


Serverless Overview

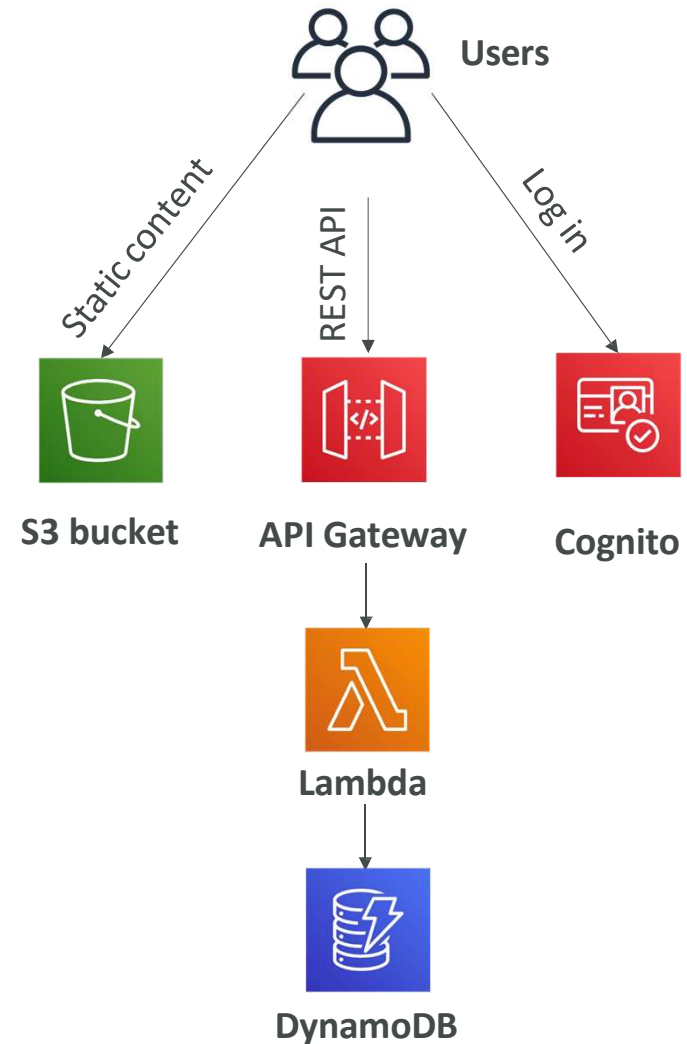


What's serverless?

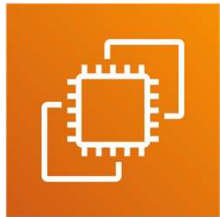
- Serverless is a new paradigm in which the developers don't have to manage servers anymore...
 - They just deploy code
 - They just deploy... functions !
 - Initially... Serverless == FaaS (Function as a Service)
 - Serverless was pioneered by AWS Lambda but now also includes anything that's managed: "databases, messaging, storage, etc."
 - Serverless does not mean there are no servers...
it means you just don't manage / provision / see them
- 

Serverless in AWS

- AWS Lambda
- DynamoDB
- AWS Cognito
- AWS API Gateway
- Amazon S3
- AWS SNS & SQS
- AWS Kinesis Data Firehose
- Aurora Serverless
- Step Functions
- Fargate



Why AWS Lambda



Amazon EC2


- Virtual Servers in the Cloud
- Limited by RAM and CPU
- Continuously running
- Scaling means intervention to add / remove servers




Amazon Lambda

- Virtual functions - no servers to manage!
- Limited by time - short executions
- Run on-demand
- Scaling is automated!

Benefits of AWS Lambda

- Easy Pricing:
 - Pay per request and compute time
 - Free tier of 1,000,000 AWS Lambda requests and 400,000 GBs of compute time
 - Integrated with the whole AWS suite of services
 - Integrated with many programming languages
 - Easy monitoring through AWS CloudWatch
 - Easy to get more resources per functions (up to 10GB of RAM!)
 - Increasing RAM will also improve CPU and network!
- 

AWS Lambda language support

- Node.js (JavaScript)
 - Python
 - Java
 - C# (.NET Core) / Powershell
 - Ruby
 - Custom Runtime API (community supported, example Rust or Golang)
 - Lambda Container Image
 - The container image must implement the Lambda Runtime API
 - ECS / Fargate is preferred for running arbitrary Docker images
- 

