docker cp wwi.bak sql1:/var/opt/mssql/backup
```

Restore the database

The backup file is now located inside the container. Before restoring the backup, it is important to know the logical file names and file types inside the backup. The following Transact-SQL commands inspect the backup and perform the restore using **sqlcmd** in the container.

TIP

This tutorial uses **sqlcmd** inside the container, because the container comes with this tool pre-installed. However, you can also run Transact-SQL statements with other client tools outside of the container, such as [Visual Studio Code](#) or [SQL Server Management Studio](#). To connect, use the host port that was mapped to port 1433 in the container. In this example, that is **localhost,1401** on the host machine and **Host_IP_Address,1401** remotely.

1. Run **sqlcmd** inside the container to list out logical file names and paths inside the backup. This is done with the **RESTORE FILELISTONLY** Transact-SQL statement.

```
sudo docker exec -it sql1 /opt/mssql-tools/bin/sqlcmd -S localhost \  
-U SA -P '<YourNewStrong!Passw0rd>' \  
-Q 'RESTORE FILELISTONLY FROM DISK = "/var/opt/mssql/backup/wwi.bak"' \  
| tr -s ' ' | cut -d ' ' -f 1-2
```

```
docker exec -it sql1 /opt/mssql-tools/bin/sqlcmd -S localhost `  
-U SA -P "<YourNewStrong!Passw0rd>"`  
-Q "RESTORE FILELISTONLY FROM DISK = '/var/opt/mssql/backup/wwi.bak'"
```

You should see output similar to the following:

LogicalName	PhysicalName
WWI_Primary	D:\Data\WideWorldImporters.mdf
WWI_UserData	D:\Data\WideWorldImporters_UserData.ndf
WWI_Log	E:\Log\WideWorldImporters.ldf
WWI_InMemory_Data_1	D:\Data\WideWorldImporters_InMemory_Data_1

- Call the **RESTORE DATABASE** command to restore the database inside the container. Specify new paths for each of the files in the previous step.

```
sudo docker exec -it sql1 /opt/mssql-tools/bin/sqlcmd \  
-S localhost -U SA -P '<YourNewStrong!Passw0rd>' \  
-Q "RESTORE DATABASE WideWorldImporters FROM DISK = '/var/opt/mssql/backup/wwi.bak' WITH MOVE  
'WWI_Primary' TO '/var/opt/mssql/data/WideWorldImporters.mdf', MOVE 'WWI_UserData' TO  
'/var/opt/mssql/data/WideWorldImporters_userdata.ndf', MOVE 'WWI_Log' TO  
'/var/opt/mssql/data/WideWorldImporters.ldf', MOVE 'WWI_InMemory_Data_1' TO  
'/var/opt/mssql/data/WideWorldImporters_InMemory_Data_1'"
```

```
docker exec -it sql1 /opt/mssql-tools/bin/sqlcmd `  
-S localhost -U SA -P "<YourNewStrong!Passw0rd>"`  
-Q "RESTORE DATABASE WideWorldImporters FROM DISK = '/var/opt/mssql/backup/wwi.bak' WITH MOVE  
'WWI_Primary' TO '/var/opt/mssql/data/WideWorldImporters.mdf', MOVE 'WWI_UserData' TO  
'/var/opt/mssql/data/WideWorldImporters_userdata.ndf', MOVE 'WWI_Log' TO  
'/var/opt/mssql/data/WideWorldImporters.ldf', MOVE 'WWI_InMemory_Data_1' TO  
'/var/opt/mssql/data/WideWorldImporters_InMemory_Data_1'"
```

You should see output similar to the following:

```
Processed 1464 pages for database 'WideWorldImporters', file 'WWI_Primary' on file 1.  
Processed 53096 pages for database 'WideWorldImporters', file 'WWI_UserData' on file 1.  
Processed 33 pages for database 'WideWorldImporters', file 'WWI_Log' on file 1.  
Processed 3862 pages for database 'WideWorldImporters', file 'WWI_InMemory_Data_1' on file 1.  
Converting database 'WideWorldImporters' from version 852 to the current version 869.  
Database 'WideWorldImporters' running the upgrade step from version 852 to version 853.  
Database 'WideWorldImporters' running the upgrade step from version 853 to version 854.  
Database 'WideWorldImporters' running the upgrade step from version 854 to version 855.  
Database 'WideWorldImporters' running the upgrade step from version 855 to version 856.  
Database 'WideWorldImporters' running the upgrade step from version 856 to version 857.  
Database 'WideWorldImporters' running the upgrade step from version 857 to version 858.  
Database 'WideWorldImporters' running the upgrade step from version 858 to version 859.  
Database 'WideWorldImporters' running the upgrade step from version 859 to version 860.  
Database 'WideWorldImporters' running the upgrade step from version 860 to version 861.  
Database 'WideWorldImporters' running the upgrade step from version 861 to version 862.  
Database 'WideWorldImporters' running the upgrade step from version 862 to version 863.  
Database 'WideWorldImporters' running the upgrade step from version 863 to version 864.  
Database 'WideWorldImporters' running the upgrade step from version 864 to version 865.  
Database 'WideWorldImporters' running the upgrade step from version 865 to version 866.  
Database 'WideWorldImporters' running the upgrade step from version 866 to version 867.  
Database 'WideWorldImporters' running the upgrade step from version 867 to version 868.  
Database 'WideWorldImporters' running the upgrade step from version 868 to version 869.  
RESTORE DATABASE successfully processed 58455 pages in 18.069 seconds (25.273 MB/sec).
```

Verify the restored database

Run the following query to display a list of database names in your container:

```
sudo docker exec -it sql1 /opt/mssql-tools/bin/sqlcmd \
-S localhost -U SA -P '<YourNewStrong!Passw0rd>' \
-Q 'SELECT Name FROM sys.Databases'
```

```
docker exec -it sql1 /opt/mssql-tools/bin/sqlcmd \
-S localhost -U SA -P "<YourNewStrong!Passw0rd>" \
-Q "SELECT Name FROM sys.Databases"
```

You should see **WideWorldImporters** in the list of databases.

Make a change

The following steps make a change in the database.

1. Run a query to view the top 10 items in the **Warehouse.StockItems** table.

```
sudo docker exec -it sql1 /opt/mssql-tools/bin/sqlcmd \
-S localhost -U SA -P '<YourNewStrong!Passw0rd>' \
-Q 'SELECT TOP 10 StockItemID, StockItemName FROM WideWorldImporters.Warehouse.StockItems ORDER BY StockItemID'
```

```
docker exec -it sql1 /opt/mssql-tools/bin/sqlcmd \
-S localhost -U SA -P "<YourNewStrong!Passw0rd>" \
-Q "SELECT TOP 10 StockItemID, StockItemName FROM WideWorldImporters.Warehouse.StockItems ORDER BY StockItemID"
```

You should see a list of item identifiers and names:

StockItemID	StockItemName
1	USB missile launcher (Green)
2	USB rocket launcher (Gray)
3	Office cube periscope (Black)
4	USB food flash drive - sushi roll
5	USB food flash drive - hamburger
6	USB food flash drive - hot dog
7	USB food flash drive - pizza slice
8	USB food flash drive - dim sum 10 drive variety pack
9	USB food flash drive - banana
10	USB food flash drive - chocolate bar

2. Update the description of the first item with the following **UPDATE** statement:

```
sudo docker exec -it sql1 /opt/mssql-tools/bin/sqlcmd \
-S localhost -U SA -P '<YourNewStrong!Passw0rd>' \
-Q 'UPDATE WideWorldImporters.Warehouse.StockItems SET StockItemName="USB missile launcher (Dark Green)" WHERE StockItemID=1; SELECT StockItemID, StockItemName FROM WideWorldImporters.Warehouse.StockItems WHERE StockItemID=1'
```

```
docker exec -it sql1 /opt/mssql-tools/bin/sqlcmd `  
-S localhost -U SA -P "<YourNewStrong!Passw0rd>"`  
-Q "UPDATE WideWorldImporters.Warehouse.StockItems SET StockItemName='USB missile launcher (Dark  
Green)' WHERE StockItemID=1; SELECT StockItemID, StockItemName FROM  
WideWorldImporters.Warehouse.StockItems WHERE StockItemID=1"
```

You should see output similar to the following text:

```
(1 rows affected)  
StockItemID StockItemName  
-----  
1 USB missile launcher (Dark Green)
```

Create a new backup

After you've restored your database into a container, you might also want to regularly create database backups inside the running container. The steps follow a similar pattern to the previous steps but in reverse.

1. Use the **BACKUP DATABASE** Transact-SQL command to create a database backup in the container. This tutorial creates a new backup file, **wwi_2.bak**, in the previously created **/var/opt/mssql/backup** directory.

```
sudo docker exec -it sql1 /opt/mssql-tools/bin/sqlcmd \  
-S localhost -U SA -P '<YourNewStrong!Passw0rd>' \  
-Q "BACKUP DATABASE [WideWorldImporters] TO DISK = N'/var/opt/mssql/backup/wwi_2.bak' WITH NOFORMAT,  
NOUNINIT, NAME = 'WideWorldImporters-full', SKIP, NOREWIND, NOUNLOAD, STATS = 10"
```

```
docker exec -it sql1 /opt/mssql-tools/bin/sqlcmd `  
-S localhost -U SA -P "<YourNewStrong!Passw0rd>"`  
-Q "BACKUP DATABASE [WideWorldImporters] TO DISK = N'/var/opt/mssql/backup/wwi_2.bak' WITH NOFORMAT,  
NOUNINIT, NAME = 'WideWorldImporters-full', SKIP, NOREWIND, NOUNLOAD, STATS = 10"
```

You should see output similar to the following:

```
10 percent processed.  
20 percent processed.  
30 percent processed.  
40 percent processed.  
50 percent processed.  
60 percent processed.  
70 percent processed.  
Processed 1200 pages for database 'WideWorldImporters', file 'WWI_Primary' on file 1.  
Processed 53096 pages for database 'WideWorldImporters', file 'WWI_UserData' on file 1.  
80 percent processed.  
Processed 3865 pages for database 'WideWorldImporters', file 'WWI_InMemory_Data_1' on file 1.  
Processed 938 pages for database 'WideWorldImporters', file 'WWI_Log' on file 1.  
100 percent processed.  
BACKUP DATABASE successfully processed 59099 pages in 25.056 seconds (18.427 MB/sec).
```

2. Next, copy the backup file out of the container and onto your host machine.

```
cd ~  
sudo docker cp sql1:/var/opt/mssql/backup/wwi_2.bak wwi_2.bak  
ls -l wwi*
```

```
cd ~  
docker cp sql1:/var/opt/mssql/backup/wwi_2.bak wwi_2.bak  
ls -l wwi*
```

Use the persisted data

In addition to taking database backups for protecting your data, you can also use data volume containers. The beginning of this tutorial created the **sql1** container with the `-v sql1data:/var/opt/mssql` parameter. The **sql1data** data volume container persists the `/var/opt/mssql` data even after the container is removed. The following steps completely remove the **sql1** container and then create a new container, **sql2**, with the persisted data.

1. Stop the **sql1** container.

```
sudo docker stop sql1
```

```
docker stop sql1
```

2. Remove the container. This does not delete the previously created **sql1data** data volume container and the persisted data in it.

```
sudo docker rm sql1
```

```
docker rm sql1
```

3. Create a new container, **sql2**, and reuse the **sql1data** data volume container.

```
sudo docker run -e 'ACCEPT_EULA=Y' -e 'MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>' \  
--name 'sql2' -e 'MSSQL_PID=Developer' -p 1401:1433 \  
-v sql1data:/var/opt/mssql -d microsoft/mssql-server-linux:2017-latest
```

```
docker run -e "ACCEPT_EULA=Y" -e "MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>" \  
--name "sql2" -e "MSSQL_PID=Developer" -p 1401:1433 \  
-v sql1data:/var/opt/mssql -d microsoft/mssql-server-linux:2017-latest
```

4. The Wide World Importers database is now in the new container. Run a query to verify the previous change you made.

```
sudo docker exec -it sql2 /opt/mssql-tools/bin/sqlcmd \  
-S localhost -U SA -P '<YourNewStrong!Passw0rd>' \  
-Q 'SELECT StockItemID, StockItemName FROM WideWorldImporters.Warehouse.StockItems WHERE  
StockItemID=1'
```

```
docker exec -it sql2 /opt/mssql-tools/bin/sqlcmd \  
-S localhost -U SA -P "<YourNewStrong!Passw0rd>" \  
-Q "SELECT StockItemID, StockItemName FROM WideWorldImporters.Warehouse.StockItems WHERE  
StockItemID=1"
```

NOTE

The SA password is not the password you specified for the **sql2** container, `MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>`. All of the SQL Server data was restored from **sql1**, including the changed password from earlier in the tutorial. In effect, some options like this are ignored due to restoring the data in `/var/opt/mssql`. For this reason, the password is `<YourNewStrong!Passw0rd>` as shown here.

Next steps

In this tutorial, you learned how to back up a database on Windows and move it to a Linux server running SQL Server 2017 RC2. You learned how to:

- Create SQL Server 2017 Linux container images.
- Copy SQL Server database backups into a container.
- Run Transact-SQL statements inside the container with **sqlcmd**.
- Create and extract backup files from a container.
- Use data volume containers in Docker to persist SQL Server data.

Next, review other Docker configuration and troubleshooting scenarios:

[Configuration guide for SQL Server 2017 on Docker](#)

Create and run SQL Server Agent jobs on Linux

2/21/2018 • 3 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

SQL Server jobs are used to regularly perform the same sequence of commands in your SQL Server database. This tutorial provides an example of how to create a SQL Server Agent job on Linux using both Transact-SQL and SQL Server Management Studio (SSMS).

- Install SQL Server Agent on Linux
- Create a new job to perform daily database backups
- Schedule and run the job
- Perform the same steps in SSMS (optional)

For known issues with SQL Server Agent on Linux, see the [Release Notes](#).

Prerequisites

The following prerequisites are required to complete this tutorial:

- Linux machine with the following prerequisites:
 - SQL Server 2017 ([RHEL](#), [SLES](#), or [Ubuntu](#)) with command-line tools.

The following prerequisites are optional:

- Windows machine with SSMS:
 - [SQL Server Management Studio](#) for optional SSMS steps.

Enable SQL Server Agent

To use SQL Server Agent on Linux, you must first enable SQL Server Agent on a machine that already has SQL Server 2017 installed.

1. To enable SQL Server Agent, follow the step below.

```
sudo /opt/mssql/bin/mssql-conf set sqlagent.enabled true
```

2. Restart SQL Server with the following command:

```
sudo systemctl restart mssql-server
```

NOTE

Starting with SQL Server 2017 CU4, SQL Server Agent is included with the **mssql-server** package and is disabled by default. For Agent set up prior to CU4 visit, [Install SQL Server Agent on Linux](#).

Create a sample database

Use the following steps to create a sample database named **SampleDB**. This database is used for the daily backup

job.

1. On your Linux machine, open a bash terminal session.
2. Use **sqlcmd** to run a Transact-SQL **CREATE DATABASE** command.

```
/opt/mssql-tools/bin/sqlcmd -S localhost -U SA -Q 'CREATE DATABASE SampleDB'
```

3. Verify the database is created by listing the databases on your server.

```
/opt/mssql-tools/bin/sqlcmd -S localhost -U SA -Q 'SELECT Name FROM sys.Databases'
```

Create a job with Transact-SQL

The following steps create a SQL Server Agent job on Linux with Transact-SQL commands. The job runs a daily backup of the sample database, **SampleDB**.

TIP

You can use any T-SQL client to run these commands. For example, on Linux you can use [sqlcmd](#) or [Visual Studio Code](#). From a remote Windows Server, you can also run queries in SQL Server Management Studio (SSMS) or use the UI interface for job management, which is described in the next section.

1. Use **sp_add_job** to create a job named `Daily SampleDB Backup`.

```
-- Adds a new job executed by the SQLServerAgent service
-- called 'Daily SampleDB Backup'
USE msdb ;
GO
EXEC dbo.sp_add_job
    @job_name = N'Daily SampleDB Backup' ;
GO
```

2. Call **sp_add_jobstep** to create a job step that creates a backup of the `SampleDB` database.

```
-- Adds a step (operation) to the job
EXEC sp_add_jobstep
    @job_name = N'Daily SampleDB Backup',
    @step_name = N'Backup database',
    @subsystem = N'TSQL',
    @command = N'BACKUP DATABASE SampleDB TO DISK = \
        N''/var/opt/mssql/data/SampleDB.bak'' WITH NOFORMAT, NOINIT, \
        NAME = ''SampleDB-full'', SKIP, NOREWIND, NOUNLOAD, STATS = 10',
    @retry_attempts = 5,
    @retry_interval = 5 ;
GO
```

3. Then create a daily schedule for your job with **sp_add_schedule**.

```
-- Creates a schedule called 'Daily'  
EXEC dbo.sp_add_schedule  
    @schedule_name = N'Daily SampleDB',  
    @freq_type = 4,  
    @freq_interval = 1,  
    @active_start_time = 233000 ;  
USE msdb ;  
GO
```

4. Attach the job schedule to the job with [sp_attach_schedule](#).

```
-- Sets the 'Daily' schedule to the 'Daily SampleDB Backup' Job  
EXEC sp_attach_schedule  
    @job_name = N'Daily SampleDB Backup',  
    @schedule_name = N'Daily SampleDB';  
GO
```

5. Use [sp_add_jobserver](#) to assign the job to a target server. In this example, the target is the local server.

```
EXEC dbo.sp_add_jobserver  
    @job_name = N'Daily SampleDB Backup',  
    @server_name = N'(LOCAL)';  
GO
```

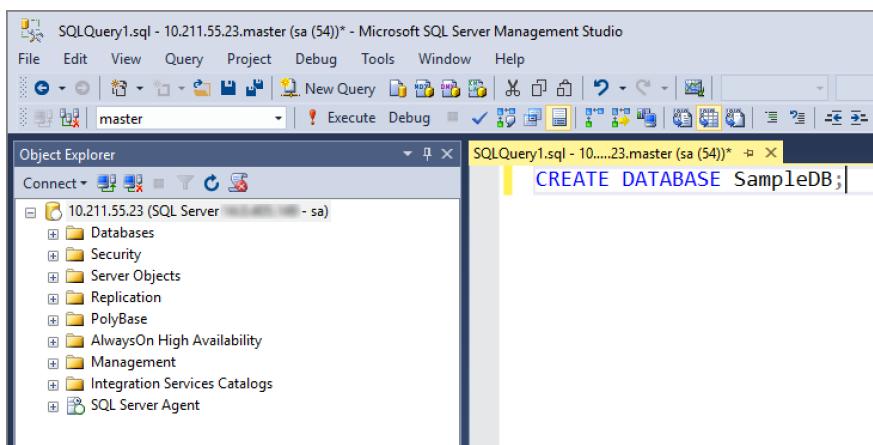
6. Start the job with [sp_start_job](#).

```
EXEC dbo.sp_start_job N'Daily SampleDB Backup';  
GO
```

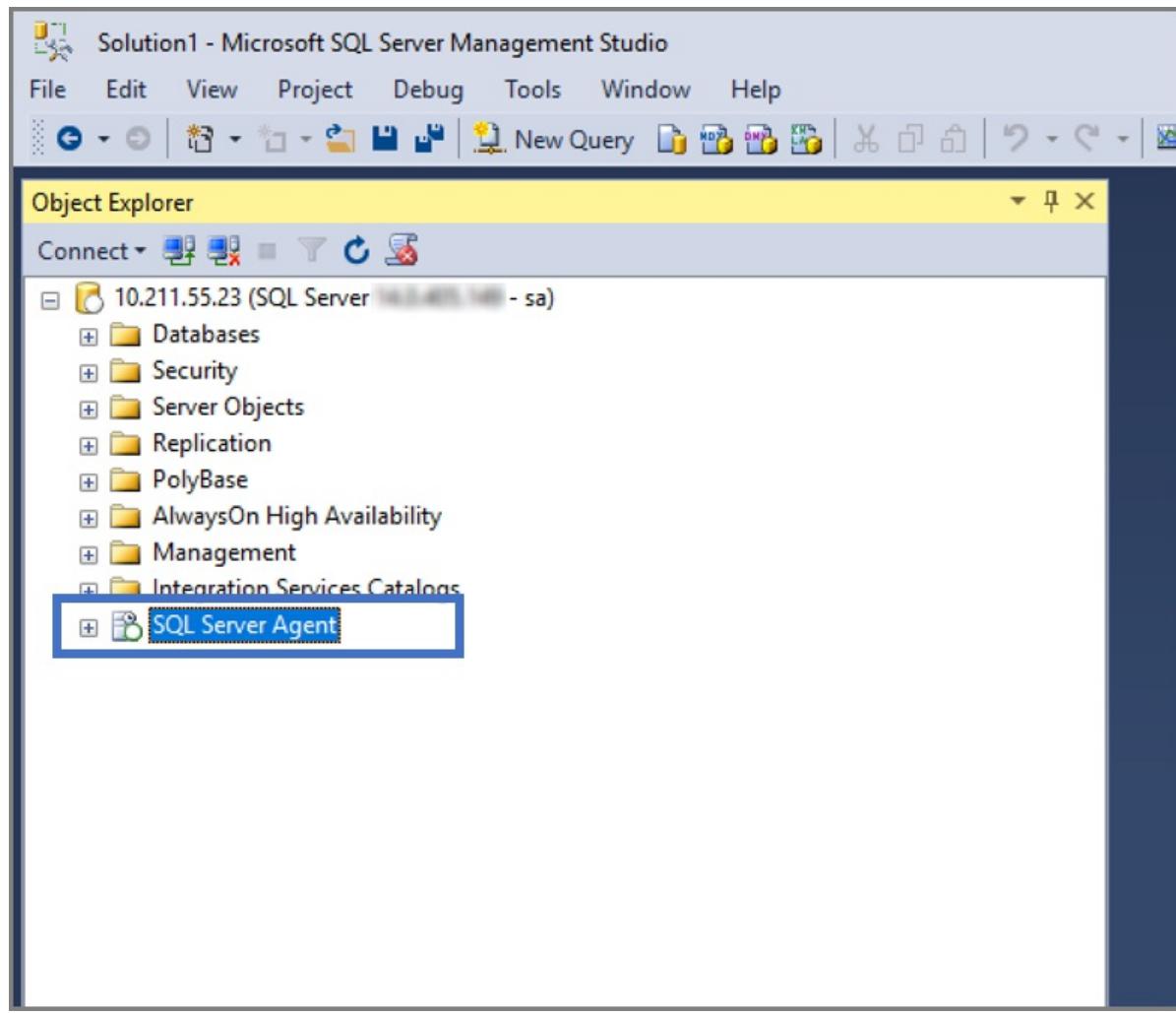
Create a job with SSMS

You can also create and manage jobs remotely using SQL Server Management Studio (SSMS) on Windows.

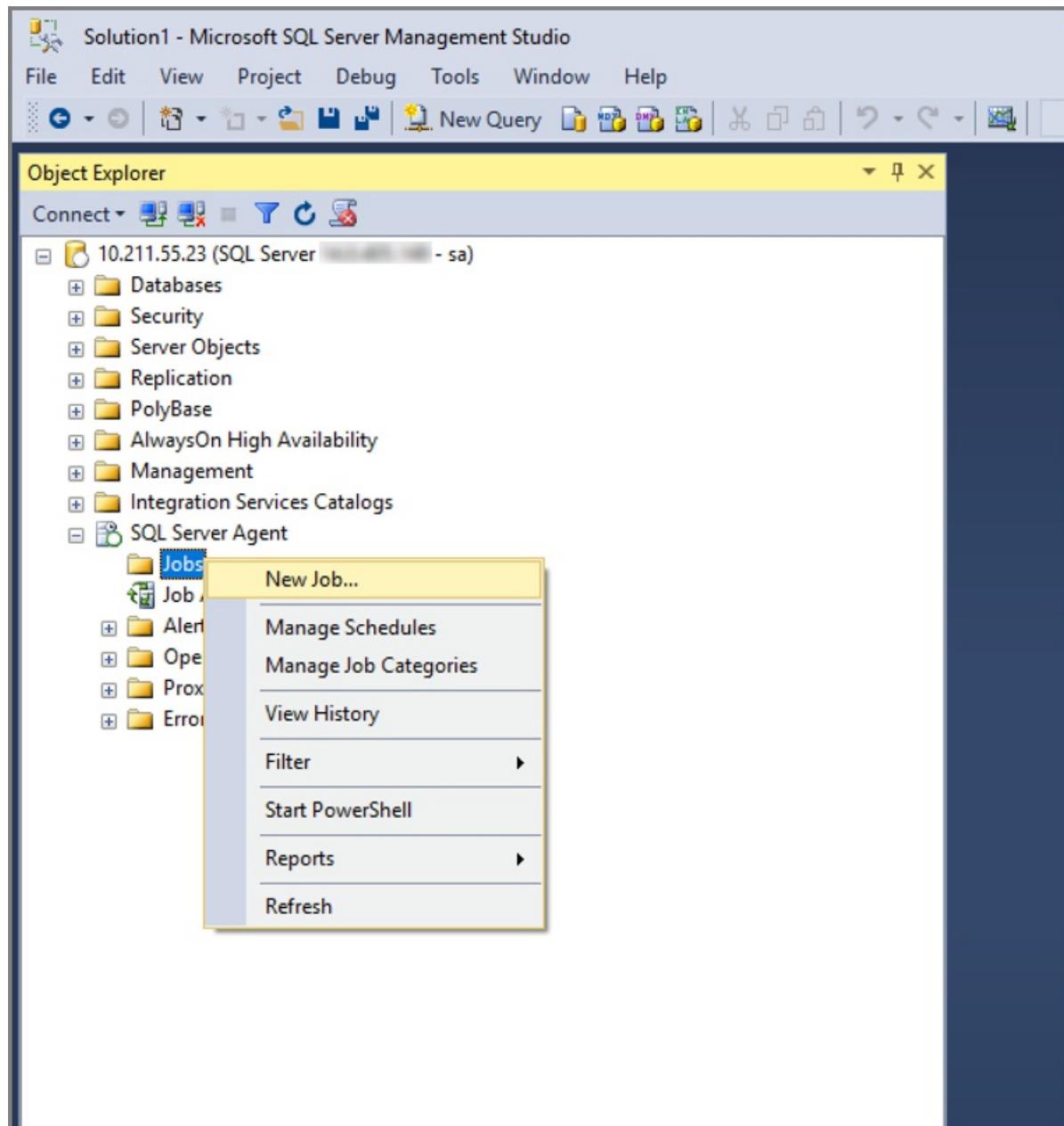
1. Start SSMS on Windows and connect to your Linux SQL Server instance. For more information, see [Manage SQL Server on Linux with SSMS](#).
2. Verify that you have created a sample database named **SampleDB**.



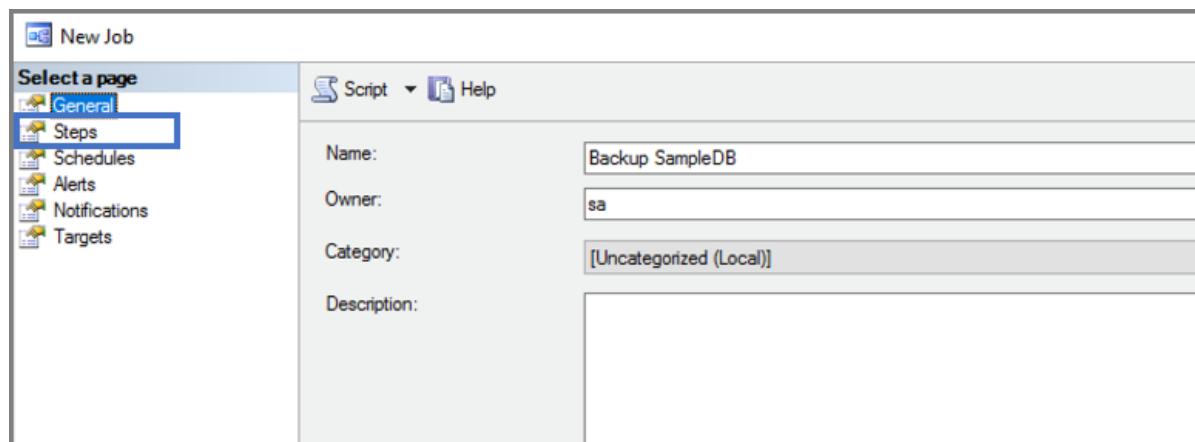
3. Verify that SQL Agent was [installed](#) and configured correctly. Look for the plus sign next to SQL Server Agent in the Object Explorer. If SQL Server Agent is not enabled, try restarting the **mssql-server** service on Linux.



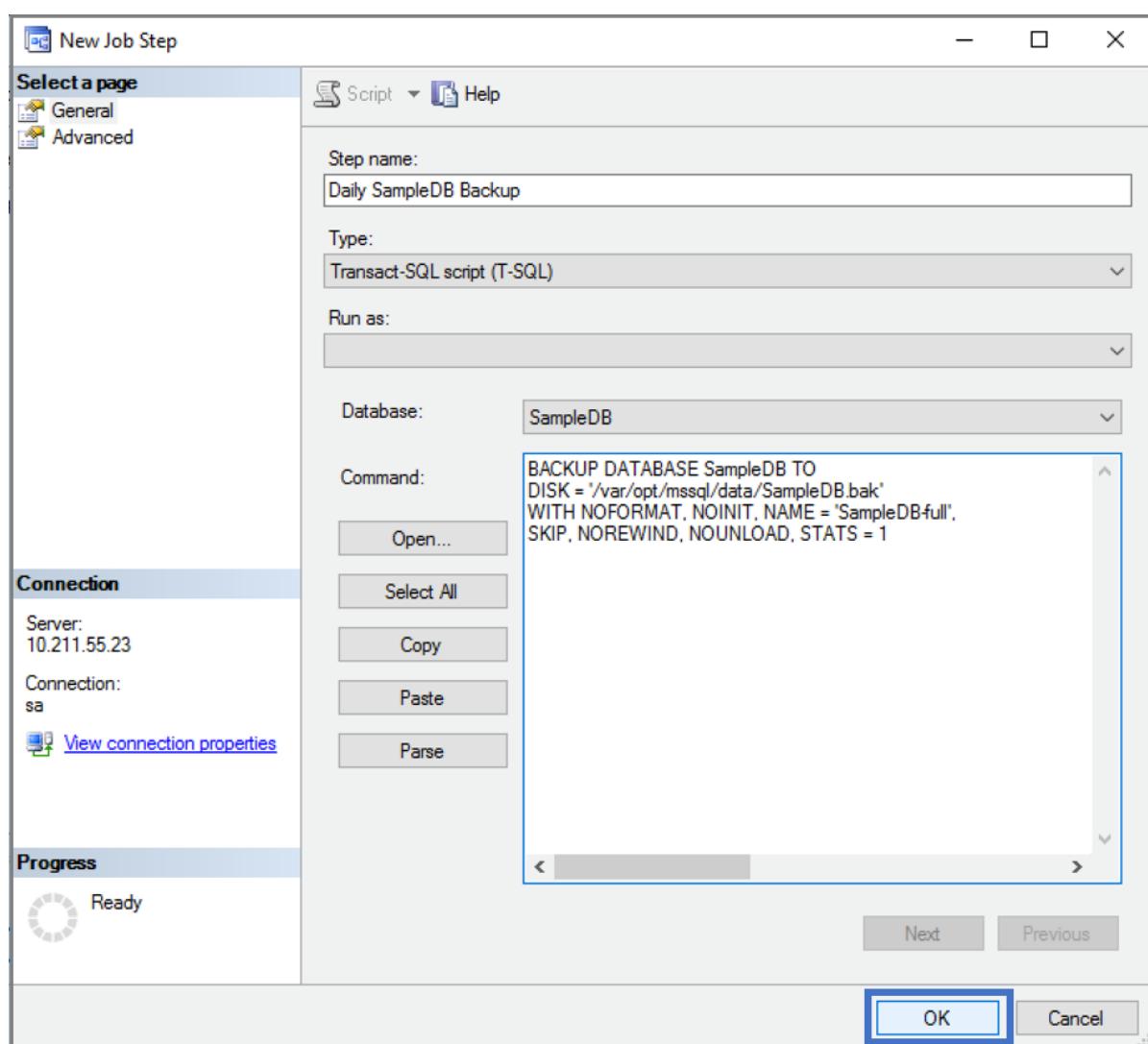
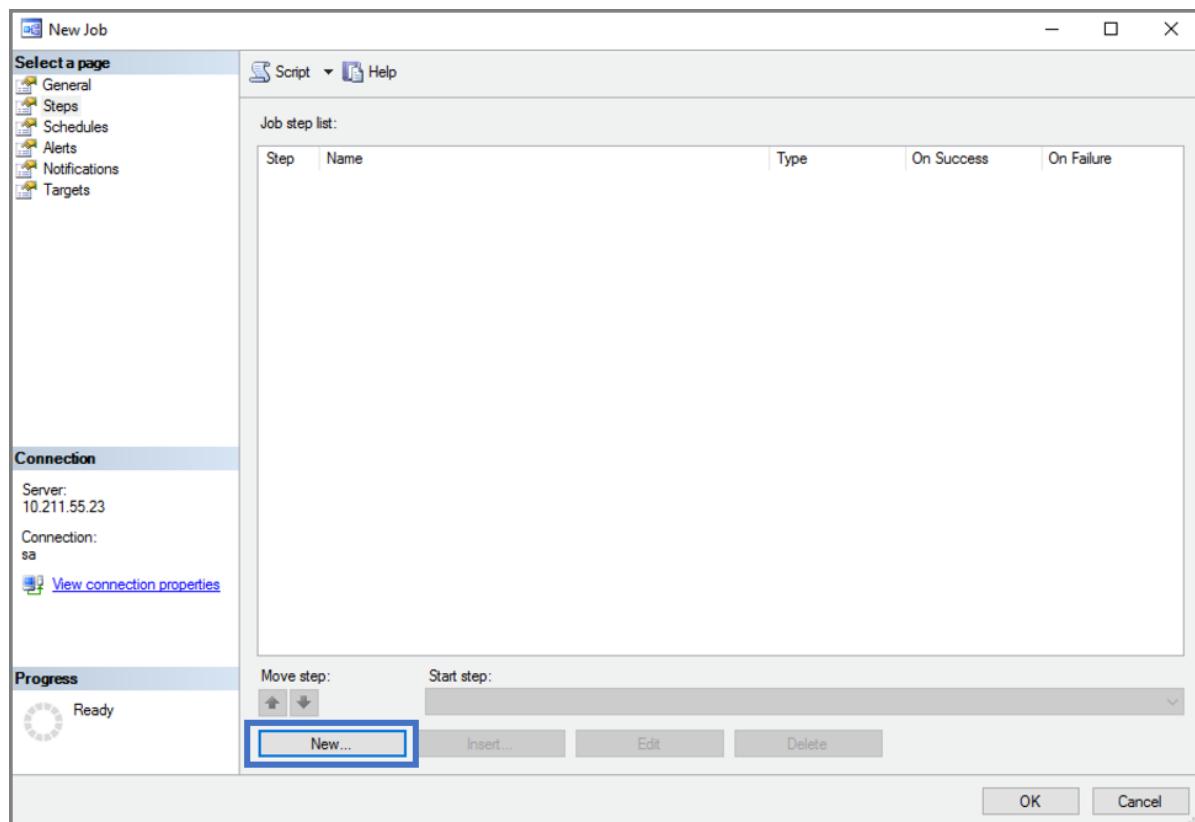
4. Create a new job.



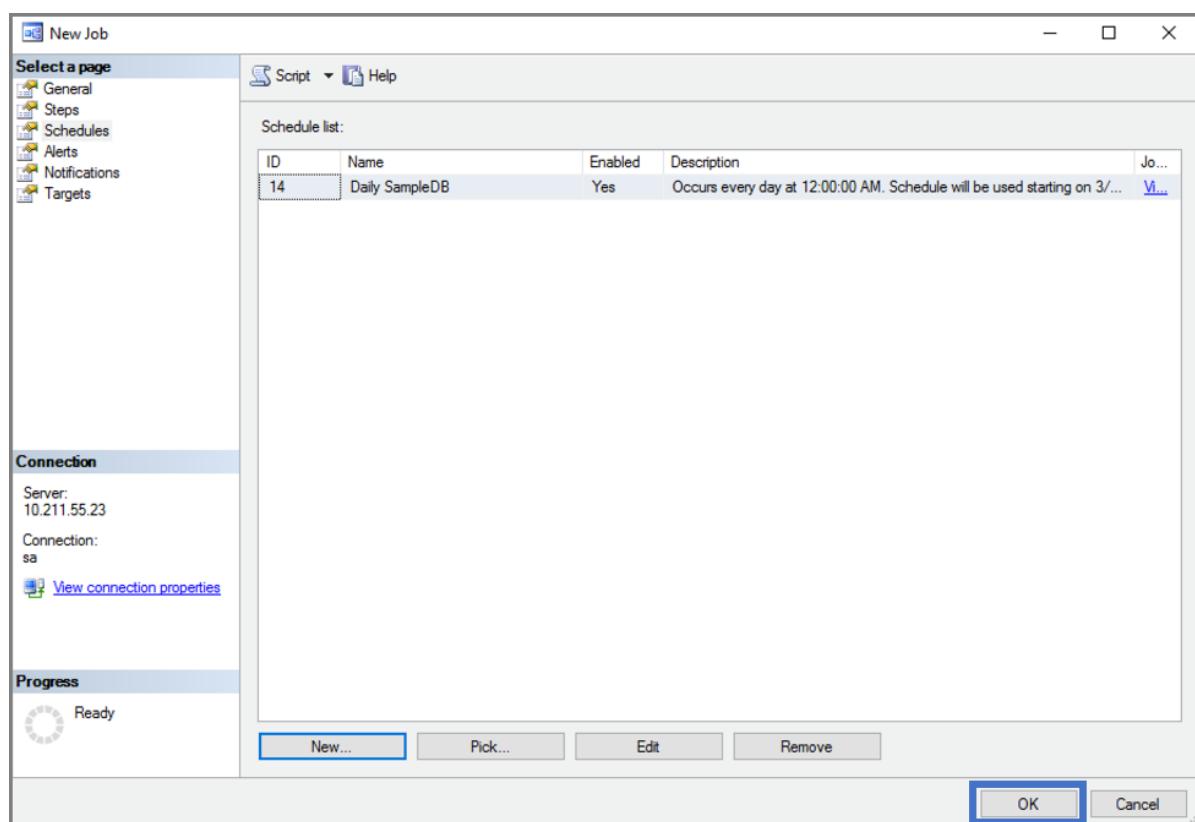
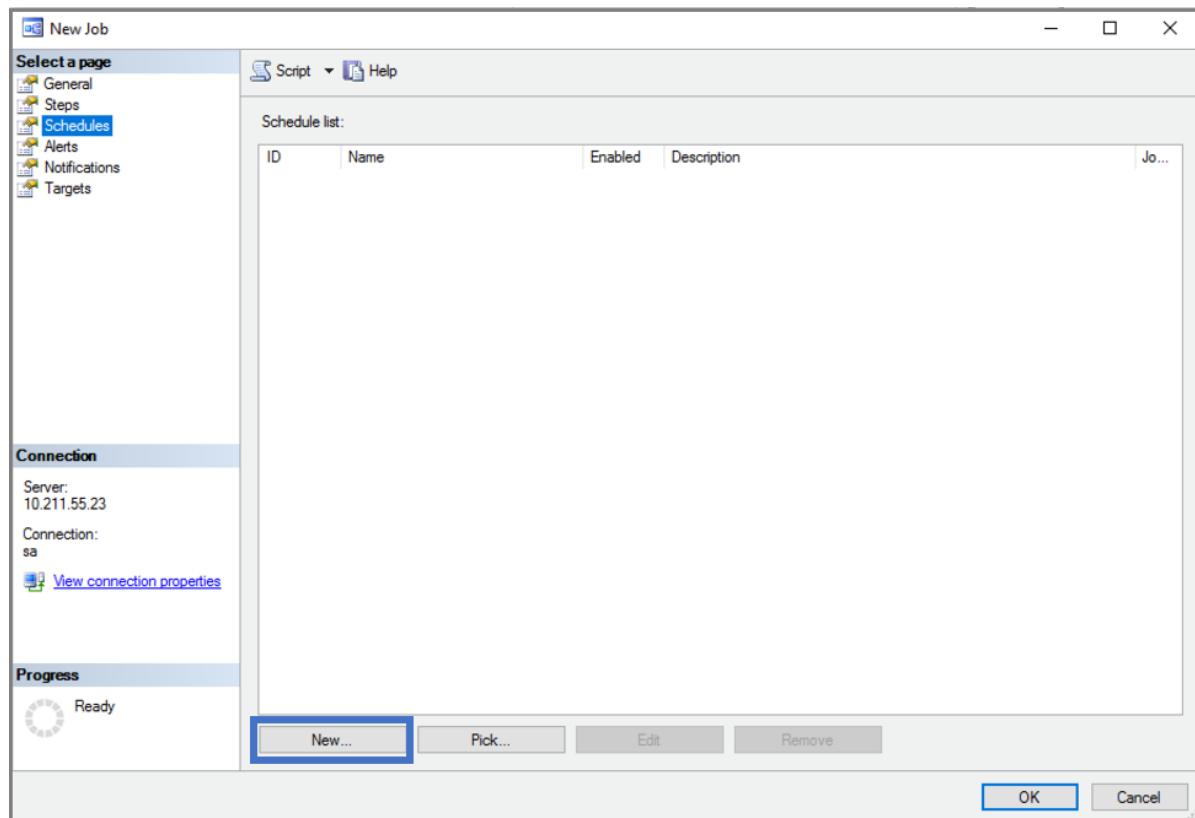
5. Give your job a name and create your job step.



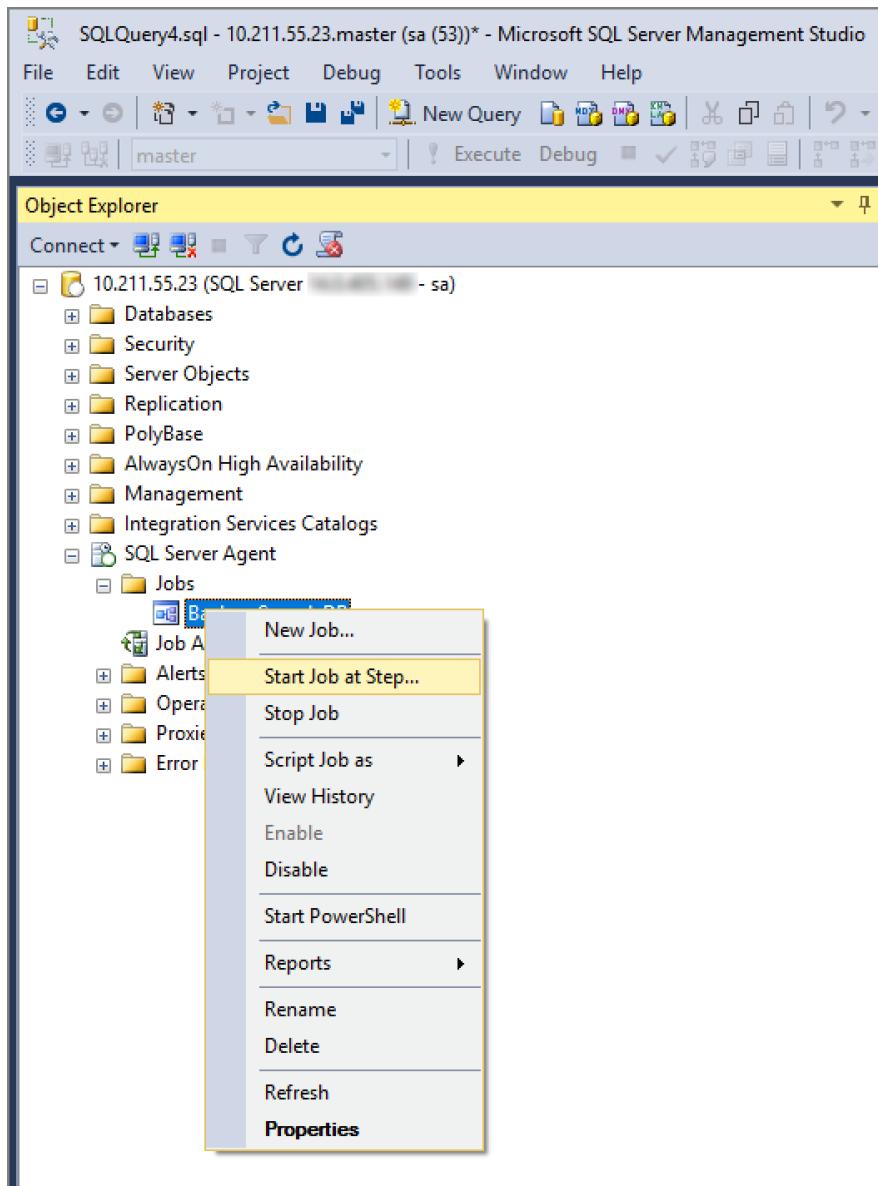
6. Specify what subsystem you want to use and what the job step should do.



7. Create a new job schedule.



8. Start your job.



Next Steps

In this tutorial, you learned how to:

- Install SQL Server Agent on Linux
- Use Transact-SQL and system stored procedures to create jobs
- Create a job that performs daily database backups
- Use SSMS UI to create and manage jobs

Next, explore other capabilities for creating and managing jobs:

[SQL Server Agent documentation](#)

Tutorial: Use Active Directory authentication with SQL Server on Linux

2/23/2018 • 8 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This tutorial explains how to configure SQL Server on Linux to support Active Directory (AD) authentication, also known as integrated authentication. For an overview, see [Active Directory authentication for SQL Server on Linux](#).

This tutorial consists of the following tasks:

- Join SQL Server host to AD domain
- Create AD user for SQL Server and set SPN
- Configure SQL Server service keytab
- Create AD-based logins in Transact-SQL
- Connect to SQL Server using AD Authentication

Prerequisites

Before you configure AD Authentication, you need to:

- Set up an AD Domain Controller (Windows) on your network
- Install SQL Server
 - [Red Hat Enterprise Linux](#)
 - [SUSE Linux Enterprise Server](#)
 - [Ubuntu](#)

Join SQL Server host to AD domain

Use the following steps to join a SQL Server host to an Active Directory domain:

1. Use **realmd** to join your host machine to your AD Domain. If you haven't already, install both the **realmd** and Kerberos client packages on the SQL Server host machine using your Linux distribution's package manager:

```
# RHEL
sudo yum install realmd krb5-workstation

# SUSE
sudo zypper install realmd krb5-client

# Ubuntu
sudo apt-get install realmd krb5-user software-properties-common python-software-properties packagekit
```

2. If the Kerberos client package installation prompts you for a realm name, enter your domain name in uppercase.

NOTE

This walkthrough uses "contoso.com" and "CONTOSO.COM" as example domain and realm names, respectively. You should replace these with your own values. These commands are case-sensitive, so make sure you use uppercase wherever it is used in this walkthrough.

3. Configure your SQL Server host machine to use your AD domain controller's IP address as a DNS nameserver.

- **Ubuntu:**

Edit the `/etc/network/interfaces` file so that your AD domain controller's IP address is listed as a dns-nameserver. For example:

```
<...>
# The primary network interface
auth eth0
iface eth0 inet dhcp
dns-nameservers **<AD domain controller IP address>**
dns-search **<AD domain name>**
```

NOTE

The network interface (eth0) might differ for different machines. To find out which one you are using, run ifconfig and copy the interface that has an IP address and transmitted and received bytes.

After editing this file, restart the network service:

```
sudo ifdown eth0 && sudo ifup eth0
```

Now check that your `/etc/resolv.conf` file contains a line like the following example:

```
nameserver **<AD domain controller IP address>**
```

- **RHEL:**

Edit the `/etc/sysconfig/network-scripts/ifcfg-eth0` file (or other interface config file as appropriate) so that your AD domain controller's IP address is listed as a DNS server:

```
<...>
PEERDNS=no
DNS1=**<AD domain controller IP address>**
```

After editing this file, restart the network service:

```
sudo systemctl restart network
```

Now check that your `/etc/resolv.conf` file contains a line like the following example:

```
nameserver **<AD domain controller IP address>**
```

4. Join the domain

Once you've confirmed that your DNS is configured properly, join the domain by running the following command. You must authenticate using an AD account that has sufficient privileges in AD to join a new machine to the domain.

Specifically, this command creates a new computer account in AD, create the `/etc/krb5.keytab` host keytab file, and configure the domain in `/etc/sssd/sssd.conf`:

```
sudo realm join contoso.com -U 'user@CONTOSO.COM' -v
<...>
* Successfully enrolled machine in realm
```

NOTE

If you see an error, "Necessary packages are not installed," then you should install those packages using your Linux distribution's package manager before running the `realm join` command again.

If you receive an error, "Insufficient permissions to join the domain," then you need to check with a domain administrator that you have sufficient permissions to join Linux machines to your domain.

SQL Server uses SSSD and NSS for mapping user accounts and groups to security identifiers (SID's). SSSD must be configured and running in order for SQL Server to create AD logins successfully. Realmd usually does this automatically as part of joining the domain, but in some cases you must do this separately.

Check out the following to configure [SSSD manually](#), and [configure NSS to work with SSSD](#)

5. Verify that you can now gather information about a user from the domain, and that you can acquire a Kerberos ticket as that user.

The following example uses `id`, `kinit`, and `klist` commands for this.

```
id user@contoso.com
uid=1348601103(user@contoso.com) gid=1348600513(domain group@contoso.com) groups=1348600513(domain
group@contoso.com)

kinit user@CONTOSO.COM
Password for user@CONTOSO.COM:

klist
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: user@CONTOSO.COM
<...>
```

NOTE

If `id user@contoso.com` returns, "No such user," make sure that the SSSD service started successfully by running the command `sudo systemctl status sssd`. If the service is running and you still see the "No such user" error, try enabling verbose logging for SSSD. For more information, see the Red Hat documentation for [Troubleshooting SSSD](#).

If `kinit user@CONTOSO.COM` returns, "KDC reply did not match expectations while getting initial credentials," make sure you specified the realm in uppercase.

For more information, see the Red Hat documentation for [Discovering and Joining Identity Domains](#).

Create AD user for SQL Server and set SPN

NOTE

The next steps use your [fully qualified domain name](#). If you are on [Azure](#), you must [create one](#) before you proceed.

1. On your domain controller, run the [New-ADUser](#) PowerShell command to create a new AD user with a password that never expires. This example names the account "mssql," but the account name can be anything you like. You will be prompted to enter a new password for the account:

```
Import-Module ActiveDirectory

New-ADUser mssql -AccountPassword (Read-Host -AsSecureString "Enter Password") -PasswordNeverExpires $true -Enabled $true
```

NOTE

It is a security best practice to have a dedicated AD account for SQL Server, so that SQL Server's credentials aren't shared with other services using the same account. However, you can reuse an existing AD account if you prefer, if you know the account's password (required to generate a keytab file in the next step).

2. Set the ServicePrincipalName (SPN) for this account using the [setspn.exe](#) tool. The SPN must be formatted exactly as specified in the following example. You can find the fully qualified domain name of the SQL Server host machine by running [hostname --all-fqdns](#) on the SQL Server host, and the TCP port should be 1433 unless you have configured SQL Server to use a different port number.

```
setspn -A MSSQLSvc/**<fully qualified domain name of host machine>**:**<tcp port>** mssql
```

NOTE

If you receive an error, "Insufficient access rights," then you need to check with a domain administrator that you have sufficient permissions to set an SPN on this account.

If you change the TCP port in the future, then you need to run the setspn command again with the new port number. You also need to add the new SPN to the SQL Server service keytab by following the steps in the next section.

3. For more information, see [Register a Service Principal Name for Kerberos Connections](#).

Configure SQL Server service keytab

1. Check the Key Version Number (kvno) for the AD account created in the previous step. Usually it is 2, but it could be another integer if you changed the account's password multiple times. On the SQL Server host machine, run the following:

```
kinit user@CONTOSO.COM

kvno MSSQLSvc/**<fully qualified domain name of host machine>**:**<tcp port>**
```

2. Create a keytab file with [ktutil](#) for the AD user you created in the previous step. When prompted, enter the password for that AD account.

```
sudo ktutil

ktutil: addent -password -p MSSQLSvc/**<fully qualified domain name of host machine>**:**<tcp
port>**@CONTOSO.COM -k **<kvno from above>** -e aes256-cts-hmac-sha1-96

ktutil: addent -password -p MSSQLSvc/**<fully qualified domain name of host machine>**:**<tcp
port>**@CONTOSO.COM -k **<kvno from above>** -e rc4-hmac

ktutil: wkt /var/opt/mssql/secrets/mssql.keytab

quit
```

NOTE

The ktutil tool does not validate the password, so make sure you enter it correctly.

3. Anyone with access to this `keytab` file can impersonate SQL Server on the domain, so make sure you restrict access to the file such that only the `mssql` account has read access:

```
sudo chown mssql:mssql /var/opt/mssql/secrets/mssql.keytab
sudo chmod 400 /var/opt/mssql/secrets/mssql.keytab
```

4. Configure SQL Server to use this `keytab` file for Kerberos authentication:

```
sudo /opt/mssql/bin/mssql-conf set network.kerberoskeytabfile /var/opt/mssql/secrets/mssql.keytab
sudo systemctl restart mssql-server
```

Create AD-based logins in Transact-SQL

1. Connect to SQL Server and create a new, AD-based login:

```
CREATE LOGIN [CONTOSO\user] FROM WINDOWS;
```

2. Verify that the login is now listed in the `sys.server_principals` system catalog view:

```
SELECT name FROM sys.server_principals;
```

Connect to SQL Server using AD Authentication

Log in to a client machine using your domain credentials. Now you can connect to SQL Server without reentering your password, by using AD Authentication. If you create a login for an AD group, any AD user who is a member of that group can connect in the same way.

The specific connection string parameter for clients to use AD Authentication depends on which driver you are using. Consider the following examples:

- `sqlcmd` on a domain-joined Linux client

Log in to a domain-joined Linux client using `ssh` and your domain credentials:

```
ssh -l user@contoso.com client.contoso.com
```

Make sure you've installed the [mssql-tools](#) package, then connect using `sqlcmd` without specifying any credentials:

```
sqlcmd -S mssql.contoso.com
```

- SSMS on a domain-joined Windows client

Log in to a domain-joined Windows client using your domain credentials. Make sure SQL Server Management Studio is installed, then connect to your SQL Server instance by specifying **Windows Authentication** in the **Connect to Server** dialog.

- AD Authentication using other client drivers
 - JDBC: [Using Kerberos Integrated Authentication to Connect SQL Server](#)
 - ODBC: [Using Integrated Authentication](#)
 - ADO.NET: [Connection String Syntax](#)

Next steps

In this tutorial, we walked through how to set up Active Directory authentication with SQL Server on Linux. You learned how to:

- Join SQL Server host to AD domain
- Create AD user for SQL Server and set SPN
- Configure SQL Server service keytab
- Create AD-based logins in Transact-SQL
- Connect to SQL Server using AD Authentication

Next, explore other security scenarios for SQL Server on Linux.

[Encrypting Connections to SQL Server on Linux](#)

Configure failover cluster instance - SQL Server on Linux (RHEL)

2/14/2018 • 8 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

A SQL Server two-node shared disk failover cluster instance provides server-level redundancy for high availability. In this tutorial, you learn how to create a two-node failover cluster instance of SQL Server on Linux. The specific steps that you will complete include:

- Set up and configure Linux
- Install and configure SQL Server
- Configure the hosts file
- Configure shared storage and move the database files
- Install and configure Pacemaker on each cluster node
- Configure the failover cluster instance

This article explains how to create a two-node shared disk failover cluster instance (FCI) for SQL Server. The article includes instructions and script examples for Red Hat Enterprise Linux (RHEL). Ubuntu distributions are similar to RHEL so the script examples will normally also work on Ubuntu.

For conceptual information, see [SQL Server Failover Cluster Instance \(FCI\) on Linux](#).

Prerequisites

To complete the following end-to-end scenario, you need two machines to deploy the two nodes cluster and another server for storage. Below steps outline how these servers will be configured.

Set up and configure Linux

The first step is to configure the operating system on the cluster nodes. On each node in the cluster, configure a linux distribution. Use the same distribution and version on both nodes. Use either one or the other of the following distributions:

- RHEL with a valid subscription for the HA add-on

Install and configure SQL Server

1. Install and set up SQL Server on both nodes. For detailed instructions, see [Install SQL Server on Linux](#).
2. Designate one node as primary and the other as secondary, for purposes of configuration. Use these terms for the following this guide.
3. On the secondary node, stop and disable SQL Server. The following example stops and disables SQL Server:

```
sudo systemctl stop mssql-server
sudo systemctl disable mssql-server
```

NOTE

At set up time, a Server Master Key is generated for the SQL Server instance and placed at `var/opt/mssql/secrets/machine-key`. On Linux, SQL Server always runs as a local account called mssql. Because it's a local account, its identity isn't shared across nodes. Therefore, you need to copy the encryption key from primary node to each secondary node so each local mssql account can access it to decrypt the Server Master Key.

4. On the primary node, create a SQL server login for Pacemaker and grant the login permission to run

```
sp_server_diagnostics
```

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

Connect to the SQL Server `master` database with the sa account and run the following:

```
USE [master]
GO
CREATE LOGIN [<loginName>] with PASSWORD= N'<loginPassword>'
ALTER SERVER ROLE [sysadmin] ADD MEMBER [<loginName>]
```

Alternatively, you can set the permissions at a more granular level. The Pacemaker login requires `VIEW SERVER STATE` to query health status with `sp_server_diagnostics`, `setupadmin` and `ALTER ANY LINKED SERVER` to update the FCI instance name with the resource name by running `sp_dropserver` and `sp_addserver`.

5. On the primary node, stop and disable SQL Server.

Configure the hosts file

On each cluster node, configure the hosts file. The hosts file must include the IP address and name of every cluster node.

1. Check the IP address for each node. The following script shows the IP address of your current node.

```
sudo ip addr show
```

2. Set the computer name on each node. Give each node a unique name that is 15 characters or less. Set the computer name by adding it to `/etc/hosts`. The following script lets you edit `/etc/hosts` with `vi`.

```
sudo vi /etc/hosts
```

The following example shows `/etc/hosts` with additions for two nodes named `sqlfcivm1` and `sqlfcivm2`.

```
127.0.0.1 localhost localhost4 localhost4.localdomain4
::1 localhost localhost6 localhost6.localdomain6
10.128.18.128 sqlfcivm1
10.128.16.77 sqlfcivm2
```

Configure storage & move database files

You need to provide storage that both nodes can access. You can use iSCSI, NFS, or SMB. Configure storage, present the storage to the cluster nodes, and then move the database files to the new storage. The following

articles explain the steps for each storage type:

- [Configure failover cluster instance - iSCSI - SQL Server on Linux](#)
- [Configure failover cluster instance - NFS - SQL Server on Linux](#)
- [Configure failover cluster instance - SMB - SQL Server on Linux](#)

Install and configure Pacemaker on each cluster node

1. On both cluster nodes, create a file to store the SQL Server username and password for the Pacemaker login.

The following command creates and populates this file:

```
sudo touch /var/opt/mssql/secrets/passwd
sudo echo '<loginName>' >> /var/opt/mssql/secrets/passwd
sudo echo '<loginPassword>' >> /var/opt/mssql/secrets/passwd
sudo chown root:root /var/opt/mssql/secrets/passwd
sudo chmod 600 /var/opt/mssql/secrets/passwd
```

2. On both cluster nodes, open the Pacemaker firewall ports. To open these ports with `firewalld`, run the following command:

```
sudo firewall-cmd --permanent --add-service=high-availability
sudo firewall-cmd --reload
```

If you're using another firewall that doesn't have a built-in high-availability configuration, the following ports need to be opened for Pacemaker to be able to communicate with other nodes in the cluster

- TCP: Ports 2224, 3121, 21064
- UDP: Port 5405

3. Install Pacemaker packages on each node.

```
sudo yum install pacemaker pcs fence-agents-all resource-agents
```

4. Set the password for the default user that is created when installing Pacemaker and Corosync packages. Use the same password on both nodes.

```
sudo passwd hacluster
```

5. Enable and start `pcsd` service and Pacemaker. This will allow nodes to rejoin the cluster after the reboot. Run the following command on both nodes.

```
sudo systemctl enable pcsd
sudo systemctl start pcsd
sudo systemctl enable pacemaker
```

6. Install the FCI resource agent for SQL Server. Run the following commands on both nodes.

```
sudo yum install mssql-server-ha
```

Configure the failover cluster instance

The FCI will be created in a resource group. This is a little bit easier since the resource group alleviates the need for constraints. However, add the resources into the resource group in the order they should start. The order they should start is:

1. Storage resource
2. Network resource
3. Application resource

This example will create an FCI in the group NewLinFCIGrp. The name of the resource group must be unique from any resource created in Pacemaker.

1. Create the disk resource. You will get no response back if there is not a problem. The way to create the disk resource depends on the storage type. The following is an example for each storage type. Use the example that applies to the storage type for your clustered storage.

iSCSI

```
sudo pcs resource create <iSCSIDiskResourceName> Filesystem  
device="/dev/<VolumeGroupName>/<LogicalVolumeName>" directory="<>FolderToMountiSCSIDisk>" fstype="<>FileSystemType>" --group RGName
```

<iSCSIDiskResourceName> is the name of the resource associated with the iSCSI disk

<VolumeGroupName> is the name of the volume group

<LogicalVolumeName> is the name of the logical volume that was created

<FolderToMountiSCSIDisk> is the folder to mount the disk (for system databases and the default location, it would be /var/opt/mssql/data)

<FileSystemType> would be EXT4 or XFS depending on how things were formatted and what the distribution supports.

NFS

```
sudo pcs resource create <NFSDiskResourceName> Filesystem device="<>IPAddressOfNFSServer>:<>FolderOnNFSServer>" directory="<>FolderToMountNFSShare>" fstype=nfs4 options="nfsvers=4.2,timeo=14,intr" --group RGName  
mount -t nfs4 <IPAddressOfNFSServer>:<FolderOnNFSServer> /var/opt/mssql/data -o
```

<NFSDiskResourceName> is the name of the resource associated with the NFS share

<IPAddressOfNFSServer> is the IP address of the NFS server that you are going to use

<FolderOnNFSServer> is the name of the NFS share

<FolderToMountNFSShare> is the folder to mount the disk (for system databases and the default location, it would be /var/opt/mssql/data)

An example is shown here:

```
mount -t nfs4 200.201.202.63:/var/nfs/fci1 /var/opt/mssql/data -o nfsvers=4.2,timeo=14,intr
```

SMB

```
sudo pcs resource create SMBDiskResourceName Filesystem device="//<ServerName>/<ShareName>" directory="  
<FolderName>" fstype=cifs options="vers=3.0,username=<UserName>,password=<Password>,domain=  
<ADDomain>,uid=<mssqlUID>,gid=<mssqlGID>,file_mode=0777,dir_mode=0777" --group <RGName>
```

<ServerName> is the name of the server with the SMB share

<ShareName> is the name of the share

<FolderName> is the name of the folder created in the last step

<UserName> is the name of the user to access the share

<Password> is the password for the user

<ADDomain> is the AD DS domain (if applicable when using a Windows Server-based SMB share)

<mssqlUID> is the UID of the mssql user

<mssqlGID> is the GID of the mssql user

<RGName> is the name of the resource group

2. Create the IP address that will be used by the FCI. You will get no response back if there is not a problem.

```
sudo pcs resource create <IPResourceName> ocf:heartbeat:IPAddr2 ip=<IPAddress> nic=<NetworkCard>  
cidr_netmask=<NetMask> --group <RGName>
```

<IPResourceName> is the name of the resource associated with the IP address

<IPAddress> is the IP address for the FCI

<NetworkCard> is the network card associated with the subnet (i.e. eth0)

<NetMask> is the netmask of the subnet (i.e. 24)

<RGName> is the name of the resource group

3. Create the FCI resource. You will get no response back if there is not a problem.

```
sudo pcs resource create FCIResourceName ocf:mssql:fci op defaults timeout=60s --group RGName
```

<FCIResourceName> is not only the name of the resource, but the friendly name that is associated with the FCI. This is what users and applications will use to connect.

<RGName> is the name of the resource group.

4. Run the command `sudo pcs resource`. The FCI should be online.
5. Connect to the FCI with SSMS or sqlcmd using the DNS/resource name of the FCI.
6. Issue the statement `SELECT @@SERVERNAME`. It should return the name of the FCI.
7. Issue the statement `SELECT SERVERPROPERTY('ComputerNamePhysicalNetBIOS')`. It should return the name of the node that the FCI is running on.
8. Manually fail the FCI to the other node(s). See the instructions under [Operate failover cluster instance - SQL Server on Linux](#).
9. Finally, fail the FCI back to the original node and remove the colocation constraint.

Summary

In this tutorial you completed the following tasks.

- Set up and configure Linux
- Install and configure SQL Server
- Configure the hosts file
- Configure shared storage and move the database files
- Install and configure Pacemaker on each cluster node
- Configure the failover cluster instance

Next steps

- [Operate failover cluster instance - SQL Server on Linux](#)

Configure failover cluster instance - iSCSI - SQL Server on Linux

2/14/2018 • 8 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Linux only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

This article explains how to configure iSCSI storage for a failover cluster instance (FCI) on Linux.

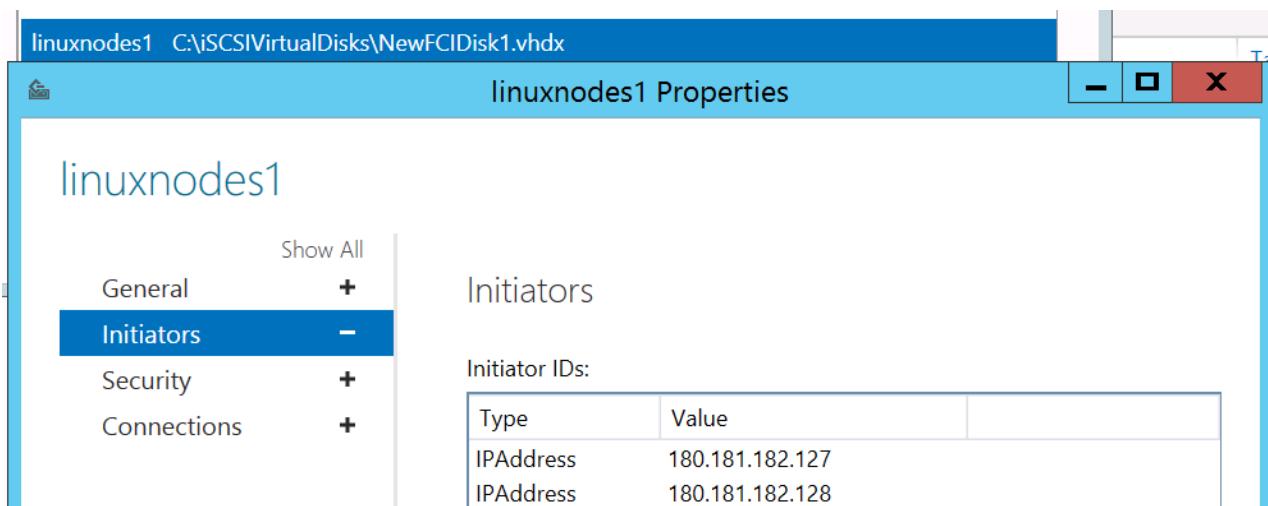
Configure iSCSI

iSCSI uses networking to present disks from a server known as a target to servers. The servers connecting to the iSCSI target require that an iSCSI initiator is configured. The disks on the target are given explicit permissions so that only the initiators that should be able to access them can do so. The target itself should be highly available and reliable.

Important iSCSI target information

While this section will not cover how to configure an iSCSI target since it is specific to the type of source you will be using, ensure that the security for the disks that will be used by the cluster nodes is configured.

The target should never be configured on any of the FCI nodes if using a Linux-based iSCSI target. For performance and availability, iSCSI networks should be separate from those used by regular network traffic on both the source and the client servers. Networks used for iSCSI should be fast. Remember that network does consume some processor bandwidth, so plan accordingly if using a regular server. The most important thing to ensure is completed on the target is that the disks that are created are assigned the proper permissions so that only those servers participating in the FCI have access to them. An example is shown below from the Microsoft iSCSI target where linuxnodes1 is the name created, and in this case, the IP addresses of the nodes are assigned so that NewFCIDisk1.vhdx appears to them.



Instructions

This section will cover how to configure an iSCSI initiator on the servers that will serve as nodes for the FCI. The instructions should work as is on RHEL and Ubuntu.

For more information on iSCSI initiator for the supported distributions, consult the following links:

- [Red Hat](#)
- [SUSE](#)

- Ubuntu

1. Choose one of the servers that will participate in the FCI configuration. It does not matter which one. iSCSI should be on a dedicated network, so configure iSCSI to recognize and use that network. Run
`sudo iscsiadadm -m iface -I <iSCSIIfaceName> -o new` where `<iSCSIIfaceName>` is the unique or friendly name for the network. The following example uses `iSCSINIC`:

```
sudo iscsiadadm -m iface -I iSCSINIC -o new
```

```
[allan@FCIN2 ~]$ sudo iscsiadadm -m iface -I iSCSINIC -o new
New interface iSCSINIC added
```

2. Edit `/var/lib/iscsi/ifaces/iSCSIIfaceName`. Make sure it has the following values completely filled out:

- `iface.net_ifacename` is the name of the network card as seen in the OS.
- `iface.hwaddress` is the MAC address of the unique name that will be created for this interface below.
- `iface.ipaddress`
- `iface.subnet_Mask`

See the following example:

```
# BEGIN RECORD 6.2.0.873-35
iface.iscsi_ifacename = iSCSINIC
iface.net_ifacename = eth1
iface.hwaddress = 00:15:5D:01:74:88
iface.ipaddress = 180.181.182.128
iface.subnet_mask = 255.255.0.0
iface.transport_name = tcp
iface.vlan_id = 0
iface.vlan_priority = 0
iface.iface_num = 0
iface.mtu = 0
iface.port = 0
iface.tos = 0
iface.ttl = 0
iface.tcp_wsf = 0
iface.tcp_timer_scale = 0
iface.def_task_mgmt_timeout = 0
-::-- iSCSINIC Top L1 (Fundamental)
```

3. Find the iSCSI target.

```
sudo iscsiadadm -m discovery -t sendtargets -I <iSCSINetName> -p <TargetIPAddress>:<TargetPort>
```

`<iSCSINetName>` is the unique/friendly name for the network, `<TargetIPAddress>` is the IP address of the iSCSI target, and `<TargetPort>` is the port of the iSCSI target.

```
[allan@FCIN2 ~]$ sudo iscsiadadm -m discovery -t sendtargets -I iSCSINIC -p 180.181.182.1
180.181.182.1:3260,1 iqn.1991-05.com.microsoft:dc1-linuxnodes1-target
200.201.202.1:3260,1 iqn.1991-05.com.microsoft:dc1-linuxnodes1-target
[2002:b4b5:b601::b4b5:b601]:3260,1 iqn.1991-05.com.microsoft:dc1-linuxnodes1-target
[2002:c8c9:ca01::c8c9:ca01]:3260,1 iqn.1991-05.com.microsoft:dc1-linuxnodes1-target
```

4. Log into the target

```
sudo iscsiadadm -m node -I <iSCSIIfaceName> -p TargetIPAddress -l
```

<iSCSIIfaceName> is the unique/friendly name for the network and <TargetIPAddress> is the IP address of the iSCSI target.

```
[allan@FCIN2 ~]$ sudo iscsiadadm -m node -I iSCSINIC -p 180.181.182.1 -l  
Logging in to [iface: iSCSINIC, target: iqn.1991-05.com.microsoft:dc1-linuxnodes1-target, portal: 180.181.182.1,3260] (multiple)  
Login to [iface: iSCSINIC, target: iqn.1991-05.com.microsoft:dc1-linuxnodes1-target, portal: 180.181.182.1,3260] successful.
```

5. Check to see that there is a connection to the iSCSI target.

```
sudo iscsiadadm -m session
```

```
[allan@FCIN2 ~]$ sudo iscsiadadm -m session  
tcp: [1] 180.181.182.1:3260,1 iqn.1991-05.com.microsoft:dc1-linuxnodes1-target (non-flash) -
```

6. Check iSCSI attached disks

```
sudo grep "Attached SCSI" /var/log/messages
```

```
[allan@FCIN2 ~]$ sudo grep "Attached SCSI" /var/log/messages  
Aug 26 11:38:26 FCIN2 kernel: sd 2:0:0:0: [sda] Attached SCSI disk  
Aug 26 11:38:32 FCIN2 kernel: sd 4:0:0:0: [sdb] Attached SCSI disk  
Aug 26 11:38:32 FCIN2 kernel: sd 4:0:0:1: [sdc] Attached SCSI disk  
Aug 26 11:38:32 FCIN2 kernel: sd 4:0:0:2: [sdd] Attached SCSI disk  
Aug 26 11:38:32 FCIN2 kernel: sd 4:0:0:3: [sde] Attached SCSI disk
```

7. Create a physical volume on the iSCSI disk.

```
sudo pvcreate /dev/<devicename>
```

<devicename> is the name of the device from the previous step.

8. Create a volume group on the iSCSI disk. Disks assigned to a single volume group are seen as a pool or collection.

```
sudo vgcreate <VolumeGroupName> /dev/devicename
```

<VolumeGroupName> is the name of the volume group and <devicename> is the name of the device from Step 6.

9. Create and verify the logical volume for the disk.

```
sudo lvcreate -Lsize -n <LogicalVolumeName> <VolumeGroupName>
```

<size> is the size of the volume to create, and can be specified with G (gigabytes), T (terabytes), etc., <LogicalVolumeName> is the name of the logical volume, and <VolumeGroupName> is the name of the volume group from the previous step.

The example below creates a 25GB volume.

```
[allan@FCIN2 ~]$ sudo lvcreate -L25G -n FCIDataLV1 FCIDataVG1
Logical volume "FCIDataLV1" created.
```

10. Execute `sudo lvs` to see the LVM that was created.

11. Format the logical volume with a supported filesystem. For EXT4, use the following example:

```
sudo mkfs.ext4 /dev/<VolumeGroupName>/<LogicalVolumeName>
```

`<VolumeGroupName>` is the name of the volume group from the previous step. `<LogicalVolumeName>` is the name of the logical volume from the previous step.

12. For system databases or anything stored in the default data location, follow these steps. Otherwise, skip to Step 13.

- Ensure that SQL Server is stopped on the server that you are working on.

```
sudo systemctl stop mssql-server
sudo systemctl status mssql-server
```

- Switch fully to be the superuser. You will not receive any acknowledgement if successful.

```
sudo -i
```

- Switch to be the mssql user. You will not receive any acknowledgement if successful.

```
su mssql
```

- Create a temporary directory to store the SQL Server data and log files. You will not receive any acknowledgement if successful.

```
mkdir <TempDir>
```

`<TempDir>` is the name of the folder. The example below creates a folder named `/var/opt/mssql/TempDir`.

```
mkdir /var/opt/mssql/TempDir
```

- Copy the SQL Server data and log files to the temporary directory. You will not receive any acknowledgement if successful.

```
cp /var/opt/mssql/data/* <TempDir>
```

`<TempDir>` is the name of the folder from the previous step.

- Verify that the files are in the directory.

```
ls \<TempDir>
```

`<TempDir>` is the name of the folder from Step d.

- Delete the files from the existing SQL Server data directory. You will not receive any acknowledgement if successful.

```
rm - f /var/opt/mssql/data/*
```

- Verify that the files have been deleted. The picture below shows an example of the entire sequence from c through h.

```
ls /var/opt/mssql/data
```

```
[allan@FCIN2 ~]$ sudo -i  
[root@FCIN2 ~]# su mssql  
bash-4.2$ mkdir /var/opt/mssql/tmp  
bash-4.2$ cp /var/opt/mssql/data/* /var/opt/mssql/tmp  
bash-4.2$ ls /var/opt/mssql/tmp  
master.mdf mastlog.ldf modellog.ldf model.mdf msdbdata.mdf msdblog.ldf tempdb.mdf templog.ldf  
bash-4.2$ rm -f /var/opt/mssql/data/*  
bash-4.2$ ls /var/opt/mssql/data  
bash-4.2$ _
```

- Type `exit` to switch back to the root user.
- Mount the iSCSI logical volume in the SQL Server data folder. You will not receive any acknowledgement if successful.

```
mount /dev/<VolumeGroupName>/<LogicalVolumeName> /var/opt/mssql/data
```

<VolumeGroupName> is the name of the volume group and <LogicalVolumeName> is the name of the logical volume that was created. The following example syntax matches the volume group and logical volume from the previous command.

```
mount /dev/FCIDataVG1/FCIDataLV1 /var/opt/mssql/data
```

- Change the owner of the mount to mssql. You will not receive any acknowledgement if successful.

```
chown mssql /var/opt/mssql/data
```

- Change ownership of the group of the mount to mssql. You will not receive any acknowledgement if successful.

```
chgrp mssql /var/opt/mssql/data
```

- Switch to the mssql user. You will not receive any acknowledgement if successful.

```
su mssql
```

- Copy the files from the temporary directory /var/opt/mssql/data. You will not receive any acknowledgement if successful.

```
cp /var/opt/mssql/TmpDir/* /var/opt/mssql/data
```

- Verify the files are there.

```
ls /var/opt/mssql/data
```

- Enter `exit` to not be mssql.

- Enter `exit` to not be root.
- Start SQL Server. If everything was copied correctly and security applied correctly, SQL Server should show as started.

```
sudo systemctl start mssql-server
sudo systemctl status mssql-server
```

- Stop SQL Server and verify it is shut down.

```
sudo systemctl stop mssql-server
sudo systemctl status mssql-server
```

13. For things other than system databases, such as user databases or backups, follow these steps. If only using the default location, skip to Step 14.

- Switch to be the superuser. You will not receive any acknowledgement if successful.

```
sudo -i
```

- Create a folder that will be used by SQL Server.

```
mkdir <FolderName>
```

`<FolderName>` is the name of the folder. The folder's full path needs to be specified if not in the right location. The example below creates a folder named `/var/opt/mssql/userdata`.

```
mkdir /var/opt/mssql/userdata
```

- Mount the iSCSI logical volume in the folder that was created in the previous step. You will not receive any acknowledgement if successful.

```
mount /dev/<VolumeGroupName>/<LogicalVolumeName> <FolderName>
```

`<VolumeGroupName>` is the name of the volume group, `<LogicalVolumeName>` is the name of the logical volume that was created, and `<FolderName>` is the name of the folder. Example syntax is shown below.

```
mount /dev/FCIDataVG2/FCIDataLV2 /var/opt/mssql/userdata
```

- Change ownership of the folder created to mssql. You will not receive any acknowledgement if successful.

```
chown mssql <FolderName>
```

`<FolderName>` is the name of the folder that was created. An example is shown below.

```
chown mssql /var/opt/mssql/userdata
```

- Change the group of the folder created to mssql. You will not receive any acknowledgement if successful.

```
chown mssql <FolderName>
```

<FolderName> is the name of the folder that was created. An example is shown below.

```
chown mssql /var/opt/mssql/userdata
```

- Type `exit` to no longer be the superuser.
- To test, create a database in that folder. The example shown below uses `sqlcmd` to create a database, switch context to it, verify the files exist at the OS level, and then deletes the temporary location. You can use SSMS.

```
[allan@FCIN2 ~]$ sqlcmd -Usa -PP@ssword1
1> drop database TestDB
2> GO
1> CREATE DATABASE TestDB ON
2> (NAME = TestDB_Data, FILENAME = '/var/opt/mssql/userdata/TestDB_Data.mdf')
3> LOG ON (NAME = TestDB_Log, FILENAME = '/var/opt/mssql/userdata/TestDB_Log.ldf')
4> go
1> use TestDB
2> GO
Changed database context to 'TestDB'.
1> exit
[allan@FCIN2 ~]$ sudo ls /var/opt/mssql/userdata
lost+found  TestDB_Data.mdf  TestDB_Log.ldf
[allan@FCIN2 ~]$ sqlcmd -Usa -PP@ssword1
1> drop database TestDB
2> go
1> exit
[allan@FCIN2 ~]$ sudo ls /var/opt/mssql/userdata
lost+found
```

- Unmount the share

```
sudo umount /dev/<VolumeGroupName>/<LogicalVolumeName> <FolderName>
```

<VolumeGroupName> is the name of the volume group, <LogicalVolumeName> is the name of the logical volume that was created, and <FolderName> is the name of the folder. Example syntax is shown below.

```
sudo umount /dev/FCIDataVG2/FCIDataLV2 /var/opt/mssql/userdata
```

14. Configure the server so that only Pacemaker can activate the volume group.

```
sudo lvmconf --enable-halvm --services -startstopservices
```

15. Generate a list of the volume groups on the server. Anything listed that is not the iSCSI disk is used by the system, such as for the OS disk.

```
sudo vgs
```

16. Modify the activation configuration section of the file `/etc/lvm/lvm.conf`. Configure the following line:

```
volume_list = [ <ListOfVGsNotUsedByPacemaker> ]
```

<ListOfVGsNotUsedByPacemaker> is the list of volume groups from the output of Step 20 that will not be

used by the FCI. Put each one in quotes and separate by a comma. An example is shown below.

```
#      is assumed.  
#  
# Example  
# volume_list = [ "vg1", "vg2/lvol1", "@tag1", "@" ]  
#  
# This configuration option does not have a default value defined.  
volume_list = [ "cl" ]
```

- When Linux starts, it will mount the file system. To ensure that only Pacemaker can mount the iSCSI disk, rebuild the root filesystem image.

Run the following command which may take a few moments to complete. You will get no message back if successful.

```
sudo dracut -H -f /boot/initramfs-$(uname -r).img $(uname -r)
```

- Reboot the server.

- On another server that will participate in the FCI, perform Steps 1 – 6. This will present the iSCSI target to the SQL Server.

- Generate a list of the volume groups on the server. It should show the volume group created earlier.

```
sudo vgs
```

- Start SQL Server and verify it can be started on this server.

```
sudo systemctl start mssql-server  
sudo systemctl status mssql-server
```

- Stop SQL Server and verify that it is shut down.

