An Architectural Blueprint of the AWS Serverless Backend

A technical deep-dive into the compute, data, and networking layers.

AWS Lambda

Amazon DynamoDB

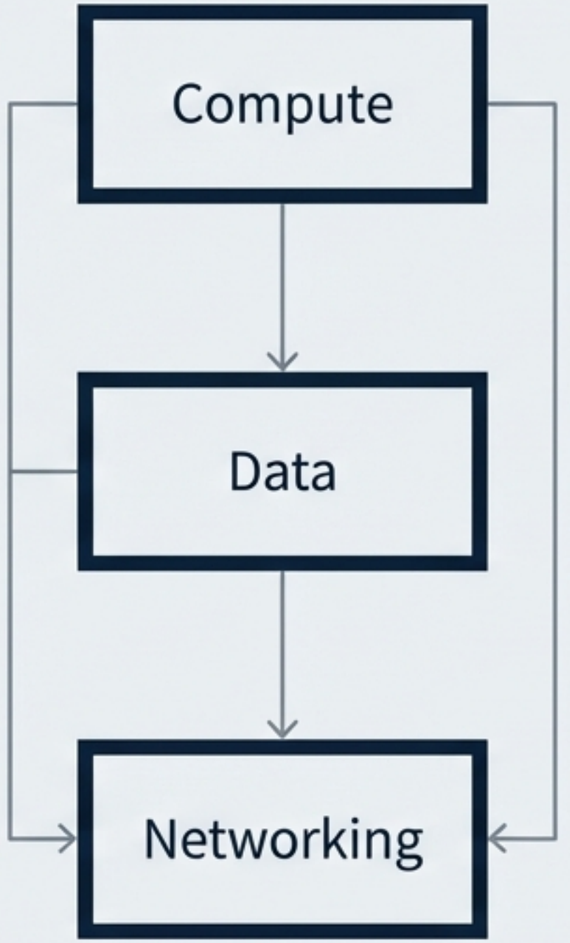
Amazon API Gateway

NotebookLM

Engineering a Scalable, Event-Driven Core

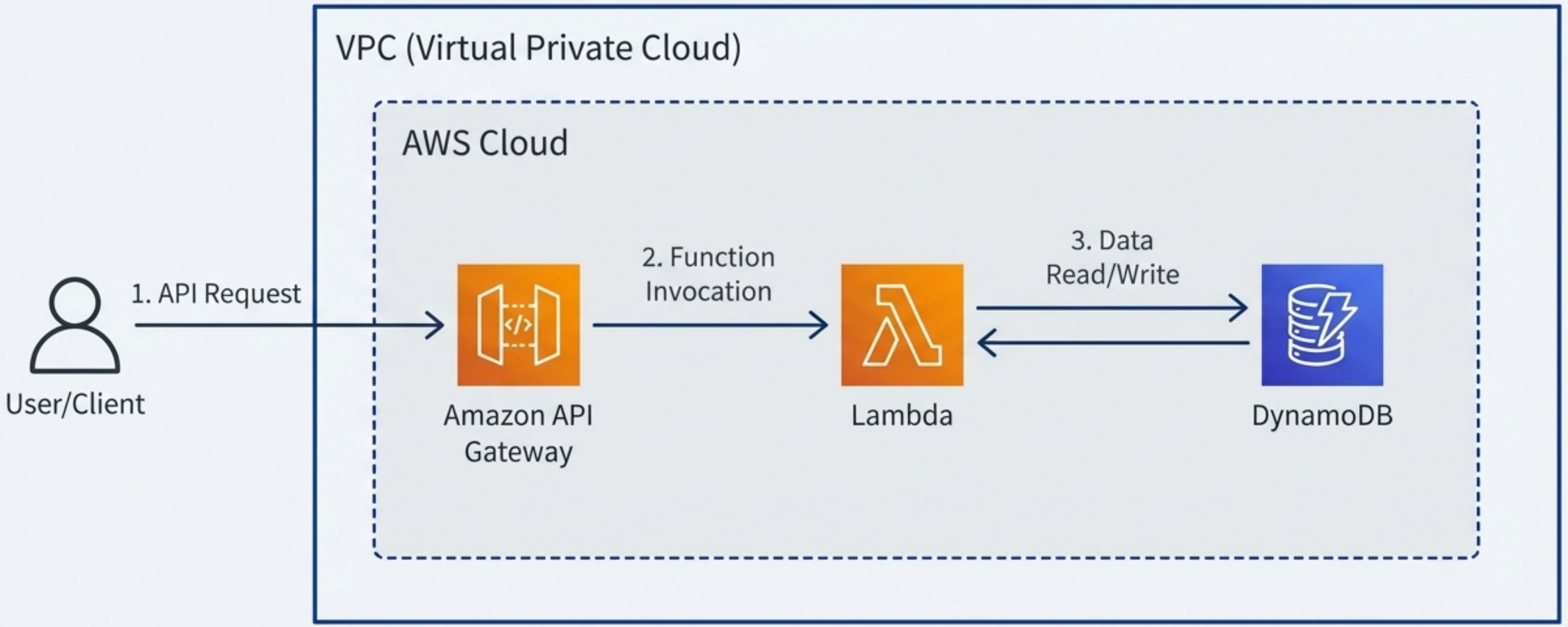
The primary objective was to build a backend that is highly scalable, resilient, and cost-efficient. Our serverless-first approach using core AWS services allows us to meet these requirements. This document details the specific implementation of our three foundational pillars:

