Concurrent Task Processor

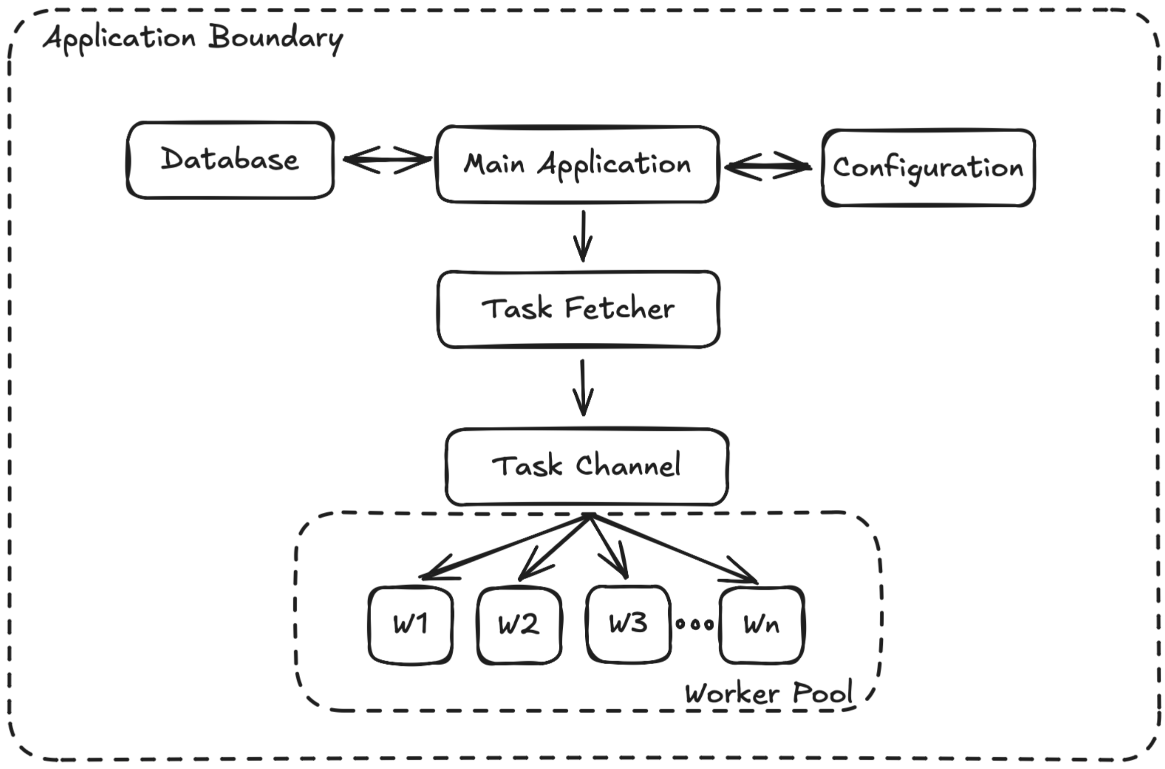
This project is a command-line application written in Go that demonstrates concurrent task processing using a worker pool pattern. The application reads tasks from a PostgreSQL database, processes them in parallel, and updates their status upon completion. It also implements graceful shutdown to ensure all in-progress tasks complete before the application exits.

Architecture

The application follows a modular architecture with the following components:

Core Components

- **Main Application**: Entry point that handles CLI commands
- **Worker Pool**: Manages concurrent task processing
- **Database Layer**: Handles interactions with PostgreSQL
- **Configuration Manager**: Loads settings from environment variables



Workflow

1. Load configuration from **.env** and connect to **PostgreSQL** database.
2. Set env variables: **WORKER_COUNT**=5, **FETCH_INTERVAL**=5s, **FETCH_LIMIT**=10 (default values).
3. Use **migrate** subcommand to run DB migration and create a table (**id**, **payload**, **status**, **result**, **created_at**, **updated_at**).
4. Run seed SQL to insert 20 sample tasks with **pending** status.

