Disaster recovery for Microsoft SQL Server

This document introduces disaster recovery (DR) strategies for Microsoft SQL Server for architects and technical leads responsible for designing and implementing disaster recovery on Google Cloud.

Databases can become unavailable for various reasons, for example, hardware or network failures. To provide continuous database access during failures, a secondary database is maintained that is a replica of a primary database. Having the secondary database in a different location increases the chances that it's available when the primary database becomes unavailable.

If the primary database becomes unavailable, your mission-critical app connects to a secondary database, continuing from the most recently known consistent data state to provide services to your users with minimal or no downtime.

The process of making a secondary database available upon failure of the primary database is called *database disaster recovery (DR)*. The secondary database recovers from the unavailability of the primary database. The secondary database ideally has the exact same consistent state as the primary database when it is unavailable or misses only a minimal set of recent transactions from the primary database.

Database DR is an essential feature for enterprise customers. The main driver is business continuity for mission-critical apps. For example, a mission-critical app generates revenue (ecommerce), provides continuous dependable services (flight or powerplant management), or supports life-preserving functionality (patient monitoring). In all of these examples, it's of the utmost importance that the app is continuously available because it's deemed mission critical.

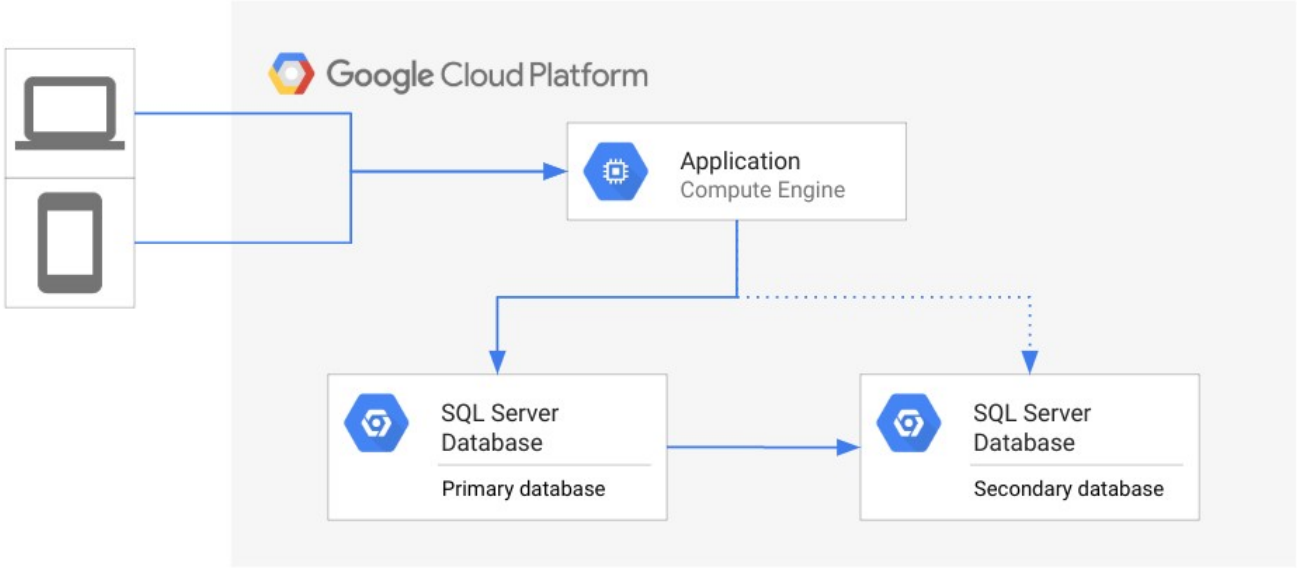
Most database management systems provide disaster recovery functionality, including Microsoft SQL Server (https://www.microsoft.com/sql-server/default.aspx). This architecture document discusses how DR features provided by SQL Server are implemented in the context of Google Cloud.

Terminology

The following sections explain the terms that are used throughout this document.

General DR architecture

The following diagram illustrates the general DR architecture topology.



In the preceding diagram, an app accesses a primary database while a secondary database is standing by and mirrors the state of the primary database. Clients are accessing the app that runs on Google Cloud.

If the primary database becomes unavailable, database admins or the operations team have to decide to start the disaster recovery process. If database disaster recovery is initiated, the app is reconnected to the secondary database. After it's connected, the app can serve its clients again. In an ideal situation, the app is available on the secondary database as soon as possible so clients might not even experience an outage. One alternative is to wait for the primary database to become accessible again, instead of initiating disaster recovery. For example, if the disaster is intermittent, it might be faster to resolve the problem instead of failing over.

Primary and secondary databases

A primary database is accessed by one or more apps to provide persistence services for the app's state management. A secondary database is related to a primary database and it contains a replica of the primary database. Ideally, the content of the secondary database exactly matches the content of the primary database at any point in time. In many cases, the secondary database lags the primary database due to delays in applying transactional changes made on the primary database. It's possible to associate more than one secondary database with a primary database, depending on the database technology. SQL Server

supports associating more than one secondary database with a primary database (https://docs.microsoft.com/sql/database-engine/availability-groups/windows/prereqs-restrictions-recommendations-always-on-availability?view=sql-server-2017)

.

Disaster recovery

If a primary database becomes unavailable, DR changes the role of the secondary database to become the primary database. If there is more than one secondary database, one of those databases is selected manually or based on a preferred failover list. Apps have to reconnect to the new primary database to continue accessing their state. If the new primary database wasn't in sync with the last-known state of the former primary database, the app starts from a past state (also known as *flashback*).

It's important to have at least one secondary database at all times for every primary database. After a disaster recovery, make sure that a new secondary database is set up to handle future disaster recovery scenarios.

Failover, switchover, and fallback

There are multiple scenarios for changing the role between primary and secondary databases:

- **Failover**: the process of changing the role of a secondary database to be the new primary database and connecting all apps to it. Failover is unintentional because it's triggered by a primary database becoming unavailable. You can configure failover to be triggered automatically or manually.
- **Switchover**: in contrast to failover, a switchover from a primary database to a secondary database (new primary database), is intentionally triggered for initial testing and scheduled maintenance. Test your DR system with a regular periodic switchover to ensure continued dependability of disaster recovery.
- **Fallback**: fallback is reversing the process where the new primary database becomes the secondary database, after the primary database is repaired. A fallback is intentionally triggered to reestablish the state before the failover or switchover was initiated. It isn't strictly necessary, but can be done based on disaster recovery requirements, such as locality or available resources.

Google Cloud zones and regions

Resources like databases are located in Google Cloud zones and regions (/docs/geography-and-regions), where each zone belongs to a region. A zone is a single point of failure domain. We recommend deploying a highly available and fault-tolerant resource in multiple zones within a region.

To guard against the outage of a whole region, establish multi-region strategies for disaster recovery. For example, the primary database is located in one region and its corresponding secondary database is located in another region.

