Flood Monitoring System: Key Components, APIs, and Structure

# 1. AWS Package Module

To handle the system on AWS, here are the services you’ll need:

* **Amazon EC2 (Elastic Compute Cloud)**: Host your Django application and APIs.
* **Amazon RDS (Relational Database Service)**: You can use PostgreSQL or MySQL to manage structured data. AWS RDS handles backups, scaling, and replication.
* **Amazon S3 (Simple Storage Service**): For storing large datasets (water level logs, reports, GIS data, etc.) that need to be easily accessible and scalable.
* **Amazon SNS (Simple Notification Service)**: For SMS and email notifications to authorities when water levels exceed thresholds.
* **Amazon Lambda**: For running serverless functions triggered by certain events, such as analyzing the water level and sending notifications.
* **AWS IoT Core**: For handling real-time data from rigs (IoT devices) measuring the river’s water levels.
* **AWS CloudWatch**: For monitoring the performance of your application and services, and setting up alarms if anomalies in system performance occur.
* **Amazon Polly**: To generate spoken alerts for authorities (optional).
* AWS API Gateway: To manage and scale the API connections between rigs, the server, and the Django backend.

Estimated pricing will depend on the usage:  
- **EC2:** Costs vary based on instance type, region and usage e.g. a t4g.micro instance in the US East costs $0.0084 per hour which is $6.05 per month  
- **RDS:** Depends on DB engine, instance type and region e.g. in the US East region, an RDS db.t4g.micro instance for MySQL costs $0.017/hr which is $12.24/month  
- **S3:** S3 pricing starts at $0.023 per GB for standard storage in the US East region, with reduced prices for higher volumes and cheaper storage classes like S3 Glacier.  
- **SNS:** $0.50 per 1 million published messages. Additional charge of $0.00645/ SMS.

**- **Amazon Lambda**: P**ricing is based on the number of requests and the compute time. The first 1 million requests are free each month, and after that, it costs $0.20 per million requests. Additional charges apply for compute time, at $0.00001667 per GB-second  
- **IoT Core**: Pricing based on messages exchanged, IoT Core charges per million messages. For example, in the US East region, it costs $1 per million messages transmitted, but you can optimize depending on exact resource usage.

- **AWS CloudWatch:** Charges depend on metrics and logs. Custom metrics costs $0.30 per metric & log data ingestion starts at $0.50/GB for up to 5GB/month.

- **Amazon Polly:** charges are based on number of characters processed. Cost is $4/1 million characters for standard voices.

- **Amazon API Gateway:** For REST API pricing starts at $3.50 per million API calls. Data transfer and caching can add to the cost depending on usage.

The prices can vary based on region and usage patterns.

Rough monthly bill for starter could be: $81.65 or KES. (This is based on a rough calculation, it’s not accurate meaning the price may be higher or lower than $ 81.65), planning with $100per month won’t hurt.

# 2. Modules and API Structure (Django)

Here’s how the system’s architecture can be broken down:

## Rigs API

The rigs send water level data every few minutes via an API endpoint. A Django REST API handles this, authenticating the rig, accepting the data, and storing it.

Fields: timestamp, water\_level, location\_id, rig\_id, rainfall\_data.

## Prediction Module

Use machine learning libraries in Django (such as scikit-learn for linear regression) to model future water levels based on past data. The model runs on incoming data and forecasts trends.

API: GET /api/predictions/

Fields: location\_id, predicted\_level, time.

## Alert Module

An alerting system that checks if water levels exceed thresholds and triggers AWS SNS for SMS alerts. This can be implemented in Django as a background task using Celery.

API: GET /api/alerts/

Fields: alert\_id, location\_id, status, sms\_sent.

## GIS Module

Integration with a GIS mapping system to visualize water levels and predicted floods. You can use Django to query GIS data from Redis and render the results via a mapping library like Leaflet.

API: GET /api/map/

Fields: location\_id, water\_level, depth, flood\_zone.

## Subscription Management

Users subscribe to flood alerts for specific rivers/regions. You can integrate a Django subscription app (e.g., django-payments or django-stripe for MPESA and banks). Banks will mostly be preferred if the system is intended for global use, else Mpesa will just do fine. I’ll go with the Bank.

API: POST /api/subscription/

Fields: user\_id, location\_id, plan, payment\_method.

## User Authentication

Django’s built-in django-allauth can handle user registration, login, and role-based access for different users (e.g., local authorities).

API: POST /api/login/

API: POST /api/register/

Fields: username, email, password.

These are the main modules, more can be added as per the system and users’ needs.

# 3. System Structure (APIs and Functionalities)

You will need the following key functionalities, grouped by the system features:

## Database:

* MongoDB: For storing raw water level data and real-time logs.
* PostgreSQL or MySQL: To store user information, subscriptions, and processed data.
* Redis (GIS): To store geospatial data for rapid access and querying.

## Flood Monitoring:

API from the rig for data input.

Data processing using linear regression or other algorithms.

Predictions and visualizations tied into the GIS module.

## Payment:

Integration with MPESA and banks (Equity, Coop, KCB) for user subscriptions. Use APIs provided by each payment service.

## SMS:

* AfricasTalking for SMS alerts to authorities and subscribers.
* AWS SNS - Notification service

## GIS:

Mapping water levels using Redis for geospatial queries.

Display real-time updates on a web interface with Django’s template engine or a front end framework like React.

## Analysis:

Analyze trends in water levels and rainfall patterns to improve prediction accuracy.

## Prediction:

Predict future water levels based on real-time and historical data. This can involve machine learning algorithms using libraries like TensorFlow or scikit-learn.

## Invoicing:

Generate and send invoices for subscription payments using Django’s template rendering system and automated emails.

## Subscription Management:

A system where users subscribe to water level alerts and manage their preferences.

# Suggested Tools and Libraries:

* Django REST Framework: For building APIs.
* Celery: For background task management (e.g., sending alerts), **Mapping** and imagery
* Redis: For caching GIS queries and rapid data access.
* Leaflet or Mapbox: For displaying GIS maps.
* AfricasTalking API: For SMS notifications. -
* MPESA/Banks API: For payment integration
* PostgreSQL/MySQL/MongoDB: For storage purposes

**API - Successful**

- User Authentication - Fully done 100%

- Monitoring - (Water levels, Humidity, Temp) - 20%

- Inclusion of system Monitoring

- Payments API - 0% -( Banking ) - ABSA, Equity.

- Prediction API - (Linear regression - 20% )

- KMD Dataset ( 0% )

- SMS - 70%

- Subscription API - 50 %

- Models - User dashboards ( 20% )

- Mapping (GIS)

-