

LATAM EDA ROADSHOW - COLOMBIA

Construyendo Serverlesspresso: Creando arquitecturas orientadas a eventos

Andrés Victoria

Sr. Solutions Architect, Serverless TFC member
AWS



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Acerca de mí



- James Beswick
 - Correo electrónico: jbeswick@amazon.com
 - Twitter: [@jbesw](https://twitter.com/jbesw)
- Manager de AWS Serverless Developer Advocacy
- Geek Serverless
- Anteriormente
 - Ingeniero de Software y Gerente de Producto
 - Tech guy de múltiples startups y Fortune 500s
 - Cliente de AWS desde 2012

Agenda

1. Introducción a Serverlesspresso
2. Decisiones de diseño
3. Lecciones aprendidas
4. Patrones útiles

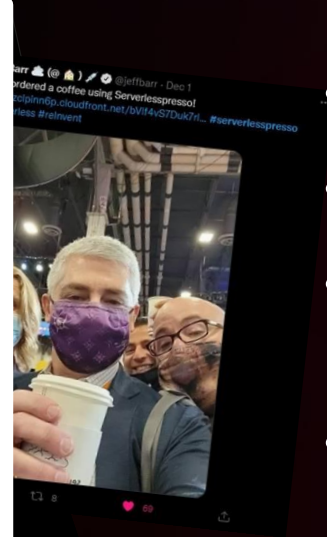
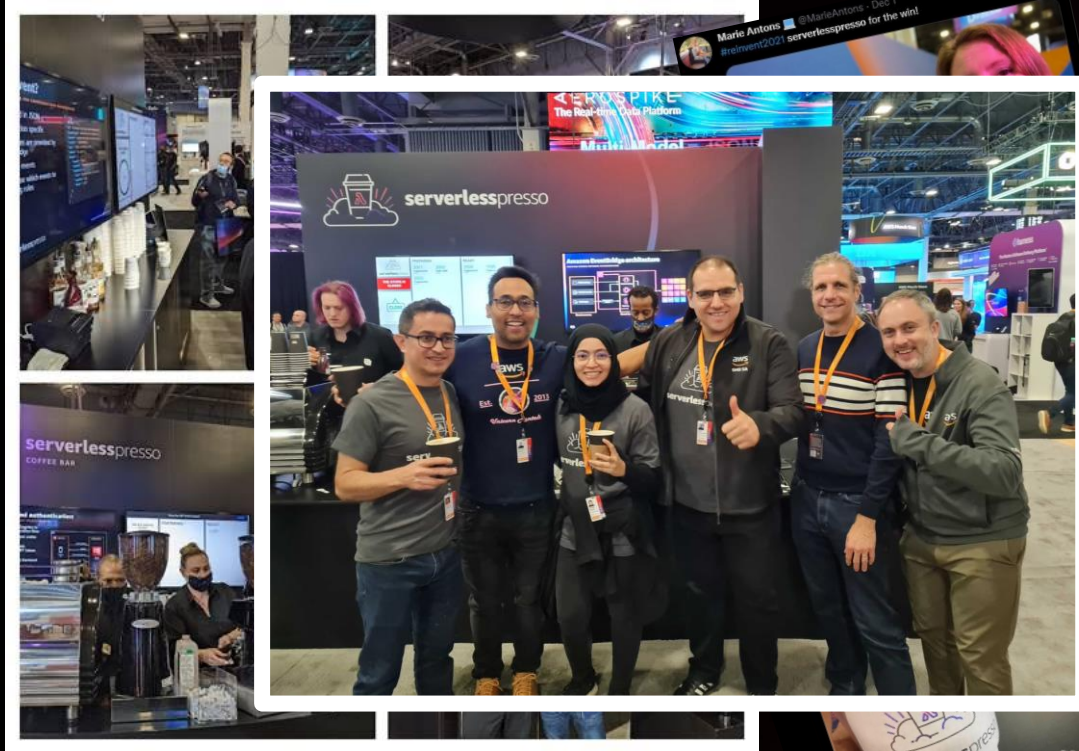


serverlesspresso

Una aplicación de pedidos de café orientada a eventos
construido con una arquitectura serverless



serverlesspresso



- 1,920 bebidas en re:Invent 2021
- 71 bebidas por hora
- Mostrada en AWS Summits, EDA Day, GOTO y otros
- Promedio 1,000 bebidas por día

¿Qué es Serverlesspresso?

El flujo de trabajo

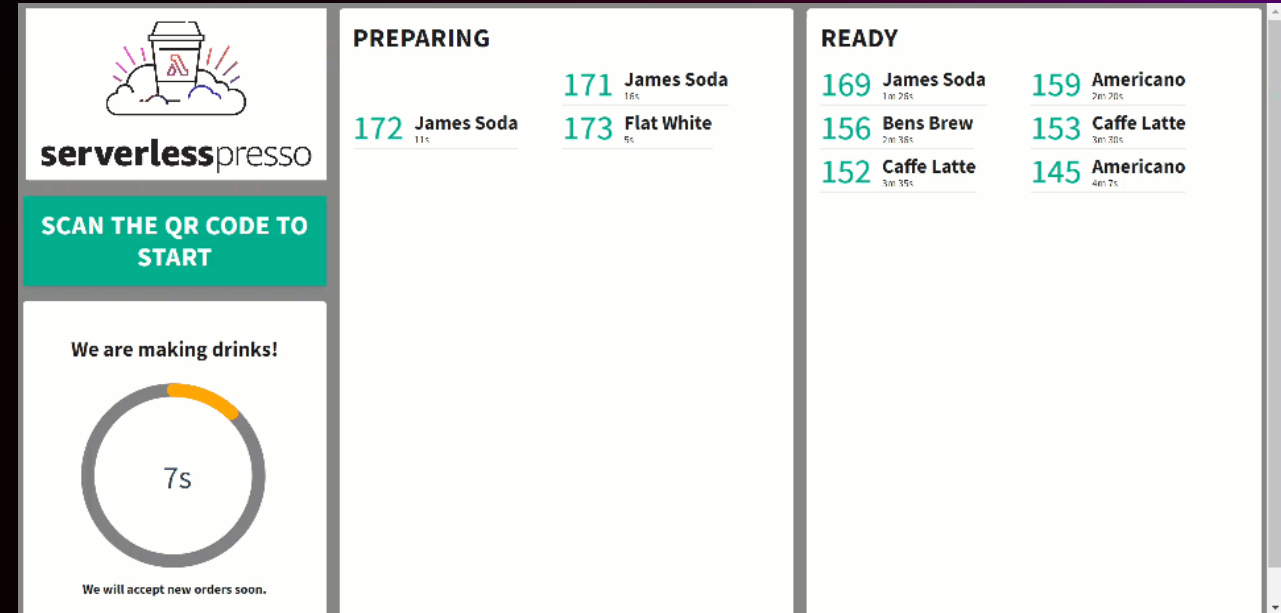
1. Escanea un código QR dinámico
2. Haz un pedido con tu dispositivo móvil
3. El pedido aparece en el monitor y en la app de la tablet del barista
4. Recibe una notificación cuando la bebida está lista



Aplicación Web de Visualización

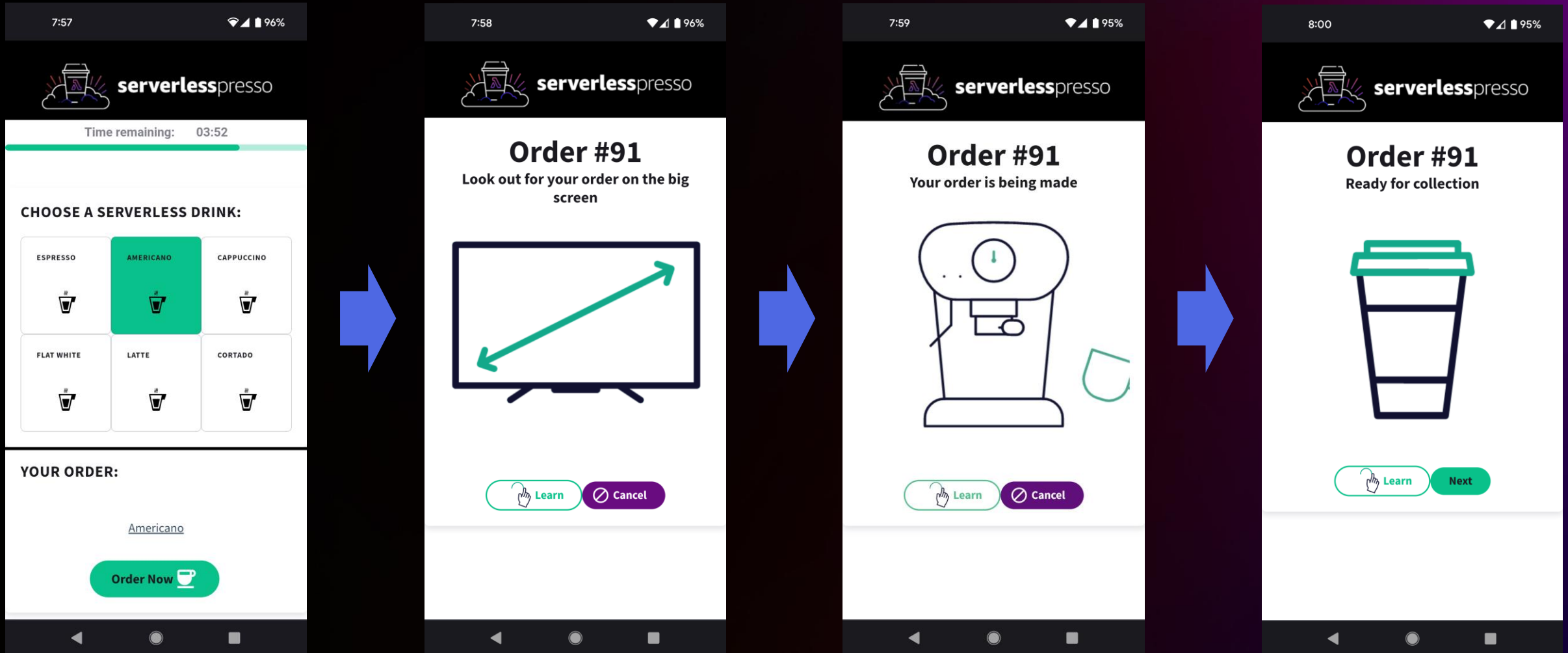
APLICACIÓN VUE.JS ALOJADAS EN AWS AMPLIFY CONSOLE

- Muestra nuevos código QR cada 5 minutos
- Recibe actualizaciones del estado del pedido
- Escucha eventos de apertura/cierre de tienda



Aplicación Web de Pedidos

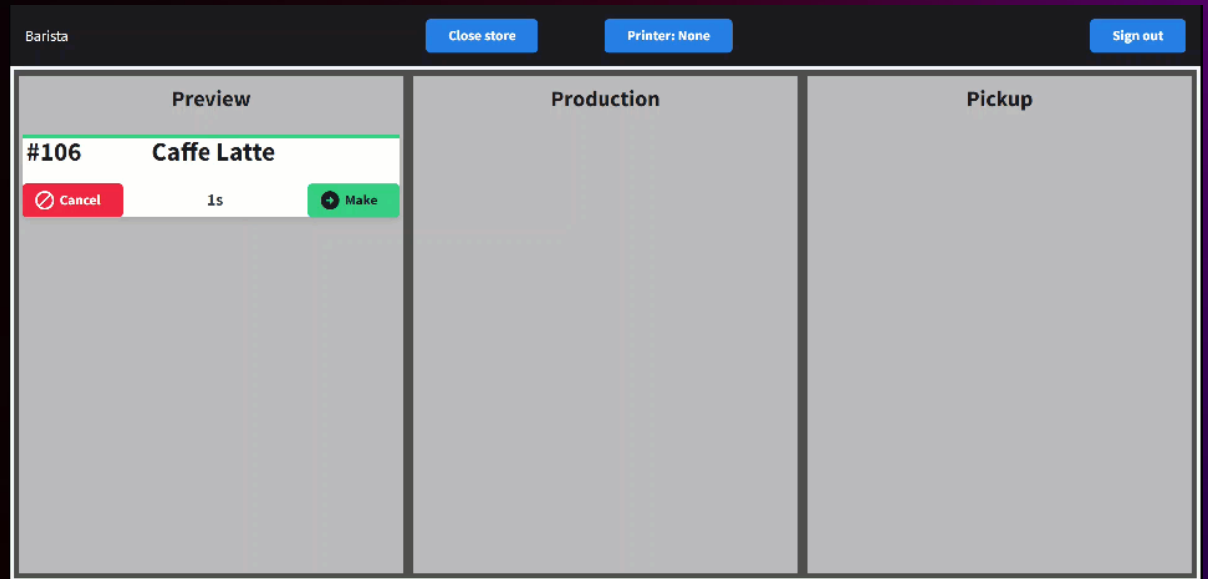
APLICACIÓN VUE.JS ALOJADA EN AWS AMPLIFY CONSOLE



Aplicación Web del Barista

APLICACIÓN VUE.JS ALOJADA EN AWS AMPLIFY CONSOLE

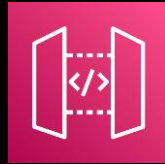
- Muestra los pedidos entrantes
- Permite a los baristas escoger los pedidos entrantes
- Permite completar o cancelar pedidos
- Permite almacenar eventos de apertura/cierre
- Imprime boletos



Servicios de AWS utilizados



AWS Amplify
console



Amazon
API Gateway



Amazon
DynamoDB



Amazon
EventBridge



AWS Step
Functions



AWS
IoT Core



AWS
Lambda

Demo



Decisiones de diseño

Lineamientos que utilizamos

Objetivos de la arquitectura

- Código mínimo
- Extensibilidad
- Escalabilidad
- Eficiencia en costos

Tenets

- Cada miembro del equipo es responsable de un componente
- No compartimos la implementación
- Cada microservicio tiene API/eventos – no se comparten los datos

Definiendo el flujo de trabajo

El proceso para preparar una bebida

- 1) El cliente escanea el código de barras
- 2) Comprobar que la tienda está abierta
- 3) Conseguir capacidad del barista
- 4) Espera el pedido del cliente — cancela si > 5 minutos
- 5) Generar un número de pedido
- 6) Espera a que el barista haga la bebida — cancela si > 15 min
- 7) Manejar cancelación por parte del cliente o barista



¿Escribirlo todo en (pseudo) código?

```
// Aceptación de pedido

if (isStoreOpen) {
  if (isBaristaBusy) {
    saveOrderToDynamoDB(timestamp)
  } else {
    rejectOrder('Barista busy')
  }
} else {
  rejectOrder('Store not open')
}
```

```
// Verificar tiempos de espera cada minuto

const orders = getAllOpenOrdersFromDDB
const now = currentTime

for (order in orders) {
  // ¿Se superó tiempo de espera de cliente?
  if (order.waitingForCustomer &&
      (now - order.timestamp) > 5 mins)
  {
    rejectOrder('Customer timed out')
    updateDynamoDB
  }

  // Luego verificar tiempo de espera de
  barista

  ...
}
```

En su lugar, diseña visualmente con Workflow Studio

REEMPLAZAR EL CÓDIGO ESPAGUETI CON MÁQUINAS DE ESTADO


Step Functions Workflow Studio [Info](#)


Search


Actions


Flow


MOST POPULAR

 AWS Lambda
Invoke


 Amazon SNS
Publish


 Amazon ECS
RunTask


 AWS Step Functions
StartExecution


 AWS Glue
StartJobRun


COMPUTE

 Amazon Data Lifecycle ...

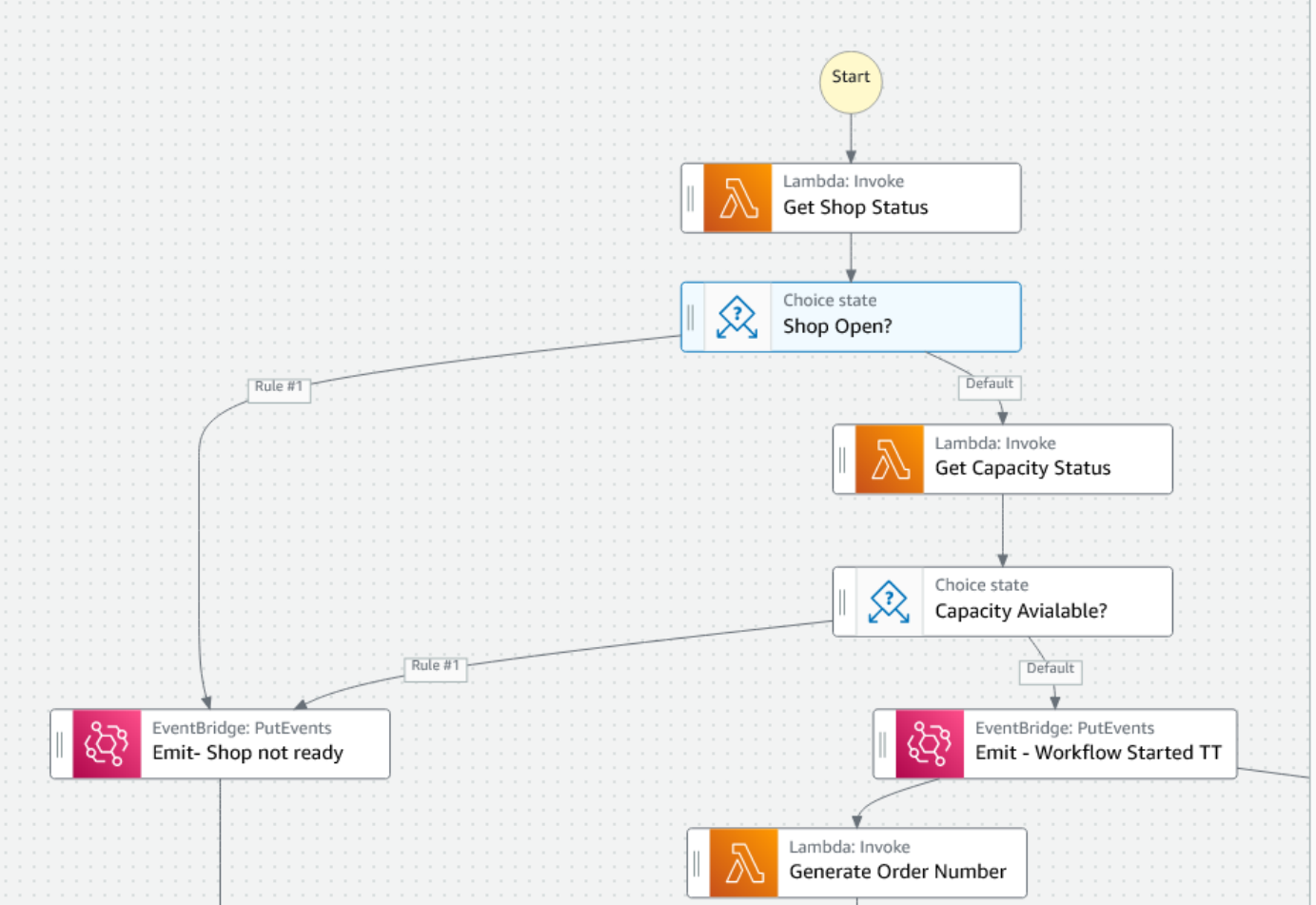
 Amazon EBS

 Amazon EC2

 AWS EC2 Instance Conn...

