Serverless Technology

- Serverless does not mean there are no servers…
  it means you just don't manage / provision / see them

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

1) AWS Lambda { Function-as-a-Service }

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!

} cost effective because doesn't need to be continuously running.

Benefits of AWS Lambda

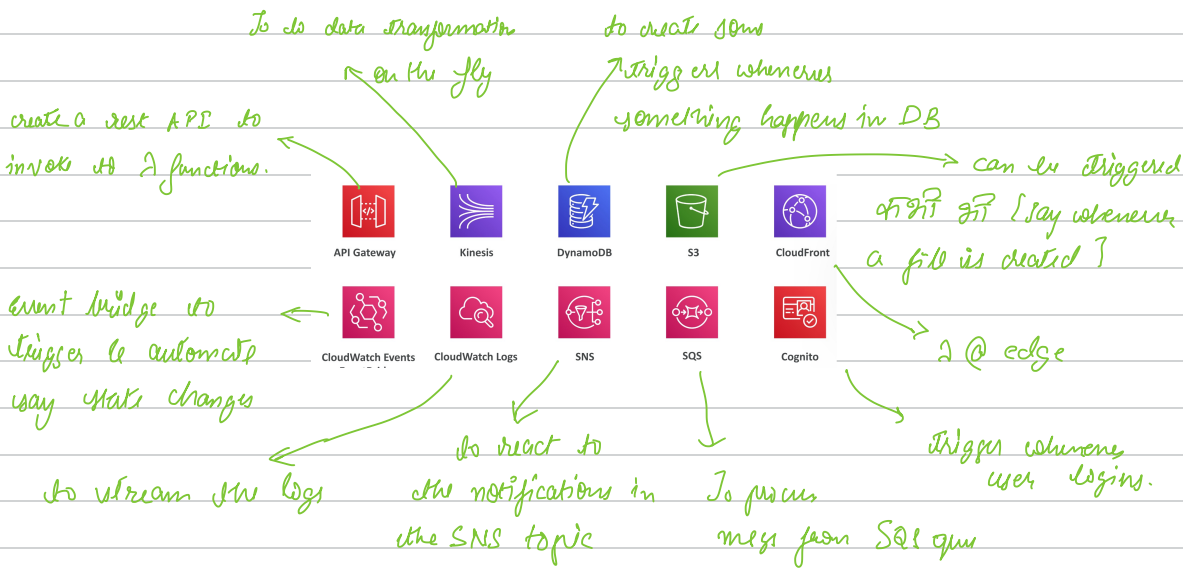
- Easy Pricing:
  - Pay per request and compute time
  - Free tier of 1,000,000 AWS Lambda requests and 400,000 GBs of compute time
- Integrated with the whole AWS suite of services
- Integrated with many programming languages
- Easy monitoring through AWS CloudWatch
- Easy to get more resources per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

(↑) RAM ⟹ (↑) CPU & (↑) network.

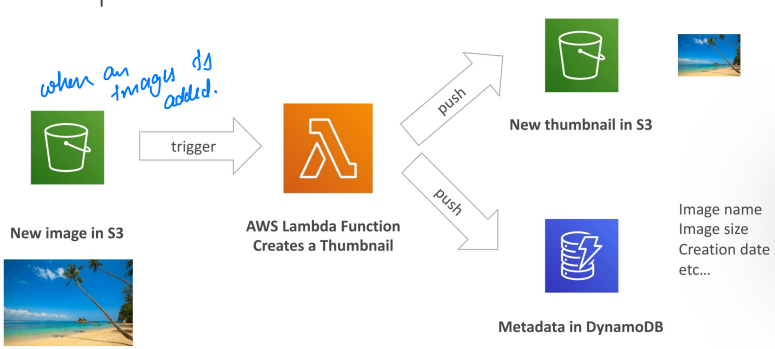
- Node.js (JavaScript)
- Python
- Java (Java 8 compatible)
- C# (.NET Core)
- Golang
- C# / Powershell
- Ruby
- Custom Runtime API (community supported, example Rust)

- Lambda Container Image
  - The container image must implement the Lambda Runtime API
  - ECS / Fargate is preferred for running arbitrary Docker images

* AWS lambda integrations



Example: Serverless Thumbnail creation



Example: Serverless CRON Job

CRON → A work schedule hypervisor that runs assignments at prescribed times. [Command RUN on Notice]

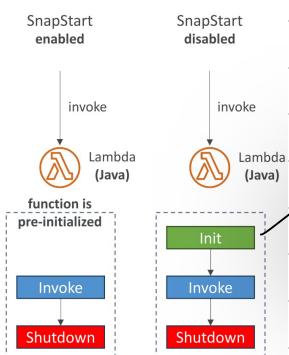
Lambda Limits !

