OpenClaw Trades

THE COMPLETE GUIDE TO BUILDING YOUR AI CRYPTO TRADING ASSISTANT

Version 1.0 · March 2026 · openclawtrades.com

> ***This guide assumes you're serious.*** *Not get-rich-quick serious — serious as in you understand that consistent edge in crypto comes from better information, faster execution, and fewer emotional mistakes. That's exactly what this guide helps you build.*

TABLE OF CONTENTS

Chapter 1: Introduction — Why Your Setup Is Costing You Money {#introduction}

THE PROBLEM WITH HOW MOST TRADERS WORK

You're checking five tabs. CoinGecko, TradingView, your exchange, Twitter, and some Telegram alpha group you half-trust. You've got a spreadsheet that's three weeks out of date. You set a price alert on your phone last month and forgot about it until it dinged at 3am.

This is not a trading setup. This is a mess.

The traders who consistently outperform aren't smarter than you. They have better systems. They've automated the mechanical parts — monitoring, alerting, data collection — so their actual mental bandwidth goes toward decision-making, not information gathering.

OpenClaw is a personal AI agent platform that runs on your own machine. It's not a cloud SaaS with a dashboard you'll forget to check. It's an agent that lives on your computer, connects to your Telegram, watches what you tell it to watch, and alerts you when something needs your attention.

This guide shows you how to build that system. By the end, you'll have:

- Automated price alerts for any asset, any condition
- A portfolio tracker that reports to you every morning
- Direct API connections to your exchanges
- A signal monitoring agent that runs 24/7
- A library of prompts that help you think clearly about trades

WHAT OPENCLAW IS (AND ISN'T)

OpenClaw is:

- A local AI agent runtime you install on your Mac, Windows, or Linux machine
- Connected to Claude (Anthropic's AI) for intelligence
- Integrated with Telegram so you can interact with it from anywhere

- Extensible via "skills" — small modules that give it new capabilities
- Completely under your control — your machine, your data

OpenClaw is not:

- A trading bot that executes trades automatically (it can be extended to do this, but that's not the default)
- A signal service telling you what to buy
- A cloud platform with someone else's access to your API keys
- A replacement for your own judgment

The goal is augmentation, not automation of your decisions. You still decide. The agent makes sure you have the right information at the right time.

WHY THIS BEATS ALTERNATIVES

vs. TradingView Alerts: TradingView alerts are one-way. They ping your phone. OpenClaw alerts are two-way — you can ask follow-up questions, request additional context, and have a conversation about what you're seeing.

vs. 3Commas / Pionex bots: Those platforms hold your API keys on their servers. You're trusting a third party with exchange access. OpenClaw runs locally. Nothing leaves your machine unless you explicitly send it.

vs. Hiring someone to monitor for you: Obviously cheaper. Also available 24/7 without sick days or timezone issues.

vs. Doing it yourself: You can't watch markets 24 hours a day. An agent can.

HOW THIS GUIDE IS STRUCTURED

Each chapter builds on the previous one. If you're new to OpenClaw, read in order. If you're already set up, jump to the chapter that's relevant to what you're building.

Code is real. Prompts are tested. Everything in here can be implemented today.

Chapter 2: Getting Started — Installation & Telegram Setup {#getting-started}

SYSTEM REQUIREMENTS

OpenClaw runs on your local machine. Minimum specs:

- **macOS:** 12 Monterey or later (Apple Silicon or Intel)
- **Windows:** Windows 10/11 (64-bit)
- **Linux:** Ubuntu 20.04+, Debian 11+, or equivalent
- **RAM:** 4GB minimum, 8GB recommended
- **Storage:** 2GB free space
- **Internet:** Required for AI model calls and market data
- **Node.js:** v18 or later (installer handles this)

INSTALLING OPENCLAW

OpenClaw is distributed as an npm package. You'll need Node.js v18 or later installed first — download it from nodejs.org if you don't have it.

macOS / Linux

Open Terminal and run:

```
npm install -g openclaw
```

That's it. Once installed, run the setup wizard to configure your API keys and workspace:

```
openclaw configure
```

The wizard walks you through everything: AI model keys, workspace location, and channel setup. Follow the prompts.

Start the gateway (the background service that keeps your agent running):

```
openclaw gateway
```

To install it as a persistent background service that survives reboots:

```
openclaw gateway install
```

Verify everything is running:

```
openclaw gateway status
```

You'll see the gateway status, port (default: 18789), and connected channels.

Windows

Install Node.js from nodejs.org, then open PowerShell and run:

```
npm install -g openclaw
openclaw configure
openclaw gateway
```

To install as a Windows service:

```
openclaw gateway install
```

CONNECTING YOUR AI MODEL

OpenClaw needs an AI model to power reasoning. The recommended option is Claude via Anthropic's API.

1. Get an API key at console.anthropic.com
2. Run the configure wizard and enter your key when prompted:

```
openclaw configure
```

The wizard will ask for your provider (select Anthropic), then your API key. It handles everything else automatically — model selection, defaults, and workspace setup.

If you've already run the wizard and just need to update your model settings, run:

```
openclaw configure --section model
```

Verify everything is connected:

```
openclaw status
```

You should see your gateway, AI model, and any connected channels all showing as active.

SETTING UP TELEGRAM

This is the part that makes everything click. Telegram becomes your trading terminal — a way to interact with your agent from anywhere, on any device.

Step 1: Create a Telegram Bot

1. Open Telegram and search for **@BotFather**
2. Send `/newbot`
3. Give it a name (e.g., "My Trading Agent")
4. Give it a username ending in `bot` (e.g., `mytradingagent_bot`)
5. BotFather gives you a token like: `1234567890:ABCdefGHIjklMNOpqrSTUvwxYZ`

Save this token — you'll need it.

Step 2: Connect the Bot to OpenClaw

Run the channel configuration wizard:

```
openclaw configure --section channels
```

Select Telegram, then paste your bot token when prompted. The wizard saves it to your config automatically.

Restart the gateway to pick up the new config:

```
openclaw gateway restart
```

Step 3: Pair Your Account

Open your new bot in Telegram and send any message. The bot will respond with a pairing code like:

```
Your Telegram user id: 5920434871
Pairing code: B4CAPQF9

Ask the bot owner to approve with:
openclaw pairing approve telegram B4CAPQF9
```

Back in your terminal, run the approval command with your code:

```
openclaw pairing approve telegram B4CAPQF9
```

Your Telegram is now linked. Any message you send to the bot goes directly to your agent.

Step 4: Configure Your Workspace

Navigate to your workspace:

```
cd ~/.openclaw/workspace
```

```
ls
```

You'll see files like `AGENTS.md`, `SOUL.md`, `USER.md`. These define your agent's behavior and identity. Edit `USER.md` to add your personal context:

```
# USER.md
- Name: Your Name
- Timezone: Europe/Amsterdam
- Primary exchange: Binance
- Portfolio: BTC, ETH, SOL (primary positions)
- Risk tolerance: Medium — no more than 5% per trade
```

This context gets loaded with every session, so your agent knows who you are and what you care about.

VERIFYING YOUR SETUP

Run this quick check:

```
openclaw status
```

You'll see a full status table showing your gateway, channels, sessions, and any warnings. Look for your Telegram channel showing `OK` in the Channels section. If anything looks wrong, `openclaw status --deep` runs a full probe with diagnostics.

If Telegram shows as connected and you can send a message to your bot and get a reply — you're good. Move on to Chapter 3.

Chapter 3: Your First Trading Skill — Price Alerts That Actually Work {#first-skill}

WHAT IS A SKILL?

A skill is a small module that gives your agent new capabilities. Skills live in your workspace under a `skills/` directory. Each skill has a `SKILL.md` file that the agent reads to understand how to use it.

The architecture is simple:

- `SKILL.md` — instructions for the agent
- Supporting scripts — the actual code that fetches data, calls APIs, etc.
- Any assets or config files the skill needs

You don't need to be a developer to use skills. But understanding the structure helps you customize them.

BUILDING A PRICE ALERT SKILL

Let's build a skill that monitors crypto prices and alerts you when conditions are met.

