통합 모델-as-a-Service(MaaS) 생태계 구축: 차세대 AI 플레이그라운드를 위한 2026 청사진

요약 (Executive Summary)

2025년과 2026년에 걸쳐 생성형 AI(Generative AI) 지형이 급격히 다변화됨에 따라 개인 및 엔터프라이즈 AI 인프라에 대한 요구 사항은 근본적으로 변화하였다. 단일 공급자에 의존하던 단순한 통합 방식은 이제 독점적인 엔드포인트, 상이한 스키마, 그리고 복잡한 규정 준수 요구 사항이 얽힌 매트릭스 구조로 진화하였다.[1] "다시 시작(Start Again)"이라는 명제는 단순히 서버를 재부팅하는 수준을 넘어, 개발자가 인공지능과 상호 작용하는 기본 계층을 재설계해야 한다는 전략적 명령으로 해석된다. 과거의 임시방편적인 래퍼(Wrapper) 방식은 멀티모달 추론, 에이전트 도구 사용(Tool Use), 그리고 Bring Your Own Key(BYOK) 아키텍처의 엄격한 보안 요구 사항을 더 이상 감당할 수 없게 되었다.[4]

본 보고서는 AWS Bedrock, Google Vertex AI, Azure OpenAI와 같은 다양한 공급자를 단일하고 일관된 인터페이스로 통합하기 위한 모범 사례를 수립하고, 통합되고 견고하며 확장 가능한 AI 플레이그라운드 및 MaaS(Model-as-a-Service) 라이브러리를 구축하기 위한 포괄적인 기술 청사진을 제시한다.[2] 80개 이상의 연구 자료를 종합하여 실시간 비용 추적, 지연 시간 벤치마킹, 그리고 모델 컨텍스트 프로토콜(MCP)을 포함한 프로토콜 표준화를 구현하는 데 필요한 아키텍처 패턴을 심층 분석하며, AI 인프라에 대한 통제권을 회복하고자 하는 숙련된 개발자를 위한 결정적인 가이드를 제공한다.

---

1. 통합 AI 게이트웨이 아키텍처의 필연성 (The Imperative for a Unified AI Gateway Architecture)

대규모 언어 모델(LLM) 시장의 파편화는 심각한 통합 병목 현상을 초래하고 있다. 개발자들은 이제 단순히 한두 개의 모델 사이에서 선택하는 것이 아니라, 코딩, 추론, 창의적 생성 등 특화된 모델들이 서로 다른 호스팅 제공업체에 분산되어 있는 거대한 "모델 쇼핑몰(Model Mall)"을 항해해야 하는 상황에 직면해 있다.[2] 이러한 환경에서 단일 애플리케이션이 추론을 위해 Azure나 OpenAI의 gpt-4o를, 코딩을 위해 Anthropic이나 AWS Bedrock의 claude-3-5-sonnet을, 그리고 긴 문맥 검색을 위해 Vertex AI의 gemini-1.5-pro를 동시에 요구하는 것은 흔한 일이 되었다.[3]

1.1 파편화가 초래한 기술적 부채 (The Fragmentation Challenge)

각 공급자는 고유한 인증 헤더, 페이로드 구조, 오류 처리 메커니즘을 강제한다. 이는 단순한

문법적 차이를 넘어 운영상의 복잡성을 기하급수적으로 증가시킨다.

- **API** 비일관성**(API Inconsistency):** OpenAI의 챗 완성(Chat Completion) 스키마가 사실상의 표준으로 자리 잡았음에도 불구하고, Google Vertex AI와 AWS Bedrock과 같은 주요 제공업체들은 여전히 자사의 네이티브 API에 대해 독자적인 스키마를 유지하고 있다.[8] 예를 들어, 메시지 배열의 구조, 시스템 프롬프트의 위치, 도구 정의 방식 등이 상이하여 개발자는 각 제공업체별로 별도의 어댑터 로직을 구현해야 한다.
- 운영 오버헤드**(Operational Overhead):** 각 공급자별로 별도의 결제 계정, 속도 제한(Rate Limit), 관측성 파이프라인을 관리하는 것은 유지보수 부담을 가중시킨다. 특정 모델의 장애 발생 시, 다른 제공업체의 모델로 신속하게 전환(Failover)하기 위해서는 사전에 구축된 복잡한 라우팅 로직이 필요하다.[10]
- 벤더 종속 위험**(Vendor Lock-in Risks):** 특정 공급자의 SDK에 직접 통합하는 방식은 코드와 인프라 간의 강한 결합(Tight Coupling)을 초래한다. 이는 가격 정책 변경이나 서비스 중단 시 유연한 대응을 불가능하게 만들며, 장기적으로 기술적 자율성을 저해한다.[12]

1.2 게이트웨이 솔루션과 전략적 이점 **(The Gateway Solution)**

이러한 마찰을 해결하기 위한 아키텍처 표준으로 "AI 게이트웨이" 또는 "서비스 애그리게이터(Service Aggregator)" 패턴이 부상하고 있다. 미들웨어 계층으로 작동하는 게이트웨이는 애플리케이션 로직을 기본 모델 공급자로부터 분리(Decoupling)한다.[13] 이 아키텍처는 모델의 '핫 스왑(Hot-swapping)', 통합 인증, 중앙화된 정책 집행을 가능하게 한다.

