Serverless

Serverless.

- **Debunking the myths:**

  Serverless does not mean there are no servers

- **You, the developer:**
  1. **Don’t need to care about servers when coding.**
  2. **Don’t have to manage physical capacity.**

Without Serverless.

- **What is the right size for my server?**
- **What capacity is left on my server?**
- **How many servers shoul I provision?**
- **How do I handle hardware failures?**

- **What OS should my server run?**
- **Whom should have access to my servers?**
- **How do I detect a compromised server?**
- **How do I keep my server patched?**

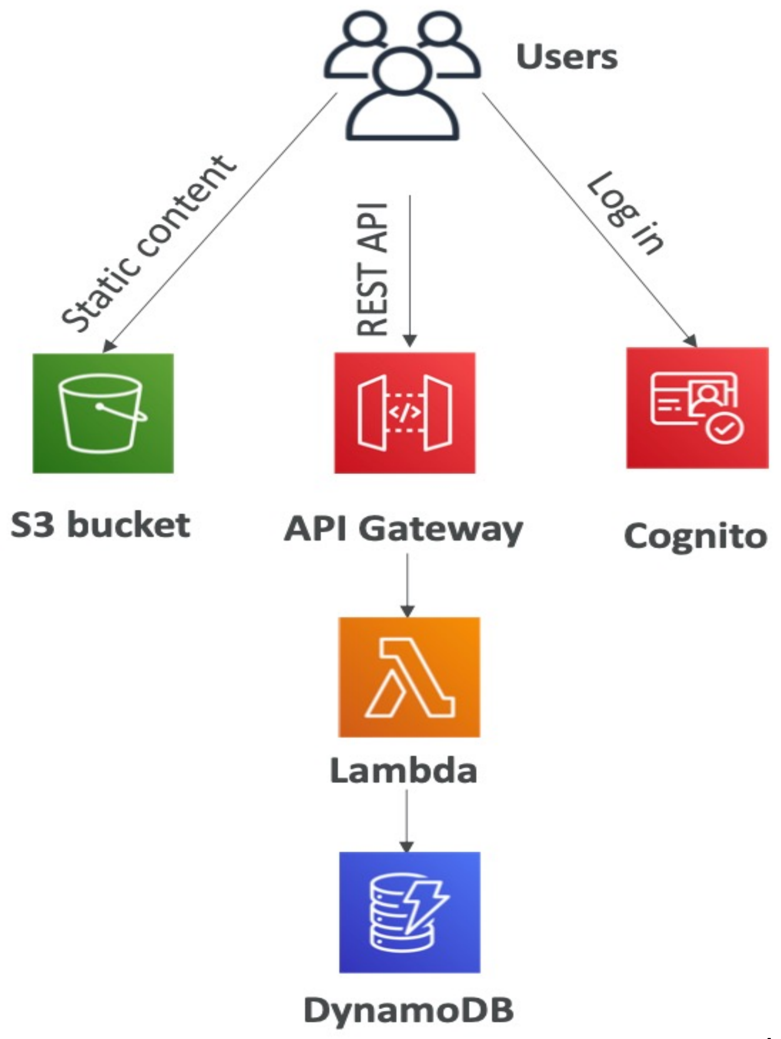
Serverless means ….

1. **No servers to provision or manage.**
2. **Scales with usage.**
3. **Pay for value.**
4. **Availability and fault tolerance built-in.**

- **Benefits:**
  1. **Greater agility.**
  2. **Less overhead.**
  3. **Increased scale.**
  4. **Better focus.**
  5. **Faster time to market.**

