LUMEN ET VIRTUS
ISSN: 2177-2789

# PROJETO E IMPLANTAÇÃO DE AGENTES DE INTELIGÊNCIA ARTIFICIAL PLUG-AND-PLAY UTILIZANDO ARQUITETURAS SERVERLESS

# DESIGN AND IMPLEMENTATION OF PLUG-AND-PLAY ARTIFICIAL INTELLIGENCE AGENTS USING SERVERLESS ARCHITECTURES

# DISEÑO E IMPLEMENTACIÓN DE AGENTES DE INTELIGENCIA ARTIFICIAL PLUG-AND-PLAY UTILIZANDO ARQUITECTURAS SIN SERVIDOR

https://doi.org/10.56238/levv17n56-035

**Data de submissão:** 12/12/2025 **Data de publicação:** 12/01/2026

**Erick Roberto Furst Brito**
Pós Graduação em Gerencimento de Banco de Dados
Instituição: Centro Universitário de Belo Horizonte (UNI-BH)
E-mail: erick.furst@gmail.com
Lattes: http://lattes.cnpq.br/5747557105880611

## RESUMO

A rápida adoção da inteligência artificial (IA) em ambientes corporativos tem evidenciado uma lacuna recorrente entre protótipos experimentais e sistemas prontos para produção. Embora modelos de linguagem de grande porte, visão computacional e pipelines de aprendizado de máquina estejam cada vez mais acessíveis, as organizações continuam enfrentando desafios relacionados à escalabilidade, ao controle de custos, à complexidade operacional e à integração ao tentar implantar soluções de IA como serviços consumíveis. Este artigo apresenta uma abordagem arquitetural serverless para a construção de agentes de IA modulares e plug-and-play, disponibilizados como APIs, com ênfase nas restrições do mundo real encontradas em ambientes de produção. A arquitetura proposta utiliza funções containerizadas, execução stateless e faturamento baseado no consumo para viabilizar implantação rápida, escalabilidade e eficiência econômica. Além disso, o artigo discute os trade-offs arquiteturais observados entre os modelos de integração REST API e HTTP API, e apresenta lições empíricas obtidas a partir da implantação de múltiplos agentes de IA nos domínios de processamento de linguagem natural, visão computacional e automação de dados.

**Palavras-chave:** Inteligência Artificial. Arquitetura sem Servidor. Computação em Nuvem. Design de API. Agentes de IA. Arquitetura de Software.

## ABSTRACT

The rapid adoption of artificial intelligence (AI) in corporate environments has exposed a recurring gap between experimental prototypes and production-ready systems. While large language models, computer vision, and machine learning pipelines are increasingly accessible, organizations continue to face challenges related to scalability, cost control, operational complexity, and integration when attempting to deploy AI solutions as consumable services. This paper presents a serverless architectural approach for building modular, plug-and-play AI agents deployed as APIs, emphasizing real-world constraints encountered in production environments. The proposed architecture leverages containerized functions, stateless execution, and consumption-based billing to enable rapid deployment, scalability, and economic efficiency. Additionally, the paper discusses architectural trade-offs observed between REST and HTTP API integration models and presents empirical lessons learned

from deploying multiple AI agents across natural language processing, computer vision, and data automation domains.

**Keywords:** Artificial Intelligence. Serverless Architecture. Cloud Computing. API Design. AI Agents. Software Architecture.

**RESUMEN**

La rápida adopción de la inteligencia artificial (IA) en entornos corporativos ha puesto de relieve la creciente brecha entre los prototipos experimentales y los sistemas listos para producción. Si bien los modelos de lenguaje a gran escala, la visión artificial y los procesos de aprendizaje automático son cada vez más accesibles, las organizaciones siguen enfrentándose a desafíos relacionados con la escalabilidad, el control de costes, la complejidad operativa y la integración al intentar implementar soluciones de IA como servicios consumibles. Este artículo presenta un enfoque arquitectónico sin servidor para la creación de agentes de IA modulares, listos para usar y distribuidos como API, haciendo hincapié en las limitaciones reales de los entornos de producción. La arquitectura propuesta utiliza funciones contenedorizadas, ejecución sin estado y facturación basada en el consumo para permitir una implementación rápida, escalabilidad y eficiencia económica. Además, el artículo analiza las compensaciones arquitectónicas observadas entre los modelos de integración de API REST y API HTTP y presenta lecciones empíricas aprendidas al implementar múltiples agentes de IA en los dominios del procesamiento del lenguaje natural, la visión artificial y la automatización de datos.

**Palabras clave:** Inteligencia Artificial. Arquitectura Sin Servidor. Computación en la Nube. Diseño de API. Agentes de IA. Arquitectura de Software.

## 1 INTRODUÇÃO

A crescente demanda por soluções de inteligência artificial em diversos setores tem resultado na proliferação de sistemas de prova de conceito (PoC) que demonstram viabilidade técnica, mas não conseguem evoluir para produtos prontos para produção. Entre os obstáculos mais comuns estão a sobrecarga de gerenciamento de infraestrutura, os custos operacionais imprevisíveis, as dificuldades de escalabilidade das cargas de trabalho e os desafios associados à integração de modelos de IA aos sistemas corporativos existentes.

A computação serverless surgiu como um paradigma promissor para enfrentar esses desafios, ao abstrair o gerenciamento da infraestrutura e permitir uma alocação de recursos mais granular. Quando combinadas com containerização e um design orientado a APIs, as arquiteturas serverless oferecem uma base sólida para a implantação de agentes de IA como serviços modulares e reutilizáveis.

Este artigo explora o projeto e a implantação de agentes de IA como serviços plug-and-play utilizando uma arquitetura serverless, com foco nas decisões arquiteturais, nos padrões de integração e nas lições aprendidas a partir de implantações reais em ambientes de produção.

## 2 DEFINIÇÃO DO PROBLEMA

Apesar dos avanços no desenvolvimento de modelos de IA, as organizações frequentemente enfrentam dificuldades em:

- Transicionar soluções de IA de ambientes experimentais para produção
- Gerenciar a infraestrutura e a complexidade operacional
- Controlar os custos associados a cargas de trabalho variáveis
- Escalar serviços de IA em resposta à demanda
- Disponibilizar interfaces padronizadas e seguras para o consumo de IA

Arquiteturas tradicionais, monolíticas ou baseadas em servidores, tendem a agravar esses desafios ao introduzir um forte acoplamento entre a infraestrutura e a lógica da aplicação. Como resultado, torna-se necessária a adoção de modelos arquiteturais que priorizem modularidade, escalabilidade e simplicidade operacional.

## 3 POR QUE PROVAS DE CONCEITO DE IA FREQUENTEMENTE NÃO EVOLUEM PARA PRODUTOS

O crescimento acelerado das tecnologias de inteligência artificial tem levado organizações de diferentes setores a investirem em provas de conceito (Proofs of Concept – PoCs) como forma de validar rapidamente ideias, algoritmos e hipóteses de negócio. Essas iniciativas costumam demonstrar viabilidade técnica em ambientes controlados, utilizando conjuntos de dados reduzidos, infraestrutura

simplificada e requisitos funcionais limitados. No entanto, uma parcela significativa dessas PoCs não consegue evoluir para produtos prontos para produção.

