IoT Data Pipeline Template

Team GR

- Gabriel Rodriguez (rodrigg@mail.uc.edu)
  - Student developer
- Peter Kroeger (peter14mail@gmail.com)
  - Project Advisor

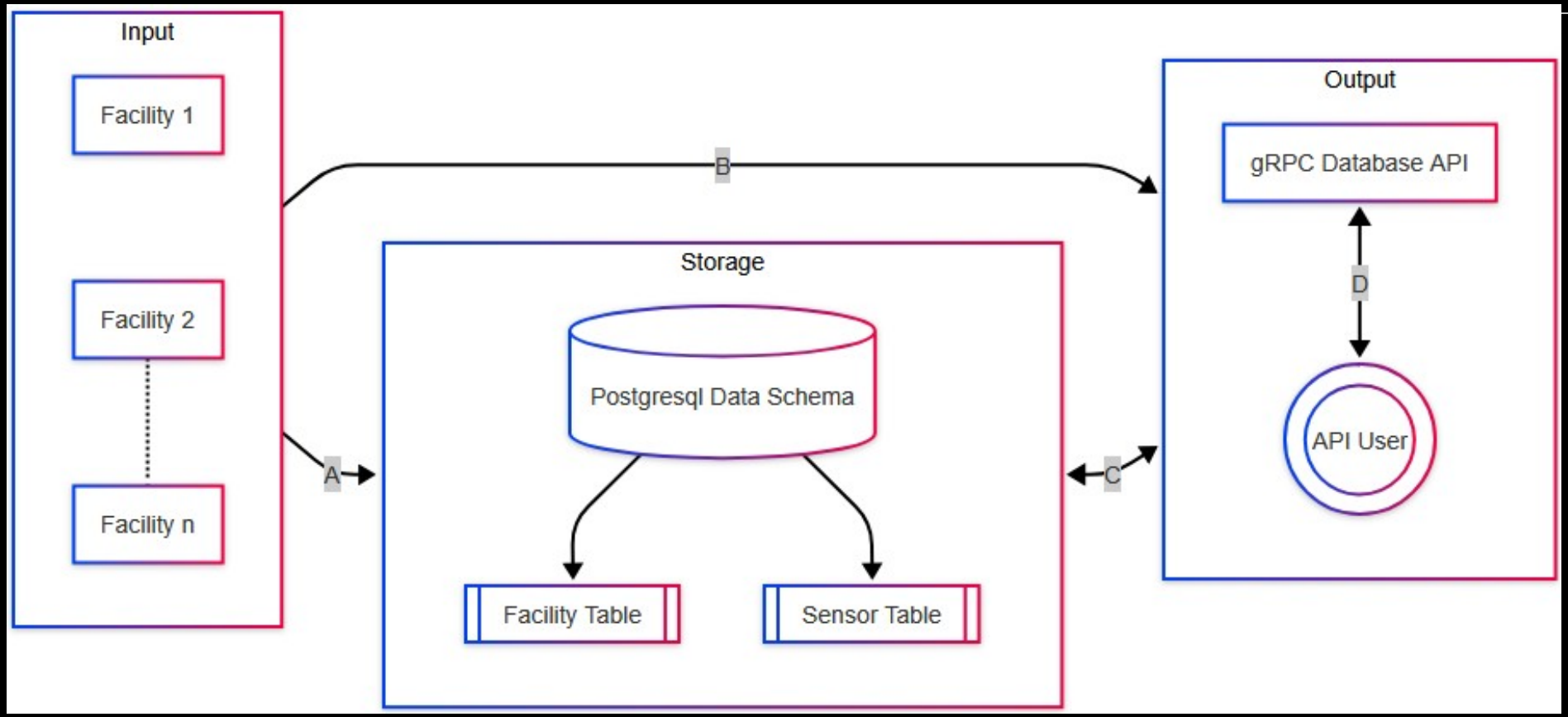
Project Description

The purpose of this project is to devise a data pipeline template that can used by other developers to create a data pipeline in house that meets their arbitrary requirements. For the purposes of this project, this data pipeline template will demonstrate how to collect atmospheric data from the interior of multiple facilities as an example exercise.

Intellectual Merits

- This project follows the microservice architectural pattern
  - This implies that the functionality of each service is limited to a few well defined tasks
  - Communication between each service uses Protobuf, which allows data to be quickly encoded/decoded in a compact binary format
  - Reproducible builds for each service are enforced by using multi-stage Docker images
- The languages used by this project were chosen to prioritize memory and concurrency safety, in addition to performance

