Building Serverless Architectures and Microservices

THINKING SERVERLESS · MICROSERVICES · AWS LAMBDA · CONTAINERS · STEP FUNCTIONS · API GATEWAY · WELL-ARCHITECTED

STUDY & REVIEW

01 MUST KNOW

if you remember five things, remember these

01 **Serverless = no servers to manage:** A service is **serverless** when you deploy an application without thinking about servers or their sizing — AWS manages the runtime environment, scaling, and availability. Four hallmark benefits: **no server management**, **pay-for-value** (billed by usage, down to milliseconds), **continuous automatic scaling**, and **built-in high availability / fault tolerance**. Serverless is a natural fit for **event-driven and microservice** architectures.

02 **Lambda's hard limits define its lane: AWS Lambda** runs code functions without provisioning servers. A function has a **15-minute maximum** duration (an AWS hard limit), **128 MB–10,240 MB** of memory (memory also sets vCPU and network bandwidth), and an event-source-mapping payload capped at **6 MB**. Deploy it as a **.zip archive** or a **container image**; you pay only for the compute time consumed.

03 **Microservices are autonomous and specialized:** A **microservice** is **autonomous** (deployed and scaled independently, shares no code, talks only over well-defined APIs) and **specialized** (one business function, stateless, owns its own data store). On AWS the classic serverless pattern is **API Gateway** → **Lambda** → **DynamoDB**, which lives in the **app and data tiers** — a microservice is only part of a full architecture, not the whole thing.

04 **Pick containers when Lambda can't fit:** Move from Lambda to **containers** when a task runs **longer than 15 minutes**, needs **more than 10 GB of memory**, is a **legacy app** to migrate without refactoring, or runs **continuously** (fixed container cost beats per-invocation Lambda cost). AWS container stack: **ECR** (registry) · **ECS**/**EKS** (orchestration) · **EC2**, **Fargate**, or **Lambda** (compute). **Fargate** = serverless container nodes.

05 **Orchestrate with Step Functions, expose with API Gateway: AWS Step Functions** coordinates microservices as a **state machine** of event-driven states written in **Amazon States Language** (JSON) — managing state, checkpoints, retries, and restarts. **Standard** workflows are exactly-once / up to 1 year; **Express** are at-least/at-most-once / up to 5 min. **Amazon API Gateway** publishes **REST**, **HTTP**, and **WebSocket** APIs as the front door to those services.

02 KEY TERMS

TERM	DEFINITION
Serverless	A service you deploy without provisioning or sizing servers; AWS manages the runtime, scaling, and availability, and you pay for usage
AWS Lambda	Event-driven compute service that runs code functions without managing servers; 15-min max, 128 MB–10,240 MB memory, .zip or container image
Lambda@Edge	Lambda extension that runs Node.js or Python functions in Amazon CloudFront regional edge locations closer to the viewer to cut latency
Event source mapping	A Lambda resource that polls a stream or queue (DynamoDB, Kinesis, SQS, DocumentDB) and invokes a function with a batched payload (max 6 MB)
Microservice	An autonomous, specialized service that performs a single business function, is stateless, owns its data store, and communicates over lightweight APIs
Event-driven architecture (EDA)	Pattern of small, decoupled services that publish, consume, or route events — the messages passed between services
AWS Fargate	Serverless compute for containers — no cluster or server management, per-second billing — usable as Amazon ECS or Amazon EKS nodes

TERM	DEFINITION
Amazon ECS	AWS-opinionated container orchestration service; simplifies clusters with automated scaling; deploys tasks from JSON task definitions
Amazon EKS	Managed Kubernetes orchestration; deploys Pods via PodSpec, Deployment, and ReplicaSet; for teams that already know Kubernetes
AWS Step Functions	Serverless orchestrator that runs workflows (state machines) of event-driven states across multiple AWS services, with built-in retries
Amazon States Language	JSON-based language that defines a state machine — a StartAt field, a States map, and Next/End transitions
Amazon API Gateway	Managed service to create, publish, and secure REST, HTTP, and WebSocket APIs as the entry point to backend services

03 MONOLITHIC VS. MICROSERVICE APPLICATION

forum app: users · topics · messages

MONOLITHIC APPLICATION	MICROSERVICE APPLICATION
one tightly coupled deployment	*independent components, lightweight APIs*
— All processes run as a **single service**	— Each process runs as an **independent service**
— A spike in one process forces scaling the **entire** app	— Services **scale, deploy, and update** independently
— New features get harder as the code base grows	— One function can support **multiple** applications
— One process failure threatens **whole-app** availability	— Total failure **degrades** function, doesn't crash the app