Uma das principais razões para esse fenômeno está na diferença estrutural entre validação técnica e operação contínua. PoCs são, por definição, experimentais. Elas são desenvolvidas para responder à pergunta *"isso é possível?"*, enquanto produtos precisam responder continuamente a *"isso é confiável, escalável, seguro e sustentável?"*. Essa mudança de foco impõe desafios que raramente são considerados na fase inicial de experimentação.

## 3.1 COMPLEXIDADE DE INFRAESTRUTURA E OPERAÇÃO

PoCs geralmente são executadas em ambientes locais ou em infraestruturas provisionadas manualmente, sem preocupação com alta disponibilidade, tolerância a falhas ou monitoramento. Quando a solução precisa operar de forma contínua, surgem demandas por:

- Escalabilidade automática
- Monitoramento e observabilidade
- Gerenciamento de versões e rollback
- Segurança e controle de acesso

A ausência de uma arquitetura preparada para essas exigências torna a transição para produção onerosa e, muitas vezes, inviável dentro dos prazos e orçamentos disponíveis.

## 3.2 CUSTOS OPERACIONAIS IMPREVISÍVEIS

Em PoCs, o custo computacional costuma ser irrelevante ou artificialmente reduzido. Em produção, entretanto, modelos de IA podem gerar custos significativos devido a:

- Execuções frequentes ou concorrentes
- Uso intensivo de recursos computacionais
- Dependência de serviços externos pagos por requisição

Sem mecanismos de controle de consumo, como faturamento por uso ou limitação de requisições, a solução rapidamente se torna financeiramente insustentável, levando ao seu abandono antes mesmo da consolidação como produto.

## 3.3 FALTA DE PADRONIZAÇÃO E INTEGRAÇÃO

Outro fator recorrente é a dificuldade de integração com sistemas corporativos existentes. PoCs frequentemente utilizam interfaces ad hoc, scripts ou notebooks que não seguem padrões de integração, autenticação ou versionamento. Em ambientes produtivos, espera-se que soluções de IA ofereçam:

- Interfaces padronizadas (APIs)
- Contratos claros de entrada e saída
- Compatibilidade com múltiplos clientes e aplicações

A ausência desses elementos limita a reutilização e a adoção da solução em escala organizacional.

## 3.4 ACOPLAMENTO EXCESSIVO ENTRE MODELO E APLICAÇÃO

Muitas PoCs são desenvolvidas com forte acoplamento entre o modelo de IA, a lógica de negócio e a infraestrutura subjacente. Essa abordagem dificulta a manutenção, a atualização de modelos e a evolução do sistema. Em produtos maduros, é desejável que modelos possam ser substituídos ou aprimorados sem impactar consumidores externos, o que exige separação clara de responsabilidades e modularidade arquitetural.

## 3.5 AUSÊNCIA DE GOVERNANÇA E CONFIABILIDADE

Aspectos como versionamento de modelos, rastreabilidade de decisões, controle de qualidade e conformidade regulatória raramente são considerados em PoCs. No entanto, em produção, esses fatores são essenciais para garantir confiança, especialmente em ambientes corporativos e regulados. A falta de governança técnica e operacional contribui para que soluções experimentais não alcancem maturidade suficiente para uso contínuo.

## 3.6 DESALINHAMENTO ENTRE TECNOLOGIA E MODELO DE NEGÓCIO

Por fim, muitas PoCs falham por não estarem associadas a um modelo claro de entrega de valor. Demonstrar que um modelo funciona tecnicamente não implica que ele possa ser oferecido como serviço, produto ou plataforma. A ausência de uma estratégia de monetização, distribuição ou consumo frequentemente impede que a iniciativa avance além do estágio experimental.

### 3.6.1 Custo, Escalabilidade, Segurança e Latência como Barreiras à Produtização da IA

A transição de soluções de inteligência artificial de provas de conceito para sistemas prontos para produção impõe um conjunto de requisitos não funcionais que frequentemente não são abordados nas fases iniciais de desenvolvimento. Entre esses requisitos, custo, escalabilidade, segurança e latência destacam-se como fatores críticos que determinam a viabilidade técnica e econômica de uma solução de IA em ambientes corporativos.

## 3.7 CUSTO

Em provas de conceito, o custo computacional tende a ser subestimado ou desconsiderado, uma vez que os experimentos são realizados com volumes reduzidos de dados, baixa concorrência e infraestrutura temporária. No entanto, quando uma solução de IA é exposta como serviço, os custos passam a crescer de forma proporcional ao uso, podendo incluir:

- Execuções concorrentes de modelos de alto custo computacional
- Consumo intensivo de CPU, memória ou GPU
- Dependência de serviços externos cobrados por requisição ou volume de dados

Sem mecanismos adequados de controle, como limitação de requisições, isolamento por cliente e faturamento baseado em consumo, o custo operacional torna-se imprevisível. Essa imprevisibilidade compromete a sustentabilidade financeira da solução e representa uma das principais razões pelas quais PoCs de IA não evoluem para produtos comerciais.

## 3.8 ESCALABILIDADE

A escalabilidade raramente é um requisito explícito em PoCs, que normalmente são executadas de forma sequencial ou com baixa concorrência. Em ambientes de produção, entretanto, sistemas de IA precisam lidar com variações abruptas de demanda, múltiplos usuários simultâneos e cargas de trabalho imprevisíveis.

Arquiteturas tradicionais baseadas em servidores fixos exigem planejamento prévio de capacidade, o que pode resultar tanto em subutilização de recursos quanto em degradação de desempenho sob picos de carga. A ausência de escalabilidade automática dificulta a expansão do serviço e reduz a confiabilidade percebida pelos usuários finais.

Soluções arquiteturais que favorecem escalabilidade horizontal e execução stateless são fundamentais para permitir que agentes de IA respondam dinamicamente à demanda, mantendo desempenho consistente sem intervenção manual.

## 3.9 SEGURANÇA

Em PoCs, práticas de segurança costumam ser mínimas ou inexistentes, já que o foco está na validação técnica do modelo. No entanto, quando uma solução de IA é implantada em produção, surgem preocupações relacionadas a:

- Autenticação e autorização de acesso
- Isolamento entre clientes e dados
- Proteção de dados sensíveis e informações confidenciais
- Conformidade com requisitos regulatórios e corporativos

A ausência de controles de segurança adequados não apenas expõe a organização a riscos técnicos e legais, como também impede a adoção da solução em ambientes empresariais. Sistemas de IA destinados à produção precisam oferecer mecanismos robustos de controle de acesso, auditoria e governança, integrados à arquitetura desde sua concepção.

### 3.10 LATÊNCIA

A latência é outro fator frequentemente negligenciado em ambientes experimentais. Em PoCs, tempos de resposta elevados podem ser aceitáveis, desde que o modelo produza resultados corretos. Em sistemas produtivos, no entanto, a latência impacta diretamente a experiência do usuário e a viabilidade da solução.

