Serverless, Lambdas, DynamoDB, Cognito, API Gateway

Lambda function

Why is serverless good? Well if you are using EC2 instances then you have to provision them if you need more compute power. You will be paying for those servers that are continuously running. It has limited RAM and CPU.

Lambda function you don't have to manage servers, you just need to write code then deploy it and it will run on-demand. And the scaling is automatic.

Can be triggered with many of AWS services.

Language support for lambda

There are a ton of support language supported for Lambda, even Rust! You can even write your own custom runtime API.

It can even use container image, but the container must conform to the lambda runtime API, but use ECS / Fargate for running containers.

Pricing on lambda

You will be paying per function invoke and the amount of execution time the function took.

$0.20 per 1 million request

$1.00 for 600,000 GB-second

So it is very cheap

Lambda integrations

API Gateway: Create a REST API endpoint to invoke lambda functions

DynamoDB: Triggers for lambda whenever there is a db update

S3: When files are stored into S3 it will have a trigger

Event Bridge: Can set up CRON task to run a function every hour say. What statement-pdf-generator used

Lots of services!

Lambda limits

The limits are per region! For each account.

The **max memory** you can allocate is 128 MB - **max of 10 GB**

The maximum time for execution is **15 minutes,** anything above that lambda isn't a good use case

4 KB of environment variables

Disk capacity of 10 GB max for temporary storage, used to load in big functions in /tmp

Concurrency execution is 1000, but can be requested to increase it. This is how many lambda function can be executed each second at the same time.

Use /tmp for uploading your code if it exceed 50 MB of compressed size, or 250 MB of uncompressed size.

Edge functions

A code that you write and attach to CloudFront distribution (CDN). The idea is that these function will run close to user to minimize latency.

Two types CloudFront functions and Lambda@Edge. These functions are deployed globally. This is used to customize CDN content that's coming out of CloudFront.

CloudFront function

Viewer request and viewer response. So basically client will sent request to CloudFront and CloudFront will make the request on behalf of the viewer to the origin, then give the response back to the client.

High performance and used for customizing CDN.

This is native feature to CloudFront and you would write code in JavaScript

Very quick response < 1 ms for execution time.

Lambda@Edge

Function write in NodeJS or Python, this is used to change CloudFront requests and responses. **Changing the origin request and origin request before it is sent as viewer response back to the client.**

Let your run code that's closer to the user.

Write the function in one region before it is replicated to other locations.

Can do lots of things in execution time 5 - 10 seconds.

Use cases

1. CloudFront functions: Cache key normalization, header manipulation, URL rewrites and redirect, request authentication and authorization
2. Lambda@Edge: Longer execution time, adjustable CPU or memory, Network access to use external services for processing, file system access or access to body of HTTP request

Lambda in VPC

Lambda launch outside of your own VPC, they launch in AWS owned VPC. So lambda cannot access resources in your VPC like RDS, ElastiCache, internal ELB.

You have to launch your lambda in your VPC to have access to those private resources. Then it will have an ENI to access private subnet.

Lambda with RDS proxy to poll database connections, this is so that it doesn't open too many connection and closing it quickly. You make lambda connect to the proxy which keeps the database connection open and is used by the lambda functions to minimize connection to RDS instance.

RDS proxy is never public so you have to launch your lambda in your VPC to take advantage of pooling db request.

Amazon DynamoDB

Fully managed database just like RDS. However, rather than MySQL it is NoSQL database, it is not relational database.

Use case: Scale to massive workloads since it is internally distributed. Millions of requests per second, trillion of rows.

You get single-digit milliseconds fast and consistent performance!

Security is managed via IAM. Low cast and always available!

Standard and infrequent access table class

Basics

DynamoDB is made of tables, the database already exists don't need to create.

Each table have primary key, then you add rows. Attributes are columns and can be added over time very easily.

Max size of an item is 400 KB, so it can't store big object.

Great choice if your schema needs to rapidly evolve, because updating RDS schema is complicated process.

Capacity modes

Provisioned mode: provision the capacity how much read/write you need per second in advance.

For steady smooth workloads.

But you can auto-scaling RCU and WCE (Write capacity unit). Increase unit when needed decrease unit if not needed.

On-demand mode: Read and write capacity scale automatically based on your workload. But this is more expensive compared to provisioned mode since you paying for read/write that your app performed. **For unpredictable workloads and sudden spikes.**

DynamoDB Accelerator

Fully managed highly available in-memory cache for DynamoDB.

This solve for read congestion by caching. Microsecond latency!

No need to rewrite any logic modification.

DynamoDB accelerator vs ElastiCache

ElastiCache is used for aggregation result.

