This document outlines the revised architecture for **Strategist** (formerly PetCommand) as a **Serverless Thick Client**.
This model eliminates the need for an expensive 24/7 backend server. Instead, it leverages the user's local machine for processing and **GitHub Actions** for cloud-based data aggregation.

Project Strategist: Technical Documentation (v2.0)

1. Executive Summary

Strategist is a cross-platform desktop application and World of Warcraft addon suite. It provides "Just-in-Time" strategy injection, combat simulation, and economic arbitrage for Pet Battles.
Core Shift:

- **Old Model:** User <-> Proprietary Server <-> WoW
- **New Model:** User <-> **Static Cloud Data (GitHub)** <-> WoW

The "Brain" is now split between the **Desktop Client** (Electron) and **Scheduled Cloud Jobs** (GitHub Actions).

---

2. System Architecture

A. The Desktop Client (The Hub)

- **Technology: Electron** (React/TypeScript).
- **Why:** Allows building a modern, complex UI (graphs, drag-and-drop) that runs outside the game but can read/write files directly to the WoW SavedVariables folder.
- **Key Responsibility:** Fetching static JSON data, running local simulations, and syncing with the WoW addon.

B. The Cloud "Pulse" (Serverless Data)

- **Technology: GitHub Actions** (Scheduled Cron Jobs).
- **Cost:** $0/month.

- **Mechanism:**
    1. Every 6 hours, a Python script spins up on GitHub.
    2. Scrapes Xu-Fu / Wowhead / Blizzard API.
    3. Compiles master_strategy_db.json and market_prices.json.
    4. Pushes these files to a public **GitHub Pages** branch.
- **Result:** The Desktop App simply downloads these static files on startup.

C. The In-Game Addon (The Executor)

- **Technology:** Lua.
- **Role:** Minimalist logic. It receives instructions from the Desktop App (via SavedVariables) and provides the secure execution buttons.

---

3. Module Specifications

Module 1: Strategist (Core Combat)

Goal: inject the perfect team for the target NPC instantly.

- **The "Ghost" Database:**
    - The Desktop App downloads strategies.json (50MB+).
    - User selects "Dragonflight World Quests" in the App.
    - App writes a tiny, optimized table to StrategistSaved.lua containing *only* those 50 teams.
    - **Benefit:** WoW loads instantly; no memory bloat.
- **Simulacrum (The Offline Simulator):**
    - *Note:* Since no plug-and-play Python simulator exists, this component ports the logic from the open-source **Pet Battle Scripts (Lua)** into a local TypeScript/Python engine within the Electron app.
    - **Feature:** "Pre-Flight Check." Before you fly to a tamer, the App simulates your currently equipped team against the Tamer's static team 100 times in the background.
    - **Alert:** "Win Rate: 15%. Speed Check Failed on Turn 2. Recommendation: Swap Slot 3 for *Ikky*."

Module 2: Goblin (Economy & Arbitrage)

Goal: Replace Undermine Exchange with a direct "Blizzard -> App" pipeline.

- **Data Source:**
    - The GitHub Action hits the **Blizzard Game Data API** (Official) to fetch raw AH dumps for supported realms.
    - It processes this massive data into a lightweight price_list.json.
- **The Smuggler (Feature):**
    - App reads price_list.json.
    - App reads user's collection from StrategistSaved.lua.
    - **Output:** "You have 3 *Spectral Tiger Cubs*. Sell one on *Area 52* for 400k profit."
- **The Breeder (Feature):**
    - App highlights pets in your journal where the **Breed ID** (e.g., S/S) increases the value by >200% vs the market average.

Module 3: Executor (Input & Automation)

Goal: 1-Button Combat.

- **The Lua Kernel:**
    - Based on a fork of tdBattlePetScript.
    - Instead of writing scripts manually, the Addon listens for a generated script string injected by the Desktop App.
- **The Hardware Hook:**
    - Uses SecureActionButtonTemplate.
    - Dynamically rewrites the button's macro text based on the Game State (Turn #, Enemy Buffs).
    - **Safe Mode:** Requires 1 hardware event (key press) per action.

---

4. Data Pipeline & Schema

The Static JSON Database (Hosted on GitHub Pages)

File: v1/strategies/dragonflight.json

JSON

```
{
  "npc_id": 193244,
  "name": "Haniko",
  "teams": [
    {
      "id": "meta_v1",
      "pets": [
        {"speciesId": 2345, "breed": 3, "abilities": [111, 222, 333]},
        {"speciesId": 1234, "breed": 4, "abilities": [444, 555, 666]}
      ],
      "script": "use(1) [enemy.hp<500]..."
    }
  ]
}
```

The In-Game Injection (SavedVariables)

File: WTF/.../SavedVariables/Strategist.lua

Lua

```lua
StrategistDB = {
  ["TargetInject"] = {
    -- The App writes THIS specific table when you click "Sync"
    [193244] = "meta_v1"
  },
  ["CollectionCache"] = {
    -- The Addon writes this for the App to read
    ["1234-5678"] = { species=2345, breed=3, level=25 }
  }
}
```

---

5. Development Roadmap

Phase 1: The Skeleton (No AI yet)

1. **Electron App:** Build a basic "Hello World" app that locates the WoW installation folder.
2. **Scraper:** Write a Python script to scrape one category (e.g., "Shadowlands Family Exorcist") from Xu-Fu and save it as JSON.

3. **Addon:** Create a Lua addon that prints "NPC ID Detected: [12345]" to chat when targeting a tamer.

Phase 2: The Sync

1. **Reader:** Electron App parses SavedVariables/Strategist.lua to see user's pets.
2. **Writer:** Electron App writes a test team to the file.
3. **Loader:** Addon reads the file and equips the team using C_PetJournal.SetPetLoadOutInfo.

Phase 3: The Brains

1. **Logic Port:** Port the tdBattlePetScript logic to TypeScript for the "Simulacrum" engine.
2. **Goblin:** Set up the Blizzard API OAuth in the Python scraper to start fetching auction data.

Phase 4: The Polish

1. **Executor:** Implement the SecureActionButton for spacebar spamming.
2. **UI:** Add graphs and "Profit/Hour" metrics to the Electron dashboard.

---

6. Technical Constraints & Safety

- **ToS Compliance:** The "Executor" must **never** press the button for the user. It can only *change* what the button does. The user must physically press the key.
- **Rate Limits:** The Blizzard API has hourly limits. The GitHub Action handles this centrally so individual users don't need API keys.
- **File Locking:** WoW only reads SavedVariables on *Load* and writes on *Logout/Reload*. The Desktop App must ask the user to /reload to apply new strategies.