AWS Lambda Integrations

Main ones



API Gateway



Kinesis



DynamoDB



S3



CloudFront



CloudWatch Events
EventBridge



CloudWatch Logs



SNS

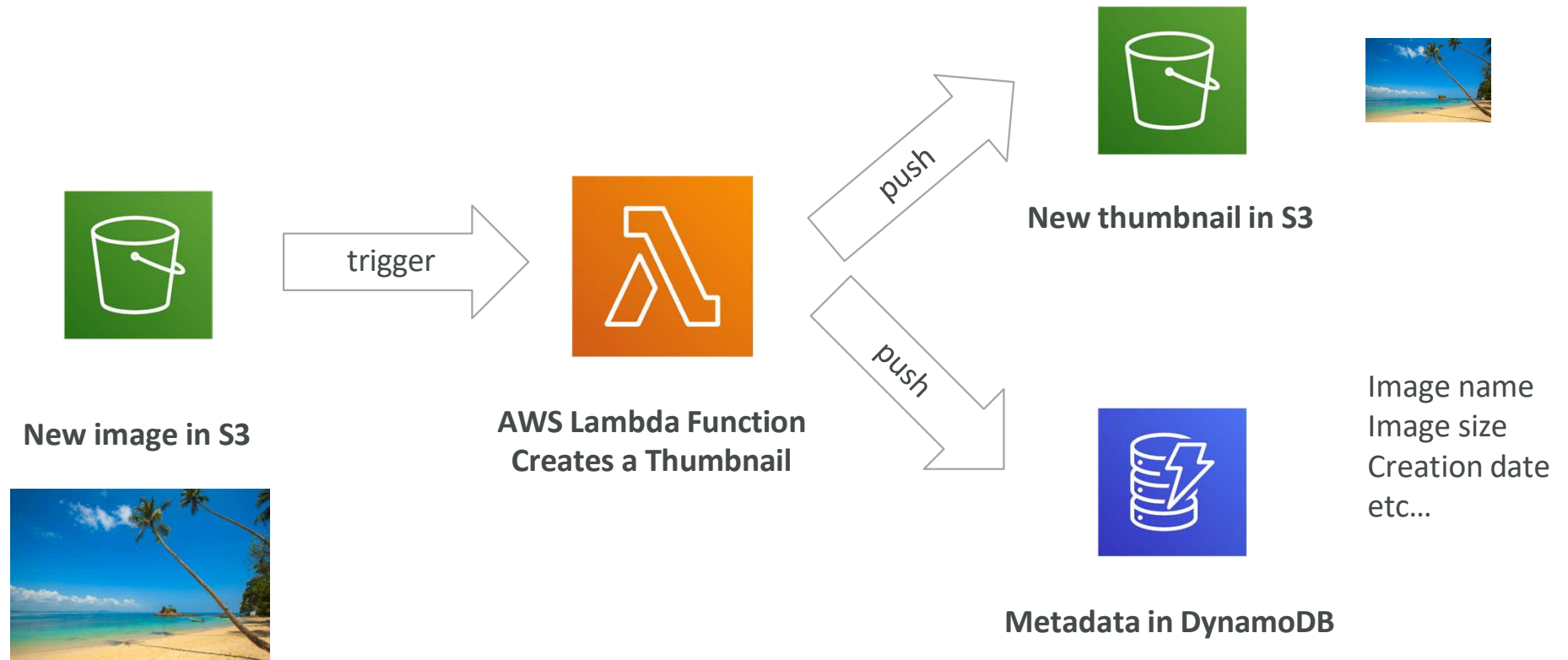


SQS



Cognito

Example: Serverless Thumbnail creation



Example: Serverless CRON Job



AWS Lambda Pricing: example


- You can find overall pricing information here:
<https://aws.amazon.com/lambda/pricing/>
- Pay per calls:
 - First 1,000,000 requests are free
 - \$0.20 per 1 million requests thereafter (\$0.0000002 per request)
- Pay per duration: (in increment of 1 ms)
 - 400,000 GB-seconds of compute time per month for ~~FREE~~
 - == 400,000 seconds if function is 1GB RAM
 - == 3,200,000 seconds if function is 128 MB RAM
 - After that \$1.00 for 600,000 GB-seconds
- It is usually very cheap to run AWS Lambda so it's very popular

AWS Lambda Limits to Know - per region

- Execution:

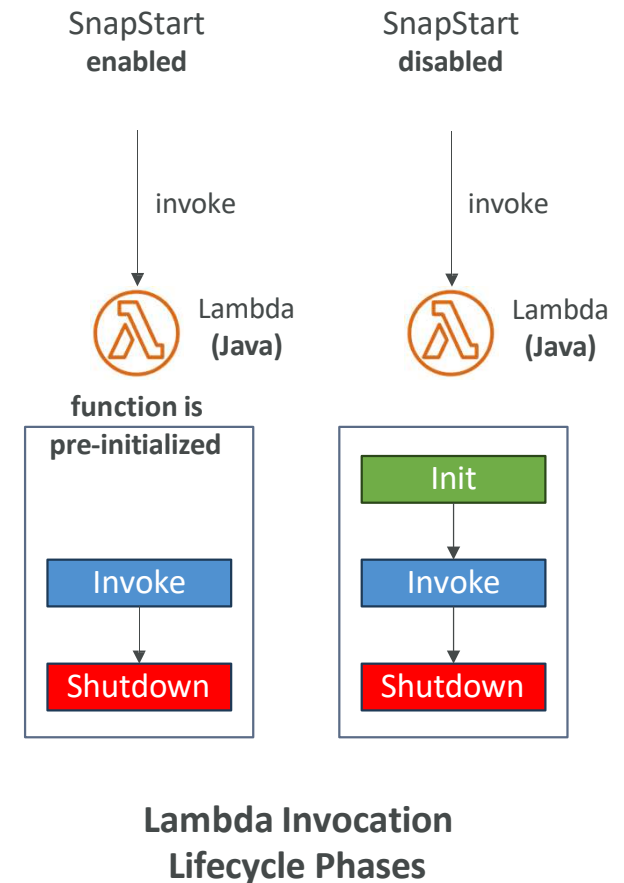
- Memory allocation: 128 MB - 10GB (1 MB increments)
- Maximum execution time: 900 seconds (15 minutes)
- Environment variables (4 KB)
- Disk capacity in the “function container” (in /tmp): 512 MB to 10GB
- Concurrency executions: 1000 (can be increased)

- Deployment:

- Lambda function deployment size (compressed .zip): 50 MB
 - Size of uncompressed deployment (code + dependencies): 250 MB
 - Can use the /tmp directory to load other files at startup
 - Size of environment variables: 4 KB
- 

Lambda SnapStart

- Improves your Lambda functions performance up to 10x at no extra cost for Java 11 and above
- When enabled, function is invoked from a pre-initialized state (no function initialization from scratch)
- When you publish a new version:
 - Lambda initializes your function
 - Takes a snapshot of memory and disk state of the initialized function
 - Snapshot is cached for low-latency access



Customization At The Edge



- Many modern applications execute some form of the logic at the edge
- Edge Function:
 - A code that you write and attach to CloudFront distributions
 - Runs close to your users to minimize latency
- CloudFront provides two types: CloudFront Functions & Lambda@Edge
- You don't have to manage any servers, deployed globally
- Use case: customize the CDN content
- Pay only for what you use
- Fully serverless

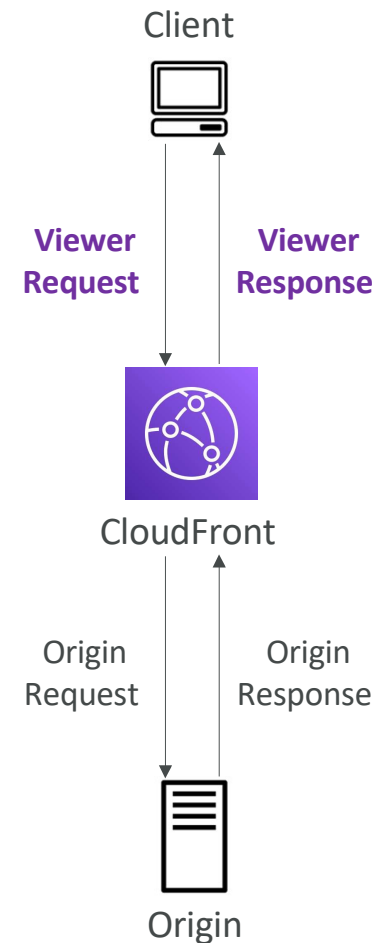
CloudFront Functions & Lambda@Edge Use Cases



