Using Amazon DynamoDB global tables

AWS Prescriptive Guidance

AWS Prescriptive Guidance: Using Amazon DynamoDB global tables

Table of Contents

Using Amazon DynamoDB global tables

Jason Hunter, Amazon Web Services (AWS)

March 2024 (document history)

Global tables build on the global Amazon DynamoDB footprint to provide you with a fully managed, multi-Region, and multi-active database that delivers fast and local read and write performance for massively scaled, global applications. Global tables replicate your DynamoDB tables automatically across your choice of AWS Regions. No application changes are required because global tables use existing DynamoDB APIs. There are no upfront costs or commitments for using global tables, and you pay only for the resources you use.

This guide explains how to use DynamoDB global tables effectively. It provides key facts about global tables, explains the feature's primary use cases, introduces a taxonomy of three different write models you should consider, walks through the four main request routing choices you might implement, discusses ways to evacuate a Region that's live or a Region that's offline, explains how to think about throughput capacity planning, and provides a checklist of things to consider when you deploy global tables.

This guide fits into a larger context of AWS multi-Region deployments, as covered in the AWS Multi-Region Fundamentals whitepaper and the Data resiliency design patterns with AWS video.

Contents

Overview of global tables

Key facts

- There are two versions of global tables: version 2017.11.29 (legacy) (sometimes called *v1*) and version 2019.11.21 (current) (sometimes called *v2*). This guide focuses exclusively on the current version.
- DynamoDB (without global tables) is a Regional service, which means that it is highly available and intrinsically resilient to failures of infrastructure, including the failure of an entire Availability Zone. A single-Region DynamoDB table is designed for 99.99% availability. For more information, see the DynamoDB service-level agreement (SLA).
- A DynamoDB global table replicates its data between two or more Regions. A multi-Region DynamoDB table is designed for 99.999% availability. With proper planning, global tables can help create an architecture that is resilient to Regional failures.
- Global tables employ an active-active replication model. From the perspective of DynamoDB, the table in each Region has equal standing to accept read and write requests. After receiving a write request, the local replica table replicates the write operation to other participating remote Regions in the background.
- Items are replicated individually. Items that are updated within a single transaction might not be replicated together.
- Each table partition in the source Region replicates its write operations in parallel with every other partition. The sequence of write operations within a remote Region might not match the sequence of write operations that happened within the source Region. For more information about table partitions, see the blog post Scaling DynamoDB: How partitions, hot keys, and split for heat impact performance.
- A newly written item is usually propagated to all replica tables within a second. Nearby Regions tend to propagate faster.
- Amazon CloudWatch provides a `ReplicationLatency` metric for each Region pair. It is calculated by looking at arriving items, comparing their arrival time with their initial write time, and computing an average. Timings are stored within CloudWatch in the source Region. Viewing the average and maximum timings can be useful for determining the average and worst-case replication lag. There is no SLA on this latency.
- If an individual item is updated at about the same time (within this `ReplicationLatency` window) in two different Regions, and the second write operation happens before the first

write operation was replicated, there's a potential for write conflicts. Global tables resolve such conflicts by using a *last writer wins* mechanism, based on the timestamp of the write operations. The first operation "loses" to the second operation. These conflicts aren't recorded in CloudWatch or AWS CloudTrail.

- Each item has a last write timestamp held as a private system property. The *last writer wins* approach is implemented by using a conditional write operation that requires the incoming item's timestamp to be greater than the existing item's timestamp.
- A global table replicates all items to all participating Regions. If you want to have different replication scopes, you can create multiple global tables and assign each table different participating Regions.
- The local Region accepts write operations even if the replica Region is offline or `ReplicationLatency` grows. The local table continues to attempt replicating items to the remote table until each item succeeds.
- In the unlikely event that a Region goes fully offline, when it comes back online later, all pending outbound and inbound replications will be retried. No special action is required to bring the tables back in sync. The *last writer wins* mechanism ensures that the data eventually becomes consistent.
- You can add a new Region to a DynamoDB table at any time. DynamoDB handles the initial sync and ongoing replication. You can also remove a Region (even the original Region), and this will delete the local table in that Region.
- DynamoDB does not have a global endpoint. All requests are made to a Regional endpoint that accesses the global table instance that's local to that Region.
- Calls to DynamoDB should not go across Regions. The best practice is for an application that is homed to one Region to directly access only the local DynamoDB endpoint for its Region. If problems are detected within a Region (in the DynamoDB layer or in the surrounding stack), end user traffic should be routed to a different application endpoint that's hosted in a different Region. Global tables ensure that the application homed in every Region has access to the same data.

Use cases

Global tables provide these common benefits:

- **Lower-latency read operations.** You can place a copy of the data closer to the end user to reduce network latency during read operations. The data is kept as fresh as the `ReplicationLatency` value.
- **Lower-latency write operations.** An end user can write to a nearby Region to reduce network latency and the time to complete the write operation. The write traffic must be carefully routed to ensure that there are no conflicts. Techniques for routing are discussed in a later section.
- **Increased resiliency and disaster recovery.** If a Region has degraded performance or a full outage, you can evacuate it (move away some or all requests going to that Region) and meet a recovery point objective (RPO) and recovery time objective (RTO) measured in seconds. Using global tables also increases the DynamoDB SLA for monthly uptime percentage from 99.99% to 99.999%.
- **Seamless Region migration.** You can add a new Region and then delete the old Region to migrate a deployment from one Region to another, without any downtime at the data layer.

For example, Fidelity Investments presented at re:Invent 2022 on how they use DynamoDB global tables for their Order Management System. Their goal was to achieve reliably low latency processing at a scale they couldn't attain with on-premises processing while also maintaining resilience to Availability Zone and Regional failures.

Write modes for global tables

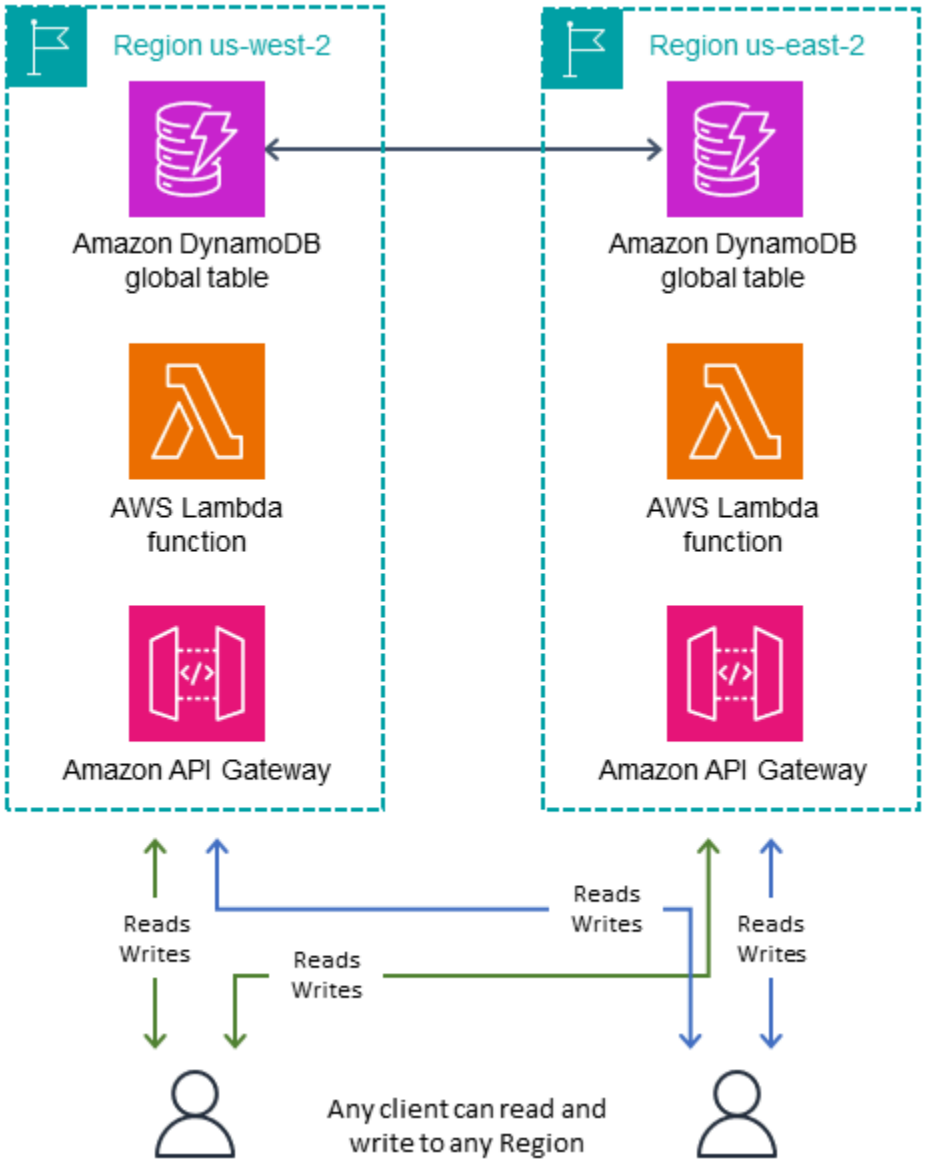
Global tables are always active-active at the table level. However, you might want to treat them as active-passive by controlling how you route write requests. For example, you might decide to route write requests to a single Region to avoid potential write conflicts.

There are three main managed write patterns, as explained in the next three sections. You should consider which write pattern fits your use case. This choice affects how you route requests, evacuate a Region, and handle disaster recovery. The guidance in later sections depends on your application's write mode.

Topics

Write to any Region mode (no primary)

The *write to any Region* write mode is fully active-active and doesn't impose restrictions on where a write operation may occur. Any Region can accept a write request at any time. This is the simplest mode; however, it can be used only with some types of applications. It's suitable when all write operations are idempotent. *Idempotent* means that they are safely repeatable so that concurrent or repeated write operations across Regions are not in conflict—for example, when a user updates their contact data. It also works well for an append-only dataset where all write operations are unique inserts under a deterministic primary key, which is a special case of being idempotent. Lastly, this mode is suitable where the risk of conflicting write operations is acceptable.



The *write to any Region* mode is the most straightforward architecture to implement. Routing is easier because any Region can be the write target at any time. Failover is easier, because any recent write operations can be replayed any number of times to any secondary Region. Where possible, you should design for this write mode.

For example, several video streaming services use global tables for tracking bookmarks, reviews, watch status flags, and so on. These deployments can use the *write to any Region* mode as long as they ensure that every write operation is idempotent. This will be the case if every update —for example, setting a new latest time code, assigning a new review, or setting a new watch status—assigns the user's new state directly, and the next correct value for an item doesn't depend on its current value. If, by chance, the user's write requests are routed to different

Regions, the last write operation will persist and the global state will settle according to the last assignment. Read operations in this mode will eventually become consistent, delayed by the latest `ReplicationLatency` value.

In another example, a financial services firm uses global tables as part of a system to maintain a running tally of debit card purchases for each customer, to calculate that customer's cash-back rewards. New transactions stream in from around the world and go to multiple Regions. This firm was able to use the *write to any Region* mode with a careful redesign. The initial design sketch maintained a single `RunningBalance` item per customer. Customer actions updated the balance with an `ADD` expression, which is not idempotent (because the new correct value depends on the current value), and the balance got out of sync if there were two write operations to the same balance at around the same time in different Regions. The redesign uses event streaming, which works like a ledger with an append-only workflow. Each customer action appends a new item to the item collection maintained for that customer. (An *item collection* is the set of items that share a primary key but have different sort keys.) Each write operation is an idempotent insert that uses the customer ID as the partition key and the transaction ID as the sort key. This design makes the calculation of the balance harder because it requires a `Query` to pull the items followed by some client-side math, but it makes all write operations idempotent and achieves significant simplifications in routing and failover. (This is discussed in more detail later in this guide.)

