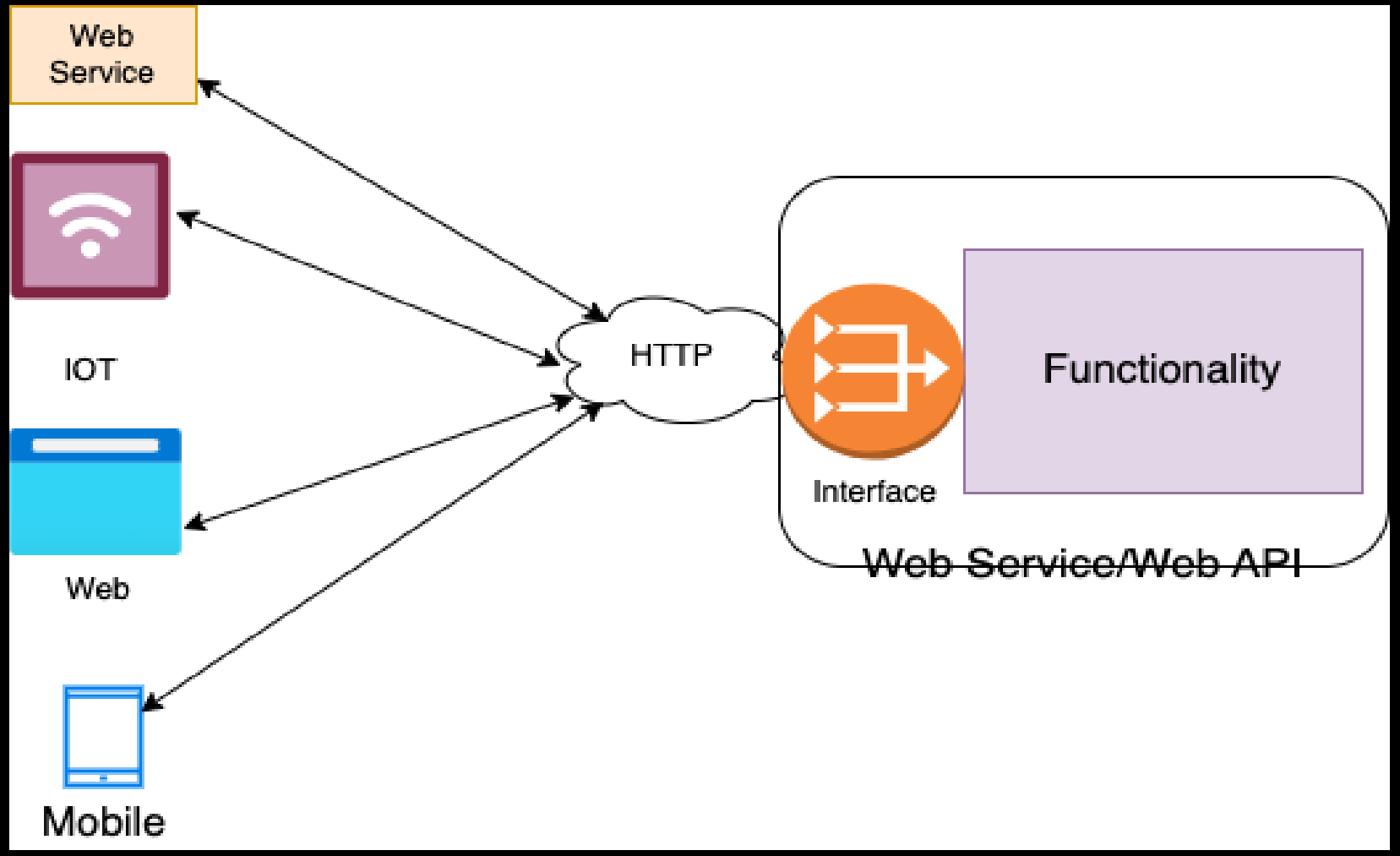
Serverless Web APIs

The AWS-related services

Serverless?