Design Specification Overview

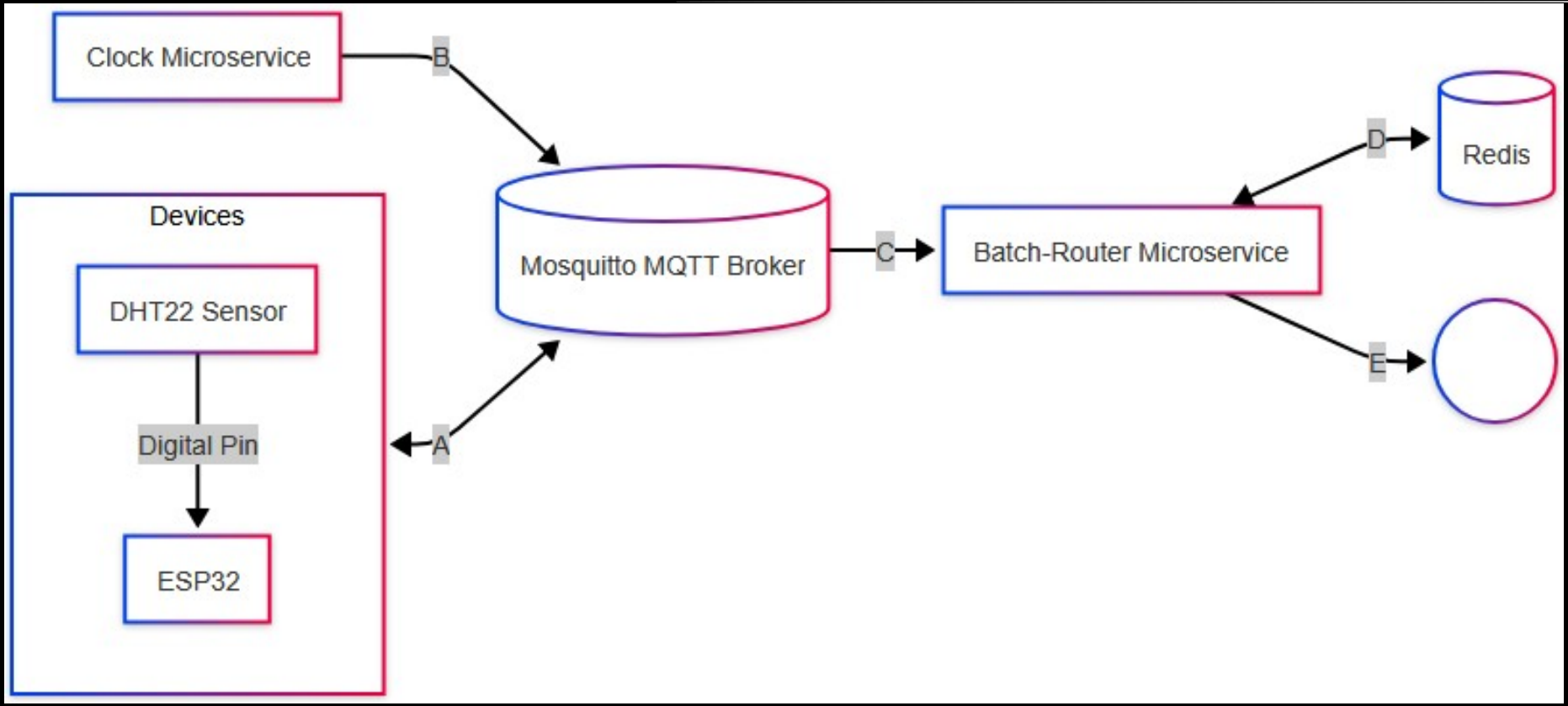


Database Table Specification

Facility		
SERIAL	facility_id	PK
VARCHAR(128)	facility_name	

Sensor			
SERIAL	entry_id	PK	
SERIAL	facility_id	PK,FK	
VARCHAR(64)	device		device ID, NOT NULL
REAL	temp		°C
REAl	rh		%
BIGINT	epoch		timestamp, NOT NULL

Facility-wide Design Specification



gRPC API Design Specification

Technology Used

- Each microservice is developed using Go, built using Docker, and executed via a Docker Compose service definition
- Firmware for the ESP32s was developed using Rust. This includes the DHT22 sensor driver
- PostgreSQL was used as this project’s database
- Communication is mainly handled by the project’s Mosquitto MQTT Broker
- Redis is used to store batches of data via Redis Queues. The connection state of each ESP32 is also tracked using Redis Keys

Results

- Current Progress:
  - An integrated system of microservices that is capable of batching, storing, and retrieving atmospheric data collect from a simulated IoT device
  - Firmware written in Rust for the ESP32 that is capable of reporting the temperature and relative humidity of a given room using a DHT22 sensor
- Next Steps:
  - Deployment to physically owned hardware
  - Deployment to the cloud, most likely Google Cloud Platform
  - Automated unit tests and software verification using formal methods tooling

Challenges

- Balancing program simplicity with performance and memory overhead for each microservice
- Prior to using Go, each microservice was implemented in C++. This made ensuring memory and concurrency safety difficult
  - Building each microservice and its dependencies in C++ was also a time consuming and complicated task when compared to Go’s build system
  - The execution environment of each microservice also needed to have the exact same libc and libstdc++ versions. This is why multi-staged Docker images were originally introduced to the project