표 **1: AI** 게이트웨이 아키텍처의 전략적 이점 **(Strategic Advantages of an AI Gateway Architecture)**

기능 **(Feature)**	설명 **(Description)**	운영상 이점 **(Operational Benefit)**
통합 인터페이스 **(Unified Interface)**	모든 다운스트림 API를 단일 표준(주로 OpenAI 호환)으로 정규화한다.	코드 복잡성을 줄이고 "한 번 작성하여 어디서나 실행(Write Once, Run Anywhere)" 로직을 가능하게 한다.[15]
스마트 라우팅 **(Smart Routing)**	지연 시간, 비용, 또는 모델 상태에 따라 요청을 동적으로 라우팅한다.	자동 장애 조치(Failover)를 통해 신뢰성을 높이고 비용을 최적화한다.[10]
중앙화된 거버넌스 **(Centralized Governance)**	엣지에서 속도 제한, PII(개인정보) 수정, 액세스	규정 준수를 보장하고 팀 간 예산 초과를 방지한다.[17]

	정책을 집행한다.	
프로토콜 변환 **(Protocol Translation)**	이질적인 형식 간에 요청/응답을 변환한다 (예: OpenAI 도구 호출을 Bedrock Converse API로 변환).	커스텀 어댑터 없이 모든 모델에서 함수 호출과 같은 고급 기능을 사용할 수 있게 한다.[19]

1.3 "다시 시작"의 철학: 2026년을 위한 재건 (The "Start Again" Philosophy)

"다시 시작"한다는 것은 과거의 깨지기 쉽고 하드코딩된 통합 방식을 거부하고, 회복 탄력적이며 설정 기반(Configuration-driven)의 아키텍처를 채택함을 의미한다. 현대의 AI 스택은 모듈성을 특징으로 한다. 모델을 정적인 의존성이 아닌 표준화된 제어 평면을 통해 접근 가능한 교체 가능한 상품으로 취급한다.[21] 이러한 전환은 복잡성을 애플리케이션 계층에서 인프라 계층으로 이동시키며, 프로토콜 변환과 상태 관리를 처리할 수 있는 강력한 MaaS 라이브러리를 필요로 한다. 이는 단순한 기술적 업그레이드가 아니라, AI 시스템을 설계하고 운영하는 사고방식의 근본적인 전환을 요구한다.

---

2. 게이트웨이 기술의 비교 분석 및 선정 (Comparative Analysis of Gateway Technologies)

MaaS 라이브러리의 올바른 기반을 선정하기 위해서는 수많은 오픈 소스 및 상용 게이트웨이 시장을 면밀히 분석해야 한다. 본 분석에서는 고성능 개인 및 팀 플레이그라운드 구축에 적합한 세 가지 주요 후보인 LiteLLM, Portkey, 그리고 Bifrost와 같은 고성능 대안을 식별하고 평가한다.

2.1 LiteLLM: 표준화를 위한 사실상의 표준 (The De Facto Standard)

LiteLLM은 LLM 상호 작용을 표준화하기 위한 선도적인 오픈 소스 라이브러리로 자리 잡았다. 가장 큰 강점은 100개 이상의 모델에 대한 방대한 지원 범위와 OpenAI 입출력 형식에 대한 엄격한 준수이다.[12]

- 메커니즘: LiteLLM은 OpenAI 형식의 요청을 가로채어 대상 공급자(예: Hugging Face, Bedrock, Vertex AI)의 기본 형식으로 변환하는 상태 비저장(Stateless) 프록시로 기능한다.[23]
- 장점: 진입 장벽이 매우 낮고, Python 기반 워크플로우와 원활하게 통합되며, 새로운 모델에 대한 커뮤니티 지원이 활발하다.[12]
- 단점: Python 기반 솔루션으로서, Go나 Rust와 같은 컴파일 언어에 비해 극심한 동시성 상황에서 지연 시간 오버헤드가 발생할 수 있다.[10] 일부 벤치마크에서는 높은 처리량에서 성능 저하가 관찰되기도 한다.[24]

2.2 Portkey: 엔터프라이즈 오케스트레이터 (The Enterprise Orchestrator)

Portkey는 AI 배포의 "Day 2" 운영, 즉 신뢰성과 관측성에 집중함으로써 차별화된다. 가상 엣지 또는 사이드카(Sidecar) 형태로 배포할 수 있는 "게이트웨이" 솔루션을 제공한다.[13]

- 메커니즘: Portkey는 재시도, 폴백(Fallback), 시맨틱 캐싱과 같은 고급 로직으로 요청을 감싼다. 코드가 아닌 외부에서 프롬프트와 구성을 관리할 수 있는 제어 평면을 제공한다.[22]
- 장점: 업계 최고 수준의 관측성 기능을 제공하며, 강력한 폴백 전략(예: Azure 장애 시 OpenAI로 라우팅)을 지원한다. 250개 이상의 모델을 지원한다.[25]
- 단점: 전체 기능 세트는 사용자를 관리형 클라우드 서비스로 유도하는 경향이 있으며, 순수한 개인용 플레이그라운드 구축 시 불필요한 복잡성을 초래할 수 있다.[13]

2.3 고성능 대안: Bifrost와 Kong (High-Performance Alternatives)

실시간 에이전트나 음성 인터페이스와 같이 원시 지연 시간(Raw Latency)과 처리량을 우선시하는 사용자에게는 Go 또는 Rust 기반의 아키텍처가 주목받고 있다.

- **Bifrost:** Go 언어로 작성된 Bifrost는 Python 기반 대안인 LiteLLM보다 훨씬 빠른 속도를 자랑하며, 11µs 미만의 오버헤드를 보장한다.[10] 이는 첫 토큰 생성 시간(Time to First Token, TTFT)이 중요한 고동시성 환경에 최적화되어 있다.
- **Kong AI Gateway:** 검증된 Nginx/Lua 아키텍처를 활용하여 기존 엔터프라이즈 API 게이트웨이 내에서 AI 트래픽 관리를 가능하게 한다. 이미 마이크로서비스 오케스트레이션을 위해 Kong을 사용하고 있는 환경에서 탁월한 성능을 발휘한다.[24]

"다시 시작" 아키텍처를 위한 분석:

개발 편의성과 기능적 풍부함의 균형을 맞춘 포괄적인 개인용 MaaS 라이브러리를 위해서는 **LiteLLM** 기반 아키텍처가 최적의 출발점이다. 그러나 스트리밍 및 모니터링 시 애플리케이션 이벤트 루프를 차단하지 않고 고동시성 요구를 처리하기 위해서는, 기본 프록시 계층으로 **Go** 또는 **Rust** 기반의 리버스 프록시(Bifrost 또는 Helicone에서 영감을 받은)를 도입하는 것이 권장된다. 이는 Python의 GIL(Global Interpreter Lock) 제약을 우회하고, 수천 개의 동시 연결을 효율적으로 처리할 수 있게 한다.

