



DVA-C02 Domain 1: SDKs, Credentials and Lambda Code

AWS Certified Developer - Associate, Domain 1 (Development with AWS Services) part 1 | SDK credential chain, retries and backoff, pagination and waiters, Lambda handler model, invocation types, limits, idempotency

DVA-C02 Domain 1 SDK Lambda boto3

🔍 Credentials: where the SDK finds them

How your code gets AWS credentials

In one sentence

●○○○

Never write keys in code: let the SDK find temporary credentials from the role the code runs under.

Practitioner

●●○○

The SDK walks a provider chain and stops at the first hit:

- parameters passed to the client in code
- environment variables `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_SESSION_TOKEN`
- web identity token file (EKS IRSA, GitHub OIDC)
- shared `~/.aws/config` and `~/.aws/credentials` profiles (SSO, assume role, `credential_process`)
- container endpoint (ECS task role, EKS Pod Identity)
- EC2 instance metadata (instance profile, IMDSv2)

Expert

●●●○

On AWS compute the role is injected for you:

- Lambda: execution role, keys in the function's environment variables
- ECS: task role via `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI`
- EC2: instance profile, credentials refreshed before expiry

Under the hood

●●●●

Temporary credentials come from STS (`AssumeRole`, `AssumeRoleWithWebIdentity`, `GetSessionToken`); the SDK refreshes them automatically before they expire.

Analogy

A hotel key card handed at the desk, valid for your stay only, instead of a copy of the master key in your pocket.

Environment variables the SDKs and the CLI read

Flag	Arg	Description
CREDENTIALS		
<code>AWS_ACCESS_KEY_ID</code> / <code>AWS_SECRET_ACCESS_KEY</code>		access key pair; beat every file-based profile <i>note: on a laptop, prefer SSO profiles</i>
<code>AWS_SESSION_TOKEN</code>		required with temporary (STS) keys e.g. set by Lambda for the execution role
<code>AWS_PROFILE</code>	<i>NAME</i>	pick a profile from <code>~/.aws/config</code> e.g. <code>AWS_PROFILE=dev</code>
<code>AWS_ROLE_ARN</code> + <code>AWS_WEB_IDENTITY_TOKEN_FILE</code>		assume a role with an OIDC token (IRSA, CI)
BEHAVIOUR		
<code>AWS_REGION</code> / <code>AWS_DEFAULT_REGION</code>	<i>REGION</i>	target region e.g. <code>eu-west-3</code>
<code>AWS_RETRY_MODE</code>	<i>legacy/standard/...</i>	retry algorithm
<code>AWS_MAX_ATTEMPTS</code>	<i>N</i>	total attempts including the first call e.g. <code>AWS_MAX_ATTEMPTS=5</code>
<code>AWS_ENDPOINT_URL</code>	<i>URL</i>	send calls to another endpoint e.g. <code>http://localhost:4566</code> for a local emulator

📖 EXAM TIP

Questions that say 'the application on EC2 must access S3 securely' want an **IAM role attached as an instance profile**, never access keys stored on the instance or in code. For ECS the answer is the **task role** (not the task execution role, which only pulls images and writes logs).

🔄 Calling APIs well: errors, retries, pages

Error	HTTP	Meaning	What the code does
<code>ThrottlingException</code> , <code>TooManyRequestsException</code>	400 / 429	request rate above the API or account quota	retry with exponential backoff and jitter; raise the quota
<code>ProvisionedThroughputExceededException</code>	400	DynamoDB partition or table capacity exceeded	SDK retries; fix hot keys, on-demand mode or more capacity
<code>AccessDeniedException</code>	403	IAM or resource policy denies	do not retry; fix the policy
<code>ValidationException</code>	400	bad parameter	do not retry; fix the request
<code>InternalServerError</code> , <code>ServiceUnavailable</code>	500 / 503	transient service-side fault	retry with backoff
<code>ConditionalCheckFailedException</code>	400	DynamoDB condition false (optimistic lock)	re-read the item, decide, retry the write

📄 Tadata | tadata.fr | contact@tadata.codes

```

boto3: retries, paginator, waiter
import boto3
from botocore.config import Config
cfg = Config(retries={"max_attempts": 8,
                    "mode": "adaptive"},
            connect_timeout=2, read_timeout=10)
s3 = boto3.client("s3", config=cfg)
# paginator follows ContinuationToken for you
for page in s3.get_paginator("list_objects_v2").paginate(
    Bucket="orders-eu", Prefix="2026/"):
    for obj in page.get("Contents", []):
        print(obj["Key"], obj["Size"])
# waiter polls HeadBucket until the bucket exists
s3.get_waiter("bucket_exists").wait(Bucket="orders-eu")

```

Backoff delays and Lambda concurrency

PROBLEM A client retries a throttled call with base 100 ms, cap 5 s, 5 retries (full jitter). The same API is called by a Lambda function receiving 200 requests per second, each lasting 300 ms.

