Instalação AstraCalls padrão

Link: https://github.com/AstraOnlineWeb/AstraCalls/tree/main

git clone https://github.com/AstraOnlineWeb/AstraCalls.git

□□ AstraCalls

Chamadas de voz e vídeo do WhatsApp em Go puro, direto do navegador — prontas para produção SaaS.

VoIP nativo (áudio + vídeo, 1:1 e em grupo), gateway **SIP/PBX**, suíte completa de **mensagens**, **disparos**, **webhooks** confiáveis, integração com **Chatwoot** e deploy em **Docker Swarm + Traefik**.

Go React whatsmeow pion PostgreSQL Docker n8n License

Visão Geral · Funcionalidades · Arquitetura · Início Rápido · Configuração · Deploy · API · n8n · Suporte

---

□□ Visão Geral

> O AstraCalls pareia uma ou mais contas do WhatsApp (QR code, **código de pareamento** ou **passkey**) e permite **fazer e receber chamadas de voz e vídeo** de qualquer navegador, ligar para **tronco/ramal SIP**, e operar a conta como canal de **mensagens** (texto, mídia, botões, listas, carrossel, enquete, formulário, pix, produto, status…).
>
> O microfone e a câmera do navegador chegam ao servidor Go por **WebRTC** (ou por **WebSocket**, com fallback automático quando o WebRTC não passa no firewall). O servidor transcodifica para os codecs do WhatsApp (**MLow** para voz, **H.264** para vídeo) e injeta a mídia na malha de **relay SRTP** do WhatsApp — e o caminho inverso traz o áudio e o vídeo do outro lado de volta ao navegador.

> Toda a pilha VoIP roda **nativamente em Go**: codec MLow, RTP/SRTP/SRTCP, STUN, transporte SCTP/DTLS do relay, pipeline H.264 e a sinalização `<call>`, integrados ao **whatsmeow** e servidos a um painel **React 19**. A única dependência em C é o codec `opus_mlow` (via cgo) — e é opcional: sem ela o servidor roda em modo **somente sinalização**.
>
> Várias contas podem ser pareadas e operadas lado a lado, cada uma com QR, status, histórico e banco próprios. Uma conta mantém **várias chamadas 1:1 simultâneas** (uma por atendente), roteadas por call ID.

> **Status:** estável e em produção (última versão estável no CHANGELOG). Chamada em **grupo** é funcional (áudio + vídeo, 3+ participantes) mas segue marcada como experimental, atrás da flag `WACALLS_GROUP_CALLS`.

□□ Funcionalidades

☎ Chamadas de voz

- **Saída e entrada 1:1** com áudio bidirecional, várias chamadas simultâneas por conta.
- **Espera (hold/resume) com música**, **atender em espera** e **transferência cega** entre atendentes (troca de dono + ponte, sem tocar a perna do WhatsApp).
- **Mudo / mão levantada** (`mute_v2`, `raise_hand`) nos dois sentidos.
- **Timeout de toque**, recusa que para o telefone de quem ligou, consent freshness (RFC 7675) — a chamada não cai sozinha aos ~20s.
- **Toque fantasma** (`POST /calls/fake`): liga e desliga antes de atender (aviso/alerta).
- **Transporte de áudio no navegador:** WebRTC com **fallback automático para WebSocket** (`WACALLS_DEFAULT_TRANSPORT=auto|webrtc|websocket`); WS de áudio **PCM16 full-duplex** (`GET /calls/{id}/ws`) pronto para agentes de voz/IA.
- **Gravação opt-in por sessão**: MP3 mono mixado dos dois lados, vira nota privada no Chatwoot e evento `recording` no webhook.

□□ Chamadas de vídeo

- **Vídeo 1:1** (câmera do atendente e do cliente), **upgrade/downgrade** no meio da chamada (áudio → vídeo e volta), religar a câmera, **MediaClock** (áudio e vídeo no mesmo relógio).
- **Orientação automática** (lê a rotação do celular no RTP e no stanza `<video>`) com **botão de girar** como ajuste manual, no painel e no widget.
- **Vídeo no widget do Chatwoot** (WebCodecs, data channel H.264).
- **Disparo de ligações de VÍDEO** com arquivo gravado (`POST /broadcast {video:true}`).

