



Flujos de trabajo y patrones de IA de agentes en AWS

AWS Orientación prescriptiva



AWS Orientación prescriptiva: Flujos de trabajo y patrones de IA de agentes en AWS

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Las marcas comerciales y la imagen comercial de Amazon no se pueden utilizar en relación con ningún producto o servicio que no sea de Amazon, de ninguna manera que pueda causar confusión entre los clientes y que menosprecie o desacredite a Amazon. Todas las demás marcas registradas que no son propiedad de Amazon son propiedad de sus respectivos propietarios, que pueden o no estar afiliados, conectados o patrocinados por Amazon.

Table of Contents

Introducción	1
Destinatarios previstos	1
Objetivos	1
Acerca de esta serie de contenido	2
Patrones de agentes	3
Agentes de razonamiento básicos	4
Arquitectura	4
Description (Descripción)	5
Capacidades	6
Limitaciones	6
Casos de uso comunes	6
Guía para la implementación	7
Resumen	7
Arquitectura	7
Description (Descripción)	8
Capacidades	9
Casos de uso comunes	9
Guía para la implementación	9
Resumen	10
Tool-based agentes para llamar a funciones	10
Arquitectura	10
Description (Descripción)	11
Capacidades	12
Casos de uso comunes	12
Guía para la implementación	12
Resumen	13
Tool-based agentes para servidores	13
Arquitectura	13
Description (Descripción)	14
Capacidades	15
Casos de uso comunes	15
Guía para la implementación	15
Resumen	16
Computer-use agentes	16

Arquitectura	16
Description (Descripción)	17
Capacidades	18
Casos de uso comunes	18
Guía para la implementación	19
Resumen	19
Agentes de codificación	19
Arquitectura	20
Description (Descripción)	20
Capacidades	21
Casos de uso comunes	21
Guía para la implementación	22
Resumen	22
Agentes de voz y voz	22
Arquitectura	22
Description (Descripción)	23
Capacidades	24
Casos de uso comunes	24
Guía para la implementación	25
Resumen	25
Agentes de orquestación del flujo de trabajo	25
Arquitectura	26
Description (Descripción)	26
Capacidades	27
Casos de uso comunes	27
Guía para la implementación	27
Resumen	28
Memory-augmented agentes	28
Arquitectura	28
Description (Descripción)	29
Capacidades	30
Casos de uso comunes	30
Implementación de agentes con memoria aumentada	30
Implementación de mensajes con inyección de memoria	31
Resumen	32
Agentes de simulación y de banco de pruebas	32

Arquitectura	32
Description (Descripción)	33
Capacidades	34
Casos de uso comunes	34
Guía para la implementación	35
Resumen	36
Agentes de observación y monitoreo	36
Arquitectura	37
Description (Descripción)	37
Capacidades	38
Casos de uso comunes	38
Guía para la implementación	38
Resumen	39
Multi-agent colaboración	39
Description (Descripción)	41
Capacidades	42
Casos de uso comunes	42
Guía para la implementación	42
Resumen	43
Conclusión	43
Conclusiones	44
Flujos de trabajo LLM	45
Descripción general de la cognición LLM-augmented	45
Flujo de trabajo para un encadenamiento rápido	46
Description (Descripción)	47
Capacidades	48
Casos de uso comunes	48
Flujo de trabajo para enrutamiento	48
Capacidades	50
Casos de uso comunes	50
Flujo de trabajo para la paralelización	50
Capacidades	52
Casos de uso comunes	52
Flujo de trabajo para la orquestación	52
Capacidades	54
Casos de uso comunes	54

Flujo de trabajo para evaluadores y ciclos de reflexión y refinamiento	54
Casos de uso comunes	56
Capacidades	56
Conclusión	56
Patrones de flujo de trabajo de los agentes	58
Desde sistemas basados en eventos hasta sistemas con cognición aumentada	58
Event-driven arquitectura	59
Cognition-augmented flujos de trabajo	60
Perspectivas fundamentales	61
Patrones de saga que se encadenan rápidamente	61
Coreografía de la saga	62
Patrón de encadenamiento rápido	63
Coreografía de agentes	63
Conclusiones	65
Enrutamiento de patrones de despacho dinámicos	65
Envío dinámico	66
LLM-based enrutamiento	67
Agente (router)	68
Conclusiones	69
Patrones de paralelización y dispersión	69
Scatter-gather	70
LLM-based paralelización (cognición dispersa-recolectora)	72
Paralelización de agentes	72
Conclusiones	73
Patrones de orquestación de Saga	73
Orquestación de eventos	74
Role-based sistema de agentes (orquestador)	75
Supervisor	75
Conclusiones	77
El evaluador refleja y refina los patrones de bucles	77
Bucle de control de retroalimentación	78
Bucle de control de retroalimentación (evaluador)	80
Evaluador	80
Conclusiones	81
Diseñar los flujos de trabajo de los agentes en AWS	81
Conclusión	82

Historial de documentos 84

..... lxxxv

Flujos de trabajo y patrones de IA de agentes en AWS

Aaron Sempff y Andrew Hooker, Amazon Web Services

Julio de 2025 (historial [del documento](#))

Las organizaciones están adoptando grandes modelos de lenguaje (LLMs) y agentes de software para resolver problemas dinámicos y multidominio mediante una nueva disciplina arquitectónica llamada patrones agentic. Los patrones de los agentes son modelos fundamentales y estructuras modulares que se utilizan para diseñar y organizar agentes de IA orientados a objetivos en muchos contextos.

Destinatarios previstos

Esta guía está dirigida a arquitectos, desarrolladores y líderes de productos que desean crear aplicaciones inteligentes que vayan más allá de la lógica estática, la lógica simbólica y la automatización determinista.

Objetivos

Esta guía proporciona un marco de diseño y un enfoque de implementación para los sistemas de agentes de IA que funcionan de forma autónoma sin dejar de ser controlables y alineados con sus objetivos. Conecta los patrones arquitectónicos basados en eventos con diversas alternativas de agentes y demuestra cómo crear sistemas de agentes aptos para producción utilizando arquitecturas nativas de la nube. En esta guía se tratan los siguientes temas:

- **Patrones de agentes:** los patrones de agentes son plantillas de diseño reutilizables que describen la estructura y el comportamiento de los agentes individuales. Esto incluye agentes de razonamiento, agentes de recuperación aumentada, agentes de codificación, interfaces de voz, orquestadores de flujos de trabajo y sistemas colaborativos de múltiples agentes. Cada patrón ilustra cómo los agentes perciben, razonan, actúan y aprenden, mapeados. Servicios de AWS
- **Flujos de trabajo de LLM:** los flujos de trabajo se centran en la forma en que los agentes utilizan el LLMs razonamiento. Exploran las estrategias de incitación y los mecanismos de planificación, y describen cómo se LLMs utilizan no solo para generar texto, sino también para impulsar comportamientos estructurados, interpretables y confiables dentro de un circuito de agentes.
- **Patrones de flujo de trabajo de los agentes:** los patrones de flujo de trabajo describen cómo interactúan varios agentes, herramientas y entornos para formar sistemas autónomos. Esto incluye

patrones para la orquestación de tareas, la delegación de subagentes, la coordinación basada en eventos, la observabilidad y el control. Estos aspectos promueven arquitecturas de IA escalables, componibles y auditables.

Acerca de esta serie de contenido

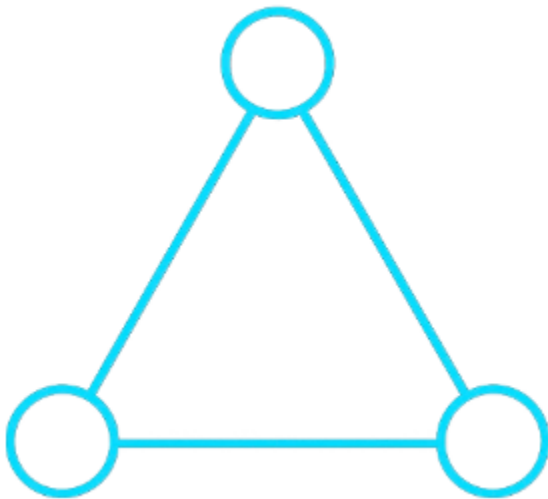
Esta guía forma parte de una serie sobre la IA de los agentes en AWS. Para obtener más información y ver las demás guías de esta serie, consulte [Agentic AI](#) en el sitio web de orientación prescriptiva. AWS

Patrones de agentes

Los patrones de agentes son componentes básicos reutilizables y componibles que se pueden adaptar a dominios, casos de uso y niveles de complejidad específicos. Sin embargo, los sistemas de agentes difieren de las aplicaciones tradicionales. En el centro de todos los diseños de agentes de IA se encuentra un modelo conceptual basado en los tres principios fundamentales siguientes:

- Asincrónico: los agentes operan en entornos poco acoplados y repletos de eventos
- Autonomía: los agentes actúan de forma independiente, sin control humano o externo
- Agencia: los agentes actúan con un propósito, en nombre de un usuario o sistema, para alcanzar objetivos específicos

El triángulo del siguiente diagrama representa los componentes básicos de un agente de software: percepción, razón y acción. Esto permite que un sistema de agencia observe, tome decisiones y actúe dentro de su entorno.



Por diseño, los patrones de agente proporcionan un lenguaje de diseño modular para crear sistemas de IA, lo que significa que son accesibles, operativos, ampliables y están listos para la producción. El diseño de estos sistemas requiere prestar especial atención a las siguientes tres dimensiones interrelacionadas, que se analizarán con más detalle más adelante en esta guía.

En esta sección

- [Agentes de razonamiento básicos](#)
- [Tool-based agentes para llamar a funciones](#)

- [Tool-based agentes para servidores](#)
- [Computer-use agentes](#)
- [Agentes de codificación](#)
- [Agentes de voz y voz](#)
- [Agentes de orquestación del flujo de trabajo](#)
- [Memory-augmented agentes](#)
- [Agentes de simulación y de banco de pruebas](#)
- [Agentes de observación y monitoreo](#)
- [Multi-agent colaboración](#)

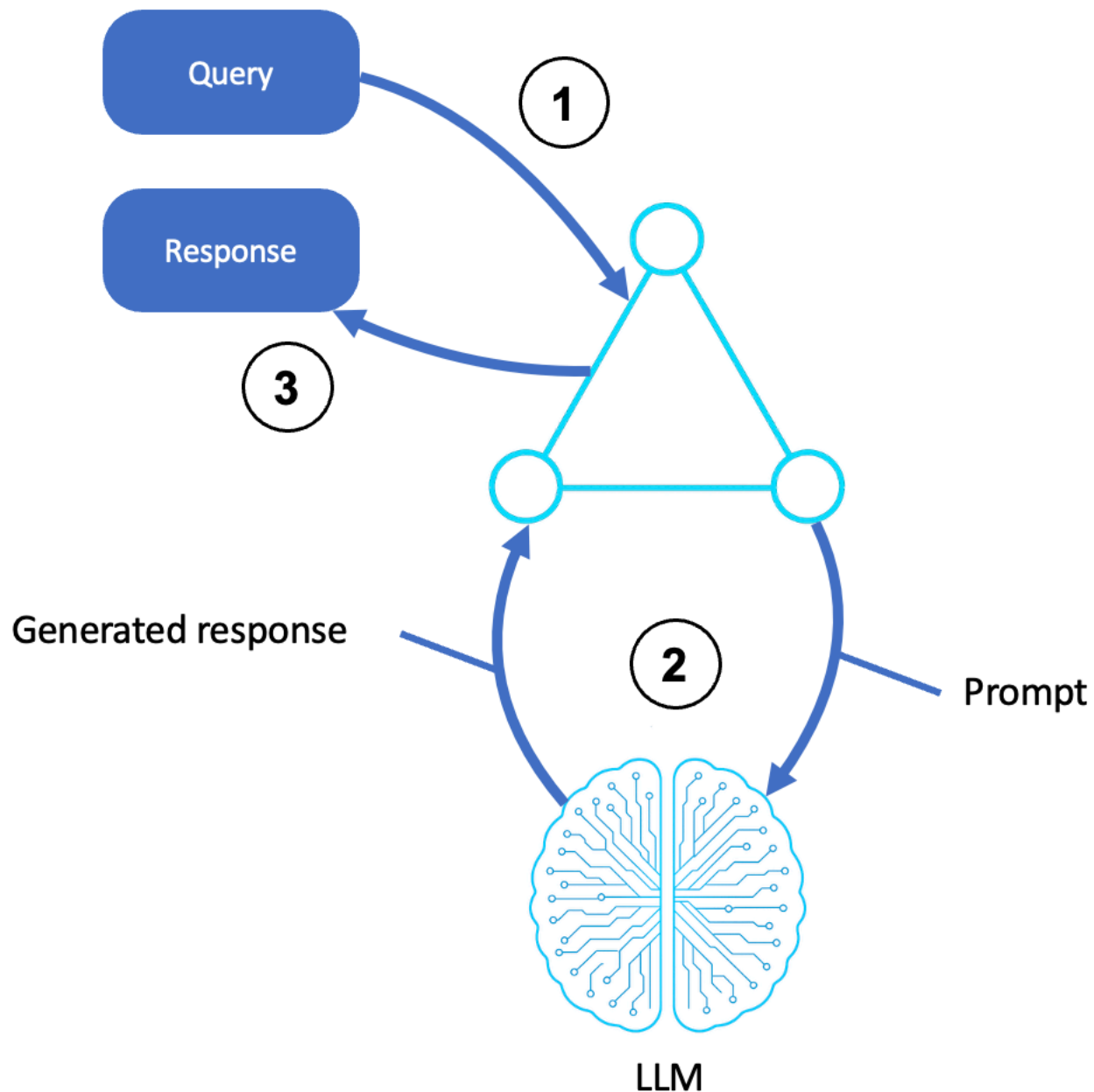
Agentes de razonamiento básicos

Un agente de razonamiento básico es la forma más simple de inteligencia artificial que realiza inferencias lógicas o toma de decisiones en respuesta a una consulta. Acepta la información de un usuario o un sistema y procesa las consultas y genera respuestas mediante indicaciones estructuradas.

Este patrón es útil para las tareas que requieren un razonamiento, una clasificación o un resumen en un solo paso en función de un contexto determinado. No usa memoria, herramientas ni administración de estados, lo que lo hace sin estado, liviano y altamente componible en grandes flujos de trabajo.

Arquitectura

El flujo de un agente de razonamiento básico se muestra en el siguiente diagrama:



Description (Descripción)

1. Recibe una entrada

- Un usuario, un sistema o un agente principal envía una consulta o instrucción.
- La entrada se transfiere al shell del agente o a la capa de orquestación.
- Este paso incluye el preprocesamiento, la creación rápida de plantillas y la identificación de los objetivos.

2. Invoca el LLM

- El agente transforma la consulta en una solicitud estructurada y la envía a un LLM (por ejemplo, a través de Amazon Bedrock).
- El LLM genera una respuesta basada en el mensaje utilizando el conocimiento y el contexto previamente entrenados.
- El resultado generado puede incluir pasos de razonamiento (cadena de pensamiento), respuestas finales u opciones clasificadas.

3. Devuelve una respuesta

- El resultado generado se retransmite a la interfaz del agente.
- Esto puede incluir el formateo, el posprocesamiento o una respuesta de la API.

Capacidades

- Admite lenguaje natural o entrada estructurada
- Utiliza una ingeniería rápida para guiar el comportamiento
- Sin estado y escalable
- Se puede integrar en la interfaz de usuario, la CLI, las API y las canalizaciones

Limitaciones

- Sin memoria ni conocimiento histórico
- Sin interacción con herramientas o fuentes de datos externas
- Limitado a lo que el LLM sabe en el momento de la inferencia

Casos de uso comunes

- Preguntas y respuestas conversacionales
- Explicaciones y resúmenes de las políticas
- Guía para la toma de decisiones
- Flujos de chatbots ligeros y automatizados
- Clasificación, etiquetado y puntuación

Guía para la implementación

Puede utilizar las siguientes herramientas y servicios para crear un agente de razonamiento básico:

- Amazon Bedrock para la invocación de LLM (Anthropic, AI21, Meta)
- Amazon API Gateway o AWS Lambda para exponerlo como un microservicio sin estado
- Plantillas de solicitudes almacenadas en el almacén de parámetros o como código AWS Secrets Manager

Resumen

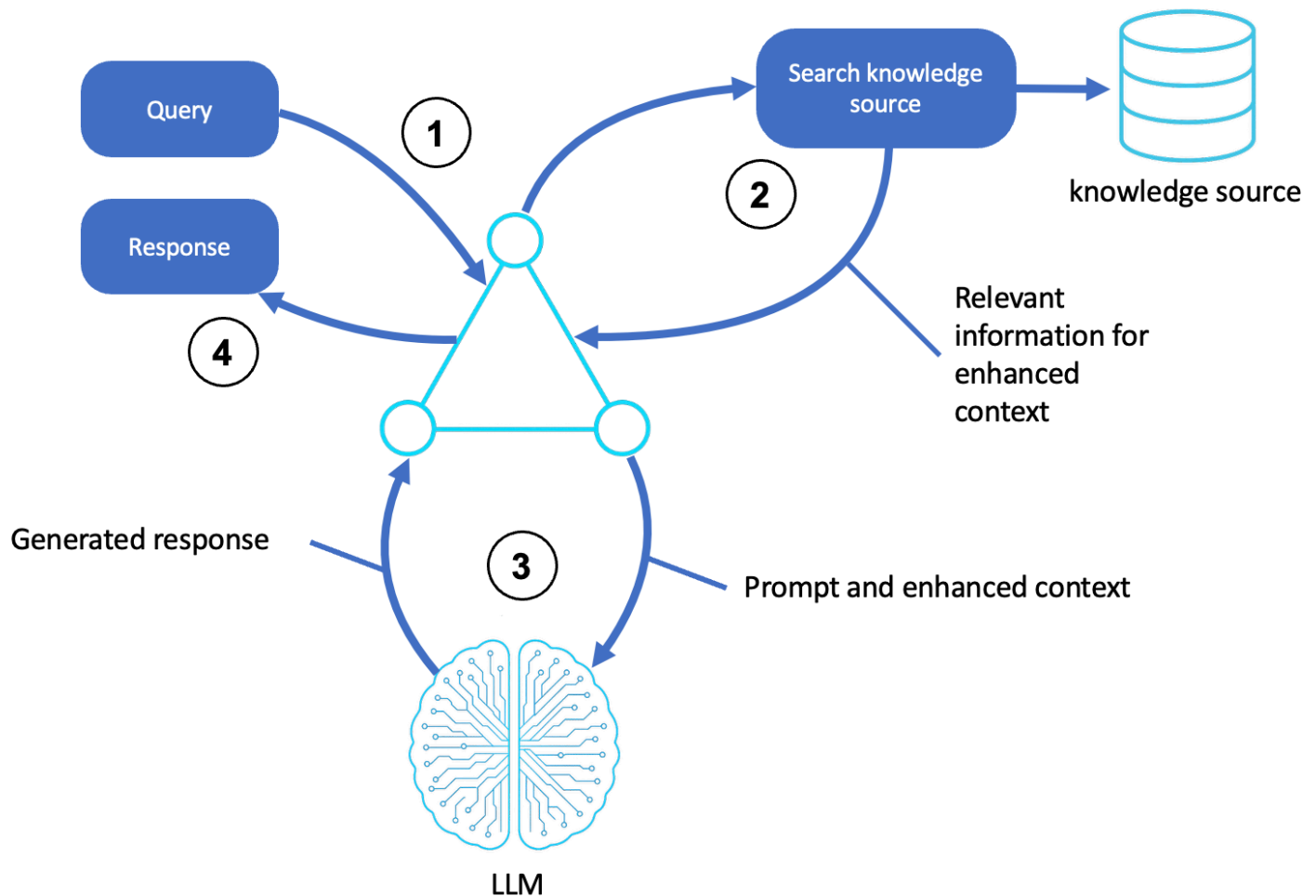
El agente de razonamiento básico es fundamental debido a su estructura simple. Tiene capacidades básicas que convierten los objetivos en vías de razonamiento que conducen a resultados inteligentes. Este patrón suele ser un punto de partida para patrones avanzados, como los agentes basados en herramientas y los agentes que utilizan la generación aumentada por recuperación (RAG). También es un componente fiable y modular de grandes flujos de trabajo.

Agente RAG

Retrieval-augmented La generación (RAG) es una técnica que combina la recuperación de información con la generación de texto para crear respuestas precisas y contextuales. El RAG permite a los agentes recuperar información externa relevante antes de contratar el LLM. Amplía la memoria efectiva y la precisión del razonamiento de un agente al basar sus decisiones en información actualizada, fáctica o específica de un dominio. A diferencia de los LLM apátridas, que se basan únicamente en pesas previamente entrenadas, RAG tiene una capa de búsqueda de conocimientos externa que mejora de forma dinámica las solicitudes en función del contexto.

Arquitectura

La lógica del patrón RAG se ilustra en el siguiente diagrama:



Description (Descripción)

1. Recibe una consulta

- Un usuario o un sistema anterior envía una consulta o un objetivo al agente.
- El intérprete de comandos acepta la solicitud y la formatea como una solicitud de razonamiento.

2. Busca una fuente externa

- El agente identifica los conceptos y la intención a partir de la consulta.
- Consulta una fuente de conocimiento, como un almacén vectorial, una base de datos o un índice de documentos, mediante la búsqueda semántica o la coincidencia de palabras clave.
- Los pasajes, documentos o entidades más relevantes se recuperan para usarlos en el siguiente paso.

3. Genera una respuesta contextual

- El agente amplía el mensaje con la información recuperada, lo que constituye una entrada mejorada por el contexto para el LLM.
- El LLM procesa cualquier entrada mediante el razonamiento generativo (por ejemplo, una cadena de pensamiento o una reflexión) para producir una respuesta precisa.

4. Devuelve el resultado final

- El agente prepara el resultado envolviéndolo en cualquier encabezado de comunicación o en el formato requerido y, a continuación, lo devuelve al usuario o al sistema de llamada.
- (Opcional) Los documentos recuperados y la salida de LLM se pueden registrar, calificar y almacenar en la memoria para futuras consultas.

Capacidades

- Fact-grounded generan resultados incluso en dominios de larga duración o específicos de la empresa
- Ampliación de la memoria sin ajustar el modelo
- Contexto dinámico basado en cada consulta y estado de usuario
- Totalmente compatible con bases de datos vectoriales, índices semánticos y filtrado de metadatos

Casos de uso comunes

- Asistentes de conocimiento empresarial
- Bots de cumplimiento normativo
- Copilotos de atención al cliente
- Search-enhanced chatbots
- Agentes de documentación para desarrolladores

Guía para la implementación

Utilice las siguientes herramientas y servicios para crear un agente que utilice RAG:

- Amazon Bedrock para la invocación de LLM
- Amazon Kendra o Amazon Aurora para documentación o una búsqueda estructurada de datos OpenSearch

- Amazon Simple Storage Service (Amazon S3) para almacenamiento de documentos
- AWS Lambda para organizar la búsqueda, la solicitud y la inferencia de LLM
- Knowledge-based integraciones con agentes (mediante complementos de memoria, recuperadores semánticos o Amazon Bedrock)

Resumen

El agente RAG conecta el razonamiento de modelos estáticos con la inteligencia dinámica del mundo real. Proporciona a los agentes la capacidad de buscar lo que no saben, sintetizar las respuestas a partir del conocimiento recuperado y producir respuestas auditables y de alta confianza.

Los patrones RAG son la base para crear agentes inteligentes que amplíen el acceso al conocimiento sin necesidad de volver a capacitarse. Suele ser un precursor de patrones de orquestación más complejos que implican el uso de herramientas, la planificación y la memoria a largo plazo.

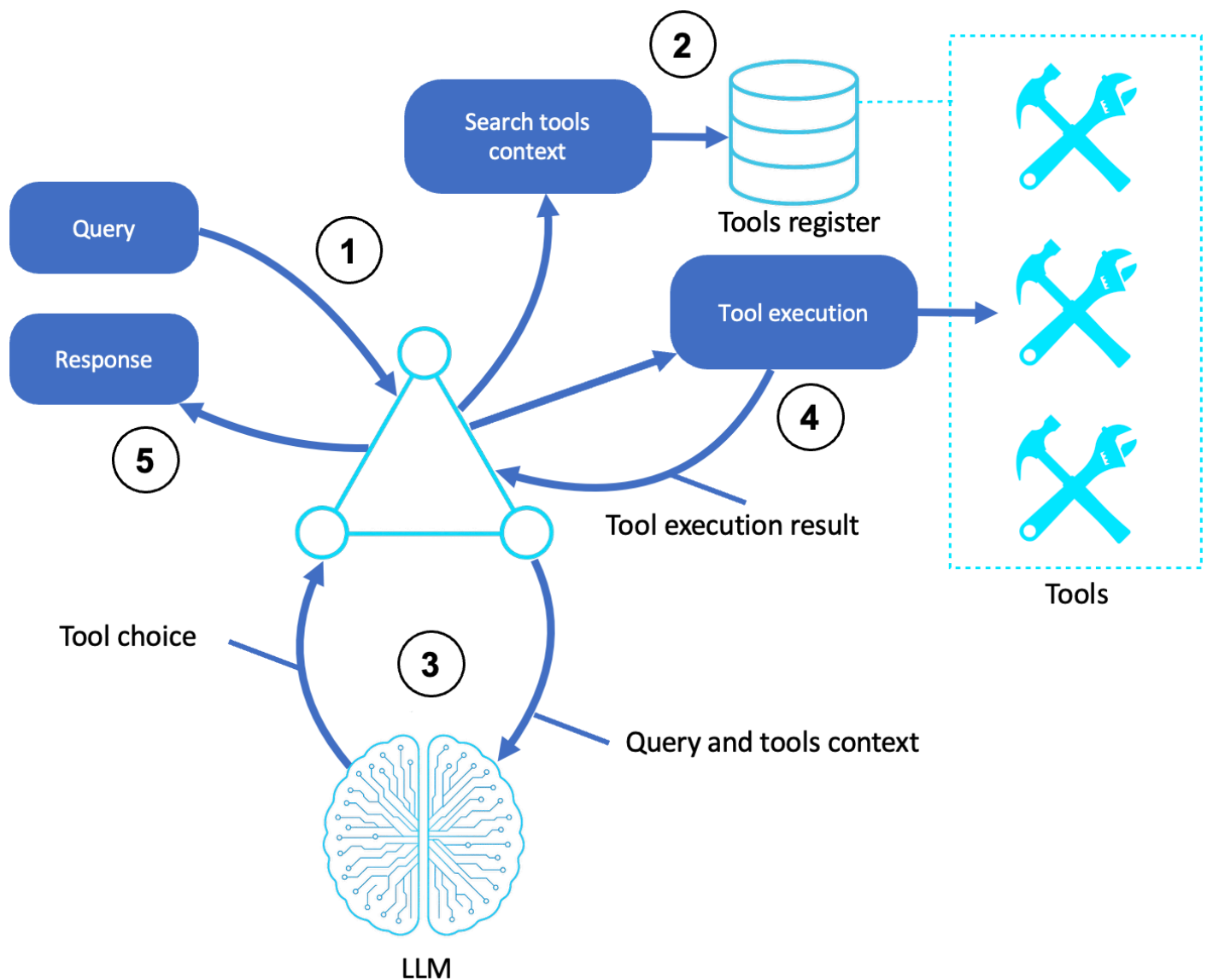
Tool-based agentes para llamar a funciones

Tool-based los agentes amplían las capacidades de los agentes de razonamiento al invocar funciones externas o API para completar tareas que van más allá del razonamiento basado únicamente en el lenguaje. Este patrón utiliza un LLM para decidir qué herramienta utilizar y, a continuación, genera argumentos de llamada e incorpora los resultados de la herramienta en su ciclo de razonamiento.

