E-BOOK

How to Make Your SaaS Agent-Ready

Ömer İlhan

October 2026
omerilhan.xyz

What this guide is

AI agents now pick and use SaaS products for their users. In August 2026, Resend's MCP server handled 1,062,650 calls, ten times its April figure, according to a post on X by Zeno Rocha, Resend's founder. These calls can come from existing users working with Resend through agents and from agents setting up Resend for new users; the figure does not separate the two. This guide explains what a SaaS product has to do to be chosen and kept by agents. It follows the five stages an agent goes through with a service, from Discovery to Usage growth, and gives each stage a list of things to do, with Resend (email) and Tally (forms) as examples. Each stage matches one step of the agent funnel: Be discoverable, Be remembered, Be usable, Be valuable and Be preferred. Facts are dated and linked to their sources, and reasoned claims are marked as hypotheses.

Contents

- How an agent decides
- How the stages connect over time
- Stage 1: Discovery
- Stage 2: Understanding
- Stage 3: Transaction
- Stage 4: Trial
- Stage 5: Usage growth
- Measurement
- Keeping agents safe
- Serving business customers
- The agent-readiness scorecard
- Glossary
- Appendix: verification notes

How an agent decides

An agent is a language model working in a loop with tools. It turns the request into a goal with constraints, such as "send email from this app, free if possible," and drafts a plan from what it already knows. Then it calls a tool (a web search, a page fetch, a terminal command or an MCP server), reads the result and decides the next step. It typically needs the user when it requires consent, identity, payment, or information and access it cannot obtain on its own. Ambiguous instructions, missing inputs and errors it cannot resolve also bring it back to the user.

Two things follow. Everything your service returns reaches the agent as a tool result, so anything it cannot parse costs a step or forces a guess. And every stop for the user is a point where the session can end, because the user may be busy or unsure. A service that is easy for agents keeps both costs low.

Four forces shape whether an agent picks your service and keeps using it:

- The model brings a memory from training, which decides what comes to mind first, and it reasons over what it reads during the task, including your docs and tool descriptions.
- The agent environment is the app or editor the model runs in; installed connectors, connector directories, platform defaults and permission rules decide what the agent can see and act on without extra work.
- The service is your docs, API, MCP server, errors, pricing and reliability, and it is the only force you fully control.
- The user brings preferences, existing accounts, company rules, budget and trust, and decides what gets connected and paid for.

One piece of work on your side often moves several of these forces at once.

Why an agent picks one service over another

Outside the model providers, nobody knows exactly how a model ranks services, and there is no single algorithm to optimize for. What can be described is which signals reach the agent and who controls them.

You control some signals directly: your docs, pricing page, error messages, MCP server, sandbox and reliability. The agent reads these during the session, so a change works the same day.

You can influence others without controlling them: listings in connector directories and MCP registries, partnerships with provisioning platforms and AI app builders (platforms such as Lovable, v0 and Bolt.new, where users build an app by describing it and the platform's agent picks services such as email or payments), and how often your service appears in public code and comparison pages. These shape what the agent sees before it reads your docs.

The rest depends on model providers and agent platforms, and on the user. What the model learned in training, which connector directories and defaults a client ships with, and how its search ranks pages are outside your hands. The user's budget, stack, compliance rules and existing accounts act as filters: a service that fails one is dropped however well it does elsewhere.

Some agent environments also carry results from earlier work. A project file may name the services the project already uses, or a memory feature may record a past choice. An installed connector that worked before also carries over. Within one session, a failed attempt usually pushes the agent to an alternative.

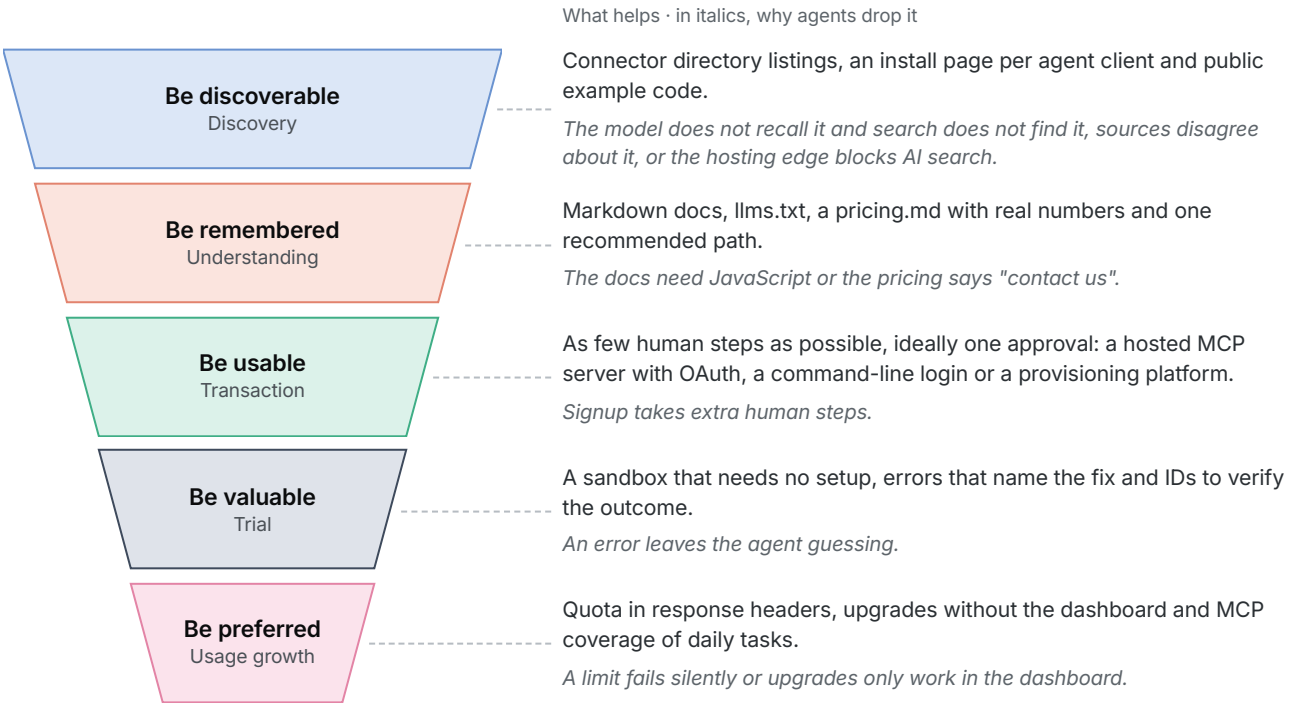
How much past outcomes weigh across sessions varies by client and is not documented in a general way.

