

Serverless Computing with AWS Lambda, API Gateway & Step Functions

Welcome to this comprehensive guide on building serverless applications using AWS's powerful suite of services. We'll explore how Lambda functions, API Gateway, and Step Functions work together to create scalable, cost-effective solutions without server management.

Welcome & Agenda

What is Serverless Computing?

"Serverless" doesn't mean "no servers" – it means **no server management**. You focus on code while AWS handles infrastructure.

Why Serverless?

Cost efficiency, automatic scaling, and event-driven architecture make serverless ideal for modern applications.

Today's Topics

1. AWS Lambda - Triggers & Permissions
2. API Gateway Integration
3. Step Functions (Intro)

Serverless Architecture Overview

Event-driven

Functions execute in response to events, not continuously running servers.

Managed Scaling

AWS automatically scales resources to match workload demands.

Pay-per-use

Only pay for compute time consumed, not idle capacity.

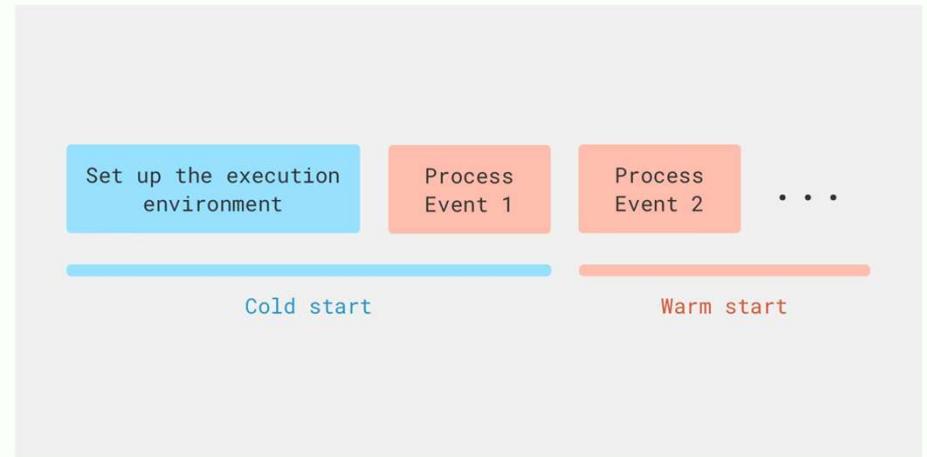
AWS Lambda Basics

What is AWS Lambda?

A serverless compute service that runs your code in response to events without provisioning or managing servers.

