Zero downtime database migration and replication to and from Cloud Spanner

by Christoph Bussler, Szabolcs Rozsnyai

Christoph Bussler
Jul 13 · 12 min read

What is zero downtime database migration and replication?

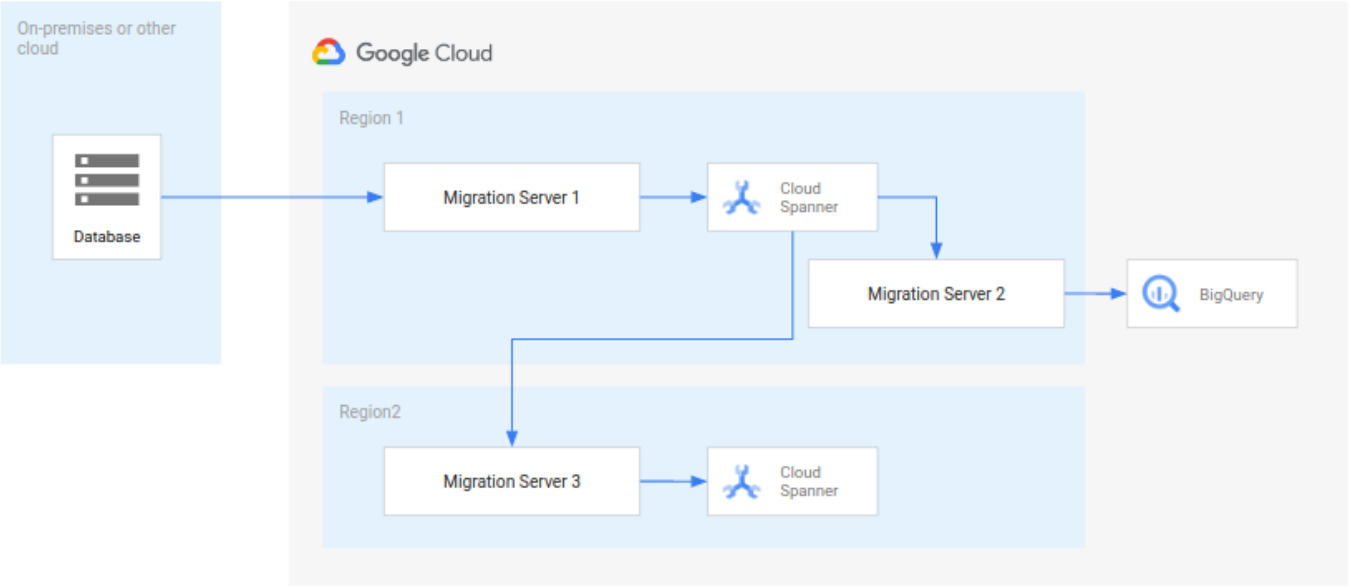
Zero downtime database migration and replication (Database Migration — Concepts and Principles (Part 1), Database Migration — Concepts and Principles (Part 2)) refers to migrating or replicating data from a source database to a target database without impacting the client's access of the source database in terms of availability or scalability. A client can continue to operate on the source database while the database migration or replication progresses.

In the general case, database migration migrates all data from the source database to a target database with the goal to retire the source database and make the target database the primary system and source of truth. The typical use case is the migration from a self-managed database in an on-premise environment to a cloud-managed relational database like Cloud Spanner in the Google Cloud.

Database replication addresses the use cases where data is continuously replicated without the intention to retire the source database. Instead, the data of the source database is made available for downstream processing, for example, analytics in BigQuery.

Zero downtime is better called near-zero downtime for the migration case since clients have to reconnect to the target database after the migration is completed and that will incur a window of being disconnected.

The following diagram shows a generic migration architecture that covers three use cases that are discussed in this blog:



Cloud Spanner database migration architecture overview

- Migration to Cloud Spanner database: migration and replication from an on-premises environment or other cloud
- Migration from Cloud Spanner database: replication to the analytics system BigQuery
- Migration between two Cloud Spanner databases: migration and replication between Cloud Spanner databases in two different regions

The diagram shows a migration technology running on a migration server for each of the three use cases that performs the migration or replication. Depending on the use case it might be the same or different migration technology since not all migration technologies support the same source and target databases.

