BIENVENIDOS

aws Cloud Clubs

aws Student Community Day

Medellín

aws Women Colombia User Group

aws Cloud Clubs IUDIGITAL

aws Cloud Clubs UD

aws Cloud Clubs UDEA

aws Cloud Clubs EAN

aws Cloud Clubs aws Student Community Day Medellín

Lambdaless: Serverless sin Lambda

Lucas Vera Toro
Staff Engineer en
Serverless Guru

Más de 10 años en software y más de 5 años con AWS
Arquitecto técnico
AWS Community Builder y entusiasta Serverless

Lucas Vera Toro

Staff Engineer
Serverless Guru

https://www.linkedin.com/in/lucas-vera-toro-1355b479/

LinkedIn

aws Cloud Clubs
aws Student Community Day
Medellín

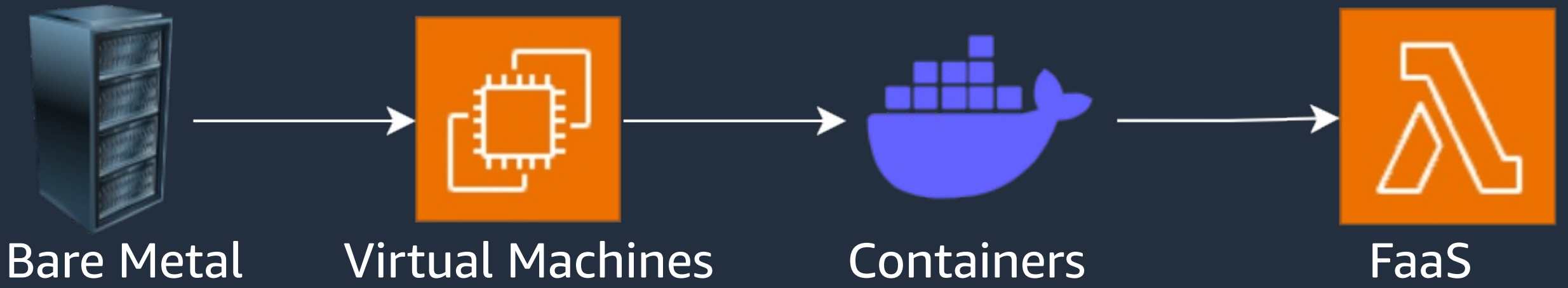
¿Qué tenemos para hoy?

- Serverless y Lambda
- Caso 1: API Gateway
- Caso 2: Step Function
- Caso 3: AppSync
- Conclusión
- Preguntas

aws Cloud Clubs
aws Student Community Day
Medellín

Serverless y Lambda

¿Qué es Serverless?



* Tomado de aws.amazon.com

aws Cloud Clubs aws Student Community Day Medellín

Objetivo de Serverless

- Correr código, Manejar datos e Integraciones “sin servidores”
- Eliminar las tareas administrativas de infraestructura
- Mayor enfoque en **clientes y producto**

Características de Servicios Serverless

Un servicio se puede considerar “Serverless” cuando

- El modelo de pago es por ejecución
- Es auto-escalable
- Alta disponibilidad

Historia de AWS Lambda

- Empezó como una iniciativa dentro de S3
- En 2013, el patrón era “generar flotas de EC2”
- Se desarrolla **Firecracker**
- Genera una instancia **efímera** que corre tú código
- Popularizó el uso de “serverless” y sus beneficios

¿Por qué mirar más allá de Lambda?

Aunque Lambda es un servicio muy poderoso y muy usado, a veces se puede revisar si su uso es adecuado en determinada arquitectura:

- Menos artefactos que mantener
- Evitar “Cold Starts”
- Costos Innecesarios

Una nota sobre arquitectura

A la hora de revisar una arquitectura hay varios factores que se deben evaluar.

- Mantenibilidad
- Escalabilidad
- Costos
- Latencia
- entre otras…

aws Cloud Clubs
aws Student Community Day
Medellín

Caso de Uso 1: API Gateway

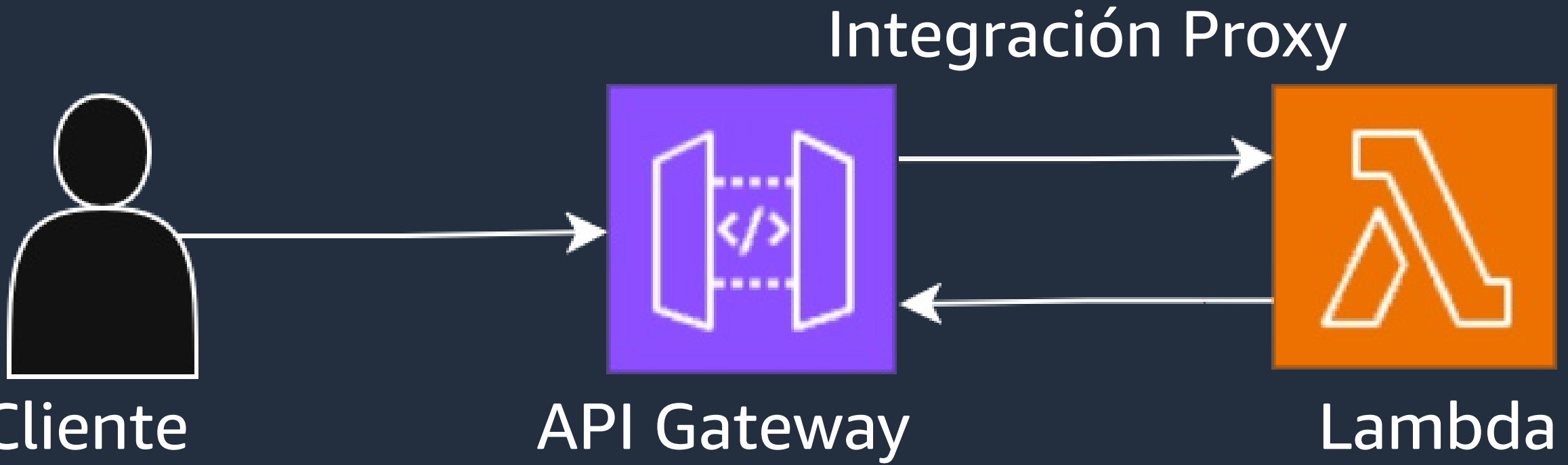
API CRUD

- Create
- Read
- Update
- Delete

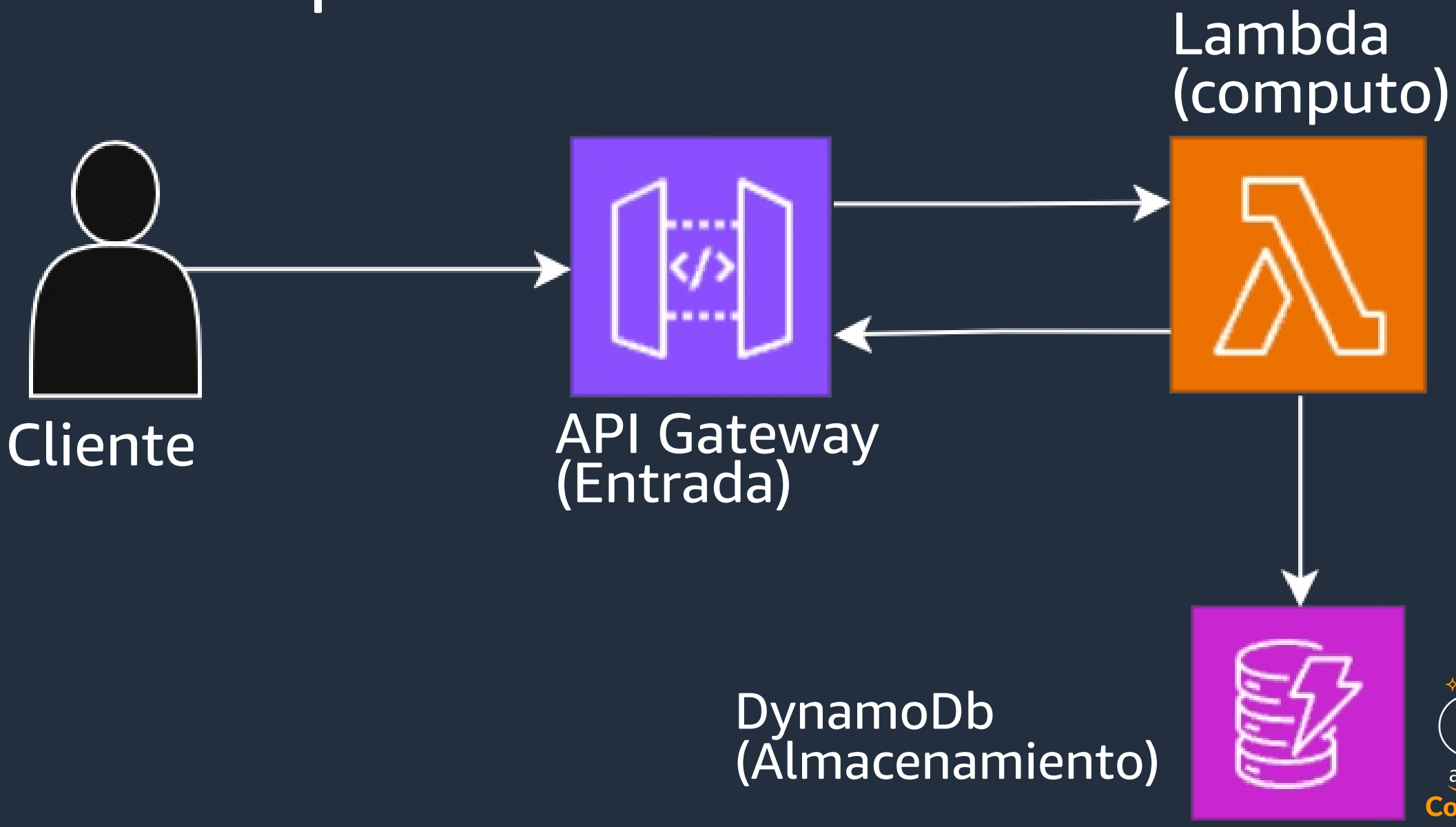
API Gateway

- Facilita la creación de APIs
- Autenticación integrada
- Servicio “Regional”
- **Integración directa** con lambda y otros 100+ servicios

