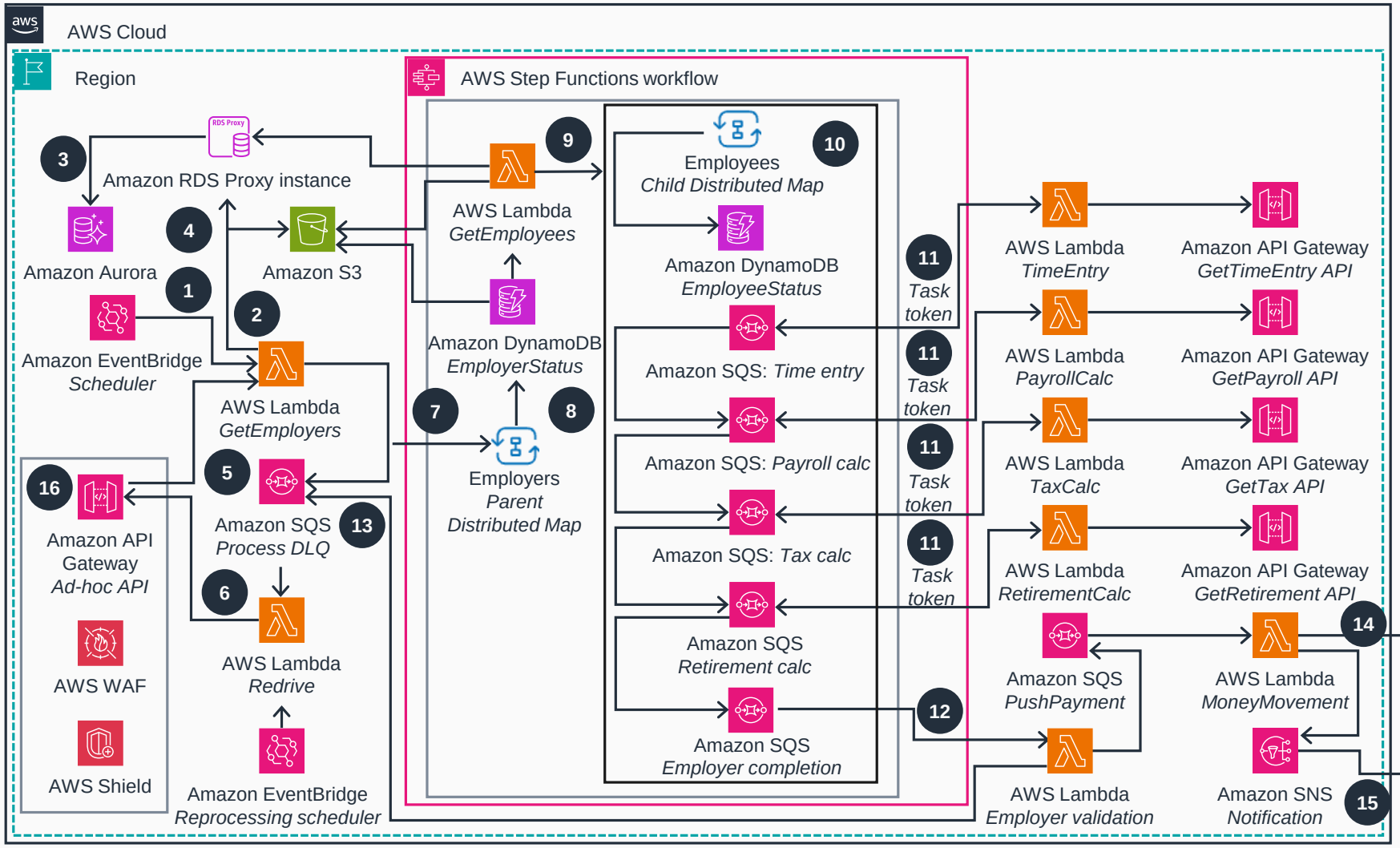
Guidance for Building Persistent and Resilient Event-Driven Patterns for Payroll Systems on AWS

This architecture diagram shows how to implement an accurate, resilient, serverless, and event-driven payroll processing system with exactly-once processing requirements and failure-handling patterns. This slide details steps 1–9; the next slide details steps 10–16.

