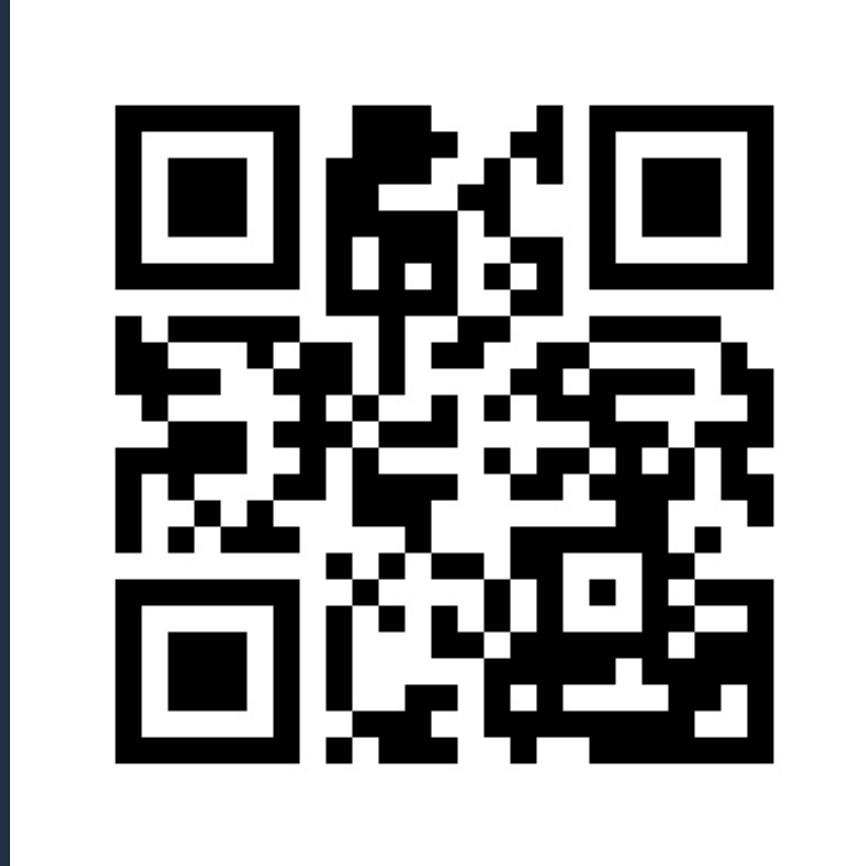




Mejores Prácticas y Patrones Avanzados de Flujo de Trabajo Serverless

Alfredo Peña
Sr. Technical Account Manager - LATAM

Todo en un solo lugar

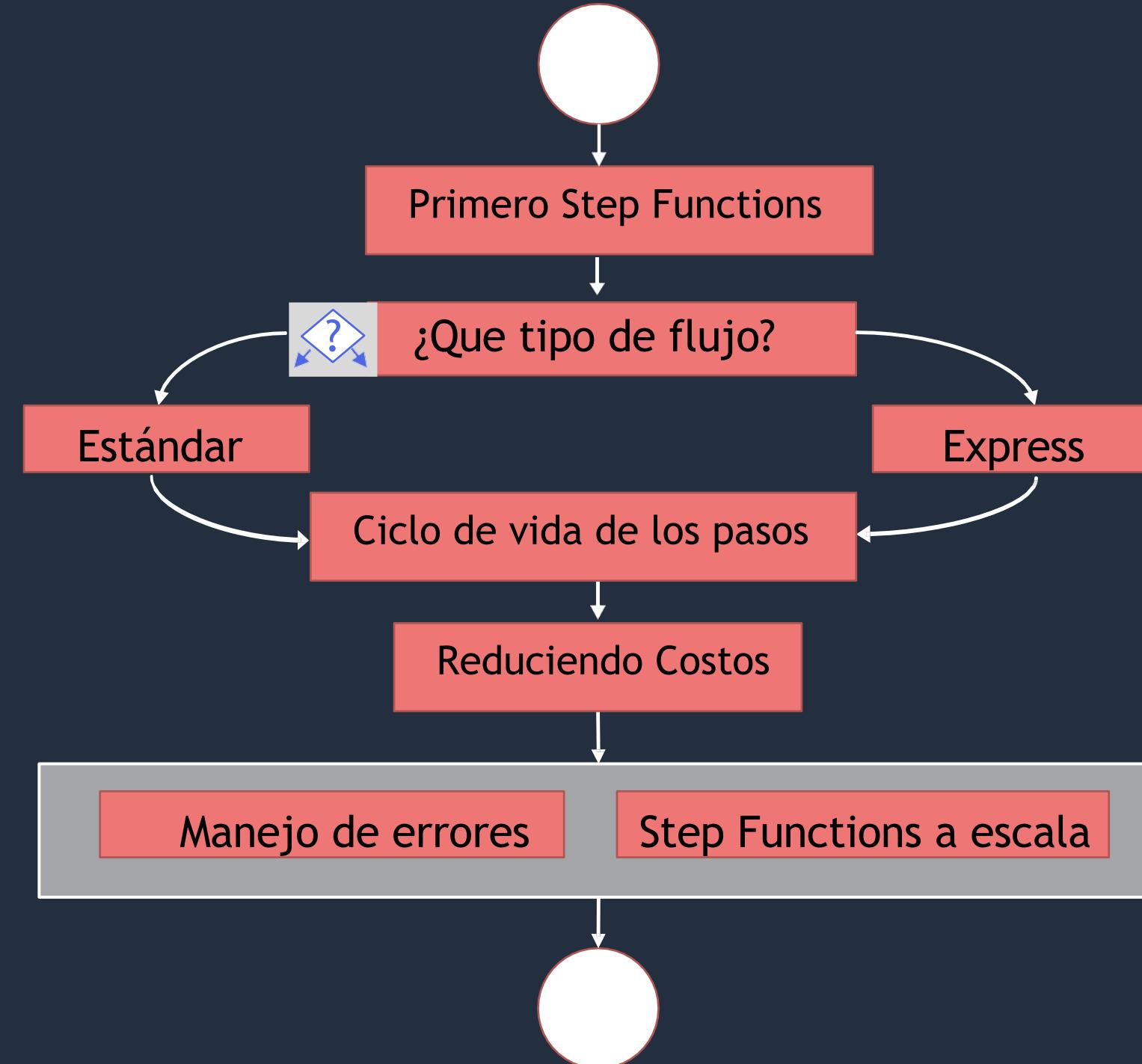


<https://s12d.com/api309>

Recursos para API309
Patrones avanzados de flujo de trabajo
serverless y mejores prácticas

- Plantillas descargables
- Publicaciones de blog
- Vídeos
- Talleres
- Ejemplos de código

Agenda (como un flujo de Step Functions)



El objetivo final



Valor al cliente



Fiabilidad



Resiliencia



Escalabilidad

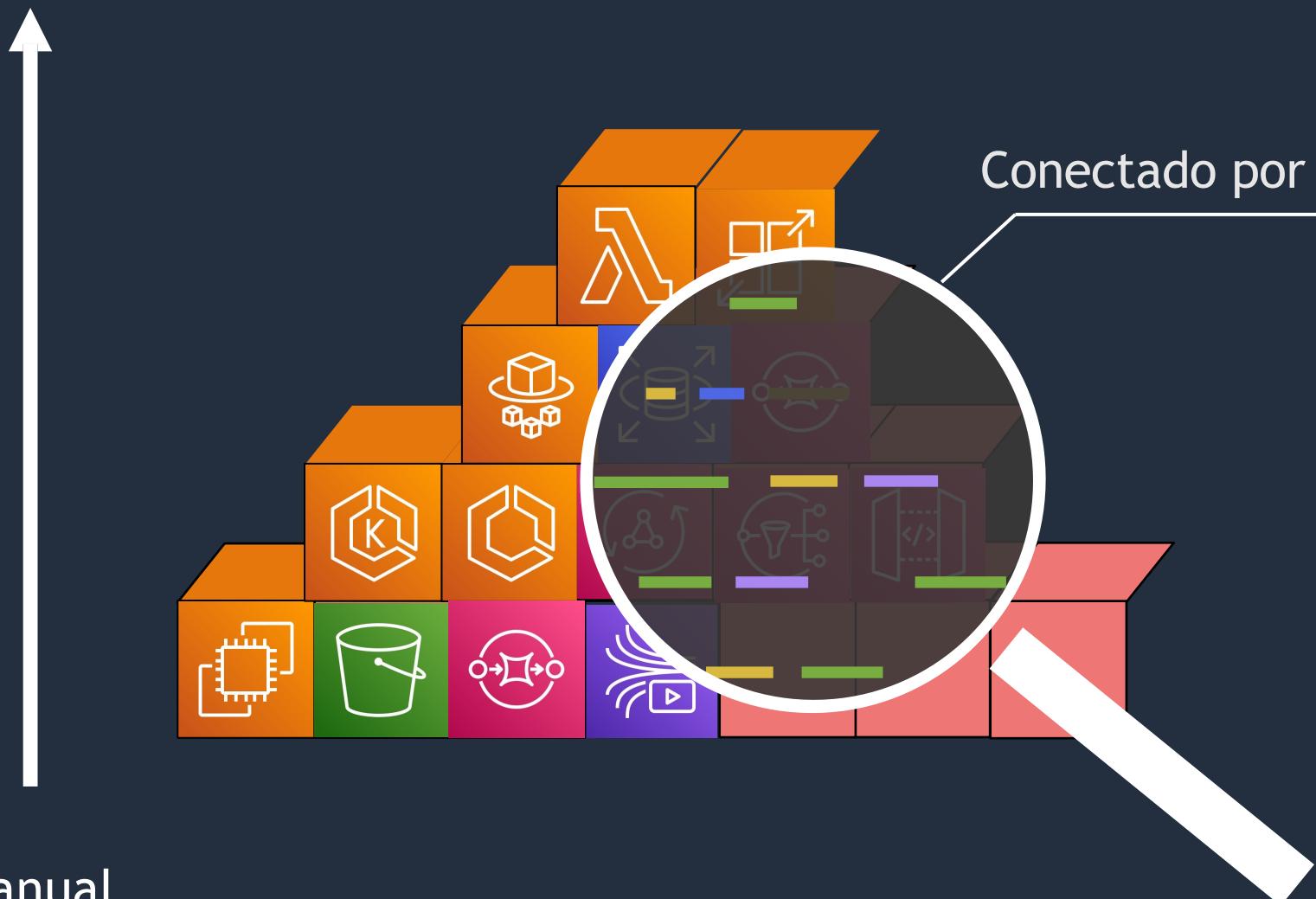


Simplicidad



Y hacerlo más rápido

Gestionado



Serverless

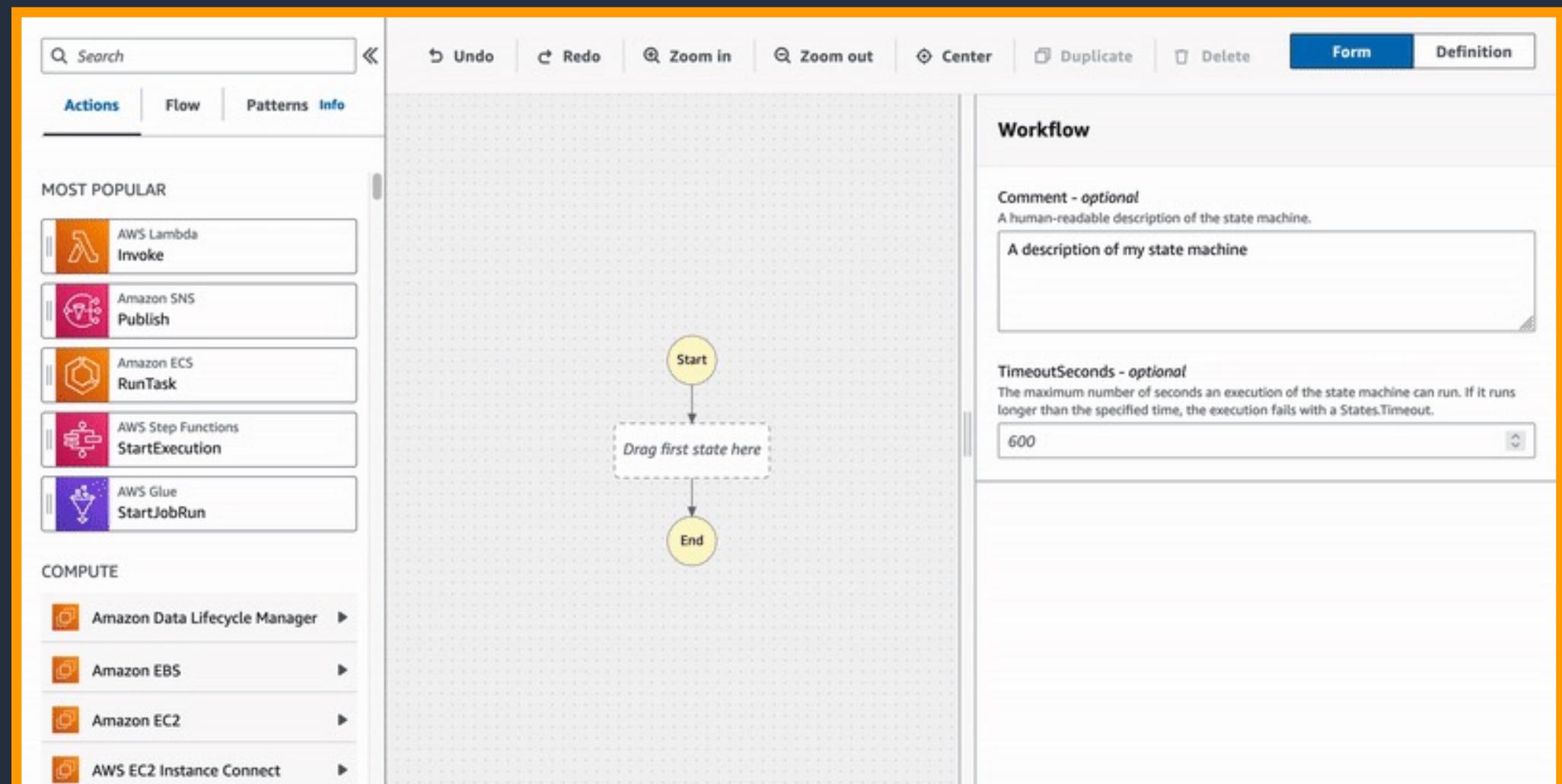
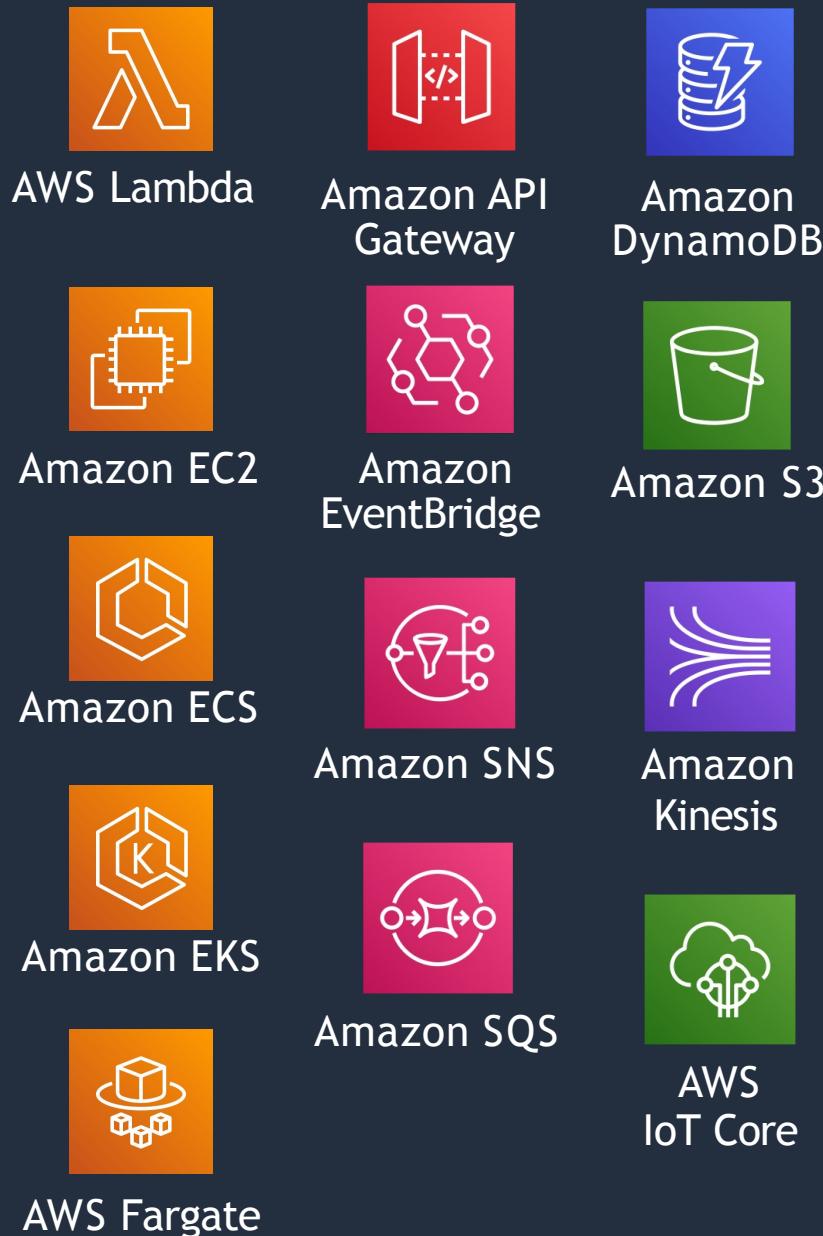
impulsada por eventos en 2023

Desafío Serverless

Cuando su aplicación funciona con varios servicios conectados, ¿cómo puede crear, rastrear, inspeccionar, visualizar y orquestar esas conexiones?



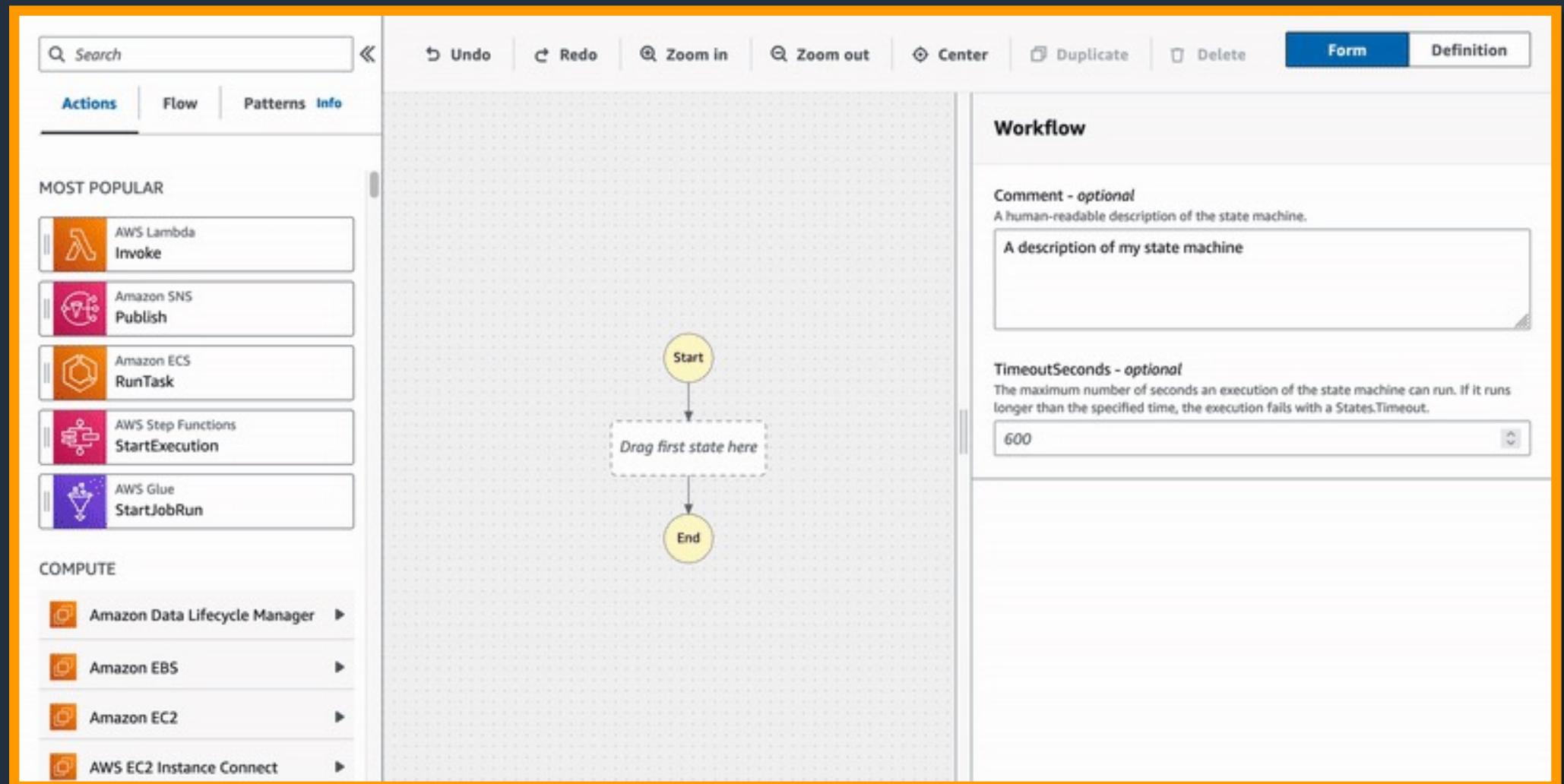
AWS Step Functions es diferente



AWS Step Functions es diferente

SERVICIO DE FLUJO DE TRABAJO VISUAL DE CÓDIGO BAJO SERVERLESS

- Pago por uso
- Escala automáticamente
- Totalmente gestionado
- Arrastrar y soltar o ASL
- Gestión de errores integrada
- Se integra con más de 200 servicios de AWS

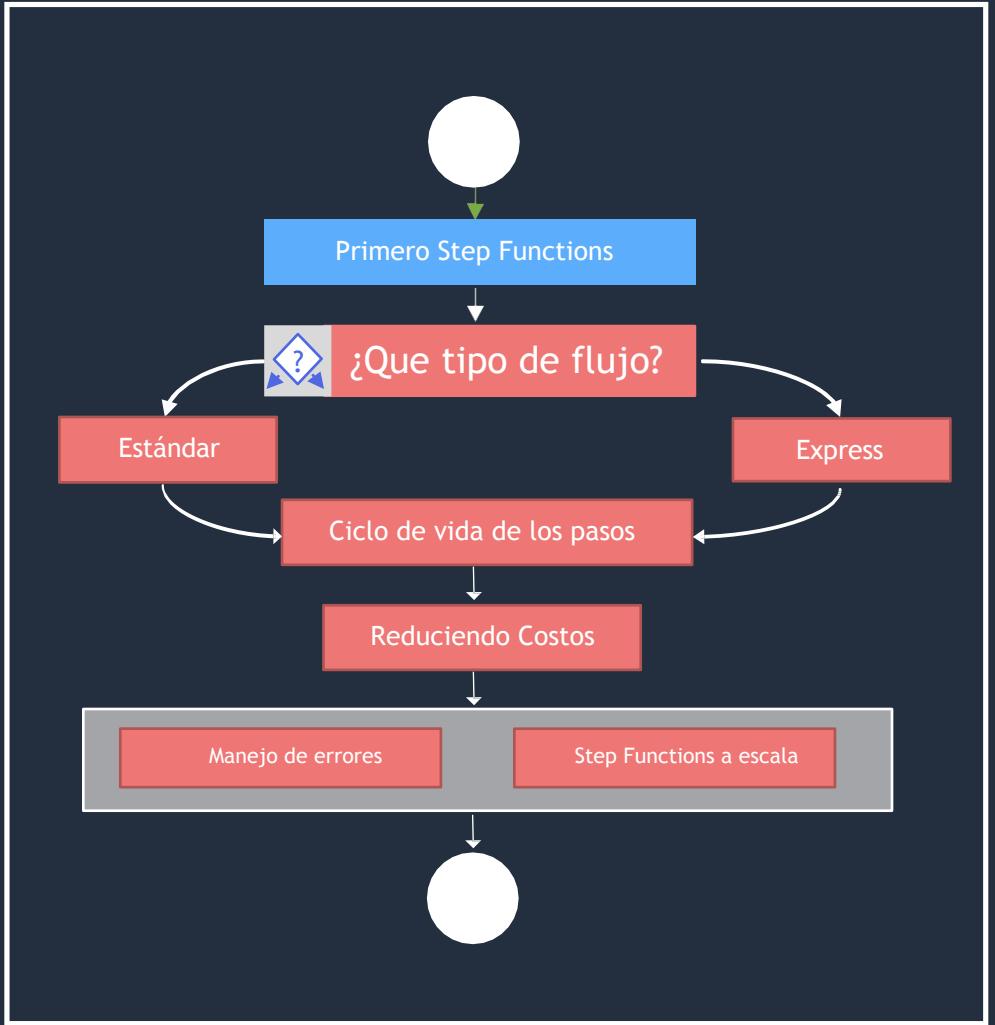


“Primero Step Functions, Step Functions siempre.”

