

# When to use Read-Replica and Troubleshooting Replica Lag

Last updated by | Hamza Aqel | Feb 14, 2022 at 6:34 AM PST

---

## When to use a read replica:

- The read replica feature helps to improve the performance and scale of read-intensive workloads.
- Read workloads can be isolated to the replicas, while write workloads can be directed to the master.
- The read replica can be used as the data source for reporting.
- The read replica feature uses PostgreSQL asynchronous replication. The feature isn't meant for synchronous replication scenarios.


There will be a measurable delay between the master and the replica. The data on the replica eventually becomes consistent with the data on the master.

- Use this feature for workloads that can accommodate this delay.
- You can create a read replica in a different region from your master server.
- Cross-region replication can be helpful for scenarios like disaster recovery planning or bringing data closer to your users.
- Every replica is enabled for storage auto-grow.


## Azure Postgres replication:

- Read replicas and logical decoding both depend on the Postgres write ahead log (WAL) for information.

## Monitor replication:

- <https://docs.microsoft.com/en-us/azure/postgresql/concepts-read-replicas> 
- Replica should be ON as shown below for Master database

Home > Postgresquerytest36

**Postgresquerytest36** | Replication 


Azure Database for PostgreSQL server

Search (Ctrl+/) << + Add Replica | Delete Replica | Stop Replication | Save | Discard

Overview  
Activity log  
Access control (IAM)  
Tags  
Diagnose and solve problems

Settings

Connection security  
Connection strings  
Server parameters  
**Replication**  
Active Directory admin

Azure replication support [Learn more](#)  OFF **REPLICA** LOGICAL

**Master**

Name	↑↓	Pricing tier	↑↓	Location	↑↓	Status	↑↓
postgresquerytest36		General Purpose, 2 vCore(s), ...		East US		Available	

**Replicas**

Name	↑↓	Pricing tier	↑↓	Location	↑↓	Status	↑↓
No results							

### To see total lag in bytes :

- This is the Kusto to use to verify the lag from master to replica.

//to check total\_lag\_in\_bytes

### Kusto Query:

```
MonDmPgSqlReplicationStatsPrimary | where LogicalServerName == "rajanipostgresserver" |project
PreciseTimeStamp, todouble(total_lag_in_bytes) |render timechart
```

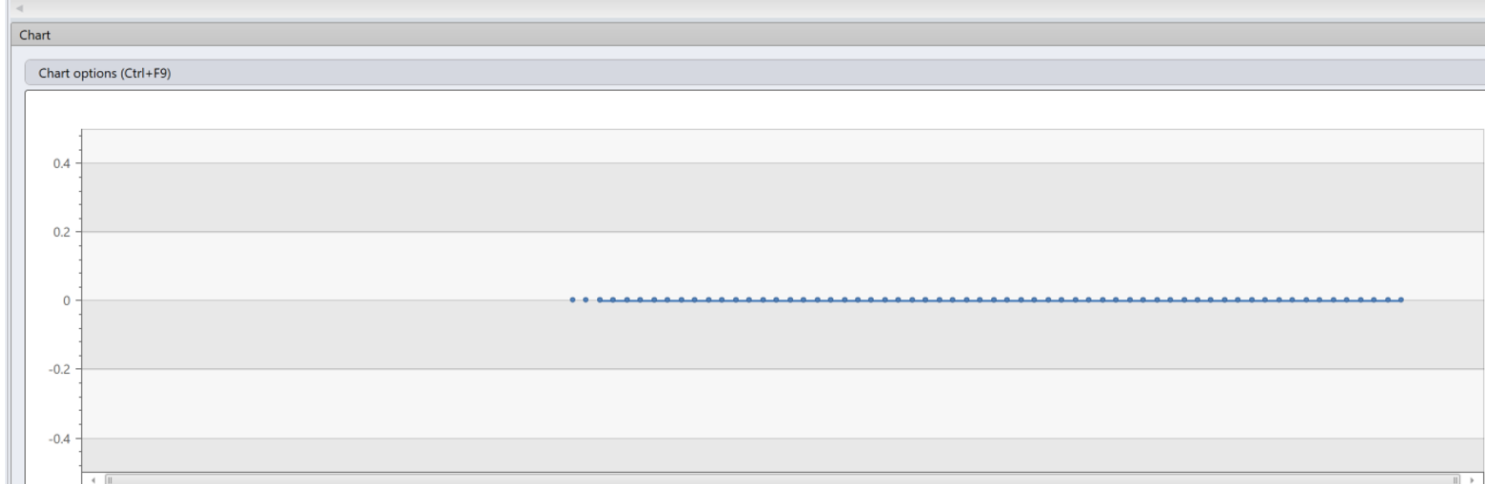
- In below screen shot, there is no lag from master to primary.
- The lag in seconds shown on replica can increase continuously if there is no activity happening on the master, since replica has nothing to apply
- Also in addition to streaming replication which is the delay being shown here, the replica also consumes the transaction log from the master that is produced every 15 minutes or 900 seconds even if there is no activity on the master. This is the reason we see the delay on replica not crossing the 900 seconds.
- But again to clarify there does not seem to be any real data lag on the replica. The lag\_in\_bytes on the master is the true measure of data lag and it is almost zero, when sampled every 5 minutes.

