# Policy Watch: Architecture & Migration Strategy

**Date:** October 26, 2023

**Author:** Senior AWS Architect

**Status:** Strategic Review

## 1. Executive Summary

**Policy Watch** is a critical data pipeline designed to automate the monitoring, acquisition, and intelligence extraction of payer policy documents. The current "As-Is" state relies on a hybrid architecture of On-Premises legacy scripts (Producer) and AWS ECS services (Consumer).

The **Migration Strategy** focuses on a transition to a **Cloud-Native, Monorepo Architecture**. This consolidation aims to eliminate On-Premises dependencies, standardize tooling, improve reliability through event-driven orchestration, and reduce operational overhead.

## 2. As-Is Architecture Overview

The system operates on a **Producer-Consumer** model with five core components:

| **Component** | **Role** | **Location** | **Schedule** | **Input** | **Output** |
| --- | --- | --- | --- | --- | --- |
| **1. Policy Loader** | Producer (Scraper) | On-Prem Server | Every 4 Hours | Payer Portals | S3 (PDF), Postgres (Metadata) |
| **2. Policy Extractor** | Consumer (Parser) | AWS ECS Service | Every 2 Hours | S3 (PDF) | S3 (JSON), Postgres (Scores) |
| **3. Policy Analyzer** | Consumer (Logic) | AWS ECS Service | Every 2 Hours | Extractor JSON | S3 (Impact JSON) |
| **4. Rules Loader** | Producer (Legacy) | On-Prem Server | Manual | Oracle (Atlas DB) | S3 (Consolidated JSON) |
| **5. Rules Processor** | Consumer (ML) | On-Prem Script | Manual | S3 (JSON) | Postgres (Embeddings) |

### Key Architectural Risks (As-Is)

* **IP Reputation:** On-Prem scrapers risk being blocked if moved to Cloud without proxy strategies.
* **Cost Inefficiency:** 24/7 ECS Services are idle 90% of the time (running only every 2 hours).
* **Fragile Scheduling:** Independent Cron jobs lead to race conditions (e.g., Analyzer running before Rules are updated).
* **Legacy Debt:** Critical Rules Loader logic relies on manual Perl scripts and Oracle DB connectivity.

## 3. Migration Strategy: The Monorepo Path

The target state consolidates all components into a single repository structure (packages/tools/), deploying them as **ECS Fargate Tasks** orchestrated by **AWS Step Functions** and **EventBridge**.

### Module 1: Policy Loader (The Scraper)

* **Strategy:** **Rewrite & Replatform (Option A)**
* **Action:** Migrate Python scraping logic to packages/tools/policy\_retriever.py.
* **Enhancement:** Implement **Direct S3 Upload** (bypassing the "Rule File Service" API).
* **Critical Requirement:** Implement **IP Rotation/Proxy Service** (e.g., BrightData) to prevent WAF blocking when running from AWS public IPs.

### Module 2: Policy Extractor (The Parser)

* **Strategy:** **Lift & Shift + Optimize**
* **Action:** Convert the 24/7 Service to a **Scheduled ECS Task** triggered by EventBridge.
* **Enhancement:** Add DocumentParser shared utility for HTML & PDF support.
* **Critical Requirement:** Implement **Dead Letter Queue (DLQ)** for failed parses and ensure gpt-4o-mini model compatibility.

### Module 3: Policy Analyzer (The Logic)

* **Strategy:** **Port & Orchestrate**
* **Action:** Port logic to packages/tools/policy\_analyzer.py.
* **Enhancement:** Use **Step Functions** to trigger this *only* after the Rules Processor has successfully updated embeddings.
* **Critical Requirement:** Validate Top-N matching logic consistency during migration.

### Module 4: Rules Loader (The Legacy Migration)

* **Strategy:** **Modernize (Perl to Python)**
* **Action:** Rewrite legacy Perl scripts into Python (packages/rules/rules\_loader.py).
* **Enhancement:** Automate manual triggers via EventBridge.
* **Critical Requirement:** Establish **Site-to-Site VPN** or **Direct Connect** for connectivity to On-Prem Oracle (Atlas DB). This is a high-risk dependency.

### Module 5: Rules Processor (The ML Engine)

* **Strategy:** **Event-Driven Automation**
* **Action:** Migrate to ECS Task (packages/tools/rules\_processor.py).
* **Enhancement:** Trigger automatically via **EventBridge** (RulesLoadedSuccess event) instead of manual invocation.
* **Critical Requirement:** Ensure strict ordering: Loader -> Processor -> Analyzer.

## 4. Architect's Review & Recommendations

### 1. Network & IP Reputation

* **Risk:** Moving scrapers to AWS Public IPs invites immediate blocking by Payer Portals (WAFs).
* **Resolution:** Do not rely on AWS IPs. Budget for and implement a commercial **Proxy Rotation Service** as a standard interface for the Policy Loader.

### 2. Orchestration vs. Scheduling

* **Risk:** Independent Cron schedules (e.g., "Every 2 Hours") are fragile and prone to race conditions (stale data).
* **Resolution:** Adopt **AWS Step Functions**. Define the workflow explicitly:
  + Start -> Parallel(Loader, Rules Chain) -> Extractor -> Analyzer -> End.

### 3. Legacy Migration Safety

* **Risk:** Rewriting Perl logic to Python carries high business risk (logic bugs).
* **Resolution:** Implement **"Shadow Mode"**. Run the new Python ECS task alongside the legacy Perl script for 2 weeks, comparing outputs to S3 without overwriting production data, before full cutover.

### 4. Infrastructure Prerequisites

* **Immediate Action:** Request **Site-to-Site VPN** or verify Direct Connect status for Oracle DB access.
* **Immediate Action:** Provision **AWS Secrets Manager** for all DB credentials and API Keys (OpenAI, Proxy Service).

## 5. Conclusion

The proposed migration to a Monorepo architecture using ECS Fargate is technically sound and strategically correct. It reduces operational toil and modernizes the stack. However, success hinges on addressing the **Network/IP risks** for the scraper and the **Connectivity risks** for the legacy database integration immediately.