5. Use **serve** subcommand to connect to DB and start task processing.
6. Initialize 5 goroutines (workers) to process tasks concurrently.
7. Start fetcher goroutine that runs every 5 seconds (**FETCH_INTERVAL**).
8. Fetch tasks using **FOR UPDATE SKIP LOCKED** to avoid duplicate picks.
9. Update fetched tasks to **in_progress** and send them to workers via channel.
10. Each worker:
  - Receives task from channel.
  - Sleeps 1–4 seconds to simulate processing.
  - Reverses task payload.
  - Randomly fails 1 in 10 tasks.
  - Updates DB with **status=done** and reversed result.
  - If failed, sets **status=error** and error message.
11. App listens for **Ctrl + C** to trigger graceful shutdown.
12. On shutdown: cancels context, waits for workers, exits cleanly.
13. Logs all major actions: DB connect, migrations, fetch, process, shutdown.

Installation

Prerequisites

- Go 1.16+ installed
- PostgreSQL 10+ server
- Git

Steps

1. Clone the repository:
```
git clone https://github.com/yourusername/task-processor.git
```

2. Install dependencies:
```
go mod download
```

3. Set up your environment variables:
```
cp .env.example .env
```

4. Run database migrations:
```
go run main.go migrate
```

Configuration

The application is configured using environment variables, which can be loaded from a .env file.

Environment Variable	Description	Default
DATABASE_URL	PostgreSQL connection string	postgres://postgres@localhost:5432/taskprocessor?sslmode=disable
WORKER_COUNT	Number of concurrent worker goroutines	5

FETCH_INTERVAL	Time interval between task fetches (seconds)	5
FETCH_LIMIT	Maximum number of tasks to fetch at once	10

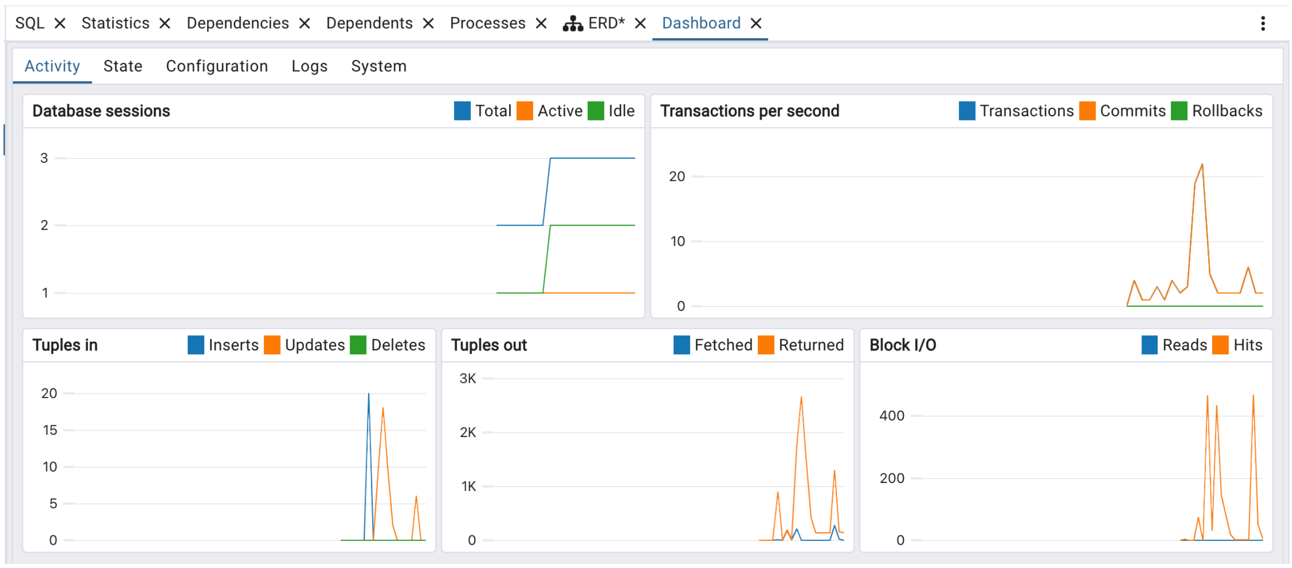
Database Schema

This project uses a PostgreSQL database with a single table to manage and track asynchronous job execution.

