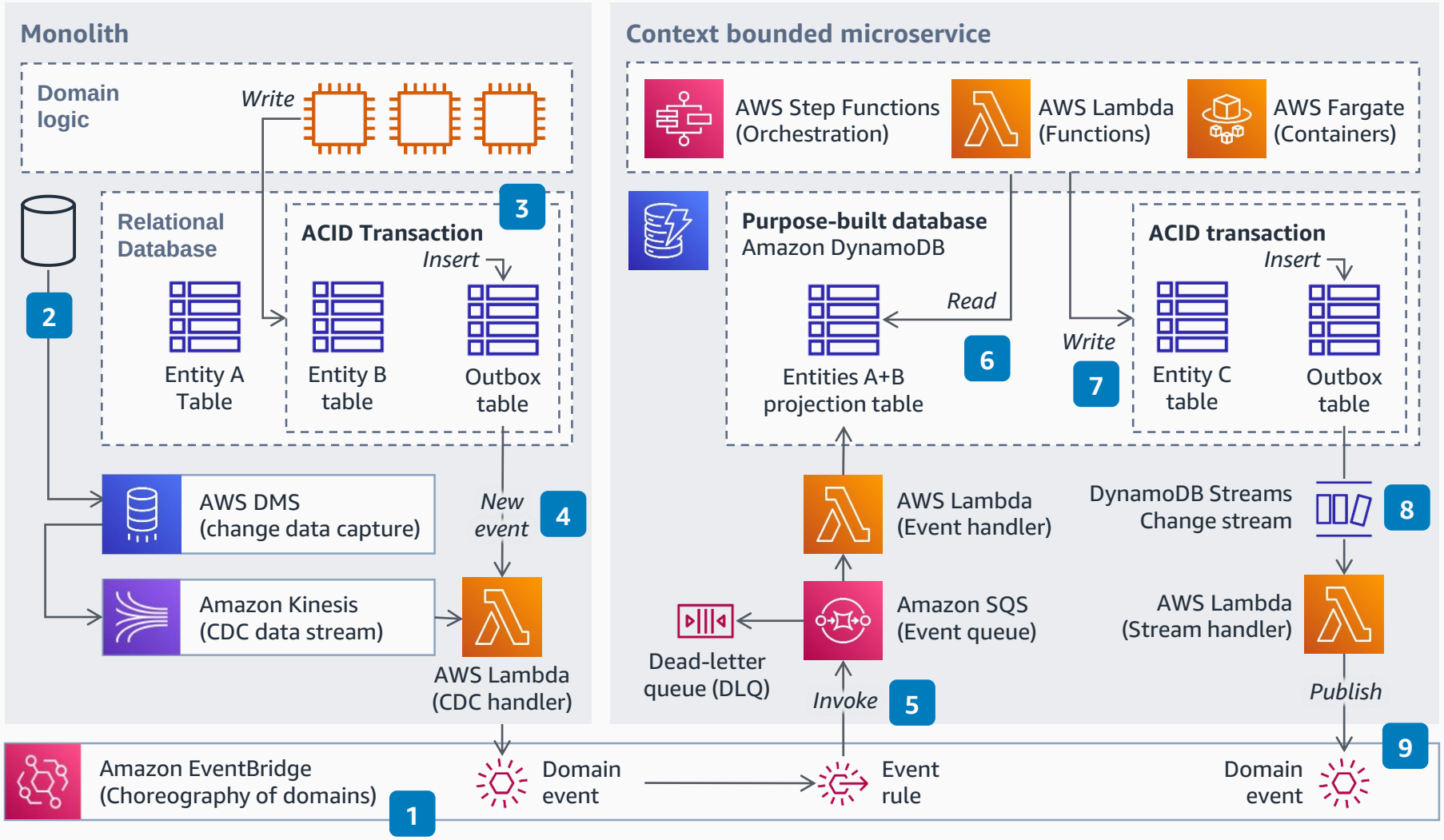
Domain consistency in event-driven architectures

Increase the resilience of your event-driven architecture by applying the transactional outbox pattern to your domain's database transactions, ensuring that all local changes raise a corresponding domain event that other domains can project.



1. To support event choreography between your domains, set up a custom event bus using **Amazon EventBridge**.
2. Detect changes on your monolith's relational database by setting up change data capture (CDC) on the database with **AWS Data Migration Service (AWS DMS)** and **Amazon Kinesis Data Streams**.
3. Create an outbox table to insert events to be published following any database change, and wrap the write of both entities and outbox tables inside a transactional operation with atomicity, consistency, isolation, and durability (ACID).
4. Capture the inserts in the outbox table using an **AWS Lambda** function listening to the **Kinesis** CDC stream, and publish them as domain events in the event bus.
5. On your context bounded microservices, capture the relevant external events by setting up a rule in **EventBridge** to push them into an **Amazon Simple Queue Service (Amazon SQS)** queue, then handle them with a **Lambda** function and an SQS dead-letter queue (DQL).
6. Use the incoming external domain events to build eventually consistent projected representations of those other domains' databases, implemented with **Amazon DynamoDB** tables, for local logic to read.
7. On your microservices, repeat the same transactional mechanism around both entity tables and the event outbox table.
8. Capture the inserts on the event outbox table using the **DynamoDB** data stream.
9. Handle the data stream with a **Lambda** function, publishing the inserts as domain events in the event bus.

Reviewed for technical accuracy March 8, 2022

AWS Reference Architecture