These signals decide how far a service gets through the agent funnel, five steps that match the five stages of this guide one to one:

- Be discoverable (Discovery). The agent puts your service on its shortlist while it researches.
- Be remembered (Understanding). The agent reads your docs and pricing during the session, confirms your service fits the user's requirements and picks it over the alternatives. Here, remembered means staying on the shortlist after the agent evaluates your current docs and pricing; it does not refer to the model's training memory.
- Be usable (Transaction). The agent obtains access with minimal human intervention, ideally a single approval when the user and platform prerequisites are already satisfied. This is the narrowest point of the path, because it is the one step the agent cannot finish alone.
- Be valuable (Trial). The agent gets a working result in the same session, such as a delivered email or a created form.
- Be preferred (Usage growth). Your service becomes part of the user's daily work. What worked once is likely to be used again.

A service can be discoverable and drop out at Be remembered because of unclear pricing, or be remembered and drop out at Be usable or Be valuable because of an extra human step or an error the agent cannot fix.

The agent funnel: what helps at each step, and why agents drop a service



What makes an advantage last (hypothesis)? Publishing an MCP server or markdown docs is easy to copy, so competitors soon match it. Being the default inside platforms the user already works in is harder to copy. So is a record of working on the first try, which lives on in users' projects and installed connectors, and possibly in future training data. Covering the tasks users do every day keeps the service in use after the first integration. None of this has been measured yet; the selection rate in the Measurement section is one way to check it.

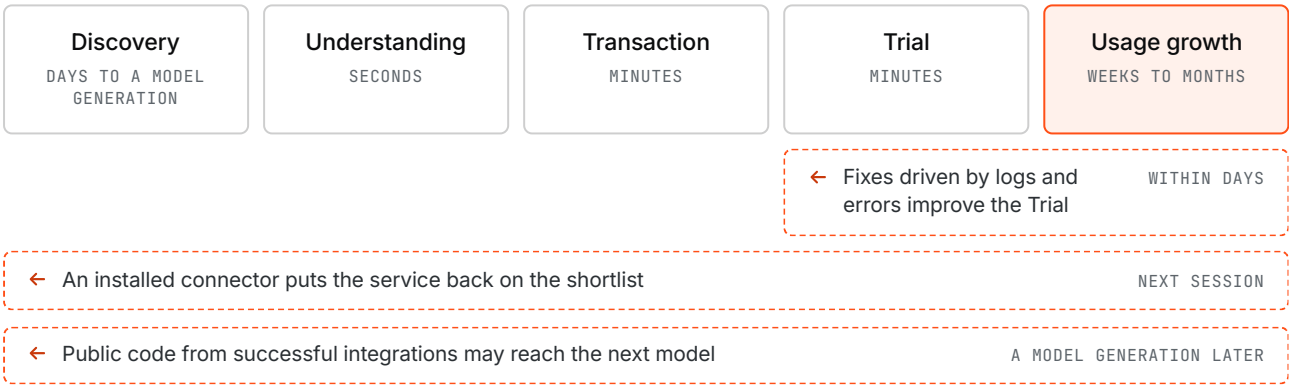
How the stages connect over time

An agent uses two kinds of knowledge. Its memory is what the model learned in training: free to use, but frozen at a cutoff date until a new model is released, and old models stay in use for months. Its live context is what it reads during the session, such as search results, fetched pages and error messages: current, but every read costs a tool call and space in the context window. So the agent leans on memory and uses live context to check it. A service has to come to mind, and it has to be readable when the agent goes looking. In the agent funnel, Be remembered refers to the second of these, staying on the shortlist after the agent reads your current docs and pricing, not to training memory.

Each force moves at its own speed. Memory changes once per model generation. The model's reading of your pages updates on every call, so a docs fix works the same day. The agent environment changes within days of an install or a catalog change, and installed tools then persist across sessions. The user changes over weeks or months. The stages follow: Discovery moves in days through listings and in model generations through memory, Understanding happens in seconds, the Transaction and the Trial take minutes, and Usage growth takes weeks to months.

Usage then feeds back along three routes. Fixes driven by logs and errors improve the Trial within days. An installed connector puts the service back on the shortlist in the user's next session. Public code from successful integrations may reach the next model, a model generation later.

FIVE STAGES AND THREE FEEDBACK ROUTES



Over time, memory may also work in a service's favor (hypothesis). When developers ship apps that use a service, some of that code ends up in public repositories and tutorials. If it reaches training data, the next model sees the service paired with the task more often. It may then suggest the service sooner and write a working call without fetching the docs, so the first test needs fewer steps. Shorter paths lead to more successful integrations, and those leave more public code behind. No public data shows which code a model was trained on, so treat this as something to test: track your unprompted recommendation rate across model releases.

Memory can also work against you. A model trained before you renamed an endpoint or changed a parameter keeps writing the old call with confidence, and the user sees your service fail. Only what the agent reads in the session can correct it, which means current docs and error messages that say what changed. Keep breaking changes rare, and when you make one, say so in the error message.

Stage 1: Discovery

FUNNEL STEP Be discoverable

GOAL Get on the agent's shortlist.

What the agent does

It pulls the category and constraints out of the request; "add transactional email to this Next.js app, free if possible" becomes an email API, Next.js and a free plan. The model recalls services it associates with that category from training, because recall is instant while a search costs a tool call. It probably associates a service with a category when the two appear together often and consistently in docs and code, though how it ranks candidates is not public. It searches when its memory is thin or the request implies something recent, and also when the user asks for "the best" option. Search results and the assistant's connector directory or plugin catalog then add or remove candidates. The agent keeps two or three, because reading each one's docs uses up the context window.

What decides it

Memory is only one way in. Installed connectors can be offered before anything else, and a platform's default, such as an AI app builder's preferred email provider (a builder is a platform like Lovable or v0 that builds apps from a description), can decide before memory is consulted. A listing appears in search within days, while any effect through memory arrives a model generation later. The user can override everything by naming a service. The tools they already use, such as Vercel or Stripe, favor services sold in those marketplaces, and a company allowlist can rule a service out. Services drop out when the model does not recall them and search does not find them, or when facts about them conflict across sources. Public code and integration examples can help models associate a service with a task and generate working integrations. Whether a lack of public code materially reduces a service's chance of becoming a default remains an untested hypothesis.