Ben Smith

Principal Developer Advocate, Serverless AWS

<https://s12d.com/api309>



Del código a un flujo de trabajo



app.js

```
const AWS = require('aws-sdk');
const docClient = new AWS.DynamoDB.DocumentClient();

var params = {
  "TableName": "reinvent2022!",
  "Key": {
    "PK": {"S": "wardrobe"},
    "SK": {"S": "shoes"}
  }
}

async function queryItems(){
  try {
    const data = await docClient.getItem(params).promise()
    return data
  } catch (err) {
    return err
  }
}

exports.handler = async (event, context) => {
  try {
    const data = await queryItems()
    return { body: JSON.stringify(data) }
  } catch (err) {
    return { error: err }
  }
}
```

Una función de Lambda que consulta Amazon DynamoDB tiene varias líneas de código



AWS Lambda



Amazon DynamoDB

Del código a un flujo de trabajo



app.js

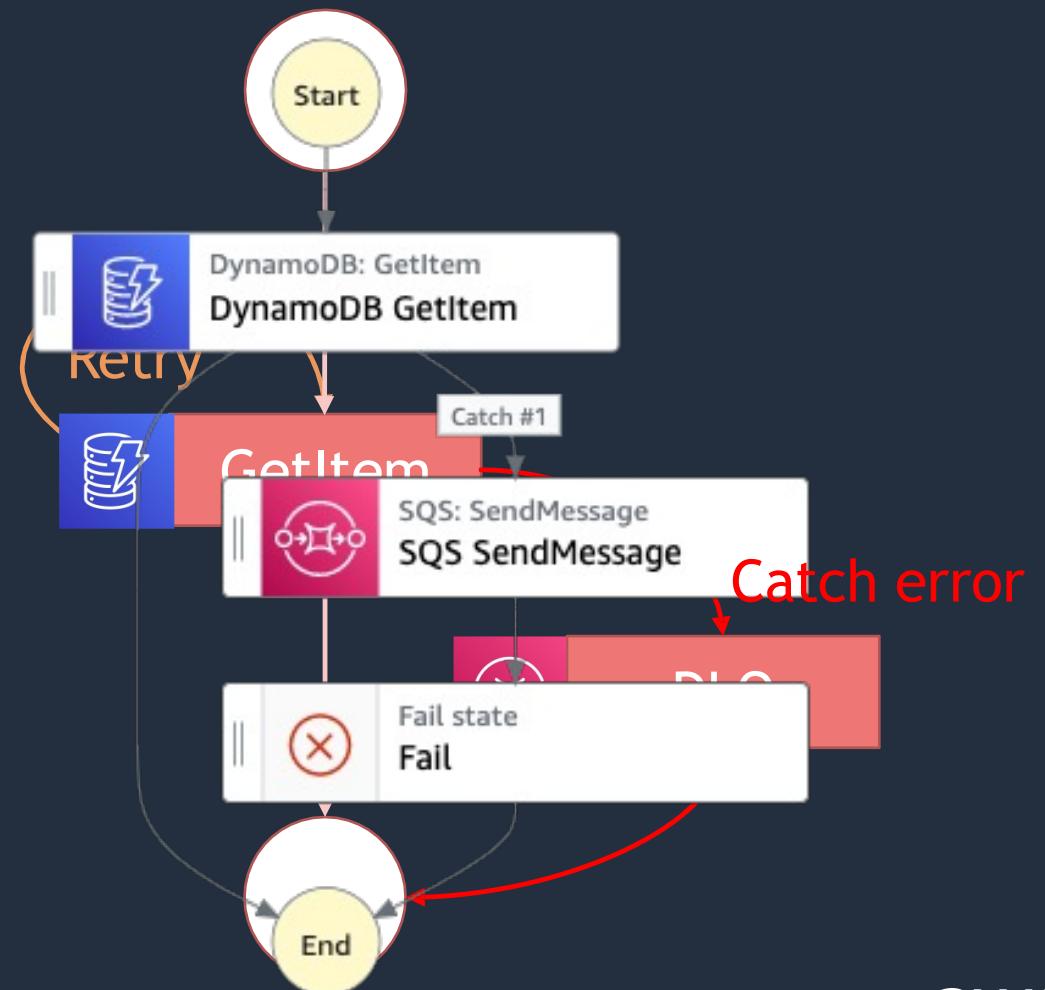
```
const AWS = require('aws-sdk');
const docClient = new AWS.DynamoDB.DocumentClient();

var params = {
  "TableName": "reinvent2022!",
  "Key": {
    "PK": {"S": "wardrobe"},
    "SK": {"S": "shoes"}
  }
}

async function queryItems(){
  try {
    const data = await docClient.getItem(params).promise()
    return data
  } catch (err) {
    return err
  }
}

exports.handler = async (event, context) => {
  try {
    const data = await queryItems()
    return { body: JSON.stringify(data) }
  } catch (err) {
    return { error: err }
  }
}
```

Incluso un «flujo de trabajo» de una sola tarea añade valor gracias a la gestión integrada de errores, la detección, el reinicio, la observabilidad, la reducción del código personalizado y el registro centralizado de cada carga de trabajo



Del código a un flujo de trabajo

Executions (10)

C View details Stop execution Start execution

Search for executions Filter by status < 1 >

Name	Status	Started	End Time
74a40347-bdb5-6655-d925-7608442c8d88	Failed	Sep 21, 2022 04:37:22.146 PM	Sep 21, 2022 04:37:22.647 PM
76180c66-a9d9-e986-3f78-0d192d284df3	Failed	Sep 21, 2022 04:37:11.989 PM	Sep 21, 2022 04:37:12.059 PM
b3195b78-3127-4ada-c1cd-4b1d5324f9b1	Failed	Sep 21, 2022 04:35:55.988 PM	Sep 21, 2022 04:35:56.075 PM
b1a392c2-cd0f-c5a2-a0e2-a63310bbc735	Failed		
8e0fdc23-6074-be1f-5afa-6a3633b33750	Succeeded		
813c077a-5bb5-7a98-71e7-59f76770c4ee	Succeeded		
d8c43467-a5f1-c658-ccf0-ae4f08b045ca	Succeeded		
478a9a01-d77d-f8fc-66d6-bcde36f932e0	Succeeded		
d9f66115-8528-43fe-a0b5-5bee5abe101e	Succeeded		
e679eed1-44d7-4a18-99f0-ca80bc571adb	Succeeded		

Graph view Data flow simulator Export Layout

```
graph TD; Start((Start)) --> Get[DynamoDB GetItem]; Get --> Send[SQS SendMessage]; Send --> End((End)); Send --> Fail((Fail))
```

In progress Failed Caught error Canceled Succeeded

Input & Output Details Definition Events

Advanced view

Input Learn more

```
1 {  
2   "item": "shoes"  
3 }
```

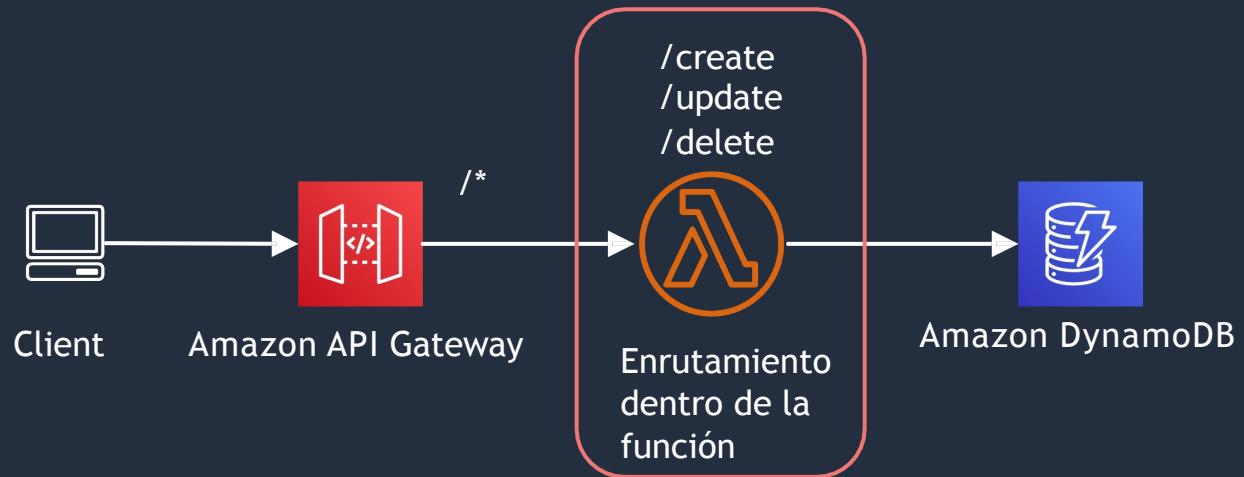
Output Learn more

```
1 {  
2   "Error": "DynamoDB.AmazonDynamoDBException",  
3   "Cause": "1 validation error detected: Value 'reinvent2022!' at 'tableName' failed to satisfy regular expression pattern: [a-zA-Z0-9_.-]+ (Service: AmazonDynamoDBv2; Status Code: ValidationException; Request ID: 8Q80L4JJQSQ5A38S9G66GJ5NPNVV4KQNSO5AEMVJF66Q9ASUAQ)"  
4 }
```

© 2023, Amazon Web Services, Inc. or its Affiliates.

aws

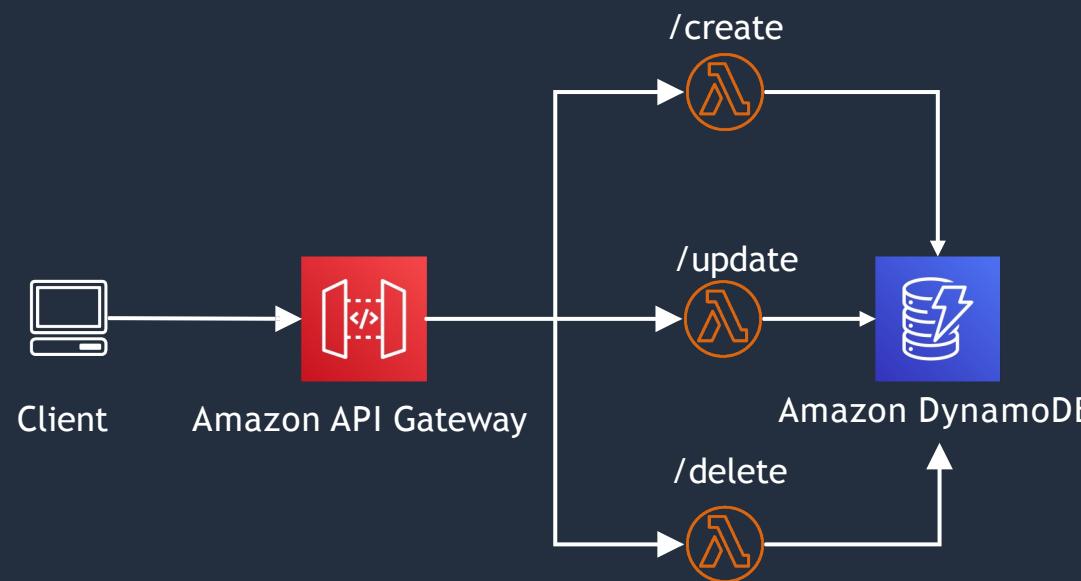
Rompiendo en partes un “Lambda-lith”



Amazon API Gateway dirige todas las solicitudes a una única función de Lambda que ejecuta el código correspondiente en función de la configuración de su ruta.

- Permisos de seguridad aplicados a todo
- Configuración de rendimiento aplicada a todo
- Los límites de duración y espacio se aplican a todo

Micro Lambda



App-delete.js

```
const AWS = require('aws-sdk');
const docClient = new AWS.DynamoDB.DocumentClient();
```

App-update.js

```
const AWS = require('aws-sdk');
const docClient = new AWS.DynamoDB.DocumentClient();
```

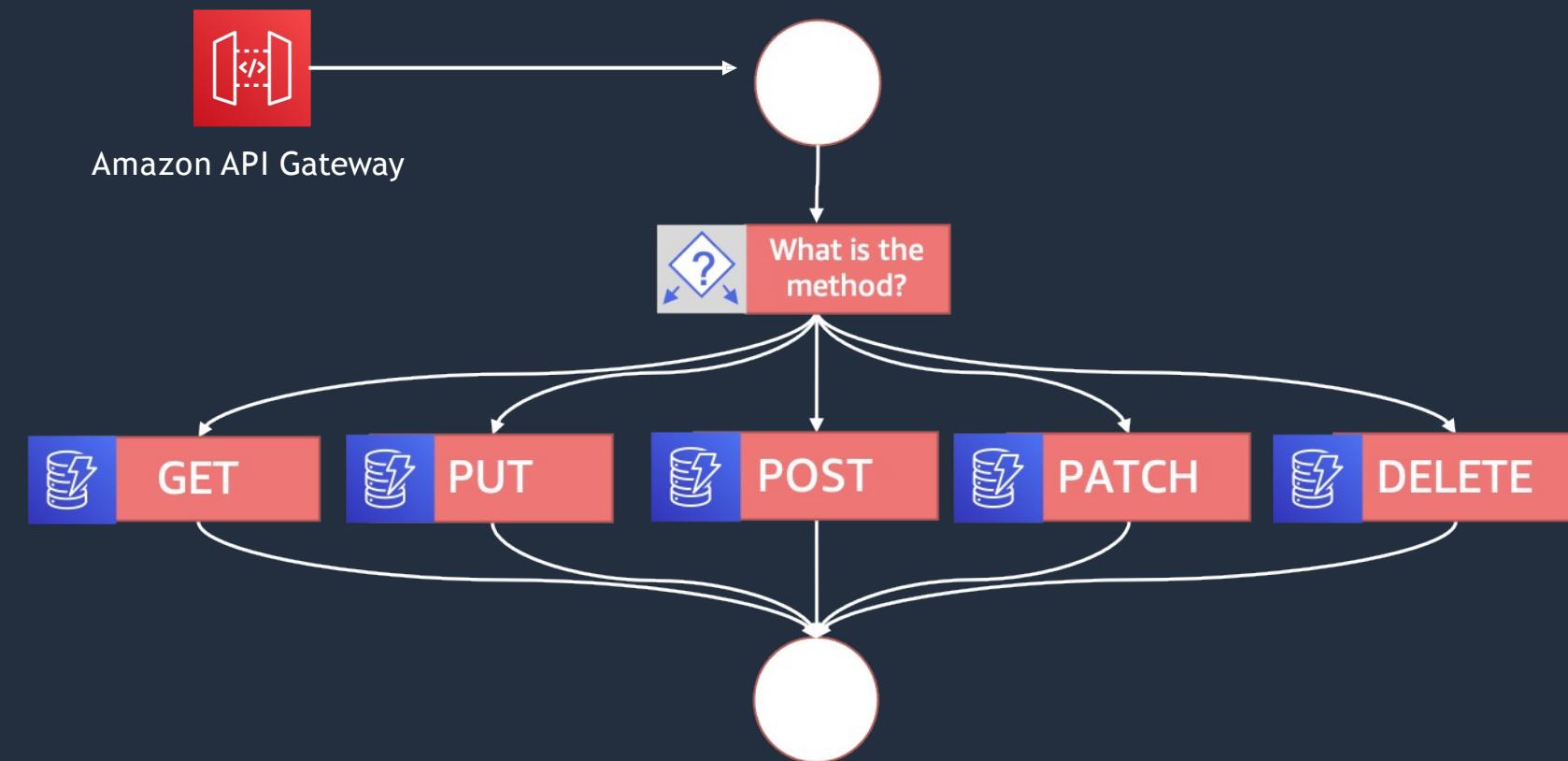
App-query.js

```
const AWS = require('aws-sdk');
const docClient = new AWS.DynamoDB.DocumentClient();
```

```
var params = {
    TableName: 'your-table-name',
    IndexName: 'some-index',
    KeyConditionExpression: '#name = :value',
    ExpressionAttributeValues: { ':value': 'shoes' },
    ExpressionAttributeNames: { '#name': 'name' }
};

async function queryItems(){
    try {
        const data = await docClient.query(params).promise();
        return data;
    } catch (err) {
        return err;
    }
}

exports.handler = async (event, context) => {
    try {
        const data = await queryItems();
        return { body: JSON.stringify(data) };
    } catch (err) {
        return { error: err };
    }
}
```



“REST” fácil

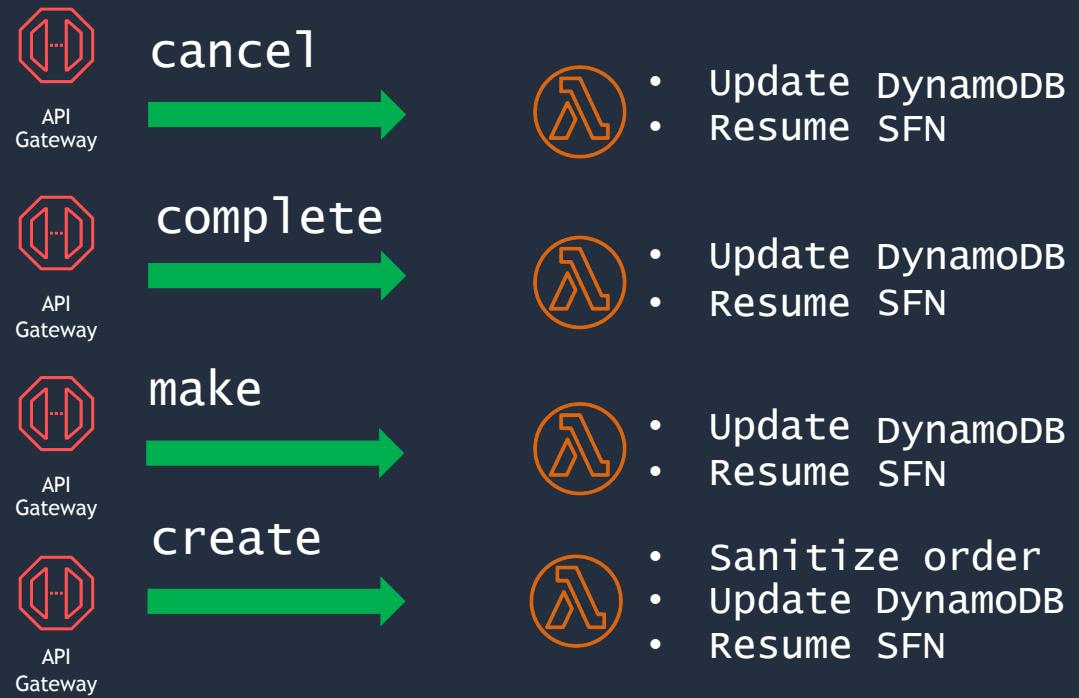
Combínelo con API Gateway y los flujos de trabajo express sincrónicos para crear un backend de API escalable y de baja latencia



Servicio de gestión de pedidos como función Lambda

Versión 1

Cada operación invoca una función Lambda



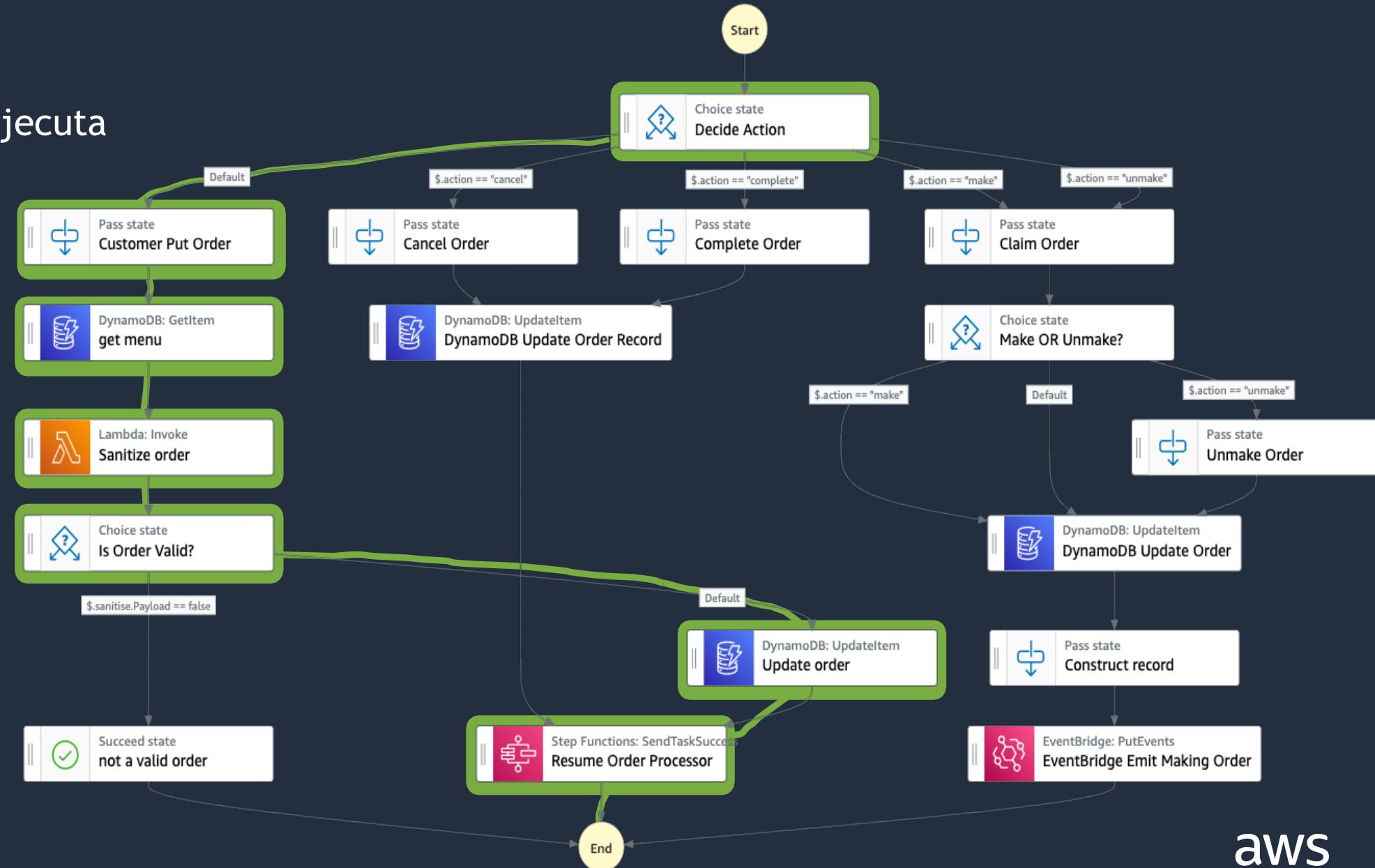
La aplicación se hizo más compleja con el tiempo y realizó múltiples tareas para gestionar una lógica empresarial cada vez más compleja, lo que llevó a:

- Una base de código estrechamente acoplada
- Cadencia de publicación más lenta
- Baja capacidad de descubrimiento
- Complejidad adicional

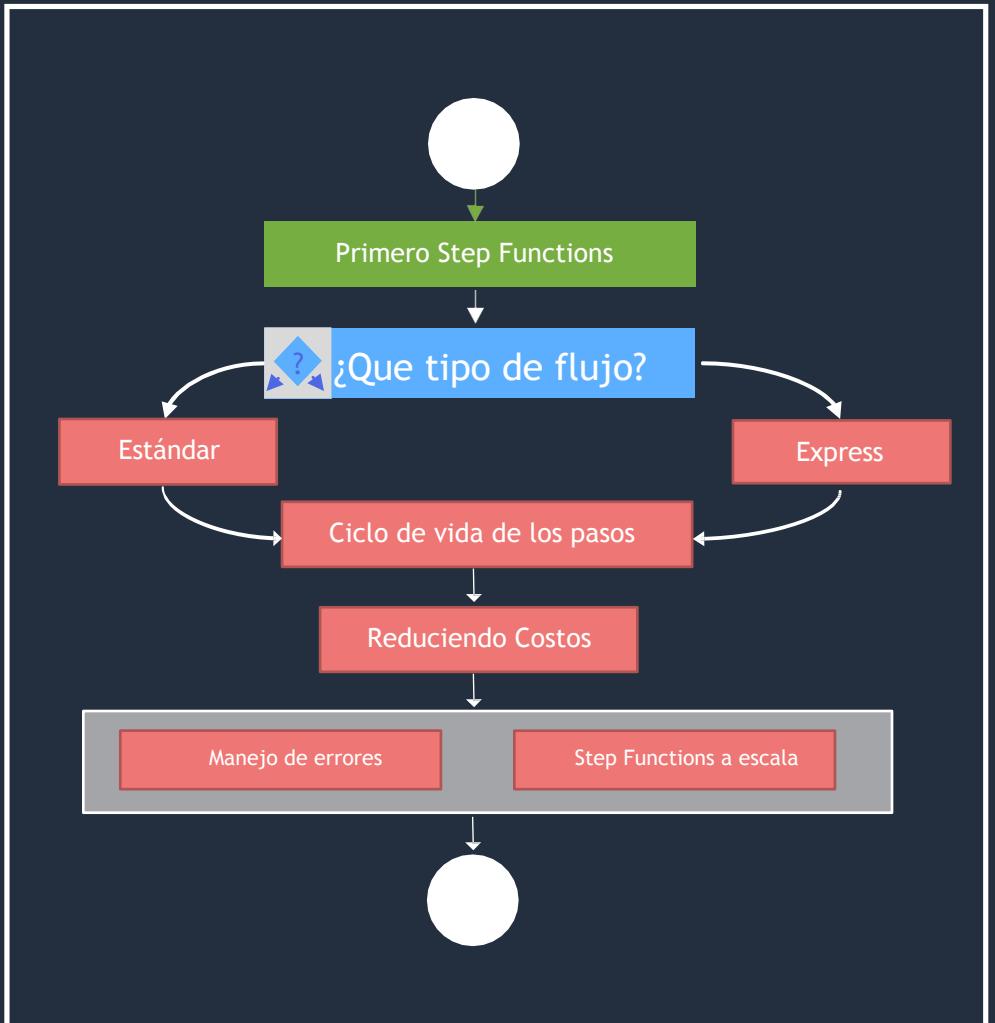
Servicio de gestión de pedidos como un flujo de trabajo

Versión 2

Un único punto final de API Gateway ejecuta un flujo de trabajo de Step Functions



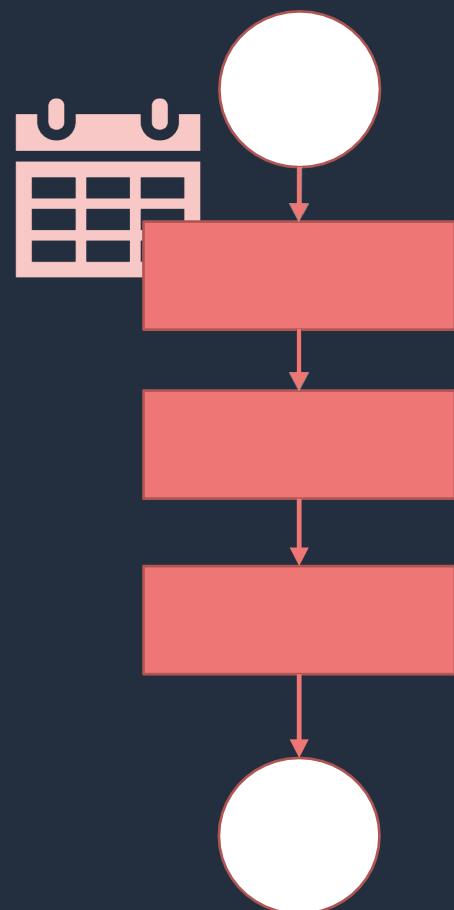
Seleccionando flujos de trabajo estándar o express



Estándar

Express

VS



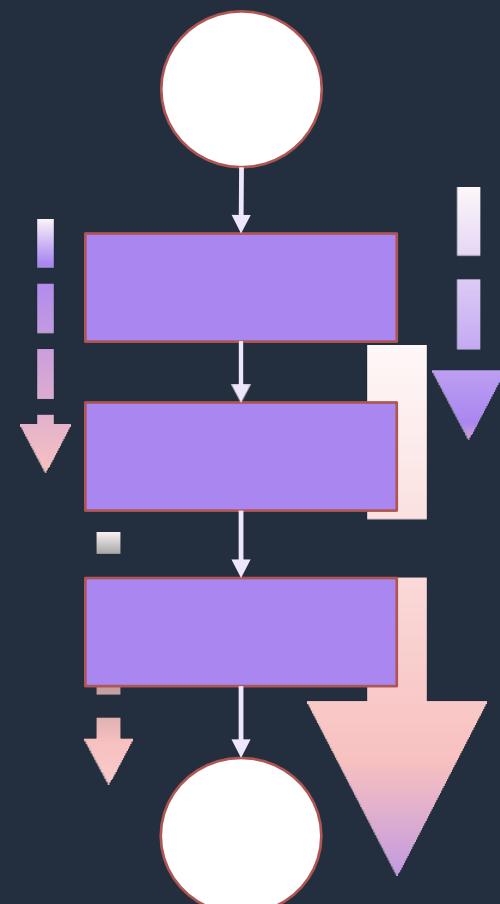
Estándar

- De larga duración
- Asincrónico
- Exactamente una vez

VS

Express

- Alto rendimiento
- Rentable
- Al menos una vez
- Sincrónico
- Asincrónico
- Corta duración



Ejemplo de la vida real

Un flujo de trabajo de un comercio electrónico

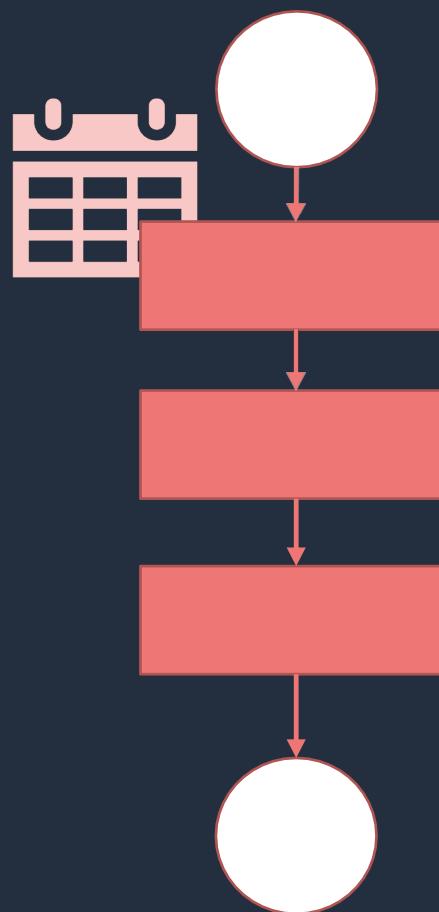
1000 como estándar

1000 como express

Creo un panel de Amazon CloudWatch para mostrar los tiempos de ejecución promedio