Directory Structure

Create the skill directory:

```
mkdir -p ~/.openclaw/workspace/skills/price-alert
cd ~/.openclaw/workspace/skills/price-alert
```

The Alert Script

Create `check-price.js`:

```
#!/usr/bin/env node
// price-alert/check-price.js
// Fetches current price from CoinGecko (free, no API key required)

const https = require('https');

const COIN_MAP = {
  BTC: 'bitcoin',
  ETH: 'ethereum',
  SOL: 'solana',
  BNB: 'binancecoin',
```

```
  XRP: 'ripple',
  ADA: 'cardano',
  AVAX: 'avalanche-2',
  DOT: 'polkadot',
  LINK: 'chainlink',
  MATIC: 'matic-network',
};

async function getPrice(symbol) {
  const coinId = COIN_MAP[symbol.toUpperCase()];
  if (!coinId) {
    console.error(`Unknown symbol: ${symbol}`);
    process.exit(1);
  }

  return new Promise((resolve, reject) => {
    const url = `https://api.coingecko.com/api/v3/simple/price?ids=${coinId}&vs_curr

    https.get(url, (res) => {
      let data = '';
      res.on('data', chunk => data += chunk);
      res.on('end', () => {
        try {
          const parsed = JSON.parse(data);
          const price = parsed[coinId]?.usd;
          const change24h = parsed[coinId]?.usd_24h_change;

          if (!price) {
            reject(new Error('Price not found'));
            return;
          }

          resolve({
            symbol: symbol.toUpperCase(),
            price,
            change24h: change24h?.toFixed(2),
            timestamp: new Date().toISOString()
          });
        } catch (e) {
          reject(e);
        }
      });
    }).on('error', reject);
  });
}
```

```
// Usage: node check-price.js BTC
const symbol = process.argv[2];
if (!symbol) {
  console.error('Usage: node check-price.js <SYMBOL>');
  process.exit(1);
}

getPrice(symbol)
  .then(data => {
    console.log(JSON.stringify(data, null, 2));
  })
  .catch(err => {
    console.error('Error:', err.message);
    process.exit(1);
  });
```

Test it:

```
node check-price.js BTC
```

Expected output:

```
{
  "symbol": "BTC",
  "price": 67420.50,
  "change24h": "-2.34",
  "timestamp": "2026-03-04T08:00:00.000Z"
}
```

The Alert Configuration

Create `alerts.json` — your list of active alerts:

```
{
  "alerts": [
```

```
    {
      "id": "btc-below-60k",
      "symbol": "BTC",
      "condition": "below",
      "threshold": 60000,
      "message": "BTC has dropped below $60,000. Check your stop loss levels.",
      "active": true
    },
    {
      "id": "eth-above-4k",
      "symbol": "ETH",
      "condition": "above",
      "threshold": 4000,
      "message": "ETH broke above $4,000. Consider taking partial profits.",
      "active": true
    },
    {
      "id": "btc-24h-drop",
      "symbol": "BTC",
      "condition": "24h_change_below",
      "threshold": -8,
      "message": "BTC down more than 8% in 24h. Potential capitulation event.",
      "active": true
    }
  ]
}
```

The Alert Checker Script

Create `run-alerts.js`:

```
#!/usr/bin/env node
// price-alert/run-alerts.js
// Checks all active alerts and outputs triggered ones

const { execSync } = require('child_process');
const fs = require('fs');
const path = require('path');

const alertsFile = path.join(__dirname, 'alerts.json');
const stateFile = path.join(__dirname, '.alert-state.json');
```

```
function loadAlerts() {
  return JSON.parse(fs.readFileSync(alertsFile, 'utf8')).alerts;
}

function loadState() {
  try {
    return JSON.parse(fs.readFileSync(stateFile, 'utf8'));
  } catch {
    return { triggered: {}, lastCheck: null };
  }
}

function saveState(state) {
  fs.writeFileSync(stateFile, JSON.stringify(state, null, 2));
}

function getPrice(symbol) {
  const result = execSync(`node ${path.join(__dirname, 'check-price.js')} ${symbol}`
    encoding: 'utf8',
    timeout: 10000
  });
  return JSON.parse(result);
}

function checkCondition(priceData, alert) {
  switch (alert.condition) {
    case 'above':
      return priceData.price > alert.threshold;
    case 'below':
      return priceData.price < alert.threshold;
    case '24h_change_above':
      return parseFloat(priceData.change24h) > alert.threshold;
    case '24h_change_below':
      return parseFloat(priceData.change24h) < alert.threshold;
    default:
      return false;
  }
}

async function main() {
  const alerts = loadAlerts().filter(a => a.active);
  const state = loadState();
  const triggered = [];

  // Group by symbol to minimize API calls
```

```
  const bySymbol = {};
  alerts.forEach(a => {
    if (!bySymbol[a.symbol]) bySymbol[a.symbol] = [];
    bySymbol[a.symbol].push(a);
  });

  for (const [symbol, symbolAlerts] of Object.entries(bySymbol)) {
    let priceData;
    try {
      priceData = getPrice(symbol);
    } catch (e) {
      console.error(`Failed to fetch ${symbol}: ${e.message}`);
      continue;
    }

    for (const alert of symbolAlerts) {
      const isTriggered = checkCondition(priceData, alert);
      const wasTriggered = state.triggered[alert.id];

      // Only alert on state change (triggered → not triggered resets it)
      if (isTriggered && !wasTriggered) {
        triggered.push({
          alert,
          priceData,
          triggeredAt: new Date().toISOString()
        });
        state.triggered[alert.id] = true;
      } else if (!isTriggered && wasTriggered) {
        // Reset — condition no longer met
        delete state.triggered[alert.id];
      }
    }
  }

  state.lastCheck = new Date().toISOString();
  saveState(state);

  if (triggered.length > 0) {
    console.log(JSON.stringify({ triggered }, null, 2));
    process.exit(0);
  } else {
    console.log(JSON.stringify({ triggered: [] }));
    process.exit(0);
  }
}
```

```
main().catch(err => {
  console.error('Fatal error:', err.message);
  process.exit(1);
});
```

The SKILL.md

This is what your agent reads to understand how to use the skill:

```
# Price Alert Skill

This skill monitors cryptocurrency prices and triggers alerts based on configured co

## Files
- `check-price.js` — Fetches current price for a symbol
- `run-alerts.js` — Checks all active alerts and returns triggered ones
- `alerts.json` — Alert configuration (edit to add/remove alerts)

## Usage

### Check a single price
```bash
node ~/.openclaw/workspace/skills/price-alert/check-price.js BTC
```

Returns JSON with price, 24h change, and timestamp.

RUN ALL ALERT CHECKS

```
node ~/.openclaw/workspace/skills/price-alert/run-alerts.js
```

Returns JSON with list of triggered alerts.

MANAGING ALERTS

To add an alert, edit `alerts.json`. Fields:
```

- `id` — Unique identifier (no spaces)
- `symbol` — Ticker (BTC, ETH, SOL, etc.)
- `condition` — One of: `above`, `below`, `24h_change_above`, `24h_change_below`
- `threshold` — Price level or percentage
- `message` — Human-readable alert message
- `active` — Set to false to disable without deleting

Automation

Run via cron every 5 minutes:

```
*/5 * * * * node ~/.openclaw/workspace/skills/price-alert/run-alerts.js
```

Supported Symbols

BTC, ETH, SOL, BNB, XRP, ADA, AVAX, DOT, LINK, MATIC (Add more to COIN_MAP in check-price.js)

````
#### Setting Up the Cron Job

Now wire it to run automatically and alert you via Telegram:

```bash
# Edit your crontab
crontab -e
````

Add this line:

```
*/5 * * * * cd ~/.openclaw/workspace && node skills/price-alert/run-alerts.js
```
```

> ***Note:*** *For automated Telegram delivery from cron scripts, check the scheduler options in your version of OpenClaw at [docs.openclaw.ai](). The built-in scheduler integrates directly with your agent and is the cleanest way to handle automated alerts.*

Or set up a system cron job (more reliable across versions):

```
# Edit your crontab
crontab -e

# Add this line:
*/5 * * * * cd ~/.openclaw/workspace && node skills/price-alert/run-alerts.js
```

TESTING YOUR ALERT SKILL

Drop a BTC alert threshold above the current price temporarily, then run:

```
node ~/.openclaw/workspace/skills/price-alert/run-alerts.js
```

You should see a triggered alert in the output. Your agent will pick this up and message you on Telegram.

Now send your agent a message in Telegram:

> *"Check BTC price"*

Your agent knows about the price alert skill (from SKILL.md) and will use it to fetch and report the current price.

Chapter 4: Portfolio Tracking — Know Your Numbers at All Times {#portfolio-tracking}

WHY MANUAL PORTFOLIO TRACKING FAILS

Most traders have a rough mental model of their portfolio. "I've got about 2 BTC, some ETH, and a bunch of alts." That's not a position — that's a guess. When markets move fast, guesses get expensive.

Good portfolio tracking tells you:

- Current value in USD/EUR
- Unrealized P&L per position and overall
- 24-hour change in absolute and percentage terms
- How your allocation has drifted from targets

BUILDING THE PORTFOLIO TRACKER

Position File

Create `~/.openclaw/workspace/portfolio/positions.json`:

```
{
  "currency": "USD",
  "positions": [
    {
      "symbol": "BTC",
      "quantity": 0.5,
      "avg_buy_price": 52000,
      "notes": "Core position, long-term hold"
    },
    {
      "symbol": "ETH",
      "quantity": 4.2,
      "avg_buy_price": 2800,
      "notes": "Staking portion + trading portion"
    },
    {
      "symbol": "SOL",
      "quantity": 45,
      "avg_buy_price": 95,
      "notes": "Alt position, target 15% portfolio"
    },
    {
```

```
      "symbol": "LINK",
      "quantity": 200,
      "avg_buy_price": 14.50,
      "notes": "Speculative, small position"
    }
  ],
  "cash_usd": 5000,
  "targets": {
    "BTC": 50,
    "ETH": 30,
    "SOL": 15,
    "LINK": 5
  }
}
```