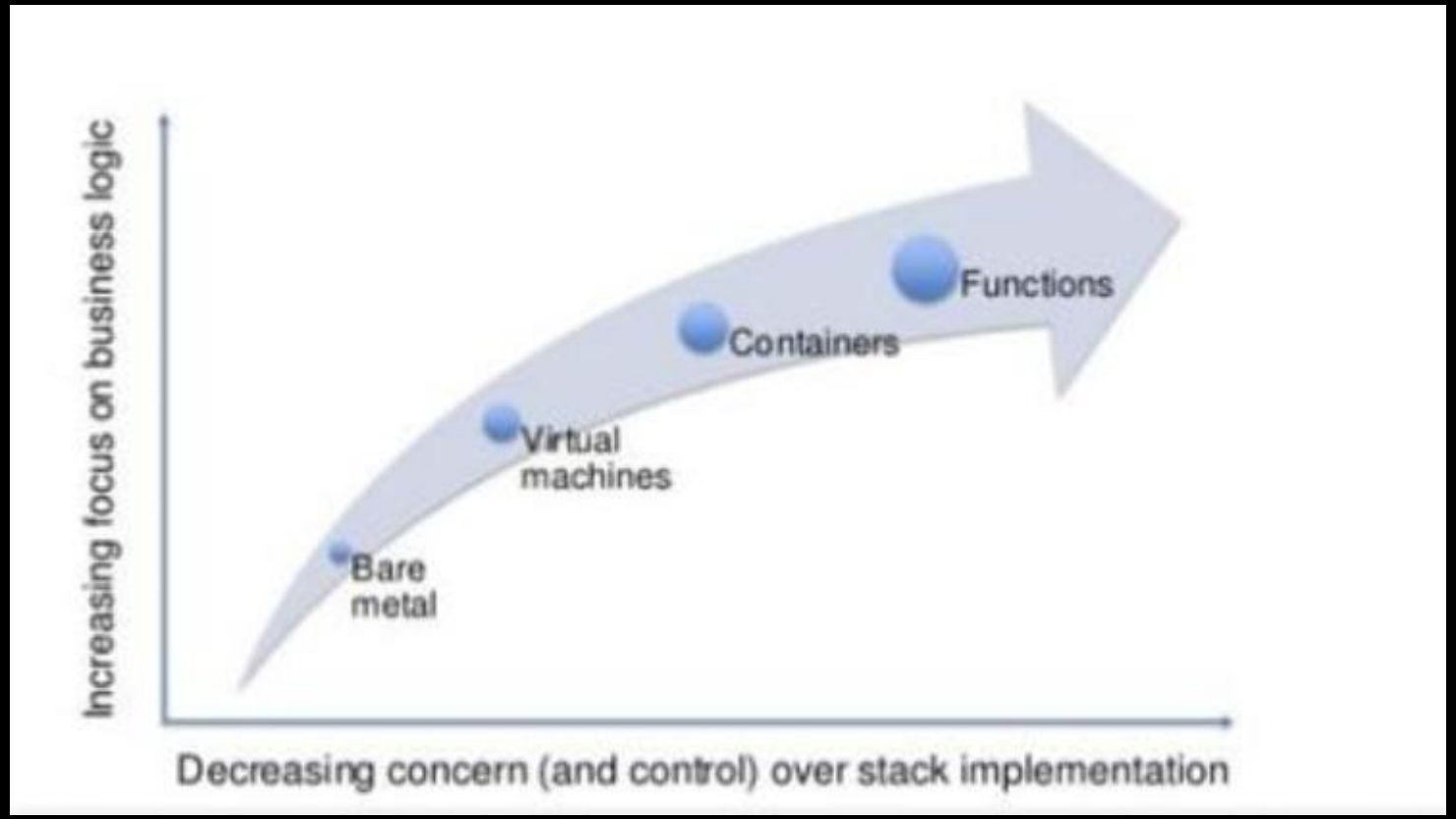
- Serverless relieves the developer of responsibilities:
    1. No servers to provision or manage.
    2. Auto-Scales with usage.
    3. No Up-front cost.
    4. Availability and fault tolerance built-in.

Web APIs?


Web Service
IOT
Web
Mobile
HTTP
Interface
Functionality
Web Service/Web API

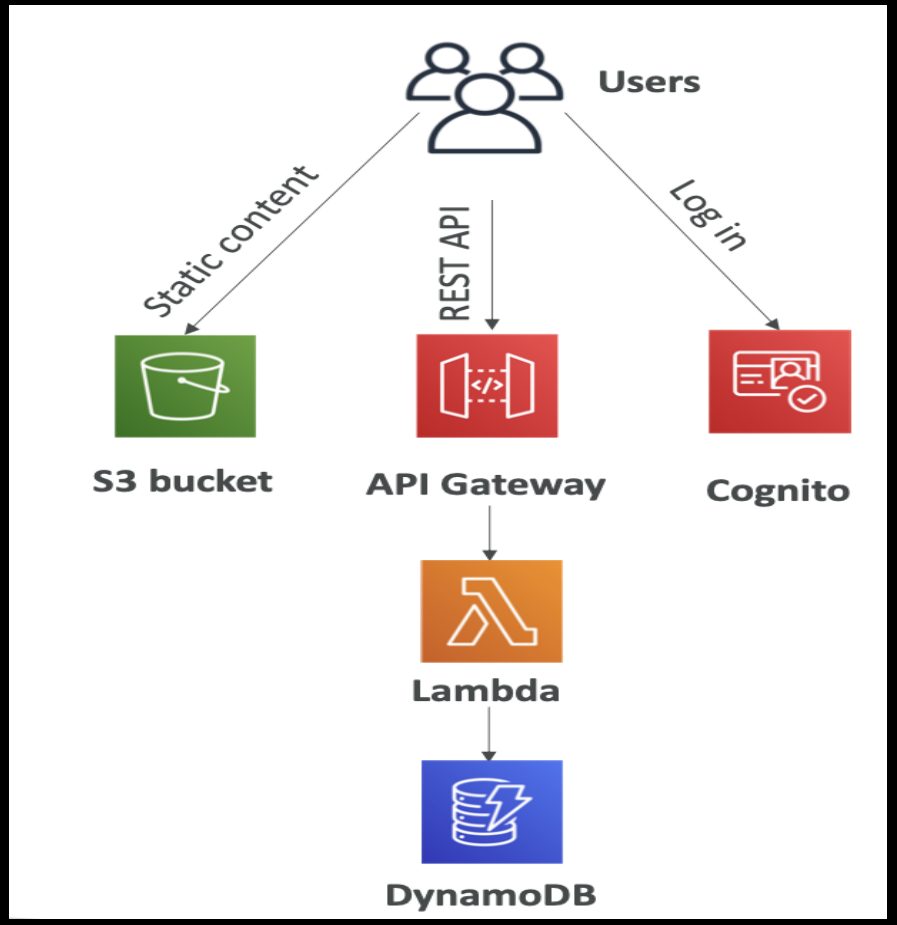
Developer’s focus.