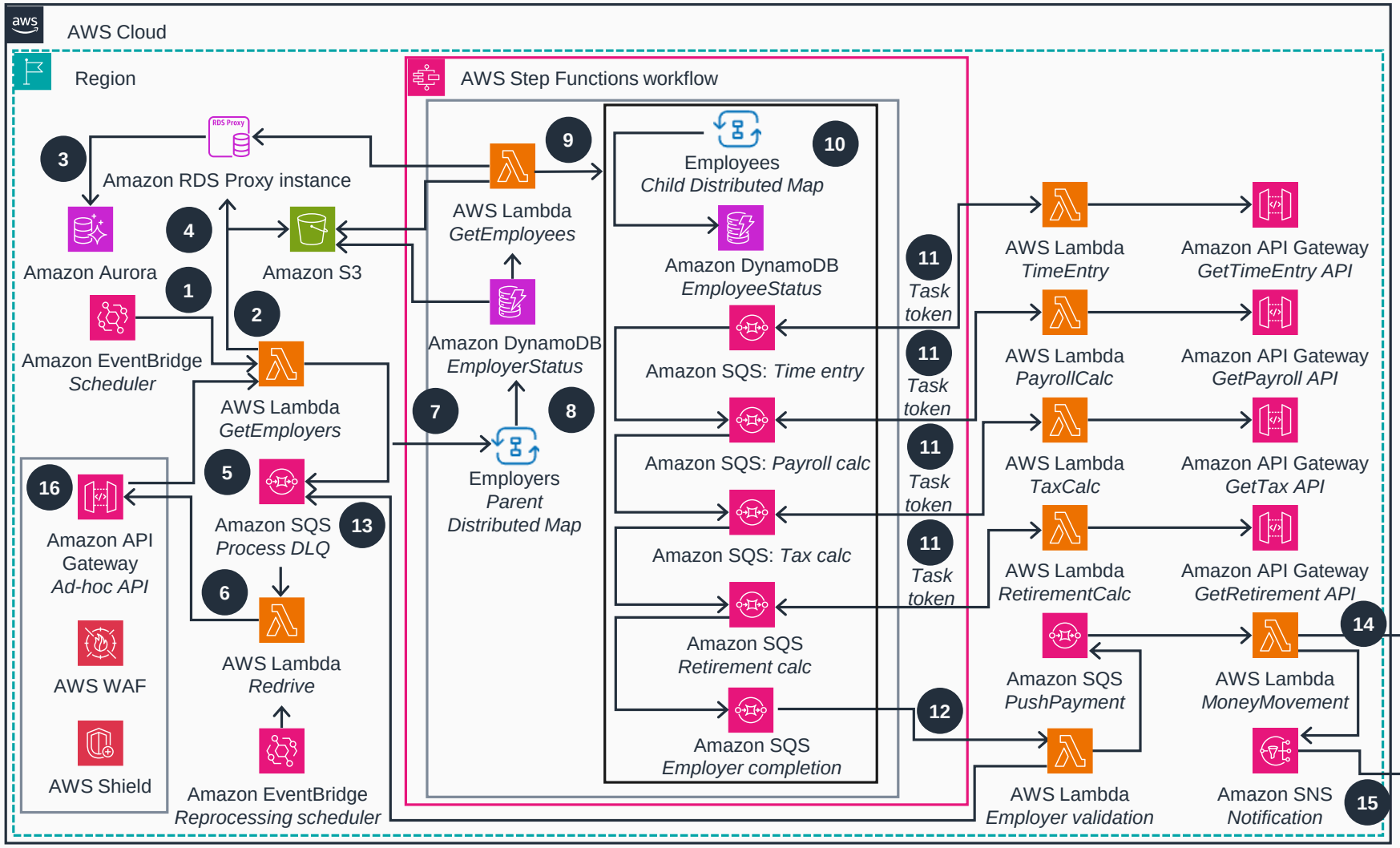


1. A scheduled event invokes the payroll process.
2. The GetEmployers **AWS Lambda** function connects to the **Amazon Relational Database Service (Amazon RDS) Proxy** as a connection pool to scalably connect to downstream relational databases at scale.
3. The GetEmployers **Lambda** function gets employer information from an **Amazon Aurora** database, as this use case relies on a relational data model common for payroll systems.
4. The GetEmployers **Lambda** function stores all the employer information in an **Amazon Simple Storage Service (Amazon S3)** bucket.
5. The Amazon Simple Queue Service (Amazon SQS) dead-letter queue (DLQ) stores messages that failed to get processed, to be processed and reviewed later.
6. The Redrive **Lambda** function, invoked by a scheduled **Amazon EventBridge** cron event, processes failed messages from the **Amazon SQS** DLQ using ad-hoc API calls.
7. The GetEmployers **Lambda** function invokes **AWS Step Functions** after the completion of steps 2–5. This standard **Step Functions** workflow uses two distributed maps: The Child-Distributed Map processes multiple employers in parallel, while the Parent-Distributed Map processes the employees of each employer in parallel.
8. The Parent-Distributed Map state reads employer information from an **Amazon S3** object. The 'Begin' status of processing each employer is written into the **Amazon DynamoDB** EmployerStatus database.
9. The GetEmployees **Lambda** function retrieves employee data for each employer from **Aurora** using **Amazon RDS Proxy**, storing the results as JSON objects in an **Amazon S3** bucket.

AWS Reference Architecture

Guidance for Building Persistent and Resilient Event-Driven Patterns for Payroll Systems on AWS

This slide details steps 10–16.



10 The Child-Distributed Map begins processing the individual employee payroll by running multiple parallel threads. It reads employee information from the **Amazon S3** object (stored in step 9) and starts by posting a 'Begin' status in the EmployeeStatus **DynamoDB** database.