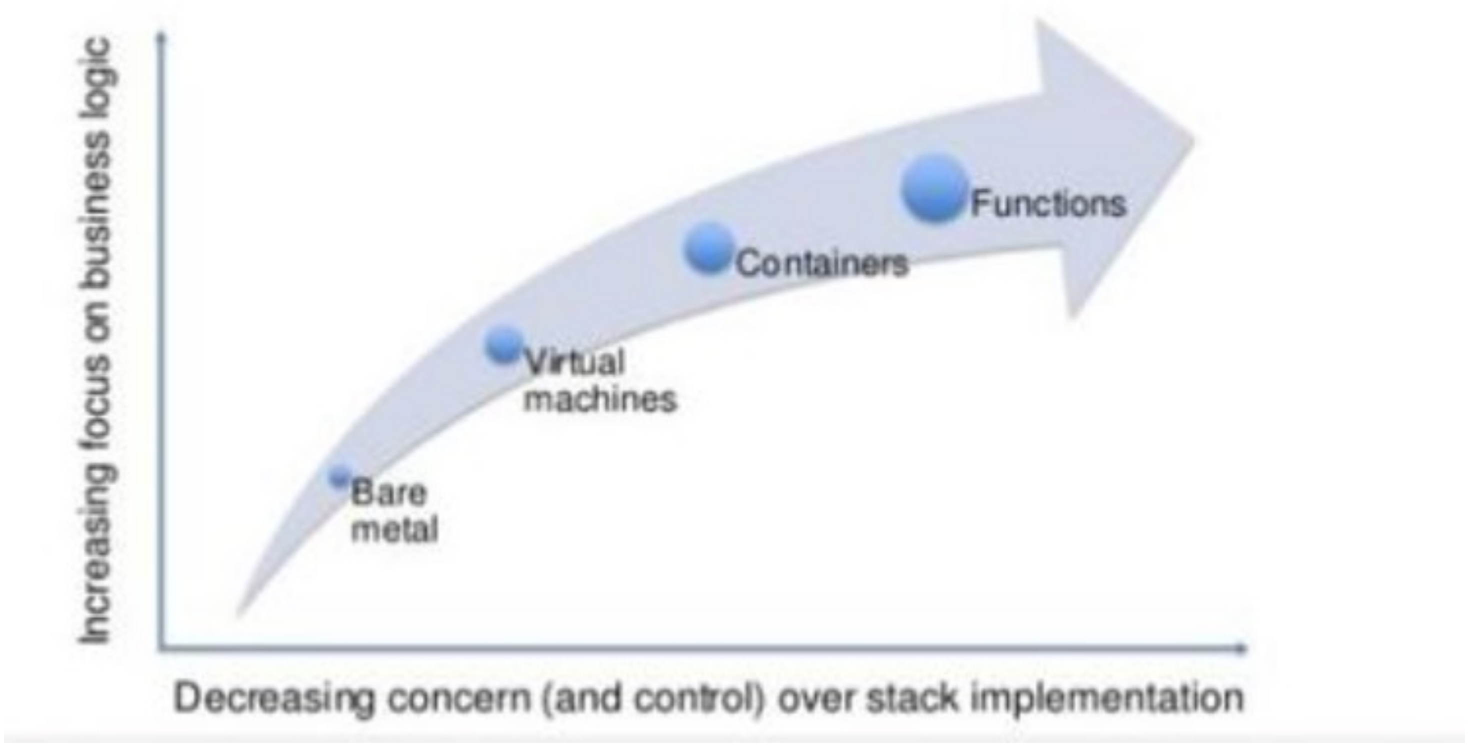
Serverless Services on AWS

- AWS Lambda. (Compute)
- DynamoDB. (NoSQL)
- AWS Cognito. (User 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**



Developer focus.

- **Developers should focus on the product, not infrastructure.**


Increasing focus on business logic
Bare metal
Virtual machines
Containers
Functions
Decreasing concern (and control) over stack implementation

AWS Lambda

(Serverless compute)