ERD (Entity Relationship Diagram)

Column	Type	Description
id	serial PRIMARY KEY	Task ID
payload	text	Task payload/input
status	text	pending' or 'done'
result	text	Result of processing
created_at	timestamp with time zone	Creation timestamp
updated_at	timestamp with time zone	Last updated timestamp

Monitoring with pgAdmin 4



Live pgAdmin dashboard view during task processing. Note the transaction activity, minimal rollbacks, and high cache hit ratio — all indicators of a healthy, responsive PostgreSQL-backed backend.

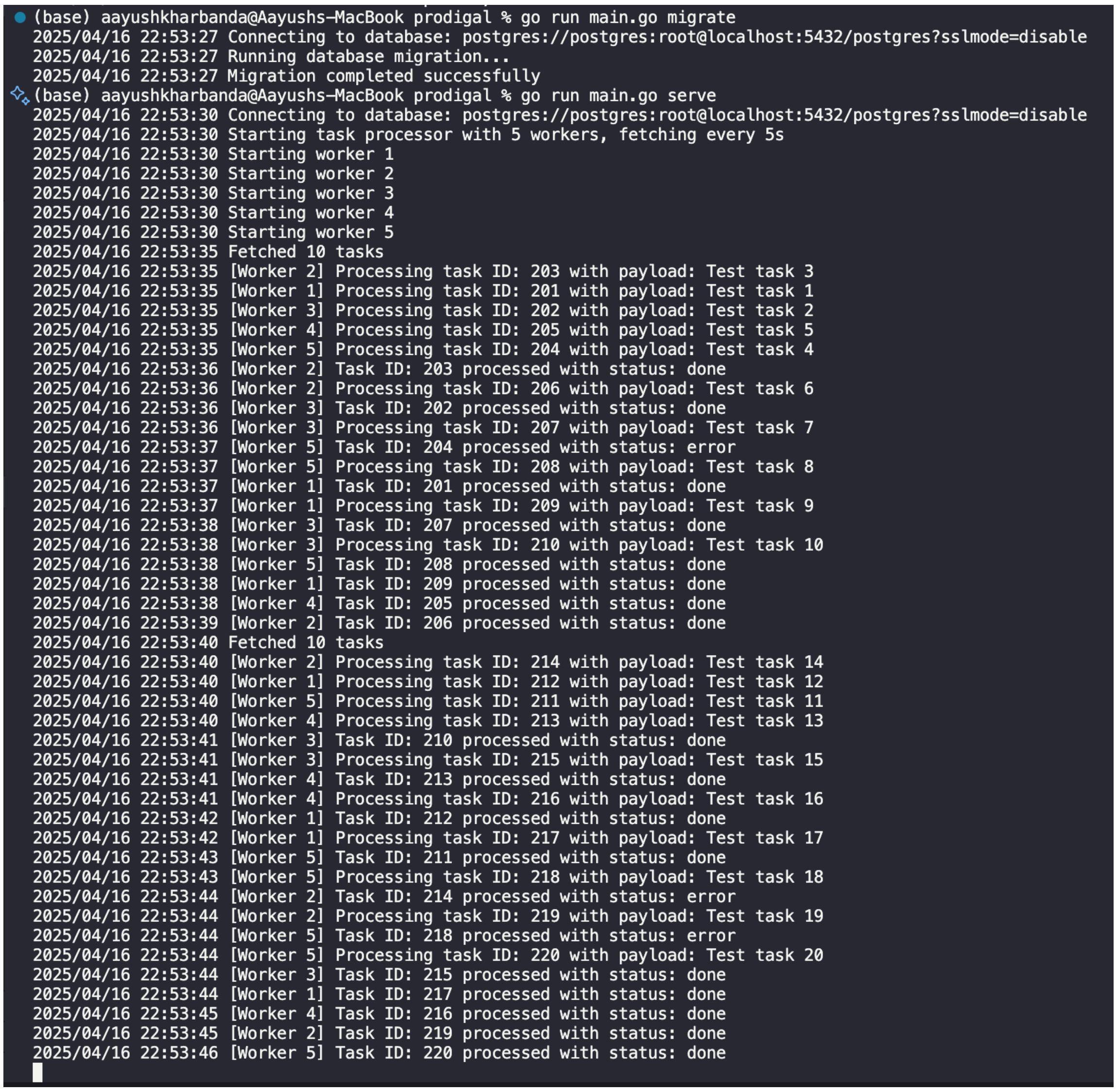
Running the Application

1. Run database migrations:
   `go run main.go migrate`

2. Create sample 20 tasks to test
   `psql "$DATABASE_URL" -f seed/seed_tasks.sql`

3. Start the task processor service:
   `go run main.go serve`

CLI Output Example



```
(base) aayushkharbanda@Aayushs-MacBook prodigal % go run main.go migrate
2025/04/16 22:53:27 Connecting to database: postgres://postgres:root@localhost:5432/postgres?sslmode=disable
2025/04/16 22:53:27 Running database migration...
2025/04/16 22:53:27 Migration completed successfully
(base) aayushkharbanda@Aayushs-MacBook prodigal % go run main.go serve
2025/04/16 22:53:30 Connecting to database: postgres://postgres:root@localhost:5432/postgres?sslmode=disable
2025/04/16 22:53:30 Starting task processor with 5 workers, fetching every 5s
2025/04/16 22:53:30 Starting worker 1
2025/04/16 22:53:30 Starting worker 2
2025/04/16 22:53:30 Starting worker 3
2025/04/16 22:53:30 Starting worker 4
2025/04/16 22:53:30 Starting worker 5
2025/04/16 22:53:35 Fetched 10 tasks
2025/04/16 22:53:35 [Worker 2] Processing task ID: 203 with payload: Test task 3
2025/04/16 22:53:35 [Worker 1] Processing task ID: 201 with payload: Test task 1
2025/04/16 22:53:35 [Worker 3] Processing task ID: 202 with payload: Test task 2
2025/04/16 22:53:35 [Worker 4] Processing task ID: 205 with payload: Test task 5
2025/04/16 22:53:35 [Worker 5] Processing task ID: 204 with payload: Test task 4
2025/04/16 22:53:36 [Worker 2] Task ID: 203 processed with status: done
2025/04/16 22:53:36 [Worker 2] Processing task ID: 206 with payload: Test task 6
2025/04/16 22:53:36 [Worker 3] Task ID: 202 processed with status: done
2025/04/16 22:53:36 [Worker 3] Processing task ID: 207 with payload: Test task 7
2025/04/16 22:53:37 [Worker 5] Task ID: 204 processed with status: error
