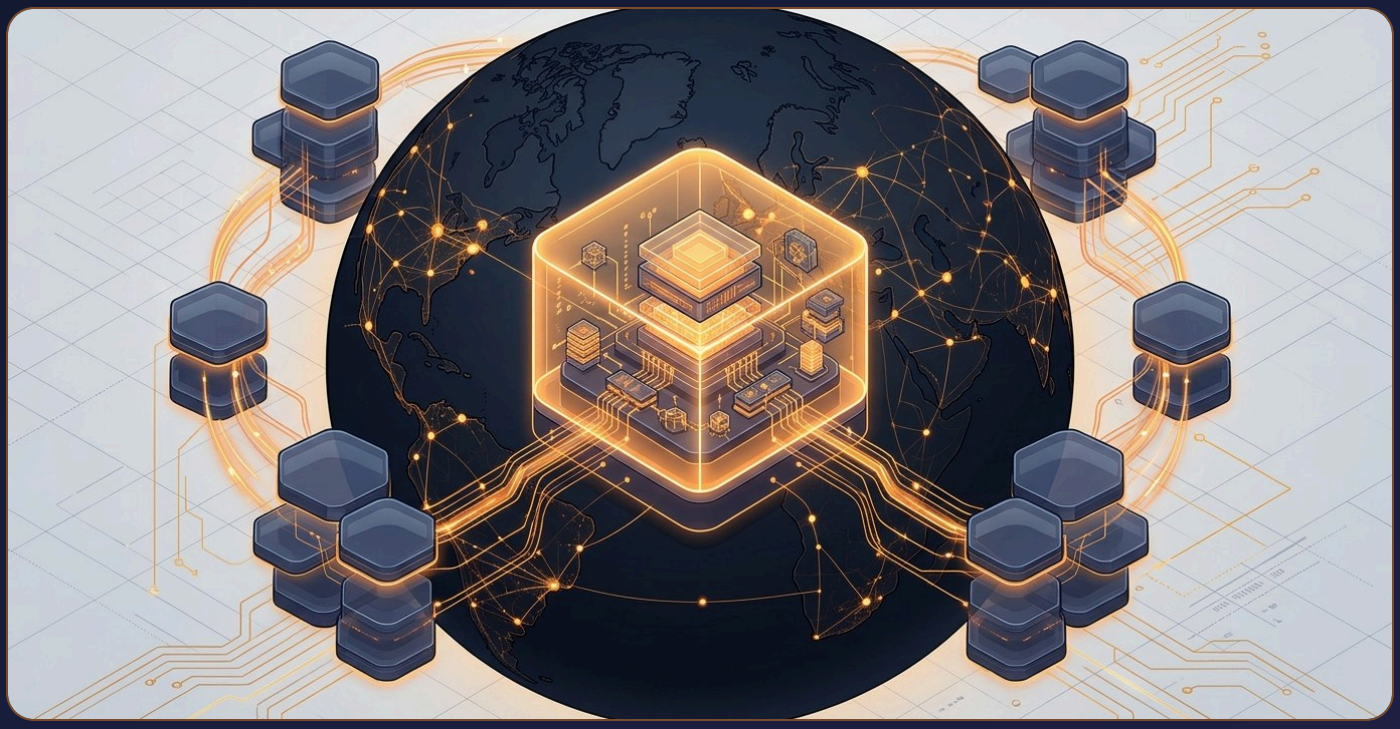
CLOUD DATA & EDGE AI ARCHITECTURE • FIELD GUIDE

Cloudflare Agents, Workers AI & Sandboxes

Deploy stateful, globally distributed AI agents on the Cloudflare edge using the Cloudflare Agents SDK, Durable Objects SQLite state, Workflows, AI Gateway, Workers AI, and Container Sandboxes.

Agents SDK & Durable Objects | Embedded SQLite & WebSockets | Remote MCP & OAuth 2.1 | Sandbox SDK 1.0 & Containers | AI Gateway & Workers AI | Cloudflare Workflows



Stateful Edge Agents

Every `Agent` runs as a globally addressable Durable Object with private SQLite storage, hibernatable WebSockets, and durable alarms.

Isolated Code Execution

Pair sub-millisecond Dynamic Workers with `ctx.container` & Sandbox SDK 1.0 to run untrusted Node, Python, and shell workloads safely.

Unified AI Control Plane

Route across Workers AI and 24+ LLM providers through AI Gateway with free caching, rate limits, fallbacks, DLP, and Llama Guardrails.

The Stateful Edge Shift: One Durable Object Per Agent

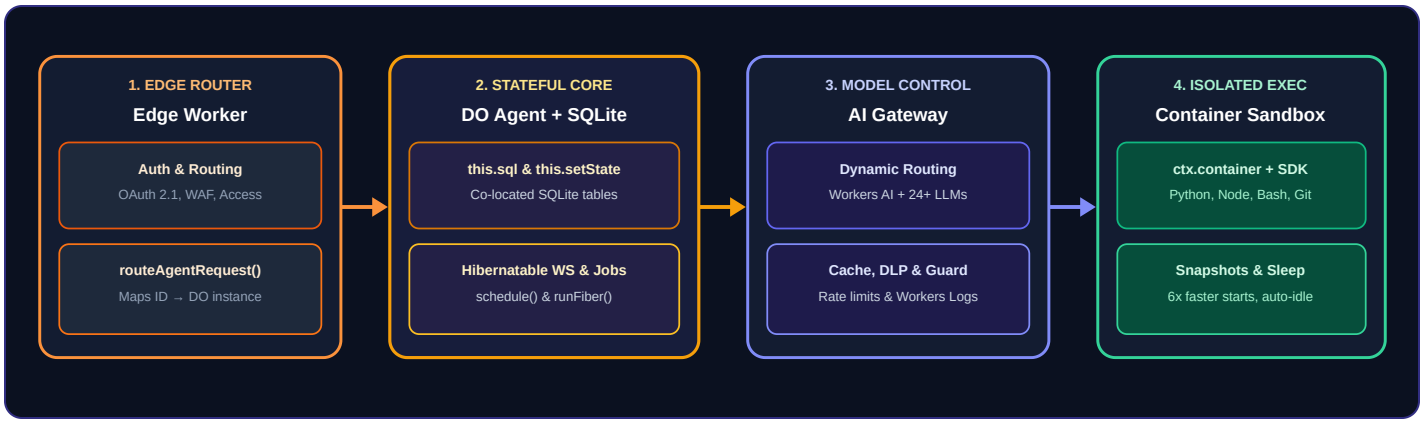
Traditional stateless serverless functions forget everything between HTTP calls, forcing teams to stitch together Redis, Postgres, queues, and WebSocket brokers. Cloudflare's **Agents SDK** (`agents`) flips the model: each agent instance is a single-threaded **Durable Object** with its own embedded SQLite database, hibernatable WebSockets, and alarm scheduler running close to the user.

