aws

Architecture Diagrams

Data and Analytics Reference Architecture Diagrams



Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Data and Analytics Reference Architecture Diagrams: Architecture Diagrams

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

Alfresco on AWS 1
Alfresco Digital Business Platform on AWS diagram 1
Further reading 2
Automated archival for Amazon Redshift 3
Automated archival for Amazon Redshift diagram 3
Further reading 4
Change data capture using AWS Database Migration Service 5
Migrate and replicate 5
Streaming 6
Amazon S3 with ETL for upsert 7
Further reading 7
Data access governance 9
Unified data access governance in a data mesh diagram 9
Further reading 10
Data lake GraphQL API 11
Expose your data lake as a GraphQL API diagram 11
Further reading 12
Data lake inserts and upserts 13
Managing inserts and upserts in a serverless data lake diagram 13
Further reading 14
Data mesh with Amazon DataZone 15
Data mesh with Amazon DataZone diagram 15
Further reading 16
Database DevOps on AWS 17
Development workflow 17
Production workflow 18
Further reading 19
DynamoDB global replication 21
DynamoDB global replication with data localization diagram 21
Further reading 22
Migration from MongoDB to Amazon DocumentDB 23
Offline approach 23
Online approach 24
Hybrid approach 25

Further reading ... 26
Modern data platform using AWS and Snowflake ... 27
Modern data platform using AWS and Snowflake diagram ... 27
Further reading ... 28
Modern data platform using AWS and Teradata ... 29
Modern data platform using AWS and Teradata diagram ... 29
Further reading ... 30
Multi-Region GraphQL ... 31
Multi-Region GraphQL API with CloudFront diagram ... 31
Further reading ... 32
OpenSearch trending queries ... 33
Amazon OpenSearch Service trending queries with AWS Glue and Amazon Bedrock diagram ... 33
Further reading ... 34
Operational analytics pipeline ... 35
Operational analytics pipeline on AWS diagram ... 35
Further reading ... 36
Oracle migration to AWS ... 37
Oracle data workload migration to AWS Cloud diagram ... 37
Further reading ... 38
Paginated search with purpose-built databases ... 39
Paginated search with purpose-built databases diagram ... 39
Further reading ... 40
SAS Grid on AWS ... 41
Running SAS Grid on AWS diagram ... 41
Further reading ... 42
Search-backed applications ... 43
Search-backed applications diagram ... 43
Further reading ... 44
Synchronous data mesh ... 45
Synchronous data mesh for GraphQL queries diagram ... 45
Further reading ... 46
TiDB backup and restore ... 47
Fast backup and restore of TiDB on AWS diagram ... 47
Further reading ... 48
TiDB with PrivateLink ... 49

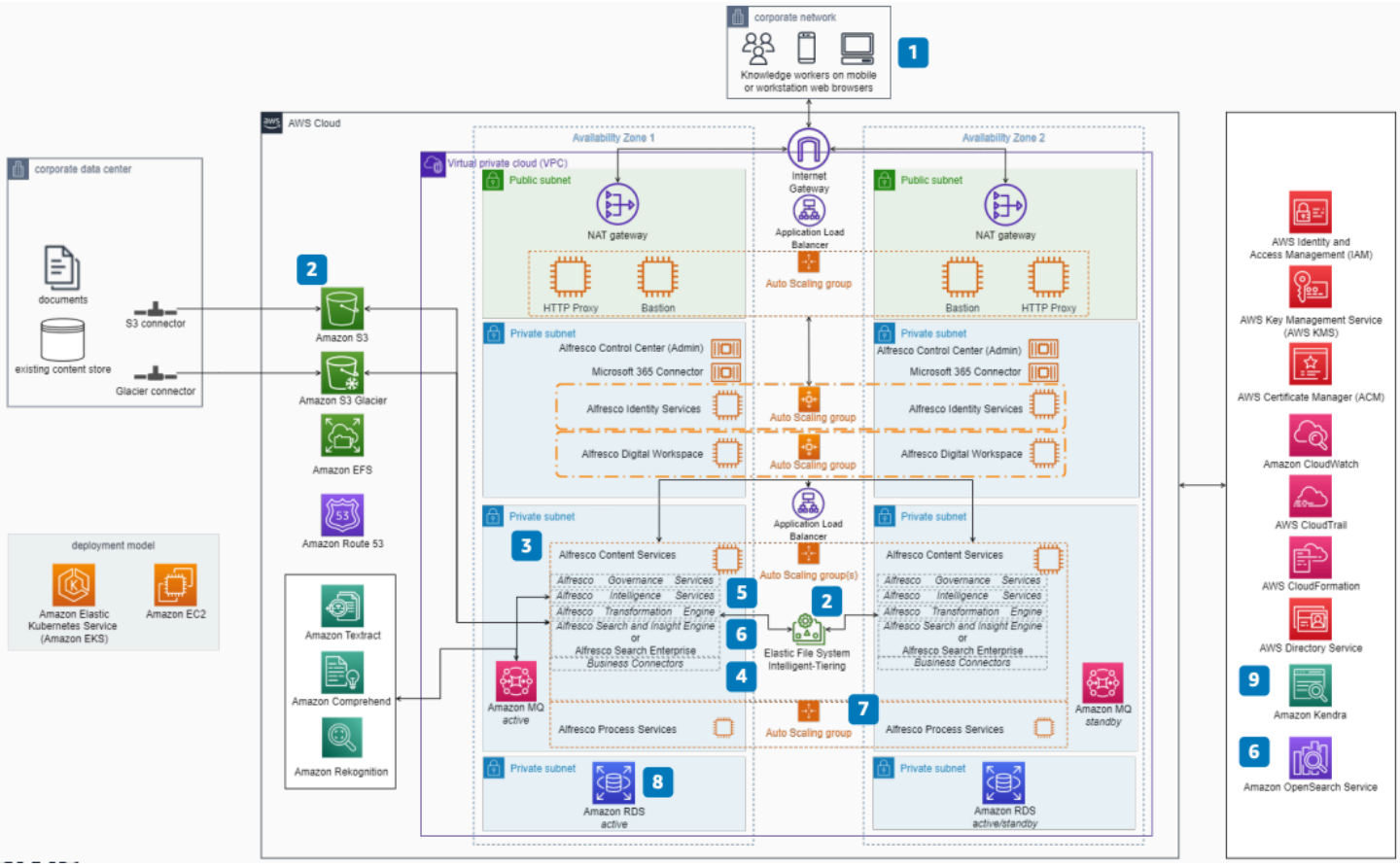
Securely access TiDB using AWS PrivateLink diagram 49
Further reading 50
Transportation and logistics data lake 51
Transportation and logistics data lake diagram 51
Further reading 52
Unified catalog with DataZone 53
Building a unified catalog with Amazon DataZone diagram 53
Further reading 54
Diagram history 55

Alfresco Digital Business Platform on AWS

Publication date: **March 1, 2023 (Diagram history)**

With this architecture, you can deploy a highly available Alfresco Digital Business Platform on AWS. The platform offers Alfresco Content Services for enterprise content management (ECM) and Alfresco Process Services for business process management (BPM). You also get add-on components for search, information governance, rendition, and machine learning (ML).

Alfresco Digital Business Platform on AWS diagram



The following steps describe the architecture:

1. Users connect to applications through web browsers on a workstation or mobile device.
2. Alfresco Content Connector provides the extended content store in Amazon Simple Storage Service or Amazon S3 Glacier. Amazon Elastic File System Intelligent-Tiering is an option.
3. Alfresco Content Services (ACS) provides ECM capabilities with information governance.

4. Business connectors provide integrations with Google Docs, Salesforce, SAP, SAP Cloud, Microsoft Teams, Outlook, and Office.
5. Alfresco Intelligence Services is an add-on module to the Alfresco Transform engine. It integrates with Amazon Textract, Amazon Rekognition, and Amazon Comprehend for metadata extraction and image and text analysis.
6. Alfresco Search and Insight Engine uses Solr on Amazon Elastic Compute Cloud and Amazon Elastic Block Store to provide full-text search across metadata stored in the database. Alfresco Search Enterprise uses Amazon OpenSearch Service as an alternative.
7. Alfresco Process Services is a BPM offering that can create, publish, and use business operations. It supports Amazon Aurora and Amazon Relational Database Service (MySQL, Oracle, PostgreSQL, and Microsoft SQL Server).
8. Use the Amazon Kendra Alfresco data source connector for intelligent enterprise searches.

Further reading

For additional information, see the following resources:

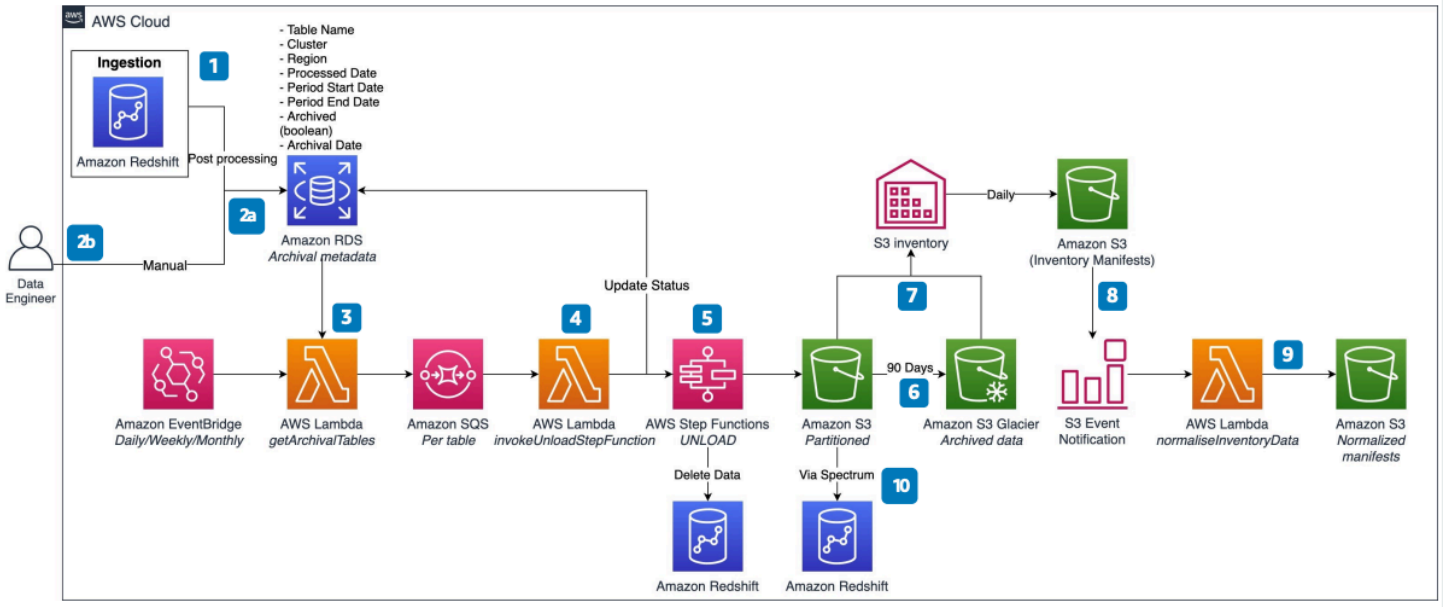
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected

Automated archival for Amazon Redshift

Publication date: **July 12, 2021 (Diagram history)**

This architecture shows how to automate the periodic data archival process for an Amazon Redshift database. The cold data remains available instantly, and you can join it with existing datasets in the Amazon Redshift cluster using Amazon Redshift Spectrum.

Automated archival for Amazon Redshift diagram



The following steps describe the architecture:

1. Ingest data into the Amazon Redshift cluster at various frequencies.
2. After every ingestion load, create a queue of metadata about tables populated into Amazon Redshift tables in Amazon RDS. Data engineers can also create the archival queue manually.
3. Use Amazon EventBridge to trigger an AWS Lambda function periodically. The function reads the queue from the RDS table and creates an Amazon Simple Queue Service message for every period due for archival.
4. A proxy Lambda function dequeues the Amazon SQS messages and invokes AWS Step Functions for every message.
5. Step Functions unloads the data from the Amazon Redshift cluster into an Amazon Simple Storage Service bucket for the given table and period.

6. Amazon S3 Lifecycle configuration moves data in buckets from S3 Standard storage class to S3 Glacier storage class after 90 days.
7. Amazon S3 inventory tool generates manifest files from the Amazon S3 bucket dedicated for cold data on a daily basis and stores them in an Amazon S3 bucket for manifest files.
8. Every time an inventory manifest file is created in a manifest Amazon S3 bucket, a Lambda function triggers through an Amazon S3 event notification.
9. A Lambda function normalizes the manifest file for easy consumption in the event of restore.
10. Query the data stored in the Amazon S3 bucket for cold data using Amazon Redshift Spectrum.

Further reading

For additional information, refer to the following resources:

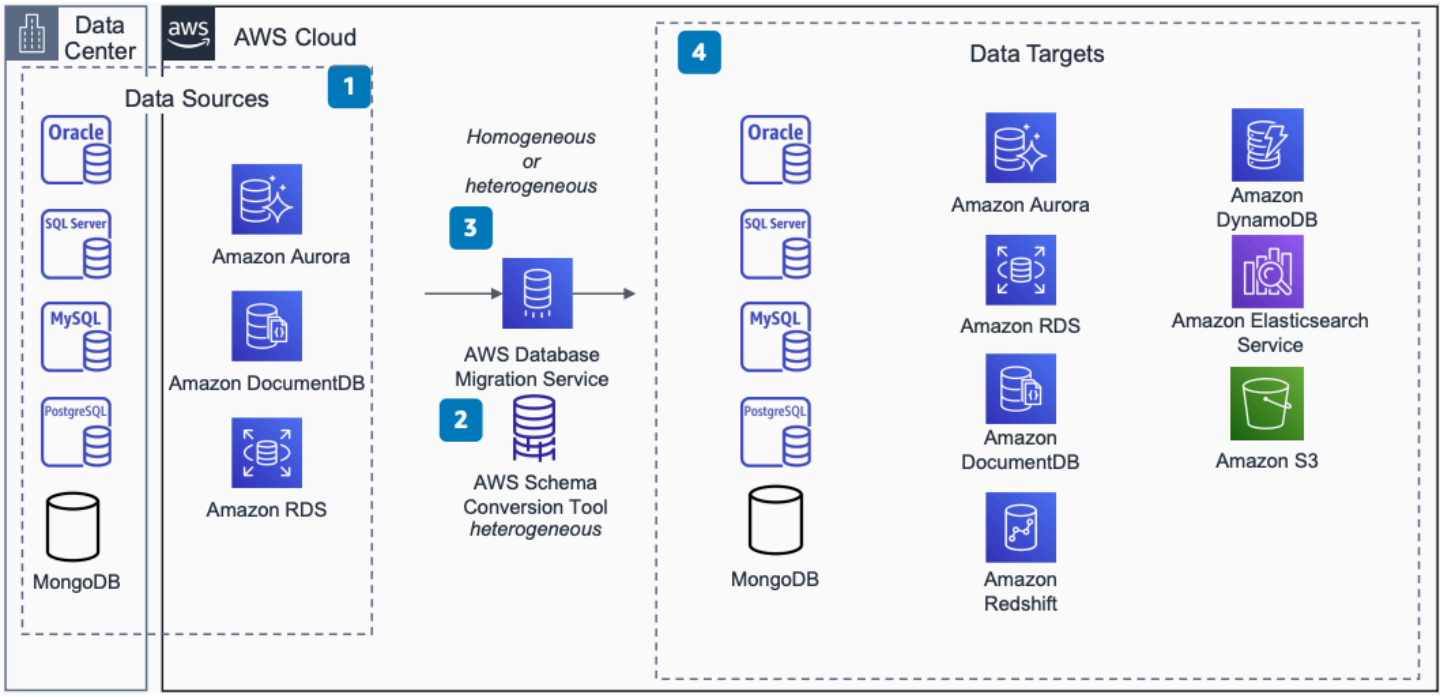
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected

Change data capture using AWS Database Migration Service

Publication date: **August 2, 2021 (Diagram history)**

This architecture shows how to use AWS Database Migration Service (AWS DMS) for change data capture (CDC) to migrate databases using ongoing replication.

Migrate and replicate



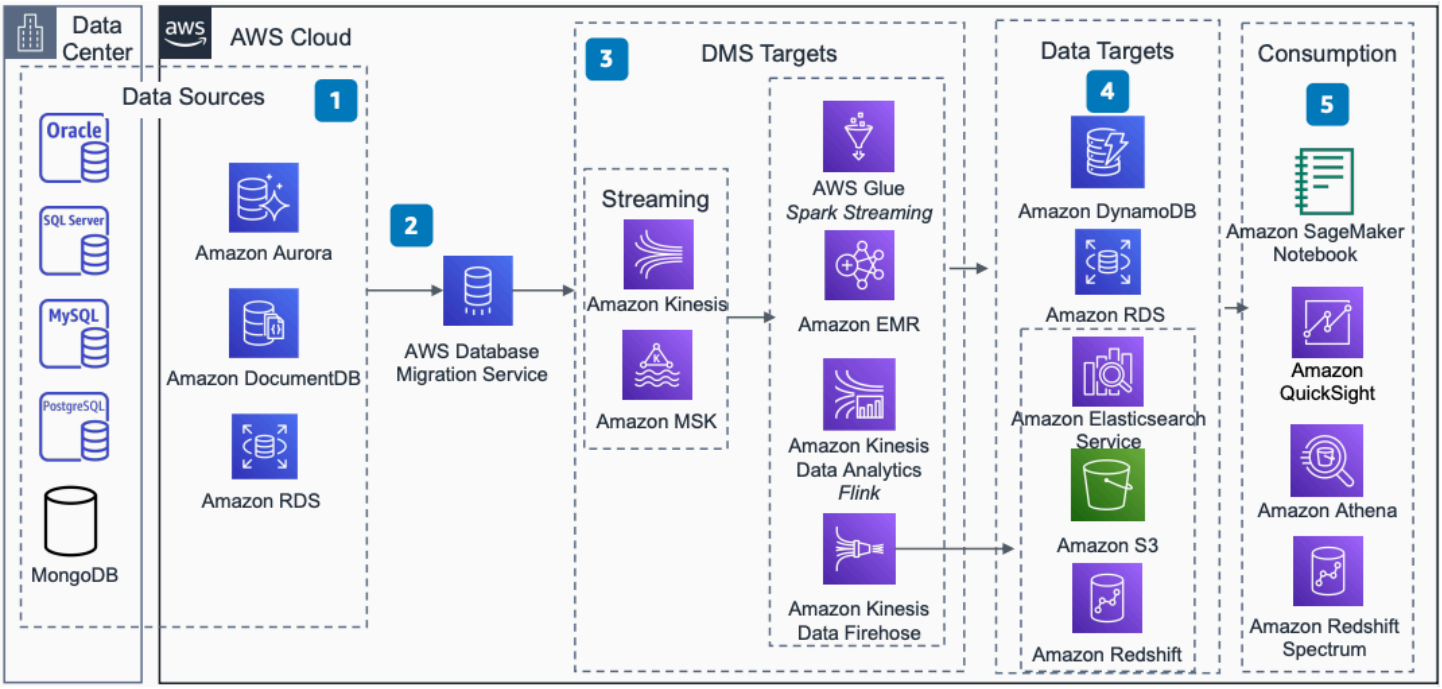
The following steps describe the architecture:

1. Sources for CDC include Amazon Aurora, Amazon DocumentDB, Amazon RDS, MongoDB, MySQL, Oracle, PostgreSQL, and SQL Server.
2. AWS Schema Conversion Tool makes heterogeneous database migrations predictable by automatically converting the source database schema and a majority of code objects to a format compatible with the target database.
3. AWS DMS helps you with one-time data migration of databases and continuous data replication. AWS DMS captures changes on the source database and applies them in a transactionally consistent way to the target.

