Inovação e Tecnologia

Agentes – Multiagentes

Inteligência Artificial

Instituto J&F Germinare TECH

ASSESSOR

Até agora a gente tem um agente que, a partir de seu entendimento do ambiente apresentado a ele (banco de dados + entrada de usuário) consegue tomar uma decisão, ou gerar uma resposta ou armazenar algo no banco de dados, ou consultar algo no banco de dado, ou atualizar algo no banco de dados.

Mas por enquanto ele consegue interagir com perguntas sobre finanças, certo?

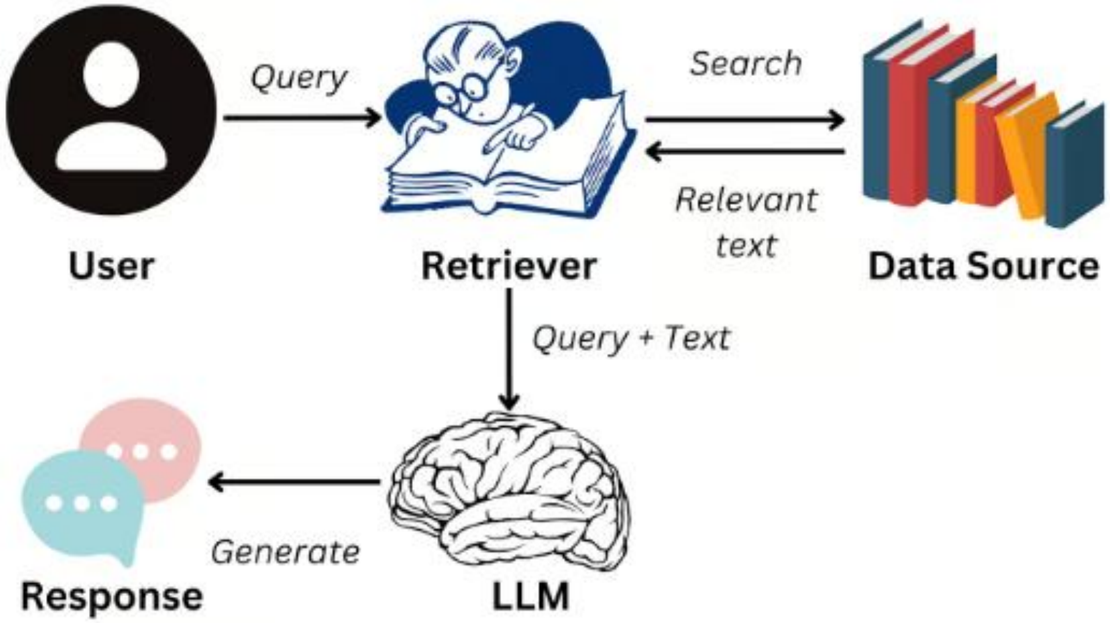
Se fizer uma pergunta sobre se tem algum compromisso hoje? Se disser que tem uma consulta próxima semana?

RAG

RAG vs Fine-Tuning

RAG: Tempo real (aumenta o conhecimento e a possibilidade resposta).

Fine-tuning: tem histórico.



AGENTE ÚNICO

Quando criamos nosso prompt, falamos que esse sistema não responderia apenas informações sobre finanças, mas também sobre agenda/compromisso. No system prompt fizemos tarefas abordando as duas situações.

Até podemos continuar fazendo como fizemos nas aulas anteriores, criando novas tools e deixando esse agente decidir. Isso seria um sistema de agente único.

Em um **sistema de agente único**, toda a responsabilidade de lidar com finanças, agendamentos, e qualquer outra tarefa, recai sobre um único agente de IA. Ele precisa ser um "faz-tudo", com acesso a todas as ferramentas e bases de conhecimento para gerenciar todas as requisições.

AGENTE ÚNICO

Problemas com um Agente Único:

• **Dificuldade de Escalabilidade:** Adicionar novas funcionalidades, como a parte de agendamento, pode sobrecarregar o agente único. Se ele já está otimizado para finanças, adicionar a agenda pode confundir suas respostas e comprometer a performance.