Quick Navigation

1. Foundations & Edge Mental Model	p. 2	7. AI Gateway, Workers AI & Fallbacks	p. 8
2. Cloudflare AI Primitive Matrix	p. 3	8. 5 Copy-Paste Agent Recipes	p. 9
3. Annotated Wrangler & Gateway HUD	p. 4	9. Zero-Trust Security & Cost Controls	p. 10
4. Stateful Agents on Durable Objects	p. 5	10. Cross-Tool Synergy Pipeline	p. 11
5. Remote MCP Servers & OAuth 2.1	p. 6	11. Troubleshooting & Pre-Flight	p. 12
6. Containers & Ephemeral Sandboxes	p. 7	Golden Rule: Co-locate State & Compute	All

End-to-End Request & State Lifecycle at the Edge

When a user or webhook targets an agent by name (e.g., `tenant-42-analyst`), Cloudflare routes the request to that exact Durable Object anywhere on the network. The agent reads local SQLite in microseconds, calls models via AI Gateway, and spins up an isolated Container Sandbox only when untrusted code execution is required:



1 Zero-Cold-DB State

SQLite runs inside the same process boundary as your Durable Object. Queries execute synchronously with zero network round-trips and survive evictions automatically.

2 Hibernation = $0 Idle

Durable Objects hibernate open WebSockets and Workflows sleep on `step.waitForEvent()` without charging wall-clock compute while waiting for humans or webhooks.

3 Isolates First, Containers Next

Run lightweight JS/TS agent logic in V8 isolates (or Dynamic Workers) in milliseconds, and attach a Linux container via `ctx.container` only when POSIX tools are needed.

Why This Matters in Late 2026

Durable Objects gained a Free tier (2025), Containers & Sandbox SDK 1.0 reached GA and were rebuilt around native `ctx.container` (Sep 2026), and `@cloudflare/workers-oauth-provider` v1.x (Oct 2026) standardized OAuth 2.1 for remote MCP servers—making Cloudflare a complete single-vendor runtime for production AI agents.

Cloudflare AI Primitive Decision Matrix

Cloudflare's AI stack is modular: each primitive solves one durability, inference, retrieval, or isolation boundary cleanly. Use this matrix to map every agent workload to its exact primitive without overpaying for idle compute.

Primitive	Best-Fit Workload	State & Execution Model	Pricing & Free Tier	Avoid Using For
Agents SDK `agents` + DO	Per-user/room stateful agents, live chat, UI state sync, MCP servers, crons.	Single-threaded DO; embedded SQLite, hibernatable WS, `schedule()`, `runFiber()`.	DO Free tier; Paid bills requests + active duration + SQLite.	Heavy video encoding or multi-GB ML training.
Workflows `WorkflowEntrypoint`	Multi-step pipelines: doc ingest, fan-out, retries, multi-day human approvals.	Step-checkpointed (`step.do`, `sleep`, `waitForEvent`); survives deploys.	Active CPU only ($0 sleeping); Free ~3M req/mo.	Sub-50ms live WS streaming to browsers.
Workers AI `env.AI.run()`	Fast serverless open models (50+ LLMs, embeddings, vision, Llama Guard).	Stateless GPU inference via binding, OpenAI REST, or AI Gateway.	**10k Neurons/day free**; $0.011 / 1k Neurons paid.	Frontier reasoning needing Claude/GPT/Gemini.
AI Gateway Control Plane	Proxy for Workers AI + 24+ LLMs: caching, fallbacks, rate limits, DLP.	Universal OpenAI/Anthropic REST & `dynamic/<route>`; zero-deploy swaps.	Core routing/cache **free**; Unified Billing +5% credit fee.	Custom stateful app logic (keep in Worker/DO).
Vectorize & AI Search	**Vectorize:** custom RAG. **AI Search** (ex-AutoRAG): managed R2 RAG + MCP.	Global vector index or turnkey R2-to-index continuous re-indexing.	Usage-based queried dimensions & stored vectors.	Transactional chat ledgers (use DO SQLite).
Container Sandboxes `ctx.container`	Untrusted Python, Node, Bash, Git repos, compilers requiring full Linux.	DO owns Linux container via `ctx.container`; runtime image, beta snapshots.	Workers Paid; active vCPU, RAM, disk & snapshots.	Pure JS/TS transforms (use Dynamic Workers).
Dynamic Workers Isolates	AI-generated JS/TS "Code Mode" tools with 100x faster cold starts.	Ephemeral V8 isolates with strict capability bindings, zero host access.	Workers compute pricing; ~ms startup overhead.	Native C/Python binaries or apt packages.

Composition Patterns That Win in Production

A **Interactive Agent + Background Workflow**

Keep the user's live WebSocket in an **Agent DO** for <50ms streaming and state sync. Spawn a **Workflow** for multi-minute audits or retries and push updates back to the Agent via RPC.