Portfolio Calculator Script

Create `~/.openclaw/workspace/skills/portfolio/portfolio.js`:

```
#!/usr/bin/env node
// portfolio/portfolio.js — Calculate portfolio value and P&L

const https = require('https');
const fs = require('fs');
const path = require('path');

const positionsFile = path.join(
  process.env.PORTFOLIO_FILE ||
  path.join(__dirname, '../../portfolio/positions.json')
);

function fetchPrices(symbols) {
  const coinMap = {
    BTC: 'bitcoin', ETH: 'ethereum', SOL: 'solana',
    BNB: 'binancecoin', XRP: 'ripple', ADA: 'cardano',
    AVAX: 'avalanche-2', DOT: 'polkadot', LINK: 'chainlink',
    MATIC: 'matic-network', UNI: 'uniswap', AAVE: 'aave',
  };

  const ids = symbols
    .map(s => coinMap[s.toUpperCase()])
    .filter(Boolean)
```

```
    .join(',');

  const url = `https://api.coingecko.com/api/v3/simple/price?ids=${ids}&vs_currencie

  return new Promise((resolve, reject) => {
    https.get(url, (res) => {
      let data = '';
      res.on('data', chunk => data += chunk);
      res.on('end', () => {
        try {
          const parsed = JSON.parse(data);
          // Remap back to symbols
          const prices = {};
          symbols.forEach(sym => {
            const id = coinMap[sym.toUpperCase()];
            if (id && parsed[id]) {
              prices[sym.toUpperCase()] = {
                usd: parsed[id].usd,
                change24h: parsed[id].usd_24h_change,
                change7d: parsed[id].usd_7d_change,
              };
            }
          });
          resolve(prices);
        } catch (e) {
          reject(e);
        }
      });
    }).on('error', reject);
  });
}

function formatCurrency(n) {
  return new Intl.NumberFormat('en-US', {
    style: 'currency',
    currency: 'USD',
    minimumFractionDigits: 2,
    maximumFractionDigits: 2
  }).format(n);
}

function formatPct(n) {
  const sign = n >= 0 ? '+' : '';
  return `${sign}${n.toFixed(2)}%`;
}
```

```
async function main() {
  const data = JSON.parse(fs.readFileSync(positionsFile, 'utf8'));
  const { positions, cash_usd = 0, targets = {} } = data;

  const symbols = positions.map(p => p.symbol);
  const prices = await fetchPrices(symbols);

  let totalValue = cash_usd;
  let totalCost = 0;
  const results = [];

  for (const pos of positions) {
    const sym = pos.symbol.toUpperCase();
    const priceData = prices[sym];

    if (!priceData) {
      console.error(`No price data for ${sym}`);
      continue;
    }

    const currentPrice = priceData.usd;
    const value = pos.quantity * currentPrice;
    const cost = pos.quantity * pos.avg_buy_price;
    const pnl = value - cost;
    const pnlPct = ((value - cost) / cost) * 100;
    const change24h = (priceData.change24h / 100) * value;

    totalValue += value;
    totalCost += cost;

    results.push({
      symbol: sym,
      quantity: pos.quantity,
      avgBuy: pos.avg_buy_price,
      currentPrice,
      value,
      cost,
      pnl,
      pnlPct,
      change24h,
      change24hPct: priceData.change24h,
      change7dPct: priceData.change7d,
      notes: pos.notes
    });
```

```
}

const totalPnl = totalValue - totalCost - cash_usd;
const totalPnlPct = (totalPnl / totalCost) * 100;

// Build report
const lines = [
  `📊 **Portfolio Report** — ${new Date().toUTCString()}`,
  ``,
  `**Total Value:** ${formatCurrency(totalValue)}`,
  `**Total P&L:** ${formatCurrency(totalPnl)} (${formatPct(totalPnlPct)})`,
  `**Cash:** ${formatCurrency(cash_usd)}`,
  ``,
  `---`,
  ``,
  `**Positions:**`,
];

// Sort by value descending
results.sort((a, b) => b.value - a.value);

for (const r of results) {
  const alloc = ((r.value / totalValue) * 100).toFixed(1);
  const target = targets[r.symbol];
  const targetStr = target ? ` (target: ${target}%)` : '';

  lines.push(`\n**${r.symbol}** — ${alloc}%${targetStr}`);
  lines.push(`  Price: ${formatCurrency(r.currentPrice)} | 24h: ${formatPct(r.char
  lines.push(`  Holdings: ${r.quantity} × ${formatCurrency(r.avgBuy)} avg`);
  lines.push(`  Value: ${formatCurrency(r.value)} | P&L: ${formatCurrency(r.pnl)}
}

// Allocation drift
lines.push(`\n---\n\n**Allocation vs Targets:**`);
for (const r of results) {
  const actual = (r.value / totalValue) * 100;
  const target = targets[r.symbol];
  if (target) {
    const drift = actual - target;
    const driftStr = drift > 0 ? `+${drift.toFixed(1)}% (overweight)` : `${drift.t
    lines.push(`  ${r.symbol}: ${actual.toFixed(1)}% vs ${target}% target → ${drif
  }
}

const report = lines.join('\n');
```

```
  console.log(report);

  // Also output JSON for programmatic use
  if (process.argv.includes('--json')) {
    const jsonOutput = {
      timestamp: new Date().toISOString(),
      totalValue,
      totalCost,
      totalPnl,
      totalPnlPct,
      cashUsd: cash_usd,
      positions: results
    };
    console.log('\n---JSON---\n' + JSON.stringify(jsonOutput, null, 2));
  }
}

main().catch(err => {
  console.error('Portfolio error:', err.message);
  process.exit(1);
});
```

Test it:

```
node ~/.openclaw/workspace/skills/portfolio/portfolio.js
```

AUTOMATING DAILY REPORTS

You want this report delivered every morning before you start your day. Set up a scheduled task:

```
# Edit your crontab
crontab -e

# Add this line — runs at 7:00 AM every day:
0 7 * * * cd ~/.openclaw/workspace && node skills/portfolio/portfolio.js
```

Your agent monitors its workspace — when the script outputs the report, the agent picks it up and delivers it to your Telegram automatically.

> ***Note:*** *For the cleanest automated delivery, check the scheduler options in your version of OpenClaw at [docs.openclaw.ai](). The built-in scheduler handles Telegram delivery natively without extra piping.*

TALKING TO YOUR PORTFOLIO

Once the skill is set up, you can query it conversationally via Telegram:

> ***You:*** *How's my portfolio looking?*

> ***Agent:*** *Running portfolio check now...*
> *Total Value: $87,420 (+$12,340 from cost basis, +16.4%)*
> *BTC is your biggest position at 52% of portfolio — currently overweight vs your 50% target by 2%. ETH is at 29.8%, nearly spot on your 30% target. SOL has run well and is now 14.2% — slightly underweight.*
> *24-hour change: -$1,847 (-2.1%), tracking the broader market pullback.*

This is the power of connecting data to conversation. You're not reading a spreadsheet — you're having a discussion with an agent that understands your context.

TRACKING HISTORICAL PERFORMANCE

Add a simple history logger to track daily snapshots:

```
// Add to end of portfolio.js before the main() call
function logSnapshot(data) {
  const historyDir = path.join(__dirname, '../../portfolio/history');
  fs.mkdirSync(historyDir, { recursive: true });

  const date = new Date().toISOString().split('T')[0];
  const file = path.join(historyDir, `${date}.json`);

  fs.writeFileSync(file, JSON.stringify({
    ...data,
```

```
    savedAt: new Date().toISOString()
  }, null, 2));
}
```

After a few weeks, you can ask your agent:

> *"How has my portfolio performed over the last 30 days?"*

It can read the history files and compute performance metrics for you.

Chapter 5: Exchange API Integration — Binance, Coinbase, Kraken {#exchange-integration}

A NOTE ON API KEY SECURITY

Before anything else: **API keys are access credentials to your money.** Treat them like passwords.

Rules:

1. **Create read-only API keys** for monitoring. Only enable trading permissions if you're building execution features.
2. **IP-restrict your keys.** Every major exchange lets you whitelist specific IP addresses. Do this — especially if your home IP is static.
3. **Store keys in environment variables,** not in code files.
4. **Never commit API keys to git** — even private repos.
5. **Rotate keys periodically** and immediately if you suspect exposure.

STORING KEYS SECURELY

OpenClaw has a built-in secrets store:

```
openclaw secrets set BINANCE_API_KEY=your_key_here
openclaw secrets set BINANCE_SECRET=your_secret_here
openclaw secrets set COINBASE_API_KEY=your_key_here
```

```
openclaw secrets set KRAKEN_API_KEY=your_key_here
openclaw secrets set KRAKEN_SECRET=your_secret_here
```

These are encrypted at rest and only accessible to your local agent. Access them in scripts via:

```
const apiKey = process.env.BINANCE_API_KEY; // injected by OpenClaw runtime
```

Or directly in shell scripts:

```
openclaw secrets get BINANCE_API_KEY
```

BINANCE INTEGRATION

Binance has one of the most complete APIs in crypto. Here's how to connect:

Install the Binance SDK

```
cd ~/.openclaw/workspace
npm init -y
npm install binance
```

