Autonoma Unified Strategic + Technical Overview

---

1. Project Identity

Project Name: Autonoma
Type: Infrastructure for creating, operating, and monetizing AI Agents as functional NFTs
Core Value Proposition: Autonoma transforms AI agents into digital workers—each agent can be created, owned, monetized, and transferred as an on-chain asset.

---

2. Vision Statement

"For the past decade, we owned NFTs, tokens, usernames. But never agents. Now we can own intelligence itself."

Autonoma is building the next generation of on-chain AI — programmable, monetizable, and autonomous. It's not just tools — it's infrastructure for intelligent labor.

---

3. Core Product Components

- **Agent Creation Interface**: No-code tool to define AI agent behavior, capabilities, and metadata.
- **NFT Packaging**: Each agent is minted as an ERC-721 NFT with attached metadata and executable pointer.
- **CDP Wallet Binding**: Each agent is linked to a smart wallet it can use to receive funds and interact on-chain.
- **Execution Layer**: Agents run in serverless containers or through integrated endpoints (e.g., Runpod).
- **Payment Gateway**: Agents are called via endpoints that enforce per-call fees using the x402 protocol.
- **Dashboard**: Public and owner-specific views of agent activity, usage logs, and revenue analytics.

---

4. Token Utility and Monetization

Token: $AUTO (SPL Token on Solana)
Total Supply: 1,000,000,000 AUTO

Utility Roles:

- Unlock agent creation, upgrades, and execution endpoints
- Priority queueing for calls and infra
- Governance over agent registry and fees
- Token-gating access to premium AI models or features

Revenue Streams (Platform-Level):

- Setup Fee ($2–5 per agent creation)
- Mint Fee (5–10% on NFT mint)
- Usage Fee (5–20% per call via x402)
- Infra Subscription (optional paid plans for high-usage agents)
- Secondary Sales Fee (ERC-2981 royalties)

---

5. Onboarding and Gating System

Platform: Galxe + Web3 Wallet Auth

Required Tasks:

- Connect wallet
- Follow Autonoma on Twitter
- Join Autonoma Discord
- (Optional but recommended) Mint Soulbound Access Badge

Gating Logic:

- API checks if wallet completed Galxe campaign or holds NFT
- Access granted to gated chat or agent dashboards

Integration:

- Wallet signature-based login (nonce + JWT)
- Profile enrichment via Discord/Twitter/email
- Galxe verification via GraphQL or webhook

---

6. Architecture Overview

Backend: Python (FastAPI) + ddtrace (Datadog APM)
Agent Orchestration: n8n Workflows, AgentKit SDK
Runtime: Docker containers (Runpod, Railway)
Memory & Logs: Supabase PostgreSQL + public usage logs
Document Handling: Google Drive API (for RAG-style context memory)

---

7. Launch Strategy

- Token deployed on Solana with CLMM pool (Raydium)
- Market Maker bot maintains liquidity ±12.5% around base price
- Initial price: $0.000025 per AUTO
- 550M tokens in circulation at launch (airdrop, LP, MM, rewards)
- Gated chat campaign to onboard first 10k users via Galxe

---

8. Roadmap (Q3–Q4 2025)

- AgentKit V2 with skill modularity and dynamic metadata
- Agent Marketplace for discovering and trading agents
- Token-bound wallet upgrade (ERC-6551 adaptation)
- Frontend: unified portal for agent creation, tracking, monetization
- Community governance layer for registry + rewards

---

9. Target Audiences

- Devs and Builders: Create revenue-generating autonomous agents
- DAOs: Use agents for treasury ops, governance automation
- Creators: Monetize personalities and content agents
- Traders: Deploy AI agents for signal trading and execution

---

10. Final Statement

Autonoma is building the infrastructure layer for on-chain intelligence — giving anyone the tools to create, own, and monetize autonomous agents. Where Web3 meets AI, Autonoma is the execution layer.

