Build Your Own Always-On AI Agent

A practical guide to running a personal Claude agent as a persistent service — with memory, a chat interface, scheduled jobs, and tool access.

What You're Building

A personal AI agent that:

- Runs 24/7 on your home server or a cheap VPS
- Talks to you through Discord (or Telegram)
- Remembers things across conversations — not just the last session
- Runs scheduled jobs while you sleep (morning briefings, email checks, finance alerts)
- Has real tools: web search, bash, file access, external APIs

This isn't a weekend demo. This is a production-grade system you actually use every day.

Prerequisites

- A Linux machine with Node.js v22+ (home server, VPS, Raspberry Pi — anything always-on)
- A Claude Max subscription or Anthropic API key
- A Discord server you admin (or Telegram bot token)
- Basic TypeScript familiarity

Runtime cost: ~$0–15/month if using Claude Max subscription. Less than a coffee.

Architecture Overview

```
You (Discord/Telegram)
    ↓
Interface Layer (discord.ts)
    ↓
Agent Core (agent.ts) — Claude Agent SDK query loop
    ├── Loop detection
    ├── Daily budget cap
    └── Session persistence
    ↓
Memory System (memory.ts → memory-db.ts)
    ├── Core Memory   — permanent facts, always in context
    ├── Recall        — every conversation, FTS5 searchable
    └── Archival      — long-term knowledge, FTS5 + semantic search
    ↓
Tools
    ├── Built-in: Bash, Read, Write, WebSearch, WebFetch
    ├── Memory MCP tools (6 tools via subprocess)
    └── Subagents (email-triage, calendar-prep, etc.)
    ↓
Cron Scheduler (cron.ts)
    └── 17 scheduled jobs — morning briefing, news digest, finance alerts
```

Everything runs as a **systemd user service**. It starts on boot, restarts on crash, logs to journald.

Part 1: Project Skeleton

```
mkdir my-agent && cd my-agent
npm init -y
npm install @anthropic-ai/claude-agent-sdk better-sqlite3 croner discord.js
npm install -D @types/node @types/better-sqlite3
```

`tsconfig.json`:

```
{
  "compilerOptions": {
    "target": "ES2022",
    "module": "NodeNext",
    "moduleResolution": "NodeNext",
    "strict": true,
    "outDir": "dist"
  }
}
```

`package.json` (add to scripts):

```
{
  "type": "module",
  "scripts": {
    "start": "tsx src/index.ts"
  }
}
```

Part 2: Config

Create `config.json` (this is your runtime config — never commit it with secrets):

```
{
  "model": "claude-sonnet-4-6",
  "workspace": "/home/you/my-agent/workspace",
  "timezone": "America/Los_Angeles",
  "discord": {
    "botToken": "YOUR_BOT_TOKEN",
    "channels": {
      "main": "CHANNEL_ID",
      "dailyLogs": "CHANNEL_ID"
    }
  }
}
```

`src/config.ts`:

```
import { readFileSync } from "fs";
import { join, dirname } from "path";
import { fileURLToPath } from "url";

const CONFIG_PATH = join(dirname(fileURLToPath(import.meta.url)), "..", "con

export const config = JSON.parse(readFileSync(CONFIG_PATH, "utf-8")) as {
  model: string;
  workspace: string;
  timezone: string;
  discord?: {
    botToken: string;
    channels: Record<string, string>;
  };
};
```

> ***Production tip:*** *Don't put secrets in config.json directly. Use an age-encrypted vault or environment variables. Regenerate config.json from secrets at startup. That way config.json can be in your repo and secrets stay safe.*

Part 3: The Memory System

This is the most important part. Without good memory, your agent is useless the next day.

Three tiers:

Tier	What it stores	Size	When loaded
Core	Who you are, who the user is, current tasks	~500 chars	Every prompt
Recall	Full conversation log	Rolling 30 days	On-demand (FTS5 search)
Archival	Long-term facts, decisions, knowledge	Permanent	On-demand (FTS5 + semantic)

The key insight: **Core Memory is always in context. Everything else is searched on demand.** This keeps your system prompt tiny (~2–3KB) even after months of use.

