Severalnines multi-cloud guide

several9s

Table of contents

1

What is multi-cloud?

Look at the software stack of most companies, and you'll typically see reliance on cloud computing providers. One of the big three—Amazon Web Services, Google Cloud Platform, and Microsoft Azure—usually make up most or all of the cloud infrastructure within a company. Increasingly, engineering teams are considering an approach to make themselves less reliant on a single provider. This multi-cloud approach can offer several benefits over a single-cloud solution.

Multi-cloud is the intentional usage of two or more cloud services of any type. Companies achieve greater control, redundancy, independence, flexibility, regulatory compliance, and other advantages through a multi-cloud strategy. The trade-off for these benefits is additional architectural complexity.

The benefits outweigh the drawbacks as large companies continue to adopt the multi-cloud approach. In early 2022, 89% of enterprises had already adopted multi-cloud, according to **Flexera's State of the Cloud report**. Even more of these companies include multi-cloud on their imminent roadmap.

While smaller companies are less likely to have adopted a multi-cloud approach, most could still see the benefits. A startup, for example, could choose the best products from multiple vendors to reduce time to market. Others might want to optimize costs and will move some workloads to a less expensive alternative.

Government regulations also play a hand in the rise of multi-cloud. If data must be stored or collected in a specific region and be subject to the country's laws, a company may risk noncompliance with its existing cloud provider. It takes considerable effort to move all infrastructure to a new vendor. The multi-cloud approach allows the company to comply while the remainder of the workloads can remain untouched.

For all but the most straightforward projects, it can be complicated to adopt a multi-cloud approach. Architectural changes, especially those that cross many areas of a company or application, require updates to processes and code. The move from physical data centers to the cloud has similar challenges. There are assumptions built into any architecture decision that must be changed. A multi-cloud approach requires flexibility to implement—but then enables companies to remove rigidity from their current practices.

Rather than being tied to a single cloud provider, multi-cloud architecture gives engineering teams control over how they build their software. In fact, the multi-cloud approach can even enable non-cloud options such as local server resources, which opens even more opportunities for customization.

Multi-cloud vs. hybrid-cloud vs. multi-environment

A multi-cloud approach includes a combination of providers and environments. Since the overarching purpose is to maximize control, there's not a single approach. Similarly, there's a fragmentation of terminology that the industry uses to describe the practice.

- **Multi-environment** is a broad term describing any use of multiple server environments. For example, running some applications on-prem and other applications in the cloud.

- **Hybrid cloud** is a subset of multi-environment and typically refers to a mix of public and private clouds where applications are extended across environments.

- **Multi-cloud** includes multiple cloud providers, whether public or private.

Frequently, hybrid cloud is seen as a path to cloud adoption for an organization starting from an on-premises (private cloud) architecture. It may also be a final destination when certain workloads are restricted or don't make sense in a public cloud. In the strictest definition, hybrid-cloud deployments choose a single public cloud provider. However, with the movement toward multi-cloud adoption, it's increasingly common to see two or more public clouds alongside any number of private clouds.

Indeed, as an organization adopts a multi-cloud approach, it may be less useful to make technical distinctions between types of environments. In order to flexibly control how applications are served, engineering teams expect a single heterogeneous architecture. In order to integrate with each environment, the differences must be abstracted away wherever possible.

The business and legal distinction between environment types remains important. Most notably, organizations want control over where data resides or is processed and which country's laws the data is subject to.

Data sovereignty means that owners of data have complete control. This extends from users up through the company and its software stack. When control over the data is discontinuous, if, for example, a service in any data layer is managed by an external organization, then data sovereignty is compromised. After all, how can you assume data sovereignty if you have limited to no control over your stack? Commonly this is due to utilizing proprietary databases or being forced to use particular environments for database deployment.

A multi-cloud solution can be used to increase data sovereignty by increasing control over database technologies, environments, and architectures. The added flexibility of a multi-cloud architecture enables choices that retain control over data, end-to-end.

What is a multi-cloud database?

Multi-cloud has become the de facto standard across organizations. Hashicorp's most recent **State of Cloud Strategy survey** found that over three-quarters of respondents are already using more than one cloud, with 86% expected within two years.

With data sovereignty among the drivers to multi-cloud, a company's databases are a large part of the move. Certainly, various computing platforms from public clouds are an important part of the equation. The transport and storage of data is often more complex. Data also factors into regulations around personally identifiable information (PII) and data residency. For that reason, no multi-cloud strategy is complete without an answer for the database requirements.

A multi-cloud database is a collection of database deployments across two or more environments. Often, these environments are public cloud providers but could also include private cloud or any other deployment.

In some cases, database administrators replicate data between the deployments. These scenarios require the same type of database in each environment, serving a similar purpose. The primary goal may be redundancy, or it could enable data residency—ensuring the operators can control where specific data is stored.

Multi-cloud database approaches are not limited to storing or processing data. Data management tasks also include backup and restore, disaster recovery, data migration, and other use cases. Finally, administrators and operators will want to consider database enhancements and efficiency features, such as query optimization or performance enhancements, which could be dependent on the environment.

Any database type is a candidate for multi-cloud deployment:

- Relational (MySQL, PostgreSQL)
- Key-value (Redis)
- Time series (TimescaleDB)

- NoSQL (MongoDB, Elasticsearch)
- Graph (Neo4j)
- Event streaming (Kafka)

Some databases need to remain in the same region as the cloud computing environment that accesses it. Technical factors that determine a multi-cloud database's environment will include the frequency of reads and writes and how quickly each should be reflected.

For example, low-latency applications that deliver results to end users likely require database queries across the same network. On the other hand, internal analytics may process data in batches or periodically update results to an interface that end users won't access.

