의 모델 컨텍스트 프로토콜 전략 AWS

AWS 권장 가이드

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

AWS 권장 가이드: 의 모델 컨텍스트 프로토콜 전략 AWS

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

Amazon의 상표 및 트레이드 드레스는 Amazon 외 제품 또는 서비스와 함께, Amazon 브랜드 이미지를 떨어뜨리거나 고객에게 혼동을 일으킬 수 있는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

Table of Contents

소개 1
대상 독자 2
목표 2
MCP란 무엇입니까? 4
도구 이해 4
MCP를 사용해야 하는 경우 7
MCP 도구 설계 전략 10
도구 범위 10
세분화 10
대략적 11
MCP 도구 크기 조정 모범 사례 13
도구 정의 14
도구 사양 접근 방식 14
Docstring 접근 방식 15
MCP 도구 정의 모범 사례 16
도구 검색 16
정적 정의 16
동적 검색 17
검색 함수 17
MCP 도구 검색 모범 사례 18
도구 구성 18
MPC 도구 조직에 대한 모범 사례 19
MCP 호스팅 전략 20
호스팅 접근 방식 20
로컬 호스팅 20
원격 호스팅 21
MCP 게이트웨이 22
MCP 서버 호스팅 모범 사례 22
MCP 거버넌스 전략 23
인증 및 권한 부여 23
MCP 인증 및 권한 부여 모범 사례 24
로드 제어 25
로드 제어 모범 사례 25
운영 지표 26

기여자 27
작성 27
검토 27
기술 작성 27
문서 기록 28
..... xxix

의 모델 컨텍스트 프로토콜 전략 AWS

Amazon Web Services(기여자)

2026년 3월(문서 기록)

이 가이드는 조직 전체에서 모델 컨텍스트 프로토콜(MCP) 전략을 개발하고 구현하여 에이전트 AI 여정을 지원하는 데 도움이 될 수 있습니다. 에이전트 및 언어 모델이 비즈니스 운영의 핵심이 됨에 따라 성공적인 에이전트 솔루션을 위해서는 MCP 전략을 수립하는 것이 중요합니다.

이 가이드에서는 MCP 전략을 구축하기 위한 세 가지 기본 원칙인 MCP 도구 설계, MCP 서버 호스팅, MCP 거버넌스를 살펴봅니다. 이러한 상호 연결된 구성 요소를 해결하면 조직은 AI 구현 전반에서 모델 컨텍스트를 관리하기 위한 확장 가능하고 안전하며 효과적인 시스템을 만들 수 있습니다. 이 지침은 초기 실험부터 전체 규모의 프로덕션 배포에 이르기까지 조직의 AI 여정의 모든 단계에서 조직을 위한 실행 가능한 인사이트와 전략적 지침을 제공합니다. 이를 통해 특정 요구 사항 및 목표에 맞는 맞춤형 MCP 솔루션을 개발할 수 있습니다.

이러한 모범 사례는 엔터프라이즈 규모로 MCP를 배포하는 조직의 실제 구현, 현재 MCP 사양 표준 분석, 프로덕션 환경에서 사용자 지정 대규모 언어 모델(LLM) 애플리케이션에서 얻은 교훈에서 파생되었습니다.

AI 시스템은 다양한 사용 사례에서 점점 더 정교해지고 강력한 LLMs을 사용합니다. LLMs 자연어를 이해하고, 인간과 유사한 응답을 생성하고, 복잡한 정보를 추론하는 데 뛰어납니다. 그러나 대화형 인터페이스에서 복잡한 작업을 자율적으로 수행할 수 있는 시스템으로 LLMs을 변환하기 위해 조직은 에이전트 AI 아키텍처, 환경을 인식하고, 목표에 대한 이유를 파악하고, 자율 결정을 내리고, 여러 단계에서 오케스트레이션하고, 사용자를 대신하여 목표를 달성하기 위한 조치를 취할 수 있는 AI 시스템을 채택하고 있습니다. 이 에이전트 접근 방식은 조직이 자연어를 통해 사용자 의도를 이해하고, 여러 데이터 소스 및 도구에서 자율적으로 조정하고, 기존 요청-응답 패턴으로는 불가능한 규모로 개인화된 경험을 제공할 수 있는 AI 시스템을 구축하는 데 도움이 됩니다. 이러한 에이전트의 역량을 높이려면 조직은 기존 도구 및 데이터에 대한 액세스를 제공하여 에이전트의 컨텍스트 이해를 강화하고 사용자를 대신하여 작업할 수 있도록 해야 합니다.

MCP는 AI 도구 통합을 위한 표준화된 프로토콜을 제공하여 에이전트와 외부 리소스 간의 일관된 통신을 지원합니다. MCP 자체가 통신 표준을 정의하지만 효과적으로 구현하려면 확장 가능하고 안전하며 유지 관리 가능한 솔루션을 달성하기 위해 아키텍처 패턴, 보안 모델, 운영 관행 및 성능 최적화 전략을 신중하게 고려해야 합니다.

이 가이드는 엔터프라이즈 MCP 배포에서 얻은 교훈을 합성하여 AWS Well-Architected Framework에 맞는 실행 가능한 권장 사항을 제공합니다. 자체 MCP 솔루션을 구축하는 데 필수적인 MCP 도구 설

계, MCP 서버 호스팅 및 MCP 거버넌스에 대한 전략을 다룹니다. 이 가이드의 권장 사항은 AWS Well-Architected Framework의 다음 다섯 가지 원칙에 매핑됩니다.

- 보안 - 토큰 격리, 범위가 축소된 자격 증명, 별도의 읽기/쓰기 권한 부여
- 운영 우수성 - 도구 선택 정확도 지표, 회귀 테스트를 위한 골든 데이터 세트
- 신뢰성 - 사용자당 및 도구당 속도 제한, 로드 셰딩
- 성능 효율성 - 워크플로 범위 도구, 도구 필터링, 컨텍스트 창 사용량을 줄이기 위한 의미 체계 검색
- 비용 최적화 - 팀 전체에서 재사용 가능한 MCP 서버, 도구 필터링을 통한 요청당 토큰 비용 절감

대상 독자

이 가이드는 조직에서 에이전트 AI 솔루션을 구현하는 아키텍트, 개발자 및 기술 리더를 대상으로 합니다. 이 가이드의 개념을 이해하려면 LLMs 작동 방식을 이해하고 MCP, 도구 및 프롬프트 엔지니어링에 대한 기본 지식을 가지고 있어야 합니다.

목표

프로덕션 준비가 완료된 Agentic AI 시스템을 구축하려면 조직의 정책을 지원하기 위해 거버넌스, 최적화 및 보안을 함께 해결해야 합니다. 아래는이 가이드가 이러한 목표를 해결하는 방법을 설명합니다.

- 거버넌스 - 중앙 집중식 거버넌스가 없으면 어떤 에이전트가 어떤 데이터에 액세스했는지, 어떤 권한이 언제 액세스했는지를 포함하여 AI 워크로드에 대한 감사 질문에 답할 수 없습니다. 버전 관리를 적용할 수도 없습니다. 이 가이드의 MCP 호스팅 전략 섹션에서는 사용자가 체계적인 적용이 부족하여 알려진 취약성이 있는 오래된 로컬 MCP 서버를 실행하는 방법을 설명합니다.

  규제 산업의 경우 거버넌스가 중요합니다. 감사자는 단일 창에서 모든 에이전트에 대한 정책 적용 및 도구 사용량 추적을 확인하려고 합니다. MCP 거버넌스는 이를 제공합니다.

  이 가이드의 권장 사항에 따라 피어 리뷰 벤치마크에서 작업 정확도를 28~32% 개선할 수 있습니다. 자세한 내용은 MARCO: 다중 에이전트 실시간 채팅 오케스트레이션(ACL Anthology 웹 사이트)을 참조하세요. 거버넌스는 규정 준수뿐만 아니라 에이전트 AI 시스템 성능도 개선합니다.
- 최적화 - 팀이 동일한 통합을 두 번 이상 구축할 수 있습니다. 예를 들어 5개의 서로 다른 팀이 AI 애플리케이션이 데이터베이스와 통신할 수 있도록 자체 데이터베이스 쿼리 스크립트를 작성하는 경우 이는 개발 비용의 5배와 유지할 버그 목록 세트 5개입니다. MCP를 사용하면 한 번 빌드하고 전체 엔지니어링 커뮤니티에서 공유할 수 있습니다. 에이전트 수가 늘어날 때의 절감 컴파운드입니다.

또한 대부분의 팀이 처음에는 알아채지 못하는 요청당 비용 문제도 있습니다. 모든 도구 정의는 컨텍스트 창 토큰을 사용합니다. 20개의 도구에서는 사용자 문의와 함께 설명에만 호출당 5,000~10,000개의 토큰을 사용합니다. 이렇게 하면 모델이 사용 가능한 도구 목록에서 올바른 도구를 선택하는 데 어려움을 겪기 때문에 지연 시간과 LLM 추론 비용이 증가하고 정확도가 저하됩니다.

구조화된 도구 래퍼를 사용하는 에이전트는 APIs 직접 액세스하는 에이전트보다 데이터베이스 작업에서 약 3배 더 정확합니다(자세한 내용은 Middleware for LLMs: Tools Are Instrumental for Language Agents in Complex Environments 참조). AI 모델에 도구를 설계하고 제공하는 방법이 중요합니다. 이 가이드에서는 도구에 명확한 스키마를 제공하고, 원시 엔드포인트 대신 실제 워크플로로 크기를 조정하고, 컨텍스트 창에서 정보를 제한하는 것을 권장합니다. 이 가이드의 MCP 도구 설계 전략 섹션에서는 이러한 측면을 자세히 살펴봅니다.

- 보안 및 규정 준수 - 정리 단계를 할루시네이션하고 프로덕션 데이터베이스를 삭제하려고 하는 에이전트 AI 시스템을 가정해 보겠습니다. 에이전트가 사용자의 전체 관리자 자격 증명을 상속한 경우 삭제가 진행될 수 있습니다. 읽기 및 생성 액세스 권한만 부여하는 토큰 격리 및 범위가 축소된 자격 증명을 사용하면 안전하게 실패합니다.

  규제된 워크플로는 이를 더욱 선명하게 만듭니다. 이 가이드에서는 예제(환자 데이터를 처리하기 전에 HIPAA 검증 및 개인 식별 정보 익명화가 필요한 의료 파이프라인)를 제공합니다. MCP 도구에 이러한 로직을 포함하면 규정 준수가 매번 결정적으로 이루어집니다.