```
sudo systemctl stop mssql-server  
sudo systemctl status mssql-server
```

- Repeat Steps 1 – 6 on any other servers that will participate in the FCI.

You are now ready to configure the FCI.

DISTRIBUTION	TOPIC
Red Hat Enterprise Linux with HA add-on	Configure Operate
SUSE Linux Enterprise Server with HA add-on	Configure

Next steps

[Configure failover cluster instance - SQL Server on Linux](#)

Configure failover cluster instance - NFS - SQL Server on Linux

2/14/2018 • 5 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Linux only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

This article explains how to configure NFS storage for a failover cluster instance (FCI) on Linux.

NFS, or network file system, is a common method for sharing disks in the Linux world but not the Windows one. Similar to iSCSI, NFS can be configured on a server or some sort of appliance or storage unit as long as it meets the storage requirements for SQL Server.

Important NFS server information

The source hosting NFS (either a Linux server or something else) must be using/compliant with version 4.2 or later. Earlier versions will not work with SQL Server on Linux.

When configuring the folder(s) to be shared on the NFS server, make sure they follow these guidelines general options:

- `rw` to ensure that the folder can be read from and written to
- `sync` to ensure guaranteed writes to the folder
- Do not use `no_root_squash` as an option; it is considered a security risk
- Make sure the folder has full rights (777) applied

Ensure that your security standards are enforced for accessing. When configuring the folder, make sure that only the servers participating in the FCI should see the NFS folder. An example of a modified `/etc/exports` on a Linux-based NFS solution is shown below where the folder is restricted to FCIN1 and FCIN2.

```
File Edit Options Buffers Tools Help
# /etc/exports: the access control list for filesystems which may be exported
#           to NFS clients.  See exports(5).
#
#
/var/nfs/fci1  FCIN1(rw,sync) FCIN2(rw,sync)
```

Instructions

1. Choose one of the servers that will participate in the FCI configuration. It does not matter which one.
2. Check to see that the server can see the mount(s) on the NFS server.

```
sudo showmount -e <IPAddressOfNFSServer>
```

`<IPAddressOfNFSServer>` is the IP address of the NFS server that you are going to use.

3. For system databases or anything stored in the default data location, follow these steps. Otherwise, skip to Step 4.
 - Ensure that SQL Server is stopped on the server that you are working on.

```
sudo systemctl stop mssql-server  
sudo systemctl status mssql-server
```

- Switch fully to be the superuser. You will not receive any acknowledgement if successful.

```
sudo -i
```

- Switch to be the mssql user. You will not receive any acknowledgement if successful.

```
su mssql
```

- Create a temporary directory to store the SQL Server data and log files. You will not receive any acknowledgement if successful.

```
mkdir <TempDir>
```

<TempDir> is the name of the folder. The following example creates a folder named /var/opt/mssql/tmp.

```
mkdir /var/opt/mssql/tmp
```

- Copy the SQL Server data and log files to the temporary directory. You will not receive any acknowledgement if successful.

```
cp /var/opt/mssql/data/* <TempDir>
```

<TempDir> is the name of the folder from the previous step.

- Verify that the files are in the directory.

```
ls TempDir
```

<TempDir> is the name of the folder from Step d.

- Delete the files from the existing SQL Server data directory. You will not receive any acknowledgement if successful.

```
rm - f /var/opt/mssql/data/*
```

- Verify that the files have been deleted.

```
ls /var/opt/mssql/data
```

- Type exit to switch back to the root user.
- Mount the NFS share in the SQL Server data folder. You will not receive any acknowledgement if successful.

```
mount -t nfs4 <IPAddressOfNFSServer>:<FolderOnNFSServer> /var/opt/mssql/data -o  
nfsvers=4.2,timeo=14,intr
```

<IPAddressOfNFSServer> is the IP address of the NFS server that you are going to use

<FolderOnNFSServer> is the name of the NFS share. The following example syntax matches the NFS information from Step 2.

```
mount -t nfs4 200.201.202.63:/var/nfs/fci1 /var/opt/mssql/data -o nfsvers=4.2,timeo=14,intr
```

- Check to see that the mount was successful by issuing mount with no switches.

```
mount
```

```
[root@FCIN2 ~]# mount 200.201.202.63:/var/nfs/fci1 /var/opt/mssql/data -o nfsvers=4.2,timeo=14,intr  
[root@FCIN2 ~]# mount  
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime,seclabel)  
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)  
devtmpfs on /dev type devtmpfs (rw,nosuid,seclabel,size=1917444k,nr_inodes=479361,mode=755)  
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)  
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev,seclabel)  
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,seclabel,gid=5,mode=620,ptmxmode=000)  
tmpfs on /run type tmpfs (rw,nosuid,nodev,seclabel,mode=755)  
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,seclabel,mode=755)  
cgroup on /sys/fs/cgroup/systemd type cgroup (rw,nosuid,nodev,noexec,relatime,xattr,release_agent=/usr/lib/systemd/systemd-cgr  
ousp-agent,name=systemd)  
pstore on /sys/fs/pstore type pstore (rw,nosuid,nodev,noexec,relatime)  
cgroup on /sys/fs/cgroup/cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)  
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,pids)  
cgroup on /sys/fs/cgroup/net_cls.net_prio type cgroup (rw,nosuid,nodev,noexec,relatime,net_prio,net_cls)  
cgroup on /sys/fs/cgroup/cpu.cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,cpuacct,cpu)  
cgroup on /sys/fs/cgroup/freezer type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)  
cgroup on /sys/fs/cgroup/perf_event type cgroup (rw,nosuid,nodev,noexec,relatime,perf_event)  
cgroup on /sys/fs/cgroup/devices type cgroup (rw,nosuid,nodev,noexec,relatime,devices)  
cgroup on /sys/fs/cgroup/hugetlb type cgroup (rw,nosuid,nodev,noexec,relatime,hugetlb)  
cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory)  
cgroup on /sys/fs/cgroup/bikio type cgroup (rw,nosuid,nodev,noexec,relatime,bikio)  
configfs on /sys/kernel/config type configfs (rw,relatime)  
/dev/mapper/cl-root on / type xfs (rw,relatime,seclabel,attr2,inode64,noquota)  
selinuxfs on /sys/fs/selinux type selinuxfs (rw,relatime)  
systemd-1 on /proc/sys/fs/binfmt_misc type autofs (rw,relatime,fd=33,pgrp=1,timeout=300,minproto=5,maxproto=5,direct)  
hugetlbfss on /dev/hugepages type hugetlbfss (rw,relatime,seclabel)  
debugfs on /sys/kernel/debug type debugfs (rw,relatime)  
mqqueue on /dev/queue type mqqueue (rw,relatime,seclabel)  
nfsd on /proc/fs/nfsd type nfsd (rw,relatime)  
/dev/sdal on /boot type xfs (rw,relatime,seclabel,attr2,inode64,noquota)  
/dev/mapper/cl-home on /home type xfs (rw,relatime,seclabel,attr2,inode64,noquota)  
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw,relatime)  
tmpfs on /run/user/1000 type tmpfs (rw,nosuid,nodev,relatime,seclabel,size=386568k,mode=700,uid=1000,gid=1000)  
gvfsd-fuse on /run/user/1000/gvfs type fuse.gvfsd-fuse (rw,nosuid,nodev,relatime,user_id=1000,group_id=1000)  
fusectl on /sys/fs/fuse/connections type fusectl (rw,relatime)  
200.201.202.63:/var/nfs/fci1 on /var/opt/mssql/data type nfs4 (rw,relatime,vers=4.2,rsize=524288,wszie=524288,namlen=255,hard,  
proto=tcp,port=0,timeo=14,retrans=2,sec=sys,clientaddr=200.201.202.128,local lock=none,addr=200.201.202.63)
```

- Switch to the mssql user. You will not receive any acknowledgement if successful.

```
su mssql
```

- Copy the files from the temporary directory /var/opt/mssql/data. You will not receive any acknowledgement if successful.

```
cp /var/opt/mssql/tmp/* /var/opt/mssql/data
```

- Verify the files are there.

```
ls /var/opt/mssql/data
```

- Enter exit to not be mssql
- Enter exit to not be root
- Start SQL Server. If everything was copied correctly and security applied correctly, SQL Server should

show as started.

```
sudo systemctl start mssql-server
sudo systemctl status mssql-server
```

- Create a database to test that security is set up properly. The following example shows that being done via Transact-SQL; it can be done via SSMS.

```
[allan@FCIN2 ~]$ sqlcmd -Usa -PP@ssword1
1> CREATE DATABASE TestDB
2> ON (NAME = TestDB_Data, FILENAME = '/var/opt/mssql/data/TestDB_Data.mdf')
3> LOG ON (NAME = TestDB_Log, FILENAME = '/var/opt/mssql/data/TestDB_Log.ldf')
4> GO
1> USE TestDB
2> GO
Changed database context to 'TestDB'.
1> exit
[allan@FCIN2 ~]$ sudo ls -l /var/opt/mssql/data
total 71552
-rw-r-----. 1 mssql mssql 4653056 Aug 27 01:18 master.mdf
-rw-r-----. 1 mssql mssql 2097152 Aug 27 01:22 mastlog.ldf
-rw-r-----. 1 mssql mssql 8388608 Aug 27 01:22 modellog.ldf
-rw-r-----. 1 mssql mssql 8388608 Aug 27 01:22 model.mdf
-rw-r-----. 1 mssql mssql 15400960 Aug 26 23:22 msdbdata.mdf
-rw-r-----. 1 mssql mssql 786432 Aug 26 23:22 msdblog.ldf
-rw-r-----. 1 mssql mssql 8388608 Aug 26 23:22 tempdb.mdf
-rw-r-----. 1 mssql mssql 8388608 Aug 26 23:22 templog.ldf
-rw-r-----. 1 mssql mssql 8388608 Aug 27 01:22 TestDB_Data.mdf
-rw-r-----. 1 mssql mssql 8388608 Aug 27 01:22 TestDB_Log.ldf
[allan@FCIN2 ~]$ sqlcmd -Usa -PP@ssword1
1> DROP DATABASE TestDB
2> GO
1> exit
[allan@FCIN2 ~]$ sudo ls -l /var/opt/mssql/data
total 55168
-rw-r-----. 1 mssql mssql 4653056 Aug 27 01:18 master.mdf
-rw-r-----. 1 mssql mssql 2097152 Aug 27 01:23 mastlog.ldf
-rw-r-----. 1 mssql mssql 8388608 Aug 27 01:22 modellog.ldf
-rw-r-----. 1 mssql mssql 8388608 Aug 27 01:22 model.mdf
-rw-r-----. 1 mssql mssql 15400960 Aug 26 23:22 msdbdata.mdf
-rw-r-----. 1 mssql mssql 786432 Aug 26 23:22 msdblog.ldf
-rw-r-----. 1 mssql mssql 8388608 Aug 26 23:22 tempdb.mdf
-rw-r-----. 1 mssql mssql 8388608 Aug 26 23:22 templog.ldf
```

- Stop SQL Server and verify it is shut down.

```
sudo systemctl stop mssql-server
sudo systemctl status mssql-server
```

- If you are not creating any other NFS mounts, unmount the share. If you are, do not unmount.

```
sudo umount <IPAddressOfNFSServer>:<FolderOnNFSServer> <FolderToMountIn>
```

<IPAddressOfNFSServer> is the IP address of the NFS server that you are going to use

<FolderOnNFSServer> is the name of the NFS share

<FolderMountedIn> is the folder created in the previous step.

4. For things other than system databases, such as user databases or backups, follow these steps. If only using the default location, skip to Step 5.

- Switch to be the superuser. You will not receive any acknowledgement if successful.

```
sudo -i
```

- Create a folder that will be used by SQL Server.

```
mkdir <FolderName>
```

<FolderName> is the name of the folder. The folder's full path needs to be specified if not in the right location. The following example creates a folder named /var/opt/mssql/userdata.

```
mkdir /var/opt/mssql/userdata
```

- Mount the NFS share in the folder that was created in the previous step. You will not receive any acknowledgement if successful.

```
Mount -t nfs4 <IPAddressOfNFSServer>:<FolderOnNFSServer> <FolderToMountIn> -o  
nfsvers=4.2,timeo=14,intr
```

<IPAddressOfNFSServer> is the IP address of the NFS server that you are going to use

<FolderOnNFSServer> is the name of the NFS share

<FolderToMountIn> is the folder created in the previous step. Below is an example.

```
mount -t nfs4 200.201.202.63:/var/nfs/fci2 /var/opt/mssql/userdata -o nfsvers=4.2,timeo=14,intr
```

- Check to see that the mount was successful by issuing mount with no switches.
- Type exit to no longer be the superuser.
- To test, create a database in that folder. The following example uses sqlcmd to create a database, switch context to it, verify the files exist at the OS level, and then deletes the temporary location. You can use SSMS.

```
[allan@FCIN2 ~]$ sqlcmd -Usa -PP@ssword1  
1> drop database TestDB  
2> GO  
1> CREATE DATABASE TestDB ON  
2> (NAME = TestDB_Data, FILENAME = '/var/opt/mssql/userdata/TestDB_Data.mdf')  
3> LOG ON (NAME = TestDB_Log, FILENAME = '/var/opt/mssql/userdata/TestDB_Log.ldf')  
4> go  
1> use TestDB  
2> GO  
Changed database context to 'TestDB'.  
1> exit  
[allan@FCIN2 ~]$ sudo ls /var/opt/mssql/userdata  
lost+found TestDB_Data.mdf TestDB_Log.ldf  
[allan@FCIN2 ~]$ sqlcmd -Usa -PP@ssword1  
1> drop database TestDB  
2> go  
1> exit  
[allan@FCIN2 ~]$ sudo ls /var/opt/mssql/userdata  
lost+found
```

- Unmount the share

```
sudo umount <IPAddressOfNFSServer>:<FolderOnNFSServer> <FolderToMountIn>
```

<IPAddressOfNFSServer> is the IP address of the NFS server that you are going to use

<FolderOnNFSServer> is the name of the NFS share

<FolderMountedIn> is the folder created in the previous step. Below is an example.

5. Repeat the steps on the other node(s).

Next steps

[Configure failover cluster instance - SQL Server on Linux](#)

Configure failover cluster instance - SMB - SQL Server on Linux

2/14/2018 • 5 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Linux only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

This article explains how to configure SMB storage for a failover cluster instance (FCI) on Linux.

In the non-Windows world, SMB is often referred to as a Common Internet File System (CIFS) share and implemented via Samba. In the Windows world, accessing an SMB share is done this way: \\SERVERNAME\\SHARENAME. For Linux-based SQL Server installations, the SMB share must be mounted as a folder.

Important source and server information

Here are some tips and notes for successfully using SMB:

- The SMB share can be on Windows, Linux, or even from an appliance as long as it is using SMB 3.0 or higher. For more information on Samba and SMB 3.0, see [SMB 3.0](#) to see if your Samba implementation is compliant with SMB 3.0.
- The SMB share should be highly available.
- Security must be set properly on the SMB share. Below is an example from /etc/samba/smb.conf, where SQLData1 is the name of the share.

```
[SQLData]
path=/var/smb/SQLData
read only = no
browseable = yes
guest ok = no
writeable = yes
valid users = SQLSambaUser
```

Instructions

1. Choose one of the servers that will participate in the FCI configuration. It does not matter which one.
2. Get information about the mssql user.

```
sudo id mssql
```

Note the uid, gid, and groups.

3. Execute `sudo smbclient -L //NameOrIP/ShareName -U User`.

<NameOrIP> is the DNS name or IP address of the server hosting the SMB share.

<ShareName> is the name of the SMB share.

4. For system databases or anything stored in the default data location follow these steps. Otherwise skip to step 5.
 - Ensure that SQL Server is stopped on the server that you are working on.

```
sudo systemctl stop mssql-server  
sudo systemctl status mssql-server
```

- Switch fully to be the superuser. You will not receive any acknowledgement if successful.

```
sudo -i
```

- Switch to be the mssql user. You will not receive any acknowledgement if successful.

```
su mssql
```

- Create a temporary directory to store the SQL Server data and log files. You will not receive any acknowledgement if successful.

```
mkdir <TempDir>
```

<TempDir> is the name of the folder. The following example creates a folder named /var/opt/mssql/tmp.

```
mkdir /var/opt/mssql/tmp
```

- Copy the SQL Server data and log files to the temporary directory. You will not receive any acknowledgement if successful.

```
cp /var/opt/mssql/data/* <TempDir>
```

<TempDir> is the name of the folder from the previous step.

- Verify that the files are in the directory.

```
ls <TempDir>
```

<TempDir> is the name of the folder from Step d.

- Delete the files from the existing SQL Server data directory. You will not receive any acknowledgement if successful.

```
rm - f /var/opt/mssql/data/*
```

- Verify that the files have been deleted.

```
ls /var/opt/mssql/data
```

- Type exit to switch back to the root user.
- Mount the SMB share in the SQL Server data folder. You will not receive any acknowledgement if successful. This example shows the syntax for connecting to a Windows Server-based SMB 3.0 share.

```
Mount -t cifs //<ServerName>/<ShareName> /var/opt/mssql/data -o vers=3.0,username=<UserName>,password=<Password>,domain=<domain>,uid=<mssqlUID>,gid=<mssqlGID>,file_mode=0777,dir_mode=0777
```

<ServerName> is the name of the server with the SMB share

<ShareName> is the name of the share

<UserName> is the name of the user to access the share

<Password> is the password for the user

<domain> is the name of Active Directory

<mssqlUID> is the UID of the mssql user

<mssqlGID> is the GID of the mssql user

- Check to see that the mount was successful by issuing mount with no switches.

```
mount
```

- Switch to the mssql user. You will not receive any acknowledgement if successful.

```
su mssql
```

- Copy the files from the temporary directory /var/opt/mssql/tmp. You will not receive any acknowledgement if successful.

```
cp /var/opt/mssql/tmp/* /var/opt/mssql/data
```

- Verify the files are there.

```
ls /var/opt/mssql/data
```

- Enter exit to not be mssql
- Enter exit to not be root
- Start SQL Server. If everything was copied correctly and security applied correctly, SQL Server should show as started.

```
sudo systemctl start mssql-server
sudo systemctl status mssql-server
```

- To test further, create a database to ensure the permissions are fine. The following example uses Transact-SQL; you can use SSMS.

```
[allan@FCIN2 ~]$ sqlcmd -Usa -PP@ssword1
1> CREATE DATABASE TestDB
2> ON (NAME = TestDB_Data, FILENAME = '/var/opt/mssql/data/TestDB_Data.mdf')
3> LOG ON (NAME = TestDB_Log, FILENAME = '/var/opt/mssql/data/TestDB_Log.ldf')
4> GO
1> USE TestDB
2> GO
Changed database context to 'TestDB'.
1> exit
[allan@FCIN2 ~]$ sudo ls -l /var/opt/mssql/data
total 71552
-rwxrwxrwx. 1 mssql mssql 4653056 Aug 27 04:08 master.mdf
-rwxrwxrwx. 1 mssql mssql 2097152 Aug 27 04:08 mastlog.ldf
-rwxrwxrwx. 1 mssql mssql 8388608 Aug 27 04:08 modellog.ldf
-rwxrwxrwx. 1 mssql mssql 8388608 Aug 27 04:08 model.mdf
-rwxrwxrwx. 1 mssql mssql 15400960 Aug 27 04:08 msdbdata.mdf
-rwxrwxrwx. 1 mssql mssql 786432 Aug 27 04:08 msdblog.ldf
-rwxrwxrwx. 1 mssql mssql 8388608 Aug 27 04:16 tempdb.mdf
-rwxrwxrwx. 1 mssql mssql 8388608 Aug 27 04:16 templog.ldf
-rwxrwxrwx. 1 mssql mssql 8388608 Aug 27 04:18 TestDB_Data.mdf
-rwxrwxrwx. 1 mssql mssql 8388608 Aug 27 04:18 TestDB_Log.ldf
[allan@FCIN2 ~]$ sqlcmd -Usa -PP@ssword1
1> DROP DATABASE TestDB
2> GO
1> exit
[allan@FCIN2 ~]$ sudo ls -l /var/opt/mssql/data
total 55168
-rwxrwxrwx. 1 mssql mssql 4653056 Aug 27 04:08 master.mdf
-rwxrwxrwx. 1 mssql mssql 2097152 Aug 27 04:08 mastlog.ldf
-rwxrwxrwx. 1 mssql mssql 8388608 Aug 27 04:08 modellog.ldf
-rwxrwxrwx. 1 mssql mssql 8388608 Aug 27 04:08 model.mdf
-rwxrwxrwx. 1 mssql mssql 15400960 Aug 27 04:08 msdbdata.mdf
-rwxrwxrwx. 1 mssql mssql 786432 Aug 27 04:08 msdblog.ldf
-rwxrwxrwx. 1 mssql mssql 8388608 Aug 27 04:16 tempdb.mdf
-rwxrwxrwx. 1 mssql mssql 8388608 Aug 27 04:16 templog.ldf
```

- Stop SQL Server and verify it is shut down. If you are going to be adding or testing other disks, do not shut down SQL Server until those are added and tested.

```
sudo systemctl stop mssql-server
sudo systemctl status mssql-server
```

- Only if finished, unmount the share. If not, unmount after finishing testing/adding any additional disks.

```
sudo umount //<IPAddressOrServerName>/<ShareName> /<FolderMountedIn>
```

<IPAddressOrServerName> is the IP address or name of the SMB host

<ShareName> is the name of the share

<FolderMountedIn> is the name of the folder where SMB is mounted

- For things other than system databases, such as user databases or backups, follow these steps. If only using the default location, skip to Step 14.

- Switch to be the superuser. You will not receive any acknowledgement if successful.

```
sudo -i
```

- Create a folder that will be used by SQL Server.

```
mkdir <FolderName>
```

<FolderName> is the name of the folder. The folder's full path needs to be specified if not in the right location. The following example creates a folder named /var/opt/mssql/userdata.

```
mkdir /var/opt/mssql/userdata
```

- Mount the SMB share in the SQL Server data folder. You will not receive any acknowledgement if successful. This example shows the syntax for connecting to a Samba-based SMB 3.0 share.

```
Mount -t cifs //<ServerName>/<ShareName> <FolderName> -o vers=3.0,username=<UserName>,password=<Password>,uid=<mssqlUID>,gid=<mssqlGID>,file_mode=0777,dir_mode=0777
```

<ServerName> is the name of the server with the SMB share

<ShareName> is the name of the share

<FolderName> is the name of the folder created in the last step

<UserName> is the name of the user to access the share

<Password> is the password for the user

<mssqlUID> is the UID of the mssql user

<mssqlGID> is the GID of the mssql user.

- Check to see that the mount was successful by issuing mount with no switches.
- Type exit to no longer be the superuser.
- To test, create a database in that folder. The following example uses sqlcmd to create a database, switch context to it, verify the files exist at the OS level, and then deletes the temporary location. You can use SSMS.
- Unmount the share

```
sudo umount //<IPAddressOrServerName>/<ShareName> /<FolderMountedIn>
```

<IPAddressOrServerName> is the IP address or name of the SMB host

<ShareName> is the name of the share

<FolderMountedIn> is the name of the folder where SMB is mounted.

6. Repeat the steps on the other node(s).

You are now ready to configure the FCI.

Next steps

[Configure failover cluster instance - SQL Server on Linux](#)

Deploy a Pacemaker cluster for SQL Server on Linux

2/14/2018 • 6 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This tutorial documents the tasks required to deploy a Linux Pacemaker cluster for a SQL Server Always On availability group (AG) or failover cluster instance (FCI). Unlike the tightly coupled Windows Server/ SQL Server stack, Pacemaker cluster creation as well as availability group (AG) configuration on Linux can be done before or after installation of SQL Server. The integration and configuration of resources for the Pacemaker portion of an AG or FCI deployment is done after the cluster is configured.

IMPORTANT

An AG with a cluster type of None does *not* require a Pacemaker cluster, nor can it be managed by Pacemaker.

- Install the high availability add-on and install Pacemaker.
- Prepare the nodes for Pacemaker (RHEL and Ubuntu only).
- Create the Pacemaker cluster.
- Install the SQL Server HA and SQL Server Agent packages.

Prerequisite

[Install SQL Server 2017](#).

Install the high availability add-on

Use the following syntax to install the packages that make up the high availability (HA) add-on for each distribution of Linux.

Red Hat Enterprise Linux (RHEL)

1. Register the server using the following syntax. You are prompted for a valid username and password.

```
sudo subscription-manager register
```

2. List the available pools for registration.

```
``bash sudo subscription-manager list --available
```

3. Run the following command to associate RHEL high availability with the subscription

```
sudo subscription-manager attach --pool=<PoolID>
```

where *PoolId* is the pool ID for the high availability subscription from the previous step.

4. Enable the repository to be able to use the high availability add-on.

```
sudo subscription-manager repos --enable=rhel-ha-for-rhel-7-server-rpms
```

5. Install Pacemaker.

```
sudo yum install pacemaker pcs fence-agents-all resource-agents
```

Ubuntu

```
sudo apt-get install pacemaker pcs fence-agents resource-agents
```

SUSE Linux Enterprise Server (SLES)

Install the High Availability pattern in YaST or do it as part of the main installation of the server. The installation can be done with an ISO/DVD as a source or by getting it online.

NOTE

On SLES, the HA add-on gets initialized when the cluster is created.

Prepare the nodes for Pacemaker (RHEL and Ubuntu only)

Pacemaker itself uses a user created on the distribution named *hacluster*. The user gets created when the HA add-on is installed on RHEL and Ubuntu.

1. On each server that will serve as a node of the Pacemaker cluster, create the password for a user to be used by the cluster. The name used in the examples is *hacluster*, but any name can be used. The name and password must be the same on all nodes participating in the Pacemaker cluster.

```
sudo passwd hacluster
```

2. On each node that will be part of the Pacemaker cluster, enable and start the `pcsd` service with the following commands (RHEL and Ubuntu):

```
sudo systemctl enable pcsd
sudo systemctl start pcsd
```

Then execute

```
sudo systemctl status pcsd
```

to ensure that `pcsd` is started.

3. Enable the Pacemaker service on each possible node of the Pacemaker cluster.

```
sudo systemctl start pacemaker
```

On Ubuntu, you see an error:

pacemaker Default-Start contains no runlevels, aborting.

This error is a known issue. Despite the error, enabling the Pacemaker service is successful, and this bug will be fixed at some point in the future.

4. Next, create and start the Pacemaker cluster. There is one difference between RHEL and Ubuntu at this step.

While on both distributions, installing `pcs` configures a default configuration file for the Pacemaker cluster, on RHEL, executing this command destroys any existing configuration and creates a new cluster.

Create the Pacemaker cluster

This section documents how to create and configure the cluster for each distribution of Linux.

RHEL

1. Authorize the nodes

```
sudo pcs cluster auth <Node1 Node2 ... NodeN> -u hacluster
```

where *NodeX* is the name of the node.

2. Create the cluster

```
sudo pcs cluster setup --name <PMClusterName Nodelist> --start --all --enable
```

where *PMClusterName* is the name assigned to the Pacemaker cluster and *Nodelist* is the list of names of the nodes separated by a space.

Ubuntu

Configuring Ubuntu is similar to RHEL. However, there is one major difference: installing the Pacemaker packages creates a base configuration for the cluster, and enables and starts `pcsd`. If you try to configure the Pacemaker cluster by following the RHEL instructions exactly, you get an error. To fix this problem, perform the following steps:

1. Remove the default Pacemaker configuration from each node.

```
sudo pcs cluster destroy
```

2. Follow the steps in the RHEL section for creating the Pacemaker cluster.

SLES

The process for creating a Pacemaker cluster is completely different on SLES than it is on RHEL and Ubuntu. The following steps document how to create a cluster with SLES.

1. Start the cluster configuration process by running

```
sudo ha-cluster-init
```

on one of the nodes. You may be prompted that NTP is not configured and that no watchdog device is found. That is fine for getting things up and running. Watchdog is related to STONITH if you use SLES's built-in fencing that is storage-based. NTP and watchdog can be configured later.

2. You are prompted to configure Corosync. You are asked for the network address to bind to, as well as the multicast address and port. The network address is the subnet that you are using; for example, 192.191.190.0. You can accept the defaults at every prompt, or change if necessary.
3. Next, you are asked if you want to configure SBD, which is the disk-based fencing. This configuration can be done later if desired. If SBD is not configured, unlike on RHEL and Ubuntu, `stonith-enabled` will by default be set to false.

4. Finally, you are asked if you want to configure an IP address for administration. This IP address is optional, but functions similar to the IP address for a Windows Server failover cluster (WSFC) in the sense that it creates an IP address in the cluster to be used for connecting to it via HA Web Konsole (HAWK). This configuration, too, is optional.

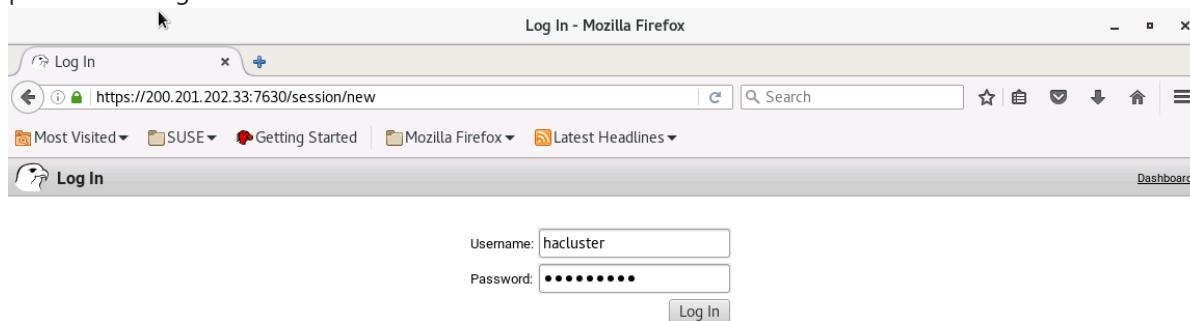
5. Ensure that the cluster is up and running by issuing

```
sudo crm status
```

6. Change the *hacluster* password with

```
sudo passwd hacluster
```

7. If you configured an IP address for administration, you can test it in a browser, which also tests the password change for *hacluster*.



8. On another SLES server that will be a node of the cluster, run

```
sudo ha-cluster-join
```

9. When prompted, enter the name or IP address of the server that was configured as the first node of the cluster in the previous steps. The server is added as a node to the existing cluster.

10. Verify the node was added by issuing

```
sudo crm status
```

11. Change the *hacluster* password with

```
sudo passwd hacluster
```

12. Repeat Steps 8-11 for all other servers to be added to the cluster.

Install the SQL Server HA and SQL Server Agent packages

Use the following commands to install the SQL Server HA package and SQL Server Agent, if they are not installed already. Installing the HA package after installing SQL Server requires a restart of SQL Server for it to be used. These instructions assume that the repositories for the Microsoft packages have already been set up, since SQL Server should be installed at this point.

NOTE

- If you will not use SQL Server Agent for log shipping or any other use, it does not have to be installed, so package *mssql-server-agent* can be skipped.
- The other optional packages for SQL Server on Linux, SQL Server Full-Text Search (*mssql-server-fts*) and SQL Server Integration Services (*mssql-server-is*), are not required for high availability, either for an FCI or an AG.

RHEL

```
sudo yum install mssql-server-ha mssql-server-agent  
sudo systemctl restart mssql-server
```

Ubuntu

```
sudo apt-get install mssql-server-ha mssql-server-agent  
sudo systemctl restart mssql-server
```

SLES

```
sudo zypper install mssql-server-ha mssql-server-agent  
sudo systemctl restart mssql-server
```

Next steps

In this tutorial, you learned how to deploy a Pacemaker cluster for SQL Server on Linux. You learned how to:

- Install the high availability add-on and install Pacemaker.
- Prepare the nodes for Pacemaker (RHEL and Ubuntu only).
- Create the Pacemaker cluster.
- Install the SQL Server HA and SQL Server Agent packages.

To create and configure an availability group for SQL Server on Linux, see:

[Create and configure an availability group for SQL Server on Linux.](#)

Create and configure an availability group for SQL Server on Linux

2/14/2018 • 16 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This tutorial covers how to create and configure an availability group (AG) for SQL Server on Linux. Unlike SQL Server 2016 and earlier on Windows, you can enable AGs with or without creating the underlying Pacemaker cluster first. Integration with the cluster, if needed, is not done until later.

The tutorial includes the following tasks:

- Enable availability groups.
- Create availability group endpoints and certificates.
- Use SQL Server Management Studio (SSMS) or Transact-SQL to create an availability group.
- Create the SQL Server login and permissions for Pacemaker.
- Create availability group resources in a Pacemaker cluster (External type only).

Prerequisite

- Deploy the Pacemaker high availability cluster as described in [Deploy a Pacemaker cluster for SQL Server on Linux](#).

Enable the availability groups feature

Unlike on Windows, you cannot use PowerShell or SQL Server Configuration Manager to enable the availability groups (AG) feature. Under Linux, you must use `mssql-conf` to enable the feature. There are two ways to enable the availability groups feature: use the `mssql-conf` utility, or edit the `mssql.conf` file manually.

IMPORTANT

The AG feature must be enabled for configuration-only replicas, even on SQL Server Express.

Use the `mssql-conf` utility

At a prompt, issue the following:

```
sudo /opt/mssql/bin/mssql-conf set hadr.hadrenabled 1
```

Edit the `mssql.conf` file

You can also modify the `mssql.conf` file, located under the `/var/opt/mssql` folder, to add the following lines:

```
[hadr]
hadr.hadrenabled = 1
```

Restart SQL Server

After enabling availability groups, as on Windows, you must restart SQL Server. That can be done by the following:

```
sudo systemctl restart mssql-server
```

Create the availability group endpoints and certificates

An availability group uses TCP endpoints for communication. Under Linux, endpoints for an AG are only supported if certificates are used for authentication. This means that the certificate from one instance must be restored on all other instances that will be replicas participating in the same AG. The certificate process is required even for a configuration-only replica.

Creating endpoints and restoring certificates can only be done via Transact-SQL. You can use non- SQL Server-generated certificates as well. You will also need a process to manage and replace any certificates that expire.

IMPORTANT

If you plan to use the SQL Server Management Studio wizard to create the AG, you still need to create and restore the certificates by using Transact-SQL on Linux.

For full syntax on the options available for the various commands (such as additional security), consult:

- [BACKUP CERTIFICATE](#)
- [CREATE CERTIFICATE](#)
- [CREATE ENDPOINT](#)

NOTE

Although you will be creating an availability group, the type of endpoint uses *FOR DATABASE_MIRRORING*, because some underlying aspects were once shared with that now-deprecated feature.

This example will create certificates for a three-node configuration. The instance names are LinAGN1, LinAGN2, and LinAGN3.

1. Execute the following on LinAGN1 to create the master key, certificate, and endpoint, as well as back up the certificate. For this example, the typical TCP port of 5022 is used for the endpoint.

```

CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<StrongPassword>';

GO

CREATE CERTIFICATE LinAGN1_Cert
WITH SUBJECT = 'LinAGN1 AG Certificate';

GO

BACKUP CERTIFICATE LinAGN1_Cert
TO FILE = '/var/opt/mssql/data/LinAGN1_Cert.cer';

GO

CREATE ENDPOINT AGEP
STATE = STARTED
AS TCP (
    LISTENER_PORT = 5022,
    LISTENER_IP = ALL)
FOR DATABASE_MIRRORING (
    AUTHENTICATION = CERTIFICATE LinAGN1_Cert,
    ROLE = ALL);

GO

```

2. Do the same on LinAGN2:

```

CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<StrongPassword>';

GO

CREATE CERTIFICATE LinAGN2_Cert
WITH SUBJECT = 'LinAGN2 AG Certificate';

GO

BACKUP CERTIFICATE LinAGN2_Cert
TO FILE = '/var/opt/mssql/data/LinAGN2_Cert.cer';

GO

CREATE ENDPOINT AGEP
STATE = STARTED
AS TCP (
    LISTENER_PORT = 5022,
    LISTENER_IP = ALL)
FOR DATABASE_MIRRORING (
    AUTHENTICATION = CERTIFICATE LinAGN2_Cert,
    ROLE = ALL);

GO

```

3. Finally, perform the same sequence on LinAGN3:

```

CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<StrongPassword>';

GO

CREATE CERTIFICATE LinAGN3_Cert
WITH SUBJECT = 'LinAGN3 AG Certificate';

GO

BACKUP CERTIFICATE LinAGN3_Cert
TO FILE = '/var/opt/mssql/data/LinAGN3_Cert.cer';

GO

CREATE ENDPOINT AGEP
STATE = STARTED
AS TCP (
    LISTENER_PORT = 5022,
    LISTENER_IP = ALL)
FOR DATABASE_MIRRORING (
    AUTHENTICATION = CERTIFICATE LinAGN3_Cert,
    ROLE = ALL);

GO

```

4. Using `scp` or another utility, copy the backups of the certificate to each node that will be part of the AG.

For this example:

- Copy LinAGN1_Cert.cer to LinAGN2 and LinAGN3
- Copy LinAGN2_Cert.cer to LinAGN1 and LinAGN3.
- Copy LinAGN3_Cert.cer to LinAGN1 and LinAGN2.

5. Change ownership and the group associated with the copied certificate files to `mssql`.

```
sudo chown mssql:mssql <CertFileName>
```

6. Create the instance-level logins and users associated with LinAGN2 and LinAGN3 on LinAGN1.

```

CREATE LOGIN LinAGN2_Login WITH PASSWORD = '<StrongPassword>';
CREATE USER LinAGN2_User FOR LOGIN LinAGN2_Login;

GO

CREATE LOGIN LinAGN3_Login WITH PASSWORD = '<StrongPassword>';
CREATE USER LinAGN3_User FOR LOGIN LinAGN3_Login;

GO

```

7. Restore LinAGN2_Cert and LinAGN3_Cert on LinAGN1. Having the other replicas' certificates is an important aspect of AG communication and security.

```
CREATE CERTIFICATE LinAGN2_Cert
AUTHORIZATION LinAGN2_User
FROM FILE = '/var/opt/mssql/data/LinAGN2_Cert.cer';

GO

CREATE CERTIFICATE LinAGN3_Cert
AUTHORIZATION LinAGN3_User
FROM FILE = '/var/opt/mssql/data/LinAGN3_Cert.cer';

GO
```

8. Grant the logins associated with LinAGN2 and LinAGN3 permission to connect to the endpoint on LinAGN1.

```
GRANT CONNECT ON ENDPOINT::AGEP TO LinAGN2_Login;

GO

GRANT CONNECT ON ENDPOINT::AGEP TO LinAGN3_Login;

GO
```

9. Create the instance-level logins and users associated with LinAGN1 and LinAGN3 on LinAGN2.

```
CREATE LOGIN LinAGN1_Login WITH PASSWORD = '<StrongPassword>';
CREATE USER LinAGN1_User FOR LOGIN LinAGN1_Login;

GO

CREATE LOGIN LinAGN3_Login WITH PASSWORD = '<StrongPassword>';
CREATE USER LinAGN3_User FOR LOGIN LinAGN3_Login;

GO
```

10. Restore LinAGN1_Cert and LinAGN3_Cert on LinAGN2.

```
CREATE CERTIFICATE LinAGN1_Cert
AUTHORIZATION LinAGN1_User
FROM FILE = '/var/opt/mssql/data/LinAGN1_Cert.cer';

GO

CREATE CERTIFICATE LinAGN3_Cert
AUTHORIZATION LinAGN3_User
FROM FILE = '/var/opt/mssql/data/LinAGN3_Cert.cer';

GO
```

11. Grant the logins associated with LinAGN1 and LinAGN3 permission to connect to the endpoint on LinAGN2.

```
GRANT CONNECT ON ENDPOINT::AGEP TO LinAGN1_Login;

GO

GRANT CONNECT ON ENDPOINT::AGEP TO LinAGN3_Login;

GO
```

12. Create the instance-level logins and users associated with LinAGN1 and LinAGN2 on LinAGN3.

```
CREATE LOGIN LinAGN1_Login WITH PASSWORD = '<StrongPassword>';
CREATE USER LinAGN1_User FOR LOGIN LinAGN1_Login;

GO

CREATE LOGIN LinAGN2_Login WITH PASSWORD = '<StrongPassword>';
CREATE USER LinAGN2_User FOR LOGIN LinAGN2_Login;

GO
```

13. Restore LinAGN1_Cert and LinAGN2_Cert on LinAGN3.

```
CREATE CERTIFICATE LinAGN1_Cert
AUTHORIZATION LinAGN1_User
FROM FILE = '/var/opt/mssql/data/LinAGN1_Cert.cer';

GO

CREATE CERTIFICATE LinAGN2_Cert
AUTHORIZATION LinAGN2_User
FROM FILE = '/var/opt/mssql/data/LinAGN2_Cert.cer';

GO
```

14. Grant the logins associated with LinAGN1 and LinAGN2 permission to connect to the endpoint on LinAGN3.

```
GRANT CONNECT ON ENDPOINT::AGEP TO LinAGN1_Login;

GO

GRANT CONNECT ON ENDPOINT::AGEP TO LinAGN2_Login;

GO
```

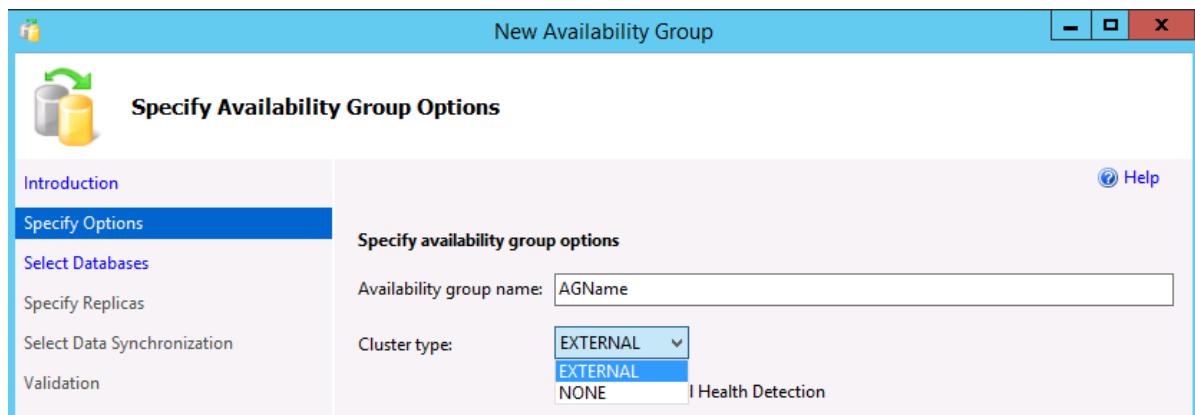
Create the availability group

This section covers how to use SQL Server Management Studio (SSMS) or Transact-SQL to create the availability group for SQL Server.

Use SQL Server Management Studio

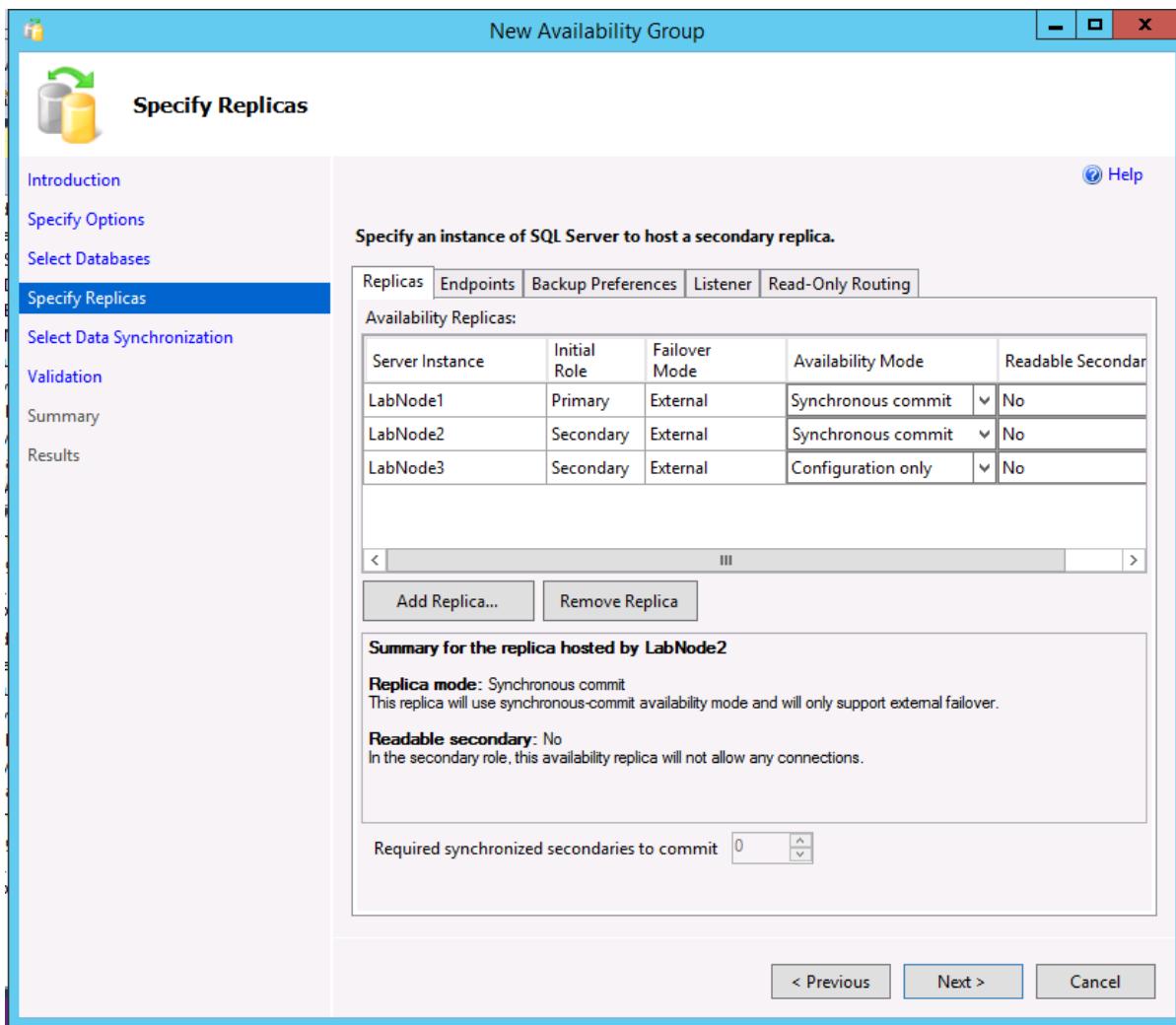
This section shows how to create an AG with a cluster type of External using SSMS with the New Availability Group Wizard.

1. In SSMS, expand **Always On High Availability**, right click **Availability Groups**, and select **New Availability Group Wizard**.
2. On the Introduction dialog, click **Next**.
3. In the Specify Availability Group Options dialog, enter a name for the availability group and select a cluster type of EXTERNAL or NONE in the dropdown. External should be used when Pacemaker will be deployed. None is for specialized scenarios, such as read scale out. Selecting the option for database level health detection is optional. For more information on this option, see [Availability group database level health detection failover option](#). Click **Next**.

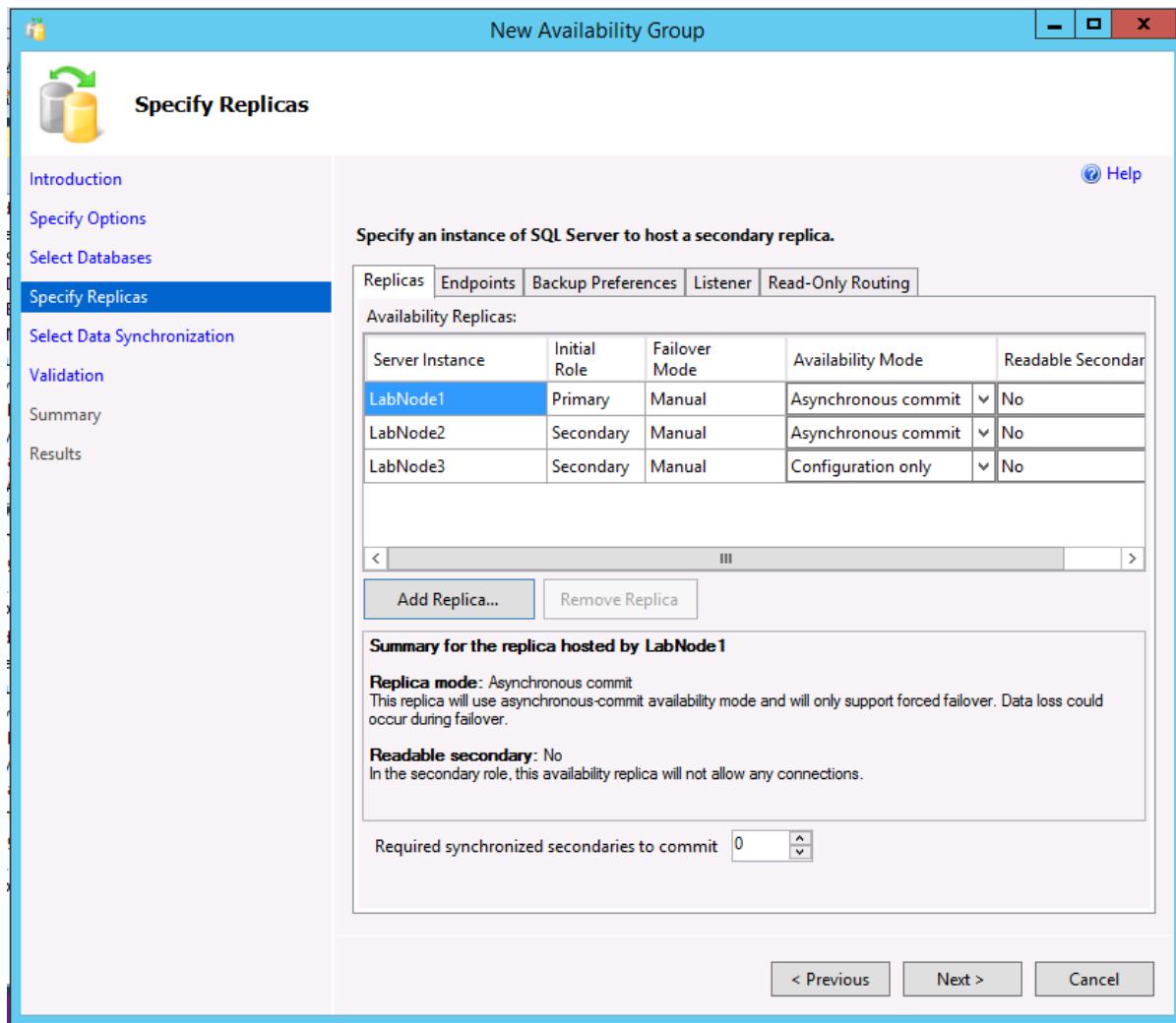


4. In the Select Databases dialog, select the database(s) that will participate in the AG. Each database must have a full backup before it can be added to an AG. Click **Next**.
5. In the Specify Replicas dialog, click **Add Replica**.
6. In the Connect to Server dialog, enter the name of the Linux instance of SQL Server that will be the secondary replica, and the credentials to connect. Click **Connect**.
7. Repeat the previous two steps for the instance that will contain a configuration-only replica or another secondary replica.
8. All three instances should now be listed on the Specify Replicas dialog. If using a cluster type of External, for the secondary replica that will be a true secondary, make sure the Availability Mode matches that of the primary replica and failover mode is set to External. For the configuration-only replica, select an availability mode of Configuration only.

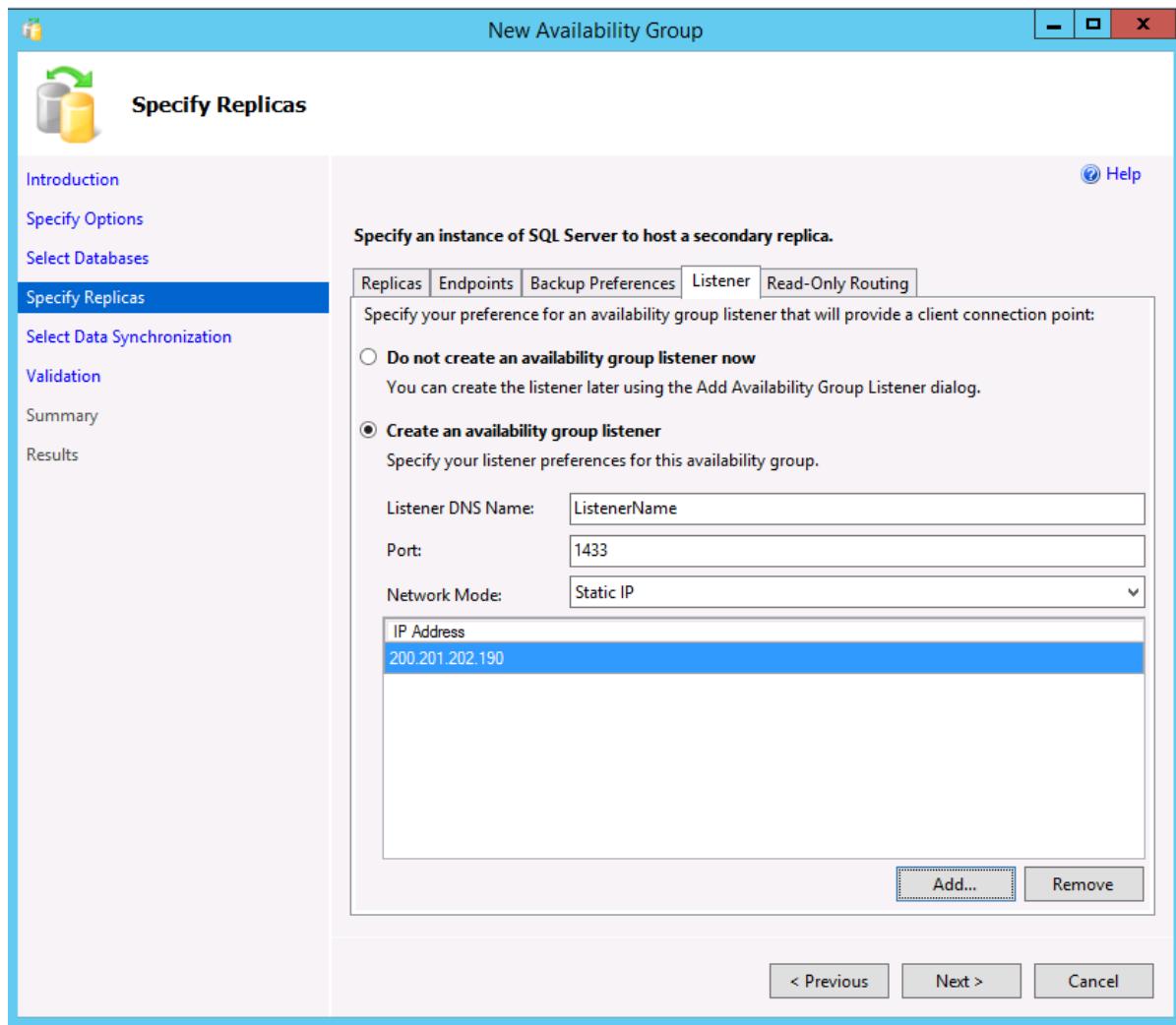
The following example shows an AG with two replicas, a cluster type of External, and a configuration-only replica.



The following example shows an AG with two replicas, a cluster type of None, and a configuration-only replica.



9. If you want to alter the backup preferences, click on the Backup Preferences tab. For more information on backup preferences with AGs, see [Configure backup on availability replicas](#).
10. If using readable secondaries or creating an AG with a cluster type of None for read-scale, you can create a listener by selecting the Listener tab. A listener can also be added later. To create a listener, choose the option **Create an availability group listener** and enter a name, a TCP/IP port, and whether to use a static or automatically assigned DHCP IP address. Remember that for an AG with a cluster type of None, the IP should be static and set to the primary's IP address.



11. If a listener is created for readable scenarios, SSMS 17.3 or later allows the creation of the read-only routing in the wizard. It can also be added later via SSMS or Transact-SQL. To add read-only routing now:
 - a. Select the Read-Only Routing tab.
 - b. Enter the URLs for the read-only replicas. These URLs are similar to the endpoints, except they use the port of the instance, not the endpoint.
 - c. Select each URL and from the bottom, select the readable replicas. To multi-select, hold down SHIFT or click-drag.
12. Click **Next**.
13. Choose how the secondary replica(s) will be initialized. The default is to use **automatic seeding**, which requires the same path on all servers participating in the AG. You can also have the wizard do a backup, copy, and restore (the second option); have it join if you have manually backed up, copied, and restored the database on the replica(s) (third option); or add the database later (last option). As with certificates, if you are manually making backups and copying them, permissions on the backup files needs to be set on the other replica(s). Click **Next**.
14. On the Validation dialog, if everything does not come back as Success, investigate. Some warnings are acceptable and not fatal, such as if you do not create a listener. Click **Next**.
15. On the Summary dialog, click **Finish**. The process to create the AG will now begin.
16. When the AG creation is complete, click **Close** on the Results. You can now see the AG on the replicas in the dynamic management views as well as under the Always On High Availability folder in SSMS.

Use Transact-SQL

This section shows examples of creating an AG using Transact-SQL. The listener and read-only routing can be configured after the AG is created. The AG itself can be modified with `ALTER AVAILABILITY GROUP`, but changing the cluster type cannot be done in SQL Server 2017. If you did not mean to create an AG with a cluster type of External, you must delete it and recreate it with a cluster type of None. More information and other options can be found at the following links:

- [CREATE AVAILABILITY GROUP \(Transact-SQL\)](#)
- [ALTER AVAILABILITY GROUP \(Transact-SQL\)](#)
- [Configure Read-Only Routing for an Availability Group \(SQL Server\)](#)
- [Create or Configure an Availability Group Listener \(SQL Server\)](#)

Example One – Two replicas with a configuration-only replica (External cluster type)

This example shows how to create a two-replica AG that uses a configuration-only replica.

1. Execute on the node that will be the primary replica containing the fully read/write copy of the database(s).

This example uses automatic seeding.

```
CREATE AVAILABILITY GROUP [<AGName>]
WITH (CLUSTER_TYPE = EXTERNAL)
FOR DATABASE <DBName>
REPLICA ON N'LinAGN1' WITH (
    ENDPOINT_URL = N'TCP://LinAGN1.FullyQualified.Name:5022',
    FAILOVER_MODE = EXTERNAL,
    AVAILABILITY_MODE = SYNCHRONOUS_COMMIT),
N'LinAGN2' WITH (
    ENDPOINT_URL = N'TCP://LinAGN2.FullyQualified.Name:5022',
    FAILOVER_MODE = EXTERNAL,
    AVAILABILITY_MODE = SYNCHRONOUS_COMMIT,
    SEEDING_MODE = AUTOMATIC),
N'LinAGN3' WITH (
    ENDPOINT_URL = N'TCP://LinAGN3.FullyQualified.Name:5022',
    AVAILABILITY_MODE = CONFIGURATION_ONLY);

GO
```

2. In a query window connected to the other replica, execute the following to join the replica to the AG and initiate the seeding process from the primary to the secondary replica.

```
ALTER AVAILABILITY GROUP [<AGName>] JOIN WITH (CLUSTER_TYPE = EXTERNAL);

GO

ALTER AVAILABILITY GROUP [<AGName>] GRANT CREATE ANY DATABASE;

GO
```

3. In a query window connected to the configuration only replica, join it to the AG.

```
ALTER AVAILABILITY GROUP [<AGName>] JOIN WITH (CLUSTER_TYPE = EXTERNAL);

GO
```

Example Two – Three replicas with read-only routing (External cluster type)

This example shows three full replicas and how read-only routing can be configured as part of the initial AG creation.

1. Execute on the node that will be the primary replica containing the fully read/write copy of the database(s).
This example uses automatic seeding.

```

CREATE AVAILABILITY GROUP [<AGName>]
WITH (CLUSTER_TYPE = EXTERNAL)
FOR DATABASE <DBName>
REPLICA ON N'LinAGN1'
WITH (
    ENDPOINT_URL = N'TCP://LinAGN1.FullyQualified.Name:5022',
    FAILOVER_MODE = EXTERNAL,
    AVAILABILITY_MODE = SYNCHRONOUS_COMMIT,
    PRIMARY_ROLE (ALLOW_CONNECTIONS = READ_WRITE, READ_ONLY_ROUTING_LIST =
    ('LinAGN2.FullyQualified.Name', 'LinAGN3.FullyQualified.Name'))),
    SECONDARY_ROLE (ALLOW_CONNECTIONS = ALL, READ_ONLY_ROUTING_URL =
    N'TCP://LinAGN1.FullyQualified.Name:1433'),
N'LinAGN2' WITH (
    ENDPOINT_URL = N'TCP://LinAGN2.FullyQualified.Name:5022',
    FAILOVER_MODE = EXTERNAL,
    SEEDING_MODE = AUTOMATIC,
    AVAILABILITY_MODE = SYNCHRONOUS_COMMIT,
    PRIMARY_ROLE (ALLOW_CONNECTIONS = READ_WRITE, READ_ONLY_ROUTING_LIST =
    ('LinAGN1.FullyQualified.Name', 'LinAGN3.FullyQualified.Name'))),
    SECONDARY_ROLE (ALLOW_CONNECTIONS = ALL, READ_ONLY_ROUTING_URL =
    N'TCP://LinAGN2.FullyQualified.Name:1433'),
N'LinAGN3' WITH (
    ENDPOINT_URL = N'TCP://LinAGN3.FullyQualified.Name:5022',
    FAILOVER_MODE = EXTERNAL,
    SEEDING_MODE = AUTOMATIC,
    AVAILABILITY_MODE = SYNCHRONOUS_COMMIT,
    PRIMARY_ROLE (ALLOW_CONNECTIONS = READ_WRITE, READ_ONLY_ROUTING_LIST =
    ('LinAGN1.FullyQualified.Name', 'LinAGN2.FullyQualified.Name'))),
    SECONDARY_ROLE (ALLOW_CONNECTIONS = ALL, READ_ONLY_ROUTING_URL =
    N'TCP://LinAGN3.FullyQualified.Name:1433'))
LISTENER '<ListenerName>' (WITH IP = ('<IPAddress>', '<SubnetMask>'), Port = 1433);

GO

```

A few things to note about this configuration:

- *AGName* is the name of the availability group.
 - *DBName* is the name of the database that will be used with the availability group. It can also be a list of names separated by commas.
 - *ListenerName* is a name that is different than any of the underlying servers/nodes. It will be registered in DNS along with *IPAddress*.
 - *IPAddress* is an IP address that is associated with *ListenerName*. It is also unique and not the same as any of the servers/nodes. Applications and end users will use either *ListenerName* or *IPAddress* to connect to the AG.
 - *SubnetMask* is the subnet mask of *IPAddress*; for example, 255.255.255.0.
2. In a query window connected to the other replica, execute the following to join the replica to the AG and initiate the seeding process from the primary to the secondary replica.

```

ALTER AVAILABILITY GROUP [<AGName>] JOIN WITH (CLUSTER_TYPE = EXTERNAL);

GO

ALTER AVAILABILITY GROUP [<AGName>] GRANT CREATE ANY DATABASE;

GO

```

3. Repeat Step 2 for the third replica.

Example Three – Two replicas with read-only routing (None cluster type)

This example shows the creation of a two-replica configuration using a cluster type of None. It is used for the read

scale scenario where no failover is expected,. This creates the listener that is actually the primary replica, as well as the read-only routing, using the round robin functionality.

1. Execute on the node that will be the primary replica containing the fully read/write copy of the database(s).
This example uses automatic seeding.

```
CREATE AVAILABILITY GROUP [<AGName>]
WITH (CLUSTER_TYPE = NONE)
FOR DATABASE <DBName>
REPLICA ON N'LinAGN1'
WITH (
    ENDPOINT_URL = N'TCP://LinAGN1.FullyQualified.Name: <PortOfEndpoint>',
    FAILOVER_MODE = MANUAL,
    AVAILABILITY_MODE = ASYNCHRONOUS_COMMIT,
    PRIMARY_ROLE (ALLOW_CONNECTIONS = READ_WRITE, READ_ONLY_ROUTING_LIST =
    ('LinAGN1.FullyQualified.Name'.'LinAGN2.FullyQualified.Name')),
    SECONDARY_ROLE (ALLOW_CONNECTIONS = ALL, READ_ONLY_ROUTING_URL =
    'TCP://LinAGN1.FullyQualified.Name:<PortOfInstance>');
    N'LinAGN2' WITH (
        ENDPOINT_URL = N'TCP://LinAGN2.FullyQualified.Name:<PortOfEndpoint>',
        FAILOVER_MODE = MANUAL,
        SEEDING_MODE = AUTOMATIC,
        AVAILABILITY_MODE = ASYNCHRONOUS_COMMIT,
        PRIMARY_ROLE (ALLOW_CONNECTIONS = READ_WRITE, READ_ONLY_ROUTING_LIST =
        ('LinAGN1.FullyQualified.Name', 'LinAGN2.FullyQualified.Name')),
        SECONDARY_ROLE (ALLOW_CONNECTIONS = ALL, READ_ONLY_ROUTING_URL =N'TCP://LinAGN2.FullyQualified.Name:<PortOfInstance>');
        LISTENER '<ListenerName>' (WITH IP = ('<PrimaryReplicaIPAddress>', '<SubnetMask>'), Port =
        <PortOfListener>);

    GO
```

Where

- *AGName* is the name of the availability group.
- *DBName* is the name of the database that will be used with the availability group. It can also be a list of names separated by commas.
- *PortOfEndpoint* is the port number used by the endpoint created.
- *PortOfInstance* is the port number used by the instance of SQL Server.
- *ListenerName* is a name that is different than any of the underlying replicas but will not actually be used.
- *PrimaryReplicaIPAddress* is the IP address of the primary replica.
- *SubnetMask* is the subnet mask of *IPAddress*. For example, 255.255.255.0.

2. Join the secondary replica to the AG and initiate automatic seeding.

```
ALTER AVAILABILITY GROUP [<AGName>] JOIN WITH (CLUSTER_TYPE = NONE);

GO

ALTER AVAILABILITY GROUP [<AGName>] GRANT CREATE ANY DATABASE;

GO
```

Create the SQL Server login and permissions for Pacemaker

A Pacemaker high availability cluster underlying SQL Server on Linux needs access to the SQL Server instance, as well as permissions on the availability group itself. These steps create the login and the associated permissions, along with a file that tells Pacemaker how to log into SQL Server.

1. In a query window connected to the first replica, execute the following:

```
CREATE LOGIN PMLogin WITH PASSWORD '<StrongPassword>';

GO

GRANT VIEW SERVER STATE TO PMLogin;

GO

GRANT ALTER, CONTROL, VIEW DEFINITION ON AVAILABILITY GROUP::<AGThatWasCreated> TO PMLogin;

GO
```

2. On Node 1, enter the command

```
sudo emacs /var/opt/mssql/secrets/passwd
```

This will open the Emacs editor.

3. Enter the following two lines into the editor:

```
PMLogin
<StrongPassword>
```

4. Hold down the CTRL key and then press X, then C, to exit and save the file.

5. Execute

```
sudo chmod 400 /var/opt/mssql/secrets/passwd
```

to lock down the file.

6. Repeat Steps 1-5 on the other servers that will serve as replicas.

Create the availability group resources in the Pacemaker cluster (External only)

After an availability group is created in SQL Server, the corresponding resources must be created in Pacemaker, when a cluster type of External is specified. There are two resources associated with an AG: the AG itself and an IP address. Configuring the IP address resource is optional if you are not using the listener functionality, but is recommended.

The AG resource that is created is a special kind of resource called a clone. The AG resource essentially has copies on each node, and there is one controlling resource called the master. The master is associated with the server hosting the primary replica. The secondary replicas (regular or configuration-only) are considered to be slaves and can be promoted to master in a failover.

1. Create the AG resource with the following syntax:

Red Hat Enterprise Linux (RHEL) and Ubuntu

```
sudo pcs resource create <NameForAGResource> ocf:mssql:ag ag_name=<AGName> --master meta notify=true
```

NOTE

On RHEL 7.4, you may encounter a warning with the use of --master. To avoid this, use

```
sudo pcs resource create <NameForAGResource> ocf:mssql:ag ag_name=<AGName> master notify=true
```

SUSE Linux Enterprise Server (SLES)

```
primitive <NameForAGResource> \
  ocf:mssql:ag \
  params ag_name=<AGName>" \
  op start timeout=60s \
  op stop timeout=60s \
  op promote timeout=60s \
  op demote timeout=10s \
  op monitor timeout=60s interval=10s \
  op monitor timeout=60s interval=11s role="Master" \
  op monitor timeout=60s interval=12s role="Slave" \
  op notify timeout=60s
ms ms-ag_cluster <NameForAGResource> \
meta master-max="1" master-node-max="1" clone-max="3" \
clone-node-max="1" notify="true" \
commit
```

where *NameForAGResource* is the unique name given to this cluster resource for the AG, and *AGName* is the name of the AG that was created.

2. Create the IP address resource for the AG that will be associated with the listener functionality.

RHEL and Ubuntu

```
sudo pcs resource create <NameForIPResource> ocf:heartbeat:IPAddr2 ip=<IPAddress> cidr_netmask=<Netmask>
```

SLES

```
crm configure \
primitive <NameForIPResource> \
  ocf:heartbeat:IPAddr2 \
  params ip=<IPAddress> \
  cidr_netmask=<Netmask>
```

where *NameForIPResource* is the unique name for the IP resource, and *IPAddress* is the static IP address assigned to the resource. On SLES, you also need to provide the netmask. For example, 255.255.255.0 would have a value of 24 for *Netmask*.

3. To ensure that the IP address and the AG resource are running on the same node, a colocation constraint must be configured.

RHEL and Ubuntu

```
sudo pcs constraint colocation add <NameForIPResource> <NameForAGResource>-master INFINITY with-rsc-role=Master
```

SLES

```
crm configure <NameForConstraint> inf: \
<NameForIPResource> <NameForAGResource>:Master
commit
```

where *NameForIPResource* is the name for the IP resource, *NameForAGResource* is the name for the AG resource, and on SLES, *NameForConstraint* is the name for the constraint.

4. Create an ordering constraint to ensure that the AG resource is up and running before the IP address. While the colocation constraint implies an ordering constraint, this enforces it.

RHEL and Ubuntu

```
sudo pcs constraint order promote <NameForAGResource>-master then start <NameForIPResource>
```

SLES

```
crm configure \
order <NameForConstraint> inf: <NameForAGResource>:promote <NameForIPResource>:start
commit
```

where *NameForIPResource* is the name for the IP resource, *NameForAGResource* is the name for the AG resource, and on SLES, *NameForConstraint* is the name for the constraint.

Next steps

In this tutorial, you learned how to create and configure an availability group for SQL Server on Linux. You learned how to:

- Enable availability groups.
- Create AG endpoints and certificates.
- Use SQL Server Management Studio (SSMS) or Transact-SQL to create an AG.
- Create the SQL Server login and permissions for Pacemaker.
- Create AG resources in a Pacemaker cluster.

For most AG administration tasks, including upgrades and failing over, see:

[Operate HA availability group for SQL Server on Linux](#)

Configure a SQL Server container in Kubernetes for high availability

2/14/2018 • 7 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Linux only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

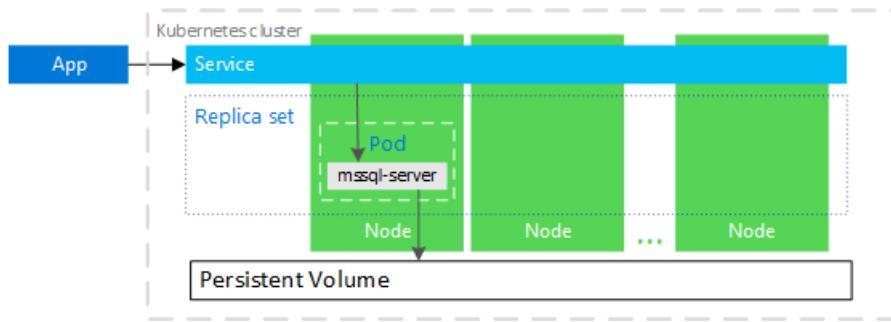
Learn how to configure a SQL Server instance on Kubernetes in Azure Container Service (AKS), with persistent storage for high availability (HA). The solution provides resiliency. If the SQL Server instance fails, Kubernetes automatically re-creates it in a new pod. AKS provides resiliency against a Kubernetes node failure.

This tutorial demonstrates how to configure a highly available SQL Server instance in containers that use AKS.

- Create an SA password
- Create storage
- Create the deployment
- Connect with SQL Server Management Studio (SSMS)
- Verify failure and recovery

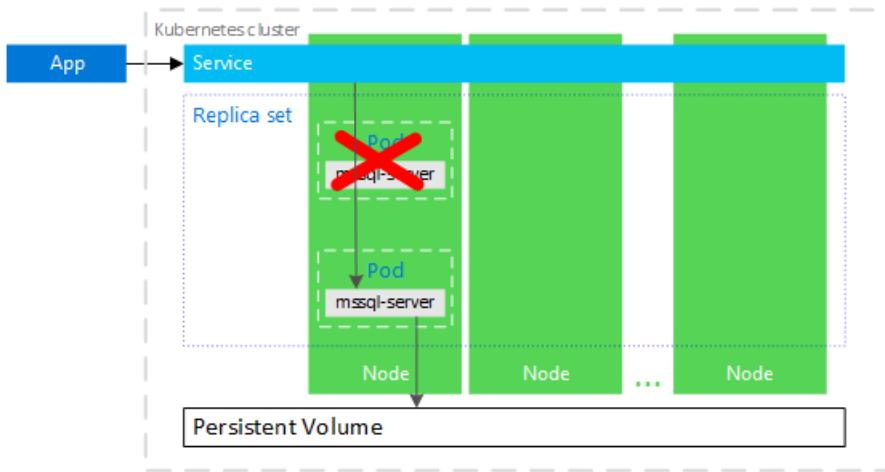
HA solution that uses Kubernetes running in Azure Container Service

Kubernetes 1.6 and later has support for [storage classes](#), [persistent volume claims](#), and the [Azure disk volume type](#). You can create and manage your SQL Server instances natively in Kubernetes. The example in this article shows how to create a [deployment](#) to achieve a high availability configuration similar to a shared disk failover cluster instance. In this configuration, Kubernetes plays the role of the cluster orchestrator. When a SQL Server instance in a container fails, the orchestrator bootstraps another instance of the container that attaches to the same persistent storage.

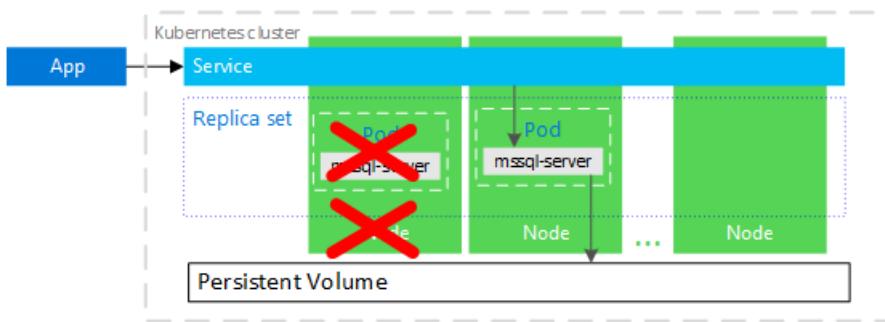


In the preceding diagram, `mssql-server` is a container in a [pod](#). Kubernetes orchestrates the resources in the cluster. A [replica set](#) ensures that the pod is automatically recovered after a node failure. Applications connect to the service. In this case, the service represents a load balancer that hosts an IP address that stays the same after failure of the `mssql-server`.

In the following diagram, the `mssql-server` container has failed. As the orchestrator, Kubernetes guarantees the correct count of healthy instances in the replica set, and starts a new container according to the configuration. The orchestrator starts a new pod on the same node, and `mssql-server` reconnects to the same persistent storage. The service connects to the re-created `mssql-server`.



In the following diagram, the node hosting the mssql-server container has failed. The orchestrator starts the new pod on a different node, and mssql-server reconnects to the same persistent storage. The service connects to the re-created mssql-server.



Prerequisites

- **Kubernetes cluster**

- The tutorial requires a Kubernetes cluster. The steps use `kubectl` to manage the cluster.
- See [Deploy an Azure Container Service \(AKS\) cluster](#) to create and connect to a single-node Kubernetes cluster in AKS with `kubectl`.

NOTE

To protect against node failure, a Kubernetes cluster requires more than one node.

- **Azure CLI 2.0.23**

- The instructions in this tutorial have been validated against Azure CLI 2.0.23.

Create an SA password

Create an SA password in the Kubernetes cluster. Kubernetes can manage sensitive configuration information, like passwords as [secrets](#).

The following command creates a password for the SA account:

```
kubectl create secret generic mssql --from-literal=SA_PASSWORD="MyC0m9l&xP@ssw0rd"
```

Replace `MyC0m9l&xP@ssw0rd` with a complex password.

To create a secret in Kubernetes named `mssql` that holds the value `MyC0m9l&xP@ssw0rd` for the `SA_PASSWORD`, run

the command.

Create storage

Configure a [persistent volume](#) and [persistent volume claim](#) in the Kubernetes cluster. Complete the following steps:

1. Create a manifest to define the storage class and the persistent volume claim. The manifest specifies the storage provisioner, parameters, and [reclaim policy](#). The Kubernetes cluster uses this manifest to create the persistent storage.

The following yaml example defines a storage class and persistent volume claim. The storage class provisioner is `azure-disk`, because this Kubernetes cluster is in Azure. The storage account type is `Standard_LRS`. The persistent volume claim is named `mssql-data`. The persistent volume claim metadata includes an annotation connecting it back to the storage class.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: azure-disk
  provisioner: kubernetes.io/azure-disk
parameters:
  storageaccounttype: Standard_LRS
  kind: Managed
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: mssql-data
  annotations:
    volume.beta.kubernetes.io/storage-class: azure-disk
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 8Gi
```

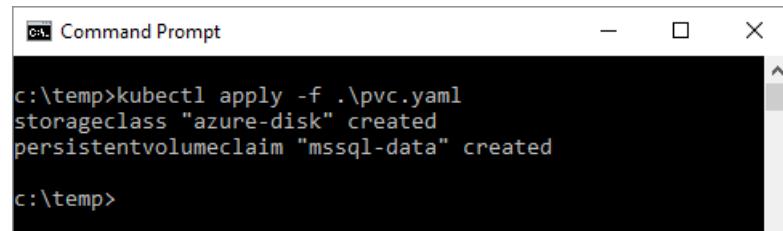
Save the file (for example, **pvc.yaml**).

2. Create the persistent volume claim in Kubernetes.

```
kubectl apply -f <Path to pvc.yaml file>
```

`<Path to pvc.yaml file>` is the location where you saved the file.

The persistent volume is automatically created as an Azure storage account, and bound to the persistent volume claim.



A screenshot of a Windows Command Prompt window titled "Command Prompt". The window shows the command `kubectl apply -f .\pvc.yaml` being run. The output indicates that a storageclass named "azure-disk" was created and a persistentvolumeclaim named "mssql-data" was created. The prompt then returns to `c:\temp>`.

```
c:\temp>kubectl apply -f .\pvc.yaml
storageclass "azure-disk" created
persistentvolumeclaim "mssql-data" created
c:\temp>
```

3. Verify the persistent volume claim.

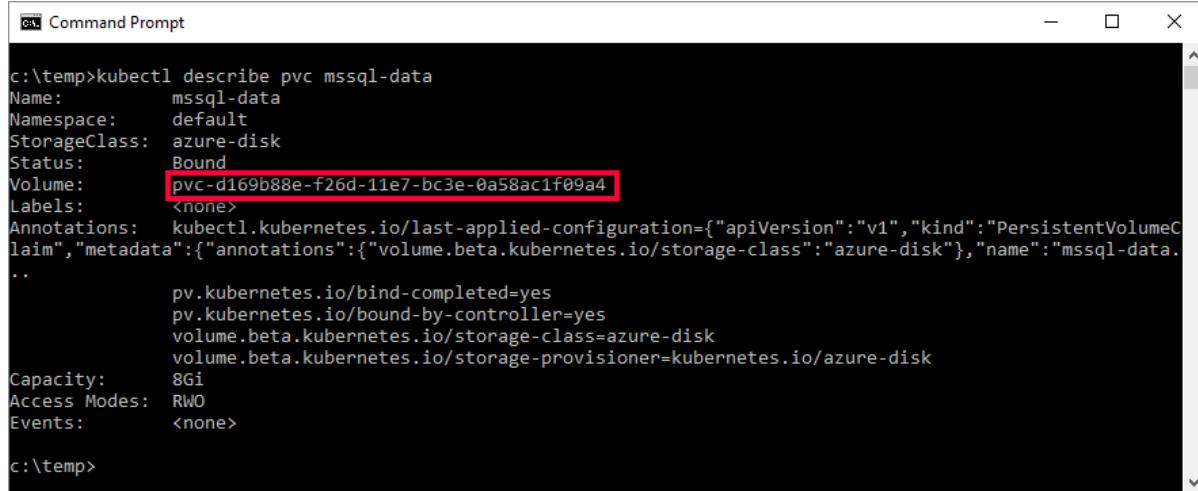
```
kubectl describe pvc <PersistentVolumeClaim>
```

<PersistentVolumeClaim> is the name of the persistent volume claim.

In the preceding step, the persistent volume claim is named `mssql-data`. To see the metadata about the persistent volume claim, run the following command:

```
kubectl describe pvc mssql-data
```

The returned metadata includes a value called `Volume`. This value maps to the name of the blob.



```
c:\temp>kubectl describe pvc mssql-data
Name:           mssql-data
Namespace:      default
StorageClass:   azure-disk
Status:         Bound
Volume:        pvc-d169b88e-f26d-11e7-bc3e-0a58ac1f09a4
Labels:         <none>
Annotations:   kubectl.kubernetes.io/last-applied-configuration={"apiVersion":"v1","kind":"PersistentVolumeClaim","metadata":{"annotations":{"volume.beta.kubernetes.io/storage-class":"azure-disk"},"name":"mssql-data"}...}
               pv.kubernetes.io/bind-completed=yes
               pv.kubernetes.io/bound-by-controller=yes
               volume.beta.kubernetes.io/storage-class:azure-disk
               volume.beta.kubernetes.io/storage-provisioner:kubernetes.io/azure-disk
Capacity:       8Gi
Access Modes:   RWO
Events:         <none>
c:\temp>
```

The value for volume matches part of the name of the blob in the following image from the Azure portal:



4. Verify the persistent volume.

```
kubectl describe pv
```

`kubectl` returns metadata about the persistent volume that was automatically created and bound to the persistent volume claim.

