Christoph Bussler

42 Followers · About

Multi-Cloud Database Management

Selecting Databases for Multi-cloud Use Cases

Christoph Bussler Just now · 12 min read

Note: this blog has the same content as the series with the same name. With medium.com series being deprecated, I post the same content unchanged as regular blog here.

This series discusses various aspects of multi-cloud database management:

- What is multi-cloud database management?
- What are use cases for managing databases within and across different public and private clouds?
- What types of database management systems are available that satisfy multi-cloud use cases?
- Which deployment architectures support multi-cloud use cases?

… and many more.

☁ If you are interested, please follow the series as it develops.

. . .

A small request …

Medium.com has optimized the display of the series' cards on mobile phones so that you as a reader can read a series by sliding screens left/right, instead of reading a web page top to bottom.

Let me know if that works, if that is useful and helpful. Leave me a comment with your feedback, experience, impression and advice.

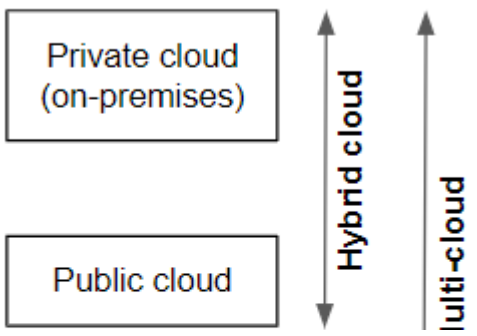
Thank you.

. . .

Multi-cloud

- **(Public) cloud**. A public cloud is an environment providing computing resources shared by many organizations. An example is Google Cloud.
- **On-premises (private cloud)**. A cloud or computing environment owned by or dedicated to a single organization.
- **Hybrid cloud**. A combination of one public and private clouds (on premises) used by an organization.
- **Multi-cloud**. At least two public clouds, possibly including private clouds.

. . .



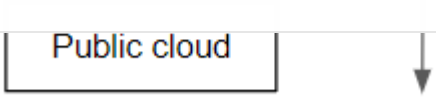


Figure 1: Hybrid cloud and multi-cloud

. . .

Database Application

A database application uses a database to manage its persistent state. It has the following components

- **User interface**. In most cases it is a browser client, mobile client, or installed client.
- **Application logic**. It implements the business logic and runs on any compute environment like virtual machines, Kubernetes, Cloud Functions, Cloud Run, etc.
- **Database**. The database implements data management functionality and it used by a database application to manage its persistent state.

. . .

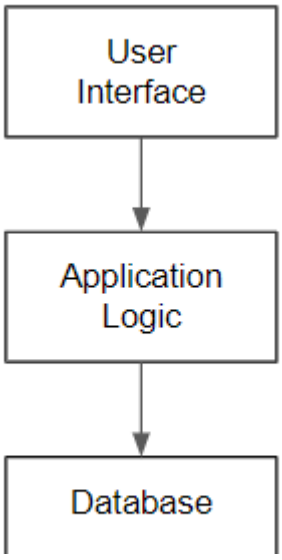


Figure 2: Database application

. . .

Networking

It is implicitly assumed that an organization that uses a multi-cloud setup needs the various involved clouds to be able to communicate with each other. The clouds are in general not used in isolation, since the deployed applications share configuration meta data, management functionality or business data.

One approach is using VPN connections. Here is an example to setup a VPN connection between two public clouds.

. . .

Multi-Cloud Use Cases

There are many different multi-cloud use cases of database applications.

In the following I'll discuss only a select few in more detail to illustrate the existing spectrum. A more detailed and more complete list will follow down the road.

The emphasis will be on the database aspect (see database application components earlier), naturally, and the application logic and client aspects are not elaborated on at all.

Here we go …

. . .

Use Case: Cloud Bursting

An important use case is cloud bursting. This use case is most interesting in the hybrid cloud case where the basic, steady state load can be handled by a private on-premises

Depending on the industry, at certain periods of the year, an increase in client activity exceeds the data center capacity and the cloud provides the additional capacity implementing the bursting needs.

During bursting the application logic also runs in the cloud for increased capacity. On the database side, it might be that the database on-premises is sufficient to handle the bursting.

If it is not sufficient, a database is deployed into the cloud as well. With two databases synchronization becomes important. The easiest would be a distributed multi-master database, more difficult would be data partitioning across two databases between the clouds to avoid conflict. Most challenging is a bi-directional replication model that requires conflict resolution in case of concurrent update of the same data.

. . .

Use Case: Best-of-Breed Service Selection

Clouds are very different from each other in many aspects, especially provider-specific technologies that are only available in a specific cloud.

If the best-of-breed strategy is followed it is possible that different parts of the application logic run on different clouds.