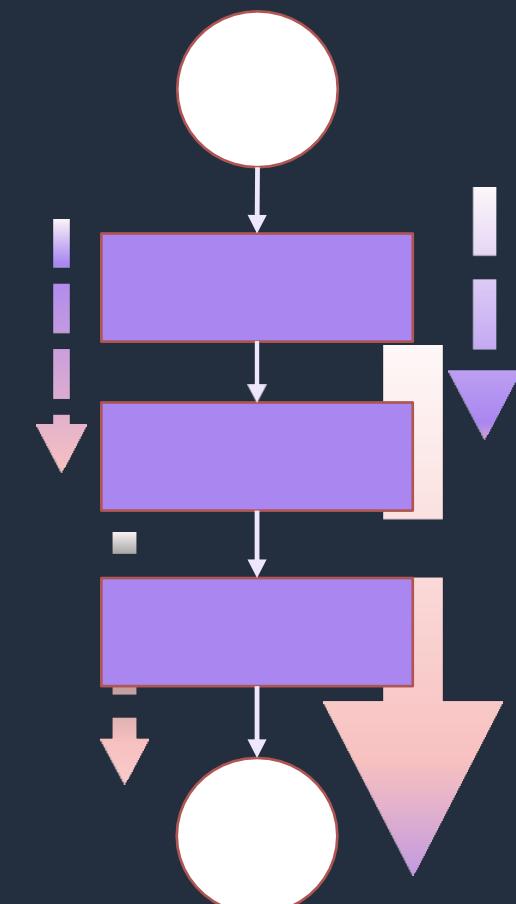
Duración promedio



Estándar
11,9s

VS

Express
11,3s



Costo por cada mil ejecuciones



Costo por cada mil ejecuciones

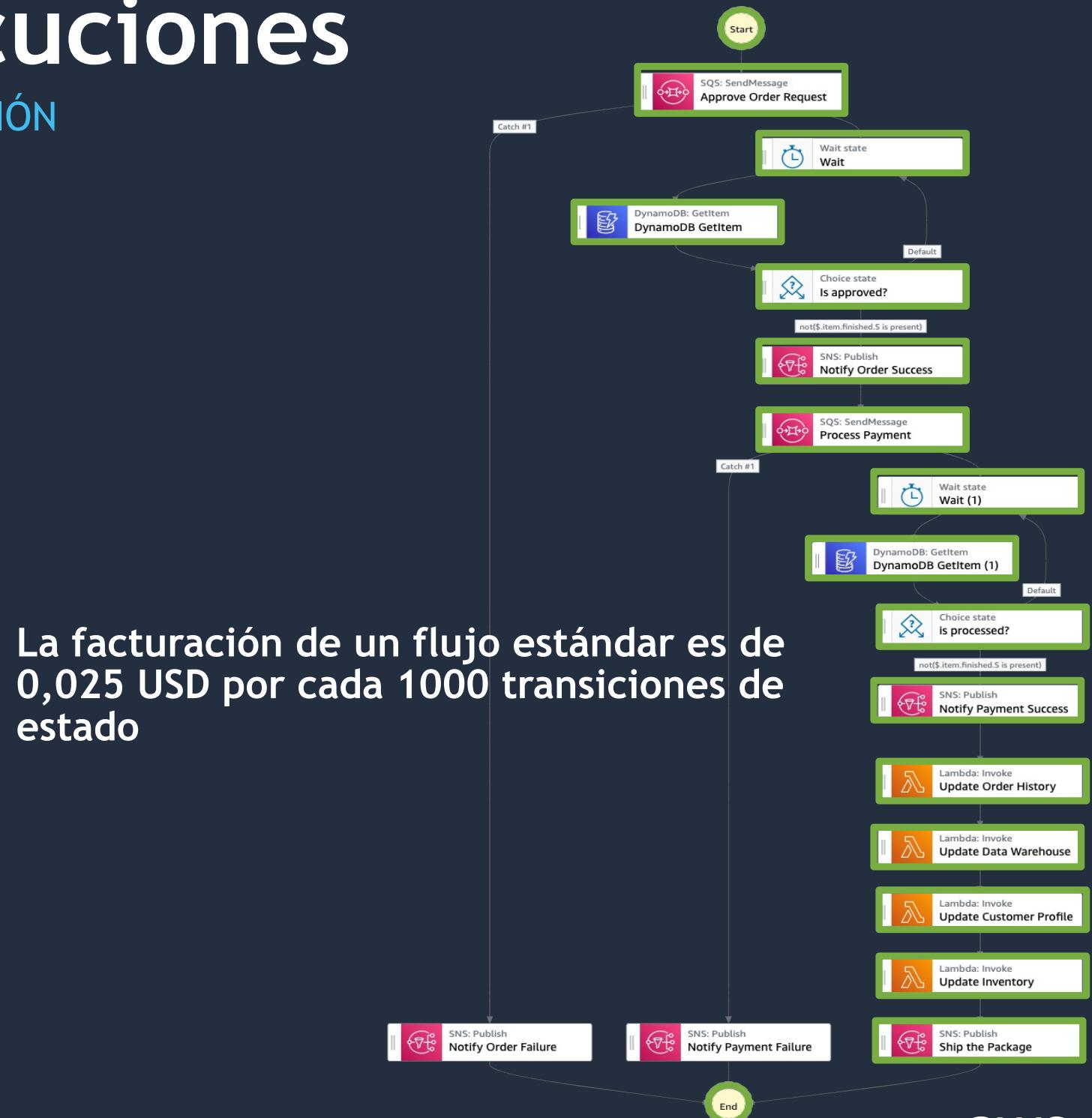
COBRO EN FUNCIÓN DEL NÚMERO DE ESTADOS DE TRANSICIÓN

Costo total de un flujo de trabajo estándar

transiciones x (# de ejecuciones x \$0.000025)

Costo total = $17 \times 0.025 = \$0.42^*$ USD

Costo total = \$0,42 USD por cada 1000 ejecuciones



Costo por cada mil ejecuciones

LOS FLUJOS DE TRABAJO EXPRÉS COBRAN EN FUNCIÓN DEL NÚMERO DE SOLICITUDES Y LA DURACIÓN

Costo total de un flujo de trabajo express

(Costo de ejecución + Costo de duración) x número de solicitudes

Costo de duración = duración de ejecución ms/100 * costo de memoria



Costo por cada mil ejecuciones

FLUJOS DE TRABAJO EXPRÉS PRECIO APROXIMADO POR 100 MS PARA DIFERENTES TAMAÑOS DE MEMORIA

Memoria (MB)	Precio por cada 100 ms (\$) para los primeros 1000 GB-hora	Precio por 100 ms (\$) para los próximos 4.000 GB-hora	Precio por cada 100 ms (\$) de GB/hora adicional
64	0.0000001042	0.0000000521	0.0000000285
128	0.0000002083	0.0000001042	0.0000000570
192	0.0000003125	0.0000001563	0.0000000856
256	0.0000004167	0.0000002083	0.0000001141
Cada 64 MB adicionales	0.0000001042	0.0000000521	0.0000000285

Página de precios de Step Functions: <https://aws.amazon.com/step-functions/pricing/>

Costo por cada mil ejecuciones

LOS FLUJOS DE TRABAJO EXPRÉS COBRAN EN FUNCIÓN DEL NÚMERO DE SOLICITUDES Y LA DURACIÓN

Costo total de un flujo de trabajo express

(Costo de ejecución + Costo de duración) x número de solicitudes

Costo de duración = (11.300 ms / 100) *
0,00001042 USD = 0,0000117746 USD

Costo de ejecución = 0,000001 USD por solicitud
(0,000001 USD + 0,0000117746 USD) x 1000 = 0,01 USD

Los flujos de trabajo express cobran en función del **número de solicitudes** y la **duración** (hasta los 100 ms más cercanos)



Costo de flujo de trabajo estándar



Estándar
1 millón de ejecuciones = **\$420**

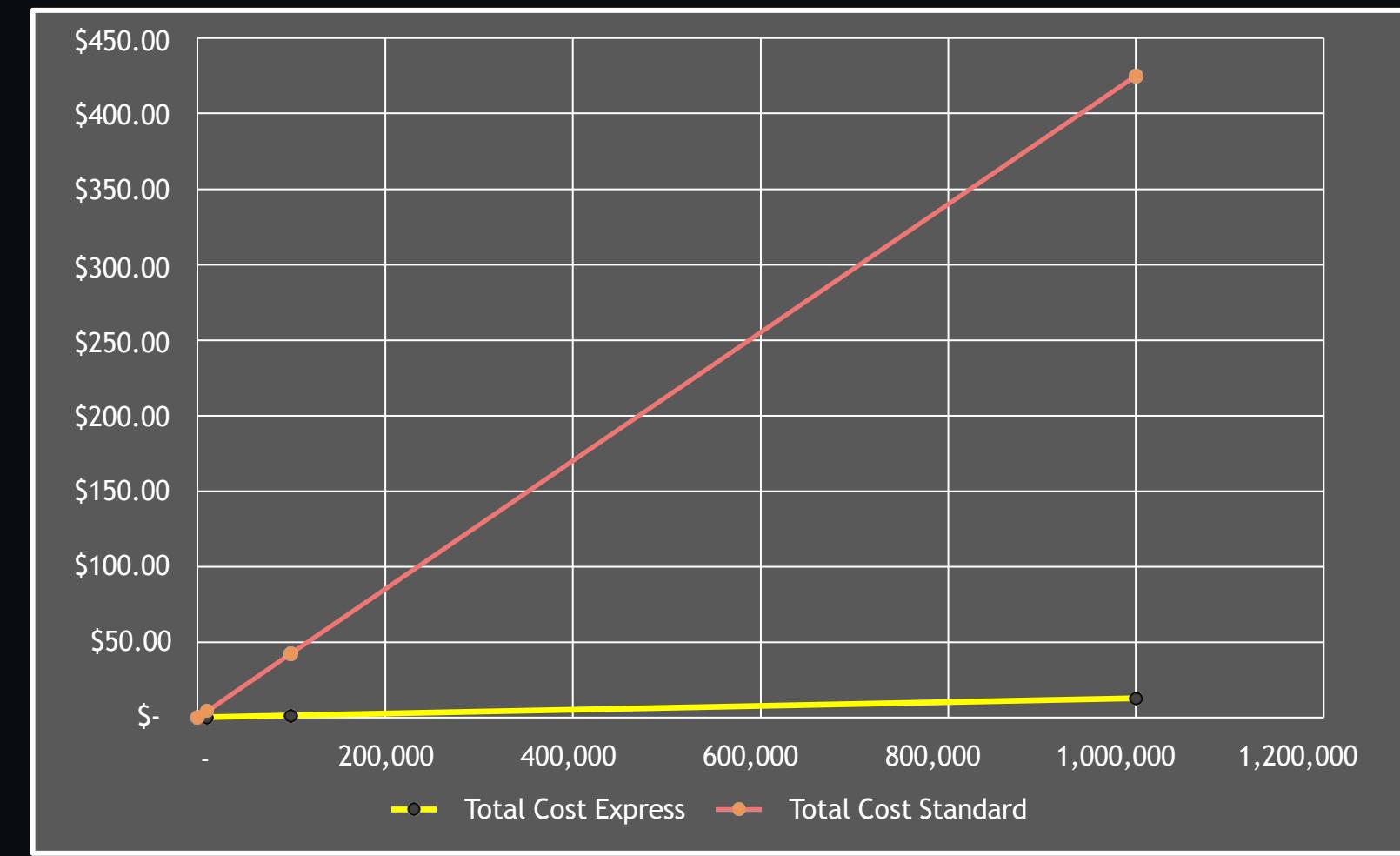
VS

1 millón de ejecuciones = **\$12.77**
Express

Costo de flujo de trabajo express



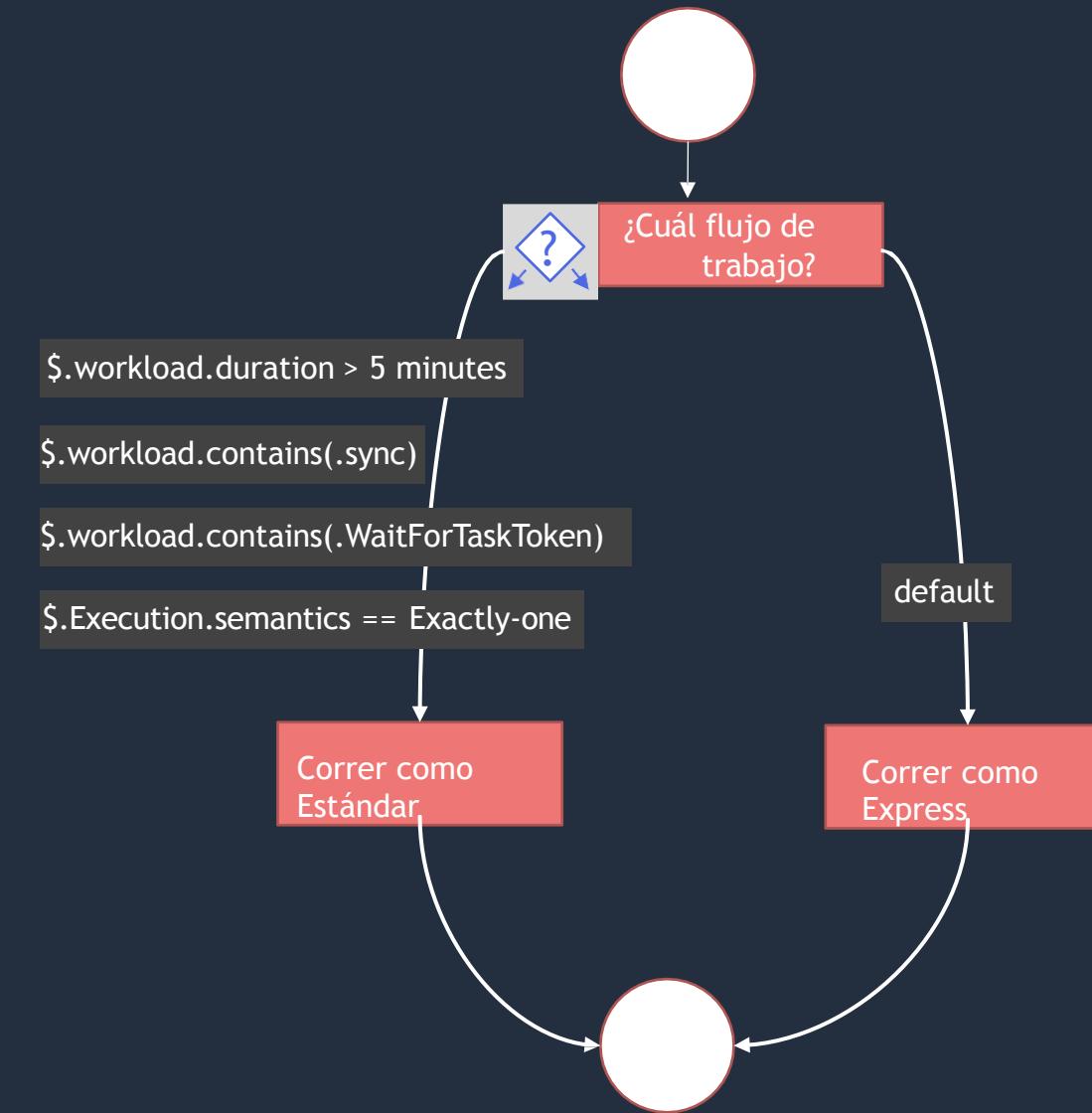
Flujo de trabajo Estándar vs Express: 1 millón de ejecuciones



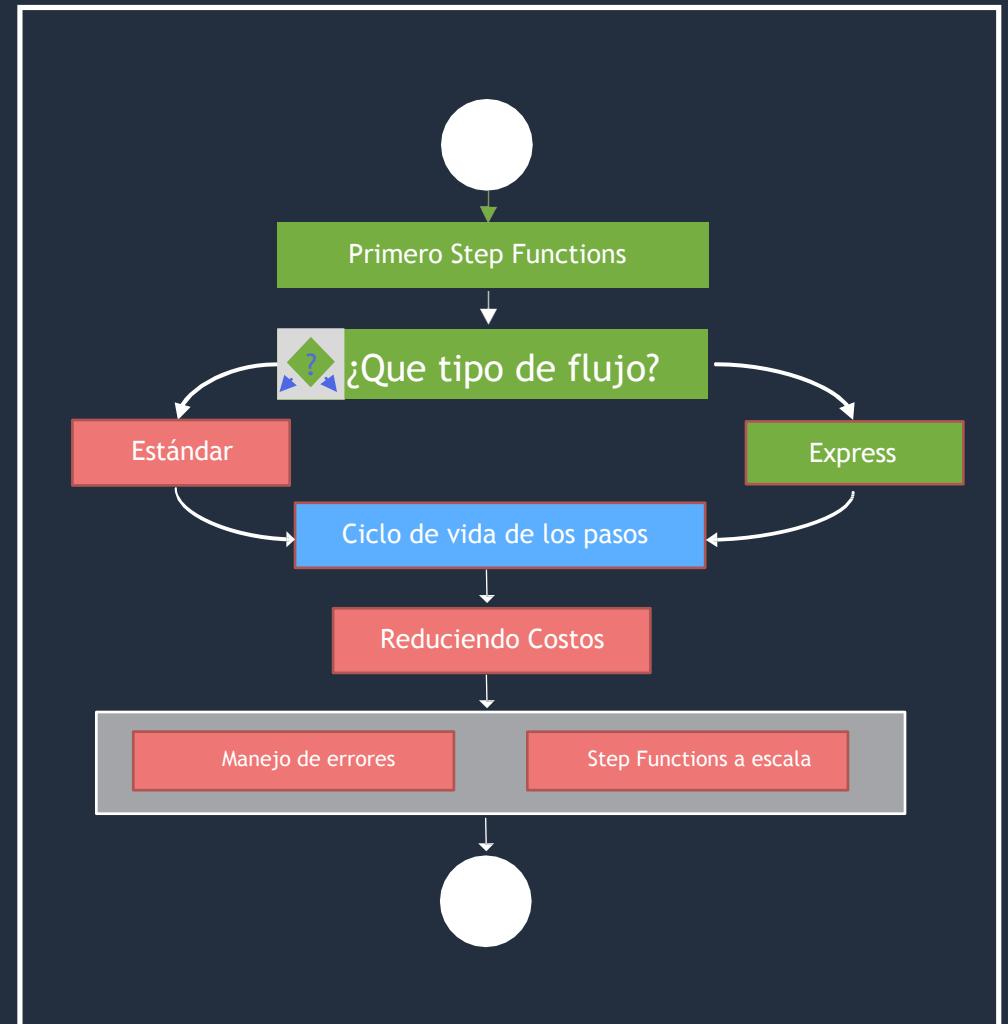
Entonces, ¿por qué elegir Estándar?

Cuando la carga de trabajo

- Se demora más de 5 minutos
- Usa los patrones Callback (.waitForTaskToken) o .Sync
- Requiere una ejecución exacta



Ciclo de vida de los pasos



Todo empieza con un evento...

```
{  
    "SimpleKey": "SimpleValue",  
    "ListKey": [  
        "List value one",  
        "List value two"  
    ],  
    "ObjectKey": {  
        "NestedKey": "Nested value"  
    }  
}
```

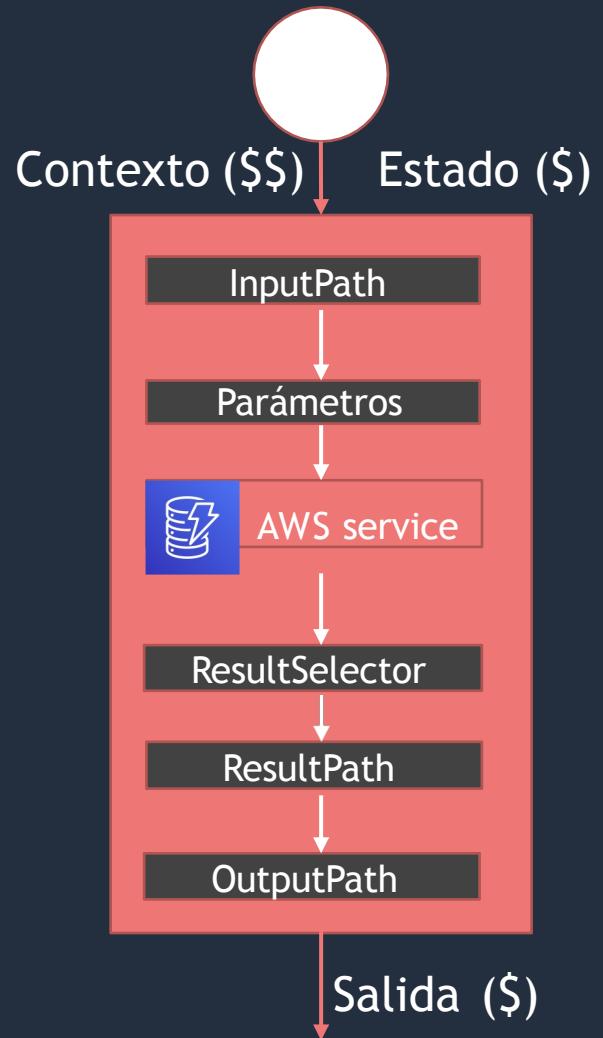
...y un poco de contexto

```
{  
  "Execution": {  
    "Id": "String",  
    "Input": {},  
    "StartTime": "Format: ISO 8601"  
  },  
  "State": {  
    "EnteredTime": "Format: ISO 8601",  
    "Name": "String",  
    "RetryCount": Number  
  },  
  "StateMachine": {  
    "Id": "String"  
  },  
  "Task": {  
    "Token": "String"  
  }  
}
```

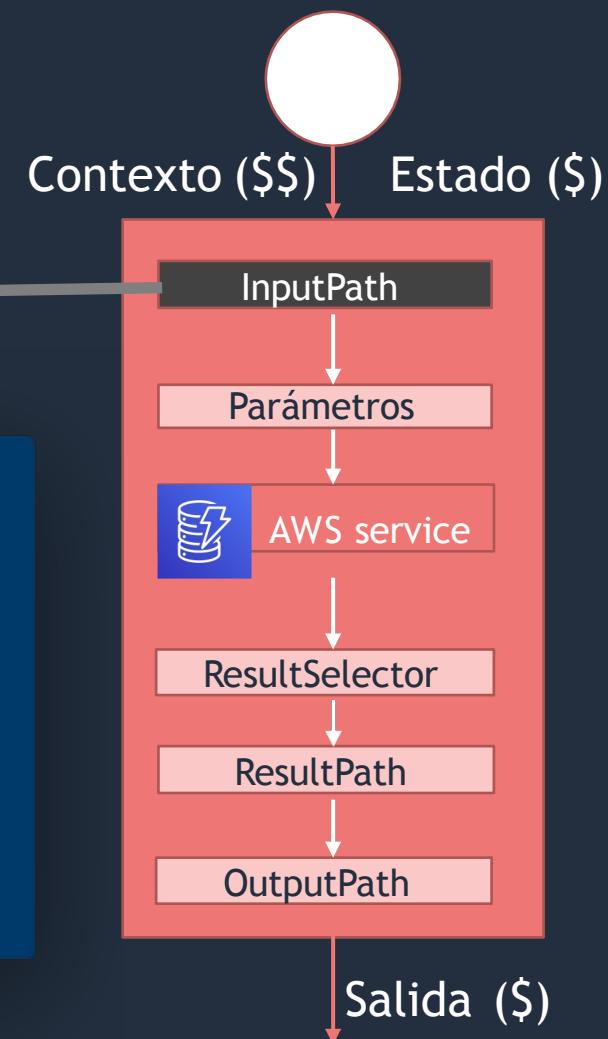
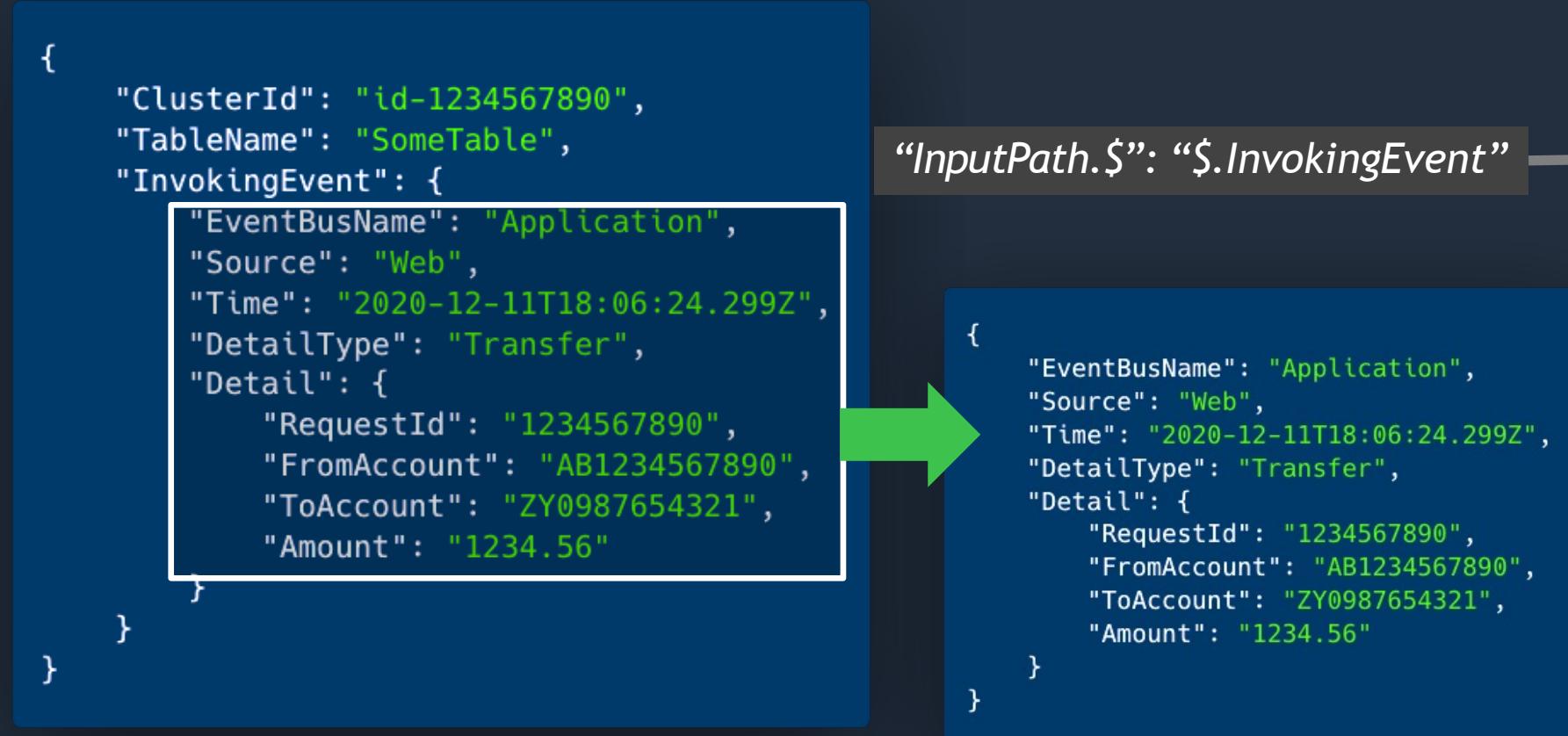
```
{  
  "SimpleKey": "SimpleValue",  
  "ListKey": [  
    "List value one",  
    "List value two"  
  ],  
  "ObjectKey": {  
    "NestedKey": "Nested value"  
  }  
}
```

Orden de operaciones

- `InputPath`
- `Parameters`
- Estado de la Tarea
- `ResultSelector`
- `ResultPath`
- `OutputPath`



“InputPath”: “\$.InvokingEvent”