- Execution:
  - Memory allocation: 128 MB – 10GB (1 MB increments) when we increase memory we have more vCPUs
  - Maximum execution time: 900 seconds (15 minutes)
  - Environment variables (4 KB)
  - Disk capacity in the "function container" (in /tmp): 512 MB to 10GB → temp space
  - 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 for big files
  - Size of environment variables: 4 KB

Lambda SnapStart

A Rapid start for >= Java 11

- 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: init function now ahead of time & saves the state
  - ✓ Lambda initializes your function
  - ✓ Takes a snapshot of memory and disk state of the initialized function
  - ✓ Snapshot is cached for low-latency access



may take a very long time in java.

Lambda Invocation Lifecycle Phases

Lambda Cold Start → The initial delay to experienced when a fun" is invoked for the first time or after being idle as the fun" runtime environment is being invoked.

→ to mitigate this we may use warming mechanisms by regularly invoking them

Lambda Layers → a .zip file archive that contains supplementary code or data.

Layers usually contain → lib dependencies
→ custom runtime
→ config files

→ used to reduce the size of the deployment package.
→ Layers can be used to share common code or data across different fun^n or AWS accounts.

Lambda workers shutdown → When λ is about to shut down it sends a shutdown event to all the registered extensions & this time can be used by extensions for clean up tasks.

This shutdown event is a response to the next API request.

The entire shutdown phase is capped to 2 mins

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 coming our of CDN
- Pay only for what you use
- Fully serverless

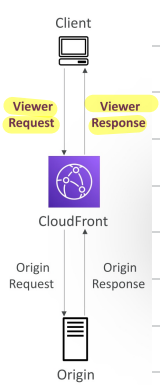
Use Cases :-

- Website Security and Privacy
- Dynamic Web Application at the Edge
- Search Engine Optimization (SEO) → for businesses
- 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

(a) Cloudfront Functions

CloudFront Functions

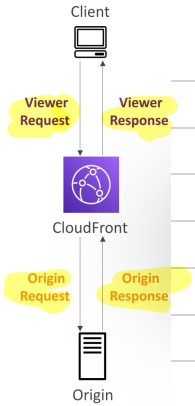
- Lightweight functions written in JavaScript
- For high-scale, latency-sensitive CDN customizations
- Sub-ms startup times, millions of requests/second
- Used to change Viewer requests and responses:
  - **Viewer Request:** after CloudFront receives a request from a viewer
  - **Viewer Response:** before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



→ used to change viewers requests & responses only!

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



→ used to change all { user req. user resp. origin req. origin resp.

same region as the cloudfront distribution.

	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 {need quick functions}	5 – 10 seconds {can have logically hard functions}
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, 1/6th price of @Edge	No free tier, charged per request & duration

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

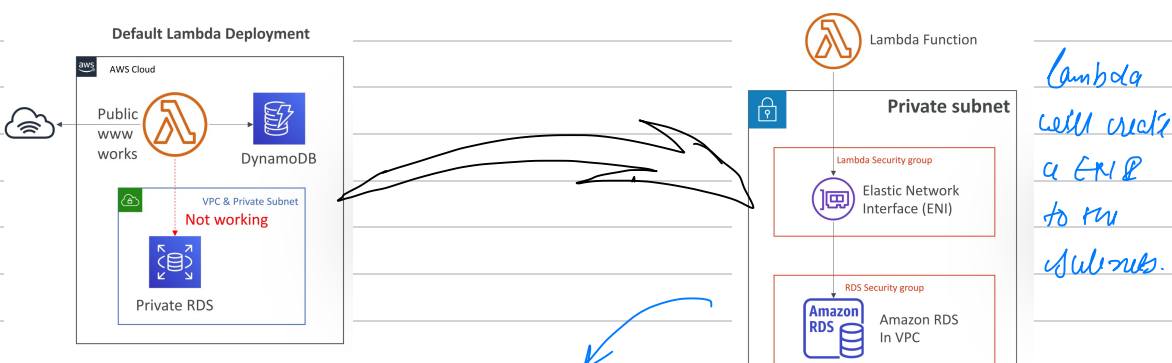
Networking fundamentals for lambda

→ by dyault a lambda function is launched outside your own VPC.

→ No resources in our VPC (EC2 instance, RDS DB, Elastic cache) can't be accessed by lambda.

→ Can be used to access any public APIs on AWS

→ We need to launch the lambda fun" in our VPC.

