Relational Database Management Systems in the Cloud: Microsoft SQL Server 2008 R2

Miles Ward

July 2011

Table of Contents

Introduction

Amazon Web Services (AWS) is a flexible, cost-effective, easy-to-use cloud computing platform. Running your own relational database on Amazon EC2 is the ideal scenario for users whose application requires a specific, traditional relational database, or for those users who require a maximum level of control and configurability. Relational Database Management Systems (RDBMS) are some of the most widely deployed software packages within the Amazon cloud.

In this white paper, we help you understand one of the most popular RDBMS options available with the AWS cloud computing platform—Microsoft's SQL Server. We provide an overview of general best practices that apply to all major RDBMS options, and we examine important Microsoft SQL Server implementation characteristics such as performance, durability, and security. We pay particular attention to identifying features that support scalability, high-availability, and fault-tolerance.

Relational Databases on Amazon EC2

AWS provides an ideal platform for running many traditional, third-party relational database systems in the cloud. Some of the unique characteristics of the public cloud provide strong benefits for RDBMS workloads. In many ways AWS will behave similarly to local, physical infrastructure; some minor differences apply to all major RDBMS systems. A general understanding of those differences can assist greatly in making good architecture decisions for your system.

AWS vs. Your Server

For example, let's compare a typical single, 1U, rack-mount server to an Amazon Elastic Compute Cloud (Amazon EC2) instance.

Example 1U Server	**EC2 Extra Large Instance (m1.xlarge)**
• 1 quad-core Xeon processor, 2.4Ghz • 16GB of memory • 2 x 300GB SATA hard drives • 1 Gigabit Ethernet Network	• 8 EC2 Compute Units (4 virtual cores with 2 EC2 Compute Units each). One EC2 Compute Unit (ECU) provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. • 15GB of memory • 1690GB of local instance storage • Ethernet Network

At first glance, these two computers are very similar: they provide roughly equivalent CPU, RAM, local disk resources, and network infrastructure. There are however significant differences:

1. The EC2 instance is rapidly replaceable, duplicable, and available on-demand.
2. The EC2 instance can grow and shrink, from a single logical CPU with 1.7GB of RAM, all the way up to 8 logical CPUs with 68.4GB of RAM. This requires instance reboot, and a simple configuration change via our API or CLI.
3. The EC2 instance only costs you money when it's on; if you can shut down even part of your instance fleet during non-peak times, you can save costs. Persistent storage options protect your data, but do have persistent costs even when your instances are "stopped." Plus, there are no up-front charges or setup fees.
4. The EC2 instance is supported by the AWS network and facilities infrastructure; you never have to touch the hardware.
5. There are potentially other virtual instances utilizing the physical infrastructure that supports your EC2 instance. While there is no contention for CPU or memory, the network is a shared resource. You might have access to only a fraction of the physical infrastructure's network connection depending on instance size.
6. Our facilities (called Availability Zones) are likely larger than your network environment, and EC2 instances (except for Cluster Compute) start in random physical locations within the Availability Zone. This is good for reliability, but it means that server to server communications could potentially have higher latencies than on a smaller local network.
7. Because of virtualization overhead we have "un-round" memory sizes (613MB, 1.7GB, 7.5GB, 15GB, 17.1GB, 22GB, 23GB, 34.2GB, 68.4GB); applications tuned for specific memory footprints might need to opt for more memory than absolutely required, or be re-tuned for these sizes.
8. There is more local disk storage on the EC2 server than our example server; however, EC2 local storage is "ephemeral," meaning that it will be deleted upon stop or termination of the instance. Ephemeral disk is best used to store temporary files, caches, and other files that don't require persistence.

In addition to the resources available inside your EC2 instance, Amazon provides other valuable resources, available via our internal network, for your database:

1. Amazon Elastic Block Store (Amazon EBS). Amazon EBS volumes are durable, high-performance, network-attached block device resources. These "virtual disks" can be attached to your servers, and can persist when servers are stopped or terminated, thus providing durable storage for databases. Amazon EBS can also natively backup to Amazon S3, providing an extension to its already very high durability.
2. Amazon Simple Storage Service (Amazon S3). Amazon S3 provides a highly durable storage infrastructure designed for mission-critical and primary data storage. S3 provides backup storage for "snapshots" of EBS disks as well as any other static content you need for your application. S3 is designed for 99.999999999% data durability, and as a result is an ideal target for your database backups.
3. Amazon CloudWatch, our internal instance monitoring service. Amazon CloudWatch provides detailed CPU, disk, and network utilization metrics for each enabled EC2 instance and EBS disk, allowing detailed reporting and management. This data is available in our web-based AWS Management Console as well as our API, which allows for infrastructure automation and orchestration based on these availability and load metrics.

Performance

The performance of a relational database instance on Amazon EC2 depends on many factors, including the EC2 instance type, the number and configuration of EBS volumes, the configuration of the database software, and the application workload. We encourage users to benchmark their actual application on several EC2 instance types and storage configurations in order to select the most appropriate configuration.

Increasing the performance of your database requires an understanding of which of the server's resources is the performance constraint. If CPU or memory limits your database performance users can scale up the memory, compute, and network resources available to the RDBMS software by choosing a larger EC2 instance types. Remember, 32-bit Amazon Machine Images (or AMI's) can't run on 64-bit instances, so if you're expecting to need the higher performance instance types, pick 64-bit instances to start. Also, remember that changing an existing instance to a different size requires a stop/start cycle.

If your performance is disk I/O limited, changes to the configuration of your disk resources may be in order. Remember that EBS volumes, the persistent block storage available to EC2 instances, are connected via the network. An increase in network performance can have a significant impact on aggregate "disk" performance, so be sure to choose the appropriate instance size. To scale up random I/O performance, you can increase the number of EBS volumes as a ratio of EBS storage (6x 100GB EBS volumes versus 1 x 600GB EBS volume), and if aggregation is required, use software RAID 0 (disk striping) across multiple EBS volumes to increase single logical volume total IOPS. Remember, utilizing RAID striping reduces operational durability of the logical volume by a degree inversely proportional to the number of EBS volumes in the stripe set. A single EBS volume can provide approximately 100 IOPS, and single instances with arrays of 10+ attached EBS disks can often reach 1,000 IOPS sustained. Data, log, and temporary files will benefit from being stored on independent EBS volumes or RAID volume aggregates because they present different I/O patterns. In order to take advantage of additional attached EBS disks, be sure to evaluate the network load to ensure that your instance size is sufficient to provide the network bandwidth required. For sequential disk access, ephemeral disks are somewhat higher performance, and don't impact your network connectivity. Some customers have found it useful to use ephemeral disks to store tempdb or other temporary files to conserve network bandwidth and EBS I/O for log and DB operations.

