Google Cloud

Spanner geo-partitioning

Authors

Szabolcs Rozsnyai

Moe Barouni

Nitin Sagar

Rachel Wang

Panagiotis Voulgaris

Dave Weissman

August 2024

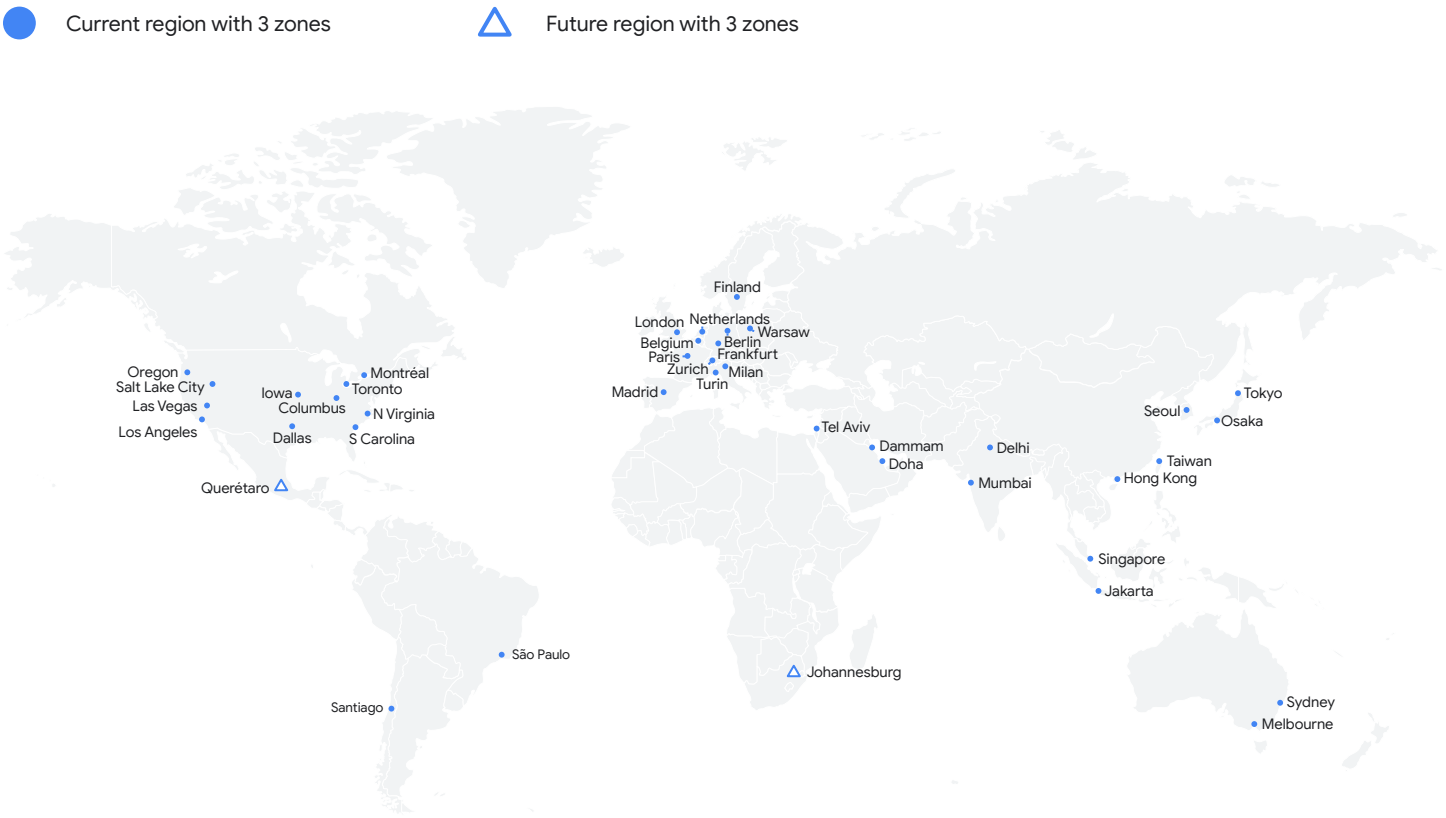
What is Spanner?

Spanner is Google's always-on, globally consistent, and virtually unlimited scale database. As the database that powers Google's billion user products such as Gmail, YouTube, and Google Photos, Spanner is known for its reliability, scale, and strong consistency, making it the trusted choice for companies worldwide, including those in financial services, retail, gaming, technology, and media. With features such as automatic sharding, zero planned downtime, seamless replication, and guaranteed strong consistency, Spanner handles the complexities of managing and operating a mission-critical database for both non-relational and relational workloads at scale.

How does Spanner replication work today?

Users have been able to deploy Spanner instances with pre-defined regions and replication topologies that are referred to as "base instance" configurations. These configurations can be extended with read-only replicas around the globe to provide additional low latency stale reads.

Spanner is available in most Google Cloud data centers globally, but adhered to predefined base configuration & replication topologies.



Two base configuration types

Base Configurations come in three main flavors:

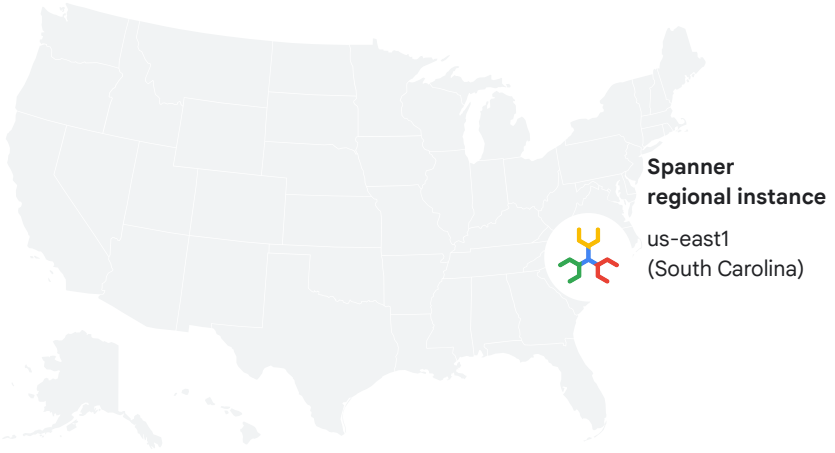
- Regional configurations: all resources reside within a single Google cloud region
- Multi-region configurations: the resources span more than two regions
- Dual-region configurations: all resources span two regions and reside within a single country

Regional configuration

Regional Spanner instances are deployed within one designated single GCP region such as us-east1 (South-Carolina). A single region deployment provides three read/write replicas per zone of a region leading to 99.99% availability.

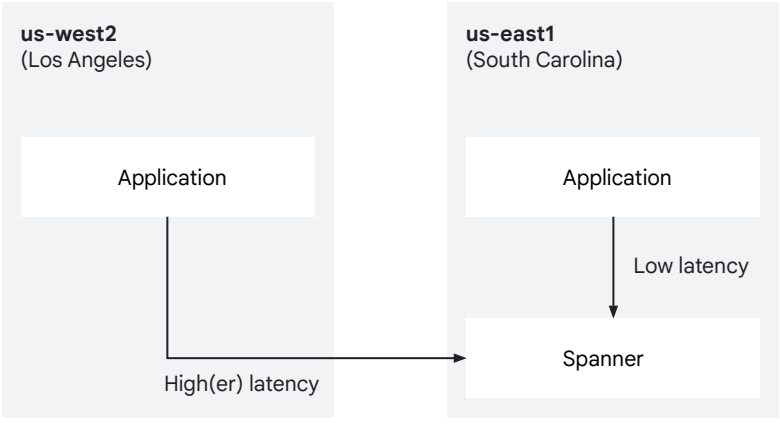
Geographic placement close to a Spanner instance is important to achieve low latency connections. The closer a client is to the Spanner instance the lower the latency is.

Regional instance



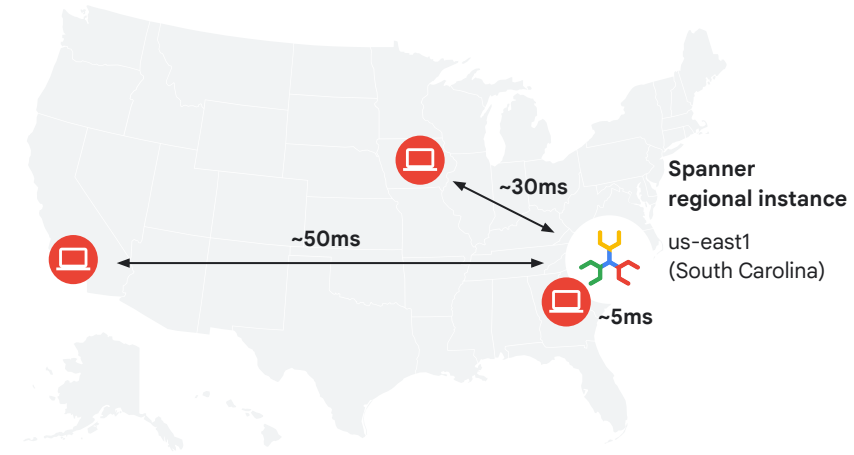
As such the optimal write and strong read latency is achieved when the client applications are co-located in the same region with the Spanner instance.