4. Targets for CDC include Oracle, SQL Server, MySQL, PostgreSQL, MongoDB, Amazon Aurora, Amazon DocumentDB, Amazon RDS, and Amazon Simple Storage Service.

Streaming

This architecture shows how to use AWS Database Migration Service (AWS DMS) to ingest real-time changes from source databases through streaming services.

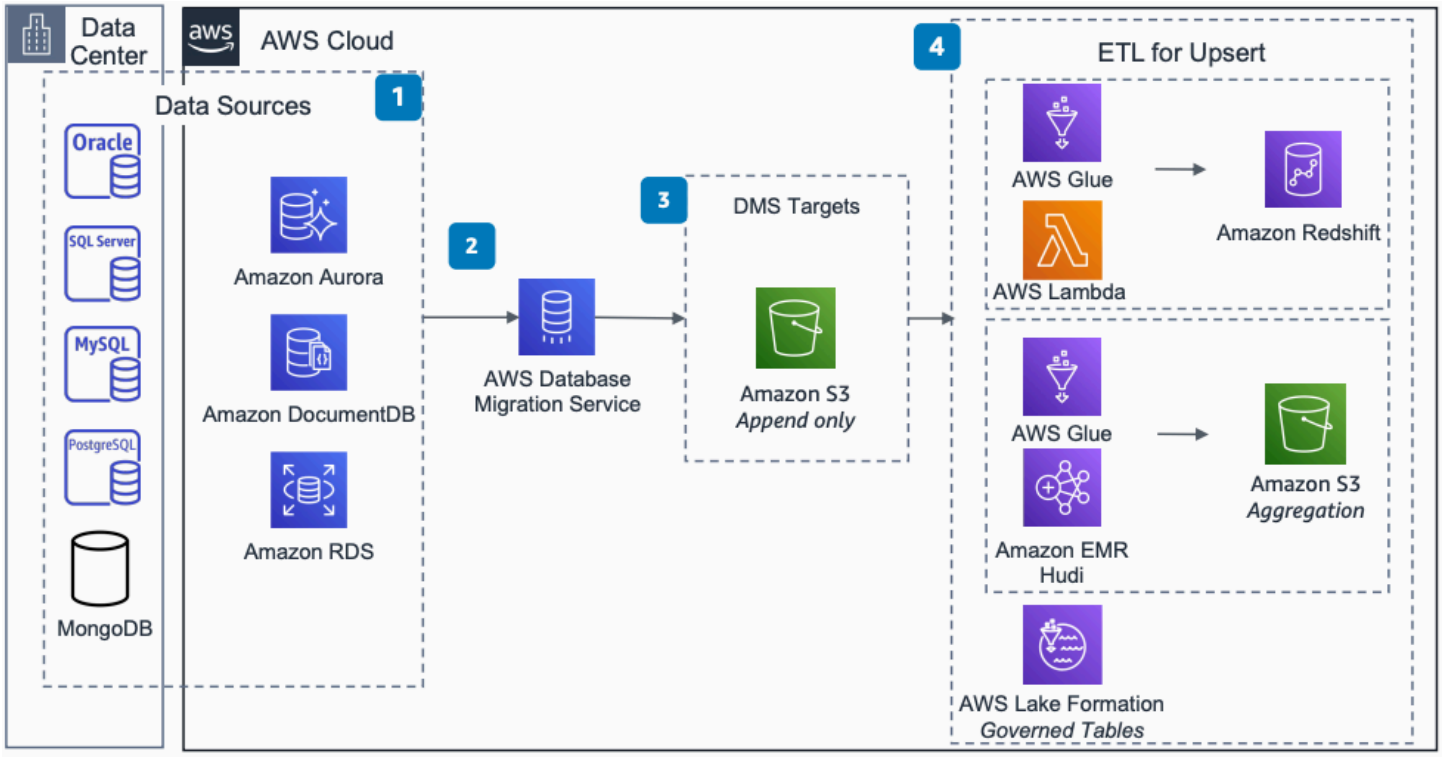


The following steps describe the architecture:

1. Sources for CDC include Oracle, SQL Server, MySQL, PostgreSQL, MongoDB, Amazon Aurora, Amazon DocumentDB, and Amazon RDS.
2. AWS DMS helps you with one-time data migration of databases and continuous data replication. AWS DMS captures changes on the source database and applies them in a transactionally consistent way to the target.
3. Use AWS DMS to stream data to an Amazon Kinesis data stream and Amazon Managed Streaming for Apache Kafka (Amazon MSK) to collect and process large streams of data.
4. Use purpose-built services for analytics in your data lake or data warehouse.
5. Visualize and consume data using AWS Quick Sight and SageMaker AI Notebooks.

Amazon S3 with ETL for upsert

This architecture shows how to use AWS Database Migration Service (AWS DMS) to create data pipelines with change data capture to build transactional data lakes.



The following steps describe the architecture:

1. Sources for CDC include Oracle, SQL Server, MySQL, PostgreSQL, MongoDB, Amazon Aurora, Amazon DocumentDB, and Amazon RDS.
2. AWS DMS helps you with one-time data migration of databases and continuous data replication. AWS DMS captures changes on the source database and applies them in a transactionally consistent way to the target.
3. The target for change data capture is Amazon S3.
4. Use AWS Glue or Amazon EMR for extract, transform, load (ETL) upsert to Amazon S3 and Amazon Redshift.

Further reading

For additional information, refer to the following resources:

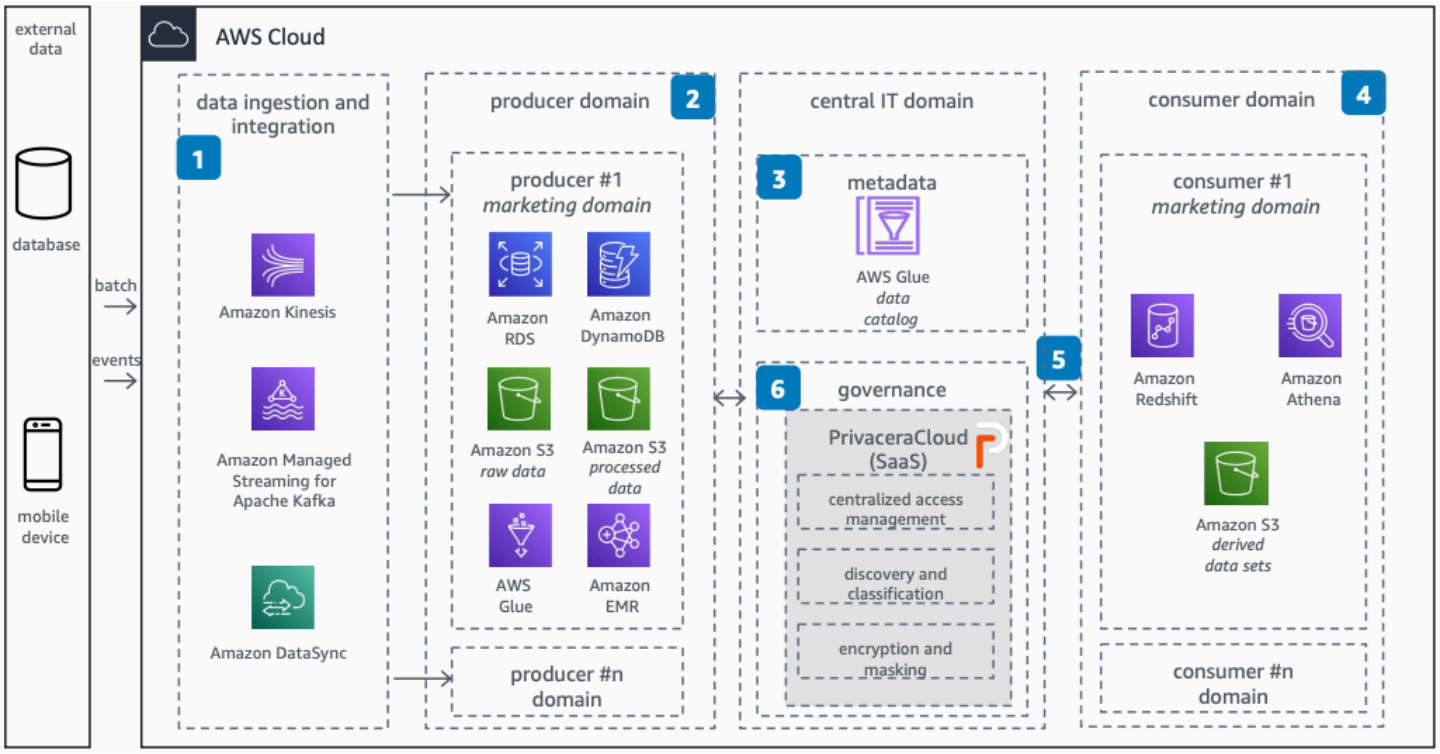
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected

Unified data access governance in a data mesh

Publication date: **February 13, 2023 (Diagram history)**

PrivaceraCloud provides federated computation governance for consistent data access control, governed data sharing, and compliance across AWS services and external on-premises data sources.

Unified data access governance in a data mesh diagram



The following steps describe the architecture:

1. Data flows into AWS through batch processing, real-time data, or change data capture (CDC).
2. The producer domain manages data sources, storing raw data in Amazon S3 and processing it using Amazon EMR, AWS Glue, DynamoDB, and Amazon RDS.
3. AWS Glue and other services manage metadata, and AWS Glue catalogs data for discovery and governance.
4. The consumer domain uses processed data sets from multiple producer domains based on business needs. Consumers build a data mart using Amazon Redshift and Athena.
5. Consumers search and filter data sets recorded in the central data catalog by name, contents, sensitivity, or custom labels.

6. PrivaceraCloud delivers unified data access governance as a fully managed SaaS solution. Built on the attribute-based access control (ABAC) policy model of Apache Ranger, it applies governance to Amazon S3, AWS Glue, Amazon EMR, Amazon RDS, DynamoDB, Athena, and Amazon Redshift.

Further reading

For additional information, refer to the following resources:

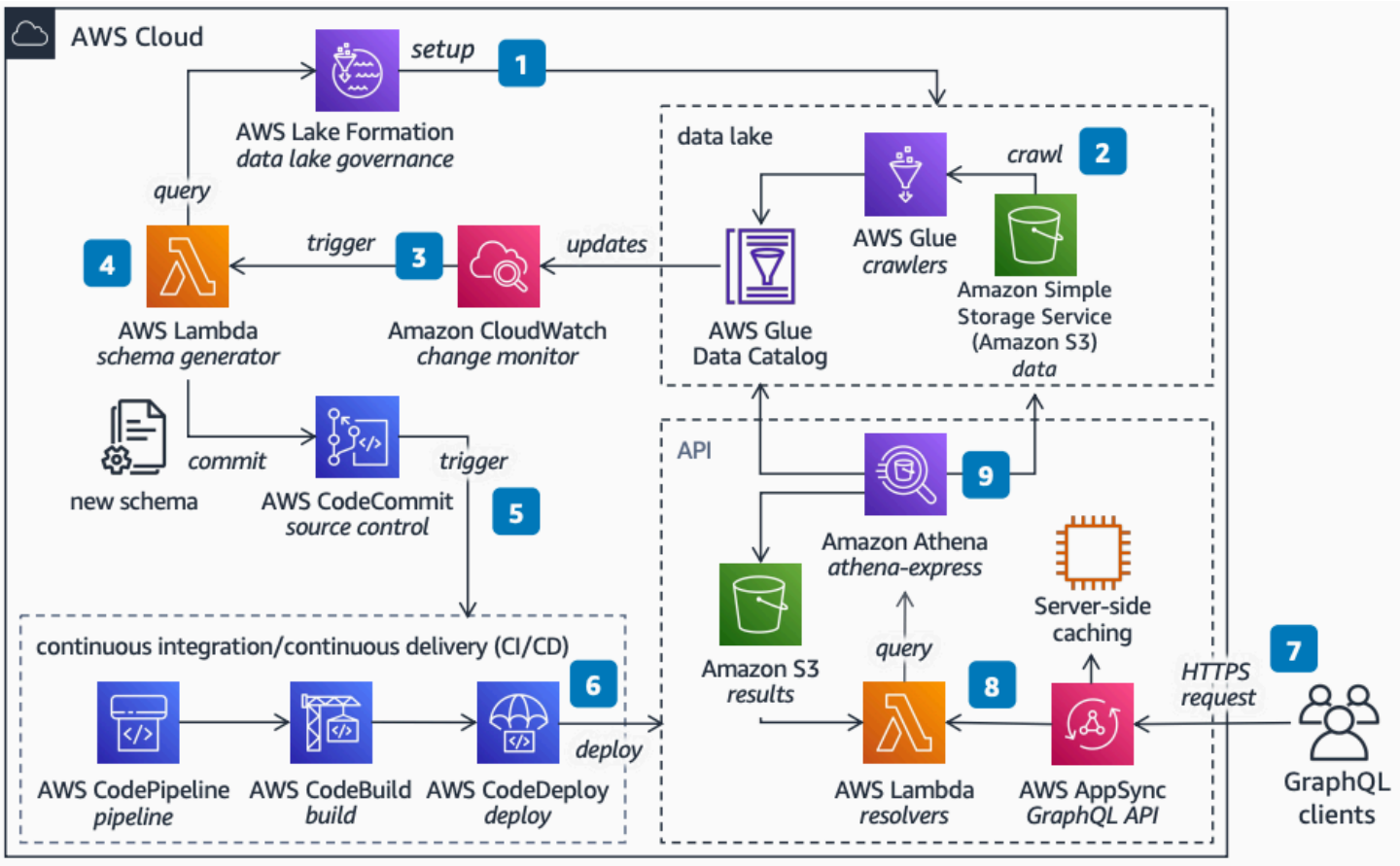
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected

Expose your data lake as a GraphQL API

Publication date: **April 25, 2022 (Diagram history)**

This architecture shows how to build a data lake using Lake Formation and share it as a self-documented GraphQL API. You can automatically build the API with a reactive CI/CD pipeline, and allow authorized external clients to query your data lake using GraphQL syntax over HTTPS.

Expose your data lake as a GraphQL API diagram



The following steps describe the architecture:

1. Set up a data lake using Lake Formation and Amazon Simple Storage Service.
2. Crawl the data in the Amazon S3 buckets with AWS Glue to build a data catalog.
3. Monitor any changes in the AWS Glue Data Catalog with Amazon CloudWatch, and trigger an AWS Lambda function on every catalog change to update the schema of the external API.

4. In the Lambda function, query the latest data catalog and security settings, automatically generate source code to build and deploy a new version of the external API's schema and resolvers, and commit the code to AWS CodeCommit.
5. Set up CodeCommit to automatically start a new CI/CD pipeline on AWS CodePipeline after any code commit.
6. Use CodePipeline to build a new version of the API with AWS CodeBuild using the newly generated source code, and use AWS CodeDeploy to deploy both the new AWS AppSync GraphQL API and the new Lambda query resolvers.
7. Allow your GraphQL clients to securely invoke the external API over HTTPS with your chosen authentication mechanism. Activate the AppSync server-side caching.
8. Use the AppSync Lambda query resolvers to translate the GraphQL query into SQL queries supported by Athena.
9. Run Athena jobs synchronously with the athena-express library, appropriately timing out and limiting the result set.

Further reading

For additional information, refer to the following resources:

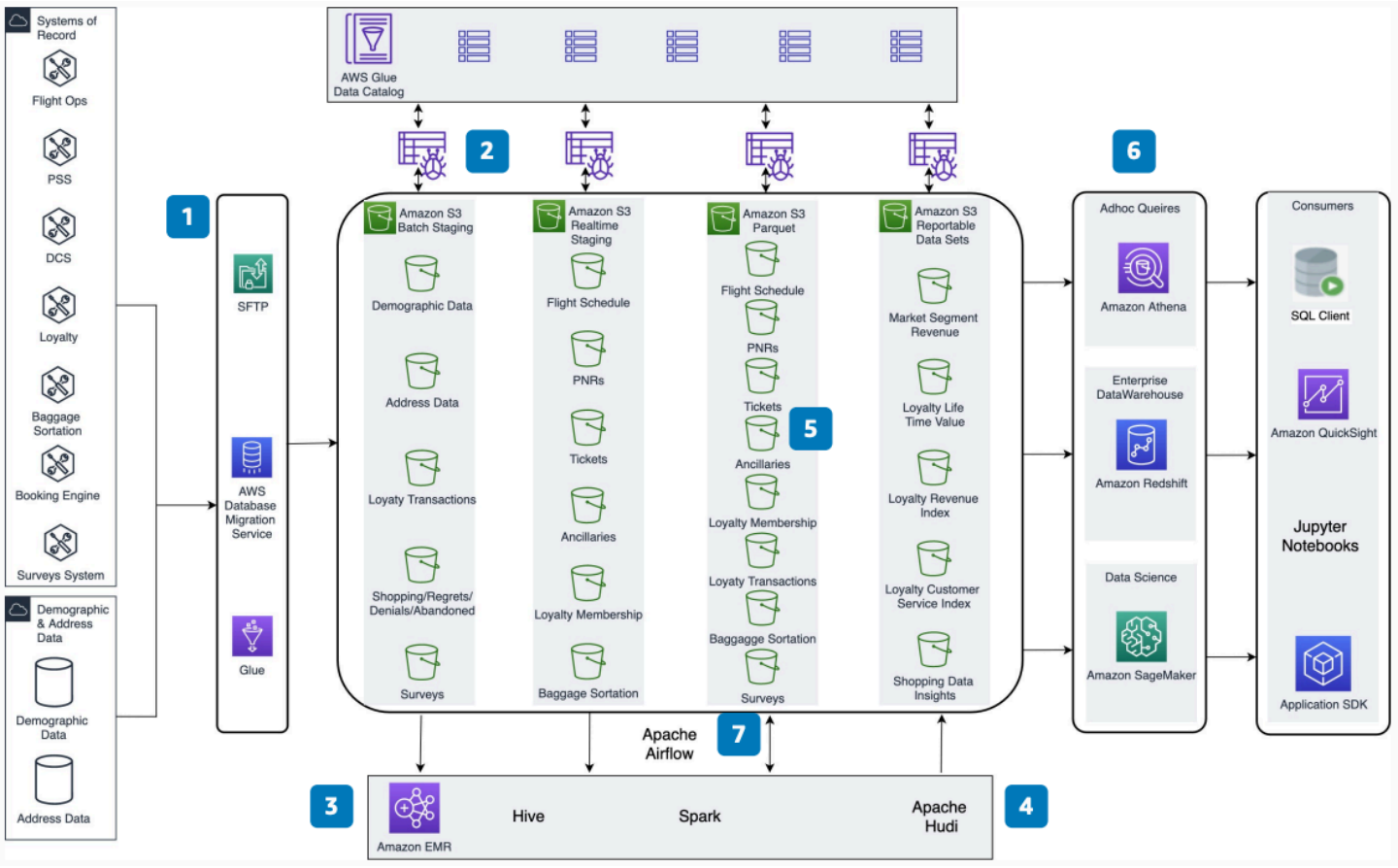
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected

Managing inserts and upserts in a serverless data lake

Publication date: **June 15, 2021 (Diagram history)**

This architecture shows how to use Apache Hudi running on Amazon EMR to process inserts and updates to datasets in Amazon Simple Storage Service. You can build a cost-effective and scalable data lake that provisions on-demand analytics and creates persistent data marts.