Active modes: active-passive and active-active

A primary database is a database open for read and write operations (DML operations) so that apps accessing it can manage their state. The primary database is called an *active* database. The corresponding secondary database is passive because it replicates the primary database, but it isn't available by any app for state changing operations. After a failover or switchover, the secondary database becomes the new primary database and becomes an active database.

The primary database as well as the secondary database can both be active databases if the database technology supports this feature, called *active-active mode*. In this case, apps can connect to one or the other because both databases are available for state management. Disaster recovery in active-active mode doesn't require a failover if only one of the active databases becomes unavailable. If one active database is unavailable, the other active database continues to be available. Active-active mode is outside the scope of this article because SQL Server doesn't support this mode.

Standby modes: hot, warm, cold, and no standby

For the primary database to be the active database, it has to be running and able to run DML statements. The secondary database doesn't need to be running; it can be shut down. If it isn't running, the time it takes to recover from a disaster increases because the new primary database has to be brought to a running state first, before assuming the role of the new primary database.

There are several variations on how to set up the secondary database:

- **Hot standby**: the secondary database is up and running and ready to be connected to by clients. The latest available change from the primary database is always applied as soon as it becomes available.
- **Warm standby**: a secondary database is up and running, however, not all changes from the primary database have necessarily been applied yet.
- **Cold standby**: a secondary database isn't running. First, it needs to be started up, and then synched to the latest available state.
- **No standby**: the database software has to be installed first and subsequently started up before all changes from the primary database are applied. This mode is the least expensive mode because it doesn't consume resources when not needed, but compared to the other modes it takes the longest to become a new primary database.

DR strategies

In the following sections, DR strategies that Microsoft SQL Server supports are explained.

Recovery strategy dimensions

There are several key dimensions to consider when selecting or implementing a database disaster recovery strategy. Each dimension has a spectrum and the behavior and the expectations of the disaster recovery strategy depend on the selection of the points on the spectrum. The key dimensions are as follows:

- **Recovery Point Objective (RPO)** (https://wikipedia.org/wiki/Disaster_recovery#Recovery_Point_Objective): The maximum acceptable length of time during which data might be lost from your app due to a major incident. This dimension varies based on the ways that the data is used. RPO can be expressed in duration (seconds, minutes or hours) from the time of primary database unavailability or it can be expressed as identifiable processing states (last full backup or last incremental backup). No matter how RPO is specified, the disaster recovery strategy must implement the particular measure so that the RPO requirement can be satisfied. The most demanding case is the last committed transaction, which means no loss from the primary database to the secondary database must happen at all.

- **Recovery Time Objective (RTO)** (https://wikipedia.org/wiki/Disaster_recovery#Recovery_time_objective). The maximum acceptable length of time that your app can be offline. This value is usually specified as part of a larger **Service-level agreement** (https://wikipedia.org/wiki/Service-level_agreement). RTO is usually expressed in terms of duration from the time of primary database unavailability, for example, the app must be fully operational within 5 minutes. The most demanding case is immediate so that app users don't notice that disaster recovery took place.
- **Single point of failure domain**. It is up to you to decide if a region is considered a single point of failure domain for your disaster recovery requirements. If a region is a single point of failure domain for you, then disaster recovery has to be set up so that two or more regions are involved in the actual setup. If the region containing the primary database fails, the secondary database in a different region is the new primary database. If the single point of failure domain is assumed to be a zone, then disaster recovery can be set up across zones within a single region. If a zone fails, disaster recovery uses a second zone and makes the new primary database available in it.

Deciding on these key dimensions is making a decision between cost and quality. The lower the RTO and RPO, the more costly the disaster recovery solution can become as more active resources are used. In the following sections, several alternative DR strategies are discussed that represent points on the dimensions in context of the Microsoft SQL Server database.

DR strategies for SQL Server

Business continuity and database recovery - SQL Server (https://docs.microsoft.com/sql/database-engine/sql-server-business-continuity-dr?view=sql-server-2017) describes availability features that you can use to implement disaster recovery strategies.

Preliminaries

SQL Server runs on both Windows and Linux. However, not all availability features are available on Linux. SQL Server (https://www.microsoft.com/sql-server/sql-server-2017-editions) has several editions, but not all availability features are available in every edition.

SQL Server distinguishes instances from databases. An instance is the executing SQL Server software whereas a database is the set of data that is managed by a SQL Server instance.

Always On availability groups

Always On availability groups provide database-level protection. An availability group has two or more replicas. One replica is the primary replica with read and write access, and the remaining replicas are secondary replicas that can provide read access. Each database replica is managed by a standalone SQL Server instance. An availability group can contain one or more databases. The number of databases that can be included in an availability group and the number of supported secondary replicas depends on the SQL Server edition. All databases in an availability group undergo the same lifecycle changes at the same time. Availability groups implement the active-passive mode because only the primary database supports write access.

When a failover happens, a secondary replica becomes the new primary replica. Because an availability group includes standalone SQL Server instances, all operations captured in transaction logs are available in the replicas. Any change not captured in a transaction log needs to be manually synchronized, for example, SQL Server instance-level logins or SQL Server Agent jobs. In order to provide database-level protection and SQL Server instance protection, you need to set up Failover Cluster Instances (FCIs). This deployment architecture is discussed later in the Always On Failover Cluster Instance section (#always-on-failover-cluster-instance).

You can shield apps from role changes by using a listener. A listener supports apps connecting to the availability group. Apps aren't aware of which SQL Server instances are managing the primary database or the secondary replicas at any point in time. Listeners require that clients use a minimum .NET version of 3.5 with an update or 4.0 and higher as outlined in Business continuity and database recovery - SQL Server (https://docs.microsoft.com/en-us/sql/database-engine/sql-server-business-continuity-dr?view=sql-server-2017)
.

Availability groups rely on underlying layers of abstraction to provide their functionality. Availability groups run in a Windows Server Failover Cluster (WSFC) as outlined in Windows Server Failover Clustering with SQL Server (https://docs.microsoft.com/sql/sql-server/failover-clusters/windows/windows-server-failover-clustering-wsfc-with-sql-server?view=sql-server-2017)
. All nodes that run SQL Server instances have to be part of the same WSFC.