 Elastic Inference

Undo Redo Zoom in Zoom out Center



```
graph TD; Start((Start)) --> Lambda1[Lambda: Invoke  
Get Shop Status]; Lambda1 --> Choice1[Choice state  
Shop Open?]; Choice1 -- Rule #1 --> Event1[EventBridge: PutEvents  
Emit- Shop not ready]; Choice1 -- Default --> Lambda2[Lambda: Invoke  
Get Capacity Status]; Lambda2 --> Choice2[Choice state  
Capacity Available?]; Choice2 -- Rule #1 --> Event2[EventBridge: PutEvents  
Emit - Workflow Started TT]; Choice2 -- Default --> Lambda3[Lambda: Invoke  
Generate Order Number]; Event1 --> Lambda3; Event2 --> Lambda3;
```

Cancel **Apply and exit**

Export

Form

Definition

Shop Open?

Configuration

Input

Output

State name

Shop Open?

State type

Choice

Choice Rules

Choice rules let you create if-then logic to determine which state the workflow should transition to next.

Rule #1

not(\$.StoreOpen.modified.storeOpen == true)

Default rule

Defines default state when no rule evaluates to true

+ Add new choice rule

Comment - optional

check if Capacity is available

Gestionar el viaje de cada café

USO DE AWS STEP FUNCTIONS PARA ADMINISTRAR CADA EJECUCIÓN DE FLUJO DE TRABAJO

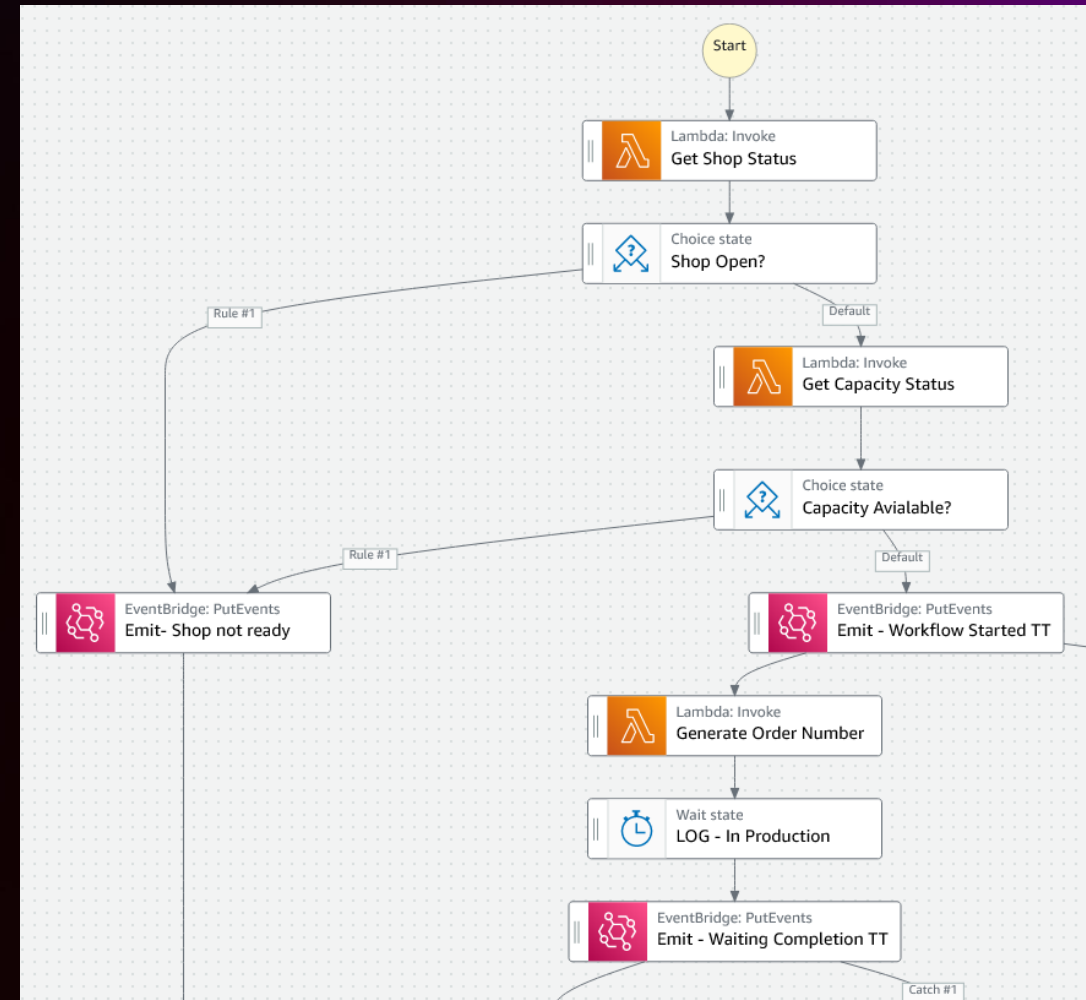
- Asegurar que la tienda esté abierta y los baristas tengan capacidad
- Permitir al cliente 5 minutos para ordenar, antes que se cancele por tiempo de espera
- Permitir que barista prepare en 15 minutos, antes de que se cancele por tiempo espera
- Utiliza funciones Lambda para lógica personalizada



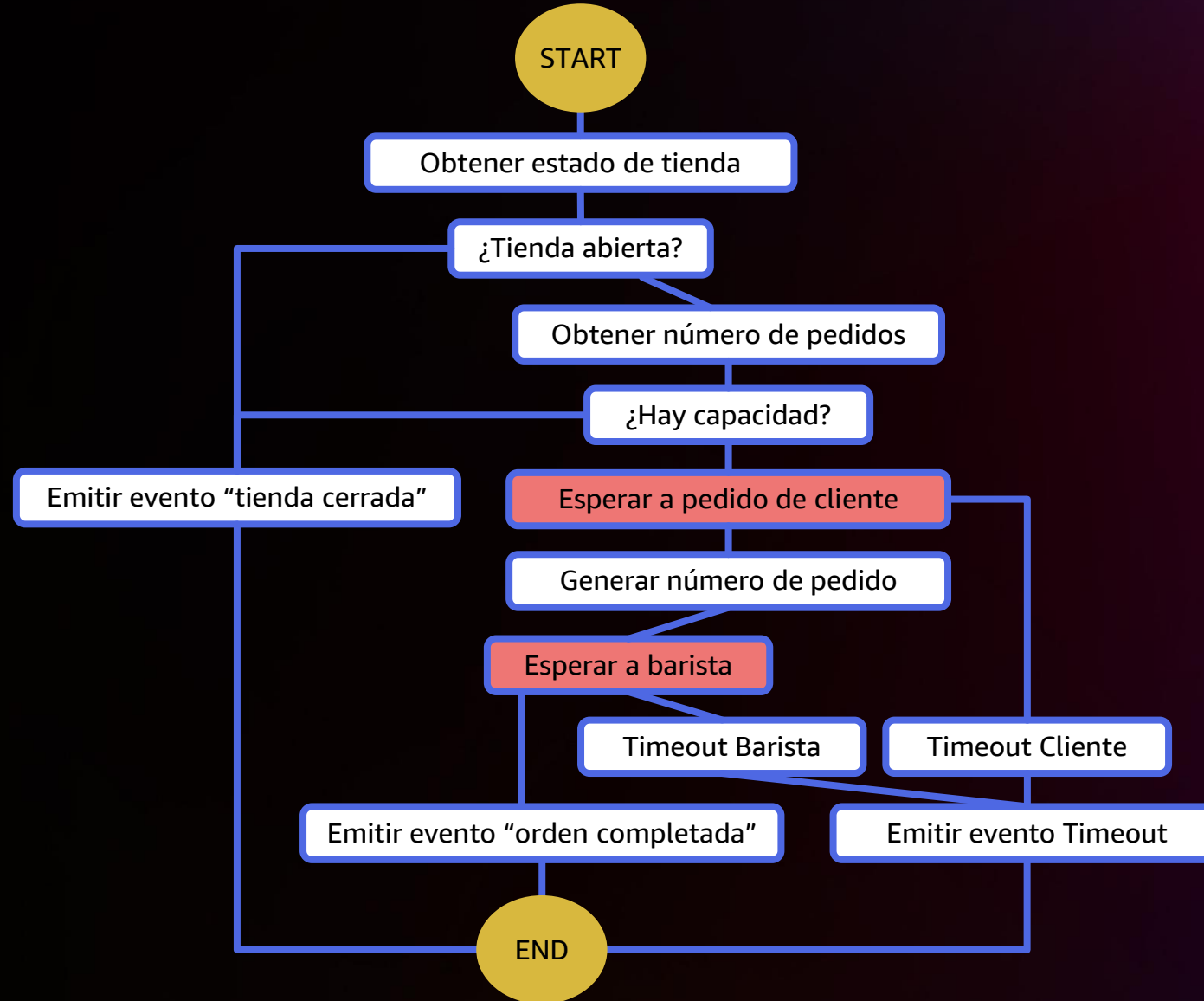
serverlesspresso



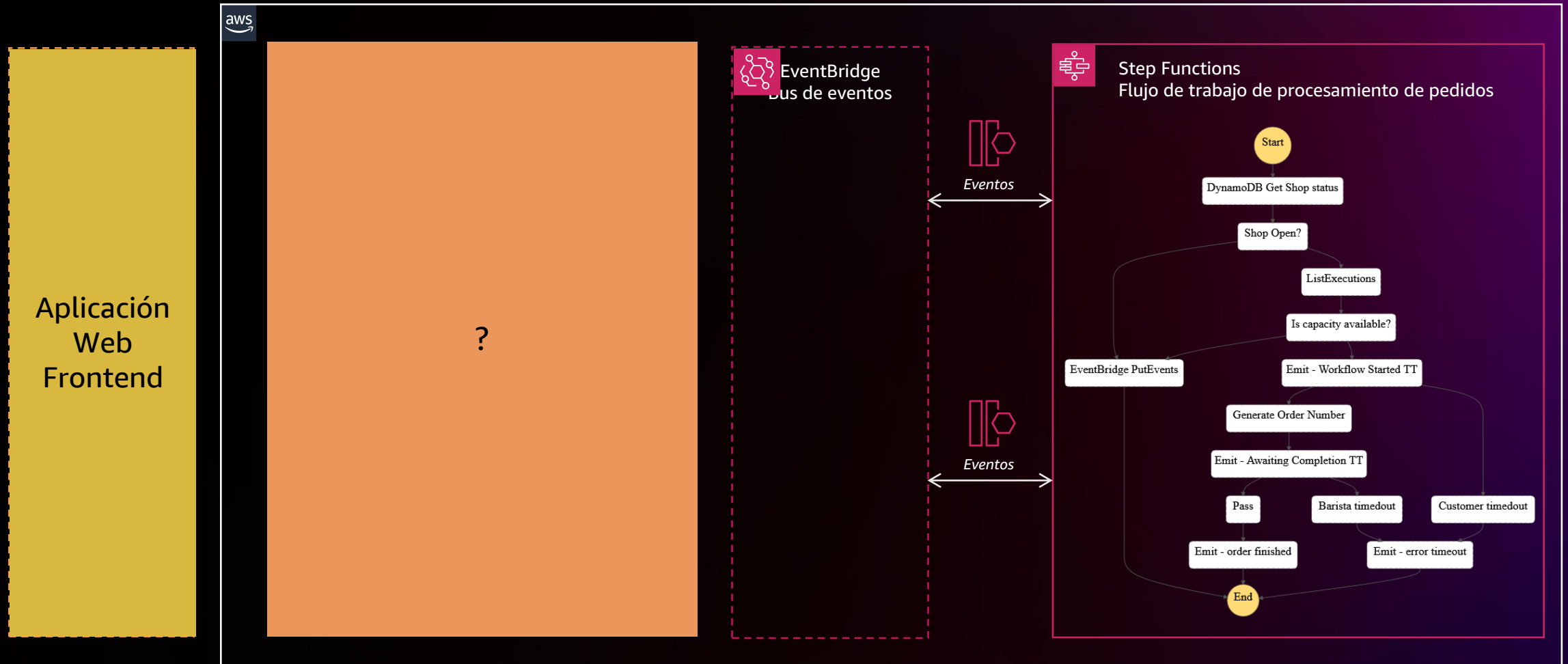
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.



La versión 1 del flujo de trabajo



Arquitectura hasta el momento



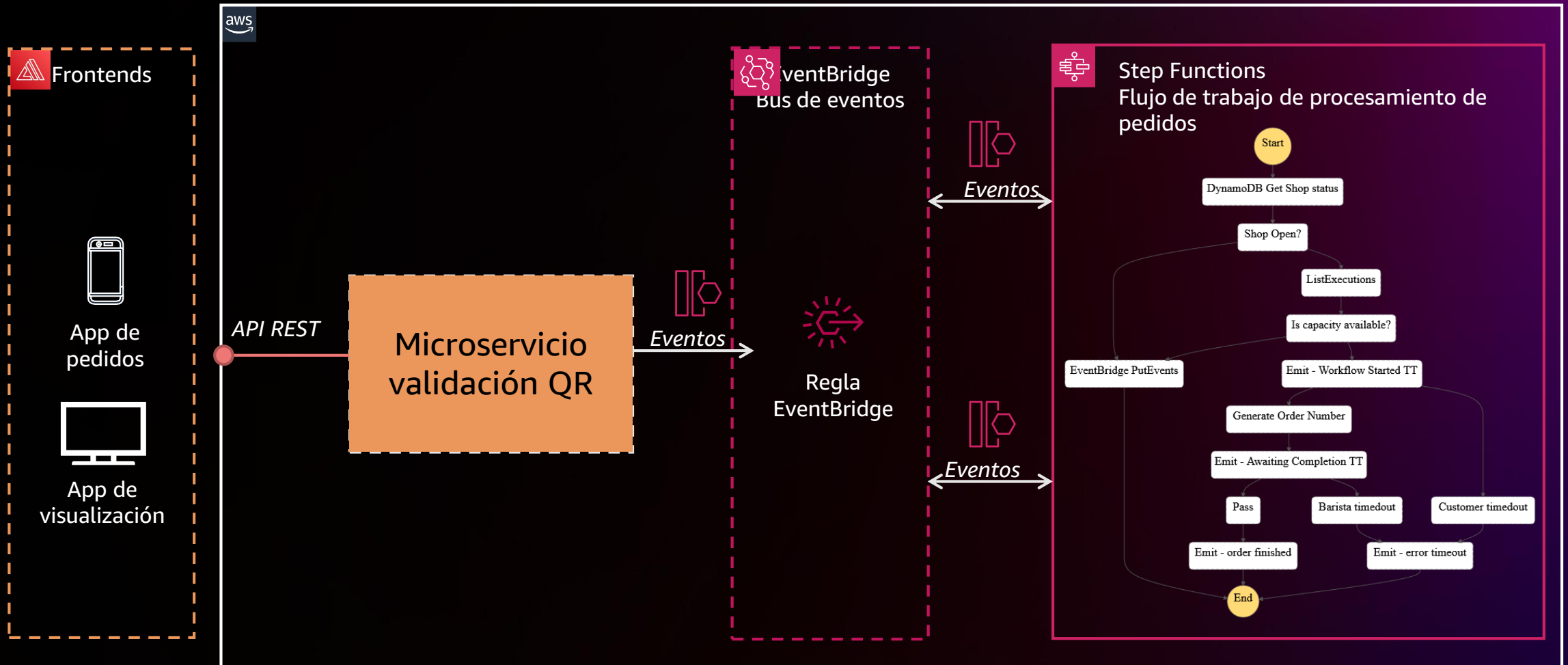
Iniciando el pedido

El microservicio de validación QR

- Limita la cola a 10 bebida por intervalo
- El intervalo es de 5 minutos por defecto
- Genera nuevos códigos QR
- QR desaparece una vez todos los escaneos han sido "utilizados"
- El escaneo exitoso inicia el pedido



Presentamos el microservicio de validación QR



Dentro del microservicio de validación QR

Generar

- App de visualización solicita nuevo código QR
- ID aleatorio almacenado en DynamoDB

Validar

- Teléfono escanea el código QR
- ID se envía como parámetro
- Devuelve éxito o fracaso
- Emite un evento si tiene éxito

