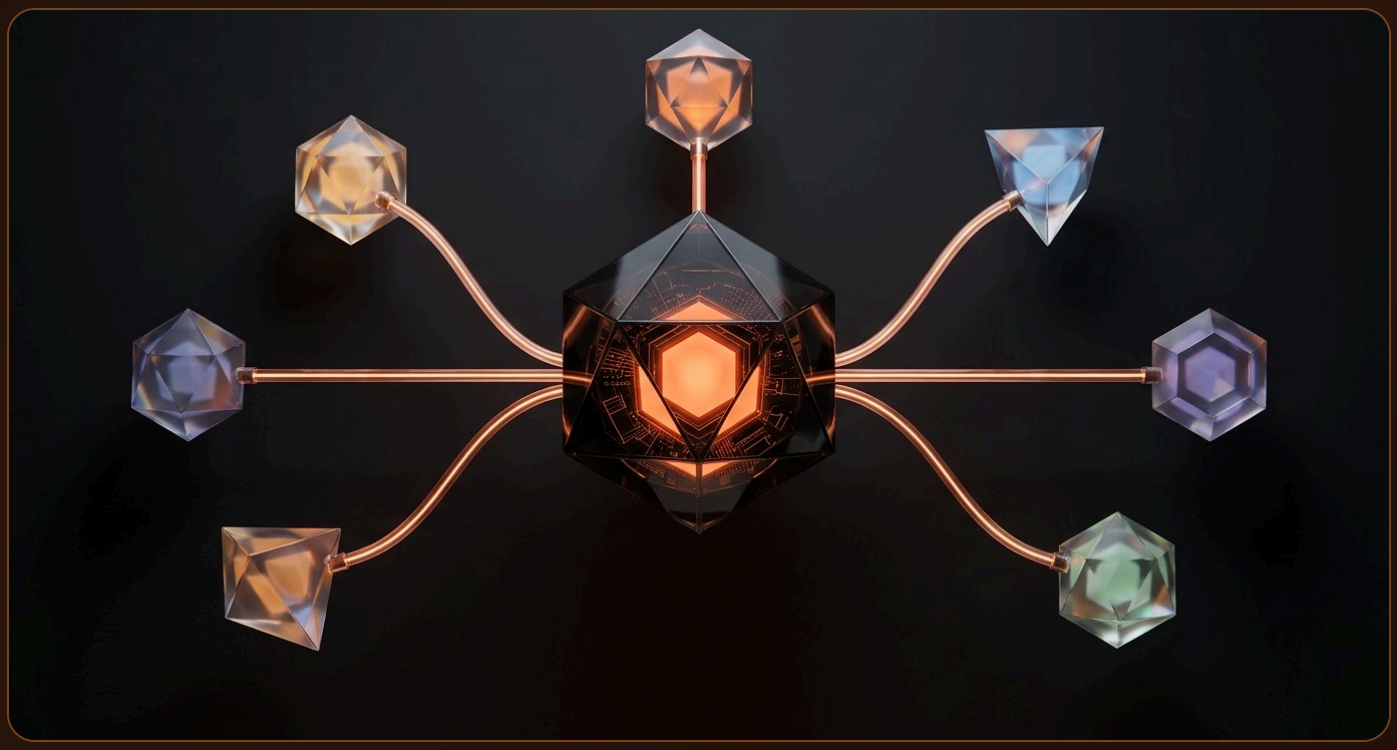
VERCEL & FULL-STACK AI APPLICATIONS • FIELD GUIDE

Multi-Platform Bots With Vercel Chat SDK

Build one unified TypeScript bot codebase that speaks native Slack, Microsoft Teams, Discord, Telegram, Google Chat, GitHub, Linear, and Liveblocks—backed by Redis thread locks, AI SDK 7 `ToolLoopAgent`, Vercel Connect OAuth, durable Workflow 5 approval gates, and isolated Vercel Sandboxes.

Unified Chat Class | `@chat-adapter/catalog` | Slack, Teams & Discord | Telegram, GitHub & Linear | Vercel Connect OAuth | HITL Cards & Sandbox



Write Bot Logic Once

Handle `@`-mentions, subscribed threads, slash commands, and streamed AI replies in one shared handler across every platform adapter.

Zero-Secret Auth & Locks

Pair `@vercel/connect` short-lived OAuth tokens with Redis distributed thread locks so parallel messages never race or leak static keys.

Durable HITL & Sandboxes

Suspend **Workflow 5** runs on Slack or Teams approval cards via `createWebhook` and audit PR diffs inside **Vercel Sandbox** microVMs.

Write Bot Logic Once, Native UX Everywhere

Every chat platform invents its own webhook signature scheme, markdown dialect, thread model, and button payload. **Vercel Chat SDK** (`chat`) normalizes incoming events into one clean TypeScript API while rendering authentic native cards, modals, and streaming replies per channel.