From a database viewpoint this can have several forms:

- The data is partitioned and no relationship exists between the parts in the different clouds.
- The data is replicated, where one cloud contains the main business data, and the other cloud requires only a replica.
- The data is shared and must be consistent in both clouds. In this case a multi-master distributed database is the easiest model.

In general, this use case supports more than two clouds as well.

. . .

Use Case: Cross-Cloud Portability

To avoid lock-in one strategy is to implement the application logic and the database in such a way that the application can be ported at any time to any cloud.

In one variant this use case might require to use the exact same technology in all clouds. This forces a least common denominator approach, as only those cloud services and technologies that are available in all clouds can be used. From a database perspective a database has to be chosen that can be deployed into all clouds.

A different variant is to not base portability on the portability of the technology used, but on the portability of models and patterns. For example, in case of databases, the relational or document model can be used as the means for portability and any database that can support the relational model will suffice. However, only those model elements that are supported by technology in any cloud can be chosen. It will be a common denominator of the model instead of the technology.

. . .

Use Case: Disaster Recovery

To prevent downtime because of a cloud failure, a disaster recovery strategy can be implemented that supports the failover and fallback between a primary and secondary cloud.

For the lowest RPO and RTO a so-called hot-hot standby deployment is required that dictates that the application logic as well as database is available and ready to take on client traffic in all involved clouds.

If some loss of transactions can be tolerated, an asynchronously replicated database system is sufficient. If RPO should be zero, a synchronously replicated master-master database system is the best alternative.

. . .

Additional Use Cases

In addition to above use cases, here are a few more only discussed briefly:

- **Application migration**. A database application is migrated from one cloud to another. In this use case there are two databases involved for the process of migrating data from one to the other. Zero downtime database migration is the preferred approach.
- **Distributed computing**. In this use case the services are deployed in more than one cloud and are serving client requests in all clouds.
- **Cost optimization**. A very extreme use case (in today's terms) is switching between clouds based on the cost model. Any time the cost basis changes an evaluation takes place and the application is migrated to the cheapest alternative.

There are more use cases in the context of multi-cloud database applications as well as additional variants of those discussed here. For the following analysis the discussed list suffices.

. . .

Use Case Discussion

Use cases can be categorized into those that are temporarily involving several clouds (like bursting) or that continuously involve several clouds during normal production operation (best-of-breed).

all at the same time.

Analyzing the use of databases in the patterns leads to four basic patterns that are discussed next.

. . .

Multi-cloud Database Management Patterns

The use cases lead to the following basic multi-cloud database management patterns:

- **Partitioned without cross-database dependency**. This pattern is the simplest: each location or cloud has a database and the databases contain partitioned data sets that are not dependent on each other. An example for this pattern is a multi-tenant application.
- **Asynchronous unidirectional replication**. This pattern contains a primary database that replicates to a secondary database. The secondary database is used for read access. An example for this pattern is best-of-breed with the secondary being used for analytics.
- **Bi-directionally replicated with conflict resolution**. This pattern contains two primary databases that are asynchronous replicated to each other. In the case of the same data written at the same time (e.g. same primary key) causing a write-write conflict then a conflict resolution has to decide during replication which state is the last state. This pattern can be used in situations where a write-write conflict is rare.
- **Fully master-master synchronized distributed system**. This pattern contains a single database that has a master-master setup so that an update of data at any master is transactionally consistent and synchronously replicated. This pattern is used for example in the distributed computing use case.

. . .

systems from the viewpoint of multi-cloud database management. These are

- **Cloud-native databases**. Cloud native databases are designed and built in such a way that they best work in the context of cloud technology. Examples are CockroachDB or YugaByte and they can be deployed into any cloud that supports containers.
- **Cloud provider managed databases**. Cloud provider managed databases are built on cloud provider specific technology and are managed as a database service by a specific cloud provider. Examples are Cloud Spanner and Cloud Bigtable.
- **Pre-cloud databases**. Pre-cloud databases existed before the development of cloud technology (sometimes for a long time) and are mainly running on bare metal hardware as well as virtual machines. Examples are PostgreSQL and MySQL.
- **Cloud partner managed databases**. Some public clouds have database partners that install and manage customers' databases in the public cloud. Customers therefore do not have to manage those databases themselves. Examples are MongoDB Atlas and MariaDB.

. . .

Database System by Distribution Model

Different database management systems are implemented following different distribution models in their architecture. The most important models for databases are

- **Single instance**. A single database instance runs on one VM or one container and is basically a centralized system. The single instance cannot be connected to any other single instance, meaning that replication is not supported by the database system.
- **Multi-instance active-passive**. A common architecture is a multi-instance system where several database instances are linked together. The most common linking is an active-passive relationship, where one instance is the active database instance supporting both writes and reads.

and write transactions while providing overall data consistency.

- **Multi-instance active-active with conflict resolution**. Another not so frequent variant is a multi-instance deployment where each instance is available for write and read access. However, the databases are synchronized in an asynchronous mode. This permits concurrent update of the same data item leading to an inconsistent state. A conflict resolution policy has to decide which of the states is considered the correct consistent state.

