Google Cloud AlloyDB for PostgreSQL Cross-Region Disaster Recovery Process

Architecture Solution Document
Christoph Bussler, January 2023

Objectives

The objective of this architecture solution is to demonstrate a complete cross-region disaster recovery protocol for AlloyDB for PostgreSQL.

After a complete execution of the disaster recovery process the end state is the same as the initial state of a primary AlloyDB cluster in one region and a secondary AlloyDB cluster in a second region. This is therefore a full “round-trip disaster recovery” after the disaster causing a failover and fallback is resolved.

Architecture and failover process overview

This section introduces the different phases of a full round-trip disaster recovery. These phases are referred to later in the specific instructions in order to show how the different phases can be executed. All phases together define the complete disaster recovery process.

For textual brevity “primary cluster” and “secondary cluster” are synonyms for “primary AlloyDB for PostgreSQL cluster” and “secondary AlloyDB for PostgreSQL cluster” in this document.

Phases of failover and recovery

There are three phases in the disaster recovery process. These are outlined next as an overview.

- **Phase 1**: initial state
  - The primary cluster is in one region (region 1), and a secondary cluster is in a second region (region 2).
- Disaster in region that hosts primary cluster
  - A disaster takes place in the region of the primary cluster (region 1).
  - **Note**: if a disaster in the secondary cluster’s region occurs (region 2), another secondary cluster is established in an available third region. This case is not further outlined in this architecture solution as it is straightforward to implement.

- **Phase 2**: promoting the secondary cluster in region 2 to primary cluster and creating a new secondary cluster in region 3.
    - A new secondary cluster has to be created in region 3 after the original secondary cluster in region 2 was promoted to the new primary cluster.
- **Phase 3**: fallback to initial state
    - This phase establishes a primary cluster again in region 1 and a secondary cluster in region 2.
    - This phase concludes the disaster recovery process.

Each phase is presented in more detail in separate sections below, with corresponding `gcloud` commands for you to get familiar with or to execute. In case edge cases or different variants of a phase exist then these are explained as well.

Secondary clusters

In the following a few notes on secondary clusters and their properties.

Replication duration

When a secondary cluster is established, replication from the primary cluster starts. Replication is not instantaneous, so it is important to wait for the replication to be complete before further cluster operations are applied like promoting a secondary cluster to a primary cluster.

Single secondary cluster restriction

Only one secondary cluster can exist for a primary cluster. This is a restriction that during the disaster recovery might cause a primary cluster not having a secondary cluster associated with it for a period of time. A secondary cluster might have to be deleted before another one can be created.

This happens when a primary cluster has a secondary cluster after failover, but needs another secondary cluster in a separate region in order to get back to the initial state. This is discussed in Phase 3 below in more detail.

One way to protect a primary cluster is to ensure a recent backup is taken before work on the secondary clusters commences.

Example disaster recovery process

The following table shows an example of a complete disaster recovery process. Note that the names of the clusters are independent of their roles in the respective phase for clarity as the roles of a cluster might change. Implementing the role change by cluster naming convention is possible, but not advised, as it might result in naming confusion, especially when looking at logs.

Cluster	Role	Region	Region #
Phase 1: initial state			
cluster-1	Primary	us-west4	1
cluster-2	Secondary	us-central1	2
Disaster in region of primary cluster (us-west4)			
cluster-1	<inaccessible>	us-west4	1
Phase 2: promoting secondary to primary and creating new secondary			
cluster-1	<deleted>	us-west4	1
cluster-2	Primary	us-central1	2
cluster-3	Secondary	europe-west3	3
Phase 3: fallback to initial state			
cluster-3	<deleted>	europe-west3	3
cluster-4	Secondary	us-west4	1
cluster-4	Primary	us-west4	1
cluster-5	Secondary	us-central1	2
cluster-2	<deleted>	us-central1	2
Disaster recovery process completed			

The available regions for AlloyDB can be found here: AlloyDB locations | AlloyDB for PostgreSQL | Google Cloud. Since this might change over time, additional regions might be available at some point in the future and gives you additional possibilities of primary cluster and secondary cluster placement.