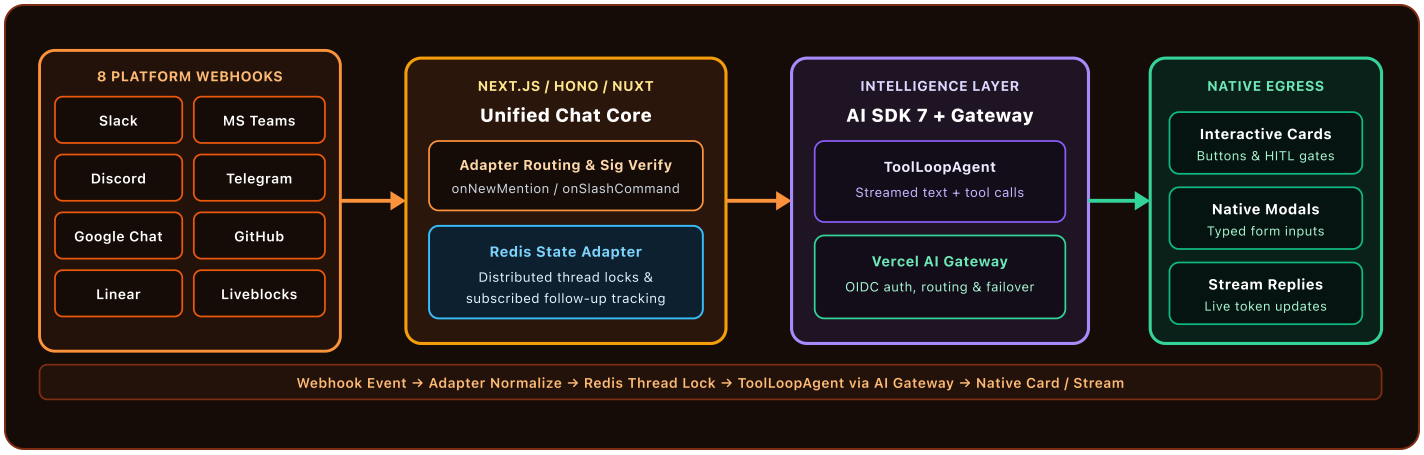
Quick Navigation

1. Unified Architecture & Mental Model p. 2
2. Quickstart, Catalog & Decision Matrix p. 3
3. Annotated Thread & Card Inspector p. 4
4. Unified Event Model & Adapters p. 5
5. Redis Locks & Handler Anatomy p. 6
6. Troubleshooting Clinic & Pre-Flight p. 12
7. Zero-Secret Connect OAuth & Blob Tools p. 7
8. HITL Workflow 5 Gates & Sandbox PRs p. 8
9. 5 Real-World Copy-Paste Recipes p. 9
10. Webhook Security, Fallbacks & Replay p. 10
11. Cross-Tool Synergy Pipeline p. 11
Golden Rule: Check Catalog Capabilities First All

Core Mental Model: Normalized Ingress, Native Egress

Instead of embedding Slack Block Kit JSON or Discord slash-command glue inside your AI agent, route every webhook endpoint into one shared `Chat` instance running in Next.js, Hono, or Nuxt. Adapters translate inbound webhooks into unified `Thread` and `Message` objects, Redis guards thread state against concurrent replies, and **AI SDK 7** streams grounded responses back out through each platform's native UI primitives.

Unified Architecture: 8 Webhook Channels Through One `Chat` Core



Mental Model Takeaway: Decouple Reasoning From Channel Plumbing

Your agent code never cares whether a question arrived from a Slack channel, a GitHub pull request comment, or a Microsoft Teams thread. Adapters handle platform auth, rate-limit formatting, and native card translation at the edge of your app.

Quickstart, `@chat-adapter/catalog` & Decision Matrix

Go from zero to a multi-platform bot in under a minute. Scaffold a Next.js template without prompts, inspect the bundled offline docs inside `node_modules/chat/`, and use `@chat-adapter/catalog` to query adapter capabilities statically.

1 60-Second Scaffold & Bundled Docs

```
# Scaffold a Next.js bot project without prompts
npx create-chat-sdk

# Or add Chat SDK + adapters to an existing repo
npm i chat @chat-adapter/slack @chat-adapter/redis
```

Every install ships version-matched markdown guides in `node_modules/chat/docs/`, 13 production recipes in `node_modules/chat/resources/`, and starter blueprints in `node_modules/chat/resources/templates.json`.

2 Adapter Catalog Discovery (`@chat-adapter/catalog`)

Replace deprecated `chat/adapters` imports with `@chat-adapter/catalog`—a zero-dependency static index of official, vendor-official, and community adapters:

- `listAdapters()` & `getCatalogEntry(slug)`: Discover npm packages, peer deps, and platform vs. state adapter roles before installing.
- `getFeatureSupport(slug)`: Check streaming, modals, and distributed locking support at build time or runtime.
- `listEnvVars(slug)` & `getSecretEnvVars(slug)`: Generate accurate onboarding checklists and secret vaults automatically.