Managing inserts and upserts in a serverless data lake diagram



The following steps describe the architecture:

1. Ingest data from source systems using batch, change data capture (CDC), or streaming into the raw layer in Amazon S3.
2. After the data persists in the raw data lake on Amazon S3, crawl the data and populate it in the AWS Glue Data Catalog using a crawler.

3. Pull raw data into an Amazon EMR cluster and read it using Hive and Spark for cleaning and transformation.
4. Apache Hudi running on Amazon EMR reads the data using Spark APIs and performs inserts and upserts on the required datasets.
5. Persist the cleaned and transformed data back into the Amazon S3 processed and reportable buckets.
6. Consume the reportable data on demand using Amazon Athena or load it into Amazon Redshift. Different users, tools, and resources can consume this data.
7. Handle the complete data movement, spin on-demand Amazon EMR clusters for batch data, and load the data using workflow orchestration with Amazon Managed Workflows for Apache Airflow (MWAA).

Further reading

For additional information, refer to the following resources:

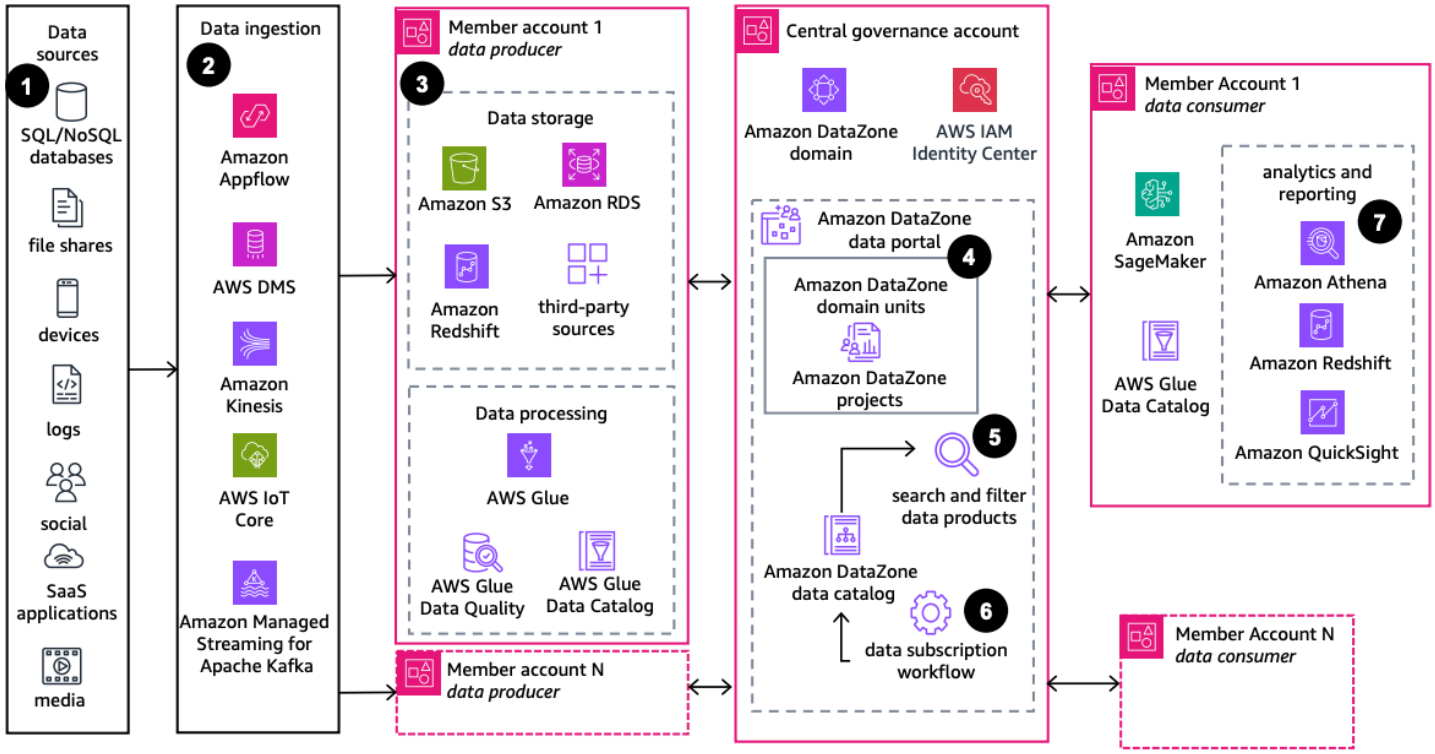
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected

Data mesh with Amazon DataZone

Publication date: **November 20, 2024 (Diagram history)**

Data mesh is a decentralized architectural and organizational framework that helps organizations accelerate innovation and drive business value. This architecture shows how to use Amazon DataZone to build a data mesh-based data solution. In this architecture, a producer account ingests and transforms raw data, then registers it as assets in the Amazon DataZone catalog. A central governance account hosts the Amazon DataZone domain and data portal, where consumers discover assets and request access through subscriptions. After the asset owner approves a subscription request, consumers access the data in the consumer account for analytics and machine learning.

Data mesh with Amazon DataZone diagram



The following steps describe the architecture:

1. Gather data from sources across the enterprise through databases, file shares, edge devices, logs, social networks, SaaS applications, and streaming media.

2. Based on the source system and end user requirements, ingest raw data using Amazon AppFlow, AWS Database Migration Service, Amazon Kinesis, AWS IoT Core, and Amazon Managed Streaming for Apache Kafka.
3. In the producer account, transform raw data using AWS Glue. Store metadata in AWS Glue Data Catalog, measure data quality using AWS Glue Data Quality, and register data in Amazon S3, Amazon Redshift, Amazon RDS, and third-party sources as assets in the Amazon DataZone catalog using data source jobs.
4. The central governance account hosts the Amazon DataZone domain and the related data portal. Associate AWS accounts of data producers and consumers with the Amazon DataZone domain, and create projects under related domain units.
5. End users log in to the Amazon DataZone data portal using IAM credentials or single sign-on (SSO) through https://docs.aws.amazon.com/singlesignon/latest/userguide/. They search, filter, and view asset information including data quality, business metadata, and technical metadata.
6. When end users find assets of interest, they request access using the subscription feature of Amazon DataZone. The asset owner approves or rejects the request based on validity.
7. After the subscription request is granted and fulfilled, access the asset in the consumer account for AI/ML model development using SageMaker AI, and for analytics and reporting use Athena, Amazon Redshift, and AWS Quick Sight.

Further reading

For additional information, refer to the following resources:

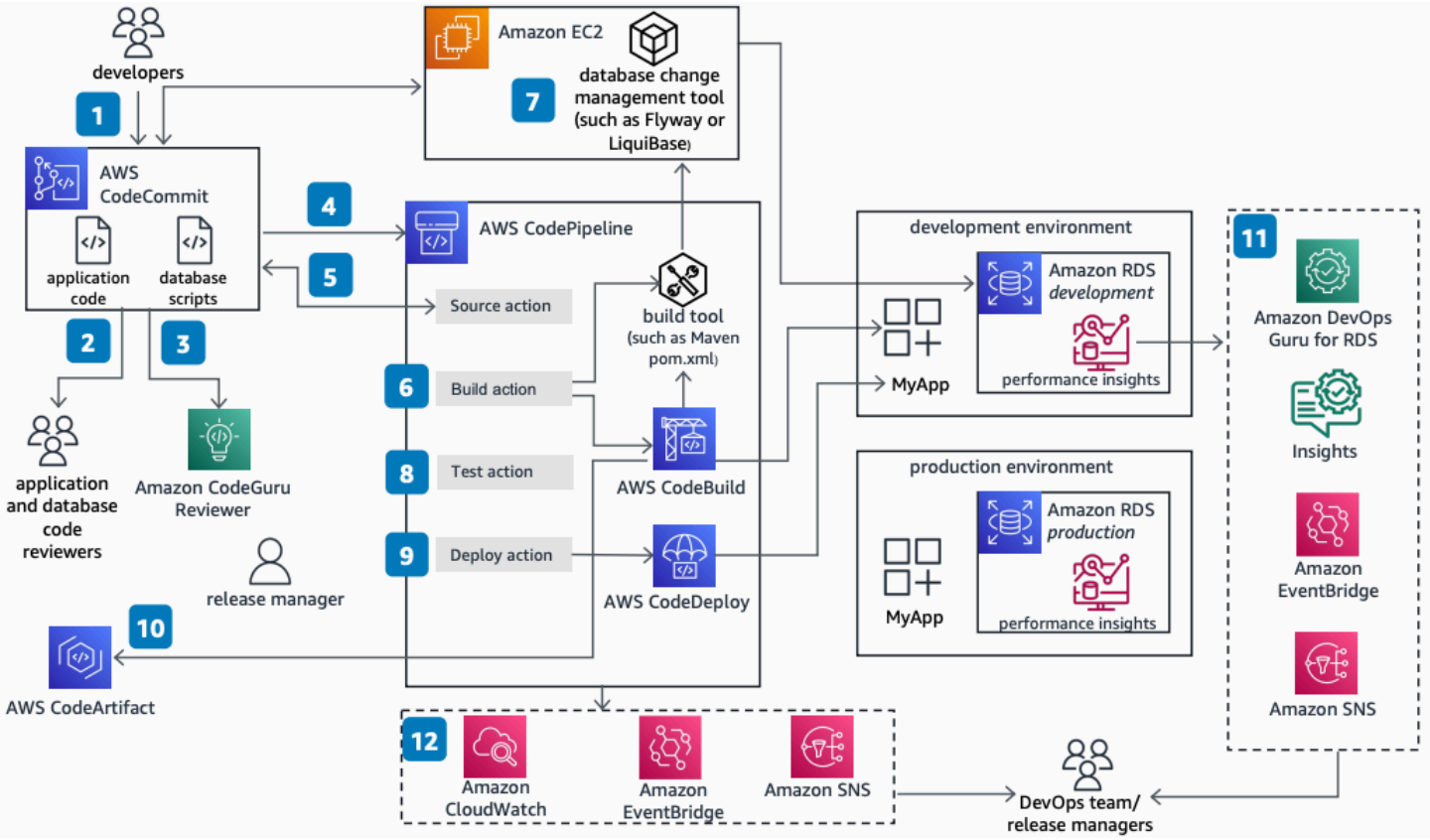
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected

Database DevOps on AWS

Publication date: **March 31, 2023 (Diagram history)**

Use this architecture in the development workflow to automate database changes as part of DevOps practice. Build an orchestration to deploy database changes at the same rate as application code, enforce database-application-joint code reviews, integrate database and application code operations, detect abnormal system behavior, and automate notifications to appointed teams.

Development workflow



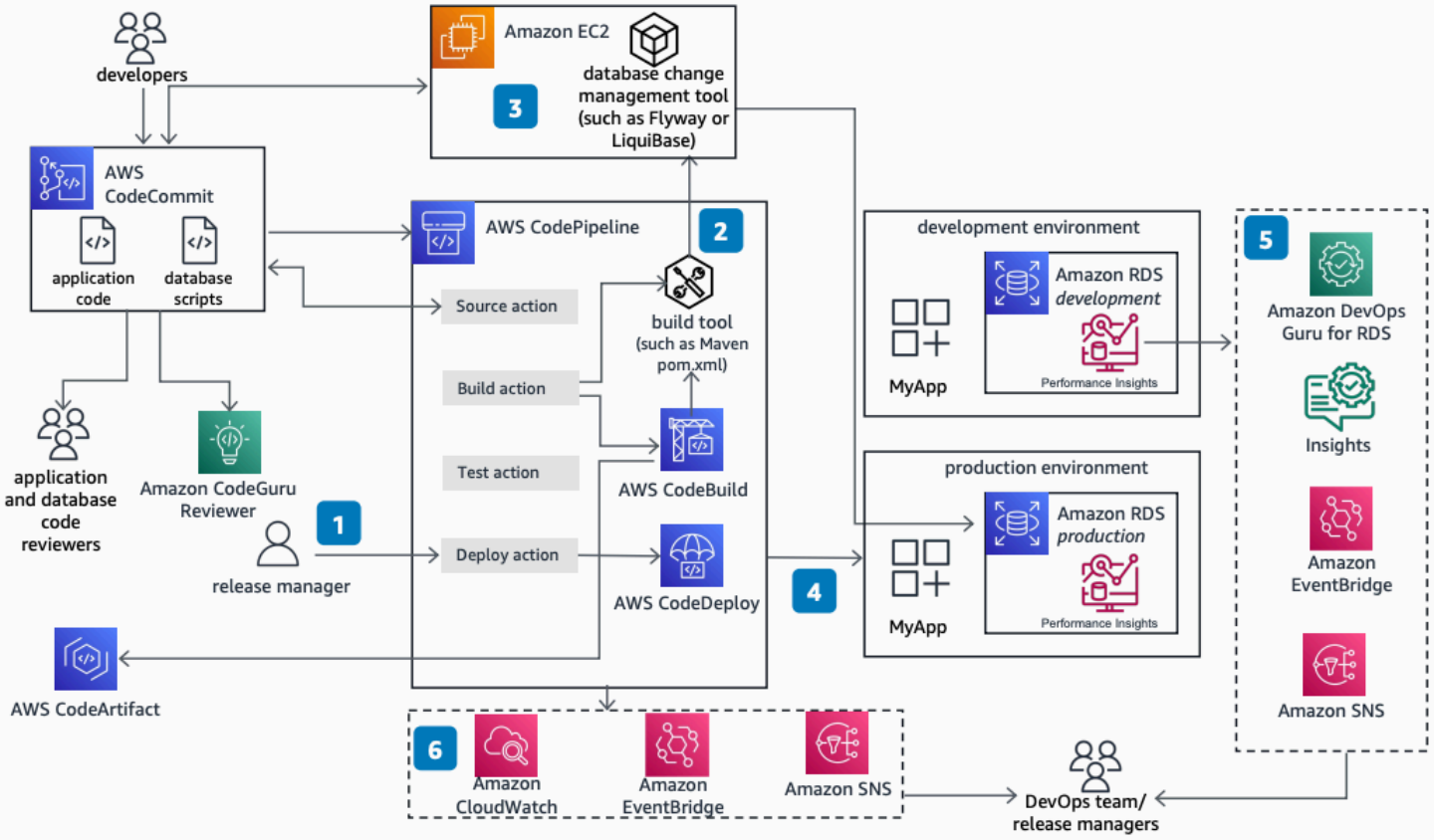
The following steps describe the architecture:

1. The developer commits both application and database code to AWS CodeCommit.
2. At every push request, AWS CodeCommit enforces code reviews through the use of approval rules.
3. Amazon CodeGuru Reviewer automatically evaluates code changes, detects errors, and offers code recommendations.

4. AWS CodeCommit initiates AWS CodePipeline to take pipeline actions.
5. The CodePipeline Source action checks out application and database code.
6. The CodePipeline Build action invokes a build management tool to start the application and database code build phase.
7. The build tool calls the database change tool. It downloads the database scripts and runs them against the target database.
8. The build tool calls database test scripts and validates output.
9. The CodePipeline Deploy action deploys code to the target environment.
10. The CodePipeline Build action publishes build artifacts to AWS CodeArtifact for others to consume.
11. (Optional) Performance Insights on Amazon RDS and Amazon DevOps Guru automatically apply ML techniques to detect performance bottlenecks and operational issues.
12. Amazon EventBridge captures events and sends them to an Amazon SNS topic for user notifications.

Production workflow

Use this architecture in the production workflow to automate database changes as part of DevOps practice. Build an orchestration to deploy database changes at the same rate as application code, detect abnormal system behavior, and automate notifications to appointed teams to take corrective actions.



The following steps describe the architecture:

1. The release manager issues a production deployment request.
2. CodePipeline invokes the build management tool which calls the database change management tool.
3. The database change tool downloads the database change scripts and runs them against the target Amazon RDS production database.
4. CodePipeline initiates the Deploy action to deploy code to the target environment.
5. (Optional) Use Performance Insights on Amazon RDS and DevOps Guru to automatically apply ML techniques to detect performance bottlenecks and operational issues.
6. EventBridge captures events and sends them to an Amazon SNS topic for user notifications.