API Gateway + Lambda



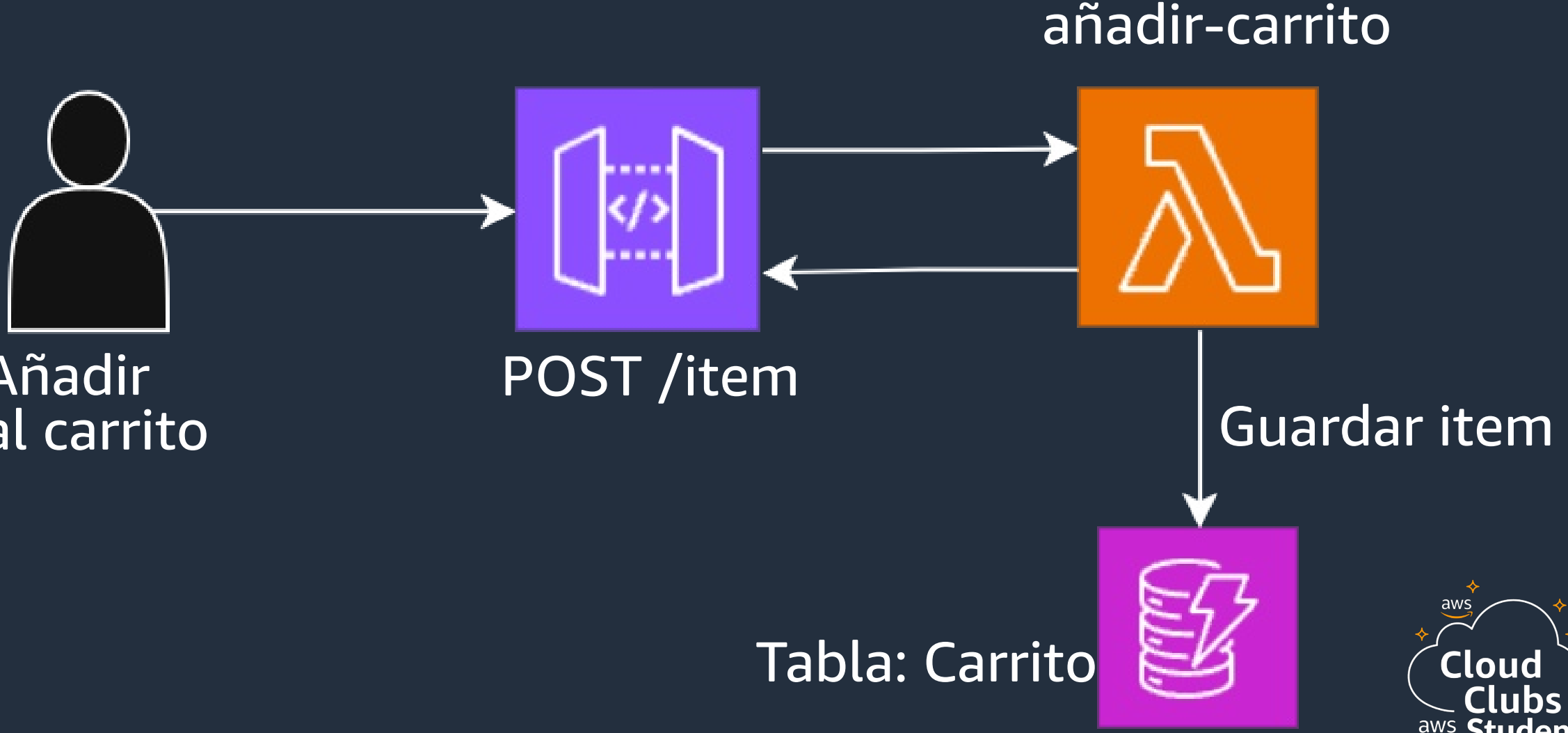
CRUD Típico Serverless



Escenarios CRUD API

- Validar autenticación y autorización
- Validar los datos
- “Mapear” datos de entrada y salida
- Leer o escribir de una base de datos
- Gestionar errores
- Logs, Metricas, Trazas

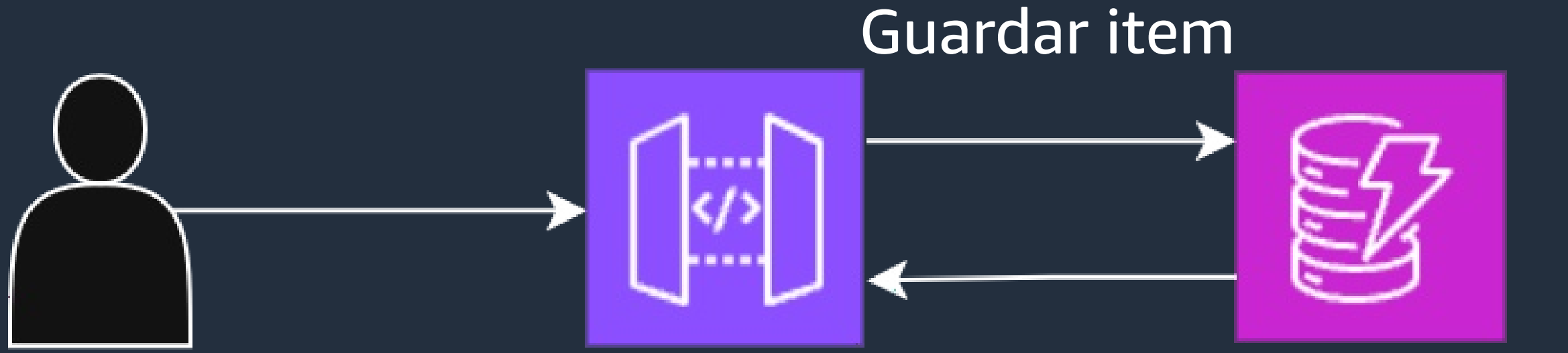
Ejemplo: Carrito de compras



Ejemplo: Carrito de compras

- Típicamente lambda realiza tareas de validación de entrada, mapeo y manejo de errores
- Realmente solo necesitamos “**validar y guardar**”
- Lambda nos añade costo adicional, latencia y otro artefacto
- Lambda cold start

Y si quitamos la lambda?



API Gateway con integración directa

- Ahora las tareas que hacía la lambda pasan a hacerse en **api gateway**
  - Validación
  - “Mapeo”
  - Manejo de errores
  - Logging y trazabilidad
- A continuación veremos un ejemplo de implementación

Ejemplo en CDK: API Gateway

```
// API Gateway with logging enabled
const shoppingCartApi = new apigateway.RestApi(this, 'ShoppingCartApi', {
  restApiName: 'cart-lambdaless',
  description: 'Manages cart items with direct DynamoDB integration',
  deployOptions: {
    accessLogDestination: new apigateway.LogGroupLogDestination(logGroup),
    accessLogFormat: apigateway.AccessLogFormat.jsonWithStandardFields(),
    loggingLevel: apigateway.MethodLoggingLevel.INFO,
    dataTraceEnabled: true,
    metricsEnabled: true,
  },
})

// IAM role for API Gateway to call DynamoDB
const integrationRole = new iam.Role(this, 'ApiGatewayDynamoDBRole', {
  assumedBy: new iam.ServicePrincipal('apigateway.amazonaws.com'),
})
table.grantReadWriteData(integrationRole)
```

Definimos nuestra API usando CDK y Typescript

Ejemplo en CDK: API Gateway

```
// API Gateway with logging enabled
const shoppingCartApi = new apigateway.RestApi(this, 'ShoppingCartApi', {
  restApiName: 'cart-lambdaless',
  description: 'Manages cart items with direct DynamoDB integration',
  deployOptions: {
    accessLogDestination: new apigateway.LogGroupLogDestination(logGroup),
    accessLogFormat: apigateway.AccessLogFormat.jsonWithStandardFields(),
    loggingLevel: apigateway.MethodLoggingLevel.INFO,
    dataTraceEnabled: true,
    metricsEnabled: true,
  },
})

// IAM role for API Gateway to call DynamoDB
const integrationRole = new iam.Role(this, 'ApiGatewayDynamoDBRole', {
  assumedBy: new iam.ServicePrincipal('apigateway.amazonaws.com'),
})
table.grantReadWriteData(integrationRole)
```