Architectural Decision Matrix: Choosing Your Bot Foundation

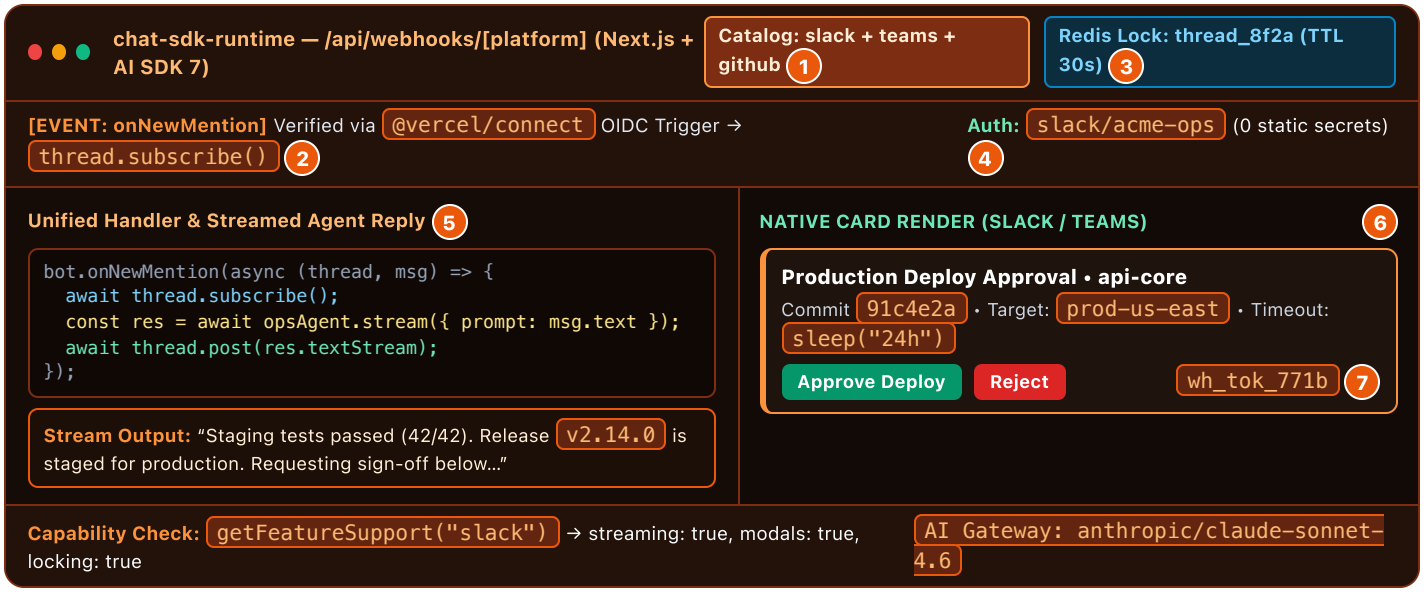
Dimension	Vercel Chat SDK (`chat`)	Raw SDKs (`@slack/bolt`, `discord.js`, `telegraf`)	`eve` Framework (Greenfield)
Primary Sweet Spot	Multi-platform bots inside Next.js, Hono, Nitro, or Nuxt apps sharing one handler codebase.	Single-platform edge cases needing esoteric socket-mode APIs outside standard messaging.	Greenfield, filesystem-first durable internal agents where Slack is one of several agent channels.
Channel Portability	**High:** Add Slack, Teams, Discord, Telegram, GChat, GitHub, Linear, or Liveblocks via adapters.	**Locked:** Rewriting a Slack Bolt bot for Teams or Discord means duplicating every route and UI card.	**Built-In:** Convention-based `agent/channels/*.ts` files wired straight to durable agent sessions.
Concurrency & State	Pluggable `@chat-adapter/redis` locks threads and persists subscriptions across serverless instances.	Manual DIY Redis locks; stateful WebSocket clients like `discord.js` break on serverless cold starts.	Durable filesystem-backed session history, schedules, and subagents managed by the runtime.
Workflow & Auth Pairing	Pairs cleanly with `@vercel/connect/chat`, AI SDK 7 `ToolLoopAgent`, and Workflow 5 `createWebhook`.	Requires manual OAuth refresh plumbing, custom webhook replay checks, and static bot tokens.	Native `@vercel/connect/eve` helpers (`connectSlackCredentials`) and Agent Runs observability.

When to Pick Chat SDK vs. `eve` Directly

Choose **Vercel Chat SDK** when you are adding a multi-platform bot or interactive webhook handler to an existing web app (Next.js, Hono, Nuxt) and want full control over routes, cards, and custom workflows. Pick **eve** directly when starting a greenfield, autonomous internal agent whose primary job is long-lived filesystem-first sessions, subagents, and scheduled Slack operations.

Annotated Tour: Cross-Platform Thread & Card Cockpit

Trace how a single user `@`-mention flows from an inbound webhook through Redis locking, AI SDK 7 streaming, and an interactive Human-in-the-Loop approval card that resumes a paused Workflow 5 run.



Interface Legend: The 7 Live Runtime Touchpoints

1. **Catalog Adapter Registry:** Verified adapter entries from `@chat-adapter/catalog` map each webhook route to its platform formatter and feature flags.
2. **Unified Event Hook & Subscription:** `onNewMention` fires on first ping; calling `thread.subscribe()` routes future thread replies to `onSubscribedMessage`.
3. **Redis Distributed Thread Lock:** Prevents overlapping serverless invocations from mutating the same thread history or posting out-of-order replies.
4. **Vercel Connect OIDC Auth:** Inbound webhooks arrive OIDC-signed by Vercel Connect while outbound posts fetch short-lived tokens dynamically.
5. **Streamed `ToolLoopAgent` Reply:** Pipes AI SDK 7 token streams directly into `thread.post()` using native live-edit updates where supported.
6. **Cross-Platform Card Renderer:** Compiles one declarative JSX/object Card into Slack Block Kit, Teams Adaptive Cards, or Discord Embeds automatically.
7. **Workflow 5 Resumption Token (`createWebhook`):** Button clicks carry an opaque token back to your handler, verifying the clicking user's authority and calling `resumeWebhook(token, payload)` to wake the suspended workflow run cleanly.

