



Pola dan alur kerja Agentic AI di AWS

AWS Bimbingan Preskriptif



AWS Bimbingan Preskriptif: Pola dan alur kerja Agentic AI di AWS

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

Merek dagang dan tampilan dagang Amazon tidak boleh digunakan sehubungan dengan produk atau layanan apa pun yang bukan milik Amazon, dengan cara apa pun yang dapat menyebabkan kebingungan di antara pelanggan, atau dengan cara apa pun yang merendahkan atau mendiskreditkan Amazon. Semua merek dagang lain yang tidak dimiliki oleh Amazon merupakan hak milik masing-masing pemiliknya, yang mungkin atau mungkin tidak terafiliasi, terkait dengan, atau disponsori oleh Amazon.

Table of Contents

Pengantar	1
Audiens yang dituju	1
Tujuan	1
Tentang seri konten ini	2
Pola agen	3
Agen penalaran dasar	4
Arsitektur	4
Deskripsi	5
Kemampuan	6
Batasan	6
Kasus penggunaan umum	6
Panduan implementasi	7
Ringkasan	7
Arsitektur	7
Deskripsi	8
Kemampuan	9
Kasus penggunaan umum	9
Panduan implementasi	9
Ringkasan	10
Tool-based agen untuk memanggil fungsi	10
Arsitektur	10
Deskripsi	11
Kemampuan	12
Kasus penggunaan umum	12
Panduan implementasi	12
Ringkasan	13
Tool-based agen untuk server	13
Arsitektur	13
Deskripsi	14
Kemampuan	15
Kasus penggunaan umum	15
Panduan implementasi	15
Ringkasan	16
Computer-use agen	16

Arsitektur	16
Deskripsi	17
Kemampuan	18
Kasus penggunaan umum	18
Panduan implementasi	19
Ringkasan	19
Agen pengkodean	19
Arsitektur	19
Deskripsi	20
Kemampuan	21
Kasus penggunaan umum	21
Panduan implementasi	22
Ringkasan	22
Agen pidato dan suara	22
Arsitektur	22
Deskripsi	23
Kemampuan	24
Kasus penggunaan umum	24
Panduan implementasi	24
Ringkasan	25
Agen orkestrasi alur kerja	25
Arsitektur	25
Deskripsi	26
Kemampuan	27
Kasus penggunaan umum	27
Panduan implementasi	27
Ringkasan	28
Memory-augmented agen	28
Arsitektur	28
Deskripsi	29
Kemampuan	30
Kasus penggunaan umum	30
Menerapkan agen yang diperbesar memori	30
Menerapkan dorongan yang disuntikkan memori	31
Ringkasan	31
Agen simulasi dan test-bed	32

Arsitektur	32
Deskripsi	33
Kemampuan	34
Kasus penggunaan umum	34
Panduan implementasi	35
Ringkasan	35
Pengamat dan agen pemantauan	36
Arsitektur	36
Deskripsi	36
Kemampuan	37
Kasus penggunaan umum	37
Panduan implementasi	38
Ringkasan	38
Multi-agent kolaborasi	39
Deskripsi	40
Kemampuan	41
Kasus penggunaan umum	41
Panduan implementasi	41
Ringkasan	42
Kesimpulan	42
Takeaways	43
Alur kerja LLM	44
Ikhtisar kognisi yang diperkuat LLM	44
Alur kerja untuk rantai cepat	45
Deskripsi	46
Kemampuan	46
Kasus penggunaan umum	47
Alur kerja untuk perutean	47
Kemampuan	48
Kasus penggunaan umum	48
Alur kerja untuk paralelisasi	48
Kemampuan	50
Kasus penggunaan umum	50
Alur kerja untuk orkestrasi	50
Kemampuan	51
Kasus penggunaan umum	51

Alur kerja untuk evaluator dan loop refleksi-refine	52
Kasus penggunaan umum	53
Kemampuan	53
Kesimpulan	53
Pola alur kerja agen	55
Dari sistem yang didorong oleh peristiwa hingga sistem yang ditambah kognisi	55
Arsitektur berbasis peristiwa	55
Alur kerja yang ditambah kognisi	56
Wawasan inti	58
Pola saga rantai yang cepat	58
Koreografi Saga	59
Pola rantai yang cepat	60
Koreografi agen	60
Takeaways	62
Merutekan pola pengiriman dinamis	62
Pengiriman dinamis	63
Perutean berbasis LLM	64
Router agen	65
Takeaways	66
Paralelisasi dan pola scatter-gathering	66
Scatter-kumpulkan	67
Paralelisasi berbasis LLM (kognisi scatter-gathering)	69
Paralelisasi agen	69
Takeaways	70
Pola orkestrasi Saga	70
Orkestrasi acara	71
Sistem agen berbasis peran (orkestrator)	72
Pengawas	72
Takeaways	74
Evaluator memantulkan pola loop	74
Lingkaran kontrol umpan balik	75
Lingkaran kontrol umpan balik (evaluator)	77
Penilai	77
Takeaways	78
Merancang alur kerja agen pada AWS	78
Kesimpulan	79

Riwayat dokumen	80
Glosarium	81
#	81
A	82
B	85
C	87
D	90
E	94
F	96
G	98
H	99
I	101
L	103
M	105
O	109
P	112
Q	115
R	115
D	118
T	122
U	123
V	124
W	124
Z	125
.....	cxxvii

Pola dan alur kerja AI agen di AWS

Aaron Sempf dan Andrew Hooker, Amazon Web Services

Juli 2025 ([sejarah dokumen](#))

Organizations mengadopsi model bahasa besar (LLMs) dan agen perangkat lunak untuk memecahkan masalah multidomain dinamis menggunakan disiplin arsitektur baru yang disebut pola agen. Pola agen adalah cetak biru dasar dan konstruksi modular yang digunakan untuk merancang dan mengatur agen AI yang berorientasi pada tujuan di banyak konteks.

Audiens yang dituju

Panduan ini ditujukan untuk arsitek, pengembang, dan pemimpin produk yang ingin membangun aplikasi cerdas yang melampaui logika statis, logika simbolik, dan otomatisasi deterministik.

Tujuan

Panduan ini menyediakan kerangka desain dan pendekatan implementasi untuk sistem agen AI yang beroperasi secara mandiri sambil tetap dapat dikontrol dan selaras dengan tujuan Anda. Ini menghubungkan pola arsitektur berbasis peristiwa dengan berbagai alternatif agen, menunjukkan cara membangun sistem agen tingkat produksi menggunakan arsitektur cloud-native. Mata pelajaran berikut dibahas dalam panduan ini:

- Pola agen — Pola agen adalah templat desain yang dapat digunakan kembali yang menggambarkan struktur dan perilaku agen individu. Ini termasuk agen penalaran, agen tambahan pengambilan, agen pengkodean, antarmuka suara, orkestra alur kerja, dan sistem multi-agen kolaboratif. Setiap pola menggambarkan bagaimana agen memandang, bernalar, bertindak, dan belajar, dipetakan. Layanan AWS
- Alur kerja LLM - Alur kerja fokus pada bagaimana agen menggunakan LLMs untuk penalaran. Mereka mengeksplorasi strategi yang mendorong dan mekanisme perencanaan, dan menguraikan bagaimana LLMs digunakan tidak hanya untuk menghasilkan teks tetapi juga untuk mendorong perilaku yang terstruktur, dapat ditafsirkan, dan dapat diandalkan dalam lingkaran agen.
- Pola alur kerja agen — Pola alur kerja menggambarkan bagaimana beberapa agen, alat, dan lingkungan berinteraksi untuk membentuk sistem otonom. Ini termasuk pola untuk orkestrasi tugas, delegasi subagen, koordinasi berbasis peristiwa, observabilitas, dan kontrol. Aspek-aspek ini mempromosikan arsitektur AI yang dapat diskalakan, dapat dikomposisi, dan dapat diaudit.

Tentang seri konten ini

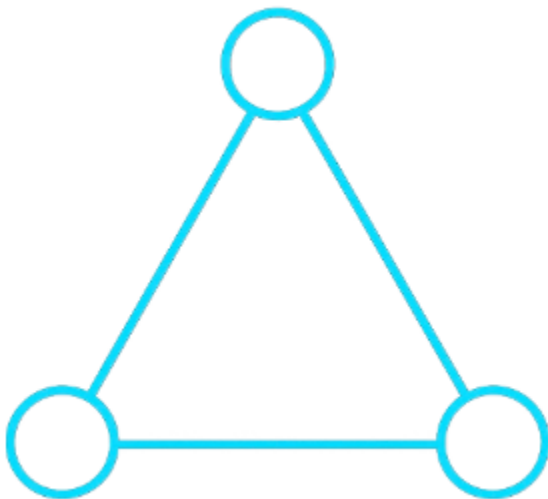
Panduan ini adalah bagian dari seri tentang AI agen di AWS. Untuk informasi lebih lanjut dan untuk melihat panduan lain dalam seri ini, lihat [Agentic AI](#) di situs web AWS Prescriptive Guidance.

Pola agen

Pola agen adalah blok bangunan yang dapat digunakan kembali dan dapat dikomposisi yang dapat disesuaikan dengan domain tertentu, kasus penggunaan, dan tingkat kompleksitas. Sistem agen berbeda, bagaimanapun, dari aplikasi tradisional. Inti dari semua desain agen AI adalah model konseptual yang berlabuh dalam tiga prinsip dasar berikut:

- Asinkron — Agen beroperasi di lingkungan yang kaya peristiwa yang digabungkan secara longgar
- Otonomi — Agen bertindak secara independen, tanpa kontrol manusia atau eksternal
- Agensi — Agen bertindak dengan tujuan, atas nama pengguna atau sistem, menuju tujuan tertentu

Segitiga dalam diagram berikut mewakili blok bangunan inti dari agen perangkat lunak: persepsi, alasan, dan tindakan. Hal ini memungkinkan sistem agen untuk mengamati, membuat keputusan, dan bertindak dalam lingkungannya.



Secara desain, pola agen menyediakan bahasa desain modular untuk membangun sistem AI, yang berarti mereka dapat diakses, operasional, dapat diperluas, dan siap produksi. Merancang sistem ini membutuhkan perhatian yang cermat terhadap tiga dimensi yang saling terkait berikut, yang selanjutnya dibahas dalam panduan ini.

Dalam bagian ini

- [Agen penalaran dasar](#)
- [Tool-based agen untuk memanggil fungsi](#)
- [Tool-based agen untuk server](#)

- [Computer-use agen](#)
- [Agen pengkodean](#)
- [Agen pidato dan suara](#)
- [Agen orkestrasi alur kerja](#)
- [Memory-augmented agen](#)
- [Agen simulasi dan test-bed](#)
- [Pengamat dan agen pemantauan](#)
- [Multi-agent kolaborasi](#)

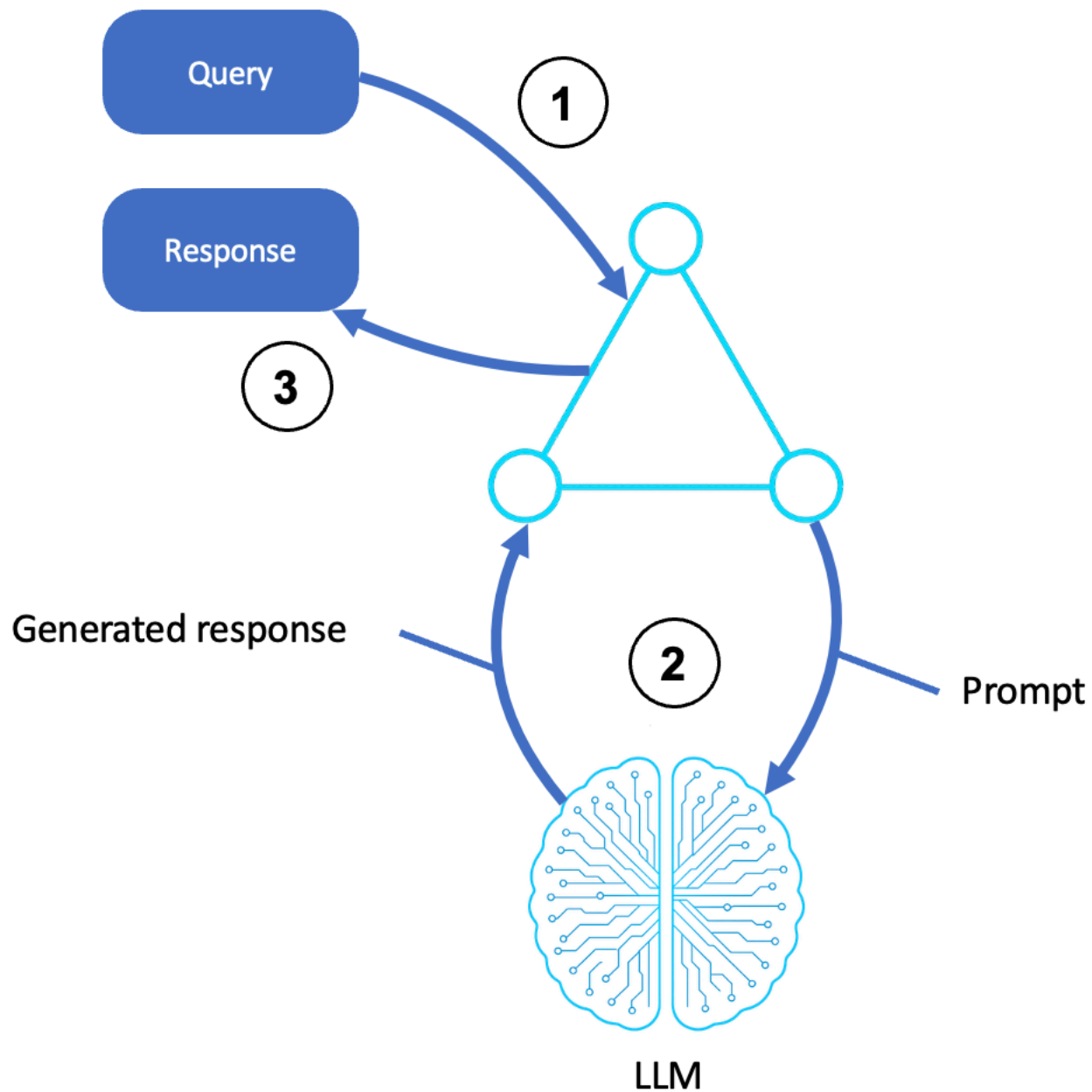
Agen penalaran dasar

Agen penalaran dasar adalah bentuk paling sederhana dari AI agen yang melakukan inferensi logis atau pengambilan keputusan dalam menanggapi kueri. Ini menerima masukan dari pengguna atau sistem dan memproses kueri dan menghasilkan tanggapan menggunakan prompt terstruktur.

Pola ini berguna untuk tugas-tugas yang membutuhkan penalaran, klasifikasi, atau ringkasan satu langkah berdasarkan konteks tertentu. Itu tidak menggunakan memori, alat, atau manajemen status, yang membuatnya stateless, ringan, dan sangat dapat dikomposisi di seluruh alur kerja besar.

Arsitektur

Aliran agen penalaran dasar ditunjukkan pada diagram berikut:



Deskripsi

1. Menerima masukan

- Pengguna, sistem, atau agen hulu mengirimkan kueri atau instruksi.
- Input diserahkan ke shell agen atau lapisan orkestrasi.
- Langkah ini mencakup setiap preprocessing, prompt templating, dan identifikasi tujuan.

2. Memanggil LLM

- Agen mengubah kueri menjadi prompt terstruktur dan mengirimkannya ke LLM (misalnya, melalui Amazon Bedrock).
- LLM menghasilkan respons berdasarkan prompt menggunakan pengetahuan dan konteks yang telah dilatih sebelumnya.
- Output yang dihasilkan dapat mencakup langkah-langkah penalaran (rantai pemikiran), jawaban akhir, atau opsi peringkat.

3. Mengembalikan respon

- Output yang dihasilkan diteruskan ke antarmuka agen.
- Ini mungkin termasuk pemformatan, postprocessing, atau respons API.

Kemampuan

- Mendukung bahasa alami atau masukan terstruktur
- Menggunakan rekayasa cepat untuk memandu perilaku
- Tanpa kewarganegaraan dan terukur
- Dapat disematkan ke UI, CLI, API, dan pipeline

Batasan

- Tidak ada ingatan atau kesadaran historis
- Tidak ada interaksi dengan alat eksternal atau sumber data
- Terbatas pada apa yang diketahui LLM pada saat inferensi

Kasus penggunaan umum

- Pertanyaan dan jawaban percakapan
- Penjelasan dan ringkasan kebijakan
- Panduan untuk membuat keputusan
- Alur chatbot yang ringan dan otomatis
- Klasifikasi, pelabelan, dan penilaian

Panduan implementasi

Anda dapat menggunakan alat dan layanan berikut untuk membuat agen penalaran dasar:

- Amazon Bedrock untuk doa LLM (Antropik, AI21, Meta)
- Amazon API Gateway atau AWS Lambda untuk mengeksposnya sebagai layanan mikro tanpa kewarganegaraan
- Template prompt disimpan di Parameter Store AWS Secrets Manager,, atau sebagai kode

Ringkasan

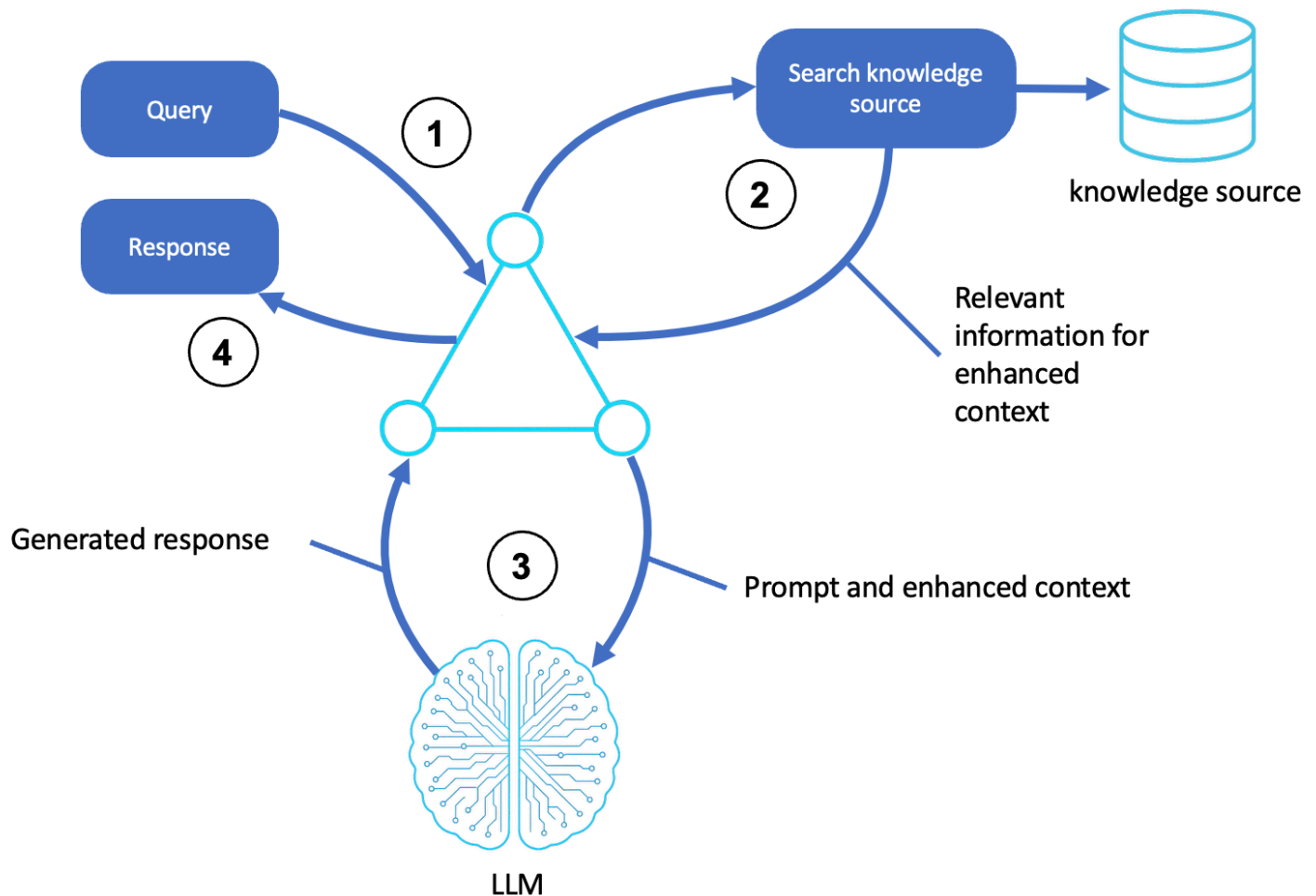
Agen penalaran dasar adalah dasar karena strukturnya yang sederhana. Ini memiliki kemampuan inti yang mengubah tujuan menjadi jalur penalaran yang mengarah pada output cerdas. Pola ini sering menjadi titik awal untuk pola lanjutan, seperti agen berbasis alat dan agen yang menggunakan retrieval-augmented generation (RAG). Ini juga merupakan komponen yang andal dan modular dari alur kerja besar.

Agen RAG

Retrieval-augmented Generasi (RAG) adalah teknik yang menggabungkan pengambilan informasi dengan pembuatan teks untuk menciptakan respons yang akurat dan kontekstual. RAG memungkinkan agen untuk mengambil informasi eksternal yang relevan sebelum melibatkan LLM. Ini memperluas memori efektif agen dan akurasi penalaran dengan membumikan keputusannya dalam informasi terkini, faktual, atau spesifik domain. Berbeda dengan LLM stateless yang hanya mengandalkan bobot yang telah dilatih sebelumnya, RAG memiliki lapisan pencarian pengetahuan eksternal yang secara dinamis meningkatkan permintaan dengan konteks.

Arsitektur

Logika pola RAG diilustrasikan dalam diagram berikut:



Deskripsi

1. Menerima kueri

- Pengguna atau sistem hulu mengirimkan kueri atau tujuan ke agen.
- Shell agen menerima permintaan dan memformatnya sebagai prompt untuk penalaran.

2. Mencari sumber eksternal

- Agen mengidentifikasi konsep dan maksud dari kueri.
- Ini menanyakan sumber pengetahuan, seperti penyimpanan vektor, database, atau indeks dokumen menggunakan pencarian semantik atau pencocokan kata kunci.
- Bagian, dokumen, atau entitas yang paling relevan diambil untuk digunakan pada langkah berikutnya.

3. Menghasilkan respons kontekstual

- Agen menambah prompt dengan informasi yang diambil, membentuk input yang ditingkatkan konteks untuk LLM.
- LLM memproses setiap input menggunakan penalaran generatif (misalnya, rantai pemikiran atau refleksi) untuk menghasilkan respons yang akurat.

4. Mengembalikan output akhir

- Agen menyiapkan output dengan membungkusnya dalam header komunikasi atau format yang diperlukan dan kemudian mengembalikannya ke pengguna atau sistem panggilan.
- (Opsional) Dokumen yang diambil dan keluaran LLM dapat dicatat, dinilai, dan disimpan dalam memori untuk kueri masa depan.

Kemampuan

- Fact-grounded output bahkan dalam domain ekor panjang atau khusus perusahaan
- Ekstensi memori tanpa menyempurnakan model
- Konteks dinamis berdasarkan setiap kueri dan status pengguna
- Sepenuhnya kompatibel dengan database vektor, indeks semantik, dan penyaringan metadata

Kasus penggunaan umum

- Asisten pengetahuan perusahaan
- Bot kepatuhan peraturan
- Copilot dukungan pelanggan
- Search-enhanced chatbots
- Agen dokumentasi pengembang

Panduan implementasi

Gunakan alat dan layanan berikut untuk membuat agen yang menggunakan RAG:

- Amazon Bedrock untuk pemanggilan LLM
- Amazon Kendra, OpenSearch, atau Amazon Aurora untuk dokumentasi atau pencarian data terstruktur
- Amazon Simple Storage Service (Amazon S3) untuk penyimpanan dokumen

- AWS Lambda untuk mengatur pencarian, prompt, dan inferensi LLM
- Knowledge-based integrasi dengan agen (dengan menggunakan plugin memori, retriever semantik, atau Amazon Bedrock)

Ringkasan

Agen RAG menghubungkan penalaran model statis dengan kecerdasan dunia nyata yang dinamis. Ini melengkapi agen dengan kemampuan untuk mencari apa yang tidak mereka ketahui, mensintesis jawaban dari pengetahuan yang diambil, dan menghasilkan respons yang dapat diaudit dengan kepercayaan tinggi.

Pola RAG adalah fondasi untuk membangun agen cerdas yang meningkatkan akses pengetahuan tanpa pelatihan ulang. Ini sering merupakan pendahulu pola orkestrasi yang lebih kompleks yang melibatkan penggunaan alat, perencanaan, dan memori jangka panjang.

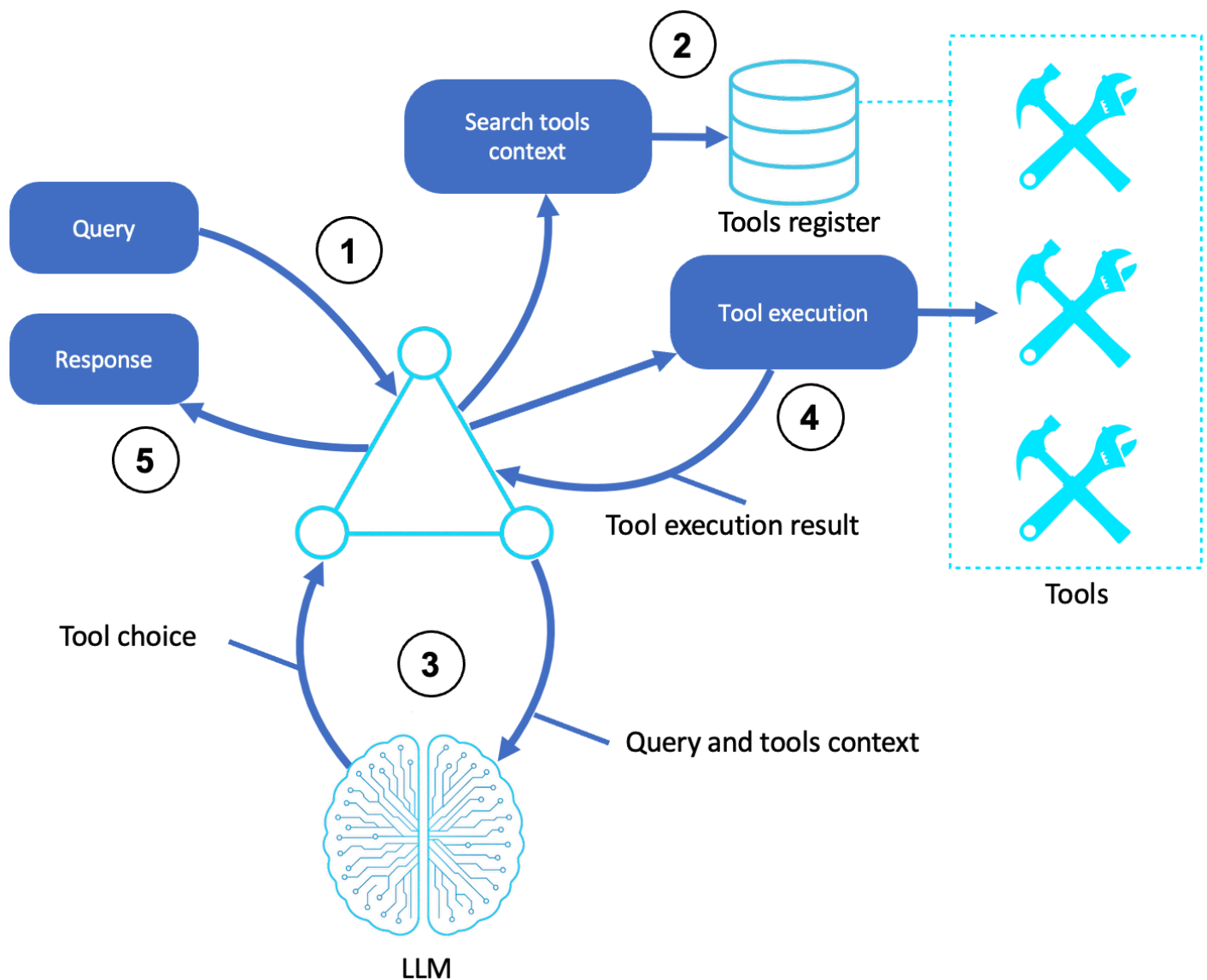
Tool-based agen untuk memanggil fungsi

Tool-based agen memperluas kemampuan agen penalaran dengan menggunakan fungsi eksternal atau API untuk menyelesaikan tugas yang melampaui penalaran bahasa saja. Pola ini menggunakan LLM untuk memutuskan alat mana yang akan digunakan dan kemudian menghasilkan argumen panggilan dan menggabungkan output alat ke dalam loop penalarannya.

Pola ini memungkinkan agen untuk bertindak daripada hanya memberikan tanggapan. Antarmuka alat mewakili kemampuan yang dapat dipanggil, mulai dari perhitungan aritmatika dan pencarian database hingga API eksternal dan layanan cloud.

Arsitektur

Agen berbasis alat untuk memanggil fungsi ditunjukkan dalam diagram berikut:



Deskripsi

1. Menerima kueri

- Agen menerima kueri atau tugas bahasa alami dari pengguna atau sistem panggilan.

2. Telusuri alat

- Agen menggunakan metadata internal atau registri alat untuk mencari alat, skema, dan kemampuan yang relevan yang tersedia.

3. Memilih dan memanggil alat

- LLM menerima kueri dan metadata alat (misalnya, nama fungsi, jenis input, dan deskripsi) dalam prompt nya.

- Ia memilih alat yang paling relevan, membangun argumen masukan, dan mengembalikan panggilan fungsi terstruktur.
4. Menjalankan alat yang dipilih
 - Shell agen atau tool runner mengeksekusi fungsi yang dipilih dan mengembalikan hasilnya (misalnya, output API, nilai database, atau komputasi).
 5. Mengembalikan respon
 - LLM meneruskan hasil ke agen, baik secara langsung atau sebagai bagian dari prompt yang diperbarui. Kemudian mengembalikan hasil bahasa alami.

Kemampuan

- Pemilihan alat dinamis berdasarkan konteks tugas
- Schema-based meminta (OpenAPI, skema AWS JSON, antarmuka fungsi)
- Interpretasi hasil dan rantai output ke dalam penalaran
- Operasi tanpa kewarganegaraan atau sadar sesi

Kasus penggunaan umum

- Asisten virtual dengan akses data eksternal
- Kalkulator dan estimator keuangan
- API-based pekerja pengetahuan
- LLM yang memanggil, SageMaker titik akhir AWS Lambda Amazon, dan layanan SaaS

Panduan implementasi

Gunakan yang berikut ini untuk membuat agen berbasis alat untuk memanggil fungsi:

- Amazon Bedrock dengan dukungan pemanggilan fungsi (Anthropic Claude)
- AWS Lambda sebagai backend eksekusi alat
- Amazon API Gateway atau AWS Step Functions untuk orkestrasi alat
- Amazon DynamoDB atau Amazon Relational Database Service (Amazon RDS) untuk metadata alat sadar konteks

- EventBridge Saluran pipa Amazon atau peta AWS Step Functions yang menyatakan untuk merutekan output

Ringkasan

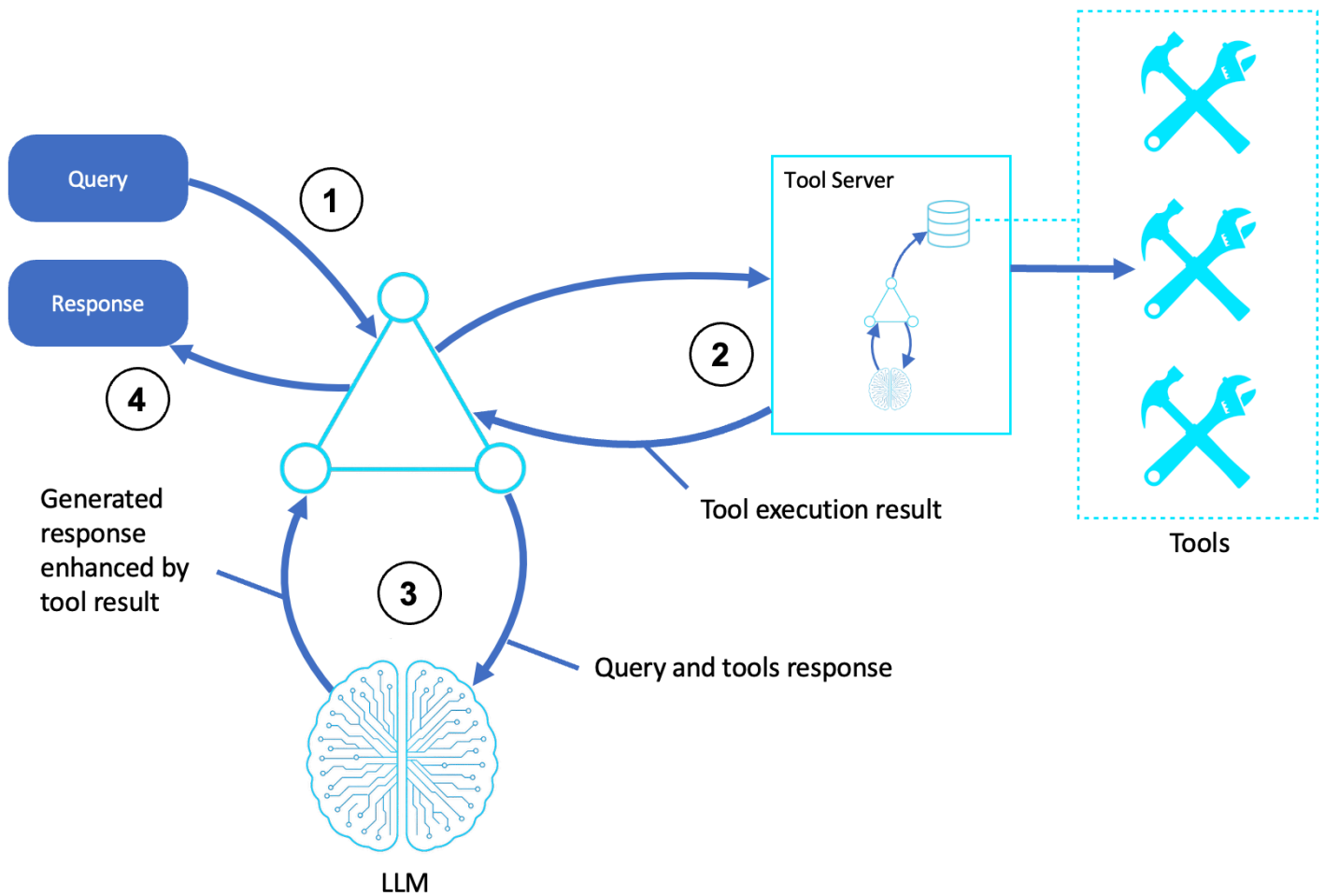
Tool-based agen pemanggil fungsi mewakili pergeseran dari pemahaman bahasa ke melakukan tindakan. Agen-agen ini menggunakan alat yang dinamis dan sadar konteks sambil mempertahankan penalaran LLM, mengubah asisten pasif menjadi sistem yang menyelesaikan tugas, mengakses layanan, dan mengintegrasikan operasi bisnis. Pola ini merupakan komponen penting dari AI agen dalam pengaturan perusahaan, terutama bila dikombinasikan dengan skema deklaratif, kerangka kerja otorisasi, dan sistem multi-agen.

Tool-based agen untuk server

Tool-based agen untuk server meningkatkan agen pemanggil fungsi dengan mendelegasikan eksekusi alat ke server eksternal yang memiliki lingkungan runtime khusus untuk alat, skrip, dan agen komposit. Tidak seperti panggilan fungsi inline yang dipilih dan dipanggil oleh loop agen, agen berbasis server mengalihdayakan logika dan pipa eksekusi ke agen atau sistem lain. Ini memberikan kemampuan canggih seperti rantai multitool, eksekusi terisolasi, dan penalaran khusus. Server alat ideal untuk tindakan yang kompleks, stateful, atau intensif sumber daya di mana alat itu sendiri mungkin melibatkan model AI, aturan bisnis, atau lingkungan yang terpisah.

Arsitektur

Berikut ini adalah pola untuk agen berbasis alat untuk server:



Deskripsi

1. Menerima kueri

- Pengguna atau sistem mengirimkan permintaan ke shell agen.
- Agen menafsirkan kueri dan bersiap untuk mengirimkannya ke server alat.

2. Menjalankan proses server alat

- Agen mengirimkan tugas, bersama dengan parameter terstruktur, ke server alat.
- Server alat kemudian dapat:
 - Jalankan skrip atau logika dalam sistem komputasi khusus (misalnya, wadah AWS Lambda, atau Amazon) SageMaker
 - Gunakan subagennya sendiri dengan penalaran LLM untuk memilih dan menjalankan alat
 - Mengelola dependensi, percobaan ulang, atau alur eksekusi multistep
 - Hasil keluaran ke agen utama saat tugas selesai

3. Menggunakan penalaran LLM dengan keluaran alat

- Agen memanggil LLM, meneruskan kueri asli dan hasil server alat sebagai bagian dari prompt.
- LLM mensintesis respons yang menggabungkan informasi yang baru diperoleh.

4. Mengembalikan respon

- Agen mengembalikan respons bahasa alami atau terstruktur ke pengguna atau sistem panggilan.
- (Opsional) Hasil dapat disimpan dalam memori atau log audit.

Kemampuan

- Alat dipanggil di luar loop eksekusi agen utama
- Eksekusi alat mungkin melibatkan panggilan LLM, rantai logika, atau subagen
- Agen bertindak sebagai pengontrol atau operator, bukan hanya pembungkus alat
- Mengaktifkan komposabilitas, skalabilitas, dan isolasi logika

Kasus penggunaan umum

- Mengatur rantai model (misalnya, dengan menggabungkan LLM, visi, dan kode)
- AI-driven jaringan pipa otomatisasi
- DevOps asisten agen dengan pelari skrip
- Kompleks perhitungan keuangan, simulasi, atau agen optimasi
- Alat multimodal (misalnya, dengan menggabungkan audio, dokumentasi, dan tindakan)

Panduan implementasi

Anda dapat membangun pola ini menggunakan yang berikut Layanan AWS:

- Amazon Bedrock (host agen dan inferensi LLM)
- AWS Lambda, Amazon ECS, AWS Fargate, atau SageMaker titik akhir Amazon sebagai runtime server alat
- Amazon API Gateway atau AWS App Runner untuk mengekspos API server alat
- Amazon EventBridge untuk pesan agen-ke-alat yang dipisahkan
- AWS Step Functions atau AWS AppFabric untuk menyusun logika multi-agen di server alat

Ringkasan

Tool-based agen yang menggunakan server sangat modular dan dapat diskalakan. Mereka memisahkan logika keputusan dari eksekusi, yang memungkinkan agen utama tetap ringan saat membongkar tindakan kompleks atau sensitif ke sistem lain. Ini penting untuk AI agen tingkat perusahaan, terutama di lingkungan yang membutuhkan tata kelola, observabilitas, isolasi, komposisi dinamis, atau kombinasi apa pun darinya.