• **Complexidade e Manutenção:** O código de um agente único se torna extremamente complexo e ruim de editar, se so a resposta de finanças precisa ser refinada, como fazer sem alterar os compromissos? A quantidade de lógica, ferramentas e prompts necessários para lidar com todas as tarefas cresce exponencialmente, tornando a manutenção e a depuração muito mais difíceis.

• **Conflito de Prioridades:** O agente pode ter dificuldade em priorizar tarefas. Por exemplo, se uma pergunta sobre finanças e um pedido de agendamento chegam ao mesmo tempo, ele precisa de uma lógica complexa para decidir qual processar primeiro.

MULTIAGENTES

Já em um **sistema multiagente**, as tarefas são divididas entre vários agentes especializados. Cada agente tem um papel específico e um conjunto de ferramentas para cumprir suas responsabilidades. Por exemplo, um agente lida apenas com as finanças, outro com a agenda, e um terceiro pode atuar como um roteador de tarefas.

MULTIAGENTES

Clareza e Especialização: Cada agente é um especialista. O **agente financeiro** entende perfeitamente de transações e saldos, enquanto o **agente de agenda** se concentra em horários e compromissos. Essa especialização garante respostas mais precisas e eficientes.

• **Modularidade e Manutenção:** O sistema é modular. Podemos desenvolver, testar e fazer a manutenção de cada agente de forma independente, sem afetar os outros. Se houver um problema no agente de finanças, ele não impacta o agente de agenda.

• **Escalabilidade:** É muito mais fácil adicionar novas funcionalidades. Queremos adicionar um agente para compras ou para lembretes de saúde? Basta criar um novo agente com suas próprias ferramentas e integrá-lo ao sistema.

• **Paralelismo:** Múltiplas tarefas podem ser processadas simultaneamente. Enquanto o agente de finanças está buscando o saldo, o agente de agenda pode estar verificando um horário livre, otimizando o tempo de resposta geral.

AGENTES

Para nosso sistema, podemos propor a criação de três agentes principais:

1.Agente Roteador (ou Agente de Orquestração): Este agente é o "gerente de tráfego". Sua função é receber a requisição inicial do usuário e decidir para qual agente ela deve ser enviada. Por exemplo, se a pergunta é sobre "saldo da conta", ele direciona para o agente de finanças. Se for sobre "criar um compromisso", ele envia para o agente de agenda.

2.Agente de Finanças: Este é o agente que vocês já desenvolveram. Ele é o especialista em lidar com todas as tarefas financeiras, como:

1. Responder perguntas sobre o saldo e transações.
2. Armazenar e recuperar dados financeiros do banco de dados.
3. Realizar cálculos e análises financeiras.

3.Agente de Agenda: Este será outro novo agente. Ele será o especialista em lidar com compromissos e horários, com as seguintes responsabilidades:

1. Criar, ler, atualizar e excluir compromissos na agenda.
2. Verificar a disponibilidade de horários.
3. Enviar lembretes de compromissos.

MULTIAGENTES

Quem fala com o usuário? Há dois padrões válidos:

“Especialista fala direto com o usuário”