In your specific context it is possible that you have all three of these use cases at the same time, or only a subset of those.

Key areas of database migration design

No matter the available migration technology, there are key areas of migration/replication design that have to be addressed and resolved. These are

- Schema difference: the source and target database might have a different schema

- Data difference: not all data are migrated necessarily, a relevant subset can be migrated as well
- Missing source database functionality: the source database provides functionality that is unavailable in the target database
- Available target database functionality: the target database has functionality that is not used or available in the source database

Schema difference

In the context of Cloud Spanner the source database and target database schema are different since Cloud Spanner is cloud-managed and has its own relational schema constructs like data types and available schema design options. The only exception is the use case of migrating between two Cloud Spanner databases that have the same schema — of course it is possible in this case as well that the schemas are different.

The schema design of the target database has to ensure that it represents the same data semantics of the source database schema. Tables have to be able to store all data from the source database in structure (tables, columns) as well as data types. The latter might require to map data types from the source database to data types in the target database. If a type does not have a direct equivalence, alternative types have to be chosen that might require transformation of the actual data being migrated during migration runtime.

While automatic schema translation tools would be an interesting technology to have, they can only provide a first approximation as a starting point as some schema design decisions also depend on the transaction mix executing against a schema. HarbourBridge is such an example that is able to generate a Cloud Spanner schema and populate the Cloud Spanner database out of a PostgreSQL pg_dump. Nevertheless, its primary goal is to bootstrap Cloud Spanner evaluation processes as it ignores many many PostgreSQL features (e.g. indexes, stored procedures, constraints) or maps data types that do not have a direct correspondence to STRING(MAX). Iterations after that are manual in order to optimize the schema design. In addition, schema definitions are (configuration) code and as such need to be controlled through a software management system like GitHub.

Data difference

Since migrating to Cloud Spanner is a migration or replication between two different systems, data type transformation might be required for data types that do not have a direct match.

In many cases all data from the source database are migrated to the target database without exception. However, this is not always the case. Database migration provides an opportunity to clean up data and migrate only data to the target database that is actually needed. Filtering can ensure that only data complying to specific criteria are migrated; data not fulfilling the criteria can be dropped during migration (or replication).

Missing source database functionality

Each database engine when it is designed and built has a different design focus and design goal resulting in a different set of available database functionality. Not every feature of every database engine is available in every other database engine.

When migrating to Cloud Spanner there might be features being used in the source database that are not directly available in Cloud Spanner at this point. In such a case the functionality (if still needed after migration) might have to be implemented in the application layer. For example, a source database might utilize stored procedures, (materialized) views, partitions, triggers, functions, sequences or certain constraints which are not provided by Cloud Spanner at this time.

In some cases, however, it is a positive situation as it forces refactoring that an engineering organization was planning for a long time anyway. Over time it has turned out that performance gains from utilizing stored procedures are insignificant compared to the downsides of encapsulating business and transaction logic in stored procedures in addition to the application layer. Stored procedures promote additional maintenance and development overhead, reduce testability (e.g. unit and integration tests), increase vendor lock-in, add complexity to CI/CD pipelines specifically around roll-out scenarios (dev/test/prod) and version control. For example, in a source database autonomous transactions might have been used that caused uncertainty about the database behavior, especially in context of failures. Cloud Spanner does not provide the concept of autonomous transactions, and thereby forces the re-implementation of autonomous

transactions as regular transactions. This improves the clarity of the implementation as well as the system behavior.

Available target database functionality

In many cases the target database has features that are not available in the source database. In Cloud Spanner's case an example is interleaved tables, aka, tables that are in a parent-child relationship. These are available to model 1:n part-of relationships as part of the schema design. Interleaved tables provide increased throughput and reduced latency since the child rows of a parent row are collocated on storage. When joining a parent and its child rows the execution will only have one storage access. However, if the child table is queried independently the performance will have an adverse effect.