Verde: Definición de API Gateway

Azul: Configuración de observabilidad

Amarillo: Permisos

Ejemplo en CDK: Modelo de validación

```
// Validation model
const validationModel = shoppingCartApi.addModel('ValidationModel', {
  contentType: 'application/json',
  modelName: 'InputValidationModel',
  schema: {
    type: apigateway.JsonSchemaType.OBJECT,
    properties: {
      userId: { type: apigateway.JsonSchemaType.STRING },
      itemId: { type: apigateway.JsonSchemaType.STRING },
      quantity: { type: apigateway.JsonSchemaType.NUMBER },
      price: { type: apigateway.JsonSchemaType.NUMBER },
    },
    required: ['userId', 'itemId', 'quantity', 'price'],
  },
})
```

El modelo de validación permite a API Gateway validar los llamados

Ejemplo en CDK: Modelo de validación

```
// Validation model
const validationModel = shoppingCartApi.addModel('ValidationModel', {
  contentType: 'application/json',
  modelName: 'InputValidationModel',
  schema: {
    type: apigateway.JsonSchemaType.OBJECT,
    properties: {
      userId: { type: apigateway.JsonSchemaType.STRING },
      itemId: { type: apigateway.JsonSchemaType.STRING },
      quantity: { type: apigateway.JsonSchemaType.NUMBER },
      price: { type: apigateway.JsonSchemaType.NUMBER },
    },
    required: ['userId', 'itemId', 'quantity', 'price'],
  },
})
```

Verde: Añadir modelo

Azul: Tipos de campos

Amarillo: Campos requeridos

aws Cloud Clubs
aws Student Community Day
Medellín

Implementación

```
(ed0b2108-1ba8-44f4-bef7-47b3e658f937) Method request body before transformations:
{
    "userId": "2",
    "itemId": "3",
    "quantity": 4,
    "price": 5
}
```

Implementación

```
(ed0b2108-1ba8-44f4-bef7-47b3e658f937) Method response body after transformations:
{
    "message": "Item added successfully"
}
```

Implementación con validación

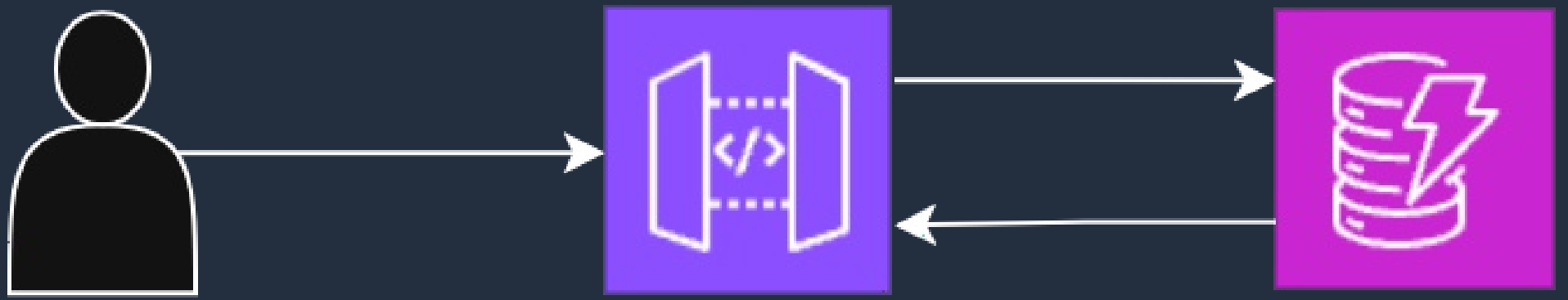
Si enviamos una propiedad que falte, cómo el precio, api gateway dará un error de validación

[object has missing required properties ([“price”])]

```
(5fc5d22e-5473-47c6-96d7-e16a8d4b5c9a) Method request body before transformations: { "userId": "2", "itemId": "3", "quantity": 4 }
(5fc5d22e-5473-47c6-96d7-e16a8d4b5c9a) Request body does not match model schema for content type application/json: [object has missing required properties (["price"])]
(5fc5d22e-5473-47c6-96d7-e16a8d4b5c9a) Method completed with status: 400
```

Lambdaless API

- Tareas que hacía la lambda ahora las hace api gateway
  - Validación
  - Mapeo
  - Errores
- Reducimos cold starts y código



Lambdaless API: Comparación

Métrica	Con Lambda	Sin Lambda
Latencia	*Promedio 300ms	*Promedio 200ms
Coste	≈ 12 USD / mes	≈ 4 USD / mes
Artefactos	API GW + Lambda + DDB	API GW + DDB

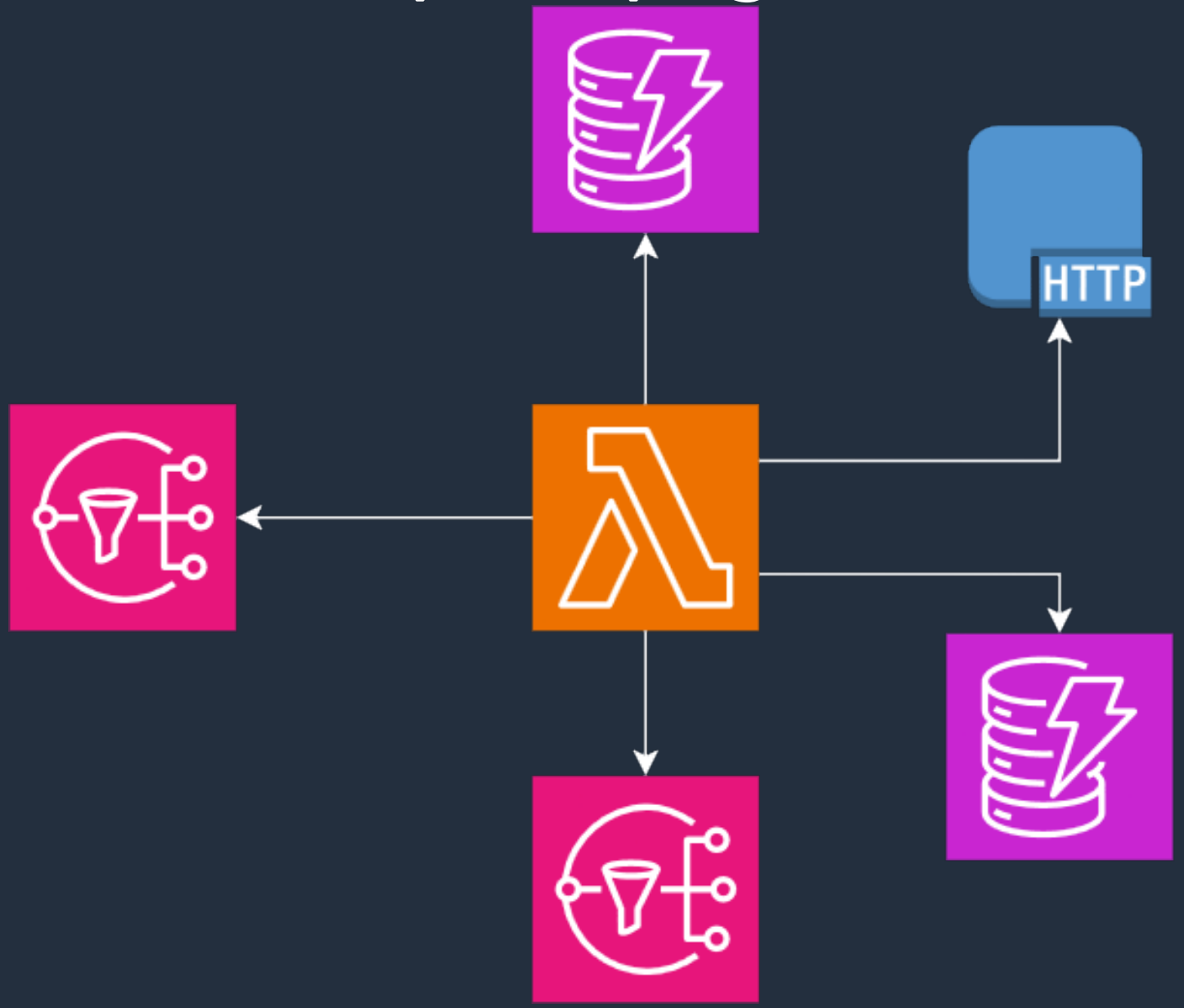
aws Cloud Clubs
aws Student Community Day
Medellín