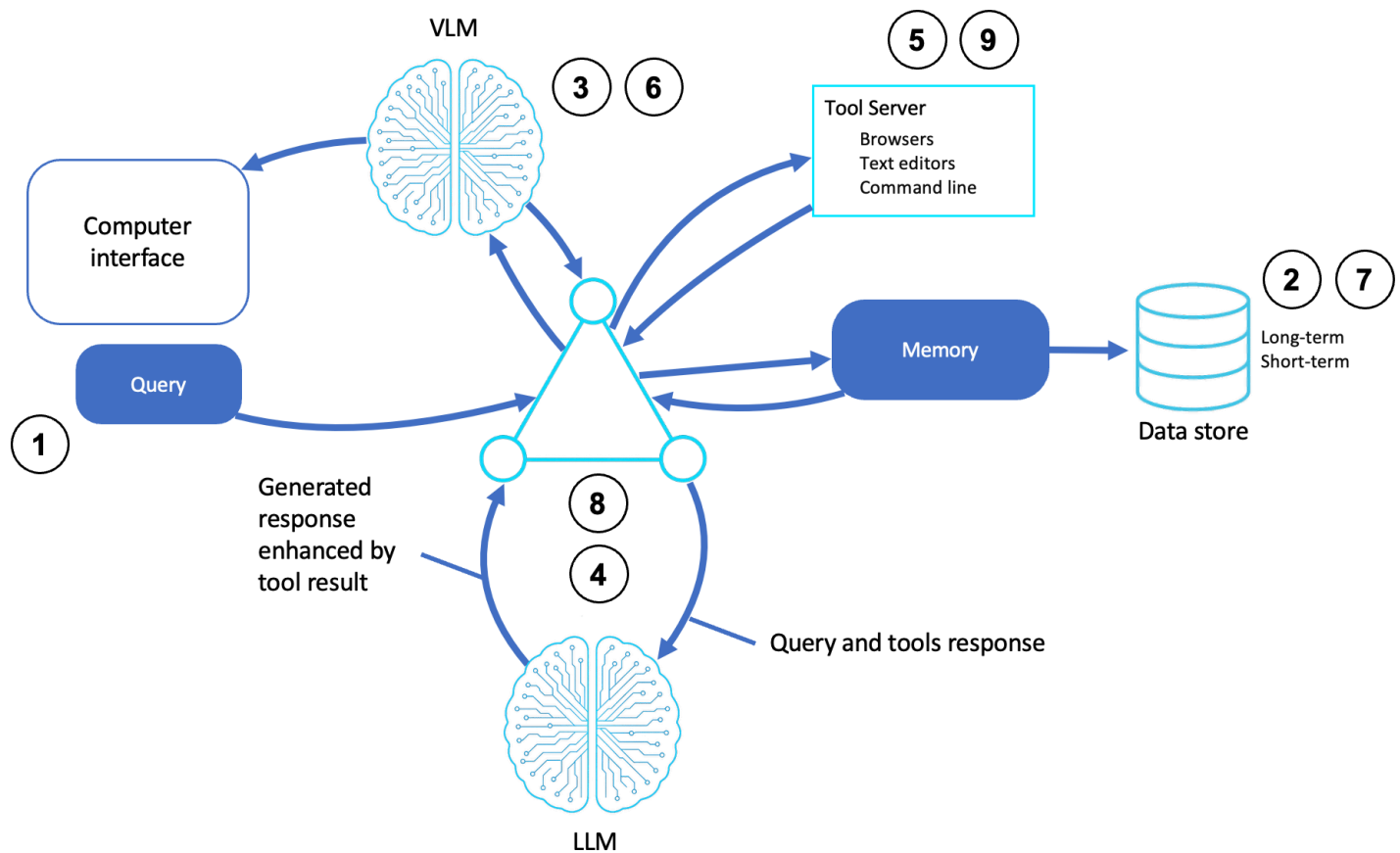
Computer-use agen

Computer-use agen dapat mensimulasikan atau mengontrol lingkungan digital seperti browser, terminal, sistem file, dan aplikasi. Agen ini menafsirkan maksud pengguna, berinteraksi dengan antarmuka visual dan teks, dan melakukan tindakan yang diarahkan pada tujuan dengan menggabungkan penalaran LLM, model bahasa visual (VLM), dan server alat yang menjalankan perintah atau mensimulasikan peristiwa input.

Pola ini penting untuk otomatisasi AI praktis, di mana agen berfungsi tidak hanya sebagai asisten tetapi juga sebagai proxy yang melakukan tindakan seperti manusia, seringkali dengan menggunakan alat dan lingkungan yang sama.

Arsitektur

Pola agen penggunaan komputer ditunjukkan pada diagram berikut:



Deskripsi

1. Menerima kueri

- Tugas atau permintaan disediakan melalui UI, API, atau antarmuka bahasa alami.

2. Mengakses memori

- Agen mengambil memori jangka pendek dan jangka panjang untuk mengingat perintah, tujuan, dan status sistem masa lalu.

3. Menganalisis konteks visual

- VLM mengamati layar komputer, status sistem, atau elemen UI untuk memahami konteks tertentu dan mengidentifikasi item yang dapat ditindaklanjuti.

4. Alasan melalui LLM

- LLM menggabungkan kueri, status memori, alat, dan respons server untuk menentukan tindakan selanjutnya.

5. Berinteraksi dengan server alat

- Agen memanggil alat yang di-host di server, yang mungkin termasuk yang berikut:

- Browser (misalnya, Chrome tanpa kepala) dan lingkungan shell
 - Editor teks dan kode
 - Antarmuka skrip kustom
6. Memperbarui input visual
- Jika UI sistem berubah atau pengamatan lebih lanjut diperlukan, VLM dapat menganalisis ulang status layar atau buffer teks.
7. Memperbarui memori
- Wawasan baru, status sistem, atau umpan balik pengguna ditulis ke memori jangka pendek dan jangka panjang.
8. Merumuskan keputusan akhir dan penjelasan
- LLM mensintesis hasil atau merekomendasikan tindakan berdasarkan kueri dan output alat.
9. Mengembalikan respon
- Agen mengembalikan hasil ke antarmuka (misalnya, tugas yang diselesaikan, konfirmasi, atau konten yang dihasilkan).

Kemampuan

- Penalaran multimodal dengan input visual dan tekstual
- Kontrol atas aplikasi melalui simulasi atau input API-driven
- Manajemen memori untuk keadaan persisten
- Otonomi dalam eksekusi urutan (aliran multistep)

Kasus penggunaan umum

- Pengembang AI yang menulis dan menjalankan kode di IDE
- Computer-use agen untuk alur kerja digital berulang
- Pengguna simulasi untuk pengujian perangkat lunak dan jaminan kualitas
- Agen aksesibilitas untuk menavigasi UI melalui suara atau instruksi tingkat tinggi
- Otomatisasi proses robot cerdas (RPA) yang ditingkatkan dengan penalaran

Panduan implementasi

- Anda dapat membangun pola ini menggunakan yang berikut Layanan AWS:
- Amazon Bedrock untuk LLM-based perencanaan dan penalaran
- Amazon Elastic Compute Cloud (Amazon EC2) AWS Lambda,, atau notebook SageMaker Amazon untuk menjalankan server alat dengan lingkungan UI simulasi
- Amazon Simple Storage Service (Amazon S3) atau Amazon DynamoDB untuk persistensi memori
- Amazon Rekognition (atau model kustom) untuk analisis gambar UI dalam skenario hybrid
- CloudWatch Log Amazon atau AWS X-Ray untuk jalur observabilitas dan audit

Ringkasan

Computer-use Agen bertindak sebagai operator digital otonom, menjembatani kesenjangan antara interaksi manusia-komputer dan tindakan. AI-driven Dengan menggabungkan memori, orkestrasi alat, dan VLM, agen ini dapat berinteraksi secara adaptif dengan sistem yang dirancang untuk manusia, menjalankan tindakan, memperbarui file, menavigasi menu, dan menghasilkan respons.

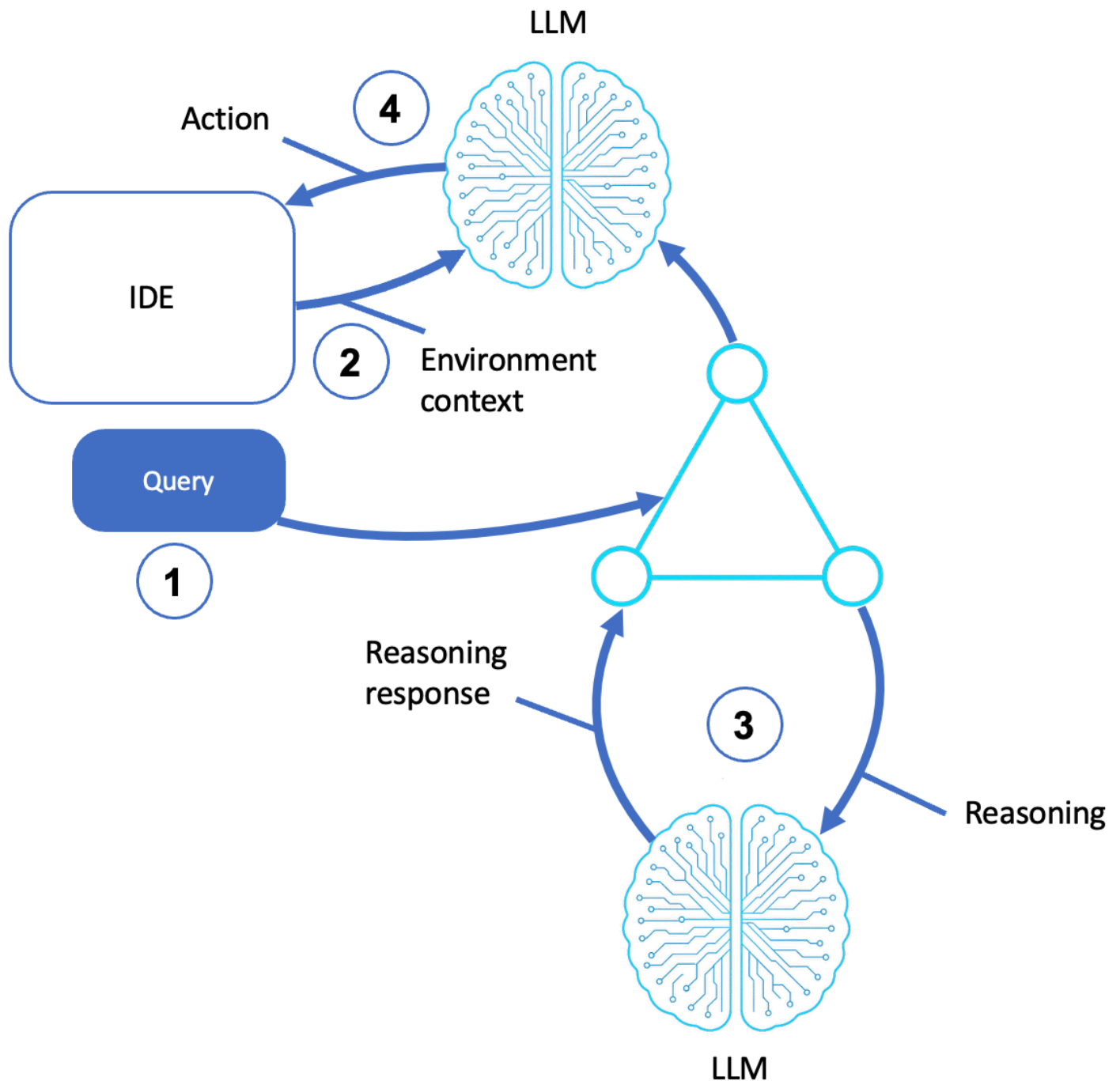
Agen pengkodean

Agen pengkodean dapat beralasan tentang tugas pemrograman, menghasilkan atau memodifikasi kode, dan berinteraksi dengan lingkungan pengembang, seperti IDE dan CLI. Agen-agen ini menggabungkan pemahaman bahasa alami dengan penalaran terstruktur untuk membantu, menambah, dan mengotomatiskan pengembangan perangkat lunak, mulai dari pembuatan fungsi hingga perbaikan bug dan penulisan pengujian.

Tidak seperti alat pelengkapan otomatis, agen pengkodean secara aktif menafsirkan tujuan pengguna, menanyakan lingkungan pengembangan untuk konteks (misalnya, membuka file dan melacak kesalahan), mengidentifikasi persyaratan, dan kemudian mengusulkan dan melakukan tindakan.

Arsitektur

Pola agen pengkodean ditunjukkan pada diagram berikut:



Deskripsi

1. Menerima kueri

- Pengguna menyediakan instruksi bahasa alami melalui palet perintah, jendela obrolan, atau CLI (misalnya, "Tambahkan logging ke fungsi ini" atau "Refactor untuk keterbacaan").

2. Ekstrak konteks lingkungan

- Agen mengumpulkan konteks dari IDE, termasuk file aktif, posisi kursor, cuplikan kode, dan tabel simbol.
- Ini mengeluarkan pesan kesalahan, hasil pengujian, dan output dari agen lain.

3. Alasan LLM

- Agen mengirimkan prompt, termasuk kueri dan konteks lingkungan, ke LLM.
 - LLM melakukan pass penalaran untuk menentukan hal-hal berikut:
 - Apa yang perlu diubah
 - Cara menghasilkan solusi
 - Setiap langkah refactoring, penulisan ulang, atau pengkodean

4. Mengeksekusi tindakan

- LLM mengembalikan output ke agen dan mengimpornya ke IDE atau lingkungan runtime.
- Ini mungkin termasuk memasukkan atau memodifikasi kode, membuat komentar atau dokumentasi, dan memicu tugas build, pengujian, dan linting hilir.

Kemampuan

- High-contextual kesadaran (misalnya, status IDE, kursor, dan pohon sintaks)
- Penalaran berulang dari tujuan dan umpan balik
- Perencanaan kode opsional dan pemisahan tindakan (misalnya, alasan pertama dan kemudian bertindak)
- Bekerja dalam alur kerja pengembang sinkron atau asinkron

Kasus penggunaan umum

- Pembuatan kode dari deskripsi tugas
- Pemfaktoran ulang dan pengoptimalan kode
- Test-case generasi dan validasi
- Penjelasan kesalahan dan debugging
- Asisten dokumentasi
- Kopilot pemrograman berpasangan

Panduan implementasi

- Anda dapat membangun pola ini menggunakan alat-alat berikut dan Layanan AWS:
- Amazon Bedrock untuk LLM-driven generasi dan penalaran
- Pengembang Amazon Q untuk saran dan penyelesaian pengkodean
- AWS Lambda atau Amazon Elastic Container Service (Amazon ECS) untuk menjalankan dan menguji lingkungan sandbox
- AWS Cloud9, ekstensi VS Code, atau integrasi IDE khusus untuk meng-host dan mengevaluasi konteks
- Amazon Simple Storage Service (Amazon S3) untuk menyimpan prompt perantara, respons, dan riwayat revisi

Ringkasan

Coding agent adalah alat AI-powered pengembangan baru yang mampu menafsirkan bahasa alami, menganalisis konteks, menghasilkan perubahan kode multistep, dan mengintegrasikan dengan siklus hidup pengembangan perangkat lunak.

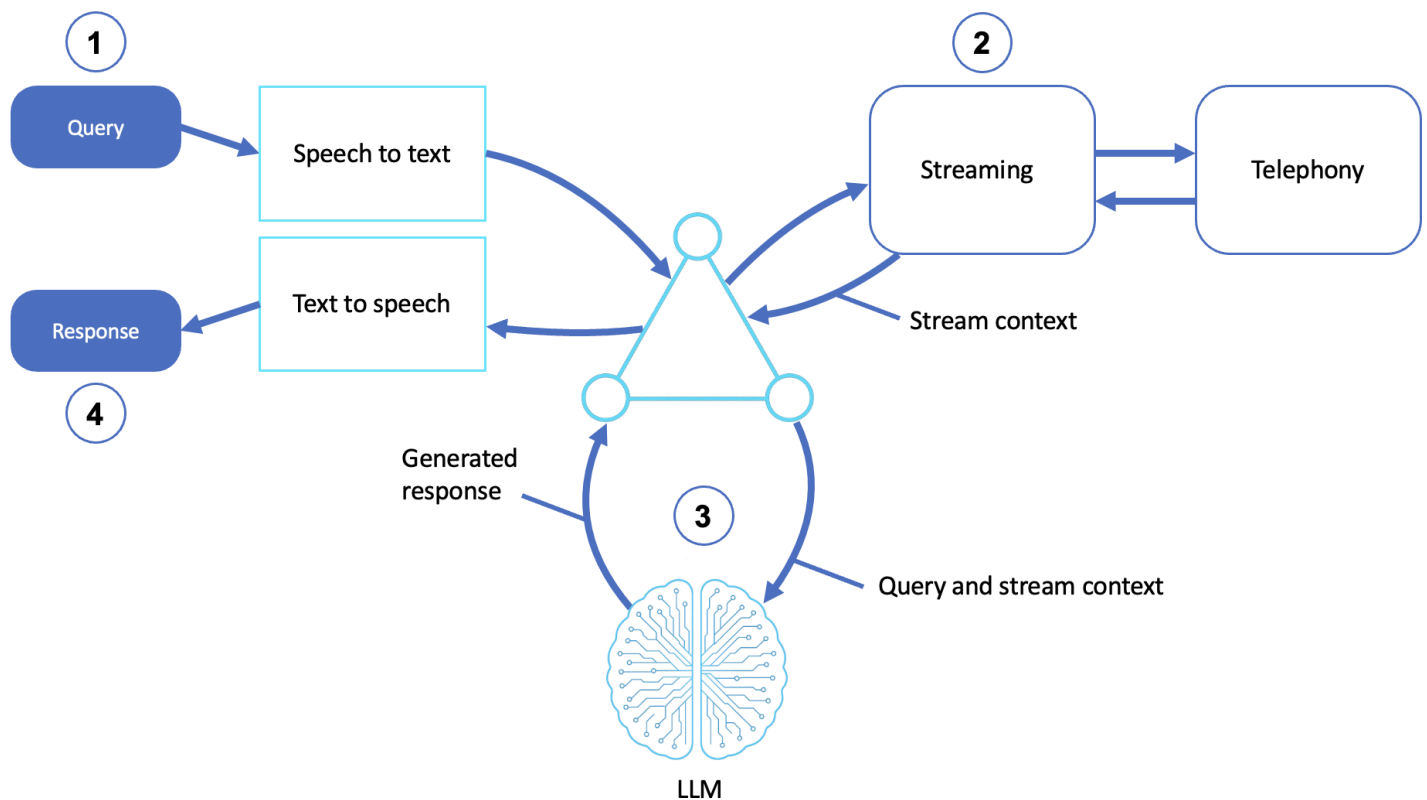
Agen pidato dan suara

Agen pidato dan suara berinteraksi dengan pengguna melalui dialog lisan. Agen-agen ini mengintegrasikan pengenalan suara, pemahaman bahasa alami, dan sintesis ucapan untuk memungkinkan AI percakapan di seluruh platform telepon, seluler, web, dan tertanam.

Agen suara sangat efektif dalam lingkungan hands-free, real-time, atau berbasis aksesibilitas. Dengan menggabungkan antarmuka streaming dengan LLM-powered penalaran, mereka memfasilitasi interaksi yang kaya dan dinamis yang terasa alami bagi pengguna.

Arsitektur

Agen pidato dan suara ditunjukkan dalam diagram berikut:



Deskripsi

1. Menerima kueri suara

- Pengguna menyuarakan permintaan ke telepon, mikrofon, atau sistem tertanam.
- Modul speech-to-text (STT) mengubah audio menjadi teks.

2. Mengintegrasikan konteks streaming dan telepon

- Agen menggunakan antarmuka streaming untuk mengelola audio I/O secara real time.
- Jika digunakan di pusat kontak atau konteks telekomunikasi, integrasi telepon menangani perutean sesi, input multi-frekuensi nada ganda (DTMF), dan transportasi media.

Catatan: DTMF mengacu pada nada yang dihasilkan saat Anda menekan tombol pada keypad telepon. Dalam konteks integrasi konteks streaming dan telepon dalam agen suara, DTMF digunakan sebagai mekanisme input sinyal selama panggilan telepon, terutama dalam sistem respons suara interaktif (IVR). Input DTMF memungkinkan agen untuk:

- Kenali pilihan menu (misalnya, “Tekan 1 untuk penagihan. Tekan 2 untuk dukungan.”)
- Kumpulkan input numerik (misalnya, nomor akun, PIN, dan nomor konfirmasi)

- Memicu alur kerja atau transisi status dalam alur panggilan
- Kembalikan dari ucapan ke nada sentuh bila perlu

1. Alasan melalui konteks aliran LLM

- Kueri dikirim ke agen, yang meneruskannya, bersama dengan metadata sesi apa pun (misalnya, ID pemanggil, konteks sebelumnya), ke LLM.
- LLM menghasilkan respons, mungkin menggunakan strategi rantai pemikiran atau memori multiturn jika interaksi sedang berlangsung.

2. Mengembalikan respons suara

- Agen mengubah responsnya terhadap ucapan menggunakan text-to-speech (TTS).
- Ini mengembalikan audio ke pengguna melalui saluran suara.

Kemampuan

- Real-time pemahaman pidato dan generasi
- Multilingual I/O dengan dukungan STT dan TTS
- Integrasi dengan API telepon atau streaming
- Kesadaran sesi dan serah terima memori di antara belokan

Kasus penggunaan umum

- Sistem IVR percakapan
- Resepsionis virtual dan penjadwal janji temu
- Voice-driven agen helpdesk
- Asisten suara yang dapat dikenakan
- Antarmuka suara untuk rumah pintar dan alat aksesibilitas

Panduan implementasi

Anda dapat membangun pola ini menggunakan alat-alat berikut dan Layanan AWS:

- Amazon Lex V2 atau Amazon Transcribe untuk STT
- Amazon Polly untuk TTS

- Amazon Chime SDK, Pelanggan Amazon Connect, atau Amazon Interactive Video Service (Amazon IVS) untuk streaming dan telepon
- Amazon Bedrock untuk penalaran dengan Anthropic, AI21, atau model yayasan lainnya
- AWS Lambda untuk menghubungkan STT, LLM, TTS, dan konteks sesi

(Opsional) Peningkatan tambahan dapat mencakup yang berikut:

- Amazon Kendra atau OpenSearch untuk RAG yang sadar konteks
- Amazon DynamoDB untuk memori sesi
- CloudWatch Log Amazon dan AWS X-Ray untuk ketertelusuran

Ringkasan

Agen bicara dan suara adalah sistem cerdas yang berinteraksi melalui percakapan alami. Dengan mengintegrasikan antarmuka ucapan dengan penalaran LLM dan infrastruktur streaming waktu nyata, agen suara memungkinkan interaksi yang mulus, dapat diakses, dan terukur.

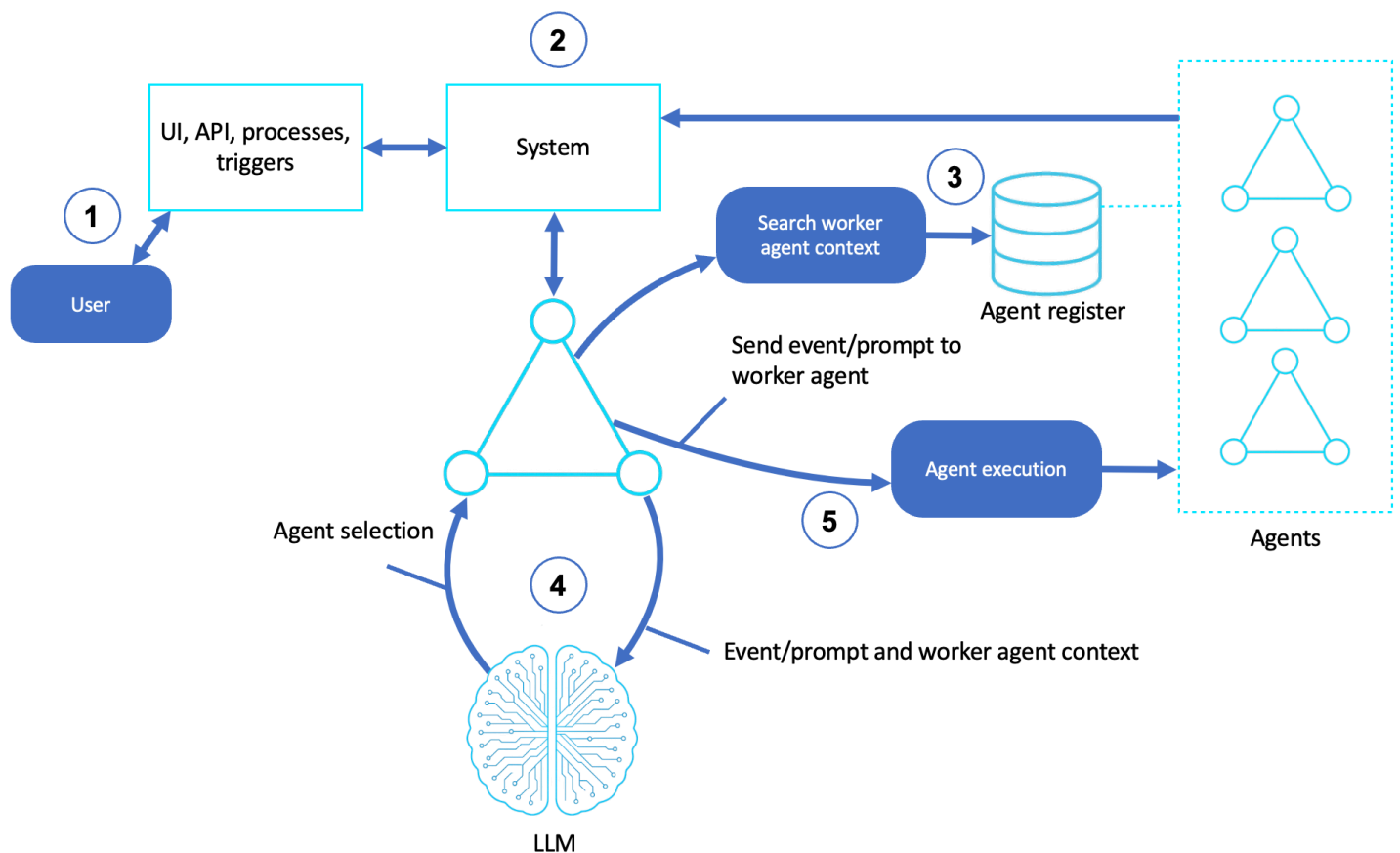
Agen orkestrasi alur kerja

Agen orkestrasi alur kerja mengelola dan mengoordinasikan tugas, proses, dan layanan multistep di seluruh sistem terdistribusi. Alih-alih bernalar dan bertindak secara terpisah, agen ini mendelegasikan pekerjaan ke subagen atau sistem lain, mempertahankan konteks eksekusi, dan beradaptasi berdasarkan hasil antara.

Agen ini adalah bagian mendasar dari arus otomatisasi. Mereka sangat berguna saat menangani tugas yang berjalan lama, komposisi multi-agen, dan integrasi lintas domain di mana berbagai agen dan alat harus dipanggil secara berurutan atau kondisional.

Arsitektur

Agen orkestrasi alur kerja ditunjukkan pada diagram berikut:



Deskripsi

1. Menerima masukan pengguna

- Pengguna (atau pemicu eksternal) memulai tugas melalui UI, API, atau peristiwa sistem.

2. Menangani peristiwa sistem

- Komponen sistem menerima permintaan dan memancarkan peristiwa atau perintah yang membutuhkan orkestrasi.

3. Mengambil konteks

- Agen alur kerja menanyakan basis pengetahuan dan pendaftar agen untuk menemukan agen pekerja yang tepat untuk tugas berdasarkan metadata, domain, dan tingkat keberhasilan sebelumnya.

4. Memilih agen LLM

- LLM membantu memilih agen atau rencana alur kerja terbaik dengan menganalisis deskripsi tugas dan opsi yang tersedia.
- Ini juga dapat merumuskan permintaan khusus tugas untuk dikirim ke agen yang dipilih.

5. Mendelegasikan dan mengeksekusi

- Agen pekerja yang dipilih menerima acara atau prompt dan mulai menjalankan perintah.
- Ini dapat melacak status eksekusi, mencoba lagi kegagalan, dan meneruskan hasil perantara ke agen berikutnya dalam urutan.

Kemampuan

- Komposisi agen (misalnya, supervisor, agen kolaborator, dan alat)
- Event-driven atau eksekusi terjadwal
- Memori dan melacak status dari waktu ke waktu
- Orkestrasi tugas hierarkis atau paralel (sinkron dibandingkan dengan alur kerja asinkron)
- Pemilihan dan rantai agen dinamis

Kasus penggunaan umum

- Otomatisasi multistep (misalnya, penyerapan dan pelaporan data)
- Perutean dan eskalasi layanan pelanggan (misalnya, agen-sebagai-koordinator)
- Agen AI berkoordinasi dengan manusia dan bot dalam lingkaran yang sama
- Mengotomatiskan proses perusahaan menggunakan logika LLM-powered
- Sistem hibrida menggabungkan agen AI dan alat orkestrasi tradisional

Panduan implementasi

Anda dapat membangun pola ini menggunakan alat-alat berikut dan Layanan AWS:

- Amazon Bedrock untuk penalaran dan pemilihan agen
- AWS Step Functions atau Amazon EventBridge untuk komposisi alur kerja
- AWS Lambda sebagai unit eksekusi atau pelari tugas
- Amazon DynamoDB, Amazon Simple Storage Service (Amazon S3) Simple Storage Service, atau Amazon RDS untuk melacak status dan hasil
- AWS AppFabric atau Amazon AppFlow untuk koordinasi lintas sistem
- (Opsional) Gunakan agen yang SageMaker dijalankan Amazon untuk meng-host agen pekerja khusus domain

Ringkasan

Agan alur kerja mengoordinasikan, beradaptasi, dan menyelaraskan tujuan di lingkungan multi-agen. Ini berarti bahwa agen AI dapat berkolaborasi, beradaptasi dengan kondisi runtime, dan memberikan hasil yang kompleks melalui alur kerja modular yang dapat dijelaskan.

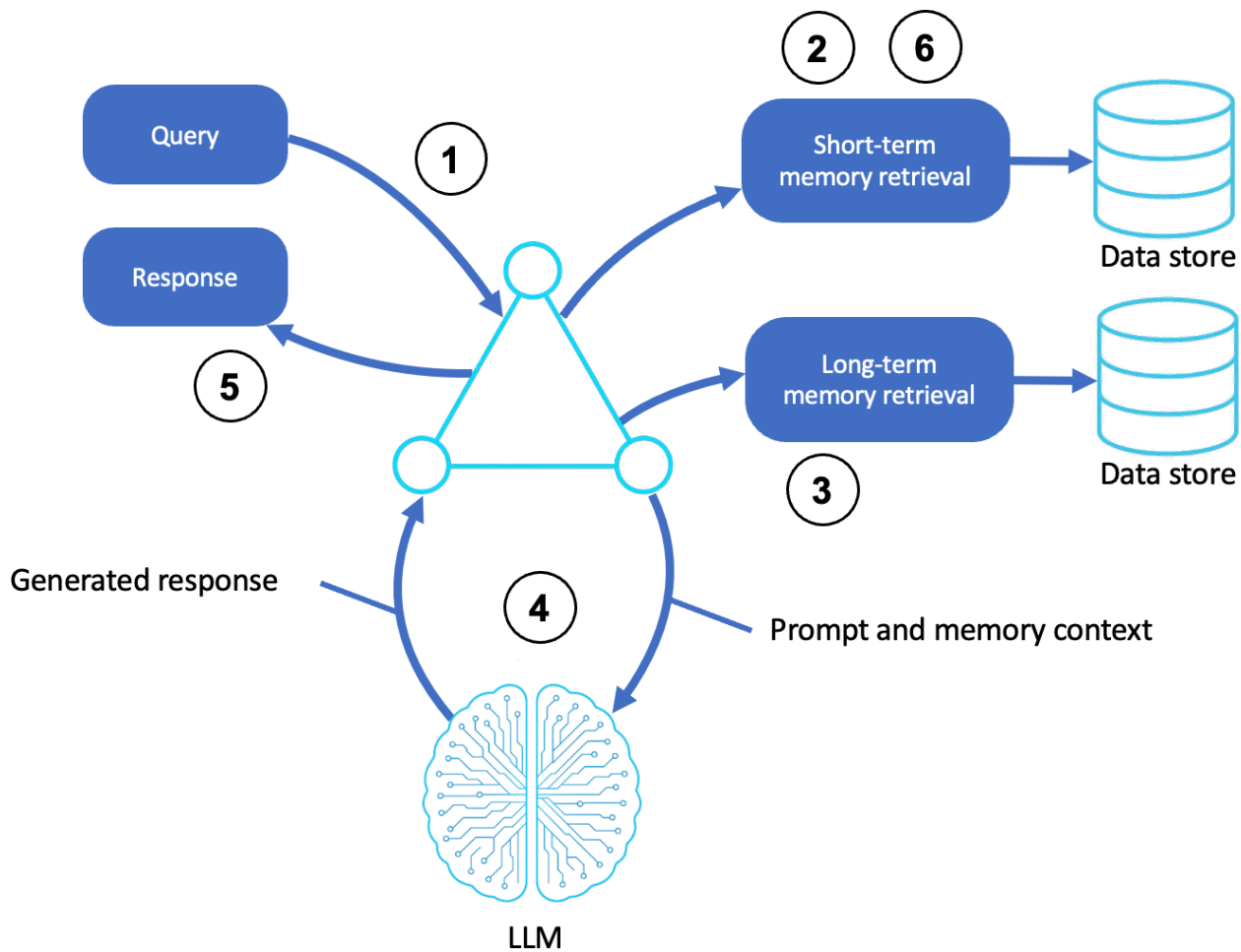
Memory-augmented agen

Memory-augmented agen ditingkatkan dengan kemampuan untuk menyimpan, mengambil, dan bernalar menggunakan memori jangka pendek dan jangka panjang. Hal ini memungkinkan mereka untuk mempertahankan konteks di berbagai tugas, sesi, dan interaksi, yang menghasilkan respons yang lebih koheren, personal, dan strategis.

Tidak seperti agen tanpa kewarganegaraan, agen tambahan memori beradaptasi dengan mereferensikan data historis, belajar dari hasil sebelumnya, dan membuat keputusan yang sesuai dengan tujuan, preferensi, dan lingkungan pengguna.

Arsitektur

Agan tambahan memori ditunjukkan pada diagram berikut:



Deskripsi

1. Menerima masukan atau acara

- Agen menerima permintaan pengguna atau peristiwa sistem. Ini mungkin teks, pemicu API, atau perubahan lingkungan.

2. Mengambil memori jangka pendek

- Agen mengambil riwayat percakapan terbaru, konteks tugas, atau status sistem yang relevan dengan sesi atau alur kerja.

3. Mengambil memori jangka panjang

- Agen menanyakan memori jangka panjang (misalnya, database vektor dan penyimpanan nilai kunci) untuk wawasan historis, seperti berikut ini:
 - Preferensi pengguna
 - Keputusan dan hasil masa lalu

- Konsep, ringkasan, atau pengalaman yang dipelajari

4. Alasan melalui LLM

- Konteks memori tertanam ke dalam prompt LLM, memungkinkan agen untuk bernalar berdasarkan input saat ini dan pengetahuan sebelumnya.

5. Menghasilkan output

- Agen menghasilkan respons, rencana, atau tindakan sadar kontekstual yang dipersonalisasi sesuai dengan riwayat tugas dan input pengguna.

6. Memperbarui memori

- Informasi baru, seperti sasaran yang diperbarui, sinyal keberhasilan dan kegagalan, dan respons terstruktur, disimpan untuk tugas-tugas masa depan.

Kemampuan

- Kontinuitas sesi di seluruh percakapan atau acara
- Kegigihan tujuan dari waktu ke waktu
- Kesadaran kontekstual berdasarkan keadaan yang berkembang
- Kemampuan beradaptasi diinformasikan oleh keberhasilan dan kegagalan sebelumnya
- Personalisasi selaras dengan preferensi dan riwayat pengguna

Kasus penggunaan umum

- Copilot percakapan yang mengingat preferensi pengguna
- Agen pengkodean yang melacak perubahan basis kode
- Agen alur kerja yang beradaptasi sesuai dengan riwayat tugas
- Kembar digital yang berkembang dari pengetahuan sistem
- Agen penelitian yang menghindari pengambilan berlebihan

Menerapkan agen yang diperbesar memori

Gunakan alat-alat berikut dan Layanan AWS untuk agen tambahan memori:

Lapisan memori

Layanan AWS

Tujuan

Short-term	Amazon DynamoDB, Redis, konteks Batuan Dasar Amazon	Pengambilan cepat status interaksi terbaru
Long-term (terstruktur)	Amazon Aurora, Amazon DynamoDB, Amazon Neptune	Fakta, hubungan, dan log
Long-term (semantik)	OpenSearch, PostgreSQL, Biji pinus	Embedding-based pengambilan (yaitu, RAG)
Penyimpanan	Amazon S3	Menyimpan transkrip, memori terstruktur, dan file
Orkestrasi	AWS Lambda atau AWS Step Functions	Mengelola injeksi memori dan memperbarui siklus hidup
Penalaran	Amazon Bedrock	Anthropic Claude atau Mistral dengan petunjuk memori

Menerapkan dorongan yang disuntikkan memori

Untuk mengintegrasikan memori ke dalam penalaran agen, gunakan kombinasi keadaan terstruktur dan injeksi konteks yang ditambah pengambilan:

- Sertakan status agen terbaru dan riwayat dialog terbaru sebagai input terstruktur saat membuat prompt untuk model bahasa, sehingga dapat beralasan dengan konteks penuh.
- Gunakan retrieval-augmented generation (RAG) untuk menarik dokumen atau fakta yang relevan dari memori jangka panjang.
- Meringkas rencana, konteks, dan interaksi sebelumnya untuk kompresi dan relevansi.
- Suntikkan modul memori eksternal, seperti penyimpanan vektor atau log terstruktur, selama inferensi untuk memandu pengambilan keputusan.