- Website Security and Privacy
- Dynamic Web Application at the Edge
- Search Engine Optimization (SEO)
- Intelligently Route Across Origins and Data Centers
- Bot Mitigation at the Edge
- Real-time Image Transformation
- A/B Testing
- User Authentication and Authorization
- User Prioritization
- User Tracking and Analytics

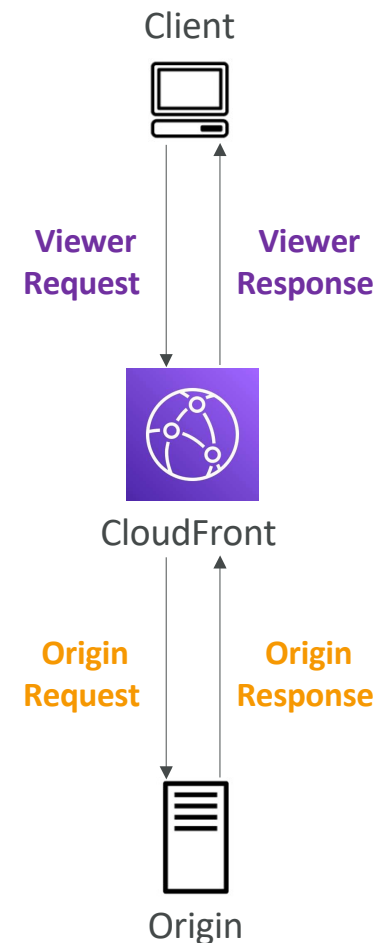
CloudFront Functions

- Lightweight functions written in JavaScript
- For high-scale, latency-sensitive CDN customizations
- Sub-ms startup times, millions of requests/second
- Used to change Viewer requests and responses:
 - Viewer Request: after CloudFront receives a request from a viewer
 - Viewer Response: before CloudFront forwards the response to the viewer
- Native feature of CloudFront (manage code entirely within CloudFront)



Lambda@Edge

- Lambda functions written in NodeJS or Python
- Scales to 1000s of requests/second
- Used to change CloudFront requests and responses:
 - Viewer Request - after CloudFront receives a request from a viewer
 - Origin Request - before CloudFront forwards the request to the origin
 - Origin Response - after CloudFront receives the response from the origin
 - Viewer Response - before CloudFront forwards the response to the viewer
- Author your functions in one AWS Region (us-east-1), then CloudFront replicates to its locations



CloudFront Functions vs. Lambda@Edge

	CloudFront Functions	Lambda@Edge
Runtime Support	JavaScript	Node.js, Python
# of Requests	Millions of requests per second	Thousands of requests per second
CloudFront Triggers	- Viewer Request/Response	- Viewer Request/Response - Origin Request/Response
Max. Execution Time	< 1 ms	5 – 10 seconds
Max. Memory	2 MB	128 MB up to 10 GB
Total Package Size	10 KB	1 MB – 50 MB
Network Access, File System Access	No	Yes
Access to the Request Body	No	Yes
Pricing	Free tier available, 1/6 th price of @Edge	No free tier, charged per request & duration

CloudFront Functions vs. Lambda@Edge - Use Cases

CloudFront Functions

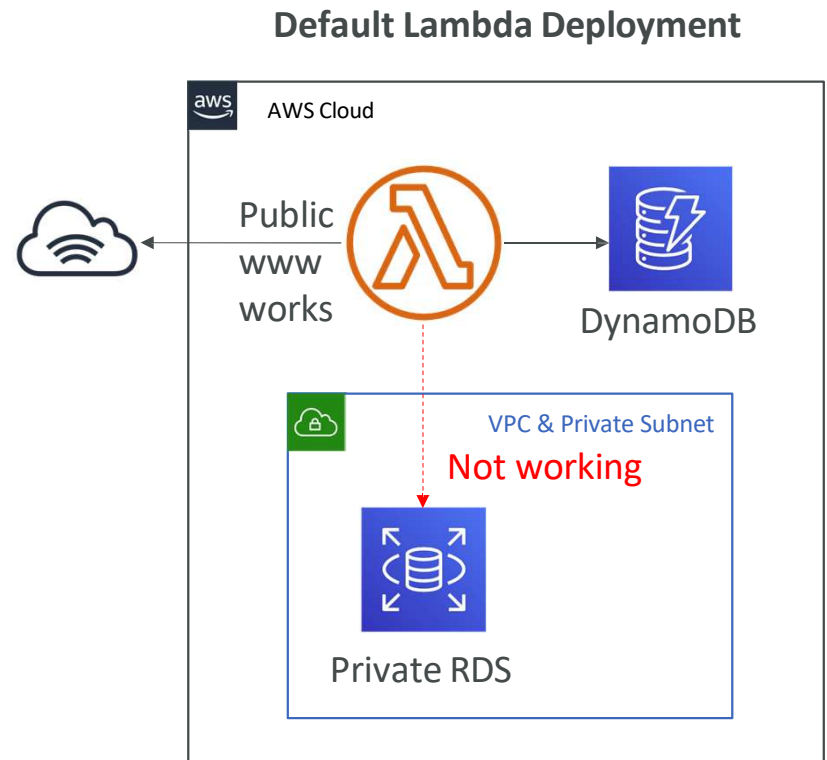
- Cache key normalization
 - Transform request attributes (headers, cookies, query strings, URL) to create an optimal Cache Key
- Header manipulation
 - Insert/modify/delete HTTP headers in the request or response
- URL rewrites or redirects
- Request authentication & authorization
 - Create and validate user-generated tokens (e.g., JWT) to allow/deny requests

Lambda@Edge

- Longer execution time (several ms)
- Adjustable CPU or memory
- Your code depends on a 3rd libraries (e.g., AWS SDK to access other AWS services)
- Network access to use external services for processing
- File system access or access to the body of HTTP requests