Este patrón permite a los agentes actuar en lugar de limitarse a dar respuestas. La interfaz de la herramienta representa cualquier capacidad que se pueda invocar, desde cálculos aritméticos y búsquedas en bases de datos hasta API externas y servicios en la nube.

Arquitectura

En el siguiente diagrama se muestra un agente basado en herramientas para llamar a funciones:



Description (Descripción)

1. Recibe una consulta

- El agente recibe una consulta o tarea en lenguaje natural del usuario o del sistema que realiza la llamada.

2. Busca herramientas

- El agente utiliza metadatos internos o un registro de herramientas para buscar las herramientas, los esquemas y las capacidades relevantes disponibles.

3. Selecciona e invoca las herramientas

- El LLM recibe los metadatos de la consulta y la herramienta (por ejemplo, los nombres de las funciones, los tipos de entrada y las descripciones) en su solicitud de datos.
 - Elige la herramienta más relevante, construye los argumentos de entrada y devuelve una llamada a una función estructurada.
4. Ejecuta la herramienta elegida
- El shell del agente o el ejecutor de herramientas ejecuta la función seleccionada y devuelve el resultado (por ejemplo, una salida de API, un valor de base de datos o un cálculo).
5. Devuelve una respuesta
- El LLM pasa los resultados al agente, ya sea directamente o como parte de una solicitud actualizada. A continuación, devuelve un resultado en lenguaje natural.

Capacidades

- Selección dinámica de herramientas basada en el contexto de la tarea
- Schema-based solicitud (OpenAPI, esquema AWS JSON, interfaz de funciones)
- Interpretación de los resultados y encadenamiento de los resultados en el razonamiento
- Operaciones apátridas o con registro de sesiones

Casos de uso comunes

- Asistentes virtuales con acceso a datos externos
- Calculadoras y estimadoras financieras
- API-based trabajadores del conocimiento
- LLM que invocan, SageMaker puntos de conexión de AWS Lambda Amazon y servicios SaaS

Guía para la implementación

Utilice lo siguiente para crear agentes basados en herramientas para llamar a funciones:

- Amazon Bedrock con soporte para llamadas a funciones (Anthropic Claude)
- AWS Lambda como un backend de ejecución de herramientas
- Amazon API Gateway o AWS Step Functions para la organización de herramientas

- Amazon DynamoDB o Amazon Relational Database Service (Amazon RDS) para metadatos de herramientas sensibles al contexto
- Amazon EventBridge Pipelines o AWS Step Functions que mapean estados para enrutar las salidas

Resumen

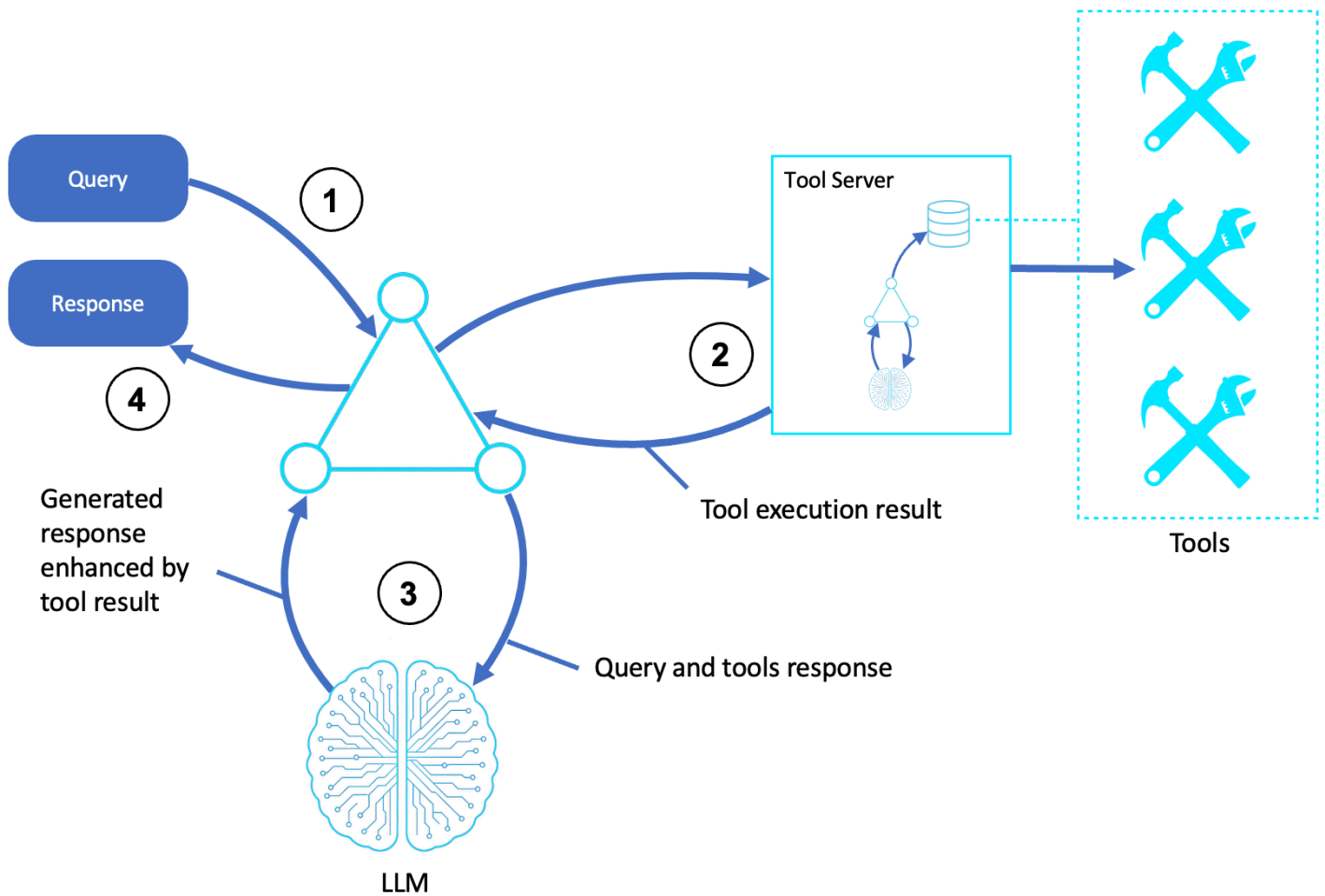
Tool-based los agentes que llaman a las funciones representan un cambio desde la comprensión del lenguaje hasta la ejecución de acciones. Estos agentes recurren a herramientas dinámicas y sensibles al contexto y, al mismo tiempo, mantienen el razonamiento LLM, transformando los asistentes pasivos en sistemas que completan tareas, acceden a los servicios e integran las operaciones comerciales. Este patrón es un componente importante de la IA de los agentes en los entornos empresariales, especialmente cuando se combina con esquemas declarativos, marcos de autorización y sistemas multiagente.

Tool-based agentes para servidores

Tool-based Los agentes para servidores mejoran las funciones de los agentes al delegar la ejecución de las herramientas a un servidor externo que tiene un entorno de ejecución dedicado a las herramientas, los scripts y los agentes compuestos. A diferencia de las llamadas a funciones en línea que el bucle de agentes selecciona e invoca, los agentes basados en servidor subcontratan la lógica y el proceso de ejecución a otros agentes o sistemas. Esto proporciona funciones avanzadas, como el encadenamiento de múltiples herramientas, la ejecución aislada y el razonamiento especializado. Los servidores de herramientas son ideales para acciones complejas, activas o que requieren muchos recursos, en las que las propias herramientas pueden implicar modelos de IA, reglas empresariales o entornos independientes.

Arquitectura

El siguiente es un patrón para los agentes basados en herramientas para servidores:



Description (Descripción)

1. Recibe una consulta

- Un usuario o un sistema envía una solicitud al shell del agente.
- El agente interpreta la consulta y se prepara para enviarla a un servidor de herramientas.

2. Ejecuta los procesos del servidor de herramientas

- El agente envía la tarea, junto con los parámetros estructurados, a un servidor de herramientas.
- El servidor de herramientas puede entonces:
 - Ejecute scripts o lógica en sistemas de cómputo dedicados (por ejemplo AWS Lambda, contenedores o Amazon SageMaker)
 - Utilice su propio subagente con el razonamiento LLM para seleccionar y ejecutar las herramientas
 - Gestione las dependencias, los reintentos o los flujos de ejecución en varios pasos

- Envíe los resultados al agente principal cuando se complete la tarea
3. Utiliza el razonamiento LLM con el resultado de la herramienta
 - El agente invoca un LLM y pasa la consulta original y el resultado del servidor de herramientas como parte de la solicitud.
 - El LLM sintetiza una respuesta que incorpora la información recién adquirida.
 4. Devuelve una respuesta
 - El agente devuelve una respuesta estructurada o en lenguaje natural al usuario o al sistema que realiza la llamada.
 - (Opcional) Los resultados se pueden almacenar en la memoria o en los registros de auditoría.

Capacidades

- Las herramientas se invocan fuera del ciclo de ejecución del agente principal
- La ejecución de la herramienta puede implicar llamadas LLM, cadenas lógicas o subagentes
- El agente actúa como controlador o despachador, no solo como un envoltorio de herramientas
- Permite la componibilidad, la escalabilidad y el aislamiento de la lógica

Casos de uso comunes

- Organización de cadenas de modelos (por ejemplo, mediante la combinación de LLM, visión y código)
- AI-driven canalizaciones de automatización
- DevOps agentes asistentes con ejecutores de guiones
- Agentes complejos de computación, simulación u optimización financiera
- Herramientas multimodales (por ejemplo, mediante la combinación de audio, documentación y acción)

Guía para la implementación

Puede crear este patrón utilizando lo siguiente: Servicios de AWS

- Amazon Bedrock (inferencia de agente, anfitrión y LLM)

- AWS Lambda, Amazon ECS o Amazon SageMaker Endpoints como entorno de ejecución del servidor de herramientas AWS Fargate
- Amazon API Gateway o AWS App Runner para exponer las API del servidor de herramientas
- Amazon EventBridge para la mensajería disociada del agente a la herramienta
- AWS Step Functions o AWS AppFabric para componer una lógica multiagente en el servidor de herramientas

Resumen

Tool-based los agentes que utilizan servidores son altamente modulares y escalables. Disocian la lógica de decisión de la ejecución, lo que permite que el agente principal sea ligero y, al mismo tiempo, delegue acciones complejas o delicadas a otros sistemas. Esto es importante para la IA de los agentes de nivel empresarial, especialmente en entornos que requieren gobernabilidad, observabilidad, aislamiento, composición dinámica o cualquier combinación de ambos.

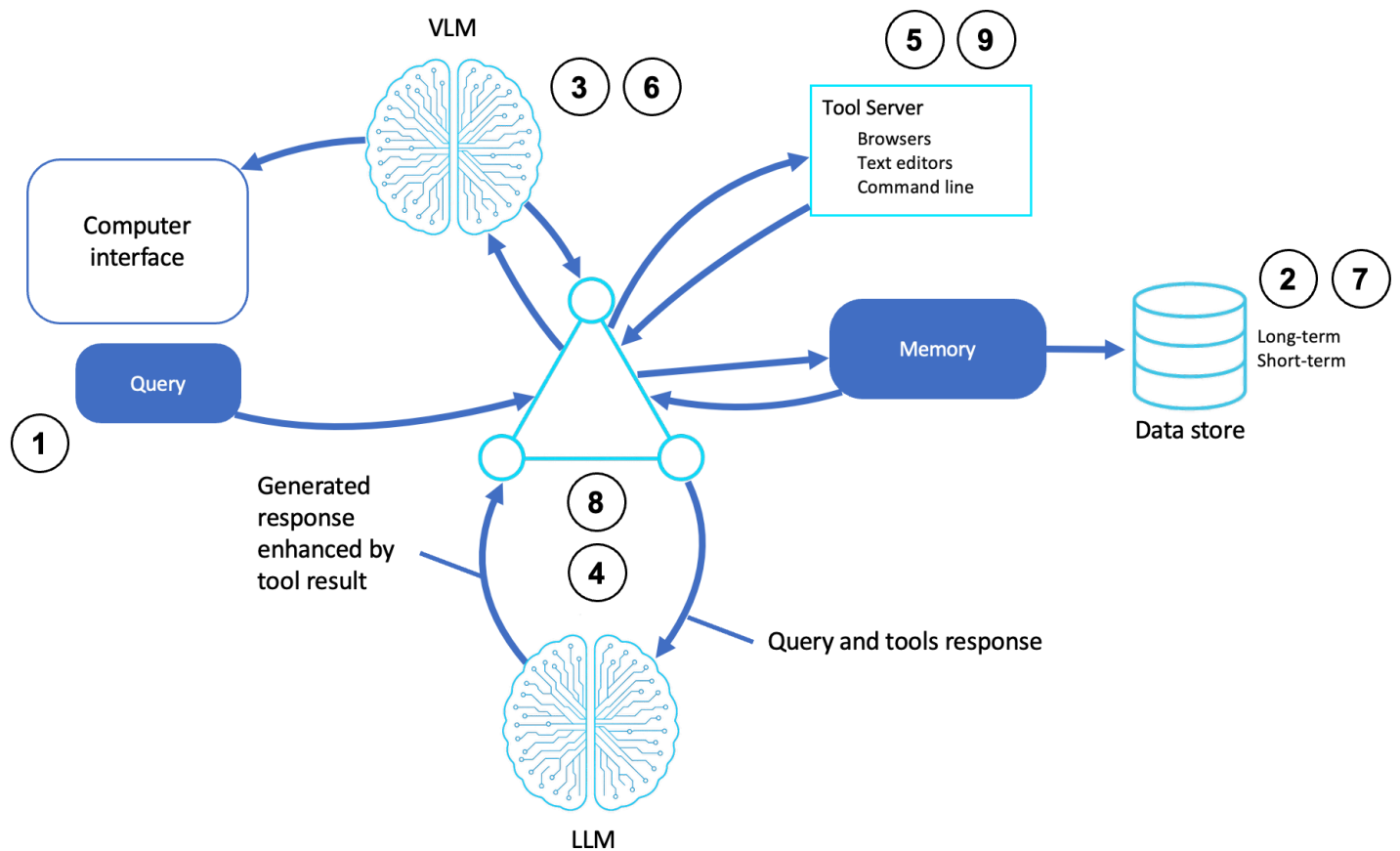
Computer-use agentes

Computer-use los agentes pueden simular o controlar entornos digitales como navegadores, terminales, sistemas de archivos y aplicaciones. Estos agentes interpretan la intención del usuario, interactúan con las interfaces visuales y de texto y realizan acciones orientadas a objetivos mediante la combinación del razonamiento LLM, los modelos de lenguaje visual (VLM) y los servidores de herramientas que ejecutan comandos o simulan eventos de entrada.

Este patrón es importante para las automatizaciones prácticas de IA, en las que el agente funciona no solo como asistente, sino también como un proxy que realiza acciones como lo haría un humano, a menudo utilizando las mismas herramientas y entornos.

Arquitectura

En el siguiente diagrama se muestra un patrón de agentes que utilizan ordenadores:



Description (Descripción)

1. Recibe una consulta

- Una tarea o solicitud se proporciona a través de una interfaz de usuario, API o lenguaje natural.

2. Accede a la memoria

- El agente recupera la memoria a corto y largo plazo para recordar órdenes, objetivos y estados del sistema anteriores.

3. Analiza el contexto visual

- Un VLM observa la pantalla de la computadora, el estado del sistema o los elementos de la interfaz de usuario para comprender un contexto determinado e identificar los elementos procesables.

4. Razones a través de un LLM

- El LLM combina la consulta, el estado de la memoria, la herramienta y la respuesta del servidor para determinar la siguiente acción.

5. Interactúa con el servidor de herramientas

- El agente invoca herramientas que están alojadas en un servidor, entre las que se incluyen las siguientes:
 - Navegadores (por ejemplo, Headless Chrome) y entornos de shell
 - Editores de texto y código
 - Interfaces de script personalizadas
6. Actualiza las entradas visuales
- Si la interfaz de usuario del sistema cambia o es necesario realizar más observaciones, el VLM puede volver a analizar el estado de la pantalla o los búferes de texto.
7. Actualiza la memoria
- Los nuevos conocimientos, los estados del sistema o los comentarios de los usuarios se escriben en la memoria a corto y largo plazo.
8. Formula las decisiones y explicaciones finales
- El LLM sintetiza los resultados o recomienda acciones en función de la consulta y el resultado de la herramienta.
9. Devuelve una respuesta
- El agente devuelve los resultados a la interfaz (por ejemplo, una tarea completada, una confirmación o un contenido generado).

Capacidades

- Razonamiento multimodal con entradas visuales y textuales
- Control de las aplicaciones mediante entradas simuladas o API-driven
- Gestión de memoria para un estado persistente
- Autonomía en la ejecución secuencial (flujos de varios pasos)

Casos de uso comunes

- Desarrolladores de IA que escriben y ejecutan código en IDE
- Computer-use agentes para flujos de trabajo digitales repetitivos
- Usuarios simulados para pruebas de software y control de calidad
- Agentes de accesibilidad para navegar por las interfaces de usuario mediante instrucciones de voz o de alto nivel

- Automatización robótica inteligente de procesos (RPA) que se mejora con el razonamiento

Guía para la implementación

- Puede crear este patrón con lo siguiente: Servicios de AWS
- Amazon Bedrock para LLM-based planificar y razonar
- Amazon Elastic Compute Cloud (Amazon EC2) AWS Lambda o SageMaker Amazon notebooks para ejecutar servidores de herramientas con entornos de interfaz de usuario simulados
- Amazon Simple Storage Service (Amazon S3) o Amazon DynamoDB para la persistencia de la memoria
- Amazon Rekognition (o modelos personalizados) para el análisis de imágenes de interfaz de usuario en escenarios híbridos
- Amazon CloudWatch Logs o AWS X-Ray para registros de observabilidad y auditoría

Resumen

Computer-use los agentes actúan como operadores digitales autónomos, cerrando la brecha entre las interacciones y las acciones entre humanos y computadoras. AI-driven AI incorporar memoria, orquestación de herramientas y VLM, estos agentes pueden interactuar de forma adaptativa con sistemas diseñados para humanos, ejecutar acciones, actualizar archivos, navegar por los menús y generar respuestas.

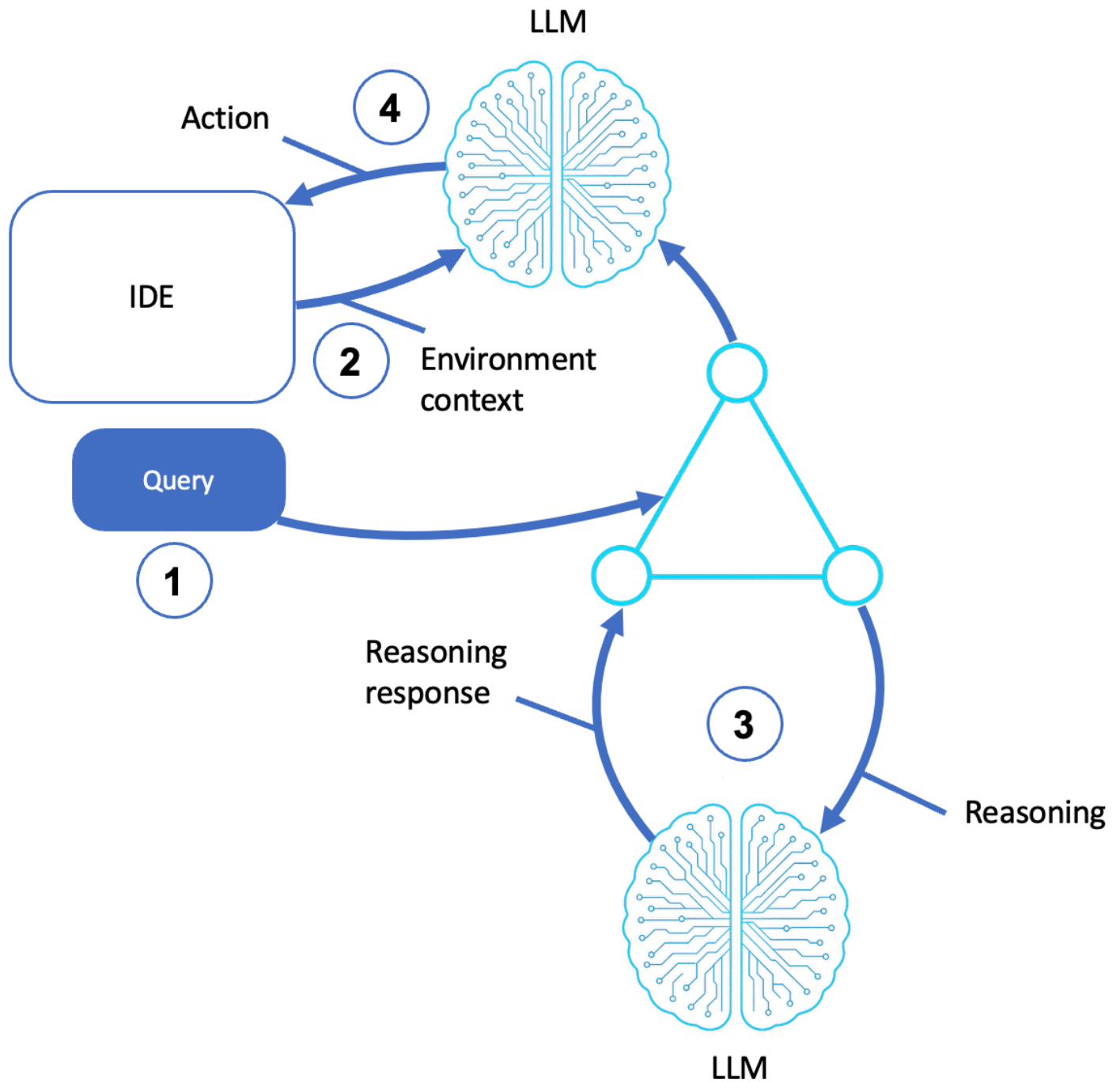
Agentes de codificación

Los agentes de codificación pueden razonar sobre las tareas de programación, generar o modificar código e interactuar con los entornos de los desarrolladores, como los IDE y las CLI. Estos agentes combinan la comprensión del lenguaje natural con el razonamiento estructurado para ayudar, aumentar y automatizar el desarrollo de software, desde la generación de funciones hasta la corrección de errores y la creación de pruebas.

A diferencia de las herramientas de autocompletado, los agentes de codificación interpretan activamente los objetivos de los usuarios, consultan el contexto del entorno de desarrollo (por ejemplo, abre archivos y rastrea los errores), identifican los requisitos y, a continuación, proponen y llevan a cabo acciones.

Arquitectura

En el siguiente diagrama se muestra un patrón de agente de codificación:



Description (Descripción)

1. Recibe la consulta

- El usuario proporciona instrucciones en lenguaje natural a través de una paleta de comandos, una ventana de chat o una CLI (por ejemplo, «Añadir registro a esta función» o «Refactorizar para facilitar la lectura»).
2. Extrae el contexto del entorno
 - El agente recopila el contexto del IDE, incluidos los archivos activos, la posición del cursor, los fragmentos de código y las tablas de símbolos.
 - Genera mensajes de error, resultados de pruebas y resultados de otros agentes.
 3. Razonamiento LLM
 - El agente envía un mensaje, que incluye la consulta y el contexto ambiental, a un LLM.
 - El LLM realiza un pase de razonamiento para determinar lo siguiente:
 - ¿Qué debe cambiar
 - ¿Cómo generar una solución
 - ¿Algún paso de refactorización, reescritura o codificación
 4. Ejecuta acciones
 - El LLM devuelve la salida al agente y la importa al IDE o al entorno de ejecución.
 - Esto puede incluir insertar o modificar código, generar comentarios o documentación y activar tareas posteriores de compilación, prueba y creación de listas.

Capacidades

- High-contextual conocimiento (por ejemplo, el estado del IDE, el cursor y el árbol de sintaxis)
- Razonamiento iterativo de objetivos y comentarios
- Planificación opcional del código y separación de acciones (por ejemplo, primero razonar y luego actuar)
- Funciona en flujos de trabajo de desarrolladores sincrónicos o asíncronos

Casos de uso comunes

- Generación de código a partir de descripciones de tareas
- Refactorización y optimización del código
- Test-case generación y validación
- Explicaciones de errores y depuración

- Asistentes de documentación
- Copilotos de programación emparejados

Guía para la implementación

- Puede crear este patrón con las siguientes herramientas y Servicios de AWS:
- Amazon Bedrock para la LLM-driven generación y el razonamiento
- Desarrollador de Amazon Q para sugerencias de codificación y finalizaciones
- AWS Lambda o Amazon Elastic Container Service (Amazon ECS) para ejecutar y probar entornos sandbox
- AWS Cloud9, extensiones de VS Code o integraciones IDE personalizadas para alojar y evaluar el contexto
- Amazon Simple Storage Service (Amazon S3) para almacenar mensajes intermedios, respuestas e historial de revisiones

Resumen

Los agentes de codificación son nuevas herramientas de AI-powered desarrollo capaces de interpretar el lenguaje natural, analizar el contexto, generar cambios de código de varios pasos e integrarse en el ciclo de vida del desarrollo del software.

