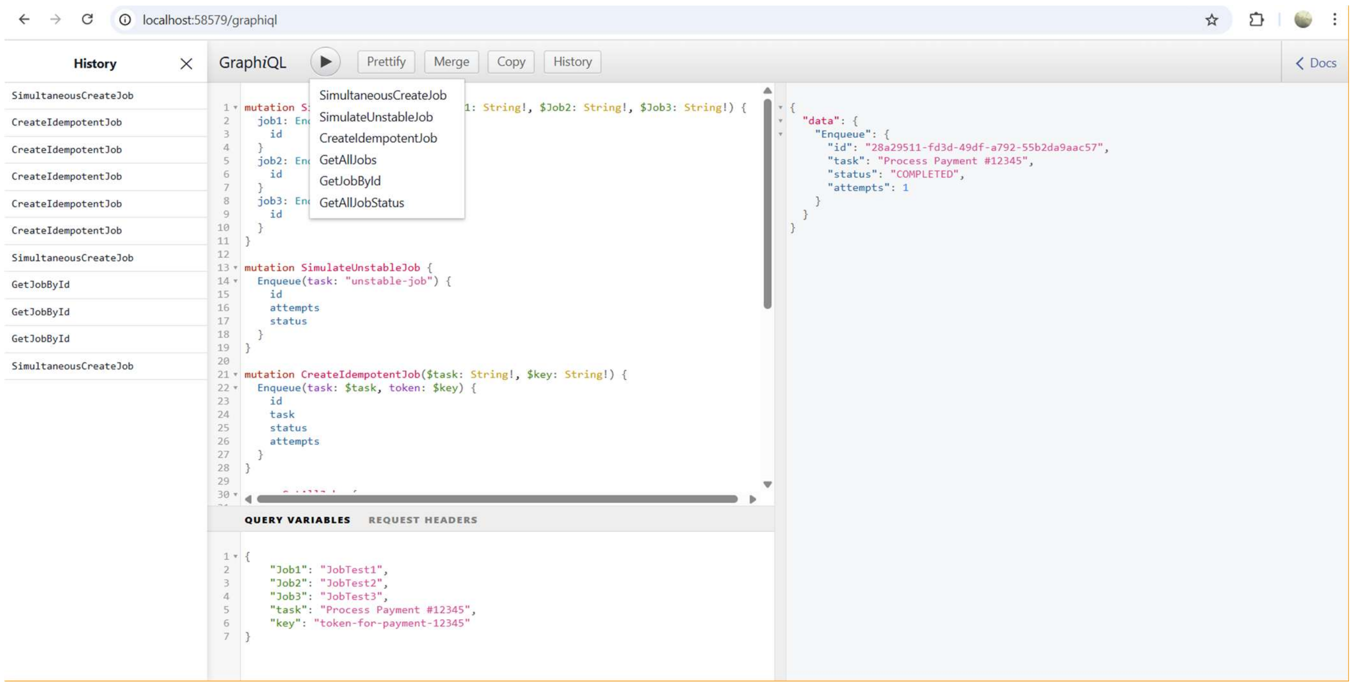
Pre-Assessment Test Pahamify Internship 2025

Documentation for Backend Path

By: Irfan Sidiq Permana



```
mutation S                   1: String!, $Job2: String!, $Job3: String!) {
  job1: En
    id
  }
  job2: En
    id
  }
  job3: En
    id
  }
}

mutation SimulateUnstableJob {
  Enqueue(task: "unstable-job") {
    id
    attempts
    status
  }
}

mutation CreateIdempotentJob($task: String!, $key: String!) {
  Enqueue(task: $task, token: $key) {
    id
    task
    status
    attempts
  }
}
```

```
{
    "Job1": "JobTest1",
    "Job2": "JobTest2",
    "Job3": "JobTest3",
    "task": "Process Payment #12345",
    "key": "token-for-payment-12345"
}
```

```
{
  "data": {
    "Enqueue": {
      "id": "28a29511-fd3d-49df-a792-55b2da9aac57",
      "task": "Process Payment #12345",
      "status": "COMPLETED",
      "attempts": 1
    }
  }
}
```

