**Project Documentation: Sentiment Analysis Web Application with Flask, React, and AWS S3**

This documentation provides a step-by-step breakdown of how we developed a **Sentiment Analysis Web Application** using **Flask** as the backend, **React** for the frontend, and **AWS S3** for storage. The system processes user input text, analyzes the sentiment using the TextBlob library, and stores the results in an AWS S3 bucket. The following sections explain how we set up the environment, deployed the application, and developed the necessary components.

**1. Setting up AWS EC2 Instance**

To deploy the Flask API and make it accessible over the web, we used **AWS EC2** (Elastic Compute Cloud).

**Steps:**

1. **Create an EC2 instance**:
   * Log in to your [AWS Console](https://aws.amazon.com/console/).
   * Navigate to **EC2** and click on "Launch Instance".
   * Choose an AMI (Amazon Machine Image) such as **Ubuntu 20.04**.
   * Select an instance type (e.g., **t2.micro** for free tier).
   * Configure instance settings, add storage, and configure security group to allow SSH and HTTP (ports 22 and 80).
   * Launch the instance and download the **key pair** for SSH access.
2. **Connect to EC2 instance via SSH**:
   * Use the downloaded **key pair** and your instance's **public IP** to SSH into the instance:
   * ssh -i your-key.pem ubuntu@your-ec2-public-ip
3. **Update and install dependencies**:
   * Update the package list:
   * sudo apt-get update
   * Install Python, pip, and other required packages:
   * sudo apt-get install python3-pip python3-dev
4. **Install Flask and other dependencies**:
   * Install Flask and TextBlob:
   * pip3 install flask textblob flask-cors boto3
5. **Verify Flask installation**:
   * Create a simple Flask application to test:
   * from flask import Flask
   * app = Flask(\_\_name\_\_)
   * @app.route('/')
   * def home():
   * return "Hello, World!"
   * if \_\_name\_\_ == '\_\_main\_\_':
   * app.run(host='0.0.0.0', port=5000)
   * Run the app:
   * python3 app.py
6. **Open port 5000 on EC2 instance**:
   * In your AWS security group, open port **5000** for HTTP traffic to access the Flask API.

**2. Setting up AWS S3 for Storage**

For storing sentiment analysis results, we used **Amazon S3** (Simple Storage Service).

**Steps:**

1. **Create an S3 bucket**:
   * Log in to AWS and navigate to **S3**.
   * Click on “Create bucket” and choose a globally unique name for your bucket, e.g., sentiment-analysis-results-harshi.
   * Configure other options and create the bucket.
2. **Configure AWS credentials on the EC2 instance**:
   * On the EC2 instance, configure **AWS CLI** to access S3:
   * aws configure
   * Enter your **AWS Access Key ID**, **Secret Access Key**, **Region**, and **output format**.

**3. Developing the Flask Backend (API)**

We developed the **Flask API** to perform sentiment analysis and store results in the S3 bucket.

**Steps:**

1. **Install dependencies**: As mentioned above, we installed **Flask**, **TextBlob**, and **boto3**.
   * Flask was used to create the REST API.
   * TextBlob was used to analyze the sentiment of input text.
   * boto3 is the AWS SDK used to interact with S3.
2. **Flask Routes**:
   * We defined two main API endpoints:
     + /sentiment: This route takes the text input and returns the sentiment (Positive, Negative, or Neutral) based on the polarity score.
     + /analyze: This route processes the text, analyzes sentiment, stores the result in the S3 bucket, and returns the result as a JSON response.
3. **Sentiment analysis logic**:
   * We used **TextBlob**'s sentiment.polarity to calculate the polarity score of the input text.
   * Based on the score, the sentiment is categorized as **Positive**, **Negative**, or **Neutral**.
4. **Storing results in S3**:
   * Using the **boto3** library, we saved the sentiment results (text, sentiment label, polarity score, and timestamp) as a JSON file in the S3 bucket.
5. **Run Flask API**:
   * The Flask app is run on **port 5000** to make it accessible via HTTP.

**4. Developing the React Frontend**

The frontend was built using **React** to provide a user-friendly interface for text input and displaying sentiment results.

**Steps:**

1. **Create a new React app**:
   * We used **Create React App** to initialize the frontend:
   * npx create-react-app sentiment-analysis-app
2. **Modify App.js**:
   * We created a simple form that allows the user to input text.
   * Upon form submission, the text is sent to the Flask backend using a **POST request** to the /sentiment API endpoint.
3. **Display Results**:
   * Once the response from the API is received, the sentiment and polarity score are displayed on the UI.
4. **Style the App**:
   * We styled the frontend with **CSS** for better user experience, including a clean layout with a title, input box, and button.

**5. Connecting Flask and React**

We connected the **React frontend** to the **Flask backend** using a **fetch API**.

**Steps:**

1. **CORS (Cross-Origin Resource Sharing)**:
   * We used the flask-cors package on the backend to enable cross-origin requests from the React frontend.
2. **Communicating with Flask**:
   * In the React app, we made an API call to the Flask backend using the **fetch** method to send the user input.
   * The backend processes the input, performs sentiment analysis, and returns the results.

**6. Testing and Deployment**

1. **Run Flask and React**:
   * To test the application locally:
     + Run the Flask API on the EC2 instance: python3 app.py.
     + Start the React development server locally using: npm start.
2. **Deploy the React app**:
   * We deployed the React app on a public domain (e.g., via **Netlify** or **Vercel**), but for testing purposes, you can run it locally.
3. **Test the flow**:
   * Open the React app in the browser and input some text for sentiment analysis.
   * Verify that the backend processes the request and sends back the sentiment results.

**7. Additional Notes**

* **S3 Bucket Permissions**:
  + Ensure your EC2 instance's IAM role has sufficient permissions to upload files to the S3 bucket.
  + Use the following permissions for the IAM role:
    - s3:PutObject
    - s3:ListBucket
* **Flask Security**:
  + For production, make sure you implement **authentication and authorization** for secure API access, such as using **API keys** or **OAuth**.
* **Error Handling**:
  + We included error handling on both the frontend (to handle API request failures) and the backend (to handle unexpected issues with S3 uploads or sentiment analysis).

**8. Final Remarks**

In this project, we integrated several technologies to build a fully functional sentiment analysis system. We used **AWS EC2** for hosting the Flask API, **AWS S3** for storing results, **Flask** for the backend API, and **React** for the frontend. This setup is scalable and can be expanded to include additional features such as user authentication, detailed analytics, and more advanced sentiment models.

Let me know if you need more specific details or if you'd like assistance with deploying the app!