Installing SQL Server on Linux

- Overview
- Downloading and installing SQL Server 2017 and tools
- Connecting to SQL Server locally and remotely
- Post-installation configuration
- Updating SQL Server 2017 and similar tasks

Assumptions

- **This deck covers installation using Red Hat Enterprise Linux (RHEL)**
  - Commands are slightly different for Ubuntu, SUSE, and Docker
  - See https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-setup for details
- **You should have an SSH tool installed locally**
  - PuTTY from https://www.putty.org/
  - MobaXterm from https://mobaxterm.mobatek.net/

System Requirements

- **SQL Server 2017 has the following system requirements on Linux:**
  - Memory: 2 GB
  - File system: XFS or EXT4 (BTRFS is unsupported)
    - XFS allows larger files and is faster when dealing with large files
  - Disk space: 6 GB
  - Processor speed: 2 GHz
  - Processor cores: 2 cores
  - Processor type: x64-compatible only
- **These are minimum requirements! More memory and more cores are better**

Repository Config File

- **When you install SQL Server on RHEL, you must configure a repository (repo)**
  - A repository is used to acquire a package that can then be installed using the yum tool
- **Three main repos**
  - mssql-server-2017 (SQL Server 2017 with latest Cumulative Update)
  - mssql-server-2017-gdr (SQL Server 2017 GDR for critical updates only)
  - mssql-server-preview(SQL Server 2019 Preview and Release Candidates)
- **mssql-server-2017 is the usual choice**

Configure RHEL Repositories

- **Verify whether you have already registered a SQL Server repo**
  - sudo ls /etc/yum.repos.d
- **Delete any old repo**
  - sudo rm -rf /etc/yum.repos.d/mssql-server.repo
- **Download and configure a new repo**
  - sudo curl -o /etc/yum.repos.d/mssql-server.repo https://packages.microsoft.com/config/rhel/7/mssql-server-2017.repo

Command-line installation

- **Run this command to install SQL Server 2017 on RHEL**
  - sudo yum install mssql-server
  - Creates a RHEL user and group called mssql and registers the mssql-server service as a serviced service so that SQL Server starts when RHEL starts
- **Run mssql-conf setup to choose an Edition and set the 'sa' password**
  - sudo /opt/mssql/bin/mssql-conf setup
  - The 'sa' account is the equivalent of combining the 'SYS' and 'SYSTEM' Oracle users
- **After installation is done, verify that the SQL Server service is running along with configuration**
  - sudo systemctl status mssql-server

Install Command-line Tools

- **SQL Server command-line tools can be used to run local commands**
  - E.g., sqlcmd for local connections to SQL Server; bcp for bulk import/export
- **Download the repo**
  - sudo curl -o /etc/yum.repos.d/msprod.repo https://packages.microsoft.com/config/rhel/7/prod.repo
- **Install mssql-tools**
  - sudo yum install mssql-tools unixODBC-devel
- **Install the mssql-cli command-line query tool, more interactive than sqlcmd**
  - sudo yum install mssql-cli
- **Add the tools pathname to your PATH**
  - echo 'export PATH="$PATH:/opt/mssql-tools/bin"' >> ~/.bash_profile
  - echo 'export PATH="$PATH:/opt/mssql-tools/bin"' >> ~/.bashrc
  - source ~/.bashrc

Configuring for Remote Connections

- **SQL Server uses port 1433 for remote connections by default**
  - You may want to change that to something less well known, e.g. 51433
- **To change the SQL Server listener port**
  - sudo /opt/mssql/bin/mssql-conf set network.tcpport 51433
  - sudo systemctl restart mssql-server
- **Configure RHEL firewall to allow whatever port you choose**
  - sudo firewall-cmd --zone=public --add-port=51433/tcp --permanent
  - sudo firewall-cmd --reload

Connect Locally

- **You can use sqlcmd to connect locally, using the following parameters:**
  - -S localhost,<port> -U sa
  - (Optionally) -P <password>
  - (Optionally) -Q 'query text to run'
- **You will see a sqlcmd command prompt if you connected successfully**
- **You can check your SQL Server build with sqlcmd as follows**
  - sqlcmd -S localhost, port -U sa -P <YourPassword> -Q 'select @@version'