Modelos de IA, especialmente aqueles baseados em processamento intensivo ou cadeias de inferência complexas, podem introduzir atrasos significativos quando executados em ambientes não otimizados. A falta de otimização arquitetural, como roteamento eficiente, execução sob demanda e redução de camadas intermediárias, pode tornar a solução impraticável para uso em tempo real ou quase em tempo real.

Arquiteturas que minimizam latência por meio de provisionamento dinâmico, isolamento de funções e integração direta entre API e execução do modelo são essenciais para atender às expectativas de desempenho em produção.

### 3.11 SÍNTESE

Em síntese, PoCs de IA não se tornam produtos não por limitações algorítmicas, mas por lacunas arquiteturais, operacionais e estratégicas. A transformação de uma PoC em produto exige uma mudança de paradigma: da experimentação isolada para a engenharia de sistemas escaláveis, modulares e economicamente sustentáveis. Arquiteturas serverless, design orientado a APIs e execução stateless surgem, nesse contexto, como abordagens capazes de reduzir essa lacuna e acelerar a transição da IA experimental para soluções efetivamente produtivas.

Custo, escalabilidade, segurança e latência não são desafios secundários, mas requisitos estruturais que determinam se uma solução de IA pode ser sustentada como produto. A negligência desses fatores na fase de prova de conceito contribui significativamente para o alto índice de iniciativas de IA que não ultrapassam o estágio experimental. Abordagens arquiteturais que incorporam esses requisitos desde o início são fundamentais para transformar experimentos bem-sucedidos em soluções de IA robustas, confiáveis e economicamente viáveis.

## 4 PRINCÍPIOS ARQUITETURAIS

A arquitetura proposta é orientada pelos seguintes princípios:

## 4.1 DESIGN SERVERLESS-FIRST

Os recursos computacionais são provisionados dinamicamente em resposta à demanda, eliminando infraestrutura ociosa e reduzindo a sobrecarga operacional.

O princípio de *design serverless-first* estabelece que os recursos computacionais devem ser alocados de forma dinâmica e sob demanda, em vez de permanecerem continuamente provisionados. Nesse modelo, a infraestrutura deixa de ser um elemento central de preocupação para o desenvolvimento da aplicação, permitindo que o foco seja direcionado à lógica de negócio e ao comportamento funcional dos agentes de inteligência artificial.

Em arquiteturas tradicionais baseadas em servidores dedicados ou clusters fixos, a necessidade de prever picos de carga frequentemente resulta em infraestrutura ociosa durante períodos de baixa utilização. Esse cenário gera ineficiência econômica e aumenta a complexidade operacional, uma vez que a capacidade deve ser constantemente monitorada, ajustada e mantida. O paradigma serverless elimina essa necessidade ao permitir que o ambiente de execução seja instanciado automaticamente apenas quando há requisições ativas.

No contexto de agentes de IA, o *design serverless-first* apresenta vantagens particularmente relevantes. Cargas de trabalho de IA costumam ser altamente variáveis, com períodos de inatividade intercalados por picos de processamento intensivo. O provisionamento dinâmico garante que os recursos computacionais sejam utilizados exclusivamente durante a execução das inferências ou tarefas de processamento, reduzindo custos e melhorando a eficiência geral do sistema.

Além disso, o modelo serverless reduz significativamente a sobrecarga operacional associada à gestão de infraestrutura. Atividades como configuração de servidores, balanceamento de carga, atualização de sistemas operacionais e gerenciamento de escalabilidade passam a ser abstraídas pela plataforma de nuvem. Essa abstração diminui o risco de falhas operacionais e acelera o ciclo de desenvolvimento e implantação, favorecendo a rápida evolução das soluções.

Outro aspecto fundamental do *design serverless-first* é sua aderência natural à escalabilidade automática. À medida que a demanda cresce, múltiplas instâncias de execução podem ser criadas de forma transparente, sem necessidade de intervenção manual. Essa característica é essencial para agentes de IA expostos como serviços, nos quais o número de requisições pode variar de forma imprevisível em função do comportamento dos usuários ou de integrações externas.

Em síntese, o *design serverless-first* oferece uma base arquitetural que alinha eficiência econômica, simplicidade operacional e escalabilidade. Ao eliminar infraestrutura ociosa e automatizar o provisionamento de recursos, esse paradigma contribui diretamente para a transformação de soluções de IA experimentais em sistemas robustos e sustentáveis em ambientes de produção.

## 4.2 AGENTES DE IA STATELESS

Cada agente de IA opera sem estado persistente, permitindo escalabilidade horizontal e tolerância a falhas.

O princípio de agentes de IA *stateless* estabelece que cada instância de execução opera de forma independente, sem manter estado persistente entre requisições. Nesse modelo, todo o contexto necessário para o processamento deve ser fornecido como entrada ou recuperado de serviços externos, evitando dependências diretas entre execuções consecutivas.

Em arquiteturas tradicionais, o estado da aplicação frequentemente é mantido em memória local ou em estruturas internas do serviço, criando acoplamento entre sessões e dificultando a escalabilidade. Em sistemas de IA expostos como serviços, essa abordagem introduz limitações significativas, pois impede a criação dinâmica de múltiplas instâncias e aumenta a complexidade de recuperação em caso de falhas.

A adoção de agentes de IA *stateless* viabiliza a **escalabilidade horizontal**, uma vez que cada requisição pode ser atendida por qualquer instância disponível, sem a necessidade de sincronização de estado. Esse modelo permite que novas instâncias sejam criadas ou removidas automaticamente conforme a demanda, garantindo desempenho consistente mesmo sob cargas variáveis.

Além disso, a ausência de estado persistente contribui diretamente para a **tolerância a falhas**. Caso uma instância de execução seja interrompida ou falhe durante o processamento, outra instância pode assumir a requisição sem impacto significativo para o sistema ou para o usuário final. Essa característica é particularmente relevante em ambientes serverless, nos quais instâncias são efêmeras por natureza.

No contexto de agentes de IA, o design *stateless* também facilita a manutenção e a evolução do sistema. Modelos de IA podem ser atualizados, substituídos ou versionados sem a necessidade de migração de estado interno, reduzindo riscos durante o processo de implantação. Dados persistentes, quando necessários, podem ser armazenados em serviços externos especializados, como bancos de dados ou sistemas de armazenamento de objetos, preservando a separação de responsabilidades.

Por fim, agentes de IA *stateless* promovem maior previsibilidade operacional. Ao eliminar dependências implícitas entre execuções, o comportamento do sistema torna-se mais determinístico, facilitando testes, monitoramento e auditoria. Essa previsibilidade é essencial para transformar soluções experimentais em serviços confiáveis e adequados para ambientes corporativos.

## 4.3 EXECUÇÃO CONTAINERIZADA

Os agentes de IA são empacotados como imagens de contêiner, garantindo consistência entre ambientes e simplificando o gerenciamento de dependências.

A execução containerizada consiste no empacotamento dos agentes de inteligência artificial como imagens de contêiner, contendo todo o conjunto de dependências, bibliotecas, modelos e configurações necessárias para sua execução. Esse modelo promove a padronização do ambiente de execução, assegurando que o comportamento do agente seja consistente entre os ambientes de desenvolvimento, teste e produção.

