api0

GUIDE 01

Comment mettre ton API existante dans Claude en moins de 10 minutes

Ton application fait déjà le travail. Voici comment la rendre utilisable depuis Claude, sans écrire de serveur MCP, avec chaque action faite au nom de la personne qui demande. Le point de départ des guides Teams, WhatsApp et Telegram.

Mis à jour le 11 novembre 2026 · Lire en ligne : api0.ai/fr/guides/ton-api-dans-claude

Ton logiciel a une API. Elle sait créer un ticket, sortir une facture, lister les interventions du jour. Mais tes utilisateurs, eux, passent leur journée dans Claude. Entre les deux, il manque un pont.

Construire ce pont soi-même, c'est écrire un serveur MCP, gérer l'OAuth pour chaque utilisateur, stocker les identifiants de ton application quelque part où l'IA ne peut pas les lire, journaliser chaque appel. Des semaines de travail.

Ce guide montre la version courte avec api0 : on importe l'API une fois, on choisit qui peut se connecter, et on l'ajoute dans Claude. Les mêmes outils servent ensuite Teams, WhatsApp et Telegram.

Ce qu'il te faut

- Une API HTTP accessible depuis Internet (REST/JSON).
- Sa description : une spec OpenAPI (YAML ou JSON), ou à défaut la liste de tes endpoints.
- Un compte Google pour te connecter à app.api0.ai.

Pas d'API sous la main pour essayer ? Prends une des specs d'exemple (CRM, tickets, facturation, interventions, RH) et pointe-la vers un serveur de mock.

Étape 1 — Importer l'API (3 minutes)

1. Connecte-toi sur **app.api0.ai**.
2. Onglet **Tools** → importe ta spec OpenAPI.
3. Chaque opération devient un **outil** : son nom, sa description et ses paramètres sont ce que Claude verra.
4. Un test intégré appelle chaque outil avant qu'il soit compté comme configuré.

Pas de spec ? Tu peux décrire tes endpoints en langage naturel, api0 les transforme en outils.

Le conseil qui change tout : soigne les descriptions. Claude choisit un outil d'après sa description. Écris *quand* l'utiliser (« à utiliser avant de créer un contact, pour éviter les doublons ») et *quoi confirmer*

avant une action irréversible (« demande confirmation avant d'envoyer une facture »). Une bonne description vaut plus que dix lignes de prompt.

Quelques détails gérés pour toi :

- les paramètres aux noms que les clients MCP refusent (`$filter`, `page[size]`) sont exposés sous un nom propre (`filter`, `page_size`) et renvoyés à ton API sous leur vrai nom ;
- une erreur de ton application revient à Claude comme un résultat lisible, pour qu'il puisse corriger et réessayer ;
- un appel qui échoue n'est jamais facturé.

Étape 2 — Régler l'accès (3 minutes)

Onglet **Access**. Deux questions.

Qui peut se connecter ? Par défaut, les gens se connectent avec Google. Une entreprise peut brancher son propre fournisseur d'identité (Entra ID ou tout fournisseur OIDC) : ses employés se connectent alors avec leur compte habituel.

Avec quel identifiant ton application est-elle appelée ? C'est la question clé. Les options, de la plus simple à la plus fine :

Option	Pour qui
Jeton d'identité signé par api0	Ta propre application : elle vérifie que l'appel vient d'api0 et pour quel tenant. Aucun secret à stocker.
Jeton ou en-têtes fixes	Une API tierce avec une clé unique.
OAuth client credentials	Une API qui délivre des jetons de service.
Identifiant de chaque personne	Chaque utilisateur ajoute son propre jeton, ou connecte son compte en OAuth : chaque action est faite **en son nom**, avec **ses** droits.

Dans tous les cas, l'identifiant reste dans api0. Claude ne le voit jamais.

Troisième réglage : **qui paie les appels d'outils**, le tenant ou chaque personne avec son propre plan.

Étape 3 — Claude (2 minutes)

Onglet **Channels** → **MCP**. Tu y trouves l'URL du serveur MCP de ton tenant et un Client ID.

Dans claude.ai → Paramètres → Connecteurs → **Ajouter un connecteur personnalisé** :

1. colle l'URL du serveur ;
2. choisis **« Use your own OAuth client »** et colle le **Client ID** (pas l'URL), sans secret ;
3. connecte-toi.

C'est tout. Demande à Claude « quels outils as-tu ? », puis pose une vraie question. Pour une équipe, crée un **Projet Claude** avec des instructions métier (un template est fourni).

Cursor, Claude Desktop et les autres clients MCP se connectent de la même façon.

Et ensuite : Teams, WhatsApp, Telegram

Les mêmes outils, avec les mêmes accès, peuvent répondre dans une messagerie. Chaque canal a son guide :

- **Microsoft Teams**, avec un Azure Bot à ton nom : le guide Teams.
- **WhatsApp**, avec ton numéro WhatsApp Business : le guide WhatsApp.
- **Telegram**, avec un bot créé chez @BotFather : le guide Telegram.

Dans chaque canal, chaque personne relie une fois son compte de messagerie à son compte api0 ; ensuite elle parle à ton API comme à un collègue, et chaque action est faite en son nom.

Ce que tu obtiens

- **Une seule configuration** : les mêmes outils dans Claude et, ensuite, dans Teams, WhatsApp et Telegram.
- **Chaque action au nom de la personne**, avec ses droits dans ton système.
- **Aucun secret dans l'IA** : les identifiants restent chiffrés dans api0.
- **Un journal de chaque appel** (onglet Activity) et des crédits par appel.
- **Rien à héberger** : pas de serveur MCP, pas de nouvelle interface.

Questions fréquentes

Mon API est privée, derrière un VPN. api0 doit pouvoir l'atteindre. Expose-la derrière une passerelle qui n'accepte que les appels signés par api0 (option « jeton d'identité signé »).

Je veux limiter ce que l'IA peut faire. N'importe que les opérations utiles, et préfère l'identifiant de chaque personne : ton application applique alors ses propres droits.