VMind

Passo a passo completo

Claude no *WhatsApp*

O guia real de como ligar os dois

Claude no WhatsApp: o guia real de como ligar os dois

O que você vai montar

No fim desse guia você vai ter a lógica completa (e um esqueleto de código funcional) de uma automação que recebe mensagem de um grupo de WhatsApp, filtra o que interessa, entrega pro Claude responder, e devolve a resposta pro mesmo grupo, tudo rodando sozinho, 24 horas por dia, sem você precisar apertar nenhum botão.

É a mesma lógica, generalizada, do que roda por trás do Marabo (o agente que atende no seu WhatsApp).

Antes de começar

Você (ou quem for montar isso pra você) precisa de:

- Um servidor ou VPS ligado 24h (pode ser bem simples, 1-2GB de RAM já basta pro início).
- Node.js instalado no servidor (versão 18 ou mais nova).
- Uma instância do **n8n** (pode ser self-hosted no mesmo servidor, via Docker, ou a versão paga na nuvem).
- Acesso à API do WhatsApp. Duas rotas comuns no Brasil:
- **WhatsApp Business Cloud API** (oficial, da Meta): mais estável, mas exige processo de verificação de negócio.
- **Evolution API** (não oficial, open source): mais rápida de configurar, mas roda por fora da política oficial da Meta, então tem risco de bloqueio se usar de forma abusiva.
- Uma chave de API da Anthropic (para chamar o Claude) ou o Claude Code instalado no servidor, se preferir rodar via CLI.
- Conhecimento básico de linha de comando Linux (copiar arquivo, editar texto, rodar comando).

1. O gatilho: o grupo de WhatsApp

1.1 Escolha o número e o grupo

Separe um número de WhatsApp só pra isso (não use seu número pessoal). Crie o grupo onde as mensagens vão chegar, ou defina que o robô responde em conversa direta.

1.2 Conecte esse número à API escolhida

Se for Cloud API oficial: você cadastra o número no Meta for Developers, gera um token de acesso e um `phone_number_id`. Se for Evolution API: você sobe o serviço (geralmente via Docker) e escaneia o QR Code com o WhatsApp do número escolhido, como se fosse o WhatsApp Web.

Como conferir: mande uma mensagem de teste pro número. Se a API que você conectou tiver um endpoint de "mensagens recebidas" (webhook), você deve ver essa mensagem chegando nos logs dela.

2. O filtro: o fluxo no n8n

2.1 Crie um Webhook node

No n8n, crie um novo workflow e adicione um node do tipo **Webhook**. Ele vai gerar uma URL (algo como `https://seu-n8n.com/webhook/whatsapp-in`). É essa URL que você configura na API do WhatsApp como destino das mensagens recebidas.

2.2 Adicione um filtro

Depois do Webhook, adicione um node **IF** (ou **Filter**) que verifica se o texto da mensagem começa com a palavra-gatilho escolhida (no nosso caso, "marabo"). Exemplo de expressão pra comparar:

```
{{ $json.message.text.toLowerCase().startsWith("marabo") }}
```

2.3 Limpe o texto antes de repassar

Ainda no n8n, adicione um node **Set** (ou **Function**) que remove a palavra-gatilho do início da mensagem, deixando só o conteúdo real:

```
return [{ json: { texto: $json.message.text.replace(/^marabo[,:]?\s*/i, '') } }];
```

2.4 Chame o seu bridge

No fim do fluxo, adicione um node **HTTP Request** apontando pro endereço do seu bridge (o script da seção 3), enviando o texto limpo, o número de quem mandou, e o ID do grupo.

Como conferir: mande "marabo, teste" no grupo. No histórico de execuções do n8n (aba Executions), você deve ver o workflow rodando e chegando até o node HTTP Request sem erro.

3. A ponte: o script bridge

3.1 O que esse script precisa fazer

Ele é um servidor HTTP simples (pode ser em Node.js com Express) que:

1. Recebe o POST vindo do n8n.
2. Monta o prompt pro Claude (o texto da mensagem, mais qualquer contexto que você queira injetar, tipo data/hora).
3. Chama a API do Claude (ou aciona o Claude Code via linha de comando).
4. Pega a resposta e manda de volta pro grupo, chamando de novo a API do WhatsApp (ou outro webhook do n8n dedicado a enviar mensagem).