Note: this solution shows a cross-region disaster recovery approach. It does not discuss failover between instances within an AlloyDB cluster. For the latter see Fail over the primary instance manually | AlloyDB for PostgreSQL | Google Cloud.

Alternative approach to disaster recovery process

The philosophy followed in this solution is that there is a disaster recovery deployment with a primary cluster and a secondary cluster in every phase along the disaster recovery process, especially after failover.

This is not the only possible approach, however. Some alternatives are discussed next.

Temporary primary cluster without secondary cluster

After a disaster takes place, you could decide to only have a primary cluster (and no secondary cluster) until the disaster has been resolved. This exposes you to the risk that the region becomes inaccessible during the disaster recovery process that hosts the (new) primary cluster. In this case you lose the primary cluster without the ability to failover again. While probably extremely unlikely that a second region becomes inaccessible, you have to decide if this approach is sufficient for your situation.

Failover, but no fallback

Another variant of the process is to failover, but not fallback. This means to execute phase 1 and phase 2, but not phase 3. If the database clients can tolerate that the primary cluster is in different regions, a failover without a fallback might be a sufficient disaster recovery strategy.

Prerequisites

A few prerequisites are necessary before following the instructions in this solution.

- Configuring connectivity to AlloyDB as outlined here: Configure connectivity to AlloyDB. Please execute these instructions before continuing.
- Additional prerequisites are not further discussed in this solution, like having the necessary APIs enabled like the AlloyDB API itself (`alloydb.googleapis.com`), `compute.googleapis.com`, and `servicenetworking.googleapis.com`. You can fulfill these requirements as they are presented to you.

Phase 1: Setting up primary cluster and secondary cluster

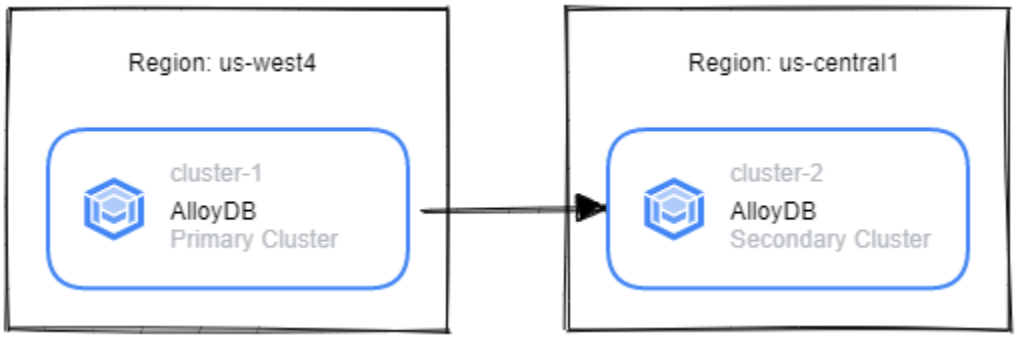

Figure 1: Primary cluster and secondary cluster

This phase sets up a primary cluster and a secondary cluster in two different regions.

For cost efficiency when executing this solution as a trial automatic backups are disabled. In a production environment, however, a proper backup strategy is advised based on the backup functionality AlloyDB provides.

Note: the `gcloud` commands in the following are using the `beta` designation as the service is in beta currently. Once the service is coming out of beta, and the beta designation is dropped by Google Cloud, then drop it from the below commands as well.