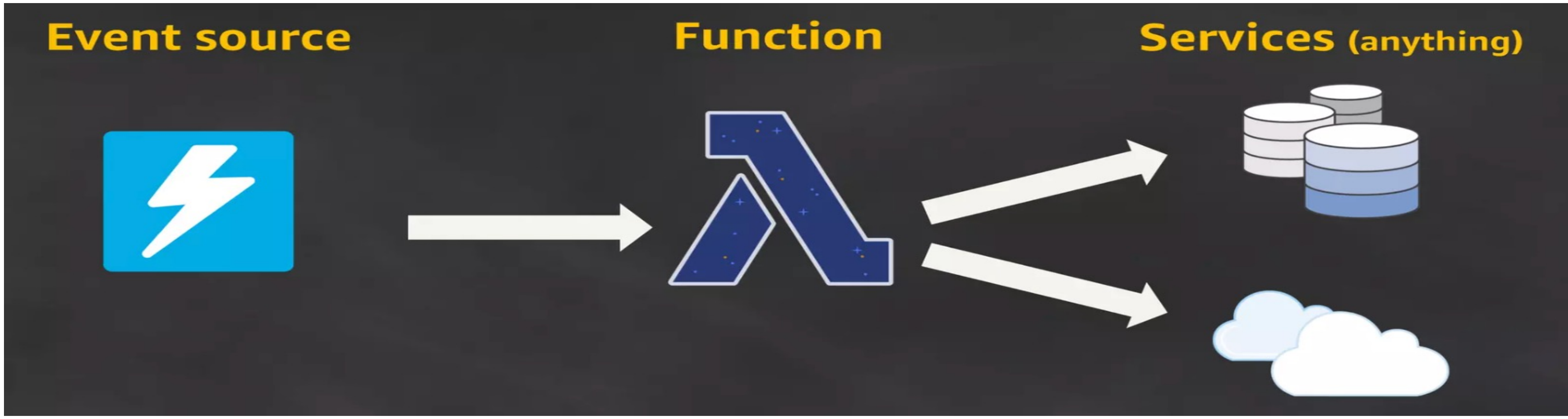
AWS Lambda

- **“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**

Serverless application



Event Source/ Trigger:
1. Request to HTTP endpoint.
2. Changes in data state – d/b, S3.
3. Changes in resources state.

Python
Node
Java
C#
Go

Anything !!!.

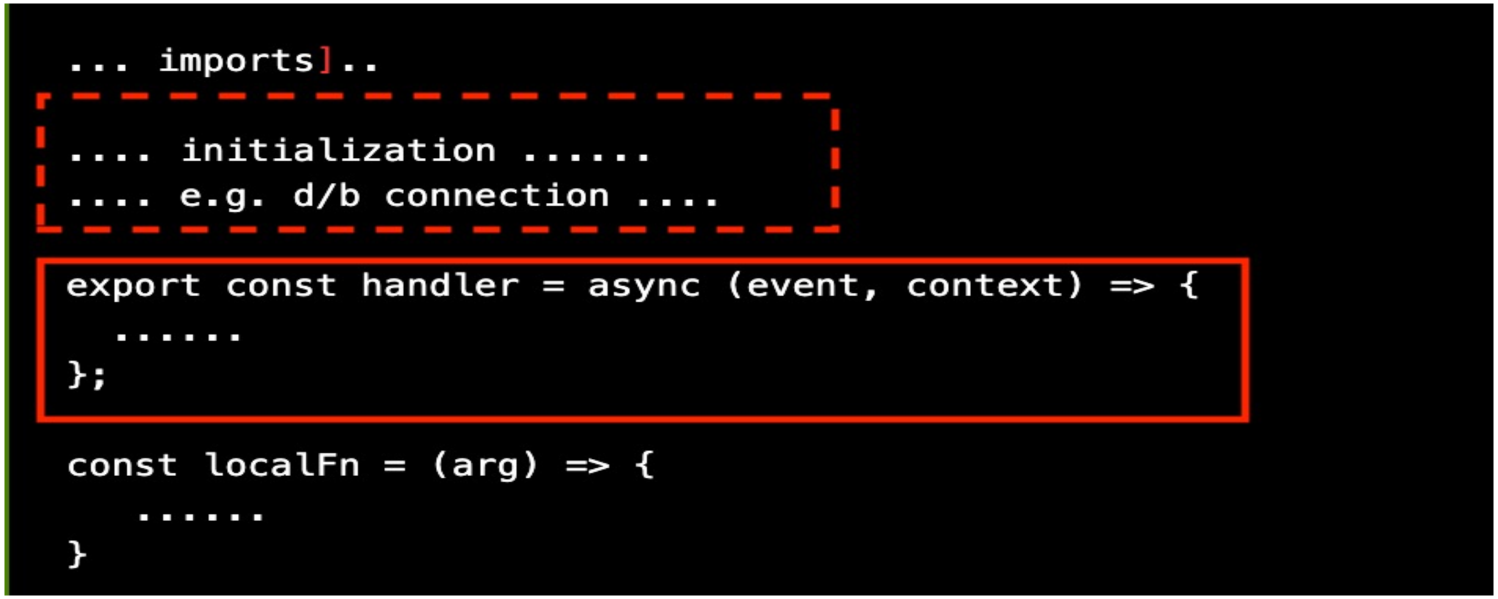
AWS Lambda

- **The Lambda service handles:**
  - **Auto scaling (horozintal)**
  - **Load balancing**
  - **OS management**
  - **Security isolation**
  - **Managing Utilization**

- **Characteristics:**
  - **Function as a unit of scale.**
  - **Function as a unit of deployment.**
  - **Stateless nature.**
  - Limited by time - short executions.
  - Run on-demand.
  - Pay per request and compute time – generous free tier.
  - **Do not pay for idle time.**