---

3. 통합 API 스키마 설계 (Designing the Unified API Schema)

MaaS 라이브러리 구축의 핵심 과제는 애플리케이션이 하나의 언어로 말하면서도 다수의 공급자와 소통할 수 있게 하는 "공용어(Lingua Franca)"를 만드는 것이다. OpenAI의 Chat Completion API 스키마가 보편적 표준으로 부상했지만, 이를 AWS Bedrock이나 Google Vertex AI와 같은 공급자 전반에 걸쳐 견고하게 구현하려면 정교한 번역 로직이 필요하다.

3.1 표준 요청 모델 (The Canonical Request Model)

시스템은 공급자에 구애받지 않는 표준 요청 객체를 정의해야 한다. 이 객체는 OpenAI 스키마를 미러링하지만, 공급자별 특정 매개변수를 위한 확장 필드를 포함해야 한다.[15]

JSON

```
{
 "model": "provider_prefix/model_name",
 "messages":,
 "temperature": 0.7,
 "max_tokens": 1000,
 "stream": true,
 "tools": [...],
 "provider_config": {
  "aws_region": "us-east-1",
  "vertex_project_id": "gen-ai-proj",
  "safety_settings": [...]
 }
}
```

이러한 구조는 클라이언트가 특정 공급자의 SDK 세부 사항을 알 필요 없이 요청을 구성할 수 있게 하며, 백엔드에서 이를 해석하여 적절한 API 호출로 변환할 수 있는 유연성을 제공한다.

3.2 간극 메우기: AWS Bedrock Converse API

과거 AWS Bedrock은 모델별로 고유한 JSON 페이로드 구조(예: Claude와 Llama가 서로 다름)를 요구했으나, **Converse API**의 도입으로 AWS 생태계 내에서의 상호 작용이 표준화되었다. 그러나 여전히 OpenAI 표준과는 차이가 존재한다.[20]

- 번역 로직 **(Translation Logic):** MaaS 라이브러리는 OpenAI의 messages 배열을 Bedrock의 messages 구조로 매핑해야 한다. 결정적으로, Bedrock은 system 프롬프트를 최상위 매개변수로 취급하는 반면, OpenAI는 이를 메시지 역할(Role)로 취급한다. 따라서 게이트웨이는 system 역할을 가진 메시지를 추출하여 Bedrock의 system 필드로 이동시키고, 나머지 사용자/어시스턴트 메시지만을 messages 배열로 구성해야 한다.[28]
- 도구 사용 매핑 **(Tool Use Mapping):** OpenAI의 functions 또는 tools 스키마는 Bedrock의 toolConfig로 변환되어야 한다. 여기에는 JSON 스키마 타입을 엄격하게 매핑하고, 대화 기록 내에서 toolUse 및 toolResult 블록이 올바르게 형식화되도록 보장하는 작업이 포함된다. 특히 Bedrock은 도구 호출 ID와 결과 간의 엄격한 순서를 요구하므로, 이에 대한 유효성 검사가 필수적이다.[19]

3.3 간극 메우기: Google Vertex AI

Vertex AI의 Gemini API는 인증 방식과 멀티모달 입력 처리에서 독자적인 복잡성을 가진다.

- 인증 **(Authentication):** OpenAI의 단순 API 키 모델과 달리, Vertex AI는 일반적으로 Google Cloud IAM 자격 증명(서비스 계정)이나 엄격한 OAuth 교환을 요구한다. MaaS 라이브러리는 토큰 갱신 수명 주기를 관리하거나, 적용 가능한 경우 새로운 Gemini API 키를 사용하여 이를 추상화해야 한다.[8]
- 매개변수 매핑 **(Parameter Mapping):** OpenAI 모델의 reasoning_effort와 같은 매개변수는 최신 추론 모델을 위해 Gemini의 thinking 매개변수로 매핑되어야 한다. 또한, Vertex 고유의 안전 설정(Safety Settings, 차단 임계값 등)은 provider_config 확장을 통해 노출되어야 하며, 이를 통해 개발자는 모델의 안전성 필터를 세밀하게 제어할 수 있다.[29]

3.4 모델 컨텍스트 프로토콜 (Model Context Protocol, MCP)

라이브러리의 미래 호환성을 확보하기 위해 모델 컨텍스트 프로토콜**(MCP)** 통합은 필수적이다. Anthropic이 도입하고 오픈 소스 커뮤니티가 채택한 MCP는 LLM이 외부 데이터 및 도구와 상호작용하는 방식을 표준화한다.[30]

- 아키텍처 **(Architecture):** 게이트웨이에 도구를 하드코딩하는 대신, 게이트웨이는 리소스(데이터)와 프롬프트(기능)를 노출하는 "MCP 서버"에 연결된다. 이는 도구의 정의와 실행 로직을 모델 제공자로부터 분리시킨다.
- 구현 **(Implementation):** MaaS 라이브러리는 MCP 클라이언트로 작동하여, 연결된 서버에서 사용 가능한 도구를 쿼리하고 이를 모델의 컨텍스트 윈도우에 동적으로 주입한다. 이를 통해 코드 변경 없이도 동일한 "Google Search" 도구를 Claude, GPT-4, Gemini 등 다양한 모델에서 사용할 수 있게 된다.[32] MCP는 에이전트가 도구의 스키마를 이해하고 실행하는 방식을 통일하여, 멀티 에이전트 시스템 구축의 복잡성을 획기적으로 낮춘다.