Binance Account Reader

Create `skills/exchange/binance.js`:

```
#!/usr/bin/env node
// exchange/binance.js — Read account balances from Binance

const { Spot } = require('binance');

const client = new Spot(
  process.env.BINANCE_API_KEY,
```

```
  process.env.BINANCE_SECRET
);

async function getBalances() {
  const account = await client.account();

  // Filter to non-zero balances
  const balances = account.data.balances
    .filter(b => parseFloat(b.free) + parseFloat(b.locked) > 0)
    .map(b => ({
      asset: b.asset,
      free: parseFloat(b.free),
      locked: parseFloat(b.locked),
      total: parseFloat(b.free) + parseFloat(b.locked)
    }))
    .sort((a, b) => b.total - a.total);

  return balances;
}

async function getOpenOrders() {
  const orders = await client.openOrders();
  return orders.data.map(o => ({
    symbol: o.symbol,
    side: o.side,
    type: o.type,
    price: parseFloat(o.price),
    origQty: parseFloat(o.origQty),
    executedQty: parseFloat(o.executedQty),
    status: o.status,
    time: new Date(o.time).toISOString()
  }));
}

async function getRecentTrades(symbol = 'BTCUSDT', limit = 10) {
  const trades = await client.myTrades(symbol, { limit });
  return trades.data.map(t => ({
    symbol: t.symbol,
    side: t.isBuyer ? 'BUY' : 'SELL',
    price: parseFloat(t.price),
    qty: parseFloat(t.qty),
    total: parseFloat(t.quoteQty),
    commission: parseFloat(t.commission),
    commissionAsset: t.commissionAsset,
    time: new Date(t.time).toISOString()
```

```
  }));
}

const command = process.argv[2] || 'balances';

switch (command) {
  case 'balances':
    getBalances().then(data => console.log(JSON.stringify(data, null, 2)));
    break;
  case 'orders':
    getOpenOrders().then(data => console.log(JSON.stringify(data, null, 2)));
    break;
  case 'trades':
    getRecentTrades(process.argv[3]).then(data => console.log(JSON.stringify(data, r
    break;
  default:
    console.error('Unknown command. Use: balances, orders, trades');
    process.exit(1);
}
```

Test it:

```
BINANCE_API_KEY=your_key BINANCE_SECRET=your_secret node skills/exchange/binance.js
```

Now you can ask your agent:

> *"What are my open orders on Binance?" "Show me my recent BTC trades" "How much USDT do I have available?"*

Setting Up API Keys on Binance

1. Log in to Binance → Account → API Management
2. Create API → name it "OpenClaw Monitor"
3. Check **Read Info** only — do NOT check "Enable Trading" unless you need it
4. Under **IP Access Restrictions**, add your IP address
5. Copy Key and Secret to OpenClaw secrets

COINBASE ADVANCED TRADE INTEGRATION

Coinbase uses OAuth2 or API key auth for their Advanced Trade API:

```
#!/usr/bin/env node
// exchange/coinbase.js — Coinbase Advanced Trade API

const crypto = require('crypto');
const https = require('https');

const API_KEY = process.env.COINBASE_API_KEY;
const API_SECRET = process.env.COINBASE_SECRET;

function sign(timestamp, method, path, body = '') {
  const message = timestamp + method + path + body;
  return crypto.createHmac('sha256', API_SECRET).update(message).digest('hex');
}

function request(method, path, body = '') {
  const timestamp = Math.floor(Date.now() / 1000).toString();
  const signature = sign(timestamp, method, path, body);

  return new Promise((resolve, reject) => {
    const options = {
      hostname: 'api.coinbase.com',
      path,
      method,
      headers: {
        'CB-ACCESS-KEY': API_KEY,
        'CB-ACCESS-SIGN': signature,
        'CB-ACCESS-TIMESTAMP': timestamp,
        'Content-Type': 'application/json',
      }
    };

    const req = https.request(options, (res) => {
      let data = '';
      res.on('data', chunk => data += chunk);
      res.on('end', () => {
        try {
          resolve(JSON.parse(data));
        } catch (e) {
          reject(e);
```

```
        }
      });
    });

    req.on('error', reject);
    if (body) req.write(body);
    req.end();
  });
}

async function getAccounts() {
  const data = await request('GET', '/api/v3/brokerage/accounts');
  return data.accounts
    .filter(a => parseFloat(a.available_balance.value) > 0)
    .map(a => ({
      currency: a.currency,
      available: parseFloat(a.available_balance.value),
      hold: parseFloat(a.hold.value)
    }));
}

async function getPortfolio() {
  const data = await request('GET', '/api/v3/brokerage/portfolios');
  return data.portfolios;
}

const command = process.argv[2] || 'accounts';
if (command === 'accounts') {
  getAccounts().then(d => console.log(JSON.stringify(d, null, 2)));
} else if (command === 'portfolio') {
  getPortfolio().then(d => console.log(JSON.stringify(d, null, 2)));
}
```

KRAKEN INTEGRATION

Kraken's API uses a different authentication scheme (nonce-based HMAC):

```
#!/usr/bin/env node
// exchange/kraken.js — Kraken API integration

const crypto = require('crypto');
const https = require('https');
```

```
const qs = require('querystring');

const API_KEY = process.env.KRAKEN_API_KEY;
const API_SECRET = process.env.KRAKEN_SECRET;

function getSignature(path, request, secret) {
  const message = request.nonce + qs.stringify(request);
  const secretBuffer = Buffer.from(secret, 'base64');
  const hash = crypto.createHash('sha256').update(request.nonce + qs.stringify(reque
  const hmac = crypto.createHmac('sha512', secretBuffer);
  hmac.update(path + hash, 'binary');
  return hmac.digest('base64');
}

function privateRequest(method, params = {}) {
  const nonce = Date.now().toString();
  const body = { nonce, ...params };
  const path = `/0/private/${method}`;
  const signature = getSignature(path, body, API_SECRET);

  return new Promise((resolve, reject) => {
    const postData = qs.stringify(body);
    const options = {
      hostname: 'api.kraken.com',
      path,
      method: 'POST',
      headers: {
        'API-Key': API_KEY,
        'API-Sign': signature,
        'Content-Type': 'application/x-www-form-urlencoded',
        'Content-Length': postData.length
      }
    };

    const req = https.request(options, (res) => {
      let data = '';
      res.on('data', chunk => data += chunk);
      res.on('end', () => {
        try {
          const parsed = JSON.parse(data);
          if (parsed.error && parsed.error.length > 0) {
            reject(new Error(parsed.error.join(', ')));
          } else {
            resolve(parsed.result);
          }
```

```
        } catch (e) {
          reject(e);
        }
      });
    });

    req.on('error', reject);
    req.write(postData);
    req.end();
  });
}

async function getBalance() {
  const balance = await privateRequest('Balance');
  return Object.entries(balance)
    .filter(([, v]) => parseFloat(v) > 0)
    .map(([asset, amount]) => ({ asset, amount: parseFloat(amount) }));
}

async function getOpenOrders() {
  const data = await privateRequest('OpenOrders');
  return Object.entries(data.open).map(([id, order]) => ({
    id,
    pair: order.descr.pair,
    type: order.descr.type,
    ordertype: order.descr.ordertype,
    price: order.descr.price,
    volume: order.vol,
    status: order.status
  }));
}

const command = process.argv[2] || 'balance';
if (command === 'balance') {
  getBalance().then(d => console.log(JSON.stringify(d, null, 2)));
} else if (command === 'orders') {
  getOpenOrders().then(d => console.log(JSON.stringify(d, null, 2)));
}
```

UNIFIED EXCHANGE SKILL

Create `skills/exchange/SKILL.md`:

```
# Exchange Integration Skill

Direct API connections to crypto exchanges.

## Supported Exchanges
- **Binance** — `node skills/exchange/binance.js [balances|orders|trades <SYMBOL>]`
- **Coinbase** — `node skills/exchange/coinbase.js [accounts|portfolio]`
- **Kraken** — `node skills/exchange/kraken.js [balance|orders]`

## Required Environment Variables (stored in OpenClaw secrets)
- `BINANCE_API_KEY`, `BINANCE_SECRET`
- `COINBASE_API_KEY`, `COINBASE_SECRET`
- `KRAKEN_API_KEY`, `KRAKEN_SECRET`

## What You Can Ask
- "What's my Binance balance?"
- "Do I have any open orders on Kraken?"
- "Show my recent trades on Coinbase"
- "What's my total exposure across exchanges?"

## Notes
All API keys are read-only. No trading actions are executed.
```

Chapter 6: Building a Signal Agent — Eyes That Never Close {#signal-agent}

WHAT MAKES A GOOD SIGNAL AGENT

A signal agent isn't a prediction machine. It's a monitoring system — something that watches multiple data sources simultaneously and alerts you when conditions worth your attention are present.