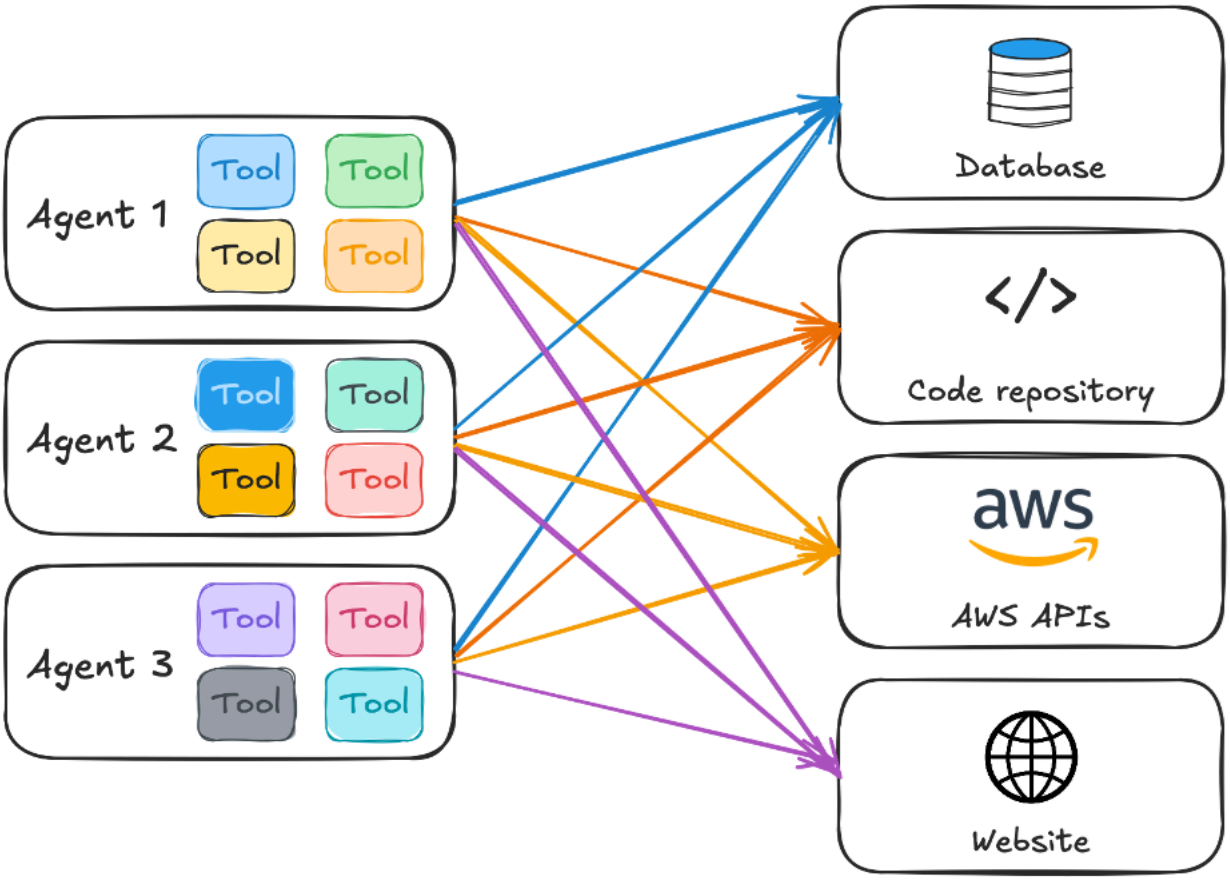
MCP란 무엇입니까?

LLMs 훈련 데이터를 기반으로 프롬프트에 대한 응답을 예측하여 작동합니다. 즉, LLM은 이미 본 데이터 및 이벤트에 대한 답변만 제공할 수 있습니다. Retrieval Augmented Generation(RAG) 및 지식 기반과 같은 방법을 사용하면 컨텍스트 데이터를 포함할 수 있습니다. 그러나 LLM에 내일의 일기 예보가 어떤 것인지 또는 데이터베이스에 몇 명의 고객이 있는지 물어보면 LLM의 사전 훈련된 지식을 벗어나기 때문에 할루시나 답변을 제공할 수 없을 가능성이 높습니다. 이러한 종류의 질문에 답하려면 에이전트가 LLM의 기본 컨텍스트 외부의 외부 기능, 데이터 및 APIs에 액세스해야 합니다.

도구 이해

도구를 통해 LLM에 추가 시스템 및 컨텍스트에 대한 액세스 권한을 부여할 수 있습니다. 도구는 명확한 목표를 달성하기 위해 LLM에 제공되는 함수입니다. 도구는 API를 호출하고, 데이터베이스를 쿼리하고, 계산기 작업을 수행하고, 코드 샌드박스를 운영하고, 웹 검색을 수행하고, 심지어 다른 AI 시스템 또는 agent-as-a-tool 호출할 수 있습니다. 각 도구에는 LLM에 도구의 기능, 사용 시기 및 수락하는 파라미터를 알려주는 설명이 포함되어야 합니다. 이를 통해 LLM은 사용자의 입력을 기반으로 호출할 도구 또는 도구 조합에 대해 미묘한 결정을 내릴 수 있습니다. 에이전트가 사용할 수 있는 도구에 대해 LLM에 알려주므로 에이전트가 도구를 호출하도록 지시하는 응답을 생성할 수 있습니다. 예를 들어 LLM에 데이터베이스에 몇 명의 고객이 있는지 묻는 경우 LLM은 특정 입력 파라미터로 `query_database` 도구를 실행하도록 요청하는 응답을 에이전트에게 다시 보냅니다. LLM은 호출할 도구와 도구 호출에 대한 입력을 결정합니다. 그런 다음 에이전트는 도구를 실행하여 자연어 입력을 구문적으로 올바른 함수 호출로 변환하고 쿼리를 실행합니다. 에이전트는 LLM의 지침에 따라 도구 또는 도구를 호출하고 해당 결과는 LLM으로 다시 전달됩니다. 이는 텍스트 기반 입력을 추론하고 작업에 적합한 도구를 선택하는 LLM의 기능을 활용합니다.

다음 이미지는 각 에이전트가 각 대상에 대한 자체 도구 세트를 관리하는 방법을 보여줍니다.



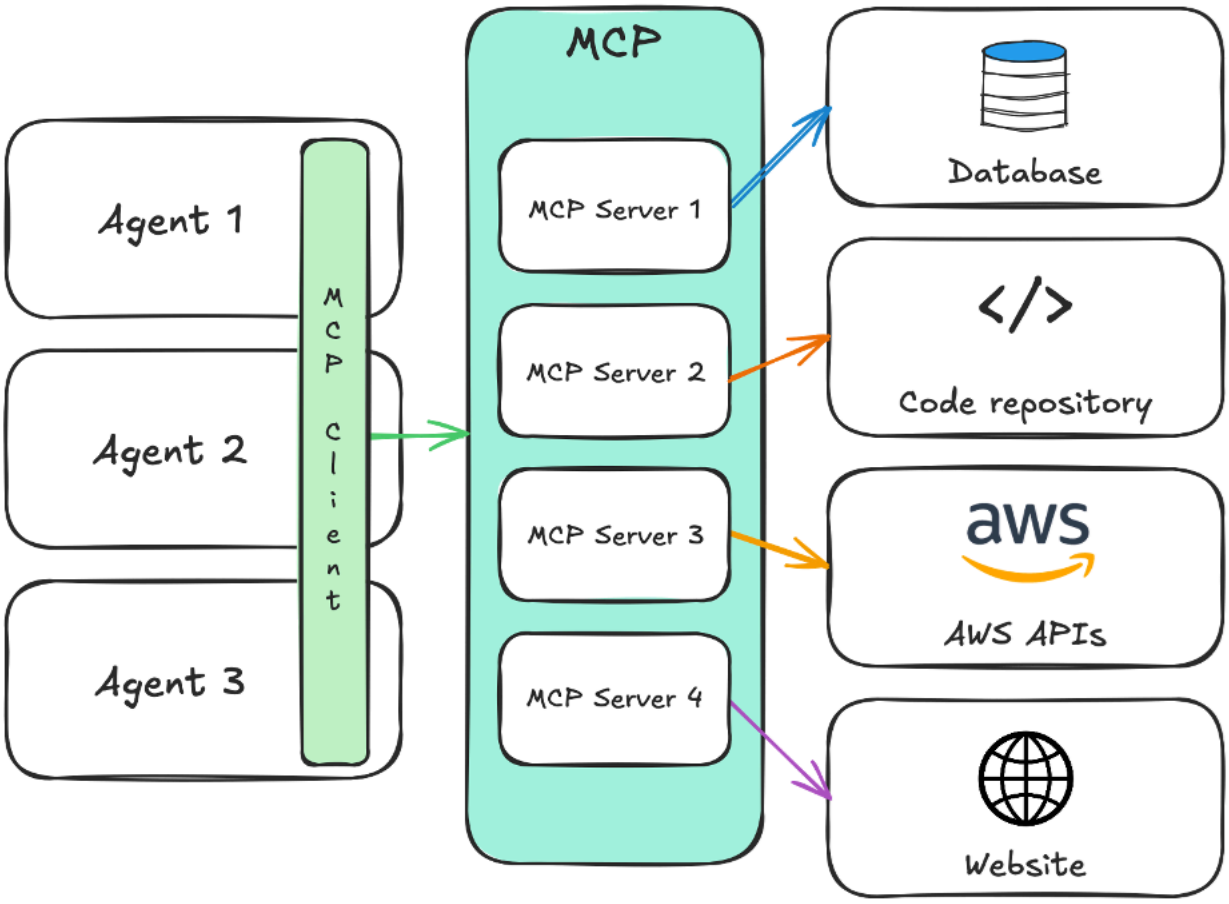
도구 액세스를 확장하면 에이전트 AI 솔루션에 문제가 발생할 수 있습니다.

- 모든 개발자가 동일한 외부 기능을 위한 자체 도구를 생성하는 경우 중복된 작업과 이러한 외부 기능과 상호 작용하는 비표준화된 방법이 많이 있습니다. 이렇게 하면 에이전트 간에 일관되지 않은 구현이 생성됩니다. 라이브러리에서 표준 도구를 개발하고 배포하여이 문제를 해결할 수 있지만 중앙 집중식 거버넌스가 부족합니다. 따라서 보안 정책을 적용하거나, 도구 사용량을 추적하거나, 팀 간 버전 관리를 관리하거나, 조직 표준을 준수하기가 어렵습니다. 또한 에이전트와 직접 도구를 임베드할 때는 새 도구가 생성되거나 기존 도구가 업데이트될 때마다 에이전트를 재배포해야 합니다.
- LLM에 도구를 제공하면 컨텍스트 창이 사용됩니다. 컨텍스트 창은 모델이 한 번에 고려할 수 있는 토큰 수(LLMs이 처리하는 텍스트 단위 - 일반적으로 단어, 단어의 일부 또는 구두점을 나타냄)입니다. LLMs에는 컨텍스트 창 제한이 있습니다. 도구 및 설명서는 시스템 프롬프트 및 사용자 프롬프트와 함께이 유한 컨텍스트 창을 사용합니다. 컨텍스트 창이 채워지면 LLMs 관련 정보 식별의 어려움, 처리 복잡성 증가, 추론 용량 감소 등 여러 요인으로 인해 성능이 저하될 수 있습니다. 챌린지는 도구 정의, 시스템 프롬프트 및 대화 기록이 각 LLM 간접 호출에 제공되므로 제한된 컨텍스트 창 공간을 놓고 경쟁할 때 복잡해집니다.