2025/04/16 22:53:37 [Worker 5] Processing task ID: 208 with payload: Test task 8
2025/04/16 22:53:37 [Worker 1] Task ID: 201 processed with status: done
2025/04/16 22:53:37 [Worker 1] Processing task ID: 209 with payload: Test task 9
2025/04/16 22:53:38 [Worker 3] Task ID: 207 processed with status: done
2025/04/16 22:53:38 [Worker 3] Processing task ID: 210 with payload: Test task 10
2025/04/16 22:53:38 [Worker 5] Task ID: 208 processed with status: done
2025/04/16 22:53:38 [Worker 1] Task ID: 209 processed with status: done
2025/04/16 22:53:38 [Worker 4] Task ID: 205 processed with status: done
2025/04/16 22:53:39 [Worker 2] Task ID: 206 processed with status: done
2025/04/16 22:53:40 Fetched 10 tasks
2025/04/16 22:53:40 [Worker 2] Processing task ID: 214 with payload: Test task 14
2025/04/16 22:53:40 [Worker 1] Processing task ID: 212 with payload: Test task 12
2025/04/16 22:53:40 [Worker 5] Processing task ID: 211 with payload: Test task 11
2025/04/16 22:53:40 [Worker 4] Processing task ID: 213 with payload: Test task 13
2025/04/16 22:53:41 [Worker 3] Task ID: 210 processed with status: done
2025/04/16 22:53:41 [Worker 3] Processing task ID: 215 with payload: Test task 15
2025/04/16 22:53:41 [Worker 4] Task ID: 213 processed with status: done
2025/04/16 22:53:41 [Worker 4] Processing task ID: 216 with payload: Test task 16
2025/04/16 22:53:42 [Worker 1] Task ID: 212 processed with status: done
2025/04/16 22:53:42 [Worker 1] Processing task ID: 217 with payload: Test task 17
2025/04/16 22:53:43 [Worker 5] Task ID: 211 processed with status: done
2025/04/16 22:53:43 [Worker 5] Processing task ID: 218 with payload: Test task 18
2025/04/16 22:53:44 [Worker 2] Task ID: 214 processed with status: error
2025/04/16 22:53:44 [Worker 2] Processing task ID: 219 with payload: Test task 19
2025/04/16 22:53:44 [Worker 5] Task ID: 218 processed with status: error
2025/04/16 22:53:44 [Worker 5] Processing task ID: 220 with payload: Test task 20
2025/04/16 22:53:44 [Worker 3] Task ID: 215 processed with status: done
2025/04/16 22:53:44 [Worker 1] Task ID: 217 processed with status: done
2025/04/16 22:53:45 [Worker 4] Task ID: 216 processed with status: done
2025/04/16 22:53:45 [Worker 2] Task ID: 219 processed with status: done
2025/04/16 22:53:46 [Worker 5] Task ID: 220 processed with status: done
```