In many cases, the constraint is more abstract. In general, users have the same database performance tuning options in the EC2 environment that they would have in a physical server environment. Users can also scale the total performance of a database system by scaling horizontally across multiple servers with sharding, caching, synchronous and asynchronous replication strategies.

Durability and Availability

The Amazon EC2 environment is designed to provide the tools you need to deliver highly durable services, and RDBMS services are no exceptions.

Durability is provided by AWS at many different levels:

- At the backup level, Amazon S3 is designed to provide 99.999999999% durability.
- At the volume level, via the 0.1% to 0.5% AFR of EBS volumes.
- At the instance level, due to the ability to detach volumes and re-attach to new instances of EC2 servers in the event of an instance failure and the rapid access to replacement instances.
- At the Region level, due to the ability to create zone independence by utilizing multiple Availability Zones.
- At the AWS system level, due to the redundancy and fault tolerance of the API control layer design. The EC2 API has a 99.95% Annual Uptime Percentage SLA.

No approach to RDBMS is complete without using the available options in each system for application level redundancy. Log shipping, mirroring, and other strategies are critical to providing the highest fault-tolerance in your RDBMS design.

Elasticity and Scalability

In many cases, users of traditional relational database solutions on Amazon EC2 can take advantage of the elasticity and scalability of the underlying AWS platform. For example, once you have configured an EC2 instance with your database solution, you can bundle the instance into a custom AMI (using the Bundle commands for instance store AMIs, or the Create Image command for EBS AMIs), then create multiple new instances of your database configuration within a few moments.

For many customers, increasing the performance of a single DB instance is the easiest way to increase the performance of their application overall. In the EC2 environment, you can simply stop an instance, increase the instance size using either the ec2-modify-instance-attribute command our AWS Management Console and restart the instance. This is particularly true if you have a set maintenance window and can tolerate system downtime. This is often referred to as *scaling up*. More advanced scaling, sharding, or otherwise spreading the DB query load across multiple EC2 instances, can be used to provide even higher performance. This is often referred to as *scaling out*.

Configuration

To create a relational database on EC2, you can either start with one of the many ready-to-use relational database AMIs provided in EC2 (http://aws.amazon.com/running_databases), or you can start an instance from an AMI with the desired base OS, then install the database using the standard OS software install processes. Remember, there's no DVD drive, so users must download the required software. Many customers use the AWS Management Console to upload setup files to Amazon S3 from their workstations and then they download the files on the instance using the console.

After your database is installed and configured on Amazon EC2, you interact with your service via its exposed interface, either by exposing that interface over the web (or a VPN) to your location, or remotely accessing the server via SSH, NX, VNC, or RDP. You can work with a database on EC2 just as you would with an on-premises database. You'll need to configure the database instance's security group to allow traffic on the port used by the DBMS. All ports except port 22 are disabled by default.

Remember, although the EBS volumes that serve as the root drives of Windows EC2 instances are often durable, and are designed to survive the physical failure of the EC2 instance host or individual EBS hardware failure, there is often complex OS and application configuration that should be preserved by re-bundling your AMI. By re-bundling your AMI, you ensure that subsequent (or additional) launches of EC2 instances will include all of your configuration changes. For directions, go to Creating Amazon EBS-backed AMIs (http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/index.html?creating-an-ami-ebs.html) in the *Amazon Elastic Cloud Compute User Guide.*

Leveraging Other Amazon Structured Storage Scenarios

Although running your own relational database on Amazon EC2 is a great solution for many users, there are a number of scenarios where other AWS solutions might be the better choice:

- **Index-and-query focused data**. Many cloud-based solutions don't require the transactional, complex join, and other features found in a traditional relational database. If your application is aimed at indexing and querying data, you may find Amazon SimpleDB to be more appropriate for your needs, and significantly easier to manage. Details on SimpleDB (which is not a RDBMS and is not ACID compliant) are available here: http://aws.amazon.com/simpledb
- **Numerous binary large objects (blobs)**. While many relational databases support blobs, if your application makes heavy use of these objects (e.g. video, audio, images, and so on), you may find Amazon Simple Storage Service (Amazon S3) to be a better choice. Many AWS customers have found it useful to store blob-style data in Amazon S3 while using Amazon SimpleDB to manage blob metadata.
- **Automatic elasticity.** Users of relational databases on AWS can often leverage the elasticity and scalability of the underlying AWS platform, but this is a manual task for system administrators or DBAs, or a scripting / automation task for your developers. Assuming that you need or want elasticity out of the box (and that your data structures are a good fit), you may opt for another structured storage choice such as Amazon SimpleDB or Amazon S3.

Microsoft SQL Server 2008 on Amazon EC2

Overview

Microsoft SQL Server 2008 is a highly manageable RDBMS with a significant first and third party application ecosystem. SQL Server runs on Microsoft Windows Server operating systems.

Licensing

As with any commercial software product, it's important that you adhere to the license, terms, and conditions that your software requires. Within AWS, there are three ways to use SQL Server:

1. Microsoft SQL Server Express Edition is available direct from Amazon, via a pre-built AMI. This AMI is billed at the standard Windows Server rate.
2. Microsoft SQL Server Standard Edition is available direct from Amazon, via a pre-built AMI. This AMI is billed at a higher hourly rate, which includes the price of the SQL Server Standard license. See details at http://aws.amazon.com/windows.
3. Start a Windows server and install your own SQL Server software. This option is called License Mobility, and as of July 2011, allows Microsoft licensees who are members of the Software Assurance program to migrate Microsoft application licenses to AWS. More details are available here (http://www.microsoft.com/licensing/software-assurance/license-mobility.aspx).