Caso de Uso 2: Step Function

Caso de uso: Procesar post-pago

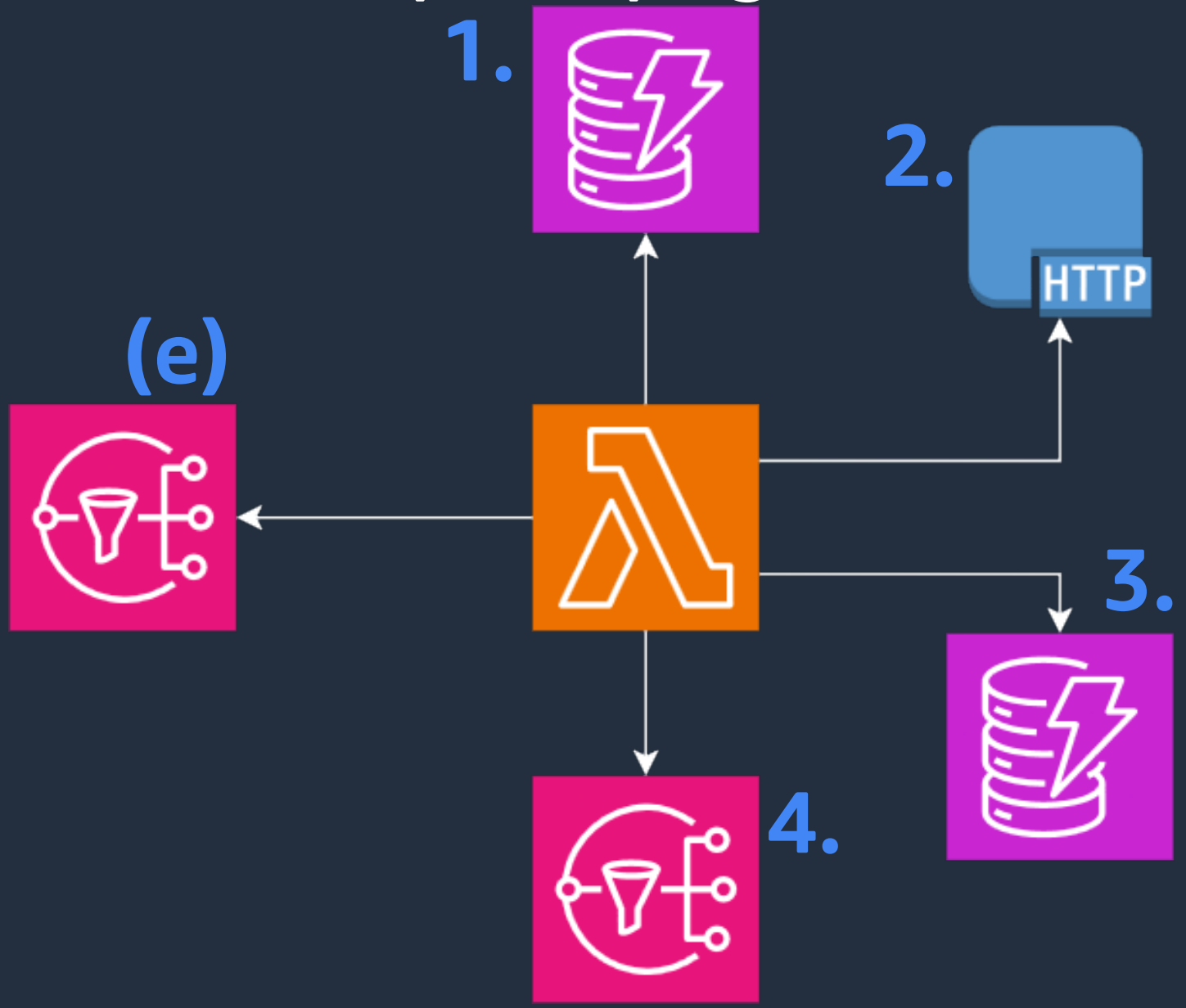
- Siguiendo con el ejemplo anterior, el usuario termina de pagar
- Hay actividades que deben realizarse como:
  - Verificar el pago
  - Enviar ítems a “fulfillment”
  - Crear historial de orden
  - Notificar detalles de envío
  - Manejar errores