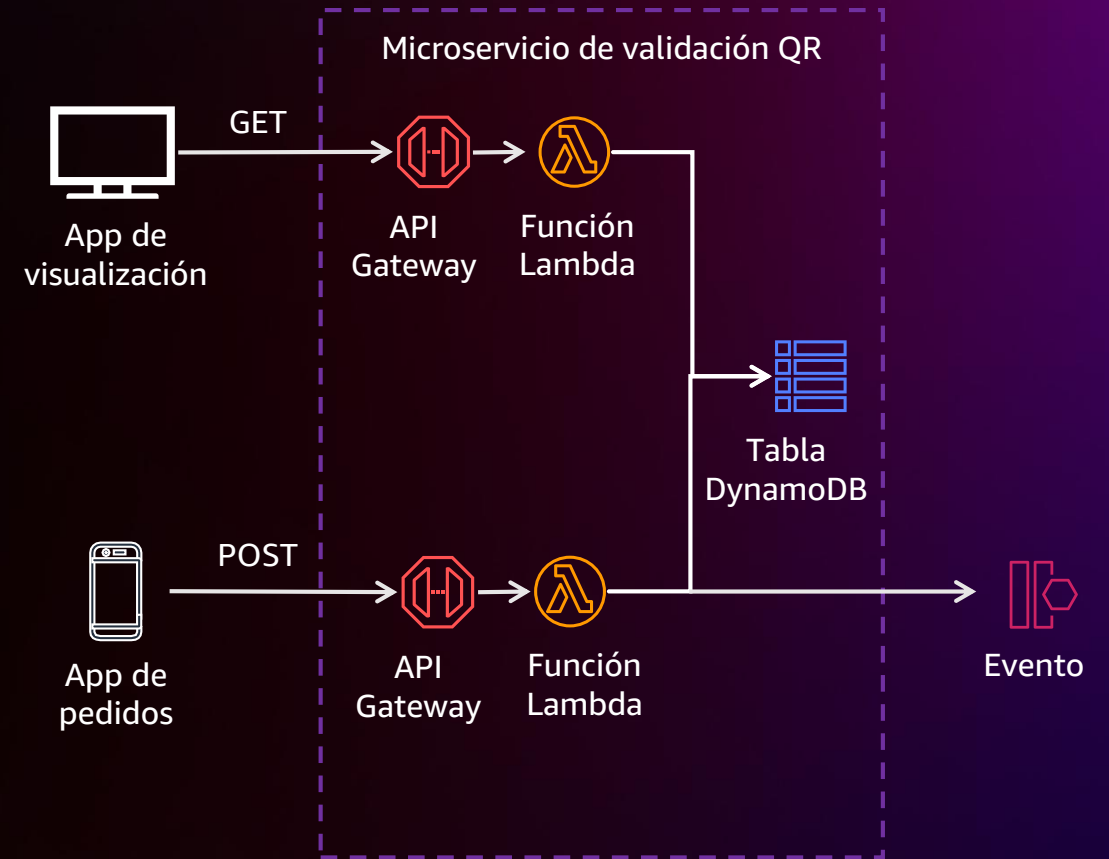


Tabla DynamoDB del validador QR

PK	Tokens disponibles	End_ts	Last_code	Last_id	Start_ts
1234423	10	1645200599999	41g_-KJHGT	1234423	1645200300000
1234123	8	1645628399999	6KJH_-FJ5Lh	1234123	1645628100000
5412322	1	1645449599999	91HHFFJHF	5412322	1645449300000
3435657	2	1645435199999	OCZomT756	3435657	1645434900000

Cuando se escanea un QR válido, se reducen los **Tokens disponibles**

Gestión de pedidos

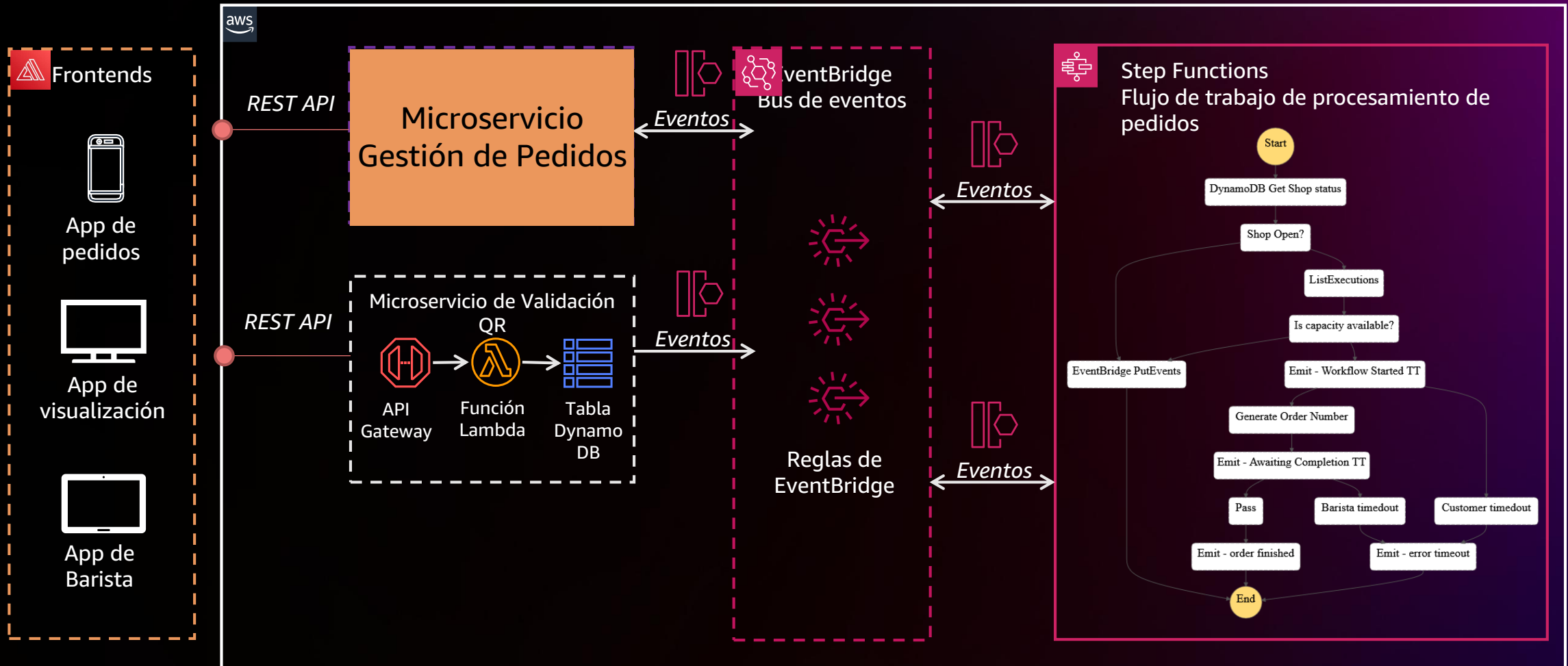
Necesidad de

- Actualizar y cancelar pedidos
- Almacenar y usar TaskTokens
- Obtener lista de pedidos abiertos o completados

Deberíamos

- ¿Construir un flujo de trabajo monolítico?
- ¿Mantener los TaskTokens en el cliente?
- ¿Consultar todos los flujos de trabajo abiertos?

Presentamos el microservicio Gestión de Pedidos



Actualización de pedidos



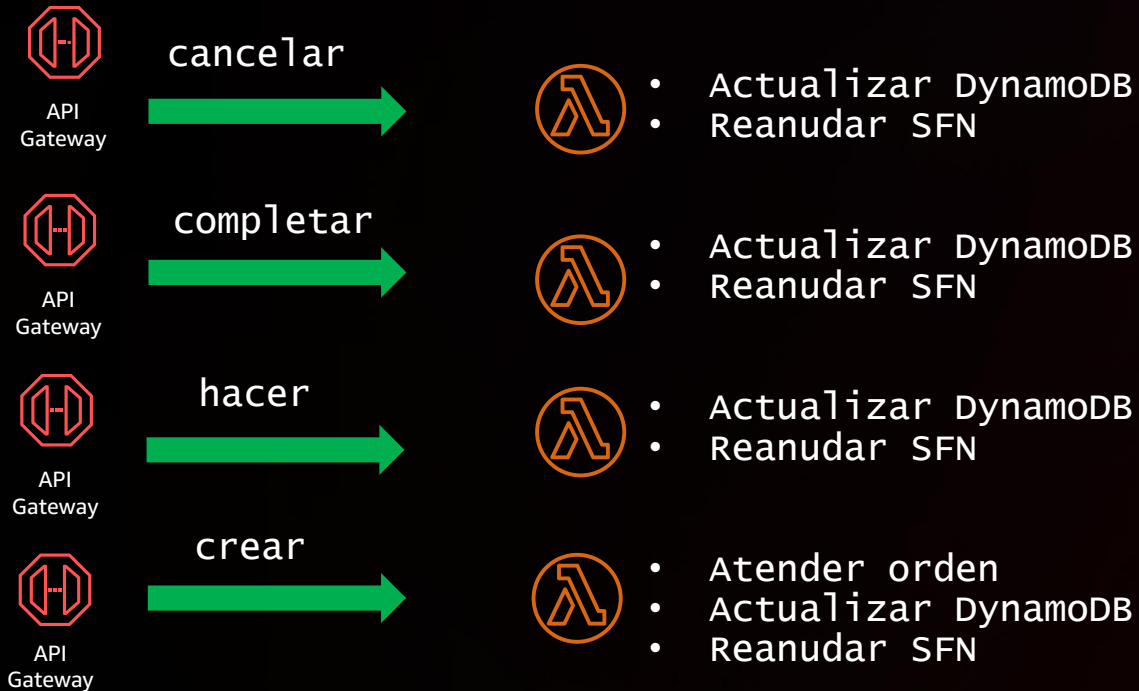
- Persiste cada orden en una tabla de DynamoDB
- Ítem actualizado en varias etapas del ciclo de vida del pedido
- Responsable de administrar tokens de tareas
- GSI en el atributo OrderState para consultar pedidos abiertos/completados

PK	SK	TS	UID	OrderNo	TaskToken	OrderState	DrinkOrder
Pedidos	2	1645726 346199	1	10	AAAAKgAAAAIAAAAAAAAAA Alp4um0gPw/FO3rqpCDvIE AL+l/h+	COMPLETED	{"userId":"1","drink":"Cappuccino","modifiers":[] ,"icon":"barista-icons_cappuccino-alternative"}

Microservicio Gestión de Pedidos

Versión 1

Cada operación invoca una función Lambda



La aplicación se hizo más compleja con el tiempo, realizando múltiples tareas para manejar una lógica de negocio cada vez más compleja

Lo que llevó...

- Código estrechamente acoplado
- Cadencia de liberación más lenta
- Mala capacidad de descubrimiento
- Complejidad adicional

Microservicio Gestión de Pedidos: Integración directa

Versión 1.5

/misPedidos - GET

/pedidos - GET

/pedidos/ {id} - GET

Consulta de front-ends a tabla
Orden directamente desde API
Gateway

Las plantillas de mapeo
modifican la solicitud entrante



```
#set($subFromJWT = $context.authorizer.claims.sub)

{
  "TableName": "serverlesspresso-order-table",
  "IndexName": "GSI-userId",
  "KeyConditionExpression": "#USERID = :USERID",
  "ExpressionAttributeNames": {
    "#USERID": "USERID"
  },
  "ExpressionAttributeValues": {
    ":USERID": {
      "S": "$subFromJWT"
    }
  },
  "ScanIndexForward": true,
  "ProjectionExpression": "PK, SK, orderNumber, robot, drinkOrder, ORDERSTATE, TS"
}
```


Microservicio Gestión de Pedidos: Operaciones CRUD

Versión 1.5

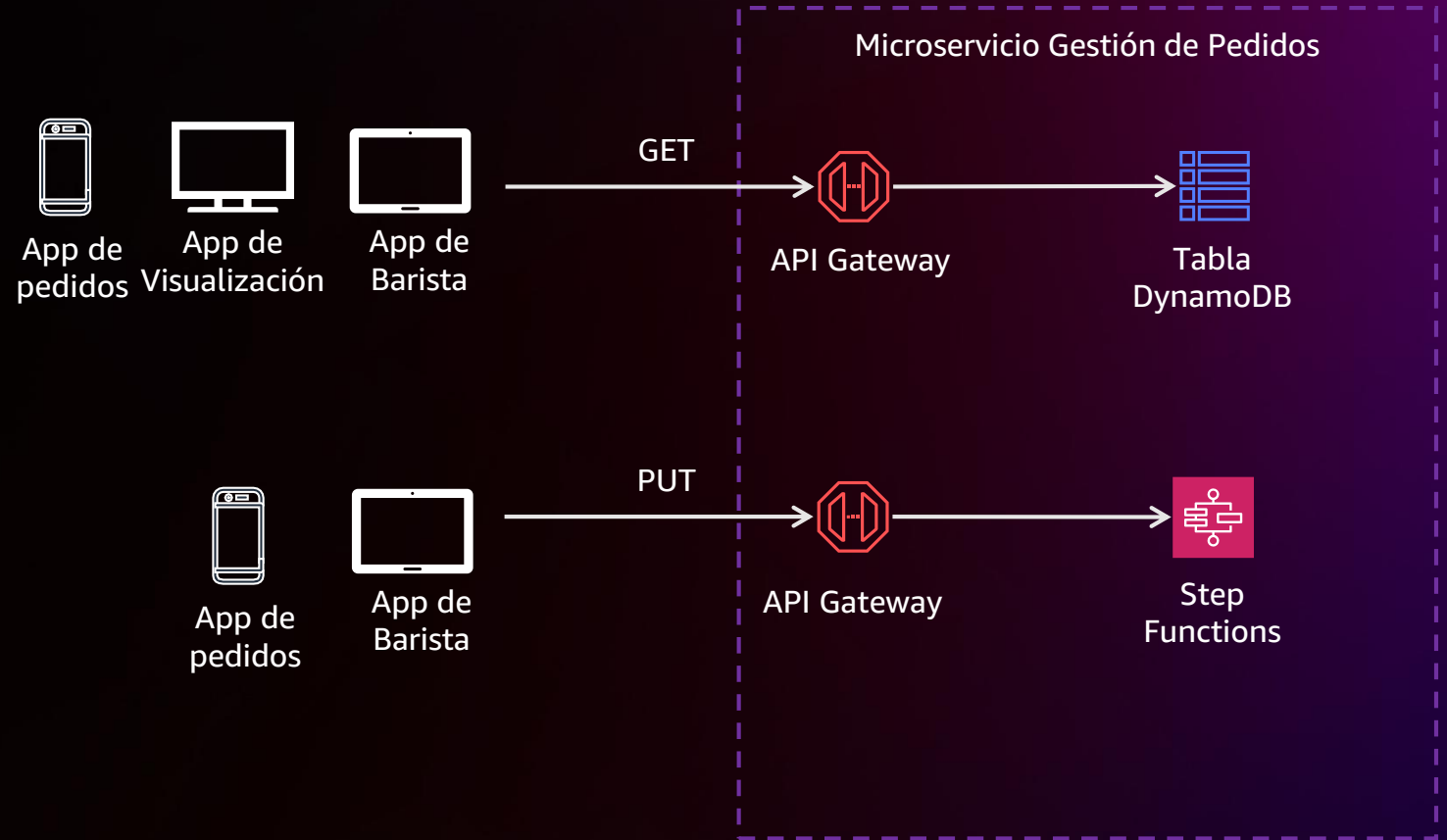
/misPedidos - GET

/pedidos - GET

/pedidos/ {id} - GET

/pedidos/ {id} - PUT

Actualizaciones realizadas a tabla
Orden via una solicitud PUT



Microservicio Gestión de Pedidos como flujo de trabajo

Versión 2

Un único endpoint de API Gateway inicia un flujo de trabajo de Step Functions



Notificar a las aplicaciones web

Necesidad de

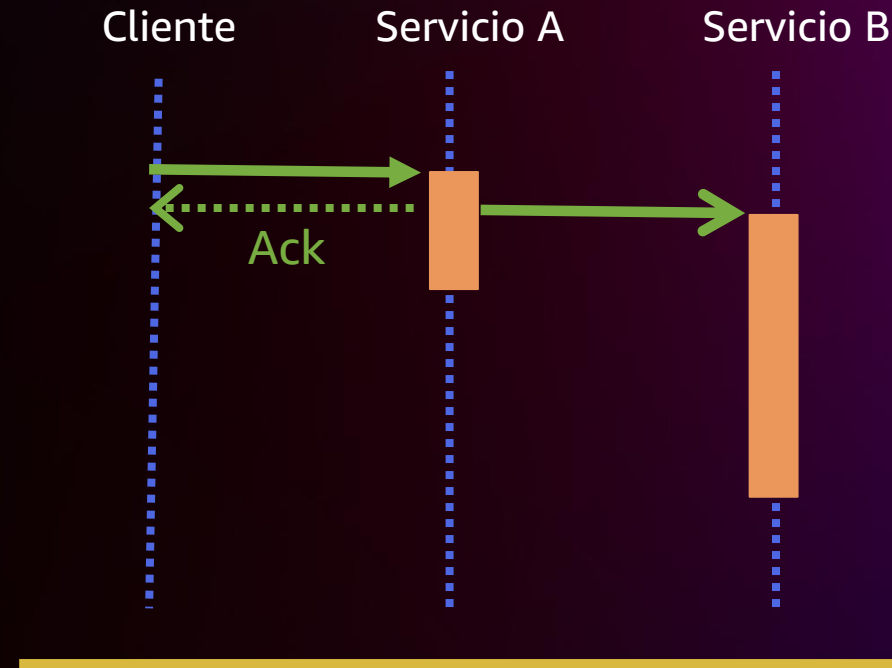
- Mantener las aplicaciones web sincronizadas con el estado del pedido
- Responder a eventos globales, como apertura/cierre de tienda
- Aproximar a tiempo real sin refrescarse
- Ser resiliente, cuando haya degradación de la red móvil

Deberíamos

- ¿Crear APIs de polling?
- ¿Usar mecanismos como SMS?
- ¿Usar API Gateway WebSockets?