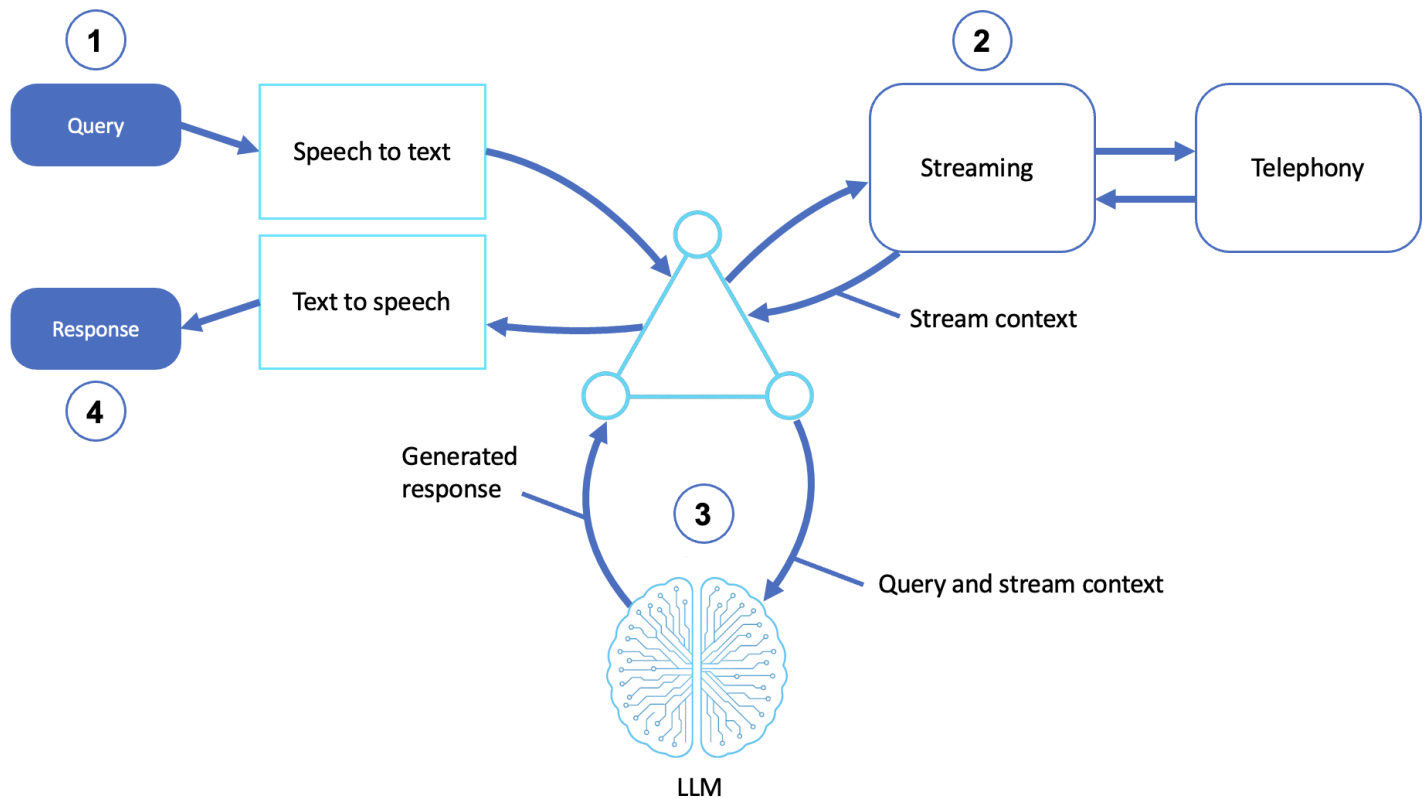
Agentes de voz y voz

Los agentes de voz y voz interactúan con los usuarios a través del diálogo oral. Estos agentes integran el reconocimiento de voz, la comprensión del lenguaje natural y la síntesis de voz para permitir la IA conversacional en plataformas de telefonía, móviles, web e integradas.

Los agentes de voz son particularmente eficaces en entornos de manos libres, en tiempo real o basados en la accesibilidad. Al combinar las interfaces de streaming con el LLM-powered razonamiento, facilitan interacciones ricas y dinámicas que resultan naturales para los usuarios.

Arquitectura

En el siguiente diagrama se muestra un agente de voz y voz:



Description (Descripción)

1. Recibe una consulta de voz

- El usuario envía una solicitud a un teléfono, micrófono o sistema integrado.
- Un módulo de voz a texto (STT) convierte el audio en texto.

2. Integra el contexto de streaming y telefonía

- El agente utiliza una interfaz de streaming para gestionar I/O el audio en tiempo real.
- Si se implementa en un contexto de contact center o de telecomunicaciones, la integración de telefonía gestiona el enrutamiento de sesiones, la entrada multifrecuencia (DTMF) de doble tono y el transporte multimedia.

Nota: El DTMF hace referencia a los tonos que se generan al pulsar los botones del teclado de un teléfono. En el contexto de la integración del contexto de la transmisión y la telefonía en los agentes de voz, el DTMF se utiliza como mecanismo de entrada de señales durante una llamada telefónica, especialmente en los sistemas de respuesta de voz interactiva (IVR). Las entradas DTMF permiten al agente:

- Reconozca las selecciones del menú (por ejemplo, «Pulse 1» para facturar). Presiona 2 para obtener asistencia»).
- Recopile entradas numéricas (por ejemplo, números de cuenta, PIN y números de confirmación)
- Activa flujos de trabajo o transiciones de estado en los flujos de llamadas
- Vuelva del habla al tono táctil cuando sea necesario

1. Razones a través del contexto de la transmisión de LLM

- La consulta se envía al agente, que la pasa, junto con los metadatos de la sesión (por ejemplo, el identificador de la persona que llama o el contexto anterior), a un LLM.
- El LLM genera una respuesta, posiblemente utilizando una estrategia de cadena de pensamiento o una memoria de varios giros si la interacción es continua.

2. Devuelve una respuesta de voz

- El agente convierte su respuesta en voz mediante la conversión de texto a voz (TTS).
- Devuelve el audio al usuario a través de un canal de voz.

Capacidades

- Real-time comprensión y generación del habla
- Multilingüe I/O con soporte para STT y TTS
- Integración con las API de telefonía o streaming
- Reconocimiento de la sesión y transferencia de memoria entre turnos

Casos de uso comunes

- Sistemas IVR conversacionales
- Recepcionistas virtuales y planificadores de citas
- Voice-driven agentes del servicio de asistencia
- Asistentes de voz portátiles
- Interfaces de voz para hogares inteligentes y herramientas de accesibilidad

Guía para la implementación

Puede crear este patrón con las siguientes herramientas y Servicios de AWS:

- Amazon Lex V2 o Amazon Transcribe para STT
- Amazon Polly para TTS
- Amazon Chime SDK, Amazon Connect Customer o Amazon Interactive Video Service (Amazon IVS) para streaming y telefonía
- Amazon Bedrock para razonar con Anthropic, AI21 u otros modelos de base
- AWS Lambda para conectar STT, LLM, TTS y el contexto de la sesión

(Opcional) Las mejoras adicionales pueden incluir las siguientes:

- Amazon Kendra o OpenSearch para RAG sensible al contexto
- Amazon DynamoDB para memoria de sesión
- Amazon CloudWatch Logs y AWS X-Ray para la trazabilidad

Resumen

Los agentes de voz y voz son sistemas inteligentes que interactúan a través de conversaciones naturales. Al integrar las interfaces de voz con la infraestructura de razonamiento y transmisión en tiempo real de LLM, los agentes de voz permiten interacciones fluidas, accesibles y escalables.

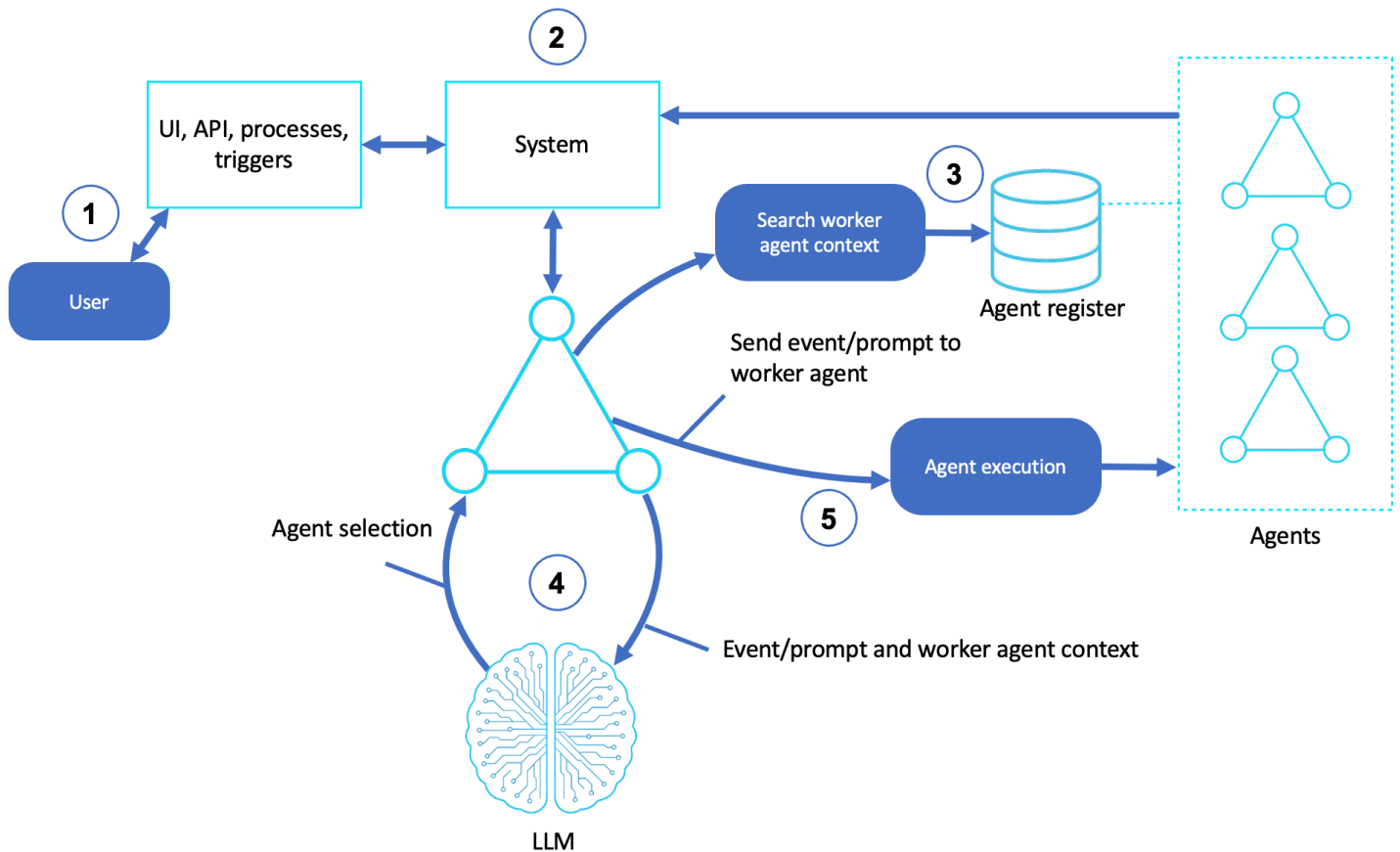
Agentes de orquestación del flujo de trabajo

Los agentes de organización del flujo de trabajo gestionan y coordinan tareas, procesos y servicios de varios pasos en sistemas distribuidos. En lugar de razonar y actuar de forma aislada, estos agentes delegan el trabajo en subagentes u otros sistemas, mantienen el contexto de ejecución y se adaptan en función de los resultados intermedios.

Estos agentes son una parte fundamental de los flujos de automatización. Son particularmente útiles cuando se gestionan tareas de larga duración, composiciones con varios agentes e integraciones entre dominios en las que es necesario utilizar varios agentes y herramientas de forma secuencial o condicional.

Arquitectura

En el siguiente diagrama se muestra un agente de orquestación del flujo de trabajo:



Description (Descripción)

1. Recibe la entrada del usuario

- Un usuario (o un desencadenador externo) inicia una tarea a través de una interfaz de usuario, una API o un evento del sistema.

2. Maneja los eventos del sistema

- Un componente del sistema recibe la solicitud y emite un evento o comando que requiere orquestación.

3. Recupera el contexto

- El agente de flujo de trabajo consulta las bases de conocimiento y los registros de agentes para encontrar el agente de trabajo adecuado para la tarea en función de los metadatos, el dominio y la tasa de éxito anterior.

4. Selecciona un agente de LLM

- Un LLM ayuda a seleccionar el mejor agente o plan de flujo de trabajo al analizar la descripción de la tarea y las opciones disponibles.
- También puede formular indicaciones específicas para cada tarea para enviarlas a un agente seleccionado.

5. Delega y ejecuta

- El agente de trabajo elegido recibe el evento o el mensaje y comienza a ejecutar los comandos.
- Puede realizar un seguimiento del estado de ejecución, volver a intentarlo en caso de error y pasar los resultados intermedios al siguiente agente de la secuencia.

Capacidades

- Composición de los agentes (por ejemplo, supervisores, agentes colaboradores y herramientas)
- Event-driven o ejecución programada
- Seguimiento de la memoria y el estado a lo largo del tiempo
- Orquestación de tareas jerárquica o paralela (flujos de trabajo síncronos en comparación con los asíncronos)
- Selección y encadenamiento dinámicos de agentes

Casos de uso comunes

- Automatización de varios pasos (por ejemplo, ingesta de datos y generación de informes)
- Enrutamiento y escalamiento del servicio de atención al cliente (por ejemplo, agente como coordinador)
- Los agentes de IA se coordinan con humanos y bots en el mismo ciclo
- Automatiza los procesos empresariales mediante LLM-powered la lógica
- Los sistemas híbridos combinan agentes de IA y herramientas de orquestación tradicionales

Guía para la implementación

Puede crear este patrón con las siguientes herramientas y Servicios de AWS:

- Amazon Bedrock para el razonamiento y la selección de agentes
- AWS Step Functions o Amazon EventBridge para la composición del flujo de trabajo

- AWS Lambda como unidades de ejecución o ejecutores de tareas
- Amazon DynamoDB, Amazon Simple Storage Service (Amazon S3) o Amazon RDS para realizar un seguimiento de los estados y los resultados
- AWS AppFabric o Amazon AppFlow para la coordinación entre sistemas
- (Opcional) Usa Amazon SageMaker run agent para alojar agentes de trabajo de dominios específicos

Resumen

Los agentes de flujo de trabajo coordinan, adaptan y alinean los objetivos en entornos con varios agentes. Esto significa que los agentes de IA pueden colaborar, adaptarse a las condiciones de tiempo de ejecución y ofrecer resultados complejos mediante flujos de trabajo modulares y explicables.

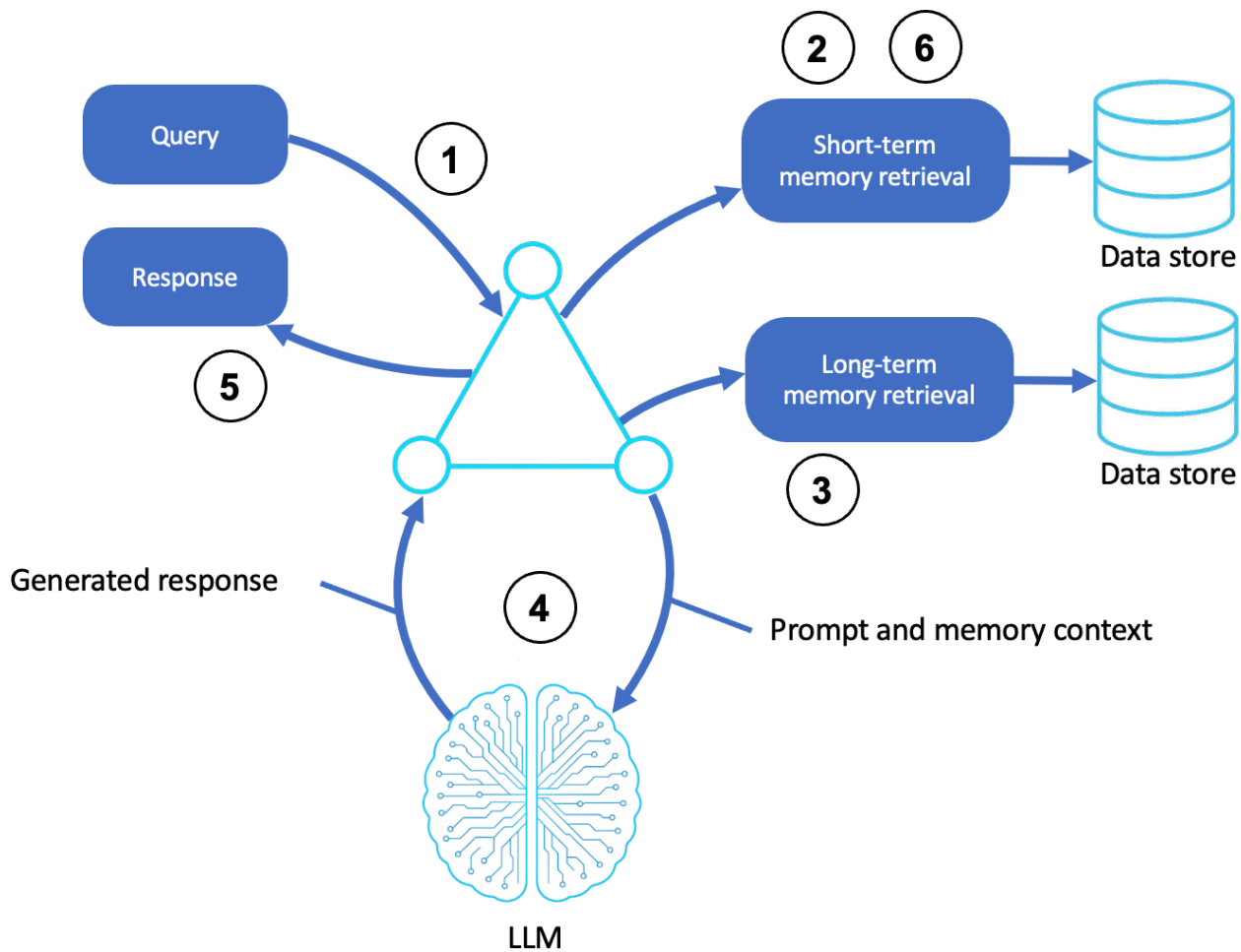
Memory-augmented agentes

Memory-augmented los agentes se han mejorado con la capacidad de almacenar, recuperar y razonar utilizando la memoria a corto y largo plazo. Esto les permite mantener el contexto en múltiples tareas, sesiones e interacciones, lo que produce respuestas más coherentes, personalizadas y estratégicas.

A diferencia de los agentes apátridas, los agentes con memoria aumentada se adaptan haciendo referencia a datos históricos, aprenden de los resultados anteriores y toman decisiones que se ajustan a los objetivos, las preferencias y el entorno del usuario.

Arquitectura

En el siguiente diagrama se muestra un agente con memoria aumentada:



Description (Descripción)

1. Recibe una entrada o un evento

- El agente recibe una consulta de usuario o un evento del sistema. Puede ser un texto, un desencadenante de la API o un cambio ambiental.

2. Recupera la memoria a corto plazo

- El agente recupera el historial de conversaciones recientes, el contexto de la tarea o el estado del sistema que sean relevantes para la sesión o el flujo de trabajo.

3. Recupera la memoria a largo plazo

- El agente consulta la memoria a largo plazo (por ejemplo, bases de datos vectoriales y almacenes de valores clave) para obtener información histórica, como la siguiente:
 - Preferencias de usuario
 - Decisiones y resultados pasados

- Conceptos, resúmenes o experiencias aprendidas

4. Razones a través del LLM

- El contexto de la memoria está integrado en el indicador LLM, lo que permite al agente razonar basándose tanto en las entradas actuales como en los conocimientos previos.

5. Genera salidas

- El agente produce una respuesta, un plan o una acción sensibles al contexto que se personaliza de acuerdo con el historial de tareas y las aportaciones del usuario.

6. Actualiza la memoria

- La información nueva, como los objetivos actualizados, las señales de éxito y fracaso y las respuestas estructuradas, se almacena para futuras tareas.

Capacidades

- Continuidad de la sesión en todas las conversaciones o eventos
- Persistencia de objetivos a lo largo del tiempo
- Conciencia contextual basada en un estado en evolución
- Adaptabilidad basada en éxitos y fracasos anteriores
- Personalización alineada con las preferencias y el historial del usuario

Casos de uso comunes

- Copilotos conversacionales que recuerdan las preferencias del usuario
- Agentes de codificación que rastrean los cambios en la base de código
- Agentes de flujo de trabajo que se adaptan según el historial de tareas
- Gemelos digitales que evolucionan a partir del conocimiento del sistema
- Investigue los agentes que eviten las recuperaciones redundantes

Implementación de agentes con memoria aumentada

Utilice las siguientes herramientas y Servicios de AWS para los agentes con memoria aumentada:

Capa de memoria

Servicio de AWS

Finalidad

Short-term	Contexto de Amazon DynamoDB, Redis y Amazon Bedrock	Recuperación rápida de los estados de interacción recientes
Long-term (estructurado)	Amazon Aurora, Amazon DynamoDB, Amazon Neptune	Datos, relaciones y registros
Long-term (semántico)	OpenSearch, PostgreSQL, Pinecone	Embedding-based recuperación (es decir, RAG)
Almacenamiento	Amazon S3	Almacenar transcripciones, memorias estructuradas y archivos
Orquestación	AWS Lambda o bien AWS Step Functions	Gestión del ciclo de vida de las inyecciones y actualizaciones de memoria
Razonamiento	Amazon Bedrock	Claude o Mistral antrópicos con indicaciones de memoria

Implementación de mensajes con inyección de memoria

Para integrar la memoria en el razonamiento de los agentes, utilice una combinación de estado estructurado e inyección de contexto con recuperación aumentada:

- Incluya el estado más reciente del agente y el historial de diálogos recientes como entrada estructurada al crear el mensaje para el modelo lingüístico, de modo que pueda razonar con todo el contexto.
- Utilice la generación de recuperación aumentada (RAG) para extraer documentos o hechos relevantes de la memoria a largo plazo.
- Resuma los planes, el contexto y las interacciones anteriores para hacerlos más comprensivos y relevantes.
- Inserte módulos de memoria externos, como almacenes vectoriales o registros estructurados, durante la inferencia para guiar la toma de decisiones.

Resumen

Memory-augmented los agentes mantienen la continuidad del pensamiento al aprender de la experiencia y recordar el contexto del usuario. Estos agentes superan la inteligencia reactiva al utilizar la colaboración a largo plazo, la personalización y el razonamiento estratégico. En términos de inteligencia artificial, la memoria permite a los agentes comportarse más como homólogos digitales adaptables y menos como herramientas apátridas.

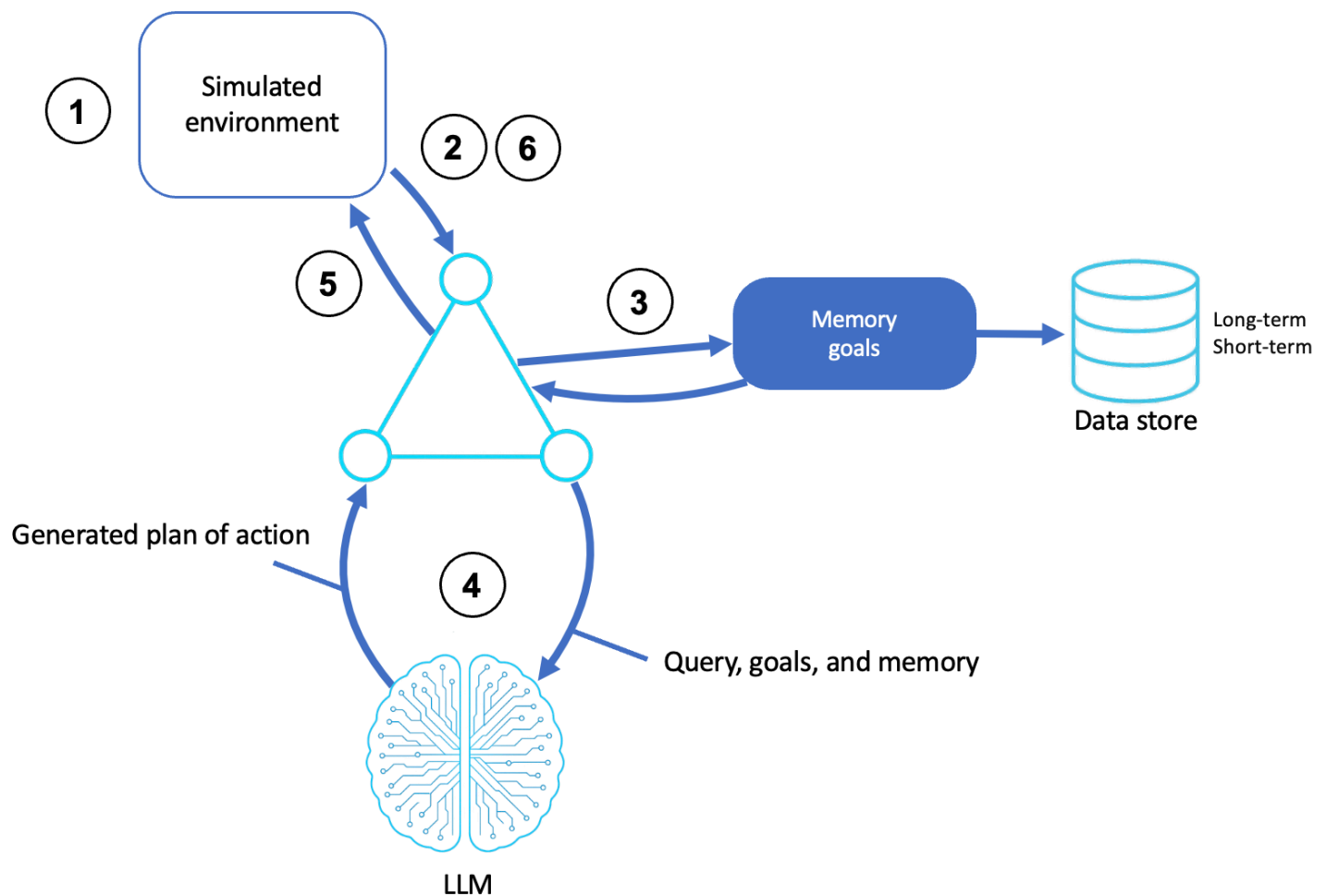
Agentes de simulación y de banco de pruebas

Los agentes de simulación y de banco de pruebas operan en entornos virtualizados o controlados donde razonan, actúan y aprenden. Estos agentes simulan el comportamiento, modelan los resultados y entrenan las estrategias en entornos repetibles antes de aplicarlos a entornos del mundo real.

Este patrón es útil para el desarrollo iterativo, el aprendizaje por refuerzo (RL), la evaluación autónoma de la toma de decisiones y las pruebas de comportamiento emergentes. Los agentes de simulación suelen operar en circuitos cerrados, reciben información de su entorno y ajustan su comportamiento en consecuencia, lo que los convierte en elementos fundamentales para las tareas que implican el razonamiento espacial, el control en tiempo real o la dinámica de sistemas complejos.

Arquitectura

El siguiente diagrama muestra un agente de simulación o de banco de pruebas:



Description (Descripción)

1. Inicia un entorno

- El agente inicia un entorno simulado (por ejemplo, un mundo 3D, un motor de física, un entorno limitado CLI o un flujo de datos sintéticos).
- El agente se carga en el entorno con una tarea, un objetivo o una política iniciales.

2. Percibe al agente

- El agente percibe el estado actual mediante telemetría de simulación (por ejemplo, emulación de sensores, cámara virtual y registros estructurados).

3. Recupera el objetivo y la memoria

- El agente recupera el objetivo asignado, las instrucciones del escenario o el objetivo contextual.
- También puede recuperar la memoria anterior, incluida la siguiente:
 - Long-term estrategias o políticas

- Mapas ambientales o restricciones conocidas
- Éxitos o fracasos pasados de simulaciones similares

4. Motivos y planes

- Un LLM interpreta el estado simulado, los objetivos de la tarea y el conocimiento aprendido.
- Genera un plan de acción o un comando de control.

5. Ejecuta acciones simuladas

- El agente ejecuta el plan, modifica el estado, navega por el espacio o interactúa con entidades virtuales.