Anatomy of a Lambda function

- **Handler() – function to be executed upon invocation.**
- **Event object – the data sent during lambda function invocation**
- **Context object – access to runtime information.**



```
... imports]..

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

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

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

- **Initialization code executes before the handler.**
  - **<u>Cold start</u> only.**

Lambda Configuration

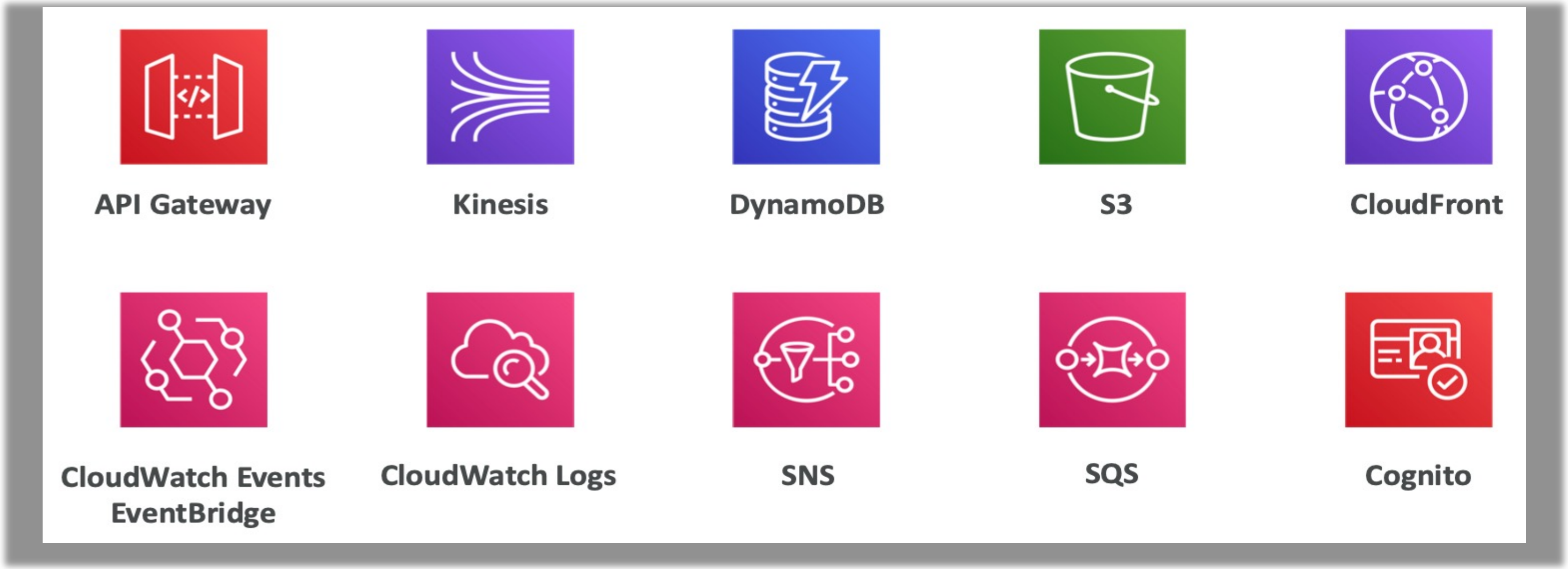
- Lambda exposes only a memory control to configure a function’s computer power.
  - The % of CPU core and network capacity are computed proportionally.
- RAM:
  - From 128MB to 3,008MB in 64MB increments
  - The more RAM you add, the more vCPU credits you get
  - At 1,792 MB, a function has the equivalent of one full vCPU
  - After 1,792 MB, you get more than one CPU, and need to use multi-threading in your code to benefit from it.
- For CPU-bound processing, increase the RAM allocation.
- Timeout: default is 3 seconds, maximum is 900 seconds (15 minutes).