□□ Chamada em grupo *(experimental, `WACALLS_GROUP_CALLS=1`)*

- Liga **para um grupo existente** (resolve os membros) com **áudio e vídeo**; o atendente ouve o **mix** de todos e fala para todos; vê a **câmera de cada participante** em tiles rotulados com número + nome.
- Protocolo completo do grupo: offer com `group_info`, chave de **epoch compartilhada com rotação** a cada entrada, relay de grupo (DTLS direto, SFU com HBH-FEC), **SRTCP** (Sender Reports + **PLI**/pedido de keyframe nos dois sentidos), anúncio `<video state>`, participante que sai some do painel.
- Endpoints `POST /calls/group`, `/calls/group/end`, WS `/calls/group/ws` (áudio PCM16) e `/calls/group/video-ws` (H.264), `GET /calls/group/roster`.

☎ Gateway SIP / PBX

- **Servidor SIP** (porta 5060 UDP/TCP) para o PBX/softphone do cliente **se registrar** no AstraCalls (credenciais por sessão, mostradas no painel) — tronco ↔ WhatsApp nos dois sentidos, áudio G.711 ↔ MLow.
- **Registro em PBX externo** (o AstraCalls vira um ramal do Asterisk/FreePBX/tronco hospedado), com **servidor, domínio e outbound proxy** separados, re-REGISTER automático e status no painel.
- RTP em **faixa fixa de portas** (`WACALLS_RTP_PORT_MIN/MAX`), publicável em `mode=host`.

□□ Conector SIP (PABX IP → AstraChat) — serviço separado

- Binário `sipconnector` na mesma imagem: liga **qualquer PABX IP** ao AstraChat, sem WhatsApp.
- Linha por caixa (o PABX registra no conector **ou** o conector registra no PABX) e **um ramal por atendente** (e-mail → ramal).

- Expõe a **mesma API de chamadas do wacalls** (SSE, accept/reject/DELETE, `/webrtc` para o navegador em PCMU sem transcodificar, `/ws` pcm16 para o agente de voz por IA) e transferência para ramal por **SIP REFER**.
- Detalhes, variáveis e deploy: `cmd/sipconnector/README.md`.

□□ Mensagens (API estilo WAHA, ~60 endpoints)

- **Envio:** texto, imagem, áudio/nota de voz (com waveform), vídeo, **nota de vídeo (PTV)**, documento, sticker, contato, localização, link com preview, **enquete** (e voto), **evento** (e resposta), **pix (BR Code)**, **produto**, **status** (texto/imagem/vídeo/áudio).
- **Interativas que renderizam no número não-oficial:** `interactive` (CTA: abrir URL, copiar código, ligar, resposta rápida), `buttons`, `list`, `carousel` e **formulário (webview)** com respostas no Chatwoot + webhook `form_response`.
- **Mensagem que some após leitura** (`disappearing`), mensagens temporárias por chat.
- **Editar, reagir, encaminhar, marcar como lida, digitando/gravando, presença.**
- **Agendamento** (`/schedule`, persiste e sobrevive a restart), **envio idempotente** (`X-Message-ID`), **busca** no histórico, **download de mídia recebida**.
- **Contatos, grupos, canais, perfil, privacidade, bloqueio, proxy por sessão**.

□□ Disparos em massa

- `POST /blast`: campanha de texto/imagem com **pacing anti-ban**, status e cancelamento.
- `POST /broadcast`: **disparo de ligações** (áudio ou vídeo) tocando um arquivo gravado.
- `GET /restriction-status`: detecta **restrição/shadow-ban** da conta (`reachout_timelock`).

□□ Webhooks confiáveis

- Eventos por sessão (`message`, `receipt`, `call_*`, `recording`, `form_response`, `group_participants`, `restriction`...), **filtro de eventos**, **assinatura HMAC-SHA256** (`X-Webhook-Signature`), **retry com backoff**, **circuit breaker**, **DLQ** com replay.

□□ Chatwoot

- Contato/conversa por telefone, mensagens **WhatsApp ↔ Chatwoot** (texto + mídia) com **1 QR só** por número; grupos e canais (opcionais); **assinatura do atendente**, **always online**, **marcar lidas**; **importar histórico sob demanda**; mensagem editada, origem de anúncio (Click to WhatsApp), nota privada da gravação.

- **Widget de chamada** (`widget.js`): botão de telefone na conversa, toca e abre sozinho na chamada recebida, áudio por WebRTC ou WebSocket, **vídeo**, token de widget por conta.
- Correção do **9º dígito BR / LID** em todos os envios; sem contato "lixo" com LID.