The recommendations in this document will focus on best usage of the SQL Server Standard Edition feature set.

Microsoft's approach to licensing for SQL Server revolves around physical processors, or sockets, rather than logical processors or cores. As of this writing, our virtualization technology presents each logical processor core to an EC2 instance as an individual socket. For example, the EC2 instance type m2.4xlarge provides 8 cores, which are represented by the Windows device manager as being 8 distinct, single core processors (i.e., 8 sockets). Be aware of this distinction when you plan your licensing. For the example, EC2 instance types m2.4xlarge and c1.xlarge, which have 8 available cores, will only utilize 4 cores due to Microsoft licensing restrictions on Standard Edition. SQL Server Enterprise, which allows up to 64 logical processors, is not effected by this limitation.

Performance

For almost all SQL Server workloads, Amazon recommends the use of EBS volumes for data and log storage, and ephemeral (local instance) volumes for tempdb storage. EBS volumes, due to the nature of their design, impart a first-write and first-read performance penalty. In testing or for rapid provisioning of production hardware, be sure to expect increased query latency, reduced IOPS, and reduced throughput for the first write and read. To prevent this effect, in particular for benchmarking or other testing, you should "pre-warm" the disk by performing a full (not a quick) format of each EBS volume.

EBS Volumes on Windows

A Windows Server instance on Amazon EC2 maps EBS volumes using the xvd* labeling, where * can be replaced with "f" through "p". As of June 2011, this results in 11 additional volumes beyond the EBS volume used for boot available for use per instance. SQL Server data should not be placed on the boot volume. Using the Windows disk manager, you can create individual or striped volumes and assign logical drive letters (D:, E:, F: etc.) to the attached EBS volumes associated with each xvdF through xvdP drive label. Remember that logical volumes/partitions on the same EBS volume are contending for disk I/O and throughput.

NTFS Allocation Unit Size

When formatting the partition that will be used for SQL Server data files, it is recommended that you use a 64KB allocation unit size for data, logs, and tempdb. Be aware however, that using allocation unit sizes greater than 4KB results in the inability to use NTFS compression on the volume. SQL Server does support read-only data on compressed volumes, but it is not recommended.

Database File Placement

- If you have a set of tables that are frequently used together, consider putting those tables on separate file groups on separate EBS volumes, which will help balance I/O between them. In a larger, more heavily used system, this could be a significant difference.
- Consider putting non-clustered indexes in a separate file group, in order to split I/O between file groups.
- Group your tables based on usage, to generate as many simultaneous reads to different file groups (and therefore EBS volumes) as possible. Grouping tables into file groups based on a maintenance need for convenient backup plans will not generate as much performance as separating the tables and indexes by usage.
- For smaller systems, use autogrow for your database files, but keep in mind that when a "grow" is initiated, transactions must wait while the database grows. In a small database or lightly queried system this is not a big issue, but if you have a 100GB OLTP database set to grow in 10 percent increments, and it runs out of space during peak times, the online users will be held up while the 10GB is allocated. Also, remember that while Amazon EBS volumes cannot be resized in situ, a snapshot of an EBS volume can be restored to a larger volume, and attached to replace the earlier volume. EBS volumes have a maximum size of 1TB.
- For a larger system, the best practice is to anticipate database growth and manually increase the database at a scheduled time. Or, choose a reasonable amount to grow by that is neither too cumbersome nor so small that it will initiate expansion too frequently.
- If you have multiple files in your file group, you will need to expand them in order to reestablish proportional fill.

Log File Placement

- Create the transaction log on separate Amazon EBS volumes from your data store. The transaction log file is written sequentially; therefore, using a separate, dedicated disk allows the disk heads to stay in place for the next write operation. EBS volumes provide write caching to accelerate small writes.
- Set your transaction log to AUTOGROW, but try to size it so it should not need to grow. The optimal size should be based on your recovery model, the level of logged activity in the database, and the interval of time between backups. Set the growth increment to a reasonable percentage, but try to anticipate when the log should be resized. If the transaction log expands too frequently or takes a long time to expand, performance can be affected.
- The size of the log should be based on your current recovery model and your application design. If you find that you need to shrink the log periodically, you should further investigate what is causing the log to fill up, in order to fix the problem at the root rather than simply fixing the symptom.

tempdb File Placement

- In most use cases, it is best to place tempdb on ephemeral disks available to each instance. If you terminate your instance, or your instance fails, the contents of tempdb will be permanently lost, so be sure that your database applications aren't misusing tempdb to store critical data. Details on making ephemeral disks available on Amazon EBS-backed AMIs are available here (http://docs.amazonwebservices.com/AWSEC2/2011-05-15/UserGuide/block-device-mapping-concepts.html#Using_OverridingAMIBDM).

Lining Up the Number of Data Files with CPUs

- It is recommended to have between .25 and 1 data files (per file group) for each virtual core on your EC2 instance. This has scalability advantages for allocation-intensive workloads.
- The scalability advantage for allocation-intensive workloads is especially true for tempdb, where the recommendation is 1 data file per virtual core.

Standard SQL Optimizations

- Data files should be of equal size. SQL Server uses a proportional fill algorithm that favors allocations in files with more free space.
- Pre-size data and log files.
- Do not rely on AUTOGROW, instead, manage the growth of these files manually. You may leave AUTOGROW on for safety reasons, but you should proactively manage the growth of the data files.
- For faster data loads, you might want to load your tables without any indexes and then create the indexes later.
- Make sure your data and log files have sufficient space to complete the operation without having to autogrow these files. The autogrow process can significantly affect the overall load time.

Validate Configurations Prior to Deployment

You might find it useful to follow more programmatic analysis of server setup. The Microsoft SQL Server 2008 R2 BPA (http://www.microsoft.com/downloads/en/details.aspx?displaylang=en&FamilyID=0fd439d7-4bff-4df7-a52f-9a1be8725591) is a diagnostic tool that performs the following functions:

- Gathers information about a server and a Microsoft SQL Server 2008 or 2008 R2 instance installed on that server
- Determines if the configurations are set according to the recommended best practices
- Reports on all configurations, indicating settings that differ from recommendations
- Indicates potential problems in the installed instance of SQL Server
- Recommends solutions to potential problems

Do basic throughput testing of the I/O subsystem prior to deploying SQL Server.