Demo

- **Objective:**
  1. **Use the CDK to provision a ‘Hello World’ lambda function.**
  2. **invoked it from the AWS CLI.**
  3. **See console.log() statement output in Cloudwatch Logs**

Lambda integration

- **Main ones.**

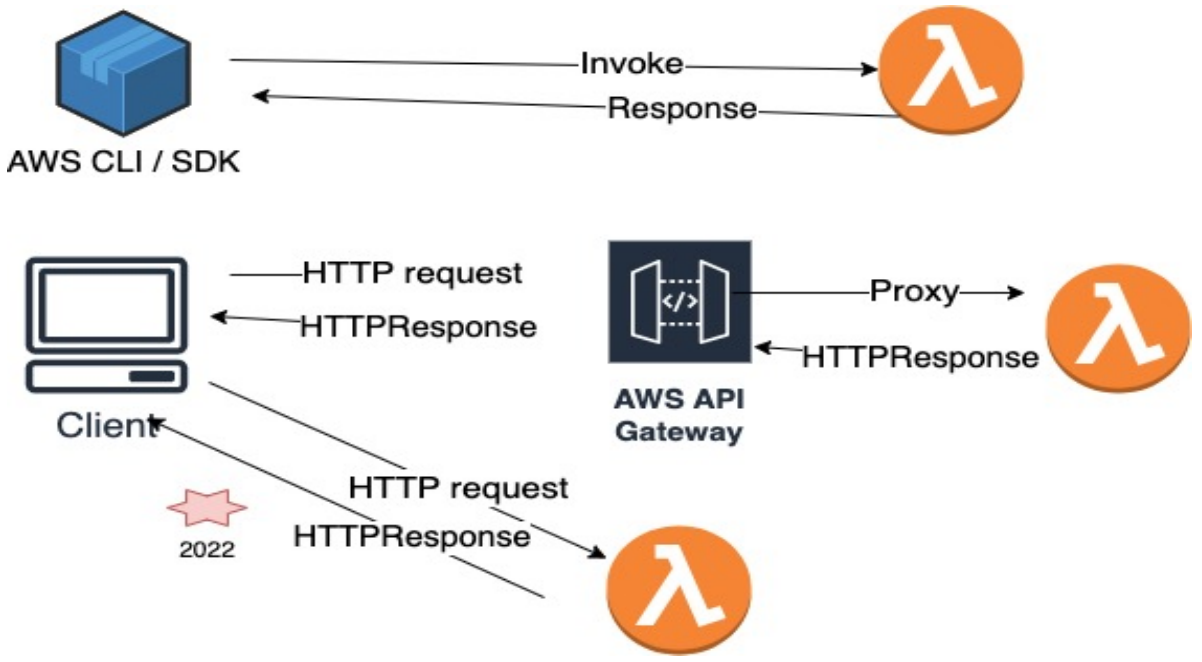


- **Integration models:**

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

Lambda – Synchronous Invocations

- **Event Source/ Trigger: CLI, SDK, API Gateway, Load Balancers, Function URLs**
  - Client waits for the results.
  - Error handling must happen client-side (retries, exponential backoff, etc...)

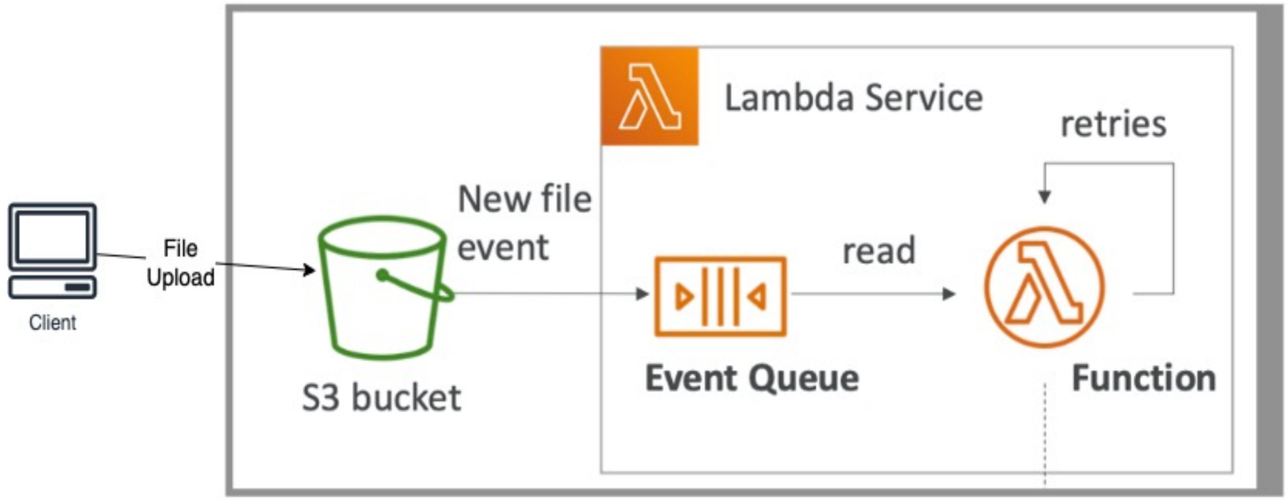


Demo

- **Objective:**
  1. **Use the CDK to provision a lambda function and generate a URL endpoint for invoking it.**
  2. **Test with Postman**
  3. **Make the endpoint private.**

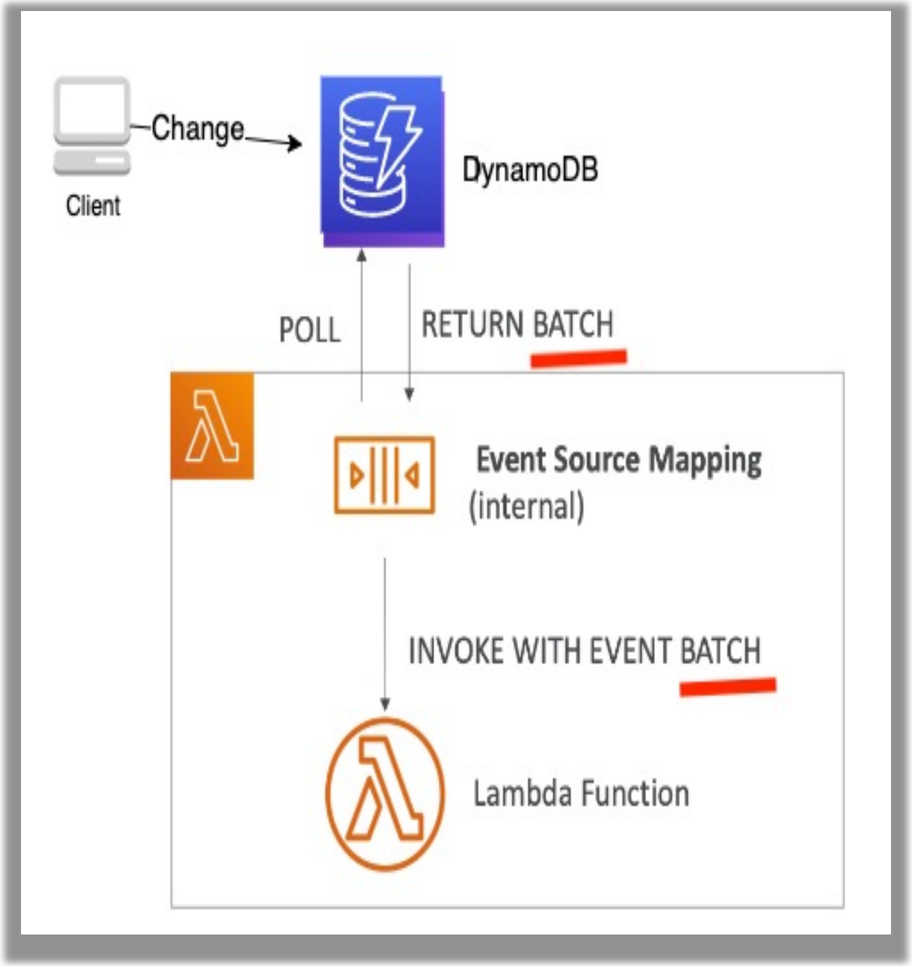
Lambda – Asynchronous Invocations

- **Trigger: S3 event, SNS, Cloudwatch Events.**
- **Lambda service places the events in a queue.**
- **Lambda service attempts to retry on errors – 3 retries, using exponential backoff.**