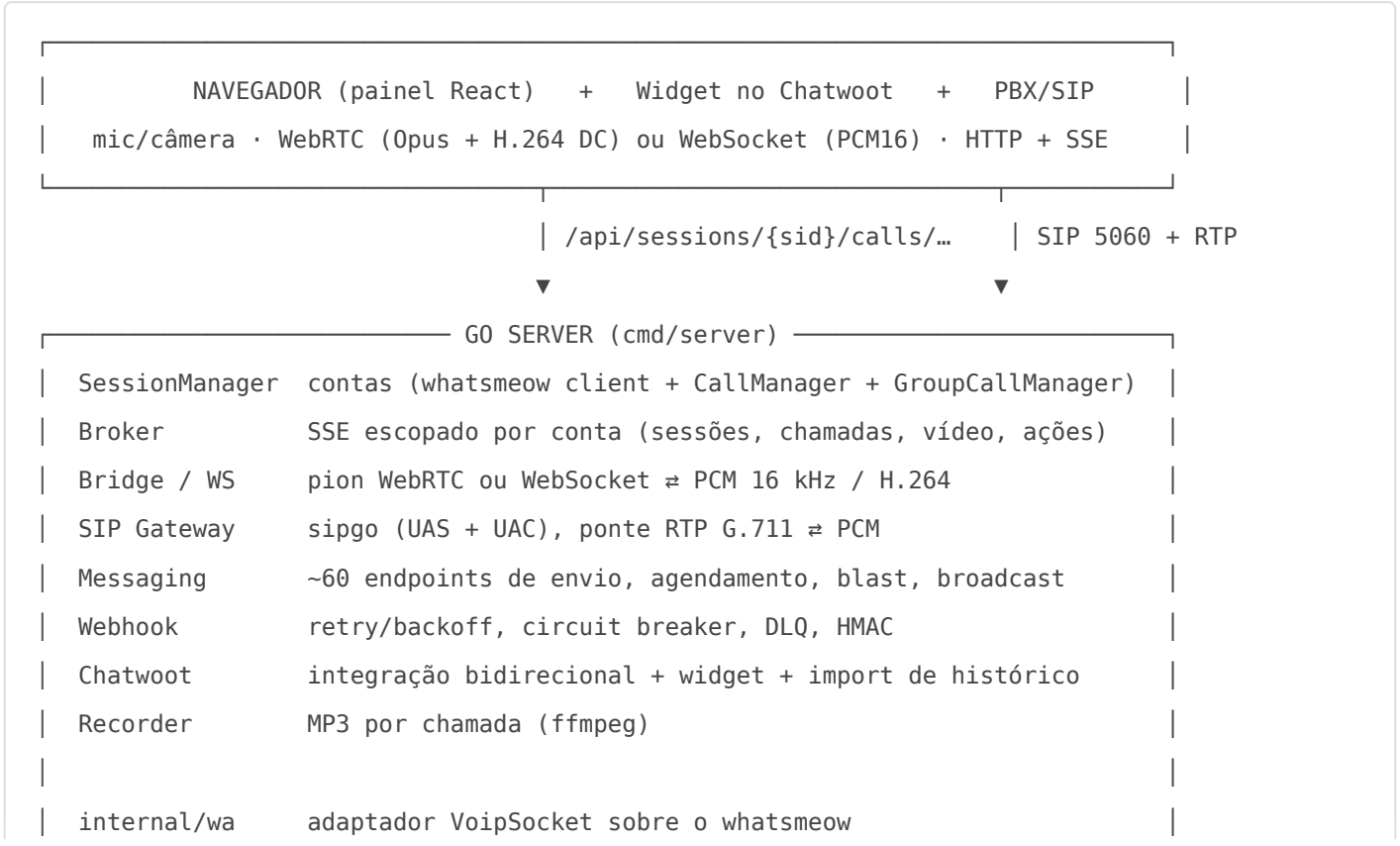
□□ Pareamento e segurança

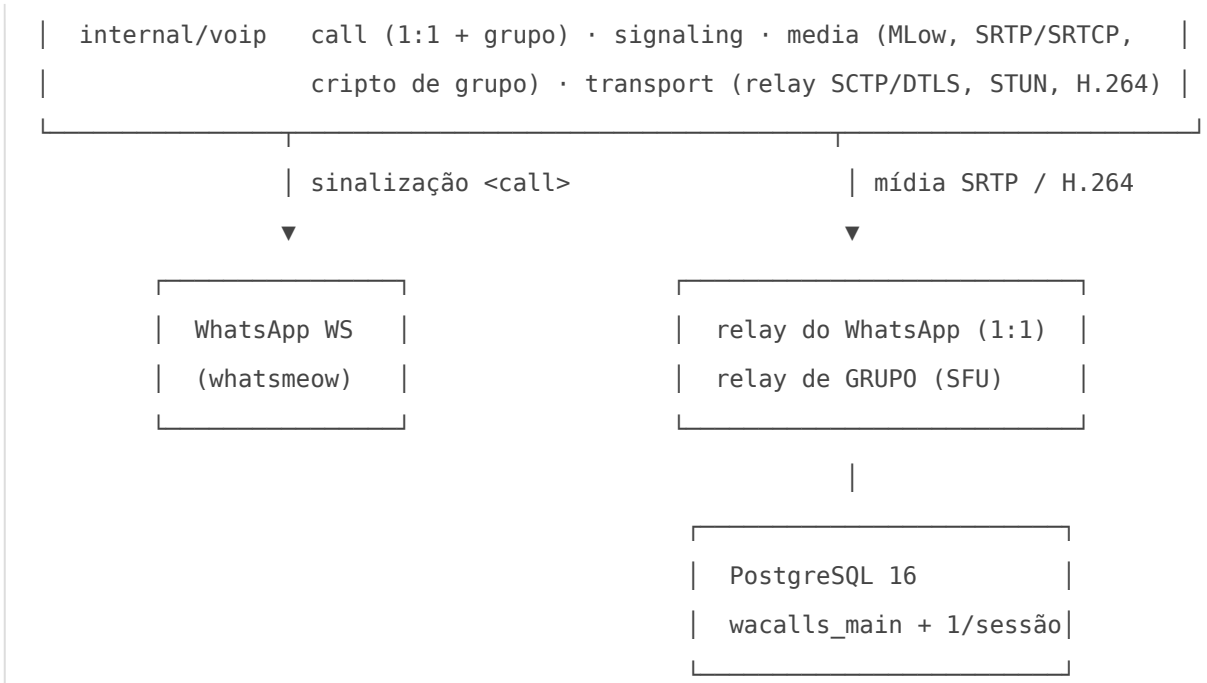
- Pareamento por **QR**, **código** (`/pair-code`) e **passkey** (`/pair-passkey`, com extensão de navegador própria em `passkey-extension/`).
- **API key** (`X-API-Key`), **tokens de widget** assinados, guarda **anti-SSRF** no download de mídia por URL, **escopo por conta** nos eventos SSE.

□□ Observabilidade e infra

- `GET /livez`, `/readyz`, `/metrics` (Prometheus), versão/commit do build em `GET /api/config` e no painel.
- **PostgreSQL por sessão** (estilo WAHA), **imagem multi-arch** (amd64 + arm64) publicada pelo CI em Docker Hub.

□□ Arquitetura





Estrutura

Caminho	Responsabilidade
`cmd/server`	HTTP/SSE, sessões + store (Postgres), ponte WebRTC/WS, auth, mensagens, agendamento, blast/broadcast, webhook, Chatwoot, gravação, **SIP** (`sip_gateway.go`, `sip_uac.go`, `sip_rtp.go`), **grupo** (`groupcall.go`)
`internal/wa`	`VoipSocket` — envia/recebe stanzas `<call>` via whatsmeow
`internal/voip/core`	Tipos de domínio, constantes, interface `VoipSocket`
`internal/voip/wanode`	Helpers de node WhatsApp e JID
`internal/voip/media`	Codec MLow, RTP, **SRTP/SRTCP**, SSRC, resampling, derivação de chaves (1:1 e **epoch de grupo**)
`internal/voip/transport`	Relay SCTP (1:1), **relay de grupo** (DTLS direto + Allocate de grupo), STUN, empacotamento H.264
`internal/voip/signaling`	Build/parse de `<call>` (offer/accept/vídeo/grupo/rekey), cripto da call-key
`internal/voip/call`	`CallManager` (1:1) e `GroupCallManager` (grupo), pipeline de vídeo, mixer de áudio
`client/`	Painel React 19 + Vite + Tailwind v4 + shadcn/ui (discador, chamadas, vídeo, grupo, sessões, SIP, Chatwoot, histórico, login)
`client/public/widget.js`	Widget de chamada (áudio + vídeo) embutível no Chatwoot
`client/public/openapi.yaml` · `api-docs.html`	Documentação OpenAPI/Swagger da API (`/api-docs.html` no servidor)
`passkey-extension/`	Extensão Chrome para pareamento por passkey