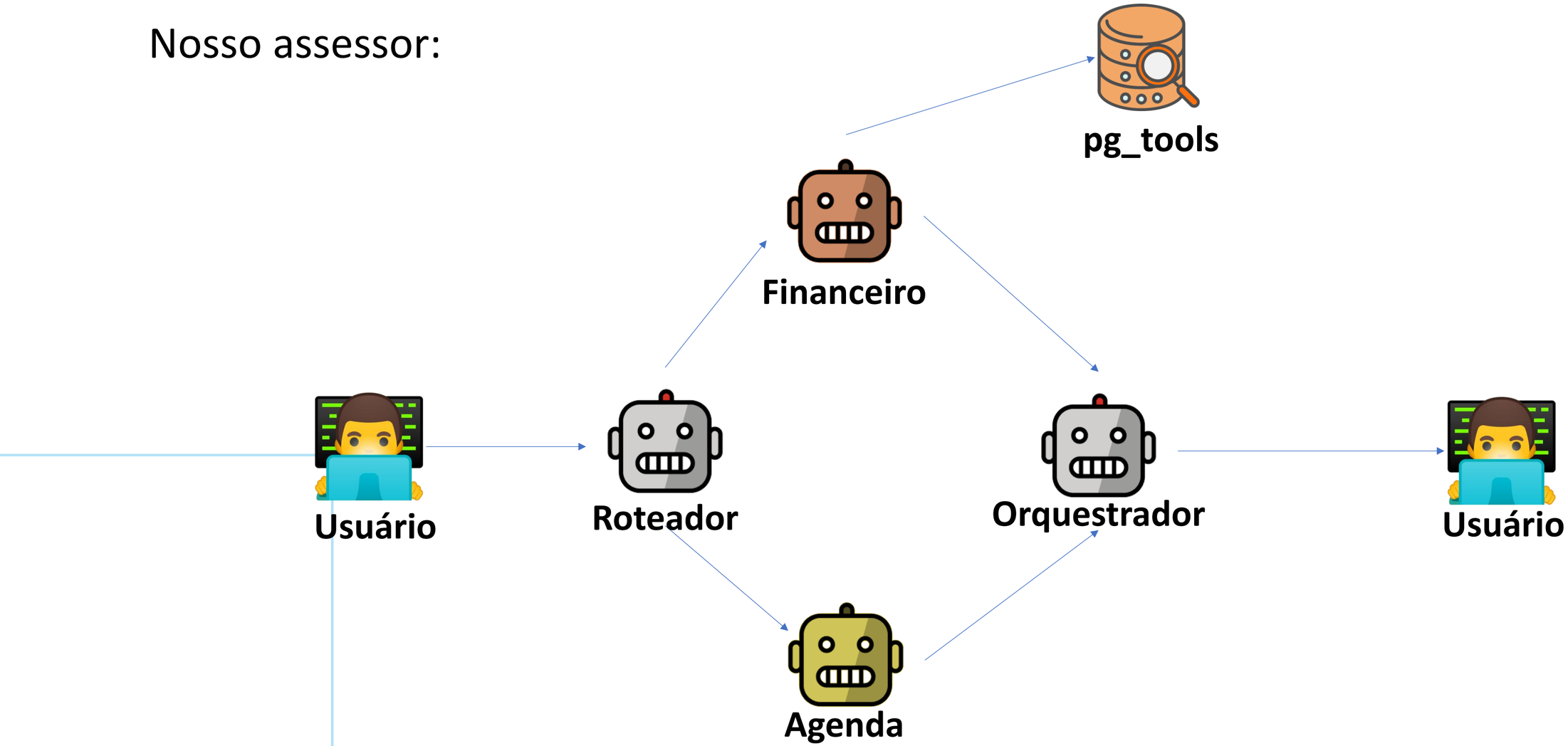
- Orquestrador classifica → chama especialista → retorna a resposta do especialista ao usuário.
- Vantagens: Menos latência, menos camadas, menos chance de “reformatar” errado.
- OBS: mais indicado quando a resposta final vem de um único especialista.

“Orquestrador como narrador”