Further reading

For additional information, refer to the following resources:

- AWS Architecture Icons

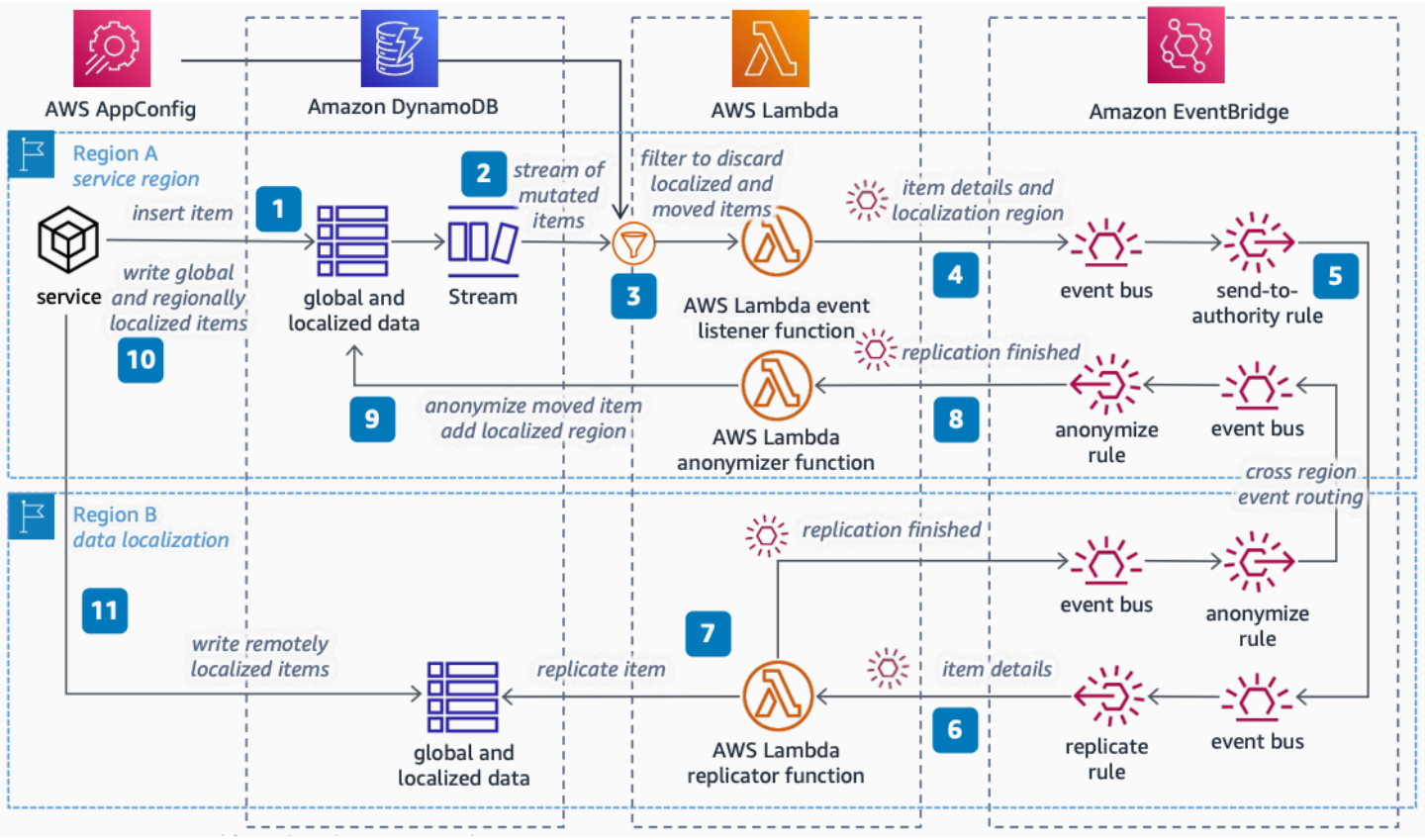
- AWS Architecture Center
- AWS Well-Architected

DynamoDB global replication with data localization

Publication date: **March 10, 2023 (Diagram history)**

Use this architecture for global data-item replication in Amazon DynamoDB with scalability and performance while working to comply with data localization mandates for data to be stored in a specific Region only.

DynamoDB global replication with data localization diagram



The following steps describe the architecture:

1. On the main AWS Region, set services to add items with a localization-authority field into a regional DynamoDB table.
2. Enable DynamoDB Streams to capture events about mutated items and handle the events with an AWS Lambda event-listener function.

3. Map localization authorities to Regions with AWS AppConfig. Read the map on the function, then discard changes on items localized in the current Region and items already moved using event filtering.
4. Raise an event containing the mutated item details and the localization Region to an event bus in Amazon EventBridge.
5. Set up EventBridge rules for cross-Region event routing. Send global items to all Regions. Send remotely localized items only to the localization Region defined on the payload.
6. On destination Regions, handle received item details with a Lambda replicator function.
7. Replicate received items in regional DynamoDB tables, then raise an event to the Region of origin indicating that the replication finished.
8. On the Region of origin, handle the finished event with an anonymizer Lambda function.
9. Set the function to anonymize sensitive information from localized moved items, stamping them with the localized Region that they moved into. Ignore global items.
10. When services update items, only update global items and localized items that belong to the same Region where they are stored.
11. If a localized item to update has already moved to a different Region, make a cross-Region call to the DynamoDB table in the localized Region to update the record.

Further reading

For additional information, refer to the following resources:

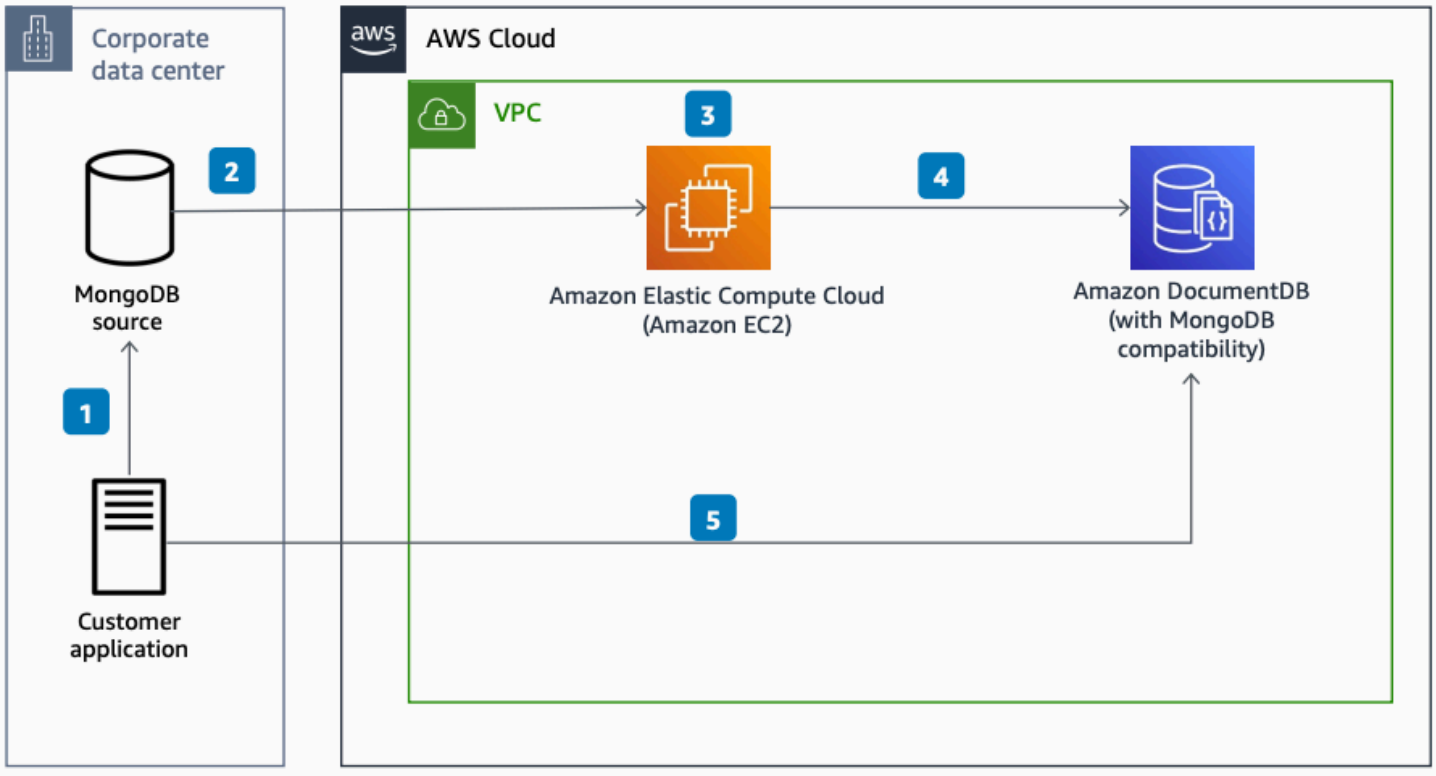
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected

Migration from MongoDB to Amazon DocumentDB

Publication date: **March 25, 2022 (Diagram history)**

This architecture shows how to migrate from MongoDB to Amazon DocumentDB using the offline approach with MongoDB utilities. This is the simplest approach, but it can result in more downtime during the migration.

Offline approach



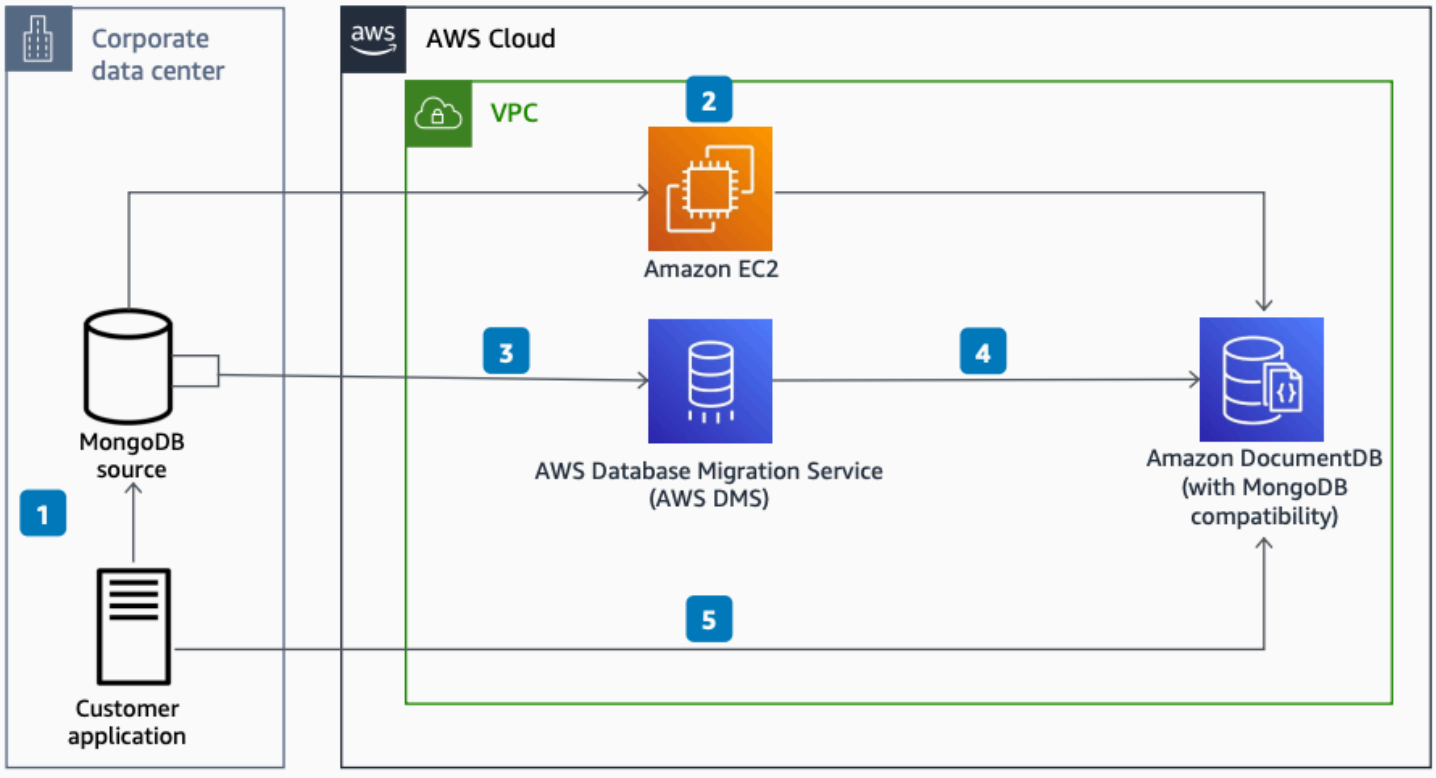
The following steps describe the architecture:

1. Before migration to the Amazon DocumentDB cluster, ensure that the MongoDB source and application drivers use version 3.6 or higher. Stop application writes to the source MongoDB to ensure data consistency.
2. Establish connectivity from the Amazon EC2 instance to your MongoDB source, then export the data and indexes to the Amazon EC2 migration instance using the mongodump tool. To reduce the impact on the source deployment, use the MongoDB read preference option to force the dump to connect to a secondary replica set member.

3. Use the Amazon DocumentDB Index Tool to check the dumped indexes for compatibility and to pre-create the indexes on the target Amazon DocumentDB cluster. This improves overall restore time.
4. Restore exported data to your target Amazon DocumentDB cluster using the mongorestore tool. To parallelize imports, use the mongorestore `numInsertionWorkersPerCollection` option.
5. After data restore is complete, switch your application's database connection string to use your new target Amazon DocumentDB cluster.

Online approach

This architecture shows how to migrate from MongoDB to Amazon DocumentDB using the online approach with AWS Database Migration Service (AWS DMS). This approach delivers near-zero downtime with medium complexity.



The following steps describe the architecture:

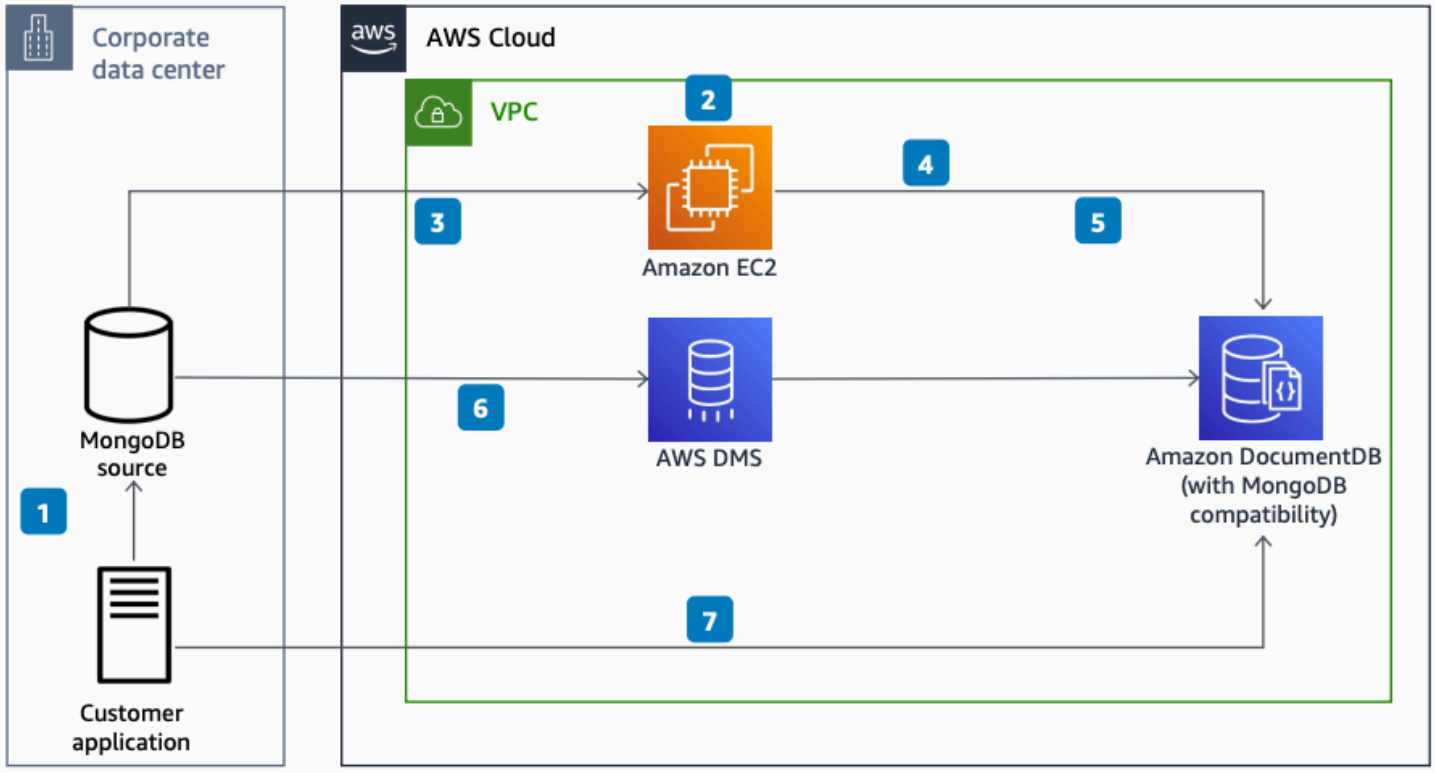
1. When you use the online method to migrate to Amazon DocumentDB, your application continues to write to the source MongoDB database.
2. Create an Amazon EC2 instance in the same VPC as your Amazon DocumentDB cluster and install the mongo shell. Establish connectivity from the Amazon EC2 instance to your

MongoDB source, then export the data indexes to the Amazon EC2 migration instance using the mongodump tool.

3. AWS DMS enables database migration using full data load and change data capture (CDC). The online migration approach uses AWS DMS to perform a full data copy and uses CDC to replicate delta changes to Amazon DocumentDB. To improve the performance of AWS DMS full data load migrations, use the auto segmentation or range segmentation option.
4. Use the Amazon EC2 instance created in step 2 to connect to the Amazon DocumentDB cluster to migrate indexes. Use the Amazon DocumentDB Index Tool to check the dumped indexes for compatibility, and pre-create the indexes on the target Amazon DocumentDB cluster.
5. When the data is in sync, switch your application's database connection string to use the new target Amazon DocumentDB cluster.

Hybrid approach

This architecture shows how to migrate from MongoDB to Amazon DocumentDB using the hybrid approach with MongoDB utilities and AWS DMS. This approach delivers near-zero downtime, and it is the suggested option for source datasets greater than 1 TB.



The following steps describe the architecture:

1. In the hybrid method to migrate to Amazon DocumentDB, your application continues to write to the source MongoDB database.
2. Create an Amazon EC2 instance in the same VPC as your Amazon DocumentDB cluster and install the mongo shell. Establish connectivity from the Amazon EC2 instance to your MongoDB source. Export the data indexes to the Amazon EC2 migration instance using the Amazon DocumentDB Index Tool.
3. Export the data from your MongoDB replica set to the Amazon EC2 migration instance using the mongodump tool. To reduce the impact of the mongodump on the source, set the `readPreference` option to `secondary` to force the dump to connect to a secondary replica set member.
4. Restore the indexes that you exported in step 2 to your target Amazon DocumentDB cluster by using the Amazon DocumentDB Index Tool.
5. To restore the data that you dumped in your target cluster in step 3, use the mongorestore utility.
6. AWS DMS enables database migration using full data load and CDC. The hybrid migration approach uses CDC to replicate changes to Amazon DocumentDB.
7. When the data is in sync, change your application's database connection string to use Amazon DocumentDB cluster.