- **Make sure the processing is <u>idempotent</u> (in case of retries)**
- **Async invocations allow you to speed up the processing if you don't need to wait for a result.**



Lambda – Event source mapping. (Poll-based)

- **Sources: DynamoDB streams, SQS, Kinesis streams.**
- **Common denominator: records need to be <u>polled</u> from the source.**
- **Your Lambda function is invoked synchronously.**

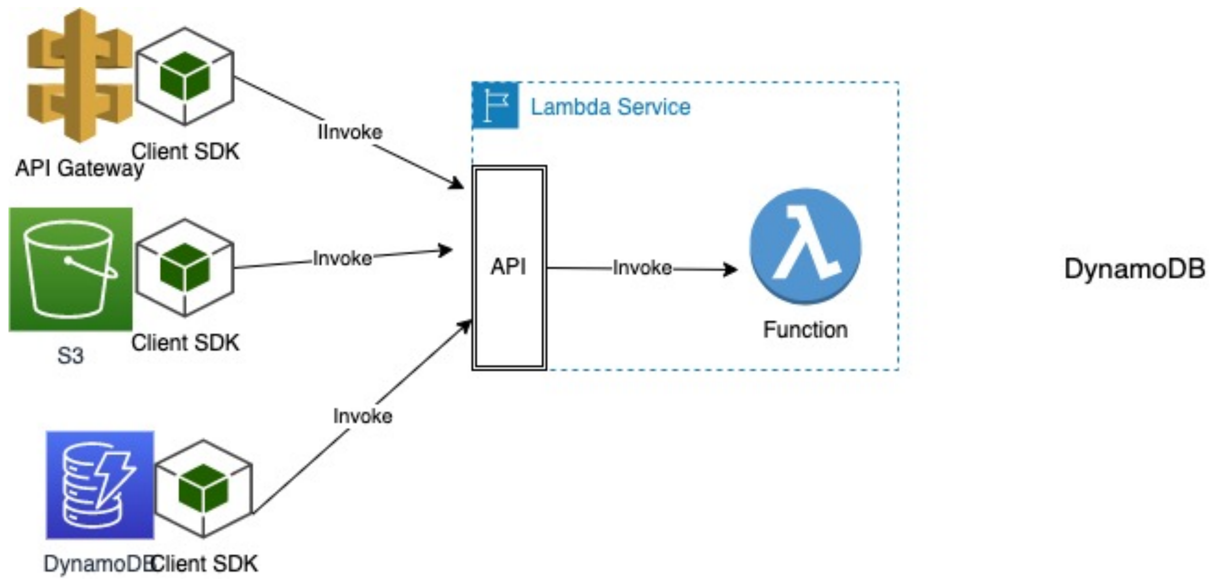


Streams & Lambda.

- Processes items in order.
- Start with new items, from the beginning or from timestamp.
- Processed items aren't removed from the stream (other consumers can read them).
- Low traffic: use batch window to accumulate records before processing.
- By default, if your function returns an error, the entire batch is reprocessed until the function succeeds, or the items in the batch expire.
- Can configure the event source mapping to:
  - discard old events
  - restrict the number of retries.

Lambda API

- Lambda Service provides an API
- Used by all other services that invoke Lambda functions across all models.
- Supports sync and async invocations.