따라서 도구 수와 문서화 방법은 응답 시간 및 정확도와 같은 LLM의 성능에 직접적인 영향을 미칩니다.

MCP는 에이전트를 외부 기능에 연결하기 위한 범용 표준을 설정합니다. 일반적으로 “AI 애플리케이션용 USB-C”라고 합니다. MCP 서버는 에이전트에 직접 도구를 등록하는 대신 JSON-RPC 2.0을 통해 검색 및 호출되는 호스팅 도구를 위한 중개자 역할을 합니다. MCP를 사용하면 에이전트에 수십 또는 수백 개의 서로 다른 도구를 추가하고 시간이 지남에 따라 유지 관리하는 대신 에이전트가 액세스할 수 있는 도구를 캡슐화하는 MCP 서버를 등록할 수 있습니다. 이 접근 방식은 도구를 패키징, 제공 및 호출하는 방법을 표준화합니다. 이를 통해 에이전트 내에서 도구 사용의 규모 및 거버넌스 문제를 해결할 수 있습니다. 또한 에이전트 개발 및 작업을 외부 기능에 사용하는 도구에서 분리합니다.

다음 그림은 MCP를 사용하여 외부 리소스에 액세스하는 에이전트를 보여줍니다.



그러나 MCP 표준이 모든 조정 및 거버넌스 문제를 해결하는 것은 아닙니다. MCP 서버의 구현은 효과적인 도구 설계, 호스팅 및 엔터프라이즈 거버넌스 전략과 결합되어야 합니다. 이 가이드는 에이전트 AI 솔루션의 일부로 MCP를 구축하고 사용하는 데 도움이 되는 각 전략의 모범 사례를 제공합니다.

MCP를 사용해야 하는 경우

MCP는 에이전트 AI 이니셔티브를 확장하기 위한 전략적 인프라를 제공합니다. MCP 서버는 도구 관리 및 거버넌스를 중앙 집중화하여 여러 에이전트에서 사용자 지정 통합을 구축하고 유지 관리하는 데 드는 누적 비용을 줄입니다. 이렇게 하면 에이전트 에코시스템이 확장됨에 따라 수익이 증가합니다.

다음과 같은 경우 MCP가 전략의 일부가 될 수 있습니다.

- 에이전트가 데이터베이스, APIs, 내부 도구 및 타사 통합과 같은 엔터프라이즈 시스템 및 서비스에 액세스하는 방법에 대한 중앙 집중식 거버넌스가 필요합니다.
- 개발자는 구현 간에 일관되지 않은 사용자 지정 통합을 작성하는 데 너무 많은 시간을 소비합니다.
- 공통 기능을 제공할 수 있는 중복 도구가 있습니다.
- 표준화된 관리형 MCP 인터페이스를 통해 외부 소비자 또는 타사 에이전트 시스템에 독점 도구 또는 데이터를 제공하여 보안 및 제어를 유지하면서 새로운 수익 스트림을 확보하려고 합니다.

MCP 서버가 전략의 일부가 될 것이라고 결정한 후 기존 오픈 소스 MCP 서버 구현이 요구 사항을 충족하는지, 개선이 필요한지 또는 사용자 지정 서버를 구축해야 하는지 평가합니다. 사전 구축된 많은 MCP 서버 구현은 퍼블릭 리포지토리에서 사용할 수 있으며 파일 시스템 액세스, 웹 브라우징, 코드 샌드박스, 데이터베이스 액세스 및 API 통합과 같은 일반적인 기능을 다룹니다.

대부분의 경우 기존 MCP 서버면 충분합니다. 예를 들어AWS MCP 서버, AI 어시스턴트와 에이전트에게 자연어 상호 작용을 AWS 서비스 통해 AWS 에 대한 안전하고 인증된 액세스를 제공하는 관리형 원격 MCP 서버인를 제공합니다. AWS MCP 서버 를 사용하면 AWS 설명서에 대한 실시간 액세스, 구문적으로 올바른 API 호출, 모범 사례를 따르는 AWS 에이전트 SOPs라는 사전 구축된 워크플로를 결합하여 복잡한 다단계 AWS 작업을 수행할 수 있습니다.는 고객 에이전트가 AWS MCP 서버 를 성공적으로 사용할 수 있도록 AWS 지속적으로 테스트합니다.

에이전트와 함께 이러한 기존 MCP 서버를 테스트하여 에이전트가 사용 사례를 충족하는지 확인해야 합니다. 에이전트가 워크플로를 완료하지 못하거나, 잘못되거나 최적화되지 않은 응답을 생성하거나, 복잡한 다단계 프로세스를 탐색하지 못하거나, 중요한 도메인별 모범 사례 또는 보안 고려 사항을 놓치는 경우 여러 차원의 개선 사항을 고려해야 합니다.

기존 MCP 서버가 요구 사항을 완전히 충족하지 못하고 기존 도구를 올바르게 사용하거나 정확한 응답을 생성하는 데 어려움을 겪는 경우 사용자 지정 서버를 구축하기 전에 다음 개선 접근 방식을 고려하세요.

- 에이전트 컨텍스트 강화 - 에이전트가 기존 MCP 서버의 도구를 올바르게 또는 효율적으로 사용하는 데 어려움을 겪는 경우 해당 도구 정의를 추가 설명서 또는 예제로 보완하는 것이 좋습니다. 이를 통해 LLM에 추가 컨텍스트를 제공할 수 있습니다.
- 보완 도구 추가 - 에이전트가 워크플로를 성공적으로 완료하는 데 필요한 추가 조직 데이터 또는 컨텍스트에 액세스하는 도구로 기존 MCP 서버를 확장합니다.
- 기본 APIs 개선 - 파라미터 복잡성을 줄이고, 더 명확한 오류 메시지를 제공하고, 에이전트가 사용할 수 있는 합리적인 기본값을 제공하여 서비스 APIs를 간소화하여 LLM 친화적으로 만듭니다.

기존 MCP 서버 구현을 사용하면 일반적인 기능의 개발이 가속화되지만 사용 사례에 특수 기능이 필요한 경우 사용자 지정 MCP 서버를 구축하는 것이 필요합니다. 사용자 지정 MCP 서버는 도메인 전문 지식을 캡슐화하고, 조직 표준을 적용하고, 복잡한 워크플로에 대한 에이전트 신뢰성을 개선하고, 보안 요구 사항 준수를 지원하는 데 도움이 됩니다. 다음과 같은 상황에서 사용자 지정 MCP 서버를 구축하는 것이 좋습니다.

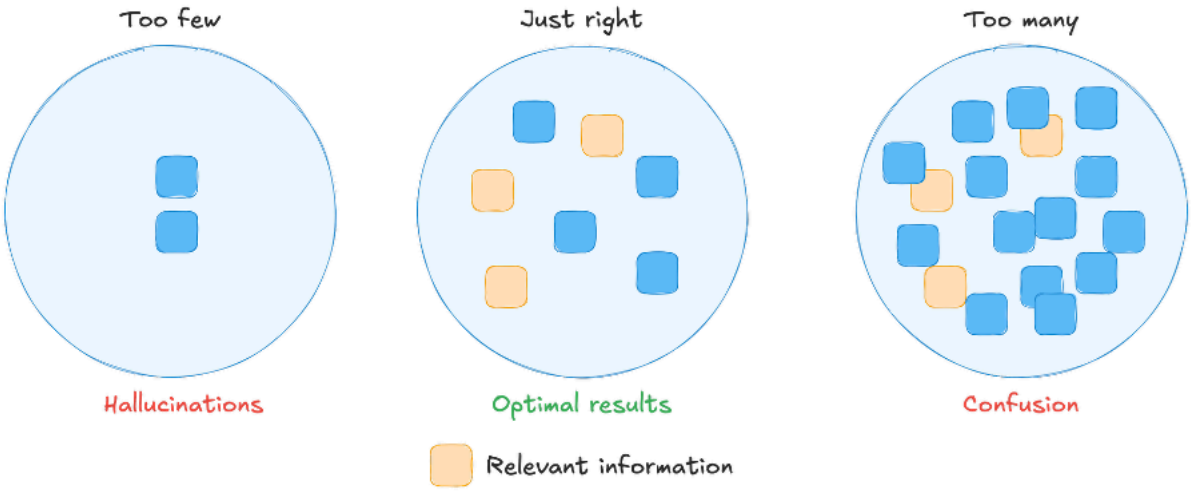
- 도메인별 워크플로 - 필요한 지식이 API 설명서에 캡처되지 않은 경우 도메인 전문 지식이 필요한 다단계 워크플로를 사용자 지정 MCP 도구에 캡슐화해야 합니다. 예를 들어 에이전트가 HIPAA(Health Insurance Portability and Accountability Act) 규정 준수를 검증하고 PII를 익명화하며 HL7 FHIR 형식으로 변환해야 하는 복잡한 의료 데이터 파이프라인을 오케스트레이션하도록 하는 대신 도메인 전문 지식을 직접 포함하는 `process_patient_data` 도구를 제공합니다. 이렇게 하면 LLM에 대한 종속성이 제거되어 워크플로 단계를 올바르게 오케스트레이션하고 실행할 수 있으므로 일관성과 규정 준수가 향상됩니다.
- 골든 경로 추상화 - 에이전트는 최적의 접근 방식을 구현하는 데 어려움을 겪을 수 있습니다. 에이전트는 조직 컨텍스트가 부족하고 조직 모범 사례가 아닌 기본 패턴을 기본으로 사용하기 때문입니다. 이러한 시나리오에서는 사용자 지정 MCP 도구에서 이러한 골든 경로를 캡슐화하여 비용, 성능 또는 보안에 대한 규범적 표준을 적용할 수 있습니다. 예를 들어 에이전트가 안전하지 않거나 비효율적일 수 있는 기본 설정으로 인프라를 배포하도록 하는 대신 조직의 표준을 직접 포함하는 도구를 제공합니다 `deploy_secure_infrastructure`.
- 복잡한 다중 서비스 오케스트레이션 - 에이전트가 각 단계에서 사용할 올바른 시퀀스와 서비스 세트를 추론하여 복잡한 워크플로를 오케스트레이션하도록 하는 대신 MCP 도구 내에서 해당 로직을 결정적으로 구축할 수 있습니다. 에이전트가 인식하지 못할 수 있는 최적의 서비스 통합 패턴에 대한 전문 지식을 제공할 수도 있습니다. 이렇게 하면 에이전트의 정확성과 효율성도 향상될 수 있습니다.
- 서비스별 모범 사례 - 이는 에이전트가 에이전트 도구를 통해 액세스하는 서비스와 관련된 암호화 정책, 액세스 제어 및 규정 준수 패턴을 구현하는 데 도움이 되는 보안 중심 도구에 일반적으로 적용됩

니다. 또한 명확하지 않은 서비스별 운영 모범 사례가 있는 경우 MCP 서버를 사용하면 MCP 서버를 구현하고 에이전트가 추론하지 않도록 하는 데 도움이 될 수 있습니다.