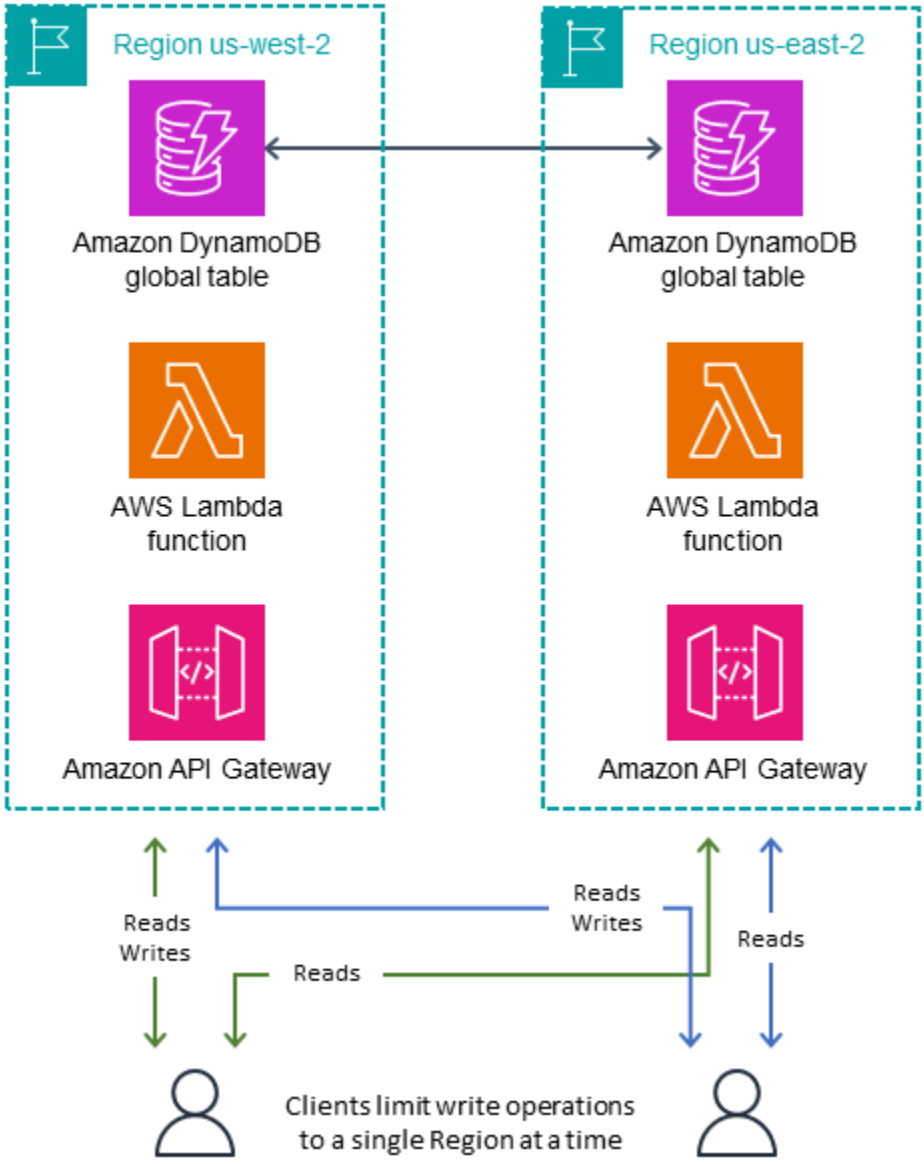
A third example involves a company that provides online ad placement services. This company decided that a low risk of data loss would be acceptable to achieve the design simplifications of the *write to any Region* mode. When they serve ads, they have just a few milliseconds to retrieve enough metadata to determine which ad to show, and then to record the ad impression so they don't repeat the same ad soon. They use global tables to get both low-latency read operations for end users across the world and low-latency write operations. They record all ad impressions for a user within a single item, which is represented as a growing list. They use one item instead of appending to an item collection, so they can remove older ad impressions as part of each write operation without paying for a delete operation. This write operation is not idempotent; if the same end user sees ads served out of multiple Regions at approximately the same time, there's a chance that one write operation for an ad impression could overwrite another. The risk is that a user might see an ad repeated once in a while. They decided that this is acceptable.

Write to one Region mode (single primary)

The *write to one Region* write mode is active-passive and routes all table write operations to a single active Region. (DynamoDB doesn't have a notion of a single active Region; the layer outside DynamoDB manages this.) The *write to one Region* mode avoids write conflicts by ensuring that

write operations flow only to one Region at a time. This write mode helps when you want to use conditional expressions or transactions. These expressions aren't possible unless you know that you're acting against the latest data, so they require sending all write requests to a single Region that has the latest data.

Eventually consistent read operations can go to any of the replica Regions to achieve lower latencies. Strongly consistent read operations must go to the single primary Region.



It's sometimes necessary to change the active Region in response to a Regional failure, as discussed later. Some users change the currently active Region on a regular schedule, such as implementing a *follow-the-sun* deployment. This places the active Region near the geography that has the most activity (usually where it's daytime, thus the name), which results in the lowest latency read and

write operations. It also has the side benefit of calling the Region-changing code daily, and making sure that it's well tested before any disaster recovery.

The passive Region(s) might keep a downscaled infrastructure surrounding DynamoDB that gets built up only if it becomes the active Region. This guide doesn't cover pilot light and warm standby designs. For more information, you can read the blog post Disaster Recovery (DR) Architecture on AWS, Part III: Pilot Light and Warm Standby.

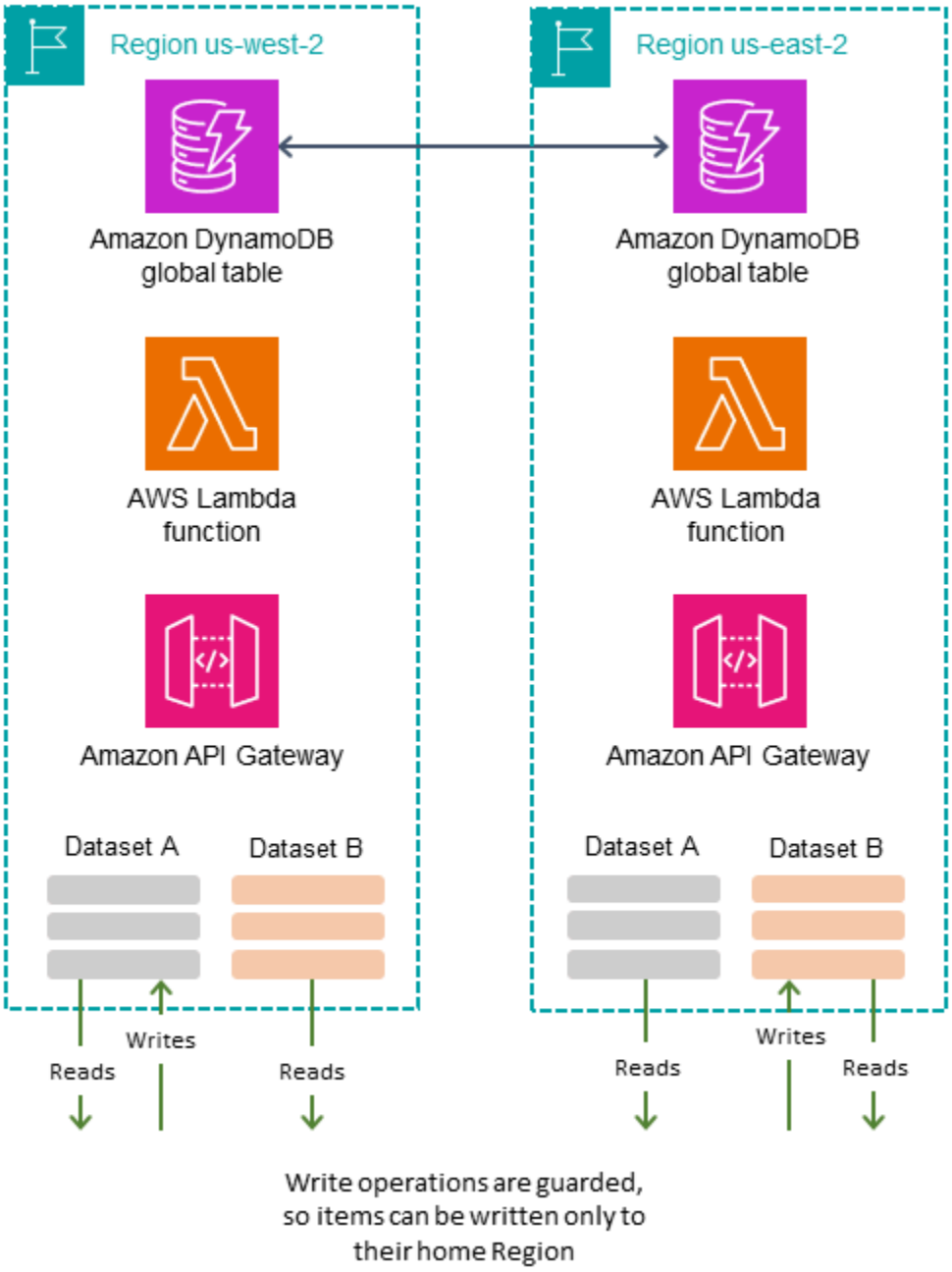
Using the *write to one Region* mode works well when you use global tables for low-latency, globally distributed read operations. An example is a large social media company that needs to have the same reference data available in every Region around the world. They don't update the data often, but when they do, they write to only one Region to avoid any potential write conflicts. Read operations are always allowed from any Region.

As another example, consider the financial services company discussed earlier that implemented the daily cash-back calculation. They used *write to any Region* mode to calculate the balance but *write to one Region* mode to track the cash-back payments. If they want to reward a penny for every $10 spent, they have to `Query` for all transactions from the previous day, calculate the total spent, write the cash-back decision to a new table, delete the queried set of items to mark them as consumed, and replace them with a singular item that stores any remainder that should go into the next day's calculations. This work requires transactions, so it works better with *write to one Region* mode. An application can mix write modes, even on the same table, as long as the workloads have no chance of overlapping.

Write to your Region mode (mixed primary)

The *write to your Region* write mode assigns different data subsets to different home Regions and allows write operations to an item only through its home Region. This mode is active-passive but assigns the active Region based on the item. Every Region is primary for its own non-overlapping dataset, and write operations must be guarded to ensure proper locality.

This mode is similar to *write to one Region* except that it enables lower-latency write operations, because the data associated with each user can be placed in closer network proximity to that user. It also spreads the surrounding infrastructure more evenly between Regions and requires less work to build out infrastructure during a failover scenario, because all Regions have a portion of their infrastructure already active.



You can determine the home Region for items in several ways:

- Intrinsic: Some aspect of the data, such as a special attribute or a value embedded within its partition key, makes its home Region clear. This technique is described in the blog post Use Region pinning to set a home Region for items in an Amazon DynamoDB global table.
- Negotiated: The home Region of each dataset is negotiated in some external manner, such as with a separate global service that maintains assignments. The assignment might have a finite duration after which it's subject to re-negotiation.

- Table-oriented: Instead of creating a single replicating global table, you create the same number of global tables as replicating Regions. Each table's name indicates its home Region. In standard operations, all data is written to the home Region while other Regions keep a read-only copy. During a failover, another Region temporarily adopts write duties for that table.

For example, imagine that you're working for a gaming company. You need low-latency read and write operations for all gamers around the world. You assign each gamer to the Region that's closest to them. That Region takes all their read and write operations, ensuring strong read-after-write consistency. However, when a gamer travels or if their home Region suffers an outage, a complete copy of their data is available in alternative Regions, and the gamer can be assigned to a different home Region.

As another example, imagine that you're working at a video conferencing company. Each conference call's metadata is assigned to a particular Region. Callers can use the Region that's closest to them for lowest latency. If there's a Region outage, using global tables allows quick recovery because the system can move the processing of the call to a different Region where a replicated copy of the data already exists.

Routing strategies for global tables

Perhaps the most complex piece of a global table deployment is managing request routing. Requests must first go from an end user to a Region that's chosen and routed in some manner. The request encounters some stack of services in that Region, including a compute layer that perhaps consists of a load balancer backed by an AWS Lambda function, container, or Amazon Elastic Compute Cloud (Amazon EC2) node, and possibly other services, including maybe another database. That compute layer communicates with DynamoDB. It should do that by using the local endpoint for that Region. The data in the global table replicates to all other participating Regions, and each Region has a similar stack of services around its DynamoDB table.

The global table provides each stack in the various Regions with a local copy of the same data. You might consider designing for a single stack in a single Region and anticipate making remote calls to a secondary Region's DynamoDB endpoint if there's an issue with the local DynamoDB table. This is **not** best practice. The latencies associated with going across Regions might be 100 times higher than for local access. A back-and-forth series of 5 requests might take milliseconds when performed locally but seconds when crossing the globe. It's better to route the end user to another Region for processing. To ensure resiliency, you need replication across multiple Regions: replication of the compute layer as well as the data layer.

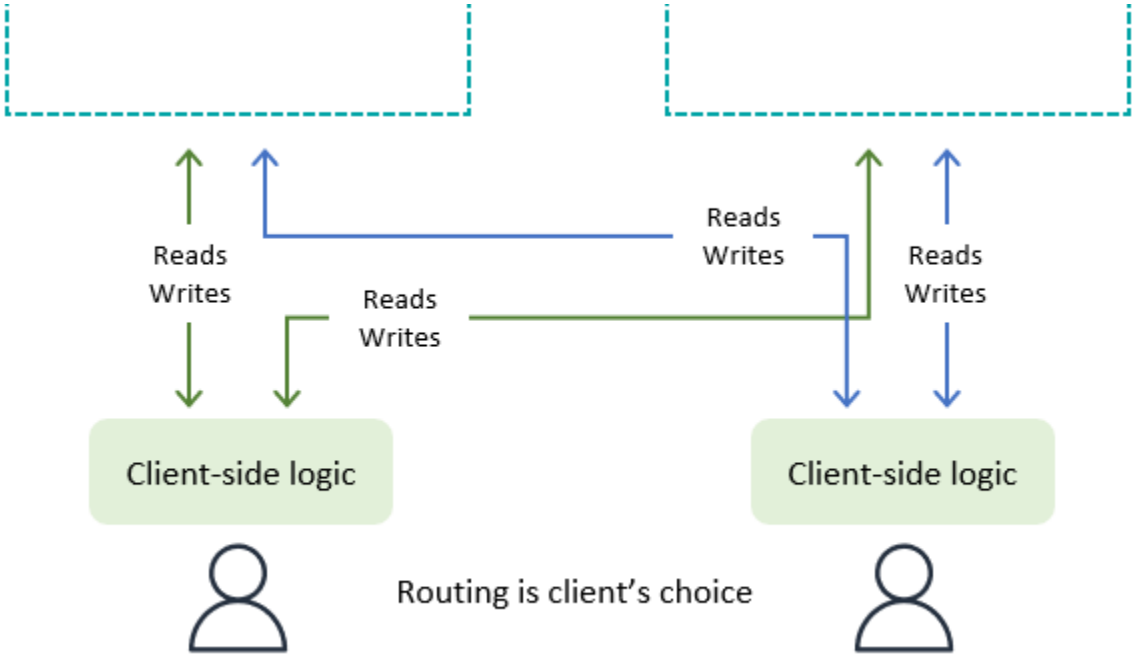
There are numerous techniques for routing an end user request to a Region for processing. The right choice depends on your write mode and your failover considerations. This section discusses four options: client-driven, compute-layer, Amazon Route 53, and AWS Global Accelerator.

