How to Build an Autonomous AI Agent on a Mac mini

A step-by-step guide to setting up an always-on AI agent with its own identity, crypto wallet, and three-tier inference fallback chain – powered by a Mac mini and the Morpheus decentralized AI network.

Introduction

This guide walks you through turning a Mac mini into a 24/7 autonomous AI agent you can text like texting a friend. The agent gets its own identity, its own crypto wallet, its own communication channels, and the ability to think using three independent inference paths: cloud AI, decentralized AI via the Morpheus network, and local models.

This is based on a real, working build – not theory. Every step has been tested on real hardware with real money and the common pitfalls are documented.

What you'll end up with:

- A Mac mini running headless, always-on, with a dedicated user account for the agent
- An AI agent with its own email, GitHub, crypto wallet, and phone number
- Three-tier inference fallback: cloud AI as primary, Morpheus (decentralized) as first fallback, local Ollama as backup
- A defined persona that makes the agent consistent and predictable
- On-chain financial guardrails via a Safe multi-sig wallet
- A kill switch that halts everything immediately

Why a Mac mini? Low power draw (~5W idle), Apple Silicon is fast for local inference, the OS keychain handles secrets natively, launchd keeps services alive across reboots, and Tailscale makes it reachable from anywhere. It's a $600 always-on server that draws less power than a light bulb.

Why Morpheus? Morpheus is a decentralized AI inference network. Instead of paying per-request to a cloud API, you stake MOR tokens to open inference sessions. The tokens are returned when the session expires – you're renting access, not spending money. This gives your

agent a way to think that doesn't depend on any single company's API keys, rate limits, or billing. If Claude goes down, if OpenAI rate-limits you, if your credit card expires – Morpheus keeps working.

Time to build: About a weekend for the basic agent. Add another day for the Morpheus integration.

Core Philosophy

These principles are non-negotiable. They're the difference between a toy demo and a real autonomous agent.

1. Identity Separation

The agent is NOT you. It has its own email, its own accounts, its own wallet. Your credentials never touch its machine. This isn't paranoia – it's the foundation of safe autonomy. If the agent is compromised, your identity isn't.

2. On-Chain Guardrails, Not Software Guardrails

Software promises are breakable – a bug, a prompt injection, a misconfiguration can bypass them. Financial limits must be enforced at a level the agent cannot override. A Safe multi-sig wallet with on-chain spending limits is enforced by the blockchain itself. The contract is the security boundary.

3. The Agent Proposes, the Human Approves

Every financial move gets explained in plain language before execution. The agent states: what it wants to do, why, what could go wrong, and how much is at risk. The human says yes or no. No action without understanding. Trust is built incrementally, not granted upfront.

4. Operational Transparency

Silence looks like death. The agent communicates what it's doing, especially when something takes time or might fail. One short message before a slow operation. Always.

5. Kill Switch

A phrase like **“STOP ALL IMMEDIATELY”** sent on any channel halts everything. Positions close, transactions stop, the agent goes quiet. No arguments. Non-negotiable.

Phase 0: Discovery

Before touching hardware, decide what you’re building. These questions shape your build plan.

What will the agent DO?

This determines which phases are mandatory vs. optional:

- **Research assistant / code reviewer:** Skip the Safe multi-sig phase. You’ll still need a wallet if using Morpheus (for staking), but not for DeFi.
- **DeFi agent / financial autonomy:** Full wallet + Safe multi-sig required.
- **General-purpose autonomous agent:** All phases recommended.

What’s your agent framework?

This guide is framework-agnostic. The important decisions – identity, communication, wallet, persona – work regardless of framework. OpenClaw, LangChain, CrewAI, AutoGPT, or a custom setup all work.

How will you communicate with the agent?

- **Signal** – most private, recommended. Requires a VoIP phone number.
- **Telegram** – simpler bot setup, less private.
- **Discord** – good for group interactions.
- **WhatsApp** – familiar, requires Meta business API.

What inference do you need?

The recommended setup is a three-tier fallback chain:

1. **Cloud AI** (Claude, GPT, etc.) – highest quality reasoning
2. **Morpheus** (decentralized) – no per-request cost, no single point of failure
3. **Local Ollama** (on a second Mac or the same machine) – free, private, always available

If any tier goes down, the next one picks up. Your agent always has a way to think.

What's the agent's name and persona?