Companies that adopt a multi-cloud architecture want control over the environments where data and computing resources can be deployed. To fully employ the flexibility of multi-cloud, each environment should be predictable. To implement multi-cloud databases with this heterogeneous architecture often requires additional effort—which is then multiplied across each combination of database type and use case.

2

Benefits & challenges of multi-cloud databases

The path to data sovereignty, flexibility, and control is likely to include multi-cloud database adoption. Most applications require significant effort to migrate from one provider or environment to multiple. There are great benefits to businesses that battle with these complex challenges.

Benefits & challenges at a glance

Benefits:

- **Increase flexibility through choice**
- **Adhere to data residency policies**
- **Reduce reliance on one single cloud provider and avoid vendor lock-in**
- **Decrease cloud costs through efficient deployments**
- **Reduce latency through geographic distribution**
- **Mitigate outages and disasters with redundancy**

Challenges:

- **Small (and large) differences between providers regarding infrastructure concepts and nomenclature**
- **Custom database flavors specific to each cloud provider**
- **Using public provider DBaaS compromises data sovereignty**
- **Database monitoring across multiple services**
- **Differing security policies and procedures**
- **Specific tools needed for each provider**
- **Data egress charges can be consequential**
- **Providers actively discourage interoperability**

Why use a multi-cloud database?

Organizations cite digital transformation, cost reductions, and avoidance of vendor lock-in as the top three in a long list of distinct drivers for adopting multi-cloud databases, according to the latest **Hashicorp survey**. The market intelligence firm IDC further suggests that key benefits of multi-cloud setups include "better performance, 24/7 availability, enhanced security, and greater compliance with regulations."

A multi-cloud database is an elegant solution for several of the most important factors in data architecture. By using more than one data service provider and by doing so in a deliberately partitioned manner, companies can achieve requirements that are difficult, if not impossible, with a single provider.

Perhaps the most obvious benefit of a multi-cloud database is reduced reliance on a single provider. This means no single point of failure and less lock-in to a data vendor. This approach is a great way to mitigate outages and guarantee uptime.

Flexibility to choose a data service provider also lets a company pick the best product for specific needs. This optionality also offers an edge in negotiating rates since the provider knows how easy it is to switch providers in a multi-cloud database setup. Further, the ability to choose efficient deployments can result in substantial operating cost reductions.

For any company operating in multiple real-world jurisdictions, including any organization with an international user base, a multi-cloud database makes it easy to partition data according to different data residency policies. European data can live in Europe, US data in the US, and so on. An additional benefit is that local data is served faster, so latency can be decreased substantially thanks to the consequent geographic distribution.

When multi-cloud databases are hard

The many benefits of multi-cloud bring new challenges, as well. Drawing from experience and from surveys like Hashicorp's State of Cloud Strategy, the following challenges are frequently cited as hindrances of a multi-cloud strategy.

Products differ

The first challenge is perhaps the most obvious: different providers are different. They have different underlying technologies, different user interfaces, different tooling, and different core competencies. Each of these differences needs to be taken into account and managed in order to ensure smooth interoperability. You can't use a provider's DBaaS

because you'll end up sacrificing your data sovereignty.

Monitoring is complex

Good monitoring is essential for any kind of database management, and this job is more challenging when data lives across multiple services. Since providers don't talk to each other, it's up to the architect to piece together cohesive monitoring. Visibility across all services together from one control plane is essential for success when using a multi-cloud database.

Security policies change

It's always important to understand security policies when setting up a database. The challenge is multiplied for each additional provider, each with its own rules and procedures. Any gap in security is a serious risk, so differences in policy require critical consideration when using multiple providers.

Workflows are inconsistent

Inconsistent workflows across cloud providers are yet another challenge when managing a multi-cloud database. Working with each distinct tool requires a distinct skill set, and the combined workflow is more prone to error.

Skills are in short supply

The factors above combine to mean that many organizations struggle to develop the in-house expertise they need to manage a robust cloud infrastructure. In the Hashicorp survey, a skills shortage was cited as by far the number one challenge hindering companies' ability to operationalize multi-cloud.

Data egress may cost you

When moving data from one service to another, there can be charges assessed by the originating data host. These egress charges, if not properly mitigated, will cut into the bottom-line cost savings that a multi-cloud approach can offer.

Interoperability isn’t Included

Lastly, since most cloud service providers strongly prefer that their customers live entirely within their product ecosystem, they may intentionally make it difficult to achieve easy interoperability. While they can’t and won’t actively disallow the use of a competitor, they can and will leave it up to the customer to figure out how to achieve such interoperability on their own. This means that those who want to adopt a multi-cloud database architecture need to find other tools to facilitate their ambitions.

Multi-cloud databases are the new standard

Fading are the days of using a single cloud provider. The rise of the multi-cloud approach across companies of all sizes is driven by the aim to achieve greater control, increased flexibility, and data sovereignty. The added complexity inherent in using multiple providers presents significant challenges, including learning to manage multiple systems with different products, policies, and problem areas. But those that overcome these challenges gain a level of control and optionality not possible otherwise.

A multi-cloud database is a powerful new tool that has emerged from the increasingly crowded landscape of data service providers. When used correctly, this clever data distribution becomes a distinct competitive advantage. In the next section, we’ll be taking a closer look at exactly when and how to deploy a multi-cloud database, including common use cases and important considerations.

3

Multi-cloud database use cases

A multi-cloud database, explained in the previous section, works particularly well for the following use cases:

- **Migrating Applications Between Providers**
- **Disaster Recovery**
- **Cloud Bursting**
- **Environment Optimization**
- **Legal Compliance**

Migrating applications between providers

A multi-cloud database allows you to easily switch between database providers to avoid **vendor lock-in** — a situation where you become dependent on a specific vendor for their proprietary products or services because it's too hard to switch. Even if you use an open-source database, you can still wind up locked into working with a vendor because switching to another would be too costly, complex, or time-consuming.