- Can pass any event payload structure you want.
- Client included in every SDK,



Lambda Execution Role (IAM Role).

- **Grants the Lambda function permissions to access AWS services / resources.**
- **Many Mmanaged policies for Lambda, e.g.**
  - *AWSLambdaBasicExecutionRole* **– Upload logs to CloudWatch.**
  - *AWSLambdaDynamoDBExecutionRole* **– Read from DynamoDB Streams**
  - *AWSLambdaSQSQueueExecutionRole* **– Read from SQS**
  - *AWSLambdaVPCAccessExecutionRole* **– Deploy Lambda function in VPC.**

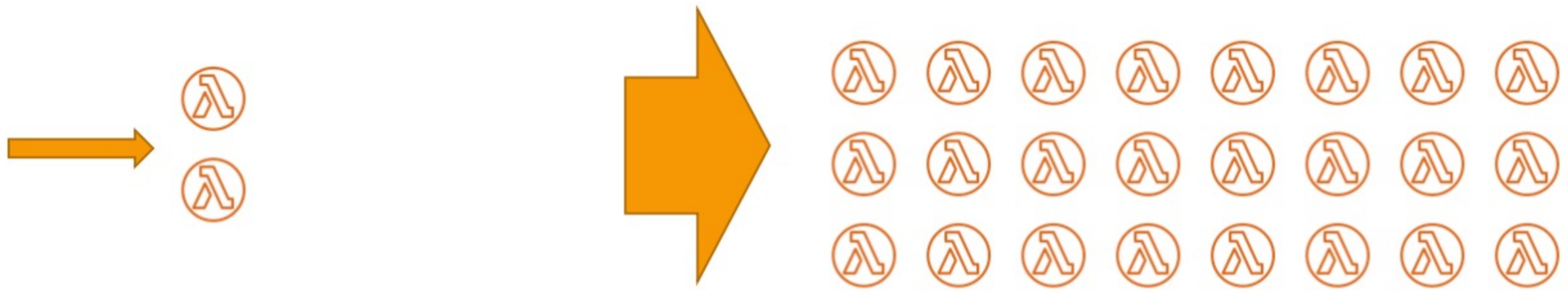
- **Best practice: create one Lambda Execution Role per function**

Lambda Resource based Policies.

- Use resource-based policies to give other AWS services (and accounts) permission to use your Lambda resources.
- Similar to S3 bucket policies for S3 bucket.
- An IAM principal can access Lambda:
  - if the IAM policy attached to the principal authorizes it (e.g. user access)
  - OR if the resource-based policy authorizes (e.g. service access)
- When an AWS service like Amazon S3 calls your Lambda function, the resource-based policy gives it access.