⚙ Início Rápido

```
git clone https://github.com/AstraOnlineWeb/AstraCalls.git
cd AstraCalls
git submodule update --init          # codec opus_mlow (necessário p/ áudio ao vivo)

go mod download                      # dependências Go
cd client && npm install && cd ..    # dependências do painel React
```

É preciso um **PostgreSQL** (usuário com `CREATEDB`):

```
export WACALLS_PG_URL='postgres://wacalls:senha@127.0.0.1:5432/postgres?sslmode=disable'
```

Rodar (somente sinalização — sem compilador C; pareia e chama, sem áudio)

```
go run ./cmd/server -addr :8080          # adicione -debug para logs verbosos
```

Rodar (áudio ao vivo — codec MLow nativo via cgo)

```
CGO_ENABLED=1 \
CGO_LDFLAGS="-L$PWD/native -Wl,-rpath,$PWD/native" \
go run -tags mlow ./cmd/server -addr :8080 -debug
```

> O áudio real exige cgo + `native/libopus_mlow.so` (compilada de opus_mlow). A imagem Docker já vem com tudo.

Abra `http://localhost:8080`, clique em **Nova sessão** e escaneie o QR em **WhatsApp → Aparelhos conectados** (ou use código de pareamento / passkey).

Painel React em modo dev

```
cd client
npm run dev      # Vite na :5173, faz proxy de /api → http://localhost:8080
```

⚙️ Configuração

Flags

Flag	Padrão	Significado
`-addr`	`:8080`	Endereço HTTP
`-pg-url` / `-pg-namespace`	env	URL do Postgres e prefixo dos bancos por sessão
`-static`	`client/dist`	Diretório do painel estático
`-debug`	`false`	Logs verbosos (inclui o log interno do whatsmeow)
`-max-calls-per-session`	`8`	Chamadas simultâneas por sessão (`0` = ilimitado)

Variáveis de ambiente

Env	Padrão	Significado
`WACALLS_PG_URL`	—	URL de manutenção do Postgres (usuário com `CREATEDB`)
`WACALLS_PG_NAMESPACE`	`wacalls`	Prefixo dos bancos (`wacalls_main` + `wacalls_<id>`)
`WACALLS_API_KEY`	—	Se setada, exige `X-API-Key` em `/api/*` (**use em produção**)
`WACALLS_PUBLIC_IP`	—	IP público p/ NAT 1:1 / ICE-TCP / SDP do SIP. `auto` descobre o IP real mesmo dentro do Docker/NAT de nuvem (interface pública → STUN → HTTPS); o IP achado e a fonte aparecem em `GET /api/config` (`publicIp`, `publicIpSource`)

Env	Padrão	Significado
`WACALLS_UDP_PORT`	—	Porta de mídia WebRTC (UDP + ICE-TCP)
`WACALLS_DEFAULT_TRANSPORT`	`auto`	Transporte de áudio do navegador: `auto` (WebRTC → fallback WS), `webrtc`, `websocket`
`WACALLS_MAX_CALLS`	`8`	Equivalente a `-max-calls-per-session`
`WACALLS_RING_TIMEOUT_SECONDS`	—	Expira chamada recebida que nunca é encerrada
`WACALLS_UNIFIED_SESSION_MINUTES`	`5`	Intervalo mínimo entre os sinais "unified session" enviados ao conectar, como faz o WhatsApp Web (`0` desliga)
`WACALLS_DEVICE_NAME`	`Windows`	Nome do aparelho em "Dispositivos conectados" (vale para pareamentos novos)
`WACALLS_DEVICE_BROWSER`	`chrome`	Ícone/plataforma do aparelho: `chrome`, `edge`, `firefox`, `safari` ou `desktop`
`WACALLS_RECORDING_DIR`	`$TMPDIR/wacalls-recordings`	Onde ficam os MP3s (retenção ~48h)
`WACALLS_PUBLIC_BASE_URL`	—	Base pública p/ URLs de gravação, formulários e widget (ex.: `https://call.seudominio.com`)
`WACALLS_WIDGET_KEY`	—	Chave-mestra para emitir tokens de widget (`POST /api/widget-tokens`)
`WACALLS_WIDGET_SIGNING_SECRET`	—	Segredo de assinatura dos tokens de widget
`WACALLS_SIP_ADDR`	`0.0.0.0:5060`	Endereço do servidor SIP (UDP + TCP)
`WACALLS_SIP_ADVERTISE_PORT`	`5060`	Porta SIP anunciada no `Contact` (registro em PBX externo)
`WACALLS_RTP_PORT_MIN` / `_MAX`	`40000` / `40049`	Faixa fixa de portas RTP do SIP (publique em `mode=host`)
`WACALLS_SIP_DEBUG`	off	Logs detalhados de mídia SIP/grupo (**só para diagnóstico** — gera muito log)
`WACALLS_GROUP_CALLS`	off	`1` liga a chamada em grupo (experimental)