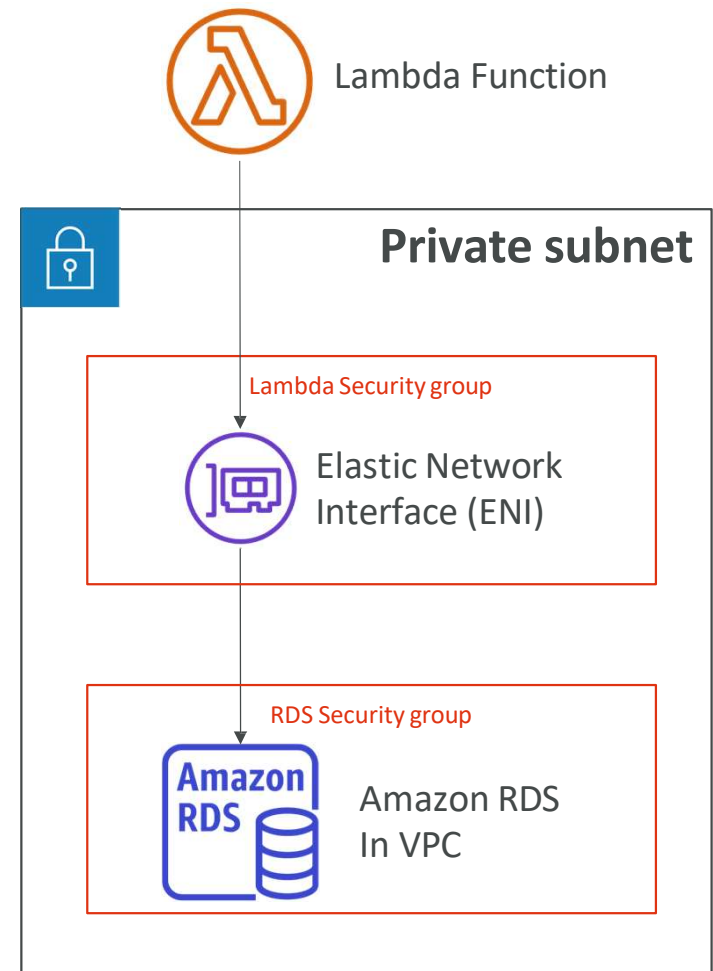
Lambda by default

- By default, your Lambda function is launched outside your own VPC (in an AWS-owned VPC)
- Therefore, it cannot access resources in your VPC (RDS, ElastiCache, internal ELB...)



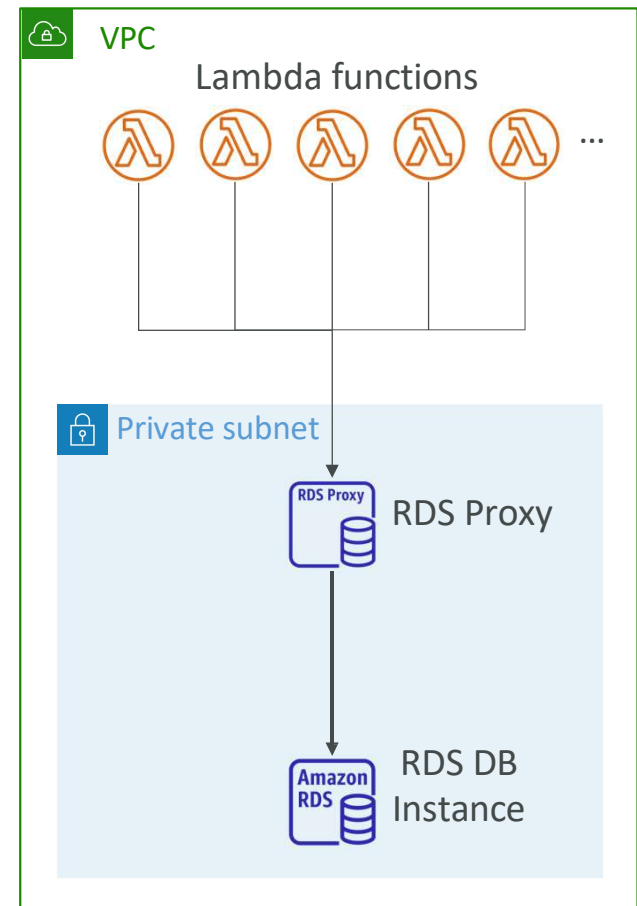
Lambda in VPC

- You must define the VPC ID, the Subnets and the Security Groups
- Lambda will create an ENI (Elastic Network Interface) in your subnets



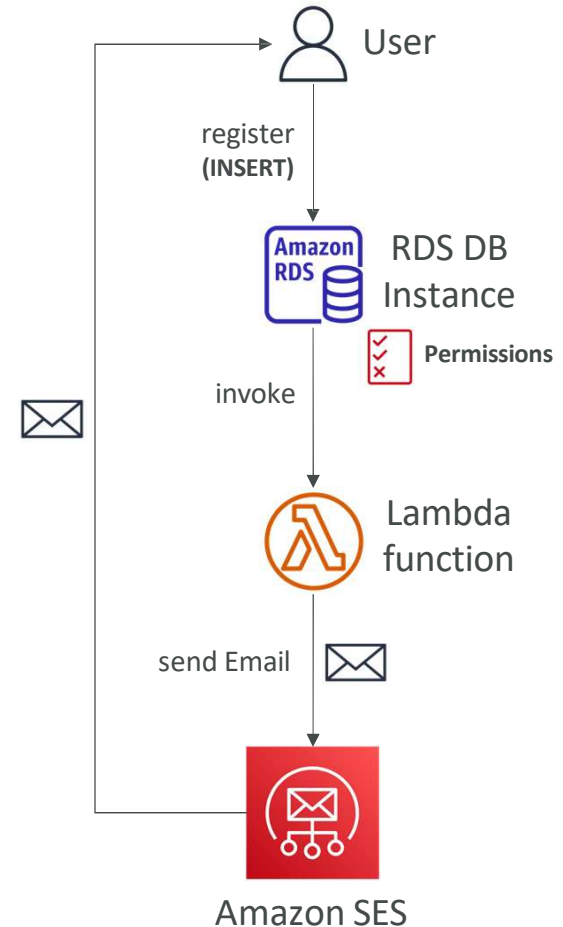
Lambda with RDS Proxy

- If Lambda functions directly access your database, they may open too many connections under high load
- RDS Proxy
 - Improve scalability by pooling and sharing DB connections
 - Improve availability by reducing by 66% the failover time and preserving connections
 - Improve security by enforcing IAM authentication and storing credentials in Secrets Manager
- The Lambda function must be deployed in your VPC, because RDS Proxy is never publicly accessible