B **Agent DO +** `ctx.container` **Sandbox**

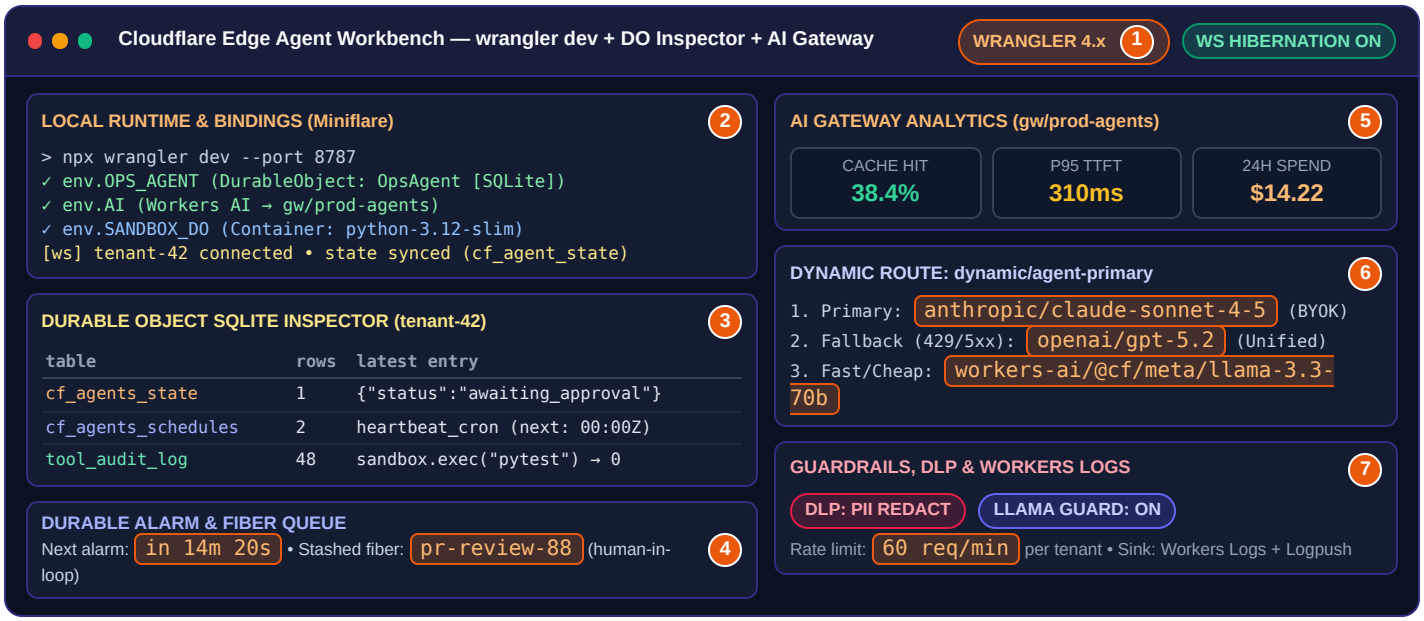
Your Agent DO holds conversation memory in SQLite and starts its attached Linux container via `this.ctx.container`, snapshotting disk state before sleeping to preserve Git checkouts.

Migration Advisory (Dec 31, 2026 Cutoff)

Rebuilt Containers (Sep 30, 2026) deliver 6x faster starts, runtime image selection, and beta filesystem snapshots natively on `this.ctx.container` + Sandbox SDK 1.0 utilities. Legacy `Container` / `Sandbox` classes receive updates only through **Dec 31, 2026**.

Annotated Workspace Tour: Wrangler Dev, DO SQLite & AI Gateway

Developing edge agents locally mirrors production: `wrangler dev` runs your Worker, Durable Object SQLite database, and hibernatable WebSockets in Miniflare while forwarding model calls through your live AI Gateway for real-time cost, cache, and guardrail telemetry.



7 Controls Every Edge Agent Engineer Must Master

1 **Wrangler CLI & Typed Env:** Run `npx wrangler types` after every `wrangler.jsonc` edit to regenerate strict TypeScript bindings for DOs, AI, R2, and Containers.

2 **Local Binding Simulator:** `wrangler dev` boots real SQLite-backed Durable Objects locally, letting you test WebSockets, alarms, and RPC calls offline before deploying.

3 **Embedded SQLite Inspector:** Inspect `cf_agents_state` (auto-managed by `setState`), `cf_agents_schedules`, and custom tables directly inside `.wrangler/state/v3/do`.

4 **Alarm & Fiber Queue:** Agents SDK multiplexes all scheduled jobs onto the DO's single hardware alarm and checkpoints paused `runFiber()` tasks so evictions never drop work.

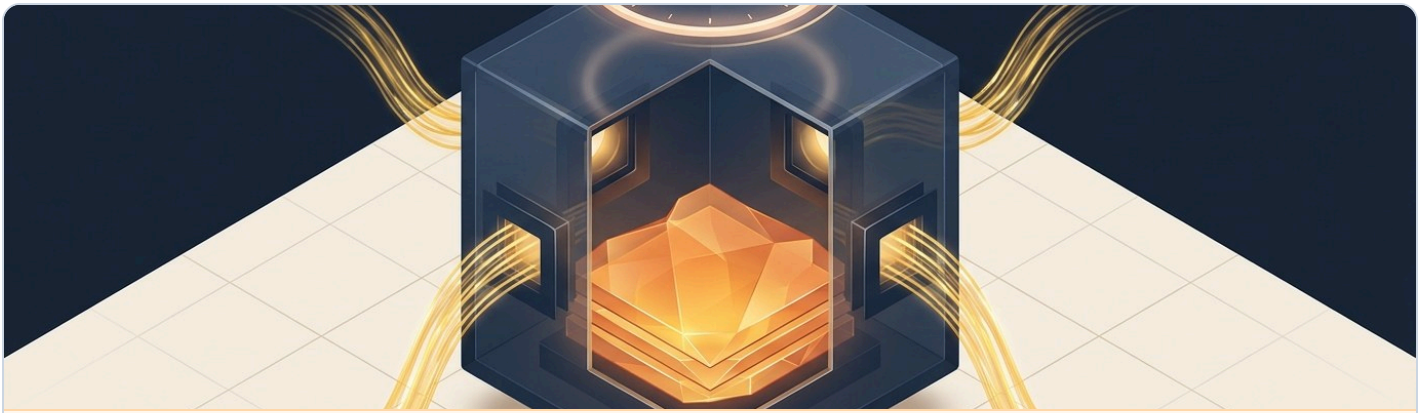
5 **AI Gateway Telemetry:** Track exact-match cache hit rate, time-to-first-token (TTFT), token counts, and real-time spend across every model provider from one pane.