> **Gravação de chamada (opt-in por sessão).** Ligue em `PUT /api/sessions/{sid}/recording {"enabled":true}` (ou pelo toggle "Gravar" no painel). O áudio dos dois lados é mixado num MP3 mono 16 kHz que vira **nota privada** no Chatwoot e evento `recording` no webhook; o arquivo fica em `GET /recordings/{callId}.mp3`. Requer `ffmpeg` (já na imagem).

□□ Deploy em produção (Docker Swarm + Traefik)

A imagem oficial é **multi-arquitetura** (`linux/amd64` + `linux/arm64`) e é publicada pelo CI a cada push: `astraonline/astracalls:develop` (contínua) e `astraonline/astracalls:vX.Y.Z`

- `:latest` (releases).

```
# deploy da stack (Postgres + servidor em rede de host + proxy Traefik)
docker stack deploy -c sua-stack.yml astracalls

# atualizar para a imagem mais nova
docker service update --image astraonline/astracalls:develop --force astracalls_astracalls
```

Notas de produção:

- O servidor roda em **rede de host** para a mídia WebRTC/SIP enxergar a interface real.
- Um serviço **socat** com labels do Traefik publica o HTTP em **HTTPS** (necessário porque `getUserMedia` só funciona em contexto seguro).
- O **PostgreSQL** dedicado escuta apenas em `127.0.0.1` (não exposto à internet).
- Defina `WACALLS_PUBLIC_IP` (ou `auto`), `WACALLS_UDP_PORT`, `WACALLS_PG_URL`, `WACALLS_PUBLIC_BASE_URL` e uma `WACALLS_API_KEY` forte.
- Para o **SIP**, publique `5060/udp`, `5060/tcp` e a faixa `WACALLS_RTP_PORT_MIN..MAX/udp` em `mode=host`, e aponte `WACALLS_PUBLIC_IP` para o IP real da máquina.
- Em nuvens que bloqueiam UDP de entrada, o **ICE-TCP** (WebRTC) e o transporte **WebSocket** (`WACALLS_DEFAULT_TRANSPORT=auto`) mantêm o áudio funcionando.
- Para buildar a própria imagem: `docker build -t astraonline/astracalls:develop .` (o Dockerfile compila o codec a partir do submódulo `opus_mlow`).

□□ API

A documentação completa (OpenAPI 3, **179 rotas**) é servida pelo próprio servidor em **`/api-docs.html`** (Swagger UI) e está em `client/public/openapi.yaml`.

Todas as rotas são escopadas por sessão (`/api/sessions/{sid}/…`). Os eventos chegam por um único canal SSE (`GET /api/events?clientId=…`), marcados com o `sessionId` de origem. Se `WACALLS_API_KEY` estiver setada, envie `X-API-Key` (ou `?apiKey=` no SSE/WS).