Unified `Chat` Event Model & Native Adapters

Every adapter translates platform-specific events into four clean lifecycle hooks, letting you write interactive streaming bots, slash commands, Cards, and Modals once while honoring per-platform capabilities from `@chat-adapter/catalog`.

1. Unified Event Hooks & Streaming Replies

```
import { Chat } from "chat";
import { createSlackAdapter } from "@chat-adapter/slack";
import { createTeamsAdapter } from "@chat-adapter/teams";
import { createRedisState } from "@chat-adapter/redis";

export const bot = new Chat({
  userName: "ops-copilot",
  adapters: { slack: createSlackAdapter(), teams:
createTeamsAdapter() },
  state: createRedisState(),
});

bot.onNewMention(async (thread, msg) => {
  await thread.subscribe();
  const stream = await supportAgent.stream({ prompt:
msg.text });
  await thread.post(stream.textStream); // Streams natively
per adapter
});

bot.onSubscribedMessage(async (thread, msg) => { /* follow-
up turn */ });
bot.onSlashCommand("/triage", async (event) => { /* modal or
card */ });
```

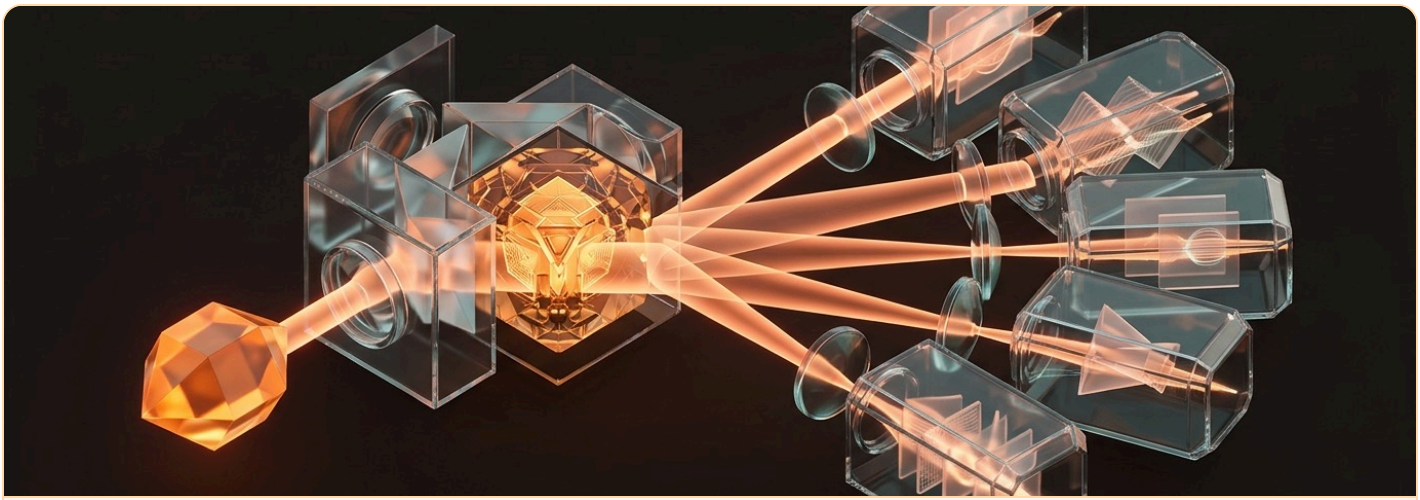
2. Capability Checks via `@chat-adapter/catalog`

Adapter	Streaming	Modals	Primary Surface
Slack / Teams	Live Edit	Native	Block Kit / Adaptive
Discord	Chunk Edit	Native	Embeds & Slash
Telegram	Draft / Edit	Inline Buttons	Commands & Topics
GitHub / Linear	Final Post	Markdown Links	PR & Issue Threads

Call `getFeatureSupport(slug)` before invoking modals or token streaming so your bot degrades gracefully on issue-tracker threads without throwing runtime errors.

Declarative Cards & Modals Compile to Native Platform Components

Use Chat SDK's built-in `Card`, `Button`, and `Modal` helpers instead of raw JSON payloads. On Slack they render as interactive Block Kit; on Microsoft Teams they emit Adaptive Cards; on GitHub or Linear they automatically render clean Markdown summaries with action links.



The Adapter Switchboard: One structured message crystal enters the central switchboard, passes through a shared distributed lock node, and fans across precision adapter lenses into distinct native channel chambers without duplicating bot logic.

Redis Thread Locks & Multi-Platform Handler Anatomy

Serverless webhook functions run concurrently across regions. Without a shared state layer, two rapid messages in the same thread overwrite each other's context and post scrambled replies. Pairing `@chat-adapter/redis` with durable Workflow handoffs keeps every conversation deterministic.