- Make sure these tests are able to achieve your I/O requirements with an acceptable latency. SQLIO is one such tool that can be used for this. A document is included with the tool with basics of testing an I/O subsystem. Download the SQLIO Disk Subsystem Benchmark Tool (http://www.microsoft.com/downloads/details.aspx?familyid=9a8b005b-84e4-4f24-8d65-cb53442d9e19&displaylang=en).

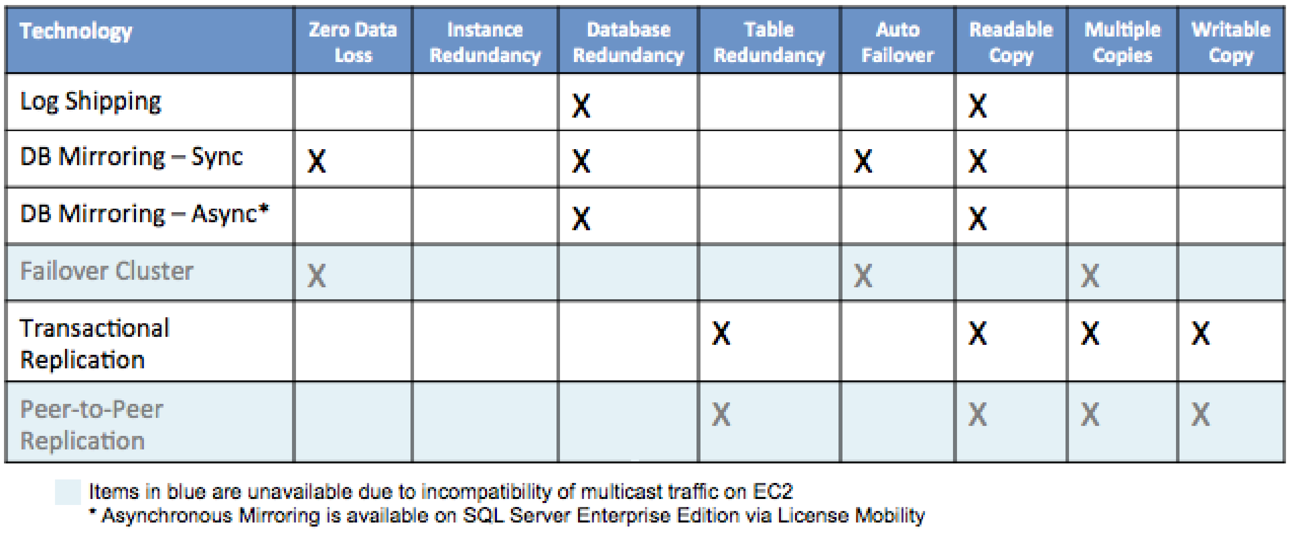
- Understand that the purpose for running the SQLIO tests is not to simulate SQL Server's exact I/O characteristics, but rather to test maximum throughput achievable by the I/O subsystem for common SQL Server I/O types.
- IOMETER can be used as an alternative to SQLIO.

Durability and Availability

Microsoft SQL Server has several approaches for providing application-level high-availability and fault tolerance.

High Availability

High availability technologies have varying characteristics and abilities, which make each of them suitable for some scenarios but not others. Table 1 below shows these characteristics in a way that allows easy comparison of the technologies.



Technology	Zero Data Loss	Instance Redundancy	Database Redundancy	Table Redundancy	Auto Failover	Readable Copy	Multiple Copies	Writable Copy
Log Shipping			X			X		
DB Mirroring – Sync	X		X		X	X		
DB Mirroring – Async*			X			X		
Failover Cluster	X				X		X	
Transactional Replication				X		X	X	X
Peer-to-Peer Replication				X		X	X	X

🟦 Items in blue are unavailable due to incompatibility of multicast traffic on EC2
* Asynchronous Mirroring is available on SQL Server Enterprise Edition via License Mobility

Backup and Recovery

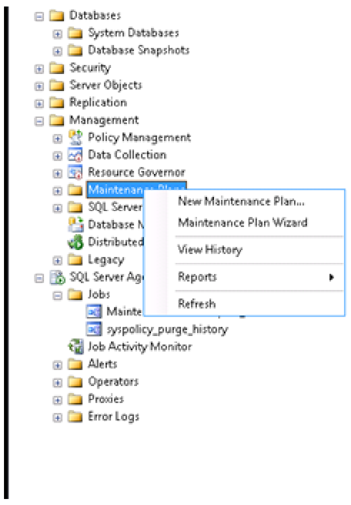
Any high-availability strategy must include a comprehensive backup strategy. Even if the high-availability strategy includes technologies to provide redundant copies of the databases being protected, there are reasons why backups should also exist:

- The failure might affect the redundant copies of the databases as well as the primary copies. In this case, without backups to allow restoration, the data might be completely lost.
- It might be possible to restore a small portion of the databases, which may be more palatable than failing over. Furthermore, it might be possible to restore that small portion of the database without affecting the portion being used by the application—effectively recovering from the failure without downtime.

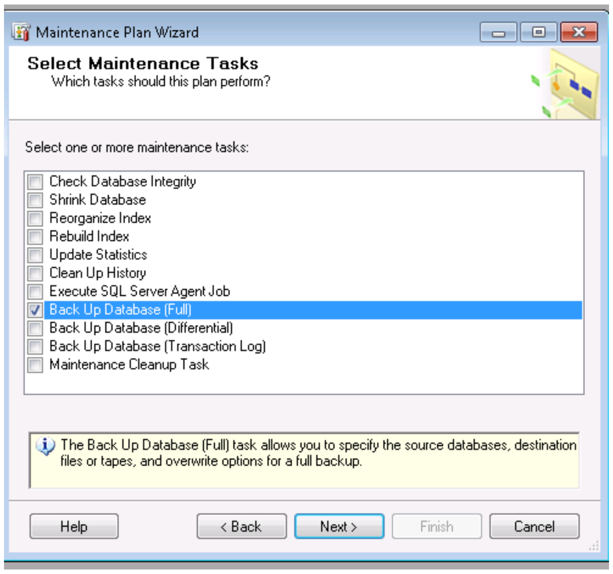
Backups require significant space, and will contend for disk I/O with your production workloads during the backup process. For databases under 5GB, you could mount an Amazon S3 bucket as a file system/drive letter using TNT drive (http://tntdrive.com), or another Windows S3 utility, and use that as the target for your backup. This will make most backups take longer than they would if you targeted an ephemeral disk or an EBS volume, but in high-utilization designs, it can be a useful alternative.