- Developers should focus on the product, not infrastructure.



Serverless Services on AWS

- AWS Lambda. (Compute)
- DynamoDB. (NoSQL)
- AWS Cognito. (User Accounts Mgt.)
- API Gateway (HTTP/REST endpoints)
- S3 (Storage)
- SNS & SQS. (Messaging)
- AWS Kinesis Data Firehose
- Aurora Serverless (RDB)
- Step Functions (Orchestration)
- Fargate (Containers)
- And more …



AWS Lambda Service

Serverless Compute

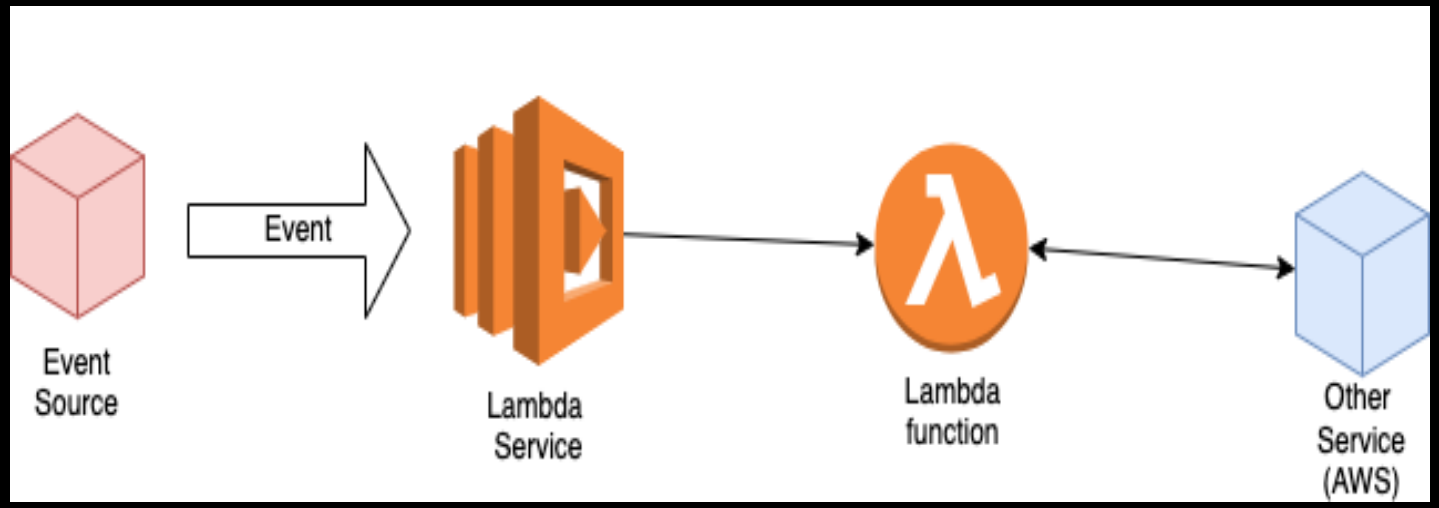
AWS Lambda

- “Lambda is an event-driven, serverless computing platform provided by AWS. It is a computing service that runs code (a function) in response to events and automatically manages the computing resources (CPU, memory, networking) required by that code. It was introduced on November 13, 2014.” Wikipedia

- “Custom code that runs in an ephemeral container” Mike Robins

- FaaS (Functions as a Service)
  - IaaS, PaaS, SaaS

Lambda runtime model.



