High-Level Design Document for Weather Application

*1. Introduction*

This document provides a high-level design of the Weather Application, which consists of a backend service built with Spring Boot and a frontend interface built with React. The application is designed to be scalable, reliable, and easy to deploy using modern cloud technologies.

*2. System Overview*

The Weather Application allows users to enter a city name and retrieve the current weather information for that city. The system is composed of two main components:

1. Backend Service: A Spring Boot application that handles API requests and interacts with external weather services to fetch weather data.
2. Frontend Interface: A React application that provides a user-friendly interface for entering city names and displaying weather information.

*3. Architecture Diagram*

Below is a high-level architecture diagram of the Weather Application:

User --> React Frontend --> Spring Boot Backend --> External Weather API

*4. Components Description*

1. User: The end-user who interacts with the application through a web browser.
2. React Frontend: The user interface of the application, built with React. It allows users to input city names and displays the weather information retrieved from the backend.
3. Spring Boot Backend: The server-side component of the application, built with Spring Boot. It processes requests from the frontend, fetches weather data from an external API, and returns the data to the frontend.
4. External Weather API: A third-party service that provides weather data. The backend interacts with this API to fetch the current weather information for the requested city.

*5. Deployment Architecture*

The application is deployed using Docker containers and Kubernetes for orchestration. This ensures scalability, reliability, and ease of management.

1. Docker: Both the frontend and backend applications are containerized using Docker. This allows for consistent environments across development, testing, and production.
2. Kubernetes: The containerized applications are deployed on a Kubernetes cluster, which manages the deployment, scaling, and operation of the application containers.

*6. Deployment Diagram*

Below is a high-level deployment diagram of the Weather Application on Azure Kubernetes Service (AKS):

User --> Azure Load Balancer --> React Pod --> Spring Boot Pod --> External Weather API

*7. Key Benefits*

1. Scalability: The use of Kubernetes allows the application to scale horizontally, handling increased load by adding more instances of the frontend and backend services.
2. Reliability: Kubernetes ensures high availability by automatically managing the health of the application instances and restarting them if they fail.
3. Ease of Deployment: Docker and Kubernetes simplify the deployment process, making it easy to deploy new versions of the application with minimal downtime.
4. User-Friendly Interface: The React frontend provides a responsive and intuitive interface for users to interact with the application.

*8. Conclusion*

The Weather Application is designed to provide a seamless and reliable experience for users seeking weather information. By leveraging modern technologies such as Docker and Kubernetes, the application ensures scalability, reliability, and ease of deployment, making it well-suited for production environments.