UPGRADING TO SQL SERVER 2016

By: Thomas LaRock
Intro

When you ask data professionals their thoughts about upgrading a database server, the responses you get are along the lines of “they are difficult,” and “everything breaks.” When it comes to upgrading a database server, there is a lot of uncertainty. The truth is that the upgrade process itself is straightforward. It is easy to install new software on new hardware and then restore backups. However, there are many moving parts, and any one of them going wrong could result in a poor experience for everyone.

With data and databases being the most critical asset that any company has, it is easy to understand why anyone would want to be extra cautious with regards to upgrades. The secret to a successful upgrade is understanding that small decisions made at the onset can have major ramifications during and after the process is complete.

Upgrades can be either simple or complex. Simple upgrades are just that: simple. You take one server and upgrade in some manner. Complex upgrades should be treated like any other IT project. These upgrades include upgrading clusters or multi-instance servers. But even simple upgrades can be high-pressure situations with lots of moving parts that need to be done in the correct order. Multiple business units may need to be involved in each step of the process, adding to the complexity.

As if that wasn’t enough pressure, you are also going to need to decide at some point if you must rollback to the earlier version. If you do decide to rollback, you need a solid plan in place. Do not leave anything to chance. Don’t think of a rollback plan as a failure option. Think of your rollback plan as a way for your business users to be confident that if something goes wrong that the system(s) can be restored to an acceptable working condition.

About this document

This document will provide an overview of the SQL Server® upgrade process. It will focus on upgrading the database schema and data, the database server instance, and the server operating system. The steps mentioned do not include any details regarding the upgrading or testing of an application that is going to be accessing the upgraded database. Application testing is outside the scope of this document.

You will want to test your application(s) and not just assume it will work perfectly even after the database has been upgraded. I would also advise that you perform the upgrade steps in a non-production environment first. I feel the need to write that out because I often find that common sense isn’t so common after all.

This document is meant to be a guide. You have many options when it comes to upgrades—you can do an upgrade in place, side-by-side, or even a rolling upgrade. No two upgrades are the same. Just because someone else failed (or succeeded) does not mean you will have the same result.

Let’s get started!

Why upgrade?

That is the usual first question whenever an upgrade project is talked about. Someone, somewhere, wants to know why they should take a perfectly good system that runs just fine and make a bunch of changes.

There exist many valid reasons to upgrade to the latest version of SQL Server. Database and system administrators do not take on upgrade projects simply because we like to make changes and watch things break. There are new performance features, security features, and scalability features in SQL Server 2016 that make it worth the time and effort to upgrade.

Here is a short list of reasons why anyone might consider upgrading to SQL Server 2016.
1. New Features in SQL Server 2016

With any new version of SQL Server we always have something shiny to play with. In SQL Server 2016 we have the following new features:

- **Always Encrypted**
- **Dynamic Data Masking**
- **Row Level Security**
- **Stretch Database**
- **Temporal tables**
- **Automatic soft NUMA**
- **Query Store**

We also have enhancements to features introduced in recent versions (links to each):

- **In-Memory OLTP** enhancements
- **AlwaysOn Availability Groups** enhancements
- **Updateable non-clustered columnstore** indexes
- **DBCC CHECKDB** enhancements

For a complete list of all the things in SQL Server 2016 that have been enhanced, check out [SQL Server 2016: It Just Runs Faster](http://example.com).

2. Supportability

End of support is fast approaching for earlier versions. This means no new service packs or updates. Yes, you can purchase extended support, but it is costly. Microsoft® has extended support for Win2008 and SQL 2008, but that doesn't mean it's a good idea to keep using them.

3. Vendor Requirements

You may be using software from a 3rd-party vendor that has strict requirements about which version of SQL Server you can be using. Yes, this goes both ways—it could require newer versions, or it could require older versions.

4. Company or Industry Standard

Some companies may not allow you to be running more than one full major version behind for any software product. And some industries may have those requirements, too. And don't forget the auditors—they like to have their own suggestions.