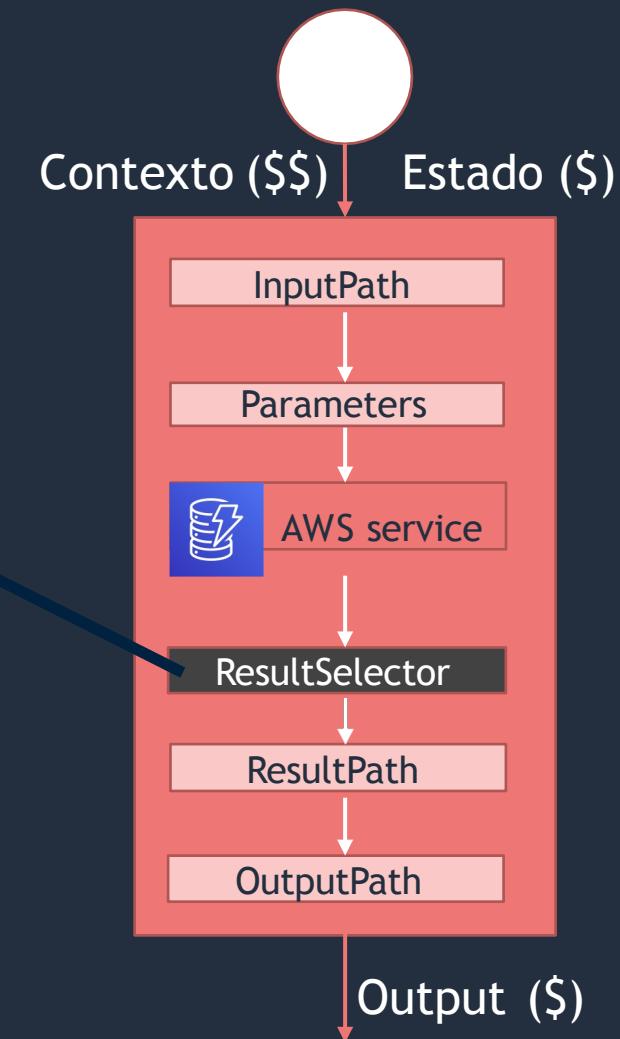
Estado de la Tarea



ResultSelector

```
Result Selector:  
{  
    "StatusCode.$": "$.SdkHttpMetadata.HttpStatusCode",  
    "RequestId.$": "$.SdkResponseMetadata.RequestId",  
    "RequestDate.$": "$.SdkHttpMetadata.HttpHeaders.Date"
```

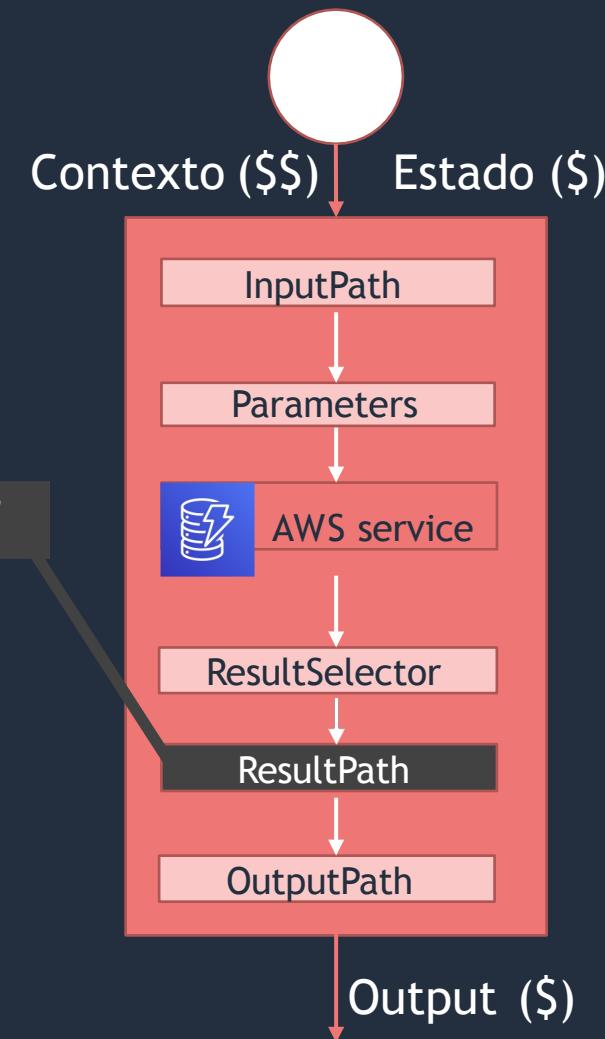
```
{  
    "StatusCode": 200,  
    "RequestId": "01edf8c3-fadd-4368-b140-75026b7deae9",  
    "RequestDate": "Fri, 11 Dec 2020 19:46:29 GMT"  
}
```



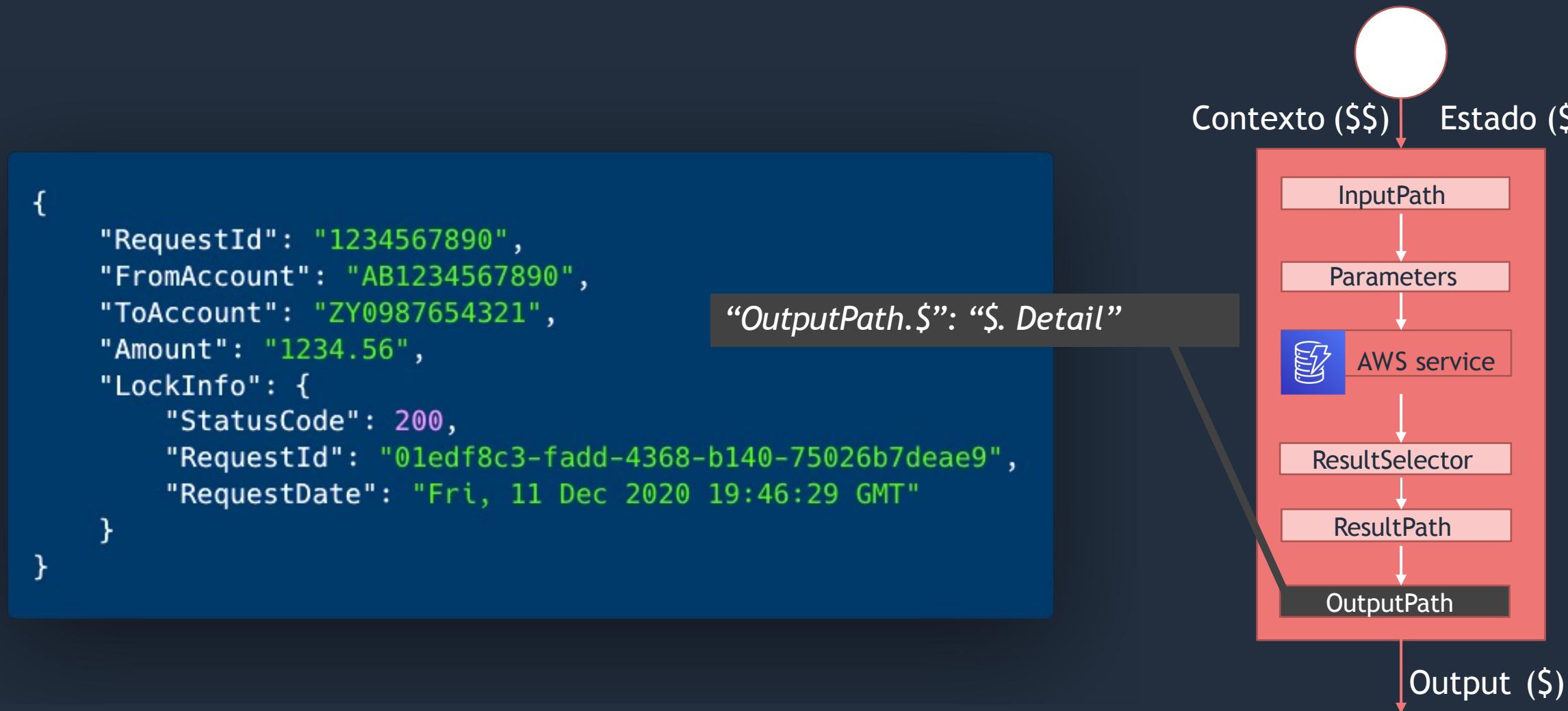
“ResultPath”: “\$.Detail.LockInfo”

```
{  
    "EventBusName": "Application",  
    "Source": "Web",  
    "Time": "2020-12-11T18:06:24.299Z",  
    "DetailType": "Transfer",  
    "Detail": {  
        "RequestId": "1234567890",  
        "FromAccount": "AB1234567890",  
        "ToAccount": "ZY0987654321",  
        "Amount": "1234.56",  
        "LockInfo": {  
            "StatusCode": 200,  
            "RequestId": "01edf8c3-fadd-4368-b140-75026b7deae9",  
            "RequestDate": "Fri, 11 Dec 2020 19:46:29 GMT"  
        }  
    }  
}
```

“ResultPath.\$”: “\$.Detail.LockInfo”

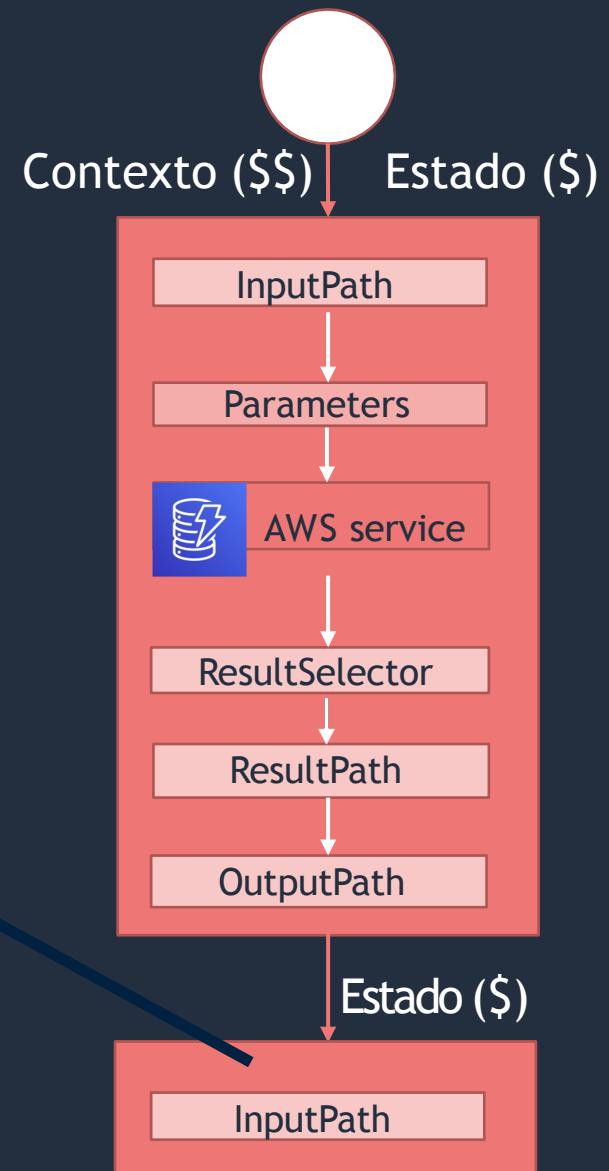


“OutputPath”: “\$.Detail”

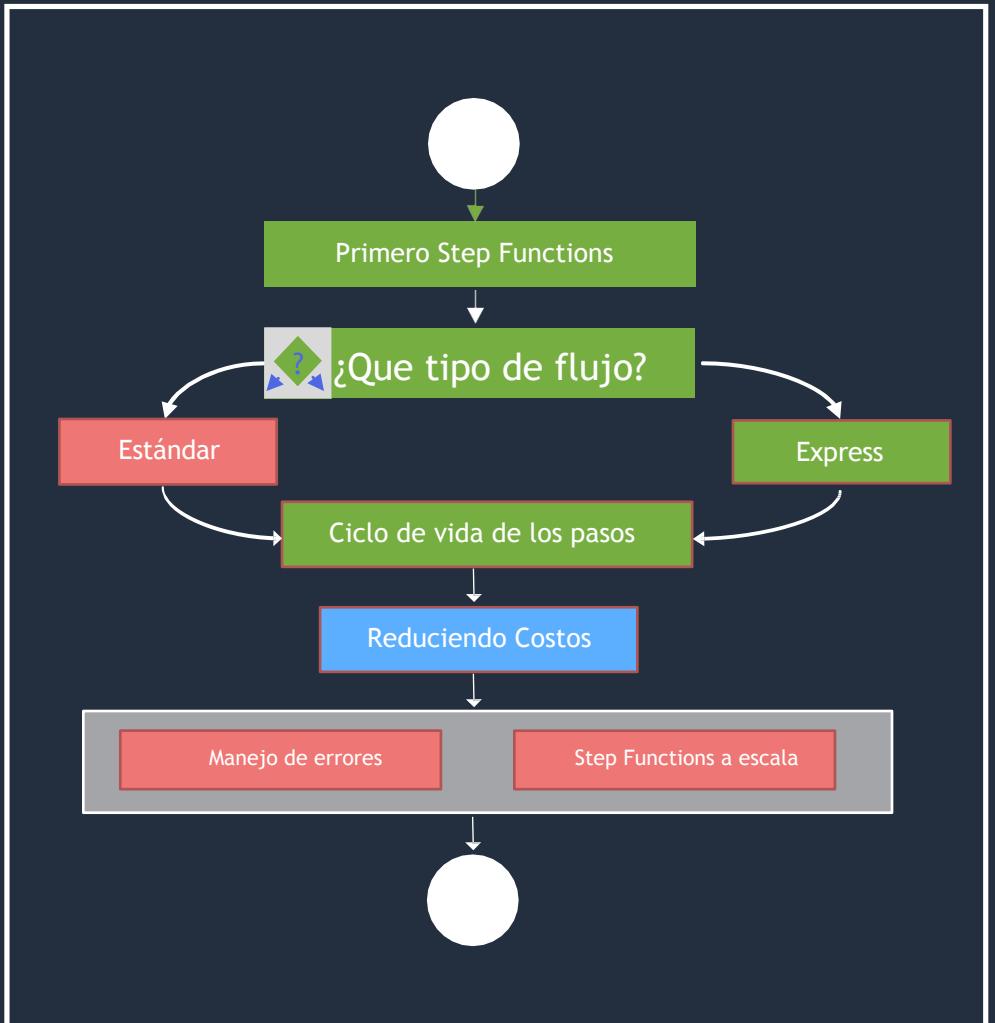


Ahora el estado se transforma

```
{  
    "RequestId": "1234567890",  
    "FromAccount": "AB1234567890",  
    "ToAccount": "ZY0987654321",  
    "Amount": "1234.56",  
    "LockInfo": {  
        "StatusCode": 200,  
        "RequestId": "01edf8c3-fadd-4368-b140-75026b7deae9",  
        "RequestDate": "Fri, 11 Dec 2020 19:46:29 GMT"  
    }  
}
```



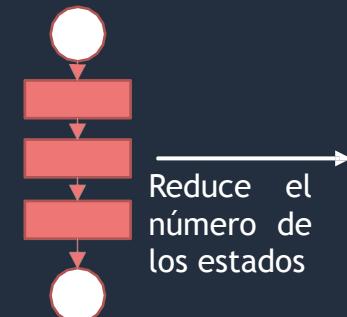
Reduciendo costos



Reduciendo costos

Estándar

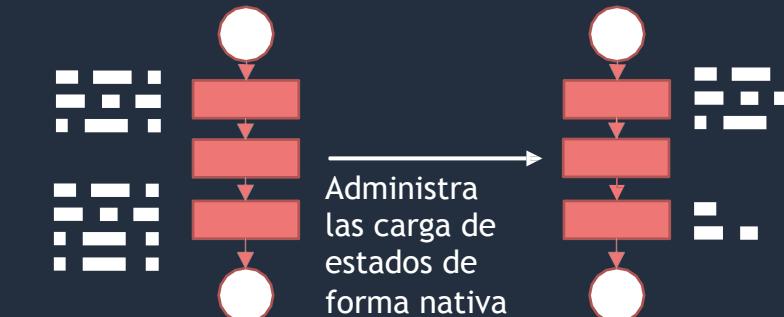
Reduce las transiciones de estado



VS

Express

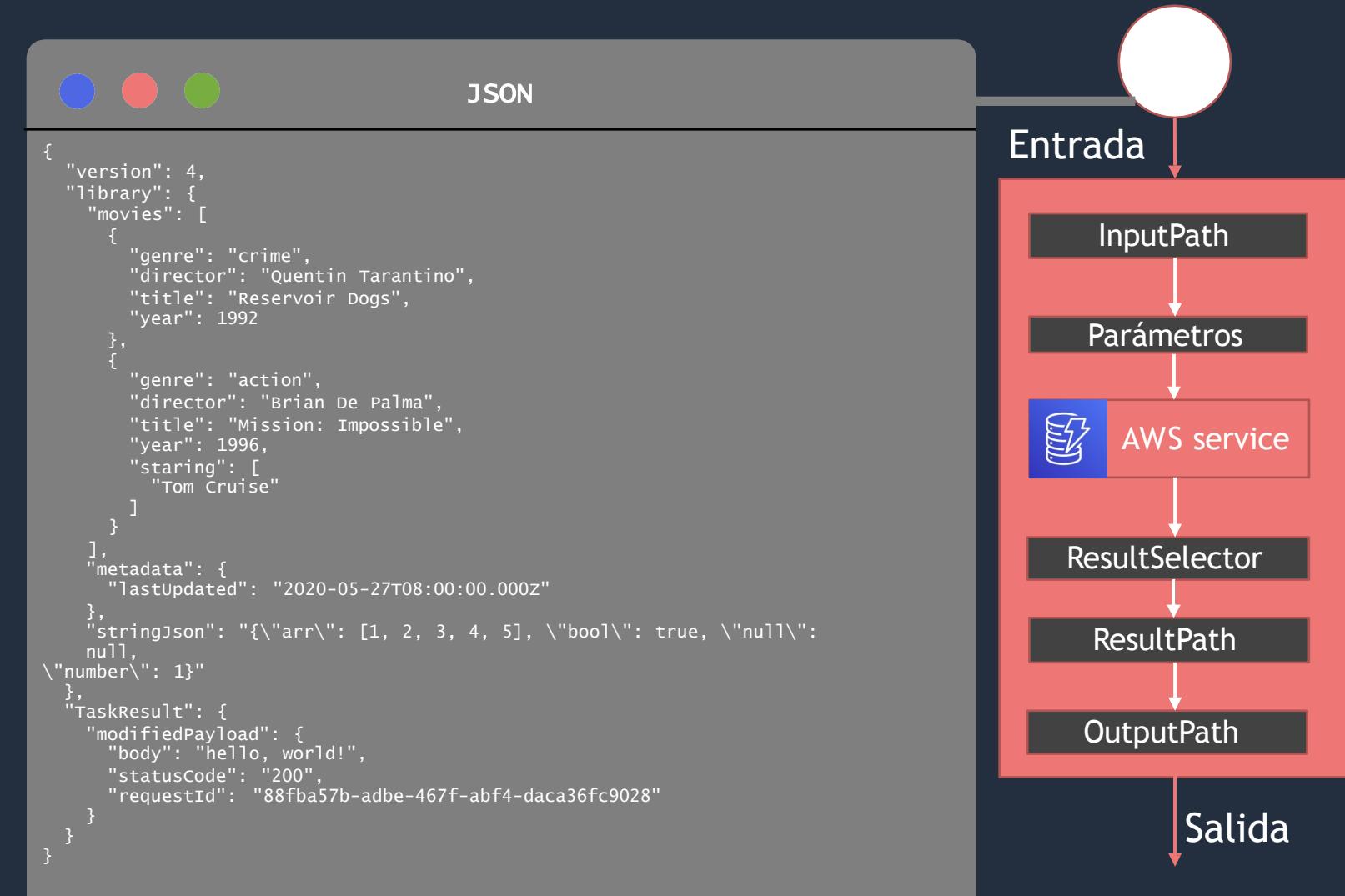
Reduce la duración de ejecución
Reduce el uso de GB de memoria



Reduciendo costos: Flujos de trabajo estándar

REDUZCA EL CÓDIGO PERSONALIZADO Y LAS TRANSICIONES DE ESTADO MEDIANTE FILTROS JSONPATH

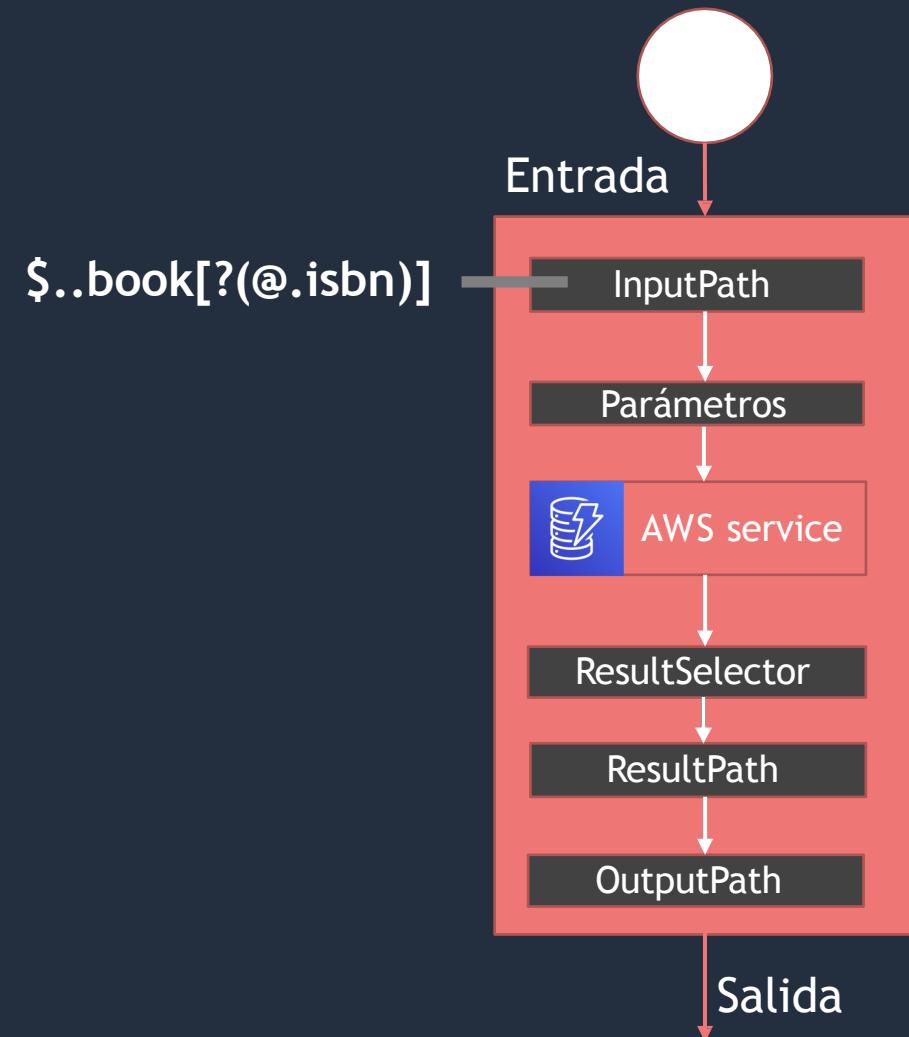
Reduzca el código personalizado y el tamaño de la carga mediante el uso de expresiones JSONPath en lugar de servicios de cómputos para filtrar las cargas de entrada



Reduciendo costos: Flujos de trabajo estándar

REDUZCA LAS TRANSICIONES PERSONALIZADAS DE CÓDIGO Y ESTADO MEDIANTE FILTROS JSONPATH

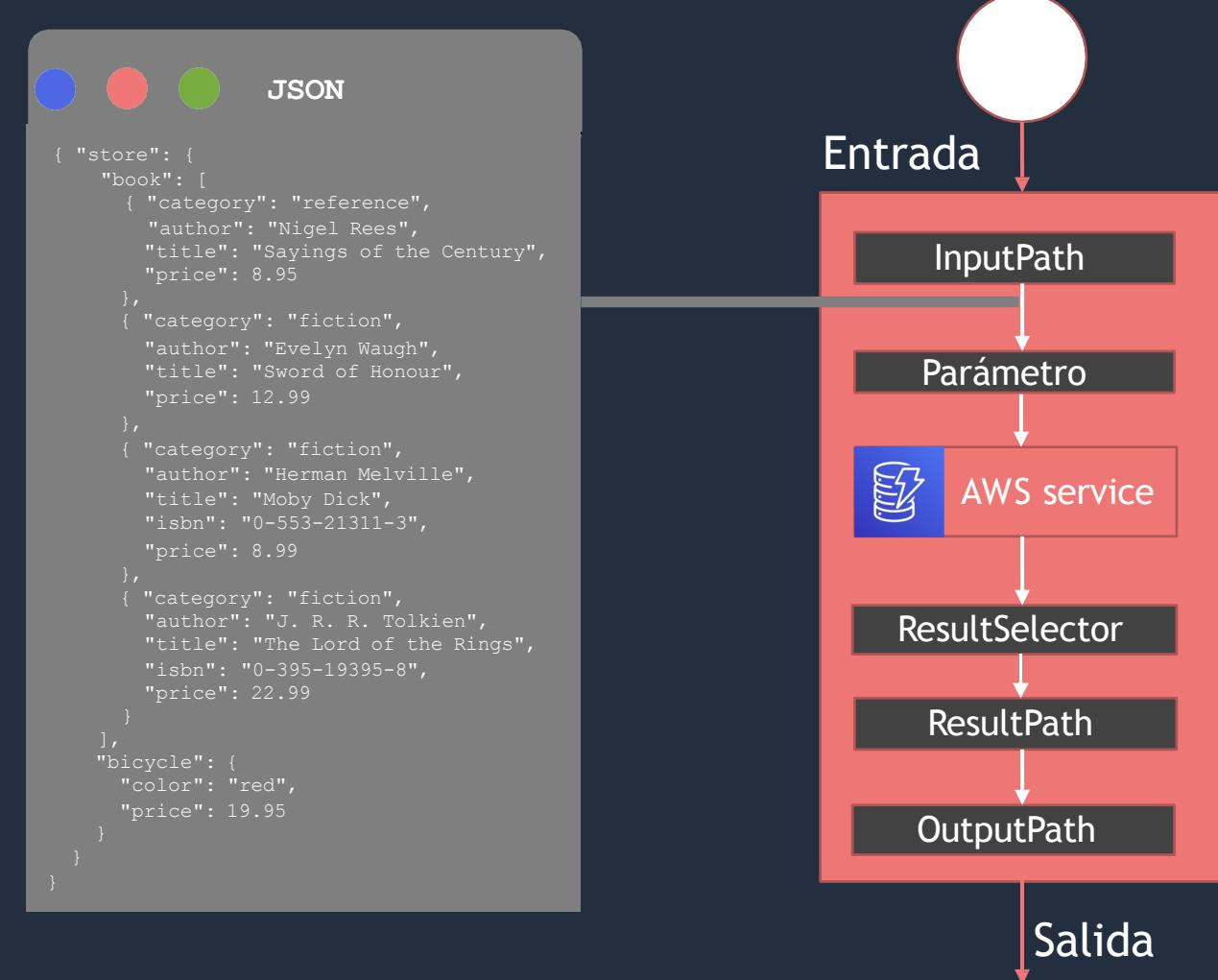
Reduzca el código personalizado y el tamaño de la carga mediante el uso de expresiones JSONPath en lugar de servicios de cómputos para filtrar las cargas de entrada



Reduciendo costos: Flujos de trabajo estándar

REDUZCA EL CÓDIGO PERSONALIZADO MEDIANTE FILTROS JSON

Reduzca el código personalizado mediante el uso de expresiones JSONPATH con el campo de InputPath



Reduciendo costos: Flujos de trabajo estándar

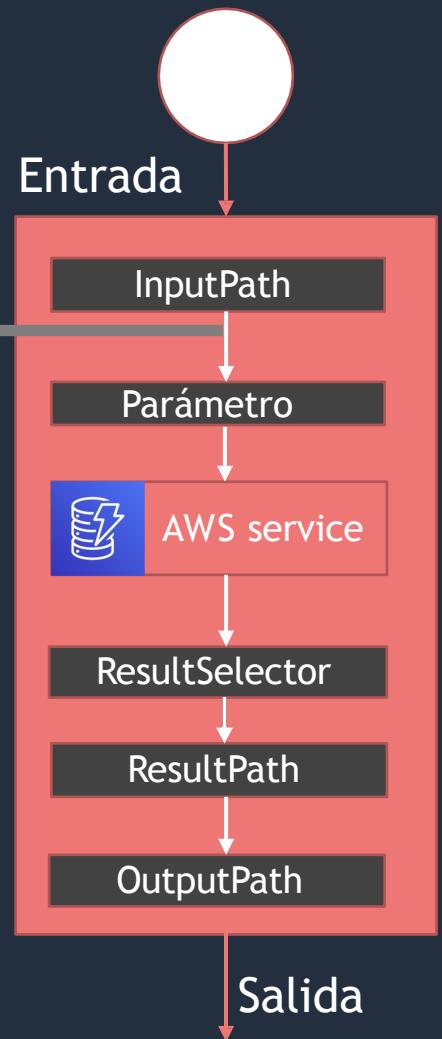
REDUZCA EL CÓDIGO PERSONALIZADO MEDIANTE FILTROS JSON

Reduzca el código personalizado mediante el uso de expresiones JSONPATH con el campo de InputPath

