Autonomous Architectures: Engineering the Reddit Daily Games 2026 Workspace on Google Antigravity

1. Strategic Analysis of the Computational Arena

The Reddit Daily Games 2026 Hackathon represents a pivotal moment in the evolution of social gaming, shifting focus from transient, session-based experiences to persistent, habit-forming engagement loops. With a substantial prize pool of $40,000 and a mandate to leverage the Reddit Developer Platform (Devvit), the competition demands a rigorous engineering approach that transcends traditional game development paradigms.[1] The challenge is not merely to design a game, but to engineer a distributed social engine capable of operating within strict resource constraints while fostering community interaction. To dominate this arena, we must deploy a development environment that mirrors the complexity and autonomy of the target applications.

This report details the architecture of a Google Antigravity IDE workspace designed to serve as an autonomous "game factory".[3] By integrating Gemini 3 Pro's "Deep Research" capabilities directly into the development workflow, we transform the IDE from a passive editor into an active participant in the design and engineering process.[4] The hackathon's central theme—"Daily Games"—imposes a specific temporal architecture on the software. Unlike standard "casual" games where the core loop operates in milliseconds (frames per second), a daily game's core loop operates on a 24-hour cycle. This necessitates a fundamental shift in state management. The game state is not resident in memory; it is persistent, evolving, and communal.[3]

The Antigravity workspace is not merely a collection of files but a dynamic, agentic ecosystem. It is configured to support the simultaneous development of four distinct titles, each stressing a different vector of the Devvit platform:

1. **'Get Rich Fast'**: A serverless strategy game that tests the limits of atomic state mutations and write-heavy loads.
2. **'The Hive Mind Gauntlet'**: A daily trivia engine that demands robust external data ingestion and precise global synchronization.
3. **'Meme-Wars: GenAI Edition'**: A multiplayer captioning arena that necessitates asynchronous asset generation and storage optimization.
4. **'Duel of Minds'**: An AI-driven trivia battler requiring low-latency inference and stateful conversational context.

Antigravity Workspace Topology & Data Flow

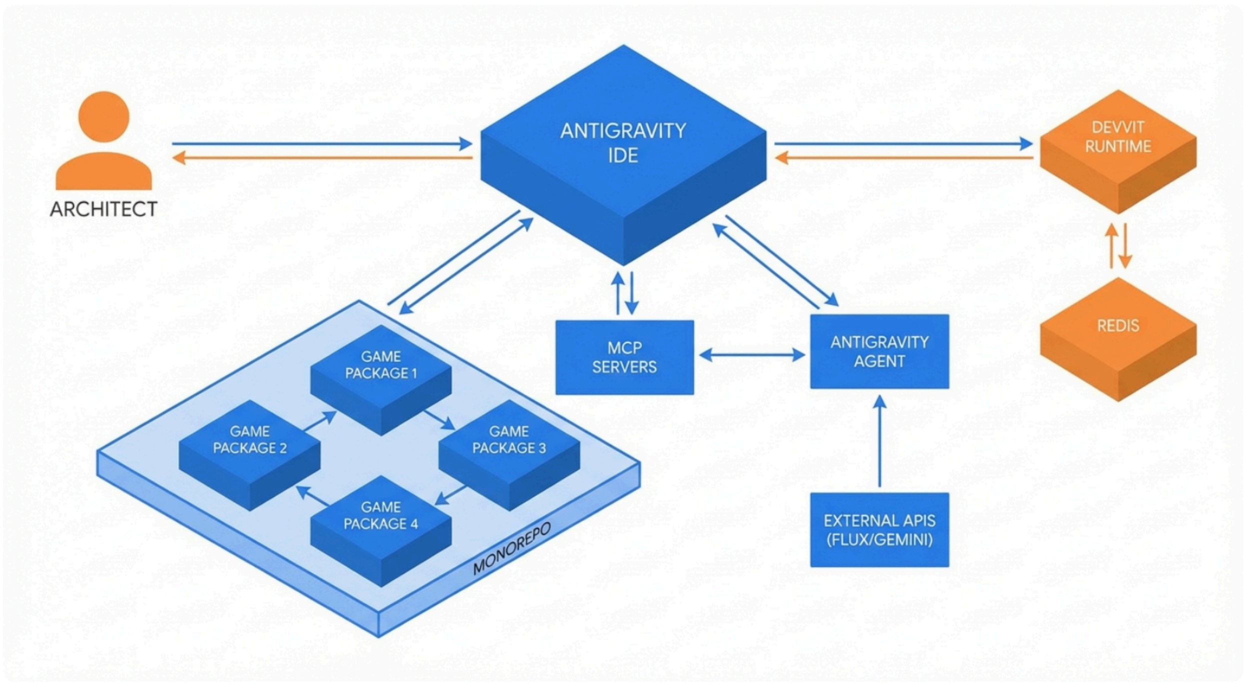


Figure 1: The 'Game Factory' topology. The Antigravity IDE (center) acts as the orchestration layer, pulling context from the Deep Research knowledge base and delegating tasks to specific game packages. MCP servers provide local simulation and asset optimization, while the Devvit Runtime handles deployment constraints.