This matters more than people think. A well-defined persona makes the agent consistent and predictable. Give it a real name – not "my assistant" or "hey bot."

Phase 1: Mac mini Setup

Goal: Mac mini running headless, accessible remotely, configured to never sleep.

Step 1: Initial macOS Setup

Unbox the Mac mini, complete macOS setup. Note the OS version for troubleshooting later.

Step 2: Create Two User Accounts

Create two macOS users, not one:

1. **Agent user** (e.g., `sam`) – Standard user. This is who the agent runs as. No admin privileges. It cannot `sudo`, install system software, or modify other accounts.
2. **Admin user** (e.g., `maint-admin`) – Administrator. This is your remote maintenance account. Used for system updates, service configuration, and troubleshooting. The agent never touches this account.

System Settings > Users & Groups > Add User for each. Make the admin user an Administrator; make the agent user a Standard account.

Why two accounts? If the agent is compromised (prompt injection, malicious tool use, supply chain attack), a non-admin agent user limits the blast radius. It can't install rootkits, modify system services, read other users' keychains, or escalate to root. The admin account is your emergency access – protected by a separate password and SSH key.

Step 3: Configure Always-On Behavior

- **Energy Saver > Never sleep.** The Mac mini must stay awake 24/7.
- **Enable auto-restart** after power loss (System Settings > General > Startup)
- **Enable auto-login** to the agent's user account
- **Plug in an HDMI dummy dongle** (~$10). Without a display connected, macOS may throttle the GPU and some services behave differently. A $10 dongle tricks it into thinking a monitor is attached.

Step 4: Enable Remote Access

- **SSH:** System Settings > General > Sharing > Remote Login. Then configure key-only authentication – edit `/etc/ssh/sshd_config` to disable `PasswordAuthentication`.
- **SSH access for non-admin users:** macOS restricts SSH to admin users by default. After creating a non-admin agent user, explicitly add it to the SSH access group:

  ```
  sudo dseditgroup -o edit -a AGENT_USERNAME -t user com.apple.access_ssh
  ```

  Do this BEFORE removing the agent from admin. If you remove admin first, you'll be locked out. Always verify the new access path works before closing the old one.
- **SSH keys for both users:** Install your public key in `~/.ssh/authorized_keys` for both the agent user AND the admin user. Test both connections before proceeding. If you can't SSH in as both users, stop and fix it.
- **Tailscale:** Install Tailscale for secure remote access from anywhere, not just your home network. Note the Tailscale IP – this is how you reach the Mini remotely.
- **Optional:** Enable Screen Sharing for emergencies.

Step 5: Security Basics

- Enable the macOS firewall (System Settings > Network > Firewall)
- Verify the agent user is NOT in the admin group: `dseditgroup -o checkmember -m AGENT_USERNAME admin` should say “no”
- All secrets go in the macOS Keychain – never in plaintext files

Step 6: Install Base Tools

```
# Homebrew
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

# Node.js LTS + Git
brew install node git

# Framework-specific dependencies as needed
```

Verification Checklist

- [ ] Can SSH into the Mac mini as the agent user via Tailscale
- [ ] Can SSH into the Mac mini as the admin user via Tailscale
- [ ] Mac mini does not sleep
- [ ] Agent user is a Standard account (not admin)
- [ ] Admin user is an Administrator account
- [ ] Agent user is in `com.apple.access_ssh` group
- [ ] SSH uses key-only authentication
- [ ]

Firewall is enabled

Phase 2: Agent Identity

Goal: The agent exists as its own entity with isolated credentials. No connection to you.

2a. Email

Create a Proton Mail account for the agent. Proton Mail is recommended because it's privacy-focused and Proton Bridge gives you headless IMAP/SMTP access on the Mac mini.

- Use this email for ALL agent registrations (GitHub, services, etc.)
- Do NOT use your personal email. The agent's identity must be fully separate.
- Set up Proton Bridge on the Mac mini (CLI/noninteractive mode):
  - IMAP: 127.0.0.1:1143 / SMTP: 127.0.0.1:1025
  - Store the bridge password in the macOS Keychain

2b. Secret Management

Use the built-in macOS Keychain via the `security` CLI:

```
# Store a secret
security add-generic-password -s "service-name" -a "account" -w "password"

# Retrieve a secret
security find-generic-password -s "service-name" -a "account" -w
```

Free, encrypted on disk, no subscriptions, already installed, CLI-accessible.