Um dos principais desafios na transição de PoCs de IA para produção está relacionado à inconsistência entre ambientes. Diferenças de versões de bibliotecas, dependências nativas e configurações do sistema operacional frequentemente resultam em falhas difíceis de reproduzir e diagnosticar. A containerização mitiga esse problema ao encapsular completamente o ambiente de execução, reduzindo a variabilidade entre diferentes estágios do ciclo de vida da aplicação.

Além de garantir consistência, a execução containerizada simplifica significativamente o gerenciamento de dependências. Agentes de IA costumam depender de bibliotecas especializadas, frameworks de aprendizado de máquina e componentes de sistema que, quando instalados diretamente no ambiente de execução, aumentam a complexidade operacional. Ao consolidar essas dependências em uma imagem de contêiner versionada, torna-se possível controlar com precisão as mudanças no ambiente e facilitar processos de atualização e rollback.

No contexto de arquiteturas serverless, a containerização amplia a flexibilidade do modelo de execução. Ao permitir que funções serverless sejam baseadas em imagens de contêiner, elimina-se a necessidade de adaptar o código a ambientes de execução predefinidos e restritivos. Isso é particularmente relevante para agentes de IA que exigem dependências específicas ou configurações customizadas, tornando a arquitetura mais adequada para workloads complexos.

Outro benefício relevante da execução containerizada é a independência tecnológica. Ao desacoplar o agente de IA da infraestrutura subjacente, o mesmo contêiner pode ser executado em diferentes plataformas compatíveis, favorecendo portabilidade e reduzindo riscos de dependência excessiva de um único provedor. Essa característica contribui para a sustentabilidade da solução a longo prazo.

Em síntese, a execução containerizada oferece uma base sólida para a operacionalização de agentes de IA em ambientes produtivos. Ao garantir consistência entre ambientes, simplificar o gerenciamento de dependências e ampliar a flexibilidade de execução, esse paradigma fortalece a confiabilidade e a manutenibilidade de arquiteturas orientadas à IA.

## 4.4 APIS COMO PRODUTOS

Cada capacidade de IA é exposta por meio de uma interface de API padronizada, permitindo implantação independente, versionamento e monetização.

O princípio de *APIs como produtos* estabelece que cada capacidade de inteligência artificial deve ser concebida, projetada e disponibilizada como um serviço independente, acessível por meio de uma interface de programação padronizada. Nesse modelo, a API deixa de ser apenas um meio técnico de integração e passa a representar a unidade fundamental de entrega de valor, consumo e evolução do sistema.

Em provas de conceito, funcionalidades de IA costumam estar fortemente acopladas a aplicações específicas, scripts ou fluxos internos, dificultando sua reutilização e expansão. Ao tratar cada capacidade de IA como um produto independente, a arquitetura promove desacoplamento entre consumidores e provedores, permitindo que diferentes aplicações utilizem os mesmos serviços sem dependências diretas de implementação.

A padronização das interfaces de API desempenha papel central nesse contexto. Contratos bem definidos de entrada e saída, formatos de dados consistentes e comportamentos previsíveis facilitam a integração com múltiplos clientes, sejam eles aplicações web, móveis ou sistemas corporativos. Além disso, APIs padronizadas reduzem a curva de adoção e ampliam a escalabilidade organizacional da solução.

Outro aspecto fundamental do modelo de APIs como produtos é a **implantação independente**. Cada agente de IA pode ser atualizado, substituído ou escalado sem impactar outros componentes do sistema, desde que o contrato da API seja preservado. Essa independência reduz riscos durante o ciclo de desenvolvimento e favorece a evolução contínua da plataforma.

O **versionamento** é igualmente essencial para garantir a estabilidade do ecossistema. Ao disponibilizar múltiplas versões de uma API, torna-se possível introduzir melhorias ou alterações sem interromper consumidores existentes. Essa abordagem é particularmente relevante em sistemas de IA, nos quais modelos e algoritmos evoluem rapidamente e podem exigir ajustes frequentes.

Além dos benefícios técnicos, o modelo de APIs como produtos viabiliza estratégias claras de **monetização e governança**. A exposição das capacidades de IA por meio de APIs permite o controle preciso de uso, a aplicação de políticas de acesso e a cobrança baseada em consumo. Essa característica transforma funcionalidades de IA em ativos econômicos mensuráveis, alinhando a arquitetura técnica aos objetivos de negócio.

Em síntese, ao tratar APIs como produtos, a arquitetura transcende a experimentação isolada e estabelece uma base sólida para a entrega sustentável de serviços de IA. Essa abordagem promove modularidade, escalabilidade, previsibilidade e viabilidade econômica, elementos essenciais para a consolidação de soluções de IA em ambientes de produção.

## 5 ARQUITETURA DO SISTEMA

A arquitetura proposta foi concebida para suportar a implantação, a operação e a evolução contínua de agentes de inteligência artificial como serviços independentes e reutilizáveis. Seu desenho prioriza modularidade, escalabilidade e simplicidade operacional, permitindo que cada componente evolua de forma autônoma sem comprometer o funcionamento global do sistema.

A arquitetura é composta pelos seguintes componentes:

### 5.1 REGISTRO DE CONTÊINERES

O registro de contêineres desempenha um papel fundamental na arquitetura proposta, atuando como o repositório central de imagens versionadas que encapsulam os agentes de inteligência artificial. Cada imagem de contêiner contém não apenas o código-fonte do agente, mas também todas as dependências de software, bibliotecas especializadas, modelos de IA e configurações necessárias para sua execução consistente.

A utilização de imagens versionadas permite um controle preciso sobre o ciclo de vida dos agentes de IA. Cada atualização — seja de código, de modelo ou de dependências — pode ser registrada como uma nova versão, possibilitando rastreabilidade completa das mudanças ao longo do tempo. Esse controle é essencial em ambientes produtivos, nos quais auditoria, conformidade e capacidade de reprodução de comportamentos anteriores são requisitos críticos.

Outro benefício relevante do uso de um registro de contêineres é a viabilização de mecanismos seguros e eficientes de *rollback*. Em caso de falhas ou degradação de desempenho após uma atualização, versões anteriores das imagens podem ser rapidamente reimplantadas, reduzindo o tempo de indisponibilidade e os riscos operacionais. Essa capacidade é particularmente importante em sistemas de IA, nos quais alterações em modelos podem produzir efeitos inesperados.

Além disso, o registro de contêineres garante a reprodutibilidade dos ambientes de execução. Ao utilizar a mesma imagem de contêiner em diferentes estágios — desenvolvimento, testes e produção — elimina-se a variabilidade causada por diferenças de configuração ou dependências. Essa consistência reduz falhas difíceis de diagnosticar e contribui para a estabilidade do sistema como um todo.

Por fim, o registro de contêineres atua como o ponto central de distribuição dos agentes de IA, permitindo que a camada de computação serverless obtenha imagens confiáveis e previamente validadas. Essa centralização fortalece a governança do ecossistema, assegurando que apenas versões autorizadas e verificadas dos agentes sejam implantadas, ao mesmo tempo em que simplifica a gestão e a escalabilidade da plataforma.