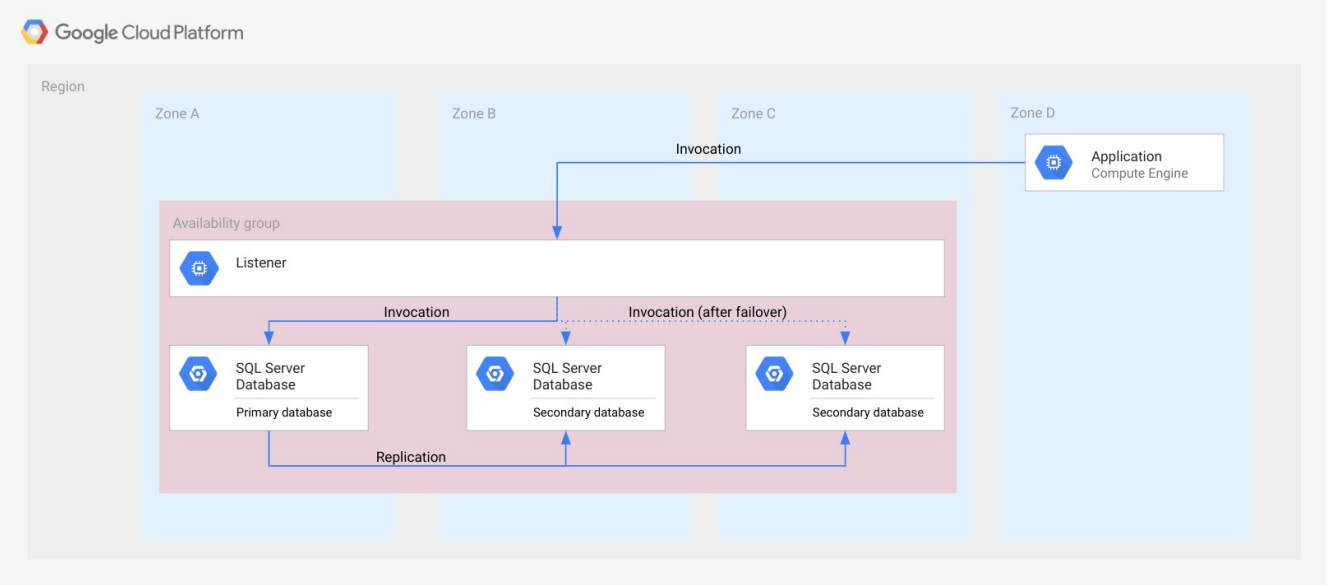
Transactions are sent from the primary database to all secondary replicas. There are two sending modes to send transactions: synchronous and asynchronous. You can independently configure each replica to use one or the other mode. In the synchronous sending mode, the transaction on the primary database only succeeds if it succeeds on all secondary replicas

that are synchronously linked. In the asynchronous mode, the transaction on the primary database can succeed even though not all of the secondary replicas have the transaction applied.

Your choice of sending mode influences the possible RTO, RPO, and your standby mode. For example, if transactions are sent to all replicas in synchronous mode, all replicas are in the exact same state. The most demanding RPO (most recent transaction) is fulfilled as all replicas are fully synchronized. The secondary replicas are hot standbys so any of those can immediately be used as a primary database.

Failover can be automatic or manual. An automatic failover is possible if all replicas are fully synchronized. In the preceding example, this is possible as all replicas are always fully synchronized.

The following figure shows an Always On availability group in a single region.



The availability group is represented as a rectangle spanning zones. This is for illustration purposes only to indicate that all databases belong to the same availability group. The availability group is not a cloud resource and as such, not implemented in a node or any other type of resource.

Always On Failover Cluster Instance

To guard against node failures, you can use Failover Cluster Instances (FCIs) instead of standalone SQL Server instances. There are two or more nodes that run SQL Server instances to manage a database (primary or secondary). The nodes managing a database form a

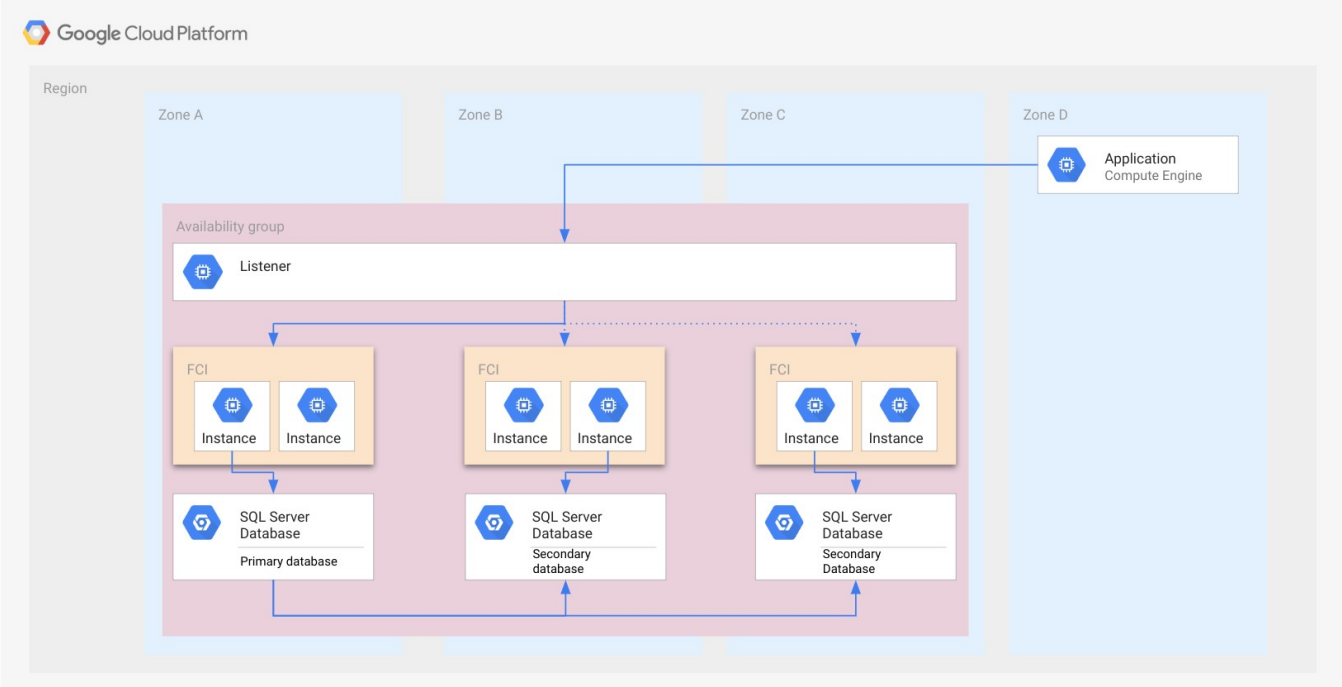
Failover Cluster. One node in the cluster is actively running a SQL Server instance while the other nodes aren't running SQL Server instances. When the node running the SQL Server instance fails, another node in the cluster starts up a SQL Server instance, taking over the management of the database (node failover). This process of automatically starting a SQL Server instance provides high availability functionality.

The FCI cluster appears as a single unit and clients accessing the cluster don't see the failover between nodes, except perhaps for a short period of unavailability. There is no data loss when a node failover occurs. Everything running inside the failed SQL Server instance is moved to another SQL Server instance in the same cluster. For example, SQL Server Agent jobs or linked servers are moved to another instance.