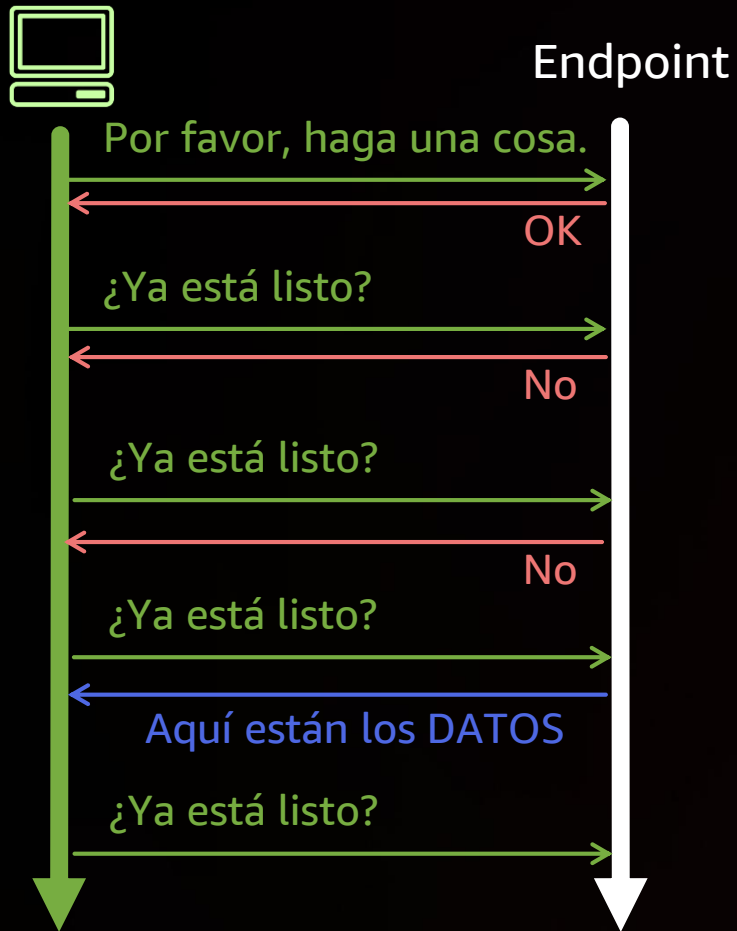
Manejo de respuesta y estado para solicitudes asíncronas

No hay ruta de retorno para proveer más información, más allá del reconocimiento inicial



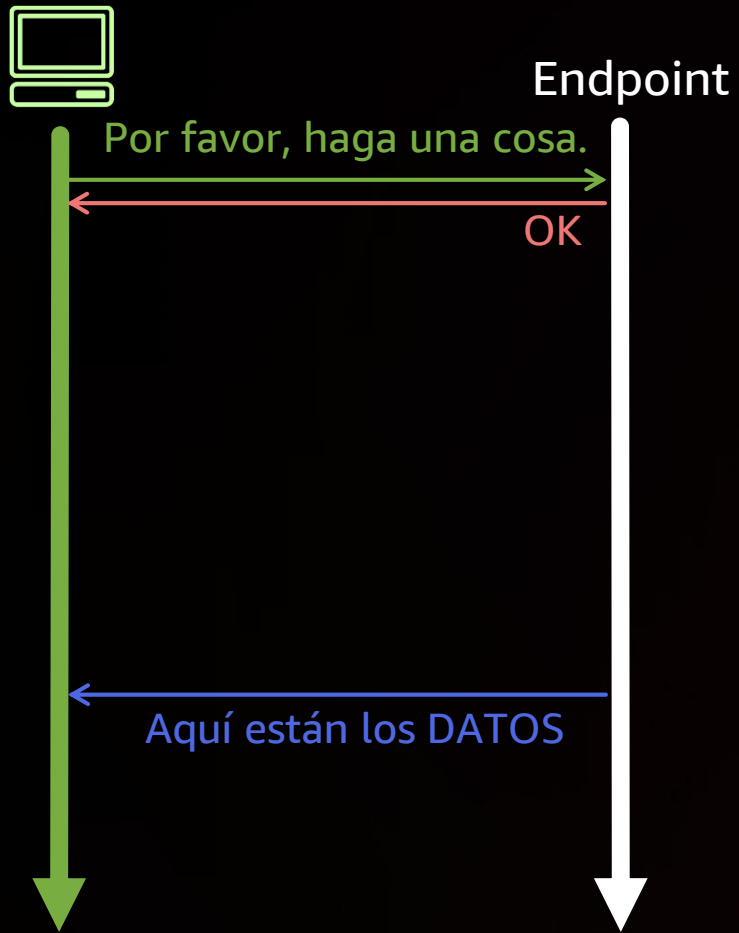
Eventos
Asíncronos

Seguimiento de una solicitud en progreso: Polling



- La solicitud inicial devuelve un identificador de seguimiento
- Crear un segundo endpoint de API para el front-end para verificar el estado de la solicitud, con referencia al ID de seguimiento
- Utilice DynamoDB para rastrear el estado de la solicitud
- Mecanismo simple para implementar
- Puede crear muchas llamadas vacías
- Retraso entre la disponibilidad y la notificación front-end

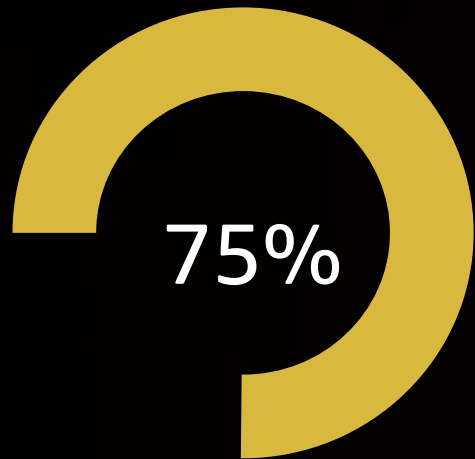
Seguimiento de una solicitud en progreso: WebSockets



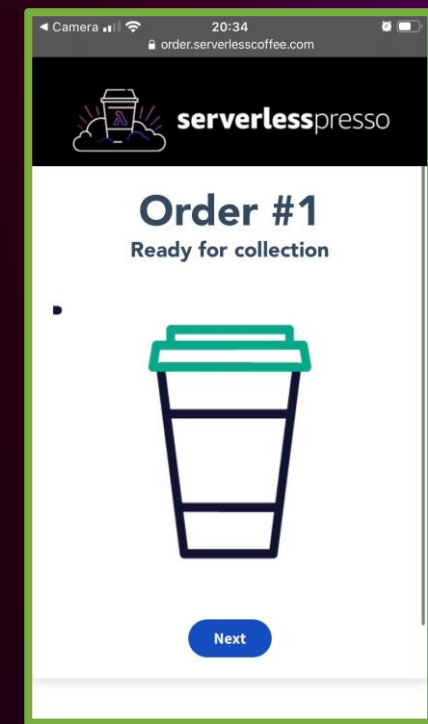
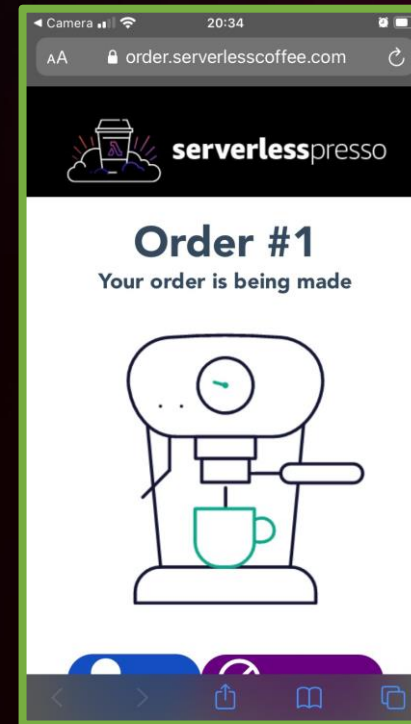
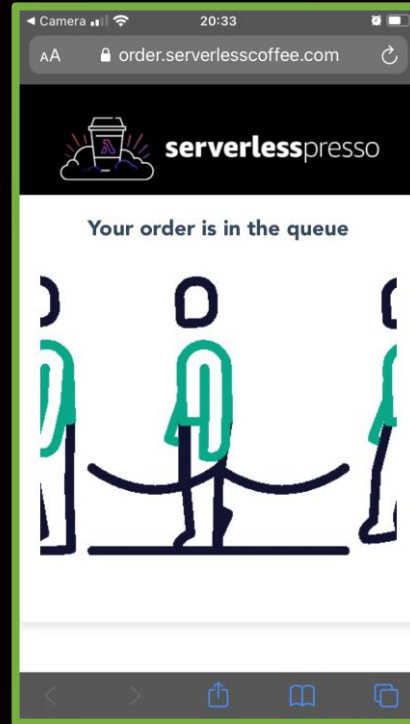
- Una conexión bidireccional entre el cliente front-end y el servicio backend
- Sus servicios de backend pueden continuar enviando datos al cliente mediante una conexión WebSocket
- Más cerca del tiempo real
- Reduce el número de mensajes entre el cliente y el sistema backend
- A menudo más complejo de implementar

Uso de AWS IoT Core para mensajería en tiempo real

LAS APLICACIONES WEB A MENUDO REQUIEREN INFORMACIÓN PARCIAL



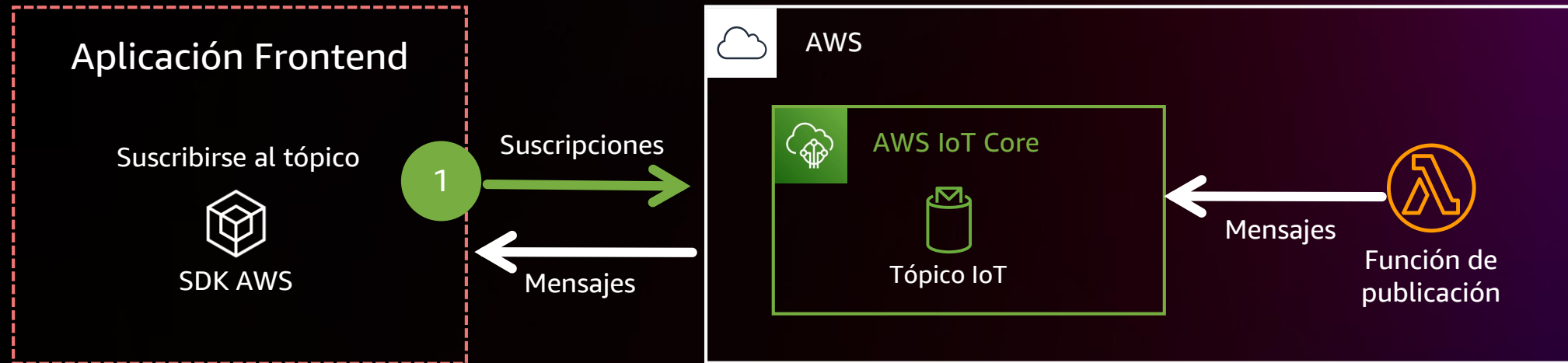
Porcentaje de completado



Cambios continuos de datos

Uso de AWS IoT Core para mensajería en tiempo real

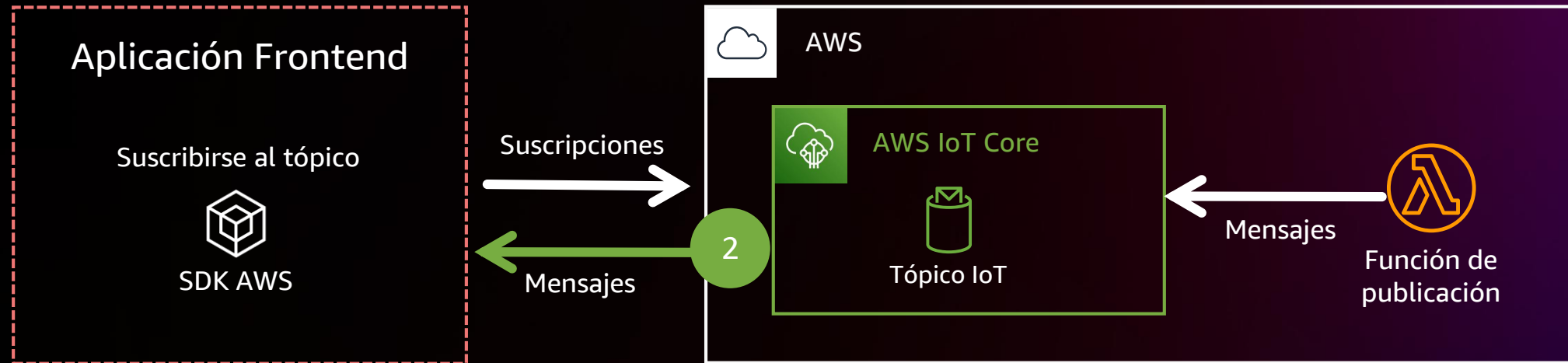
La aplicación frontend utiliza pub-sub para “escuchar” las actualizaciones de eventos



Frontend utiliza AWS SDK para suscribirse a un tópico basado en el ID único del usuario

Uso de AWS IoT Core para mensajería en tiempo real

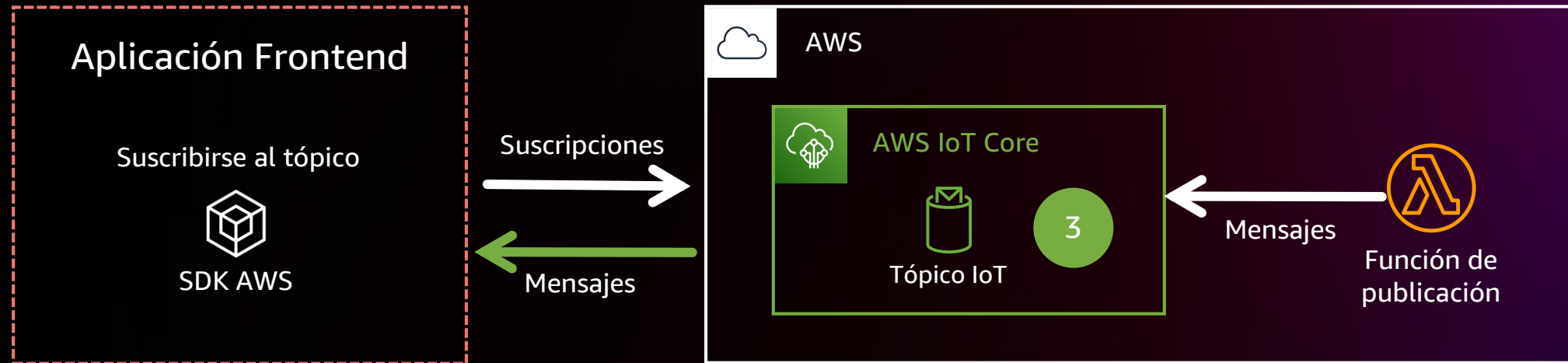
La aplicación Frontend utiliza pub-sub para “escuchar” las actualizaciones de eventos



Recibe mensajes publicados por el backend sobre este tópico

Uso de AWS IoT Core para mensajería en tiempo real

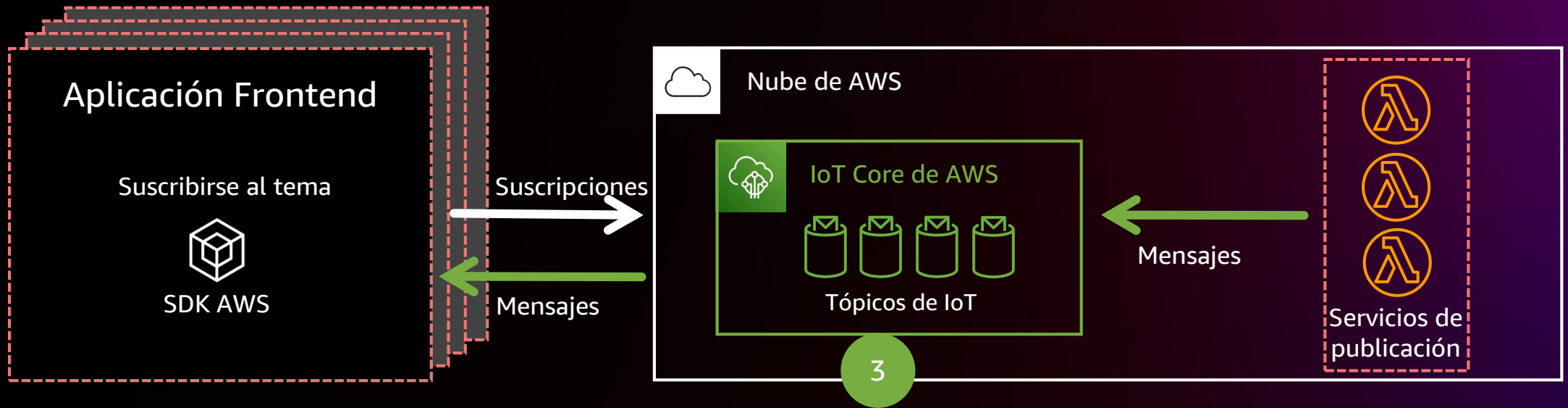
La aplicación Frontend utiliza pub-sub para “escuchar” las actualizaciones de eventos



Los mensajes se categorizan usando tópicos

Uso de AWS IoT Core para mensajería en tiempo real

La aplicación Frontend utiliza pub-sub para “escuchar” las actualizaciones de eventos



El servicio AWS IoT Core gestiona la conexión WebSocket entre publicadores y suscriptores

Esto permite la funcionalidad de **fanout** a miles de dispositivos frontend

Uso de AWS IoT Core para mensajería en tiempo real

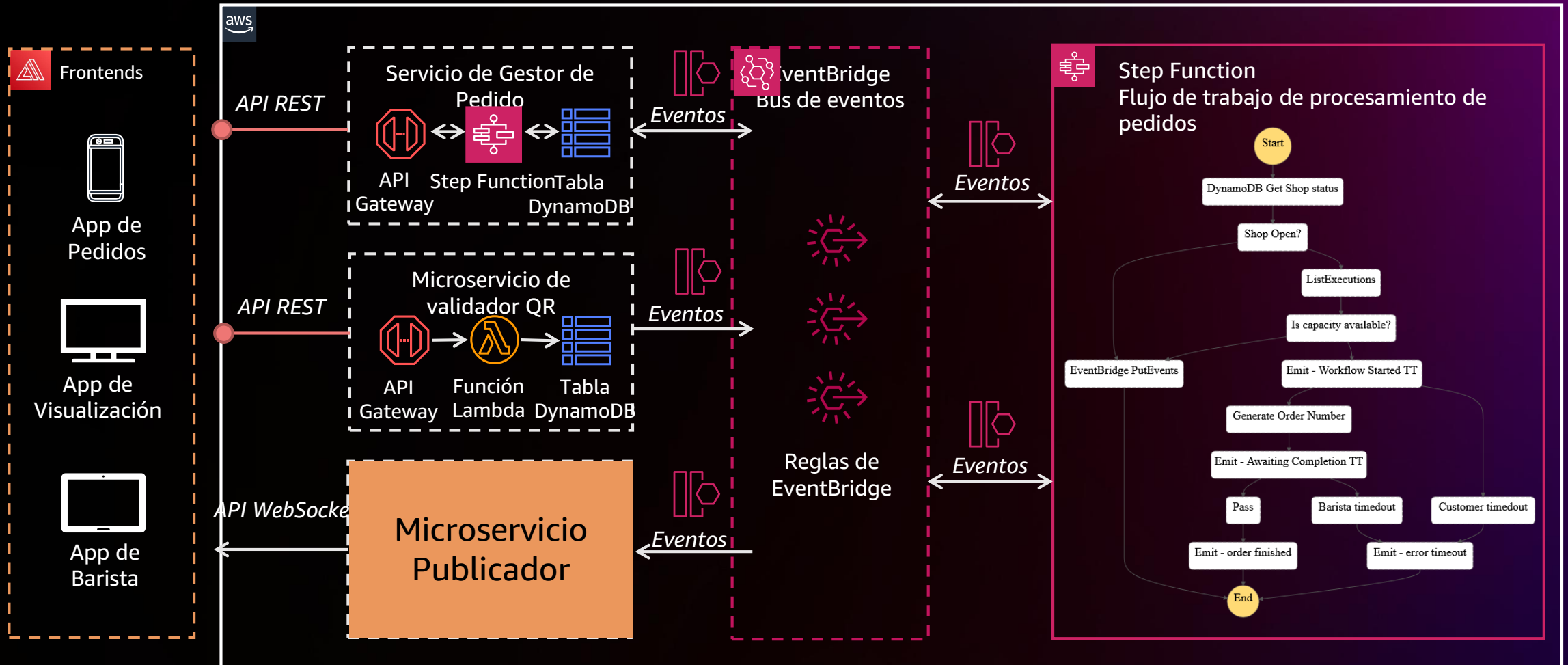
La clase *IotData* en el SDK de AWS devuelve un cliente que utiliza el protocolo MQTT. Ejemplos de manejadores en el frontend:

```
mqttClient.on('connect', function () {  
    console.log('mqttClient connected')  
})
```

```
mqttClient.on('error', function (err) {  
    console.log('mqttClient error: ', err)  
})
```