- **Compute**: AWS Lambda for event-driven processing.
- **Data**: Amazon DynamoDB with a Single Table Design for high-performance access.
- **Networking**: A secure, private network configuration within the AWS Academy environment.



NotebookLM

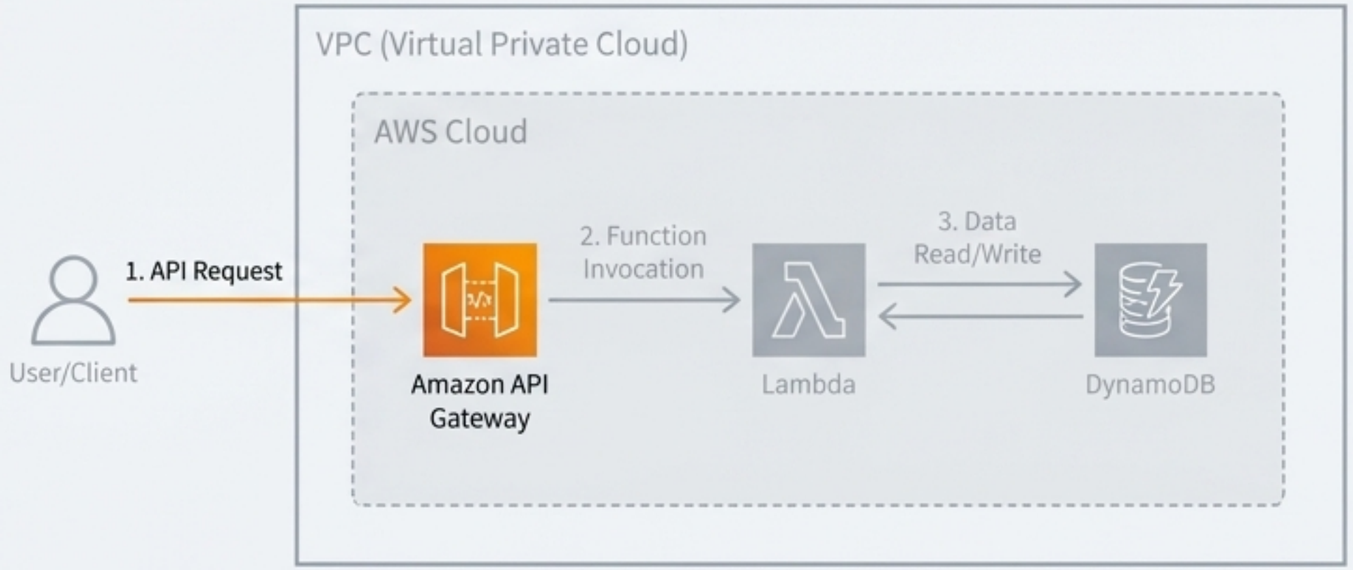
The Complete System Architecture at a Glance



NotebookLM

The Front Door: Managing Requests with API Gateway

- **Role**: Serves as the single, managed entry point for all incoming HTTP requests.
- **Implementation**: Utilizes an HTTP API for lower latency and cost-effectiveness compared to a REST API.
- **Integration**: Configured with a direct Lambda proxy integration, passing the entire request payload to the compute layer for processing.
- **Functionality**: Handles request validation, routing, and throttling, offloading these concerns from the application logic.

