I've been working on a throughout Monitor to SQL Server Read Scale Availability Groups, and I'd like to share it with you. Though it was built in an RSAG environment, it will work just as fine in a good old, Cluster-based, Always On Availability Groups (without covering clusters events of course).

## ****Where do we begin? In the end****

I guess covering all the problems and conclusions is not the main goal of everyone, but **maybe some of you just want the solution, so** [**here it is**](https://github.com/MadeiraData/MadeiraToolbox/tree/9e7b75153e80602854cb513daec983366e543b75/Monitoring%20Solutions/Availability%20Groups%20Monitoring)**,** with a Read Me file explaining what to do, a Design Excel file explaining every object, an All Together script for simple creation of all the objects (without jobs, read the Read Me file), and every object in a separated script. I'd also like to start with credit to a **great** [**post**](https://www.sqlshack.com/measuring-availability-group-synchronization-lag/) I've read and based some of my solution on it. It is a great "starting point".

## ****Now let us dig dipper with some background -****

When talking about monitoring the synchronization process of two (or more) replicas of Availability Group, we think about:

1.  Syncing Status - Knowing what the syncing status is now, and having this information documented historically. Knowing about changes in these statuses and alerting when an AG is not synching. As it is the most critical thing to monitor, short changes (due to short network disconnections) happened frequently and raised alerts. For that I implemented a 3 samples-based monitor. When a status changes between one sample (1st) to another (2nd), the code will enter an inner loop. A third sample will be taken 10 seconds after the first one, and if it is still different than the first one, than it will alert. I also filtered out “synchronizing” statuses, a decision taken in my organization.

2.  Syncing Rate - To measure the syncing process flow - its rate, how much data has been copied from one replica to another in a measured constant time diff. I used here a query from the [post](https://www.sqlshack.com/measuring-availability-group-synchronization-lag/) mentioned before, and kept it as a "background" monitor, without any alert. It is a "nice to have" metric for network/syncing issues investigations.

3.  Lags - This is the most complex thing to measure, starting with what do we call a "lag", what is a lag to the organization we work at, what counts as "high lag". All these questions are also explained in the blog I’ve mentioned, but there were several more difficulties I ran into when creating this solution.

Doing the Opposite - Pays Off

Most common and popular way to measure a lag is based on the Redo Queue and Send Queue, and there are great queries ready to use throughout the web. But, when data movement is suspended/instance is down, these metrics return NULL. Microsoft also bothered to mention that any information regarding Secondary replicas from DMV’s/Performance Counters, read from the Primary replica, is inaccurate, from the base fact that it takes time to update the Primary, that the Secondary has already committed. Another way would be measuring Send Queue at Primary and Redo Queue at the Secondary, dealing with them separately. I avoided it since my goal was to keep it as simple and easy for the next person, and with an aim of having one table in one place with everything I need.  
  
After testing everything I found, I decided to base my monitor on **Last Commit End Time.** It does have edge cases needed to be taken care of, like – database not being used at all for a while, then a transaction happens, and it shows a huge lag because it has been a while since the last commit happened. Another issue was selecting data regarding the Secondary from the Primary when the commit has not been updated yet (even though it did) and it resulted with a fake lag.   
All these issues (and more), together with the fact you always have a short lag of milliseconds (depending on your network bandwidth) results with a “price” of up to 2 seconds standard deviation. Rest assure you are safe if it keeps like that.  
In my alerting procedure, I check what happens along 3 following samples, and alert when there is a potential risk, and not for every short lag that is gone a second later.

I’d like to sum up saying all the “tricks” and logics implemented in this solution were made to match the needs of the organization I worked with. I believe it will serve well others as well, but you might find things that will be better different for your purposes. Tried to keep the code simple and documented as possible, and more technical explanations can be found in the excel sheet.