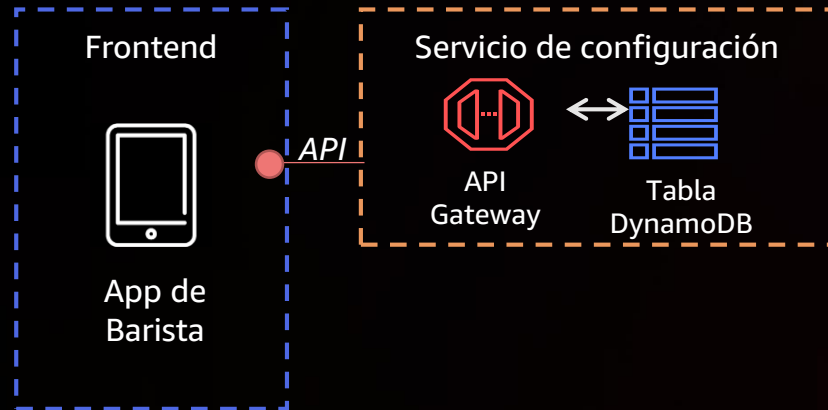
```
mqttClient.on('message', function (topic, payload) {  
    const msg = JSON.parse(payload.toString())  
    console.log('IoT msg: ', topic, msg)  
})
```

Presentamos el microservicio Publicador



Combinando múltiples enfoques para su aplicación frontend

Muchas aplicaciones frontend pueden combinar modelos de respuesta síncrona y asíncrona



Serverlesspresso envía una solicitud síncrona inicial para recuperar el “estado” actual

Combinando múltiples enfoques para su aplicación frontend

Muchas aplicaciones frontend pueden combinar modelos de respuesta síncrona y asíncrona



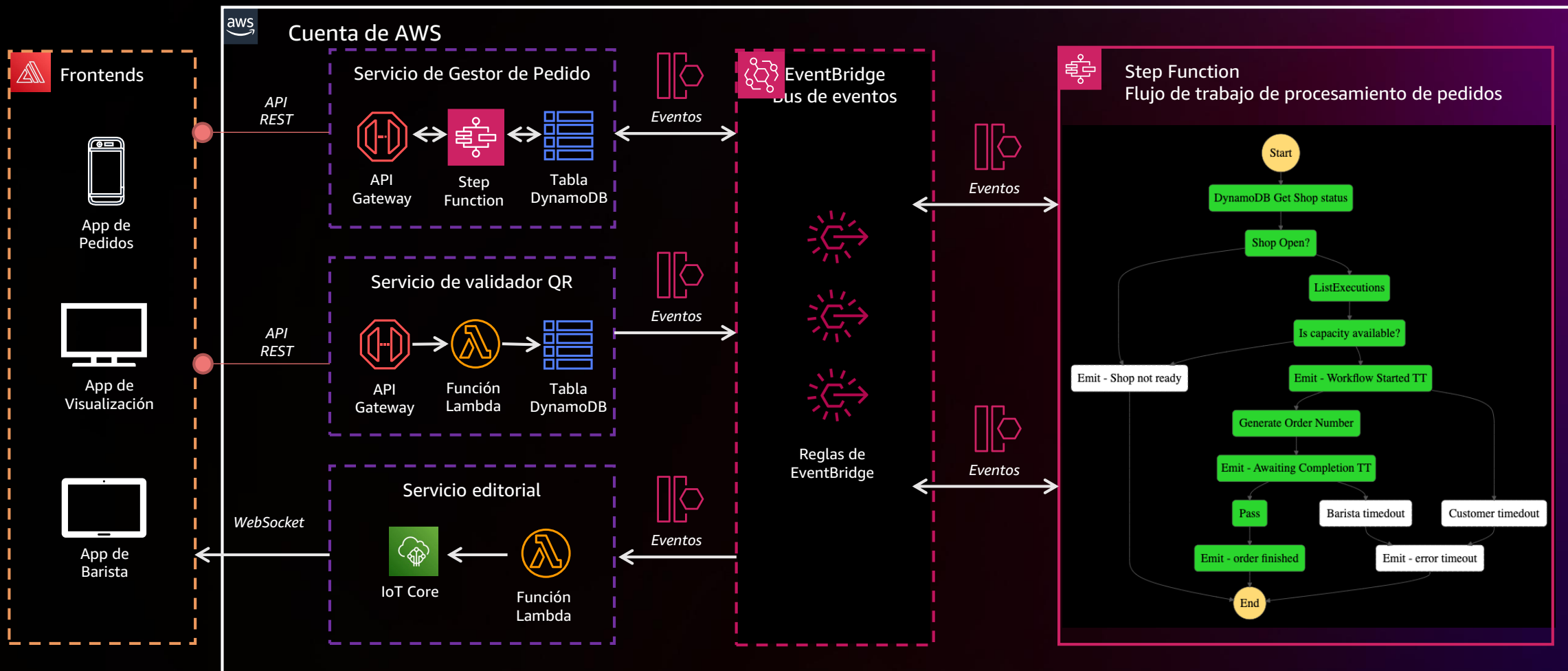
Simultáneamente, el frontend se suscribe a tópicos IoT globales y basados en el usuari

Combinando múltiples enfoques para su aplicación frontend



Se publican nuevos pedidos en la aplicación, procesados de forma asincrónica, con actualizaciones publicadas en el frontend a través del tema IoT

Arquitectura final



Lecciones aprendidas



¿Qué deben contener los eventos?

- Productores crean eventos
- Un evento se define en JSON
- Amazon EventBridge proporciona los atributos del evento
- detalle, tipo de detalle, y fuente son enteramente tu elección
- ¿Eventos gordos o eventos delgados?
- Incluir metadatos como versionamiento

```
{
  "version": "0",
  "id": "6ac4e27b-1234-1234-1234-5fb02c880319",
  "detail-type": "?????",
  "source": "?????",
  "account": "123456789012",
  "time": "2021-11-28T13:12:30Z",
  "region": "us-west-2",
  "detail": {
    ?
  }
}
```

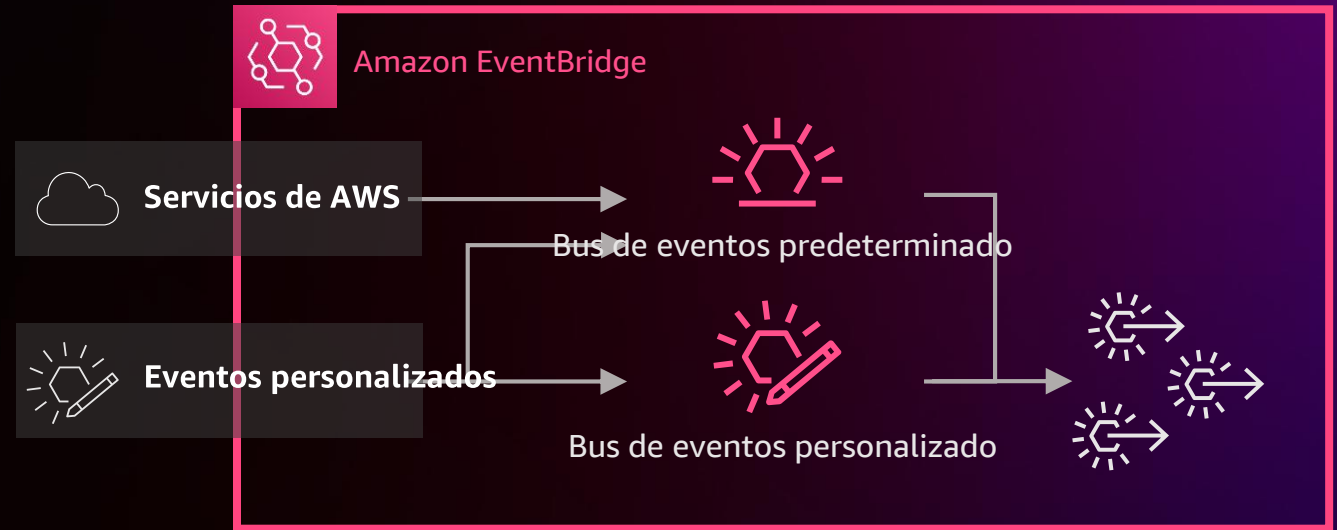
¿Qué eventos debes generar?

- Publicar más eventos de los que usa actualmente ayuda con la extensibilidad
- Step Functions genera algunos eventos automáticamente; también puede emitir eventos manualmente
- Convenciones de nomenclatura:
`Microservicio.CosaQuePasó`

```
ConfigService.ConfigChanged  
OrderJourney.AllEventsStored  
OrderManager.MakeOrder  
OrderManager.OrderCancelled  
OrderManager.OrderCompleted  
OrderManager.WaitingCompletion  
OrderProcessor.OrderTimeout  
OrderProcessor.ShopNotready  
OrderProcessor.WaitingCompletion  
OrderProcessor.WaitingProduction  
OrderProcessor.WorkflowStarted  
QueueService.OrderCancelled  
QueueService.OrderCompleted  
QueueService.OrderStarted  
Validator.NewOrder
```

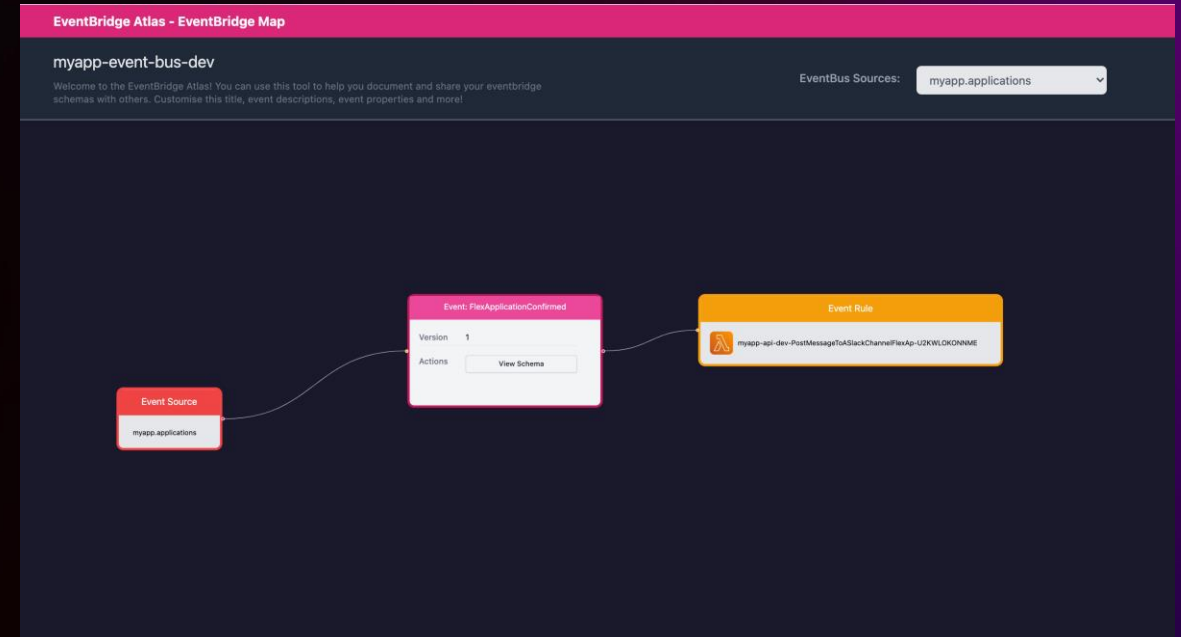

Amazon EventBridge: ¿Un bus de eventos o dos?

- El bus predeterminado ya existe, solo destino para eventos de AWS
- Agregar eventos aquí ayuda al descubrimiento y la extensibilidad
- El bus personalizado puede actuar como un límite de seguridad
- Trade-off: es posible que otros equipos no puedan construir sus eventos



Ayudando al descubrimiento de eventos

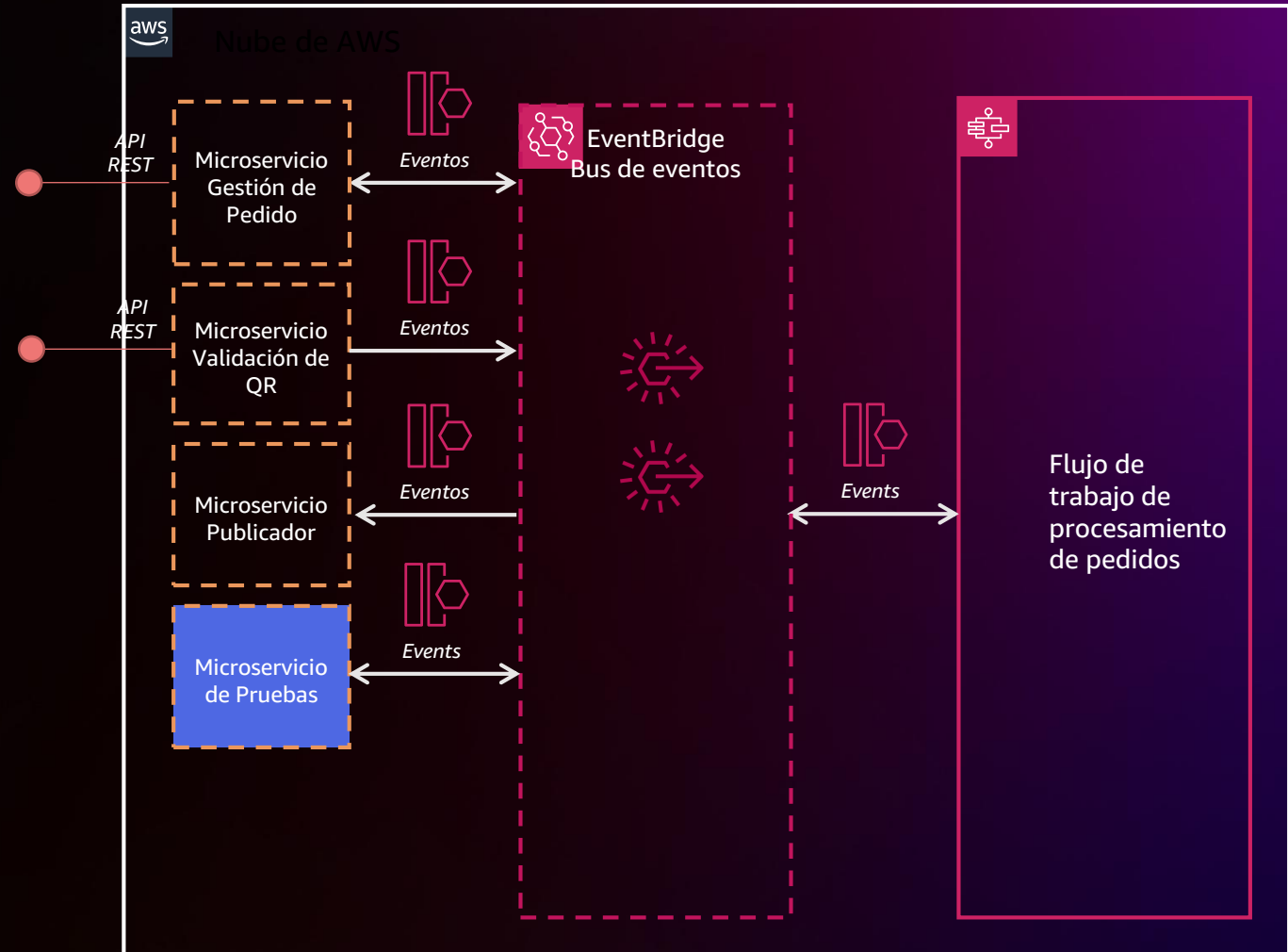
- Los eventos cambiarán en el desarrollo. Use versionamiento para implementar contratos.
- Use EventBridge Schema Registry para realizar un seguimiento de todos sus eventos
- Use herramientas de código abierto como EventBridge Atlas para visualizar y documentar flujos
- Empezar a documentar eventos desde el principio



Atlas EventBridge: <https://eventbridge-atlas.netlify.app/>

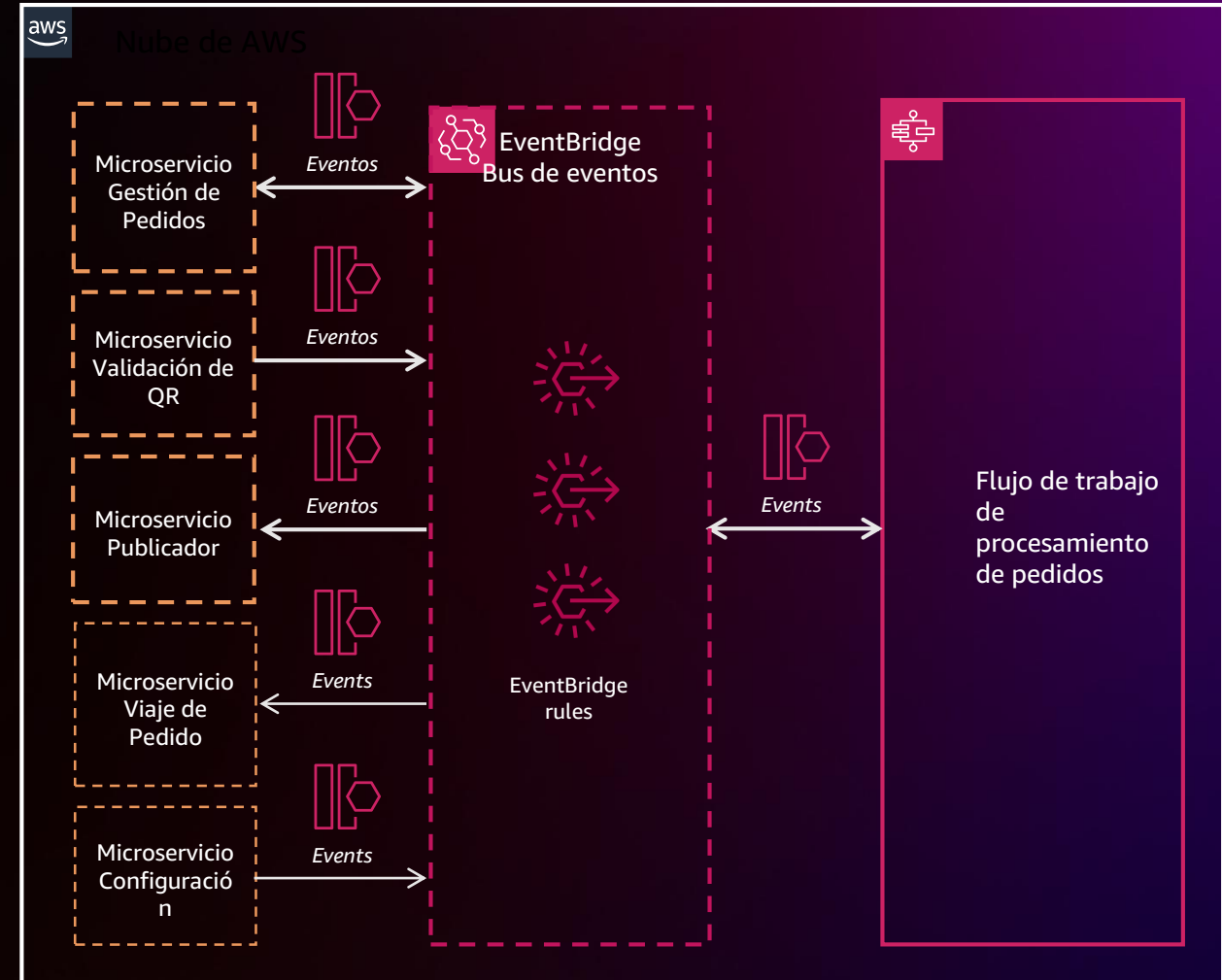
Pruebas con inyección de eventos

- En lugar de llamar a APIs, pon eventos en el bus
- Nuestro Robot Tester es otro microservicio: genera y completa pedidos a escala
- Usar archive/repetición para probar servicios en dev con datos reales
- Cree una regla para enviar todos los eventos a CloudWatch Logs (en dev/test). Lee los logs con el CLI.



