Smart Disaster Resource Coordination Platform

Phase 7: Integration & External Access

Named Credentials Configuration

Secure Integration Setup:

Weather Service Integration

Named Credential: WeatherGov_API **Purpose:** Secure connection to National Weather Service API for real-time weather alerts and disaster prediction data.

Use Case: Emergency response operations require current weather conditions and forecast data to make informed decisions about resource deployment, evacuation planning, and shelter operations. This integration provides automated weather monitoring integrated with disaster declarations.

Configuration Details:

- **Endpoint URL:** https://api.weather.gov
- **Authentication:** API Key authentication with service account
- **Certificate:** Government-issued SSL certificate for secure communication
- **Timeout Settings:** 30-second timeout for reliable emergency operations
- **Retry Logic:** Automatic retry with exponential backoff for critical weather data

Security Implementation:

- **Credential Storage:** Encrypted storage of API credentials
- **Access Control:** Restricted access to authorized system processes only
- **Audit Trail:** Complete logging of all external API interactions
- **Compliance:** Meets government security standards for inter-agency data sharing

Government Emergency Management Integration

Named Credential: FEMA_Integration **Purpose:** Bidirectional integration with Federal Emergency Management Agency systems for disaster declaration coordination and resource sharing.

Business Value:

- **Disaster Declaration Sync:** Automatic synchronization of federal disaster declarations
- **Resource Coordination:** Cross-agency resource sharing and availability updates
- **Reporting Compliance:** Automated submission of required federal reports

- **Funding Documentation:** Accurate record keeping for federal reimbursement

External Services Implementation

Automated Weather Monitoring Service

Service Name: WeatherMonitoringService **Purpose:** Continuous monitoring of weather conditions with automatic disaster severity updates and alert generation.

Use Case: Disasters often escalate based on changing weather conditions. This service monitors weather patterns and automatically updates disaster severity levels, triggers additional resource allocation, and generates stakeholder alerts.

Service Functionality:

- **Continuous Monitoring:** Every 15-minute weather condition polling
- **Alert Processing:** Automatic parsing of weather service alerts
- **Severity Mapping:** Weather condition mapping to disaster severity levels
- **Automated Updates:** Direct disaster record updates based on weather changes

Integration Architecture:

apex

```
public class WeatherIntegrationService {

    @future(callout=true)

  public static void updateDisasterWeatherData(Id disasterId) {

    HttpRequest req = new HttpRequest();

    req.setEndpoint('callout:WeatherGov_API/alerts/active');

    req.setMethod('GET');

     Http http = new Http();

    HttpResponse res = http.send(req);

     if (res.getStatusCode() == 200) {

       processWeatherAlerts(disasterId, res.getBody());

    }

  }

}
```

Web Services (REST/SOAP) Implementation

RESTful Resource Sharing API

API Endpoint: /services/apexrest/emergency/resources/ **Purpose:** Provides external access to resource availability data for inter-agency coordination and mutual aid agreements.

Use Case: During large-scale disasters, multiple agencies and organizations need to coordinate resource sharing. This API enables automated resource availability sharing and cross-agency resource requests.

REST API Implementation:apex

```
@RestResource(urlMapping='/emergency/resources/*')

global with sharing class EmergencyResourceAPI { @HttpGet

   global static ResourceResponse getAvailableResources() {

      List<Resource__c> resources = [

         SELECT Resource_Name__c, Current_Stock_Level__c,

              Storage_Location__c, Resource_Type__c

         FROM Resource__c

         WHERE Current_Stock_Level__c > Minimum_Threshold__c

      ];  return new ResourceResponse(resources, 'success');

   }   @HttpPost

   global static String submitResourceRequest(ResourceRequest request){

      Request__c newRequest = new Request__c();

      newRequest.Requested_Resource__c = request.resourceId;

      newRequest.Quantity_Requested__c = request.quantity;

      newRequest.Priority_Level__c = request.priority;

      insert newRequest;

      return newRequest.Id;

   }

}
```

API Security:

- **OAuth 2.0 Authentication:** Secure API access with token-based authentication
- **Role-Based Access:** API permissions based on requesting organization's access level
- **Rate Limiting:** Request throttling to prevent system overload
- **Data Encryption:** TLS 1.2 encryption for all API communications

SOAP Integration for Legacy Systems

Service Name: EmergencyCoordinationSOAP **Purpose:** SOAP-based integration for legacy emergency management systems requiring XML-based communication protocols.