Azure Data Studio (ADS)

- **Cross-platform management tool for SQL Server, Azure SQL Database, and Azure SQL Data Warehouse**
  - Runs on Windows, MacOS and Linux
- **Download and install Azure Data Studio**
  - https://docs.microsoft.com/sql/azure-data-studio/download
- **Recommended system requirements:**
  - Four CPU cores and 8 GB of RAM

SQL Server Management Studio (SSMS)

- **Windows-based management tool, more familiar to SQL Server users**
  - Supported on x64 Windows 7 SP1, Windows Server 2008 R2 and later
- **Lets you manage SQL Server 2017 on Linux instances from a Windows client**
  - Just like you do with SQL Server on Windows
- **Download SQL Server Management Studio**
  - https://docs.microsoft.com/sql/ssms/

Connect remotely using Azure Data Studio

- In the Servers tab, click the New Connection icon
- Enter the name/IP address followed by a comma and the port number
- Specify SQL Login for the authentication type, sa, and the password
- Right-click the server name and select Manage
- In the dashboard that appears, you can select the New Query button

Post-install Configuration with mssql-conf

- **You can configure some SQL Server options using the mssql-conf script**
  - sudo /opt/mssql/bin/mssql-conf set <option details>
  - sudo /opt/mssql/bin/mssql-conf unset <option details>
- **The various options are documented here**
  - https://docs.microsoft.com/sql/linux/sql-server-linux-configure-mssql-conf
- **Make sure to make changes in all nodes in an AG or shared-disk cluster**
- **You must restart SQL Server for changes to take effect**
  - sudo systemctl restart mssql-server
  - Or in a shared-disk cluster configuration, take the resource offline then online

Manually View/Edit the mssql-conf file

- **You can view current settings with this command**
  - sudo cat /var/opt/mssql/mssql.conf
- **Manually edit the mssql-conf file (be careful!) using the following**
  - sudo nano /var/opt/mssql/mssql.conf
  - (Install the GNU nano editor using: sudo yum install nano)
- **You must restart SQL Server for any changes to go into effect**
  - sudo systemctl restart mssql-server

Examples of Using mssql-conf

- **Set a 16GB memory limit with mssql-conf**
  - sudo /opt/mssql/bin/mssql-conf set memory.memorylimitmb 16384
  - By default this is set to 80% of system memory, which may be too low on some systems
- **Change the TCP port that SQL Server listens to for connections**
  - sudo /opt/mssql/bin/mssql-conf set network.tcpport <new_tcp_port>
  - Set to make system less vulnerable to 'port scanning ' security attacks that use brute-force password guessing
- **If someone forgets the sa password, it can be reset, but SQL Server has to be stopped first**
  - sudo systemctl stop mssql-server
  - sudo /opt/mssql/bin/mssql-conf set-sa-password
  - sudo systemctl start mssql-server
- **To change the locale to a supported language ID**
  - sudo /opt/mssql/bin/mssql-conf set language.lcid

Setting Trace Flags

- **Enable/disabling two startup trace flags**
  - sudo /opt/mssql/bin/mssql-conf traceflag 460 3226 on
  - sudo /opt/mssql/bin/mssql-conf traceflag 460 3226 off
  - sudo systemctl restart mssql-server
- **3226 stops SQL Server flooding the error log with backup success messages**
- **460 (in 2017 CU12+) enables better diagnostics when string or binary data would be truncated during an insert**
- **Modern versions of SQL Server require far fewer trace flags than older versions**
- **More on trace flags at https://docs.microsoft.com//sql/t-sql/database-console-commands/dbcc-traceon-trace-flags-transact-sql**

Setting the Default Data and Log Directories

- **Set the default data and log directories**
  - sudo /opt/mssql/bin/mssql-conf set filelocation.defaultdatadir /usr/sqldata
  - sudo /opt/mssql/bin/mssql-conf set filelocation.defaultlogdir /usr/sqllog
- **This provides separation from system databases and binaries**
- **Specific locations will depend on storage capacity and capabilities, plus manageability and availability requirements of the databases and workload**