5. Scalability

The SQL Server engine has had many enhancements in the past ten years to address scalability concerns. I listed a few of those above (Columnstore, Availability Groups, In-Memory OLTP), but the engine itself has been updated to include things like **new cardinality estimation techniques** to help build better query plans based upon the distribution of your data. Upgrading to SQL Server 2016 will bring you greater scalability opportunities than previous versions.
After you have decided that upgrading is something you want, you will need to start putting together a project plan. The simplest plan involves three steps:

1. Pre-upgrade tasks
2. Upgrade tasks
3. Post-upgrade tasks

Seems simple, right? Well, it can be, especially if you take the time to review the details in this document. Let's take a look at the pre-upgrade tasks.

**Pre-upgrade tasks**

Before the upgrade begins, you need to do a lot of legwork. Trust me when I tell you that the extra legwork now will save you headaches later. Here’s my list of things you will want to review before any data is migrated.

**Know your path(s)**

To get to SQL Server 2016, you can upgrade directly from:

- SQL Server 2014
- SQL Server 2012 SP1
- SQL Server 2008R2 SP2
- SQL Server 2008 SP3

If you are running SQL Server 2005 or earlier, you need to upgrade to an intermediate version before you can upgrade to SQL Server 2016.

For those folks running SQL Server 2000 instances (yes, we KNOW you still exist), you are not able to upgrade directly to SQL Server 2016 without first upgrading to an intermediary version. You have two options to choose from when going from pre-SQL Server 2008 versions. The first option is to do an upgrade in place to SQL Server 2008 or SQL Server 2008 R2. The second option is to do a backup (or detach) your database and restore/attach to an instance running SQL Server 2008 or SQL Server 2008 R2. At that point, you will be able to complete the upgrade to SQL 2016.

**Licensing changes**

You should know the changes that may be in place with SQL Server 2016 compared to your current version. As of SQL Server 2012, licensing is done per core, not per socket. But SQL Server 2016 Standard edition does allow for Server + CAL licensing, too. Because of the change from socket to core, in-place upgrades may come with a hefty cost increase. You should check out for yourself all the details on [SQL Server 2016 licensing](#).

Also worth mentioning is that SQL Server 2016 SP1 allows for many features that were once Enterprise-only, such as Availability Groups, data compression, partitioning, Columnstore, etc. You will want to evaluate licensing costs and the list of features now available in Standard edition prior to starting any upgrade project.

**Know your options**

As complex as upgrades may appear, all upgrades can be placed into one of two scenarios: in-place or side-by-side.

**In-place upgrades** are when you upgrade the current instance of SQL Server by running the installation wizard. These are the easiest to perform, but the hardest to rollback. They have the potential for the smallest amount of downtime. There is no need to move to a new database server. The server retains the current name, allowing for applications to connect without any changes (providing the applications support the new version of SQL Server).

**Side-by-side upgrades** are when you install the new version of SQL Server as a new instance on the existing server—or a new server, which is what I prefer, especially for production scenarios—and migrate databases over as necessary. The fresh SQL Server installation allows for better testing of the system before bringing it online for production. It also allows for more options for rollback. The applications may need to be redirected to the new server, but the use of DNS aliases can reduce that headache.

There is also the concept of a rolling upgrade. This is when you use a high-availability feature such as mirroring, clustering, or Availability Groups. The idea is that you upgrade a secondary node, failover, and continue upgrading all nodes in this manner until you upgrade the primary node, and then fail back if needed.
During an upgrade, some downtime may be required. Rolling upgrades can minimize, but are not always guaranteed, to eliminate downtime. The less downtime allowed, the more expensive the project generally is, as it includes several different types of resources—human and physical.

I’m a huge fan of rolling upgrades, but only when it is necessary to minimize downtime due to a business requirement. If no such requirement exists, then my preference is to do a side-by-side upgrade and migrate the data by taking a backup and restoring it to the new server. But that is my preference; that doesn’t mean it must be yours. You do what’s best for your shop.

Gather inventory details
You will want to collect information about the servers and the database instances that are considered in scope for the upgrade. There are many tools available to help you collect these details such as [Microsoft Assessment and Planning (MAP) Toolkit](https://www.microsoft.com/en-us/download/details.aspx?id=52520) and [SQL Power Doc](https://www.sqlpowerdoc.com), as well as 3rd-party tools.

Gathering a list of server and database names may not be enough. You will want to collect details about the databases as well. Even a simple count of tables is a valuable piece of information to have. If you have 873 tables at the start of the upgrade, then you will want to make sure you have 873 when you are done. Same for stored procedures. You should consider collecting details at the column level, too. You will want to make certain that datatypes and collations are intact, that view definitions haven’t changed, etc.

