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

**Date:** October 26, 2023

**Module:** Rules Loader

**Migration Type:** Hybrid - SSH Invocation

**Orchestration:** AWS Step Functions + EventBridge

## 1. Overview & Strategy

The **Rules Loader** is a critical legacy component responsible for fetching business rules from an On-Premises Oracle database (Atlas DB) and consolidating them into JSON format for the cloud pipeline. It currently runs as a manual Perl script (rules\_loader.pl) on an On-Prem server.

**The Strategy:**

* **Code:** The legacy Perl script remains "As-Is" on the On-Prem server to avoid complex Oracle DB driver migration and rewrite risks.
* **Trigger:** Automated via **AWS EventBridge** (replacing manual invocation).
* **Orchestration:** **AWS Step Functions** triggers the job remotely via **Secure SSH**.
* **Monitoring:** Uses the **Task Token Callback Pattern** to ensure AWS Step Functions waits for the job to complete before proceeding to the Rules Processor.

## 2. Step-by-Step Implementation Guide

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

We need to prepare the On-Prem server to accept the remote command and execute the legacy Perl script safely.

**Step 1.1: Deploy the Wrapper Script**

We will deploy a Python wrapper (rules\_wrapper.py) alongside the legacy Perl script. This wrapper handles the asynchronous communication with AWS.

* **Role:**
  1. Accepts a taskToken and script\_path from the SSH command.
  2. Executes the legacy Perl script (rules\_loader.pl).
  3. Captures the exit code and STDOUT/STDERR.
  4. Calls AWS Step Functions API (SendTaskSuccess / SendTaskFailure) using the token.

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

import sys  
import subprocess  
import boto3  
import argparse  
import logging  
  
# Configure Logging  
logging.basicConfig(filename='/var/log/rules\_loader\_wrapper.log', level=logging.INFO,   
 format='%(asctime)s %(message)s')  
  
# Parse Arguments  
parser = argparse.ArgumentParser()  
parser.add\_argument("--token", required=True, help="Step Functions Task Token")  
parser.add\_argument("--script", default="/opt/legacy/rules\_loader.pl", help="Path to legacy script")  
args = parser.parse\_args()  
  
# Initialize AWS Client  
# Assumes ~/.aws/credentials or IAM Roles Anywhere is configured  
client = boto3.client('stepfunctions', region\_name='us-east-1')  
  
try:  
 logging.info(f"Starting Rules Loader. Script: {args.script}")  
   
 # 1. Run the Legacy Perl Script  
 # 'check=True' raises CalledProcessError on non-zero exit code  
 result = subprocess.run(["perl", args.script], check=True, capture\_output=True, text=True)  
   
 logging.info("Perl script completed successfully.")  
   
 # 2. On Success: Send Callback  
 client.send\_task\_success(  
 taskToken=args.token,  
 output='{"status": "success", "message": "Rules Consolidated to S3"}'  
 )  
  
except subprocess.CalledProcessError as e:  
 # 3. On Script Failure: Send Failure Callback  
 error\_msg = f"Perl Script Failed. Exit Code: {e.returncode}. Stderr: {e.stderr}"  
 logging.error(error\_msg)  
   
 client.send\_task\_failure(  
 taskToken=args.token,  
 error="LegacyScriptError",  
 cause=error\_msg[:250] # Truncate to fit API limit  
 )  
  
except Exception as e:  
 # 4. On Wrapper/System Failure  
 logging.error(f"Wrapper Infrastructure Error: {str(e)}")  
 client.send\_task\_failure(  
 taskToken=args.token,  
 error="WrapperInfraError",  
 cause=str(e)  
 )

**Step 1.2: Configure Prerequisites (On-Prem)**

1. **Python 3:** Ensure Python 3.9+ and boto3 are installed (pip install boto3).
2. **Permissions:** Ensure the user running the SSH command (e.g., svc\_aws\_runner) has execute permissions on rules\_loader.pl.
3. **AWS Access:** Configure credentials or IAM Roles Anywhere certificate to allow states:SendTaskSuccess and states:SendTaskFailure.

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

**Step 2.1: Infrastructure & Connectivity**

* **VPN/Direct Connect:** Verify connectivity between the AWS VPC Private Subnet and the On-Prem Server IP (e.g., 10.20.1.50).
* **Security Groups:**
  + **Lambda SG:** Allow Outbound TCP 22 to 10.20.1.50.
  + **NACLs:** Verify Network ACLs do not block ephemeral return ports.

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

Reuse the same ssh\_invoker.py Lambda used for the Policy Loader, or deploy a shared one. It simply executes a command remotely.