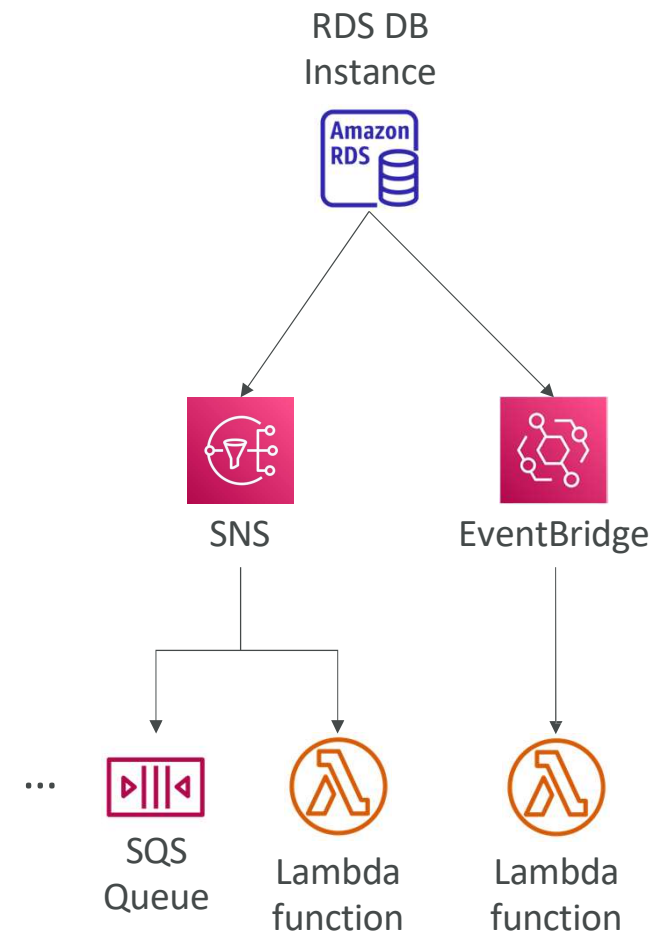
Invoking Lambda from RDS & Aurora

- Invoke Lambda functions from within your DB instance
- Allows you to process data events from within a database
- Supported for RDS for PostgreSQL and Aurora MySQL
- Must allow outbound traffic to your Lambda function (Public, NAT GW, VPC Endpoints)
- DB instance must have the required permissions to invoke the Lambda function (Lambda Resource-based Policy & IAM Policy)



RDS Event Notifications

- Notifications that tells information about the DB instance itself (created, stopped, start, ...)
- You don't have any information about the data itself
- Subscribe to the following event categories: DB instance, DB snapshot, DB Parameter Group, DB Security Group, RDS Proxy, Custom Engine Version
- Near real-time events (up to 5 minutes)
- Send notifications to SNS or subscribe to events using EventBridge

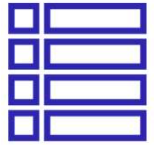


Amazon DynamoDB



- Fully managed, highly available with replication across multiple AZs
- NoSQL database - not a relational database - with transaction support
- Scales to massive workloads, distributed database
- Millions of requests per seconds, trillions of row, 100s of TB of storage
- Fast and consistent in performance (single-digit millisecond)
- Integrated with IAM for security, authorization and administration
- Low cost and auto-scaling capabilities
- No maintenance or patching, always available
- Standard & Infrequent Access (IA) Table Class

DynamoDB - Basics




- DynamoDB is made of Tables
- Each table has a Primary Key (must be decided at creation time)
- Each table can have an infinite number of items (= rows)
- Each item has attributes (can be added over time - can be null)
- Maximum size of an item is 400KB
- Data types supported are:
 - Scalar Types - String, Number, Binary, Boolean, Null
 - Document Types - List, Map
 - Set Types - String Set, Number Set, Binary Set
- Therefore, in DynamoDB you can rapidly evolve schemas

DynamoDB - Table example

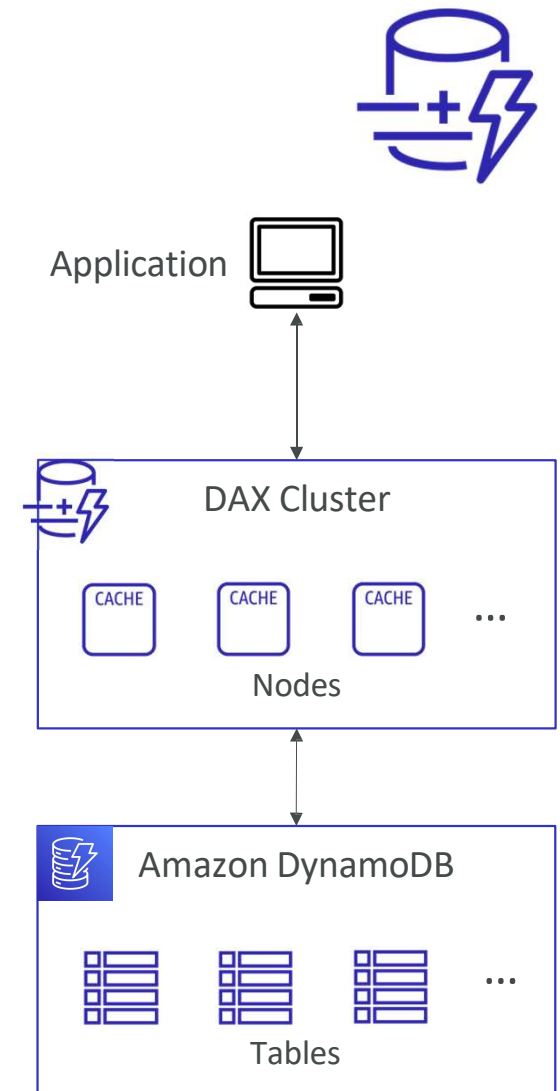
Primary Key		Attributes	
Partition Key	Sort Key		
User_ID	Game_ID	Score	Result
7791a3d6-...	4421	92	Win
873e0634-...	1894	14	Lose
873e0634-...	4521	77	Win