Autonoma Onboarding and Gating System

---

1. Purpose

Define and implement the unified onboarding funnel for Autonoma users, using Galxe as the social/on-chain verification layer and integrating wallet-based authentication with backend gating logic.

---

2. User Journey Overview

1. Land on onboarding page with CTA to "Unlock the AI Chat"
2. Complete tasks via embedded Galxe campaign:
    - Connect wallet
    - Follow @Autonoma_AI on Twitter
    - Join Discord server
    - (Optional) Mint a soulbound "Autonoma Access Badge"
3. Return to Autonoma site and authenticate via wallet (signature)

4. Gain access to chat or agent interface if verified

---

3. Galxe Campaign Configuration

- **Platform:** https://dashboard.galxe.com/
- **Space:** Autonoma AI
- **Campaign Name:** "🔥 Unlock the Autonoma AI Chat — Join the Resistance 🔐"
- **Tasks:**
    - Wallet connection
    - Twitter follow: @Autonoma_AI
    - Discord join: Autonoma Server
    - Mint access badge (NFT SBT) — optional
- **Campaign Duration:** Continuous with refreshable quests

---

4. Soulbound Access Badge (NFT)

- **Name:** Autonoma Access Badge
- **Standard:** ERC-721, non-transferable (SBT logic enforced)
- **Purpose:** Acts as proof of access for backend gating
- **Metadata:** Description, issuance date, campaign ID
- **Minting Mechanism:** Galxe + Autonoma-linked contract

---

5. Authentication Layer (Web3 Wallet)

- **Flow:**
    - Wallet connects to frontend (Coinbase Wallet, MetaMask, etc.)

    - Call `GET /api/auth/nonce?wallet=0x...`
    - User signs nonce message: "Sign this nonce: "
    - Signature sent to `POST /api/auth/verify`
    - JWT token issued if verified
- **JWT Payload:**
    - `sub`: wallet address
    - `exp`: 24h
    - `hasAccessBadge`: boolean (optional)
- **Optional Enrichment Modal:**
    - Collects email (required), Twitter, and Discord handles (optional)
    - Sent to `POST /api/user/profile`

---

6. Backend Gating Logic

Option A: Galxe API Check (recommended)

- Query GraphQL to verify task completion and/or badge ownership
- If valid, allow access

Option B: NFT Check

- Read wallet NFTs via RPC
- If "Autonoma Access Badge" present, allow access

Pseudocode:

```
if (user.wallet.hasNFT("autonoma-access-badge")) {
  allowAccess();
} else {
  redirectToGalxe();
}
```

---

7. Metrics and Goals

Metric	Target
Completed onboarding tasks	10,000
New Twitter followers	8,000+
Discord members	6,000+
Chat accesses	5,000 active
NFTs minted	7,000+

---

8. Marketing Tactics

1. **Twitter Virals:** "Talk to the most insane AI ever — unlock access in 3 steps."
2. **Cyberpunk Visuals:** QR code posters linked to Galxe
3. **TikTok + Shorts:** Demo chat → CTA to unlock
4. **Web3 Influencers:** Paid micro-KOLs with affiliate codes
5. **Discord Gamification:** Leaderboard of top interacting users

---

9. Next Steps

- Deploy and verify campaign on Galxe
- Finalize soulbound NFT contract and metadata
- Integrate backend gating logic with authentication JWT
- Launch marketing with defined creatives and tracking

Autonoma Agent NFT Architecture

---

1. Concept Overview

Autonoma agents are packaged as fully functional digital workers represented by NFTs. Each NFT not only symbolizes identity but also links to an executable AI agent, a wallet, and monetization rights.