To use the Maintenance Plan feature of SQL Server:

1. Launch the Maintenance Plan Wizard by right-clicking **Maintenance Plans** in the **Management** folder in the SQL Server Management Studio.



2. Select **Back Up Database (Full)**. Optionally, select additional maintenance tasks, such as **Rebuild Indexes** or **Update Statistics**, to ensure that your databases run in an optimized manner.



Maintenance Plan Wizard

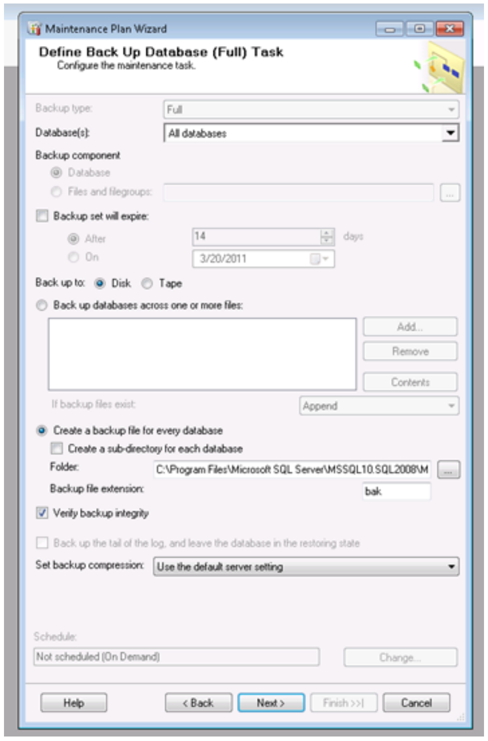
Select Maintenance Tasks
Which tasks should this plan perform?

Select one or more maintenance tasks:

- [ ] Check Database Integrity
- [ ] Shrink Database
- [ ] Reorganize Index
- [ ] Rebuild Index
- [ ] Update Statistics
- [ ] Clean Up History
- [ ] Execute SQL Server Agent Job
- [x] Back Up Database (Full)
- [ ] Back Up Database (Differential)
- [ ] Back Up Database (Transaction Log)
- [ ] Maintenance Cleanup Task

The Back Up Database (Full) task allows you to specify the source databases, destination files or tapes, and overwrite options for a full backup.

3. Further define your selected tasks by selecting **All databases** from the drop-down menu and verifying that **Create a backup file for every database** is selected. As additional databases are created, they will be included in this back up job.



Maintenance Plan Wizard

Define Back Up Database (Full) Task
Configure the maintenance task.

Backup type: Full
Database(s): All databases
Backup component
● Database
○ Files and filegroups:
- [ ] Backup set will expire:
  ● After 14 days
  ○ On 3/20/2011

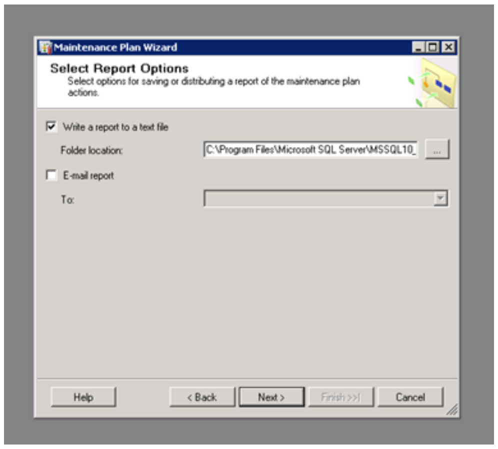
Back up to: ● Disk ○ Tape
○ Back up databases across one or more files:
If backup files exist: Append
● Create a backup file for every database
- [ ] Create a sub-directory for each database

Folder: C:\Program Files\Microsoft SQL Server\MSSQL10.SQL2008\M
Backup file extension: bak
- [x] Verify backup integrity
- [ ] Back up the tail of the log, and leave the database in the restoring state

Set backup compression: Use the default server setting

Schedule: Not scheduled (On Demand)

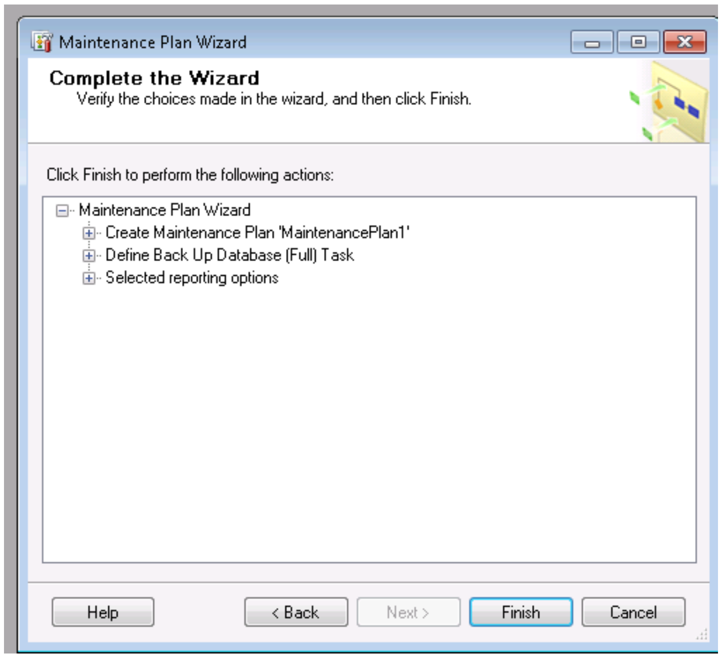
4. Select reporting options.



Select Report Options
Select options for saving or distributing a report of the maintenance plan actions.