Create the deployment

In this example, the container hosting the SQL Server instance is described as a Kubernetes deployment object. The deployment creates a replica set. The replica set creates the pod.

In this step, create a manifest to describe the container based on the SQL Server [mssql-server-linux](#) Docker image. The manifest references the `mssql-server` persistent volume claim, and the `mssql` secret that you already applied to the Kubernetes cluster. The manifest also describes a `service`. This service is a load balancer. The load balancer guarantees that the IP address persists after SQL Server instance is recovered.

1. Create a manifest (a YAML file) to describe the deployment. The following example describes a deployment, including a container based on the SQL Server container image.

```

apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: mssql-deployment
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: mssql
    spec:
      terminationGracePeriodSeconds: 10
      containers:
        - name: mssql
          image: microsoft/mssql-server-linux
          ports:
            - containerPort: 1433
          env:
            - name: ACCEPT_EULA
              value: "Y"
            - name: SA_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mssql
                  key: SA_PASSWORD
          volumeMounts:
            - name: mssqldb
              mountPath: /var/opt/mssql
      volumes:
        - name: mssqldb
          persistentVolumeClaim:
            claimName: mssql-data
---
apiVersion: v1
kind: Service
metadata:
  name: mssql-deployment
spec:
  selector:
    app: mssql
  ports:
    - protocol: TCP
      port: 1433
      targetPort: 1433
  type: LoadBalancer

```

Copy the preceding code into a new file, named `sqldeployment.yaml`. Update the following values:

- `value: "Developer"` : Sets the container to run SQL Server Developer edition. Developer edition is not licensed for production data. If the deployment is for production use, set the appropriate edition (`Enterprise`, `Standard`, or `Express`).

NOTE

For more information, see [How to license SQL Server](#).

- `persistentVolumeClaim` : This value requires an entry for `claimName:` that maps to the name used for the persistent volume claim. This tutorial uses `mssql-data`.
- `name: SA_PASSWORD` : Configures the container image to set the SA password, as defined in this section.

```
valueFrom:  
  secretKeyRef:  
    name: mssql  
    key: SA_PASSWORD
```

When Kubernetes deploys the container, it refers to the secret named `mssql` to get the value for the password.

NOTE

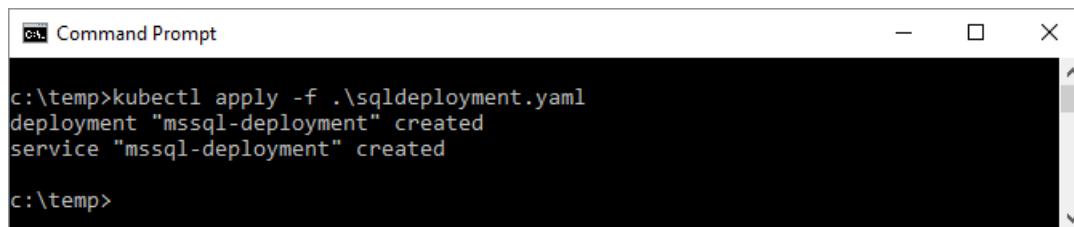
By using the `LoadBalancer` service type, the SQL Server instance is accessible remotely (via the internet) at port 1433.

Save the file (for example, **sqldeployment.yaml**).

2. Create the deployment.

```
kubectl apply -f <Path to sqldeployment.yaml file>
```

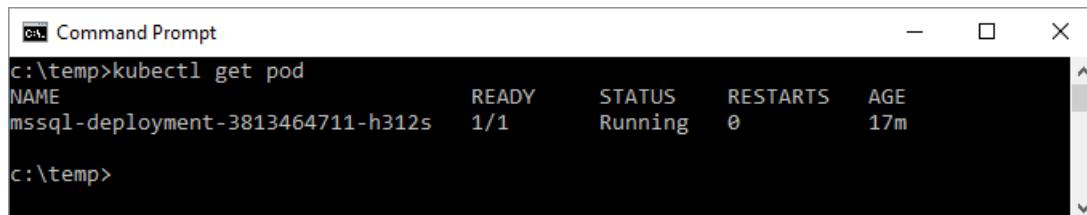
`<Path to sqldeployment.yaml file>` is the location where you saved the file.



```
c:\temp>kubectl apply -f ./sqldeployment.yaml  
deployment "mssql-deployment" created  
service "mssql-deployment" created  
  
c:\temp>
```

The deployment and service are created. The SQL Server instance is in a container, connected to persistent storage.

To view the status of the pod, type `kubectl get pod`.



```
c:\temp>kubectl get pod  
NAME           READY   STATUS    RESTARTS   AGE  
mssql-deployment-3813464711-h312s   1/1     Running   0          17m  
  
c:\temp>
```

In the preceding image, the pod has a status of `Running`. This status indicates that the container is ready. This may take several minutes.

NOTE

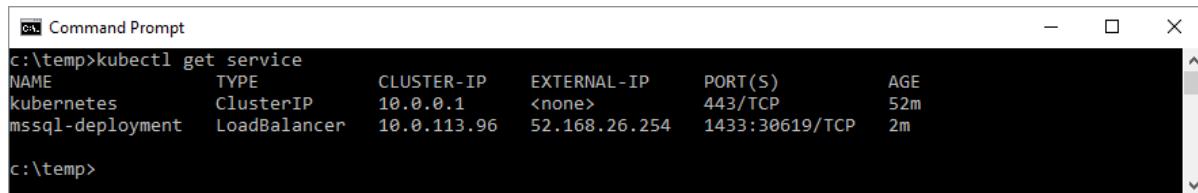
After the deployment is created, it can take a few minutes before the pod is visible. The delay is because the cluster pulls the `mssql-server-linux` image from the Docker hub. After the image is pulled the first time, subsequent deployments might be faster if the deployment is to a node that already has the image cached on it.

3. Verify the services are running. Run the following command:

```
kubectl get services
```

This command returns services that are running, as well as the internal and external IP addresses for the

services. Note the external IP address for the `mssql-deployment` service. Use this IP address to connect to SQL Server.



NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.0.0.1	<none>	443/TCP	52m
mssql-deployment	LoadBalancer	10.0.113.96	52.168.26.254	1433:30619/TCP	2m

For more information about the status of the objects in the Kubernetes cluster, run:

```
az aks browse --resource-group <MyResourceGroup> --name <MyKubernetesClustername>
```

Connect to the SQL Server instance

If you configured the container as described, you can connect with an application from outside the Azure virtual network. Use the `sa` account and the external IP address for the service. Use the password that you configured as the Kubernetes secret.

You can use the following applications to connect to the SQL Server instance.

- [SSMS](#)
- [SSDT](#)
- `sqlcmd`

To connect with `sqlcmd`, run the following command:

```
sqlcmd -S <External IP Address> -U sa -P "MyC0m9l&xP@ssw0rd"
```

Replace the following values:

- `<External IP Address>` with the IP address for the `mssql-deployment` service
- `MyC0m9l&xP@ssw0rd` with your password

Verify failure and recovery

To verify failure and recovery, you can delete the pod. Do the following steps:

1. List the pod running SQL Server.

```
kubectl get pods
```

Note the name of the pod running SQL Server.

2. Delete the pod.

```
kubectl delete pod mssql-deployment-0
```

`mssql-deployment-0` is the value returned from the previous step for pod name.

Kubernetes automatically re-creates the pod to recover a SQL Server instance, and connect to the persistent storage. Use `kubectl get pods` to verify that a new pod is deployed. Use `kubectl get services` to verify that the IP address for the new container is the same.

Summary

In this tutorial, you learned how to deploy SQL Server containers to a Kubernetes cluster for high availability.

- Create an SA password
- Create storage
- Create the deployment
- Connect with SQL Server Management Studios (SSMS)
- Verify failure and recovery

Next steps

[Introduction to Kubernetes](#)

Installation guidance for SQL Server on Linux

3/8/2018 • 6 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article provides guidance for installing, updating, and uninstalling SQL Server 2017 on Linux.

TIP

This guide covers several deployment scenarios. If you are only looking for step-by-step installation instructions, jump to one of the quickstarts:

- [RHEL quickstart](#)
- [SLES quickstart](#)
- [Ubuntu quickstart](#)
- [Docker quickstart](#)

For answers to frequently asked questions, see the [SQL Server on Linux FAQ](#).

Supported platforms

SQL Server 2017 is supported on Red Hat Enterprise Linux (RHEL), SUSE Linux Enterprise Server (SLES), and Ubuntu. It is also supported as a Docker image, which can run on Docker Engine on Linux or Docker for Windows/Mac.

PLATFORM	SUPPORTED VERSION(S)	GET
Red Hat Enterprise Linux	7.3 or 7.4	Get RHEL 7.4
SUSE Linux Enterprise Server	v12 SP2	Get SLES v12 SP2
Ubuntu	16.04	Get Ubuntu 16.04
Docker Engine	1.8+	Get Docker

NOTE

It is sometimes possible to install and run SQL Server on other closely-related Linux platforms, but SQL Server is only tested and supported on the platforms listed in the preceding table.

Microsoft supports deploying and managing SQL Server containers by using OpenShift and Kubernetes.

For the latest support policy for SQL Server 2017, see [Technical support policy for Microsoft SQL Server](#).

System requirements

SQL Server 2017 has the following system requirements for Linux:

Memory	2 GB
File System	XFS or EXT4 (other file systems, such as BTRFS , are unsupported)
Disk space	6 GB
Processor speed	2 GHz
Processor cores	2 cores
Processor type	x64-compatible only

If you use **Network File System (NFS)** remote shares in production, note the following support requirements:

- Use NFS version **4.2 or higher**. Older versions of NFS do not support required features, such as fallocate and sparse file creation, common to modern file systems.
- Locate only the **/var/opt/mssql** directories on the NFS mount. Other files, such as the SQL Server system binaries, are not supported.
- Ensure that NFS clients use the 'nolock' option when mounting the remote share.

Configure source repositories

When you install or upgrade SQL Server, you get the latest version of SQL Server 2017 from your configured Microsoft repository. The quickstarts use the **Cumulative Update (CU)** repository. But you can instead configure the **GDR** repository. For more information on repositories and how to configure them, see [Configure repositories for SQL Server on Linux](#).

IMPORTANT

If you previously installed a CTP or RC version of SQL Server 2017, you must remove the preview repository and register a General Availability (GA) one. For more information, see [Configure repositories for SQL Server on Linux](#).

Install SQL Server

You can install SQL Server on Linux from the command line. For instructions, see one of the following quickstarts:

- [Install on Red Hat Enterprise Linux](#)
- [Install on SUSE Linux Enterprise Server](#)
- [Install on Ubuntu](#)
- [Run on Docker](#)
- [Provision a SQL VM in Azure](#)

Update SQL Server

To update the **mssql-server** package to the latest release, use one of the following commands based on your platform:

PLATFORM	PACKAGE UPDATE COMMAND(S)
RHEL	<code>sudo yum update mssql-server</code>
SLES	<code>sudo zypper update mssql-server</code>
Ubuntu	<code>sudo apt-get update</code> <code>sudo apt-get install mssql-server</code>

These commands download the newest package and replace the binaries located under `/opt/mssql/`. The user generated databases and system databases are not affected by this operation.

Rollback SQL Server

To rollback or downgrade SQL Server to a previous release, use the following steps:

1. Identify the version number for the SQL Server package you want to downgrade to. For a list of package numbers, see the [Release notes](#).
2. Downgrade to a previous version of SQL Server. In the following commands, replace `<version_number>` with the SQL Server version number you identified in step one.

PLATFORM	PACKAGE UPDATE COMMAND(S)
RHEL	<code>sudo yum downgrade mssql-server-<version_number>.x86_64</code>
SLES	<code>sudo zypper install --oldpackage mssql-server-<version_number></code>
Ubuntu	<code>sudo apt-get install mssql-server=<version_number></code> <code>sudo systemctl start mssql-server</code>

NOTE

It is only supported to downgrade to a release within the same major version, such as SQL Server 2017.

Check installed SQL Server version

To verify your current version and edition of SQL Server on Linux, use the following procedure:

1. If not already installed, install the [SQL Server command-line tools](#).
2. Use **sqlcmd** to run a Transact-SQL command that displays your SQL Server version and edition.

```
sqlcmd -S localhost -U SA -Q 'select @@VERSION'
```

Uninstall SQL Server

To remove the **mssql-server** package on Linux, use one of the following commands based on your platform:

PLATFORM	PACKAGE REMOVAL COMMAND(S)
RHEL	<code>sudo yum remove mssql-server</code>
SLES	<code>sudo zypper remove mssql-server</code>
Ubuntu	<code>sudo apt-get remove mssql-server</code>

Removing the package does not delete the generated database files. If you want to delete the database files, use the following command:

```
sudo rm -rf /var/opt/mssql/
```

Unattended install

You can perform an unattended installation in the following way:

- Follow the initial steps in the [quickstarts](#) to register the repositories and install SQL Server.
- When you run `mssql-conf setup`, set [environment variables](#) and use the `-n` (no prompt) option.

The following example configures the Developer edition of SQL Server with the **MSSQL_PID** environment variable. It also accepts the EULA (**ACCEPT_EULA**) and sets the SA user password (**MSSQL_SA_PASSWORD**). The `-n` parameter performs an unprompted installation where the configuration values are pulled from the environment variables.

```
sudo MSSQL_PID=Developer ACCEPT_EULA=Y MSSQL_SA_PASSWORD='<YourStrong!Passw0rd>' /opt/mssql/bin/mssql-conf -n setup
```

You can also create a script that performs other actions. For example, you could install other SQL Server packages.

For a more detailed sample script, see the following examples:

- [Red Hat unattended installation script](#)
- [SUSE unattended installation script](#)
- [Ubuntu unattended installation script](#)

Offline install

If your Linux machine does not have access to the online repositories used in the [quick starts](#), you can download the package files directly. These packages are located in the Microsoft repository, <https://packages.microsoft.com>.

TIP

If you successfully installed with the steps in the quick starts, you do not need to download or manually install the SQL Server package(s). This section is only for the offline scenario.

- Download the database engine package for your platform.** Find package download links in the package details section of the [Release Notes](#).
- Move the downloaded package to your Linux machine.** If you used a different machine to download

the packages, one way to move the packages to your Linux machine is with the **scp** command.

3. Install the database engine package. Use one of the following commands based on your platform.

Replace the package file name in this example with the exact name you downloaded.

PLATFORM	PACKAGE REMOVAL COMMAND
RHEL	<code>sudo yum localinstall mssql-server_versionnumber.x86_64.rpm</code>
SLES	<code>sudo zypper install mssql-server_versionnumber.x86_64.rpm</code>
Ubuntu	<code>sudo dpkg -i mssql-server_versionnumber_amd64.deb</code>

NOTE

You can also install the RPM packages (RHEL and SLES) with the `rpm -ivh` command, but the commands in the previous table automatically install dependencies if available from approved repositories.

4. Resolve missing dependencies: You might have missing dependencies at this point. If not, you can skip this step. On Ubuntu, if you have access to approved repositories containing those dependencies, the easiest solution is to use the `apt-get -f install` command. This command also completes the installation of SQL Server. To manually inspect dependencies, use the following commands:

PLATFORM	LIST DEPENDENCIES COMMAND
RHEL	<code>rpm -qpR mssql-server_versionnumber.x86_64.rpm</code>
SLES	<code>rpm -qpR mssql-server_versionnumber.x86_64.rpm</code>
Ubuntu	<code>dpkg -I mssql-server_versionnumber_amd64.deb</code>

After resolving the missing dependencies, attempt to install the mssql-server package again.

5. Complete the SQL Server setup. Use **mssql-conf** to complete the SQL Server setup:

```
sudo /opt/mssql/bin/mssql-conf setup
```

Optional SQL Server features

After installation, you can also install or enable optional SQL Server features.

- [SQL Server command-line tools](#)
- [SQL Server Agent](#)
- [SQL Server Full Text Search](#)
- [SQL Server Integration Services \(Ubuntu\)](#)

ⓘ Get Help

- [UserVoice - Suggestion to improve SQL Server?](#)
- [Stack Overflow \(tag sql-server\) - ask SQL development questions](#)

- [Setup and Upgrade - MSDN Forum](#)
- [SQL Server Data Tools - MSDN forum](#)
- [Reddit - general discussion about SQL Server](#)
- [Microsoft SQL Server License Terms and Information](#)
- [Support options for business users](#)
- [Contact Microsoft](#)

TIP

For answers to frequently asked questions, see the [SQL Server on Linux FAQ](#).

Install sqlcmd and bcp the SQL Server command-line tools on Linux

2/22/2018 • 5 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The following steps install the command-line tools, Microsoft ODBC drivers, and their dependencies. The **mssql-tools** package contains:

- **sqlcmd**: Command-line query utility.
- **bcp**: Bulk import-export utility.

Install the tools for your platform:

- [Red Hat Enterprise Linux](#)
- [Ubuntu](#)
- [SUSE Linux Enterprise Server](#)
- [macOS](#)
- [Docker](#)

This article describes how to install the command-line tools. If you are looking for examples of how to use **sqlcmd** or **bcp**, see the [links](#) at the end of this topic.

Install tools on RHEL 7

Use the following steps to install the **mssql-tools** on Red Hat Enterprise Linux.

1. Enter superuser mode.

```
sudo su
```

2. Download the Microsoft Red Hat repository configuration file.

```
curl https://packages.microsoft.com/config/rhel/7/prod.repo > /etc/yum.repos.d/msprod.repo
```

3. Exit superuser mode.

```
exit
```

4. If you had a previous version of **mssql-tools** installed, remove any older unixODBC packages.

```
sudo yum remove unixODBC-utf16 unixODBC-utf16-devel
```

5. Run the following commands to install **mssql-tools** with the unixODBC developer package.

```
sudo yum install mssql-tools unixODBC-devel
```

NOTE

To update to the latest version of **mssql-tools** run the following commands:

```
sudo yum check-update  
sudo yum update mssql-tools
```

6. **Optional:** Add `/opt/mssql-tools/bin/` to your **PATH** environment variable in a bash shell.

To make **sqlcmd/bcp** accessible from the bash shell for login sessions, modify your **PATH** in the `~/.bash_profile` file with the following command:

```
echo 'export PATH="$PATH:/opt/mssql-tools/bin"' >> ~/.bash_profile
```

To make **sqlcmd/bcp** accessible from the bash shell for interactive/non-login sessions, modify the **PATH** in the `~/.bashrc` file with the following command:

```
echo 'export PATH="$PATH:/opt/mssql-tools/bin"' >> ~/.bashrc  
source ~/.bashrc
```

Install tools on Ubuntu 16.04

Use the following steps to install the **mssql-tools** on Ubuntu.

1. Import the public repository GPG keys.

```
curl https://packages.microsoft.com/keys/microsoft.asc | sudo apt-key add -
```

2. Register the Microsoft Ubuntu repository.

```
curl https://packages.microsoft.com/config/ubuntu/16.04/prod.list | sudo tee  
/etc/apt/sources.list.d/msprod.list
```

3. Update the sources list and run the installation command with the unixODBC developer package.

```
sudo apt-get update  
sudo apt-get install mssql-tools unixodbc-dev
```

NOTE

To update to the latest version of **mssql-tools** run the following commands:

```
sudo apt-get update  
sudo apt-get install mssql-tools
```

4. **Optional:** Add `/opt/mssql-tools/bin/` to your **PATH** environment variable in a bash shell.

To make **sqlcmd/bcp** accessible from the bash shell for login sessions, modify your **PATH** in the `~/.bash_profile` file with the following command:

```
echo 'export PATH="$PATH:/opt/mssql-tools/bin"' >> ~/.bash_profile
```

To make **sqlcmd/bcp** accessible from the bash shell for interactive/non-login sessions, modify the **PATH** in the **~/.bashrc** file with the following command:

```
echo 'export PATH="$PATH:/opt/mssql-tools/bin"' >> ~/.bashrc
source ~/.bashrc
```

Install tools on SLES 12

Use the following steps to install the **mssql-tools** on SUSE Linux Enterprise Server.

1. Add the Microsoft SQL Server repository to Zypper.

```
sudo zypper addrepo -fc https://packages.microsoft.com/config/sles/12/prod.repo
sudo zypper --gpg-auto-import-keys refresh
```

2. Install **mssql-tools** with the unixODBC developer package.

```
sudo zypper install mssql-tools unixODBC-devel
```

NOTE

To update to the latest version of **mssql-tools** run the following commands:

```
sudo zypper refresh
sudo zypper update mssql-tools
```

3. **Optional:** Add `/opt/mssql-tools/bin/` to your **PATH** environment variable in a bash shell.

To make **sqlcmd/bcp** accessible from the bash shell for login sessions, modify your **PATH** in the **~/.bash_profile** file with the following command:

```
echo 'export PATH="$PATH:/opt/mssql-tools/bin"' >> ~/.bash_profile
```

To make **sqlcmd/bcp** accessible from the bash shell for interactive/non-login sessions, modify the **PATH** in the **~/.bashrc** file with the following command:

```
echo 'export PATH="$PATH:/opt/mssql-tools/bin"' >> ~/.bashrc
source ~/.bashrc
```

Install tools on macOS

A preview of **sqlcmd** and **bcp** is now available on macOS. For more information, see the [announcement](#).

Install [Homebrew](#) if you don't have it already:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

To install the tools for Mac El Capitan and Sierra, use the following commands:

```
# brew untap microsoft/mssql-preview if you installed the preview version  
brew tap microsoft/mssql-release https://github.com/Microsoft/homebrew-mssql-release  
brew update  
brew install --no-sandbox mssql-tools  
#for silent install:  
#ACCEPT_EULA=y brew install --no-sandbox mssql-tools
```

Docker

Starting with SQL Server 2017 CTP 2.0, the SQL Server command-line tools are included in the Docker image. If you attach to the image with an interactive command-prompt, you can run the tools locally.

Offline installation

If your Linux machine does not have access to the online repositories used in the previous sections, you can download the package files directly. These packages are located in the Microsoft repository, <https://packages.microsoft.com>.

TIP

If you successfully installed with the steps in the previous sections, you do not need to download or manually install the package(s) below. This is only for the offline scenario.

The following table provides the location for the latest tools packages:

TOOLS PACKAGE	VERSION	DOWNLOAD
Red Hat RPM tools package	14.0.5.0-1	mssql-tools RPM package
SLES RPM tools package	14.0.5.0-1	mssql-tools RPM package
Ubuntu 16.04 Debian tools package	14.0.5.0-1	mssql-tools Debian package
Ubuntu 16.10 Debian tools package	14.0.5.0-1	mssql-tools Debian package

These packages depend on **msodbcsql**, which must be installed first. The **msodbcsql** pacakage also has a dependency on either **unixODBC-devel** (RPM) or **unixodbc-dev** (Debian). The location of the **msodbcsql** packages are listed in the following table:

MSODBCSQL PACKAGE	VERSION	DOWNLOAD
Red Hat RPM msodbcsql package	13.1.6.0-1	msodbcsql RPM package
SLES RPM msodbcsql package	13.1.6.0-1	msodbcsql RPM package
Ubuntu 16.04 Debian msodbcsql package	13.1.6.0-1	msodbcsql Debian package
Ubuntu 16.10 Debian msodbcsql package	13.1.6.0-1	msodbcsql Debian package

To manually install these packages, use the following steps:

- Move the downloaded packages to your Linux machine.** If you used a different machine to download the packages, one way to move the packages to your Linux machine is with the **scp** command.
- Install the and packages:** Install the **mssql-tools** and **msodbc** packages. If you get any dependency errors, ignore them until the next step.

PLATFORM	PACKAGE INSTALL COMMANDS
Red Hat	<pre>sudo yum localinstall msodbcsql-13.1.6.0-1.x86_64.rpm sudo yum localinstall mssql-tools-14.0.5.0-1.x86_64.rpm</pre>
SLES	<pre>sudo zypper install msodbcsql-13.1.6.0-1.x86_64.rpm sudo zypper install mssql-tools-14.0.5.0-1.x86_64.rpm</pre>
Ubuntu	<pre>sudo dpkg -i msodbcsql_13.1.6.0-1_amd64.deb sudo dpkg -i mssql-tools_14.0.5.0-1_amd64.deb</pre>

- Resolve missing dependencies:** You might have missing dependencies at this point. If not, you can skip this step. In some cases, you must manually locate and install these dependencies.

For RPM packages, you can inspect the required dependencies with the following commands:

```
rpm -qpR msodbcsql-13.1.6.0-1.x86_64.rpm
rpm -qpR mssql-tools-14.0.5.0-1.x86_64.rpm
```

For Debian packages, if you have access to approved repositories containing those dependencies, the easiest solution is to use the **apt-get** command:

```
sudo apt-get -f install
```

NOTE

This command completes the installation of the SQL Server packages as well.

If this does not work for your Debian package, you can inspect the required dependencies with the following commands:

```
dpkg -I msodbcsql_13.1.6.0-1_amd64.deb | grep "Depends:"
dpkg -I mssql-tools_14.0.5.0-1_amd64.deb | grep "Depends:"
```

Next steps

For an example of how to use **sqlcmd** to connect to SQL Server and create a database, see one of the following quickstarts:

- [Install on Red Hat Enterprise Linux](#)
- [Install on SUSE Linux Enterprise Server](#)
- [Install on Ubuntu](#)

- Run on Docker

For an example of how to use **bcp** to bulk import and export data, see [Bulk copy data to SQL Server on Linux](#).

Install SQL Server Agent on Linux

2/21/2018 • 2 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The [SQL Server Agent](#) runs scheduled SQL Server jobs. Starting with SQL Server 2017 CU4, SQL Server Agent is included with the **mssql-server** package and is disabled by default. For information on the features supported for this release of the SQL Server Agent along with version information, see the [Release Notes](#).

Install/Enable the SQL Server Agent:

- [For Versions 2017 CU4 and above, Enable the SQL Server Agent](#)
- [For Versions 2017 CU3 and Below, Install the SQL Server Agent](#)

For Versions 2017 CU4 and above, Enable the SQL Server Agent

To enable SQL Server Agent, follow the steps below.

```
sudo /opt/mssql/bin/mssql-conf set sqlagent.enabled true  
sudo systemctl restart mssql-server
```

NOTE

If you are upgrading from 2017 CU3 or below with Agent installed, SQL Server Agent will be enabled automatically and previous Agent packages will be uninstalled.

For Versions 2017 CU3 and Below, Install the SQL Server Agent

NOTE

The install instructions below apply to SQL Server Versions 2017 CU3 and below. Before installing SQL Server Agent, first [install SQL Server 2017](#). This configures the keys and repositories that you use when you install the **mssql-server-agent** package.

Install the SQL Server Agent for your platform:

- [Red Hat Enterprise Linux](#)
- [Ubuntu](#)
- [SUSE Linux Enterprise Server](#)

Install on RHEL

Use the following steps to install the **mssql-server-agent** on Red Hat Enterprise Linux.

```
sudo yum install mssql-server-agent  
sudo systemctl restart mssql-server
```

If you already have **mssql-server-agent** installed, you can update to the latest version with the following commands:

```
sudo yum check-update  
sudo yum update mssql-server-agent  
sudo systemctl restart mssql-server
```

If you need an offline installation, locate the SQL Server Agent package download in the [Release notes](#). Then use the same offline installation steps described in the article [Install SQL Server](#).

Install on Ubuntu

Use the following steps to install the **mssql-server-agent** on Ubuntu.

```
sudo apt-get update  
sudo apt-get install mssql-server-agent  
sudo systemctl restart mssql-server
```

If you already have **mssql-server-agent** installed, you can update to the latest version with the following commands:

```
sudo apt-get update  
sudo apt-get install mssql-server-agent  
sudo systemctl restart mssql-server
```

If you need an offline installation, locate the SQL Server Agent package download in the [Release notes](#). Then use the same offline installation steps described in the article [Install SQL Server](#).

Install on SLES

Use the following steps to install the **mssql-server-agent** on SUSE Linux Enterprise Server.

Install **mssql-server-agent**

```
sudo zypper install mssql-server-agent  
sudo systemctl restart mssql-server
```

If you already have **mssql-server-agent** installed, you can update to the latest version with the following commands:

```
sudo zypper refresh  
sudo zypper update mssql-server-agent  
sudo systemctl restart mssql-server
```

If you need an offline installation, locate the SQL Server Agent package download in the [Release notes](#). Then use the same offline installation steps described in the article [Install SQL Server](#).

Next steps

For more information on how to use SQL Server Agent to create, schedule, and run jobs, see [Run a SQL Server Agent job on Linux](#).

Install SQL Server Full-Text Search on Linux

2/14/2018 • 5 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The following steps install [SQL Server Full-Text Search \(mssql-server-fts\)](#) on Linux. Full-Text Search enables you to run full-text queries against character-based data in SQL Server tables. For known issues for this release, see the [Release Notes](#).

NOTE

Before installing SQL Server Full-Text Search, first [install SQL Server](#). This configures the keys and repositories that you use when installing the **mssql-server-fts** package.

Install SQL Server Full-Text Search for your platform:

- [Red Hat Enterprise Linux](#)
- [Ubuntu](#)
- [SUSE Linux Enterprise Server](#)

Install on RHEL

Use the following commands to install the **mssql-server-fts** on Red Hat Enterprise Linux.

```
sudo yum install -y mssql-server-fts
```

If you already have **mssql-server-fts** installed, you can update to the latest version with the following commands:

```
sudo yum check-update  
sudo yum update mssql-server-fts
```

If you need an offline installation, locate the Full-text Search package download in the [Release notes](#). Then use the same offline installation steps described in the article [Install SQL Server](#).

Install on Ubuntu

Use the following commands to install the **mssql-server-fts** on Ubuntu.

```
sudo apt-get update  
sudo apt-get install -y mssql-server-fts
```

If you already have **mssql-server-fts** installed, you can update to the latest version with the following commands:

```
sudo apt-get update  
sudo apt-get install -y mssql-server-fts
```

If you need an offline installation, locate the Full-text Search package download in the [Release notes](#). Then use the same offline installation steps described in the article [Install SQL Server](#).

Install on SLES

Use the following commands to install the **mssql-server-fts** on SUSE Linux Enterprise Server.

```
sudo zypper install mssql-server-fts
```

If you already have **mssql-server-fts** installed, you can update to the latest version with the following commands:

```
sudo zypper refresh  
sudo zypper update mssql-server-fts
```

If you need an offline installation, locate the Full-text Search package download in the [Release notes](#). Then use the same offline installation steps described in the article [Install SQL Server](#).

Supported languages

Full-Text Search uses [word breakers](#) that determine how to identify individual words based on language. You can get a list of registered word breakers by querying the **sys.fulltext_languages** catalog view. Word breakers for the following languages are installed with SQL Server 2017:

LANGUAGE	LANGUAGE ID
Neutral	0
Arabic	1025
Bengali (India)	1093
Bokmål	1044
Brazilian	1046
British English	2057
Bulgarian	1026
Catalan	1027
Chinese (Hong Kong SAR, PRC)	3076
Chinese (Macao SAR)	5124
Chinese (Singapore)	4100
Croatian	1050
Czech	1029
Danish	1030
Dutch	1043

LANGUAGE	LANGUAGE ID
English	1033
French	1036
German	1031
Greek	1032
Gujarati	1095
Hebrew	1037
Hindi	1081
Icelandic	1039
Indonesian	1057
Italian	1040
Japanese	1041
Kannada	1099
Korean	1042
Latvian	1062
Lithuanian	1063
Malay - Malaysia	1086
Malayalam	1100
Marathi	1102
Polish	1045
Portuguese	2070
Punjabi	1094
Romanian	1048
Russian	1049
Serbian (Cyrillic)	3098
Serbian (Latin)	2074

LANGUAGE	LANGUAGE ID
Simplified Chinese	2052
Slovak	1051
Slovenian	1060
Spanish	3082
Swedish	1053
Tamil	1097
Telugu	1098
Thai	1054
Traditional Chinese	1028
Turkish	1055
Ukrainian	1058
Urdu	1056
Vietnamese	1066

Filters

Full-Text Search also works with text stored in binary files. But in this case, an installed filter is required to process the file. For more information about filters, see [Configure and Manage Filters for Search](#).

You can see a list of installed filters by calling **sp_help_fulltext_system_components 'filter'**. For SQL Server 2017, the following filters are installed:

COMPONENT NAME	CLASS ID	VERSION
.a	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.ans	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.asc	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.ascx	E0CA5340-4534-11CF-B952-00AA0051FE20	12.0.6828.0
.asm	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0

COMPONENT NAME	CLASS ID	VERSION
.asp	E0CA5340-4534-11CF-B952-00AA0051FE20	12.0.6828.0
.aspx	E0CA5340-4534-11CF-B952-00AA0051FE20	12.0.6828.0
.asx	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.bas	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.bat	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.bcp	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.c	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.cc	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.cls	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.cmd	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.cpp	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.cs	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.csa	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.css	E0CA5340-4534-11CF-B952-00AA0051FE20	12.0.6828.0
.csv	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.cxx	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.dbs	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.def	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0

COMPONENT NAME	CLASS ID	VERSION
.dic	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.dos	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.dsp	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.dsw	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.ext	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.faq	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.fky	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.h	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.hhc	E0CA5340-4534-11CF-B952-00AA0051FE20	12.0.6828.0
.hpp	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.hta	E0CA5340-4534-11CF-B952-00AA0051FE20	12.0.6828.0
.htm	E0CA5340-4534-11CF-B952-00AA0051FE20	12.0.6828.0
.html	E0CA5340-4534-11CF-B952-00AA0051FE20	12.0.6828.0
.htt	E0CA5340-4534-11CF-B952-00AA0051FE20	12.0.6828.0
.htw	E0CA5340-4534-11CF-B952-00AA0051FE20	12.0.6828.0
.htx	E0CA5340-4534-11CF-B952-00AA0051FE20	12.0.6828.0
.hxx	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.i	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0

COMPONENT NAME	CLASS ID	VERSION
.ibq	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.ics	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.idl	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.idq	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.inc	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.inf	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.ini	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.inl	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.inx	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.jav	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.java	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.js	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.kci	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.lgn	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.log	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.lst	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.m3u	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.mak	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0

COMPONENT NAME	CLASS ID	VERSION
.mk	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.odc	E0CA5340-4534-11CF-B952-00AA0051FE20	12.0.6828.0
.odh	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.odl	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.pkgdef	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.pkgundef	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.pl	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.prc	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.rc	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.rc2	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.rct	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.reg	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.rgs	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.rtf	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.rul	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.s	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.scc	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.shtm	E0CA5340-4534-11CF-B952-00AA0051FE20	12.0.6828.0

COMPONENT NAME	CLASS ID	VERSION
.shtml	E0CA5340-4534-11CF-B952-00AA0051FE20	12.0.6828.0
.snippet	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.sol	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.sor	E0CA5340-4534-11CF-B952-00AA0051FE20	12.0.6828.0
.srf	E0CA5340-4534-11CF-B952-00AA0051FE20	12.0.6828.0
.stm	E0CA5340-4534-11CF-B952-00AA0051FE20	12.0.6828.0
.tab	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.tdl	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.tlh	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.tli	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.trg	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.txt	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.udf	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.udt	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.url	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.usr	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.vbs	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.viw	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0

COMPONENT NAME	CLASS ID	VERSION
.vsct	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.vsixlangpack	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.vsixmanifest	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.vspssc	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.vsscc	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.vssccc	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.wri	C1243CA0-BF96-11CD-B579-08002B30BFEB	12.0.6828.0
.wtx	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	12.0.6828.0
.xml	41B9BE05-B3AF-460C-BF0B-2CDD44A093B1	12.0.9735.0

Semantic search

[Semantic Search](#) builds on the Full-Text Search feature to extract and index statistically relevant *key phrases*. This enables you to query the meaning within documents in your database. It also helps to identify documents that are similar.

In order to use Semantic Search, you must first restore the Semantic Language Statistics database to your machine.

1. Use a tool, such as [sqlcmd](#), to run the following Transact-SQL command on your Linux SQL Server instance.

This command restores the Language Statistics database.

```
RESTORE DATABASE [semanticsdb] FROM
DISK = N'/opt/mssql/misc/semanticsdb.bak' WITH FILE = 1,
MOVE N'semanticsdb' TO N'/var/opt/mssql/data/semanticsDB.mdf',
MOVE N'semanticsdb_log' TO N'/var/opt/mssql/data/semanticsdb_log.ldf', NOUNLOAD, STATS = 5
GO
```

NOTE

If necessary, update the paths in the previous RESTORE command to adjust for your configuration.

2. Run the following Transact-SQL command to register the semantic language statistics database.

```
EXEC sp_fulltext_semantic_register_language_statistics_db @dbname = N'semanticsdb';
GO
```

Next steps

For information about Full-Text Search, see [SQL Server Full-Text Search](#).

Install SQL Server Integration Services (SSIS) on Linux

3/1/2018 • 2 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✔ SQL Server (Linux only) ✖ Azure SQL Database ✖ Azure SQL Data Warehouse ✖ Parallel Data Warehouse

Follow the steps in this article to install SQL Server Integration Services (`mssql-server-is`) on Linux. For info about the features supported in this release of Integration Services for Linux, see the [Release Notes](#).

Install SQL Server Integration Servers for your platform:

- [Ubuntu](#)
- [Red Hat Enterprise Linux](#)

Install SSIS on Ubuntu

To install the `mssql-server-is` package on Ubuntu, follow these steps:

1. Import the public repository GPG keys.

```
curl https://packages.microsoft.com/keys/microsoft.asc | sudo apt-key add -
```

2. Register the Microsoft SQL Server Ubuntu repository.

```
sudo add-apt-repository "$(curl https://packages.microsoft.com/config/ubuntu/16.04/mssql-server-2017.list)"
```

3. Run the following commands to install SQL Server Integration Services.

```
sudo apt-get update  
sudo apt-get install -y mssql-server-is
```

4. After installing Integration Services, run `ssis-conf`. For more info, see [Configure SSIS on Linux with ssis-conf](#).

```
sudo /opt/ssis/bin/ssis-conf setup
```

5. After the configuration is done, set the path.

```
export PATH=/opt/ssis/bin:$PATH
```

Update SSIS

If you already have `mssql-server-is` installed, you can update to the latest version with the following command:

```
sudo apt-get install mssql-server-is
```

Remove SSIS

To remove `mssql-server-is`, you can run following command:

```
sudo apt-get remove mssql-server-is
```

Install SSIS on RHEL

To install the `mssql-server-is` package on RHEL, follow these steps:

1. Download the Microsoft SQL Server Red Hat repository configuration file.

```
sudo curl -o /etc/yum.repos.d/mssql-server.repo https://packages.microsoft.com/config/rhel/7/mssql-server-2017.repo
```

2. Run the following commands to install SQL Server Integration Services.

```
sudo yum install -y mssql-server-is
```

3. After installation, run `ssis-conf`. For more info, see [Configure SSIS on Linux with ssis-conf](#).

```
sudo /opt/ssis/bin/ssis-conf setup
```

4. Once the configuration is done, set path.

```
export PATH=/opt/ssis/bin:$PATH
```

Update SSIS

If you already have `mssql-server-is` installed, you can update to the latest version with the following command:

```
sudo yum update mssql-server-is
```

Remove SSIS

To remove `mssql-server-is`, you can run following command:

```
sudo yum remove mssql-server-is
```

Unattended installation

To run an unattended installation when you run `ssis-conf setup`, do the following things:

1. Specify the `-n` (no prompt) option.
2. Provide required values by setting environment variables.

The following example does the following things:

- Installs SSIS.
- Specifies the Developer edition by providing a value for the `SSIS_PID` environment variable.
- Accepts the EULA by providing a value for the `ACCEPT_EULA` environment variable.
- Runs an unattended installation by specifying the `-n` (no prompt) option.

```
sudo SSIS_PID=Developer ACCEPT_EULA=Y /opt/ssis/bin/ssis-conf -n setup
```

Environment variables for unattended installation

ENVIRONMENT VARIABLE	DESCRIPTION
ACCEPT_EULA	Accepts the SQL Server license agreement when set to any value (for example, <code>Y</code>).
SSIS_PID	Sets the SQL Server edition or product key. Here are the possible values: Evaluation Developer Express Web Standard Enterprise A product key If you specify a product key, the product key must be in the form <code>#####-#####-#####-#####-#####</code> , where <code>#</code> is a letter or a number.

Next steps

To run SSIS packages on Linux, see [Extract, transform, and load data for SQL Server on Linux with SSIS](#).

To configure additional SSIS settings on Linux, see [Configure SQL Server Integration Services on Linux with ssis-conf](#).

Related content about SSIS on Linux

- [Extract, transform, and load data on Linux with SSIS](#)
- [Configure SQL Server Integration Services on Linux with ssis-conf](#)
- [Limitations and known issues for SSIS on Linux](#)
- [Schedule SQL Server Integration Services package execution on Linux with cron](#)

Configure repositories for installing and upgrading SQL Server on Linux

2/14/2018 • 4 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article describes how to configure the correct repository for SQL Server 2017 installations and upgrades on Linux.

IMPORTANT

If you previously installed a CTP or RC version of SQL Server 2017, you must use the steps in this article to register a General Availability (GA) repository and upgrade or reinstall. The preview releases of SQL Server 2017 are not supported and will expire.

Repositories

When you install SQL Server on Linux, you must configure a Microsoft repository. This repository is used to acquire the database engine package, **mssql-server**, and related SQL Server packages. There are currently three main repositories:

REPOSITORY	NAME	DESCRIPTION
Preview	mssql-server	Preview repository for CTP and RC releases of SQL Server. This repository is not supported for SQL Server 2017.
CU	mssql-server-2017	SQL Server 2017 Cumulative Update (CU) repository.
GDR	mssql-server-2017-gdr	SQL Server 2017 GDR repository for critical updates only.

Cumulative Update versus GDR

It is important to note that there are two main types of repositories for each distribution:

- **Cumulative Updates (CU):** The Cumulative Update (CU) repository contains packages for the base SQL Server release and any bug fixes or improvements since that release. Cumulative updates are specific to a release version, such as SQL Server 2017. They are released on a regular cadence.
- **GDR:** The GDR repository contains packages for the base SQL Server release and only critical fixes and security updates since that release. These updates are also added to the next CU release.

Each CU and GDR release contains the full SQL Server package and all previous updates for that repository. Updating from a GDR release to a CU release is supported by changing your configured repository for SQL Server. You can also [downgrade](#) to any release within your major version (ex: 2017).

NOTE

You can update from a GDR release to CU release at any time by changing repositories. Updating from a CU release to a GDR release is not supported.

Configure a repository

The following sections describe how to verify and configure a repository for the following supported platforms:

- [Red Hat Enterprise Server](#)
- [Ubuntu](#)
- [SUSE Linux Enterprise Server](#)

Configure RHEL repositories

Use the following steps to configure repositories on Red Hat Enterprise Server (RHEL).

Check for previously configured repositories (RHEL)

First verify whether you have already registered a SQL Server repository.

1. View the files in the **/etc/yum.repos.d** directory with the following command:

```
sudo ls /etc/yum.repos.d
```

2. Look for a file that configures the SQL Server directory, such as **mssql-server.repo**.

3. Print out the contents of the file.

```
sudo cat /etc/yum.repos.d/mssql-server.repo
```

4. The **name** property is the configured repository. You can identify it with the table in the [Repositories](#) section of this article.

Remove old repository (RHEL)

If necessary, remove the old repository with the following command.

```
sudo rm -rf /etc/yum.repos.d/mssql-server.repo
```

This command assumes that the file identified in the previous section was named **mssql-server.repo**.

Configure new repository (RHEL)

Configure the new repository to use for SQL Server installations and upgrades. Use one of the following commands to configure the repository of your choice.

REPOSITORY	COMMAND
CU	<pre>sudo curl -o /etc/yum.repos.d/mssql-server.repo https://packages.microsoft.com/config/rhel/7/mssql- server-2017.repo</pre>
GDR	<pre>sudo curl -o /etc/yum.repos.d/mssql-server.repo https://packages.microsoft.com/config/rhel/7/mssql- server-2017-gdr.repo</pre>

Configure SLES repositories

Use the following steps to configure repositories on SLES.

Check for previously configured repositories (SLES)

First verify whether you have already registered a SQL Server repository.

1. Use **zypper info** to get information about any previously configured repository.

```
sudo zypper info mssql-server
```

2. The **Repository** property is the configured repository. You can identify it with the table in the [Repositories](#) section of this article.

Remove old repository (SLES)

If necessary, remove the old repository. Use one of the following commands based on the type of previously configured repository.

REPOSITORY	COMMAND TO REMOVE
Preview	<pre>sudo zypper removerepo 'packages-microsoft-com-mssql-server'</pre>
CU	<pre>sudo zypper removerepo 'packages-microsoft-com-mssql-server-2017'</pre>
GDR	<pre>sudo zypper removerepo 'packages-microsoft-com-mssql-server-2017-gdr'</pre>

Configure new repository (SLES)

Configure the new repository to use for SQL Server installations and upgrades. Use one of the following commands to configure the repository of your choice.

REPOSITORY	COMMAND
CU	<pre>sudo zypper addrepo -fc https://packages.microsoft.com/config/sles/12/mssql-server-2017.repo</pre>
GDR	<pre>sudo zypper addrepo -fc https://packages.microsoft.com/config/sles/12/mssql-server-2017-gdr.repo</pre>

Configure Ubuntu repositories

Use the following steps to configure repositories on Ubuntu.

Check for previously configured repositories (Ubuntu)

First verify whether you have already registered a SQL Server repository.

1. View the contents of the **/etc/apt/sources.list** file.

```
sudo cat /etc/apt/sources.list
```

2. Examine the package URL for mssql-server. You can identify it with the table in the [Repositories](#) section of

this article.

Remove old repository (Ubuntu)

If necessary, remove the old repository. Use one of the following commands based on the type of previously configured repository.

REPOSITORY	COMMAND TO REMOVE
Preview	<pre>sudo add-apt-repository -r 'deb [arch=amd64] https://packages.microsoft.com/ubuntu/16.04/mssql- server xenial main'</pre>
CU	<pre>sudo add-apt-repository -r 'deb [arch=amd64] https://packages.microsoft.com/ubuntu/16.04/mssql- server-2017 xenial main'</pre>
GDR	<pre>sudo add-apt-repository -r 'deb [arch=amd64] https://packages.microsoft.com/ubuntu/16.04/mssql- server-2017-gdr xenial main'</pre>

Configure new repository (Ubuntu)

Configure the new repository to use for SQL Server installations and upgrades.

1. Import the public repository GPG keys.

```
sudo curl https://packages.microsoft.com/keys/microsoft.asc | sudo apt-key add -
```

2. Use one of the following commands to configure the repository of your choice.

REPOSITORY	COMMAND
CU	<pre>sudo add-apt-repository "\$(curl https://packages.microsoft.com/config/ubuntu/16.04/mssql- server-2017.list)"</pre>
GDR	<pre>sudo add-apt-repository "\$(curl https://packages.microsoft.com/config/ubuntu/16.04/mssql- server-2017-gdr.list)"</pre>

3. Run **apt-get update**.

```
sudo apt-get update
```

Next steps

After you have configured the correct repository, you can proceed to [install](#) or [update](#) SQL Server and any related packages from the new repository.

IMPORTANT

At this point, if you choose to use one of the installation articles, such as the [quickstarts](#), remember that you have already configured the target repository. Do not repeat that step in the tutorials. This is especially true if you configure the GDR repository, because the quickstarts use the CU repository.

For more information on how to install SQL Server 2017 on Linux, see [Installation guidance for SQL Server on](#)

Linux.

Configure SQL Server on Linux with the mssql-conf tool

2/21/2018 • 14 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

mssql-conf is a configuration script that installs with SQL Server 2017 for Red Hat Enterprise Linux, SUSE Linux Enterprise Server, and Ubuntu. You can use this utility to set the following parameters:

Agent	Enable SQL Server Agent
Collation	Set a new collation for SQL Server on Linux.
Customer feedback	Choose whether or not SQL Server sends feedback to Microsoft.
Database Mail Profile	Set the default database mail profile for SQL Server on Linux
Default data directory	Change the default directory for new SQL Server database data files (.mdf).
Default log directory	Changes the default directory for new SQL Server database log (.ldf) files.
Default master database file directory	Changes the default directory for the master database files on existing SQL installation.
Default master database file name	Changes the name of master database files.
Default dump directory	Change the default directory for new memory dumps and other troubleshooting files.
Default error log directory	Changes the default directory for new SQL Server ErrorLog, Default Profiler Trace, System Health Session XE, and Hekaton Session XE files.
Default backup directory	Change the default directory for new backup files.
Dump type	Choose the type of dump memory dump file to collect.
High availability	Enable Availability Groups.
Local Audit directory	Set a a directory to add Local Audit files.
Locale	Set the locale for SQL Server to use.
Memory limit	Set the memory limit for SQL Server.

TCP port	Change the port where SQL Server listens for connections.
TLS	Configure Transport Level Security.
Traceflags	Set the traceflags that the service is going to use.

TIP

Some of these settings can also be configured with environment variables. For more information, see [Configure SQL Server settings with environment variables](#).

Usage tips

- For Always On Availability Groups and shared disk clusters, always make the same configuration changes on each node.
- For the shared disk cluster scenario, do not attempt to restart the **mssql-server** service to apply changes. SQL Server is running as an application. Instead, take the resource offline and then back online.
- These examples run mssql-conf by specifying the full path: **/opt/mssql/bin/mssql-conf**. If you choose to navigate to that path instead, run mssql-conf in the context of the current directory: **./mssql-conf**.

Enable SQL Server Agent

The **sqlagent.enabled** setting enables [SQL Server Agent](#). By default, SQL Server Agent is disabled.

To change this setting, use the following steps:

1. Enable the SQL Server Agent:

```
sudo /opt/mssql/bin/mssql-conf set sqlagent.enabled true
```

2. Restart the SQL Server service:

```
sudo systemctl restart mssql-server
```

Change the SQL Server collation

The **set-collation** option changes the collation value to any of the supported collations.

1. First [backup any user databases](#) on your server.
2. Then use the [sp_detach_db](#) stored procedure to detach the user databases.
3. Run the **set-collation** option and follow the prompts:

```
sudo /opt/mssql/bin/mssql-conf set-collation
```

4. The mssql-conf utility will attempt to change to the specified collation value and restart the service. If there are any errors, it rolls back the collation to the previous value.
5. Restore your user database backups.

For a list of supported collations, run the `sys.fn_helpcollations` function: `SELECT Name FROM sys.fn_helpcollations()`.

Configure customer feedback

The **telemetry.customerfeedback** setting changes whether SQL Server sends feedback to Microsoft or not. By default, this value is set to **true**. To change the value, run the following commands:

1. Run the `mssql-conf` script as root with the `set` command for **telemetry.customerfeedback**. The following example turns off customer feedback by specifying **false**.

```
sudo /opt/mssql/bin/mssql-conf set telemetry.customerfeedback false
```

2. Restart the SQL Server service:

```
sudo systemctl restart mssql-server
```

For more information, see [Customer Feedback for SQL Server on Linux](#).

Change the default data or log directory location

The **filelocation.defaultdatadir** and **filelocation.defaultlogdir** settings change the location where the new database and log files are created. By default, this location is `/var/opt/mssql/data`. To change these settings, use the following steps:

1. Create the target directory for new database data and log files. The following example creates a new **/tmp/data** directory:

```
sudo mkdir /tmp/data
```

2. Change the owner and group of the directory to the **mssql** user:

```
sudo chown mssql /tmp/data
sudo chgrp mssql /tmp/data
```

3. Use `mssql-conf` to change the default data directory with the `set` command:

```
sudo /opt/mssql/bin/mssql-conf set filelocation.defaultdatadir /tmp/data
```

4. Restart the SQL Server service:

```
sudo systemctl restart mssql-server
```

5. Now all the database files for the new databases created will be stored in this new location. If you would like to change the location of the log (.ldf) files of the new databases, you can use the following "set" command:

```
sudo /opt/mssql/bin/mssql-conf set filelocation.defaultlogdir /tmp/log
```

6. This command also assumes that a `/tmp/log` directory exists, and that it is under the user and group **mssql**.

Change the default master database file directory location

The **filelocation.masterdatafile** and **filelocation.masterlogfile** setting changes the location where the SQL Server engine looks for the master database files. By default, this location is /var/opt/mssql/data.

To change these settings, use the following steps:

1. Create the target directory for new error log files. The following example creates a new **/tmp/masterdatabasedir** directory:

```
sudo mkdir /tmp/masterdatabasedir
```

2. Change the owner and group of the directory to the **mssql** user:

```
sudo chown mssql /tmp/masterdatabasedir  
sudo chgrp mssql /tmp/masterdatabasedir
```

3. Use mssql-conf to change the default master database directory for the master data and log files with the **set** command:

```
sudo /opt/mssql/bin/mssql-conf set filelocation.masterdatafile /tmp/masterdatabasedir/master.mdf  
sudo /opt/mssql/bin/mssql-conf set filelocation.masterlogfile /tmp/masterdatabasedir/mastlog.ldf
```

4. Stop the SQL Server service:

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

5. Move the master.mdf and masterlog.ldf:

```
sudo mv /var/opt/mssql/data/master.mdf /tmp/masterdatabasedir/master.mdf  
sudo mv /var/opt/mssql/data/mastlog.ldf /tmp/masterdatabasedir/mastlog.ldf
```

6. Start the SQL Server service:

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

NOTE

If SQL Server cannot find master.mdf and mastlog.ldf files in the specified directory, a templated copy of the system databases will be automatically created in the specified directory, and SQL Server will successfully start up. However, metadata such as user databases, server logins, server certificates, encryption keys, SQL agent jobs, or old SA login password will not be updated in the new master database. You will have to stop SQL Server and move your old master.mdf and mastlog.ldf to the new specified location and start SQL Server to continue using the existing metadata.

Change the name of master database files.

The **filelocation.masterdatafile** and **filelocation.masterlogfile** setting changes the location where the SQL Server engine looks for the master database files. By default, this location is /var/opt/mssql/data. To change these settings, use the following steps:

1. Stop the SQL Server service:

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

2. Use mssql-conf to change the expected master database names for the master data and log files with the **set** command:

```
sudo /opt/mssql/bin/mssql-conf set filelocation.masterdatafile /var/opt/mssql/data/masternew.mdf  
sudo /opt/mssql/bin/mssql-conf set filelocation.mastlogfile /var/opt/mssql/data /mastlognew.ldf
```

3. Change the name of the master database data and log files

```
sudo mv /var/opt/mssql/data/master.mdf /var/opt/mssql/data/masternew.mdf  
sudo mv /var/opt/mssql/data/mastlog.ldf /var/opt/mssql/data/mastlognew.ldf
```

4. Start the SQL Server service:

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

Change the default dump directory location

The **filelocation.defaultdumpdir** setting changes the default location where the memory and SQL dumps are generated whenever there is a crash. By default, these files are generated in `/var/opt/mssql/log`.

To set up this new location, use the following commands:

1. Create the target directory for new dump files. The following example creates a new **/tmp/dump** directory:

```
sudo mkdir /tmp/dump
```

2. Change the owner and group of the directory to the **mssql** user:

```
sudo chown mssql /tmp/dump  
sudo chgrp mssql /tmp/dump
```

3. Use mssql-conf to change the default data directory with the **set** command:

```
sudo /opt/mssql/bin/mssql-conf set filelocation.defaultdumpdir /tmp/dump
```

4. Restart the SQL Server service:

```
sudo systemctl restart mssql-server
```

Change the default error log file directory location

The **filelocation.errorlogfile** setting changes the location where the new error log, default profiler trace, system health session XE and Hekaton session XE files are created. By default, this location is `/var/opt/mssql/log`. The directory in which SQL errorlog file is set becomes the default log directory for other logs.

To change these settings:

1. Create the target directory for new error log files. The following example creates a new **/tmp/logs**

directory:

```
sudo mkdir /tmp/logs
```

2. Change the owner and group of the directory to the **mssql** user:

```
sudo chown mssql /tmp/logs  
sudo chgrp mssql /tmp/logs
```

3. Use mssql-conf to change the default errorlog filename with the **set** command:

```
sudo /opt/mssql/bin/mssql-conf set filelocation.errorlogfile /tmp/logs/errorlog
```

4. Restart the SQL Server service:

```
sudo systemctl restart mssql-server
```

Change the default backup directory location

The **filelocation.defaultbackupdir** setting changes the default location where the backup files are generated. By default, these files are generated in `/var/opt/mssql/data`.

To set up this new location, use the following commands:

1. Create the target directory for new backup files. The following example creates a new **/tmp/backup** directory:

```
sudo mkdir /tmp/backup
```

2. Change the owner and group of the directory to the **mssql** user:

```
sudo chown mssql /tmp/backup  
sudo chgrp mssql /tmp/backup
```

3. Use mssql-conf to change the default backup directory with the "set" command:

```
sudo /opt/mssql/bin/mssql-conf set filelocation.defaultbackupdir /tmp/backup
```

4. Restart the SQL Server service:

```
sudo systemctl restart mssql-server
```

Specify core dump settings

If an exception occurs in one of the SQL Server processes, SQL Server creates a memory dump.

There are two options for controlling the type of memory dumps that SQL Server collects:

coredump.coredumptype and **coredump.captureminiandfull**. These relate to the two phases of core dump capture.

The first phase capture is controlled by the **coredump.coredumpctype** setting, which determines the type of dump file generated during an exception. The second phase is enabled when the **coredump.captureminiandfull** setting. If **coredump.captureminiandfull** is set to true, the dump file specified by **coredump.coredumpctype** is generated and a second mini dump is also generated. Setting **coredump.captureminiandfull** to false disables the second capture attempt.

1. Decide whether to capture both mini and full dumps with the **coredump.captureminiandfull** setting.

```
sudo /opt/mssql/bin/mssql-conf set coredump.captureminiandfull <true or false>
```

Default: **false**

2. Specify the type of dump file with the **coredump.coredumpctype** setting.

```
sudo /opt/mssql/bin/mssql-conf set coredump.coredumpctype <dump_type>
```

Default: **miniplus**

The following table lists the possible **coredump.coredumpctype** values.

TYPE	DESCRIPTION
mini	Mini is the smallest dump file type. It uses the Linux system information to determine threads and modules in the process. The dump contains only the host environment thread stacks and modules. It does not contain indirect memory references or globals.
miniplus	MiniPlus is similar to mini, but it includes additional memory. It understands the internals of SQLPAL and the host environment, adding the following memory regions to the dump: <ul style="list-style-type: none">- Various globals- All memory above 64TB- All named regions found in /proc/\$pid/maps- Indirect memory from threads and stacks- Thread information- Associated Teb's and Peb's- Module Information- VMM and VAD tree
filtered	Filtered uses a subtraction-based design where all memory in the process is included unless specifically excluded. The design understands the internals of SQLPAL and the host environment, excluding certain regions from the dump.
full	Full is a complete process dump that includes all regions located in /proc/\$pid/maps . This is not controlled by coredump.captureminiandfull setting.

Set the default database mail profile for SQL Server on Linux

The **sqlagent.databasemailprofile** allows you to set the default DB Mail profile for email alerts.

```
sudo /opt/mssql/bin/mssql-conf set sqlagent.databasemailprofile <profile_name>
```

High Availability

The **hadr.hadrenabled** option enables availability groups on your SQL Server instance. The following command enables availability groups by setting **hadr.hadrenabled** to 1. You must restart SQL Server for the setting to take effect.

```
sudo /opt/mssql/bin/mssql-conf set hadr.hadrenabled 1  
sudo systemctl restart mssql-server
```

For information how this is used with availability groups, see the following two topics.

- [Configure Always On Availability Group for SQL Server on Linux](#)
- [Configure read-scale availability group for SQL Server on Linux](#)

Set local audit directory

The **telemetry.userrequestedlocalauditdirectory** setting enables Local Audit and lets you set the directory where the Local Audit logs are created.

1. Create a target directory for new Local Audit logs. The following example creates a new **/tmp/audit** directory:

```
sudo mkdir /tmp/audit
```

2. Change the owner and group of the directory to the **mssql** user:

```
sudo chown mssql /tmp/audit  
sudo chgrp mssql /tmp/audit
```

3. Run the mssql-conf script as root with the **set** command for **telemetry.userrequestedlocalauditdirectory**:

```
sudo /opt/mssql/bin/mssql-conf set telemetry.userrequestedlocalauditdirectory /tmp/audit
```

4. Restart the SQL Server service:

```
sudo systemctl restart mssql-server
```

For more information, see [Customer Feedback for SQL Server on Linux](#).

Change the SQL Server locale

The **language.lcid** setting changes the SQL Server locale to any supported language identifier (LCID).

1. The following example changes the locale to French (1036):

```
sudo /opt/mssql/bin/mssql-conf set language.lcid 1036
```

2. Restart the SQL Server service to apply the changes:

```
sudo systemctl restart mssql-server
```

Set the memory limit

The **memory.memorylimitmb** setting controls the amount physical memory (in MB) available to SQL Server. The default is 80% of the physical memory.

1. Run the mssql-conf script as root with the **set** command for **memory.memorylimitmb**. The following example changes the memory available to SQL Server to 3.25 GB (3328 MB).

```
sudo /opt/mssql/bin/mssql-conf set memory.memorylimitmb 3328
```

2. Restart the SQL Server service to apply the changes:

```
sudo systemctl restart mssql-server
```

Change the TCP port

The **network.tcpport** setting changes the TCP port where SQL Server listens for connections. By default, this port is set to 1433. To change the port, run the following commands:

1. Run the mssql-conf script as root with the "set" command for "network.tcpport":

```
sudo /opt/mssql/bin/mssql-conf set network.tcpport <new_tcp_port>
```

2. Restart the SQL Server service:

```
sudo systemctl restart mssql-server
```

3. When connecting to SQL Server now, you must specify the custom port with a comma (,) after the hostname or IP address. For example, to connect with SQLCMD, you would use the following command:

```
sqlcmd -S localhost,<new_tcp_port> -U test -P test
```

Specify TLS settings

The following options configure TLS for an instance of SQL Server running on Linux.

OPTION	DESCRIPTION
network.forceencryption	If 1, then SQL Server forces all connections to be encrypted. By default, this option is 0.
network.tlscert	The absolute path to the certificate file that SQL Server uses for TLS. Example: <code>/etc/ssl/certs/mssql.pem</code> The certificate file must be accessible by the mssql account. Microsoft recommends restricting access to the file using <code>chown mssql:mssql <file>; chmod 400 <file></code> .
network.tlskey	The absolute path to the private key file that SQL Server uses for TLS. Example: <code>/etc/ssl/private/mssql.key</code> The certificate file must be accessible by the mssql account. Microsoft recommends restricting access to the file using <code>chown mssql:mssql <file>; chmod 400 <file></code> .

OPTION	DESCRIPTION
network.tlsprotocols	A comma-separated list of which TLS protocols are allowed by SQL Server. SQL Server always attempts to negotiate the strongest allowed protocol. If a client does not support any allowed protocol, SQL Server rejects the connection attempt. For compatibility, all supported protocols are allowed by default (1.2, 1.1, 1.0). If your clients support TLS 1.2, Microsoft recommends allowing only TLS 1.2.
network.tlsciphers	Specifies which ciphers are allowed by SQL Server for TLS. This string must be formatted per OpenSSL's cipher list format . In general, you should not need to change this option. By default, the following ciphers are allowed: <pre>ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES128-SHA:AES256-GCM-SHA384:AES128-GCM-SHA256:AES256-SHA256:AES128-SHA256:AES128-SHA</pre>
network.kerberoskeytabfile	Path to the Kerberos keytab file

For an example of using the TLS settings, see [Encrypting Connections to SQL Server on Linux](#).

Enable/Disable traceflags

This **traceflag** option enables or disables traceflags for the startup of the SQL Server service. To enable/disable a traceflag use the following commands:

1. Enable a traceflag using the following command. For example, for Traceflag 1234:

```
sudo /opt/mssql/bin/mssql-conf traceflag 1234 on
```

2. You can enable multiple traceflags by specifying them separately:

```
sudo /opt/mssql/bin/mssql-conf traceflag 2345 3456 on
```

3. In a similar way, you can disable one or more enabled traceflags by specifying them and adding the **off** parameter:

```
sudo /opt/mssql/bin/mssql-conf traceflag 1234 2345 3456 off
```

4. Restart the SQL Server service to apply the changes:

```
sudo systemctl restart mssql-server
```

Remove a setting

To unset any setting made with `mssql-conf set`, call **mssql-conf** with the `unset` option and the name of the setting. This clears the setting, effectively returning it to its default value.

1. The following example clears the **network.tcpport** option.

```
sudo /opt/mssql/bin/mssql-conf unset network.tcpport
```

2. Restart the SQL Server service.

```
sudo systemctl restart mssql-server
```

View current settings

To view any configured settings, run the following command to output the contents of the **mssql.conf** file:

```
sudo cat /var/opt/mssql/mssql.conf
```

Note that any settings not shown in this file are using their default values. The next section provides a sample **mssql.conf** file.

mssql.conf format

The following **/var/opt/mssql/mssql.conf** file provides an example for each setting. You can use this format to manually make changes to the **mssql.conf** file as needed. If you do manually change the file, you must restart SQL Server before the changes are applied. To use the **mssql.conf** file with Docker, you must have Docker [persist your data](#). First add a complete **mssql.conf** file to your host directory and then run the container. There is an example of this in [Customer Feedback](#).

```
[EULA]
accepteula = Y

[coredump]
captureminiandfull = true
coredumpctype = full

[filelocation]
defaultbackupdir = /var/opt/mssql/data/
defaultdatadir = /var/opt/mssql/data/
defaultdumpdir = /var/opt/mssql/data/
defaultlogdir = /var/opt/mssql/data/

[hadr]
hadrenabled = 0

[language]
lcid = 1033

[memory]
memorylimitmb = 4096

[network]
forceencryption = 0
ipaddress = 10.192.0.0
kerberoskeytabfile = /var/opt/mssql/secrets/mssql.keytab
tcpport = 1401
tlscert = /etc/ssl/certs/mssql.pem
tlsciphers = ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA:ECDHE-RSA-AES128-SHA:AES256-GCM-SHA384:AES128-GCM-SHA256:AES256-SHA256:AES128-SHA256:AES256-SHA:AES128-SHA
tlskey = /etc/ssl/private/mssql.key
tlsprotocols = 1.2,1.1,1.0

[sqlagent]
databasemailprofile = default
errorlogfile = /var/opt/mssql/log/sqlagentlog.log
errorlogginglevel = 7

[telemetry]
customerfeedback = true
userrequestedlocalauditdirectory = /tmp/audit

[traceflag]
traceflag0 = 1204
traceflag1 = 2345
traceflag = 3456
```

Next steps

To instead use environment variables to make some of these configuration changes, see [Configure SQL Server settings with environment variables](#).

For other management tools and scenarios, see [Manage SQL Server on Linux](#).

Configure SQL Server settings with environment variables on Linux

2/21/2018 • 2 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Linux only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

You can use several different environment variables to configure SQL Server 2017 on Linux. These variables are used in two scenarios:

- To configure initial setup with the `mssql-conf setup` command.
- To configure a new [SQL Server container in Docker](#).

TIP

If you need to configure SQL Server after these setup scenarios, see [Configure SQL Server on Linux with the mssql-conf tool](#).

Environment variables

ENVIRONMENT VARIABLE	DESCRIPTION
ACCEPT_EULA	Accept the SQL Server license agreement when set to any value (for example, 'Y').
MSSQL_SA_PASSWORD	Configure the SA user password.
MSSQL_PID	Set the SQL Server edition or product key. Possible values include: Evaluation Developer Express Web Standard Enterprise A product key If specifying a product key, it must be in the form of #####-#####-#####-#####-#####, where '#' is a number or a letter.
MSSQL_LCID	Sets the language ID to use for SQL Server. For example 1036 is French.
MSSQL_COLLATION	Sets the default collation for SQL Server. This overrides the default mapping of language id (LCID) to collation.
MSSQL_MEMORY_LIMIT_MB	Sets the maximum amount of memory (in MB) that SQL Server can use. By default it is 80% of the total physical memory.

ENVIRONMENT VARIABLE	DESCRIPTION
MSSQL_TCP_PORT	Configure the TCP port that SQL Server listens on (default 1433).
MSSQL_IP_ADDRESS	Set the IP address. Currently, the IP address must be IPv4 style (0.0.0.0).
MSSQL_BACKUP_DIR	Set the Default backup directory location.
MSSQL_DATA_DIR	Change the directory where the new SQL Server database data files (.mdf) are created.
MSSQL_LOG_DIR	Change the directory where the new SQL Server database log (.ldf) files are created.
MSSQL_DUMP_DIR	Change the directory where SQL Server will deposit the memory dumps and other troubleshooting files by default.
MSSQL_ENABLE_HADR	Enable Availability Group. For example, '1' is enabled, and '0' is disabled
MSSQL_AGENT_ENABLED	Enable SQL Server Agent. For example, 'true' is enabled and 'false' is disabled. By default, agent is disabled.
MSSQL_MASTER_DATA_FILE	Sets the location of the master database data file.
MSSQL_MASTER_LOG_FILE	Sets the location of the master database log file.

Example: initial setup

This example runs `mssql-conf setup` with configured environment variables. The following environment variables are specified:

- **ACCEPT_EULA** accepts the end user license agreement.
- **MSSQL_PID** specifies the freely licensed Developer Edition of SQL Server for non-production use.
- **MSSQL_SA_PASSWORD** sets a strong password.
- **MSSQL_TCP_PORT** sets the TCP port that SQL Server listens on to 1234.

```
sudo ACCEPT_EULA='Y' MSSQL_PID='Developer' MSSQL_SA_PASSWORD='<YourStrong!Passw0rd>' MSSQL_TCP_PORT=1234
/opt/mssql/bin/mssql-conf setup
```

Example: Docker

This example docker command uses the following environment variables to create a new SQL Server 2017 container:

- **ACCEPT_EULA** accepts the end user license agreement.
- **MSSQL_PID** specifies the freely licensed Developer Edition of SQL Server for non-production use.
- **MSSQL_SA_PASSWORD** sets a strong password.
- **MSSQL_TCP_PORT** sets the TCP port that SQL Server listens on to 1234. This means that instead of mapping port 1433 (default) to a host port, the custom TCP port must be mapped with the `-p 1234:1234` command in this example.

If you are running Docker on Linux/macOS, use the following syntax with single quotes:

```
docker run -e ACCEPT_EULA=Y -e MSSQL_PID='Developer' -e MSSQL_SA_PASSWORD='<YourStrong!Passw0rd>' -e  
MSSQL_TCP_PORT=1234 -p 1234:1234 -d microsoft/mssql-server-linux:2017-latest
```

If you are running Docker on Windows, use the following syntax with double quotes:

```
docker run -e ACCEPT_EULA=Y -e MSSQL_PID="Developer" -e MSSQL_SA_PASSWORD="<YourStrong!Passw0rd>" -e  
MSSQL_TCP_PORT=1234 -p 1234:1234 -d microsoft/mssql-server-linux:2017-latest
```

NOTE

The process for running production editions in containers is slightly different. For more information, see [Run production container images](#).

Next steps

For other SQL Server settings not listed here, see [Configure SQL Server on Linux with the mssql-conf tool](#).

For more information on how to install and run SQL Server on Linux, see [Install SQL Server on Linux](#).

Configure SQL Server 2017 container images on Docker

2/26/2018 • 14 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article explains how to configure and use the [mssql-server-linux container image](#) with Docker. This image consists of SQL Server running on Linux based on Ubuntu 16.04. It can be used with the Docker Engine 1.8+ on Linux or on Docker for Mac/Windows.

NOTE

This article specifically focuses on using the mssql-server-linux image. The Windows image is not covered, but you can learn more about it on the [mssql-server-windows Docker Hub page](#).

Pull and run the container image

To pull and run the Docker container image for SQL Server 2017, follow the prerequisites and steps in the following quickstart:

- [Run the SQL Server 2017 container image with Docker](#)

This configuration article provides additional usage scenarios in the following sections.

Run production container images

The quickstart in the previous section runs the free Developer edition of SQL Server from Docker Hub. Most of the information still applies if you want to run production container images, such as Enterprise, Standard, or Web editions. However, there are a few differences that are outlined here.

- You can only use SQL Server in a production environment if you have a valid license. You can obtain a free SQL Server Express production license [here](#). SQL Server Standard and Enterprise Edition licenses are available through [Microsoft Volume Licensing](#).
- Production SQL Server container images must be pulled from [Docker Store](#). If you don't already have one, create an account on Docker Store.
- The Developer container image on Docker Store can be configured to run the production editions as well. Use the following steps to run production editions:

1. First, log in to your docker id from the command line.

```
docker login
```

2. Next, you need to obtain the free Developer container image on Docker Store. Go to <https://store.docker.com/images/mssql-server-linux>, click **Proceed to Checkout**, and follow the instructions.
3. Review the requirements and run procedures in the [quickstart](#). But there are two differences. You must pull the image **store/microsoft/mssql-server-linux:<tag-name>** from Docker Store. And

you must specify your production edition with the **MSSQL_PID** environment variable. The following example shows how to run the latest SQL Server 2017 container image for the Enterprise Edition:

```
docker run --name sqlenterprise \
-e 'ACCEPT_EULA=Y' -e 'MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>' \
-e 'MSSQL_PID=Enterprise' -p 1433:1433 \
-d store/microsoft/mssql-server-linux:2017-latest
```

```
docker run --name sqlenterprise \
-e "ACCEPT_EULA=Y" -e "MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>" \
-e "MSSQL_PID=Enterprise" -p 1433:1433 \
-d "store/microsoft/mssql-server-linux:2017-latest"
```

IMPORTANT

By passing the value **Y** to the environment variable **ACCEPT_EULA** and an edition value to **MSSQL_PID**, you are expressing that you have a valid and existing license for the edition and version of SQL Server that you intend to use. You also agree that your use of SQL Server software running in a Docker container image will be governed by the terms of your SQL Server license.

NOTE

For a full list of possible values for **MSSQL_PID**, see [Configure SQL Server settings with environment variables on Linux](#).

Connect and query

You can connect and query SQL Server in a container from either outside the container or from within the container. The following sections explain both scenarios.

Tools outside the container

You can connect to the SQL Server instance on your Docker machine from any external Linux, Windows, or macOS tool that supports SQL connections. Some common tools include:

- [sqlcmd](#)
- [Visual Studio Code](#)
- [SQL Server Management Studio \(SSMS\) on Windows](#)

The following example uses **sqlcmd** to connect to SQL Server running in a Docker container. The IP address in the connection string is the IP address of the host machine that is running the container.

```
sqlcmd -S 10.3.2.4 -U SA -P '<YourPassword>'
```

```
sqlcmd -S 10.3.2.4 -U SA -P "<YourPassword>"
```

If you mapped a host port that was not the default **1433**, add that port to the connection string. For example, if you specified `-p 1400:1433` in your `docker run` command, then connect by explicitly specify port 1400.

```
sqlcmd -S 10.3.2.4,1400 -U SA -P '<YourPassword>'
```

```
sqlcmd -S 10.3.2.4,1400 -U SA -P "<YourPassword>"
```

Tools inside the container

Starting with SQL Server 2017 CTP 2.0, the [SQL Server command-line tools](#) are included in the container image. If you attach to the image with an interactive command-prompt, you can run the tools locally.

1. Use the `docker exec -it` command to start an interactive bash shell inside your running container. In the following example `e69e056c702d` is the container ID.

```
docker exec -it e69e056c702d "bash"
```

TIP

You don't always have to specify the entire container id. You only have to specify enough characters to uniquely identify it. So in this example, it might be enough to use `e6` or `e69` rather than the full id.

2. Once inside the container, connect locally with `sqlcmd`. Note that `sqlcmd` is not in the path by default, so you have to specify the full path.

```
/opt/mssql-tools/bin/sqlcmd -S localhost -U SA -P '<YourPassword>'
```

3. When finished with `sqlcmd`, type `exit`.
4. When finished with the interactive command-prompt, type `exit`. Your container continues to run after you exit the interactive bash shell.

Run multiple SQL Server containers

Docker provides a way to run multiple SQL Server containers on the same host machine. This is the approach for scenarios that require multiple instances of SQL Server on the same host. Each container must expose itself on a different port.

The following example creates two SQL Server containers and maps them to ports **1401** and **1402** on the host machine.

```
docker run -e 'ACCEPT_EULA=Y' -e 'MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>' -p 1401:1433 -d microsoft/mssql-server-linux:2017-latest
docker run -e 'ACCEPT_EULA=Y' -e 'MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>' -p 1402:1433 -d microsoft/mssql-server-linux:2017-latest
```

```
docker run -e "ACCEPT_EULA=Y" -e "MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>" -p 1401:1433 -d microsoft/mssql-server-linux:2017-latest
docker run -e "ACCEPT_EULA=Y" -e "MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>" -p 1402:1433 -d microsoft/mssql-server-linux:2017-latest
```

Now there are two instances of SQL Server running in separate containers. Clients can connect to each SQL Server instance by using the IP address of the Docker host and the port number for the container.

```
sqlcmd -S 10.3.2.4,1401 -U SA -P '<YourPassword>'
sqlcmd -S 10.3.2.4,1402 -U SA -P '<YourPassword>'
```

```
sqlcmd -S 10.3.2.4,1401 -U SA -P "<YourPassword>"  
sqlcmd -S 10.3.2.4,1402 -U SA -P "<YourPassword>"
```

Persist your data

Your SQL Server configuration changes and database files are persisted in the container even if you restart the container with `docker stop` and `docker start`. However, if you remove the container with `docker rm`, everything in the container is deleted, including SQL Server and your databases. The following section explains how to use **data volumes** to persist your database files even if the associated containers are deleted.

IMPORTANT

For SQL Server, it is critical that you understand data persistence in Docker. In addition to the discussion in this section, see Docker's documentation on [how to manage data in Docker containers](#).

Mount a host directory as data volume

The first option is to mount a directory on your host as a data volume in your container. To do that, use the `docker run` command with the `-v <host directory>:/var/opt/mssql` flag. This allows the data to be restored between container executions.

```
docker run -e 'ACCEPT_EULA=Y' -e 'MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>' -p 1433:1433 -v <host  
directory>:/var/opt/mssql -d microsoft/mssql-server-linux:2017-latest
```

```
docker run -e "ACCEPT_EULA=Y" -e "MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>" -p 1433:1433 -v <host  
directory>:/var/opt/mssql -d microsoft/mssql-server-linux:2017-latest
```

This technique also enables you to share and view the files on the host outside of Docker.

IMPORTANT

Host volume mapping for Docker on Mac with the SQL Server on Linux image is not supported at this time. Use data volume containers instead. This restriction is specific to the `/var/opt/mssql` directory. Reading from a mounted directory works fine. For example, you can mount a host directory using `-v` on Mac and restore a backup from a .bak file that resides on the host.

Use data volume containers

The second option is to use a data volume container. You can create a data volume container by specifying a volume name instead of a host directory with the `-v` parameter. The following example creates a shared data volume named **sqlvolume**.

```
docker run -e 'ACCEPT_EULA=Y' -e 'MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>' -p 1433:1433 -v  
sqlvolume:/var/opt/mssql -d microsoft/mssql-server-linux:2017-latest
```

```
docker run -e "ACCEPT_EULA=Y" -e "MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>" -p 1433:1433 -v  
sqlvolume:/var/opt/mssql -d microsoft/mssql-server-linux:2017-latest
```

NOTE

This technique for implicitly creating a data volume in the run command does not work with older versions of Docker. In that case, use the explicit steps outlined in the Docker documentation, [Creating and mounting a data volume container](#).

Even if you stop and remove this container, the data volume persists. You can view it with the `docker volume ls` command.

```
docker volume ls
```

If you then create another container with the same volume name, the new container uses the same SQL Server data contained in the volume.

To remove a data volume container, use the `docker volume rm` command.

WARNING

If you delete the data volume container, any SQL Server data in the container is *permanently* deleted.

Backup and restore

In addition to these container techniques, you can also use standard SQL Server backup and restore techniques. You can use backup files to protect your data or to move the data to another SQL Server instance. For more information, see [Backup and restore SQL Server databases on Linux](#).

WARNING

If you do create backups, make sure to create or copy the backup files outside of the container. Otherwise, if the container is removed, the backup files are also deleted.

Execute commands in a container

If you have a running container, you can execute commands within the container from a host terminal.

To get the container ID run:

```
docker ps
```

To start a bash terminal in the container run:

```
docker exec -ti <Container ID> /bin/bash
```

Now you can run commands as though you are running them at the terminal inside the container. When finished, type `exit`. This exits in the interactive command session, but your container continues to run.

Copy files from a container

To copy a file out of the container, use the following command:

```
docker cp <Container ID>:<Container path> <host path>
```

Example:

```
docker cp d6b75213ef80:/var/opt/mssql/log/errorlog /tmp/errorlog
```

```
docker cp d6b75213ef80:/var/opt/mssql/log/errorlog C:\Temp\errorlog
```

Copy files into a container

To copy a file into the container, use the following command:

```
docker cp <Host path> <Container ID>:<Container path>
```

Example:

```
docker cp /tmp/mydb.mdf d6b75213ef80:/var/opt/mssql/data
```

```
docker cp C:\Temp\mydb.mdf d6b75213ef80:/var/opt/mssql/data
```

Run a specific SQL Server container image

There are scenarios where you might not want to use the latest SQL Server container image. To run a specific SQL Server container image, use the following steps:

1. Identify the Docker **tag** for the release you want to use. To view the available tags, see [the mssql-server-linux Docker hub page](#).
2. Pull the SQL Server container image with the tag. For example, to pull the RC1 image, replace `<image_tag>` in the following command with `rc1`.

```
docker pull microsoft/mssql-server-linux:<image_tag>
```

3. To run a new container with that image, specify the tag name in the `docker run` command. In the following command, replace `<image_tag>` with the version you want to run.

```
docker run -e 'ACCEPT_EULA=Y' -e 'MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>' -p 1401:1433 -d  
microsoft/mssql-server-linux:<image_tag>
```

```
docker run -e "ACCEPT_EULA=Y" -e "MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>" -p 1401:1433 -d  
microsoft/mssql-server-linux:<image_tag>
```

These steps can also be used to downgrade an existing container. For example, you might want to rollback or downgrade a running container for troubleshooting or testing. To downgrade a running container, you must be using a persistence technique for the data folder. Follow the same steps outlined in the [upgrade section](#), but specify the tag name of the older version when you run the new container.

