AWS LAMBDA (NODE.JS) CheatSheet

1) Core mental model

- You write a handler; Lambda runs it on each invocation.
- Cold start: first invoke in a fresh environment runs init code (top-level module code) + handler.
- Warm invoke: same environment reused; keep expensive clients outside handler (DB, AWS SDK clients).
- Your code ends when:
  - the handler returns, throws, times out, or the process exits.

2) Runtimes (what to pick)

- Common current Node runtimes include nodejs22.x, nodejs20.x, nodejs18.x (availability and deprecation dates can change; always verify in Lambda runtimes page)

3) Handler signatures (async is the default style)

CommonJS

```
// index.js
exports.handler = async (event, context) => {
  return { ok: true };
};
```

ESM

```
// index.mjs
export const handler = async (event, context) => {
  return { ok: true };
};
```

AWS notes Lambda supports callback-based handlers for Node.js runtimes (details/constraints in docs)

4) The event + context objects you'll actually use

Context highlights

- context.awsRequestId
- context.functionName, context.functionVersion
- context.getRemainingTimeInMillis()
- context.invokedFunctionArn

```
export const handler = async (event, context) => {
  console.log("requestId:", context.awsRequestId);
  console.log("remaining(ms):", context.getRemainingTimeInMillis());
  return { ok: true };
};
```

5) Logging (CloudWatch)

- Use console.log/info/warn/error
- Log JSON for searchability:

```
console.log(JSON.stringify({ level: "info", msg: "hello", eventType: event?.type }));
```

6) Environment variables (config without redeploying code)

- Read via process.env.MY_KEY
- Don't store secrets in plain env vars; prefer AWS Secrets Manager (AWS explicitly recommends this).

```
const TABLE = process.env.TABLE_NAME;

export const handler = async () => {
  if (!TABLE) throw new Error("Missing TABLE_NAME");
  return { table: TABLE };
};
```

BY: PAUL CARLO TORDECILLA

AWS LAMBDA (NODE.JS) CheatSheet

6) Environment variables (config without redeploying code)

- **Read via process.env.MY_KEY**
- **Don't store secrets in plain env vars; prefer AWS Secrets Manager (AWS explicitly recommends this)**

```
const TABLE = process.env.TABLE_NAME;

export const handler = async () => {
 if (!TABLE) throw new Error("Missing TABLE_NAME");
 return { table: TABLE };
};
```

7) Packaging & dependencies (ZIP is most common)

Lambda Node deployments are typically:

- **.zip archive (your handler file + node_modules)**
- **or container image (when you need custom OS/libs)**

Basic ZIP structure

```
my-fn.zip
 index.js (or .mjs)
 node_modules/
 package.json
```

Layers (share deps across functions)

- **Put Node deps under nodejs/node_modules/... inside the layer zip.**

8) "Hello HTTP" responses (API Gateway proxy)

If you use API Gateway Lambda proxy integration, your Lambda returns a specific structure (statusCode/ headers/body).

HTTP JSON response

```
export const handler = async (event) => {
 return {
  statusCode: 200,
  headers: { "content-type": "application/json" },
  body: JSON.stringify({ message: "OK" }),
 };
};
```

Common event fields you'll read

- **REST API (v1) proxy: event.httpMethod, event.path, event.queryStringParameters, event.headers, event.body**
- **HTTP API (v2) proxy: event.requestContext.http.method, event.rawPath, event.rawQueryString, event.headers, event.body**

9) Timeouts, memory, ephemeral storage (practical rules)

- **Timeout: set high enough for worst-case; always watch getRemainingTimeInMillis()**
- **Memory: also affects CPU; bump memory if slow CPU-bound work**
- **/tmp: use for temp files (cache, downloads). (Size configurable in Lambda settings.)**

10) Concurrency & scaling

- **Lambda scales by running multiple environments in parallel.**
- **Key knobs:**
  - **Reserved concurrency (cap + guarantee)**
  - **Provisioned concurrency (reduce cold starts)**
- **For Node.js runtimes, AWS also documents Lambda Managed Instances behavior and concurrency model**

BY: PAUL CARLO TORRECILLA

AWS LAMBDA (NODE.JS) CheatSheet

11) Retries, DLQ, and Destinations (don't lose events)

- **Async invocations can retry automatically; configure DLQ (SQS/SNS) or Destinations (on success/failure).**
- **For stream sources (SQS/Kinesis/DynamoDB Streams), handle partial failures carefully (batch item failures).**

(If you tell me your trigger type, I can drop the exact retry + failure patterns to use.)

12) Permissions (IAM) quick map

- **Lambda execution role controls what your code can access.**
- **Add least-privilege policies for:**
  - **DynamoDB (dynamodb:GetItem, etc.)**
  - **S3 (s3:GetObject, etc.)**
  - **SQS/SNS**
  - **CloudWatch Logs**
- **For networked resources in a VPC (RDS, private services), configure VPC settings.**

13) Versions, aliases, and safe deploys

- **Publish version for immutable builds.**
- **Use aliases like dev, staging, prod.**
- **Gradual rollout:**
  - **weighted aliases (canary) + monitoring + rollback.**

14) Local testing tips

- **Write handlers as pure functions where possible.**
- **Use fixture JSON files for event payloads.**
- **For API Gateway, save sample events from the console and replay locally.**

15) AWS CLI quick commands

Create function

```
aws lambda create-function \
  --function-name myFn \
  --runtime nodejs22.x \
  --handler index.handler \
  --zip-file fileb://function.zip \
  --role arn:aws:iam::<ACCOUNT_ID>:role/<ROLE_NAME>
```

Update code

```
aws lambda update-function-code \
  --function-name myFn \
  --zip-file fileb://function.zip
```

Invoke

```
aws lambda invoke --function-name myFn out.json
cat out.json
```

***(Runtime identifiers vary; confirm on the Lambda runtimes page.)**

BY: PAUL CARLO TORDECILLA

AWS LAMBDA (NODE.JS) CheatSheet

16) Common Node.js Lambda patterns (battle-tested)

Reuse clients across invocations

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
const ddb = new DynamoDBClient({}); // created once (init)

export const handler = async () => {
  // use ddb here
  return { ok: true };
};
```

Guard against hanging work

```
export const handler = async (event, context) => {
  const deadline = context.getRemainingTimeInMillis() - 200; // 200ms safety
  // if work might exceed deadline, stop early
  return { ok: true };
};
```

17) Quick troubleshooting checklist

- **Timeouts → increase timeout, optimize code, raise memory (CPU), remove slow network calls, ensure VPC config is correct**
- **Cannot find module → zip structure wrong / missing node_modules**
- **Large bundle → trim deps, bundle with esbuild, move big deps to a layer**
- **API Gateway 502/format issues → ensure proxy response shape (statusCode, body as string)**

BY: PAUL CARLO TORDECILLA
BY: PAUL CARLO TORDECILLA