Ringkasan

Memory-augmented agen mempertahankan kontinuitas pikiran dengan belajar dari pengalaman dan mengingat konteks pengguna. Agen-agen ini melampaui kecerdasan reaktif dengan menggunakan

kolaborasi jangka panjang, personalisasi, dan penalaran strategis. Dalam hal AI agen, memori memungkinkan agen untuk berperilaku lebih seperti rekan digital adaptif dan kurang seperti alat stateless.

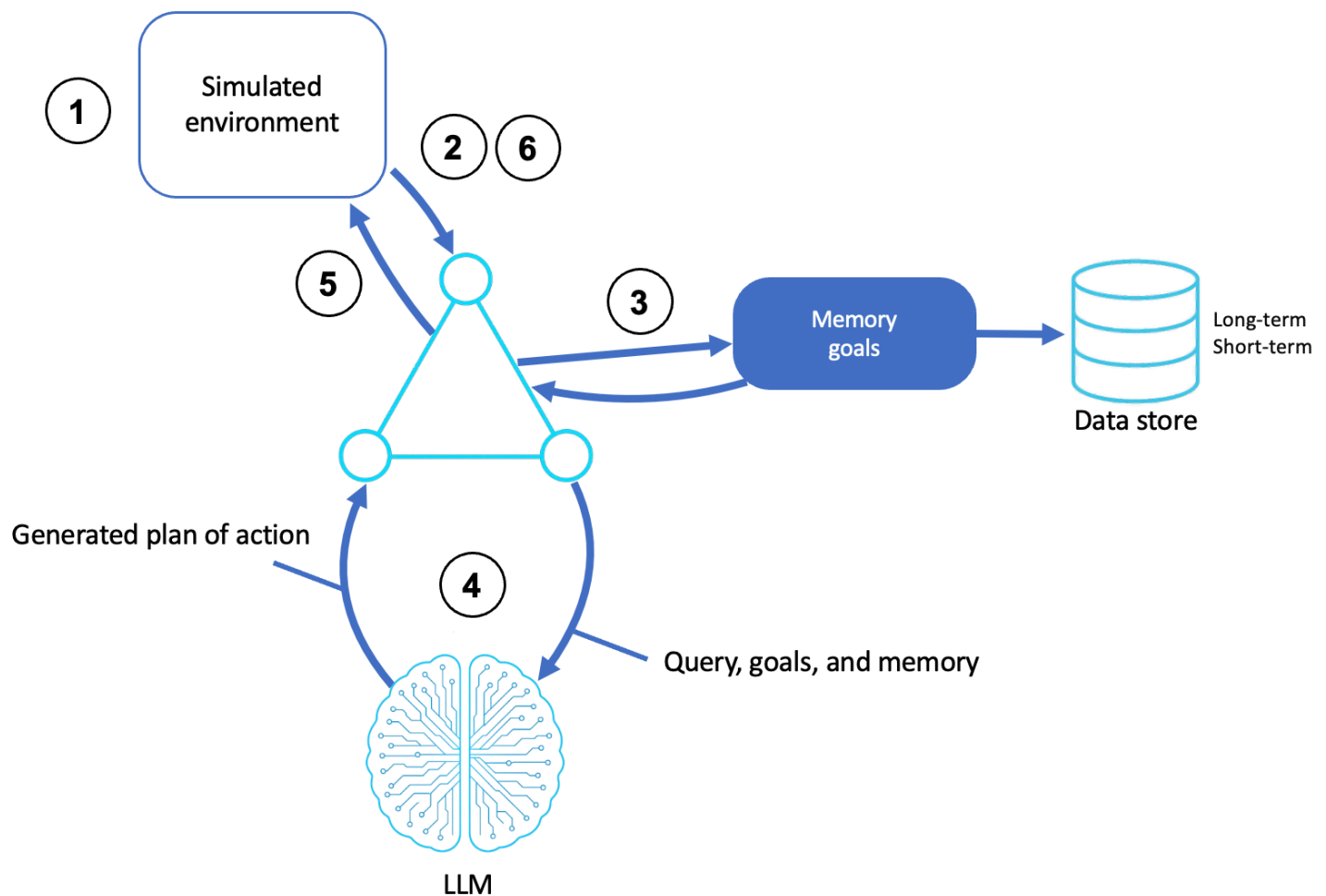
Agen simulasi dan test-bed

Agen simulasi dan test-bed beroperasi dalam lingkungan virtual atau terkontrol di mana mereka bernalar, bertindak, dan belajar. Agen ini mensimulasikan perilaku, memodelkan hasil, dan melatih strategi dalam pengaturan yang dapat diulang sebelum menerapkannya ke lingkungan dunia nyata.

Pola ini berguna untuk pengembangan berulang, pembelajaran penguatan (RL), evaluasi pengambilan keputusan otonom, dan pengujian perilaku yang muncul. Agen simulasi sering beroperasi dalam loop tertutup, menerima umpan balik dari lingkungan mereka dan menyesuaikan perilaku mereka sesuai, menjadikannya penting untuk tugas-tugas yang melibatkan penalaran spasial, kontrol waktu nyata, atau dinamika sistem yang kompleks.

Arsitektur

Diagram berikut menunjukkan simulasi atau agen test-bed:



Deskripsi

1. Memulai lingkungan

- Agen memulai lingkungan simulasi (misalnya, dunia 3D, mesin fisika, kotak pasir CLI, atau aliran data sintesis).
- Agen dimuat ke lingkungan dengan tugas awal, tujuan, atau kebijakan.

2. Merasakan agen

- Agen merasakan keadaan saat ini melalui telemetri simulasi (misalnya, emulasi sensor, kamera virtual, dan log terstruktur).

3. Mengambil tujuan dan memori

- Agen mengambil tujuan yang ditetapkan, instruksi skenario, atau tujuan kontekstual.
- Ini juga dapat mengambil memori sebelumnya, termasuk yang berikut:
 - Long-term strategi atau kebijakan

- Peta lingkungan atau kendala yang diketahui
- Keberhasilan atau kegagalan masa lalu dari simulasi serupa

4. Alasan dan rencana

- LLM menafsirkan keadaan simulasi, tujuan tugas, dan pengetahuan yang dipelajari.
- Ini menghasilkan rencana tindakan atau perintah kontrol.

5. Menjalankan tindakan simulasi

- Agen mengeksekusi rencana, memodifikasi status, menavigasi ruang, atau berinteraksi dengan entitas virtual.

6. Belajar

- Agen mengevaluasi hasil tindakan
- Tergantung pada konfigurasi agen, mungkin melakukan hal berikut:
 - Lakukan RL
 - Catat hasil untuk fine-tuning di masa mendatang
 - Sesuaikan strategi secara real time

Kemampuan

- Beroperasi dalam lingkungan sintetis atau virtual
- Mendukung pembelajaran coba-coba, penyempurnaan kebijakan, dan pemodelan sistem
- Low-risk pengujian untuk perilaku, penanganan kegagalan, dan kasus tepi
- Mengaktifkan analisis perilaku agen yang muncul dalam pengaturan multi-agen
- Mendukung kontrol loop tertutup dan eksplorasi manusia-in-the-loop

Kasus penggunaan umum

- Pembelajaran penguatan untuk robotika, drone, dan game
- Pelatihan kendaraan otonom di jalan virtual
- Simulasi UI atau CLI untuk DevOps skenario test-bed
- Eksperimen perilaku yang muncul dalam simulasi sosial
- Validasi keamanan logika keputusan sebelum produksi

Panduan implementasi

Anda dapat membuat simulasi dan agen test-bed menggunakan alat-alat berikut dan: Layanan AWS

Komponen	Layanan AWS	Tujuan
Lingkungan	Amazon ECS, Amazon EC2, atau simulator khusus di lab studio Amazon SageMaker	Jalankan dunia virtual (Gazebo, Unity, Unreal) atau CLI kotak pasir
Logika agen	Amazon Bedrock, Amazon SageMaker, atau AWS Lambda	LLM-based perencana atau agen RL
Lingkaran umpan balik	Pembelajaran SageMaker penguatan Amazon, Amazon CloudWatch, atau log khusus	Pelacakan hadiah, penilaian hasil, dan pencatatan perilaku
Memori dan replay	Amazon S3, Amazon DynamoDB, atau Amazon RDS	Status persisten, riwayat episode, atau data skenario
Visualisasi	CloudWatch Dasbor Amazon atau notebook Amazon SageMaker	Amati perubahan kebijakan, hasil, dan metrik pelatihan

Berikut ini adalah aplikasi tambahan:

- [AWS SimSpace Weaver](#) untuk simulasi spasial skala besar
- [AWS IoT Core](#) untuk menguji perangkat bayangan
- [SageMaker Eksperimen Amazon](#) untuk evaluasi dan perbandingan agen

Ringkasan

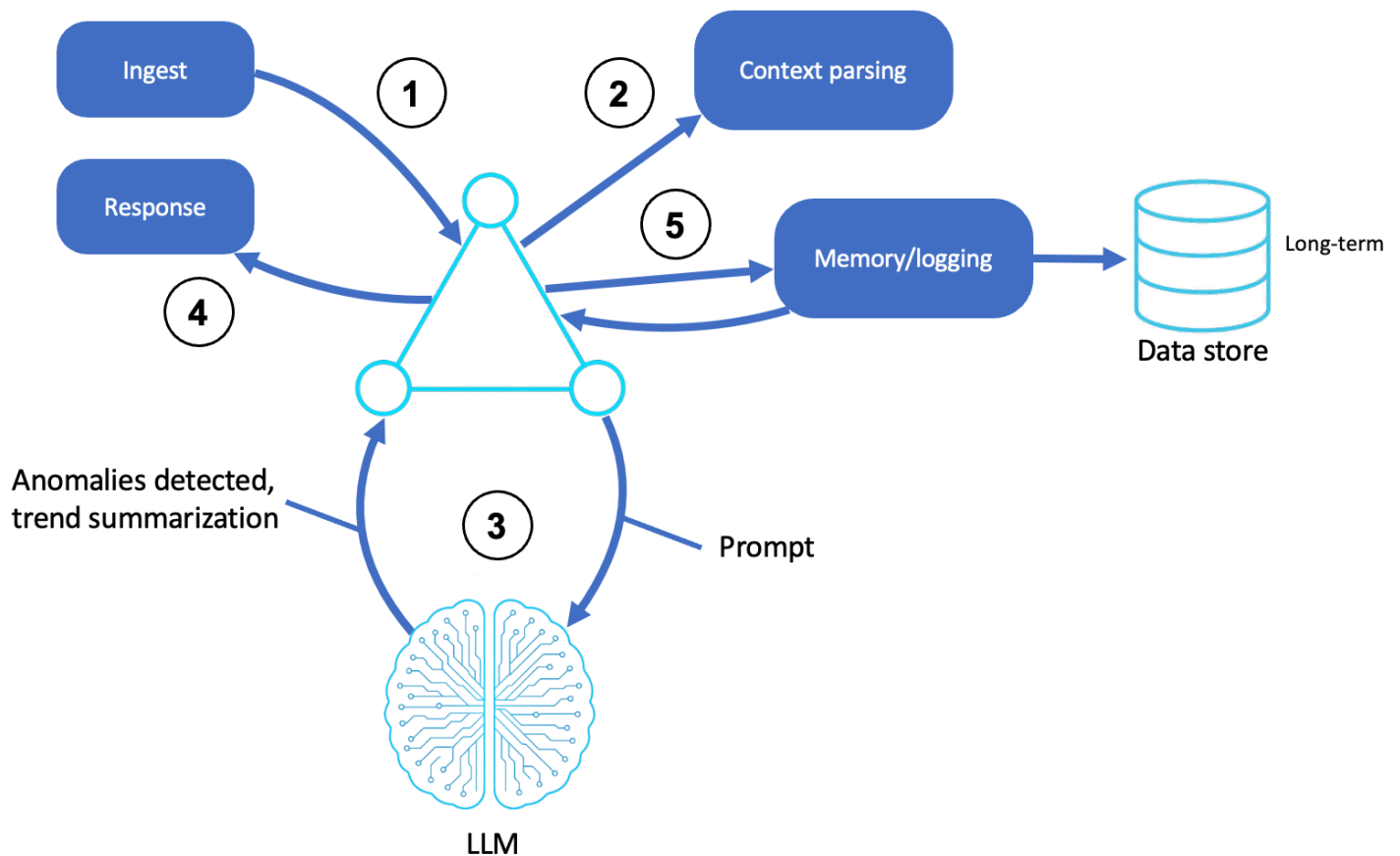
Agan simulasi dan test-bed adalah untuk eksplorasi terstruktur sebelum dikerahkan ke sistem produksi. Gunakan agen ini untuk melatih kebijakan navigasi otonom, menguji proses bisnis di lingkungan sintesis, dan mengevaluasi kawanan untuk pola koordinasi.

Pengamat dan agen pemantauan

Pengamat dan agen pemantauan secara pasif mengamati sistem, lingkungan, dan interaksi untuk mendeteksi pola, menghasilkan wawasan, dan memicu tindakan. Sebagai pengamat cerdas, mereka meningkatkan peringatan, diagnostik, dan audit tanpa langsung memulai perilaku.

Agen-agen ini unggul di mana pemantauan tradisional tidak memiliki kemampuan beradaptasi atau penalaran, terutama untuk AI-in-the-loop pemantauan, deteksi anomali, pengawasan kepatuhan, dan intelijen keamanan. Agen pengamat adalah pendengar acara yang terus memantau telemetry sistem dan interaksi pengguna. Agen tergantung pada persepsi, interpretasi, dan eskalasi atau pelaporan bersyarat.

Arsitektur



Deskripsi

1. Telemetri telemetri

- Agen menerima masukan dari satu atau lebih sumber sistem, seperti berikut ini:

- Log (aplikasi, infrastruktur, keamanan)
- Metrik (kinerja, latensi, penggunaan)
- Acara (panggilan API, tindakan pengguna, data sensor)

2. Mengurai konteks

- Input mentah diuraikan, terstruktur, dan diperkaya dengan metadata, seperti stempel waktu, identitas aktor, status sistem, dan ID jejak.

3. Alasan menggunakan LLM

- Agen menggunakan LLM atau modul logika untuk menafsirkan input yang diurai dengan mengidentifikasi anomali, meringkas tren, dan berkorelasi di seluruh jejak terdistribusi atau jendela waktu.

4. Klasifikasi atau waspada

- Agen menentukan apakah perilaku yang diamati menjamin hal-hal berikut:
 - Peringatan atau eskalasi
 - Laporan atau pembaruan dasbor
 - Pemicu respons (misalnya, remediasi otomatis dan penegakan kebijakan)

5. Memori log atau loop umpan balik

- Agen menyimpan acara dan keputusan untuk pembelajaran jangka panjang, audit, atau referensi future untuk agen lain.

Kemampuan

- Pasif dan noninvasif (agen tidak langsung bertindak)
- Sangat skalabel dan asinkron
- AI-driven korelasi di seluruh sinyal berisik atau terdistribusi
- Mendukung audit, kepatuhan, dan wawasan waktu nyata
- Dapat memberi makan agen hilir atau alur kerja manusia

Kasus penggunaan umum

- AI-augmented observabilitas untuk layanan mikro dan API
- Pemantauan penyimpangan model, pelanggaran kebijakan, atau perilaku out-of-band
- Analisis aktivitas pelanggan atau ringkasan interaksi

- Agen peninjau kode yang memantau komit atau penerapan
- Pemantauan log keamanan atau kepatuhan menggunakan penalaran LLM

Panduan implementasi

Anda dapat membangun pengamat dan agen pemantauan menggunakan alat-alat berikut dan Layanan AWS:

Komponen	Layanan AWS	Tujuan
Konsumsi acara	Amazon EventBridge, CloudWatch Log Amazon, Amazon Kinesis, Amazon S3	Menelan telemetry terstruktur dan tidak terstruktur
Pemrosesan awal	AWS Lambda, AWS Glue, AWS Step Functions	Ubah data mentah menjadi petunjuk terstruktur
Mesin penalaran	Batuan Dasar Amazon, Amazon SageMaker, AWS Lambda	Menganalisis peristiwa, mengklasifikasikan perilaku, menghasilkan wawasan
Penyimpanan dan memori	Amazon S3, Amazon DynamoDB, OpenSearch	Pengamatan, ringkasan, dan output yang terus-menerus
Peringatan dan eskalasi	Amazon SNS, AWS AppFabric Amazon EventBridge	Memicu sistem atau agen hilir

Berikut ini adalah aplikasi tambahan:

- [AWS Security Hub CSPM](#) untuk pemantauan log keamanan
- [Amazon Quick](#) untuk memvisualisasikan output agen

Ringkasan

Pengamat dan agen pemantauan melacak sistem dan perilaku secara real time. Mereka mendeteksi anomali, mengaudit keamanan, dan mengumpulkan intelijen operasi dengan mengidentifikasi pola

yang mungkin diabaikan oleh manusia atau aturan. Kemampuan ini membantu menciptakan sistem yang dapat beradaptasi dengan perubahan kondisi dan membuat keputusan berdasarkan analisis data yang komprehensif.

Multi-agent kolaborasi

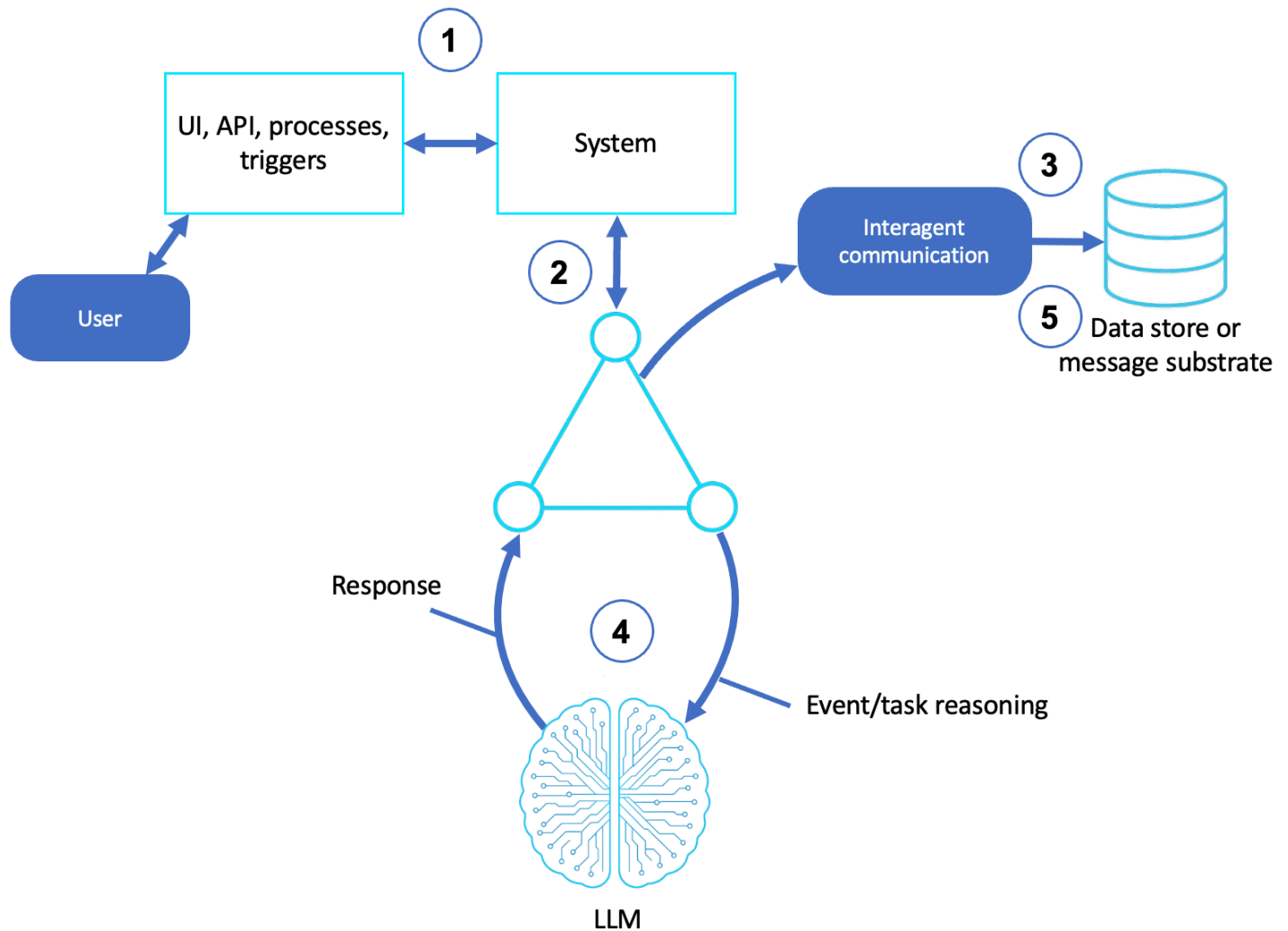
Multi-agent Kolaborasi mengacu pada pola di mana beberapa agen otonom, masing-masing dengan peran, spesialisasi, atau tujuan yang berbeda, bernegosiasi untuk menyelesaikan tugas-tugas yang kompleks. Agen ini dapat beroperasi secara independen atau dengan agen lain dengan berbagi informasi, membagi tanggung jawab, dan secara kolektif beralasan menuju suatu tujuan.

Pola ini berbeda dari agen alur kerja, yang secara terpusat mengoordinasikan dan mendelegasikan tugas ke agen bawahan dalam alur terstruktur. Sebaliknya, kolaborasi multi-agen menekankan koordinasi peer-to-peer atau emergent dengan memungkinkan adaptivitas, paralelisme, dan pembagian kognisi. Tabel berikut membandingkan kolaborasi multi-agen dengan agen alur kerja:

Fitur	Agen alur kerja	Tujuan
Pengendalian	Koordinator terpusat	Rekan-rekan yang terdesentralisasi, didistribusikan, atau berbasis peran
Interaksi	Satu agen mendelegasikan dan melacak eksekusi	Beberapa agen bernegosiasi, berbagi, dan beradaptasi
Desain	Urutan tugas yang telah ditentukan	Distribusi tugas yang muncul dan fleksibel
Koordinasi	Orkestrasi prosedural	Interaksi kooperatif atau kompetitif
Kasus penggunaan	Otomatisasi proses perusahaan	Penalaran kompleks, eksplorasi, dan strategi yang muncul

Arsitektur

Diagram berikut menunjukkan kolaborasi multi-agen:



Deskripsi

1. Memulai tugas

- Pengguna atau sistem memancarkan tujuan atau masalah tingkat tinggi.
- Agen “manajer” atau konteks awal mendefinisikan tujuan.

2. Menetapkan atau menemukan peran

- Agen menugaskan sendiri (logika simbolis atau penalaran) atau didelegasikan (broker acara) ke peran lain, seperti perencana, peneliti, pelaksana, kritikus, atau penjelasan.

3. Berkomunikasi dengan agen lain

- Agen berkomunikasi melalui memori bersama, antrian pesan, atau rantai cepat.
- Mereka mungkin berdebat, menanyakan, atau mengusulkan subtask satu sama lain.

4. Menggunakan penalaran khusus

- Setiap agen menggunakan model atau logika domainnya sendiri untuk memecahkan bagian masalahnya.
- Agen dapat menggunakan LLM dengan petunjuk dan memori khusus peran.

5. Mengkoordinasikan output atau tujuan

- Agen mensintesis kontribusi menjadi jawaban akhir, rencana, atau tindakan.
- (Opsional) Agen pengawas dapat memvalidasi atau meringkas output yang disintesis.

Kemampuan

- Peer-level agen dengan peran atau keterampilan khusus
- Perilaku yang muncul melalui komunikasi atau negosiasi
- Pemrosesan paralel masalah kompleks atau multifaset
- Mendukung musyawarah, koreksi diri, dan iterasi reflektif
- Model dinamika sosial, kolaborasi ilmiah, atau peran tim perusahaan

Kasus penggunaan umum

- Tim peneliti otonom (agen pencarian, peringkas, dan validator)
- Pengembangan perangkat lunak (perencana, pembuat kode, dan penguji)
- Pemodelan skenario bisnis (keuangan, kebijakan, dan kepatuhan)
- Negosiasi, penawaran, atau penalaran multipihak
- Tugas multimodal (gambar, teks, dan logika)

Panduan implementasi

Anda dapat membangun sistem multi-agen menggunakan alat-alat berikut dan Layanan AWS:

Komponen	Layanan AWS	Tujuan
Agan hosting	Batuan Dasar Amazon, Amazon SageMaker, AWS Lambda	Tuan rumah LLM-driven agen individu

Lapisan komunikasi	Amazon SQS, Amazon, EventBridge AWS AppFabric	Pesan dan koordinasi antar agen
Memori bersama	Amazon DynamoDB, Amazon S3, atau OpenSearch	Multi-agent memori atau sistem papan tulis
Lapisan orkestrasi	AWS Step Functions, AWS Lambda jaringan pipa	Logika kickoff, timeout, fallback, dan coba lagi
Identifikasi agen	Agen Amazon Bedrock (ditentukan peran), dan AWS AppConfig Amazon Bedrock converse API (agen di luar Amazon Bedrock)	Role-based alat atau pemanggilan agen dan penegakan batas
Interaksi yang muncul	EventBridge Saluran pipa Amazon atau pendaftar agen	Aktifkan perutean tugas dinamis atau eskalasi

Ringkasan

Multi-agent kolaborasi mendistribusikan tugas pemecahan masalah di seluruh agen modular yang digerakkan oleh peran. Tidak seperti orkestrasi alur kerja, pola kolaborasi menggunakan kecerdasan, ketahanan, dan skalabilitas yang muncul yang mencerminkan bagaimana manusia memecahkan masalah. Ini sangat berharga untuk domain terbuka, tugas kreatif, penalaran multimodal, dan lingkungan yang mendapat manfaat dari beragam perspektif.

Kesimpulan

Pola yang dibahas sebelumnya menggambarkan pendekatan dasar untuk implementasi AI agen di dunia nyata. Dari penalaran dasar hingga kecerdasan yang ditambah memori, setiap pola dikonfigurasi secara unik untuk persepsi, kognisi, dan tindakan yang didasarkan pada otonomi, asinkron, dan agensi.

Pola-pola ini berbagi kosakata dan cetak biru teknis untuk membangun sistem yang cerdas dan diarahkan pada tujuan. Apakah pola disematkan dalam antarmuka pengguna, diatur melalui layanan cloud, atau dikoordinasikan di seluruh tim agen, setiap pola dapat disesuaikan dan modular.

Takeaways

- Pola agen dapat dikomposisi — Sebagian besar agen dunia nyata memadukan dua atau lebih pola (misalnya, agen suara dengan penalaran dan memori berbasis alat).
- Desain agen bersifat kontekstual — Pilih pola berdasarkan permukaan interaksi, kompleksitas tugas, toleransi latensi, dan kendala khusus domain.
- AWSimplementasi asli dapat dicapai — Dengan Amazon Bedrock SageMaker, Amazon,, AWS Lambda, dan arsitektur berbasis peristiwa AWS Step Functions, setiap pola agen dapat dikirimkan dalam skala besar.

Alur kerja LLM

Dalam pola agen, kami mengeksplorasi pola agen AI yang umum, masing-masing dibangun di sekitar serangkaian kemampuan modular: persepsi, tindakan, pembelajaran, dan kognisi. Inti dari modul kognitif dalam banyak pola agen adalah model bahasa besar (LLM) yang mampu penalaran, perencanaan, dan pengambilan keputusan. Namun, menerapkan LLM saja tidak cukup untuk menghasilkan perilaku yang cerdas dan diarahkan pada tujuan.

Untuk melakukan tugas kompleks dengan andal, agen harus menanamkan LLM dalam alur kerja terstruktur, di mana kemampuan model ditambah dengan alat, memori, loop perencanaan, dan logika koordinasi. Alur kerja LLM ini memungkinkan agen untuk memecah tujuan, merutekan subtugas, memanggil layanan eksternal, merefleksikan hasil, dan berkoordinasi dengan agen lain.

Bab ini memperkenalkan pola desain inti untuk membangun modul kognitif berbasis LLM yang kuat, dapat diperluas, dan cerdas, yang diatur di sekitar alur kerja yang dapat digunakan kembali.

Dalam bagian ini

- [Ikhtisar kognisi yang diperkuat LLM](#)
- [Alur kerja untuk rantai cepat](#)
- [Alur kerja untuk perutean](#)
- [Alur kerja untuk paralelisasi](#)
- [Alur kerja untuk orkestrasi](#)
- [Alur kerja untuk evaluator dan loop refleksi-refine](#)
- [Kesimpulan](#)

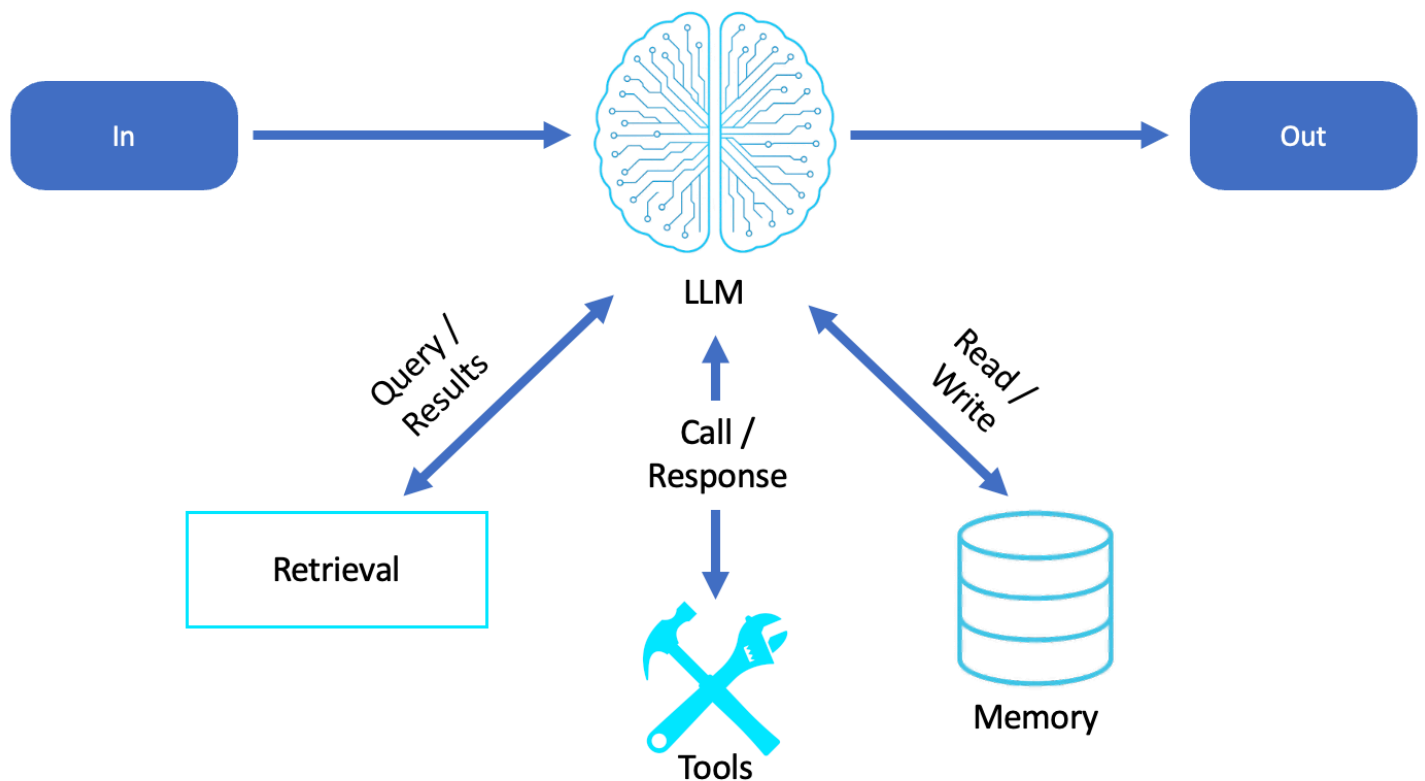
Ikhtisar kognisi yang diperkuat LLM

Pada intinya, modul kognitif agen perangkat lunak dapat dilihat sebagai LLM yang dibungkus dengan augmentasi. Agen dapat menggunakan blok bangunan berikut untuk bernalar secara efektif dalam lingkungannya:

- **Meminta** — Membingkai input menggunakan konteks, instruksi, contoh, dan memori
- **Pengambilan** — Memberikan up-to-date atau pengetahuan khusus domain ke prompt LLM melalui pencarian vektor atau memori semantik, misalnya, melalui retrieval-augmented generation (RAG)

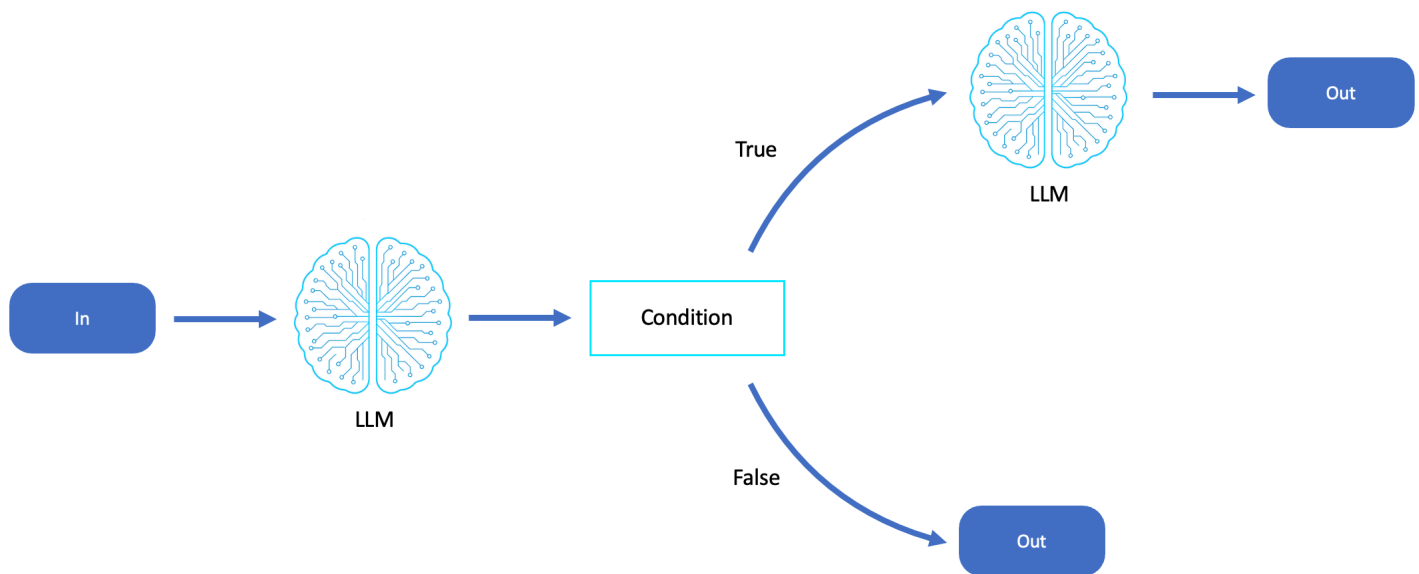
- Penggunaan alat - Mengaktifkan LLM untuk memanggil APIs atau memanggil fungsi untuk mengambil atau bertindak berdasarkan informasi
- Memori — Menggabungkan keadaan persisten atau berbasis sesi ke dalam loop penalaran, baik dengan menggunakan database terstruktur atau ringkasan kontekstual

Augmentasi ini terdiri dari alur kerja yang menentukan bagaimana LLM digunakan dari waktu ke waktu dan di seluruh tugas, mengubahnya dari mesin stateless menjadi agen penalaran dinamis.



Alur kerja untuk rantai cepat

Prompt chaining menguraikan tugas-tugas kompleks menjadi urutan langkah-langkah, di mana setiap langkah adalah pemanggilan LLM diskrit yang memproses atau membangun di atas output dari yang sebelumnya.



Alur kerja rantai prompt cocok untuk skenario di mana tugas dapat secara logis dibagi menjadi langkah-langkah penalaran berurutan, dan di mana output perantara menginformasikan tahap berikutnya. Ini unggul dalam alur kerja yang membutuhkan pemikiran terstruktur, transformasi progresif, atau analisis berlapis, seperti tinjauan dokumen, pembuatan kode, ekstraksi pengetahuan, dan penyempurnaan konten.

Deskripsi

- Kompleksitas tugas melebihi jendela konteks atau kedalaman penalaran dari satu panggilan LLM.
- Output dari satu langkah (misalnya, analisis, ringkasan, atau perencanaan) menjadi masukan untuk keputusan tindak lanjut atau fase generasi.
- Anda memerlukan transparansi dan kontrol di seluruh tahap penalaran (misalnya, hasil antara yang dapat diaudit).
- Anda ingin memasukkan validasi eksternal, pemfilteran, atau logika pengayaan di antara langkah-langkah.
- Ini ideal untuk agen yang beroperasi dalam loop penalaran gaya pipa, seperti agen penelitian, asisten editorial, sistem perencanaan, dan kopilot multistage.