Topics

Client-driven request routing

With client-driven request routing, the end user client (an application, a web page with JavaScript, or another client) keeps track of the valid application endpoints (for example, an Amazon API Gateway endpoint rather than a literal DynamoDB endpoint) and uses its own embedded logic

to choose the Region to communicate with. It might choose based on random selection, lowest observed latencies, highest observed bandwidth measurements, or locally performed health checks.



As an advantage, client-driven request routing can adapt to things such as real-world public internet traffic conditions to switch Regions if it notices any degraded performance. The client must be aware of all potential endpoints, but launching a new Regional endpoint is not a frequent occurrence.

With *write to any Region* mode, a client can unilaterally select its preferred endpoint. If its access to one Region becomes impaired, the client can route to another endpoint.

With *write to one Region* mode, the client needs a mechanism to route its write requests to the currently active Region. This could be a basic mechanism, such as empirically testing which Region is presently accepting write requests (noting any write rejections and falling back to an alternate). Or it can be a complex mechanism, such as using a global coordinator to query for the current application state (perhaps built on the Application Recovery Controller (ARC) (ARC) routing control, which provides a five-Region, quorum-driven system to maintain global state for needs such as this). The client can decide if read requests can go to any Region for eventual consistency or must be routed to the active Region for strong consistency.

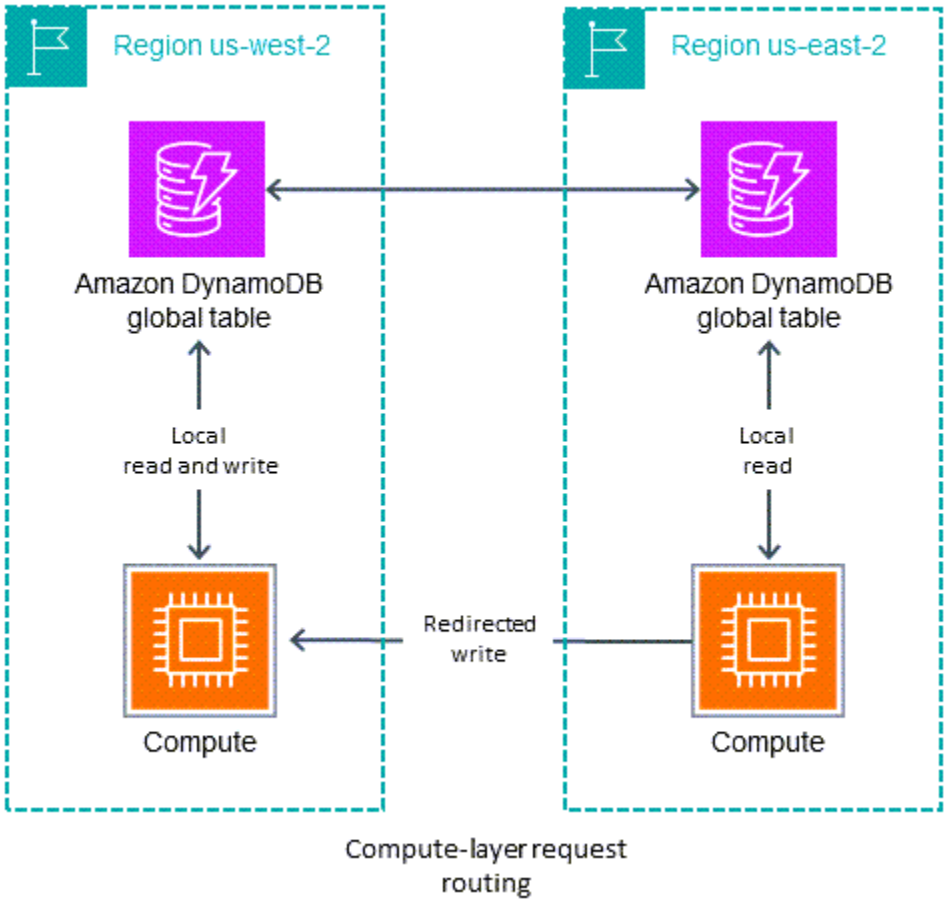
With the *write to your Region* mode, the client needs to determine the home Region for the dataset it's working with. For example, if the client corresponds to a user account and each user account

is homed to a Region, the client can request the appropriate endpoint assignment to use with its credentials from a global login system.

For example, a financial services company that helps users manage their business finances through the web uses global tables with a *write to your Region* mode. Each user must log in to a central service. That service returns credentials as well as the endpoint for the Region where those credentials will work. The Region that's returned is based on where the user's dataset is currently homed. The credentials are valid for a short time. After that, the webpage auto-negotiates a new login, which provides an opportunity to potentially redirect the user's activity to a new Region.

Compute-layer request routing

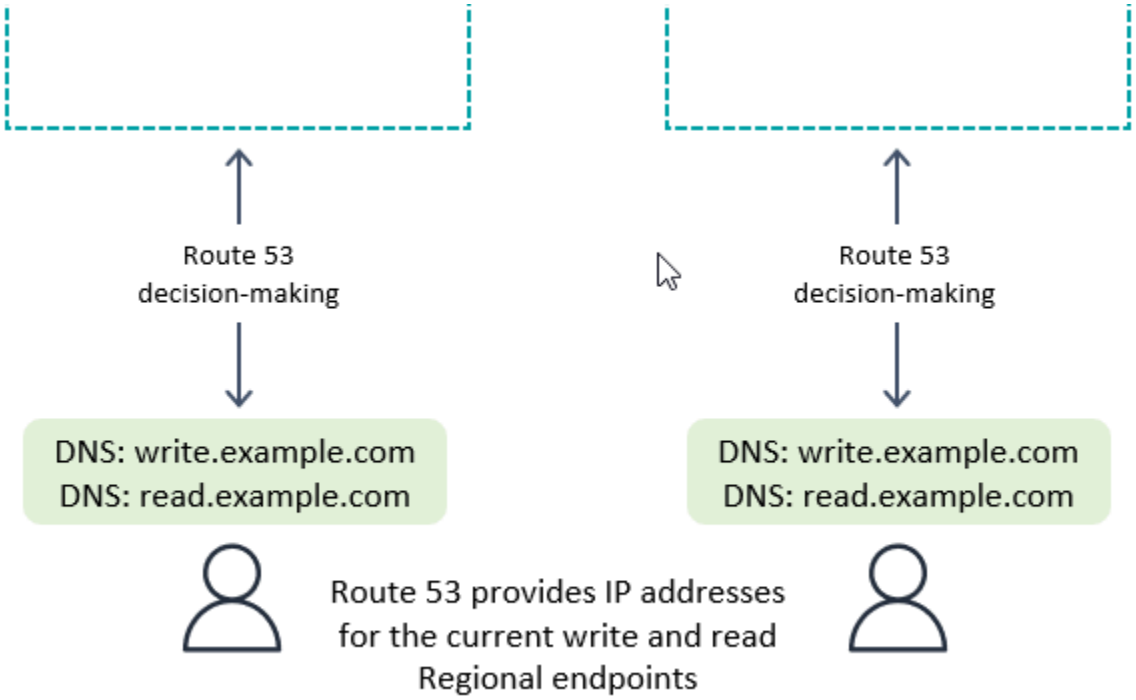
With compute-layer request routing, the code that runs in the compute layer determines whether to process the request locally or pass it to a copy of itself that's running in another Region. When you use the *write to one Region* mode, the compute layer might detect that it's not the active Region and allow local read operations while forwarding all write operations to another Region. This compute layer code must be aware of data topology and routing rules, and enforce them reliably, based on the latest settings that specify which Regions are active for which data. The outer software stack within the Region doesn't have to be aware of how read and write requests are routed by the microservice. In a robust design, the receiving Region validates whether it is the current primary for the write operation. If it isn't, it generates an error that indicates that the global state needs to be corrected. The receiving Region might also buffer the write operation for a while if the primary Region is in the process of changing. In all cases, the compute stack in a Region writes only to its local DynamoDB endpoint, but the compute stacks might communicate with one another.



The Vanguard Group uses a system called Global Orchestration and Status Tool (GOaST) and a library called Global Multi-Region library (GMRlib) for this routing process, as presented at re:Invent 2022. They use a follow-the-sun single primary model. GOaST maintains the global state, similar to the ARC routing control discussed in the previous section. It uses a global table to track which Region is the primary Region and when the next primary switch is scheduled. All read and write operations go through GMRlib, which coordinates with GOaST. GMRlib allows read operations to be performed locally, at low latency. For write operations, GMRlib checks if the local Region is the current primary Region. If so, the write operation completes directly. If not, GMRlib forwards the write task to the GMRlib in the primary Region. That receiving library confirms that it also considers itself the primary Region and raises an error if it isn't, which indicates a propagation delay with the global state. This approach provides a validation benefit by not writing directly to a remote DynamoDB endpoint.

Route 53 request routing

Amazon Route 53 is a Domain Name Service (DNS) technology. With Route 53, the client requests its endpoint by looking up a well-known DNS domain name, and Route 53 returns the IP address that corresponds to the Regional endpoint(s) it determines most appropriate. Route 53 has a long list of routing policies it uses to determine the appropriate Region. It also can do failover routing to route traffic away from Regions that fail health checks.



With *write to any Region* mode, or if combined with the compute-layer request routing on the backend, Route 53 can be given full freedom to return the Region based on any complex internal rules, such as choosing the Region in the closest network or geographic proximity, or any other choice.

With *write to one Region* mode, you can configure Route 53 to return the currently active Region (by using ARC). If the client wants to connect to a passive Region (for example, for read operations), it could look up a different DNS name.

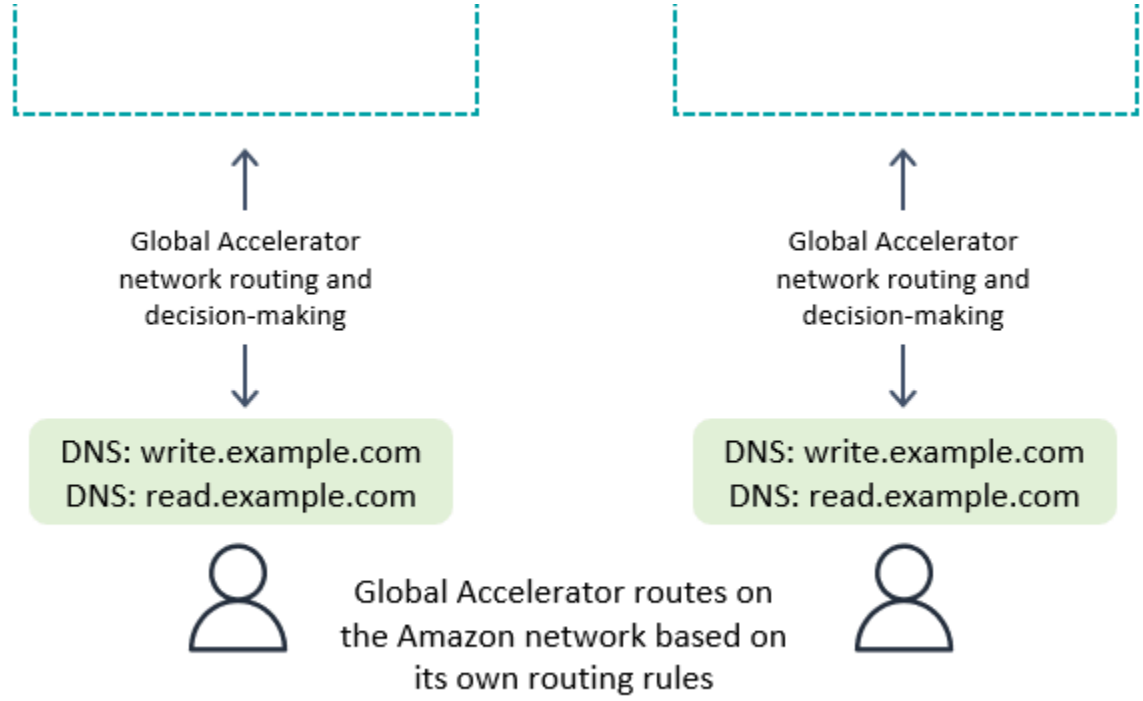
> **Note**
>
> Clients cache the IP addresses in the response from Route 53 for a time indicated by the time to live (TTL) setting on the domain name. A longer TTL extends the recovery time objective (RTO) for all clients to recognize the new endpoint. A value of 60 seconds is

typical for failover use. Not all software perfectly adheres to DNS TTL expiration, and there might be multiple levels of DNS caching, such as at the operating system, virtual machine, and application.

With *write to your Region* mode, it's best to avoid Route 53 unless you're also using compute-layer request routing.

