

Basic Design Document (BDD) – Distributed Imaging System

1. Overview

This system is designed to process images in a distributed manner using three C++ applications (image-generator, feature-extractor, data-logger), RabbitMQ as a message broker, and PostgreSQL as a storage backend. The architecture ensures reliable and continuous image processing with support for large images.

Objectives:

- Continuous ingestion and delivery of images.
- Feature extraction using OpenCV (SIFT).
- Reliable storage of processed results in a database.
- Cross-platform support and Dockerized deployment.

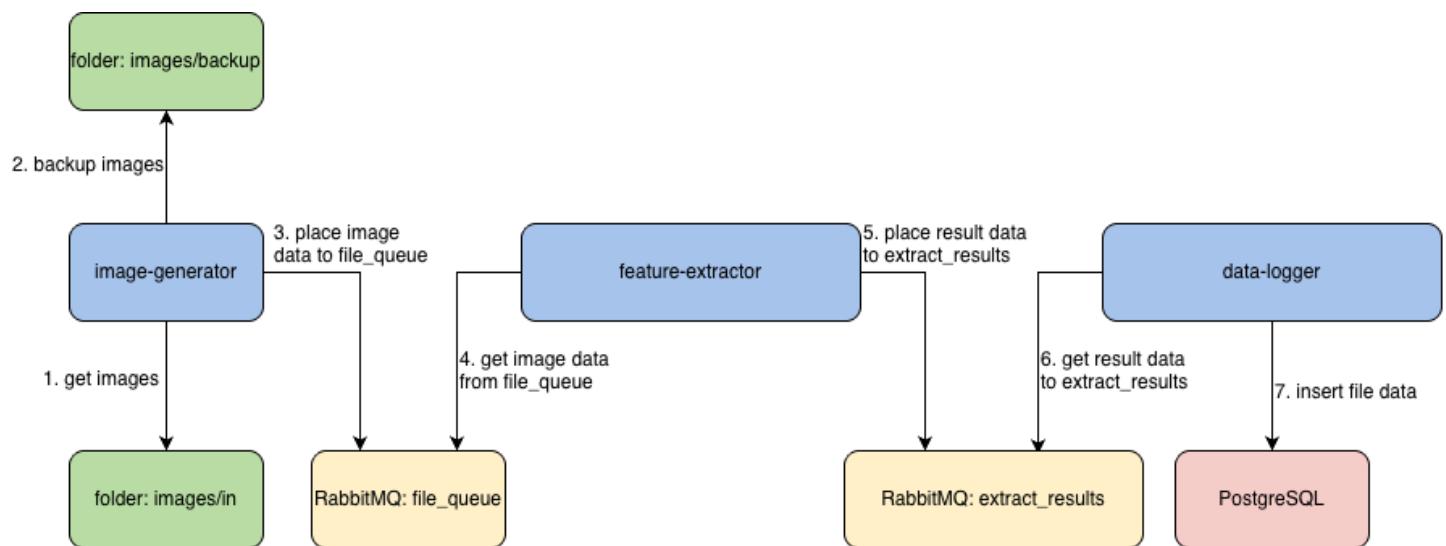
2. System Architecture

2.1 Components

- **image-generator** – Image Generator / Publisher
 - Reads images from `image/in` folder.
 - Adds date/time prefix and stores backups in `image/backup/YYYY/MM`.
 - Publishes images as binary data to RabbitMQ queue `file_queue`.
 - Deletes files after successful publishing.
- **feature-extractor** – Feature Extractor
 - Listens on RabbitMQ queue `file_queue`.
 - Receives image data and metadata (filename, backup path).
 - Uses OpenCV SIFT to extract keypoints.
 - Publishes JSON results containing image metadata and keypoints to RabbitMQ queue `extract_results`.
- **data-logger** – Data Logger
 - Listens on RabbitMQ queue `extract_results`.
 - Persists JSON data into PostgreSQL database (`voyis_main`) table files.
 - Supports large JSON payloads and concurrent inserts.
- **RabbitMQ**
 - Message broker for inter-process communication.
 - Queues:
 - `file_queue` → images from image-generator to feature-extractor.
 - `extract_results` → processed SIFT results from feature-extractor to data-logger.

- PostgreSQL
 - Database to store image metadata and SIFT results.
 - Table files schema:
 - CREATE TABLE files (
 - id SERIAL PRIMARY KEY,
 - filename VARCHAR(100) NOT NULL,
 - backup_path VARCHAR(255) NOT NULL,
 - json_data JSONB NOT NULL,
 - created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
 -);

2.2 High-Level Flow



2.3 Docker Deployment

- Base Image: my-base-image with OpenCV, Boost, SimpleAmqpClient, and build tools.
- Services (docker-compose):
 - rabbitmq
 - postgres
 - cpp-image-generator-app (image-generator)
 - cpp-feature-extractor-app (feature-extractor)
 - cpp-data-logger-app (data-logger)
- Volumes:
 - ./images:/images → share host image folder with image-generator
 - postgres_data:/var/lib/postgresql/data → persist DB

3. Design Decisions

- IPC Mechanism: RabbitMQ for reliable asynchronous communication.
- Backup Strategy: Images stored under backup/YYYY/MM with timestamped filenames.
- Continuous Loop: image-generator keeps sending images in a loop; supports images of varying sizes (KB → 30MB+).
- Error Handling:
 - feature-extractor skips corrupted or empty images.
 - data-logger validates JSON before DB insert.

4. Technology Stack

Component	Technology / Library
Programming	C++17
Image Processing	OpenCV (SIFT)
Messaging	RabbitMQ, SimpleAmqpClient
Database	PostgreSQL
JSON Handling	nlohmann/json
Build & Packaging	Docker, g++

5. Folder Structure

project-root/

 └── image-generator/

 └── Dockerfile

 └── main.cpp

 └── feature-extractor/

 └── Dockerfile

 └── main.cpp

 └── data-logger/

 └── Dockerfile

 └── main.cpp

 └── db/

 └── init.sql

 └── Dockerfile.base

 └── docker-compose.yml

 └── images/

 └── in/

 └── backup/

6. Future Enhancements

- Support multiple image formats dynamically.
- Add retry and dead-letter queue for failed messages.
- Add monitoring for queue sizes and processing latency.
- Optional GPU acceleration for feature extraction.
- Web interface for visualizing processed images and keypoints.

7. References

- OpenCV documentation: <https://docs.opencv.org>
- SimpleAmqpClient: <https://github.com/alanxz/SimpleAmqpClient>
- RabbitMQ: <https://www.rabbitmq.com/>
- PostgreSQL: <https://www.postgresql.org/>