## 5.2 CAMADA DE COMPUTAÇÃO SERVERLESS

A camada de computação serverless é responsável pela execução dos agentes de inteligência artificial containerizados em resposta direta às requisições encaminhadas pela camada de API. Nesse modelo, cada solicitação recebida resulta na criação de uma instância de execução efêmera, isolada e independente, que processa a tarefa solicitada e é descartada ao final da execução.

A principal característica dessa abordagem é a eliminação do provisionamento manual de infraestrutura. Diferentemente de arquiteturas baseadas em servidores persistentes, nas quais é necessário planejar capacidade, gerenciar recursos e realizar manutenção contínua, o paradigma serverless delega essas responsabilidades à plataforma de nuvem. Isso reduz significativamente a complexidade operacional e permite que equipes concentrem esforços no desenvolvimento e na evolução dos agentes de IA.

A criação dinâmica de instâncias de execução conforme a demanda viabiliza escalabilidade horizontal automática. À medida que o volume de requisições aumenta, múltiplas instâncias podem ser inicializadas simultaneamente para atender à carga, sem necessidade de configuração prévia. Da mesma forma, em períodos de baixa demanda, os recursos são automaticamente liberados, evitando desperdício computacional.

A natureza efêmera das execuções reforça o modelo *stateless* adotado pelos agentes de IA. Como não há garantia de reutilização da mesma instância entre requisições, todo o processamento é realizado de forma independente, sem dependência de estado local. Essa característica contribui diretamente para a tolerância a falhas, uma vez que a interrupção ou falha de uma instância não compromete o funcionamento global do sistema, podendo ser imediatamente compensada por outra execução.

Do ponto de vista econômico, a computação serverless introduz um modelo de cobrança baseado no tempo de execução e nos recursos efetivamente consumidos. Esse modelo alinha custos ao uso real do serviço, reduzindo investimentos iniciais e tornando a solução financeiramente viável mesmo para cargas de trabalho intermitentes, típicas de aplicações de IA expostas como serviços.

Em conjunto, essas características fazem da camada de computação serverless um elemento central da arquitetura proposta, oferecendo escalabilidade, resiliência e eficiência econômica. Ao abstrair a complexidade da infraestrutura e favorecer um modelo de execução dinâmico e desacoplado, essa camada contribui de forma decisiva para a transformação de agentes de IA experimentais em serviços robustos e prontos para produção.

## 5.3 CAMADA DE API GATEWAY

A camada de API Gateway atua como o ponto de entrada único para os serviços de inteligência artificial, sendo responsável por expor os agentes de IA como endpoints HTTP padronizados e por

intermediar a comunicação entre aplicações clientes e a camada de computação serverless baseada em funções AWS Lambda. Essa integração direta estabelece um fluxo eficiente no qual cada requisição recebida pelo gateway resulta na invocação controlada de uma função Lambda correspondente ao agente solicitado.

No contexto da arquitetura proposta, dois modelos de integração do API Gateway foram avaliados: **HTTP API** e **REST API**. Cada abordagem apresenta características distintas, que impactam diretamente a governança, a flexibilidade e o grau de controle sobre o tráfego e o comportamento das requisições.

Durante as fases iniciais de implantação, a **HTTP API** foi adotada como mecanismo de exposição dos serviços devido à sua configuração simplificada, menor latência e menor custo operacional. Essa abordagem mostrou-se adequada para acelerar o *time-to-market* e validar rapidamente o consumo dos agentes de IA. Entretanto, a simplicidade da HTTP API impõe limitações relevantes em cenários que demandam maior controle sobre requisições e respostas.

À medida que a arquitetura evoluiu e requisitos mais avançados foram incorporados, tornou-se necessária a adoção da **REST API**. Diferentemente da HTTP API, a REST API oferece suporte completo a funcionalidades essenciais de governança, incluindo:

- Configuração detalhada e consistente de **CORS (Cross-Origin Resource Sharing)**
- Manipulação e transformação de payloads de requisição e resposta
- Suporte robusto a cargas binárias (como imagens, PDFs e arquivos CSV)
- Integrações do tipo *Lambda Proxy* e *Non-Proxy*
- Padronização rigorosa de headers e códigos de resposta

Em particular, a **habilitação completa e confiável de CORS**, fundamental para o consumo dos serviços de IA por aplicações frontend modernas, só foi plenamente alcançada por meio da REST API. Esse nível de controle mostrou-se indispensável para garantir interoperabilidade segura entre aplicações web e os agentes de IA executados no Lambda.

Além disso, a REST API possibilitou a implementação mais refinada de mecanismos de autenticação, autorização e limitação de requisições (*throttling*), reforçando a proteção da camada de computação serverless e assegurando previsibilidade de custos e desempenho. Ao centralizar essas responsabilidades no API Gateway, o código dos agentes de IA permaneceu desacoplado de preocupações relacionadas à segurança e ao controle de tráfego.

Em síntese, a experiência prática evidenciou que a escolha entre HTTP API e REST API não deve ser tratada como uma decisão binária, mas como uma questão de maturidade arquitetural e requisitos funcionais. Enquanto a HTTP API atende de forma eficaz cenários iniciais e de baixa complexidade, a REST API se mostrou essencial para suportar requisitos avançados de governança,

segurança e integração, consolidando a arquitetura proposta como adequada para ambientes de produção.

Tabela 1

| Aspecto | HTTP API | REST API |
|---|---|---|
| Objetivo principal | Exposição simplificada de APIs | Exposição completa e altamente configurável de APIs |
| Complexidade de configuração | Baixa | Alta |
| Custo por requisição | Menor | Maior |
| Latência | Menor | Maior (devido a camadas adicionais) |
| Integração com AWS Lambda | Direta e simplificada | Direta, com suporte a proxy e não-proxy |
| Suporte a CORS | Básico e limitado | Completo e altamente configurável |
| Controle de headers | Limitado | Granular e detalhado |
| Transformação de payload | Não suportada | Suportada (request/response mapping) |
| Suporte a payloads binários | Nativo | Requer configuração explícita (Binary Media Types) |
| Versionamento de APIs | Limitado | Suportado de forma nativa |
| Governança e controle | Básico | Avançado |
| Limitação de requisições (throttling) | Básica | Avançada e configurável |
| Adequação para MVP | Alta | Média |
| Adequação para produção corporativa | Média | Alta |
| Curva de aprendizado | Baixa | Alta |
| Casos de uso recomendados | MVPs, validações rápidas | Sistemas maduros, ambientes produtivos |

Fonte: Autores.

## 5.4 APLICAÇÕES FRONTEND E CLIENTES

As aplicações frontend e os clientes externos constituem a camada de consumo da arquitetura proposta, sendo responsáveis por interagir com os agentes de inteligência artificial exclusivamente por meio das APIs expostas pela camada de API Gateway. Nesse modelo, os consumidores não possuem conhecimento direto sobre a infraestrutura subjacente, os mecanismos de execução serverless ou a implementação interna dos agentes de IA, o que promove um forte desacoplamento entre consumo e execução.