MCP 도구 설계 전략

MCP 클라이언트 및 서버의 주요 작업은 LLM을 사용하여 응답을 개선할 수 있도록 LLM에 도구를 검색하고 제공하는 것입니다. 따라서 MCP 도구 설계는 효과적인 MCP 솔루션을 구축하기 위한 가장 중요한 전략 중 하나입니다. 모델의 관점에서 도구는 더 정확하고 완전한 응답을 제공하기 위해 필요에 따라 호출할 수 있는 함수입니다. 함수 인터페이스는 단일 API 호출 주위의 래퍼부터 복잡한 워크플로 로직에 이르기까지 도구의 기본 구현을 추상화합니다.

그러나 LLM에 제공된 도구 수량과 균형을 맞춰야 합니다. 도구가 너무 적으면 LLM이 올바른 컨텍스트와 정보를 수집하지 못할 수 있으므로 모델 내에서 사용 가능한 정보로 가장 잘 추측할 수 있습니다. 도구가 너무 많으면 LLM이 올바른 도구 선택 및 시퀀스에 대해 혼동되어 할루시네이션이 발생할 수 있습니다. 목표는 적절한 수의 도구를 확보하는 것입니다. 다음 이미지는 너무 적거나 너무 많은 도구의 문제를 보여줍니다.



솔루션을 사용하려면 제공할 도구 수와 각 도구의 범위를 지정하는 방법을 이해해야 합니다. 개별 API 호출에 매핑하든 워크플로를 완료하든 상관없이 도구의 세분성은 에이전트가 필요로 하는 총 도구 수와 이를 얼마나 효과적으로 사용할 수 있는지에 직접적인 영향을 미칩니다. 이 섹션에서는 MCP 도구 범위 지정, 도구 정의 생성, 도구 검색 및 구성에 대한 모범 사례를 제공합니다.

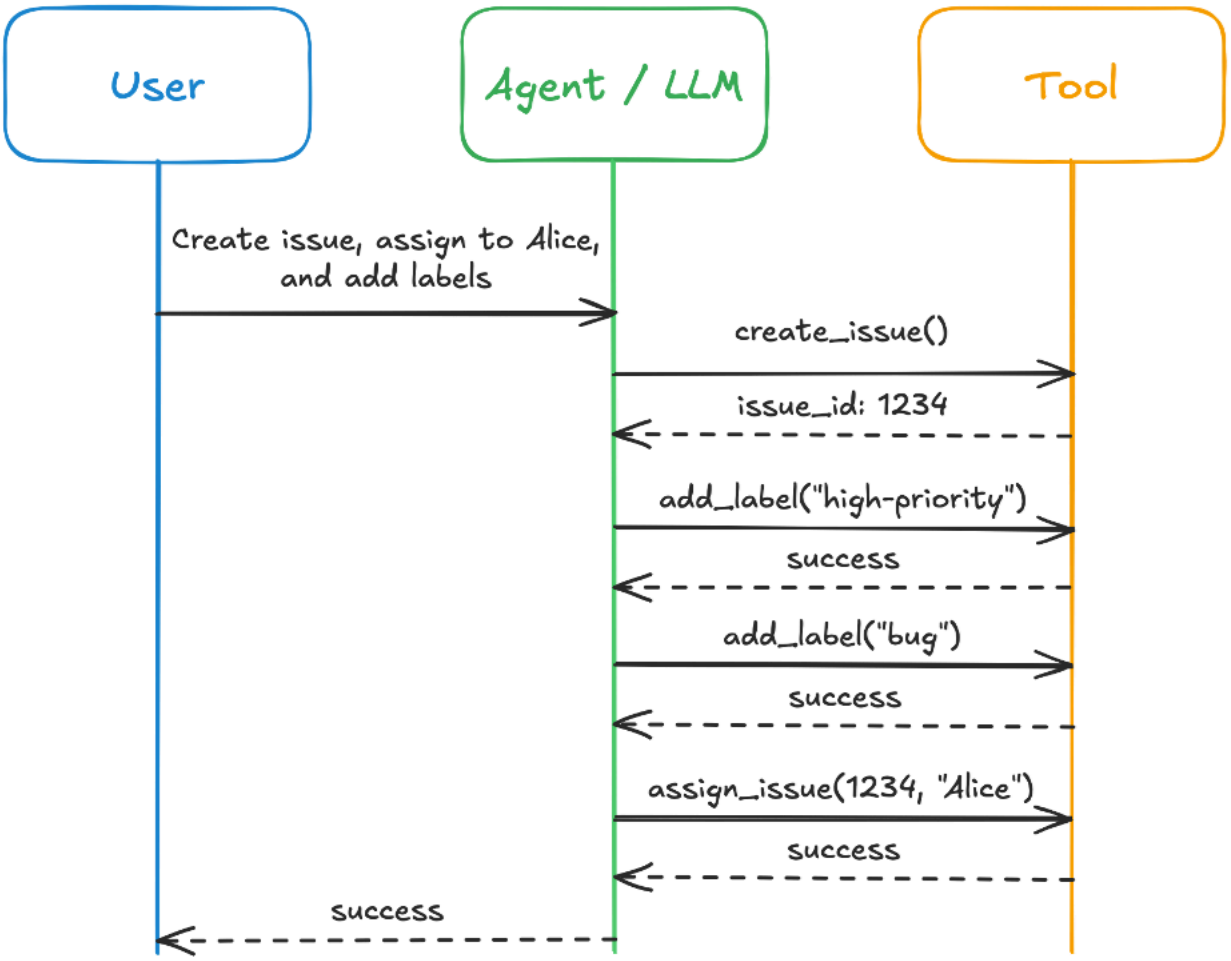
도구 범위

도구를 개발하는 방법에는 세분화된 방식과 대략적인 방식이라는 두 가지가 있습니다.

세분화

세분화된 접근 방식에서는 API, 작업 또는 쿼리별로 도구를 생성합니다. 예를 들어 Git 리포지토리에 대한 , `create_issue` `get_issueadd_label`, `assign_issue`, 및 `close_issue` 도구를 생성할

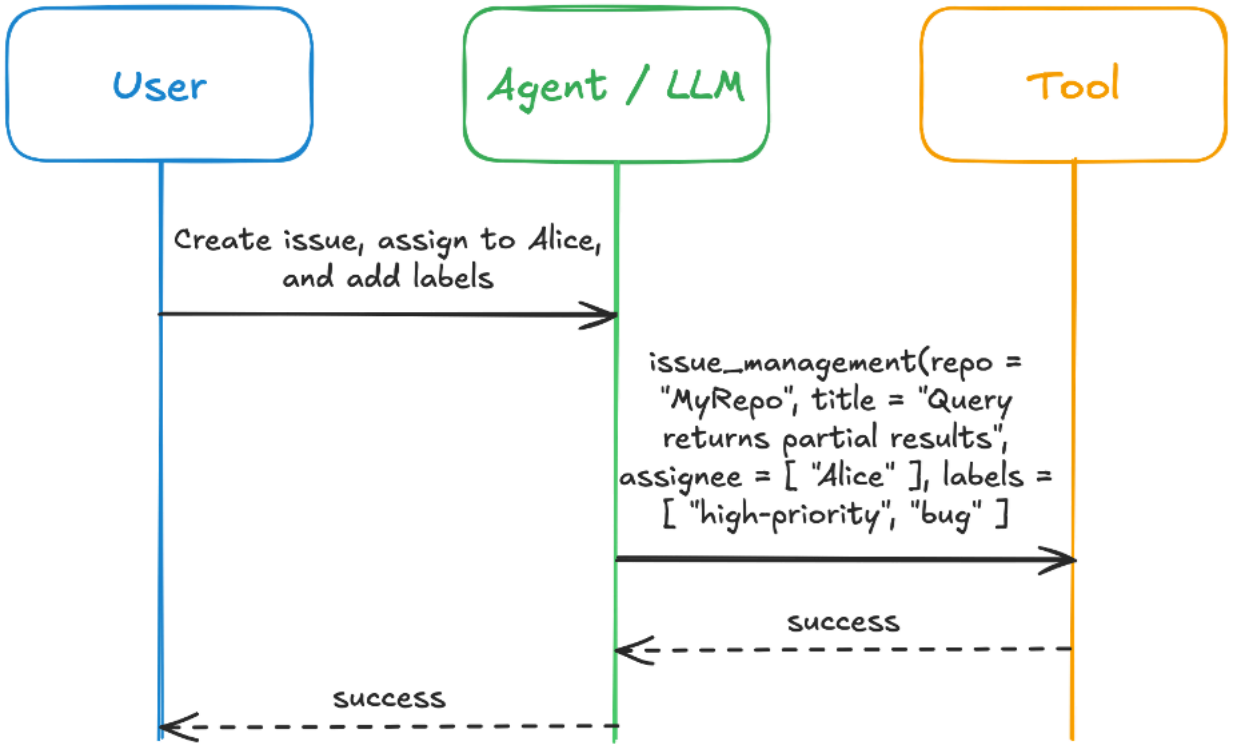
수 있습니다. 이렇게 하면 LLM이 각 API를 세부적으로 호출하고 필요에 따라 각 API를 오케스트레이션할 수 있습니다. '쿼리가 부분 결과만 반환합니다'라는 제품 서비스에 대한 문제를 생성하고, 버그 및 높은 우선 순위로 레이블을 지정하고, Alice에 할당합니다.'라는 프롬프트를 생각해 보세요. 다음 이미지는 tool-per-API 접근 방식이이 프롬프트에 어떻게 응답하는지 보여줍니다.



이 접근 방식에서는 시스템 프롬프트와 등록된 모든 도구 정의가 각 호출 시 LLM에 제공됩니다. 이렇게 하면 추가 컨텍스트가 사용되며 각 도구 호출이 LLM에 대한 개별 호출을 나타내기 때문에 지연 시간 페널티가 발생합니다. 또한 워크플로 내에서 오류를 처리하는 복잡성을 높입니다.

대략적

대략적이거나 워크플로 중심의 접근 방식은 워크플로 지향적인 도구입니다. 이 도구는 API 구조를 통한 end-to-end 사용자 의도에 중점을 둡니다. tool-per-API 대신 많은 APIs. 이전 Git 리포지토리 예제를 사용하여 에이전트가 한 번 호출하는 `create_and_setup_issue` 도구를 생성할 수 있습니다. 도구 구현은 문제를 생성하고, 레이블을 추가하고, 도구에 제공된 파라미터를 기반으로 사용자에게 할당합니다. 다음 이미지는 대략적인 접근 방식이 동일한 프롬프트를 처리하는 방법을 보여줍니다.