A key advantage of a multi-cloud database is that you can quickly move to another vendor if your current one makes changes you don't want to live with. For example, if your current cloud provider raises the subscription fee for your database, you can quickly switch to a more cost-effective offering from a different provider. When you control your data, you can more easily choose to move to the cloud database service that best meets your requirements.

Disaster recovery

What are your options if your cloud database goes down? Are your business-critical applications and databases deployed to multiple locations? If you have a single database service provider and they have a major outage, it could shut down your product or service. You need to make sure you can easily switch your applications to another database if a **disaster happens**.

The more distributed your database resources and backups are, the more resilient your applications and services are. A multi-cloud database eliminates single points of failure. You'll have **Hot Standby** clustering, which will immediately replace and take over the primary system in real-time. A multi-cloud database also has clusters with nodes in different regions so you can failover and have non-stop service operations.

Cloud bursting

Demand isn't always consistent, but you want your performance to be. It doesn't take long for applications to reach full capacity, especially if the database runs on a private or hybrid cloud environment. Legacy applications running on an on-premises or hybrid environment can reach max capacity quickly without the addition of cloud resources. You may need to have options to scale rapidly to meet demand, increasing capacity in your local environment as needed. You can use cloud bursting to scale beyond your data center.

Cloud bursting lets you deal with peaks in demand by directing the overflow traffic to a public cloud. When your on-premises or private cloud is at maximum capacity, you "burst" the overflow traffic to public cloud resources. You could burst the traffic automatically or manually. However, if you need to cloud burst, you might as well go fully multi-cloud since applications that need to scale in this way typically get moved to the cloud anyway

Environment optimization

Optimizing your environments can help you build high-performing applications and systems. With a multi-cloud database, you can combine the best parts of different cloud services or partition your resource needs across various providers. By picking the best pieces of each cloud service, you can:

- **Improve application performance**
- **Reduce latency**
- **Decrease overhead costs**
- **Better support mission-critical functionality**

To optimize your dev environment, you might use a service like DigitalOcean because it is developer-friendly and cost-effective. However, you might choose AWS for product staging because of its high performance and robust features. Perhaps you select AWS for the dev environment because it's easy to spin up and control each database instance. With a multi-cloud setup, you can pick and choose the cloud service capabilities that will let you create the most optimized developer environment to build your applications.

Legal compliance

One of the most critical use cases for a multi-cloud database is legal compliance — the ability to adhere to the data handling and privacy laws of different jurisdictions. If you want to do business in other countries, you must comply with their laws, and every

country has different rules when it comes to data handling. For example, in Europe, data must be kept within the country of origin. You can store the data on-premises or in the cloud, but it must remain within that local country. A multi-cloud database setup allows you to satisfy data requirements more easily.

The legal landscape for data handling constantly changes. For example, GDPR dictates how businesses must handle the personal data of EU citizens. However, the **Schrems II verdict** invalidated the **EU-US Data Protection Shield**, preventing the transfer of data to other countries while still complying with GDPR. To counter the Schrems II verdict, the U.S. and European Commission recently **announced** a commitment to a Trans-Atlantic Data Privacy Framework to foster secure and private trans-Atlantic data flows.

With a multi-cloud database, you can create data policies based on different jurisdictions and enable end-to-end encryption, which many governments require to ensure data protection. You can select specific locations to store data because you have the option to choose databases from different cloud service providers. And those providers have cloud server locations all over the world.

Say you have an application and data in the AWS US-EAST region, but you have some customers in Brazil, and laws in that country require you to have a local copy of their data. Or for example, a hotel booking chain may need copies of data stored in each country they operate in for compliance reasons. A multi-cloud database allows you to keep the data at the locations needed to comply with these data handling requirements and in many cases, may also offer performance benefits as well.

4

Multi-cloud database architecture

Now that we've defined multi-cloud databases let's dive into the specifics of architecting one. We'll cover different providers you could choose for your multi-cloud database architecture, discuss cluster topologies to fit your applications, and review important considerations for security, privacy, compliance, and recovery.

Choosing cloud providers

Cloud database providers offer cloud-managed services where you get access to a database without having to set up physical hardware, install database software, or configure your database — referred to as database-as-a-service (DBaaS).

In addition to these characteristics, DBaaS also provides:

1. **Automatic failover** - If the Primary DB instance fails (for non-serverless DBaaS). For example, AWS Aurora is a serverless architecture, so failure detection and failover don't apply.

2. **Self-healing** - When a DB instance fails in VM hosting, it is automatically replaced with a new one.

3. **Automatic backups** - The DBaaS automatically backs up your DB instances, encrypts them, and stores them for 'retention,' set at a specific number of days.

4. **Software updates** - Automatically apply software updates to the database, underlying OS, and third-party software during a customer-prescribed maintenance window.

DBaaS eliminates most of the hassle for DevOps and DBA work, abstracting away those tasks to a managed service. Using a managed cloud service can improve productivity, ease of use, speed, availability, scalability, and security.

DBaaS providers

You have many options regarding cloud DBaaS providers, but the big three are **Amazon Web Services (AWS)**, **Google Cloud Platform (GCP)**, and **Microsoft Azure**. All three provide a wide range of scalable managed cloud databases.