Interleaved tables can be used for optimization in context of migrating many-to-many relationships to Cloud Spanner. A many-to-many relationship requires at least three tables, two containing data, and one relationship table. The schema design options to optimize the relationship table depends on the query access patterns (i.e. query direction) of the application. If the n:m relationship is mostly resolved unidirectional the three tables can be interleaved instead resulting into two or only one table, however, this would require data duplication.

In some cases bi-directional low latency queries need to be provided in case of n:m relationships. This can be achieved by creating two relationship tables serving both query directions in an interleaved fashion. Downside is that the application needs to duplicate data to two intermediate tables to resolve the relationships and the resulting table is not normalized in terms of a sound relational model.

Another feature that supports high throughput are indexes using the STORING clause that stores data from the indexed table into the index itself reducing the number of round trips.

Additional design choices with Cloud Spanner need to be made when migrating table keys. While the key data type from the source can be mapped to a Cloud Spanner supported equivalent, the key ranges and queries need to be examined and adjusted to avoid hot-spotting (see more on Schema design best practices).

On the application side, the usage of transactions might need to be adjusted and potentially re-designed. The source database might offer a different set of isolation levels and transaction features that need to be adjusted and re-designed to get the most benefits out of Cloud Spanner or to replicate the same behavior. Cloud Spanner supports strong consistency.

Cloud Spanner supports different types of transactions such as read only (both time-bound stale as well as strongly consistent reads) and read/write transactions, whereas the latter relies on pessimistic locking which may abort transactions, requiring application side retries. These are usually taken care of by the officially supported Cloud Spanner Client libraries, but nevertheless within a transaction scope only idempotent operations are allowed as in case of an error they can't be rolled back and might have side-effects.

Another example are mutations (API) and discussion in DML and Mutations — a tale of two data altering techniques in Cloud Spanner. Mutations is an interface that allows to collect queries as part of a transaction whereby the execution of these queries is only started when the transaction commit is issued. This means that only a single database round trip is needed independent of how many queries are part of the transaction. This increases the throughput significantly from an application perspective.

These Cloud Spanner specific design features (and additional features not discussed here like partitioned reads and partitioned DML statements) need to be taken into account in the context of building or migrating an application to take full advantage of Cloud Spanner and its capabilities.

Database migration technologies

Several different migration technologies are available that provide database migration and replication support for different types of use cases.

- **Migration and replication systems**. Migration and replication systems are technologies that can ingest data from a multitude of sources and migrate or replicate the sources to Cloud Spanner. For example, Striim is such a migration technology. It can connect to many sources using change data capture technology or batch readers and migrate/replicate the received data to Cloud Spanner. A Qwiklabs

hands-on lab shows how zero downtime migration to Cloud Spanner works: Online Data Migration to Cloud Spanner using Striim.

- **Export/import**. If database downtime is an option for the period of data migration from the source to the target, export/import can be used. If you are migrating from another Google Cloud database (or another instance of Cloud Spanner — from a regional instance to a multi regional instance) using export/import, we recommend you use Dataflow to accomplish this. Follow the instructions here for export and import. From the source exported data will have to be transformed in order to match the schema in the Cloud Spanner target database (unless the source is Cloud Spanner itself and the schemas are the same).

- **Incremental batch reading**. Another available technology is incremental batch reading. This approach is based on a column in the source schema that indicates if a row was changed. A batch reader filters for not-yet-read rows, extracts those, and writes those to a target system. Example are Spanner Batch Reader, cloudspannerecosystem/spanner-change-watcher or Deploying event-sourced systems with Cloud Spanner.

- **Evaluation tooling**. Some technology provides the ability to simplify the evaluation of a migration to Cloud Spanner. A tool for PostgreSQL is HarbourBridge.

- **Dual-write**. Dual-write is an often cited approach for zero downtime migration or replication, however, it is fraught with problems and not recommended except for perhaps super-narrow specialized use cases: Online Database Migration by Dual-Write: This is not for Everyone.

The migration technology mentioned above is an overview of the currently available technology and as time passes more technologies will be available and being built.