In the example to the right, clients from the Los Angeles area, experience a network latency round-trip time (RTT) of approximately 50ms, whereas clients located closer to the Spanner instance region such as for instance in Iowa get approximately 30ms or less and clients placed in the same region is usually 5ms or less.



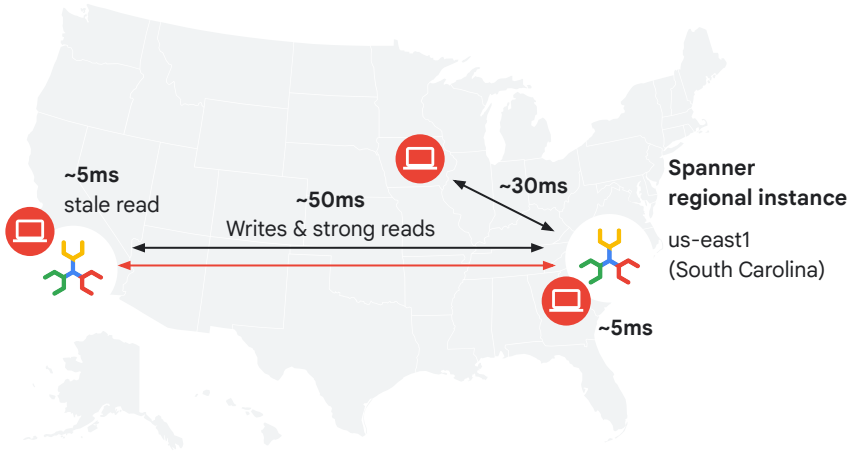
Certain use cases call to cater for geographically dispersed audiences and as such the need to optimize for latency. For this purpose, architects have the option to attach optional read-only regions to regional deployments. These allow spanner instances to serve low latency, but stale, reads from other geographic regions.

Regional instance



In the example to the right, clients on the west coast can read stale data with very low latency. While data is replicated to read only replicas extremely quickly, if there are no updates a read only replica will not know that it's up to date until a lease interval. Because of this, we recommend users set a max_staleness of 15 seconds if they want to ensure nearly all reads will be served without talking to another replica.. On the other hand, some strong reads (depending on how up to date the read only replica is) and all writes will require some communication with the leader region, thus removing the latency benefits of the read only replica.

Regional instance



Multi-region configuration

Multi-region Spanner instances are deployed across multiple designated GCP regions with usually:

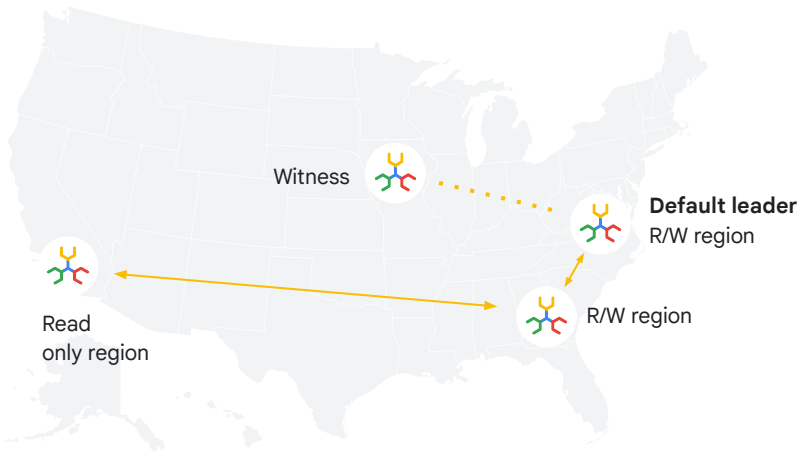
- Two read/writes regions (of which one is designated the leader region)
- A witness region
- And either standard or additional optional read-only regions

A multi-region instance has at least 5 replicas of the database distributed across 3 or more regions providing 99.999% of availability.

The example to the right illustrates the nam3 multi-region configuration. The read-write regions are spread across us-east4 (Northern Virginia) and us-east1 (South Carolina) with the witness being located in us-central1 (Iowa) as a regional protection. In this example an optional read-only replica was provisioned in us-west2 (Los Angeles) to optimize latency for stale reads on the west coast.

The default leader region is designated in the base configuration, but can be swapped with other read/write regions. The witness region itself does not store a full copy of the database, but is an important component as it participates in voting to reach quorums. This is relevant to protect against regional outages.

Multi-region instance - nam3



Dual-region configuration

Dual-region instances are deployed across two regions and replicate data cross zones and regions in a single country. This configuration type:

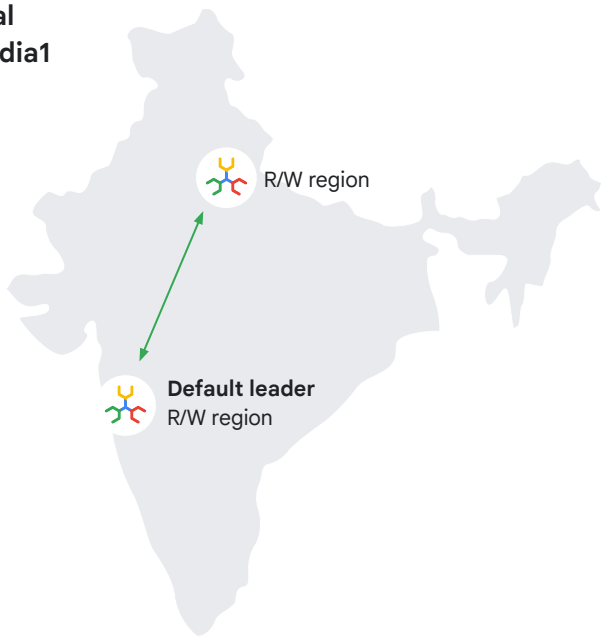
- Serves reads from two regions in a single country
- Provide higher availability (99.999%) and SLAs than regional configurations
- Can help to meet certain data residency requirements

A dual region configuration has six replicas across two regions (three in each region). In contrast to a multi-region configuration, in each region there are two read-write replicas and one witness replica. A minimum of two replicas in each region is required to form a quorum and as such this type of configuration provides up to 99.999% availability.

This configuration type can help to meet certain data residency requirements in the countries listed under available dual region configurations.

The example to the right shows the dual region configuration India. In this configuration the read-write regions are located in asia-south1 (Mumbai) and asia-south2 (Delhi). The default leader region is designated in Mumbai, but can be swapped by configuration with the other read/write region.

Dual-regional instance - india1



What is geo-partitioning and how can it help?

Geo-partitioning allows Spanner customers to define database instances that can be partitioned by configuration and geographic regions. This marks a shift from the previous approach, where each instance adhered to a single predefined "base instance" configuration. By strategically placing leader regions closer to primary user bases and distributing data, geo-partitioning significantly reduces latency of writes and reads and enhances user experience. It also optimizes costs for businesses with asymmetrical global workloads where the data can be partitioned by aligning storage and processing capacity to regional demands, preventing over-provisioning. Moreover, geo-partitioning aids in complying with evolving data regulations by centralizing management and maintaining robust access controls, monitoring, and data provenance while minimizing legal risks.

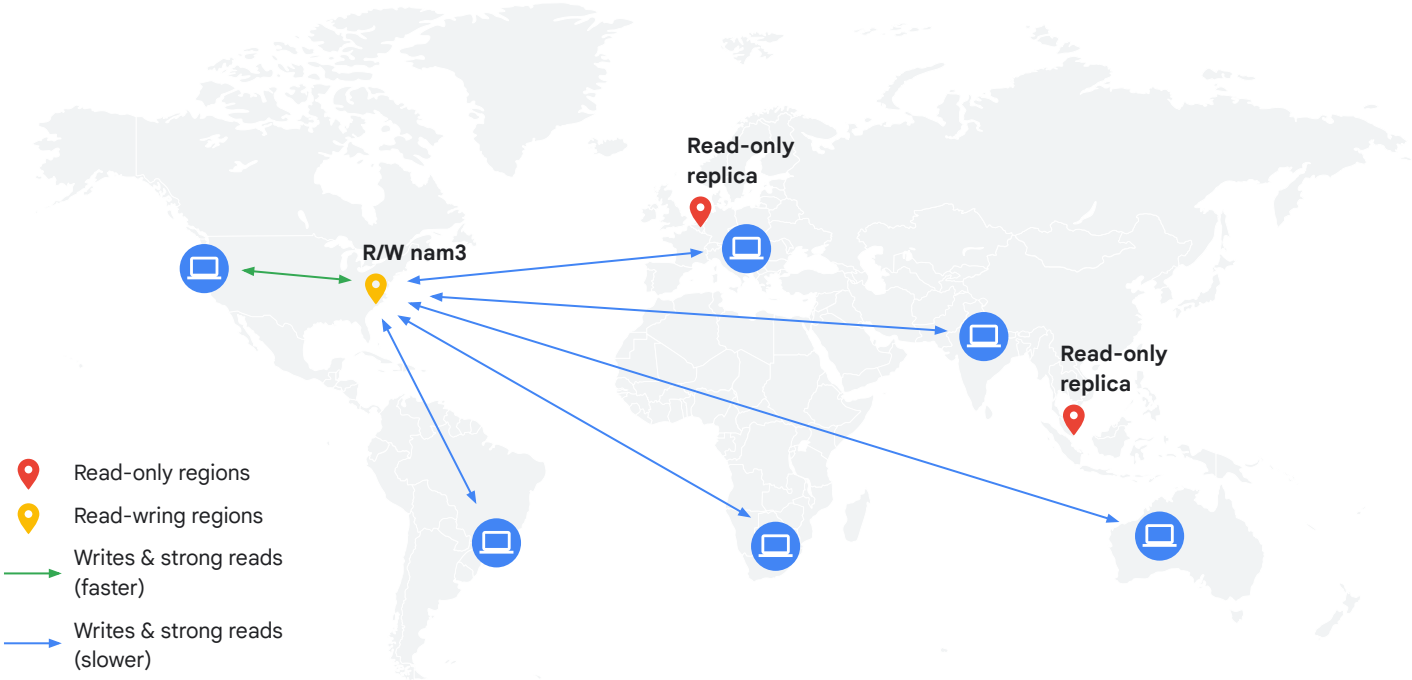
As such Spanner's geo-partition can help to:

