

Based on the **Realco OS Architectural Blueprint** and the capabilities of **Google Antigravity** detailed in your sources, here is the execution plan.

Because Antigravity operates on an "**Agent-First**" paradigm (where you act as the Architect and agents execute tasks across the Editor, Terminal, and Browser), we will not start by writing React components. We will start by establishing the **Data Spine** and **Governance Rules**.

## Phase 1: Infrastructure & The "Context Lake" Setup

**Objective:** Connect Antigravity's reasoning engine to your existing data (Databricks) without moving the data.

- **Initialize the Workspace & Lead Agent**
- **Action:** Launch Antigravity and initialize a new workspace named Realco-OS-Core.
- **Command:** Spawn a "Lead Agent" to scaffold the repository structure.
- **Artifact Required:** The agent must generate an instructions.md (or .agent/instructions.md) file. This file will house the **Governance & Intelligence Control** rules (Module 1), explicitly stating: "No hallucination tolerance," "Read-first, enrich-second," and "Unit-level truth over averages" 1-3.
- **Establish the Model Context Protocol (MCP)**
- **Context:** Antigravity uses MCP to connect LLMs to tools and data sources 4, 5.
- **Action:** Direct the agent to configure an **MCP Server** for Databricks.
- **Why:** This allows the Antigravity agents to "see" your existing DLD transactions, Land Registry records, and Monday.com exports inside Databricks without you manually copying CSVs. This creates the "Context Lake" 5, 6.

## Phase 2: The Canonical Data & Identity Layer

**Objective:** Define the single source of truth before building any UI.

- **Schema Definition (Schema-First Prompting)**
- **Context:** Your sources emphasize that Unit is the atomic truth 3, 7.
- **Action:** Instruct the Antigravity agent to generate the **Canonical Data Schema** (Postgres or Firestore) based on your Databricks Unit and Transaction tables.
- **Constraint:** The agent must define strict relationships: Zone → Area → Master Project → Project → Unit. If the agent detects data in Databricks that doesn't fit this hierarchy (e.g., "orphan" listings), it must create an "Unresolved\_Data" queue, not guess the relationship 7, 8.
- **Identity Resolution Agent**
- **Action:** Task a specific agent to write the logic for **Module 3 (Identity Resolution)**.
- **Logic:** The agent must write a script that takes incoming records (from Monday or Pixxi), hashes their key attributes (Unit Code, Phone), and checks them against the Canonical Registry.
- **Verification:** Use Antigravity's "Terminal" surface to run a test script against a sample 100-row dataset to prove it deduplicates correctly 9, 10.

## Phase 3: The "Observer" & Intelligence Engines

**Objective:** Build the logic that processes data into insights.

- **The Observer Pattern Implementation**
- **Context:** The "Observer" creates the feed for the Pulse Agent 11.
- **Action:** Create a background service (Cloud Run function) via Antigravity that acts as the **Pattern Engine** 12.
- **Task:** It should scan the "New Listings" table every hour. If it detects a price drop >5% on a Unit tracked in the "Canonical" list, it generates a "Signal" object. It does *not* send a WhatsApp message yet; it only creates the Signal.
- **Market Intelligence (Aggregator)**
- **Action:** Build **Module 4 (Market & Area Intelligence)** using Databricks SQL queries via the MCP connection.
- **Task:** The agent should write a query that calculates "Price Per SqFt" and "Transaction Volume" trends for the last 3 months per Area.
- **Governance:** Enforce the rule: "Aggregation must originate from unit-level entities." The agent must prove the math by tracing the aggregate back to specific Unit IDs in the logs 13.

## Phase 4: Interface & Verification (The "Antigravity Special")

**Objective:** Deploy the UI and use Agents to test it.

- **Scaffold the "Realco Intelligence" Dashboard**
- **Action:** Instruct the agent to build the Next.js/React frontend using the shadcn/ui library (standard in your sources) 14, 15.
- **Components:** Create the "Inventory Intelligence" view and the "Market Heatmap."
- **Binding:** Bind these components directly to the Canonical Data APIs built in Phase 2.
- **Browser-Based Verification ("Computer Use")**
- **Context:** Antigravity agents can control the browser to test apps 16, 17.
- **Action:** Spin up a "QA Agent." Give it the instruction: "Log in as an Agent, search for 'Villa in Al Furjan', and verify that the Price Per SqFt matches the Databricks source."
- **Outcome:** The agent will click through the UI, take screenshots of the result, and compare it against the raw data query, providing a pass/fail Artifact 17, 18.

### Summary Checklist for Day 1 on Antigravity:

1. **Infrastructure:** Configure project.json and AGENTS.md with Governance Rules (Module 1).
2. **Connectivity:** Set up MCP connection to Databricks (The Brain).
3. **Data:** Define the Unit canonical entity (Module 3).
4. **Verification:** Run a "Headless" test where an agent fetches a unit from Databricks and confirms it matches the Canonical Schema.

**Start here.** Do not build the WhatsApp bot or the flashy dashboard until the **MCP connection to Databricks** and the **Canonical Schema** are verified by an agent.