Persistent

Case Study

US-based ISV utilizes Lambda to automate event processing

About the Client

The customer is an ISV in the United States that wanted to develop an artificial intelligence parenting platform to address various parenthood challenges such as:

\ Anxiety about child's early stage emotional, cognitive well-being, and development.

\ Day-to-day lack of intimacy, engagement, and knowledge of their child's activities.

\ Screen-based technologies and their impact on child's social behavior.

Problem Statement

The customer needed a serverless framework for handling multiple requirements such as codec conversion and event processing for events collected from various mobile, tablet, and desktop application clients to do backend mobile application and video/image processing.

What We Implemented

Overview
The entire event pipeline was developed using Lambda. In total, 25 Lambda functions were developed with 1 million invocations during the beta phase and over 25 million invocations per month in production.

Multiple solutions were implemented using Lambda functions:

\ Codec conversions to convert the mobile video format (3GPP) to a more acceptable format like MP4.

\ User sessionization to find out user activities and priorities in every session.

\ User location updates to send "unknown" location alerts for child safety.

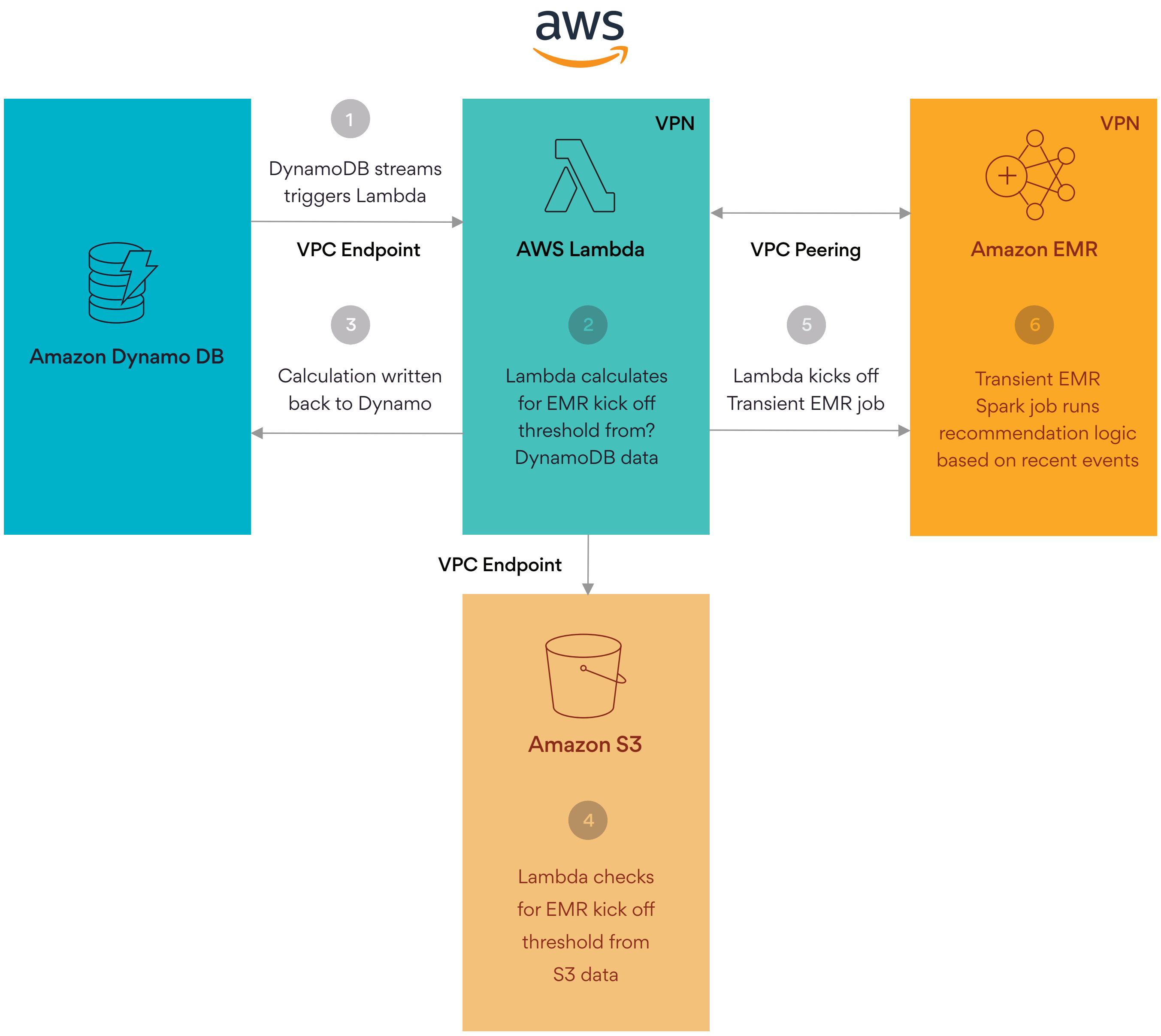
\ Thumbnail generation for uploaded images captured by camera from application clients.