1 Redis Subscriptions & Distributed Locks

- **Thread Subscriptions:** Calling `await thread.subscribe()` stores the active thread ID in Redis so users can reply naturally without re-typing `@bot` every turn.
- **Per-Thread Mutual Exclusion:** Redis distributed locks serialize concurrent webhook deliveries on the same thread key, protecting message history and tool side-effects.
- **Zero Cold-Start Drift:** Any Vercel Function instance can pick up the next turn seamlessly using shared Redis state.

2 Never Poll With `setInterval` in Webhooks

- **Fast Webhook Ack:** Messaging platforms retry or drop webhooks that stall past 3 seconds.
- **Avoid `setTimeout` / `setInterval`:** Polling loops burn function timeouts and vanish on redeploys.
- **Offload Long Jobs:** Acknowledge the mention, post a streaming status message, and trigger a durable **Workflow 5** run via `start()` for multi-minute builds or approvals.

Color-Coded Anatomy of a Production Multi-Platform Handler

[1. Event Trigger] [2. Redis Lock & State] [3. ToolLoopAgent + Gateway] [4. Native Stream / Card] [5. Fallback & Stop Guard]

Listen on bot.onNewMention & bot.onSubscribedMessage after adapter signature verification succeeds. Acquire the Redis thread lock, load recent thread messages, and mark thread.subscribe() active. Invoke new ToolLoopAgent({ model: "anthropic/claude-sonnet-4.6", tools, stopWhen: isStepCount(6) }) via AI Gateway. Stream textStream into thread.post() and append an interactive Card when human approval is needed. Check getFeatureSupport(thread.adapterName) before opening modals; release the Redis lock in finally {}.

3-Phase Execution Lifecycle Across Serverless Invocations

Lifecycle Phase	Primary Mechanism	Why It Matters in Production
1. Ingress & Lock	Adapter Verify + Redis Lock	Rejects forged payloads immediately and queues duplicate Slack/Teams retries behind a single thread mutex.
2. Reason & Stream	`ToolLoopAgent` + AI Gateway	Routes models via OIDC with automatic provider failover while streaming incremental tokens back to the thread.
3. Pause & Resume	`Card` Action + Workflow 5	Converts multi-hour human sign-offs into zero-compute durable suspensions resumed via `resumeWebhook()`.

Pro Tip: Bound Agent Loops With `stopWhen: isStepCount(N)`

Never rely on legacy `maxSteps` (removed in AI SDK 5+). Pass `stopWhen: isStepCount(6)` to `ToolLoopAgent` so runaway tool calls terminate predictably before exhausting platform response windows.

Zero-Secret Auth with `@vercel/connect` & Blob Tools

Static long-lived bot tokens sitting inside `.env` files create rotation headaches and wide blast radii. Pair Chat SDK with **Vercel Connect** (`@vercel/connect/chat`) for short-lived OAuth tokens and OIDC-signed triggers, and equip agents with approval-gated **Vercel Blob** tools.

Stop Copying Static `SLACK_BOT_TOKEN` & `GITHUB_APP_PRIVATE_KEY` Secrets

With **Vercel Connect**, your deployment authenticates using its ephemeral `VERCEL_OIDC_TOKEN`. Connect holds refresh credentials in its managed vault and mints short-lived `app` or per-`user` provider tokens strictly at request time.

How `@vercel/connect/chat` Works

- **App vs. User Subject:** Request `subject: { type: "app" }` to post as the bot, or `subject: { type: "user", id }` to act strictly with an individual user's OAuth consent.
- **Signed Trigger Forwarding:** Connect verifies inbound Slack, Teams, Discord, GitHub, and Linear webhooks and forwards them to your route as signed Vercel OIDC requests.
- **Native Exceptions:** Telegram and Notion continue using their native inbound webhook verification tokens alongside Connect outbound calls.

2 Approval-Gated Vercel Blob / Files SDK Tools

- `createFileTools` **Factory:** Expose Vercel Blob buckets directly to `ToolLoopAgent` so Slack users can list, search, and read reports conversationally.
- **Read-Only vs. Write Modes:** Allow safe read/list calls automatically while placing uploads and deletes behind explicit human approval.
- **Scoped Paths:** Restrict prefixes per channel or tenant (e.g. `reports/eng/`) so bots never traverse unrelated buckets.

```
// 1. Provision & link connector via CLI: vercel connect create slack && vercel connect attach slack/acme-ops
import { getToken } from "@vercel/connect";
import { ToolLoopAgent, isStepCount } from "ai";
import { createFileTools } from "@vercel/blob/ai"; // Approval-gated Files SDK tools

const slackToken = await getToken("slack/acme-ops", { subject: { type: "app" } });
const fileTools = createFileTools({ prefix: "team-docs/", readOnly: false, requireWriteApproval: true });

export const docsAgent = new ToolLoopAgent({
  model: "openai/gpt-5.4", // Routed via AI Gateway with VERCEL_OIDC_TOKEN
  tools: fileTools,
  stopWhen: isStepCount(6),
});
```

Local Dev Tip: Refresh Ephemeral OIDC Tokens With `vercel env pull`

On Vercel deployments, `VERCEL_OIDC_TOKEN` rotates automatically. Locally, link your directory with `vercel link` and run `vercel env pull` whenever your short-lived development OIDC token expires.