FCI cluster nodes can be set up in different Google Cloud zones. This architecture not only provides high availability on node failure, but also for zone failures. An example deployment of this strategy is discussed in the DR deployment alternatives section (#dr-deployment-alternatives).

Even though different nodes manage the same database and share the database, there is no common storage required among the nodes of an FCI cluster. SQL Server uses the Storage Spaces Direct (S2D) functionality to manage databases on dedicated node disks. For more information, see Configuring SQL Server Failover Cluster Instances (/compute/docs/instances/sql-server/configure-failover-cluster-instance).

The example of the previous section Always On availability groups (#always-on-availability-groups) with FCIs instead of standalone SQL Server instances is shown in the following figure. Each FCI has one active SQL Server instance managing the database.



As in the case of the availability group, an FCI is represented as a rectangle. This is for illustration purposes only to indicate that the nodes all belong to the same FCI. An FCI is not a cloud resource and as such, not implemented in a node or any other type of resource.

For a more detailed discussion, see Always On Failover Cluster Instances (SQL Server) (https://docs.microsoft.com/sql/sql-server/failover-clusters/windows/always-on-failover-cluster-instances-sql-server?view=sql-server-2017)

.

Distributed availability groups

Distributed availability groups are a special type of availability group. A distributed availability group spans two availability groups, one is in the role of the primary availability group and one is in the role of the secondary availability group. Distributed availability groups can forward transactions in synchronous as well as asynchronous mode from the primary availability group to the secondary availability group.

Even though each of the availability groups have their own primary database, this isn't an active-active deployment. Only the primary database of the primary availability group can receive write operations. The primary database of the secondary availability group is called forwarder. The forwarder receives the transactions from the primary availability group and forwards those to the secondary databases of the secondary availability group. A failover from

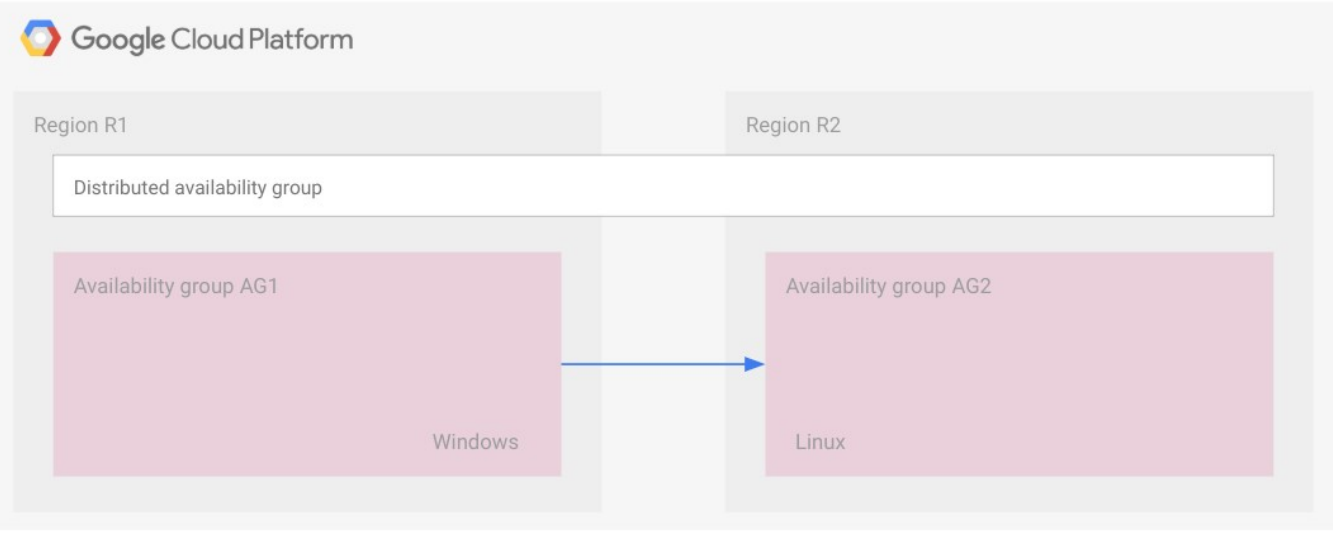
the primary availability group to the secondary availability group would make the primary database of the new primary availability group accessible for write operations.

The primary and secondary availability groups don't have to be in the same location and they don't have to be on the same operating system. However, each availability group has to have a listener installed. The distributed availability group itself doesn't have a listener. Distributed availability groups don't require that the two availability groups are in the same WSFC. All functionality required to make distributed availability groups work is contained within the SQL Server functionality and doesn't require the additional installation of underlying components.

A distributed availability group spans exactly two availability groups. An availability group can be part of two distributed availability groups. This possibility supports different topologies. One is a daisy-chaining topology from availability group to availability group across several locations. Another topology is a tree-like topology where the primary availability group is part of two different and separate distributed availability groups.

Distributed availability groups are the primary means to implement disaster recovery across operating systems. For example, the primary availability group can be set up on Windows, and a corresponding second availability group on Linux, with both availability groups forming a distributed availability group.

The following diagram shows two availability groups that are part of a distributed availability group.



Availability group 1 is the primary availability group and availability group 2 is the secondary availability group.

As in the case of the FCIs, a distributed availability group is represented as a rectangle. This is for illustration purposes only to indicate that the availability groups all belong to the same distributed availability group. A distributed availability group, like an availability group, is not a cloud resource and as such, not implemented in a node or any other type of resource.

For more information, see Distributed availability groups (https://docs.microsoft.com/sql/database-engine/availability-groups/windows/distributed-availability-groups?view=sql-server-2017)

.

Log shipping

Transaction log shipping is a SQL Server availability feature when RTO and RPO aren't as strict (low RTO and/or recent RPO) because the discrepancy in state between a primary database and its secondary database is significantly larger. The discrepancy is larger in terms of state because a transaction log file contains many state changes. The discrepancy is also larger in terms of lag time because transaction log files are transported asynchronously and have to be applied in their entirety to a secondary database.

Transaction log files are created by the primary database and backed up, for example, to Cloud Storage. Every transaction log file is copied to every secondary database and applied to it. Because the secondary database lags behind the primary database, they are in warm standby mode. Objects and changes that aren't captured by transaction logs have to be applied manually to the secondary databases to establish complete synchronization without loss.