You also need to collect details on the in-house and 3rd-party vendor applications using the database server. You will want to list out the availability requirements for every application using the instance and you need to include the applications that are connecting remotely.

Also worth noting is that applications can have specific configurations applied to the server O/S, database instance, and the database itself. You need to know what non-default configurations are in use by the application using the instance.

This might seem like overkill for many reading this, but if you have ever had the unpleasant situation where a table was missed because of a migration and/or upgrade, you will know it can be quite valuable.

Data Migration Assistant
Formerly known as the SQL Server Upgrade Advisor, the [Data Migration Assistant](https://www.microsoft.com/en-us/download/details.aspx?id=52520) (DMA) will help to identify any breaking or behavioral changes as well as deprecated features. You can use DMA to identify issues that need to be resolved before upgrading to the desired version of SQL Server.

You should be aware that the DMA is similar to a consultant: it doesn’t fix everything that is wrong, but merely advises you on what actions you should take when upgrading to SQL Server 2016. The actions the DMA recommends will come in two forms: those actions to be done prior to a migration, and those actions to be completed post-migration. The DMA is really good at finding what I call the “stub-your-big-toe” things that need fixing prior to a migration. But it is not foolproof, it may not identify every last detail. You will need to play the role of an actual DBA when migrating to a new version. Many of the items below will help you to do just that.

The DMA can also help identify parts of your database that may benefit from new features in SQL Server 2016, as well as migrate your data for you.

You could also decide to use Visual Studio to migrate a database to a new version of SQL Server. You can create a database project, change the build to the desired version (including backwards, if you needed to revert to a previous version), publish to the target server, and then export/import the data.

Deprecated features
With each new version of SQL Server, there are some features that are marked as deprecated. Deprecated does not mean the features have been removed. Deprecated means that the features will possibly be removed in a future version, and you should not use these features for any new development work. You can find the list of deprecated database engine features for SQL Server 2016 [here](https://docs.microsoft.com/en-us/sql/previous-versions/sql-server/sql-server-2016/what's-new/deprecated-features).

Discontinued features
With SQL Server 2016, Microsoft has started publishing a list of items that are discontinued. This means the items have been removed completely. You can see the list of discontinued items [here](https://docs.microsoft.com/en-us/sql/previous-versions/sql-server/sql-server-2016/what's-new/discontinued). You should review these items and make certain your applications are not relying on a feature that will not be there.
Breaking changes

Did you know that Microsoft publishes a list of breaking changes for each version of SQL Server? Well, you do now. You should review them to the point that they are familiar to you. You don’t have to memorize them all, just be familiar with them so that if something odd happens you can think to yourself “... hey, is this odd behavior listed in the breaking changes section of the Books Online (BOL)?” I would like to believe that the DMA will alert you to many of these breaking changes, but the truth is, the DMA is not as dynamic as the BOL. That means the BOL may have an entry or two that doesn’t make it into the DMA checklist, and that is why you should review this section.

Behavioral changes

Previous versions of SQL Server have published a list of behavioral changes for the database engine. Similar to the breaking changes, the behavioral changes are changes that could still affect you in an adverse way. They are worth reviewing, and they are also things that the DMA is likely to never report back to you about, because they aren’t things that will break, but merely things that could break.

However, I was not able to find a BOL entry for SQL Server 2016 for the database engine. There are pages for Analysis Services, Integration Services, and Reporting Services. I suspect that there could be one coming for the database engine at some point as issues are tracked and become known. You could always review the previous versions over at the SQL Server 2012 page and use that as a starting list.

Read the release notes

Because you’re a geek, that’s why. Take a few minutes and read the release notes (link). No, they aren’t as funny as the release notes for apps on your phone, but they can be useful for you to review anyway. It’s good to have as complete a picture as possible for the new version, should something not work as expected, and there are details in the release notes you may not find elsewhere.

New environment requirements

SQL Server 2016 can be installed on Windows Server® 2012/R2, so you may need to update your server O/S as part of the upgrade to SQL Server 2016.