DynamoDB – Read/Write Capacity Modes

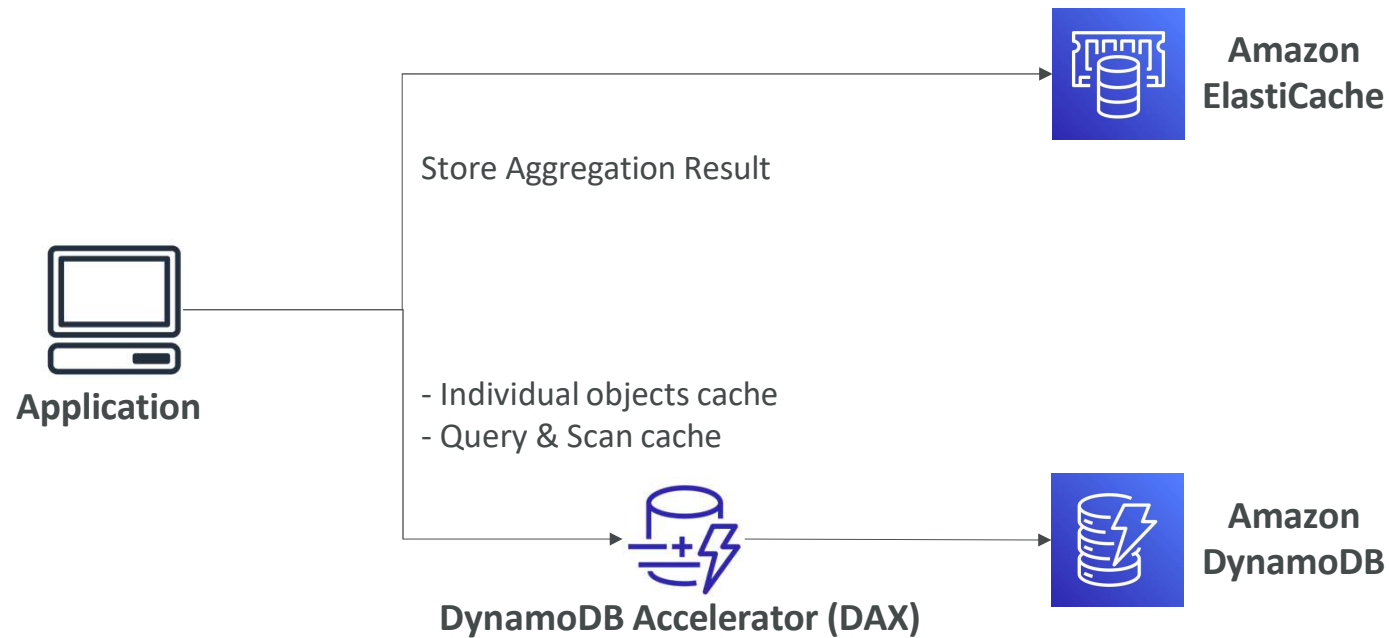
- Control how you manage your table's capacity (read/write throughput)
 - Provisioned Mode (default)
 - You specify the number of reads/writes per second
 - You need to plan capacity beforehand
 - Pay for provisioned Read Capacity Units (RCU) & Write Capacity Units (WCU)
 - Possibility to add auto-scaling mode for RCU & WCU
 - On-Demand Mode
 - Read/writes automatically scale up/down with your workloads
 - No capacity planning needed
 - Pay for what you use, more expensive (\$\$\$)
 - Great for unpredictable workloads, steep sudden spikes
- 

DynamoDB Accelerator (DAX)

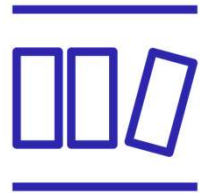
- Fully-managed, highly available, seamless in-memory cache for DynamoDB
- Help solve read congestion by caching
- Microseconds latency for cached data
- Doesn't require application logic modification (compatible with existing DynamoDB APIs)
- 5 minutes TTL for cache (default)



DynamoDB Accelerator (DAX) vs. ElastiCache



DynamoDB - Stream Processing



- Ordered stream of item-level modifications (create/update/delete) in a table
- Use cases:
 - React to changes in real-time (welcome email to users)
 - Real-time usage analytics
 - Insert into derivative tables
 - Implement cross-region replication
 - Invoke AWS Lambda on changes to your DynamoDB table