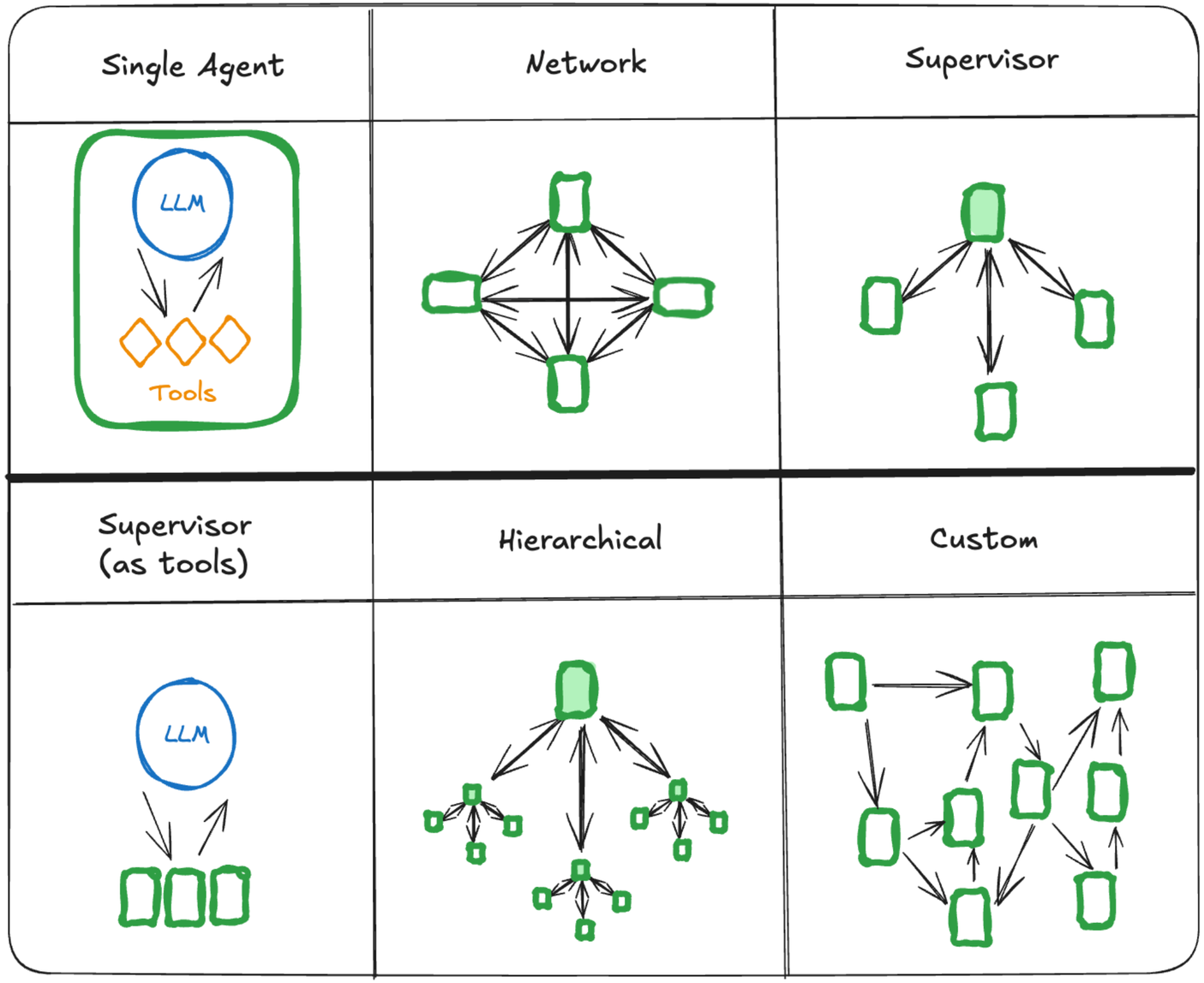
- Orquestrador classifica → chama especialista → pós-processa (padroniza formato, insere Acompanhamento, checa conflitos) → envia ao usuário.
- Vantagens: Camada de governança (consistência, red teaming, políticas, cortes de tokens), agrega saídas de vários especialistas antes de responder.
- OBS: Mais latência/complexidade.

MULTIAGENTES

Nosso assessor:



Arquitetura SMA



Arquitetura

Como iremos começar a adicionar camadas, vamos pensar em arquitetura. Sim, um sistema de agentes de IA é um sistema e precisa ser estruturado, na arquitetura, podemos apresentar o sistema de 3 formas:

Arquitetura de camadas (sequencial):

- É a forma mais comum de visualizarmos um sistema, pensando em como os dados e as requisições fluem de uma ponta a outra. É uma visão sequencial e hierárquica. Para começar o projeto.

Arquitetura de componentes (técnicos):