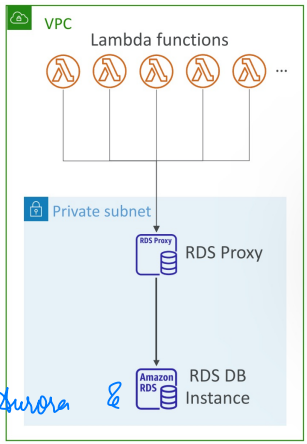


lambda will create a ENI to the subnets.

For lambda fun" in our VPC, we must define VPC ID, the subnets we want to launch in & the security groups.

Lambda with RDS Proxy

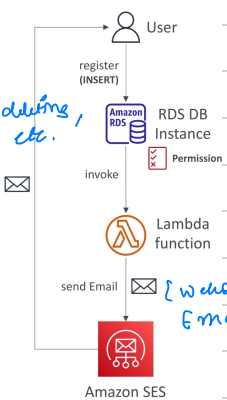
- If Lambda functions directly access your database, they may open too many connections under high load may lead to timeout issues.
- RDS Proxy
  - Improve scalability by pooling and sharing DB connections
  - Improve availability by reducing by 66% the failover time and preserving connections
  - Improve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



imporves:
(1) scalability by pooling & sharing DB also in time of failover, this will improve availability

Invoking Lambda from RDS & Aurora

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

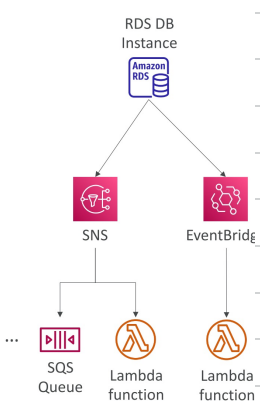


→ need to use connection from within DB cannot be done from management console.

{info about the data in DB}

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



{info about the DB}

② Dynamo DB → standard
→ Intelligent accn.

→ NoSQL (key - value storage)
→ Fully managed, serverless, with replication across multi - AZs
→ scale workloads due to internally distributed.
→ Fast
→ Integrated with IAM
→ Low cost

DynamoDB - Basics

- DynamoDB is made of Tables → or partition key.
- 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 – Read/Write Capacity Modes

{ for an application which goes from 1000 to 10M transactions then on demand is good. }

- Control how you manage your table's capacity (read/write throughput)

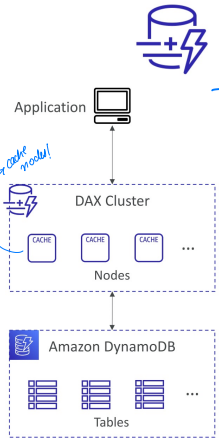
- Provisioned Mode (default)
  - You specify the number of reads/writes per second
  - You need to plan capacity beforehand
  - Pay for provisioned Read Capacity Units (RCU) & Write Capacity Units (WCU)
  - Possibility to add auto-scaling mode for RCU & WCU → use capacity calculator for understanding

- 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

{ No concept of RCU or WCU & pay for exactly what you use. }

DynamoDB Accelerator (DAX)

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

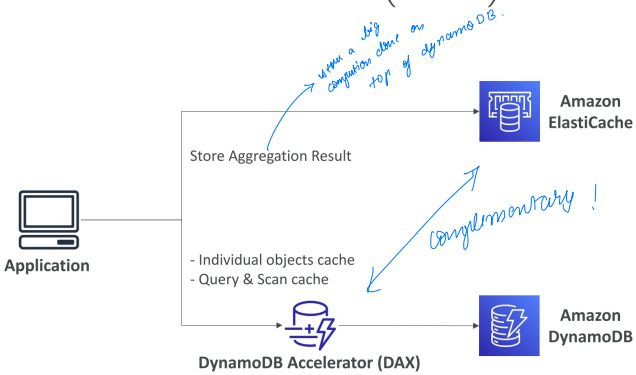


→ DAX cluster to scale the read workloads!

→ NO CODE Changes Required *

→ offloading the heavy reads on hot keys off ur DynamoDB table & prevent "Provisioned Throughput Exceeded Exception"

