Karbos - Software Requirements Specification (SRS)

Project Name: Karbos

Version: 1.0

Date: 03-12-2025

Document Status: Final

1. Introduction

1.1 Purpose

The purpose of this document is to define the functional and non-functional requirements for **Karbos**, a Carbon-Aware Distributed Workload Orchestrator. This system is designed to minimize the carbon footprint of software operations by intelligently scheduling computing tasks during periods of low carbon intensity in the power grid.

1.2 Scope

Karbos is a backend-heavy distributed system with a frontend dashboard.

- **In Scope:**
  - Ingestion of real-time carbon intensity data from external APIs.
  - A scheduling engine optimizing for carbon efficiency within user-defined deadlines.
  - A worker pool for executing containerized tasks (Docker).
  - A dashboard for monitoring job status and carbon savings.
- **Out of Scope:**
  - Hardware-level power management (DVFS).
  - Billing/Payment gateways for cloud resources.
  - Mobile application development.

1.3 Definitions and Acronyms

- **SLA (Service Level Agreement):** The deadline by which a job must complete.
- **Carbon Intensity (**gCO_2/kWh**):** Grams of Carbon Dioxide emitted per kilowatt-hour of electricity generated.
- **Job/Task:** A discrete unit of computation (e.g., a Docker container execution).
- **gRPC:** Google Remote Procedure Call.

2. Overall Description

2.1 Product Perspective

Karbos functions as a middleware layer between the user (or CI/CD pipeline) and the compute infrastructure. It replaces standard "immediate execution" queues with a "smart delay" mechanism.

2.2 System Architecture Diagram

The system is composed of four primary microservices.

1. **Gateway API:** Handles REST/gRPC requests.
2. **Scheduler Service:** Queries carbon data and calculates execution windows.
3. **Worker Service:** Pulls jobs from Redis and manages Docker lifecycles.
4. **Carbon Intelligence Service:** Adapter for external Grid APIs (e.g., ElectricityMaps).

2.3 User Characteristics

- **DevOps Engineers:** Will use the CLI/API to integrate Karbos into pipelines.
- **Sustainability Managers:** Will use the dashboard to generate ESG reports.

3. Specific Requirements

3.1 Functional Requirements (FR)

FR-1: Job Submission

- **FR-1.1:** The system shall accept job submissions via a REST API endpoint `POST /jobs`.
- **FR-1.2:** A job submission must include: Docker Image URI, Command/Args, and Deadline (Timestamp).
- **FR-1.3:** The system shall validate that `Deadline > (Current Time + Estimated Duration)`.

FR-2: Carbon-Aware Scheduling Algorithm

- **FR-2.1:** The Scheduler shall poll the Carbon Intelligence Service for a 24-hour forecast of the target region.
- **FR-2.2:** The system must implement a **Sliding Window Minimum** algorithm to identify the time window W of duration d (job duration) where the average carbon intensity is minimal.
- **FR-2.3:** If the calculated start time is in the future, the job shall be placed in a `DELAYED` queue.

FR-3: Job Execution

- **FR-3.1:** Worker nodes shall pull jobs from the `READY` queue.
- **FR-3.2:** Workers shall execute the specified Docker container.
- **FR-3.3:** Workers shall capture `stdout` / `stderr` logs and store them in the database.
- **FR-3.4:** Upon failure, the system shall retry the job up to 3 times before marking as `FAILED`.

FR-4: Reporting and Visualization

- **FR-4.1:** The Dashboard shall display a line chart comparing "Actual Execution Time Intensity" vs. "Immediate Execution Intensity."
- FR-4.2: The system shall calculate "Total Saved CO_2" using the formula:$$Saved = (Intensity_{now} - Intensity_{scheduled}) \times PowerUsage \times Duration$$

3.2 Non-Functional Requirements (NFR)

NFR-1: Performance & Scalability

- **NFR-1.1:** The Scheduler must process a job placement decision within **200ms**.
- **NFR-1.2:** The system must support **1,000 concurrent jobs** in the queue.
- **NFR-1.3:** The worker pool must be horizontally scalable (add more Go binaries to increase throughput).

NFR-2: Reliability

- **NFR-2.1: Circuit Breaker:** If the Carbon API is unreachable, the system must fallback to a "historical average" mode to ensure jobs are not blocked indefinitely.
- **NFR-2.2: Persistence:** All job states must be persisted in PostgreSQL. Redis is used only for ephemeral queuing; data loss in Redis should not result in lost jobs (Postgres acts as the source of truth).

NFR-3: Security

- **NFR-3.1:** All API requests must be authenticated via **API Keys** (stored in headers).
- **NFR-3.2:** Worker nodes must have restricted permissions; they cannot delete logs, only append.

4. System Interfaces

4.1 External Interface Requirements

- **Carbon Data Provider:** The system shall interface with **ElectricityMaps API** or **WattTime API** using HTTPS.
- **Container Engine:** The system shall interface with the **Docker Engine API** (via Unix Socket or TCP) to spawn containers.

4.2 Database Design (ER Diagram Description)

- **Entity: Job** (ID, UserID, Image, Status, ScheduledTime, CreatedAt, Deadline)
- **Entity: ExecutionLog** (ID, JobID, Output, ExitCode, Duration)
- **Entity: CarbonCache** (Region, Timestamp, IntensityValue)

5. Technology Stack Specifications

- **Programming Language (Backend):** Go (Golang) v1.25.5
- **Programming Language (Frontend):** TypeScript (Next.js 16)
- **Database:** PostgreSQL 18
- **Message Broker:** Redis 8.4
- **Containerization:** Docker
- **API Protocol:** REST (External), gRPC (Internal Microservices)

6. Assumptions and Dependencies

- **Assumption:** The host machine running the Worker Node has Docker installed and the daemon is running.
- **Dependency:** Accurate carbon forecasts depend on the availability of the third-party API.
- **Constraint:** The system currently assumes a homogeneous power usage profile for all containers (e.g., assuming average CPU usage) for the sake of calculation simplicity in Version 1.0.

7. Developer

- **Developed By:** *Sambit Mondal*
- **Date:** *03-12-2025*