

Initialize Subscription from a Backup (MI COPY_ONLY backup)

Last updated by | Holger Linke | Dec 7, 2022 at 1:26 AM PST

Contents

- [Issue](#)
- [Investigation / Analysis](#)
- [Mitigation](#)
- [More Information](#)
 - [Overview of the basic steps](#)
 - [Putting the pieces together](#)
- [Public Doc Reference](#)
- [Internal Reference](#)

NOTE: This article only applies to replicating from a Managed Instance using a COPY_ONLY backup. It does NOT apply to replicating from a Managed Instance using a Point-in-Time-restored backup. It does NOT apply to replicating from an on-premise SQL Server.

For the Managed Instance PITR scenario, please see article [Initialize Subscription from a Backup \(MI to MI PITR\)](#).

For the on-premise to MI scenario, please see article [Initialize Subscription from a Backup \(avoid using Snapshot\)](#).

Issue

The customer has configured Transactional Replication on a Managed Instance and now wants to add a subscription based on a backup of the published MI database. They created a COPY_ONLY backup of the published database and restored it to the target instance. When the customer now executes the `sp_addsubscription` stored procedure specifying `@sync_type = 'initialized with backup'`, the stored procedure fails with error 18782:

Msg 18782, Level 16, State 1, Server [sqlmitest.xxxxxx.database.windows.net](#) [], Procedure sys.sp_MSextractlastlsnfrombackup, Line 208 Could not locate backup header information for database '[[sqlmitest.xxxxxx.database.windows.net](#) []].[repltest]' in the specified backup device. Specify a backup device that contains a backup of the Publisher database.

They want to use a COPY_ONLY backup because they can't perform a point-in-time restore to the intended Subscriber instance/server. The disadvantage of the COPY_ONLY backup is that TDE needs to be disabled before creating the backup, which might not be possible for other customers.

Investigation / Analysis

The `sys.sp_MSextractlastlsnfrombackup` procedure, that shows up in the error message, is called within the `sp_addsubscription` procedure.

This procedure performs the following call sequence:

- `sp_addsubscription ==> sys.sp_MSrepl_addsubscription ==> sys.sp_MSaddautosyncsubscription ==> sys.sp_MSextractlastlsnfrombackup ==> sys.fn_convertnumericlsntobinary10(LastLSN)`

Basically `sys.sp_MSextractlastlsnfrombackup` extracts the last LSN from the backup device ensuring that the backup set was taken from the Publisher.

Roughly, logic is performed as in the following code:

```
-- Create temp table to store RESTORE HEADERONLY information
-- Execute RESTORE HEADERONLY, and store information into temp table
-- Execute the following to extract LastLSN and convert to binary
select top 1 @lastlsn = sys.fn_convertnumericlsntobinary10(LastLsn)
from #restore_headeronly
where upper(ServerName collate database_default) = upper(@backupservername)
      and DatabaseName = @backupdatabase
order by LastLsn desc
if @@error<>0
begin
    select @retcode = 1
    goto Failure
end
if @lastlsn is null
begin
    declare @quoteddb nvarchar(4000)
    select @quoteddb = sys.fn_replquotename(@backupservername, default) collate database_default + N'.' +
                    sys.fn_replquotename(@backupdatabase, default) collate database_default
    raiserror(18782,16,-1,@quoteddb)
    select @retcode = 1
    goto Failure
end
```

To ensure that the backup was taken from the publisher, the `ServerName` is retrieved from the header information in the backup set and compared with the server name (instance name) which is the SQL instance that is executing `sp_addsubscription` procedure.

But if we look at the `ServerName` in the backup set (using the `RESTORE HEADERONLY` command), the server name has no relevance with the SQL MI instance name.

| | BackupName | BackupDescription | BackupType | ExpirationDate | Compressed | Position | DeviceType | UserName | ServerName | DatabaseName |
|---|---------------------|-------------------|------------|----------------|------------|----------|------------|-----------|--------------|--------------|
| 1 | Reptest-Full Backup | NULL | 1 | NULL | 0 | 1 | 9 | sqlbadmin | BC2C1209A862 | Reptest |