1. Reduce write and strong read latency
Writes and strong reads often require round trips to the leader region potentially increasing latency for geographically distant clients. Geo-partitioning helps enterprises to mitigate the latency delays by distributing the data and locating the leader regions closer to the primary user bases. This strategy can significantly improve the performance and the user experience.

2. Optimize costs for asymmetrical global workloads
Businesses with varying workload shapes and customer bases across different geographic regions need to optimize costs effectively. For instance, if one region (e.g. the EU) has a larger traffic or more data than another (e.g., the United States), uniformly scaling Spanner instances across both regions might not be cost efficient. Instead, organizations can adopt cost optimization strategies that align the processing and storage capacity with the workload demands, allowing them to avoid over-provisioning and thus reduce unnecessary expenses.

Reduce write and read latency

Today, architects can reduce only stale read latency by adding read-only replicas in regions closer to clients, but writes and strong reads still require round trips to the leader region. The impact of latency is exceedingly amplified in large geographically distributed systems.



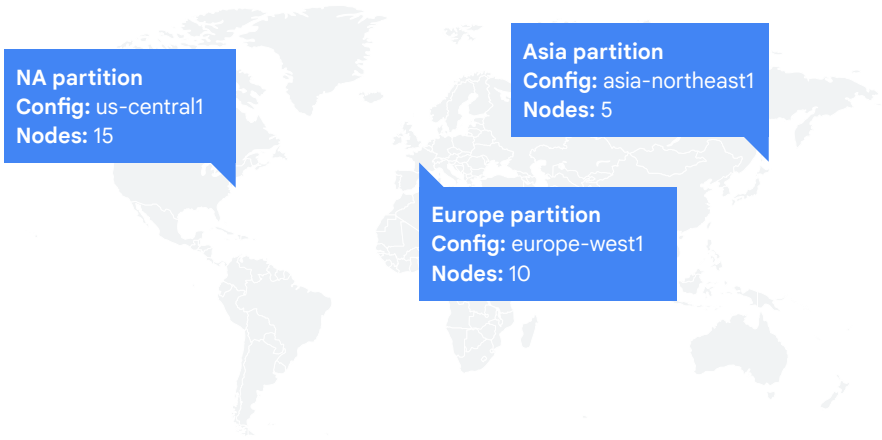
The above illustration has a multi-region nam3 instance with its default leader in Northern Virginia along with multiple read-only replicas spread across the globe. This configuration would only speed up latency for stale reads globally, but any strong read or write from e.g. Europe or Asia would need to go to the default leader in the US resulting in a high round trip latency.

For use cases that cater for a global audience, high write and strong read latencies might not be acceptable.

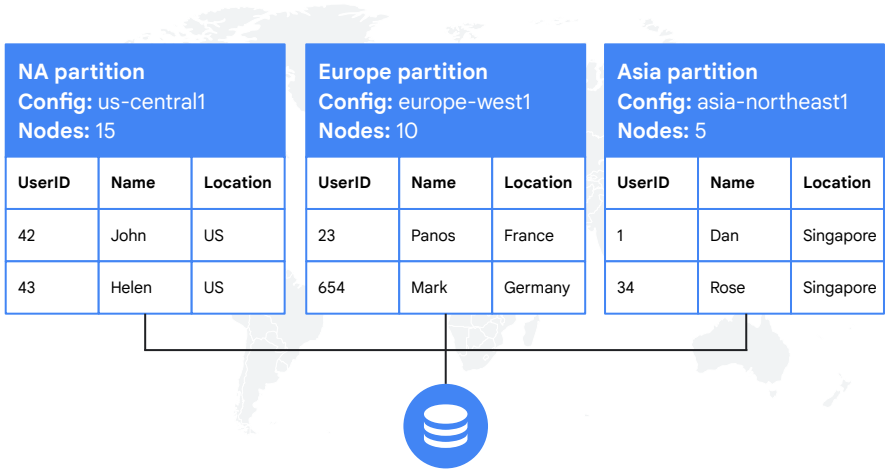
There are two options to address this problem:

1) Multiple independent instances: Create multiple independent instances - each in the desired geo-location. This means also creating separate databases within these instances, each with their own resources and data. In addition to the complexity of managing multiple instances, this approach would require applications to manage routing requests to instances hosting the relevant data, would remove potentially useful functionality like cross-partition transactions, and would require complex migration pipelines to move data among geographic partitions.

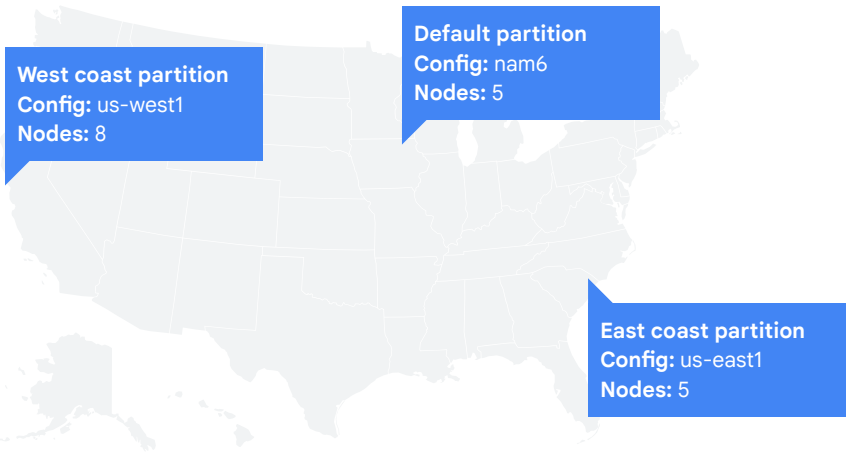
2) Geo-partitioned instance: Create a single database that can span multiple locations (partitions), allowing you to distribute data for performance. In contrast to multiple independent instances, using geo-partitioned instances, one can configure one single logical Spanner deployment with four instance partitions where the data can be spread globally close to its consumers, speeding up not only stale, but also strong reads and writes in their respective geographies.



The data in a table is partitioned into instance partitions close to the clients such as data that is consumed in North America is stored in the NA partition and data that is processed in Europe is stored in the Europe partition. At partition-level, Spanner provides partial at-rest guarantees, because split ranges, global indexes and foreign keys might be located outside the local partition.



Geo-partitioning Spanner instances, for the purpose of optimizing latency, are also useful for use cases within a single continent. As an example, partitions could be located on the east and west coasts of the United States. This deployment configuration can be especially useful if the user audiences on both east & west coast need to be served with low latency. By deploying independent Spanner instances a trade-off would need to be made in terms of latency.

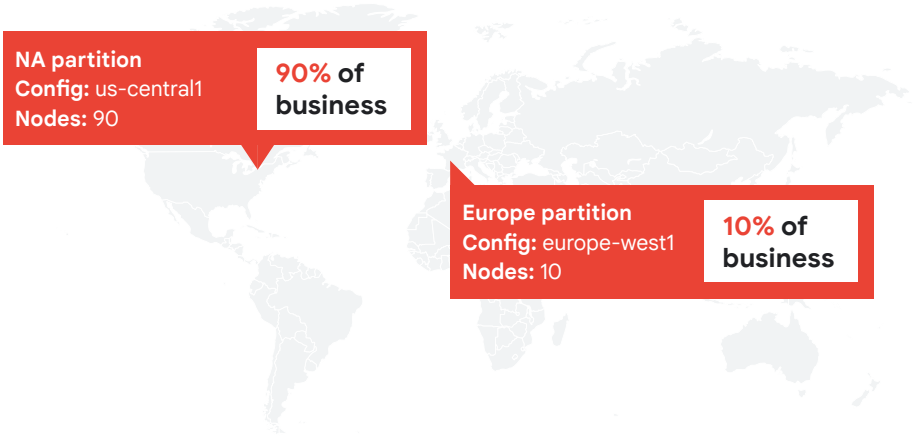


Optimize costs for asymmetrical global workloads

Geo-partitioning enables customers to set up multiple instance partitions, each with its own configuration (i.e. regional or multi-region) and with their own node count (note that utilizing this requires the data to be partionable, for example based on entity location).