Setting up primary cluster

The following two commands set up a primary cluster and a primary instance. The configuration is kept to a minimum for purposes of explanation. Please see About AlloyDB | AlloyDB for PostgreSQL | Google Cloud for additional configuration options that are more appropriately suited for your production deployment.

```
gcloud beta alloydb clusters create cluster-1 --region=us-west4
--password=postgres --disable-automated-backup

gcloud beta alloydb instances create instance-1 --cluster=cluster-1
--region=us-west4 --instance-type=PRIMARY --cpu-count=2
```

Setting up secondary cluster

The following two commands set up a secondary cluster for the primary cluster in a separate region and a secondary instance as well.

```
gcloud beta alloydb clusters create-secondary cluster-2 --region=us-central1
--primary-cluster=projects/alloydb-dr/locations/us-west4/clusters/cluster-1

gcloud beta alloydb instances create-secondary instance-2 --cluster=cluster-2
--region=us-central1
```

Note: No `cpu-count` is specified for the secondary instance. AlloyDB ensures that this instance configuration corresponds to the primary instance.

Console display

The initial state looks as follows in the console:

Resource name ↑	Resource type	Location
cluster-1	Primary cluster	us-west4
instance-1	Primary instance	us-west4-b
cluster-2	Secondary cluster	us-central1
instance-2	Secondary instance	us-central1

Disaster takes place

A disaster does not necessarily mean a crash and burn of all zones in a region because of severe weather or other reasons. A region might not be accessible because of network problems. Another reason might be an electrical fault. Another reason might be a cloud service outage. There are many reasons why a region might not be able to support the execution of an AlloyDB cluster.

In principle terms, a disaster means that the primary cluster is not accessible for a period of time that is considered inacceptable by applications that access the primary cluster as they would be inoperational for that period of time.

For purposes of this solution, the region of the primary cluster (`us-west4` in the example) is considered inaccessible and that constitutes a disaster. The organization decides to failover the primary cluster to the secondary cluster in order to reestablish the database service.

Simulating a disaster would be possible by stopping or deleting the primary cluster and promoting the secondary cluster. In the context of AlloyDB, however, it is not possible to delete the primary cluster without having deleted the secondary cluster first - and this would defeat the purpose. It is also not possible to stop an AlloyDB cluster preventing it from serving clients.

Therefore, we are left with promoting the secondary cluster and afterwards deleting the primary cluster. Later on, especially for implementing a switchover (voluntary failover and fallback), the section “Application design considerations” below outlines a recommendation to implement a database access path that supports quiescing database clients.

Phase 2: Promoting secondary cluster to primary cluster and creating new secondary cluster

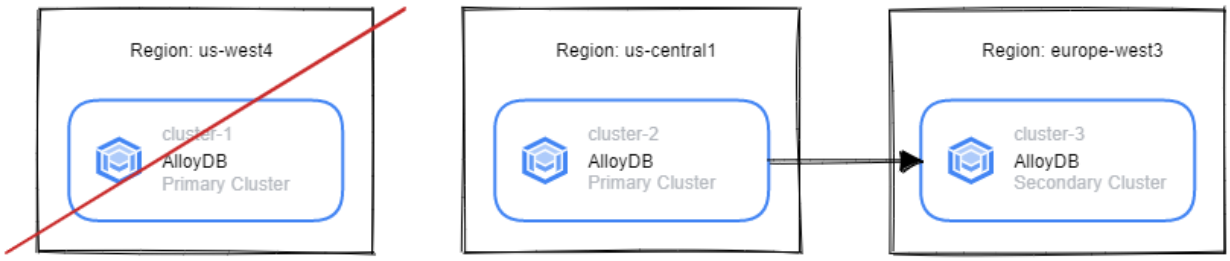

Figure 2: Secondary cluster promoted to primary cluster and new secondary cluster

Since region 1 (in the example us-west4) is unavailable due to a disaster, a new primary cluster has to be established with a corresponding secondary cluster .

Promote secondary cluster to new primary cluster

The following command promotes the secondary cluster to become the new primary cluster.

```
gcloud beta alloydb clusters promote cluster-2 --region=us-central1
```

Establishing new secondary cluster

There are two main cases depending on how quickly the disaster in region 1 is resolved (in the example we follow case 1 below).