Área	Rotas (resumo)
Sessões	`GET/POST /api/sessions`, `DELETE /{sid}`, `/logout`, `/pair`, `/pair-code`, `/pair-passkey`, `/proxy`, `/recording`, `/restriction-status`
Chamadas	`POST /calls` (`{phone, video?, record?}`), `/calls/{id}/webrtc` (SDP), `GET /calls/{id}/ws` (PCM16), `/accept`, `/reject`, `DELETE /calls/{id}`, `/hold`, `/resume`, `/pickup`, `/transfer`, `/mute`, `/hand`, `/video/{request|accept|reject|stop}`, `/spy` (espiar: WebRTC recvonly / `GET /spy/ws` PCM16), `/volume` (ganho 0..3), `/silence` (silenciar o toque só na tela), `POST /calls/fake`, `GET /calls`, `GET /history`
Grupo	`POST /calls/group`, `POST /calls/group/end`, `GET /calls/group/ws`, `GET /calls/group/video-ws`, `GET /calls/group/roster`
Disparos	`POST /broadcast` (ligações, áudio/vídeo), `POST /blast` + `GET /blasts`, `/blasts/{id}`, `/blasts/{id}/cancel`
Mensagens	`POST /messages/{text|image|audio|video|ptv|document|sticker|contact|location|link-preview|poll|poll-vote|event|event-response|pix|product|product-native|interactive|buttons|list|carousel|form|disappearing|seen|typing|forward}`, `PUT /messages/{edit|react}`, `GET /messages`, `/messages/search`, `/messages/{id}/media`, `/messages/new-message-id`, `DELETE /messages`, `/schedule`
Status / presença	`POST /status/{text|image|video|audio}`, `POST /presence`, `/presence/{jid}/subscribe`
Contatos / grupos / canais	`GET /contacts`, `/contacts/check`, `/contacts/{jid}`, `/picture`, `/business`, `/block`, `/unblock`, `GET /blocklist`; `GET/POST /groups`, `/groups/{gid}/…` (participantes, convite, pedidos, configurações, foto, assunto, descrição, sair; `POST /groups/{gid}/history/share` e `shareHistory` no add: compartilhar o histórico recente com quem entrou); `GET/POST /communities`, `/communities/{cid}` (+ grupos), `/communities/{cid}/groups` (listar/criar), `/groups/link`, `/groups/unlink`, `/participants` (+ `/participants/add`); `GET /channels`, `/channels/{id}/…` (seguir, mutar, mensagens)
Chats / perfil / privacidade	`GET /chats`, `/chats/{chatId}/messages`, `PUT /chats/{chatId}/disappearing`, `PUT /disappearing`; `GET /profile`, `/profile/qr`, `PUT /profile/status`; `GET/PUT /privacy`, `GET /privacy/status`
Webhooks	`GET/POST/DELETE /webhook` (eventos, `secret` HMAC), `GET /webhooks/status`, `/webhooks/dlq`, `POST /webhooks/dlq/{id}/replay`, `/webhooks/reenable`
Chatwoot	`GET/POST/DELETE /chatwoot`, `POST /chatwoot/webhook`, `/chatwoot/import-history`, `/chatwoot/groups/{gid}/open`, `/chatwoot/channels/{id}/open`, `GET /api/chatwoot/resolve`

Área	Rotas (resumo)
SIP	`GET /api/sip/status`, `GET/POST /sip` (credenciais p/ o PBX registrar aqui), `GET/POST /sip-ext` (registrar num PBX externo: host, porta, usuário, senha, ramal, proxy)
Widget / formulários	`GET /api/widget-key`, `POST /api/widget-tokens`, `GET /forms/{token}`, `POST /forms/{token}/submit` (públicas, token como capability)
Infra	`GET /api/config` (flags, versão, commit), `GET /livez`, `/readyz`, `/metrics`, `GET /recordings/{id}`

□□ Integração com n8n

Automatize o WhatsApp no **n8n sem escrever código**: o node oficial `n8n-nodes-astracalls` já está publicado no npm, todo em português.

Instalação: no n8n, abra **Configurações → Community Nodes → Install**, digite `n8n-nodes-astracalls` e clique em **Install**. No Docker, use `N8N_COMMUNITY_PACKAGES=n8n-nodes-astracalls`.

Credencial: a URL do seu AstraCalls e a API Key (`WACALLS_API_KEY`).

Node	O que faz
AstraCalls	55 operações em 8 recursos: **Mensagem** (texto, mídia, documento, enquete, botões, lista, formulário nativo, reagir, editar, apagar…), **Chamada** (ligar, ligação fantasma, disparo de ligações com áudio/vídeo), **Grupo**, **Comunidade**, **Contato**, **Sessão**, **Formulário** e **Requisição Personalizada** (qualquer uma das 179 rotas)
AstraCalls Trigger	Inicia o fluxo com 15 eventos: mensagem recebida, leitura, reação, voto em enquete, formulário respondido, entrada/saída de grupo, gravação de chamada, desconexão da sessão… Configura o webhook da sessão sozinho ao ativar

- **Fácil de usar:** a sessão é escolhida numa lista (nome + número), o destino aceita número com ou sem o 9º dígito e a mídia vem por URL ou de um arquivo de etapa anterior.
- **Pronto para responder:** o gatilho entrega `de`, `nome`, `texto` e `responderPara` prontos para usar no envio, e ignora as mensagens da própria conta (sem loop).
- **Agentes de IA:** o node pode ser usado como ferramenta pelos agentes de IA do n8n.