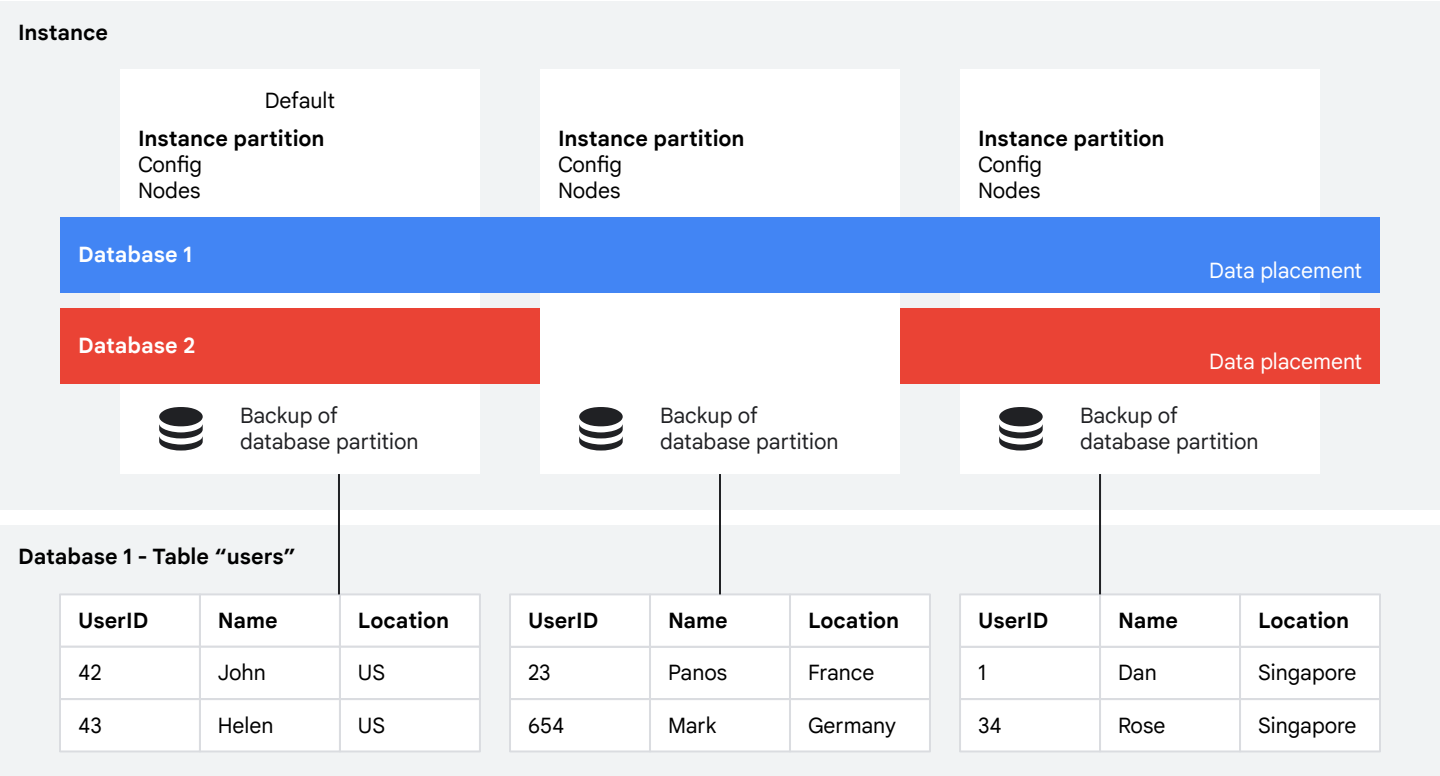
As a consequence, users can not only select cost effective configurations tailored to their current needs, but also independently scale each instance partition to accommodate business growth.

For example, a retailer that has 90% of its business in the United States and 10% of its business in Europe could set up an instance with 90 nodes of us-central1 and 10 nodes of europe-west1.



How does it work?

Core concepts



In Spanner's geo-partitioned architecture, an instance serves as a fundamental container holding one or more instance partitions. Within these partitions, one is designated as the default partition.

An **instance partition** maps directly to a Spanner base configuration. This configuration may include optional extensions, such as read-only replicas, to improve stale read latency. The partition is allocated processing power in the form of nodes or processing units that also drives the addressable storage space.

The **default partition,** which acts as the primary segment within the instance, stores database objects like tables and views by default and manages split ranges and other data distribution specifics. Spanner divides data into "splits", with each split holding a range of contiguous rows. Split boundaries – the start and end keys of the split range – will be stored in the default partition.

A **geo-partitioned** database has a collection of one or more database placements, each associated with a specific instance partition (which may include the default partition). This relationship allows the database to distribute its components efficiently across partitions, leveraging the processing power and configurations that each partition offers. Different placements can map to the same partition. For example, in the graph above, France and Germany both map to eur-partition.

Within the database, placement tables are organized in a way that the placement of rows is determined by a special attribute called PLACEMENT_KEY. This attribute ensures that rows, including interleaved child rows, are placed in their appropriate partitions, supporting efficient data organization and retrieval.

A **database** can also have **global tables** that are in the default partition. In such cases no PLACEMENT_KEY is defined.

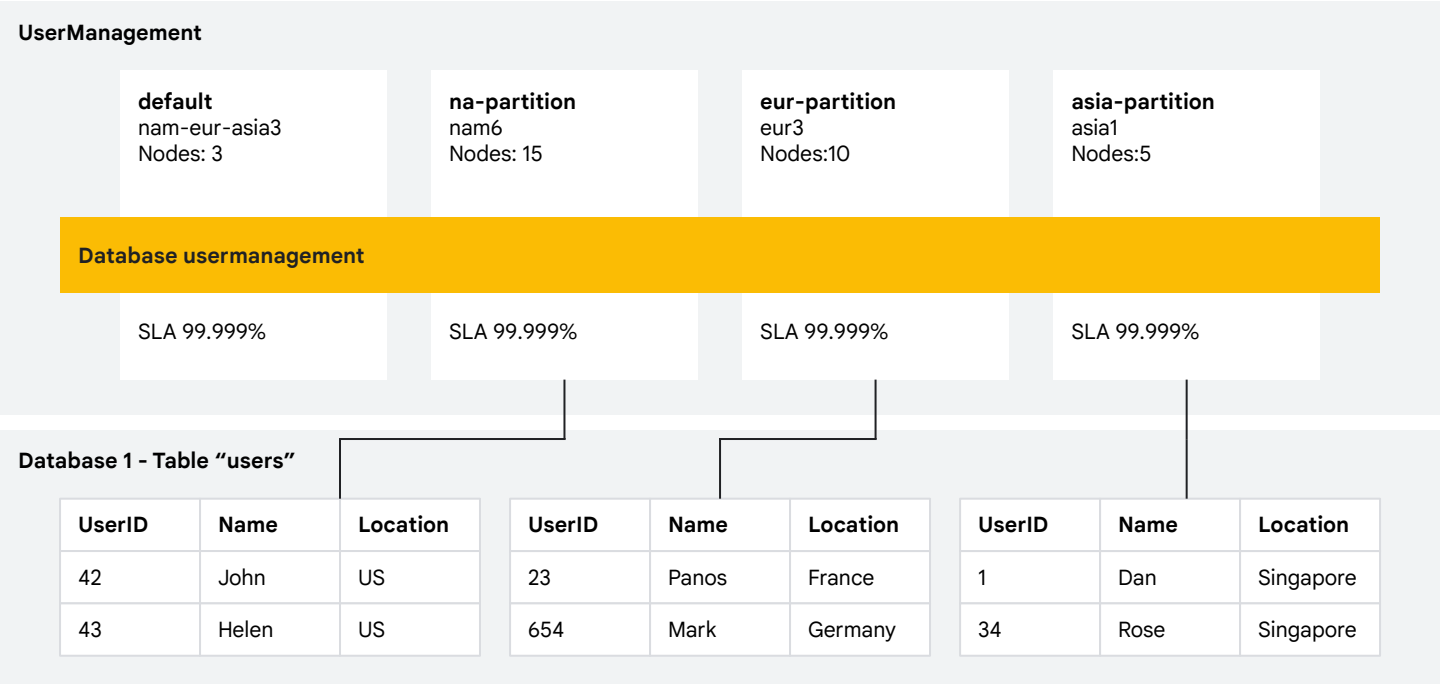
Backups are initiated at the instance level on a per-database basis. However, backups will be partitioned and stored in the appropriate instance partitions (Disclaimer: not yet supported refer to section Preview Limitations, GA and beyond).

Instance	• A container that holds one or more instance partitions • One instance partition is the designated default partition
Instance partition	• Maps to a Spanner base configuration with optional extensions such as read-only replicas • Assigned processing power (i.e. nodes or processing units)
Default partition	• Stores any data that is in global tables (not placement tables or their interleaved tables) • Stores split ranges and other split information
Geo-partitioned Database	• A database that has one or more placements defined in the schema (i.e. CREATE PLACEMENT). Each of which is associated with exactly one instance partition
Placement table	• Placement of a row is determined by a PLACEMENT_KEY attribute (incl. Interleaved child rows)
Global table	• A table that doesn't have PLACEMENT_KEY column and is not an interleave of a placement table

Geo-partitioning example

Example 1
Placement as value

In this example we are going to create a multi-partitioned Spanner instance with one database that manages a user table. Rows in that table will be spread across US, Europe and Asia to facilitate low latency strong reads and writes and confine user data in the respective regions. The default instance partition will be a global configuration with read/write regions in NA and read replicas in Europe & Asia. It will act as the primary segment within the instance to manage split ranges and other data distribution specifics.