Last Program Commit:
27 June 2025, 06:58:24 AM

Problem Author: @arkanmis

Problem Link: https://github.com/arkanmis/traffic-light-simulator

Problem Description

In Pre-Assessment Backend Path for Pahamify Internship 2025, we are tasked to develop a Simple Distributed Job Queue Simulation using Go. The core objective is to build a robust, in-memory system capable of managing and processing asynchronous jobs, simulating a real-world distributed task processor.

Key technical requirements include:

- **Concurrency and Performance**: The system must handle a load of 50-100 concurrent jobs without crashing or significant slowdown.
- **Idempotency**: Job creation must be idempotent, preventing duplicate processing when a client sends the same request multiple times with a unique key.
- **Reliability and Retries**: The system must demonstrate resilience by implementing a retry mechanism for tasks that fail, specifically for a pre-defined "unstable job" that is expected to fail twice before succeeding.
- **Graceful Shutdown**: The application must handle termination signals gracefully, ensuring all in-progress jobs are completed before the system exits to prevent data loss.

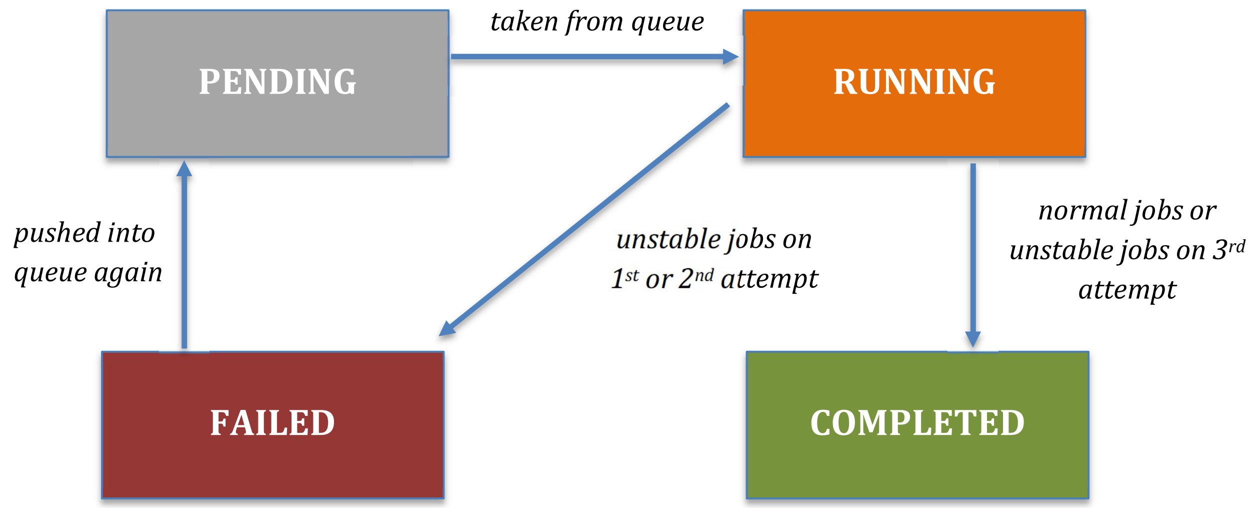
Solution Design

Job States & Processing Logic

First, we define the possible states for our tasks and their implications:

- **Pending**: Jobs that are in the job queue, waiting to be taken and processed → eventually will be taken and marked as running
- **Running**: Jobs that have been popped from job queue and are currently being processed → after processing will either be marked as completed or failed
- **Completed**: Jobs that have been processed completely
- **Failed**: Unstable jobs fail on their 1st and 2nd attempt → mark them as pending and add them to the job queue again

(state illustration below)


PENDING
taken from queue
RUNNING
pushed into queue again
unstable jobs on 1st or 2nd attempt
normal jobs or unstable jobs on 3rd attempt
FAILED
COMPLETED

