SQL Server Replication Monitoring

Documentation

Table of Contents

1 Introduction

This document describes simple solution for monitoring of replication subscriptions. When it is crucial for your use case replication shoul be always functional. When there is some problem you should be informaed accordingly and react to it ASAP to bring your environment to stable state. Whole restore monitoring is described in details further in document. Way of monitoring can differ from your own proccess, but that is not definitely bad it is only my approach to doing so and can be wrong, but I have some bullet proof arguments, so feel free to start discussion.

Tested on SQL Server versions >= 2008, so all older versions are not supported and you are running scripts/procedures on your own risk!

2 Technical preview

Whole solution consist of one view and one stored procedure created in distribution database. Procedure can be called directly or from SQL Agent job steps. View is just for simplyfying code as it contains longer SELECT statement querying system tables in distribution database. Stored procedure is querying this view and deciding based on seuncribers state or warnings if to rise alert or not. By default you can see not properly working subscriptions in result set.

- **v_ReplicationMonitorData** – querying systems table
- **usp_ReplicationMonitor** – logic to decide if rise alert based on monitoring data from above view

For regular checking SQL Agent job is created, it only contains one step for calling above mentioned stored proceedure and send notification to given emails via given database mail profile. You have to properly configure Database Mail if you want to use notification on regular basis.

Nice article about configuration of Database Mail can be found on Brent Ozar's website.

All important information included in every procedure header also, for example any versions info and small release notes can be found there also

```
/*
Purpose: This procedure can be used for regular checking of replication status on distribution server. It is using builtin stored procedure sp_replmonitorhelpsubscription.
    Returning table of subscriptions that are not working properly based on agreed rules. Use output parameter to react accordingly in workflow where called.

Author: Tomas Rybnicky trybnicky@inwk.com
Date of last update:
    v1.0 - 27.11.2019 - final state where all functionality tested and ready for production
List of previous revisions:
    v0.1 - 27.11.2019 - initial release of this stored procedure

Execution example:
    DECLARE @RiseAlert BIT
    EXEC [distribution].[dbo].[usp_ReplicationMonitor] @p_RiseAlert = @RiseAlert OUTPUT
    SELECT @RiseAlert
*/
```

Figure 1 - procedure header info

3 Deployment

Only thing you have to do is to copy deplyemnt script from its storage on GitHub. Copy script to SQL Server Management Studio and run it aganst SQL Server instance you are connected to or use multiquery from Registered Servers. Running script using multi-query is especially benefical when creating procedures on AG replicas, you will avoid unnecesarry clicking when connecting to every replica and running one by one.

Just set script run variables to match your environment needs. Find folowing code at the beginning of deployment script

```
29  -- you can change folowing variables according to your needs
30  SET @AlertRecipients = '<your email addresses here>'       -- specify recipients of email notifications
31  SET @DbMailProfile   = '<your database mail profile here>'  -- specify name of your configured database mail profile
```

You can also change script body if there is something that you do not like there, but only by your own responsibility!

Deplyment script is doing nothing magical, only creating one view and one stored procedure in mdistribution database. Firts it checks if object already exists and drop and re-create it. **So please be aware of overwriting of your objects if you have same namming.** You can also find some important info in script header, reather to read it before running anything against your servers, you should be aware of what you are doing also on non-production servers.

4 After deployment

4.1 Direct messages

After proper execution you can check messages for detailed steps which have been done over instance and also for possible related error messages.

```
SQL Server Replication Health Monitoring - deployment of solution
------------------------------------------------------------------------
STEP : Created view  [dbo].[v_ReplicationMonitorData] in distribution database.
STEP : Created stored procedure [dbo].[usp_ReplicationMonitor] in distribution database.
STEP : SQL Agent job "Warning: Replication Health", it is not scheduled.
------------------------------------------------------------------------
SQL Server Replication Health Monitoring - deployed to SQLREPPROD04
```

Figure 3 Script messages

4.2 Views

You can see new view dbo.v_ReplicationMonitorData created in dstribution database.
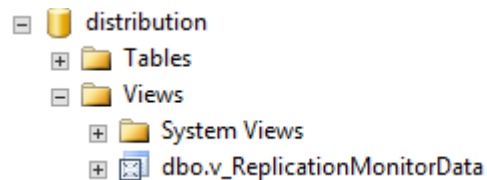