Aside: Adding Drives for SQL Server

- **For RHEL, new drives need to be:**
  - Partitioned using: sudo fdisk /dev/<device>
  - Formatted using: sudo mkfs –t xfs /dev/<partition>
  - Mounted using: sudo mount /dev/<partition> <directory>
  - Added to /etc/fstab so they are mounted at system start time
- **And then for SQL Server, access must be granted using:**
  - sudo chown mssql <directory>
  - sudo chgrp mssql <directory>

Post-install: Multiple tempdb Data Files

- **Under many workloads, tempdb benefits from having multiple data files**
  - This helps alleviate contention for allocation data structures in memory
- **Done during install on Windows, but Linux only has a single tempdb data file**
- **Start with 4 to 8 tempdb data files, all the same size and same autogrowth**
  - For allocation contention, the files do not have to be on separate drives
    - This is because SQL Server just needs to have the allocation data structures for the different files in memory, to allow efficient round-robin allocation over multiple files
- **Use ALTER DATABASE tempdb ADD FILE to add files**

Post-install: Process Affinity

- **Testing has shown that SQL Server runs with best performance on Linux when the process affinity is set, even if all CPUs/NUMA nodes will be available**
- **Can be set by CPU or NUMA node, for example:**
  - ALTER SERVER CONFIGURATION SET PROCESS AFFINITY NUMANODE = 0 TO 3
    - To use all NUMA nodes on a system with four NUMA nodes
  - ALTER SERVER CONFIGURATION SET PROCESS AFFINITY CPU = 3 to 7
    - To disallow the use of CPUs 0, 1, 2 on a system with 8 CPUs
- **No restart is required**

Post-install: Linux Options

- **There are number of Linux kernel options that should be set for optimal performance, around:**
  - CPU settings and power management
  - Disk readahead and task scheduling
  - Memory and NUMA
- **These are documented at**
  - https://docs.microsoft.com/sql/linux/sql-server-linux-performance-best-practices

Post-install: Alerts

- **SQL Server can automatically send alerts on error conditions**
- **Steps required:**
  - Install SQL Server Agent
    - https://docs.microsoft.com/sql/linux/sql-server-linux-setup-sql-agent
  - Configure Database Mail
    - https://docs.microsoft.com/sql/linux/sql-server-linux-db-mail-sql-agent
  - Configure alerts
    - Script from https://www.sqlskills.com/blogs/glenn/creating-sql-server-agent-alerts-for-critical-errors/

Using SSMS/ADS to Configure SQL Server

- **You can use the SSMS UI to configure SQL Server on Linux**
  - This works just the same as it does for SQL Server on Windows
- **You can use T-SQL commands in SSMS or ADS to configure SQL Server**
  - This works just the same as it does for SQL Server on Windows
  - You have more control with T-SQL commands, since not all settings are exposed in the UI

Updating SQL Server 2017

- **To update the mssql-server package to the latest release of the configured repository**
  - sudo yum update mssql-server
- **This works the same way on all repositories (GDR and CU)**
  - You must have internet connectivity to use this method
- **The CU repository (mssql-server-2017) is the preferred option in most cases**
  - You get more frequent bug fixes and also new features and improvements
  - GDR repository only gets critical security updates
- **You can find out about SQL Server 2017 Cumulative Update builds here:**
  - https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-release-notes

Downgrading SQL Server 2017

- **Get the build number you want to go back to from the release notes**
- **Run this command**
  - sudo yum downgrade mssql-server-<version_number>.x86_64
- **For example, to go back to SQL Server 2017 CU9**
  - sudo yum downgrade mssql-server-14.0.3030.27.x86_64
- **Be careful of behavior changes in earlier builds!**
  - E.g., before CU6 there was no force-flush to guarantee data durability in all cases

Uninstalling SQL Server 2017

- **Run this command**
  - sudo yum remove mssql-server
- **This will not remove your database files**
- **You can run this command to remove database files**
  - sudo rm -rf /var/opt/mssql/
  - Make sure to do the same thing for any other data/log directories used

Summary

- Download and installation of SQL Server 2017 on RHEL is very simple
- Tools exist for easy local and remote connections
- Post-installation configuration should be done for best practices
- The lab that accompanies this module will guide you through all these steps, including provisioning a RHEL Azure VM to install SQL Server on

Configuring HA on Linux

- Failover Cluster Instance on Linux
- Availability Groups on Linux
- How they work and how to configure them