Kemampuan

- Rantai linier atau bercabang dari panggilan LLM
- Hasil antara diteruskan sebagai input terstruktur atau disematkan ke dalam petunjuk tindak lanjut

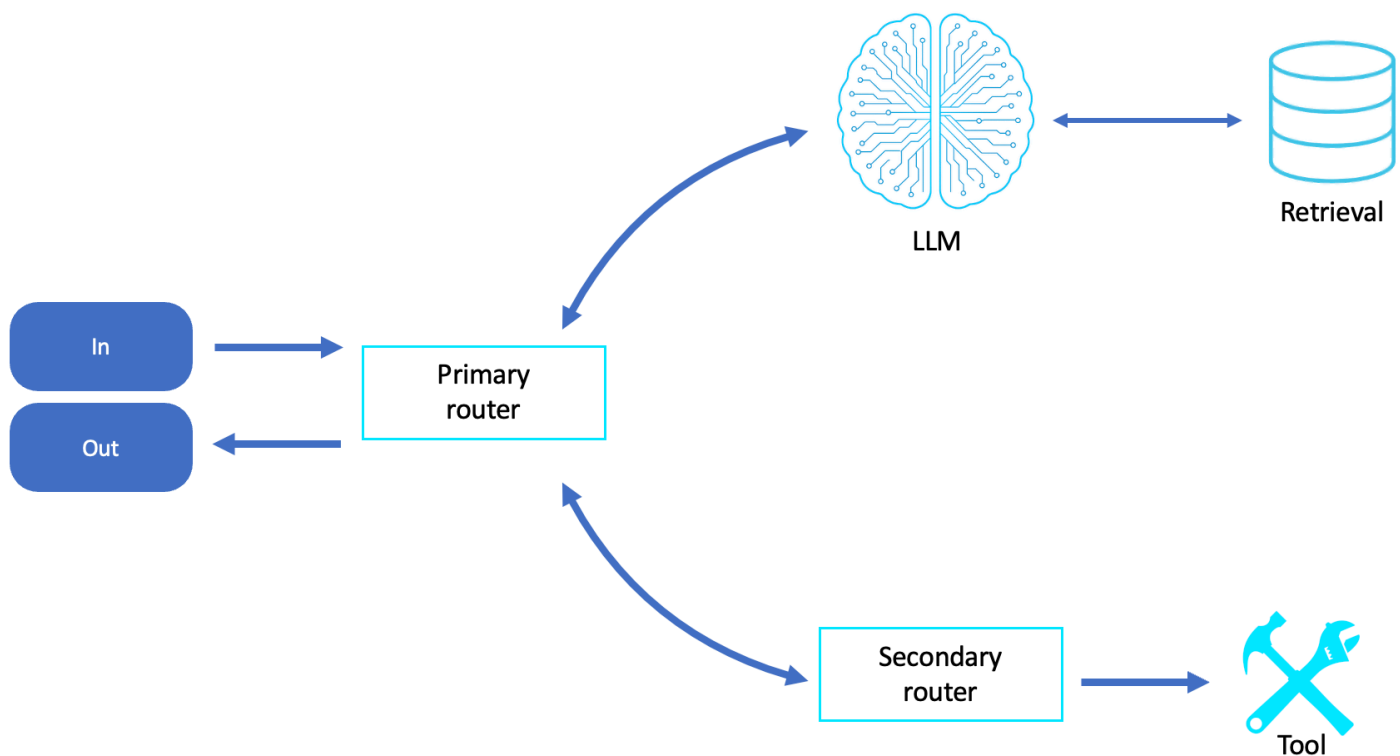
- Dapat diatur dengan,, atau pelari AWS Step Functions khusus AWS Lambda agen

Kasus penggunaan umum

- Tugas penalaran multistep (misalnya, “meringkas penulisan ulang kritik”)
- Asisten peneliti mensintesis keluaran berlapis (misalnya, “pertanyaan jawaban fakta ekstrak pencarian”)
- Pipa pembuatan kode (“hasilkan rencana tulis kode uji kode jelaskan output”)

Alur kerja untuk perutean

Dalam pola perutean, pengklasifikasi atau agen router menggunakan LLM untuk menafsirkan maksud atau kategori kueri, lalu merutekan input ke tugas atau agen hilir khusus.



Alur kerja perutean digunakan dalam skenario di mana agen harus mengklasifikasikan intent input, tipe tugas, atau domain dengan cepat, lalu mendelegasikan permintaan ke subagent, alat, atau alur kerja khusus. Ini sangat berguna dalam agen kapabilitas, seperti yang berfungsi sebagai asisten umum, pintu depan ke fungsi perusahaan, atau antarmuka AI yang menghadap pengguna yang menjangkau domain.

Routing sangat efektif ketika:

- Triaging permintaan di berbagai tugas (misalnya, pencarian, ringkasan, pemesanan, perhitungan).
- Input harus diproses sebelumnya atau dinormalisasi sebelum memasuki alur kerja yang lebih khusus.
- Jenis input yang berbeda (misalnya, gambar vs. teks, kueri terstruktur vs. tidak terstruktur) memerlukan penanganan khusus.
- Agen bertindak sebagai switchboard percakapan, mendelegasikan tugas ke agen khusus atau layanan mikro.
- Alur kerja ini umum terjadi pada kopilot khusus domain, bot dukungan pelanggan, router layanan perusahaan, dan agen multimodal, di mana pengiriman cerdas menentukan kualitas dan efisiensi perilaku agen.

Kemampuan

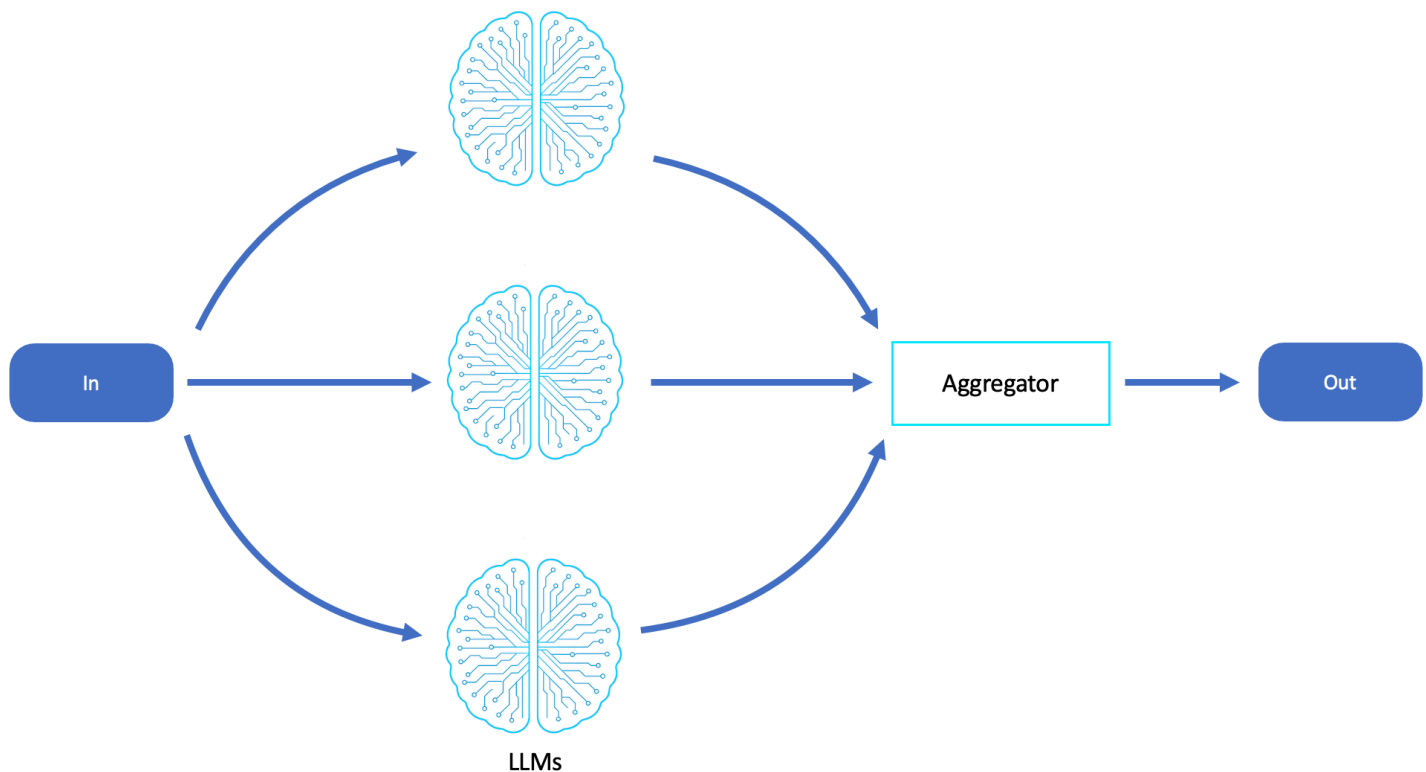
- LLM first-pass bertindak sebagai operator
- Rute dapat memanggil alur kerja yang berbeda atau bahkan pola agen lainnya
- Mendukung perluasan kemampuan modular

Kasus penggunaan umum

- Asisten multidomain (“apakah ini pertanyaan hukum, medis, atau keuangan?”)
- Pohon keputusan ditingkatkan dengan penalaran LLM
- Pemilihan alat dinamis (misalnya, pencarian vs. pembuatan kode)

Alur kerja untuk paralelisasi

Alur kerja ini melibatkan pemecahan tugas menjadi subtugas independen yang dapat ditangani secara bersamaan oleh beberapa panggilan atau agen LLM. Output kemudian digabungkan secara terprogram dan disintesis menjadi hasil.



Alur kerja Paralelisasi digunakan ketika tugas dapat dibagi menjadi subtask independen dan nonsekuensial yang dapat diproses secara bersamaan, secara signifikan meningkatkan efisiensi, throughput, dan skalabilitas. Ini sangat kuat dalam ruang masalah yang berat data, berorientasi batch, atau multiperspektif di mana agen harus menganalisis atau menghasilkan konten di beberapa input.

Paralelisasi sangat efektif ketika:

- Subtask tidak bergantung pada hasil antara satu sama lain, memungkinkan mereka berjalan secara paralel tanpa koordinasi.
- Tugas melibatkan pengulangan proses penalaran yang sama di banyak item (misalnya, meringkas beberapa dokumen atau mengevaluasi daftar opsi).
- Berbagai hipotesis atau perspektif dieksplorasi secara paralel untuk mempromosikan keragaman, kreativitas, atau ketahanan.
- Anda perlu mengurangi latensi untuk permintaan volume tinggi atau frekuensi tinggi melalui eksekusi LLM bersamaan.
- Alur kerja ini biasanya digunakan dalam agen pengolah dokumen, mesin survei atau perbandingan, ringkasan batch, brainstormer multi-agen, dan tugas klasifikasi atau pelabelan yang dapat diskalakan, terutama di mana penalaran paralel yang cepat merupakan keunggulan kinerja.

Kemampuan

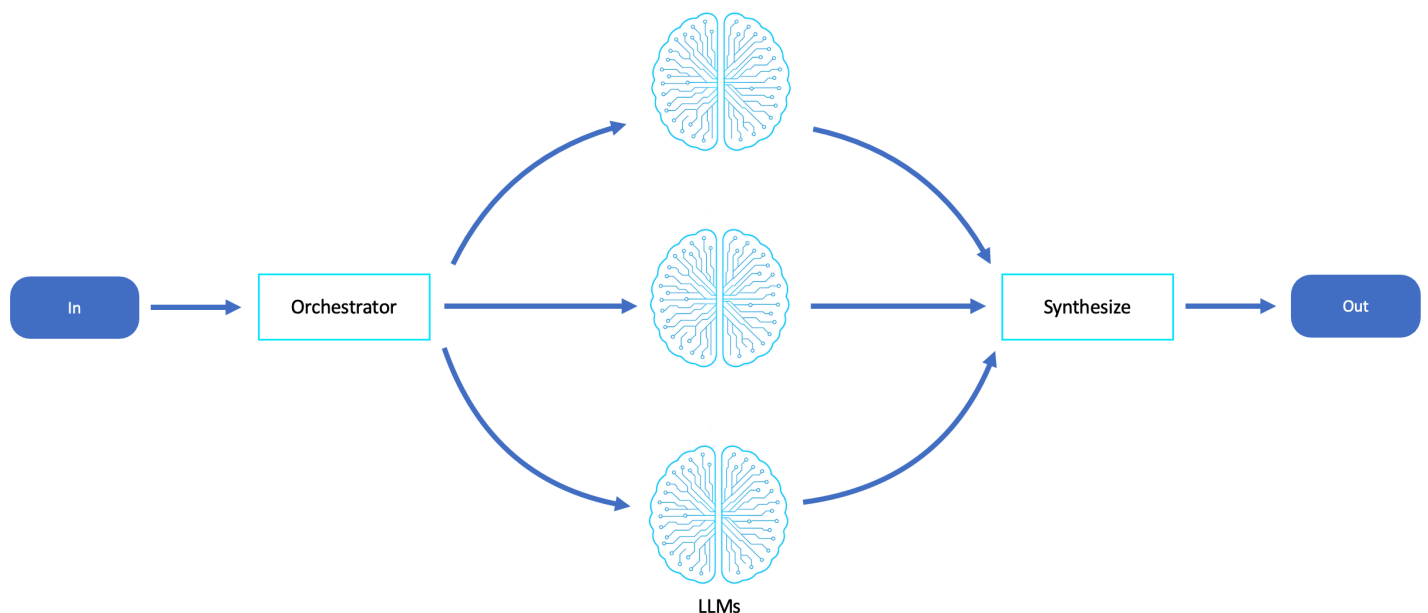
- Eksekusi paralel tugas LLM (dengan menggunakan AWS Lambda, AWS Fargate, atau status AWS Step Functions peta)
- Membutuhkan penyelarasan hasil, validasi, atau deduplikasi pada tahap sintesis
- Sangat cocok untuk loop agen stateless

Kasus penggunaan umum

- Menganalisis berbagai dokumen atau perspektif secara paralel
- Menghasilkan beragam konsep, ringkasan, atau rencana
- Mempercepat throughput di seluruh pekerjaan batch

Alur kerja untuk orkestrasi

Agen orkestrator pusat menggunakan LLM untuk merencanakan, menguraikan, dan mendelegasikan subtugas ke agen atau model pekerja khusus, masing-masing dengan peran atau keahlian domain tertentu. Ini mencerminkan struktur tim manusia dan mendukung perilaku yang muncul di berbagai agen.



Alur kerja orkestrasi sangat ideal untuk skenario yang kompleks, hierarkis, atau multidisiplin, yang membutuhkan dekomposisi terstruktur dan eksekusi khusus. Ini sangat cocok untuk tugas-tugas yang

membutuhkan pembagian kerja, di mana subkomponen tugas yang berbeda paling baik ditangani oleh agen dengan kemampuan, pengetahuan, atau perangkat yang berbeda.

Alur kerja ini sangat efektif ketika:

- Tugas dapat dibagi menjadi subtugas yang bervariasi dalam ruang lingkup, jenis, atau penalaran (misalnya, rencana, penelitian, implementasi, dan pengujian).
- LLM atau meta-agen harus mengoordinasikan agen lain, memantau kemajuan, dan mensintesis hasil.
- Anda ingin memodulasi tanggung jawab agen, memungkinkan skalabilitas, penggunaan kembali, dan penyetelan khusus.
- Sistem ini membutuhkan perilaku berbasis peran, meniru bagaimana tim manusia (misalnya, manajer proyek, pengembang, dan pengulas) beroperasi dalam kolaborasi.

Orkestrasi sangat ideal untuk agen perencanaan multitur, kopilot pengembangan perangkat lunak, agen proses perusahaan, dan pelaksana proyek otonom. Ini sangat berguna ketika menerapkan sistem multi-agen yang memerlukan pemecahan tugas terpusat tetapi logika eksekusi terdistribusi, memungkinkan ekstensibilitas dan perilaku yang lebih dapat dijelaskan di seluruh lapisan agen.

Kemampuan

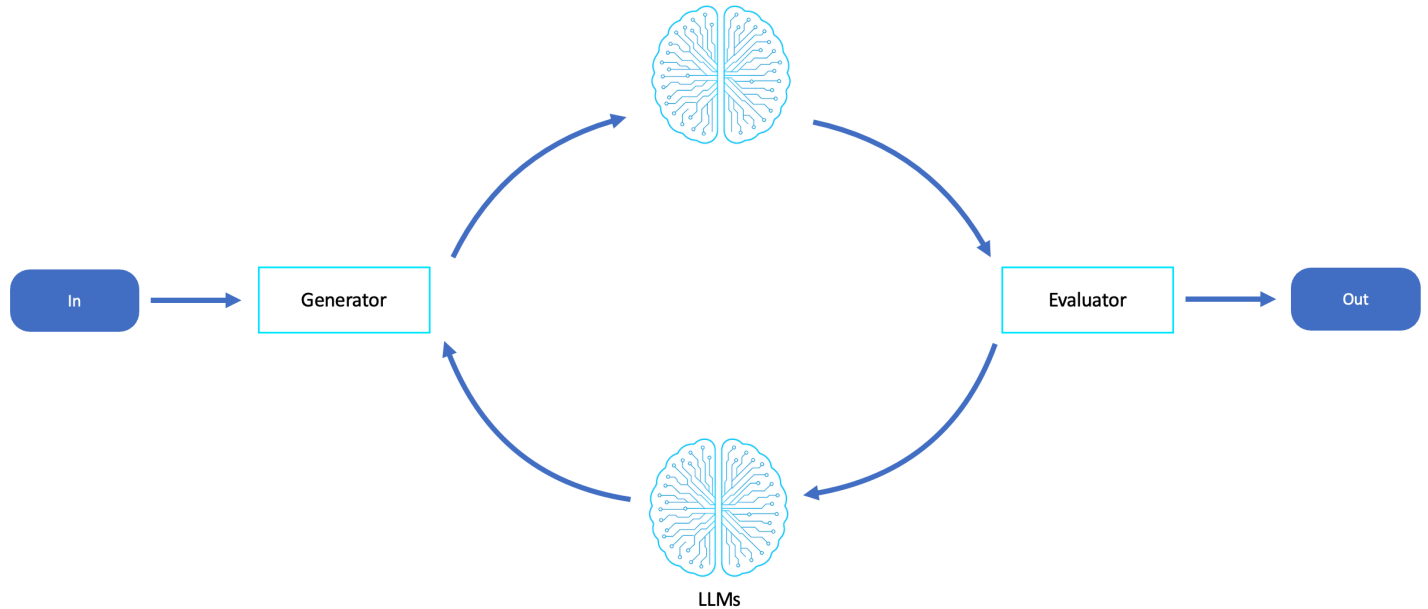
- Orkestrator melakukan meta-penalaran tujuan
- Agen pekerja dapat menyertakan akses alat, memori, atau permintaan khusus domain
- Bisa hierarkis (yaitu, delegasi tugas bertingkat)

Kasus penggunaan umum

- Manajer proyek, peneliti koordinasi, penulis, dan agen jaminan kualitas
- Coding copilot yang menggabungkan perencanaan, pelaksanaan, dan pengujian
- Agen yang mengawasi rantai alat atau pola akses API

Alur kerja untuk evaluator dan loop refleksi-refine

Alur kerja ini menyediakan loop umpan balik di mana satu LLM menghasilkan hasil, dan yang lain mengevaluasi atau mengkritik hasilnya. Ini mempromosikan refleksi diri, optimasi, dan peningkatan berulang.



Alur kerja evaluator sangat ideal untuk skenario di mana kualitas output, akurasi, dan penyelarasan penting dan di mana generasi single-pass tidak dapat diandalkan atau tidak memadai. Alur kerja ini unggul ketika agen harus mengkritik diri sendiri, mengulangi, dan menyempurnakan output mereka—baik untuk memenuhi standar kebenaran yang lebih tinggi atau untuk mengeksplorasi alternatif yang lebih baik berdasarkan umpan balik.

Alur kerja ini sangat efektif ketika:

- Output melibatkan metrik kualitas subjektif (misalnya, gaya, nada, dan keterbacaan) atau kriteria obyektif (misalnya, kebenaran, keamanan, dan kinerja).
- Agen harus bernalar melalui trade-off, mengevaluasi kendala, atau mengoptimalkan menuju tujuan.
- Anda memerlukan redundansi bawaan dan jaminan kualitas, terutama dalam domain yang diatur, berhadapan dengan pelanggan, atau kreatif.
- Human-in-the-loop ulasan mahal atau tidak tersedia, dan validasi otonom diinginkan.

Alur kerja ini digunakan untuk pembuatan konten, sintesis dan peninjauan kode, penegakan kebijakan, pemeriksaan penyelarasan, penyetelan instruksi, dan pasca pemrosesan RAG. Ini juga

berguna untuk agen peningkatan diri, di mana umpan balik berkelanjutan membantu membentuk respons yang lebih baik dari waktu ke waktu untuk membangun loop keputusan otonom yang dapat dipercaya.

Kasus penggunaan umum

- Agen tim merah dibandingkan dengan agen tim biru
- Agen yang menghasilkan, mengevaluasi, dan merevisi kode atau rencana
- Jaminan kualitas, deteksi halusinasi, dan penegakan gaya

Kemampuan

- Mendukung generasi dan evaluasi terpisah menggunakan model yang berbeda (misalnya, Claude untuk generasi dan Mistral untuk evaluasi)
- Umpan balik terstruktur dan digunakan untuk meminta output yang direvisi
- Mendukung beberapa iterasi atau ambang konvergensi

Kesimpulan

LLMs menyediakan inti kognitif agen perangkat lunak modern, tetapi pemanggilan model mentah tidak cukup untuk mencapai kecerdasan yang terarah, kuat, dan terkendali. Untuk beralih dari generasi keluaran ke penalaran terstruktur dan perilaku yang selaras dengan tujuan, LLMs harus disematkan dalam pola alur kerja yang disengaja yang menentukan bagaimana model memproses input, mengelola konteks, dan mengoordinasikan tindakan.

Alur kerja LLM memperkenalkan fondasi untuk membangun modul kognitif agen:

- Rantai cepat memecah penalaran kompleks menjadi langkah-langkah modular yang dapat diaudit.
- Routing memungkinkan klasifikasi tugas cerdas dan delegasi yang ditargetkan.
- Paralelisasi mempercepat throughput dan mempromosikan beragam penalaran.
- Orkestrasi agen menyusun kolaborasi multi-agen melalui dekomposisi tugas dan eksekusi berbasis peran.
- Evaluator (reflekt-refine loop) memungkinkan perbaikan diri, kontrol kualitas, dan pemeriksaan keselarasan.

Setiap alur kerja mewakili pola composable yang dapat disesuaikan dengan kebutuhan agen, kompleksitas tugas, dan harapan pengguna. Alur kerja ini tidak saling eksklusif. Mereka adalah blok bangunan yang sering digabungkan menjadi arsitektur hibrida yang mendukung penalaran dinamis, koordinasi multi-agen, dan keandalan tingkat perusahaan.

Saat Anda beralih ke babak berikutnya tentang pola alur kerja agen, alur kerja LLM ini akan muncul kembali sebagai struktur tertanam dalam sistem yang lebih besar, mendukung delegasi tujuan, orkestrasi alat, loop keputusan, dan otonomi siklus hidup. Menguasai alur kerja LLM ini sangat penting untuk merancang agen perangkat lunak yang tidak hanya memprediksi teks tetapi alasan, beradaptasi, dan bertindak dengan sengaja.

Pola alur kerja agen

Pola alur kerja agen mengintegrasikan agen perangkat lunak modular dengan alur kerja model bahasa besar (LLM) terstruktur, memungkinkan penalaran dan tindakan otonom. Meskipun terinspirasi oleh arsitektur tanpa server dan berbasis peristiwa tradisional, pola-pola ini mengubah logika inti dari kode statis ke agen yang diperkuat LLM, memberikan peningkatan kemampuan beradaptasi dan pengambilan keputusan kontekstual. Evolusi ini mengubah arsitektur cloud konvensional dari sistem deterministik menjadi yang mampu interpretasi dinamis dan augmentasi cerdas, sambil mempertahankan prinsip-prinsip dasar skalabilitas dan responsif.

Dalam bagian ini

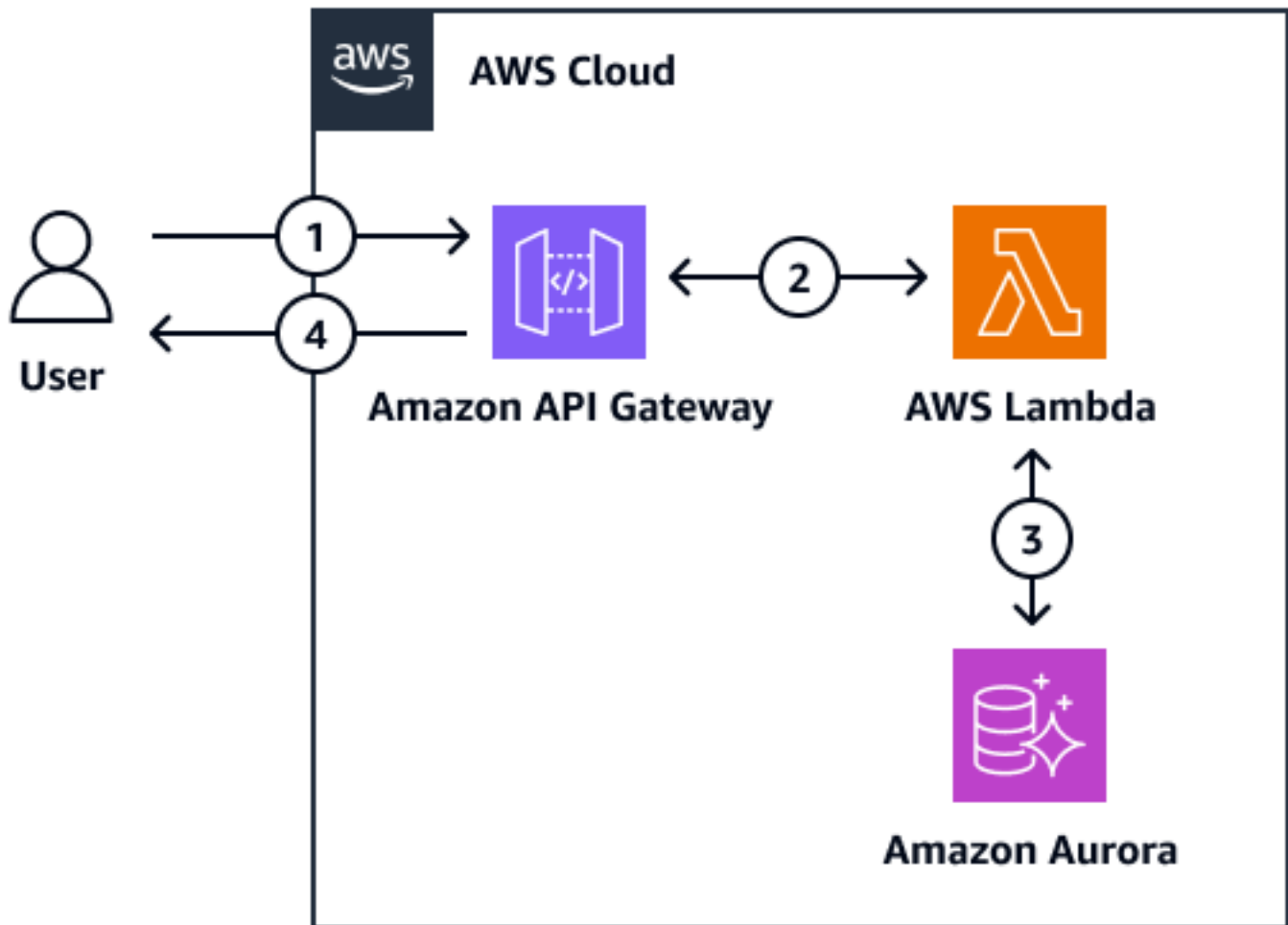
- [Dari sistem yang didorong oleh peristiwa hingga sistem yang ditambah kognisi](#)
- [Pola saga rantai yang cepat](#)
- [Merutekan pola pengiriman dinamis](#)
- [Paralelisasi dan pola scatter-gathering](#)
- [Pola orkestrasi Saga](#)
- [Evaluasi memantulkan pola loop](#)
- [Merancang alur kerja agen pada AWS](#)
- [Kesimpulan](#)

Dari sistem yang didorong oleh peristiwa hingga sistem yang ditambah kognisi

Arsitektur cloud modern, terutama yang dibangun di atas prinsip tanpa server dan berbasis peristiwa, secara tradisional mengandalkan pola seperti perutean, fan-out, dan pengayaan untuk menciptakan sistem yang responsif dan dapat diskalakan. Sistem AI agen dibangun di atas fondasi ini sambil membingkainya kembali di sekitar penalaran yang ditambah LLM dan fleksibilitas kognitif. Pendekatan ini memungkinkan kemampuan pemecahan masalah dan otomatisasi yang lebih canggih, berpotensi merevolusi bagaimana tugas kompleks ditangani di lingkungan cloud.

Arsitektur berbasis peristiwa

Diagram berikut menunjukkan sistem terdistribusi yang khas:

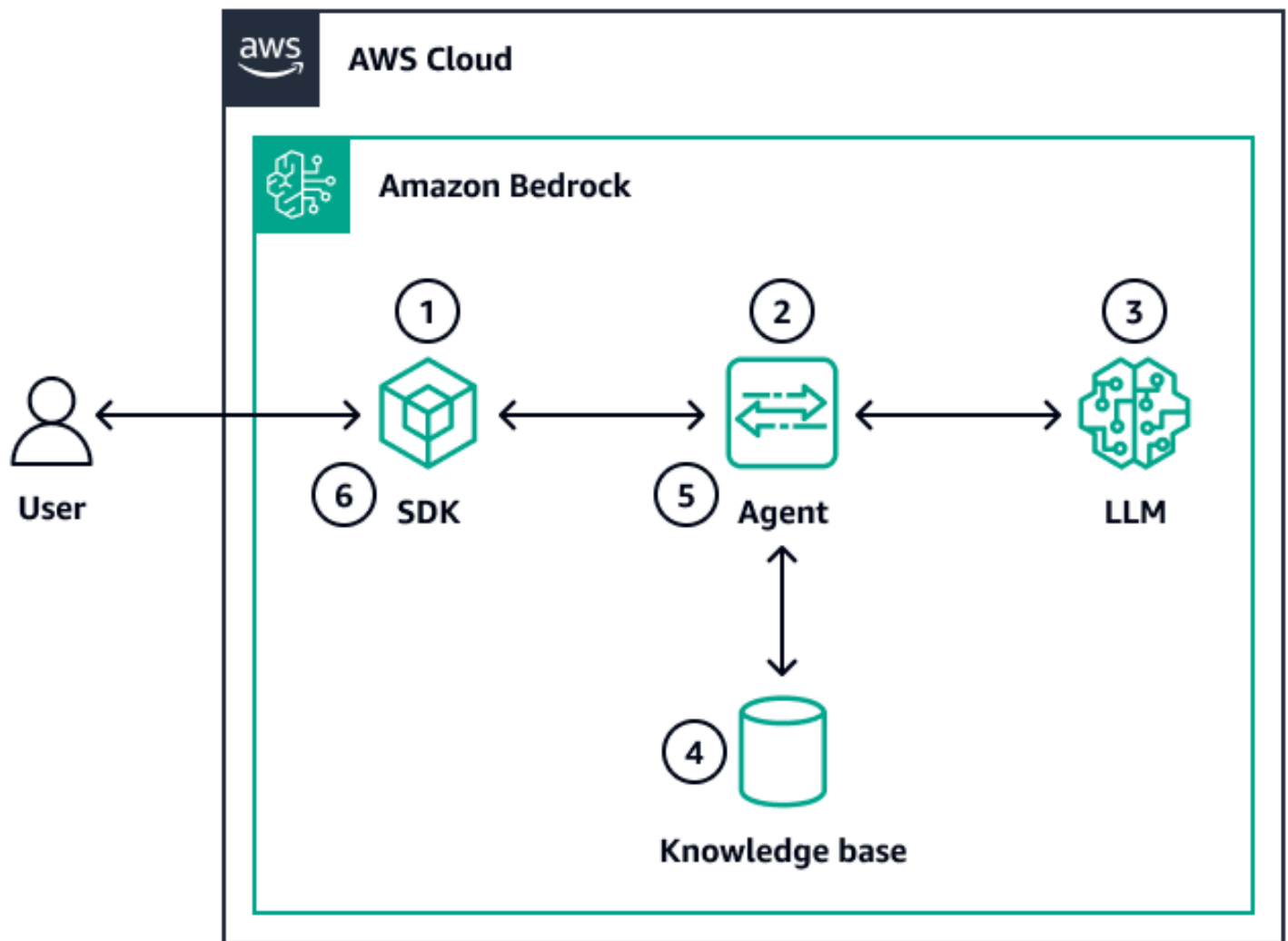


1. Pengguna mengirimkan permintaan ke Amazon API Gateway.
2. Amazon API Gateway merutekan permintaan ke suatu AWS Lambda fungsi.
3. AWS Lambda melakukan pengayaan data dengan menanyakan database Amazon Aurora
4. Amazon API Gateway mengembalikan payload yang diperkaya ke pemanggil.

Struktur ini dapat diandalkan dan dapat diskalakan, tetapi pada dasarnya statis. Aturan bisnis dan jalur logika harus dikodekan secara eksplisit, dan beradaptasi dengan perubahan konteks atau informasi yang tidak lengkap terbatas.

Alur kerja yang ditambah kognisi

Arsitektur agen menambahkan augmentasi kognitif ke sistem yang digerakkan oleh peristiwa. Diagram berikut menunjukkan ekuivalen agen:



1. Pengguna mengirimkan kueri melalui panggilan SDK atau API.
2. Agen Amazon Bedrock menerima kueri.
3. Agen menafsirkan kueri dengan memanggil LLM
4. Agen melakukan pengayaan semantik dengan mencari basis pengetahuan Amazon Bedrock atau sumber data eksternal lainnya.
5. LLM mensintesis respons yang kaya konteks dan selaras dengan tujuan.
6. Sistem mengembalikan respons yang disintesis kepada pengguna.

Dalam alur ini, LLM menggunakan logika, memahami maksud, mengambil dan menggabungkan konteks yang relevan, dan kemudian memutuskan cara terbaik untuk merespons. Pola ini mencerminkan pola pengayaan tradisional, di mana pesan ditambah dengan data eksternal sebelum

diarahkan lebih lanjut. Namun, dalam sistem agen, pengayaan ini bukan pencarian statis. Sebaliknya, pengayaan bersifat dinamis, dibimbing secara semantik, dan didorong oleh tujuan.

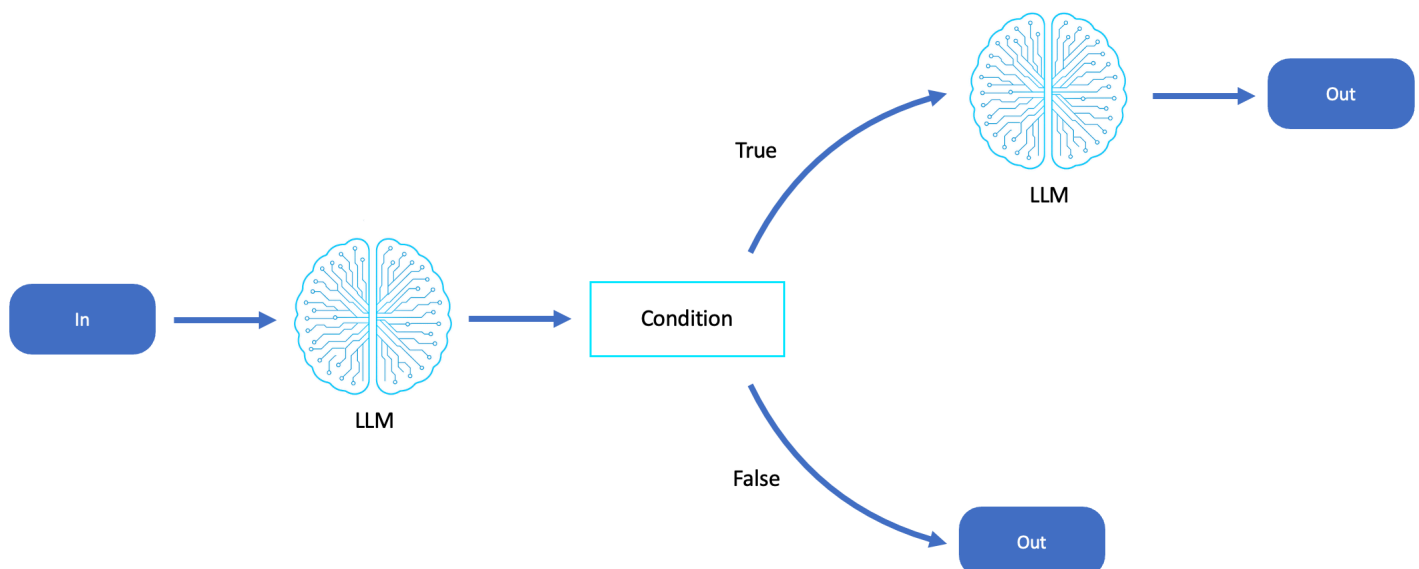
Wawasan inti

Setiap alur kerja LLM dapat dipetakan ke pola alur kerja agen, yang mencerminkan dan mengembangkan gaya arsitektur berbasis peristiwa tradisional. Blok bangunan dasar alur kerja agen adalah kemampuan untuk menambah konteks LLM dengan data, alat, dan memori. Ini menciptakan loop penalaran yang diinformasikan, adaptif, dan selaras dengan maksud pengguna. Di mana sistem tradisional memperkaya pesan dengan data pencarian, sistem agen memungkinkan perangkat lunak untuk bertindak kurang seperti skrip dan lebih seperti kolaborator cerdas.

Pola saga rantai yang cepat

Dengan menata ulang rantai prompt LLM sebagai kisah yang digerakkan oleh peristiwa, kami membuka model operasional baru: alur kerja menjadi terdistribusi, dapat dipulihkan, dan dikoordinasikan secara semantik di seluruh agen otonom. Setiap langkah prompt respons dibingkai ulang sebagai tugas atom, dipancarkan sebagai peristiwa, dikonsumsi oleh agen khusus, dan diperkaya dengan metadata kontekstual.

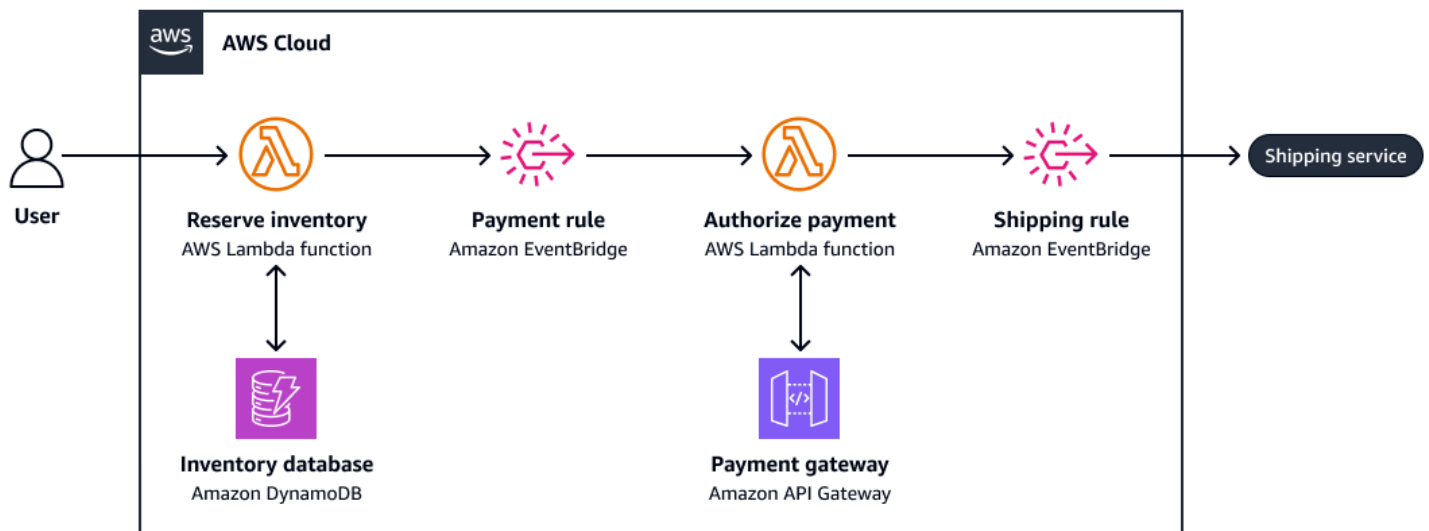
Diagram berikut adalah contoh rantai prompt LLM:



Koreografi Saga

Pola koreografi saga adalah pendekatan implementasi dalam sistem terdistribusi yang tidak memiliki koordinator pusat. Sebagai gantinya, setiap layanan atau komponen menerbitkan peristiwa yang memicu tindakan alur kerja berikutnya. Pola ini banyak digunakan dalam sistem terdistribusi untuk mengelola transaksi di berbagai layanan. Dalam sebuah kisah, sistem menjalankan serangkaian transaksi lokal yang terkoordinasi. Jika salah satu gagal, sistem memicu tindakan kompensasi untuk menjaga konsistensi.