`src/memory-db.ts` (abbreviated):

```
import Database from "better-sqlite3";

let db: Database.Database;

export function initMemoryDb(): void {
  db = new Database("/path/to/memory.db");
  db.pragma("journal_mode = WAL");

  db.exec(`
    CREATE TABLE IF NOT EXISTS core_memory (
      block TEXT PRIMARY KEY,
      content TEXT NOT NULL DEFAULT ''
    );
    CREATE TABLE IF NOT EXISTS recall (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      timestamp TEXT DEFAULT (datetime('now')),
      chat_id TEXT NOT NULL,
      role TEXT NOT NULL,
      content TEXT NOT NULL,
      summary TEXT
    );
    CREATE TABLE IF NOT EXISTS archival (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      timestamp TEXT DEFAULT (datetime('now')),
      content TEXT NOT NULL,
      tags TEXT,
      source TEXT
    );
  `);

  // FTS5 for fast text search
  db.exec(`CREATE VIRTUAL TABLE IF NOT EXISTS recall_fts
    USING fts5(content, summary, content='recall', content_rowid='id')`);
  db.exec(`CREATE VIRTUAL TABLE IF NOT EXISTS archival_fts
    USING fts5(content, tags, content='archival', content_rowid='id')`);

  // Seed default blocks
  db.prepare("INSERT OR IGNORE INTO core_memory (block, content) VALUES (?,
    .run("persona", "Your agent's name, personality, and role.");
  db.prepare("INSERT OR IGNORE INTO core_memory (block, content) VALUES (?,
    .run("user", "Your name, preferences, and context.");
  db.prepare("INSERT OR IGNORE INTO core_memory (block, content) VALUES (?,
    .run("tasks", "Current active tasks and priorities.");
}
```

`src/memory.ts` — builds the system prompt on every invocation:

```
import { getCoreMemory } from "./memory-db.js";
import { config } from "./config.js";

export function loadSystemPrompt(): string {
  const now = new Date();
  const date = now.toLocaleDateString("en-CA", { timeZone: config.timezone }
  const time = now.toLocaleTimeString("en-US", {
    timeZone: config.timezone, hour: "numeric", minute: "2-digit", hour12: t
  });

  // Core memory blocks — always loaded
  const blocks = getCoreMemory();
  const coreMemory = Object.entries(blocks)
    .map(([k, v]) => `### ${k}\n${v}`)
    .join("\n\n");

  return [
    `# Who You Are\n[SOUL.md contents here — your agent's personality]`,
    `# Core Memory\n${coreMemory}`,
    `# Current date: ${date}\n# Current time: ${time}`,
    `# Execution Rules
- DO THE THING. Execute directly, don't describe what you'd do.
- If a tool call fails twice, try a different approach.
- Prove completion with output, not descriptions.
- When you make a mistake, note it somewhere for next time.`,
  ].join("\n\n---\n\n");
}
```

Part 4: Memory as MCP Tools

The agent needs to be able to *update* its own memory. You expose this as MCP tools.

Why MCP as a subprocess? The Claude Agent SDK has a race condition on bidirectional channels (issue #148). Running the memory server as a standalone stdio JSON-RPC subprocess sidesteps it completely.

`src/memory-tools.ts` (using the SDK's `createSdkMcpServer`):

```
import { createSdkMcpServer, tool } from "@anthropic-ai/claude-agent-sdk";
import { z } from "zod/v4";
import { getCoreMemory, replaceCoreBlock, appendCoreBlock,
         searchRecall, insertArchival, searchArchival, deleteArchival } from

const memoryTools = [
  tool("core_memory_view", "View all core memory blocks.", {},
    async () => {
      const blocks = getCoreMemory();
      const text = Object.entries(blocks).map(([k, v]) => `=== ${k} ===\n${v
      return { content: [{ type: "text" as const, text }] };
    }
  ),

  tool("core_memory_replace",
    "Replace text in a core memory block. old_text must match exactly.",
    {
      block: z.enum(["persona", "user", "tasks"]),
      old_text: z.string(),
      new_text: z.string(),
    },
    async (args) => {
      const result = replaceCoreBlock(args.block, args.old_text, args.new_te
      return { content: [{ type: "text" as const, text: result.ok ? `Updated
    }
  ),

  tool("archival_store",
    "Store important facts for long-term retrieval.",
    {
      content: z.string(),
      tags: z.string().optional(),
    },
    async (args) => {
      const id = insertArchival(args.content, args.tags);
      return { content: [{ type: "text" as const, text: `Archived (id: ${id}
    }
  ),

  tool("archival_search",
    "Search long-term knowledge store.",
    { query: z.string(), limit: z.number().optional().default(10) },
    async (args) => {
      const results = searchArchival(args.query, args.limit);
      if (!results.length) return { content: [{ type: "text" as const, text:
      const text = results.map(r => `[id:${r.id}] ${r.content}`).join("\n\n"
      return { content: [{ type: "text" as const, text }] };
    }
  ),
];

export const memoryMcpServer = createSdkMcpServer({
  name: "my-agent-memory",
  version: "1.0.0",
  tools: memoryTools,
});

export const MEMORY_TOOL_NAMES = memoryTools.map(t => `mcp__my-agent-memory_
```