Gotcha: The login keychain locks in SSH sessions. Run `security unlock-keychain -p "password" ~/Library/Keychains/login.keychain-db` before accessing secrets over SSH. **Note:** This puts the Keychain password in your shell history. Either clear the history afterward (`history -c`) or use `security unlock-keychain` without `-p` to be prompted interactively.

Dedicated Keychains for Automated Scripts

For scripts that run unattended (cron jobs, daemons), create a dedicated keychain instead of using the login keychain. This isolates the automated credentials from everything else:

```
# Generate a random password for the dedicated keychain
KEYCHAIN_PASS=$(openssl rand -base64 24)

# Create it
security create-keychain -p "$KEYCHAIN_PASS" ~/.my-automation.keychain-db
security set-keychain-settings ~/.my-automation.keychain-db

# Store the secret in the dedicated keychain
security add-generic-password -a account -s service-name -w "SECRET" ~/.my-
automation.keychain-db

# Save the keychain password to a file (readable only by the agent user)
echo "$KEYCHAIN_PASS" > ~/.my-keychain-pass
chmod 0400 ~/.my-keychain-pass
chflags uchg ~/.my-keychain-pass
```

Your automated script unlocks the dedicated keychain using the password file, reads the secret, and continues. Even if the password file is compromised, the attacker only gets access to secrets in that dedicated keychain – not the login keychain (which holds email credentials, other API keys, etc.). Point your scripts at the dedicated keychain via environment variables:

```
SAFE_KEYCHAIN_DB=/Users/agent/.my-automation.keychain-db
SAFE_KEYCHAIN_PASS_FILE=/Users/agent/.my-keychain-pass
```

Rule: No API keys, passwords, or private keys in plaintext. Ever. Not in config files, not in environment variables, not in scripts. Keychain only. The one exception is the dedicated keychain's password file – and that file should contain ONLY the keychain password, not the secrets themselves.

2c. GitHub

If the agent will write or review code:

- Create a GitHub account using the agent's Proton email
- Choose a username that fits the agent's persona
- Store the password in the Keychain
- Set up `gh` CLI auth on the Mac mini
- Configure git identity:

```
git config --global user.name "Your Agent Name"
git config --global user.email "agent@pm.me"
```

2d. Crypto Wallet

The agent needs a wallet for two things: staking MOR on the Morpheus network, and (optionally) executing DeFi strategies.

- Generate a new wallet using Foundry: `cast wallet new-mnemonic`
- Store the private key in the macOS Keychain – NOT in a config file
- Write the mnemonic seed phrase on paper. Store it physically secure. No digital copies.
- Fund with a small amount of ETH for gas on Base (an Ethereum L2 chain)
- This wallet (EOA) is a SIGNER on a Safe multi-sig – it does NOT hold operational funds directly

2e. Safe Multi-Sig Wallet

If the agent will handle money beyond MOR staking:

- Deploy a Safe wallet on Base with two signers:
  - The agent's key (generated above)
  - Your key (you remain a co-signer)
- Enable the AllowanceModule for daily transfer caps:
  - Register the agent as a delegate
  - Set daily token allowances (e.g., 50 MOR/day, 0.05 ETH/day)

    - Allowances reset automatically every 24 hours
- The agent draws funds within its daily allowance without co-signatures
- Anything above the allowance – or any admin change – requires the human's co-signature
- The safe-treasury repo has tested scripts for the full workflow: `safe-deploy.mjs`, `safe-configure.mjs`, `safe-propose.mjs`, `safe-refill.mjs`, `safe-status.mjs`
- After deployment, run `node scripts/safe-status.mjs` to verify configuration and check balances, allowance usage, and pending transactions (read-only, no private key needed)
- **The Safe IS the guardrail.** On-chain enforcement, not software promises.
- **Future:** For DeFi operations beyond simple transfers (e.g., swaps, lending), the Zodiac Roles Modifier v2 provides contract-level permission scoping by address, function selector, and parameter values.

Separation Checklist

- [ ] Agent has its own macOS user account (Standard, not admin)
- [ ] Separate admin account exists for system maintenance
- [ ] Agent has its own email (no connection to you)
- [ ] Agent has its own GitHub (no connection to you)
- [ ] Agent has its own Keychain secrets
- [ ] Automated scripts use a dedicated keychain (not the login keychain)
- [ ] Agent has its own crypto wallet
- [ ] Your credentials are NOT accessible from the agent's user account
- [ ] Agent cannot sudo or escalate privileges

Phase 3: Communication Channel

Goal: You can text the agent like texting a person.