Diagram berikut adalah contoh koreografi saga:



1. Persediaan cadangan
2. Otorisasi pembayaran
3. Buat pesanan pengiriman

Jika langkah 3 gagal, sistem akan meminta tindakan kompensasi (misalnya, membatalkan pembayaran atau merilis inventaris).

Pola ini sangat berharga dalam arsitektur yang digerakkan oleh peristiwa di mana layanan digabungkan secara longgar dan status harus diselesaikan secara konsisten dari waktu ke waktu, bahkan di hadapan kegagalan sebagian.

Pola rantai yang cepat

Prompt chaining menyerupai pola saga baik dalam struktur maupun tujuan. Ini mengeksekusi serangkaian langkah penalaran yang membangun secara berurutan sambil mempertahankan konteks dan memungkinkan untuk rollback dan revisi.

Koreografi agen

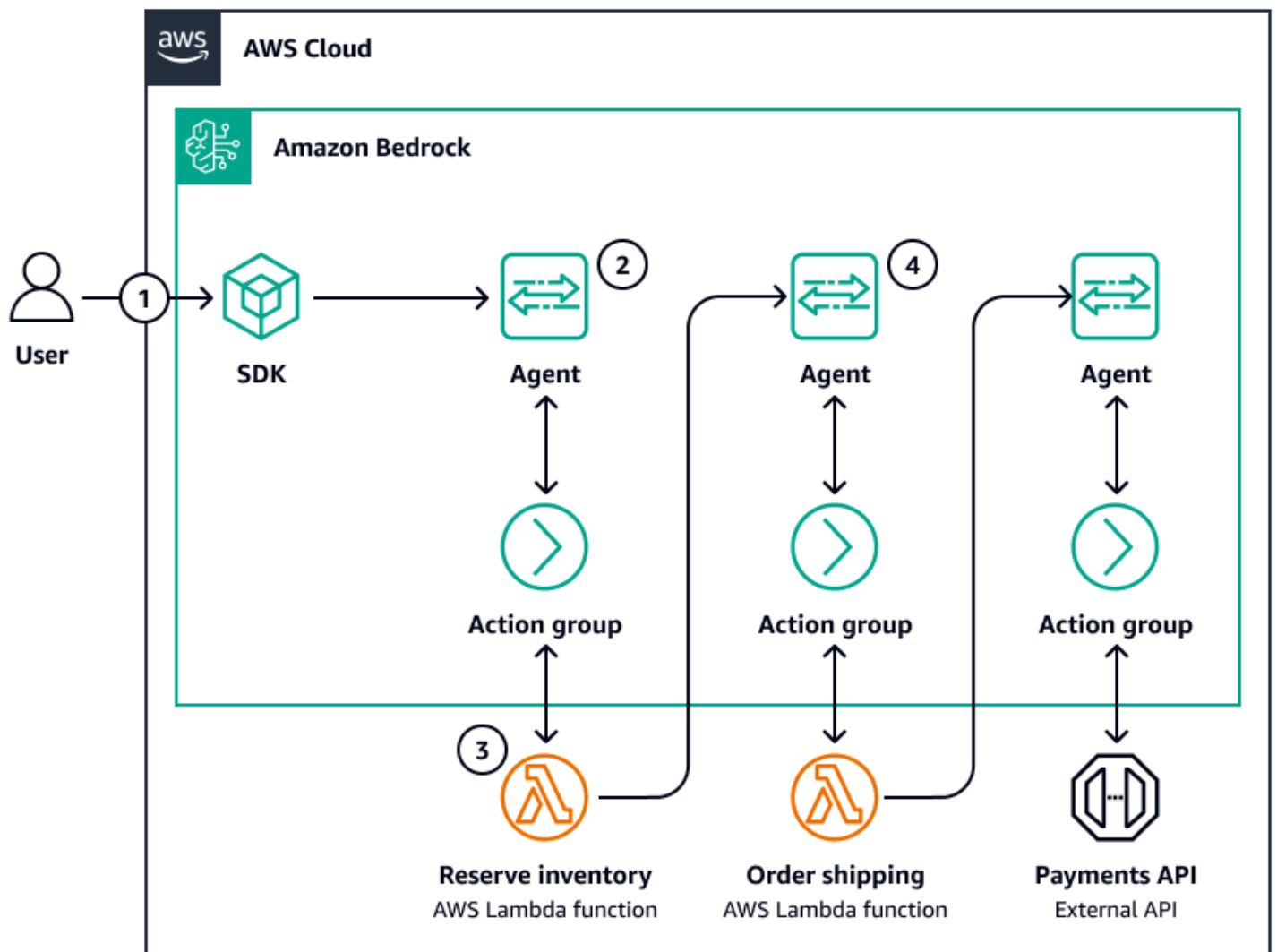
1. LLM menafsirkan kueri pengguna yang kompleks dan menghasilkan hipotesis
2. LLM menguraikan rencana untuk menyelesaikan tugas
3. LLM mengeksekusi subtugas (misalnya, dengan menggunakan panggilan alat atau mengambil pengetahuan)
4. LLM menyempurnakan output atau meninjau kembali langkah sebelumnya jika dianggap hasilnya tidak memuaskan

Jika hasil antara cacat, sistem dapat melakukan salah satu hal berikut:

- Coba lagi langkah-langkahnya menggunakan pendekatan yang berbeda
- Kembalikan ke prompt sebelumnya dan rencanakan ulang
- Gunakan loop evaluator (misalnya, dari pola evaluator-optimizer) untuk mendeteksi dan memperbaiki kegagalan

Seperti pola saga, prompt chaining memungkinkan untuk kemajuan sebagian dan mekanisme rollback. Ini terjadi melalui penyempurnaan berulang dan koreksi yang diarahkan LLM daripada melalui kompensasi transaksi database.

Diagram berikut adalah contoh koreografi agen:



1. Pengguna mengirimkan kueri melalui SDK.
2. Agen Amazon Bedrock mengatur penalaran melalui hal-hal berikut:
 - Interpretasi (LLM)
 - Perencanaan (LLM)
 - Eksekusi melalui alat atau basis pengetahuan
 - Konstruksi respons
3. Jika alat gagal atau mengembalikan data yang tidak mencukupi, agen dapat merencanakan ulang atau menyusun ulang tugas secara dinamis.
4. Memori (misalnya, penyimpanan vektor jangka pendek) dapat mempertahankan statusnya di seluruh langkah

Takeaways

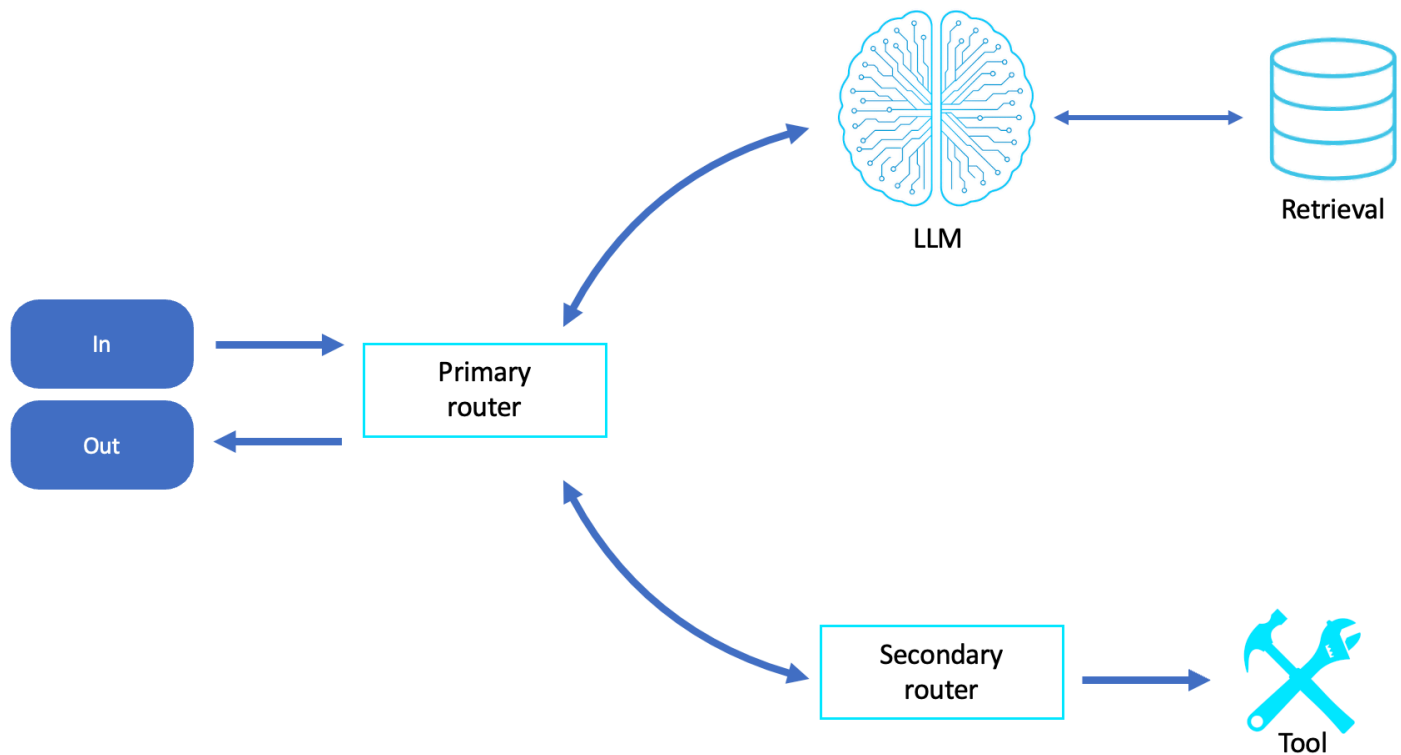
Di mana pola saga mengelola panggilan layanan terdistribusi dengan logika kompensasi, rantai cepat mengelola tugas penalaran dengan pengurutan reflektif dan perencanaan ulang adaptif. Kedua sistem memungkinkan kemajuan tambahan, poin keputusan terdesentralisasi, dan pemulihan kegagalan, dan melakukan semua ini melalui penalaran yang diinformasikan daripada kemunduran yang kaku.

Prompt chaining memperkenalkan penalaran transaksional, yang merupakan padanan kognitif dari saga. Artinya, setiap “pemikiran” dievaluasi kembali, direvisi, atau ditinggalkan sebagai bagian dari dialog yang diarahkan pada tujuan yang lebih luas.

Merutekan pola pengiriman dinamis

Dalam sistem agen modern, di mana tugas berkisar dari penguraian dokumen hingga pembuatan perangkat lunak otonom, kemampuan untuk merutekan permintaan secara dinamis ke model bahasa besar (LLM) atau agen yang paling mampu menjadi penting. Logika perutean statis, sering tertanam dalam skrip orkestrasi atau lapisan API, tidak memiliki kemampuan beradaptasi yang diperlukan untuk lingkungan real-time, multi-model, multi-kemampuan. Untuk mengatasi hal ini, alur kerja perutean LLM dapat diubah menjadi arsitektur berbasis peristiwa yang memanfaatkan pola pengiriman dinamis, mengubah panggilan LLM menjadi peristiwa yang diarahkan secara cerdas dan sadar konteks.

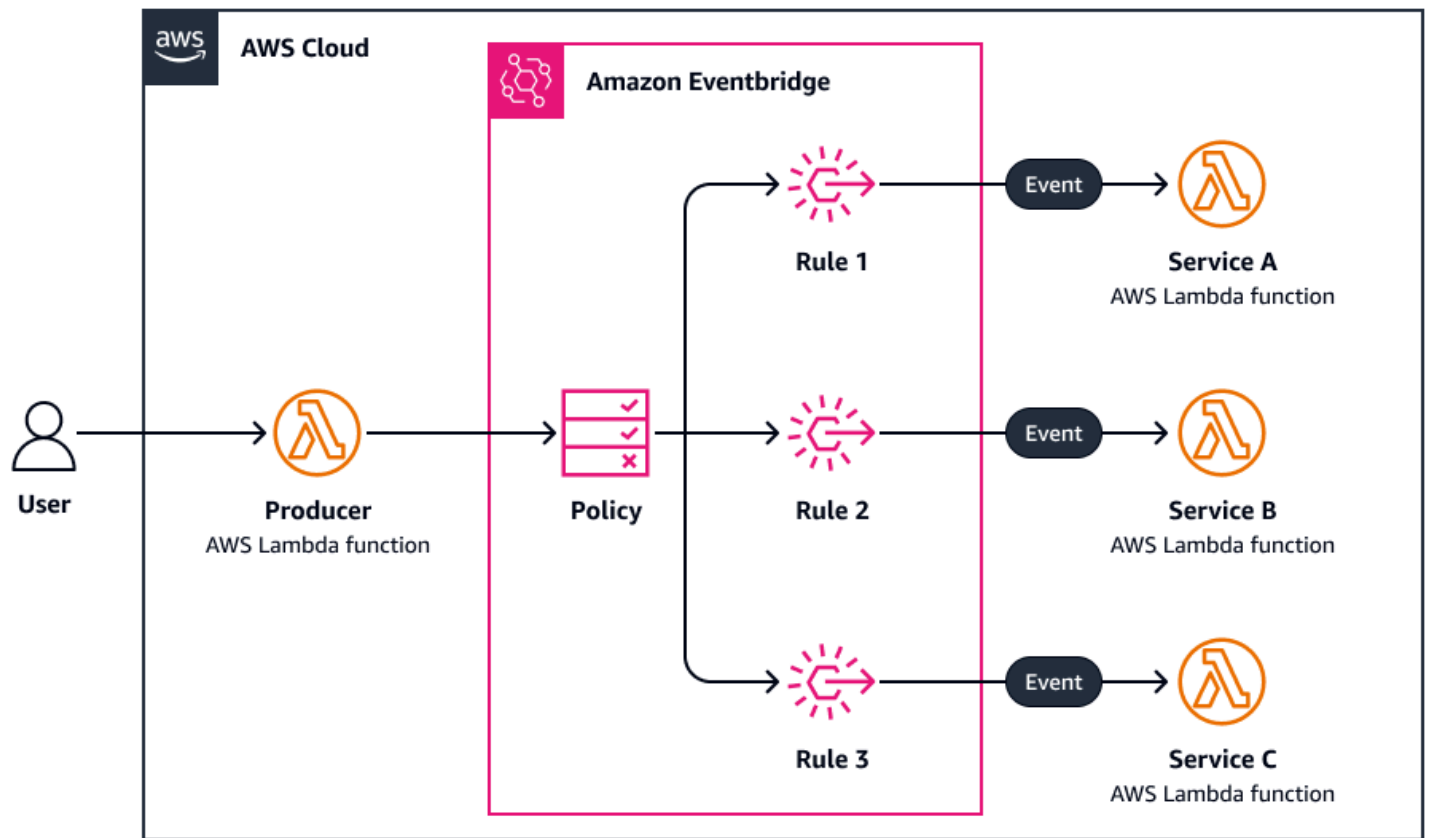
Diagram berikut adalah contoh perutean LLM:



Pengiriman dinamis

Dalam sistem terdistribusi tradisional, pola pengiriman dinamis memilih dan memanggil layanan tertentu saat runtime berdasarkan atribut peristiwa yang masuk, seperti jenis peristiwa, sumber, dan muatan. Ini biasanya diterapkan menggunakan Amazon EventBridge, yang dapat mengevaluasi dan merutekan peristiwa yang masuk ke target yang sesuai (misalnya, AWS Lambda fungsi AWS Step Functions, atau tugas Amazon Elastic Container Service).

Diagram berikut adalah contoh pengiriman dinamis:



1. Aplikasi memancarkan peristiwa (misalnya, `{"type": "orderCreated", "priority": "high"}`).
2. Amazon EventBridge mengevaluasi acara tersebut terhadap aturan peruteannya.
3. Berdasarkan atribut acara, sistem secara dinamis mengirimkan ke yang berikut:
 - `HighPriorityOrderProcessor`(layanan A)
 - `StandardOrderProcessor`(layanan B)
 - `UpdateOrderProcessor`(layanan C)

Pola ini mendukung kopling longgar, spesialisasi berbasis domain, dan ekstensibilitas runtime. Hal ini memungkinkan sistem untuk merespons secara cerdas terhadap perubahan persyaratan dan semantik peristiwa.

Perutean berbasis LLM

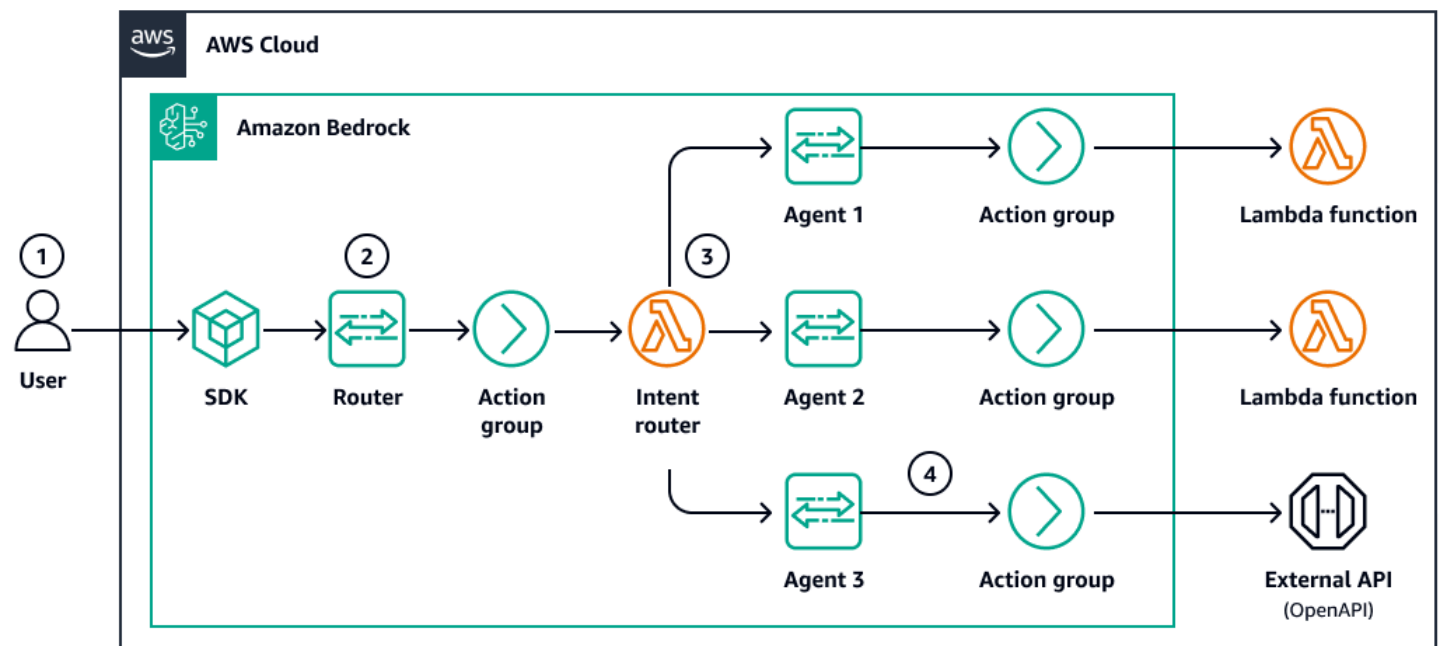
Dalam sistem agen, perutean juga melakukan pendelegasian tugas dinamis — tetapi alih-alih EventBridge aturan Amazon atau filter metadata, LLM mengklasifikasikan dan menafsirkan maksud pengguna melalui bahasa alami. Hasilnya adalah bentuk pengiriman yang fleksibel, semantik, dan adaptif.

Router agen

Arsitektur ini memungkinkan pengiriman berbasis niat yang kaya tanpa skema atau jenis acara yang telah ditentukan sebelumnya, yang ideal untuk input tidak terstruktur dan kueri kompleks.

1. Seorang pengguna mengajukan permintaan “Dapatkan Anda membantu saya meninjau persyaratan kontrak saya?”
2. LLM menafsirkan ini sebagai tugas dokumen hukum.
3. Agen merutekan tugas ke satu atau beberapa hal berikut:
 - Templat prompt peninjauan kontrak
 - Penalaran hukum subagent
 - Alat penguraian dokumen

Diagram berikut adalah contoh router agen:



1. Pengguna mengirimkan permintaan bahasa alami melalui SDK.
2. Agen Amazon Bedrock menggunakan LLM untuk mengklasifikasikan tugas (misalnya, hukum, teknis, atau penjadwalan).
3. Agen secara dinamis merutekan tugas melalui grup tindakan untuk memanggil agen yang diperlukan:
 - Agen khusus domain

- Rantai alat khusus
- Konfigurasi prompt kustom

4. Handler yang dipilih memproses tugas dan mengembalikan respons yang disesuaikan.

Takeaways

Jika pengiriman dinamis tradisional menggunakan EventBridge aturan Amazon untuk perutean berdasarkan atribut peristiwa terstruktur, perutean agen digunakan untuk secara semantik mengklasifikasikan dan LLMs merutekan tugas berdasarkan makna dan maksud. Ini memperluas fleksibilitas sistem dengan mengaktifkan hal-hal berikut:

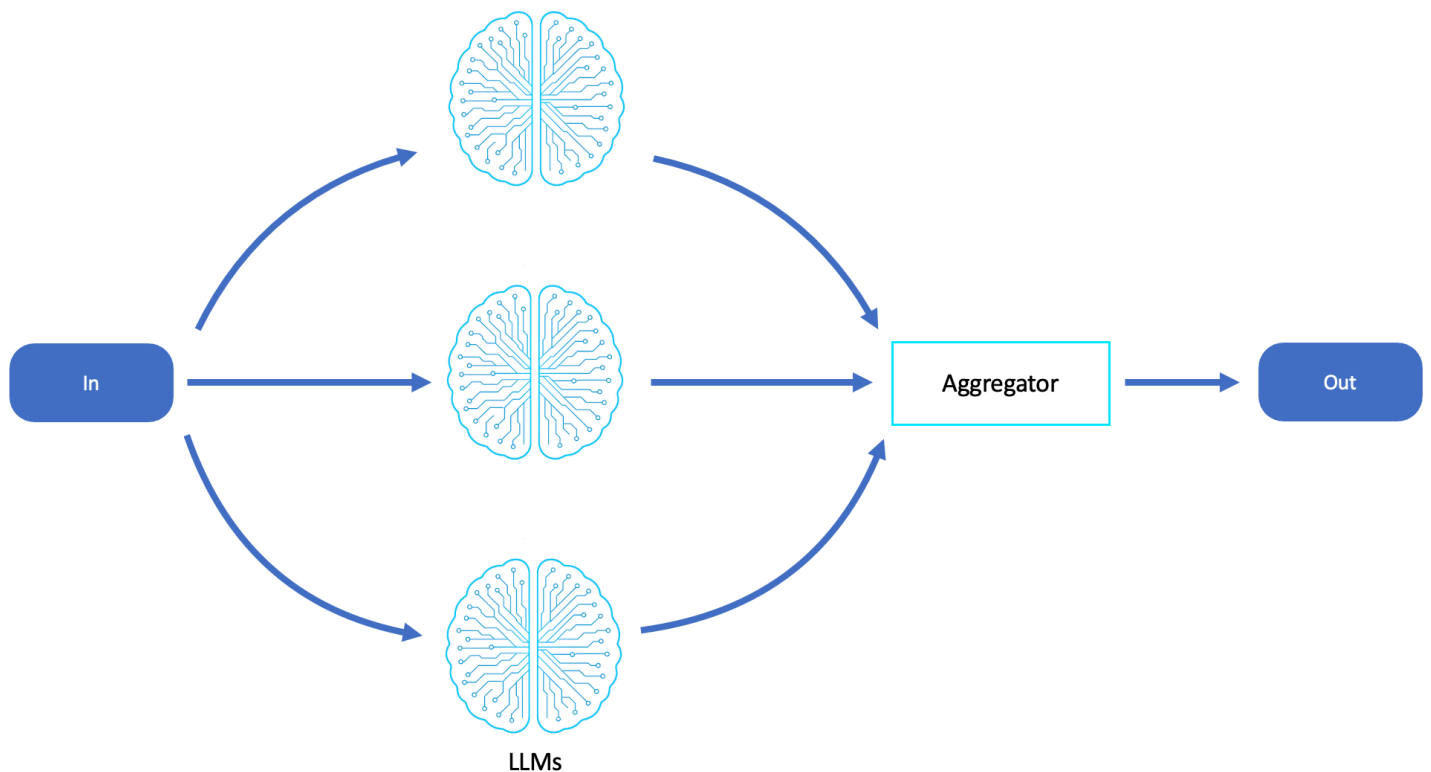
- Pemahaman masukan yang lebih luas
- Fallback cerdas dan pemilihan alat
- Ekstensibilitas alami melalui peran agen baru atau gaya prompt

Perutean agen menggantikan aturan kaku dengan pengiriman kognitif dinamis, yang memungkinkan sistem berkembang dengan bahasa daripada kode.

Paralelisasi dan pola scatter-gathering

Banyak tugas penalaran dan pembuatan tingkat lanjut - seperti meringkas dokumen besar, mengevaluasi beberapa jalur solusi, atau membandingkan perspektif yang beragam - mendapat manfaat dari eksekusi paralel petunjuk. Alur kerja berurutan tradisional gagal ketika skalabilitas, responsif, dan toleransi kesalahan diperlukan. Untuk mengatasinya, paralelisasi berbasis LLM dapat ditata ulang menggunakan pola scatter-gathering yang digerakkan oleh peristiwa, di mana tugas secara dinamis menyebar ke agen otonom dan hasilnya disintesis secara cerdas.

Diagram berikut adalah contoh alur kerja paralelisasi LLM:



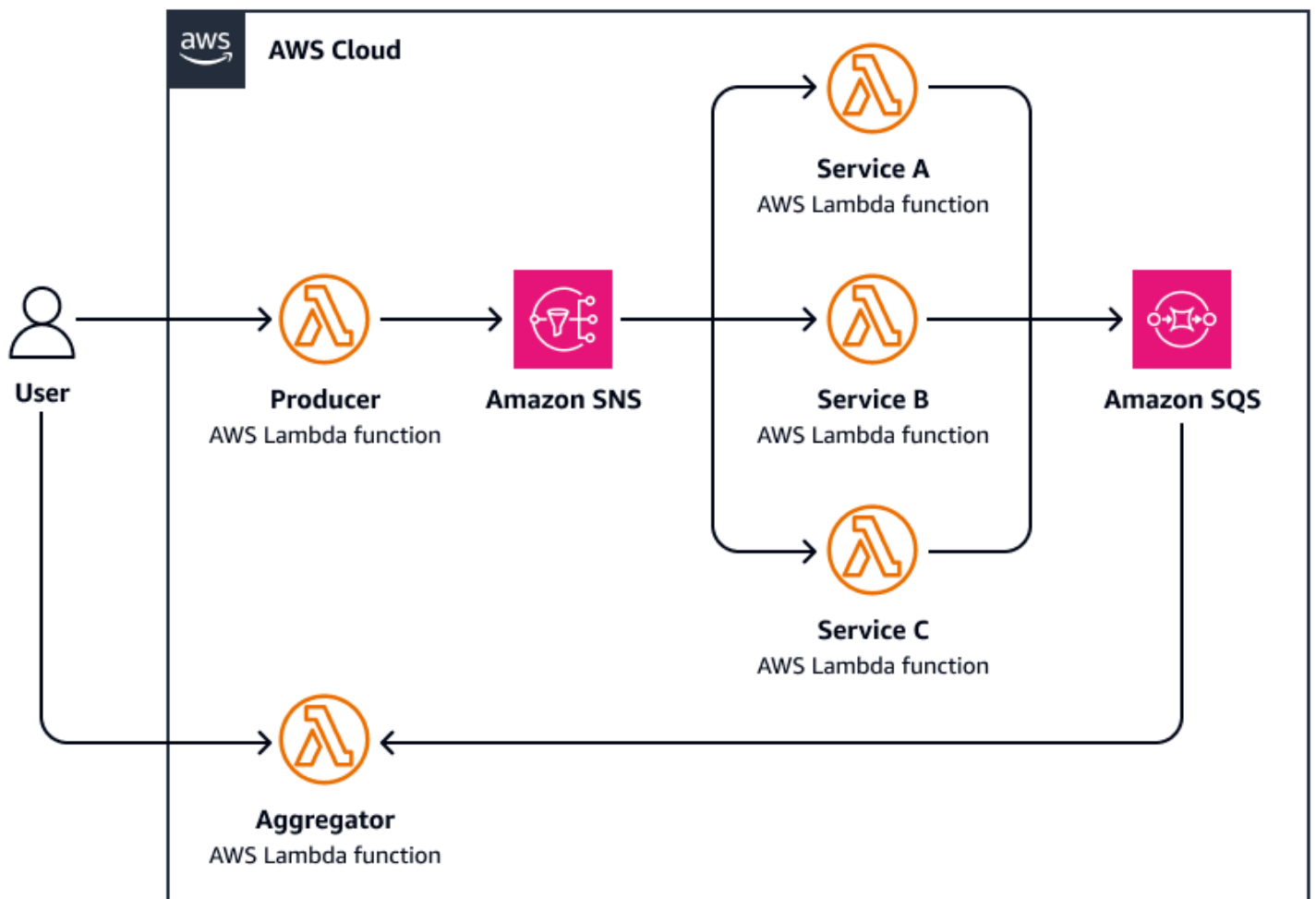
Scatter-kumpulkan

Dalam sistem terdistribusi, pola scatter-gathering mengirimkan tugas ke beberapa layanan atau unit pemrosesan secara paralel, menunggu tanggapan mereka, dan kemudian mengumpulkan hasil menjadi output terkonsolidasi. Tidak seperti fan-out, scatter-gathering dikoordinasikan karena mengharapkan respons dan biasanya menerapkan logika untuk menggabungkan, membandingkan, dan memilih hasil.

Implementasi umum untuk paralelisasi dan scatter-gathering meliputi:

- AWS Step Functions memetakan status untuk eksekusi tugas paralel
- AWS Lambda dengan konkurensi, mengoordinasikan hasil dari beberapa fungsi yang dipanggil
- Amazon EventBridge dengan alur kerja korelasi IDs dan agregasi
- Pola pengontrol khusus untuk mengelola fan-out dan mengumpulkan hasil dengan menggunakan Amazon Simple Storage Service (Amazon S3), Amazon DynamoDB, atau antrian

Diagram berikut adalah contoh scatter-gathering:



1. Pengguna mengirimkan permintaan ke fungsi koordinator pusat yang menyebarkan tugas dengan menerbitkan pesan paralel ke topik Amazon Simple Notification Service (Amazon SNS).
2. Setiap pesan mencakup metadata tugas dan diarahkan ke pekerja khusus. AWS Lambda
3. Setiap pekerja AWS Lambda secara independen memproses subtugas yang ditetapkan (misalnya, menanyakan API eksternal, memproses dokumen, dan menganalisis data).
4. Hasil ditulis ke lapisan penyimpanan umum, seperti Amazon Simple Queue Service (Amazon SQS).
5. Fungsi agregator menunggu semua tanggapan selesai, dan kemudian melakukan hal berikut:
 - Mengumpulkan dan mengumpulkan hasil (misalnya, menggabungkan ringkasan, memilih kecocokan terbaik)
 - Mengirim respons akhir atau memicu alur kerja hilir

Kasus penggunaan umum untuk pola scatter-gathering meliputi yang berikut:

- Pencarian federasi
- Mesin perbandingan harga
- Analisis data agregat
- Inferensi multimodel

Paralelisasi berbasis LLM (kognisi scatter-gathering)

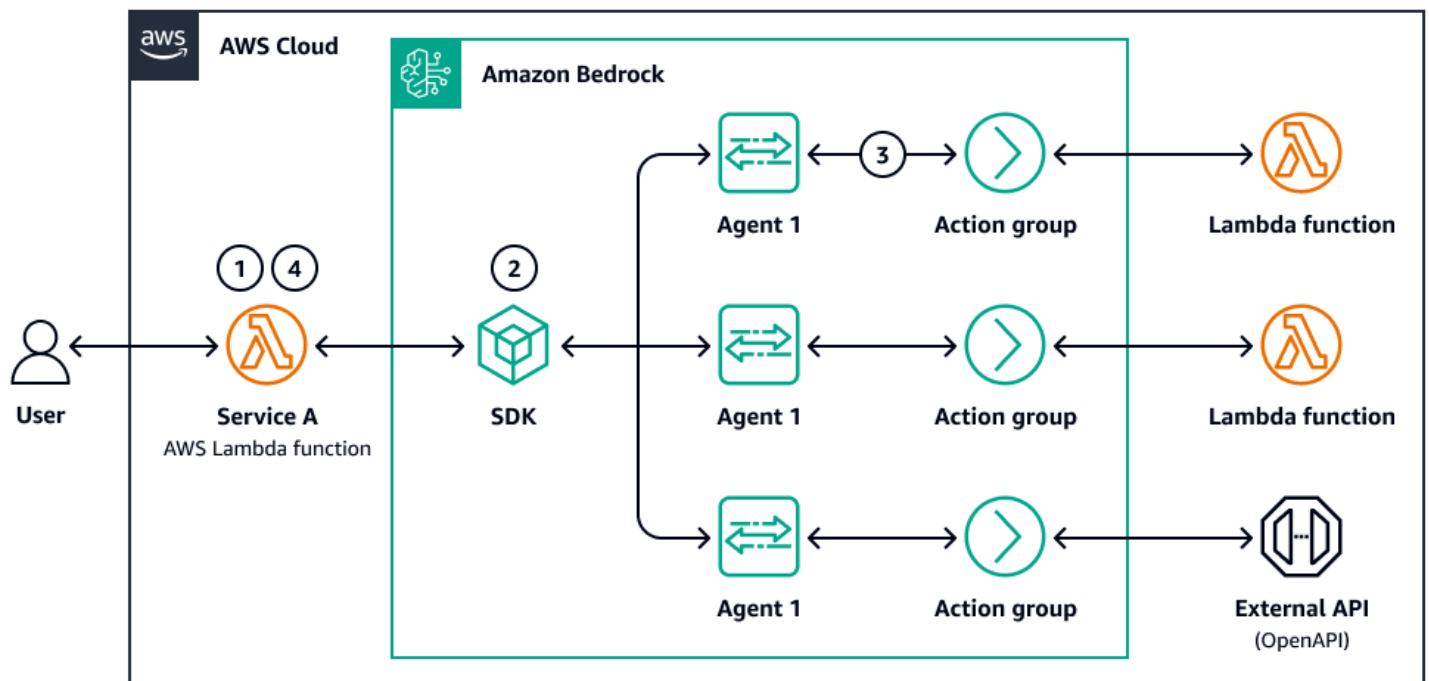
Dalam sistem agen, paralelisasi sangat mencerminkan scatter-gathering dengan mendistribusikan subtugas di beberapa panggilan atau agen LLM, masing-masing secara independen bernalar melalui sebagian masalah. Hasil yang dikembalikan dikumpulkan dan disintesis oleh proses agregasi, yang seringkali merupakan LLM atau agen pengontrol lainnya.

Paralelisasi agen

1. Seorang agen mengajukan permintaan “Ringkas wawasan di 10 laporan ini.”
2. Ini menyebarkan laporan ke 10 tugas ringkasan LLM paralel.
3. Ketika mengembalikan semua ringkasan, agen melakukan hal berikut:
 - Agregat ringkasan menjadi briefing terpadu
 - Mengidentifikasi tema atau kontradiksi
 - Mengirim output yang disintesis ke pengguna

Alur kerja agen ini memungkinkan penalaran paralel yang dapat diskalakan, modular, dan adaptif. Ini sangat ideal untuk kasus penggunaan yang membutuhkan throughput kognitif tinggi.

Diagram berikut adalah contoh paralelisasi agen:



1. Seorang pengguna mengirimkan kueri multipart atau set dokumen.
2. Pengontrol AWS Lambda atau fungsi langkah mendistribusikan subtugas. Setiap tugas memanggil panggilan Amazon Bedrock LLM atau subagent dengan prompt sendiri.
3. Ketika panggilan dan subtugas selesai, hasil disimpan (misalnya, di Amazon S3 atau penyimpanan memori), dan langkah agregasi menggabungkan, membandingkan, atau memfilter output.
4. Sistem mengembalikan respons akhir kepada pengguna atau agen hilir.

Sistem ini memiliki loop penalaran terdistribusi dengan ketertelusuran, toleransi kesalahan, dan pembobotan hasil opsional atau logika seleksi.