Further reading

For additional information, refer to the following resources:

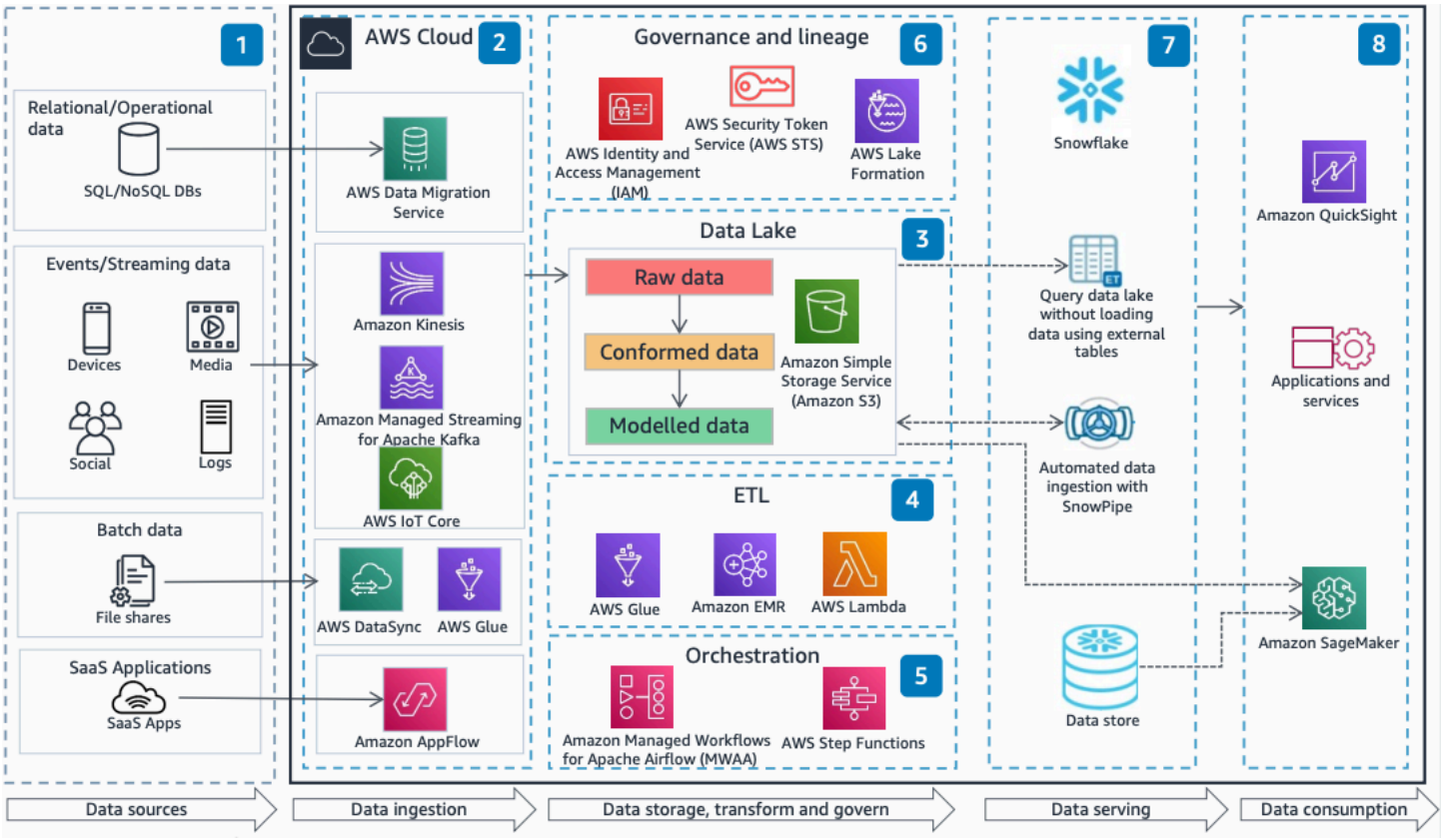
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected

Modern data platform using AWS and Snowflake

Publication date: **March 11, 2022 (Diagram history)**

This architecture shows how to build an end-to-end modern data analytics platform using AWS and Snowflake. You can ingest data from multiple sources, store it in a data lake, and use Snowflake as a virtual data warehouse for analytics.

Modern data platform using AWS and Snowflake diagram



The following steps describe the architecture:

1. Collect data from multiple data sources across the enterprise, software as a service (SaaS) applications, edge devices, logs, streaming data, and social media networks.
2. Based on the type of data source, use AWS Database Migration Service, AWS DataSync, Amazon Kinesis, Amazon Managed Streaming for Apache Kafka, AWS IoT Core, AWS Glue, and Amazon AppFlow to ingest the data into the data lake.
3. Amazon Simple Storage Service provides fully managed, highly available, and scalable data lake storage.

4. AWS Glue extracts, transforms, and ingests data across multiple data stores. Amazon EMR provides the cloud big data platform for processing vast amounts of data using open-source analytics frameworks. AWS Lambda and Amazon EC2 provide compute capability for data enrichment.
5. Amazon Managed Workflows for Apache Airflow (MWAA) or AWS Step Functions orchestrates end-to-end data pipelines.
6. Lake Formation makes it easy to build, secure, and manage your data lake. It provides a single place to enforce data classification and manage fine-grained access. IAM and AWS STS manage access permissions and temporary credentials.
7. Snowflake serves as a virtual data warehouse with the ability to query Amazon S3 using external tables, and automated and continuous data ingestion using Snowpipe.
8. SageMaker AI builds, trains, and deploys machine learning (ML) models and adds intelligence to your applications. AWS Quick Sight provides ML-powered business intelligence (BI).

Further reading

For additional information, refer to the following resources:

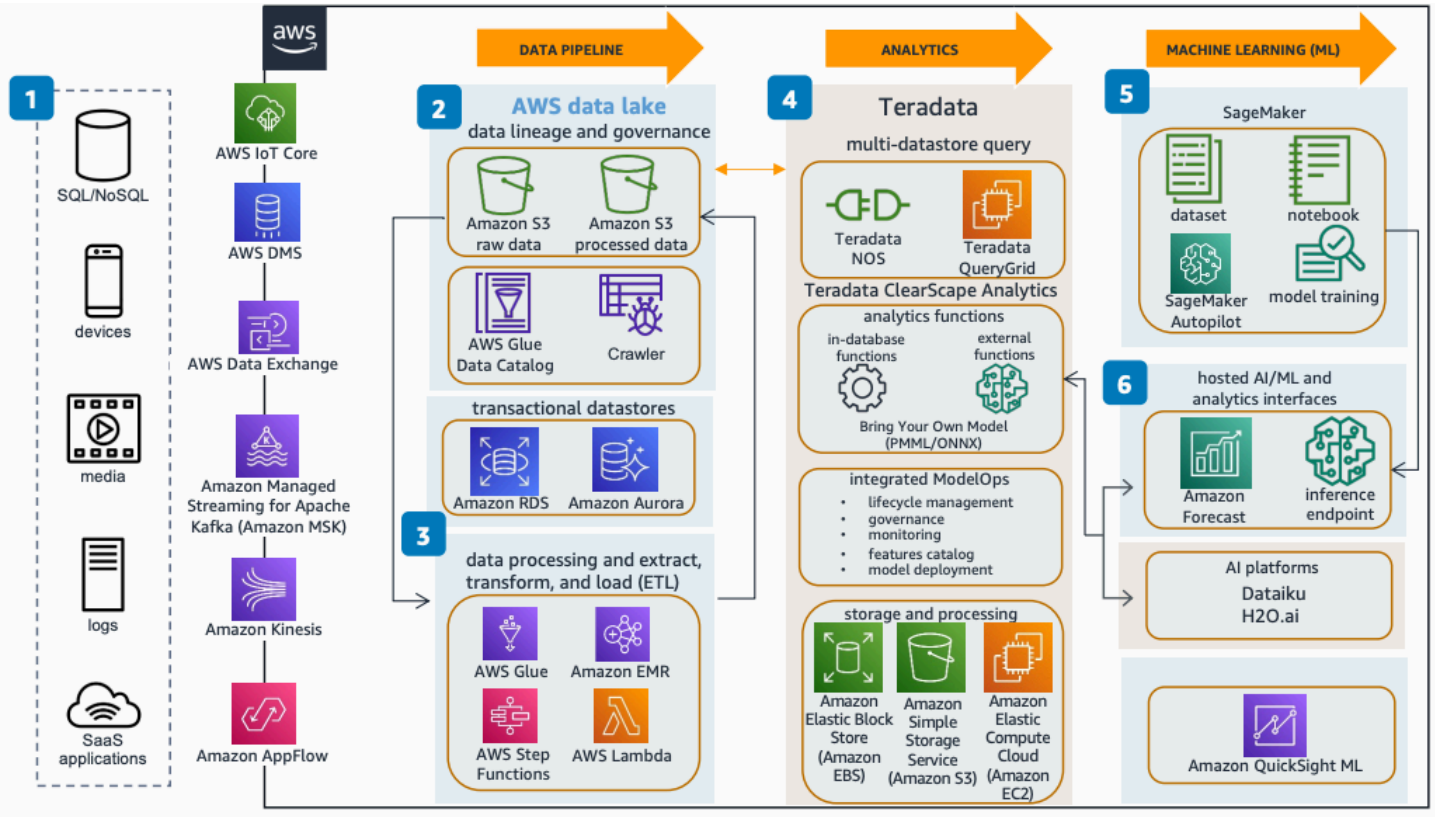
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected

Modern data platform using AWS and Teradata

Publication date: **November 18, 2022 (Diagram history)**

Teradata VantageCloud Enterprise is part of the Teradata VantageCloud offering, the complete cloud analytics and data platform that includes Teradata ClearScape Analytics and integration with Amazon SageMaker AI.

Modern data platform using AWS and Teradata diagram



The following steps describe the architecture:

1. Collect data from multiple sources across the enterprise and SaaS applications. Ingest data using AWS Data Exchange, AWS Database Migration Service (AWS DMS), AWS IoT Core, and Amazon Kinesis. AWS Data Exchange, AWS DMS, and Amazon AppFlow use Amazon S3 as a transitional or permanent datastore accessed through Teradata Native Object Store (NOS).
2. The AWS data lake manages data lifecycle and governance. Amazon S3 provides highly available storage for permanent and transitional data in the data lake.

3. Process the data using AWS Glue and Amazon EMR, and store it to Amazon S3 or directly into Teradata using AWS Glue streaming.
4. Teradata Vantage on AWS serves as a highly scalable data warehouse and analytics platform. With Teradata QueryGrid, the analytics platform can query other datastores such as SQL databases and Amazon EMR. Teradata NOS allows Teradata Vantage to access and configure data on Amazon S3 as an external table.
5. SageMaker AI retrieves training data from both Teradata and Amazon S3 using the Teradata SQL kernel or Python library. Deploy trained models to a SageMaker AI endpoint or import them as a bring your own model (BYOM) into Teradata for in-database analytics.
6. External functions within Teradata SQL interface get inference from AWS inference endpoints and third-party AI platform providers.

Further reading

For additional information, refer to the following resources:

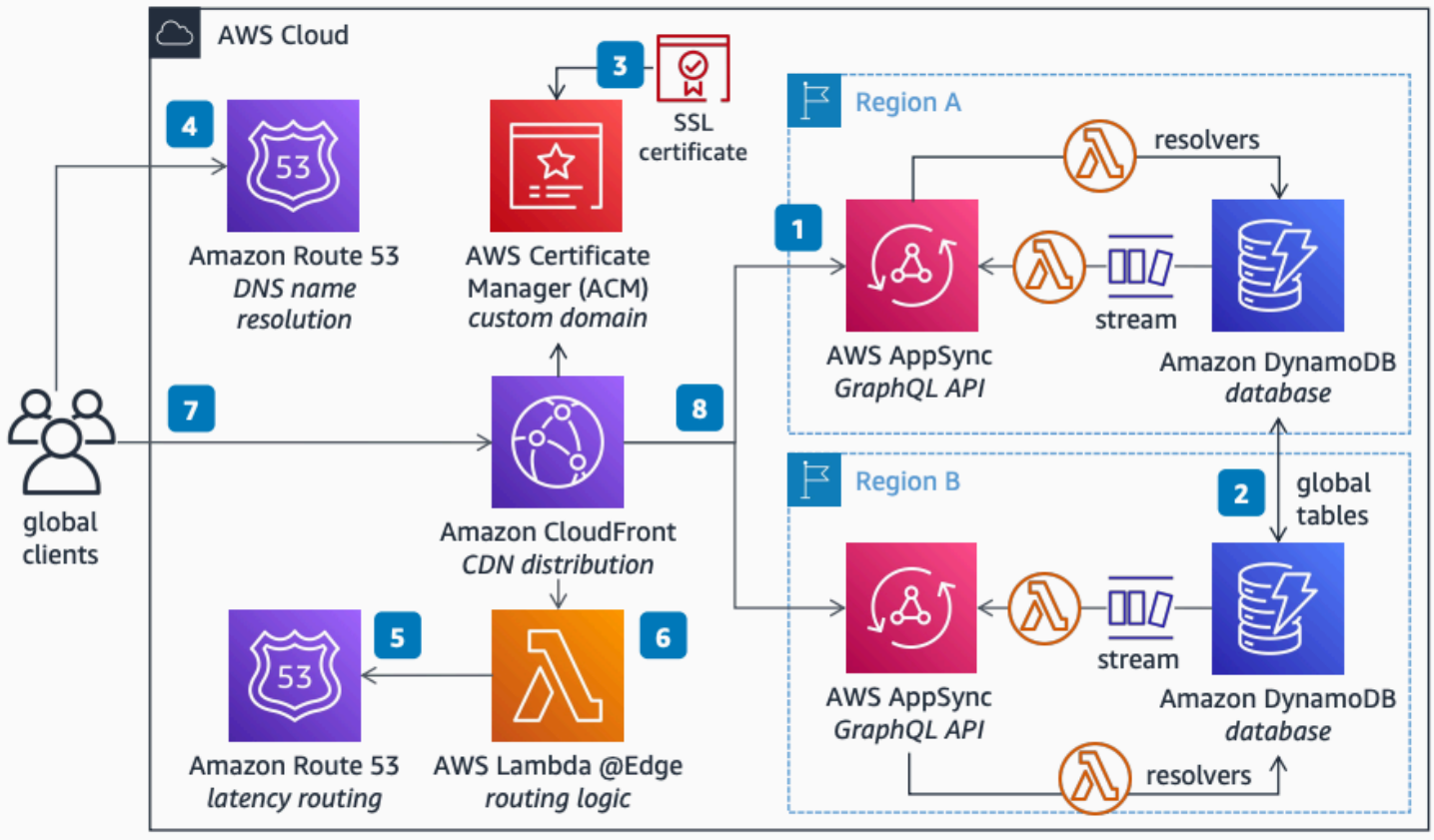
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected

Multi-Region GraphQL API with CloudFront

Publication date: **April 25, 2022 (Diagram history)**

This architecture shows how to reduce latency for end users while increasing your application's availability. You can provide GraphQL API endpoints in multiple AWS Regions with active/active real-time data synchronization supported by Amazon DynamoDB global tables.

Multi-Region GraphQL API with CloudFront diagram



The following steps describe the architecture:

1. Deploy a GraphQL API in two or more AWS Regions using AWS AppSync, then handle the AppSync commands and queries using AWS Lambda resolvers connected to a DynamoDB database.
2. To notify clients about data changes across all Regions, enable DynamoDB global tables to keep data in sync across Regions. Handle DynamoDB data streams with a Lambda handler, triggering purposely built GraphQL schema subscriptions.

3. To support custom domains, upload the domain's SSL certificate into AWS Certificate Manager (ACM) and attach it to an Amazon CloudFront distribution.
4. Point your domain name to CloudFront by using Amazon Route 53 as your DNS name resolution service.
5. Set up a routing rule on Route 53 to route your global clients to the Region with the lowest latency to their location.
6. To authenticate clients seamlessly to AppSync endpoints in any Region, use Lambda@Edge to query Route 53 for the best Region to forward the request to, and to normalize authorization by abstracting the specificities of each Regional AppSync.
7. Clients across the globe can then connect to your GraphQL API on a single endpoint available in edge locations.
8. CloudFront seamlessly routes client requests to the API in the Region with the lowest latency to the client's location.

Further reading

For additional information, refer to the following resources:

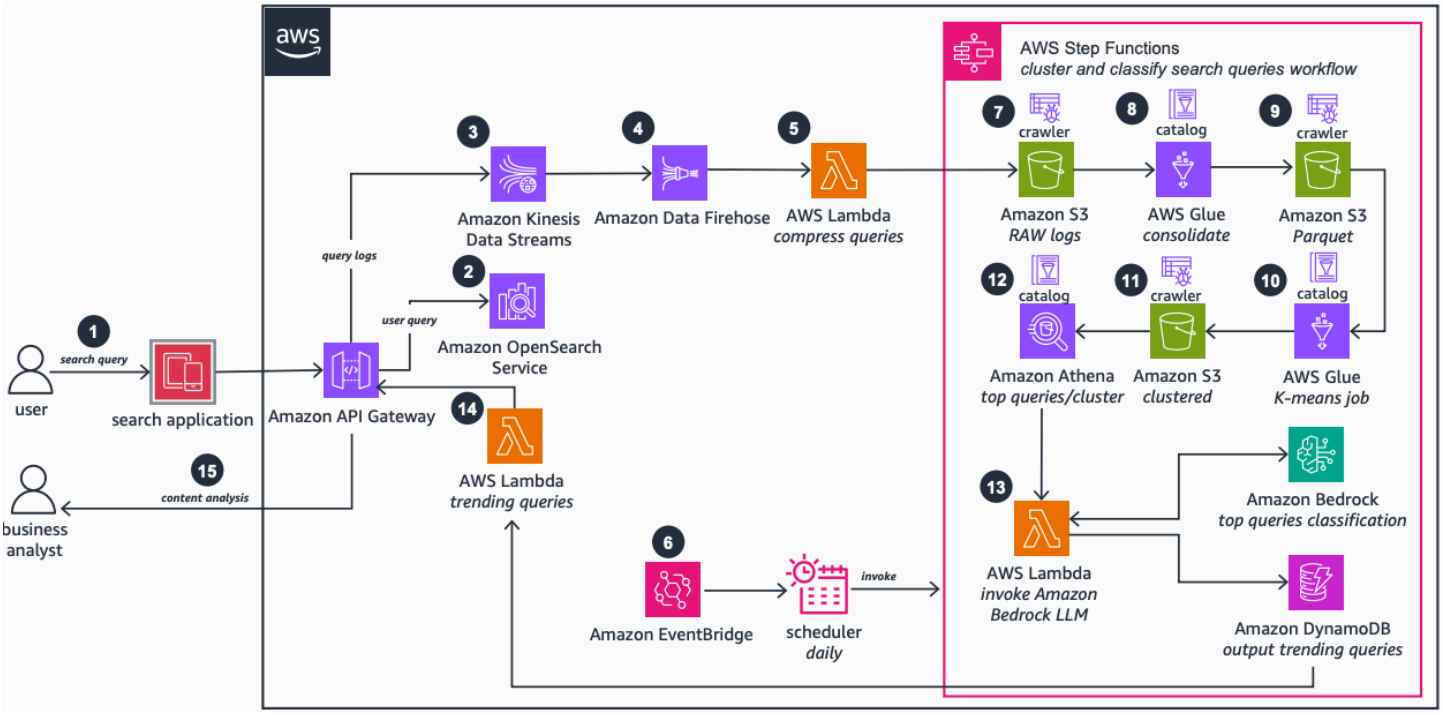
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected

Amazon OpenSearch Service trending queries with AWS Glue and Amazon Bedrock

Publication date: **December 18, 2024 (Diagram history)**

This architecture demonstrates how to use AWS services like AWS Glue, Amazon Bedrock, Amazon OpenSearch Service, vector embedding, K-means clustering, and LLMs to identify top trending search queries for optimizing content strategy and improving user experience.