Recommended: Signal via VoIP

Signal is the most private option and fits the separate-identity philosophy. The agent gets its own phone number that isn't tied to your personal accounts.

Step 1: Get a VoIP Number

JMP.chat is recommended:

- No personal info required to sign up
- Accepts cryptocurrency payment
- ~$4/month for a US or Canadian phone number
- SMS delivered via XMPP (you need an XMPP client to receive verification codes during setup)

Step 2: Install signal-cli

```
brew install signal-cli
```

Gotcha (ARM Macs): If your agent framework auto-downloads signal-cli, it may grab the Linux x86-64 binary instead of the macOS ARM binary. Always install via Homebrew and symlink to wherever the framework expects it.

Step 3: Register with Signal

Signal requires a captcha for new registrations. On a headless Mac mini, you can't solve it locally. Do it on your laptop:

1. Visit https://signalcaptchas.org in a browser
2. Solve the captcha
3. Copy the token from the `signalcaptcha://` redirect URL

Then register on the Mac mini:

```
signal-cli -u +YOUR_JMP_NUMBER register --captcha CAPTCHA_TOKEN
```

Signal sends a verification code via SMS to the JMP number. It arrives in your XMPP client.

```
signal-cli -u +YOUR_JMP_NUMBER verify CODE
```

Step 4: Pair Your Personal Signal

So you can text the agent from your phone, pair your Signal account via the agent framework's setup wizard or manually via signal-cli.

Step 5: Test

Text the agent's number from your phone. If the agent framework is running, it responds. This is the moment – you're texting an AI that lives on a Mac mini in your closet.

Alternatives

Platform	Setup Complexity	Privacy	Phone Number Required
Signal via VoIP	Medium	High	Yes (VoIP)
Telegram Bot	Low	Medium	No
Discord Bot	Low	Low	No

Phase 4: Agent Framework + Model Routing

Goal: The agent is live, thinking, and responding through three independent inference paths.

4a. Install the Agent Framework

Follow your chosen framework's installation guide. Configure it under the agent's macOS user (not yours). Verify it starts and listens for incoming messages from your communication channel.

4b. Cloud AI (Primary)

- Store your API key in the macOS Keychain, not in a config file
- Configure the framework to use your preferred primary model (Claude Opus recommended for complex reasoning)
- This is the highest-quality tier – use it for important decisions and nuanced conversations

4c. Morpheus Decentralized Inference

This is what makes the setup resilient. Morpheus gives your agent inference that doesn't depend on any single company.

How Morpheus Works

Morpheus is a peer-to-peer network of AI providers. Instead of paying per-request, you temporarily lock ("stake") MOR tokens to open a session with a provider. A 7-day session costs about 2 MOR. When the session expires, **your tokens come back**. You're renting access, not spending money.

Install the Proxy-Router

The proxy-router connects your Mac mini to the Morpheus P2P network. It talks to the blockchain to find providers, manage sessions, and route requests.

- Download the proxy-router from the Morpheus releases page
- Configure `~/morpheus/.env` with a Base RPC endpoint. The proxy-router also needs the wallet private key – it reads it from the `.env` file at startup. This is a known compromise: the proxy-router doesn't support Keychain natively. Mitigate by restricting `.env` permissions (`chmod 0600`) and never committing it to git.
- Start the router – it connects to the blockchain, sees your MOR balance, and waits for instructions
- **Bind to 127.0.0.1 only** – the router should NOT be accessible from the network

Stake MOR

```
# The proxy-router handles staking through its API
# Approve MOR for the Diamond contract, then open a session
# Sessions last 7 days, tokens return automatically on expiry
```

Fund the agent's wallet with MOR tokens on Base. Each session stakes ~2 MOR, so ~50 MOR (~$10 at time of writing, but check current price) gives you enough for many concurrent sessions. The tokens are reusable – they return when each session expires.

Install the Proxy Bridge

Between the router (which speaks blockchain) and your agent framework (which speaks standard OpenAI-compatible API), you need a translator. This is a small Node.js script that listens on a local port and converts standard AI requests into Morpheus-routed requests.