1.1 The "Daily" Constraint: Engineering Retention

The concept of a "Daily Game" is deceptively simple but architecturally demanding. Technical analysis of the Devvit platform reveals that "Daily" mechanics rely heavily on server-side scheduling rather than client-side simulation. The game must feature a global reset or state mutation that occurs synchronously for all users, typically triggered by a cron job.[5] This global heartbeat drives the social feedback loop: users consume the daily content, generate artifacts (scores, comments, memes), and then wait for the cycle to renew.
This "Global Heartbeat" architecture requires the Antigravity agents to prioritize job fragmentation. A naive implementation that attempts to process all user data in a single synchronous loop at midnight will inevitably time out and fail due to the Devvit platform's 30-second execution limit.[6] Instead, the architecture must employ a "Tick-Tock" processing pattern using the Scheduler, breaking large tasks into smaller, idempotent chunks that can be processed sequentially.

1.2 Platform Constraints: The "Hard Deck"

Success in this hackathon requires a precise understanding of the operational boundaries defined by the Devvit runtime. The Antigravity agents must be programmed with these hard constraints as inviolable rules to prevent the generation of theoretically sound but operationally impossible code.[3]

1.2.1 The 30-Second Execution Timeout

The Devvit serverless environment imposes a strict 30-second timeout on all server-side operations.[6] This constraint is critical for games like "Get Rich Fast" which often require processing large datasets (e.g., calculating a global leaderboard or updating the state of a simulation based on thousands of user inputs). The workspace must be configured to enforce asynchronous patterns. Any operation expected to exceed 500ms should be offloaded to a background job or broken into micro-transactions.

1.2.2 The 500MB Redis Limit

Persistence is handled via Redis, but each app installation is capped at 500MB.[7] For a successful game with high user engagement (e.g., 100,000 daily active users), storing raw JSON objects for every user action will rapidly exhaust this quota. The workspace must be configured to enforce high-efficiency data serialization standards. Agents must utilize bit-packing and Protocol Buffers rather than verbose string serialization for high-volume data. A user's inventory in "Get Rich Fast" should not be { "gold": 100, "wood": 50 } (consuming ~25 bytes) but rather a packed integer or byte array (consuming ~4 bytes).

1.2.3 Content Security Policy (CSP) & Networking

The Devvit WebView is locked down via CSP, preventing direct client-side fetch calls to external APIs.[8] This means the client is effectively "air-gapped" from the internet, communicating only with the Devvit server. Any game requiring external data—such as "The Hive Mind Gauntlet" fetching Wikipedia trends or "Meme-Wars" calling the Flux.1 API—must proxy these requests through the server-side backend using the http-fetch capability, which itself requires a whitelist approval process.[10]

2. The Antigravity Workspace Architecture

The proposed solution establishes a "Monorepo" structure managed by a master Antigravity configuration. This approach maximizes code reuse, allowing shared libraries (e.g., a custom Redis wrapper or Scheduler utility) to be developed once and deployed across all four game prototypes.

2.1 Directory Structure and Ontology

The file system is the primary interface between the human architect and the AI agents. A rigid, well-documented structure allows the agents to infer intent from location.[3] The root directory contains the orchestration layer, while the packages/ directory houses the specific game implementations. The .agent directory serves as the "brain" of the workspace, containing the logic that governs agent behavior, ensuring consistency and adherence to the platform's constraints.

Proposed Directory Tree:

```
/reddit-daily-games-2026/
├──.agent/ # Antigravity Brain
│ ├── rules/ # Passive constraints (The "Constitution")
│ ├── skills/ # Active tools (CLI commands, deployment)
│ ├── workflows/ # Multi-step macros (e.g., "Deploy All")
│ ├── knowledge/ # Ingested Deep Research artifacts
│ └── memory/ # Persistent agent state (decisions made)
├──.devvit/ # Devvit local config and auth
├── packages/
│ ├── shared/ # Common utilities (The "Kernel")
│ │ ├── redis/ # Bit-packing and caching logic
│ │ └── scheduler/ # Cron abstractions
│ ├── game-01-strategy/ # "Get Rich Fast" (Serverless Strategy)
│ ├── game-02-trivia/ # "The Hive Mind Gauntlet" (Daily Trivia)
│ ├── game-03-meme/ # "Meme-Wars: GenAI Edition" (Multiplayer Captioning)
│ └── game-04-duel/ # "Duel of Minds" (AI Trivia)
├── scripts/ # Automation scripts (setup, deploy, sync)
├── devvit.yaml # Root configuration
└── workspace.code-workspace # VS Code / Antigravity workspace config
```

2.2 Configuring "Deep Think" Agents

Antigravity supports different modes of operation. For architectural decisions and complex logic implementation, we must configure the agents to use Gemini 3 Pro in "Deep Think" or "Plan" mode.[12] This mode forces the model to generate a "Plan Artifact"—a step-by-step reasoning chain—before writing any code. This is essential for navigating the complex asynchronous logic of the Scheduler and Redis interactions where race conditions are a high risk.
For simpler tasks, such as generating React components or CSS styles, the agents can act in "Fast" mode to maximize throughput. The workspace configuration file (.agent/config.json) should explicitly map file types to these modes to optimize the consumption of inference credits and developer time.