04 AWS SERVERLESS SERVICES BY LAYER

serverless-first across the stack

LAYER	SERVICES	ROLE
Compute	Lambda, Lambda@Edge, Fargate	Run code or containers with no servers to manage
API publishing	API Gateway, AWS AppSync	REST/HTTP/WebSocket APIs; AppSync serves GraphQL
Messaging	Amazon SNS, Amazon SQS	SNS = pub/sub; SQS = queues that decouple components
Orchestration & events	Step Functions, EventBridge	Sequence services; event bus routes and checks schemas
Data stores	S3, EFS, DynamoDB, Aurora/Redshift/ Neptune/OpenSearch Serverless	Object, file, key-value, relational, graph, and search
Auth · hosting · CDN	Cognito, Amplify, CloudFront	User pools/JWT; build & host apps; deliver static content

05 REQUEST FLOW: SERVERLESS THREE-TIER WEB APP

static front end + authorized API

01 Browser client requests the app → 02 **CloudFront** serves the static front end from an **S3** bucket → 03 **Cognito** user pool issues a **JWT** for the signed-in user → 04 **API Gateway** receives the API call and validates the JWT → 05 **Lambda** runs the business logic → 06 **DynamoDB** returns the application data

06 API GATEWAY: CHOOSING AN API TYPE

REST vs. HTTP vs. WebSocket

API TYPE	BEST FOR	TRAITS
REST	Full API management & control	Usage plans, payload validation, private endpoints, resource policies; CORS; **stateless**
HTTP	Microservices (Lambda/HTTP backends)	**Lower latency & lower cost** than REST; CORS; stateless
WebSocket	Real-time applications	Persistent **session** between client and backend; **stateful**

07 EXAM TRAPS

where the knowledge check gets you

- × "Serverless means there are no servers." — Servers still run your code; **AWS** provisions, sizes, patches, and scales them. Serverless means *you* never manage or size a server — not that none exists.
- × "A Lambda function can run as long as it needs to." — A function times out at a **hard 15-minute maximum**; workloads over 15 min or over **10 GB** of memory belong in **containers**.
- × "A microservice is a complete three-tier application." — Microservices operate only in the **app and data tiers**; they are one part of a larger architecture, not a standalone full solution.
- × "Lambda can reach resources in your VPC by default." — By default it cannot. You attach the function to your VPC via managed **Hyperplane ENIs** (VPC-to-VPC NAT); once attached it has **no internet** access unless you route out through a **NAT gateway**.
- × "Express Workflows run each step exactly once." — Only **Standard** workflows are exactly-once. **Express** is **at-least-once** (synchronous) or **at-most-once** (asynchronous), so a step can run more than once or not at all.
- × "Use REST APIs for everything." — For microservices, **HTTP APIs** give lower latency and lower cost; use **WebSocket APIs** for stateful, real-time apps. Reserve **REST** for when you need full API-management features.
- × "ECS and EKS both autoscale out of the box." — **ECS** provides automated scaling based on demand; **EKS** requires **manual** configuration of autoscaling groups — more control, more setup.
- × "A Lambda function URL and API Gateway are interchangeable." — A **function URL** endpoint is fixed, but changing it forces a **browser-app update**; behind **API Gateway**, the function can change with **no** client change.

08 SELF-QUIZ

answers are upside-down below

Q1 What is the maximum duration of a single AWS Lambda function invocation?

Q2 What are the minimum and maximum memory settings for a Lambda function?

Q3 What is the maximum payload size an event source mapping delivers to a Lambda function?

Q4 Name the two Lambda deployment package types.

Q5 Which three AWS services combine into the classic serverless microservice pattern (API, compute, data)?

Q6 Which Step Functions workflow type is exactly-once and can run up to one year?

Q7 Give two conditions that make a workload a better fit for containers than for Lambda.

Q8 Which API Gateway API type is stateful and built for real-time applications?

Q9 Which AWS container orchestration service uses the Kubernetes toolset?

Q10 In what language is a Step Functions state machine defined?

ANSWER KEY (rotate the page)

Q1 15 minutes (an AWS hard limit) **Q2** 128 MB and 10,240 MB **Q3** 6 MB **Q4** .zip file archive and container image **Q5** Amazon API Gateway, AWS Lambda, and Amazon DynamoDB **Q6** Standard Workflows **Q7** Runs longer than 15 minutes, or needs more than 10 GB of memory (also legacy migration or continuous fixed-cost workloads) **Q8** WebSocket APIs **Q9** Amazon EKS **Q10** Amazon States Language (JSON-based)