UserID	Name	Location
42	John	US
43	Helen	US

UserID	Name	Location
23	Panos	France
654	Mark	Germany

UserID	Name	Location
1	Dan	Singapore
34	Rose	Singapore

Create infrastructure

Create instance container with default instance partition:

```
gcloud spanner instances \
   create user-instance \
   --config=nam-eur-asia3 \
   --nodes=3 \
   --description=user-instance
```

Create North American instance partition

```
gcloud beta spanner instance-partitions \
   create na-partition \
   --instance=user-instance \
   --config=regional-us-central1 \
   --nodes=15 \
   --description=na-partition
```

Create European instance partition

```
gcloud beta spanner instance-partitions \
   create eur-partition \
   --instance=user-instance \
   --config=regional-europe-west1 \
   --nodes=10 \
   --description=eur-partition
```

Create Asian instance partition

```
gcloud beta spanner instance-partitions \
   create asia-partition \
   --instance=user-instance \
   --config=regional-asia-northeast1 \
   --nodes=5  \
   --description=asia-partition
```

The Spanner infrastructure that is provisioned for this instance:

The Spanner infrastructure that is provisioned for this instance:

Instance partition: Default
Configuration: nam-eur-asia3
Nodes: 3

Read/write regions

- **Iowa:** us-central1
  South Carolina: us-east1

Witness

- **Oklahoma:** us-central2

Read/write regions

- **Belgium:** europe-west1
- **Netherlands:** europe-west4
- **Taiwan:** asia-east1

Instance partition: NA partition (na-partition)
Configuration: us-central1
Nodes: 15

Instance partition: Europe partition (eur-partition)
Configuration: europe-west1
Nodes: 10

Instance partition: Asia partition (asia-partition)
Configuration: asia-northeast1
Nodes: 5

Create database, schema and placement policies

Database artifacts such as creating a table, defining placement policies or inserting/updating records can be done either through APIs, gcloud or the Spanner UI.

Create database

```
gcloud spanner databases \
   create usermanagement \
   --instance=user-instance
```

Opt in to geo-partitioning preview

```
gcloud spanner databases \
  ddl update usermanagement \
    --ddl="ALTER DATABASE db SET OPTIONS
(opt_in_dataplacement_preview = true)" \
    --instance=user-instance
```

Table schema

```
gcloud beta spanner databases \
    ddl update usermanagement \
    --ddl="CREATE TABLE Users (UserId INT64 NOT NULL, Name
STRING(MAX) NOT NULL, Location STRING(MAX) NOT NULL PLACEMENT
KEY) PRIMARY KEY(UserId)" \
    --instance=user-instance
```

```
Users (
  UserId INT64 NOT NULL,
  Name STRING(MAX) NOT NULL,
  Location STRING(MAX) NOT NULL PLACEMENT KEY
) PRIMARY KEY(UserId);
```

Define and create placement policy

```
gcloud beta spanner databases \
  ddl update usermanagement \
    --ddl="CREATE PLACEMENT US OPTIONS
(instance_partition='na-partition')" \
  --instance=user-instance
```

Using similar DDL commands, following data placement policies will be defined

```
CREATE PLACEMENT US OPTIONS
(instance_partition='na-partition');

CREATE PLACEMENT Germany OPTIONS
(instance_partition='eur-partition');

CREATE PLACEMENT France OPTIONS
(instance_partition='eur-partition');

CREATE PLACEMENT Singapore OPTIONS
(instance_partition='asia-partition');
```

The equivalent DDL statement for usermanagement database is:

```
ALTER DATABASE usermanagement SET OPTIONS (
 opt_in_dataplacement_preview = true
);
CREATE PLACEMENT France OPTIONS (
 instance_partition = 'eur-partition'
);
CREATE PLACEMENT Germany OPTIONS (
 instance_partition = 'eur-partition'
);
CREATE PLACEMENT Singapore OPTIONS (
 instance_partition = 'asia-partition'
);
CREATE PLACEMENT US OPTIONS (
 instance_partition = 'na-partition'
);
CREATE TABLE Users (
 UserId INT64 NOT NULL,
 Name STRING(MAX) NOT NULL,
 Location STRING(MAX) NOT NULL PLACEMENT KEY,
) PRIMARY KEY(UserId);
```

Insert records and move them between partitions

Gcloud command like the following can be used to insert and move records. Replace the string in **--sql=""** with the SQL text in each sub section below.

```
gcloud beta spanner databases execute-sql \
   usermanagement \
   --instance=user-instance \
    --sql="INSERT INTO Users(UserId, Name, Location)
VALUES (4242, 'John', 'US')"
```

Records placed on the North American partition na-partition:

```
INSERT INTO Users(UserId, Name, Location)
VALUES (4242, 'John', 'US');

INSERT INTO Users(UserId, Name, Location)
VALUES (3435, 'Helen', 'US');
```

Records placed on the Europe partition eur-partition:

```
INSERT INTO Users(UserId, Name, Location)
VALUES (4567, 'Panos', 'France');

INSERT INTO Users(UserId, Name, Location)
VALUES (6546, 'Mark', 'Germany');
```

Records placed on the Asia partition asia-partition:

```
INSERT INTO Users(UserId, Name, Location)
VALUES (8754, 'Dan', 'Singapore');

INSERT INTO Users(UserId, Name, Location)
VALUES (1348, 'Rose', 'Singapore');
```

Moving records between partitions:

```
UPDATE Users u SET u.Location='Germany'

WHERE u.UserId=4242;
```

Moving sets of records between partitions with interleaved tables:

Interleaved tables serve as a schema modeling tool specific to Spanner that optimizes query performance in parent-child relationships by co-locating child rows with parent rows in storage. Additionally, they facilitate the geo-placement of entire sets of related records.

Consider an interleaved table where a user can have one or more user generated content.

```
CREATE TABLE UserContent (
 UserId INT64 NOT NULL,
 ContentId STRING(MAX) NOT NULL,
 content STRING(MAX) NOT NULL,
) PRIMARY KEY(UserId, ContentId),
INTERLEAVE IN PARENT Users;
```

In the following example all user created content managed in the UserContent table will be colocated with the parent record in the Users table, Germany in this case for the Users with the UserId 6546.

```
INSERT INTO UserContent(UserId, ContentId, content)
VALUES (6546, "5cc371d7-d6dc-43b1-b763-981ba9849fab",
"mycontent1");

INSERT INTO UserContent(UserId, ContentId, content)
VALUES (6546, "e45a218d-449b-47d9-a5b5-0f815cde1e63",
"mycontent2");

INSERT INTO UserContent(UserId, ContentId, content)
VALUES (6546, "6cfdb80d-6140-46f7-b048-d0872c621a4c",
"mycontent3");
```

Changing the placement of the parent record from “Germany” to “US” will result in moving the entire interleaved set of records of the respective UserId to the designated placement region:

```
UPDATE Users SET Location = "US"
WHERE UserId = 6546;
```