Next, we define how to choose a job and process it. In the specification we're required to handle a load of many concurrent requests at once, therefore we will use the **Worker Pool pattern**:

- We define the **capacity** of our worker pool (in `constant/constant.go`), e.g. twice the size of the workers to prevent system overload
- **Create** the pool and the workers by creating its channel and goroutines (in `service/job.go` when we create a new JobService),
- **Push** new jobs into the job queue whenever client sends a valid request (in `service/job.go` when we enqueue a job) → we first check if the job has the same token as earlier jobs for idempotency
- Workers will **take** the jobs from queue one by one and process it before taking another job from the queue (in `service/job.go` when we create a new worker, they'll listen to the channel for new jobs)

For implementing the job queue, we will use two mechanisms:

- **JobRepository** for storing all the jobs → used as the "single source of truth", whenever we receive or update a job (changing its number of attempts, its status, etc.) we'll always immediately save it into jobRepository for consistency
- **JobChannel** for sending messages to the worker → worker will always retrieve jobs from repository when it receives message from the channel for ensuring consistency

Pseudocode for those steps:

```
function NewJobService():
    Create a channel jobChannel with pool size twice the number of workers
```

```
  Call startWorkers(numWorkers)

function startWorkers(numWorkers: int):
  for i ← 1 to numWorkers do
    Increment waitGroup mutex by 1 (for cleanup later)
    Create worker goroutine with id = i that listens to jobChannel and pulls from it

function enqueue(taskName: string, token: string) -> Job:
  if token is the same as the token from earlier jobs then
    return

  Create a new job with given taskName and token
  Save it into jobRepository
  Push the job into jobChannel
```

Each worker will pull the job from the channel and process it in the following manner:

- If job is a **stable job** or an **unstable job with 3 attempts**, just mark it as completed
- If job is an **unstable job with less than 3 attempts**, print failed message in log and push it into the job queue again

Pseudocode for the above step:

```
function processJob(jobMessage: Job):
  job ← find the actual job in jobRepository by jobMessage.ID
  if job not found in jobRepository then
    return

  job.Attempts ← job.Attempts + 1
  job.Status ← "RUNNING"
  Save this updated job in jobRepository for consistency

  Simulate processing job by waiting for a few seconds
  jobSuccessful ← true

  if job.Task = "unstable-job" and job.Attempts < 3 then
    jobSuccessful ← false

  if jobSuccessful then
    job.Status ← "COMPLETED"
    Save this updated job in jobRepository
  else
    Print log message with status "FAILED"
```

```
        job.Status ← "PENDING"
        Save this updated job in jobRepository
        Waits for 2 seconds, then push this job to the jobChannel again
```

We have now completed the main logic for our program.

Idempotency

To implement the idempotency feature, we need a way to tell that two jobs are the same when enqueuing them into the job queue, so that if the enqueued job is the same as an earlier job then we won't process it and just return the earlier job to the client.

For this we will add a new property into our jobs: **Token** (as briefly mentioned before), which differentiates a job from one another. Keep in mind that this token is different from ID, which in our implementation is automatically generated by UUID so there's an astronomically small chance for two jobs to have the same ID. We will use this token property for the client to tell the server that two jobs are actually the same.

First, we modify the Enqueue method to include token as an optional parameter:

delivery/graphql/schema/mutation.graphql

```
type Mutation {
  Enqueue(task: String!, token: String): Job
}
```

We also change the Job definition to include token as an optional attribute:

entity/job.go

```
type Job struct {
  ID       string  `json:"id"`
  Token    *string `json:"token,omitempty"`
  Task     string  `json:"task"`
  Status   string  `json:"status"`
  Attempts int32   `json:"attempts"`
}
```

Then, we add a hashmap in JobRepository for keeping track of existing tokens:

repository/inmem/job.go

```
type jobRepository struct {
      mu          sync.RWMutex
      inMemDb     map[string]*entity.Job
      tokenIndex       map[string]*entity.Job
}
```

Finally, we prepare the functions to use this new token hashmap:

- Modify jobRepository.Save() to also save new job tokens into the hashmap
- Add a new function jobRepository.findByToken() to be used in processJob() before actually processing a job

Graceful Shutdown

Lastly, we update our program to handle shutdown gracefully, that is closing all channels and cleaning all workers on program termination/crash.