Procesar post-pago con lambda



Podemos usar una lambda para orquestar todas las tareas

Procesar post-pago con lambda

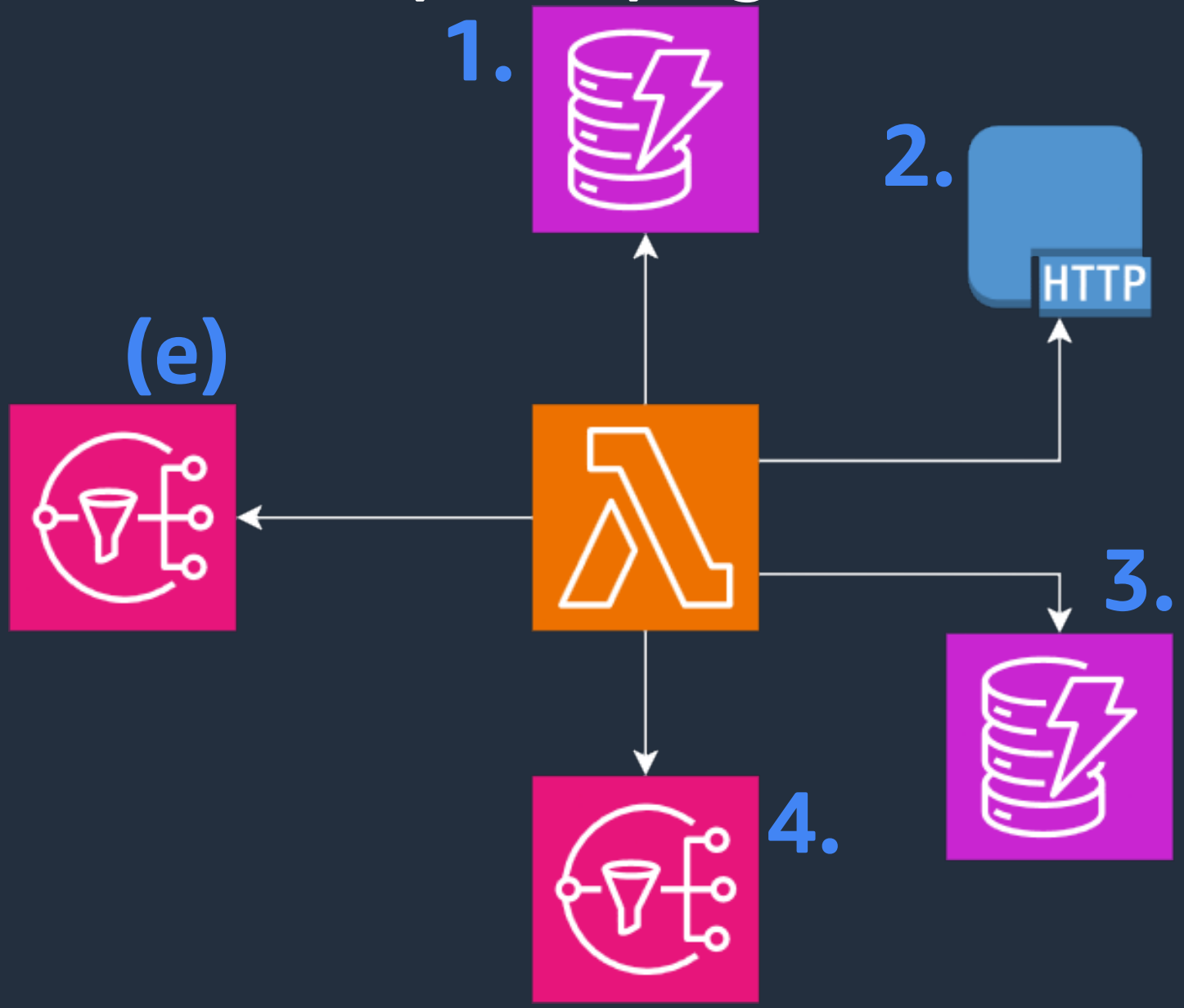


1. Leer pago y validar
2. Enviar a fulfillment
3. Crear historial
4. Enviar notificación

(e). Si hay errores, notificar a agente de soporte

aws Cloud Clubs
aws Student Community Day
Medellín

Procesar post-pago con lambda

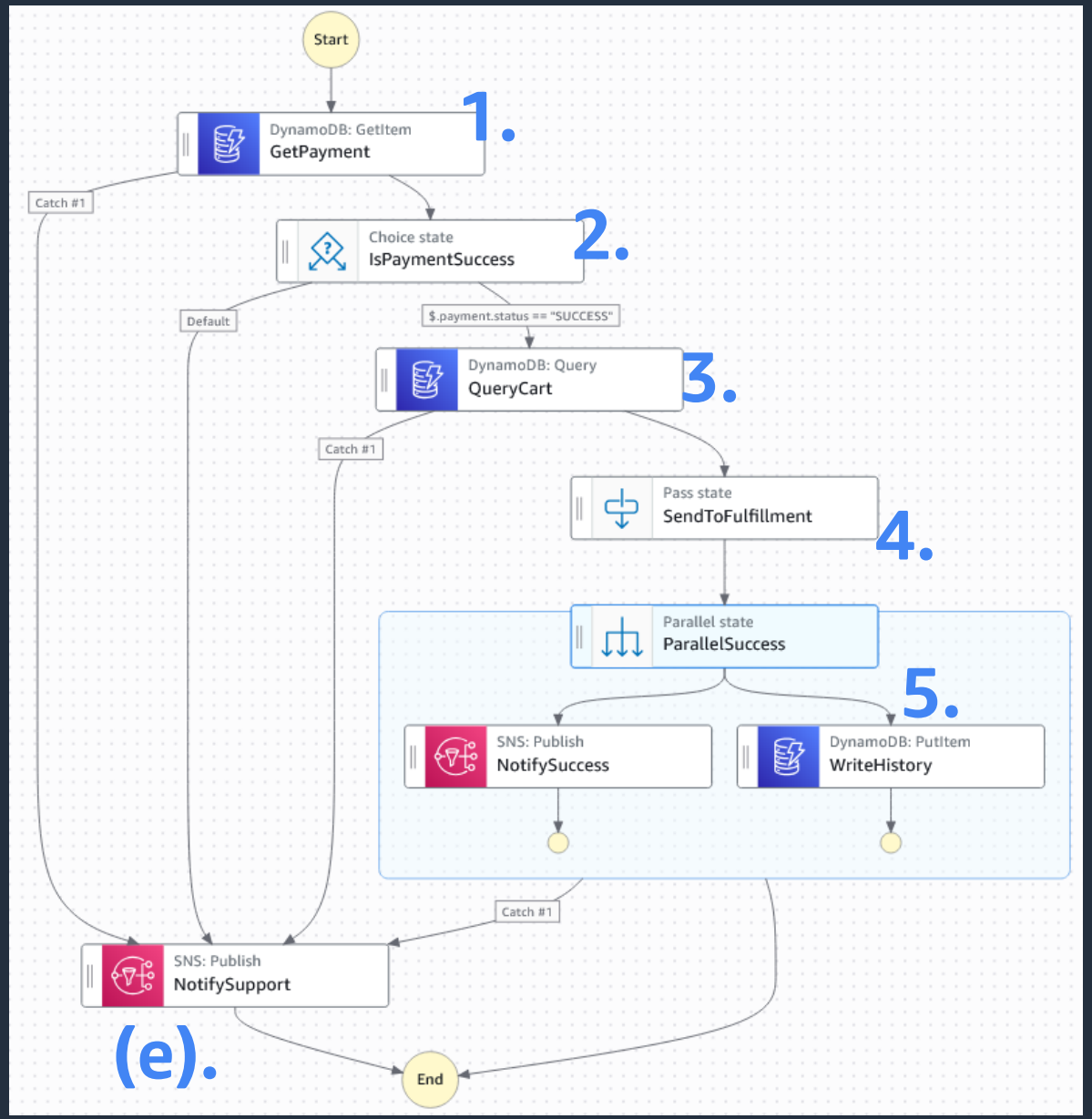


- Demasiada responsabilidad
- Complejidad alta
- Código altamente acoplado
- Manejo de errores y retries manual

Step Functions

- Servicio “orquestador” serverless
- Permite construir aplicaciones distribuidas
- Permite flujos síncronos y asíncronos
- Diseño de flujo visual e intuitivo
- Permite Agregar un paso manual o “humano”
- El tiempo de procesamiento puede ser mayor a 15 minutos
- Podemos evitar el “lambda-lith”

¿Por qué Step Functions?



1. Leer pago
2. Validar pago
3. Leer carrito
4. enviar a fulfillment
5. Notificar y crear historial

(e). Manejo de errores

aws Cloud Clubs aws Student Community Day Medellín

Ejemplo en CDK: Tasks

```
// Tasks
const notifySupport = new tasks.SnsPublish(this, 'NotifySupport', {···
})

const getPayment = new tasks.DynamoGetItem(this, 'GetPayment', {···
}).addCatch(notifySupport, { resultPath: '$.error' })

const queryCart = new tasks.CallAwsService(this, 'QueryCart', {···
}).addCatch(notifySupport, { resultPath: '$.error' })
```

Usando CDK y Typescript podemos definir y configurar cada paso dentro de la Step Function

Ejemplo en CDK: Tasks

```
// Tasks
const notifySupport = new tasks.SnsPublish(this, 'NotifySupport', {···
})

const getPayment = new tasks.DynamoGetItem(this, 'GetPayment', {···
}).addCatch(notifySupport, { resultPath: '$.error' })

const queryCart = new tasks.CallAwsService(this, 'QueryCart', {···
}).addCatch(notifySupport, { resultPath: '$.error' })
```

- **Verde**: Definimos qué tasks para cada paso
- **Azul**: Podemos definir error handling para cada task

Ejemplo en CDK: Seguridad y Observabilidad

```
const smRole = new iam.Role(this, 'StateMachineRole', {
  assumedBy: new iam.ServicePrincipal('states.amazonaws.com'),
})
paymentsTable.grantReadData(smRole)
cartTable.grantReadData(smRole)
inventoryTable.grantReadWriteData(smRole)
historyTable.grantWriteData(smRole)
supportTopic.grantPublish(smRole)
successTopic.grantPublish(smRole)
logGroup.grantWrite(smRole)

const sm = new stepfunctions.StateMachine(this, 'PostPaymentStateMachine', {
  definition,
  role: smRole,
  logs: { destination: logGroup, level: stepfunctions.LogLevel.ALL },
})
```

Definir el rol y la configuración de observabilidad es sencillo

Ejemplo en CDK: Seguridad y Observabilidad

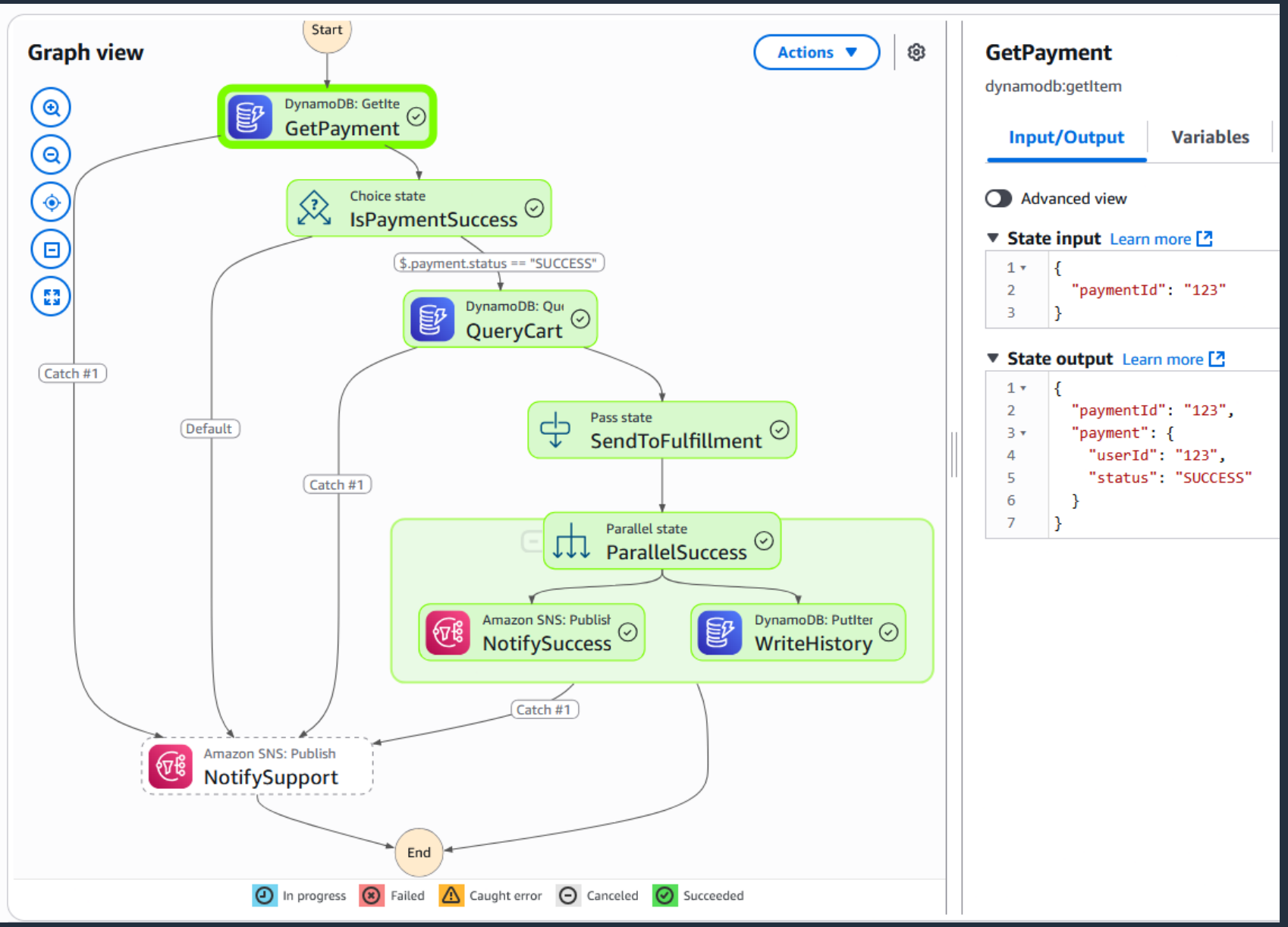
```
const smRole = new iam.Role(this, 'StateMachineRole', {
  assumedBy: new iam.ServicePrincipal('states.amazonaws.com'),
})
paymentsTable.grantReadData(smRole)
cartTable.grantReadData(smRole)
inventoryTable.grantReadWriteData(smRole)
historyTable.grantWriteData(smRole)
supportTopic.grantPublish(smRole)
successTopic.grantPublish(smRole)
logGroup.grantWrite(smRole)

const sm = new stepfunctions.StateMachine(this, 'PostPaymentStateMachine', {
  definition,
  role: smRole,
  logs: { destination: logGroup, level: stepfunctions.LogLevel.ALL },
})
```