HITL Approval Cards & `Sandbox` PR Reviewers

Combine Chat SDK with **Workflow 5** to pause multi-day approvals at zero compute cost until someone clicks a button in Slack or Teams, and spin up **Vercel Sandbox** microVMs whenever a developer `@`-mentions your bot on a GitHub pull request.

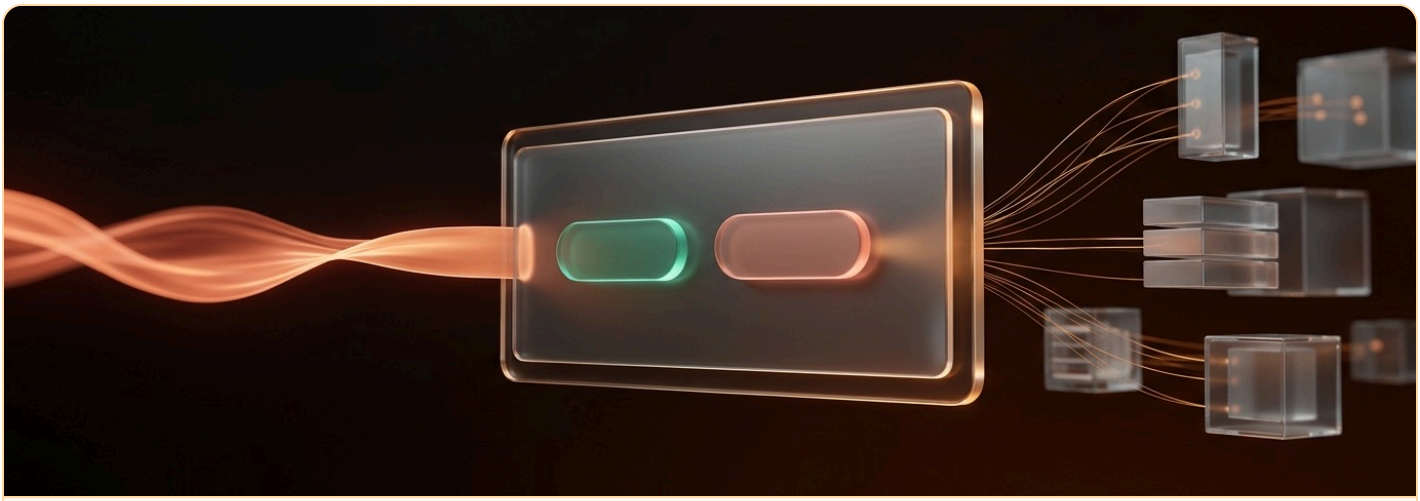
1 Suspending Workflow 5 on Chat Buttons

- **Create Resumable Gate:** Call `const hook = createWebhook()` inside a `"use workflow"` function and pass `hook.token` into your Chat SDK approval `Card` buttons.
- **Race With Durable Timeouts:** Pair `Promise.race([hook, sleep("24h")])` so stale deployment requests expire cleanly without holding any server memory.
- **Validate Approver Identity:** Inside `bot.onAction`, verify `event.user` belongs to the authorized release group before calling `resumeWebhook()`.

2 GitHub PR Reviews in Vercel Sandbox

- **Trigger on PR Mention:** `@chat-adapter/github` captures `@bot review` comments on any PR thread.
- **Isolated Firecracker microVM:** Launch `await Sandbox.create({ networkPolicy: "deny-all" })` via `@vercel/sandbox@2`, clone the PR branch, and run tests safely.
- **Grounded PR Feedback:** Feed exact diff hunks and failing test output to `ToolLoopAgent` and post crisp inline comments back to the PR.

```
// Workflow 5 HITL gate: suspends cleanly until Slack/Teams button click or 24h durable sleep()
import { createWebhook, sleep } from "workflow";
export async function deployApprovalWorkflow(releaseTag: string, channelId: string) {
  "use workflow";
  const approvalHook = createWebhook();
  await postApprovalCardStep(channelId, releaseTag, approvalHook.token);
  const decision = await Promise.race([approvalHook, sleep("24h").then(() => ({ approved: false, reason: "timeout" }))]);
  if (decision.approved) await promoteProductionStep(releaseTag);
}
```



Durable Human-in-the-Loop Bridge: A Workflow 5 execution ribbon pauses with zero idle compute at an illuminated approval card until an authorized reviewer clicks Approve, unlocking downstream deployment pods or sandboxed verifications.

5 Real-World Copy-Paste Recipes

Start faster with these five production architectures drawn from `node_modules/chat/resources/guides/`. Each recipe pairs the exact adapters, AI SDK 7 primitives, and durability guards required for real traffic.

Recipe & Channels	Stack & Adapters	Implementation Blueprint

1. Streaming AI Slack Bot
Slack Threads & DMs
	`@chat-adapter/slack`
	`@chat-adapter/redis`
	`ToolLoopAgent` + Gateway
		Subscribe on `onNewMention`, acquire the Redis thread lock, call `agent.stream()` with `anthropic/claude-sonnet-4.6` via AI Gateway, and pipe `stream.textStream` into `thread.post()` for live Slack message updates.