DynamoDB Accelerator (DAX) vs. ElastiCache



371

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

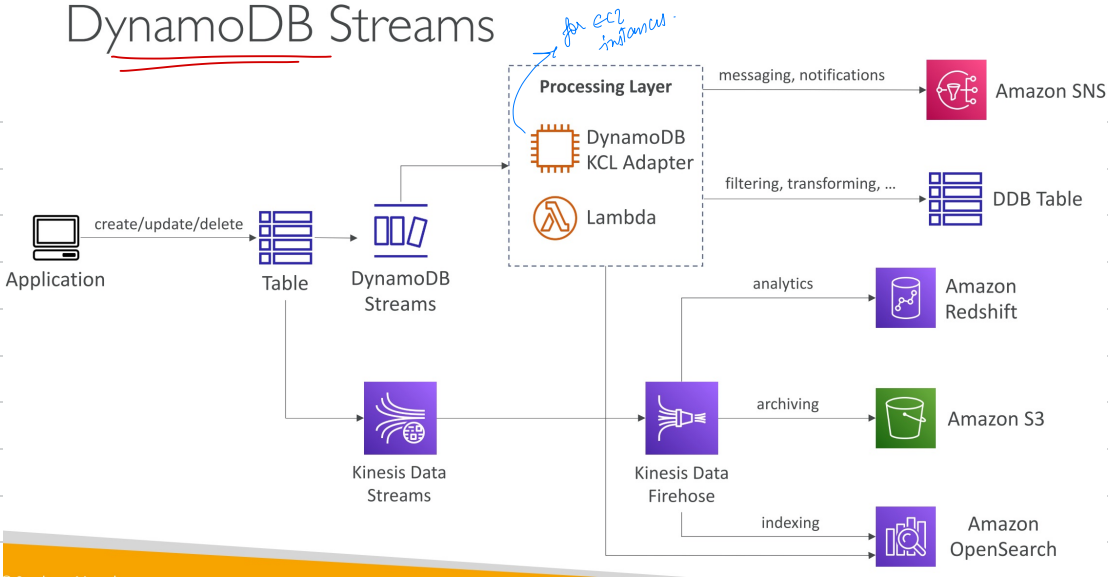
DynamoDB Streams

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

Kinesis Data Streams (newer)

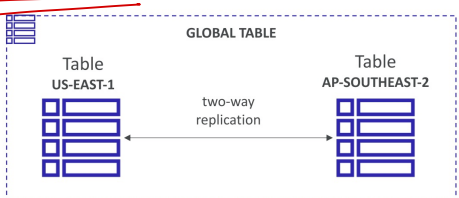
- 1 year retention
- High # of consumers
- Process using AWS Lambda, Kinesis Data Analytics, Kineis Data Firehose, AWS Glue Streaming ETL…

DynamoDB Streams



DynamoDB Global Tables

{ enable DynamoDB streams }



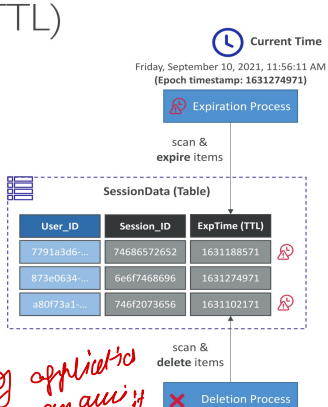
- Make a DynamoDB table accessible with low latency in multiple-regions
- Active-Active replication
- Applications can READ and WRITE to the table in any region
- Must enable DynamoDB Streams as a pre-requisite underlying technique to replicate.

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…

deleting data after 2Y?

(Eg: for 2 hrs session any kind of application can avail it)



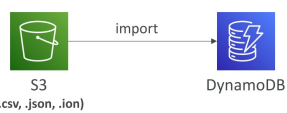
DynamoDB – Backups for disaster recovery

- 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 explicitely 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

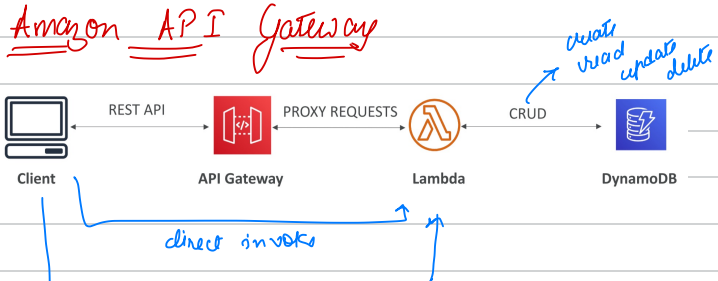
DynamoDB – Integration with Amazon S3

- Export to S3 (must enable PITR) → point in time recovery!
  - Works for any point of time in the last 35 days {continuous backup}
  - 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



(3) Amazon API Gateway



→ A serverless offering from AWS to not just expose it as a Http endpoint but also do authentication & shit.

