



# AWS Serverless

Verified 2026-10-09

High-yield reference for AWS Lambda, API Gateway, EventBridge, DynamoDB, Step Functions, and serverless architecture patterns.

---

## Contents

|                                       |    |
|---------------------------------------|----|
| Mental model .....                    | 2  |
| Core compute and routing .....        | 2  |
| Data and storage .....                | 4  |
| Event integration and messaging ..... | 5  |
| Orchestration and security .....      | 7  |
| Deployment and IaC .....              | 8  |
| Further reading .....                 | 10 |

# Mental model

AWS Serverless is an architectural paradigm where cloud infrastructure management is fully offloaded to AWS. Applications are decomposed into event-driven, stateless components that execute on demand, auto-scale from zero to peak demand, and bill strictly for consumed resources.

## Core compute and routing

| Component     | Responsibility                          |
|---------------|-----------------------------------------|
| AWS Lambda    | Stateless execution triggered by events |
| API Gateway   | REST & HTTP endpoints, routing, auth    |
| Lambda Layers | Shared code libraries and binaries      |

AWS Lambda executes code in isolated environments. Warm starts reuse existing execution contexts, while cold starts initialize new instances. API Gateway maps incoming HTTP requests directly to Lambda event

payloads and converts return objects into HTTP responses.

```
export async function handler(  
  event: { body?: string }  
) {  
  const data = JSON.parse(  
    event.body ?? "{}"  
  );  
  return {  
    statusCode: 200,  
    body: JSON.stringify({  
      message: "Success",  
      data,  
    }),  
  };  
}
```

**Gotcha:** Global variables persist across invocations during warm starts. Never store request-specific state in global variables.

## Data and storage

| Datastore   | Best for                                |
|-------------|-----------------------------------------|
| DynamoDB    | Key-value & document NoSQL datastore    |
| Amazon S3   | Scalable object storage for files/blobs |
| ElastiCache | In-memory caching for low latency       |

DynamoDB provides single-digit millisecond performance at scale. Single-table design uses composite primary keys ( Partition Key + Sort Key ) to store multiple entity types in one table, querying related data in a single request without costly joins.

```
type Item = { pk: string; sk: string };  
  
function buildParams(item: Item) {
```

```
return {  
    TableName: "MainTable",  
    Item: {  
        PK: { S: item.pk },  
        SK: { S: item.sk },  
    },  
};  
}
```

**Warning:** DynamoDB scans inspect the entire table and consume high capacity. Always prefer queries on primary keys or Secondary Indexes.

## Event integration and messaging

| Service     | Delivery model                          |
|-------------|-----------------------------------------|
| EventBridge | Event bus for decoupling services       |
| SQS         | Queueing for asynchronous processing    |
| SNS         | Pub/Sub topic for multi-receiver fanout |

Serverless systems use asynchronous messaging to decouple producers and consumers. EventBridge filters and routes domain events across microservices. SQS queues handle backpressure and retry buffering, while SNS broadcasts messages to multiple subscribers simultaneously.

```
{
  "source": "app.orders",
  "detail-type": "OrderCreated",
  "detail": {
    "orderId": "ord-9921",
    "amount": 49.99
  }
}
```

**Tip:** Pair SQS FIFO queues with Lambda when strict FIFO ordering and message deduplication are required.

## Orchestration and security

| Tool               | Purpose                                |
|--------------------|----------------------------------------|
| Step Functions     | State machine workflow orchestration   |
| IAM Execution Role | Grants permissions to Lambda functions |
| Resource Policy    | Controls access to API Gateway/SQS     |

AWS Step Functions coordinates multi-step workflows with built-in retries, branching logic, and error handling without custom orchestrator code. Security operates under the principle of least privilege using IAM execution roles assigned directly to compute resources.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",
```

```
"Action": [
    "dynamodb:GetItem",
    "dynamodb:PutItem"
],
"Resource": "arn:aws:dynamodb:*:*:table/
Data"
}
]
```

**Gotcha:** Granting wildcard permissions ( \* ) in IAM execution roles exposes the serverless resource to unauthorized escalation.

## Deployment and IaC

| Framework | Focus                                    |
|-----------|------------------------------------------|
| AWS SAM   | Open-source framework for serverless IaC |

| Framework            | Focus                                    |
|----------------------|------------------------------------------|
| AWS CDK              | Define cloud infrastructure in TS/Python |
| Serverless Framework | Multi-cloud serverless deployment        |

Infrastructure-as-Code (IaC) allows serverless stacks to be defined declaratively and versioned in source control. AWS SAM extends CloudFormation with concise syntax for functions, APIs, and databases. AWS CDK enables constructing serverless stacks using programming languages.

```
AWSTemplateFormatVersion: '2010-09-09'
```

```
Transform: AWS::Serverless-2016-10-31
```

```
Resources:
```

```
  GetOrderFunction:
```

```
    Type: AWS::Serverless::Function
```

```
    Properties:
```

```
      CodeUri: src/
```

**Handler:** app.handler

**Runtime:** nodejs22.x

**Events:**

**Api:**

**Type:** Api

**Properties:**

**Path:** /orders/{id}

**Method:** get

**Tip:** Use SAM local ( `sam local invoke` ) to test Lambda functions locally against simulated API Gateway events before deploying.

## Further reading

- [AWS Serverless Application Model \(SAM\) Guide](#)
- [AWS Lambda Developer Documentation](#)
- [Amazon DynamoDB Developer Guide](#)
- [Amazon EventBridge User Guide](#)