---

4. 보안 아키텍처: BYOK의 필수적 구현 (Security Architecture: The BYOK Imperative)

"다시 시작" 아키텍처에서 보안은 선택 사항이 아니다. AI 접근의 중앙화는 공격자들에게 매력적인 표적이 되며, 숙련된 사용자들은 데이터와 결제에 대한 통제권을 유지하기 위해 **Bring Your Own Key (BYOK)** 기능을 요구한다.[4]

4.1 클라이언트 측 키 노출의 위험성 (The Risks of Client-Side Keys)

초기 AI 플레이그라운드에서 흔히 볼 수 있었던 안티 패턴은 사용자가 프론트엔드에 직접 API 키를 입력하게 하고 이를 localStorage에 저장하는 방식이었다. 이는 XSS 공격이나 악성 브라우저 확장 프로그램에 높은 권한의 자격 증명을 그대로 노출시키는 결과를 초래한다.[34]

- 불변의 원칙 **(The Golden Rule):** API 키는 절대로 클라이언트 브라우저에 노출되어서는

안 된다. 이들은 반드시 서버 측에서 관리되거나, 서버가 블라인드 프록시(Blind Proxy) 역할을 할 수 있도록 암호화되어야 한다.[34]

4.2 안전한 BYOK 구현 전략 (Implementing Secure BYOK)

BYOK를 안전하게 지원하기 위해 아키텍처는 "볼트(Vault)" 패턴을 채택해야 한다.

1. 키 수집 **(Key Ingestion):** 사용자는 보안 HTTPS 폼을 통해 API 키를 제공한다. 이 데이터는 전송 계층 보안(TLS)을 통해 보호된다.
2. 저장 시 암호화 **(Encryption at Rest):** 키는 데이터베이스에 저장되기 전에 즉시 대칭 암호화 알고리즘(예: AES-256-GCM)을 사용하여 암호화된다. 이때 암호화에 사용되는 마스터 키(Master Key)는 전용 비밀 관리자(예: AWS Secrets Manager, HashiCorp Vault)에 저장되어야 하며, 애플리케이션 코드 내에 하드코딩되어서는 안 된다.[37]
3. 봉투 암호화 **(Envelope Encryption):** 더 높은 보안 수준을 위해 봉투 암호화를 사용한다. 데이터 암호화 키(DEK)로 API 키를 암호화하고, 키 암호화 키(KEK)로 DEK를 암호화하는 방식이다. 이는 전체 데이터를 재암호화하지 않고도 키를 주기적으로 교체(Rotation)할 수 있게 하여 보안 운영의 유연성을 제공한다.[38]
4. 런타임 복호화 **(Runtime Decryption):** 요청이 발생하면 프록시는 암호화된 키를 검색하고, 보안 인클레이브(Secure Enclave)나 임시 워커 내의 메모리에서 복호화한 뒤, 요청 헤더에 주입하고 트랜잭션 직후 메모리에서 즉시 폐기한다.[39] 이는 키가 메모리에 머무르는 시간을 최소화하여 덤프 공격 등의 위험을 줄인다.

4.3 개인용 플레이그라운드를 위한 종단간 암호화(E2EE) (End-to-End Encryption)

극도의 보안을 요구하는 개인용 플레이그라운드의 경우, **Web Crypto API**를 활용한 클라이언트 측 E2EE 모델을 구현할 수 있다.

- 메커니즘**:** 사용자의 브라우저가 키 쌍(Key Pair)을 생성한다. 개인 키는 절대 장치를 떠나지 않으며, API 키는 서버로 전송되기 전 공개 키로 암호화된다.
- 난제 및 해결**:** 서버는 LLM 공급자를 호출하기 위해 평문 키가 필요하다. 이를 위해 하이브리드 솔루션을 적용한다. 서버는 암호화된 키를 저장하고, 사용자가 세션을 시작할 때 복호화 패스프레이즈를 제공(또는 세션 토큰에서 개인 키를 파생)하여 서버가 활성 세션이나 요청이 진행되는 동안에만 키를 복호화할 수 있도록 허용한다.[40] 이 방식은 편의성과 보안성 사이의 균형을 맞추며, 서버가 침해당하더라도 저장된 키의 안전을 보장한다.

---

5. 관측성 및 비용 엔지니어링 (Observability and Cost Engineering)

통합 MaaS 라이브러리는 투명성을 제공해야 한다. 추론 모델의 단일 실행 비용이 일반 쿼리보다 훨씬 높을 수 있는 시대에, 내부 상황을 모르는 "깜깜이(Flying Blind)" 운영은 허용되지

않는다.

5.1 실시간 비용 추적 (Real-Time Cost Tracking)

비용 추정은 실행 전(Pre-flight)과 실행 후(Post-flight) 두 단계에서 이루어져야 한다.

- 토큰 계산 **(Token Counting):** 요청을 전송하기 전, 라이브러리는 토크나이저(OpenAI의 tiktoken 또는 기타 모델별 토크나이저)를 사용하여 입력 비용을 추정해야 한다. 제공자마다 토큰 정의가 다르므로 모델별 맞춤형 토크나이저 통합이 필수적이다.[42]
- 가격 인덱스 **(Pricing Indexes):** 시스템은 모델 ID를 입력, 출력, 캐시 쓰기 등에 대한 백만 토큰당 비용으로 매핑하는 동적 가격 카탈로그(JSON 또는 데이터베이스 테이블)를 필요로 한다. 가격 정책은 빈번하게 변경되므로 이 카탈로그는 업데이트 가능해야 한다.[43]
- 사용량 로깅 **(Usage Logging):** 모든 요청의 메타데이터(사용된 모델, 입력 토큰, 출력 토큰, 제공자 상태)는 분석 저장소(예: ClickHouse, PostgreSQL)에 기록되어 대시보드 시각화를 지원해야 한다.[45] 이를 통해 프로젝트별, 사용자별 비용 할당(Chargeback) 및 예산 관리가 가능해진다.

5.2 지연 시간 벤치마킹: TTFT와 E2E (Latency Benchmarking)

성능 지표는 모델 선택에 있어 결정적인 요소이다.