Esse desacoplamento arquitetural favorece a interoperabilidade com diferentes plataformas e tecnologias. Aplicações web, móveis e sistemas corporativos podem consumir os serviços de IA de maneira uniforme, independentemente de sua linguagem de programação, ambiente de execução ou arquitetura interna. Como resultado, as capacidades de IA tornam-se reutilizáveis em múltiplos contextos, ampliando significativamente seu potencial de adoção.

A padronização das interfaces de consumo desempenha um papel central nesse processo. APIs com contratos bem definidos, formatos de dados consistentes e comportamentos previsíveis reduzem a complexidade de integração e minimizam erros de comunicação entre clientes e serviços. Essa previsibilidade é particularmente importante em ambientes corporativos, nos quais múltiplas equipes ou sistemas podem consumir os mesmos serviços de IA simultaneamente.

Além disso, a utilização de APIs padronizadas facilita a evolução das aplicações clientes de forma independente. Novas versões de agentes de IA podem ser implantadas sem exigir alterações imediatas nos consumidores, desde que os contratos das APIs sejam preservados. Essa independência reduz riscos durante a evolução do sistema e contribui para a estabilidade do ecossistema como um todo.

Do ponto de vista operacional, o modelo de consumo via APIs também favorece o controle e a governança. Métricas de uso, limites de requisição e políticas de acesso podem ser aplicados de forma consistente a todos os clientes, independentemente da plataforma de origem. Essa uniformidade simplifica o monitoramento e permite uma gestão mais eficiente do consumo dos serviços de IA.

Em síntese, a camada de aplicações frontend e clientes desempenha um papel essencial na arquitetura ao viabilizar o consumo amplo, seguro e padronizado das capacidades de IA. O desacoplamento entre consumidores e agentes de IA, aliado à padronização das interfaces, contribui diretamente para a escalabilidade, a interoperabilidade e a sustentabilidade da solução em ambientes produtivos.

## 5.5 FATURAMENTO E CONTROLE DE ACESSO

O componente de faturamento e controle de acesso desempenha um papel estratégico na arquitetura proposta, ao estabelecer políticas claras de uso, governança e monetização dos serviços de inteligência artificial. Ao tratar cada agente de IA como um serviço consumível, torna-se essencial identificar consumidores, controlar o volume de requisições e associar o uso efetivo a modelos de cobrança sustentáveis.

A utilização de chaves de API como mecanismo de identificação permite distinguir consumidores individuais ou aplicações clientes de forma consistente. Cada requisição pode ser associada a uma chave específica, possibilitando a aplicação de políticas diferenciadas de acesso, limites de uso e níveis de serviço. Esse modelo favorece tanto cenários internos, nos quais diferentes áreas consomem os serviços, quanto contextos externos, nos quais clientes distintos acessam a plataforma.

O controle de requisições, por meio de limitação (*throttling*) e cotas de consumo, contribui diretamente para a estabilidade operacional do sistema. Ao impor limites por consumidor, evita-se o uso excessivo ou indevido dos recursos computacionais, protegendo a camada de computação serverless contra sobrecarga e assegurando previsibilidade de desempenho e custos.

Do ponto de vista econômico, o registro detalhado de métricas de consumo viabiliza modelos de cobrança baseados em uso real. Esse modelo de faturamento alinha custos operacionais ao valor efetivamente entregue, reduzindo barreiras de adoção e tornando a plataforma financeiramente viável mesmo para cargas de trabalho variáveis. A monetização por consumo também favorece a

escalabilidade do negócio, à medida que o crescimento do uso se traduz diretamente em aumento de receita.

Além da monetização, o controle de acesso contribui de forma significativa para a segurança da arquitetura. A autenticação baseada em chaves de API, associada a políticas de autorização, reduz o risco de acessos não autorizados e facilita a auditoria das interações com os serviços de IA. Esse nível de controle é particularmente relevante em ambientes corporativos, nos quais requisitos de conformidade e rastreabilidade são fundamentais.

Essa estrutura de faturamento e controle de acesso, integrada às demais camadas da arquitetura, reforça a modularidade do sistema. Cada agente de IA pode evoluir de forma independente, com políticas de uso e monetização ajustadas conforme suas características, sem comprometer o comportamento operacional consistente da plataforma como um todo. Dessa forma, a arquitetura proposta estabelece uma base sólida para a oferta sustentável de serviços de IA em ambientes produtivos.

## 5.6 SÍNTESE DA ARQUITETURA

A arquitetura apresentada neste trabalho foi projetada para viabilizar a implantação de agentes de inteligência artificial como serviços modulares, escaláveis e economicamente sustentáveis, superando as limitações frequentemente observadas em provas de conceito que não evoluem para ambientes de produção. Seu desenho adota princípios *serverless-first*, execução *stateless*, containerização e exposição das capacidades de IA por meio de APIs tratadas como produtos.

No núcleo da arquitetura, os agentes de IA são empacotados como imagens de contêiner versionadas e armazenados em um registro centralizado, garantindo consistência de execução, rastreabilidade de mudanças e reprodutibilidade entre diferentes estágios do ciclo de vida da aplicação. Essas imagens são consumidas pela camada de computação serverless, que executa os agentes de forma efêmera e sob demanda, eliminando a necessidade de provisionamento manual de infraestrutura e permitindo escalabilidade horizontal automática.

A camada de API Gateway atua como intermediária entre os consumidores e a camada de execução baseada em AWS Lambda, expondo os agentes como endpoints HTTP padronizados. A diferenciação entre HTTP API e REST API permite equilibrar simplicidade inicial e controle avançado, sendo a REST API essencial para a implementação completa de funcionalidades de governança, como configuração detalhada de CORS, controle de headers, transformação de payloads e suporte robusto a cargas binárias. Ao centralizar preocupações relacionadas a segurança, autenticação, autorização e limitação de requisições, o API Gateway desacopla essas responsabilidades do código dos agentes de IA, simplificando seu desenvolvimento e manutenção.

As aplicações frontend e os clientes externos consomem os serviços de IA exclusivamente por meio das APIs expostas, sem dependência direta da infraestrutura ou da implementação interna dos agentes. Esse desacoplamento promove interoperabilidade entre diferentes plataformas e tecnologias, reduz a complexidade de integração e amplia o alcance das capacidades de IA oferecidas pela plataforma.

Complementarmente, o componente de faturamento e controle de acesso estabelece políticas claras de uso, governança e monetização, utilizando chaves de API para identificar consumidores, aplicar limites de requisição e registrar métricas de consumo. Esse modelo viabiliza a cobrança baseada em uso real, contribui para a segurança do sistema e assegura previsibilidade operacional e financeira.

Em conjunto, esses componentes formam uma arquitetura modular que permite a evolução independente de cada agente de IA, mantendo um comportamento operacional consistente em todo o sistema. Ao alinhar decisões arquiteturais a requisitos não funcionais críticos — como custo, escalabilidade, segurança e latência — a arquitetura proposta oferece uma base sólida para a transformação de soluções de IA experimentais em serviços prontos para produção, escaláveis e sustentáveis em ambientes corporativos.

