Great question 👌 — let’s step back and talk **real-world architecture** for a project where you already have **Spring Boot services** and want to use **Lambda, Step Functions, and ALB / API Gateway**.

## 🔹 Two Common Real-World Patterns

### ****1. API Gateway (or ALB) → Lambda (Spring Boot adapter) → Step Functions****

Best when you want **external clients (mobile/web)** to hit your system via **API Gateway** (or ALB if you prefer).

The Lambda acts as the **entrypoint**: parses the request, starts the Step Function workflow, returns an immediate response (executionArn or “job accepted”).

**Step Functions** then orchestrate business Lambdas or even call ECS/EKS microservices.

✅ Used in:

Booking systems (flight → booking → payment).

Order systems (cart → order → payment → shipment).

### ****2. Direct Lambda Microservices with ALB (no Step Functions upfront)****

ALB routes directly to multiple Lambdas (/flight, /booking, /payment).

Each Lambda does synchronous work and responds to client.

**Step Functions** are used internally for **asynchronous flows** like payment settlement, fraud check, notifications.

✅ Used in:

Event-driven systems where **user only needs a quick response** but backend processing continues.

Payment gateways, ticket booking.

## 🔹 Example: ****Flight Booking Application****

Let’s imagine your company is building a **travel booking platform** with **Spring Boot services**:

### Services

**Flight Service** (search flights)

**Booking Service** (reserve a seat)

**Payment Service** (handle payment)

**Notification Service** (send confirmation email/SMS)

### ****Workflow (Step Functions)****

User hits /bookings via **API Gateway** or **ALB**.

Request goes to **Edge Lambda (Spring Boot Function)**.

Edge Lambda starts **Step Function workflow**:

Step 1 → Invoke Flight Lambda (Spring Boot microservice packaged for Lambda).

Step 2 → Invoke Booking Lambda.

Step 3 → Invoke Payment Lambda.

Step 4 → Invoke Notification Lambda.

### ****Architecture Diagram (Real-World)****

[ Client (Web/Mobile) ]

|

HTTPS

|

+----------------+

| API Gateway | (or ALB with Path routing)

+----------------+

|

v

[ Edge Lambda (Spring Boot)]

|

calls StartExecution

|

+------------------------+

| Step Functions |

| Flight → Booking → |

| Payment → Notify |

+------------------------+

| | |

v v v

Flight Lambda Booking Lambda Payment Lambda

(Spring Boot) (Spring Boot) (Spring Boot)

## 🔹 How It Looks in Code (Spring Boot + Spring Cloud Function)

### Edge Lambda (starts workflow)

@Bean

public Function<Map<String, Object>, Map<String, Object>> startWorkflow(SfnClient sfn) {

return input -> {

StartExecutionResponse resp = sfn.startExecution(

StartExecutionRequest.builder()

.stateMachineArn(System.getenv("STATE\_MACHINE\_ARN"))

.input(new ObjectMapper().writeValueAsString(input))

.build()

);

return Map.of("message", "Workflow started", "executionArn", resp.executionArn());

};

}

### Flight Lambda

@Bean

public Function<Map<String,Object>, Map<String,Object>> flight() {

return input -> Map.of(

"status", "ok",

"flightId", "FL-" + UUID.randomUUID()

);

}

(similar for Booking, Payment, Notification)

## 🔹 Terraform Project Structure

A **real project repo** will usually look like this:

infra/

├── main.tf

├── variables.tf

├── stepfunctions.tf

├── lambdas.tf

├── alb.tf

└── api\_gateway.tf (optional if using ALB)

services/

├── edge-service/ (Spring Boot Edge Lambda)

├── flight-service/ (Spring Boot Lambda)

├── booking-service/ (Spring Boot Lambda)

├── payment-service/ (Spring Boot Lambda)

└── notification-service/ (Spring Boot Lambda)

**services/** → Each Spring Boot microservice, packaged as a Lambda JAR.

**infra/** → Terraform to provision ALB/API Gateway, Lambdas, IAM roles, Step Functions.

## 🔹 Deployment Flow (real project in CI/CD)

**Build Services**

./mvnw clean package

→ produces target/\*.jar for each service.

**Upload JARs to S3** (for Lambda code).

filename = "build/libs/edge-service.jar"

s3\_bucket = "my-lambda-artifacts"

s3\_key = "edge-service.jar"

**Terraform Apply**

Creates Lambdas.

Creates ALB/API Gateway.

Creates Step Functions workflow.

**Test**

Call https://my-api/bookings → triggers Step Functions → orchestrates Lambdas.

## 🔹 Rule of Thumb: ALB vs API Gateway

**API Gateway** → Better for **public APIs**, request validation, auth (Cognito, JWT), throttling.

**ALB** → Better for **internal services** or if you’re already running containers (ECS/Fargate).

Both can forward traffic to **Lambda**.

👉 In real projects, we often use:

**API Gateway + Step Functions + Lambda** for external-facing APIs (secure, scalable).

**ALB + ECS/Fargate** for heavier Spring Boot apps that don’t fit well in Lambda.

**Mix**: Some parts in Lambda/Step Functions (async workflows), some in ECS (long-running services).

Would you like me to **draw a full architecture diagram** (with ALB, API Gateway, Step Functions, and Spring Boot Lambdas all in one picture) so you can see how real projects mix them?