Takeaways

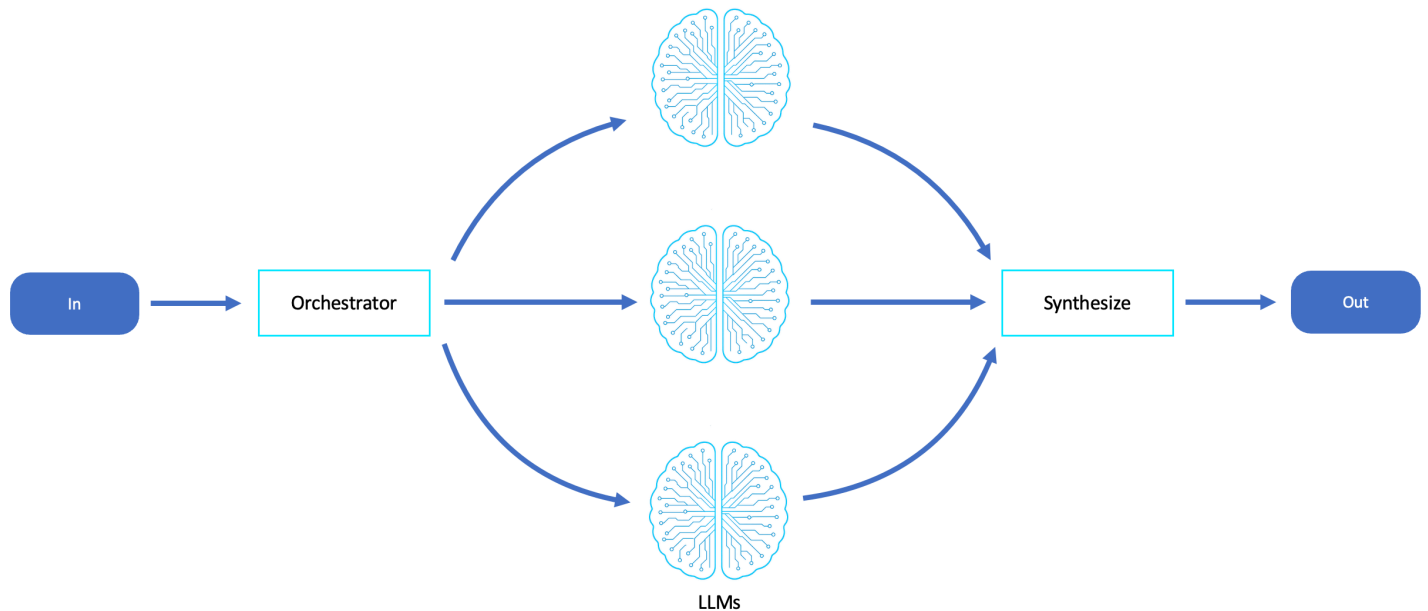
Paralelisasi agen menggunakan pola scatter-gathering untuk mendistribusikan tugas LLM, memungkinkan pemrosesan paralel dan sintesis hasil cerdas.

Pola orkestrasi Saga

Karena alur kerja didorong oleh LLMs menjadi semakin kompleks, mencakup rantai cepat, langkah-langkah pemrosesan data, pemanggilan alat, dan kolaborasi agen, kebutuhan akan orkestrasi cerdas menjadi penting. Daripada mengandalkan skrip yang digabungkan secara ketat atau alur

eksekusi statis yang telah ditentukan sebelumnya, alur kerja ini dapat diimplementasikan sebagai pola orkestrasi yang digerakkan oleh peristiwa, memungkinkan sistem berbasis LLM untuk secara dinamis mengoordinasikan, memantau, dan mengadaptasi tugas multi-langkah di seluruh agen otonom.

Diagram berikut adalah contoh orkestrator:



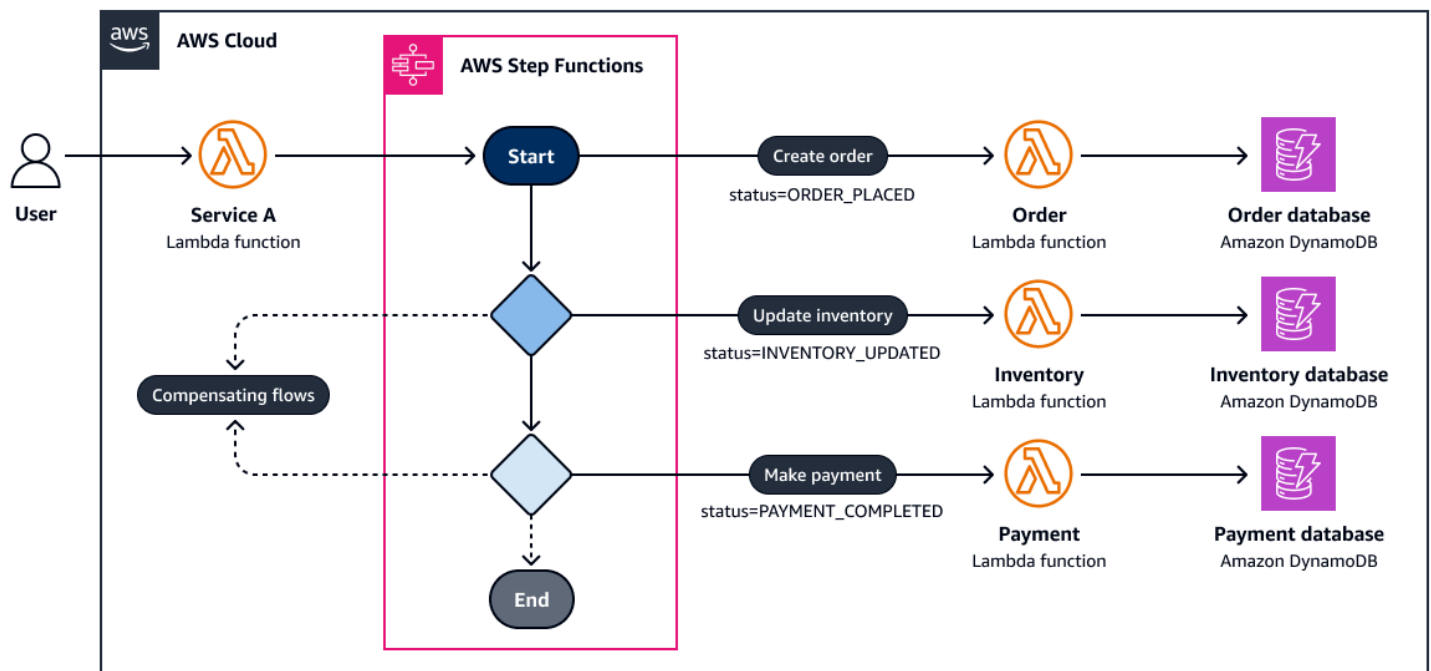
Orkestrasi acara

Dalam sistem terdistribusi tradisional, orkestrasi peristiwa mengacu pada pola di mana koordinator pusat mengelola alur kerja yang kompleks dengan secara eksplisit mengarahkan aliran kontrol di berbagai layanan atau tugas. Tidak seperti koreografi acara (di mana setiap layanan bereaksi secara independen), orkestrasi memberikan logika terpusat, visibilitas, dan kontrol atas seluruh proses.

Ini biasanya diimplementasikan dengan menggunakan alat-alat berikut:

- AWS Step Functions— Tentukan dan jalankan alur kerja stateful
- AWS Lambda— Melaksanakan tugas-tugas diskrit dalam aliran yang diatur
- Amazon SQS atau Amazon EventBridge - Memicu langkah atau respons asinkron

Diagram berikut adalah contoh orkestrasi saga:



AWS Step Functions Alur kerja mengelola proses pesanan pelanggan:

1. Buat pesanan (AWS Lambda)
2. Perbarui inventaris (AWS Lambda)
3. Lakukan pembayaran (AWS Lambda)

Orkestrator mengoordinasikan setiap langkah dengan mengelola percobaan ulang, cabang paralel, batas waktu, dan kegagalan.

Sistem agen berbasis peran (orkestrator)

Dalam sistem agen, pola orkestrator mencerminkan orkestrasi peristiwa tetapi mendistribusikan logika di beberapa agen penalaran, masing-masing dengan peran atau spesialisasi yang ditentukan. Agen orkestrator pusat menafsirkan tugas keseluruhan, menguraikannya menjadi subtask, dan mendelegasikannya ke agen pekerja, masing-masing dioptimalkan untuk domain tertentu (misalnya, penelitian, pengkodean, ringkasan, tinjauan).

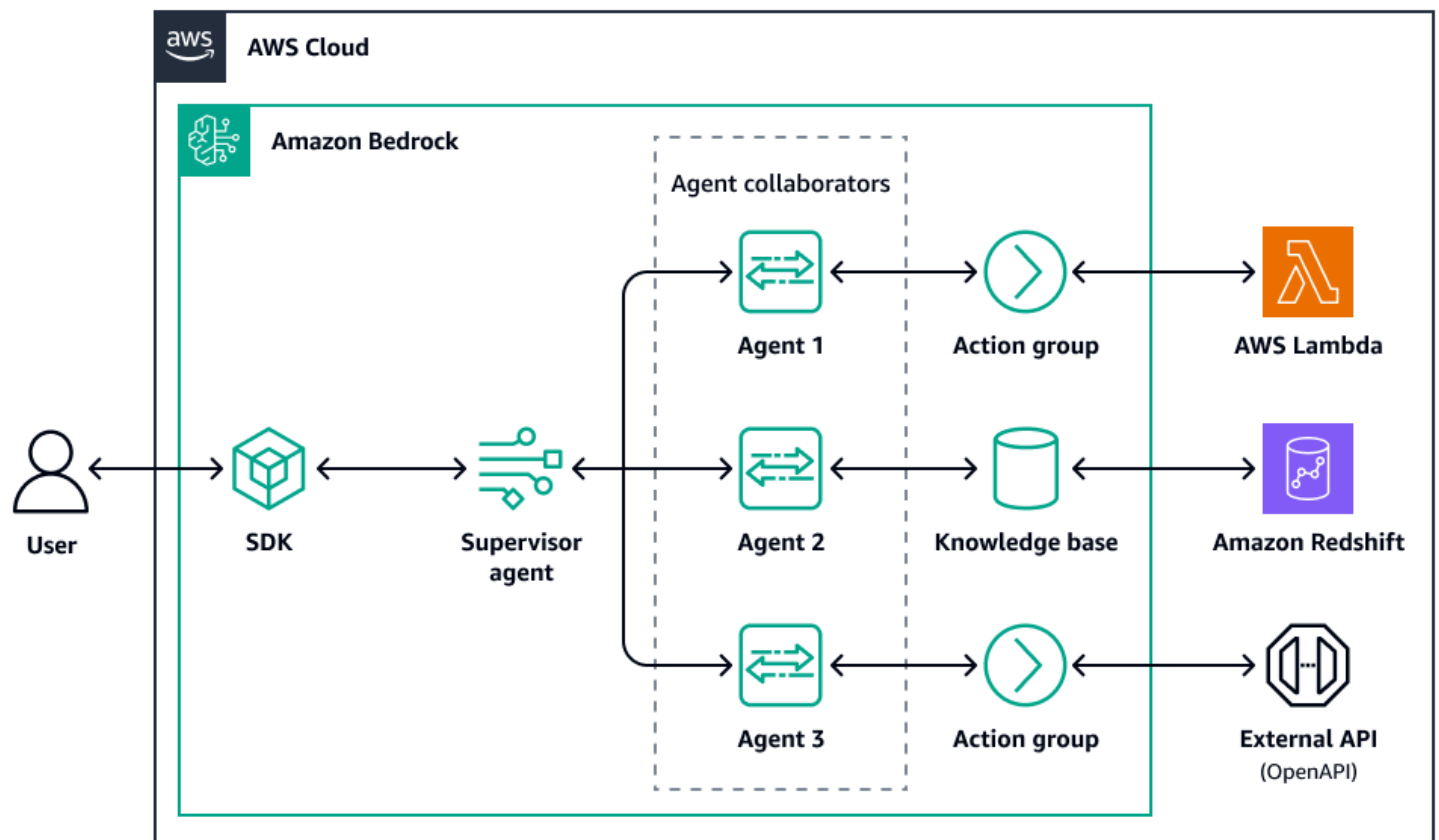
Pengawas

1. Seorang pengguna mengirimkan kueri "Buat ringkasan proyek dan rangkum 5 pesaing teratas."
2. Agen orkestrator melakukan hal berikut:

- Menugaskan agen riset untuk menemukan data pesaing
- Mengirim temuan mentah ke agen ringkasan
- Memberikan hasil ke agen penulis singkat
- Mengompilasi output akhir untuk pengguna

Setiap agen beroperasi secara independen, tetapi orkestrator mengoordinasikan tugas. Ini seperti fungsi Lambda yang menangani tugas alur kerja.

Diagram berikut contoh seorang supervisor:



1. Seorang pengguna mengirimkan tugas ke agen supervisor Amazon Bedrock.
2. Agen supervisor mem-parsing permintaan menjadi subtugas untuk setiap kolaborator agen.
3. Setiap subtugas ditugaskan ke agen kolaborator dengan prompt atau rantai alat khusus peran.
4. Agen pekerja memanggil eksternal APIs atau alat melalui kelompok tindakan.
5. Setiap agen pekerja mengembalikan output dalam format terstruktur.
6. Ketika semua pekerja mengembalikan hasil mereka, supervisor mengevaluasi, mensintesis, dan mengembalikan respons akhir.

Struktur ini memungkinkan modularitas, kemampuan beradaptasi, dan introspeksi di seluruh alur kerja agen multistep yang kompleks.

Takeaways

Di mana orkestrasi peristiwa menggunakan kontrol terpusat (misalnya, AWS Step Functions) untuk mengarahkan eksekusi layanan, sistem agen berbasis peran menggunakan agen orkestrator bertenaga LLM untuk bernalar tentang tujuan, mendelegasikan subtugas ke agen pekerja, dan mensintesis hasil akhir.

Dalam kedua paradigma, orkestrator melakukan hal berikut:

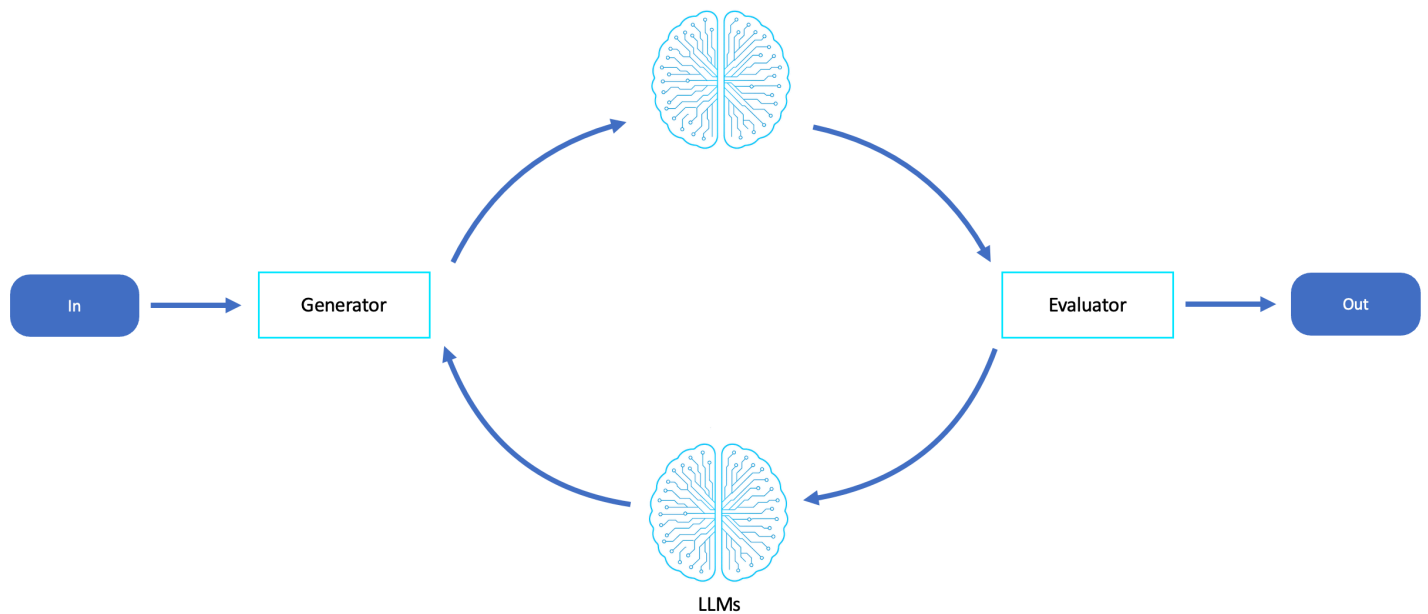
- Mempertahankan konteks dan alur eksekusi
- Menangani percabangan, pengurutan, dan penanganan kesalahan
- Menghasilkan hasil terpadu dari komponen terdistribusi

Orkestrasi agen, bagaimanapun, menambah penalaran, kemampuan beradaptasi, dan delegasi semantik. Ini membuatnya sangat cocok untuk tugas terbuka, ambigu, dan berkembang.

Evaluators memantulkan pola loop

Tugas-tugas seperti pembuatan kode, ringkasan, atau pengambilan keputusan otonom mendapat manfaat besar dari umpan balik runtime, memungkinkan sistem untuk berkembang melalui pengamatan dan penyempurnaan. Untuk mengoperasionalkan ini, siklus reflekt-perbaikan dapat diimplementasikan sebagai loop kontrol umpan balik yang digerakkan oleh peristiwa - pola yang terinspirasi oleh rekayasa sistem, yang disesuaikan untuk alur kerja yang otonom dan cerdas.

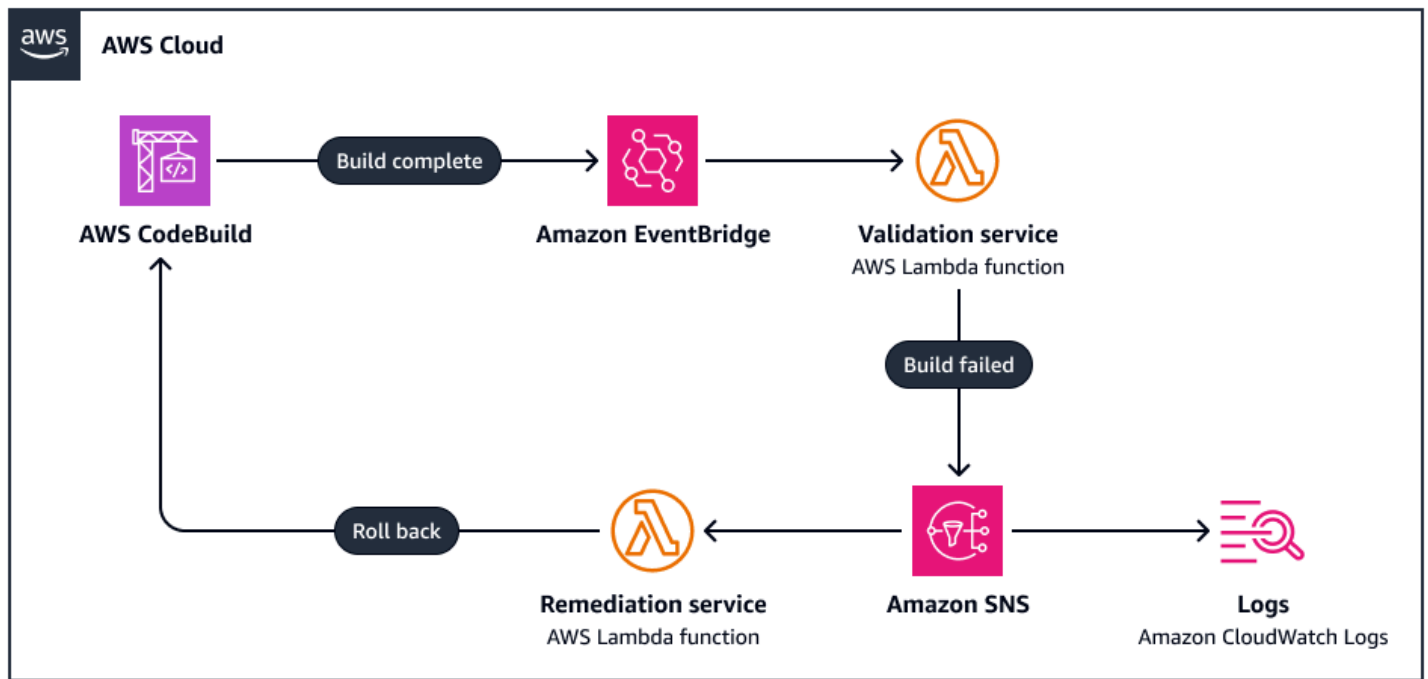
Diagram berikut adalah contoh dari evaluator reflect-refine feedback loop:



Lingkaran kontrol umpan balik

Loop kontrol umpan balik adalah pola yang memantau keluaran dan perilakunya sendiri, mengevaluasinya terhadap kriteria yang ditentukan atau keadaan yang diinginkan, dan kemudian menyesuaikan tindakannya sesuai dengan itu. Arsitektur ini terinspirasi oleh teori kontrol dan merupakan dasar dalam domain seperti otomatisasi, integrasi berkelanjutan dan jalur pipa pengiriman berkelanjutan (CI/CD), dan operasi pembelajaran mesin.

Diagram berikut adalah contoh loop kontrol umpan balik:



1. Pipeline penerapan memancarkan peristiwa BuildComplete.
2. Peristiwa ini memicu pekerjaan pengujian atau evaluasi otomatis yang memvalidasi build.
3. Jika validasi gagal (misalnya, karena pengujian yang gagal, masalah keamanan, atau pelanggaran kebijakan), sistem:
 - Memancarkan acara BuildComplete
 - Mencatat masalah atau mengirim pemberitahuan
 - Memicu remediasi atau tindakan korektif, seperti rollback, patching, atau coba lagi

Loop berlanjut sampai menghasilkan hasil atau eskalasi yang dapat diterima, atau time out terjadi. Pola ini biasanya digunakan untuk hal-hal berikut:

- EventBridge Aturan Amazon untuk merutekan acara ke tugas evaluasi atau remediasi
- AWS Step Functions untuk logika coba lagi berulang dan percabangan pada hasil evaluasi
- Amazon Simple Notification Service (Amazon SNS) atau alarm CloudWatch Amazon untuk pemicu dan peringatan umpan balik
- AWS Lambda fungsi atau pekerja kontainer untuk menerapkan tindakan korektif

Lingkaran kontrol umpan balik (evaluator)

Alur kerja evaluator adalah loop umpan balik kognitif yang didukung oleh LLMs atau agen penalaran. Prosesnya terdiri dari yang berikut:

1. Agen generator atau LLM menghasilkan output (misalnya, rencana, jawaban, atau draf).
2. Agen evaluator meninjau hasilnya menggunakan prompt kritik atau rubrik evaluasi.
3. Berdasarkan umpan balik, agen asli atau agen pengoptimal baru merevisi output.

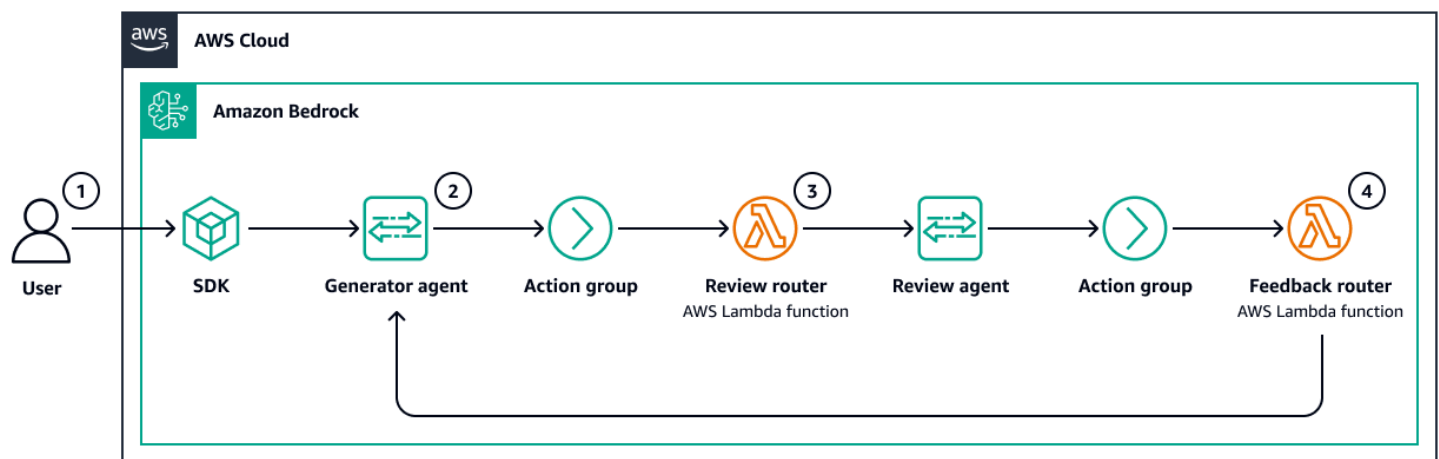
Loop berulang sampai hasilnya memenuhi serangkaian kriteria, disetujui, atau mencapai batas coba lagi.

Penilai

1. Seorang pengguna meminta agen untuk menulis ringkasan kebijakan.
2. Agen generator menyusunnya.
3. Agen evaluator memeriksa cakupan, nada, dan kebenaran hukum.
4. Jika respons tidak memadai, itu disempurnakan dan dikirim ulang hingga loop umpan balik menyatu.

Ini memungkinkan penilaian diri, penyempurnaan berulang, dan kontrol keluaran adaptif — semuanya tanpa masukan manusia.

Diagram berikut adalah contoh loop kontrol umpan balik (evaluator):



1. Seorang pengguna mengeluarkan tugas (misalnya, menyusun strategi bisnis).

2. Agen Amazon Bedrock menghasilkan draf awal menggunakan LLM.
3. Agen kedua (atau prompt tindak lanjut) melakukan evaluasi terstruktur (misalnya, “nilai output ini dengan kejelasan, kelengkapan, dan nada”).
4. Jika peringkat jatuh di bawah ambang batas, respons direvisi oleh:
 - Menerapkan kembali generator dengan kritik tertanam
 - Mengirim umpan balik ke agen penyuling khusus
 - Iterasi sampai respon yang dapat diterima tercapai

Komponen opsional seperti AWS Lambda pengontrol atau AWS Step Functions dapat mengelola ambang umpan balik, percobaan ulang, dan strategi mundur.

Takeaways

Di mana loop kontrol umpan balik tradisional menggunakan peristiwa, metrik, dan logika remediasi untuk memvalidasi dan menyesuaikan perilaku sistem, loop evaluator agen menggunakan agen penalaran untuk mengevaluasi, merefleksikan, dan merevisi output secara dinamis.

Dalam kedua paradigma:

- Output dievaluasi setelah dihasilkan
- Tindakan korektif atau pemurnian dipicu berdasarkan umpan balik
- Sistem terus beradaptasi terhadap kualitas target atau tujuan

Versi agen mengubah validasi statis menjadi refleksi semantik, memungkinkan agen peningkatan diri yang mengevaluasi efektivitas mereka sendiri.

Merancang alur kerja agen pada AWS

Setiap pola dalam panduan ini dapat dibangun menggunakan Layanan AWS. Agen Amazon Bedrock menyediakan orkestrasi, akses data, dan saluran interaksi.

Komponen	Layanan AWS	Tujuan
Alasan LLM	Amazon Bedrock	Logika agen, perencanaan, penggunaan alat

Eksekusi alat	AWS Lambda, Amazon ECS, Amazon SageMaker	Host alat eksternal untuk agen
Memori dan RAG	Basis pengetahuan Amazon Bedrock, Amazon S3, OpenSearch	Memori persisten dan semantik
Orkestrasi	AWS Step Functions	Tugas multistep dan koordinasi agen
Perutean acara	Amazon EventBridge, Amazon SQS	Pesan interagen terpisah
Antarmuka pengguna	Amazon API Gateway, AWS AppSync, SDK	Titik masuk untuk aplikasi atau sistem
Memantau	Amazon CloudWatch, AWS X-Ray, AWS Distro untuk OpenTelemetry	Observabilitas dan introspeksi agen

Kesimpulan

Pola alur kerja agen adalah tahap evolusi berikutnya dari arsitektur berbasis peristiwa, di mana logika bisnis tidak didefinisikan secara statis tetapi beralasan secara dinamis melalui penggunaan kognisi yang ditingkatkan model bahasa besar (LLM). Dengan menggabungkan primitif cloud-native tradisional dengan alur kerja LLM dan pola desain agen, organisasi dapat membangun sistem adaptif, cerdas, dan modular yang merespons dengan tujuan dan belajar dari pengalaman.

Dalam pola ini, Amazon Bedrock adalah pintu gerbang ke kognisi agen, memungkinkan agen berbasis LLM untuk mengakses alur kerja peristiwa, berinteraksi dengan alat dan memori, dan memberikan hasil yang terstruktur, dapat dilacak, dan selaras.

Saat Anda merancang dan menerapkan sistem agen, pola alur kerja ini menyediakan cetak biru untuk membangun arsitektur AI yang otonom dan dapat dikomposisi. Sistem ini didasarkan pada praktik terbaik tanpa server dan ditambah dengan model pondasi cerdas.

Riwayat dokumen

Tabel berikut menjelaskan perubahan signifikan pada panduan ini. Jika Anda ingin diberi tahu tentang pembaruan masa depan, Anda dapat berlangganan umpan [RSS](#).

Perubahan	Deskripsi	Tanggal
Publikasi awal	—	Juli 14, 2025

AWS Glosarium Panduan Preskriptif

Berikut ini adalah istilah yang umum digunakan dalam strategi, panduan, dan pola yang disediakan oleh Panduan AWS Preskriptif. Untuk menyarankan entri, silakan gunakan tautan Berikan umpan balik di akhir glosarium.

Nomor

7 Rs

Tujuh strategi migrasi umum untuk memindahkan aplikasi ke cloud. Strategi ini dibangun di atas 5 Rs yang diidentifikasi Gartner pada tahun 2011 dan terdiri dari yang berikut:

- Refactor/re-architect — Pindahkan aplikasi dan modifikasi arsitekturnya dengan memanfaatkan sepenuhnya fitur cloud-native untuk meningkatkan kelincahan, kinerja, dan skalabilitas. Ini biasanya melibatkan porting sistem operasi dan database. Contoh: Migrasikan database Oracle lokal Anda ke Amazon Aurora Edition. PostgreSQL-Compatible
- Replatform (angkat dan bentuk ulang) — Pindahkan aplikasi ke cloud, dan perkenalkan beberapa tingkat pengoptimalan untuk memanfaatkan kemampuan cloud. Contoh: Memigrasikan database Oracle lokal Anda ke Amazon Relational Database Service (Amazon RDS) untuk Oracle di AWS Cloud
- Pembelian kembali (drop and shop) - Beralih ke produk yang berbeda, biasanya dengan beralih dari lisensi tradisional ke model SaaS. Contoh: Migrasikan sistem manajemen hubungan pelanggan (CRM) Anda ke Salesforce.com
- Rehost (lift dan shift) — Pindahkan aplikasi ke cloud tanpa membuat perubahan apa pun untuk memanfaatkan kemampuan cloud. Contoh: Migrasikan database Oracle lokal Anda ke Oracle pada instans EC2 di AWS Cloud
- Relokasi (hypervisor-level lift and shift) — Pindahkan infrastruktur ke cloud tanpa membeli perangkat keras baru, menulis ulang aplikasi, atau memodifikasi operasi yang ada. Anda memigrasikan server dari platform lokal ke layanan cloud untuk platform yang sama. Contoh: Migrasikan Microsoft Hyper-V aplikasi ke AWS.
- Pertahankan (kunjungi kembali) - Simpan aplikasi di lingkungan sumber Anda. Ini mungkin termasuk aplikasi yang memerlukan refactoring besar, dan Anda ingin menunda pekerjaan itu sampai nanti, dan aplikasi lama yang ingin Anda pertahankan, karena tidak ada pembenaran bisnis untuk memigrasikannya.

- **Pensiun** — Menonaktifkan atau menghapus aplikasi yang tidak lagi diperlukan di lingkungan sumber Anda.

A

A2A () Agent-to-Agent

Protokol stateful untuk kolaborasi agen-ke-agen yang mendukung delegasi tugas dan transfer negara.

ABAC

Lihat [kontrol akses berbasis atribut](#).

layanan abstrak

Lihat [layanan terkelola](#).

ASAM

Lihat [atomisitas, konsistensi, isolasi, daya tahan](#).

migrasi aktif-aktif

Metode migrasi database di mana basis data sumber dan target tetap sinkron (dengan menggunakan alat replikasi dua arah atau operasi penulisan ganda), dan kedua database menangani transaksi dari menghubungkan aplikasi selama migrasi. Metode ini mendukung migrasi dalam batch kecil yang terkontrol alih-alih memerlukan pemotongan satu kali. Ini lebih fleksibel tetapi membutuhkan lebih banyak pekerjaan daripada migrasi [aktif-pasif](#).

migrasi aktif-pasif

Metode migrasi database di mana database sumber dan target disimpan dalam sinkron, tetapi hanya database sumber yang menangani transaksi dari menghubungkan aplikasi sementara data direplikasi ke database target. Basis data target tidak menerima transaksi apa pun selama migrasi.

Agen

Sistem AI yang dapat secara mandiri bernalar, merencanakan, dan mengambil tindakan menggunakan alat untuk mencapai tujuan.

Agen Ops

Praktik operasional untuk membangun, menguji, menyebarkan, dan menjalankan agen AI dalam produksi dalam skala besar.

fungsi agregat

Fungsi SQL yang beroperasi pada sekelompok baris dan menghitung nilai pengembalian tunggal untuk grup. Contoh fungsi agregat meliputi SUM dan MAX.

AI

Lihat [kecerdasan buatan](#).

AIOps

Lihat [operasi kecerdasan buatan](#).

anonimisasi

Proses menghapus informasi pribadi secara permanen dalam kumpulan data. Anonimisasi dapat membantu melindungi privasi pribadi. Data anonim tidak lagi dianggap sebagai data pribadi.

anti-pola

Solusi yang sering digunakan untuk masalah berulang di mana solusinya kontra-produktif, tidak efektif, atau kurang efektif daripada alternatif.

kontrol aplikasi

Pendekatan keamanan yang memungkinkan penggunaan hanya aplikasi yang disetujui untuk membantu melindungi sistem dari malware.

portofolio aplikasi

Kumpulan informasi rinci tentang setiap aplikasi yang digunakan oleh organisasi, termasuk biaya untuk membangun dan memelihara aplikasi, dan nilai bisnisnya. Informasi ini adalah kunci untuk [penemuan portofolio dan proses analisis dan](#) membantu mengidentifikasi dan memprioritaskan aplikasi yang akan dimigrasi, dimodernisasi, dan dioptimalkan.

kecerdasan buatan (AI)

Bidang ilmu komputer yang didedikasikan untuk menggunakan teknologi komputasi untuk melakukan fungsi kognitif yang biasanya terkait dengan manusia, seperti belajar, memecahkan masalah, dan mengenali pola. Untuk informasi lebih lanjut, lihat [Apa itu Kecerdasan Buatan?](#)

operasi kecerdasan buatan (AIOps)

Proses menggunakan teknik pembelajaran mesin untuk memecahkan masalah operasional, mengurangi insiden operasional dan intervensi manusia, dan meningkatkan kualitas layanan. Untuk informasi selengkapnya tentang cara AIOps digunakan dalam strategi AWS migrasi, lihat [panduan integrasi operasi](#).

enkripsi asimetris

Algoritma enkripsi yang menggunakan sepasang kunci, kunci publik untuk enkripsi dan kunci pribadi untuk dekripsi. Anda dapat berbagi kunci publik karena tidak digunakan untuk dekripsi, tetapi akses ke kunci pribadi harus sangat dibatasi.

atomisitas, konsistensi, isolasi, daya tahan (ACID)

Satu set properti perangkat lunak yang menjamin validitas data dan keandalan operasional database, bahkan dalam kasus kesalahan, kegagalan daya, atau masalah lainnya.

kontrol akses berbasis atribut (ABAC)

Praktik membuat izin berbutir halus berdasarkan atribut pengguna, seperti departemen, peran pekerjaan, dan nama tim. Untuk informasi selengkapnya, lihat [ABAC untuk AWS](#) dokumentasi AWS Identity and Access Management (IAM).

sumber data otoritatif

Lokasi di mana Anda menyimpan versi utama data, yang dianggap sebagai sumber informasi yang paling dapat diandalkan. Anda dapat menyalin data dari sumber data otoritatif ke lokasi lain untuk tujuan memproses atau memodifikasi data, seperti menganonimkan, menyunting, atau membuat nama samaran.

Zona Ketersediaan

Lokasi berbeda di dalam Wilayah AWS yang terisolasi dari kegagalan di Availability Zone lainnya dan menyediakan konektivitas jaringan latensi rendah yang murah ke Availability Zone lainnya di Wilayah yang sama.

AWS Kerangka Adopsi Cloud (AWS CAF)

Kerangka pedoman dan praktik terbaik AWS untuk membantu organisasi mengembangkan rencana yang efisien dan efektif untuk bergerak dengan sukses ke cloud. AWS CAF mengatur panduan ke dalam enam area fokus yang disebut perspektif: bisnis, orang, tata kelola, platform, keamanan, dan operasi. Perspektif bisnis, orang, dan tata kelola fokus pada keterampilan dan proses bisnis; perspektif platform, keamanan, dan operasi fokus pada keterampilan dan proses teknis. Misalnya, perspektif masyarakat menargetkan pemangku kepentingan yang menangani

sumber daya manusia (SDM), fungsi kepegawaian, dan manajemen orang. Untuk perspektif ini, AWS CAF memberikan panduan untuk pengembangan, pelatihan, dan komunikasi orang untuk membantu mempersiapkan organisasi untuk adopsi cloud yang sukses. Untuk informasi lebih lanjut, lihat [situs web AWS CAF](#) dan [whitepaper AWS CAF](#).

AWS Kerangka Kualifikasi Beban Kerja (AWS WQF)

Alat yang mengevaluasi beban kerja migrasi database, merekomendasikan strategi migrasi, dan memberikan perkiraan kerja. AWS WQF disertakan dengan AWS Schema Conversion Tool (AWS SCT). Ini menganalisis skema database dan objek kode, kode aplikasi, dependensi, dan karakteristik kinerja, dan memberikan laporan penilaian.

B

bot buruk

[Bot](#) yang dimaksudkan untuk mengganggu atau menyebabkan kerugian bagi individu atau organisasi.