Verde: Crear el rol

Azul: Dar permisos al rol

Amarillo: Configurar rol y logs

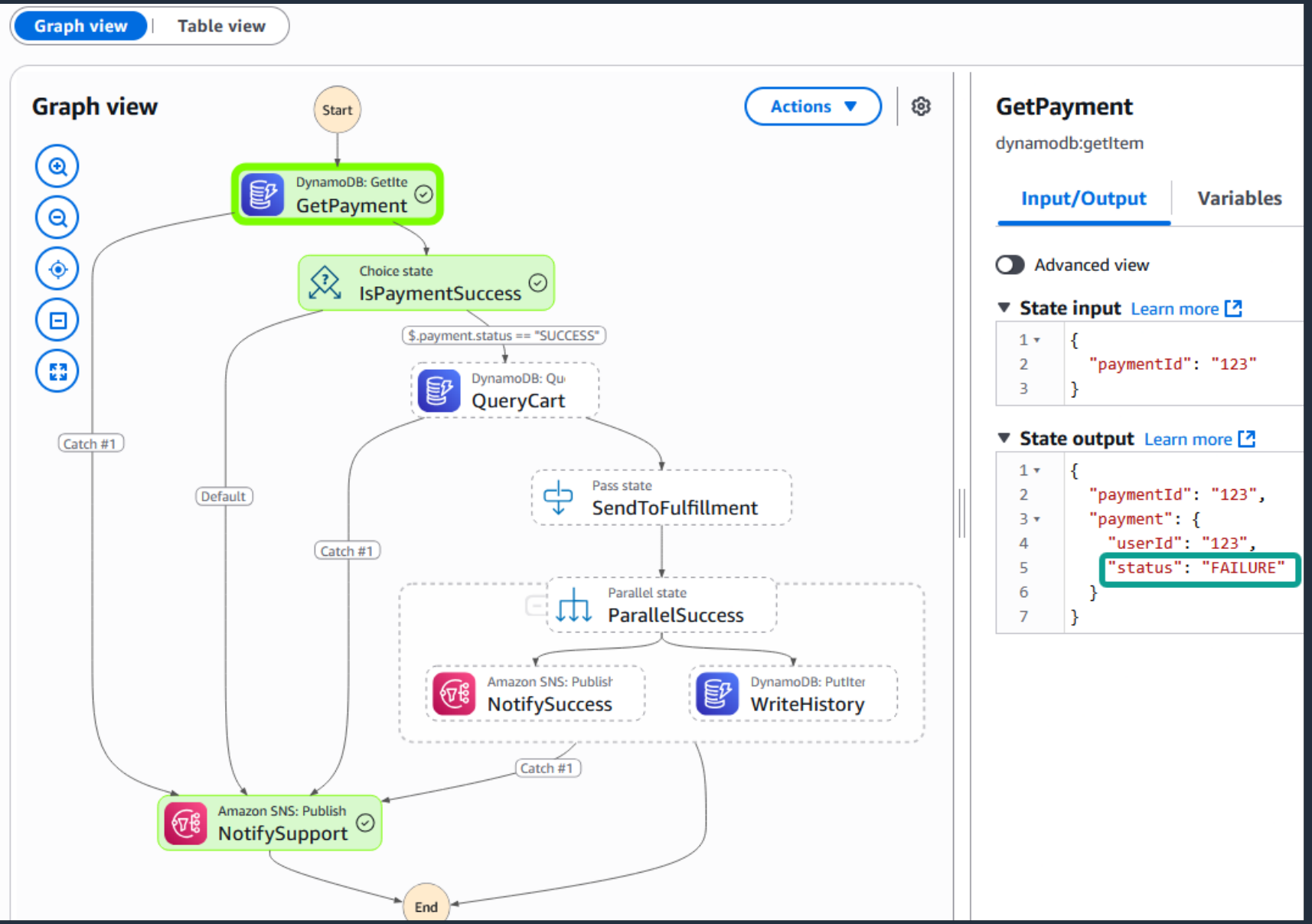
Ejemplo ejecución



Podemos visualizar las ejecuciones, paso a paso

aws Cloud Clubs
aws Student Community Day
Medellín

Ejemplo ejecución sin pago



En este caso, podemos ver el resultado de la operación. El pago no fue validado

aws Cloud Clubs
aws Student Community Day
Medellín

Ventajas de este enfoque

- Workflow visual facilita revisión, seguimiento y debugging
- Manejo de errores y retries nativo
- Estados desacoplados e independientes
- Menos código y menos deuda técnica

Comparativa

Aspecto	Lambda	Step Functions
Lógica central	Muchas líneas de código (handler)	5-10 estados declarativos en CDK
Errores y retries	Implementación Manual	Implementación nativa
Visibilidad	Logs y trazas	Timeline visual de la ejecución

aws Cloud Clubs
aws Student Community Day
Medellín

Caso de Uso 3: AppSync

AppSync

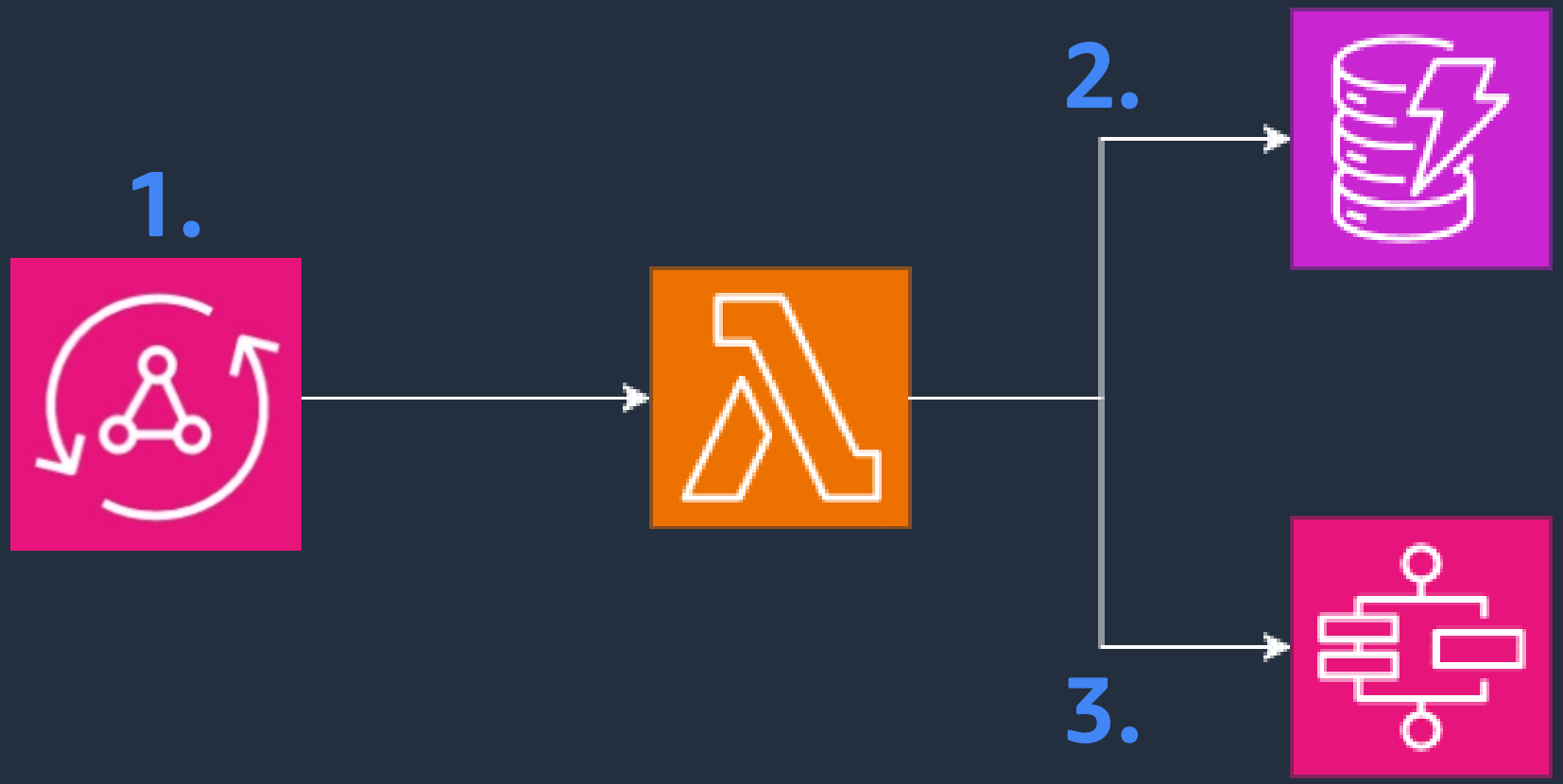
- Servicio para crear APIs Graphql serverless
- Crear APIs escalables y flexibles
- Integraciones nativas con muchos servicios de AWS

GraphQL 101

```
enum ItemStatus {
  AVAILABLE
  OUT_OF_STOCK
  DISCONTINUED
}

type Item {
  id: ID!
  name: String!
  price: Float!
  status: ItemStatus!
  description: String
}

type Query {
  getItem(id: ID!): Item!
  listItems(status: ItemStatus): [Item]!
}
```