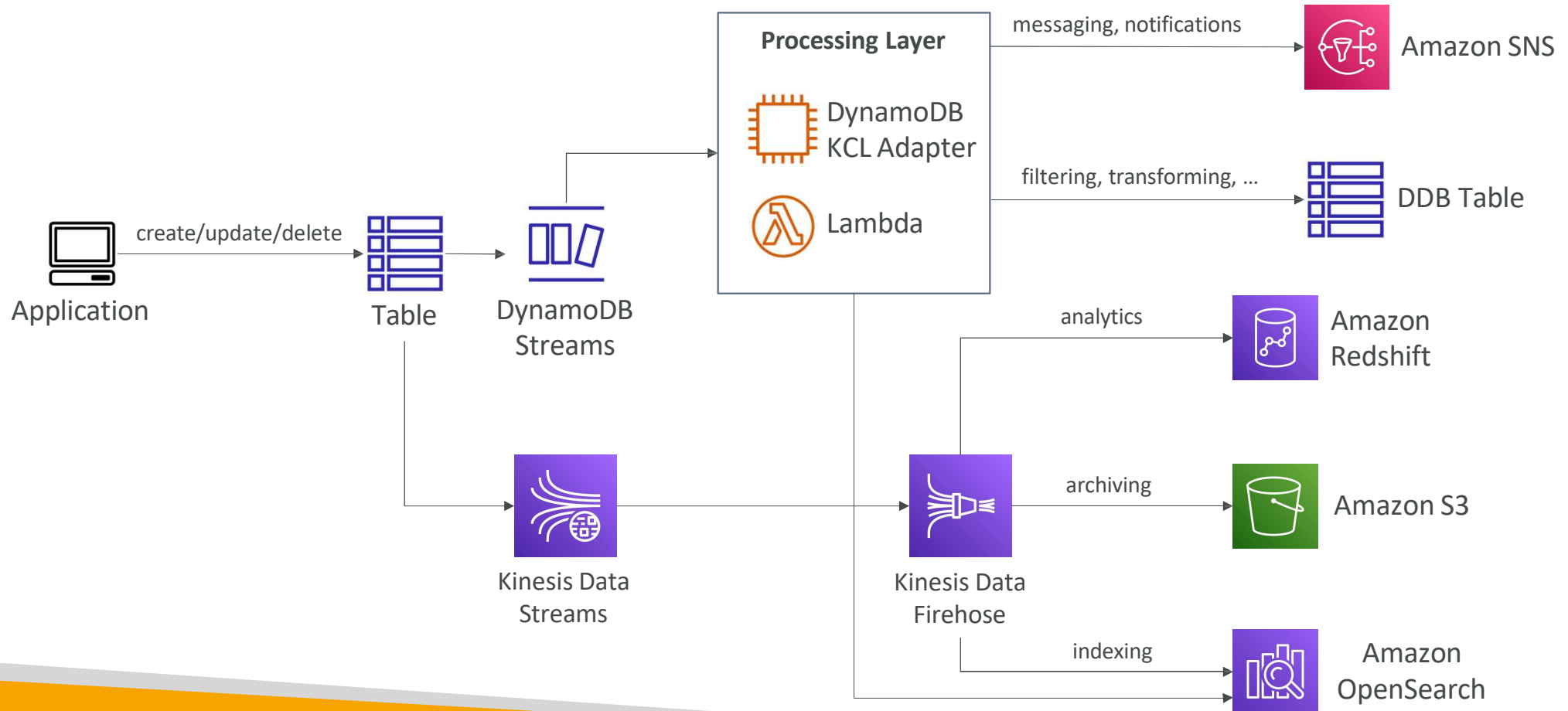
DynamoDB Streams

- 24 hours retention
- Limited # of consumers
- Process using AWS Lambda Triggers, or DynamoDB Stream Kinesis adapter

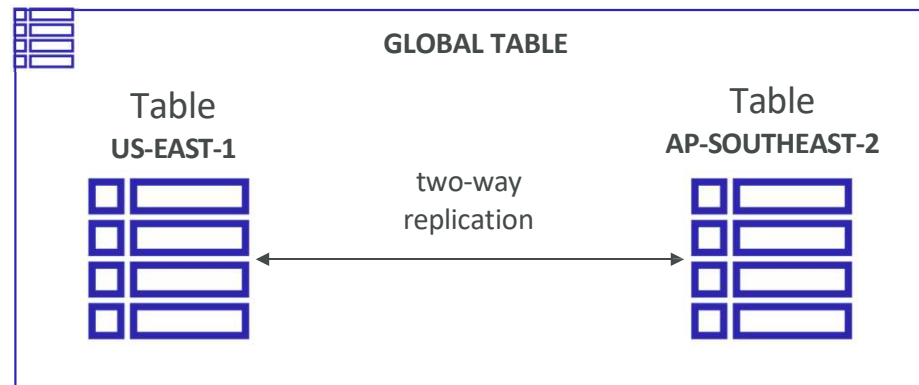
Kinesis Data Streams (newer)

- 1 year retention
- High # of consumers
- Process using AWS Lambda, Kinesis Data Analytics, Kinesis Data Firehose, AWS Glue Streaming ETL...

DynamoDB Streams



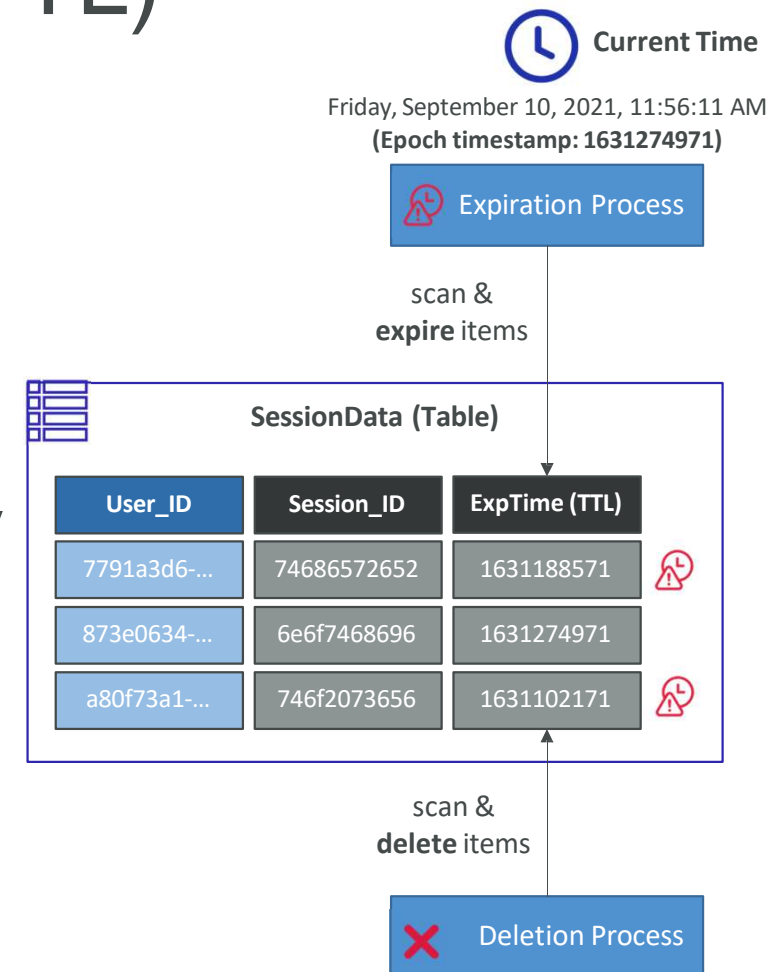
DynamoDB Global Tables




- Make a DynamoDB table accessible with low latency in multiple-regions
- Active-Active replication
- Applications can READ and WRITE to the table in any region
- Must enable DynamoDB Streams as a pre-requisite

DynamoDB - Time To Live (TTL)

- Automatically delete items after an expiry timestamp
- Use cases: reduce stored data by keeping only current items, adhere to regulatory obligations, web session handling...