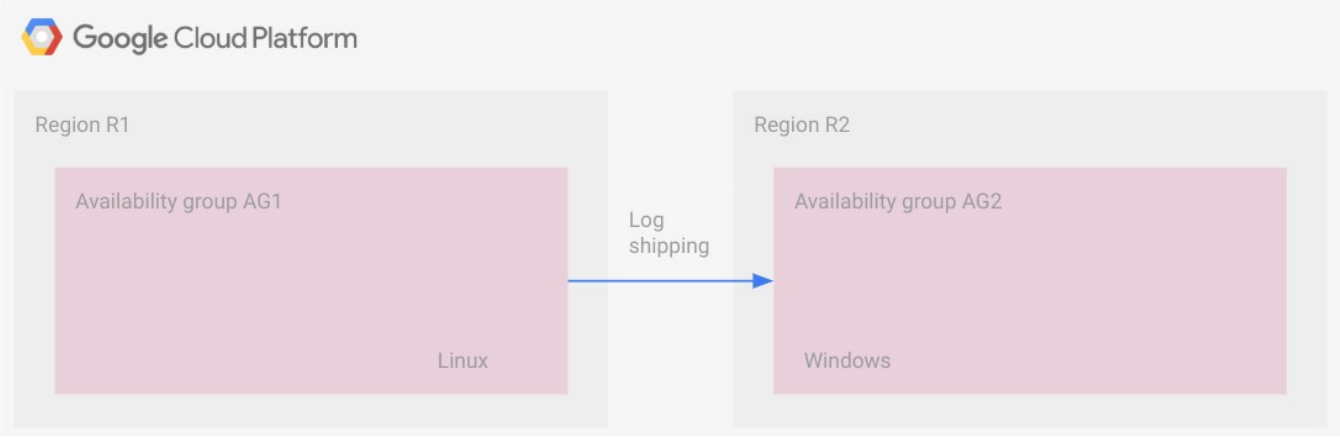
The SQL Server Agent automates the overall process of creating, copying, and applying transaction logs. Log shipping has to be set up for each database individually. If an availability group manages more than one database, then as many log shipping processes have to be set up.

In case of failure, the disaster recovery process has to be initiated manually because there is no automated support. In addition, client access isn't abstracted from the primary database and secondary databases by a listener. In case of failover, clients have to be able to deal with the role change of a database from the secondary role to the new primary role on their own by connecting to the new primary after a disaster recovery. It's possible to build separate abstractions independently of SQL Server instances, for example, floating IP addresses as described in Best practices for floating IP addresses (/solutions/best-practices-floating-ip-addresses).

Because log shipping is in part a manual process, you can delay applying copied log files to the secondary databases intentionally (in contrast to availability groups and distributed availability groups where changes are applied immediately). A possible use case is preventing data modification errors on the primary database to be applied to secondary databases until the data modification errors are addressed. In this case, a secondary database not yet having a data modification error applied could become the primary database until the data modification error is addressed. Afterward, normal processing can resume.

Like in the case of distributed availability groups, you can use log shipping for cross-platform solutions where, for example, the primary database is running on Linux while secondary databases are on Linux and Windows.

The following diagram illustrates a cross-platform deployment with log shipping. Note that there is no common configuration across regions like a distributed availability group in this topology.



The availability groups are in separate regions, with one running on Linux and the other on Windows.

For more information about SQL Server log shipping, read About Log Shipping (SQL Server) (https://docs.microsoft.com/sql/database-engine/log-shipping/about-log-shipping-sql-server?view=sql-server-2017)

.

Combining SQL Server availability features

You can deploy SQL Server availability features in different combinations. For example, in the preceding use case, log shipping was used with different availability groups installed on different operating systems.

The following is a list of possible combinations of SQL server availability features:

- Use log shipping between availability groups installed on the same operating system.
- Have a primary availability group using FCIs with a secondary availability group that uses only SQL Server standalone instances.
- Use a distributed availability group between nearby regions, and log shipping across regions located in different continents.

These are only some of the possible combinations of SQL Server availability features.

The flexibility that SQL Server availability features provide support fine tuning of a disaster recovery strategy according to the stated requirements.

SQL Server replication

SQL Server replication isn't generally considered an availability feature, but this section describes briefly how this feature can be used for disaster recovery.

The replication feature supports the creation and maintenance of replicas of databases. Different types of SQL Server agents collaborate to capture changes, to transmit captured changes, and to apply those changes to the replicas. This process is asynchronous and replicas usually lag behind the replicating database to various degrees.

For example, it's possible to have a replica of a production database. In terms of disaster recovery, the production database is the primary database and the replica is the secondary database. The SQL Server replication feature doesn't know that the databases assume different roles in the context of disaster recovery. Hence, replication doesn't have operations that support the disaster recovery process, for example, role changes. The disaster recovery process has to be implemented separately from the SQL Server functionality and run by the organization implementing because there are no client access abstrations.

Backup file shipping

Backup file shipping is another disaster recovery implementation strategy. A standard approach to setting up and continuously updating a secondary database is taking an initial full backup of the primary database and incremental backups of it thereafter. All incremental backups are applied to secondary databases in the correct order. There are many variations to

this approach depending on the frequency of incremental backups and then backup file storage location (global location or actually copied between locations).

This strategy doesn't involve any SQL Server availability feature when replicating state changes from the primary database to any secondary database. It doesn‘t use the SQL Server Agent that is used in the case of log shipping.

For more information, see the section about Example: backup-and-restore DR strategy (#example-backup-and-restore-dr-strategy).

Detailed backup and restore instructions of SQL Server are discussed in the following solution: Using Microsoft SQL Server backups for point-in-time recovery on Compute Engine (/solutions/backup-and-archival-of-sql-with-point-in-time-recovery).

Compared to the replication approach discussed in the previous section, both replication and backup file shipping have in common that the disaster recovery process is implemented outside and separate from the SQL Server feature set. From the perspective of shipping captured changes, SQL Server replication is more convenient as it implements this part automatically by means of SQL Server Agents.

Note on interaction between database lifecycle and app lifecycle

A database failover isn't fully separate and independent of the apps accessing the database. In principle, there are two failure scenarios.

First, the app stays operational while the database is failing over. From the time of primary database unavailability to the point of the new primary database being operational, the apps cannot access the database at all. Existing connections fail and no new connections are established. During this time, the app is unable to serve its clients, at least to the extent where functionality requires database access. The apps need to recognize when the new primary database is available so that they can resume normal processing.

Apps might have state outside of the database, for example, in main memory caches. The app makes sure the cache is consistent (synchronized) with the new primary database. If there was no loss of transactions at all during failover, the cache might be consistent without further maintenance. However, if transaction (data) loss took place during failover, the cache might not be consistent in relation to the new primary database. The analogous discussion applies for shared state when for example, some of the data in the database is also part of messages

in queues, or files in the file system. This aspect of data consistency is outside the scope of this document because it isn't directly related to database disaster recovery.

Second, one or more apps might become unavailable at the same time the primary database becomes unavailable. For example, if a region goes offline, an application system executing in that region will be as unavailable as the primary database in that same region. In this case, the app has to be recovered as well, not just the primary database system. Along with the database disaster recovery process, you need to initiate a similar app recovery process. The recovered app must connect to the new primary database and be reconfigured (for example, floating IP addresses). App recovery is outside the scope of this document.