Example 2
Placement as key

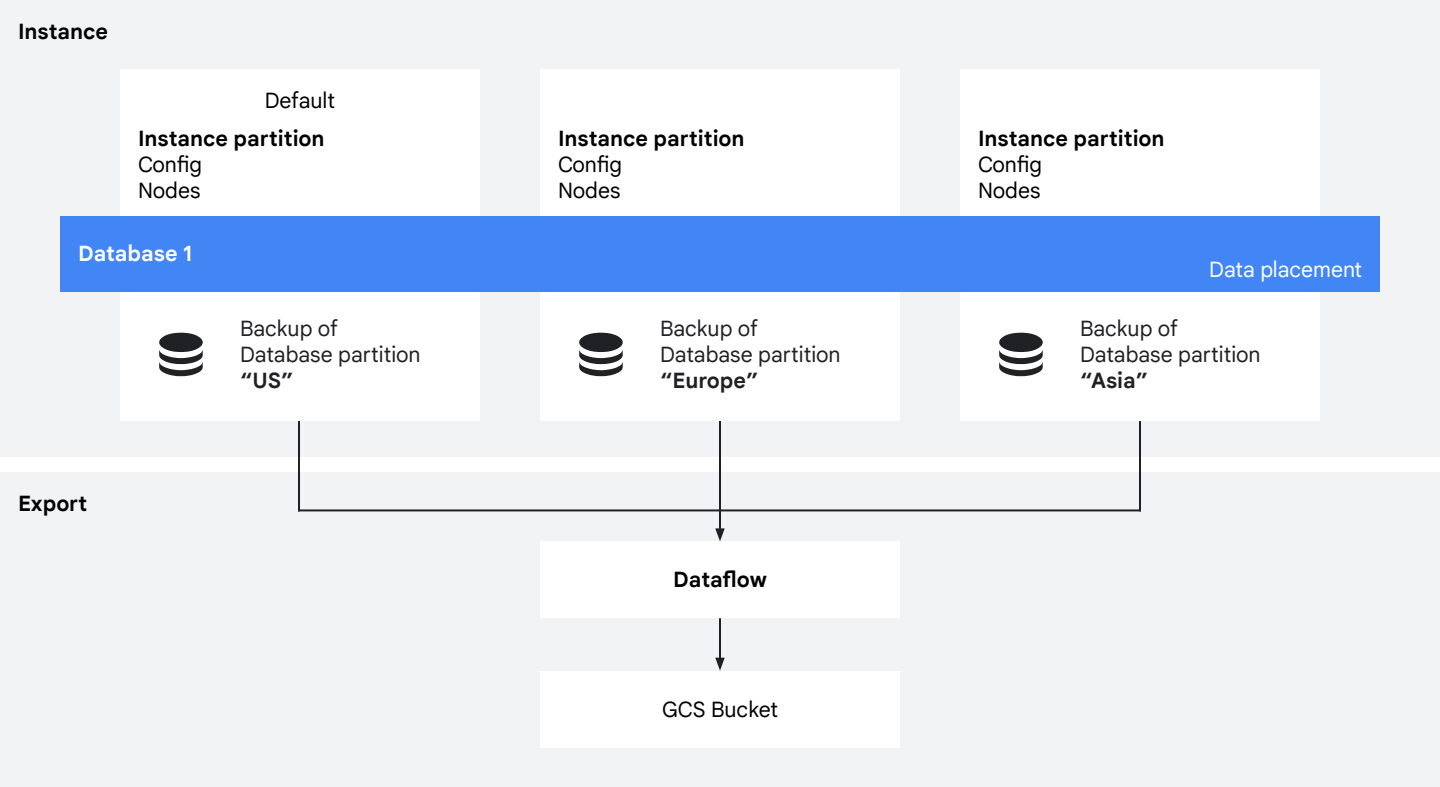
An alternative schema design to manage a users table could be to use the placement as the key column. With this schema design we advise to create a separate table (e.g: placements) and interleave relevant tables (i.e. users in this example).

As a primary key with interleaving

```
CREATE TABLE Locations (
  Location STRING(MAX) NOT NULL PLACEMENT KEY
) PRIMARY KEY(Location);
CREATE TABLE Users (
  Location STRING(MAX) NOT NULL,
  UserId INT64 NOT NULL,
  Name STRING(MAX) NOT NULL,
  ... more columns...
) PRIMARY KEY(Location, UserId),
INTERLEAVE IN Locations;
```

	Pros	Cons
Placement as non-primary Key	• Lookup by UserId • UserId is globally unique • Easy moves	
Placement as PK with interleaving	• Less compute overhead (due to metadata for placement rows) • More performant range queries, support for regional indexes	• Lookups require Location, UserId is unique within a location only

Backups



Managed backups will be instantiated at the instance level per database. Backups will be partitioned and stored in the corresponding instance partition. This means that partitioned data resides in the instance partition's regional configuration adhering to potential jurisdictional requirements and constrained by the availability of that particular regional configuration.

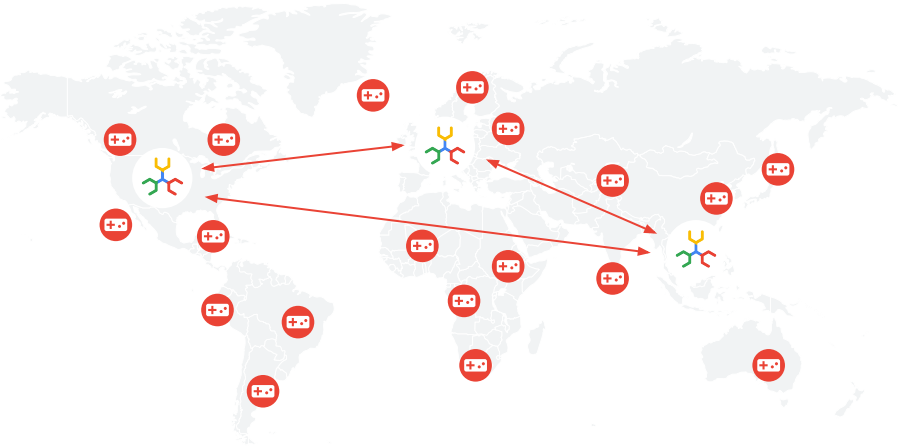
Database exports (through Dataflow) are able to export the entire database into one single bucket which is either in a specific data center region or multi-region.

Sample architectures

Scenario 1: Gaming

The critical role of data management systems

The design of a gaming platform involves complex considerations, particularly when it comes to managing data across a global user base. Data management systems in multiplayer gaming platforms are not only about handling large data volumes - they are crucial in shaping the player's experience, ensuring security, and fostering community. It directly influences user engagement, security, and overall gameplay experience.



A well-architected system ensures that players have a seamless and engaging experience, regardless of their geographical location.

The solutions often involve using distributed databases that can handle high volumes of simultaneous transactions and provide near-real-time access to data to power functions such as:

User management: security and accessibility

User management is the backbone of personalization and security in gaming platforms. It encompasses several key functions:

- User data: Managing personal information securely, ensuring data integrity and privacy.
- Authentication and authorization: Critical for securing user accounts and preventing unauthorized access, this involves verifying user identities and granting correct access levels based on permissions.
- Entitlements: Customer Identity and Access Management (CIAM) services manage user identities and profile data across a broad spectrum, enabling personalized experiences and managing digital identities efficiently.

A robust data management system ensures these processes are seamless and secure, enhancing user trust and platform reliability.

Social interactions: Building community

Social features within games, such as profiles and friend systems, not only enhance user engagement but also build a sense of community among players. Data management systems track and store complex social graphs, friend lists, and interaction histories, facilitating:

- Profiles: Storing and retrieving player information and preferences, allowing players to customize their gaming identity.
- Friends: Managing connections between players, including adding friends, blocking users, and viewing online status, which supports in-game communication and collaboration.

These social elements require a responsive and reliable data infrastructure to ensure real-time updates and maintain the continuity of social interactions, vital for retaining players and fostering community spirit.

Gameplay dynamics: Enriching the experience

The most direct impact of data management on players is in gameplay dynamics, which include:

- Quests and activities: Tracking the status of missions and challenges, storing results, and updating progress in real time.
- Progress: Systematic recording of player advancement through different levels or stages in the game.
- Inventory and items: Managing databases that store information about player-owned items, their properties, and statuses.
- Rank: Calculating and updating player rankings based on game outcomes and achievements.

Effective management of this data ensures a seamless and engaging gameplay experience, with immediate feedback and updates that are crucial for games requiring quick reflexes and strategic planning.

The challenges

Developers & architects must navigate issues related to database uniformity, latency, mobility, cross-region interactions, and policy enforcement.

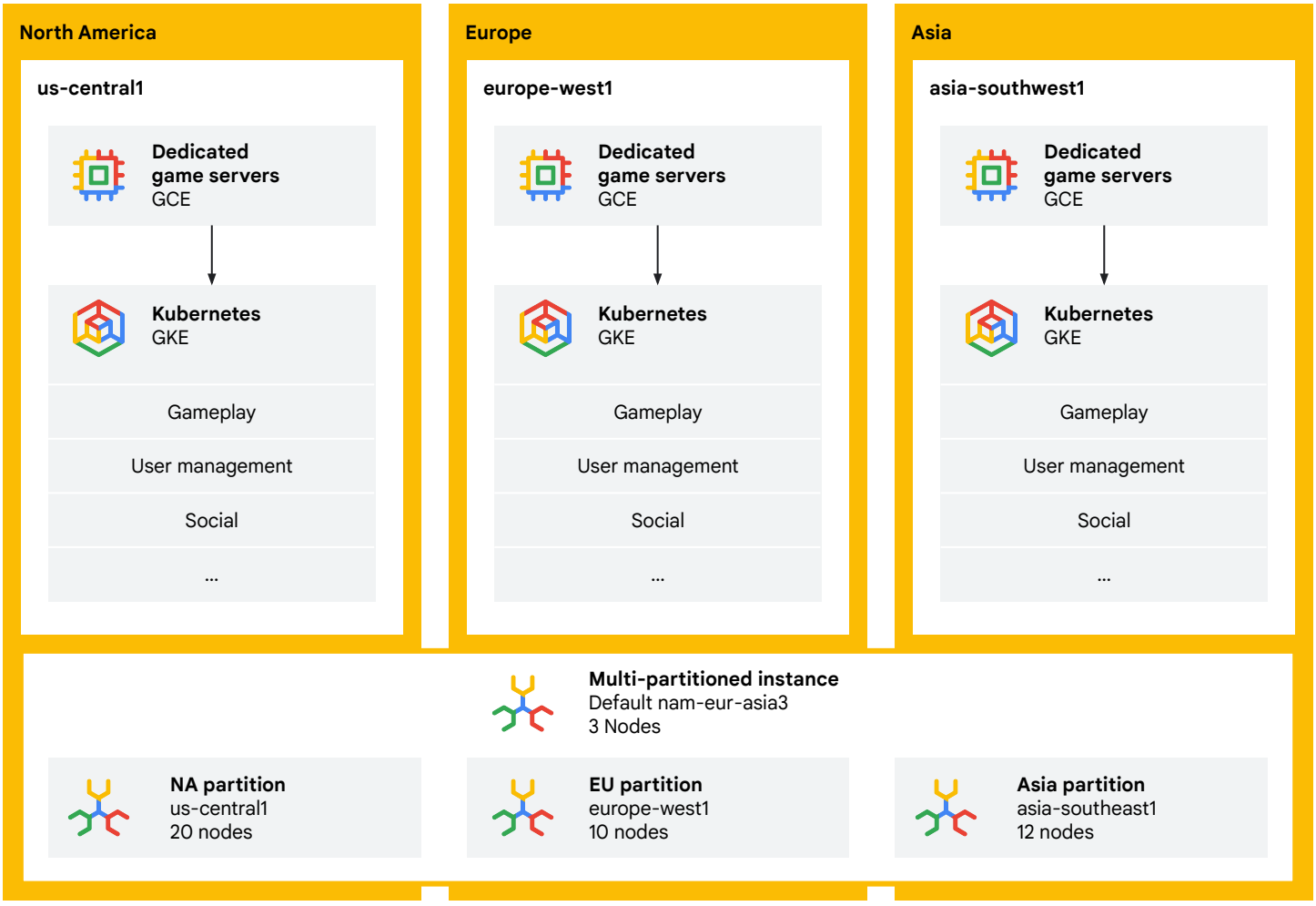
Global (single) source of truth vs complexity: A single consistent source of truth is desirable to store and manage all aspects of such a gaming platform. This brings scalability & geo-distribution challenges with traditional database systems and would require complex sharding and replication topologies to be built and maintained. Reconciliation of dispersed silos or synchronization between different systems in case of changes are not acceptable both due to potential lag and inconsistent states in between model changes.