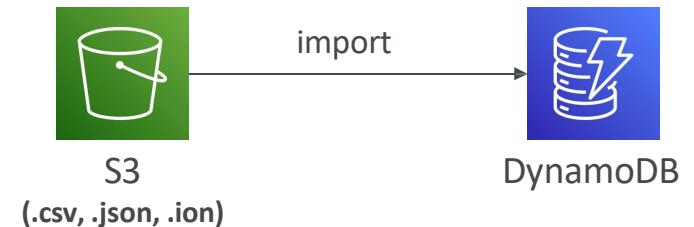
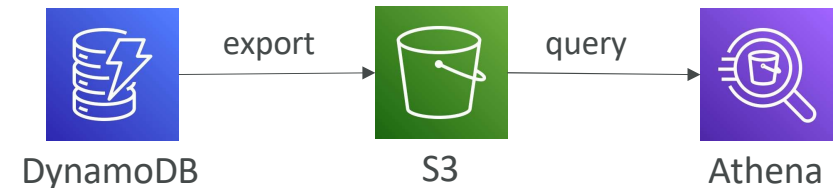


DynamoDB - Backups for disaster recovery

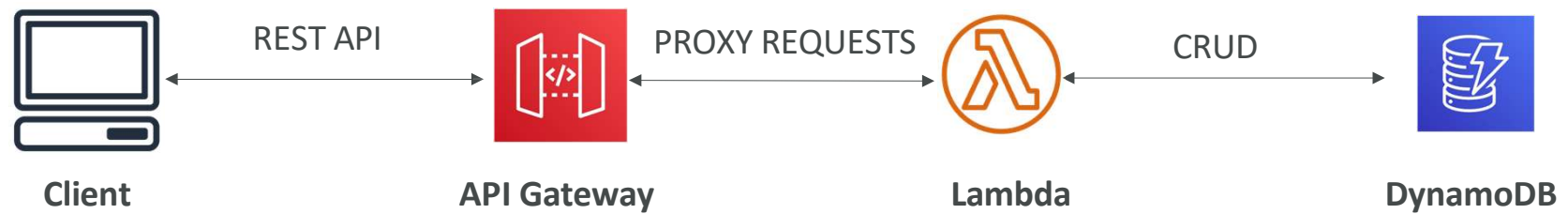
- Continuous backups using point-in-time recovery (PITR)
 - Optionally enabled for the last 35 days
 - Point-in-time recovery to any time within the backup window
 - The recovery process creates a new table
 - On-demand backups
 - Full backups for long-term retention, until explicitly deleted
 - Doesn't affect performance or latency
 - Can be configured and managed in AWS Backup (enables cross-region copy)
 - The recovery process creates a new table
- 

DynamoDB – Integration with Amazon S3

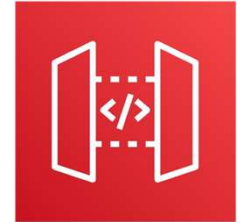
- Export to S3 (must enable PITR)
 - Works for any point of time in the last 35 days
 - Doesn't affect the read capacity of your table
 - Perform data analysis on top of DynamoDB
 - Retain snapshots for auditing
 - ETL on top of S3 data before importing back into DynamoDB
 - Export in DynamoDB JSON or ION format
- Import from S3
 - Import CSV, DynamoDB JSON or ION format
 - Doesn't consume any write capacity
 - Creates a new table
 - Import errors are logged in CloudWatch Logs



Example: Building a Serverless API




AWS API Gateway



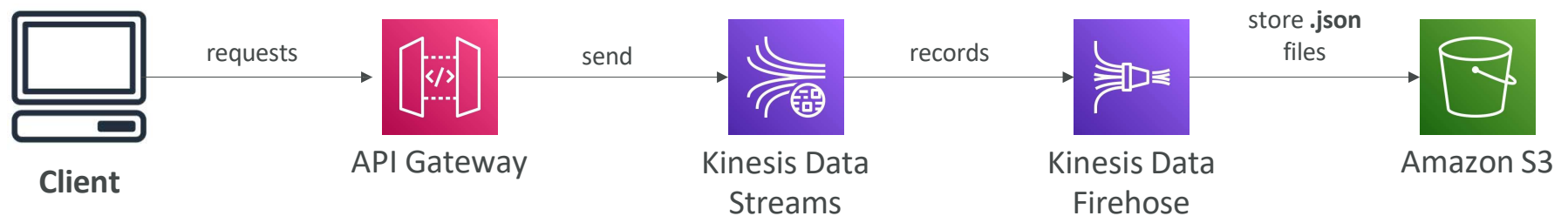
- AWS Lambda + API Gateway: No infrastructure to manage
- Support for the WebSocket Protocol
- Handle API versioning (v1, v2...)
- Handle different environments (dev, test, prod...)
- Handle security (Authentication and Authorization)
- Create API keys, handle request throttling
- Swagger / Open API import to quickly define APIs
- Transform and validate requests and responses
- Generate SDK and API specifications
- Cache API responses

API Gateway - Integrations High Level


- Lambda Function
 - Invoke Lambda function
 - Easy way to expose REST API backed by AWS Lambda
 - HTTP
 - Expose HTTP endpoints in the backend
 - Example: internal HTTP API on premise, Application Load Balancer...
 - Why? Add rate limiting, caching, user authentications, API keys, etc...
 - AWS Service
 - Expose any AWS API through the API Gateway
 - Example: start an AWS Step Function workflow, post a message to SQS
 - Why? Add authentication, deploy publicly, rate control...
- 

API Gateway - AWS Service Integration


Kinesis Data Streams example



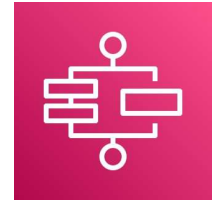
API Gateway - Endpoint Types

- Edge-Optimized (default): For global clients
 - Requests are routed through the CloudFront Edge locations (improves latency)
 - The API Gateway still lives in only one region
 - Regional:
 - For clients within the same region
 - Could manually combine with CloudFront (more control over the caching strategies and the distribution)
 - Private:
 - Can only be accessed from your VPC using an interface VPC endpoint (ENI)
 - Use a resource policy to define access
- 

API Gateway - Security

- User Authentication through
 - IAM Roles (useful for internal applications)
 - Cognito (identity for external users - example mobile users)
 - Custom Authorizer (your own logic)
 - Custom Domain Name HTTPS security through integration with AWS Certificate Manager (ACM)
 - If using Edge-Optimized endpoint, then the certificate must be in us-east-1
 - If using Regional endpoint, the certificate must be in the API Gateway region
 - Must setup CNAME or A-alias record in Route 53
- 

AWS Step Functions



- Build serverless visual workflow to orchestrate your Lambda functions
- Features: sequence, parallel, conditions, timeouts, error handling, ...
- Can integrate with EC2, ECS, On-premises servers, API Gateway, SQS queues, etc...
- Possibility of implementing human approval feature
- Use cases: order fulfillment, data processing, web applications, any workflow

■ In Progress ■ Succeeded ■ Failed ■ Cancelled ■ Caught Error