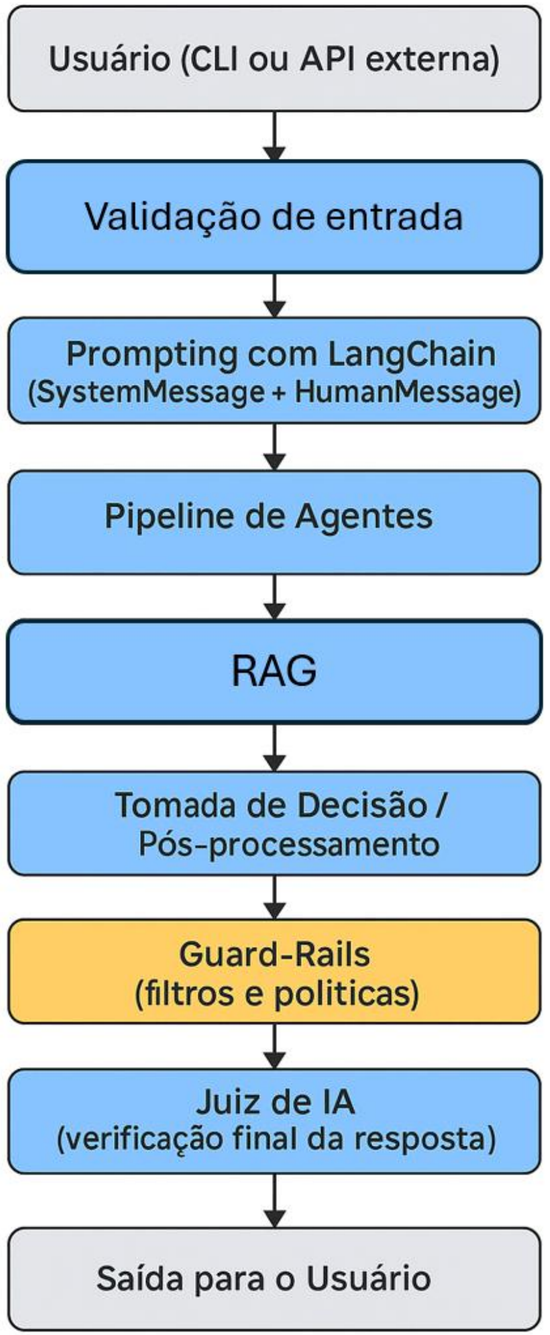
- É uma visão mais técnica. Ela foca nas partes ou módulos independentes que compõem o sistema. Bancos, ferramentas, APIs, etc. Para a parte técnica apenas.

Arquitetura de interação (alto nível):

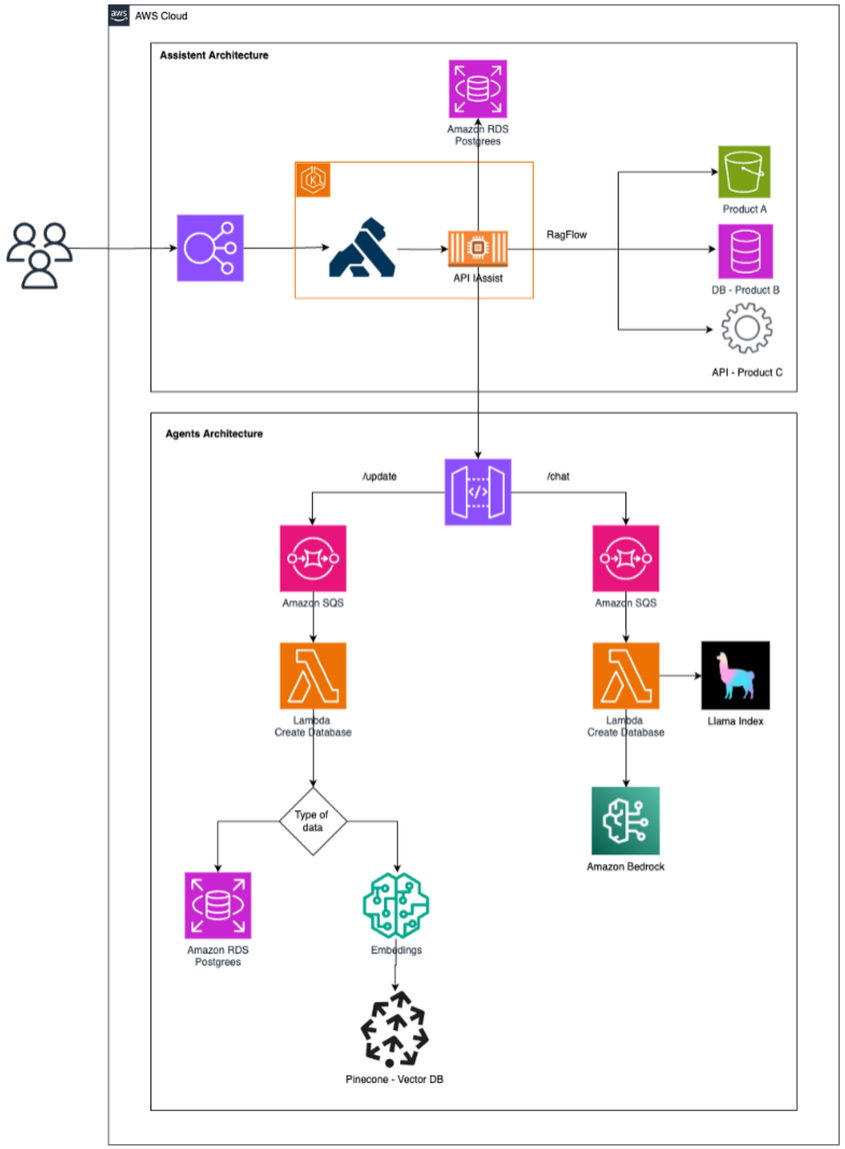
- É uma visão **conceitual** e de **alto nível**. Ela se concentra em como os agentes se comunicam entre si e com o ambiente, sem se aprofundar nos detalhes técnicos. É como um mapa (ou fluxograma) que mostra as relações entre os diferentes "personagens" do sistema. Para o cliente.