- Event Source (Trigger), e.g. HTTP request; Change in data store, e.g. database, S3; Change in resource state, e.g. EC2 instance.
- Lambda function: Python, Node, Java, Go, C#, etc.

AWS Lambda service

- The Lambda service manages:
  1. Auto scaling (horizontal)
  2. Load balancing.
  3. OS maintenance.
  4. Security isolation.
  5. Utilization (Memory, CPU, Networking)

- Characteristics:
  1. Function as a unit of scale.
  2. Function as a unit of deployment.
  3. Stateless nature.
  4. Limited by time - short executions.
  5. Run on-demand.
  6. Pay per execution and compute time – generous free tier.
  7. Do not pay for idle time.

Anatomy of a Lambda function

```
... imports]..

.... initialization ......
.... e.g. d/b connection ....

export const handler = async (event, context) => {
  ......
};

const localFn = (arg) => {
   ......
}
```

- Handler() – function to be executed upon invocation.
- Event object – the payload and metadata provided by the event source.
- Context object – access to runtime information.
- Initialization code executes before the handler; Cold starts only.

Lambda function configuration

- Lambda service provides a memory control to configure a function's compute power requirements.
  - The % of CPU core and network capacity are computed in proportion to its RAM.
- RAM:
  - From 128MB to 3,008 MB in 64 MB increments
  - The more RAM you add, the more vCPU credits you get.
    - 1,792 MB RAM allocation → one full vCPU reserved.
    - After 1,792 MB → more than one CPU assigned → should use multi-threading to fully utilize.

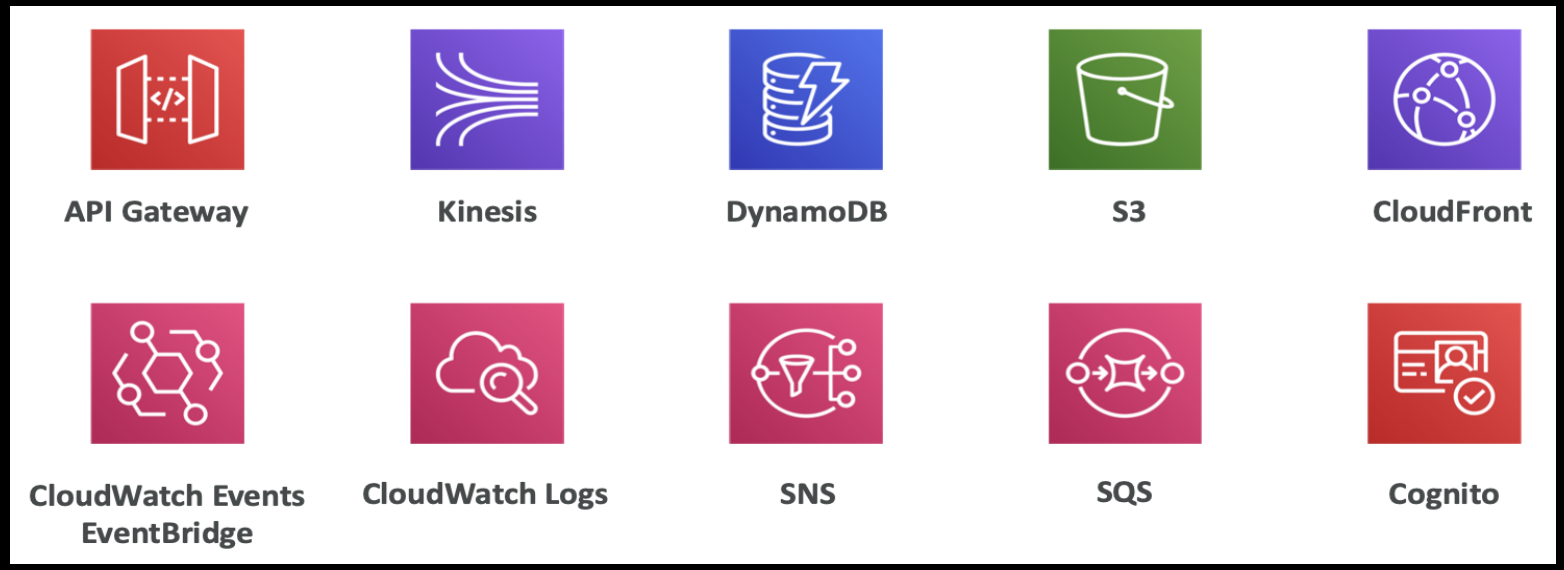
- For CPU-bound processing, increase the RAM allocation.
- Timeout setting: Max. runtime allowed.
  - Default 3 seconds; maximum 900 seconds (15 minutes).

Demo

- Objective:
  - Use the CDK to provision a ‘Hello World’ (Typescript/Node) lambda function.
  - Invok it (trigger it) using the AWS CLI.
  - See console.log() statement output in the CloudWatch Logs.

Lambda integration

- What services can trigger a lambda function?