이 접근 방식은 LLM 계층에서 모든 복잡성을 숨기는 방법을 보여줍니다. 오케스트레이션 로직이 도구 구현에 포함되면 모든 순차적 단계, 로깅, 재시도 로직, 회로 차단기 및 속도 제한이 도구에서 결정론적으로 수행됩니다. 워크플로 기반 접근 방식을 사용하면 LLM이 올바른 파라미터를 사용하여 올바른 도구를 더 간단하게 호출할 수 있습니다. Amazon EC2 APIs와 같은 일부 `RunInstances` API는 이미 워크플로 의도를 제공할 수 있습니다. 이러한 경우 tool-per-API 원하는 워크플로 지향 설계를 제공할 수 있습니다.

그러나 도구도 너무 거칠어질 수 있습니다. 단일 워크플로 도구가 너무 많은 작업을 시도하고 가능한 파라미터가 많은 경우 LLM은 도구를 올바르게 사용하는 방법을 추론하는 데 어려움을 겪을 수 있습니다. 또한 파라미터 선택 및 오류 처리와 관련된 문제를 일으킬 수 있습니다. 따라서 도구 개발은 사용자 의도에 부합하고 단일 도구에서 기능이 너무 적거나 너무 많지 않도록 균형을 맞춰야 합니다. 전체 사용자 워크플로, 일반적으로 함께 발생하는 번들링 작업(예: 3개 이상의 API 직접 호출)을 중심으로 도구를 설계하는 것이 좋습니다. 또한 8개 이상의 파라미터를 초과하거나 여러 개의 고유한 사용자 의도를 처리하는 도구를 분해하는 것이 좋습니다. 실제 프롬프트로 테스트하여 에이전트가 각 도구를 올바르게 사용할 수 있는지 확인합니다.

결정적 도구로 쉽게 캡슐화할 수 없는 복잡하고 동적인 워크플로가 있는 경우 agent-as-tool 패턴을 사용하는 것이 좋습니다. 기본 에이전트가 워크플로에서 복잡한 작업을 오케스트레이션하려고 하는 대신 특수 에이전트가 도구 역할을 할 수 있습니다. 이러한 유형의 도구는 고급 의사 결정 및 분기를 구현할 수 있으며, 오류를 처리하고 결정적 코드에서 쉽게 관리할 수 없는 로직을 재시도할 수 있습니다. 이

는 Agent2Agent(A2A) 프로토콜과 유사하지만 구별됩니다. A2A 프로토콜은 상호 보완적이며 에이전트 프레임워크에서 에이전트 간의 상호 운용성과 협업을 제공합니다.

각 에이전트에 필요한 핵심 기능을 식별하기 위해 가장 일반적인 사용자 워크플로를 매핑하여 워크플로 분석을 시작하는 것이 좋습니다. 이렇게 하면 실행 가능한 최소 도구 세트가 설정됩니다. 대규모로 MCP 서버를 개발한 경험에 따라 다음 사례를 따르는 것이 좋습니다. 이러한 관행이 충돌하는 경우 사용자 의도와 워크플로의 우선순위를 지정합니다.

MCP 도구 크기 조정 모범 사례

- 사용자 스토리를 생각하고 일반적인 작업을 번들링 - 도구는 여러 작업을 오케스트레이션할 필요 없이 사용자 상호 작용을 완료하도록 직접 매핑해야 합니다. 워크플로에 일반적으로 3개 이상의 개별 호출이 필요한 경우 이를 단일 도구로 결합합니다. 이렇게 하면 LLM의 인지 부하가 줄어들고, 도구 호출 수가 최소화되며, 작업을 완료하는 데 필요한 컨텍스트 소비와 지연 시간이 줄어들고, 정확도와 지연 시간이 향상됩니다.
- 파라미터를 8개 이하로 제한 - 도구가 8개의 파라미터를 초과하는 경우 여러 도구로 분해합니다. LLMs 복잡성이 증가함에 따라 파라미터 선택에 어려움을 겪습니다.

> **Note**
>
> 번들링 작업에 8개 이상의 파라미터가 필요한 경우 워크플로를 간소화하는 것이 엄격한 파라미터 제한보다 더 중요하므로 파라미터 수보다 번들링을 우선시하세요.

- 별도의 읽기 및 쓰기 작업 - 데이터를 읽고 수정할 수 있는 다양한 도구를 제공합니다. 이러한 분리는 에이전트가 잠재적으로 파괴적인 작업을 수행할 때 이를 명시적으로 만들고, 다양한 권한 부여 정책을 활성화하며, 정보 수집 중에 의도하지 않은 수정의 위험을 줄입니다.
- 합리적인 기본값 제공 - LLM이 개별 요청과 관련된 파라미터만 지정하도록 도구를 설계합니다. 기본값은 LLM이 고려해야 하는 정보를 최소화하여 파라미터 복잡성을 줄이고 도구 선택 정확도를 개선합니다.
- 결정론적 실행 선호 - 가능하면 도구 실행 및 출력을 결정론적으로 만듭니다. 결정적 도구는 더 안정적이고 테스트하기 쉽습니다. 결정적 코드에서 쉽게 관리할 수 없는 지능형 오케스트레이션, 분기 로직 또는 고급 오류 처리가 필요한 복잡한 워크플로의 경우 특수 에이전트를 도구로 사용하는 것이 좋습니다. 그러나이 패턴은 복잡성을 추가하므로 선택적으로 사용합니다.

도구 정의

LLM은 직접 처리할 수 없는 요청을 수신하면 사용 가능한 도구를 검토하여 요청을 완료하는 데 도움이 됩니다. LLM은 제공된 도구의 이름 및 설명과 프롬프트에 제공된 지침에 대한 의미론적 이해를 기반으로 도구를 선택합니다. 그런 다음 정의된 입력 스키마를 기반으로 입력을 생성하고 출력 스키마를 기반으로 출력을 예상합니다. 따라서 LLM이 도구를 효과적으로 선택하는 데 도움이 되려면 설명이 포함된 도구 정의와 검증된 입력 및 출력 스키마를 만드는 것이 중요합니다. 이 설명서를 생성하는 데는 일반적으로 도구 사양 접근 방식과 문서 문자열 접근 방식이라는 두 가지 접근 방식이 있습니다.

도구 사양 접근 방식

권장되는 접근 방식은 도구를 정의할 때 MCP 도구 사양을 직접 따르는 것입니다. 다음 예제는 Strands Agent 도구 데코레이터를 사용하여 표시됩니다.

```
@tool(
  name = "search_website",
  description = "This tool searches the provided website for semantic matches to the
 query provided",
  inputSchema = {
    "json": {
      "type": "object",
      "properties": {
        "url": {
          "type": "string",
          "description": "The url of the website to load and search."
        },
        "query": {
          "type": "string",
          "description": "The content you want to try and match in the website."
      }
    },
    "required": ["url", "query"]
  },
  outputSchema = {
    "json": {
      "type": "object",
      "properties": {
      "results": {
          "type": "array",
          "items": {
            "type": "string"
          }
```

```
        }
      }
    }
  }
)
def search_website:
  …
```

, name, inputSchema, description등의 표준 필드를 사용하면 모든 도구에 LLM과 사람이 모두 이해할 수 있는 일관된 설명서가 있는지 outputSchema 확인할 수 있습니다. 모든 도구는 최소한 이러한 필드를 정의하고 선택적으로 도구 동작에 대한 선택적 힌트인 제목과 주석을 제공해야 합니다. 가능하면 파라미터 값에 열거형을 사용하여 LLM이 올바른 옵션을 쉽게 선택할 수 있도록 합니다. 열거형은 상태 또는 우선 순위 값과 같은 유한 집합에 가장 적합하지만 자유 형식 텍스트, 동적 값, 임의 숫자 또는 리소스 식별자에는 적합하지 않습니다. 이러한 경우 대신 명확한 설명과 예제를 제공합니다. LLM이 올바른 옵션이 무엇인지 추측할 필요가 없도록 가능하면 기본값도 포함합니다. 도구 정의는 각 호출의 LLM 프롬프트에 포함되어 시스템 지침 및 대화 기록과 함께 컨텍스트 창 공간을 사용한다는 점에 유의하세요.

Docstring 접근 방식

또 다른 접근 방식인 Python으로 도구를 작성하는 경우는 문서 문자열을 사용하여 도구의 설명, 사용 및 출력을 제공하는 것입니다. 다음은이 접근 방식의 예입니다.

```
def search_website(url: str, query: str) -> list:

    """
    This tool loads the specified website and then attempts to find content that
 matches the provided query through semantic search. It provides back a list of strings
 that are the sentences that match the query.
    Args:
      url: the website url to load
      query: the content you want to semantically match in the website
    """
```

Docstring은 스키마 또는 표준화된 형식을 적용하지 않습니다. 이 접근 방식을 사용하면 도구 개발자가 각 도구를 문서화하도록 선택하는 방법에 따라 일관되지 않은 결과를 얻을 수 있습니다. 이 접근 방식을 따르는 경우 조직 전체의 표준을 정의하고 적용하는 것이 중요합니다.

MCP 도구 정의 모범 사례

- MCP 도구 사양 준수 - 모든 도구에 대해 `name`, `description`, `inputSchema`, 및 `outputSchema` 필드를 제공합니다. Python 구현의 경우 Pydantic 모델을 사용하여 필드 설명, 자동 유형 검증 및 열거형을 통한 제한된 값을 통해 인라인 설명서를 제공합니다. 이렇게 하면 스키마가 자체 문서화되고 유효한 파라미터 옵션에 대한 LLM 이해가 향상됩니다.
- 프롬프트로 설명 작성 - 도구 설명은 LLM 의사 결정을 안내하는 지침입니다. 도구 용도의 필수 구성 요소(도구가 수행하는 작업), 도구 사용 시기(사용자 의도 패턴 또는 시나리오), 출력 컨텍스트(출력이 사용되는 용도), 파라미터 및 오류 조건을 포함합니다.
- 구체적인 예제 제공 - 실제 값과 함께 워크플로 예제를 포함하는 것이 올바른 도구 사용에 대해 LLMs을 안내하는 가장 효과적인 방법입니다.
- 문서 종속성 명시적 - 사전 조건, 번호가 매겨진 시퀀스, 상태 변경 및 후속 작업을 포함합니다.

도구 검색

에이전트에서 도구를 검색하고 MCP 서버에 등록하는 방법에는 정적 정의, 동적 검색 및 검색 함수라는 세 가지가 있습니다.