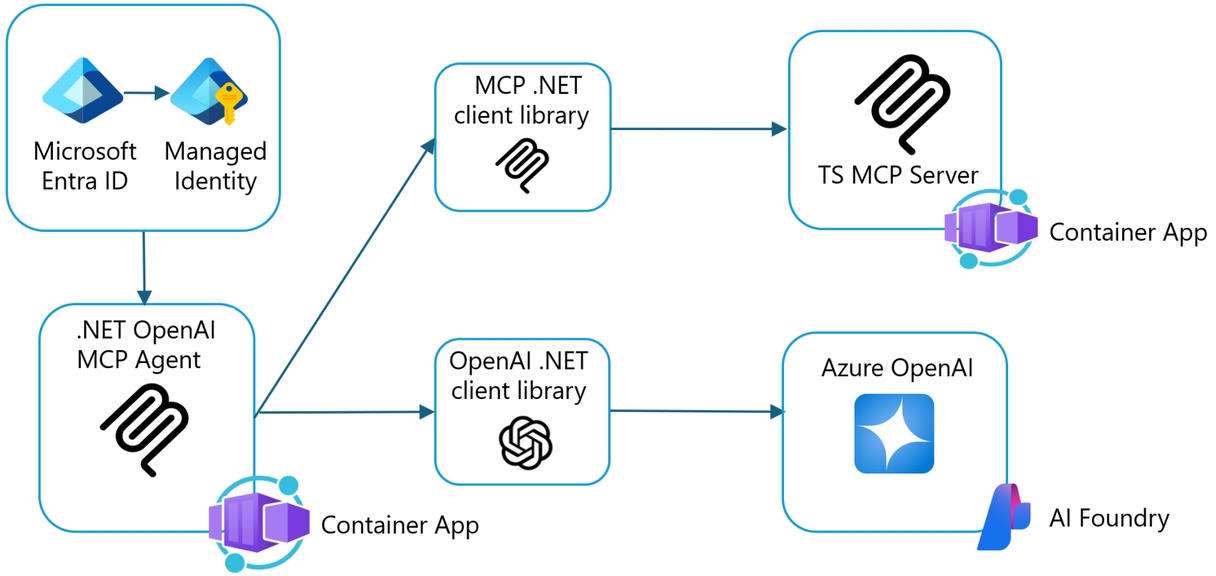
Arquitetura de camadas

- **Validação de entrada**: Recebe a requisição, sanitiza o texto e verifica formato/autorização.
- **Prompting Estruturado (LangChain)**: Definimos o papel do modelo e o conteúdo dinâmico enviado pelo usuário.
- **Pipeline de Agentes**: Encadeamento de agentes autônomos, cada um especializado em tarefas (ex: consulta, geração, validação).
- **RAG**: Busca de documentos relacionados à entrada, enriquecendo a geração.
- **Decisão e Pós-processamento**: Integra as respostas dos agentes e formata a saída.
- **Guard-Rails**: Restrições de segurança, escopo e ética.
- **Juiz de IA**: Valida a resposta antes de apresentá-la ao usuário.



Arquitetura de componentes





Arquitetura de alto nível