Migration to Cloud Spanner database

Broadly speaking, the migration or replication of a source database to a Cloud Spanner database has several phases:

- **Migration/replication setup**. This phase addresses the key areas of database migration and replication. One area that cannot be emphasized enough is that Cloud Spanner provides features that are not available in source database systems and

those might make a huge difference in performance, throughput and scalability, like interleaved tables, distributed query processing and dynamic scaling.

- **Testing**. Testing the functionality of the migration as well as the scalability are key and are an important activity.
- **Migration/replication execution**. The execution is a multi-step process that includes initial load and continued migration (for both, migration and replication). In the case of migration additional steps follow: draining, cut-over and source database deletion. The steps are outlined in detail in Database migration: Concepts and principles (Part 2).
- **Possible fallback**. It is possible that problems arise not directly after the cut-over, but down the road, days or weeks after the migration is completed. For these situations it might be important to prepare a fallback plan: this requires a reverse migration setup, as described here.

A discussion in context of a concrete source database system is Migrating from an Oracle® OLTP system to Cloud Spanner.

Migration from Cloud Spanner database

Cloud Spanner cannot only be a target database to migrate to, it can be a source database as well. Since Cloud Spanner does not provide a transaction log or a change data capture interface, the migration tool of choice is an incremental batch reader, as discussed above in Database migration technologies.

As outlined this requires an additional column in order to record when rows have changed so that an incremental batch reader can detect new changes since its last read.

One example, as shown in the architecture diagram above is replicating data from Cloud Spanner to BigQuery for continued business data analysis. The migration system used must not only be able to read from Cloud Spanner, but also transform the data into the configured BigQuery schema, use the BigQuery interfaces for inserting or merging data as well as aware of the limits BigQuery sets when it comes to the data load activities.

Like migrating to Cloud Spanner, a replication from Cloud Spanner to for example BigQuery is a database migration/replication design and execution project.

Migration between two Cloud Spanner databases

A special case is the migration or replication between two Cloud Spanner databases. In this case the source and target database are the same database system, and most likely the schemas are the same.

Use cases for a Cloud Spanner to Cloud Spanner migration or replication are

- **Migration from single to multi-region deployment**. A single region deployment was sufficient initially, however, with changing requirements a multi-region deployment is necessary. This requires a migration from a single region Cloud Spanner deployment to a multi-region deployment.
- **Asynchronous replication**. For reasons of synchronous replication, a Cloud Spanner database is replicated to one or several additional regions. This is a use case where data, once created, must be available for read access in independent geographic areas.
- **Multi-tenant data management.** In multi-tenant deployments tenants' data might be stored in separate databases so that each tenant is independent of any other tenant. It might be required for legal or technical reasons (e.g., noisy neighbor, skewed growth) that a tenant's database must be moved from one Cloud Spanner instance to another Cloud Spanner instance.
- **Cloud Spanner instance consolidation**. Over time several Cloud Spanner instances might have been created and the original architectural and technical requirements changed so that databases can be consolidated in fewer Cloud Spanner instances. In this case databases are migrated between Cloud Spanner instances.

The above use cases are only a subset and additional use cases might exist in customer projects depending on the context and requirements.

Summary

Concluding, the relational database management system Cloud Spanner can be the source and target database system for database migration and replication. Several migration technologies exist that can be deployed to implement the required use cases

therefore making Cloud Spanner an excellent option for migrating with zero downtime to a scalable and consistent relational database management system.

What's next

- Review the complexities and perils of database migration and the migration tools that make it repeatable: Database migration: Concepts and principles (Part 1), Database migration: Concepts and principles (Part 2), Online Database Migration by Dual-Write: This is not for Everyone.
- Setup a database migration yourself in this hands-on lab: Online Data Migration to Cloud Spanner using Striim.
- Learn more about Cloud Spanner and Google Cloud.

Disclaimer

Christoph Bussler is a Solutions Architect and Szabolcs Rozsnyai is a Data Management Specialist at Google, Inc. (Google Cloud). The opinions stated here are our own, not those of Google, Inc.

Google Cloud | Cloud Spanner | Database Migration | Distributed Database | Data Integration