. . .

What about Database Sharding?

An orthogonal distribution aspect is multi-instance sharding. Sharding is based on managing (disjoint) partitions of data. Each partition is managed by a separate database instance. On the one hand this scales as more shards can be added dynamically over time, on the other hand cross-shard queries might not be possible.

Each of the above distribution models can in principle support sharding and be a sharded system. However, not all systems provide a sharding option as many have not been designed for it.

. . .

Mapping of Database Systems to Multi-cloud Database Management Patterns

For a given use case like cloud bursting and the database management pattern chosen for it, a database system can be selected based on the database deployment architecture and database distribution model.

In the following examples two different approaches for cloud bursting are discussed separately. First, the database management pattern is chosen, and then the database.

. . .

Example: Cloud bursting with single database

The first example is based on a requirement that only one database system is used.

- **Pattern**. The pattern is fully partitioned with only one database system in the primary cloud, and no primary system in the secondary cloud. This means that the code bursts, but not the database. All database access is served by the single database.
- **Database choice**: Any database supported in the primary cloud is a contender as there is no second database in the secondary cloud. Any deployment architecture and any distribution model fulfils the requirements.

. . .

Example: Cloud bursting with full database replication

The second example is based on a fully replicated state so that primary and secondary clouds are 100% consistent.

- **Pattern**. The pattern is “Fully master-master synchronized distributed system” or “Bi-directionally replicated with conflict resolution” since these are the only two patterns that support synchronization of several instances.
- **Database choice**. Each of the available databases in every deployment architecture has to be reviewed to see if it supports the distribution model across the two clouds. If a lot of conflicts are expected based on the workload behavior a database implementing conflict resolution might not be the best choice.

. . .

Multi-cloud Database Selection

Based on the multi-cloud database management patterns, the database system deployment architecture and distribution model discussed earlier, a high level database selection decision tree supports the following two starting points:

- **New database system**. If you do not have an existing database or you are looking for a different choice, start at this point.

In the following you find a separate section for each of the two entry points.

. . .

Decision Tree: Existing Database System

This part of the decision tree is fairly straightforward:

- If the multi-cloud use case is supported by your existing database system, and if you'd like to keep it, then there is nothing else to decide on. The decision to keep the existing database system is the best if the database is operating reliably and can be managed without any issue.
- Even though you have an existing database system, you might decide to change the system or to explore alternatives. This decision might be important if the current system will not support the use case in the best way (e.g. it cannot provide replication out of the box, or cannot run in both clouds) or you can foresee upcoming challenges, for example, it will not scale as required.

In the latter case continue to the decision tree entry point for a new database system discussed next.

. . .

Decision Tree: New Database System

A new database system (or several) are selected primarily based on the relationship between the data in the two clouds:

- **Partitioned data**. If the data is partitioned and no relationship exists between the data in the clouds, select a database system for each cloud. This can be the same

- **Unidirectional replication**. If one cloud receives replicated data from the other cloud, the choice is to either select a database system that implements replication (active-passive system), or find a replication system as a separate technology that can read from the source database and replicate (write) to the target database if the database system does not support replication itself.
- **Bi-directional replication**. If the use case requires that the data is fully synchronized between the two clouds the decision will be to select a database system that provides transactional synchronous consistency. Or if conflict resolution is acceptable, a database system that replicates in both directions with the ability to detect and to resolve conflicts.

. . .

Mixed Multi-Cloud Use Cases

It is possible that you have or will have more than one use case at the same time. In this situation an overall architecture decision has to be made before starting the database selection process: will one database system have to support all use cases, or will every use case have its own database system?

In the former the use case with the strictest requirements will be driving the selection. For example, if you have a partitioned use case and a fully synchronized use case then the database selected must support the fully synchronized use case. This system by definition will support the partitioned use case as well.

In the latter, for each use case a separate database system selection has to be made.

A mixed scenario is possible as well where some use cases require a dedicated database and other use cases are supported together by one database system.

. . .

concepts and several multi-cloud use cases. Based on a use case analysis several multi-cloud database management patterns were identified. The series then continued with two database categorizations that are relevant for multi-cloud use cases and a mapping of databases to patterns. Finally a decision tree provided you with a starting point to select a database system for your multi-cloud use cases.

This series is a solid foundation for your multi-cloud database journey and equips you with the key aspects to consider and the key decision points for a successful multi-cloud architecture from a database viewpoint.

. . .

Acknowledgements

I'd like to thank James Brookbank for the thorough review and many comments to improve the accuracy of this content.

Disclaimer

Christoph Bussler is a Solutions Architect at Google, Inc. (Google Cloud). The opinions stated here are my own, not those of Google, Inc.

Google Cloud | Multi Cloud | Database Management | Stateful Applications