DBaaS products by database type

	AWS	GCP	Microsoft Azure
Relational	Amazon Aurora Amazon RDS Amazon Redshift	Cloud SQL Cloud Spanner AlloyDB for PostgreSQL Bare Metal Solution for Oracle BigQuery	Azure SQL Database Azure SQL Managed Instance SQL Server on Virtual Machines Azure Database for PostgreSQL Azure Database for MySQL Azure Database for MariaDB
Key-Value	Amazon DynamoDB	Cloud Bigtable	Azure Cache for Redis Azure Cosmos DB
Document	Amazon DocumentDB (with MongoDB compatibility)	Firestore Firebase Realtime Database	Azure Cosmos DB
In-Memory	Amazon ElastiCache Amazon MemoryDB for Redis Amazon DynamoDB	Memorystore	Azure Cache for Redis
Additional NoSQL	Amazon DynamoDB	MongoDB Atlas	Azure Cosmos DB
Graph	Amazon Neptune	JanusGraph Neo4j	Azure Cosmos DB (Gremlin API)
Column-Family	Amazon DynamoDB	Cloud Bigtable	Azure Cosmos DB
Wide Column	Amazon Keyspaces	Cloud Bigtable	Azure Cosmos DB
Time Series	Amazon Timestream	Cloud Bigtable InfluxDB Cloud	InfluxDB
Ledger	Amazon Ledger Database Services (QLDB)	Cloud Spanner	Azure SQL Database
All DB Options	AWS Databases	GCP Databases	Azure Databases

Chart information gathered on 7/14/22

Other DBaaS providers

AWS, GCP, and Azure offer both proprietary databases and open source databases running on their cloud infrastructure. Providers sometimes differ in the open-source databases they support with managed services, so be sure to check for compatibility.

For example, IBM Cloud **offers managed cloud services** for open source databases such as MongoDB, Elasticsearch, etcd, PostgreSQL, Redis, and RabbitMQ. GCP has managed offerings for a number of open-source databases, including Neo4j, MongoDB, Datastax, and Redis Labs.

Looking for other open-source options? In our blog article, "**The Top Five DBaaS Providers for Open Source Databases**," you can learn more about some of our favorite open-source database options, including Alibaba Cloud and Aiven, in addition to the providers we've covered here.

Choosing a cluster topology

Once you've chosen your DBaaS providers, you need to next select a cluster topology for your **multi-cloud database architecture**.

DBaaS architectural models

A DBaaS architecture consolidates database resources according to a consolidation model. The simplest approach is server consolidation through the use of virtualization. Implementing your DBaaS with server virtualization allows you to spawn instances on the fly and on demand and allows different tenants or providers through cloud APIs. Virtualization also allows you to run multiple OS on the same hardware sharing physical resources such as CPU, memory, or storage devices such as other VMs.

Platform consolidation consolidates multiple databases on the same operating system or a cluster. Schema consolidation consolidates further and has the capability to host multiple schemas from different tenants within the same database. A good example of this is orchestrating a set of container nodes with Kubernetes in the form of pods.

What is a cluster topology?

RavenDB provides an apt **definition**:

"The cluster topology is defined by the type and state of each node in the cluster and the relation between them."

Your cluster topology is essentially how you configure your nodes to work together according to the consolidation model, such as virtualized server nodes working together to provide a database service.

While technically, one node can be considered a cluster, you typically see clusters consisting of three or more nodes, with the nodes in odd numbers. That said, you may want to use a single cluster topology for the proof of concept or application testing.

A database cluster topology will have some combination of **master nodes** and, optionally **slave nodes**. Master nodes read and write data, while slave nodes only read data from the master nodes. In this way, the master nodes are the authority on data entering the cluster, while slave nodes act as backups.

A cluster must have at least one master node and can have many working together simultaneously. Slave nodes are not required for all cluster topologies, and their inclusion depends on the purpose of the overall system. Commonly, slave nodes are used for clusters with the primary purpose of highly redundant storage or for data ingress efficiency since they do not have to be up-to-date for new data to be written to the cluster master nodes.

Data replication between nodes can be **synchronous**, meaning it happens together with other nodes, or **asynchronous**, meaning it happens on a periodic schedule or with some other delay. The delay replicating data to a slave node is referred to as slave lag and is an important consideration when architecting a topology for data recovery since data that comes into the cluster during the slave lag time period will not be stored in the slave nodes.

Types of database cluster topologies

Cluster topologies come in many forms. Different topologies work better for different purposes, such as high availability, high redundancy, or load balancing. Each type will utilize a different combination of nodes with different patterns for replicating data between them. For example, you could have:

- One cluster stretched across multiple sites.
- Multiple clusters kept in sync via asynchronous replication.
- A mixture of synchronous and asynchronous replication.

The way that all the master and slave nodes in a cluster relate to each other is what defines the cluster topology. A topology with all master nodes is referred to as a **Master-Master** topology, while one including slave nodes is referred to as a **Master-Slave** topology.

Database clusters may also define **Active** nodes vs. **Passive** nodes. An Active node is always operational, while a Passive node only becomes operational in the cluster when there is a need for the additional resources.

The best cluster topology solution depends primarily on the purpose of the cluster as well as factors like WAN latency, data consistency, and budget. The primary concerns for a database are data redundancy, load balancing, and availability.

Failover/High Availability cluster topologies will have an arrangement that prioritizes disaster recovery and uptime in case of an interruption to one of the active nodes. These clusters usually involve cooperating synchronous master nodes so that if any one node fails, the others are immediately available with a complete copy of the database. These topologies are common for user-facing services like e-commerce sites that want to have services always available.

High-Performance cluster topologies are designed with high throughput in mind rather than availability. Nodes work together to maximize resources spent on the same data requests.