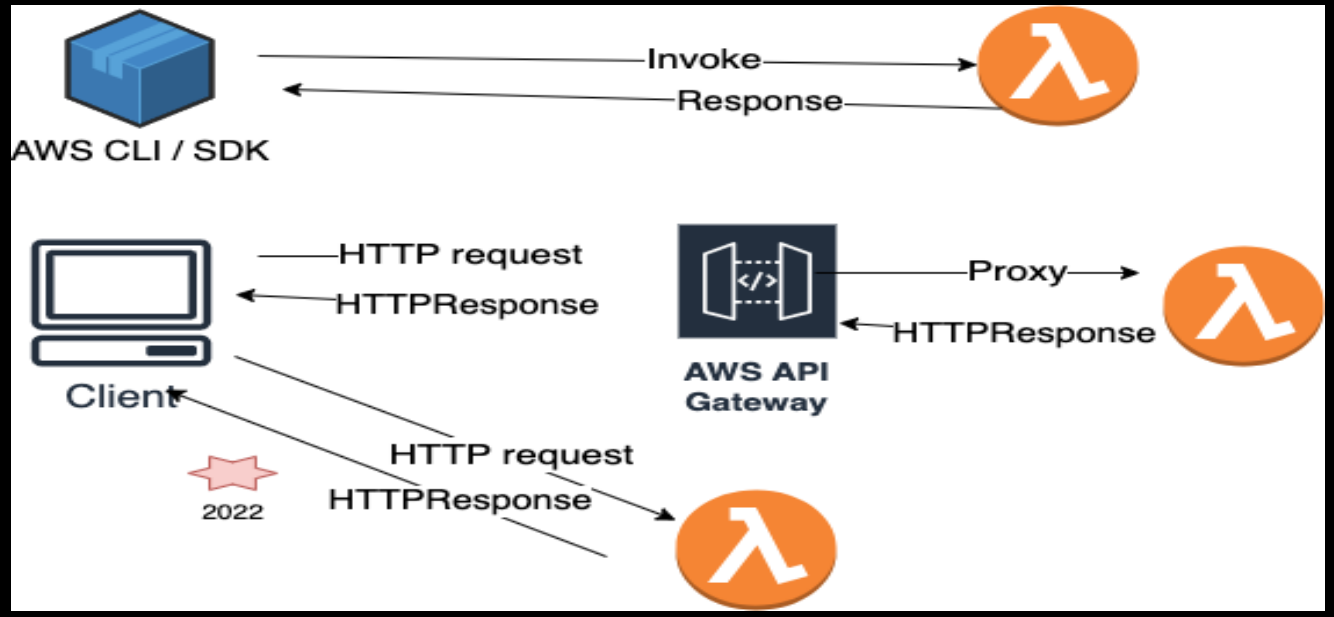


- Integration models:

  1. Synchronous. 2. Asynchronous. 3. Poll-based

Synchronous Integration

- Trigger (Event Source) options: CLI, SDK, API Gateway, Load Balancers, Function URLs.
- Client (Event source) waits for the response, i.e. synchronuous.
- Client should handle error responses (retries, exponential backoff, etc.)

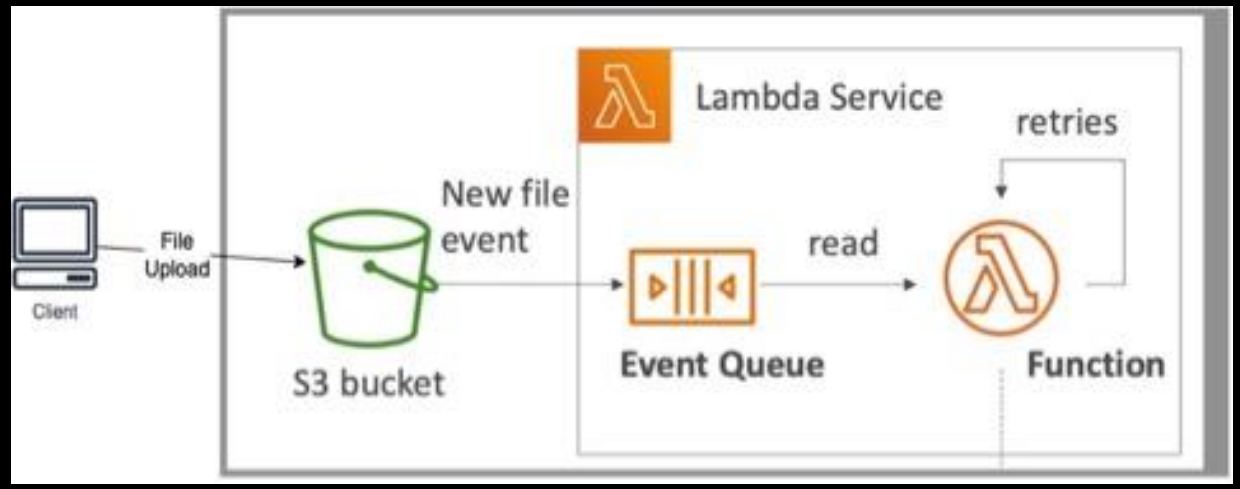


Demo

- Objective:
  - Use the CDK to:
    1. Provision a lambda function.
    2. Get the Lambda service to generate a URL endpoint.
  - Test the URL with Postman client.
  - Configure the endpoint as private and invoke it.