6. Aprende

- El agente evalúa los resultados de las acciones
- Según la configuración del agente, puede hacer lo siguiente:
 - Realice RL
 - Registre los resultados para realizar ajustes en el futuro
 - Adapte las estrategias en tiempo real

Capacidades

- Funciona en entornos sintéticos o virtuales
- Soporta el aprendizaje mediante prueba y error, el perfeccionamiento de políticas y el modelado de sistemas
- Low-risk pruebas de comportamiento, gestión de fallos y casos extremos
- Permite analizar el comportamiento de los agentes emergentes en configuraciones con varios agentes
- Soporta tanto el control en circuito cerrado como la exploración humana en bucle

Casos de uso comunes

- Aprendizaje reforzado para robótica, drones y juegos
- Entrenamiento de vehículos autónomos en carreteras virtuales
- Interfaces de usuario o CLI simuladas para escenarios de banco DevOps de pruebas
- Experimentos de comportamiento emergente en simulaciones sociales
- Validación de seguridad de la lógica de decisión antes de la producción

Guía para la implementación

Puede crear un agente de simulación y de banco de pruebas utilizando las siguientes herramientas y: Servicios de AWS

Componente	Servicio de AWS	Finalidad
Entorno	Amazon ECS, Amazon EC2 o un simulador personalizado en Amazon Studio Lab SageMaker	Ejecute mundos virtuales (Gazebo, Unity, Unreal) o CLI de entorno aislado
Lógica de agentes	Amazon Bedrock SageMaker, Amazon o AWS Lambda	LLM-based planificadores o agentes de RL
Bucle de retroalimentación	Amazon SageMaker Reinforcement Learning CloudWatch, Amazon o registros personalizados	Seguimiento de recompensas, puntuación de resultados y registro del comportamiento
Memoria y reproducción	Amazon S3, Amazon DynamoDB o Amazon RDS	Datos persistentes de estado, historial de episodios o escenarios
Visualización	Cuadernos de Amazon o CloudWatch tableros de Amazon SageMaker	Observe los cambios en las políticas, los resultados y las métricas de capacitación

Las siguientes son aplicaciones adicionales:

- [AWS SimSpace Weaver](#) para simulaciones espaciales a gran escala
- [AWS IoT Core](#) para probar dispositivos de sombra
- [Amazon SageMaker Experiments](#) para la evaluación de agentes y la evaluación comparativa

Resumen

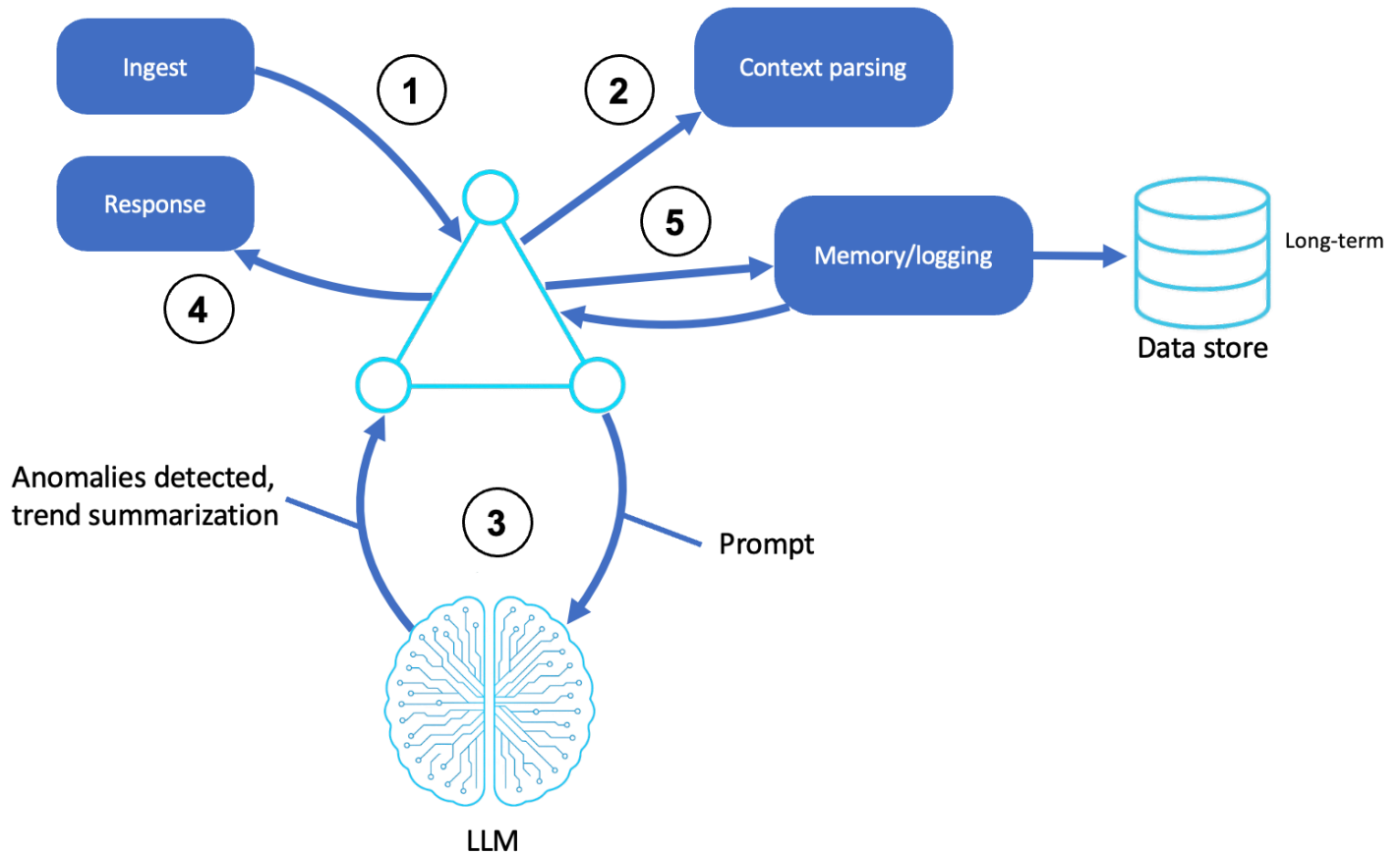
Los agentes de simulación y de banco de pruebas sirven para una exploración estructurada antes de desplegarlos en los sistemas de producción. Utilice estos agentes para entrenar políticas de navegación autónoma, probar procesos empresariales en entornos sintéticos y evaluar los patrones de coordinación de los enjambres.

Agentes de observación y monitoreo

Los agentes observadores y de monitoreo observan de forma pasiva los sistemas, los entornos y las interacciones para detectar patrones, generar información y activar acciones. Como observadores inteligentes, mejoran las alertas, los diagnósticos y las auditorías sin iniciar directamente el comportamiento.

Estos agentes se destacan cuando el monitoreo tradicional carece de adaptabilidad o razonamiento, especialmente en lo que respecta al AI-in-the-loop monitoreo, la detección de anomalías, la supervisión del cumplimiento y la inteligencia de seguridad. Los agentes observadores son detectores de eventos que monitorean continuamente la telemetría del sistema y las interacciones de los usuarios. El agente depende de la percepción, la interpretación y la intensificación condicional o la presentación de informes.

Arquitectura



Description (Descripción)

1. Telemetría de ingestión

- El agente recibe información de una o más fuentes del sistema, como las siguientes:
 - Registros (aplicación, infraestructura, seguridad)
 - Métricas (rendimiento, latencia, uso)
 - Eventos (llamadas a la API, acciones de los usuarios, datos de sensores)

2. Analice el contexto

- La entrada sin procesar se analiza, estructura y enriquece con metadatos, como la marca de tiempo, la identidad del actor, el estado del sistema y el identificador de seguimiento.

3. ¿Por qué usar LLM

- El agente utiliza un LLM o módulo lógico para interpretar las entradas analizadas mediante la identificación de anomalías, el resumen de las tendencias y la correlación entre trazas distribuidas o ventanas temporales.

4. Clasifica o alerta

- El agente determina si el comportamiento observado justifica lo siguiente:
 - ¿Una alerta o escalada
 - Una actualización de un informe o un panel
 - Un desencadenante de respuesta (por ejemplo, una corrección automática y la aplicación de políticas)

5. Registre la memoria o los bucles de retroalimentación

- El agente almacena eventos y decisiones para el aprendizaje a largo plazo, las auditorías o como referencia futura para otros agentes.

Capacidades

- Pasivo y no invasivo (el agente no actúa directamente)
- Altamente escalable y asíncrono
- AI-driven correlación entre señales ruidosas o distribuidas
- Soporta la auditoría, el cumplimiento y la información en tiempo real
- Puede alimentar los flujos de trabajo humanos o de los agentes intermedios

Casos de uso comunes

- AI-augmented observabilidad de microservicios y API
- Supervisión de la desviación del modelo, la infracción de las políticas o el comportamiento fuera de banda
- Análisis de la actividad del cliente o resúmenes de interacciones
- Agentes de revisión de código que supervisan las confirmaciones o las implementaciones
- Supervisión de los registros de seguridad o cumplimiento mediante el razonamiento LLM

Guía para la implementación

Puede crear un observador y un agente de monitoreo utilizando las siguientes herramientas y Servicios de AWS:

Componente	Servicio de AWS	Finalidad
Ingestión de eventos	Amazon EventBridge, Amazon CloudWatch Logs, Amazon Kinesis, Amazon S3	Ingiera telemetría estructurada y no estructurada
Procesamiento previo	AWS Lambda, AWS Glue, AWS Step Functions	Transforme los datos sin procesar en indicaciones estructuradas
Motor de razonamiento	Amazon Bedrock, Amazon SageMaker, AWS Lambda	Analice eventos, clasifique el comportamiento, genere información
Almacenamiento y memoria	Amazon S3, Amazon DynamoDB, OpenSearch	Observaciones, resúmenes y resultados persistentes
Alertas y escalamiento	Amazon SNS, AWS AppFabric Amazon EventBridge	Activa sistemas o agentes descendentes

Las siguientes son aplicaciones adicionales:

- [AWS Security Hub CSPM](#) para la supervisión de los registros de seguridad
- [Amazon Quick](#) para visualizar las salidas de los agentes

Resumen

Los agentes observadores y de monitoreo rastrean los sistemas y los comportamientos en tiempo real. Detectan anomalías, auditan la seguridad y recopilan información sobre las operaciones mediante la identificación de patrones que las personas o las reglas podrían pasar por alto. Esta capacidad ayuda a crear sistemas que pueden adaptarse a las condiciones cambiantes y tomar decisiones basadas en un análisis exhaustivo de los datos.

Multi-agent colaboración

Multi-agent la colaboración se refiere a un patrón en el que varios agentes autónomos, cada uno con una función, especialización u objetivo distintos, negocian para resolver tareas complejas. Estos

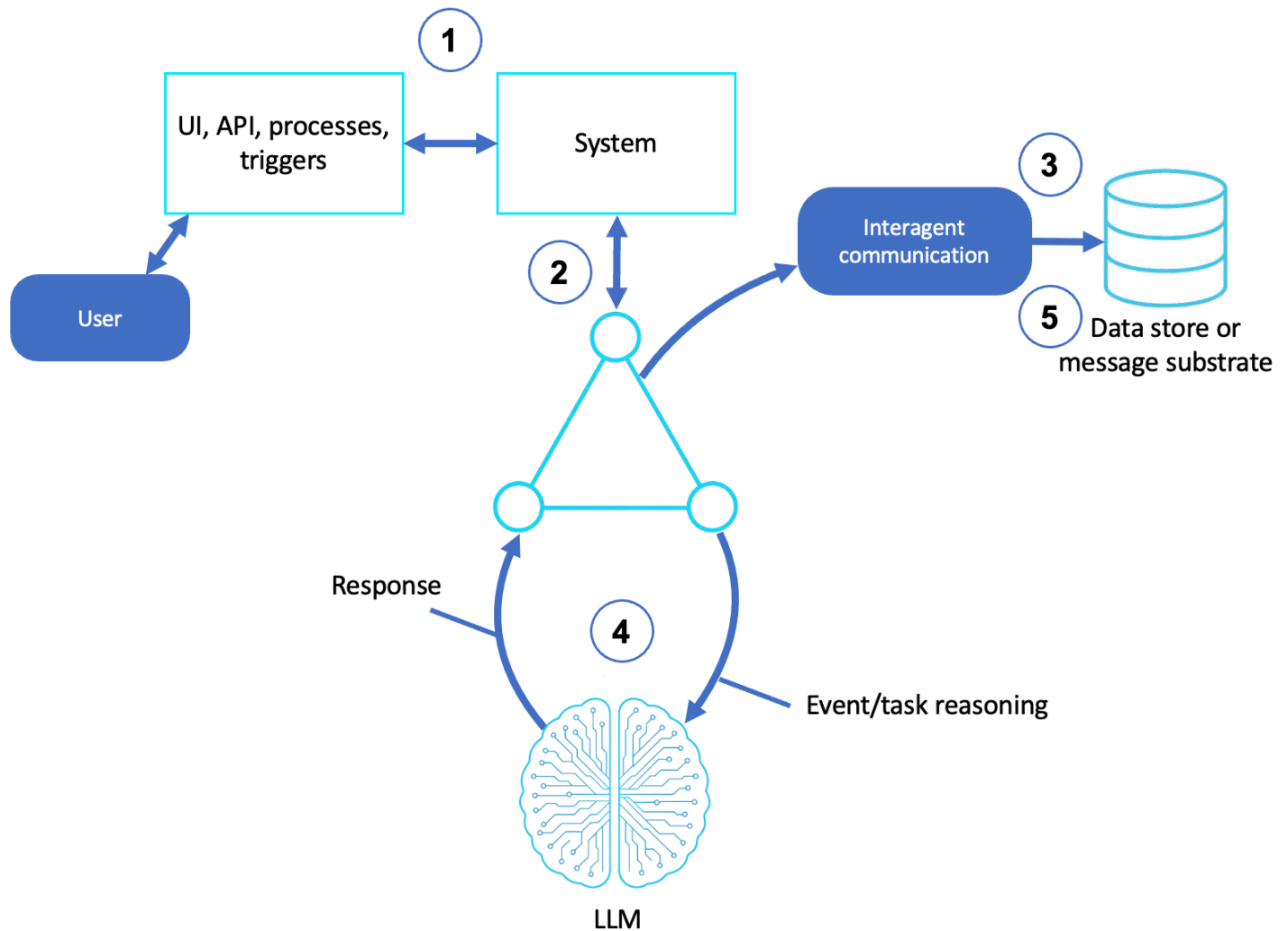
agentes pueden operar de forma independiente o con otros agentes al compartir información, dividir las responsabilidades y razonar colectivamente para alcanzar un objetivo.

Este patrón difiere del de los agentes de flujo de trabajo, que coordinan y delegan tareas de forma centralizada a los agentes subordinados en un flujo estructurado. Por el contrario, la colaboración entre múltiples agentes hace hincapié en la coordinación entre pares o emergente al permitir la adaptabilidad, el paralelismo y la división de la cognición. En la siguiente tabla se compara la colaboración entre varios agentes y los agentes de flujo de trabajo:

Característica	Agentes de flujo de trabajo	Finalidad
Controlar	Coordinador centralizado	Compañeros descentralizados, distribuidos o basados en roles
Interacción	Un agente delega y realiza un seguimiento de la ejecución	Varios agentes negocian, comparten y se adaptan
Diseño	Secuencia de tareas predefinida	Distribución de tareas flexible y emergente
Coordinación	Orquestación procedimental	Interacciones cooperativas o competitivas
Casos de uso	Automatización de procesos empresariales	Razonamiento complejo, exploración y estrategias emergentes

Arquitectura

El siguiente diagrama muestra la colaboración entre varios agentes:



Description (Descripción)

1. Inicia una tarea

- Un usuario o un sistema emite un objetivo o un problema de alto nivel.
- Un «administrador», agente o contexto iniciador define el objetivo.

2. Asigna o descubre roles

- Los agentes se autoasignan (razonamiento o lógica simbólica) o se les delega (agentes de eventos) otras funciones, como las de planificador, investigador, ejecutor, crítico o explicador.

3. Se comunica con otros agentes

- Los agentes se comunican a través de la memoria compartida, las colas de mensajes o el encadenamiento de mensajes.
- Pueden debatir, realizar consultas o proponerse subtareas entre sí.

4. Utiliza un razonamiento especializado
- Cada agente usa su propio modelo o lógica de dominio para resolver su parte del problema.
 - Los agentes pueden usar LLM con instrucciones y memoria específicas para cada función.
5. Coordina los resultados o los objetivos
- Los agentes sintetizan las contribuciones en una respuesta, un plan o una acción finales.
 - (Opcional) Un agente supervisor puede validar o resumir el resultado sintetizado.

Capacidades

- Peer-level agentes con funciones o habilidades especializadas
- Comportamiento emergente a través de la comunicación o la negociación
- Procesamiento paralelo de problemas complejos o multifacéticos
- Soporta la deliberación, la autocorrección y la iteración reflexiva
- Modele la dinámica social, la colaboración científica o las funciones del equipo empresarial

Casos de uso comunes

- Equipos de investigación autónomos (agente de búsqueda, resumidor y validador)
- Desarrollo de software (planificador, codificador y probador)
- Modelado de escenarios empresariales (finanzas, políticas y cumplimiento)
- Negociación, licitación o razonamiento multipartidista
- Tareas multimodales (imagen, texto y lógica)

Guía para la implementación

Puede crear un sistema multiagente mediante las siguientes herramientas y: Servicios de AWS

Componente	Servicio de AWS	Finalidad
Alojamiento de agentes	Amazon Bedrock, Amazon SageMaker, AWS Lambda	Aloje a agentes individuales LLM-driven

Capa de comunicación	Amazon SQS, Amazon, EventBridge AWS AppFabric	Mensajería y coordinación entre agentes
Memoria compartida	Amazon DynamoDB, Amazon S3 o OpenSearch	Multi-agent memoria o sistema de pizarra
Capa de orquestación	AWS Step Functions, AWS Lambda canalizaciones	Lógica de inicio, tiempo de espera, retroceso y reintento
Identificación del agente	Agentes de Amazon Bedrock (función definida) y API conversada de AWS AppConfig Amazon Bedrock (agentes ajenos a Amazon Bedrock)	Role-based invocación de herramientas o agentes y aplicación de límites
Interacción emergente	Amazon EventBridge Pipelines o registros de agentes	Habilite el enrutamiento o la escalación dinámicos de tareas

Resumen

Multi-agent la colaboración distribuye las tareas de resolución de problemas entre agentes modulares y basados en funciones. A diferencia de la organización del flujo de trabajo, los patrones de colaboración utilizan inteligencia, resiliencia y escalabilidad emergentes que reflejan la forma en que las personas resuelven los problemas. Es especialmente valioso para dominios abiertos, tareas creativas, razonamiento multimodal y entornos que se benefician de diversas perspectivas.

Conclusión

Los patrones analizados anteriormente ilustran los enfoques fundamentales de las implementaciones de la IA de los agentes en el mundo real. Desde el razonamiento básico hasta la inteligencia con memoria aumentada, cada patrón está configurado de forma única para la percepción, la cognición y la acción, y se basa en la autonomía, la asincronía y la agencia.

Estos patrones comparten vocabularios y modelos técnicos para crear sistemas inteligentes y orientados a objetivos. Ya sea que un patrón esté integrado en una interfaz de usuario, se organice

mediante servicios en la nube o se coordine entre equipos de agentes, cada patrón es adaptable y modular.

Conclusiones

- Los patrones de los agentes son componibles: la mayoría de los agentes del mundo real combinan dos o más patrones (por ejemplo, un agente de voz con memoria y razonamiento basados en herramientas).
- El diseño del agente es contextual: elija patrones en función de la superficie de interacción, la complejidad de la tarea, la tolerancia a la latencia y las restricciones específicas del dominio.
- AWS La implementación nativa se puede lograr: con Amazon Bedrock SageMaker, Amazon y las arquitecturas basadas en eventos AWS Lambda AWS Step Functions, todos los patrones de agentes se pueden ofrecer a escala.

Flujos de trabajo LLM

En cuanto a los patrones de los agentes, exploramos los patrones más comunes de los agentes de la IA, cada uno creado en torno a un conjunto de capacidades modulares: percepción, acción, aprendizaje y cognición. En el centro del módulo cognitivo de muchos patrones de agentes se encuentra un modelo de lenguaje amplio (LLM) que es capaz de razonar, planificar y tomar decisiones. Sin embargo, invocar un LLM por sí solo no es suficiente para producir un comportamiento inteligente y dirigido a objetivos.

Para realizar tareas complejas de forma fiable, los agentes deben integrar el LLM en un flujo de trabajo estructurado, en el que las capacidades del modelo se amplíen con herramientas, memoria, ciclos de planificación y lógica de coordinación. Estos flujos de trabajo de LLM permiten a un agente desglosar los objetivos, enrutar las subtarear, llamar a servicios externos, reflexionar sobre los resultados y coordinarse con otros agentes.

Este capítulo presenta los patrones de diseño básicos para crear módulos LLM-driven cognitivos robustos, ampliables e inteligentes, organizados en torno a flujos de trabajo reutilizables.

En esta sección

- [Descripción general de la cognición LLM-augmented](#)
- [Flujo de trabajo para un encadenamiento rápido](#)
- [Flujo de trabajo para enrutamiento](#)
- [Flujo de trabajo para la paralelización](#)
- [Flujo de trabajo para la orquestación](#)
- [Flujo de trabajo para evaluadores y ciclos de reflexión y refinamiento](#)
- [Conclusión](#)

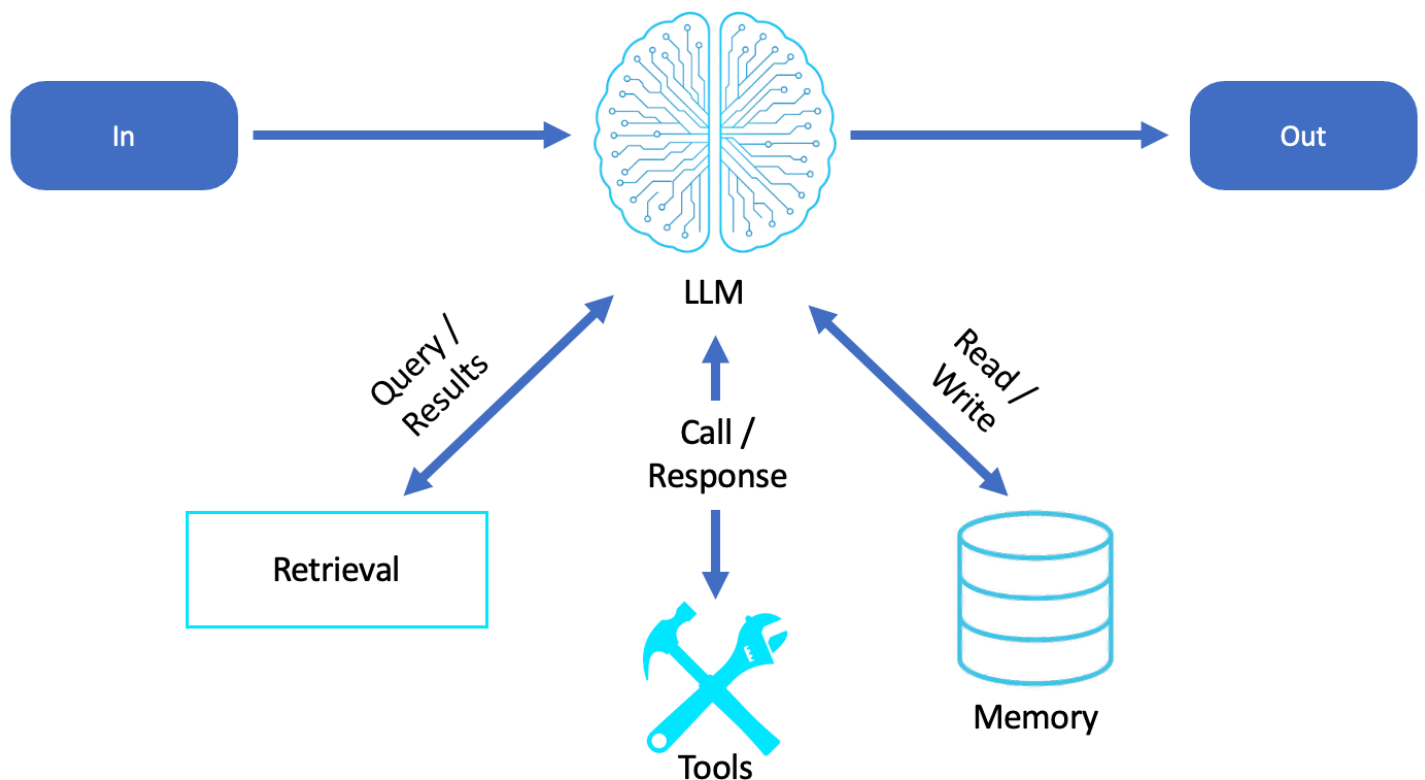
Descripción general de la cognición LLM-augmented

En esencia, el módulo cognitivo de un agente de software puede considerarse un LLM repleto de mejoras. El agente puede utilizar los siguientes componentes básicos para razonar eficazmente en su entorno:

- **Solicitud:** enmarcar la entrada utilizando el contexto, las instrucciones, los ejemplos y la memoria

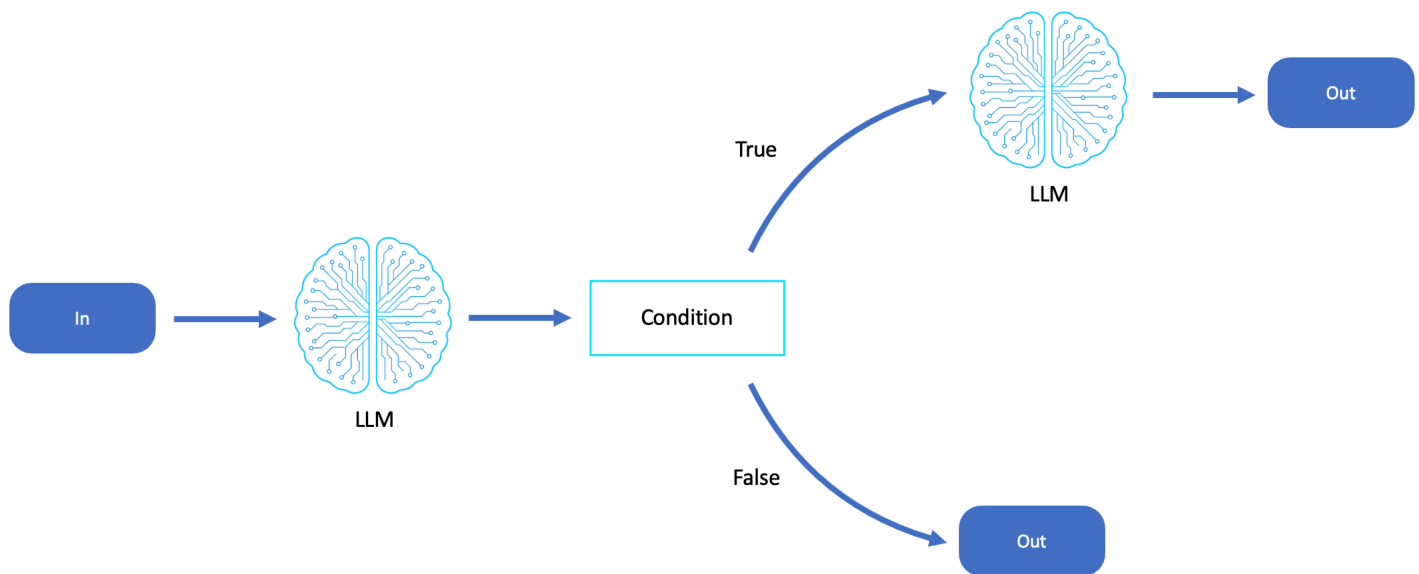
- Recuperación: proporcionar conocimientos actualizados o específicos de un dominio a la solicitud del LLM mediante la búsqueda vectorial o la memoria semántica, por ejemplo, mediante la generación de recuperación aumentada (RAG)
- Uso de herramientas: permite al LLM invocar API o llamar a funciones para recuperar información o actuar en función de ella
- Memoria: incorporar un estado persistente o basado en una sesión en el ciclo de razonamiento, ya sea mediante bases de datos estructuradas o resúmenes contextuales

Estos aumentos se componen de flujos de trabajo que definen cómo se utiliza el LLM a lo largo del tiempo y en todas las tareas, lo que lo transforma de un motor sin estado en un agente de razonamiento dinámico.