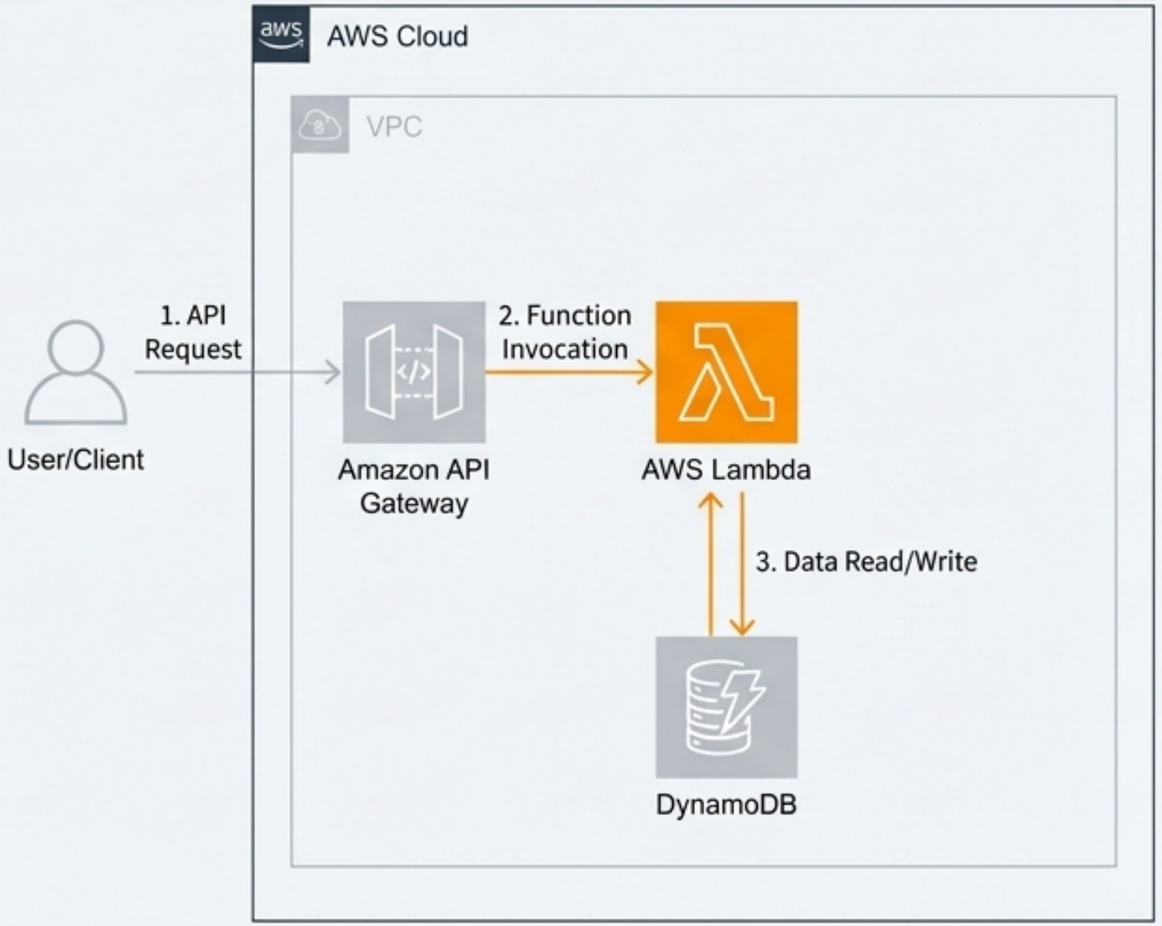


NotebookLM

The Brains of the Operation: AWS Lambda

At the heart of our backend is a single, multi-purpose Lambda function. We chose Lambda for its key serverless advantages:

- **Automatic Scaling**: Scales from zero to thousands of requests per second without manual intervention.
- **Event-Driven**: Natively integrates with API Gateway, executing code only in response to incoming requests.
- **No Server Management**: Eliminates the operational overhead of provisioning or managing servers.