Your agent framework just talks to `http://127.0.0.1:8083` like any other AI API endpoint. It doesn't need to know about blockchains, staking, or sessions.

Wire into the Framework

Add Morpheus as a model provider in your framework's config, pointing at the proxy bridge. Set it as the first fallback after your cloud provider.

What to Expect

- Models available on Morpheus vary by what providers are running. Expect open-weight models like Llama, Qwen, DeepSeek, and others.
- Provider availability is better during daytime hours.
- Some models are more responsive than others. Test different model IDs.
- **Gotcha:** The default `models-config.json` ships with incorrect model IDs. Verify against the actual network.

4d. Local Ollama (Backup)

If you have a second Mac with significant RAM (or the Mac mini itself has enough), run Ollama for always-available local inference.

- Install Ollama: `curl -fsSL https://ollama.com/install.sh | sh`

- Pull models appropriate for your RAM:
  - 16GB: 7B–14B parameter models (Phi-4, Qwen 14B)
  - 32GB: 14B–32B parameter models (Qwen 32B, CodeLlama 34B)
  - 64GB+: 70B parameter models (Llama 3.3 70B runs comfortably on a Mac Studio M3 Ultra 96GB)
- Expose on network: `OLLAMA_HOST=0.0.0.0 ollama serve`
- Add the Ollama endpoint to your framework's model config as the last fallback

The Fallback Chain

```
1. Claude Opus (cloud)     -- best quality, paid subscription
2. Morpheus (decentralized) -- no per-request cost, tokens returned after use
3. Ollama (local)           -- free, private, always available
```

If Claude is down or rate-limited, the agent falls back to Morpheus. If Morpheus providers are sparse (late night), it falls back to local Ollama. The agent always has a way to think.

Verification

- [ ] Send a message to the agent, get a response
- [ ] Confirm which model responded (check framework logs)
- [ ] Test failover by temporarily disabling the primary provider

---

Phase 5: Persona + Workspace Files

Goal: The agent has a defined personality, knows its boundaries, and behaves consistently.

The agent's behavior is defined by text files in its workspace. Change the files, change the behavior. This is where your agent stops being a generic chatbot and becomes a specific entity.

SOUL.md – Who the Agent IS

The character definition. This is the most important file. It should cover:

- Personality traits, communication style, values
- How it approaches problems
- Strengths and honest knowledge gaps
- Sense of humor (if any)
- Relationship to the human (partner, not servant)

Writing tips:

- Write in first person from the agent's perspective
- Be specific – "dry humor, deadpan delivery" is better than "sometimes funny"
- Include flaws and growth edges – perfect agents are boring and untrustworthy
- Define how the agent handles mistakes (surface them, don't hide them)
- Define its relationship to money if it will handle finances (cautious? aggressive? methodical?)

IDENTITY.md – The Facts

Name, pronouns, accounts, wallet address, what kind of entity it is. The hard data.

USER.md – Who the Human Is

Name, timezone, communication preferences. How to work with them. What they're good at and where they need the agent's help. The golden rule: **"If the human doesn't understand it, the agent doesn't do it."**

AGENTS.md – The Rules

- Financial guardrails (propose, explain, wait for approval)
- Kill switch phrase and behavior
- Escalation rules (what's auto-approved vs. requires permission)
- Spending limits (reference the Safe configuration)
- Operational transparency rules (don't go dark)
- Memory management instructions (daily notes + curated long-term memory)

- Heartbeat behavior (what to check, when to reach out, when to stay quiet)

TOOLS.md – Permissions

- What tools the agent can use freely (file system, web search, memory)
- What requires notification (shell commands, git operations)
- What requires explicit approval (financial transactions, sending emails)

MEMORY.md – Continuity

The agent wakes up fresh each session. This file is its long-term memory. Seed it with: identity facts, infrastructure details (ports, IPs, model names), key relationships, and lessons learned. The agent updates this file over time as it learns.

Deploy

Place all files in the agent's workspace directory. Verify by texting the agent "who are you?" – it should respond in character, in its own voice. Iterate over time through conversation and file updates. The persona develops and sharpens with use.

Phase 6: Security Hardening

Goal: The agent is safe to leave running unattended.