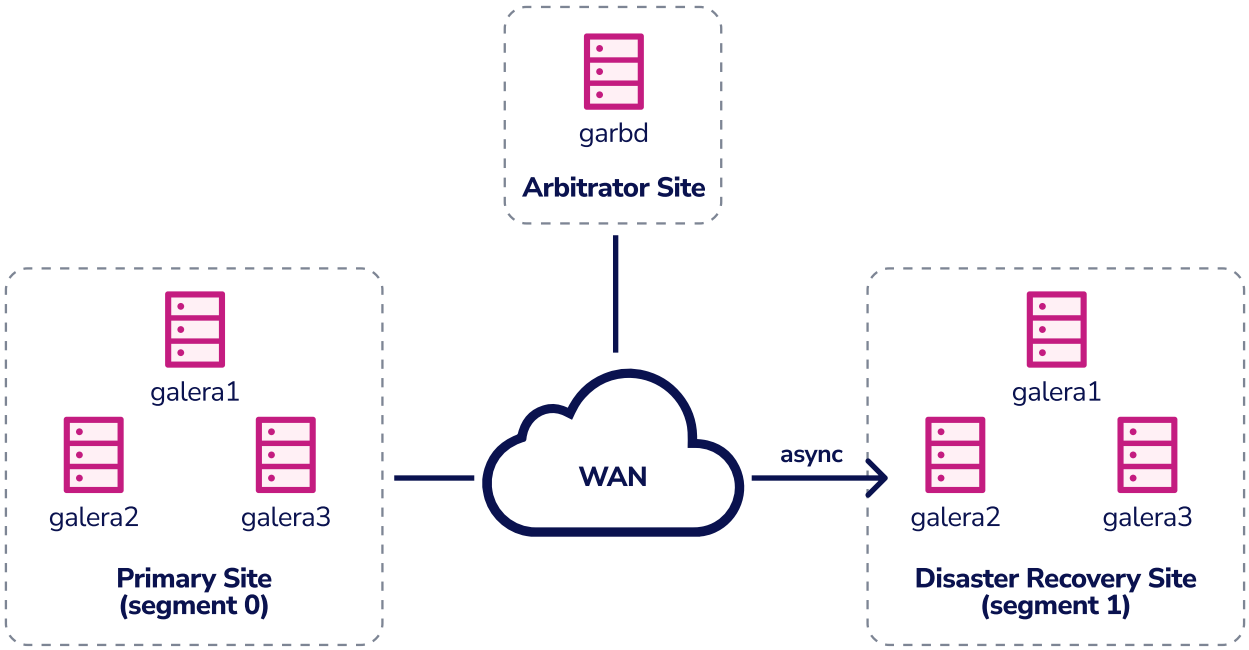
Load-Balancing cluster topologies are designed to maximize efficiency in handling parallel data requests by involving a load balancer that assigns tasks to nodes based on their current load. In this way, nodes are each working on their own assignment, and the cluster can scale up dynamically based on demand.

Example cluster technology: Galera

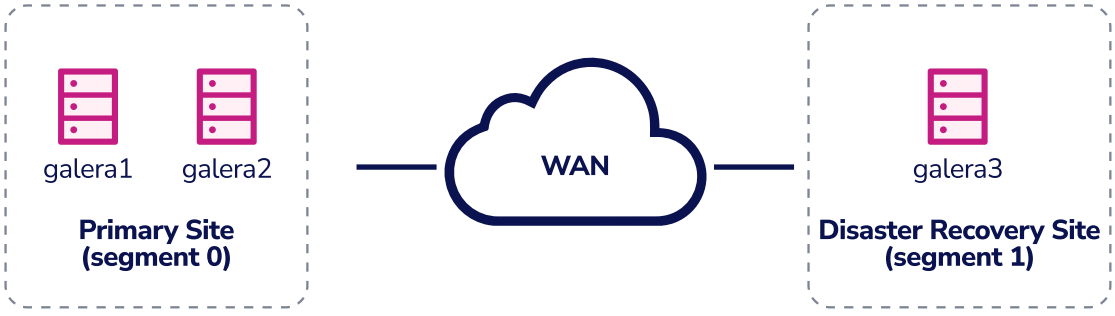
A concrete example of a common Master-Master cluster technology is Galera. Galera runs on MySQL and utilizes synchronous replication. The following section illustrates some possible cluster topologies of a Galera cluster.

One cluster stretched across multiple sites.

You could set up an ***"Active Passive Master-Master Cluster"*** where you have six Galera nodes on two sites, forming a Galera cluster across WAN. You would also have a third site to act as an arbitrator, voting for a quorum and preserving the "primary component" if the primary site is unreachable.



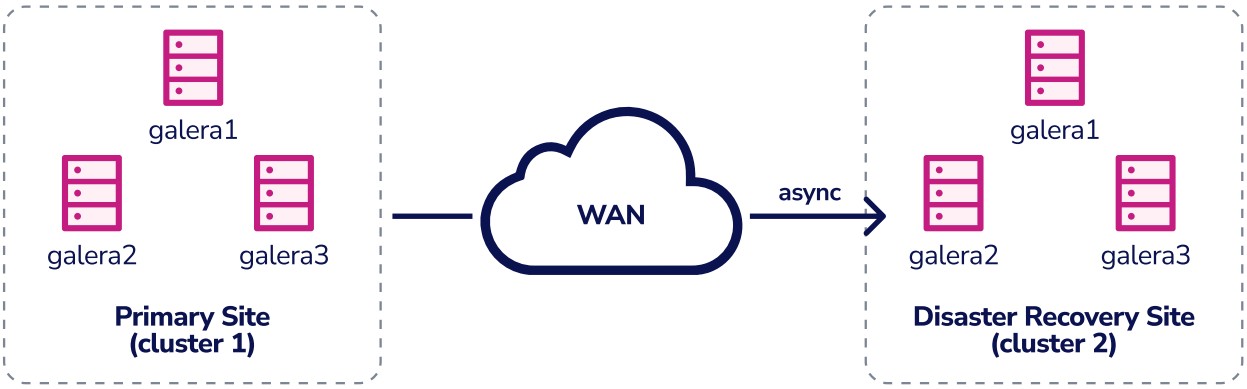
You can also set up an "***Active Passive Master-Master Node***", where two Galera nodes are located in the primary site, and a third node is located in a disaster recovery site. If the primary site goes down, the cluster will fail as it is out of quorum.



