# FETCH: AGENTIC EXECUTION PLAN

Module D: Engineering Roadmap & Agent Tooling

Classification: INTERNAL / PRIVATE

Version: 2.0.0

# 1. PRE-PHASE: ENVIRONMENT & ANTIGRAVITY

**Objective:** Establish a high-fidelity context window for AI Agents to prevent hallucination regarding the hybrid infrastructure.

## 1.1. Model Context Protocol (MCP) Configuration

Configure the following MCP servers in ~/.cursor/mcp.json or equivalent:

1. **filesystem-server**:
   * **Scope:** Read/Write access to ./fetch-monorepo.
   * **Directive:** Agents must index the directory tree before proposing file changes.
2. **postgres-server** (Phase 1+):
   * **Connection:** postgresql://postgres:[PASSWORD]@[NODE\_A\_IP]:5432/fetch\_db
   * **Directive:** Agents must introspect information\_schema before writing SQL queries.
3. **docker-server**:
   * **Scope:** Read-only access to Node A Docker socket (via SSH tunnel or TCP).
   * **Directive:** Agents must verify container health (supabase-auth, supabase-rest) before deploying schema migrations.

## 1.2. Grounding Context (.cursorrules)

Create a .cursorrules file at the root containing:

PROJECT IDENTITY: Fetch (Privacy-Sovereign Executive Assistant).

PERSONA: Archimedes (Wise, Sassy, Empathetic Owl-Dog).

STACK:

* **Client:** React Native, Expo Router v3, Tamagui, TanStack Query, MMKV.
* **Server (Node A):** Synology NAS, Docker Compose, Supabase (Self-Hosted), Node.js Orchestrator.
* Brain (Node B): Python FastAPI, Ollama (Llama 3.1 / DeepSeek-R1), CUDA.  
  CONSTRAINT: No external AI APIs. Zero Trust Network (Tailscale).  
  CRITICAL: All database I/O on Node A must target the NVMe volume.

# 2. PHASED ROLLOUT OBJECTIVES

## Phase 1: The Nervous System (Infrastructure)

* **Goal:** Stand up the data layer and the application shell.
* **Deliverables:**
  1. Synology Docker Stack (Supabase) exposing ports 8000/8443.
  2. PostgreSQL Schema v1 (Emails, Auth, RLS).
  3. Android App Shell (Auth Screen + Navigation).
  4. **Milestone:** User can log in via Service Account Impersonation and see an empty Inbox.

## Phase 2: Obedience (Orchestration)

* **Goal:** Establish the "Closed-Loop Watchdog" and Batch Logic.
* **Deliverables:**
  1. Node.js Orchestrator (Watchdog) running on Node A.
  2. Wake-on-LAN + HTTP Polling mechanism verified.
  3. "The Deck" (Swipe UI) implemented in React Native.
  4. **Milestone:** Manual "Whistle" (Pull-to-refresh) wakes Node B.

## Phase 3: The Therapist (Intelligence)

* **Goal:** Activate the Brain (Node B) and RAG Pipeline.
* **Deliverables:**
  1. Python API (FastAPI) on Node B.
  2. Ollama integration (Llama 3.1 for Summary, DeepSeek-R1 for Drafts).
  3. ai\_exclusions CAS table and SHA-256 ingestion logic.
  4. **Milestone:** Incoming emails are summarized; Drafts include "Reasoning Trace."

## Phase 4: Watchdog (Hardware)

* **Goal:** Physical world hooks.
* **Deliverables:**
  1. Financial OCR pipeline (PDF -> Text -> JSON).
  2. Home Assistant integration (Calendar -> Lights).
  3. Backup Health Monitor dashboard.

# 3. AGENT WORKSTREAMS & PROMPTS

## 3.1. SysAdmin Agent (Node A Infrastructure)

Context: Synology DSM 7.2, Docker Compose.

Prompt:

ROLE: Senior DevOps Engineer.

TASK: Generate a docker-compose.yml for a self-hosted Supabase stack tailored for a Synology NAS.

