# Policy Loader: End-to-End Migration Guide (Hybrid SSH Pattern)

**Module:** Policy Loader (Scraper)

**Migration Type:** Hybrid - SSH Invocation

**Orchestration:** AWS Step Functions + EventBridge

## 1. Overview & Strategy

The **Policy Loader** is responsible for scraping payer portals for policy documents (PDFs). Currently, it runs as a Python script on an On-Prem server, triggered by a local Cron job.

**The Strategy:**

* **Code:** Remains effectively "As-Is" on the On-Prem server to leverage trusted IPs.
* **Trigger:** Moved from local Cron to **AWS EventBridge**.
* **Orchestration:** **AWS Step Functions** triggers the job remotely via SSH.
* **Monitoring:** Uses the **Task Token Callback Pattern** to ensure AWS knows when the job finishes (or fails).

## 2. Step-by-Step Implementation Guide

### Phase 1: On-Prem Preparation (The "Target")

We need to prepare the On-Prem server to accept commands and report back status.

**Step 1.1: Deploy the Wrapper Script**

Instead of modifying the legacy scraper extensively, we deploy a lightweight Python wrapper (loader\_wrapper.py) alongside it.

* **Role:**
  1. Accepts a taskToken from the command line.
  2. Runs the legacy scraper (policy\_loader\_main.py).
  3. Captures exit code/errors.
  4. Calls AWS Step Functions API (SendTaskSuccess / SendTaskFailure) using the token.

**Code Snippet (loader\_wrapper.py):**

import sys  
import subprocess  
import boto3  
import argparse  
  
# Parse Arguments  
parser = argparse.ArgumentParser()  
parser.add\_argument("--token", required=True, help="Step Functions Task Token")  
args = parser.parse\_args()  
  
# Initialize AWS Client (Using local credentials or IAM Roles Anywhere)  
client = boto3.client('stepfunctions', region\_name='us-east-1')  
  
try:  
 # 1. Run the Legacy Scraper  
 print("Starting Policy Loader...")  
 # 'check=True' raises CalledProcessError if script fails  
 result = subprocess.run(["python3", "policy\_loader\_main.py"], check=True, capture\_output=True, text=True)  
   
 # 2. On Success: Send Callback  
 print("Job Succeeded. Sending Callback...")  
 client.send\_task\_success(  
 taskToken=args.token,  
 output='{"status": "success", "files\_processed": "See Logs"}'  
 )  
  
except subprocess.CalledProcessError as e:  
 # 3. On Script Failure: Send Failure Callback  
 error\_msg = f"Script Failed. Exit Code: {e.returncode}. Stderr: {e.stderr}"  
 print(error\_msg)  
 client.send\_task\_failure(  
 taskToken=args.token,  
 error="ScraperError",  
 cause=error\_msg[:250] # Truncate to fit limit  
 )  
  
except Exception as e:  
 # 4. On Wrapper Failure  
 client.send\_task\_failure(  
 taskToken=args.token,  
 error="WrapperError",  
 cause=str(e)  
 )

**Step 1.2: Configure IAM Roles (On-Prem)**

The wrapper script needs permission to call stepfunctions:SendTaskSuccess.

* **Option A (Standard):** Create an IAM User svc\_policy\_loader with a policy allowing states:SendTask\*. Configure ~/.aws/credentials on the server.
* **Option B (Modern):** Use **AWS IAM Roles Anywhere** to authenticate the on-prem server via x.509 certificates (more secure).

### Phase 2: AWS Configuration (The "Controller")

**Step 2.1: Infrastructure Setup**

* **VPC:** Ensure the VPC has a **VPN Connection** or **Direct Connect** to the On-Prem network.
* **Security Groups:**
  + **Lambda SG:** Allow Outbound TCP 22 (SSH) to the On-Prem Server IP.
  + **On-Prem Firewall:** Allow Inbound TCP 22 from the Lambda SG CIDR.

**Step 2.2: The "SSH Invoker" Lambda**

We need a Lambda function that Step Functions can call to perform the actual SSH connection.