Multiple clusters kept in sync via asynchronous replication.

Another cluster topology option is setting up an "***Active Passive Master-Slave Cluster via***

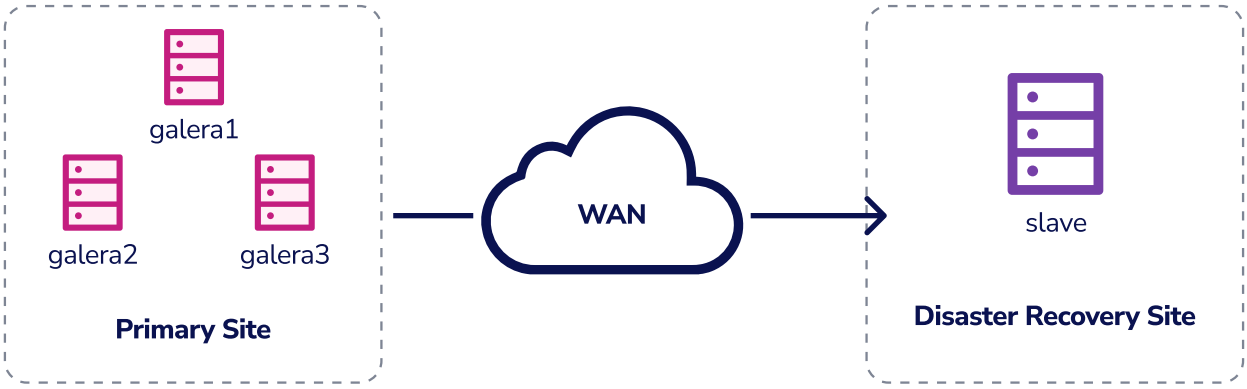
Async Replication." Here, you would have two separate, loosely coupled clusters. This setup makes the primary and DR site independent of each other, loosely connected with asynchronous replication.



One of the Galera nodes in the DR site would be a slave that replicates from one of the Galera nodes (master) in the primary site. You ship updates on the active cluster asynchronously to the backup site. You could choose to have a dedicated slave instance as the replication relay instead of using one of the Galera nodes as a slave.

A mixture of synchronous and asynchronous replication

You could have a cluster topology that leverages synchronous and asynchronous replication. For example, you might have an "***Active Passive Master-Slave Replication Node***" where the synced Galera cluster on the primary site replicates to a single-instance slave in the DR site using asynchronous MySQL replication with GTID.



We discuss all these cluster topologies in detail in this **blog post**, along with their advantages and disadvantages.

Cluster topology considerations

The best cluster topology solution for your use case will depend on your project requirements, your need for data consistency, and your budget.

When choosing a cluster topology, you need to consider several factors, including:

- The type of application you want to run
- Your throughput and latency requirements
- How you configure queries
- Overall complexity
- What is the uptime (a.k.a availability) requirement for the application demanded by the business

Application type

What kind of application do you plan on running? eCommerce, IoT, enterprise? Does your application run on microservices?

You'll need to choose your cluster topology carefully based on the type of application you want to run on a multi-cloud database.

For example, IBM Business Process Manager **supports** a three-cluster topology pattern where the deployment environment functions for Application, Messaging, and Support are divided among three separate clusters.

If your application runs on microservices, you could have three clusters deployed out of regions (in distant areas). In other words, you would have multiregional application deployment. If you run your application on multiple regions, it should continue operating in the event of a region failure. This clustering pattern is helpful if your application has high availability requirements.

Throughput and latency requirements

You have two primary **sources of latency**: geographic distance and hardware. The longer the distance, the bigger the latency. Every hop in the network and router the packet must pass through adds more latency. Functions within the database can also add latency, and those functions and the added latency differ from database to database. For example, this **blog post** covers the effects of high latency in high-availability MySQL and MariaDB solutions.

Load balancing is one of the best ways to reduce database read latency (it won't reduce the write latency). You need to make sure your DBaaS providers have excellent load-balancing capabilities. Read this **blog post** to learn more about load balancing and databases.

When you build a distributed system, you often need to make tradeoffs between latency and resiliency — making choosing the right cluster topology all the more critical. Some topologies optimize for latency, while others optimize for stability and other factors. Consider which factors are most important for your application.

Query configuration

Your queries are affected by your choice of cluster topology, and the way you **configure your queries** will impact database latency and performance. Users want queries to complete quickly, but they may accept slower queries if it improves the stability of the query execution time. You can improve the performance of your database and reduce latency by performing **query tuning**. We explain MySQL query tuning in detail in this **webinar**.

Complex cluster topologies

You may want to deploy a complex cluster topology but find the manual process can be daunting. Rather than configuring the topology piecemeal, the easiest way is to **employ a tool** that lets you deploy clusters and load balancers with the click of a button. Say you want to run your database on GCP and AWS. With a click-and-launch tool, you'd only have to go through a few quick steps. The tool would spin up instances and configure the databases with the providers for you.

You wouldn't have to spend hours writing code to deploy your databases or launch load balancers on multiple cloud providers. You can deploy all your databases in any environment and manage them from one pane of glass.

Other DBaaS platform considerations

Cluster topology is one of the most critical aspects of a multi-cloud database architecture. However, you have other important aspects to consider.

Data security and privacy

When choosing a DBaaS provider, you need to ensure they can keep the data secure — whether in transit or in storage. The provider must follow modern security protocols and best practices. They must encrypt the data with the latest TLS version if it is in transit. Data at rest requires block-level or full file system **disk encryption** and Transparent Data Encryption (TDE). Also, some security regulations require encryption, like PCI-DSS.