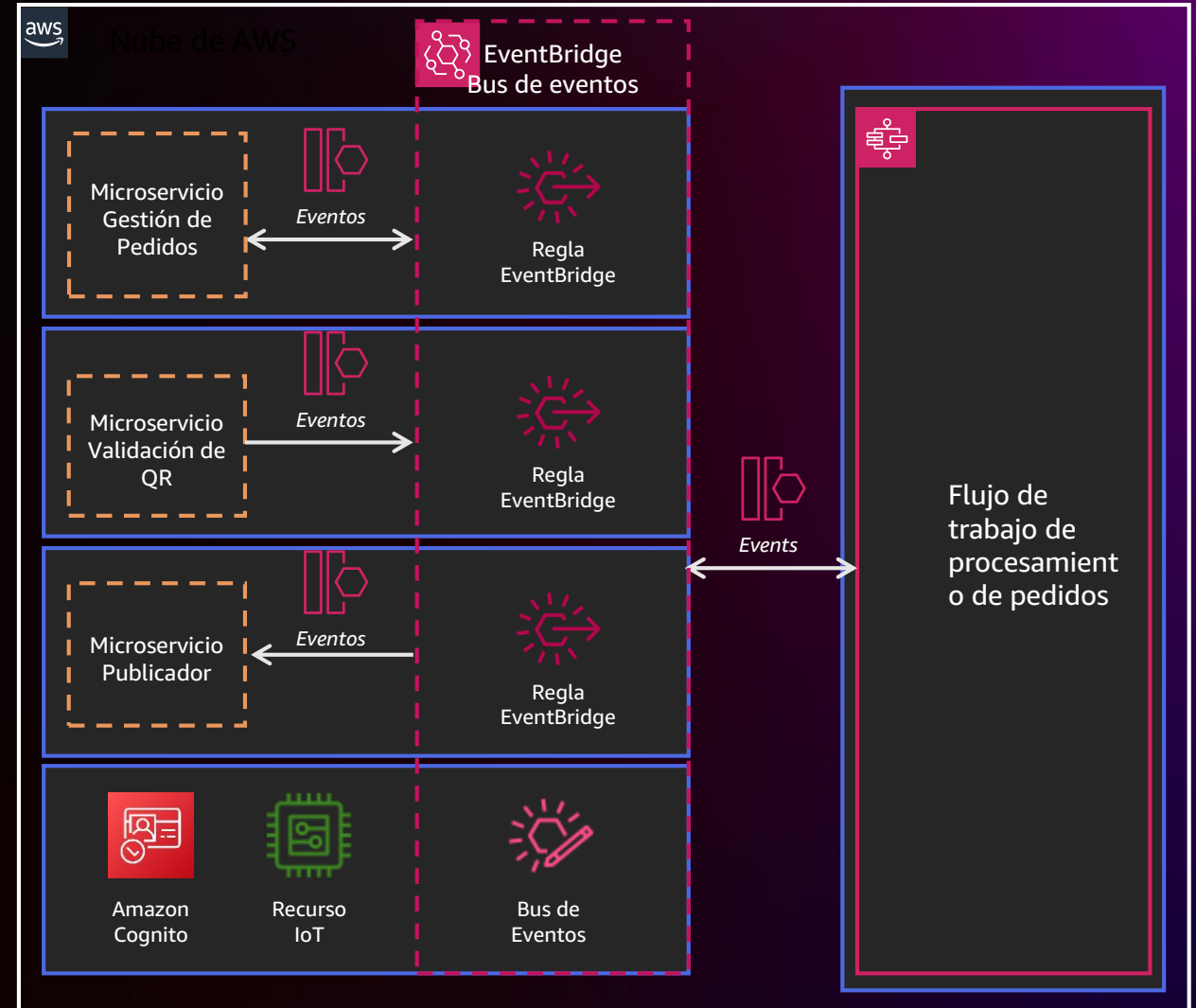
Comunicación entre microservicios

- El flujo de eventos impulsa la aplicación
- Los eventos coreografían los microservicios, mientras Step Functions orquesta las transacciones
- Reemplazar APIs privadas con eventos cuando sea posible
- Agregar nuevos microservicios sin cambiar el código existente
- Los microservicios emiten eventos independientemente de los consumidores



Rompiendo los despliegues monolíticos

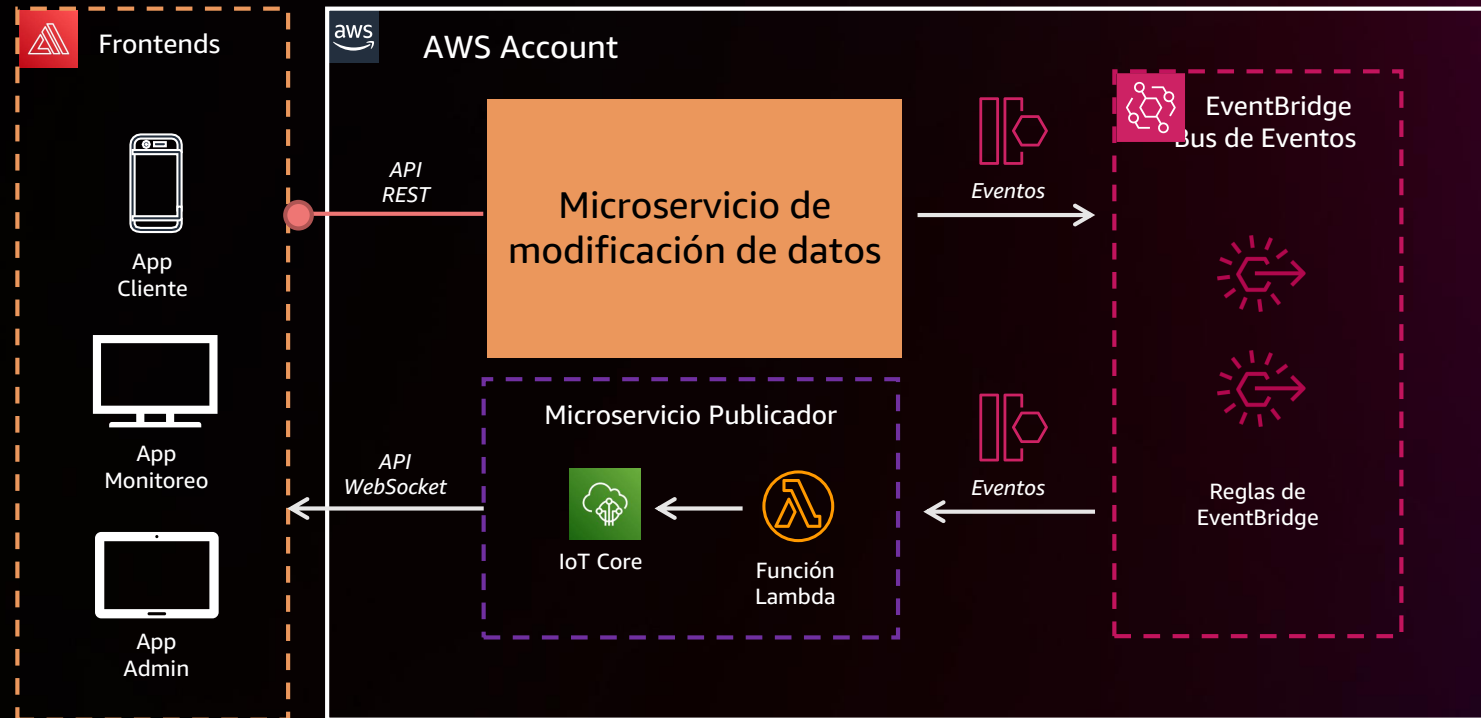
- Originalmente una plantilla AWS SAM grande
- Usar una plantilla principal para recursos comunes
- Cada microservicio es una plantilla AWS SAM que incluye una regla
- Cada flujo de trabajo es una plantilla de AWS SAM
- Implementación más rápida y segura



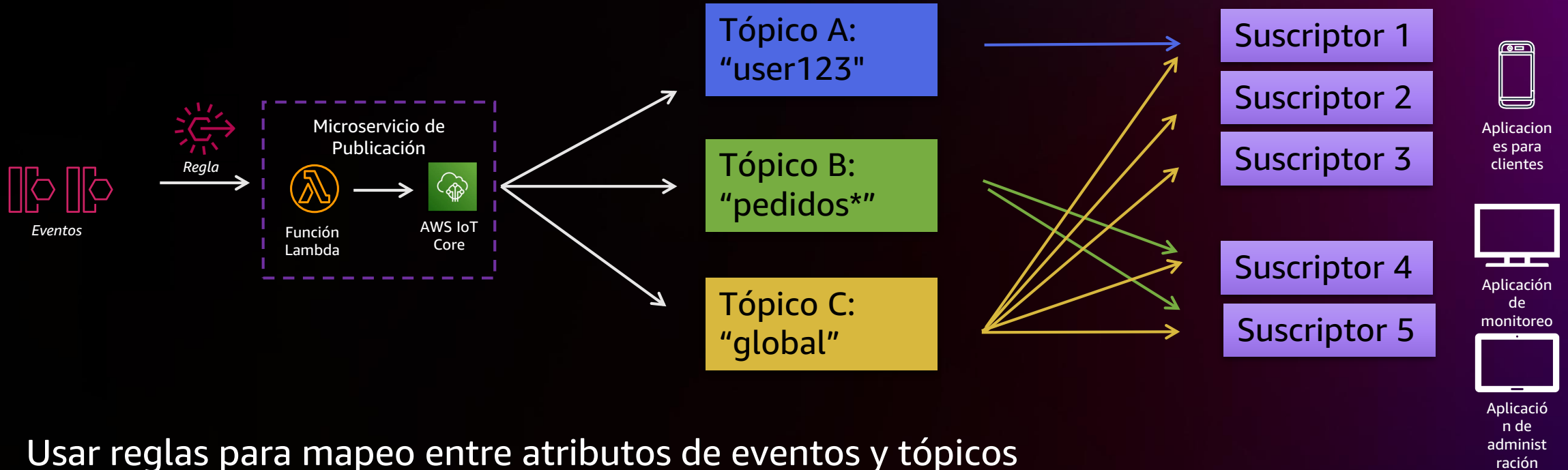
Patrones útiles



Separación de Responsabilidad de Comandos y Consultas (CQRS)... con WebSockets



CQRS con WebSockets



- Usar reglas para mapeo entre atributos de eventos y tópicos
- Usar reglas de filtrado de contenido para hacer coincidir varios valores (ejm. wildcards)

Orquestación y coreografía

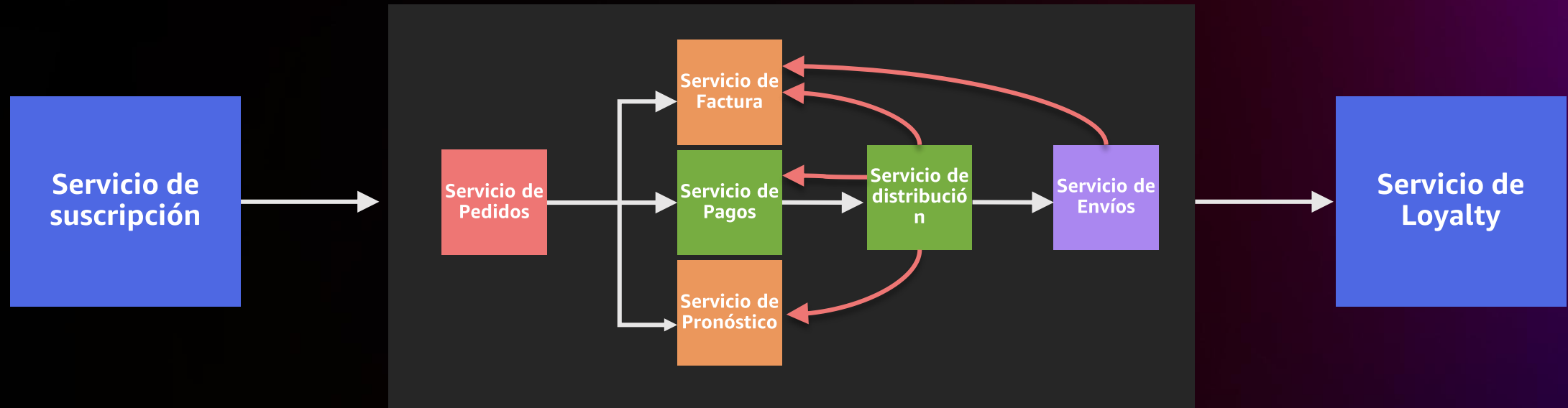
Orquestación con AWS Step Functions

- Un sistema controla el flujo entre componentes
- Supervisión de extremo a extremo más sencilla
- Timeout, reintentos, manejo de errores robusto
- Lógica de negocio centralizada

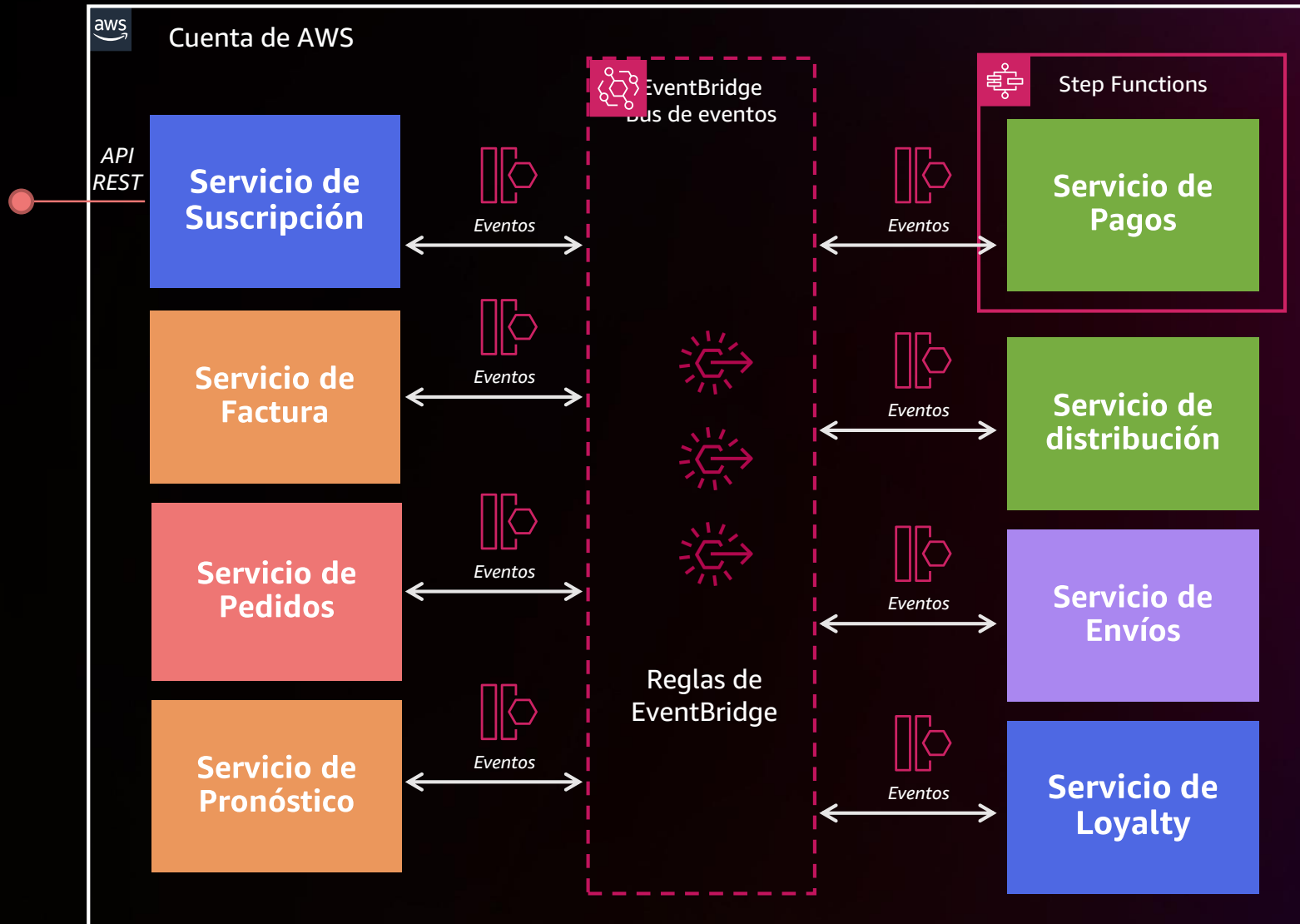
Coreografía con Amazon EventBridge

- Paso de mensajes entre el contexto delimitado de los microservicio servicios
- El flujo es una propiedad emergente de los eventos que se envían
- Más fácil de extender, modificar y construir sobre los mensajes que se pasan