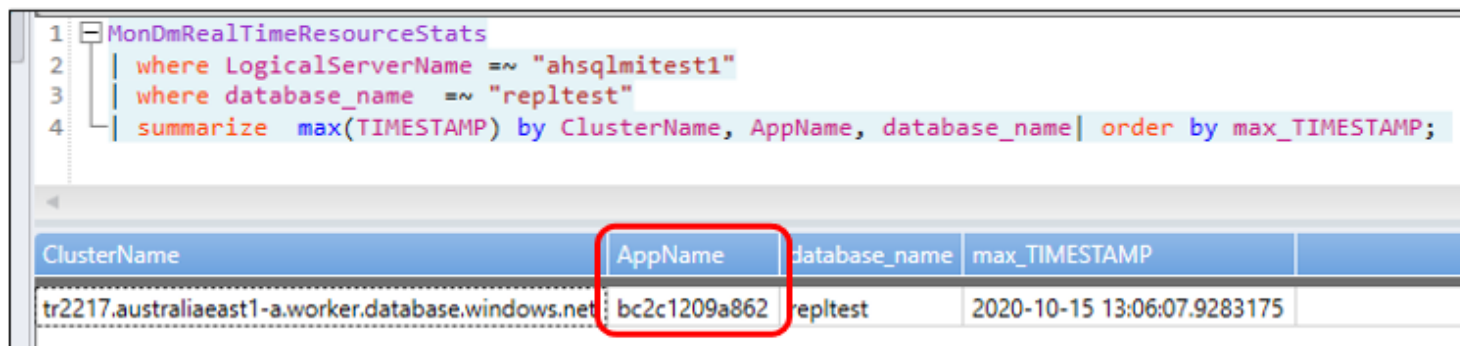
The SQL MI names I have in my test environment are the following:

- `ahsqlmitest1.2f28dfcd8d93.database.windows.net`
- `ahsqlmitest2.1da9c1a56b79.database.windows.net`

In addition, the `@backupservername` in the above snippet code is assigned from `@@servername` which is the SQL MI name (or the Publisher name). Thus, consequently the stored procedure will always fail since the `WHERE` clause condition will never meet.

You might be wondering where that name in the `ServerName` column is coming from (or you might had already

guessed). The name is actually coming from the AppName in the service fabric.



```
1 MonDmRealTimeResourceStats
2 where LogicalServerName =~ "ahsqlmitest1"
3 where database_name =~ "repltest"
4 summarize max(TIMESTAMP) by ClusterName, AppName, database_name | order by max_TIMESTAMP;
```

| ClusterName | AppName | database_name | max_TIMESTAMP |
|---|--------------|---------------|-----------------------------|
| tr2217.australiaeast1-a.worker.database.windows.net | bc2c1209a862 | repltest | 2020-10-15 13:06:07.9283175 |

Mitigation

As of this writing, PG is working on a fix. In the meantime, there is a workaround which is by specifying `@sync_type = 'initialize from lsn'` instead of `'initialize with backup'` in the `sp_addsubscription` call. You can get the last LSN from the `RESTORE HEADERONLY` command and set that for the `@subscriptionlsn` parameter. A sample is described in the following section.

More Information

The workaround is relatively straight forward, although there are some work involved in terms of scripting or automation.

Overview of the basic steps

The basic steps to go through are outline here:

- Run `RESTORE HEADERONLY` command and retrieve the LastLSN.
- Convert the LastLSN value to binary format (using a "conversion" function which the customer will have to create).
- Call `sp_addsubscription` setting the `@subscriptionlsn` parameter to the converted LastLSN.

The conversion function that has to be created in the publisher database:

```

CREATE FUNCTION fn_ConvertNumericLSNToBinary10 (
    @numericlsn NUMERIC(25,0)
) RETURNS BINARY(10)
AS
BEGIN
    /*
    Declare components to be one step larger than the intended type to avoid sign overflow problems.
    e.g. convert(smallint, convert(numeric(25,0),65535)) will fail but convert(binary(2),
    convert(int,convert(numeric(25,0),65535))) will give the intended result of 0xffff.
    */
    DECLARE @high4bytelsncomponent bigint,
    @mid4bytelsncomponent bigint,
    @low2bytelsncomponent int

    SELECT @high4bytelsncomponent = CONVERT(BIGINT, FLOOR(@numericlsn / 1000000000000000))
    SELECT @numericlsn = @numericlsn - CONVERT(NUMERIC(25,0), @high4bytelsncomponent) * 1000000000000000
    SELECT @mid4bytelsncomponent = CONVERT(BIGINT, FLOOR(@numericlsn / 100000))
    SELECT @numericlsn = @numericlsn - CONVERT(NUMERIC(25,0), @mid4bytelsncomponent) * 100000
    SELECT @low2bytelsncomponent = CONVERT(INT, @numericlsn)

    RETURN CONVERT(BINARY(4), @high4bytelsncomponent) +
        CONVERT(BINARY(4), @mid4bytelsncomponent) +
        CONVERT(BINARY(2), @low2bytelsncomponent)
END