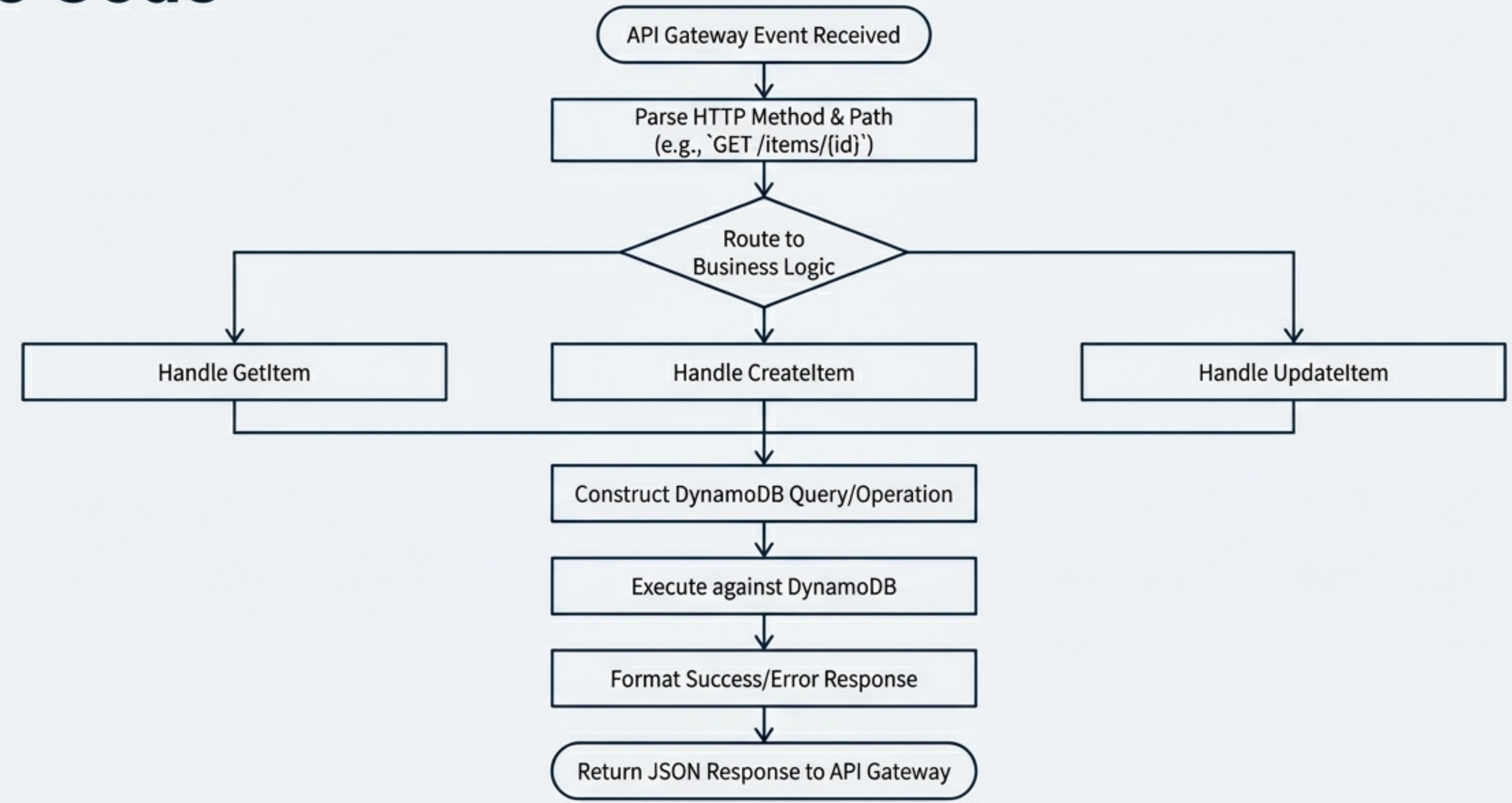


Key Configuration

Runtime: Node.js 18.x
Memory: 256 MB
Timeout: 15 seconds
IAM Role: lambda-dynamodb-execution-role (Provides least-privilege access to the DynamoDB table).