Amazon OpenSearch Service trending queries with AWS Glue and Amazon Bedrock diagram



The following steps describe the architecture:

1. End users search articles on the search page. Queries are sent to Amazon OpenSearch Service for results retrieval.
2. An Amazon API Gateway proxy streams search query logs to Amazon Kinesis Data Streams.
3. Amazon Data Firehose consolidates search query logs every 15 minutes at the maximum buffer limit.

4. An AWS Lambda function compresses search query logs for Amazon S3 storage optimization.
5. Amazon EventBridge daily scheduler triggers AWS Step Functions for trending query identification.
6. An AWS Glue crawler creates catalog tables for search query logs stored in Amazon S3.
7. An AWS Glue job consolidates and transforms query logs to Parquet files to boost query performance.
8. An AWS Glue crawler creates a catalog table for Parquet files stored in Amazon S3.
9. An AWS Glue job processes data using K-means clustering, creates search query clusters, and stores them in Amazon S3.
10. An AWS Glue crawler creates a catalog table for search query clusters stored in Amazon S3.
11. Amazon Athena queries the top n queries per cluster and passes a CSV file as input to a Lambda function.
12. Lambda processes the CSV file in a loop, invokes Amazon Bedrock to identify the most relevant search query per cluster, and stores results in Amazon DynamoDB.
13. When the user opens the search page, application logic uses API Gateway to retrieve top trending queries using Lambda and the DynamoDB table for display on the search page.
14. Business analysts use a trending query API to analyze trending search queries and define content strategy.

Further reading

For additional information, refer to the following resources:

- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected

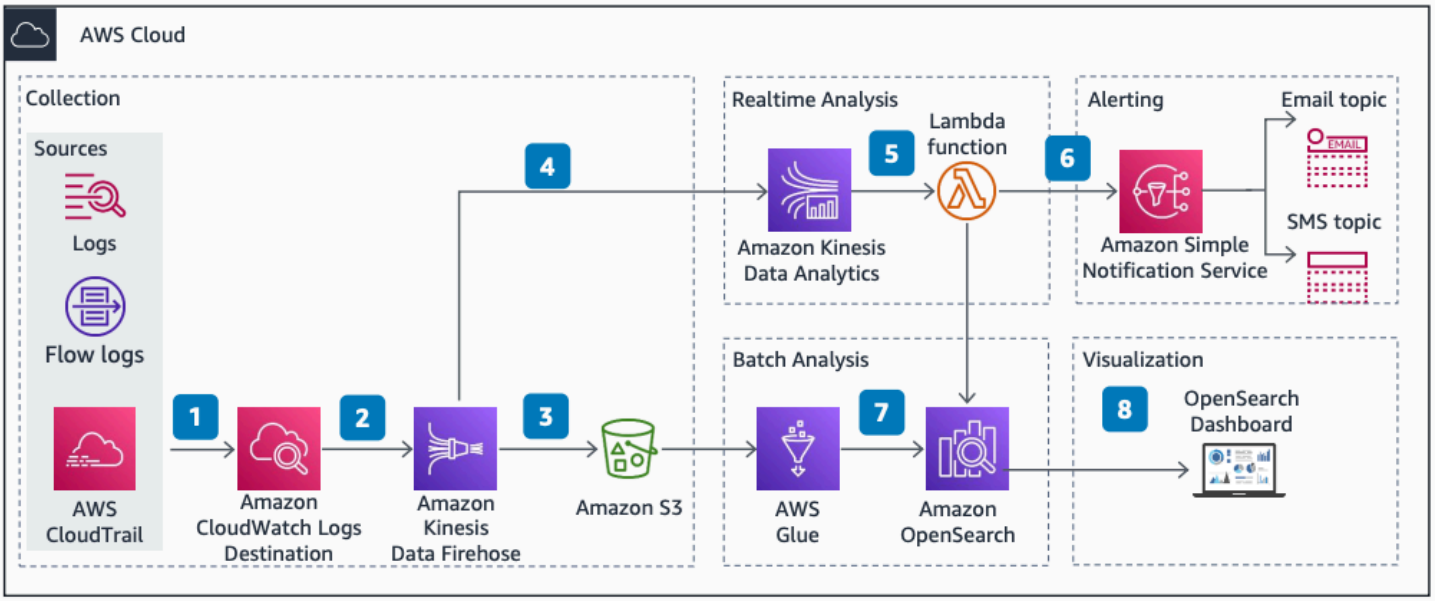
Operational analytics pipeline on AWS

Batch and real-time operational analytics on a modern data architecture

Publication date: **January 1, 2021 (Diagram history)**

With this architecture, you can perform operational analytics in batch and real time. You use log information from operational data sources such as AWS CloudTrail and Amazon VPC flow logs. You visualize operational insights in an OpenSearch dashboard and distribute alerts through Amazon Simple Notification Service (Amazon SNS).

Operational analytics pipeline on AWS diagram



The following steps describe the architecture:

1. Logs stream to Amazon CloudWatch from various data sources. Example sources include AWS CloudTrail and Amazon VPC flow logs.
2. Amazon Managed Service for Apache Flink receives logs intended for real-time analysis.
3. An Amazon S3 bucket receives logs intended for batch processing. This bucket serves as a backup for storing the raw logs.
4. Amazon CloudWatch subscription filters stream log data to Amazon Data Firehose for distribution.

5. Amazon Managed Service for Apache Flink derives metrics and sends them to a AWS Lambda function for processing. The Lambda function transforms the metrics and ingests them into Amazon OpenSearch Service. If a condition requires alerting, the Lambda function publishes a message to Amazon SNS for distribution.
6. AWS Glue collects the logs from Amazon S3 and performs bulk processing, transformation, and analysis. AWS Glue uses the Amazon OpenSearch Service connector to ingest the results into Amazon OpenSearch Service.
7. An OpenSearch dashboard visualizes the metrics from Amazon OpenSearch Service.
8. Amazon SNS notifies consumers of alerts by publishing to email and SMS notification topics.

Further reading

For additional information, see the following resources:

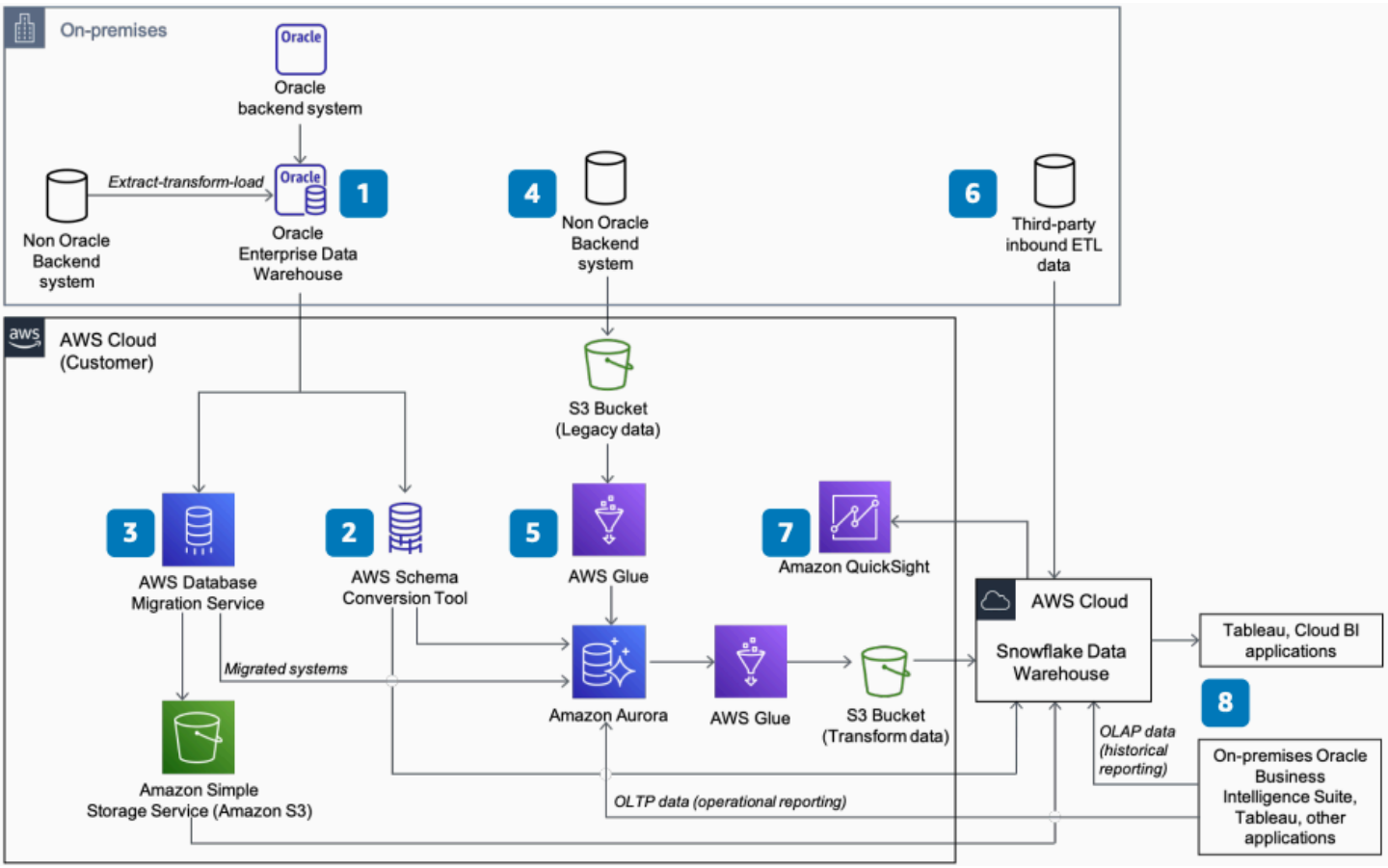
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected

Oracle data workload migration to AWS Cloud

Publication date: **January 1, 2020 (Diagram history)**

With this architecture, you can migrate Oracle and non-Oracle workloads from on-premises to AWS. You migrate online transaction processing (OLTP) data to Amazon Aurora for operational reporting. You migrate online analytical processing (OLAP) data to Snowflake Data Warehouse for historical reporting.

Oracle data workload migration to AWS Cloud diagram



The following steps describe the architecture:

1. Oracle and non-Oracle backend systems regularly load transactional data into an Oracle Enterprise Data Warehouse.

2. AWS Schema Conversion Tool extracts Oracle data definition language (DDL) scripts and schema from all Oracle database tables. It publishes the DDL scripts to Amazon Aurora and sends the schema to Snowflake Data Warehouse.
3. AWS Database Migration Service (AWS DMS) extracts OLAP and OLTP data from all Oracle tables. AWS DMS stores OLAP data in Amazon S3 in CSV format. AWS DMS loads OLTP data directly into tables created in Amazon Aurora.
4. Legacy systems load data into an Amazon S3 bucket. AWS Glue transforms data from the Amazon S3 bucket and loads it into Amazon Aurora.
5. AWS Glue transforms data from Amazon Aurora and stores the transformed data in an Amazon S3 bucket. Snowflake Data Warehouse loads transformed data from the Amazon S3 bucket into OLAP tables.
6. Cloud-based extract-transform-load (ETL) platforms extract data from non-migrated data warehouse systems. These platforms load the data into Snowflake Data Warehouse.
7. AWS Quick Sight receives data for analysis and reporting. Tableau and cloud business intelligence (BI) tools also receive the data.
8. Cloud-based and on-premises BI tools can query OLAP data directly from Snowflake for historical reporting. These tools can also query OLTP data directly from Amazon Aurora for operational reporting.

Further reading

For additional information, see the following resources:

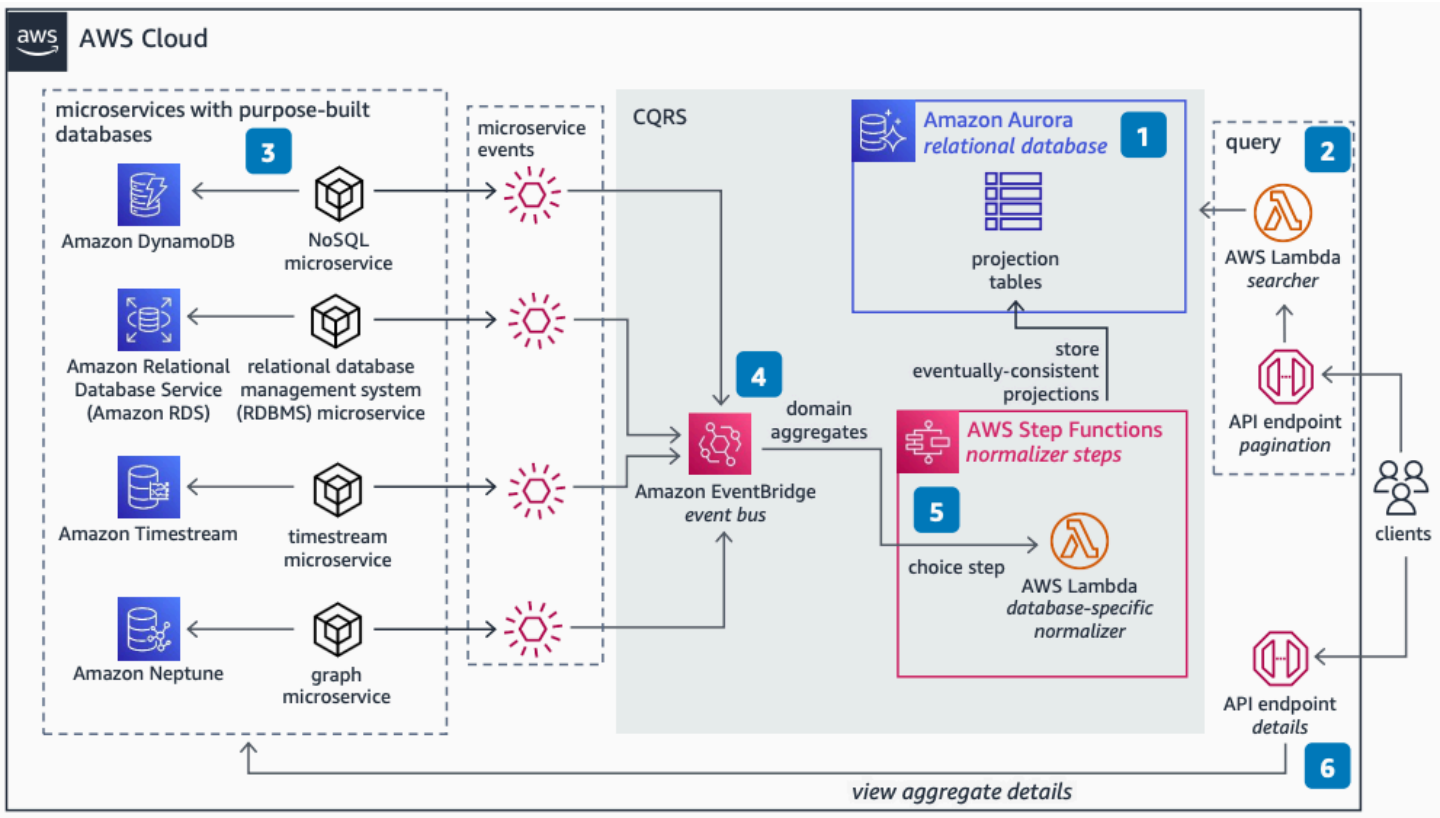
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected

Paginated search with purpose-built databases

Publication date: **August 25, 2022 (Diagram history)**

This architecture shows how to create a searchable, paginated list on domain aggregates from purpose-built databases with a step transformation and SQL queries. You can apply the command query responsibility segregation (CQRS) pattern to an event-driven microservice architecture. This enables searching, pagination, and sorting by querying eventually consistent data projections in Amazon Aurora.

Paginated search with purpose-built databases diagram



The following steps describe the architecture:

1. Create a relational database using Amazon Aurora to hold projections of the domain aggregates you need to search, paginate, and sort. Work backward from the user interfaces to design the tables and columns.

2. Create an AWS Lambda function to invoke a SQL query against the relational database, then connect it to your pagination API endpoint to handle search requests. The function uses a `Where` clause for searching, `OrderBy` for sorting, and `Limit`, `Offset`, and `Count` for pagination.
3. Decoupled microservices raise events for changes in the domain aggregates they own. Each microservice selects the purpose-built database that suits their use case, while eventually keeping the projection database up to date consistently.
4. Use Amazon EventBridge to create an event bus to collect the domain events from your microservices.
5. Normalize domain aggregates with database-specific normalizers to match the projection tables in the database and calculate additional analytics. To manage the workflow and direct domain aggregates to the right normalizer, create a choice step within AWS Step Functions. For complex normalizations, the state machine can integrate and wait for long-running containers on Amazon Elastic Container Service.
6. Clients can request details on a returned aggregate, referring to it by a unique key. The relevant microservice manages this through the API endpoint.