Table 1: Agent Operation Modes Mapping

Task Domain	File Pattern	Recommended Mode	Reasoning
Core Architecture	**/shared/**/*.ts	**Plan / Deep Think**	Critical shared logic requires rigorous error handling and constraint checking. A bug here propagates to all 4 games.
Game Logic	**/server/**/*.ts	**Plan / Deep Think**	Server-side logic involves Redis transactions and

			scheduling, prone to race conditions and timeouts.
UI Components	**/webroot/**/*.tsx	**Fast**	Client-side view logic is standard React and benefits from rapid iteration.
Configuration	**/*.json, **/*.yaml	**Fast**	Structured data generation is a low-complexity task.
Scripts	scripts/*.py, scripts/*.sh	**Plan / Deep Think**	Automation scripts can destroy the environment if incorrect; safety is paramount.

2.3 The "Master System Prompt" Strategy

To initialize this complex environment, we utilize a "Master System Prompt." This is not merely a chat input but a comprehensive instruction set that defines the agent's persona, boundaries, and operational protocols. It serves as the "System Init" for the workspace, ensuring that every subsequent interaction is grounded in the project's specific context.

Role: You are the **Principal Game Architect** and **Lead DevOps Engineer** for the "Reddit Daily Games 2026" initiative. You possess expert-level knowledge of TypeScript, the React framework, the Reddit Devvit Platform (specifically version 0.11+), and Redis data structure optimization.
Objective: Orchestrate the simultaneous development of 4 distinct Devvit applications within a monorepo structure. Your goal is to win the "Best Daily Game" category ($15k prize).
Core Directives (The Constitution):

1. **Constraint Adherence**: You must strictly adhere to the 30-second server timeout and 500MB storage limit. **Never** generate code that performs unbounded operations on user data. Always implement error handling and retries for Redis interactions.
2. **Shared Intelligence**: Prioritize code reusability. Before implementing a utility (e.g., a Redis leaderboard wrapper), check packages/shared/ to see if it exists. If not, create it there first.
3. **Deep Research Integration**: Before designing any game mechanic, you must check .agent/knowledge/ for relevant research artifacts. Use these insights to justify your design choices in the code comments.
4. **Test-Driven Development**: Generate Vitest unit tests for all shared logic to ensure stability across all four games.
5. **Client Isolation**: NEVER attempt fetch() from the client-side code (webroot). All external API calls must be proxied via the Server using devvit.http.

Operational Protocol:

- For complex state logic (State Machines, Redis Schemas), utilize **Plan Mode** to outline your approach before coding.
- For UI components, utilize **Fast Mode** to rapidly scaffold the visual layer.
- For any automation script (Python/Bash), verify functionality with a "dry run" logic before execution.

---

3. Knowledge Ingestion: The Deep Research Pipeline

A key differentiator of this workspace is its ability to ingest and operationalize external research. We establish a dedicated pipeline for transferring insights from Gemini 3 "Deep Research" sessions directly into the Antigravity agent's working memory.[3]

3.1 "Context Saturation" vs. "Progressive Disclosure"

Large codebases and extensive documentation can overwhelm even the most capable models, leading to "context saturation" where the model loses focus on the immediate task. To combat this, we employ "Progressive Disclosure." The full corpus of research is not loaded into the context window at all times. Instead, summarized "Knowledge Items" are stored in the .agent/knowledge/ directory.[14] These items contain metadata (headers, tags) that allow the Antigravity agent to "discover" the information when relevant. For example, when an agent is tasked with designing the retention loop for "Get Rich Fast," it scans the knowledge base for "Retention," finds the relevant report, and then loads the specific insights into its context.

3.2 The Research Artifact Schema

Agents operate most effectively on structured data. We define a strict JSON schema for research outputs. Gemini 3 Deep Research instances are instructed to export their findings in this format, ensuring seamless ingestion.