- 첫 토큰 생성 시간 **(TTFT, Time to First Token):** 사용자가 인지하는 지연 시간이다. 요청 전송부터 응답 페이로드의 첫 바이트가 도착할 때까지의 시간을 측정한다. 이는 입력 컨텍스트를 처리하는 프리필(Prefill) 단계에 크게 영향을 받는다.[47]
- 종단간 지연 시간 **(E2E Latency):** 전체 응답이 완료될 때까지의 총 시간이다.
- 초당 토큰 수 **(TPS, Tokens Per Second):** 생성 속도(디코드 단계)를 나타내는 지표이다.
- 계측 **(Instrumentation):** 게이트웨이는 요청 시작, 첫 청크 수신(스트리밍), 스트림 종료 시점에 타임스탬프를 주입해야 한다. 이 델타 값들을 계산하고 저장하여 제공자 간 성능을 비교하는 히트맵(예: "Llama 3 구동 시 Groq가 Bedrock보다 실제로 더 빠른가?")을 생성할 수 있다.[49] 이는 정량적 데이터에 기반한 모델 선택을 가능하게 한다.

5.3 시맨틱 캐싱 (Semantic Caching)

비용과 지연 시간을 획기적으로 줄이기 위해 라이브러리는 시맨틱 캐싱을 구현해야 한다. 단순 문자열 일치 기반 캐싱 대신, 임베딩을 사용하여 의미적으로 유사한 쿼리를 찾는다. 사용자가 캐시된 쿼리와 95% 유사한 질문을 하면, 시스템은 값비싼 API 호출 대신 저장된 응답을 반환한다.[51] 이를 위해서는 고성능 벡터 데이터베이스(예: Redis, Qdrant)와의 통합이 필요하며, 캐시 히트율(Cache Hit Rate)은 중요한 비용 절감 지표로 추적되어야 한다.

---

6. 인프라 및 멀티 테넌시 (Infrastructure and Multi-Tenancy)

"플레이그라운드"를 구축한다는 것은 실험을 격리하고, 추적하고, 반복할 수 있는 환경을

조성함을 의미한다. 이를 위해서는 데이터와 리소스의 논리적 분리가 필수적이다.

6.1 멀티 테넌시를 위한 데이터베이스 스키마 (Database Schema for Multi-Tenancy)

단일 사용자 도구라 할지라도, 멀티 테넌트 데이터베이스 스키마는 "프로젝트" 또는 "워크스페이스"의 논리적 분리를 가능하게 한다.

- 테넌시 모델 **(Tenancy Model):** PostgreSQL의 "테넌트별 스키마(Schema-per-tenant)" 또는 "행 수준 보안(Row-level security, RLS)" 모델이 권장된다. projects 테이블은 api_keys, prompts, logs와 연결된다. 모든 쿼리는 자동으로 project_id 또는 user_id로 필터링되어 데이터 격리를 보장한다.[53]
- 엔티티 관계 **(Entity Relationships):**
  - Provider: 공급자에 대한 구성을 저장한다 (예: "My Azure Instance").
  - Model: 공급자와 연결되며, 가격 책정 및 기능 플래그(supports_vision, supports_tools)를 저장한다.
  - RequestLog: 상호 작용의 전체 트레이스(Trace)를 저장하며, 프롬프트 스냅샷, 모델 구성, 비용 지표를 포함한다.[55]

6.2 배포 전략 (Deployment Strategy)

개인 또는 소규모 팀을 위한 설정에서는 Docker Compose나 Kubernetes를 통한 컨테이너화된 접근 방식이 표준이다.

- 제어 평면 **(Control Plane):** UI와 API를 제공하는 경량 Node.js (Next.js) 또는 Python (FastAPI) 애플리케이션.
- 데이터 평면 **(Data Plane):** 프록시/게이트웨이 (LiteLLM 또는 커스텀 Go 서비스).
- 저장소 계층 **(Storage Layer):** 관계형 데이터를 위한 PostgreSQL과 속도 제한 및 캐싱을 위한 Redis.[16]
- 서버리스 고려사항**:** AWS Lambda나 Vercel Edge와 같은 서버리스 함수는 인기가 높지만, "콜드 스타트(Cold Start)" 지연 시간을 유발하여 TTFT에 부정적인 영향을 미칠 수 있다. 고성능 플레이그라운드를 위해서는 프록시 계층에 장기 실행 컨테이너나 엣지 최적화 워커(Cloudflare Workers)를 사용하는 것이 바람직하다.[57]

---

7. 사용자 경험**:** 플레이그라운드 **UI** 구축 **(The User Experience: Building the Playground UI)**

인터페이스는 아키텍처가 사용자와 만나는 접점이다. 2026년형 플레이그라운드는 단순한 채팅창을 넘어선 통합 개발 환경(IDE)에 가까운 경험을 제공해야 한다.

7.1 핵심 **UI** 컴포넌트 **(Key UI Components)**

- 통합 구성 패널 **(Unified Configuration Panel):** 사용자가 여러 공급자의 키를 한 번 입력하고, 채팅 중에 드롭다운 메뉴에서 이를 선택할 수 있는 중앙화된 설정 영역이다.