NotebookLM

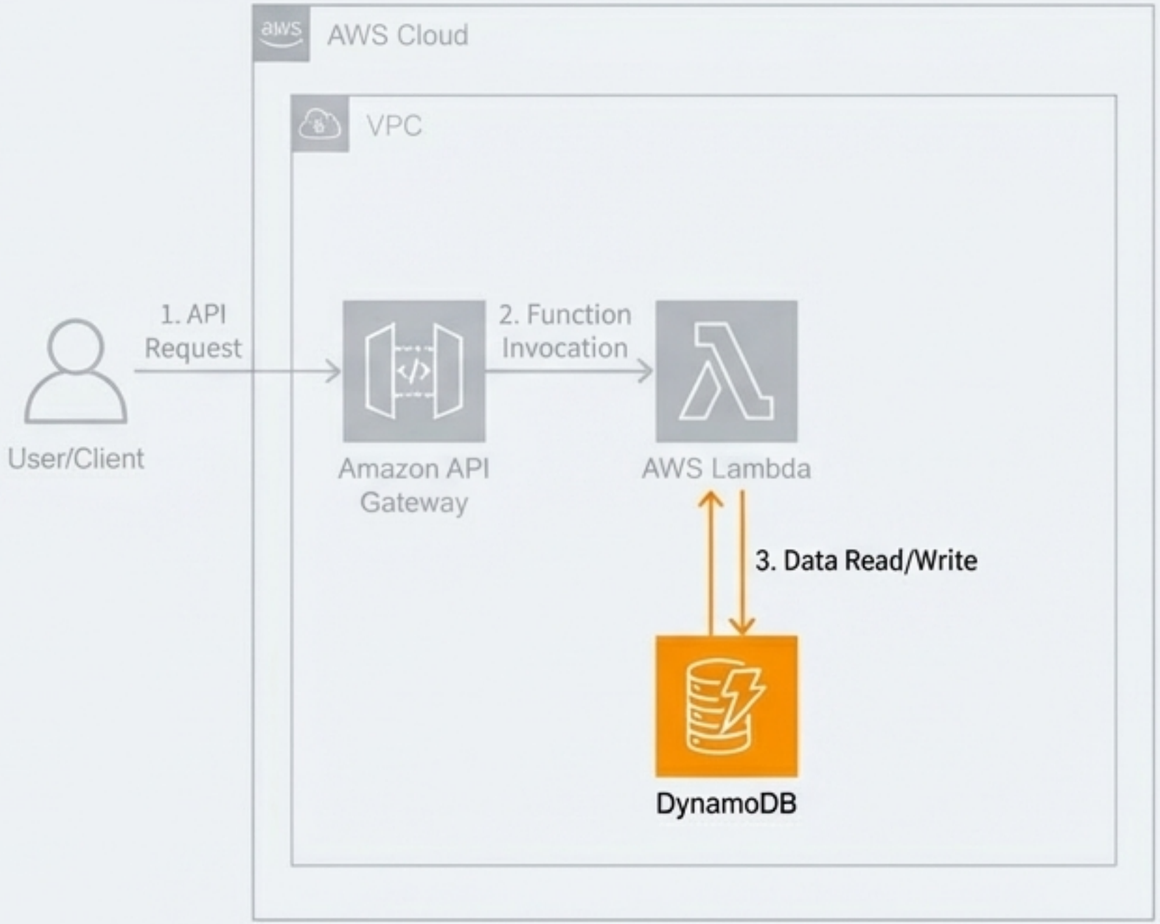
Inside the Function: A Request's Path Through the Code



NotebookLM

The System's Memory: High-Performance Data Storage with DynamoDB

- **Why DynamoDB?**: Chosen for its single-digit millisecond latency at any scale, inherent scalability, and serverless pricing model (On-Demand Capacity).
- **Data Model:** A NoSQL key-value and document store, which provides flexibility as our application's data requirements evolve.
- **Core Strategy:** We employ a Single Table Design to optimize for our application's specific access patterns and maximize query performance. This is a critical architectural decision we will explore next.