[✓] Write a report to a text file
Folder location: C:\Program Files\Microsoft SQL Server\MSSQL10_
[ ] E-mail report
To:

5. Complete the Wizard by clicking **Finish**.



Complete the Wizard
Verify the choices made in the wizard, and then click Finish.

Click Finish to perform the following actions:

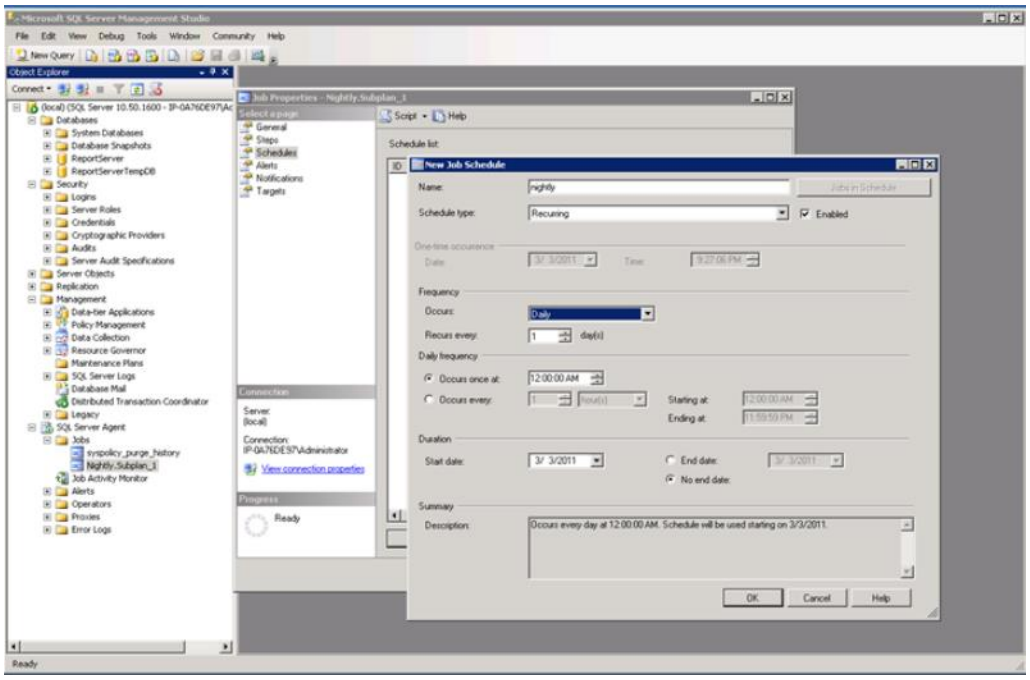
- Maintenance Plan Wizard
  - Create Maintenance Plan 'MaintenancePlan1'
  - Define Back Up Database (Full) Task
  - Selected reporting options

6. After the Maintenance Plan has been created, you can right-click the new plan, and execute. Verify that your backups of the System databases completed successfully at the default location, unless another location was specified in the Wizard (for example the Amazon S3 bucket mounted as a drive letter):

```
C:\Program Files\Microsoft SQL Server\MSSQL10_50.MSSQLSERVER\MSSQL\Backup
```

7. You can select a schedule to run this regularly with the SQL Server Agent Job that is automatically created.

8. Create or select a schedule for the Maintenance Plan job. You can select the automatically created Maintenance Job created, double-click to open the job and select a schedule, and complete the dialog box with the schedule you want.


Microsoft SQL Server Management Studio
Job Properties - Nightly.Subplan_1
New Job Schedule
OK
Cancel
Help

Name: nightly
Schedule type: Recurring
Enabled [✓]
Frequency
Occurs: Daily
Recurs every: 1 day(s)
Daily frequency
[✓ Occurs once at: 12:00:00 AM]
Duration
Start date: 3/ 3/2011
[✓ No end date]
Summary
Description: Occurs every day at 12:00:00 AM. Schedule will be used starting on 3/3/2011.

Replication Strategies

Log Shipping

Log shipping is the simplest way to provide one or more redundant copies of a single database. The primary database on the primary server is backed up and then restored to one or more secondary servers. Transaction log backups are then repeatedly taken of the primary database, shipped (that is, copied over the network) to the secondary server(s), and then restored. In this manner, the secondary databases are continually being updated with changes from the primary database through transaction log restores. An optional third SQL Server instance can be used to monitor the primary and secondary servers to ensure that transaction log backups are being taken and restored regularly.

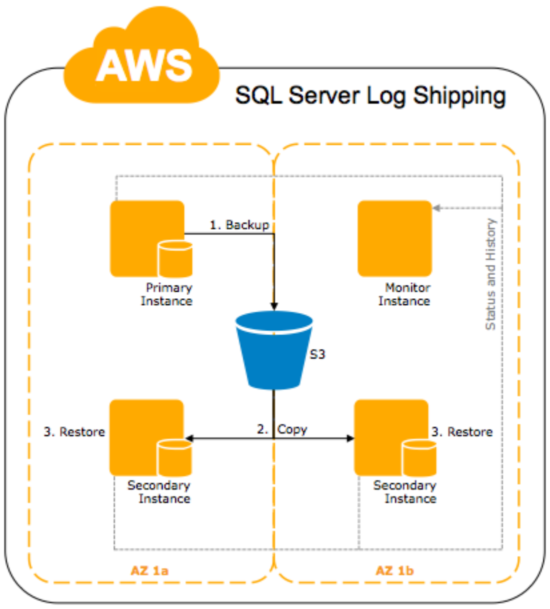


Figure 1: Example log shipping configuration

Figure 1 illustrates the steps performed by backup, copy, and restore jobs, as follows:

- The primary server instance runs the transaction log backup job on the primary database. This instance then places the log backup into a primary log-backup file, which it sends to the backup folder. In this figure, the backup folder is on a shared directory, known as the “Backup Share.” This could be built as a separate EC2 instance, or could use an S3 bucket mounted as a file system instead.
- Each of the three secondary server instances runs its own copy job to copy the primary log-backup file to its own local destination folder.
- Each secondary server instance runs its own restore job to restore the log backup from the local destination folder onto the local secondary database.