Part 5: The Agent Core

This is the main loop. Every message goes through here.

`src/agent.ts`:

```
import { query } from "@anthropic-ai/claude-agent-sdk";
import { config } from "./config.js";
import { loadSystemPrompt } from "./memory.js";
import { memoryMcpServer, MEMORY_TOOL_NAMES } from "./memory-tools.js";
import { insertRecall, getSession, setSession, getDailySpend, addDailySpend

export interface AgentResult {
  text: string;
  cost: number;
}

// Detect infinite tool loops (same call 3x in a row)
function detectLoop(history: string[]): boolean {
  if (history.length < 3) return false;
  const last3 = history.slice(-3);
  return last3[0] === last3[1] && last3[1] === last3[2];
}

export async function runAgent(
  prompt: string,
  chatId: string = "default",
  modelOverride?: string,
): Promise<AgentResult> {
  // Daily budget guard
  const today = new Date().toLocaleDateString("en-CA", { timeZone: config.ti
  const currentSpend = getDailySpend(today);
  if (currentSpend >= 20.0) {
    return { text: `Daily budget cap reached ($${currentSpend.toFixed(2)}).
  }

  const systemPrompt = loadSystemPrompt();
  const existingSession = getSession(chatId);

  const q = query({
    prompt,
    options: {
      model: modelOverride ?? config.model,
      systemPrompt,
      permissionMode: "bypassPermissions",
      maxTurns: 30,
      cwd: process.env.HOME ?? "/",
      mcpServers: { "my-agent-memory": memoryMcpServer },
      allowedTools: ["Bash", "Read", "Write", "WebSearch", "WebFetch", ...ME
      ...(existingSession ? { resume: existingSession } : {}),
    },
  });

  let resultText = "";
  let cost = 0;
  let sessionId = existingSession ?? "";
  const toolHistory: string[] = [];

  for await (const message of q) {
    if (message.type === "assistant") {
      resultText = "";
      for (const block of message.message.content) {
        if (block.type === "text") resultText += block.text;
        if (block.type === "tool_use") toolHistory.push(block.name);
      }
      if (detectLoop(toolHistory)) {
        await q.interrupt();
        resultText += "\n\n(Stopped: loop detected.)";
        break;
      }
    } else if (message.type === "result") {
      sessionId = message.session_id;
      if (message.subtype === "success") {
        cost = message.total_cost_usd;
        resultText = message.result || resultText;
      }
    }
  }

  // Persist session for continuity across restarts
  if (sessionId) setSession(chatId, sessionId);
  addDailySpend(today, cost);

  // Log to recall memory
  insertRecall(chatId, "user", prompt.slice(0, 500));
  insertRecall(chatId, "assistant", resultText.slice(0, 1000));
```

```
    return { text: resultText, cost };
  }
```

Part 6: Discord Interface

Create a Discord bot:

1. Go to discord.com/developers/applications
2. New Application → Bot → Enable all Privileged Gateway Intents
3. OAuth2 → Bot → add `bot` + `applications.commands` scopes, `Send Messages` + `Read Message History` permissions
4. Copy bot token → put in config.json