Global Accelerator request routing

With AWS Global Accelerator a client looks up the well-known domain name in Route 53. However, instead of getting back an IP address that corresponds to a Regional endpoint, the client gets back an anycast static IP address that routes to the nearest AWS edge location. Starting from that edge location, all traffic gets routed on the private AWS network to some endpoint (Network Load Balancers, Application Load Balancers, EC2 instances, or elastic IP addresses) in a Region chosen by routing rules that are maintained within Global Accelerator. Compared with routing based on Route 53 rules, Global Accelerator request routing has lower latencies because it reduces the amount of traffic on the public internet. In addition, because Global Accelerator doesn't depend on DNS TTL expiration to change routing rules, it can adjust routing more quickly.



With *write to any Region* mode, or if combined with the compute-layer request routing on the backend, Global Accelerator works seamlessly. The client connects to the nearest edge location and doesn't have to be concerned about which Region receives the request.

With *write to one Region* mode, Global Accelerator routing rules must send requests to the currently active Region. You can use health checks that artificially report a failure on any Region that's not considered by your global system to be the active Region. As with DNS, it's possible to use an alternative DNS domain name for routing read requests, if the requests can be from any Region.

With *write to your Region* mode, it's best to avoid Global Accelerator unless you're also using compute-layer request routing.

Evacuation processes for global tables

Evacuating a Region is the process of migrating activity—usually write activity, possibly read activity—away from that Region.

Evacuating a live Region

You might decide to evacuate a live Region for a number of reasons: as part of usual business activity (for example, if you're using a follow-the-sun, *write to one Region* mode), due to a business decision to change the currently active Region, in response to failures in the software stack outside DynamoDB, or because you're encountering general issues such as higher than usual latencies within the Region.

With *write to any Region* mode, evacuating a live Region is straightforward. You can route traffic to alternative Regions by using any routing system, and let the write operations that have already happened in the evacuated Region replicate over as usual.

With *write to one Region* and *write to your Region* modes, you must make sure that all write operations to the active Region have been fully recorded, stream processed, and globally propagated before starting write operations in the new active Region, to ensure that future write operations are processed against the latest version of the data.

Let's say that Region A is active and Region B is passive (either for the full table or for items that are homed in Region A). The typical mechanism to perform an evacuation is to pause write operations to A, wait long enough for those operations to have fully propagated to B, update the architecture stack to recognize B as active, and then resume write operations to B. There is no metric to indicate with absolute certainty that Region A has fully replicated its data to Region B. If Region A is healthy, pausing write operations to Region A and waiting 10 times the recent maximum value of the `ReplicationLatency` metric would typically be sufficient to determine that replication is complete. If Region A is unhealthy and shows other areas of increased latencies, you would choose a larger multiple for the wait time.

Evacuating an offline Region

There's a special case to consider: What if Region A goes fully offline without notice? This is extremely unlikely but should be considered nevertheless. If this happens, any write operations in

Region A that were not yet propagated are held and propagated after Region A comes back online. The write operations aren't lost, but their propagation is indefinitely delayed.

How to proceed in this event is the application's decision. For business continuity, write operations might need to proceed to the new primary Region B. However, if an item in Region B receives an update while there is a pending propagation of a write operation for that item from Region A, the propagation is suppressed under the *last writer wins* model. Any update in Region B might suppress an incoming write request.

With *write to any Region* mode, read and write operations can continue in Region B, trusting that the items in Region A will propagate to Region B eventually and recognizing the potential for missing items until Region A comes back online. When possible, such as with idempotent write operations, you should consider replaying recent write traffic (for example, by using an upstream event source) to fill in the gap of any potentially missing write operations and let the *last writer wins* conflict resolution suppress the eventual propagation of the incoming write operation.

With the other write modes, you have to consider the degree to which work can continue with a slightly out-of-date view of the world. Some small duration of write operations, as tracked by `ReplicationLatency`, will be missing until Region A comes back online. Can business move forward? In some use cases it can, but in others it might not without additional mitigation mechanisms.

For example, imagine that you have to maintain an available credit balance without interruption even after a full outage of a Region. You could split the balance into two different items, one homed in Region A and one in Region B, and start each with half the available balance. This would use the *write to your Region* mode. Transactional updates processed in each Region would write against the local copy of the balance. If Region A goes fully offline, work could still proceed with transaction processing in Region B, and write operations would be limited to the balance portion held in Region B. Splitting the balance like this introduces complexities when the balance gets low or the credit has to be rebalanced, but it does provide one example of safe business recovery even with uncertain pending write operations.

As another example, imagine that you're capturing web form data. You can use optimistic concurrency control (OCC) to assign versions to data items and embed the latest version into the web form as a hidden field. On each submit, the write operation succeeds only if the version in the database still matches the version that the form was built against. If the versions don't match, the web form can be refreshed (or carefully merged) based on the current version in the database, and the user can proceed again. The OCC model usually protects against another client overwriting and producing a new version of the data, but it can also help during failover where a client might

encounter older versions of data. Let's imagine that you're using the timestamp as the version. The form was first built against Region A at 12:00 but (after failover) tries to write to Region B and notices that the latest version in the database is 11:59. In this scenario, the client can either wait for the 12:00 version to propagate to Region B and then write on top of that version, or build on 11:59 and create a new 12:01 version (which, after writing, would suppress the incoming version after Region A recovers).

As a third example, a financial services company holds data about customer accounts and their financial transactions in a DynamoDB database. In the event of a complete Region A outage, they want to make sure that any write activity related to their accounts is either fully available in Region B, or they want to quarantine their accounts as *known partial* until Region A comes back online. Instead of pausing all business, they decided to pause business only to the tiny fraction of accounts that they determined had unpropagated transactions. To achieve this, they used a third Region, which we will call Region C. Before they processed any write operations in Region A, they placed a succinct summary of those pending operations (for example, a new transaction count for an account) in Region C. This summary was sufficient for Region B to determine if its view was fully up to date. This action effectively locked the account from the time of writing in Region C until Region A accepted the write operations and Region B received them. The data in Region C wasn't used except as part of a failover process, after which Region B could cross-check its data with Region C to check if any of its accounts were out of date. Those accounts would be marked as quarantined until the Region A recovery propagated the partial data to Region B. If Region C were to fail, a new Region D could be spun up for use instead. The data in Region C was very transient, and after a few minutes Region D would have a sufficiently up-to-date record of the in-flight write operations to be fully useful. If Region B were to fail, Region A could continue accepting write requests in cooperation with Region C. This company was willing to accept higher latency writes (to two Regions: C and then A) and was fortunate to have a data model where the state of an account could be succinctly summarized.

Throughput capacity planning for global tables

Migrating traffic from one Region to another requires careful consideration of DynamoDB table settings regarding capacity.

Here are some considerations for managing write capacity:

- A global table must be in on-demand mode or provisioned with auto scaling enabled.
- If provisioned with auto scaling, the write settings (minimum, maximum, and target utilization) are replicated across Regions. Although the auto scaling settings are synchronized, the actual provisioned write capacity might float independently between Regions.
- One reason you might see different provisioned write capacity is due to the time to live (TTL) feature. When you enable TTL in DynamoDB, you can specify an attribute name whose value indicates the time of expiration for the item, in Unix epoch time format in seconds. After that time, DynamoDB can delete the item without incurring write costs. With global tables, you can configure TTL in any Region, and the setting is automatically replicated to other Regions that are associated with the global table. When an item is eligible for deletion through a TTL rule, that work can be done in any Region. The delete operation is performed without consuming write units on the source table, but the replica tables will get a replicated write of that delete operation and **will** incur replicated write unit costs.
- If you're using auto scaling, make sure that the maximum provisioned write capacity setting is sufficiently high to handle all write operations as well as all potential TTL delete operations. Auto scaling adjusts each Region according to its write consumption. On-demand tables have no maximum provisioned write capacity setting, but the *table-level maximum write throughput limit* specifies the maximum sustained write capacity the on-demand table will allow. The default limit to 40,000, but it is adjustable. We recommend that you set it high enough to handle all write operations (including TTL write operations) that the on-demand table might need. This value must be the same across all participating Regions when you set up global tables.

Here are some considerations for managing read capacity:

- Read capacity management settings are allowed to differ between Regions because it's assumed that different Regions might have independent read patterns. When you first add a global replica to a table, the capacity of the source Region is propagated. After creation you can adjust the read capacity settings, which aren't transferred to the other side.

- When you use DynamoDB auto scaling, make sure that the maximum provisioned read capacity settings are sufficiently high to handle all read operations across all Regions. During standard operations the read capacity will perhaps be spread across Regions, but during failover the table should be able to automatically adapt to the increased read workload. On-demand tables have no maximum provisioned read capacity setting, but the *table-level maximum read throughput limit* specifies the maximum sustained read capacity the on-demand table will allow. The default limit is 40,000, but it is adjustable. We recommend that you set it high enough to handle all read operations that the table might need if all read operations were to route to this single Region.
- If a table in one Region doesn't usually receive read traffic but might have to absorb a large amount of read traffic after a failover, you can raise the provisioned read capacity of the table, wait for the table to finish updating, and then provision the table down again. You can either leave the table in provisioned mode or switch it to on-demand mode. This pre-warms the table for accepting a higher level of read traffic.

ARC has readiness checks that can be useful in confirming that DynamoDB Regions have similar table settings and account quotas, whether or not you use Route 53 to route requests. These readiness checks also help you adjust account-level quotas to make them match.

Preparation checklist for global tables

Use the following checklist for decisions and tasks when you deploy global tables.

- Determine how many and which Regions should participate in the global table.
- Determine your application's write mode.
- Plan your routing strategy, based on your write mode.
- Define your evacuation plan, based on your write mode and routing strategy.
- Capture metrics on the health, latency, and errors across each Region. For a list of DynamoDB metrics, see the AWS blog post Monitoring Amazon DynamoDB for operational awareness. You should also use synthetic canaries (artificial requests designed to detect failures) as well as live observation of customer traffic. Not all issues appear in the DynamoDB metrics.
- Set alarms for any sustained increase in `ReplicationLatency`. An increase might indicate an accidental misconfiguration in which the global table has different write settings in different Regions, which leads to failed replicated requests and increased latencies. It could also indicate that there is a Regional disruption. A good example is to generate an alert if the recent average exceeds 180,000 milliseconds. You might also watch for `ReplicationLatency` dropping to 0, which indicates stalled replication.
- Assign sufficient maximum read and write settings for each global table.
- Identify the conditions where you would evacuate a Region. If the decision involves human judgment, document all considerations. This work should be done carefully in advance, not under stress.
- Maintain a runbook for every action that must take place when you evacuate a Region. Usually very little work is involved for the global tables, but moving the rest of the stack might be complex.

> **Note**
>
> With failover procedures, it's best practice to rely only on data plane operations and not on control plane operations, because some control plane operations might be degraded during Region failures. For more information, see the AWS blog post Build resilient applications with Amazon DynamoDB global tables: Part 4.

- Test all aspects of the runbook periodically, including Region evacuations. An untested runbook is an unreliable runbook.

- Consider using AWS Resilience Hub to evaluate the resilience of your entire application (including global tables). This service provides a comprehensive view of the resiliency status of your application portfolio through its dashboard.
- Consider using ARC readiness checks to evaluate the current configuration of your application and track any deviances from best practices.
- When you write health checks for use with Route 53 or Global Accelerator, make a set of calls that cover the full database flow. If you limit your check to confirm only that the DynamoDB endpoint is up, you won't be able to cover many failure modes such as AWS Identity and Access Management (IAM) configuration errors, code deployment problems, failure in the stack outside DynamoDB, higher than average read or write latencies, and so on.

Global tables FAQ

This section provides answers to frequently asked questions about DynamoDB global tables.

What is the pricing for global tables?

- A write operation in a traditional DynamoDB table is priced in write capacity units (WCUs) for provisioned tables or write request units (WRUs) for on-demand tables. If you write a 5 KB item, it incurs a charge of 5 units. A write to a global table is priced in replicated write capacity units (rWCUs) for provisioned tables or replicated write request units (rWRUs) for on-demand tables.
- rWCUs and rWRUs include the cost of the streaming infrastructure needed to manage the replication. As such, they are priced 50 percent higher than WCUs and WRUs. Cross-Region data transfer fees apply.
- rWCU and rWRU charges are incurred in every Region where the item is written directly or written through replication.
- Writing to a global secondary index (GSI) is considered a local write operation and uses regular write units.
- There is no reserved capacity available for rWCUs at this time. Purchasing reserved capacity for WCUs might still be beneficial for tables where GSIs consume write units.
- When you add a new Region to a global table, DynamoDB bootstraps the new Region automatically and charges you as if it were a table restore, based on the GB size of the table. It also charges cross-Region data transfer fees.

Which Regions do global tables support?

Global tables support all AWS Regions.

How are GSIs handled with global tables?

In global tables (current, version 2019), when you create a GSI in one Region, it's automatically created in other participating Regions and automatically backfilled.

How do I stop the replication of a global table?

You can delete a replica table the same way you would delete any other table. Deleting the global table stops replication to that Region and deletes the table copy kept in that Region. However, you cannot stop replication while keeping copies of the table as independent entities, nor can you pause replication.

How do Amazon DynamoDB Streams interact with global tables?

Each global table produces an independent stream based on all its write operations, wherever they started from. You can choose to consume the DynamoDB stream in one Region or in all Regions (independently). If you want to process local but not replicated write operations, you can add your own Region attribute to each item to identify the writing Region. You can then use an AWS Lambda event filter to call the Lambda function only for write operations in the local Region. This helps with insert and update operations, but not delete operations.

How do global tables handle transactions?