IMPORTANT

Upgrade and downgrade are only supported between RC1 and RC2 at this time.

Upgrade SQL Server in containers

To upgrade the container image with Docker, first identify the tag for the release for your upgrade. Pull this version from the registry with the `docker pull` command:

```
docker pull microsoft/mssql-server-linux:<image_tag>
```

This updates the SQL Server image for any new containers you create, but it does not update SQL Server in any running containers. To do this, you must create a new container with the latest SQL Server container image and migrate your data to that new container.

1. Make sure you are using one of the [data persistence techniques](#) for your existing SQL Server container. This enables you to start a new container with the same data.
2. Stop the SQL Server container with the `docker stop` command.
3. Create a new SQL Server container with `docker run` and specify either a mapped host directory or a data volume container. Make sure to use the specific tag for the your SQL Server upgrade. The new container now uses a new version of SQL Server with your existing SQL Server data.

IMPORTANT

Upgrade is only supported between RC1, RC2, and GA at this time.

4. Verify your databases and data in the new container.
5. Optionally, remove the old container with `docker rm`.

Troubleshooting

The following sections provide troubleshooting suggestions for running SQL Server in containers.

Docker command errors

If you get errors for any `docker` commands, make sure that the docker service is running, and try to run with elevated permissions.

For example, on Linux, you might get the following error when running `docker` commands:

```
Cannot connect to the Docker daemon. Is the docker daemon running on this host?
```

If you get this error on Linux, try running the same commands prefaced with `sudo`. If that fails, verify the docker service is running, and start it if necessary.

```
sudo systemctl status docker
sudo systemctl start docker
```

On Windows, verify that you are launching PowerShell or your command-prompt as an Administrator.

SQL Server container startup errors

If the SQL Server container fails to run, try the following tests:

- If you get an error such as '**failed to create endpoint CONTAINER_NAME on network bridge. Error starting proxy: listen tcp 0.0.0.0:1433 bind: address already in use.**', then you are attempting to map the container port 1433 to a port that is already in use. This can happen if you're running SQL Server locally on the host machine. It can also happen if you start two SQL Server containers and try to map them both to the same host port. If this happens, use the `-p` parameter to map the container port 1433 to a different host port. For example:

```
docker run -e 'ACCEPT_EULA=Y' -e 'MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>' -p 1400:1433 -d  
microsoft/mssql-server-linux:2017-latest`.
```

```
docker run -e "ACCEPT_EULA=Y" -e "MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>" -p 1400:1433 -d  
microsoft/mssql-server-linux:2017-latest`.
```

- Check to see if there are any error messages from container.

```
docker logs e69e056c702d
```

- Make sure that you meet the minimum memory and disk requirements specified in the [Requirements](#) section of this topic.
- If you are using any container management software, make sure it supports container processes running as root. The sqlservr process in the container runs as root.
- Review the [SQL Server setup and error logs](#).

Enable dump captures

If the SQL Server process is failing inside the container, you should create a new container with **SYS_PTRACE** enabled. This adds the Linux capability to trace a process, which is necessary for creating a dump file on an exception. The dump file can be used by support to help troubleshoot the problem. The following docker run command enables this capability.

```
docker run -e "ACCEPT_EULA=Y" -e "MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>" -e "MSSQL_PID=Developer" --cap-add  
SYS_PTRACE -p 1401:1433 -d microsoft/mssql-server-linux:2017-latest
```

SQL Server connection failures

If you can't connect to the SQL Server instance running in your container, try the following tests:

- Make sure that your SQL Server container is running by looking at the **STATUS** column of the `docker ps -a` output. If not, use `docker start <Container ID>` to start it.
- If you mapped to a non-default host port (not 1433), make sure you are specifying the port in your connection string. You can see your port mapping in the **PORTS** column of the `docker ps -a` output. For example, the following command connects sqlcmd to a container listening on port 1401:

```
sqlcmd -S 10.3.2.4,1401 -U SA -P '<YourPassword>'
```

```
sqlcmd -S 10.3.2.4,1401 -U SA -P "<YourPassword>"
```

- If you used `docker run` with an existing mapped data volume or data volume container, SQL Server

ignores the value of `MSSQL_SA_PASSWORD`. Instead, the pre-configured SA user password is used from the SQL Server data in the data volume or data volume container. Verify that you are using the SA password associated with the data you're attaching to.

- Review the [SQL Server setup and error logs](#).

SQL Server Availability Groups

If you are using Docker with SQL Server Availability Groups, there are two additional requirements.

- Map the port that is used for replica communication (default 5022). For example, specify `-p 5022:5022` as part of your `docker run` command.
- Explicitly set the container host name with the `-h YOURHOSTNAME` parameter of the `docker run` command. This host name is used when you configure your Availability Group. If you don't specify it with `-h`, it defaults to the container ID.

SQL Server setup and error logs

You can look at the SQL Server setup and error logs in `/var/opt/mssql/log`. If the container is not running, first start the container. Then use an interactive command-prompt to inspect the logs.

```
docker start e69e056c702d
docker exec -it e69e056c702d "bash"
```

From the bash session inside your container, run the following commands:

```
cd /var/opt/mssql/log
cat setup*.log
cat errorlog
```

TIP

If you mounted a host directory to `/var/opt/mssql` when you created your container, you can instead look in the `log` subdirectory on the mapped path on the host.

Next steps

Get started with SQL Server 2017 container images on Docker by going through the [quickstart](#).

Also, see the [mssql-docker GitHub repository](#) for resources, feedback, and known issues.

Customer Feedback for SQL Server on Linux

2/14/2018 • 3 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

By default, Microsoft SQL Server collects information about how its customers are using the application. Specifically, SQL Server collects information about the installation experience, usage, and performance. This information helps Microsoft improve the product to better meet customer needs. For example, Microsoft collects information about what kinds of error codes customers encounter so that we can fix related bugs, improve our documentation about how to use SQL Server, and determine whether features should be added to the product to better serve customers.

This document provides details about what kinds of information are collected and about how to configure Microsoft SQL Server on Linux to send that collected information to Microsoft. SQL Server 2017 includes a privacy statement that explains what information we do and do not collect from users. Please read the privacy statement.

Specifically, Microsoft does not send any of the following types of information through this mechanism:

- Any values from inside user tables
- Any logon credentials or other authentication information
- Personally Identifiable Information (PII)

SQL Server 2017 always collects and sends information about the installation experience from the setup process so that we can quickly find and fix any installation problems that the customer is experiencing. SQL Server 2017 can be configured not to send information (on a per-server instance basis) to Microsoft through **mssql-conf**. mssql-conf is a configuration script that installs with SQL Server 2017 for Red Hat Enterprise Linux, SUSE Linux Enterprise Server, and Ubuntu.

NOTE

You can disable the sending of information to Microsoft only in paid versions of SQL Server.

Disable Customer Feedback

This option lets you change if SQL Server sends feedback to Microsoft or not. By default, this value is set to true. To change the value, run the following commands:

On Red Hat, SUSE, and Ubuntu

1. Run the mssql-conf script as root with the **set** command for **telemetry.customerfeedback**. The following example turns off customer feedback by specifying **false**.

```
sudo /opt/mssql/bin/mssql-conf set telemetry.customerfeedback false
```

2. Restart the SQL Server service:

```
sudo systemctl restart mssql-server
```

On Docker

To disable Customer Feedback on docker you must have Docker [persist your data](#).

1. Add an `mssql.conf` file with the lines `[telemetry]` and `customerfeedback = false` in the host directory:

```
echo '[telemetry]' >> <host directory>/mssql.conf
```

```
echo 'customerfeedback = false' >> <host directory>/mssql.conf
```

2. Run the container image

```
docker run -e 'ACCEPT_EULA=Y' -e 'MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>' -p 1433:1433 -v <host directory>:/var/opt/mssql -d microsoft/mssql-server-linux:2017-latest
```

```
docker run -e "ACCEPT_EULA=Y" -e "MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>" -p 1433:1433 -v <host directory>:/var/opt/mssql -d microsoft/mssql-server-linux:2017-latest
```

Local Audit for SQL Server on Linux Usage Feedback Collection

Microsoft SQL Server 2017 contains Internet-enabled features that can collect and send information about your computer or device ("standard computer information") to Microsoft. The Local Audit component of SQL Server Usage Feedback collection can write data collected by the service to a designated folder, representing the data (logs) that will be sent to Microsoft. The purpose of the Local Audit is to allow customers to see all data Microsoft collects with this feature, for compliance, regulatory or privacy validation reasons.

In SQL Server on Linux, Local Audit is configurable at instance level for SQL Server Database Engine. Other SQL Server components and SQL Server Tools do not have Local Audit capability for usage feedback collection.

Enable Local Audit

This option enables Local Audit and lets you set the directory where the Local Audit logs are created.

1. Create a target directory for new Local Audit logs. The following example creates a new **/tmp/audit** directory:

```
sudo mkdir /tmp/audit
```

2. Change the owner and group of the directory to the **mssql** user:

```
sudo chown mssql /tmp/audit  
sudo chgrp mssql /tmp/audit
```

3. Run the mssql-conf script as root with the **set** command for **telemetry.userrequestedlocalauditdirectory**:

```
sudo /opt/mssql/bin/mssql-conf set telemetry.userrequestedlocalauditdirectory /tmp/audit
```

4. Restart the SQL Server service:

```
sudo systemctl restart mssql-server
```

On Docker

To enable Local Audit on docker you must have Docker [persist your data](#).

1. The target directory for new Local Audit logs will be in the container. Create a target directory for new Local Audit logs in the host directory on your machine. The following example creates a new **/audit** directory:

```
sudo mkdir <host directory>/audit
```

2. Add an `mssql.conf` file with the lines `[telemetry]` and
`userrequestedlocalauditdirectory = <host directory>/audit` in the host directory:

```
echo '[telemetry]' >> <host directory>/mssql.conf
```

```
echo 'userrequestedlocalauditdirectory = <host directory>/audit' >> <host directory>/mssql.conf
```

3. Run the container image

```
docker run -e 'ACCEPT_EULA=Y' -e 'MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>' -p 1433:1433 -v <host directory>:/var/opt/mssql -d microsoft/mssql-server-linux:2017-latest
```

```
docker run -e "ACCEPT_EULA=Y" -e "MSSQL_SA_PASSWORD=<YourStrong!Passw0rd>" -p 1433:1433 -v <host directory>:/var/opt/mssql -d microsoft/mssql-server-linux:2017-latest
```

Next steps

For more information about SQL Server on Linux, see the [Overview of SQL Server on Linux](#).

How to get started developing applications for SQL Server on Linux

2/14/2018 • 2 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

You can create applications that connect to and use SQL Server 2017 on Linux from a variety of programming languages, such as C#, Java, Node.js, PHP, Python, Ruby, and C++. You can also use popular web frameworks and Object Relational Mapping (ORM) frameworks.

TIP

These same development options also enable you to target SQL Server on other platforms. Applications can target SQL Server running on-premises or in the cloud, on Linux, Windows, or Docker on macOS. Or you can target Azure SQL Database and Azure SQL Data Warehouse.

Try the tutorials

The best way to get started and build applications with SQL Server is to try it out for yourself.

- Browse to [Getting Started Tutorials](#).
- Select your language and development platform.
- Try the code samples.

TIP

If you want to develop for SQL Server 2017 on Docker, take a look at the [macOS](#) tutorials.

Create new applications

If you're creating a new application, take a look at a list of the [Connectivity libraries](#) for a summary of the connectors and popular frameworks available for various programming languages.

Use existing applications

If you have an existing database application, you can simply change its connection string to target SQL Server 2017 on Linux. Make sure to read about the [Known Issues](#) in SQL Server 2017 on Linux.

Use existing SQL tools on Windows with SQL Server on Linux

Tools that currently run on Windows such as SSMS, SSDT, and PowerShell, also work with SQL Server 2017 on Linux. Although they do not run natively on Linux, you can still manage remote SQL Server instances on Linux.

See the following topics for more information:

- [SQL Server Management Studio \(SSMS\)](#)
- [SQL Server Data Tools \(SSDT\)](#)

- [SQL PowerShell](#)

NOTE

Make sure that you are using the latest versions of these tools for the best experience.

Use new SQL tools for Linux

You can use the new [mssql extension](#) for [Visual Studio Code](#) on Linux, macOS, and Windows. For a step-by-step walkthrough, see the following tutorial:

- [Use Visual Studio Code](#)

You can also use new command-line tools that are native for Linux. These tools include the following:

- [sqlcmd](#)
- [bcp](#)
- [mssql-conf](#)

Next steps

To get started, install SQL Server on Linux using one of the following quickstarts:

- [Install on Red Hat Enterprise Linux](#)
- [Install on SUSE Linux Enterprise Server](#)
- [Install on Ubuntu](#)
- [Run on Docker](#)

Connectivity libraries and frameworks for Microsoft SQL Server

2/14/2018 • 1 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Check out the [Getting Started Tutorials](#) to quickly get started with programming languages such as C#, Java, Node.js, PHP, and Python and build an app using SQL Server on Linux or Windows or Docker on macOS.

The following table lists connectivity libraries or *drivers* that client applications can use from a variety of languages to connect to and use Microsoft SQL Server running on-premises or in the cloud, on Linux, Windows or Docker and also to Azure SQL Database and Azure SQL Data Warehouse.

LANGUAGE	PLATFORM	ADDITIONAL RESOURCES	DOWNLOAD	GET STARTED
C#	Windows, Linux, macOS	Microsoft ADO.NET for SQL Server	Download	Get Started
Java	Windows, Linux, macOS	Microsoft JDBC Driver for SQL Server	Download	Get Started
PHP	Windows, Linux, macOS	PHP SQL Driver for SQL Server	Operating System: * Windows * Linux * macOS	Get Started
Node.js	Windows, Linux, macOS	Node.js Driver for SQL Server	Get Started	
Python	Windows, Linux, macOS	Python SQL Driver * pyodbc	Get Started	
Ruby	Windows, Linux, macOS	Ruby Driver for SQL Server	Get Started	
C++	Windows, Linux, macOS	Microsoft ODBC Driver for SQL Server	Download	

The following table lists a few examples of Object Relational Mapping (ORM) frameworks and web frameworks that client applications can use with Microsoft SQL Server running on-premises or in the cloud, on Linux, Windows or Docker and also to Azure SQL Database and Azure SQL Data Warehouse.

LANGUAGE	PLATFORM	ORM(S)
C#	Windows, Linux, macOS	Entity Framework Entity Framework Core
Java	Windows, Linux, macOS	Hibernate ORM

LANGUAGE	PLATFORM	ORM(S)
PHP	Windows, Linux	Laravel (Eloquent)
Node.js	Windows, Linux, macOS	Sequelize ORM
Python	Windows, Linux, macOS	Django
Ruby	Windows, Linux, macOS	Ruby on Rails

Related links

- [SQL Server Drivers](#) for connecting from client applications

Use Visual Studio Code to create and run Transact-SQL scripts for SQL Server

2/14/2018 • 6 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Linux only) ✖ Azure SQL Database ✖ Azure SQL Data Warehouse ✖ Parallel Data Warehouse

This article shows how to use the **mssql** extension for Visual Studio Code (VS Code) to develop SQL Server databases.

Visual Studio Code is a graphical code editor for Linux, macOS, and Windows that supports extensions. The [mssql extension for VS Code](#) enables you to connect to SQL Server, query with Transact-SQL (T-SQL), and view the results.

Install VS Code

1. If you have not already installed VS Code, [Download and install VS Code](#) on your machine.
2. Start VS Code.

Install the mssql extension

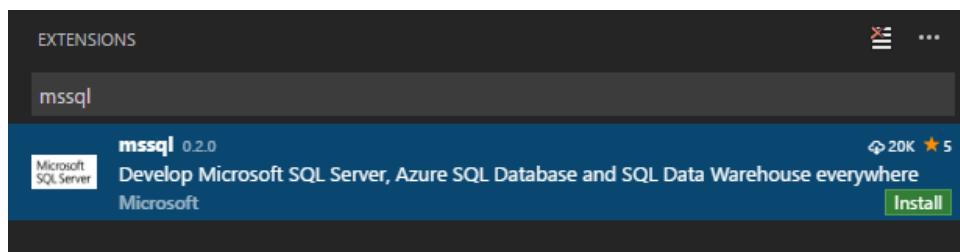
The following steps explain how to install the mssql extension.

1. Press **CTRL+SHIFT+P** (or **F1**) to open the Command Palette in VS Code.
2. Select **Install Extension** and type **mssql**.

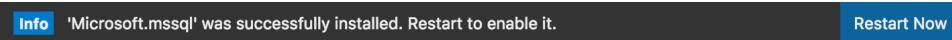
TIP

For macOS, the **CMD** key is equivalent to **CTRL** key on Linux and Windows.

3. Click **install mssql**.



4. The **mssql** extension takes up to one minute to install. Wait for the prompt that tells you it successfully installed.



NOTE

For macOS, you must install OpenSSL. This is a pre-requisite for .Net Core used by the mssql extension. Follow the **install pre-requisite** steps in the [.Net Core instructions](#). Or, you can run the following commands in your macOS Terminal.

```
brew update  
brew install openssl  
ln -s /usr/local/opt/openssl/lib/libcrypto.1.0.0.dylib /usr/local/lib/  
ln -s /usr/local/opt/openssl/lib/libssl.1.0.0.dylib /usr/local/lib/
```

NOTE

For Windows 8.1, Windows Server 2012 or lower versions, you must download and install the [Windows 10 Universal C Runtime](#). Download and open the zip file. Then run the installer (.msu file) targeting your current OS configuration.

Create or open a SQL file

The **mssql** extension enables mssql commands and T-SQL IntelliSense in the editor when the language mode is set to **SQL**.

1. Press **CTRL+N**. Visual Studio Code opens a new 'Plain Text' file by default.
2. Press **CTRL+K,M** and change the language mode to **SQL**.

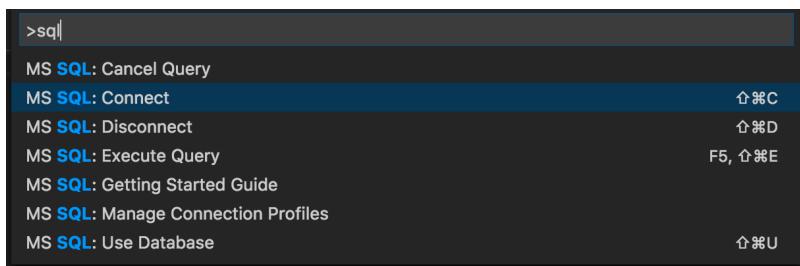


3. Alternatively, open an existing file with .sql file extension. The language mode is automatically **SQL** for files that have the .sql extension.

Connect to SQL Server

The following steps show how to connect to SQL Server with VS Code.

1. In VS Code, press **CTRL+SHIFT+P** (or **F1**) to open the Command Palette.
2. Type **sql** to display the mssql commands.



3. Select the **MS SQL: Connect** command. You can simply type **sqlcon** and press **ENTER**.
4. Select **Create Connection Profile**. This creates a connection profile for your SQL Server instance.
5. Follow the prompts to specify the connection properties for the new connection profile. After specifying each value, press **ENTER** to continue.

The following table describes the Connection Profile properties.

SETTING	DESCRIPTION
Server name	The SQL Server instance name. For this tutorial, use localhost to connect to the local SQL Server instance on your machine. If connecting to a remote SQL Server, enter the name of the target SQL Server machine or its IP address.
[Optional] Database name	The database that you want to use. For purposes of this tutorial, don't specify a database and press ENTER to continue.
User name	Enter the name of a user with access to a database on the server. For this tutorial, use the default SA account created during the SQL Server setup.
Password (SQL Login)	Enter the password for the specified user.
Save Password?	Type Yes to save the password. Otherwise, type No to be prompted for the password each time the Connection Profile is used.
[Optional] Enter a name for this profile	The Connection Profile name. For example, you could name the profile localhost profile .

TIP

You can create and edit connection profiles in User Settings file (settings.json). Open the settings file by selecting **Preference** and then **User Settings** in the VS Code menu. For more information, see [manage connection profiles](#).

- Press the **ESC** key to close the info message that informs you that the profile is created and connected.

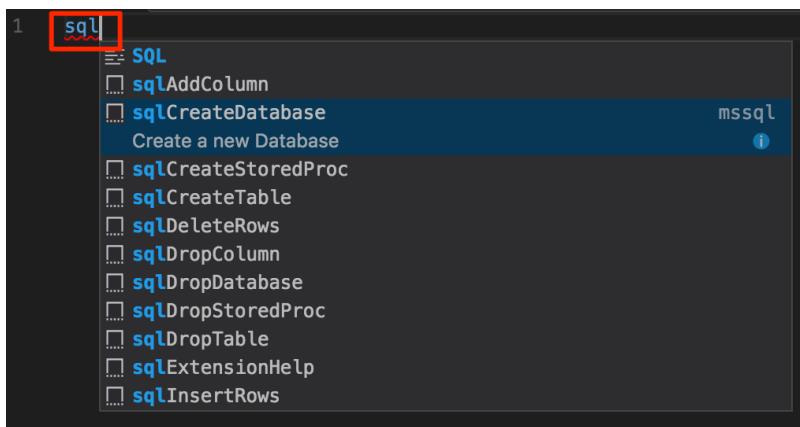
TIP

If you get a connection failure, first attempt to diagnose the problem from the error message in the **Output** panel in VS Code (select **Output** on the **View** menu). Then review the [connection troubleshooting recommendations](#).

- Verify your connection in the status bar.

Create a database

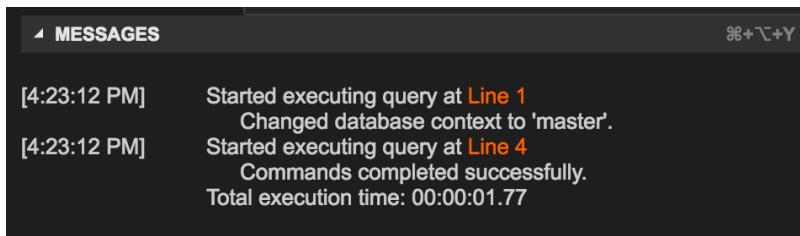
- In the editor, type **sql** to bring up a list of editable code snippets.



2. Select **sqlCreateDatabase**.
3. In the snippet, type **TutorialDB** for the database name.

```
USE master
GO
IF NOT EXISTS (
    SELECT name
    FROM sys.databases
    WHERE name = N'TutorialDB'
)
CREATE DATABASE [TutorialDB]
GO
```

4. Press **CTRL+SHIFT+E** to execute the Transact-SQL commands. View the results in the query window.

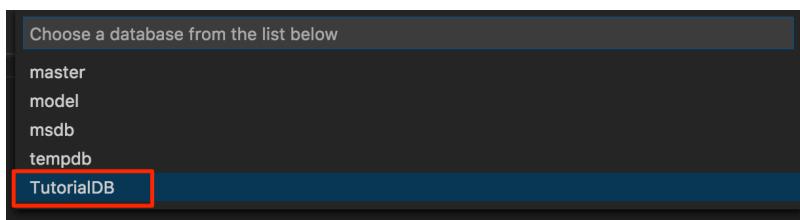


TIP

You can customize shortcut key bindings for the mssql extension commands. See [customize shortcuts](#).

Create a table

1. Remove the contents of the editor window.
2. Press **F1** to display the Command Palette.
3. Type **sql** in the Command Palette to display the SQL commands or type **sqluse** for **MS SQL:Use Database** command.
4. Click **MS SQL:Use Database**, and select the **TutorialDB** database. This changes the context to the new database created in the previous section.



5. In the editor, type **sql** to display the snippets, and then select **sqlCreateTable** and press **enter**.
6. In the snippet, type **Employees** for the table name.
7. Press **Tab**, and then type **dbo** for the schema name.

NOTE

After adding the snippet, you must type the table and schema names without changing focus away from the VS Code editor.

8. Change the column name for **Column1** to **Name** and **Column2** to **Location**.

```
-- Create a new table called 'Employees' in schema 'dbo'  
-- Drop the table if it already exists  
IF OBJECT_ID('dbo.Employees', 'U') IS NOT NULL  
DROP TABLE dbo.Employees  
GO  
-- Create the table in the specified schema  
CREATE TABLE dbo.Employees  
(  
    EmployeesId      INT      NOT NULL  PRIMARY KEY, -- primary key column  
    Name            [NVARCHAR](50)  NOT NULL,  
    Location        [NVARCHAR](50)  NOT NULL  
);  
GO
```

9. Press **CTRL+SHIFT+E** to create the table.

Insert and query

1. Add the following statements to insert four rows into the **Employees** table. Then select all the rows.

```
-- Insert rows into table 'Employees'  
INSERT INTO Employees  
    ([EmployeesId],[Name],[Location])  
VALUES  
    ( 1, N'Jared', N'Australia'),  
    ( 2, N'Nikita', N'India'),  
    ( 3, N'Tom', N'Germany'),  
    ( 4, N'Jake', N'United States')  
GO  
-- Query the total count of employees  
SELECT COUNT(*) as EmployeeCount FROM dbo.Employees;  
-- Query all employee information  
SELECT e.EmployeesId, e.Name, e.Location  
FROM dbo.Employees as e  
GO
```

TIP

While you type, use the assistance of the T-SQL IntelliSense.

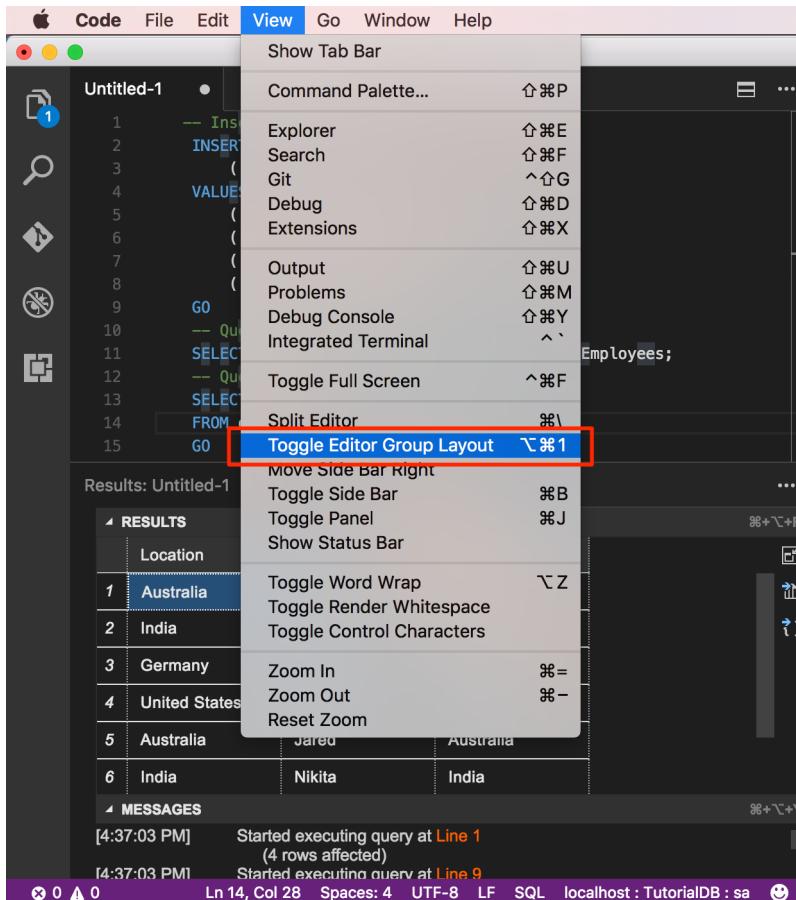
```
10    -- Query the total count of employees
11    SELECT COUNT(*) as EmployeeCount FROM dbo.Employees;
12    -- Query all employee information
13    SELECT e.
14    FROM dbo.e. Id
15    GO
16        column Id(PK, int, not null)
17        Location
18        Name
```

2. Press **CTRL+SHIFT+E** to execute the commands. The two result sets display in the **Results** window.

RESULTS			
	EmployeeCount		
1	8		
	Location	Name	Location
1	Australia	Jared	Australia
2	India	Nikita	India
3	Germany	Tom	Germany
4	United States	Jake	United States
5	Australia	Jared	Australia
6	India	Nikita	India
7	Germany	Tom	Germany
8	United States	Jake	United States

View and save the result

1. On the **View** menu, select **Toggle Editor Group Layout** to switch to vertical or horizontal split layout.



2. Click the **Results** and **Messages** panel header to collapse and expand the panel.

Results: Untitled-1 x

▶ RESULTS
◀ MESSAGES

[4:37:03 PM] Started executing query at Line 1
(4 rows affected)

TIP

You can customize the default behavior of the mssql extension. See [customize extension options](#).

3. Click the maximize grid icon on the second result grid to zoom in.

	Location	Name	Location	
1	Australia	Jared	Australia	
2	India	Nikita	India	
3	Germany	Tom	Germany	

NOTE

The maximize icon displays when your T-SQL script has two or more result grids.

4. Open the grid context menu with the right mouse button on a grid.

	Location	Name	Location	
1	Australia	Jared	Australia	
2	India		India	
3	Germany		Germany	
4	United States	Jake	United States	
5	Australia	Jared	Australia	

5. Select **Select All**.

6. Open the grid context menu and select **Save as JSON** to save the result to a json file.

7. Specify a file name for the JSON file. For this tutorial, type **employees.json**.

8. Verify that the JSON file is saved and opened in VS Code.

```
Untitled-1 Info mssql: Successfully saved results to /var/folders/91/1svnjj... Close ...
```

```
1 [
```

```
2 {
```

```
3   "Location": "Australia",
```

```
4   "Name": "Jared",
```

```
5   "Location": "Australia"
```

```
6 },
```

```
7 {
```

```
8   "Location": "India",
```

```
9   "Name": "Nikita",
```

```
10  "Location": "India"
```

```
11 },
```

```
12 {
```

```
13   "Location": "Germany",
```

```
14   "Name": "Tom",
```

```
15   "Location": "Germany"
```

Next steps

In a real-world scenario, you might create a script that you need to save and run later (either for administration

or as part of a larger development project). In this case, you can save the script with a **.sql** extension.

If you're new to T-SQL, see [Tutorial: Writing Transact-SQL Statements](#) and the [Transact-SQL Reference \(Database Engine\)](#).

For more information on using or contributing to the mssql extension, see the [mssql extension project wiki](#).

For more information on using VS Code, see the [Visual Studio Code documentation](#).

Use SQL Server Management Studio (SSMS) on Windows to manage SQL Server on Linux

2/14/2018 • 2 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✔ SQL Server (Linux only) ✖ Azure SQL Database ✖ Azure SQL Data Warehouse
✖ Parallel Data Warehouse

This article shows how to use [SQL Server Management Studio \(SSMS\)](#) to connect to SQL Server 2017 on Linux. SSMS is a Windows application, so use SSMS when you have a Windows machine that can connect to a remote SQL Server instance on Linux.

After successfully connecting, you run a simple Transact-SQL (T-SQL) query to verify communication with the database.

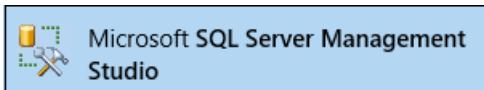
Install the newest version of SQL Server Management Studio

When working with SQL Server, you should always use the most recent version of SQL Server Management Studio (SSMS). The latest version of SSMS is continually updated and optimized and currently works with SQL Server 2017 on Linux. To download and install the latest version, see [Download SQL Server Management Studio](#). To stay up-to-date, the latest version of SSMS prompts you when there is a new version available to download.

Connect to SQL Server on Linux

The following steps show how to connect to SQL Server 2017 on Linux with SSMS.

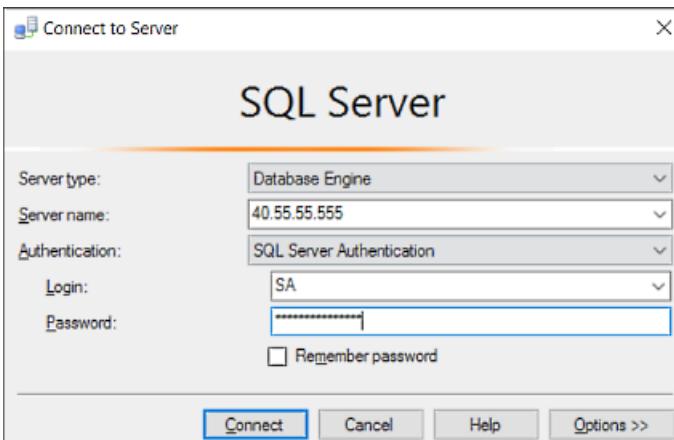
1. Start SSMS by typing **Microsoft SQL Server Management Studio** in the Windows search box, and then click the desktop app.



2. In the **Connect to Server** window, enter the following information (if SSMS is already running, click **Connect > Database Engine** to open the **Connect to Server** window):

SETTING	DESCRIPTION
Server type	The default is database engine; do not change this value.
Server name	Enter the name of the target Linux SQL Server machine or its IP address.
Authentication	For SQL Server 2017 on Linux, use SQL Server Authentication .
Login	Enter the name of a user with access to a database on the server (for example, the default SA account created during setup).

SETTING	DESCRIPTION
Password	Enter the password for the specified user (for the SA account, you created this during setup).

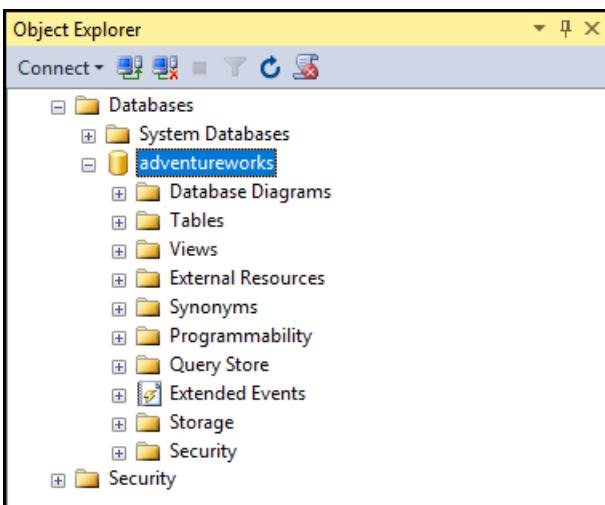


3. Click **Connect**.

TIP

If you get a connection failure, first attempt to diagnose the problem from the error message. Then review the [connection troubleshooting recommendations](#).

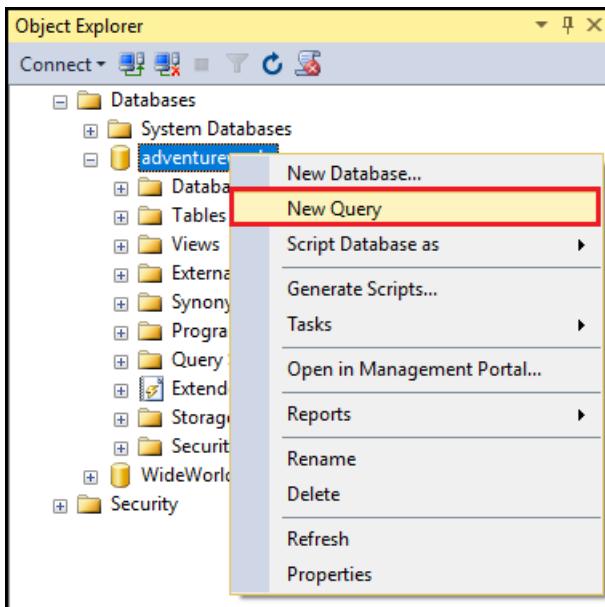
4. After successfully connecting to your SQL Server, **Object Explorer** opens and you can now access your database to perform administrative tasks or query data.



Run sample queries

After you connect to your server, you can connect to a database and run a sample query. If you are new to writing queries, see [Writing Transact-SQL Statements](#).

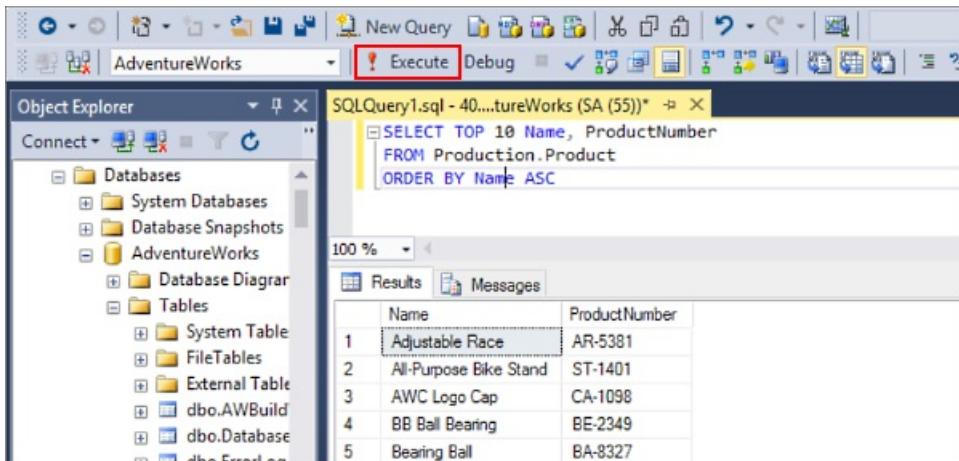
1. Identify a database to use to run a query against. This could be a new database you created in the [Transact-SQL tutorial](#). Or it could be the **AdventureWorks** sample database that you [downloaded and restored](#).
2. In **Object Explorer**, navigate to the target database on the server.
3. Right-click the database and then select **New Query**:



4. In the query window, write a Transact-SQL query to select data from one of the tables. The following example selects data from the **Production.Product** table of the **AdventureWorks** database.

```
SELECT TOP 10 Name, ProductNumber  
FROM Production.Product  
ORDER BY Name ASC
```

5. Click the **Execute** button:



Next steps

In addition to queries, you can use T-SQL statements to create and manage databases.

If you're new to T-SQL, see [Tutorial: Writing Transact-SQL Statements](#) and the [Transact-SQL Reference \(Database Engine\)](#).

For more information on how to use SSMS, see [Use SQL Server Management Studio](#).

Use Visual Studio to create databases for SQL Server on Linux

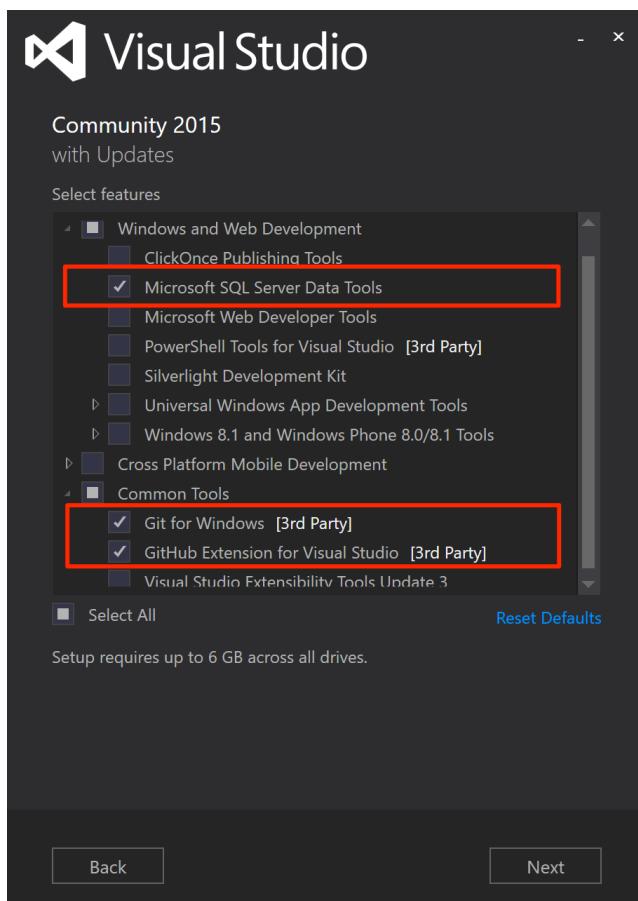
2/14/2018 • 2 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Linux only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

SQL Server Data Tools (SSDT) turns Visual Studio into a powerful development and database lifecycle management (DLM) environment for SQL Server on Linux. You can develop, build, test, and publish your database from a source-controlled project, just like you develop your application code.

Install Visual Studio and SQL Server Data Tools

1. If you have not already installed Visual Studio on your Windows machine, [Download and Install Visual Studio](#). If you do not have a Visual Studio license, Visual Studio Community edition is a free, fully-featured IDE for students, open-source and individual developers.
2. During the Visual Studio installation, select **Custom** for the **Choose the type of installation** option. Click **Next**.
3. Select **Microsoft SQL Server Data Tools**, **Git for Windows**, and **GitHub Extension for Visual Studio** from the feature selection list.



4. Continue and finish the installation of Visual Studio. It can take a few minutes.

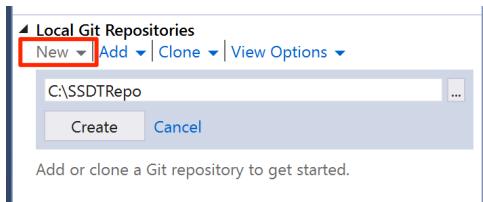
Upgrade SQL Server Data Tools to SSDT 17.0 RC release

SQL Server 2017 on Linux is supported by SSDT version 17.0 RC or later.

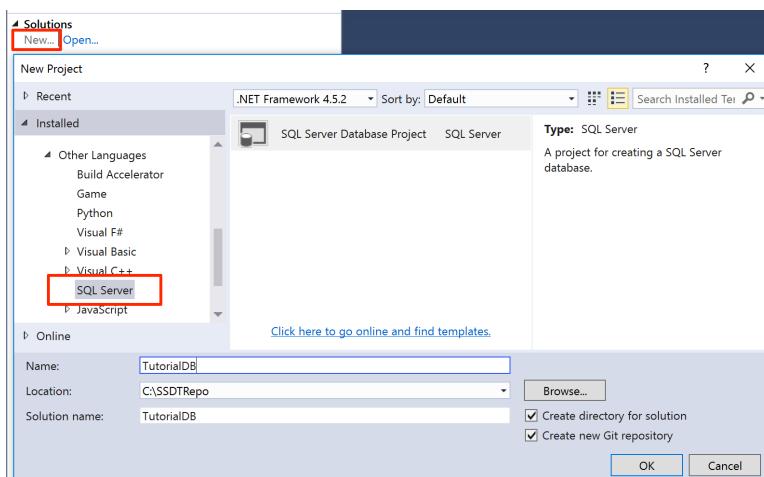
- Download and Install SSDT 17.0 RC2.

Create a new database project in source control

1. Launch Visual Studio.
2. Select **Team Explorer** on the **View** menu.
3. Click **New** in **Local Git Repository** section on the **Connect** page.



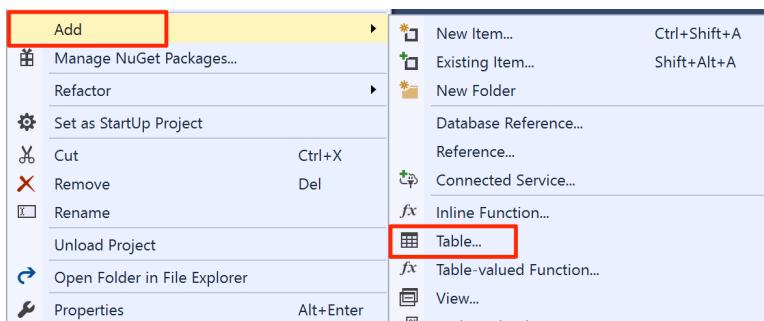
4. Click **Create**. After the local Git repository is created, double click **SSDTRepo**.
5. Click **New** in the **Solutions** section. Select **SQL Server** under **Other Languages** node in the **New Project** dialog.



6. Type in **TutorialDB** for the name and click **OK** to create a new database project.

Create a new table in the database project

1. Select **Solution Explorer** on the **View** menu.
2. Open the database project menu by right-clicking on **TutorialDB** in Solution Explorer.
3. Select **Table** under **Add**.



4. Using table designer, add two columns, Name `nvarchar(50)` and Location `nvarchar(50)`, as shown in the picture. SSDT generates the `CREATE TABLE` script as you add the columns in the designer.

The screenshot shows the SQL Server Object Explorer with a table named 'Table1'. In the 'Design' tab, two columns, 'Name' and 'Location', are selected and highlighted with a red box. The 'T-SQL' tab displays the generated CREATE TABLE script:

```

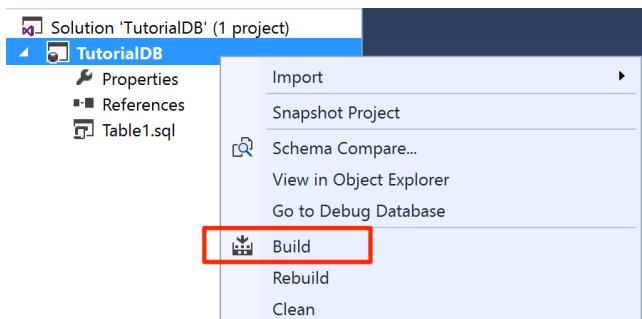
CREATE TABLE [dbo].[Table1]
(
    [Id] INT NOT NULL PRIMARY KEY,
    [Name] NVARCHAR(50) NULL,
    [Location] NVARCHAR(50) NULL
)

```

5. Save the **Table1.sql** file.

Build and validate the database

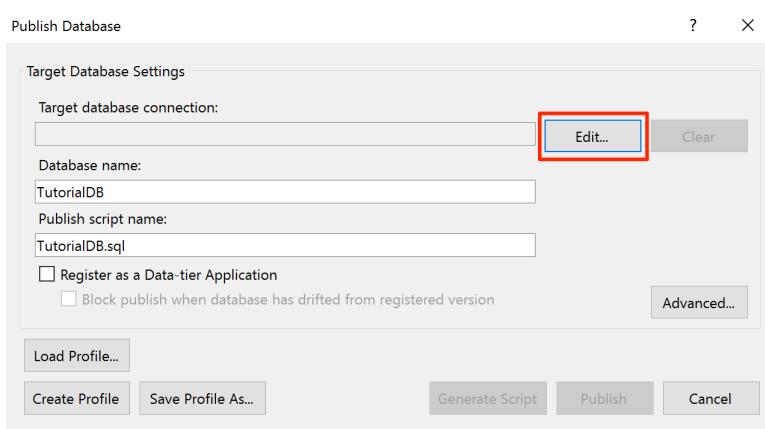
1. Open the database project menu on **TutorialDB** and select **Build**. SSDT compiles .sql source code files in your project and builds a Data-tier Application package (dACPAC) file. This can be used to publish a database to your SQL Server 2017 instance on Linux.



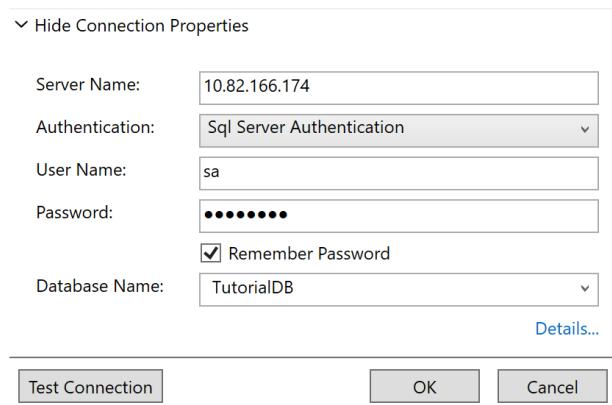
2. Check the build success message in **Output** window in Visual Studio.

Publish the database to SQL Server 2017 instance on Linux

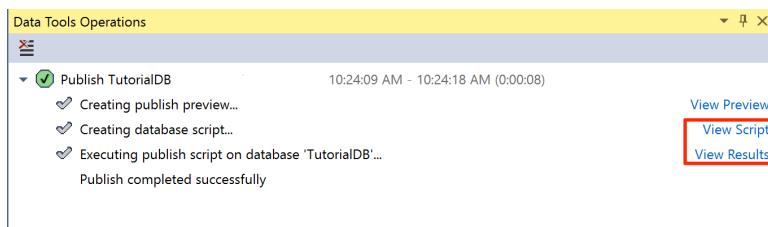
1. Open the database project menu on **TutorialDB** and select **Publish**.
2. Click **Edit** to select your SQL Server instance on Linux.



3. On the connection dialog, type in the IP address or host name of your SQL Server instance on Linux, user name and password.



4. Click the **Publish** button on the publish dialog.
5. Check the publish status in the **Data Tools Operations** window.
6. Click **View Result** or **View Script** to see details of the database publish result on your SQL Server on Linux.



You have successfully created a new database on SQL Server instance on Linux and learned the basics of developing a database with a source-controlled database project.

Next steps

If you're new to T-SQL, see [Tutorial: Writing Transact-SQL Statements](#) and the [Transact-SQL Reference \(Database Engine\)](#).

For more information about developing a database with SQL Data Tools, see [SSDT MSDN documents](#)

Choose the right tool to manage SQL Server on Linux

2/14/2018 • 1 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

There are several ways to manage SQL Server 2017 on Linux. The following section provides a quick overview of different management tools and techniques with pointers to more resources.

mssql-conf

The **mssql-conf** tool configures SQL Server on Linux. For more information, see [Configure SQL Server on Linux with mssql-conf](#).

Transact-SQL

Almost everything you can do in a client tool can also be accomplished with Transact-SQL statements. SQL Server provides [Dynamic Management Views \(DMVs\)](#) that query the status and configuration of SQL Server. There are also [Transact-SQL commands](#) for database management tasks. You can run these commands in any client tool that supports connecting to SQL Server and running Transact-SQL queries, for example [sqlcmd](#) or [Visual Studio Code](#).

SQL Server Operations Studio (preview)

The new Microsoft SQL Operations Studio (preview) is a cross-platform tool for managing SQL Server. For more information, see [Microsoft SQL Operations Studio \(preview\)](#).

SQL Server Management Studio on Windows

SQL Server Management Studio (SSMS) is a Windows application that provides a graphical user interface for managing SQL Server. Although it currently runs only on Windows, you can use it to remotely connect to your Linux SQL Server instances. For more information on using SSMS to manage SQL Server, see [Use SSMS to Manage SQL Server on Linux](#).

mssql-cli (preview)

Microsoft has released a new cross-platform scripting tool for SQL Server, [mssql-cli](#). This tool is currently in preview.

PowerShell

PowerShell provides a rich command-line environment to manage SQL Server on Linux. For more information, see [Use PowerShell to Manage SQL Server on Linux](#).

Next steps

For more information about SQL Server on Linux, see [SQL Server on Linux](#).

Use SQL Server Management Studio on Windows to manage SQL Server on Linux

2/14/2018 • 3 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article introduces [SQL Server Management Studio \(SSMS\)](#) and walks you through a couple of common tasks. SSMS is a Windows application, so use SSMS when you have a Windows machine that can connect to a remote SQL Server instance on Linux.

[SQL Server Management Studio \(SSMS\)](#) is part of a suite of SQL tools that Microsoft offers free of charge for your development and management needs. SSMS is an integrated environment to access, configure, manage, administer, and develop all components of SQL Server running on-premises or in the cloud, on Linux, Windows or Docker on macOS and Azure SQL Database and Azure SQL Data Warehouse. SSMS combines a broad group of graphical tools with a number of rich script editors to provide access to SQL Server to developers and administrators of all skill levels.

SSMS offers a broad set of development and management capabilities for SQL Server, including tools to:

- Configure, monitor and administer single or multiple instances of SQL Server
- Deploy, monitor, and upgrade data-tier components such as databases and data warehouses
- Backup and restore databases
- Build and execute T-SQL queries and scripts and see results
- Generate T-SQL scripts for database objects
- View and edit data in databases
- Visually design T-SQL queries and database objects such as views, tables and stored procedures

See [Use SQL Server Management Studio](#) for more information.

Install the newest version of SQL Server Management Studio (SSMS)

When working with SQL Server, you should always use the most recent version of SQL Server Management Studio (SSMS). The latest version of SSMS is continually updated and optimized and currently works with SQL Server 2017 on Linux. To download and install the latest version, see [Download SQL Server Management Studio](#). To stay up-to-date, the latest version of SSMS prompts you when there is a new version available to download.

Before you begin

- See [Use SSMS on Windows to connect to SQL Server on Linux](#) for how to Connect and Query using SSMS
- Read the [Known Issues](#) for SQL Server 2017 on Linux

Create and manage databases

While connected to the *master* database, you can create databases on the server and modify or drop existing databases. The following steps describe how to accomplish several common database management tasks through Management Studio. To perform these tasks, make sure you are connected to the *master* database with the server-level principal login that you created when you set up SQL Server 2017 on Linux.

Create a new database

1. Start SSMS and connect to your server in SQL Server 2017 on Linux
2. In Object Explorer, right-click on the *Databases* folder, and then click **New Database...**
3. In the *New Database* dialog, enter a name for your new database, and then click *OK*

The new database is successfully created in your server. If you prefer to create a new database using T-SQL, then see [CREATE DATABASE \(SQL Server Transact-SQL\)](#).

Drop a database

1. Start SSMS and connect to your server in SQL Server 2017 on Linux
2. In Object Explorer, expand the *Databases* folder to see a list of all the database on the server.
3. In Object Explorer, right-click on the database you wish to drop, and then click *Delete*
4. In the *Delete Object* dialog, check *Close existing connections* and then click *OK*

The database is successfully dropped from your server. If you prefer to drop a database using T-SQL, then see [DROP DATABASE \(SQL Server Transact-SQL\)](#).

Use Activity Monitor to see information about SQL Server activity

The [Activity Monitor](#) tool is built-in into SQL Server Management Studio (SSMS) and displays information about SQL Server processes and how these processes affect the current instance of SQL Server.

1. Start SSMS and connect to your server in SQL Server 2017 on Linux
2. In Object Explorer, right-click the *server* node, and then click *Activity Monitor*

Activity Monitor shows expandable and collapsible panes with information about the following:

- Overview
- Processes
- Resource Waits
- Data File I/O
- Recent Expensive Queries
- Active Expensive Queries

When a pane is expanded, Activity Monitor queries the instance for information. When a pane is collapsed, all querying activity stops for that pane. You can expand one or more panes at the same time to view different kinds of activity on the instance.

See also

- [Use SQL Server Management Studio](#)
- [Export and Import a database with SSMS](#)
- [Tutorial: SQL Server Management Studio](#)
- [Tutorial: Writing Transact-SQL Statements](#)
- [Server Performance and Activity Monitoring](#)

Use PowerShell on Windows to Manage SQL Server on Linux

2/14/2018 • 3 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article introduces [SQL Server PowerShell](#) and walks you through a couple of examples on how to use it with SQL Server 2017 on Linux. PowerShell support for SQL Server is currently available on Windows, so you can use it when you have a Windows machine that can connect to a remote SQL Server instance on Linux.

Install the newest version of SQL PowerShell on Windows

[SQL PowerShell](#) on Windows is included with [SQL Server Management Studio \(SSMS\)](#). When working with SQL Server, you should always use the most recent version of SSMS and SQL PowerShell. The latest version of SSMS is continually updated and optimized and currently works with SQL Server 2017 on Linux. To download and install the latest version, see [Download SQL Server Management Studio](#). To stay up-to-date, the latest version of SSMS prompts you when there is a new version available to download.

Before you begin

Read the [Known Issues](#) for SQL Server 2017 on Linux.

Launch PowerShell and import the `sqlserver` module

Let's start by launching PowerShell on Windows. Open a *command prompt* on your Windows computer, and type **PowerShell** to launch a new Windows PowerShell session.

```
PowerShell
```

SQL Server provides a Windows PowerShell module named **SqlServer** that you can use to import the SQL Server components (SQL Server provider and cmdlets) into a PowerShell environment or script.

Copy and paste the following command at the PowerShell prompt to import the **SqlServer** module into your current PowerShell session:

```
Import-Module SqlServer
```

Type the following command at the PowerShell prompt to verify that the **SqlServer** module was imported correctly:

```
Get-Module -Name SqlServer
```

PowerShell should display information similar to the following output:

ModuleType	Version	Name	ExportedCommands
Script	0.0	SqlServer	
Manifest	20.0	SqlServer	{Add-SqlAvailabilityDatabase, Add-SqlAvailabilityGroupList...

Connect to SQL Server and get server information

Let's use PowerShell on Windows to connect to your SQL Server 2017 instance on Linux and display a couple of server properties.

Copy and paste the following commands at the PowerShell prompt. When you run these commands, PowerShell will:

- Display the *Windows PowerShell credential request* dialog that prompts you for the credentials (*SQL username* and *SQL password*) to connect to your SQL Server 2017 instance on Linux
- Load the SQL Server Management Objects (SMO) assembly
- Create an instance of the [Server](#) object
- Connect to the **Server** and display a few properties

Remember to replace **<your_server_instance>** with the IP address or the hostname of your SQL Server 2017 instance on Linux.

```
# Prompt for credentials to login into SQL Server
$serverInstance = "<your_server_instance>"
$credential = Get-Credential

# Load the SMO assembly and create a Server object
[System.Reflection.Assembly]::LoadWithPartialName('Microsoft.SqlServer.SMO') | out-null
$server = New-Object ('Microsoft.SqlServer.Management.Smo.Server') $serverInstance

# Set credentials
$server.ConnectionContext.LoginSecure=$false
$server.ConnectionContext.set_Login($credential.UserName)
$server.ConnectionContext.set_SecurePassword($credential.Password)

# Connect to the Server and get a few properties
$server.Information | Select-Object Edition, HostPlatform, HostDistribution | Format-List
# done
```

PowerShell should display information similar to the following output:

```
Edition      : Developer Edition (64-bit)
HostPlatform : Linux
HostDistribution : Ubuntu
```

NOTE

If nothing is displayed for these values, the connection to the target SQL Server instance most likely failed. Make sure that you can use the same connection information to connect from SQL Server Management Studio. Then review the [connection troubleshooting recommendations](#).

Examine SQL Server error logs

Let's use PowerShell on Windows to examine error logs connect on your SQL Server 2017 instance on Linux. We will also use the **Out-GridView** cmdlet to show information from the error logs in a grid view display.

Copy and paste the following commands at the PowerShell prompt. They might take a few minutes to run. These commands do the following:

- Display the *Windows PowerShell credential request* dialog that prompts you for the credentials (*SQL username* and *SQL password*) to connect to your SQL Server 2017 instance on Linux
- Use the **Get-SqlErrorLog** cmdlet to connect to the SQL Server 2017 instance on Linux and retrieve error logs since **Yesterday**
- Pipe the output to the **Out-GridView** cmdlet

Remember to replace **<your_server_instance>** with the IP address or the hostname of your SQL Server 2017 instance on Linux.

```
# Prompt for credentials to login into SQL Server
$serverInstance = "<your_server_instance>"
$credential = Get-Credential

# Retrieve error logs since yesterday
Get-SqlErrorLog -ServerInstance $serverInstance -Credential $credential -Since Yesterday | Out-GridView
# done
```

See also

- [SQL Server PowerShell](#)

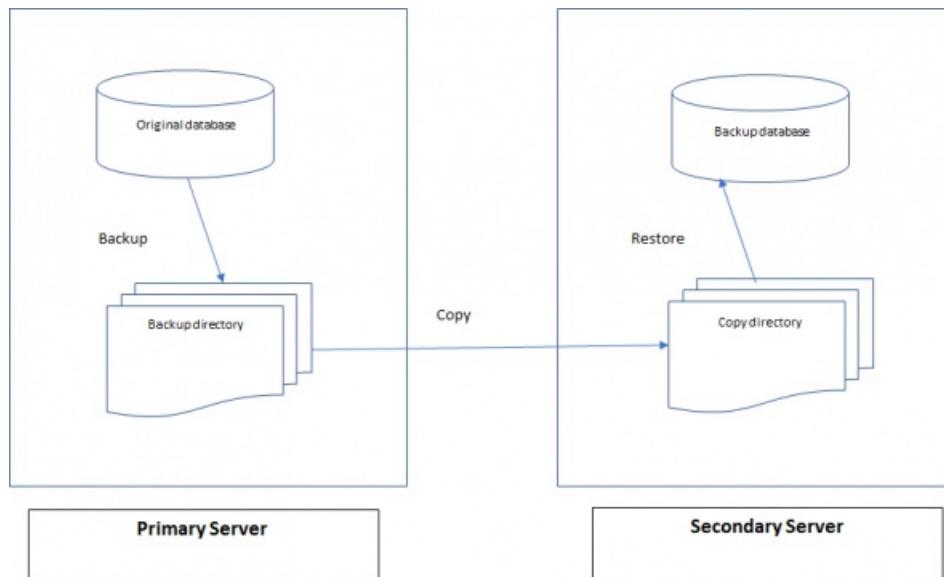
Get started with Log Shipping on Linux

2/14/2018 • 4 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse

Parallel Data Warehouse

SQL Server Log shipping is a HA configuration where a database from a primary server is replicated onto one or more secondary servers. In a nutshell, a backup of the source database is restored onto the secondary server. Then the primary server creates transaction log backups periodically, and the secondary servers restore them, updating the secondary copy of the database.



As described in the this picture, a log shipping session involves the following steps:

- Backing up the transaction log file on the primary SQL Server instance
- Copying the transaction log backup file across the network to one or more secondary SQL Server instances
- Restoring the transaction log backup file on the secondary SQL Server instances

Prerequisites

- [Install SQL Server Agent on Linux](#)

Setup a network share for Log Shipping using CIFS

NOTE

This tutorial uses CIFS + Samba to setup the network share. If you want to use NFS, leave a comment and we will add it to the doc.

Configure Primary Server

- Run the following to install Samba

```
sudo apt-get install samba #For Ubuntu  
sudo yum -y install samba #For RHEL/CentOS
```

- Create a directory to store the logs for Log Shipping and give mssql the required permissions

```
mkdir /var/opt/mssql/tlogs  
chown mssql:mssql /var/opt/mssql/tlogs  
chmod 0700 /var/opt/mssql/tlogs
```

- Edit the /etc/samba/smb.conf file (you need root permissions for that) and add the following section:

```
[tlogs]  
path=/var/opt/mssql/tlogs  
available=yes  
read only=yes  
browsable=yes  
public=yes  
writable=no
```

- Create a mssql user for Samba

```
sudo smbpasswd -a mssql
```

- Restart the Samba services

```
sudo systemctl restart smbd.service nmbd.service
```

Configure Secondary Server

- Run the following to install the CIFS client

```
sudo apt-get install cifs-utils #For Ubuntu  
sudo yum -y install cifs-utils #For RHEL/CentOS
```

- Create a file to store your credentials. Use the password you recently set for your mssql Samba account

```
vim /var/opt/mssql/.tlogcreds  
#Paste the following in .tlogcreds  
username=mssql  
domain=<domain>  
password=<password>
```

- Run the following commands to create an empty directory for mounting and set permission and ownership correctly

```
mkdir /var/opt/mssql/tlogs  
sudo chown root:root /var/opt/mssql/tlogs  
sudo chmod 0550 /var/opt/mssql/tlogs  
sudo chown root:root /var/opt/mssql/.tlogcreds  
sudo chmod 0660 /var/opt/mssql/.tlogcreds
```

- Add the line to etc/fstab to persist the share

```
//<ip_address_of_primary_server>/tlogs /var/opt/mssql/tlogs cifs  
credentials=/var/opt/mssql/.tlogcreds,ro,uid=mssql,gid=mssql 0 0
```

- Mount the shares

```
sudo mount -a
```

Setup Log Shipping via T-SQL

- Run this script from your primary server

```
BACKUP DATABASE SampleDB  
TO DISK = '/var/opt/mssql/tlogs/SampleDB.bak'  
GO
```

```

DECLARE @LS_BackupJobId AS uniqueidentifier
DECLARE @LS_PrimaryId AS uniqueidentifier
DECLARE @SP_Add_RetCode As int
EXEC @SP_Add_RetCode = master.dbo.sp_add_log_shipping_primary_database
    @database = N'SampleDB'
    ,@backup_directory = N'/var/opt/mssql/tlogs'
    ,@backup_share = N'/var/opt/mssql/tlogs'
    ,@backup_job_name = N'LSBackup_SampleDB'
    ,@backup_retention_period = 4320
    ,@backup_compression = 2
    ,@backup_threshold = 60
    ,@threshold_alert_enabled = 1
    ,@history_retention_period = 5760
    ,@backup_job_id = @LS_BackupJobId OUTPUT
    ,@primary_id = @LS_PrimaryId OUTPUT
    ,@overwrite = 1

IF (@@ERROR = 0 AND @SP_Add_RetCode = 0)
BEGIN

DECLARE @LS_BackUpScheduleUID As uniqueidentifier
DECLARE @LS_BackUpScheduleID AS int

EXEC msdb.dbo.sp_add_schedule
    @schedule_name =N'LSBackupSchedule'
    ,@enabled = 1
    ,@freq_type = 4
    ,@freq_interval = 1
    ,@freq_subday_type = 4
    ,@freq_subday_interval = 15
    ,@freq_recurrence_factor = 0
    ,@active_start_date = 20170418
    ,@active_end_date = 99991231
    ,@active_start_time = 0
    ,@active_end_time = 235900
    ,@schedule_uid = @LS_BackUpScheduleUID OUTPUT
    ,@schedule_id = @LS_BackUpScheduleID OUTPUT

EXEC msdb.dbo.sp_attach_schedule
    @job_id = @LS_BackupJobId
    ,@schedule_id = @LS_BackUpScheduleID

EXEC msdb.dbo.sp_update_job
    @job_id = @LS_BackupJobId
    ,@enabled = 1

END

EXEC master.dbo.sp_add_log_shipping_alert_job

EXEC master.dbo.sp_add_log_shipping_primary_secondary
    @primary_database = N'SampleDB'
    ,@secondary_server = N'<ip_address_of_secondary_server>'
    ,@secondary_database = N'SampleDB'
    ,@overwrite = 1

```

- Run this script from your secondary server

```

RESTORE DATABASE SampleDB FROM DISK = '/var/opt/mssql/tlogs/SampleDB.bak'
WITH NORECOVERY;

```

```

DECLARE @LS_Secondary__CopyJobId AS uniqueidentifier
DECLARE @LS_Secondary__RestoreJobId AS uniqueidentifier
DECLARE @LS_Secondary__SecondaryId AS uniqueidentifier

```

```

DECLARE @LS_Add_RetCode As int

EXEC @LS_Add_RetCode = master.dbo.sp_add_log_shipping_secondary_primary
    ,@primary_server = N'<ip_address_of_primary_server>'
    ,@primary_database = N'SampleDB'
    ,@backup_source_directory = N'/var/opt/mssql/tlogs/'
    ,@backup_destination_directory = N'/var/opt/mssql/tlogs/'
    ,@copy_job_name = N'LSCopy_SampleDB'
    ,@restore_job_name = N'LSShare_SampleDB'
    ,@file_retention_period = 4320
    ,@overwrite = 1
    ,@copy_job_id = @LS_Secondary__CopyJobId OUTPUT
    ,@restore_job_id = @LS_Secondary__RestoreJobId OUTPUT
    ,@secondary_id = @LS_Secondary__SecondaryId OUTPUT

IF (@@ERROR = 0 AND @LS_Add_RetCode = 0)
BEGIN

DECLARE @LS_SecondaryCopyJobScheduleUID As uniqueidentifier
DECLARE @LS_SecondaryCopyJobScheduleID AS int

EXEC msdb.dbo.sp_add_schedule
    ,@schedule_name =N'DefaultCopyJobSchedule'
    ,@enabled = 1
    ,@freq_type = 4
    ,@freq_interval = 1
    ,@freq_subday_type = 4
    ,@freq_subday_interval = 15
    ,@freq_recurrence_factor = 0
    ,@active_start_date = 20170418
    ,@active_end_date = 99991231
    ,@active_start_time = 0
    ,@active_end_time = 235900
    ,@schedule_uid = @LS_SecondaryCopyJobScheduleUID OUTPUT
    ,@schedule_id = @LS_SecondaryCopyJobScheduleID OUTPUT

EXEC msdb.dbo.sp_attach_schedule
    ,@job_id = @LS_Secondary__CopyJobId
    ,@schedule_id = @LS_SecondaryCopyJobScheduleID

DECLARE @LS_SecondaryRestoreJobScheduleUID As uniqueidentifier
DECLARE @LS_SecondaryRestoreJobScheduleID AS int

EXEC msdb.dbo.sp_add_schedule
    ,@schedule_name =N'DefaultRestoreJobSchedule'
    ,@enabled = 1
    ,@freq_type = 4
    ,@freq_interval = 1
    ,@freq_subday_type = 4
    ,@freq_subday_interval = 15
    ,@freq_recurrence_factor = 0
    ,@active_start_date = 20170418
    ,@active_end_date = 99991231
    ,@active_start_time = 0
    ,@active_end_time = 235900
    ,@schedule_uid = @LS_SecondaryRestoreJobScheduleUID OUTPUT
    ,@schedule_id = @LS_SecondaryRestoreJobScheduleID OUTPUT

EXEC msdb.dbo.sp_attach_schedule
    ,@job_id = @LS_Secondary__RestoreJobId
    ,@schedule_id = @LS_SecondaryRestoreJobScheduleID

END
DECLARE @LS_Add_RetCode2 As int
IF (@@ERROR = 0 AND @LS_Add_RetCode = 0)
BEGIN

EXEC @LS_Add_RetCode2 = master.dbo.sp_add_log_shipping_secondary_database
    ,@secondary_database = N'SampleDB'

```

```

,@primary_server = N'<ip_address_of_primary_server>'
,@primary_database = N'SampleDB'
,@restore_delay = 0
,@restore_mode = 0
,@disconnect_users = 0
,@restore_threshold = 45
,@threshold_alert_enabled = 1
,@history_retention_period = 5760
,@overwrite = 1

END

IF (@@error = 0 AND @LS_Add_RetCode = 0)
BEGIN

EXEC msdb.dbo.sp_update_job
    @job_id = @LS_Secondary__CopyJobId
    ,@enabled = 1

EXEC msdb.dbo.sp_update_job
    @job_id = @LS_Secondary__RestoreJobId
    ,@enabled = 1

END

```

Verify Log Shipping works

- Verify that Log Shipping works by starting the following job on the primary server

```

USE msdb ;
GO

EXEC dbo.sp_start_job N'LBackup_SampleDB' ;
GO

```

- Verify that Log Shipping works by starting the following job on the secondary server

```

USE msdb ;
GO

EXEC dbo.sp_start_job N'LSCopy_SampleDB' ;
GO
EXEC dbo.sp_start_job N'LRestore_SampleDB' ;
GO
RESTORE DATABASE SampleDB WITH RECOVERY;

```

DB Mail and Email Alerts with SQL Agent on Linux

2/21/2018 • 1 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The following steps show you how to set up DB Mail and use it with SQL Server Agent (**mssql-server-agent**) on Linux.

1. Enable DB Mail

```
USE master
GO
sp_configure 'show advanced options',1
GO
RECONFIGURE WITH OVERRIDE
GO
sp_configure 'Database Mail XPs', 1
GO
RECONFIGURE
GO
```

2. Create a new account

```
EXECUTE msdb.dbo.sysmail_add_account_sp
@account_name = 'SQLAlerts',
@description = 'Account for Automated DBA Notifications',
@email_address = 'sqlagenttest@gmail.com',
@replyto_address = 'sqlagenttest@gmail.com',
@display_name = 'SQL Agent',
@mailserver_name = 'smtp.gmail.com',
@port = 587,
@Enable_ssl = 1,
@username = 'sqlagenttest@gmail.com',
@password = '<password>'
GO
```

3. Create a default profile

```
EXECUTE msdb.dbo.sysmail_add_profile_sp
@profile_name = 'default',
@description = 'Profile for sending Automated DBA Notifications'
GO
```

4. Add the Database Mail account to a Database Mail profile

```
EXECUTE msdb.dbo.sysmail_add_principalprofile_sp
@profile_name = 'default',
@principal_name = 'public',
@is_default = 1 ;
```

5. Add account to profile

```
EXECUTE msdb.dbo.sysmail_add_profileaccount_sp
@profile_name = 'default',
@account_name = 'SQLAlerts',
@sequence_number = 1;
```

6. Send test email

NOTE

You might have to go to your email client and enable the "allow less secure clients to send mail." Not all clients recognize DB Mail as an email daemon.

```
EXECUTE msdb.dbo.sp_send_dbmail
@profile_name = 'default',
@recipients = 'recipient-email@gmail.com',
@Subject = 'Testing DBMail',
@Body = 'This message is a test for DBMail'
GO
```

7. Set DB Mail Profile using mssql-conf or environment variable

You can use the mssql-conf utility or environment variables to register your DB Mail profile. In this case, let's call our profile default.

```
# via mssql-conf
sudo /opt/mssql/bin/mssql-conf set sqlagent.databasemailprofile default
# via environment variable
MSSQL_AGENT_EMAIL_PROFILE=default
```

8. Set up an operator for SQLAgent job notifications

```
EXEC msdb.dbo.sp_add_operator
@email_address=N'recipient-email@gmail.com',
@category_name=N'[Uncategorized]'
GO
```

9. Send email when 'Agent Test Job' succeeds

```
EXEC msdb.dbo.sp_update_job
@job_name='Agent Test Job',
@notify_level_email=1,
@notify_email_operator_name=N'JobAdmins'
GO
```

Next steps

For more information on how to use SQL Server Agent to create, schedule, and run jobs, see [Run a SQL Server](#)

[Agent job on Linux.](#)

Configure multiple-subnet Always On Availability Groups and failover cluster instances

2/14/2018 • 3 min to read • [Edit Online](#)

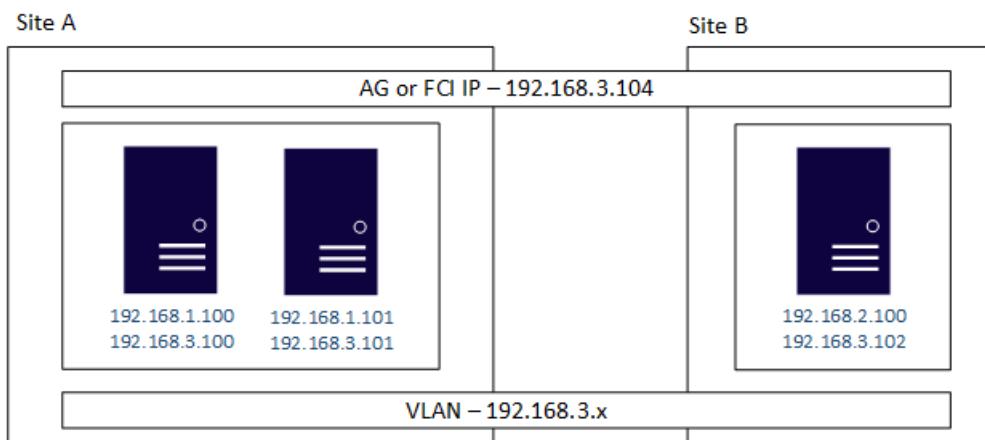
THIS TOPIC APPLIES TO: ✓ SQL Server (Linux only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

When an Always On Availability Group (AG) or failover cluster instance (FCI) spans more than one site, each site usually has its own networking. This often means that each site has its own IP addressing. For example, Site A's addresses start with 192.168.1.x and Site B's addresses start with 192.168.2.x, where x is the part of the IP address that is unique to the server. Without some sort of routing in place at the networking layer, these servers will not be able to communicate with each other. There are two ways to handle this scenario: set up a network that bridges the two different subnets, known as a VLAN, or configure routing between the subnets.

VLAN-based solution

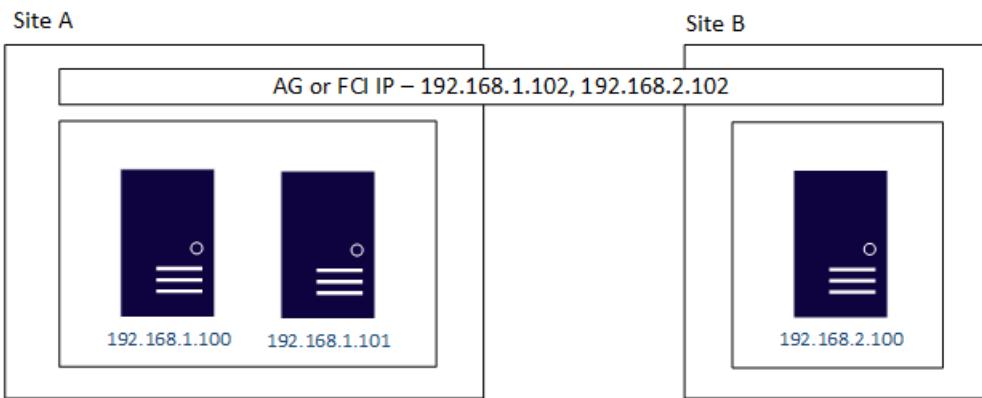
Prerequisite: For a VLAN-based solution, each server participating in an AG or FCI needs two network cards (NICs) for proper availability (a dual port NIC would be a single point of failure on a physical server), so that it can be assigned IP addresses on its native subnet as well as one on the VLAN. This is in addition to any other network needs, such as iSCSI, which also needs its own network.

The IP address creation for the AG or FCI is done on the VLAN. In the following example, the VLAN has a subnet of 192.168.3.x, so the IP address created for the AG or FCI is 192.168.3.104. Nothing additional needs to be configured, since there is a single IP address assigned to the AG or FCI.



Configuration with Pacemaker

In the Windows world, a Windows Server Failover Cluster (WSFC) natively supports multiple subnets and handles multiple IP addresses via an OR dependency on the IP address. On Linux, there is no OR dependency, but there is a way to achieve a proper multi-subnet natively with Pacemaker, as shown by the following. You cannot do this by simply using the normal Pacemaker command line to modify a resource. You need to modify the cluster information base (CIB). The CIB is an XML file with the Pacemaker configuration.



Update the CIB

1. Export the CIB.

Red Hat Enterprise Linux (RHEL) and Ubuntu

```
sudo pcs cluster cib <filename>
```

SUSE Linux Enterprise Server (SLES)

```
sudo cibadmin -Q > <filename>
```

Where *filename* is the name you want to call the CIB.

2. Edit the file that was generated. Look for the `<resources>` section. You will see the various resources that were created for the AG or FCI. Find the one associated with the IP address. Add a `<instance_attributes>` section with the information for the second IP address either above or below the existing one, but before `<operations>`. It is similar to the following syntax:

```
<instance_attributes id=<NameForAttribute> score=<Score>>
  <rule id=<RuleName> score="INFINITY">
    <expression id=<ExpressionName> attribute="\#uname" operation="eq" value=<NodeNameInSubnet2>"
  />
  </rule>
  <nvpair id=<NameForSecondIP> name="ip" value=<IPAddress>"/>
  <nvpair id=<NameForSecondIPNetmask> name="cidr\_netmask" value=<Netmask>"/>
</instance_attributes>
```

where *NameForAttribute* is the unique name for this attribute, *Score* is the number assigned to the attribute, which must be higher than the primary subnet's, *RuleName* is the name of the rule, *ExpressionName* is the name of the expression, *NodeNameInSubnet2* is the name of the node in the other subnet, *NameForSecondIP* is the name associated with the second IP address, *IPAddress* is the IP address for the second subnet, *NameForSecondIPNetmask* is the name associated with the netmask, and *Netmask* is the netmask for the second subnet.

The following shows an example.

```
<instance_attributes id="Node3-2nd-IP" score="2">
  <rule id="Subnet2-IP" score="INFINITY">
    <expression id="Subnet2-Node" attribute="\#uname" operation="eq" value="Node3" />
  </rule>
  <nvpair id="IP-In-Subnet-2" name="ip" value="192.168.2.102"/>
  <nvpair id="Netmask-For-IP2" name="cidr\_netmask" value="24" />
</instance_attributes>
```

3. Import the modified CIB and reconfigure Pacemaker.

RHEL/Ubuntu

```
sudo pcs cluster cib-push <filename>
```

SLES

```
sudo cibadmin -R -x <filename>
```

where *filename* is the name of the CIB file with the modified IP address information.

Check and verify failover

1. After the CIB is successfully applied with the updated configuration, ping the DNS name associated with the IP address resource in Pacemaker. It should reflect the IP address associated with the subnet currently hosting the AG or FCI.
2. Fail the AG or FCI to the other subnet.
3. After the AG or FCI is fully online, ping the DNS name associated with the IP address. It should reflect the IP address in the second subnet.
4. If desired, fail the AG or FCI back to the original subnet.

Migrate databases and structured data to SQL Server on Linux

2/14/2018 • 1 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

You can migrate your databases and data to SQL Server 2017 running on Linux. The method you choose to use depends on the source data and your specific scenario. The following sections provide best practices for various migration scenarios.

Migrate from SQL Server on Windows

If you want to migrate SQL Server databases on Windows to SQL Server 2017 on Linux, the recommended technique is to use SQL Server backup and restore.

1. Create a backup of the database on the Windows machine.
2. Transfer the backup file to the target SQL Server Linux machine.
3. Restore the backup on the Linux machine.

For a tutorial on migrating a database with backup and restore, see the following topic:

- [Restore a SQL Server database from Windows to Linux](#).

It is also possible to export your database to a BACPAC file (a compressed file that contains your database schema and data). If you have a BACPAC file, you can transfer this file to your Linux machine and then import it to SQL Server. For more information, see the following topics:

- [Export and import a database with SSMS or SqlPackage.exe](#)

Migrate from other database servers

You can migrate databases on other database systems to SQL Server 2017 on Linux. This includes Microsoft Access, DB2, MySQL, Oracle, and Sybase databases. In this scenario, use the SQL Server Management Assistant (SSMA) to automate the migration to SQL Server on Linux. For more information, see [Use SSMA to migrate databases to SQL Server on Linux](#).

Migrate structured data

There are also techniques for importing raw data. You might have structured data files that were exported from other databases or data sources. In this case, you can use the bcp tool to bulk insert the data. Or you can run SQL Server Integration Services on Windows to import the data into a SQL Server database on Linux. SQL Server Integration Services enables you to run more complex transformations on the data during the import.