`src/discord.ts` (abbreviated):

```
import { Client, GatewayIntentBits, Events } from "discord.js";
import { config } from "./config.js";
import { runAgent } from "./agent.js";

const client = new Client({
  intents: [GatewayIntentBits.Guilds, GatewayIntentBits.GuildMessages, Gatew
});

export function startDiscord(): void {
  client.on(Events.MessageCreate, async (msg) => {
    if (msg.author.bot) return;
    if (!msg.mentions.has(client.user!)) return; // only respond when mentio

    const prompt = msg.content.replace(/<@!?\d+>/g, "").trim();
    if (!prompt) return;

    await msg.channel.sendTyping();

    const result = await runAgent(prompt, `discord:${msg.channelId}`);

    // Discord has a 2000-char limit — split if needed
    const chunks = result.text.match(/.{1,1900}/gs) ?? [result.text];
    for (const chunk of chunks) {
      await msg.reply(chunk);
    }
  });

  client.login(config.discord!.botToken);
}

// Post to a specific channel (for cron jobs)
export async function sendDiscordChannel(channelId: string, text: string): P
  const channel = await client.channels.fetch(channelId);
  if (!channel?.isTextBased()) return;
  const chunks = text.match(/.{1,1900}/gs) ?? [text];
  for (const chunk of chunks) {
    await (channel as any).send(chunk);
    }
  }
```

Part 7: Scheduled Jobs

Use `croner` — it handles timezones correctly and doesn't drift.

`src/cron.ts`:

```
import { Cron } from "croner";
import { execSync } from "child_process";
import { config } from "./config.js";
import { runAgent } from "./agent.js";
import { sendDiscordChannel } from "./discord.js";

export function startCron(): void {
  // Morning briefing — 4 AM daily
  new Cron("0 4 * * *", { timezone: config.timezone }, async () => {
    const prompt = `Morning briefing. Do all of these:
1. Search for today's weather in [your city]
2. Check email: run bash: gog gmail search --account you@gmail.com 'newer_th
3. System health: check disk and memory, report only if something is wrong
Format it tight. No filler.`;

    const result = await runAgent(prompt, "cron:briefing", "claude-haiku-4-5
    await sendDiscordChannel(config.discord!.channels.dailyLogs, result.text
  });

  // Add more jobs here
  new Cron("15 8 * * *", { timezone: config.timezone }, async () => {
    const result = await runAgent(
      "Search for notable AI news from the last 24 hours. Write a tight 5-bu
      "cron:ai-news",
      "claude-haiku-4-5-20251001"
    );
    await sendDiscordChannel(config.discord!.channels.main, result.text);
  });
}
```

Key pattern: Use `claude-haiku-4-5-20251001` for cron jobs (cheap, fast), reserve Sonnet for interactive conversations.

Part 8: Entry Point + systemd

`src/index.ts`:

```
import { existsSync, readFileSync, writeFileSync, unlinkSync } from "fs";
import { initMemoryDb } from "./memory-db.js";
import { startDiscord } from "./discord.js";
import { startCron } from "./cron.js";

// PID lockfile — prevents double-start
const LOCKFILE = "/tmp/my-agent.pid";
if (existsSync(LOCKFILE)) {
  const oldPid = parseInt(readFileSync(LOCKFILE, "utf-8").trim());
  try {
    process.kill(oldPid, 0);
    console.error(`Already running (PID ${oldPid}). Exiting.`);
    process.exit(1);
  } catch { /* stale lockfile, continue */ }
}
writeFileSync(LOCKFILE, String(process.pid));
process.on("exit", () => { try { unlinkSync(LOCKFILE); } catch {} });
process.on("SIGTERM", () => process.exit(0));

// Suppress known SDK race condition (issue #148)
process.env.CLAUDE_CODE_STREAM_CLOSE_TIMEOUT = "300000";
process.on("unhandledRejection", (reason) => {
  if (reason instanceof Error && reason.message === "ProcessTransport is not
  console.error("Unhandled rejection:", reason);
});

initMemoryDb();
startCron();
startDiscord();

console.log("Agent ready.");
```

`~/.config/systemd/user/my-agent.service`:

```
[Unit]
Description=My Personal AI Agent
After=network-online.target
Wants=network-online.target

[Service]
WorkingDirectory=/home/you/my-agent
ExecStart=/usr/bin/node --import tsx/esm src/index.ts
Restart=on-failure
RestartSec=10
Environment=NODE_ENV=production

[Install]
WantedBy=default.target
```

```
systemctl --user enable my-agent
systemctl --user start my-agent
systemctl --user status my-agent
journalctl --user -u my-agent -f  # watch logs
```

Part 9: Optional Upgrades

Semantic Memory Search

If your agent has been running for months and has hundreds of archival entries, FTS5 keyword search starts missing things. Add vector search:

```
npm install @xenova/transformers
```