정적 정의

먼저 에이전트 코드에서 사용 가능한 도구를 직접 정적으로 정의할 수 있습니다. 이 접근 방식에서는 MCP 클라이언트가 액세스하는 MCP 서버에서 제공하는 각 도구에 대해 원격 도구(Strands Agent SDK와 같은 프레임워크의 클라이언트 측 참조 객체)를 정의합니다. 다음 예제에서는 스트리밍 가능한 HTTP 전송을 사용합니다.

```
from mcp.client.streamable_http import streamablehttp_client
from strands import Agent
from strands.tools.mcp import MCPClient

streamable_http_mcp_client = MCPClient(
    lambda: streamablehttp_client("https://mcp1:8000/mcp")
)

reverse_text = RemoteTool(
    name="reverseText",
    client=streamable_http_mcp_client
)
```

```
agent = Agent(tools=[reverse_text])
```

개별 도구 등록을 사용하면 LLM에서 사용할 수 있는 도구를 매우 선택적으로 선택할 수 있으므로 사용되는 컨텍스트 기간이 최소화됩니다. 단점은 사용 가능한 도구의 이름을 알아야 하며 MCP 서버에서 사용 가능한 도구가 변경될 경우 부서질 수 있다는 것입니다.

동적 검색

다음 접근 방식은 동적 검색을 사용하고 에이전트에 사용 가능한 모든 도구를 등록하는 것입니다. 이 접근 방식은 MCP 서버에 더 많은 도구가 추가될 때 컨텍스트를 선형으로 사용합니다. 다음은이 접근 방식의 예입니다.

```
from mcp.client.streamable_http import streamablehttp_client
from strands import Agent
from strands.tools.mcp import MCPClient

streamable_http_mcp_client = MCPClient(
    lambda: streamablehttp_client("https://mcp1:8000/mcp")
)

with streamable_http_mcp_client:
    tools = streamable_http_mcp_client.list_tools_sync()
    agent = Agent(tools=tools)
```

일반적인 도구 정의에서 약 250~500개의 토큰(이름, 설명 및 스키마 포함)을 사용하는 시나리오를 생각해 보세요. 20개의 도구를 등록하면 컨텍스트 창에 5,000~10,000개의 토큰이 사용됩니다. MCP 서버 수가 적고 도구 수를 제어할 수 있는 경우이 옵션을 구현하는 것이 가장 간단합니다. 그러나 도구 목록이 증가할 것으로 예상되는 경우 에이전트에서 자동 컨텍스트 관리 문제가 발생할 수 있습니다. 이 접근 방식의 다른 변형은 Strands Agents SDK가 제공하는 것과 `list_tools`같이를 호출할 때 도구 필터 파라미터를 사용하여 에이전트에 등록된 도구 수를 줄이는 것입니다.

검색 함수

세 번째 옵션은 검색 함수를 사용하여 런타임 중에 관련 도구를 찾는 것입니다. MCP 서버에서 사용 가능한 모든 도구를 나열한 다음 사용자 프롬프트를 기반으로 해당 도구에 대한 의미 체계 검색을 수행합니다. 그러면 결과 도구가 에이전트에 등록됩니다. Amazon Bedrock AgentCore Gateway는 이러한 종류의 솔루션을 더 쉽게 구현할 수 있는 네이티브 의미 체계 검색 기능을 제공합니다.

MCP 도구 검색 모범 사례

- 컨텍스트 기간 보존 - 컨텍스트 기간을 최대한 많이 절약하는 도구 검색 및 등록 접근 방식을 선택합니다.
- 도구 필터링 또는 의미 체계 검색 기능 사용 - LLM에 선택할 수 있는 범위가 축소된 도구 세트를 동적으로 제공하여 올바른 도구를 선택할 때 정확성과 효율성을 개선합니다. 도구 필터링은 도구 이름(정확한 일치 또는 패턴), 도구 설명(시맨틱 일치) 또는 도메인 또는 범주 태그에서 작동할 수 있습니다. 의미 체계 검색은 사용자 의도를 도구 설명과 일치시키는 데 특히 효과적입니다. 두 접근 방식 모두 컨텍스트 창 사용을 줄입니다.

도구 구성

올바른 도구를 찾고 LLM이 이를 효과적으로 사용할 수 있도록 하는 것은 효과적인 도구 개발의 가장 중요한 부분 중 하나입니다. MCP 서버 개발을 시작할 때 다음을 결정하는 전략이 필요합니다.

- 하나의 MCP 서버로 이동하는 도구 수
- 동일한 MCP 서버에 넣으면 안 되는 도구
- 도구 이름을 지정하여 검색 가능하게 만들고 이름 충돌을 방지하는 방법(이름이 같은 다른 도구)
- LLM에서 쉽게 사용할 수 있도록 도구 및 MCP 서버를 문서화하는 방법

네임스페이스 조직은 도구 이름 충돌을 방지하고, 관련 기능을 그룹화하고, LLMs. 이 패턴은 비정형 누적이 아닌 정형 스토리지 시스템과 유사한 정형 분류를 설정합니다. 도구 이름 지정에는 domain-noun-verb 패턴을 사용하는 것이 좋습니다. 예: `github_issue_create`, `github_issue_list`, `github_issue_update`, `github_pullrequest_create`, `github_pullrequest_list`또는 `github_pullrequest_merge`. 이 패턴의 장점은 알파벳 정렬 동작을 검사할 때 분명합니다. 도구가 알파벳순으로 나열되면 모든 문제 관련 작업이 함께 클러스터링되고(`create`, `list`, `update`) 그 뒤에 풀 요청 작업(`create`, `list`,)이 옵니다`merge`. 명사(리소스 유형)는 조직 경계 역할을 합니다. 이 구조는 관련 기능이 자연스럽게 함께 그룹화되므로 LLM 도구 스캔과 인적 문서 탐색을 모두 용이하게 합니다.

MCP 서버는 도메인 수준에서 경계가 지정되어야 하지만 제공하는 기능에 대한 업무 분리에 따라 세분화될 수 있습니다. 예를 들어 데이터베이스에 쓰기 작업과 읽기 작업을 위한 별도의 MCP 서버가 있을 수 있습니다. 이러한 분리를 적용하려면 사용자 의도 및 권한에 따라 액세스할 수 있는 MCP 서버를 제한하는 가드레일을 에이전트 수준에서 구현하는 것이 좋습니다. 이는 다음과 같은 조합을 통해 달성할 수 있습니다.

- 조건부 서버 로드 - 에이전트가 사용자 입력에서 읽기 작업을 감지한 경우에만 읽기 전용 MCP 서버를 로드합니다.
- 권한 기반 필터링 - 사용자 권한을 사용하여 적절한 MCP 서버에만 액세스 권한을 부여합니다.

마지막으로 MCP 서버에서 제공하는 도구 수에 대한 상한을 생성해야 합니다. 에이전트가 MCP 서버를 사용하는 방법을 가정하지 마세요. 사용 가능한 모든 도구를 순진하게 나열하고 LLM에 모두 제공할 수 있습니다. 단일 서버에 50개 이상의 도구가 있는 경우 여러 서버로 분할하는 것을 고려해야 합니다.

MPC 도구 조직에 대한 모범 사례

- 도구에 domain-noun-verb 이름 지정 표준 사용 - MCP 서버와 에이전트 모두에서 이름 충돌을 방지하는 전략을 구현합니다.
- 상한 설정 - 단일 MCP 서버의 도구 수를 제한합니다.
- MCP 서버 분할 - 업무 분리를 사용하여 MCP 서버를 논리적 그룹으로 분할합니다.

MCP 호스팅 전략

사용 가능한 도구를 MCP 서버로 추출하면 에이전트 개발이 사용 가능한 도구와 분리됩니다. 이렇게 하면 MCP 서버를 호스팅하는 위치와 해당 서버 내에서 도구가 구성되는 방식에 대한 문제가 발생합니다.

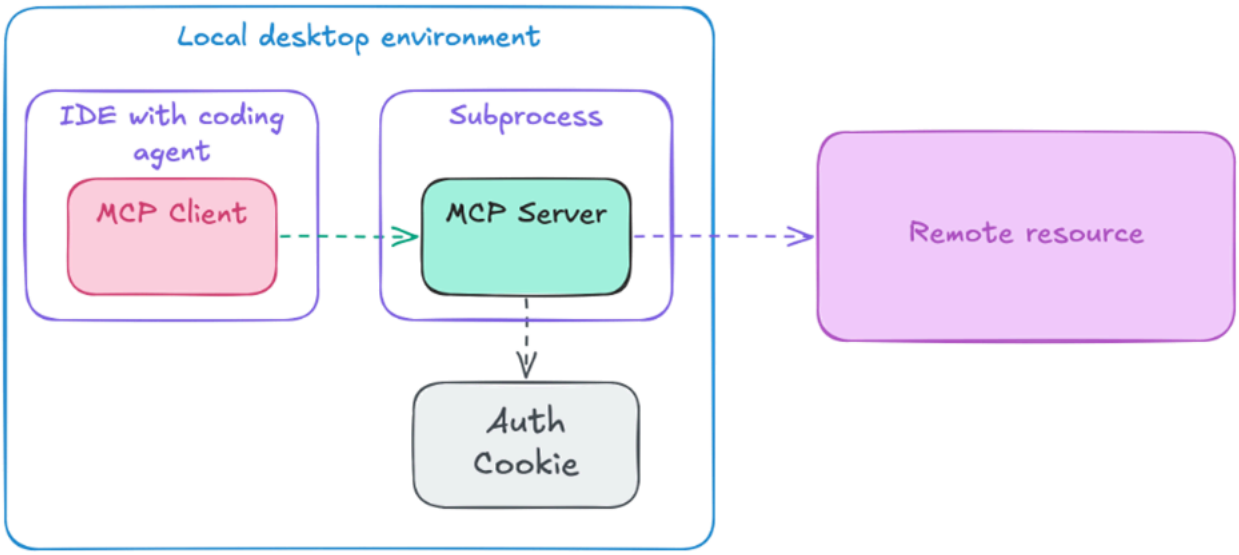
호스팅 접근 방식

MCP 서버를 호스팅하는 옵션은 최종 사용자 시스템에서 로컬로 실행하거나, 원격으로 호스팅하거나, MCP 게이트웨이를 통해 호스팅하는 세 가지입니다. 각 옵션에는 장점과 장단점이 있습니다.

로컬 호스팅

로컬 호스팅은 표준 입력 및 출력 스트림을 통해 JSON-RPC를 사용하여 서버와 통신하는 에이전트와 함께 로컬 시스템에서 MCP 서버를 하위 프로세스로 실행합니다. 이 접근 방식은 클라이언트와 서버 간의 인증이 필요하지 않습니다. 도구는 로컬 애플리케이션 및 파일과 상호 작용하고, 로컬에 저장된 자격 증명을 사용하고, 사용자 로컬 시스템의 네트워크 액세스를 상속할 수 있습니다. 이는 가장 간단한 호스팅 패턴이며 몇 가지 이점이 있습니다.