Flujo de trabajo para un encadenamiento rápido

El encadenamiento rápido descompone las tareas complejas en una secuencia de pasos, donde cada paso es una invocación discreta de LLM que procesa o se basa en el resultado de la anterior.



El flujo de trabajo de encadenamiento rápido es adecuado para escenarios en los que las tareas se pueden dividir de forma lógica en pasos de razonamiento secuenciales y en los que los resultados intermedios sirven de base para la siguiente etapa. Destaca en los flujos de trabajo que requieren un pensamiento estructurado, una transformación progresiva o un análisis por capas, como la revisión de documentos, la generación de código, la extracción de conocimientos y el perfeccionamiento del contenido.

Description (Descripción)

- La complejidad de la tarea supera la ventana de contexto o la profundidad de razonamiento de una sola llamada de LLM.
- Los resultados de un paso (por ejemplo, el análisis, el resumen o la planificación) se convierten en insumos para una fase de generación o decisión de seguimiento.
- Se necesita transparencia y control en todas las etapas del razonamiento (por ejemplo, resultados intermedios auditables).
- Desea conectar la lógica externa de validación, filtrado o enriquecimiento entre los pasos.
- Es ideal para los agentes que trabajan en ciclos de razonamiento de tipo continuo, como los agentes de investigación, los asistentes de redacción, los sistemas de planificación y los copilotos de varias etapas.

Capacidades

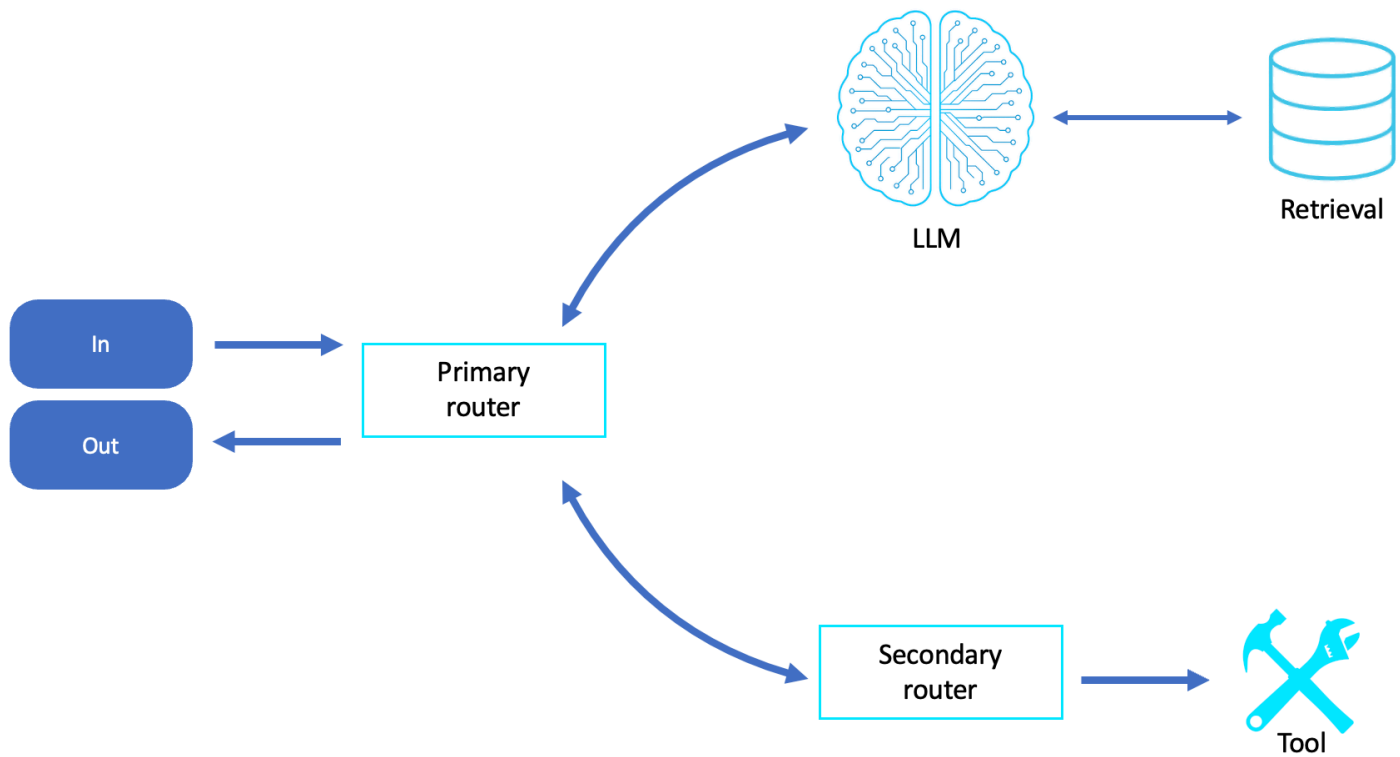
- Cadenas lineales o ramificadas de llamadas LLM
- Los resultados intermedios se transmiten como datos estructurados o se incluyen en las instrucciones de seguimiento
- Se puede orquestar con corredores o con AWS Step Functions un AWS Lambda agente específico

Casos de uso comunes

- Tareas de razonamiento de varios pasos (por ejemplo, «resumir, criticar, reescribir»)
- Los asistentes de investigación sintetizan resultados estratificados (por ejemplo, «buscar, extraer datos, responder a una pregunta»)
- Procesos de generación de código («generar un plan, escribir el código, probar el código, explicar el resultado»)

Flujo de trabajo para enrutamiento

En el patrón de enrutamiento, un clasificador o agente de enrutamiento utiliza un LLM para interpretar la intención o la categoría de una consulta y, a continuación, enruta la entrada a un agente o tarea posterior especializado.



El flujo de trabajo de enrutamiento se utiliza en situaciones en las que un agente debe clasificar rápidamente la intención de entrada, el tipo de tarea o el dominio y, a continuación, delegar la solicitud a un subagente, herramienta o flujo de trabajo especializados. Resulta especialmente útil en el caso de los agentes de capacidades, como los que actúan como asistentes generales, son los encargados de dar acceso a las funciones empresariales o las interfaces de IA orientadas al usuario que abarcan varios dominios.

El enrutamiento es particularmente efectivo cuando:

- Clasifica las solicitudes en función de una variedad de tareas (por ejemplo, búsqueda, resumen, reserva o cálculos).
- Las entradas deben preprocesarse o normalizarse antes de entrar en flujos de trabajo más especializados.
- Los diferentes tipos de entrada (por ejemplo, imágenes frente a texto, consultas estructuradas frente a consultas no estructuradas) requieren un tratamiento personalizado.
- Un agente actúa como una centralita conversacional y delega tareas a agentes especializados o microservicios.

- Este flujo de trabajo es común en los copilotos de dominios específicos, los bots de atención al cliente, los enrutadores de servicios empresariales y los agentes multimodales, donde el despacho inteligente determina tanto la calidad como la eficiencia del comportamiento de los agentes.

Capacidades

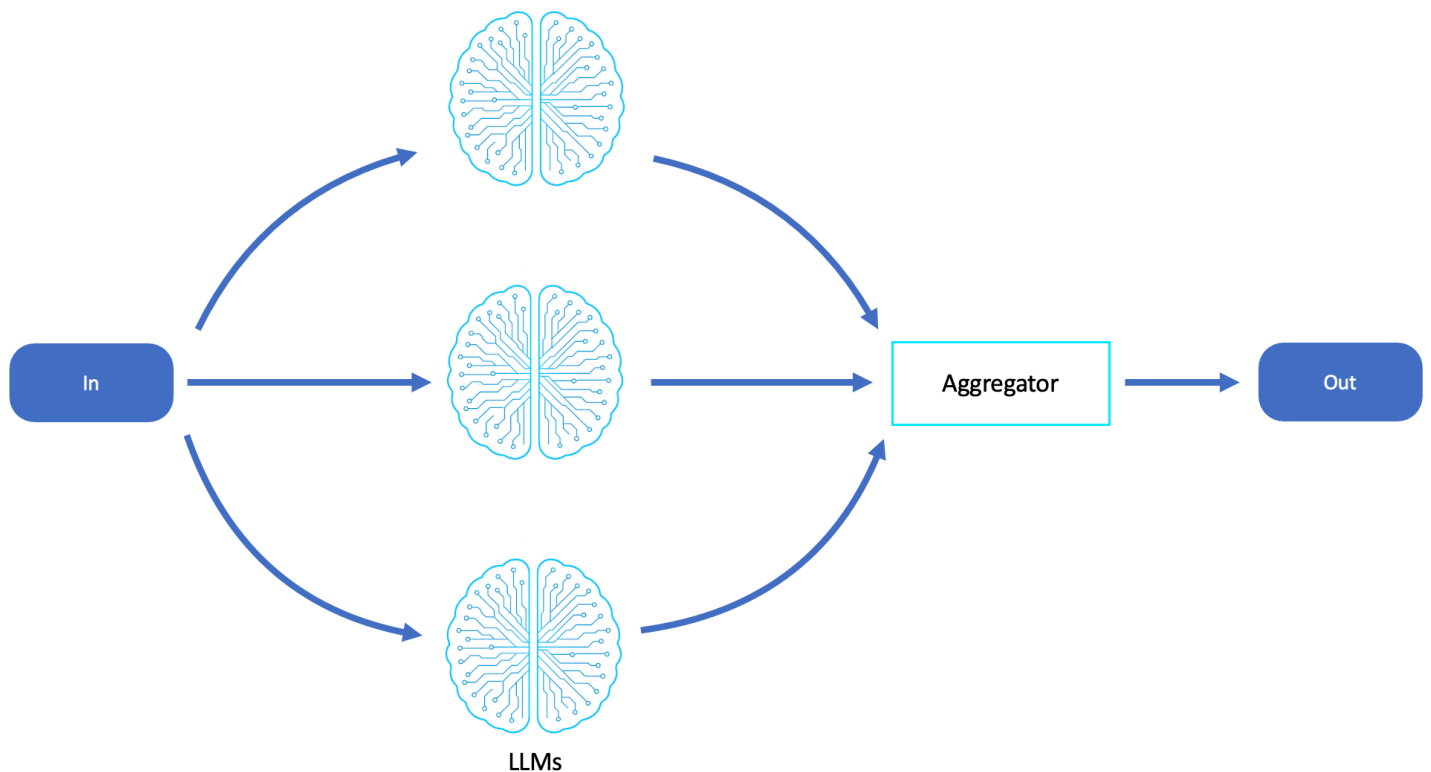
- Un LLM de primer paso actúa como despachador
- Las rutas pueden invocar flujos de trabajo distintos o incluso otros patrones de agentes
- Soporta la expansión modular de las capacidades

Casos de uso comunes

- Asistentes multidominio («¿se trata de una cuestión legal, médica o financiera?»)
- Árboles de decisión mejorados con el razonamiento LLM
- Selección dinámica de herramientas (por ejemplo, búsqueda frente a generación de código)

Flujo de trabajo para la paralelización

Este flujo de trabajo implica dividir una tarea en subtareas independientes que pueden ser gestionadas simultáneamente por varios agentes o llamadas de LLM. Luego, los resultados se agregan mediante programación y se sintetizan en un resultado.



El flujo de trabajo de paralelización se utiliza cuando una tarea se puede dividir en subtarefas independientes y no secuenciales que se pueden procesar simultáneamente, lo que mejora significativamente la eficiencia, el rendimiento y la escalabilidad. Es especialmente eficaz en espacios problemáticos con gran cantidad de datos, orientados a lotes o con múltiples perspectivas, en los que el agente debe analizar o generar contenido a partir de múltiples entradas.

La paralelización es particularmente eficaz cuando:

- Las subtarefas no dependen de los resultados intermedios de las demás, lo que les permite ejecutarse en paralelo sin coordinación.
- Una tarea implica repetir el mismo proceso de razonamiento en varios elementos (por ejemplo, resumir varios documentos o evaluar una lista de opciones).
- Se exploran múltiples hipótesis o perspectivas en paralelo para promover la diversidad, la creatividad o la solidez.
- Debe reducir la latencia de las solicitudes de gran volumen o alta frecuencia mediante la ejecución simultánea de la LLM.
- Este flujo de trabajo se suele utilizar en agentes de procesamiento de documentos, motores de encuestas o comparación, resumidores de lotes, sesiones de intercambio de ideas con varios

agentes y tareas de clasificación o etiquetado escalables, especialmente cuando el razonamiento rápido y paralelo es una ventaja de rendimiento.

Capacidades

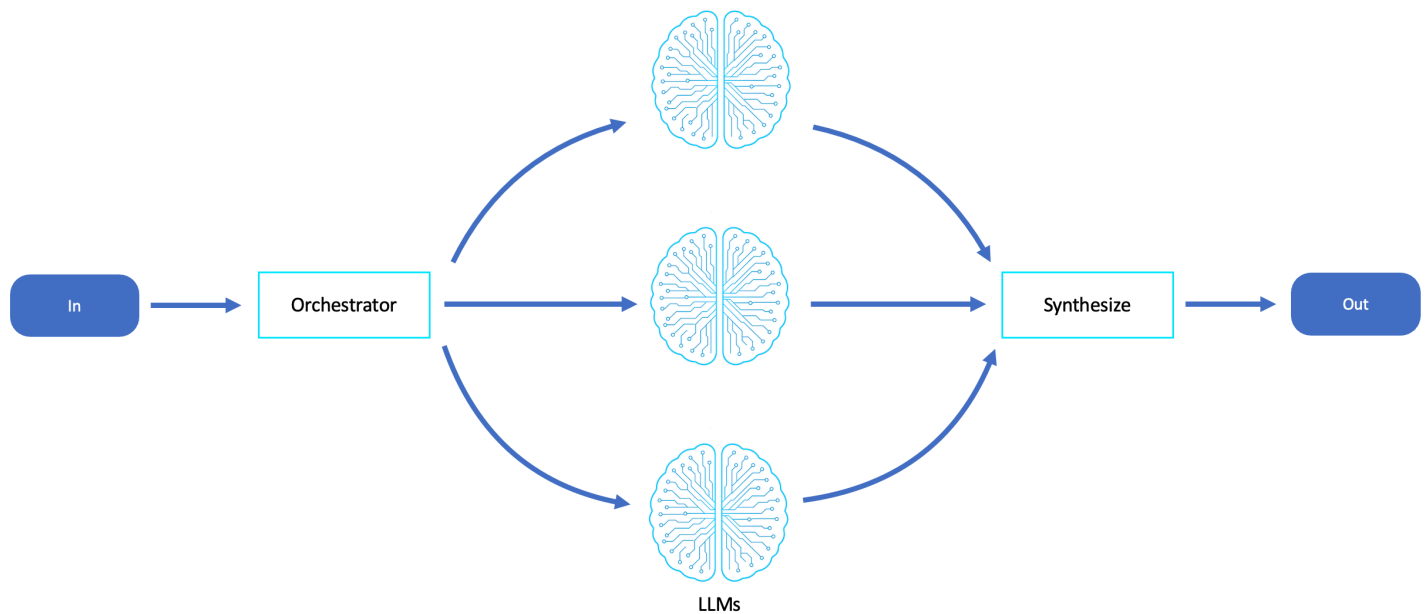
- Ejecución paralela de tareas de LLM (mediante el uso de AWS Lambda, AWS Fargate o un estado del mapa) AWS Step Functions
- Requiere alinear, validar o deduplicar los resultados en la etapa de síntesis
- Well-suited para bucles de agentes sin estado

Casos de uso comunes

- Analizar varios documentos o perspectivas en paralelo
- Generar diversos borradores, resúmenes o planes
- Acelerar el rendimiento en los trabajos por lotes

Flujo de trabajo para la orquestación

Un agente orquestador central utiliza un LLM para planificar, descomponer y delegar subtareas en agentes o modelos trabajadores especializados, cada uno con una función específica o experiencia en el campo. Esto refleja las estructuras de los equipos humanos y apoya el comportamiento emergente de varios agentes.



El flujo de trabajo de orquestación es ideal para escenarios complejos, jerárquicos o multidisciplinarios que requieren una descomposición estructurada y una ejecución especializada. Es especialmente adecuado para tareas que requieren división del trabajo, en las que es mejor que agentes con distintas capacidades, conocimientos o conjuntos de herramientas se encarguen mejor de los diferentes subcomponentes de una tarea.

Este flujo de trabajo es particularmente eficaz cuando:

- Las tareas se pueden dividir en subtareas que varían en cuanto a alcance, tipo o razonamiento (por ejemplo, planificar, investigar, implementar y probar).
- Un LLM o un metaagente debe coordinar a otros agentes, supervisar el progreso y sintetizar los resultados.
- Desea modular las responsabilidades de los agentes para permitir la escalabilidad, la reutilización y el ajuste especializado.
- El sistema requiere un comportamiento basado en roles, que imite la forma en que los equipos humanos (por ejemplo, directores de proyectos, desarrolladores y revisores) trabajan en colaboración.

Orchestration es ideal para agentes de planificación con múltiples turnos, copilotos de desarrollo de software, agentes de procesos empresariales y ejecutores de proyectos autónomos. Resulta especialmente útil cuando se implementan sistemas con varios agentes que requieren un desglose

centralizado de las tareas pero una lógica de ejecución distribuida, lo que permite una mayor extensibilidad y un comportamiento más explicable en todos los niveles de agentes.

Capacidades

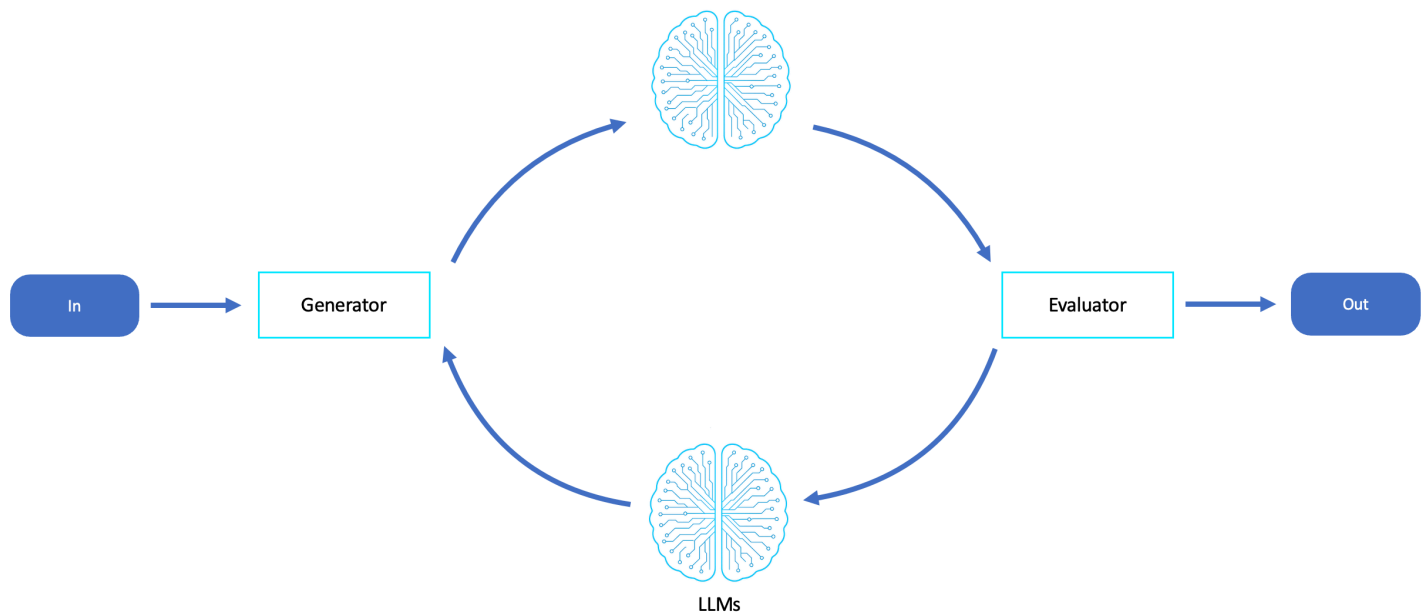
- Orchestrator lleva a cabo un metarrazonamiento de objetivos
- Los agentes de trabajo pueden incluir el acceso a las herramientas, la memoria o las solicitudes específicas de un dominio
- Puede ser jerárquico (es decir, delegación de tareas en varios niveles)

Casos de uso comunes

- Directores de proyectos, investigadores coordinadores, redactores y agentes de control de calidad
- Copilotos de codificación que combinan la planificación, la ejecución y las pruebas
- Agentes que supervisan las cadenas de herramientas o los patrones de acceso a las API

Flujo de trabajo para evaluadores y ciclos de reflexión y refinamiento

Este flujo de trabajo proporciona un circuito de retroalimentación en el que un LLM genera un resultado y otro evalúa o critica el resultado. Esto promueve la autorreflexión, la optimización y las mejoras iterativas.



El flujo de trabajo del evaluador es ideal para escenarios en los que la calidad, la precisión y la alineación de los resultados son importantes y en los que la generación de una sola pasada no es fiable o insuficiente. Este flujo de trabajo destaca cuando los agentes deben autocriticarse, repetir y refinar sus resultados, ya sea para cumplir con un estándar más alto de corrección o para explorar alternativas mejoradas en función de los comentarios.

Este flujo de trabajo es particularmente eficaz cuando:

- El resultado incluye métricas de calidad subjetivas (por ejemplo, estilo, tono y legibilidad) o criterios objetivos (por ejemplo, corrección, seguridad y rendimiento).
- El agente debe razonar haciendo concesiones, evaluar las limitaciones u optimizar la búsqueda de una meta.
- Necesita redundancia y garantía de calidad integradas, especialmente en los ámbitos regulados, orientados al cliente o creativos.
- Human-in-the-loop la revisión es cara o no está disponible, y se desea una validación autónoma.

Este flujo de trabajo se utiliza para la generación de contenido, la síntesis y revisión del código, la aplicación de políticas, la comprobación de la alineación, el ajuste de las instrucciones y el posprocesamiento del RAG. También es útil para los agentes que se mejoran a sí mismos, ya que la retroalimentación continua ayuda a dar forma a mejores respuestas a lo largo del tiempo para crear ciclos de decisión autónomos y confiables.

Casos de uso comunes

- Red-team agentes en comparación con los agentes del equipo azul
- Agentes que generan, evalúan y revisan códigos o planes
- Control de calidad, detección de alucinaciones y control del estilo

Capacidades

- Soporta la generación y la evaluación disociadas utilizando diferentes modelos (por ejemplo, Claude para la generación y Mistral para la evaluación)
- La retroalimentación está estructurada y se utiliza para generar resultados revisados
- Soporta múltiples iteraciones o umbrales de convergencia

Conclusión

Los LLM constituyen el núcleo cognitivo de los agentes de software modernos, pero la invocación de modelos sin procesar no es suficiente para lograr una inteligencia sólida, controlable y útil. Para pasar de la generación de resultados al razonamiento estructurado y a un comportamiento alineado con los objetivos, los LLM deben integrarse en patrones de flujo de trabajo intencionales que definan la forma en que los modelos procesan las entradas, gestionan los contextos y coordinan las acciones.

Los flujos de trabajo de la LLM introducen las bases para construir el módulo cognitivo de un agente:

- El encadenamiento rápido divide el razonamiento complejo en pasos modulares y auditables.
- El enrutamiento permite una clasificación inteligente de las tareas y una delegación específica.
- La paralelización acelera el rendimiento y promueve la diversidad de razonamientos.
- La orquestación de agentes estructura la colaboración entre varios agentes mediante la descomposición de tareas y la ejecución basada en roles.
- El evaluador (bucle reflect-refine) permite la superación personal, el control de calidad y la comprobación de la alineación.

Cada flujo de trabajo representa un patrón componible que se puede adaptar a las necesidades del agente, a la complejidad de la tarea y a las expectativas del usuario. Estos flujos de trabajo no se excluyen mutuamente. Son componentes básicos que a menudo se combinan en arquitecturas

híbridas que respaldan el razonamiento dinámico, la coordinación entre múltiples agentes y la confiabilidad de nivel empresarial.

A medida que pase al siguiente capítulo sobre los patrones de flujo de trabajo de los agentes, estos flujos de trabajo LLM volverán a aparecer como estructuras integradas en sistemas más grandes, lo que favorecerá la delegación de objetivos, la organización de herramientas, los ciclos de decisión y la autonomía del ciclo de vida. Dominar estos flujos de trabajo de LLM es esencial para diseñar agentes de software que no se limiten a predecir el texto, sino que también razonen, se adapten y actúen con determinación.

Patrones de flujo de trabajo de los agentes

Los patrones de flujo de trabajo de Agentic integran agentes de software modulares con flujos de trabajo estructurados de modelos de lenguaje grande (LLM), lo que permite razonar y actuar de forma autónoma. Si bien se inspiran en las arquitecturas tradicionales sin servidor y basadas en eventos, estos patrones cambian la lógica central del código estático a los LLM-augmented agentes, lo que mejora la adaptabilidad y la toma de decisiones contextuales. Esta evolución transforma las arquitecturas de nube convencionales, pasando de ser sistemas deterministas a sistemas capaces de realizar interpretaciones dinámicas y ampliaciones inteligentes, manteniendo al mismo tiempo los principios fundamentales de escalabilidad y capacidad de respuesta.