Case 1: Region 1 inaccessible for a prolonged period of time

Create a new secondary cluster in a 3rd region to shorten the period of vulnerability of having only a primary cluster to the extent possible. We chose `europe-west3` as there is no third region available in the US currently.

```
gcloud beta alloydb clusters create-secondary cluster-3 --region=europe-west3
--primary-cluster=projects/alloydb-dr/locations/us-central1/clusters/cluster-2
```

```
gcloud beta alloydb instances create-secondary instance-3 --cluster=cluster-3
--region=europe-west3
```

Delete the original primary cluster (to mimic the disaster).

```
gcloud beta alloydb clusters delete cluster-1 --region=us-west4 --force
```

Since this command forces the deletion of the instances (and all stored data) the command provides a prompt for you to confirm that you want to execute it.

Case 2: Region 1 recovered from disaster quickly

It might be that the disaster was short-lived and resolved while the secondary cluster in region 2 was promoted to the new primary cluster.

There are several cases to be considered:

- **New primary cluster in region 2 was not yet put into production:** since in this case the new primary cluster did not receive any modifying queries, production can continue on the original primary cluster in region 1 if it is available again. In this case delete the new primary cluster in region 2 and create a secondary cluster again in region 2 for the original primary cluster in region 1.
- **New primary cluster in region 2 was already put into production**: since in this case the new primary cluster received modifying queries, create a new secondary cluster in region 1 and delete the original primary cluster in region 1 after making it inaccessible to clients.

Managing the original primary cluster

A disaster does not necessarily automatically destroy the original primary cluster in region 1.

Special care has to be taken to manage the original primary cluster to avoid a split brain situation (where both the original and the new primary cluster receive client traffic concurrently). This could happen if region 1 becomes accessible again, and if one or more database clients continue accessing the original primary cluster, and not the new primary cluster that was created because of the disaster.

In the above example the original primary cluster is deleted. However, there is a timeframe between the new primary cluster being available and the original primary cluster being deleted that allows a client to access the original primary cluster. Deletion of the original primary cluster does not solve the split brain problem.

In a production environment an additional abstraction layer has to be implemented to remove the ability to access the original and the new primary cluster. This abstraction is a layer that is aware of the various clusters and which of the clusters is the only actual primary cluster at any given point in time. Any client must access the cluster using the abstraction layer and the abstraction layer makes sure that the correct primary cluster is accessed.

Console display

After phase 2 is complete the following clusters are up and running, one primary cluster, and one secondary cluster, in two regions that are different from the region of the original primary cluster.

Resource name ↑	Resource type	Location
cluster-2	Primary cluster	us-central1
instance-2	Primary instance	us-central1-b
cluster-3	Secondary cluster	europe-west3
instance-3	Secondary instance	europe-west3

Phase 3: Fallback to initial state

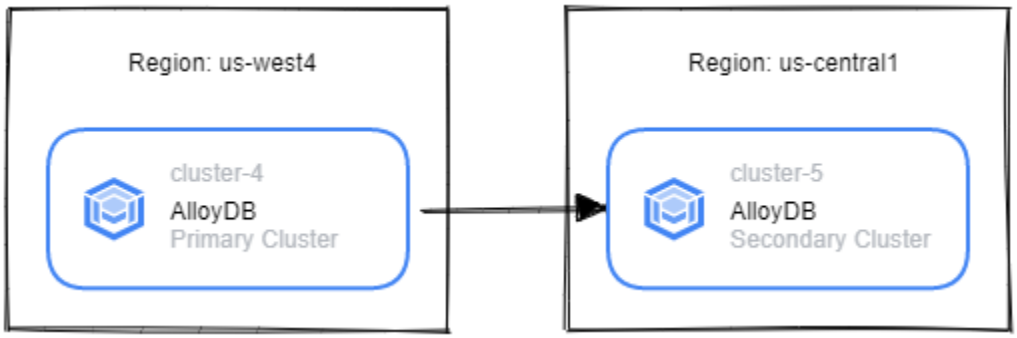

Figure 3: Initial state reached after failover and fallback

Once the failover is completed (phase 2), a stable state is reached with a primary cluster and a secondary cluster.