- 사이드 바이 사이드 비교 **(Side-by-Side Comparison):** "아레나(Arena)" 뷰를 통해 동일한 프롬프트를 두 개 이상의 모델에 동시에 실행하여 출력 품질, 지연 시간, 추론 패턴을 직관적으로 비교할 수 있어야 한다.[59] 이는 모델 선정 과정에서 필수적인 도구이다.
- 프롬프트 엔지니어링 워크벤치 **(Prompt Engineering Workbench):** 프롬프트 템플릿 저장, 버전 관리, 변수 주입(예: {{customer_name}}) 기능을 제공해야 한다. LangFuse나 Agenta와 같은 도구와의 통합은 프롬프트 관리의 효율성을 높인다.[55]
- 실시간 지표 **HUD (Real-Time Metrics HUD):** 현재 세션의 비용과 마지막 응답의 지연 시간을 실시간으로 보여주는 헤드업 디스플레이 기능이다. 이는 사용자가 비용 효율적인 결정을 내리는 데 도움을 준다.[62]

7.2 스트리밍 및 아티팩트 처리 **(Handling Streaming & Artifacts)**

- 스트리밍 프로토콜**:** Server-Sent Events (SSE)는 LLM 토큰을 프론트엔드로 전달하는 표준이다. UI는 이러한 스트림을 견고하게 처리하며, 마크다운과 코드 블록을 실시간으로 파싱해야 한다.[63]
- 생성형 **UI/**아티팩트 **(Generative UI/Artifacts):** Claude Artifacts에서 영감을 받아, 플레이그라운드는 모델이 생성한 HTML/React 컴포넌트를 샌드박스 처리된 iframe 내에서 렌더링할 수 있어야 한다. 이를 통해 채팅 인터페이스는 단순한 텍스트 교환을 넘어 코딩 및 프로토타이핑 워크스페이스로 변모한다.[65]

---

8. 법적 및 규정 준수 고려사항 **(Legal and Compliance Considerations)**

기술적 구현 외에도, MaaS 라이브러리를 구축하고 운영할 때는 법적, 규제적 함의를 반드시 고려해야 한다. 이는 특히 데이터를 제3자 모델 공급자에게 전송하는 구조적 특성 때문이다.

8.1 데이터 주권 및 저작권 **(Data Sovereignty & Copyright)**

사용자의 프롬프트 데이터와 모델의 출력물에 대한 권리 관계를 명확히 해야 한다. 최근 법원 판결과 정책 동향은 AI 모델 학습에 저작물 사용을 공정 이용(Fair Use)으로 인정하는 경향이 있으나, 출력물에 대한 책임 소재는 여전히 복잡한 문제이다.[66] BYOK 모델을 채택함으로써 데이터 처리의 주체를 사용자로 명확히 하고, 라이브러리 운영자의 법적 리스크를 경감할 수 있다.

8.2 오픈 소스 라이선스 준수 **(Open Source Compliance)**

통합하는 오픈 소스 모델(Llama, Mistral 등)의 라이선스 조건을 준수해야 한다. 일부 모델은 상업적 이용에 제한을 두거나 특정 고지 의무를 부과한다. MaaS 라이브러리는 사용자가 선택한 모델의 라이선스 정보를 UI상에 명시적으로 표시하여 사용자가 규정을 준수하도록 유도해야 한다.[68]

---

9. 결론: 미래를 향한 경로 (Conclusion: The Path Forward)

"다시 시작"이라는 명령은 초기 생성형 AI 시대의 기술적 부채를 청산할 수 있는 기회이다. 통합 게이트웨이 아키텍처를 채택하고, 핵심에서 BYOK 보안을 강제하며, 관측성을 일등 시민으로 대우함으로써, 개발자는 단순한 래퍼가 아닌 회복 탄력적인 플랫폼으로서의 MaaS 라이브러리를 구축할 수 있다.

이 아키텍처는 미래 지향적인 기반을 마련한다. 다음 혁신이 OpenAI에서 오든, Google에서 오든, 아니면 로컬 GPU에서 실행되는 오픈 소스 모델에서 오든, 게이트웨이는 안정적인 오케스트레이터로 남아 있을 것이다. 이를 통해 구축자는 API 통합이라는 배관 작업보다는 지능을 응용하여 실제 문제를 해결하는 본질적인 가치에 집중할 수 있게 된다. 인프라의 복잡성은 이를 무시함으로써 해결되는 것이 아니라, 견고하고 표준화되었으며 안전한 설계 안에 캡슐화함으로써 길들여진다.

참고 자료

1. Vertex AI Platform | Google Cloud, 2월 11, 2026에 액세스, https://cloud.google.com/vertex-ai
2. AWS Bedrock vs. Azure AI vs. Google Vertex | Xenoss Blog, 2월 11, 2026에 액세스, https://xenoss.io/blog/aws-bedrock-vs-azure-ai-vs-google-vertex-ai
3. Azure AI Foundry vs AWS Bedrock vs Google Vertex AI: The 2025 Guide - GoPenAI, 2월 11, 2026에 액세스, https://blog.gopenai.com/azure-ai-foundry-vs-aws-bedrock-vs-google-vertex-ai-the-2025-guide-25a69c1d19b1
4. What is Bring Your Own Key (BYOK) Encryption? - Thales, 2월 11, 2026에 액세스, https://cpl.thalesgroup.com/faq/key-secrets-management/what-bring-your-own-key-byok
5. BYOK Explained: A Comprehensive Guide to Bring Your Own Key - Daon, 2월 11, 2026에 액세스, https://www.daon.com/resource/how-byok-empowers-organizations-with-true-data-ownership/
6. Amazon Bedrock vs Azure OpenAI vs Google Vertex AI: An In-Depth Analysis, 2월 11, 2026에 액세스, https://www.cloudoptimo.com/blog/amazon-bedrock-vs-azure-openai-vs-google-vertex-ai-an-in-depth-analysis/
7. Building a Foundation Model as a Service (FMaaS) on AWS, 2월 11, 2026에 액세스, https://builder.aws.com/content/2hPu03PLX1F3UYpMzVwxi2Abk6O/building-a-foundation-model-as-a-service-fmaas-on-aws
8. VertexAI [Gemini] - LiteLLM Docs, 2월 11, 2026에 액세스, https://docs.litellm.ai/docs/providers/vertex
9. AWS Bedrock - LiteLLM, 2월 11, 2026에 액세스, https://docs.litellm.ai/docs/providers/bedrock
10. List of Top 5 AI Gateways: Features + Comprehensive comparison - DEV