많은 고객이 로컬 서버를 사용하여 MCP를 시작합니다. 이를 통해 엔지니어는 로컬 환경에서 다양한 문제를 신속하게 반복하고 해결할 수 있습니다. 엔지니어의 코딩이 사용하는 Git 리포지토리에 연결하는 MCP 서버를 생각해 보세요. MCP 서버를 로컬로 유지하는 것은 엔지니어의 고유한 자격 증명을 사용하여 리포지토리에 액세스할 수 있고 원격 MCP 서버에 네트워크 호출을 추가하지 않기 때문에 매우 유용합니다. 다음 이미지는 IDE에서 코딩 에이전트와 함께 사용되는 로컬 호스팅 MCP 서버를 보여줍니다.



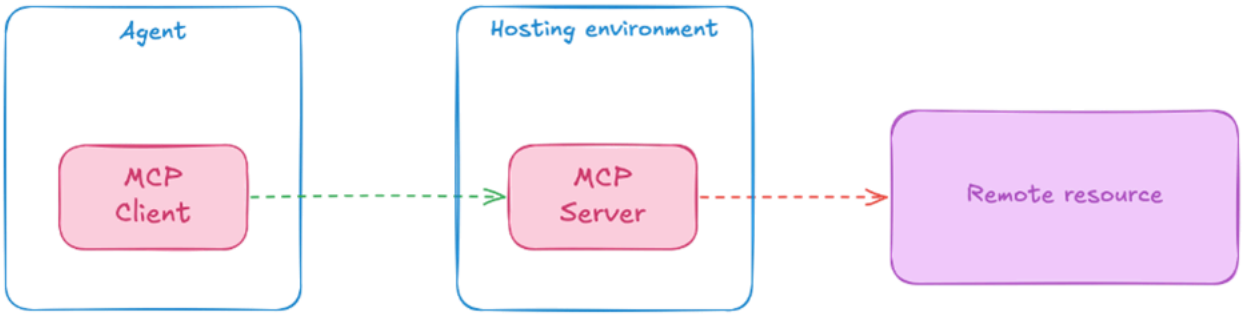
이러한 유형의 배포의 경우 MCP 서버를 개발하고 배포하는 방법을 고려해야 합니다. 대부분의 고객은 최종 사용자가 서버를 등록하고 다운로드할 수 있는 MCP 레지스트리를 개발합니다. 이는 사용자가 특정 기능을 검색하고 필요에 적합한 MCP 서버를 찾을 수 있는 컨테이너 레지스트리와 매우 유사합니다.

공식 MCP 레지스트리와 같은 퍼블릭 MCP 레지스트리가 있으며 비공개 호스팅 레지스트리가 있습니다. 조직은 일반적으로 오픈 소스 소프트웨어 배포, 컨테이너 레지스트리 및 내부 패키지 관리에 대한 기존 정책에 MCP 레지스트리 전략을 맞춥니다. 보안 스캔, 승인 워크플로 및 규정 준수 요구 사항과 같은 요소를 고려해야 합니다.

그러나 로컬 호스팅은 조직이 고려해야 할 운영 문제를 야기합니다. 먼저 최종 사용자는 MCP 서버를 독립적으로 검색, 다운로드 및 구성해야 합니다. 이렇게 하면 로컬에서 사용하는 각 개별 MCP 서버를 시작하는 데 복잡성이 가중될 수 있습니다. 둘째, MCP 서버 수명 주기를 제어할 수 없습니다. 즉, 사용자는 보안 취약성 또는 누락된 기능이 있는 오래된 버전을 로컬에서 계속 실행할 수 있습니다. 이로 인해 규정 준수 요구 사항이 복잡해질 수 있습니다. Kiro와 같은 일부 IDEs 및 CLI 도구를 사용하면 조직이 사용 가능한 MCP 도구를 관리하고 제어하여 팀 간의 일관성과 보안을 보장할 수 있습니다.

원격 호스팅

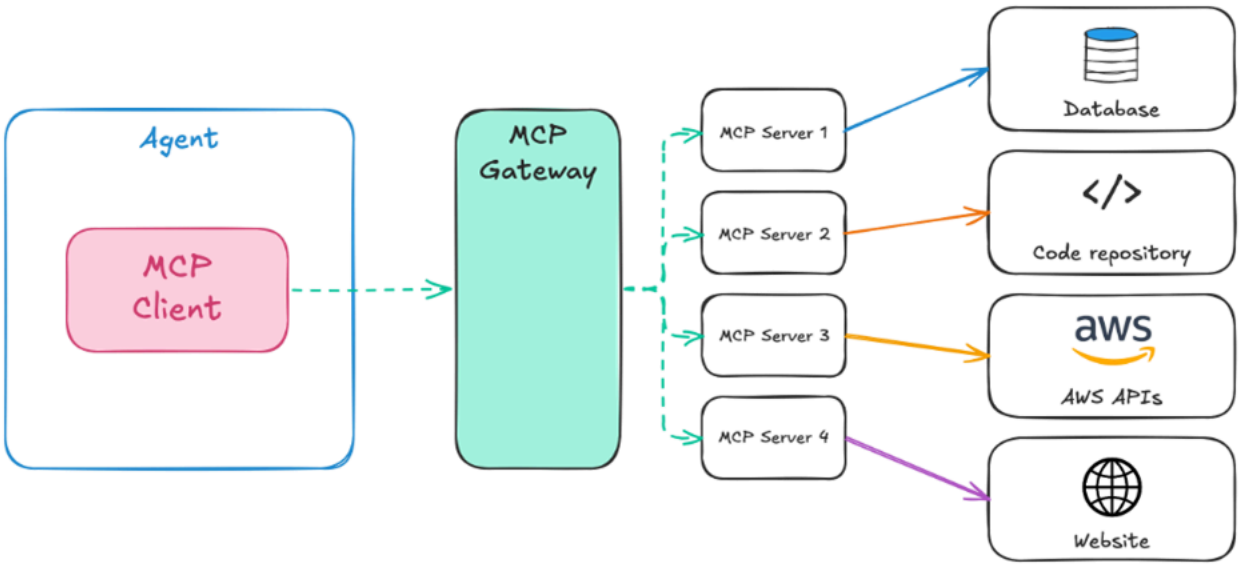
두 번째 옵션은 HTTP 또는 HTTPS를 통해 액세스하는 원격 MCP 서버를 호스팅하는 것입니다. 이렇게 하면 모든 네트워크 연결 클라이언트에 액세스할 수 있습니다. 원격 호스팅을 사용하면 MCP 리소스 및 기능에 대한 액세스를 중앙에서 제어하고, 인증 및 권한 부여를 구현하고, MCP 서버 로직의 버전 관리 및 업데이트를 제어할 수 있습니다. 원격 호스팅을 사용하려면 최종 사용자가 에이전트와 함께 사용하려는 MCP 서버를 검색할 수 있도록 MCP 레지스트리를 사용해야 합니다. 다음 이미지는 원격 호스팅 접근 방식을 보여줍니다.



에이전트 개발 관점에서 볼 때 MCP 서버가 로컬이든 원격이든 환경은 비슷합니다. 가장 중요한 변경 사항은 MCP 서버에 대한 에이전트의 액세스와 외부 리소스에 대한 서버의 액세스를 모두 포함하여 인증 및 권한 부여를 구현하는 것입니다. 다중 테넌트 액세스 및 권한 관리를 고려하려면 원격 MCP 서버 구현을 신중하게 계획해야 합니다. MCP 거버넌스 전략 장에는 인증 및 권한 부여 고려 사항에 대한 자세한 정보가 포함되어 있습니다.

MCP 게이트웨이

마지막 옵션은 MCP 게이트웨이를 사용하는 것입니다. MCP 게이트웨이는 MCP 클라이언트와 서버 간의 중앙 집중식 프록시 역할을 하며 등록된 MCP 서버에 대한 액세스를 오케스트레이션합니다. 게이트웨이가 없으면 각 에이전트는 사용하려는 모든 원격 MCP 서버를 등록해야 합니다. 게이트웨이를 사용하면 에이전트가 인증, 권한 부여, 라우팅 및 프로토콜 번역을 관리하는 단일 엔드포인트에 연결할 수 있습니다. 새로운 MCP 서버와 도구를 동적으로 추가하고 에이전트가 즉시 사용할 수 있습니다. 다음 이미지는 MCP 게이트웨이 접근 방식을 보여줍니다.



Docker MCP Gateway와 같은 일부 게이트웨이 솔루션은 MCP 서버의 수명 주기도 관리하여 필요에 따라 온디맨드로 서버를 시작합니다. Amazon Bedrock AgentCore Gateway와 같은 MCP 게이트웨이는 네이티브 의미 체계 검색 기능을 제공하여 도구 검색을 관리하는 데도 도움이 될 수 있습니다. 이렇게 하면 에이전트가 MCP 클라이언트와 연결할 수 있는 단일 엔드포인트를 제공하고 컨텍스트 창 사용을 최적화하는 데 도움이 됩니다. 그 결과 MCP 도구를 효과적으로 선택하고 사용할 수 있는 간단한 에이전트가 생성됩니다. 그러나 원격 MCP 서버 접근 방식과 유사한 자격 증명 관련 문제가 있습니다.

MCP 서버 호스팅 모범 사례

- 호스팅 옵션의 스펙트럼은 모든 것에 맞는 한 가지 크기가 아닙니다. 오늘날 MCP 서버 사용의 대부분은 로컬입니다.
- 원격 MCP 서버를 사용하기 시작할 때 주요 고려 사항은 MCP 서버에 대한 일관된 인증 및 권한 부여와 MCP 서버가 다운스트림 리소스에 대한 인증 및 권한 부여를 수행하는 방식입니다.
- MCP 게이트웨이는 여러 원격 MCP 서버를 호스팅하기 위한 연결 및 인증과 권한 부여를 간소화합니다. 또한 적용 가능한 도구를 검색하여 컨텍스트 기간 관리를 개선하는 기능도 제공합니다.

MCP 거버넌스 전략

MCP가 조직에 제공하는 또 다른 중요한 기능은 중앙 집중식 거버넌스를 지원하는 것입니다. MCP 거버넌스 전략은 MCP 서버와 MCP 서버가 액세스하는 리소스 모두에 대한 인증 및 권한 부여를 해결해야 합니다. 또한 다운스트림 리소스를 보호하기 위한 속도 제한, 도구 사용 및 성능을 모니터링하기 위한 운영 지표, MCP 서버의 배포 및 배포 관리를 해결해야 합니다.

인증 및 권한 부여

인증 및 권한 부여 전략에서 가장 중요한 부분 중 하나는 MCP 서버에서 다운스트림 리소스 액세스를 관리하는 것입니다. 사용자가 에이전트를 호출하면 인증 및 권한 부여가 수행되어 사용자에게 에이전트를 호출할 권한이 있는지 확인합니다. 그런 다음 에이전트는 MCP 서버에서 특정 도구를 호출하도록 오케스트레이션합니다. 도구별로 액세스를 승인하는 방법을 결정해야 합니다.

