Como criar um copiloto de IA sem pagar por API

A arquitetura em dois processos usada pelo copiloto deste site — sem chamada cobrada de API de IA.

Arquitetura	Dois serviços (site + ponte isolada)
Autenticação da IA	Login pessoal do assistente, sem cobrança por chamada
Isolamento	Usuário de sistema dedicado + sandbox só-leitura
Nível	Avançado
Tutorial online	iiot.esp32.tec.br/tutoriais/copiloto-de-ia-sem-api-paga-arquitetura

Este manual descreve, passo a passo, a arquitetura que roda o copiloto de chat visível no canto deste site — um assistente de IA autenticado por login (sem custo por chamada), isolado do processo principal por dois serviços separados. Cobre a decisão de segurança que forçou essa separação, o usuário de sistema dedicado que não tem acesso a nada além do necessário, um bug real de sandbox que quase fez o assistente inventar respostas, e onde essa arquitetura deixa de ser a escolha certa.

1. Por que duas camadas, não uma

O caminho mais simples para um chat de IA num site é uma única chamada: o visitante pergunta, o servidor repassa para uma API de IA na nuvem, a resposta volta. Esse caminho não coube aqui pelo motivo errado de sempre citado — custo — mas por um motivo de segurança mais concreto.

O processo do site roda com um conjunto de restrições de sistema (systemd hardening) pensado para conter qualquer coisa exposta à internet pública. Uma dessas restrições impede o processo de trocar de usuário do sistema operacional — e trocar de usuário é exatamente o que um assistente de IA por login precisa fazer para rodar isolado do resto do servidor.

A tentação é afrouxar essa restrição só para o assistente funcionar. É o tipo de atalho que resolve o sintoma e cria o próximo incidente de segurança. A solução adotada foi outra: tirar essa responsabilidade do processo do site inteiramente.

O resultado são dois processos, falando só por uma chamada interna autenticada:

```
Visitante -> Site (hardened, sem trocar de usuario)
                 |  chamada HTTP interna + token
                 v
         Ponte (processo proprio, sem essa restricao)
                 |  sudo -u usuario_dedicado
                 v
     Assistente de IA (sandbox so-leitura)
```

2. A ponte isolada

A ponte é um serviço systemd próprio, ouvindo só em loopback (127.0.0.1), autenticado por um token de cabeçalho próprio — quem não tiver o token não fala com ela, mesmo estando na mesma máquina. É ela, não o site, que executa o comando que troca de usuário:

```
const child = spawn("sudo", [
  "-u", USUARIO_DEDICADO,
  COMANDO_ASSISTENTE, "exec", "--sandbox", "read-only", prompt,
]);
```

O código completo (com cache e verificação de token) está em **src/bridge-server.js**, no kit que acompanha este manual.

Do lado do site, a chamada é um fetch comum — nenhuma linha ali precisa de privilégio nenhum:

```
await fetch(BRIDGE_URL, {
  method: "POST",
  headers: { "X-Bridge-Token": BRIDGE_TOKEN },
  body: JSON.stringify({ prompt, question, cache }),
});
```

3. O usuário dedicado e o que ele não pode fazer

A peça que faz a separação valer a pena é o usuário de sistema criado só para rodar o assistente: sem shell interativo, sem permissão de login, guardando apenas a sessão autenticada da ferramenta de IA.

O que esse usuário NÃO tem acesso:

• Segredos do site (chaves de API, credenciais de banco, tokens de outros serviços).

• Escrita fora do próprio diretório de trabalho.

• Permissão de login interativo ou execução de shell.

• Qualquer arquivo de outro usuário do sistema, incluindo o do próprio site.

É o mesmo princípio de qualquer sandbox: não é impedir que algo dê errado ali dentro, é garantir que dar errado ali dentro não signifique dar errado em nenhum outro lugar. Se o assistente for instruído — por engano ou por uma pergunta maliciosa de um visitante — a fazer algo indevido, o alcance máximo dele é o próprio espaço isolado.

4. Cache, limite de uso e o bug que quase inventava resposta

Sem controle de uso, todo copiloto de IA vira um jeito caro de responder a mesma pergunta várias vezes. Duas camadas resolvem isso: um limite de perguntas por visitante numa janela de tempo, e um cache de respostas chaveado pela pergunta normalizada (minúscula, sem acento, sem pontuação). Medido em produção: a primeira vez que uma pergunta é feita leva a maior parte de vinte segundos; a mesma pergunta, cacheada, volta em uma fração de segundo.

O bug do sandbox

O assistente foi instruído a consultar arquivos internos antes de responder — em vez de responder "de memória". O isolamento de processo por baixo do sandbox de execução, porém, precisa de um tipo específico de soquete de rede só para montar sua própria interface de loopback interna. A restrição de rede da ponte não incluía esse tipo de soquete, e todo comando do assistente passou a falhar silenciosamente.

O efeito colateral foi o mais perigoso possível: em vez de avisar que não conseguiu ler o arquivo, o modelo às vezes preenchia a lacuna com uma resposta plausível e errada. Fluente e convincente, e falsa. A correção foi permitir explicitamente esse tipo de soquete na lista de restrições — sem abrir rede nenhuma de verdade.

A lição não é sobre esse soquete específico: é que um sandbox bem-feito ainda pode falhar de um jeito silencioso, e a única defesa real é instruir o assistente a admitir incerteza em vez de supor que ele sempre vai avisar sozinho.

5. Quando essa arquitetura não é a certa

Nenhuma escolha de arquitetura serve para tudo. Vale o contraponto antes de copiar este desenho:

Login pessoal não é para produto comercial de alto tráfego.

A autenticação por login foi pensada para uso individual, não para servir milhares de visitantes simultâneos atrás de um único login. Passado certo volume, o caminho correto volta a ser uma API paga, com contrato e limite adequados ao uso comercial.

Isolar processo dá trabalho de operação.

Dois serviços systemd, um usuário de sistema dedicado, um token para sincronizar entre eles — é mais peça para manter no ar do que uma chamada HTTP direta a uma API. Para um projeto pessoal ou um site de conteúdo, o custo evitado compensa. Para uma equipe pequena sem tempo de operação, pode não compensar.

Sandbox reduz risco, não elimina.

O bug do soquete de rede é a prova: um isolamento bem-feito ainda pode falhar de um jeito que não estava no plano. Combine isolamento com instrução clara ao modelo para admitir incerteza, e teste isso na prática antes de confiar.

6. Checklist de implantação

[ ]	Usuário de sistema dedicado criado, sem shell e sem login interativo
[ ]	Ponte rodando como serviço systemd próprio, sem NoNewPrivileges
[ ]	Token de autenticação gerado, sincronizado nas duas pontas, fora do código-fonte
[ ]	Ponte ouvindo só em loopback (127.0.0.1), nunca exposta direto à internet
[ ]	Limite de perguntas por visitante configurado e testado
[ ]	Cache de respostas funcionando (medir 1ª chamada vs. chamada repetida)
[ ]	Assistente instruído a admitir quando não conseguiu confirmar algo
[ ]	Teste de sandbox: comando do assistente falhando é percebido, não silenciado

Kit de código de referência (bridge-server.js, site-client.ts, unidade systemd) disponível junto deste manual. Artigo completo em iiot.esp32.tec.br/tutoriais/copiloto-de-ia-sem-api-paga-arquitetura.