Good signals to monitor:

- **Price action** — significant moves, breakouts, breakdowns
- **Volume spikes** — unusual trading volume often precedes price moves
- **Funding rates** — perpetual futures funding indicates market sentiment extremes

- **Fear & Greed Index** — useful for contrarian positioning
- **On-chain metrics** — exchange inflows/outflows, whale activity

BUILDING THE SIGNAL ENGINE

Create `skills/signal-agent/`:

Fear & Greed Monitor

```
// skills/signal-agent/fear-greed.js
const https = require('https');

function getFearGreed() {
  return new Promise((resolve, reject) => {
    https.get('https://api.alternative.me/fng/?limit=2', (res) => {
      let data = '';
      res.on('data', chunk => data += chunk);
      res.on('end', () => {
        try {
          const parsed = JSON.parse(data);
          const current = parsed.data[0];
          const previous = parsed.data[1];

          resolve({
            current: {
              value: parseInt(current.value),
              classification: current.value_classification,
              timestamp: current.timestamp
            },
            previous: {
              value: parseInt(previous.value),
              classification: previous.value_classification,
            },
            delta: parseInt(current.value) - parseInt(previous.value)
          });
        } catch (e) {
          reject(e);
        }
      });
    }).on('error', reject);
  });
}
```

```
getFearGreed().then(d => console.log(JSON.stringify(d, null, 2)));
```

Volume Spike Detector

```
// skills/signal-agent/volume-spike.js
// Detects unusually high trading volume using CoinGecko data

const https = require('https');

const WATCH_LIST = ['bitcoin', 'ethereum', 'solana', 'binancecoin'];
const SPIKE_THRESHOLD = 2.0; // Alert if volume is 2x the 7-day average

function getVolumeData(coinId) {
  return new Promise((resolve, reject) => {
    const url = `https://api.coingecko.com/api/v3/coins/${coinId}/market_chart?vs_cu

    https.get(url, (res) => {
      let data = '';
      res.on('data', chunk => data += chunk);
      res.on('end', () => {
        try {
          const parsed = JSON.parse(data);
          const volumes = parsed.total_volumes.map(v => v[1]);

          // Last entry is current day (possibly partial)
          const historicalVolumes = volumes.slice(0, -1);
          const currentVolume = volumes[volumes.length - 1];
          const avgVolume = historicalVolumes.reduce((a, b) => a + b, 0) / historica
          const ratio = currentVolume / avgVolume;

          resolve({
            coinId,
            currentVolume,
            avgVolume,
            ratio,
            isSpike: ratio > SPIKE_THRESHOLD
          });
        } catch (e) {
          reject(e);
        }
      });
```

```
    }).on('error', reject);
  });
}

async function checkAllVolumes() {
  const results = await Promise.all(WATCH_LIST.map(getVolumeData));
  const spikes = results.filter(r => r.isSpike);

  console.log(JSON.stringify({
    checked: WATCH_LIST,
    spikes,
    hasSpikes: spikes.length > 0,
    timestamp: new Date().toISOString()
  }, null, 2));
}

checkAllVolumes();
```

Funding Rate Monitor

For perpetuals traders, funding rates are critical. When funding is extremely positive (longs paying shorts), it signals overleveraged longs — often a setup for a pullback.

```
// skills/signal-agent/funding-rates.js
const https = require('https');

// Binance funding rate API (public, no auth required)
function getFundingRates() {
  return new Promise((resolve, reject) => {
    const url = 'https://fapi.binance.com/fapi/v1/premiumIndex';

    https.get(url, (res) => {
      let data = '';
      res.on('data', chunk => data += chunk);
      res.on('end', () => {
        try {
          const parsed = JSON.parse(data);

          const rates = parsed
            .filter(item => item.symbol.endsWith('USDT'))
            .map(item => ({
```

```
            symbol: item.symbol.replace('USDT', ''),
            fundingRate: parseFloat(item.lastFundingRate),
            annualized: parseFloat(item.lastFundingRate) * 3 * 365 * 100, // 3 set
            markPrice: parseFloat(item.markPrice),
          }))
          .sort((a, b) => Math.abs(b.fundingRate) - Math.abs(a.fundingRate));

        // Flag extremes (> 0.1% per 8h or < -0.05%)
        const extremes = rates
          .filter(r => r.fundingRate > 0.001 || r.fundingRate < -0.0005)
          .slice(0, 10);

        console.log(JSON.stringify({
          timestamp: new Date().toISOString(),
          topByRate: rates.slice(0, 10),
          extremes,
          hasExtremes: extremes.length > 0
        }, null, 2));
      } catch (e) {
        reject(e);
      }
    });
  }).on('error', reject);
});
}

getFundingRates();
```

Signal Aggregator

The master script that runs all signal checks and formats an alert:

```
// skills/signal-agent/run-signals.js
const { execSync } = require('child_process');
const path = require('path');

function run(script) {
  try {
    const output = execSync(`node ${path.join(__dirname, script)}`, {
      encoding: 'utf8',
      timeout: 15000
    });
```

```
    return JSON.parse(output);
  } catch (e) {
    return { error: e.message };
  }
}

async function main() {
  const results = {
    timestamp: new Date().toISOString(),
    signals: []
  };

  // Fear & Greed
  const fg = run('fear-greed.js');
  if (!fg.error) {
    const val = fg.current.value;
    if (val ≤ 20) {
      results.signals.push({
        type: 'EXTREME_FEAR',
        severity: 'HIGH',
        message: `Fear & Greed Index: ${val} (${fg.current.classification}). Histori
        data: fg
      });
    } else if (val ≥ 80) {
      results.signals.push({
        type: 'EXTREME_GREED',
        severity: 'HIGH',
        message: `Fear & Greed Index: ${val} (${fg.current.classification}). Conside
        data: fg
      });
    } else if (Math.abs(fg.delta) ≥ 15) {
      results.signals.push({
        type: 'SENTIMENT_SHIFT',
        severity: 'MEDIUM',
        message: `Fear & Greed shifted by ${fg.delta} points in 24h. Current: ${val}
        data: fg
      });
    }
  }

  // Volume spikes
  const vol = run('volume-spike.js');
  if (!vol.error && vol.hasSpikes) {
    vol.spikes.forEach(spike ⇒ {
      results.signals.push({
```

```
        type: 'VOLUME_SPIKE',
        severity: 'MEDIUM',
        message: `${spike.coinId} volume spike: ${spike.ratio.toFixed(1)}x average`,
        data: spike
      });
    });
  }

  // Funding rates
  const funding = run('funding-rates.js');
  if (!funding.error && funding.hasExtremes) {
    funding.extremes.slice(0, 3).forEach(rate => {
      const direction = rate.fundingRate > 0 ? 'longs paying shorts (crowded long)'
      results.signals.push({
        type: 'EXTREME_FUNDING',
        severity: rate.fundingRate > 0.002 || rate.fundingRate < -0.001 ? 'HIGH' : '
        message: `${rate.symbol} funding: ${(rate.fundingRate * 100).toFixed(4)}%/8h
        data: rate
      });
    });
  }

  results.hasSignals = results.signals.length > 0;
  results.highPriority = results.signals.filter(s => s.severity === 'HIGH').length;

  console.log(JSON.stringify(results, null, 2));
}

main();
```

SKILL.md for Signal Agent

```
# Signal Agent Skill

Monitors multiple market data sources and generates actionable alerts.

## Scripts
- `run-signals.js` — Master runner, executes all checks
- `fear-greed.js` — Crypto Fear & Greed Index
- `volume-spike.js` — Detects unusual trading volume
- `funding-rates.js` — Perpetual futures funding rates (Binance)
```

```
## Scheduling
Run every 30 minutes for active monitoring:
```bash
openclaw cron add --name "market-signals" --schedule "*/30 * * * *" --run "node skil
```

Signal Types

- `EXTREME_FEAR` / `EXTREME_GREED` — Sentiment extremes
- `SENTIMENT_SHIFT` — Rapid sentiment change
- `VOLUME_SPIKE` — Volume significantly above average
- `EXTREME_FUNDING` — Funding rates suggesting overleveraged positions

What You Can Ask

- "Any signals right now?"
- "What's the current Fear & Greed?"
- "Are funding rates elevated on BTC?"
- "Any volume spikes in the last hour?"

```
### Scheduling Signal Monitoring

```bash
# Every 30 minutes during market hours (crypto is 24/7 but this saves API calls)
openclaw cron add \
  --name "signal-check" \
  --schedule "*/30 * * * *" \
  --run "node ~/.openclaw/workspace/skills/signal-agent/run-signals.js" \
  --notify-if "hasSignals === true"
```

The `--notify-if` flag means you only get pinged when there's actually something to act on. No notifications for clean checks.

Chapter 7: Overnight Agents — Working While You Sleep {#overnight-agents}

THE 24/7 PROBLEM

Crypto markets don't close. Significant moves happen at 3am. By the time you wake up and check your phone, you've already missed the entry or — worse — are now deep in a loss you didn't know about.

The overnight agent solves this. It runs continuously, makes decisions about what's worth waking you up for, and only alerts you when something genuinely requires your attention.

DESIGNING FOR MINIMAL SLEEP DISRUPTION

The worst outcome is an agent that alerts you for everything. You stop trusting it, turn off notifications, and end up worse than before.

Design principles for overnight monitoring:

1. **High threshold for alerts.** If it happens every night, it's not useful.
2. **One message per event.** Don't repeat alerts for the same condition.
3. **Clear severity levels.** "FYI" vs "Wake up now" are different things.
4. **Morning digest for low-priority items.** Batch the minor stuff.