For more information on these techniques, see the following topics:

- [Bulk copy data with bcp](#)
- [Extract, transform, and load data for SQL Server on Linux with SSIS](#)

Export and import a database on Linux with SSMS or SqlPackage.exe on Windows

2/14/2018 • 2 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

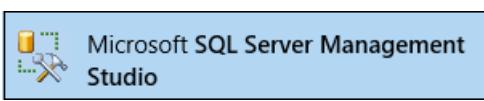
This article shows how to use [SQL Server Management Studio \(SSMS\)](#) and [SqlPackage.exe](#) to export and import a database on SQL Server 2017 on Linux. SSMS and SqlPackage.exe are Windows applications, so use this technique when you have a Windows machine that can connect to a remote SQL Server instance on Linux.

You should always install and use the most recent version of SQL Server Management Studio (SSMS) as described in [Use SSMS on Windows to connect to SQL Server on Linux](#)

NOTE

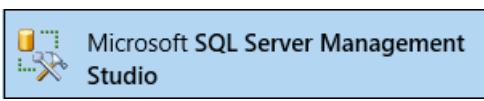
If you are migrating a database from one SQL Server instance to another, the recommendation is to use [Backup and restore](#).

Export a database with SSMS

1. Start SSMS by typing **Microsoft SQL Server Management Studio** in the Windows search box, and then click the desktop app.

2. Connect to your source database in Object Explorer. The source database can be in Microsoft SQL Server running on-premises or in the cloud, on Linux, Windows or Docker and Azure SQL Database or Azure SQL Data Warehouse.
3. Right-click the source database in the Object Explorer, point to **Tasks**, and click **Export Data-Tier Application...**
4. In the export wizard, click **Next**, and then on the **Settings** tab, configure the export to save the BACPAC file to either a local disk location or to an Azure blob.
5. By default, all objects in the database are exported. Click the **Advanced tab** and choose the database objects that you wish to export.
6. Click **Next** and then click **Finish**.

The *.BACPAC file is successfully created at the location you chose and you are ready to import it into a target database.

Import a database with SSMS

1. Start SSMS by typing **Microsoft SQL Server Management Studio** in the Windows search box, and then click the desktop app.


2. Connect to your target server in Object Explorer. The target server can be Microsoft SQL Server running on-premises or in the cloud, on Linux, Windows or Docker and Azure SQL Database or Azure SQL Data Warehouse.
3. Right-click the **Databases** folder in the Object Explorer and click **Import Data-tier Application...**
4. To create the database in your target server, specify a BACPAC file from your local disk or select the Azure storage account and container to which you uploaded your BACPAC file.
5. Provide the New database name for the database. If you are importing a database on Azure SQL Database, set the Edition of Microsoft Azure SQL Database (service tier), Maximum database size, and Service Objective (performance level).
6. Click **Next** and then click **Finish** to import the BACPAC file into a new database in your target server.

The *.BACPAC file is imported to create a new database in the target server you specified.

SqlPackage command-line option

It is also possible to use the SQL Server Data Tools (SSDT) command-line tool, [SqlPackage.exe](#), to export and import BACPAC files.

The following example command exports a BACPAC file:

```
SqlPackage.exe /a:Export /ssn:tcp:<your_server> /sdn:<your_database> /su:<username> /sp:<password> /tf:<path_to_bacpac>
```

Use the following command to import database schema and user data from a .BACPAC file:

```
SqlPackage.exe /a:Import /tsn:tcp:<your_server> /tdn:<your_database> /tu:<username> /tp:<password> /sf:<path_to_bacpac>
```

See also

For more information on how to use SSMS, see [Use SQL Server Management Studio](#). For more information on SqlPackage.exe, see the [SqlPackage reference documentation](#).

Automate database migration to Linux with the SQL Server Migration Assistant

2/14/2018 • 1 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article introduces [SQL Server Migration Assistant \(SSMA\)](#) that helps you easily migrate databases to SQL Server 2017 on Linux from Microsoft Access, DB2, MySQL, Oracle, and Sybase. SSMA is a Windows application, so use SSMA when you have a Windows machine that can connect to a remote SQL Server instance on Linux.

SSMA supports a variety of source databases including Oracle, MySQL, Sybase, DB2, and Microsoft Access to SQL Server 2017 on Linux and helps automate migration tasks such as:

- Assess your source database
- Convert the source database schema to Microsoft SQL Server schema
- Migrate the schema
- Migrate the data
- Test the migration

To get started, download SQL Server Migration Assistant (SSMA) for your source database from the following list:

- [SSMA for Access](#)
- [SSMA for DB2](#)
- [SSMA for MySQL](#)
- [SSMA for Oracle](#)
- [SSMA for Sybase ASE](#)

Next, follow the [SQL Server Migration Assistant \(SSMA\)](#) to migrate your source database to SQL Server 2017 on Linux.

See also

- [Microsoft Data Migration blog](#)
- [SQL Server Migration Assistant \(SSMA\) blog](#)

Bulk copy data with bcp to SQL Server on Linux

2/14/2018 • 4 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article shows how to use the `bcp` command-line utility to bulk copy data between an instance of SQL Server 2017 on Linux and a data file in a user-specified format.

You can use `bcp` to import large numbers of rows into SQL Server tables or to export data from SQL Server tables into data files. Except when used with the `queryout` option, `bcp` requires no knowledge of Transact-SQL. The `bcp` command-line utility works with Microsoft SQL Server running on-premises or in the cloud, on Linux, Windows or Docker and Azure SQL Database and Azure SQL Data Warehouse.

This article shows you how to:

- Import data into a table using the `bcp in` command
- Export data from a table using the `bcp out` command

Install the SQL Server command-line tools

`bcp` is part of the SQL Server command-line tools, which are not installed automatically with SQL Server on Linux. If you have not already installed the SQL Server command-line tools on your Linux machine, you must install them. For more information on how to install the tools, select your Linux distribution from the following list:

- [Red Hat Enterprise Linux \(RHEL\)](#)
- [Ubuntu](#)
- [SUSE Linux Enterprise Server \(SLES\)](#)

Import data with bcp

In this tutorial, you create a sample database and table on the local SQL Server instance (**localhost**) and then use `bcp` to load into the sample table from a text file on disk.

Create a sample database and table

Let's start by creating a sample database with a simple table that is used in the rest of this tutorial.

1. On your Linux box, open a command terminal.
2. Copy and paste the following commands into the terminal window. These commands use the `sqlcmd` command-line utility to create a sample database (**BcpSampleDB**) and a table (**TestEmployees**) on the local SQL Server instance (**localhost**). Remember to replace the `<username>` and `<your_password>` as necessary before running the commands.

Create the database **BcpSampleDB**:

```
sqlcmd -S localhost -U sa -P <your_password> -Q "CREATE DATABASE BcpSampleDB;"
```

Create the table **TestEmployees** in the database **BcpSampleDB**:

```
sqlcmd -S localhost -U sa -P <your_password> -d BcpSampleDB -Q "CREATE TABLE TestEmployees (Id INT IDENTITY(1,1) NOT NULL PRIMARY KEY, Name NVARCHAR(50), Location NVARCHAR(50));"
```

Create the source data file

Copy and paste the following command into your terminal window. We use the built-in `cat` command to create a sample text data file with three records save the file in your home directory as `~/test_data.txt`. The fields in the records are delimited by a comma.

```
cat > ~/test_data.txt << EOF
1,Jared,Australia
2,Nikita,India
3,Tom,Germany
EOF
```

You can verify that the data file was created correctly by running the following command in your terminal window:

```
cat ~/test_data.txt
```

This should display the following in your terminal window:

```
1,Jared,Australia
2,Nikita,India
3,Tom,Germany
```

Import data from the source data file

Copy and paste the following commands into the terminal window. This command uses `bcp` to connect to the local SQL Server instance (`localhost`) and import the data from the data file (`~/test_data.txt`) into the table (`TestEmployees`) in the database (`BcpSampleDB`). Remember to replace the username and `<your_password>` as necessary before running the commands.

```
bcp TestEmployees in ~/test_data.txt -S localhost -U sa -P <your_password> -d BcpSampleDB -c -t ','
```

Here's a brief overview of the command-line parameters we used with `bcp` in this example:

- `-S` : specifies the instance of SQL Server to which to connect
- `-U` : specifies the login ID used to connect to SQL Server
- `-P` : specifies the password for the login ID
- `-d` : specifies the database to connect to
- `-c` : performs operations using a character data type
- `-t` : specifies the field terminator. We are using `comma` as the field terminator for the records in our data file

NOTE

We are not specifying a custom row terminator in this example. Rows in the text data file were correctly terminated with `newline` when we used the `cat` command to create the data file earlier.

You can verify that the data was successfully imported by running the following command in your terminal window. Remember to replace the `username` and `<your_password>` as necessary before running the command.

```
sqlcmd -S localhost -d BcpSampleDB -U sa -P <your_password> -I -Q "SELECT * FROM TestEmployees;"
```

This should display the following results:

Id	Name	Location
1	Jared	Australia
2	Nikita	India
3	Tom	Germany

(3 rows affected)

Export data with bcp

In this tutorial, you use `bcp` to export data from the sample table we created earlier to a new data file.

Copy and paste the following commands into the terminal window. These commands use the `bcp` command-line utility to export data from the table **TestEmployees** in the database **BcpSampleDB** to a new data file called `~/test_export.txt`. Remember to replace the username and `<your_password>` as necessary before running the command.

```
bcp TestEmployees out ~/test_export.txt -S localhost -U sa -P <your_password> -d BcpSampleDB -c -t ','
```

You can verify that the data was exported correctly by running the following command in your terminal window:

```
cat ~/test_export.txt
```

This should display the following in your terminal window:

```
1,Jared,Australia  
2,Nikita,India  
3,Tom,Germany
```

See also

- [bcp utility](#)
- [Data Formats for Compatibility when Using bcp](#)
- [Import Bulk Data by Using BULK INSERT](#)
- [BULK INSERT \(Transact-SQL\)](#)

Extract, transform, and load data on Linux with SSIS

3/1/2018 • 4 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article describes how to run SQL Server Integration Services (SSIS) packages on Linux. SSIS solves complex data integration problems by extracting data from multiple sources and formats, transforming and cleansing the data, and loading the data into multiple destinations.

SSIS packages running on Linux can connect to Microsoft SQL Server running on Windows on-premises or in the cloud, on Linux, or in Docker. They can also connect to Azure SQL Database, Azure SQL Data Warehouse, ODBC data sources, flat files, and other data sources including ADO.NET sources, XML files, and OData services.

For more info about the capabilities of SSIS, see [SQL Server Integration Services](#).

Prerequisites

To run SSIS packages on a Linux computer, first you have to install SQL Server Integration Services. SSIS is not included in the installation of SQL Server for Linux computers. For installation instructions, see [Install SQL Server Integration Services](#).

You also have to have a Windows computer to create and maintain packages. The SSIS design and management tools are Windows applications that are not currently available for Linux computers.

Run an SSIS package

To run an SSIS package on a Linux computer, do the following things:

1. Copy the SSIS package to the Linux computer.
2. Run the following command: `$ dtexec /F \<package name\> /DE <protection password>`

Run an encrypted (password-protected) package

There are three ways to run an SSIS package that's encrypted with a password:

1. Set the value of the environment variable `SSIS_PACKAGE_DECRYPT`, as shown in the following example:

```
SSIS_PACKAGE_DECRYPT=test /opt/ssis/bin/dtexec /f package.dtsx
```

2. Specify the `/de[crypt]` option to enter the password interactively, as shown in the following example:

```
/opt/ssis/bin/dtexec /f package.dtsx /de  
Enter decryption password:
```

3. Specify the `/de` option to provide the password on the command line, as shown in the following example.

This method is not recommended because it stores the decryption password with the command in the command history.

```
opt/ssis/bin/dtexec /f package.dtsx /de test

Warning: Using /De[crypt] <password> may store decryption password in command history.

You can use /De[crypt] instead to enter interactive mode,
or use environment variable SSIS_PACKAGE_DECRYPT to set decryption password.
```

Design packages

Connect to ODBC data sources. With SSIS on Linux CTP 2.1 Refresh and later, SSIS packages can use ODBC connections on Linux. This functionality has been tested with the SQL Server and the MySQL ODBC drivers, but is also expected to work with any Unicode ODBC driver that observes the ODBC specification. At design time, you can provide either a DSN or a connection string to connect to the ODBC data; you can also use Windows authentication. For more info, see the [blog post announcing ODBC support on Linux](#).

Paths. Provide Windows-style paths in your SSIS packages. SSIS on Linux does not support Linux-style paths, but maps Windows-style paths to Linux-style paths at run time. Then, for example, SSIS on Linux maps the Windows-style path `c:\test` to the Linux-style path `/test`.

Deploy packages

You can only store packages in the file system on Linux in this release. The SSIS Catalog database and the legacy SSIS service are not available on Linux for package deployment and storage.

Schedule packages

You can use Linux system scheduling tools such as `cron` to schedule packages. You can't use SQL Agent on Linux to schedule package execution in this release. For more info, see [Schedule SSIS packages on Linux with cron](#).

Limitations and known issues

For detailed info about the limitations and known issues of SSIS on Linux, see [Limitations and known issues for SSIS on Linux](#).

More info about SSIS on Linux

For more info about SSIS on Linux, see the following blog posts:

- [SSIS on Linux is available in SQL Server 2017 CTP2.1](#)
- [ODBC is supported in SSIS on Linux \(SQL Server 2017 CTP 2.1 refresh\)](#)

More info about SSIS

Microsoft SQL Server Integration Services (SSIS) is a platform for building high-performance data integration solutions, including extraction, transformation, and loading (ETL) packages for data warehousing. For more info about SSIS, see [SQL Server Integration Services](#).

SSIS includes the following features:

- Graphical tools and wizards for building and debugging packages on Windows
- A variety of tasks for performing workflow functions such as FTP operations, executing SQL statements, and sending e-mail messages
- A variety of data sources and destinations for extracting and loading data
- A variety of transformations for cleaning, aggregating, merging, and copying data

- Application programming interfaces (APIs) for extending SSIS with your own custom scripts and components

To get started with SSIS, download the latest version of [SQL Server Data Tools \(SSDT\)](#).

To learn more about SSIS, see the following articles:

- [Learn more about SQL Server Integration Services](#)
- [SQL Server Integration Services \(SSIS\) Development and Management Tools](#)
- [SQL Server Integration Services Tutorials](#)

Related content about SSIS on Linux

- [Install SQL Server Integration Services \(SSIS\) on Linux](#)
- [Configure SQL Server Integration Services on Linux with ssis-conf](#)
- [Limitations and known issues for SSIS on Linux](#)
- [Schedule SQL Server Integration Services package execution on Linux with cron](#)

Limitations and known issues for SSIS on Linux

3/1/2018 • 3 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article describes current limitations and known issues for SQL Server Integration Services (SSIS) on Linux.

General limitations and known issues

The following features are not supported in this release of SSIS on Linux:

- SSIS Catalog database
- Scheduled package execution by SQL Agent
- Windows Authentication
- Third-party components
- Change Data Capture (CDC)
- SSIS Scale Out
- Azure Feature Pack for SSIS
- Hadoop and HDFS support
- Microsoft Connector for SAP BW

For other limitations and known issues with SSIS on Linux, see the [Release Notes](#).

Supported and unsupported components

The following built-in Integration Services components are supported on Linux. Some of them have limitations on the Linux platform, as described in the following tables.

Built-in components that are not listed here are not supported on Linux.

Supported control flow tasks

- Bulk Insert Task
- Data Flow Task
- Data Profiling Task
- Execute SQL Task
- Execute T-SQL Statement Task
- Expression Task
- FTP Task
- Web Service Task
- XML Task

Control flow tasks supported with limitations

TASK	LIMITATIONS
Execute Process task	Only supports in-process mode.
File System task	The <i>Move directory</i> and <i>Set file attributes</i> actions are not supported.

TASK	LIMITATIONS
Script task	Only supports standard .NET Framework APIs.
Send Mail task	Only supports anonymous user mode.
Transfer Database task	UNC paths are not supported.

Supported control flow containers

- Sequence Container
- For Loop Container
- Foreach Loop Container

Supported data flow sources and destinations

- Raw File source and destination
- XML Source

Data flow sources and destinations supported with limitations

COMPONENT	LIMITATIONS
ADO.NET source and destination	Only support the SQLClient data provider.
Flat File source and destination	Only support Windows-style file paths, to which the default path mapping rule is applied. For example <div style="border: 1px solid black; padding: 2px; display: inline-block;">D:\home\ssis\travel.csv</div> becomes <div style="border: 1px solid black; padding: 2px; display: inline-block;">/home/ssis/travel.csv</div> .
OData source	Only supports Basic authentication.
ODBC source and destination	Supports 64-bit Unicode ODBC drivers on Linux. Depends on the UnixODBC driver manager on Linux.
OLE DB source and destination	Only support SQL Server Native Client 11.0 and Microsoft OLE DB Provider for SQL Server.

Supported data flow transformations

- Aggregate
- Audit
- Balanced Data Distributor
- Character Map
- Conditional Split
- Copy Column
- Data Conversion
- Derived Column
- Export Column
- Fuzzy Grouping
- Fuzzy Lookup
- Import Column

- Lookup
- Merge
- Merge Join
- Multicast
- Pivot
- Row Count
- Slowly Changing Dimension
- Sort
- Term Lookup
- Union All
- Unpivot

Data flow transformations supported with limitations

COMPONENT	LIMITATIONS
OLE DB Command transformation	Same limitations as the OLE DB source and destination.
Script component	Only supports standard .NET Framework APIs.

Supported and unsupported log providers

All the built-in SSIS log providers are supported on Linux except the Windows Event Log provider.

The SQL Server log provider supports only SQL Authentication; it does not support Windows Authentication.

The SSIS log providers for Text files, for XML files, and for SQL Server Profiler write their output to a file that you specify. The following considerations apply to the file path:

- If you don't provide a path, the log provider writes to the current directory of the host. If the current user doesn't have permission to write to the current directory of the host, the log provider raises an error.
- You can't use an environment variable in a file path. If you specify an environment variable, the literal text that you specify appears in the file path. For example, if you specify `%TMP%/log.txt`, the log provider appends the literal text `/%TMP%/log.txt` to the current host directory.

Related content about SSIS on Linux

- [Extract, transform, and load data on Linux with SSIS](#)
- [Install SQL Server Integration Services \(SSIS\) on Linux](#)
- [Configure SQL Server Integration Services on Linux with ssis-conf](#)
- [Schedule SQL Server Integration Services package execution on Linux with cron](#)

Configure SQL Server Integration Services on Linux with ssis-conf

3/1/2018 • 3 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✔ SQL Server (Linux only) ✖ Azure SQL Database ✖ Azure SQL Data Warehouse ✖ Parallel Data Warehouse

You run the `ssis-conf` configuration script when you install SQL Server Integration Services (SSIS) for Red Hat Enterprise Linux and Ubuntu. For more info about installing SSIS, see [Install SQL Server Integration Services \(SSIS\) on Linux](#).

You can also use the `ssis-conf` utility to configure the following properties:

COMMAND	DESCRIPTION
set-edition	Set the edition of SQL Server
telemetry	Enable or disable SQL Server Integration Services telemetry service
setup	Initialize and set up Microsoft SQL Server Integration Services

Run ssis-conf

The examples in this article run `ssis-conf` by specifying the full path: `/opt/ssis/bin/ssis-conf`. If you navigate to that location before you run `ssis-conf`, you can run the utility in the context of the current directory:

```
./ssis-conf .
```

Be sure to run the commands that are described in this article with root privileges. For example, run `sudo /opt/ssis/bin/ssis-conf setup` and not `/opt/ssis/bin/ssis-conf setup`.

To run these commands with prompts in the language that you prefer, you can specify a locale. For example, to receive prompts in Chinese, run the following command: `sudo LC_ALL=zh_CN.UTF-8 /opt/ssis/bin/ssis-conf setup`.

Use set-edition to set the edition of SQL Server Integration Services

The edition of SSIS is aligned with the edition of SQL Server.

Enter the following command: `$ sudo /opt/ssis/bin/ssis-conf set-edition`.

After you enter the command, you'll receive the following prompt:

Choose an edition of SQL Server:

- 1) Evaluation (free, no production use rights, 180-day limit)
- 2) Developer (free, no production use rights)
- 3) Express (free)
- 4) Web (PAID)
- 5) Standard (PAID)
- 6) Enterprise (PAID)
- 7) Enterprise Core (PAID)
- 8) I bought a license through a retail sales channel and have a product key to enter.

Details about editions can be found at <https://go.microsoft.com/fwlink/?LinkId=852748&clcid=0x409>.

Use of PAID editions of this software requires separate licensing through a Microsoft Volume Licensing program.

By choosing a PAID edition, you are verifying that you have the appropriate number of licenses in place to install and run this software.

Enter your edition (1-8):

If you enter a value from 1 to 7, the system configures a free or PAID edition. If you enter 8, the utility prompts you to enter the product key that you bought:

Enter the 25-character product key:

Use telemetry to configure customer feedback

The `telemetry` command determines whether SSIS sends feedback to Microsoft.

For free editions (that is, Express, Developer, and Evaluation editions), the telemetry service is always enabled. If you have a free edition, you can't use the `telemetry` command to disable telemetry.

Enter the following command: `$ sudo /opt/ssis/bin/ssis-conf telemetry`.

For PAID editions, after you enter the command, you'll receive the following prompt:

Send feature usage data to Microsoft. Feature usage data includes information about your hardware configuration and how you use SQL Server Integration Services.

[Yes/No]:

If you select **Yes**, the telemetry service is enabled and starts running. The service starts automatically after each boot. If you select **No**, the telemetry service stops and is disabled.

Use setup to initialize and set up Microsoft SQL Server Integration Services

Use the `setup` command every time you install SSIS.

Enter the following command: `sudo /opt/ssis/bin/ssis-conf setup`.

The utility prompts you to acknowledge or provide values for the following items:

- Product license
- EULA agreement
- Telemetry service
- The language used by Integration Services

To run the `setup` command with prompts in the language that you prefer, you can specify a locale. For example, to receive prompts in Chinese, run the following command:

```
sudo LC_ALL=zh_CN.UTF-8 /opt/ssis/bin/ssis-conf setup .
```

ssis.conf format

The following `/var/opt/ssis/ssis.conf` file provides an example for each setting.

For SQL Server, you can change system settings by changing the values in the `mssql.conf` file. For SSIS, you *cannot* change system settings by changing the values in the `ssis.conf` file. The `ssis.conf` file shows only the results of the setup. If you want to change the settings for SSIS, you can delete the `ssis.conf` file and run the `setup` command again.

Here is a sample `ssis.conf` file. Each field corresponds to the result of one setup step.

```
[LICENSE]
registered = Y

pid = enterprisecore

[EULA]
accepteula = Y

[TELEMETRY]
enabled = Y

[language]
lcid = 2052
```

Related content about SSIS on Linux

- [Extract, transform, and load data on Linux with SSIS](#)
- [Install SQL Server Integration Services \(SSIS\) on Linux](#)
- [Limitations and known issues for SSIS on Linux](#)
- [Schedule SQL Server Integration Services package execution on Linux with cron](#)

Schedule SQL Server Integration Services package execution on Linux with cron

3/1/2018 • 2 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

When you run SQL Server Integration Services (SSIS) and SQL Server on Windows, you can automate the execution of SSIS packages by using SQL Server Agent. When you run SQL Server and SSIS on Linux, however, the SQL Server Agent utility isn't available to schedule jobs on Linux. Instead, you use the cron service, which is widely used on Linux platforms to automate package execution.

This article provides examples that show how to automate the execution of SSIS packages. The examples are written to run on Red Hat Enterprise. The code is similar for other Linux distributions, such as Ubuntu.

Prerequisites

Before you use the cron service to run jobs, check to see whether it is running on your computer.

To check the status of the cron service, use the following command: `systemctl status crond.service`.

If the service is not active (that is, it is not running), consult your administrator to set up and configure the cron service properly.

Create jobs

A cron job is a task that you can configure to run regularly at a specified interval. The job can be as simple as a command that you would normally type directly in the console or run as a shell script.

For easy management and maintenance purposes, we recommend that you put your package-execution commands in a script that contains a descriptive name.

Here is an example of a simple shell script for running a package. It contains only a single command, but you can add more commands as required.

```
# A simple shell script that contains a simple package execution command
# Script name: SSISpackageName.daily

/opt/ssis/bin/dtexec /F yourSSISpackageName.dtsx >> $HOME/tmp/out 2>&1
```

Schedule jobs with the cron service

After you have defined your jobs, you can schedule them to run automatically by using the cron service.

To add your job for cron to run, add the job in the crontab file. To open the crontab file in an editor where you can add or update the job, use the following command: `crontab -e`.

To schedule the previously described job to run daily at 2:10 AM, add the following line to the crontab file:

```
# run <SSIS package name> at 2:10 AM every day
10 2 \* \* \* $HOME/SSIS/jobs/SSISpackageName.daily
```

Save the crontab file, and then quit the editor.

To understand the format of the sample command, review the information in the following section.

Format of a crontab file

The following image shows the format description of the job line that's added to the crontab file.

```
# Example of job definition:  
# ----- minute (0 - 59)  
# | ----- hour (0 - 23)  
# | | ----- day of month (1 - 31)  
# | | | ----- month (1 - 12) OR jan,feb,mar,apr ...  
# | | | | ----- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat  
# | | | |  
# * * * * * command to be executed
```

To get a more detailed description of the crontab file format, use the following command: `man 5 crontab`.

Here's a partial example of the output that helps to explain the example in this article:

```
EXAMPLE CRON FILE  
# use /bin/sh to run commands, no matter what /etc/passwd says  
SHELL=/bin/sh  
# mail any output to 'paul', no matter whose crontab this is  
MAILTO=paul  
#  
CRON_TZ=Japan  
# run five minutes after midnight, every day  
5 0 * * * $HOME/bin/daily.job >> $HOME/tmp/out 2>&1  
# run at 2:15pm on the first of every month -- output mailed to paul  
15 14 1 * * $HOME/bin/monthly  
# run at 10 pm on weekdays, annoy Joe  
0 22 * * 1-5 mail -s "It's 10pm" joe@Joe,%%Where are your kids?<%  
23 0-23/2 * * * echo "run 23 minutes after midn, 2am, 4am ... , everyday"  
5 4 * * sun echo "run at 5 after 4 every sunday"
```

Related content about SSIS on Linux

- [Extract, transform, and load data on Linux with SSIS](#)
- [Install SQL Server Integration Services \(SSIS\) on Linux](#)
- [Configure SQL Server Integration Services on Linux with ssis-conf](#)
- [Limitations and known issues for SSIS on Linux](#)

Business continuity and database recovery - SQL Server on Linux

2/14/2018 • 28 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article provides an overview of business continuity solutions for high availability and disaster recovery in SQL Server.

One common task everyone deploying SQL Server has to account for is making sure that all mission critical SQL Server instances and the databases within them are available when the business and end users need them, whether that is 9 to 5 or around the clock. The goal is to keep the business up and running with minimal or no interruption. This concept is also known as business continuity.

SQL Server 2017 introduces many new features or enhancements to existing ones, some of which are for availability. The biggest addition to SQL Server 2017 is the support for SQL Server on Linux distributions. For a full list of the new features in SQL Server 2017, see the topic [What's new in SQL Server](#).

This article is focused on covering the availability scenarios in SQL Server 2017 as well as the new and enhanced availability features in SQL Server 2017. The scenarios include hybrid ones that will be able to span SQL Server deployments on both Windows Server and Linux, as well as ones that can increase the number of readable copies of a database. While this article does not cover availability options external to SQL Server, such as those provided by virtualization, everything discussed here applies to SQL Server installations inside a guest virtual machine whether in the public cloud or hosted by an on-premises hypervisor server.

SQL Server 2017 scenarios using the availability features

Availability groups, FCIs, and log shipping can be used in a variety of ways, and not necessarily just for availability purposes. There are four main ways the availability features can be used:

- High availability
- Disaster recovery
- Migrations and upgrades
- Scaling out readable copies of one or more databases

Each section will discuss the relevant features that can be used for that particular scenario. The one feature not covered is [SQL Server replication](#). While not officially designated as an availability feature under the Always On umbrella, it is often used for making data redundant in certain scenarios. Replication will be added to SQL Server on Linux in a future release.

IMPORTANT

The SQL Server availability features do not replace the requirement to have a robust, well tested backup and restore strategy, the most fundamental building block of any availability solution.

High availability

Ensuring that SQL Server instances or database are available in the case of a problem that is local to a data center or single region in the cloud region is important. This section will cover how the SQL Server availability features can assist in that task. All of the features described are available both on Windows Server as well as on Linux.

Always on availability groups

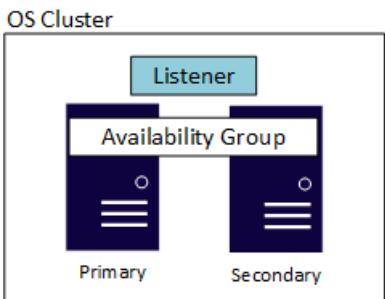
Introduced in SQL Server 2012, Always On Availability Groups (availability groups) provide database-level protection by sending each transaction of a database to another instance, known as a replica, that contains a copy of that database in a special state. An availability group can be deployed on Standard or Enterprise Editions. The instances participating in an availability group can be either standalone or Always On Failover Cluster Instances (FCIs, described in the next section). Since the transactions are sent to a replica as they happen, availability groups are recommended where there are requirements for lower recovery point and recovery time objectives. Data movement between replicas can be synchronous or asynchronous, with Enterprise Edition allowing up to three replicas (including the primary) as synchronous. An availability group has one fully read/write copy of the database which is on the primary replica, while all secondary replicas cannot receive transactions directly from end users or applications.

NOTE

Always On is an umbrella term for the availability features in SQL Server and covers both availability groups and FCIs. Always On is not the name of the availability group feature.

Because availability groups only provide database-level, and not instance-level, protection, anything not captured in the transaction log or configured in the database will need to manually synchronized for each secondary replica. Some examples of objects that must be synchronized manually are logins at the instance level, linked servers, and SQL Server Agent jobs.

An availability group also has another component called the listener, which allows applications and end users to connect without needing to know which SQL Server instance is hosting the primary replica. Each availability group would have its own listener. While the implementations of the listener are slightly different on Windows Server versus Linux, the functionality it provides and how it is used is the same. The picture below shows a Windows Server-based availability group which is using a Windows Server Failover Cluster (WSFC). An underlying cluster at the OS layer is required for availability whether it is on Linux or Windows Server. The example shows a simple two server, or node, configuration where a WSFC is the underlying cluster.



Standard and Enterprise Edition have different maximums when it comes to replicas. An availability group in Standard Edition, known as a Basic Availability Group, supports two replicas (one primary and one secondary) with only a single database in the availability group. Enterprise Edition not only allows multiple databases to be configured in a single availability group, but also can have up to nine total replicas (one primary, eight secondary). Enterprise edition also provides other optional benefits such as readable secondary replicas, the ability to make backups off of a secondary replica, and more.

NOTE

Database mirroring, which was deprecated in SQL Server 2012, is not available on the Linux version of SQL Server nor will it be added. Customers still using database mirroring should start planning to migrate to availability groups, which is the replacement for database mirroring.

When it comes to availability, availability groups can provide either automatic or manual failover. Automatic

failover can occur if synchronous data movement is configured and the database on the primary and secondary replica are in a synchronized state. As long as the listener is used and the application uses a later version of .NET (3.5 with an update, or 4.0 and above), the failover should be handled with minimal to no impact to end users if a listener is utilized. Failover to make a secondary replica the new primary replica can be configured to be automatic or manual, and generally is measured in seconds.

The list below highlights some differences with availability groups on Windows Server versus Linux:

- Due to differences in the way the underlying cluster works on Linux and Windows Server, all failovers (manual or automatic) of availability groups are done via the cluster on Linux. On Windows Server-based availability group deployments, manual failovers must be done via SQL Server. Automatic failovers are handled by the underlying cluster on both Windows Server and Linux.
- In SQL Server 2017, the recommended configuration for availability groups on Linux will be a minimum of three replicas. This is due to the way that the underlying clustering works. An improved solution for a two replica configuration will come post-release.
- On Linux, the common name used by each listener is defined in DNS and not in the cluster like it is on Windows Server.

In SQL Server 2017, there are some new features and enhancements to availability groups:

- Cluster types
- REQUIRED_SECONDARIES_TO_COMMIT
- Enhanced Microsoft Distributor Transaction Coordinator (DTC) support for Windows Server-based configurations
- Additional scale out scenarios for read only databases (described later in this article)

Always on availability group cluster types

The built-in availability form of clustering in Windows Server is enabled via a feature named Failover Clustering. It allows you to build a WSFC to be used with an availability group or FCI. Integration for availability groups and FCIs is provided by a cluster-aware resource DLLs shipped by SQL Server.

Each supported Linux distribution ships its own version of the Pacemaker cluster solution. SQL Server 2017 on Linux supports the use of Pacemaker. Pacemaker is an open stack solution that each distribution can then integrate with their stack. While the distributions ship Pacemaker, it is not as integrated as the Failover Clustering feature in Windows Server.

A WSFC and Pacemaker are more similar than different. Both provide a way to take individual servers and combine them in a configuration to provide availability, and have concepts of things like resources, constraints (even if implemented differently), failover, and so on. To support Pacemaker for both availability group and FCI configurations including things like automatic failover, Microsoft provides the mssql-server-ha package, which is similar to, but not exactly the same as, the resource DLLs in a WSFC, for Pacemaker. One of the differences between a WSFC and Pacemaker is that there is no network name resource in Pacemaker, which is the component that helps to abstract the name of the listener (or the name of the FCI) on a WSFC. DNS provides that name resolution on Linux.

Because of the difference in the cluster stack, some changes needed to be made for availability groups because SQL Server has to handle some of the metadata that is natively handled by a WSFC. The most [!IMPORTANT] change is the introduction of a cluster type for an availability group. This is stored in sys.availability_groups in the cluster_type and cluster_type_desc columns. There are three cluster types:

- WSFC
- External
- None

All availability groups that require availability must use an underlying cluster, which in the case of SQL Server 2017 means a WSFC or Pacemaker. For Windows Server-based availability groups that use an underlying WSFC, the

default cluster type is WSFC and does not need to be set. For Linux-based availability groups, when creating the availability group, the cluster type must be set to External. The integration with Pacemaker is configured after the availability group is created, whereas on a WSFC, it is done at creation time.

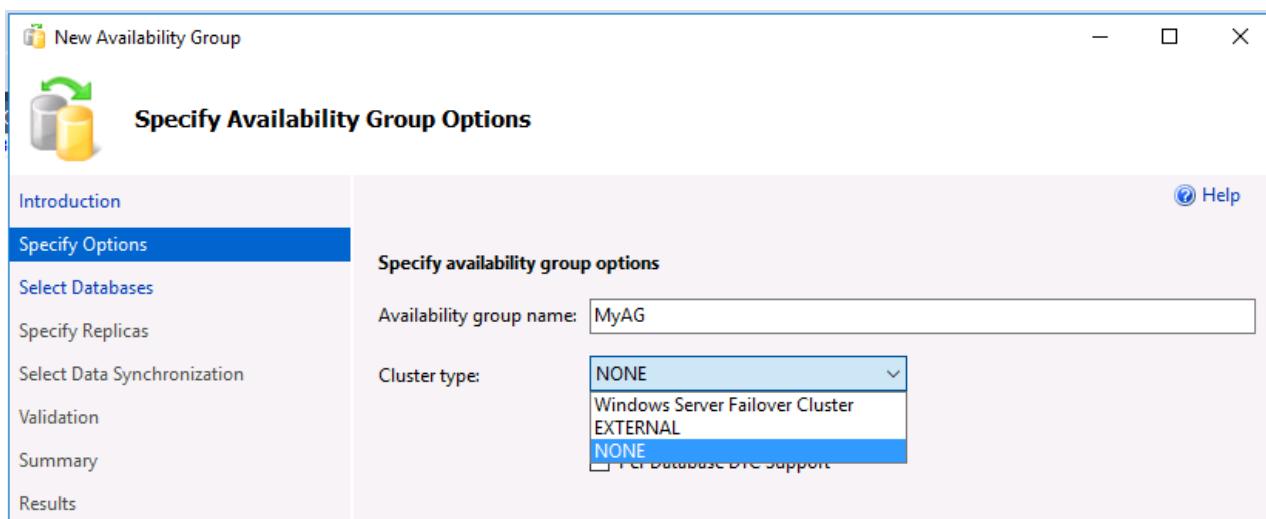
A cluster type of None can be used with both Windows Server and Linux availability groups. Setting the cluster type to None means that the availability group does not require an underlying cluster. This means SQL Server 2017 is the first version of SQL Server to support availability groups without a cluster, but the tradeoff is that this configuration is not supported as a high availability solution.

IMPORTANT

SQL Server 2017 does not allow the ability to change a cluster type for an availability group after it is created. This means that an availability group cannot be switched from None to External or WSFC, or vice versa.

For those who are only looking to just add additional read only copies of a database, or like what an availability group provides for migration/upgrades but do not want to be tied to the additional complexity of an underlying cluster or even the replication, an availability group with a cluster type of None is a perfect solution. For more information, see the sections [Migrations and Upgrades](#) and [read-scale](#).

The screenshot below shows the support for the different kinds of cluster types in SSMS. You must be running version 17.1 or later. The screenshot below is from version 17.2.



REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT

SQL Server 2016 increased support for the number of synchronous replicas from two to three in Enterprise Edition. However, if one secondary replica was synchronized but the other was having a problem, there was no way to control the behavior to tell the primary to either wait for the misbehaving replica or to allow it to move on. This means that the primary replica at some point would continue to receive write traffic even though the secondary replica would not be in a synchronized state, which means that there is data loss on the secondary replica. In SQL Server 2017, there is now an option to be able to control the behavior of what happens when there are synchronous replicas named REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT. The option works as follows:

- There are three possible values: 0, 1, and 2
- The value is the number of secondary replicas that must be synchronized, which has implications for data loss, availability group availability, and failover
- For WSFCs and a cluster type of None, the default value is 0, and can be manually set to 1 or 2
- For a cluster type of External, by default, the cluster mechanism will set this and it can be overridden manually. For three synchronous replicas, the default value will be 1. On Linux, the value for REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT is configured on the availability group resource in the cluster. On Windows, it is set via Transact-SQL.

A value that is higher than 0 ensures higher data protection because if the required number of secondary replicas is not available, the primary will not be available until that is resolved.

REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT also affects failover behavior since automatic failover could not occur if the right number of secondary replicas were not in the proper state. On Linux, a value of 0 will not allow automatic failover, so on Linux, when using synchronous with automatic failover, the value must be set higher than 0 to achieve automatic failover. 0 on Windows Server is the SQL Server 2016 and earlier behavior.

Enhanced Microsoft distributed transaction coordinator support

Before SQL Server 2016, the only way to get availability in SQL Server for applications that require distributed transactions which use DTC underneath the covers was to deploy FCIs. A distributed transaction can be done in one of two ways:

- A transaction that spans more than one database in the same SQL Server instance
- A transaction that spans more than one SQL Server instance or possibly involves a non-SQL Server data source

SQL Server 2016 introduced partial support for DTC with availability groups that covered the latter scenario. SQL Server 2017 completes the story by supporting both scenarios with DTC.

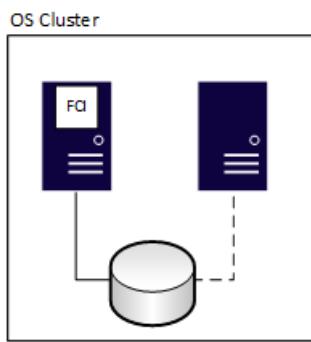
Another enhancement to DTC support for availability groups is that in SQL Server 2016, enabling support for DTC to an availability group could only be done when the availability group was created, and could not be added later. In SQL Server 2017, DTC support can also be added to an availability group after it is created.

NOTE

DTC support can only be configured for databases in Windows Server-based SQL Server instances. If DTC is a requirement for your application, you must use Windows Server as the OS for your SQL Server deployment, and cannot use Linux.

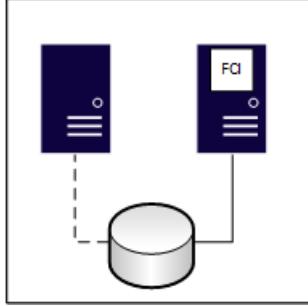
Always on failover cluster instances

Clustered installations have been a feature of SQL Server since version 6.5. FCIs are a proven method of providing availability for the entire installation of SQL Server, known as an instance. This means that everything inside the instance, including databases, SQL Server Agent jobs, linked servers, et al., will move to another server should the underlying server encounter a problem. All FCIs require some sort of shared storage, even if it is provided via networking. The FCI's resources can only be running and owned by one node at any given time. In the picture below, the first node of the cluster owns the FCI, which also means it owns the shared storage resources associated with it denoted by the solid line to the storage.



After a failover, ownership changes as is seen in the picture below.

OS Cluster



There is zero data loss with an FCI, but the underlying shared storage is a single point of failure since there is one copy of the data. FCIs are often combined with another availability method, such as an availability group or log shipping, to have redundant copies of databases. The additional method deployed should use physically separate storage from the FCI. When the FCI fails over to another node, it stops on one node and starts on another, not unlike powering a server off and turning it on. An FCI goes through the normal recovery process, meaning any transactions that need to be rolled forward will be, and any transactions that are incomplete will be rolled back. Therefore, the database is consistent from a data point to the time of the failure or manual failover, hence no data loss. Databases are only available after recovery is complete, so recovery time will depend on many factors, and will generally be longer than failing over an availability group. The tradeoff is that when you fail over an availability group, there may be additional tasks required to make a database usable, such as enabling a SQL Server Agent jobs job.

Like an availability group, FCIs abstract which node of the underlying cluster is hosting it. An FCI always retains the same name. Applications and end users never connect to the nodes; the unique name assigned to the FCI is used. An FCI can participate in an availability group as one of the instances housing either a primary or secondary replica.

The list below highlights some differences with FCIs on Windows Server versus Linux:

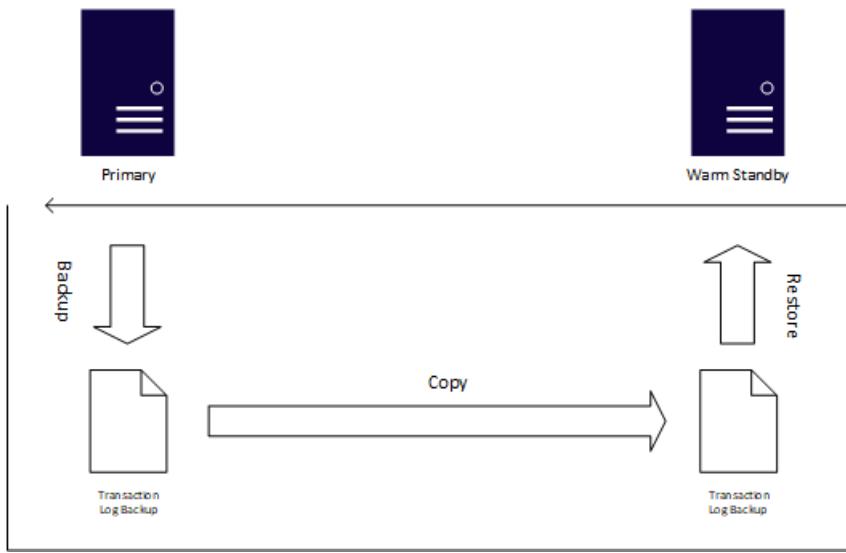
- On Windows Server, an FCI is part of the installation process. An FCI on Linux is configured after installing SQL Server.
- Linux only supports a single installation of SQL Server per host, so all FCIs will be a default instance. Windows Server supports up to 25 FCIs per WSFC.
- The common name used by FCIs in Linux is defined in DNS, and should be the same as the resource created for the FCI.

Log shipping

If recovery point and recovery time objectives are more flexible, or databases are not considered to be highly mission critical, log shipping is another proven availability feature in SQL Server. Based on SQL Server's native backups, the process for log shipping automatically generates transaction log backups, copies them to one or more instances known as a warm standby, and automatically applies the transaction log backups to that standby. Log shipping uses SQL Server Agent jobs to automate the process of backing up, copying, and applying the transaction log backups.

IMPORTANT

On Linux, SQL Server Agent jobs is not included as part of the installation of SQL Server itself. It is available in the package mssql-server-Agent jobs which must also be installed to use log shipping.



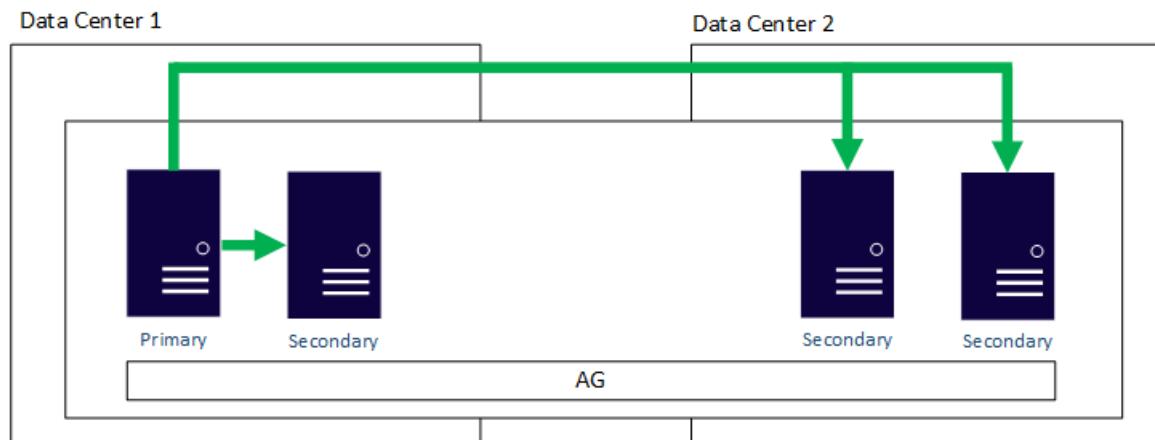
Arguably the biggest advantage of using log shipping in some capacity is that it accounts for human error. The application of transaction logs can be delayed. Therefore, if someone issues something like an UPDATE without a WHERE clause, the standby may not have the change so you could switch to that while you repair the primary system. While log shipping is easy to configure, switching from the primary to a warm standby, known as a role change, is always manual. A role change is initiated via Transact-SQL, and like an availability group, all objects not captured in the transaction log must be manually synchronized. Log shipping also needs to be configured per database, whereas a single availability group can contain multiple databases. Unlike an availability group or FCI, log shipping has no abstraction for a role change. Applications must be able to handle this. Techniques such as a DNS alias (CNAME) could be employed, but there are pros and cons, such as the time it takes for DNS to refresh after the switch.

Disaster recovery

When your primary availability location experiences a catastrophic event like an earthquake or flood, the business must be prepared to have its systems come online elsewhere. This section will cover how the SQL Server availability features can assist with business continuity.

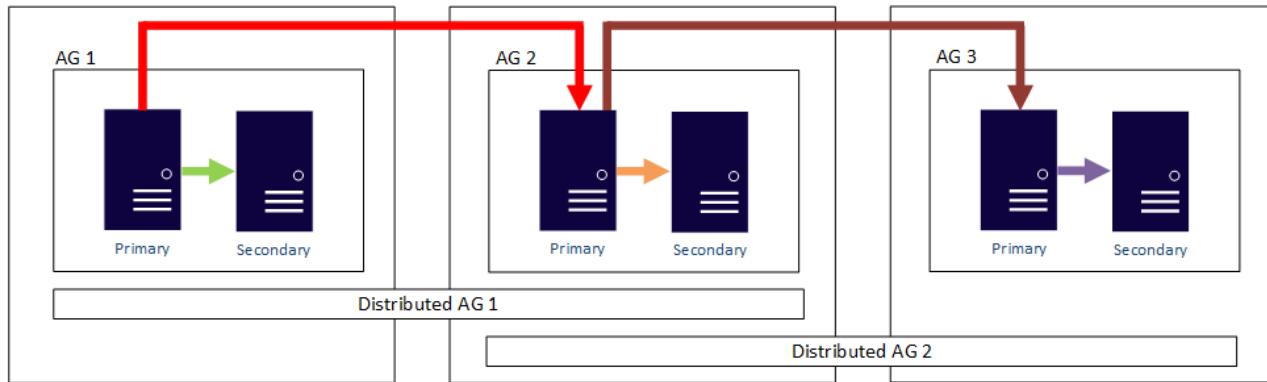
Always on availability groups

One of the benefits of availability groups is that both high availability and disaster recovery can be configured using a single feature. Without the requirement for ensuring that shared storage is also highly available, it is much easier to have replicas that are local in one data center for high availability, and remote ones in other data centers for disaster recovery each with separate storage. Having additional copies of the database is the tradeoff for ensuring redundancy. An example of an availability group that spans multiple data centers is shown below. One primary replica is responsible for keeping all secondary replicas synchronized.



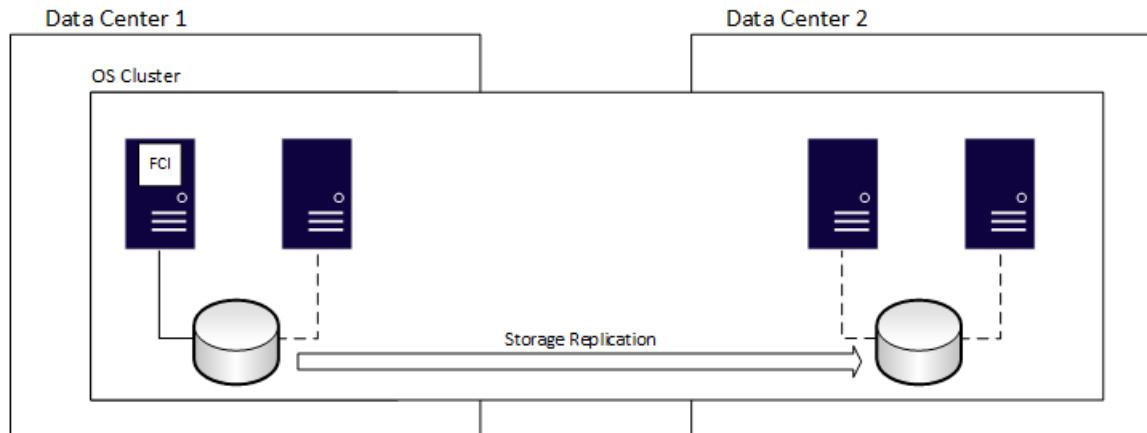
Outside of an availability group with a cluster type of none, an availability group requires that all replicas are part

of the same underlying cluster whether it is a WSFC or Pacemaker. This means that in the picture above, the WSFC is stretched to work in two different data centers which adds complexity. regardless of the platform (Windows Server or Linux). Stretching clusters across distance adds complexity. Introduced in SQL Server 2016, a distributed availability group allows an availability group to span availability groups configured on different clusters. This decouples the requirement to have the nodes all participate in the same cluster, which makes configuring disaster recovery much easier. For more information on distributed availability groups, see [Distributed availability groups](#).



Always on failover cluster instances

FCIs can be used for disaster recovery. As with a normal availability group, the underlying cluster mechanism must also be extended to all locations which adds complexity. There is an additional consideration for FCIs: the shared storage. The same disks need to be available in the primary and secondary sites, so an external method such as functionality provided by the storage vendor at the hardware layer or using storage Replica in Windows Server, is required to ensure that the disks used by the FCI exist elsewhere.



Log shipping

Log shipping is one of the oldest methods of providing disaster recovery for SQL Server databases. Log shipping is often used in conjunction with availability groups and FCIs to provide cost-effective and simpler disaster recovery where other options may be challenging due to environment, administrative skills, or budget. Similar to the high availability story for log shipping, many environments will delay the loading of a transaction log to account for human error.

Migrations and upgrades

When deploying new instances or upgrading old ones, a business cannot tolerate long outage. This section will discuss how the availability features of SQL Server can be used to minimize the downtime in a planned architecture change, server switch, platform change (such as Windows Server to Linux or vice versa), or during patching.

NOTE

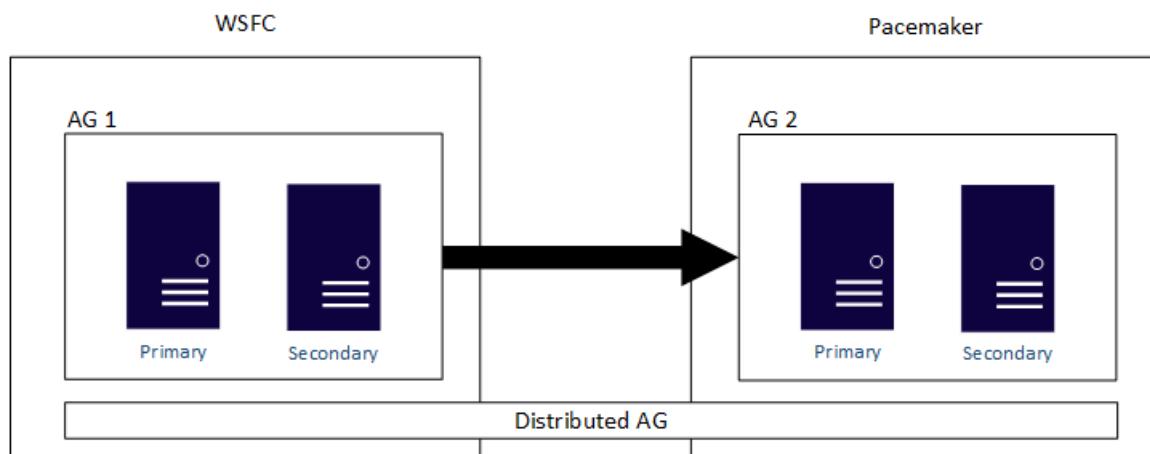
Other methods, such as using backups and restoring them elsewhere, can also be used for migrations and upgrades. They are not discussed in this paper.

Always on availability groups

An existing instance containing one or more availability groups can be upgraded in place to SQL Server 2017. While this will require some amount of downtime, with the right amount of planning, it can be minimized.

If the goal is to migrate to new servers and not change the configuration (including the operating system or SQL Server version), those servers could be added as nodes to the existing underlying cluster and added to the availability group. Once the replica or replicas are in the right state, a manual failover could occur to a new server, and then the old ones could be removed from the availability group, and ultimately, decommissioned.

Distributed AGs are also another method to migrate to a new configuration or upgrade SQL Server. Because a distributed AG supports different underlying AGs on different architectures, for example, you could change from SQL Server 2016 running on Windows Server 2012 R2 to SQL Server 2017 running on Windows Server 2016.



Finally, availability groups with a cluster type of None can also be used for migration or upgrading. You cannot mix and match cluster types in a typical availability group configuration, so all replicas would need to be a type of None. A distributed availability group can be used to span availability groups configured with different cluster types. This method is also supported across the different OS platforms.

All variants of availability groups for migrations and upgrades allow the most time consuming portion of the work to be done over time – data synchronization. When it comes time to initiate the switch to the new configuration, the cutover will be a brief outage versus one long period of downtime where all the work, including data synchronization, would need to be completed.

Availability groups can provide minimal downtime during patching of the underlying OS by manually failing over the primary to a secondary replica while the patching is being completed. From an operating system perspective, doing this would be more common on Windows Server since often, but not always, servicing the underlying OS may require a reboot. Patching Linux sometimes needs a reboot, but it can be infrequent.

[Patching SQL Server instances participating in an availability group](#) can also minimize downtime depending on how complex the availability group architecture is. To patch servers participating in an availability group, a secondary replica is patched first. Once the right number of replicas are patched, the primary replica is manually failed over to another node to do the upgrade. Any remaining secondary replicas at that point can be upgraded, too.

Always on failover cluster instances

FCIs on their own cannot assist with a traditional migration or upgrade; an availability group or log shipping would have to be configured for the databases in the FCI and all other objects accounted for. However, FCIs under

Windows Server are still a popular option for when the underlying Windows Servers need to be patched. A manual failover can be initiated, which means a brief outage instead of having the instance completely unavailable for the entire time Windows Server is being patched. An FCI can be upgraded in place to SQL Server 2017. For information, see [Upgrade a SQL Server Failover Cluster Instance](#).

Log shipping

Log shipping is still a popular option to both migrate and upgrade databases. Similar to availability groups, but this time using the transaction log as the synchronization method, the data propagation can be started well in advance of the server switch. At the time of the switch, once all traffic is stopped at the source, a final transaction log would need to be taken, copied, and applied to the new configuration. At that point, the database can be brought online. Log shipping is often more tolerant of slower networks, and while the switch may be slightly longer than one done using an availability group or a distributed availability group, it is usually measured in minutes – not hours, days, or weeks.

Similar to availability groups, log shipping can provide a way to switch to another server in the event of patching.

Other SQL Server deployment methods and availability

There are two other deployment methods for SQL Server on Linux: containers and using Azure (or another public cloud provider). The general need for availability as presented throughout this paper exists regardless of how SQL Server is deployed. These two methods have some special considerations when it comes to making SQL Server highly available.

[Containers using Docker](#) are a new way of deploying SQL Server, either for Windows Server or Linux. A container is a complete image of SQL Server that is ready to run. However, there is currently no native support for clustering, and thus, direct high availability or disaster recovery. Currently, the options to make SQL Server databases available using containers would be log shipping and backup and restore. While an availability group with a cluster type of None can be configured, as noted earlier, it is not considered a true availability configuration. Microsoft is looking at ways to enable availability groups or FCIs using containers.

If you are using containers today, if the container is lost, depending on the container platform, it can be deployed again and attached to the shared storage that was used. Some of this mechanism is provided by the container orchestrator. While this does provide some resiliency, there will be some downtime associated with database recovery and is not truly highly available as it would be if using an availability group or FCI.

Linux IaaS virtual machines can be deployed with SQL Server installed using Azure. As with on premises-based installations, a supported installation requires the use of STONITH (Shoot the Other Node in the Head) which is external to Pacemaker itself. STONITH is provided via fencing availability agents. Some distributions ship them as part of the platform, others rely on external hardware and software vendors. Check with your preferred Linux distribution to see what forms of STONITH are provided so that a supported solution can be deployed in the public cloud.

Cross-platform and Linux distribution interoperability

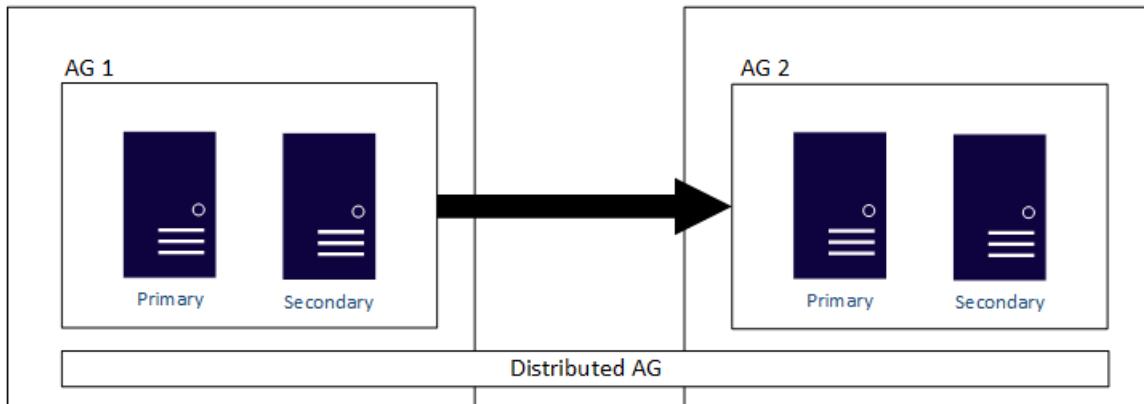
With SQL Server now supported on both Windows Server and Linux, this section covers the scenarios of how they can work together for availability in addition to other purposes, as well as the story for solutions that will incorporate more than one Linux distribution.

Before covering the cross-platform and interoperability scenarios, two facts need to be stated:

- There are no scenarios where a WSFC-based FCI or availability group will work with a Linux-based FCI or availability group directly. A WSFC cannot be extended by a Pacemaker node and vice versa.
- Mixing Linux distributions is not supported with FCIs or an availability group that has a cluster type of External. All availability group replicas in that scenario must be configured not only the same Linux distribution, but also the same version. The two supported ways that SQL Server can operate across the two platforms or multiple distributions of Linux are availability groups and log shipping.

Distributed availability groups

Distributed availability groups are designed to span availability group configurations, whether those two underlying clusters underneath the availability groups are two different WSFCs, Linux distributions, or one on a WSFC and the other on Linux. A distributed availability group will be the primary method of having a cross platform solution. A distributed availability group is also the primary solution for migrations such as converting from a Windows Server-based SQL Server infrastructure to a Linux-based one if that is what your company wants to do. As noted above, availability groups, and especially distributed availability groups, would minimize the time that an application would be unavailable for use. An example of a distributed availability group that spans a WSFC and Pacemaker is shown below.



If an availability group is configured with a cluster type of None, it can span Windows Server and Linux as well as multiple Linux distributions. Since this is not a true high availability configuration, it should not be used for mission critical deployments, but for read-scale or migration/upgrade scenarios.

Log shipping

Since log shipping is just based on backup and restore, and there are no differences in the databases, file structures, etc., for SQL Server on Windows Server versus SQL Server on Linux. This means that log shipping can be configured between a Windows Server-based SQL Server installation and a Linux one as well as between distributions of Linux. Everything else remains the same. The only caveat is that log shipping, just like an availability group, cannot work when the source is at a higher SQL Server major version against a target that is at a lower version of SQL Server.

read-scale

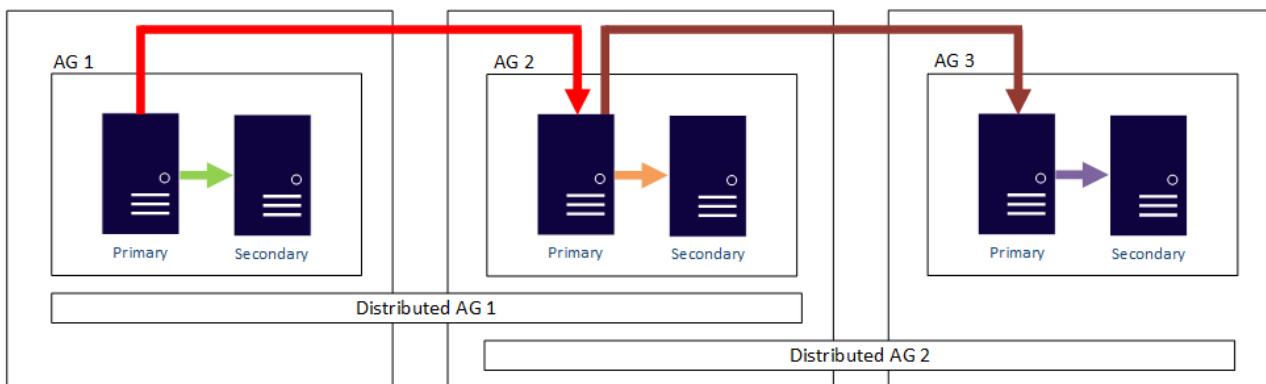
Since their introduction in SQL Server 2012, secondary replicas have had the ability to be used for read-only queries. There are two ways that can be achieved with an availability group: by allowing direct access to the secondary as well as [configuring read only routing](#) which requires the use of the listener. SQL Server 2016 introduced the ability to load balance read-only connections via the listener using a round robin algorithm, allowing read-only requests to be spread across all readable replicas.

NOTE

Readable secondary replicas is a feature only in Enterprise Edition, and each instance hosting a readable replica would need a SQL Server license.

Scaling readable copies of a database via availability groups was first introduced with distributed availability groups in SQL Server 2016. This would allow companies to have read-only copies of the database not only locally, but regionally and globally with a minimal amount of configuration and reduce network traffic and latency by

having queries executed locally. Each primary replica of an availability group can seed two other availability groups even if it is not the fully read/write copy, so each distributed availability group can support up to 27 copies of the data that are readable.



Starting with SQL Server 2017, it is possible to create a near-real time, read-only solution with availability groups configured with a cluster type of None. If the goal is to use availability groups for readable secondary replicas and not availability, doing this removes the complexity of using a WSFC or Pacemaker, and gives the readable benefits of an availability group in a simpler deployment method.

The only major caveat is that due to no underlying cluster with a cluster type of None, configuring read only routing is a little different. From a SQL Server perspective, a listener is still required to route the requests even though there is no cluster. Instead of configuring a traditional listener, the IP address or name of the primary replica is used. The primary replica is then used to route the read only requests.

A log shipping warm standby can technically be configured for readable usage by restoring the database WITH STANDBY. However, because the transaction logs require exclusive use of the database for restoration, it means that users cannot be accessing the database while that happens. This makes log shipping a less than ideal solution – especially if near real-time data is required.

One thing that should be noted for all read-scale scenarios with availability groups is that unlike using transactional replication where all of the data is live, each secondary replica is not in a state where unique indexes can be applied, the replica is an exact copy of the primary. This means that if any indexes are required for reporting or data needs to be manipulated, it must be done on the database(s) on the primary replica. If you need that flexibility, replication is a better solution for readable data.

Summary

Instances and databases of SQL Server 2017 can be made highly available using the same features on both Windows Server and Linux. Besides standard availability scenarios of local high availability and disaster recovery, downtime associated with upgrades and migrations can be minimized with the availability features in SQL Server. Availability groups can also provide additional copies of a database as part of the same architecture to scale out readable copies. Whether you are deploying a new solution using SQL Server 2017 or considering an upgrade, SQL Server 2017 has the availability and reliability you require.

SQL Server availability basics for Linux deployments

2/14/2018 • 18 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Starting with SQL Server 2017, SQL Server is supported on both Linux and Windows. Like Windows-based SQL Server deployments, SQL Server databases and instances need to be highly available under Linux. This article covers the technical aspects of planning and deploying highly available Linux-based SQL Server databases and instances, as well as some of the differences from Windows-based installations. Because SQL Server may be new for Linux professionals, and Linux may be new for SQL Server professionals, the article at times introduces concepts that may be familiar to some and unfamiliar to others.

SQL Server availability options for Linux deployments

Besides backup and restore, the same three availability features are available on Linux as for Windows-based deployments:

- Always On Availability Groups (AGs)
- Always On Failover Cluster Instances (FCIs)
- [Log Shipping](#)

On Windows, FCIs always require an underlying Windows Server failover cluster (WSFC). Depending on the deployment scenario, an AG usually requires an underlying WSFC, with the exception being the new None variant in SQL Server 2017. A WSFC does not exist in Linux. Clustering implementation in Linux is discussed in the section [Pacemaker for Always On Availability Groups and failover cluster instances on Linux](#).

A quick Linux primer

While some Linux installations may be installed with an interface, most are not, meaning that nearly everything at the operating system layer is done via command line. The common term for this command line in the Linux world is a *bash shell*.

In Linux, many commands need to be executed with elevated privileges, much like many things need to be done in Windows Server as an administrator. There are two main methods to execute with elevated privileges:

1. Run in the context of the proper user. To change to a different user, use the command `su`. If `su` is executed on its own without a username, as long as you know the password, you will now be in a shell as *root*.
2. The more common and security conscious way to run things is to use `sudo` before executing anything.
Many of the examples in this article use `sudo`.

Some common commands, each of which have various switches and options that can be researched online:

- `cd` – change the directory
- `chmod` – change the permissions of a file or directory
- `chown` – change the ownership of a file or directory
- `ls` – show the contents of a directory
- `mkdir` – create a folder (directory) on a drive
- `mv` – move a file from one location to another
- `ps` – show all of the working processes

- `rm` – delete a file locally on a server
- `rmdir` – delete a folder (directory)
- `systemctl` – start, stop, or enable services
- Text editor commands. On Linux, there are various text editor options, such as vi and emacs.

Common tasks for availability configurations of SQL Server on Linux

This section covers tasks that are common to all Linux-based SQL Server deployments.

Ensure that files can be copied

Copying files from one server to another is a task that anyone using SQL Server on Linux should be able to do. This task is very important for AG configurations.

Things like permission issues can exist on Linux as well as on Windows-based installations. However, those familiar with how to copy from server to server on Windows may not be familiar with how it is done on Linux. A common method is to use the command-line utility `scp`, which stands for secure copy. Behind the scenes, `scp` uses OpenSSH. SSH stands for secure shell. Depending on the Linux distribution, OpenSSH itself may not be installed. If it is not, OpenSSH needs to be installed first. For more information on configuring OpenSSH, see the information at the following links for each distribution:

- [Red Hat Enterprise Linux \(RHEL\)](#)
- [SUSE Linux Enterprise Server \(SLES\)](#)
- [Ubuntu](#)

When using `scp`, you must provide the credentials of the server if it is not the source or destination. For example, using

```
scp MyAGCert.cer username@servername:/folder/subfolder
```

copies the file `MyAGCert.cer` to the folder specified on the other server. Note that you must have permissions – and possibly ownership – of the file to copy it, so `chown` may also need to be employed before copying. Similarly, on the receiving side, the right user needs access to manipulate the file. For example, to restore that certificate file, the `mssql` user must be able to access it.

Samba, which is the Linux variant of server message block (SMB), can also be used to create shares accessed by UNC paths such as `\\\$ERVERNAME\\SHARE`. For more information on configuring Samba, see the information at the following links for each distribution:

- [RHEL](#)
- [SLES](#)
- [Ubuntu](#)

Windows-based SMB shares can also be used; SMB shares do not need to be Linux-based, as long as the client portion of Samba is configured properly on the Linux server hosting SQL Server and the share has the right access. For those in a mixed environment, this would be one way to leverage existing infrastructure for Linux-based SQL Server deployments.

One thing that is important is that the version of Samba deployed should be SMB 3.0 compliant. When SMB support was added in SQL Server 2012, it required all shares to support SMB 3.0. If using Samba for the share and not Windows Server, the Samba-based share should be using Samba 4.0 or later, and ideally 4.3 or later, which supports SMB 3.1.1. A good source of information on SMB and Linux is [SMB3 in Samba](#).

Finally, using a network file system (NFS) share is an option. Using NFS is not an option on Windows-based deployments of SQL Server, and can only be used for Linux-based deployments.

Configure the firewall

Similar to Windows, Linux distributions have a built-in firewall. If your company is using an external firewall to the servers, disabling the firewalls in Linux may be acceptable. However, regardless of where the firewall is enabled, ports need to be opened. The following table documents the common ports needed for highly available SQL Server deployments on Linux.

PORT NUMBER	TYPE	DESCRIPTION
111	TCP/UDP	NFS – <code>rpcbind/sunrpc</code>
135	TCP	Samba (if used) – End Point Mapper
137	UDP	Samba (if used) – NetBIOS Name Service
138	UDP	Samba (if used) – NetBIOS Datagram
139	TCP	Samba (if used) – NetBIOS Session
445	TCP	Samba (if used) – SMB over TCP
1433	TCP	SQL Server – default port; if desired, can change with <code>mssql-conf set network.tcpport <portnumber></code>
2049	TCP, UDP	NFS (if used)
2224	TCP	Pacemaker – used by <code>pcsd</code>
3121	TCP	Pacemaker – Required if there are Pacemaker Remote nodes
3260	TCP	iSCSI Initiator (if used) – Can be altered in <code>/etc/iscsi/iscsid.config</code> (RHEL), but should match port of iSCSI Target
5022	TCP	SQL Server - default port used for an AG endpoint; can be changed when creating the endpoint
5403	TCP	Pacemaker
5404	UDP	Pacemaker – Required by Corosync if using multicast UDP
5405	UDP	Pacemaker – Required by Corosync
21064	TCP	Pacemaker – Required by resources using DLM
Variable	TCP	AG endpoint port; default is 5022

PORT NUMBER	TYPE	DESCRIPTION
Variable	TCP	NFS – port for <code>LOCKD_TCPPORT</code> (found in <code>/etc/sysconfig/nfs</code> on RHEL)
Variable	UDP	NFS – port for <code>LOCKD_UDPPORT</code> (found in <code>/etc/sysconfig/nfs</code> on RHEL)
Variable	TCP/UDP	NFS – port for <code>MOUNTD_PORT</code> (found in <code>/etc/sysconfig/nfs</code> on RHEL)
Variable	TCP/UDP	NFS – port for <code>STATD_PORT</code> (found in <code>/etc/sysconfig/nfs</code> on RHEL)

For additional ports that may be used by Samba, see [Samba Port Usage](#).

Conversely, the name of the service under Linux can also be added as an exception instead of the port; for example, `high-availability` for Pacemaker. Refer to your distribution for the names if this is the direction you wish to pursue. For example, on RHEL the command to add in Pacemaker is

```
sudo firewall-cmd --permanent --add-service=high-availability
```

Firewall documentation:

- [RHEL](#)
- [SLES](#)

Install SQL Server packages for availability

On a Windows-based SQL Server installation, some components are installed even in a basic engine install, while others are not. Under Linux, only the SQL Server engine is installed as part of the installation process. Everything else is optional. For highly available SQL Server instances under Linux, two packages should be installed with SQL Server: SQL Server Agent (`mssql-server-agent`) and the high availability (HA) package (`mssql-server-ha`). While SQL Server Agent is technically optional, it is SQL Server's scheduler for jobs and is required by log shipping, so installation is recommended. On Windows-based installations, SQL Server Agent is not optional.

NOTE

For those new to SQL Server, SQL Server Agent is SQL Server's built-in job scheduler. It is a common way for DBAs to schedule things like backups and other SQL Server maintenance. Unlike a Windows-based installation of SQL Server where SQL Server Agent is a completely different service, on Linux, SQL Server Agent runs in context of SQL Server itself.

When AGs or FCIs are configured on a Windows-based configuration, they are cluster-aware. Cluster awareness means that SQL Server has specific resource DLLs that a WSFC knows about (sqagtres.dll and sqsrvres.dll for FCIs, hadrres.dll for AGs) and are used by the WSFC to ensure that the SQL Server clustered functionality is up, running, and functioning properly. Because clustering is external not only to SQL Server but Linux itself, Microsoft had to code the equivalent of a resource DLL for Linux-based AG and FCI deployments. This is the `mssql-server-ha` package, also known as the SQL Server resource agent for Pacemaker. To install the `mssql-server-ha` package, see [Install the HA and SQL Server Agent packages](#).

The other optional packages for SQL Server on Linux, SQL Server Full-Text Search (`mssql-server-fts`) and SQL Server Integration Services (`mssql-server-is`), are not required for high availability, either for an FCI or an AG.

Pacemaker for Always On Availability Groups and failover cluster

instances on Linux

As previously noted, the only clustering mechanism currently supported by Microsoft for AGs and FCIs is Pacemaker with Corosync. This section covers the basic information to understand the solution, as well as how to plan and deploy it for SQL Server configurations.

HA add-on/extension basics

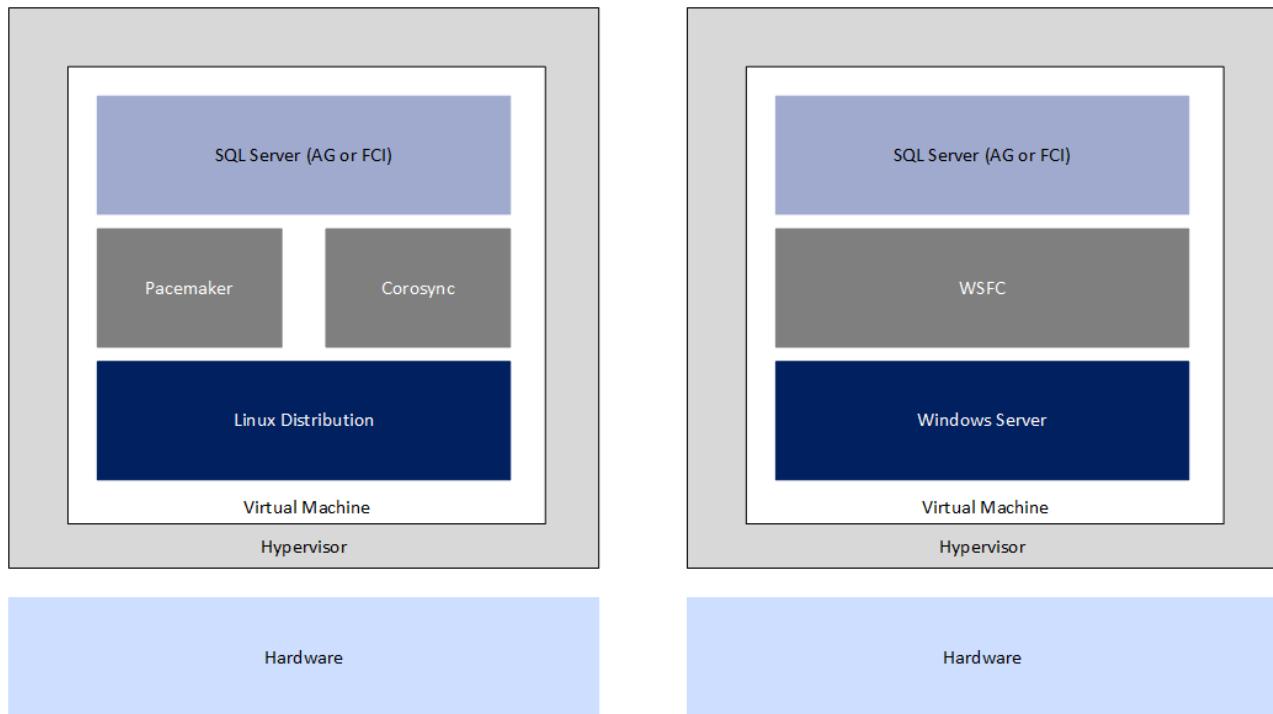
All of the currently supported distributions ship a high availability add-on/extension, which is based on the Pacemaker clustering stack. This stack incorporates two key components: Pacemaker and Corosync. All the components of the stack are:

- Pacemaker – The core clustering component, that does things like coordinate across the clustered machines.
- Corosync – A framework and set of APIs that provides things like quorum, the ability to restart failed processes, and so on.
- libQB – Provides things like logging.
- Resource agent – Specific functionality provided so that an application can integrate with Pacemaker.
- Fence agent – Scripts/functionality that assist in isolating nodes and deal with them if they are having issues.

NOTE

The cluster stack is commonly referred to as Pacemaker in the Linux world.

This solution is in some ways similar to, but in many ways different from deploying clustered configurations using Windows. In Windows, the availability form of clustering, called a Windows Server failover cluster (WSFC), is built into the operating system, and the feature that enables the creation of a WSFC, failover clustering, is disabled by default. In Windows, AGs and FCIs are built on top of a WSFC, and share tight integration because of the specific resource DLL that is provided by SQL Server. This tightly coupled solution is possible by and large because it is all from one vendor.



On Linux, while each supported distribution has Pacemaker available, each distribution can customize and have slightly different implementations and versions. Some of the differences will be reflected in the instructions in this article. The clustering layer is open source, so even though it ships with the distributions, it is not tightly integrated in the same way a WSFC is under Windows. This is why Microsoft provides `mssql-server-ha`, so that SQL Server and the Pacemaker stack can provide close to, but not exactly the same, experience for AGs and FCIs as under

Windows.

For full documentation on Pacemaker, including a more in-depth explanation of what everything is with full reference information, for RHEL and SLES:

- [RHEL](#)
- [SLES](#)

Ubuntu does not have a guide for availability.

For more information about the whole stack, also see the official [Pacemaker documentation page](#) on the Clusterlabs site.

Pacemaker concepts and terminology

This section documents the common concepts and terminology for a Pacemaker implementation.

Node

A node is a server participating in the cluster. A Pacemaker cluster natively supports up to 16 nodes. This number can be exceeded if Corosync is not running on additional nodes, but Corosync is required for SQL Server.

Therefore, the maximum number of nodes a cluster can have for any SQL Server-based configuration is 16; this is the Pacemaker limit, and has nothing to do with maximum limitations for AGs or FCIs imposed by SQL Server.

Resource

Both a WSFC and a Pacemaker cluster have the concept of a resource. A resource is specific functionality that runs in context of the cluster, such as a disk or an IP address. For example, under Pacemaker both FCI and AG resources can get created. This is not dissimilar to what is done in a WSFC, where you see a SQL Server resource for either an FCI or an AG resource when configuring an AG, but is not exactly the same due to the underlying differences in how SQL Server integrates with Pacemaker.

Pacemaker has standard and clone resources. Clone resources are ones that run simultaneously on all nodes. An example would be an IP address that runs on multiple nodes for load balancing purposes. Any resource that gets created for FCIs uses a standard resource, since only one node can host an FCI at any given time.

When an AG is created, it requires a specialized form of a clone resource called a multi-state resource. While an AG only has one primary replica, the AG itself is running across all nodes that it is configured to work on, and can potentially allow things such as read-only access. Because this is a “live” use of the node, the resources have the concept of two states: master and slave. For more information, see [Multi-state resources: Resources that have multiple modes](#).

Resource groups/sets

Similar to roles in a WSFC, a Pacemaker cluster has the concept of a resource group. A resource group (called a set in SLES) is a collection of resources that function together and can fail over from one node to another as a single unit. Resource groups cannot contain resources that are configured as master/slave; thus, they cannot be used for AGs. While a resource group can be used for FCIs, it is not generally a recommended configuration.

Constraints

WSFCs have various parameters for resources as well as things like dependencies, which tell the WSFC of a parent/child relationship between two different resources. A dependency is just a rule telling the WSFC which resource needs to be online first.

A Pacemaker cluster does not have the concept of dependencies, but there are constraints. There are three kinds of constraints: colocation, location, and ordering.

- A colocation constraint enforces whether or not two resources should be running on the same node.
- A location constraint tells the Pacemaker cluster where a resource can (or cannot) run.
- An ordering constraint tells the Pacemaker cluster the order in which the resources should start.

NOTE

Colocation constraints are not required for resources in a resource group, since all of those are seen as a single unit.

Quorum, fence agents, and STONITH

Quorum under Pacemaker is somewhat similar to a WSFC in concept. The whole purpose of a cluster's quorum mechanism is to ensure that the cluster stays up and running. Both a WSFC and the HA add-ons for the Linux distributions have the concept of voting, where each node counts towards quorum. You want a majority of the votes up, otherwise, in a worst case scenario, the cluster will be shut down.

Unlike a WSFC, there is no witness resource to work with quorum. Like a WSFC, the goal is to keep the number of voters odd. Quorum configuration has different considerations for AGs than FCIs.

WSFCs monitor the status of the nodes participating and handle them when a problem occurs. Later versions of WSFCs offer such features as quarantining a node that is misbehaving or unavailable (node is not on, network communication is down, etc.). On the Linux side, this type of functionality is provided by a fence agent. The concept is sometimes referred to as fencing. However, these fence agents are generally specific to the deployment, and often provided by hardware vendors and some software vendors, such as those who provide hypervisors. For example, VMware provides a fence agent that can be used for Linux VMs virtualized using vSphere.