Relationship between backup and restore to disaster recovery

Backing up a database (https://docs.microsoft.com/sql/relational-databases/backup-restore/create-a-full-database-backup-sql-server?view=sql-server-2017) is independent and orthogonal to database disaster recovery. The purpose of database backup is to be able to restore a consistent state, for example, in case a database is lost or becomes corrupt, or a previous state has to be recovered due to app failures or bugs. Backup, archival, and recovery in context of Google Cloud is discussed in Using Microsoft SQL Server backups for point-in-time recovery on Compute Engine (/solutions/backup-and-archival-of-sql-with-point-in-time-recovery).

The following section discusses how you can use backups as one possible mechanism to implement database disaster recovery. In this scenario, you copy backup files to the secondary database's location so that the secondary database can be restored. However, backup files are not a prerequisite for disaster recovery; the previous discussion of the availability features presented alternatives.

High availability and disaster recovery

Both high availability and disaster recovery have in common that they provide solutions for database unavailability. If a primary database becomes unavailable, then a secondary database becomes the new primary database that is consistent and available.

The difference between high availability and disaster recovery is the single point of failure domain. High availability addresses an outage within a region, for example, when a single zone fails or a node fails. A high availability solution provides a new primary database in another

zone in the same region. In addition, high availability addresses node failures, not only database failures. If a node running a SQL Server instance fails, a new node is made available running a new SQL Server instance (see the discussion in section Always On Failover Cluster Instance (#always-on-failover-cluster-instance)).

Disaster recovery involves at least two regions. It addresses the case where a whole region becomes unavailable. Disaster recovery can provide a new primary database in a different region.

The SQL Server high availability features support solutions for high availability and disaster recovery at the same time. A single availability group can span the zones within a region as well as regions themselves. An availability group can contain Failover Cluster Instances to address high availability.

SQL Server can establish availability groups within one region for high availability and zone failures, and combine it with log shipping across regions to address disaster recovery.

DR deployment alternatives

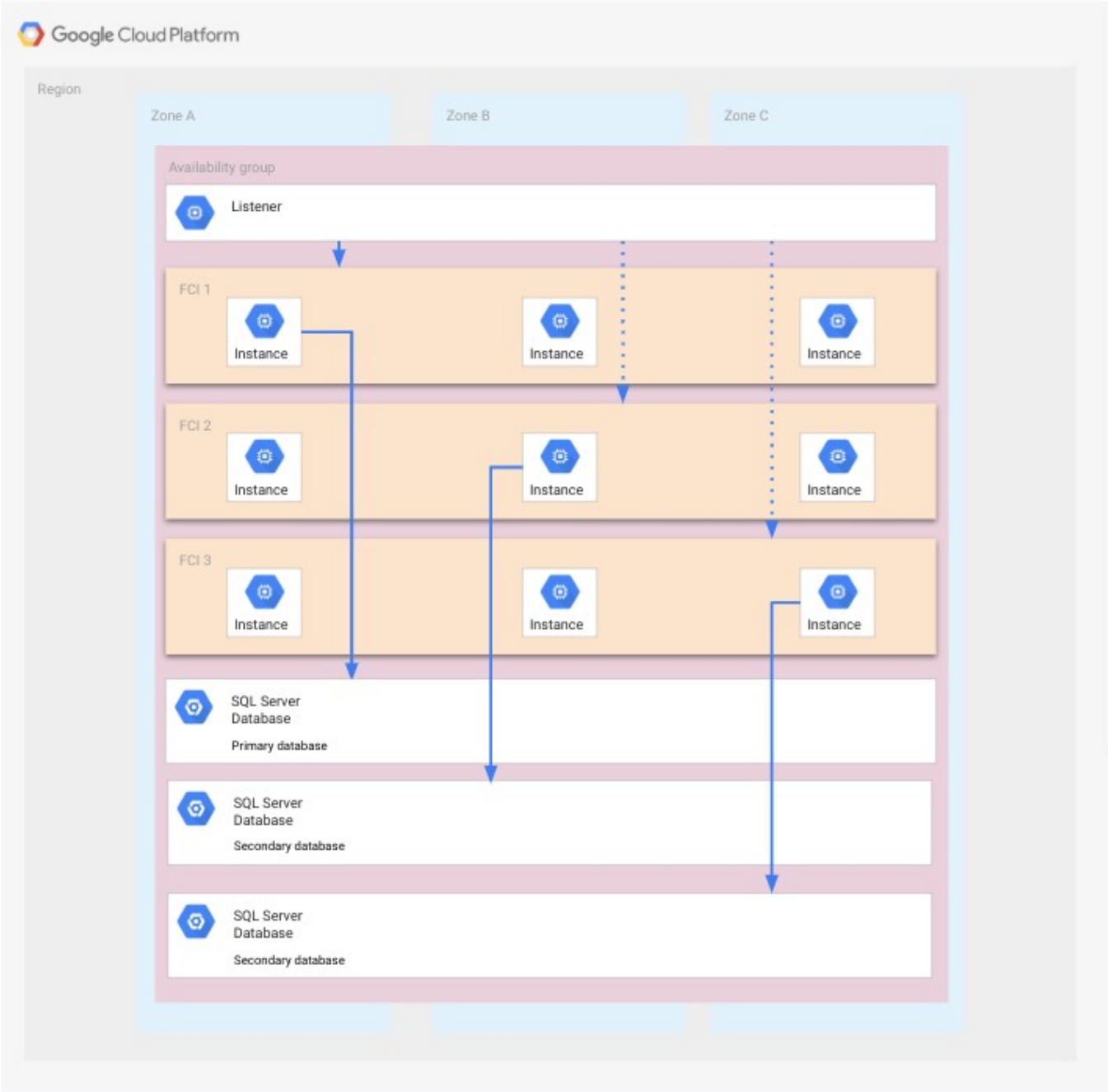
In the following sections, a few possible disaster recovery topologies are shown in addition to those discussed so far. These topologies satisfy different RPO and RTO requirements. This list isn't exhaustive.

Intraregional DR and HA

This deployment is a variation of an availability group containing FCIs, within a region consisting of three zones. Zones are considered the single point of failure domain in this scenario.

Compared to the deployment shown earlier, each FCI consists of three nodes where each node is running in a different zone. The benefit of this setup is that any one or any two zones can fail without requiring a disaster recovery process.

The following diagram shows this setup.



FCIs span all zones, and each FCI has one running SQL Server instance accessing the corresponding database. There are two more SQL Server instances that aren't running in each FCI that can be started up when a zone fails. The databases are shown across zones as each database uses the disks of all nodes in a given FCI. An app isn't shown for clarity.

Interregional DR: availability group spanning regions

In this scenario, an availability group runs on a Windows Server Failover Cluster and spans two regions. Regions are considered a single point of failure domain.