---

2. Token Standard and Metadata

- **Standard:** ERC-721 (with ERC-2981 for royalties)
- **Binding:** Optional ERC-6551 (Token Bound Account)
- **Metadata Fields:**
    - `name`: Agent display name
    - `description`: Summary of behavior
    - `promptHash`: Hash of base prompt stored off-chain
    - `endpoint`: URL for execution or agent relay
    - `created_by`: Creator wallet address
    - `created_at`: Timestamp
    - `model_id`: (optional) reference to LLM model or version

---

3. Agent Ownership and Wallet Binding

- Each agent NFT is linked to a **CDP Wallet**, enabling it to:
    - Receive payments from users
    - Hold assets (if ERC-6551 is used)
    - Pay for infra or interact with other contracts autonomously
- Ownership transfer (via NFT transfer) includes:

    - Transfer of all future usage revenue (via x402)
    - Admin access to update prompt/model metadata
    - Agent execution rights and stats history

---

4. Minting Flow

1. Creator defines agent behavior via onboarding UI
2. Prompt is uploaded to IPFS/Arweave (fallback: Supabase)
3. Metadata JSON is generated and pinned
4. NFT is minted via `mintAgent()` function
    - Contract collects Mint Fee (5–10%)
    - Optional royalties encoded via ERC-2981

---

5. Execution Layer

- Agent execution triggered via:
    - HTTP endpoint: `/agent/<id>/call`
    - x402 Pay-to-Execute: returns signed receipt
- Execution handled by:
    - Runpod container
    - Railway function
    - AgentKit endpoint
- Runtime context:
    - Wallet (CDP) address
    - Prompt and memory

    - External tools (e.g., webhook, Supabase memory, Drive context)

---

6. AgentKit SDK Responsibilities

- Validate x402 payment
- Load and parse metadata
- Route request to proper container or LLM
- Log execution (usage, timestamp, caller)
- Send earnings to NFT owner wallet

---

7. Marketplace and Royalties

- NFT agents may be resold on any ERC-721-compatible marketplace
- Platform enforces:
    - 5–10% creator royalty
    - Optional 2.5% platform fee (split with infra wallet)
- On resale, new owner inherits:
    - Revenue stream
    - Execution rights
    - Reputation/stats

---

8. Token-Gated Execution (Optional)

- Agents can be configured to require:
    - Minimum $AUTO balance
    - Holding a specific NFT

    - Time-locked access token
- Gating logic implemented at endpoint level

---

9. Transparency and Logs

- Public dashboard for each agent includes:
    - Total calls
    - Monthly revenue
    - Owner address
    - Execution latency
    - Last 10 callers (anonymized if needed)
- All logs stored on Supabase, viewable via REST API

---

10. Future Extensions

- Modular skill slots per agent
- Agent-level staking and delegations
- Agent DAOs with shared governance tokens
- Composable agents (one agent calling another)
- Federated agents with shared memory pools

Autonoma Authentication and Profile Layer

---

1. Objective

To implement a secure, wallet-based authentication system with optional user profile enrichment. This ensures minimal friction for Web3 users while enabling richer interaction and targeting.

---

2. Authentication Flow (Wallet Signature)

- **Step 1: Connect Wallet**
    - User connects Coinbase Wallet, MetaMask, or any EVM-compatible provider.
- **Step 2: Request Nonce**
    - Frontend calls: `GET /api/auth/nonce?wallet=0x...`
    - Backend generates random nonce, stores in Redis/in-memory with TTL (5–10 mins)
- **Step 3: Signature Request**
    - User signs message: `Sign this nonce: <nonce>`
- **Step 4: Verification**
    - Frontend calls: `POST /api/auth/verify`
    - Backend verifies:
        - Signature matches nonce
        - Recovered address matches `wallet`
    - If valid, issues JWT token:
        - `sub`: wallet address
        - `exp`: 24 hours

---

3. Token Storage and Headers

- JWT stored securely in browser (HttpOnly cookie or localStorage)

- Sent with all authenticated requests as `Authorization: Bearer <token>`

---

4. Profile Collection (Optional)

- **After Login:**
    - Check if user profile exists via `GET /api/user/profile?wallet=...`
    - If not, show modal to collect:
        - Required: Email
        - Optional: Twitter handle, Discord ID
- **Submission Endpoint:**
    - `POST /api/user/profile`

Payload:

```
{
 "wallet": "0x...",
 "email": "...",
 "twitter": "...",
 "discord": "..."
}
```