Quorum and fencing ties into another concept called STONITH, or Shoot the Other Node in the Head. STONITH is required to have a supported Pacemaker cluster on all Linux distributions. For more information, see [Fencing in a Red Hat High Availability Cluster](#) (RHEL).

corosync.conf

The `corosync.conf` file contains the configuration of the cluster. It is located in `/etc/corosync`. In the course of normal day-to-day operations, this file should never have to be edited if the cluster is set up properly.

Cluster log location

Log locations for Pacemaker clusters differ depending on the distribution.

- RHEL and SLES – `/var/log/cluster/corosync.log`
- Ubuntu – `/var/log/corosync/corosync.log`

To change the default logging location, modify `corosync.conf`.

Plan Pacemaker clusters for SQL Server

This section discusses the important planning points for a Pacemaker cluster.

Virtualizing Linux-based Pacemaker clusters for SQL Server

Using virtual machines to deploy Linux-based SQL Server deployments for AGs and FCIs is covered by the same rules as for their Windows-based counterparts. There is a base set of rules for supportability of virtualized SQL Server deployments provided by Microsoft in [Microsoft Support KB 956893](#). Different hypervisors such as Microsoft's Hyper-V and VMware's ESXi may have different variances on top of that, due to differences in the platforms themselves.

When it comes to AGs and FCIs under virtualization, ensure that anti-affinity is set for the nodes of a given Pacemaker cluster. When configured for high availability in an AG or FCI configuration, the VMs hosting SQL Server should never be running on the same hypervisor host. For example, if a two-node FCI is deployed, there would need to be *at least* three hypervisor hosts so that there is somewhere for one of the VMs hosting a node to go in the event of a host failure, especially if using features like Live Migration or vMotion.

For more specifics, consult:

- Hyper-V Documentation – [Using Guest Clustering for High Availability](#)

- Whitepaper (written for Windows-based deployments, but most of the concepts still apply) – [Planning Highly Available, Mission Critical SQL Server Deployments with VMware vSphere](#)

NOTE

RHEL with a Pacemaker cluster with STONITH is not yet supported by Hyper-V. Until that is supported, for more information and updates, consult [Support Policies for RHEL High Availability Clusters](#).

Networking

Unlike a WSFC, Pacemaker does not require a dedicated name or at least one dedicated IP address for the Pacemaker cluster itself. AGs and FCIs will require IP addresses (see the documentation for each for more information), but not names, since there is no network name resource. SLES does allow the configuration of an IP address for administration purposes, but it is not required, as can be seen in [Create the Pacemaker cluster](#).

Like a WSFC, Pacemaker would prefer redundant networking, meaning distinct network cards (NICs or pNICs for physical) having individual IP addresses. In terms of the cluster configuration, each IP address would have what is known as its own ring. However, as with WSFCs today, many implementations are virtualized or in the public cloud where there is really only a single virtualized NIC (vNIC) presented to the server. If all pNICs and vNICs are connected to the same physical or virtual switch, there is no true redundancy at the network layer, so configuring multiple NICs is a bit of an illusion to the virtual machine. Network redundancy is usually built into the hypervisor for virtualized deployments, and is definitely built into the public cloud.

One difference with multiple NICs and Pacemaker versus a WSFC is that Pacemaker allows multiple IP addresses on the same subnet, whereas a WSFC does not. For more information on multiple subnets and Linux clusters, see the article [Configure multiple subnets](#).

Quorum and STONITH

Quorum configuration and requirements are related to AG or FCI-specific deployments of SQL Server.

STONITH is required for a supported Pacemaker cluster. Use the documentation from the distribution to configure STONITH. An example is at [Storage-based Fencing](#) for SLES. There is also a STONITH agent for VMware vCenter for ESXi-based solutions. For more information, see [Stonith Plugin Agent for VMWare VM VCenter SOAP Fencing \(Unofficial\)](#).

NOTE

As of the writing of this article, Hyper-V does not have a solution for STONITH. This is true for on premises deployments and also impacts Azure-based Pacemaker deployments using certain distributions such as RHEL.

Interoperability

This section documents how a Linux-based cluster can interact with a WSFC or with other distributions of Linux.

WSFC

Currently, there is no direct way for a WSFC and a Pacemaker cluster to work together. This means that there is no way to create an AG or FCI that works across a WSFC and Pacemaker. However, there are two interoperability solutions, both of which are designed for AGs. The only way an FCI can participate in a cross-platform configuration is if it is participating as an instance in one of these two scenarios:

- An AG with a cluster type of None. For more information, see the Windows [availability groups documentation](#).
- A distributed AG, which is a special type of availability group that allows two different AGs to be configured as their own availability group. For more information on distributed AGs, see the documentation [Distributed Availability Groups](#).

Other Linux distributions

On Linux, all nodes of a Pacemaker cluster must be on the same distribution. For example, this means that a RHEL node cannot be part of a Pacemaker cluster that has a SLES node. The main reason for this was previously stated: the distributions may have different versions and functionality, so things could not work properly. Mixing distributions has the same story as mixing WSFCs and Linux: use None or distributed AGs.

Next steps

[Deploy a Pacemaker cluster for SQL Server on Linux](#)

Backup and restore SQL Server databases on Linux

2/14/2018 • 3 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✔ SQL Server (Linux only) ✖ Azure SQL Database ✖ Azure SQL Data Warehouse ✖ Parallel Data Warehouse

You can take backups of databases from SQL Server 2017 on Linux with the same tools as other platforms. On a Linux server, you can use **sqlcmd** to connect to the SQL Server and take backups. From Windows, you can connect to SQL Server on Linux and take backups with the user interface. The backup functionality is the same across platforms. For example, you can backup databases locally, to remote drives, or to [Microsoft Azure Blob storage service](#).

Backup a database

In the following example **sqlcmd** connects to the local SQL Server instance and takes a full backup of a user database called `demodb`.

```
sqlcmd -S localhost -U SA -Q "BACKUP DATABASE [demodb] TO DISK = N'/var/opt/mssql/data/demodb.bak' WITH NOFORMAT, NOINIT, NAME = 'demodb-full', SKIP, NOREWIND, NOUNLOAD, STATS = 10"
```

When you run the command, SQL Server will prompt for a password. After you enter the password, the shell will return the results of the backup progress. For example:

```
Password:  
10 percent processed.  
21 percent processed.  
32 percent processed.  
40 percent processed.  
51 percent processed.  
61 percent processed.  
72 percent processed.  
80 percent processed.  
91 percent processed.  
Processed 296 pages for database 'demodb', file 'demodb' on file 1.  
100 percent processed.  
Processed 2 pages for database 'demodb', file 'demodb_log' on file 1.  
BACKUP DATABASE successfully processed 298 pages in 0.064 seconds (36.376 MB/sec).
```

Backup the transaction log

If your database is in the full recovery model, you can also make transaction log backups for more granular restore options. In the following example, **sqlcmd** connects to the local SQL Server instance and takes a transaction log backup.

```
sqlcmd -S localhost -U SA -Q "BACKUP LOG [demodb] TO DISK = N'/var/opt/mssql/data/demodb_LogBackup.bak' WITH NOFORMAT, NOINIT, NAME = N'demodb_LogBackup', NOSKIP, NOREWIND, NOUNLOAD, STATS = 5"
```

Restore a database

In the following example **sqlcmd** connects to the local instance of SQL Server and restores the `demodb` database. Note that the `NORECOVERY` option is used to allow for additional restores of log file backups. If you do not plan to restore additional log files, remove the `NORECOVERY` option.

```
sqlcmd -S localhost -U SA -Q "RESTORE DATABASE [demodb] FROM DISK = N'/var/opt/mssql/data/demodb.bak' WITH FILE = 1, NOUNLOAD, REPLACE, NORECOVERY, STATS = 5"
```

TIP

If you accidentally use NORECOVERY but do not have additional log file backups, run the command `RESTORE DATABASE demodb` with no additional parameters. This finishes the restore and leaves your database operational.

Restore the transaction log

The following command restores the previous transaction log backup.

```
sqlcmd -S localhost -U SA -Q "RESTORE LOG demodb FROM DISK = N'/var/opt/mssql/data/demodb_LogBackup.bak'"
```

Backup and Restore with SQL Server Management Studio (SSMS)

You can use SSMS from a Windows computer to connect to a Linux database and take a backup through the user-interface.

NOTE

Use the latest version of SSMS to connect to SQL Server. To download and install the latest version, see [Download SSMS](#). For more information on how to use SSMS, see [Use SSMS to Manage SQL Server on Linux](#).

The following steps walk through taking a backup with SSMS.

1. Start SSMS and connect to your server in SQL Server 2017 on Linux.
2. In Object Explorer, right-click on your database, Click **Tasks**, and then click **Back Up...**.
3. In the **Backup Up Database** dialog, verify the parameters and options, and click **OK**.

SQL Server completes the database backup.

Restore with SQL Server Management Studio (SSMS)

The following steps walk you through restoring a database with SSMS.

1. In SSMS right-click **Databases** and click **Restore Databases....**
2. Under **Source** click **Device:** and then click the ellipses (...).
3. Locate your database backup file and click **OK**.
4. Under **Restore plan**, verify the backup file and settings. Click **OK**.
5. SQL Server restores the database.

See also

- [Create a Full Database Backup \(SQL Server\)](#)
- [Back up a Transaction Log \(SQL Server\)](#)
- [BACKUP \(Transact-SQL\)](#)
- [SQL Server Backup to URL](#)

SQL Server on Linux VDI client SDK Specification

2/14/2018 • 10 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This document covers the interfaces provided by the SQL Server on Linux virtual device interface (VDI) client SDK. Independent software vendors (ISVs) can use the Virtual Backup Device Application Programming Interface (API) to integrate SQL Server into their products. In general, VDI on Linux behaves similarly to VDI on Windows with the following changes:

- Windows Shared Memory becomes POSIX shared memory.
- Windows Semaphores become POSIX semaphores.
- Windows types like HRESULT and DWORD are changed to integer equivalents.
- The COM interfaces are removed and replaced with a pair of C++ Classes.
- SQL Server on Linux does not support named instances so references to instance name have been removed.
- The shared library is implemented in libsqlvdi.so installed at /opt/mssql/lib/libsqlvdi.so

This document is an addendum to **vbackup.chm** that details the Windows VDI Specification. Download the [Windows VDI Specification](#).

Also review the sample VDI backup solution on the [SQL Server Samples GitHub repository](#).

User Permissions Setup

On Linux, POSIX primitives are owned by the user creating them and their default group. For objects created by SQL Server, these will by default be owned by the mssql user and the mssql group. To allow sharing between SQL Server and the VDI client, one of the following two methods are recommended:

1. Run the VDI Client as the mssql user

Execute the following command to switch to mssql user:

```
sudo su mssql
```

2. Add the mssql user to the vdiuser's group, and the vdiuser to the mssql group.

Execute the following commands:

```
sudo useradd vdiuser
sudo usermod -a -G mssql vdiuser
sudo usermod -a -G vdiuser mssql
```

Restart the server to pick up new groups for SQL Server and vdiuser

Client Functions

This chapter contains descriptions of each of the client functions. The descriptions include the following information:

- Function purpose
- Function syntax

- Parameter list
- Return values
- Remarks

ClientVirtualDeviceSet::Create

Purpose This function creates the virtual device set.

Syntax

```
int ClientVirtualDeviceSet::Create (
    char * name,           // name for the set
    VDConfig * cfg         // configuration for the set
);
```

PARAMETERS	ARGUMENT	EXPLANATION
	name	This identifies the virtual device set. The rules for names used by CreateFileMapping() must be followed. Any character except backslash () may be used. This is a character string. Prefixing the string with the user's product or company name and database name is recommended.
	cfg	This is the configuration for the virtual device set. For more information, see "Configuration" later in this document.

RETURN VALUES	ARGUMENT	EXPLANATION
	NOERROR	The function succeeded.
	VD_E_NOTSUPPORTED	One or more of the fields in the configuration was invalid or otherwise unsupported.
	VD_E_PROTOCOL	The virtual device set already exists.

Remarks The Create method should be called only once per BACKUP or RESTORE operation. After invoking the Close method, the client can reuse the interface to create another virtual device set.

ClientVirtualDeviceSet::GetConfiguration

Purpose This function is used to wait for the server to configure the virtual device set. **Syntax**

```
int ClientVirtualDeviceSet::GetConfiguration (
    time_t      timeout,    // in milliseconds
    VDConfig *   cfg // selected configuration
);
```

PARAMETERS	ARGUMENT	EXPLANATION
------------	----------	-------------

PARAMETERS	ARGUMENT	EXPLANATION
	timeout	This is the time-out in milliseconds. Use INFINITE or any negative integer to prevent time-out.
	cfg	Upon successful execution, this contains the configuration selected by the server. For more information, see "Configuration" later in this document.
RETURN VALUES	ARGUMENT	EXPLANATION
	NOERROR	The configuration was returned.
	VD_E_ABORT	SignalAbort was invoked.
	VD_E_TIMEOUT	The function timed out.

Remarks This function blocks in an Alertable state. After successful invocation, the devices in the virtual device set may be opened.

ClientVirtualDeviceSet::OpenDevice

Purpose This function opens one of the devices in the virtual device set. **Syntax**

```
int ClientVirtualDeviceSet::OpenDevice (
    char *          name,      // name for the set
    ClientVirtualDevice ** ppVirtualDevice // returns interface to device
);
```

PARAMETERS	ARGUMENT	EXPLANATION
	name	This identifies the virtual device set.
	ppVirtualDevice	When the function succeeds, a pointer to the virtual device is returned. This device is used for GetCommand and CompleteCommand.
RETURN VALUES	ARGUMENT	EXPLANATION
	NOERROR	The function succeeded.
	VD_E_ABORT	Abort was requested.
	VD_E_OPEN	All devices are open.
	VD_E_PROTOCOL	The set is not in the initializing state or this particular device is already open.
	VD_E_INVALID	The device name is invalid. It is not one of the names known to comprise the set.

Remarks VD_E_OPEN may be returned without problem. The client may call OpenDevice by means of a loop until this code is returned. If more than one device is configured, for example *n* devices, the virtual device set will return *n* unique device interfaces.

The `GetConfiguration` function can be used to wait until the devices can be opened. If this function does not succeed, then a null value is returned through the `ppVirtualDevice`.

ClientVirtualDevice::GetCommand

Purpose This function is used to obtain the next command queued to a device. When requested, this function waits for the next command.

Syntax

```
int ClientVirtualDevice::GetCommand (
    time_t      timeout,    // time-out in milliseconds
    VDC_Command** ppCmd    // returns the next command
);
```

PARAMETERS	ARGUMENT	EXPLANATION
	timeout	This is the time to wait, in milliseconds. Use INFINITE to wait indefinitely. Use 0 to poll for a command. VD_E_TIMEOUT is returned if no command is currently available . If the time-out occurs, the client decides the next action.
	Timeout	This is the time to wait, in milliseconds. Use INFINITE or a negative value to wait indefinitely. Use 0 to poll for a command. VD_E_TIMEOUT is returned if no command is available before the timeout expires. If the timeout occurs, the client decides the next action.
	ppCmd	When a command is successfully returned, the parameter returns the address of a command to execute. The memory returned is read-only. When the command is completed, this pointer is passed to the CompleteCommand routine. For details about each command, see "Commands" later in this document.
RETURN VALUES	ARGUMENT	EXPLANATION
	NOERROR	A command was fetched.
	VD_E_CLOSE	The device has been closed by the server.
	VD_E_TIMEOUT	No command was available and the time-out expired.

RETURN VALUES	ARGUMENT	EXPLANATION
	VD_E_ABORT	Either the client or the server has used the SignalAbort to force a shutdown.

Remarks When VD_E_CLOSE is returned, SQL Server has closed the device. This is part of the normal shutdown. After all devices have been closed, the client invokes ClientVirtualDeviceSet::Close to close the virtual device set. When this routine must block to wait for a command, the thread is left in an Alertable condition.

ClientVirtualDevice::CompleteCommand

Purpose This function is used to notify SQL Server that a command has finished. Completion information appropriate for the command should be returned. For more information, see "Commands" later in this document.

Syntax

```
int ClientVirtualDevice::CompleteCommand (
    VDC_Command pCmd,           // the command
    int completionCode,         // completion code
    unsigned long bytesTransferred, // bytes transferred
    int64_t position           // current position
);
```

PARAMETERS	ARGUMENT	EXPLANATION
	pCmd	This is the address of a command previously returned from ClientVirtualDevice::GetCommand.
	completionCode	This is a status code that indicates the completion status. This parameter must be returned for all commands. The code returned should be appropriate to the command being performed. ERROR_SUCCESS is used in all cases to denote a successfully executed command. For the complete list of possible codes, see the file, vdierror.h. A list of typical status codes for each command appears in "Commands" later in this document.
	bytesTransferred	This is the number of successfully transferred bytes. This is returned only for data transfer commands Read and Write.
	position	This is a response to the GetPosition command only.
RETURN VALUES	ARGUMENT	EXPLANATION
	NOERROR	The completion was correctly noted.
	VD_E_INVALID	pCmd was not an active command.

RETURN VALUES	ARGUMENT	EXPLANATION
	VD_E_ABORT	Abort was signaled.
	VD_E_PROTOCOL	The device is not open.

Remarks None

ClientVirtualDeviceSet::SignalAbort

Purpose This function is used to signal that an abnormal termination should occur.

Syntax

```
int ClientVirtualDeviceSet::SignalAbort ();
```

PARAMETERS	ARGUMENT	EXPLANATION
	None	Not applicable
RETURN VALUES	ARGUMENT	EXPLANATION
	NOERROR	The Abort notification was successfully posted.

Remarks At any time, the client may choose to abort the BACKUP or RESTORE operation. This routine signals that all operations should cease. The state of the overall virtual device set enters an Abnormally Terminated state. No further commands are returned on any devices. All uncompleted commands are automatically completed, returning ERROR_OPERATION_ABORTED as a completion code. The client should call ClientVirtualDeviceSet::Close after it has safely terminated any outstanding use of buffers provided to the client. For more information, see "Abnormal Termination" earlier in this document.

ClientVirtualDeviceSet::Close

Purpose This function closes the virtual device set created by ClientVirtualDeviceSet::Create. It results in the release of all resources associated with the virtual device set.

Syntax

```
int ClientVirtualDeviceSet::Close ();
```

PARAMETERS	ARGUMENT	EXPLANATION
	None	Not applicable
RETURN VALUES	ARGUMENT	EXPLANATION
	NOERROR	This is returned when the virtual device set was successfully closed.

RETURN VALUES	ARGUMENT	EXPLANATION
	VD_E_PROTOCOL	No action was taken because the virtual device set was not open.
	VD_E_OPEN	Devices were still open.

Remarks The invocation of Close is a client declaration that all resources used by the virtual device set should be released. The client must ensure that all activity involving data buffers and virtual devices is terminated before invoking Close. All virtual device interfaces returned by OpenDevice are invalidated by Close. The client is permitted to issue a Create call on the virtual device set interface after the Close call is returned. Such a call would create a new virtual device set for a subsequent BACKUP or RESTORE operation. If Close is called when one or more virtual devices are still open, VD_E_OPEN is returned. In this case, SignalAbort is internally triggered, to ensure a proper shutdown if possible. VDI resources are released. The client should wait for a VD_E_CLOSE indication on each device before invoking ClientVirtualDeviceSet::Close. If the client knows that the virtual device set is already in an Abnormally Terminated state, then it should not expect a VD_E_CLOSE indication from GetCommand, and may invoke ClientVirtualDeviceSet::Close as soon as activity on the shared buffers is terminated. For more information, see "Abnormal Termination" earlier in this document.

ClientVirtualDeviceSet::OpenInSecondary

Purpose This function opens the virtual device set in a secondary client. The primary client must have already used Create and GetConfiguration to set up the virtual device set.

Syntax

```
int ClientVirtualDeviceSet::OpenInSecondary (
    char * setName           // name of the set
);
```

PARAMETERS	ARGUMENT	EXPLANATION
	setName	This identifies the set. This name is case-sensitive and must match the name used by the primary client when it invoked ClientVirtualDeviceSet::Create.
RETURN VALUES	ARGUMENT	EXPLANATION
	NOERROR	The function succeeded.
	VD_E_PROTOCOL	The virtual device set has not been created, has already been opened on this client, or the virtual device set is not ready to accept open requests from secondary clients.
	VD_E_ABORT	The operation is being aborted.

Remarks When using a multiple process model, the primary client is responsible for detecting normal and abnormal termination of secondary clients.

ClientVirtualDeviceSet::GetBufferHandle

Purpose Some applications may require more than one process to operate on the buffers returned by ClientVirtualDevice::GetCommand. In such cases, the process that receives the command can use GetBufferHandle to obtain a process independent handle that identifies the buffer. This handle can then be communicated to any other process that also has the same Virtual Device Set open. That process would then use ClientVirtualDeviceSet::MapBufferHandle to obtain the address of the buffer. The address will likely be a different address than in its partner because each process may be mapping buffers at different addresses.

Syntax

```
int ClientVirtualDeviceSet::GetBufferHandle (
    uint8_t*      pBuffer,          // in: buffer address
    unsigned int*   pBufferHandle    // out: buffer handle
);
```

PARAMETERS	ARGUMENT	EXPLANATION
	pBuffer	This is the address of a buffer obtained from a Read or Write command.
	BufferHandle	A unique identifier for the buffer is returned.
RETURN VALUES	ARGUMENT	EXPLANATION
	NOERROR	The function succeeded.
	VD_E_PROTOCOL	The virtual device set is not currently open.
	VD_E_INVALID	The pBuffer is not a valid address.

Remarks The process that invokes the GetBufferHandle function is responsible for invoking ClientVirtualDevice::CompleteCommand when the data transfer is complete.

ClientVirtualDeviceSet::MapBufferHandle

Purpose This function is used to obtain a valid buffer address from a buffer handle obtained from some other process.

Syntax

```
int ClientVirtualDeviceSet::MapBufferHandle (
    int      dwBuffer,    // in: buffer handle
    uint8_t** ppBuffer     // out: buffer address
);
```

PARAMETERS	ARGUMENT	EXPLANATION
	dwBuffer	This is the handle returned by ClientVirtualDeviceSet::GetBufferHandle.
	ppBuffer	This is the address of the buffer that is valid in the current process.

RETURN VALUES	ARGUMENT	EXPLANATION
	NOERROR	The function succeeded.
	VD_E_PROTOCOL	The virtual device set is not currently open.
	VD_E_INVALID	The ppBuffer is an invalid handle.

Remarks Care must be taken to communicate the handles correctly. Handles are local to a single virtual device set. The partner processes sharing a handle must ensure that buffer handles are used only within the scope of the virtual device set from which the buffer was originally obtained.

Failover Cluster Instances - SQL Server on Linux

2/14/2018 • 5 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Linux only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

This article explains the concepts related to SQL Server failover cluster instances (FCI) on Linux.

To create a SQL Server FCI on Linux, see [Configure SQL Server FCI on Linux](#)

The Clustering Layer

- In RHEL, the clustering layer is based on Red Hat Enterprise Linux (RHEL) [HA add-on](#).

NOTE

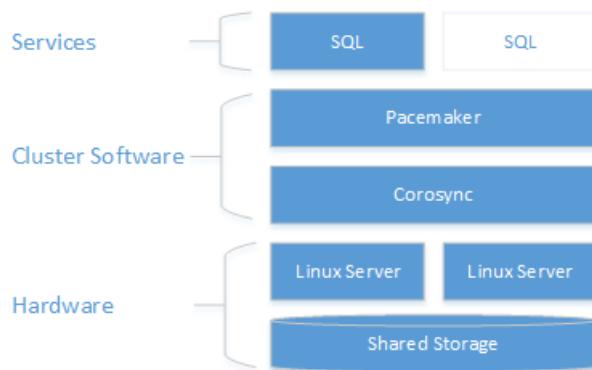
Access to Red Hat HA add-on and documentation requires a subscription.

- In SLES, the clustering layer is based on SUSE Linux Enterprise [High Availability Extension \(HAE\)](#).

For more information on cluster configuration, resource agent options, management, best practices, and recommendations, see [SUSE Linux Enterprise High Availability Extension 12 SP2](#).

Both the RHEL HA add-on and the SUSE HAE are built on [Pacemaker](#).

As the following diagram shows, storage is presented to two servers. Clustering components - Corosync and Pacemaker - coordinate communications and resource management. One of the servers has the active connection to the storage resources and the SQL Server. When Pacemaker detects a failure the clustering components manage moving the resources to the other node.



NOTE

At this point, SQL Server's integration with Pacemaker on Linux is not as coupled as with WSFC on Windows. From within SQL, there is no knowledge about the presence of the cluster, all orchestration is outside in and the service is controlled as a standalone instance by Pacemaker. Also, virtual network name is specific to WSFC, there is no equivalent of the same in Pacemaker. It is expected that @@servername and sys.servers to return the node name, while the cluster dmvs sys.dm_os_cluster_nodes and sys.dm_os_cluster_properties will no records. To use a connection string that points to a string server name and not use the IP, they will have to register in their DNS server the IP used to create the virtual IP resource (as explained in the following sections) with the chosen server name.

Number of Instances and Nodes

One key difference with SQL Server on Linux is that there can only be one install of SQL Server per Linux server. That installation is called an instance. This means that unlike Windows Server which supports up to 25 FCIs per Windows Server failover cluster (WSFC), a Linux-based FCI will only have a single instance. This one instance is also a default instance; there is no concept of a named instance on Linux.

A Pacemaker cluster can only have up to 16 nodes when Corosync is involved, so a single FCI can span up to 16 servers. An FCI implemented with Standard Edition of SQL Server supports up to two nodes of a cluster even if the Pacemaker cluster has the maximum 16 nodes.

In a SQL Server FCI, the SQL Server instance is active on either one node or the other.

IP Address and Name

On a Linux Pacemaker cluster, each SQL Server FCI needs its own unique IP address and name. If the FCI configuration spans multiple subnets, one IP address will be required per subnet. The unique name and IP address(es) are used to access the FCI so that applications and end users do not need to know which underlying server of the Pacemaker cluster.

The name of the FCI in DNS should be the same as the name of the FCI resource that gets created in the Pacemaker cluster. Both the name and IP address must be registered in DNS.

Shared Storage

All FCIs, whether they are on Linux or Windows Server, require some form of shared storage. This storage is presented to all servers that can possibly host the FCI, but only a single server can use the storage for the FCI at any given time. The options available for shared storage under Linux are:

- iSCSI
- Network File System (NFS)
- Server Message Block (SMB) Under Windows Server, there are slightly different options. One option not currently supported for Linux-based FCIs is the ability to use a disk that is local to the node for TempDB, which is SQL Server's temporary workspace.

In a configuration that spans multiple locations, what is stored at one data center must be synchronized with the other. In the event of a failover, the FCI will be able to come online and the storage is seen to be the same.

Achieving this will require some external method for storage replication, whether it is done via the underlying storage hardware or some software-based utility.

NOTE

For SQL Server 2017, Linux-based deployments using disks presented directly to a server such must be formatted with XFS or EXT4. Other file systems are currently not supported. Any changes will be reflected here.

The process for presenting shared storage is the same for the different supported methods:

- Configure the shared storage
- Mount the storage as a folder to the servers that will serve as nodes of the Pacemaker cluster for the FCI
- If required, move the SQL Server system databases to shared storage
- Test that SQL Server works from each server connected to the shared storage

One major difference with SQL Server on Linux is that while you can configure the default user data and log file location, the system databases must always exist at `/var/opt/mssql/data`. On Windows Server, there is the ability to move the system databases including TempDB. This fact plays into how shared storage is configured for an FCI.

The default paths for non-system databases can be changed using the `mssql-conf` utility. For information on how to change the defaults, [Change the default data or log directory location](#). You can also store SQL Server data and transaction in other locations as long as they have the proper security even if it is not a default location; the location would need to be stated.

The following topics explain how to configure supported storage types for a Linux based SQL Server FCI:

- [Configure failover cluster instance - iSCSI - SQL Server on Linux](#)
- [Configure failover cluster instance - NFS - SQL Server on Linux](#)
- [Configure failover cluster instance - SMB - SQL Server on Linux](#)

Configure Red Hat Enterprise Linux shared disk cluster for SQL Server

2/14/2018 • 10 min to read • [Edit Online](#)

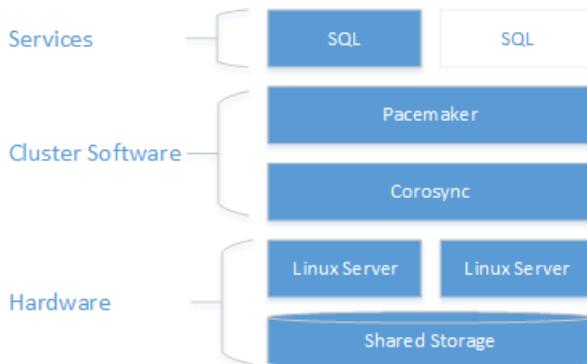
THIS TOPIC APPLIES TO: ✓ SQL Server (Linux only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

This guide provides instructions to create a two-node shared disk cluster for SQL Server on Red Hat Enterprise Linux. The clustering layer is based on Red Hat Enterprise Linux (RHEL) [HA add-on](#) built on top of [Pacemaker](#). The SQL Server instance is active on either one node or the other.

NOTE

Access to Red Hat HA add-on and documentation requires a subscription.

As the following diagram shows, storage is presented to two servers. Clustering components - Corosync and Pacemaker - coordinate communications and resource management. One of the servers has the active connection to the storage resources and the SQL Server. When Pacemaker detects a failure the clustering components manage moving the resources to the other node.



For more information on cluster configuration, resource agents options, and management, visit [RHEL reference documentation](#).

NOTE

At this point, SQL Server's integration with Pacemaker is not as coupled as with WSFC on Windows. From within SQL, there is no knowledge about the presence of the cluster, all orchestration is outside in and the service is controlled as a standalone instance by Pacemaker. Also for example, cluster dmvs sys.dm_os_cluster_nodes and sys.dm_os_cluster_properties will have no records. To use a connection string that points to a string server name and not use the IP, they will have to register in their DNS server the IP used to create the virtual IP resource (as explained in the following sections) with the chosen server name.

The following sections walk through the steps to set up a failover cluster solution.

Prerequisites

To complete the following end-to-end scenario, you need two machines to deploy the two nodes cluster and another server to configure the NFS server. Below steps outline how these servers will be configured.

Setup and configure the operating system on each cluster node

The first step is to configure the operating system on the cluster nodes. For this walk through, use RHEL with a valid subscription for the HA add-on.

Install and configure SQL Server on each cluster node

1. Install and setup SQL Server on both nodes. For detailed instructions, see [Install SQL Server on Linux](#).
2. Designate one node as primary and the other as secondary, for purposes of configuration. Use these terms for the following this guide.
3. On the secondary node, stop and disable SQL Server.

The following example stops and disables SQL Server:

```
sudo systemctl stop mssql-server
sudo systemctl disable mssql-server
```

NOTE

At setup time, a Server Master Key is generated for the SQL Server instance and placed at `/var/opt/mssql/secrets/machine-key`. On Linux, SQL Server always runs as a local account called mssql. Because it's a local account, its identity isn't shared across nodes. Therefore, you need to copy the encryption key from primary node to each secondary node so each local mssql account can access it to decrypt the Server Master Key.

4. On the primary node, create a SQL server login for Pacemaker and grant the login permission to run `sp_server_diagnostics`. Pacemaker uses this account to verify which node is running SQL Server.

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

Connect to the SQL Server `master` database with the sa account and run the following:

```
USE [master]
GO
CREATE LOGIN [<loginName>] with PASSWORD= N'<loginPassword>'

ALTER SERVER ROLE [sysadmin] ADD MEMBER [<loginName>]
```

Alternatively, you can set the permissions at a more granular level. The Pacemaker login requires `VIEW SERVER STATE` to query health status with `sp_server_diagnostics`, `setupadmin` and `ALTER ANY LINKED SERVER` to update the FCI instance name with the resource name by running `sp_dropserver` and `sp_addserver`.

5. On the primary node, stop and disable SQL Server.
6. Configure the hosts file for each cluster node. The host file must include the IP address and name of every cluster node.

Check the IP address for each node. The following script shows the IP address of your current node.

```
sudo ip addr show
```

Set the computer name on each node. Give each node a unique name that is 15 characters or less. Set the

computer name by adding it to `/etc/hosts`. The following script lets you edit `/etc/hosts` with `vi`.

```
sudo vi /etc/hosts
```

The following example shows `/etc/hosts` with additions for two nodes named `sqlfcivm1` and `sqlfcivm2`.

```
127.0.0.1 localhost localhost4 localhost4.localdomain4
::1 localhost localhost6 localhost6.localdomain6
10.128.18.128 sqlfcivm1
10.128.16.77 sqlfcivm2
```

In the next section you will configure shared storage and move your database files to that storage.

Configure shared storage and move database files

There are a variety of solutions for providing shared storage. This walk-through demonstrates configuring shared storage with NFS. We recommend to follow best practices and use Kerberos to secure NFS (you can find an example here: <https://www.certdepot.net/rhel7-use-kerberos-control-access-nfs-network-shares/>).

WARNING

If you do not secure NFS, then anyone who can access your network and spoof the IP address of a SQL node will be able to access your data files. As always, make sure you threat model your system before using it in production. Another storage option is to use SMB fileshare.

Configure shared storage with NFS

IMPORTANT

Hosting database files on a NFS server with version <4 is not supported in this release. This includes using NFS for shared disk failover clustering as well as databases on non-clustered instances. We are working on enabling other NFS server versions in the upcoming releases.

On the NFS Server do the following:

1. Install `nfs-utils`

```
sudo yum -y install nfs-utils
```

2. Enable and start `rpcbind`

```
sudo systemctl enable rpcbind && sudo systemctl start rpcbind
```

3. Enable and start `nfs-server`

```
sudo systemctl enable nfs-server && sudo systemctl start nfs-server
```

4. Edit `/etc/exports` to export the directory you want to share. You need 1 line for each share you want. For example:

```
/mnt/nfs 10.8.8.0/24(rw,sync,no_subtree_check,no_root_squash)
```

5. Export the shares

```
sudo exportfs -rav
```

6. Verify that the paths are shared/exported, run from the NFS server

```
sudo showmount -e
```

7. Add exception in SELinux

```
sudo setsebool -P nfs_export_all_rw 1
```

8. Open the firewall the server.

```
sudo firewall-cmd --permanent --add-service=nfs
sudo firewall-cmd --permanent --add-service=mountd
sudo firewall-cmd --permanent --add-service=rpc-bind
sudo firewall-cmd --reload
```

Configure all cluster nodes to connect to the NFS shared storage

Do the following steps on all cluster nodes.

1. Install `nfs-utils`

```
sudo yum -y install nfs-utils
```

2. Open up the firewall on clients and NFS server

```
sudo firewall-cmd --permanent --add-service=nfs
sudo firewall-cmd --permanent --add-service=mountd
sudo firewall-cmd --permanent --add-service=rpc-bind
sudo firewall-cmd --reload
```

3. Verify that you can see the NFS shares on client machines

```
sudo showmount -e <IP OF NFS SERVER>
```

4. Repeat these steps on all cluster nodes.

For more information about using NFS, see the following resources:

- [NFS servers and firewalld | Stack Exchange](#)
- [Mounting an NFS Volume | Linux Network Administrators Guide](#)
- [NFS server configuration](#)

Mount database files directory to point to the shared storage

1. **On the primary node only**, save the database files to a temporary location. The following script creates a new temporary directory, copies the database files to the new directory, and removes the old database files. As SQL Server runs as local user mssql, you need to make sure that after data transfer to the mounted share,

local user has read-write access to the share.

```
$ sudo su mssql
$ mkdir /var/opt/mssql/tmp
$ cp /var/opt/mssql/data/* /var/opt/mssql/tmp
$ rm /var/opt/mssql/data/*
$ exit
```

2. On all cluster nodes edit `/etc/fstab` file to include the mount command.

```
<IP OF NFS SERVER>:<shared_storage_path> <database_files_directory_path> nfs timeo=14,intr
```

The following script shows an example of the edit.

```
10.8.8.0:/mnt/nfs /var/opt/mssql/data nfs timeo=14,intr
```

NOTE

If using a File System (FS) resource as recommended here, there is no need to preserve the mounting command in `/etc/fstab`. Pacemaker will take care of mounting the folder when it starts the FS clustered resource. With the help of fencing, it will ensure the FS is never mounted twice.

3. Run `mount -a` command for the system to update the mounted paths.
4. Copy the database and log files that you saved to `/var/opt/mssql/tmp` to the newly mounted share `/var/opt/mssql/data`. This only needs to be done **on the primary node**. Make sure that you give read write permissions to 'mssql' local user.

```
$ sudo chown mssql /var/opt/mssql/data
$ sudo chgrp mssql /var/opt/mssql/data
$ sudo su mssql
$ cp /var/opt/mssql/tmp/* /var/opt/mssql/data/
$ rm /var/opt/mssql/tmp/*
$ exit
```

5. Validate that SQL Server starts successfully with the new file path. Do this on each node. At this point only one node should run SQL Server at a time. They cannot both run at the same time because they will both try to access the data files simultaneously (to avoid accidentally starting SQL Server on both nodes, use a File System cluster resource to make sure the share is not mounted twice by the different nodes). The following commands start SQL Server, check the status, and then stop SQL Server.

```
sudo systemctl start mssql-server
sudo systemctl status mssql-server
sudo systemctl stop mssql-server
```

At this point both instances of SQL Server are configured to run with the database files on the shared storage. The next step is to configure SQL Server for Pacemaker.

Install and configure Pacemaker on each cluster node

1. On both cluster nodes, create a file to store the SQL Server username and password for the Pacemaker login. The following command creates and populates this file:

```
sudo touch /var/opt/mssql/secrets/passwd
echo '<loginName>' | sudo tee -a /var/opt/mssql/secrets/passwd
echo '<loginPassword>' | sudo tee -a /var/opt/mssql/secrets/passwd
sudo chown root:root /var/opt/mssql/secrets/passwd
sudo chmod 600 /var/opt/mssql/secrets/passwd
```

- On both cluster nodes, open the Pacemaker firewall ports. To open these ports with `firewalld`, run the following command:

```
sudo firewall-cmd --permanent --add-service=high-availability
sudo firewall-cmd --reload
```

If you're using another firewall that doesn't have a built-in high-availability configuration, the following ports need to be opened for Pacemaker to be able to communicate with other nodes in the cluster

- TCP: Ports 2224, 3121, 21064
- UDP: Port 5405

- Install Pacemaker packages on each node.

```
sudo yum install pacemaker pcs fence-agents-all resource-agents
```

- Set the password for the default user that is created when installing Pacemaker and Corosync packages. Use the same password on both nodes.

```
sudo passwd hacluster
```

- Enable and start `pcsd` service and Pacemaker. This will allow nodes to rejoin the cluster after the reboot. Run the following command on both nodes.

```
sudo systemctl enable pcsd
sudo systemctl start pcsd
sudo systemctl enable pacemaker
```

- Install the FCI resource agent for SQL Server. Run the following commands on both nodes.

```
sudo yum install mssql-server-ha
```

Create the cluster

- On one of the nodes, create the cluster.

```
sudo pcs cluster auth <nodeName1 nodeName2 ...> -u hacluster
sudo pcs cluster setup --name <clusterName> <nodeName1 nodeName2 ...>
sudo pcs cluster start --all
```

RHEL HA add-on has fencing agents for VMWare and KVM. Fencing needs to be disabled on all other

hypervisors. Disabling fencing agents is not recommended in production environments. As of timeframe, there are no fencing agents for HyperV or cloud environments. If you are running one of these configurations, you need to disable fencing. **This is NOT recommended in a production system!**

The following command disables the fencing agents.

```
sudo pcs property setstonith-enabled=false  
sudo pcs property set start-failure-is-fatal=false
```

2. Configure the cluster resources for SQL Server, File System and virtual IP resources and push the configuration to the cluster. You need the following information:

- **SQL Server Resource Name:** A name for the clustered SQL Server resource.
- **Floating IP Resource Name:** A name for the virtual IP address resource.
- **IP Address:** The IP address that clients will use to connect to the clustered instance of SQL Server.
- **File System Resource Name:** A name for the File System resource.
- **device:** The NFS share path
- **device:** The local path that it's mounted to the share
- **fstype:** File share type (i.e. nfs)

Update the values from the following script for your environment. Run on one node to configure and start the clustered service.

```
sudo pcs cluster cib cfg  
sudo pcs -f cfg resource create <sqlServerResourceName> ocf:mssql:fci  
sudo pcs -f cfg resource create <floatingIPResourceName> ocf:heartbeat:IPaddr2 ip=<ip Address>  
sudo pcs -f cfg resource create <fileShareResourceName> Filesystem device=<networkPath> directory=  
<localPath> fstype=<fileShareType>  
sudo pcs -f cfg constraint colocation add <virtualIPResourceName> <sqlResourceName>  
sudo pcs -f cfg constraint colocation add <fileShareResourceName> <sqlResourceName>  
sudo pcs cluster cib-push cfg
```

For example, the following script creates a SQL Server clustered resource named `mssqlha`, and a floating IP resources with IP address `10.0.0.99`. It also creates a Filesystem resource and adds constraints so all resources are colocated on same node as SQL resource.

```
sudo pcs cluster cib cfg  
sudo pcs -f cfg resource create mssqlha ocf:mssql:fci  
sudo pcs -f cfg resource create virtualip ocf:heartbeat:IPaddr2 ip=10.0.0.99  
sudo pcs -f cfg resource create fs Filesystem device="10.8.8.0:/mnt/nfs" directory="/var/opt/mssql/data"  
fstype="nfs"  
sudo pcs -f cfg constraint colocation add virtualip mssqlha  
sudo pcs -f cfg constraint colocation add fs mssqlha  
sudo pcs cluster cib-push cfg
```

After the configuration is pushed, SQL Server will start on one node.

3. Verify that SQL Server is started.

```
sudo pcs status
```

The following examples shows the results when Pacemaker has successfully started a clustered instance of SQL Server.

```
fs      (ocf::heartbeat:Filesystem):    Started sqlfcivm1
virtualip   (ocf::heartbeat:IPaddr2):    Started sqlfcivm1
mssqlha   (ocf::mssql:fci): Started sqlfcivm1

PCSD Status:
sqlfcivm1: Online
sqlfcivm2: Online

Daemon Status:
corosync: active/disabled
pacemaker: active/enabled
pcsd: active/enabled
```

Additional resources

- [Cluster from Scratch](#) guide from Pacemaker

Next steps

[Operate SQL Server on Red Hat Enterprise Linux shared disk cluster](#)

Operate Red Hat Enterprise Linux shared disk cluster for SQL Server

2/14/2018 • 5 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Linux only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

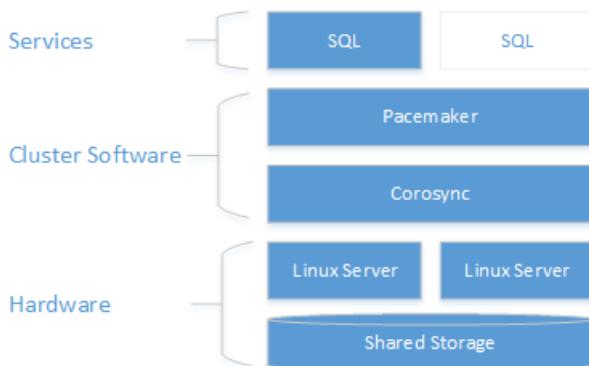
This document describes how to do the following tasks for SQL Server on a shared disk failover cluster with Red Hat Enterprise Linux.

- Manually failover the cluster
- Monitor a failover cluster SQL Server service
- Add a cluster node
- Remove a cluster node
- Change the SQL Server resource monitoring frequency

Architecture description

The clustering layer is based on Red Hat Enterprise Linux (RHEL) [HA add-on](#) built on top of [Pacemaker](#). Corosync and Pacemaker coordinate cluster communications and resource management. The SQL Server instance is active on either one node or the other.

The following diagram illustrates the components in a Linux cluster with SQL Server.



For more information on cluster configuration, resource agents options, and management, visit [RHEL reference documentation](#).

Failover cluster manually

The `resource move` command creates a constraint forcing the resource to start on the target node. After executing the `move` command, executing `resource clear` will remove the constraint so it is possible to move the resource again or have the resource automatically fail over.

```
sudo pcs resource move <sqlResourceName> <targetNodeName>
sudo pcs resource clear <sqlResourceName>
```

The following example moves the **mssqlha** resource to a node named **sqlfcivm2**, and then removes the constraint so that the resource can move to a different node later.

```
sudo pcs resource move mssqlha sqlfcivm2
sudo pcs resource clear mssqlha
```

Monitor a failover cluster SQL Server service

View the current cluster status:

```
sudo pcs status
```

View live status of cluster and resources:

```
sudo crm_mon
```

View the resource agent logs at `/var/log/cluster/corosync.log`

Add a node to a cluster

1. Check the IP address for each node. The following script shows the IP address of your current node.

```
ip addr show
```

2. The new node needs a unique name that is 15 characters or less. By default in Red Hat Linux the computer name is `localhost.localdomain`. This default name may not be unique and is too long. Set the computer name the new node. Set the computer name by adding it to `/etc/hosts`. The following script lets you edit `/etc/hosts` with `vi`.

```
sudo vi /etc/hosts
```

The following example shows `/etc/hosts` with additions for three nodes named `sqlfcivm1`, `sqlfcivm2`, and `sqlfcivm3`.

```
127.0.0.1 localhost localhost4 localhost4.localdomain4
::1 localhost localhost6 localhost6.localdomain6
10.128.18.128 fcivm1
10.128.16.77 fcivm2
10.128.14.26 fcivm3
```

The file should be the same on every node.

3. Stop the SQL Server service on the new node.
4. Follow the instructions to mount the database file directory to the shared location:

From the NFS server, install `nfs-utils`

```
sudo yum -y install nfs-utils
```

Open up the firewall on clients and NFS server

```
sudo firewall-cmd --permanent --add-service=nfs
sudo firewall-cmd --permanent --add-service=mountd
sudo firewall-cmd --permanent --add-service=rpc-bind
sudo firewall-cmd --reload
```

Edit /etc/fstab file to include the mount command:

```
<IP OF NFS SERVER>:<shared_storage_path> <database_files_directory_path> nfs timeo=14,intr
```

Run `mount -a` for the changes to take effect.

5. On the new node, create a file to store the SQL Server username and password for the Pacemaker login. The following command creates and populates this file:

```
sudo touch /var/opt/mssql/passwd
sudo echo "<loginName>" >> /var/opt/mssql/secrets/passwd
sudo echo "<loginPassword>" >> /var/opt/mssql/secrets/passwd
sudo chown root:root /var/opt/mssql/passwd
sudo chmod 600 /var/opt/mssql/passwd
```

6. On the new node, open the Pacemaker firewall ports. To open these ports with `firewalld`, run the following command:

```
sudo firewall-cmd --permanent --add-service=high-availability
sudo firewall-cmd --reload
```

NOTE

If you're using another firewall that doesn't have a built-in high-availability configuration, the following ports need to be opened for Pacemaker to be able to communicate with other nodes in the cluster

- TCP: Ports 2224, 3121, 21064
- UDP: Port 5405

7. Install Pacemaker packages on the new node.

```
sudo yum install pacemaker pcs fence-agents-all resource-agents
```

8. Set the password for the default user that is created when installing Pacemaker and Corosync packages. Use the same password as the existing nodes.

```
sudo passwd hacluster
```

9. Enable and start `pcsd` service and Pacemaker. This will allow the new node to rejoin the cluster after the reboot. Run the following command on the new node.

```
sudo systemctl enable pcsd
sudo systemctl start pcsd
sudo systemctl enable pacemaker
```

10. Install the FCI resource agent for SQL Server. Run the following commands on the new node.

```
sudo yum install mssql-server-ha
```

11. On an existing node from the cluster, authenticate the new node and add it to the cluster:

```
sudo pcs    cluster auth <nodeName3> -u hacluster  
sudo pcs    cluster node add <nodeName3>
```

The following example adds a node named **vm3** to the cluster.

```
sudo pcs    cluster auth  
sudo pcs    cluster start
```

Remove nodes from a cluster

To remove a node from a cluster run the following command:

```
sudo pcs    cluster node remove <nodeName>
```

Change the frequency of sqlservr resource monitoring interval

```
sudo pcs    resource op monitor interval=<interval>s <sqlResourceName>
```

The following example sets the monitoring interval to 2 seconds for the mssql resource:

```
sudo pcs    resource op monitor interval=2s mssqlha
```

Troubleshoot Red Hat Enterprise Linux shared disk cluster for SQL Server

In troubleshooting the cluster it may help to understand how the three daemons work together to manage cluster resources.

DAEMON	DESCRIPTION
Corosync	Provides quorum membership and messaging between cluster nodes.
Pacemaker	Resides on top of Corosync and provides state machines for resources.
PCSD	Manages both Pacemaker and Corosync through the <code>pcs</code> tools

PCSD must be running in order to use `pcs` tools.

Current cluster status

`sudo pcs status` returns basic information about the cluster, quorum, nodes, resources, and daemon status for each node.

An example of a healthy pacemaker quorum output would be:

```
Cluster name: MyAppSQL
Last updated: Wed Oct 31 12:00:00 2016 Last change: Wed Oct 31 11:00:00 2016 by root via crm_resource on
sqlvmmnode1
Stack: corosync
Current DC: sqlvmmnode1 (version 1.1.13-10.el7_2.4-44eb2dd) - partition with quorum
3 nodes and 1 resource configured

Online: [ sqlvmmnode1 sqlvmmnode2 sqlvmmnode3]

Full list of resources:

mssqlha (ocf::sql:fci): Started sqlvmmnode1

PCSD Status:
sqlvmmnode1: Online
sqlvmmnode2: Online
sqlvmmnode3: Online

Daemon Status:
corosync: active/disabled
pacemaker: active/enabled
```

In the example, `partition with quorum` means that a majority quorum of nodes is online. If the cluster loses a majority quorum of nodes, `pcs status` will return `partition WITHOUT quorum` and all resources will be stopped.

`online: [sqlvmmnode1 sqlvmmnode2 sqlvmmnode3]` returns the name of all nodes currently participating in the cluster. If any nodes are not participating, `pcs status` returns `OFFLINE: [<nodename>]`.

`PCSD Status` shows the cluster status for each node.

Reasons why a node may be offline

Check the following items when a node is offline.

- **Firewall**

The following ports need to be open on all nodes for Pacemaker to be able to communicate.

- **TCP: 2224, 3121, 21064

- **Pacemaker or Corosync services running**

- **Node communication**

- **Node name mappings**

Additional resources

- [Cluster from Scratch](#) guide from Pacemaker

Next steps

[Configure Red Hat Enterprise Linux shared disk cluster for SQL Server](#)

Configure SLES shared disk cluster for SQL Server

2/14/2018 • 8 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This guide provides instructions to create a two-nodes shared disk cluster for SQL Server on SUSE Linux Enterprise Server (SLES). The clustering layer is based on SUSE [High Availability Extension \(HAE\)](#) built on top of [Pacemaker](#).

For more information on cluster configuration, resource agent options, management, best practices, and recommendations, see [SUSE Linux Enterprise High Availability Extension 12 SP2](#).

Prerequisites

To complete the following end-to-end scenario, you need two machines to deploy the two nodes cluster and another server to configure the NFS share. Below steps outline how these servers will be configured.

Setup and configure the operating system on each cluster node

The first step is to configure the operating system on the cluster nodes. For this walk through, use SLES with a valid subscription for the HA add-on.

Install and configure SQL Server on each cluster node

1. Install and setup SQL Server on both nodes. For detailed instructions, see [Install SQL Server on Linux](#).
2. Designate one node as primary and the other as secondary, for purposes of configuration. Use these terms for the following this guide.
3. On the secondary node, stop and disable SQL Server. The following example stops and disables SQL Server:

```
sudo systemctl stop mssql-server
sudo systemctl disable mssql-server
```

NOTE

At setup time, a Server Master Key is generated for the SQL Server instance and placed at `/var/opt/mssql/secrets/machine-key`. On Linux, SQL Server always runs as a local account called mssql. Because it's a local account, its identity isn't shared across nodes. Therefore, you need to copy the encryption key from primary node to each secondary node so each local mssql account can access it to decrypt the Server Master Key.

4. On the primary node, create a SQL server login for Pacemaker and grant the login permission to run `sp_server_diagnostics`. Pacemaker uses this account to verify which node is running SQL Server.

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

Connect to the SQL Server master database with the 'sa' account and run the following:

```
USE [master]
GO
CREATE LOGIN [<loginName>] with PASSWORD= N'<loginPassword>'
GRANT VIEW SERVER STATE TO <loginName>
```

5. On the primary node, stop and disable SQL Server.
6. Follow the directions [in the SUSE documentation](#) to configure and update the hosts file for each cluster node. The 'hosts' file must include the IP address and name of every cluster node.

To check the IP address of the current node run:

```
sudo ip addr show
```

Set the computer name on each node. Give each node a unique name that is 15 characters or less. Set the computer name by adding it to `/etc/hostname` using [yast](#) or [manually](#).

The following example shows `/etc/hosts` with additions for two nodes named `SLES1` and `SLES2`.

```
127.0.0.1 localhost
10.128.18.128 SLES1
10.128.16.77 SLES2
```

NOTE

All cluster nodes must be able to access each other via SSH. Tools like `hb_report` or `crm_report` (for troubleshooting) and Hawk's History Explorer require passwordless SSH access between the nodes, otherwise they can only collect data from the current node. In case you use a non-standard SSH port, use the `-X` option ([see man page](#)). For example, if your SSH port is 3479, invoke an `crm_report` with:

```
crm_report -X "-p 3479" [...]
```

For more information, see [the Administration Guide].(https://www.suse.com/documentation/sle-ha-12/singlehtml/book_sleha/book_sleha.html#sec.ha.troubleshooting.misc)

In the next section you will configure shared storage and move your database files to that storage.

Configure shared storage and move database files

There are a variety of solutions for providing shared storage. This walk-through demonstrates configuring shared storage with NFS. We recommend to follow best practices and use Kerberos to secure NFS:

- [Sharing File Systems with NFS](#)

If you do not follow this guidance, anyone who can access your network and spoof the IP address of a SQL node will be able to access your data files. As always, make sure you threat model your system before using it in production.

Another storage option is to use SMB file share:

- [Samba section of SUSE documentation](#)

Configure an NFS server

To configure an NFS server, see the following steps in the SUSE documentation: [Configuring NFS Server](#).

Configure all cluster nodes to connect to the NFS shared storage

Before configuring the client NFS to mount the SQL Server database files path to point to the shared storage location, make sure you save the database files to a temporary location to be able to copy them later on the share:

1. **On the primary node only**, save the database files to a temporary location. The following script, creates a new temporary directory, copies the database files to the new directory, and removes the old database files. As SQL Server runs as local user mssql, you need to make sure that after data transfer to the mounted share, local user has read-write access to the share.

```
su mssql
mkdir /var/opt/mssql/tmp
cp /var/opt/mssql/data/* /var/opt/mssql/tmp
rm /var/opt/mssql/data/*
exit
```

Configure the NFS client on all cluster nodes:

- [Configuring Clients](#)

NOTE

It is recommended to follow SUSE's best practices and recommendations regarding Highly Available NFS storage: [Highly Available NFS Storage with DRBD and Pacemaker](#).

2. Validate that SQL Server starts successfully with the new file path. Do this on each node. At this point only one node should run SQL Server at a time. They cannot both run at the same time because they will both try to access the data files simultaneously (to avoid accidentally starting SQL Server on both nodes, use a File System cluster resource to make sure the share is not mounted twice by the different nodes). The following commands start SQL Server, check the status, and then stop SQL Server.

```
sudo systemctl start mssql-server
sudo systemctl status mssql-server
sudo systemctl stop mssql-server
```

At this point both instances of SQL Server are configured to run with the database files on the shared storage. The next step is to configure SQL Server for Pacemaker.

Install and configure Pacemaker on each cluster node

1. **On both cluster nodes, create a file to store the SQL Server username and password for the Pacemaker login.** The following command creates and populates this file:

```
sudo touch /var/opt/mssql/secrets/passwd
sudo echo '<loginName>' >> /var/opt/mssql/secrets/passwd
sudo echo '<loginPassword>' >> /var/opt/mssql/secrets/passwd
sudo chown root:root /var/opt/mssql/secrets/passwd
sudo chmod 600 /var/opt/mssql/secrets/passwd
```

2. **All cluster nodes must be able to access each other via SSH.** Tools like hb_report or crm_report (for troubleshooting) and Hawk's History Explorer require passwordless SSH access between the nodes, otherwise they can only collect data from the current node. In case you use a non-standard SSH port, use the -X option (see man page). For example, if your SSH port is 3479, invoke an hb_report with:

```
crm_report -X "-p 3479" [...]
```

For more information, see [System Requirements and Recommendations in the SUSE documentation](#).

3. **Install the High Availability extension.** To install the extension, follow the steps in the following SUSE topic:

[Installation and Setup Quick Start](#)

4. **Install the FCI resource agent for SQL Server.** Run the following commands on both nodes:

```
sudo zypper addrepo -fc https://packages.microsoft.com/config/sles/12/mssql-server-2017.repo
sudo zypper --gpg-auto-import-keys refresh
sudo zypper install mssql-server-ha
```

5. **Automatically set up the first node.** The next step is to setup a running one-node cluster by configuring the first node, SLES1. Follow the instructions in the SUSE topic, [Setting Up the First Node](#).

When finished, check the cluster status with `crm status`:

```
crm status
```

It should show that one node, SLES1, is configured.

6. **Add nodes to an existing cluster.** Next join the SLES2 node to the cluster. Follow the instructions in the SUSE topic, [Adding the Second Node](#).

When finished, check the cluster status with **crm status**. If you have successfully added a second node, the output will be similar to the following:

```
2 nodes configured
1 resource configured
Online: [ SLES1 SLES2 ]
Full list of resources:
admin_addr    (ocf::heartbeat:IPAddr2):     Started SLES1
```

NOTE

admin_addr is the virtual IP cluster resource which is configured during initial one-node cluster setup.

7. **Removal procedures.** If you need to remove a node from the cluster, use the **ha-cluster-remove** bootstrap script. For more information, see [Overview of the Bootstrap Scripts](#).

Configure the cluster resources for SQL Server

The following steps explain how to configure the cluster resource for SQL Server. There are two settings that you need to customize.

- **SQL Server Resource Name:** A name for the clustered SQL Server resource.
- **Timeout Value:** The timeout value is the amount of time that the cluster waits while a resource is brought online. For SQL Server, this is the time that you expect SQL Server to take to bring the `master` database online.

Update the values from the following script for your environment. Run on one node to configure and start the clustered service.

```
sudocrm configure
primitive <sqlServerResourceName> ocf:mssql:fci op start timeout=<timeout_in_seconds>
colocation <constraintName> inf: <virtualIPResourceName> <sqlServerResourceName>
show
commit
exit
```

For example, the following script creates a SQL Server clustered resource named mssqlha.

```
sudocrm configure
primitive mssqlha ocf:mssql:fci op start timeout=60s
colocation admin_addr_mssqlha inf: admin_addr mssqlha
show
commit
exit
```

After the configuration is committed, SQL Server will start on the same node as the virtual IP resource.

For more information, see [Configuring and Managing Cluster Resources \(Command Line\)](#).

Verify that SQL Server is started.

To verify that SQL Server is started, run the **crm status** command:

```
crm status
```

The following examples shows the results when Pacemaker has successfully started as clustered resource.

```
2 nodes configured
2 resources configured

Online: [ SLES1 SLES2 ]

Full list of resources:

admin_addr    (ocf::heartbeat:IPAddr2):      Started SLES1
mssqlha      (ocf::mssql:fci):           Started SLES1
```

Managing cluster resources

To manage your cluster resources, see the following SUSE topic: [Managing Cluster Resources](#)

Manual failover

Although resources are configured to automatically fail over (or migrate) to other nodes of the cluster in the event of a hardware or software failure, you can also manually move a resource to another node in the cluster using either the Pacemaker GUI or the command line.

Use the **migrate** command for this task. For example, to migrate the SQL resource to a cluster node names SLES2 execute:

```
crm resource
migrate mssqlha SLES2
```

Additional resources

Always On Availability Groups on Linux

3/13/2018 • 13 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article describes the characteristics of Always On Availability Groups (AGs) under Linux-based SQL Server installations. It also covers differences between Linux- and Windows Server failover cluster (WSFC)-based AGs. See the [Windows-based documentation](#) for the basics of AGs, as they work the same on Windows and Linux except for the WSFC.

From a high-level standpoint, availability groups under SQL Server on Linux are the same as they are on WSFC-based implementations. That means that all the limitations and features are the same, with some exceptions. The main differences include:

- Microsoft Distributed Transaction Coordinator (DTC) is not supported under Linux in SQL Server 2017. If your applications require the use of distributed transactions and need an AG, deploy SQL Server on Windows.
- Linux-based deployments use Pacemaker instead of a WSFC.
- Unlike most configurations for AGs on Windows except for the Workgroup Cluster scenario, Pacemaker never requires Active Directory Domain Services (AD DS).
- How to fail an AG from one node to another is different between Linux and Windows.
- Certain settings such as `required_synchronized_secondaries_to_commit` can only be changed via Pacemaker on Linux, whereas a WSFC-based install uses Transact-SQL.

Number of replicas and cluster nodes

An AG in SQL Server Standard can have two total replicas: one primary, and one secondary that can only be used for availability purposes. It cannot be used for anything else, such as readable queries. An AG in SQL Server Enterprise can have up to nine total replicas: one primary and up to eight secondaries, of which up to three (including the primary) can be synchronous. If using an underlying cluster, there can be a maximum of 16 nodes total when Corosync is involved. An availability group can span at most nine of the 16 nodes with SQL Server Enterprise, and two with SQL Server Standard.

A two-replica configuration that requires the ability to automatically fail over to another replica requires the use of a configuration-only replica, as described in [Configuration-only replica and quorum](#). Configuration-only replicas were introduced in SQL Server 2017 Cumulative Update 1 (CU1), so that should be the minimum version deployed for this configuration.

If Pacemaker is used, it must be properly configured so it remains up and running. That means that quorum and STONITH must be implemented properly from a Pacemaker perspective, in addition to any SQL Server requirements such as a configuration-only replica.

Readable secondary replicas are only supported with SQL Server Enterprise.

Cluster type and failover mode

New to SQL Server 2017 is the introduction of a cluster type for AGs. For Linux, there are two valid values: External and None. A cluster type of External means that Pacemaker will be used underneath the AG. Using External for cluster type requires that the failover mode be set to External as well (also new in SQL Server 2017). Automatic failover is supported, but unlike a WSFC, failover mode is set to External, not automatic, when Pacemaker is used. Unlike a WSFC, the Pacemaker portion of the AG is created after the AG is configured.

A cluster type of None means that there is no requirement for, nor will the AG use, Pacemaker. Even on servers that have Pacemaker configured, if an AG is configured with a cluster type of None, Pacemaker will not see or manage that AG. A cluster type of None only supports manual failover from a primary to a secondary replica. An AG created with None is primarily targeted for the read-scale out scenario as well as upgrades. While it can work in scenarios like disaster recovery or local availability where no automatic failover is necessary, it is not recommended. The listener story is also more complex without Pacemaker.

Cluster type is stored in the SQL Server dynamic management view (DMV) `sys.availability_groups`, in the columns `cluster_type` and `cluster_type_desc`.

required_synchronized_secondaries_to_commit

New to SQL Server 2017 is a setting that is used by AGs called `required_synchronized_secondaries_to_commit`. This tells the AG the number of secondary replicas that must be in lockstep with the primary. This enables things like automatic failover (only when integrated with Pacemaker with a cluster type of External), and controls the behavior of things like the availability of the primary if the right number of secondary replicas is either online or offline. To understand more about how this works, see [High availability and data protection for availability group configurations](#). The `required_synchronized_secondaries_to_commit` value is set by default and maintained by Pacemaker/ SQL Server. You can manually override this value.

The combination of `required_synchronized_secondaries_to_commit` and the new sequence number (which is stored in `sys.availability_groups`) informs Pacemaker and SQL Server that, for example, automatic failover can happen. In that case, a secondary replica would have the same sequence number as the primary, meaning it is up to date with all the latest configuration information.

There are three values that can be set for `required_synchronized_secondaries_to_commit`: 0, 1, or 2. They control the behavior of what happens when a replica becomes unavailable. The numbers correspond to the number of secondary replicas that must be synchronized with the primary. The behavior is as follows under Linux:

- 0 – No automatic failover is possible since no secondary replica is required to be synchronized. The primary database is available at all times.
- 1 – One secondary replica must be in a synchronized state with the primary; automatic failover is possible. The primary database is unavailable until a secondary synchronous replica is available.
- 2 – Both secondary replicas in a three or more node AG configuration must be synchronized with the primary; automatic failover is possible.

`required_synchronized_secondaries_to_commit` controls not only the behavior of failovers with synchronous replicas, but data loss. With a value of 1 or 2, a secondary replica is always required to be synchronized, so there will always be data redundancy. That means no data loss.

To change the value of `required_synchronized_secondaries_to_commit`, use the following syntax:

NOTE

Changing the value causes the resource to restart, meaning a brief outage. The only way to avoid this is to set the resource to not be managed by the cluster temporarily.

Red Hat Enterprise Linux (RHEL) and Ubuntu

```
sudo pcs resource update <AGResourceName> required_synchronized_secondaries_to_commit=<Value>
```

SUSE Linux Enterprise Server (SLES)

```
sudo crm resource param ms-<AGResourceName> set required_synchronized_secondaries_to_commit <value>
```

where *AGResourceName* is the name of the resource configured for the AG, and *Value* is 0, 1, or 2. To set it back to the default of Pacemaker managing the parameter, execute the same statement with no value.

Automatic failover of an AG is possible when the following conditions are met:

- The primary and the secondary replica are set to synchronous data movement.
- The secondary has a state of synchronized (not synchronizing), meaning the two are at the same data point.
- The cluster type is set to External. Automatic failover is not possible with a cluster type of None.
- The `sequence_number` of the secondary replica to become the primary has the highest sequence number – in other words, the secondary replica's `sequence_number` matches the one from the original primary replica.

If these conditions are met and the server hosting the primary replica fails, the AG will change ownership to a synchronous replica. The behavior for synchronous replicas (of which there can be three total: one primary and two secondary replicas) can further be controlled by `required_synchronized_secondaries_to_commit`. This works with AGs on both Windows and Linux, but is configured completely differently. On Linux, the value is configured automatically by the cluster on the AG resource itself.

Configuration-only replica and quorum

Also new in SQL Server 2017 as of CU1 is a configuration-only replica. Because Pacemaker is different than a WSFC, especially when it comes to quorum and requiring STONITH, having just a two-node configuration will not work when it comes to an AG. For an FCI, the quorum mechanisms provided by Pacemaker can be fine, because all FCI failover arbitration happens at the cluster layer. For an AG, arbitration under Linux happens in SQL Server, where all the metadata is stored. This is where the configuration-only replica comes into play.

Without anything else, a third node and at least one synchronized replica would be required. This would not work for SQL Server Standard, since it can only have two replicas participating in an AG. The configuration-only replica stores the AG configuration in the master database, same as the other replicas in the AG configuration. The configuration-only replica does not have the user databases participating in the AG. The configuration data is sent synchronously from the primary. This configuration data is then used during failovers, whether they are automatic or manual.

For an AG to maintain quorum and enable automatic failovers with a cluster type of External, it either must:

- Have three synchronous replicas (SQL Server Enterprise only); or
- Have two replicas (primary and secondary) as well as a configuration only replica.

Manual failovers can happen whether using External or None cluster types for AG configurations. While a configuration-only replica can be configured with an AG that has a cluster type of None, it is not recommended, since it complicates the deployment. For those configurations, manually modify

`required_synchronized_secondaries_to_commit` to have a value of at least 1, so that there is at least one synchronized replica.

A configuration-only replica can be hosted on any edition of SQL Server, including SQL Server Express. This will minimize licensing costs and ensures it works with AGs in SQL Server Standard. This means that the third required server just needs to meet the minimum specification for SQL Server, since it is not receiving user transaction traffic for the AG.

When a configuration-only replica is used, it has the following behavior:

- By default, `required_synchronized_secondaries_to_commit` is set to 0. This can be manually modified to 1 if desired.
- If the primary fails and `required_synchronized_secondaries_to_commit` is 0, the secondary replica will become the

new primary and be available for both reading and writing. If the value is 1, automatic failover will occur, but will not accept new transactions until the other replica is online.

- If a secondary replica fails and `required_synchronized_secondaries_to_commit` is 0, the primary replica still accepts transactions, but if the primary fails at this point, there is no protection for the data nor failover possible (manual or automatic), since a secondary replica is not available.
- If the configuration-only replicas fails, the AG will function normally, but no automatic failover is possible.
- If both a synchronous secondary replica and the configuration-only replica fail, the primary cannot accept transactions, and there is nowhere for the primary to fail to.

In CU1 there is a known bug in the logging in the corosync.log file that is generated via `mssql-server-ha`. If a secondary replica is not able to become the primary due to the number of required replicas available, the current message says "Expected to receive 1 sequence numbers but only received 2. Not enough replicas are online to safely promote the local replica." The numbers should be reversed, and it should say "Expected to receive 2 sequence numbers but only received 1. Not enough replicas are online to safely promote the local replica."

Multiple availability groups

More than one AG can be created per Pacemaker cluster or set of servers. The only limitation is system resources. AG ownership is shown by the master. Different AGs can be owned by different nodes; they do not all need to be running on the same node.

Drive and folder location for databases

As on Windows-based AGs, the drive and folder structure for the user databases participating in an AG should be identical. For example, if the user databases are in `/var/opt/mssql/userdata` on Server A, that same folder should exist on Server B. The only exception to this is noted in the section [Interoperability with Windows-based availability groups and replicas](#).

The listener under Linux

The listener is optional functionality for an AG. It provides a single point of entry for all connections (read/write to the primary replica and/or read-only to secondary replicas) so that applications and end users do not need to know which server is hosting the data. In a WSFC, this is the combination of a network name resource and an IP resource, which is then registered in AD DS (if needed) as well as DNS. In combination with the AG resource itself, it provides that abstraction. For more information on a listener, see [Listeners, Client Connectivity, and Application Failover](#).

The listener under Linux is configured differently, but its functionality is the same. There is no concept of a network name resource in Pacemaker, nor is an object created in AD DS; there is just an IP address resource created in Pacemaker that can run on any of the nodes. An entry associated with the IP resource for the listener in DNS with a "friendly name" needs to be created. The IP resource for the listener will only be active on the server hosting the primary replica for that availability group.

If Pacemaker is used and an IP address resource is created that is associated with the listener, there will be a brief outage as the IP address stops on the one server and starts on the other, whether it is automatic or manual failover. While this provides abstraction through the combination of a single name and IP address, it does not mask the outage. An application must be able to handle the disconnect by having some sort of functionality to detect this and reconnect.

However, the combination of the DNS name and IP address is still not enough to provide all the functionality that a listener on a WSFC provides, such as read-only routing for secondary replicas. When configuring an AG, a "listener" still needs to be configured in SQL Server. This can be seen in the wizard as well as the Transact-SQL syntax. There are two ways that this can be configured to function the same as on Windows:

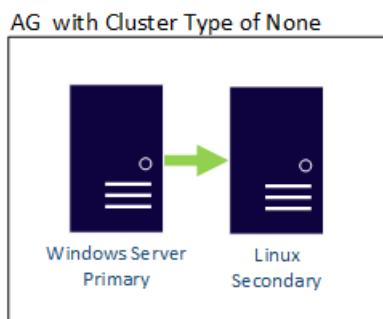
- For an AG with a cluster type of External, the IP address associated with the “listener” created in SQL Server should be the IP address of the resource created in Pacemaker.
- For an AG created with a cluster type of None, use the IP address associated with the primary replica.

The instance associated with the provided IP address then becomes the coordinator for things like the read-only routing requests from applications.

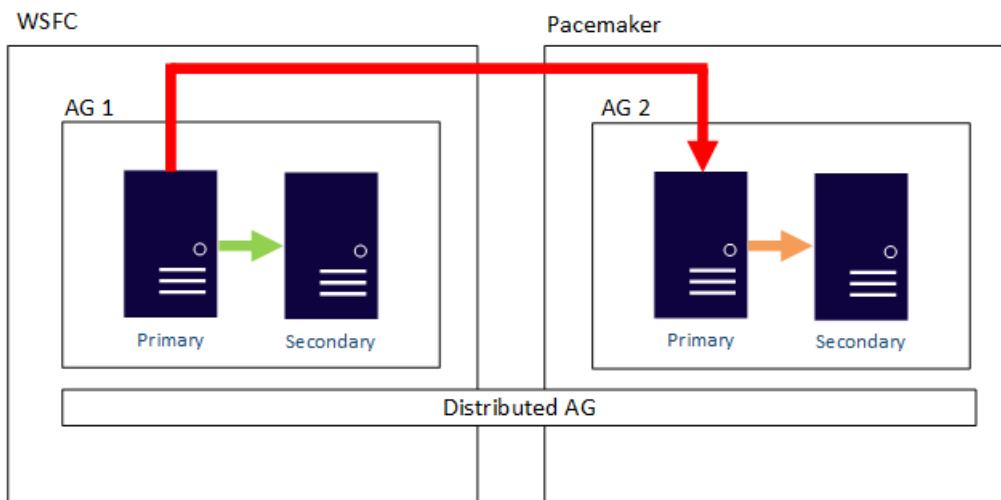
Interoperability with Windows-based availability groups and replicas

An AG that has a cluster type of External or one that is WSFC cannot have its replicas cross platforms. This is true whether the AG is SQL Server Standard or SQL Server Enterprise. That means in a traditional AG configuration with an underlying cluster, one replica cannot be on a WSFC and the other on Linux with Pacemaker.

An AG with a cluster type of NONE can have its replicas cross OS boundaries, so there could be both Linux- and Windows-based replicas in the same AG. An example is shown here where the primary replica is Windows-based, while the secondary is on one of the Linux distributions.



A distributed AG can also cross OS boundaries. The underlying AGs are bound by the rules for how they are configured, such as one configured with External being Linux-only, but the AG that it is joined to could be configured using a WSFC. Consider the following example:



Next steps

[Configure availability group for SQL Server on Linux](#)

[Configure read-scale availability group for SQL Server on Linux](#)

[Add availability group Cluster Resource on RHEL](#)

[Add availability group Cluster Resource on SLES](#)

[Add availability group Cluster Resource on Ubuntu](#)

Configure a cross-platform availability group

High availability and data protection for availability group configurations

2/14/2018 • 9 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Linux only) ✖ Azure SQL Database ✖ Azure SQL Data Warehouse ✖ Parallel Data Warehouse

This article presents supported deployment configurations for SQL Server Always On availability groups on Linux servers. An availability group supports high availability and data protection. Automatic failure detection, automatic failover, and transparent reconnection after failover provide high availability. Synchronized replicas provide data protection.

On a Windows Server Failover Cluster (WSFC), a common configuration for high availability uses two synchronous replicas and a third server or file share to provide quorum. The file-share witness validates the availability group configuration - status of synchronization, and the role of the replica, for example. This configuration ensures that the secondary replica chosen as the failover target has the latest data and availability group configuration changes.

The WSFC synchronizes configuration metadata for failover arbitration between the availability group replicas and the file-share witness. When an availability group is not on a WSFC, the SQL Server instances store configuration metadata in the master database.

For example, an availability group on a Linux cluster has `CLUSTER_TYPE = EXTERNAL`. There is no WSFC to arbitrate failover. In this case the configuration metadata is managed and maintained by the SQL Server instances. Because there is no witness server in this cluster, a third SQL Server instance is required to store configuration state metadata. All three SQL Server instances together provide distributed metadata storage for the cluster.

The cluster manager can query the instances of SQL Server in the availability group, and orchestrate failover to maintain high availability. In a Linux cluster, Pacemaker is the cluster manager.

SQL Server 2017 CU 1 enables high availability for an availability group with `CLUSTER_TYPE = EXTERNAL` for two synchronous replicas plus a configuration only replica. The configuration only replica can be hosted on any edition of SQL Server 2017 CU1 or later - including SQL Server Express edition. The configuration only replica maintains configuration information about the availability group in the master database but does not contain the user databases in the availability group.

How the configuration affects default resource settings

SQL Server 2017 introduces the `REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT` cluster resource setting. This setting guarantees the specified number of secondary replicas write the transaction data to log before the primary replica commits each transaction. When you use an external cluster manager, this setting affects both high availability and data protection. The default value for the setting depends on the architecture at the time the cluster resource is created. When you install the SQL Server resource agent - `mssql-server-ha` - and create a cluster resource for the availability group, the cluster manager detects the availability group configuration and sets `REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT` accordingly.

If supported by the configuration, the resource agent parameter `REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT` is set to the value that provides high availability and data protection. For more information, see [Understand SQL Server resource agent for pacemaker](#).

The following sections explain the default behavior for the cluster resource.

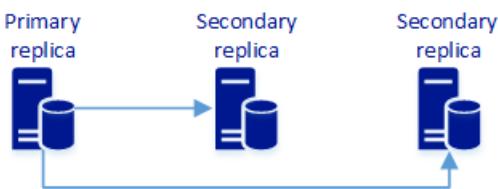
Choose an availability group design to meet specific business requirements for high availability, data protection, and read-scale.

The following configurations describe the availability group design patterns and the capabilities of each pattern. These design patterns apply to availability groups with `CLUSTER_TYPE = EXTERNAL` for high availability solutions.

- **Three synchronous replicas**
- **Two synchronous replicas**
- **Two synchronous replicas and a configuration only replica**

Three synchronous replicas

This configuration consists of three synchronous replicas. By default, it provides high availability and data protection. It can also provide read-scale.



An availability group with three synchronous replicas can provide read-scale, high availability, and data protection. The following table describes availability behavior.

	READ-SCALE	HIGH AVAILABILITY & DATA PROTECTION	DATA PROTECTION
<code>REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT=</code>		1*	2
Primary outage	Manual failover. Might have data loss. New primary is R/W.	Automatic failover. New primary is R/W.	Automatic failover. New primary is not available for user transactions until former primary recovers and joins availability group as secondary.
One secondary replica outage	Primary is R/W. No automatic failover if primary fails.	Primary is R/W. No automatic failover if primary fails as well.	Primary is not available for user transactions.

* Default

Two synchronous replicas

This configuration enables data protection. Like the other availability group configurations, it can enable read-scale. The two synchronous replicas configuration does not provide automatic high availability.



An availability group with two synchronous replicas provides read-scale and data protection. The following table describes availability behavior.

	READ SCALE	DATA PROTECTION
REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT=		1
Primary outage	Manual failover. Might have data loss. New primary is R/W.	Automatic failover. New primary is not available for user transactions until former primary recovers and joins availability group as secondary.
One secondary replica outage	Primary is R/W, running exposed to data loss.	Primary is not available for user transactions until secondary recovers.

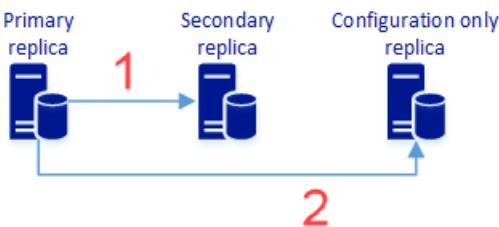
* Default

NOTE

The preceding scenario is the behavior prior to SQL Server 2017 CU 1.

Two synchronous replicas and a configuration only replica

An availability group with two (or more) synchronous replicas and a configuration only replica provides data protection and may also provide high availability. The following diagram represents this architecture:



1. Synchronous replication of user data to the secondary replica. It also includes availability group configuration metadata.
2. Synchronous replication of availability group configuration metadata. It does not include user data.

In the availability group diagram, a primary replica pushes configuration data to both the secondary replica and the configuration only replica. The secondary replica also receives user data. The configuration only replica does not receive user data. The secondary replica is in synchronous availability mode. The configuration only replica does not contain the databases in the availability group - only metadata about the availability group. Configuration data on the configuration only replica is committed synchronously.

NOTE

An availability group with configuration only replica is new for SQL Server 2017 CU1. All instances of SQL Server in the availability group must be SQL Server 2017 CU1 or later.

The default value for REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT is 0. The following table describes availability behavior.

	HIGH AVAILABILITY & DATA PROTECTION	DATA PROTECTION
REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT=		1

	HIGH AVAILABILITY & DATA PROTECTION	DATA PROTECTION
Primary outage	Automatic failover. New primary is R/W.	Automatic failover. New primary is not available for user transactions.
Secondary replica outage	Primary is R/W, running exposed to data loss (if primary fails and cannot be recovered). No automatic failover if primary fails as well.	Primary is not available for user transactions. No replica to fail over to if primary fails as well.
Configuration only replica outage	Primary is R/W. No automatic failover if primary fails as well.	Primary is R/W. No automatic failover if primary fails as well.
Synchronous secondary + configuration only replica outage	Primary is not available for user transactions. No automatic failover.	Primary is not available for user transactions. No replica to failover to if primary fails as well.

* Default

NOTE

The instance of SQL Server that hosts the configuration only replica can also host other databases. It can also participate as a configuration only database for more than one availability group.

Requirements

- All replicas in an availability group with a configuration only replica must be SQL Server 2017 CU 1 or later.
- Any edition of SQL Server can host a configuration only replica, including SQL Server Express.
- The availability group needs at least one secondary replica - in addition to the primary replica.
- Configuration only replicas do not count towards the maximum number of replicas per instance of SQL Server. SQL Server standard edition allows up to three replicas, SQL Server Enterprise Edition allows up to 9.

Considerations

- No more than one configuration only replica per availability group.
- A configuration only replica cannot be a primary replica.
- You cannot modify the availability mode of a configuration only replica. To change from a configuration only replica to a synchronous or asynchronous secondary replica, remove the configuration only replica, and add a secondary replica with the required availability mode.
- A configuration only replica is synchronous with the availability group metadata. There is no user data.
- An availability group with one primary replica and one configuration only replica, but no secondary replica is not valid.
- You cannot create an availability group on an instance of SQL Server Express edition.