Further reading

For additional information, refer to the following resources:

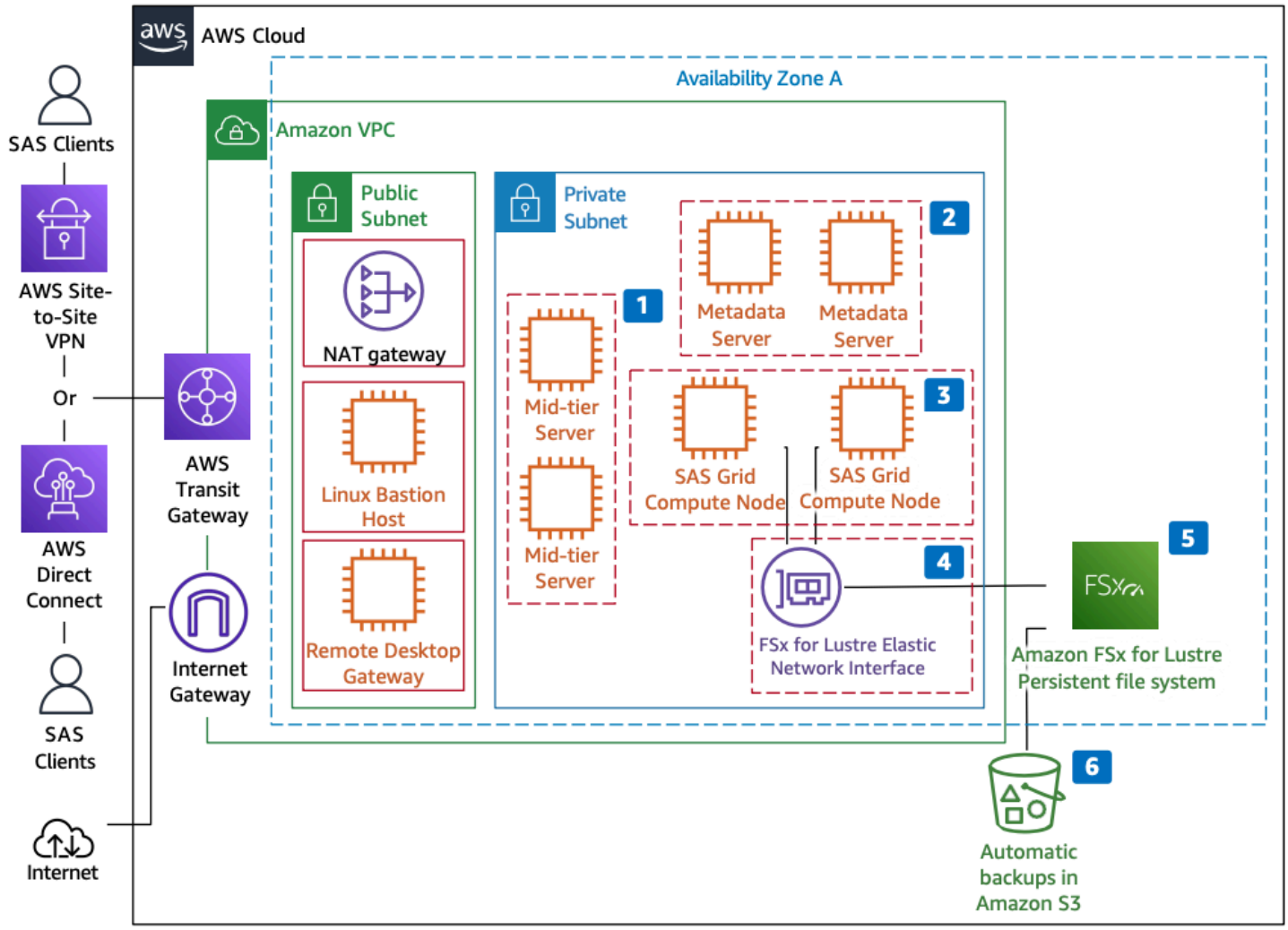
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected

Running SAS Grid on AWS

Publication date: **July 25, 2022 (Diagram history)**

With this architecture, you can deploy high-performing Amazon FSx for Lustre file system storage for SAS Grid. You also get guidance on the Amazon EC2 instance types best suited for the SAS Grid compute tier.

Running SAS Grid on AWS diagram



The following steps describe the architecture:

1. For mid-tier servers, select Amazon EC2 r5 instance types. Use these to run Platform Web Services (PWS) and the Load Sharing Facility (LSF) client. Two or more instances are not a SAS requirement unless you require high availability (HA).

2. For metadata servers, select Amazon EC2 r5 instance types that meet or exceed the minimum recommendations from SAS. For memory, use the larger of 8 GB per physical core or 24 GB. Three or more instances are not a SAS requirement unless you require HA.
3. For SAS Grid compute nodes running the LSF platform, select Amazon EC2 instance types that meet or exceed the minimum recommendations from SAS. For memory, use 8 GB per physical core. For file system performance, use 100-125 MB/s per physical core.

   We recommend the Amazon EC2 m5n and r5n instance types when hosting /SASDATA, /SASWORK, and /UTILLOC on FSx for Lustre. Use i3en instance types when offloading /SASWORK to local instance store volumes.
4. The FSx for Lustre file system is accessible through an elastic network interface in your virtual private cloud (VPC). Use standard Amazon VPC security groups to control network access to your file system.
5. Use FSx for Lustre persistent file systems for all SAS Grid libraries. These libraries include /SASDATA, /SASWORK, and /UTILLOC.

   FSx for Lustre is a fully managed Lustre file system. It delivers hundreds of gigabytes per second of throughput and millions of IOPS. It provides submillisecond latencies and supports encryption of data at rest and in transit.
6. FSx for Lustre stand-alone file systems provide automatic, highly durable, file-system-consistent, incremental backups. FSx for Lustre stores these backups in Amazon S3.

Further reading

For additional information, see the following resources:

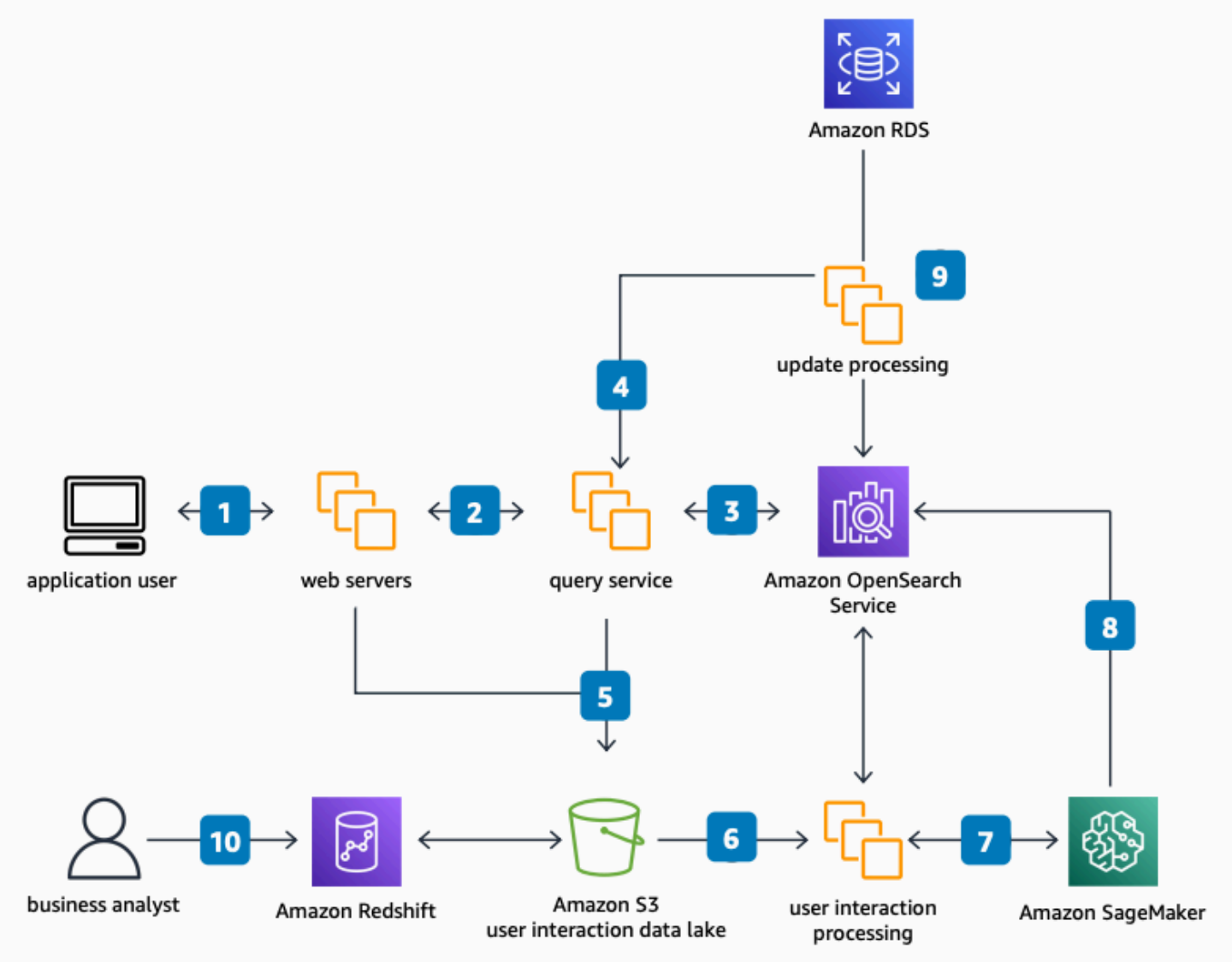
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected

Search-backed applications

Publication date: **November 17, 2022 (Diagram history)**

This architecture outlines the process to add or improve search for an existing application. It uses Amazon OpenSearch Service for indexing and retrieval, Amazon SageMaker AI for ML enrichment, and Amazon Simple Storage Service for user interaction data storage.

Search-backed applications diagram



The following steps describe the architecture:

1. An application user sends a query to the web servers.

2. The web servers deliver the query to the query service. The query service can use ML models from SageMaker AI for user segmentation, concept and entity extraction, and query-to-click analysis.
3. The query service enriches or rewrites the query based on user segmentation from SageMaker AI, user preferences from Amazon RDS, and past query performance. It sends the augmented query to Amazon OpenSearch Service.
4. Send only searchable data to Amazon OpenSearch Service, and use a relational or NoSQL system as the system of record. The query service retrieves only keys in the search results and fetches the full record from the system of record.
5. The web servers and query service send user interaction data to an Amazon S3 data lake or Amazon Redshift.
6. An offline process pulls user interaction data from the data lake.
7. The offline process takes data such as clicks to augment the records in the catalog and updates models in SageMaker AI.
8. Records are updated in Amazon OpenSearch Service as needed.
9. The web servers send catalog updates to Amazon OpenSearch Service, or a change data capture process brings those updates to Amazon OpenSearch Service.
10. Business analysts generate reporting and KPIs from the processed user interactions.

Further reading

For additional information, refer to the following resources:

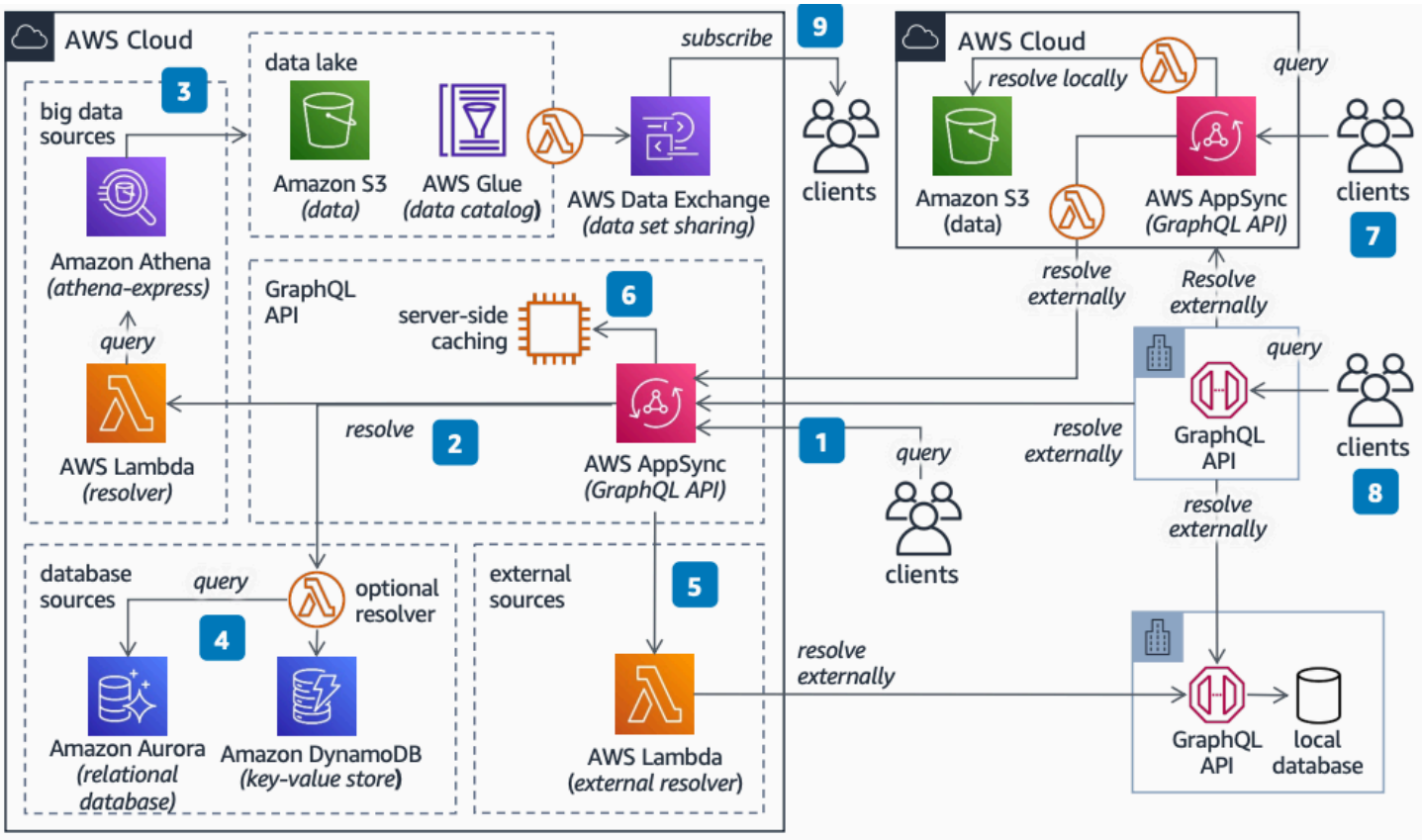
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected

Synchronous data mesh for GraphQL queries

Publication date: **April 25, 2022 (Diagram history)**

This architecture shows how to use an API composition pattern to build a modern, distributed, and decentralized data architecture. It enables clients to query data where it lives, without first transporting it to a data lake or data warehouse. It also allows domain-specific teams to own and serve data as a product.

Synchronous data mesh for GraphQL queries diagram



The following steps describe the architecture:

1. Expose your domain's schema through an HTTPS API with AWS AppSync, allowing users to dynamically query a domain's data with GraphQL syntax.
2. Compose the returning query results with a combination of resolvers built with AWS Lambda functions.

3. For data partially available in a data lake, retrieve the data by running SQL queries supported by Athena. Run the Athena jobs asynchronously with the athena-express library.
4. For data partially available in databases, retrieve the data either directly from AWS AppSync, or by using Lambda functions as proxy resolvers to your databases.
5. For data partially available in external sources, use Lambda resolvers to fetch the data by invoking remote HTTP APIs.
6. To improve your API's performance, enable server-side caching on AWS AppSync.
7. Other parties using AWS can replicate this architecture in their domains, also allowing clients to query their APIs using GraphQL, and composing the results with internal and external resolvers.
8. Parties that do not use AWS can also expose their domains with GraphQL APIs built with other technologies, providing a seamless experience to clients.
9. To retrieve large datasets, clients can subscribe to AWS Data Exchange instead.

Further reading

For additional information, refer to the following resources:

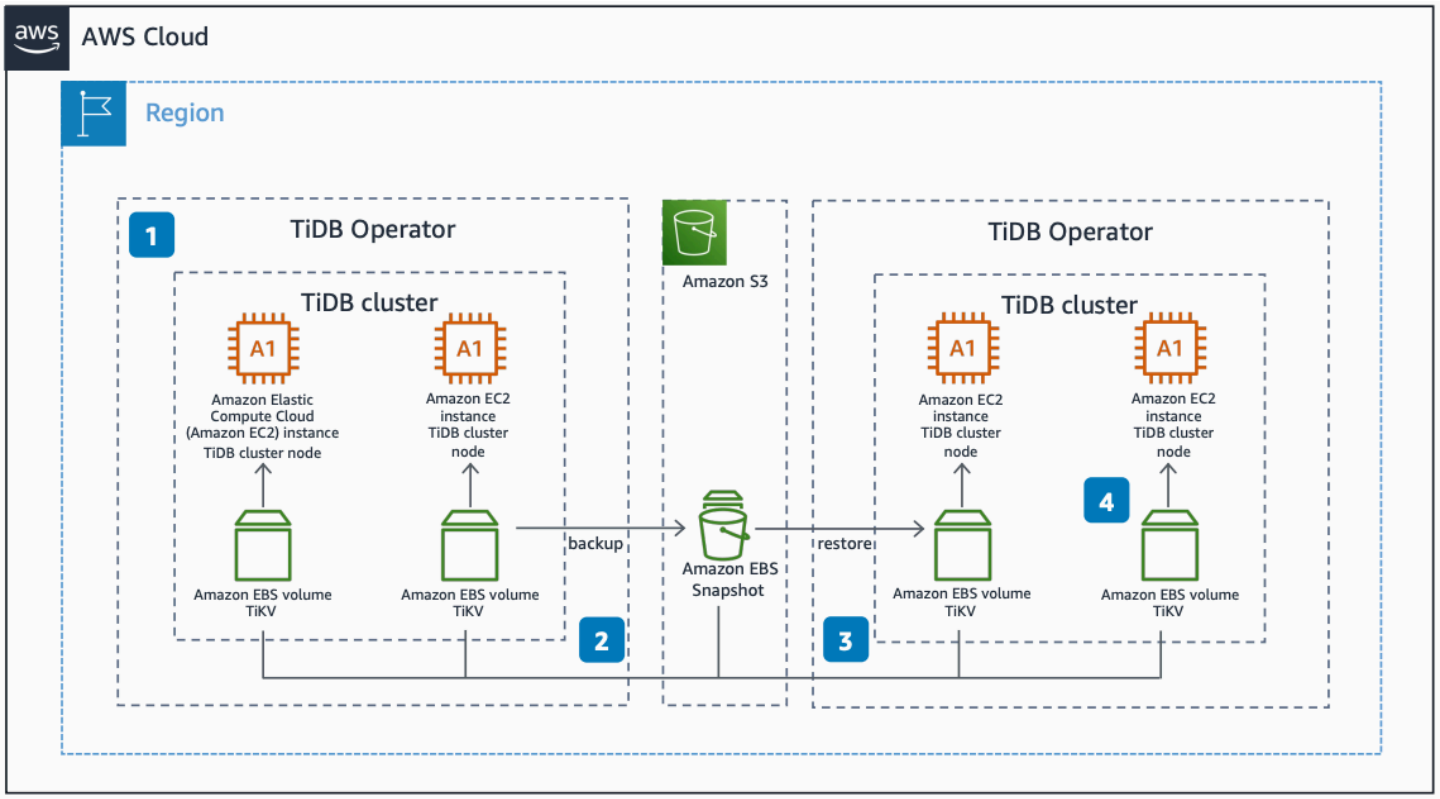
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected

Fast backup and restore of TiDB on AWS

Publication date: **February 16, 2023 (Diagram history)**

TiDB is an open-source MySQL-compatible database that supports hybrid transactional and analytical processing (HTAP). TiDB clusters deployed on AWS use Amazon EBS snapshots for fast backup and restore of several terabytes of data in one hour without impacting cluster performance.