Community, 2월 11, 2026에 액세스, https://dev.to/pranay_batta/list-of-top-5-ai-gateways-features-comprehensive-comparison-h0g
11. AI API Aggregation: Managing Costs And Complexity Across Multiple LLMs - CloudZero, 2월 11, 2026에 액세스, https://www.cloudzero.com/blog/ai-api-aggregation/
12. Introducing any-llm: A unified API to access any LLM provider - Mozilla.ai Blog, 2월 11, 2026에 액세스, https://blog.mozilla.ai/introducing-any-llm-a-unified-api-to-access-any-llm-provider/
13. List of Top 5 LLM Gateways in 2025 - Maxim AI, 2월 11, 2026에 액세스, https://www.getmaxim.ai/articles/list-of-top-5-llm-gateways-in-2025/
14. What Is An AI Gateway? | IBM, 2월 11, 2026에 액세스, https://www.ibm.com/think/topics/ai-gateway
15. Guidance for Multi-Provider Generative AI Gateway on AWS, 2월 11, 2026에 액세스, https://aws.amazon.com/solutions/guidance/multi-provider-generative-ai-gateway-on-aws/
16. Building a Unified AI Models API with Node.js | by JawherKl | Feb, 2026 - Medium, 2월 11, 2026에 액세스, https://medium.com/@jawherkl/building-a-unified-ai-models-api-with-node-js-cc4b611e126a
17. What is an AI Gateway? - Kong Inc., 2월 11, 2026에 액세스, https://konghq.com/blog/enterprise/what-is-an-ai-gateway
18. Create a Generative AI Gateway to allow secure and compliant consumption of foundation models | Artificial Intelligence - AWS, 2월 11, 2026에 액세스, https://aws.amazon.com/blogs/machine-learning/create-a-generative-ai-gateway-to-allow-secure-and-compliant-consumption-of-foundation-models/
19. Use a tool to complete an Amazon Bedrock model response, 2월 11, 2026에 액세스, https://docs.aws.amazon.com/bedrock/latest/userguide/tool-use.html
20. [Feature]: Using the Amazon Bedrock Converse API · Issue #4000 · BerriAI/litellm - GitHub, 2월 11, 2026에 액세스, https://github.com/BerriAI/litellm/issues/4000
21. Models-as-a-Service: Let's use AI, not just talk about it - Red Hat, 2월 11, 2026에 액세스, https://www.redhat.com/en/blog/models-service-lets-use-ai-not-just-talk-about-it
22. Best AI Gateway Solutions, 2월 11, 2026에 액세스, https://portkey.ai/buyers-guide/ai-gateway-solutions
23. LiteLLM - Getting Started | liteLLM, 2월 11, 2026에 액세스, https://docs.litellm.ai/
24. AI Gateway Benchmark: Kong AI Gateway, Portkey, and LiteLLM | Kong Inc., 2월 11, 2026에 액세스, https://konghq.com/blog/engineering/ai-gateway-benchmark-kong-ai-gateway-portkey-litellm
25. Portkey-AI/gateway: A blazing fast AI Gateway with integrated guardrails. Route to 200+ LLMs, 50+ AI Guardrails with 1 fast & friendly API. - GitHub, 2월 11, 2026에

액세스, https://github.com/Portkey-AI/gateway
26. Most Trusted Open Source API Gateway - Kong Inc., 2월 11, 2026에 액세스, https://konghq.com/products/kong-gateway
27. Build multi-cloud multi-model generative AI applications with Azure OpenAI, Vertex AI and Amazon Bedrock | Medium, 2월 11, 2026에 액세스, https://medium.com/@adtanasa/multi-cloud-multi-model-generative-ai-apps-with-azure-openai-vertex-ai-and-amazon-bedrock-73c2a90faca5
28. OpenAI models - Amazon Bedrock - AWS Documentation, 2월 11, 2026에 액세스, https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-openai.html
29. Gemini - Google AI Studio - LiteLLM, 2월 11, 2026에 액세스, https://docs.litellm.ai/docs/providers/gemini
30. What is Model Context Protocol (MCP)? A guide - Google Cloud, 2월 11, 2026에 액세스, https://cloud.google.com/discover/what-is-model-context-protocol
31. Introducing the Model Context Protocol - Anthropic, 2월 11, 2026에 액세스, https://www.anthropic.com/news/model-context-protocol
32. What is the Model Context Protocol (MCP)? - Model Context Protocol, 2월 11, 2026에 액세스, https://modelcontextprotocol.io/
33. How to Build Secure AI Agents with Kong MCP Proxy, 2월 11, 2026에 액세스, https://konghq.com/blog/engineering/secure-ai-agents-volcano-sdk-kong-mcp-proxy
34. Best Practices for API Key Safety | OpenAI Help Center, 2월 11, 2026에 액세스, https://help.openai.com/en/articles/5112595-best-practices-for-api-key-safety
35. Bring your own API key on client side? : r/react - Reddit, 2월 11, 2026에 액세스, https://www.reddit.com/r/react/comments/17lsmf4/bring_your_own_api_key_on_client_side/
36. Best practices for managing API keys | Authentication - Google Cloud Documentation, 2월 11, 2026에 액세스, https://docs.cloud.google.com/docs/authentication/api-keys-best-practices
37. 10 API Key Management Best Practices - Serverion, 2월 11, 2026에 액세스, https://www.serverion.com/uncategorized/10-api-key-management-best-practices/
38. Demystifying AWS KMS key operations, bring your own key (BYOK), custom key store, and ciphertext portability | AWS Security Blog, 2월 11, 2026에 액세스, https://aws.amazon.com/blogs/security/demystifying-kms-keys-operations-bring-your-own-key-byok-custom-key-store-and-ciphertext-portability/
39. How to Store API Keys Securely - Strapi, 2월 11, 2026에 액세스, https://strapi.io/blog/how-to-store-API-keys-securely
40. End-to-End Encrypted Chat with JS & Web Crypto API - GetStream.io, 2월 11, 2026에 액세스, https://getstream.io/blog/web-crypto-api-chat/
41. Where and how to store private keys in web applications for private messaging with web browsers - Cryptography Stack Exchange, 2월 11, 2026에 액세스, https://crypto.stackexchange.com/questions/35530/where-and-how-to-store-private-keys-in-web-applications-for-private-messaging-wi
42. LLM Cost Estimation Guide: From Token Usage to Total Spend | by Alpha