6 **Dynamic Routing Rules:** Swap primary models, shift traffic splits, or add 429/5xx fallbacks inside `dynamic/<route>` without redeploying your Worker code.

7 **Guardrails, DLP & Workers Logs:** Inspect prompts with free DLP scanning and Llama Guardrails, enforce per-tenant rate limits, and query full traces in Workers Logs.

Embedded SQLite, Hibernatable WebSockets & Durable Scheduling

Subclassing `Agent<Env, State>` from the `agents` package turns a Cloudflare Durable Object into a persistent, real-time agent runtime with four built-in durability primitives—requiring zero external databases or message brokers.



Anatomy of a Durable Agent: Embedded SQLite plates (`this.sql` & `this.setState`) sit directly inside the isolate, surrounded by hibernatable WebSocket channels and a SQLite-backed cron/alarm scheduler.

1 Dual Persistence: `setState()` + `this.sql`

Call `this.setState(next)` to atomically write JSON state to `cf_agents_state` and broadcast a `cf_agent_state` sync frame to all connected React/JS clients. Use tagged-template `this.sql`...`` for relational audit logs, message histories, and tool ledgers.

2 Hibernatable WebSockets & `useAgent`

WebSockets stay open at the Cloudflare edge even when the DO hibernates to zero memory. Store per-socket metadata with `connection.setState()`; the official React `useAgent()` hook auto-reconnects and syncs state bi-directionally.

3 Durable Jobs: `this.schedule()`

Schedule delayed seconds (`60`), exact `Date` timestamps, or cron strings (`"0 * * * *"`). The SDK stores jobs in SQLite and multiplexes them onto the DO's single alarm so independent schedules never overwrite each other.

4 Long Tasks & Human-in-the-Loop

Wrap critical active work in `keepAliveWhile()` to prevent mid-call eviction, or use `runFiber()` + `stash()` to checkpoint long multi-turn runs and pause for human approval at $0 idle cost.

```
import { Agent, routeAgentRequest } from "agents";

interface OpsState { pendingApproval: string | null; runs: number; }

export class OpsAgent extends Agent<Env, OpsState> {
  initialState: OpsState = { pendingApproval: null, runs: 0 };

  async onStart() {
    this.sql`CREATE TABLE IF NOT EXISTS audit (id TEXT PRIMARY KEY, action TEXT, ts INTEGER)`;
    await this.schedule("0 * * * *", "hourlyDigest", { channel: "ops" }); // SQLite-backed cron
  }

  async requestDeploy(releaseId: string) {
    this.sql`INSERT INTO audit (id, action, ts) VALUES (${crypto.randomUUID()}, ${releaseId}, ${Date.now()})`;
    this.setState({ ...this.state, pendingApproval: releaseId }); // Persists & pushes over WS
  }

  async hourlyDigest(payload: { channel: string }) { /* wakes DO via alarm, runs, then hibernates */ }
}

export default { fetch: (req: Request, env: Env) => routeAgentRequest(req, env) };
```

Remote MCP Servers with `McpAgent` & OAuth 2.1

Cloudflare is a primary edge host for remote **Model Context Protocol (MCP)** servers over Streamable HTTP (`/mcp`). By pairing a Durable Object MCP server with `@cloudflare/workers-oauth-provider` v1.x (MCP 2026-07-28 spec), any client—Claude, Cursor, Windsurf, or another Agent—can authenticate with GitHub, Google, or Cloudflare Access and call your tools globally.

Color-Coded 5-Part Remote MCP & OAuth Anatomy

1. DO Binding & Compat
2. SQLite Migration
3. OAuth 2.1 & MCP_RESOURCE
4. McpAgent Tool Class
5. Protected /mcp Route

```
// wrangler.jsonc
{ "name": "edge-mcp-server", "main": "src/index.ts", "compatibility_date": "2026-10-01",
  "compatibility_flags": ["nodejs_compat"], "durable_objects": { "bindings": [{ "name": "MY_MCP", "class_name": "EdgeMCP" }] } },
  "migrations": [{ "tag": "v1", "new_sqlite_classes": ["EdgeMCP"] }],
  "kv_namespaces": [{ "binding": "OAUTH_KV", "id": "<kv-id>" }], "vars": { "MCP_RESOURCE": "https://mcp.example.com/mcp" } }

// src/index.ts
export class EdgeMCP extends McpAgent<Env, State, AuthProps> { server = new McpServer({ name: "ops", version: "1.0.0" });
  async init() { this.server.tool("deploy_status", { env: z.string() }, async ({ env }) => ({ content: [{ type: "text", text: `OK:${env}` }] })); } }
export default new OAuthProvider({ apiRoute: "/mcp", apiHandler: EdgeMCP.serve("/mcp", { binding: "MY_MCP" }), defaultHandler: GitHubAuthHandler });
```

1 Why `MCP_RESOURCE` & Split Auth Matter

In `@cloudflare/workers-oauth-provider` v1.x (Oct 2026), every issued OAuth 2.1 access token is cryptographically bound to the exact `MCP_RESOURCE` URL. For enterprise fleets, split `OAuthAuthorizationServer` and `OAuthResourceServer` into two Workers connected over a zero-latency Service Binding with step-up auth.