GIVEN

- base $b = 100$ ms, cap $c = 5\,000$ ms, retry number $n = 0.4$
- full jitter: $\text{wait} = \text{random}(0, \min(c, b \times 2^n))$
- arrival rate $\lambda = 200$ requests/s, average duration $d = 0.3$ s

1 Upper bound of each wait

100, 200, 400, 800, 1600 ms

Why: doubles each retry, below the 5 s cap

2 Worst-case total wait

$100 + 200 + 400 + 800 + 1600 = 3100$ ms

Why: jitter draws between 0 and the bound, so the average is about half: 1.55 s

3 Concurrency needed

$C = \lambda \times d = 200 \times 0.3 = 60$

Why: Little's law: concurrent executions = rate $\times$ duration

4 Check against the quota

$60 < 1000$ default concurrency per region

Why: room left for other functions; reserve some if this one is critical

ANSWER Retries add at most 3.1 s of waiting; the function needs about 60 concurrent environments.

CHECK Doubling the duration to 0.6 s doubles concurrency to 120: speed up code before raising quotas.

🔗 Retry modes

standard (default in current SDKs) retries throttling, 5xx and transient network errors with exponential backoff and jitter, 3 attempts in total by default. adaptive adds client-side rate limiting: use it for batch jobs that hammer one API. legacy is the old per-SDK behaviour. Never retry 4xx errors other than throttling.

🔗 Lambda: the programming model

```

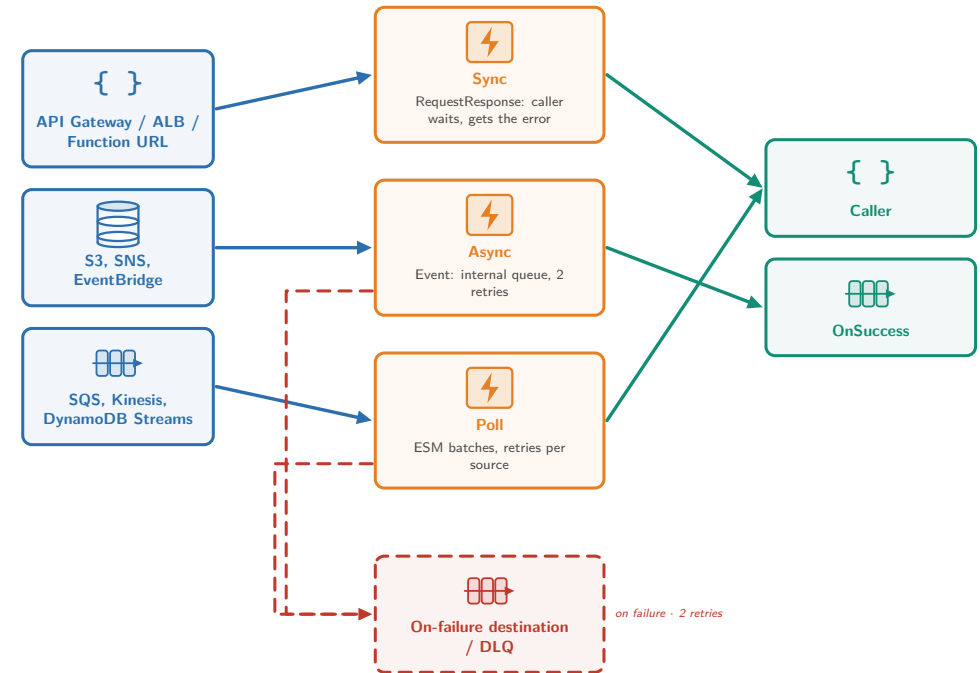
handler.py: init once, handle many
import json, os, boto3
TABLE = boto3.resource("dynamodb").Table(
    os.environ["TABLE_NAME"]) # init: once per env

def handler(event, context):
    order = json.loads(event["body"]) # API GW proxy
    TABLE.put_item(Item=order,
        ConditionExpression="attribute_not_exists(pk)")
    left = context.get_remaining_time_in_millis()
    print(json.dumps({"rid": context.aws_request_id,
        "ms_left": left}))

    return {"statusCode": 201,
        "headers": {"Content-Type": "application/json"},
        "body": json.dumps({"id": order["pk"]})}