The primary and secondary server instances send their own history and status to the monitor server instance.

Database Mirroring

Database mirroring provides a redundant copy of a single database that is automatically updated with changes. Database mirroring works by sending transaction log records from the main database (called the *principal*) to the redundant database (called the *mirror*). The transaction log records are then replayed on the mirror database continuously (that is, the mirror database is constantly running recovery using the transaction log records sent from the principal).

The principal and mirror databases are usually hosted on separate SQL Server 2008 instances (called the *principal server* and *mirror server*) on separate physical servers, and often in separate data centers. This basic configuration is shown in Figure 2.

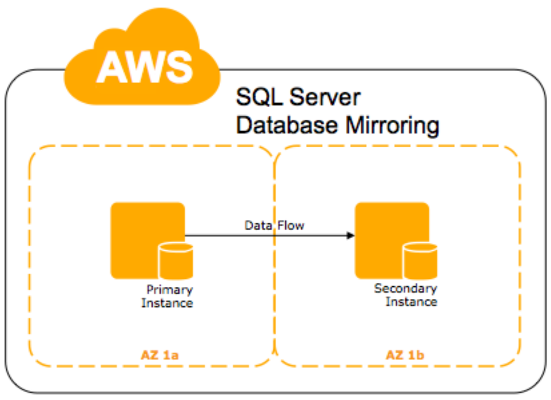


Figure 2: Basic database mirroring configuration

Application connections are only accepted to the principal database—connections attempted to the mirror database may be redirected to the principal database using client redirection, which is explained later in this section. A mirror database can be used for querying, however, through the use of a database snapshot. Additionally, database mirroring supports only a single mirror.

Database mirroring is different from log shipping or transactional replication in the following ways:

- Failures are automatically detected.
- Failovers can be made automatic.

If database mirroring is configured to support automatic failure detection and failover, it provides a hot standby solution.

There are two modes of database mirroring—synchronous and asynchronous. With synchronous mirroring, transactions cannot commit on the principal until all transaction log records have been successfully copied to the mirror (but not necessarily replayed yet). This means that if a failure occurs on the principal and the principal and mirror are synchronized, committed transactions are present in the mirror when it comes online—in other words, it is possible to achieve zero data loss. SQL Server Standard edition allows synchronous mirroring. SQL Server Enterprise edition, available on EC2 via License Mobility, adds asynchronous mirroring as a supported replication approach. Using this more latency tolerant synchronization method allows SQL Server systems to automatically replicate across EC2 regions (for example from US West to US East) for increased redundancy. Implementations of this type also often include WAN optimization to minimize the impact of packet loss and other WAN limitations on replication performance.

Synchronous mirroring can be configured to provide automatic failover, through the use of a third SQL Server 2008 instance called the *witness server* (usually hosted on a separate instance). The sole purpose of the witness is to agree (or not) with the mirror that the principal cannot be contacted. If the witness and mirror agree, the mirror can initiate failover automatically. If synchronous mirroring is configured with a witness, the operating mode is known as *high-availability mode* and provides a hot standby solution. When no witness is defined, the operating mode is known as *high-safety* mode, which provides a warm standby solution. Note: the witness server does not need to have the same performance characteristic as the Primary and Secondary servers.

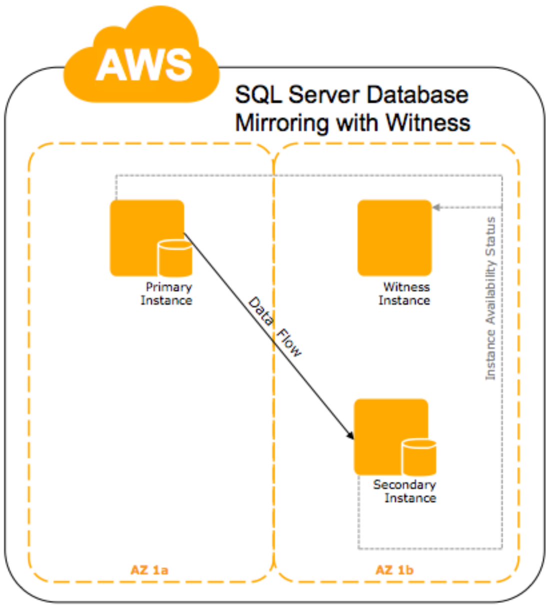


Figure 3: High-availability configuration using a witness server

After a failover, the application must reconnect to the database, but the application does not know whether the (old) principal server or the mirror server is now the principal. There are two ways for the application to connect, using implicit or explicit client redirection. With explicit client redirection, the application specifies both mirroring partners in the connection string, so no matter which server is the primary, the connection should succeed. With implicit client redirection, only one partner is specified, but if it is the current mirror, it redirects the connection to the current principal. However, if the specified partner is not available to perform the redirection (for example, it is completely offline), the connection fails. For this reason, it is recommended to always use explicit client redirection. See more details here: http://technet.microsoft.com/en-us/library/cc917681.aspx.

Transactional Replication

Transactional replication involves creating a publication (data in one or more tables in a database) in a publication database on a Publisher instance. The initial state of the publication is copied to one or more Subscriber instances where it becomes the subscription in a subscription database. This initialization process can be performed using a database backup or using a snapshot (in which replication itself works out what needs to be copied and only copies what is needed, rather than creating a full database backup). After the subscriptions are initialized, transactions are replicated from the Publisher to the Subscribers, using the process shown in Figure 4.