2 `McpAgent` vs. `createMcpHandler` & RPC

Use `McpAgent` (`agents/mcp`) when each MCP client session needs stateful SQLite memory inside a DO. For stateless tools or MCP SDK v2 migration, use `createMcpHandler` (`agents/mcp/server`). When an `Agent` and MCP server live in the same Worker, connect them via `RPCClientTransport` with zero HTTP overhead.

Client / Tool	Connection Command / Endpoint	Auth & Transport Note
Claude Code / Streamable HTTP	`claude mcp add --transport http ops https://mcp.example.com/mcp`	Native Streamable HTTP + automatic browser OAuth 2.1 PKCE login flow.
Stdio-Only Desktop Clients	`npx -y mcp-remote https://mcp.example.com/mcp`	Bridges legacy stdio clients to your remote Cloudflare OAuth endpoint.
Local Debugging	`npx @modelcontextprotocol/inspector` → `http://localhost:8787/mcp`	Inspect tool schemas, test OAuth scopes, and tail logs with `wrangler tail`.

Version Pinning Rule for `agents/mcp`

Pin `@modelcontextprotocol/sdk` and `@modelcontextprotocol/server` to the exact versions expected by your installed `agents` release (verify with `npm ls @modelcontextprotocol/sdk`) so TypeScript never sees two conflicting `McpServer` types.

Cloudflare Containers & Ephemeral Code Sandboxes

When an AI agent writes Python data scripts, runs `npm test`, or edits a Git repository, you must never execute that untrusted code inside your application's memory space. Cloudflare's rebuilt **Containers** and **Sandbox SDK 1.0** (Sep 2026) let each Durable Object spawn, exec, snapshot, and sleep an isolated Linux workspace directly at the edge.

1 Native `this.ctx.container` Architecture

- **6x Faster Agent Starts:** Rebuilt container control plane boots on-demand Linux sandboxes directly from your Durable Object.
- **Runtime Selection:** Choose Docker image, instance size, and idle sleep policy dynamically in code per tenant or task.
- **Beta Filesystem Snapshots:** Snapshot disk state before sleeping so a coding agent resumes its exact repo checkout and build cache.

2 Isolates First, Containers On Demand

- **Dynamic Workers:** For AI-generated JS/TS transforms and "Code Mode" tool calls, spawn ephemeral V8 isolates ~100x faster than Linux containers.
- **Sandbox SDK 1.0:** Utility helpers (command `exec`, file read/write, streaming stdout, preview ports) that work alongside `ctx.container`.
- **`@cloudflare/computer`:** Higher-level workspace combining Dynamic Workers + Containers with a synchronized filesystem.

Safe Untrusted Execution Pattern Inside a Durable Object

```
import { Agent } from "agents";
import { execInSandbox, writeFiles } from "@cloudflare/sandbox";

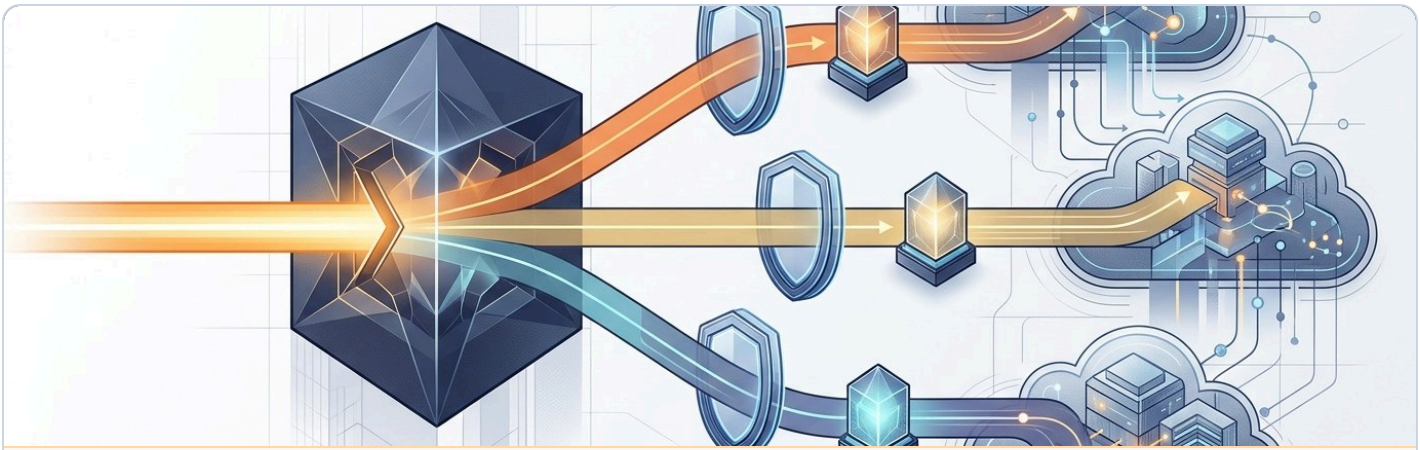
export class CodeRunnerAgent extends Agent<Env> {
  async runUserScript(pyCode: string) {
    // 1. Start or wake the DO's dedicated Linux container with runtime image & tight idle sleep
    await this.ctx.container.start({
      image: "registry.cloudflare.com/your-acct/py-sandbox:3.12",
      instanceType: "lite",
      sleepAfter: "5m",
      enableInternet: false, // Block lateral/egress exfiltration by default
    });
    // 2. Write untrusted code & execute with hard timeout; credentials stay in DO, never on disk
    await writeFiles(this.ctx.container, [{ path: "/workspace/task.py", content: pyCode }]);
    const res = await execInSandbox(this.ctx.container, "python3 /workspace/task.py", { timeoutMs: 15_000 });
    return { exitCode: res.exitCode, stdout: res.stdout.slice(0, 8000), stderr: res.stderr.slice(0, 2000) };
  }
}
```