If customer has some continuous load on the master then they will see that the lag\_in\_seconds also will move closer to zero because replica will continuously apply changes from the master.

```

278 //to check total_lag_in_bytes
279 MonDmPsqlReplicationStatsPrimary
280 | where LogicalServerName == "rajanipostgresserver"
281 | project PreciseTimeStamp, todouble(total_lag_in_bytes)
282 | render timechart
283

```



### Understand pg\_wal\_lsn\_diff:

- pg\_wal\_lsn\_diff calculates the difference in bytes between two write-ahead log locations. It can be used with pg\_stat\_replication or some functions shown in Table 9.84 to get the replication lag.
- The lag indicates the amount of activity on the master that has to be applied on the replica.
- If you run an insert and a delete and repeat it a thousand times, there won't be any data size increase on the master or new data inserted in the replica, but the same activity has to be repeated on the replica by replaying the log.
- This lag indicates the lag in terms of amount of activity that has to be applied on the replica.

### Understanding transaction log:

If there are repeated pattern of transactions going on the master which means no obvious workload increased corresponded to any lag. The transaction log size may give you idea on the change of database increased(below is the kusto).

AlrBackup | where TIMESTAMP >= ago(32h) | where LogicalServerName =~ "rajanipostgresserver" or logical\_server\_name =~ "rajanipostgresserver" | where event\_type contains "Metadata\_details" | project TIMESTAMP, LogicalServerName, backup\_type, event, event\_type, backup\_service\_state, backup\_start\_date, backup\_end\_date, backup\_path, backup\_size, uncompressed\_backup\_size, code\_package\_version

### Verifying log file:

Check if the log file is archived with below command and output looks like below and text column says that the transaction log file archived.

MonRdmsPsqlSandbox |where TIMESTAMP >= ago(7d) |where LogicalServerName == "rajanipostgresserver" //|where text contains "restored transaction log" |where text contains "archived" or text contains "pg\_start\_backup" or text contains "pg\_stop\_backup" |order by originalEventTimestamp asc |project originalEventTimestamp,NodeName, process\_id, text=trim("((\r)?(\n)?)\*",text)

Result should look like below:

originalEventTimestamp NodeName process\_id text 2020-09-14 13:43:24.6296436 DB.3 71216 2020-09-14 13:43:24 UTC-5df08c6a.114-INFO: archived transaction log file "000000010000001B0000004E"

2020-09-14 13:53:51.2322380 DB.3 71216 2020-09-14 13:53:51UTC-5df08c6a.114-INFO: archived transaction log file "000000010000001B000000CB"

Compare the timestamps where there is lag and the transaction log file archived immediately after lag.

- Each transaction log is max 16 MB.
- Postgres closes a transaction log once it reaches 16MB or at the end of 15 minutes which ever is earlier. Even if there is no activity at all, at end of 15 minutes PG closes the a 16MB transaction log. At that point it will be archived.
- If you can compare both the timestamps where any lag is logged with transaction log timestamp and if the timestamps matches, we can confirm that the metric is being taken immediately after a transaction log is closed. From that I can say there was not much or no activity in that last few minutes before the delay is captured and as soon as an 16MB transaction is closed and archived, since we are taking the delay at that time, we might see about 15 minutes. something MB of delay because replica still has to apply that empty log file of 16MB even if there are no real transactions in the file to apply.
- For the customer, best way to calculate real data lag is to insert a timestamp in master server and query the max timestamp on replica and calculate the difference in current time and that max timestamp on the replica