En esta sección

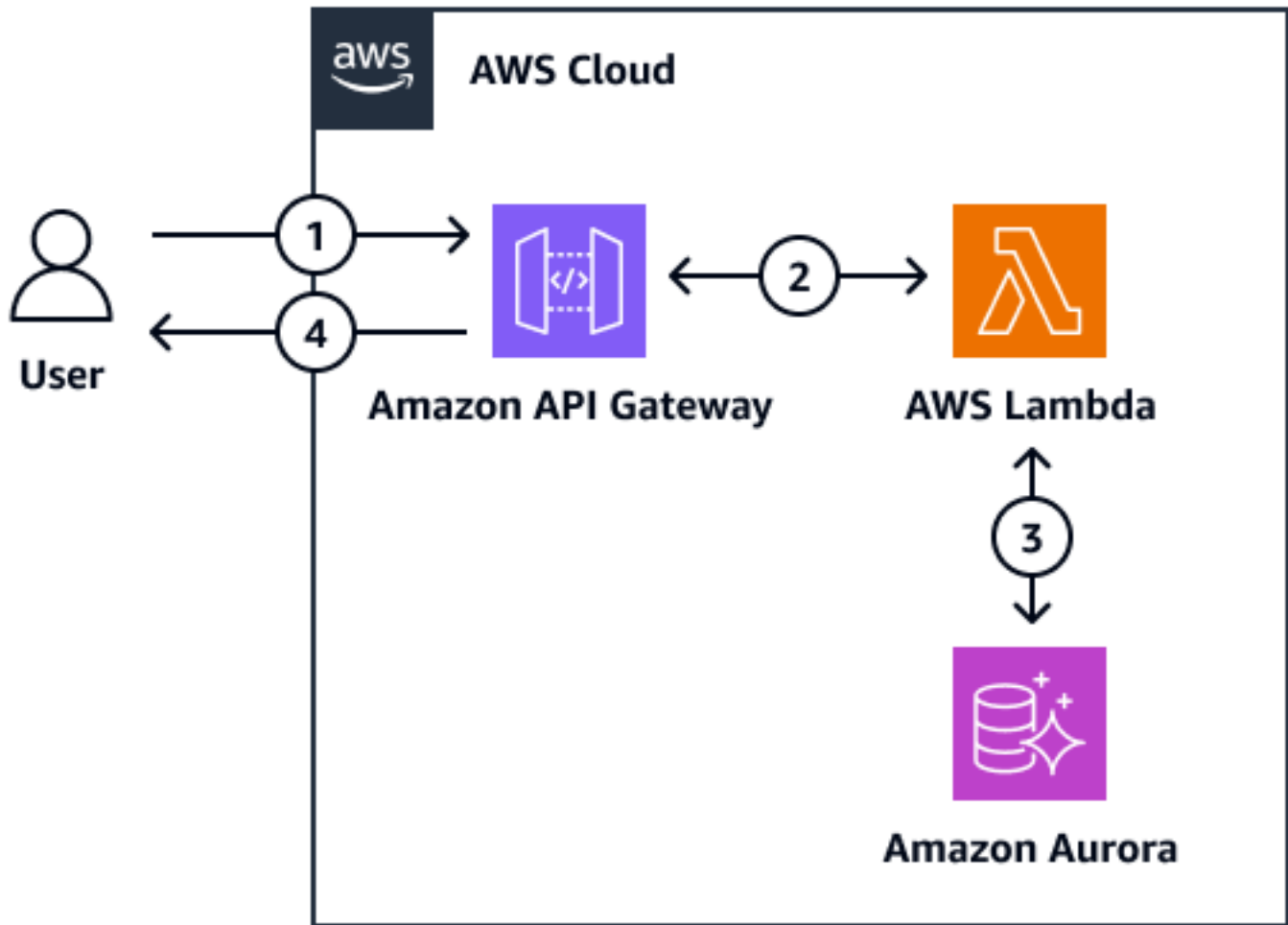
- [Desde sistemas basados en eventos hasta sistemas con cognición aumentada](#)
- [Patrones de saga que se encadenan rápidamente](#)
- [Enrutamiento de patrones de despacho dinámicos](#)
- [Patrones de paralelización y dispersión](#)
- [Patrones de orquestación de Saga](#)
- [El evaluador refleja y refina los patrones de bucles](#)
- [Diseñar los flujos de trabajo de los agentes en AWS](#)
- [Conclusión](#)

Desde sistemas basados en eventos hasta sistemas con cognición aumentada

Las arquitecturas de nube modernas, especialmente las que se basan en los principios de la ausencia de servidores y las basadas en eventos, se han basado tradicionalmente en patrones como el enrutamiento, la distribución y el enriquecimiento para crear sistemas escalables y con capacidad de respuesta. Los sistemas de inteligencia artificial de las agencias se basan en estos fundamentos y, al mismo tiempo, los reformulan en torno al razonamiento y la flexibilidad cognitiva. LLM-augmented Este enfoque permite capacidades de automatización y resolución de problemas más sofisticadas, lo que podría revolucionar la forma en que se gestionan las tareas complejas en los entornos de nube.

Event-driven arquitectura

El siguiente diagrama muestra un sistema distribuido típico:

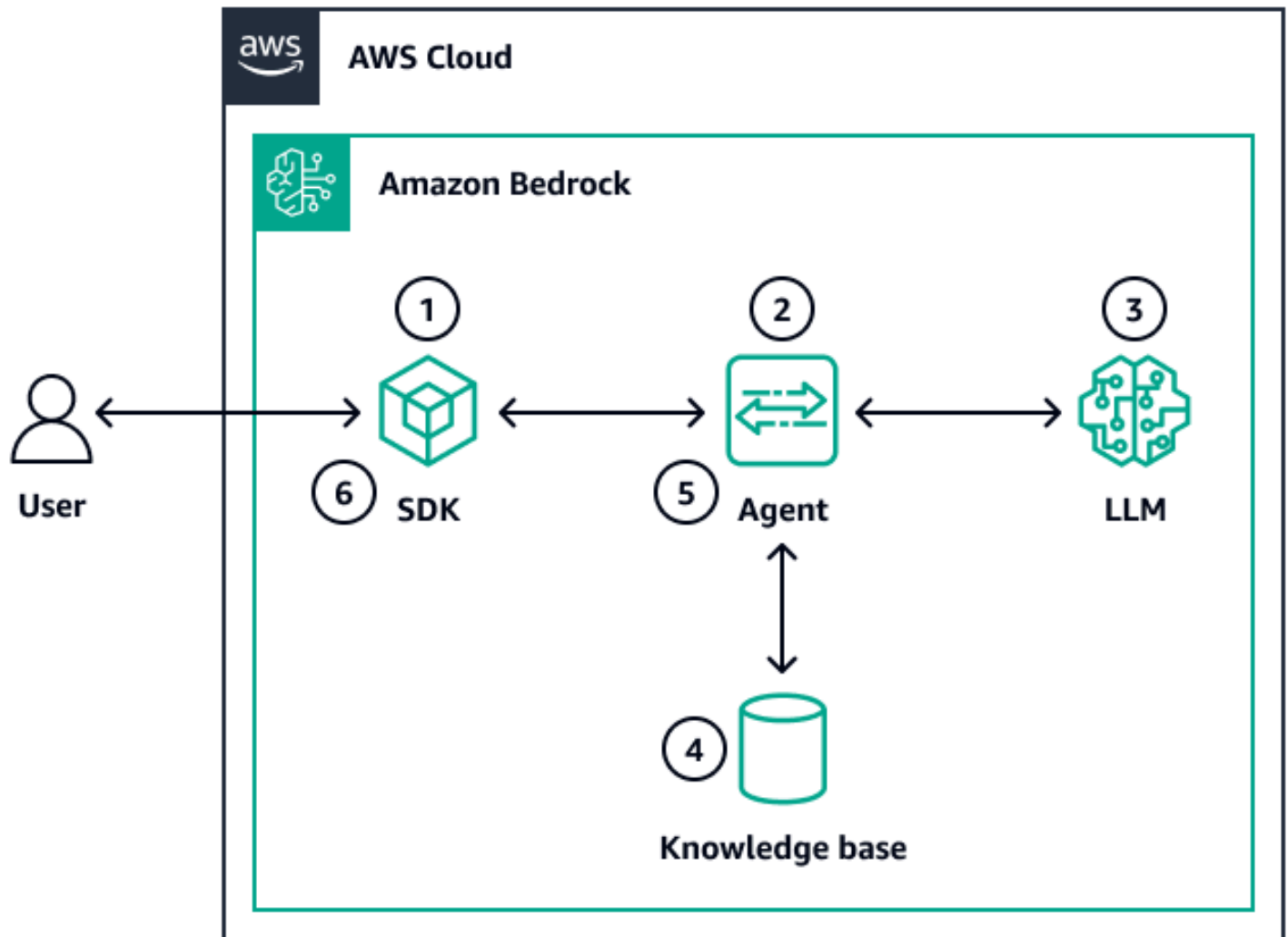


1. Un usuario envía una solicitud a Amazon API Gateway.
2. Amazon API Gateway enruta la solicitud a una función AWS Lambda.
3. AWS Lambda enriquece los datos consultando una base de datos de Amazon Aurora.
4. Amazon API Gateway devuelve la carga útil enriquecida a la persona que llama.

Esta estructura es fiable y escalable, pero es fundamentalmente estática. Las reglas empresariales y las rutas lógicas deben codificarse de forma explícita, y la adaptación a los contextos cambiantes o a la información incompleta es limitada.

Cognition-augmented flujos de trabajo

Las arquitecturas de los agentes añaden un aumento cognitivo a un sistema basado en eventos. El siguiente diagrama muestra un equivalente de un agente:



1. Un usuario envía una consulta a través de una llamada al SDK o a la API.
2. Un agente de Amazon Bedrock recibe la consulta.
3. El agente interpreta la consulta invocando un LLM
4. El agente realiza un enriquecimiento semántico buscando en la base de conocimiento de Amazon Bedrock u otras fuentes de datos externas.
5. El LLM sintetiza una respuesta rica en contexto y alineada con los objetivos.
6. El sistema devuelve una respuesta sintetizada al usuario.

En este flujo, el LLM utiliza la lógica, entiende la intención, recupera y combina el contexto relevante y, a continuación, decide la mejor manera de responder. Este patrón refleja el patrón de enriquecimiento tradicional, en el que los mensajes se complementan con datos externos antes de seguir enrutándolos. Sin embargo, en los sistemas de agencia, este enriquecimiento no es una búsqueda estática. Por el contrario, el enriquecimiento es dinámico, está guiado semánticamente y está impulsado por un propósito.

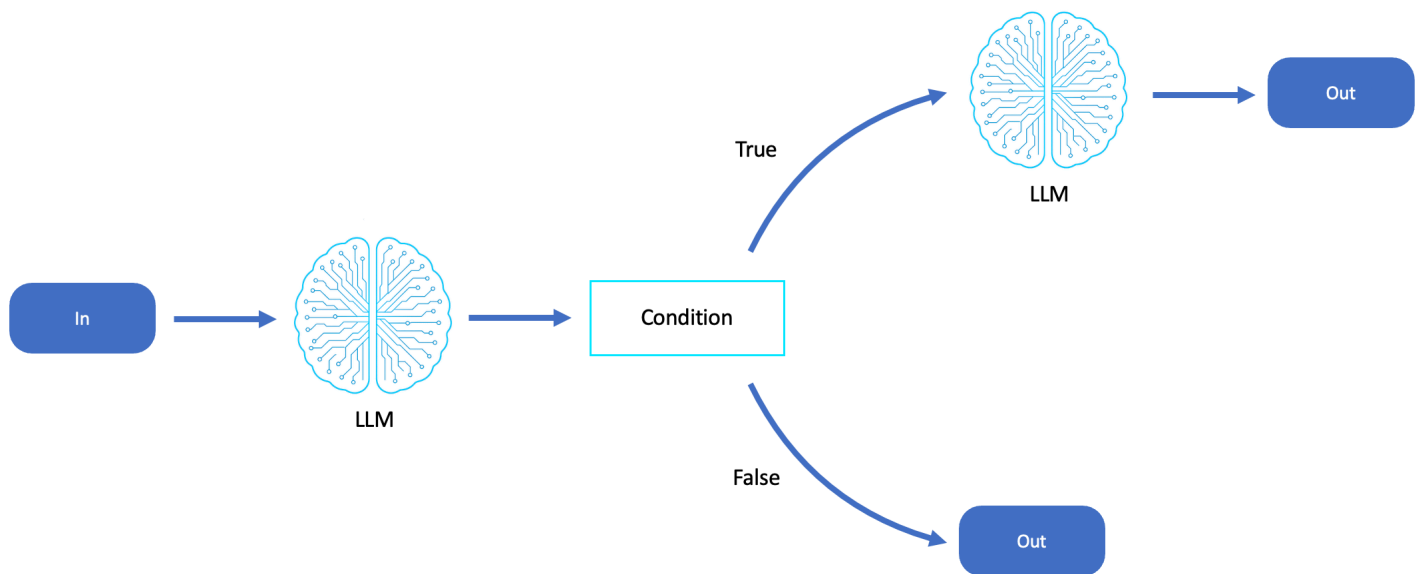
Perspectivas fundamentales

Cada flujo de trabajo de LLM se puede asignar a un patrón de flujo de trabajo de un agente, que refleja y evoluciona los estilos de arquitectura tradicionales basados en eventos. Un elemento básico de los flujos de trabajo de los agentes es la capacidad de aumentar el contexto de un LLM con datos, herramientas y memoria. Esto crea un ciclo de razonamiento informado, adaptable y alineado con la intención del usuario. Mientras que los sistemas tradicionales enriquecen los mensajes con datos de búsqueda, los sistemas de agencia permiten que el software actúe menos como guiones y más como colaboradores inteligentes.

Patrones de saga que se encadenan rápidamente

Al reinventar el encadenamiento rápido de LLM como una saga basada en eventos, descubrimos un nuevo modelo operativo: los flujos de trabajo se distribuyen, se recuperan y se coordinan semánticamente entre agentes autónomos. Cada paso de respuesta rápida se reformula como una tarea atómica, se emite como un evento, lo consume un agente especializado y se enriquece con metadatos contextuales.

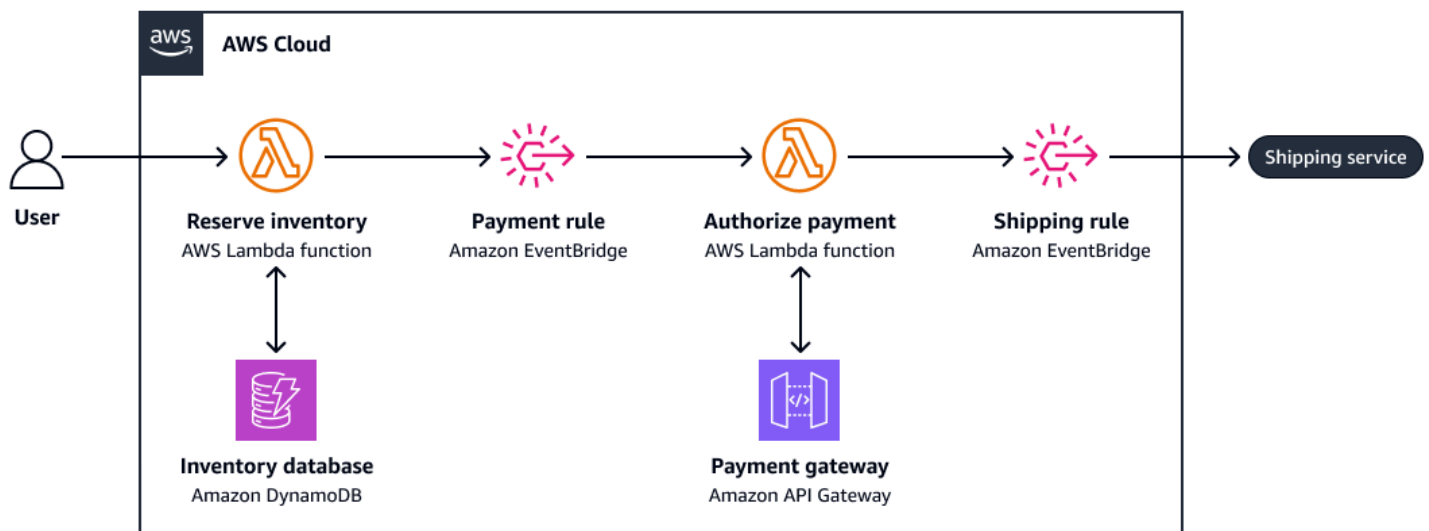
El siguiente diagrama es un ejemplo de encadenamiento de mensajes LLM:



Coreografía de la saga

El patrón coreográfico de la saga es un enfoque de implementación en sistemas distribuidos que no tiene un coordinador central. En su lugar, cada servicio o componente publica eventos que desencadenan la siguiente acción del flujo de trabajo. Este patrón se utiliza ampliamente en los sistemas distribuidos para gestionar las transacciones en varios servicios. En una saga, el sistema ejecuta una serie de transacciones locales coordinadas. Si una falla, el sistema activa acciones compensatorias para mantener la coherencia.

El siguiente diagrama es un ejemplo de coreografía de saga:



1. Inventario de reserva

2. Autoriza el pago
3. Crea una orden de envío

Si el paso 3 no funciona, el sistema invoca acciones compensatorias (por ejemplo, cancelar un pago o liberar el inventario).

Este patrón es especialmente valioso en las arquitecturas basadas en eventos, en las que los servicios están poco acoplados y los estados deben resolverse de forma coherente a lo largo del tiempo, incluso en caso de que se produzca un fallo parcial.

Patrón de encadenamiento rápido

El encadenamiento rápido se parece al patrón de la saga tanto en estructura como en propósito. Ejecuta una serie de pasos de razonamiento que se desarrollan de forma secuencial, preservando el contexto y permitiendo retrocesos y revisiones.

Coreografía de agentes

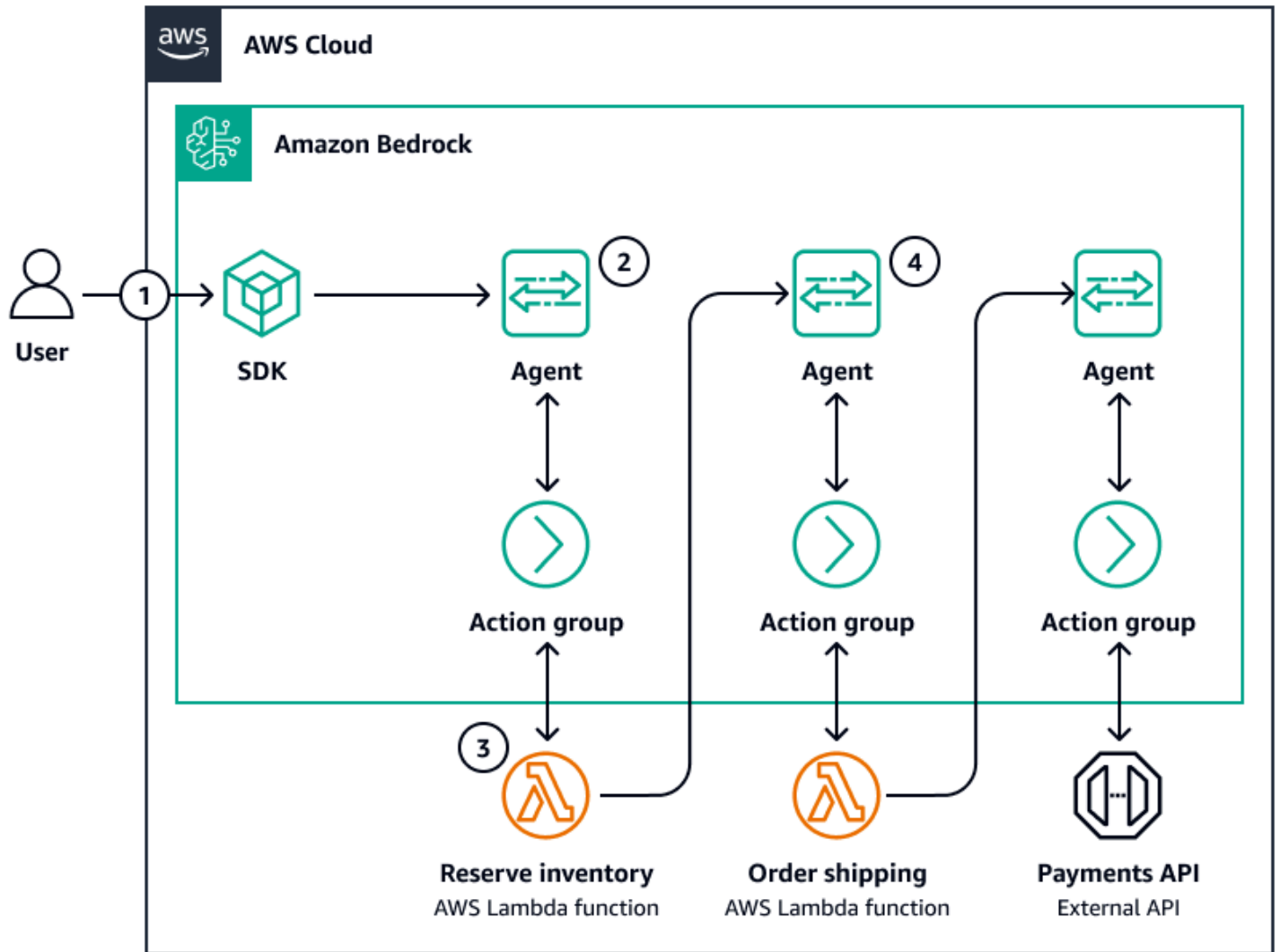
1. LLM interpreta una consulta de usuario compleja y genera una hipótesis
2. LLM elabora un plan para resolver la tarea
3. LLM ejecuta una subtarea (por ejemplo, mediante una llamada a una herramienta o recuperando conocimientos)
4. El LLM refina el resultado o retoma un paso anterior si considera que el resultado no es satisfactorio

Si un resultado intermedio es defectuoso, el sistema puede realizar una de las siguientes acciones:

- Vuelva a intentar los pasos con un enfoque diferente
- Vuelva a la solicitud anterior y vuelva a planificar
- Utilice un bucle de evaluación (por ejemplo, del patrón evaluador-optimizador) para detectar y corregir los errores

Al igual que el patrón de las sagas, el encadenamiento rápido permite avanzar parcialmente y retroceder. Esto sucede mediante el refinamiento y la LLM-directed corrección iterativos en lugar de compensar las transacciones de la base de datos.

El siguiente diagrama es un ejemplo de coreografía de agentes:



1. Un usuario envía una consulta a través de un SDK.
2. Un agente de Amazon Bedrock organiza el razonamiento de la siguiente manera:
 - Interpretación (LLM)
 - Planificación (LLM)
 - Ejecución a través de una herramienta o base de conocimientos
 - Construcción de respuestas
3. Si una herramienta falla o no devuelve datos suficientes, el agente puede replanificar o reformular la tarea de forma dinámica.
4. La memoria (por ejemplo, un almacén vectorial a corto plazo) puede conservar su estado en todos los pasos

Conclusiones

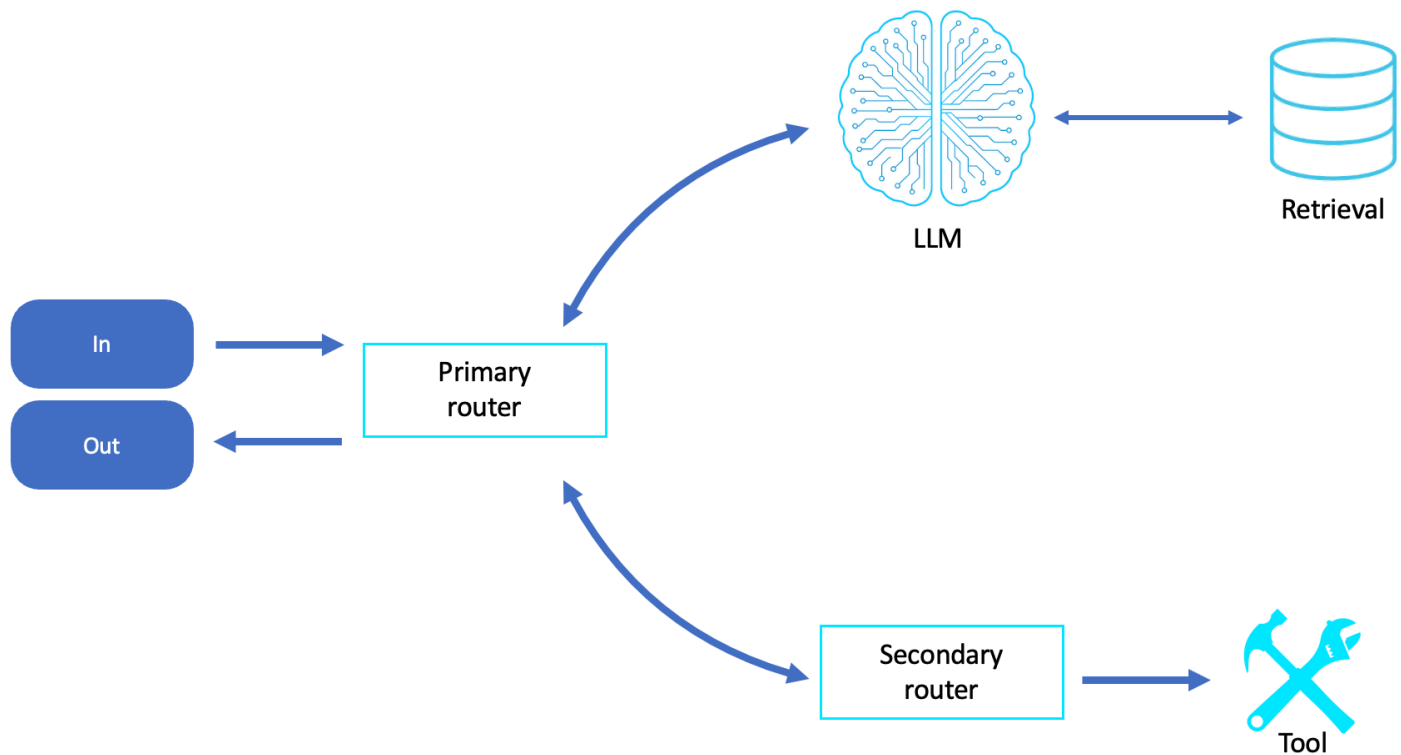
Mientras que el patrón saga gestiona las llamadas de servicio distribuidas con una lógica de compensación, el encadenamiento rápido gestiona las tareas de razonamiento mediante una secuenciación reflexiva y una replanificación adaptativa. Ambos sistemas permiten el progreso gradual, la descentralización de los puntos de decisión y la recuperación de los fallos, y todo ello mediante un razonamiento fundamentado en lugar de una reversión rígida.

El encadenamiento rápido introduce el razonamiento transaccional, que es el equivalente cognitivo de las sagas. Es decir, cada «pensamiento» es reevaluado, revisado o abandonado como parte de un diálogo más amplio dirigido a un objetivo.

Enrutamiento de patrones de despacho dinámicos

En los sistemas de agencia modernos, donde las tareas van desde el análisis de documentos hasta la generación autónoma de software, la capacidad de dirigir dinámicamente las solicitudes al agente o modelo de lenguaje grande (LLM) más competente se vuelve fundamental. La lógica de enrutamiento estática, que suele estar integrada en los scripts de orquestación o en las capas de API, carece de la adaptabilidad necesaria para los entornos en tiempo real, con varios modelos y múltiples capacidades. Para solucionar este problema, los flujos de trabajo de enrutamiento de LLM se pueden transformar en una arquitectura basada en eventos que aproveche un patrón de envío dinámico y convierta las llamadas de LLM en eventos enrutados de forma inteligente y sensibles al contexto.

