AWS Step Functions and Lambda Durable Functions

Version durable-execution-sdk-js 3.0 · verified 2026-11-02

Orchestrate with Step Functions state machines or Lambda durable functions — states, retries, Map, callbacks, replay, pricing and a choice table.

Contents

Mental model 2

States in ASL 2

Standard vs Express 3

Data, variables and size 4

Errors and retries 5

Integrations and callbacks 6

Map and Distributed Map 7

Lambda durable functions 8

Durable callbacks and configuration 9

Pricing and choosing 11

Further reading 11

Mental model

A workflow that waits hours for a person or retries a flaky API cannot live in one function invocation, so something must remember where it stopped. Step Functions remembers from outside your code: you declare states in JSON, and the service runs them and records every transition. Lambda durable functions remember from inside your code: the SDK checkpoints each step, and on resume your handler runs again from the top while completed steps return their saved results.

States in ASL

State	Does
`Task`	Calls Lambda or an AWS API
`Choice`	Branches on input
`Wait`	Pauses for a time or until a timestamp
`Parallel` / `Map`	Runs branches or items concurrently
`Pass`, `Succeed`, `Fail`	Shape data, or end

A state machine is Amazon States Language (ASL): a `StartAt` name and a `States` map where each state names its `Next` or sets `End`. Set `QueryLanguage` to `JSONata` for new machines; if omitted it defaults to JSONPath.

```
{
  "QueryLanguage": "JSONata",
  "StartAt": "Charge",
```

```
  "States": {
    "Charge": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "Arguments": { "FunctionName": "charge",
        "Payload": "{% $states.input %}" },
      "Output": "{% $states.result.Payload %}",
      "End": true
    }
  }
}
```

Standard vs Express

Property	Standard	Express
Max duration	1 year	5 minutes
Semantics	Exactly-once	At-least-once (async)
History	25,000 events, kept 90 days	CloudWatch Logs only
Billing	Per state transition	Per request plus duration

Standard suits long, auditable business processes; Express suits high-volume, short event processing. The type is fixed at creation. Synchronous Express executions are at-most-once.

```
aws stepfunctions create-state-machine \
  --name ingest --type EXPRESS \
```

```
  --definition file://ingest.asl.json \
  --role-arn "$SFN_ROLE"
```

> **Gotcha:** A Standard execution fails when its history reaches 25,000 events. A long loop of small states hits that limit; move the loop into a child execution or a Distributed Map.

Data, variables and size

Field (JSONata)	Purpose
`Arguments`	Input to the task
`Output`	What the state passes on
`Assign`	Store variables for later states
`$states.input` / `.result`	Reserved variables

JSONata replaces JSONPath's five fields (`InputPath`, `Parameters`, `ResultSelector`, `ResultPath`, `OutputPath`) with two. Variables set with `Assign` let later states read earlier results without threading them through every output.

```
"Lookup": {
  "Type": "Task",
  "Resource": "arn:aws:states:::lambda:invoke",
  "Arguments": { "FunctionName": "lookup" },
  "Assign": {
    "customer": "{% $states.result.Payload %}"
```

```
  },
  "Next": "Notify"
}
```

> **Gotcha:** Input and output for every state, task and execution are capped at 256 KiB, and exceeding it fails with `States.DataLimitExceeded`. Store large data in S3 and pass the key.

Errors and retries

Retry field	Default
`IntervalSeconds`	1
`MaxAttempts`	3 (0 = never retry)
`BackoffRate`	2.0
`MaxDelaySeconds`	No cap
`JitterStrategy`	`NONE` (or `FULL`)

`Retry` re-runs the failed state; `Catch` moves to a fallback state when retries run out. Both match on `ErrorEquals`, checked in order. `States.ALL` matches any error and must come last.

```
"Retry": [{
  "ErrorEquals": [
    "Lambda.TooManyRequestsException"],
  "IntervalSeconds": 2, "MaxAttempts": 6,
  "BackoffRate": 2, "JitterStrategy": "FULL"
```

```
}],
"Catch": [{
  "ErrorEquals": ["States.ALL"],
  "Next": "Refund"
}]
```

> **Gotcha:** ASL sets no useful task timeout by default (`TimeoutSeconds` defaults to 99,999,999), so a worker that never answers leaves the execution stuck. Set a timeout, plus `HeartbeatSeconds` on callback tasks.