MonDmPgSqlReplicationStatsPrimary

```

280 | where LogicalServerName contains "rajanipostgresserver"
281 | project PreciseTimestamp, LogicalServerName, ClusterName, NodeName, pid, slot_name, active, application_name, wal_sender_state, total_lag_in_bytes, sync_state, sent_lsn, write_lsn, flush_lsn, replay_lsn
282 | order by PreciseTimestamp, LogicalServerName desc
283
284
285 | AllBackup

```

PreciseTimestamp	LogicalServerName	ClusterName	NodeName	pid	slot_name	active	application_name	wal_sender_state	total_lag_in_bytes	sync_state	sent_lsn	write_lsn	flush_lsn
2020-09-21 04:06:14.9589164	rajanipostgresserver	tr236.eastus1-a.worker.database.windows.net	DB.3	281344	azure_repl_slot_e4bfa75b	True	rajanipostgresserver-replicacrosregion	streaming	0	async	7F/A906CDE0	7F/A906CDE0	7F/A906CDE0
2020-09-21 04:06:14.9589164	rajanipostgresserver	tr236.eastus1-a.worker.database.windows.net	DB.3	282444	azure_repl_slot_9041f676	True	rajanipostgresserver-replicasameregion	streaming	0	async	7F/A906CDE0	7F/A906CDE0	7F/A906CDE0
2020-09-21 04:01:08.2551486	rajanipostgresserver	tr236.eastus1-a.worker.database.windows.net	DB.3	281344	azure_repl_slot_e4bfa75b	True	rajanipostgresserver-replicacrosregion	streaming	0	async	7F/A9006570	7F/A9006570	7F/A9006570
2020-09-21 04:01:08.2551486	rajanipostgresserver	tr236.eastus1-a.worker.database.windows.net	DB.3	282444	azure_repl_slot_9041f676	True	rajanipostgresserver-replicasameregion	streaming	0	async	7F/A9006570	7F/A9006570	7F/A9006570
2020-09-21 03:56:01.5355087	rajanipostgresserver	tr236.eastus1-a.worker.database.windows.net	DB.3	281344	azure_repl_slot_e4bfa75b	True	rajanipostgresserver-replicacrosregion	streaming	0	async	7F/A9000108	7F/A9000108	7F/A9000108
2020-09-21 03:56:01.5355087	rajanipostgresserver	tr236.eastus1-a.worker.database.windows.net	DB.3	282444	azure_repl_slot_9041f676	True	rajanipostgresserver-replicasameregion	streaming	0	async	7F/A9000108	7F/A9000108	7F/A9000108
2020-09-21 03:50:54.4723629	rajanipostgresserver	tr236.eastus1-a.worker.database.windows.net	DB.3	281344	azure_repl_slot_e4bfa75b	True	rajanipostgresserver-replicacrosregion	streaming	0	async	7F/A8054950	7F/A8054950	7F/A8054950
2020-09-21 03:50:54.4723629	rajanipostgresserver	tr236.eastus1-a.worker.database.windows.net	DB.3	282444	azure_repl_slot_9041f676	True	rajanipostgresserver-replicasameregion	streaming	0	async	7F/A8054950	7F/A8054950	7F/A8054950

- The replay\_lsn on the replica is the same as the sent\_lsn on the master which indicates that there is no lag.
- Observe if the read LSN and write LSN are same.

MonDmPgSqlReplicationStatsPrimary

```

272 | where LogicalServerName contains "rajanipostgresserver"
273 | project PreciseTimestamp, LogicalServerName, ClusterName, NodeName, pid, slot_name, active, application_name, wal_sender_state, total_lag_in_bytes, sync_state, sent_lsn, write_lsn, flush_lsn, replay_lsn
274 | order by PreciseTimestamp, LogicalServerName desc
275
276