2. GitHub PR Diff Reviewer
GitHub PR Comments
	`@chat-adapter/github`
	`@vercel/sandbox@2`
	Hono + Redis
		On PR `@bot` mention, enqueue a background step that boots an isolated `Sandbox.create()` microVM, checks out the PR head SHA, runs `pnpm test`, analyzes changed files with `ToolLoopAgent`, and replies with exact file-and-line findings.

3. HITL Deploy Gate Card
Slack + MS Teams + Linear
	`@chat-adapter/slack`
	`@chat-adapter/teams`
	Workflow 5 `createWebhook`
		Trigger `start(deployApprovalWorkflow)` from `/deploy`. Inside the workflow, call `createWebhook()`, post a native `Card` with the token, race against `sleep("24h")`, validate `event.user` on click, and call `resumeWebhook()`.

4. Discord & Telegram Knowledge Bot
Discord + Telegram Topics
	`@chat-adapter/discord`
	`@chat-adapter/telegram`
	Nuxt + Redis Locks
		Share one `onSubscribedMessage` handler across Discord threads and Telegram topics. Check `getFeatureSupport()` so Discord opens native `Modal` forms while Telegram falls back cleanly to threaded prompt questions.

5. Zero-Secret Connect Channel Automation
Slack + GitHub + Linear
	`@vercel/connect/chat`
	`VERCEL_OIDC_TOKEN`
	Connect Triggers
		Link connectors via `vercel connect attach`, verify inbound webhooks via Vercel OIDC trigger headers instead of raw secrets, and mint short-lived `app` or per-`user` tokens with `getToken()` right before posting digests.

Scaling Pattern: Prototype on One Channel, Enable Seven More in Config

Because every recipe above isolates business logic inside `ToolLoopAgent` and Workflow 5 steps, expanding Recipe 1 from Slack to Microsoft Teams and Google Chat takes adding two entries to `adapters: {}` and attaching their webhook routes.

Webhook Security, Feature Fallbacks & Replay Ops

Production bots face forged webhook probes, bursty retry storms, and subtle differences across messaging platforms. Lock down verification at the edge, inspect adapter capabilities before rendering rich UI, and test every channel offline with deterministic replay fixtures.

Webhook Signature & Replay Defense

- **Verify Before Parsing:** Let each adapter (or Vercel Connect's OIDC verifier) validate raw HMAC/Ed25519 signatures and timestamp drift before running any agent code.
- **Idempotent Event Keys:** Messaging APIs retry if responses exceed 3s. Deduplicate inbound event IDs in Redis with a short TTL so duplicate webhooks never trigger double deploys or double LLM bills.
- **Fast Ack, Async Work:** Return `200 OK` quickly and offload heavy tool loops via Vercel Functions `waitUntil` or Workflow 5 `start()`.

Graceful Adapter Feature Fallbacks

- **Check `getFeatureSupport(slug)`:** Never assume every adapter supports live token streaming or popup modals.
- **Modal Fallback:** When `modals` is unsupported (e.g., GitHub PR comments or Telegram), collect fields via sequential threaded questions or a signed web link button.
- **Streaming Fallback:** When `streaming` is buffered rather than live-edited, buffer chunks cleanly and post one crisp formatted reply to avoid rate-limit `429` throttling.

Production Operations & Deterministic Testing Matrix

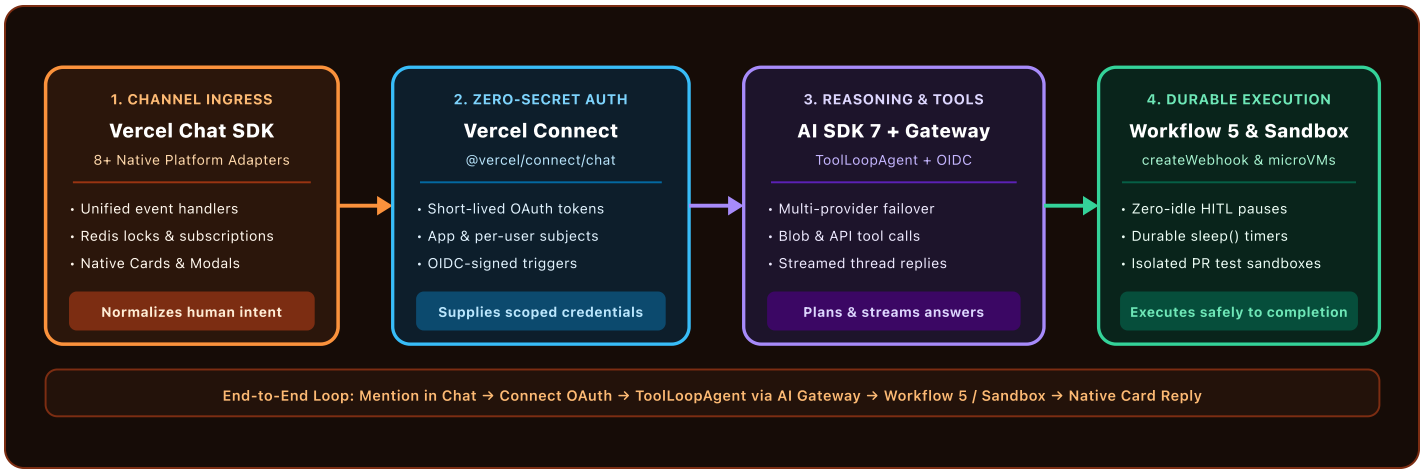
Operational Area	Chat SDK / Vercel Mechanism	Recommended Production Practice
1. Replay Fixtures	`fixtures/replay/*.json`	Capture real Slack, Teams, Discord, and GitHub webhook payloads into `fixtures/replay/` to run deterministic Vitest suites in CI.
2. Rate-Limit Protection	Adapter edit throttling + Redis	Throttle live stream message edits (e.g., every 400–600ms) so rapid LLM tokens never trigger Slack or Discord `429` penalties.
3. Preview Environment Auth	`VERCEL_OIDC_TOKEN` / Connect	Attach separate dev/staging Connect connectors or verify preview webhook URLs have active OIDC and AI Gateway credentials.