Transactional operations provide atomicity, consistency, isolation, durability (ACID) guarantees **only** within the Region where the write operation originally occurred. Transactions are not supported across Regions in global tables. For example, if you have a global table with replicas in the US East (Ohio) and US West (Oregon) Regions and perform a `TransactWriteItems` operation in the US East (Ohio) Region, you might observe partially completed transactions in the US West (Oregon) Region as changes are replicated. Changes are replicated to other Regions only after they have been committed in the source Region.

How do global tables interact with the DynamoDB Accelerator (DAX) cache?

Global tables bypass DAX by updating DynamoDB directly, so DAX isn't aware that it's holding stale data. The DAX cache is refreshed only when the cache's TTL expires.

Do tags on tables propagate?

No, tags do not automatically propagate.

Should I back up tables in all Regions or just one?

The answer depends on the purpose of the backup.

- If you want to ensure data durability, DynamoDB already provides that safeguard. The service ensures durability.
- If you want to keep a snapshot for historical records (for example, to meet regulatory requirements), backing up in one Region should suffice. You can copy the backup to additional Regions by using AWS Backup.
- If you want to recover erroneously deleted or modified data, use DynamoDB point-in-time recovery (PITR) in one Region.

How do I deploy global tables by using AWS CloudFormation?

- CloudFormation represents a DynamoDB table and a global table as two separate resources: `AWS::DynamoDB::Table` and `AWS::DynamoDB::GlobalTable`. One approach is to create all tables that can potentially be global by using the `GlobalTable` construct, keep them as standalone tables initially, and add Regions later, if necessary.
- In CloudFormation, each global table is controlled by a single stack, in a single Region, regardless of the number of replicas. When you deploy your template, CloudFormation creates and updates all replicas as part of a single stack operation. You should not deploy the same AWS::DynamoDB::GlobalTable resource in multiple Regions. This will result in errors and is unsupported. If you deploy your application template in multiple Regions, you can use conditions to create the `AWS::DynamoDB::GlobalTable` resource in a single Region. Alternatively, you can choose to define your `AWS::DynamoDB::GlobalTable` resources in a stack that's separate from your application stack, and make sure that it's deployed to a single Region.
- If you have a regular table and you want to convert it to a global table while keeping it managed by CloudFormation: Set the deletion policy to `Retain`, remove the table from the stack, convert the table to a global table in the console, and then import the global table as a new resource to the stack. For more information, see the AWS GitHub repository amazon-dynamodb-table-to-global-table-cdk.
- Cross-account replication is not supported at this time.

Conclusion and resources

DynamoDB global tables have very few controls but still require careful consideration. You must determine your write mode, routing model, and evacuation processes. You must instrument your application across every Region and be ready to adjust your routing or perform an evacuation to maintain global health. The reward is having a globally distributed dataset with low-latency read and write operations that is designed for 99.999% availability.

For more information about DynamoDB global tables, see the following resources:

- Amazon DynamoDB documentation
- Amazon Route 53 Application Recovery Controller
- ARC readiness checks (AWS documentation)
- Route 53 routing policies (AWS documentation)
- AWS Global Accelerator
- DynamoDB service-level agreement
- AWS Multi-Region Fundamentals (AWS whitepaper)
- Data resiliency design patterns with AWS (AWS re:Invent 2022 presentation)
- How Fidelity Investments and Reltio modernized with Amazon DynamoDB (AWS re:Invent 2022 presentation)
- Multi-Region design patterns and best practices (AWS re:Invent 2022 presentation)
- Disaster Recovery (DR) Architecture on AWS, Part III: Pilot Light and Warm Standby (AWS blog post)
- Use Region pinning to set a home Region for items in an Amazon DynamoDB global table (AWS blog post)
- Monitoring Amazon DynamoDB for operational awareness (AWS blog post)
- Scaling DynamoDB: How partitions, hot keys, and split for heat impact performance (AWS blog post)

Document history

The following table describes significant changes to this guide. If you want to be notified about future updates, you can subscribe to an RSS feed.

Change	Description	Date
Updated AWS Global Accelerator information	Corrected the endpoints for Global Accelerator request routing.	March 14, 2024
Updated AWS Region support information	Updated the FAQ to indicate that global tables now support all AWS Regions.	November 15, 2023
Initial publication	—	May 19, 2023

AWS Prescriptive Guidance glossary

The following are commonly used terms in strategies, guides, and patterns provided by AWS Prescriptive Guidance. To suggest entries, please use the **Provide feedback** link at the end of the glossary.

Numbers

7 Rs

Seven common migration strategies for moving applications to the cloud. These strategies build upon the 5 Rs that Gartner identified in 2011 and consist of the following:

- Refactor/re-architect – Move an application and modify its architecture by taking full advantage of cloud-native features to improve agility, performance, and scalability. This typically involves porting the operating system and database. Example: Migrate your on-premises Oracle database to the Amazon Aurora PostgreSQL-Compatible Edition.
- Replatform (lift and reshape) – Move an application to the cloud, and introduce some level of optimization to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Amazon Relational Database Service (Amazon RDS) for Oracle in the AWS Cloud.
- Repurchase (drop and shop) – Switch to a different product, typically by moving from a traditional license to a SaaS model. Example: Migrate your customer relationship management (CRM) system to Salesforce.com.
- Rehost (lift and shift) – Move an application to the cloud without making any changes to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Oracle on an EC2 instance in the AWS Cloud.
- Relocate (hypervisor-level lift and shift) – Move infrastructure to the cloud without purchasing new hardware, rewriting applications, or modifying your existing operations. You migrate servers from an on-premises platform to a cloud service for the same platform. Example: Migrate a Microsoft Hyper-V application to AWS.
- Retain (revisit) – Keep applications in your source environment. These might include applications that require major refactoring, and you want to postpone that work until a later time, and legacy applications that you want to retain, because there's no business justification for migrating them.

- Retire – Decommission or remove applications that are no longer needed in your source environment.

A

ABAC

See attribute-based access control.

abstracted services

See managed services.

ACID

See atomicity, consistency, isolation, durability.

active-active migration

A database migration method in which the source and target databases are kept in sync (by using a bidirectional replication tool or dual write operations), and both databases handle transactions from connecting applications during migration. This method supports migration in small, controlled batches instead of requiring a one-time cutover. It's more flexible but requires more work than active-passive migration.

active-passive migration

A database migration method in which in which the source and target databases are kept in sync, but only the source database handles transactions from connecting applications while data is replicated to the target database. The target database doesn't accept any transactions during migration.

aggregate function

A SQL function that operates on a group of rows and calculates a single return value for the group. Examples of aggregate functions include `SUM` and `MAX`.

AI

See artificial intelligence.

AIOps

See artificial intelligence operations.

anonymization

The process of permanently deleting personal information in a dataset. Anonymization can help protect personal privacy. Anonymized data is no longer considered to be personal data.

anti-pattern

A frequently used solution for a recurring issue where the solution is counter-productive, ineffective, or less effective than an alternative.

application control

A security approach that allows the use of only approved applications in order to help protect a system from malware.

application portfolio

A collection of detailed information about each application used by an organization, including the cost to build and maintain the application, and its business value. This information is key to the portfolio discovery and analysis process and helps identify and prioritize the applications to be migrated, modernized, and optimized.

artificial intelligence (AI)

The field of computer science that is dedicated to using computing technologies to perform cognitive functions that are typically associated with humans, such as learning, solving problems, and recognizing patterns. For more information, see What is Artificial Intelligence?

artificial intelligence operations (AIOps)

The process of using machine learning techniques to solve operational problems, reduce operational incidents and human intervention, and increase service quality. For more information about how AIOps is used in the AWS migration strategy, see the operations integration guide.

asymmetric encryption

An encryption algorithm that uses a pair of keys, a public key for encryption and a private key for decryption. You can share the public key because it isn't used for decryption, but access to the private key should be highly restricted.

atomicity, consistency, isolation, durability (ACID)

A set of software properties that guarantee the data validity and operational reliability of a database, even in the case of errors, power failures, or other problems.

attribute-based access control (ABAC)

The practice of creating fine-grained permissions based on user attributes, such as department, job role, and team name. For more information, see ABAC for AWS in the AWS Identity and Access Management (IAM) documentation.

authoritative data source

A location where you store the primary version of data, which is considered to be the most reliable source of information. You can copy data from the authoritative data source to other locations for the purposes of processing or modifying the data, such as anonymizing, redacting, or pseudonymizing it.

Availability Zone

A distinct location within an AWS Region that is insulated from failures in other Availability Zones and provides inexpensive, low-latency network connectivity to other Availability Zones in the same Region.

AWS Cloud Adoption Framework (AWS CAF)

A framework of guidelines and best practices from AWS to help organizations develop an efficient and effective plan to move successfully to the cloud. AWS CAF organizes guidance into six focus areas called perspectives: business, people, governance, platform, security, and operations. The business, people, and governance perspectives focus on business skills and processes; the platform, security, and operations perspectives focus on technical skills and processes. For example, the people perspective targets stakeholders who handle human resources (HR), staffing functions, and people management. For this perspective, AWS CAF provides guidance for people development, training, and communications to help ready the organization for successful cloud adoption. For more information, see the AWS CAF website and the AWS CAF whitepaper.

AWS Workload Qualification Framework (AWS WQF)

A tool that evaluates database migration workloads, recommends migration strategies, and provides work estimates. AWS WQF is included with AWS Schema Conversion Tool (AWS SCT). It analyzes database schemas and code objects, application code, dependencies, and performance characteristics, and provides assessment reports.

B

bad bot

A bot that is intended to disrupt or cause harm to individuals or organizations.

BCP

See business continuity planning.

behavior graph

A unified, interactive view of resource behavior and interactions over time. You can use a behavior graph with Amazon Detective to examine failed logon attempts, suspicious API calls, and similar actions. For more information, see Data in a behavior graph in the Detective documentation.

big-endian system

A system that stores the most significant byte first. See also endianness.

binary classification

A process that predicts a binary outcome (one of two possible classes). For example, your ML model might need to predict problems such as “Is this email spam or not spam?" or "Is this product a book or a car?"

bloom filter

A probabilistic, memory-efficient data structure that is used to test whether an element is a member of a set.

blue/green deployment

A deployment strategy where you create two separate but identical environments. You run the current application version in one environment (blue) and the new application version in the other environment (green). This strategy helps you quickly roll back with minimal impact.

bot

A software application that runs automated tasks over the internet and simulates human activity or interaction. Some bots are useful or beneficial, such as web crawlers that index information on the internet. Some other bots, known as *bad bots*, are intended to disrupt or cause harm to individuals or organizations.

botnet

Networks of bots that are infected by malware and are under the control of a single party, known as a *bot herder* or *bot operator*. Botnets are the best-known mechanism to scale bots and their impact.

branch

A contained area of a code repository. The first branch created in a repository is the *main branch*. You can create a new branch from an existing branch, and you can then develop features or fix bugs in the new branch. A branch you create to build a feature is commonly referred to as a *feature branch*. When the feature is ready for release, you merge the feature branch back into the main branch. For more information, see About branches (GitHub documentation).

break-glass access

In exceptional circumstances and through an approved process, a quick means for a user to gain access to an AWS account that they don't typically have permissions to access. For more information, see the Implement break-glass procedures indicator in the AWS Well-Architected guidance.

brownfield strategy

The existing infrastructure in your environment. When adopting a brownfield strategy for a system architecture, you design the architecture around the constraints of the current systems and infrastructure. If you are expanding the existing infrastructure, you might blend brownfield and greenfield strategies.

buffer cache

The memory area where the most frequently accessed data is stored.

business capability

What a business does to generate value (for example, sales, customer service, or marketing). Microservices architectures and development decisions can be driven by business capabilities. For more information, see the Organized around business capabilities section of the Running containerized microservices on AWS whitepaper.

business continuity planning (BCP)

A plan that addresses the potential impact of a disruptive event, such as a large-scale migration, on operations and enables a business to resume operations quickly.

C

CAF

See AWS Cloud Adoption Framework.

canary deployment

The slow and incremental release of a version to end users. When you are confident, you deploy the new version and replace the current version in its entirety.

CCoE

See Cloud Center of Excellence.

CDC

See change data capture.

change data capture (CDC)

The process of tracking changes to a data source, such as a database table, and recording metadata about the change. You can use CDC for various purposes, such as auditing or replicating changes in a target system to maintain synchronization.

chaos engineering

Intentionally introducing failures or disruptive events to test a system's resilience. You can use AWS Fault Injection Service (AWS FIS) to perform experiments that stress your AWS workloads and evaluate their response.

CI/CD

See continuous integration and continuous delivery.

classification

A categorization process that helps generate predictions. ML models for classification problems predict a discrete value. Discrete values are always distinct from one another. For example, a model might need to evaluate whether or not there is a car in an image.

client-side encryption

Encryption of data locally, before the target AWS service receives it.

Cloud Center of Excellence (CCoE)

A multi-disciplinary team that drives cloud adoption efforts across an organization, including developing cloud best practices, mobilizing resources, establishing migration timelines, and leading the organization through large-scale transformations. For more information, see the CCoE posts on the AWS Cloud Enterprise Strategy Blog.

cloud computing

The cloud technology that is typically used for remote data storage and IoT device management. Cloud computing is commonly connected to edge computing technology.

cloud operating model

In an IT organization, the operating model that is used to build, mature, and optimize one or more cloud environments. For more information, see Building your Cloud Operating Model.

cloud stages of adoption

The four phases that organizations typically go through when they migrate to the AWS Cloud:

- Project – Running a few cloud-related projects for proof of concept and learning purposes
- Foundation – Making foundational investments to scale your cloud adoption (e.g., creating a landing zone, defining a CCoE, establishing an operations model)
- Migration – Migrating individual applications
- Re-invention – Optimizing products and services, and innovating in the cloud

These stages were defined by Stephen Orban in the blog post The Journey Toward Cloud-First & the Stages of Adoption on the AWS Cloud Enterprise Strategy blog. For information about how they relate to the AWS migration strategy, see the migration readiness guide.