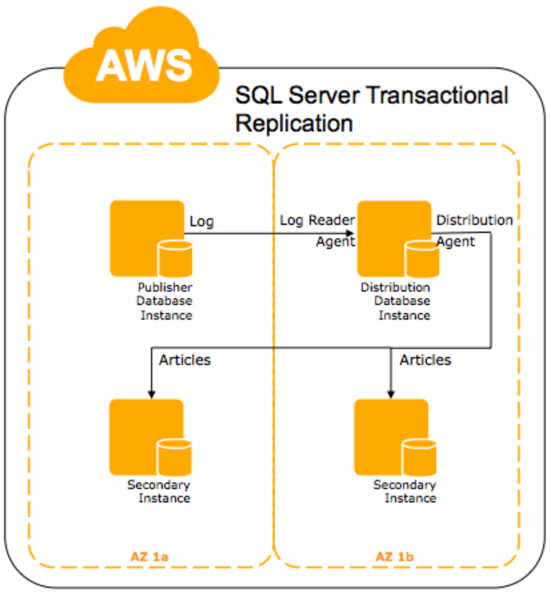


Figure 4: Transactional replication process

Committed transactions that affect the publication are read from the transaction log on the Publisher, stored in the distribution database on the Distributor, and then applied to the subscription database on the Subscriber(s). In this way, the subscriptions are constantly being updated with changes that occur to the publication. If the Publisher goes offline, transactional replication automatically restarts when it comes online again. Also, if a Subscriber goes offline, the transactions that must be propagated to it are held in the distribution database until the Subscriber comes online again.

The intermediate storage of the transactions is necessary so that the transactions do not have to be stored in the transaction log of the publication database. Ideally, the Distributor is a separate SQL Server instance running on a separate server, because this provides extra redundancy and offloads the distribution workload from either the Publisher or Subscriber instances.

As with log shipping, replication does not provide automatic detection of a Publisher failure or automatic failover to a Subscriber, so it is also a warm standby solution. Additionally, there is latency between a transaction occurring on the Publisher and being propagated to the Subscribers, so data loss is possible at the time of a failure. Finally, transactional replication only protects the data in the publication—it cannot be used to protect an entire database or group of databases.

As with log shipping, a failover is not be detected by client applications. This means extra logic must be added to the client or possibly in a mid-tier to handle their redirection. This logic must be added separately.

For additional detail see:

- Mirroring: http://technet.microsoft.com/en-us/library/ms188712.aspx
- Log-Shipping: http://msdn.microsoft.com/en-us/library/ms188698.aspx
- Replication: http://technet.microsoft.com/en-us/library/ms151198.aspx
- High availability: TechNet: High Availability with SQL Server 2008 R2

Security

Network

To access SQL Server resources from other Amazon EC2 instances, or remote servers, you need to configure your Security group to permit ingress over TCP port 1433 or whatever custom port you've configured SQL Server to utilize. Ports can be opened to all computers in another EC2 Security Group, or to a specific IP address or CIDR IP address range. Security group parameters can be changed at any time. For extra security, some users use automation scripts to open these ports when needed and close them when SQL Server resources are not in use.

Best practice in a multi-tier architecture is to permit operational access to the database persistence tier only from servers in the security group that requires access, and control access only from known administrative IP addresses.

Encryption in-transit

Amazon EC2 resources, if configured to permit this access, can connect directly to the public Internet. There are few design topologies where a database server is connected to the public Internet as a best-practice, but if your use-case requires it, be sure to require SSL encryption to protect the communication of data into and out of your RDBMS.

Encryption at-rest

Data stored within your database may represent sensitive information. It is a security best practice to encrypt not only in-transit data, but also at-rest data. Transparent Data Encryption (TDE) requires SQL Server Enterprise, so often SQL Server data on Amazon EC2 is encrypted using cell-level encryption.

To encrypt at the cell level:

1. Create a Master Key.

   ```
   Create Master Key Encryption by Password='*****'
   ```

2. Create a certificate with a subject.

   ```
   Create Certificate mycert1 With Subject='this is my test subject'
   ```

3. Create a symmetric key and encrypt it with the certificate using AES-256.

   ```
   Create Symmetric Key mykey1 With Algorithm=AES_256 encryption by
   Certificate mycert1
   ```

4. Open the symmetric key for use.

   ```
   Open Symmetric Key mykey1 Decryption By Certificate mycert1
   ```

5. Get a GUID for the Key, and encrypt data with it.

   ```
   Declare @enc uniqueidentifier
   Set @enc=KEY_GUID('mykey1')
   Select encryptbykey(@enc,CONVERT(VARBINARY(256),'the text I want to
   encrypt'))
   ```

6. Decrypt data using the same key.

   ```
   Select convert(int,DECRYPTBYKEY(encryptbykey(KEY_GUID('mykey1'),1))))
   ```

7. Close the key when the operation is done with it.

   ```
   Close Symmetric Key mykey1
   ```

8. Grant control of the key to another user.

   ```
   Grant Control on Certificate :: [mycert1] to [user]
   Go
   Grant View Definition on symmetric key :: [mykey1] To [user]
   ```

Significant performance penalties are associated with cell-level encryption on SQL Server data. First, columns encrypted in this way do not benefit from indexes. In addition, due to decryption tasks in each operation, a simple select with a single encrypted column can see a 20% reduction of performance. This performance impact inversely scales with workload size, potentially up to several multiples of the original execution time. Be sure to factor in the performance ramifications of this design choice in your system planning.

A potential way to mitigate to this impact is to use another column that stores only hashes for the encrypted column. A hash match reflects an identical value.

In addition, TrueCrypt (http://www.truecrypt.org), a third-party open-source application, allows full disk encryption of EBS volumes.

SQL Server Enterprise edition, available on EC2 only through the License Mobility program, provides another encryption option via Transparent Data Encryption (TDE). This system is built on a generated encryption key, which is typically protected by a Hardware Security Module (HSM), which uses either a TPM security module within the server or several USB keys, neither of which are available on EC2. Several workarounds exist to enable TDE, but to date all of them simply guarantee that if an EC2 instance fails, the encrypted data that is stored within is permanently destroyed. Systems, which leverage TDE on EC2, must carefully implement synchronous replication strategies to ensure the replication of data at the application level.

Summary

The AWS cloud provides a unique platform for any RDBMS application, including Microsoft's SQL Server. With capacities that can meet dynamic needs, cost that is based on use, and easy integration with other AWS products like Amazon CloudWatch, the AWS cloud enables you to run a variety of RDBMS applications without having to manage the hardware yourself.

In addition, management features inherent in an RDBMS can be leveraged in the cloud. This paper provided specific information on Microsoft SQL Server, including licensing considerations, using EBS volumes, data and tempdb device management, SQL optimizations, high availability recommendations, backups, log shipping, and mirroring, and security.

For information on Amazon AWS, see http://aws.amazon.com.