**Verify SQL Backups with SQL Powershell**

One of our main responsibilities as DBAs is to ensure our backups are working properly. This can be a time consuming task each day. However, if you automate this task with SQL Powershell, you can check all your backups in minutes, whether you have ten SQL servers or 100. I’ve outlined an automated backup verification process below. Having monitoring on your failed SQL Agent jobs or some other type of monitoring is great, but sometimes when jobs hang, they don’t notify you. By going directly to the SQL metadata each morning and pulling the backup info for each server, you get a true picture of how your backups are performing.

Although this article is specific to verifying SQL database backups, this same process can be used to gather all types of useful information DBAs want to know about the SQL Servers they support (patch levels, version, db file size, ETC). The following articles on SQL and Powershell helped me tremendously putting this process together. I was inspired by their ideas and took what they taught me and added some ideas of my own and put together this process. I hope it is useful to you!

**Helpful Powershell Articles:**

Chad Miller: <http://blogs.technet.com/b/heyscriptingguy/archive/2010/11/01/use-powershell-to-collect-server-data-and-write-to-sql.aspx>

Ben Miller: http://www.sqlservercentral.com/stairway/91327/

Tim Radney: <http://timradney.com/2014/02/17/how-to-check-for-last-sql-server-backup/>

Luca Sturlese: <http://9to5it.com/powershell-script-template/>

**Here is a high level summary of the 4 step process:**

1. Create a ServerList table that contains a list of the servers you want to check.
2. Create a backup table where you will store all your database backup data from each of your servers.
3. Run the CheckBackups.ps1 PowerShell script. It will connect to each instance in your ServerList table one by one, run a sql script which will gather backup information about your servers, store the output in a PowerShell “in memory table”, then it will write all that data into the backup table you created in step 2.
4. Check your backups by querying your backup table to verify if your backups were successful.

**Detailed Summary of the Process:**

1. **Create a table that contains a list of your servers.**

Here are the columns I used for my table, but your Serverlist table doesn’t need to be this detailed; or it can be even more detailed depending on your needs. All that is required is the SQLConnectionString column. I like to have the columns below to keep track of a lot of other information about my servers. Once the table is created, populate it with your server info. I use something called SQLConnectionString which contains the machine name and port number. For example, sqlserver01,1433. I have a few other columns for joins, which we’ll go into in Step 4.

CREATE TABLE [dbo].[ServerList](

 [MachineName] [nvarchar](255) NULL,

 [AssociatedApp] [nvarchar](255) NULL,

 [App\_ID] [nvarchar](255) NULL,

 [SQLConnectionString] [nvarchar](255) NULL,

 [InstanceName] [nvarchar](255) NULL,

 [IsActive] [nvarchar](255) NULL,

 [Domain] [nvarchar](255) NULL,

 [Environment] [nvarchar](255) NULL,

 [PDBA] [nvarchar](255) NULL,

 [SDBA] [nvarchar](255) NULL

) ON [PRIMARY]

1. **Create a backup table using the attached script.**

This table needs to have these exact data types. If it doesn’t, the PowerShell script will fail on the insert. The bottom 5 columns are only there because PowerShell created these when it gathers the info from the backup query mentioned in Step 3. I’m pretty new to SQL PowerShell and I’m not sure why these are created, but they aren’t hurting anything, so I just added them so the insert won’t fail.

CREATE TABLE [dbo].[Backups](

 [DatabaseName] [nvarchar](255) NULL,

 [RecoveryModel] [nvarchar](255) NULL,

 [LastFullBackup] [nvarchar](255) NULL,

 [LastDiffBackup] [nvarchar](255) NULL,

 [LastTranLogBackup] [nvarchar](255) NULL,

 [2ndToLastTranLogBackup] [nvarchar](255) NULL,

 [CheckDate] [datetime] NULL,

 [DaysSinceLastBackup] [nvarchar](255) NULL,

 [MachineName] SQL\_Variant NULL,

 [ServerName] SQL\_Variant NULL,

 [InstanceName] SQL\_Variant NULL,

 [ComputerNamePhysicalNetBIOS] SQL\_Variant NULL,

 [RowError] varchar(255) NULL,

 [RowState] varchar(255) NULL,

 [Table] varchar(255) NULL,

 [ItemArray] varchar(255) NULL,

 [HasErrors] varchar(255) NULL)

1. **Run the CheckBackups Powershell script to Gather backup info from Your Servers.**

Before you can run the CheckBackups.ps1 script, you’ll need to download some files. First, the Backups.SQL script which comes mostly from Tim Radney’s sql blog <http://timradney.com/2014/02/17/how-to-check-for-last-sql-server-backup/>. His coding is so clean and gathers such great information. Wish I could code like that! Thanks Tim!!! I added a few things to his script like checkdate and some ServerProperties so when I load it into the Backup table, I am able to see which server is associated with the db backup info.