Latency vs experience: Having a single source of truth cross region is desirable, but with a global user base this brings latency challenges. If databases aren't located close to the users optimized for both read and write latencies, it can adversely impact user experience resulting.

Scalability vs cost effectiveness vs user base distribution: Gaming platforms require scalability to be able to serve data with consistently low latency times in an economical way. This requires to efficiently deal (i.e. elastically scale up & down) with weekend, day/night patterns as well as handling peak events (launches, promotional events). At the same time the user bases might be unevenly distributed globally with a concentration in specific regions, but future upside potential for growth in others.

Data mobility: Data mobility is relevant in cases where players relocate across different geographical regions where their data must transition smoothly to maintain a consistent gaming experience.

Architecture example



The architecture diagram illustrates a globally distributed microservices-based system designed to serve an online game to users across different geographic regions addressing the mentioned four major challenges.

To cater to these requirements, the architecture is split across three major regions: North America, Europe and Asia. Each region has its own dedicated infrastructure resources to serve users within its proximity, ensuring low latency and good performance.

Each region has the following components:

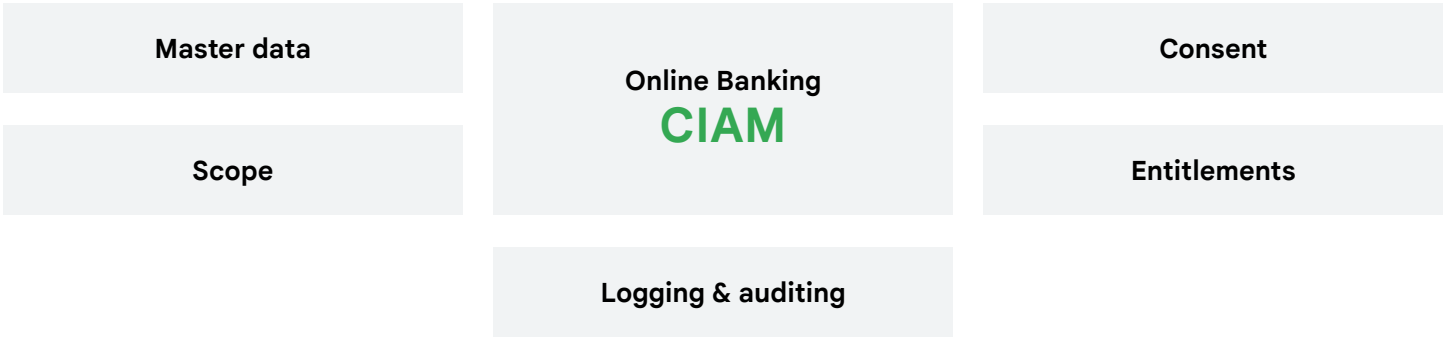
Dedicated game servers (GCE)
These are dedicated servers and environments specifically for hosting game logic, game sessions, and managing game-related configurations and settings.

Kubernetes cluster (GKE)
A Kubernetes cluster is deployed in each region, acting as a container orchestration platform for running various microservices or components of the application. These components are organized into different layers or modules, such as gameplay, user management, social features, etc.

Regionally partitioned data in a global instance
While each region has its dedicated resources, there is a single geo-partitioned Spanner instance that acts as a single global & logical database storing data, such as user accounts, game assets, social graphs, player progress, etc... Each region has its own transparent partition of the database, storing data specific to users and activities within that geographic area. This approach promotes data locality and reduces the need to access remote data centers for most operations.

Scenario 2:
CIAM in Finance (online banking and payments)

Customer identity and access management (CIAM) are core service components in modern architectures managing user identities and profile data enabling personalized experiences and efficiently managing digital identities. Many organizations design and develop CIAM solutions with Spanner at its core to custom tailor user experiences along customer journeys both in the consumer space and enterprise segments. These solutions call for modern, seamless, friction-free and personalized customer experiences across multi-channel (web, mobile, phone, ...).

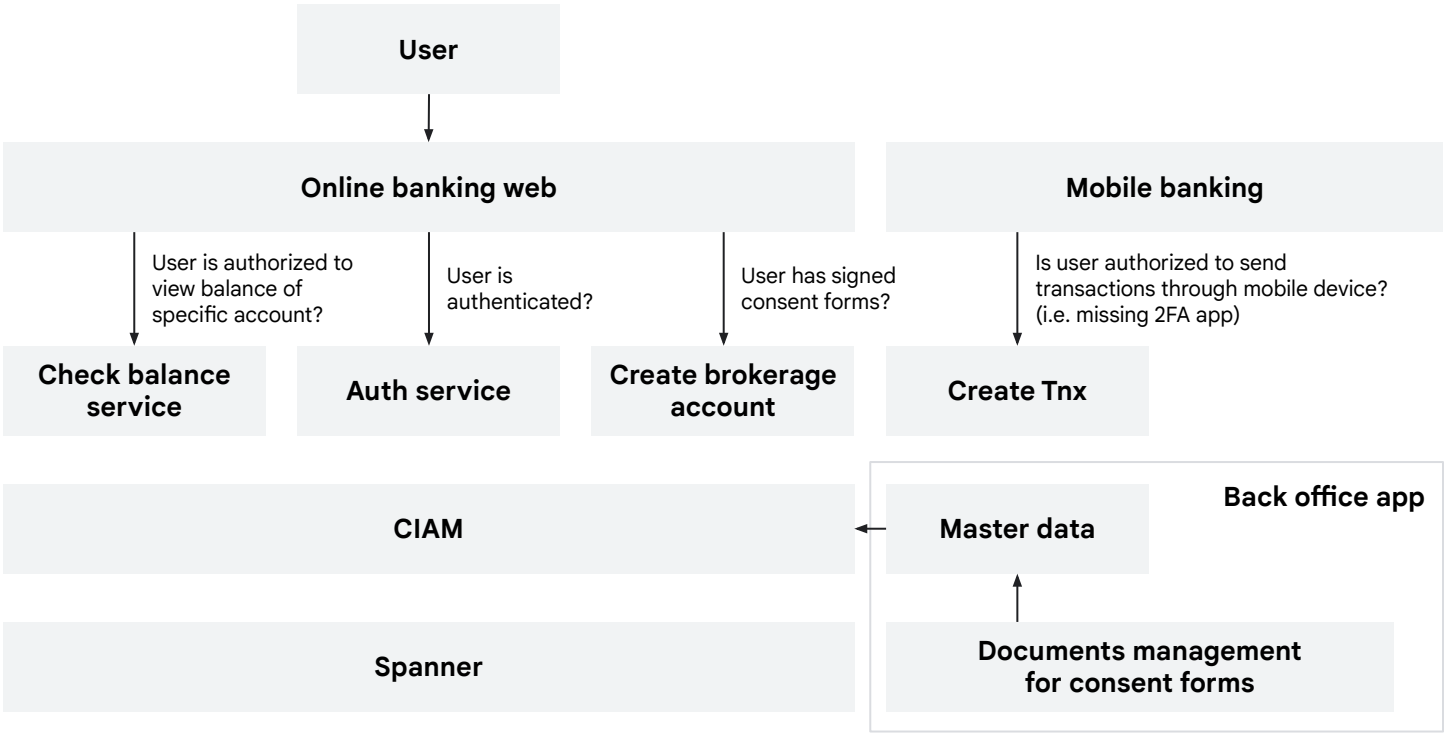


As an example in online banking and payments, CIAM data structures require managing customers, partners, accounts and product master data which might be centrally consolidated from federated identity systems in an event-driven architecture. This master data is then enriched in CIAM with organization specific domain entities and workflows such as catering for regulatory driven entitlements as part of the authorization scopes.