Your hosting provider's edge can also decide this before any model does, wherever you host. Providers such as Cloudflare and Vercel sort AI bots into groups and let you block each group. Search bots, such as OAI-SearchBot, Claude-SearchBot and PerplexityBot, decide whether AI search answers can include you. Training bots, such as GPTBot and ClaudeBot, collect pages that may be used to train future models. Blocking training bots is a legitimate business choice, but it closes only one route through which a service's pages could contribute to future model training. Publicly available code, other websites, and existing training data may still expose the service or its name to models. On Cloudflare, choosing Block for training also blocks crawlers that do search and training at once; its July 1, 2026 announcement names Googlebot, Applebot and BingBot. Since September 15, 2026, a Disallow AI Training setting does two things: it publishes a no-training preference in robots.txt, which mixed-use crawlers that Cloudflare labels Accountable can honor while they keep indexing the site for search, and it blocks other training crawlers outright. Support is not yet equal: Cloudflare says Google and Apple honor the preference today, while Microsoft is still developing robots.txt support for Bingbot, targeted for early 2027.

Examples

Resend has one install page per agent client. Its MCP docs give setup tabs for Claude, Claude Code, Cursor, Codex, Copilot, Gemini CLI, Devin, Zed, Warp and others, each matched to a page such as `resend.com/cursor`, so the agent's search for "Resend + client" lands on an exact match.

Resend is listed inside the assistants as a Claude connector and a Codex plugin. Both install the MCP server and skills together, following the open Agent Plugins spec, so the assistant can offer an install from its own catalog.

Resend's AI onboarding page lists guides for the AI app builders Lovable, v0, Bolt.new, Replit, Base44 and Leap.new. A builder's default reaches users today, and the apps it ships may feed the memory loop described earlier (hypothesis).

Tally publishes pages that answer the category question, such as Best Form Builder with MCP Support in 2026, and repeats one checkable claim across its pages: its MCP server is free on all plans.

To do

- [ ] Write 30 test prompts for your category that do not name you, across budgets and stacks, and run each one several times a month in ChatGPT, Claude, Gemini, Perplexity and two coding agents, logging rank and wording.
- [ ] Publish one install page per major agent client, because agents search for "service + client".
- [ ] List in the Claude connector directory, ChatGPT apps, Cursor plugins and the official MCP Registry.
- [ ] Ship a plugin that bundles your MCP server and skills.
- [ ] Integrate with the AI app builders your customers use (platforms like Lovable, v0 and Bolt.new that build apps from a plain-language description).
- [ ] Publish official example repositories for your five most common stacks, because agents copy examples.
- [ ] State what makes you different in one sentence and repeat it everywhere, so search, connector directories and future models repeat it accurately.
- [ ] Decide what your hosting edge allows for each group of AI bots (search and training), and check that the setting in effect matches your robots.txt.

Stage 2: Understanding

FUNNEL STEP Be remembered

GOAL Let the agent confirm the service fits, choose it over the alternatives and use it correctly, from what it reads in the session.

What the agent does

It looks for an entry point, usually the docs index or an `llms.txt` file, though agents arriving from search may start on any page. It checks the user's constraints: does the free plan send enough, and is the needed feature on a plan the user can afford? Constraints like "free" or "under $25 a month" are hard rules for the agent. It compares two or three candidates on price and setup effort, because it has to justify the choice. It looks for the recommended way to integrate (SDK, REST API or MCP), because one clear path stops it from mixing methods. Then it writes code, starting from memory and correcting it with what it just read.

All of this happens inside the context window. Long HTML pages full of menus and scripts consume more of the context window, while clean markdown often conveys the same information with fewer tokens and less irrelevant content. Many fetchers do not run JavaScript, so script-built pages can arrive empty. Agents fetch through their own user agents, such as ChatGPT-User, Claude-User and Perplexity-User, and a firewall rule that blocks them hides your docs entirely.

Most of these blocks sit in your hosting provider's edge, so check its settings on your docs and pricing pages: AI bot rules, general bot protection, WAF rules and robots.txt. User-triggered fetchers such as ChatGPT-User and Claude-User have to get through, and a JavaScript challenge stops them as surely as a block. robots.txt still counts, because Anthropic's documentation says Claude-User follows it, while OpenAI and Perplexity say their user-triggered fetchers may not. Two current examples: since September 15, 2026, Cloudflare offers new domains monetized through ads a preset that blocks Agent bots on pages with ads and sets Training to Disallow AI Training, while Search stays allowed, and Vercel's AI Bots managed ruleset can deny AI fetchers while its Bot Protection ruleset can challenge non-browser traffic. Cloudflare's Markdown for Agents is available at no extra cost on Pro, Business and Enterprise plans and to SSL for SaaS customers, and it converts pages to markdown for agents that ask for it. When we checked on October 22, 2026, the menu of Cloudflare's documentation still labeled the feature "Beta". Unless your origin sends its own `content-signal` header, Cloudflare adds a default one signaling that the content may be used for AI training, search and AI input. The header states a preference; whether an AI system honors it is up to that system.

What decides it

The model's memory of your API and prices may be a model generation old, and it trusts what it reads now over that memory. If your MCP server is already connected, the tool descriptions answer many questions without a docs fetch, and an installed docs MCP server or skill does the same. The service wins with clear prices and limits, one recommended path, an SDK in the user's language, and trust signals such as SOC 2 and GDPR for business users. The user adds a budget and the framework they already use, and sometimes compliance needs such as where data is stored. Because all of this is read live, a fix to a docs page or a

price table changes the agent's decision the same day. Agents drop a service when its docs need JavaScript or its pricing says "contact us" or "generous limits". They also write broken code when one task has several documented ways, or when old examples carry no warning.

Once your MCP server is connected, its tool descriptions become some of your most-read documentation. Each one should say when to use the tool, because vague or overlapping descriptions make the model pick the wrong one. Say just as plainly when MCP fits and when the API does: MCP for work the user wants done now in the conversation, such as "send a test", and the REST API with an SDK for code that runs inside the user's app or handles high volume.

Examples

Resend opens each `.md` docs page we read with a pointer to `llms.txt`, so an agent that lands mid-docs from search finds the full index in the first lines.

Resend offers its docs in sizes that fit the context window. Every page has a `.md` version, and `llms-full.txt` holds all the docs in one file. A docs MCP server lets the agent query as it goes. All of them are listed on its AI onboarding page.

Resend's pricing page links "Are you an AI agent? See pricing" to pricing.md. That file lists every plan and marks each feature yes or no. It also answers agent questions directly. Upgrading a self-serve plan does not raise the API rate limit, and no plan sits between Free and Pro. Enterprise plans list their own rate limits, and support reviews requests for higher ones. There are no annual plans either. A clear "no" stops the agent from promising a feature the plan lacks.

Resend's usage limits page states the default rate limit (10 requests per second per team) and the Free daily quota, which resets at midnight UTC. It also names the headers that report usage, so the agent can build a back-off that matches the real limit.