`$.book[2] //el 3 libro`

JSON

```
{ "store": { "book": [ { "category": "reference", "author": "Nigel Rees", "title": "Sayings of the Century", "price": 8.95 }, { "category": "fiction", "author": "Evelyn Waugh", "title": "Sword of Honour", "price": 12.99 }, { "category": "fiction", "author": "Herman Melville", "title": "Moby Dick", "isbn": "0-553-21311-3", "price": 8.99 }, { "category": "fiction", "author": "J. R. R. Tolkien", "title": "The Lord of the Rings", "isbn": "0-395-19395-8", "price": 22.99 } ], "bicycle": { "color": "red", "price": 19.95 } } }
```

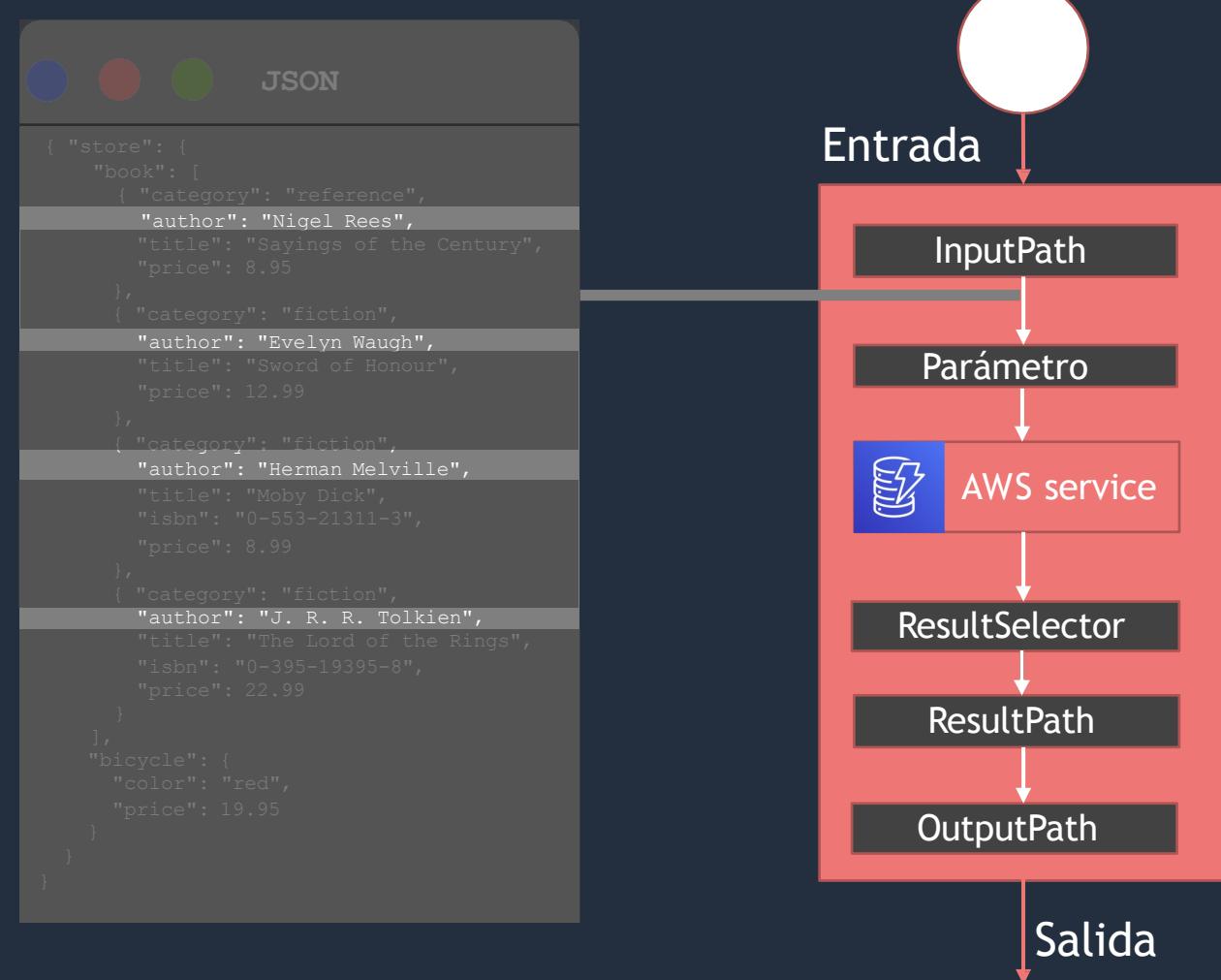


Reduciendo costos: Flujos de trabajo estándar

REDUZCA EL CÓDIGO PERSONALIZADO MEDIANTE FILTROS JSON

Reduzca el código personalizado mediante el uso de expresiones JSONPATH con el campo de InputPath

```
$.store.book[*].author  
// los autores de todos los libros en la tienda
```



Reduciendo costos: Flujos de trabajo estándar

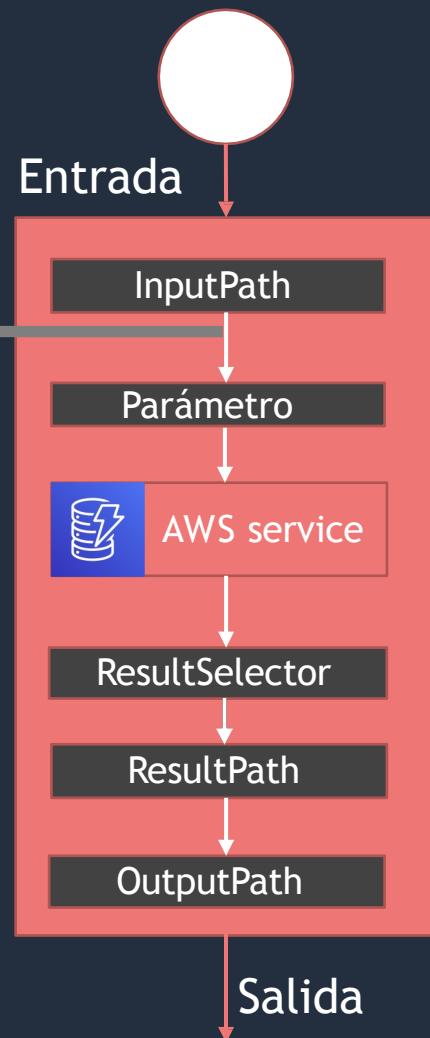
REDUZCA EL CÓDIGO PERSONALIZADO MEDIANTE FILTROS JSON

Reduzca el código personalizado mediante el uso de expresiones JSONPATH con el campo de InputPath

```
$..book[?(@.category==fiction && @.price < 10)]  
// filtra todos los libros más baratos que 10 y la  
categoría es ficción
```

JSON

```
{ "store": {  
    "book": [  
        { "category": "reference",  
          "author": "Nigel Rees",  
          "title": "Sayings of the Century",  
          "price": 8.95  
        },  
        { "category": "fiction",  
          "author": "Evelyn Waugh",  
          "title": "Sword of Honour",  
          "price": 12.99  
        },  
        { "category": "fiction",  
          "author": "Herman Melville",  
          "title": "Moby Dick",  
          "isbn": "0-553-21311-3",  
          "price": 8.99  
        },  
        { "category": "fiction",  
          "author": "J. R. R. Tolkien",  
          "title": "The Lord of the Rings",  
          "isbn": "0-395-19395-8",  
          "price": 22.99  
        }  
      ],  
      "bicycle": {  
        "color": "red",  
        "price": 19.95  
      }  
    }  
}
```

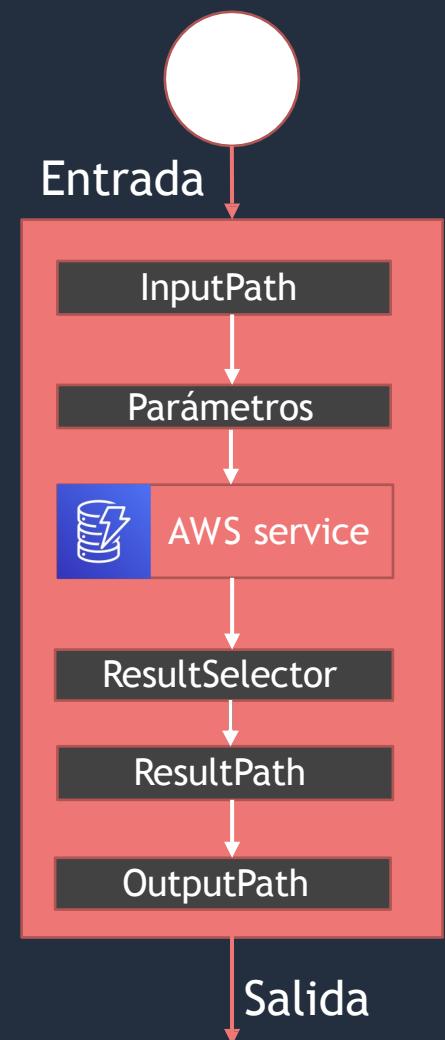


Reduciendo costos: Flujos de trabajo estándar

REDUZCA EL CÓDIGO PERSONALIZADO Y LAS TRANSICIONES DE ESTADO USANDO FUNCIONES INTRÍNSECAS

Utilice **funciones intrínsecas** en lugar de servicios de computación para realizar **transformaciones de datos simples**

Reduzca el costo de ejecución de sus flujos de trabajo al disminuir el número de estados y el número de transiciones.



AWS Step Functions: Funciones Intrínsecas



Matrices



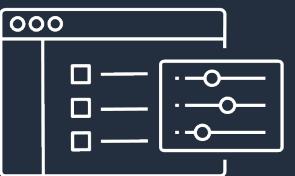
Manipulación de
datos JSON



Codificación y
decodificación



Operaciones
matemáticas



Operaciones
String



Generación de
identificadores únicos

STATES.

AWS Step Functions: Funciones Intrínsecas

REALIZAR OPERACIONES BÁSICAS DE PROCESAMIENTO DE DATOS SIN UTILIZAR UNA TAREA

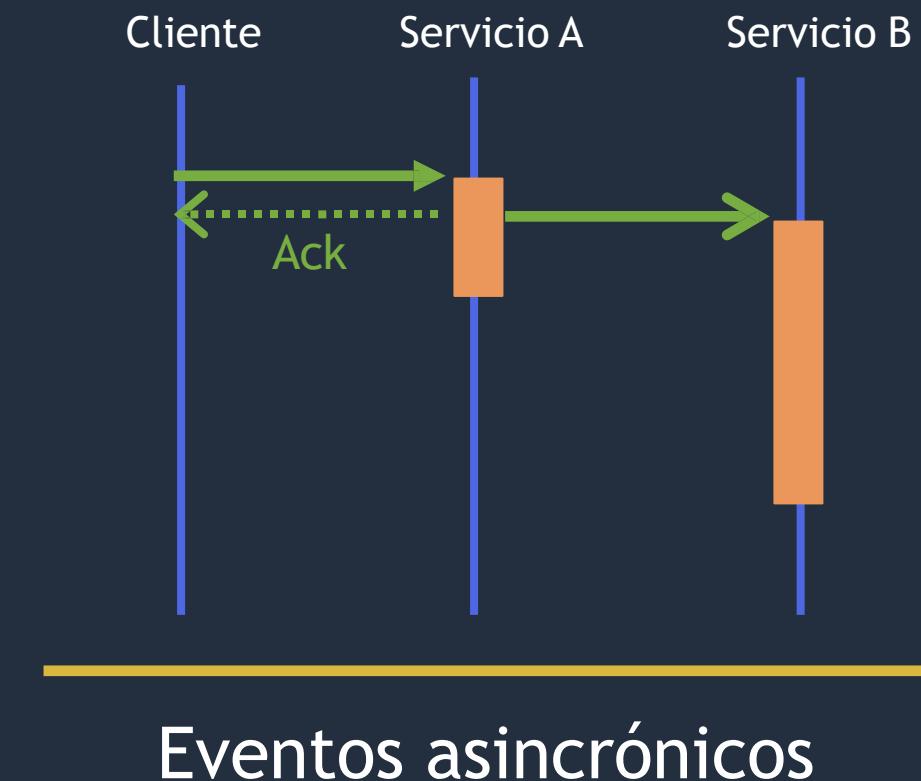
Function	Description
States.Array	Crea una matriz con una entrada dada
States.ArrayContains	Comprueba si la matriz dada contiene tu coincidencia
States.UUID	Genera un identificador único universal (UUID v4)
States.Base64Encode	Codifica datos según el esquema de codificación MIME Base64
States.Base64Decode	Decodifica datos basándose en el esquema de decodificación MIME Base64
States.Hash	Genera el valor de hash de la entrada dada (MD5, SHA-1, SHA-256, SHA-384, SHA-512)

Reduciendo las transiciones de estado y la duración con el patron de Callback

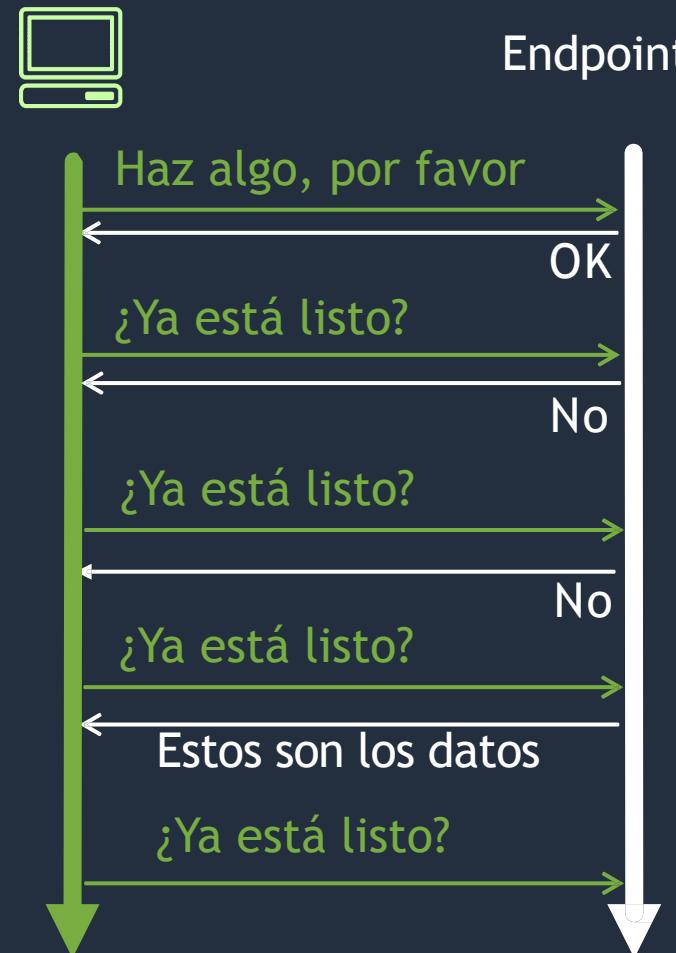


Manejo de los valores de respuesta y el estado de las solicitudes asincrónicas

No hay ruta de retorno para proporcionar más información más allá del acuse de recibo inicial

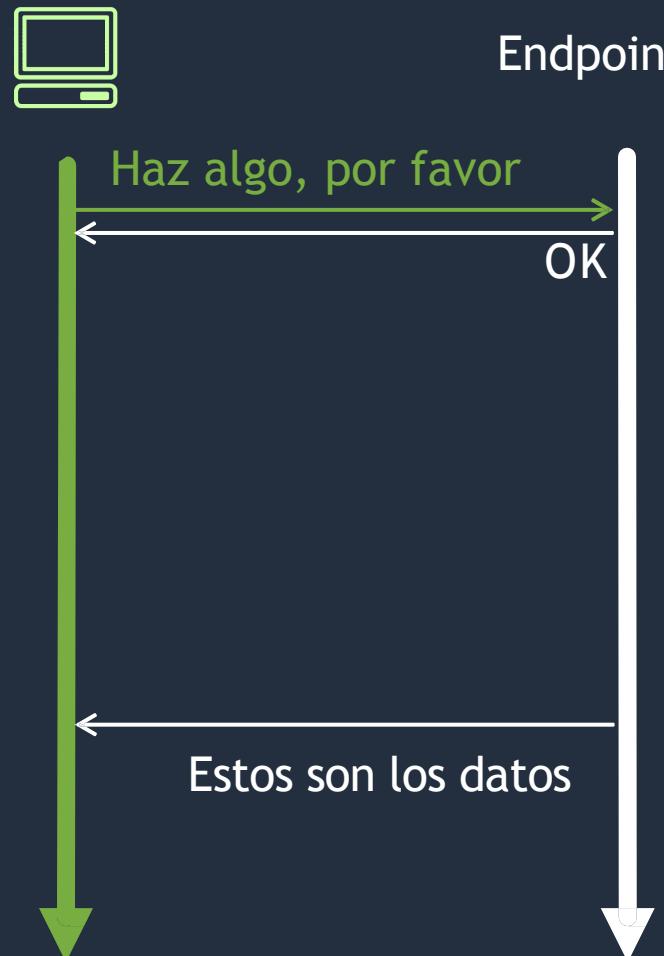


Seguimiento de una solicitud a bordo: Polling

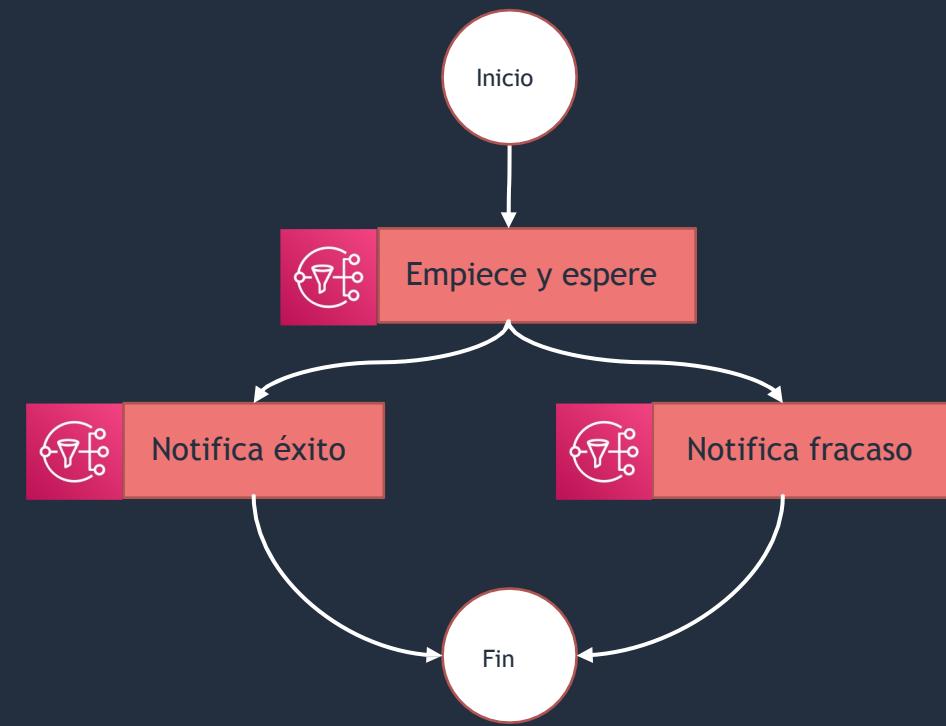


- La solicitud inicial devuelve un identificador de seguimiento
- Cree un segundo punto final de API para que la interfaz compruebe el estado de la solicitud, haciendo referencia al ID de seguimiento
- Utilice DynamoDB u otro banco de datos para realizar un seguimiento del estado de la solicitud
- Mecanismo sencillo de implementar
- Puede crear muchas llamadas vacías
- Retraso entre la disponibilidad y la notificación frontal

Seguimiento de una solicitud a bordo: Callback



- Una conexión bidireccional entre el servicio iniciador y el servicio descendente
- Sus servicios posteriores pueden enviar datos directamente al cliente.
- Más cerca del tiempo real
- Reduce la cantidad de mensajes entre el cliente y el sistema de fondo
- A menudo es más complejo de implementar

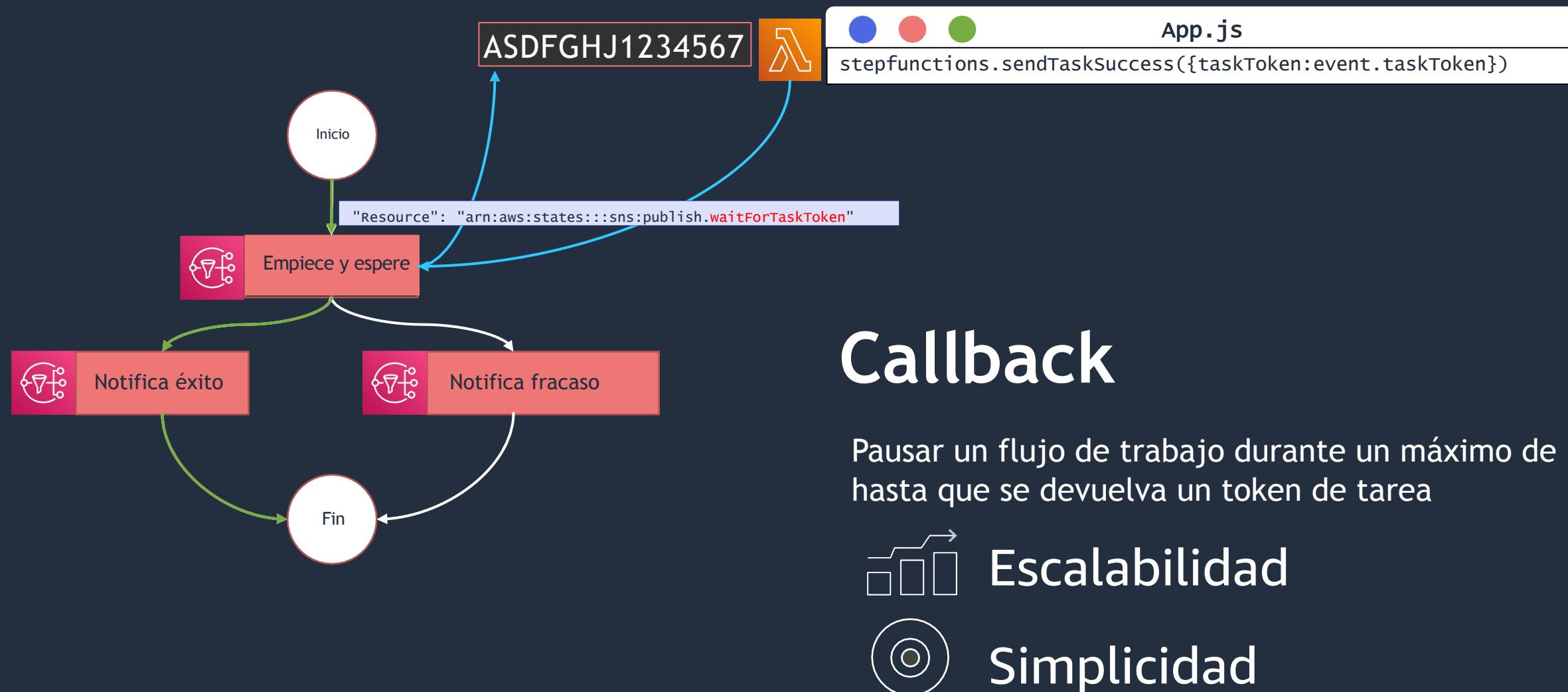


Callback

Pausar un flujo de trabajo durante un máximo de un año o hasta que se devuelva un token de tarea

Escalabilidad

Simplicidad

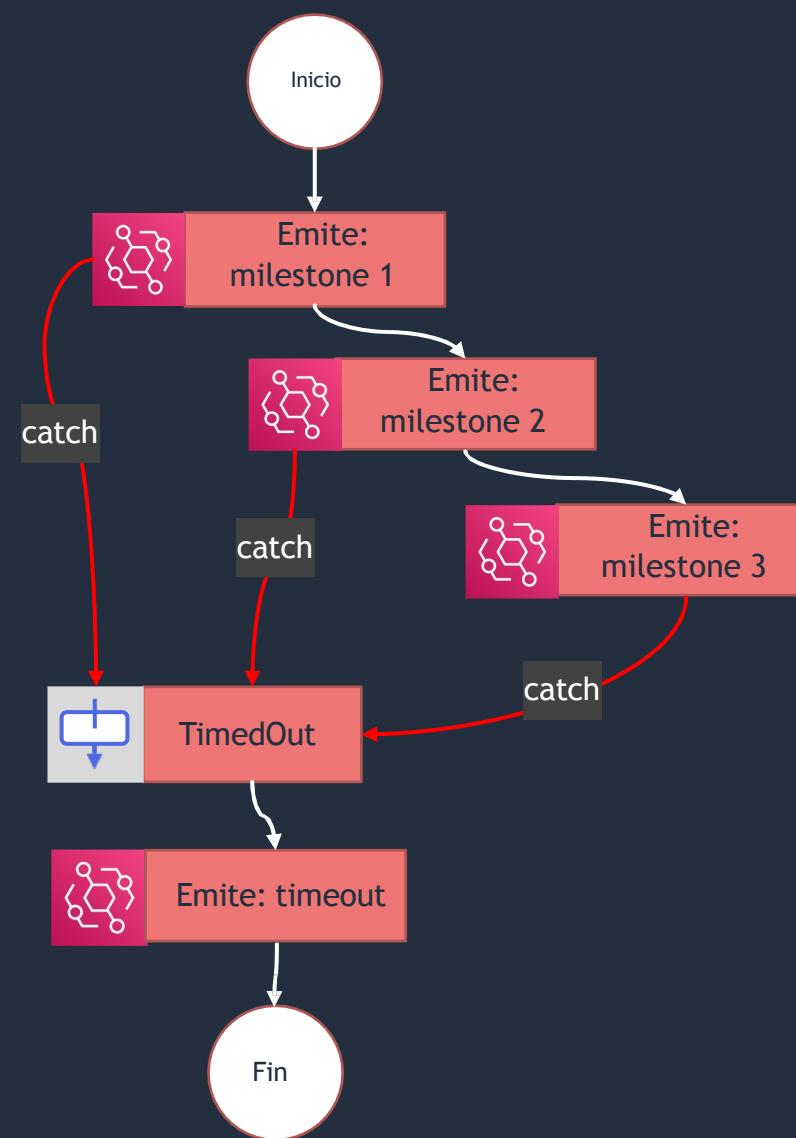


Callback

Pausar un flujo de trabajo durante un máximo de un año o hasta que se devuelva un token de tarea