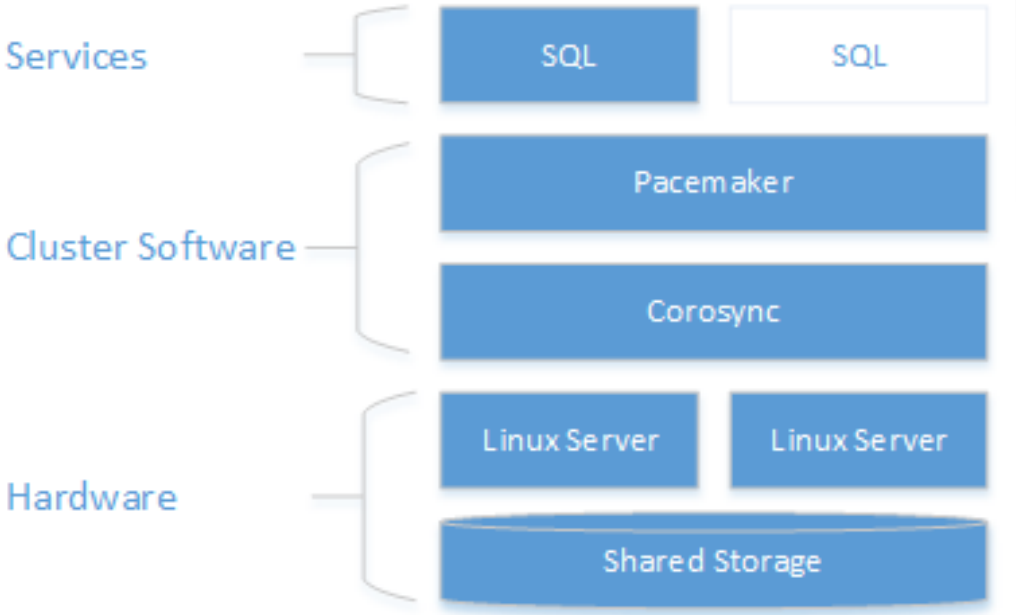
High Availability Options for SQL Server

- **Failover cluster instance**
  - The SQL Server instance moves to another node if the current node is unavailable
  - Shared storage
- **Availability Group**
  - The database (or group of databases) becomes available on another node if unavailable on the current node
  - Allows multiple copies of databases, with synchronous or asynchronous updates from the primary
  - Allows offloading of backups and read-only query workload to a copy
  - No shared storage

Failover Clustered Instances on Linux

- **Linux FCI Limitations**
  - Only 1 instance: SQL instance active on only one node at a time regardless of node count
  - Virtual Network Name not in Cluster: DNS A Record assignment to IP
- **Install and configure SQL Server on each cluster node**
  - Same as standalone instance installation and configuration
  - FCI configuration performed after instance install
- **Shared Storage**
  - iSCSI
  - NFS
  - tempdb not allowed on local storage under Linux FCI
  - Must be formatted with XFS or EXT4

SQL FCI on Linux Topology



Pacemaker/Corosync Configuration

- **RHEL requires a subscription for HA Add-on**
- **Firewall ports must be open for high-availability service**
  - TCP ports 2224, 3121, 21064 and UDP port 5405
- **Create /etc/hosts file entries for all nodes pointing to other nodes**
- **Designate a primary node for configuration purposes**
  - Create SQL Server login for Pacemaker with VIEW SERVER STATE, ALTER ANY LINKED SERVER, and set up admin fixed server role
    - Stored in passwd file under /var/opt/mssql/secrets/passwd
    - Set cluster password (same on all nodes)
- **Enable pcsd and Pacemaker services**
- **Install SQL Server HA resource agent**
  - Disabling fencing agents for non-VMware/KVM hypervisors
  - Configurations with no fencing agent not recommended for production

Configuring a SQL Server FCI

- **Configure shared storage on all nodes**
  - Copy database files to temporary location on primary node
  - Configure NFS client on all cluster nodes
    - /var/opt/mssql/data should be the mounted path for system databases
  - Move system database files to the shared storage at /var/opt/mssql/data
    - Set permissions with chown/chgrp for the mssql account
  - Backup Service Master Key and restore to each node so each local mssql account can decrypt the key
- **Start SQL Server one node at a time to verify access to shared storage**