CMDB

See configuration management database.

code repository

A location where source code and other assets, such as documentation, samples, and scripts, are stored and updated through version control processes. Common cloud repositories include GitHub or AWS CodeCommit. Each version of the code is called a *branch*. In a microservice structure, each repository is devoted to a single piece of functionality. A single CI/CD pipeline can use multiple repositories.

cold cache

A buffer cache that is empty, not well populated, or contains stale or irrelevant data. This affects performance because the database instance must read from the main memory or disk, which is slower than reading from the buffer cache.

cold data

Data that is rarely accessed and is typically historical. When querying this kind of data, slow queries are typically acceptable. Moving this data to lower-performing and less expensive storage tiers or classes can reduce costs.

computer vision (CV)

A field of AI that uses machine learning to analyze and extract information from visual formats such as digital images and videos. For example, AWS Panorama offers devices that add CV to on-premises camera networks, and Amazon SageMaker provides image processing algorithms for CV.

configuration drift

For a workload, a configuration change from the expected state. It might cause the workload to become noncompliant, and it's typically gradual and unintentional.

configuration management database (CMDB)

A repository that stores and manages information about a database and its IT environment, including both hardware and software components and their configurations. You typically use data from a CMDB in the portfolio discovery and analysis stage of migration.

conformance pack

A collection of AWS Config rules and remediation actions that you can assemble to customize your compliance and security checks. You can deploy a conformance pack as a single entity in an AWS account and Region, or across an organization, by using a YAML template. For more information, see Conformance packs in the AWS Config documentation.

continuous integration and continuous delivery (CI/CD)

The process of automating the source, build, test, staging, and production stages of the software release process. CI/CD is commonly described as a pipeline. CI/CD can help you automate processes, improve productivity, improve code quality, and deliver faster. For more information, see Benefits of continuous delivery. CD can also stand for *continuous deployment*. For more information, see Continuous Delivery vs. Continuous Deployment.

CV

See computer vision.

D

data at rest

Data that is stationary in your network, such as data that is in storage.

data classification

A process for identifying and categorizing the data in your network based on its criticality and sensitivity. It is a critical component of any cybersecurity risk management strategy because it helps you determine the appropriate protection and retention controls for the data. Data classification is a component of the security pillar in the AWS Well-Architected Framework. For more information, see Data classification.

data drift

A meaningful variation between the production data and the data that was used to train an ML model, or a meaningful change in the input data over time. Data drift can reduce the overall quality, accuracy, and fairness in ML model predictions.

data in transit

Data that is actively moving through your network, such as between network resources.

data mesh

An architectural framework that provides distributed, decentralized data ownership with centralized management and governance.

data minimization

The principle of collecting and processing only the data that is strictly necessary. Practicing data minimization in the AWS Cloud can reduce privacy risks, costs, and your analytics carbon footprint.

data perimeter

A set of preventive guardrails in your AWS environment that help make sure that only trusted identities are accessing trusted resources from expected networks. For more information, see Building a data perimeter on AWS.

data preprocessing

To transform raw data into a format that is easily parsed by your ML model. Preprocessing data can mean removing certain columns or rows and addressing missing, inconsistent, or duplicate values.

data provenance

The process of tracking the origin and history of data throughout its lifecycle, such as how the data was generated, transmitted, and stored.

data subject

An individual whose data is being collected and processed.

data warehouse

A data management system that supports business intelligence, such as analytics. Data warehouses commonly contain large amounts of historical data, and they are typically used for queries and analysis.

database definition language (DDL)

Statements or commands for creating or modifying the structure of tables and objects in a database.

database manipulation language (DML)

Statements or commands for modifying (inserting, updating, and deleting) information in a database.

DDL

See database definition language.

deep ensemble

To combine multiple deep learning models for prediction. You can use deep ensembles to obtain a more accurate prediction or for estimating uncertainty in predictions.

deep learning

An ML subfield that uses multiple layers of artificial neural networks to identify mapping between input data and target variables of interest.

defense-in-depth

An information security approach in which a series of security mechanisms and controls are thoughtfully layered throughout a computer network to protect the confidentiality, integrity, and availability of the network and the data within. When you adopt this strategy on AWS, you add multiple controls at different layers of the AWS Organizations structure to help secure resources. For example, a defense-in-depth approach might combine multi-factor authentication, network segmentation, and encryption.

delegated administrator

In AWS Organizations, a compatible service can register an AWS member account to administer the organization's accounts and manage permissions for that service. This account is called the *delegated administrator* for that service. For more information and a list of compatible services, see Services that work with AWS Organizations in the AWS Organizations documentation.

deployment

The process of making an application, new features, or code fixes available in the target environment. Deployment involves implementing changes in a code base and then building and running that code base in the application's environments.

development environment

See environment.

detective control

A security control that is designed to detect, log, and alert after an event has occurred. These controls are a second line of defense, alerting you to security events that bypassed the preventative controls in place. For more information, see Detective controls in *Implementing security controls on AWS*.

development value stream mapping (DVSM)

A process used to identify and prioritize constraints that adversely affect speed and quality in a software development lifecycle. DVSM extends the value stream mapping process originally designed for lean manufacturing practices. It focuses on the steps and teams required to create and move value through the software development process.

digital twin

A virtual representation of a real-world system, such as a building, factory, industrial equipment, or production line. Digital twins support predictive maintenance, remote monitoring, and production optimization.

dimension table

In a star schema, a smaller table that contains data attributes about quantitative data in a fact table. Dimension table attributes are typically text fields or discrete numbers that behave like text. These attributes are commonly used for query constraining, filtering, and result set labeling.

disaster

An event that prevents a workload or system from fulfilling its business objectives in its primary deployed location. These events can be natural disasters, technical failures, or the result of human actions, such as unintentional misconfiguration or a malware attack.

disaster recovery (DR)

The strategy and process you use to minimize downtime and data loss caused by a disaster. For more information, see Disaster Recovery of Workloads on AWS: Recovery in the Cloud in the AWS Well-Architected Framework.

DML

See database manipulation language.

domain-driven design

An approach to developing a complex software system by connecting its components to evolving domains, or core business goals, that each component serves. This concept was introduced by Eric Evans in his book, *Domain-Driven Design: Tackling Complexity in the Heart of Software* (Boston: Addison-Wesley Professional, 2003). For information about how you can use domain-driven design with the strangler fig pattern, see Modernizing legacy Microsoft ASP.NET (ASMX) web services incrementally by using containers and Amazon API Gateway.

DR

See disaster recovery.

drift detection

Tracking deviations from a baselined configuration. For example, you can use AWS CloudFormation to detect drift in system resources, or you can use AWS Control Tower to detect changes in your landing zone that might affect compliance with governance requirements.

DVSM

See development value stream mapping.

E

EDA

See exploratory data analysis.

edge computing

The technology that increases the computing power for smart devices at the edges of an IoT network. When compared with cloud computing, edge computing can reduce communication latency and improve response time.

encryption

A computing process that transforms plaintext data, which is human-readable, into ciphertext.

encryption key

A cryptographic string of randomized bits that is generated by an encryption algorithm. Keys can vary in length, and each key is designed to be unpredictable and unique.

endianness

The order in which bytes are stored in computer memory. Big-endian systems store the most significant byte first. Little-endian systems store the least significant byte first.

endpoint

See service endpoint.

endpoint service

A service that you can host in a virtual private cloud (VPC) to share with other users. You can create an endpoint service with AWS PrivateLink and grant permissions to other AWS accounts or to AWS Identity and Access Management (IAM) principals. These accounts or principals can connect to your endpoint service privately by creating interface VPC endpoints. For more information, see Create an endpoint service in the Amazon Virtual Private Cloud (Amazon VPC) documentation.

enterprise resource planning (ERP)

A system that automates and manages key business processes (such as accounting, MES, and project management) for an enterprise.

envelope encryption

The process of encrypting an encryption key with another encryption key. For more information, see Envelope encryption in the AWS Key Management Service (AWS KMS) documentation.

environment

An instance of a running application. The following are common types of environments in cloud computing:

- development environment – An instance of a running application that is available only to the core team responsible for maintaining the application. Development environments are used to test changes before promoting them to upper environments. This type of environment is sometimes referred to as a *test environment*.
- lower environments – All development environments for an application, such as those used for initial builds and tests.
- production environment – An instance of a running application that end users can access. In a CI/CD pipeline, the production environment is the last deployment environment.
- upper environments – All environments that can be accessed by users other than the core development team. This can include a production environment, preproduction environments, and environments for user acceptance testing.

epic

In agile methodologies, functional categories that help organize and prioritize your work. Epics provide a high-level description of requirements and implementation tasks. For example, AWS CAF security epics include identity and access management, detective controls, infrastructure security, data protection, and incident response. For more information about epics in the AWS migration strategy, see the program implementation guide.

ERP

See enterprise resource planning.

exploratory data analysis (EDA)

The process of analyzing a dataset to understand its main characteristics. You collect or aggregate data and then perform initial investigations to find patterns, detect anomalies, and check assumptions. EDA is performed by calculating summary statistics and creating data visualizations.

F

fact table

The central table in a star schema. It stores quantitative data about business operations. Typically, a fact table contains two types of columns: those that contain measures and those that contain a foreign key to a dimension table.

fail fast

A philosophy that uses frequent and incremental testing to reduce the development lifecycle. It is a critical part of an agile approach.

fault isolation boundary

In the AWS Cloud, a boundary such as an Availability Zone, AWS Region, control plane, or data plane that limits the effect of a failure and helps improve the resilience of workloads. For more information, see AWS Fault Isolation Boundaries.

feature branch

See branch.

features

The input data that you use to make a prediction. For example, in a manufacturing context, features could be images that are periodically captured from the manufacturing line.

feature importance

How significant a feature is for a model's predictions. This is usually expressed as a numerical score that can be calculated through various techniques, such as Shapley Additive Explanations (SHAP) and integrated gradients. For more information, see Machine learning model interpretability with :AWS.

feature transformation

To optimize data for the ML process, including enriching data with additional sources, scaling values, or extracting multiple sets of information from a single data field. This enables the ML model to benefit from the data. For example, if you break down the "2021-05-27 00:15:37" date into "2021", "May", "Thu", and "15", you can help the learning algorithm learn nuanced patterns associated with different data components.

FGAC

See fine-grained access control.

fine-grained access control (FGAC)

The use of multiple conditions to allow or deny an access request.

flash-cut migration

A database migration method that uses continuous data replication through change data capture to migrate data in the shortest time possible, instead of using a phased approach. The objective is to keep downtime to a minimum.

G

geo blocking

See geographic restrictions.

geographic restrictions (geo blocking)

In Amazon CloudFront, an option to prevent users in specific countries from accessing content distributions. You can use an allow list or block list to specify approved and banned countries. For more information, see Restricting the geographic distribution of your content in the CloudFront documentation.

Gitflow workflow

An approach in which lower and upper environments use different branches in a source code repository. The Gitflow workflow is considered legacy, and the trunk-based workflow is the modern, preferred approach.

greenfield strategy

The absence of existing infrastructure in a new environment. When adopting a greenfield strategy for a system architecture, you can select all new technologies without the restriction of compatibility with existing infrastructure, also known as brownfield. If you are expanding the existing infrastructure, you might blend brownfield and greenfield strategies.

guardrail

A high-level rule that helps govern resources, policies, and compliance across organizational units (OUs). *Preventive guardrails* enforce policies to ensure alignment to compliance standards. They are implemented by using service control policies and IAM permissions boundaries. *Detective guardrails* detect policy violations and compliance issues, and generate alerts

for remediation. They are implemented by using AWS Config, AWS Security Hub, Amazon GuardDuty, AWS Trusted Advisor, Amazon Inspector, and custom AWS Lambda checks.

H

HA

See high availability.

heterogeneous database migration

Migrating your source database to a target database that uses a different database engine (for example, Oracle to Amazon Aurora). Heterogeneous migration is typically part of a re-architecting effort, and converting the schema can be a complex task. AWS provides AWS SCT that helps with schema conversions.

high availability (HA)

The ability of a workload to operate continuously, without intervention, in the event of challenges or disasters. HA systems are designed to automatically fail over, consistently deliver high-quality performance, and handle different loads and failures with minimal performance impact.

historian modernization

An approach used to modernize and upgrade operational technology (OT) systems to better serve the needs of the manufacturing industry. A *historian* is a type of database that is used to collect and store data from various sources in a factory.

homogeneous database migration

Migrating your source database to a target database that shares the same database engine (for example, Microsoft SQL Server to Amazon RDS for SQL Server). Homogeneous migration is typically part of a rehosting or replatforming effort. You can use native database utilities to migrate the schema.

hot data

Data that is frequently accessed, such as real-time data or recent translational data. This data typically requires a high-performance storage tier or class to provide fast query responses.

hotfix

An urgent fix for a critical issue in a production environment. Due to its urgency, a hotfix is usually made outside of the typical DevOps release workflow.

hypercare period

Immediately following cutover, the period of time when a migration team manages and monitors the migrated applications in the cloud in order to address any issues. Typically, this period is 1–4 days in length. At the end of the hypercare period, the migration team typically transfers responsibility for the applications to the cloud operations team.