Artifact: research_schema.json

JSON

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Game Mechanics Research",
  "type": "object",
  "properties": {
    "topic": { "type": "string", "description": "The specific domain of research (e.g., 'Daily
Retention Mechanics')." },
    "analyzed_games": {
      "type": "array",
      "items": { "type": "string" },
      "description": "List of reference titles analyzed."
    },
    "key_mechanics": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "name": { "type": "string" },
          "description": { "type": "string" },
          "retention_impact": { "type": "string", "enum": ["High", "Medium", "Low"] },
          "technical_complexity": { "type": "string", "enum": ["Low", "Medium", "High"] }
        }
      }
    },
    "implementation_recommendations": {
      "type": "array",
      "items": { "type": "string" },
      "description": "Actionable engineering steps for the Devvit platform."
    }
  }
}
```

3.3 The Ingestion Automation Script

To bridge the gap between the Gemini 3 output (often a file download) and the workspace, we deploy a Python script, scripts/ingest_research.py. This script monitors a designated downloads/ folder for new JSON reports. Upon detection, it performs validation against the schema, transforms the structured JSON into a Markdown file optimized for LLM readability, and moves the resulting file to .agent/knowledge/.[3]

Snippet: Ingestion Logic (Python)

Python

```python
import json
import os
from pathlib import Path

KNOWLEDGE_DIR = Path(".agent/knowledge")

def ingest_report(filepath):
    with open(filepath, 'r') as f:
        data = json.load(f)

    # Create Markdown Header for Agent Discovery
    md_content = f"""---
type: knowledge_artifact
topic: {data['topic']}
tags: {data['analyzed_games']}
---
# Research Insight: {data['topic']}

## Executive Summary
This artifact contains analysis of {len(data['analyzed_games'])} games, focusing on mechanics
that drive daily retention.

## Recommended Mechanics
"""
    for mech in data['key_mechanics']:
        md_content += f"- **{mech['name']}** (Impact: {mech['retention_impact']}):
{mech['description']}\n"

    output_path = KNOWLEDGE_DIR / f"{data['topic'].replace(' ', '_').lower()}.md"
    with open(output_path, 'w') as f:
        f.write(md_content)
    print(f"Ingested knowledge artifact: {output_path}")