Phase 2 can result into one of three scenarios

- **Scenario 1**: new primary cluster in region 2, new secondary cluster in region 3
    - This scenario can exist because region 1 has a prolonged disaster and a complete primary cluster and secondary cluster is set up during failover.
    - Phase 3 must now establish a new primary cluster in region 1 and a new secondary cluster in region 2 to be back at the initial state.
- **Scenario 2**: new primary cluster in region 2, new secondary cluster in region 1
    - This scenario can exist because the new primary cluster in region 2 was put into production, but region 1 was available fast enough to establish the secondary cluster in region 1.
    - Phase 3 must now promote the secondary cluster in region 1 and create a new secondary cluster in region 2 to be back at the initial state.
- **Scenario 3**: new primary cluster in region 1, new secondary cluster in region 2

    - This is already the initial state and it came about because the disaster was so short-lived that the original primary cluster was kept as primary cluster and a new secondary cluster was created in region 2.
    - In phase 3 nothing else has to be done for this scenario.

Scenario 1: no secondary cluster in region 1

This scenario establishes a new primary cluster in region 1, and a new secondary cluster in region 2, starting from an existing primary cluster in region 2 and a secondary cluster in region 3.

Note: it is impossible to create more than one secondary cluster for a primary cluster. This means the following for the example: the existing secondary cluster in `europe-west3` has to be deleted first before a new secondary cluster can be created in `us-west4` (that will eventually be promoted to the new primary cluster).

The following are the necessary steps:

- Delete original secondary cluster in region 3
    - `gcloud beta alloydb clusters delete cluster-3 --region=europe-west3 --force`

- Create new secondary cluster in region 1 for primary cluster in region 2
    - `gcloud beta alloydb clusters create-secondary cluster-4 --region=us-west4 --primary-cluster=projects/alloydb-dr/locations/us-central1/clusters/cluster-2`
    - `gcloud beta alloydb instances create-secondary instance-4 --cluster=cluster-4 --region=us-west4`

- Promote secondary cluster in region 1 to new primary cluster
    - `gcloud beta alloydb clusters promote cluster-4 --region=us-west4`

- Create new secondary cluster in region 2
    - `gcloud beta alloydb clusters create-secondary cluster-5 --region=us-central1 --primary-cluster=projects/alloydb-dr/locations/us-west4/clusters/cluster-4`
    - `gcloud beta alloydb instances create-secondary instance-5 --cluster=cluster-5 --region=us-central1`

At this point the initial state is reached and the superfluous clusters can be deleted:

- Delete original primary cluster in region 2

    - `gcloud beta alloydb clusters delete cluster-2 --region=us-central1 --force`

Scenario 2: secondary cluster already in region 1

This scenario requires less steps compared to scenario 1 since the new secondary cluster is already in region 1 and the new primary in region 2 (as established by phase 2).

- Promote secondary cluster in region 1 to be the new primary cluster
- Create new secondary cluster in region 2 for the new primary cluster in region 1

This establishes the initial state again.

- Delete former secondary cluster in region 1 as it is not needed anymore

Cleanup superfluous clusters

When falling back several new clusters will be created and replication between primary cluster and secondary cluster will take time to complete. Only after replication from the primary cluster to the secondary cluster is completed the fully recoverable state is achieved.

While new clusters are created and replication started, it is possible to at the same time delete any superfluous clusters that are not needed anymore as soon as possible. However, from a process execution certainty point of view, it might be more productive to establish a production setup first, before dealing with clusters that are not needed anymore.