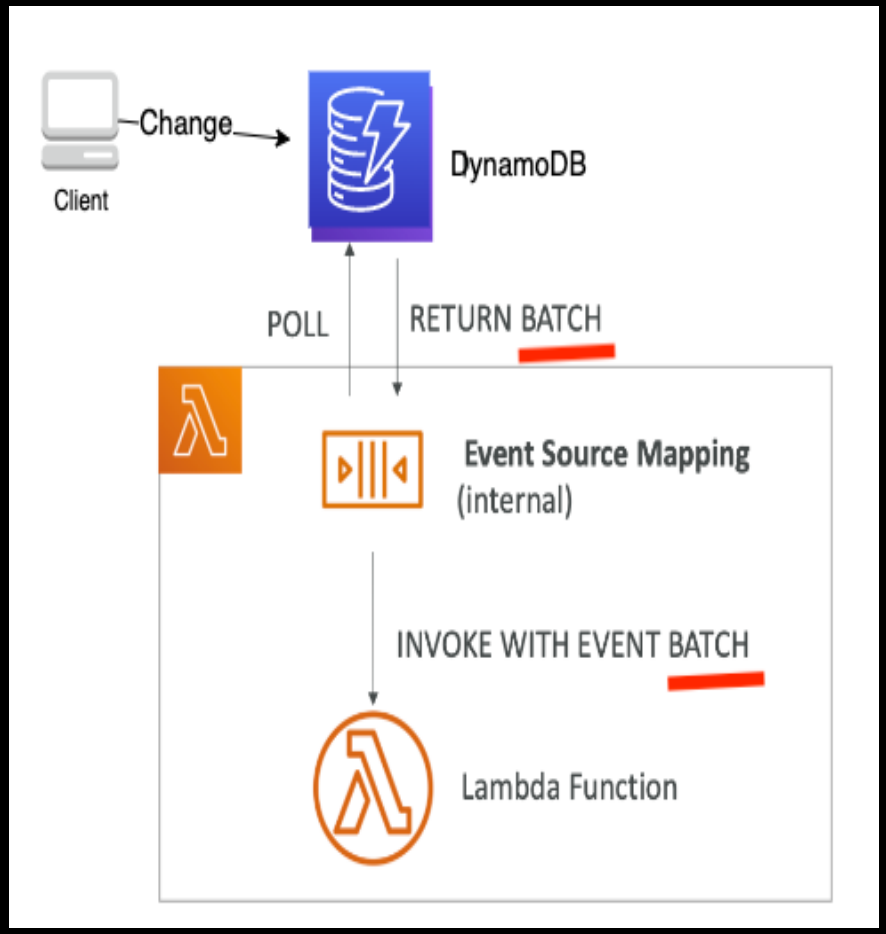
Asynchronous Integration.

- Trigger (Event Source): S3, SNS, CloudWatch.
- Lambda service places the events in a queue (see below image).
- Lambda service retries on errors – 3 retries, using exponential backoff algorithm.
- Function's processing should be idempotent (due to retries)
- Suitable when application does not require the function result immediately.



