Developing Event-Driven Serverless Solutions

SERVERLESS · AWS LAMBDA · EVENT SOURCES & INVOCATION · IAM PERMISSIONS · CONFIG & DEPLOYMENT · CLOUDWATCH & X-RAY

STUDY & REVIEW

01 MUST KNOW

if you remember five things, remember these

01 **Serverless, event-driven compute: AWS Lambda runs your code without provisioning or managing servers.** It invokes your code in response to **events** you configure, scales automatically with demand, has built-in fault tolerance, and provides built-in monitoring and logging with **Amazon CloudWatch**. You pay only for the compute time you use — nothing while the code isn't running.

02 **Functions + event sources:** the two core components. An **event source** is the entity that publishes events; a **Lambda function** is your custom code that processes them. Each source uses either a **push** model (a service invokes the function directly) or a **pull/polling** model (Lambda polls a stream or queue via an *event source mapping* and invokes with batches of records).

03 **Three invocation models: synchronous** — caller waits for the response, **no built-in retry** (a `200` ack); **asynchronous** — Lambda queues the event (a `202` ack), auto-retries **twice**, can send failures to a dead-letter queue; **polling** — event source mapping reads a stream or queue. For AWS-service triggers the invocation type is **predetermined** — you don't control it.

04 **Two kinds of permissions:** a **resource-based (function) policy** grants an event source the `lambda:InvokeFunction` action — who may *invoke* the function. An **execution role** (an IAM policy plus a *trust policy* that lets `lambda.amazonaws.com` assume it) grants what the function may *do* to other services. Polling sources also need the execution role to read the stream or queue.

05 **Configure, then deploy:** tune **memory** (128 MB–10,240 MB; CPU scales with it), **timeout** (1 s–15 min), and **concurrency** (1,000 per Region soft limit); use **provisioned concurrency** to avoid cold starts. Deploy a **.zip** archive or a **container image**. **Versions** are immutable ($LATEST is the mutable working copy); **aliases** are mutable pointers; **layers** (up to 5, 250 MB) share dependencies.

02 KEY TERMS

TERM	DEFINITION
Serverless computing	A model with no OS instances to manage, flexible scaling, pay-per-usage, built-in fault tolerance, and zero maintenance
AWS Lambda	The serverless compute service that runs code in response to events and scales automatically, with no servers to manage
Event source	The entity that publishes events to Lambda (an AWS service, an HTTP endpoint, a queue or stream, or a schedule)
Event source mapping	A Lambda resource that polls a stream or queue and invokes the function with batches of records (the pull model)
Concurrency	The number of function invocations running at one time; how Lambda scales horizontally on demand
Cold start	The startup latency incurred when Lambda must initialize a new environment; shown as the Init duration in logs
Execution role	The IAM role that grants a function permission to access AWS services and resources; Lambda assumes it via a trust policy
Resource-based policy	A policy on the function (a function policy) that grants event sources permission to take the lambda:InvokeFunction action
Function handler	The entry-point method Lambda calls to run your code; always receives an event object and a context object

TERM	DEFINITION
Context object	Runtime information for the handler — includes awsRequestId, logStreamName, and getRemainingTimeInMillis()
Alias	A mutable pointer to a specific function version; promote or roll back by repointing the alias, not by changing code
Lambda layer	A .zip archive of libraries, a custom runtime, or dependencies shared across functions (up to 5 layers, 250 MB extracted)

03 PUSH INVOCATION: SYNCHRONOUS VS. ASYNCHRONOUS

both are push events — the difference is who waits and what retries

SYNCHRONOUS

the caller waits for the response

- The service invokes the function directly and blocks for the result
- Acknowledgement is a `200` status
- **No built-in retry** — manage retries in your application code
- Sources: **API Gateway**, ELB, Cognito, Lex, Alexa, CloudFront, CloudFormation, Kinesis Data Firehose

ASYNCHRONOUS

send it and forget it

- Lambda **queues** the event, then invokes the function
- Acknowledgement is a `202` status (event received)
- Auto-retries **twice**; failures can go to a **dead-letter queue**
- Sources: **S3**, SNS, SES, CloudWatch Logs, CloudWatch Events, Config, CloudFormation

04 INVOCATION MODELS & EVENT SOURCES AT A GLANCE

match the model to the source and its failure behavior

MODEL	HOW LAMBDA HANDLES IT	RETRY / FAILURE HANDLING	EXAMPLE SOURCES
Push · sync	Service invokes directly and waits	**None built-in** — retry in your code	API Gateway, ELB, Lex, Alexa, CloudFront
Push · async	Lambda queues the event, returns `202`, then invokes	Retries **twice**; failures → dead-letter queue	S3, SNS, SES, Config, CloudWatch Events
Pull · stream	Mapping polls the stream; delivers records in **batches** by shard	Retries until success or **24-hour** expiry; a bad record can block the shard	DynamoDB Streams, Kinesis Data Streams
Pull · queue	Mapping polls the queue and invokes with records	Failed message returns to the queue, retries to a limit, then DLQ	Amazon SQS