Microsoft lists the minimum requirements for installing SQL Server 2016 on this page. However, those are the minimums there. Chances are if your servers don’t already meet those requirements then you aren’t looking to upgrade anytime soon anyway. But if you are upgrading, then it might be time to upgrade your hardware as well. Heck, you may even consider going virtual (if you aren’t already), which will still require you to examine your hardware requirements.

But here’s the real reason you will want to upgrade your hardware: new features. Let’s say that you are thinking of upgrading to SQL Server 2016 in order to take advantage of Hekaton. Considering there are a lot of shiny new things in SQL Server 2016, you’ll want to do the extra legwork here to scope out what hardware you’ll need. Otherwise you won’t be able to leverage many of the new features.

Take baselines

You will want to collect performance baselines before you begin the upgrade process. If you don’t, you won’t have any way of knowing if performance is better or worse when the upgrade is complete. Since each SQL Server implementation is unique, there will be different performance metrics that are important to you and your business users.

You must also include a baseline of the current server operating system. Perfmon works well for this, but there are many 3rd-party tools that can capture these details as well. If you are using Perfmon, here is a possible list of useful counters for you to track over time to use as a baseline:
For CPU usage:
- `\Processor(_Total)\% Privileged Time`
- `\Processor(_Total)\% Processor Time`
- `\Process(_Total)\% Processor Time`
- `\Process(_Total)\% Processor Time`
- `\Process(_Total)\% Processor Time`
- `\Process(_Total)\% Processor Time`
- `\MSSQL$SQL:SQL Statistics\SQL Compilations/sec`
- `\MSSQL$SQL:Databases(_Total)\Transactions/sec`

For physical memory:
- `\MSSQL$SQL:Buffer Manager\Checkpoint Pages/sec`
- `\MSSQL$SQL:Buffer Manager\Page Life Expectancy`
- `\MSSQL$SQL:Buffer Manager\Lazy writes/sec`
- `\MSSQL$SQL:Memory Manager\Memory Grants Pending`
- `\Memory\Available Mbytes`

For physical disk:
- `\PhysicalDisk(_Total)\Avg. Disk sec/Read`
- `\PhysicalDisk(_Total)\Avg. Disk sec/Write`
- `\PhysicalDisk(_Total)\Disk Transfers/sec`
- `\PhysicalDisk(_Total)\% Idle Time`

For network:
- `\MSSQL$SQL:SQL Statistics\Batch Requests/sec`
- `\Network Interface\Bytes Total/sec`
- `\Network Interface\Current Bandwidth`

You can output these counters to a file and then use the Performance Analysis of Logs (PAL) tool to analyze the output. Or, of course, you could use 3rd-party tools to do all of the collection and analysis for you, too.

You will also want to know how your system will be expected to grow over time. For physical implementations, this means you will try to size the physical server for end-of-life expectations. For virtualized servers, you will try to size for your performance needs now and expand later as needed.

Capture workloads

You can use the Distributed Replay feature to capture a production workload from a source server and replay it on a target server. Doing so will help to assess the impact of upgrading SQL Server by comparing the workload performance against both systems. Distributed Replay is most useful for scenarios that have high concurrency and a single client cannot simulate the workload properly.

The Database Experimentation Assistant is a new tool currently available in Technical Preview. It uses Distributed Reply along with R services to give the user a way to do A/B testing of workloads. Using statistical analysis of workloads allows for greater confidence when upgrading to newer versions of SQL Server.

Testing the Server O/S

You can use tools like iPerf® and DskSpd to test the server network and disk performance to verify it is as expected before installing SQL Server. These tools are good at helping to identify if there are any possible configuration issues with the network and disk layout. It is better to check for such issue now, before the installation of SQL Server 2016 begins.
Take backups

Before you start any upgrade process, make certain you take backups of everything: databases, application files, and the server O/S. Sometimes you can utilize a VM snapshot (or checkpoint) to help with this process. I would recommend that when it comes to backups to consider the Computer Backup Rule of Three.

Also worth noting: backups are only good if they can be restored. So, you will want to test the restore process before you move forward with upgrading.

In the event of a rollback during the upgrade process, decisions must be made as to how to handle the amount of data that could potentially be lost. For example, if you are running a production-parallel scenario, the business may need to redo a full day's worth of data entry. It's better to have those discussions now, not later.