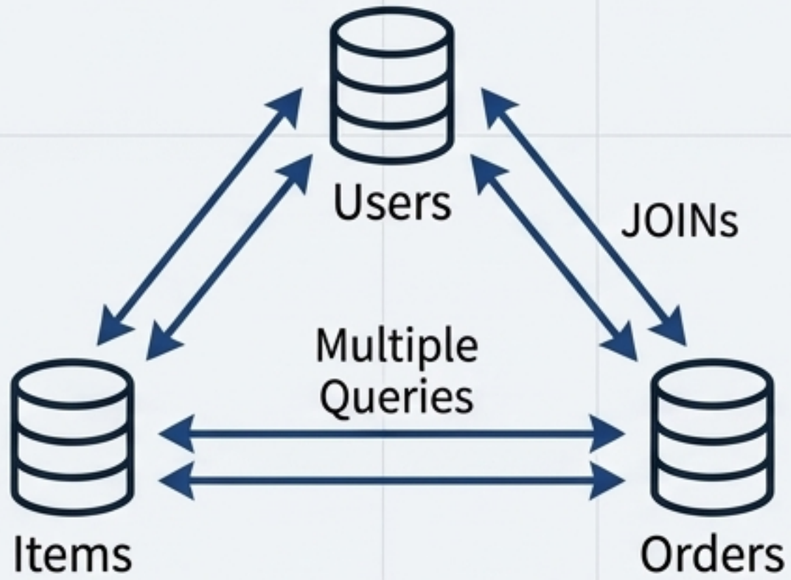


NotebookLM

The Power of Single Table Design

Instead of creating multiple tables (e.g., `Users`, `Items`, `Orders`), we store all entity types within a single DynamoDB table. While this requires more upfront data modeling, the performance gains are significant.

Relational / Multi-Table Model



Single Table Design

Key Benefit: This design minimizes the number of round trips to the database, allowing us to fetch complex, hierarchical data in a single, highly performant API call.

NotebookLM

Visualizing Our Data Model in a Single Table

Sort Key: Uniquely identifies and organizes different items within the user's partition.

Partition Key: Groups all data for a single user together.

PK	SK	EntityType	email / task	status
USER#1234	METADATA	User	jane.doe@example.com	N/A
USER#1234	ITEM#a1b2	Item	Finalize Q3 report	IN_PROGRESS
USER#1234	ITEM#c3d4	Item	Prepare deck for review	TODO
USER#5678	METADATA	User	john.smith@example.com	N/A
USER#5678	ITEM#e5f6	Item	Onboard new engineer	DONE

NotebookLM

Serving Key Access Patterns with Precision

The PK/SK structure is designed to answer our application's most common questions with a single query.

The Requirement	The DynamoDB Query
• 1. Fetch a user's profile.	1. **Query:** `PK = "USER#1234" and SK = "METADATA"`
• 2. Fetch all to-do items for a specific user.	2. **Query:** `PK = "USER#1234" and SK begins with "ITEM#"`
• 3. Fetch a single, specific to-do item.	3. **GetItem:** `PK = "USER#1234" and SK = "ITEM#a1b2"`

NotebookLM

Key Architectural Decisions: The Data Layer

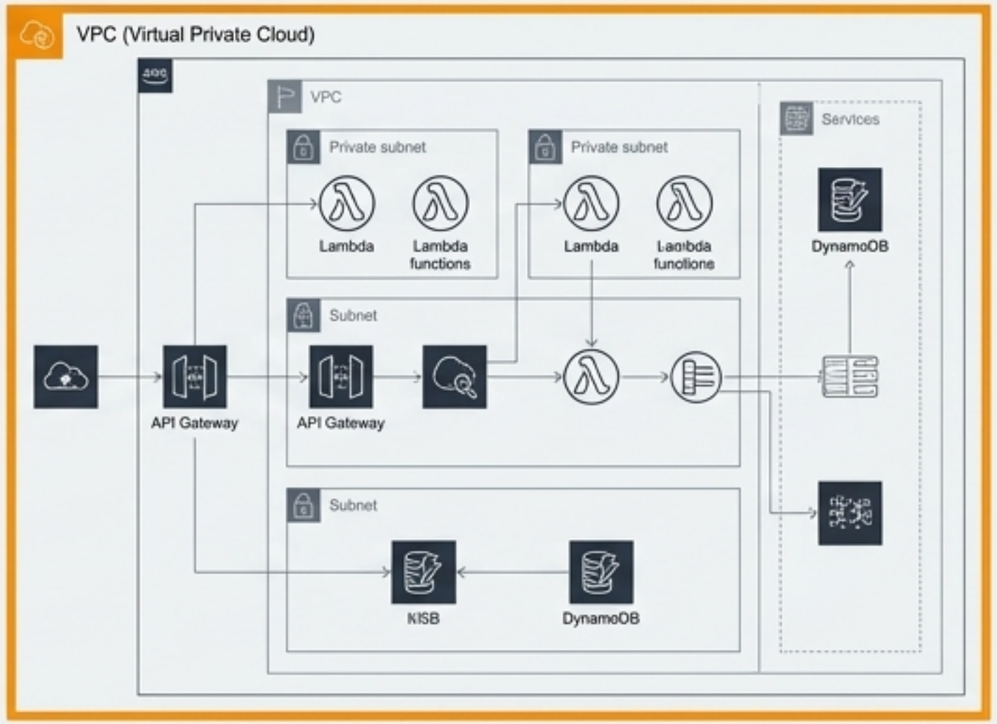
- **Single Table Design for Performance**: Prioritized minimizing query latency over data model simplicity by co-locating related data.
- **User-Centric Partitioning**: The USER#<id> partition key ensures all data for a given user is stored together, optimizing for the most common access patterns.
- **On-Demand Capacity Mode**: Chosen to handle unpredictable traffic patterns without provisioning capacity, aligning with our serverless, pay-for-value philosophy.
- **Flexible Schema**: The NoSQL model allows for the addition of new attributes or even new entity types in the future with minimal friction.