Forming the SQL Server FCI

- **Configuring cluster resources**
  - SQL Resource Name
  - Floating IP Address Name: Resource Name for cluster from DNS
  - SQL FCI IP Address
  - File System Resource Name
    - Shared storage path
    - Local path mounted on share
    - File share type
- **Push the cluster configuration to all nodes and SQL Server should start on one node**
  - Verify this with sudo pcs status

Failing Over Services Manually

- Use the sudo pcs resource move command to create a constraint forcing the resource to start on the target node
- After SQL Server starts on the new target node, use the sudo pcs resource clear command to remove the constraint and allow automatic failover again

Rolling Updates on Linux

- **Create a location constraint using sudo pcs constraint location <resource> avoids <node> for the node to be updated**
- **Update the SQL Server packages from the source repository using sudo yum update**
  - mssql-server
  - mssql-server-agent (SQL Server Agent)
  - mssql-server-fts (Full-Text Search)
  - mssql-server-ha
- **Remove the location constraint using sudo pcs constraint remove**
- **Failover to the updated node and repeat the steps until all nodes are patched**

Adding a Node to Linux SQL Server FCI

- Install SQL Server Services
- Create /etc/hosts file entries for all nodes pointing to other nodes
- Stop the SQL Server Service
- Install NFS utilities to connect to storage - open firewall rules
  - Mount the storage to the correct database files directory path
- Create cluster password in /var/opt/mssql/secrets/passwd for Pacemaker login and password
- Install Pacemaker and open Firewall ports
- Set the hacluster password and enable/start pscd and Pacemaker
- Install mssql-server-ha add-on for sql server
- Authenticate and add the new node to the cluster from an existing node
- Don't forget the Service Master Key backup/restore step

AG Configuration General

- **Install and configure SQL Server on each cluster node**
  - Same as standalone instance installation and configuration
- **Enable Always On Availability Groups with mssql-conf**
  - /opt/mssql/bin/mssql-conf set hadr.hadrenabled 1
  - Requires restart of SQL Services
- **Create Master Key and Certificate for Endpoint security between replicas on primary server**
  - Specify certificate EXPIRY_DATE in future to avoid expiration
  - Backup certificate and private key from primary replica
  - Create certificate with private key on secondary replicas

AG Configuration General (2)

- Create endpoint on each replica FOR DATABASE_MIRRORING specifying AUTHENTICATION=CERTIFICATE <certificate_name>
- Alter endpoint to STATE=STARTED
- Open firewall rules for endpoint TCP port (default is 5022 in SSMS)
- Databases must use the FULL recovery model and have at least one FULL database backup taken prior to being added to an availability group
- Enable AlwaysOn_Health event session with STARTUP_STATE=ON

Creating a Read-scale Availability Group

- **Does not require Corosync/Pacemaker configuration**
- **Availability Group created with CLUSTER_TYPE=NONE on primary**
  - CREATE AVAILABILITY GROUP DDL command
- **Join each secondary replica to the Availability Group**
  - ALTER AVAILABILITY GROUP <name> JOIN DDL command
  - If SEEDING_MODE=AUTOMATIC each replica needs ALTER AVAILABILITY GROUP <name> GRANT CREATE ANY DATABASE run to allow seeding
- **On the primary replica server add databases**
  - ALTER AVAILABILITY GROUP <name> ADD DATABASE <dbname>

Failing Over a Read-Scale Availability Group Without Data Loss

- **Alter the availability group to AVAILABILITY_MODE = SYNCHRONOUS_COMMIT for the target replica server**
- **SET REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT = 1**
  - This forces commit on the secondary replica for every transaction
- **Demote the primary replica to read-only**
  - ALTER AVAILABILITY GROUP <name> SET (ROLE = SECONDARY)
- **Promote the secondary to primary**
  - ALTER AVAILABILITY GROUP <name> FORCE_FAILOVER_ALLOW_DATA_LOSS
  - SET REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT = 0 and AVAILABILITY_MODE = ASYNCHRONOUS_COMMIT if required