Security Model – Defense in Depth

Layer	What It Protects
Hardware isolation	Dedicated Mac mini, separate macOS user
Account separation	Non-admin agent user + separate admin user
Identity isolation	Own email, wallet, accounts, keychain
Credential management	macOS Keychain only, no plaintext secrets
Credential isolation	Dedicated keychain for automated scripts (not login keychain)
Wallet security	Safe multi-sig with on-chain spending limits
Transaction limits	AllowanceModule daily caps per token
Network security	Services bound to 127.0.0.1, firewall enabled
Prompt injection defense	Input scanning, severity scoring
Uptime monitoring	Health checks, auto-restart via cron or launchd
Kill switch	Immediate halt phrase on any channel
Audit trail	All actions logged, agent explains every decision

Hardening the Agent User

The agent user should be a Standard (non-admin) macOS account. This limits what a compromised agent can do:

- **No sudo:** Cannot install software, modify system config, or access other users' files
- **No keychain cross-access:** Cannot read the admin user's keychain or other users' secrets
- **Limited blast radius:** A prompt injection or supply chain attack is contained to the agent's home directory

Order of operations matters. When hardening a machine that already has the agent running as admin:

1. Create the admin user and install SSH keys for it

2. Verify SSH works for the admin user
3. Add the agent user to `com.apple.access_ssh`
4. Verify SSH works for the agent user
5. **Only then** remove the agent from the admin group
6. Verify SSH still works for both users

Never remove an access path until the replacement is tested. Getting this wrong locks you out of a headless machine.

Dedicated Keychain for Automated Scripts

If the agent runs cron jobs or daemons that need secrets (e.g., a wallet key for auto-refilling), use a dedicated keychain rather than the login keychain. The login keychain on a headless Mac is often locked and contains everything – email credentials, API keys, the wallet key. A dedicated keychain:

- Contains only what the automated script needs (e.g., just the wallet key)
- Has its own random password stored in a file (`chmod 0400` , `chflags uchg`)
- If compromised, doesn't expose email, API keys, or other secrets

See Phase 2b for setup instructions.

Hardening Checklist

- [ ] Agent user is Standard (not admin): `dseditgroup -o checkmember -m AGENT admin` returns "no"
- [ ] Separate admin user exists and can SSH in
- [ ] Agent user is in `com.apple.access_ssh` group
- [ ] All services (proxy-router, bridges) bound to 127.0.0.1, not 0.0.0.0
- [ ] SSH key-only, password auth disabled
- [ ] macOS firewall enabled

- [ ] No plaintext secrets anywhere (verified with `grep -r "sk-" ~/.openclaw/` etc.)
- [ ] Gateway/API tokens rotated from defaults
- [ ] Automated scripts use dedicated keychain, not login keychain
- [ ] Safe wallet deployed with spending limits (if applicable)
- [ ] Critical services run via cron (headless) or launchd (with GUI session)
- [ ] Proxy-router listens on localhost only

Phase 7: Ongoing Operations

Once the agent is live, these are the ongoing responsibilities.

For the Human

- Monitor the agent's daily reports
- Approve/reject financial proposals
- Review and give feedback on the agent's decisions
- Gradually expand auto-approved actions as trust builds
- Renew subscriptions, fund gas, and cycle MOR sessions as needed

For the Agent

- Daily memory maintenance (write daily notes, periodically curate long-term memory)
- Monitor Morpheus session status (renew before expiry)
- Monitor positions and infrastructure health
- Proactive communication (don't wait to be asked)
- Track its own performance

Scheduled Tasks on Headless Macs

Use `crontab -e` (as the agent user) for scheduled tasks like wallet refills. On headless Macs without a GUI session, macOS LaunchAgents cannot load – they require the GUI domain (`gui/UID`), which only exists when a user is logged in at the graphical console. An HDMI dummy dongle and auto-login may not be sufficient; macOS may still not create a full GUI session without a real display.

Cron works in any session type:

```
# Example: refill hot wallet every 6 hours
0 */6 * * * cd /Users/agent/safe-treasury && /opt/homebrew/opt/node@22/bin/node scripts/safe-refill.mjs >> /Users/agent/morpheus/data/logs/refill.log 2>&1
```

Note: Use absolute paths to `node` in crontab – cron doesn't load your shell profile, so `node` won't be on the PATH.