## 6 REST API VS HTTP API: LIÇÕES APRENDIDAS EM PRODUÇÃO

Durante as fases iniciais de implantação da arquitetura proposta, optou-se pela adoção de uma integração baseada em **HTTP API**, com o objetivo de acelerar o *time-to-market* e viabilizar a validação rápida do consumo dos agentes de inteligência artificial. Essa escolha mostrou-se adequada em um contexto de baixa complexidade operacional, oferecendo menor esforço de configuração, redução de latência e integração simplificada com a camada de computação serverless baseada em AWS Lambda.

A HTTP API apresentou vantagens importantes nesse estágio inicial, como a facilidade de exposição de endpoints, o suporte nativo a chamadas *cross-origin* e o manuseio simplificado de uploads de arquivos. Essas características permitiram que os agentes de IA fossem rapidamente disponibilizados para aplicações clientes, favorecendo ciclos curtos de desenvolvimento e feedback.

Entretanto, à medida que os requisitos do sistema evoluíram e a arquitetura passou a atender cenários mais complexos, tornou-se evidente a necessidade de maior controle sobre o comportamento das requisições e respostas. Nesse contexto, a arquitetura foi revisitada para incorporar integrações baseadas em **REST API**, que oferecem maior flexibilidade e capacidades avançadas de governança.

As REST APIs permitem controle mais granular sobre diversos aspectos críticos da comunicação entre clientes e agentes de IA, incluindo:

- Tratamento detalhado de cargas binárias, como imagens, documentos PDF e arquivos CSV
- Configuração completa e consistente de políticas de *Cross-Origin Resource Sharing* (CORS)
- Mapeamento e transformação de payloads de requisição e resposta

- Suporte a integrações do tipo *Lambda Proxy* e *Non-Proxy*

Essas capacidades mostraram-se essenciais para suportar aplicações frontend mais sofisticadas, integração com sistemas corporativos e padronização do comportamento dos serviços em ambientes produtivos. Em particular, a habilitação completa e confiável de CORS revelou-se um requisito fundamental para a interoperabilidade com aplicações web modernas, sendo plenamente atendida apenas por meio da REST API.

A adoção da REST API, contudo, introduziu uma complexidade adicional de configuração e exigiu maior maturidade arquitetural. Aspectos como definição explícita de *binary media types*, escolha adequada entre integrações proxy e não proxy, e padronização de headers e códigos de resposta demandaram um entendimento mais profundo do funcionamento do API Gateway e de sua integração com o AWS Lambda.

Por meio de um processo iterativo de refinamento e aprendizado, as REST APIs foram configuradas com sucesso para uso em produção. A padronização das integrações proxy, o tratamento adequado de cargas binárias e a imposição de esquemas de resposta consistentes resultaram em uma arquitetura estável, previsível e alinhada aos requisitos operacionais do sistema.

Essa experiência evidencia que a REST API não representa uma limitação técnica, mas sim uma solução mais poderosa que requer maior grau de maturidade arquitetural e expertise operacional. Os resultados obtidos reforçam que a escolha entre HTTP API e REST API deve ser orientada pelo estágio de maturidade do sistema e pelos requisitos funcionais e não funcionais, e não por uma avaliação simplista de superioridade tecnológica.

## 7 CASOS DE APLICAÇÃO

A arquitetura proposta foi validada por meio de sua aplicação em múltiplos domínios de inteligência artificial, abrangendo diferentes tipos de dados, padrões de consumo e requisitos computacionais. Em todos os casos, os agentes de IA foram implantados como serviços independentes, expostos por meio de APIs padronizadas e executados em uma infraestrutura serverless, demonstrando a flexibilidade e a reutilização do modelo arquitetural.

### 7.1 VISÃO COMPUTACIONAL

No domínio de visão computacional, a arquitetura foi utilizada para a classificação de imagens e a detecção de danos em objetos, cenários que demandam processamento intensivo e suporte a cargas binárias. Os agentes de IA responsáveis por essas tarefas recebem imagens como entrada, executam inferências baseadas em modelos de aprendizado profundo e retornam resultados estruturados por meio da API.

A execução serverless mostrou-se adequada para esse tipo de carga de trabalho, permitindo que os recursos computacionais fossem alocados apenas durante o processamento das imagens. A integração com REST API possibilitou o tratamento adequado de *binary media types* e a padronização das respostas, assegurando interoperabilidade com aplicações frontend e sistemas corporativos.

## 7.2 PROCESSAMENTO DE LINGUAGEM NATURAL

A arquitetura também foi aplicada a agentes de processamento de linguagem natural, incluindo análise de sentimentos, sumarização de documentos e análise automatizada de currículos. Esses agentes operam sobre dados textuais, frequentemente enviados em formatos estruturados ou como arquivos, e produzem saídas que apoiam processos decisórios em ambientes corporativos.

A modularidade da arquitetura permitiu que cada capacidade de NLP fosse implementada como um agente independente, facilitando atualização de modelos, versionamento de APIs e controle de uso por tipo de serviço. Além disso, a execução *stateless* garantiu previsibilidade de comportamento e escalabilidade em cenários de uso concorrente.

## 7.3 AUTOMAÇÃO DE DADOS

No contexto de automação de dados, a arquitetura foi empregada para a geração de consultas SQL a partir de prompts em linguagem natural. Nesse caso, os agentes de IA atuam como intermediários inteligentes entre usuários finais e sistemas de banco de dados, traduzindo intenções expressas em linguagem natural em comandos estruturados.

A exposição desse tipo de funcionalidade por meio de APIs padronizadas permitiu sua integração com diferentes aplicações, sem dependência direta de um banco de dados específico. Essa abordagem favoreceu a reutilização do agente em múltiplos contextos e reforçou o desacoplamento entre lógica de IA e sistemas de persistência.

## 7.4 GERAÇÃO DE MÍDIA

A arquitetura também foi aplicada à geração automatizada de mídia, especificamente na criação de podcasts a partir de documentos textuais. Nesse cenário, os agentes de IA executam pipelines que envolvem processamento de texto, geração de roteiro e síntese de voz, produzindo arquivos de áudio como saída.

A execução containerizada mostrou-se essencial para suportar as dependências específicas desse tipo de pipeline, enquanto a computação serverless permitiu lidar com tempos de execução variáveis e picos de demanda. A exposição do serviço por meio de APIs facilitou sua integração com aplicações externas e possibilitou a monetização baseada em consumo.

### 7.5 SÍNTESE DOS CASOS DE APLICAÇÃO

Em todos os domínios analisados, a implantação dos agentes como serviços independentes evidenciou a capacidade da arquitetura de acomodar diferentes cargas de trabalho, formatos de dados e padrões de uso. A reutilização dos mesmos princípios arquiteturais — serverless, execução *stateless*, containerização e APIs como produtos — reforça a generalidade e a robustez da abordagem proposta, validando sua aplicabilidade em cenários reais de produção.