Failing Over a Read-Scale Availability Group With Data Loss

- **Should only be done when the primary replica is lost and cannot be recovered**
- **ALTER AVAILABILITY GROUP <name> FORCE_FAILOVER_ALLOW_DATA_LOSS on target secondary replica**
- **NOTE: Previous primary replica recovery will also assume primary role = "split brain"**
  - ALTER AVAILABILITY GROUP <name> SET (ROLE = SECONDARY) must be run to prevent this

Creating an Availability Group for HA

- **Requires Corosync/Pacemaker configuration similar to Failover Clustered Instance**
  - Does not require shared storage configuration
- **Requires a minimum of three replicas for high availability**
  - Three synchronous commit replicas
  - Two replicas and one CONFIGURATION_ONLY replica for quorum
    - CONFIGURATION_ONLY replica may be Express Edition
- **Availability Group created with CLUSTER_TYPE=EXTERNAL**

Joining Replicas to an Availability Group

- **Join each secondary replica to the Availability Group**
  - ALTER AVAILABILITY GROUP <name> JOIN DDL command
- **If SEEDING_MODE=AUTOMATIC each replica needs**
  - ALTER AVAILABILITY GROUP <name> GRANT CREATE ANY DATABASE run to allow seeding
- **If SEEDING_MODE=NONE databases will not be automatically created on secondary replicas and must be joined manually through backup/restore**
  - Requires FULL backup and at least one transaction log backup

Configure Cluster in Pacemaker

- Install the pacemaker packages on all replicas
- Set the password for the hacluster user on all nodes
- Enable and start pcsd and pacemaker
- Create the cluster with pcs and start the cluster
- Install the mssql-server-ha package for the resource agent
- Configure fencing (STONITH) for production workloads
- Configure cluster-recheck-interval: 2 minutes recommended
- Configure start-failure-is-fatal: true recommended

Configure SQL Server Login for Pacemaker

- **Create a SQL Server login with password for Pacemaker usage**
- **Login for Pacemaker must be a member of the sysadmin fixed server role**
  - May be reduced to ALTER, CONTROL, VIEW DEFINITION permissions only on the Availability Group after it is created fully
- **Save the credentials for the SQL Server login**
  - /var/opt/mssql/secrets/passwd/pacemaker-passwd
  - Mark as only readable by root with chown and chmod 400

Create Availability Group Cluster Resources

- **Create the Availability Group cluster resource**
  - sudo pcs resource create ag_cluster ocf:mssql:ag ag_name=ag1 meta failure-timeout=60s master notify=true
- **Create virtual IP resource on available static IP address for servers**
  - sudo pcs resource create virtualip ocf:heartbeat:IPaddr2 ip=<10.128.16.240>
  - Multi-Subnet configuration requires exporting the cluster information base (CIB) for Pacemaker and manually adding the additional IP resources to the XML, then importing the CIB back into Pacemaker
- **Add colocation constraint to virtual IP resource of INFINITY**
  - sudo pcs constraint colocation add virtualip ag_cluster-master INFINITY with-rsc-role=Master
- **Add ordering constraint to the colocation constraint – forces virtual IP resource to be moved before the Availability Group during failover**
  - sudo pcs constraint order promote ag_cluster-master then start virtualip

Manual Failover for High Availability

- **DO NOT use Transact-SQL, SSMS or Powershell to failover a Linux HA Availability Group**
- **When CLUSTER_TYPE=EXTERNAL the FAILOVER_MODE is EXTERNAL as well**
- **Use sudo pcs resource move to move failover the resource in Linux**
  - Remove location constraint created by the pcs move command after failover
  - A location constraint is not created during a true automated failover

Rolling Updates to Availability Groups

- **Create a location constraint for the node to be updated using:**
  - sudo pcs constraint location <resource> avoids <node>
- **Update the SQL Server packages from the source repository using:**
  - sudo yum update mssql-server
  - sudo yum update mssql-server-agent (SQL Server Agent)
  - sudo yum update mssql-server-fts (Full-Text Search)
  - sudo yum update mssql-server-ha
- **Remove the location constraint using:**
  - sudo pcs constraint remove
- **Failover to the updated node and repeat the steps until all nodes are patched**

Summary

- Possible to go very deep on all of these topics
- Much more information available in docs.microsoft.com and external blogs
- SQL Server 2017 provides excellent HA features