To do this, we just add a new function in jobService for shutdown:

```
function Shutdown():
    Close the jobChannel
    Wait for all workers to be terminated using WaitGroup
```

And call this in our main function upon program termination or crash:

```
function main():
    …(program setup)…

    Start server asynchronously with goroutine to make it non-blocking
    Make a new channel quit that listens for OS interrupts for termination

    Listen to the quit channel until an interrupt message comes
    Call jobService.Shutdown()
    Close the HTTP server
```

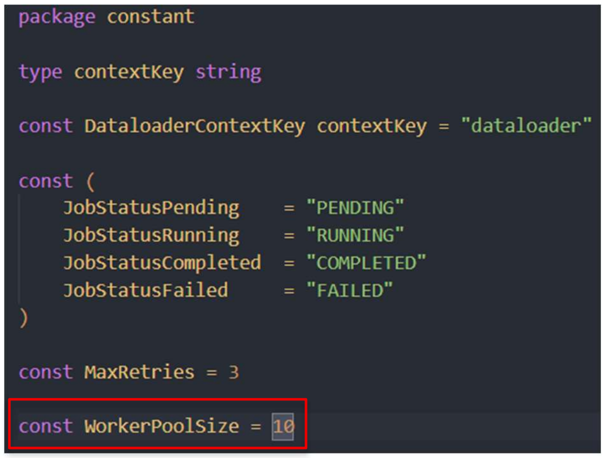
Testing

Below are tests that you can do for checking the evaluation criteria:

a. Stress Test (100 concurrent request)

We can test this using a client program with goroutines and for loops to send 100 concurrent request to the server.

Before running the main program, you can adjust the number of workers you want to use in `pkg/constant/constant.go`:

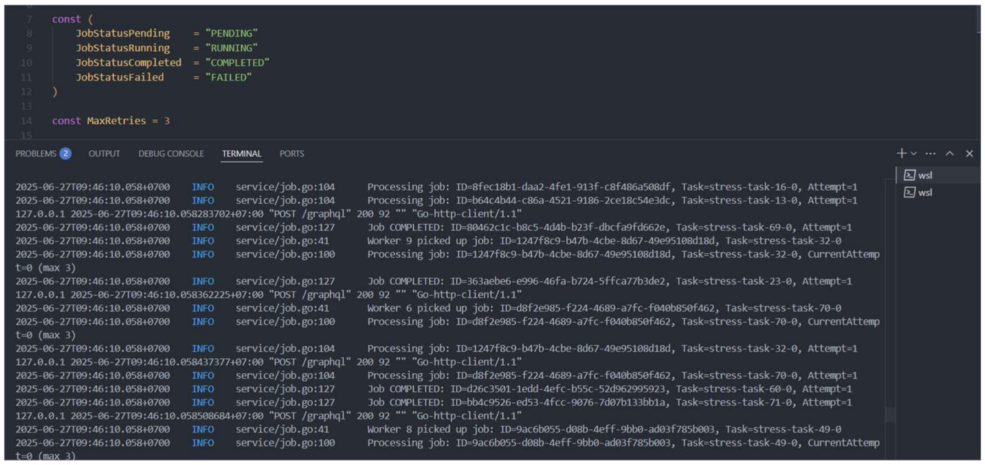


```go
package constant

type contextKey string

const DataloaderContextKey contextKey = "dataloader"

const (
    JobStatusPending    = "PENDING"
    JobStatusRunning    = "RUNNING"
    JobStatusCompleted  = "COMPLETED"
    JobStatusFailed     = "FAILED"
)

const MaxRetries = 3

const WorkerPoolSize = 10
```