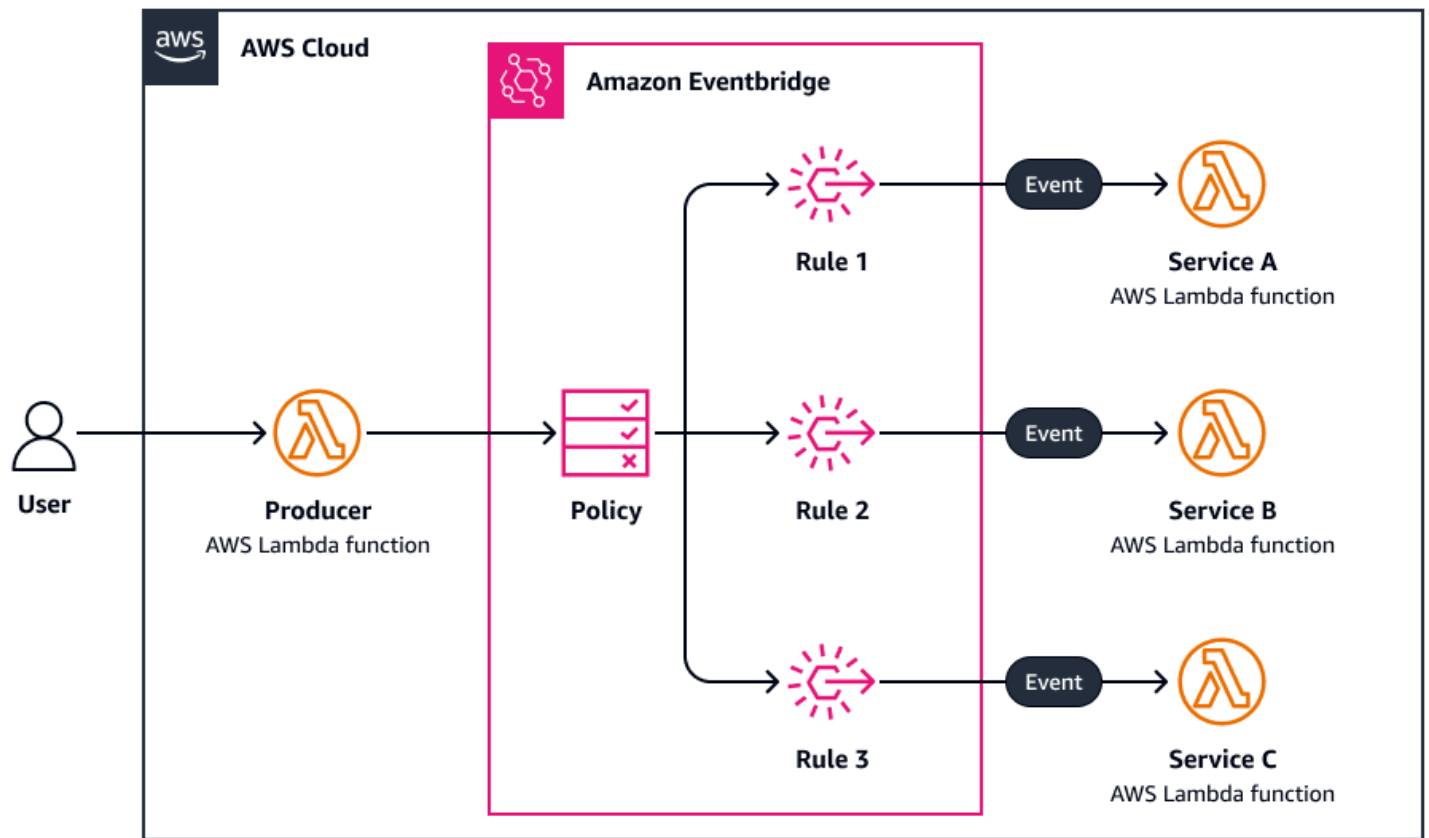
El siguiente diagrama es un ejemplo de enrutamiento LLM:



Envío dinámico

En los sistemas distribuidos tradicionales, el patrón de envío dinámico selecciona e invoca servicios específicos en tiempo de ejecución en función de los atributos del evento entrante, como el tipo de evento, la fuente y la carga útil. Por lo general, esto se implementa con Amazon EventBridge, que puede evaluar y dirigir los eventos entrantes a los objetivos adecuados (por ejemplo AWS Step Functions, AWS Lambda funciones o tareas de Amazon Elastic Container Service).

El siguiente diagrama es un ejemplo de envío dinámico:



1. Una aplicación emite un evento (por ejemplo, `{"type": "orderCreated", "priority": "high"}`).
2. Amazon EventBridge evalúa el evento según sus reglas de enrutamiento.
3. En función de los atributos de un evento, el sistema realiza envíos de forma dinámica a lo siguiente:
 - `HighPriorityOrderProcessor(servicio A)`
 - `StandardOrderProcessor(servicio B)`
 - `UpdateOrderProcessor(servicio C)`

Este patrón admite el acoplamiento flexible, la especialización basada en el dominio y la extensibilidad del tiempo de ejecución. Esto permite que los sistemas respondan de forma inteligente a los cambios en los requisitos y a la semántica de los eventos.

LLM-based enrutamiento

En los sistemas de agencia, el enrutamiento también realiza una delegación dinámica de tareas, pero en lugar de EventBridge las reglas de Amazon o los filtros de metadatos, el LLM clasifica e interpreta

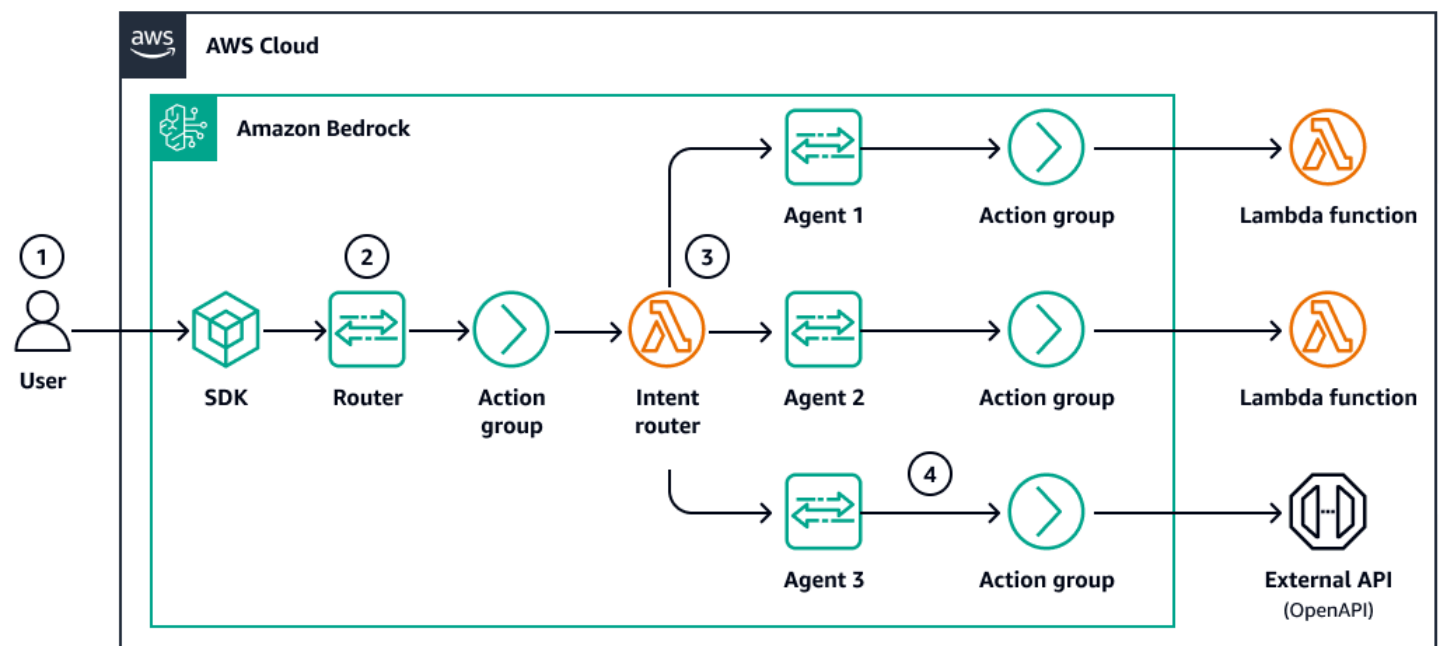
la intención del usuario a través de un lenguaje natural. El resultado es una forma de distribución flexible, semántica y adaptativa.

Agente (router)

Esta arquitectura permite un amplio envío basado en la intención sin esquemas o tipos de eventos predefinidos, lo que resulta ideal para entradas no estructuradas y consultas complejas.

1. Un usuario envía la siguiente solicitud: «¿Me pueden ayudar a revisar las condiciones de mi contrato?»
2. El LLM interpreta esto como una tarea de documentación legal.
3. El agente dirige la tarea a una o más de las siguientes opciones:
 - Plantilla de solicitud de revisión de contratos
 - Subagente de razonamiento legal
 - Herramienta de análisis de documentos

El siguiente diagrama es un ejemplo de un router de agente:



1. Un usuario envía una solicitud en lenguaje natural a través de un SDK.
2. Un agente de Amazon Bedrock utiliza un LLM para clasificar la tarea (por ejemplo, legal, técnica o de programación).

3. El agente dirige la tarea de forma dinámica a través de un grupo de acciones para invocar al agente requerido:
 - Domain-specific agente
 - Cadena de herramientas especializada
 - Configuración rápida personalizada
4. El controlador seleccionado procesa la tarea y devuelve una respuesta personalizada.

Conclusiones

Mientras que el envío dinámico tradicional utiliza EventBridge las reglas de Amazon para el enrutamiento en función de los atributos de los eventos estructurados, el enrutamiento agencial usa LLM para clasificar semánticamente y enrutar las tareas en función del significado y la intención. Esto amplía la flexibilidad del sistema al permitir lo siguiente:

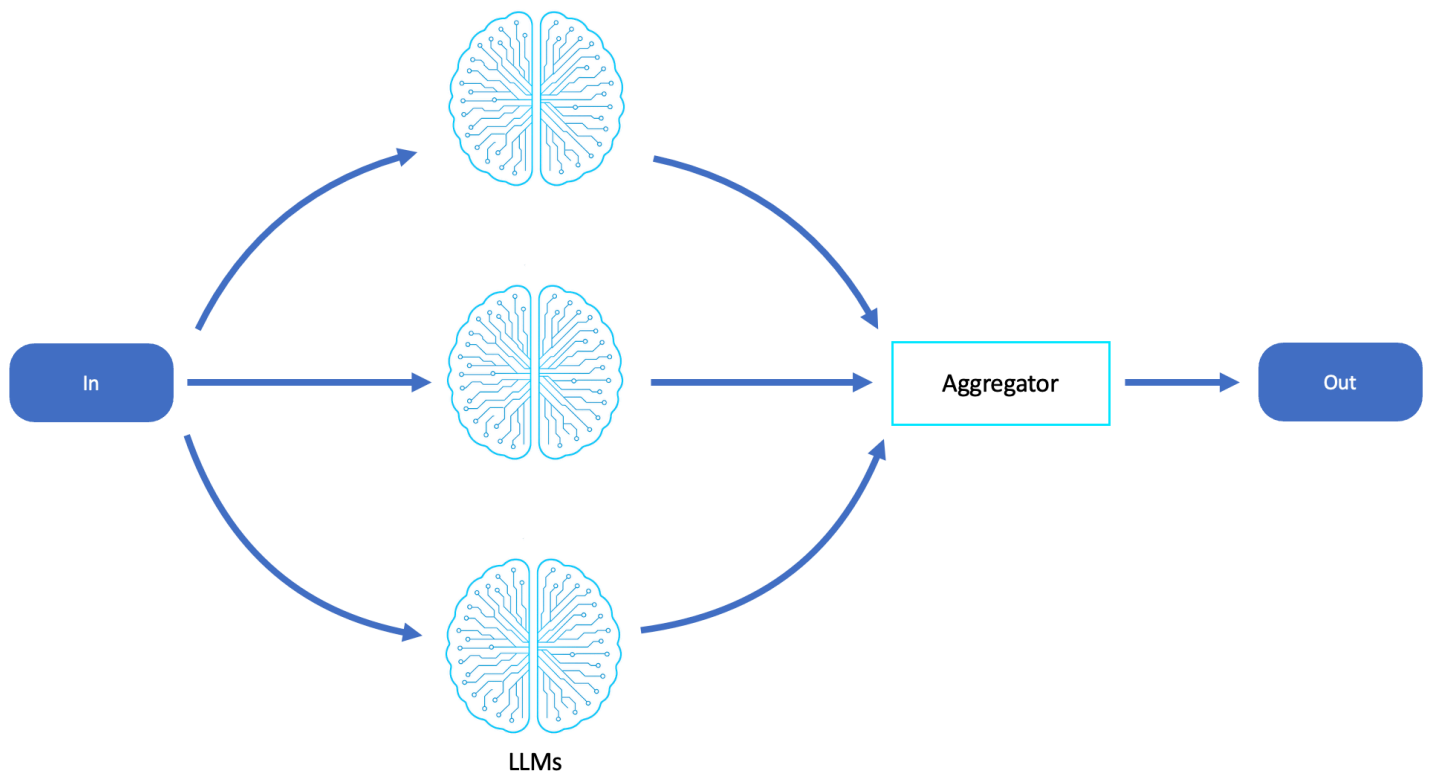
- Comprensión más amplia de las entradas
- Alternativa inteligente y selección de herramientas
- Extensibilidad natural mediante nuevos roles de agente o estilos de programación

El enrutamiento de los agentes reemplaza las reglas rígidas por una distribución cognitiva dinámica, que permite que los sistemas evolucionen con el lenguaje y no con el código.

Patrones de paralelización y dispersión

Muchas tareas avanzadas de razonamiento y generación, como resumir documentos de gran tamaño, evaluar múltiples rutas de solución o comparar diversas perspectivas, se benefician de la ejecución paralela de las solicitudes. Los flujos de trabajo secuenciales tradicionales son insuficientes cuando se requieren escalabilidad, capacidad de respuesta y tolerancia a errores. Para superar esta situación, la LLM-based paralelización puede reinventarse mediante un patrón de dispersión y recopilación basado en eventos, en el que las tareas se distribuyen dinámicamente entre agentes autónomos y los resultados se sintetizan de forma inteligente.

El siguiente diagrama es un ejemplo de un flujo de trabajo de paralelización de LLM:



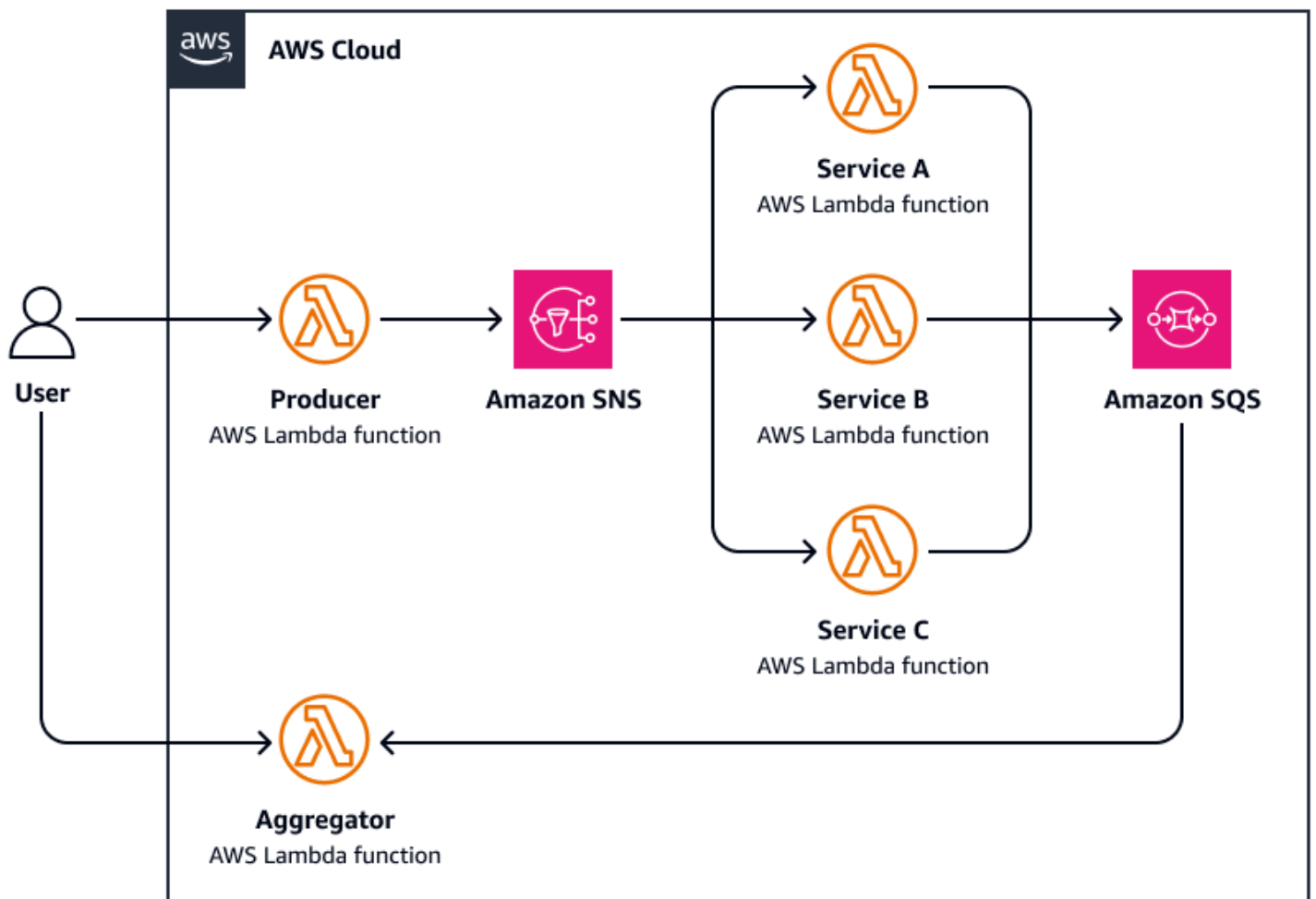
Scatter-gather

En los sistemas distribuidos, un patrón de dispersión y recopilación envía las tareas a varios servicios o unidades de procesamiento en paralelo, espera sus respuestas y, a continuación, agrega los resultados en una salida consolidada. A diferencia de la dispersión, la dispersión y recolección se coordina porque espera respuestas y, por lo general, aplica la lógica para combinar, comparar y seleccionar los resultados.

Entre las implementaciones habituales de la paralelización y la recopilación de dispersión se incluyen las siguientes:

- AWS Step Functions mapear un estado para la ejecución de tareas en paralelo
- AWS Lambda con simultaneidad, coordinando los resultados de múltiples funciones invocadas
- Amazon EventBridge con identificadores de correlación y flujos de trabajo de agregación
- Patrón de controlador personalizado para gestionar la distribución y recopilar resultados mediante Amazon Simple Storage Service (Amazon S3), Amazon DynamoDB o colas

El siguiente diagrama es un ejemplo de scatter-gather:



1. Un usuario envía una solicitud a una función de coordinación central que dispersa la tarea publicando mensajes paralelos en un tema del Amazon Simple Notification Service (Amazon SNS).
2. Cada mensaje incluye metadatos de la tarea y se envía a un trabajador especializado. AWS Lambda
3. Cada trabajador procesa de AWS Lambda forma independiente la subtarea que se le ha asignado (por ejemplo, consultar una API externa, procesar un documento y analizar los datos).
4. Los resultados se escriben en una capa de almacenamiento común, como Amazon Simple Queue Service (Amazon SQS).
5. La función de agregación espera a que se completen todas las respuestas y, a continuación, hace lo siguiente:
 - Recopila y agrega los resultados (por ejemplo, combina resúmenes y selecciona las mejores coincidencias)

- Envía una respuesta final o desencadena un flujo de trabajo posterior

Los casos de uso más comunes de los patrones de dispersión y recolección incluyen los siguientes:

- Búsqueda federada
- Motores de comparación de precios
- Análisis de datos agregados
- Inferencia multimodelo

LLM-based paralelización (cognición dispersa-recolectora)

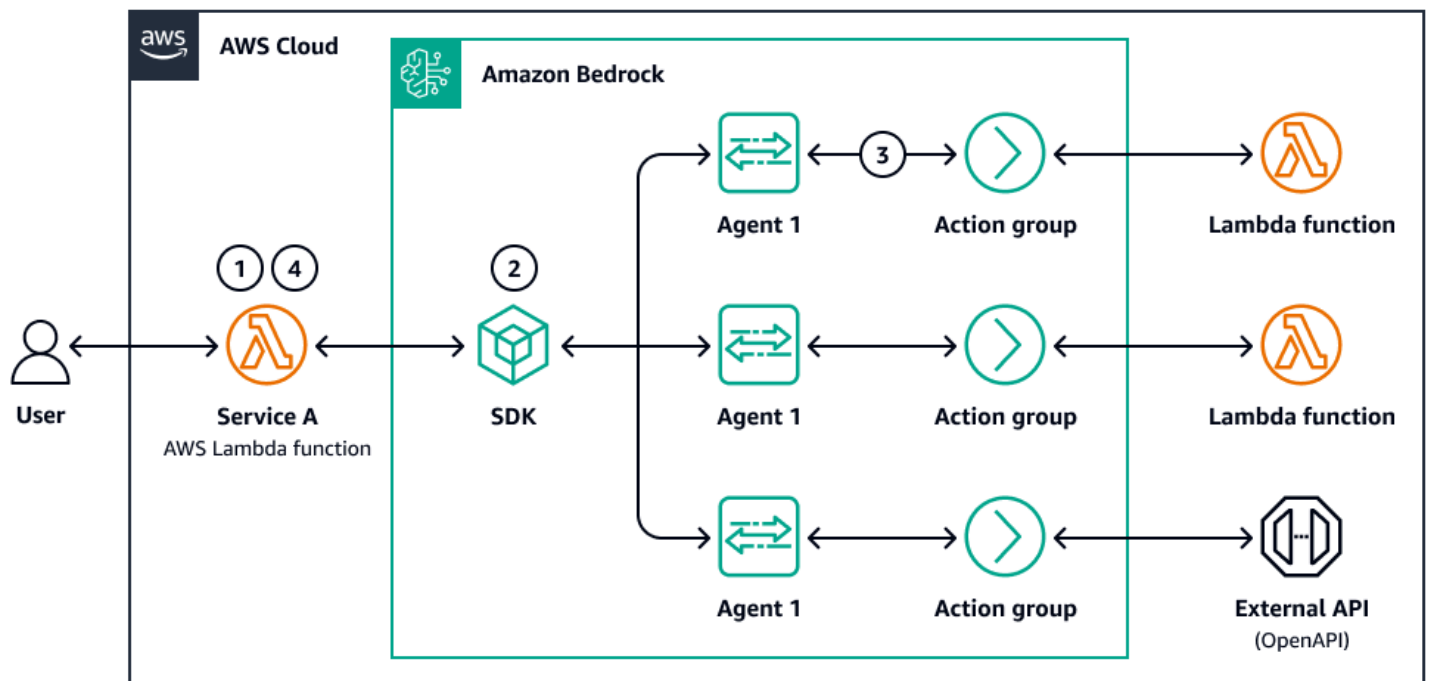
En los sistemas agénticos, la paralelización refleja de cerca la dispersión y la recopilación mediante la distribución de las subtarefas entre varios agentes o llamadas de LLM, cada uno de los cuales analiza de forma independiente una parte del problema. Los resultados devueltos se recopilan y sintetizan mediante un proceso de agregación, que suele ser otro LLM o agente controlador.

Paralelización de agentes

1. Un agente envía una solicitud para «Resumir la información recopilada en estos 10 informes».
2. Distribuye los informes en 10 tareas de resumen LLM paralelas.
3. Cuando devuelve todos los resúmenes, el agente hace lo siguiente:
 - Reúne los resúmenes en un resumen unificado
 - Identifica temas o contradicciones
 - Envía la salida sintetizada al usuario

Este flujo de trabajo de agencia permite un razonamiento paralelo escalable, modular y adaptativo. Esto es ideal para casos de uso que requieren un alto rendimiento cognitivo.

El siguiente diagrama es un ejemplo de paralelización de agentes:



1. Un usuario envía una consulta o un conjunto de documentos de varias partes.
2. Un controlador AWS Lambda o una función escalonada distribuye las subtareas. Cada tarea invoca una llamada o un subagente de Amazon Bedrock LLM con su propio mensaje.
3. Cuando se completan las llamadas y las subtareas, los resultados se almacenan (por ejemplo, en Amazon S3 o en un almacén de memoria) y un paso de agregación fusiona, compara o filtra los resultados.
4. El sistema devuelve la respuesta final al usuario o al agente intermedio.

Este sistema tiene un circuito de razonamiento distribuido con trazabilidad, tolerancia a errores y lógica opcional de ponderación o selección de resultados.

Conclusiones

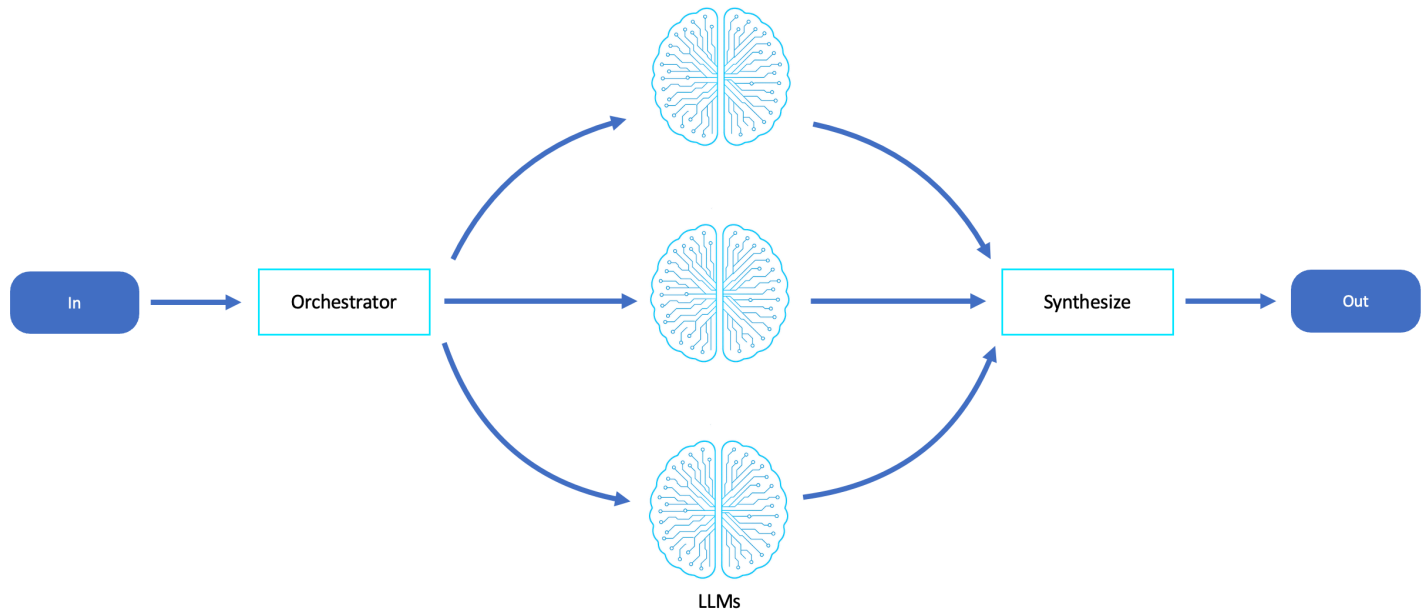
La paralelización de agentes utiliza patrones de dispersión y recolección para distribuir las tareas de LLM, lo que permite el procesamiento paralelo y la síntesis inteligente de resultados.