On one hand the access scopes & hierarchies need to cater for channel-specific customer experience related authorizations. These include simple things such as enabling/disabling certain online banking features, views or workflows. As an example, default permissions can be defined and applied to customer segments e.g.

- Online banking users have by default access to mobile banking and balance views
- And for instance these users cohorts who have granted online banking authorization can change profile data such as their postal address

But it can be also used centrally to manage access to specific products through certain channels in defined postal code areas.



These hierarchies & scopes extend to regulatory privilege consent workflows such as a natural person needs to consent and sign waivers to be able to trade derivatives if they activate a brokerage account. The workflow status needs to be tracked in the CIAM system and is often powered by 3rd party applications such as a document management or workflow system.

The challenges

CIAM requires secure, highly reliable and flexible solutions to support such workflows. Specifically in the finance sector compliance rules drive the requirement for such systems (e.g. entitlement management) as they need auditable authorization models to resources based on verification & consent processes. CIAM systems often need to be designed for multi-tenancy to be able to centrally manage and integrate acquisitions and subsidiaries.

The reasons why Spanner is attractive for this type of use case is due to these seven requirements:

High availability: CIAM is a core gatekeeper component to all other services. If this component is down no other dependent service can function.

Single (global) consistent view: A single consistent source of truth is desirable to store and manage all aspects of a CIAM component. This brings scalability & geo-distribution challenges with traditional database systems and would require complex sharding and replication topologies to be built and maintained. Reconciliation of dispersed silos or synchronization between different systems in case of an authorization change are not acceptable both due to potential lag and inconsistent states in between model changes.

Catering for global user base: If CIAM serves a global user base, a globally available datastore is required. Applications close to the customer need to serve data with relatively low latency. This requires the option to place consistent replicas close to the data consumer.

Very high durability: User data, authorization rules, intermediate workflow states etc. need to have a very high durability not just from the sake of a good user experience (you don't lose data), but also in finance this is driven by regulatory frameworks. Many jurisdictions require data to be redundantly stored in geo-separated regions with RPO-0 in case of a disaster.

Scalable & cost efficient: CIAM datastores require scalability to be able to serve data with consistently low latency times in an economical way. This requires to efficiently deal (i.e. elastically scale up & down) with weekend, day/night patterns as well as handling peak events (launches, promotional events). CIAMs power a broad range of services. When new functions are connected, the database needs to scale with the additional demand.

In finance a CIAM platform might power initially web online banking. Later the platform might get extended to mobile, chatbots, integrate into call centers or extend to 3rd party open banking applications. Another example is also if these banking platforms plan for multi-tenants such as onboarding acquisitions or subsidiaries.

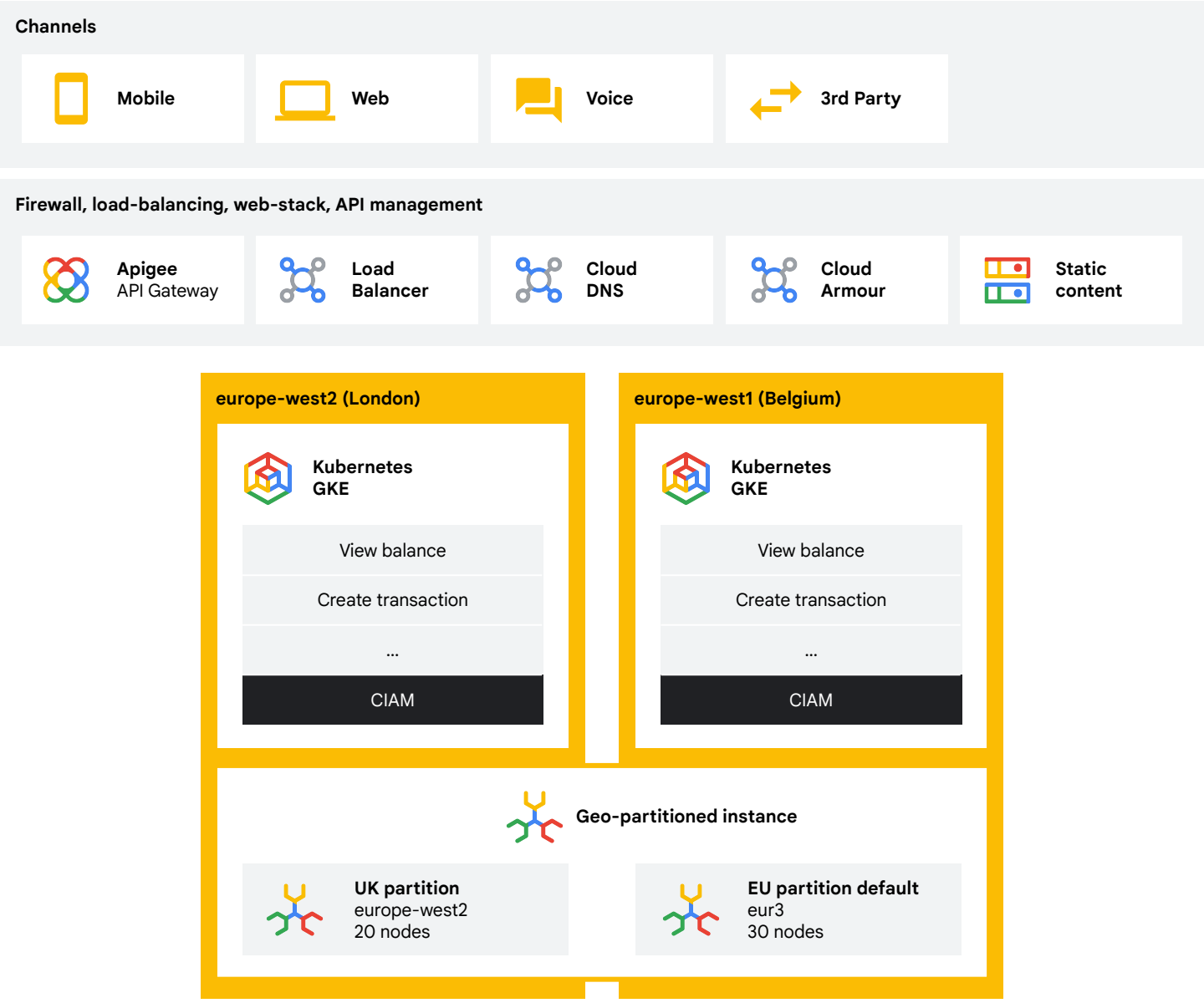
Audit: Changes to authorization rules, hierarchies and respectively authentication & authorization requests and responses need to be tracked. These include granted scopes, attempts, violations, redirect URIs, … . These logs can be used for debugging or fraud & abuse analytics purposes. In some industries this is also a regulatory requirement with immutable & tamper proof retention locks.

Security (CMEK, EKM): Some industries or specific features (e.g. handling financial transactions & payments) require a very high standard in terms of security. The baseline is that the database encrypts data at-rest & in-transit. Some organizations require in addition customer managed encryption keys (CMEK) or integration of external key management systems (EKM).

Audit: Changes to authorization rules, hierarchies and respectively authentication & authorization requests and responses need to be tracked. These include granted scopes, attempts, violations, redirect URIs, … . These logs can be used for debugging or fraud & abuse analytics purposes. In some industries this is also a regulatory requirement with immutable & tamper proof retention locks.

Security (CMEK, EKM): Some industries or specific features (e.g. handling financial transactions & payments) require a very high standard in terms of security. The baseline is that the database encrypts data at-rest & in-transit. Some organizations require in addition customer managed encryption keys (CMEK) or integration of external key management systems (EKM).

Architecture example



The architecture diagram illustrates CIAM services in the context of a multi-channel online banking application along with various related (micro) services.

The core application is deployed across two regions: europe-west1 (Belgium) and europe-west2 (London). In each region, there is a Kubernetes cluster (GKE) hosting various microservices or components, such as view balance, create transaction, ...

Each region has a CIAM (customer identity and access management) component, which is one of the most critical components as all services depend on its availability.

Therefore the architecture employs a geo-partitioning strategy to manage data and services across different regions and cater for differently sized user populations.

- UK Partition in europe-west1 London and
- EU Partition in eur3 (Belgium, Netherlands and witness in Finland)

The UK partition is a regional instance where the resources are spread across one data center in London adhering to DR regulatory requirements which are satisfied by the zone separation of that particular data center campus.

The EU partition is spread out in a multi-region deployment to provide geo-redundancy with RPO-0 across a larger geographical region.

Preview limitations, GA and beyond

Spanner geo-partitioning comes with a range of limitations at preview time that will be subsequently removed throughout future release cycles (GA and beyond). Unavailable features are subject to change. These include, but are not limited to:

- Data residency guarantees
- CMEK
- Managed backups
- PostgreSQL
- Limited client library support
- Monitoring gaps
- ChangeStreams
- DML limitations will apply

For most up-to-date information on current limitations, please look at our current documentation at

https://cloud.google.com/spanner/docs/geo-partitioning#limitations.