A testing client has been prepared in `stresstest/main.go`.
Steps for doing the test:

- In the root project directory, run `go run main.go` in the terminal
- In another terminal (still in the root project directory), run `go run stresstest/main.go`
- Check for the logs in the main program terminal
- Periodically check the jobs' status by running `GetAllJobStatus()` in the client

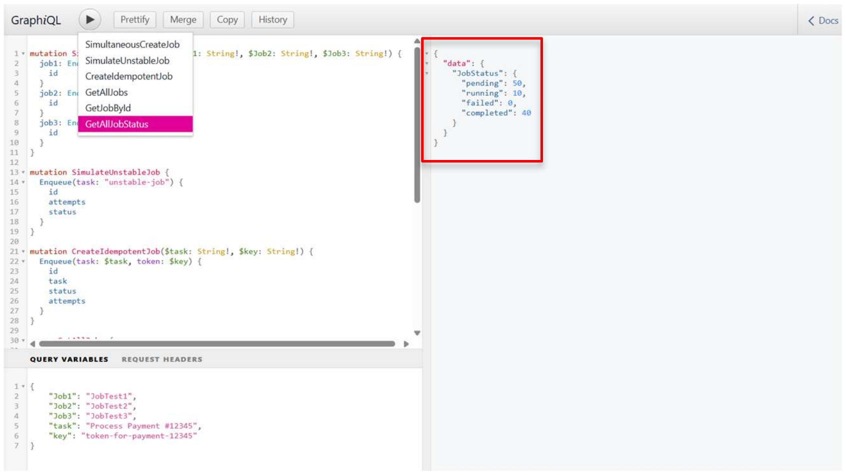
The test program will send 100 simultaneous request to the server, which the server will respond by processing the request one by one using the workers available.



```go
const (
    JobStatusPending    = "PENDING"
    JobStatusRunning    = "RUNNING"
    JobStatusCompleted  = "COMPLETED"
    JobStatusFailed     = "FAILED"
)

const MaxRetries = 3
```

```
2025-06-27T09:46:10.058+0700    INFO    service/job.go:104      Processing job: ID=8fec18b1-daa2-4fe1-913f-c8f486a508df, Task=stress-task-16-0, Attempt=1
2025-06-27T09:46:10.058+0700    INFO    service/job.go:104      Processing job: ID=b64c4b44-c86a-4521-9186-2ce18c54e3dc, Task=stress-task-13-0, Attempt=1
127.0.0.1 2025-06-27T09:46:10.058283702+07:00 "POST /graphql" 200 92 "" "Go-http-client/1.1"
2025-06-27T09:46:10.058+0700    INFO    service/job.go:127      Job COMPLETED: ID=80462c1c-b8c5-4d4b-b23f-dbcfa9fd662e, Task=stress-task-69-0, Attempt=1
2025-06-27T09:46:10.058+0700    INFO    service/job.go:41       Worker 9 picked up job: ID=1247f8c9-b47b-4cbe-8d67-49e95108d18d, Task=stress-task-32-0
2025-06-27T09:46:10.058+0700    INFO    service/job.go:100      Processing job: ID=1247f8c9-b47b-4cbe-8d67-49e95108d18d, Task=stress-task-32-0, CurrentAttemp
t=0 (max 3)
2025-06-27T09:46:10.058+0700    INFO    service/job.go:127      Job COMPLETED: ID=363aebe6-e996-46fa-b724-5ffca77b3de2, Task=stress-task-23-0, Attempt=1
127.0.0.1 2025-06-27T09:46:10.058362225+07:00 "POST /graphql" 200 92 "" "Go-http-client/1.1"
2025-06-27T09:46:10.058+0700    INFO    service/job.go:41       Worker 6 picked up job: ID=d8f2e985-f224-4689-a7fc-f040b850f462, Task=stress-task-70-0
2025-06-27T09:46:10.058+0700    INFO    service/job.go:100      Processing job: ID=d8f2e985-f224-4689-a7fc-f040b850f462, Task=stress-task-70-0, CurrentAttemp
t=0 (max 3)
2025-06-27T09:46:10.058+0700    INFO    service/job.go:104      Processing job: ID=1247f8c9-b47b-4cbe-8d67-49e95108d18d, Task=stress-task-32-0, Attempt=1
127.0.0.1 2025-06-27T09:46:10.058437377+07:00 "POST /graphql" 200 92 "" "Go-http-client/1.1"
2025-06-27T09:46:10.058+0700    INFO    service/job.go:104      Processing job: ID=d8f2e985-f224-4689-a7fc-f040b850f462, Task=stress-task-70-0, Attempt=1
2025-06-27T09:46:10.058+0700    INFO    service/job.go:127      Job COMPLETED: ID=d26c3501-1edd-4efc-b55c-52d962995923, Task=stress-task-60-0, Attempt=1
2025-06-27T09:46:10.058+0700    INFO    service/job.go:127      Job COMPLETED: ID=bb4c9526-ed53-4fcc-9076-7d07b133bb1a, Task=stress-task-71-0, Attempt=1
127.0.0.1 2025-06-27T09:46:10.058508684+07:00 "POST /graphql" 200 92 "" "Go-http-client/1.1"
2025-06-27T09:46:10.058+0700    INFO    service/job.go:41       Worker 8 picked up job: ID=9ac6b055-d08b-4eff-9bb0-ad03f785b003, Task=stress-task-49-0
2025-06-27T09:46:10.058+0700    INFO    service/job.go:100      Processing job: ID=9ac6b055-d08b-4eff-9bb0-ad03f785b003, Task=stress-task-49-0, CurrentAttemp
t=0 (max 3)
```