BCP

Lihat [perencanaan kontinuitas bisnis](#).

grafik perilaku

Pandangan interaktif yang terpadu tentang perilaku dan interaksi sumber daya dari waktu ke waktu. Anda dapat menggunakan grafik perilaku dengan Amazon Detective untuk memeriksa upaya login yang gagal, panggilan API yang mencurigakan, dan tindakan serupa. Untuk informasi selengkapnya, lihat [Data dalam grafik perilaku](#) di dokumentasi Detektif.

sistem big-endian

Sistem yang menyimpan byte paling signifikan terlebih dahulu. Lihat juga [endianness](#).

klasifikasi biner

Sebuah proses yang memprediksi hasil biner (salah satu dari dua kelas yang mungkin). Misalnya, model ML Anda mungkin perlu memprediksi masalah seperti “Apakah email ini spam atau bukan spam?” atau “Apakah produk ini buku atau mobil?”

filter mekar

Struktur data probabilistik dan efisien memori yang digunakan untuk menguji apakah suatu elemen adalah anggota dari suatu himpunan.

blue/green penyebaran

Strategi penyebaran tempat Anda membuat dua lingkungan yang terpisah namun identik. Anda menjalankan versi aplikasi saat ini di satu lingkungan (biru) dan versi aplikasi baru di lingkungan lain (hijau). Strategi ini membantu Anda dengan cepat memutar kembali dengan dampak minimal.

bot

Aplikasi perangkat lunak yang menjalankan tugas otomatis melalui internet dan mensimulasikan aktivitas atau interaksi manusia. Beberapa bot berguna atau bermanfaat, seperti perayap web yang mengindeks informasi di internet. Beberapa bot lain, yang dikenal sebagai bot buruk, dimaksudkan untuk mengganggu atau membahayakan individu atau organisasi.

botnet

Jaringan [bot](#) yang terinfeksi oleh [malware](#) dan berada di bawah kendali satu pihak, yang dikenal sebagai bot herder atau operator bot. Botnet adalah mekanisme paling terkenal untuk skala bot dan dampaknya.

cabang

Area berisi repositori kode. Cabang pertama yang dibuat dalam repositori adalah cabang utama. Anda dapat membuat cabang baru dari cabang yang ada, dan Anda kemudian dapat mengembangkan fitur atau memperbaiki bug di cabang baru. Cabang yang Anda buat untuk membangun fitur biasanya disebut sebagai cabang fitur. Saat fitur siap dirilis, Anda menggabungkan cabang fitur kembali ke cabang utama. Untuk informasi selengkapnya, lihat [Tentang cabang](#) (GitHub dokumentasi).

akses break-glass

Dalam keadaan luar biasa dan melalui proses yang disetujui, cara cepat bagi pengguna untuk mendapatkan akses ke Akun AWS yang biasanya tidak memiliki izin untuk mengaksesnya. Untuk informasi lebih lanjut, lihat indikator [Implementasikan prosedur break-glass](#) dalam panduan. AWS Well-Architected

strategi brownfield

Infrastruktur yang ada di lingkungan Anda. Saat mengadopsi strategi brownfield untuk arsitektur sistem, Anda merancang arsitektur di sekitar kendala sistem dan infrastruktur saat ini. Jika Anda memperluas infrastruktur yang ada, Anda dapat memadukan strategi brownfield dan [greenfield](#).

cache penyangga

Area memori tempat data yang paling sering diakses disimpan.

kemampuan bisnis

Apa yang dilakukan bisnis untuk menghasilkan nilai (misalnya, penjualan, layanan pelanggan, atau pemasaran). Arsitektur layanan mikro dan keputusan pengembangan dapat didorong oleh kemampuan bisnis. Untuk informasi selengkapnya, lihat bagian [Terorganisir di sekitar kemampuan bisnis](#) dari [Menjalankan layanan mikro kontainer](#) di whitepaper. AWS

perencanaan kelangsungan bisnis (BCP)

Rencana yang membahas dampak potensial dari peristiwa yang mengganggu, seperti migrasi skala besar, pada operasi dan memungkinkan bisnis untuk melanjutkan operasi dengan cepat.

C

KAFE

Lihat [Kerangka Adopsi AWS Cloud](#).

penyebaran kenari

Rilis versi yang lambat dan bertahap untuk pengguna akhir. Ketika Anda yakin, Anda menyebarkan versi baru dan mengganti versi saat ini secara keseluruhan.

CCoE

Lihat [Cloud Center of Excellence](#).

CDC

Lihat [mengubah pengambilan data](#).

ubah pengambilan data (CDC)

Proses melacak perubahan ke sumber data, seperti tabel database, dan merekam metadata tentang perubahan tersebut. Anda dapat menggunakan CDC untuk berbagai tujuan, seperti mengaudit atau mereplikasi perubahan dalam sistem target untuk mempertahankan sinkronisasi.

rekayasa kekacauan

Sengaja memperkenalkan kegagalan atau peristiwa yang mengganggu untuk menguji ketahanan sistem. Anda dapat menggunakan [AWS Fault Injection Service \(AWS FIS\)](#) untuk melakukan eksperimen yang menekankan AWS beban kerja Anda dan mengevaluasi responsnya.

CI/CD

Lihat [integrasi berkelanjutan dan pengiriman berkelanjutan](#).

klasifikasi

Proses kategorisasi yang membantu menghasilkan prediksi. Model ML untuk masalah klasifikasi memprediksi nilai diskrit. Nilai diskrit selalu berbeda satu sama lain. Misalnya, model mungkin perlu mengevaluasi apakah ada mobil dalam gambar atau tidak.

Pengembang Warga

Pengguna bisnis yang membuat aplikasi AI menggunakan platform tanpa code/low kode tanpa keterampilan teknis khusus.

Enkripsi sisi klien

Enkripsi data secara lokal, sebelum target Layanan AWS menerimanya.

Cloud Center of Excellence (CCoE)

Tim multi-disiplin yang mendorong upaya adopsi cloud di seluruh organisasi, termasuk mengembangkan praktik terbaik cloud, memobilisasi sumber daya, menetapkan jadwal migrasi, dan memimpin organisasi melalui transformasi skala besar. Untuk informasi selengkapnya, lihat [posting CCoE](#) di Blog Strategi AWS Cloud Perusahaan.

komputasi cloud

Teknologi cloud yang biasanya digunakan untuk penyimpanan data jarak jauh dan manajemen perangkat IoT. Cloud computing umumnya terhubung ke teknologi [edge computing](#).

model operasi cloud

Dalam organisasi TI, model operasi yang digunakan untuk membangun, mematangkan, dan mengoptimalkan satu atau lebih lingkungan cloud. Untuk informasi selengkapnya, lihat [Membangun Model Operasi Cloud Anda](#).

tahap adopsi cloud

Empat fase yang biasanya dilalui organisasi ketika mereka bermigrasi ke AWS Cloud:

- Proyek — Menjalankan beberapa proyek terkait cloud untuk bukti konsep dan tujuan pembelajaran
- Foundation — Melakukan investasi dasar untuk meningkatkan adopsi cloud Anda (misalnya, membuat landing zone, mendefinisikan CCoE, membuat model operasi)
- Migrasi — Migrasi aplikasi individual
- Re-invention — Mengoptimalkan produk dan layanan, dan berinovasi di cloud

Tahapan ini didefinisikan oleh Stephen Orban dalam posting blog [The Journey Toward Cloud-First & the Stages of Adoption](#) di blog Strategi AWS Cloud Perusahaan. Untuk informasi tentang bagaimana kaitannya dengan strategi AWS migrasi, lihat [panduan kesiapan migrasi](#).

CMDB

Lihat [database manajemen konfigurasi](#).

repositori kode

Lokasi di mana kode sumber dan aset lainnya, seperti dokumentasi, sampel, dan skrip, disimpan dan diperbarui melalui proses kontrol versi. Repositori cloud umum termasuk GitHub atau Bitbucket Cloud. Setiap versi kode disebut cabang. Dalam struktur layanan mikro, setiap repositori dikhususkan untuk satu bagian fungsionalitas. Satu CI/CD pipa dapat menggunakan beberapa repositori.

cache dingin

Cache buffer yang kosong, tidak terisi dengan baik, atau berisi data basi atau tidak relevan. Ini mempengaruhi kinerja karena instance database harus membaca dari memori utama atau disk, yang lebih lambat daripada membaca dari cache buffer.

data dingin

Data yang jarang diakses dan biasanya historis. Saat menanyakan jenis data ini, kueri lambat biasanya dapat diterima. Memindahkan data ini ke tingkat atau kelas penyimpanan yang berkinerja lebih rendah dan lebih murah dapat mengurangi biaya.

visi komputer (CV)

Bidang [AI](#) yang menggunakan pembelajaran mesin untuk menganalisis dan mengekstrak informasi dari format visual seperti gambar dan video digital. Misalnya, Amazon SageMaker AI menyediakan algoritma pemrosesan gambar untuk CV.

konfigurasi drift

Untuk beban kerja, konfigurasi berubah dari status yang diharapkan. Ini dapat menyebabkan beban kerja menjadi tidak patuh, dan biasanya bertahap dan tidak disengaja.

database manajemen konfigurasi (CMDB)

Repositori yang menyimpan dan mengelola informasi tentang database dan lingkungan TI, termasuk komponen perangkat keras dan perangkat lunak dan konfigurasinya. Anda biasanya menggunakan data dari CMDB dalam penemuan portofolio dan tahap analisis migrasi.

paket kesesuaian

Kumpulan AWS Config aturan dan tindakan remediasi yang dapat Anda kumpulkan untuk menyesuaikan kepatuhan dan pemeriksaan keamanan Anda. Anda dapat menerapkan paket kesesuaian sebagai entitas tunggal di Akun AWS dan Region, atau di seluruh organisasi, dengan menggunakan templat YAMM. Untuk informasi selengkapnya, lihat [Paket kesesuaian dalam dokumentasi](#). AWS Config

integrasi berkelanjutan dan pengiriman berkelanjutan (CI/CD)

Proses mengotomatiskan sumber, membangun, menguji, pementasan, dan tahap produksi dari proses rilis perangkat lunak. CI/CD biasanya digambarkan sebagai pipa. CI/CD dapat membantu Anda mengotomatiskan proses, meningkatkan produktivitas, meningkatkan kualitas kode, dan memberikan lebih cepat. Untuk informasi lebih lanjut, lihat [Manfaat pengiriman berkelanjutan](#). CD juga dapat berarti penerapan berkelanjutan. Untuk informasi selengkapnya, lihat [Continuous Delivery vs Continuous Deployment](#).

CV

Lihat [visi komputer](#).

D

data saat istirahat

Data yang stasioner di jaringan Anda, seperti data yang ada di penyimpanan.

klasifikasi data

Proses untuk mengidentifikasi dan mengkategorikan data dalam jaringan Anda berdasarkan kekritisannya dan sensitivitasnya. Ini adalah komponen penting dari setiap strategi manajemen risiko keamanan siber karena membantu Anda menentukan perlindungan dan kontrol retensi yang tepat untuk data. Klasifikasi data adalah komponen pilar keamanan dalam AWS Well-Architected Framework. Untuk informasi selengkapnya, lihat [Klasifikasi data](#).

penyimpangan data

Variasi yang berarti antara data produksi dan data yang digunakan untuk melatih model ML, atau perubahan yang berarti dalam data input dari waktu ke waktu. Penyimpangan data dapat mengurangi kualitas, akurasi, dan keadilan keseluruhan dalam prediksi model ML.

data dalam transit

Data yang aktif bergerak melalui jaringan Anda, seperti antara sumber daya jaringan.

jala data

Kerangka arsitektur yang menyediakan kepemilikan data terdistribusi dan terdesentralisasi dengan manajemen dan tata kelola terpusat.

minimalisasi data

Prinsip pengumpulan dan pemrosesan hanya data yang sangat diperlukan. Mempraktikkan minimalisasi data di dalamnya AWS Cloud dapat mengurangi risiko privasi, biaya, dan jejak karbon analitik Anda.

perimeter data

Satu set pagar pembatas pencegahan di AWS lingkungan Anda yang membantu memastikan bahwa hanya identitas tepercaya yang mengakses sumber daya tepercaya dari jaringan yang diharapkan. Untuk informasi selengkapnya, lihat [Membangun perimeter data pada AWS](#).

prapemrosesan data

Untuk mengubah data mentah menjadi format yang mudah diuraikan oleh model ML Anda. Preprocessing data dapat berarti menghapus kolom atau baris tertentu dan menangani nilai yang hilang, tidak konsisten, atau duplikat.

asal data

Proses melacak asal dan riwayat data sepanjang siklus hidupnya, seperti bagaimana data dihasilkan, ditransmisikan, dan disimpan.

subjek data

Individu yang datanya dikumpulkan dan diproses.

gudang data

Sistem manajemen data yang mendukung intelijen bisnis, seperti analitik. Gudang data biasanya berisi sejumlah besar data historis, dan biasanya digunakan untuk kueri dan analisis.

bahasa definisi database (DDL)

Pernyataan atau perintah untuk membuat atau memodifikasi struktur tabel dan objek dalam database.

bahasa manipulasi basis data (DHTML)

Pernyataan atau perintah untuk memodifikasi (memasukkan, memperbarui, dan menghapus) informasi dalam database.

DDL

Lihat [bahasa definisi database](#).

ansambel yang dalam

Untuk menggabungkan beberapa model pembelajaran mendalam untuk prediksi. Anda dapat menggunakan ansambel dalam untuk mendapatkan prediksi yang lebih akurat atau untuk memperkirakan ketidakpastian dalam prediksi.

pembelajaran mendalam

Subbidang ML yang menggunakan beberapa lapisan jaringan saraf tiruan untuk mengidentifikasi pemetaan antara data input dan variabel target yang diinginkan.

pertahanan-mendalam

Pendekatan keamanan informasi di mana serangkaian mekanisme dan kontrol keamanan dilapisi dengan cermat di seluruh jaringan komputer untuk melindungi kerahasiaan, integritas, dan ketersediaan jaringan dan data di dalamnya. Saat Anda mengadopsi strategi ini AWS, Anda menambahkan beberapa kontrol pada lapisan AWS Organizations struktur yang berbeda untuk membantu mengamankan sumber daya. Misalnya, pendekatan defense-in-depth mungkin menggabungkan otentikasi multi-faktor, segmentasi jaringan, dan enkripsi.

administrator yang didelegasikan

Di AWS Organizations, layanan yang kompatibel dapat mendaftarkan akun AWS anggota untuk mengelola akun organisasi dan mengelola izin untuk layanan tersebut. Akun ini disebut administrator yang didelegasikan untuk layanan itu. Untuk informasi selengkapnya dan daftar layanan yang kompatibel, lihat [Layanan yang berfungsi dengan AWS Organizations](#) AWS Organizations dokumentasi.

deployment

Proses pembuatan aplikasi, fitur baru, atau perbaikan kode tersedia di lingkungan target. Deployment melibatkan penerapan perubahan dalam basis kode dan kemudian membangun dan menjalankan basis kode itu di lingkungan aplikasi.

lingkungan pengembangan

Lihat [lingkungan](#).

kontrol detektif

Kontrol keamanan yang dirancang untuk mendeteksi, mencatat, dan memperingatkan setelah suatu peristiwa terjadi. Kontrol ini adalah garis pertahanan kedua, memperingatkan Anda tentang peristiwa keamanan yang melewati kontrol pencegahan yang ada. Untuk informasi selengkapnya, lihat Kontrol [Detektif dalam Menerapkan kontrol](#) keamanan pada. AWS

pemetaan aliran nilai pengembangan (DVSM)

Sebuah proses yang digunakan untuk mengidentifikasi dan memprioritaskan kendala yang mempengaruhi kecepatan dan kualitas dalam siklus hidup pengembangan perangkat lunak. DVSM memperluas proses pemetaan aliran nilai yang awalnya dirancang untuk praktik manufaktur ramping. Ini berfokus pada langkah-langkah dan tim yang diperlukan untuk menciptakan dan memindahkan nilai melalui proses pengembangan perangkat lunak.

kembar digital

Representasi virtual dari sistem dunia nyata, seperti bangunan, pabrik, peralatan industri, atau jalur produksi. Kembar digital mendukung pemeliharaan prediktif, pemantauan jarak jauh, dan optimalisasi produksi.

tabel dimensi

Dalam [skema bintang](#), tabel yang lebih kecil yang berisi atribut data tentang data kuantitatif dalam tabel fakta. Atribut tabel dimensi biasanya bidang teks atau angka diskrit yang berperilaku seperti teks. Atribut ini biasanya digunakan untuk pembatasan kueri, pemfilteran, dan pelabelan set hasil.

musibah

Peristiwa yang mencegah beban kerja atau sistem memenuhi tujuan bisnisnya di lokasi utama yang digunakan. Peristiwa ini dapat berupa bencana alam, kegagalan teknis, atau akibat dari tindakan manusia, seperti kesalahan konfigurasi yang tidak disengaja atau serangan malware.

pemulihan bencana (DR)

Strategi dan proses yang Anda gunakan untuk meminimalkan downtime dan kehilangan data yang disebabkan oleh [bencana](#). Untuk informasi selengkapnya, lihat [Disaster Recovery of Workloads on AWS: Recovery in the Cloud](#) in the AWS Well-Architected Framework.

DML~

Lihat [bahasa manipulasi database](#).

desain berbasis domain

Pendekatan untuk mengembangkan sistem perangkat lunak yang kompleks dengan menghubungkan komponennya ke domain yang berkembang, atau tujuan bisnis inti, yang dilayani setiap komponen. Konsep ini diperkenalkan oleh Eric Evans dalam bukunya, *Domain-Driven Design: Tackling Complexity in the Heart of Software* (Boston: Addison-Wesley Professional, 2003). Untuk informasi tentang cara menggunakan desain berbasis domain dengan pola gambar pencekik, lihat *Memodernisasi layanan [web Microsoft ASP.NET \(ASMX\) lama](#) secara bertahap menggunakan container dan Amazon API Gateway*.

DR

Lihat [pemulihan bencana](#).

deteksi drift

Melacak penyimpangan dari konfigurasi dasar. Misalnya, Anda dapat menggunakan AWS CloudFormation untuk [mendeteksi penyimpangan dalam sumber daya sistem](#), atau Anda dapat menggunakannya AWS Control Tower untuk [mendeteksi perubahan di landing zone](#) yang mungkin memengaruhi kepatuhan terhadap persyaratan tata kelola.

DVSM

Lihat [pemetaan aliran nilai pengembangan](#).

E

EDA

Lihat [analisis data eksplorasi](#).

EDI

Lihat [pertukaran data elektronik](#).

komputasi tepi

Teknologi yang meningkatkan daya komputasi untuk perangkat pintar di tepi jaringan IoT. Jika dibandingkan dengan [komputasi awan](#), komputasi tepi dapat mengurangi latensi komunikasi dan meningkatkan waktu respons.

pertukaran data elektronik (EDI)

Pertukaran otomatis dokumen bisnis antar organisasi. Untuk informasi selengkapnya, lihat [Apa itu Pertukaran Data Elektronik](#).

enkripsi

Proses komputasi yang mengubah data plaintext, yang dapat dibaca manusia, menjadi ciphertext.

kunci enkripsi

String kriptografi dari bit acak yang dihasilkan oleh algoritma enkripsi. Panjang kunci dapat bervariasi, dan setiap kunci dirancang agar tidak dapat diprediksi dan unik.

endianness

Urutan byte disimpan dalam memori komputer. Big-endian sistem menyimpan byte paling signifikan terlebih dahulu. Little-endian sistem menyimpan byte paling tidak signifikan terlebih dahulu.

titik akhir

Lihat [titik akhir layanan](#).

layanan endpoint

Layanan yang dapat Anda host di cloud pribadi virtual (VPC) untuk dibagikan dengan pengguna lain. Anda dapat membuat layanan endpoint dengan AWS PrivateLink dan memberikan izin kepada prinsipal lain Akun AWS atau ke AWS Identity and Access Management (IAM). Akun atau prinsipal ini dapat terhubung ke layanan endpoint Anda secara pribadi dengan membuat titik akhir VPC antarmuka. Untuk informasi selengkapnya, lihat [Membuat layanan titik akhir](#) di dokumentasi Amazon Virtual Private Cloud (Amazon VPC).

perencanaan sumber daya perusahaan (ERP)

Sistem yang mengotomatiskan dan mengelola proses bisnis utama (seperti akuntansi, [MES](#), dan manajemen proyek) untuk suatu perusahaan.

enkripsi amplop

Proses mengenkripsi kunci enkripsi dengan kunci enkripsi lain. Untuk informasi selengkapnya, lihat [Enkripsi amplop](#) dalam dokumentasi AWS Key Management Service (AWS KMS).

lingkungan

Sebuah contoh dari aplikasi yang sedang berjalan. Berikut ini adalah jenis lingkungan yang umum dalam komputasi awan:

- **Development Environment** — Sebuah contoh dari aplikasi yang berjalan yang hanya tersedia untuk tim inti yang bertanggung jawab untuk memelihara aplikasi. Lingkungan pengembangan digunakan untuk menguji perubahan sebelum mempromosikannya ke lingkungan atas. Jenis lingkungan ini kadang-kadang disebut sebagai lingkungan pengujian.

- lingkungan yang lebih rendah — Semua lingkungan pengembangan untuk aplikasi, seperti yang digunakan untuk build awal dan pengujian.
- lingkungan produksi — Sebuah contoh dari aplikasi yang berjalan yang dapat diakses oleh pengguna akhir. Dalam sebuah CI/CD pipeline, lingkungan produksi adalah lingkungan penyebaran terakhir.
- lingkungan atas — Semua lingkungan yang dapat diakses oleh pengguna selain tim pengembangan inti. Ini dapat mencakup lingkungan produksi, lingkungan praproduksi, dan lingkungan untuk pengujian penerimaan pengguna.

epik

Dalam metodologi tangkas, kategori fungsional yang membantu mengatur dan memprioritaskan pekerjaan Anda. Epik memberikan deskripsi tingkat tinggi tentang persyaratan dan tugas implementasi. Misalnya, epos keamanan AWS CAF mencakup manajemen identitas dan akses, kontrol detektif, keamanan infrastruktur, perlindungan data, dan respons insiden. Untuk informasi selengkapnya tentang epos dalam strategi AWS migrasi, lihat [panduan implementasi program](#).

ERP

Lihat [perencanaan sumber daya perusahaan](#).

analisis data eksplorasi (EDA)

Proses menganalisis dataset untuk memahami karakteristik utamanya. Anda mengumpulkan atau mengumpulkan data dan kemudian melakukan penyelidikan awal untuk menemukan pola, mendeteksi anomali, dan memeriksa asumsi. EDA dilakukan dengan menghitung statistik ringkasan dan membuat visualisasi data.

F

tabel fakta

Tabel tengah dalam [skema bintang](#). Ini menyimpan data kuantitatif tentang operasi bisnis. Biasanya, tabel fakta berisi dua jenis kolom: kolom yang berisi ukuran dan yang berisi kunci asing ke tabel dimensi.

gagal cepat

Filosofi yang menggunakan pengujian yang sering dan bertahap untuk mengurangi siklus hidup pengembangan. Ini adalah bagian penting dari pendekatan tangkas.

batas isolasi kesalahan

Dalam AWS Cloud, batas seperti Availability Zone, Wilayah AWS, control plane, atau data plane yang membatasi efek kegagalan dan membantu meningkatkan ketahanan beban kerja. Untuk informasi selengkapnya, lihat [Batas Isolasi AWS Kesalahan](#).

cabang fitur

Lihat [cabang](#).

fitur

Data input yang Anda gunakan untuk membuat prediksi. Misalnya, dalam konteks manufaktur, fitur bisa berupa gambar yang diambil secara berkala dari lini manufaktur.

pentingnya fitur

Seberapa signifikan fitur untuk prediksi model. Ini biasanya dinyatakan sebagai skor numerik yang dapat dihitung melalui berbagai teknik, seperti Shapley Additive Explanations (SHAP) dan gradien terintegrasi. Untuk informasi lebih lanjut, lihat [Interpretabilitas model pembelajaran mesin](#) dengan AWS

transformasi fitur

Untuk mengoptimalkan data untuk proses ML, termasuk memperkaya data dengan sumber tambahan, menskalakan nilai, atau mengekstrak beberapa set informasi dari satu bidang data. Hal ini memungkinkan model ML untuk mendapatkan keuntungan dari data. Misalnya, jika Anda memecah tanggal "2021-05-27 00:15:37" menjadi "2021", "Mei", "Kamis", dan "15", Anda dapat membantu algoritme pembelajaran mempelajari pola bernuansa yang terkait dengan komponen data yang berbeda.

beberapa tembakan mendorong

Menyediakan [LLM](#) dengan sejumlah kecil contoh yang menunjukkan tugas dan output yang diinginkan sebelum memintanya untuk melakukan tugas serupa. Teknik ini adalah aplikasi pembelajaran dalam konteks, di mana model belajar dari contoh (bidikan) yang tertanam dalam petunjuk. Few-shot prompt bisa efektif untuk tugas-tugas yang membutuhkan pemformatan, penalaran, atau pengetahuan domain tertentu. Lihat juga [bidikan nol](#).

FGAC

Lihat kontrol [akses berbutir halus](#).

kontrol akses berbutir halus (FGAC)

Penggunaan beberapa kondisi untuk mengizinkan atau menolak permintaan akses.

migrasi flash-cut

Metode migrasi database yang menggunakan replikasi data berkelanjutan melalui [pengambilan data perubahan](#) untuk memigrasikan data dalam waktu sesingkat mungkin, alih-alih menggunakan pendekatan bertahap. Tujuannya adalah untuk menjaga downtime seminimal mungkin.

FM

Lihat [model pondasi](#).

model pondasi (FM)

Jaringan saraf pembelajaran mendalam yang besar yang telah melatih kumpulan data besar-besaran data umum dan tidak berlabel. FM mampu melakukan berbagai tugas umum, seperti memahami bahasa, menghasilkan teks dan gambar, dan berbicara dalam bahasa alami. Untuk informasi selengkapnya, lihat [Apa itu Model Foundation](#).

Gerbang FM

[Perantara terpusat yang mengontrol dan menormalkan akses ke model pondasi](#). Juga dikenal sebagai gateway LLM.

G

AI generatif

Subset model [AI](#) yang telah dilatih pada sejumlah besar data dan yang dapat menggunakan prompt teks sederhana untuk membuat konten dan artefak baru, seperti gambar, video, teks, dan audio. Untuk informasi lebih lanjut, lihat [Apa itu AI Generatif](#).

pemblokiran geografis

Lihat [pembatasan geografis](#).

pembatasan geografis (pemblokiran geografis)

Di Amazon CloudFront, opsi untuk mencegah pengguna di negara tertentu mengakses distribusi konten. Anda dapat menggunakan daftar izinkan atau daftar blokir untuk menentukan negara yang disetujui dan dilarang. Untuk informasi selengkapnya, lihat [Membatasi distribusi geografis konten Anda](#) dalam dokumentasi. CloudFront

Alur kerja Gitflow

Pendekatan di mana lingkungan bawah dan atas menggunakan cabang yang berbeda dalam repositori kode sumber. Alur kerja Gitflow dianggap warisan, dan [alur kerja berbasis batang](#) adalah pendekatan modern yang lebih disukai.

gambar emas

Sebuah snapshot dari sistem atau perangkat lunak yang digunakan sebagai template untuk menyebarkan instance baru dari sistem atau perangkat lunak itu. Misalnya, di bidang manufaktur, gambar emas dapat digunakan untuk menyediakan perangkat lunak pada beberapa perangkat dan membantu meningkatkan kecepatan, skalabilitas, dan produktivitas dalam operasi manufaktur perangkat.

strategi greenfield

Tidak adanya infrastruktur yang ada di lingkungan baru. [Saat mengadopsi strategi greenfield untuk arsitektur sistem, Anda dapat memilih semua teknologi baru tanpa batasan kompatibilitas dengan infrastruktur yang ada, juga dikenal sebagai brownfield.](#) Jika Anda memperluas infrastruktur yang ada, Anda dapat memadukan strategi brownfield dan greenfield.

pagar pembatas

Aturan tingkat tinggi yang membantu mengatur sumber daya, kebijakan, dan kepatuhan di seluruh unit organisasi (OU). Pagar pembatas preventif menegakkan kebijakan untuk memastikan keselarasan dengan standar kepatuhan. Mereka diimplementasikan dengan menggunakan kebijakan kontrol layanan dan batas izin IAM. Detective guardrails mendeteksi pelanggaran kebijakan dan masalah kepatuhan, dan menghasilkan peringatan untuk remediasi. Mereka diimplementasikan dengan menggunakan AWS Config, AWS Security Hub CSPM, Amazon GuardDuty AWS Trusted Advisor, Amazon Inspector, dan pemeriksaan khusus AWS Lambda .

pagar pembatas (AI)

Mekanisme keamanan yang menyaring, memvalidasi, dan membatasi input dan output [agen](#) untuk membantu memastikan perilaku AI yang bertanggung jawab dan aman.

H

HA

Lihat [ketersediaan tinggi](#).

migrasi database heterogen

Memigrasi database sumber Anda ke database target yang menggunakan mesin database yang berbeda (misalnya, Oracle ke Amazon Aurora). Migrasi heterogen biasanya merupakan bagian dari upaya arsitektur ulang, dan mengubah skema dapat menjadi tugas yang kompleks. [AWS menyediakan AWS SCT](#) yang membantu dengan konversi skema.

ketersediaan tinggi (HA)

Kemampuan beban kerja untuk beroperasi terus menerus, tanpa intervensi, jika terjadi tantangan atau bencana. Sistem HA dirancang untuk gagal secara otomatis, secara konsisten memberikan kinerja berkualitas tinggi, dan menangani beban dan kegagalan yang berbeda dengan dampak kinerja minimal.

modernisasi sejarawan

Pendekatan yang digunakan untuk memodernisasi dan meningkatkan sistem teknologi operasional (OT) untuk melayani kebutuhan industri manufaktur dengan lebih baik. Sejarawan adalah jenis database yang digunakan untuk mengumpulkan dan menyimpan data dari berbagai sumber di pabrik.

data penahanan

Sebagian dari data historis berlabel yang ditahan dari kumpulan data yang digunakan untuk melatih model pembelajaran [mesin](#). Anda dapat menggunakan data penahanan untuk mengevaluasi kinerja model dengan membandingkan prediksi model dengan data penahanan.

manusia-dalam-lingkaran (HiTL)

Pola alur kerja di mana eksekusi [agen](#) berhenti untuk peninjauan dan persetujuan manusia pada titik keputusan kritis.

migrasi database homogen

Memigrasi database sumber Anda ke database target yang berbagi mesin database yang sama (misalnya, Microsoft SQL Server ke Amazon RDS for SQL Server). Migrasi homogen biasanya merupakan bagian dari upaya rehosting atau replatforming. Anda dapat menggunakan utilitas database asli untuk memigrasi skema.

data panas

Data yang sering diakses, seperti data real-time atau data translasi terbaru. Data ini biasanya memerlukan tingkat atau kelas penyimpanan berkinerja tinggi untuk memberikan respons kueri yang cepat.

perbaikan terbaru

Perbaikan mendesak untuk masalah kritis dalam lingkungan produksi. Karena urgensinya, perbaikan terbaru biasanya dibuat di luar alur kerja DevOps rilis biasa.

periode hypercare

Segera setelah cutover, periode waktu ketika tim migrasi mengelola dan memantau aplikasi yang dimigrasi di cloud untuk mengatasi masalah apa pun. Biasanya, periode ini panjangnya 1-4 hari. Pada akhir periode hypercare, tim migrasi biasanya mentransfer tanggung jawab untuk aplikasi ke tim operasi cloud.

I

IAC

Lihat [infrastruktur sebagai kode](#).

kebijakan berbasis identitas

Kebijakan yang dilampirkan pada satu atau beberapa prinsip IAM yang mendefinisikan izin mereka dalam lingkungan. AWS Cloud

aplikasi idle

Aplikasi yang memiliki penggunaan CPU dan memori rata-rata antara 5 dan 20 persen selama periode 90 hari. Dalam proyek migrasi, adalah umum untuk menghentikan aplikasi ini atau mempertahankannya di tempat.

IIoT

Lihat [Internet of Things industri](#).

infrastruktur yang tidak dapat diubah

Model yang menyebarkan infrastruktur baru untuk beban kerja produksi alih-alih memperbarui, menambal, atau memodifikasi infrastruktur yang ada. [Infrastruktur yang tidak dapat diubah secara inheren lebih konsisten, andal, dan dapat diprediksi daripada infrastruktur yang dapat berubah](#). Untuk informasi selengkapnya, lihat praktik terbaik [Deploy using immutable infrastructure](#) in the Framework. AWS Well-Architected

masuk (masuknya) VPC

Dalam arsitektur AWS multi-akun, VPC yang menerima, memeriksa, dan merutekan koneksi jaringan dari luar aplikasi. [Arsitektur Referensi AWS Keamanan](#) merekomendasikan pengaturan

akun Jaringan Anda dengan VPC masuk, keluar, dan inspeksi untuk melindungi antarmuka dua arah antara aplikasi Anda dan internet yang lebih luas.

migrasi inkremental

Strategi cutover di mana Anda memigrasikan aplikasi Anda dalam bagian-bagian kecil alih-alih melakukan satu cutover penuh. Misalnya, Anda mungkin hanya memindahkan beberapa layanan mikro atau pengguna ke sistem baru pada awalnya. Setelah Anda memverifikasi bahwa semuanya berfungsi dengan baik, Anda dapat secara bertahap memindahkan layanan mikro atau pengguna tambahan hingga Anda dapat menonaktifkan sistem lama Anda. Strategi ini mengurangi risiko yang terkait dengan migrasi besar.

Industri 4.0

Sebuah istilah yang diperkenalkan oleh [Klaus Schwab](#) pada tahun 2016 untuk merujuk pada modernisasi proses manufaktur melalui kemajuan dalam konektivitas, data real-time, otomatisasi, analitik, dan AI/ML.

infrastruktur

Semua sumber daya dan aset yang terkandung dalam lingkungan aplikasi.

infrastruktur sebagai kode (IAC)

Proses penyediaan dan pengelolaan infrastruktur aplikasi melalui satu set file konfigurasi. IAC dirancang untuk membantu Anda memusatkan manajemen infrastruktur, menstandarisasi sumber daya, dan menskalakan dengan cepat sehingga lingkungan baru dapat diulang, andal, dan konsisten.

Internet of Things industri (IIoT)

Penggunaan sensor dan perangkat yang terhubung ke internet di sektor industri, seperti manufaktur, energi, otomotif, perawatan kesehatan, ilmu kehidupan, dan pertanian. Untuk informasi selengkapnya, lihat [Membangun strategi transformasi digital Internet of Things \(IIoT\) industri](#).

inspeksi VPC

Dalam arsitektur AWS multi-akun, VPC terpusat yang mengelola inspeksi lalu lintas jaringan antara VPC (dalam hal yang sama atau berbeda Wilayah AWS), internet, dan jaringan lokal. [Arsitektur Referensi AWS Keamanan](#) merekomendasikan pengaturan akun Jaringan Anda dengan VPC masuk, keluar, dan inspeksi untuk melindungi antarmuka dua arah antara aplikasi Anda dan internet yang lebih luas.

Internet of Things (IoT)

Jaringan objek fisik yang terhubung dengan sensor atau prosesor tertanam yang berkomunikasi dengan perangkat dan sistem lain melalui internet atau melalui jaringan komunikasi lokal. Untuk informasi selengkapnya, lihat [Apa itu IoT?](#)

interpretabilitas

Karakteristik model pembelajaran mesin yang menggambarkan sejauh mana manusia dapat memahami bagaimana prediksi model bergantung pada inputnya. Untuk informasi lebih lanjut, lihat [Interpretabilitas model pembelajaran mesin](#) dengan. AWS

IoT

Lihat [Internet of Things](#).

Perpustakaan informasi TI (ITIL)

Serangkaian praktik terbaik untuk memberikan layanan TI dan menyelaraskan layanan ini dengan persyaratan bisnis. ITIL menyediakan dasar untuk ITSM.

Manajemen layanan TI (ITSM)

Kegiatan yang terkait dengan merancang, menerapkan, mengelola, dan mendukung layanan TI untuk suatu organisasi. Untuk informasi tentang mengintegrasikan operasi cloud dengan alat ITSM, lihat panduan [integrasi operasi](#).

ITIL

Lihat [perpustakaan informasi TI](#).

ITSM

Lihat [manajemen layanan TI](#).

L

kontrol akses berbasis label (LBAC)

Implementasi kontrol akses wajib (MAC) di mana pengguna dan data itu sendiri masing-masing secara eksplisit diberi nilai label keamanan. Persimpangan antara label keamanan pengguna dan label keamanan data menentukan baris dan kolom mana yang dapat dilihat oleh pengguna.

landing zone

Landing zone adalah AWS lingkungan multi-akun yang dirancang dengan baik yang dapat diskalakan dan aman. Ini adalah titik awal dari mana organisasi Anda dapat dengan cepat meluncurkan dan menyebarkan beban kerja dan aplikasi dengan percaya diri dalam lingkungan keamanan dan infrastruktur mereka. Untuk informasi selengkapnya tentang zona pendaratan, lihat [Menyiapkan lingkungan multi-akun AWS yang aman dan dapat diskalakan](#).

model bahasa besar (LLM)

Model [AI](#) pembelajaran mendalam yang dilatih sebelumnya pada sejumlah besar data. LLM dapat melakukan beberapa tugas, seperti menjawab pertanyaan, meringkas dokumen, menerjemahkan teks ke bahasa lain, dan menyelesaikan kalimat. Untuk informasi lebih lanjut, lihat [Apa itu LLM](#).

migrasi besar

Migrasi 300 atau lebih server.

LBAC

Lihat [kontrol akses berbasis label](#).

hak istimewa paling sedikit

Praktik keamanan terbaik untuk memberikan izin minimum yang diperlukan untuk melakukan tugas. Untuk informasi selengkapnya, lihat [Menerapkan izin hak istimewa terkecil dalam dokumentasi IAM](#).

angkat dan geser

Lihat [7 Rs](#).

sistem endian kecil

Sebuah sistem yang menyimpan byte paling tidak signifikan terlebih dahulu. Lihat juga [endianness](#).

LLM

Lihat [model bahasa besar](#).

lingkungan yang lebih rendah

Lihat [lingkungan](#).

M

pembelajaran mesin (ML)

Jenis kecerdasan buatan yang menggunakan algoritma dan teknik untuk pengenalan pola dan pembelajaran. ML menganalisis dan belajar dari data yang direkam, seperti data Internet of Things (IoT), untuk menghasilkan model statistik berdasarkan pola. Untuk informasi selengkapnya, lihat [Machine Learning](#).

cabang utama

Lihat [cabang](#).

malware

Perangkat lunak yang dirancang untuk membahayakan keamanan atau privasi komputer. Malware dapat mengganggu sistem komputer, membocorkan informasi sensitif, atau mendapatkan akses yang tidak sah. Contoh malware termasuk virus, worm, ransomware, Trojan horse, spyware, dan keyloggers.

layanan terkelola

Layanan AWS yang AWS mengoperasikan lapisan infrastruktur, sistem operasi, dan platform, dan Anda mengakses titik akhir untuk menyimpan dan mengambil data. Amazon Simple Storage Service (Amazon S3) dan Amazon DynamoDB adalah contoh layanan terkelola. Ini juga dikenal sebagai layanan abstrak.

sistem eksekusi manufaktur (MES)

Sistem perangkat lunak untuk melacak, memantau, mendokumentasikan, dan mengendalikan proses produksi yang mengubah bahan baku menjadi produk jadi di lantai toko.

PETA

Lihat [Program Percepatan Migrasi](#).

MCP

Lihat [Protokol Konteks Model](#).

Protokol Konteks Model (MCP)

Protokol stateless untuk komunikasi [agen](#) -to- [alat](#).