NotebookLM

Securing the Perimeter: VPC and Private Networking

The entire backend operates within a Virtual Private Cloud (VPC), isolating it from the public internet. This is a critical security requirement, especially given the constraints of the AWS Academy environment.

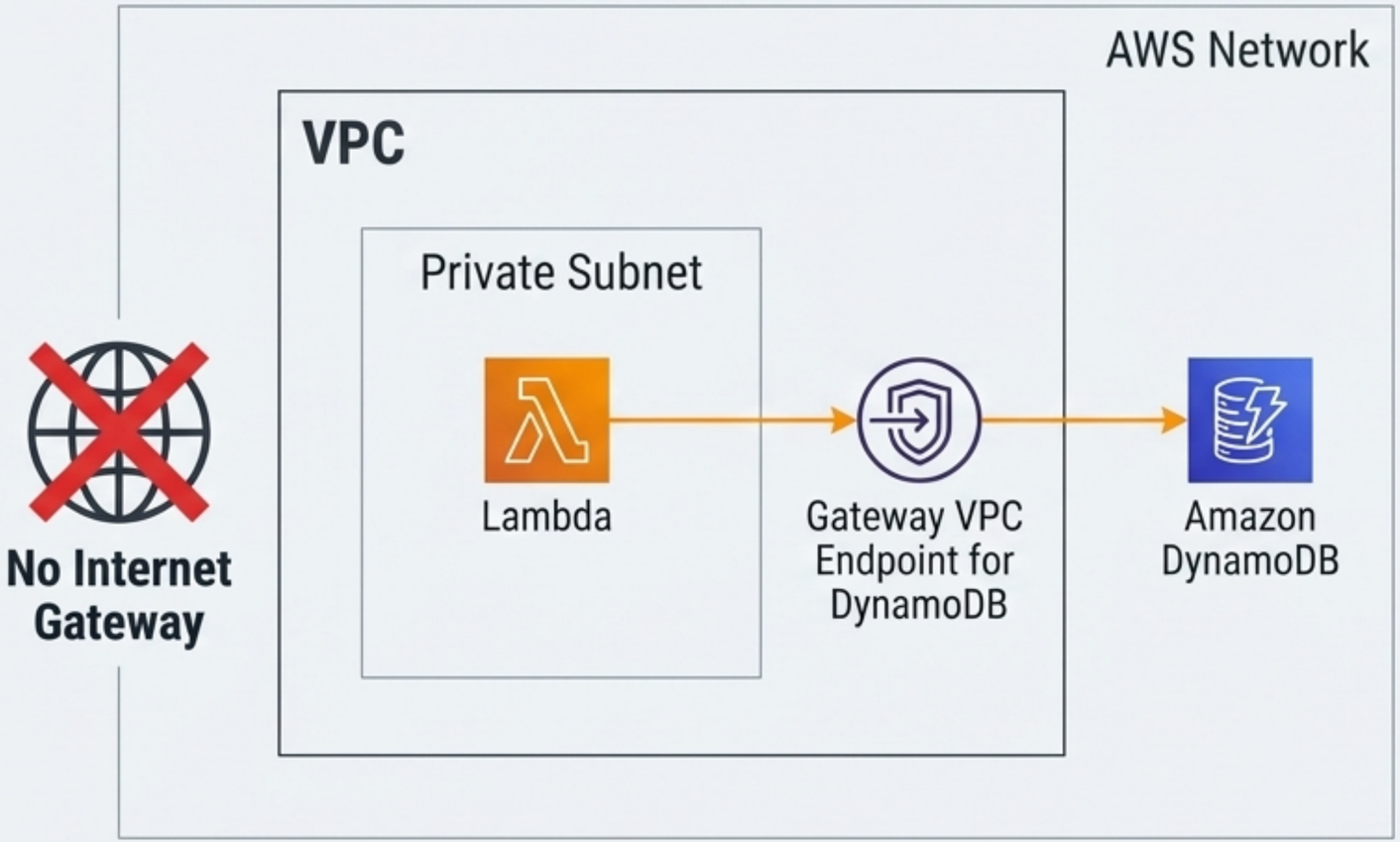
- **Isolation**: Lambda functions are deployed into private subnets with no direct outbound internet access.
- **Controlled Access**: All communication between services happens over the private AWS network, not the public internet.
- **Security by Default**: This configuration significantly reduces the attack surface and prevents accidental data exposure.