5 Non-Negotiable Sandbox Security & Cost Rules

Control	Implementation Rule	Threat / Failure Prevented
No Secrets on Disk	Never write API keys or OAuth tokens into container env files or filesystem snapshots; proxy outbound API calls through the parent DO.	Prompt-injected code reading `/proc/self/environ` or leaking snapshot credentials.
Default-Deny Egress	Set `enableInternet: false` unless required, or route egress through a Worker allowlist proxy.	Data exfiltration, SSRF against internal endpoints, and crypto-mining abuse.
Hard Exec Timeouts	Enforce 10–30s command timeouts and truncate stdout/stderr before feeding output back to the LLM context.	Infinite loops burning vCPU and token-bombing your LLM bill.
Per-Tenant Instance Caps	Track active container leases in DO SQLite / KV and enforce a max concurrent sandbox count per organization.	Runaway parallel tool calls causing unexpected Workers Paid container spend.
Aggressive Idle Sleep	Set `sleepAfter` to `2m–5m` and snapshot only clean workspace directories (excluding `/tmp`).	Paying for idle Linux containers between sporadic user chat turns.

AI Gateway, Workers AI & Multi-Provider Fallbacks

Hardcoding a single LLM vendor inside an edge agent invites 429 rate-limit outages and runaway token bills. Routing every model call through **Cloudflare AI Gateway** gives you free exact-match caching, dynamic fallbacks across 24+ providers plus **Workers AI**, DLP redaction, and Llama Guardrails without redeploying code.



Single Control Plane: One universal endpoint routes through Guardrails, DLP, and edge caching before fanning out to Workers AI, Anthropic, OpenAI, or Google AI Studio with automatic fallback.

1 Workers AI Edge Models

Run 50+ open models (Llama 3.3, Qwen, Inkling 2, BGE embeddings, Whisper) via `env.AI.run()`. **10,000 Neurons/day free**; $0.011 per 1,000 Neurons paid—ideal for fast routing, summaries, and embeddings.

2 Dynamic Routes & SDK

Use `createAI()` from `agents/models/ai-sdk` or call `dynamic/<route>` via the OpenAI/Anthropic-compatible endpoint. Fail over from Claude → GPT → Workers AI on 429/5xx automatically.

3 BYOK vs. Unified Billing

Store provider keys securely in AI Gateway (BYOK, $0 markup) or opt into **Unified Billing** (one Cloudflare invoice; 0% token markup, 5% fee on credit purchases).

```
// Calling a Dynamic Route via AI Gateway with exact-match cache TTL & tenant metadata
const res = await env.AI.gateway("prod-agents").run({
  provider: "dynamic",
  endpoint: "agent-primary", // Configured in dashboard: Claude Sonnet -> GPT-5.2 -> Workers AI
  headers: {
    "cf-aig-cache-ttl": "3600",                // Free exact-match edge cache for 1 hour
    "cf-aig-metadata": JSON.stringify({ tenantId: this.name, workflow: "triage" }),
  },
  query: { messages, temperature: 0.2, max_tokens: 1024 },
});
```

Caching & Cost Levers

Set `cf-aig-cache-ttl` on deterministic classification, extraction, and RAG synthesis prompts. Combine with `cf-aig-skip-cache: true` on streaming creative turns so hit rates stay 30–55% without stale answers.

Guardrails, DLP & Workers Logs

AI Gateway DLP scanning is free (expanded with Zero Trust DLP); Guardrails runs Llama Guard on Workers AI (billed per Neuron). Gateways created on/after Sep 24, 2026 store traces in **Workers Logs**.

5 Real-World Copy-Paste Cloudflare Agent Recipes

Every production edge agent combines two or three primitives. Pick a blueprint below, copy the verified snippet, run `npx wrangler types`, and ship with `npx wrangler deploy`.

Recipe	Primitives Combined	Core Mechanism	Primary Outcome
1. Support Triage Agent	`Agent` DO + SQLite + AI Search + Gateway	Streams answers over WS, queries R2 docs via AI Search, caches FAQs.	<400ms TTFT, 40%+ cache hit.
2. Code Reviewer	`Agent` DO + `ctx.container` + Webhook	Wakes on PR webhook, runs tests in Linux sandbox, posts diff notes.	Zero-trust isolated tests.
3. OAuth Enterprise MCP	`McpAgent` + `workers-oauth-provider`	Exposes internal tools over `/mcp` with GitHub/Access OAuth 2.1.	Instant Claude/Cursor access.
4. Human-in-Loop Flow	`WorkflowEntrypoint` + `waitForEvent`	Pauses refund/deploy pipeline until Slack/UI approval arrives ($0 idle).	Crash-proof governance.
5. Market Monitor	`Agent` DO + `schedule()` + Browser	Cron alarm wakes DO hourly, scrapes pages, diffs in SQLite, alerts.	Zero-server monitoring.

Recipe A: Durable Human-in-the-Loop Workflow (`src/approval-workflow.ts`)

```
import { WorkflowEntrypoint, WorkflowEvent, WorkflowStep } from "cloudflare:workers";

export class DeployApprovalWorkflow extends WorkflowEntrypoint<Env, { releaseId: string; agentId: string }> {
  async run(event: WorkflowEvent<{ releaseId: string; agentId: string }>, step: WorkflowStep) {
    const summary = await step.do("risk-brief", () =>
      this.env.AI.run("@cf/meta/llama-3.3-70b-instruct-fp8-fast", { prompt: `Risks for ${event.payload.releaseId}` })
    );
    // Sleeps at $0 compute cost until operator clicks Approve in the Agent UI
    const decision = await step.waitForEvent<{ approved: boolean }>("human-approval", {
      type: "deploy-decision", timeout: "24 hours",
    });
    if (decision.payload.approved) {
      await step.do("notify-agent", async () => {
        const stub = this.env.OPS_AGENT.get(this.env.OPS_AGENT.idFromName(event.payload.agentId));
        await stub.fetch("https://agent/approved", { method: "POST", body: JSON.stringify(summary) });
      });
    }
  }
}
```