Compliance offerings

You should make sure each provider offers the specific security and privacy compliance standards you need:

- **GCP Compliance Offerings**
- **Azure Compliance**
- **AWS Compliance Programs**

If you plan on building a healthcare application, you'll need to make sure each DBaaS provider offers HIPAA compliance. If you want to develop an eCommerce app that serves customers in the European Economic Area (EEA), you'd need all your cloud database providers to comply with GDPR. You might also need to demonstrate SOC 2 or PCI compliance. GCP, Azure, and AWS offer a comprehensive selection of compliance offerings.

Database backups and disaster recovery

You need providers who can effectively back up your databases and provide disaster recovery capabilities. Review how each provider replicates your data — what replication methods are they using? You have many **choices for data replication**, including transactional, snapshot, key-based, merge, and peer-to-peer.

Does the provider use synchronous or asynchronous replication? In some cases, you may need synchronous replication, where the data is written in the primary database and the replica simultaneously. Or you might need asynchronous replication, where the data is written in the replica after it is written to the original database.

The type of replication you'll need goes beyond the scope of this guide. However, we've published several blog posts about this topic. See the "further reading and references" section at the end of this document.

You also have several disaster recovery strategies, such as:

- **Database forking** - Spinning a new DB cluster from a backup. You create a copy of a database, making changes to that copy without impacting the original database
- **Point-in-time Recovery (PITR)** - You perform a full backup of the database, manually or automatically, so that you can restore the database up to a specific point in time.

5

Deploying and maintaining a multi-cloud database

Now that you know the ideal use cases for a multi-cloud database and details on how to architect one, let's focus on database deployment and maintenance.

Deploying your multi-cloud database

Before you can deploy your multi-cloud database, you need to set up the environment for each cloud provider you plan on using.

First, set up the environment for your primary cloud provider. After that, you'll need to take additional steps to prepare environments for your subsequent cloud providers.

The basic steps for multi-cloud database preparation and deployment are:

- **Configure cloud credentials and SSH** – You need to set up your cloud credentials and SSH access information. Then you define the database version, data directory, port, and admin credentials.

- **Allow traffic** – Make sure you allow database and SSH traffic.

- **Deploy a cluster in the cloud** – How you deploy a cluster depends on the cloud platform. The cloud provider should offer tools to help you, like SDKs, quick start guides, and web-based UIs.

- **Deploy a VPN** – If the database is for real-world production, you should create a VPN (virtual private network) that connects instances between your cloud providers, e.g., AWS instances with GCP instances. A VPN allows you to treat all instances as local, even though they are in disparate clouds.

- **Enable necessary ports** – Make sure you've only enabled the ports needed for deployment. Those ports may include SSH (22), xtrabackup (9999), and database (3306).

- **Choose synchronous or asynchronous replication** – You should choose whether to have synchronous replication, asynchronous replication, or a combination of both. Pick the one that best suits your use case.

- **Create one or more slaves** – Depending on your use case, you'll need to create one or more slaves. Start by creating a slave on your secondary cloud provider. Create additional slaves on your other cloud providers, if required.

- **Launch load balancers** – Make sure you launch load balancers which help maintain the high performance of your cloud database.

- **Deploy alerting and trending tools** – You should set up tools that monitor your database, sending alerts when problems arise. These tools should also allow you to access

reports that show current trends. For example, you could deploy **Grafana** for trending reports and **Nagios** or **OpsGenie** to receive alerts.

Maintaining your multi-cloud database

Once you've deployed your database, you need to ensure it performs well with 99.999% to 100% uptime. Effectively maintaining your database involves three key things:

- **Monitoring and alerts**
- **Investigating database issues and errors**
- **Backups and disaster recovery**

Database monitoring and alerts

One of the best ways to keep your database in working order is to implement monitoring tools that automatically send alerts that inform when you should perform routine maintenance or when issues arise. These blog posts go into detail about database monitoring and alerts:

- **Best Practices for Enabling Database Alarms & Notifications**
- **PostgreSQL Database Monitoring: Tips for What to Monitor**
- **What to Monitor in MySQL 8.0**
- **How to Monitor Your Databases with ClusterControl and PagerDuty**

Most monitoring tools allow you to set up alerts for various **database metrics**.

Common database alerts include:

- **Database service availability** – You'll get notified if the database process or daemon service stops running.
- **Network availability** – Lets you know if the database server has been disconnected from the network and when it goes back online.
- **Cluster health** – Get a notification if a database cluster has trouble processing read or write requests.
- **Database health** – Monitoring database health involves many factors. You would get

alerts for multiple things, such as replication lag, lock contention, data inconsistency, and duplicate indexes.

- **Hardware health** – You would receive alarms about factors impacting the hardware the database relies on, such as data storage (e.g., SSD, RAID), memory, and CPU.

- **Cluster or node recovery state** – Many enterprises set up databases in a cluster or node recovery state configuration. You would receive alerts if the configuration will soon fail.

You would monitor the above things for all types of databases in the cloud. However, you can also monitor and receive alerts for operating system and database-specific metrics.

Operating system and database-specific alerts include:

You could monitor these specific operating system and database metrics for a **PostgreSQL** or **MySQL 8.0** database:

- **CPU Usage** – Get notified when CPU usage nears or exceeds the limit.

- **RAM or SWAP usage** – Receive an alert when usage becomes high, which means you should check your database configuration.

- **Disk usage** – Get an alert when the monitoring tool detects an abnormal increase in disk space usage or excessive disk access consumption. You must always have space for new data, temporary files, snapshots, or backups.