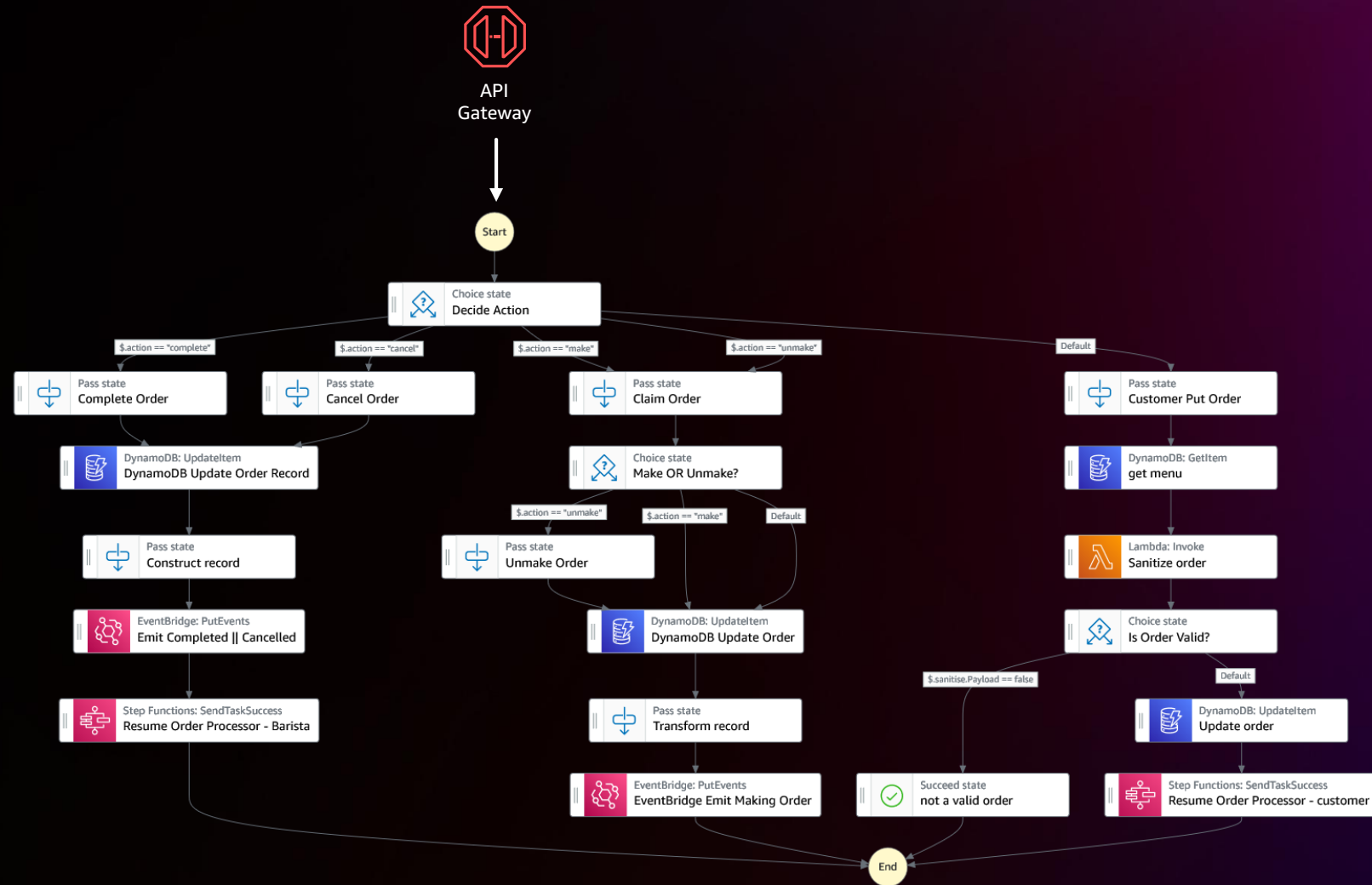
Pasando desde arquitecturas impulsadas por API...



... hacia arquitecturas impulsadas por eventos



Flujo de trabajo CRUD con Step Functions



Flujo de trabajo CRUD con Step Functions

¿Cómo?

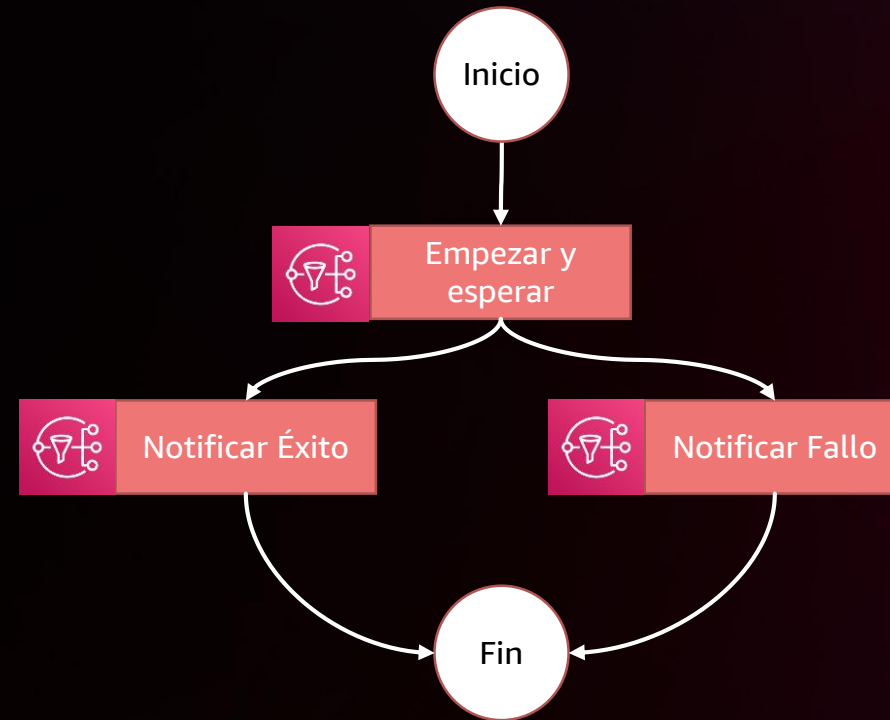
- Un endpoint de API Gateway inicia el flujo de trabajo
- Primera transición con rutas de acción CRUD
- Utiliza integraciones de servicios siempre que sea posible para reducir el cómputo
- Utiliza Lambda para lógicas personalizada
- También puede usar flujos de trabajo Express si el valor de retorno es necesario

Beneficios

- Complejidad reducida (archivo ASL único)
- Supervisión y depuración más fáciles
- Más fácil de implementar reintentos y manejo de errores
- Puede ser más rentable
- Puede reducir la latencia
- Permite el versionamiento de flujos de trabajo

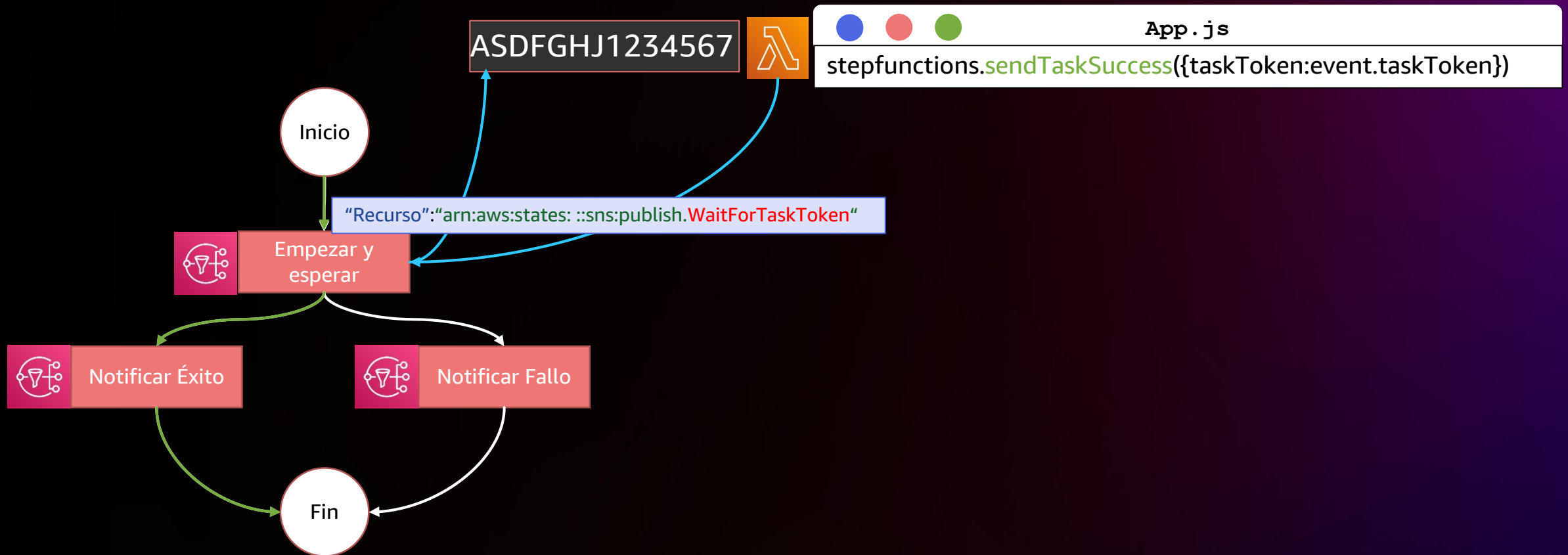
Uso de flujos de trabajo para gestionar la espera

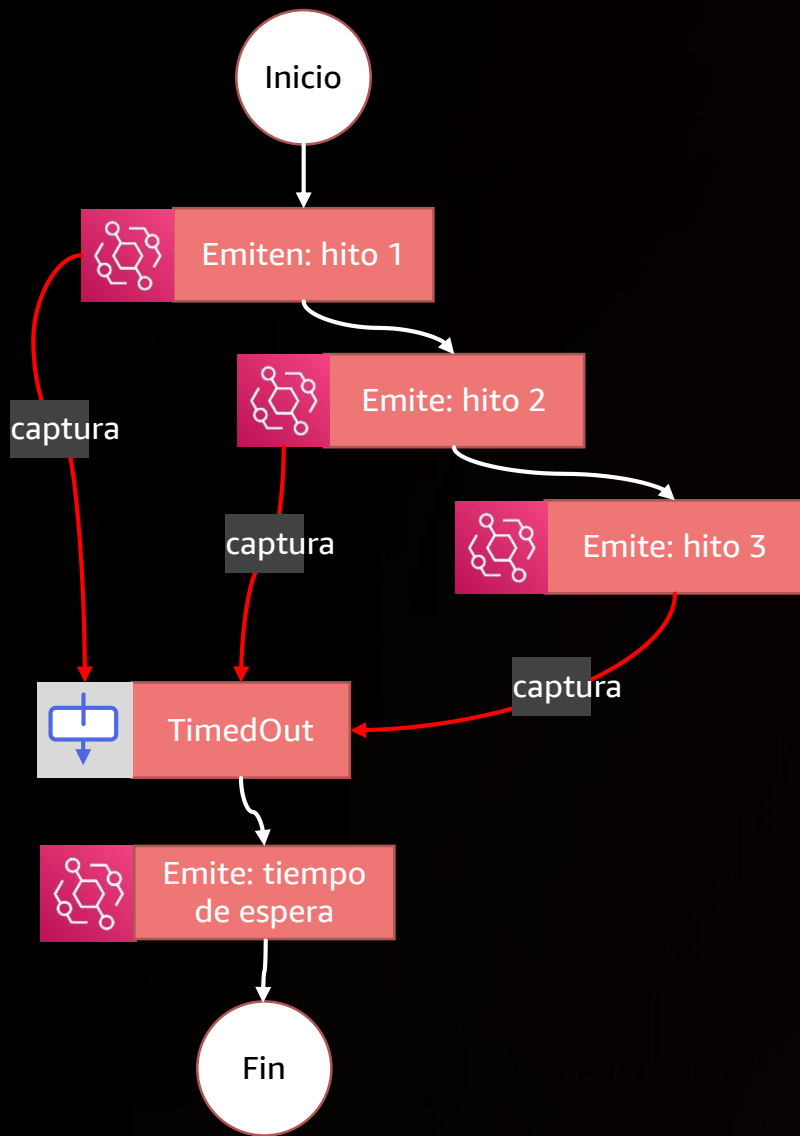
PAUSAR UN FLUJO DE TRABAJO HASTA POR 1 AÑO HASTA QUE SE DEVUELVA UN TOKEN DE TAREA



Uso de flujos de trabajo para gestionar la espera

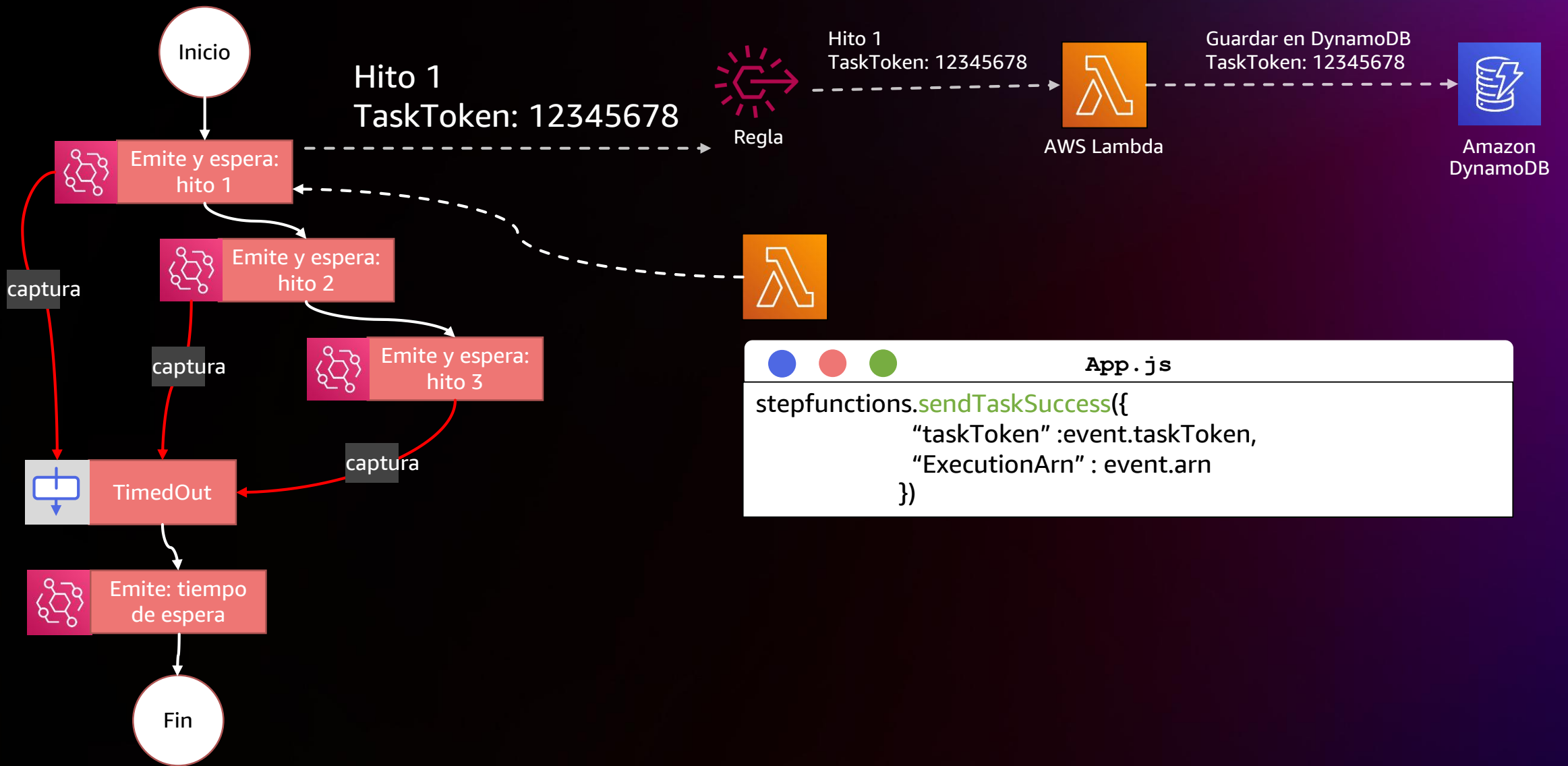
PAUSAR UN FLUJO DE TRABAJO HASTA POR 1 AÑO HASTA QUE SE DEVUELVA UN TOKEN DE TAREA