3.2 Esqueleto do código

Um exemplo mínimo, só pra entender a estrutura (você vai precisar adaptar com sua chave de API e o SDK oficial da Anthropic):

```
import express from 'express';
import Anthropic from '@anthropic-ai/sdk';

const app = express();
app.use(express.json());
const anthropic = new Anthropic({ apiKey: process.env.ANTHROPIC_API_KEY });

app.post('/bridge', async (req, res) => {
  const { texto, grupoId } = req.body;

  const resposta = await anthropic.messages.create({
    model: 'claude-sonnet-5',
    max_tokens: 1024,
    messages: [{ role: 'user', content: texto }]
  });

  // aqui você chama a função que manda a resposta de volta pro WhatsApp
  await enviarParaWhatsapp(grupoId, resposta.content[0].text);

  res.json({ ok: true });
});

app.listen(8790, () => console.log('Bridge rodando na porta 8790'));
```

3.3 Onde roda

Esse script fica no mesmo servidor do n8n (ou em outro, desde que o n8n consiga alcançar ele pela rede). Ele escuta numa porta fixa (no exemplo, 8790).

Como conferir: rode o script manualmente (`node bridge.js`) e use um `curl` pra simular o n8n chamando ele:

```
curl -X POST http://localhost:8790/bridge -H "Content-Type: application/json" -
d '{"texto":"oi, teste","grupoId":"123"}'
```

Se voltar `{"ok":true}` e a mensagem chegar no grupo, a ponte está funcionando.

4. Rodando 24h: o serviço que nunca desliga

4.1 Por que isso importa

Se você só rodar `node bridge.js` no terminal, ele para assim que você fechar a conexão SSH ou o servidor reiniciar. Isso não serve pra produção.

4.2 Criando um serviço systemd

No Linux, crie um arquivo em `/etc/systemd/system/meu-bridge.service`:

```
[Unit]
Description=Bridge WhatsApp -> Claude
After=network-online.target

[Service]
Type=simple
User=seu_usuario
ExecStart=/usr/bin/node /caminho/para/bridge.js
Restart=always
RestartSec=5

[Install]
WantedBy=multi-user.target
```

4.3 Ativando o serviço

Rode, em sequência:

```
sudo systemctl daemon-reload
sudo systemctl enable meu-bridge.service
sudo systemctl start meu-bridge.service
```

Como conferir: rode `sudo systemctl status meu-bridge.service`. Deve aparecer "active (running)" em verde. Se cair por qualquer motivo, o `Restart=always` faz ele voltar sozinho em 5 segundos.

Erros comuns

Erro 1: o webhook do n8n nunca recebe nada. Causa: a URL do webhook não está acessível de fora (firewall bloqueando, ou n8n rodando só em `localhost`). Correção: confirme que a porta do n8n está liberada no firewall e que a URL configurada na API do WhatsApp é a pública, não a interna.

Erro 2: o bridge recebe a mensagem mas nunca responde no grupo. Causa: geralmente é erro na chamada de volta pra API do WhatsApp (token expirado, formato de número errado). Correção: teste a função de enviar mensagem isoladamente, sem depender do fluxo inteiro, pra isolar o problema.

Erro 3: toda mensagem do grupo aciona o Claude, até as que não eram pra você. Causa: esqueceu o filtro da palavra-gatilho (seção 2.2). Correção: sem esse filtro, qualquer post do grupo vira uma chamada de API paga e uma resposta indesejada. O filtro não é opcional.

Erro 4: o serviço cai e não volta sozinho. Causa: faltou o `Restart=always` no arquivo do systemd, ou o script trava em vez de dar erro e encerrar. Correção: sempre configure a política de restart, e adicione tratamento de erro no script pra ele não travar em silêncio.

Erro 5: usar Evolution API (não oficial) pensando que é tão seguro quanto a oficial. Causa: confundir "funciona igual" com "tem o mesmo risco". Correção: a API não oficial pode ter o número banido pela Meta se detectar uso automatizado fora do padrão dela. Pra uso comercial sério, o caminho mais seguro no médio prazo é migrar pra Cloud API oficial.

Próximo nível

Depois que a ponte básica estiver de pé, o próximo passo é dar memória à automação: hoje, no exemplo acima, cada mensagem é tratada isolada, sem lembrar da conversa anterior. Adicionar um banco de dados simples (ou um arquivo por conversa) pra guardar o histórico recente é o que transforma um "responde e esquece" numa automação que realmente parece estar acompanhando a conversa.