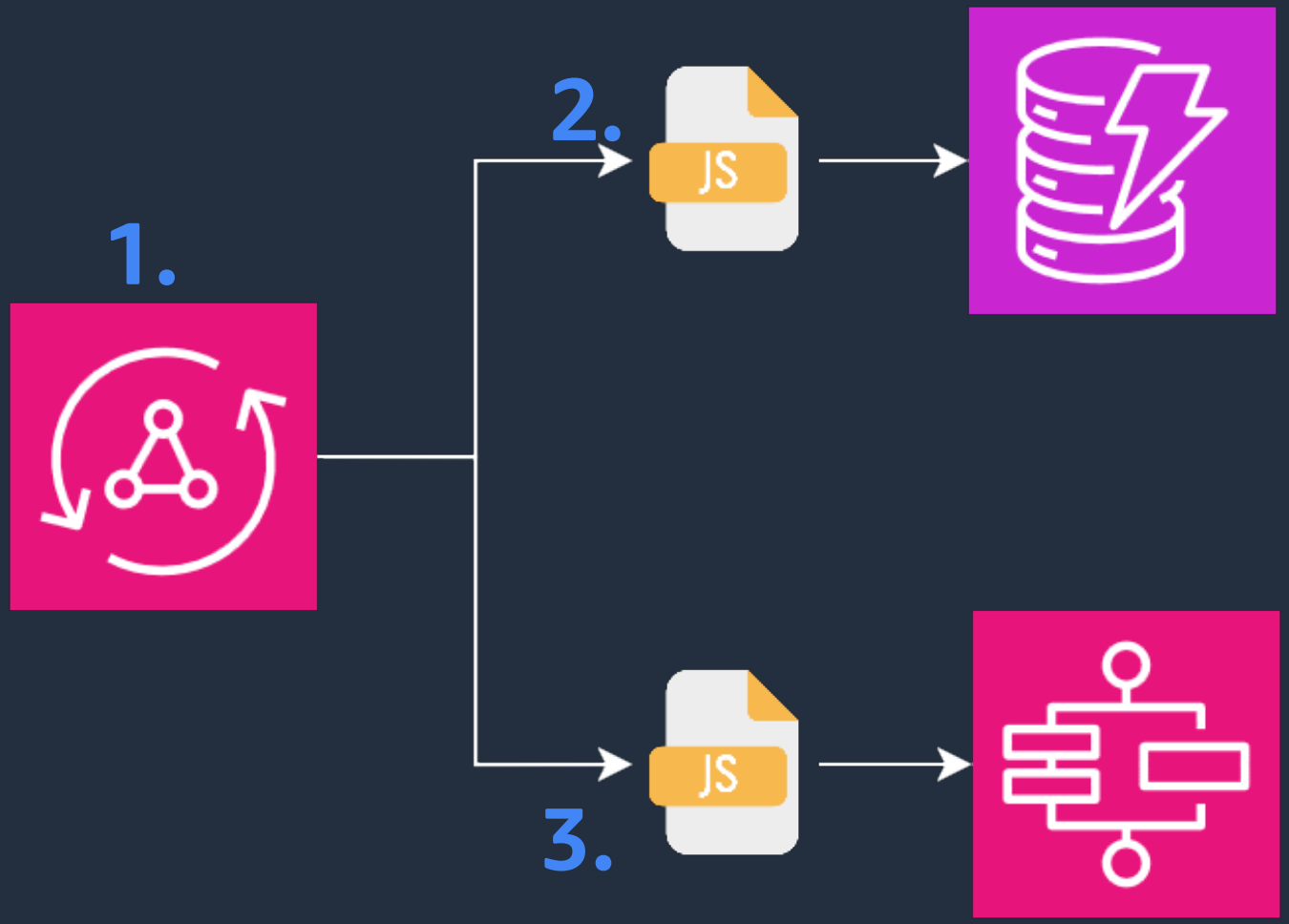
- Runtime y lenguaje que se encarga de completar operaciones sobre datos
- Queries, Mutations, Subscriptions
- Elimina “over-fetching”
- Flexible y con capacidades “real-time”
- Schema es enforzado

Caso de Uso: Invocar step function anterior



1. Recibir request “procesar post-pago”
2. Lambda lee tabla de pagos
3. Lambda Invoca step function

Caso de Uso: Invocar step function anterior



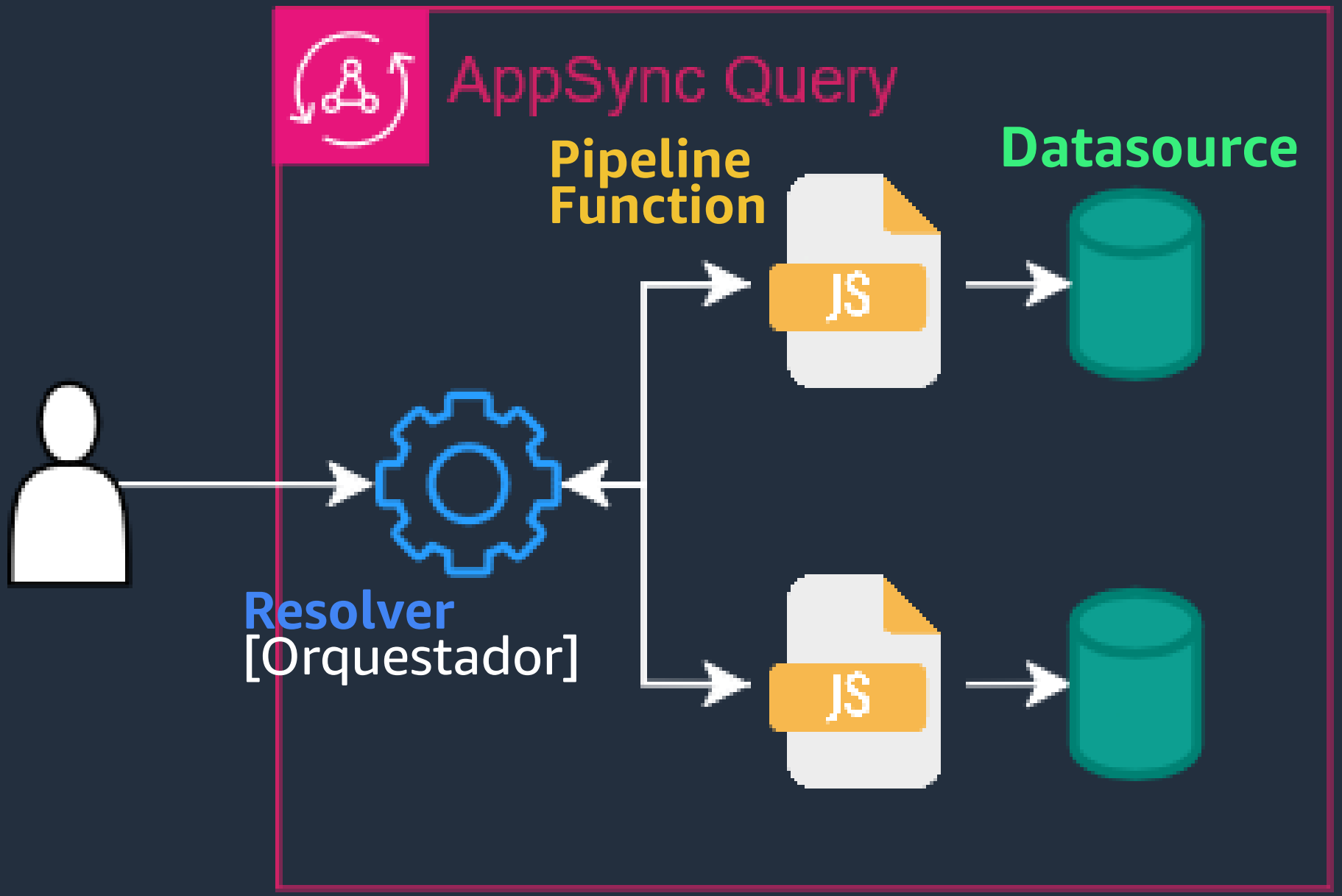
1. Recibir request “procesar post-pago”
2. AppSync **resolver** lee tabla de pagos
3. AppSync **resolver** invoca step function

aws Cloud Clubs aws Student Community Day Medellín

AppSync Pipeline Resolver

- Es una función aislada que recibe una entrada y entrega una salida, actuando sobre un “data source”
- Javascript o VTL
- Permite construir apis robustas y con componentes reusables

Componentes de AppSync



Resolver: Decide el orden sequencial

Pipeline Function: Componente conectado a un único datasource

Datasource: Dónde se lee o escribe

aws Cloud Clubs aws Student Community Day Medellín

Ejemplo de Implementación: Definir API

```yaml
plugins:
  - serverless-appsync-plugin

appSync:
  name: ${self:service}-api
  authentication:
    type: API_KEY
  apiKeys:
    - name: lambdaless-appsync-api-key
      description: API key for the Lambdaless AppSync API
  logging:
    enabled: true
    level: INFO
  xrayEnabled: true
  environment:
    paymentTableName: ${self:custom.paymentTableName}
    stepFunctionArn: ${self:custom.stepFunctionArn}
```

Podemos usar **serverless framework** para definir nuestra AppSync API

Ejemplo de Implementación: Definir API

```yaml
plugins:
  - serverless-appsync-plugin

appSync:
  name: ${self:service}-api
  authentication:
    type: API_KEY
  apiKeys:
    - name: lambdaless-appsync-api-key
      description: API key for the Lambdaless AppSync API
  logging:
    enabled: true
    level: INFO
  xrayEnabled: true
  environment:
    paymentTableName: ${self:custom.paymentTableName}
    stepFunctionArn: ${self:custom.stepFunctionArn}
```

Verde: Plugin para sls-framework

Azul: Declaración de auth

Amarillo: Configuración de observabilidad

Ejemplo de Implementación: Datasource

```yaml
dataSources:
  processPaymentStepFunctionDs:
    type: HTTP
    config:
      endpoint: "https://states.${self:provider.region}.amazonaws.com"
      serviceRoleArn: !GetAtt [AppSyncServiceRole, Arn]
      authorizationConfig:
        authorizationType: AWS_IAM
        awsIamConfig:
          signingRegion: ${self:provider.region}
          signingServiceName: states
  paymentTableDs:
    type: AMAZON_DYNAMODB
    config:
      tableName: ${self:custom.paymentTableName}
      serviceRoleArn: !GetAtt [AppSyncServiceRole, Arn]
```

Definimos los **Data source** primero.