Server MCP

Layanan yang mengekspos satu atau lebih [alat](#) melalui [Protokol Konteks Model](#).

mekanisme

Proses lengkap di mana Anda membuat alat, mendorong adopsi alat, dan kemudian memeriksa hasilnya untuk melakukan penyesuaian. Mekanisme adalah siklus yang memperkuat dan meningkatkan dirinya sendiri saat beroperasi. Untuk informasi selengkapnya, lihat [Membangun mekanisme](#) dalam AWS Well-Architected Kerangka Kerja.

akun anggota

Semua Akun AWS selain akun manajemen yang merupakan bagian dari organisasi di AWS Organizations. Akun dapat menjadi anggota dari hanya satu organisasi pada suatu waktu.

MES

Lihat [sistem eksekusi manufaktur](#).

Transportasi Telemetri Antrian Pesan (MQTT)

[Protokol komunikasi mesin-ke-mesin \(M2M\) yang ringan, berdasarkan pola publish/subscribe, untuk perangkat IoT yang dibatasi sumber daya.](#)

layanan mikro

Layanan kecil dan independen yang berkomunikasi melalui API yang terdefinisi dengan baik dan biasanya dimiliki oleh tim kecil yang mandiri. Misalnya, sistem asuransi mungkin mencakup layanan mikro yang memetakan kemampuan bisnis, seperti penjualan atau pemasaran, atau subdomain, seperti pembelian, klaim, atau analitik. Manfaat layanan mikro termasuk kelincahan, penskalaan yang fleksibel, penyebaran yang mudah, kode yang dapat digunakan kembali, dan ketahanan. Untuk informasi selengkapnya, lihat [Mengintegrasikan layanan mikro dengan menggunakan layanan tanpa AWS server](#).

arsitektur microservices

Pendekatan untuk membangun aplikasi dengan komponen independen yang menjalankan setiap proses aplikasi sebagai layanan mikro. Layanan mikro ini berkomunikasi melalui antarmuka yang terdefinisi dengan baik dengan menggunakan API ringan. Setiap layanan mikro dalam arsitektur ini dapat diperbarui, digunakan, dan diskalakan untuk memenuhi permintaan fungsi tertentu dari suatu aplikasi. Untuk informasi selengkapnya, lihat [Menerapkan layanan mikro di AWS](#).

Program Percepatan Migrasi (MAP)

AWS Program yang menyediakan dukungan konsultasi, pelatihan, dan layanan untuk membantu organisasi membangun fondasi operasional yang kuat untuk pindah ke cloud, dan untuk membantu mengimbangi biaya awal migrasi. MAP mencakup metodologi migrasi untuk

mengeksekusi migrasi lama dengan cara metodis dan seperangkat alat untuk mengotomatisasi dan mempercepat skenario migrasi umum.

migrasi dalam skala

Proses memindahkan sebagian besar portofolio aplikasi ke cloud dalam gelombang, dengan lebih banyak aplikasi bergerak pada tingkat yang lebih cepat di setiap gelombang. Fase ini menggunakan praktik terbaik dan pelajaran yang dipetik dari fase sebelumnya untuk mengimplementasikan pabrik migrasi tim, alat, dan proses untuk merampingkan migrasi beban kerja melalui otomatisasi dan pengiriman tangkas. Ini adalah fase ketiga dari [strategi AWS migrasi](#).

pabrik migrasi

Cross-functional tim yang merampingkan migrasi beban kerja melalui pendekatan otomatis dan gesit. Tim pabrik migrasi biasanya mencakup operasi, analis dan pemilik bisnis, insinyur migrasi, pengembang, dan DevOps profesional yang bekerja di sprint. Antara 20 dan 50 persen portofolio aplikasi perusahaan terdiri dari pola berulang yang dapat dioptimalkan dengan pendekatan pabrik. Untuk informasi selengkapnya, lihat [diskusi tentang pabrik migrasi](#) dan [panduan Pabrik Migrasi Cloud](#) di kumpulan konten ini.

metadata migrasi

Informasi tentang aplikasi dan server yang diperlukan untuk menyelesaikan migrasi. Setiap pola migrasi memerlukan satu set metadata migrasi yang berbeda. Contoh metadata migrasi termasuk subnet target, grup keamanan, dan akun. AWS

pola migrasi

Tugas migrasi berulang yang merinci strategi migrasi, tujuan migrasi, dan aplikasi atau layanan migrasi yang digunakan. Contoh: Rehost migrasi ke Amazon EC2 AWS dengan Layanan Migrasi Aplikasi.

Penilaian Portofolio Migrasi (MPA)

Alat online yang menyediakan informasi untuk memvalidasi kasus bisnis untuk bermigrasi ke. AWS Cloud MPA menyediakan penilaian portofolio terperinci (ukuran kanan server, harga, perbandingan TCO, analisis biaya migrasi) serta perencanaan migrasi (analisis data aplikasi dan pengumpulan data, pengelompokan aplikasi, prioritas migrasi, dan perencanaan gelombang). [Alat MPA](#) (memerlukan login) tersedia gratis untuk semua AWS konsultan dan konsultan APN Partner.

Penilaian Kesiapan Migrasi (MRA)

Proses mendapatkan wawasan tentang status kesiapan cloud organisasi, mengidentifikasi kekuatan dan kelemahan, dan membangun rencana aksi untuk menutup kesenjangan yang diidentifikasi, menggunakan CAF. AWS Untuk informasi selengkapnya, lihat [panduan kesiapan migrasi](#). MRA adalah tahap pertama dari [strategi AWS migrasi](#).

strategi migrasi

Pendekatan yang digunakan untuk memigrasikan beban kerja ke. AWS Cloud Untuk informasi lebih lanjut, lihat entri [7 Rs](#) di glosarium ini dan lihat [Memobilisasi organisasi Anda untuk mempercepat](#) migrasi skala besar.

ML

Lihat [pembelajaran mesin](#).

modernisasi

Mengubah aplikasi usang (warisan atau monolitik) dan infrastrukturnya menjadi sistem yang gesit, elastis, dan sangat tersedia di cloud untuk mengurangi biaya, mendapatkan efisiensi, dan memanfaatkan inovasi. Untuk informasi selengkapnya, lihat [Strategi untuk memodernisasi aplikasi di. AWS Cloud](#)

penilaian kesiapan modernisasi

Evaluasi yang membantu menentukan kesiapan modernisasi aplikasi organisasi; mengidentifikasi manfaat, risiko, dan dependensi; dan menentukan seberapa baik organisasi dapat mendukung keadaan masa depan aplikasi tersebut. Hasil penilaian adalah cetak biru arsitektur target, peta jalan yang merinci fase pengembangan dan tonggak untuk proses modernisasi, dan rencana aksi untuk mengatasi kesenjangan yang diidentifikasi. Untuk informasi lebih lanjut, lihat [Mengevaluasi kesiapan modernisasi untuk](#) aplikasi di. AWS Cloud

aplikasi monolitik (monolit)

Aplikasi yang berjalan sebagai layanan tunggal dengan proses yang digabungkan secara ketat. Aplikasi monolitik memiliki beberapa kelemahan. Jika satu fitur aplikasi mengalami lonjakan permintaan, seluruh arsitektur harus diskalakan. Menambahkan atau meningkatkan fitur aplikasi monolitik juga menjadi lebih kompleks ketika basis kode tumbuh. Untuk mengatasi masalah ini, Anda dapat menggunakan arsitektur microservices. Untuk informasi lebih lanjut, lihat [Mengurai monolit](#) menjadi layanan mikro.

MPA

Lihat [Penilaian Portofolio Migrasi](#).

MQTT

Lihat [Transportasi Telemetry Antrian Pesan](#).

klasifikasi multiclass

Sebuah proses yang membantu menghasilkan prediksi untuk beberapa kelas (memprediksi satu dari lebih dari dua hasil). Misalnya, model ML mungkin bertanya “Apakah produk ini buku, mobil, atau telepon?” atau “Kategori produk mana yang paling menarik bagi pelanggan ini?”

infrastruktur yang bisa berubah

Model yang memperbarui dan memodifikasi infrastruktur yang ada untuk beban kerja produksi. Untuk meningkatkan konsistensi, keandalan, dan prediktabilitas, AWS Well-Architected Framework merekomendasikan penggunaan [infrastruktur yang tidak dapat diubah](#) sebagai praktik terbaik.

O

OAC

Lihat [kontrol akses asal](#).

OAI

Lihat [identitas akses asal](#).

OCM

Lihat [manajemen perubahan organisasi](#).

migrasi offline

Metode migrasi di mana beban kerja sumber diturunkan selama proses migrasi. Metode ini melibatkan waktu henti yang diperpanjang dan biasanya digunakan untuk beban kerja kecil dan tidak kritis.

OI

Lihat [integrasi operasi](#).

OLA

Lihat [perjanjian tingkat operasional](#).

migrasi online

Metode migrasi di mana beban kerja sumber disalin ke sistem target tanpa diambil offline. Aplikasi yang terhubung ke beban kerja dapat terus berfungsi selama migrasi. Metode ini melibatkan waktu henti nol hingga minimal dan biasanya digunakan untuk beban kerja produksi yang kritis.

OPC-UA

Lihat [Komunikasi Proses Terbuka - Arsitektur Terpadu](#).

Komunikasi Proses Terbuka - Arsitektur Terpadu () OPC-UA

Protokol komunikasi mesin-ke-mesin (M2M) untuk otomasi industri. OPC-UA menyediakan standar interoperabilitas dengan enkripsi data, otentikasi, dan skema otorisasi.

perjanjian tingkat operasional (OLA)

Perjanjian yang menjelaskan apa yang dijanjikan kelompok TI fungsional untuk diberikan satu sama lain, untuk mendukung perjanjian tingkat layanan (SLA).

Tinjauan Kesiapan Operasional (ORR)

Daftar pertanyaan dan praktik terbaik terkait yang membantu Anda memahami, mengevaluasi, mencegah, atau mengurangi ruang lingkup insiden dan kemungkinan kegagalan. Untuk informasi selengkapnya, lihat [Ulasan Kesiapan Operasional \(ORR\) dalam Kerangka Kerja AWS Well-Architected](#)

teknologi operasional (OT)

Sistem perangkat keras dan perangkat lunak yang bekerja dengan lingkungan fisik untuk mengendalikan operasi industri, peralatan, dan infrastruktur. Di bidang manufaktur, integrasi sistem OT dan teknologi informasi (TI) adalah fokus utama untuk transformasi [Industri 4.0](#).

integrasi operasi (OI)

Proses modernisasi operasi di cloud, yang melibatkan perencanaan kesiapan, otomatisasi, dan integrasi. Untuk informasi selengkapnya, lihat [panduan integrasi operasi](#).

jejak organisasi

Jejak yang dibuat oleh AWS CloudTrail itu mencatat semua peristiwa untuk semua Akun AWS dalam organisasi di AWS Organizations. Jejak ini dibuat di setiap Akun AWS bagian organisasi dan melacak aktivitas di setiap akun. Untuk informasi selengkapnya, lihat [Membuat jejak untuk organisasi](#) dalam CloudTrail dokumentasi.

manajemen perubahan organisasi (OCM)

Kerangka kerja untuk mengelola transformasi bisnis utama yang mengganggu dari perspektif orang, budaya, dan kepemimpinan. OCM membantu organisasi mempersiapkan, dan transisi ke, sistem dan strategi baru dengan mempercepat adopsi perubahan, mengatasi masalah transisi, dan mendorong perubahan budaya dan organisasi. Dalam strategi AWS migrasi, kerangka kerja ini disebut percepatan orang, karena kecepatan perubahan yang diperlukan dalam proyek adopsi cloud. Untuk informasi lebih lanjut, lihat [panduan OCM](#).

kontrol akses asal (OAC)

Di CloudFront, opsi yang disempurnakan untuk membatasi akses untuk mengamankan konten Amazon Simple Storage Service (Amazon S3) Anda. OAC mendukung semua bucket S3 di semua Wilayah AWS, enkripsi sisi server dengan AWS KMS (SSE-KMS), dan dinamis PUT dan DELETE permintaan ke bucket S3.

identitas akses asal (OAI)

Di CloudFront, opsi untuk membatasi akses untuk mengamankan konten Amazon S3 Anda. Saat Anda menggunakan OAI, CloudFront buat prinsipal yang dapat diautentikasi oleh Amazon S3. Prinsipal yang diautentikasi dapat mengakses konten dalam bucket S3 hanya melalui distribusi tertentu. CloudFront Lihat juga [OAC](#), yang menyediakan kontrol akses yang lebih terperinci dan ditingkatkan.

ORR

Lihat [tinjauan kesiapan operasional](#).

OT

Lihat [teknologi operasional](#).

keluar (jalan keluar) VPC

Dalam arsitektur AWS multi-akun, VPC yang menangani koneksi jaringan yang dimulai dari dalam aplikasi. [Arsitektur Referensi AWS Keamanan](#) merekomendasikan pengaturan akun Jaringan Anda dengan VPC masuk, keluar, dan inspeksi untuk melindungi antarmuka dua arah antara aplikasi Anda dan internet yang lebih luas.

P

batas izin

Kebijakan manajemen IAM yang dilampirkan pada prinsipal IAM untuk menetapkan izin maksimum yang dapat dimiliki pengguna atau peran. Untuk informasi selengkapnya, lihat [Batas izin](#) dalam dokumentasi IAM.

Informasi Identifikasi Pribadi (PII)

Informasi yang, jika dilihat secara langsung atau dipasangkan dengan data terkait lainnya, dapat digunakan untuk menyimpulkan identitas individu secara wajar. Contoh PII termasuk nama, alamat, dan informasi kontak.

PII

Lihat informasi yang [dapat diidentifikasi secara pribadi](#).

buku pedoman

Serangkaian langkah yang telah ditentukan sebelumnya yang menangkap pekerjaan yang terkait dengan migrasi, seperti mengirimkan fungsi operasi inti di cloud. Buku pedoman dapat berupa skrip, runbook otomatis, atau ringkasan proses atau langkah-langkah yang diperlukan untuk mengoperasikan lingkungan modern Anda.

PLC

Lihat [pengontrol logika yang dapat diprogram](#).

PLM

Lihat [manajemen siklus hidup produk](#).

kebijakan

[Objek yang dapat menentukan izin \(lihat kebijakan berbasis identitas\), menentukan kondisi akses \(lihat kebijakan berbasis sumber daya\), atau menentukan izin maksimum untuk semua akun dalam organisasi di \(lihat kebijakan kontrol layanan\). AWS Organizations](#)

persistensi poliglot

Secara independen memilih teknologi penyimpanan data microservice berdasarkan pola akses data dan persyaratan lainnya. Jika layanan mikro Anda memiliki teknologi penyimpanan data yang sama, mereka dapat menghadapi tantangan implementasi atau mengalami kinerja yang buruk. Layanan mikro lebih mudah diimplementasikan dan mencapai kinerja dan skalabilitas yang lebih baik jika mereka menggunakan penyimpanan data yang paling sesuai dengan kebutuhan mereka.

penilaian portofolio

Proses menemukan, menganalisis, dan memprioritaskan portofolio aplikasi untuk merencanakan migrasi. Untuk informasi selengkapnya, lihat [Mengevaluasi kesiapan migrasi](#).

predikat

Kondisi kueri yang mengembalikan `true` atau `false`, biasanya terletak di `WHERE` klausa.

predikat pushdown

Teknik pengoptimalan kueri database yang menyaring data dalam kueri sebelum transfer. Ini mengurangi jumlah data yang harus diambil dan diproses dari database relasional, dan meningkatkan kinerja kueri.

kontrol preventif

Kontrol keamanan yang dirancang untuk mencegah suatu peristiwa terjadi. Kontrol ini adalah garis pertahanan pertama untuk membantu mencegah akses tidak sah atau perubahan yang tidak diinginkan ke jaringan Anda. Untuk informasi selengkapnya, lihat [Kontrol pencegahan dalam Menerapkan kontrol](#) keamanan pada AWS.

principal

Entitas AWS yang dapat melakukan tindakan dan mengakses sumber daya. Entitas ini biasanya merupakan pengguna root untuk Akun AWS, peran IAM, atau pengguna. Untuk informasi selengkapnya, lihat Prinsip dalam [istilah dan konsep Peran](#) dalam dokumentasi IAM.

privasi berdasarkan desain

Pendekatan rekayasa sistem yang memperhitungkan privasi melalui seluruh proses pengembangan.

zona host pribadi

Container yang menyimpan informasi tentang bagaimana Anda ingin Amazon Route 53 merespons kueri DNS untuk domain dan subdomainnya dalam satu atau beberapa VPC. Untuk informasi selengkapnya, lihat [Bekerja dengan zona yang dihosting pribadi](#) di dokumentasi Route 53.

kontrol proaktif

[Kontrol keamanan](#) yang dirancang untuk mencegah penyebaran sumber daya yang tidak sesuai. Kontrol ini memindai sumber daya sebelum disediakan. Jika sumber daya tidak sesuai dengan kontrol, maka itu tidak disediakan. Untuk informasi selengkapnya, lihat [panduan referensi Kontrol](#)

dalam AWS Control Tower dokumentasi dan lihat [Kontrol proaktif](#) dalam Menerapkan kontrol keamanan pada AWS.

manajemen siklus hidup produk (PLM)

Manajemen data dan proses untuk suatu produk di seluruh siklus hidupnya, mulai dari desain, pengembangan, dan peluncuran, melalui pertumbuhan dan kematangan, hingga penurunan dan penghapusan.

lingkungan produksi

Lihat [lingkungan](#).

pengontrol logika yang dapat diprogram (PLC)

Di bidang manufaktur, komputer yang sangat andal dan mudah beradaptasi yang memantau mesin dan mengotomatiskan proses manufaktur.

rantai cepat

Menggunakan output dari satu prompt [LLM](#) sebagai input untuk prompt berikutnya untuk menghasilkan respons yang lebih baik. Teknik ini digunakan untuk memecah tugas yang kompleks menjadi subtugas, atau untuk secara iteratif memperbaiki atau memperluas respons awal. Ini membantu meningkatkan akurasi dan relevansi respons model dan memungkinkan hasil yang lebih terperinci dan dipersonalisasi.

pseudonimisasi

Proses penggantian pengenalan pribadi dalam kumpulan data dengan nilai placeholder. Pseudonimisasi dapat membantu melindungi privasi pribadi. Data pseudonim masih dianggap sebagai data pribadi.

publish/subscribe (pub/sub)

Pola yang memungkinkan komunikasi asinkron antara layanan mikro untuk meningkatkan skalabilitas dan daya tanggap. Misalnya, dalam [MES](#) berbasis layanan mikro, layanan mikro dapat mempublikasikan pesan peristiwa ke saluran yang dapat berlangganan layanan mikro lainnya. Sistem dapat menambahkan layanan mikro baru tanpa mengubah layanan penerbitan.

Q

rencana kueri

Serangkaian langkah, seperti instruksi, yang digunakan untuk mengakses data dalam sistem database relasional SQL.

regresi rencana kueri

Ketika pengoptimal layanan database memilih rencana yang kurang optimal daripada sebelum perubahan yang diberikan ke lingkungan database. Hal ini dapat disebabkan oleh perubahan statistik, kendala, pengaturan lingkungan, pengikatan parameter kueri, dan pembaruan ke mesin database.

R

Matriks RACI

Lihat [bertanggung jawab, akuntabel, dikonsultasikan, diinformasikan \(RACI\)](#).

LAP

Lihat [Retrieval Augmented Generation](#).

ransomware

Perangkat lunak berbahaya yang dirancang untuk memblokir akses ke sistem komputer atau data sampai pembayaran dilakukan.

Matriks RASCI

Lihat [bertanggung jawab, akuntabel, dikonsultasikan, diinformasikan \(RACI\)](#).

RCAC

Lihat [kontrol akses baris dan kolom](#).

replika baca

Salinan database yang digunakan untuk tujuan read-only. Anda dapat merutekan kueri ke replika baca untuk mengurangi beban pada database utama Anda.

arsitek ulang

Lihat [7 Rs](#).

tujuan titik pemulihan (RPO)

Jumlah waktu maksimum yang dapat diterima sejak titik pemulihan data terakhir. Ini menentukan apa yang dianggap sebagai kehilangan data yang dapat diterima antara titik pemulihan terakhir dan gangguan layanan.

tujuan waktu pemulihan (RTO)

Penundaan maksimum yang dapat diterima antara gangguan layanan dan pemulihan layanan.

refactor

Lihat [7 Rs](#).

Region

Kumpulan AWS sumber daya di wilayah geografis. Masing-masing Wilayah AWS terisolasi dan independen dari yang lain untuk memberikan toleransi kesalahan, stabilitas, dan ketahanan. Untuk informasi selengkapnya, lihat [Menentukan Wilayah AWS akun yang dapat digunakan](#).

regresi

Teknik ML yang memprediksi nilai numerik. Misalnya, untuk memecahkan masalah “Berapa harga rumah ini akan dijual?” Model ML dapat menggunakan model regresi linier untuk memprediksi harga jual rumah berdasarkan fakta yang diketahui tentang rumah (misalnya, luas persegi).

rehost

Lihat [7 Rs](#).

melepaskan

Dalam proses penyebaran, tindakan mempromosikan perubahan pada lingkungan produksi.

memindahkan

Lihat [7 Rs](#).

memplatform ulang

Lihat [7 Rs](#).

pembelian kembali

Lihat [7 Rs](#).

ketahanan

Kemampuan aplikasi untuk melawan atau pulih dari gangguan. [Ketersediaan tinggi](#) dan [pemulihan bencana](#) adalah pertimbangan umum ketika merencanakan ketahanan di AWS Cloud. Untuk informasi lebih lanjut, lihat [AWS Cloud Ketahanan](#).

kebijakan berbasis sumber daya

Kebijakan yang dilampirkan ke sumber daya, seperti bucket Amazon S3, titik akhir, atau kunci enkripsi. Jenis kebijakan ini menentukan prinsipal mana yang diizinkan mengakses, tindakan yang didukung, dan kondisi lain yang harus dipenuhi.

matriks yang bertanggung jawab, akuntabel, dikonsultasikan, diinformasikan (RACI)

Matriks yang mendefinisikan peran dan tanggung jawab untuk semua pihak yang terlibat dalam kegiatan migrasi dan operasi cloud. Nama matriks berasal dari jenis tanggung jawab yang didefinisikan dalam matriks: bertanggung jawab (R), akuntabel (A), dikonsultasikan (C), dan diinformasikan (I). Jenis dukungan (S) adalah opsional. Jika Anda menyertakan dukungan, matriks disebut matriks RASCI, dan jika Anda mengecualikannya, itu disebut matriks RACI.

kontrol responsif

Kontrol keamanan yang dirancang untuk mendorong remediasi efek samping atau penyimpangan dari garis dasar keamanan Anda. Untuk informasi selengkapnya, lihat [Kontrol responsif](#) dalam Menerapkan kontrol keamanan pada AWS.

melestarikan

Lihat [7 Rs](#).

pensiun

Lihat [7 Rs](#).

Retrieval Augmented Generation (RAG)

Teknologi [AI generatif](#) di mana [LLM](#) mereferensikan sumber data otoritatif yang berada di luar sumber data pelatihannya sebelum menghasilkan respons. Misalnya, model RAG mungkin melakukan pencarian semantik dari basis pengetahuan organisasi atau data kustom. Untuk informasi lebih lanjut, lihat [Apa itu RAG](#).

rotasi

Proses memperbarui [rahasia](#) secara berkala untuk membuatnya lebih sulit bagi penyerang untuk mengakses kredensial.

kontrol akses baris dan kolom (RCAC)

Penggunaan ekspresi SQL dasar dan fleksibel yang telah menetapkan aturan akses. RCAC terdiri dari izin baris dan topeng kolom.

RPO

Lihat [tujuan titik pemulihan](#).

RTO

Lihat [tujuan waktu pemulihan](#).

buku runbook

Satu set prosedur manual atau otomatis yang diperlukan untuk melakukan tugas tertentu. Ini biasanya dibangun untuk merampingkan operasi berulang atau prosedur dengan tingkat kesalahan yang tinggi.

D

SAML 2.0

Standar terbuka yang digunakan oleh banyak penyedia identitas (IdPs). Fitur ini memungkinkan sistem masuk tunggal gabungan (SSO), sehingga pengguna dapat masuk ke Konsol Manajemen AWS atau memanggil operasi AWS API tanpa Anda harus membuat pengguna di IAM untuk semua orang di organisasi Anda. Untuk informasi lebih lanjut tentang federasi berbasis SAMP 2.0, lihat [Tentang federasi berbasis SAMP 2.0](#) dalam dokumentasi IAM.

SCADA

Lihat [kontrol pengawasan dan akuisisi data](#).

SCP

Lihat [kebijakan kontrol layanan](#).

Rahasia

Dalam AWS Secrets Manager, informasi rahasia atau terbatas, seperti kata sandi atau kredensial pengguna, yang Anda simpan dalam bentuk terenkripsi. Ini terdiri dari nilai rahasia dan metadatanya. Nilai rahasia dapat berupa biner, string tunggal, atau beberapa string. Untuk informasi selengkapnya, lihat [Apa yang ada di rahasia Secrets Manager?](#) dalam dokumentasi Secrets Manager.

keamanan dengan desain

Pendekatan rekayasa sistem yang memperhitungkan keamanan melalui seluruh proses pengembangan.

kontrol keamanan

Pagar pembatas teknis atau administratif yang mencegah, mendeteksi, atau mengurangi kemampuan pelaku ancaman untuk mengeksploitasi kerentanan keamanan. [Ada empat jenis kontrol keamanan utama: preventif, detektif, responsif, dan proaktif.](#)

pengerasan keamanan

Proses mengurangi permukaan serangan untuk membuatnya lebih tahan terhadap serangan. Ini dapat mencakup tindakan seperti menghapus sumber daya yang tidak lagi diperlukan, menerapkan praktik keamanan terbaik untuk memberikan hak istimewa paling sedikit, atau menonaktifkan fitur yang tidak perlu dalam file konfigurasi.

sistem informasi keamanan dan manajemen acara (SIEM)

Alat dan layanan yang menggabungkan sistem manajemen informasi keamanan (SIM) dan manajemen acara keamanan (SEM). Sistem SIEM mengumpulkan, memantau, dan menganalisis data dari server, jaringan, perangkat, dan sumber lain untuk mendeteksi ancaman dan pelanggaran keamanan, dan untuk menghasilkan peringatan.

otomatisasi respons keamanan

Tindakan yang telah ditentukan dan diprogram yang dirancang untuk secara otomatis merespons atau memulihkan peristiwa keamanan. Otomatisasi ini berfungsi sebagai kontrol keamanan [detektif](#) atau [responsif](#) yang membantu Anda menerapkan praktik terbaik AWS keamanan. Contoh tindakan respons otomatis termasuk memodifikasi grup keamanan VPC, menambal instans Amazon EC2, atau memutar kredensial.

enkripsi sisi server

Enkripsi data di tujuannya, oleh Layanan AWS yang menerimanya.

kebijakan kontrol layanan (SCP)

Kebijakan yang menyediakan kontrol terpusat atas izin untuk semua akun di organisasi. AWS Organizations SCP menentukan pagar pembatas atau menetapkan batasan pada tindakan yang dapat didelegasikan oleh administrator kepada pengguna atau peran. Anda dapat menggunakan SCP sebagai daftar izin atau daftar penolakan, untuk menentukan layanan atau tindakan mana

yang diizinkan atau dilarang. Untuk informasi selengkapnya, lihat [Kebijakan kontrol layanan](#) dalam AWS Organizations dokumentasi.

titik akhir layanan

URL titik masuk untuk file Layanan AWS. Anda dapat menggunakan endpoint untuk terhubung secara terprogram ke layanan target. Untuk informasi selengkapnya, lihat [Layanan AWS titik akhir](#) di Referensi Umum AWS.

perjanjian tingkat layanan (SLA)

Perjanjian yang menjelaskan apa yang dijanjikan oleh tim TI untuk diberikan kepada pelanggan mereka, seperti uptime dan kinerja layanan.

indikator tingkat layanan (SLI)

Pengukuran aspek kinerja layanan, seperti tingkat kesalahan, ketersediaan, atau throughputnya.

tujuan tingkat layanan (SLO)

Metrik target yang mewakili kesehatan layanan, yang diukur dengan indikator [tingkat layanan](#).

model tanggung jawab bersama

Model yang menjelaskan tanggung jawab yang Anda bagikan AWS untuk keamanan dan kepatuhan cloud. AWS bertanggung jawab atas keamanan cloud, sedangkan Anda bertanggung jawab atas keamanan di cloud. Untuk informasi selengkapnya, lihat [Model tanggung jawab bersama](#).

Bayangan AI

Aplikasi [AI](#) yang tidak sah dibuat atau digunakan di luar saluran yang diatur dalam suatu organisasi.

SIEM

Lihat [informasi keamanan dan sistem manajemen acara](#).

titik kegagalan tunggal (SPOF)

Kegagalan dalam satu komponen penting dari aplikasi yang dapat mengganggu sistem.

SLA

Lihat [perjanjian tingkat layanan](#).

SLI

Lihat [indikator tingkat layanan](#).

SLO

Lihat [tujuan tingkat layanan](#).

model split-and-seed

Pola untuk menskalakan dan mempercepat proyek modernisasi. Ketika fitur baru dan rilis produk didefinisikan, tim inti berpisah untuk membuat tim produk baru. Ini membantu meningkatkan kemampuan dan layanan organisasi Anda, meningkatkan produktivitas pengembang, dan mendukung inovasi yang cepat. Untuk informasi lebih lanjut, lihat [Pendekatan bertahap untuk memodernisasi aplikasi](#) di AWS Cloud

SPOF

Lihat [satu titik kegagalan](#).

skema bintang

Struktur organisasi database yang menggunakan satu tabel fakta besar untuk menyimpan data transaksional atau terukur dan menggunakan satu atau lebih tabel dimensi yang lebih kecil untuk menyimpan atribut data. Struktur ini dirancang untuk digunakan dalam [gudang data](#) atau untuk tujuan intelijen bisnis.

pola ara pencekik

Pendekatan untuk memodernisasi sistem monolitik dengan menulis ulang secara bertahap dan mengganti fungsionalitas sistem sampai sistem warisan dapat dinonaktifkan. Pola ini menggunakan analogi pohon ara yang tumbuh menjadi pohon yang sudah mapan dan akhirnya mengatasi dan menggantikan inangnya. Pola ini [diperkenalkan oleh Martin Fowler](#) sebagai cara untuk mengelola risiko saat menulis ulang sistem monolitik. Untuk contoh cara menerapkan pola ini, lihat [Memodernisasi layanan web ASP.NET Microsoft \(ASMX\) lama secara bertahap menggunakan container dan Amazon API Gateway](#).

subnet

Rentang alamat IP dalam VPC Anda. Subnet harus berada di Availability Zone tunggal.

kontrol pengawasan dan akuisisi data (SCADA)

Di bidang manufaktur, sistem yang menggunakan perangkat keras dan perangkat lunak untuk memantau aset fisik dan operasi produksi.

enkripsi simetris

Algoritma enkripsi yang menggunakan kunci yang sama untuk mengenkripsi dan mendekripsi data.

pengujian sintetis

Menguji sistem dengan cara yang mensimulasikan interaksi pengguna untuk mendeteksi potensi masalah atau untuk memantau kinerja. Anda dapat menggunakan [Amazon CloudWatch Synthetics](#) untuk membuat tes ini.

sistem prompt

Teknik untuk memberikan konteks, instruksi, atau pedoman ke [LLM](#) untuk mengarahkan perilakunya. Permintaan sistem membantu mengatur konteks dan menetapkan aturan untuk interaksi dengan pengguna.

T

tag

Key-value pasangan yang bertindak sebagai metadata untuk mengatur sumber daya Anda AWS . Tag membantu Anda mengelola, mengidentifikasi, mengatur, dan memfilter sumber daya. Untuk informasi selengkapnya, lihat [Menandai sumber daya AWS](#).

variabel target

Nilai yang Anda coba prediksi dalam ML yang diawasi. Ini juga disebut sebagai variabel hasil. Misalnya, dalam pengaturan manufaktur, variabel target bisa menjadi cacat produk.

daftar tugas

Alat yang digunakan untuk melacak kemajuan melalui runbook. Daftar tugas berisi ikhtisar runbook dan daftar tugas umum yang harus diselesaikan. Untuk setiap tugas umum, itu termasuk perkiraan jumlah waktu yang dibutuhkan, pemilik, dan kemajuan.

lingkungan uji

Lihat [lingkungan](#).

pelatihan

Untuk menyediakan data bagi model ML Anda untuk dipelajari. Data pelatihan harus berisi jawaban yang benar. Algoritma pembelajaran menemukan pola dalam data pelatihan yang memetakan atribut data input ke target (jawaban yang ingin Anda prediksi). Ini menghasilkan model ML yang menangkap pola-pola ini. Anda kemudian dapat menggunakan model ML untuk membuat prediksi pada data baru yang Anda tidak tahu targetnya.

alat

Fungsi atau API yang dapat [dipanggil agen](#) untuk melakukan operasi di sistem eksternal.

gerbang transit

Hub transit jaringan yang dapat Anda gunakan untuk menghubungkan VPC dan jaringan lokal Anda. Untuk informasi selengkapnya, lihat [Apa itu gateway transit](#) dalam AWS Transit Gateway dokumentasi.

alur kerja berbasis batang

Pendekatan di mana pengembang membangun dan menguji fitur secara lokal di cabang fitur dan kemudian menggabungkan perubahan tersebut ke cabang utama. Cabang utama kemudian dibangun untuk pengembangan, praproduksi, dan lingkungan produksi, secara berurutan.

akses tepercaya

Memberikan izin ke layanan yang Anda tentukan untuk melakukan tugas di organisasi Anda di dalam AWS Organizations dan di akunnya atas nama Anda. Layanan tepercaya menciptakan peran terkait layanan di setiap akun, ketika peran itu diperlukan, untuk melakukan tugas manajemen untuk Anda. Untuk informasi selengkapnya, lihat [Menggunakan AWS Organizations dengan AWS layanan lain](#) dalam AWS Organizations dokumentasi.

penyetelan

Untuk mengubah aspek proses pelatihan Anda untuk meningkatkan akurasi model ML. Misalnya, Anda dapat melatih model ML dengan membuat set pelabelan, menambahkan label, dan kemudian mengulangi langkah-langkah ini beberapa kali di bawah pengaturan yang berbeda untuk mengoptimalkan model.

tim dua pizza

Sebuah DevOps tim kecil yang bisa Anda beri makan dengan dua pizza. Ukuran tim dua pizza memastikan peluang terbaik untuk berkolaborasi dalam pengembangan perangkat lunak.

U

waswas

Sebuah konsep yang mengacu pada informasi yang tidak tepat, tidak lengkap, atau tidak diketahui yang dapat merusak keandalan model ML prediktif. Ada dua jenis ketidakpastian:

ketidakpastian epistemik disebabkan oleh data yang terbatas dan tidak lengkap, sedangkan ketidakpastian aleatorik disebabkan oleh kebisingan dan keacakan yang melekat dalam data.

tugas yang tidak terdiferensiasi

Juga dikenal sebagai angkat berat, pekerjaan yang diperlukan untuk membuat dan mengoperasikan aplikasi tetapi itu tidak memberikan nilai langsung kepada pengguna akhir atau memberikan keunggulan kompetitif. Contoh tugas yang tidak terdiferensiasi termasuk pengadaan, pemeliharaan, dan perencanaan kapasitas.

lingkungan atas

Lihat [lingkungan](#).

V

menyedot debu

Operasi pemeliharaan database yang melibatkan pembersihan setelah pembaruan tambahan untuk merebut kembali penyimpanan dan meningkatkan kinerja.

kendali versi

Proses dan alat yang melacak perubahan, seperti perubahan kode sumber dalam repositori.

Peering VPC

Koneksi antara dua VPC yang memungkinkan Anda merutekan lalu lintas dengan menggunakan alamat IP pribadi. Untuk informasi selengkapnya, lihat [Apa itu peering VPC](#) di dokumentasi VPC Amazon.

kerentanan

Kelemahan perangkat lunak atau perangkat keras yang membahayakan keamanan sistem.

W

cache hangat

Cache buffer yang berisi data terkini dan relevan yang sering diakses. Instance database dapat membaca dari cache buffer, yang lebih cepat daripada membaca dari memori utama atau disk.

data hangat

Data yang jarang diakses. Saat menanyakan jenis data ini, kueri yang cukup lambat biasanya dapat diterima.

fungsi jendela

Fungsi SQL yang melakukan perhitungan pada sekelompok baris yang berhubungan dengan catatan saat ini. Fungsi jendela berguna untuk memproses tugas, seperti menghitung rata-rata bergerak atau mengakses nilai baris berdasarkan posisi relatif dari baris saat ini.

beban kerja

Kumpulan sumber daya dan kode yang memberikan nilai bisnis, seperti aplikasi yang dihadapi pelanggan atau proses backend.

aliran kerja

Grup fungsional dalam proyek migrasi yang bertanggung jawab atas serangkaian tugas tertentu. Setiap alur kerja independen tetapi mendukung alur kerja lain dalam proyek. Misalnya, alur kerja portofolio bertanggung jawab untuk memprioritaskan aplikasi, perencanaan gelombang, dan mengumpulkan metadata migrasi. Alur kerja portofolio mengirimkan aset ini ke alur kerja migrasi, yang kemudian memigrasikan server dan aplikasi.

CACING

Lihat [menulis sekali, baca banyak](#).

WQF

Lihat [AWS Kerangka Kualifikasi Beban Kerja](#).

tulis sekali, baca banyak (WORM)

Model penyimpanan yang menulis data satu kali dan mencegah data dihapus atau dimodifikasi. Pengguna yang berwenang dapat membaca data sebanyak yang diperlukan, tetapi mereka tidak dapat mengubahnya. Infrastruktur penyimpanan data ini dianggap [tidak dapat diubah](#).

Z

eksploitasi zero-day

Serangan, biasanya malware, yang memanfaatkan kerentanan [zero-day](#).

kerentanan zero-day

Cacat atau kerentanan yang tak tanggung-tanggung dalam sistem produksi. Aktor ancaman dapat menggunakan jenis kerentanan ini untuk menyerang sistem. Pengembang sering menyadari kerentanan sebagai akibat dari serangan tersebut.

bidikan zero-shot

Memberikan [LLM](#) dengan instruksi untuk melakukan tugas tetapi tidak ada contoh (tembakkan) yang dapat membantu membimbingnya. LLM harus menggunakan pengetahuan pra-terlatih untuk menangani tugas. Efektivitas bidikan nol tergantung pada kompleksitas tugas dan kualitas prompt. Lihat juga beberapa [bidikan yang diminta](#).

aplikasi zombie

Aplikasi yang memiliki CPU rata-rata dan penggunaan memori di bawah 5 persen. Dalam proyek migrasi, adalah umum untuk menghentikan aplikasi ini.

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.