\ Event pipeline processing to aggregate likes, views on the media contents.

\ EFK (ElasticSearch-Fluent-Kibana) pipeline for debugging, log collection and analysis.

\ Running transient EMR Spark jobs for recommendations based on certain number of events in DynamoDB to run a machine learning (ML) model for content recommendations based on recency.

AWS Lambda is triggered from many other AWS services such as S3, DynamoDB Streams, API Gateway, SQS, SNS, SES, Cloudwatch, and ElasticSearch. Lambda output is written to S3, DynamoDB, RDS, SNS, ElasticSearch, and Transient EMR jobs. Data processing on events comes from API Gateway and video rendering events comes from S3. DynamoDB captures events emitted by all application clients. Transient EMR Spark jobs are run to train recommendation models for the recent events. DynamoDB stream batches will trigger Lambda functions which aggregates events per customer and writes back to DynamoDB. Once it reaches the threshold of events per customer, Lambda checks if enough data for customers is available in S3, then calls a transient EMR Spark job to train the recommendation model per customer based on recent events.


aws
Amazon Dynamo DB
1
DynamoDB streams triggers Lambda
VPC Endpoint
3
Calculation written back to Dynamo
VPN
AWS Lambda
2
Lambda calculates for EMR kick off threshold from? DynamoDB data
VPC Peering
5
Lambda kicks off Transient EMR job
VPN
Amazon EMR
6
Transient EMR Spark job runs recommendation logic based on recent events
VPC Endpoint
Amazon S3
4
Lambda checks for EMR kick off threshold from S3 data

Third party applications and solutions were used such as ffmpeg for video encoding, open source python libraries for thumbnail generation, Google Cloud APIs - Cloud Vision, Places, Location (Geocode and reverse Geocoding), Natural Language, Books APIs, and other third-party APIs – Wikipedia, Oxford Dictionary, Meetup, Eventbrite.

Many AWS best practices were utilized on the project such as using VPC endpoint to access S3 and DynamoDB, using VPC peering to access EMR, creating transient EMR, using DynamoDB streams batch as trigger, utilizing Lambda within VPC accessing EMR over VPC peered network, using important configurations in S3 bucket or AWS Parameter Store and not in the code, using Python or NodeJS to avoid code start issues, and instrumenting Lambda using x-ray for tracing.

Solution Characteristics
The Persistent team did basic performance testing for workloads expected during the limited beta production phase. Resource based IAM policies were defined for other AWS services to invoke Lambda functions with one IAM per function. Failed Lambda functions writes the payload to an SNS topic and sends a CloudWatch alert and management of multiple microservice functions was done using Github as code repository. The concurrency limit is reserved for each function, so the impact is isolated to only that function if the number of events surges for any reason.

CloudWatch alarms are setup to notify when events such as concurrent executions or invocations exceed the threshold. Monitoring of alarms and metrics of AWS Lambda functions is done using CloudWatch triggers and all types of CloudWatch metrics — performance, concurrency and invocation are used.

Lessons Learned
The Persistent team learned and implemented the following best practices:

\ SNS for inter Lambda communication

\ VPC endpoint to securely connect to S3 and DynamoDB

\ DynamoDB streams batch mode to trigger Lambda

\ CI/CD for Lambda so that code can be version controlled and deployed automatically

\ Invoking transient EMR jobs from Lambda

Outcomes and Benefits

Originally, the customer had an objection for using Lambda as they wanted to keep platform implementation cloud neutral, so that it can be ported to other public clouds should the need arise. The Persistent team overcame this by keeping Lambda code in Github and having a CI/CD process to **automate Lambda deployment**. Since code was written in Python, the **porting** on other cloud equivalent services should be **easy**.

The platform was built on Persistent's five pillars of operational excellence; security, reliability, performance, efficiency, and cost optimization. After the project was complete, **new event processing requirements could be added** for an existing event or new event without impacting the existing pipeline. The platform **saves 30% of effort** in onboarding and processing new events. The Persistent team calls this component the "Lambda Factory" in the platform.