```

Three ways Lambda is invoked



Limit (as of 2026)	Value	Why it bites
Memory	128 MB to 10 240 MB, 1 MB steps	CPU grows with memory (about 1 vCPU at 1 769 MB)
Timeout	max 900 s (15 min)	API Gateway integration stops waiting long before

Synchronous payload	6 MB request and response	large files go through S3 presigned URLs
Asynchronous payload	1 MB since late 2025 (was 256 KB)	check the current quota page
Environment variables	4 KB in total	secrets belong in Secrets Manager / Parameter Store
Deployment package	50 MB zipped upload, 250 MB unzipped with layers	container images up to 10 GB
Layers	5 per function	count toward the 250 MB unzipped
/tmp ephemeral storage	512 MB default, up to 10 240 MB	not shared between environments
Concurrency	1 000 per region per account by default (soft)	a burst can throttle every function

Version — An immutable snapshot of code and configuration (: 1, : 2); \$LATEST is the only mutable one.

Alias — A named pointer to a version (prod → 7) that can split traffic between two versions with weights; event sources and API stages should call the alias.

⚠ Failures, duplicates and idempotency

- 1 **Synchronous** the caller receives the error (FunctionError header); the client or API Gateway decides to retry
- 2 **Asynchronous** Lambda retries twice (configurable 0-2), with delays; event age up to 6 h; then on-failure destination or DLQ (SQS / SNS)
- 3 **SQS event source** batch returns to the queue after the visibility timeout; report batchItemFailures to retry only failed messages; redrive to a DLQ after maxReceiveCount
- 4 **Kinesis / DynamoDB Streams** retries the batch until success or record expiry and blocks the shard; set BisectBatchOnFunctionError, MaximumRetryAttempts, on-failure destination

What happens when the handler throws

MYTH

✗ Lambda runs each event exactly once

✗ A global variable is shared by all invocations

✗ Raising the timeout fixes slow API calls

FACT

✓ Delivery is at least once
Retries and duplicate events happen: make writes idempotent (conditional writes, idempotency key table, Powertools idempotency utility).

✓ It lives in one execution environment only
Concurrent invocations run in separate environments; use it for caches and SDK clients, never for shared state.

✓ Fix the cause and set client timeouts below the function timeout
A hung downstream call then fails fast and is retried instead of burning 15 minutes.

⚠ Recursive loops

A function that writes to the S3 bucket or SQS queue that triggers it invokes itself forever. Lambda detects some recursive loops between Lambda, SQS, SNS and S3 and stops them after about 16 iterations, but design them out: write to another bucket or prefix and filter the trigger.

Developer baseline for a Lambda function

✓ baseline / mandatory □ recommended – verify in your context

- ✓ Create SDK clients and read configuration outside the handler
- ✓ Least-privilege execution role, one per function
- ✓ Secrets read from Secrets Manager or Parameter Store and cached, never in plain environment variables
- **MUST** Idempotency key for every write triggered by an event
- **SHOULD** Structured JSON logs with the request id [Lambda JSON log format](#) or [Powertools Logger](#)
- **MUST** Destinations or DLQ on every asynchronous or stream trigger

🚩 Recap and self-test

Key takeaways

01

Roles, not keys

instance profile, task role, execution role; the SDK finds them

02

Retry only what is transient

throttling and 5xx with backoff and jitter

03

Paginate and wait with SDK helpers

never assume one page

04

Init outside the handler

clients and config reused across invocations

05

At least once everywhere

idempotent writes, DLQs and destinations

Q1 Which credential source wins: environment variables or a profile in ~/.aws/credentials?

A Environment variables: they come before shared files in the provider chain.

Q2 Which role does an ECS container use to call DynamoDB?

A The task role (taskRoleArn); the execution role only pulls images and writes logs.

Q3 Which SDK retry mode adds client-side rate limiting?

A adaptive.

Q4 A List API returned 1 000 items and a NextToken. What do you use?

A A paginator (or loop on NextToken until it is absent).

Q5 Maximum Lambda timeout and memory?

A 900 seconds; 10 240 MB.

Q6 How many times does Lambda retry a failed asynchronous invocation by default?

A Two retries (three attempts), then the on-failure destination or DLQ.

Q7 How do you retry only the failed messages of an SQS batch?

A Enable `ReportBatchItemFailures` and return `batchItemFailures` with the failed message IDs.

Q8 Which context method tells how long the invocation may still run?

A `get_remaining_time_in_millis()` (Python; `getRemainingTimeInMillis()` in Node.js).

Q9 Where should an API Gateway stage or an SQS trigger point: version, alias or \$LATEST?

A An alias, so you can shift traffic and roll back without changing the trigger.