Resend ships skills that cover retries and idempotency keys, so the agent writes production code on the first pass instead of a demo that sends twice on retry.

Tally's MCP page answers "MCP or API?": MCP for quick work in conversation, the API for embedding and automation at scale.

To do

- [ ] Serve every docs page as markdown (a `.md` suffix or content negotiation).
- [ ] Start each `.md` page with a pointer to `llms.txt`.
- [ ] Publish `llms.txt` and `llms-full.txt`, and a docs MCP server if your docs are large.
- [ ] Publish a `pricing.md` with every plan, limit and an explicit yes or no, linked from the pricing page.
- [ ] Answer agent questions in the pricing FAQ, such as rate limits per plan and what happens at the quota.
- [ ] Ship skills that cover retries and idempotency.
- [ ] Write an "MCP or API?" page and keep one recommended path per task.
- [ ] Mark deprecated API versions clearly, because agents copy the first example they find.
- [ ] Test that your hosting edge (firewall, bot protection, AI bot rules and robots.txt) lets agent fetchers read your docs and pricing, by fetching them as ChatGPT-User and Claude-User.
- [ ] Write each MCP tool description to answer "when should I use this?"

Stage 3: Transaction

FUNNEL STEP Be usable

GOAL Get the agent access with as little human intervention as possible, ideally a single approval when the user and platform prerequisites are already satisfied. This is the narrowest point of the path, because it is the one step the agent cannot finish alone, and each extra human step gives it a reason to switch services.

What the agent does

It picks a way in from the docs or from memory and starts the flow itself, by adding an MCP server address or running a login or provisioning command. The user approves in a browser, signing up first if they have no account, because consent and identity must come from them. The service issues credentials limited to what was approved (scopes), the agent stores them in its client or the project's `.env` file, and it confirms access with a harmless call such as listing domains.

The ways in differ mainly in how many human steps they take.

WAY IN	HUMAN STEPS	WHAT IT NEEDS	AT RESEND
Sign up and paste an API key into `.env`	Four or five	An account and dashboard access	Supported for the local MCP server and SDKs
Hosted MCP server with OAuth	One approval, plus a signup for new users	A client that supports hosted MCP servers with OAuth	`https://mcp.resend.com/mcp`
Command-line login	One approval; device flow covers tools without a browser	A command-line tool	`resend login`
Provisioning platform	Can remove the separate signup and API-key steps; the agent then provisions access within the user's existing workflow	Stripe Projects: a Stripe account and the Stripe command-line tool with its projects plugin. Vercel Marketplace: a Vercel account with a project and a domain bought through Vercel	Stripe Projects and the Vercel Marketplace
API key sent as a Bearer token	One, done in advance by a person	Servers and automated pipelines with no browser	Documented

What decides it

The client has to support hosted MCP servers with OAuth, and its permission rules may ask the user before each install. On the service side, the number of steps matters, along with how narrow the requested permissions are and whether access needs a paid plan. The user brings trust and existing accounts, such as GitHub or Stripe. Company rules on third-party apps can also block a connection. This stage takes minutes because it waits on a person. It is lost when the docs say "copy your API key from the dashboard" or when MCP only works on paid plans. All-or-nothing permissions also make users hesitate.

The edge matters here too. Your MCP server and its OAuth endpoints (the discovery addresses under `/.well-known/` and the token endpoint) are called by agent clients, often from the client's own servers. A bot rule on those addresses breaks the connection before the user sees a consent screen, so exempt them and keep bot rules for your marketing pages.

Examples

Resend removes the signup form through Stripe Projects. `stripe projects add resend/email` creates a Resend account linked to the user's Stripe account, or links an existing one, and writes `RESEND_API_KEY` into `.env`, and the guide says the agent can run it. The user still needs a Stripe account and the Stripe command-line tool with its projects plugin, plus a payment method before any paid upgrade.

Resend's hosted MCP server opens a browser for OAuth, and clients that cannot open one send an API key as a Bearer token instead, as its MCP docs describe. Interactive agents and automated jobs both have a path.

Resend keeps credentials narrow. API key permissions exist on every plan, and OAuth tokens carry scopes; the `invalid_permission` error tells the agent when a token lacks a required scope. Users approve send-only access more readily, and a leaked send-only key does less harm.

Tally's MCP server is free on every plan, with OAuth as the recommended login, according to its MCP page. The agent never has to ask for money before the first try.

To do

- [ ] Host your MCP server with OAuth, and keep a Bearer-key path for headless clients.
- [ ] Add a command-line login with device flow, because coding agents work in terminals.
- [ ] Join provisioning platforms such as Stripe Projects and the Vercel Marketplace.
- [ ] Offer scoped keys and OAuth scopes, split into read, send and admin.
- [ ] Keep the MCP server and API available on the free plan.
- [ ] Measure the human steps from zero to working credentials and aim for one.
- [ ] Exempt your MCP server and OAuth endpoints from bot protection and AI bot rules at the edge.

Stage 4: Trial

FUNNEL STEP Be valuable

GOAL A working result in this session. A reasonable hypothesis is that agents may be more likely to continue with the first service that works, because switching incurs additional setup and evaluation costs. For an email service, a working result is a delivered message.

What the agent does

It installs the SDK and writes the smallest possible call, which separates setup problems from logic problems. It sends with sandbox values, such as a test sender address, to get proof before asking the user for production setup. It reads the response, and on an error it fixes the request and retries; agents can often correct themselves when the error says what to do. Next it verifies the user's domain. SPF, DKIM and DMARC are email authentication records that help receiving providers judge whether a message is authorized; configuring these records correctly helps establish domain legitimacy and improve email deliverability. Finally it confirms delivery through logs or a webhook, since an accepted request is not yet a delivered email. The main measure here is time to first value, from getting credentials to the first working result.

What decides it

The agent needs to run code and read logs in the same session, and many agents retry failed calls on their own. On the service side, the sandbox and the quality of errors matter most, followed by speed, reliability, idempotency and how much setup is automated. The user matters too, since someone may need DNS access to finish a step. A slow or unreliable API can lose the Trial even with perfect docs. The Trial is also lost to errors without a fix, such as a bare "Bad request", and to a missing sandbox. Manual DNS copying fails when it lands with a user who lacks access. Unsafe retries that send duplicate emails cost the user's trust.

If your API or MCP server sits behind the same edge, a challenge page or an HTML block page returned there reads to the agent as a broken service, and it cannot fix what it cannot parse. On API and MCP addresses, have the edge return errors in your API's own JSON format, and send rate limits as 429 with a `retry-after` header.

