**1.Installing Required LAMP Stack**

sudo apt update

sudo apt install apache2 mysql-server php libapache2-mod-php php-mysql

sudo systemctl start apache2 # Start Apache

sudo systemctl enable apache2 # Enable Apache on boot

sudo systemctl start mysql # Start MySQL

sudo systemctl enable mysql # Enable MySQL on boot

sudo systemctl stop apache2 # stop Apache

sudo systemctl stop mysql # stop MySQL

sudo systemctl status apache2

sudo systemctl status mysql

**2.Configuring php apache mysql**

## ****Step 2: Verify Installations****

### ****Check Apache****

Open a web browser and enter:

http://localhost

If Apache is running, you’ll see the **Apache default page**.

### ****Check PHP****

Create a test PHP file:

sudo nano /var/www/html/info.php

Add the following:

<?php

phpinfo();

?>

Save and exit (CTRL+X, then Y, then Enter).

Now, open in browser:

http://localhost/info.php

If PHP is working, you will see PHP configuration details.

**3.Creating Databases**

**To fetch disaster data from APIs (NASA API) and store it in the MySQL database, follow these steps:**

### ****Step 1: Get API Key for NASA & Google Maps****

#### ****1. Get a NASA API Key****

NASA provides various APIs, such as Earth imagery, asteroid tracking, and space weather. To get an API key:

1. Go to [**NASA API Portal**](https://api.nasa.gov/)
2. Scroll down to the "Get Your API Key" section.
3. Fill in your name and email address.
4. Click **"Generate API Key"**.
5. Copy and store the API key securely.

#### ****2. Get a Google Maps API Key****

Google Maps API is needed to display maps and location-based data. Follow these steps to get an API key:

1. Go to **https://console.cloud.google.com/**
2. Sign in with your Google account.
3. Create a **new project** (or select an existing one).
4. Navigate to **APIs & Services > Library**
5. Search for **"Maps JavaScript API"**, **"Geocoding API"**, and **"Places API"**, then enable them.
6. Go to **APIs & Services > Credentials**
7. Click **"Create Credentials" > API Key**
8. Copy the generated **API key** and restrict it for security (e.g., limit it to your domain).

## ****Step 2: Create PHP Script to Fetch NASA Disaster Data****

NASA provides disaster-related data via its **EONET API** (https://eonet.gsfc.nasa.gov/api/v3/events).

Create a PHP script fetch\_nasa\_data.php to fetch and store data.

## ****Step 3: Automate Fetching NASA Data****

You can schedule the script to run automatically using **Cron Jobs** (Linux) or **Task Scheduler** (Windows).

### ****Linux Cron Job****

Run every **6 hours**:

crontab -e

Add:

0 \*/6 \* \* \* /usr/bin/php /var/www/html/fetch\_nasa\_data.php

## ****Step 4: Fetch Google Maps Data****

Google Maps API is used to **visualize disaster locations**. In index.html, add:

**Step 5: Fetch Data from Database (get\_disasters.php)**

CREATE DATABASE IF NOT EXISTS disaster\_management;

USE disaster\_management;

-- Disaster Alerts Table (Storing NASA EONET API Data)

CREATE TABLE disaster\_alerts (

id INT AUTO\_INCREMENT PRIMARY KEY,

eonet\_id VARCHAR(50) UNIQUE NOT NULL,

disaster\_type VARCHAR(50) NOT NULL,

category VARCHAR(100) NOT NULL, -- NEW: Disaster Category (e.g., Wildfire, Storm)

severity ENUM('Low', 'Moderate', 'Severe', 'Extreme') DEFAULT 'Moderate',

timestamp DATETIME DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

latitude DECIMAL(10, 6) NOT NULL,

longitude DECIMAL(10, 6) NOT NULL,

region VARCHAR(255) NOT NULL, -- NEW: Affected region for filtering

description TEXT NOT NULL,

source\_url VARCHAR(255),

closed BOOLEAN DEFAULT FALSE

);

-- Indexes for Performance

CREATE INDEX idx\_disaster\_type ON disaster\_alerts(disaster\_type);

CREATE INDEX idx\_category ON disaster\_alerts(category); -- NEW

CREATE INDEX idx\_timestamp ON disaster\_alerts(timestamp);