I

IaC

See infrastructure as code.

identity-based policy

A policy attached to one or more IAM principals that defines their permissions within the AWS Cloud environment.

idle application

An application that has an average CPU and memory usage between 5 and 20 percent over a period of 90 days. In a migration project, it is common to retire these applications or retain them on premises.

IIoT

See industrial Internet of Things.

immutable infrastructure

A model that deploys new infrastructure for production workloads instead of updating, patching, or modifying the existing infrastructure. Immutable infrastructures are inherently more consistent, reliable, and predictable than mutable infrastructure. For more information, see the Deploy using immutable infrastructure best practice in the AWS Well-Architected Framework.

inbound (ingress) VPC

In an AWS multi-account architecture, a VPC that accepts, inspects, and routes network connections from outside an application. The AWS Security Reference Architecture recommends

setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

incremental migration

A cutover strategy in which you migrate your application in small parts instead of performing a single, full cutover. For example, you might move only a few microservices or users to the new system initially. After you verify that everything is working properly, you can incrementally move additional microservices or users until you can decommission your legacy system. This strategy reduces the risks associated with large migrations.

Industry 4.0

A term that was introduced by Klaus Schwab in 2016 to refer to the modernization of manufacturing processes through advances in connectivity, real-time data, automation, analytics, and AI/ML.

infrastructure

All of the resources and assets contained within an application's environment.

infrastructure as code (IaC)

The process of provisioning and managing an application's infrastructure through a set of configuration files. IaC is designed to help you centralize infrastructure management, standardize resources, and scale quickly so that new environments are repeatable, reliable, and consistent.

industrial Internet of Things (IIoT)

The use of internet-connected sensors and devices in the industrial sectors, such as manufacturing, energy, automotive, healthcare, life sciences, and agriculture. For more information, see Building an industrial Internet of Things (IIoT) digital transformation strategy.

inspection VPC

In an AWS multi-account architecture, a centralized VPC that manages inspections of network traffic between VPCs (in the same or different AWS Regions), the internet, and on-premises networks. The AWS Security Reference Architecture recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

Internet of Things (IoT)

The network of connected physical objects with embedded sensors or processors that communicate with other devices and systems through the internet or over a local communication network. For more information, see What is IoT?

interpretability

A characteristic of a machine learning model that describes the degree to which a human can understand how the model's predictions depend on its inputs. For more information, see Machine learning model interpretability with AWS.

IoT

See Internet of Things.

IT information library (ITIL)

A set of best practices for delivering IT services and aligning these services with business requirements. ITIL provides the foundation for ITSM.

IT service management (ITSM)

Activities associated with designing, implementing, managing, and supporting IT services for an organization. For information about integrating cloud operations with ITSM tools, see the operations integration guide.

ITIL

See IT information library.

ITSM

See IT service management.

L

label-based access control (LBAC)

An implementation of mandatory access control (MAC) where the users and the data itself are each explicitly assigned a security label value. The intersection between the user security label and data security label determines which rows and columns can be seen by the user.

landing zone

A landing zone is a well-architected, multi-account AWS environment that is scalable and secure. This is a starting point from which your organizations can quickly launch and deploy workloads and applications with confidence in their security and infrastructure environment. For more information about landing zones, see Setting up a secure and scalable multi-account AWS environment.

large migration

A migration of 300 or more servers.

LBAC

See label-based access control.

least privilege

The security best practice of granting the minimum permissions required to perform a task. For more information, see Apply least-privilege permissions in the IAM documentation.

lift and shift

See 7 Rs.

little-endian system

A system that stores the least significant byte first. See also endianness.

lower environments

See environment.

M

machine learning (ML)

A type of artificial intelligence that uses algorithms and techniques for pattern recognition and learning. ML analyzes and learns from recorded data, such as Internet of Things (IoT) data, to generate a statistical model based on patterns. For more information, see Machine Learning.

main branch

See branch.

malware

Software that is designed to compromise computer security or privacy. Malware might disrupt computer systems, leak sensitive information, or gain unauthorized access. Examples of malware include viruses, worms, ransomware, Trojan horses, spyware, and keyloggers.

managed services

AWS services for which AWS operates the infrastructure layer, the operating system, and platforms, and you access the endpoints to store and retrieve data. Amazon Simple Storage Service (Amazon S3) and Amazon DynamoDB are examples of managed services. These are also known as *abstracted services*.

manufacturing execution system (MES)

A software system for tracking, monitoring, documenting, and controlling production processes that convert raw materials to finished products on the shop floor.

MAP

See Migration Acceleration Program.

mechanism

A complete process in which you create a tool, drive adoption of the tool, and then inspect the results in order to make adjustments. A mechanism is a cycle that reinforces and improves itself as it operates. For more information, see Building mechanisms in the AWS Well-Architected Framework.

member account

All AWS accounts other than the management account that are part of an organization in AWS Organizations. An account can be a member of only one organization at a time.

MES

See manufacturing execution system.

Message Queuing Telemetry Transport (MQTT)

A lightweight, machine-to-machine (M2M) communication protocol, based on the publish/subscribe pattern, for resource-constrained IoT devices.

microservice

A small, independent service that communicates over well-defined APIs and is typically owned by small, self-contained teams. For example, an insurance system might include

microservices that map to business capabilities, such as sales or marketing, or subdomains, such as purchasing, claims, or analytics. The benefits of microservices include agility, flexible scaling, easy deployment, reusable code, and resilience. For more information, see Integrating microservices by using AWS serverless services.

microservices architecture

An approach to building an application with independent components that run each application process as a microservice. These microservices communicate through a well-defined interface by using lightweight APIs. Each microservice in this architecture can be updated, deployed, and scaled to meet demand for specific functions of an application. For more information, see Implementing microservices on AWS.

Migration Acceleration Program (MAP)

An AWS program that provides consulting support, training, and services to help organizations build a strong operational foundation for moving to the cloud, and to help offset the initial cost of migrations. MAP includes a migration methodology for executing legacy migrations in a methodical way and a set of tools to automate and accelerate common migration scenarios.

migration at scale

The process of moving the majority of the application portfolio to the cloud in waves, with more applications moved at a faster rate in each wave. This phase uses the best practices and lessons learned from the earlier phases to implement a *migration factory* of teams, tools, and processes to streamline the migration of workloads through automation and agile delivery. This is the third phase of the AWS migration strategy.

migration factory

Cross-functional teams that streamline the migration of workloads through automated, agile approaches. Migration factory teams typically include operations, business analysts and owners, migration engineers, developers, and DevOps professionals working in sprints. Between 20 and 50 percent of an enterprise application portfolio consists of repeated patterns that can be optimized by a factory approach. For more information, see the discussion of migration factories and the Cloud Migration Factory guide in this content set.

migration metadata

The information about the application and server that is needed to complete the migration. Each migration pattern requires a different set of migration metadata. Examples of migration metadata include the target subnet, security group, and AWS account.

migration pattern

A repeatable migration task that details the migration strategy, the migration destination, and the migration application or service used. Example: Rehost migration to Amazon EC2 with AWS Application Migration Service.

Migration Portfolio Assessment (MPA)

An online tool that provides information for validating the business case for migrating to the AWS Cloud. MPA provides detailed portfolio assessment (server right-sizing, pricing, TCO comparisons, migration cost analysis) as well as migration planning (application data analysis and data collection, application grouping, migration prioritization, and wave planning). The MPA tool (requires login) is available free of charge to all AWS consultants and APN Partner consultants.

Migration Readiness Assessment (MRA)

The process of gaining insights about an organization's cloud readiness status, identifying strengths and weaknesses, and building an action plan to close identified gaps, using the AWS CAF. For more information, see the migration readiness guide. MRA is the first phase of the AWS migration strategy.

migration strategy

The approach used to migrate a workload to the AWS Cloud. For more information, see the 7 Rs entry in this glossary and see Mobilize your organization to accelerate large-scale migrations.

ML

See machine learning.

modernization

Transforming an outdated (legacy or monolithic) application and its infrastructure into an agile, elastic, and highly available system in the cloud to reduce costs, gain efficiencies, and take advantage of innovations. For more information, see Strategy for modernizing applications in the AWS Cloud.

modernization readiness assessment

An evaluation that helps determine the modernization readiness of an organization's applications; identifies benefits, risks, and dependencies; and determines how well the organization can support the future state of those applications. The outcome of the assessment is a blueprint of the target architecture, a roadmap that details development phases and

milestones for the modernization process, and an action plan for addressing identified gaps. For more information, see Evaluating modernization readiness for applications in the AWS Cloud.

monolithic applications (monoliths)

Applications that run as a single service with tightly coupled processes. Monolithic applications have several drawbacks. If one application feature experiences a spike in demand, the entire architecture must be scaled. Adding or improving a monolithic application's features also becomes more complex when the code base grows. To address these issues, you can use a microservices architecture. For more information, see Decomposing monoliths into microservices.

MPA

See Migration Portfolio Assessment.

MQTT

See Message Queuing Telemetry Transport.

multiclass classification

A process that helps generate predictions for multiple classes (predicting one of more than two outcomes). For example, an ML model might ask "Is this product a book, car, or phone?" or "Which product category is most interesting to this customer?"

mutable infrastructure

A model that updates and modifies the existing infrastructure for production workloads. For improved consistency, reliability, and predictability, the AWS Well-Architected Framework recommends the use of immutable infrastructure as a best practice.

O

OAC

See origin access control.

OAI

See origin access identity.

OCM

See organizational change management.

offline migration

A migration method in which the source workload is taken down during the migration process. This method involves extended downtime and is typically used for small, non-critical workloads.

OI

See operations integration.

OLA

See operational-level agreement.

online migration

A migration method in which the source workload is copied to the target system without being taken offline. Applications that are connected to the workload can continue to function during the migration. This method involves zero to minimal downtime and is typically used for critical production workloads.

OPC-UA

See Open Process Communications - Unified Architecture.

Open Process Communications - Unified Architecture (OPC-UA)

A machine-to-machine (M2M) communication protocol for industrial automation. OPC-UA provides an interoperability standard with data encryption, authentication, and authorization schemes.

operational-level agreement (OLA)

An agreement that clarifies what functional IT groups promise to deliver to each other, to support a service-level agreement (SLA).

operational readiness review (ORR)

A checklist of questions and associated best practices that help you understand, evaluate, prevent, or reduce the scope of incidents and possible failures. For more information, see Operational Readiness Reviews (ORR) in the AWS Well-Architected Framework.

operational technology (OT)

Hardware and software systems that work with the physical environment to control industrial operations, equipment, and infrastructure. In manufacturing, the integration of OT and information technology (IT) systems is a key focus for Industry 4.0 transformations.

operations integration (OI)

The process of modernizing operations in the cloud, which involves readiness planning, automation, and integration. For more information, see the operations integration guide.

organization trail

A trail that's created by AWS CloudTrail that logs all events for all AWS accounts in an organization in AWS Organizations. This trail is created in each AWS account that's part of the organization and tracks the activity in each account. For more information, see Creating a trail for an organization in the CloudTrail documentation.

organizational change management (OCM)

A framework for managing major, disruptive business transformations from a people, culture, and leadership perspective. OCM helps organizations prepare for, and transition to, new systems and strategies by accelerating change adoption, addressing transitional issues, and driving cultural and organizational changes. In the AWS migration strategy, this framework is called *people acceleration*, because of the speed of change required in cloud adoption projects. For more information, see the OCM guide.

origin access control (OAC)

In CloudFront, an enhanced option for restricting access to secure your Amazon Simple Storage Service (Amazon S3) content. OAC supports all S3 buckets in all AWS Regions, server-side encryption with AWS KMS (SSE-KMS), and dynamic PUT and DELETE requests to the S3 bucket.

origin access identity (OAI)

In CloudFront, an option for restricting access to secure your Amazon S3 content. When you use OAI, CloudFront creates a principal that Amazon S3 can authenticate with. Authenticated principals can access content in an S3 bucket only through a specific CloudFront distribution. See also OAC, which provides more granular and enhanced access control.

ORR

See operational readiness review.

OT

See operational technology.

outbound (egress) VPC

In an AWS multi-account architecture, a VPC that handles network connections that are initiated from within an application. The AWS Security Reference Architecture recommends

setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

P

permissions boundary

An IAM management policy that is attached to IAM principals to set the maximum permissions that the user or role can have. For more information, see Permissions boundaries in the IAM documentation.

personally identifiable information (PII)

Information that, when viewed directly or paired with other related data, can be used to reasonably infer the identity of an individual. Examples of PII include names, addresses, and contact information.

PII

See personally identifiable information.

playbook

A set of predefined steps that capture the work associated with migrations, such as delivering core operations functions in the cloud. A playbook can take the form of scripts, automated runbooks, or a summary of processes or steps required to operate your modernized environment.

PLC

See programmable logic controller.