Fast backup and restore of TiDB on AWS diagram



The following steps describe the architecture:

1. When the user submits a backup request, TiDB Operator checks and collects information from volumes mounted to TiKV and creates a backup job. The backup job pauses the scheduler and garbage collection.
2. The backup job asks AWS to create an Amazon EBS snapshot and resumes the schedulers and garbage collection. After the snapshot completes, the backup job saves the metadata to Amazon S3.

3. When the user submits a restore request, the backup job creates a restore job that retrieves backup metadata from Amazon S3 and extracts snapshot information. It invokes the AWS API to create volumes from the Amazon EBS snapshot and returns the created volume information to TiDB Operator.
4. TiDB Operator configures Amazon EKS and mounts the restored volumes to corresponding nodes. After the data restore completes, the TiDB cluster exits restore mode and starts all TiDB nodes to serve requests.

Further reading

For additional information, refer to the following resources:

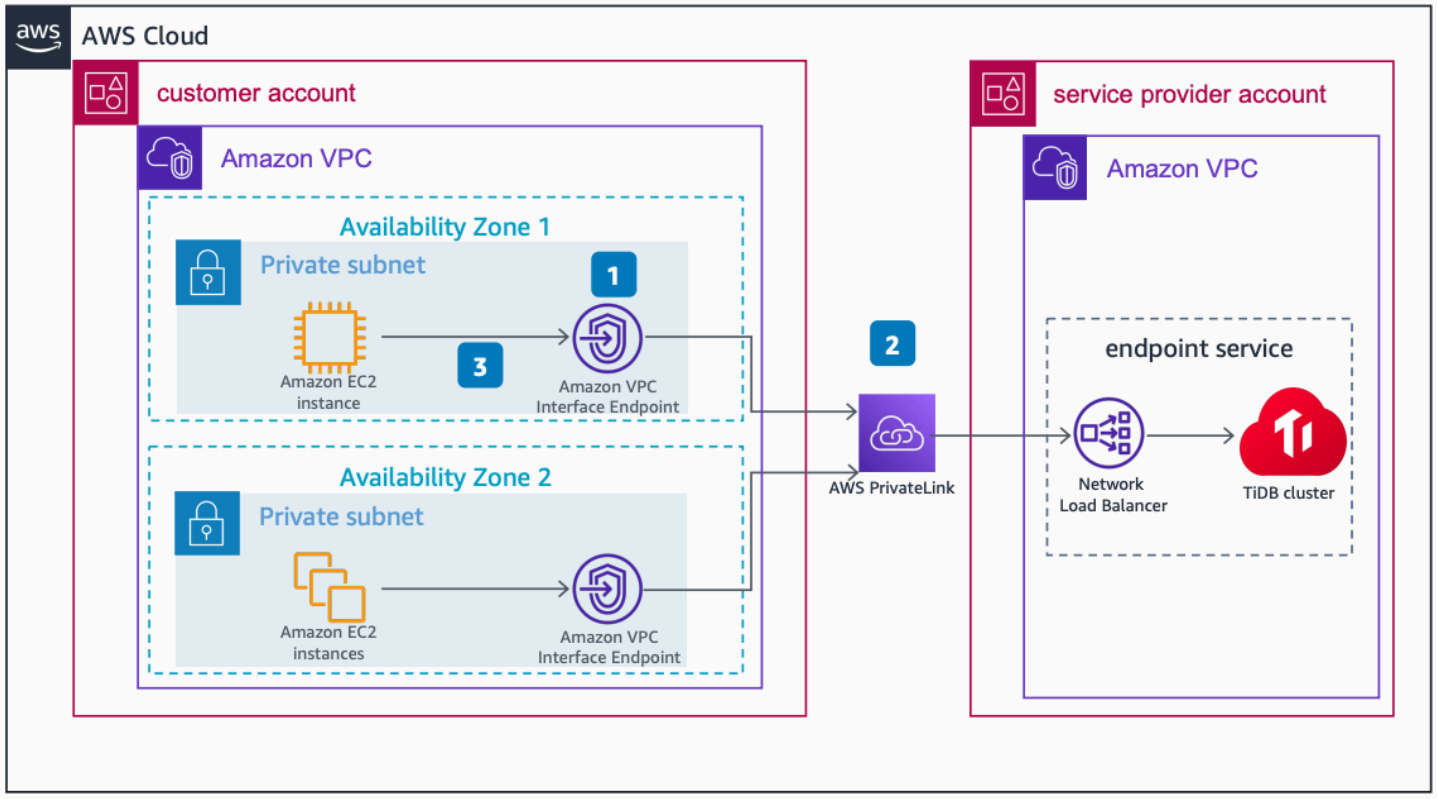
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected

Securely access TiDB using AWS PrivateLink

Publication date: **February 22, 2023 (Diagram history)**

TiDB is an open-source MySQL-compatible database that supports hybrid transactional and analytical processing (HTAP). TiDB Cloud supports highly secure and unidirectional access to the TiDB Cloud service hosted in an Amazon VPC by using AWS PrivateLink. PingCAP sets up the AWS PrivateLink service for TiDB Cloud.

Securely access TiDB using AWS PrivateLink diagram



The following steps describe the architecture:

1. Create an Amazon VPC interface endpoint to the TiDB Cloud AWS PrivateLink endpoint service.
2. Powered by AWS PrivateLink, the endpoint connection is secure and private, and does not expose data to the public internet. The AWS PrivateLink connection also supports a secure connection between VPCs with overlapping Classless Inter-Domain Routing (CIDR).
3. Amazon EC2 customer instances securely connect to the TiDB Cloud service hosted in the provider Amazon VPC through VPC endpoints, without traffic traversing the public internet.

With private DNS enabled, consumers use the same public DNS when connecting over AWS PrivateLink.

Further reading

For additional information, refer to the following resources:

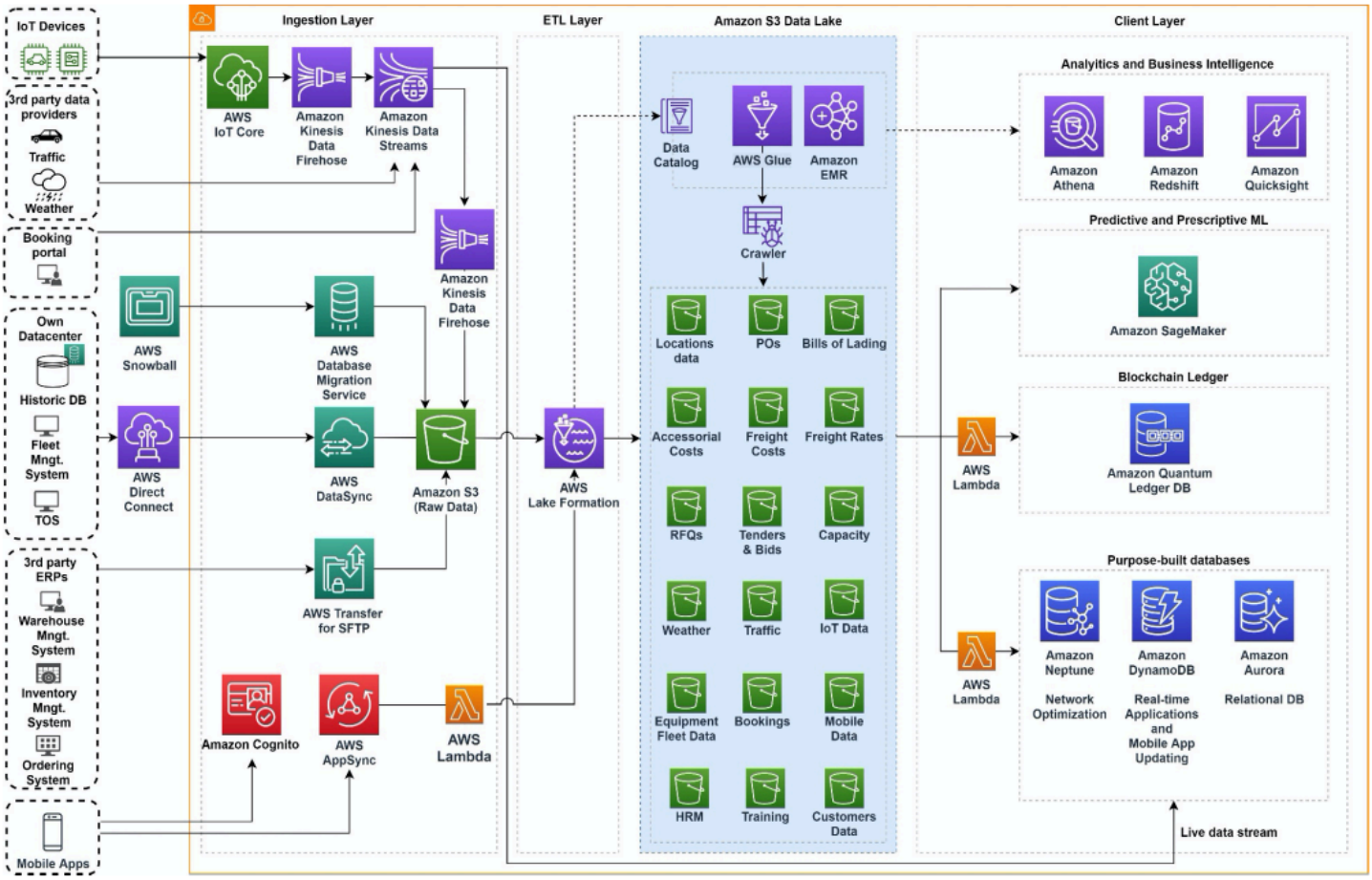
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected

Transportation and logistics data lake

Publication date: **March 15, 2020 (Diagram history)**

This architecture shows how to break data silos and create a single repository for transportation and logistics data from multiple sources. You can use a fully managed, pay-per-use, and scalable architecture to remove the complexity and costs of managing on-premises databases.

Transportation and logistics data lake diagram



The following steps describe the architecture:

1. Ingest data from multiple sources: historical and large databases with AWS Snowball, real-time data with Amazon Kinesis Data Streams, ERP data synchronization with AWS DataSync, and mobile app data with AWS AppSync.
2. Automatically extract, transform, and load raw data with Lake Formation. Discover schemas and create a data catalog for further data exploration.

3. Create an Amazon Simple Storage Service data lake where you can store, organize, and access business-critical data securely. Use Amazon EMR for fine-grained control over extraction, transformation, and loading (ETL) jobs. Use AWS Glue to manage your data catalog.
4. Turn raw data into actionable knowledge with Athena and AWS Quick Sight. Disseminate key business intelligence insights across your organization. Use SageMaker AI to deliver ML-based predictive and prescriptive analytics.
5. Use AWS fully managed databases to deliver specific capabilities: Amazon Neptune graph database for network optimization, Amazon Aurora for fast relational queries, and Amazon DynamoDB for interactive mobile and web apps with real-time updates. Avoid data latency with live data stream ingestion from Kinesis Data Streams.

Further reading

For additional information, refer to the following resources:

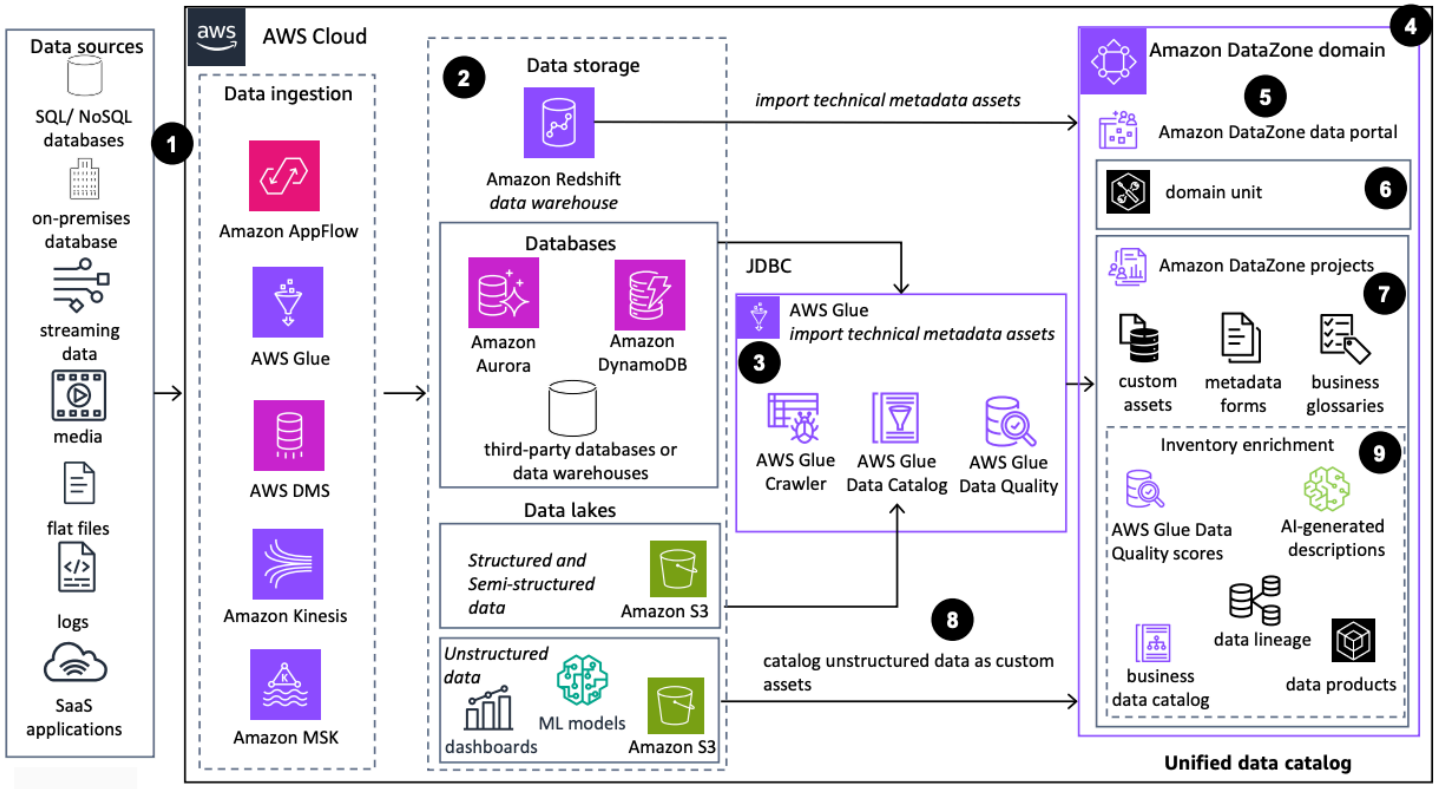
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected

Building a unified catalog with Amazon DataZone

Publication date: **November 20, 2024 (Diagram history)**

The unified data catalog acts as a central repository for all your organization's data assets. This architecture shows how Amazon DataZone breaks down data silos by bringing data from various sources together and fostering improved search functionality and trust in data.

Building a unified catalog with Amazon DataZone diagram



The following steps describe the architecture:

1. Extract data using Amazon AppFlow, AWS Glue, Amazon Kinesis Data Streams, Amazon Managed Streaming for Apache Kafka (Amazon MSK), or AWS Database Migration Service (AWS DMS).
2. Based on end user requirements, store the data in Amazon S3, Amazon Redshift, or purpose-built databases like Amazon Aurora or DynamoDB. Design data lakes to store raw structured, semi-structured, and unstructured data at low cost.

3. Crawl structured and semi-structured data from Amazon S3 using AWS Glue Crawler, which writes metadata to AWS Glue Data Catalog. Perform data quality checks on catalog tables at rest using AWS Glue Data Quality. Use Java Database Connectivity (JDBC) data sources to crawl and catalog the data.
4. The Amazon DataZone domain acts as the central catalog repository hub. Use the AWS Glue Data Catalog data source to onboard existing data assets from data lakes and databases. With Amazon Redshift, automatically extract technical metadata of database tables and views to Amazon DataZone.
5. Use the Amazon DataZone data portal to discover, catalog, share, and govern data in a self-serve fashion.
6. Organize Amazon DataZone entities under different levels of hierarchy using domain units.
7. Amazon DataZone data projects provide a collaborative space where members onboard data assets from their respective business units. Business glossaries define business terms associated with data assets. Metadata forms augment business context of asset metadata. Custom assets expand the catalog beyond predefined system assets.
8. Catalog unstructured data assets stored in Amazon S3 and publish them to Amazon DataZone by tagging them with relevant custom asset types.
9. Every data asset onboarded to Amazon DataZone is part of Inventory. Enrich the data asset with business catalogs, data lineage, and data quality to aid discoverability.

Further reading

For additional information, refer to the following resources:

- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected

Diagram history

The following table describes updates to the reference architecture diagrams in this guide. To be notified about updates, subscribe to the RSS feed.

Change	Description	Date
Amazon OpenSearch Service trending queries with AWS Glue and Amazon Bedrock	Reference architecture diagram first published.	December 18, 2024
Data mesh with Amazon DataZone	Reference architecture diagram first published.	November 20, 2024
Building a unified catalog with Amazon DataZone	Reference architecture diagram first published.	November 20, 2024
Database DevOps on AWS	Reference architecture diagram first published.	March 31, 2023
DynamoDB global replication with data localization	Reference architecture diagram first published.	March 10, 2023
Alfresco Digital Business Platform on AWS	Reference architecture diagram first published.	March 1, 2023
Securely access TiDB using AWS PrivateLink	Reference architecture diagram first published.	February 22, 2023
Fast backup and restore of TiDB on AWS	Reference architecture diagram first published.	February 16, 2023
Unified data access governance in a data mesh	Reference architecture diagram first published.	February 13, 2023
Modern data platform using AWS and Teradata	Reference architecture diagram first published.	November 18, 2022

Search-backed applications	Reference architecture diagram first published.	November 17, 2022
Paginated search with purpose-built databases	Reference architecture diagram first published.	August 25, 2022
Running SAS Grid on AWS	Reference architecture diagram first published.	July 25, 2022
Synchronous data mesh for GraphQL queries	Reference architecture diagram first published.	April 25, 2022
Multi-Region GraphQL API with CloudFront	Reference architecture diagram first published.	April 25, 2022
Expose your data lake as a GraphQL API	Reference architecture diagram first published.	April 25, 2022
Migration from MongoDB to Amazon DocumentDB	Reference architecture diagram first published.	March 25, 2022
Modern data platform using AWS and Snowflake	Reference architecture diagram first published.	March 11, 2022
Change data capture using AWS Database Migration Service	Reference architecture diagram first published.	August 2, 2021
Automated archival for Amazon Redshift	Reference architecture diagram first published.	July 12, 2021
Managing inserts and upserts in a serverless data lake	Reference architecture diagram first published.	June 15, 2021
Operational analytics pipeline on AWS	Reference architecture diagram first published.	January 1, 2021
Transportation and logistics data lake	Reference architecture diagram first published.	March 15, 2020

Oracle data workload migration to AWS Cloud	Reference architecture diagram first published.	January 1, 2020

Note

To subscribe to RSS updates, you must have an RSS plugin enabled for the browser you are using.