Figure 2 View in distribution database

4.3 Stored procedures

You can see two new stored procedures in master database + CommandExecute procedure from Ola Halengreen (if not there already)
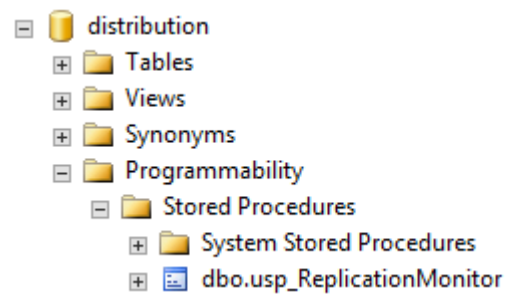

Figure 4 Stored procedures in master database

4.4 SQL Agent jobs

You can see SQL Agent job created with name "Warning: Replication Health" containing one step for calling stored procedure **usp_ReplicationMonitor**. Just keep in mind it is created without any schedule so you have to pick what fits best for your requirements.
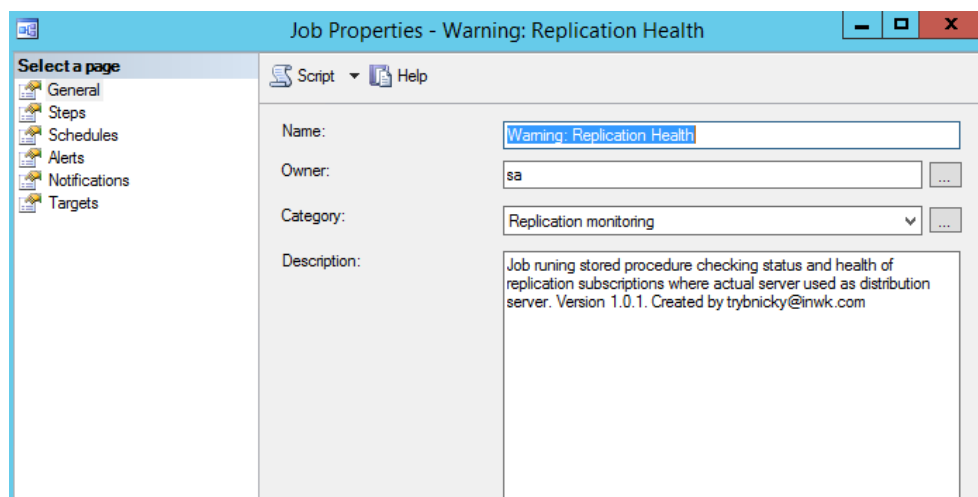

Figure 5 SQL Agent job detail

Only one step inlcuded in this SQL Agent job is running folowing T-SQL code to call stored procedure **usp_ReplicationMonitor** and report problem using database mail.

```sql
SET NOCOUNT ON
DECLARE @RaiseAlert BIT
DECLARE @AlertResultTable NVARCHAR(MAX)
DECLARE @Subject NVARCHAR(512) = 'Warning: Replication Health ' + @@SERVERNAME
DECLARE @Recipients NVARCHAR(512) = 'trybnicky@inwk.com'

EXEC [distribution].[dbo].[usp_ReplicationMonitor]
            @p_SuppressResults = 1,
            @p_HTMLTableResults = @AlertResultTable OUTPUT,
            @p_RaiseAlert = @RaiseAlert OUTPUT

IF @RaiseAlert > 0
BEGIN
            DECLARE @EmailBody NVARCHAR(MAX)

            SET @EmailBody = '<html><body>'
            SET @EmailBody += '<p>Some replication requires your attention. Below
listed subscriptions are not working properly.
<br><br>Run EXEC [distribution].[dbo].[usp_ReplicationMonitor] to check actual
state
<br><br>Problematic subscriptions:</p>'
            SET @EmailBody += @AlertResultTable
            SET @EmailBody += '<html><body>'

            EXEC msdb.dbo.sp_send_dbmail
                        @profile_name = 'mail.iwprint.com',
                        @recipients = @Recipients,
                        @subject = @Subject,
                        @body = @EmailBody,
                        @body_format = 'HTML',
                        @exclude_query_output = 1,
                        @importance = 'High'
            PRINT 'Replication problems reported to ' + @Recipients
END
ELSE
BEGIN
            PRINT 'All replication subscriptions are working properly.'
END
```

5 Execution of stored procedures

OK so you are all set now and you can start enjoying new stored procedure. You can use it just for manual check if all subscribers using your distribution servers are working. Or you can use it in your further T-SQL development where replication state plays major role. What is prepared for you is using this procedure in SQL Agent job informing you where there is some problem.

5.1 Parameters

Stored procedure has some input paramaters that are optional to use and has their default values.