Console display

The following shows the state after phase 3 when the system reached the initial state again.

Resource name ↑	Resource type	Location
cluster-4	Primary cluster	us-west4
instance-4	Primary instance	us-west4-a
cluster-5	Secondary cluster	us-central1
instance-5	Secondary instance	us-central1

Regular testing with switchover and fallback

It is highly recommended to test the discovery process regularly by voluntarily executing it by intentional switchover and fallback.

A switchover and fallback does not only test AlloyDB's functionality, but also reinforces the organization's knowledge and process readiness of the disaster recovery process. Regular execution of switchover ensures that everybody who plays a role in the disaster recovery process is aware of its tasks and dependencies.

Over time it is expected that AlloyDB will be available in additional regions. Since region selection is an important aspect of disaster recovery, a regular switchover might include a review of currently available regions. If additional regions are available an update of the recovery process can be performed at this time to place clusters in regions that were not available before.

If special application design considerations were implemented (some possibilities are outlined below) then these are tested in a switchover and fallback as well assuring the correct interplay between the clusters and the accessing application systems.

Application design considerations

Ideally the disaster recovery process can be executed independently of any application logic and without the application systems accessing the clusters even being aware of a disaster recovery process (aside from possibly a short database cluster unavailability).

However, this ideal situation is not always possible to achieve. Discussed next are a few use cases that applications have to be aware of and might require additional abstractions to be implemented.

Preventing client access to a database cluster

It is important to be able to prevent client access to a database cluster so that it is impossible for clients to gain read or modify database access.

For example, if a primary cluster becomes unavailable and the disaster recovery process starts, a secondary cluster is promoted to be the new primary cluster. When the unavailable region becomes available again, the original primary cluster will usually become available again as well.

In order to ensure that the original primary cluster is not being accessed, an abstraction layer intercepting database client access can ensure that a former primary cluster will never be accessed after a failure recovery (also discussed as avoiding the split brain problem). This

abstraction layer must be aware of the different phases of the failover and fallback process in order to ensure that only one primary cluster can be accessed at any point in time.

For the switchover process such an abstraction layer is important as well since the switchover is voluntarily started and a disaster has to be simulated. The best way is to stop clients from accessing the primary cluster - and that also simulates a disaster in the switchover use case.

Database access latency tolerance

In a disaster recovery or switchover process the location of the primary cluster changes to a different region. Applications themselves are executing their logic in specific regions.

In case of a disaster recovery or switchover, the distance (network latency) between the application regions and the primary cluster region might change (increase or reduce, both is possible). Applications must be able to tolerate changes in latency in these cases and have to be designed for it.

Optional fallback after failover

While this solution provided a full round trip in disaster recovery so that after the disaster resolution the primary cluster is in the original region, it does not mean that it is the only possible approach.

It is perfectly acceptable to failover, but not to fallback. This means that if after a disaster a new primary cluster and secondary cluster is established and this is the production deployment going forward. This requires, however, that the applications are prepared for the case that the region of the primary cluster changes.

Cluster naming abstraction

As proposed in this document, the name of the clusters are independent of their role and location (region). The reason is to avoid confusion when a cluster with a name containing a role name is not in the role the cluster name suggests. As you can observe, there are several clusters being created (or promoted), and those have different names.

From an application design perspective the particular cluster names should be abstracted so that the application configuration remains stable despite cluster name changes.

Conclusion

This solution architecture document outlined a full round-trip disaster recovery process that ensures that after a disaster in a region is resolved the original state of a primary cluster and

secondary cluster are re-established. While this is one possible approach, alternatives are discussed as well, like failing over, but not falling back to the original state.

The solution also discusses a few application design considerations as well as the need for regular testing of the disaster recovery process by means of voluntary switchover.

Since the overall process is complex, and the duration directly proportional to the data volume stored, I recommend to start implementing and testing the process regularly to avoid losing data in a disaster situation.