```

Putting the pieces together

The following T-SQL code shows implementation of configuring a publication followed with the workaround described in this TSG. It assumes the distributor had been configured.

```

EXEC sp_addlogreader_agent
    @job_login = N'<Login Name>',
    @job_password = N'<Login Name Password>',
    @publisher_security_mode = 0,
    @publisher_login = N'<Publisher Login>',
    @publisher_password = N'<Publisher Login Password>;

-- Run at the Publisher on the publication database.
EXEC sp_addpublication
    @publication = N'<Publication Name>',
    @status = N'active',
    @allow_push = N'true',
    @allow_pull = N'true',
    @independent_agent = N'true',
    @immediate_sync = N'true',
    @allow_initialize_from_backup = N'true'

/*
    Run at the Publisher on the master database.
    You can check sys.credentials for existing credentials or to confirm the credential created.
*/
CREATE CREDENTIAL [BlobContainerUrl]
WITH IDENTITY = 'SHARED ACCESS SIGNATURE',
SECRET = 'sv=...';

/*
    Remove encryption key to be able to perform database backup of the publication database.
*/
ALTER DATABASE [REPLTEST] SET ENCRYPTION OFF

-- Run at the Publisher on the publication database.
DROP DATABASE ENCRYPTION KEY

-- Run at the Publisher on the master database.
BACKUP DATABASE PubDb TO URL = N'<Blob Container URL>/<DB backup file>' WITH COPY_ONLY

/*
    Assuming the Subscriber is hosted by another SQL instance, create a credential on the Subscriber SQL instance
    You can skip this step if the Subscriber will be hosted by the same SQL instance.
*/
CREATE CREDENTIAL [BlobContainerUrl]
WITH IDENTITY = 'SHARED ACCESS SIGNATURE',
SECRET = 'sv=2020-08-04...';

/*
    Perform a database restore at the subscriber.
*/
RESTORE DATABASE [SubDb] FROM URL = N'<Blob Container URL>/<DB backup file>'

/*
    Retrieve the LastLSN from the database backup file.
    The following will return output from the RESTORE HEADERONLY command.
    Copy the LastLSN which will be used for the fn_ConvertNumericLSNToBinary10 conversion function created earlier
*/
declare @SubLsn numeric(25,0)
declare @lastlsn binary(10)
declare @cmd nvarchar(500)
set @cmd = N'RESTORE HEADERONLY from url = N'<Blob Container URL>/<DB backup file>'
exec(@cmd)

/*
    Run the conversion function with the LastLSN as its parameter.
    The following is an example using a LastLSN copied from the RESTORE HEADERONLY command output.
*/
select 'ConvertedLastLSN' = dbo.fn_ConvertNumericLSNToBinary10(95000000245600001)

/*

```

Now run the `sp_addsubscription` stored procedure at the Publisher on the publication database. Set the `@subscriptionlsn` parameter with the converted LastLSN retrieved from the previous step. The following example uses the converted LastLSN returned from the previous step.

If you get error 21399, backup the publication database with the "WITH FORMAT" option and rerun the RESTORE H. In addition, consider to drop the subscriber database and perform a restore database from the new backup. Then all tables and their records are in complete sync between the Publisher and the Subscriber.

--<Error 21399>--

The transactions required for synchronizing the subscription with the specified log sequence number (LSN) are
--<>--

*/

```
EXEC sp_addsubscription
    @publication = N'<Publication Name>',
    @subscriber = N'<SubscriberServer>',
    @destination_db = N'<SubscriberDatabase>',
    @sync_type = N'initialize from lsn',
    @subscriptionlsn = 0x0000005F000009980001,
    @backupdevicetype = 'URL',
    @backupdevicename = N'<Blob Container URL>/<DB backup file>',
    @subscription_type = N'push',
    @update_mode = N'read only'
```

-- Assuming to create a PUSH subscription



```
EXEC sp_addpushsubscription_agent
    @publication = N'<Publication Name>',
    @subscriber = N'<Subscriber>',
    @subscriber_db = N'<Subscriber Database>',
    @job_login = N'<Job login>',
    @job_password = N'<Job Login Password>',
    @subscriber_login = N'<Subscriber Login>',
    @subscriber_password = N'<Subscriber Login Password>',
    @subscriber_security_mode = 0,
```



Public Doc Reference

[Initialize a Transactional Subscription from a Backup](#) 

Internal Reference

- [Initialize Subscription from a Backup \(avoid using Snapshot\)](#)
- [Initialize Subscription from a Backup \(MI to MI PITR\)](#)
- [Initialize Subscription from a Backup \(on-premise to MI\)](#)
- [ICM 208292273](#) 
- [ICM 208304098](#) 

How good have you found this content?



-