Also worth mentioning at this point is the use of Azure Virtual Machines (VMs) for proof-of-concept (POC) testing prior to performing an upgrade. Azure makes it easy to spin up a VM, use it as needed, and then get rid of it. It is far cheaper than buying new hardware, and faster than waiting for new hardware to be order, shipped, and installed.

Upgrade tasks

I mentioned earlier that there are only two types of upgrades: in-place or side-by-side. With in-place upgrades, there is no need to worry about the transferring of data. Side-by-side upgrades will require you to move data from one server to another. There are four main options for you to consider using for data migrations:

- **Backup and restore.** Good option for smaller systems and if you want piecemeal migrations. You may also consider detach and attach here.
- **Pre-staging** the data using full, differential, and transaction log backups to minimize the data transfer. Log shipping is also a consideration.
- **Database mirroring.** This allows for easy migration of data from the old system to the new.
- **Availability Groups.** More complex than database mirroring because multiple databases can be involved.

The concept of rolling upgrades was mentioned earlier. This is when you use a high-availability feature such as mirroring, clustering, or Availability Groups. The idea is that you can upgrade a secondary node, failover, and continue upgrading all nodes in this manner until you upgrade the primary node, and then fail back if needed.

Let's look at the steps involved for each.

**Steps for an in-place upgrade**

In-place upgrades are the easiest to perform, but the most difficult to rollback should there be any issues. The steps involved in an in-place upgrade are as follows:

- Verify that backups exist for all databases (user and system). If you have a database that is not in SIMPLE recovery mode, make certain a transaction log backup exists. Verify that these backups can be restored.
- Review the list of prerequisites for SQL Server 2016, and install whatever is needed.
- Run the SQL Server 2016 installation media.
- Perform your post-upgrade tasks.
- Test, test, and test that everything is working as expected.

**Steps for a side-by-side upgrade**

Side-by-side upgrades have more steps and are considered more complex. But they also give you more flexibility for rolling back, because you are not going to be touching the original system while it is still in use.
The steps involved for a side-by-side upgrade are similar for both an existing and new database server. The only difference is that for a new server, you will need to install SQL Server. Here are the steps:

- Verify that backups exist for all databases (user and system). If you have a database that is not in SIMPLE recovery mode, make certain a transaction log backup exists. Verify that these backups are able to be restored.
- Script out any and all necessary system objects.
- Script out any and all necessary SSIS packages (either from MSDB or as flat files).
- **For a new instance on a new server:**
  - Review the list of prerequisites for SQL Server 2016 and install whatever prerequisites are needed.
  - Install the desired version and edition of SQL Server 2016.
- Use script(s) from old server to create necessary system objects on the new server.
- Migrate SSIS packages to MSDB (or as flat files, if applicable).
- Select database(s) to migrate, take offline.
- Migrate data to new instance. Repeat for each database.
- Perform your post-upgrade tasks.
- Test, test, and test that everything is working as expected.

### Steps for a Rolling Upgrade

Rolling upgrades can minimize downtime during upgrades. Rolling upgrades using database mirroring is my preferred method for doing SQL Server upgrades. But we could also use log shipping or Availability Groups—the choice of which feature you want is up to you. Just make sure you have a solid rollback plan for whichever feature you are using.

The steps involved for rolling upgrades are as follows:

- Choose your high-availability method (log-shipping, mirroring, Availability Groups).
- **Choose one of the following:**
  - Upgrade one of the secondary nodes following the in-place upgrade instructions above
  - Install SQL Server 2016 on a new server (and add it as a node if applicable)
- Fail over to that secondary node.
- Perform any post-upgrade tasks.
- Test, test, and test that everything is working as expected.
- Repeat the upgrade for any remaining secondary nodes.
- Perform any post-upgrade tasks.
- Test, test, and test that everything is working as expected for each node.
- Repeat the upgrade for the primary node.
- Perform any post-upgrade tasks.
- Test, test, and test that everything is working as expected for the primary node.

With rolling upgrades, you don't have to fail back to the original server (the primary node). It is perfectly fine to configure database mirroring just for the single purpose of a rolling upgrade. After you have failed over to the secondary, you then break the mirror and remove the server from your inventory. This is the same result as a side-by-side migration, but with less downtime than doing the traditional method of backup/restore or detach/attach. And for very large databases (VLDBs), this concept is crucial, because restoring can be a cumbersome task.