Use Case: Many government agencies operate legacy emergency management systems that require SOAP-based integration. This service provides backward compatibility while maintaining modern API capabilities.

SOAP Service Features:

- **XML Schema Compliance:** Strict adherence to government XML standards
- **WS-Security:** Enhanced security for government system integration
- **Transaction Management:** Reliable message delivery with confirmation
- **Legacy Protocol Support:** Support for older authentication methods

Callout Implementation

Multi-System Integration Architecture

Callout Strategy: Emergency response requires integration with multiple external systems including weather services, government databases, and partner organization APIs. The callout architecture supports simultaneous integration while maintaining system performance.

Implementation Pattern:

apex

```
public class ExternalSystemCallouts {
  @future(callout=true)
  public static void notifyPartnerAgencies(List<Id> disasterIds) {
    for (Id disasterId : disasterIds) {
      notifyWeatherService(disasterId);
      notifyGovernmentSystems(disasterId);
      notifyPartnerOrganizations(disasterId);
    }}
  private static void notifyWeatherService(Id disasterId) {
      HttpRequest req = new HttpRequest();
      req.setEndpoint('callout:WeatherGov_API/disasters');
      req.setMethod('POST');
      req.setBody(buildDisasterNotification(disasterId));
      Http http = new Http();
      HttpResponse res = http.send(req);
      processWeatherResponse(res);
  }}
```

Callout Benefits:

- **Asynchronous Processing:** Non-blocking external system communication
- **Error Handling:** Comprehensive error management with retry logic
- **Performance Optimization:** Efficient resource utilization through future methods
- **Monitoring:** Complete audit trail of external system interactions

Platform Events Implementation

Real-Time Coordination Events

Event Name: Emergency_Status_Update__e **Purpose:** Real-time notification system for critical emergency status changes across all integrated systems and stakeholders.

Use Case: During emergency response, status changes must be communicated instantly to all relevant personnel and systems. Platform events provide real-time notification capability with guaranteed delivery.

Event Structure:

apex

```
// Platform Event Definition
public class EmergencyStatusUpdate extends SObject {
   public String Event_Type__c;       // Disaster, Shelter, Resource, Volunteer
   public String Record_Id__c;        // Related record identifier
   public String Status_Change__c;    // Previous and current status
   public String Severity_Level__c;   // Emergency priority level
```

```
    public DateTime Event_Timestamp__c; // Precise event timing
    public String Affected_Areas__c;    // Geographic impact areas
}
```

Event Publishing:

apex

```
public static void publishEmergencyUpdate(String eventType, Id recordId, String statusChange) {
    Emergency_Status_Update__e event = new Emergency_Status_Update__e();
    event.Event_Type__c = eventType;
    event.Record_Id__c = recordId;
    event.Status_Change__c = statusChange;
    event.Event_Timestamp__c = System.now();

    EventBus.publish(event);
}
```

Event Subscription:

- **Real-Time Dashboard Updates:** Automatic dashboard refresh for status changes
- **Mobile Notifications:** Push notifications to field personnel mobile devices
- **External System Sync:** Automatic synchronization with partner organization systems
- **Audit Trail:** Complete event history for post-incident analysis

Change Data Capture Implementation

Critical Data Synchronization

CDC Object Configuration: Shelter__c, Resource__c, Volunteer__c **Purpose:** Automatic synchronization of critical data changes with external systems and backup databases for disaster recovery.

Use Case: Emergency response data must be continuously synchronized with backup systems and partner organizations to ensure data availability during system failures or communication interruptions.