Para nuestro ejemplo tenemos una tabla de dynamodb y la step function

Ejemplo de Implementación: Datasource

```yaml
dataSources:
  processPaymentStepFunctionDs:
    type: HTTP
    config:
      endpoint: "https://states.${self:provider.region}.amazonaws.com"
      serviceRoleArn: !GetAtt [AppSyncServiceRole, Arn]
      authorizationConfig:
        authorizationType: AWS_IAM
        awsIamConfig:
          signingRegion: ${self:provider.region}
          signingServiceName: states
  paymentTableDs:
    type: AMAZON_DYNAMODB
    config:
      tableName: ${self:custom.paymentTableName}
      serviceRoleArn: !GetAtt [AppSyncServiceRole, Arn]
```

Verde: Nombre del Datasource

Azul: Tipo de Datasource

Amarillo: Configuración de auth

Ejemplo de Implementación: Pipeline Function

```
import { util } from '@aws-appsync/utils';
import * as dynamoDb from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
  const { paymentId } = ctx.args;

  return dynamoDb.get({
    key: { paymentId }
  });
}

export function response(ctx) {
  const { error, result } = ctx;

  if (error) {
    console.error('Error retrieving payment:', error);
    return util.error(`Error retrieving payment: ${error.message}`);
  }

  return result;
}
```

Verde: Request/Response

Azul: Cómo interactúa con el datasource

Amarillo: Manejo de errores en el response

Ejemplo de Implementación: Resolver

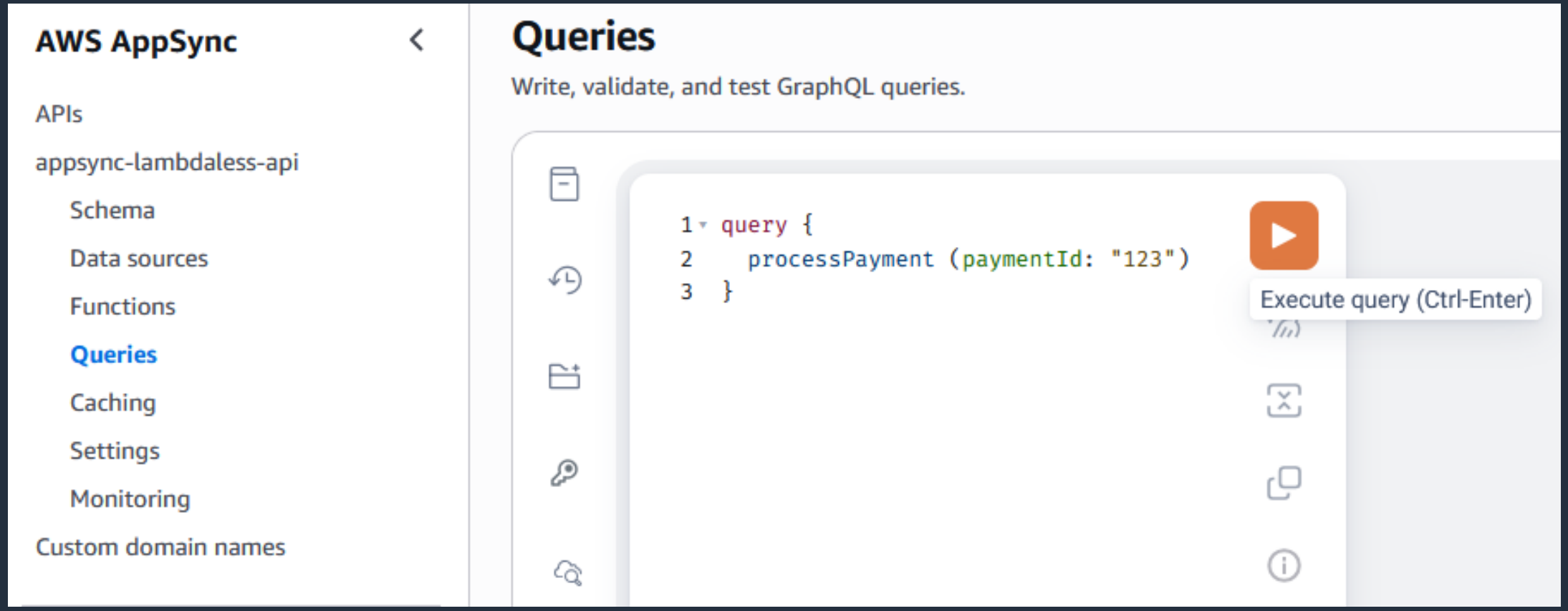
```
schema {
  query: Query
}
type Query @aws_api_key {
  processPayment(paymentId: String!): String
}
```

```
resolvers:
  Query.processPayment:
    functions:
      - retrievePayment
      - startPaymentProcess
```

```
pipelineFunctions:
  retrievePayment: ···
  startPaymentProcess: ···
```

Podemos definir el resto de componentes y el schema

Ejemplo de ejecución



Después de desplegar es fácil ejecutar queries desde la consola de aws

Ejemplo de ejecución

```
775aae30-7b89-441f-9719-3988d6c9c3b7 Begin Request
775aae30-7b89-441f-9719-3988d6c9c3b7 GraphQL Query: query { processPayment (paymentId: "123") }, Operation:
775aae30-7b89-441f-9719-3988d6c9c3b7 Begin Execution - Type Name: Query and Field Name: processPayment
{"logType":"BeforeRequestFunctionEvaluation","path":["processPayment"],"fieldName":"processPayment","resolv
{"logType":"RequestFunctionEvaluation","fieldName":"processPayment","resolverArn":"arn:aws:appsync:us-east-
{"logType":"ResponseFunctionEvaluation","fieldName":"processPayment","resolverArn":"arn:aws:appsync:us-east
{"logType":"RequestFunctionEvaluation","fieldName":"processPayment","resolverArn":"arn:aws:appsync:us-east-
{"logType":"ResponseFunctionEvaluation","fieldName":"processPayment","resolverArn":"arn:aws:appsync:us-east
{"logType":"AfterResponseFunctionEvaluation","path":["processPayment"],"fieldName":"processPayment","resolv
775aae30-7b89-441f-9719-3988d6c9c3b7 End Field Execution
775aae30-7b89-441f-9719-3988d6c9c3b7 Resolver Count: 1
{"duration":117185454,"logType":"ExecutionSummary","requestId":"775aae30-7b89-441f-9719-3988d6c9c3b7","star
{"logType":"RequestSummary","requestId":"775aae30-7b89-441f-9719-3988d6c9c3b7","graphQLAPIId":
775aae30-7b89-441f-9719-3988d6c9c3b7 Request Headers: {content-length=[61], referer=[https://us-east-1.cons
775aae30-7b89-441f-9719-3988d6c9c3b7 Response Headers: {X-Amzn-Trace-Id=Root=1-685cc29b-638c8fc5055a925e474
775aae30-7b89-441f-9719-3988d6c9c3b7 Tokens Consumed: 1
775aae30-7b89-441f-9719-3988d6c9c3b7 End Request
```

Verde: Inicio y Fin

Azul: Ejecución de las funciones dentro del pipeline

Amarillo: Resumen de ejecución

Ejemplo de ejecución con error

Verde: Inicio y Fin

Azul: Ejecución de las funciones dentro del pipeline

Amarillo: Error arrojado

aws Cloud Clubs
aws Student Community Day
Medellín

Ventajas de este enfoque

- Desacoplamiento de componentes en funciones reusables
- Autenticación y autorización nativas
- Facilidad de integración
- Menor costo y latencia al no tener la ejecución de lambdas innecesarias

aws Cloud Clubs
aws Student Community Day
Medellín

Conclusión

¿Cuando usar lambdas?

- Tú lógica negocio es compleja
- Validaciones muy custom
- Necesitas SDKs o dependencias externas
- El equipo de desarrollo tiene experiencia (velocidad)

Trade-offs y análisis

- Flexibilidad vs Simplicidad
  - Lambda: más lógica (código)
  - Lambdaless: **Menos código, más configuración** en infra (VTL, CDK, YAML)
- Testing
  - Lambda: unit + integration tests
  - Lambdaless: **no es tan fácil o estándar**
- Coste y performance
  - Lambda: coste por ms + invocación, cold starts
  - Lambdaless: **pago por uso, latencia baja**

Trade-offs y análisis

Consideración	Pregunta
Equipo	¿Con qué enfoque somos más productivos y podemos iterar rápido?
Mantenimiento	¿Quién mantendrá esto en 6-12 meses?
Producto	¿Qué expectativas hay en cuanto a latencia y costos?

Lo más importante

- No se trata de eliminar Lambdas por que si, sino **elegir la herramienta correcta**
- Al diseñar una solución “Lambdaless”, evalúa frecuencia de cambio, familiaridad del equipo y mantenimiento
- Para proyectos nuevos, considera empezar con integración directa en las **operaciones sencillas** e ir refactorizando a medida que se requiera
- Revisa tus APIs actuales: **identifica dónde pueden cortarse lambdas** sin afectar funcionalidad. Estoy seguro que encontrarás!

aws Cloud Clubs
aws Student Community Day
Medellín

Cloud Clubs aws Student Community Day Medellín

¡Gracias!

Lucas Vera Toro

https://www.linkedin.com/in/lucas-vera-toro-1355b479/

LinkedIn

https://github.com/LucasVera/Lambdaless-APIs

Github