Understand SQL Server resource agent for pacemaker

SQL Server 2017 CTP 1.4 added `sequence_number` to `sys.availability_groups` to allow Pacemaker to identify how up-to-date secondary replicas are with the primary replica. `sequence_number` is a monotonically increasing BIGINT that represents how up-to-date the local availability group replica is. Pacemaker updates the `sequence_number` with each availability group configuration change. Examples of configuration changes include failover, replica addition, or removal. The number is updated on the primary, then replicated to secondary replicas. Thus a secondary replica that has up-to-date configuration has the same sequence number as the primary.

When Pacemaker decides to promote a replica to primary, it first sends a *pre-promote* notification to all replicas. The replicas return the sequence number. Next, when Pacemaker actually tries to promote a replica to primary, the replica only promotes itself if its sequence number is the highest of all the sequence numbers. If its own sequence number does not match the highest sequence number, the replica rejects the promote operation. In this way only the replica with the highest sequence number can be promoted to primary, ensuring no data loss.

This process requires at least one replica available for promotion with the same sequence number as the previous primary. The Pacemaker resource agent sets `REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT` such that at least one synchronous secondary replica is up-to-date and available to be the target of an automatic failover by default. With each monitoring action, the value of `REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT` is computed (and updated if necessary). The `REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT` value is 'number of synchronous replicas' divided by 2. At failover time, the resource agent requires (`total number of replicas - REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT` replicas) to respond to the pre-promote notification. The replica with the highest `sequence_number` is promoted to primary.

For example, An availability group with three synchronous replicas - one primary replica and two synchronous secondary replicas.

- `REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT` is 1; $(3 / 2 \rightarrow 1)$.
- The required number of replicas to respond to pre-promote action is 2; $(3 - 1 = 2)$.

In this scenario, two replicas have to respond for the failover to be triggered. For successful automatic failover after a primary replica outage, both secondary replicas need to be up-to-date and respond to the pre-promote notification. If they are online and synchronous, they have the same sequence number. The availability group promotes one of them. If only one of the secondary replicas responds to the pre-promote action, the resource agent cannot guarantee that the secondary that responded has the highest `sequence_number`, and a failover is not triggered.

IMPORTANT

When `REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT` is 0 there is risk of data loss. During a primary replica outage, the resource agent does not automatically trigger a failover. You can either wait for primary to recover, or manually fail over using `FORCE_FAILOVER_ALLOW_DATA_LOSS`.

You can choose to override the default behavior, and prevent the availability group resource from setting `REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT` automatically.

The following script sets `REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT` to 0 on an availability group named `<**ag1**>`. Before you run replace `<**ag1**>` with the name of your availability group.

```
sudo pcs resource update <**ag1**> required_synchronized_secondaries_to_commit=0
```

To revert to default value, based on the availability group configuration run:

```
sudo pcs resource update <**ag1**> required_synchronized_secondaries_to_commit=
```

NOTE

When you run the preceding commands, the primary is temporarily demoted to secondary, then promoted again. The resource update causes all replicas to stop and restart. The new value for `REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT` is only set once replicas are restarted, not instantaneously.

See also

[Availability groups on Linux](#)

Configure SQL Server Always On Availability Group for high availability on Linux

2/14/2018 • 12 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article describes how to create a SQL Server Always On Availability Group (AG) for high availability on Linux. There are two configuration types for AGs. A *high availability* configuration uses a cluster manager to provide business continuity. This configuration can also include read-scale replicas. This document explains how to create the AG for high availability.

You can also create an AG without a cluster manager for *read-scale*. The AG for read scale only provides read-only replicas for performance scale-out. It does not provide high availability. To create an AG for read-scale, see [Configure a SQL Server Availability Group for read-scale on Linux](#).

Configurations that guarantee high availability and data protection require either two or three synchronous commit replicas. With three synchronous replicas, the AG can automatically recover even if one server is not available. For more information, see [High availability and data protection for Availability Group configurations](#).

All servers must be either physical or virtual, and virtual servers must be on the same virtualization platform. This requirement is because the fencing agents are platform specific. See [Policies for Guest Clusters](#).

Roadmap

The steps to create an AG on Linux servers for high availability are different from the steps on a Windows Server failover cluster. The following list describes the high-level steps:

1. [Configure SQL Server on three cluster servers](#).

IMPORTANT

All three servers in the AG need to be on the same platform - physical or virtual - because Linux high availability uses fencing agents to isolate resources on servers. The fencing agents are specific for each platform.

2. Create the AG. This step is covered in this current article.
3. Configure a cluster resource manager, like Pacemaker.

The way to configure a cluster resource manager depends on the specific Linux distribution. See the following links for distribution specific instructions:

- [RHEL](#)
- [SUSE](#)
- [Ubuntu](#)

IMPORTANT

Production environments require a fencing agent, like STONITH for high availability. The demonstrations in this documentation do not use fencing agents. The demonstrations are for testing and validation only.

A Linux cluster uses fencing to return the cluster to a known state. The way to configure fencing depends on the distribution and the environment. Currently, fencing is not available in some cloud environments. For more information, see [Support Policies for RHEL High Availability Clusters - Virtualization Platforms](#).

For SLES, see [SUSE Linux Enterprise High Availability Extension](#).

4. Add the AG as a resource in the cluster.

The way to add the AG as a resource in the cluster depends on the Linux distribution. See the following links for distribution specific instructions:

- [RHEL](#)
- [SLES](#)
- [Ubuntu](#)

Prerequisites

Before you create the availability group, you need to:

- Set your environment so that all the servers that will host availability replicas can communicate.
- Install SQL Server.

NOTE

On Linux, you must create an availability group before you add it as a cluster resource to be managed by the cluster. This document provides an example that creates the availability group. For distribution-specific instructions to create the cluster and add the availability group as a cluster resource, see the links under "Next steps."

1. Update the computer name for each host.

Each SQL Server name must be:

- 15 characters or less.
- Unique within the network.

To set the computer name, edit `/etc/hostname`. The following script lets you edit `/etc/hostname` with `vi`:

```
sudo vi /etc/hostname
```

2. Configure the hosts file.

NOTE

If hostnames are registered with their IP in the DNS server, you don't need to do the following steps. Validate that all the nodes intended to be part of the availability group configuration can communicate with each other. (A ping to the hostname should reply with the corresponding IP address.) Also, make sure that the `/etc/hosts` file doesn't contain a record that maps the localhost IP address 127.0.0.1 with the hostname of the node.

The hosts file on every server contains the IP addresses and names of all servers that will participate in the availability group.

The following command returns the IP address of the current server:

```
sudo ip addr show
```

Update `/etc/hosts`. The following script lets you edit `/etc/hosts` with `vi`:

```
sudo vi /etc/hosts
```

The following example shows `/etc/hosts` on **node1** with additions for **node1**, **node2**, and **node3**. In this document, **node1** refers to the server that hosts the primary replica. And **node2** and **node3** refer to servers that host the secondary replicas.

```
127.0.0.1 localhost localhost4 localhost4.localdomain4
::1 localhost localhost6 localhost6.localdomain6
10.128.18.12 node1
10.128.16.77 node2
10.128.15.33 node3
```

Install SQL Server

Install SQL Server. The following links point to SQL Server installation instructions for various distributions:

- [Red Hat Enterprise Linux](#)
- [SUSE Linux Enterprise Server](#)
- [Ubuntu](#)

Enable AlwaysOn availability groups and restart mssql-server

Enable AlwaysOn availability groups on each node that hosts a SQL Server instance. Then restart `mssql-server`. Run the following script:

```
sudo /opt/mssql/bin/mssql-conf set hadr.hadrenabled 1
sudo systemctl restart mssql-server
```

Enable an AlwaysOn_health event session

You can optionally enable AlwaysOn availability groups extended events to help with root-cause diagnosis when you troubleshoot an availability group. Run the following command on each instance of SQL Server:

```
ALTER EVENT SESSION AlwaysOn_health ON SERVER WITH (STARTUP_STATE=ON);
GO
```

For more information about this XE session, see [AlwaysOn extended events](#).

Create a database mirroring endpoint user

The following Transact-SQL script creates a login named `dbm_login` and a user named `dbm_user`. Update the script with a strong password. To create the database mirroring endpoint user, run the following command on all SQL Server instances:

```
CREATE LOGIN dbm_login WITH PASSWORD = '**<1Sample_Strong_Password!@#>**';
CREATE USER dbm_user FOR LOGIN dbm_login;
```

Create a certificate

The SQL Server service on Linux uses certificates to authenticate communication between the mirroring endpoints.

The following Transact-SQL script creates a master key and a certificate. It then backs up the certificate and secures the file with a private key. Update the script with strong passwords. Connect to the primary SQL Server instance. To create the certificate, run the following Transact-SQL script:

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '**<Master_Key_Password>**';
CREATE CERTIFICATE dbm_certificate WITH SUBJECT = 'dbm';
BACKUP CERTIFICATE dbm_certificate
    TO FILE = '/var/opt/mssql/data/dbm_certificate.cer'
    WITH PRIVATE KEY (
        FILE = '/var/opt/mssql/data/dbm_certificate.pvk',
        ENCRYPTION BY PASSWORD = '**<Private_Key_Password>**'
    );
```

At this point, your primary SQL Server replica has a certificate at `/var/opt/mssql/data/dbm_certificate.cer` and a private key at `/var/opt/mssql/data/dbm_certificate.pvk`. Copy these two files to the same location on all servers that will host availability replicas. Use the mssql user, or give permission to the mssql user to access these files.

For example, on the source server, the following command copies the files to the target machine. Replace the `**<node2>**` values with the names of the SQL Server instances that will host the replicas.

```
cd /var/opt/mssql/data
scp dbm_certificate.* root@**<node2>**:/var/opt/mssql/data/
```

On each target server, give permission to the mssql user to access the certificate.

```
cd /var/opt/mssql/data
chown mssql:mssql dbm_certificate.*
```

Create the certificate on secondary servers

The following Transact-SQL script creates a master key and a certificate from the backup that you created on the primary SQL Server replica. The command also authorizes the user to access the certificate. Update the script with strong passwords. The decryption password is the same password that you used to create the .pvk file in a previous step. To create the certificate, run the following script on all secondary servers:

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '**<Master_Key_Password>**';
CREATE CERTIFICATE dbm_certificate
    AUTHORIZATION dbm_user
    FROM FILE = '/var/opt/mssql/data/dbm_certificate.cer'
    WITH PRIVATE KEY (
        FILE = '/var/opt/mssql/data/dbm_certificate.pvk',
        DECRYPTION BY PASSWORD = '**<Private_Key_Password>**'
    );
```

Create the database mirroring endpoints on all replicas

Database mirroring endpoints use the Transmission Control Protocol (TCP) to send and receive messages between the server instances that participate in database mirroring sessions or host availability replicas. The database mirroring endpoint listens on a unique TCP port number.

The following Transact-SQL script creates a listening endpoint named `Hadr_endpoint` for the availability group. It starts the endpoint and gives connection permission to the user that you created. Before you run the script, replace the values between `**< ... >**`. Optionally you can include an IP address `LISTENER_IP = (0.0.0.0)`. The listener IP address must be an IPv4 address. You can also use `0.0.0.0`.

Update the following Transact-SQL script for your environment on all SQL Server instances:

```
CREATE ENDPOINT [Hadr_endpoint]
AS TCP (LISTENER_PORT = **<5022>**)
FOR DATA_MIRRORING (
    ROLE = ALL,
    AUTHENTICATION = CERTIFICATE dbm_certificate,
    ENCRYPTION = REQUIRED ALGORITHM AES
);
ALTER ENDPOINT [Hadr_endpoint] STATE = STARTED;
GRANT CONNECT ON ENDPOINT::[Hadr_endpoint] TO [dbm_login];
```

NOTE

If you use SQL Server Express Edition on one node to host a configuration-only replica, the only valid value for `ROLE` is `WITNESS`. Run the following script on SQL Server Express Edition:

```
CREATE ENDPOINT [Hadr_endpoint]
AS TCP (LISTENER_PORT = **<5022>**)
FOR DATA_MIRRORING (
    ROLE = WITNESS,
    AUTHENTICATION = CERTIFICATE dbm_certificate,
    ENCRYPTION = REQUIRED ALGORITHM AES
);
ALTER ENDPOINT [Hadr_endpoint] STATE = STARTED;
GRANT CONNECT ON ENDPOINT::[Hadr_endpoint] TO [dbm_login];
```

The TCP port on the firewall must be open for the listener port.

IMPORTANT

For the SQL Server 2017 release, the only authentication method supported for the database mirroring endpoint is `CERTIFICATE`. The `WINDOWS` option will be enabled in a future release.

For more information, see [The database mirroring endpoint \(SQL Server\)](#).

Create the AG

For a high availability configuration that ensures automatic failover, the AG requires at least three replicas. Either of the following configurations can support high availability:

- [Three synchronous replicas](#)
- [Two synchronous replicas plus a configuration replica](#)

For information, see [High availability and data protection for Availability Group configurations](#).

NOTE

The availability groups can include additional synchronous or asynchronous replicas.

Create the AG for high availability on Linux. Use the [CREATE AVAILABILITY GROUP](#) with `CLUSTER_TYPE = EXTERNAL`.

- Availability group - `CLUSTER_TYPE = EXTERNAL` Specifies that an external cluster entity manages the AG. Pacemaker is an example of an external cluster entity. When the AG cluster type is external,
- Set Primary and secondary replicas `FAILOVER_MODE = EXTERNAL`. Specifies that the replica interacts with an external cluster manager, like Pacemaker.

The following Transact-SQL scripts create an AG for high availability named `ag1`. The script configures the AG replicas with `SEEDING_MODE = AUTOMATIC`. This setting causes SQL Server to automatically create the database on each secondary server. Update the following script for your environment. Replace the `<node1>`, `<node2>`, or `<node3>` values with the names of the SQL Server instances that host the replicas. Replace the `<5022>` with the port you set for the data mirroring endpoint. To create the AG, run the following Transact-SQL on the SQL Server instance that hosts the primary replica.

Run **only one** of the following scripts:

- [Create Availability Group with three synchronous replicas](#).
 - [Create Availability Group with two synchronous replicas and a configuration replica](#)
 - [Create Availability Group with two synchronous replicas](#).
- Create AG with three synchronous replicas

```
CREATE AVAILABILITY GROUP [ag1]
    WITH (DB_FAILOVER = ON, CLUSTER_TYPE = EXTERNAL)
    FOR REPLICA ON
        N'<node1>'
        WITH (
            ENDPOINT_URL = N'tcp://<node1>:<5022>',
            AVAILABILITY_MODE = SYNCHRONOUS_COMMIT,
            FAILOVER_MODE = EXTERNAL,
            SEEDING_MODE = AUTOMATIC
        ),
        N'<node2>'
        WITH (
            ENDPOINT_URL = N'tcp://<node2>:<5022>',
            AVAILABILITY_MODE = SYNCHRONOUS_COMMIT,
            FAILOVER_MODE = EXTERNAL,
            SEEDING_MODE = AUTOMATIC
        ),
        N'<node3>'
        WITH(
            ENDPOINT_URL = N'tcp://<node3>:<5022>',
            AVAILABILITY_MODE = SYNCHRONOUS_COMMIT,
            FAILOVER_MODE = EXTERNAL,
            SEEDING_MODE = AUTOMATIC
        );
ALTER AVAILABILITY GROUP [ag1] GRANT CREATE ANY DATABASE;
```

IMPORTANT

After you run the preceding script to create an AG with three synchronous replicas, do not run the following script:

- Create AG with two synchronous replicas and a configuration replica:

IMPORTANT

This architecture allows any edition of SQL Server to host the third replica. For example, the third replica can be hosted on SQL Server Enterprise Edition. On Enterprise Edition, the only valid endpoint type is **WITNESS**.

```
CREATE AVAILABILITY GROUP [ag1]
    WITH (CLUSTER_TYPE = EXTERNAL)
    FOR REPLICA ON
        N'<node1>' WITH (
            ENDPOINT_URL = N'tcp://<node1>:<5022>',
            AVAILABILITY_MODE = SYNCHRONOUS_COMMIT,
            FAILOVER_MODE = EXTERNAL,
            SEEDING_MODE = AUTOMATIC
        ),
        N'<node2>' WITH (
            ENDPOINT_URL = N'tcp://<node2>:<5022>',
            AVAILABILITY_MODE = SYNCHRONOUS_COMMIT,
            FAILOVER_MODE = EXTERNAL,
            SEEDING_MODE = AUTOMATIC
        ),
        N'<node3>' WITH (
            ENDPOINT_URL = N'tcp://<node3>:<5022>',
            AVAILABILITY_MODE = CONFIGURATION_ONLY
        );
ALTER AVAILABILITY GROUP [ag1] GRANT CREATE ANY DATABASE;
```

- Create AG with two synchronous replicas

Include two replicas with synchronous availability mode. For example, the following script creates an AG called **ag1**. **node1** and **node2** host replicas in synchronous mode, with automatic seeding and automatic failover.

IMPORTANT

Only run the following script to create an AG with two synchronous replicas. Do not run the following script if you ran either preceding script.

```
CREATE AVAILABILITY GROUP [ag1]
    WITH (CLUSTER_TYPE = EXTERNAL)
    FOR REPLICA ON
        N'node1' WITH (
            ENDPOINT_URL = N'tcp://node1:5022',
            AVAILABILITY_MODE = SYNCHRONOUS_COMMIT,
            FAILOVER_MODE = EXTERNAL,
            SEEDING_MODE = AUTOMATIC
        ),
        N'node2' WITH (
            ENDPOINT_URL = N'tcp://node2:5022',
            AVAILABILITY_MODE = SYNCHRONOUS_COMMIT,
            FAILOVER_MODE = EXTERNAL,
            SEEDING_MODE = AUTOMATIC
        );
ALTER AVAILABILITY GROUP [ag1] GRANT CREATE ANY DATABASE;
```

You can also configure an AG with **CLUSTER_TYPE=EXTERNAL** using SQL Server Management Studio or PowerShell.

Join secondary replicas to the AG

The following Transact-SQL script joins a SQL Server instance to an AG named `ag1`. Update the script for your environment. On each SQL Server instance that hosts a secondary replica, run the following Transact-SQL to join the AG.

```
ALTER AVAILABILITY GROUP [ag1] JOIN WITH (CLUSTER_TYPE = EXTERNAL);

ALTER AVAILABILITY GROUP [ag1] GRANT CREATE ANY DATABASE;
```

Add a database to the availability group

Ensure that the database you add to the availability group is in full recovery mode and has a valid log backup. If this is a test database or a newly created database, take a database backup. On the primary SQL Server, run the following Transact-SQL script to create and back up a database called `db1`:

```
CREATE DATABASE [db1];
ALTER DATABASE [db1] SET RECOVERY FULL;
BACKUP DATABASE [db1]
    TO DISK = N'/var/opt/mssql/data/db1.bak';
```

On the primary SQL Server replica, run the following Transact-SQL script to add a database called `db1` to an availability group called `ag1`:

```
ALTER AVAILABILITY GROUP [ag1] ADD DATABASE [db1];
```

Verify that the database is created on the secondary servers

On each secondary SQL Server replica, run the following query to see if the `db1` database was created and is synchronized:

```
SELECT * FROM sys.databases WHERE name = 'db1';
GO
SELECT DB_NAME(database_id) AS 'database', synchronization_state_desc FROM
sys.dm_hadr_database_replica_states;
```

IMPORTANT

After you create the AG, you must configure integration with a cluster technology like Pacemaker for high availability. For a read-scale configuration using AGs, starting with SQL Server 2017, setting up a cluster is not required.

If you followed the steps in this document, you have an AG that is not yet clustered. The next step is to add the cluster. This configuration is valid for read-scale/load balancing scenarios, it is not complete for high availability. For high availability, you need to add the AG as a cluster resource. See [Next steps](#) for instructions.

Notes

IMPORTANT

After you configure the cluster and add the AG as a cluster resource, you cannot use Transact-SQL to fail over the AG resources. SQL Server cluster resources on Linux are not coupled as tightly with the operating system as they are on a Windows Server Failover Cluster (WSFC). SQL Server service is not aware of the presence of the cluster. All orchestration is done through the cluster management tools. In RHEL or Ubuntu use `pcs`. In SLES use `crm`.

IMPORTANT

If the AG is a cluster resource, there is a known issue in current release where forced failover with data loss to an asynchronous replica does not work. This will be fixed in the upcoming release. Manual or automatic failover to a synchronous replica succeeds.

Next steps

[Configure Red Hat Enterprise Linux Cluster for SQL Server Availability Group Cluster Resources](#)

[Configure SUSE Linux Enterprise Server Cluster for SQL Server Availability Group Cluster Resources](#)

[Configure Ubuntu Cluster for SQL Server Availability Group Cluster Resources](#)

Configure RHEL Cluster for SQL Server Availability Group

2/14/2018 • 10 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This document explains how to create a three-node availability group cluster for SQL Server on Red Hat Enterprise Linux. For high availability, an availability group on Linux requires three nodes - see [High availability and data protection for availability group configurations](#). The clustering layer is based on Red Hat Enterprise Linux (RHEL) HA add-on built on top of [Pacemaker](#).

NOTE

Access to Red Hat full documentation requires a valid subscription.

For more information on cluster configuration, resource agents options, and management, visit [RHEL reference documentation](#).

NOTE

SQL Server is not as tightly integrated with Pacemaker on Linux as it is with Windows Server failover clustering. A SQL Server instance is not aware of the cluster. Pacemaker provides cluster resource orchestration. Also, the virtual network name is specific to Windows Server failover clustering - there is no equivalent in Pacemaker. Availability group dynamic management views (DMVs) that query cluster information return empty rows on Pacemaker clusters. To create a listener for transparent reconnection after failover, manually register the listener name in DNS with the IP used to create the virtual IP resource.

The following sections walk through the steps to set up a Pacemaker cluster and add an availability group as resource in the cluster for high availability.

Roadmap

The steps to create an availability group on Linux servers for high availability are different from the steps on a Windows Server failover cluster. The following list describes the high-level steps:

1. [Configure SQL Server on the cluster nodes](#).
2. [Create the availability group](#).
3. Configure a cluster resource manager, like Pacemaker. These instructions are in this document.

The way to configure a cluster resource manager depends on the specific Linux distribution.

IMPORTANT

Production environments require a fencing agent, like STONITH for high availability. The demonstrations in this documentation do not use fencing agents. The demonstrations are for testing and validation only.

A Linux cluster uses fencing to return the cluster to a known state. The way to configure fencing depends on the distribution and the environment. Currently, fencing is not available in some cloud environments. For more information, see [Support Policies for RHEL High Availability Clusters - Virtualization Platforms](#).

4. Add the availability group as a resource in the cluster.

Configure high availability for RHEL

To configure high availability for RHEL, enable the high availability subscription and then configure Pacemaker.

Enable the high availability subscription for RHEL

Each node in the cluster must have an appropriate subscription for RHEL and the High Availability Add on. Review the requirements at [How to install High Availability cluster packages in Red Hat Enterprise Linux](#). Follow these steps to configure the subscription and repos:

1. Register the system.

```
sudo subscription-manager register
```

Provide your user name and password.

2. List the available pools for registration.

```
sudo subscription-manager list --available
```

From the list of available pools, note the pool ID for the high availability subscription.

3. Update the following script. Replace <pool id> with the pool ID for high availability from the preceding step. Run the script to attach the subscription.

```
sudo subscription-manager attach --pool=<pool id>
```

4. Enable the repository.

```
sudo subscription-manager repos --enable=rhel-ha-for-rhel-7-server-rpms
```

For more information, see [Pacemaker – The Open Source, High Availability Cluster](#).

After you have configured the subscription, complete the following steps to configure Pacemaker:

Configure Pacemaker

After you register the subscription, complete the following steps to configure Pacemaker:

1. On all cluster nodes, open the Pacemaker firewall ports. To open these ports with `firewalld`, run the following command:

```
sudo firewall-cmd --permanent --add-service=high-availability  
sudo firewall-cmd --reload
```

If the firewall doesn't have a built-in high-availability configuration, open the following ports for Pacemaker.

- TCP: Ports 2224, 3121, 21064
- UDP: Port 5405

2. Install Pacemaker packages on all nodes.

```
sudo yum install pacemaker pcs fence-agents-all resource-agents
```

- Set the password for the default user that is created when installing Pacemaker and Corosync packages.
Use the same password on all nodes.

```
sudo passwd hacluster
```

- To allow nodes to rejoin the cluster after the reboot, enable and start `pcsd` service and Pacemaker. Run the following command on all nodes.

```
sudo systemctl enable pcsd
sudo systemctl start pcsd
sudo systemctl enable pacemaker
```

- Create the Cluster. To create the cluster, run the following command:

```
sudo pcs cluster auth <node1> <node2> <node3> -u hacluster -p <password for hacluster>
sudo pcs cluster setup --name <clusterName> <node1> <node2> <node3>
sudo pcs cluster start --all
```

NOTE

If you previously configured a cluster on the same nodes, you need to use `--force` option when running `pcs cluster setup`. This option is equivalent to running `pcs cluster destroy`. To re-enable pacemaker, run `sudo systemctl enable pacemaker`.

- Install SQL Server resource agent for SQL Server. Run the following commands on all nodes.

```
sudo yum install mssql-server-ha
```

After Pacemaker is configured, use `pcs` to interact with the cluster. Execute all commands on one node from the cluster.

Configure fencing (STONITH)

Pacemaker cluster vendors require STONITH to be enabled and a fencing device configured for a supported cluster setup. STONITH stands for "shoot the other node in the head." When the cluster resource manager cannot determine the state of a node or of a resource on a node, fencing brings the cluster to a known state again.

Resource level fencing ensures that there is no data corruption in case of an outage by configuring a resource. For example, you can use resource level fencing to mark the disk on a node as outdated when the communication link goes down.

Node level fencing ensures that a node does not run any resources. This is done by resetting the node. Pacemaker supports a great variety of fencing devices. Examples include an uninterruptible power supply or management interface cards for servers.

For information about STONITH, and fencing, see the following articles:

- [Pacemaker Clusters from Scratch](#)
- [Fencing and STONITH](#)

- [Red Hat High Availability Add-On with Pacemaker: Fencing](#)

Because the node level fencing configuration depends heavily on your environment, disable it for this tutorial (it can be configured later). The following script disables node level fencing:

```
sudo pcs property set stonith-enabled=false
```

IMPORTANT

Disabling STONITH is just for testing purposes. If you plan to use Pacemaker in a production environment, you should plan a STONITH implementation depending on your environment and keep it enabled. RHEL does not provide fencing agents for any cloud environments (including Azure) or Hyper-V. Consequentially, the cluster vendor does not offer support for running production clusters in these environments. We are working on a solution for this gap that will be available in future releases.

Set cluster property start-failure-is-fatal to false

`start-failure-is-fatal` indicates whether a failure to start a resource on a node prevents further start attempts on that node. When set to `false`, the cluster decides whether to try starting on the same node again based on the resource's current failure count and migration threshold. After failover occurs, Pacemaker retries starting the availability group resource on the former primary once the SQL instance is available. Pacemaker demotes the replica to secondary and it automatically rejoins the availability group.

To update the property value to `false` run:

```
sudo pcs property set start-failure-is-fatal=false
```

WARNING

After an automatic failover, when `start-failure-is-fatal = true` the resource manager will attempt to start the resource. If it fails on the first attempt, manually run `pcs resource cleanup <resourceName>` to clean up the resource failure count and reset the configuration.

For information on Pacemaker cluster properties, see [Pacemaker Clusters Properties](#).

Create a SQL Server login for Pacemaker

1. **On all SQL Servers, create a Server login for Pacemaker.** The following Transact-SQL creates a login:

```
USE [master]
GO
CREATE LOGIN [pacemakerLogin] with PASSWORD= N'ComplexP@$$w0rd!'
ALTER SERVER ROLE [sysadmin] ADD MEMBER [pacemakerLogin]
```

Alternatively, you can set the permissions at a more granular level. The Pacemaker login requires ALTER, CONTROL, and VIEW DEFINITION PERMISSION for managing the availability group as well as VIEW SERVER STATE for the login to be able to run `sp_server_diagnostics`. For more information, see [GRANT Availability Group Permissions \(Transact-SQL\)](#) and [sp_server_diagnostic permissions](#).

The following Transact-SQL grants only the required permission to the Pacemaker login. In the statement below 'ag1' is the name of the availability group that will be added as a cluster resource.

```
GRANT ALTER, CONTROL, VIEW DEFINITION ON AVAILABILITY GROUP::ag1 TO pacemakerLogin  
GRANT VIEW SERVER STATE TO pacemakerLogin
```

2. On all SQL Servers, save the credentials for the SQL Server login.

```
echo 'pacemakerLogin' >> ~/pacemaker-passwd  
echo 'ComplexP@$$w0rd!' >> ~/pacemaker-passwd  
sudo mv ~/pacemaker-passwd /var/opt/mssql/secrets/passwd  
sudo chown root:root /var/opt/mssql/secrets/passwd  
sudo chmod 400 /var/opt/mssql/secrets/passwd # Only readable by root
```

Create availability group resource

To create the availability group resource, use `pcs resource create` command and set the resource properties. The following command creates a `ocf:mssql:ag` master/slave type resource for availability group with name `ag1`.

```
sudo pcs resource create ag_cluster ocf:mssql:ag ag_name=ag1 master notify=true
```

NOTE

When you create the resource, and periodically afterwards, the Pacemaker resource agent automatically sets the value of `REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT` on the availability group based on the availability group's configuration. For example, if the availability group has three synchronous replicas, the agent will set `REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT` to `1`. For details and additional configuration options, see [High availability and data protection for availability group configurations](#).

Create virtual IP resource

To create the virtual IP address resource, run the following command on one node. Use an available static IP address from the network. Replace the IP address between `<10.128.16.240>` with a valid IP address.

```
sudo pcs resource create virtualip ocf:heartbeat:IPAddr2 ip=<10.128.16.240>
```

There is no virtual server name equivalent in Pacemaker. To use a connection string that points to a string server name instead of an IP address, register the virtual IP resource address and desired virtual server name in DNS. For DR configurations, register the desired virtual server name and IP address with the DNS servers on both primary and DR site.

Add colocation constraint

Almost every decision in a Pacemaker cluster, like choosing where a resource should run, is done by comparing scores. Scores are calculated per resource. The cluster resource manager chooses the node with the highest score for a particular resource. If a node has a negative score for a resource, the resource cannot run on that node.

On a pacemaker cluster, you can manipulate the decisions of the cluster with constraints. Constraints have a score. If a constraint has a score lower than `INFINITY`, Pacemaker regards it as recommendation. A score of `INFINITY` is mandatory.

To ensure that primary replica and the virtual ip resources run on the same host, define a colocation constraint with a score of INFINITY. To add the colocation constraint, run the following command on one node.

```
sudo pcs constraint colocation add virtualip ag_cluster-master INFINITY with-rsc-role=Master
```

Add ordering constraint

The colocation constraint has an implicit ordering constraint. It moves the virtual IP resource before it moves the availability group resource. By default the sequence of events is:

1. User issues `pcs resource move` to the availability group primary from node1 to node2.
2. The virtual IP resource stops on node 1.
3. The virtual IP resource starts on node 2.

NOTE

At this point, the IP address temporarily points to node 2 while node 2 is still a pre-failover secondary.

4. The availability group primary on node 1 is demoted to secondary.
5. The availability group secondary on node 2 is promoted to primary.

To prevent the IP address from temporarily pointing to the node with the pre-failover secondary, add an ordering constraint.

To add an ordering constraint, run the following command on one node:

```
sudo pcs constraint order promote ag_cluster-master then start virtualip
```

IMPORTANT

After you configure the cluster and add the availability group as a cluster resource, you cannot use Transact-SQL to fail over the availability group resources. SQL Server cluster resources on Linux are not coupled as tightly with the operating system as they are on a Windows Server Failover Cluster (WSFC). SQL Server service is not aware of the presence of the cluster. All orchestration is done through the cluster management tools. In RHEL or Ubuntu use `pcs` and in SLES use `crm` tools.

Manually fail over the availability group with `pcs`. Do not initiate failover with Transact-SQL. For instructions, see [Failover](#).

Next steps

[Operate HA availability group](#)

Configure SLES Cluster for SQL Server Availability Group

2/14/2018 • 13 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This guide provides instructions to create a three-node cluster for SQL Server on SUSE Linux Enterprise Server (SLES) 12 SP2. For high availability, an availability group on Linux requires three nodes - see [High availability and data protection for availability group configurations](#). The clustering layer is based on SUSE [High Availability Extension \(HAE\)](#) built on top of [Pacemaker](#).

For more information on cluster configuration, resource agent options, management, best practices, and recommendations, see [SUSE Linux Enterprise High Availability Extension 12 SP2](#).

NOTE

At this point, SQL Server's integration with Pacemaker on Linux is not as coupled as with WSFC on Windows. SQL Server service on Linux is not cluster aware. Pacemaker controls all of the orchestration of the cluster resources, including the availability group resource. On Linux, you should not rely on Always On Availability Group Dynamic Management Views (DMVs) that provide cluster information like sys.dm_hadr_cluster. Also, virtual network name is specific to WSFC, there is no equivalent of the same in Pacemaker. You can still create a listener to use it for transparent reconnection after failover, but you will have to manually register the listener name in the DNS server with the IP used to create the virtual IP resource (as explained in the following sections).

Roadmap

The procedure for creating an availability group for high availability differs between Linux servers and a Windows Server failover cluster. The following list describes the high-level steps:

1. [Configure SQL Server on the cluster nodes](#).
2. [Create the availability group](#).
3. Configure a cluster resource manager, like Pacemaker. These instructions are in this document.

The way to configure a cluster resource manager depends on the specific Linux distribution.

IMPORTANT

Production environments require a fencing agent, like STONITH for high availability. The examples in this article do not use fencing agents. They are for testing and validation only.

A Pacemaker cluster uses fencing to return the cluster to a known state. The way to configure fencing depends on the distribution and the environment. At this time, fencing is not available in some cloud environments. See [SUSE Linux Enterprise High Availability Extension](#).

4. [Add the availability group as a resource in the cluster](#).

Prerequisites

To complete the following end-to-end scenario, you need three machines to deploy the three nodes cluster. The

following steps outline how to configure these servers.

Setup and configure the operating system on each cluster node

The first step is to configure the operating system on the cluster nodes. For this walk through, use SLES 12 SP2 with a valid subscription for the HA add-on.

Install and configure SQL Server service on each cluster node

1. Install and setup SQL Server service on all nodes. For detailed instructions, see [Install SQL Server on Linux](#).
2. Designate one node as primary and other nodes as secondaries. Use these terms throughout this guide.
3. Make sure nodes that are going to be part of the cluster can communicate to each other.

The following example shows `/etc/hosts` with additions for three nodes named SLES1, SLES2, and SLES3.

```
127.0.0.1 localhost
10.128.16.33 SLES1
10.128.16.77 SLES2
10.128.16.22 SLES3
```

All cluster nodes must be able to access each other via SSH. Tools like `hb_report` or `crm_report` (for troubleshooting) and Hawk's History Explorer require passwordless SSH access between the nodes, otherwise they can only collect data from the current node. In case you use a non-standard SSH port, use the `-X` option (see [man](#) page). For example, if your SSH port is 3479, invoke a `crm_report` with:

```
sudo crm_report -X "-p 3479" [...]
```

For more information, see the [SLES Administration Guide - Miscellaneous section](#).

Create a SQL Server login for Pacemaker

1. **On all SQL Servers, create a Server login for Pacemaker.** The following Transact-SQL creates a login:

```
USE [master]
GO
CREATE LOGIN [pacemakerLogin] WITH PASSWORD= N'ComplexP@$$w0rd!'
ALTER SERVER ROLE [sysadmin] ADD MEMBER [pacemakerLogin]
```

Alternatively, you can set the permissions at a more granular level. The Pacemaker login requires ALTER, CONTROL, and VIEW DEFINITION PERMISSION for managing the availability group as well as VIEW SERVER STATE for the login to be able to run `sp_server_diagnostics`. For more information, see [GRANT Availability Group Permissions \(Transact-SQL\)](#) and [sp_server_diagnostic permissions](#).

The following Transact-SQL grants only the required permission to the Pacemaker login. In the statement below 'ag1' is the name of the availability group that will be added as a cluster resource.

```
GRANT ALTER, CONTROL, VIEW DEFINITION ON AVAILABILITY GROUP::ag1 TO pacemakerLogin
GRANT VIEW SERVER STATE TO pacemakerLogin
```

2. **On all SQL Servers, save the credentials for the SQL Server login.**

```
echo 'pacemakerLogin' >> ~/pacemaker-passwd
echo 'ComplexP@$$w0rd!' >> ~/pacemaker-passwd
sudo mv ~/pacemaker-passwd /var/opt/mssql/secrets/passwd
sudo chown root:root /var/opt/mssql/secrets/passwd
sudo chmod 400 /var/opt/mssql/secrets/passwd # Only readable by root
```

Configure an Always On Availability Group

On Linux servers, configure the availability group and then configure the cluster resources. To configure the availability group, see [Configure Always On Availability Group for SQL Server on Linux](#)

Install and configure Pacemaker on each cluster node

1. Install the High Availability extension

For reference, see [Installing SUSE Linux Enterprise Server and High Availability Extension](#)

2. Install SQL Server resource agent package on both nodes.

```
sudo zypper install mssql-server-ha
```

Set up the first node

Refer to [SLES installation instructions](#)

1. Log in as `root` to the physical or virtual machine you want to use as cluster node.
2. Start the bootstrap script by executing:

```
sudo ha-cluster-init
```

If NTP has not been configured to start at boot time, a message appears.

If you decide to continue anyway, the script automatically generates keys for SSH access and for the Csync2 synchronization tool and start the services needed for both.

3. To configure the cluster communication layer (Corosync):
 - a. Enter a network address to bind to. By default, the script proposes the network address of eth0. Alternatively, enter a different network address, for example the address of bond0.
 - b. Enter a multicast address. The script proposes a random address that you can use as default.
 - c. Enter a multicast port. The script proposes 5405 as default.
 - d. To configure `SBD ()`, enter a persistent path to the partition of your block device that you want to use for SBD. The path must be consistent across all nodes in the cluster. Finally, the script will start the Pacemaker service to bring the one-node cluster online and enable the Web management interface Hawk2. The URL to use for Hawk2 is displayed on the screen.
4. For any details of the setup process, check `/var/log/sleha-bootstrap.log`. You now have a running one-node cluster. Check the cluster status with `crm status`:

```
sudo crm status
```

You can also see cluster configuration with `crm configure show xml` or `crm configure show`.

5. The bootstrap procedure creates a Linux user named hacluster with the password linux. Replace the default password with a secure one as soon as possible:

```
sudo passwd hacluster
```

Add nodes to the existing cluster

If you have a cluster running with one or more nodes, add more cluster nodes with the ha-cluster-join bootstrap script. The script only needs access to an existing cluster node and will complete the basic setup on the current machine automatically. Use the following steps:

If you have configured the existing cluster nodes with the YaST cluster module, make sure the following prerequisites are fulfilled before you run `ha-cluster-join`:

- The root user on the existing nodes has SSH keys in place for passwordless login.
- Csync2 is configured on the existing nodes. For more information, see Configuring Csync2 with YaST.

1. Log in as root to the physical or virtual machine supposed to join the cluster.
2. Start the bootstrap script by executing:

```
sudo ha-cluster-join
```

If NTP has not been configured to start at boot time, a message appears.

3. If you decide to continue anyway, you will be prompted for the IP address of an existing node. Enter the IP address.
4. If you have not already configured a passwordless SSH access between both machines, you will also be prompted for the root password of the existing node.

After logging in to the specified node, the script copies the Corosync configuration, configures SSH and Csync2, and brings the current machine online as new cluster node. Apart from that, it starts the service needed for Hawk. If you have configured shared storage with OCFS2, it also automatically creates the mountpoint directory for the OCFS2 file system.

5. Repeat the previous steps for all machines you want to add to the cluster.
6. For details of the process, check `/var/log/ha-cluster-bootstrap.log`.
7. Check the cluster status with `sudo crm status`. If you have successfully added a second node, the output will be similar to the following:

```
sudo crm status

3 nodes configured
1 resource configured
Online: [ SLES1 SLES2 SLES3]
Full list of resources:
admin_addr    (ocf::heartbeat:IPAddr2):        Started node1
```

NOTE

`admin_addr` is the virtual IP cluster resource which is configured during initial one-node cluster setup.

After adding all nodes, check if you need to adjust the no-quorum-policy in the global cluster options. This is especially important for two-node clusters. For more information, see Section 4.1.2, Option no-quorum-policy.

Set cluster property start-failure-is-fatal to false

`start-failure-is-fatal` indicates whether a failure to start a resource on a node prevents further start attempts on that node. When set to `false`, the cluster decides whether to try starting on the same node again based on the resource's current failure count and migration threshold. So, after failover occurs, Pacemaker retries starting the availability group resource on the former primary once the SQL instance is available. Pacemaker takes care of demoting the replica to secondary and it automatically rejoins the availability group. Also, if `start-failure-is-fatal` is set to `false`, the cluster falls back to the configured failcount limits configured with migration-threshold. Make sure default for migration threshold is updated accordingly.

To update the property value to false run:

```
sudo crm configure property start-failure-is-fatal=false  
sudo crm configure rsc_defaults migration-threshold=5000
```

If the property has the default value of `true`, if first attempt to start the resource fails, user intervention is required after an automatic failover to clean up the resource failure count and reset the configuration using:

`sudo crm resource cleanup <resourceName>` command.

For more information on Pacemaker cluster properties, see [Configuring Cluster Resources](#).

Configure fencing (STONITH)

Pacemaker cluster vendors require STONITH to be enabled and a fencing device configured for a supported cluster setup. When the cluster resource manager cannot determine the state of a node or of a resource on a node, fencing is used to bring the cluster to a known state again.

Resource level fencing ensures mainly that there is no data corruption during an outage by configuring a resource. You can use resource level fencing, for instance, with DRBD (Distributed Replicated Block Device) to mark the disk on a node as outdated when the communication link goes down.

Node level fencing ensures that a node does not run any resources. This is done by resetting the node and the Pacemaker implementation of it is called STONITH (which stands for "shoot the other node in the head"). Pacemaker supports a great variety of fencing devices, such as an uninterruptible power supply or management interface cards for servers.

For more information, see [Pacemaker Clusters from Scratch, Fencing and Stonith](#) and [SUSE HA documentation: Fencing and STONITH](#).

At cluster initialization time, STONITH is disabled if no configuration is detected. It can be enabled later by running following command:

```
sudo crm configure property stonith-enabled=true
```

IMPORTANT

Disabling STONITH is just for testing purposes. If you plan to use Pacemaker in a production environment, you should plan a STONITH implementation depending on your environment and keep it enabled. SUSE does not provide fencing agents for any cloud environments (including Azure) or Hyper-V. Consequentially, the cluster vendor does not offer support for running production clusters in these environments. We are working on a solution for this gap that will be available in future releases.

Configure the cluster resources for SQL Server

Refer to [SLES Administration Guid](#)

Create availability group resource

The following command creates and configures the availability group resource for three replicas of availability group [ag1]. The monitor operations and timeouts have to be specified explicitly in SLES based on the fact that timeouts are highly workload-dependent and need to be carefully adjusted for each deployment. Run the command on one of the nodes in the cluster:

1. Run `crm configure` to open the crm prompt:

```
sudo crm configure
```

2. In the crm prompt, run the following command to configure the resource properties.

```
primitive ag_cluster \
  ocf:mssql:ag \
  params ag_name="ag1" \
  op start timeout=60s \
  op stop timeout=60s \
  op promote timeout=60s \
  op demote timeout=10s \
  op monitor timeout=60s interval=10s \
  op monitor timeout=60s interval=11s role="Master" \
  op monitor timeout=60s interval=12s role="Slave" \
  op notify timeout=60s
  ms ms-ag_cluster ag_cluster \
  meta master-max="1" master-node-max="1" clone-max="3" \
  clone-node-max="1" notify="true" \
  commit
```

NOTE

When you create the resource, and periodically afterwards, the Pacemaker resource agent automatically sets the value of `REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT` on the availability group based on the availability group's configuration. For example, if the availability group has three synchronous replicas, the agent will set `REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT` to `1`. For details and additional configuration options, see [High availability and data protection for availability group configurations](#).

Create virtual IP resource

If you did not create the virtual IP resource when you ran `ha-cluster-init` you can create this resource now. The following command creates a virtual IP resource. Replace `<**0.0.0.0**>` with an available address from your network and `<**24**>` with the number of bits in the CIDR subnet mask. Run on one node.

```
crm configure \
primitive admin_addr \
    ocf:heartbeat:IPAddr2 \
    params ip=<**0.0.0.0**> \
    cidr_netmask=<**24**>
```

Add colocation constraint

Almost every decision in a Pacemaker cluster, like choosing where a resource should run, is done by comparing scores. Scores are calculated per resource, and the cluster resource manager chooses the node with the highest score for a particular resource. (If a node has a negative score for a resource, the resource cannot run on that node.) We can manipulate the decisions of the cluster with constraints. Constraints have a score. If a constraint has a score lower than INFINITY, it is only a recommendation. A score of INFINITY means it is a must. We want to ensure that primary of the availability group and the virtual ip resource are run on the same host, so we define a colocation constraint with a score of INFINITY.

To set colocation constraint for the virtual IP to run on same node as the master, run the following command on one node:

```
crm configure
colocation vip_on_master inf: \
    admin_addr ms-ag_cluster:Master
commit
```

Add ordering constraint

The colocation constraint has an implicit ordering constraint. It moves the virtual IP resource before it moves the availability group resource. By default the sequence of events is:

1. User issues resource migrate to the availability group master from node1 to node2.
2. The virtual IP resource stops on node 1.
3. The virtual IP resource starts on node 2. At this point, the IP address temporarily points to node 2 while node 2 is still a pre-failover secondary.
4. The availability group master on node 1 is demoted to slave.
5. The availability group slave on node 2 is promoted to master.

To prevent the IP address from temporarily pointing to the node with the pre-failover secondary, add an ordering constraint. To add an ordering constraint, run the following command on one node:

```
crm configure \
order ag_first inf: ms-ag_cluster:promote admin_addr:start
```

IMPORTANT

After you configure the cluster and add the availability group as a cluster resource, you cannot use Transact-SQL to fail over the availability group resources. SQL Server cluster resources on Linux are not coupled as tightly with the operating system as they are on a Windows Server Failover Cluster (WSFC). SQL Server service is not aware of the presence of the cluster. All orchestration is done through the cluster management tools. In SLES use `crm`.

Manually fail over the availability group with `crm`. Don't initiate failover with Transact-SQL. For more information, see [Failover](#).

For more information, see:

- [Managing cluster resources](#).

- [HA Concepts](#)
- [Pacemaker Quick Reference](#)

Next steps

[Operate HA availability group](#)

Configure Ubuntu Cluster and Availability Group Resource

2/14/2018 • 10 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This document explains how to create a three-node cluster on Ubuntu and add a previously created availability group as a resource in the cluster. For high availability, an availability group on Linux requires three nodes - see [High availability and data protection for availability group configurations](#).

NOTE

At this point, SQL Server's integration with Pacemaker on Linux is not as coupled as with WSFC on Windows. From within SQL, there is no knowledge about the presence of the cluster, all orchestration is outside in, and the service is controlled as a standalone instance by Pacemaker. Also, virtual network name is specific to WSFC, there is no equivalent of the same in Pacemaker. Always On dynamic management views that query cluster information return empty rows. You can still create a listener to use it for transparent reconnection after failover, but you have to manually register the listener name in the DNS server with the IP used to create the virtual IP resource (as explained in the following sections).

The following sections walk through the steps to set up a failover cluster solution.

Roadmap

The steps to create an availability group on Linux servers for high availability are different from the steps on a Windows Server failover cluster. The following list describes the high-level steps:

1. [Configure SQL Server on the cluster nodes](#).
2. [Create the availability group](#).
3. Configure a cluster resource manager, like Pacemaker. These instructions are in this document.

The way to configure a cluster resource manager depends on the specific Linux distribution.

IMPORTANT

Production environments require a fencing agent, like STONITH for high availability. The demonstrations in this documentation do not use fencing agents. The demonstrations are for testing and validation only.

A Linux cluster uses fencing to return the cluster to a known state. The way to configure fencing depends on the distribution and the environment. At this time, fencing is not available in some cloud environments. See [Support Policies for RHEL High Availability Clusters - Virtualization Platforms](#) for more information.

4. [Add the availability group as a resource in the cluster](#).

Install and configure Pacemaker on each cluster node

1. On all nodes open the firewall ports. Open the port for the Pacemaker high-availability service, SQL Server instance, and the availability group endpoint. The default TCP port for server running SQL Server is 1433.

```
sudo ufw allow 2224/tcp  
sudo ufw allow 3121/tcp  
sudo ufw allow 21064/tcp  
sudo ufw allow 5405/udp  
  
sudo ufw allow 1433/tcp # Replace with TDS endpoint  
sudo ufw allow 5022/tcp # Replace with DATA_MIRRORING endpoint  
  
sudo ufw reload
```

Alternatively, you can just disable the firewall:

```
sudo ufw disable
```

2. Install Pacemaker packages. On all nodes, run the following commands:

```
sudo apt-get install pacemaker pcs fence-agents resource-agents
```

3. Set the password for the default user that is created when installing Pacemaker and Corosync packages. Use the same password on all nodes.

```
sudo passwd hacluster
```

Enable and start pcasd service and Pacemaker

The following command enables and starts pcasd service and pacemaker. Run on all nodes. This allows the nodes to rejoin the cluster after reboot.

```
sudo systemctl enable pcasd  
sudo systemctl start pcasd  
sudo systemctl enable pacemaker
```

NOTE

Enable pacemaker command may complete with the error 'pacemaker Default-Start contains no runlevels, aborting.' This is harmless, cluster configuration can continue.

Create the Cluster

1. Remove any existing cluster configuration from all nodes.

Running 'sudo apt-get install pcs' installs pacemaker, corosync, and pcs at the same time and starts running all 3 of the services. Starting corosync generates a template '/etc/cluster/corosync.conf' file. To have next steps succeed this file should not exist – so the workaround is to stop pacemaker / corosync and delete '/etc/cluster/corosync.conf', and then next steps complete successfully. 'pcs cluster destroy' does the same thing, and you can use it as a one time initial cluster setup step.

The following command removes any existing cluster configuration files and stops all cluster services. This permanently destroys the cluster. Run it as a first step in a pre-production environment. Note that 'pcs cluster destroy' disabled the pacemaker service and needs to be reenabled. Run the following command on all nodes.

WARNING

The command destroys any existing cluster resources.

```
sudo pcs cluster destroy  
sudo systemctl enable pacemaker
```

2. Create the cluster.

WARNING

Due to a known issue that the clustering vendor is investigating, starting the cluster ('pcs cluster start') fails with following error. This is because the log file configured in /etc/corosync/corosync.conf which is created when the cluster setup command is run, is wrong. To work around this issue, change the log file to: /var/log/corosync/corosync.log. Alternatively you could create the /var/log/cluster/corosync.log file.

```
Job for corosync.service failed because the control process exited with error code.  
See "systemctl status corosync.service" and "journalctl -xe" for details.
```

The following command creates a three-node cluster. Before you run the script, replace the values between < ... >. Run the following command on the primary node.

```
sudo pcs cluster auth <node1> <node2> <node3> -u hacluster -p <password for hacluster>  
sudo pcs cluster setup --name <clusterName> <node1> <node2...> <node3>  
sudo pcs cluster start --all
```

NOTE

If you previously configured a cluster on the same nodes, you need to use '--force' option when running 'pcs cluster setup'. Note this is equivalent to running 'pcs cluster destroy' and pacemaker service needs to be reenabled using 'sudo systemctl enable pacemaker'.

Configure fencing (STONITH)

Pacemaker cluster vendors require STONITH to be enabled and a fencing device configured for a supported cluster setup. When the cluster resource manager cannot determine the state of a node or of a resource on a node, fencing is used to bring the cluster to a known state again. Resource level fencing ensures mainly that there is no data corruption in case of an outage by configuring a resource. You can use resource level fencing, for instance, with DRBD (Distributed Replicated Block Device) to mark the disk on a node as outdated when the communication link goes down. Node level fencing ensures that a node does not run any resources. This is done by resetting the node and the Pacemaker implementation of it is called STONITH (which stands for "shoot the other node in the head"). Pacemaker supports a great variety of fencing devices, e.g. an uninterruptible power supply or management interface cards for servers. For more information, see [Pacemaker Clusters from Scratch](#) and [Fencing and Stonith](#)

Because the node level fencing configuration depends heavily on your environment, we disable it for this tutorial (it can be configured at a later time). Run the following script on the primary node:

```
sudo pcs property set stonith-enabled=false
```

IMPORTANT

Disabling STONITH is just for testing purposes. If you plan to use Pacemaker in a production environment, you should plan a STONITH implementation depending on your environment and keep it enabled. Note that at this point there are no fencing agents for any cloud environments (including Azure) or Hyper-V. Consequentially, the cluster vendor does not offer support for running production clusters in these environments.

Set cluster property start-failure-is-fatal to false

`start-failure-is-fatal` indicates whether a failure to start a resource on a node prevents further start attempts on that node. When set to `false`, the cluster decides whether to try starting on the same node again based on the resource's current failure count and migration threshold. So, after failover occurs, Pacemaker retries starting the availability group resource on the former primary once the SQL instance is available. Pacemaker demotes the replica to secondary and it automatically rejoins the availability group.

To update the property value to `false` run the following script:

```
sudo pcs property set start-failure-is-fatal=false
```

WARNING

After an automatic failover, when `start-failure-is-fatal = true` the resource manager attempts to start the resource. If it fails on the first attempt you have to manually run `pcs resource cleanup <resourceName>` to clean up the resource failure count and reset the configuration.

Install SQL Server resource agent for integration with Pacemaker

Run the following commands on all nodes.

```
sudo apt-get install mssql-server-ha
```

Create a SQL Server login for Pacemaker

1. **On all SQL Servers, create a Server login for Pacemaker.** The following Transact-SQL creates a login:

```
USE [master]
GO
CREATE LOGIN [pacemakerLogin] with PASSWORD= N'ComplexP@$$w0rd!'
ALTER SERVER ROLE [sysadmin] ADD MEMBER [pacemakerLogin]
```

Alternatively, you can set the permissions at a more granular level. The Pacemaker login requires ALTER, CONTROL, and VIEW DEFINITION PERMISSION for managing the availability group as well as VIEW SERVER STATE for the login to be able to run `sp_server_diagnostics`. For more information, see [GRANT Availability Group Permissions \(Transact-SQL\)](#) and [sp_server_diagnostic permissions](#).

The following Transact-SQL grants only the required permission to the Pacemaker login. In the statement below 'ag1' is the name of the availability group that will be added as a cluster resource.

```
GRANT ALTER, CONTROL, VIEW DEFINITION ON AVAILABILITY GROUP::ag1 TO pacemakerLogin  
GRANT VIEW SERVER STATE TO pacemakerLogin
```

2. On all SQL Servers, save the credentials for the SQL Server login.

```
echo 'pacemakerLogin' >> ~/pacemaker-passwd  
echo 'ComplexP@$$w0rd!' >> ~/pacemaker-passwd  
sudo mv ~/pacemaker-passwd /var/opt/mssql/secrets/passwd  
sudo chown root:root /var/opt/mssql/secrets/passwd  
sudo chmod 400 /var/opt/mssql/secrets/passwd # Only readable by root
```

Create availability group resource

To create the availability group resource, use `pcs resource create` command and set the resource properties.

Below command creates a `ocf:mssql:ag` master/slave type resource for availability group with name `ag1`.

```
sudo pcs resource create ag_cluster ocf:mssql:ag ag_name=ag1 --master meta notify=true
```

NOTE

When you create the resource, and periodically afterwards, the Pacemaker resource agent automatically sets the value of `REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT` on the availability group based on the availability group's configuration. For example, if the availability group has three synchronous replicas, the agent will set `REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT` to `1`. For details and additional configuration options, see [High availability and data protection for availability group configurations](#).

Create virtual IP resource

To create the virtual IP address resource, run the following command on one node. Use an available static IP address from the network. Before you run the script, replace the values between `< ... >` with a valid IP address.

```
sudo pcs resource create virtualip ocf:heartbeat:IPAddr2 ip=<10.128.16.240>
```

There is no virtual server name equivalent in Pacemaker. To use a connection string that points to a string server name and not use the IP address, register the IP resource address and desired virtual server name in DNS. For DR configurations, register the desired virtual server name and IP address with the DNS servers on both primary and DR site.

Add colocation constraint

Almost every decision in a Pacemaker cluster, like choosing where a resource should run, is done by comparing scores. Scores are calculated per resource, and the cluster resource manager chooses the node with the highest score for a particular resource. (If a node has a negative score for a resource, the resource cannot run on that node.) Use constraints to configure the decisions of the cluster. Constraints have a score. If a constraint has a score lower than INFINITY, it is only a recommendation. A score of INFINITY means it is mandatory. To ensure that primary replica and the virtual ip resource are on the same host, define a colocation constraint with a score of INFINITY. To add the colocation constraint, run the following command on one node.

```
sudo pcs constraint colocation add virtualip ag_cluster-master INFINITY with-rsc-role=Master
```

Add ordering constraint

The colocation constraint has an implicit ordering constraint. It moves the virtual IP resource before it moves the availability group resource. By default the sequence of events is:

1. User issues `pcs resource move` to the availability group primary from node1 to node2.
2. The virtual IP resource stops on node1.
3. The virtual IP resource starts on node2.

NOTE

At this point, the IP address temporarily points to node2 while node2 is still a pre-failover secondary.

4. The availability group primary on node1 is demoted to secondary.

5. The availability group secondary on node2 is promoted to primary.

To prevent the IP address from temporarily pointing to the node with the pre-failover secondary, add an ordering constraint.

To add an ordering constraint, run the following command on one node:

```
sudo pcs constraint order promote ag_cluster-master then start virtualip
```

IMPORTANT

After you configure the cluster and add the availability group as a cluster resource, you cannot use Transact-SQL to fail over the availability group resources. SQL Server cluster resources on Linux are not coupled as tightly with the operating system as they are on a Windows Server Failover Cluster (WSFC). SQL Server service is not aware of the presence of the cluster. All orchestration is done through the cluster management tools. In RHEL or Ubuntu use `pcs`.

Next steps

[Operate HA availability group](#)

Always On Availability Group failover on Linux

3/6/2018 • 6 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Within the context of an availability group (AG), the primary role and secondary role of availability replicas are typically interchangeable in a process known as failover. Three forms of failover exist: automatic failover (without data loss), planned manual failover (without data loss), and forced manual failover (with possible data loss), typically called *forced failover*. Automatic and planned manual failovers preserve all your data. An AG fails over at the availability-replica level. That is, an AG fails over to one of its secondary replicas (the current failover target).

For background information about failover, see [Failover and failover modes](#).

Manual failover

Use the cluster management tools to fail over an AG managed by an external cluster manager. For example, if a solution uses Pacemaker to manage a Linux cluster, use `pcs` to perform manual failovers on RHEL or Ubuntu. On SLES use `crm`.

IMPORTANT

Under normal operations, do not fail over with Transact-SQL or SQL Server management tools like SSMS or PowerShell.

When `CLUSTER_TYPE = EXTERNAL`, the only acceptable value for `FAILOVER_MODE` is `EXTERNAL`. With these settings, all manual or automatic failover actions are executed by the external cluster manager. For instructions to force failover with potential data loss, see [Force failover](#).

To fail over, the secondary replica that will become the primary replica must be synchronous. If a secondary replica is asynchronous, [change the availability mode](#).

Manually fail over in two steps.

First, [manually fail over by moving AG resource](#) from the cluster node that owns the resources to a new node.

The cluster fails the AG resource over and adds a location constraint. This constraint configures the resource to run on the new node. Remove this constraint in order to successfully fail over in the future.

Second, [remove the location constraint](#).

Step 1. Manually fail over by moving availability group resource

To manually fail over an AG resource named `ag_cluster` to cluster node named `nodeName2`, run the appropriate command for your distribution:

- **RHEL/Ubuntu example**

```
sudo pcs resource move ag_cluster-master nodeName2 --master
```

- **SLES example**

```
crm resource migrate ag_cluster nodeName2
```

IMPORTANT

After you manually fail over a resource, you need to remove a location constraint that is automatically added.

Step 2. Remove the location constraint

During a manual failover, the `pcs` command `move` or `crm` command `migrate` adds a location constraint for the resource to be placed on the new target node. To see the new constraint, run the following command after manually moving the resource:

- **RHEL/Ubuntu example**

```
sudo pcs constraint --full
```

- **SLES example**

```
crm config show
```

Remove the location constraint so future failovers - including automatic failover - succeed.

To remove the constraint, run the following command:

- **RHEL/Ubuntu example**

In this example `ag_cluster-master` is the name of the resource that failed over.

```
sudo pcs resource clear ag_cluster-master
```

- **SLES example**

In this example `ag_cluster` is the name of the resource that failed over.

```
crm resource clear ag_cluster
```

Alternatively, you can run the following command to remove the location constraint.

- **RHEL/Ubuntu example**

In the following command `cli-prefer-ag_cluster-master` is the ID of the constraint that needs to be removed. `sudo pcs constraint --full` returns this ID.

```
sudo pcs constraint remove cli-prefer-ag_cluster-master
```

- **SLES example**

In the following command `cli-prefer-ms-ag_cluster` is the ID of the constraint. `crm config show` returns this ID.

```
crm configure  
delete cli-prefer-ms-ag_cluster  
commit
```

NOTE

Automatic failover does not add a location constraint, so no cleanup is necessary.

For more information:

- [Red Hat - Managing Cluster Resources](#)
- [Pacemaker - Move Resources Manually SLES Administration Guide - Resources](#)

Force failover

A forced failover is intended strictly for disaster recovery. In this case, you cannot fail over with cluster management tools because the primary datacenter is down. If you force failover to an unsynchronized secondary replica, some data loss is possible. Only force failover if you must restore service to the AG immediately and are willing to risk losing data.

If you cannot use the cluster management tools for interacting with the cluster - for example, if the cluster is unresponsive due to a disaster event in the primary data center, you might have to force failover to bypass the external cluster manager. This procedure is not recommended for regular operations because it risks data loss. Use it when the cluster management tools fail to execute the failover action. Functionally, this procedure is similar to [performing a forced manual failover](#) on an AG in Windows.

This process for forcing failover is specific to SQL Server on Linux.

1. Verify that the AG resource is not managed by the cluster any more.

- Set the resource to unmanaged mode on the target cluster node. This command signals the resource agent to stop resource monitoring and management. For example:

```
sudo pcs resource unmanage <resourceName>
```

- If the attempt to set the resource mode to unmanaged mode fails, delete the resource. For example:

```
sudo pcs resource delete <resourceName>
```

NOTE

When you delete a resource, it also deletes all of the associated constraints.

2. On the instance of SQL Server that hosts the secondary replica, set the session context variable `external_cluster`.

```
EXEC sp_set_session_context @key = N'external_cluster', @value = N'yes';
```

3. Fail over the AG with Transact-SQL. In the following example, replace `<MyAg>` with the name of your AG. Connect to the instance of SQL Server that hosts the target secondary replica and run the following command:

```
ALTER AVAILABILITY GROUP <MyAg> FORCE_FAILOVER_ALLOW_DATA_LOSS;
```

4. After a forced failover, bring the AG to a healthy state before either restarting the cluster resource

monitoring and management or recreating the AG resource. Review the [Essential Tasks After a Forced Failover](#).

5. Either restart cluster resource monitoring and management:

To restart the cluster resource monitoring and management, run the following command:

```
sudo pcs resource manage <resourceName>
sudo pcs resource cleanup <resourceName>
```

If you deleted the cluster resource, recreate it. To recreate the cluster resource, follow the instructions at [Create availability group resource](#).

IMPORTANT

Do not use the preceding steps for disaster recovery drills because they risk data loss. Instead change the asynchronous replica to synchronous, and the instructions for [normal manual failover](#).

Database level monitoring and failover trigger

For `CLUSTER_TYPE=EXTERNAL`, the failover trigger semantics are different compared to WSFC. When the AG is on an instance of SQL Server in a WSFC, transitioning out of `ONLINE` state for the database causes the AG health to report a fault. In response, the cluster manager triggers a failover action. On Linux, the SQL Server instance cannot communicate with the cluster. Monitoring for database health is done *outside-in*. If user opted in for database level failover monitoring and failover (by setting the option `DB_FAILOVER=ON` when creating the AG), the cluster will check if the database state is `ONLINE` every time it runs a monitoring action. The cluster queries the state in `sys.databases`. For any state different than `ONLINE`, it will trigger a failover automatically (if automatic failover conditions are met). The actual time of the failover depends on the frequency of the monitoring action as well as the database state being updated in `sys.databases`.

Automatic failover requires at least one synchronous replica.

Next steps

[Configure Red Hat Enterprise Linux Cluster for SQL Server Availability Group Cluster Resources](#)

[Configure SUSE Linux Enterprise Server Cluster for SQL Server Availability Group Cluster Resources](#)

[Configure Ubuntu Cluster for SQL Server Availability Group Cluster Resources](#)

Operate Always On Availability Groups on Linux

3/6/2018 • 4 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Upgrade availability group

Before you upgrade an availability group, review the patterns and practices at [Upgrading availability group replica instances](#).

The following sections explain how to perform a rolling upgrade with SQL Server instances on Linux with availability groups.

Upgrade steps on Linux

When availability group replicas are on instances of SQL Server in Linux, the cluster type of the availability group is either `EXTERNAL` or `NONE`. An availability group that is managed by a cluster manager besides Windows Server Failover Cluster (WSFC) is `EXTERNAL`. Pacemaker with Corosync is an example of an external cluster manager. An availability group with no cluster manager has cluster type `NONE`. The upgrade steps outlined here are specific for availability groups of cluster type `EXTERNAL` or `NONE`.

The order in which you upgrade instances depends on if their role is secondary and whether or not they host synchronous or asynchronous replicas. Upgrade instances of SQL Server that host asynchronous secondary replicas first. Then upgrade instances that host synchronous secondary replicas.

NOTE

If an availability group only has asynchronous replicas, to avoid any data loss change one replica to synchronous and wait until it is synchronized. Then upgrade this replica.

Before you begin, back up each database.

1. Stop the resource on the node hosting the secondary replica targeted for upgrade.

Before running the upgrade command, stop the resource so the cluster will not monitor it and fail it unnecessarily. The following example adds a location constraint on the node that will result on the resource to be stopped. Update `ag_cluster-master` with the resource name and `nodeName1` with the node hosting the replica targeted for upgrade.

```
pcs constraint location ag_cluster-master avoids nodeName1
```

2. Upgrade SQL Server on the secondary replica.

The following example upgrades `mssql-server` and `mssql-server-ha` packages.

```
sudo yum update mssql-server
sudo yum update mssql-server-ha
```

3. Remove the location constraint.

Before running the upgrade command, stop the resource so the cluster will not monitor it and fail it

unnecessarily. The following example adds a location constraint on the node that will result on the resource to be stopped. Update `ag_cluster-master` with the resource name and `nodeName1` with the node hosting the replica targeted for upgrade.

```
pcs constraint remove location-ag_cluster-master-rhel1--INFINITY
```

As a best practice, ensure the resource is started (using `pcs status` command) and the secondary replica is connected and synchronized state after upgrade.

4. After all secondary replicas are upgraded, manually fail over to one of the synchronous secondary replicas.

For availability groups with `EXTERNAL` cluster type, use the cluster management tools to fail over; availability groups with `NONE` cluster type should use Transact-SQL to fail over. The following example fails over an availability group with the cluster management tools. Replace `<targetReplicaName>` with the name of the synchronous secondary replica that will become primary:

```
sudo pcs resource move ag_cluster-master <targetReplicaName> --master
```

IMPORTANT

The following steps only apply to availability groups that do not have a cluster manager.

If the availability group cluster type is `NONE`, manually fail over. Complete the following steps in order:

- a. The following command sets the primary replica to secondary. Replace `AG1` with the name of your availability group. Run the Transact-SQL command on the instance of SQL Server that hosts the primary replica.

```
ALTER AVAILABILITY GROUP [ag1] SET (ROLE = SECONDARY);
```

- b. The following command sets a synchronous secondary replica to primary. Run the following Transact-SQL command on the target instance of SQL Server - the instance that hosts the synchronous secondary replica.

```
ALTER AVAILABILITY GROUP [ag1] FAILOVER;
```

5. After failover, upgrade SQL Server on the old primary replica by repeating the preceding procedure.

The following example upgrades `mssql-server` and `mssql-server-ha` packages.

```
# add constraint for the resource to stop on the upgraded node
# replace 'nodename2' with the name of the cluster node targeted for upgrade
pcs constraint location ag_cluster-master avoids nodeName2
sudo yum update mssql-server
sudo yum update mssql-server-ha
```

```
# upgrade mssql-server and mssql-server-ha packages
sudo yum update mssql-server
sudo yum update mssql-server-ha
```

```
# remove the constraint; make sure the resource is started and replica is connected and synchronized  
pcs constraint remove location-ag_cluster-master-rhel1--INFINITY
```

6. For an availability groups with an external cluster manager - where cluster type is EXTERNAL, clean up the location constraint that was caused by the manual failover.

```
sudo pcs constraint remove cli-prefer-ag_cluster-master
```

7. Resume data movement for the newly upgraded secondary replica - the former primary replica. This step is required when a higher version instance of SQL Server is transferring log blocks to a lower version instance in an availability group. Run the following command on the new secondary replica (the previous primary replica).

```
ALTER DATABASE database_name SET HADR RESUME;
```

After upgrading all servers, you can fail back. Fail over back to the original primary - if necessary.

Drop an availability group

To delete an availability group, run [DROP AVAILABILITY GROUP](#). If the cluster type is `EXTERNAL` or `NONE` run the command on every instance of SQL Server that hosts a replica. For example, to drop an availability group named `group_name` run the following command:

```
DROP AVAILABILITY GROUP group_name
```

Next steps

[Configure Red Hat Enterprise Linux Cluster for SQL Server Availability Group Cluster Resources](#)

[Configure SUSE Linux Enterprise Server Cluster for SQL Server Availability Group Cluster Resources](#)

[Configure Ubuntu Cluster for SQL Server Availability Group Cluster Resources](#)

Configure a SQL Server Availability Group for read-scale on Linux

2/14/2018 • 10 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

You can configure a SQL Server Always On Availability Group (AG) for read-scale workloads on Linux. There are two types of architectures for AGs. A architecture for high availability uses a cluster manager to provide improved business continuity. This architecture also can include read-scale replicas. To create the high-availability architecture, see [Configure SQL Server Always On Availability Group for high availability on Linux](#). The other architecture supports only read-scale workloads. This article explains how to create an AG without a cluster manager for read-scale workloads. This architecture provides read-scale only. It doesn't provide high availability.

NOTE

An availability group with `CLUSTER_TYPE = NONE` can include replicas hosted on different operating system platforms. It cannot support high availability.

Prerequisites

Before you create the availability group, you need to:

- Set your environment so that all the servers that will host availability replicas can communicate.
- Install SQL Server.

NOTE

On Linux, you must create an availability group before you add it as a cluster resource to be managed by the cluster. This document provides an example that creates the availability group. For distribution-specific instructions to create the cluster and add the availability group as a cluster resource, see the links under "Next steps."

1. Update the computer name for each host.

Each SQL Server name must be:

- 15 characters or less.
- Unique within the network.

To set the computer name, edit `/etc/hostname`. The following script lets you edit `/etc/hostname` with `vi`:

```
sudo vi /etc/hostname
```

2. Configure the hosts file.

NOTE

If hostnames are registered with their IP in the DNS server, you don't need to do the following steps. Validate that all the nodes intended to be part of the availability group configuration can communicate with each other. (A ping to the hostname should reply with the corresponding IP address.) Also, make sure that the /etc/hosts file doesn't contain a record that maps the localhost IP address 127.0.0.1 with the hostname of the node.

The hosts file on every server contains the IP addresses and names of all servers that will participate in the availability group.

The following command returns the IP address of the current server:

```
sudo ip addr show
```

Update `/etc/hosts`. The following script lets you edit `/etc/hosts` with `vi`:

```
sudo vi /etc/hosts
```

The following example shows `/etc/hosts` on **node1** with additions for **node1**, **node2**, and **node3**. In this document, **node1** refers to the server that hosts the primary replica. And **node2** and **node3** refer to servers that host the secondary replicas.

```
127.0.0.1    localhost localhost4 localhost4.localdomain4
::1          localhost localhost6 localhost6.localdomain6
10.128.18.12 node1
10.128.16.77 node2
10.128.15.33 node3
```

Install SQL Server

Install SQL Server. The following links point to SQL Server installation instructions for various distributions:

- [Red Hat Enterprise Linux](#)
- [SUSE Linux Enterprise Server](#)
- [Ubuntu](#)

Enable AlwaysOn availability groups and restart mssql-server

Enable AlwaysOn availability groups on each node that hosts a SQL Server instance. Then restart `mssql-server`. Run the following script:

```
sudo /opt/mssql/bin/mssql-conf set hadr.hadrenabled 1
sudo systemctl restart mssql-server
```

Enable an AlwaysOn_health event session

You can optionally enable AlwaysOn availability groups extended events to help with root-cause diagnosis when you troubleshoot an availability group. Run the following command on each instance of SQL Server:

```
ALTER EVENT SESSION AlwaysOn_health ON SERVER WITH (STARTUP_STATE=ON);
GO
```

For more information about this XE session, see [AlwaysOn extended events](#).

Create a database mirroring endpoint user

The following Transact-SQL script creates a login named `dbm_login` and a user named `dbm_user`. Update the script with a strong password. To create the database mirroring endpoint user, run the following command on all SQL Server instances:

```
CREATE LOGIN dbm_login WITH PASSWORD = '**<1Sample_Strong_Password!@#>**';
CREATE USER dbm_user FOR LOGIN dbm_login;
```

Create a certificate

The SQL Server service on Linux uses certificates to authenticate communication between the mirroring endpoints.

The following Transact-SQL script creates a master key and a certificate. It then backs up the certificate and secures the file with a private key. Update the script with strong passwords. Connect to the primary SQL Server instance. To create the certificate, run the following Transact-SQL script:

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '**<Master_Key_Password>**';
CREATE CERTIFICATE dbm_certificate WITH SUBJECT = 'dbm';
BACKUP CERTIFICATE dbm_certificate
    TO FILE = '/var/opt/mssql/data/dbm_certificate.cer'
    WITH PRIVATE KEY (
        FILE = '/var/opt/mssql/data/dbm_certificate.pvk',
        ENCRYPTION BY PASSWORD = '**<Private_Key_Password>**'
    );
```

At this point, your primary SQL Server replica has a certificate at `/var/opt/mssql/data/dbm_certificate.cer` and a private key at `/var/opt/mssql/data/dbm_certificate.pvk`. Copy these two files to the same location on all servers that will host availability replicas. Use the `mssql` user, or give permission to the `mssql` user to access these files.

For example, on the source server, the following command copies the files to the target machine. Replace the `**<node2>**` values with the names of the SQL Server instances that will host the replicas.

```
cd /var/opt/mssql/data
scp dbm_certificate.* root@**<node2>**:/var/opt/mssql/data/
```

On each target server, give permission to the `mssql` user to access the certificate.

```
cd /var/opt/mssql/data
chown mssql:mssql dbm_certificate.*
```

Create the certificate on secondary servers

The following Transact-SQL script creates a master key and a certificate from the backup that you created on the primary SQL Server replica. The command also authorizes the user to access the certificate. Update the script with strong passwords. The decryption password is the same password that you used to create the .pkv file in a previous step. To create the certificate, run the following script on all secondary servers:

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '**<Master_Key_Password>**';
CREATE CERTIFICATE dbm_certificate
    AUTHORIZATION dbm_user
    FROM FILE = '/var/opt/mssql/data/dbm_certificate.cer'
    WITH PRIVATE KEY (
        FILE = '/var/opt/mssql/data/dbm_certificate.pvk',
        DECRYPTION BY PASSWORD = '**<Private_Key_Password>**'
    );
```

Create the database mirroring endpoints on all replicas

Database mirroring endpoints use the Transmission Control Protocol (TCP) to send and receive messages between the server instances that participate in database mirroring sessions or host availability replicas. The database mirroring endpoint listens on a unique TCP port number.

The following Transact-SQL script creates a listening endpoint named `Hadr_endpoint` for the availability group. It starts the endpoint and gives connection permission to the user that you created. Before you run the script, replace the values between `**< ... >**`. Optionally you can include an IP address `LISTENER_IP = (0.0.0.0)`. The listener IP address must be an IPv4 address. You can also use `0.0.0.0`.

Update the following Transact-SQL script for your environment on all SQL Server instances:

```
CREATE ENDPOINT [Hadr_endpoint]
    AS TCP (LISTENER_PORT = **<5022>**)
    FOR DATA_MIRRORING (
        ROLE = ALL,
        AUTHENTICATION = CERTIFICATE dbm_certificate,
        ENCRYPTION = REQUIRED ALGORITHM AES
    );
ALTER ENDPOINT [Hadr_endpoint] STATE = STARTED;
GRANT CONNECT ON ENDPOINT::[Hadr_endpoint] TO [dbm_login];
```

NOTE

If you use SQL Server Express Edition on one node to host a configuration-only replica, the only valid value for `ROLE` is `WITNESS`. Run the following script on SQL Server Express Edition:

```
CREATE ENDPOINT [Hadr_endpoint]
    AS TCP (LISTENER_PORT = **<5022>**)
    FOR DATA_MIRRORING (
        ROLE = WITNESS,
        AUTHENTICATION = CERTIFICATE dbm_certificate,
        ENCRYPTION = REQUIRED ALGORITHM AES
    );
ALTER ENDPOINT [Hadr_endpoint] STATE = STARTED;
GRANT CONNECT ON ENDPOINT::[Hadr_endpoint] TO [dbm_login];
```

The TCP port on the firewall must be open for the listener port.

IMPORTANT

For the SQL Server 2017 release, the only authentication method supported for the database mirroring endpoint is `CERTIFICATE`. The `WINDOWS` option will be enabled in a future release.

For more information, see [The database mirroring endpoint \(SQL Server\)](#).

Create the AG

Create the AG. Set `CLUSTER_TYPE = NONE`. In addition, set each replica with `FAILOVER_MODE = NONE`. Client applications running analytics or reporting workloads can directly connect to the secondary databases. You also can create a read-only routing list. Connections to the primary replica forward read connection requests to each of the secondary replicas from the routing list in a round-robin fashion.

The following Transact-SQL script creates an AG named `ag1`. The script configures the AG replicas with `SEEDING_MODE = AUTOMATIC`. This setting causes SQL Server to automatically create the database on each secondary server after it is added to the AG. Update the following script for your environment. Replace the `<node1>` and `<node2>` values with the names of the SQL Server instances that host the replicas. Replace the `<5022>` value with the port you set for the endpoint. Run the following Transact-SQL script on the primary SQL Server replica:

```
CREATE AVAILABILITY GROUP [ag1]
    WITH (CLUSTER_TYPE = NONE)
    FOR REPLICA ON
        N'<node1>' WITH (
            ENDPOINT_URL = N'tcp://<node1>:<5022>',
            AVAILABILITY_MODE = ASYNCHRONOUS_COMMIT,
            FAILOVER_MODE = MANUAL,
            SEEDING_MODE = AUTOMATIC,
            SECONDARY_ROLE (ALLOW_CONNECTIONS = ALL)
        ),
        N'<node2>' WITH (
            ENDPOINT_URL = N'tcp://<node2>:<5022>',
            AVAILABILITY_MODE = ASYNCHRONOUS_COMMIT,
            FAILOVER_MODE = MANUAL,
            SEEDING_MODE = AUTOMATIC,
            SECONDARY_ROLE (ALLOW_CONNECTIONS = ALL)
        );
ALTER AVAILABILITY GROUP [ag1] GRANT CREATE ANY DATABASE;
```

Join secondary SQL Servers to the AG

The following Transact-SQL script joins a server to an AG named `ag1`. Update the script for your environment. On each secondary SQL Server replica, run the following Transact-SQL script to join the AG:

```
ALTER AVAILABILITY GROUP [ag1] JOIN WITH (CLUSTER_TYPE = NONE);
ALTER AVAILABILITY GROUP [ag1] GRANT CREATE ANY DATABASE;
```

Add a database to the availability group

Ensure that the database you add to the availability group is in full recovery mode and has a valid log backup. If this is a test database or a newly created database, take a database backup. On the primary SQL Server, run the following Transact-SQL script to create and back up a database called `db1`:

```
CREATE DATABASE [db1];
ALTER DATABASE [db1] SET RECOVERY FULL;
BACKUP DATABASE [db1]
    TO DISK = N'/var/opt/mssql/data/db1.bak';
```

On the primary SQL Server replica, run the following Transact-SQL script to add a database called `db1` to an availability group called `ag1`:

```
ALTER AVAILABILITY GROUP [ag1] ADD DATABASE [db1];
```

Verify that the database is created on the secondary servers

On each secondary SQL Server replica, run the following query to see if the `db1` database was created and is synchronized:

```
SELECT * FROM sys.databases WHERE name = 'db1';
GO
SELECT DB_NAME(database_id) AS 'database', synchronization_state_desc FROM
sys.dm_hadr_database_replica_states;
```

This AG isn't a high-availability configuration. If you need high availability, follow the instructions at [Configure an Always On Availability Group for SQL Server on Linux](#). Specifically, create the AG with `CLUSTER_TYPE=WSFC` (in Windows) or `CLUSTER_TYPE=EXTERNAL` (in Linux). Then integrate with a cluster manager by using either Windows Server failover clustering on Windows or Pacemaker on Linux.

Connect to read-only secondary replicas

There are two ways to connect to read-only secondary replicas. Applications can connect directly to the SQL Server instance that hosts the secondary replica and query the databases. They also can use read-only routing, which requires a listener.

- [Readable secondary replicas](#)
- [Read-only routing](#)

Fail over the primary replica on a read-scale Availability Group

Each AG has only one primary replica. The primary replica allows reads and writes. To change which replica is primary, you can fail over. In an AG for high availability, the cluster manager automates the failover process. In an AG with cluster type NONE, the failover process is manual.

There are two ways to fail over the primary replica in an AG with cluster type NONE:

- Forced manual failover with data loss
- Manual failover without data loss

Forced manual failover with data loss

Use this method when the primary replica isn't available and can't be recovered.