## 8 DISCUSSÃO

Os resultados apresentados ao longo deste trabalho evidenciam que a principal barreira para a transformação de soluções de inteligência artificial em produtos prontos para produção não reside na sofisticação dos modelos, mas nas decisões arquiteturais e operacionais que sustentam sua execução em ambientes reais. A arquitetura proposta demonstrou ser eficaz ao abordar, de forma integrada, requisitos não funcionais críticos como custo, escalabilidade, segurança e latência.

Um dos principais achados deste estudo é a relevância do paradigma *serverless-first* para workloads de IA caracterizados por demanda intermitente e alta variabilidade de uso. A computação serverless mostrou-se particularmente adequada para mitigar custos operacionais e eliminar infraestrutura ociosa, ao mesmo tempo em que fornece escalabilidade automática e tolerância a falhas. No entanto, esse modelo também impõe restrições, como limites de tempo de execução e dependência de inicializações dinâmicas, que precisam ser consideradas no desenho dos agentes de IA.

A adoção de agentes de IA *stateless* revelou-se um fator determinante para a escalabilidade horizontal e a previsibilidade operacional da plataforma. Ao eliminar dependências de estado entre execuções, tornou-se possível tratar cada requisição de forma independente, favorecendo a resiliência do sistema e simplificando processos de manutenção e atualização. Esse modelo, entretanto, exige uma separação clara entre lógica de processamento e persistência de dados, o que demanda maior disciplina arquitetural.

A execução containerizada destacou-se como um mecanismo essencial para garantir consistência entre ambientes e reduzir falhas decorrentes de incompatibilidades de dependências. Ao encapsular código, modelos e bibliotecas em imagens versionadas, a arquitetura promoveu reprodutibilidade e controle rigoroso do ciclo de vida dos agentes de IA. Por outro lado, a adoção de contêineres em ambientes serverless requer atenção ao tamanho das imagens e ao impacto potencial sobre tempos de inicialização.

A análise comparativa entre HTTP API e REST API revelou que a escolha do modelo de integração deve ser orientada pelo estágio de maturidade do sistema e pelos requisitos funcionais e de governança. Enquanto a HTTP API mostrou-se adequada para fases iniciais, oferecendo simplicidade e menor latência, a REST API tornou-se indispensável para atender requisitos avançados, como

configuração completa de CORS, manipulação de cargas binárias e controle refinado de requisições e respostas. Essa experiência reforça que decisões arquiteturais eficazes são contextuais e evolutivas, e não absolutas.

Outro aspecto relevante discutido neste trabalho é o papel das APIs como produtos. Ao expor cada capacidade de IA como um serviço independente, a arquitetura possibilitou implantação autônoma, versionamento controlado e monetização baseada em uso. Esse modelo não apenas favorece a reutilização técnica, como também alinha a arquitetura aos objetivos de negócio, transformando funcionalidades de IA em ativos mensuráveis e governáveis.

Por fim, os casos de aplicação apresentados demonstram a generalidade da arquitetura proposta, que se mostrou capaz de acomodar diferentes domínios de IA — visão computacional, processamento de linguagem natural, automação de dados e geração de mídia — sem alterações estruturais significativas. Essa versatilidade sugere que a abordagem pode ser replicada em outros contextos e setores, desde que os princípios arquiteturais fundamentais sejam respeitados.

Em síntese, a discussão reforça que a produtização da IA exige uma mudança de foco, da experimentação isolada para a engenharia de sistemas robustos, escaláveis e economicamente sustentáveis. A arquitetura apresentada oferece um caminho viável para essa transição, ao integrar práticas consolidadas de engenharia de software e computação em nuvem às demandas específicas de sistemas de inteligência artificial em produção.

## 9 CONCLUSÃO

Este trabalho apresentou uma arquitetura serverless para a implantação de agentes de inteligência artificial como serviços modulares, escaláveis e economicamente sustentáveis, abordando desafios recorrentes que impedem a evolução de provas de conceito para sistemas prontos para produção. Ao longo do artigo, demonstrou-se que as principais limitações enfrentadas por iniciativas de IA em ambientes corporativos não estão relacionadas à capacidade dos modelos, mas à ausência de uma base arquitetural adequada para sua operacionalização contínua.

A arquitetura proposta fundamenta-se em princípios como *serverless-first*, execução *stateless*, containerização e exposição das capacidades de IA por meio de APIs tratadas como produtos. Esses princípios permitiram alinhar requisitos não funcionais críticos — custo, escalabilidade, segurança e latência — às demandas práticas de ambientes produtivos, reduzindo a complexidade operacional e aumentando a confiabilidade do sistema.

A análise comparativa entre HTTP API e REST API evidenciou que decisões arquiteturais devem ser orientadas pelo estágio de maturidade do sistema e pelos requisitos de governança. Enquanto abordagens mais simples se mostraram adequadas para fases iniciais, a REST API revelou-se essencial para suportar funcionalidades avançadas, como configuração completa de CORS,

manipulação de cargas binárias e controle refinado de requisições, consolidando-se como elemento-chave para ambientes de produção.

Os casos de aplicação apresentados demonstraram a versatilidade da arquitetura ao acomodar diferentes domínios de inteligência artificial, incluindo visão computacional, processamento de linguagem natural, automação de dados e geração de mídia, sem necessidade de alterações estruturais significativas. Essa generalidade reforça a aplicabilidade da abordagem proposta em diversos contextos organizacionais e setoriais.

Em síntese, este trabalho contribui ao demonstrar que a produtização da inteligência artificial requer uma mudança de paradigma: da experimentação isolada para a engenharia de sistemas orientados a serviços, escaláveis e governáveis. A arquitetura apresentada oferece um caminho prático e replicável para essa transição, servindo como referência para organizações que buscam transformar soluções de IA em produtos robustos, reutilizáveis e sustentáveis em ambientes de produção.

## 9.1 CONSIDERAÇÕES FINAIS

Os experimentos e validações funcionais descritos neste trabalho foram conduzidos por meio de uma interface interativa de testes, que atua como cliente genérico das APIs, simulando diferentes cenários de consumo dos agentes de IA.

A implementação de referência utilizada nos experimentos está disponível como material suplementar, com acesso restrito às funcionalidades de demonstração.[1]

[1] https://agents-marketplace.i4uai.com/

## REFERÊNCIAS


Amershi, S. et al. Software Engineering for Machine Learning: A Case Study. IEEE/ACM International Conference on Software Engineering (ICSE), 2019.

Polyzotis, N. et al. Data Management Challenges in Production Machine Learning. SIGMOD Record, 2018.

Amazon Web Services (AWS). Serverless Architectures with AWS Lambda. AWS Whitepaper, 2020.

Merkel, D. Docker: Lightweight Linux Containers for Consistent Development and Deployment. Linux Journal, 2014.

Amazon Web Services (AWS). Best Practices for Container Images. AWS Documentation.

Amazon Web Services (AWS). Amazon API Gateway Developer Guide. AWS Documentation.

W3C. Cross-Origin Resource Sharing (CORS). W3C Recommendation, 2014.

Amazon Web Services (AWS). API Gateway Usage Plans and API Keys. AWS Documentation.