

Equipo de trabajo:

Karen Julieth Soler Ardila

Sarita Andrea Salazar Urbano

Erick Hernando Parrado Moyano

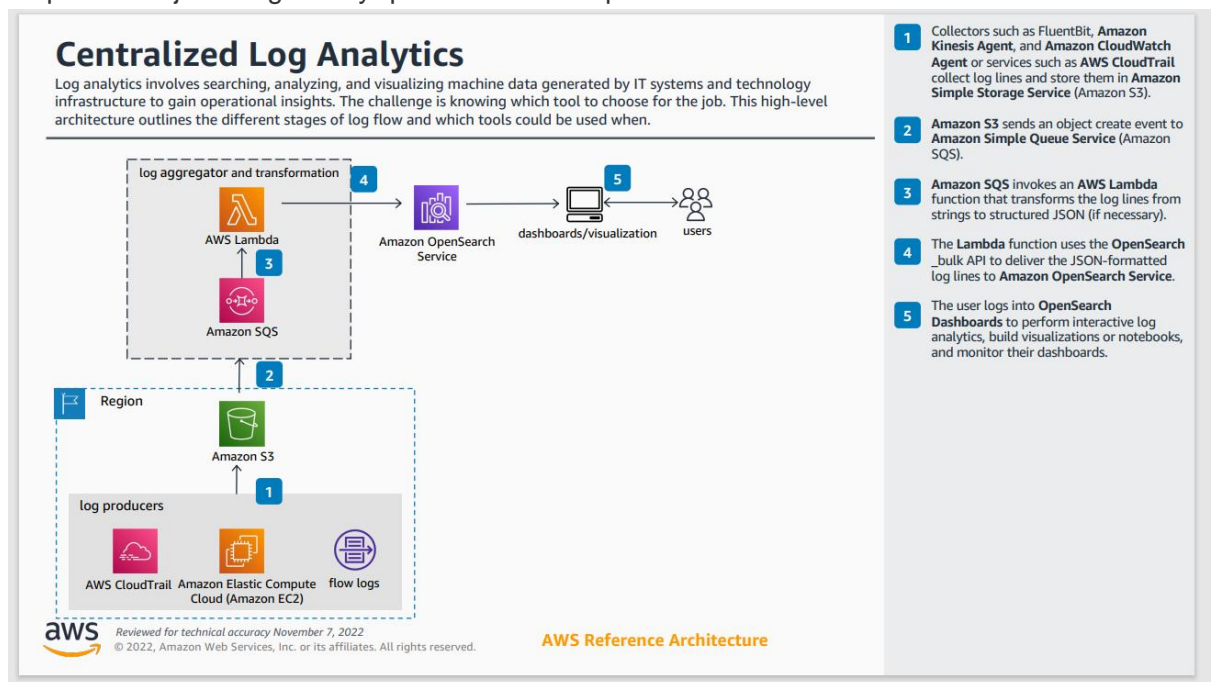
Programa: Arquitectura de Software

ARQUITECTURAS DE REFERENCIAS METAL K

Aplicativo web para la gestión de portafolio de servicios para la empresa Metal K, a continuación, se relacionan las diferentes arquitecturas de referencia:

Centro de arquitectura de AWS:

1. **Centralized Log Analytics:** Esta arquitectura de alto nivel destaca las diferentes etapas del flujo de registros y qué herramientas pueden utilizarse en cada una.

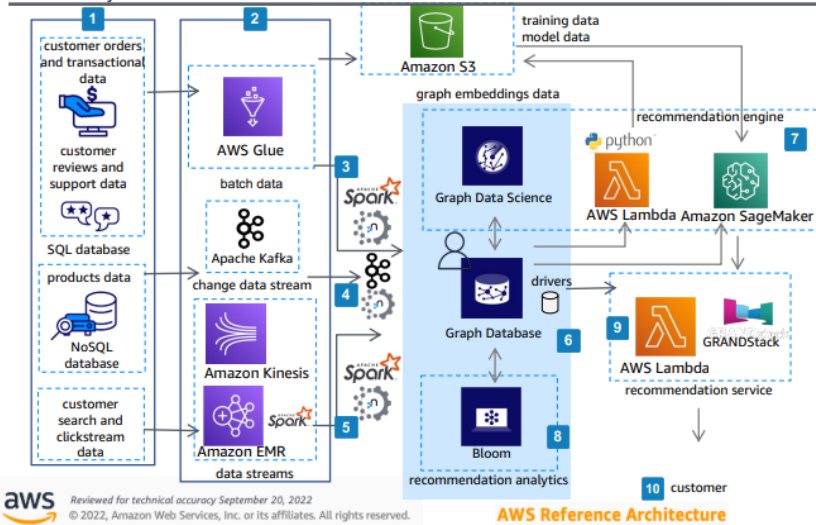


2. **Product Recommendations Powered by Neo4j:** Esta arquitectura muestra cómo puede crear recomendaciones personalizadas e inteligentes de productos relevantes para compradores en casi tiempo real y asignar usuarios a productos, servicios, información u otras personas según su perfil de usuario, preferencias y actividad en línea anterior.