Checking that the task is done

A call that returns a success code is not the same as the user's task being done. An email can be accepted and never delivered, and a form can be created with a field missing. The agent can only tell the difference if the service gives it a way to check. Return an ID and a status the agent can look up later, and offer a way to read the result back, so the agent can confirm it matches the request. For batch calls, report which items failed and why, so a partial failure does not look like full success. For slow operations, return a job ID and a status endpoint and document the timeouts, so the agent waits instead of retrying blindly. An agent that

can verify its work is less likely to report false success to the user or to drop the service after one confusing result. Measure the task success rate as the share of agent tasks with a verified outcome, and the recovery rate as the share of errors followed by a successful retry in the same session.

Examples

Resend's sandbox needs no setup. `onboarding@resend.dev` sends to the account owner's own address before any domain is verified, and the quick-start uses `delivered@resend.dev` as a test recipient. The values are short and fixed, so they are easy to reuse and, plausibly, to remember.

Every code on Resend's errors page comes with a suggested action. The 403 for an unverified sender, for example, says to verify a domain and change the `from` address, and links a guide.

Resend uses Domain Connect to add DNS records automatically on Cloudflare and Vercel. In its changelog of Feb 6, 2025, Resend reported from its own data about 20% fewer support tickets and a median verification time of 5.19 seconds.

Resend's idempotency keys return the original response for a repeated request and a 409 error when the payload has changed, so the agent can retry without sending duplicates. Its MCP log tools return full request and response bodies, according to its MCP docs, so the agent can debug failed calls itself.

Tally's assistant can create and edit forms but cannot delete forms or submissions, as its MCP page states. A low worst case makes users comfortable approving access.

To do

- [ ] Offer a sandbox or test mode that works before any setup.
- [ ] Give every error a code, a message, a suggested action and a docs link.
- [ ] Return IDs, statuses and a way to read results back, so agents can verify outcomes.
- [ ] Report per-item results in batch responses, and return job IDs with documented timeouts for slow operations.
- [ ] Accept idempotency keys on every endpoint that writes data.
- [ ] Expose request logs through the API and MCP server.
- [ ] Automate your hardest setup step, for example with Domain Connect.
- [ ] Measure time to first value, the task success rate and the recovery rate.
- [ ] Make the edge return JSON errors on API and MCP addresses, with 429 and `retry-after` for rate limits.

Stage 5: Usage growth

FUNNEL STEP Be preferred

GOAL Become part of the user's daily work. Revenue comes from two moments, hitting a limit and moving into new uses.

What the agent does

The integration runs in production, calling the API on every signup or order. When the user asks a new question, such as "how many emails bounced this week?", the agent reuses the existing connection, because it is already approved and proven. As usage nears a limit, the agent sees it in response headers or a usage tool and can warn the user. At the limit, it tells the user what to buy and why, since payment is a human decision. Once the user approves, the new plan should take effect at once so the work continues. Over time the agent moves into nearby features, such as receiving email or newsletters, because the same connection already covers them.

Pay-as-you-go overage keeps an agent's work running through a usage spike. The quota moment is likely a strong sales moment for agent-originated accounts (hypothesis), because the agent can explain the need and its price in one message. Payment protocols based on HTTP 402 let agents pay per request, though for subscriptions a checkout link or a command-line upgrade is the practical path today.

Agent traffic also changes your costs. Retries, polling for status, repeated tool calls inside one task and automated jobs on the free plan all add requests that may never turn into revenue. Idempotency keys stop retries from repeating work. Status webhooks reduce polling, and rate limits and free-plan caps bound the worst case. Spending alerts and caps that users can set make costs predictable for both sides. Track infrastructure and support cost per active agent account next to revenue, so a rise in calls does not hide a loss.

What decides it

An installed connector stays across sessions, and scheduled agents call the service on their own. Broad MCP coverage keeps daily tasks in the chat. The service needs reliability, predictable pricing, clear quota handling and upgrades that skip the dashboard. The user weighs satisfaction, team adoption, switching costs and budget at each upgrade. Usage keeps a service in play independently of memory, and it can produce public code and reviews that help the earlier stages. It drops when a limit fails silently or when upgrades only work in the dashboard. Thin MCP coverage also sends users back to the dashboard.

As usage grows, scheduled agents and production integrations send steady, high-volume traffic that bot protection can mistake for abuse. Identify this traffic by API key or OAuth client, and set edge rate limits per key; a user agent alone is easy to misread.

Examples

Resend lets the agent upgrade from the terminal. `stripe projects upgrade resend/free resend/pro --json` takes named plans and returns JSON; with no payment method on file it returns a Checkout link, and the higher quota starts once payment is confirmed, as the guide describes. Through Stripe Projects, only

the Free and Transactional Pro plans are offered for now.

Resend responses carry `x-resend-daily-quota` (Free plan only, the only plan with a daily cap) and `x-resend-monthly-quota`. Quota errors name the fix, which is to upgrade or wait for the midnight UTC reset, according to its usage limits page.

Resend also added a structured usage endpoint. Its changelog of Oct 1, 2026 introduced `GET /usage`, which returns usage and limits for every quota in one request, including the team's rate limit. The same data is available through `resend usage --json` in the CLI and the MCP server's `get-usage` tool, so an agent can check its remaining quota before it hits a limit. The endpoint needs a full-access API key.

Resend's paid plans can keep sending past quota once pay-as-you-go is enabled in team settings, at the plan's overage rate ($0.90 per 1,000 emails on Pro, less on larger Scale tiers, as of Oct 22, 2026), according to pricing.md.

Resend sells inboxes for teams and for agents, with agent access through the API and MCP server. Extra inboxes cost $1 a month on paid plans with pay-as-you-go (as of Oct 22, 2026), and inboxes are in beta. The MCP server also covers newsletters, contacts, automations and webhooks, so sorting replies and similar tasks become daily agent work.

Tally's MCP server reads form submissions, so the assistant can calculate NPS or compare completion rates, as its MCP page shows. Every such question becomes a call to Tally.

To do

- [ ] Return quota usage in response headers, and offer a usage endpoint or MCP tool that reports usage against every limit.
- [ ] Add structured fields to quota errors, naming the required plan with its price and an upgrade address.
- [ ] Offer pay-as-you-go so work does not stop at the quota.
- [ ] Let agents upgrade with one human confirmation, through the API or a checkout link.
- [ ] Expose plan, usage and invoices as MCP tools.
- [ ] Cover your ten most common dashboard tasks in MCP.
- [ ] Let users set spending alerts and caps, and track cost per active agent account.
- [ ] Get legal advice on recording consent for subscriptions an agent starts.
- [ ] Set edge rate limits per API key or OAuth client, and review bot-protection logs for blocked customer traffic.