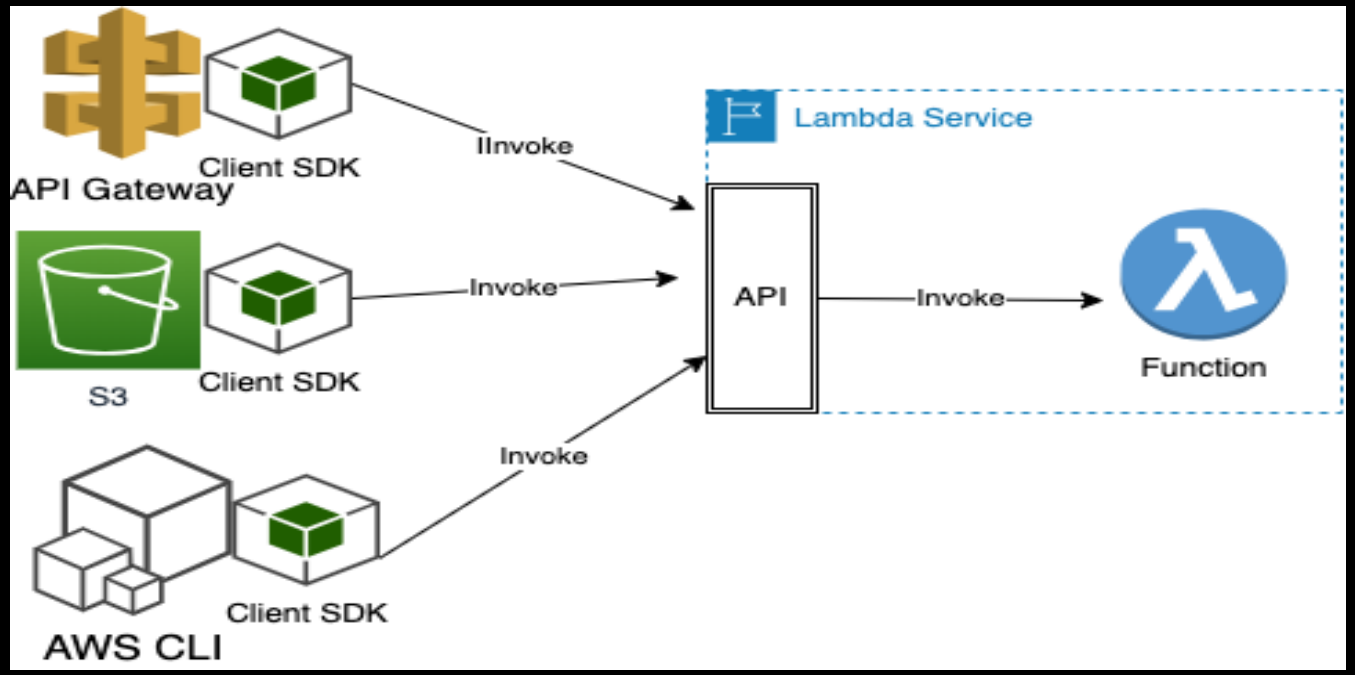
Event source mapping. (Poll-based Integration)

- Event Sources/Trigger: DynamoDB streams, SQS, Kinesis streams.
- Lambda service polls the source for event records.
- Lambda service invokes the function synchronously.



Lambda service API & SDK

- Lambda Service provides an API.
- Used by all other services that trigger Lambda functions across all models (sync, async, poll-based).
- Can pass any event payload structure you want.



Execution Role (IAM Role)

- Grants the lambda function permissions to access specified AWS services / resources.
- Many predefined / Managed policies, e.g.
  - AWSLambdaBasicExecutionRole – Upload logs to CloudWatch.
  - AWSLambdaDynamoDBExecutionRole – Read from DynamoDB Streams.
  - AWSLambdaSQSQueueExecutionRole – Read from SQS queue
  - AWSLambdaVPCAccessExecutionRole – Deploy function in VPC.

- Best practice: Create one Execution Role per function.

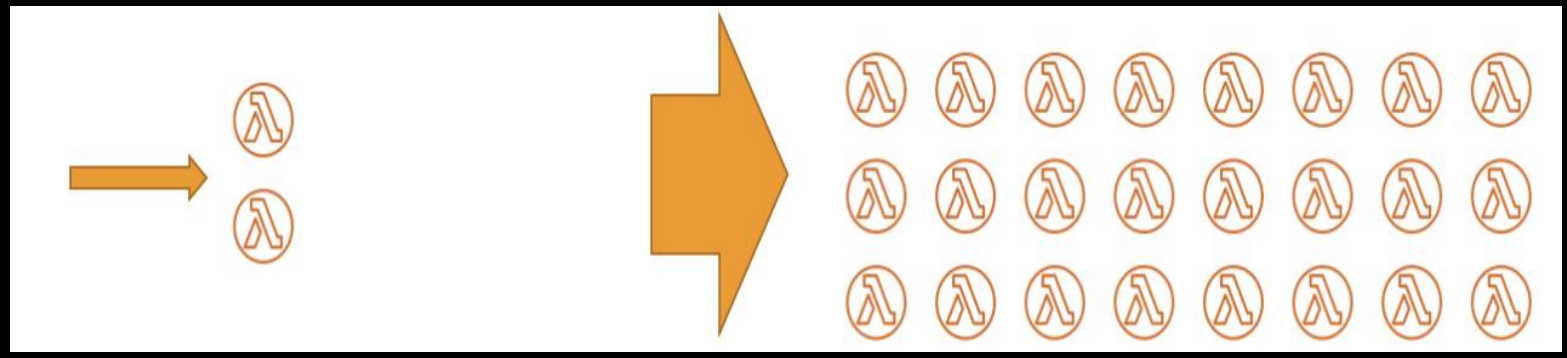
Resource based Policies.

- Use resource-based policies to give other AWS services (and accounts) permission to use your Lambda resource/function.

- An IAM principal (e.g. user, service) can access a Lambda resource:
  - if the IAM policy attached to the principal authorizes it,
  - OR if the function’s resource-based policy authorizes it.