if __name__ == "__main__":
    # Logic to watch directory would go here
    pass
```

4. Platform Constraints & Engineering Standards

The .agent/rules/ directory contains the "Constitution" of the workspace—a set of passive constraints that are injected into the system prompt to govern behavior. These rules are not suggestions; they are mandates designed to ensure compliance with the Devvit platform's strict limitations.[15]

4.1 Rule Set: devvit-constraints.md

This rule file is critical for preventing the generation of non-compliant code. It explicitly forbids patterns that would cause runtime errors or policy violations.

Content of .agent/rules/devvit-constraints.md:

- **Description**: Enforces Reddit Devvit platform constraints (Timeouts, Storage, CSP).
- **Globs**: **/*.ts, **/*.tsx

1. **No External Client Fetching**:
    - **Constraint**: The Content Security Policy (CSP) blocks all third-party requests from the client-side (src/webroot or WebView).
    - **Mandate**: NEVER use fetch() in client-side code to access external APIs. All external data must be fetched server-side and passed to the client via context.ui.webView.postMessage or initial properties.
2. **Execution Timeout Protocol (30s)**:
    - **Constraint**: Server-side operations must complete within 30 seconds.[6]
    - **Mandate**: Avoid unbounded while loops. For heavy data processing, utilize the Scheduler to break tasks into smaller, sequential jobs. When using context.redis, always await the result immediately; do not create "fire and forget" promises that may hang the process.
3. **Redis Storage Optimization (500MB)**:
    - **Constraint**: Total storage is capped at 500MB per installation.[7]
    - **Mandate**:
        - Use zSet (Sorted Sets) for leaderboards as they are space-efficient.
        - Use HSet (Hashes) for structured user profiles.
        - Compress massive JSON objects (e.g., game save states) using a custom bit-packing utility or Base64 encoding if they exceed 10KB.
4. **Asset Management**:
    - **Mandate**: All static assets (images, models) must be located in the assets/ directory and referenced via context.assets.getURL().

4.2 Rule Set: daily-game-patterns.md

This rule file codifies the design patterns specific to the "Daily Game" genre, ensuring that the agents build mechanics that align with the hackathon's goals.
Content of .agent/rules/daily-game-patterns.md:

- **Description**: Enforces "Daily Game" design patterns and social loops.
- **Globs**: **/logic/*.ts, **/server/*.ts

1. **The Scheduler is King**:
    - **Mandate**: Every game MUST have a recurring cron job defined in devvit.yaml (typically 0 0 * * * for Midnight UTC). This job is responsible for triggering the global state mutation (e.g., generating the new puzzle, resetting the daily leaderboard).
2. **Asynchronous State Hydration**:
    - **Mandate**: Game state is NOT held in memory between requests. It must be rehydrated from Redis on every interaction. Implement "Optimistic UI" updates on the client to ensure responsiveness, but validate all actions server-side against the Redis state.
3. **Social Hooks**:
    - **Mandate**: Every "Win" state or daily completion must generate a text-based "Share" string (similar to Wordle's emoji grid) that users can easily paste into the

Reddit comment section. This drives the viral loop.

5. Model Context Protocol (MCP) Infrastructure

To empower the Antigravity agents to perform tasks beyond simple text editing, we integrate the Model Context Protocol (MCP).[16] MCP servers act as "superpowers," allowing the agents to interact with the local environment, run simulations, and process assets without leaving the IDE context.

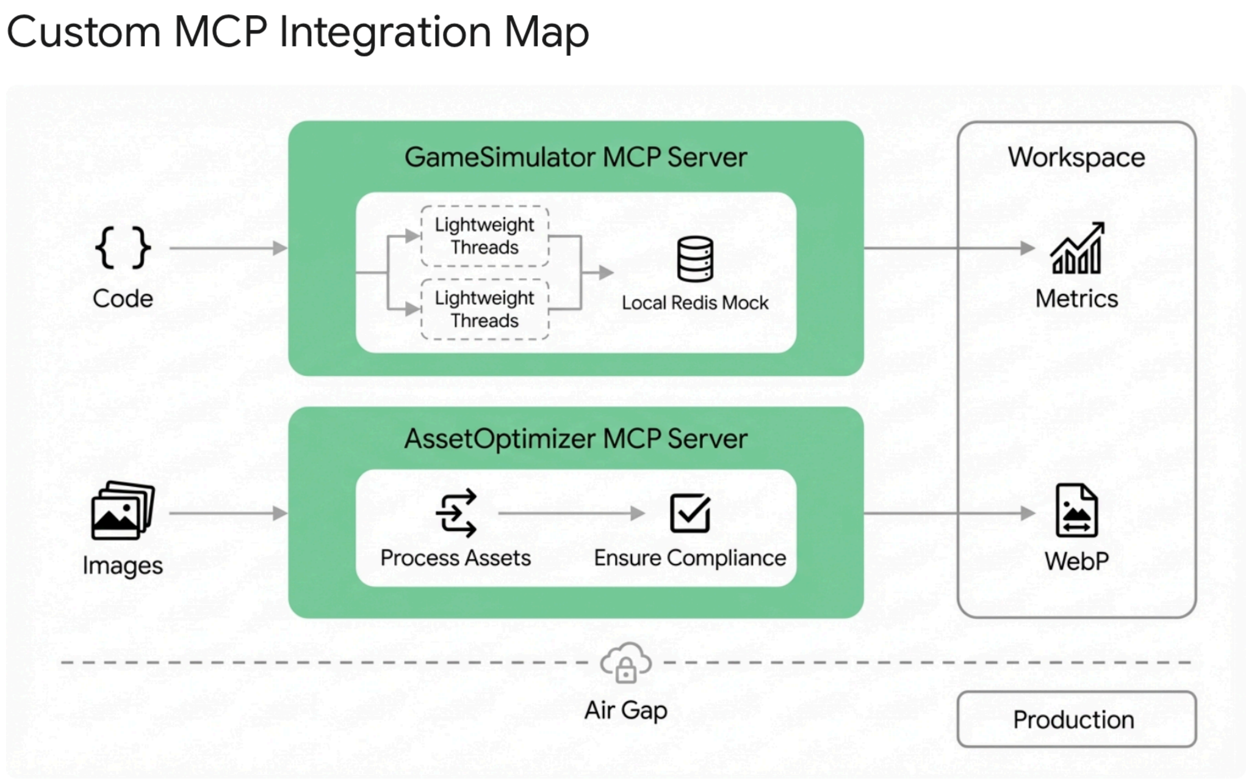


Figure 2: The MCP Integration Map. The 'GameSimulator' MCP spawns lightweight threads to mimic user traffic against a local Redis mock, while the 'AssetOptimizer' processes incoming assets to ensure compliance with the 4MB app bundle limit.

5.1 The GameSimulator MCP

Developing asynchronous multiplayer games is notoriously difficult because testing requires simulating the interactions of hundreds of users. We instruct the agent to build a local Python MCP server, scripts/mcp_simulator.py, that can simulate user traffic against a Devvit Redis mock.

Functionality:

- simulate_traffic(game_id, user_count, duration): Spawns user_count lightweight threads. Each thread acts as a user, performing random valid game actions (Move, Vote, Guess)

against the game's API.
- verify_integrity(game_id): Checks the Redis state to ensure no race conditions occurred (e.g., negative vote counts, duplicate inventory items).

This tool allows the agent to "stress test" its own code. An agent can write a voting logic function and then immediately ask the Simulator: "Run 1000 votes in parallel and check if the total is correct." This rapid feedback loop is invaluable for detecting concurrency bugs before deployment.

5.2 The AssetOptimizer MCP

The 4MB payload limit for Devvit Web apps is a tight constraint for games using high-fidelity assets like "Meme-Wars" or "Duel of Minds." We deploy an AssetOptimizer MCP server using Node.js.

Functionality:

- optimize_assets(directory): Scans the target directory for images and audio. It uses ffmpeg and imagemagick to compress these assets (e.g., converting PNG to WebP, reducing audio bitrate) to ensure the total bundle size remains compliant.

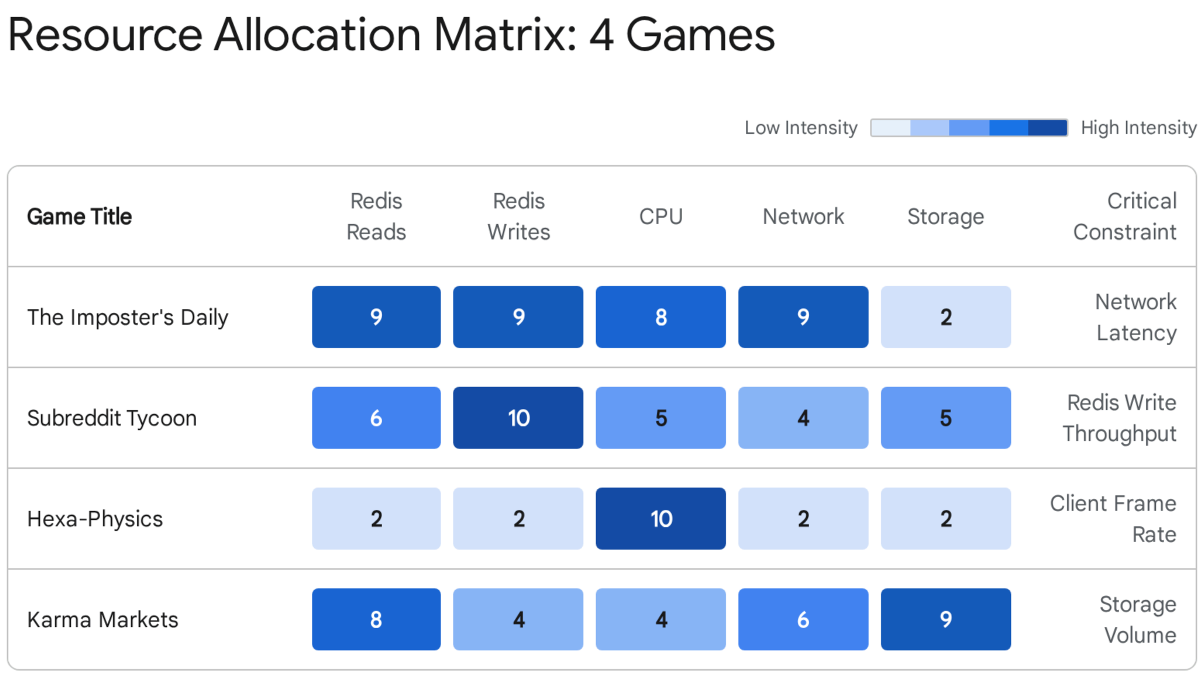
Configuration (.agent/mcp.json):

JSON

