# Policy Watch: Master Migration Strategy & Architecture

**Status:** Final Strategic Review (Hybrid SSH + ECS on EC2)

## 1. Executive Summary

**Policy Watch** is a mission-critical data pipeline for monitoring payer policy documents. The objective of this migration is to modernize the orchestration and intelligence layers while mitigating high risks associated with legacy network and database dependencies.

**Strategic Pivot:**

Initial plans for a "Cloud-Native" lift-and-shift have been revised to a **Hybrid Architecture**.

* **Retained On-Premises:** The **Policy Loader** (Scraper) and **Rules Loader** (Oracle Integration) will remain on existing servers to preserve trusted IP reputations and legacy database connectivity.
* **Cloud Modernization:** The **Policy Extractor**, **Analyzer**, and **Rules Processor** will migrate to **AWS ECS (running on EC2)**.
* **Orchestration:** A central **AWS Step Functions** workflow will coordinate both environments, using secure **SSH commands** to trigger On-Prem tasks.

## 2. As-Is Architecture vs. Target Hybrid State

| **Module** | **Role** | **As-Is Location** | **Target Location** | **Migration Action** |
| --- | --- | --- | --- | --- |
| **1. Policy Loader** | Scraper | On-Prem Server | **On-Prem Server** | Keep script local. Trigger via SSH from AWS. |
| **2. Policy Extractor** | Parser | ECS Service (Idle) | **AWS ECS (EC2)** | Convert to Scheduled Task. Add DLQ/Retries. |
| **3. Policy Analyzer** | Logic Engine | ECS Service | **AWS ECS (EC2)** | Port to Monorepo. Trigger via Step Functions. |
| **4. Rules Loader** | Legacy Data | On-Prem Server | **On-Prem Server** | Keep Perl script local. Trigger via SSH from AWS. |
| **5. Rules Processor** | ML Engine | On-Prem Script | **On-Prem Server** | Keep local. Trigger via local hook or SSH. |

## 3. Detailed Migration Strategy

### 3.1. The "Hybrid Control Plane" (Orchestration)

**AWS Step Functions** acts as the single source of truth for the entire pipeline. It manages the state, retries, and dependencies across the hybrid divide.

* **Trigger:** Amazon EventBridge Scheduler (Every 4 Hours).
* **Pattern:** **Task Token Callback**.
  1. Step Functions generates a unique token.
  2. It pauses and waits.
  3. A Lambda function executes an SSH command on the On-Prem server, passing the token.
  4. The On-Prem script runs, finishes, and calls back to AWS with the token to resume the workflow.

### 3.2. Module-Specific Plans

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

* **Challenge:** Moving scraping logic to AWS introduces IP reputation risks (WAF blocking).
* **Strategy:** **Hybrid SSH Invocation**.
* **Implementation:**
  + Deploy a lightweight Python wrapper (loader\_wrapper.py) on the On-Prem server.
  + Configure AWS Lambda (ssh\_invoker) to connect via VPN/Direct Connect.
  + **No API Development Required:** We avoid building complex FastAPI wrappers by using standard SSH protocols.

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

* **Challenge:** Existing ECS Service runs 24/7 but processes data only every 2 hours (High Cost).
* **Strategy:** **Event-Driven ECS Task**.
* **Infrastructure:** **ECS on EC2** (Fargate unavailable).
  + Use **Capacity Providers** to scale EC2 instances to zero when idle.
* **Resilience:** Replace SQS Dead Letter Queues with **Step Functions Native Retries** and **SNS Alerts** for immediate failure visibility.

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

* **Challenge:** Race conditions where Analysis runs before Rules are updated.
* **Strategy:** **Strict dependency**.
* **Implementation:** Step Functions will *only* transition to the Analyzer state after receiving a "Success" signal from the Rules Loader/Processor branch.

#### **Module 4: Rules Loader (Legacy Oracle)**

* **Challenge:** Rewriting Perl/Oracle logic is high risk and requires complex DB drivers.
* **Strategy:** **Hybrid SSH Wrapper**.
* **Implementation:**
  + Retain rules\_loader.pl On-Prem.
  + Trigger via SSH similarly to the Policy Loader.
  + This eliminates the need for AWS-to-Oracle connectivity.

#### **Module 5: Rules Processor (ML/Embeddings)**

* **Challenge:** Currently a manual/on-demand process.
* **Strategy:** **Automated Chaining**.
* **Implementation:** The On-Prem wrapper for the Rules Loader will automatically trigger the Rules Processor script locally upon successful completion of the Perl script.

## 4. Architect's Critical Recommendations

### 4.1. Network Connectivity (The Lifeline)

The Hybrid architecture depends entirely on the stability of the link between AWS and the On-Prem Data Center.

* **Requirement:** Stable **Site-to-Site VPN** or **Direct Connect**.
* **Validation:** Use the "Connectivity Pathfinder" plan (deploying a simple echo probe) *before* migrating any business logic.

### 4.2. SSH Security

* **Key Management:** SSH Private Keys must **never** be hardcoded. Store them in **AWS Secrets Manager**.
* **Least Privilege:** The SSH user on the On-Prem server (svc\_aws\_runner) should have restricted permissions, only able to execute the specific wrapper scripts.

### 4.3. Failure Handling (No Silent Failures)

* **Timeout:** Configure strict timeouts in Step Functions (e.g., 4 hours for Loader). If the On-Prem server hangs, AWS must know.
* **Alerting:** Use a Catch block in Step Functions to route all failures (Timeouts, Script Errors) to an **SNS Topic** that emails the DevOps team immediately.

### 4.4. ECS Capacity Management

* Since Fargate is not used, **Cluster Capacity Management** is critical.
* Configure **Auto Scaling Groups (ASG)** with **Managed Scaling** enabled to ensure EC2 instances spin up when tasks are queued and terminate when idle to save costs.

## 5. Conclusion & Next Steps

This plan represents the lowest-risk, highest-value path forward. By keeping the "eyes and ears" (Scrapers/Loaders) On-Premises, we avoid external blocking and internal database complexity. By moving the "brain" (Orchestration/Analysis) to AWS, we gain visibility, reliability, and scale.

**Immediate Next Steps:**

1. **Network Validation:** Run probe.py from AWS VPC to On-Prem.
2. **SSH Setup:** Provision service accounts and keys.
3. **Wrapper Deployment:** Push loader\_wrapper.py to the On-Prem server.