Supported Languages

- Node.js, Python, Java
- Go, .NET, Ruby
- Custom runtimes via containers



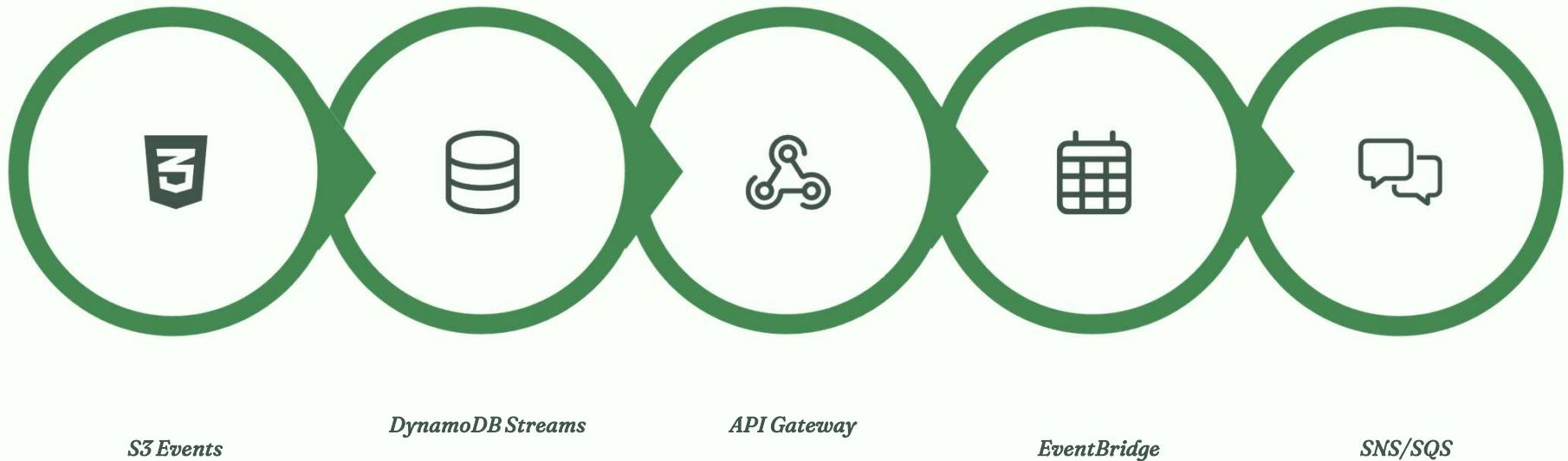
Key Concepts

- **Stateless functions** - don't assume state persists
- **Memory limits:** 128MB to 10GB
- **Timeout:** up to 15 minutes
- **Pricing:** pay per request and compute time

Lambda Triggers

What is a Trigger?

An event source that invokes your Lambda function when specific conditions occur.



Synchronous Invocation

Caller waits for function to complete and return response (API Gateway, ALB)

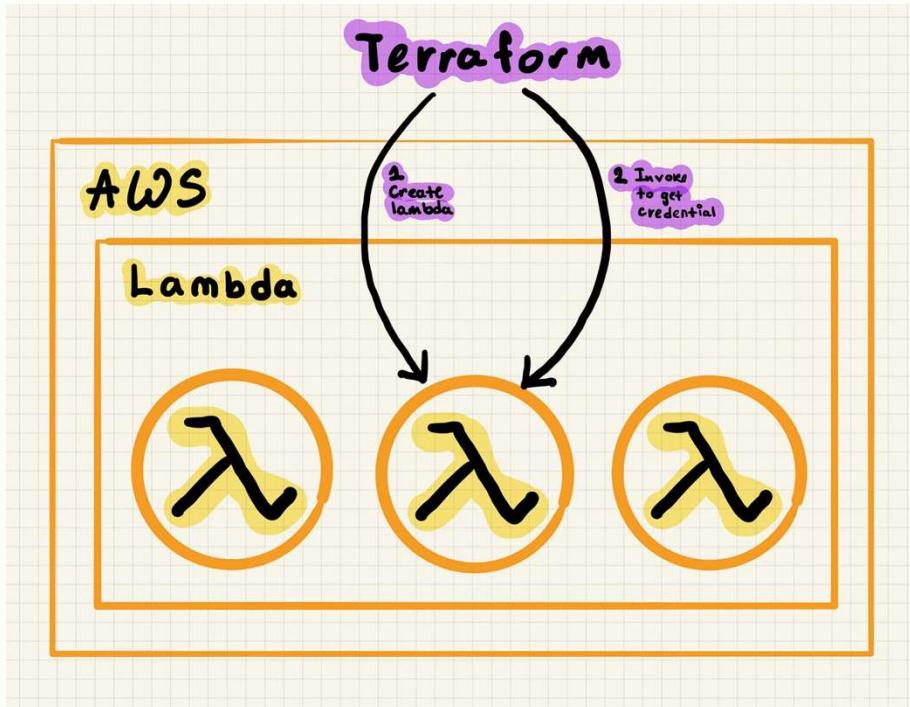
Asynchronous Invocation

Event placed in queue, response sent immediately (S3, SNS)

Lambda Permissions

Execution Role (IAM)

Defines what **Lambda** can access - AWS services and resources your function needs to use.



Resource-based Policies

Define **who can invoke Lambda** - which services or accounts can trigger your function.

```
{  
  "Effect": "Allow",  
  "Principal": { "Service": "s3.amazonaws.com" },  
  "Action": "lambda:InvokeFunction",  
  "Resource":  
    "arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME",  
  "Condition": {  
    "StringEquals": {  
      "AWS:SourceAccount": "123456789012"  
    },  
    "ArnLike": {  
      "AWS:SourceArn": "arn:aws:s3:::my-source-bucket-  
demo-2025"  
    }  
  }  
}
```

API Gateway + Lambda Integration

Lambda Proxy Integration

Passes entire request to Lambda (headers, query strings, path parameters). Lambda returns full response object.

Lambda Non-proxy Integration

Uses mapping templates to transform request/response. More control but more complex.

API Gateway handles authentication, throttling, and caching while Lambda focuses on business logic.

Demo Flow Diagram



This flow demonstrates a complete serverless application architecture. Notice how triggers connect components and permissions control access at each step.