NotebookLM

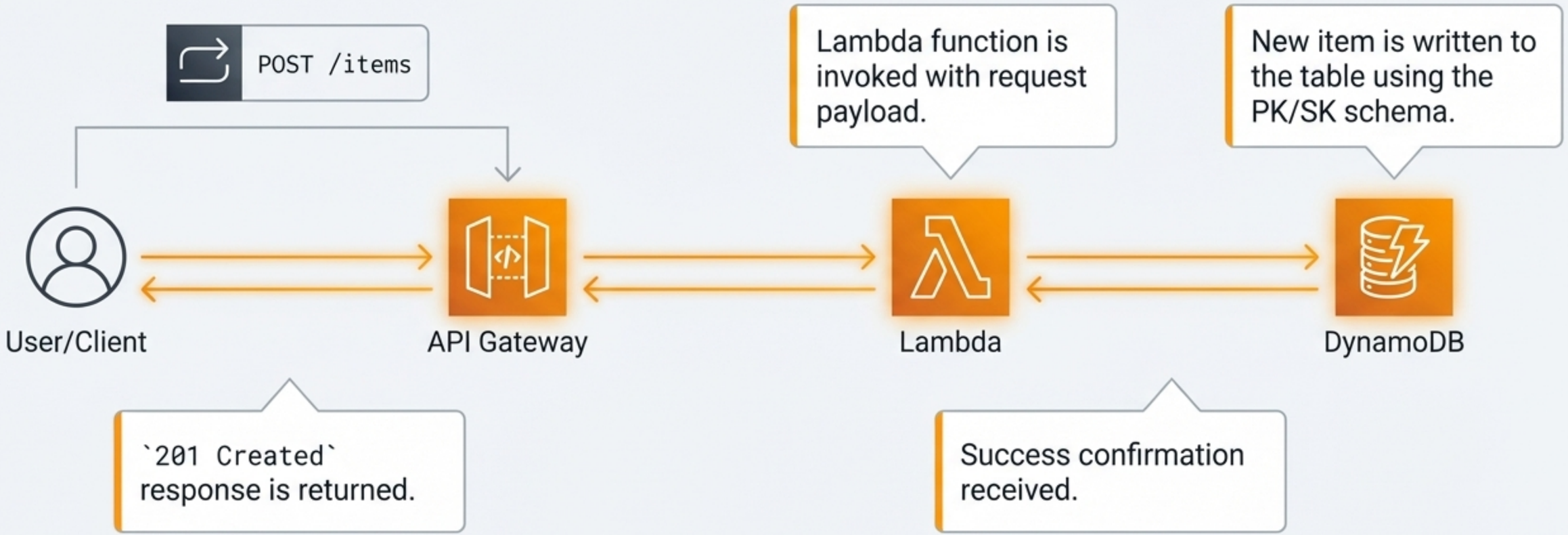
Private & Secure Communication via VPC Endpoints

To allow our Lambda function in a private subnet to access DynamoDB (which is a public AWS service), we use a **Gateway VPC Endpoint.**
This routes traffic directly to DynamoDB over the private AWS network, ensuring data never traverses the public internet.
This is both more secure and often more performant.



NotebookLM

Putting It All Together: The Full Journey of a Single Request



This seamless, secure, and scalable flow is the foundation of our serverless backend.

NotebookLM

An Architecture Built for the Future

This serverless architecture provides a powerful foundation that is not only efficient and scalable today but is also designed for future evolution.

Performance

Single Table Design and direct service integrations minimize latency at every step.

Scalability

By leveraging managed AWS services, the system scales automatically with user demand without manual intervention.

Operational Excellence

The serverless model drastically reduces management overhead, allowing developers to focus on building features, not managing infrastructure.

NotebookLM