Input:
- **@p_SuppressResults** *BIT* – You can use it for supressing resultset.Useful when you just want to check if alert state happening but no more information needed. By default set to **0.**
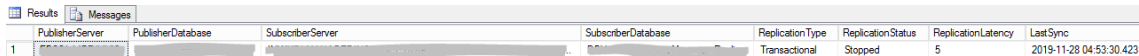
Output:
- **@p_HTMLTableResults** *NVARCHAR(MAX)* – This parameter contains resultset formatted as HTML table. Use it in HTML context to directly put results into website or HTML formatted email message.
- **@p_RiseAlert** *BIT* - This parameter is holding flag if some alert situation is happening with subscriptions. You can use it to pass this flag out of stored procedure and use it in your firther program workflow.

5.2 Simple run for checking actual state (no parameters)

You will just manually check actual state of your subscriptions and get some rows in resultset containing information about not properly working ones.

```
EXEC [distribution].[dbo].[usp_ReplicationMonitor]
```


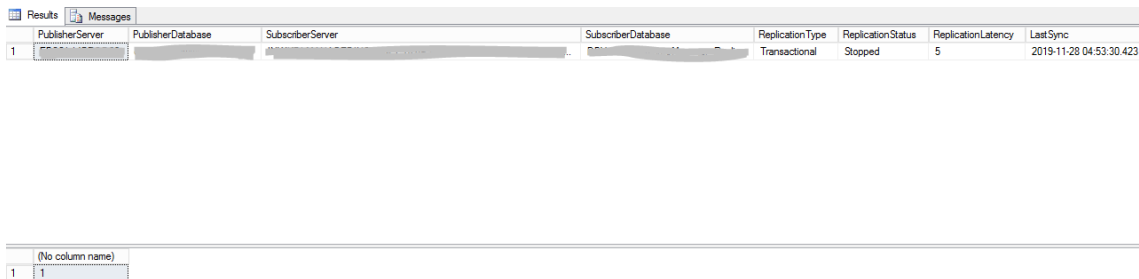|	PublisherServer	PublisherDatabase	SubscriberServer	SubscriberDatabase	ReplicationType	ReplicationStatus	ReplicationLatency	LastSync
1					Transactional	Stopped	5	2019-11-28 04:53:30.423

Figure 6 Result set with problematic subscriptions

5.3 Check if there is some problem (output parameter @p_RiseAlert)

Now passing some variable as container to hold value from output parameter **@p_RiseAlert**. After executing procedure you have this flag for making decisions. This logis is used in SQL Agent job prepared for you. When there is **@p_RiseAlert = 1** it is sending email notification.

```sql
DECLARE @RaiseAlert BIT
DECLARE @ResultTable NVARCHAR(MAX)

EXEC [distribution].[dbo].[usp_ReplicationMonitor]
        @p_SuppressResults = 1,
        @p_HTMLTableResults = @ResultTable OUTPUT,
        @p_RaiseAlert = @RaiseAlert OUTPUT
```


	PublisherServer	PublisherDatabase	SubscriberServer	SubscriberDatabase	ReplicationType	ReplicationStatus	ReplicationLatency	LastSync
1					Transactional	Stopped	5	2019-11-28 04:53:30.423

	(No column name)
1	1

Figure 3 Result set + value of output parameter

5.4 Supress result set outcome (parameter @p_SuppressResults set to 1)

If you don't need any data except alert flag you can suppress returing of resultset. Just nice to avoid doing something that is not usefull in further processing.

```sql
DECLARE @RaiseAlert BIT
DECLARE @ResultTable NVARCHAR(MAX)

EXEC [distribution].[dbo].[usp_ReplicationMonitor]
        @p_SuppressResults = 1,
        @p_HTMLTableResults = @ResultTable OUTPUT,
        @p_RaiseAlert = @RaiseAlert OUTPUT
```

6 Possible problems

There was testing of the solution for debugging and tuning purposes and all known problems has been fixed already, but as everything also this script can cause some issues in different environments.

I'm assuming only following possible issues:

- problems with not properly working Database Mail - if you are using SQL Server 2016 there is known bug fixed in CUs
- Failed to initialize sqlcmd library with error number -2147467259

And some other possible problems can be related to OH stuff in the solution so, please be so kind and try to check this FAQ https://ola.hallengren.com/frequently-asked-questions.html first before asking me directly.

6.1 Reporting issues

Please report all found issues, current version of the solution is the first one and require some debugging to be "perfect". Here are some contacts you can write to via email or LYNC:

- tomas.rybnicky@wetory.eu
- Use GitHub issues channel https://github.com/wetory/SQL-Server-Replication-Monitoring

7 Change history

Version	Date	Author	Approved by	Change history
V1.0	28.11.2019	Tomáš Rybnický	-	First version of this document