You can check the progress periodically by running `GetAllJobStatus()`:



```
mutation S                              1: String!, $Job2: String!, $Job3: String!) {
  job1: En
    id
  }
  job2: En
    id
  }
  job3: En
    id
  }
}

mutation SimulateUnstableJob {
  Enqueue(task: "unstable-job") {
    id
    attempts
    status
  }
}

mutation CreateIdempotentJob($task: String!, $key: String!) {
  Enqueue(task: $task, token: $key) {
    id
    task
    status
    attempts
  }
}
```

```
{
    "Job1": "JobTest1",
    "Job2": "JobTest2",
    "Job3": "JobTest3",
    "task": "Process Payment #12345",
    "key": "token-for-payment-12345"
}
```

```
{
  "data": {
    "JobStatus": {
      "pending": 50,
      "running": 10,
      "failed": 0,
      "completed": 40
    }
  }
}
```

b. Idempotency

To test this, we just need to run `CreateIdempotentJob` multiple times successively without changing the "key" variable in the query variables. Upon testing, you should see that:

- `CreateIdempotentJob` always returns the same Job object (can be checked by its ID).
- `GetAllJobs` or `GetAllJobStatus` shows that the number of job only increases by one no matter how many times we run `CreateIdempotentJob`.

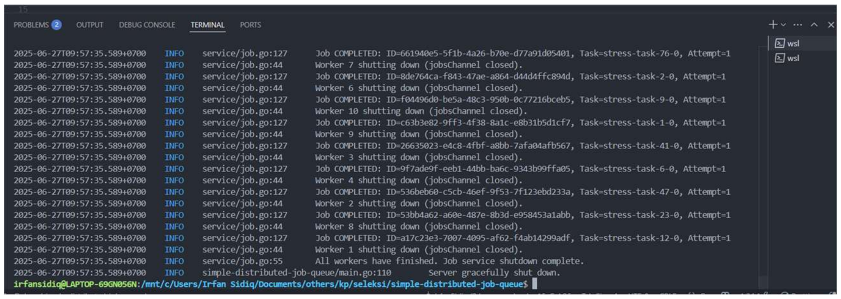
c. Graceful Shutdown

To test this, we just need to enqueue multiple jobs concurrently (can be done by calling `SimultaneousCreateJob`, running the stress test program, etc.) and then **immediately terminate the server** (using Ctrl+C for example).