Trust-Building Timeline

Period	Level	Description
Week 1–2	Watch-only	Agent reports what it WOULD do. No execution.
Week 3–4	Small budget	Agent executes within tight limits. Human reviews everything.
Month 2+	Expanded autonomy	Auto-approved actions increase. Limits widen.
Ongoing	Continuous	Trust is earned by consistent good judgment, not by time passing.

Reference: Cost Structure

Item	Cost	Type
Mac mini M4 (16GB)	~$600	One-time
HDMI dummy plug	~$10	One-time
Proton Mail Plus (for Bridge)	~$4/month	Recurring
VoIP number (JMP.chat)	~$4/month	Recurring
Cloud AI (Claude subscription)	~$20–200/month	Recurring
ETH for gas (Base)	~$10–50	Refillable
MOR for staking	~$10	Reusable (tokens return)
Ollama (local inference)	Free	Needs a Mac with RAM

Total recurring: ~$28–208/month depending on your cloud AI subscription tier.

The Morpheus angle: Most of that recurring cost is the cloud AI subscription. As Morpheus provider quality improves, you can shift more inference to the decentralized network – where the cost is MOR staking, not monthly bills, and the tokens come back.

Reference: Common Gotchas

These are real issues encountered during the original build:

1. signal-cli platform mismatch. Agent frameworks may auto-download the Linux x86-64 binary on ARM Macs. Install via Homebrew (`brew install signal-cli`) and symlink to wherever the framework expects it.

2. Signal captcha on headless machines. You can't solve a captcha without a browser. Solve it on your laptop at signalcaptchas.org, copy the token, paste it into the `signal-cli register` command on the Mac mini.

3. macOS Keychain locks in SSH sessions. Run `security unlock-keychain` before accessing secrets over SSH. This catches everyone the first time.

4. LaunchAgents don't work on headless Macs. LaunchAgents require the GUI domain (`gui/UID`), which macOS only creates when a user is logged in at the graphical console. On a headless Mac mini – even with auto-login enabled and an HDMI dummy dongle – the GUI session may not exist. `launchctl bootstrap gui/UID` will fail with "Could not find domain for." Use cron instead for scheduled tasks; it works in any session type.

5. Removing a user from admin kills SSH access. macOS Remote Login defaults to admin users only. If you remove the agent from admin without first adding it to `com.apple.access_ssh` , SSH connections will be accepted (key auth succeeds) then immediately dropped. This locks you out of a headless machine. Always add to `com.apple.access_ssh` first, test SSH, then remove from admin.

6. Morpheus proxy-router overflow bug (v5.11.0). The `increaseAllowance` + `maxUint256` combo caused session open failures. Fix: reset allowance to 0 with `cast send` , let the router manage allowances incrementally. Check for v5.11.1+ which may have this fixed – always use the latest release from the Morpheus GitHub.

7. Morpheus model IDs are wrong out of the box. The shipped `models-config.json` has incorrect model IDs. Verify against the actual Morpheus network before relying on them.

8. Apple Silicon local inference is fast. A Mac Studio M3 Ultra with 96GB runs Llama 3.3 70B comfortably. Even a Mac mini M4 with 16GB runs 14B models well. Don't underestimate local inference on Apple Silicon.

9. Heredocs don't paste cleanly over SSH. Multi-line heredocs corrupt when pasted into SSH terminals. Use single-line echo commands or `scp` files directly to the Mac mini.

10. Public RPC stale nonce after Safe transactions. Public RPC endpoints (e.g., BlastAPI) serve stale nonce reads immediately after a transaction confirms on-chain. Sequential Safe transactions fail with `GS026` (invalid signature) or `GS013` (tx failed) because the next transaction hash is computed with a stale nonce. Fix: add a 5-second delay between sequential Safe transactions, or use a private RPC with consistent reads.

Build Progress Tracker

Use this checklist to track your build across sessions:

- [ ] Phase 0: Discovery – what are we building?
- [ ] Phase 1: Mac mini setup – headless, remote, always-on
- [ ] Phase 2: Agent identity – email, secrets, wallet, GitHub
- [ ] Phase 3: Communication channel – Signal / Telegram / Discord
- [ ] Phase 4: Agent framework + model routing (cloud + Morpheus + local)
- [ ] Phase 5: Persona + workspace files
- [ ] Phase 6: Security hardening
- [ ] Phase 7: Ongoing operations started

Every step in this guide was tested on real hardware with real money. Built over 5 days, February 2026.