



AWS Lambda

Verified 2026-10-23

Run code without servers — the handler model, cold starts, sync vs async vs polled invocation, concurrency, retries, and hard limits.

Contents

Mental model	2
Handler and lifecycle	2
Configuration	4
Invocation types	5
Permissions	6
Concurrency and scaling	7
Errors and retries	8
Packaging and versions	10
Limits worth memorizing	11
Further reading	12

Mental model

Lambda runs your function in a short-lived, isolated execution environment that AWS creates when traffic arrives and discards when it goes quiet. Each environment handles one request at a time, so concurrency equals the number of environments running in parallel. Anything outside the handler survives only as long as its environment does, so treat memory and `/tmp` as a cache, never as storage.

Handler and lifecycle

Phase	What runs	How often
Init	Code outside the handler	Once per environment
Invoke	The handler	Once per request
Shutdown	Runtime cleanup	When AWS recycles it

The first request to a new environment pays the **cold start**: environment setup plus init. Later requests reuse

the environment (a **warm start**). Create SDK clients and read config in init so warm requests skip that work.

```
import { S3Client } from "@aws-sdk/client-s3";

const s3 = new S3Client({});

export const handler = async (event, context) => {
  console.log(context.awsRequestId);
  return { ok: true, n: event.items?.length ??
0 };
};
```

Gotcha: Globals leak between requests that share an environment. Per-request state in a module-level variable shows up in the next caller's request.

Configuration

Setting	Range
Memory	128–10,240 MB
Timeout	Up to 900 s (15 min)
/tmp storage	512–10,240 MB
Environment variables	4 KB total

CPU scales with memory: 1,769 MB gives one vCPU. Raising memory often makes a CPU-bound function cheaper, because it finishes faster. Measure before guessing.

```
aws lambda update-function-configuration \  
  --function-name my-fn \  
  --memory-size 1769 \  
  --timeout 30
```

Gotcha: The default timeout is 3 seconds. A function that works in a quick test and dies with

```
Task timed out after 3.00 seconds still has the default.
```

Invocation types

Type	Caller waits?	Retries
Synchronous	Yes	Caller's job
Asynchronous	No, queued	2 by Lambda
Event source mapping	No, Lambda polls	Source-specific

API Gateway, function URLs, and the SDK's

`RequestResponse` call are synchronous: the caller sees your return value or error. S3, SNS, and EventBridge invoke asynchronously: Lambda queues the event and retries on failure. SQS, Kinesis, and DynamoDB Streams use an **event source mapping**, where Lambda polls the source and invokes you with batches.

```
aws lambda invoke \  
  --function-name my-fn \  
  --cli-binary-format raw-in-base64-out \  
  --payload '{"items":[1,2]}' \  
  out.json
```

Gotcha: A function error still returns HTTP 200 from `invoke`. Check `FunctionError` in the response, not the status code.

Permissions

Policy	Controls
Execution role	What your function can call
Resource-based policy	Who can invoke your function

The execution role is an IAM role Lambda assumes for you. It needs at least `AWSLambdaBasicExecutionRole` to write logs. A service such as S3 or API Gateway invokes

you only if the function's resource-based policy allows it. The console and most IaC tools add that policy when you wire a trigger.

Tip: Grant the execution role the exact actions and resources the function uses. One shared admin role across functions turns a single bug into an account-wide problem.

Concurrency and scaling

Setting	Effect
Account limit	1,000 environments per Region by default
Reserved concurrency	Guarantees and caps one function
Provisioned concurrency	Keeps environments pre-initialized

Each function adds at most 1,000 new environments every 10 seconds, up to the account limit. Reserved

concurrency carves capacity out of the account pool for a function and caps it at that number. Set it to 0 to stop a function entirely. Provisioned concurrency removes cold starts for that many environments, at a cost whether or not they are used.

Gotcha: One busy function can use the whole account pool and throttle every other function with `TooManyRequestsException` (HTTP 429). Reserve concurrency on the functions that must stay available.

Errors and retries

Mechanism	Applies to
Async retries (0-2)	Asynchronous invocation
On-failure destination or DLQ	Asynchronous invocation
Partial batch response	SQS, Kinesis, DynamoDB Streams

Async events that still fail after the retries go to the **on-failure destination** or dead-letter queue you configured, and are dropped if you configured neither. For batch sources, return the IDs that failed so Lambda retries only those records.

```
export const handler = async (event) => {
  const failed = [];
  for (const r of event.Records) {
    try {
      await work(JSON.parse(r.body));
    } catch {
      failed.push({ itemIdentifier:
r.messageId });
    }
  }
  return { batchItemFailures: failed };
};
```

Gotcha: Retries mean at-least-once delivery, so make handlers idempotent or a retry charges the card twice. For SQS, enable `ReportBatchItemFailures` on the mapping or the returned list is ignored.

Packaging and versions

Item	Limit or rule
Zip package	50 MB zipped, 250 MB unzipped
Container image	Up to 10 GB
Layers	5 per function, count toward 250 MB
\$LATEST	Mutable working copy

Publishing a **version** freezes the code and configuration. An **alias** such as `prod` is a pointer to a version that you can move, or split by weight for canary releases. Point triggers at the alias, not at `$LATEST`.

```
aws lambda publish-version --function-name my-fn
aws lambda update-alias \
  --function-name my-fn \
  --name prod \
  --function-version 7
```

Limits worth memorizing

Limit	Value
Sync payload	6 MB request, 6 MB response
Async payload	1 MB
Streamed response	200 MB
Max duration	900 s

The 6 MB sync cap is the one that bites. For larger data, put the object in S3 and pass its key, or use response streaming. Some quotas are raisable through Service Quotas, so check your account's values before designing near an edge.

Further reading

- [Lambda Developer Guide](#)
- [Lambda quotas](#)
- [Serverless Land patterns](#)