11 Each employee's payroll is processed through a series of **Amazon SQS** queues, which invoke **Lambda** functions using a task token. The task-token approach gives the workflow the flexibility to call the downstream systems asynchronously.

In this step, the responses from downstream systems (like the TimeEntry, PayrollCalc, TaxCalc, and RetirementCalc **Lambda** functions and their corresponding APIs) are returned to corresponding **Amazon SQS** queues using the task token. This completes the task and launches the next step in the **Step Functions** workflow.

12 The Employer completion **Amazon SQS** queue invokes the Employer validation **Lambda** function, which validates that data for all the employees of an employer was processed successfully.

13 If some employees' data was not processed successfully, the Employer validation **Lambda** function sends the failed messages to the **Amazon SQS** Process Dead-Letter Queue (DLQ).

14 If all the employees' payrolls were calculated correctly for the employer, the Employer validation **Lambda** function sends a message to the PushPayment **Amazon SQS** queue. This will initiate the process of making the payments through an external payment system, facilitated by the MoneyMovement **Lambda** function.

15 Once the payment is successful, the MoneyMovement **Lambda** function sends the completion notification to users through **Amazon Simple Notification Service (Amazon SNS)**.

16 The messages previously sent to the **Amazon SQS** process DLQ are processed through one-time via ad-hoc API calls using **Amazon API Gateway**, depending on the type of error and resolution. **AWS WAF** protects those calls against common web exploits and bots that can affect availability, compromise security, or consume excessive resources. **AWS Shield**, a managed distributed denial of service (DDoS) protection service, safeguards applications running on AWS.

AWS Reference Architecture