Recipe B: React Client (`src/App.tsx`)

```
import { useAgent } from "agents/react";
import { useState } from "react";

export function OpsConsole({ id }: { id: string }) {
  const [st, setSt] = useState({ runs: 0 });
  const agent = useAgent({
    agent: "ops-agent", name: id,
    onStateUpdate: (next) => setSt(next),
  });
  return (
    <button onClick={() =>
      agent.call("requestDeploy", ["v2.14"])
    }>Deploy ({st.runs})</button>
  );
}
```

Recipe C: `wrangler.jsonc` Bindings

```
{
  "name": "edge-agent-platform",
  "main": "src/index.ts",
  "compatibility_date": "2026-10-01",
  "compatibility_flags": ["nodejs_compat"],
  "ai": { "binding": "AI" },
  "durable_objects": { "bindings": [
    { "name": "OPS_AGENT", "class_name": "OpsAgent" }
  ]},
  "migrations": [
    { "tag": "v1", "new_sqlite_classes": ["OpsAgent"] }
  ],
  "workflows": [{ "name": "deploy-approval",
    "binding": "DEPLOY_WORKFLOW",
    "class_name": "DeployApprovalWorkflow" }]
}
```

Security, Zero-Trust Access & Edge Cost Controls

Autonomous agents hold tool credentials, execute code, and call metered LLMs. Securing them at the Cloudflare edge means enforcing identity before a Durable Object ever wakes, isolating every tenant's storage and sandbox, and bounding inference spend at the gateway.

1 Edge Auth & Zero Trust Access

- **Cloudflare Access:** Gate internal admin UIs and `/agents/*` endpoints with SSO + JWT validation at the edge before invoking Workers.
- **OAuth 2.1 for Remote MCP:** Use `@cloudflare/workers-oauth-provider` with resource-bound tokens (`MCP_RESOURCE`) and short TTLs.
- **WAF & AI Crawl Control:** Block unauthorized scrapers and automated abuse at L7 while allowing verified webhook/API signatures.

2 Prompt & Data Loss Prevention

- **AI Gateway DLP:** Scan outbound prompts and inbound completions for SSNs, API keys, and PCI patterns before data leaves Cloudflare.
- **Llama Guardrails:** Filter prompt-injection attempts and unsafe tool arguments inline on Workers AI.
- **App-Layer Redaction:** Strip sensitive PII before writing to `this.sql` or Workers Logs to satisfy SOC 2 / GDPR retention policies.

Defense-in-Depth & Cost Governance Matrix

Risk Surface	Cloudflare Edge Control	Operational Budget / Metric Target
Cross-Tenant Data Leak	Derive DO IDs strictly from verified JWT `sub` / `orgId` claims—never from raw client query params.	100% isolated SQLite file & sandbox per tenant ID.
Runaway LLM Loops	Set AI Gateway rate limits (req/min + tokens/hr) and cap agent tool-call loops to `maxSteps ≤ 12`.	Hard circuit breaker before $10/tenant/day spike.
Sandbox Breakout / Leak	Disable container internet by default; never store secrets on snapshot disks; enforce 15s `exec` timeouts.	Zero credentials inside container filesystem.
Idle Compute Waste	Use hibernatable WebSockets, `step.waitForEvent()` in Workflows, and `sleepAfter: "3m"` on Containers.	$0 idle compute while waiting for human input.
Opaque Production Failures	Enable Workers Logs (real-time structured JSON) + Logpush to R2/Datadog with `cf-aig-metadata` tags.	100% trace correlation from WS frame → LLM → Sandbox.

Tiered Model Routing Saves 60–75%

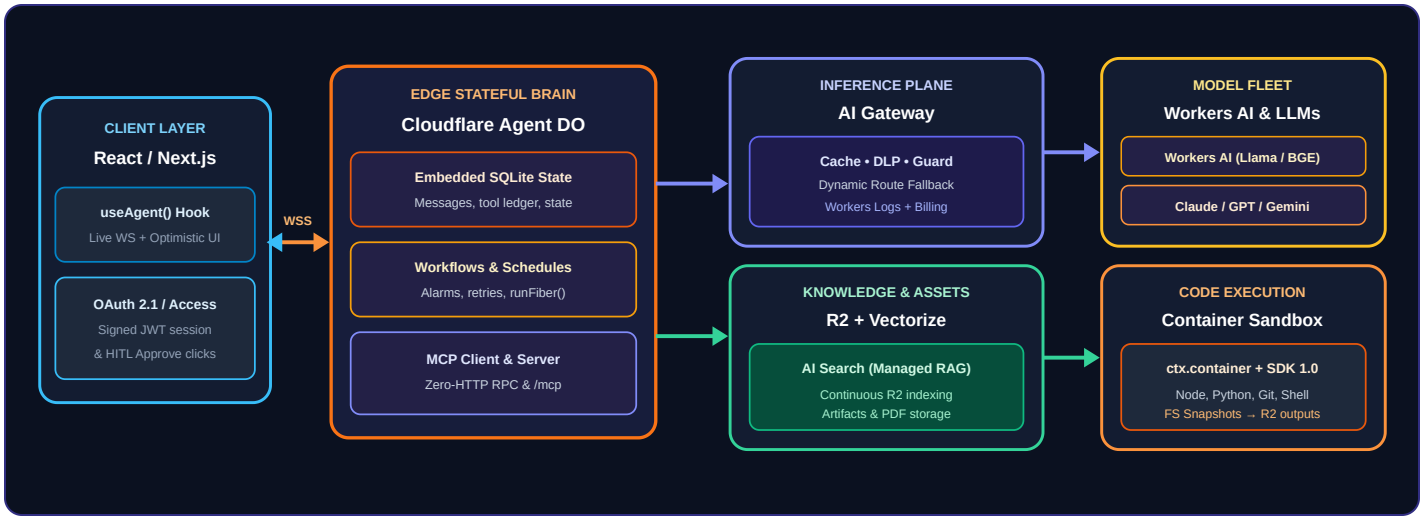
Route intent classification, JSON extraction, and Guardrails to **Workers AI** ($0.011 / 1k Neurons after 10k free/day). Escalate only complex planning and final code synthesis through AI Gateway to frontier models.