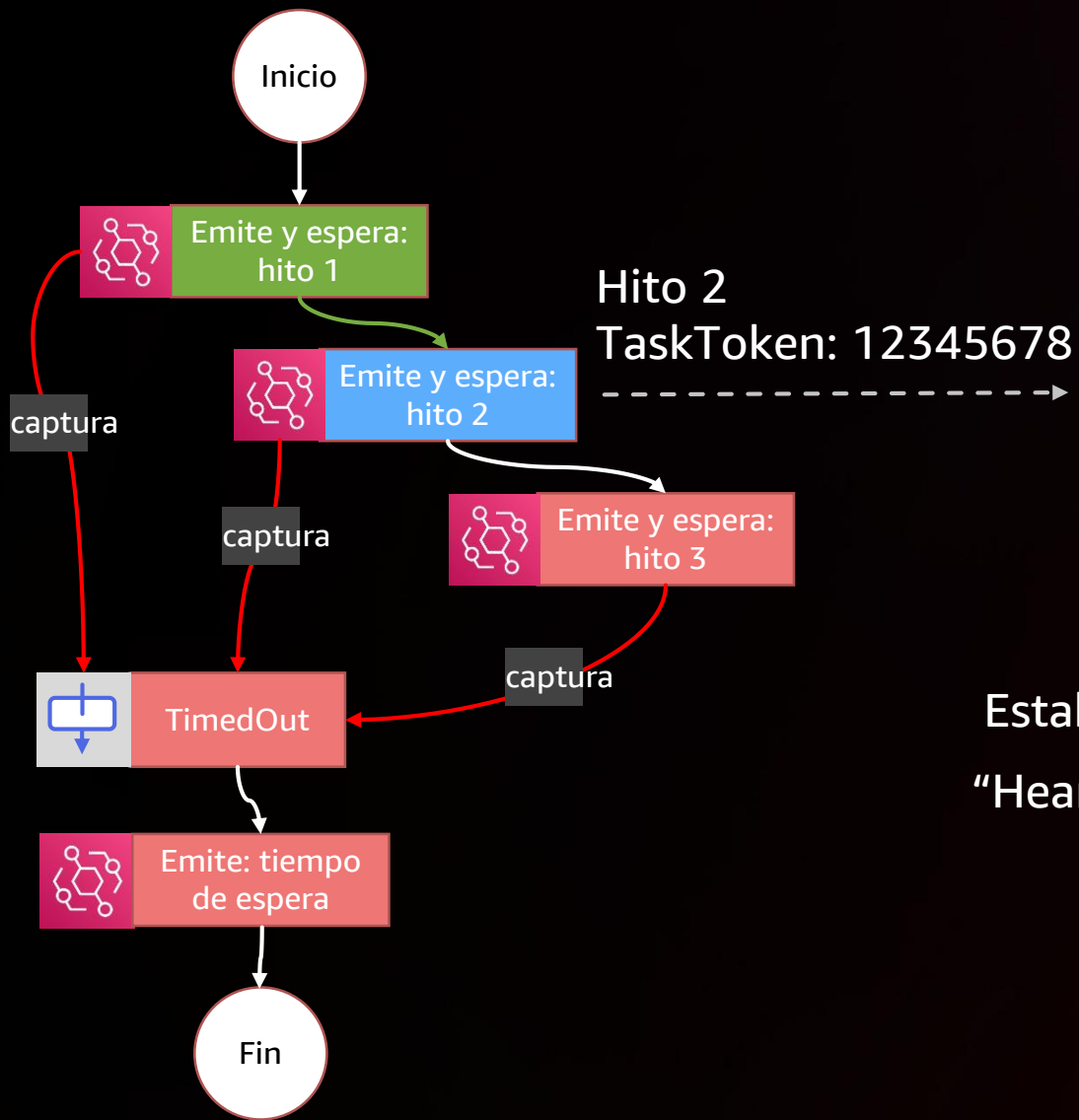




El patrón de “emitir y esperar”

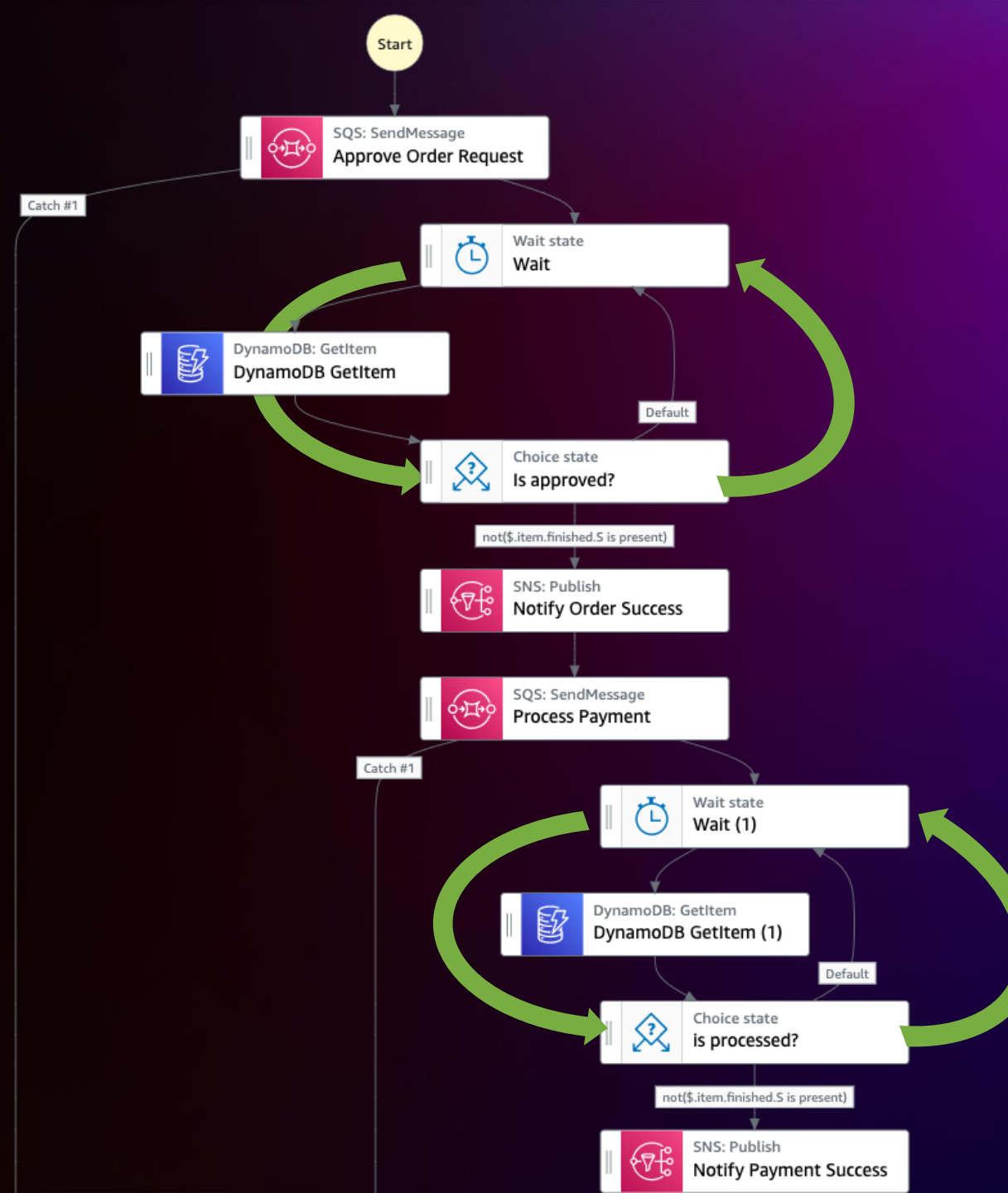
Emit eventos de hito que invocan microservicios externos desacoplados. Emite eventos de error si no responden a tiempo



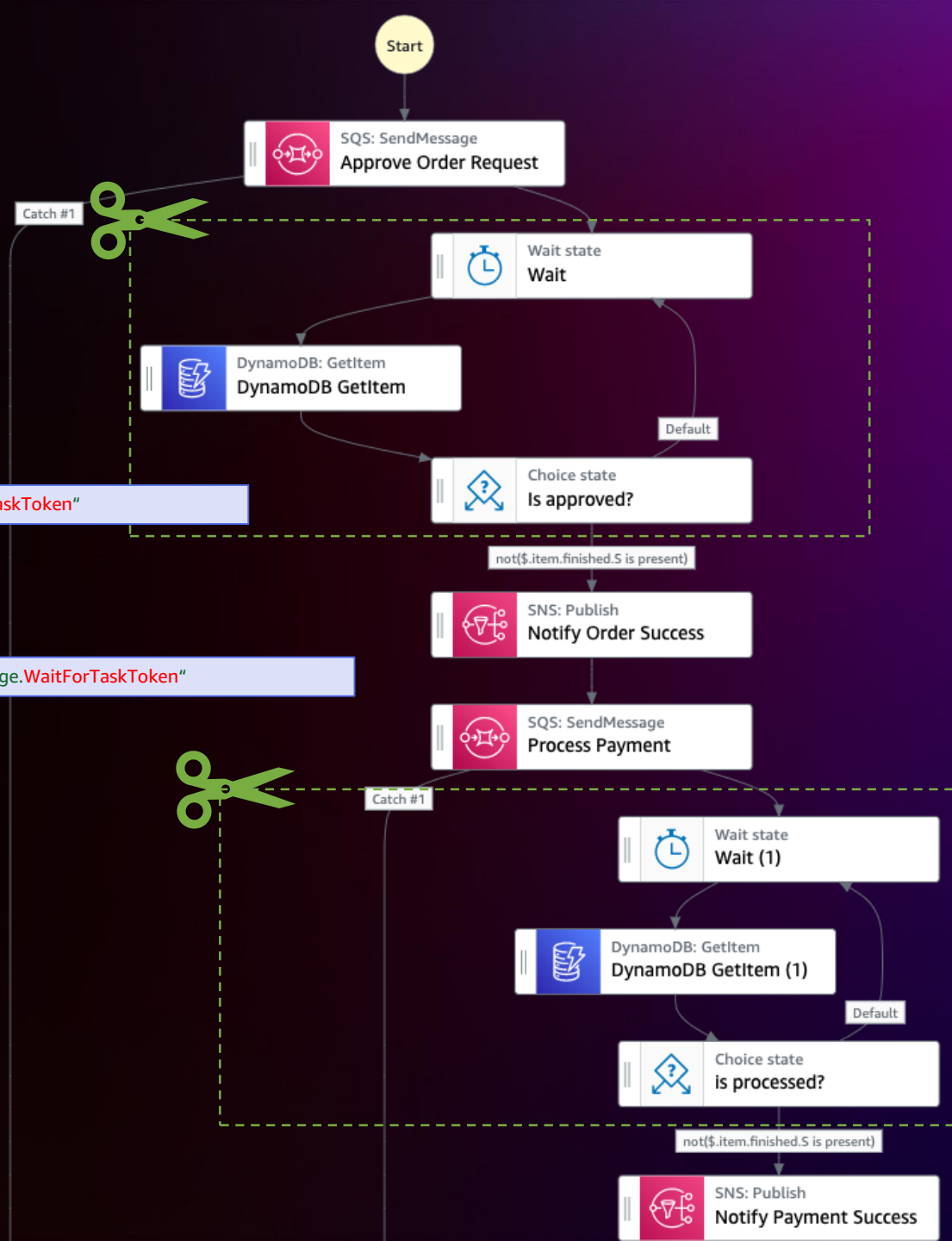
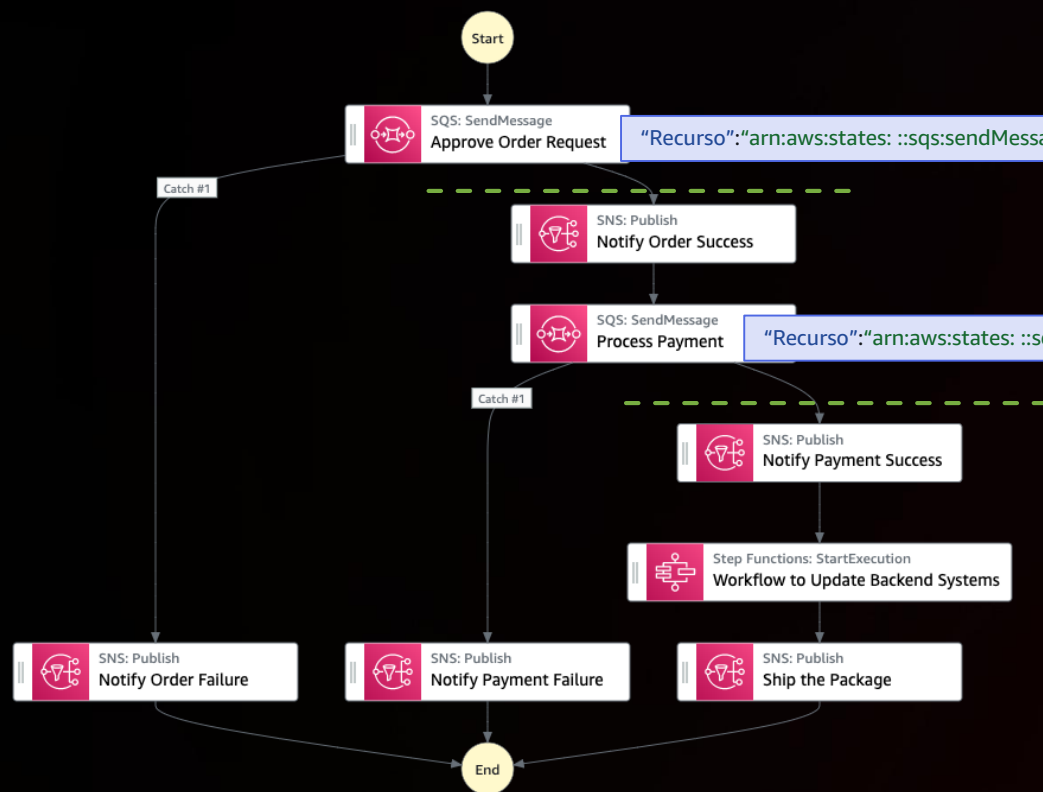


Establece un límite de tiempo para esperar el token de tarea
"HeartBeatSeconds": 20

Elimina polling con patrón de devolución de llamada



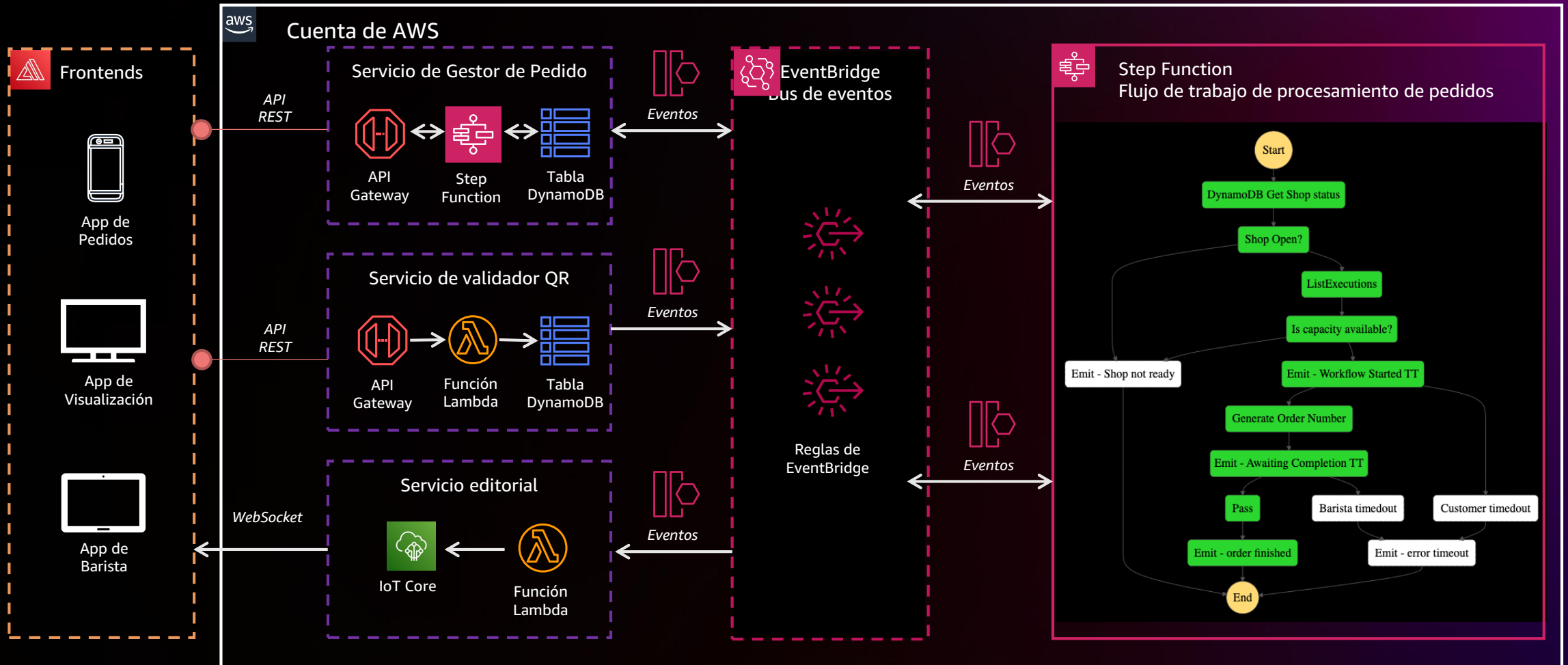
Elimina polling con patrón de devolución de llamada



Discusión



¿Qué tal si...?



¿Cuánto cuesta ejecutar esta carga de trabajo?

Podemos servir hasta 960 bebidas a 960 clientes todos los días

Servicio	Costo Diario	Con Capa Gratuita
AWS Amplify console	\$0.28	Free
Amazon API Gateway	\$0.01	Free
Amazon Cognito	\$0.00	Free
Amazon DynamoDB	\$0.01	Free
Amazon EventBridge	\$0.01	Free
AWS IoT Core	\$0.01	Free
AWS Lambda	\$0.01	Free
Amazon SNS (SMS messaging)	\$7.98	\$7.98
AWS Step Functions	\$0.29	Free



serverlesspresso



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Obtenga más información sobre la capa gratuita de AWS: <https://aws.amazon.com/free>

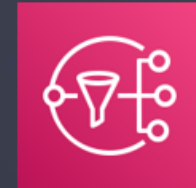
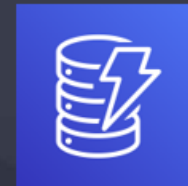
Resumen

- Diseño: comenzar con el flujo de trabajo; adjuntar microservicios; agregar frontend
- Microservicios: comunicarse con eventos; usar API si están orientados hacia afuera
- Tiempo real frontiendo: uso de AWS IoT Core para WebSockets sin servidor
- Utilice Step Functions para orquestar dentro del contexto del microservicio. Utilice EventBridge para comunicarse a través de esos límites.
- La combinación de orquestación con coreografía puede crear cargas de trabajo rentables, altamente extensibles y de bajo código

[New](#)[Blogs](#)[Videos](#)[Learn](#)[Events ▾](#)[Patterns](#)[About](#)

Welcome to Serverless Land

This site brings together all the latest blogs, videos, and training for AWS Serverless. Learn to use and build apps that scale automatically on low-cost, fully-managed serverless architecture.

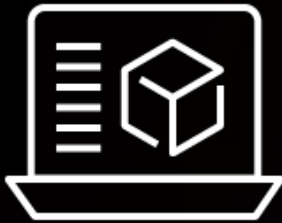
[Learn More](#)

Para obtener más recursos de aprendizaje sin servidor, visite:

<https://serverlessland.com>

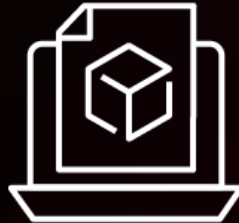
Continúe con su aprendizaje de AWS Serverless

Aprende a tu
propio ritmo



Amplíe su conocimiento en Serverless
AWS Skill Builder

Aumenta tu
conocimiento



Utilice nuestras **guías de Ramp-up**
para construir su conocimiento en
Serverless

Gane insignias en
AWS Serverless



Demuestre su
conocimiento al lograr
insignias digitales



<https://s12d.com/serverless-learning>



¡Gracias!

Andrés Victoria
andvic@amazon.com



Encuesta



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.