Escalabilidad

Simplicidad



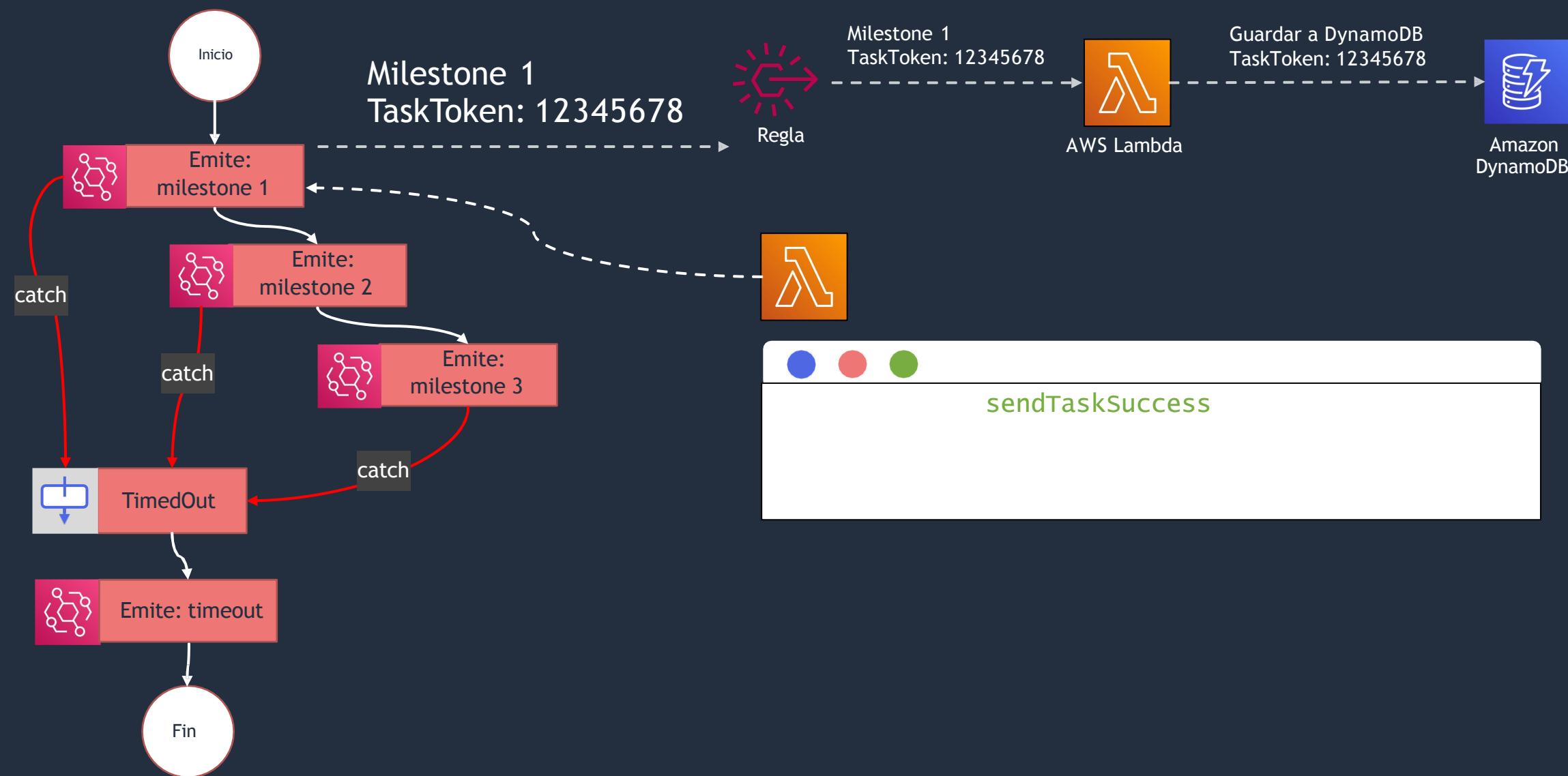
El “emite y espera”

Emite eventos hitos que invocan microservicios externos desacoplados y emiten eventos de error si no responden

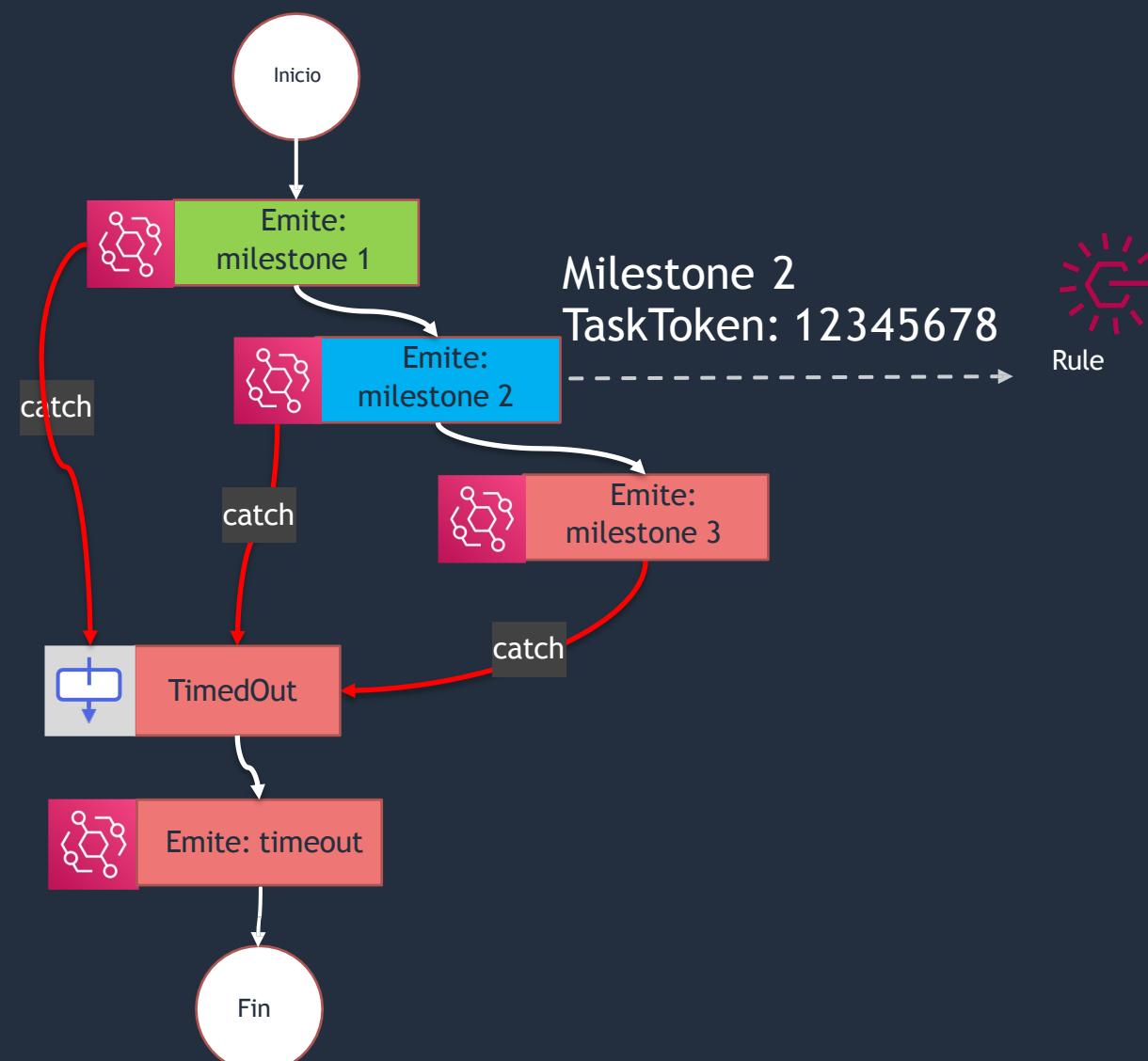


Fiabilidad





Fiabilidad

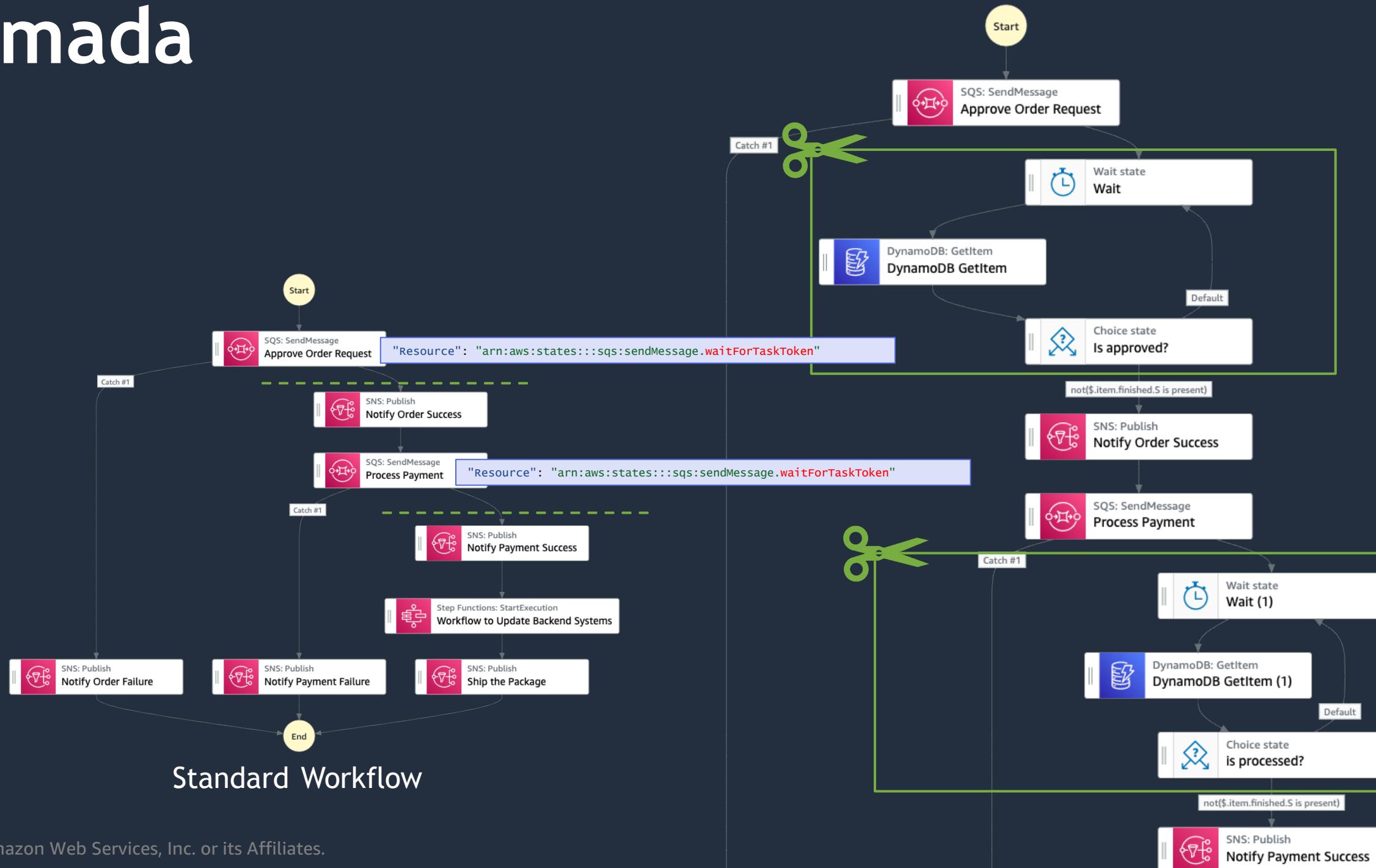


Establece un límite de tiempo para esperar al token de la tarea "HeartbeatSeconds": 20



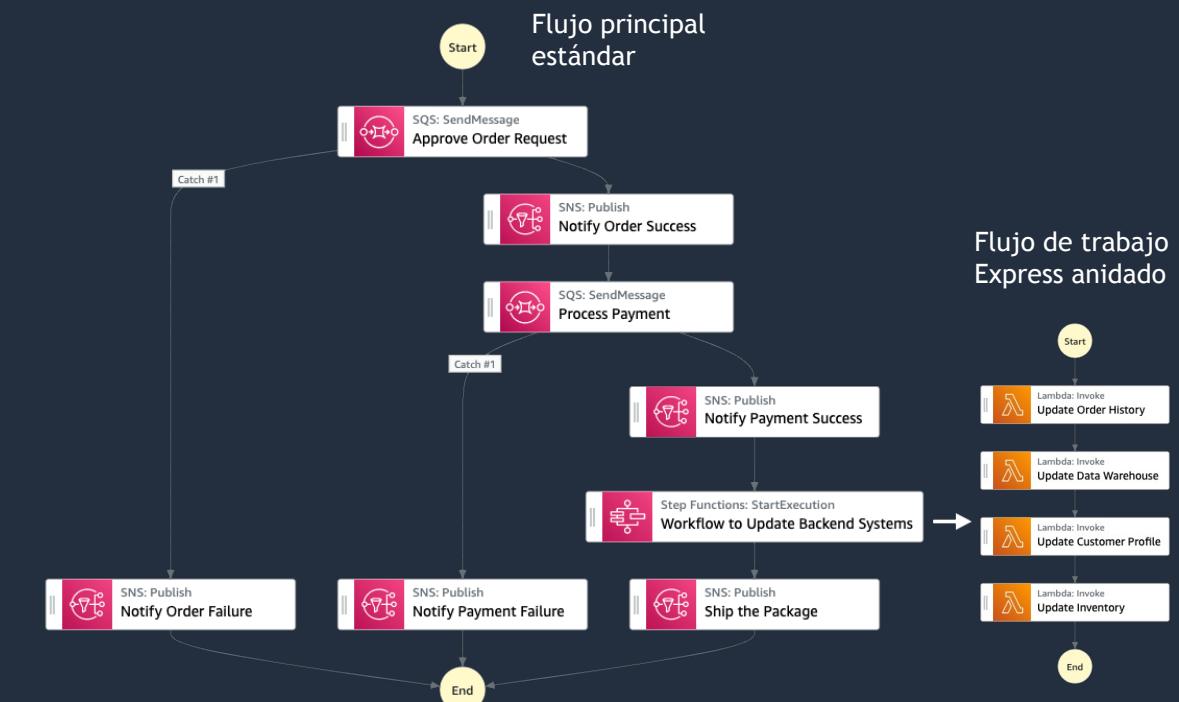
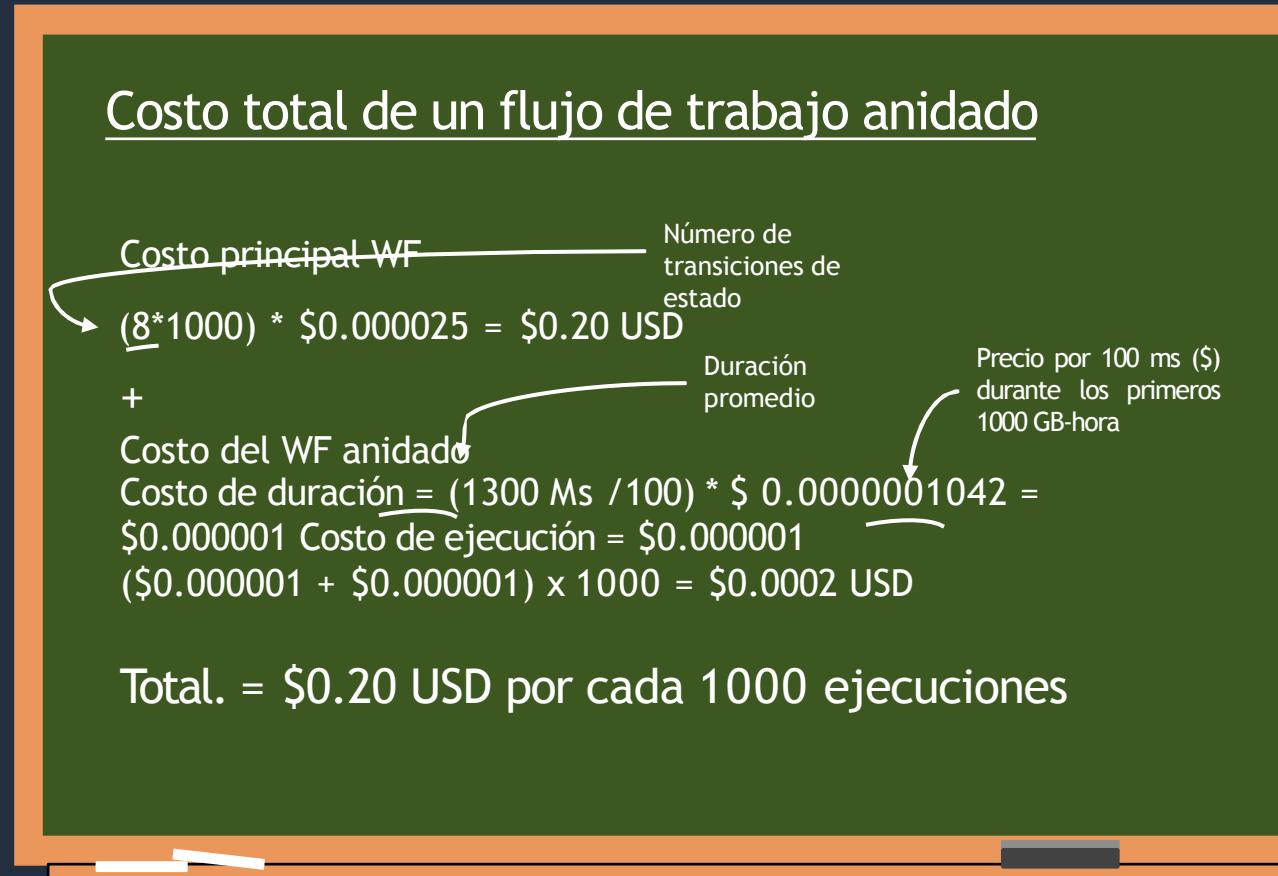
Fiabilidad

Eliminar el Polling con un patrón de Callback de llamada



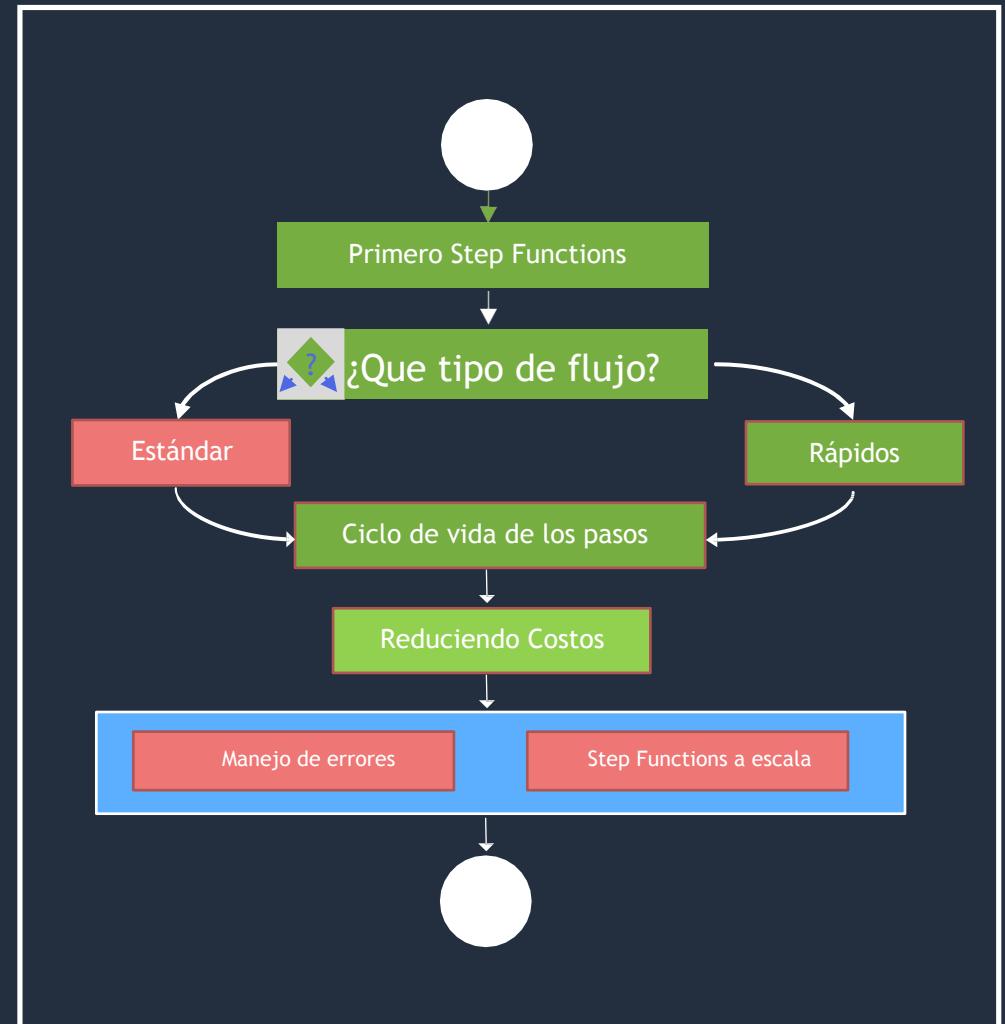
Costo por cada mil ejecuciones

LOS FLUJOS DE TRABAJO EXPRÉS SE COBRAN EN FUNCIÓN DEL NÚMERO DE SOLICITUDES Y LA DURACIÓN



Esta nueva combinación de flujo de trabajo se ejecuta 1000 veces y cuesta un costo total de 20 centavos

Gestión de fallos

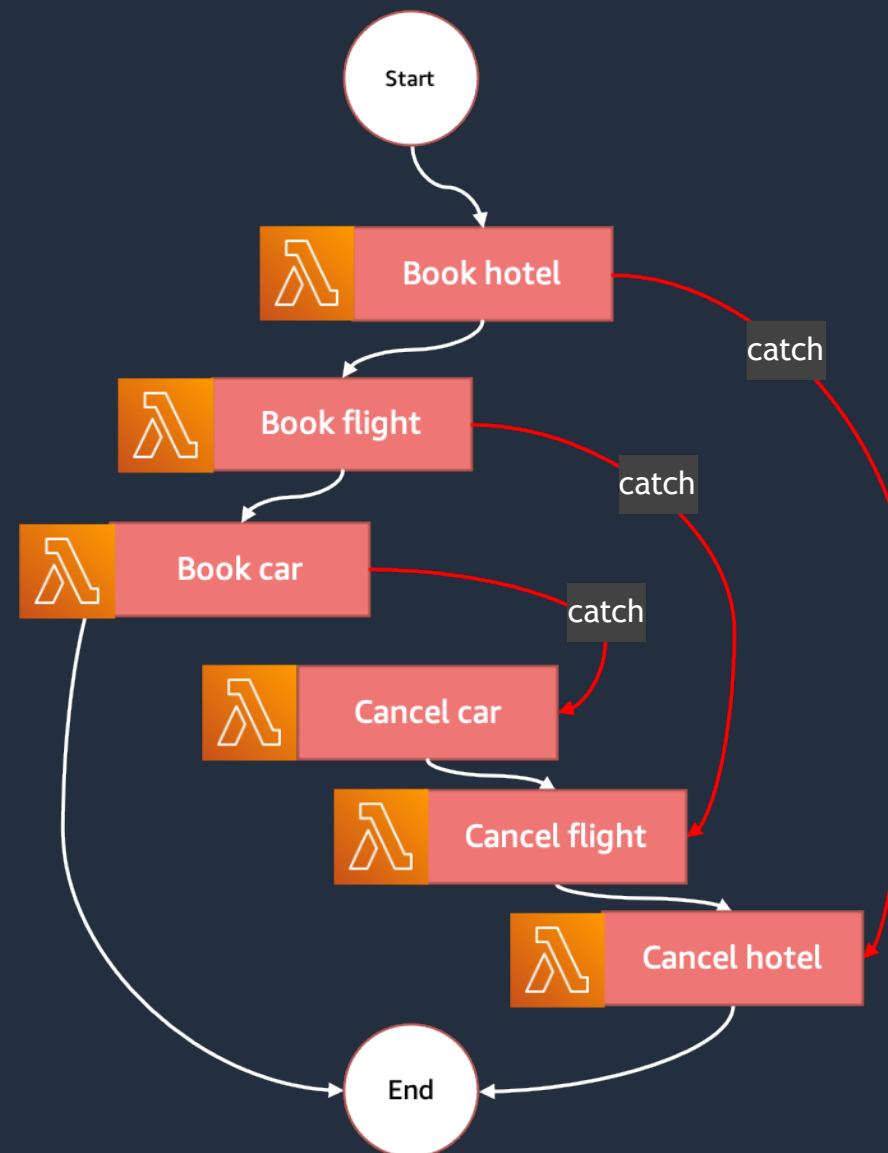




Everything fails, all the time



Werner Vogels
VP & CTO
Amazon.com

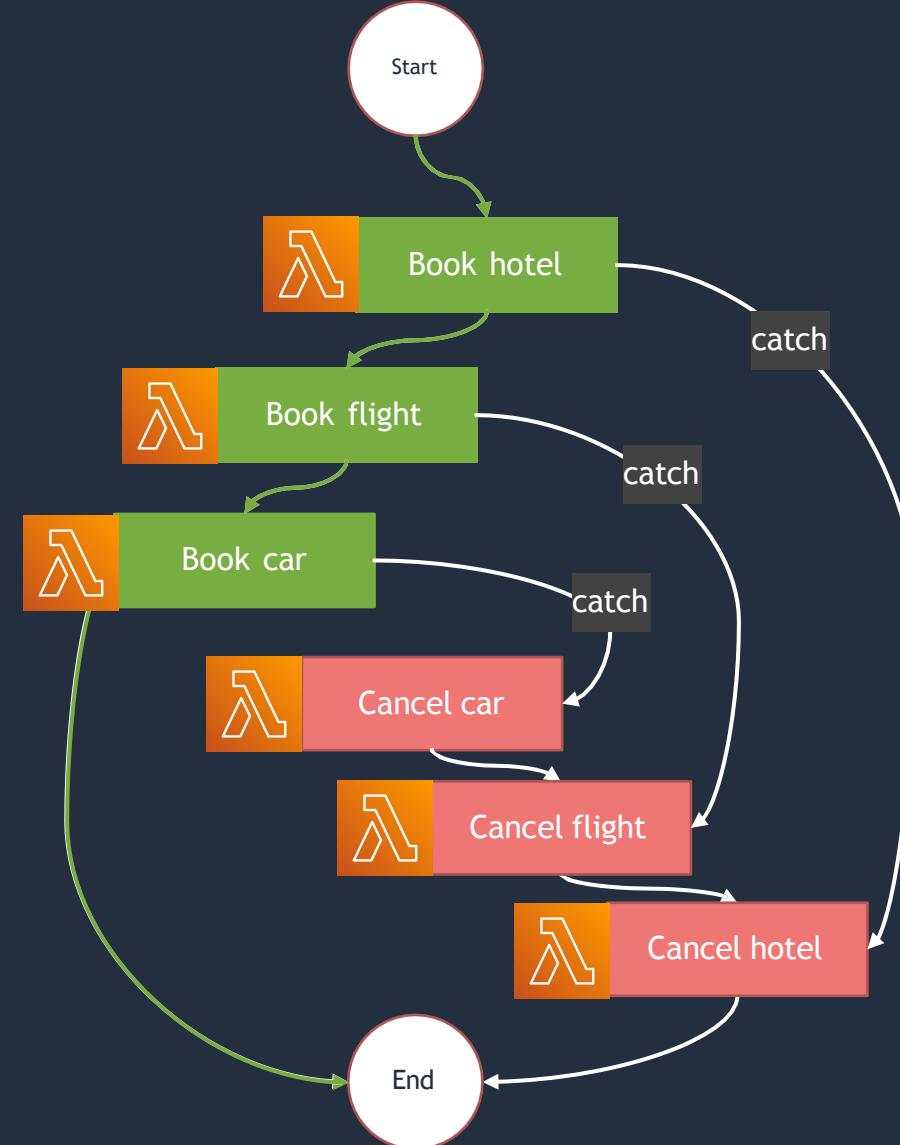


La “saga” continúa

Utilice el manejo de errores inherente para revertir las fallas secuenciales del sistema para transacciones de larga duración