```

pid	LogicalServerName	ClusterName	NodeName	pid	slot_name	active	application_name	wal_sender_state	total_lag_in_bytes	sync_state	sent_lsn	write_lsn	flush_lsn	replay_lsn
20:12.9528773	rajanipostgresserver	tr236.eastus1-a.worker.database.windows.net	DB.3	281344	azure_repl_slot_e4bfa75b	True	rajanipostgresserver-replicacrosregion	streaming	0	async	7F/A50A7078	7F/A50A7078	7F/A50A7078	7F/A50A7078
20:12.9528773	rajanipostgresserver	tr236.eastus1-a.worker.database.windows.net	DB.3	282444	azure_repl_slot_9041f676	True	rajanipostgresserver-replicasameregion	streaming	0	async	7F/A50A7078	7F/A50A7078	7F/A50A7078	7F/A50A7078
15:06.2647578	rajanipostgresserver	tr236.eastus1-a.worker.database.windows.net	DB.3	282444	azure_repl_slot_9041f676	True	rajanipostgresserver-replicacrosregion	streaming	0	async	7F/A5000108	7F/A5000108	7F/A5000108	7F/A5000108
15:06.2647578	rajanipostgresserver	tr236.eastus1-a.worker.database.windows.net	DB.3	281344	azure_repl_slot_e4bfa75b	True	rajanipostgresserver-replicacrosregion	streaming	0	async	7F/A5000108	7F/A5000108	7F/A5000108	7F/A5000108
09:59.5921757	rajanipostgresserver	tr236.eastus1-a.worker.database.windows.net	DB.3	282444	azure_repl_slot_9041f676	True	rajanipostgresserver-replicasameregion	streaming	0	async	7F/A40526D0	7F/A40526D0	7F/A40526D0	7F/A40526D0
09:59.5921757	rajanipostgresserver	tr236.eastus1-a.worker.database.windows.net	DB.3	281344	azure_repl_slot_e4bfa75b	True	rajanipostgresserver-replicacrosregion	streaming	0	async	7F/A40526D0	7F/A40526D0	7F/A40526D0	7F/A40526D0
04:52.8104311	rajanipostgresserver	tr236.eastus1-a.worker.database.windows.net	DB.3	281344	azure_repl_slot_e4bfa75b	True	rajanipostgresserver-replicacrosregion	streaming	0	async	7F/A404D0A8	7F/A404D0A8	7F/A404D0A8	7F/A404D0A8

Dumped Around 34 GB of Data on the Master using Generate\_Series.

- Master has 34GB and replica server has 17G when it was replicated and recovering slowly.
- It took an hour to complete total data replication.
- SELECT pg\_size\_pretty(pg\_database\_size('postgres'));-to verify database size

- `SELECT pg_size_pretty( pg_total_relation_size('testmaster'));` to verify table size
- `select pg_wal_lsn_diff(pg_current_wal_lsn(), replay_lsn)`

AS `total_log_delay_in_bytes` from `pg_stat_replication`;- query the master server directly to get the replication lag in bytes on all replicas

- If a master server or read replica restarts, the time it takes to restart and catch up is reflected in the Replica Lag metric.

## Note:

There are two modes of replication, streaming replication and file shipping replication. In some cases, streaming replication was broken, but data is flowing using file shipping replication. The replica continued to do replication in file shipping replica mode from the transaction log archive location. When this happens, primary shows a greater size than replica is because the primary stills keeps the transaction log files for the replica from where the replica has cut off at streaming replication. The primary is unaware that the replica already applied the new changes using file shipping replication. In file-shipping replication mode, primary has no knowledge of the existence of the replica. Only in streaming mode since replica connects to primary it knows where the replica is w.r.t applying the changes. Once the streaming replication is established after the issue is mitigated by the team, the primary releases all the transaction log files which are no longer needed since the replica already has applied them in file shipping replication mode.

## Check MonDmPgsQLReplicationStatsPrimary to see if streaming replication is running or not

```
MonDmPgsQLReplicationStatsPrimary
| where TIMESTAMP > ago (1h)
| where LogicalServerName == '{primary_server_name}'
| where slot_name == '{slot_name}'
| project PreciseTimeStamp, LogicalServerName, AppName, slot_name, active, wal_sender_state, application_name,
| order by PreciseTimeStamp desc
```

latest 2 records have 'active' column set to 'True' and 'wal\_sender\_state' column is set to 'streaming'

## Check if replica is still replicating using FILE SHIPPING REPLICATION.

When streaming breaks and in other conditions where there is a lot to catchup, replica switches to use file shipping replication method.