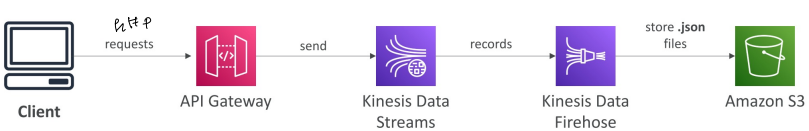
AWS API Gateway

- AWS Lambda + API Gateway: No infrastructure to manage
- Support for the WebSocket Protocol { real time streaming }
- Handle API versioning (v1, v2…)
- Handle different environments (dev, test, prod…)
- Handle security (Authentication and Authorization)
- Create API keys, handle request throttling { to many requests }
- Swagger / Open API import to quickly define APIs
- Transform and validate requests and responses { to ensure that invocations are correct }
- 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



** people send data into KDS but w/o giving them to AWS credentials!

→ We can expose any AWS service to the outside through an API gateway.

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

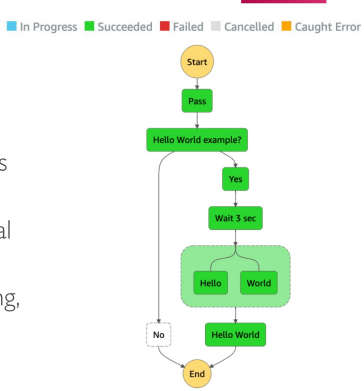
- User Authentication through
  - IAM Roles (useful for internal applications)
  - Cognito (identity for external users – example mobile users)
  - Custom Authorizer (your own logic) { a jwt ? }

- 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 [to point to the domain in API gateway]

default timeout for API gateway is 29 sec

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



Amazon Cognito

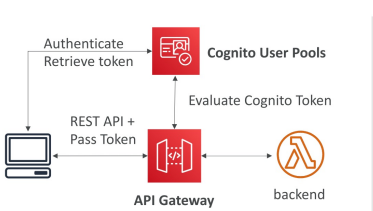
- Give users an identity to interact with our web or mobile application
- Cognito User Pools:
  - Sign in functionality for app users
  - Integrate with API Gateway & Application Load Balancer
- Cognito Identity Pools (Federated Identity):
  - Provide AWS credentials to users so they can access AWS resources directly
  - Integrate with Cognito User Pools as an identity provider
- Cognito vs IAM: "hundreds of users", "mobile users", "authenticate with SAML"

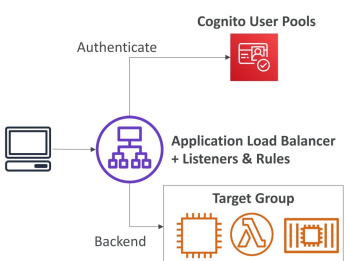
Cognito User Pools (CUP) – User Features

- Create a serverless database of user for your web & mobile apps
- Simple login: Username (or email) / password combination
- Password reset
- Email & Phone Number Verification
- Multi-factor authentication (MFA)
- Federated Identities: users from Facebook, Google, SAML...

Cognito User Pools (CUP) - Integrations

- CUP integrates with API Gateway and Application Load Balancer

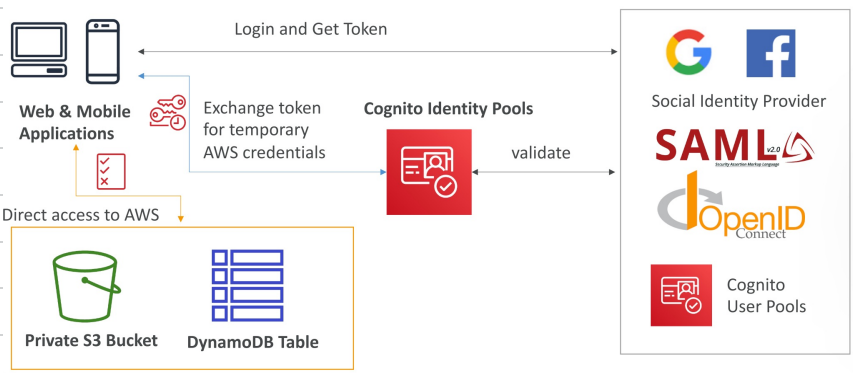




2 Cognito Identity Pools (Federated Identities)

- Get identities for "users" so they obtain temporary AWS credentials
- Users source can be Cognito User Pools, 3rd party logins, etc…
- Users can then access AWS services directly or through API Gateway
- The IAM policies applied to the credentials are defined in Cognito
- They can be customized based on the user_id for fine grained control
- Default IAM roles for authenticated and guest users

Cognito Identity Pools – Diagram




```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem", "dynamodb:BatchGetItem", "dynamodb:Query",
                "dynamodb:PutItem", "dynamodb:UpdateItem", "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/MyTable"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:LeadingKeys": [
                        "${cognito-identity.amazonaws.com:sub}"
                    ]
                }
            }
        }
    ]
}
```