You’ll also need to download the four functions below because the CheckBackups.ps1 script will need to dot source these when it runs. If you are not sure what dot sourcing means, it simply means my CheckBackups.ps1 file relies on the code in these functions. To understand more about the logging function, and overall neatness in Powershell coding, check out this article written by Luca Sturlese: <http://9to5it.com/powershell-script-template/> . Thanks Luca!!!

To understand more about the other three functions, do yourself a favor and spend some quality time with this article by Chad Miller. <http://blogs.technet.com/b/heyscriptingguy/archive/2010/11/01/use-powershell-to-collect-server-data-and-write-to-sql.aspx>

Seriously, changed my life as a DBA. I learned how to utilize the out-datatable and write-datatable functions from this article written in 2010. He shows us how to pull data directly from SQL and write directly to SQL without using any text files. If you are reading this, THANK YOU CHAD!!! : )



Once you’ve downloaded the backups.sql script and the four functions above, make note of where they are on your system so you can update the CheckBackups.ps1 script with their locations.

It’s almost time to check our backups! We just need to download the CheckBackups.ps1 script and then update a few items. Download the CheckBackups.ps1 file and open it up to edit. I prefer using Windows Powershell ISE for my editing.



Go to line 29-32 and update these locations with wherever you downloaded your functions.



Go to line 39 and 40 and update it so your logging will write to the location you want it to.



Go to line 58 & 59 and update your SQL Server and database info where you have your serverlist table.



Go to line 68. This section contains some information you will need to update. For example, if you are using windows authentication for this process, and you are logged on with your windows account which has access to your SQL Servers, you can delete the username and password lines on 71 & 72. However, be extremely careful what you put in your sql script in your input file on line 73. If you are running under your account and you have sa permissions and there is some type of DDL in there, it will run it on every single server in your serverlist. Imagine running a DELETE or UPDATE or worse by mistake. It might be wiser to create a sql account that has limited read only permissions to run tasks against all your SQL Servers. That’s what I decided to do to be safe.



Go to line 90 and update your SQL Server where you want to write to…



Now you are ready to run CheckBackups.ps1. You may want to just test this out on a couple of your test systems at first. You can do that by going to line 60 to narrow your list of servers from your serverlist and modifying your query like this…



If you are not having success with populating your backups table, you may not have created your backups table with the correct columns and data types. Another reason could be you’re not able to connect to your servers. Troubleshoot by using your log output and don’t give up!

1. **Query your backup table.**

Now that your Backups table is populated with useful information, use these queries to find valuable information…

1. No Backups

SELECT \* FROM backups

WHERE DaysSinceLastBackup = 'NEVER'

2. Which databases haven’t been backed up recently?

SELECT SL.MachineName, SL.AssociatedApp, SL.Environment as ENV, SL.PDBA, SL.SDBA, BU.DatabaseName, BU.LastFullBackup, BU.CheckDate, BU.DaysSinceLastBackup

FROM Backups BU

JOIN ServerList SL

ON BU.MachineName = SL.MachineName

AND datediff(dd, getdate(),checkdate) >= -0 -- -0 will give us today. -1 will give us --yesterday, and so on.

AND SL.Environment = 'PROD'

AND BU.DaysSincelastBackup <> ' 0'

AND BU.DaysSinceLastBackup <> ' 1'

AND BU.DaysSincelastBackup <> ' 2'

ORDER BY MachineName

3. Gives us a quick look at all of our prod servers sorted by the most out of date backups

SELECT SL.MachineName, SL.AssociatedApp, SL.Environment as ENV, SL.PDBA, SL.SDBA, BU.DatabaseName, BU.LastFullBackup, BU.CheckDate, BU.DaysSinceLastBackup

FROM Backups BU

JOIN ServerList SL

ON BU.MachineName = SL.MachineName

AND datediff(dd, getdate(),checkdate) >= -0

AND SL.Environment = 'PROD'

ORDER BY DaysSinceLastBackup

DESC

1. Full Recovery, no log backups?

select \* from backups

where RecoveryModel = 'Full'

and LastTranLogBackup = 'No Log'

1. Missing from your daily backup collection? May indicate a connection issue from the ps1 process to the server or that the server just doesn’t have any backup info. Could be a DR server.

SELECT DISTINCT SL.MachineName

FROM DBAUTIL.dbo.ServerList SL

WHERE IsActive = 1

AND NOT EXISTS ( SELECT 1

 FROM DBAUTIL.dbo.Backups BU

 WHERE SL.MachineName = BU.MachineName

 AND datediff(dd, getdate(),bu.checkdate) >= -0

 )

Hopefully this process will help improve your life as a DBA by saving you time and energy while ensuring your databases are safe and able to be recovered!

I’d like to acknowledge a couple of my co-workers who were crucial in helping me with this process. Vernon Crandall helped me understand general coding practices and Joseph Mills helped me work through some logging challenges. Thanks Guys!