Never Trust Client-Supplied Agent Names

If your Worker blindly calls `routeAgentRequest(req, env)` without verifying the caller's session token matches the requested agent name, any user can connect to another tenant's Durable Object WebSocket.

End-to-End Edge Agent Synergy Pipeline

Production AI applications rarely stop at a single chat box. Here is how a React/Next.js frontend, Cloudflare Agent Durable Object, AI Gateway, R2/Vectorize RAG layer, and Container Sandbox interlock into one globally distributed, low-latency pipeline.



5-Stage Operational Flow (From Click to Verified Output)

1 Zero-Hop Client Sync

Next.js authenticates the user via OAuth 2.1 / Access and opens a persistent WebSocket to the tenant's `Agent` DO via `useAgent()`. Every state mutation pushes instantaneously to all open browser tabs.

2 Grounded Edge Retrieval

Before calling a frontier model, the Agent queries recent turn context from local SQLite and fetches top-K domain passages from **Vectorize** or **AI Search** (backed by R2 buckets) in a single region.

3 Guarded Multi-Model Inference

The prompt passes through **AI Gateway** for DLP redaction, exact-match cache lookup, and dynamic routing—failing over seamlessly if the primary provider rate-limits.

4 Sandboxed Execution & Artifacts

When the LLM emits a code or repo tool call, the DO wakes `this.ctx.container`, runs the script with a hard timeout, uploads generated charts/CSVs to **R2**, and returns a signed URL to the client.

Stage 5: Durable Handoff & Hibernation

Once the turn completes, the Agent checkpoints its ledger in SQLite, schedules any follow-up check via `this.schedule()`, lets the Container Sandbox sleep after 3 minutes, and hibernates its own WebSocket—dropping steady-state cost to near zero until the next event arrives.

Troubleshooting Clinic & 6-Point Pre-Flight Checklist

Most Cloudflare Agents, MCP, and Sandbox deploy errors trace back to a handful of `wrangler.jsonc`, dependency-pinning, or hibernation gotchas. Diagnose them fast below, then run the 6-point checklist before shipping to production.

Top 7 Production & Deploy Fixes

Symptom / Error	Root Cause	Verified Fix
DO deploy fails on SQLite `new_sqlite_classes required`	Class bound in `durable_objects.bindings` without a matching SQLite migration entry.	Add `"migrations": [{ "tag": "v1", "new_sqlite_classes": ["MyAgent"] }]` and match exported class name exactly.
`McpServer` **not assignable** TypeScript build error	Two different versions of `@modelcontextprotocol/sdk` installed in `node_modules`.	Run `npm ls @modelcontextprotocol/sdk` and pin the exact version required by your `agents` release.
`Could not resolve "ai"` Bundler fails on MCP deploy	`agents` dynamically imports Vercel's `ai` package on client paths even if unused.	Install `ai` in `dependencies` (`npm i ai`) and ensure `compatibility_flags: ["nodejs_compat"]` is set.
`Env` **untyped / red squiggles**	Added a new DO, AI, KV, or Workflow binding in `wrangler.jsonc` without regenerating types.	Run `npx wrangler types` after every config edit (and in CI before `tsc --noEmit`).
State lost after WS idle	Stored session variables on in-memory `this.foo` fields instead of SQLite/WebSocket state.	Persist shared state with `this.setState()` / `this.sql` and per-socket state via `connection.setState()`.
OAuth MCP 401 Invalid Token	`MCP_RESOURCE` in `wrangler.jsonc` still points to `localhost` instead of prod URL.	Set `MCP_RESOURCE` to the exact public `https://.../mcp` URL clients connect to; redeploy.
No snapshots / slow container	Still extending legacy `Container` or `Sandbox` wrapper classes (frozen Dec 31, 2026).	Migrate to native `this.ctx.container` with `durable_object` scheduling + Sandbox SDK 1.0 helpers.

6-Point Production Launch Pre-Flight Checklist

- ☑ **1. Tenant-Bound Routing:** Edge Worker validates OAuth 2.1 / Access JWT and derives the Durable Object ID from verified claims—never raw URL params.
- ☑ **2. SQLite Migrations & Types:** Every `Agent` and `McpAgent` is listed in `new_sqlite_classes`, `nodejs_compat` is enabled, and `wrangler types` passes cleanly.
- ☑ **3. AI Gateway Guardrails:** All model calls route via `dynamic/<route>` with a fallback provider, `cf-aig-cache-ttl`, rate limits, DLP, and Llama Guardrails enabled.
- ☑ **4. Hardened Sandboxes:** Containers use native `ctx.container` with `enableInternet: false`, 15s `exec` timeouts, 3–5m idle sleep, and zero secrets on snapshot disks.
- ☑ **5. Hibernation & Loop Caps:** Verified state survives DO eviction, Workflows sleep on `waitForEvent()`, and agent tool loops enforce a hard `maxSteps` ceiling.
- ☑ **6. End-to-End Observability:** Workers Logs active, `cf-aig-metadata` tagged with tenant & workflow IDs, and `npx wrangler tail` smoke-tested on staging.

Ready to Deploy Your First Edge Agent?

Scaffold in seconds with `npm create cloudflare@latest -- --template=cloudflare/agents-starter`, inspect SQLite locally with `npx wrangler dev`, and ship globally with `npx wrangler deploy`.