한 가지 옵션은 사용자 동의 또는 상호 작용이 필요하지 않은 machine-to-machine 권한 부여입니다. 예를 들어 시간 기반 에이전트 호출은 MCP 서버를 사용하여 애플리케이션에서 로그를 수집하고 분석합니다. 이 시나리오에서 에이전트는 지정된 데이터에 액세스할 수 있는 사전 권한이 있습니다. 두 번째 옵션은 사용자가 사용자별 데이터 및 리소스에 액세스하는 데 동의하는 사용자 위임 액세스입니다.

다음 표에는 인증 및 권한 부여 패턴이 나와 있습니다.

팩터	사용자 위임 액세스	Machine-to-machine
데이터 소유권	데이터에 대한 사용자별 권한 부여	시스템 또는 조직 전체의 데이터
사용자 상호 작용	사용자가 있고 동의할 수 있음	사용자 상호 작용 없음
작업 타이밍	대화형 또는 실시간	배경, 예약 또는 배치
권한 범위	권한은 사용자마다 다릅니다.	에이전트 수준에서 일관된 권한

사용자 위임 액세스에는 신중한 구현이 필요하며 보안 팀과 함께 개발해야 합니다. 에이전트는 LLM이 선택한 도구와 추가 권한이 필요한지 여부를 평가할 수 있어야 합니다. MCP 도구에는 인증 및 권한 부여 요구 사항과 액세스 토큰을 검색할 위치를 나타내는 설명이 포함되어야 합니다. 클라이언트 애플리

케이션은 중간 인증 요청을 지원해야 하며, MCP 클라이언트는 각 도구 호출에 대해 검색된 자격 증명을 에이전트에 다시 제공해야 합니다.

MCP 도구에 항상 외부 기능에 액세스하기 위한 자체 토큰이 있고 액세스가 로깅 및 감사되는지 확인해야 합니다. 사용자 자격 증명은 에이전트 시스템을 통해 전파되어서는 안 됩니다. 예를 들어 MCP 서버는 에이전트를 호출하는 데 사용된 데이터에 액세스하는 데 동일한 토큰을 사용해서는 안 됩니다. 다운스트림 호출은 명시적으로 범위가 지정되고 용도에 따라 생성된 토큰을 사용해야 합니다. 이렇게 하면 작업을 대신하여 의도하지 않은 데이터 액세스를 방지하는 추가 가드레일을 제공할 수 있습니다. 또한 할루시네이션이 의도하지 않은 결과를 생성하는 것을 방지하는 데 도움이 될 수 있습니다. 전체 관리자 권한이 있는 사용자가 에이전트에게 사전 프로덕션에 사용할 프로덕션 데이터베이스를 복제하도록 요청한다고 가정해 보겠습니다. 이렇게 하려면 사용자에게 READ 및 CREATE 권한만 필요합니다. LLM이이 요청의 일부로 이전 데이터베이스를 정리해야 한다고 생각한다고 가정해 보겠습니다. 사용자의 자격 증명을 재사용하면 사용자의 원래 자격 증명에 DELETE 권한이 있기 때문에 성공할 수 있습니다. 대신 MCP 서버가 READ 및 CREATE 권한만 있는 요청에 의도적으로 범위가 지정된 토큰을 사용하는 경우 프로덕션 데이터베이스를 삭제하려는 시도가 실패합니다.

Amazon Bedrock AgentCore 자격 증명을 사용하여 이러한 패턴을 구현할 수 있습니다. MCP 서버에서 호스팅하는 도구를 나열하고 호출할 수 있는 권한이 MCP 서버가 노출하는 외부 기능에 대한 권한을 암시하는지 여부를 의도적으로 선택해야 합니다. 이 자격 증명 흐름은 MCP 서버에서 리소스로, 그리고 다시 사용자에게 사용 중인 인증 및 권한 부여 서비스의 유형에 따라 달라집니다. MCP 서버의 대규모 처리 방법을 결정해야 합니다.

인증 및 권한 부여 패턴을 설계할 때 액세스 중인 각 도구에 대해 서로 다른 액세스 토큰을 검색하는 토큰 격리 메커니즘을 구현합니다. 도구와 서버 간에 토큰을 재사용하지 마십시오. AgentCore 자격 증명은이 토큰 격리 기능을 제공합니다. 워크로드 토큰(machine-to-machine 인증용)과 사용자 토큰(사용자 위임 액세스용)을 모두 자동으로 관리하여 적절한 분리를 보장하고 권한 에스컬레이션을 방지합니다. 이는 원격 MCP 서버 또는 MCP 게이트웨이를 통합할 때 특히 중요합니다.

MCP 인증 및 권한 부여 모범 사례

- 토큰 분리 - 호출자의 보유자 토큰을 다운스트림 서비스에 전달하지 마십시오. aud(대상) 필드가 토큰을 수신하는 서버와 일치하는지 확인합니다. 대상 클레임은 토큰이 의도한 서비스를 지정하여 여러 MCP 서버에서 무단 토큰 재사용을 방지합니다.
- 액세스 접근 방식 선택 - MCP 서버가 제공하는 각 도구에 대해 machine-to-machine 액세스와 사용자 위임 액세스 중에서 선택합니다. 동일한 인증 패턴을 사용하는 동일한 MCP 서버에서 도구를 그룹화하는 것이 좋습니다.

로드 제어

분산 시스템과 마찬가지로 MCP 서버 플릿의 부하를 제어하는 방법을 고려해야 합니다. 먼저 MCP 서버에서 속도 제한을 구현할지 여부와 제한을 구현할 위치를 고려합니다. 속도 제한을 구현하지 않기로 선택한 경우 다운스트림 리소스에서 수행하는 속도 제한을 전달합니다. 많은 시스템은 사용자 또는 계정 ID와 같은 요청 속성을 기반으로 제한을 평가하도록 선택합니다. 다운스트림 서비스로 전송된 요청이 동일한 속성을 포함하는지 확인하여 여러 사용자가 다른 사용자의 부하로 인해 영향을 받지 않도록 합니다.

속도 제한을 구현하기로 선택한 경우 권장되는 접근 방식은 MCP 서버 수준에서 기본 속도 제한을 구현하는 것입니다. 백엔드 서비스는 보조 보호를 제공하고 에이전트는 속도 제한 피드백을 기반으로 동작을 조정합니다. 속도 제한이 MCP 서버당인지 도구당인지 고려합니다. MCP당 서버 속도 제한은 다중 테넌트 환경에서 MCP 서버 플릿 및 서비스를 보호하는 데 도움이 됩니다. 그러나 이는 매우 거칠 수 있습니다. 도구당 속도 제한은 충분히 속도를 제한하지 않을 수 있는 압도적인 다운스트림 리소스를 방지하도록 설계되었습니다. 도구가 여러 APIs를 호출하는 경우 해당 APIs에서 허용하는 최저 속도에 맞게 속도 제한을 설정해야 합니다.

HTTP 헤더의 전달 속도 제한 정보는 사용자와 자동화된 시스템이 자체 요청 속도 및 재시도 전략을 관리하는 데 도움이 되는 유용한 지표일 수도 있습니다. 예를 들어 다음 예제와 같이 이러한 헤더를 MCP 서버에서 에이전트로 다시 보낼 수 있습니다.

```
X-RateLimit-Limit: 100
X-RateLimit-Remaining: 45
X-RateLimit-Reset: 1640995200
```

또한 단일 고객이 속도 제한을 초과하지 않지만 로드가 시스템 성능에 영향을 미치는 경우 전체 서비스를 보호하기 위해 로드 셰딩을 고려합니다.

로드 제어 모범 사례

- 속도 제한 접근 방식 선택 - 다운스트림 리소스 사용 또는 MCP 서버 및 도구 사용을 기반으로 개별 사용자의 속도를 제한하도록 계획합니다.
- 로드 셰딩 고려 - 단일 또는 소수의 고객이 구동하지 않는 일반적인 오버로드로부터 MCP 서버 플릿을 보호합니다.

운영 지표

MCP 구현을 위해 캡처할 주요 지표는 고객이 제공하는 고객 경험에 초점을 맞춰야 합니다. 이러한 지표에는 일반적으로 토큰 사용, 도구 선택 정확도, 에이전트에 등록된 도구 수, 도구 지연 시간이 포함됩니다. 예를 들어 각 도구에서 반환된 출력 토큰을 모니터링하면 도구가 컨텍스트 기간 사용에 대한 임계값을 초과할 때 경보를 설정할 수 있습니다. 도구가 해당 임계값을 초과하면 도구의 동작을 검토하는 것이 좋습니다. 이는 MCP 도구 설계 전략과도 관련이 있습니다. 도구 선택 정확도 지표는 에이전트가 지정된 작업에 적합한 도구를 얼마나 잘 선택하는지 나타내는 반면, 실행 속도와 성공률은 성능 병목 현상과 신뢰성 문제를 강조합니다.

예를 들어 도구 선택 및 도구 사용 정확도 지표를 평가하기 위해 AWS 팀은 회귀 테스트를 위한 골든 데이터 세트를 생성했습니다. 데이터 세트는 사용자 쿼리 시 과거 API 호출 로그의 LLMs 사용하여 합성 방식으로 생성되었습니다. 사전 정의된 도구 선택 및 도구 사용 지표(예: 도구 선택 정확도, 도구 파라미터 정확도, 다중 턴 함수 호출 정확도)를 사용하여 AWS 팀은 AI 에이전트가 적절한 도구를 올바르게 식별하고, 파라미터를 정확한 값으로 채우고, 대화형 턴 전반에 걸쳐 일관된 도구 호출 시퀀스를 유지하는 능력을 객관적으로 평가할 수 있습니다.

에이전트에 등록된 도구 수에 대한 지표를 측정하면 MCP 서버에서 제공하는 사용 가능한 도구의 변경 사항뿐만 아니라 잠재적인 컨텍스트 기간 관리 문제를 식별하는 데 도움이 될 수 있습니다. MCP 서버 및 도구에 대한 사용자 경험을 나타내는 운영 지표를 정기적으로 검토해야 합니다.

기여자

작성

- Alex Torres, 선임 솔루션 아키텍트, AWS
- Saikat Gomes, 선임 고객 솔루션 관리자, AWS
- Mike Haken, 수석 수석 솔루션 아키텍트, AWS
- Sreeja Das, 수석 엔지니어, AWS

검토

- Ted Swinyar, 솔루션 아키텍트 관리자, AWS
- Raju Patil, 선임 데이터 과학자, AWS

기술 작성

- GxP AbouHarb, Senior Technical Writer, AWS

문서 기록

아래 표에 이 가이드의 주요 변경 사항이 설명되어 있습니다. 향후 업데이트에 대한 알림을 받으려면 RSS 피드를 구독하십시오.

변경 사항	설명	날짜
최초 게시	—	2026년 3월 16일

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.