To force failover with data loss, connect to the SQL Server instance that hosts the target secondary replica and run:

```
ALTER AVAILABILITY GROUP [ag1] FORCE_FAILOVER_ALLOW_DATA_LOSS;
```

Manual failover without data loss

Use this method when the primary replica is available, but you need to temporarily or permanently change the configuration and change the SQL Server instance that hosts the primary replica. Before you issue the manual failover, ensure that the target secondary replica is up to date to avoid potential data loss.

To manually fail over without data loss:

1. Make the target secondary replica `SYNCHRONOUS_COMMIT`.

```
ALTER AVAILABILITY GROUP [ag1]
    MODIFY REPLICA ON N'<node2>'
        WITH (AVAILABILITY_MODE = SYNCHRONOUS_COMMIT);
```

- Run the following query to identify that active transactions are committed to the primary replica and at least one synchronous secondary replica:

```
SELECT ag.name,
    drs.database_id,
    drs.group_id,
    drs.replica_id,
    drs.synchronization_state_desc,
    ag.sequence_number
FROM sys.dm_hadr_database_replica_states drs, sys.availability_groups ag
WHERE drs.group_id = ag.group_id;
```

The secondary replica is synchronized when `synchronization_state_desc` is `SYNCHRONIZED`.

- Update `REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT` to 1.

The following script sets `REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT` to 1 on an AG named `ag1`. Before you run the following script, replace `ag1` with the name of your AG:

```
ALTER AVAILABILITY GROUP [ag1]
    SET REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT = 1;
```

This setting ensures that every active transaction is committed to the primary replica and at least one synchronous secondary replica.

- Demote the primary replica to a secondary replica. After the primary replica is demoted, it's read-only. Run this command on the SQL Server instance that hosts the primary replica to update the role to `SECONDARY`:

```
ALTER AVAILABILITY GROUP [ag1]
    SET (ROLE = SECONDARY);
```

- Promote the target secondary replica to primary.

```
ALTER AVAILABILITY GROUP ag1 FORCE_FAILOVER_ALLOW_DATA_LOSS;
```

NOTE

To delete an AG, use [DROP AVAILABILITY GROUP](#). For an AG created with cluster type NONE or EXTERNAL, the command must be executed on all replicas that are part of the AG.

Next steps

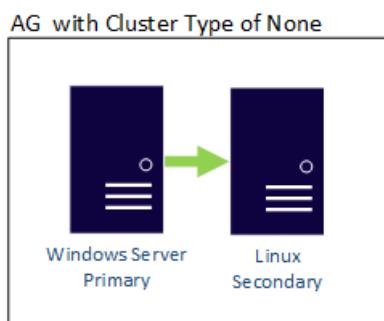
- [Configure a distributed Availability Group](#)
- [Learn more about availability groups](#)
- [Perform a forced manual failover](#)

Configure SQL Server Always On Availability Group on Windows and Linux (cross-platform)

2/14/2018 • 8 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✔ SQL Server (starting with 2017) ✖ Azure SQL Database ✖ Azure SQL Data Warehouse
✖ Parallel Data Warehouse

This article explains the steps to create an Always On Availability Group (AG) with one replica on a Windows server and the other replica on a Linux server. This configuration is cross-platform because the replicas are on different operating systems. Use this configuration for migration from one platform to the other or disaster recovery (DR). This configuration does not support high-availability because there is no cluster solution to manage a cross-platform configuration.



Before proceeding, you should be familiar with installation and configuration for SQL Server instances on Windows and Linux.

Scenario

In this scenario, two servers are on different operating systems. A Windows Server 2016 named `WinSQLInstance` hosts the primary replica. A Linux server named `LinuxSQLInstance` host the secondary replica.

Configure the AG

The steps to create the AG are the same as the steps to create an AG for read-scale workloads. The AG cluster type is NONE, because there is no cluster manager.

NOTE

For the scripts in this article, angle brackets `<` and `>` identify values that you need to replace for your environment. The angle brackets themselves are not required for the scripts.

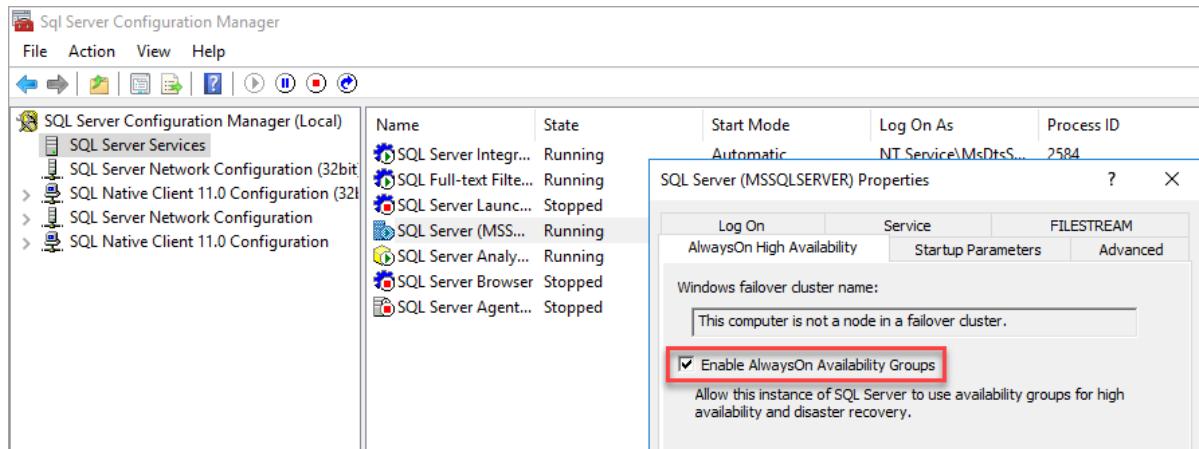
1. Install SQL Server 2017 on Windows Server 2016, enable Always On Availability Groups from SQL Server Configuration Manager, and set mixed mode authentication.

TIP

If you are validating this solution in Azure, place both servers in the same availability set to ensure they are separated in the data center.

Enable Availability Groups

For instructions, see [Enable and disable Always On Availability Groups \(SQL Server\)](#).



SQL Server Configuration Manager notes that the computer is not a node in a failover cluster.

After you enable Availability Groups, restart SQL Server.

Set mixed mode authentication

For instructions, see [Change server authentication mode](#).

2. Install SQL Server 2017 on Linux. For instructions, see [Install SQL Server](#). Enable `hadr` via mssql-conf.

To enable `hadr` via mssql-conf from a shell prompt, issue the following command:

```
sudo /opt/mssql/bin/mssql-conf set hadr.hadrenabled 1
```

After you enable `hadr`, restart the SQL Server instance.

The following image shows this complete step.

A screenshot of a terminal window with a black background and white text. It shows the command 'sudo /opt/mssql/bin/mssql-conf set hadr.hadrenabled 1' being run, followed by a message about needing a restart. Then, the command 'sudo systemctl restart mssql-server.service' is run. The session is for user 'domainadmin' on a host named 'linuxsql'.

3. Configure hosts file on both servers or register the server names with DNS.
4. Open up firewall ports for TPC 1433 and 5022 on both Windows and Linux.
5. On the primary replica, create a database login and password.

```
CREATE LOGIN dbm_login WITH PASSWORD = '<C0m9L3xP@55w0rd!>';
CREATE USER dbm_user FOR LOGIN dbm_login;
GO
```

6. On the primary replica, create a master key and certificate, then back up the certificate with a private key.

```

CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<C0m9L3xP@55w0rd!>';
CREATE CERTIFICATE dbm_certificate WITH SUBJECT = 'dbm';
BACKUP CERTIFICATE dbm_certificate
TO FILE = 'C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\MSSQL\DATA\dbm_certificate.cer'
WITH PRIVATE KEY (
    FILE = 'C:\Program Files\Microsoft SQL
Server\MSSQL14.MSSQLSERVER\MSSQL\DATA\dbm_certificate.pvk',
    ENCRYPTION BY PASSWORD = '<C0m9L3xP@55w0rd!>'
);
GO

```

7. Copy the certificate and private key to the Linux server (secondary replica) at `/var/opt/mssql/data`. You can use `pscp` to copy the files to the Linux server.

8. Set the group and ownership of the private key and the certificate to `mssql:mssql`.

The following script sets the group and ownership of the files.

```

sudo chown mssql:mssql /var/opt/mssql/data/dbm_certificate.pvk
sudo chown mssql:mssql /var/opt/mssql/data/dbm_certificate.cer

```

In the following diagram, ownership and group are set correctly for the certificate and key.

```

MINGW64:/c/Users/mikeray/test
domainadmin@linuxsql:~$ sudo ls -l /var/opt/mssql/data
total 54224
-rw-r--r-- 1 mssql mssql      680 Jan 30 18:08 dbm_certificate.cer
-rw-r--r-- 1 mssql mssql     1212 Jan 30 18:08 dbm_certificate.pvk
-rw-r----- 1 mssql mssql 4194304 Jan 30 17:53 master.mdf
-rw-r----- 1 mssql mssql 1572864 Jan 30 18:23 mastlog.ldf
-rw-r----- 1 mssql mssql 8388608 Jan 30 16:42 modellog.ldf
-rw-r----- 1 mssql mssql 8388608 Jan 30 16:42 model.mdf
-rw-r----- 1 mssql mssql 15400960 Jan 30 16:42 msdbdata.mdf
-rw-r----- 1 mssql mssql 786432 Jan 30 16:42 msdblog.ldf
-rw-r----- 1 mssql mssql 8388608 Jan 30 16:42 tempdb.mdf
-rw-r----- 1 mssql mssql 8388608 Jan 30 16:42 templog.ldf
domainadmin@linuxsql:~$ 

```

9. On the secondary replica, create a database login and password and create a master key.

```

CREATE LOGIN dbm_login WITH PASSWORD = '<C0m9L3xP@55w0rd!>';
CREATE USER dbm_user FOR LOGIN dbm_login;
GO
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<M@st3rKeyP@55w0rD!>'
GO

```

10. On the secondary replica, restore the certificate you copied to `/var/opt/mssql/data`.

```

CREATE CERTIFICATE dbm_certificate
    AUTHORIZATION dbm_user
    FROM FILE = '/var/opt/mssql/data/dbm_certificate.cer'
    WITH PRIVATE KEY (
        FILE = '/var/opt/mssql/data/dbm_certificate.pvk',
        DECRYPTION BY PASSWORD = '<C0m9L3xP@55w0rd!>'
    )
GO

```

11. On the primary replica, create an endpoint.

```
CREATE ENDPOINT [Hadr_endpoint]
AS TCP (LISTENER_IP = (0.0.0.0), LISTENER_PORT = 5022)
FOR DATA_MIRRORING (
    ROLE = ALL,
    AUTHENTICATION = CERTIFICATE dbm_certificate,
    ENCRYPTION = REQUIRED ALGORITHM AES
);
ALTER ENDPOINT [Hadr_endpoint] STATE = STARTED;
GRANT CONNECT ON ENDPOINT::[Hadr_endpoint] TO [dbm_login]
GO
```

IMPORTANT

The firewall must be open for the listener TCP port. In the preceding script, the port is 5022. Use any available TCP port.

12. On the secondary replica, create the endpoint. Repeat the preceding script on the secondary replica to create the endpoint.
13. On the primary replica, create the AG with `CLUSTER_TYPE = NONE`. The example script uses `SEEDING_MODE = AUTOMATIC` to create the AG.

NOTE

When the Windows instance of SQL Server uses different paths for data and log files, automatic seeding fails to the Linux instance of SQL Server because these paths do not exist on the secondary replica. To use the following script for a cross-platform AG, the database requires the same path for the data and log files on the Windows server.

Alternatively you can update the script to set `SEEDING_MODE = MANUAL` and then back up and restore the database with `NORECOVERY` to seed the database.

This behavior applies to Azure Marketplace images.

For more information about automatic seeding, see [Automatic Seeding - Disk Layout](#).

Before you run the script, update the values for your AGs.

- Replace `<WinSQLInstance>` with the server name of the primary replica SQL Server instance.
- Replace `<LinuxSQLInstance>` with the server name of the secondary replica SQL Server instance.

To create the AG, update the values and run the script on the primary replica.

```

CREATE AVAILABILITY GROUP [ag1]
    WITH (CLUSTER_TYPE = NONE)
    FOR REPLICA ON
        N'<WinSQLInstance>'
    WITH (
        ENDPOINT_URL = N'tcp://<WinSQLInstance>:5022',
        AVAILABILITY_MODE = ASYNCHRONOUS_COMMIT,
        SEEDING_MODE = AUTOMATIC,
        FAILOVER_MODE = MANUAL,
        SECONDARY_ROLE (ALLOW_CONNECTIONS = ALL)
    ),
        N'<LinuxSQLInstance>'
    WITH (
        ENDPOINT_URL = N'tcp://<LinuxSQLInstance>:5022',
        AVAILABILITY_MODE = ASYNCHRONOUS_COMMIT,
        SEEDING_MODE = AUTOMATIC,
        FAILOVER_MODE = MANUAL,
        SECONDARY_ROLE (ALLOW_CONNECTIONS = ALL)
    )
GO

```

For more information, see [CREATE AVAILABILITY GROUP \(Transact-SQL\)](#).

14. On the secondary replica, join the AG.

```

ALTER AVAILABILITY GROUP [ag1] JOIN WITH (CLUSTER_TYPE = NONE)
ALTER AVAILABILITY GROUP [ag1] GRANT CREATE ANY DATABASE
GO

```

15. Create a database for the AG. The example steps use a database named <TestDB>. If you are using automatic seeding, set the same path for both the data and the log files.

Before you run the script, update the values for your database.

- Replace <TestDB> with the name of your database.
- Replace <F:\Path> with the path for your database and log files. Use the same path for the database and log files.

You can also use the default paths.

To create your database, run the script.

```

CREATE DATABASE [<TestDB>]
    CONTAINMENT = NONE
    ON PRIMARY ( NAME = N'<TestDB>', FILENAME = N'<F:\Path>\<TestDB>.mdf')
    LOG ON ( NAME = N'<TestDB>_log', FILENAME = N'<F:\Path>\<TestDB>_log.ldf')
GO

```

16. Take a full backup of the database.

17. If you are not using automatic seeding, restore the database on the secondary replica (Linux) server. [Migrate a SQL Server database from Windows to Linux using backup and restore](#). Restore the database **WITH NORECOVERY** on the secondary replica.

18. Add the database to the AG. Update the example script. Replace <TestDB> with the name of your database. On the primary replica, run the SQL query to add the database to the AG.

```
ALTER AG [ag1] ADD DATABASE <TestDB>
GO
```

19. Verify that the database is getting populated on the secondary replica.

Fail over the primary replica

Each AG has only one primary replica. The primary replica allows reads and writes. To change which replica is primary, you can fail over. In an AG for high availability, the cluster manager automates the failover process. In an AG with cluster type NONE, the failover process is manual.

There are two ways to fail over the primary replica in an AG with cluster type NONE:

- Forced manual failover with data loss
- Manual failover without data loss

Forced manual failover with data loss

Use this method when the primary replica isn't available and can't be recovered.

To force failover with data loss, connect to the SQL Server instance that hosts the target secondary replica and run:

```
ALTER AVAILABILITY GROUP [ag1] FORCE_FAILOVER_ALLOW_DATA_LOSS;
```

Manual failover without data loss

Use this method when the primary replica is available, but you need to temporarily or permanently change the configuration and change the SQL Server instance that hosts the primary replica. Before you issue the manual failover, ensure that the target secondary replica is up to date to avoid potential data loss.

To manually fail over without data loss:

1. Make the target secondary replica `SYNCHRONOUS_COMMIT`.

```
ALTER AVAILABILITY GROUP [ag1]
MODIFY REPLICA ON N'<node2>'
WITH (AVAILABILITY_MODE = SYNCHRONOUS_COMMIT);
```

2. Run the following query to identify that active transactions are committed to the primary replica and at least one synchronous secondary replica:

```
SELECT ag.name,
       drs.database_id,
       drs.group_id,
       drs.replica_id,
       drs.synchronization_state_desc,
       ag.sequence_number
  FROM sys.dm_hadr_database_replica_states drs, sys.availability_groups ag
 WHERE drs.group_id = ag.group_id;
```

The secondary replica is synchronized when `synchronization_state_desc` is `SYNCHRONIZED`.

3. Update `REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT` to 1.

The following script sets `REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT` to 1 on an AG named `ag1`. Before you run the following script, replace `ag1` with the name of your AG:

```
ALTER AVAILABILITY GROUP [ag1]
    SET REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT = 1;
```

This setting ensures that every active transaction is committed to the primary replica and at least one synchronous secondary replica.

4. Demote the primary replica to a secondary replica. After the primary replica is demoted, it's read-only. Run this command on the SQL Server instance that hosts the primary replica to update the role to **SECONDARY**:

```
ALTER AVAILABILITY GROUP [ag1]
    SET (ROLE = SECONDARY);
```

5. Promote the target secondary replica to primary.

```
ALTER AVAILABILITY GROUP ag1 FORCE_FAILOVER_ALLOW_DATA_LOSS;
```

NOTE

To delete an AG, use [DROP AVAILABILITY GROUP](#). For an AG created with cluster type NONE or EXTERNAL, the command must be executed on all replicas that are part of the AG.

This article reviewed the steps to create a cross-platform AG to support migration or read-scale workloads. It can be used for manual disaster recovery. It also explained how to fail over the AG. A cross-platform AG uses cluster type **NONE** and does not support high availability because there is no cluster tool across-platforms.

Next steps

[Overview of Always On Availability Groups](#)

[SQL Server availability basics for Linux deployments](#)

Security limitations for SQL Server on Linux

2/14/2018 • 1 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

SQL Server on Linux currently has the following limitations:

- A standard password policy is provided. MUST_CHANGE is the only option you may configure.
- Extensible Key Management is not supported.
- Using keys stored in the Azure Key Vault is not supported.
- SQL Server generates its own self-signed certificate for encrypting connections. SQL Server can be configured to use a user provided certificate for TLS.

For more information about security features available in SQL Server, see the [Security Center for SQL Server Database Engine and Azure SQL Database](#).

Next steps

For common security tasks, see [Get started with security features of SQL Server on Linux](#). For a script to change the TCP port number, the SQL Server directories, and configure traceflags or collation, see [Configure SQL Server on Linux with mssql-conf](#).

Walkthrough for the security features of SQL Server on Linux

2/23/2018 • 8 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

If you are a Linux user who is new to SQL Server, the following tasks walk you through some of the security tasks. These are not unique or specific to Linux, but it helps to give you an idea of areas to investigate further. In each example, a link is provided to the in-depth documentation for that area.

NOTE

The following examples use the **AdventureWorks2014** sample database. For instructions on how to obtain and install this sample database, see [Restore a SQL Server database from Windows to Linux](#).

Create a login and a database user

Grant others access to SQL Server by creating a login in the master database using the [CREATE LOGIN](#) statement. For example:

```
CREATE LOGIN Larry WITH PASSWORD = '*****';
```

NOTE

Always use a strong password in place of the asterisks in the previous command.

Logins can connect to SQL Server and have access (with limited permissions) to the master database. To connect to a user-database, a login needs a corresponding identity at the database level, called a database user. Users are specific to each database and must be separately created in each database to grant them access. The following example moves you into the AdventureWorks2014 database, and then uses the [CREATE USER](#) statement to create a user named Larry that is associated with the login named Larry. Though the login and the user are related (mapped to each other), they are different objects. The login is a server-level principle. The user is a database-level principal.

```
USE AdventureWorks2014;
GO
CREATE USER Larry;
GO
```

- A SQL Server administrator account can connect to any database and can create more logins and users in any database.
- When someone creates a database they become the database owner, which can connect to that database. Database owners can create more users.

Later you can authorize other logins to create more logins by granting them the [ALTER ANY LOGIN](#) permission. Inside a database, you can authorize other users to create more users by granting them the [ALTER ANY USER](#) permission. For example:

```
GRANT ALTER ANY LOGIN TO Larry;
```

```
GO
```

```
USE AdventureWorks2014;
```

```
GO
```

```
GRANT ALTER ANY USER TO Jerry;
```

```
GO
```

Now the login Larry can create more logins, and the user Jerry can create more users.

Granting access with least privileges

The first people to connect to a user-database will be the administrator and database owner accounts. However these users have all the the permissions available on the database. This is more permission than most users should have.

When you are just getting started, you can assign some general categories of permissions by using the built-in *fixed database roles*. For example, the `db_datareader` fixed database role can read all tables in the database, but make no changes. Grant membership in a fixed database role by using the `ALTER ROLE` statement. The following example add the user `Jerry` to the `db_datareader` fixed database role.

```
USE AdventureWorks2014;
```

```
GO
```

```
ALTER ROLE db_datareader ADD MEMBER Jerry;
```

For a list of the fixed database roles, see [Database-Level Roles](#).

Later, when you are ready to configure more precise access to your data (highly recommended), create your own user-defined database roles using `CREATE ROLE` statement. Then assign specific granular permissions to you custom roles.

For example, the following statements create a database role named `Sales`, grants the `Sales` group the ability to see, update, and delete rows from the `Orders` table, and then adds the user `Jerry` to the `Sales` role.

```
CREATE ROLE Sales;
GRANT SELECT ON Object::Sales TO Orders;
GRANT UPDATE ON Object::Sales TO Orders;
GRANT DELETE ON Object::Sales TO Orders;
ALTER ROLE Sales ADD MEMBER Jerry;
```

For more information about the permission system, see [Getting Started with Database Engine Permissions](#).

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 following steps 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:

```
USE AdventureWorks2014;
GO

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 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:

```
USE AdventureWorks2014;
GO
ALTER TABLE Person.EmailAddress
ALTER COLUMN EmailAddress
ADD MASKED WITH (FUNCTION = 'email()');
```

Create a new user `TestUser` with `SELECT` 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 EmailAddressID, EmailAddress FROM Person.EmailAddress;
REVERT;
```

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

EMAILADDRESSID	EMAILADDRESS
1	ken0@adventure-works.com

into

EMAILADDRESSID	EMAILADDRESS
1	kXXX@XXXX.com

Enable Transparent Data Encryption

One threat to your database is the risk that someone will steal the database files off of your hard-drive. This could happen with an intrusion that gets elevated access to your system, through the actions of a problem employee, or by theft of the computer containing the files (such as a laptop).

Transparent Data Encryption (TDE) encrypts the data files as they are stored on the hard drive. The master database of the SQL Server database engine has the encryption key, so that the database engine can manipulate the data. The database files cannot be read without access to the key. High-level administrators can manage, backup, and recreate the key, so the database can be moved, but only by selected people. When TDE is configured, the `tempdb` database is also automatically encrypted.

Since the Database Engine can read the data, Transparent Data Encryption does not protect against unauthorized access by administrators of the computer who can directly read memory, or access SQL Server through an administrator account.

Configure TDE

- Create a master key
- Create or obtain a certificate protected by the master key
- Create a database encryption key and protect it by the certificate
- Set the database to use encryption

Configuring TDE requires `CONTROL` permission on the master database and `CONTROL` permission on the user

database. Typically an administrator configures TDE.

The following example illustrates encrypting and decrypting the `AdventureWorks2014` database using a certificate installed on the server named `MyServerCert`.

```
USE master;
GO

CREATE MASTER KEY ENCRYPTION BY PASSWORD = '*****';
GO

CREATE CERTIFICATE MyServerCert WITH SUBJECT = 'My Database Encryption Key Certificate';
GO

USE AdventureWorks2014;
GO

CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_256
ENCRYPTION BY SERVER CERTIFICATE MyServerCert;
GO

ALTER DATABASE AdventureWorks2014
SET ENCRYPTION ON;
```

To remove TDE, execute `ALTER DATABASE AdventureWorks2014 SET ENCRYPTION OFF;`

The encryption and decryption operations are scheduled on background threads by SQL Server. You can view the status of these operations using the catalog views and dynamic management views in the list that appears later in this topic.

WARNING

Backup files of databases that have TDE enabled are also encrypted by using the database encryption key. As a result, when you restore these backups, the certificate protecting the database encryption key must be available. This means that in addition to backing up the database, you have to make sure that you maintain backups of the server certificates to prevent data loss. Data loss will result if the certificate is no longer available. For more information, see [SQL Server Certificates and Asymmetric Keys](#).

For more information about TDE, see [Transparent Data Encryption \(TDE\)](#).

Configure backup encryption

SQL Server has the ability to encrypt the data while creating a backup. By specifying the encryption algorithm and the encryptor (a certificate or asymmetric key) when creating a backup, you can create an encrypted backup file.

WARNING

It is very important to back up the certificate or asymmetric key, and preferably to a different location than the backup file it was used to encrypt. Without the certificate or asymmetric key, you cannot restore the backup, rendering the backup file unusable.

The following example creates a certificate, and then creates a backup protected by the certificate.

```
USE master;
GO
CREATE CERTIFICATE BackupEncryptCert
    WITH SUBJECT = 'Database backups';
GO
BACKUP DATABASE [AdventureWorks2014]
TO DISK = N'/var/opt/mssql/backups/AdventureWorks2014.bak'
WITH
    COMPRESSION,
    ENCRYPTION
    (
        ALGORITHM = AES_256,
        SERVER CERTIFICATE = BackupEncryptCert
    ),
    STATS = 10
GO
```

For more information, see [Backup Encryption](#).

Next steps

For more information about the security features of SQL Server, see [Security Center for SQL Server Database Engine and Azure SQL Database](#).

Active Directory authentication for SQL Server on Linux

2/23/2018 • 1 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article provides an overview of Active Directory (AD) authentication for SQL Server on Linux. AD authentication is also known as Integrated authentication in SQL Server.

AD authentication overview

AD authentication enables domain-joined clients on either Windows or Linux to authenticate to SQL Server using their domain credentials and the Kerberos protocol.

AD Authentication has the following advantages over SQL Server Authentication:

- Users authenticate via single sign-on, without being prompted for a password.
- By creating logins for AD groups, you can manage access and permissions in SQL Server using AD group memberships.
- Each user has a single identity across your organization, so you don't have to keep track of which SQL Server logins correspond to which people.
- AD enables you to enforce a centralized password policy across your organization.

Configuration steps

In order to use Active Directory authentication, you must have an AD Domain Controller (Windows) on your network.

The details for how to configure AD authentication are provided in the tutorial, [Tutorial: Use Active Directory authentication with SQL Server on Linux](#). The following list provides a summary with a link to each section in the tutorial:

1. [Join a SQL Server host to an Active Directory domain.](#)
2. [Create an AD user for SQL Server and set the ServicePrincipalName.](#)
3. [Configure the SQL Server service keytab.](#)
4. [Create AD-based SQL Server logins in Transact-SQL.](#)
5. [Connect to SQL Server using AD authentication.](#)

Known issues

- At this time, the only authentication method supported for database mirroring endpoint is CERTIFICATE. WINDOWS authentication method will be enabled in a future release.
- Third-party AD tools like Centrify, Powerbroker, and Vintela are not supported.

Next Steps

For more information on how to implement Active Directory authentication for SQL Server on Linux, see [Tutorial: Use Active Directory authentication with SQL Server on Linux](#).

Encrypting Connections to SQL Server on Linux

2/14/2018 • 3 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

SQL Server on Linux can use Transport Layer Security (TLS) to encrypt data that is transmitted across a network between a client application and an instance of SQL Server. SQL Server supports the same TLS protocols on both Windows and Linux: TLS 1.2, 1.1, and 1.0. However, the steps to configure TLS are specific to the operating system on which SQL Server is running.

Requirements for Certificates

Before getting started, you need to make sure your certificates follow these requirements:

- The current system time must be after the Valid from property of the certificate and before the Valid to property of the certificate.
- The certificate must be meant for server authentication. This requires the Enhanced Key Usage property of the certificate to specify Server Authentication (1.3.6.1.5.5.7.3.1).
- The certificate must be created by using theKeySpec option of AT_KEYEXCHANGE. Usually, the certificate's key usage property (KEY_USAGE) also includes key encipherment (CERT_KEY_ENCIPHERMENT_KEY_USAGE).
- The Subject property of the certificate must indicate that the common name (CN) is the same as the host name or fully qualified domain name (FQDN) of the server computer. Note: Wild Card Certificates are supported.

Overview

TLS is used to encrypt connections from a client application to SQL Server. When configured correctly, TLS provides both privacy and data integrity for communications between the client and the server. TLS connections can either be client initiated or server initiated.

Client Initiated Encryption

- **Generate certificate** (/CN should match your SQL Server host fully qualified domain name)

NOTE

For this example we use a Self-Signed Certificate, this should not be used for production scenarios. You should use CA certificates.

```
openssl req -x509 -nodes -newkey rsa:2048 -subj '/CN=mssql.contoso.com' -keyout mssql.key -out mssql.pem -days 365
sudo chown mssql:mssql mssql.pem mssql.key
sudo chmod 600 mssql.pem mssql.key
sudo mv mssql.pem /etc/ssl/certs/
sudo mv mssql.key /etc/ssl/private/
```

- **Configure SQL Server**

```

systemctl stop mssql-server
cat /var/opt/mssql/mssql.conf
sudo /opt/mssql/bin/mssql-conf set network.tlscert /etc/ssl/certs/mssqlfqn.pem
sudo /opt/mssql/bin/mssql-conf set network.tlskey /etc/ssl/private/mssqlfqn.key
sudo /opt/mssql/bin/mssql-conf set network.tlsprotocols 1.2
sudo /opt/mssql/bin/mssql-conf set network.forceencryption 0

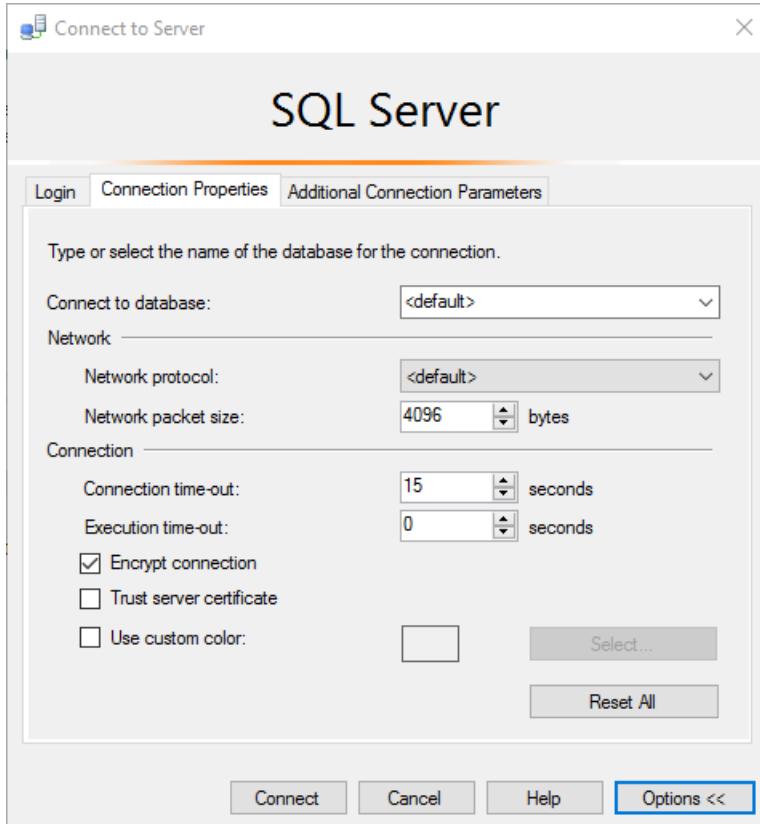
```

- **Register the certificate on your client machine (Windows, Linux, or macOS)**

- If you are using CA signed certificate, you have to copy the Certificate Authority (CA) certificate instead of the user certificate to the client machine.
- If you are using the self-signed certificate, just copy the .pem file to the following folders respective to distribution and execute the commands to enable them
 - **Ubuntu:** Copy cert to `/usr/share/ca-certificates/` rename extension to .crt use `dpkg-reconfigure ca-certificates` to enable it as system CA certificate.
 - **RHEL:** Copy cert to `/etc/pki/ca-trust/source/anchors/` use `update-ca-trust` to enable it as system CA certificate.
 - **SUSE:** Copy cert to `/usr/share/pki/trust/anchors/` use `update-ca-certificates` to enable it as system CA certificate.
 - **Windows:** Import the .pem file as a certificate under current user -> trusted root certification authorities -> certificates
 - **macOS:**
 - Copy the cert to `/usr/local/etc/openssl/certs`
 - Run the following command to get the hash value:
`/usr/local/Cellar/openssl/1.0.21/openssl x509 -hash -in mssql.pem -noout`
 - Rename the cert to value. For example: `mv mssql.pem dc2dd900.0`. Make sure dc2dd900.0 is in `/usr/local/etc/openssl/certs`

- **Example connection strings**

- **SQL Server Management Studio**



- **SQLCMD**

```
sqlcmd -S <sqlhostname> -N -U sa -P '<YourPassword>'
```

- **ADO.NET**

```
"Encrypt=True; TrustServerCertificate=False;"
```

- **ODBC**

```
"Encrypt=Yes; TrustServerCertificate=no;"
```

- **JDBC**

```
"encrypt=true; trustServerCertificate=false;"
```

Server Initiated Encryption

- **Generate certificate** (/CN should match your SQL Server host fully-qualified domain name)

```
openssl req -x509 -nodes -newkey rsa:2048 -subj '/CN=mssql.contoso.com' -keyout mssql.key -out mssql.pem -days 365
sudo chown mssql:mssql mssql.pem mssql.key
sudo chmod 600 mssql.pem mssql.key
sudo mv mssql.pem /etc/ssl/certs/
sudo mv mssql.key /etc/ssl/private/
```

- **Configure SQL Server**

```
systemctl stop mssql-server
cat /var/opt/mssql/mssql.conf
sudo /opt/mssql/bin/mssql-conf set network.tlscert /etc/ssl/certs/mssqlfqdn.pem
sudo /opt/mssql/bin/mssql-conf set network.tlskey /etc/ssl/private/mssqlfqdn.key
sudo /opt/mssql/bin/mssql-conf set network.tlsprotocols 1.2
sudo /opt/mssql/bin/mssql-conf set network.forceencryption 1
```

- **Example connection strings**

- **SQLCMD**

```
sqlcmd -S <sqlhostname> -U sa -P '<YourPassword>'
```

- **ADO.NET**

```
"Encrypt=False; TrustServerCertificate=False;"
```

- **ODBC**

```
"Encrypt=no; TrustServerCertificate=no;"
```

- **JDBC**

```
"encrypt=false; trustServerCertificate=false;"
```

NOTE

Set **TrustServerCertificate** to True if the client cannot connect to CA to validate the authenticity of the cert

Common connection errors

ERROR MESSAGE	FIX
The certificate chain was issued by an authority that is not trusted.	This error occurs when clients are unable to verify the signature on the certificate presented by SQL Server during the TLS handshake. Make sure the client trusts either the SQL Server certificate directly, or the CA which signed the SQL Server certificate.
The target principal name is incorrect.	Make sure that Common Name field on SQL Server's certificate matches the server name specified in the client's connection string.
An existing connection was forcibly closed by the remote host.	This error can occur when the client doesn't support the TLS protocol version required by SQL Server. For example, if SQL Server is configured to require TLS 1.2, make sure your clients also support the TLS 1.2 protocol.

Performance best practices and configuration guidelines for SQL Server 2017 on Linux

2/14/2018 • 4 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article provides best practices and recommendations to maximize performance for database applications that connect to SQL Server on Linux. These recommendations are specific to running on the Linux platform. All normal SQL Server recommendations, such as index design, still apply.

The following guidelines contain recommendations for configuring both SQL Server and the Linux operating system.

SQL Server configuration

It is recommended to perform the following configuration tasks after you install SQL Server on Linux to achieve best performance for your application.

Best practices

- **Use PROCESS AFFINITY for Node and/or CPUs**

It is recommended to use `ALTER SERVER CONFIGURATION` to set `PROCESS AFFINITY` for all the **NUMANODEs** and/or CPUs you are using for SQL Server (which is typically for all NODEs and CPUs) on a Linux Operating System. Processor affinity helps maintain efficient Linux and SQL Scheduling behavior. Using the **NUMANODE** option is the simplest method. Note, you should use **PROCESS AFFINITY** even if you have only a single NUMA Node on your computer. See the [ALTER SERVER CONFIGURATION](#) documentation for more information on how to set **PROCESS AFFINITY**.

- **Configure multiple tempdb data files**

Because a SQL Server on Linux installation does not offer an option to configure multiple tempdb files, we recommend that you consider creating multiple tempdb data files after installation. For more information, see the guidance in the article, [Recommendations to reduce allocation contention in SQL Server tempdb database](#).

Advanced Configuration

The following recommendations are optional configuration settings that you may elect to perform after installation of SQL Server on Linux. These choices are based on the requirements of your workload and configuration of your Linux Operating System.

- **Set a memory limit with mssql-conf**

In order to ensure there is enough free physical memory for the Linux Operating System, the SQL Server process uses only 80% of the physical RAM by default. For some systems which have a large amount of physical RAM, 20% might be a significant number. For example, on a system with 1 TB of RAM, the default setting would leave around 200 GB of RAM unused. In this situation, you might want to configure the memory limit to a higher value. See the documentation on the **mssql-conf** tool and the `memory.memorylimitmb` setting that controls the memory visible to SQL Server (in units of MB).

When changing this setting, be careful not to set this value too high. If you do not leave enough memory, you could experience problems with the Linux Operating System and other Linux applications.

Linux OS Configuration

Consider using the following Linux Operating System configuration settings to experience the best performance for a SQL Server Installation.

Kernel settings for high performance

These are the recommended Linux Operating System settings related to high performance and throughput for a SQL Server installation. See your Linux Operating System documentation for the process to configure these settings.

NOTE

For Red Hat Enterprise Linux (RHEL) users, the throughput-performance profile will configure these settings automatically (except for C-States).

The following table provides recommendations for CPU settings:

SETTING	VALUE	MORE INFORMATION
CPU frequency governor	performance	See the cpupower command
ENERGY_PERF_BIAS	performance	See the x86_energy_perf_policy command
min_perf_pct	100	See your documentation on intel p-state
C-States	C1 only	See your Linux or system documentation on how to ensure C-States is set to C1 only

The following table provides recommendations for disk settings:

SETTING	VALUE	MORE INFORMATION
disk readahead	4096	See the blockdev command
sysctl settings	kernel.sched_min_granularity_ns = 10000000 kernel.sched_wakeup_granularity_ns = 15000000 vm.dirty_ratio = 40 vm.dirty_background_ratio = 10 vm.swappiness=10	See the sysctl command

Kernel setting auto numa balancing for multi-node NUMA systems

If you install SQL Server on a multi-node **NUMA** systems, the following **kernel.numa_balancing** kernel setting is enabled by default. To allow SQL Server to operate at maximum efficiency on a **NUMA** system, disable auto numa balancing on a multi-node NUMA system:

```
sysctl -w kernel.numa_balancing=0
```

Kernel settings for Virtual Address Space

The default setting of **vm.max_map_count** (which is 65536) may not be high enough for a SQL Server installation.

Change this value (which is an upper limit) to 256K.

```
sysctl -w vm.max_map_count=262144
```

Disable last accessed date/time on file systems for SQL Server data and log files

Use the **noatime** attribute with any file system that is used to store SQL Server data and log files. Refer to your Linux documentation on how to set this attribute.

Leave Transparent Huge Pages (THP) enabled

Most Linux installations should have this option on by default. We recommend for the most consistent performance experience to leave this configuration option enabled.

swapfile

Ensure you have a properly configured swapfile to avoid any out of memory issues. Consult your Linux documentation for how to create and properly size a swapfile.

Virtual Machines and Dynamic Memory

If you are running SQL Server on Linux in a virtual machine, ensure you select options to fix the amount of memory reserved for the virtual machine. Do not use features like Hyper-V Dynamic Memory.

Next steps

To learn more about SQL Server features that improve performance, see [Get started with Performance features](#).

For more information about SQL Server on Linux, see [Overview of SQL Server on Linux](#).

Walkthrough for the performance features of SQL Server on Linux

2/14/2018 • 4 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

If you are a Linux user who is new to SQL Server, the following tasks walk you through some of the performance features. These are not unique or specific to Linux, but it helps to give you an idea of areas to investigate further. In each example, a link is provided to the depth documentation for that area.

NOTE

The following examples use the AdventureWorks sample database. For instructions on how to obtain and install this sample database, see [Restore a SQL Server database from Windows to Linux](#).

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.

1. Add a Columnstore index to the SalesOrderDetail table by executing the following Transact-SQL commands:

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

2. Execute the following query that uses 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
```

3. 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 following query to check the current compatibility level of AdventureWorks:

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

If necessary, update the level to 130:

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

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
GO
```

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
GO
ALTER DATABASE AdventureWorks ADD FILE (NAME='AdventureWorks_mod',
FILENAME='/var/opt/mssql/data/AdventureWorks_mod') TO FILEGROUP AdventureWorks_mod
GO
```

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
```

1. Insert 1,000,000 rows:

```
EXEC usp_InsertSampleCarts 1000000
```

2. Verify the rows have been inserted:

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

Learn More About In-Memory OLTP

For more information about In-Memory OLTP, see the following topics:

- [Quick Start 1: In-Memory OLTP Technologies for Faster Transact-SQL Performance](#)
- [Migrating to In-Memory OLTP](#)
- [Faster temp table and table variable by using memory optimization](#)
- [Monitor and Troubleshoot Memory Usage](#)
- [In-Memory OLTP \(In-Memory Optimization\)](#)

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;
```

Sample: Unattended SQL Server installation script for Red Hat Enterprise Linux

2/14/2018 • 4 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This sample Bash script installs SQL Server 2017 on Red Hat Enterprise Linux (RHEL) without interactive input. It provides examples of installing the database engine, the SQL Server command-line tools, SQL Server Agent, and performs post-install steps. You can optionally install full-text search and create an administrative user.

TIP

If you do not need an unattended installation script, the fastest way to install SQL Server is to follow the [quickstart for Red Hat](#). For other setup information, see [Installation guidance for SQL Server on Linux](#).

Prerequisites

- You need at least 2 GB of memory to run SQL Server on Linux.
- The file system must be **XFS** or **EXT4**. Other file systems, such as **BTRFS**, are unsupported.
- For other system requirements, see [System requirements for SQL Server on Linux](#).

Sample script

Save the sample script to a file and then to customize it, replace the variable values in the script. You can also set any of the scripting variables as environment variables, as long as you remove them from the script file.

```
#!/bin/bash -e

# Use the following variables to control your install:

# Password for the SA user (required)
MSSQL_SA_PASSWORD='<YourStrong!Passw0rd>'

# Product ID of the version of SQL server you're installing
# Must be evaluation, developer, express, web, standard, enterprise, or your 25 digit product key
# Defaults to developer
MSSQL_PID='evaluation'

# Install SQL Server Agent (recommended)
SQL_INSTALL_AGENT='y'

# Install SQL Server Full Text Search (optional)
# SQL_INSTALL_FULLTEXT='y'

# Create an additional user with sysadmin privileges (optional)
# SQL_INSTALL_USER='<Username>'
# SQL_INSTALL_USER_PASSWORD='<YourStrong!Passw0rd>'

if [ -z $MSSQL_SA_PASSWORD ]
then
    echo Environment variable MSSQL_SA_PASSWORD must be set for unattended install
    exit 1
fi
```

```

echo Adding Microsoft repositories...
sudo curl -o /etc/yum.repos.d/mssql-server.repo https://packages.microsoft.com/config/rhel/7/mssql-server-2017.repo
sudo curl -o /etc/yum.repos.d/msprod.repo https://packages.microsoft.com/config/rhel/7/prod.repo

echo Installing SQL Server...
sudo yum install -y mssql-server

echo Running mssql-conf setup...
sudo MSSQL_SA_PASSWORD=$MSSQL_SA_PASSWORD \
    MSSQL_PID=$MSSQL_PID \
    /opt/mssql/bin/mssql-conf -n setup accept-eula

echo Installing mssql-tools and unixODBC developer...
sudo ACCEPT_EULA=Y yum install -y mssql-tools unixODBC-devel

# Add SQL Server tools to the path by default:
echo Adding SQL Server tools to your path...
echo PATH="$PATH:/opt/mssql-tools/bin" >> ~/.bash_profile
echo 'export PATH="$PATH:/opt/mssql-tools/bin"' >> ~/.bashrc

# Optional SQL Server Agent installation:
if [ ! -z $SQL_INSTALL_AGENT ]
then
    echo Installing SQL Server Agent...
    sudo yum install -y mssql-server-agent
fi

# Optional SQL Server Full Text Search installation:
if [ ! -z $SQL_INSTALL_FULLTEXT ]
then
    echo Installing SQL Server Full-Text Search...
    sudo yum install -y mssql-server-fts
fi

# Configure firewall to allow TCP port 1433:
echo Configuring firewall to allow traffic on port 1433...
sudo firewall-cmd --zone=public --add-port=1433/tcp --permanent
sudo firewall-cmd --reload

# Example of setting post-installation configuration options
# Set trace flags 1204 and 1222 for deadlock tracing:
#echo Setting trace flags...
#sudo /opt/mssql/bin/mssql-conf traceflag 1204 1222 on

# Restart SQL Server after making configuration changes:
echo Restarting SQL Server...
sudo systemctl restart mssql-server

# Connect to server and get the version:
counter=1
errstatus=1
while [ $counter -le 5 ] && [ $errstatus = 1 ]
do
    echo Waiting for SQL Server to start...
    sleep 5s
    /opt/mssql-tools/bin/sqlcmd \
        -S localhost \
        -U SA \
        -P $MSSQL_SA_PASSWORD \
        -Q "SELECT @@VERSION" 2>/dev/null
    errstatus=$?
    ((counter++))
done

# Display error if connection failed:
if [ $errstatus = 1 ]
then
    echo Cannot connect to SQL Server, installation aborted

```

```

exit $errstatus
fi

# Optional new user creation:
if [ ! -z $SQL_INSTALL_USER ] && [ ! -z $SQL_INSTALL_USER_PASSWORD ]
then
    echo Creating user $SQL_INSTALL_USER
    /opt/mssql-tools/bin/sqlcmd \
        -S localhost \
        -U SA \
        -P $MSSQL_SA_PASSWORD \
        -Q "CREATE LOGIN [$SQL_INSTALL_USER] WITH PASSWORD=N'$SQL_INSTALL_USER_PASSWORD', DEFAULT_DATABASE=[master], CHECK_EXPIRATION=ON, CHECK_POLICY=ON; ALTER SERVER ROLE [sysadmin] ADD MEMBER [$SQL_INSTALL_USER]"
fi

echo Done!

```

Running the script

To run the script

1. Paste the sample into your favorite text editor and save it with a memorable name, like `install_sql.sh`.
2. Customize `MSSQL_SA_PASSWORD`, `MSSQL_PID`, and any of the other variables you'd like to change.
3. Mark the script as executable

```
chmod +x install_sql.sh
```

4. Run the script

```
./install_sql.sh
```

Understanding the script

The first thing the Bash script does is set a few variables. These can be either scripting variables, like the sample, or environment variables. The variable `MSSQL_SA_PASSWORD` is **required** by SQL Server installation, the others are custom variables created for the script. The sample script performs the following steps:

1. Import the public Microsoft GPG keys.
2. Register the Microsoft repositories for SQL Server and the command-line tools.
3. Update the local repositories
4. Install SQL Server
5. Configure SQL Server with the `MSSQL_SA_PASSWORD` and automatically accept the End-User License Agreement.
6. Automatically accept the End-User License Agreement for the SQL Server command-line tools, install them, and install the `unixodbc-dev` package.
7. Add the SQL Server command-line tools to the path for ease of use.
8. Install the SQL Server Agent if the scripting variable `SQL_INSTALL_AGENT` is set, on by default.
9. Optionally install SQL Server Full-Text search, if the variable `SQL_INSTALL_FULLTEXT` is set.

10. Unblock port 1433 for TCP on the system firewall, necessary to connect to SQL Server from another system.
11. Optionally set trace flags for deadlock tracing. (requires uncommenting the lines)
12. SQL Server is now installed, to make it operational, restart the process.
13. Verify that SQL Server is installed correctly, while hiding any error messages.
14. Create a new server administrator user if `SQL_INSTALL_USER` and `SQL_INSTALL_USER_PASSWORD` are both set.

Next steps

Simplify multiple unattended installs and create a stand-alone Bash script that sets the proper environment variables. You can remove any of the variables the sample script uses and put them in their own Bash script.

```
#!/bin/bash
export MSSQL_SA_PASSWORD='<YourStrong!Passw0rd>'
export MSSQL_PID='evaluation'
export SQL_INSTALL_AGENT='y'
export SQL_INSTALL_USER='<Username>'
export SQL_INSTALL_USER_PASSWORD='<YourStrong!Passw0rd>'
export SQL_INSTALL_AGENT='y'
```

Then run the Bash script as follows:

```
. ./my_script_name.sh
```

For more information about SQL Server on Linux, see [SQL Server on Linux overview](#).

Sample: Unattended SQL Server installation script for SUSE Linux Enterprise Server

2/14/2018 • 4 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This sample Bash script installs SQL Server 2017 on SUSE Linux Enterprise Server (SLES) v12 SP2 without interactive input. It provides examples of installing the database engine, the SQL Server command-line tools, SQL Server Agent, and performs post-install steps. You can optionally install full-text search and create an administrative user.

TIP

If you do not need an unattended installation script, the fastest way to install SQL Server is to follow the [quickstart for SLES](#). For other setup information, see [Installation guidance for SQL Server on Linux](#).

Prerequisites

- You need at least 2 GB of memory to run SQL Server on Linux.
- The file system must be **XFS** or **EXT4**. Other file systems, such as **BTRFS**, are unsupported.
- For other system requirements, see [System requirements for SQL Server on Linux](#).

IMPORTANT

SQL Server 2017 requires libsss_nss_idmap0, which is not provided by the default SLES repositories. You can install it from the SLES v12 SP2 SDK.

Sample script

```
#!/bin/bash -e

# Use the following variables to control your install:

# Password for the SA user (required)
MSSQL_SA_PASSWORD='<YourStrong!Passw0rd>'

# Product ID of the version of SQL server you're installing
# Must be evaluation, developer, express, web, standard, enterprise, or your 25 digit product key
# Defaults to developer
MSSQL_PID='evaluation'

# Install SQL Server Agent (recommended)
SQL_INSTALL_AGENT='y'

# Install SQL Server Full Text Search (optional)
# SQL_INSTALL_FULLTEXT='y'

# Create an additional user with sysadmin privileges (optional)
# SQL_INSTALL_USER='<Username>'
# SQL_INSTALL_USER_PASSWORD='<YourStrong!Passw0rd>'

if [ -z $MSSQL_SA_PASSWORD ]
```

```

then
echo Environment variable MSSQL_SA_PASSWORD must be set for unattended install
exit 1
fi

echo Adding Microsoft repositories...
sudo zypper addrepo -fc https://packages.microsoft.com/config/sles/12/mssql-server-2017.repo
sudo zypper addrepo -fc https://packages.microsoft.com/config/sles/12/prod.repo
sudo zypper --gpg-auto-import-keys refresh

#Add the SLES v12 SP2 SDK to obtain libsss_nss_idmap0
sudo SUSEConnect -p sle-sdk/12.2/x86_64

echo Installing SQL Server...
sudo zypper install -y mssql-server

echo Running mssql-conf setup...
sudo MSSQL_SA_PASSWORD=$MSSQL_SA_PASSWORD \
    MSSQL_PID=$MSSQL_PID \
    /opt/mssql/bin/mssql-conf -n setup accept-eula

echo Installing mssql-tools and unixODBC developer...
sudo ACCEPT_EULA=Y zypper install -y mssql-tools unixODBC-devel

# Add SQL Server tools to the path by default:
echo Adding SQL Server tools to your path...
echo PATH="$PATH:/opt/mssql-tools/bin" >> ~/.bash_profile
echo 'export PATH="$PATH:/opt/mssql-tools/bin"' >> ~/.bashrc

# Optional SQL Server Agent installation:
if [ ! -z $SQL_INSTALL_AGENT ]
then
    echo Installing SQL Server Agent...
    sudo zypper install -y mssql-server-agent
fi

# Optional SQL Server Full Text Search installation:
if [ ! -z $SQL_INSTALL_FULLTEXT ]
then
    echo Installing SQL Server Full-Text Search...
    sudo zypper install -y mssql-server-fts
fi

# Configure firewall to allow TCP port 1433:
echo Configuring SuSEfirewall2 to allow traffic on port 1433...
sudo SuSEfirewall2 open INT TCP 1433
sudo SuSEfirewall2 stop
sudo SuSEfirewall2 start

# Example of setting post-installation configuration options
# Set trace flags 1204 and 1222 for deadlock tracing:
# echo Setting trace flags...
# sudo /opt/mssql/bin/mssql-conf traceflag 1204 1222 on

# Restart SQL Server after making configuration changes:
echo Restarting SQL Server...
sudo systemctl restart mssql-server

# Connect to server and get the version:
counter=1
errstatus=1
while [ $counter -le 5 ] && [ $errstatus = 1 ]
do
    echo Waiting for SQL Server to start...
    sleep 5s
    /opt/mssql-tools/bin/sqlcmd \
        -S localhost \
        -U SA \
        -P $MSSQL_SA_PASSWORD \

```

```
-Q "SELECT @@VERSION" 2>/dev/null
errstatus=$?
((counter++))
done

# Display error if connection failed:
if [ $errstatus = 1 ]
then
    echo Cannot connect to SQL Server, installation aborted
    exit $errstatus
fi

# Optional new user creation:
if [ ! -z $SQL_INSTALL_USER ] && [ ! -z $SQL_INSTALL_USER_PASSWORD ]
then
    echo Creating user $SQL_INSTALL_USER
    /opt/mssql-tools/bin/sqlcmd \
        -S localhost \
        -U SA \
        -P $MSSQL_SA_PASSWORD \
        -Q "CREATE LOGIN [$SQL_INSTALL_USER] WITH PASSWORD=N'$SQL_INSTALL_USER_PASSWORD', DEFAULT_DATABASE=[master], CHECK_EXPIRATION=ON, CHECK_POLICY=ON; ALTER SERVER ROLE [sysadmin] ADD MEMBER [$SQL_INSTALL_USER]"
fi

echo Done!
```

Running the script

To run the script

- Paste the sample into your favorite text editor and save it with a memorable name, like `install_sql.sh`.
 - Customize `MSSQL_SA_PASSWORD`, `MSSQL_PID`, and any of the other variables you'd like to change.
 - Mark the script as executable

```
chmod +x install_sql.sh
```

- #### 4. Run the script

```
./install_sql.sh
```

Understanding the script

The first thing the Bash script does is set a few variables. These can be either scripting variables, like the sample, or environment variables. The variable `MSSQL_SA_PASSWORD` is **required** by SQL Server installation, the others are custom variables created for the script. The sample script performs the following steps:

1. Import the public Microsoft GPG keys.
 2. Register the Microsoft repositories for SQL Server and the command-line tools.
 3. Update the local repositories
 4. Install SQL Server
 5. Configure SQL Server with the `MSSQL_SA_PASSWORD` and automatically accept the End-User License Agreement.
 6. Automatically accept the End-User License Agreement for the SQL Server command-line tools, install them, and install the unixodbc-dev package.

7. Add the SQL Server command-line tools to the path for ease of use.
8. Install the SQL Server Agent if the scripting variable `SQL_INSTALL_AGENT` is set, on by default.
9. Optionally install SQL Server Full-Text search, if the variable `SQL_INSTALL_FULLTEXT` is set.
10. Unblock port 1433 for TCP on the system firewall, necessary to connect to SQL Server from another system.
11. Optionally set trace flags for deadlock tracing. (requires uncommenting the lines)
12. SQL Server is now installed, to make it operational, restart the process.
13. Verify that SQL Server is installed correctly, while hiding any error messages.
14. Create a new server administrator user if `SQL_INSTALL_USER` and `SQL_INSTALL_USER_PASSWORD` are both set.

Next steps

Simplify multiple unattended installs and create a stand-alone Bash script that sets the proper environment variables. You can remove any of the variables the sample script uses and put them in their own Bash script.

```
#!/bin/bash
export MSSQL_SA_PASSWORD='<YourStrong!Passw0rd>'
export MSSQL_PID='evaluation'
export SQL_INSTALL_AGENT='y'
export SQL_INSTALL_USER='<Username>'
export SQL_INSTALL_USER_PASSWORD='<YourStrong!Passw0rd>'
export SQL_INSTALL_AGENT='y'
```

Then run the Bash script as follows:

```
. ./my_script_name.sh
```

For more information about SQL Server on Linux, see [SQL Server on Linux overview](#).

Sample: Unattended SQL Server installation script for Ubuntu

2/14/2018 • 4 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This sample Bash script installs SQL Server 2017 on Ubuntu 16.04 without interactive input. It provides examples of installing the database engine, the SQL Server command-line tools, SQL Server Agent, and performs post-install steps. You can optionally install full-text search and create an administrative user.

TIP

If you do not need an unattended installation script, the fastest way to install SQL Server is to follow the [quickstart for Ubuntu](#). For other setup information, see [Installation guidance for SQL Server on Linux](#).

Prerequisites

- You need at least 2 GB of memory to run SQL Server on Linux.
- The file system must be **XFS** or **EXT4**. Other file systems, such as **BTRFS**, are unsupported.
- For other system requirements, see [System requirements for SQL Server on Linux](#).

Sample script

```
#!/bin/bash -e

# Use the following variables to control your install:

# Password for the SA user (required)
MSSQL_SA_PASSWORD='<YourStrong!Passw0rd>'

# Product ID of the version of SQL server you're installing
# Must be evaluation, developer, express, web, standard, enterprise, or your 25 digit product key
# Defaults to developer
MSSQL_PID='evaluation'

# Install SQL Server Agent (recommended)
SQL_INSTALL_AGENT='y'

# Install SQL Server Full Text Search (optional)
# SQL_INSTALL_FULLTEXT='y'

# Create an additional user with sysadmin privileges (optional)
# SQL_INSTALL_USER='<Username>'
# SQL_INSTALL_USER_PASSWORD='<YourStrong!Passw0rd>'

if [ -z $MSSQL_SA_PASSWORD ]
then
    echo Environment variable MSSQL_SA_PASSWORD must be set for unattended install
    exit 1
fi

echo Adding Microsoft repositories...
sudo curl https://packages.microsoft.com/keys/microsoft.asc | sudo apt-key add -
repoargs="$(curl https://packages.microsoft.com/config/ubuntu/16.04/mssql-server-2017.list)"
```

```

sudo add-apt-repository "${repoargs}"
repoargs=$(curl https://packages.microsoft.com/config/ubuntu/16.04/prod.list)"
sudo add-apt-repository "${repoargs}"

echo Running apt-get update -y...
sudo apt-get update -y

echo Installing SQL Server...
sudo apt-get install -y mssql-server

echo Running mssql-conf setup...
sudo MSSQL_SA_PASSWORD=$MSSQL_SA_PASSWORD \
    MSSQL_PID=$MSSQL_PID \
    /opt/mssql/bin/mssql-conf -n setup accept-eula

echo Installing mssql-tools and unixODBC developer...
sudo ACCEPT_EULA=Y apt-get install -y mssql-tools unixodbc-dev

# Add SQL Server tools to the path by default:
echo Adding SQL Server tools to your path...
echo PATH="$PATH:/opt/mssql-tools/bin" >> ~/.bash_profile
echo 'export PATH="$PATH:/opt/mssql-tools/bin"' >> ~/.bashrc

# Optional SQL Server Agent installation:
if [ ! -z $SQL_INSTALL_AGENT ]
then
    echo Installing SQL Server Agent...
    sudo apt-get install -y mssql-server-agent
fi

# Optional SQL Server Full Text Search installation:
if [ ! -z $SQL_INSTALL_FULLTEXT ]
then
    echo Installing SQL Server Full-Text Search...
    sudo apt-get install -y mssql-server-fts
fi

# Configure firewall to allow TCP port 1433:
echo Configuring UFW to allow traffic on port 1433...
sudo ufw allow 1433/tcp
sudo ufw reload

# Optional example of post-installation configuration.
# Trace flags 1204 and 1222 are for deadlock tracing.
# echo Setting trace flags...
# sudo /opt/mssql/bin/mssql-conf traceflag 1204 1222 on

# Restart SQL Server after installing:
echo Restarting SQL Server...
sudo systemctl restart mssql-server

# Connect to server and get the version:
counter=1
errstatus=1
while [ $counter -le 5 ] && [ $errstatus = 1 ]
do
    echo Waiting for SQL Server to start...
    sleep 3s
    /opt/mssql-tools/bin/sqlcmd \
        -S localhost \
        -U SA \
        -P $MSSQL_SA_PASSWORD \
        -Q "SELECT @@VERSION" 2>/dev/null
    errstatus=$?
    ((counter++))
done

# Display error if connection failed:
if [ $errstatus = 1 ]

```

```

then
    echo Cannot connect to SQL Server, installation aborted
    exit $errstatus
fi

# Optional new user creation:
if [ ! -z $SQL_INSTALL_USER ] && [ ! -z $SQL_INSTALL_USER_PASSWORD ]
then
    echo Creating user $SQL_INSTALL_USER
    /opt/mssql-tools/bin/sqlcmd \
        -S localhost \
        -U SA \
        -P $MSSQL_SA_PASSWORD \
        -Q "CREATE LOGIN [$SQL_INSTALL_USER] WITH PASSWORD=N'$SQL_INSTALL_USER_PASSWORD', DEFAULT_DATABASE=
[master], CHECK_EXPIRATION=ON, CHECK_POLICY=ON; ALTER SERVER ROLE [sysadmin] ADD MEMBER [$SQL_INSTALL_USER]"
fi

echo Done!

```

Running the script

To run the script

1. Paste the sample into your favorite text editor and save it with a memorable name, like `install_sql.sh`.
2. Customize `MSSQL_SA_PASSWORD`, `MSSQL_PID`, and any of the other variables you'd like to change.
3. Mark the script as executable

```
chmod +x install_sql.sh
```

4. Run the script

```
./install_sql.sh
```

Understanding the script

The first thing the Bash script does is set a few variables. These can be either scripting variables, like the sample, or environment variables. The variable `MSSQL_SA_PASSWORD` is **required** by SQL Server installation, the others are custom variables created for the script. The sample script performs the following steps:

1. Import the public Microsoft GPG keys.
2. Register the Microsoft repositories for SQL Server and the command-line tools.
3. Update the local repositories
4. Install SQL Server
5. Configure SQL Server with the `MSSQL_SA_PASSWORD` and automatically accept the End-User License Agreement.
6. Automatically accept the End-User License Agreement for the SQL Server command-line tools, install them, and install the unixodbc-dev package.
7. Add the SQL Server command-line tools to the path for ease of use.
8. Install the SQL Server Agent if the scripting variable `SQL_INSTALL_AGENT` is set, on by default.
9. Optionally install SQL Server Full-Text search, if the variable `SQL_INSTALL_FULLTEXT` is set.
10. Unblock port 1433 for TCP on the system firewall, necessary to connect to SQL Server from another system.

11. Optionally set trace flags for deadlock tracing. (requires uncommenting the lines)
12. SQL Server is now installed, to make it operational, restart the process.
13. Verify that SQL Server is installed correctly, while hiding any error messages.
14. Create a new server administrator user if `SQL_INSTALL_USER` and `SQL_INSTALL_USER_PASSWORD` are both set.

Next steps

Simplify multiple unattended installs and create a stand-alone Bash script that sets the proper environment variables. You can remove any of the variables the sample script uses and put them in their own Bash script.

```
#!/bin/bash
export MSSQL_SA_PASSWORD='<YourStrong!Passw0rd>'
export MSSQL_PID='evaluation'
export SQL_INSTALL_AGENT='y'
export SQL_INSTALL_USER='<Username>'
export SQL_INSTALL_USER_PASSWORD='<YourStrong!Passw0rd>'
export SQL_INSTALL_AGENT='y'
```

Then run the Bash script as follows:

```
. ./my_script_name.sh
```

For more information about SQL Server on Linux, see [SQL Server on Linux overview](#).

Troubleshoot SQL Server on Linux

2/23/2018 • 5 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Linux only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This document describes how to troubleshoot Microsoft SQL Server running on Linux or in a Docker container.

When troubleshooting SQL Server on Linux, remember to review the supported features and known limitations in the [SQL Server on Linux Release Notes](#).

TIP

For answers to frequently asked questions, see the [SQL Server on Linux FAQ](#).

Troubleshoot connection failures

If you are having difficulty connecting to your Linux SQL Server, there are a few things to check.

- Verify that the server name or IP address is reachable from your client machine.

TIP

To find the IP address of your Ubuntu machine, you can run the ifconfig command as in the following example:

```
sudo ifconfig eth0 | grep 'inet addr'
```

For Red Hat, you can use the ip addr as in the following example:

```
sudo ip addr show eth0 | grep "inet"
```

One exception to this technique relates to Azure VMs. For Azure VMs, [find the public IP for the VM in the Azure portal](#).

- If applicable, check that you have opened the SQL Server port (default 1433) on the firewall.
- For Azure VMs, check that you have a [network security group rule for the default SQL Server port](#).
- Verify that the user name and password do not contain any typos or extra spaces or incorrect casing.
- Try to explicitly set the protocol and port number with the server name like the following example:
tcp:servername,1433.
- Network connectivity issues can also cause connection errors and timeouts. After verifying your connection information and network connectivity, try the connection again.

Manage the SQL Server service

The following sections show how to start, stop, restart, and check the status of the SQL Server service.

Manage the mssql-server service in Red Hat Enterprise Linux (RHEL) and Ubuntu

Check the status of the SQL Server service using this command:

```
sudo systemctl status mssql-server
```

You can stop, start, or restart the SQL Server service as needed using the following commands:

```
sudo systemctl stop mssql-server
sudo systemctl start mssql-server
sudo systemctl restart mssql-server
```

Manage the execution of the mssql Docker container

You can get the status and container ID of the latest created SQL Server Docker container by running the following command (The ID is under the **CONTAINER ID** column):

```
sudo docker ps -l
```

You can stop or restart the SQL Server service as needed using the following commands:

```
sudo docker stop <container ID>
sudo docker restart <container ID>
```

TIP

For more troubleshooting tips for Docker, see [Troubleshooting SQL Server Docker containers](#).

Access the log files

The SQL Server engine logs to the /var/opt/mssql/log/errorlog file in both the Linux and Docker installations. You need to be in 'superuser' mode to browse this directory.

The installer logs here: /var/opt/mssql/setup-< time stamp representing time of install> You can browse the errorlog files with any UTF-16 compatible tool like 'vim' or 'cat' like this:

```
sudo cat errorlog
```

If you prefer, you can also convert the files to UTF-8 to read them with 'more' or 'less' with the following command:

```
sudo iconv -f UTF-16LE -t UTF-8 <errorlog> -o <output errorlog file>
```

Extended events

Extended events can be queried via a SQL command. More information about extended events can be found [here](#):

Crash dumps

Look for dumps in the log directory in Linux. Check under the /var/opt/mssql/log directory for Linux Core dumps (.tar.gz2 extension) or SQL minidumps (.mdmp extension)

For Core dumps

```
sudo ls /var/opt/mssql/log | grep .tar.gz2
```

For SQL dumps

```
sudo ls /var/opt/mssql/log | grep .mdmp
```

Start SQL Server in Minimal Configuration or in Single User Mode

Start SQL Server in Minimal Configuration Mode

This is useful if the setting of a configuration value (for example, over-committing memory) has prevented the server from starting.

```
sudo -u mssql /opt/mssql/bin/sqlservr -f
```

Start SQL Server in Single User Mode

Under certain circumstances, you may have to start an instance of SQL Server in single-user mode by using the startup option -m. For example, you may want to change server configuration options or recover a damaged master database or other system database. For example, you may want to change server configuration options or recover a damaged master database or other system database

Start SQL Server in Single User Mode

```
sudo -u mssql /opt/mssql/bin/sqlservr -m
```

Start SQL Server in Single User Mode with SQLCMD

```
sudo -u mssql /opt/mssql/bin/sqlservr -m SQLCMD
```

WARNING

Start SQL Server on Linux with the "mssql" user to prevent future startup issues. Example "sudo -u mssql /opt/mssql/bin/sqlservr [STARTUP OPTIONS]"

If you have accidentally started SQL Server with another user, you must change ownership of SQL Server database files back to the 'mssql' user prior to starting SQL Server with systemd. For example, to change ownership of all database files under /var/opt/mssql to the 'mssql' user, run the following command

```
chown -R mssql:mssql /var/opt/mssql/
```

Common issues

1. You cannot connect to your remote SQL Server instance.

See the troubleshooting section of the article, [Connect to SQL Server on Linux](#).

2. ERROR: Hostname must be 15 characters or less.

This is a known-issue that happens whenever the name of the machine that is trying to install the SQL Server Debian package is longer than 15 characters. There are currently no workarounds other than changing the name of the machine. One way to achieve this is by editing the hostname file and rebooting the machine. The following [website guide](#) explains this in detail.

3. Resetting the system administration (SA) password.

If you have forgotten the system administrator (SA) password or need to reset it for some other reason, follow these steps.

NOTE

The following steps stop the SQL Server service temporarily.

Log into the host terminal, run the following commands and follow the prompts to reset the SA password:

```
sudo systemctl stop mssql-server
sudo /opt/mssql/bin/mssql-conf setup
```

4. Using special characters in password.

If you use some characters in the SQL Server login password you may need to escape them when using them in the Linux terminal. You must escape the \$ anytime using the backslash character you are using it in a terminal command/shell script:

Does not work:

```
sudo sqlcmd -S myserver -U sa -P Test$$
```

Works:

```
sqlcmd -S myserver -U sa -P Test\$\\$
```

Resources: [Special characters Escaping](#)

ⓘ Get Help

- [UserVoice - Suggestion to improve SQL Server?](#)
- [Stack Overflow \(tag sql-server\) - ask SQL development questions](#)
- [Setup and Upgrade - MSDN Forum](#)
- [SQL Server Data Tools - MSDN forum](#)
- [Reddit - general discussion about SQL Server](#)
- [Microsoft SQL Server License Terms and Information](#)
- [Support options for business users](#)
- [Contact Microsoft](#)

SQL Server Documentation

2/28/2018 • 3 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

SQL Server is a central part of the Microsoft data platform. SQL Server is an industry leader in operational database management systems (ODBMS). This documentation helps you install, configure, and use SQL Server. The content includes end-to-end examples, code samples, and videos. For SQL Server language topics, see [Language Reference](#).

WHAT'S NEW	RELEASE NOTES
What's New in SQL Server 2017	SQL Server 2017 Release Notes
What's New in SQL Server 2016	SQL Server 2016 Release Notes
What's New in SQL Server 2014	SQL Server 2014 Release Notes

Try SQL Server!

- [!\[\]\(d2b87a7d2d43279ba25b7ab15dccec94_img.jpg\) Download SQL Server](#)
- [!\[\]\(6636c140b39b6097e2202625df86d50d_img.jpg\) Download SQL Server Management Studio \(SSMS\)](#)
- [!\[\]\(d43b564cfd13ca7f5aa8ee17cc46e7c5_img.jpg\) Download SQL Server Data Tools \(SSDT\)](#)
- [!\[\]\(0042ef4f8777137aab9cda2039456d29_img.jpg\) Get a Virtual Machine with SQL Server](#)

SQL Server Technologies



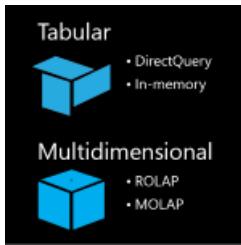
Database Engine

The Database Engine is the core service for storing, processing, and securing data. The Database Engine provides controlled access and rapid transaction processing to meet the requirements of the most demanding data consuming applications within your enterprise. The Database Engine also provides rich support for sustaining high availability.



Integration Services

Integration Services is a platform for building high performance data integration solutions, including packages that provide extract, transform, and load (ETL) processing for data warehousing.



Analysis Services

Analysis Services is an analytical data platform and toolset for personal, team, and corporate business intelligence. Servers and client designers support traditional OLAP solutions, new tabular modeling solutions, as well as self-service analytics and collaboration using Power Pivot, Excel, and a SharePoint Server environment. Analysis Services also includes Data Mining so that you can uncover the patterns and relationships hidden inside large volumes of data.



Reporting Services

Reporting Services delivers enterprise, Web-enabled reporting functionality. You can create reports that draw content from a variety of data sources, publish reports in various formats, and centrally manage security and subscriptions.



Machine Learning Services

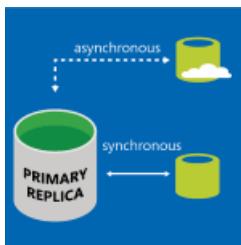
Microsoft Machine Learning Services supports integration of machine learning, using the popular R and Python languages, into enterprise workflows.

Machine Learning Services (In-Database) integrates R and Python with SQL Server, making it easy to build, retrain, and score models by calling stored procedures. Microsoft Machine Learning Server provides enterprise-scale support for R and Python, without requiring SQL Server.



Data Quality Services

SQL Server Data Quality Services (DQS) provides you with a knowledge-driven data cleansing solution. DQS enables you to build a knowledge base, and then use that knowledge base to perform data correction and deduplication on your data, using both computer-assisted and interactive means. You can use cloud-based reference data services, and you can build a data management solution that integrates DQS with SQL Server Integration Services and Master Data Services.



Replication

Replication is a set of technologies for copying and distributing data and database objects from one database to another, and then synchronizing between databases to maintain consistency. By using replication, you can distribute data to different locations and to remote or mobile users by means of local and wide area networks, dial-up connections, wireless connections, and the Internet.



Master Data Services

Master Data Services is the SQL Server solution for master data management. A solution built on Master Data Services helps ensure that reporting and analysis is based on the right information. Using Master Data Services, you create a central repository for your master data and maintain an auditable, securable record of that data as it changes over time.

Migrate and move data

- [Import and Export Data with the SQL Server Import and Export Wizard](#)
- [Microsoft Data Migration Assistant](#)
- [Migrate your SQL Server database to Azure SQL Database](#)

Earlier SQL Server versions

- [SQL Server Update Center - links and information for all supported versions](#)
- [SQL Server 2014 documentation](#)
- [SQL Server 2012 documentation](#)
- [SQL Server 2008 R2 documentation](#)
- [SQL Server 2008 documentation](#)
- [SQL Server 2005 archived documentation](#)

Samples

- [Wide World Importers sample database](#)
- [AdventureWorks sample databases and scripts for SQL Server 2016](#)
- [SQL Server samples on GitHub](#)

ⓘ Get Help

- [UserVoice - Suggestion to improve SQL Server?](#)
- [Stack Overflow \(tag sql-server\) - ask SQL development questions](#)
- [Setup and Upgrade - MSDN Forum](#)
- [SQL Server Data Tools - MSDN forum](#)
- [Reddit - general discussion about SQL Server](#)
- [Microsoft SQL Server License Terms and Information](#)
- [Support options for business users](#)
- [Contact Microsoft](#)

SQL Server monitoring partners

12/4/2017 • 2 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

To monitor your SQL Server services, choose from a wide variety of industry-leading tools. This article highlights Microsoft partner companies with monitoring solutions supporting Microsoft SQL Server.

Microsoft monitoring partners

PARTNER	DESCRIPTION	LINKS
	<p>IDERA SQL Diagnostic Manager is a powerful performance monitoring and diagnostics solution that proactively alerts administrators to health, performance, and availability problems within SQL Server environments all from a central console.</p> <p>System requirements</p>	Website Twitter Video
	<p>Quest Spotlight on SQL Server Enterprise - Ensure peak performance around the clock with unmatched monitoring, diagnosis and optimization of SQL Server environments. Spotlight provides access to the details you need for optimal fitness of your SQL Server databases. Get intuitive overviews of health and performance, automated alerts and actions, and mobile device support.</p>	Marketplace Website Video
	<p>Redgate SQL Monitor from Redgate is a SQL Server monitoring tool that helps teams looking after SQL Server be more proactive. Not only does SQL Monitor alert you to current issues, it gives you the information you need to stop them happening in the future. Ideally suited to large SQL Server estates, SQL Monitor makes sure you always have the answer to questions about performance.</p>	Website Twitter LinkedIn Video

Partner	Description	Links
	<p>SentryOne</p> <p>SentryOne solutions empower Microsoft data professionals to achieve breakthrough performance across physical, virtual, and cloud environments. With SentryOne, data professionals can consolidate their tool sets, reduce infrastructure costs, and increase database speed and efficiency for peak performance across their Microsoft data platform environments, regardless of size or complexity.</p>	Website Datasheet Twitter LinkedIn
	<p>Solarwinds</p> <p>Database Performance Analyzer provides visibility across application requests, SQL statements, database resources, host/OS, network, virtualization, and storage performance. DPA incorporates wait-time analysis so the focus is not only on health, but on the speed at which the database responds to application requests. DPA provides full coverage of your databases, no matter how it is deployed. Physical, virtual, cloud, or DBaaS, we've got you covered in a single pane of glass.</p>	Marketplace Website Datasheet LinkedIn Video

Next steps

To learn more about some of our other partners, see [High availability and disaster recovery partners](#), [management partners](#), and [development partners](#).

SQL Server high availability and disaster recovery partners

2/14/2018 • 3 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

To provide high availability and disaster recovery for your SQL Server services, choose from a wide variety of industry-leading tools. This article highlights Microsoft partner companies with high availability and disaster recovery solutions supporting Microsoft SQL Server.

Our high availability and disaster recovery partners

PARTNER	DESCRIPTION	LINKS
	<p>Azure Site Recovery Site Recovery replicates workloads running on virtual machines or physical servers so that they remain available in a secondary location if the primary site isn't available. You can replicate and fail over SQL Server virtual machines from on-premises data center to Azure or to another on-premises data center or from one Azure data centers to another Azure data center.</p> <p>Enterprise and Standard editions of SQL Server 2008 R2- SQL Server 2016</p>	Website Marketplace Datasheet Twitter Video
	<p>DH2i DxEnterprise is Smart Availability software for Windows, Linux & Docker that helps you achieve the nearest-to-zero planned and unplanned downtime, unlocks huge cost savings, drastically simplifies management and gets you both physical and logical consolidation.</p> <p>SQL Server 2005+, Windows Server 2008R2+, Ubuntu 16+, RHEL 7+, CentOS 7+</p>	Website Datasheet Twitter Video

PARTNER	DESCRIPTION	LINKS
 Hewlett Packard Enterprise	<p>HPE Serviceguard Protect your critical SQL Server 2017 workloads on Linux ® from unplanned and planned downtime through a multitude of infrastructure and applications faults across physical and virtual environments over any distance with HPE Serviceguard for Linux (SGLX). HPE SGLX A.12.20.00 and later offers context sensitive monitoring and recovery options for Failover Cluster Instance and Always On Availability Groups SQL Server workloads. Maximize uptime with HPE SGLX without compromising data integrity and performance.</p> <p>SQL Server 2017 on Linux - RedHat 7.3, 7.4, SUSE 12 SP2, SP3</p>	Website Datasheet Download Evaluation Blog Twitter
	<p>IDERA SQL Safe Backup is a high-performance backup and recovery solution for SQL Server that saves money by reducing database backup time and backup file size, and by providing instant read and write access to databases within backup files.</p> <p>Microsoft SQL Server: 2005 SP1 or later, 2008, 2008 R2, 2012, 2014, 2016; all editions</p>	Website
	<p>NEC ExpressCluster is a comprehensive and fully automated high-availability and disaster recovery solution against all major failures including hardware, software, network and site failures for SQL Server and associated applications running on physical or virtual machines in on-premises or cloud environments.</p> <p>Microsoft SQL Server: 2005 or later; all editions</p>	Website Datasheet Video Download

PARTNER	DESCRIPTION	LINKS
	<p>Portworx Portworx is the solution for stateful containers running in production. With Portworx, users can manage any database or stateful service on any infrastructure using any container scheduler, including Kubernetes, Mesosphere DC/OS, and Docker Swarm. Portworx solves the five most common problems DevOps teams encounter when running containerized databases and other stateful services in production: persistence, high availability, data automation, support for multiple data stores and infrastructure, and security.</p> <p>SQL Server 2017 on Docker</p>	Website Documentation Video
	<p>Veeam Veeam Backup & Replication is a powerful, easy-to-use and affordable backup and availability solution. It provides fast, flexible and reliable recovery of virtualized applications and data, bringing VM (virtual machine) backup and replication together in a single software solution. Veeam Backup & Replication delivers award-winning support for VMware vSphere and Microsoft Hyper-V virtual environments.</p> <p>SQL Server 2005 SP4 – SQL Server 2016 on Windows</p>	Website Datasheet Twitter Video

Next steps

To learn more about some of our other partners, see [monitoring](#), [management partners](#), and [development partners](#).

SQL Server managing partners

12/4/2017 • 1 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

To manage your SQL Server services, choose from a wide variety of industry-leading tools. This article highlights Microsoft partner companies with management solutions supporting Microsoft SQL Server.

Our management partners

PARTNER	DESCRIPTION	LINKS
	<p>IDERA SQL Compliance Manager is a comprehensive auditing solution that displays who did what, when and how on SQL Servers to help ensure compliance with regulatory and data security requirements.</p> <p>System requirements</p>	Website Twitter Video

Next steps

To learn more about some of our other partners, see [High availability, and disaster recovery partners](#), [management partners](#), and [development partners](#).

SQL Server development partners

12/5/2017 • 1 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

For support developing SQL Server database solutions, choose from a wide variety of industry-leading tools. This article highlights Microsoft partner companies with development solutions for Microsoft SQL Server.

Our development partners

PARTNER	DESCRIPTION	LINKS
	<p>IDERA Rapid SQL is an intelligent SQL integrated development environment empowering database developers and database administrators to create high-performing SQL code on all major database platforms including SQL Server from a single interface.</p> <p>System requirements</p>	Website Twitter Video
	<p>Click2Cloud Inc. Click2Cloud Inc. enables developer community with the flexibility in choosing programming languages, frameworks and services with use of proper toolset. Using Click2Cloud's toolkit, developers can create 'SQL on Linux' containers and attach it to an application, while still enabling the secure, multi-tenant architecture. The containers can be independently deployed on any cloud based container platform.</p>	Marketplace Website Twitter Video

Next steps

To learn more about some of our other partners, see [High availability, and disaster recovery partners](#), [management partners](#), and [monitoring partners](#).