It is important to note that data movement in a rolling upgrade is in one direction only. You can migrate from an older version to a newer version of SQL Server, not the other way around. So if you are doing a rolling upgrade and you move your data to an upgraded node, you can't go back without recovering from backups on the original server. You will see error messages indicating this as well, so don't be alarmed when they start. Just recognize that SQL is telling you that you cannot migrate down to an earlier version.
**Post-upgrade tasks**

After the upgrade is complete, you will need to perform a series of tasks to verify that database is ready to be handed over to the end-users for further testing.

**Take backups**

Right now. Before you do anything else. You’re a DBA. Backups should be in your DNA. You should have taken one prior to the start of any upgrade or migration, and you had better take one right now, and again before you turn that database over to your end-users. Also, you should save any output from the items listed here, as it could prove helpful should something go awry later. (bonus item – **make sure your backups are good!**)

**DBCC CHECKDB**

One of your post-migration or upgrade tasks should be to run the following statement:

`DBCC CHECKDB WITH DATA_PURITY;`

This statement will check your data for values that are no longer valid for the column datatype. For databases created prior to SQL 2005 (and you know they are still out there), this step is rather important to take. For databases created in SQL 2005 and later, the `DATA_PURITY` check is supposed to be done automatically with a regular CHECKDB.

But what about a database that was created in SQL 2000, migrated (poorly) to a SQL 2008 instance, and left in the SQL 2000 (80) backward compatibility mode? What about that little feller? Do you want to assume that the `DATA_PURITY` check has been getting done? Here’s a thought: just go run it yourself anyway. That way you know it is getting done.

Also worth noting here that column integrity checks are not performed when the `PHYSICAL_ONLY` option is used.

**DBCC UPDATEUSAGE**

While not as critical as the `DATA_PURITY` command noted previously, this one still has a place in any migration or upgrade process:

`DBCC UPDATEUSAGE(db_name);`

This command will help fix any page count inaccuracies that are resulting in the `sp_spaceused` stored procedure returning wrong results. The best time to run this command is now, but be aware that it can take some time to run depending upon table or database size. Ideally you would run this on a regular basis for one of the following reasons:

- You suspect that you are seeing incorrect values returned for `sp_spaceused`.
- Your database has a high volume of DDL statements (CREATE, ALTER, or DROP).

**Updating statistics**

This one is **not** to be skipped and is simply a **MUST** for any migration or upgrade checklist:

`USE db_name; GO EXEC sp_updatestats;`  

This command will update the statistics for all the tables in your database. It issues the `UPDATE STATISTICS` command, which warrants mentioning because you may want to use that command with the `FULLSCAN` option. I'm the type of person that would rather be safe than sorry and therefore would end up running something like this:

`USE db_name; GO EXEC sp_MSforeachtable @command1='UPDATE STATISTICS ? WITH FULLSCAN';`

Bottom line: don’t forget to update the statistics after an upgrade. Failure to do so could result in your queries running slowly as you start your testing and may end up wasting your time while you try to troubleshoot the possible bottlenecks. With SQL Server 2016, there is also a new Cardinality Estimator (CE). Since the query optimizer relies on accurate statistics for plan estimation purposes, you will want your statistics to be as accurate as possible before you begin any testing. Take care of the stats now and you won’t have to worry about them later.

**Refresh view definitions**

Believe it or not, every now and then, someone will build a view that spans into another database on the same instance. And, in what may be a complete surprise to many, sometimes these views will go across a linked server as well. The point here is that your view may not be of data that is contained in just the database on that single instance. In what could be the most dramatic twist of all, sometimes these views are created using a `SELECT *` syntax.
I know, I know... what are the odds that you could have such code in your shop? But it happens. And when you have bad code on top of views that go to other databases (or views of views of views of whatever else some sadistic person built), you are going to want to use `sp_refreshview` to refresh those views.

So, if you are migrating a database in your environment to a new server, then it would be a good idea to refresh your views using `sp_refreshview`. Most of the time, it won't do anything for you—just like a burger topped with [veggie bacon](#). But there is that one chance that it will dramatically improve performance and your customer will be happy as a result. Using `sp_refreshview` is a lot like flossing: it doesn't take much effort, and the end result is usually worth it.