PLM

See product lifecycle management.

policy

An object that can define permissions (see identity-based policy), specify access conditions (see resource-based policy), or define the maximum permissions for all accounts in an organization in AWS Organizations (see service control policy).

polyglot persistence

Independently choosing a microservice's data storage technology based on data access patterns and other requirements. If your microservices have the same data storage technology, they can encounter implementation challenges or experience poor performance. Microservices are more easily implemented and achieve better performance and scalability if they use the data store best adapted to their requirements. For more information, see Enabling data persistence in microservices.

portfolio assessment

A process of discovering, analyzing, and prioritizing the application portfolio in order to plan the migration. For more information, see Evaluating migration readiness.

predicate

A query condition that returns `true` or `false`, commonly located in a `WHERE` clause.

predicate pushdown

A database query optimization technique that filters the data in the query before transfer. This reduces the amount of data that must be retrieved and processed from the relational database, and it improves query performance.

preventative control

A security control that is designed to prevent an event from occurring. These controls are a first line of defense to help prevent unauthorized access or unwanted changes to your network. For more information, see Preventative controls in *Implementing security controls on AWS*.

principal

An entity in AWS that can perform actions and access resources. This entity is typically a root user for an AWS account, an IAM role, or a user. For more information, see *Principal* in Roles terms and concepts in the IAM documentation.

Privacy by Design

An approach in system engineering that takes privacy into account throughout the whole engineering process.

private hosted zones

A container that holds information about how you want Amazon Route 53 to respond to DNS queries for a domain and its subdomains within one or more VPCs. For more information, see Working with private hosted zones in the Route 53 documentation.

proactive control

A security control designed to prevent the deployment of noncompliant resources. These controls scan resources before they are provisioned. If the resource is not compliant with the control, then it isn't provisioned. For more information, see the Controls reference guide in the AWS Control Tower documentation and see Proactive controls in *Implementing security controls on AWS*.

product lifecycle management (PLM)

The management of data and processes for a product throughout its entire lifecycle, from design, development, and launch, through growth and maturity, to decline and removal.

production environment

See environment.

programmable logic controller (PLC)

In manufacturing, a highly reliable, adaptable computer that monitors machines and automates manufacturing processes.

pseudonymization

The process of replacing personal identifiers in a dataset with placeholder values. Pseudonymization can help protect personal privacy. Pseudonymized data is still considered to be personal data.

publish/subscribe (pub/sub)

A pattern that enables asynchronous communications among microservices to improve scalability and responsiveness. For example, in a microservices-based MES, a microservice can publish event messages to a channel that other microservices can subscribe to. The system can add new microservices without changing the publishing service.

Q

query plan

A series of steps, like instructions, that are used to access the data in a SQL relational database system.

query plan regression

When a database service optimizer chooses a less optimal plan than it did before a given change to the database environment. This can be caused by changes to statistics, constraints, environment settings, query parameter bindings, and updates to the database engine.

R

RACI matrix

See responsible, accountable, consulted, informed (RACI).

ransomware

A malicious software that is designed to block access to a computer system or data until a payment is made.

RASCI matrix

See responsible, accountable, consulted, informed (RACI).

RCAC

See row and column access control.

read replica

A copy of a database that's used for read-only purposes. You can route queries to the read replica to reduce the load on your primary database.

re-architect

See 7 Rs.

recovery point objective (RPO)

The maximum acceptable amount of time since the last data recovery point. This determines what is considered an acceptable loss of data between the last recovery point and the interruption of service.

recovery time objective (RTO)

The maximum acceptable delay between the interruption of service and restoration of service.

refactor

See 7 Rs.

Region

A collection of AWS resources in a geographic area. Each AWS Region is isolated and independent of the others to provide fault tolerance, stability, and resilience. For more information, see Specify which AWS Regions your account can use.

regression

An ML technique that predicts a numeric value. For example, to solve the problem of "What price will this house sell for?" an ML model could use a linear regression model to predict a house's sale price based on known facts about the house (for example, the square footage).

rehost

See 7 Rs.

release

In a deployment process, the act of promoting changes to a production environment.

relocate

See 7 Rs.

replatform

See 7 Rs.

repurchase

See 7 Rs.

resiliency

An application's ability to resist or recover from disruptions. High availability and disaster recovery are common considerations when planning for resiliency in the AWS Cloud. For more information, see AWS Cloud Resilience.

resource-based policy

A policy attached to a resource, such as an Amazon S3 bucket, an endpoint, or an encryption key. This type of policy specifies which principals are allowed access, supported actions, and any other conditions that must be met.

responsible, accountable, consulted, informed (RACI) matrix

A matrix that defines the roles and responsibilities for all parties involved in migration activities and cloud operations. The matrix name is derived from the responsibility types defined in the

matrix: responsible (R), accountable (A), consulted (C), and informed (I). The support (S) type is optional. If you include support, the matrix is called a *RASCI matrix*, and if you exclude it, it's called a *RACI matrix*.

responsive control

A security control that is designed to drive remediation of adverse events or deviations from your security baseline. For more information, see Responsive controls in *Implementing security controls on AWS*.

retain

See 7 Rs.

retire

See 7 Rs.

rotation

The process of periodically updating a secret to make it more difficult for an attacker to access the credentials.

row and column access control (RCAC)

The use of basic, flexible SQL expressions that have defined access rules. RCAC consists of row permissions and column masks.

RPO

See recovery point objective.

RTO

See recovery time objective.

runbook

A set of manual or automated procedures required to perform a specific task. These are typically built to streamline repetitive operations or procedures with high error rates.

S

SAML 2.0

An open standard that many identity providers (IdPs) use. This feature enables federated single sign-on (SSO), so users can log into the AWS Management Console or call the AWS API

operations without you having to create user in IAM for everyone in your organization. For more information about SAML 2.0-based federation, see About SAML 2.0-based federation in the IAM documentation.

SCADA

See supervisory control and data acquisition.

SCP

See service control policy.

secret

In AWS Secrets Manager, confidential or restricted information, such as a password or user credentials, that you store in encrypted form. It consists of the secret value and its metadata. The secret value can be binary, a single string, or multiple strings. For more information, see What's in a Secrets Manager secret? in the Secrets Manager documentation.

security control

A technical or administrative guardrail that prevents, detects, or reduces the ability of a threat actor to exploit a security vulnerability. There are four primary types of security controls: preventative, detective, responsive, and proactive.

security hardening

The process of reducing the attack surface to make it more resistant to attacks. This can include actions such as removing resources that are no longer needed, implementing the security best practice of granting least privilege, or deactivating unnecessary features in configuration files.

security information and event management (SIEM) system

Tools and services that combine security information management (SIM) and security event management (SEM) systems. A SIEM system collects, monitors, and analyzes data from servers, networks, devices, and other sources to detect threats and security breaches, and to generate alerts.

security response automation

A predefined and programmed action that is designed to automatically respond to or remediate a security event. These automations serve as detective or responsive security controls that help you implement AWS security best practices. Examples of automated response actions include modifying a VPC security group, patching an Amazon EC2 instance, or rotating credentials.

server-side encryption

Encryption of data at its destination, by the AWS service that receives it.

service control policy (SCP)

A policy that provides centralized control over permissions for all accounts in an organization in AWS Organizations. SCPs define guardrails or set limits on actions that an administrator can delegate to users or roles. You can use SCPs as allow lists or deny lists, to specify which services or actions are permitted or prohibited. For more information, see Service control policies in the AWS Organizations documentation.

service endpoint

The URL of the entry point for an AWS service. You can use the endpoint to connect programmatically to the target service. For more information, see AWS service endpoints in *AWS General Reference*.

service-level agreement (SLA)

An agreement that clarifies what an IT team promises to deliver to their customers, such as service uptime and performance.

service-level indicator (SLI)

A measurement of a performance aspect of a service, such as its error rate, availability, or throughput.

service-level objective (SLO)

A target metric that represents the health of a service, as measured by a service-level indicator.

shared responsibility model

A model describing the responsibility you share with AWS for cloud security and compliance. AWS is responsible for security *of* the cloud, whereas you are responsible for security *in* the cloud. For more information, see Shared responsibility model.

SIEM

See security information and event management system.

single point of failure (SPOF)

A failure in a single, critical component of an application that can disrupt the system.

SLA

See service-level agreement.

SLI

See service-level indicator.

SLO

See service-level objective.

split-and-seed model

A pattern for scaling and accelerating modernization projects. As new features and product releases are defined, the core team splits up to create new product teams. This helps scale your organization's capabilities and services, improves developer productivity, and supports rapid innovation. For more information, see Phased approach to modernizing applications in the AWS Cloud.

SPOF

See single point of failure.

star schema

A database organizational structure that uses one large fact table to store transactional or measured data and uses one or more smaller dimensional tables to store data attributes. This structure is designed for use in a data warehouse or for business intelligence purposes.

strangler fig pattern

An approach to modernizing monolithic systems by incrementally rewriting and replacing system functionality until the legacy system can be decommissioned. This pattern uses the analogy of a fig vine that grows into an established tree and eventually overcomes and replaces its host. The pattern was introduced by Martin Fowler as a way to manage risk when rewriting monolithic systems. For an example of how to apply this pattern, see Modernizing legacy Microsoft ASP.NET (ASMX) web services incrementally by using containers and Amazon API Gateway.

subnet

A range of IP addresses in your VPC. A subnet must reside in a single Availability Zone.

supervisory control and data acquisition (SCADA)

In manufacturing, a system that uses hardware and software to monitor physical assets and production operations.

symmetric encryption

An encryption algorithm that uses the same key to encrypt and decrypt the data.

synthetic testing

Testing a system in a way that simulates user interactions to detect potential issues or to monitor performance. You can use Amazon CloudWatch Synthetics to create these tests.

T

tags

Key-value pairs that act as metadata for organizing your AWS resources. Tags can help you manage, identify, organize, search for, and filter resources. For more information, see Tagging your AWS resources.

target variable

The value that you are trying to predict in supervised ML. This is also referred to as an *outcome variable*. For example, in a manufacturing setting the target variable could be a product defect.

task list

A tool that is used to track progress through a runbook. A task list contains an overview of the runbook and a list of general tasks to be completed. For each general task, it includes the estimated amount of time required, the owner, and the progress.

test environment

See environment.

training

To provide data for your ML model to learn from. The training data must contain the correct answer. The learning algorithm finds patterns in the training data that map the input data attributes to the target (the answer that you want to predict). It outputs an ML model that captures these patterns. You can then use the ML model to make predictions on new data for which you don't know the target.

transit gateway

A network transit hub that you can use to interconnect your VPCs and on-premises networks. For more information, see What is a transit gateway in the AWS Transit Gateway documentation.

trunk-based workflow

An approach in which developers build and test features locally in a feature branch and then merge those changes into the main branch. The main branch is then built to the development, preproduction, and production environments, sequentially.

trusted access

Granting permissions to a service that you specify to perform tasks in your organization in AWS Organizations and in its accounts on your behalf. The trusted service creates a service-linked role in each account, when that role is needed, to perform management tasks for you. For more information, see Using AWS Organizations with other AWS services in the AWS Organizations documentation.

tuning

To change aspects of your training process to improve the ML model's accuracy. For example, you can train the ML model by generating a labeling set, adding labels, and then repeating these steps several times under different settings to optimize the model.

two-pizza team

A small DevOps team that you can feed with two pizzas. A two-pizza team size ensures the best possible opportunity for collaboration in software development.

U

uncertainty

A concept that refers to imprecise, incomplete, or unknown information that can undermine the reliability of predictive ML models. There are two types of uncertainty: *Epistemic uncertainty* is caused by limited, incomplete data, whereas *aleatoric uncertainty* is caused by the noise and randomness inherent in the data. For more information, see the Quantifying uncertainty in deep learning systems guide.

undifferentiated tasks

Also known as *heavy lifting*, work that is necessary to create and operate an application but that doesn't provide direct value to the end user or provide competitive advantage. Examples of undifferentiated tasks include procurement, maintenance, and capacity planning.

upper environments

See environment.

V

vacuuming

A database maintenance operation that involves cleaning up after incremental updates to reclaim storage and improve performance.

version control

Processes and tools that track changes, such as changes to source code in a repository.

VPC peering

A connection between two VPCs that allows you to route traffic by using private IP addresses. For more information, see What is VPC peering in the Amazon VPC documentation.

vulnerability

A software or hardware flaw that compromises the security of the system.

W

warm cache

A buffer cache that contains current, relevant data that is frequently accessed. The database instance can read from the buffer cache, which is faster than reading from the main memory or disk.

warm data

Data that is infrequently accessed. When querying this kind of data, moderately slow queries are typically acceptable.

window function

A SQL function that performs a calculation on a group of rows that relate in some way to the current record. Window functions are useful for processing tasks, such as calculating a moving average or accessing the value of rows based on the relative position of the current row.

workload

A collection of resources and code that delivers business value, such as a customer-facing application or backend process.

workstream

Functional groups in a migration project that are responsible for a specific set of tasks. Each workstream is independent but supports the other workstreams in the project. For example, the portfolio workstream is responsible for prioritizing applications, wave planning, and collecting migration metadata. The portfolio workstream delivers these assets to the migration workstream, which then migrates the servers and applications.

WORM

See write once, read many.

WQF

See AWS Workload Qualification Framework.

write once, read many (WORM)

A storage model that writes data a single time and prevents the data from being deleted or modified. Authorized users can read the data as many times as needed, but they cannot change it. This data storage infrastructure is considered immutable.

Z

zero-day exploit

An attack, typically malware, that takes advantage of a zero-day vulnerability.

zero-day vulnerability

An unmitigated flaw or vulnerability in a production system. Threat actors can use this type of vulnerability to attack the system. Developers frequently become aware of the vulnerability as a result of the attack.

zombie application

An application that has an average CPU and memory usage below 5 percent. In a migration project, it is common to retire these applications.