Weather Alert System using AWS Lambda

Prerak

Thursday 10th July, 2025

Contents

1 Overview

The weather alert system is an AWS Lambda function that evaluates environmental rules and conditions, fetching weather data from PostgreSQL, applying rule-based logic, and sending alerts using SNS, SQS, or App channels. It supports both scheduled runs and HTTP POST API triggers.

2 Lambda Handler

Main orchestration function that handles API or scheduled trigger, retrieves rules from DynamoDB, evaluates weather data, and triggers notifications.

Pseudo Code

```
function lambda_handler(event, context):
    connect to PostgreSQL with retry
    if event is API POST:
        parse rule_id and farm_id from body
    else:
        fetch all rules
    for each rule:
        validate_rule(rule)
        for each farm:
            weather_data = get_majority_weather_data(...)
            for data in weather_data:
                if evaluate_conditions(data, rule):
                    severity = determine_severity(...)
                    for action in rule.actions:
                        message = compose_message(...)
                        for channel in select_channels(...):
                            send_notification(channel, message,
                                ...)
                    optionally queue SQS or store report
    return JSON response with results
```

3 Rule Validation: `validate_rule`

Ensures the rule has correct structure, types, and allowed operators/metrics.

Pseudo Code

```
function validate_rule(rule):
    apply legacy field conversions
    check for missing or invalid fields
    for each condition:
        recursively verify operator, metric, and value format
    for each action:
        ensure type and message are present
    return (is_valid, error_message)
```

4 Weather Aggregation: `get_majority_weather_data`

Fetches and aggregates data using a majority rule or fallback average.

Pseudo Code

```
function get_majority_weather_data(cursor, rule, farm_id):
    query PostgreSQL for latest data
    simulate multiple sources (4 APIs)
    for each timestamp:
        if 2 or more sources agree:
            take majority value
        else:
            average numeric values
    return list of aggregated weather records
```

5 Condition Evaluation: `evaluate_conditions`

Evaluates nested AND/OR/NOT/SEQUENCE logic or direct comparison.

Pseudo Code

```
function evaluate_conditions(data, conditions, ...):
    if condition is a list:
        recursively evaluate all items
    if operator is AND/OR/NOT/SEQUENCE:
        evaluate sub_conditions
    else:
        evaluate_condition(data, condition)
    return (is_valid, explanation)
```

6 Determine Severity

This function classifies each rule evaluation into **low**, **medium**, or **high** severity based on metric thresholds and optionally on delta values (e.g., change in temperature).

Threshold Table

Defined thresholds for weather metrics under `weather` rule type:

- **Temperature** (`temperature_c`): Low $\geq$ 32°C, Medium $\geq$ 38°C, High $\geq$ 42°C
- **Humidity** (`humidity_percent`): Low $\geq$ 70%, Medium $\geq$ 85%, High $\geq$ 95%
- **Rainfall** (`rainfall_mm`): Low $\geq$ 5 mm, Medium $\geq$ 25 mm, High $\geq$ 60 mm
- **Chance of Rain** (`chance_of_rain_percent`): Low $\geq$ 50%, Medium $\geq$ 80%, High $\geq$ 95%
- **Wind Speed** (`wind_speed_mps`): Low $\geq$ 10 m/s, Medium $\geq$ 18 m/s, High $\geq$ 25 m/s

Delta-based Escalation

If metric change (delta) exceeds threshold, severity is escalated:

- **Temperature change** (`delta`) $>$ **5°C**: Increase severity

- **Humidity change $>$ 20%**
- **Rainfall change $>$ 30 mm**

Pseudo Code

```
function determine_severity(rule_type, metric, value, delta=None):
    fetch thresholds for rule_type and metric
    if value >= high threshold:
        severity = 'high'
    elif value >= medium threshold:
        severity = 'medium'
    elif value >= low threshold:
        severity = 'low'
    if delta and abs(delta) > delta threshold:
        increase severity by one level
    return severity
```

Pseudo Code

```
function determine_severity(rule_type, metric, value, delta):
    check thresholds from predefined map
    escalate severity if delta exceeds threshold
    return severity level
```

7 Compose Message

Generates a formatted stakeholder-specific message using rule templates and runtime values.

Pseudo Code

```
function compose_message(action, rule, data, ...):
    select template based on stakeholder and language
    inject variables like farm_id, metric, value, delta, timestamp
    handle special cases like trend or sequence data
    return message string
```

8 Condition Evaluation and Alert Triggering

The system evaluates rule conditions against real-time or forecasted weather data. Based on the evaluation, alerts are triggered according to stakeholder-specific rules and severity thresholds.

Alert Policy

An alert is sent if either of the following is true:

- A rule is triggered (i.e., its conditions evaluate to `True`).
- The resulting severity is classified as `high`.

This ensures that critical conditions are never missed and are always communicated, even when normal channel logic might skip notifications.

Reporting and Notification Behavior

- **If rule is triggered but severity is low :**
  - Alerts are sent only if the stakeholder's logic allows (e.g., management or science team).
  - If stakeholder has no eligible channel, result is logged but not sent.
- **If severity is high:**
  - Alert is always sent, regardless of stakeholder logic.
  - Overrides any "quiet hours" for all roles except farmers.
- **In all cases:**
  - An entry is added to the final `results` list returned by the Lambda function.
  - For science team, a full report is stored (DynamoDB and S3).

Updated Evaluation Logic (Pseudo Code)

Pseudo Code

```
function evaluate_rule_and_send_alert(rule, data, farm_id):
    for each record in data:
        is_triggered = evaluate_conditions(record, rule.conditions)
        if is_triggered:
            severity = determine_severity(rule, record)
            message = compose_message(...)

            // Always alert if triggered or high severity
            if is_triggered or severity == 'high':
                channels = select_channels(rule.stakeholder,
                    severity, record.timestamp)
                for ch in channels:
                    send_notification(ch, message, ...)
            if rule.stakeholder == 'science_team':
                store_science_team_report(...)

            return {'triggered': True, 'severity': severity}
    return {'triggered': False}
```

9 Queue Batch Notification

Forwards medium/low severity messages to science team via SQS for batching.

Pseudo Code

```
function queue_batch_notification(farm_id, stakeholder, rule_id,
    ...):
    package message in JSON
    send to preconfigured SQS queue
```

10 Resolve Conflict

Resolves multiple triggered rules based on conflict strategy (priority, severity).

Pseudo Code

```
function resolve_conflict(rules, triggered_actions):
    if strategy == 'highest_priority':
        return rule with max priority
    elif strategy == 'most_severe':
        return rule with highest severity
    else:
        return first matching rule
```