Patrones de orquestación de Saga

A medida que los flujos de trabajo impulsados por los LLM se vuelven cada vez más complejos y abarcan cadenas de mensajes, pasos de procesamiento de datos, invocaciones de herramientas

y colaboración entre agentes, la necesidad de una orquestación inteligente se vuelve esencial. En lugar de basarse en scripts estrechamente acoplados o en flujos de ejecución estáticos predeterminados, estos flujos de trabajo se pueden implementar como patrones de orquestación basados en eventos, lo que permite a los LLM-based sistemas coordinar, monitorear y adaptar de forma dinámica las tareas de varios pasos entre agentes autónomos.

El siguiente diagrama es un ejemplo de un orquestador:



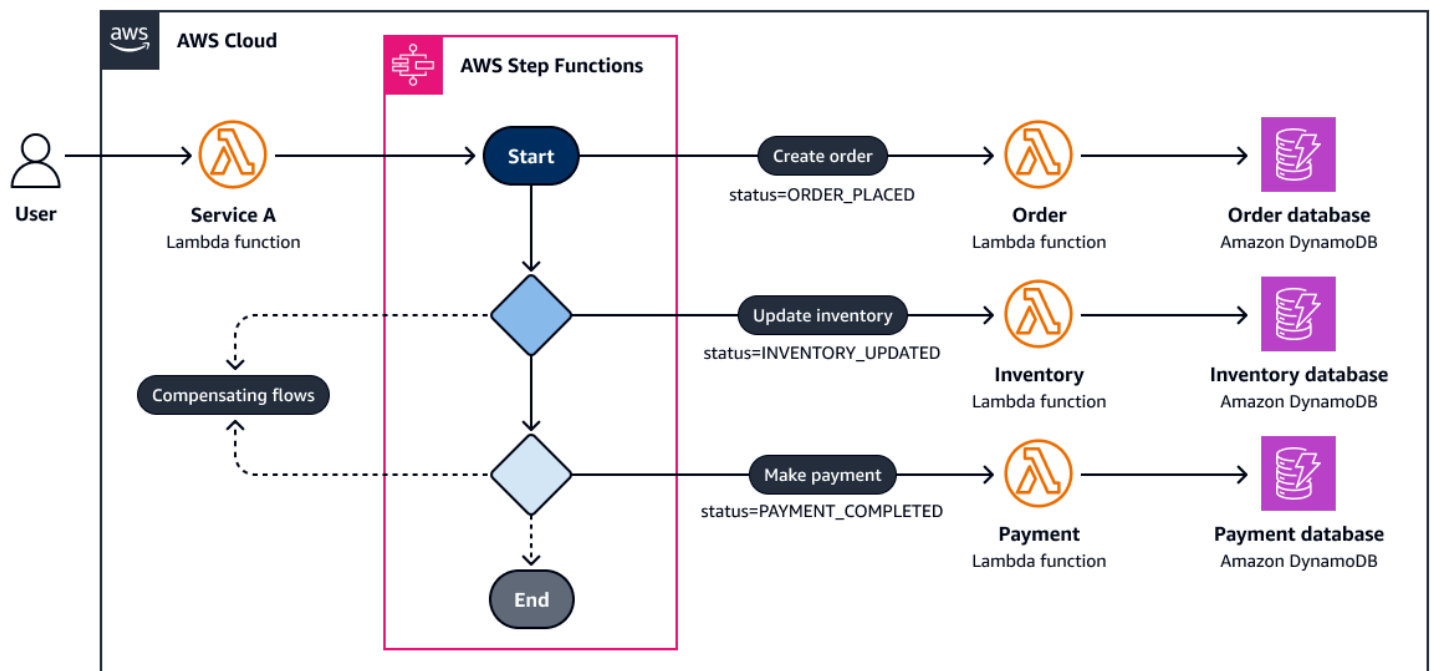
Orquestación de eventos

En los sistemas distribuidos tradicionales, la orquestación de eventos se refiere a un patrón en el que un coordinador central gestiona un flujo de trabajo complejo al dirigir explícitamente el flujo de control entre varios servicios o tareas. A diferencia de la coreografía de eventos (en la que cada servicio reacciona de forma independiente), la orquestación proporciona una lógica, visibilidad y control centralizados sobre todo el proceso.

Por lo general, esto se implementa con las siguientes herramientas:

- AWS Step Functions— Defina y ejecute flujos de trabajo con estado
- AWS Lambda— Realice tareas discretas dentro del flujo orquestado
- Amazon SQS o Amazon EventBridge: activa pasos o respuestas asíncronos

El siguiente diagrama es un ejemplo de la orquestación de una saga:



Un AWS Step Functions flujo de trabajo gestiona el proceso de pedido de un cliente:

1. Crear pedido (AWS Lambda)
2. Actualizar inventario (AWS Lambda)
3. Realizar el pago (AWS Lambda)

El orquestador coordina cada paso gestionando los reintentos, las ramificaciones paralelas, los tiempos de espera y los errores.

Role-based sistema de agentes (orquestador)

En los sistemas agénticos, el patrón del orquestador refleja la orquestación de eventos, pero distribuye la lógica entre varios agentes de razonamiento, cada uno con una función o especialización definida. Un agente orquestador central interpreta la tarea general, la descompone en subtareas y las delega en agentes trabajadores, cada una optimizada para un dominio concreto (por ejemplo, investigación, codificación, resumen o revisión).

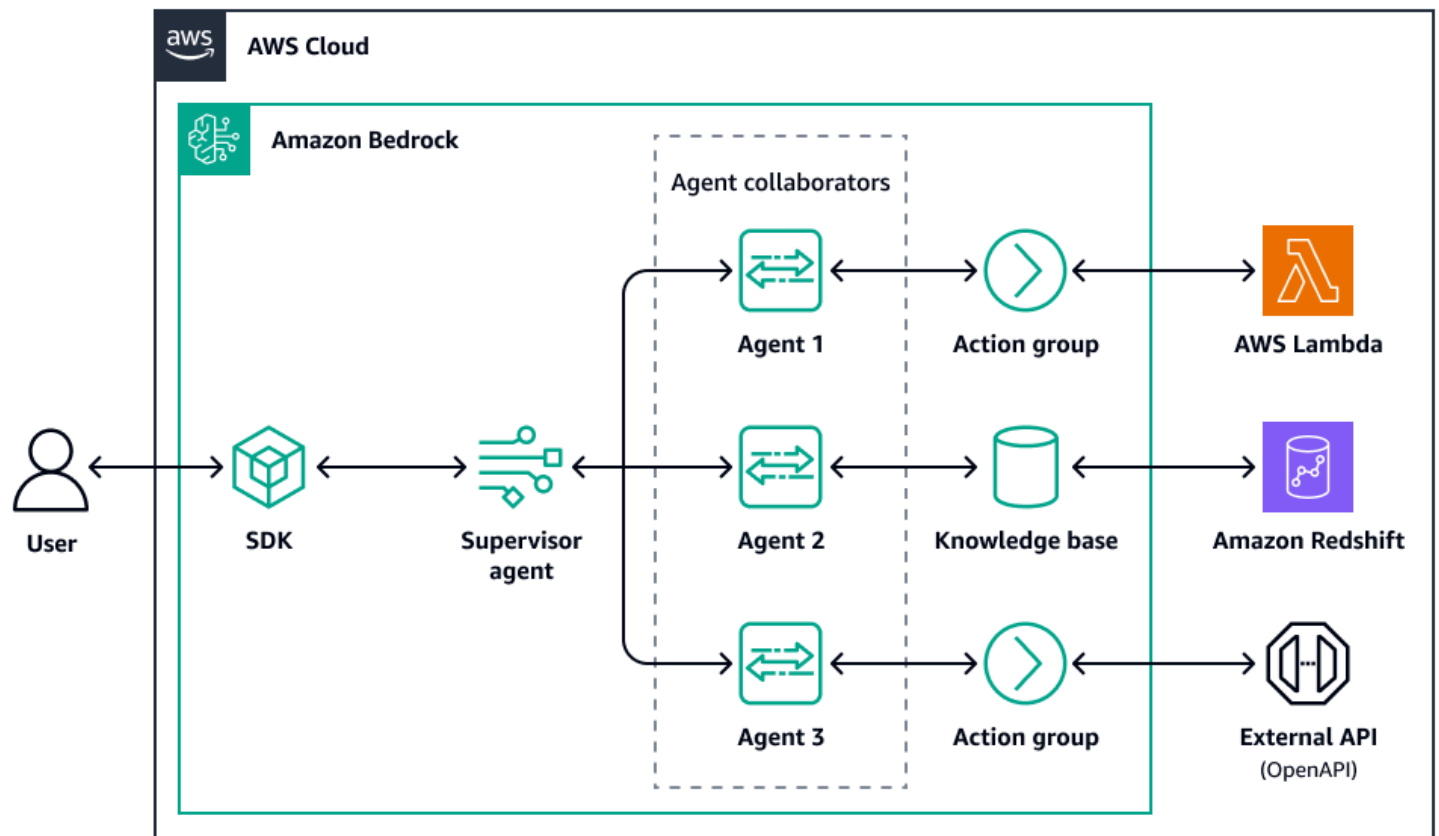
Supervisor

1. Un usuario envía la siguiente pregunta: «Cree un resumen del proyecto y resuma los cinco principales competidores».
2. El agente orquestador hace lo siguiente:

- Asigna un agente de investigación para buscar datos de la competencia
- Envía los resultados sin procesar a un agente de resumen
- Pasa los resultados a un agente redactor de informes
- Compila el resultado final para el usuario

Cada agente funciona de forma independiente, pero el orquestador coordina las tareas. Es como una función Lambda que gestiona las tareas del flujo de trabajo.

El siguiente diagrama es un ejemplo de un supervisor:



1. Un usuario envía una tarea a un agente supervisor de Amazon Bedrock.
2. El agente supervisor analiza la solicitud en subtareas para cada agente colaborador.
3. Cada subtarea se asigna a un agente colaborador con instrucciones o cadenas de herramientas específicas para cada función.
4. Los agentes de trabajo llaman a las API o herramientas externas a través de un grupo de acción.
5. Cada agente de trabajo devuelve el resultado en un formato estructurado.

6. Cuando todos los trabajadores devuelven sus resultados, el supervisor evalúa, sintetiza y devuelve la respuesta final.

Esta estructura permite la modularidad, la adaptabilidad y la introspección en flujos de trabajo complejos de agentes de varios pasos.

Conclusiones

Mientras que la orquestación de eventos utiliza un control centralizado (por ejemplo AWS Step Functions) para dirigir la ejecución del servicio, los sistemas de agentes basados en roles utilizan un agente LLM-powered orquestador para razonar sobre el objetivo, delegar subtareas a los agentes trabajadores y sintetizar el resultado final.

En ambos paradigmas, el orquestador hace lo siguiente:

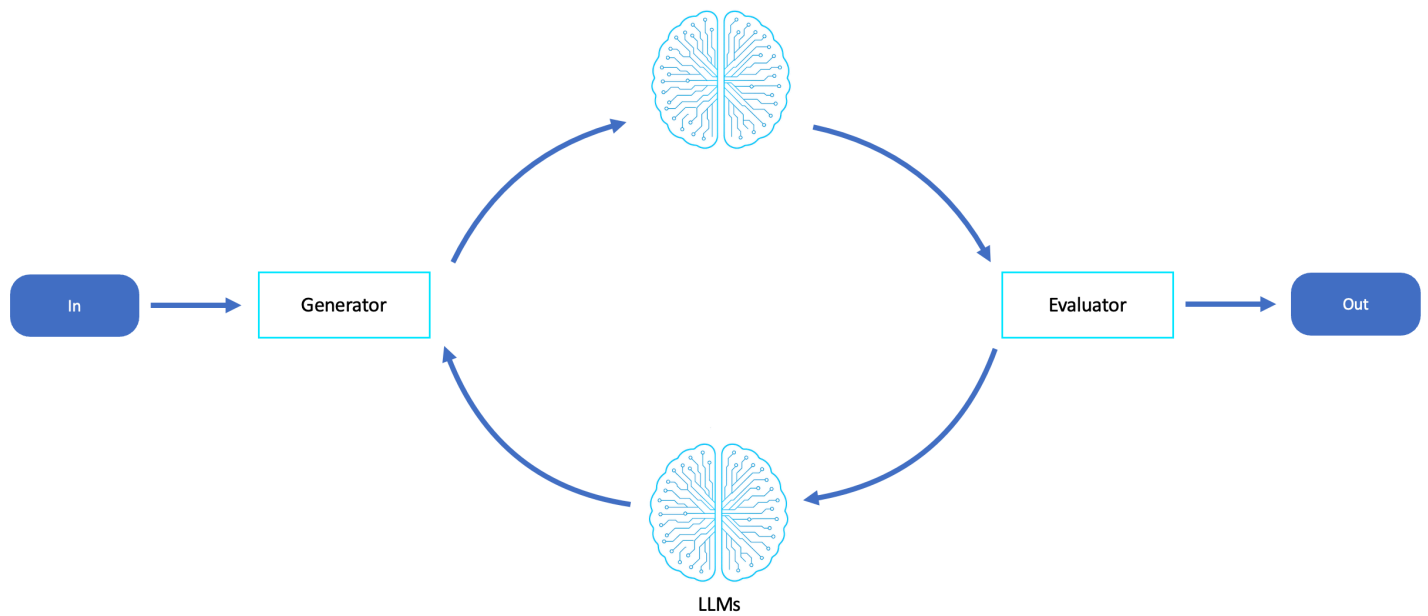
- Mantiene el contexto y el flujo de ejecución
- Gestiona la ramificación, la secuenciación y la gestión de errores
- Produce un resultado unificado a partir de componentes distribuidos

Sin embargo, la orquestación agencial añade razonamiento, adaptabilidad y delegación semántica. Esto lo hace ideal para tareas abiertas, ambiguas y en evolución.

El evaluador refleja y refina los patrones de bucles

Las tareas como la generación de código, el resumen o la toma de decisiones autónoma se benefician en gran medida de la retroalimentación sobre el tiempo de ejecución, lo que permite que el sistema evolucione mediante la observación y el perfeccionamiento. Para ponerlo en práctica, el ciclo de reflexión y refinamiento se puede implementar como un circuito de control de retroalimentación basado en eventos, un patrón inspirado en la ingeniería de sistemas y adaptado a flujos de trabajo autónomos e inteligentes.

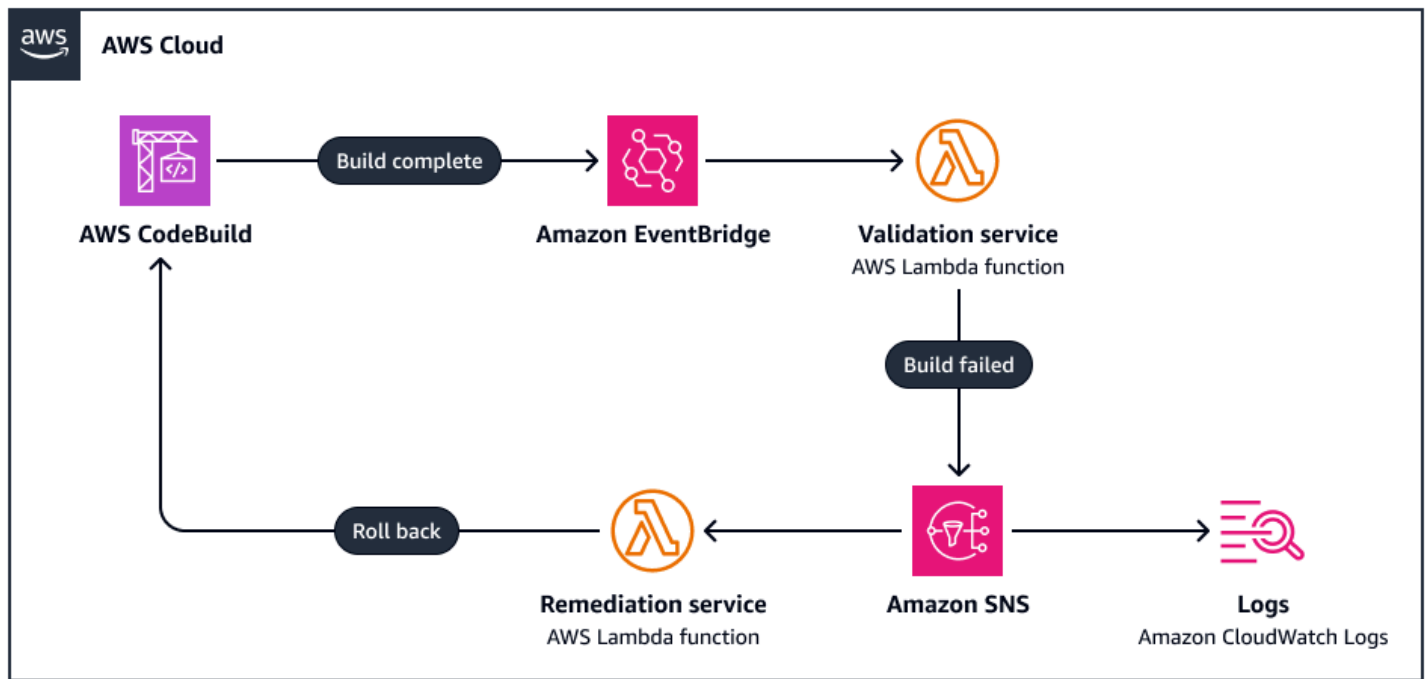
El siguiente diagrama es un ejemplo de un circuito de retroalimentación entre reflexión y refinamiento del evaluador:



Bucle de control de retroalimentación

Un circuito de control de retroalimentación es un patrón que monitorea sus propios resultados y comportamientos, los evalúa en función de criterios definidos o de un estado deseado y, a continuación, ajusta sus acciones en consecuencia. Esta arquitectura se inspira en la teoría del control y es fundamental en ámbitos como la automatización, los procesos de integración y entrega continuas (...CI/CD) y las operaciones de aprendizaje automático.

El siguiente diagrama es un ejemplo de un circuito de control de retroalimentación:



1. Una canalización de despliegue emite un evento BuildComplete.
2. El evento desencadena un trabajo de prueba o evaluación automatizado que valida la compilación.
3. Si la validación falla (por ejemplo, debido a pruebas fallidas, problemas de seguridad o una infracción de una política), el sistema:
 - Emite un evento BuildComplete
 - Registra el problema o envía una notificación
 - Activa una acción correctiva o correctiva, como revertirla, parchearla o volver a intentarlo

El ciclo continúa hasta que produce un resultado o una escalada aceptables, o hasta que se agota el tiempo de espera. Este patrón se suele utilizar para lo siguiente:

- EventBridge Reglas de Amazon para dirigir los eventos a tareas de evaluación o corrección
- AWS Step Functions para la lógica de reintentos iterativos y la ramificación de los resultados de la evaluación
- Amazon Simple Notification Service (Amazon SNS) o alarmas de CloudWatch Amazon para activar comentarios y alertas
- AWS Lambda funciones o trabajadores agrupados en contenedores para aplicar medidas correctivas

Bucle de control de retroalimentación (evaluador)

El flujo de trabajo de un evaluador es un circuito de retroalimentación cognitiva impulsado por los LLM o agentes de razonamiento. El proceso consiste en lo siguiente:

1. Un agente generador o LLM produce un resultado (por ejemplo, un plan, una respuesta o un borrador).
2. Un agente evaluador revisa el resultado utilizando una guía de crítica o una rúbrica de evaluación.
3. En función de los comentarios, el agente original o un nuevo agente optimizador revisan el resultado.

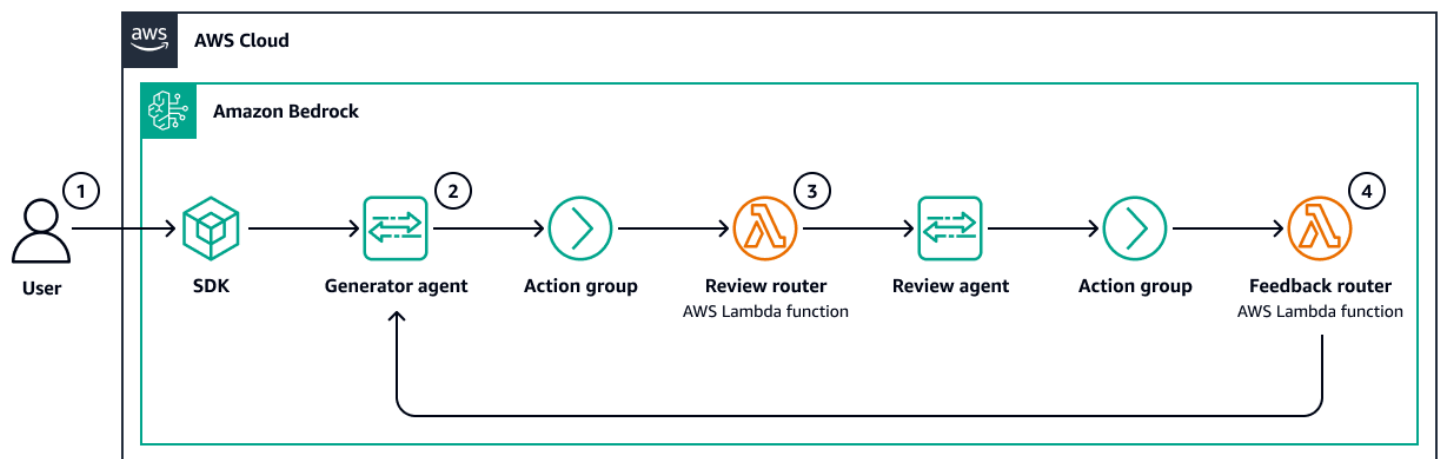
El ciclo se repite hasta que el resultado cumpla una serie de criterios, se apruebe o alcance un límite de reintentos.

Evaluador

1. Un usuario le pide a un agente que redacte un resumen de la política.
2. El agente generador lo redacta.
3. Un agente evaluador comprueba la cobertura, el tono y la corrección legal.
4. Si la respuesta es inadecuada, se refina y se vuelve a enviar hasta que converja el ciclo de retroalimentación.

Esto permite la autoevaluación, el refinamiento iterativo y el control adaptativo de los resultados, todo ello sin intervención humana.

El siguiente diagrama es un ejemplo de un circuito de control de retroalimentación (evaluador):



1. Un usuario emite una tarea (por ejemplo, redactar una estrategia empresarial).
2. Un agente de Amazon Bedrock genera un borrador inicial mediante un LLM.
3. Un segundo agente (o un agente de seguimiento) realiza una evaluación estructurada (por ejemplo, «califica este resultado según su claridad, integridad y tono»).
4. Si la calificación cae por debajo de un umbral, la respuesta se revisa de la siguiente manera:
 - Reinvocar el generador con una crítica incrustada
 - Enviar los comentarios a un agente refinador especializado
 - Iterando hasta alcanzar una respuesta aceptable

Los componentes opcionales, como AWS Lambda los controladores, AWS Step Functions pueden gestionar los umbrales de retroalimentación, los reintentos y las estrategias alternativas.

Conclusiones

Mientras que los bucles de control de retroalimentación tradicionales utilizan eventos, métricas y lógica de remediación para validar y ajustar el comportamiento del sistema, los bucles de evaluación de los agentes utilizan agentes de razonamiento para evaluar, reflejar y revisar los resultados de forma dinámica.

En ambos paradigmas:

- La salida se evalúa después de generarse
- Las acciones correctivas o de refinamiento se activan en función de los comentarios
- El sistema se adapta continuamente a un objetivo, calidad u objetivo

La versión agencial transforma la validación estática en reflexión semántica, lo que permite a los agentes que se mejoran a sí mismos evaluar su propia eficacia.

Diseñar los flujos de trabajo de los agentes en AWS

Cada patrón de esta guía se puede crear utilizando. Servicios de AWS Los agentes de Amazon Bedrock proporcionan canales de organización, acceso a datos e interacción.

Componente	Servicio de AWS	Finalidad
------------	-----------------	-----------

Razonamiento LLM	Amazon Bedrock	Lógica del agente, planificación, uso de herramientas
Ejecución de herramientas	AWS Lambda, Amazon ECS, Amazon SageMaker	Aloje herramientas externas para los agentes
Memoria y RAG	Base de conocimiento de Amazon Bedrock, Amazon S3, OpenSearch	Memoria persistente y semántica
Orquestación	AWS Step Functions	Coordinación de tareas y agentes en varios pasos
Enrutamiento de eventos	Amazon EventBridge, Amazon SQS	Mensajería interagente desacoplada
Interfaz de usuario	Amazon API Gateway AWS AppSync, SDK	Puntos de entrada para aplicaciones o sistemas
Supervisión	Amazon CloudWatch AWS X-Ray, AWS Distro para OpenTelemetry	Observabilidad e introspección de los agentes

Conclusión

Los patrones de flujo de trabajo de los agentes son la siguiente etapa evolutiva de las arquitecturas basadas en eventos, en las que la lógica empresarial no se define de forma estática, sino que se razona de forma dinámica mediante el uso de una cognición mejorada mediante un modelo de lenguaje amplio (LLM). Al combinar las primitivas tradicionales nativas de la nube con los flujos de trabajo de la LLM y los patrones de diseño de los agentes, las organizaciones pueden crear sistemas adaptables, inteligentes y modulares que respondan con un propósito y aprendan de la experiencia.

En estos patrones, Amazon Bedrock es la puerta de entrada a la cognición de los agentes, ya que permite a LLM-based los agentes acceder a los flujos de trabajo de los eventos, interactuar con las herramientas y la memoria y ofrecer resultados estructurados, rastreables y alineados.

Al diseñar e implementar sistemas de agentes, estos patrones de flujo de trabajo proporcionan modelos para crear arquitecturas de IA autónomas y componibles. Estos sistemas se basan en las mejores prácticas sin servidor y se complementan con modelos básicos inteligentes.

Historial de documentos

En la siguiente tabla, se describen cambios significativos de esta guía. Si quiere recibir notificaciones de futuras actualizaciones, puede suscribirse a las [notificaciones RSS](#).

Cambio	Descripción	Fecha
Publicación inicial	—	14 de julio de 2025

Las traducciones son generadas a través de traducción automática. En caso de conflicto entre la traducción y la version original de inglés, prevalecerá la version en inglés.