SETTING UP OVERNIGHT MODE

Create `~/.openclaw/workspace/skills/overnight/overnight.js`:

```
#!/usr/bin/env node
// overnight/overnight.js — Overnight monitoring with smart alerting

const fs = require('fs');
const path = require('path');
const { execSync } = require('child_process');

const CONFIG = {
  // Only send WAKE_UP alerts between these hours (UTC)
  quietHours: { start: 23, end: 7 }, // 11pm - 7am UTC

  // Thresholds for alerts
```

```
  thresholds: {
    btcMovePercent: 8,        // Alert if BTC moves >8% while sleeping
    altMovePercent: 15,       // Alert if major alt moves >15%
    fearGreedExtreme: 15,     // Alert if F&G drops below 15 or above 85
    fundingExtreme: 0.003,    // Alert if any funding rate > 0.3%/8h
  }
};

const stateFile = path.join(__dirname, '.overnight-state.json');

function loadState() {
  try {
    return JSON.parse(fs.readFileSync(stateFile, 'utf8'));
  } catch {
    return {
      alerts: [],
      lastMorningDigest: null,
      sessionStart: new Date().toISOString()
    };
  }
}

function saveState(state) {
  fs.writeFileSync(stateFile, JSON.stringify(state, null, 2));
}

function isQuietHour() {
  const hour = new Date().getUTCHours();
  const { start, end } = CONFIG.quietHours;
  if (start > end) { // overnight range crosses midnight
    return hour ⩾ start || hour < end;
  }
  return hour ⩾ start && hour < end;
}

function shouldSendAlert(severity) {
  if (severity ≡ 'CRITICAL') return true; // Always wake up for critical
  if (severity ≡ 'HIGH' && !isQuietHour()) return true;
  if (severity ≡ 'MEDIUM' && !isQuietHour()) return true;
  return false; // Queue for morning digest
}

async function runChecks() {
  const state = loadState();
  const immediateAlerts = [];
```

```
const digestAlerts = [];

// --- Price movement check ---
try {
  const btcData = JSON.parse(execSync(
    `node ${path.join(__dirname, '../price-alert/check-price.js')} BTC`,
    { encoding: 'utf8', timeout: 10000 }
  ));

  const btcChange = Math.abs(parseFloat(btcData.change24h));
  if (btcChange > CONFIG.thresholds.btcMovePercent) {
    const direction = parseFloat(btcData.change24h) > 0 ? '📈 UP' : '📉 DOWN';
    const severity = btcChange > 15 ? 'CRITICAL' : 'HIGH';
    const alert = {
      type: 'BTC_MOVE',
      severity,
      message: `BTC ${direction} ${btcData.change24h}% in 24h. Current price: $${b
      time: new Date().toISOString()
    };

    if (shouldSendAlert(severity)) {
      immediateAlerts.push(alert);
    } else {
      digestAlerts.push(alert);
    }
  }
} catch (e) {
  // Price check failed — not worth alerting for
}

// --- Fear & Greed ---
try {
  const fg = JSON.parse(execSync(
    `node ${path.join(__dirname, '../signal-agent/fear-greed.js')}`,
    { encoding: 'utf8', timeout: 10000 }
  ));

  const val = fg.current.value;
  if (val ⩽ CONFIG.thresholds.fearGreedExtreme || val ⩾ (100 - CONFIG.thresholds
    const severity = val ⩽ 10 || val ⩾ 90 ? 'HIGH' : 'MEDIUM';
    const alert = {
      type: 'SENTIMENT_EXTREME',
      severity,
      message: `Fear & Greed: ${val} (${fg.current.classification})`,
      time: new Date().toISOString()
```

```
      };

      if (shouldSendAlert(severity)) {
        immediateAlerts.push(alert);
      } else {
        digestAlerts.push(alert);
      }
    }
  } catch (e) {}

  // Update state
  state.alerts = [...state.alerts, ...immediateAlerts, ...digestAlerts].slice(-100);
  saveState(state);

  const output = {
    timestamp: new Date().toISOString(),
    isQuietHour: isQuietHour(),
    immediateAlerts,
    digestAlerts,
    hasImmediateAlerts: immediateAlerts.length > 0
  };

  console.log(JSON.stringify(output, null, 2));
}

// Morning digest — call separately at 7am
async function morningDigest() {
  const state = loadState();
  const since = state.lastMorningDigest || state.sessionStart;

  const recentAlerts = state.alerts.filter(a => a.time > since);

  state.lastMorningDigest = new Date().toISOString();
  saveState(state);

  const digest = {
    type: 'MORNING_DIGEST',
    period: `${since} to ${state.lastMorningDigest}`,
    alertCount: recentAlerts.length,
    alerts: recentAlerts
  };

  console.log(JSON.stringify(digest, null, 2));
}
```

```
const command = process.argv[2];
if (command === 'digest') {
  morningDigest();
} else {
  runChecks();
}
```

SCHEDULING OVERNIGHT OPERATIONS

```
# Overnight monitoring — every 15 minutes
openclaw cron add \
  --name "overnight-watch" \
  --schedule "*/15 * * * *" \
  --run "node skills/overnight/overnight.js" \
  --notify-if "hasImmediateAlerts === true"

# Morning digest at 7:00 AM local time
openclaw cron add \
  --name "morning-digest" \
  --schedule "0 7 * * *" \
  --run "node skills/overnight/overnight.js digest" \
  --notify telegram
```

ALERT MESSAGE FORMATTING

When your agent sends you an overnight alert, you want it to be clear and actionable. Define how alerts should be formatted in your agent's context:

Add to your `~/.openclaw/workspace/USER.md`:

```
## Alert Preferences
- CRITICAL alerts: Wake me up. No preamble, straight to the point.
- HIGH alerts: Send immediately with full context.
- MEDIUM alerts: Batch into morning digest.
- LOW alerts: Skip unless I ask.

## Alert Format for Overnight
```

```
Start with the severity level. Give me price, context, and one suggested action to d
Don't pad. Don't hedge. I'll make the decision.
```

HANDLING WAKE-UP ALERTS

When you receive a 3am alert and need context fast, message your agent:

> *"What happened? Full context."*

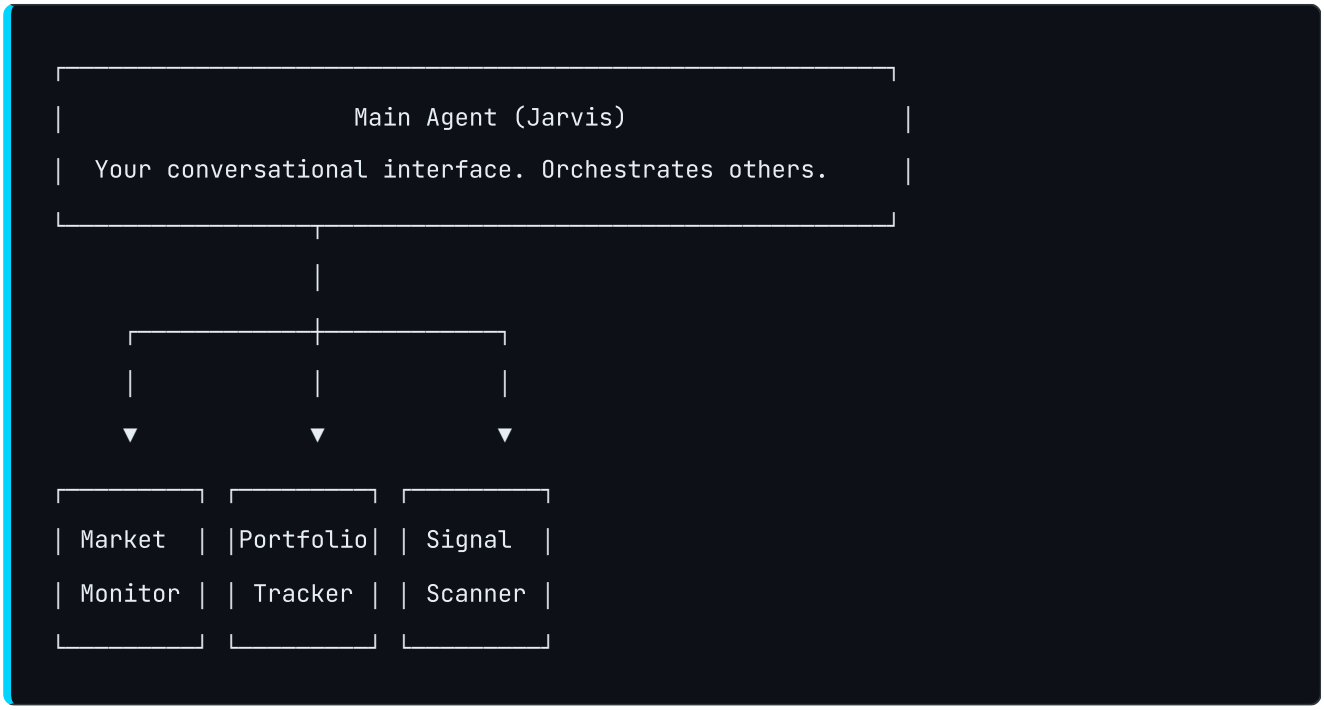
The agent has the overnight state file and can reconstruct exactly what happened, what triggered the alert, and where things stand now. You get a briefing, not raw data.

Chapter 8: Advanced Strategies — Multi-Agent Setups & Risk Management {#advanced-strategies}

MULTI-AGENT ARCHITECTURE

For serious trading, a single agent doing everything is a bottleneck. You want specialized agents that excel at specific tasks and coordinate to give you better information.

Proposed architecture:



Each sub-agent has a specific workspace and role. They run independently and report to the main agent when something needs attention.