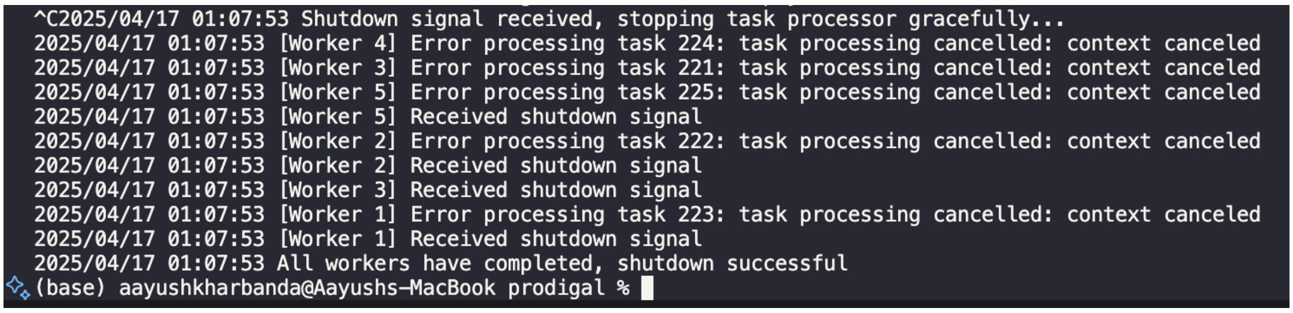
Above screenshot illustrates the task processor running with multiple workers. Each worker fetches tasks from the PostgreSQL database every 5 seconds, processes the payload, and updates the task's status (done or error). The log messages displayed in the CLI provide real-time visibility into task execution, status updates, and concurrency.

Graceful Shutdown Testing

I tested graceful shutdown by:

1. Starting the service with a longer processing time:
   ```
   go run main.go serve
   ```

2. Monitoring the logs while sending termination signals:
   ```
   # Ctrl + c
   ```



```
^C2025/04/17 01:07:53 Shutdown signal received, stopping task processor gracefully...
2025/04/17 01:07:53 [Worker 4] Error processing task 224: task processing cancelled: context canceled
2025/04/17 01:07:53 [Worker 3] Error processing task 221: task processing cancelled: context canceled
2025/04/17 01:07:53 [Worker 5] Error processing task 225: task processing cancelled: context canceled
2025/04/17 01:07:53 [Worker 5] Received shutdown signal
2025/04/17 01:07:53 [Worker 2] Error processing task 222: task processing cancelled: context canceled
2025/04/17 01:07:53 [Worker 2] Received shutdown signal
2025/04/17 01:07:53 [Worker 3] Received shutdown signal
2025/04/17 01:07:53 [Worker 1] Error processing task 223: task processing cancelled: context canceled
2025/04/17 01:07:53 [Worker 1] Received shutdown signal
2025/04/17 01:07:53 All workers have completed, shutdown successful
(base) aayushkharbanda@Aayushs-MacBook prodigal %
```

Graceful shutdown: signal received, tasks completed, workers exited, application closed cleanly.