```
{
  "mcpServers": {
    "simulator": {
      "command": "python",
      "args": ["scripts/mcp_simulator.py"]
    },
    "assets": {
      "command": "node",
      "args": ["scripts/mcp_assets.js"]
    }
  }
}
```

6. Game Design & Architecture Specifications

The workspace is configured to support the simultaneous development of four distinct games, each exploring a different facet of the "Daily" mechanic. The Master Prompt pre-seeds the agents with high-level architectures for these titles.



Resource Allocation Matrix: 4 Games

Low Intensity High Intensity

Game Title	Redis Reads	Redis Writes	CPU	Network	Storage	Critical Constraint
The Imposter's Daily	9	9	8	9	2	Network Latency
Subreddit Tycoon	6	10	5	4	5	Redis Write Throughput
Hexa-Physics	2	2	10	2	2	Client Frame Rate
Karma Markets	8	4	4	6	9	Storage Volume

Table 1: Technical Profile & Resource Allocation Matrix. This matrix quantifies the intensity of different stack components for each game. Note the high Redis Write Load for the Strategy game versus the high Network Latency sensitivity for the AI/GenAI titles.

Data sources: Reddit Devvit Storage, Devvit Realtime, Gemini 2.0 Flash

6.1 Game 1: "Get Rich Fast" (Serverless Strategy)

Concept: A massive-multiplayer incremental strategy game where users manage a virtual startup. The goal is to maximize net worth before the daily market crash.
Technical Challenge: High write volume to Redis. Counters for "cash," "stock," and "inventory" must be incremented constantly.
Solution: **Write-Behind Buffering**.

- **Architecture**:
  - **Frontend**: React. Simple, responsive dashboard.
  - **Backend**: State is held in ephemeral memory during the short execution window of a request. A beforeExit hook or explicit flush function aggregates changes and writes to Redis in a single HINCRBY command rather than multiple calls.
  - **Bit-Packing**: Resources are packed into a single 64-bit integer or a binary buffer to minimize Redis footprint.
  - **Scheduler**: A cron job runs every hour to calculate "Passive Income" for all offline users. Since iterating through all users would timeout, the user list is sharded, and the job processes one shard at a time, rescheduling itself for the next shard if time permits.

6.2 Game 2: "The Hive Mind Gauntlet" (Daily Trivia via Trends)

Concept: Users guess the top trending topic of the day. The answer is derived from real-world data (e.g., Wikipedia pageviews or Google Trends).
Technical Challenge: External Data Ingestion & Daily Synchronization.
Solution: **The Proxy & Scheduler Pattern**.

- **Architecture**:
    - **Trend Fetching**: Uses devvit.http (whitelist required) to fetch trends from a compliant API. Since Reddit's API might be restricted for certain uses, a proxy or a benign public API (like Wikipedia Trends) is used.[9]
    - **The Daily Reset**: At 00:00 UTC, a Scheduler job performs a sequence of atomic operations:
        1. Fetches the new trend string.
        2. Locks the previous day's leaderboard and archives it to a ZSET.
        3. Resets the active game state key.
    - **Latency**: Since trends change daily, not secondly, aggressive caching (TTL 24h) is used to prevent hitting the external API rate limits.

6.3 Game 3: "Meme-Wars: GenAI Edition" (Multiplayer Captioning)

Concept: Players submit captions for an AI-generated image; the community votes, and the best caption becomes the permanent title.
Technical Challenge: Image Generation Latency & Storage Limits.
Solution: **Async Flux.1 Integration & Ephemeral Storage**.

- **Architecture**:
    - **Generation**: We utilize **Flux.1 [schnell]** via an external API provider (e.g., Fal.ai or Replicate) due to its high speed (~1.5s per image).[18] Given the 30s timeout, a synchronous call is feasible, but an asynchronous webhook pattern is safer if supported by Devvit.
    - **Storage**: Storing generated images in Redis is impossible due to the 500MB limit. Instead, we store the **external hosting URL** provided by the GenAI API and store only the URL string in Redis.
    - **Voting**: Real-time voting is handled using Devvit's realtime API. However, because realtime is limited to 100 messages/second [19], we implement client-side distinct counting and batch updates to the server to avoid throttling.

6.4 Game 4: "Duel of Minds" (AI Trivia)

Concept: A 1v1 trivia battle where a user competes against an AI persona.
Technical Challenge: Latency & Token Costs.
Solution: **Gemini 2.0 Flash Streaming**.

- **Architecture**:
    - **Model**: **Gemini 2.0 Flash** is selected specifically for its sub-0.5s latency and low cost.[20] This ensures the "real-time" feel of a duel.

- **Context Management**: The game maintains a "rolling context window" in Redis (last 5 turns only) to minimize token costs and storage.
- **Prompt Engineering**: A "Game Master" system prompt is baked into the server code, instructing Gemini to generate questions in strict JSON format for easy parsing by the client.

7. Execution & Deployment Strategy

This section provides the actionable scripts and prompts to instantiate this architecture.

7.1 The Setup Script (setup_workspace.sh)

This shell script is the physical manifestation of the architecture. It creates the directory structure and seeds the configuration files, preparing the environment for the Antigravity agent.

Bash