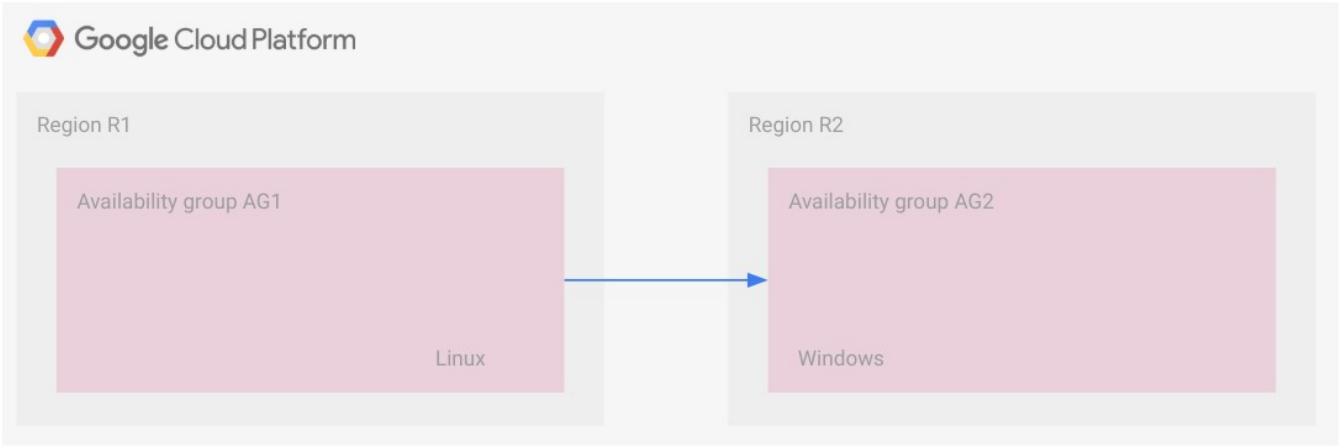
The following diagram illustrates this setup.

In order to address potential latency issues, you can configure the replicas within region R1 to use synchronous transaction propagation whereas the replicas in region R2 are configured to use asynchronous transaction propagation.

Interregional DR: backup file transfer

This scenario uses backup file transfer. Two availability groups in two regions are linked. Each availability group has its replicas receiving the transactions synchronously, therefore, each region's secondary replicas are in a hot standby configuration.

The following diagram illustrates this setup.



However, the two availability groups are connected by backup file transfer. The availability group AG1 is the primary availability group and the availability group AG2 is the secondary availability group. As the backup files are made available to the secondary availability group, they're applied there. This scenario is discussed in more detail in the following section, Example: backup-and-restore DR strategy (#example-backup-and-restore-dr-strategy).

Dual location and tertiary location topology

If there are only two databases, a primary database and a secondary database each in a separate region, then there is an unprotected duration after a failover from the time the new primary database is running and the new secondary database is ready. If the new primary database becomes unavailable while the secondary database isn't yet running, then a hard downtime occurs that can only be recovered from when a new primary database is established. The same applies to availability groups.

A third location running another secondary database or availability group can eliminate the unprotected duration after a failover. This setup needs to ensure that one of the two secondary databases remains a secondary database and is reassigned to a new primary database so that no data loss occurs. Like before the same applies to availability groups.

DR lifecycle

Regardless of the disaster recovery solution you choose, there are common lifecycle steps that apply.

In an actual disaster recovery situation, all stakeholders (app owners, operation groups, and database admins) need to be available and actively participate in managing the disaster recovery. The stakeholders have to decide on the decision authority (a master of ceremony, MC) and the decision processes they follow. In addition, the stakeholders have to agree on their terminology and communication methods.

Deciding on starting a failover process

Unless failover is initiated automatically, the stakeholders must make a decision to initiate a failover. The various stakeholders need to tightly coordinate on the decision when they decide to start failover.

Starting a failover process depends on several factors, mainly on the root cause of the primary database becoming unavailable.

If the disaster recovery process takes longer than the expected time to resolve the primary's database unavailability, then a failover would be detrimental. First, you must assess if restoring the primary database is a feasible option.

The better the disaster recovery strategy is tested, and the faster it's implemented, the easier it is to initiate the failover process because less uncertainty is to be considered in the decision.

Failover process execution

The failover process is ideally tested regularly and therefore, well known to the various stakeholders.

The MC must be aware of all steps that are taking place and of all unexpected problems coming up. The MC drives the failover process and the stakeholders are responsible for supporting the MC.

You should keep statistics for postmortem analysis and failover process improvement; including durations of activities, issues that came up, and any confusion in the failover process steps.

Missing protection

If you only have one secondary database, from the time the new primary database is available and operational until a new secondary database is setup, no DR protection exists. An unavailability during this time might cause a hard downtime because no failover to another database is possible. If that situation arises, another primary database has to be set up and the RPA is the last point that can be reconstructed based on the available backups.

Unless the disaster recovery strategy is set up so that there is protection at all times, every stakeholder has to be aware of this duration of missing protection to take extra precautions during setup or environment configuration changes.

You can avoid this unprotected time if app access to the new primary database is delayed until the new secondary database is up and running. As soon as changes from the primary database are applied, the primary database is available to apps. While this approach avoids any time where apps are unprotected from DR, it delays the completion of the disaster recovery process.

Avoid split-brain situations

It is important that apps can't access a primary database and a secondary database at the same time, issuing DML operations. In this situation, a data inconsistency happens where both

the primary database and the secondary database disagree on data values of the same data item (Split-brain (https://wikipedia.org/wiki/Split-brain_(computing))). This architecture is especially important if the primary database becomes unavailable while it's continuing to run and can receive write operations. If the unavailability is caused by intermittent network partitioning, the partitioning can stop at any point, and an app might have access again. If a failover process is happening at that time, then changes to the old primary database might be lost, or some apps start operating on the new primary database while others are still accessing the old primary database.

All app access is turned off to any database during the failover process so that no state change can happen in any of the databases. After the failover, only one database is available for write operations–the new primary database.

Declaration of completion

After the failover process is completed, all stakeholders need to be explicitly informed by the MC that the process is done. Any issue showing up after completion needs to be treated as a separate incident that isn't part of the failover process anymore but part of regular processing. The issue might be a consequence of a problem with the failover process, or an independent issue altogether. However, the approach of addressing the issue after the failover process is completed might be different from how it is addressed during the execution of the failover process.

Post mortem analysis and report

For future reference and to improve your failover process, arrange a post mortem analysis immediately to make note of important aspects, findings, and action items.

Write a report that summarizes the disaster recovery event, root causes, and all actions taken. This report might be mandatory if you're implementing regulatory requirements.

DR test and verification

Because disaster recovery isn't a part of normal day-to-day operations, your DR solution has to be regularly tested to ensure its proper functioning when it's actually needed.