**Lambda Configuration:**

* **Environment Variables:**
  + ON\_PREM\_IP: 10.20.1.50
  + SSH\_USER: svc\_aws\_runner
  + SSH\_KEY\_SECRET\_NAME: prod/ssh/onprem\_key

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

**Step 3.1: Define the State Machine**

This Step Function orchestrates the parallel execution of Policy Loader and Rules Loader. Below is the specific definition for the **Rules Loader Branch**.

**ASL Definition (JSON Snippet):**

{  
 "StartAt": "ParallelIngestion",  
 "States": {  
 "ParallelIngestion": {  
 "Type": "Parallel",  
 "Next": "TriggerRulesProcessor",  
 "Branches": [  
 {  
 "StartAt": "InvokeRulesLoader",  
 "States": {  
 "InvokeRulesLoader": {  
 "Type": "Task",  
 "Resource": "arn:aws:states:::lambda:invoke.waitForTaskToken",   
 "Parameters": {  
 "FunctionName": "ssh\_invoker\_function",  
 "Payload": {  
 "token.$": "$$.Task.Token",  
 "command": "nohup python3 /opt/scripts/rules\_wrapper.py --token '$$.Task.Token' &"  
 }  
 },  
 "TimeoutSeconds": 7200, // 2 Hours Max  
 "Retry": [ { "ErrorEquals": ["States.TaskFailed"], "IntervalSeconds": 300, "MaxAttempts": 2 } ],  
 "Catch": [ { "ErrorEquals": ["States.Timeout", "LegacyScriptError"], "Next": "NotifyRulesFailure" } ],  
 "End": true  
 },  
 "NotifyRulesFailure": {  
 "Type": "Task",  
 "Resource": "arn:aws:states:::sns:publish",  
 "Parameters": {  
 "TopicArn": "arn:aws:sns:us-east-1:123456789012:CriticalAlerts",  
 "Message": "CRITICAL: Rules Loader Failed. Old rules will be used."  
 },  
 "End": true  
 }  
 }  
 },  
 {  
 "Comment": "Parallel Branch for Policy Loader (See separate guide)"  
 }  
 ]  
 },  
 "TriggerRulesProcessor": {  
 "Type": "Task",  
 "Resource": "arn:aws:states:::lambda:invoke",  
 "Comment": "Only runs if Rules Loader succeeds",  
 "End": true  
 }  
 }  
}

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

**1. Trigger (EventBridge)**

* **Schedule:** Every 4 Hours (synchronized with Policy Loader).
* **Action:** Step Functions starts execution.

**2. Token Generation & SSH**

* Step Functions enters InvokeRulesLoader.
* It generates Token: TOKEN\_XYZ\_123.
* Lambda connects via SSH: ssh svc\_aws\_runner@10.20.1.50.
* Command Executed: python3 rules\_wrapper.py --token 'TOKEN\_XYZ\_123' &
* Lambda returns success (Command sent). Step Functions enters **Wait** state.

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

* rules\_wrapper.py starts.
* It launches perl rules\_loader.pl.
* The Perl script connects to the local Oracle DB, fetches rules, and uploads a JSON to S3. This takes ~30 minutes.

**4. The Callback**

* **Success:** Perl script exits with code 0. Wrapper calls client.send\_task\_success(taskToken='TOKEN\_XYZ\_123').
* **Failure:** Perl script creates an empty file or exits with error. Wrapper calls client.send\_task\_failure.

**5. Workflow Continuation**

* Step Functions receives the Success signal.
* It moves to the TriggerRulesProcessor step (which might trigger *another* SSH command to run the embedding generator, or run an ECS task if that logic was migrated).

## 4. Failure Scenarios & Recovery

| **Scenario** | **System Behavior** | **Alerting** |
| --- | --- | --- |
| **VPN Down** | Lambda cannot connect via SSH. Throws exception. | Step Function retries Lambda 3 times, then alerts SNS. |
| **Script Crash** | Perl script exits with error. Wrapper catches it and sends TaskFailure. | Step Function catches failure, alerts SNS. Pipeline stops. |
| **Server Hang** | Wrapper never calls back. | Step Function hits TimeoutSeconds (2 hours), alerts SNS. |
| **Bad Token** | Wrapper fails to authenticate with AWS. | Check on-prem logs /var/log/rules\_loader\_wrapper.log. |

## 5. Prerequisites Checklist

1. [ ] **Legacy Script Location:** Verify rules\_loader.pl path (e.g., /opt/legacy/).
2. [ ] **Python Environment:** Install Python 3 and boto3 on the Oracle server.
3. [ ] **SSH Key:** Generate new key pair. Add public key to authorized\_keys on Oracle server. Store private key in AWS Secrets Manager.
4. [ ] **Firewall:** Ensure AWS Lambda subnet CIDR is whitelisted for SSH.
5. [ ] **IAM:** Ensure the On-Prem credentials have permission to talk to the specific Step Function ARN.