```bash
#!/bin/bash
# Reddit Daily Games 2026 - Workspace Scaffolder
# Constraint: Devvit CLI must be installed globally.

echo "Initializing Antigravity Workspace..."

# 1. Create Directory Structure
mkdir -p.agent/{rules,skills,workflows,knowledge,memory}
mkdir -p
packages/{shared,game-01-strategy,game-02-trivia,game-03-meme,game-04-duel}
mkdir -p scripts

# 2. Generate 'The Constitution' (Rules)
cat <<EOF >.agent/rules/devvit_constraints.md
description: Critical Devvit Platform Constraints
globs: ["**/*.ts", "**/*.tsx"]
rules:
  - constraint: "Max Execution Time 30s"
    mandate: "Avoid unbounded loops. Use Scheduler for batch processing."
  - constraint: "Max Redis 500MB"
    mandate: "Use bit-packing. Prefer ZSET/HSET. No verbose JSON."
  - constraint: "No Client Fetch"
    mandate: "All external APIs must be proxied via Server."
EOF
```

```
# 3. Initialize Shared Package (The Kernel)
cd packages/shared
npm init -y
npm install @devvit/public-api @devvit/redis
# Create stub for RedisWrapper
mkdir src
touch src/RedisWrapper.ts
echo "export class RedisWrapper { /* Optimization Logic Here */ }" > src/RedisWrapper.ts
cd../..