**Check compatibility levels**

If you have been going through SQL Server upgrades for the past ten years, then you are likely to have noticed that the compatibility level does not get set to the newest version after the migration is complete. You need to manually set the compatibility level yourself. With SQL Server 2016, this becomes more important than in previous versions due to the new Cardinality Estimator (CE).

There is a great white paper from Joe Sack that details the good, the bad, and the ugly with the new CE. The TL;DR version of the whitepaper is this: you'll want to take advantage of the new CE except for the times when you won't. Part of this is knowing which compatibility level you are using. I'd recommend you update every database on the SQL Server 2016 instance to compatibility mode 130 and test, test, test. (This assumes that you have baselined performance for your critical queries prior to the migration, so that you can verify if the new CE is working for or against you.)

**Verify counts of objects**

Remember the counts of objects such as tables and stored procedures that you took before? Now is when you want to review those counts. Make sure you have the same number of objects that you started with prior to the upgrade and migration. Remember the SQL Server upgrade motto: no table left behind!

**Check configurations**

As part of the pre-upgrade tasks, we collected details on the in-house and 3rd-party vendor applications using the database server. We also collected information about the specific configurations applied to the server O/S, database instance, and the database itself. Now is the time for you to review those details and make sure the configurations have been applied to the new server.

At this point, I will tell you that the use of a POC test system saves you a lot of time with the "after" phase, because you work through any issues early on in testing and incorporate them into your upgrade plans. Also worth mentioning again is how easy Azure makes this for you.

This is also a good time to note that sometimes it is worth running "production parallel," where you have two systems running at the same time, with both considered production. How the data is kept in sync is up to you, but the idea is that the business users get a chance to verify that the new system is working as expected.
Summary

Upgrades are a necessary part of any development lifecycle. The chances of having a successful upgrade increase proportionally with the amount of planning and preparation you invest in building a proper upgrade process. If you are planning to upgrade to SQL Server 2016, you can use this document as a guide to help put together your checklist.

If you haven't started building up your SQL 2016 migration or upgrade checklist yet, now is the time, and get these items included. They will save you pain, I promise.

Here's a list of tips and tricks you will find useful:

- Rolling upgrades are awesome, but you only get to go one way: upwards. However, these are great for large systems or systems that need minimal downtime.
- New server installs and backup/restore detach/attach are easy to do as well. They work for small-mid size databases but require downtime for cutover.
- Simple counts of objects and comparing datatypes/collations is worth the effort.
- Script everything you can.
- Consider Azure for POC.
- Microsoft offers the SQL Server Upgrade Target Decision matrix to help you understand what target environment is best for your needs: Upgrading/consolidation/physical/virtual/cloud.
- DMA is not perfect, but still good.
- Use tools to assess inventory; script everything.
- Performance baselines for when users complain about slow performance.
- Build a checklist; top queries.
- Use Query Store to freeze/unfreeze plans.

Database Performance Analyzer

Understand how database upgrades impact application performance. Don't define a successful upgrade just by the narrow scope of is it functional or not. Get the full performance picture of your database environment before and after you upgrade with Database Performance Analyzer.
Author

Thomas LaRock is a Head Geek at SolarWinds and a Microsoft® Certified Master, SQL Server® MVP, VMware® vExpert, and a Microsoft Certified Trainer. He has over 15 years’ experience in the IT industry in roles including programmer, developer, analyst, and database administrator.

REFERENCES

- [http://tinyurl.com/msdn-always-encrypted](http://tinyurl.com/msdn-always-encrypted)
- [http://tinyurl.com/msdn-stretch-db](http://tinyurl.com/msdn-stretch-db)
- [http://tinyurl.com/msdb-temporal-tables](http://tinyurl.com/msdb-temporal-tables)
- [http://tinyurl.com/sql2016-sp1-released](http://tinyurl.com/sql2016-sp1-released)
- [http://tinyurl.com/sql2016-requirements](http://tinyurl.com/sql2016-requirements)
- [http://tinyurl.com/install-config-check](http://tinyurl.com/install-config-check)
- [http://tinyurl.com/upgrade-decision-matrix](http://tinyurl.com/upgrade-decision-matrix)