SETTING UP SUB-AGENT COORDINATION

The main agent in `AGENTS.md` can spawn sub-agents for specific tasks:

```
# Sub-Agent Config (add to AGENTS.md)

## Available Sub-Agents

### market-monitor
Role: Continuous price and volume monitoring
Workspace: skills/overnight/
Reports to: Main agent via Telegram when thresholds exceeded

### portfolio-tracker
Role: Daily P&L calculation and reporting
Workspace: skills/portfolio/
Reports to: Main agent at 7:00 AM daily

### signal-scanner
Role: Market sentiment and technical signals
Workspace: skills/signal-agent/
Reports to: Main agent every 30 minutes if signals detected
```

THE RISK MANAGEMENT LAYER

This is the part most traders skip, and it's the part that costs them the most.

Add a risk management prompt to your agent context that it applies before any position-related discussion:

Add to `~/.openclaw/workspace/USER.md`:

```
## Risk Rules (Agent Must Apply These)

Before discussing any new position or trade idea:
1. Calculate what % of total portfolio this represents
2. Flag if single position > 10% of portfolio
```

```
3. Flag if total crypto exposure > 80% of net worth
4. Calculate max loss at -50% price decline
5. Ask if there's a stop-loss level defined

My hard limits:
- Maximum single position: 10% of crypto portfolio
- Minimum cash buffer: 15% of portfolio in USDT/stablecoins
- Maximum leverage: 3x (preferably none)
- If down >20% on any position, require explicit review before adding

Do not make me feel good about trades that violate these rules.
Tell me plainly if I'm overextended.
```

RISK CALCULATOR SKILL

```
// skills/risk/calculator.js — Position sizing and risk calculations

const args = process.argv.slice(2);
const command = args[0];

function calculatePositionSize({
  portfolioSize,
  riskPercent,
  entryPrice,
  stopLoss
}) {
  const riskAmount = portfolioSize * (riskPercent / 100);
  const priceRisk = Math.abs(entryPrice - stopLoss);
  const priceRiskPercent = (priceRisk / entryPrice) * 100;
  const units = riskAmount / priceRisk;
  const positionSize = units * entryPrice;
  const positionPercent = (positionSize / portfolioSize) * 100;

  return {
    riskAmount,
    units,
    positionSize,
    positionPercent,
    priceRiskPercent,
    maxLoss: riskAmount,
    recommendation: positionPercent > 15
```

```
      ? `WARNING: Position is ${positionPercent.toFixed(1)}% of portfolio — consider
      : `Position size looks appropriate at ${positionPercent.toFixed(1)}% of portfo
  };
}

function calculateRR({
  entry,
  stopLoss,
  target
}) {
  const risk = Math.abs(entry - stopLoss);
  const reward = Math.abs(target - entry);
  const rr = reward / risk;

  return {
    risk,
    reward,
    rr,
    riskPercent: (risk / entry) * 100,
    rewardPercent: (reward / entry) * 100,
    assessment: rr < 2 ? 'Poor R:R — consider passing or adjusting target'
      : rr < 3 ? 'Acceptable R:R'
      : 'Good R:R'
  };
}

if (command === 'size') {
  // node calculator.js size --portfolio 50000 --risk 1 --entry 65000 --stop 62000
  const params = {};
  for (let i = 1; i < args.length; i += 2) {
    params[args[i].replace('--', '')] = parseFloat(args[i + 1]);
  }

  const result = calculatePositionSize({
    portfolioSize: params.portfolio,
    riskPercent: params.risk || 1,
    entryPrice: params.entry,
    stopLoss: params.stop
  });

  console.log(JSON.stringify(result, null, 2));
} else if (command === 'rr') {
  // node calculator.js rr --entry 65000 --stop 62000 --target 72000
  const params = {};
  for (let i = 1; i < args.length; i += 2) {
```

```
    params[args[i].replace('--', '')] = parseFloat(args[i + 1]);
  }

  const result = calculateRR({
    entry: params.entry,
    stopLoss: params.stop,
    target: params.target
  });

  console.log(JSON.stringify(result, null, 2));
}
```

Now you can ask your agent:

> *"I want to buy ETH at $3,200 with a stop at $2,900, targeting $3,800. My portfolio is $50,000 and I risk 1% per trade. What size should I take?"*

The agent runs the calculator and gives you the exact number — plus a risk/reward assessment.

COMBINING SIGNALS FOR BETTER DECISIONS

A single signal in isolation is noise. Multiple signals pointing the same direction is a trade setup worth considering.

Create a signal confluence checker:

```
// skills/signal-agent/confluence.js
// Scores market conditions based on multiple signals

const { execSync } = require('child_process');
const path = require('path');

function runScript(script) {
  try {
    return JSON.parse(execSync(`node ${path.join(__dirname, script)}`, {
      encoding: 'utf8', timeout: 15000
    }));
  } catch { return null; }
}
```

```
async function scoreConfluence() {
  const fg = runScript('fear-greed.js');

  let bullishPoints = 0;
  let bearishPoints = 0;
  const factors = [];

  if (fg) {
    const val = fg.current.value;
    if (val < 25) {
      bullishPoints += 3; // Fear = contrarian buy signal
      factors.push({ factor: 'Fear & Greed', signal: 'BULLISH', weight: 3, detail: `
    } else if (val > 75) {
      bearishPoints += 2;
      factors.push({ factor: 'Fear & Greed', signal: 'BEARISH', weight: 2, detail: `
    } else {
      factors.push({ factor: 'Fear & Greed', signal: 'NEUTRAL', weight: 0, detail: `
    }
  }

  // Add more signal sources here as you build them out
  // Each should contribute to bullishPoints or bearishPoints with a weight

  const totalWeight = bullishPoints + bearishPoints;
  const score = totalWeight === 0 ? 0 : ((bullishPoints - bearishPoints) / totalWeig

  return {
    timestamp: new Date().toISOString(),
    confluenceScore: Math.round(score), // -100 to +100
    bullishPoints,
    bearishPoints,
    bias: score > 20 ? 'BULLISH' : score < -20 ? 'BEARISH' : 'NEUTRAL',
    factors,
    summary: score > 50 ? 'Strong bullish confluence — multiple signals aligning'
      : score > 20 ? 'Mild bullish lean — some positive signals'
      : score < -50 ? 'Strong bearish confluence — multiple warning signals'
      : score < -20 ? 'Mild bearish lean — some caution signals'
      : 'Mixed signals — no clear directional bias'
  };
}

scoreConfluence().then(d => console.log(JSON.stringify(d, null, 2)));
```

BUILDING YOUR TRADING DECISION FRAMEWORK

Add this to your agent context to enforce systematic thinking:

```
# Trading Decision Framework (add to USER.md or create TRADING.md)

## Before Any Trade

Step 1 — WHY are we entering?
- What's the thesis? Write it in one sentence.
- Is this trend-following, mean-reversion, or breakout?
- What signals are confirming this?

Step 2 — WHAT is the trade structure?
- Entry price (exact or range)
- Stop loss (mandatory)
- Target (mandatory)
- Position size (calculated via risk calculator)
- R:R ratio (minimum 2:1 required)

Step 3 — WHEN do we exit?
- Profit target reached
- Stop loss hit
- Time-based exit (if thesis doesn't play out in X days)
- Conditions that would invalidate the thesis

Step 4 — WHAT could go wrong?
- Name the two most likely ways this trade fails
- Is there a way to reduce that risk?

When agent helps me think through a trade, follow this structure.
Do not skip steps. Do not let me skip steps.
```

Chapter 9: Prompts & Templates — Your Trading Prompt Library {#prompts-templates}

HOW TO USE THIS SECTION

These prompts are designed to be sent directly to your agent via Telegram. Copy, customize with your details, and send. Each one is engineered for a specific use case.

TRADE ANALYSIS PROMPTS

Morning Briefing

```
Morning briefing. Pull my portfolio status, check for any overnight signals,
and give me the current Fear & Greed. What should I be paying attention to today?
```

Trade Idea Analysis

```
Analyze this trade idea:

Asset: [SYMBOL]
Thesis: [Why you think it moves]
Entry: [Price]
Stop: [Price]
Target: [Price]
Portfolio: $[Amount]
Risk per trade: 1%

Give me: position size, R:R ratio, and 3 reasons this trade could fail.
Be direct. No cheerleading.
```

Position Review

```
Review my [SYMBOL] position:
```

```
Entry: $[Price]
Current: $[Price]
Size: [Units]
Stop loss: $[Price or 'not set']
Original thesis: [What you expected to happen]

Has anything changed that would invalidate the thesis?
Should I stay, adjust, or exit?
```

Stop Loss Check

```
I need to set a stop loss for [SYMBOL] at $[Entry Price].

Look at the daily and 4h structure. Where's the first significant support below entry?
Where would I definitively be wrong?

Don't give me a percentage-based stop — give me a structure-based one.
```

Portfolio Rebalance

```
Run my portfolio report. Identify:
1. Which positions have drifted >5% from targets
2. The 2 positions I'm most overweight in
3. The 2 positions I'm most underweight in

What would a sensible rebalance look like, and at what prices?
```

Drawdown Analysis

```
My portfolio is down [X]% from its high.

Break down:
1. Which positions are responsible for most of the drawdown?
2. Are any of these fundamental thesis failures or just price noise?
3. At current allocation, what's the portfolio value if [BTC/ETH] drops another 20%?
4. Do I have enough cash buffer to buy the dip if it continues?
```

MARKET CONTEXT PROMPTS

Macro Check

```
Quick macro check. What's the current narrative driving crypto?
Interest rates, ETF flows, any major news this week.
Give me 3 bullet points — facts only, no speculation.
```