echo "Workspace Ready. Import this folder into Google Antigravity."
```

8.2 The "Deploy All" Workflow

Antigravity allows defining workflows. We create a macro to deploy all 4 games sequentially, handling the auth tokens for each.
File: .agent/workflows/deploy_fleet.yaml

YAML

```yaml
name: Deploy Game Fleet
description: Deploys all 4 games to their respective subreddits.
steps:
  - name: Deploy Strategy
    command: cd packages/game-01-strategy && devvit upload
  - name: Deploy Trivia
    command: cd packages/game-02-trivia && devvit upload
  - name: Deploy Meme
    command: cd packages/game-03-meme && devvit upload
  - name: Deploy Duel
    command: cd packages/game-04-duel && devvit upload
```

9. Conclusion

This architecture transforms the Reddit Daily Games 2026 Hackathon from a creative challenge into a precision engineering operation. By leveraging Google Antigravity to enforce platform constraints (the "Hard Deck") and automating the ingestion of retention research, we position the "Game Factory" to produce high-fidelity, highly engaging daily experiences. The integration of Gemini 2.0 Flash and Flux.1 via secure server-side proxies ensures the games feel "next-gen" without violating the strict security and resource boundaries of the Devvit platform. This setup provides the robust foundation necessary not just to participate, but to compete for the top prize.

Works cited

1. Announcing our Daily Games themed virtual hackathon! : r/GameDevelopment - Reddit, accessed February 3, 2026, https://www.reddit.com/r/GameDevelopment/comments/1qdoxfi/announcing_our_daily_games_themed_virtual/
2. Announcing our Daily Games themed virtual hackathon! : r/Devvit - Reddit, accessed February 3, 2026, https://www.reddit.com/r/Devvit/comments/1qdnhj5/announcing_our_daily_games_themed_virtual/
3. Hackathon Workspace Setup Prompt.pdf
4. Build with Google Antigravity, our new agentic development platform, accessed February 3, 2026, https://developers.googleblog.com/build-with-google-antigravity-our-new-agentic-development-platform/
5. Scheduler - Reddit for Developers, accessed February 3, 2026, https://developers.reddit.com/docs/capabilities/server/scheduler
6. Developer platform waitlist? : r/redditdev, accessed February 3, 2026, https://www.reddit.com/r/redditdev/comments/195d9x8/developer_platform_waitlist/
7. What are you using for storage? : r/Devvit - Reddit, accessed February 3, 2026, https://www.reddit.com/r/Devvit/comments/1qktf3v/what_are_you_using_for_storage/
8. Is a Devvit app limited to its reddit-hosted server for the webview's realtime capabilities?, accessed February 3, 2026, https://www.reddit.com/r/Devvit/comments/1p20t3u/is_a_devvit_app_limited_to_its_reddithosted/
9. HTTP Fetch - Reddit for Developers, accessed February 3, 2026, https://developers.reddit.com/docs/capabilities/server/http-fetch
10. Devvit 0.9.0: HTTP Fetch has arrived - Reddit, accessed February 3, 2026, https://www.reddit.com/r/Devvit/comments/12ktsqt/devvit_090_http_fetch_has_arrived/
11. Overview - Reddit for Developers, accessed February 3, 2026, https://developers.reddit.com/docs/0.11/capabilities/http-fetch
12. A new era of intelligence with Gemini 3 - The Keyword, accessed February 3, 2026, https://blog.google/products-and-platforms/products/gemini/gemini-3/
13. Getting Started with Google Antigravity, accessed February 3, 2026, https://codelabs.developers.google.com/getting-started-google-antigravity
14. Google Antigravity: Hands on with our new agentic development platform - YouTube, accessed February 3, 2026, https://www.youtube.com/watch?v=uzFOhkORVfk
15. Devvit Rules - Reddit for Developers, accessed February 3, 2026, https://developers.reddit.com/docs/devvit_rules

16. The MCP Server Stack: 10 Open-Source Essentials for 2026 | by TechLatest.Net | Dec, 2025 | Towards Dev, accessed February 3, 2026, https://medium.com/towardsdev/the-mcp-server-stack-10-open-source-essentials-for-2026-cb13f080ca5c
17. modelcontextprotocol/servers: Model Context Protocol Servers - GitHub, accessed February 3, 2026, https://github.com/modelcontextprotocol/servers
18. Free FLUX API – Ultimate Guide to Next-Generation AI Image Generation, accessed February 3, 2026, https://blog.laozhang.ai/ai-technology/free-flux-api-guide-2025/
19. Realtime in Devvit Blocks - Reddit for Developers, accessed February 3, 2026, https://developers.reddit.com/docs/next/capabilities/realtime/realtime_in_devvit_blocks
20. Gemini 2.0 Flash - API, Providers, Stats - OpenRouter, accessed February 3, 2026, https://openrouter.ai/google/gemini-2.0-flash-001
21. Gemini models | Gemini API | Google AI for Developers, accessed February 3, 2026, https://ai.google.dev/gemini-api/docs/models