CDC Processing:

apex

```
public class EmergencyDataSyncHandler {

    @future(callout=true)
    public static void processShelterChanges(List<ShelterChangeEvent> events) {
        for (ShelterChangeEvent event : events) {
            syncToBackupSystem(event.getEventUuid(), event.recordIds);
            notifyPartnerSystems(event.changedFields, event.recordIds);
        }
    }
}
```

Synchronization Benefits:

- **Data Resilience:** Continuous backup for disaster recovery scenarios
- **Partner Coordination:** Real-time data sharing with mutual aid partners
- **Compliance:** Audit trail maintenance for regulatory requirements
- **Performance:** Efficient change-based synchronization minimizing system load

Salesforce Connect Implementation

External Data Integration

External Data Source: Government_Emergency_Database **Purpose:** Real-time access to government emergency management data without data duplication, maintaining single source of truth for official disaster information.

Use Case: Emergency response requires access to official government disaster declarations, resource allocations, and coordination information without duplicating sensitive government data in local systems.

Connect Configuration:

- **Data Source Type:** OData 4.0 for government system compatibility
- **Authentication:** Certificate-based authentication for government security compliance
- **Sync Mode:** Real-time for critical data, scheduled for historical information
- **Field Mapping:** Government data field mapping to Salesforce object structure

External Object Benefits:

- **Data Currency:** Always current government information without manual updates
- **Security Compliance:** Government data remains in authorized systems
- **Integration Efficiency:** No ETL processes required for government data access
- **Scalability:** Unlimited government data access without storage constraints

API Limits Management

Governor Limit Optimization

API Usage Strategy: Emergency response operations require high-volume API usage for real-time coordination. The API management strategy optimizes usage while maintaining system reliability.

Optimization Techniques:

1. **Callout Pooling:**

apex

```
public class APICalloutManager {
   private static final Integer MAX_CONCURRENT_CALLS = 10;
   private static List<HttpRequest> calloutQueue = new List<HttpRequest>();

   public static void queueCallout(HttpRequest request) {
      calloutQueue.add(request);
      processQueue();
   }
   private static void processQueue() {
      // Process callouts in batches respecting limits
   }
}
```

2. **Caching Strategy:**
- **Static Resource Caching:** Cache frequently accessed reference data
- **Session Caching:** Store API responses for session duration
- **Platform Cache:** Long-term caching for stable external data

3. **Batch Processing:**
- **Scheduled Batch Jobs:** Process non-urgent integrations during off-peak hours
- **Queueable Chaining:** Chain related API calls for efficiency
- **Future Method Optimization:** Minimize future method usage for API calls

OAuth & Authentication Implementation

Secure Multi-System Authentication

Authentication Architecture: Emergency response requires secure authentication across multiple partner systems while maintaining rapid access during crisis situations.

OAuth 2.0 Implementation:

apex

```
public class OAuthManager {
   public static String getAccessToken(String namedCredential) {
        HttpRequest req = new HttpRequest();
        req.setEndpoint('callout:' + namedCredential + '/oauth/token');
        req.setMethod('POST');
        req.setHeader('Content-Type', 'application/x-www-form-urlencoded');
        req.setBody('grant_type=client_credentials');
        Http http = new Http();
        HttpResponse res = http.send(req);
        Map<String, Object> tokenResponse = (Map<String, Object>)
          JSON.deserializeUntyped(res.getBody());
              return (String) tokenResponse.get('access_token');
    }

     }
```

Security Features:

- **Token Refresh:** Automatic token renewal before expiration
- **Secure Storage:** Encrypted token storage in named credentials
- **Access Control:** Role-based API access permissions
- **Audit Logging:** Complete authentication event tracking

Remote Site Settings Configuration

External System Access Control

Remote Site Configurations:

1. **Weather Service Access**
    - **Remote Site URL:** https://api.weather.gov
    - **Active:** Enabled for continuous weather monitoring
    - **Description:** National Weather Service API access for disaster forecasting
2. **Government Integration**
    - **Remote Site URL:** https://api.fema.gov
    - **Active:** Enabled for federal coordination
    - **Description:** FEMA system integration for disaster coordination
3. **Partner Organization APIs**
    - **Remote Site URL:** https://api.redcross.org
    - **Active:** Enabled for mutual aid coordination
    - **Description:** American Red Cross resource sharing integration