Correlation Check

```
Is crypto currently correlated with equities (specifically S&P 500)?
High correlation means macro events matter more.
Low correlation means crypto is trading on its own fundamentals.
Which is it right now and what does that mean for how I should trade?
```

DeFi Yield Check

```
Check current stablecoin yields on major protocols.
Compare: Aave USDC, Curve 3pool, GMX GLP.
```

```
I have [$AMOUNT] in cash sitting idle.
What's the risk-adjusted best place to park it short-term?
```

RISK MANAGEMENT PROMPTS

Leverage Audit

```
Audit my leverage exposure.

Current positions:
[List positions and any leveraged ones]

What's my effective leverage ratio?
What's my liquidation risk if BTC drops 25%?
Should I reduce? Where?
```

Worst Case Scenario

```
Run a worst case scenario for my portfolio.

Scenario: Crypto market crashes -60% from current levels over 3 months
(similar to 2022 bear market onset).

What's my portfolio worth at the bottom?
Which positions would I want to hold through it?
Which would I want to cut?
What's the actual dollar impact on my net worth?

No sugarcoating.
```

Position Sizing Check

```
I want to add to my [SYMBOL] position.

Current position: [Units] at avg $[Price]
Current portfolio value: $[Amount]
I want to add: $[Amount]

After adding, what % of portfolio is this?
Does it violate my 10% single-position limit?
What's my new average buy price?
What's the risk if this goes to $[Price]?
```

ALERT SETUP PROMPTS

Set Up New Alert

```
Set up a price alert for [SYMBOL]:
- Alert me if it drops below $[Price]
- Alert me if it breaks above $[Price] with conviction (not a wick)
- Alert me if 24h change exceeds ±10%

Write the alert config I need to add to alerts.json.
```

Alert Review

```
Review my active alerts.
Which ones are too close to current price (likely to trigger on noise)?
```

```
Which ones are stale (thesis no longer valid)?
Suggest which to keep, remove, or adjust.
```

WEEKLY REVIEW TEMPLATE

Weekly Review

```
Weekly trading review. Help me go through this:

Week: [Date range]

1. Trades taken this week: [List]
2. Results: [P&L]
3. Trades I passed on that worked: [If any]
4. Biggest mistake of the week: [What happened]

Analyze:
- Did I follow my rules?
- Where did I let emotion override process?
- What's the one thing I should do differently next week?

Be honest. I need an objective read, not validation.
```

Chapter 10: Troubleshooting & Tips {#troubleshooting}

COMMON ISSUES AND FIXES

Agent Not Responding on Telegram

Symptom: You send a message to your bot and get no reply.

Check 1 — Gateway running?

```
openclaw gateway status
```

If not running: `openclaw gateway start`

Check 2 — Bot token correct?

Open `~/.openclaw/openclaw.json` and verify the `channels.telegram.botToken` matches what BotFather gave you.

Check 3 — Your account approved?

Make sure you've completed the pairing process. Message the bot, get the pairing code, and run:

```
openclaw pairing approve telegram <YOUR_CODE>
```

Check 4 — Gateway logs

```
openclaw logs --follow
```

Look for error messages about Telegram or API connections. Logs also live at `/tmp/openclaw/`.

Price API Rate Limiting

Symptom: `429 Too Many Requests` from CoinGecko.

CoinGecko's free tier allows ~10-30 requests per minute. If you're running multiple cron jobs, you'll hit this.

Fix 1 — Add delays between requests:

```
function sleep(ms) { return new Promise(r => setTimeout(r, ms)); }
// Between API calls:
await sleep(1000); // 1 second delay
```

Fix 2 — Cache results:

```
// Simple file cache
function getCached(key, maxAgeSeconds = 60) {
  const cacheFile = `/tmp/openclaw-cache-${key}.json`;
  try {
    const cached = JSON.parse(fs.readFileSync(cacheFile, 'utf8'));
    if (Date.now() - cached.timestamp < maxAgeSeconds * 1000) {
      return cached.data;
    }
  } catch {}
  return null;
}

function setCache(key, data) {
  const cacheFile = `/tmp/openclaw-cache-${key}.json`;
  fs.writeFileSync(cacheFile, JSON.stringify({ data, timestamp: Date.now() }));
}
```

Fix 3 — Use CoinGecko Pro API: $129/month but removes rate limits. Worth it if you're doing serious monitoring.

Skills Not Found by Agent

Symptom: You ask the agent about a skill and it doesn't know how to use it.

Skills are discovered via `SKILL.md` files. The agent reads these when loading context.

Fix: Make sure each skill directory has a properly formatted `SKILL.md`. The agent uses this file to understand the skill's capabilities.

Also ensure the workspace path is correct:

```
ls ~/.openclaw/workspace/skills/
```

Exchange API Errors

Binance: Invalid timestamp

```
{"code":-1021,"msg":"Timestamp for this request is outside of the recvWindow."}
```

Your system clock is out of sync. Fix:

```
# macOS
sudo sntp -sS time.apple.com

# Linux
sudo timedatectl set-ntp true
```

Binance: Invalid API-key Double-check the key is for the correct environment (mainnet vs testnet). Verify the key has the required permissions. Check IP whitelist on Binance — your current IP must be whitelisted.

Kraken: Invalid nonce Kraken requires nonces to be monotonically increasing. If multiple requests fire at the same millisecond, this breaks. Add a small delay or use a counter-based nonce:

```
let nonceCounter = Date.now();
function getNonce() {
  return (++nonceCounter).toString();
}
```

Cron Jobs Not Running

Check current cron jobs:

```
crontab -l
```

Check cron job logs:

```
# macOS — cron output goes to mail, or redirect to a log file:
0 7 * * * cd ~/.openclaw/workspace && node skills/portfolio/portfolio.js >> /tmp/ope
```

Common issues:

- Script path is relative, not absolute — use full paths in cron
- Environment variables not available in cron context — export them explicitly
- Node.js not in PATH for cron — use full path: `/usr/local/bin/node` or `$(which node)`

Test your cron command manually first:

```
/usr/local/bin/node /Users/you/.openclaw/workspace/skills/portfolio/portfolio.js
```

PERFORMANCE TIPS

Batch API Calls

Instead of calling APIs one at a time, fetch multiple symbols in one request:

```
// Bad — 5 separate API calls
for (const symbol of ['BTC', 'ETH', 'SOL', 'BNB', 'ADA']) {
  const price = await getPrice(symbol);
}
```

```
// Good — 1 API call
const prices = await getPrices(['BTC', 'ETH', 'SOL', 'BNB', 'ADA']);
```

Avoid Polling — Use Events Where Possible

Polling every minute for price changes is fine for alerts, but consider WebSocket connections for real-time monitoring:

```
// Binance WebSocket for real-time price
const ws = require('ws');
const socket = new ws('wss://stream.binance.com:9443/ws/btcusdt@trade');
socket.on('message', (data) => {
  const trade = JSON.parse(data);
  const price = parseFloat(trade.p);
  // Process in real-time without polling
});
```

Keep Agent Context Lean

The more you put in `USER.md` and `AGENTS.md`, the more token overhead per request. Keep context files focused — put detailed trading rules in a separate `TRADING.md` and reference it only when needed.

SECURITY CHECKLIST

Run through this regularly:

```
☐ API keys stored in OpenClaw secrets (not hardcoded in files)
☐ Exchange API keys are IP-restricted
☐ Exchange API keys are read-only (unless you need trading)
☐ Workspace directory not backed up to cloud with API keys in plain text
☐ Telegram bot only allows your user ID
☐ Gateway not exposed to internet (runs on localhost only)
☐ Node.js and OpenClaw are up to date
☐ API keys rotated in last 90 days
```

BUILDING FROM HERE

This guide gives you the foundation. Where you go next depends on your trading style.

If you're a position trader:

- Focus on the portfolio tracker and morning briefings
- Set wide overnight alerts (only major moves)
- Use the weekly review template consistently

If you're an active trader:

- Build the signal agent out with more data sources
- Add TradingView webhook support to receive alerts from custom indicators
- Consider connecting to exchange WebSocket streams for real-time data

If you're running a DeFi strategy:

- Add on-chain monitoring via Etherscan/Alchemy APIs
- Track wallet balances and positions programmatically
- Monitor protocol TVL and yield rates automatically

If you want execution:

- Add trading permissions to your exchange API keys (carefully)
- Build order execution scripts with strict guards and confirmation prompts
- Implement a paper trading mode first to test logic

RESOURCES

APIs Used in This Guide

- CoinGecko: coingecko.com/api — Free price data
- Alternative.me: alternative.me/crypto/fear-and-greed-index — Fear & Greed
- Binance API: binance-docs.github.io — Exchange data
- Coinbase API: docs.cdp.coinbase.com — Exchange data
- Kraken API: docs.kraken.com — Exchange data

OpenClaw

- Documentation: docs.openclaw.ai
- Skills repository: clawhub.com

FINAL NOTE

The system you've built in this guide is a competitive advantage — not because it predicts markets, but because it removes the two biggest sources of trading losses: **missing information** and **emotional decisions**.

You'll have better data than 90% of retail traders. Your agent will enforce your risk rules even when you don't want it to. And you'll sleep better knowing something is watching while you're not.

Build it. Use it. Iterate.

OpenClaw Trades Guide v1.0 | openclaw.dev