Integrations and callbacks

Pattern	Resource suffix	Waits for
Request Response	none	The HTTP response
Run a job	`.sync`	Job completion (Standard only)
Wait for callback	`.waitForTaskToken`	`SendTaskSuccess`
AWS SDK	`aws-sdk:<service>:<action>`	Any API action

Optimized integrations such as `lambda:invoke` or `sqs:sendMessage` call services without a Lambda in between; SDK integrations reach over 200 more services. A callback task hands its token to an external system and pauses until that system calls `SendTaskSuccess` or `SendTaskFailure`.

```
"Approve": {
  "Type": "Task",
  "Resource":
  "arn:aws:states:::lambda:invoke.waitForTaskToken",
  "HeartbeatSeconds": 3600,
  "Arguments": { "FunctionName": "ask-approver",
    "Payload": {
      "token": "{% $states.context.Task.Token %}"
    } },
  "Next": "Ship"
}
```

Map and Distributed Map

Mode	Input	Concurrency
Inline	JSON array in state input	Limited beyond ~40 items
Distributed	Array or S3 objects (CSV, JSON, Parquet)	Up to 10,000 child executions

Distributed mode runs each item, or batch of items, as a child execution, so one map can process millions of objects without hitting the parent's history or payload limits. Write results to S3 with `ResultWriter` when they exceed 256 KiB.

```
"Each": {
  "Type": "Map", "MaxConcurrency": 500,
```

```
  "ItemReader": {
    "Resource": "arn:aws:states:::s3:listObjectsV2",
    "Arguments": { "Bucket": "in", "Prefix": "d/" }
  },
  "ItemProcessor": {
    "ProcessorConfig": { "Mode": "DISTRIBUTED",
      "ExecutionType": "EXPRESS" },
    "StartAt": "Work", "States": { "Work": {
      "Type": "Pass", "End": true } }
  },
  "End": true
}
```

> **Tip:** Set `MaxConcurrency` to what the downstream service tolerates. 10,000 parallel Lambda calls will throttle an account with the default concurrency limit.

Lambda durable functions

`DurableContext` method	Does
`step(name, fn)`	Runs `fn` once, checkpoints the result
`wait(duration)`	Suspends with no compute charge
`invoke(name, fnArn, input)`	Calls another function durably
`map` / `parallel`	Concurrent durable work
`runInChildContext`	Groups several operations

Wrap the handler with `withDurableExecution`. Managed runtimes support Node.js (22, 24), Python (3.13, 3.14), Java (17, 21, 25) and .NET (8, 10). Node.js and Python runtimes include the SDK, while Java and .NET need it bundled; bundling is recommended everywhere for production.

```
import { withDurableExecution }
  from "@aws/durable-execution-sdk-js";

export const handler = withDurableExecution(
  async (event, ctx) => {
    const hold = await ctx.step("reserve",
      async () => reserve(event.orderId));
    await ctx.wait({ hours: 24 });
    return ctx.step("charge",
      async () => charge(hold));
  });
```

> **Gotcha:** On every resume the handler re-runs from the top. Code outside `step()` runs again, so `Date.now()`, random IDs and API calls there produce different values on replay. Put them inside a step.

Durable callbacks and configuration

Setting	Value
`ExecutionTimeout`	Required, 1–31,622,400 s (366 days)

Setting	Value
`RetentionPeriodInDays`	1–90, default 14
Operations per execution	30,000
Payload per operation	1 MB

Durable execution is enabled only at function creation, via `--durable-config`, and cannot run on an unqualified function name. `createCallback` or `waitForCallback` returns a callback ID; an external system completes it with `SendDurableExecutionCallbackSuccess`, `Failure` or `Heartbeat`.

```
{
  "Effect": "Allow",
  "Action": [
    "lambda:SendDurableExecutionCallbackSuccess",
    "lambda:SendDurableExecutionCallbackFailure",
    "lambda:SendDurableExecutionCallbackHeartbeat"
  ],
  "Resource":
    "arn:aws:lambda:eu-west-1:1234:function:f:*"
}
```

> **Warning:** Invoking through `$LATEST` means a deploy mid-execution replays checkpoints against different code. Use `$LATEST` only for prototyping; invoke a published version or an alias that points to one.

Pricing and choosing

Need	Choose
Visual, auditable, mostly AWS API calls	Step Functions Standard
Very high volume, under 5 minutes	Step Functions Express
Logic best written as code, one team	Lambda durable functions
Fan out over millions of S3 objects	Distributed Map

Step Functions Standard bills per state transition; Express bills per request plus GB-seconds. Durable functions add charges for durable operations, data written, and retention to normal Lambda compute, and waits incur no duration charge. Prices below are us-east-1 list prices on the verification date.

```
Standard  $0.025 per 1,000 transitions
          (4,000 free per month)
Express   $1.00 per 1M requests + duration
Durable   $8.00 per 1M operations
          + $0.25 per GB written + retention
```

Further reading

- Step Functions Developer Guide
- Handling errors in Step Functions
- Lambda durable functions
- AWS Durable Execution SDK Developer Guide