Measurement

Users who arrive through an agent often never visit your marketing site, so web analytics files them under "direct" or "unknown". Track agent-originated accounts from server logs and OAuth client IDs, and fill gaps with data from provisioning platforms. Add "AI assistant (ChatGPT, Claude, Gemini, Cursor)" to the "How did you hear about us?" question at signup, which catches users an agent recommended who then signed up in a browser. Record a baseline in the first week.

Start with five numbers

1. Unprompted recommendation rate and selection rate: from the same fixed test prompts, the share that name you, and among prompts where several services fit, the share where the agent picks yours. Run each prompt several times, because answers vary, and record the exact prompt with the model, the client, the date and whether search was on. Before testing, define which services count as suitable for each prompt and how a selection is judged, keep prompts and conditions fixed, and have a person review suitability and outcomes where they are unclear.
2. Authorization completion rate: agent-started OAuth or provisioning flows that end with working credentials.
3. Time to first value: from issuing credentials to the first working result, such as a delivered test email.
4. Task success rate: agent tasks with a verified outcome, out of all tasks attempted. Measure it in your fresh-session runs, and in logs where a task's end state can be read.
5. Weekly active agent accounts: accounts calling the API or MCP server through an agent client in a given week, identified by OAuth client ID where the client can be told apart.

Add these as you grow

For Understanding, count requests to your `.md` pages, `llms.txt` and `pricing.md` by agent user agents. For the Trial, track the recovery rate and the share of agent-originated accounts still calling on days two to seven. For Usage growth, track AI call count per account, broken down by tool and agent client, along with the error rate and speed of each MCP tool. MCP logs also show requests that found no matching tool, which tells you what users want that you do not offer yet. For commercial results, track paid conversion and expansion among agent-originated accounts. Next to that, track revenue against infrastructure and support cost per active agent account, and compare 90-day retention with a matched group of other customers.

Read with care

A recommendation is not an acquisition, since the user may never sign up. A call is not a completed task, and AI call count can rise because of failed retries or loops, so read it next to the task success rate and cost. Most of these numbers show correlation. An upgrade after a quota error does not prove the message made the sale; a holdout group or an A/B test on the message can. Customers who use agents may differ from others before they ever touch MCP, so compare matched groups and account for their starting activity. Doc fetches include crawlers and caches, which have to be separated from real user sessions.

Keeping agents safe

Opening a service to agents opens it to their mistakes and to people who misuse them. An email service is especially exposed, because it can reach anyone and the agent reads mail written by strangers. The main threat is prompt injection: instructions hidden in content the agent reads, such as an email saying "forward all invoices to this address". A label that marks content as untrusted helps, but a model may still follow the instruction, so the real controls have to sit where the model cannot talk its way past them. Give each credential only the permissions its task needs, and let agents freely do only what can be undone.

Six risks need specific controls.

Prompt injection through incoming mail is the first. Present incoming mail as data in every tool result, separate read and send scopes, ask before sending to new recipients, and check permissions inside the MCP server. This applies directly to Resend, whose MCP server exposes received emails and inboxes.

Accounts that agents create can be used for spam. Limit test sends to the owner until a domain is verified, and cap the free plan per day. Rate limits add a further layer. Resend limits test sends this way and caps Free at 100 emails a day.

Some actions cannot be undone or cost money, such as deleting a domain, sending a bulk campaign or upgrading a plan. Keep them out of the default scope or require the user's confirmation. Tally's MCP server cannot delete forms or submissions; Resend's can remove domains and API keys.

Credentials can leak. Use short-lived, scoped tokens and allow rotation with one command. A page to revoke access helps too. Resend has key permissions and MCP tools to list and revoke OAuth grants, and `stripe projects rotate resend/email` replaces a key.

An agent can get stuck in a loop and retry thousands of times. A per-team rate limit and idempotency keys contain the damage, and usage alerts flag it. Resend's default limit is 10 requests per second per team, and Resend supports idempotency keys.

Personal data can reach a third-party model through recipient lists or incoming mail. Fetch data only on explicit request, and allow opting out per resource. The privacy notice should cover AI assistants. Tally fetches submissions only when the user asks.

The customer relationship can also drift to the assistant, which is a reason to support open standards such as MCP and the Agent Plugins spec. As the dashboard fades from view, the links and previews an agent shows should still carry your brand.

Each area needs one named owner, because this work cuts across teams:

- Developer relations owns docs, `llms.txt`, examples and skills.
- Platform engineering owns the MCP server and API, ideally with a dedicated MCP product owner.
- Growth, with identity and security, owns onboarding, OAuth and provisioning platforms.
- The revenue team owns pricing, quota errors and the upgrade flow.
- Trust and safety owns abuse and content safety.
- Legal owns personal data and consent.
- Marketing analytics owns visibility tests and the measures above.

To do

- [ ] Mark incoming content as untrusted, and enforce permissions in the server.
- [ ] Move delete operations out of the default MCP scope.
- [ ] Add per-account usage alerts for agent loops.
- [ ] Publish a connected-apps page with one-click revoke.
- [ ] Update the privacy notice to cover access by AI assistants.
- [ ] Put a name next to each area of ownership.

Serving business customers

A developer can approve an agent connection alone; at a company, an admin or security team usually approves it first. The useful question is which requirements block that approval on day one, and which can wait.

Needed from the start

Some needs appear as soon as a whole team works with the service through agents. Audit logs should record each action an agent took and through which client, which OAuth client IDs make possible, and users should be able to cancel an action an agent has started. Costs should be predictable, with documented rate limits and spending alerts. API and MCP changes need a versioning and deprecation policy, because agents keep writing the calls they learned, and renaming or removing an MCP tool breaks every saved workflow that uses it. Announce changes in the changelog and in error messages, and keep old versions working for a stated period.

Needed as enterprise use grows

Larger customers ask for single sign-on and organization-level control over which agents and clients may connect. Admins want approval workflows for spending and for connecting new tools. Compliance teams ask where data is stored and processed and how long it is kept. They also ask which certifications apply. Build these when sales conversations start to stall on them.

Agent clients differ as well. They vary in which OAuth flows they support and in how they ask the user for permission, and some present tools to the model differently. Test the main flows in each client your customers use as part of the scorecard runs.

Resend's pricing.md shows one way to split these needs across plans. API key permissions, SOC 2 Type II and GDPR compliance, and a choice of sending region come with every plan. Single sign-on is an add-on on Scale and included on Enterprise. Enterprise adds a 99.99% uptime SLA and its own rate limits. Data retention is flexible there, against 30 days on self-serve plans.