CREATE INDEX idx\_region ON disaster\_alerts(region); -- NEW

-- Relief Measures Table

CREATE TABLE relief\_measures (

id INT AUTO\_INCREMENT PRIMARY KEY,

disaster\_id INT,

organization\_name VARCHAR(100) NOT NULL,

contact\_person VARCHAR(100),

contact\_email VARCHAR(100),

contact\_phone VARCHAR(20),

relief\_description TEXT,

FOREIGN KEY (disaster\_id) REFERENCES disaster\_alerts(id) ON DELETE CASCADE

);

-- Users Table (Storing User Preferences)

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

phone VARCHAR(20) UNIQUE,

location VARCHAR(255)

);

-- User Preferences Table (Merged with Subscriptions)

CREATE TABLE user\_preferences (

id INT AUTO\_INCREMENT PRIMARY KEY,

user\_id INT,

notification\_type ENUM('Email', 'SMS', 'Push') NOT NULL,

preferred\_disaster\_type VARCHAR(100), -- NEW: Allows users to select disasters they care about

preferred\_region VARCHAR(255), -- NEW: Users can set their preferred region

FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE CASCADE

);

-- User Notifications Table

CREATE TABLE user\_notifications (

id INT AUTO\_INCREMENT PRIMARY KEY,

user\_id INT,

disaster\_id INT,

notification\_type ENUM('Email', 'SMS', 'Push') NOT NULL,

sent\_at DATETIME DEFAULT CURRENT\_TIMESTAMP,

status ENUM('Sent', 'Failed') DEFAULT 'Sent',

FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE CASCADE,

FOREIGN KEY (disaster\_id) REFERENCES disaster\_alerts(id) ON DELETE CASCADE

);

--GDACS

CREATE TABLE gdacs\_disasters (

id INT AUTO\_INCREMENT PRIMARY KEY,

event\_id VARCHAR(50) UNIQUE,

event\_type VARCHAR(50),

alert\_level VARCHAR(20),

severity FLOAT,

depth FLOAT,

latitude FLOAT,

longitude FLOAT,

country VARCHAR(100),

timestamp DATETIME,

source\_url VARCHAR(255),

image\_url VARCHAR(255)

);

https://eonet.gsfc.nasa.gov/docs/v3

JAVA import com.fasterxml.jackson.annotation.JsonInclude;

import com.fasterxml.jackson.annotation.JsonProperty;

import com.fasterxml.jackson.databind.ObjectMapper;

import java.util.List;

@JsonInclude(JsonInclude.Include.NON\_NULL)

class GeoJsonResponse {

@JsonProperty("type")

private String type;

@JsonProperty("features")

private List<Feature> features;

// Getters and Setters

}

@JsonInclude(JsonInclude.Include.NON\_NULL)

class Feature {

@JsonProperty("type")

private String type;

@JsonProperty("properties")

private Properties properties;

@JsonProperty("geometry")

private Geometry geometry;

// Getters and Setters

}

@JsonInclude(JsonInclude.Include.NON\_NULL)

class Properties {

@JsonProperty("id")

private String id;

@JsonProperty("title")

private String title;

@JsonProperty("description")

private String description;

@JsonProperty("link")

private String link;

@JsonProperty("closed")

private String closed;

@JsonProperty("date")

private String date;

@JsonProperty("magnitudeValue")

private Double magnitudeValue;

@JsonProperty("magnitudeUnit")

private String magnitudeUnit;

@JsonProperty("categories")

private List<Category> categories;

@JsonProperty("sources")

private List<Source> sources;

@JsonProperty("geometryDates")

private List<String> geometryDates;

// Getters and Setters

}

@JsonInclude(JsonInclude.Include.NON\_NULL)

class Category {

@JsonProperty("id")

private String id;

@JsonProperty("title")

private String title;

// Getters and Setters

}

@JsonInclude(JsonInclude.Include.NON\_NULL)

class Source {

@JsonProperty("id")

private String id;

@JsonProperty("url")

private String url;

// Getters and Setters

}

@JsonInclude(JsonInclude.Include.NON\_NULL)

class Geometry {

@JsonProperty("type")

private String type;

@JsonProperty("coordinates")

private Object coordinates; // Can be List<Double> (Point) or List<List<Double>> (LineString)

// Getters and Setters

}