□□ **Tutorial passo a passo**: instalar, criar a credencial, enviar a primeira mensagem e montar um robô que responde, com fluxo pronto para importar e 10 fluxos de exemplo (menu com botões,

atendente de IA, enquete, disparo de ligações…).

□□ MCP para agentes de IA

O AstraCalls traz um **servidor MCP** (Model Context Protocol) embutido: Claude, ChatGPT, Cursor, VS Code, n8n e outros agentes de IA usam o WhatsApp direto, sem instalar nada.

Chaves por sessão (recomendado): no painel, botão **MCP** da sessão → **Nova chave**. A chave fica presa àquele número, pode ser **somente leitura** e é revogada a qualquer momento. Ela não permite parear, desconectar nem mexer em webhooks, Chatwoot, SIP ou proxy. O painel mostra, uma única vez, a URL e as configurações prontas para copiar. Pela API: `GET/POST /api/sessions/{sid}/mcp-keys` e `DELETE /api/sessions/{sid}/mcp-keys/{id}`.

Cliente	Como conectar
claude.ai, ChatGPT (conector personalizado)	URL `https://SEU-ASTRACALLS/mcp/<chave>`
Claude Code	`claude mcp add --transport http astracalls https://SEU-ASTRACALLS/mcp --header "Authorization: Bearer <chave>"`
Claude Desktop, Cursor, VS Code	`mcpServers` → `{"type":"http","url":"https://SEU-ASTRACALLS/mcp","headers":{"Authorization":"Bearer <chave>"}}`
n8n	Ferramenta **MCP Client** do agente de IA, endereço `/mcp` e autenticação *Bearer*

Acesso de administrador: `https://SEU-ASTRACALLS/api/mcp` com a API Key da instância (`WACALLS_API_KEY`) enxerga todas as sessões; o agente escolhe o número por nome, telefone ou ID.

44 ferramentas: enviar texto, mídia, documento, localização, contato, enquete, botões e lista; reagir, editar, apagar, marcar como lida e "digitando..."; **ler conversas e buscar mensagens**; ligar, encerrar, toque fantasma e disparo de ligações; contatos, grupos e comunidades; limite de conversas novas e restrição; e `api_request` para as demais rotas da sessão.

> Quem tem a chave usa o número: trate-a como senha e revogue as que não usa mais.

□□ Testes

```
go test ./...                # mídia (SRTP/SRTCP, STUN, RTP, H.264, cripto de grupo), sinalizaç
cd client && npm run build   # type-check + build de produção do painel
```

□□ Segurança

- Em produção, **sempre** defina `WACALLS_API_KEY` — sem ela qualquer um com acesso HTTP pode criar contas, fazer chamadas, enviar mensagens e ler histórico.
- O banco de cada sessão guarda **credenciais do WhatsApp**. Mantenha o Postgres protegido e fora da internet.
- Exponha sempre por **HTTPS** (`getUserMedia` exige contexto seguro).
- Use `secret` no webhook e verifique `X-Webhook-Signature` no consumidor.
- O SIP na 5060 recebe varredura da internet o tempo todo: só sessões com credenciais válidas registram (Digest MD5); considere restringir a porta por firewall aos IPs dos PBXs.

□□ Suporte Profissional

Precisa de ajuda para melhorar, customizar ou implementar o projeto?

□□ **WhatsApp:** +55 61 9 9687-8959

□□ Temos uma equipe especializada para:

□ Customizações e melhorias □ Implementação e deploy completo □ Configuração de arquitetura SaaS □ Integração com outras APIs □ Desenvolvimento de features específicas □ Suporte técnico dedicado □ Consultoria em automação WhatsApp □ Treinamento e documentação

---

□□ Licença

O AstraCalls é distribuído sob a licença **GNU AGPL-3.0** — veja LICENSE.

Isso significa que qualquer uso em rede (inclusive SaaS) exige disponibilizar o código-fonte das modificações aos usuários do serviço.

O AstraCalls é um fork do WaCalls, que é licenciado sob **MIT**. Conforme exigido pela MIT, o aviso de copyright original (© 2026 jotadev66) é preservado em LICENSE.WaCalls. As porções originais permanecem sob os termos MIT; o trabalho derivado, como um todo, é licenciado sob AGPL-3.0.

Revision #4
Created 2026-11-06 14:56:27 UTC by Administrador
Updated 2026-11-09 02:01:36 UTC by Administrador