The frequency of the testing depends on the operational requirements and varies by database, app, and enterprise. In addition, changes to the environment, such as network configuration changes and infrastructure component updates should trigger a disaster recovery test if the changes are made to the systems that the chosen disaster recovery solution relies on. Any change might cause the disaster recovery solution to fail, or might require adjusting the disaster recovery process.

You can test manually by starting the switchover process, or automatically by following a chaos engineering approach as described in Chaos engineering (https://wikipedia.org/wiki/Chaos_engineering). With manual testing, you can minimize business impact in case noticeable downtime is expected.

An important aspect to testing is collecting well-defined statistics. Some important statistics to consider are as follows:

- Recovery time actual (https://wikipedia.org/wiki/Disaster_recovery#Recovery_Time_Actual): measure the actual recovery time and compare it with RTO.
- Recovery point actual (https://wikipedia.org/wiki/Disaster_recovery#Recovery_Point_Objective): observe the actual point of recovery and compare it with RPO.
- Time to failure detection: the time it took for DBAs or the operations team to realize the need for failover.
- Time to recovery initiation: the time it took to start the failover process after the failure was detected.
- Dependability: how closely the failover process was followed or were deviations from it required? Did unexpected issues arise that need to be investigated, possibly resulting in a change of recovery strategy?

Based on the statistics collected, the failover process might have to be adjusted or improved to better match the RPO and RTO expectations.

Example: backup-and-restore DR strategy

The following sections outline an example backup-and-restore disaster recovery strategy. This scenario minimizes the use of SQL Server availability features in order to demonstrate the effort required to specify a DR backup-and-restore strategy and to discuss aspects that are invisible in more automated setups. It refers to the related solution, Using Microsoft SQL

Server backups for point-in-time recovery on Compute Engine (/solutions/backup-and-archival-of-sql-with-point-in-time-recovery).

Use case

A primary Always On availability group is located and operational in region R1. The secondary Always On availability group, is added in region R2 for additional cross-regional protection and available as failover or switchover target.

Strategy

The disaster recovery strategy is based on database backups. An initial full backup is taken followed by subsequent differential backups. The backups are applied to the secondary Always On availability group as they are taken. All backups are stored in a Cloud Storage bucket.

In this example, it's acceptable after failover completion that the new primary Always On availability group in R2 is active and unprotected for a limited time until the new secondary Always On availability group in R1 is operational.

No fallback has to take place as the Always On availability group in each of the regions is equally qualified to serve as a production Always On availability group.

RTO and RPO

RPO is defined in this example to be a maximum of 60 minutes so a differential backup is taken every 60 minutes.

RTO isn't set explicitly to a duration of time, but is to be as minimal as possible— immediate is the best case. The secondary availability group has to be set up as a hot standby. In case of a hot standby, all backups are immediately applied so that the failover isn't delayed applying backups.

High-level DR strategy

The following sections outline the DR strategy. It is kept brief to focus on the essential steps.

Initial setup

- Create secondary Always On availability group in region R2.
- Prevent app access to the secondary availability group so that no split brain situation can accidentally occur.
- Create backup file bucket B1 in Cloud Storage to contain the initial full backup of Always On availability group in R1 and the subsequent hourly differential backups of Always On availability group in R1. The correct order of the differential backups must be established so that the process applying the backups to the secondary availability group can infer the correct order. One approach might be a naming convention that allows establishing the correct chronological order based on date and time being part of the various file names.

Launch strategy

- Apply the full backup to the secondary Always On availability group in region R2.
- As differential backups become available, apply those immediately to the secondary Always On availability group in R2. The immediate application is necessary in order to address the RTO.
- After the initial full backup and all incremental backups are applied, the secondary Always On availability group is ready.
- Test the DR strategy by performing a switchover from the primary availability group to the secondary availability group. At least one incremental backup should be available during testing.

Failover or switchover case

- In R2, the essential steps are as follows:
    - Ensure that the latest differential backup was applied to the secondary Always On availability group in R2.
    - Designate R2 as the new primary Always On availability group.
    - Create a new bucket B2, take a full backup as a baseline, and open the new primary availability group for app access.
    - Start taking differential backups.

- In R1, the essential steps are as follows:
    - Remove bucket B1 because it isn't needed anymore.
    - When the Always On availability group in R1 becomes available again (as a new secondary Always On availability group), prevent app access, and remove all data from the database or reset it to its initial (empty) state (unless it had to be freshly created).
    - Apply the full backup from the new primary Always On availability group in R2, and keep applying differential backups immediately as they become available (stored in bucket B2).

Possible improvements

One possible improvement to the DR strategy is to avoid taking a full backup after a failover or switch over, while still being able to set up the new secondary availability group quickly. Instead of a single full backup and subsequent differential backups, take a full backup every week and create a weekly bucket that contains the full backup of the week and all subsequent differential backups for that week. The new primary availability group has to create differential backups only after failover (and not a full backup) and add those to the bucket. The new secondary availability group simply applies all backups in the current week bucket. If this weekly approach is used, you need to implement a cleanup or purge strategy to remove obsolete backups.

Another improvement is based on the fact that the new secondary availability group was the former primary availability group. If the database exists and is operational after being available again, a point in time recovery to its last differential backup avoids having to fully restore it from the last full backup as described in Restore a SQL Server Database to a Point in Time (Full Recovery Model) (https://docs.microsoft.com/sql/relational-databases/backup-restore/restore-a-sql-server-database-to-a-point-in-time-full-recovery-model?view=sql-server-2017). This scenario reduces the effort and the amount of time the new primary availability group is unprotected.

Production best practices

This solution doesn't specify if the SQL Server instances in the Always On availability groups are standalone or FCI instances. The type of instances used needs to be decided before implementation.

Until a new secondary Always On availability group is operational after a failover, there is a time where DR isn't protected. You should set up a third Always On availability group in a third region.

In addition, you should implement monitoring to ensure that any failure or error is detected. Monitoring is outside the scope of this document, but is essential for a working disaster recovery solution.

What's next

- Configuring SQL Server Always On availability groups (/compute/docs/instances/sql-server/configure-availability).
- Deploying a multi-subnet SQL Server 2016 Always On availability group on Compute Engine (/solutions/deploy-multi-subnet-sql-server).
- Configuring SQL Server Failover Cluster Instances (/compute/docs/instances/sql-server/configure-failover-cluster-instance).
- Running Windows server failover clustering (/compute/docs/tutorials/running-windows-server-failover-clustering).
- How to enable Cloud Logging, Cloud Monitoring, and Error Reporting for .NET apps (/blog/products/gcp/how-to-enable-google-stackdriver-logging-monitoring-and-error-reporting-for-net-apps)
- Installing the Cloud Monitoring agent (/monitoring/agent/install-agent).
- Try out other Google Cloud features for yourself. Have a look at our tutorials (/docs/tutorials).

Last updated 2020-04-20.