Test Multi-Platform Formatting Without Pinging Live Channels

Replaying captured JSON events from `fixtures/replay/` through your local `bot.webhooks` handler lets you verify mention parsing, Redis lock contention, and Card fallbacks in milliseconds inside CI before deploying.

Cross-Tool Synergy: From Mention to Durable Action

Vercel Chat SDK serves as the conversational front door of a unified full-stack AI architecture—connecting zero-secret OAuth credentials, gateway-routed reasoning, and crash-safe execution pods without custom glue servers.



How the Pieces Reinforce Each Other in Production

1 Ingress & Identity (Chat SDK + Connect)

Vercel Chat SDK absorbs platform webhook differences and manages Redis thread locks, while **Vercel Connect** guarantees zero static API keys sit in your deployment config and verifies forwarded webhook triggers via OIDC.

2 Reasoning & Durability (AI SDK 7 + Workflow + Sandbox)

AI SDK 7 drives tool-calling loops across **AI Gateway** models with automatic failover, handing risky code checks to isolated **Vercel Sandboxes** and multi-hour human sign-offs to **Workflow 5** webhooks.

Architecture Payoff: Swap Models, Secrets, or Channels Independently

Need to switch from Anthropic to OpenAI, rotate a Slack installation, or expand from Discord into Linear? Each boundary updates in a single line of configuration without touching your core agent prompts or workflow logic.

Troubleshooting Clinic & Pre-Flight Checklist

Most multi-platform bot bugs trace back to one of seven predictable adapter, concurrency, or auth mistakes. Apply these ten-second fixes, then run the six-point checklist before shipping to production.

Symptom / Anti-Pattern	Root Cause	10-Second Fix
1. Mixing `@slack/bolt` or `telegraf`	Platform-specific bot frameworks hijack routes, bypass Redis locks, and break multi-platform portability.	Remove `@slack/bolt`, `discord.js`, and `telegraf`; use `chat` with `@chat-adapter/slack`, `discord`, or `telegram`.
2. Parallel thread replies race or overwrite context	Serverless webhook invocations execute simultaneously without a distributed thread mutex.	Configure `state: createRedisState()` with `@chat-adapter/redis` so Chat SDK locks per thread ID automatically.
3. Hardcoded bot tokens in `.env`	Static `SLACK_BOT_TOKEN` or GitHub private keys invite rotation failures and wide secret exposure.	Migrate to `@vercel/connect/chat` with `getToken()` and Connect-forwarded OIDC webhook triggers.
4. `setTimeout` / `setInterval` polling times out	Polling inside serverless webhook functions hits max-duration limits and loses state across deploys.	Move multi-step waits into a `"use workflow"` function using durable `sleep()` and `createWebhook()`.
5. Importing deprecated `chat/adapters`	Legacy subpath is deprecated and pulls unnecessary coupling into onboarding or tooling code.	Import `listAdapters`, `getFeatureSupport`, and `listEnvVars` directly from `@chat-adapter/catalog`.
6. Streaming or modals crash on GitHub / Telegram	Bot invokes live modals or token-stream edits on adapters whose platform API does not support them.	Inspect `getFeatureSupport(slug)` first and branch cleanly to threaded follow-ups or buffered final posts.
7. Preview webhooks fail AI calls (401/403)	Missing `AI_GATEWAY_API_KEY` or expired local/preview `VERCEL_OIDC_TOKEN` on branch deploys.	Verify Vercel OIDC is enabled for Preview deployments (or set `AI_GATEWAY_API_KEY`), and refresh locally with `vercel env pull`.

6-Point Pre-Flight Checklist Before Going Live

- ☑ **Catalog Capabilities Checked:** Verified `getFeatureSupport()` fallbacks for streaming and modals on every enabled adapter.
- ☑ **Redis Thread Locks Active:** Confirmed `@chat-adapter/redis` handles distributed locks and thread subscriptions across instances.
- ☑ **Zero Static Bot Secrets:** Replaced hardcoded tokens with `@vercel/connect` connectors and OIDC webhook verification.
- ☑ **Bounded `ToolLoopAgent` Loops:** Configured `stopWhen: isStepCount(N)` and AI Gateway `provider/model` routing.
- ☑ **Durable HITL & Sandboxes:** Moved long approvals to Workflow 5 `createWebhook` + `sleep()` and PR code checks to Vercel Sandbox.
- ☑ **Replay Fixtures Green:** Tested webhook signatures, idempotency, and Card formatting against `fixtures/replay/` in CI.