    -

---

5. Backend API Specification

- `GET /api/auth/nonce?wallet=...`
    - Returns: `{ nonce: "abc123" }`
- `POST /api/auth/verify`
    - Input: `{ wallet, signature }`
    - Returns: `{ token, user }`

- `GET /api/user/profile?wallet=...`
    - Returns user profile
- `POST /api/user/profile`
    - Upserts profile info

---

6. Database Schema

Table: users

Column	Type	Notes
id	UUID	Primary key
wallet	TEXT	Unique, checksummed address
email	TEXT	Required
twitter	TEXT	Optional
discord	TEXT	Optional
created_at	TIMESTAMP	Default: now()
updated_at	TIMESTAMP	Default: now()

---

7. Security Considerations

- Nonces must be unique, time-limited
- JWTs signed with backend secret
- Ethereum addresses checksummed
- Rate-limiting on all `/auth/*` routes
- User profile write access gated to the wallet owner

---

8. Future Enhancements

- Role-based access (e.g. creator, admin, moderator)
- Token-gated features in dashboard
- Integration with social graphs (Lens, Farcaster, etc.)
- Encrypted profile fields (for privacy-sensitive data)

Autonoma Monitoring Stack — Datadog Integration

---

1. Objective

Implement centralized observability using Datadog to monitor backend, serverless agents, and agent usage patterns. Ensure real-time tracking of system health, performance, and behavior anomalies.

---

2. Monitoring Scope

- Backend (Python FastAPI): API calls, errors, latency
- Serverless agents (RunPod): call volume, execution time, failure rates
- Agent usage logs: per-user and per-agent metrics
- Custom business metrics: task completion, onboarding steps, revenue logs

---

3. Tools and Libraries

Layer	Tool
General Observability	Datadog (APM, logs)
Backend Instrumentation	ddtrace, datadog SDK
Serverless Agents	Embedded SDK logging

Metric Tagging	Custom agent/user tags

---

4. Backend Setup

Dependencies:

```
pip install datadog ddtrace
```

Environment Variables:

```
DATADOG_API_KEY=...
DATADOG_APP_KEY=...
DATADOG_HOST=http-intake.logs.datadoghq.com
```

Directory Structure:

```
monitoring/
├── datadog/
│   ├── ddtrace_config.py
│   ├── logger.py
│   └── metrics.py
```

ddtrace_config.py:

```
from ddtrace import patch_all, tracer
import os

patch_all()
tracer.configure(hostname=os.getenv("DATADOG_HOST",
"http-intake.logs.datadoghq.com"), port=443)
```

metrics.py:

```
from datadog import initialize, api
import os

initialize(api_key=os.getenv("DATADOG_API_KEY"),
app_key=os.getenv("DATADOG_APP_KEY"))

def log_agent_usage(agent_id: str, user_id: str):
    api.Metric.send(
        metric="autonoma.agent.calls",
        points=1,
        tags=[f"agent:{agent_id}", f"user:{user_id}"]
```

)

logger.py:

```
from datadog import initialize
import logging
import os

initialize(api_key=os.getenv("DATADOG_API_KEY"),
app_key=os.getenv("DATADOG_APP_KEY"))
logger = logging.getLogger("autonoma")
logger.setLevel(logging.INFO)

def log_event(message: str, level: str = "info", tags: list = []):
    log_line = {"message": message, "tags": tags}
    getattr(logger, level)(log_line)
```

---

5. Backend Usage Example

```
from monitoring.datadog.metrics import log_agent_usage
from monitoring.datadog.logger import log_event

def process_agent_call(agent_id, user_id):
    log_event(f"Agent {agent_id} called", tags=[f"agent:{agent_id}", f"user:{user_id}"])
    log_agent_usage(agent_id, user_id)
```