05 LAMBDA EXECUTION FLOW — AUTHOR TO INVOCATION

you own the code and config; Lambda owns the environment

`01` Upload code or write it in the Lambda editor → `02` Set permissions — **execution role** + **resource policy** → `03` Choose a **runtime**; package code + dependencies → `04` Configure **memory, timeout, concurrency** → `05` An **event source** invokes the function → `06` Lambda runs your **handler** in an ephemeral environment; logs → CloudWatch, pay per use

06 CONFIGURATION KNOBS & LIMITS

the numbers the knowledge check likes to ask

SETTING	VALUE / LIMIT	NOTES
Memory	128 MB – 10,240 MB	CPU allocated **linearly** with memory; drives duration cost
Timeout	1 second – 15 minutes	The function is stopped if it runs past this
Concurrency (account)	1,000 per Region	**Soft** limit; reserved concurrency carves out a per-function slice
Provisioned concurrency	Set per function	Keeps environments **warm** to avoid cold starts; **priced separately**
Async invocation	0–2 retries · up to 6 h	Event held up to 6 hours; optional dead-letter queue

SETTING	VALUE / LIMIT	NOTES
.zip package	50 MB zipped · 250 MB unzipped	Console editor limited to 3 MB; 512 KB per individual file
Layers	Up to 5 · 250 MB	Function + all layers ≤ 250 MB extracted

07 EXAM TRAPS

where the knowledge check gets you

- × “Serverless means there are no servers.” — Servers still run your code; you simply don't **provision, patch, or manage** them. AWS handles the OS, scaling, capacity, and availability.
- × “You control the Lambda runtime environment.” — Environments are **ephemeral**, created on demand and torn down by Lambda; you have no direct control. Treat every function as **stateless** and reuse connections only opportunistically.
- × “Synchronous invocations get automatic retries.” — Only **asynchronous** invocations retry automatically (twice) and support a dead-letter queue. For synchronous calls you must handle retries in your own code.
- × “You pick sync or async when an AWS service triggers your function.” — For AWS-service event sources the invocation type is **predetermined**; you can't change it. You choose the type only when you call the invoke operation yourself.
- × “One IAM policy covers everything.” — Lambda needs **two** distinct permissions: a **resource-based policy** (who may invoke) and an **execution role** (what the function may do). Polling sources also rely on the execution role to read the stream or queue.
- × “Reserved and provisioned concurrency are the same thing.” — **Reserved** concurrency reserves/caps a slice of the Regional pool for a function (no extra charge); **provisioned** concurrency keeps environments warm to kill cold starts and is **priced separately**.
- × “Publishing a new version edits the existing one.” — Versions are **immutable** snapshots; only $LATEST changes. Point an **alias** (mutable) at a version so you can promote or roll back without touching event-source configuration.
- × “Add a missing dependency by editing code in the console or listing it in environment variables.” — The console editor is only for small code without custom libraries; package dependencies in the deployment **.zip/container**, or share them through a **layer**.

08 SELF-QUIZ

answers are upside-down below

Q1 What are the two core components of Lambda?

Q2 Name Lambda's three invocation models.

Q3 For an asynchronous invocation, how many times does Lambda automatically retry a failed event, and where can repeated failures go?

Q4 Which HTTP status acknowledges a synchronous invoke, and which acknowledges an asynchronous one?

Q5 Which permission controls what a Lambda function is allowed to do to other AWS services, and what two policies make it up?

Q6 Which permission grants an event source the lambda:InvokeFunction action so it can call the function?

Q7 What is the maximum time a Lambda function can run before it times out?

Q8 What two objects does a function handler always receive?

Q9 Which feature keeps environments warm to eliminate cold-start latency?

Q10 To add missing dependencies without bloating the deployment package, what do you create and attach (up to 5, 250 MB extracted)?

ANSWER KEY (rotate the page)

Q1 functions and event sources **Q2** synchronous, asynchronous, and polling (event source mapping) **Q3** twice; to a dead-letter queue (or an on-failure destination) **Q4** 200 for synchronous, 202 for asynchronous **Q5** the execution role — an IAM policy plus a trust policy **Q6** the resource-based (function) policy **Q7** 15 minutes **Q8** the event object and the context object **Q9** provisioned concurrency **Q10** a Lambda layer