Server won't immediately terminate, but instead give itself just a few seconds to complete the currently processed job and do cleanup.

Upon testing and checking the logs, you should see that:

- The server closes the jobChannel and waits for all workers to finish first
- All workers eventually shuts down after completing its current job
- Server then completes its shutdown process



```
2025-06-27T09:57:35.589+0700    INFO    service/job.go:127      Job COMPLETED: ID=661940e5-5f1b-4a26-b70e-d77a91d05401, Task=stress-task-76-0, Attempt=1
2025-06-27T09:57:35.589+0700    INFO    service/job.go:44       Worker 7 shutting down (jobsChannel closed).
2025-06-27T09:57:35.589+0700    INFO    service/job.go:127      Job COMPLETED: ID=8de764ca-f843-47ae-a864-d44d4ffc894d, Task=stress-task-2-0, Attempt=1
2025-06-27T09:57:35.589+0700    INFO    service/job.go:44       Worker 6 shutting down (jobsChannel closed).
2025-06-27T09:57:35.589+0700    INFO    service/job.go:127      Job COMPLETED: ID=f04496d0-be5a-48c3-950b-0c77216bceb5, Task=stress-task-9-0, Attempt=1
2025-06-27T09:57:35.589+0700    INFO    service/job.go:44       Worker 10 shutting down (jobsChannel closed).
2025-06-27T09:57:35.589+0700    INFO    service/job.go:127      Job COMPLETED: ID=c63b3e82-9ff3-4f38-8a1c-e8b31b5d1cf7, Task=stress-task-1-0, Attempt=1
2025-06-27T09:57:35.589+0700    INFO    service/job.go:44       Worker 9 shutting down (jobsChannel closed).
2025-06-27T09:57:35.589+0700    INFO    service/job.go:127      Job COMPLETED: ID=26635023-e4c8-4fbf-a8bb-7afa04afb567, Task=stress-task-41-0, Attempt=1
2025-06-27T09:57:35.589+0700    INFO    service/job.go:44       Worker 3 shutting down (jobsChannel closed).
2025-06-27T09:57:35.589+0700    INFO    service/job.go:127      Job COMPLETED: ID=9f7ade9f-eeb1-44bb-ba6c-9343b99ffa05, Task=stress-task-6-0, Attempt=1
2025-06-27T09:57:35.589+0700    INFO    service/job.go:44       Worker 4 shutting down (jobsChannel closed).
2025-06-27T09:57:35.589+0700    INFO    service/job.go:127      Job COMPLETED: ID=536beb60-c5cb-46ef-9f53-7f123ebd233a, Task=stress-task-47-0, Attempt=1
2025-06-27T09:57:35.589+0700    INFO    service/job.go:44       Worker 2 shutting down (jobsChannel closed).
2025-06-27T09:57:35.589+0700    INFO    service/job.go:127      Job COMPLETED: ID=53bb4a62-a60e-487e-8b3d-e958453a1abb, Task=stress-task-23-0, Attempt=1
2025-06-27T09:57:35.589+0700    INFO    service/job.go:44       Worker 8 shutting down (jobsChannel closed).
2025-06-27T09:57:35.589+0700    INFO    service/job.go:127      Job COMPLETED: ID=a17c23e3-7007-4095-af62-f4ab14299adf, Task=stress-task-12-0, Attempt=1
2025-06-27T09:57:35.589+0700    INFO    service/job.go:44       Worker 1 shutting down (jobsChannel closed).
2025-06-27T09:57:35.589+0700    INFO    service/job.go:55       All workers have finished. Job service shutdown complete.
2025-06-27T09:57:35.589+0700    INFO    simple-distributed-job-queue/main.go:110        Server gracefully shut down.
irfansidiq@LAPTOP-69GN056N:/mnt/c/Users/Irfan Sidiq/Documents/others/kp/seleksi/simple-distributed-job-queue$
```