For Deployment based on pure AWS, I will follow the process as mentioned below:

a) Database selection and setting security groups:

1. Use Amazon Relational Database Service (RDS) to create a managed PostgreSQL database instance.
2. Set up appropriate security groups, network access control lists, and VPC settings to secure the database.
3. Create tables to store weather data records and calculation results in the PostgreSQL database using DDL statements or ORM tools.

b) Ingestion:

- Use AWS Lambda and AWS CloudWatch Events to create a scheduled job to ingest data from the raw text files into the PostgreSQL database at regular intervals.
- The ingestion function should read the raw data files, check for duplicates, and insert new records into the database.
- The function should also log the start and end times and the number of records ingested.

c) Data Analysis:

- Use AWS Lambda to perform data analysis and store the calculated results in the PostgreSQL database.
- The Lambda function should query the raw data from the database and calculate the required statistics.
- The function should store the results in a new table in the PostgreSQL database.

d) REST API:

- Use AWS Elastic Beanstalk to deploy a Flask or Django REST API application that provides the two endpoints (/api/weather and /api/weather/stats) to retrieve the data from the PostgreSQL database.
- Use AWS Application Load Balancer to distribute the traffic to multiple instances of the API application and provide secure HTTPS connections.
- Use AWS Route 53 to manage the domain name and route traffic to the load balancer.

e) Documentation:

- Use AWS API Gateway to create a REST API that provides the Swagger/OpenAPI documentation for the endpoints.
- Use AWS Lambda and AWS Integration to connect the API Gateway to the Flask or Django application.

f) Deployment:

- Use AWS CloudFormation to create a template that includes all the required AWS resources for the application and API deployment.

- Use AWS CodePipeline to automate the deployment process, including building the code, running tests, and deploying the application and API to the AWS environment.

g) Secure the API: Secure the API using Amazon Cognito to authenticate users and control access to the API resources.

h) Monitor and scale: Use Amazon CloudWatch to monitor API and database performance and receive alerts in case of issues. Scale application horizontally or vertically as needed to handle increases in traffic.

i) Backup and restore: Set up automated backups of database using AWS Backup to ensure that data is protected and can be restored in case of data loss.