Check the first query below to see if it is actively doing it. You should see some logs applied in the CURRENT TIME - 20/25 MINs.

```
MonRdmsPgsQLSandbox
| where LogicalServerName == "replica_server_name"
| where text contains "restored transaction log" //Sometimes just search for "restored" for PFS servers
```



You can use this query to see where the primary is

```
MonRdmsPgSqlSandbox
| where LogicalServerName == "primary_server_name"
| where text contains "archived transaction log" //Sometimes just search for "archived" for PFS servers
```

The difference in the log file names of what primary is archiving and replica is restoring will tell how far behind the replica is. As long as replica is close to primary here it is still getting the latest data, by file shipping and not thru streaming replication.

It is still important to fix the streaming replication, because primary is holding all these files still for the replica. Only in streaming replication mode, it knows where the replica is, in file shipping mode it does not know that replica already applied all these files.

In this case monitor the storage\_percent metric of the primary server to see if it is not reaching 90%, engage PR DRI if you see a risk of storage full.

Reference TSG: [https://microsoft.sharepoint.com/teams/orcasql/\\_layouts/OneNote.aspx?id=%2Fteams%2Focsql%2FSiteAssets%2FProject Orcas Notebook&wd=target\(MonitorAlertMitigate.one|204DE02D-FF35-40E7-94AE-F25835AE59B4%2FPgSQL-Replica-001%3A Server Reports INACTIVE PHYSICAL Replication slots|2A12D672-4B34-463C-85A3-6FAC52B418AF%2F\)](https://microsoft.sharepoint.com/teams/orcasql/_layouts/OneNote.aspx?id=%2Fteams%2Focsql%2FSiteAssets%2FProject%20Orcas%20Notebook&wd=target(MonitorAlertMitigate.one|204DE02D-FF35-40E7-94AE-F25835AE59B4%2FPgSQL-Replica-001%3A%20Server%20Reports%20INACTIVE%20PHYSICAL%20Replication%20slots|2A12D672-4B34-463C-85A3-6FAC52B418AF%2F)|onenote:https://microsoft.sharepoint.com/teams/orcasql/_layouts/OneNote.aspx?id=%2Fteams%2Focsql%2FSiteAssets%2FProject%20Orcas%20Notebook/MonitorAlertMitigate.one#PgSQL-Replica-001%3A%20Server%20Reports%20INACTIVE%20PHYSICAL%20Replication%20slots&section-id={204DE02D-FF35-40E7-94AE-F25835AE59B4}&page-id={2A12D672-4B34-463C-85A3-6FAC52B418AF}&end)  onenote:[https://microsoft.sharepoint.com/teams/orcasql/\\_layouts/OneNote.aspx?id=%2Fteams%2Focsql%2FSiteAssets/Project Orcas Notebook/MonitorAlertMitigate.one#PgSQL-Replica-001 Server Reports INACTIVE PHYSICAL Replication slots&section-id={204DE02D-FF35-40E7-94AE-F25835AE59B4}&page-id={2A12D672-4B34-463C-85A3-6FAC52B418AF}&end](https://microsoft.sharepoint.com/teams/orcasql/_layouts/OneNote.aspx?id=%2Fteams%2Focsql%2FSiteAssets/Project Orcas Notebook/MonitorAlertMitigate.one#PgSQL-Replica-001 Server Reports INACTIVE PHYSICAL Replication slots&section-id={204DE02D-FF35-40E7-94AE-F25835AE59B4}&page-id={2A12D672-4B34-463C-85A3-6FAC52B418AF}&end) 

### Note :

1- In heavy workload , it is expected to have replication lag , nothing we can do on this case and you can share the below RCA with the customer if needed:

For most workloads read replicas offer near-real-time updates from the primary. However, with persistent heavy write-intensive primary workloads, the replication lag could continue to grow and may never be able to catch-up with the primary. This may also increase storage usage at the primary as the WAL files are not deleted until they are received at the replica. If this situation persists, deleting and recreating the read replica after the write-intensive workloads completes is the option to bring the replica back to a good state with respect to lag. Asynchronous read replicas are not suitable for such heavy write workloads. When evaluating read replicas for your application, monitor the lag on the replica for a full app work load cycle thru its peak and non-peak times to access the possible lag and the expected RTO/RPO at various points of the workload cycle.

Read replicas - Azure Database for PostgreSQL - Single Server | Microsoft Docs

2- Changing Replica admin password is not supported.

**How good have you found this content?**