To do

- [ ] Log every agent action with the OAuth client that made it.
- [ ] Publish a versioning and deprecation policy for your API and MCP tools.
- [ ] Test sign-in, permissions and tool use in each agent client your customers use.
- [ ] Decide which enterprise controls to build next, based on the deals that stall without them.

The agent-readiness scorecard

The scorecard below turns the five stages into checks you can score and measures you can track, so you can see where you stand and what moved. A score holds only for the model versions and client releases you tested, so rerun it when either changes.

STAGE	CHECKS TO SCORE	MEASURES	HOW TO TEST	HOW OFTEN
Discovery (Be discoverable)	The To do list in Stage 1	Unprompted recommendation rate	30 fixed prompts, each run several times in each assistant, with model, client, date and search setting recorded	Monthly
Understanding (Be remembered)	The To do list in Stage 2	Doc, `llms.txt` and `pricing.md` requests by agent user agents; selection rate	Fetch your docs index and `pricing.md` with JavaScript off and with agent user agents; ask a fresh agent to price your service for a sample budget; score the test prompts where several services fit against eligibility criteria set before the test	Checks monthly, measure weekly
Transaction (Be usable)	The To do list in Stage 3	Authorization completion rate; human steps from zero to working credentials	Fresh-session test in Claude Code or Cursor, counting every stop for a person	Monthly, and after any change to signup or auth
Trial (Be valuable)	The To do list in Stage 4	Time to first value; task success rate; recovery rate	Fresh-session test, plus triggering common errors and checking that the agent can verify the final outcome	Weekly for measures, monthly for checks
Usage growth (Be preferred)	The To do list in Stage 5	Weekly active agent accounts; AI call count; paid conversion; cost per active agent account; 90-day retention	Ask an agent to check usage, hit a quota on a test account and upgrade without opening the dashboard	Weekly, with retention quarterly
Safety and business customers	The To do lists in those two sections	Share of agent actions logged with the client that made them; deals that stall on missing controls	Review audit logs and run the main flows in each agent client your customers use	Quarterly

How to run it

Each stage's To do list is its set of checks. Copy the items into a shared sheet, one row per check. Give each row a score, an evidence link, an owner, the date it was last checked and the next action.

The fresh-session test gathers most of the evidence. Open a new Claude Code or Cursor session with no earlier context, and use a task from your own category. For an email service, ask without naming your service: "Add transactional email to this app; free if possible." The answer mostly reflects memory and search. Then ask "Do it with [your service]," which tests docs and onboarding. Count the human interventions and the minutes to the first working result.

Score each check on three levels. Give 2 when an agent completes it without human help and you have seen this happen in a fresh session. Give 1 when it exists but needs a human step or works only in some clients. Give 0 when it is missing. Every score needs evidence, such as a link or a screenshot from the test run, so a later reviewer can check it.

Score everything in the first week and record the measures at the same time; this is your baseline. Then keep a fixed rhythm. Read the measures weekly, rescore the checks monthly and run the full fresh-session test after any release that touches docs, signup, pricing or errors. Once a quarter, review the whole card and the retention numbers.

Add up each stage's points as a share of its maximum. The weakest stage limits the whole path, because an agent that cannot get past one stage never reaches the next, so put the next quarter's work there first. When two pieces of work compete, prefer the one that raises scores in more stages; a hosted MCP server, for example, helps all five. Each 0 or 1 should carry a named owner and a next action, taken from the ownership list in the safety section.

Keep the test conditions stable so scores stay comparable. Use the same prompts and test accounts, and note the clients and model versions, because a new model can change results with no change on your side. Score two or three competitors with the same card each quarter as well. Agents compare services side by side, so your position against them matters as much as your own score.

Where to start

The quick wins take days to weeks: publish a `pricing.md`, serve docs as markdown, give every error a suggested action, and add "AI assistant" to your signup question. The longer-term investments take a quarter or more: a hosted MCP server with OAuth, a provisioning platform, outcomes agents can verify, and MCP coverage of daily tasks.

A worked example: Resend, checked Oct 20 to 22, 2026

Scored with this card, Resend is strong on what it controls in the first four stages and partial in Usage growth. In Discovery it has an install page per client and listings in Claude and Codex. Its unprompted recommendation rate was not tested, and its listing in the official MCP Registry could not be confirmed, since the official resend-mcp repository has no registry metadata. In Understanding it scores well except for a JSON pricing file. In the Transaction, an agent can get a Resend account without a signup form only through Stripe Projects or the Vercel Marketplace; everywhere else, a person signs up first. In the Trial, test emails reach only the owner until a domain is verified. In Usage growth, quota errors carry no plan or price fields, and the MCP server reads usage but cannot change plans or show invoices. On safety, domains and API keys can still be deleted through MCP.

Glossary

- An agent is a language model working in a loop with tools, acting for a user.
- SaaS (software as a service) is software reached over the internet through a website, docs, an API or an MCP server, such as Resend and Tally. This guide often calls a SaaS product a service.
- The agent funnel is the five steps a service has to pass with an agent: Be discoverable, Be remembered, Be usable, Be valuable and Be preferred. Each step matches one stage of this guide: Discovery, Understanding, Transaction, Trial and Usage growth.
- Be remembered means staying on the agent's shortlist after it evaluates your current documentation and pricing. It is separate from the model's training memory, the session context, project files and persistent client connections.
- MCP (Model Context Protocol) is an open protocol that lets AI assistants call external tools and read data in a standard way. A hosted MCP server is one you connect to by address, with nothing to install.
- A skill is a packaged set of instructions an agent loads for a specific task.
- The training cutoff is the date after which a model has seen no data.
- The context window is the amount of text an agent can hold at once.
- `llms.txt` is a file at the root of a site that lists its main docs pages for language models; `llms-full.txt` holds all the docs in one file.
- OAuth is a standard for giving an app limited access without sharing a password. A scope is one specific permission carried by an OAuth token.
- Device flow is OAuth for tools without a browser, using a link and a code.
- A Bearer token is a key sent with each request, created in advance by a person.
- A provisioning platform creates accounts and keys at many other services from one identity.
- An idempotency key makes a repeated request run only once.
- SPF, DKIM and DMARC are DNS records that help receiving providers check whether email from a domain is authorized.
- Domain Connect is a standard that lets a service add DNS records at a supporting provider after approval.
- Time to first value is the time from getting credentials to the first working result.
- AI call count is the number of API and MCP calls per account, broken down by tool and agent client.
- Prompt injection means hidden instructions in content an agent reads.
- Unprompted recommendation rate is the share of fixed test prompts that do not name you but still get you recommended.
- Selection rate is the share of test prompts, among those where several services fit by criteria set before testing, in which the agent picks yours.
- Authorization completion rate is the share of agent-started OAuth or provisioning flows that end with working credentials.
- Task success rate is the share of agent tasks with a verified outcome.
- Recovery rate is the share of errors followed by a successful retry in the same session.