- **Queries** – Get notified when the database has too many queries running simultaneously or queries become too long. You could also monitor the amount of SELECT, INSERT, UPDATE, or DELETEs on each node.

- **Active sessions** – Find out when the number of active sessions nears or exceeds the limit. If the number of active sessions gets too high, you would need to find out if the database has a problem or if you only need to increase the max_connections value.

- **Database locks** – Get notified if the database has a high number of queries waiting for other queries to complete. An update of the database or a database issue could lead to an increased number of locks.

- **Replication** – Monitoring replication is crucial to ensuring a high availability environment, and it typically involves looking at the lag and the replication state. If you have a MySQL database, you should monitor and receive alerts for SLAVE STATUS along with these parameters:

    - SLAVE_RUNNING
    - SLAVE_IO_Running
    - SLAVE_SQL_RUNNING
    - LAST_SQL_ERRNO
    - SECONDS_BEHIND_MASTER

- **Database logs** – You should monitor your database logs, looking for errors like FATAL and deadlock. Also, keep an eye out for authentication problems and long-running queries. The database logs provide a list of errors and helpful information to help fix them.

- You can monitor your database and set up automatic alerts with database monitoring tools, such as:

- Nagios
- PagerDuty
- Percona Monitoring and Management (PMM)
- SolarWinds Database Performance Monitor (formerly VividCortex)
- Zabbix

Investigating database issues and errors

Monitoring and alert tools will help you know when you have problems with your database. But you must thoroughly investigate each issue before you can find the best solution.

Unfortunately, databases can have many issues, too many to go into detail here. Some problems may cause database errors, while others lead to database failure. Either way, you need to figure out the cause.

Start with the logs

Logs can point you in the right direction when investigating an error or problem with your database. For example, MySQL error logs can tell you if you have configuration, permission, or memory errors. If your database has crashed, you could look at InnoDB logging files to see if the crash is due to an InnoDB bug.

If your MySQL database keeps running slow, you could start your investigation by checking MySQL's slow query logs. These logs provide a list of queries the system has identified as slow based on a given set of values.

Check the host system's health

Use database monitoring and reporting tools to analyze the health of the host system. Your database may experience problems, like slowness and errors, because the host system has issues. For example, the PostgreSQL database host system may suffer from overload or memory usage increases where the overflow gets sent to SWAP.

Monitoring tools will help you catch problems before they lead to severe problems or failures. We've published a number of blog posts that could help you investigate and fix issues with your database:

- **Understanding the MySQL Error Log**
- **How to Fix a Lock Wait Timeout Exceeded Error in MySQL**
- **How to Identify MySQL Performance Issues with Slow Queries**
- **6 Common Failure Scenarios for MySQL & MariaDB, and How to Fix Them**
- **Understanding Deadlocks in MySQL & PostgreSQL**
- **PostgreSQL Running Slow? Tips & Tricks to Get to the Source**

Backups and disaster recovery

You can do a lot to prevent your database from experiencing errors or failing. However, database failure is always a possibility. You need to make sure you maintain proper backups of your database in case a disaster does occur.

A quality database will allow you to implement disaster recovery options. For example, PostgreSQL has **streaming replication**, where you can promote a failover site or a Hot Standby to the master if the main database goes down. We explain PostgreSQL Streaming Replication in this **blog post**.

Another disaster recovery option in PostgreSQL is Point in Time Recovery (PITR). With PITR, you can bring back a copy of the database based on a specific point in time. This method is slower than a Hot Standby, but you can recover a database snapshot before a significant event occurs, like a deleted table or data corruption.

You should frequently create backup copies of your database and store them in multiple locations. This blog post provides tips for storing PostgreSQL backups on the Google Cloud Platform (GCP).

6

Checklist of best practices for multi-cloud database operations

We've compiled a list of best practices you should follow to ensure you have a high-availability and well-performing database.

Security

- [ ] Ensure the database encrypts data at rest and in transit.
- [ ] Use data services designed for multi-cloud environments.
- [ ] Enable role-based access control and authentication (**blog post**).
- [ ] Regularly audit system activities.
- [ ] Limit database access to known network devices.
- [ ] Implement firewalls and VPNs.

Scaling

- [ ] Choose the best scaling option — vertical or horizontal — for your use case.
- [ ] Make sure your environment meets the requirements for scaling, which includes data replication, load balancing, and read/write split. (**blog post**).
- [ ] Automate database scaling processes where possible.

Performance

- [ ] Reduce data latency by connecting cloud networks.
- [ ] Implement database monitoring and alert tools.
- [ ] Make sure you have load balancers enabled.
- [ ] Automate database health checks (**blog post**).

Governance

- [ ] Ensure you have a practical governance framework in place.
- [ ] Make sure you comply with data privacy laws, like GDPR and HIPAA. (**blog post**).

Backups and Disasters

- [ ] Make sure you have database backups stored in multiple locations.
- [ ] Use the proper replication – e.g., synchronous, asynchronous, or a mix of both.
- [ ] Set up automatic database backup verification (**blog post**).

Miscellaneous

Consider taking advantage of:

- [ ] Automated database deployment tools.
- [ ] Cloud managed services and managed database services.

7

Further reading and references

[Types of Replication in SQL Server on Linux]

[Database Backups in the Cloud for Disaster Recovery]

[Webinar Replay: Disaster Recovery Planning for MySQL & MariaDB]

[How to Setup Asynchronous Replication Between MariaDB Galera Clusters]

[When and How to Use Asynchronous Commit Mode for MS SQL Server Always On]

[Exploring Synchronous Commit Mode for SQL Server Always On]

[MySQL Replication Best Practices]

[MongoDB Replication Best Practices]

[PostgreSQL Replication Best Practices – Part 1]

[PostgreSQL Replication Best Practices – Part 2]