CONSTRAINTS:

1. **Port Mapping:** Map internal Supabase ports (8000, 8443, 5432) to non-conflicting host ports (e.g., 5433 for PG to avoid Synology's internal Postgres).
2. **Volume Pinning:** All data volumes must map to /volume1/docker/fetch/data (Assumed NVMe pool).
3. **Services:** Include studio, kong, auth, rest, realtime, storage, meta, postgres.
4. Orchestrator: Add a custom node-orchestrator service building from ./backend/orchestrator.  
   OUTPUT: The docker-compose.yml, .env.example, and a Makefile for deployment.

## 3.2. Mobile Foundry Agent (Client Shell)

Context: Expo SDK 50, TypeScript, Tamagui.

Prompt:

ROLE: Principal React Native Engineer.

TASK: Initialize the "Fetch" mobile client.

REQUIREMENTS:

1. **Structure:** Use Expo Router v3. Create groups (auth) and (tabs).
2. **UI Library:** Install and configure tamagui with a custom theme (Teal/Orange).
3. **State:** Set up TanStack Query with persistQueryClient using MMKV as the storage adapter (Offline-First).
4. Auth: Implement a useSession hook that mocks Google Service Account masquerading (for Phase 1 dev).  
   OUTPUT: app/\_layout.tsx, app/(tabs)/\_layout.tsx, and lib/query-client.ts.

## 3.3. AI Engineer Agent (Node B Brain)

Context: Python 3.11, CUDA, Ollama.

Prompt:

ROLE: AI Systems Architect.

TASK: Build the Inference API for Node B.

STACK: FastAPI, ollama python lib, sentence-transformers.

ENDPOINTS:

1. POST /vectorize: Accept text, return 768d float array (using all-MiniLM-L6-v2).
2. POST /draft: Accept { email\_body, context\_docs[], persona\_instruction }.
   * Logic: Call ollama.generate(model='deepseek-r1').
   * System Prompt: Inject the "Archimedes" persona.
   * Output: Parse JSON to return { draft, reasoning\_trace, sass\_comment }.  
     CONSTRAINT: Implement a GPU check on startup. Fail if CUDA is unavailable.

# 4. CLASS DEFINITIONS (SOURCE OF TRUTH)

## 4.1. TypeScript Interfaces (Client/Orchestrator)

// Core Notification Payload  
export interface FetchNotification {  
 id: string;  
 type: 'BATCH\_READY' | 'EMERGENCY\_INTERRUPT' | 'WATCHDOG\_FAILURE';  
 priority: 'high' | 'normal' | 'low';  
 payload: {  
 batchSize?: number;  
 triggerKeyword?: string; // e.g., "suicide", "911"  
 nodeStatus?: 'offline' | 'unreachable';  
 };  
 timestamp: number;  
}  
  
// The Avatar State Machine  
export interface DogState {  
 mood: 'idle' | 'alert' | 'processing' | 'judging' | 'comforting';  
 sassLevel: number; // 0.0 (Pure Empathy) to 1.0 (Full Snark)  
 lastInteraction: number;  
 activeUser: 'melissa' | 'husband';  
}  
  
// Calendar Negotiation Object  
export interface NegotiationCard {  
 id: string;  
 emailId: string;  
 proposal: {  
 title: string;  
 start: string; // ISO 8601  
 end: string;  
 };  
 evidence: {  
 quote: string; // "Can you do Tuesday at 2?"  
 sourceMessageId: string;  
 };  
 reality: {  
 isClear: boolean;  
 conflictingEvent?: string;  
 };  
}

## 4.2. Python Pydantic Models (Inference API)

from pydantic import BaseModel, Field  
from typing import List, Optional  
  
class VectorizeRequest(BaseModel):  
 text: str  
 normalize: bool = True  
  
class DraftRequest(BaseModel):  
 email\_body: str  
 sender: str  
 # RAG Context: List of strings from historical emails  
 context\_documents: List[str]  
 # Anti-Cannibalization: Ensure these aren't AI generated  
 # (Handled upstream by Node A, but good to validate)  
   
class DraftResponse(BaseModel):  
 draft\_content: str  
 # The "Chain of Thought" from DeepSeek-R1  
 reasoning\_trace: str   
 # The persona's meta-commentary  
 archimedes\_remark: str   
 confidence\_score: float