---

6. RunPod Integration

In `agent_runpod_service/handler.py`:

```
from monitoring.datadog.logger import log_event
from monitoring.datadog.metrics import log_agent_usage

def handle_input(input_text: str, agent_id="default", user_id="anon") -> str:
    log_event("RunPod Agent Called", tags=[f"agent:{agent_id}", f"user:{user_id}"])
    log_agent_usage(agent_id, user_id)
    return agent.process(input_text)
```

---

7. Datadog Dashboards and Alerts

- **Metric:** `autonoma.agent.calls`

    - Tags: `agent:<id>`, `user:<id>`
    - Visualization: timeseries, bar chart by agent
- **Logs:** Search `message:"RunPod Agent Called"`
    - View: tags, timestamps, input trace, latency
- **Alerts:**
    - High error rate (5xx) on FastAPI endpoints
    - Latency spikes on agent execution
    - Drop in call volume per agent (weekly trend)

---

8. Future Extensions

- Log-to-metric conversion for financial KPIs
- Anomaly detection via Datadog Machine Learning
- Trace correlation with user sessions (JWT sub)
- Embedded usage explorer for creators (via API + dashboard UI)

Token Launch Strategy — Base Blockchain + CLMM + Market Maker

---

1. Objective

Launch the $AUTO token on the Base network using a Concentrated Liquidity Market Maker (CLMM) approach, supported by a dual-sided market maker (MM) strategy to ensure efficient price discovery, low slippage, and early liquidity.

---

2. Token Overview

- **Token Name:** Autonoma ($AUTO)

- **Standard:** ERC-20 (Base-compatible)
- **Total Supply:** 1,000,000,000 AUTO

Utility:

- Power agent creation, execution, and upgrades
- Unlock premium infra and model access
- Participate in governance of registry and fee systems
- Bind to agent NFTs for enhanced rights and revenue share

---

3. Supply Allocation

Allocation	%	Amount	Notes
CLMM Liquidity Pool	40%	400,000,000	For initial liquidity on Base DEX
Protocol Reserve	25%	250,000,000	Development, grants, ops
User Onboarding Airdrop	15%	150,000,000	Campaigns for creators and builders
Agent Rewards Vault	10%	100,000,000	Execution and hosting incentives
MM Reserve	10%	100,000,000	Active liquidity management on DEXs

---

4. Initial Liquidity Deployment

- **DEX:** Uniswap v3 on Base (or alternative with CLMM support)
- **USDC/AUTO Pool:**
    - USDC Seed: $10,000
    - AUTO: 400,000,000

- Initial Price (P0): 1 AUTO = $0.000025
- CLMM Range: $0.000015 — $0.000035 (±40% buffer around P0)

---

5. Market Maker Role

- **Bot Functions:**
  - Monitor CLMM pool state and ticks
  - Inject buys below $0.000025 and sells above $0.000025
  - Use dynamic spread strategy (±12.5%)
  - Rebalance token and USDC inventory
- **MM Budget:**
  - USDC: $2,000
  - AUTO: 100,000,000

---

6. Token Sinks and Utility Flow

- **Sink #1:** Burn upon agent creation or upgrade
- **Sink #2:** Fee payments for CDP Wallet ops and infra
- **Sink #3:** Locked with agent NFTs for activation rights

Utility Loop:

- User stakes AUTO → deploys agent → users call agent → revenue partially in AUTO → value captured → auto-burns or recycled

---

7. Launch Scenarios & Contingencies

Scenario A: Surge in Demand

- CLMM price breaches upper bound
- MM injects token supply at new bounds
- Vault may backstop in rare overflows (optional)

Scenario B: Price Drop / Sell Pressure

- CLMM converts to USDC
- MM provides buy support at lower tick range

Scenario C: MM Reserve Exhaustion

- Temporarily halt support
- Evaluate vault intervention or protocol buybacks

---

8. Execution Checklist

- Deploy $AUTO token contract (ERC-20 on Base)
- Setup CLMM pool on supported DEX
- Seed liquidity (USDC + AUTO)
- Launch MM bot (monitor, inject, rebalance)
- Begin gated onboarding with reward integration
- Enable sinks and infra-based payments