Resiliencia



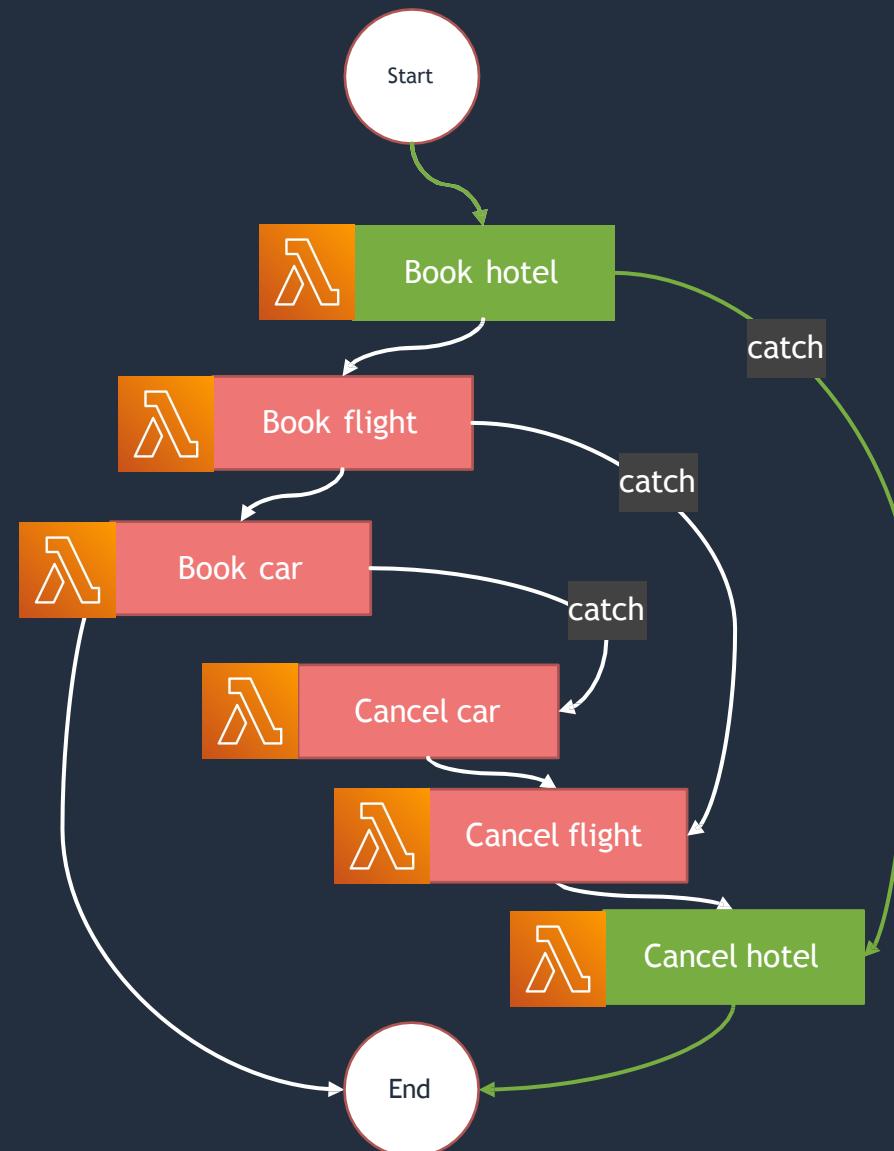
Caso de éxito

La “saga” continúa

Utilice el manejo de errores inherente para revertir las fallas secuenciales del sistema para transacciones de larga duración



Resiliencia



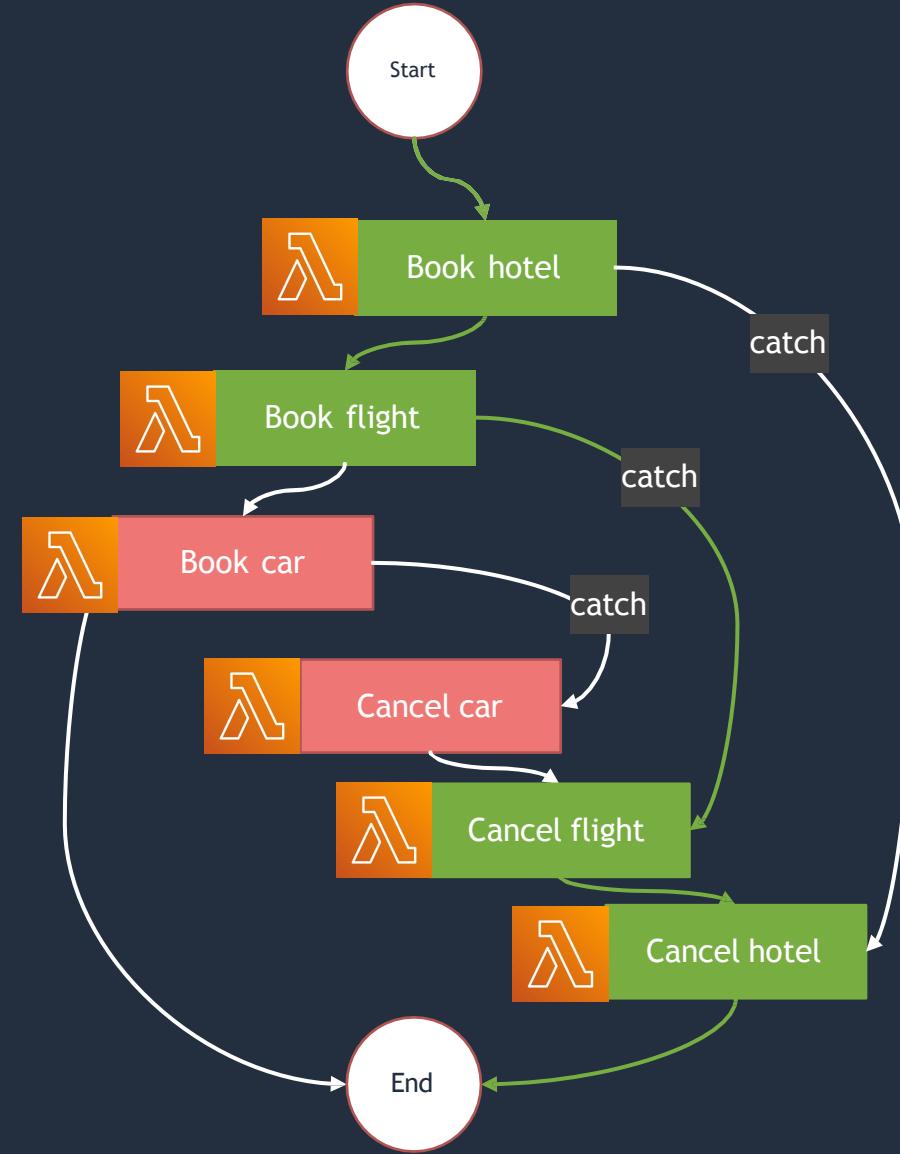
Caso de fracaso

La “saga” continúa

Utilice el manejo de errores inherente para revertir las fallas secuenciales del sistema para transacciones de larga duración



Resiliencia



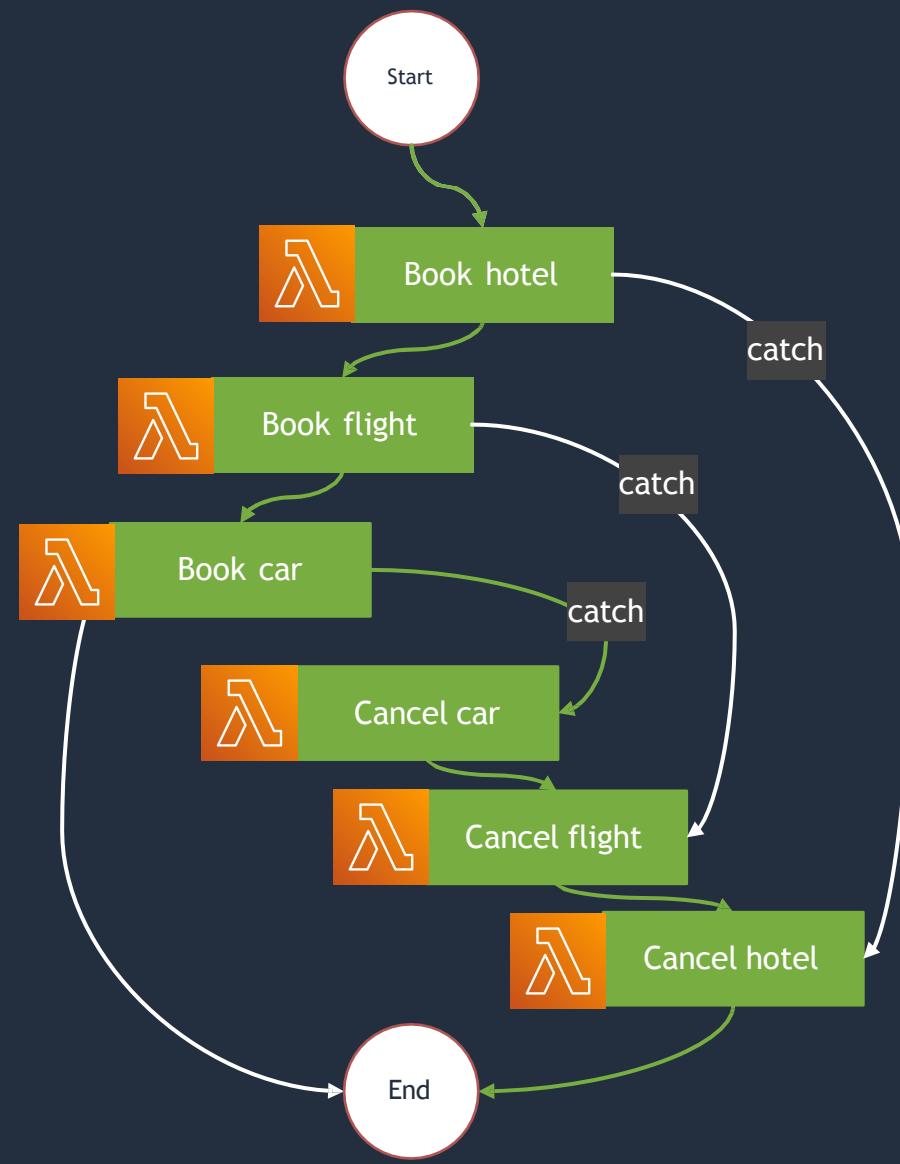
Caso de fracaso

La “saga” continúa

Utilice el manejo de errores inherente para revertir las fallas secuenciales del sistema para transacciones de larga duración



Resiliencia



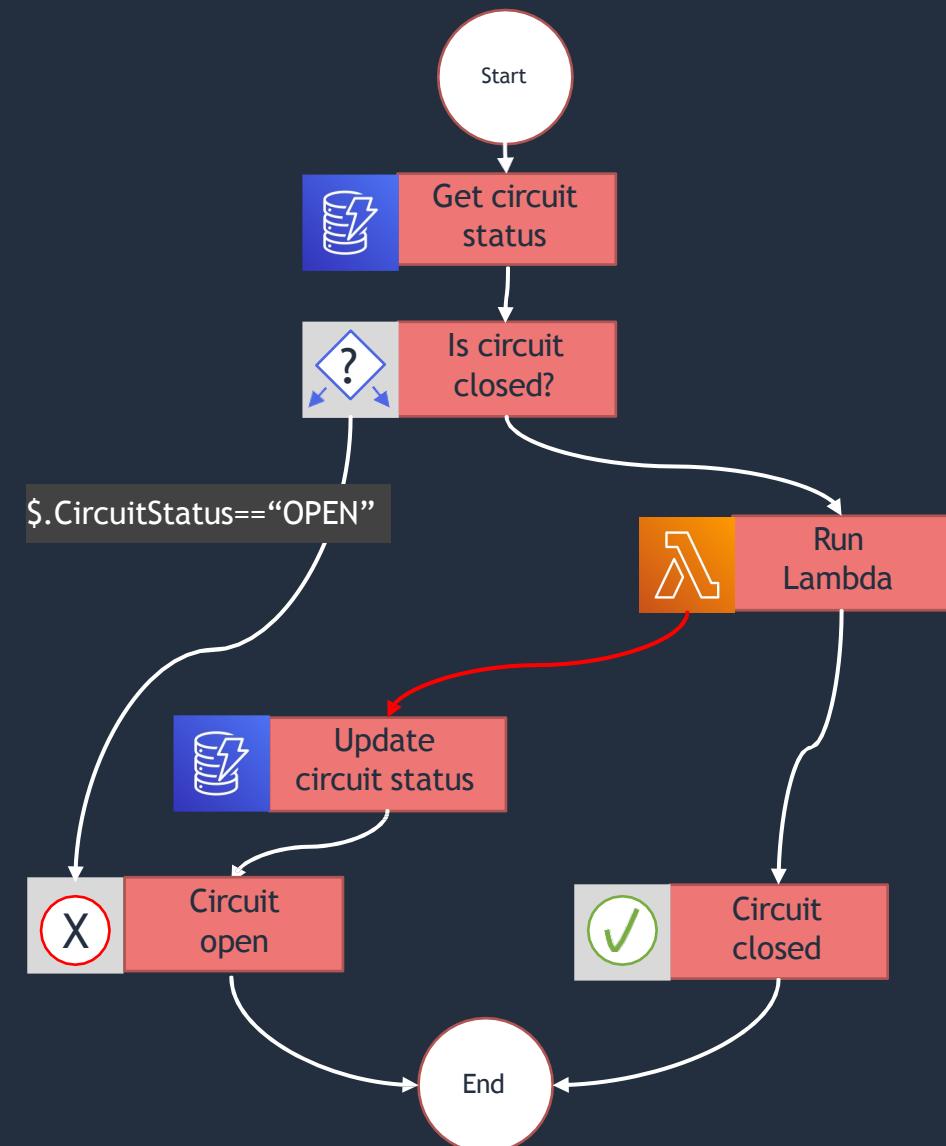
Caso de fracaso

La “saga” continúa

Utilice el manejo de errores inherente para revertir las fallas secuenciales del sistema para transacciones de larga duración



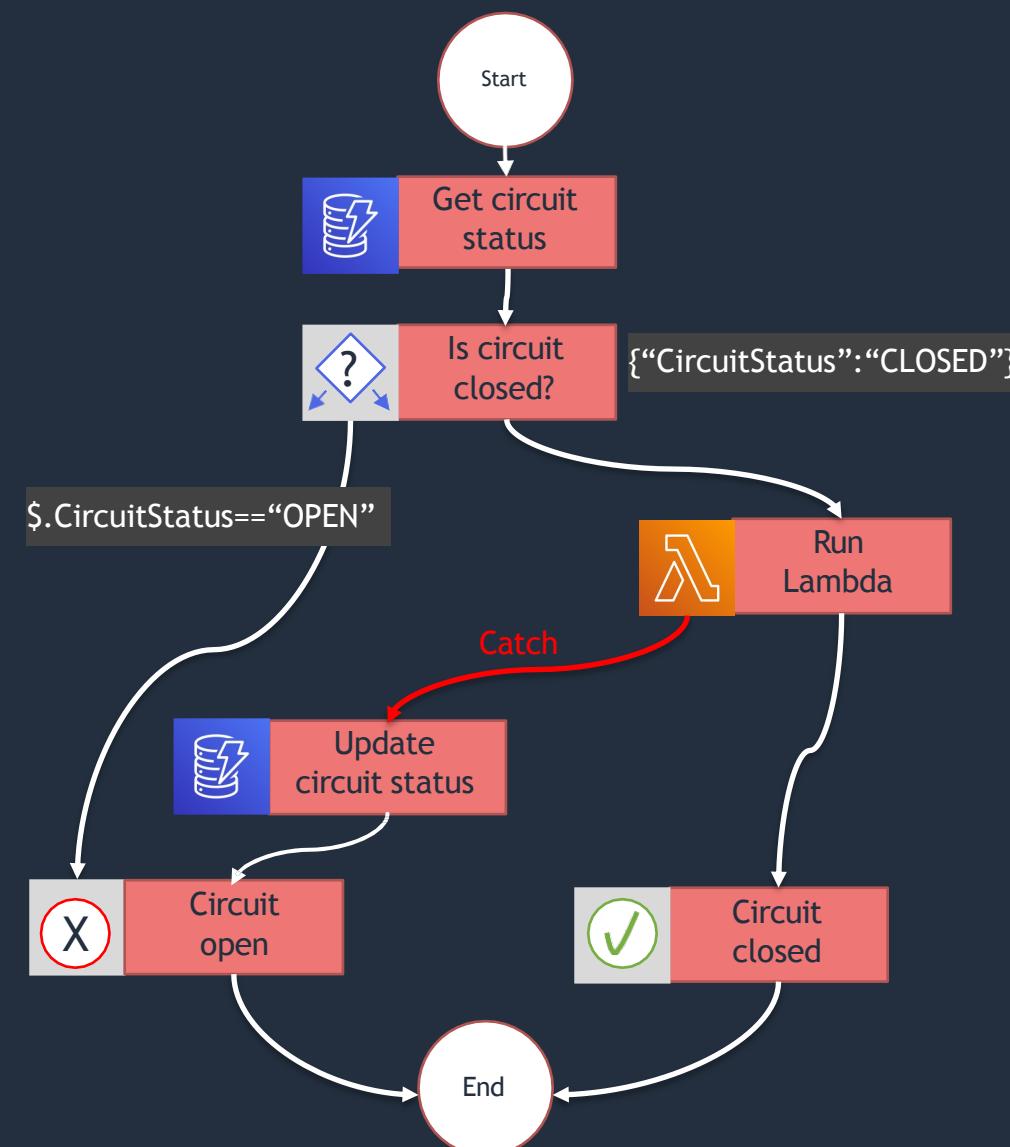
Resiliencia



El “circuit breaker”

Prevenga que el servicio de la persona que llama vuelva a intentar otra llamada de servicio, que previamente haya provocado repetidos tiempos de espera o errores

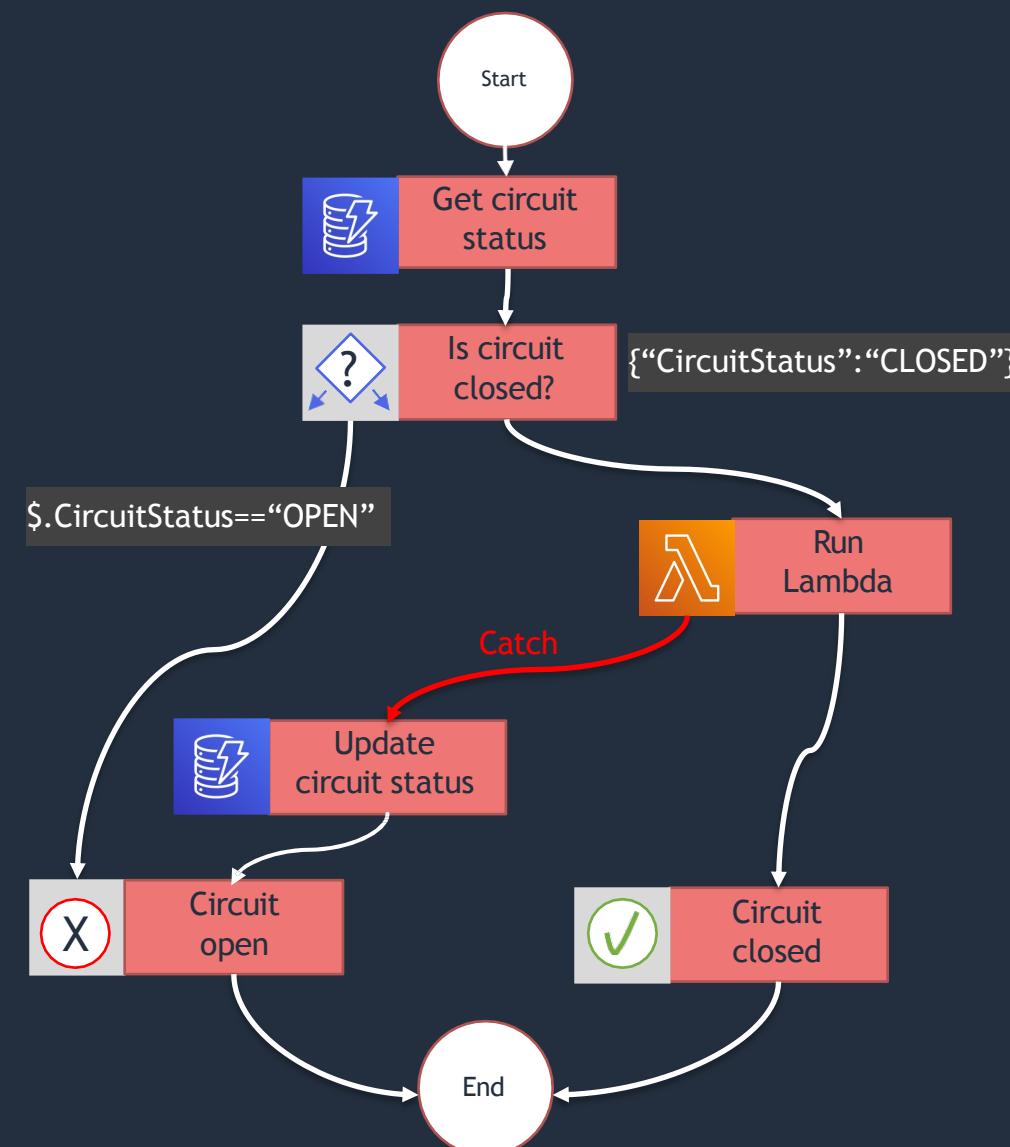




Casos de éxito

El “circuit breaker”

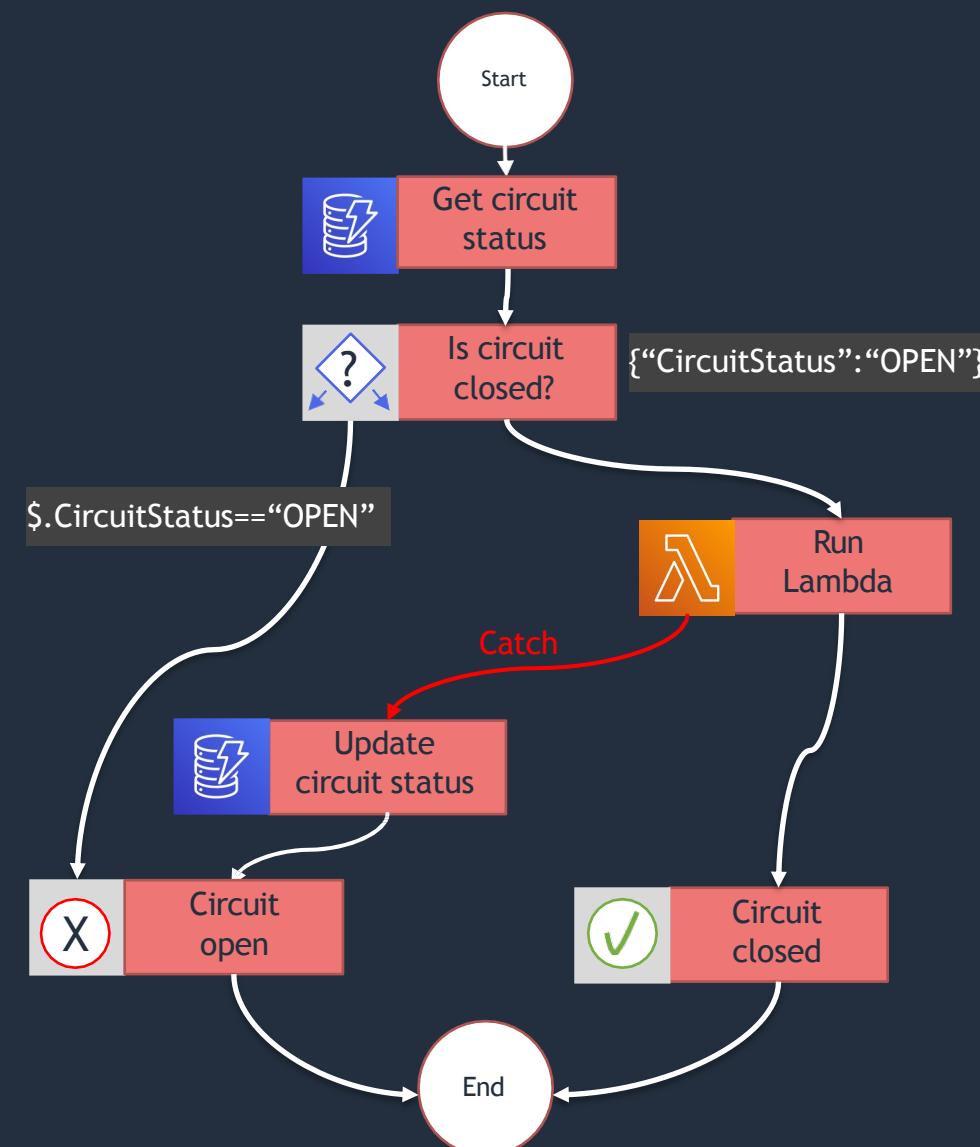
Prevenga que el servicio de la persona que llama vuelva a intentar otra llamada de servicio, que previamente haya provocado repetidos tiempos de espera o errores



Casos de éxito

El “circuit breaker”

Prevenga que el servicio de la persona que llama vuelva a intentar otra llamada de servicio, que previamente haya provocado repetidos tiempos de espera o errores

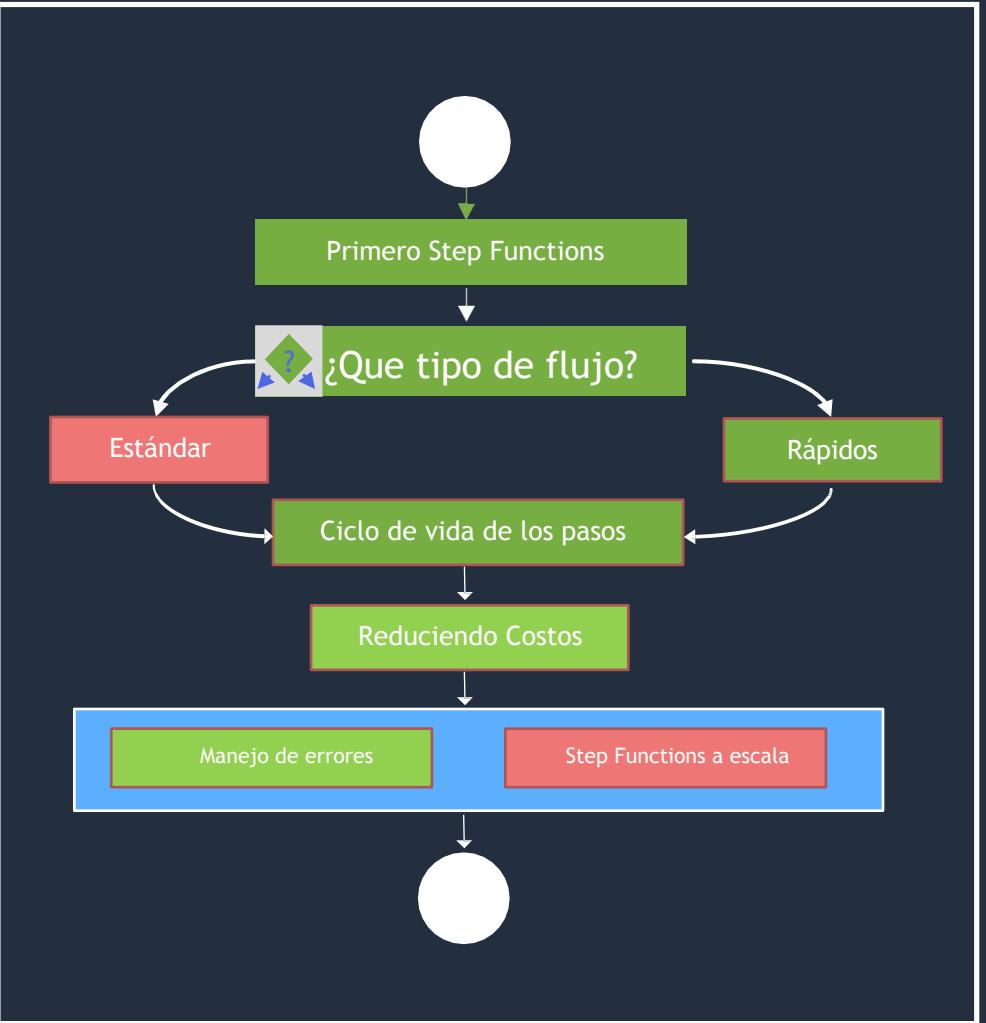


El “circuit breaker”

Prevenga que el servicio de la persona que llama vuelva a intentar otra llamada de servicio, que previamente haya provocado repetidos tiempos de espera o errores