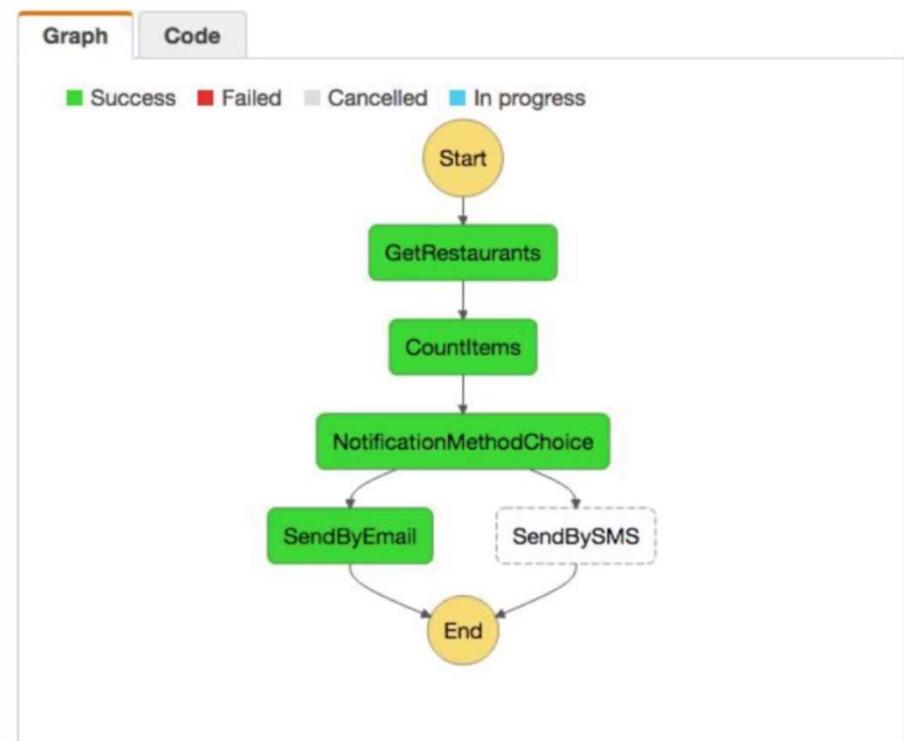
AWS Step Functions Intro

What are Step Functions?

A serverless orchestration service that lets you coordinate multiple AWS services into workflows.

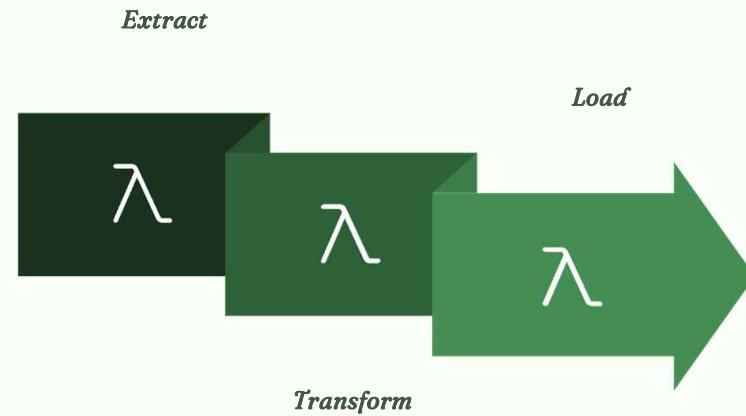
State Machine Basics

- **States:** Task, Choice, Parallel, Map, Wait, etc.
- **Transitions:** Define flow between states
- **Parallelism:** Execute branches simultaneously



The AWS Console provides a visual workflow builder that generates the underlying JSON/YAML definition.

Step Functions + Lambda



Step Functions excel at orchestrating complex workflows across multiple Lambda functions, providing:

- Built-in error handling and retry logic
- Visual monitoring of workflow execution
- Long-running processes beyond Lambda's 15-minute limit
- Conditional branching based on function outputs

Cost & Best Practices

Cost Optimization Tips

- Minimize function memory & runtime
- Implement batch processing where possible
- Use Step Functions for orchestration instead of chaining Lambdas
- Optimize cold starts with provisioned concurrency

Best Practices

- Apply least privilege IAM roles
- Use environment variables for configuration
- Implement comprehensive error handling
- Keep functions focused on single responsibilities
- Consider SQS for handling traffic spikes

 Remember: Serverless isn't always cheaper, but it often provides better value through reduced operational overhead and improved developer productivity.