---

9. Post-Launch Analytics & Adjustments

- Monitor:
    - Pool health (liquidity, ticks, swaps)
    - Token velocity and holder count

    - Sink effectiveness (burn rate, wallet locks)
    - MM performance (spread profit, depth coverage)
- Adjust:
    - Range and depth of CLMM
    - Airdrop pacing and vault unlocks
    - Staking multipliers and infra discount rates

Autonoma Revenue Model — Streams and Fee Structures

---

1. Overview

Autonoma is monetized through direct value-aligned interactions with the agent lifecycle — from creation to execution and resale. All revenue streams are linked to real usage or infrastructure provisioning, avoiding speculation-based income.

---

2. Revenue Streams Summary

Stream	Trigger Event	Payer	Destination
Setup Fee	Agent creation	Agent creator	Autonoma treasury
Mint Fee	NFT mint	Agent creator	Contract fee split
Usage Fee	Each call to agent endpoint	End user	Shared: owner + infra
Infra Subscription	Monthly runtime plan	Agent owner	Autonoma treasury
Marketplace Royalty	NFT resale on secondary DEXs	NFT buyer/seller	Creator + platform split

---

3. Detailed Fee Descriptions

📅 Setup Fee

- **When Charged:** On agent initialization via UI or SDK
- **Amount:** $2–5 in stablecoins or equivalent $AUTO
- **Purpose:** Prevent spam, cover infra setup costs
- **Collected Via:** x402 or direct contract call

💋 Mint Fee

- **When Charged:** On `mintAgent()` function call
- **Amount:** 5–10% of mint value (or fixed)
- **Purpose:** Monetize NFT issuance process
- **Mechanism:** Auto-deducted in minting contract

♻️ Usage Fee (Pay-per-Call)

- **When Charged:** Every API call to agent
- **Amount:** 5–20% of call value
- **Routing:**
    - 80–95% to `ownerOf(tokenId)`
    - 5–20% to platform/infra wallet
- **Collected Via:** x402 receipt + smart middleware

🛠️ Infra Subscription (Revised Model)

- **When Charged:** Monthly based on selected plan
- **Tiers:**

Tier	Execution	Call Limit	Monthly Fee	Description
Free	Shared runtime	100 calls/month	$0	Throttled, community supported

Basic	Dedicated CPU	1,000 calls/month	$5	Ideal for creators and testing
Pro	Dedicated GPU	5,000 calls/month	$15	For high-throughput agents
Premium	Scalable compute	Custom	$30+	For production-grade deployment

- **Collected Via:** Stablecoin or $AUTO recurring
- **Managed Through:** Admin dashboard + webhook billing

🌐 Marketplace Royalty

- **When Charged:** NFT resale on OpenSea, LooksRare, etc.
- **Mechanism:** ERC-2981 royalty standard
- **Default Rate:** 7.5% (5% to creator, 2.5% to platform)
- **Adjustable By:** Agent creator at mint time

---

4. Transparent Logging

All transactions tied to revenue are logged and queryable:

- `agent_logs` table in Supabase
- Public REST API for:
    - Revenue per agent
    - Daily volume
    - Top earners

---

5. Forecast Example (1,000 Agents)

Metric	Volume	Revenue

Agents created	1,000	$3,000 (setup)
NFTs minted	1,000	$5,000 (mint)
Monthly calls	50,000	$2,500/mo
Infra subscriptions	200/mo	$1,500/mo
Resale volume ($/mo)	$100,000	$7,500/mo

---

6. Sustainability Strategy

- Emphasize usage-based income over speculation
- Align platform incentives with agent success
- Support creators with automated share delivery
- Leverage $AUTO sinks to reinforce token value
- Enforce mandatory runtime subscriptions for active agents

---

7. Future Revenue Enhancements

- Token staking for infra discount multipliers
- Fee-backed governance proposals (e.g. reallocating splits)
- Ad slot integration in public dashboards
- Creator referral revenue sharing