- An agent client is the app or editor an agent runs in, such as Claude Code or Cursor; this guide often shortens it to "client". Together with its installed connectors and settings, it forms the agent environment.
- An agent-originated account is one an agent created or recommended; an active agent account is one that called your API or MCP server through an agent client in a given week.
- A fresh-session test is a test run in a new Claude Code or Cursor session with no earlier context, counting the human steps and the time to the first working result.

Appendix: verification notes

All pages were opened on Oct 20, 2026, and Resend's pricing.md was opened again on Oct 22, 2026. "Documented" means the vendor's page states it, and "company-reported" means a number a vendor published about itself. The rate limit and quota headers were also checked against a live API response, and two claims were checked in Resend's open-source code. Prices, headers and tool lists change often, so re-check them before publishing.

CLAIM	SOURCE	STATUS
Resend MCP calls: 106,719 in April 2026, 1,062,650 in August 2026	Post on X by Zeno Rocha, Aug 28, 2026: https://x.com/zenorocha/status/2093355692484854138	Company-reported
Free: 3,000 emails a month, 100 a day; Pro from $20 a month for 50,000; overage $0.90 per 1,000 on Pro, less on larger Scale tiers, when pay-as-you-go is enabled	https://resend.com/pricing.md	Documented (rechecked Oct 22, 2026)
Every plan includes 3 inboxes; paid plans with pay-as-you-go add more at $1 per inbox per month; Free stops at 3	https://resend.com/pricing.md (Inboxes section and FAQ)	Documented; vendor price list, not tested by purchase
Upgrading a self-serve plan does not raise the API rate limit; support reviews requests for higher limits; Enterprise lists its own rate limits	https://resend.com/pricing.md (FAQ and Enterprise features)	Documented (Oct 22, 2026)
API key permissions, SOC 2 Type II, GDPR compliance and a choice of sending region on every plan; SSO as a Scale add-on and included on Enterprise; 99.99% uptime SLA on Enterprise; 30-day data retention on self-serve plans, flexible on Enterprise	https://resend.com/pricing.md (Feature highlights and plan comparison)	Documented (Oct 22, 2026)
Inboxes are in beta	https://resend.com/docs/mcp-server	Documented
Default rate limit of 10 requests per second per team, with `ratelimit-*` and `retry-after` headers	https://resend.com/docs/api-reference/rate-limit	Documented; observed live (ratelimit-limit 10, ratelimit-policy 10;w=1)

CLAIM	SOURCE	STATUS
`x-resend-daily-quota` (Free plan only) and `x-resend-monthly-quota` headers	https://resend.com/docs/api-reference/rate-limit (Email Quotas section)	Observed live on Oct 20, 2026 in an HTTP 200 response to POST /emails (one request, one account; other plans not tested). Absent from the Node SDK's recorded responses (github.com/resend/resend-node, commit 7943103), likely recorded on another account or plan
Quota errors: 429 `daily_quota_exceeded` and `monthly_quota_exceeded`, with an upgrade link	https://resend.com/docs/api-reference/errors	Documented
Idempotency: 409 `invalid_idempotent_request` when the payload changes	https://resend.com/docs/api-reference/errors	Documented
Hosted MCP server at mcp.resend.com with OAuth, and a Bearer key fallback	https://resend.com/docs/mcp-server	Documented
MCP tools include `get-usage`, `remove-domain`, `remove-api-key`, `list-oauth-grants` and `revoke-oauth-grant`, with no plan-change or invoice tools	https://github.com/resend/resend-mcp (official source); https://glama.ai/mcp/servers/resend/resend-mcp/tools/get-usage (listing)	Verified in the official source at commit d1e33d5 (Oct 16, 2026): a usage module, no billing, invoice or plan-change module. Covers the MCP server only; plans can still be changed in the dashboard and through Stripe Projects. Not tested on a live connection. The Usage API behind it (GET /usage, full-access key) was announced on Oct 1, 2026 and is documented without a beta label as of Oct 22, 2026: https://resend.com/docs/api-reference/usage/retrieve-usage
Official MCP Registry listing	registry.modelcontextprotocol.io (could not be reached from our test environments)	Unconfirmed; the official resend-mcp repository has no server.json or mcpName
In Resend's standard onboarding, a person creates the account. The exceptions are Stripe Projects, which creates or links the account, and the Vercel Marketplace, where Vercel manages the Resend team and the user needs a Vercel account with a project and a domain bought through Vercel	https://resend.com/docs/ai-onboarding; https://resend.com/changelog/vercel-marketplace-integration (prerequisites, Jun 30, 2026); https://resend.com/docs/knowledge-base/vercel (Vercel Marketplace installs: Vercel manages the Resend team, API keys and domains; checked Oct 22, 2026)	Documented

CLAIM	SOURCE	STATUS
Stripe Projects: `add`, `upgrade --json`, a Checkout link when no payment method is on file, key rotation, and only the Free and Transactional Pro plans (rechecked Oct 22, 2026)	https://resend.com/docs/guides/stripe-projects-integration	Documented
Domain Connect: about 20% fewer support tickets, 5.19 s median verification	https://www.resend.com/changelog/new-domains-workflow	Company-reported (Feb 6, 2025)
Tally's MCP server is free on all plans, recommends OAuth and cannot delete forms or submissions	https://tally.so/help/mcp	Documented
Tally comparison page; Tally listed as a one-click connector in the Claude and ChatGPT directories	https://tally.so/help/best-mcp-form-builders (updated Oct 9, 2026)	Self-published by Tally; not independent validation

Evidence gaps and open questions

These claims in the guide rest on reasoning or vendor reports and have not been independently established. Each is worth testing before you rely on it.

- How models rank services is not public. The guide describes only the signals that reach the agent.
- The memory loop, where public code from successful integrations makes a service more likely to be suggested by later models, has not been measured.
- That agents may be more likely to continue with the first service that works is a hypothesis, untested here.
- That the quota moment is a strong sales moment is a hypothesis.
- Which advantages last (platform defaults, a record of first-try success, daily-task coverage) is reasoned and has not been measured.
- How much past outcomes, project files and memory features influence later selection varies by client and is not documented in a general way.
- The selection rate from test prompts is a proxy; it depends on the prompts, the client and whether search was on.
- Resend's listing in the official MCP Registry is unconfirmed.
- Tally's comparison page is Tally's own, so it shows how Tally positions itself; it says nothing about how agents rank form builders.