DAX is for individual object cache (for rows that you retrieve)

DynamoDB stream processing

Stream processing allow you to stream each item level modification.

Allow you to react to changes in real-time as your table changes. You can invoke AWS lambda on changes to your DynamoDB table.

DynamoDB streams: 24 hour retention, process using lambda triggers or kinesis adapter. **But can only be consumed by limited number of consumers**

Kinesis data streams: Sent the changes to Kinesis data stream, higher retention of 1 year, higher consumer, the consumer can be AWS lambda, data analytics, firehose.

DynamoDB global table

Global table is replicated cross region. You can write to either table and it will be replicated.

Global table is to make table accessible with low latency in multiple-regions. Application can read and write table in any region. This is active-active replication.

Have to enable DynamoDB stream to have replication possible.

DynamoDB Time to live

Automatically delete items after some time. As soon as the time expires then the item is deleted.

This is good for **web session handling**. Their session can have a expiry time then the session data is deleted after.

Backups for disaster recovery

Continuous backups using point-in-time recovery. Enable for last 35 days. The recovery creates new table

On-demand backups: Full backup for long-term retention until deleted explicitly.

Integration with S3

You can export table to S3, PITR must be enabled. This is so that you can do query on S3 with Athena.

This can also be a snapshot of your table.

Export DYNAMODB to JSON or ION format. Then you can import it back from S3 using CSV, JSON OR ion.

Amazon API Gateway

So there are multiple ways that a lambda gets triggered by the client. Client directly invoking it, front the lambda with an ALB to expose the lambda as a HTTP endpoint. There is anothe rway.

API Gateway is a serverless service that create REST API endpoint which can proxy request to the lambda, triggering it. You will get a DNS endpoint for invoking the API after you deploy it.

Proxy integration enrich the event for your integration. For example, it will enrich the event that's passed into your lambda so that it can use it to generate correct response.

Features

API Gateway handles API versioning, v1, v2,...

It also handles security for authentication and authorization, and handles different environments.

Transform and validate requests and response. Generate SDK and API specification, and cache API responses. This provides way more feature than a simple ALB that front the lambda function.

Combination

You usually use gateway with lambda function.

But it can also be combine with HTTP endpoint in the backend. This adds rate limiting, caching, user authentication

Combine with any AWS service: **You can expose any AWS API through the API gateway**, start AWS step function workflow, post a message to SQS. You would do this to add authentication, deploy your services publicly via a REST API. Basically you can expose your existing AWS resources via REST endpoint to the public.

Endpoint types

Edge-optimized (default): Make your API gateway accessible anywhere in the world. Your gateway will only be in one region, but request are routed from CloudFront to your API gateway to improve latency

Regional: You expect all user in the region you deploy in. But you can manually combine it with CloudFront to have more control/

Private: Not public, only accessed within your VPC using ENI.

Security

Authentication: You can authenticate user using IAM roles, (EC2 instance that wants to access API gateway). Cognito for external users. Or you can implement your own authorizer using lambda functions.

Custom domain name HTTPS: Integrated with AWS Certificate Manager, if you want HTTPS on your endpoint then you need to install certificate. For edge-optimize install it in us-east-1, regional then install it in the region you are exposing the gateway. You have to setup the CNAME and A record in Route 53.

Step Function

You use visual workflow to orchestrate your lambda function.

It can actually integrate with other services besides lambda function like EC2, ECS, API gateway!

You can have work flow of your data that goes through the step function, and then processes it at every stage, then also have human approval at certain stages.

Use for: Data processing, order fulfillment, any workflow that need graph to visualize.

Amazon Cognito

Give user an identity to interact with web and mobile application. They don't have AWS accounts but still want to use your resources that you have deploy.

Cognito user pools: Sign-in functionality for app user

Cognito identity pools (Federated identity, what we are using!): To give temporary AWS credentials to user to access resources directly.

Cognito VS IAM: Cognito is used for mobile and web application that sits outside of AWS, can also used for hundreds of users and authenticate with whatever you want.

Cognito user pools

Serverless database of user for your web and mobile apps. Can have simple login and password reset. MFA. Federated identities (users from other platform)

You can use CUP with API Gateway and ALB to authenticate user.

Cognito identity pools

Give direct AWS account using temporary AWS credentials.

This is how Cloudsentry give us that temporary AWS credential access.

You can use CIP with CUP to retrieve temporary access. And provide fine-grained, row access to DynamoDB. So that the user cannot read/write every row in the DynamoDB.

Revision #7
Created 2023-02-24 04:09:50 UTC by Tamarine
Updated 2023-02-28 17:43:48 UTC by Tamarine