`src/embeddings.ts`:

```
import { pipeline, env } from "@xenova/transformers";

env.cacheDir = "/path/to/models"; // cache locally

let _pipe: any = null;

export async function embed(text: string): Promise<number[]> {
  if (!_pipe) {
    _pipe = await pipeline("feature-extraction", "Xenova/all-MiniLM-L6-v2",
  }
  const output = await _pipe(text, { pooling: "mean", normalize: true });
  return Array.from(output.data as Float32Array);
}

export function cosine(a: number[], b: number[]): number {
  let dot = 0, na = 0, nb = 0;
  for (let i = 0; i < a.length; i++) { dot += a[i]*b[i]; na += a[i]*a[i]; nb
  return dot / (Math.sqrt(na) * Math.sqrt(nb) + 1e-10);
}
```

Add an `archival_vec` table (stores JSON-serialized 384-dim float arrays), expose an `archival_semantic_search` MCP tool. Works at 74 entries in ~50ms with no GPU.

Nightly Memory Summarization

Your recall log grows without bound. At 2:30 AM, run a job that:

1. Fetches recent recall entries (last 24h)
2. Runs a Haiku pass to extract key facts
3. Stores those facts as archival entries
4. Optionally prunes old recall rows

This is how the agent's knowledge compounds over time without the context window ballooning.

Self-Updating Rules

Create a `LEARNINGS.md` file — append-only, one rule per line. Load it into every system prompt. Tell the agent: *"When you make a mistake, append a rule here immediately."*

After a few weeks, it stops repeating the same errors.

Common Pitfalls

"My agent forgets everything on restart." Session IDs are ephemeral by default. Persist them to SQLite and pass `resume: sessionId` on every `query()` call.

"My agent loops forever on failing tool calls." Track the last N tool call names in an array. If the last 3 are identical, call `q.interrupt()` and break out of the loop.

"My cron job runs but nothing appears in Discord." The Discord client needs to be fully connected before you try to send to channels. Await a `ClientReady` event before starting cron jobs that post to Discord.

"I can't trigger my agent from inside a Claude Code session." Claude Code sets a `CLAUDECODE` env var that blocks nested agent spawns. Send a message to your agent via Discord instead — it's running in a clean process.

"The memory server subprocess crashes and takes down the whole agent." Catch errors in the MCP tool handlers. The subprocess being down should fail gracefully, not kill the parent.

The SOUL.md File

The most underrated part of this whole setup.

This is a markdown file you write once that defines who your agent *is* — not just what it can do. Personality, communication style, what it should never do, how it prioritizes your time, what matters to you.

Load it into every system prompt. The Claude Agent SDK will follow it more consistently than you'd expect.

Good starters:

- What's the agent's name and primary role?
- How should it communicate? (Direct? Formal? With dark humor?)
- What should it never do without asking first?

- What are your standing preferences? (No meeting before 10am, no replies to newsletters, always check cash flow before suggesting spending)

Final Stack

```
Runtime:      Node.js v22+, TypeScript, tsx
LLM:          Claude Sonnet 4.6 (interactive), Claude Haiku 4.5 (cron)
Memory:       SQLite + WAL + FTS5 + optional vector embeddings
Interface:    Discord (discord.js) or Telegram (telegraf)
Scheduler:    Croner
Auth:         Claude Max OAuth or ANTHROPIC_API_KEY
Service:      systemd user service
Secrets:      age-encrypted vault or env vars — never plain config files
```

Monthly cost at real usage:

- Claude Max subscription: $20/month flat (no per-token charges)
- VPS or home server electricity: $5–15/month
- Everything else: free

Total: **~$25–35/month** for an agent that never sleeps.

What to Build Next

Once the core is running:

1. **Email triage subagent** — runs on a cron, reads your inbox, flags only what matters
2. **Finance alerts** — sync bank data via Plaid, run a 14-day projection, alert on overdraft risk
3. **GitHub/news digest** — scrape trending repos and AI news overnight, summarized by morning
4. **Site uptime monitor** — ping your services every 30 minutes, alert on downtime

5. **Weekly build log** — auto-generate a summary of what changed this week and post it somewhere

The framework is the same for all of them: `runAgent(prompt, chatId, model)` → post result to Discord. The hard part is writing good prompts.

Built on the Claude Agent SDK. The memory architecture is inspired by MemGPT's three-tier model. The MCP subprocess pattern is a workaround for a known SDK race condition (issue #148 — hopefully fixed upstream eventually).