Product Recommendations Powered by Neo4j

Near real-time Personalized Product Recommendations With Neo4j Graph Technology on AWS

Personalized and intelligent recommendations of relevant products for shoppers in near real-time; mapping users to products, services, information, or other people based on their user profile, preferences, and past online activity.

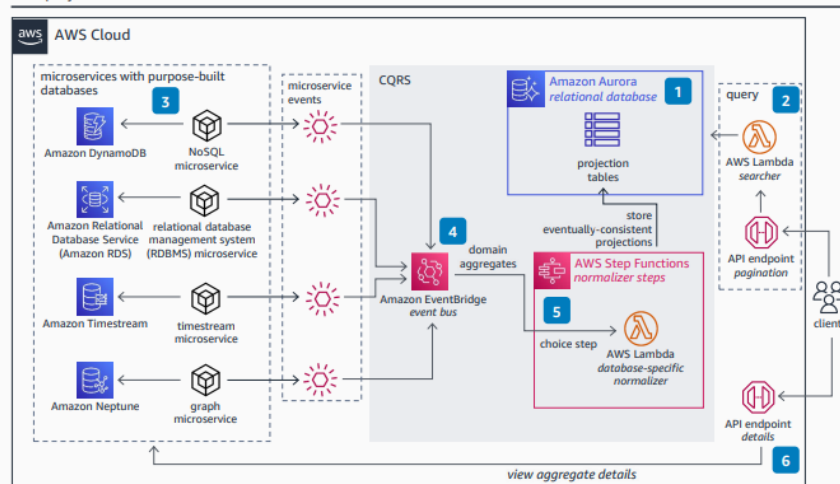


- 1 Customer orders, product data and customer application clickstream data flow through Amazon Redshift, Amazon Relational Database Service (Amazon RDS), Amazon EMR, Amazon Simple Storage Service (Amazon S3), and Amazon Kinesis.
- 2 Data is ingested, transformed, and enriched using AWS Glue, Apache Spark on Amazon EMR, AWS Lambda, or any number of other means.
- 3 Bulk and batch data are loaded to Neo4j using the Neo4j Spark Connector for Amazon EMR or Neo4j APIs (including Cypher and Arrow) running in Lambda.
- 4 Near real-time data from Apache Kafka on AWS by Confluent flows through the Neo4j Kafka Connector to the Neo4j database. Transaction events from Neo4j can also be pushed to Kafka.
- 5 Clickstream data from Amazon Kinesis flows through the Neo4j Spark Connector running on Amazon EMR to the Neo4j database.
- 6 The Neo4j Graph Database (GDB) and Neo4j Graph Data Science (GDS) let you store, query, analyze, and manage highly-connected data. Neo4j Aura is deployed as a fully-managed service on AWS. Neo4j Enterprise runs on Amazon Elastic Compute Cloud (Amazon EC2) infrastructure instances.
- 7 Data scientists create features on Neo4j GDB and GDS. Embeddings are exported to Amazon SageMaker for smarter ML. Data scientists can also use the Spark and Python APIs to implement complex ML on Neo4j. Data scientists can push graph features back to relational source systems where they can be easily visualized or reanalyzed.
- 8 Amazon QuickSight and Neo4j Bloom visualization tools let analysts explore data and present findings plainly.
- 9 Developers use official and unofficial drivers to access Neo4j using Java, Node, JavaScript, C#, Python, Go, Ruby, PHP, Erlang, and Perl from AWS Lambda or any application. Others choose a low-code approach using GraphQL API and the GRANDStack framework, using Amazon API Gateway, AWS CodeDeploy, and other development services.
- 10 The recommended services using Neo4j GDB and GDS run fast, at scale, in near real-time. This framework can be used to build similar recommendation engines for additional use cases.

3. **Paginated Search with Purpose Built Databases:** Esta arquitectura de referencia muestra cómo crear una lista paginada que admite búsquedas de agregados de dominio desde bases de datos personalizadas con una transformación por pasos y consultas SQL.

Paginated Search with Purpose Built Databases

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



- 1 Create a relational database using Amazon Aurora to hold projections of the domain aggregates you need to search, paginate, and sort. Working backwards from the user interfaces, decide how 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. The paginated results include the unique keys of the returned domain aggregates.
- 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 ECS.
- 6 Clients can request details on a returned aggregate, referring to it by a unique key. This is managed by the relevant microservice through the API endpoint.