Step Functions a escala



Step Functions a escala

Paralelismo

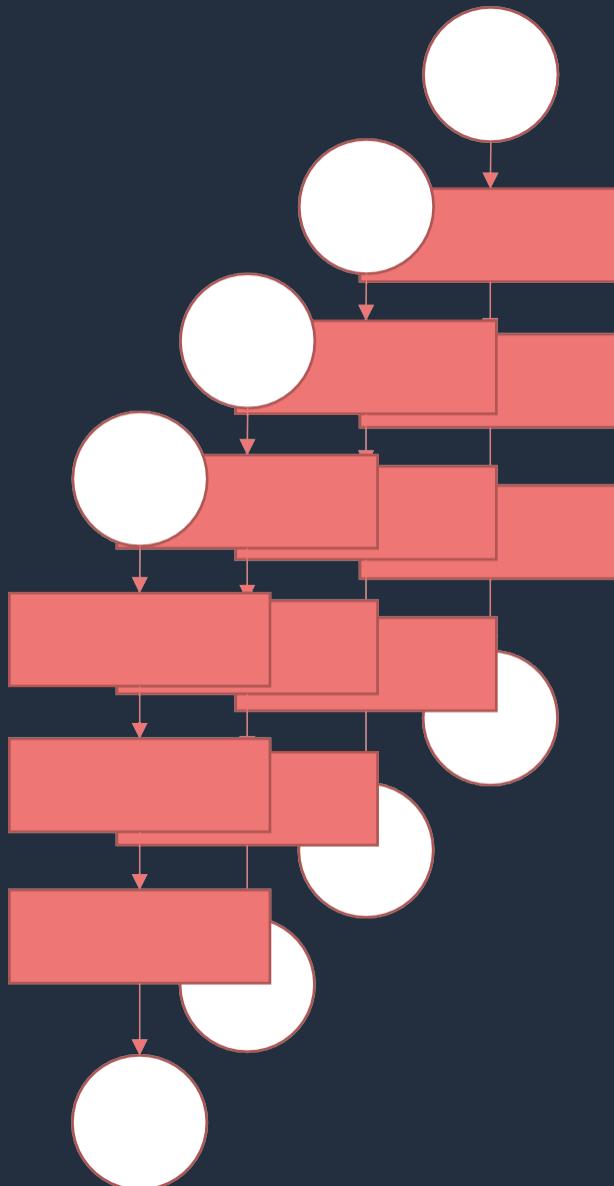
Hacer muchas cosas a la vez
Misma entrada, diferentes
pasos

VS

Concurrencia

Gestionar muchas cosas a la vez Mismos
pasos, diferentes aportaciones

Invocaciones paralelas,
muchos flujos de trabajo
independientes que
hacen lo suyo

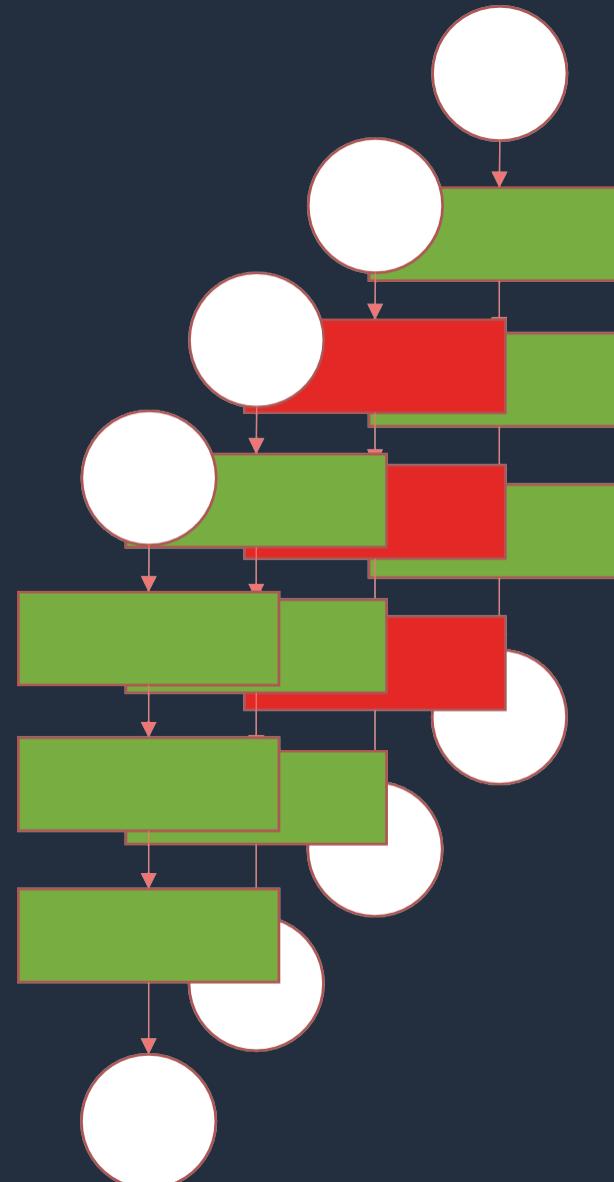


Se ejecuta la misma definición de ASL al mismo tiempo, con datos diferentes

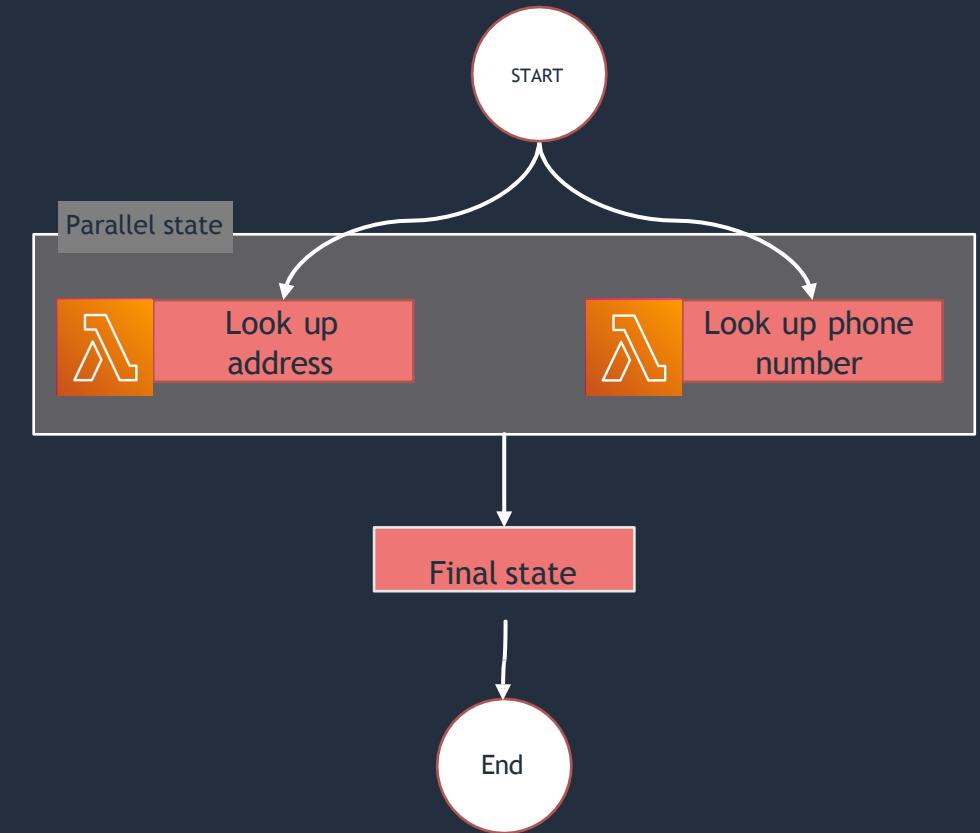
Cada proceso es independiente

El error en un proceso no afecta a otro.

Por ejemplo, el flujo de trabajo de corrección de un recurso no conforme. Limitación: cotización de la cuenta (en su mayoría límites flexibles)

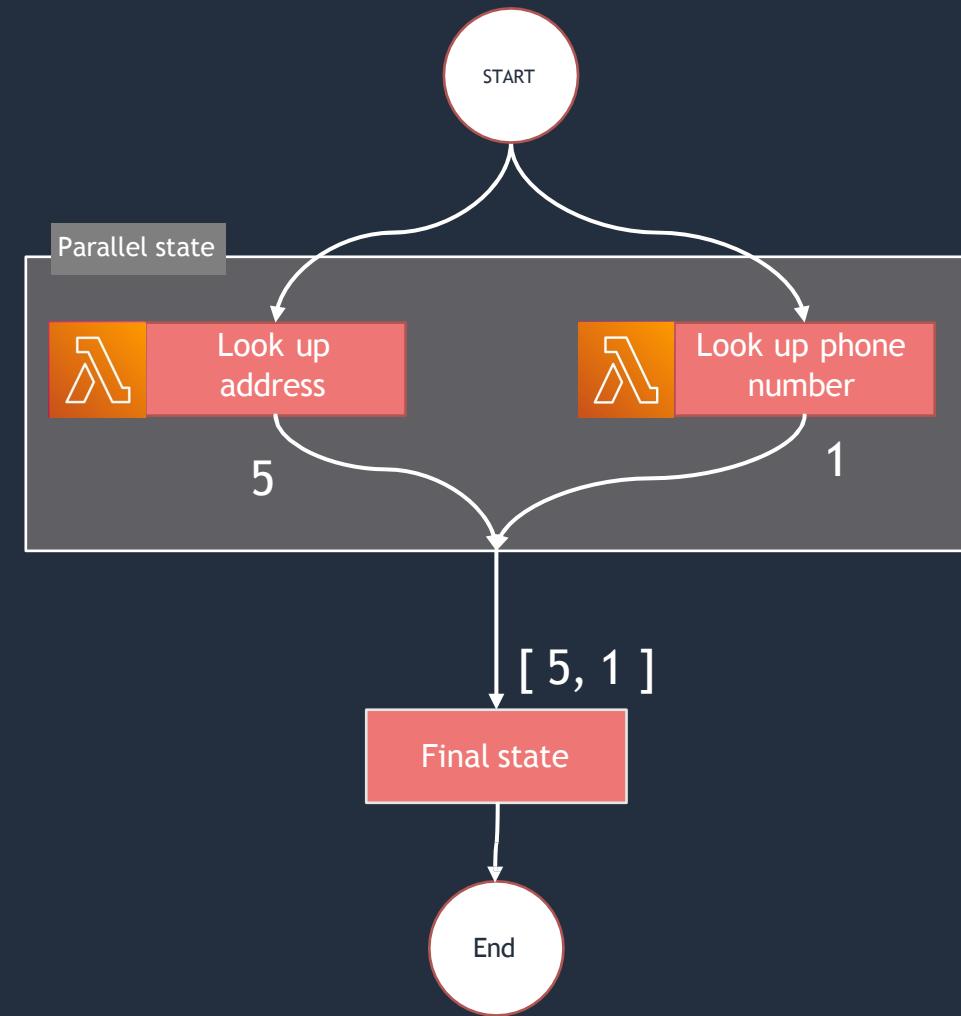


Estado paralelo: ejecuta un número fijo de ramas en paralelo



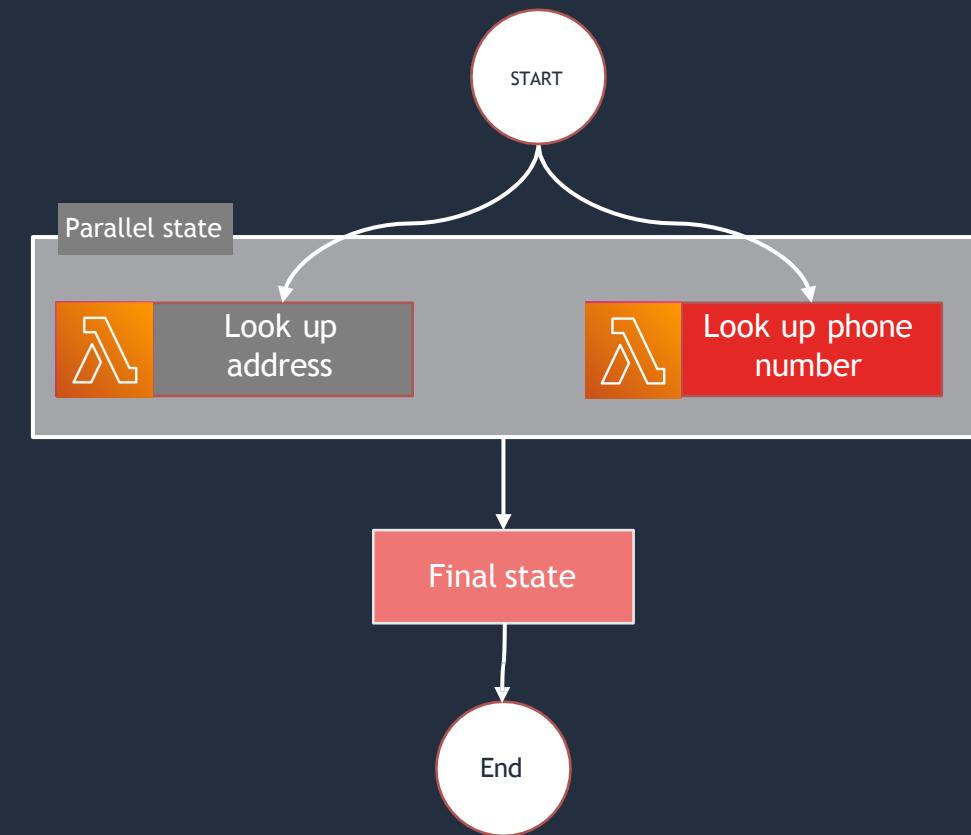
Ejecuta varias ramas de pasos utilizando la misma entrada

La salida del estado del mapa es una matriz con un elemento para cada resultado de rama



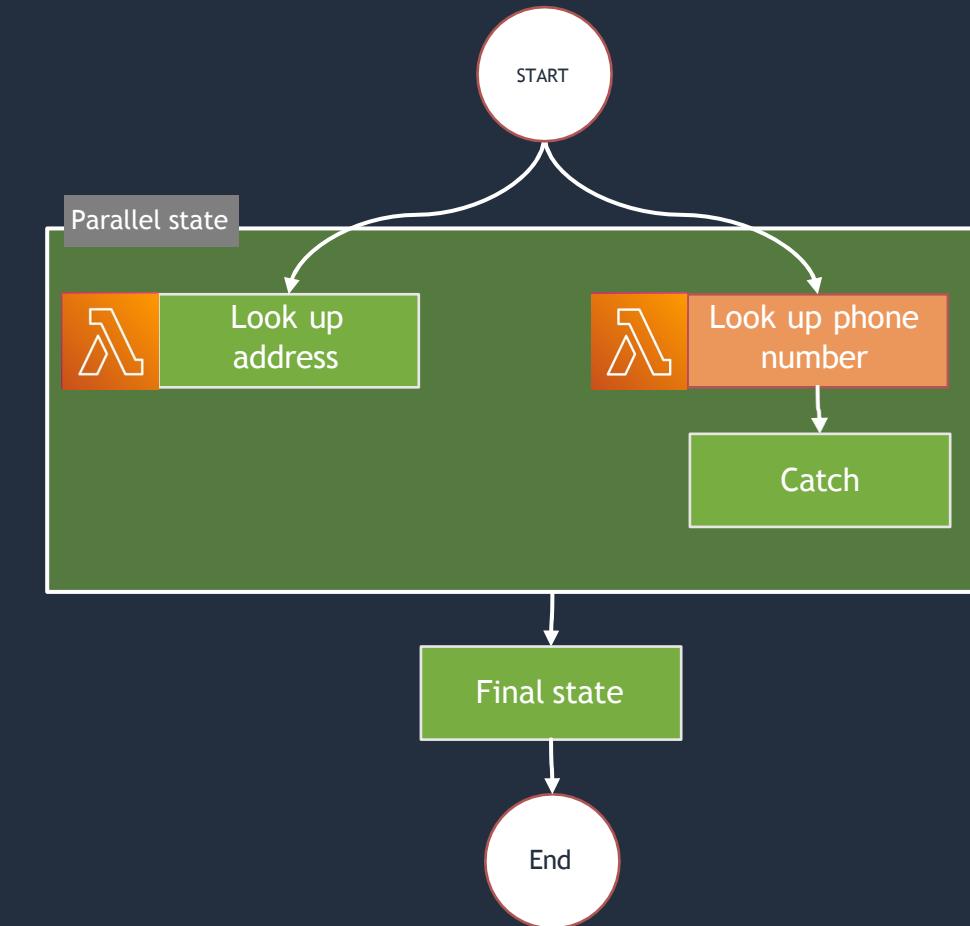
Espere hasta que todas las ramas terminen
(alcancen un estado terminal) antes de procesar
el siguiente estado.

Si alguna rama falla debido a un error no
controlado o a la transición a un estado de
error, se considera que todo el estado paralelo
ha fallado y todas sus ramas se detienen

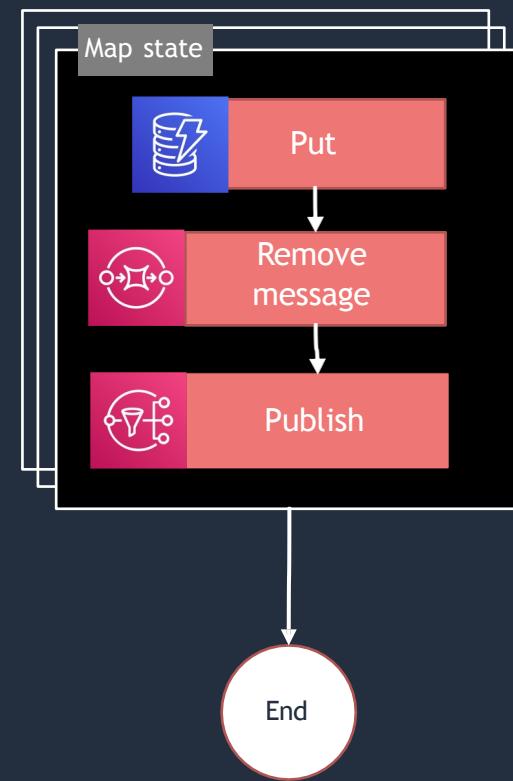


Detecta errores para permitir que otras ramas se completen

Todos los resultados de la ejecución se agregan al estado final



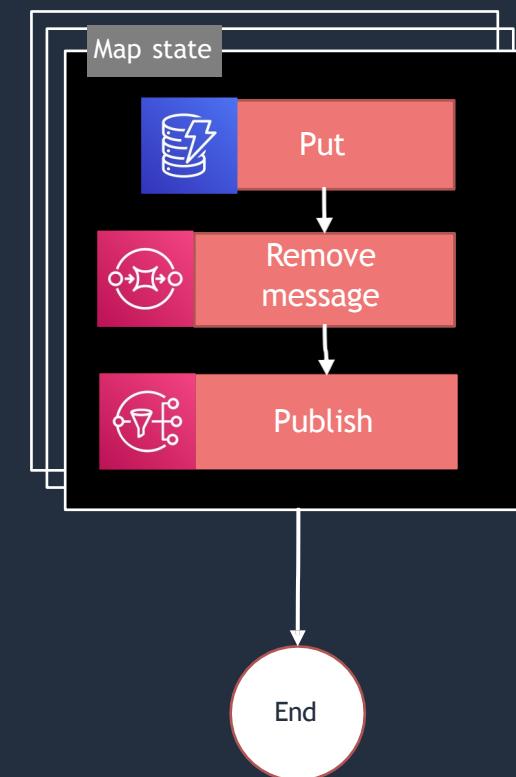
**Paralelismo dinámico:
un subflujo de trabajo
para realizar el trabajo
más rápido mediante
el uso de más recursos
al mismo tiempo**



Ejecute los mismos pasos para varias entradas de una matriz de entrada

MaxConcurrency controla el número de iteraciones que se ejecutarán en paralelo

Establezca en 1 para integraciones de un solo subprocesso;
Establezca en 0 para un parallelismo máximo



Presentando la colección de flujos de trabajo de Serverless

Descubra, implemente y comparta los flujos de trabajo de Step Functions

<https://s12d.com/workflows>

New Blogs Videos Learn Patterns Workflows Office Hours Search Search

Serverless Workflows Collection

Submit a workflow

New to AWS Step Functions? Try the new interactive [Step Functions workshop](#).

Filters (16 workflows) Reset

Workflow Type

- Express
- Standard

Use case

- Data Processing
- Production line
- SaaS Integration
- Scheduled job
- Security automation
- Team collaboration

Level

- 1 - Fundamental
- 2 - Pattern
- 3 - Application

Services

- Amazon API Gateway
- Amazon Athena
- AWS Batch
- Amazon Comprehend
- Amazon DynamoDB
- Amazon EventBridge
- AWS Lambda
- Amazon S3
- AWS Step Functions
- Amazon SNS

Use Step Functions workflows to quickly build applications using AWS SAM and CDK templates. Filter by use-case and copy the template or workflow definition directly into your application.

This workflow diagram illustrates a Concurrency Controller. It starts with a 'Start' state, followed by a 'Put state ModifyState' step. This leads to a 'Lambda invoke VerifyPolicy' step. A 'Decision state ChooseAction' follows, which branches into two paths: one for 'Allowed action == Next' leading to 'Lambda invoke Author', and another for 'Allowed action == Skip' leading to 'Lambda invoke RevertPolicy'. Both paths converge back to a 'Decision state Minimise' step, which then leads to a 'Put state Minimise' step and finally ends.

Concurrency Controller

Standard 3 - Application View

This workflow diagram shows an Automated policy orchestrator. It begins with a 'Start' state, followed by a 'Put state ModifyState' step. This is followed by a 'Lambda invoke VerifyPolicy' step. A 'Decision state ChooseAction' step follows, which branches into three paths: one for 'Allowed action == Next' leading to 'Lambda invoke Author', one for 'Allowed action == Skip' leading to 'Lambda invoke RevertPolicy', and one for 'Allowed action == Ignored' leading to 'Lambda invoke Ignored'. All three paths converge back to a 'Decision state Minimise' step, which then leads to a 'Put state Minimise' step and finally ends.

Automated policy orchestrator

Standard 3 - Application View

This workflow diagram depicts an Automated Zendesk support ticket moderator. It starts with a 'Start' state, followed by a 'DynamoDB GetItem GetCircuitStatus' step. This is followed by a 'Decision state Is Circuit Closed' step. If the circuit is closed, it proceeds to a 'Lambda invoke Execute Lambda' step. If the circuit is open, it goes to a 'Fail state Circuit Open' step. Both paths converge to a 'DynamoDB PutItem Update Circuit Status' step, which then leads to a 'Success state Circuit Closed' step and finally ends.

Automated Zendesk support ticket moderator

Standard 3 - Application View

This workflow diagram illustrates a Circuit breaker workflow. It starts with a 'Start' state, followed by a 'DynamoDB GetItem GetCircuitStatus' step. This is followed by a 'Decision state Is Circuit Closed' step. If the circuit is closed, it proceeds to a 'Lambda invoke Execute Lambda' step. If the circuit is open, it goes to a 'Fail state Circuit Open' step. Both paths converge to a 'DynamoDB PutItem Update Circuit Status' step, which then leads to a 'Success state Circuit Closed' step and finally ends.

Circuit breaker workflow

Standard 2 - Pattern View

Descubra ejemplos de flujos de trabajo

1. Flujo visual



2. Plantilla de laC

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: >
  Sample SAM Template for circuit breaker pattern

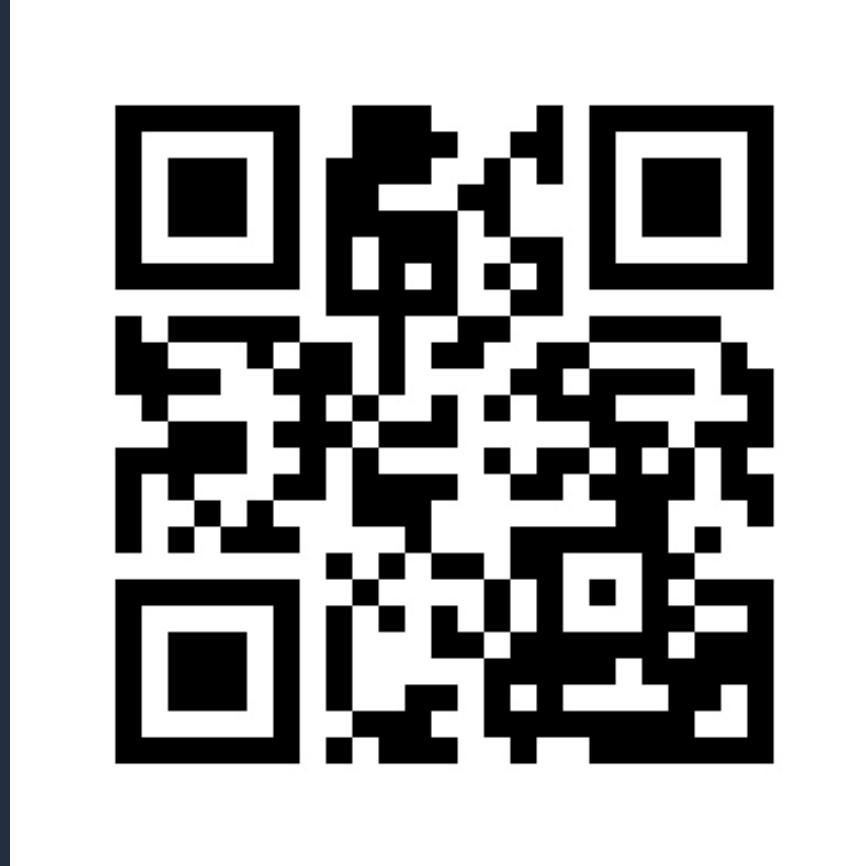
Resources:
  CircuitBreakerStateMachine:
    Type: AWS::Serverless::StateMachine # More info about State Machine Resource
    Properties:
      DefinitionUri: statemachine/circuitbreaker.asl.json
      DefinitionSubstitutions:
        GetCircuitStatusFunctionArn: !GetAtt GetCircuitStatusFunction.Arn
        UpdateCircuitStatusFunctionArn: !GetAtt UpdateCircuitStatusFunction.Arn
        DDBPutItem: !Sub arn:${AWS::Partition}:states:::dynamodb:putItem
        DDBTable: !Ref CircuitBreakerTable
      Policies: # Find out more about SAM policy templates: https://docs.aws.amazon.com/serverless/latest/developerguide/policy.html
        - AWSLambdaRole
        - DynamoDBWritePolicy:
            TableName: !Ref CircuitBreakerTable

  GetCircuitStatusFunction:
    Type: AWS::Serverless::Function # More info about Function Resource: https://docs.aws.amazon.com/serverless/latest/developerguide/functions.html
    Properties:
      CodeUri: ./GetCircuitStatusLambda/src/GetCircuitStatusLambda/
      Handler: GetCircuitStatusLambda::GetCircuitStatusLambda.GetCircuitStatus
      Runtime: dotnetcore3.1
      MemorySize: 256
      Timeout: 30
```

3. Definición de ASL

```
{
  "StartAt": "Get Circuit Status",
  "States": {
    "Get Circuit Status": {
      "Next": "Is Circuit Closed",
      "Type": "Task",
      "Comment": "Get Circuit Status",
      "Resource": "${GetCircuitStatusFunctionArn}"
    },
    "Is Circuit Closed": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.CircuitStatus",
          "StringEquals": "OPEN",
          "Next": "Circuit Open"
        },
        {
          "Variable": "$.CircuitStatus",
          "StringEquals": "",
          "Next": "Execute Lambda"
        }
      ]
    },
    "Circuit Open": {
      "Type": "Fail"
    }
  }
}
```

Todo en un solo lugar



<https://s12d.com/api309>

Recursos para API309
Patrones avanzados de flujo de trabajo
serverless y mejores prácticas

- Plantillas descargables
- Publicaciones de blog
- Vídeos
- Talleres
- Ejemplos de código

¡Gracias!

Alfredo Peña

penaalfr@amazon.com

LinkedIn: /alfredo-peña-0829556



Encuesta

<https://www.pulse.aws/survey/824XTRK9> - P

¡Gracias!

Alfredo Peña

penaalfr@amazon.com

LinkedIn: /alfredo-peña-0829556



Encuesta

[https://www.pulse.aws/survey/6YYQQFFC - C](https://www.pulse.aws/survey/6YYQQFFC-C)