**Lambda Code Snippet (ssh\_invoker.py):**

import boto3  
import paramiko  
import os  
  
def lambda\_handler(event, context):  
 # 1. Get Inputs from Step Functions  
 token = event['token'] # Step Functions passes this automatically  
 host = os.environ['ON\_PREM\_IP']  
 user = os.environ['SSH\_USER']  
 # Securely retrieve SSH Key from Secrets Manager (omitted for brevity)  
 pkey = get\_key\_from\_secrets\_manager()   
  
 # 2. Establish SSH Connection  
 ssh = paramiko.SSHClient()  
 ssh.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())  
 ssh.connect(host, username=user, pkey=pkey)  
  
 # 3. Construct Command  
 # Run in background (nohup) so SSH can disconnect immediately  
 cmd = f"nohup python3 /opt/scripts/loader\_wrapper.py --token '{token}' > /tmp/loader.log 2>&1 &"  
  
 # 4. Execute  
 stdin, stdout, stderr = ssh.exec\_command(cmd)  
   
 # 5. Return (Step Function stays in 'Wait' state until Callback)  
 return {"status": "Command Sent", "host": host}

### Phase 3: The Orchestration Workflow (Step Functions)

**Step 3.1: Define the State Machine**

We define a workflow that generates the token, calls the Lambda, and waits.

**ASL Definition (JSON Snippet):**

{  
 "StartAt": "InvokePolicyLoader",  
 "States": {  
 "InvokePolicyLoader": {  
 "Type": "Task",  
 "Resource": "arn:aws:states:::lambda:invoke.waitForTaskToken",   
 "Parameters": {  
 "FunctionName": "ssh\_invoker\_function",  
 "Payload": {  
 "token.$": "$$.Task.Token" // <--- Magic happens here: Generates Token  
 }  
 },  
 "TimeoutSeconds": 14400, // 4 Hours (Job Limit)  
 "Retry": [ { "ErrorEquals": ["States.TaskFailed"], "IntervalSeconds": 60, "MaxAttempts": 3 } ],  
 "Catch": [ { "ErrorEquals": ["States.Timeout"], "Next": "NotifyFailure" } ],  
 "Next": "SuccessState"  
 },  
 "NotifyFailure": {  
 "Type": "Task",  
 "Resource": "arn:aws:states:::sns:publish",  
 "Parameters": {  
 "TopicArn": "arn:aws:sns:us-east-1:123456789012:Alerts",  
 "Message": "Policy Loader STUCK or FAILED on Premise!"  
 },  
 "End": true  
 },  
 "SuccessState": {  
 "Type": "Succeed"  
 }  
 }  
}

## 3. Operational Flow: "The Life of a Job"

**1. Schedule Trigger (08:00 AM)**

* **EventBridge** fires the "Start Execution" signal.

**2. Token Generation**

* Step Functions enters the InvokePolicyLoader state.
* It automatically generates a unique token: AAAA-1234-BBBB-5678.
* It pauses execution ("Wait" mode).

**3. The Handoff (AWS -> On-Prem)**

* The ssh\_invoker Lambda runs.
* It SSHs into the On-Prem server.
* It runs: python3 loader\_wrapper.py --token AAAA-1234-BBBB-5678 &
* The Lambda finishes successfully. Step Functions *remains paused*.

**4. Execution (On-Prem)**

* The loader\_wrapper.py starts the heavy policy\_loader\_main.py.
* The job runs for 45 minutes scraping PDFs.

**5. The Callback (On-Prem -> AWS)**

* **Scenario A (Success):** The scraper finishes. The wrapper catches exit code 0. It calls SendTaskSuccess(AAAA-1234-BBBB-5678).
* **Scenario B (Crash):** The scraper crashes. The wrapper catches exit code 1. It calls SendTaskFailure(...).
* **Scenario C (Stuck/Server Fire):** The wrapper never calls back.

**6. Resolution (AWS)**

* **Success:** Step Functions receives the callback, unpauses, and moves to the next state (e.g., trigger Policy Extractor).
* **Failure:** Step Functions receives the error and sends an SNS Alert.
* **Timeout:** If no callback is received after 4 hours, Step Functions times out and sends an SNS Alert ("Job Stuck").

## 4. Monitoring & Visualization

* **Console View:** In the AWS Step Functions Console, you will see a visual graph. The InvokePolicyLoader box will turn **Blue** (Running) while waiting for the token. It turns **Green** (Success) when the callback arrives, or **Red** (Fail) if it times out/errors.
* **Logging:** All SSH attempts are logged in **CloudWatch Logs** (from the Lambda). All script outputs (from the wrapper) can be sent to CloudWatch via the CloudWatch Agent on-prem, or just kept in local logs.

## 5. Prerequisites Checklist

1. [ ] **Network:** VPN verified. Lambda can ping On-Prem IP.
2. [ ] **SSH Key:** Private key stored in AWS Secrets Manager. Public key in ~/.ssh/authorized\_keys on-prem.
3. [ ] **IAM:** Role created for loader\_wrapper.py to call Step Functions API.
4. [ ] **Code:** loader\_wrapper.py deployed to /opt/scripts/ on-prem.