Lambda Concurrency and Throttling

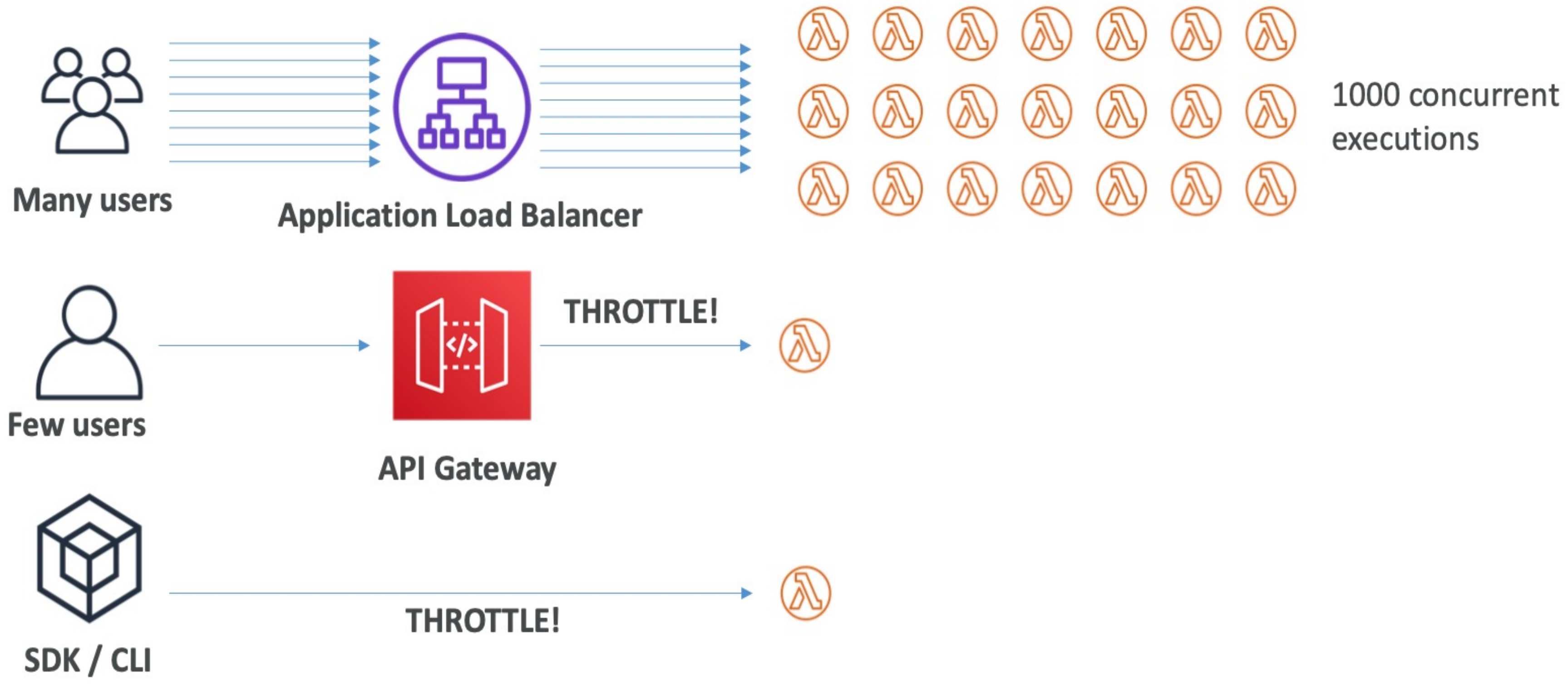
- Concurrency limit: up to 1000 concurrent executions



- Can set a "reserved concurrency" at the function level (=limit)
- Each invocation over the concurrency limit will trigger a "Throttle"
- Throttle behavior:
  - If synchronous invocation => return ThrottleError – 429
  - If asynchronous invocation => retry automatically and then go to DLQ
- • If you need a higher limit, open a support ticket

Lambda Concurrency Issue

- If you don't reserve (=limit) concurrency, the following can happen:



Cold Start & Provisioned Concurrency.

- **Cold Start:**
  - **New instance => code is loaded and code outside the handler runs (init)**
  - **If the init is large (code, dependencies, SDK...), this process can take some time.**
  - **First request served by a new instances has higher latency than the rest.**
- **Provisioned Concurrency:**
  - **Concurrency is allocated before the function is invoked (in advance)**
  - **So the cold start never happens and all invocations have low latency.**

To be continued ……