- Ex.: An AWS S3 service can trigger a Lambda function if the function’s resource-based policy permits it.

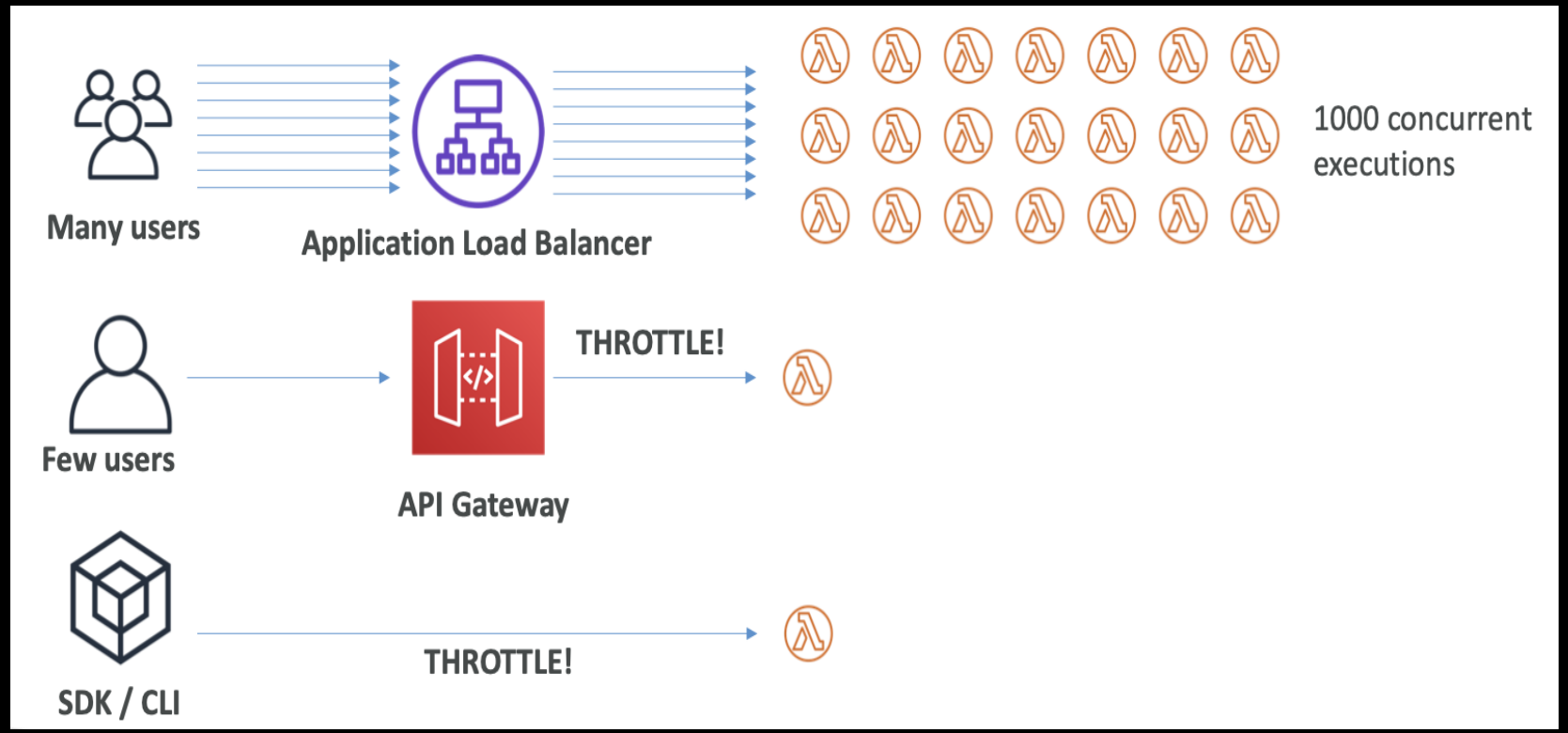
Concurrency and Throttling



- AWS account concurrency limit is set to 1000 concurrent executions
- Can set a reserved concurrency at the function level (= limit).
  - "Throttle" error response when limit exceeded.
- Throttle behavior:
  - Synchronous invocation => return ThrottleError – HTTP status 429
  - Asynchronous invocation => retry automatically and then go to a DLQ (Dead Letter Queue).

Concurrency issues

- The following can happen withoutyou reserve (=limit) concurrency:



Cold Starts & Provisioned Concurrency.

- Cold Start:
  - Source Event → New micro VM instance created → function's initialization code executes → handler code executes
- Warm start – Micro VM reused; Init code not executed.
- Lengthy init code (LoC, dependencies, SDK) effects overall event processing time.
  - Greater latency with cold start executions.
- Solution - Provisioned Concurrency:
  - Micro VMs are pre-allocated in advance; Init code executed during pre-allocation.
  - Cold starts avoided (minimized)
  - lower latency on average.