



Problem: Disaster Prediction and Alert System API

Scenario:

You are tasked with building a **Disaster Prediction and Alert System API** for a government agency. This API will predict potential disaster risks (such as floods, earthquakes, and wildfires) for specified regions and send alerts to affected communities. The system integrates with external APIs to gather real-time environmental data and uses a simple scoring algorithm to assess risk levels.

Requirements:

1. API Endpoints:

- **POST /api/regions:** Allows users to add regions with specific location coordinates and types of disasters they want to monitor.
- **POST /api/alert-settings:** Allows users to configure alert settings for each region, including thresholds for disaster risk scores.
- **GET /api/disaster-risks:** Triggers the disaster risk assessment, fetching real-time environmental data for all regions and calculating risk scores. This endpoint should return risk levels for each region and indicate if any alerts should be sent.
- **POST /api/alerts/send:** Sends an alert for regions identified as high-risk and stores the alert in the database. Integrate with a messaging API (e.g., Twilio, SendGrid) to send alerts via SMS or email.
- **GET /api/alerts:** Returns a list of recent alerts for each region, stored in a database.

2. Input Data:

- **Regions:**
 - **Region ID:** Unique identifier for each region.
 - **Location Coordinates:** Latitude and longitude of the region.
 - **Disaster Types:** List of disaster types to monitor (e.g., flood, wildfire, earthquake).

- **Alert Settings:**
 - **Region ID:** Identifier for the region.
 - **Disaster Type:** Type of disaster (must match one monitored by the region).
 - **Threshold Score:** Risk score threshold that triggers an alert for this disaster type.

3. **Data Sources and Risk Calculation:**

- **External Data Sources:** Integrate with APIs such as OpenWeather or Earthquake data sources (like USGS) to fetch real-time data, such as rainfall levels, seismic activity, and temperature.
- **Risk Calculation:** Use the fetched data to calculate a risk score for each disaster type in the region. For example:
 - **Flood Risk:** Based on rainfall data in mm, e.g., rainfall over 50mm has a high risk.
 - **Earthquake Risk:** Based on seismic activity data, e.g., a magnitude over 5.0 has a high risk.
 - **Wildfire Risk:** Based on temperature and humidity data, e.g., high temperatures with low humidity increase the risk.
- **Threshold Check:** Compare the calculated risk scores against the configured thresholds. If any score meets or exceeds the threshold, generate an alert.

4. **Output:**

- **Disaster Risk Report:** Return a list for each region containing:
 - **Region ID**
 - **Disaster Type**
 - **Risk Score**
 - **Risk Level:** Low, Medium, or High based on the risk score.
 - **Alert Triggered:** True/False, indicating whether an alert has been triggered.
- **Alert Data:** When an alert is sent, it should include:

- **Region ID**
- **Disaster Type**
- **Risk Level**
- **Alert Message:** Detailed message including the reason for the alert.
- **Timestamp:** Time of alert generation.

5. Special Features:



- **Data Caching with Redis:** Store fetched environmental data and calculated risk scores in Redis with a 15-minute expiration to avoid redundant API calls and improve performance.
- **Azure Deployment:** Deploy the API to Azure and share the live link.
- **Logging:** Log all alert activities and external API calls for monitoring and debugging purposes.

Example:

Suppose you have the following data:

- **Regions:**

JSON ▾

 Copy  Caption ***

```
json
Copy code
[
  {
    "RegionID": "R1",
    "LocationCoordinates": {"latitude": 34.0522, "longitude": -118.2437},
    "DisasterTypes": ["flood", "earthquake"]
  },
  {
    "RegionID": "R2",
    "LocationCoordinates": {"latitude": 36.7783, "longitude": -119.4179},
    "DisasterTypes": ["wildfire"]
  }
]
```

Alert Settings:

json

Copy code

```
[
  {
    "RegionID": "R1",
    "DisasterType": "flood",
    "ThresholdScore": 75
  },
  {
    "RegionID": "R2",
    "DisasterType": "wildfire",
    "ThresholdScore": 80
  }
]
```

Expected Output for GET /api/disaster-risks:

```
json
Copy code
[
  {
    "RegionID": "R1",
    "DisasterType": "flood",
    "RiskScore": 82,
    "RiskLevel": "High",
    "AlertTriggered": true},
  {
    "RegionID": "R2",
    "DisasterType": "wildfire",
    "RiskScore": 65,
    "RiskLevel": "Medium",
    "AlertTriggered": false}
]
```

Challenge:

1. **Build the API:** Implement each endpoint with the specified functionality.
2. **External API Integration:** Set up integration with external environmental data sources for real-time data.
3. **Caching with Redis:** Use Redis to cache environmental data and risk calculations to minimize redundant external API calls.
4. **Azure Deployment:** Deploy the solution on Azure and provide a live demo.
5. **Error Handling:** Manage scenarios like:
 - Failed external API calls.
 - Missing data from external sources.
 - Regions with no available data.

6. **Messaging API Integration:** Implement alert-sending functionality via a messaging API to notify people in high-risk regions.
7. **Logging:** Track API usage and alerts for auditing purposes.

This problem emphasizes real-time data processing, API design, integration with external data sources, and cache management, as well as deploying a scalable solution on Azure. Good luck!

การส่งแบบทดสอบ

1.ส่งวิดีโอพร้อมอธิบายการทำงานของโจทย์ที่ได้รับ

2.ส่ง Source code แบบทดสอบ