Iterations, 2월 11, 2026에 액세스, https://medium.com/@alphaiterations/llm-cost-estimation-guide-from-token-usage-to-total-spend-fba348d62824
43. Introducing OpenInfraQuote: An Open-Source Terraform Cost Estimation Tool, 2월 11, 2026에 액세스, https://blog.localstack.cloud/introducing-openinfraquote-open-source-terraform-cost-estimation-tool/
44. Model Usage & Cost Tracking for LLM applications (open source) - Langfuse, 2월 11, 2026에 액세스, https://langfuse.com/docs/observability/features/token-and-cost-tracking
45. Cost Tracking - Tetrate, 2월 11, 2026에 액세스, https://tetrate.io/learn/ai/cost-tracking
46. Use the AI Proxy - Braintrust, 2월 11, 2026에 액세스, https://www.braintrust.dev/docs/deploy/ai-proxy
47. How to Estimate LLM Time to First Token (TTFT) - ApX Machine Learning, 2월 11, 2026에 액세스, https://apxml.com/posts/how-to-estimate-llm-time-to-first-token-ttft
48. Metrics — NVIDIA NIM LLMs Benchmarking, 2월 11, 2026에 액세스, https://docs.nvidia.com/nim/benchmarking/llm/latest/metrics.html
49. Solved: Measuring latency metrics like TTFT, TBT when depl... - Databricks Community, 2월 11, 2026에 액세스, https://community.databricks.com/t5/generative-ai/measuring-latency-metrics-like-ttft-tbt-when-deploying-agents-on/td-p/138208
50. Announcing VDBBench 1.0: Open-Source Vector Database Benchmarking with Your Real-World Production Workloads - Milvus, 2월 11, 2026에 액세스, https://milvus.io/blog/vdbbench-1-0-benchmarking-with-your-real-world-production-workloads.md
51. Managing AI APIs: Best Practices for Secure and Scalable AI API Consumption, 2월 11, 2026에 액세스, https://devops.com/managing-ai-apis-best-practices-for-secure-and-scalable-ai-api-consumption/
52. What Is LLM Proxy? - TrueFoundry, 2월 11, 2026에 액세스, https://www.truefoundry.com/blog/llm-proxy
53. Multi-tenant Application Database Design | by Blake Howe - Medium, 2월 11, 2026에 액세스, https://blakebhowe.medium.com/multi-tenant-application-database-design-e4c2d161f3dd
54. Designing a multi-tenant database for a scenario with multiple user types, 2월 11, 2026에 액세스, https://dba.stackexchange.com/questions/264567/designing-a-multi-tenant-database-for-a-scenario-with-multiple-user-types
55. 8 Best Prompt Engineering Tools for AI Teams in 2025 - Maxim AI, 2월 11, 2026에 액세스, https://www.getmaxim.ai/articles/8-best-prompt-engineering-tools-for-ai-teams-in-2025/

56. Best practices for implementing machine learning on Google Cloud, 2월 11, 2026에 액세스, https://docs.cloud.google.com/architecture/ml-on-gcp-best-practices
57. Introducing Moltworker: a self-hosted personal AI agent, minus the minis, 2월 11, 2026에 액세스, https://blog.cloudflare.com/moltworker-self-hosted-ai-agent/
58. AI proxy: fostering a more open ecosystem - Blog - Braintrust, 2월 11, 2026에 액세스, https://www.braintrust.dev/blog/ai-proxy
59. Model Compare Playground UI - LiteLLM Docs, 2월 11, 2026에 액세스, https://docs.litellm.ai/docs/proxy/model_compare_ui
60. Side-by-Side LLM Output Comparison — Data Labeling - Label Studio, 2월 11, 2026에 액세스, https://labelstud.io/templates/llm_side_by_side
61. 8 Best Prompt Engineering Tools in 2025 - Mirascope, 2월 11, 2026에 액세스, https://mirascope.com/blog/prompt-engineering-tools
62. How do you guys handle using multiple AI APIs? : r/ArtificialInteligence - Reddit, 2월 11, 2026에 액세스, https://www.reddit.com/r/ArtificialInteligence/comments/1o640gv/how_do_you_guys_handle_using_multiple_ai_apis/
63. Workers AI LLM Playground, 2월 11, 2026에 액세스, https://playground.ai.cloudflare.com/
64. Optimizing AI responsiveness: A practical guide to Amazon Bedrock latency-optimized inference | Artificial Intelligence - AWS, 2월 11, 2026에 액세스, https://aws.amazon.com/blogs/machine-learning/optimizing-ai-responsiveness-a-practical-guide-to-amazon-bedrock-latency-optimized-inference/
65. lobehub/lobehub: The ultimate space for work and life — to ... - GitHub, 2월 11, 2026에 액세스, https://github.com/lobehub/lobe-chat
66. Revisiting the Library Copyright Alliance Statement on AI and Copyright, 2월 11, 2026에 액세스, https://www.arl.org/blog/revisiting-the-library-copyright-alliance-statement-on-ai-and-copyright/
67. Federal Court Finds That Training AI on Copyrighted Books is “Quintessentially” Transformative Fair Use | Neal, Gerber & Eisenberg LLP, 2월 11, 2026에 액세스, https://www.nge.com/news-insights/publication/federal-court-finds-that-training-ai-on-copyrighted-books-is-quintessentially-transformative-fair-use/
68. Open Source AI – definition and selected legal challenges | Kluwer Copyright Blog, 2월 11, 2026에 액세스, https://legalblogs.wolterskluwer.com/copyright-blog/open-source-ai-definition-and-selected-legal-challenges/